EDAgraffiti: The Book

Paul McLellan

EDAgraffiti: the book

To purchase: www.lulu.com

Introduction

This book is an outgrowth from the blog EDAgraffiti on EDN magazine online. Although the basis of the book is the original blog entries, there is new material and the old material has been updated and reorganized.

When moving the material from a blog into a book I had to decide what to do about what were links to other parts of the web. I decided that rather than putting in long URLs in footnotes, which are unpleasant to type, I'd make sure that there was enough information to find everything with a search engine. So, for example, rather than giving an explicit URL for a book on Amazon, if you have the title it should be straightforward to find.

I'd like to thank Ed Lee and Jim Hogan for encouraging me to do the blog in the first place, and the editors of EDN magazine for hosting it. Jim also created the map of semiconductor company relationships that appears in this book. My son Sam designed the EDAgraffiti logo and wall.

The blog is here.

Paul McLellan

San Francisco, March 2010

Email: paul@greenfolder.com

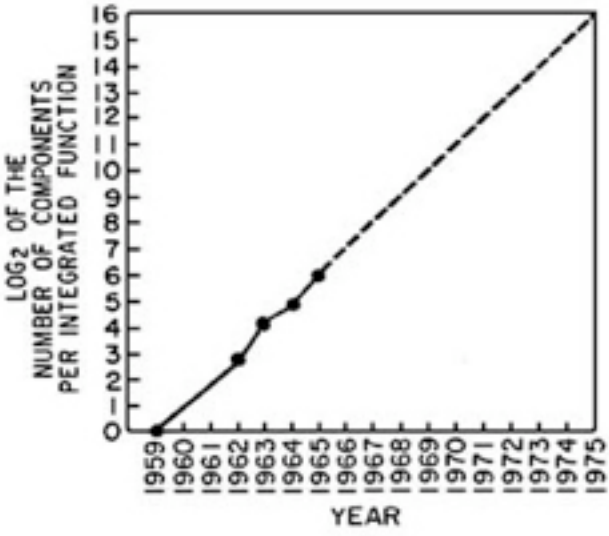
Chapter 1: Major industry trends

What did Moore really say?

Every EDA marketing presentation starts off by pointing out that Moore's law is making some problem worse. Of course, just the problem that the EDA product pitched in the rest of the presentation is designed to solve.

Everyone sorta knows Moore's law but few people realize just what it was he said over 40 years ago, and just how prescient he was, or have even read his original paper.

In 1965, Gordon Moore was Head of R&D at Fairchild. This was several years before Moore left Fairchild to found Intel. Moore noticed that the number of transistors on the integrated circuits that Fairchild was building seemed to double every two years, as shown in the graph here from Moore's original article. As he pointed out there, "Integrated circuits will lead to such wonders as home computers, automatic controls for automobiles, and personal portable communications equipment." Remember that this was 1965, when an integrated circuit contained 64 transistors: this was an extraordinary prediction.



Surprisingly, over 40 years on, semiconductors seem still to be increasing in complexity at this rate. Gordon Moore's original remark is, of course, now known as "Moore's Law" and is expected to continue for some time.

Exponential growth like this over a sustained period of time, rather like compound interest, has a dramatic effect. In the seventies a chip may have contained a few hundred transistors. Today a chip can contain billions of transistors and in the future the predictions are for chips with several hundred billion transistors. This is how all the electronics for a high-end mainframe computer can be compressed into a single chip. Only we attach a radio to it and call it a cell-phone. Or we put a lens on it and call it a digital camera. Or we attach a dish to it and call it satellite TV. Or we take it on a plane and use it to write about Moore's Law.

But it is possible to look at Moore's Law the other way round: the cost of any given functionality implemented in electronics halves every two years or so. Over a period of twenty years this is a thousand-fold reduction. A video-game console, which is so cheap that children can buy them from their allowances, has far more computing power and much better graphics than the highest-end flight simulators of the 1970s, which cost millions of dollars. An ink-jet printer has far more computing power than NASA had at its disposal for the moon-shots (supposedly a total of 1 MIPS[1] on all the computers they had put together). It is this exponential driving down of electronic costs that had transformed so many aspects of our lives in the last twenty years or so since integrated circuits became cheap enough to go into consumer products.

Here is Gordon Moore again, this time from a 1995 Fortune article: "The whole point of integrated circuits is to absorb the functions of what previously were discrete electronic components, to incorporate them in a single new chip, and then to give them back for free, or at least for a lot less money than what they cost as individual parts. Thus, semiconductor technology eats everything, and people who oppose it get trampled."

Moore's Law started as an observation, became a prediction, and eventually transformed into a blueprint for the semiconductor

industry. The International Technology Roadmap for Semiconductors is largely an analysis of what it will take to make Moore's Law continue to be true. Moore's Law has thus become a self-fulfilling prophecy for the time being.

Of course the really interesting question is for how much longer?

[1] Have you noticed that people like to write or say 1 MIP as if MIPS was plural. But, of course, the S stands for "seconds". End of today's nitpicking.

Why EDA differs from ERP by more than one letter

What is it about EDA that makes it different from other software businesses? When the CFO of Texas Instruments buys Oracle or SAP, he or she doesn't study what algorithms they use in their relational database. EDA purchasers are the only people who "take the cylinder head off and look at the valves" before buying.

I think it is the speed of change. EDA, as we are all tired of hearing, is driven by Moore's law. But the effect is that every few years a complete technology re-investment needs to be made and the incumbent does not have much of an advantage in discontinuous change. It's not like that in other software industries. For Oracle, the last major change in databases was the relational database superseding hierarchical databases, and that was starting in the 1970s (first at IBM and then when Larry Ellison founded Oracle under its original name, Software Development Laboratories). It looks like there may be another change starting, driven by the internet, to schema-free databases which scale better to thousands of servers. So one turn of the handle of Moore's law in 30 years, 15 to 20 times slower.

You could give me half a billion in VC money and I'm not going to be able to put together a startup to displace Oracle, no matter how many of the best database programmers I hire. Because it is not mainly about technology. And even if, by some miracle, I succeeded it would take 20 years. By contrast, when EDA companies miss a transition they tend to vanish quickly. Calma

was not a major force in gate-level design. Daisy missed synthesis despite Synopsys being staffed with many of the same people, and, to add insult to injury, in the same buildings. Cadence lost its Dracula physical verification franchise to Mentor's Calibre as quickly as it took 0.35um to come online, a few years. Yes, mistakes were made. Daisy made an ill executed acquisition of Cadentix. Cadence decided to make its own hierarchical DRC, Vampire, incompatible with Dracula so it didn't harm its cash-cow. But mistakes will always be made ("new" Coke, Ford Edsel..) but in other industries one mistake is rarely fatal.

The speed of change also means that startups can get traction in a way that they don't in other industries. The barriers to entry are really low, just a few people who really know the technology and it is possible to build a world-class product that is better than anything else out there. Most importantly, customers will buy it since the risk of using a product from a startup was lower than the risk of not doing so. Other businesses don't move so fast. Waiting for the big guys to have it is usually the safest approach.

This may be changing as the importance of integration increases and so the important of point technology diminishes. I spent years at Compass Design Automation with a fully-integrated toolset that was very productive. But customers would only buy "best-in-class point tools" and do all the integration themselves. We were selling an engine when people wanted to buy their own ignition system, their own fuel-injectors and make the wiring harness themselves. Despite the limited commercial success (Compass only reached $55M before it was acquired) I still believe that the integrated approach really was superior for everything except unusual chips like memories and microprocessors. The evidence was that every chip VLSI Technology produced until the late 1990s was produced entirely on Compass tools on very short time-frames. In that era, for example, a huge percentage of mobile phone chips, then and now a market with short product cycles, were done that way.

It is no longer clear that most semiconductor companies have the inclination or manpower to look at tools from startups. Their focus is more on reducing cost and reducing the number of

vendors they use. It is completely unclear how EDA will evolve in the current downturn. In Mike Santorini's memorable image, semiconductor is the speedboat that pulls the EDA water-skier. When it is at full speed we ski well. Right now I think the water is around our waist and we are still sinking.

Recutting the semiconductor pie

The large semiconductor companies have historically owned their own fabs for at least a large portion of their manufacturing capacity. They then needed to have enough product lines to fill the fabs, which in turn meant a certain scale. They were called IDMs, integrated device manufacturers, to distinguish them from fabless semiconductor companies who did design, marketing, sales but left the manufacturing to TSMC and UMC in Taiwan. Even if they were wanted to own their own fabs for some reason, they didn't have enough products to keep them full. However, as fabs have got more expensive and the technology development necessary to have a state-of-the-art process has risen, the size necessary to justify owning a fab has increased beyond the volume of almost any semiconductor company outside of memory and Intel. So companies like AMD and Texas Instruments are going completely fabless.

Once semiconductor companies are fabless, the motivation for having the existing set of divisions in the same company doesn't necessarily make that much sense. NXP (née Philips Semiconductors) has sold its wireless division to ST. That transaction was probably driven by the need to raise some cash by NXP's private equity owners, but I think it is typical of the sort of transaction we are likely to see. Freescale (née Motorola semiconductor division) has its wireless division up for sale too.

It is interesting (well, to some people like me) as to why companies exist at all. Why don't we all just be independent contractors and take orders for our services. The first person to think much about this was Ronald Coase who realized that the transaction costs involved in us all operating that way would swamp us, and so it is much more efficient to organize into companies with a more directive style. His work is known as

(surprise) the Coase Theorem and dates from 1937. For a semiconductor company with a fab, it was more efficient to organize into a company built around manufacturing and have enough design and selling capacity to feed that beast.

I expect that many more changes in how the semiconductor industry is carved up into different companies will come about in the next year or two. Not just mergers, although there will probably be some of those, but divisions being sold so that when the dust settles there will be just a handful of serious competitors in any market space. For example, it looks like wireless is going to come down to TI, ST, Qualcomm and Samsung.

Tragedy of the commons and EDA

The tragedy of the commons is an article published in Science magazine in 1968 by Garrett Hardin. It has since become very well known and is applicable widely when resources are shared without a market. The canonical example is common land being over-grazed or a common ocean being over-fished. It is in every fisherman's interest to fish as much as he can even though he knows that the area is being over-fished. If fishes a bit more than his quota (assuming there is one) he gets to keep all the value of the extra fish but the cost of the over-fishing is spread among all the other fisherman.

So what does this have to do with EDA?

Each semiconductor company knows that they need EDA investment in R&D to be healthy. However, when they negotiate with the EDA vendors, of course they want to get the lowest price possible. They get all the money they save, but the impact of the reduced revenue is spread among them and all their competitors, perhaps a 5% problem for them.

But just like over-fishing the oceans, each vendor pursuing this strategy means that EDA risks being starved for investment. Each semiconductor company's dream is that they get their EDA software for almost nothing, but that all the other semiconductor companies over-pay so EDA has plenty to invest. But that is not

the situation we are in today, even before the recent economic chaos.

A friend was arguing with me the other day that EDA is dead, using the Monty Python dead parrot sketch ("This parrot is no more! He has ceased to be!") to emphasize the point. If EDA was, say, the newspaper industry then this would be unarguable. The only debate is when, and how, and what comes next. But EDA is not a buggy-whip business, it cannot go away. It can only change its form. As a venture investment, I agree it is dead. As a business, probably not. And as a technology, certainly not.

Since almost all electronics depends on semiconductors and since semiconductors can only be designed with EDA tools, this is a potential problem brewing. Electronics is about 3 trillion dollars, semiconductor is about 400 billion dollars, and EDA is just 5 billion. It is tempting to think of EDA as a hair attempting to wag the tail and have the tail wag the dog. But EDA is more like the pituitary gland, a tiny bit of the animal but without which nothing else works.

Lithography for dummies

You probably already know that designs are transferred onto chips using a photographic process. The wafer is coated in a solution called photoresist and then exposed to light passed through a "mask" which alters its chemical composition. The exposed (or sometimes unexposed, depending on type) photoresist is then removed with a powerful acid and some semiconductor process takes place through the gaps created: diffusion of impurities, implantation of ions, etching of metal and so forth.

Originally a mask was the size of an entire wafer and all the die (technically the plural of die is dice but it looks so Las Vegas I'll stick with die) were exposed to light at the same time through a mask the same size as the wafer (which was 1", 2", 3" or 4" back then; now they are 12" with 18" in planning). For about the last twenty years, though, each die has been exposed individually through a reticle, a smaller mask that is stepped across the chip one die at a time by an expensive piece of equipment called a

stepper. The reticle is a multiple of the actual die size and the stepper has reduction optics rather like a photographic enlarger in reverse.

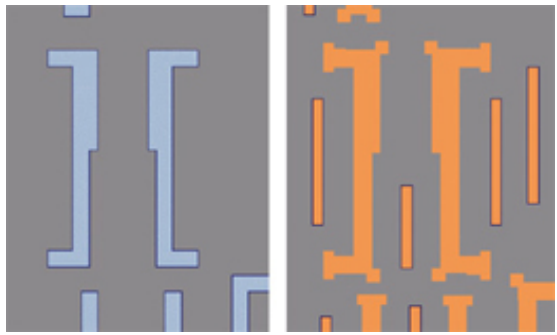
Originally we used 436nm mercury lamps which was a much shorter wavelength than the 1um or so feature size we were trying to achieve on the die so we didn't have to worry about all those strange things in optics that you may remember from high-school or college physics: Young's slits, diffraction gratings, wave interference. What was drawn on the layout designer's screen, what was put on the mask and what ended up on the silicon were pretty much the same thing.

As feature sizes got smaller, we reduced the wavelength of light, first to 248nm and then to 193nm. We are still at 193nm for two reasons. We had developed technology for DUV (deep-ultra-violet) at 157nm but it was really expensive and unattractive. We also discovered immersion lithography where the gap between the lens and the wafer is filled with water not air, which improves things enough that we can continue to use 193nm for the time being.

The basic problem is that as the wavelength gets shorter and shorter, we are moving out of the part of the electromagnetic spectrum where we can focus light with lenses, and into the part where we essentially have X-rays that go straight through the lens and through pretty much anything else too. The next step looks like it will have to be e-beam lithography, where a beam of electrons is steered in the same way as in an old TV. This is well-understood technically but it has a very slow write speed which, so far, makes the whole process uneconomical for mass production.

But being stuck at 193nm means we have a new problem. We have feature sizes on chips that are much less than 193nm (which is around 0.18um which was many process nodes ago). All sorts of optical effects happen due to wave interference of light and we needed to put very different patterns on the mask from the original layout, in order to get the eventual feature on the die to match what we first thought of. It became anything but WYSIWYG.

There is a whole gamut of techniques that have come to be known as RET, for resolution enhancement technologies. Optical proximity correction (OPC) changes the shape of what is on the mask so that what ends up on the wafer is what is required. For example, corners have extra lumps added so that they don't get etched away. Phase shift masking (PSM) etches the reticle by fractions of a wavelength so that the interference that results is desirable. The generic name for putting these extra features onto the mask is known as RET decoration. Since this might multiply the billion or so shapes on a layer by a factor of ten it is computationally very expensive.



A whole subsegment of EDA grew up when this first became important, under the generic name of DFM, design for manufacturability. Many companies were started in the segment and it is instructive to look at this since it is the most recent example of an area of technology where the basic cycle from foundation to exit is pretty much complete.

Design For Manufacturing

The need to continue using 193nm light at the 90nm technology node created a discontinuity. Ad hoc approaches would not longer be enough. Venture capitalists realized this was an opportunity, and also a number of manufacturing companies that had some relevant software internally. They created about thirty EDA companies in the early 2000s. There were also some existing companies that moved to bring products to market in the space.

In a cohort of companies founded to address a problem, there is a winner-take-all dynamic. This is true of most industries, not just EDA. Most of the profit goes to the #1 player, some goes to the

#2 player and #3 on down pretty much either break even or lose money. For startups, the #1 player is acquired for a lot of money, #2 for a reasonable sum, and everyone else become what VCs optimistically call a technology sale, meaning they'll take whatever they can get to get it off their hands, like selling a car for spare parts.

The earliest company into the space was OPC solutions, which was probably too early. Mentor acquired it in 1998 and used as a starting point for its DFM solutions in Calibre, which is still the market leader in OPC. Then Numerical Technologies, which Synopsys acquired in 2003.

If we look at the next generation, the companies fall into three main groups.

Firstly, mask analysis: examining the polygons on the mask and checking whether they were matched what was meant to be there. This was mainly the province of the equipment vendors (plus Brion): KLA, AMAT, Brion, ASML and Nikon.

Next were the simulation companies. They would analyze the mask, work out the effect of all the wave interference of light in the stepper optics, simulate the lithography and work out what would end up on the wafer. This could then be used to check that it was close enough to the original layout, or to adjust timing or to search for hot spots, areas of the wafer where manufacturing problems (such as bridging of one piece of metal to another) were too statistically likely. Brion, Clearshape, ASML and Mentor all had products here.

Finally, optimization, working out the impact of the RET decoration and making changes to it to improve manufacturing yield. ClearShape and Blaze played here.

But there were dozens of other companies. Process optimization with HPL Technology, IC Scope, ISE, PAL, PDF solutions, Sigma-C, Silvaco, Stone Pillar, Syntricity. Preventing catastrophic failures and increasing yields were Anchor, CMP, Bindkey, ESCad, Prediction software, ChipMD, Invarium, , Ubitech and Xyalis. Hot spot tools and critical area identification came from Ponte, Mentor, Cadence, Synopsys. Mask optimization produced another group consisting of Aprio, ASML

masktools, Blaze, Brion, K2, Clearshape, Fortis, IC Scope, Magma/Mojave, Takumi and, of course, Mentor's Calibre. Phew, that's a lot of companies and I'm sure I've probably missed some too.

You will notice that most of these companies are no longer around. Actually they are around, just not independent. The big successes among the startups were Brion, which was acquired by ASML (a lithographic equipment company) for about $270M and Clearshape (acquired by Cadence for around $50M). K2 was also acquired by Cadence for an undisclosed, supposedly not large, sum.

Mentor's Calibre is probably still the market leader in OPC. Next are Synopsys's DFM tools which originally came from TMA (via Avant!) and Numerical Technologies. Cadence, despite a few acquisitions, are an also-ran. PDF solutions still exists (it is a public company). Blaze (which had already absorbed Aprio) existed as late as last November and its assets will show up somewhere soon. Takumi still exists, mostly doing business in Japan. Anchor still exists, with Xillinx among others as a customer. The rest are mostly gone or, in some cases, reabsorbed back into the parent that they were optimistically spun out of.

So what is the moral of the story. As usual, one conclusion is the venture capitalists make sheep look like independent thinkers. Every VC that invested in EDA wanted to have a play in the DFM space.

Another observation is that the technology was tricky: optics is not a normal part of EDA. The business models were trickier: did you sell to design groups, the manufacturing groups in fabless companies, the foundries themselves or the mask houses? How statistical (manufacturing) versus pass/fail (design) did you make things? Did you get a royalty of some sort or just license fees?

For sale, fab, cost $40/second

Semiconductor technology is a mass-production technology. Enormous functionality can be delivered in a chip that costs a few dollars. But only if you want to buy a lot of them. Further, to keep Moore's Law on track, the scale of manufacture keeps increasing. Chips were originally manufactured on circular wafers that were 1" or 2" in diameter, cramming as many of die onto the wafer as possible, and perhaps building a few wafers per day. Then wafers became 4", 6", 8". Today the latest fabs use 12" wafers and may manufacture 50,000 wafers a weekor more. At the same time, as the wafers have got larger the size of the elements on the chip have got smaller and smaller, going from over 10 microns to 30-90 nm today.

All the fab equipment is extremely expensive and the cost of a fab has gone from a few tens of millions of dollars to around $5-6B today. Since a fab has a useful lifetime of about three years, it depreciates at around $40/second. Taking into account all the other costs (silicon, design, marketing) means that to own a modern fab means a company must do business at $200/second or $6B/year. Few companies are this big and so not many companies, even those that call themselves semiconductor companies, can afford to own their own fabs any more. Jerry Sanders's comment that "real men have fabs" is no longer true at all. Only Intel seems big enough to go it alone on the manufacturing side, along with TSMC for foundry and Samsung for DRAM.

None of this is particularly positive for EDA, or the non-IP bulk of EDA. Fewer chips produced in higher and higher volumes is the EDA nightmare. The EDA dream is hundreds of companies designing chips, many of which don't even go into production, not far off the situation in the late 1980s and early 1990s when ASIC democratized design and pushed it out into the system companies. Not coincidentally this was also the heyday of EDA from a growth and business point of view.

Fab5

For some time I have been talking about the semiconductor industry as the Fab 5, since there have been five process “clubs”. A few players hedge their bets and are in more than one club. The fab five are Intel (a club on its own), UMC (along with Xilinx and Texas Instruments), IBM (along with Samsung, ST, Infineon, AMD, Sony, Freescale and Chartered), Japan Inc (Renasas, Toshiba, Fujitsu, OKI, Sharp, Sanyo, Matsushita) and the big one TSMC (with AMD, TI, NXP, ST, LSI, Sony, Qualcomm). Japan Inc in particular is messy with Toshiba tied closely to NEC (in the TSMC club but now merging into Renasas) but also to Sony (in the IBM club too), Renasas and Fujitsu are still sort of going it alone. Japanese politics would indicate that they will all get together somehow.

Big changes are afoot. Here are some of the things going on, ST, NXP and Ericsson wireless are all merged together into a new company (called, yawn, ST-Ericsson). Nokia has also sold its wireless unit to ST so it is presumably in there somewhere. Toshiba looks like it is going to really join Japan Inc (as if there was any doubt). TI and Freescale are both trying to find a home for their wireless groups but nobody wants them at a price they want to sell. The IBM club have deepened their technology agreements and ARM (although fabless) seems to be sort of joining the IBM club to help create energy-efficient SoCs, with Samsung both building and consuming the volume (and so I hereby rename the IBM club the Samsung club).

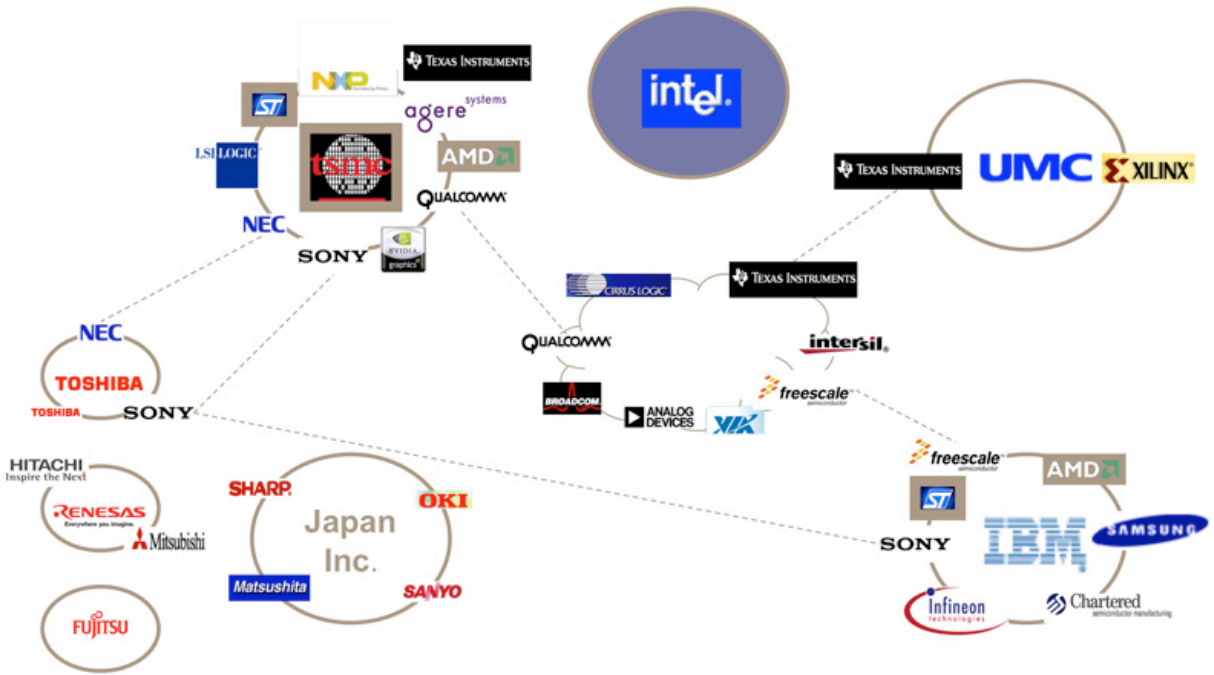
What about everyone else? AMD, ATI (also in AMD for now), MIPS, nVidia, UMC, NXP, Infineon, Motorola, Texas Instruments, Freescale were all bleeding cash even before the downturn got really bad, and they are reducing their footprints. All of Japan Inc except maybe Toshiba were also bleeding money (and Toshiba would have been except for all that flash going into phones and iPods, and is now hurting more after losing Xilinx to Samsung over price).

So based simply on financial strength it looks like the 3 fabs are going to be TSMC, Intel and Samsung (taking over the name badge for the IBM club) long-term. Of course other people like

ST won't lose their fabs overnight but they won't be able to afford to keep up. And it is unclear how many of the memory houses will make it through the current downturn. Qimonda is clearly comatose already and isn't going to wake up.

So the Fab 5 will become the Fab 3. For EDA this just emphasizes that there are too many EDA companies, as I've said before. Or maybe that EDA will go internal again, which is a discussion for another day.

Who would have predicted 20 years ago when TSMC was a small foundry with a non-competitive Philips process that it would be the dominant player. Kind of like predicting that Ringo would be the last Beatle of the Fab 4…oh wait, maybe that's going to happen too.


NXP
Texas Instruments
ST
agere systems
LSI LOGIC
tsmc
AMD
QUALCOMM
NEC
SONY
intel
Texas Instruments
UMC
XILINX
CIRRUS LOGIC
Texas Instruments
QUALCOMM
intersil
BROADCOM
ANALOG DEVICES
freescale
NEC
TOSHIBA
TOSHIBA
SONY
HITACHI
Inspire the Next
RENESAS
Mitsubishi
FUJITSU
SHARP
OKI
Japan Inc.
Matsushita
SANYO
freescale
AMD
ST
SONY
IBM
SAMSUNG
Infineon
Chartered

Arma virumque cano

Of arms and the man I sing. ARM is the leading microprocessor vendor in the world, at least if you count the right way. Over 10 billion processors have been shipped and the 1.5 billion mobile phones per year must contain at least another 1 or 2 billion. That's about 5 million per day or 100 per second. That's a lot of compute power.

I have a long history with ARM, although I never worked for them. Acorn (the A in ARM originally stood for Acorn, a British personal computer manufacturer) decided in about 1983 to design their own RISC processor for their next generation product instead of continuing to use the 6502. They also decided to use VLSI Technology to manufacture it.

Back then there was no real EDA industry, design tools were captive inside semiconductor vendors. If you wanted to do a design with VLSI Technology then you did it with VLSI Tools. This was also way before VLSI had offices in the UK or even Europe. So the tools needed to be installed, but we had no local application engineers, so I was the guy that got sent, presumably because I was British, even though I was a programmer not an AE. Anyway, as a result, I installed the design tools on which the first ARM was designed. The lead designer who would use them was Jamie Urquhart who eventually went on to be CEO of ARM for a time.

Acorn fell on hard times as the PC market consolidated and it was acquired by Olivetti (yes, the typewriter people from Italy although by then they were in electronics too).

In 1989, Apple decided to build the Newton. The back-story is actually much more complicated than this. Larry Tesler of Apple looked around the various processors that they might use and decided that the ARM had the best MIPS per watt, which was really important since battery life was critical (the Newton wouldn't be any use at all if its battery only lasted an hour) but the computation needs to do handwriting recognition and other things were significant. But they also decided they couldn't use it

if the design team and compiler teams were all buried inside a minor division of Olivetti.

So ARM was spun out as a joint venture between Acorn/Olivetti, Apple and VLSI Technology. I had to fly from France, where I was by then living, to a mysterious meeting in Cambridge. I wasn't even allowed to know what it was about until I got there. VLSI provided all the design tools that the nascent company needed in return for some equity, 5 or 10% I think, and also built the silicon. Remember, at this stage the idea was not to license the ARM widely, but rather to sit on the rocket-ship of the Newton as Apple created an explosively growing PDA industry. John Sculley, Apple's CEO, was publicly saying the market for PDAs and content would reach $3 trillion. VLSI would sell ARM chips (this was just before a processor was small enough to be embedded) to other companies for other products and we would pay ARM a royalty plus pay them engineering fees to design the next generation. Or something like that, I forget the details.

Well, we all know how the Newton story played out.

Back then, microprocessors were not licensed except in extremely controlled ways. They would be second-sourced since large customers didn't want to depend on a single semiconductor supplier in case their fab burned down or some other disaster interrupted supply. For instance, AMD originally entered the x86 business as a second source to Intel. VLSI was a second source to the Hitachi H8. The second source could also do its own business with the processor but it was never expected to be significant (hence all the lawsuits between Intel and AMD when AMD turned out to want to compete seriously against them).

Once it was clear the Newton was not going to be a success, VLSI continued trying to sell ARM and ARM-based designs to other customers. But nobody had heard of ARM and they were very reluctant to use what was then a largely untried microprocessor. There was too much technical risk.

Meanwhile, ARM had to work out how to make some money other than selling through VLSI. I have no idea if it was deliberate but just like IBM thought nothing of letting Microsoft license DOS to others (who would license it?) ARM had

complete freedom to do this. Under Robin Saxby (now brave, brave Sir Robin) they licensed a dozen semiconductor vendors. Suddenly for VLSI Technology, nobody worried about technical risk any more, they had heard of ARM and wanted it. But VLSI also had a dozen competitors with almost the same product.

Also, around this time, cell-phones were transitioning from using 8-bit microprocessors for their control processors to delivering more compute power. They largely skipped 16 bit and so the ARM7 (or more accurately the ARM7TDMI) was designed into a good percentage of cell-phones. And luckily cell-phones did promptly take off like the Newton rocket was supposed to have done.

VLSI's cell-phone business exploded too, with Ericsson representing almost 40% of VLSI's total business at one point, almost all of it whatever was the current version GSM baseband chipset. Ironically, Ericsson, at that time, didn't use ARM, they used their own implementation of the Z80.

When Compass was spun out from VLSI Technology, we inherited the ARM deal, namely providing everything ARM needed for free. Of course VLSI didn't see fit to give us the ARM equity that was the payment for this, or Compass would have ended up being wildly profitable. It fell to me to renegotiate the terms with Tudor Brown (now President of ARM). It was difficult for both sides to arrive at some sort of agreement. ARM, not unreasonably, expected the price to continue to be $0 (which was what they had in their budget) and Compass wanted the deal to be on arms-length(!) commercial terms. It was an over-constrained problem and Compass never got anything like the money it should have done from such an important customer.

I eventually left Compass (I would return later as CEO) and ended up back in VLSI where one of my responsibilities was re-negotiating the VLSI contract with ARM for future microprocessors. It is surprising to realize that even by 1996 ARM was still not fully-accepted; I remember we had to pay money, along with other semiconductor licensees, to create an operating system club so that ARM in turn could use the funds pay Wind River, Green Hills and others to port their real-time

operating systems to the ARM processor. Today they could probably charge for the privilege.

The business dynamics of ARM have certainly come a long way.

Japan, lost in introspection

There has been a lot of speculation about what will happen to the Japanese electronics companies, and in particular their semiconductor divisions, all of which are bleeding money.

If you visit Japan you get some idea of the problem. Everything is too inward looking. All the mobile phones are great and seem in some ways to be ahead of what we have in the US, and they are all made by Japanese manufacturers. But that is the problem, they are made by manufacturers who have given up in the rest of the world.

Greg Hinckley, the COO of Mentor Graphics, once told me about interviewing a candidate for a finance position who came from American Airlines. Their focus, the candidate said, was to touch down 30 seconds ahead of United. It was as if Southwest and Jet Blue and all the rest didn't even exist. Being the best airline just meant being the best legacy airline: beat United, Delta and the others.

The Japanese cell-phone companies are like that. They are so competitive for their share of the Japanese market that they have given up on the global market and what it takes to compete there. Of course, the Japanese cell-phone transmission standards are different which means that you have to decide whether to compete in Japan, overseas or both. Those different standards may have looked like a giving a good unfair advantage to the Japanese since Nokia, Ericsson or Samsung were unlikely to focus on the Japanese standard first even during the initial high-growth period. Even today, Nokia, the world's biggest cell-phone manufacturer has less than 1% market share in Japan. But on the other hand the Japanese manufacturers have no market share in the rest of the world, which is orders of magnitude bigger. Sony is an exception (Sony is almost always an exception) but perhaps

only because it has a joint venture with Ericsson rather than going it alone.

Motorola had the same problem in the digital transition away from analog phones, where it was the biggest manufacturer in the world. The rest of the world went digital with GSM (which back then stood for Groupe Spécial Mobile before it got renamed as Global System for Mobile communications). The US initially decided to simply make the voice channels of their analog system AMPS into digital channels to form D-AMPS, which was what AT&T wanted. So Motorola had to focus on making handsets and base stations for that American-only standard (I think it was used in Israel too) and largely missed the transition in the rest of the world by focusing inward. Much later, AT&T gave up on IS-136 that D-AMPS had morphed into and switched to GSM (although the current AT&T uses GSM mainly because SBC and PacBell Wireles went with GSM from the start and ended up acquiring the old AT&T). When you look where it came from, it is amazing that Motorola's wireless division looks unlikely to survive.

I was in Japan most recently a year ago when I was CEO of Envis. On a completely off-topic note I finally did something I've wanted to do for a long time: I got up at 5 in the morning and visited the Tsukiji fish-market. I recommend making the effort, and with jet-lag you'll probably be awake at 5 in the morning in any case. Nothing like unagi (eel) and green-tea for an early breakfast.

Visiting Japan really is captured well in the movie "Lost in Translation." Being awake in the middle of the night with jetlag, the weird stuff on TV, the atmosphere of the bars in the international hotels, Shinjuku in the rush-hour. Unfortunately I've never had the Scarlett Johansen lying on my bed bit.

Visiting the usual semiconductor companies I got the feeling that they were all only competing with each other. By and large they were making chips to go into consumer electronics products for the Japanese market. There were obviously far more products and far more chips being done than could possibly make money, just like all those cell-phones and cell-phone chips couldn't be

making money (not to mention that the Japanese market is already saturated).

With too many companies, and too many uncompetitive semiconductor divisions, consolidation is to be expected. But Japanese politics is inward facing too and so they can only merge with each other and gradually move towards what I call Japan Inc in the semiconductor world (to be fair, this same issue is one that affects my American Airlines example; British Airways or Lufthansa is simply not allowed to buy a major stake, recapitalize them and clean them up because congress has laws preventing it). So it looks like gradually the semiconductor companies will consolidate into a memory company (Elpida) and a logic company and, based on past history, they won't take the hard decisions necessary to be competitive globally rather than just in Japan.

ARM, Atom, PowerPC

Xxx

What is a MID? It's a Mobile Internet Device also known as a netbook. A huge battle is brewing as to whether a MID is more like a smartphone or more like a PC. It has major implications in the microprocessor market, the operating system market, for the smartphone manufacturers, for Apple and probably even the wireless network providers. Let's look at the processors.

In the blue corner is Intel, obviously with a stronghold in the desktop and notebook PC market. They have AMD to contend with there but I'm afraid I don't see how AMD can survive and I predict they will fall by the wayside. But that type of chip is too big and power-hungry, not to mention expensive, for other markets and so they have come out with Atom, which is a low-end embeddable x86 processor. However, it is still burdened with the x86 instruction set, which means that it requires a large and

power-consuming instruction decode unit.

In the other blue corner is ARM, with a stronghold in the cell-phone market including the smart-phone market. All those 25,000 applications in the iPhone store run on ARM. Blackberries are ARM-based to, although just to add a wrinkle, manufactured by Intel (Intel acquired an ARM license when they acquired the semiconductor business of the old Digital Equipment Corporation, and renamed StrongARM to Xscale).

The battleground for the upcoming fight is the MID . These are notebook PCs with smaller screens and a much lower price point than a PC, but with larger screens than a smartphone. Intel with Atom is betting, along with Microsoft so far, that this market will demand windows binary compatibility and thus will require a Microsoft operating system and an x86 processor. ARM are betting that this is not true, that MIDs will hide the operating system, run new applications and so nobody will care what the underlying operating system will be. Which means that it will be some form of Linux such or perhaps Google's Android (or if Apple enters this market as expected, OS-X which also Unix under the hood). Lurking around, of course, are the other smartphone operating systems, Symbian and Windows Mobile although they seem unlikely candidates for major success in the MID space (but primarily running on ARM in any case).

The really interesting wrinkle is whether Microsoft supports ARM with Windows 7 for this space. That would not give complete Windows binary compatibility but if Office was available (not just the operating system) that could be a very compelling compromise. Intel would be the big loser of this since Atom has poor power consumption and higher cost and really its only attraction is backwards compatibility with full-size PCs.

The big downside to Microsoft of supporting ARM, apart from the engineering cost, is the fallout it would likely provoke with Intel. But Microsoft has done this before when, while publicly committed to Itanium, they ported Windows to 64-bit x86 with AMD. By the way, this was done using Virtutech virtualization technology (before I worked there) with the result that Windows64 booted successfully the first day silicon was available, an extraordinary achievement.

One other wrinkle is the manufacturing. ARM is, of course, available from a huge range of suppliers. Intel will build Atom-based parts but is not in the ASIC business. TSMC will build Atom-based parts based on their recent announcement. However, the TSMC press release talks of expanding the "Intel Atom's availability for Intel customers" which may just be marketing getting the word Intel in as many times as possible, or really may mean some serious restrictions on availability. Furthermore, the Atom is not a soft core and so can't be prototyped in FPGAs. Whether this is a critical success factor remains to be seen. Based on my previous experience dealing with Intel, they won't make any netlist available. Sometimes being paranoid to survive has its downside.

Lurking quietly in the 3rd corner of the microprocessor ring is PowerPC. This is heavily used in Avionics, automotive and networking (routers and cellular base-stations). It used to be the processor in the Mac, but Apple switched to Intel reportedly because they couldn't persuade IBM to produce a low power PowerPC to keep Macbooks competitive. Both IBM and especially Freescale manufacture chips using it but somehow it is off the radar compared to ARM and Intel. One interesting facet is that Apple acquired PA Semiconductor who were developing a very low powered version of PowerPC. Apple are rumored to be producing chips embedding this processor so future Apple MIDs and possibly even future iPhones could end up with PowerPC, although it seems unlikely that Macs themselves will switch back due to the body of software that has just been expensively converted to Intel.

Ignoring the PowerPC (which at most may be a player with Apple) the bottom line is that Atom is more power-hungry and more costly (because it really is more expensive to manufacture) than ARM. Intel may be banking on getting a generation ahead in manufacturing process as a way to reduce both power and cost, but that won't help anyone going through TSMC. ARM is much lower powered and so offers the prospect of a MID that has days of battery life (like the (ARM-based) Amazon Kindle has already, but with very different screen technology).

My gut feel is that a MID will be more like a souped up smartphone than a dumbed down PC, and so Atom will lose to ARM. In fact I think the smartphone and MID markets will converge. Microsoft will lose unless they port to ARM. There will be no overall operating system winner (like with smartphones). But a few minutes with Google will find you lots of people with an opposing view to mine.

ESL and software signoff

Gary Smith pointed out recently that one of the reasons that Cadence is struggling is that the fastest growing part of the market has been ESL, the most advanced design groups are using more and more ESL tools and Cadence has no offering in that space (although they have now introduced their CtoRTL product). Of course Gary is famous for predicting the last 5 booms in ESL but this time I think he might be right.

However, I think the problem may be worse than this, from an EDA perspective. The most advanced design groups such as Nokia and Apple aren't designing much at even the ESL level. Nokia has transferred its semiconductor design group to ST. Apple didn't do much (any?) semiconductor design, as far as I know, in the iPhone and what they did in the iPod was subcontracted to eSilicon and PortalPlayer. However, with the iPad they seem to be doing at least some design themselves again. The differentiation in most electronic systems is now in the software. But EDA companies can't say this too loudly even if they realize it, since the bulk of their money comes from semiconductor designers.

The opportunity for EDA would be to expand to encompass the entire design

process, at the very least the semiconductor, board, software subsystem, even if not the mechanical and manufacturing part. But nobody knows how to make money at this. It is probably a consulting business and it is quite possible that the current downturn will throw up someone who can put the pieces together. I'd bet on someone like PTC or Dassault rather than Synopsys or Cadence to do this though. They already see the bigger picture.

One missing link is modeling. To do software design for electronic products requires a model of the electronics, and it is hard to produce that automatically. As more transactional level SystemC modeling is done, and as technology from companies like Carbon improve, the models thrown off as a by-product of the semiconductor deign process are starting to be much more useful for this. ARM are switching to using automatically generated Carbonized models instead of writing their own cycle-level accurate models going forward, for example.

This moves us closer to what I call "software signoff" where the electronic design process becomes very software-centric. The purposes of semiconductors and microprocessors are simply to run the software fast enough and at low enough power to make the end-product successful. The underlying technology to do this is some mixture of high-level C/C++ synthesis, IP blocks, automatic assembly of peripherals, buses and device- drivers, modeling to link the hardware and software. In short, what we call ESL. But the perspective is a bit different. The purpose of software signoff is not to produce a chip for people to program, but rather to accelerate a software implementation with very little effort. Once the software implements what you need, it should be pushbutton (or at least fairly automatic) to build a chip or to map the software onto an existing platform.

I took a dig at Gary Smith for being early predicting huge growth for ESL, but I can remember preaching about software in semiconductor companies when I was at VLSI over a decade ago. So I was even further ahead of reality in predicting the move of differentiation to software.

The Economist on semiconductor

Jerry Sanders, the erstwhile CEO of AMD, was famous for saying that "real men have fabs." So of course it was interesting that AMD should be just about the first integrated device manufacturer (IDM) to go fabless when it sold off its fabs to Middle Eastern private equity that renamed them Global Foundries.

There's an interesting in a recent Economist (the magazine that insists on calling itself a newpaper) on the state of the semiconductor industry. They also have the view that there will primarily be three fabs, Samsung in memory, Intel in microprocessors and TSMC for foundry. The rest will be "nationalistic" ventures in need of regular government bailouts.

For instance, they open with a look at Saxony (Dresden) where there are two main fabs. Qimonda (bankrupt and unlikely to resurface in anything like its original form) and Global Foundries whose German fab I would describe as "not closed yet" depending on the health or otherwise of AMD. Part of the problem with fabs is that they have got too expensive even for governments to simply pour money into. Fabs are a capital-intensive business with long-leadtimes so they tend to be feast or famine. And right now in the current downturn they are famine. DRAM spot prices are a quarter of what they were a year ago; good money then, not so much any more.

One comparison that I hadn't thought of is that fabs cost a couple of times as much as a nuclear power station. They are increasingly automated so don't even create much employment, and with electronic systems increasingly removed from their manufacture, the high value part of the chain isn't helped by having a fab nearby. "Designed by Apple in California and manufactured in China" as it says on the iPhone box. With, I believe, an Infineon chipset presumably made in Germany.

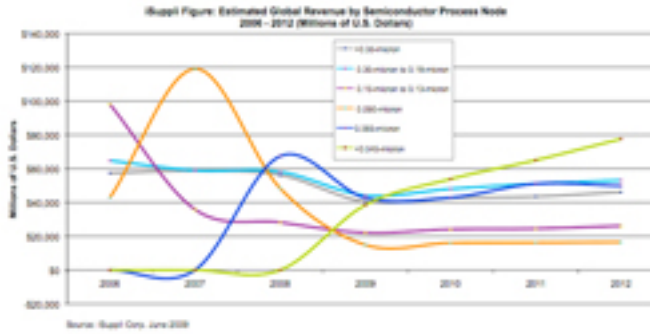
Europe, in particular, is in bad shape in semiconductor since it has so few fabless semiconductor companies (except in Israel, traditionally treated as part of Europe for the electronic market). European technology has always suffered from big company

syndrome, especially in France and Germany. I once asked a senior executive at Bull (then a large not-dead-yet French computer manufacturer) why there were no equivalents of Sun Microsystems in Europe. “Because the European governments would pour money into Siemens and Bull to build workstations, and we’d be successful enough to kill off any small companies but not successful enough to win,” was the gist of his reply. Semiconductor is like that: it’s all NXP (the old Philips Semiconductors), Infineon (the old Siemens semiconductor division), and ST Microelectronics (a merger of French Thomson Microelectronics with Italian SGS). All European champions who have consumed any Euros available for investment without actually achieving a world-class scale (ST being far and away the most successful of the three).

With 18” wafers maybe on the horizon (but with the semiconductor equipment manufacturers balking since they haven’t even yet recovered the investment they had to make for 12” conversion) the price of entry into the fab world is only going to go up. Semiconductor delivers chips incredibly cheaply, but it is a mass production process. And the required mass is going up not down, meaning greater and greater returns to scale. Intel talks of only requiring a single fab for their entire production and wanting separate fabs mainly for risk reduction (fire, political instability, losing the process etc).

iSuppli report on process transitions

A recent iSuppli report has been getting a lot of attention. It somewhat predicts the end of Moore’s law. If you look at the graph you can see that no process is ever predicted to make as much money at its peak as 90nm but that all the different subsequent process generations live on for a long time as a many-horse race.



I’ve often said that Moore’s law is an economic

law as much as a technical one. Semiconductor is a mass production technology, and the mass (volume) required to justify it is increasing all the time because the cost of the fabs is going up all the time. This is Moore's second law: the cost of the fab is also increasing exponentially over time.

So the cost of fabs is increasing exponentially over time and the number of transistors on a chip is increasing exponentially over time. In the past, say the 0.5um to 0.25um transition, the economics were such that the cost per transistor dropped, in this case by about 50%. This meant that even if you didn't need to do a 0.25um chip, if you were quite happy with 0.5um for area, performance and power, then you still needed to move to 0.25um as fast as possible or else your competitors would have an enormous cost advantage over you.

We are at a different point on those curves now. Consider moving a design from 65nm to 32nm. The performance is better, but not as much as it used to be moving from one process node to another. The power is a bit better, but we can't reduce the supply voltage enough, so it is not as big a saving as it used to be moving from one node to another and the leakage is probably worse. The cost is less, but only at high enough volumes to amortize the huge engineering cost, so not as much as it used to be. This means that the pressure to move process generation is much less than it used to be and this is showing up in the iSuppli graph as those flattening lines.

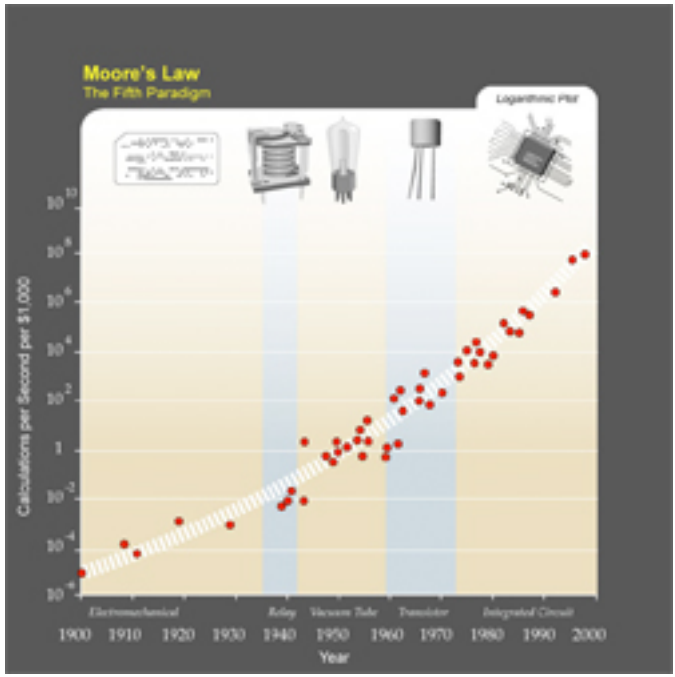
Some designs will move to the most advanced process since they have high enough margins, need every bit of performance, every bit of power saving, and manufacture in high enough volume to make the new process cheaper. Microprocessors, graphics chips are obvious candidates.

FPGAs are the ultimate way to aggregate designs that don't have enough volume to get the advantages of new process nodes. But there is a "valley of death", where there is no good technology, and it is widening. The valley of death is where volume is too high for an FPGA price to be low enough (say, for some consumer products) but the volume isn't high enough to justify designing a special chip. Various technologies have tried to step into the valley of death: quick turnaround ASIC like LSI's

RapidChip, FPGAs that can be mass produced with metal mask programming, laser programming, e-beam direct-write. But they all have died in the valley of death too. Canon (steppers) to the left of them, Canon to the right of them, into the valley of death rode the six hundred.

Talking of "The charge of the light brigade," light and charge are the heart of the problem. Moore's law involves many technologies but the heart of them all is lithography and the wavelength of light used. With lithography we are running into real physical limitations writing 22nm features with 193nm light, with no good way to build lenses for shorter wavelengths. And on the charge side no good way to speed up ebeam write enough.

So today, the most successful way to live in the valley of death is to use an old process. Design costs are cheap, mask costs are cheap, the fab is depreciated. Much better price per chip than FPGA, better power than FPGA, nowhere near the cost of designing in a state-of-the-art process. For really low volumes, you can never beat an FPGA, for really high volumes you won't beat the most advanced process, but in the valley of death different processes have their advantages and disadvantages.



However, if we step back a bit and look at “Moore’s law” over an even larger period, we can look at Ray Kurzweil’s graph of computing power growth over time. This is pretty much continuous logarithmic growth for over a century through five different technologies (electromechanical, relay, tube/valve, transistor, integrated circuit). If this logarithmic growth continues then it might turn out to be bad news for semiconductor, just as it was bad news for vacuum tube manufacturers by the 1970s. Something new will come along. Alternatively, it might be something different in the same way as integrated circuits contain transistors but are just manufactured in a way that is orders of magnitude more effective.

We don’t need silicon. We need the capabilities that most recently silicon has delivered as the substrate of choice. On a historical basis, I wouldn’t bet against human ingenuity in this area. Software performance will increase somehow.

What will I want from my devices?

Earlier this year, Paolo Gargini gave the keynote at DesignCon. He is an Intel fellow and director of technology strategy for their manufacturing group. He discussed three market enablers that would drive innovation and new products. He wasn't being particularly Intel-centric but rather looking at the industry, since this was a keynote. Those three drivers were:

Over a billion mobile Internet users

100 megabit/second wireless throughput

Availability of over a billion transistors for portable chip designs.

The first interesting thing to notice about these three drivers is that they are all portable. Most silicon is going

into one of two areas: portable devices, or server farms to form the compute cloud to talk to the portable devices. The first is much more important as a market since we all have a few portable devices of our own whereas we only make occasional use ("1,700,000 results in 0.22 seconds") of the cloud. Enormous though those server farms are, with literally hundreds of thousands of servers, they are a shared resource. I remember reading somewhere (I can't find the reference) that 2008 was the first year that more memory was shipped in cell-phones than in PCs. Even for memory, which has to be the majority of the silicon area in a server, the cell-phones contain more.

This is the end of a transition that has been going on for decades: that which was over the air is moving onto wires; that which was wired is moving onto the air. TV is moving from broadcast to cable (a transition that is largely complete). Telephone is moving from wire to wireless, a transition that is complete for anyone under about 30. My kids will never own a landline phone. When they move into an apartment they call the cable company for TV and internet; it doesn't cross their mind to call the phone company, they've already got a cell-phone.

Internet is halfway through the transition. Within the home and office it has largely moved onto the air to wireless routers, but then goes over wire backhaul. With smartphones like the iPhone, Blackberry and Palm Pre it is moving more and more onto the air completely, bypassing the router. It is not that far off that we'll be dropping our home internet service since we get all that with our cell-phone and our laptops have it built-in too, or maybe they parasitically piggy-back on our phones. We could today with our iPhones if AT&T would let us, or if we spend about 5 minutes on the internet to find the right file to install to turn on tethering anyway. But even 3G data is still too slow for more than occasional use.

Next up is the netbook space (or whatever they end up being called, apparently "netbook" is a Psion trademark). If all the intelligence is in the cloud we can get away with lower-powered machines at our end. Although there are some interesting technical and business issues (Atom vs ARM, Linux vs Android vs Windows vs Symbian) I think the most interesting challenge is

to decide how big we want our devices to be. I had a Palm for years, from back when they were still called Palm Pilot and were made by US Robotics. But I switched my Treo for an iPhone, but the screen is still too small for lots of things. I have a Kindle, great for reading but no color and a crappy keyboard. I have a MacBook but it is heavy and doesn't fit in my pocket, and not a great screen for reading a book on. I don't have the big KindleDX but the one person I know who does loves it. As screen and compute technology improve, the human interaction will be the limiting factor. Voice recognition seems to be pretty solid now, Nintendo Wii type technology works fine and there are demos out there of the same sort of thing without needing a controller at all, just a camera to watch you.

It is going to be fascinating to find out what I actually want.

Changes in relative value

It's interesting how the relative values of things change over time. Agatha Christie, looking back on her early life, remarked that she "couldn't imagine being too poor to afford servants, nor so rich as to be able to afford a car." I assume that by the time she died she drove but had no servants, like most of the rest of us.

One of the biggest drivers of changes in relative values has been the exponential improvement in semiconductor technology due to Moore's law. Even those of us in the business underestimate it. People just aren't very good about thinking about exponential change. I can remember running the numbers and working out (a long time ago) that we should have workstations that ran at 10MIPS, with a megabyte of memory and 100 megabytes of disk. What didn't even occur to me was that these would not be refrigerator-sized boxes, they would be notebook computers; or even Palm Pilots. And a high-end 1 BIPS "supercomputer" with 16 gigabytes memory and a 2 terabyte disk would have seemed totally unbelievable to me, even as I read the numbers off the graphs. But that's what I'm typing this on.

If you are not in a business where exponential change is the norm, people find it really had to think about. For example, in a study called "How laypeople and experts misperceive the effect

of economic growth" people were asked what would be the overall increase in national income in 25 years if it grew at 5% per year. Over 90% underestimated and only 10% of them were even within 50%. Surprisingly, the experts weren't much better than the laypeople. Quick, what is the percentage increase? See the end of the entry for the answer.

If the timescales are extended more then the numbers become even more dramatic. Alex Tabarrok in his TED talk showed that if the world GDP continues to increase at 3.3% per year for the rest of this century (below what it has been running at) then the average per capita income in the world will be $200,000. That's the *world* average, not the US which should be in the millions. Our great-grandchildren will be much richer than us (if we manage to avoid catastrophes like blowing up the world).

However, there is also a problem with our thinking when going the other way. Those of us in electronics and semiconductor tend to think other industries are basically like ours, with R&D driving an underlying exponential growth and thus the accompanying fast upgrading of old equipment. Battery technology, for example, doesn't increase exponentially in line with Moore's law. It would be great if an AA battery could contain 1000 times as much power as it could back in 1990, let alone a million times as much as it held in 1970. You'd only need one for your Tesla roadster.

Our cell-phones don't last too long, not because they break but because the new ones are so much more powerful. So we junk them after a couple of years, along with our computers. But that's not true for cars. No matter what great new change in cars happens (better MPG, lower emissions, super airbags, whatever) then it takes 20 years for most cars to get it. Many of the cars that will be on the road in ten years are already on the road today. Power stations, bridges, railroads, aircraft are all on even longer timescales. For example, I just looked and over 60% of all Boeing 747s ever built are still active, including some that first flew in 1969.

When part of life improves exponentially and part doesn't is when we get the type of dissonance that Agatha Christie experienced from unexpected changes in relative costs. Amazingly, and luckily, disk drive capacity has improved even

faster than Moore's law even though it depends (mostly) on different technology breakthroughs. But things involving large amounts of physical stuff, like metal, just can't change very fast. Henry Ford would be amazed at various features of our cars, but he'd still recognize them. Early computer pioneers wouldn't have a clue about a microprocessor.

The answer to what would be the overall increase in national income if it grows at 5% per year for 25 years is about 250%. A good rule of thumb everyone should know is that if something increases exponentially (compound interest) by x% then it takes 70/x years to double. So in this case it will double in 14 years and almost double again in 28 years. So about 3.5x in 25 years, which is a 250% increase.

What should EDA do next?

Which are the interesting areas of EDA right now? As a general rule, I think that the answer is "the ends" which today means the architectural level and the transistor layout level. There will always be some interesting areas in between too, of course, but the main flow from RTL to layout along with the respective verification methodologies are largely solved and so there is limited scope for major innovation.

The transistor layout level is really about the interface between EDA and semiconductor process. There are two things that make it a challenge. One is the changes in lithography which have complex effects on what can and cannot be put on a mask in a form that will print. The second is that EDA largely operates with a pass/fail model, whereas process is actually statistical. It is like the way we regard signals as digital, which works most of the time except occasionally the analog nature of signals breaks through when a signal changes too slowly or some other unusual effect causes the illusion to break down.

The architectural level is where chips and software intersect. Chip design people tend to think of the architectural level as somewhere that the system designers make a start on chip design. But a better way is to think of the software as a specification of the system and the only purpose of the chip is to run the software.

Why would you not just run code on one of the on-chip microprocessors? Only for 3 reasons: to do so would be too slow, to do so would consume too much power, or you can't do it in software without a special peripheral (for example, analog). Increasingly SoCs are processors, buses and memory, along with specialized IP blocks (which may themselves contain processors) for performance, power or analog reasons.

The big challenge in a system like that is getting the software right. I keep waiting for the virtual platform concept to really take off, since I'm convinced it is a better way to do development. Look at all the complaints about inaccuracy in the iPhone simulator (since it just cross-compiles) or the difficulty of doing performance analysis since you need to do it on the real phone. SoCs are much more complicated since typically they have multiple processors with different architectures since code running on (say) a Tensilica or ARC processor optimized for audio processing has very different characteristics from running the same code on an embedded PowerPC.

But the block diagram of the virtual platform is actually the chip specification as well.

I think that moving up to the architectural level should focus on this platform level. Like Goldilock's porridge, it is just right. It contains just the right amount of detail. By using the platform to run code, the software development can be done much more productively. By using the platform as a specification on how to integrate all the processors and IP, the chip can be created. It is like using RTL but at a much higher level. With RTL we can simulate it to get the chip functionality right, and we can use it as an input to a (fairly) automatic process to create the silicon. The virtual platform has the potential to play this role.

That would mean that the architectural virtual platform level would become a handoff between the engineers creating the systems and the lower level implementation. With synthesis timing was the unifying thread across the handoff; with this sort of architectural handoff it is communication within the software, which interacts with timing, functionality and power, of course, making it possible to optimize the SoC implementation.

People looking at ESL only as behavioral synthesis I think are missing the point. It is like software engineers arguing about details of language syntax. The hard problems are all about writing large scale software or integrating dozens (or even hundreds) of IP blocks quickly and getting the software working. Yes, behavioral synthesis has its place as the ultimate in "inlining" functions with extremely high performance and low power, just as in the software world people occasionally hand craft assembly code and sometimes measure cache hit-rates.

As Yoshihito Kondo, general manager of Sony's design platform division said, "We don't want our engineers writing Verilog, we want them inventing concepts and transferring them into silicon and software using automated processes."

That one sentence is a vision for what EDA should aim to become.

GM and Cadence

What is the similarity between the problems and GM and those at Cadence? Well, there are certainly plenty of differences, GM has all sorts of problems which stem from over-generous union contracts for one. But the thing that brought problems to a head at both Cadence and GM have a similar basis.

In the early 2000s GM and Chrysler (and Ford and Toyota and everyone else) sold a total of 16 million cars, small trucks and SUVs each year. With low interest rates, basically everyone who wanted and could just about afford one, bought a new car. Do the math: 16 million times 6 years is 96 million vehicles which is getting on for the number of households in the US (111 million), or 40% of all adults (228 million). Going forward, not many people are going to be buying new cars since everyone already has one. Sales are expected to be maybe 10 million this year and that may turn out to be optimistic. Cars last a long time these days so, unless you crash your car and have it written off, it is a discretionary purchase that can be delayed for years. This problem is not unique to GM, Toyota's sales numbers have fallen about the same. In essence, the auto industry sold in 6 years all the cars everyone would want for ten years while credit was easy

and consumer confidence was high. GM is just the most vulnerable due to bad management and bad union contracts, but less vulnerable since you and I will be picking up the tab for all this. I fully expect GM to behave just like British Leyland in Britain, also publicly owned, did. They'll lose a bucket of money but the government will just shovel more money into their gaping maw rather than see them shut down. (Here's another statistic: GM has to be worth more than it was in the early 2000s at its peak before the taxpayers to get any of their money back).

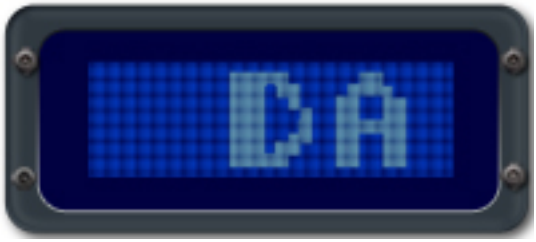
Cadence in the Fister/Bushby era repeated the same mistake from the Olsen era of selling in a period of time almost all the licenses anyone was going to need for a much longer period of time. Once you've sold in 3 years all the software anyone needs in 5 years, it gets hard to make your number in the out 2 years.

Of course, both these are problems that time will fix. Eventually people will want new cars again and probably GM will still be there to sell them (since we're covering all their losses). At least with Cadence it's not us that are going to pay. And with Cadence it is a much shorter time period. Probably by the end of next year they'll have eaten a good part of their way through the "supply tools, but accept that we've already been paid" and their number should start to improve.

GM may take longer. And, of course, both companies have other competitors (Synopsy, Toyota etc) ready to try to capitalize on any upturn.

But to put things in perspective, on the day I wrote this, Cadence's market cap at $1.4B was almost exactly twice General Motors ($700M).

Take the E out of EDA



As I said recently, I think Sony laid down the perfect long-range plan for the EDA

industry. Here's the money quote from Kondo-san again: "We don't want our engineers writing Verilog, we want them inventing concepts and transferring them into silicon and software using automated processes."

First, note that this is not just about designing integrated circuits. It's about the big strategic issue of how you design products. Those of us in EDA think that it is a fascinating industry with a strange combination of deep technology and a sufficiently large market to be an interesting business. As opposed to, say, TCAD, the software used to design semiconductor processes and develop process models without building silicon. I mean I'm sure it's interesting and there's a market but it's not EDA. It is a market of 5 PhDs in each fab in the world or about $20M/year. Certainly necessary but not at the top of anyone's list of problems on any given day.

Unfortunately, how we view TCAD is how the rest of the world views EDA: an esoteric geeky thing that some people need to get their job done but it's not solving the real business problem. Rocket science for rocket scientists.

There were always rumors that Cadence or Synopsys would buy Wind River, the leader in embedded operating systems and tools for embedded software development. I'm pretty sure discussions took place but obviously no deal was ever done and Intel bought them recently (as an interesting aside, that means that the PowerPC guys, primarily Freescale, are largely dependent on Intel for their RTOS and tools). So Wind River is on the verge of becoming Intel's captive embedded software capability. However, EDA companies thinking about acquiring Wind River was at least thinking in the right kind of way. How does the E get dropped from EDA? How does it just become Design Automation, encompassing everything from software to silicon, boards, packages, supply-chain management. In short, how does EDA achieve the Sony vision of inventing products and then implementing them using automated processes.

How many people in EDA know what a BOM is? It is a bill of materials, a list including the price, of every component in a product. In most consumer industries, design is getting the BOM right because otherwise the product cannot be built for a price

that the market will support. Design costs figure into the equation to some extent, but in the end for a volume product the final price of all the components is what matters. DA without the E is at least somewhat about BOM optimization.

RIM, the Canadian company that sells Blackberry, didn't actually design it. I don't know the details but I assume they came up with the basic concept and presumably wrote a lot of the higher-level software, both on the phone and on the server systems that implement the push mail. But then they used an "automated process" to get the guts designed. They subcontracted it to TTPcom in Cambridge England who put a lot of experts in software and phone design on the job. They wrote all the Verilog, and the call processing stack and designed the radio. RIM stayed focused on the user experience and how to deliver that in software applications.

But that's not the true "automated process" Sony wants to have access to.

EDA press

Listen to all those marketing engines are revving up to a fever pitch waiting for the green light. But who should they pitch to? Customers, obviously, you eventually have to win the ground war. But what about the air war? There isn't really a press following EDA any more, but there are lots of us bloggers and some newsletters, and without really planning it we've become one of the channels that potentially marketing can use to reach their customers.

But it's a new game and nobody knows how to play yet. I've been approached by several PR agencies and marketing folk about product announcements, interviews and so on. Individual product announcements are not interesting to me, and I'm assuming you readers wouldn't want to wade through them all anyway. There are other places for that. But product announcements in aggregate are interesting: What are the new trends? Which new areas are hot? Which new startups are interesting in those areas? What hard problems are getting cracked?

It is a major challenge for a smaller company to get its message out in this brave new world. Big companies like Cadence and Synopsys have their own internal tradeshows and regularly meet customer executives to brief them. Somebody commented on one of my blog entries about a TSMC engineer saying “I don’t go to DAC any more; if I want to talk to an EDA company I make them come to us.” That’s fine as long as you know about the company, but if you take that attitude you’ll never find out early about hot new technology that might turn out to be important.

Remember Bill Joy’s law: no matter where you are, the smartest people are somewhere else. You just don’t know what is going to turn out to be important, so you need to look at it all. But it is increasingly difficult to immerse yourself in the stream of raw information that might allow you to spot something. In its heyday, when both Richard Goering and Mike Santarini and more were there, not much happened in EDA that you’d miss if you read EEtimes each week. Now, not so much.

That’s one reason that, for the time being, I think DAC remains strong. It’s the only place for that kind of serendipity. Everyone has a story of some major customer finding them by chance at DAC. Not the big companies of course (“Synopsys. I didn’t know you had any synthesis products!") but startups. When I was at VaST we acquired Intel as a customer (or “a large Santa Clara based microprocessor company” since I don’t think Intel likes anyone claiming them as a customer) when a couple of engineers happened to pass by the booth.

2009 was the first DAC I've been to where I was officially classified as "press." I got in for free as press, I got invited to various press/analyst events (but not all of them), I got invited to various other events since I'm on the press list. "I have seen the future and it is us." In some ways it feels like EDA has been abandoned by the traditional press so we'd better just do it ourselves, and with our deeper knowledge do it better. I don't know if I succeed but that's certainly part of what I try and do on this blog.

It’s not clear what the channels to reach customers are going to morph into. To tell the truth, since it is so unmeasurable, it was always unclear even how much EDA customers were reading the

right articles in EEtimes versus us EDA insiders keeping an eye on the competition.

Of course what is happening in the EDA trade press is mirroring what is going on in the wider world of journalism in general. Even the New York Times is struggling financially and probably will not make it in its present form. The San Francisco Chronicle's days are almost certainly limited. Time and Newsweek are hemorrhaging subscribers. Nobody knows what anyone will pay for (except the Economist, which seems to have some unique hard to reproduce formula). If it is hard to get your message out in EDA, it is also getting harder to get it out if you are Toyota or Colgate too. Nobody watches the same channels, they all have DVRs and skip ads, they don't read so many paper magazines. When everyone is watching YouTube and updating their Facebook wall, they are not learning about the existence of new products, even of ones they'd love to discover.

Mac and PC

The PC market is obviously one of the huge markets for semiconductors. I think that the semiconductor content in cell-phones (in aggregate) is now greater than in PCs but I can't find the reference I remember.

I was at Google I/O last year. One thing a friend had told me was that essentially all web development is now done on Mac. It seemed to be true. I would guess that only about 5% of the machines that I saw over those two days were Windows PCs, the rest were all Macs. Of course Apple is riding high with the iPod and the iPhone as well, it is no longer just a computer company (leading it to drop "computer" from its official name).

Steve Ballmer isn't worried, or if he is he is trying not to show it.

“Apple's share globally cost us nothing,” he said. “Now, hopefully, we will take share back from Apple, but you know, Apple still only sells about 10 million PCs, so it is a limited opportunity.” It might even be true, since a Windows license probably costs the same whatever the power of the computer.

The PC market is really a number of different markets at different price points, and Apple doesn’t play in the low end of the market. Apple has the same strategy in the phone market. Nokia is the volume leader by a long way, but a lot of that volume made up of very low-end low-margin phones. At the high end, where Apple plays, Nokia is way behind iPhone and RIM’s BlackBerry. In the consumer (as opposed to business market) iPhone is the clear leader. And it shows in the numbers: Apple makes more profit in mobile phones than Nokia (or anyone else for that matter).

But the change is starting to show up in the numbers. According to NPD, Apple has 91% market share for PCs costing over $1,000. Of course a cynic would say that’s just because Macs are so expensive, but these are the computers used by professional programmers, graphic designers and musicians. It is true that the average ASP of a Windows PC was $515 but for Mac it was $1,400. I would guess the profit is much more than 3 times as much per Mac as per PC.

So those “laptop hunters” ads have it correct. You can get a PC for much less than a Mac, but it’s not really an equivalent machine. Apple has gone from 60% of the over $1,000 market at the start of last year to that 91% number now. Last week Apple beat analysts estimates and shipped 2.7M Macs.

These numbers are retail sales, so it does ignore the PCs stronghold of large businesses that, presumably, buy through channels not classified as retail. The market seems to be splitting into two. Inside big businesses, programmers and users all use PCs and develop applications that run on top of their SAP and Oracle systems. And when it comes to cell-phones they use Blackberrys since they can be centrally administered. Outside big businesses and in internet companies, programmers and high-end users all use Macs. And when they get a cell-phone it’s an iPhone.

Microsoft announced its results too. They missed by $1B for a 17% year on year decline, and the Windows PC division declined 29%, nearly twice as bad as the overall company. Of course there's a recession on (although Apple doesn't seem to be noticing), and Microsoft is just on the point of shipping Windows 7, so this might eat into sales for a couple of quarters before, although it's never been particularly cyclical in the past.

So just like Nokia's CEO statement that iPhone is a "niche product," Apple ships "only 10 million PCs." But it ships all the high margin ones to the prime customer demographic. If you look at profit and not volume, you are not so keen to use words like "niche" and "only." Especially given that high end phones become low end phones as they get cheaper, not the other way around.

PowerPC

At DAC, I happened to bump into Kaveh Massoudian of IBM, who is also the CTO of power.org, the consortium that deals with all things PowerPC. I previously met him when I was at Virtutech which was the era when power.org was formally established. A little bit of history: PowerPC was created in 1991 jointly by IBM, Freescale (then Motorola Semiconductor) and Apple (Macs would all become PowerPC based before the switch to Intel architecture a few years ago). So it was always a multi-company effort. It was designed as a 64/32 bit scalable architecture from the beginning. power.org was created in 2004 to pull together the whole ecosystem around the architecture.

PowerPC is really the third architecture, along with Intel and ARM. Their high level strategy is to let Intel own the PC market, let ARM own the wireless market ("let" as in admit that it is game-over in those markets) and try and own as much as possible of everything else: video games, aerospace, military, networking, base-stations, automotive etc. Did you know that the Wii, the Xbox360 and the Playstation game consoles are all based on PowerPC? Of course MIPS is still around, as are other processors (especially in automotive) but they are largely confined to certain

segments. For instance, MIPS is dominant in the set-top-box market (all those DVRs).

The challenge that PowerPC faces is that, outside of video game consoles, most of these markets are not that large individually. To design an SoC really requires the possibility of shipping 10M units, and if the market is going to be shared with other competitors then that means a market of, perhaps, 50M units. There just aren't that many markets that big outside of PC, wireless and video-game.

Processor business is all about software. Once a software base has been built up then binary compatibility means that it is impossible to displace one processor with another based on any features of the processor. So power.org is focusing much more on the software dimension. IBM is putting some of the Rational technology together with (I'm assuming) Eclipse and so forth to create much more productive software development environments for embedded design. It has been a constant theme this DAC that somebody needs to do something to improve the way embedded software is developed. IBM, for one, seems to at least be trying.

Another thing I asked him was what he thought of Wind River being acquired by Intel. A lot of Wind River's business is tied to the PowerPC architecture; for example, the Boeing 787 software is PowerPC and Wind River based. It is the same the other way round. A lot of PowerPC business is tied to Wind River. It is not completely so, Montavista and Green Hills also have strong positions in certain end markets. Kaveh said he didn't entirely understand why Intel had done it. Wind River is not used much on Intel architecture designs, even Atom-based ones. Maybe Intel wants to change that or maybe they simply decided to own a key piece of their competitors' infrastructure. Intel has historically, of course, owned the PC (along with AMD shipping a little) but they've not done very well outside that space. With the Atom deal with TSMC they are clearly trying to change that and start to get traction in the SoC space. If systems are software and silicon, then Intel clearly intends to be a player there. Plus they now have a relationship with many PowerPC customers that they'd like to

understand better and, presumably, eventually switch to Intel silicon in the long term.

Semiconductor cost models

One of the most important and under-rated tasks in a semiconductor company is creating the cost model. This is needed in order to be able to price products, and is especially acute in an ASIC or foundry business where there is no sense of a market price because the customer and not the manufacturer owns the intellectual property and thus the profit due to differentiation.

For a given design in a given volume the cost model will tell you how much it will cost to manufacture. Since a design can (usually) only be manufactured a whole wafer at a time, this is usually split into two, how many good die you can expect to get on a wafer, and what the cost per wafer is. The first part is fairly easy to calculate based on defect densities and die size and is not controversial.

In fabs that run only very long runs of standard products there may be a standard wafer price. As long as the setup costs of the design are dwarfed by other costs since so many lots are run in a row, then this is a reasonable reflection of reality. Every wafer is simply assumed to cost the standard wafer price.

In fabs that run ASIC or foundry work, many runs are relatively short. Not every product is running in enormous volume. For a start, prototypes run in tiny volumes and a single wafer is way more than is needed although it used to be, and may still be, that a minimum of 3 wafers is run to provide some backup against misprocessing of a wafer and making it less likely to have to restart the prototype run from scratch.

Back when I was in VLSI we initially had a fairly simple cost model and it made it look like we were making money on all sorts of designs. Everyone knew, however, that although the cost model didn't say it explicitly the company made lots of money if we ran high volumes of wafers of about 350 mils on a side, which seemed to be some sort of sweet spot. Then we had a full-time

expert on cost-models and upgraded the cost-model to be much more accurate. In particular to do a better job about the setup cost of all the equipment when switching from one design to the next, which happened a lot. VLSI brought a design into production on average roughly daily and would be running lots of designs, and some prototypes, on any given day. The valuable fab equipment spent a lot of the day depreciating while the steppers were switched from the reticles for one design to the next. Other equipment would have to be switched to match the appropriate process because VLSI wasn't large enough to have a fab for each process generation so all processes were run in the same fab (for a time there were two so this wasn't completely true). Intel and TSMC and other high volume manufacturers would typically build a fab for each process generation and rarely run any other process in that fab.

The new cost model shocked everyone. Finally it showed that the sweet spot of the fab was high volume runs of 350 mils on a side. Large enough that the design was complex and difficult (which we were good at) but small enough not to get into the part of the yield curve where too many die were bad. But the most shocking thing was that it showed that all the low volume runs, I think about 80% of VLSI's business at the time, lost money.

This changed the ASIC business completely since everyone realized that, in reality, there were only about 50 sockets a year in the world that were high enough volume to be worth competing for and the rest were a gamble, a gamble that they might be chips from an unknown startup that became the next Apple or the next Nintendo. VLSI could improve its profitability by losing most of its customers.

Another wrinkle on any cost model is that in any given month the cost of the fab turns out to be different from what it should be. If you add up the cost of all the wafers for the month according the cost model, they don't total to the actual cost of running the fab if you look at the big picture: depreciation, maintenance, power, water, chemicals and so on. The difference is called the fab variance. There seemed to be two ways of handling this. One, which Intel did at least back then, was to scale everyone's wafer price for the month so it matched the total price. So anyone

running a business would have wafer prices that varied from one month to the next depending on just how well the fab was running. The other is simply to take the variance and treat it as company overhead and treat it the same way as other company overhead. In the software group of VLSI we used to be annoyed to have our expenses miss budget due to our share of the fab variance, since not only did we have no control over it (like everyone else) it didn't have anything to do with our business at all.

Software signoff again

What do you think the dominant design paradigm for electronic systems is going to be going forward?

As I've said before, I believe that it is going to be taking software, probably written in C and C++ , and synthesizing parts of it into FPGAs and compiling the rest into binary to run on processors in the FPGA. This is what I've been calling software signoff for a long time. It's more than just the software necessary to run on the FPGA or SoC. It is signing off hardware that co-optimizes the software. The idea that conceptually we need to get the software that specifies the system right, and then hardware design is just creating a silicon fabric (SoC or FPGA) which is able to run the software high enough performance and at low enough power (because otherwise why bother to do anything other than simply execute it). Power, performance and price, the 3Ps again.

There are two key pieces of technology here. The first is high-level synthesis, which should be thought of as a type of compilation of behavior into hardware. In the end the system product delivers a behavior or application. It is not as simple as some sort of productivity tool as RTL designers move up the next level. RTL designers will be bypassed not made more productive.

The other key technology is FPGA technology itself. Today FPGAs offer almost unlimited capacity and unlimited pins. FPGAs will become the default implementation medium. The classic argument for not using FPGAs used to be that you could

reduce the cost enough to amortize the cost of designing a chip. But very few designs will run in high enough volume to amortize the cost of doing an SoC or ASIC in today's most leading edge processes, and the cost and risk of dealing with the variability (in terms of simulating hundreds of "corners" and the difficulty of getting design closure) is rising fast. FPGA takes a lot of the silicon risk out of the implementation.

Did you know that FPGAs represent more than half the volume of leading edge process nodes at the big foundries like TSMC and Samsung? FPGAs are the first logic in a new foundry process and drive the semiconductor learning curve. This is due to FPGA structural regularity that is much like memories but in a standard CMOS logic process.

If you need to do a 45nm design then far and away the easiest approach is to go and talk to Xilinx or Altera. To design your own chip is a $50M investment minimum so you'd better be wanting tens of millions of them when you are done. Only the highest volume consumer markets, such as cell-phones, or the most cutting edge performance needs, such as graphics processors, can justify it.

The decline in the FPGA market in the current downturn conceals the fact that new designs in the largest and most complex devices is growing at over 30% CAGR. It may only be 12% of the market (which, by the way, is something over 15,000 designs per year) but it generates 40% of the FPGA revenue. These designs, and the methodology for creating them, will go mainstream until it represents the bulk of the market. Not just the FPGA market, the electronic system market. Designing your own chip will be an esoteric niche methodology akin to analog design today. Howeve these new high complexity FPGA require an ASIC-like design methodology, not just a bunch of low-end tools from the FPGA vendor.

The challenge for EDA in this new world is to transition their technology base to take account of this new reality and go where system-scale designs are implemented in FPGAs. That is largely not in the big semiconductor companies that currently represent the 20% of customers that brings 80% of EDA revenue. It is much more dispersed similar to the last time that design was

democratized with the invention of ASIC in the early 1980s that pushed design out into the system companies.

A lot of RTL level simulation will be required. And one of the high level synthesis companies will be a big winner. In the startup world there are a few companies attempting to offer HLS: Synfora, Forte and AutoESL. Synfora and Forte has been at it for a while (although Forte may be disqualifying themselves in this vision of the future by only supporting SystemC). AutoESL has started to make some progress as well, with one group at Microsoft using just this methodology. Mentor is the current leader with its Catapult synthesis; Cadence has created their own CtoSilicon technology. But Synopsys, who has synthesis running through their veins, have no real high level synthesis product (and, unless they are doing it with people who are unknown in the field, don't have one in development). Synopsys does have FPGA DNA through the acquisition of Synplicity. My opinion is that once it becomes clear which HLS company is going to win, Synopsys will likely acquire them and for a serious price to complete their FPGA offering.

Is silicon valley dead?

Here's a quote from Tom Siebel, the founder of Siebel Systems that pioneered customer-relationship management before Salesforce.com started to eat their lunch and Oracle bought them.

"I think Silicon Valley has been toppled from its pedestal. I think information technology is much less important in the global picture today than it was even 10-20 years ago. ... I think the areas where people will be making a difference and making important social and economic contributions will be in the area of energy and bio-engineering. While there will be contributions in bio-technology and bio-engineering and energy technology that will come out of the valley, I do not believe it will have the type of global leadership position in those areas that it did in information technology."

I'm not sure this is right. Firstly, I'm not sure information technology is less important than it used to be. Of course biotechnology and energy technology will become more

important and so IT will decline relatively, but I doubt it will decline absolutely. I mean it's not like the Internet is done innovating.

I'm a little skeptical about some of the energy technology. In the long term it will be very important. In the short term it is simply farming government subsidies which is not a recipe for creating game-changing technology. I'm with Vinhod Khosla that unless the technology makes sense in the context of China and India, then it is just feel-good technology akin to putting solar panels on the roof of the Moscone center in the world's most famously foggy city.

Biotechnology is clearly increasing in importance. But there seems to be plenty of it in Silicon Valley (which, of course, no longer produces any silicon directly since that is not the highest value part of the chain). Silicon Valley certainly knows how to do innovation, create companies, manage intellectual property, embrace change. Not attributes that are thick on the ground in, say, Detroit. Or Washington, although it is now a big hub for technology simply because of the vacuum sucking money up from all over the US and disbursing lots of it locally in DC. Interesting to note that Morgan Stanley held their big partner offsite in Washington, not New York, for the first time ever. In industry after industry, success is starting to come from getting the government to write the rules your way rather than outright competition (see automotive, pharmaceutical, banking, insurance, to go along with the long standing agriculture, energy).

So I think that Silicon Valley will do fine (depending on California's state governance not getting into a tax/exodus death spiral as places like Detroit have done) although it is true that as technologies develop they also proliferate into other parts of the world. Silicon manufacturing was originated in Silicon Valley almost exactly 50 years ago; now it is mostly in Asia. A lot of the ideas in the PC (and Mac) originated in Silicon Valley at SRI and Xerox PARC (not to mention Berkeley and Stanford) but development is done everywhere. I'm sure other technologies will be the same.

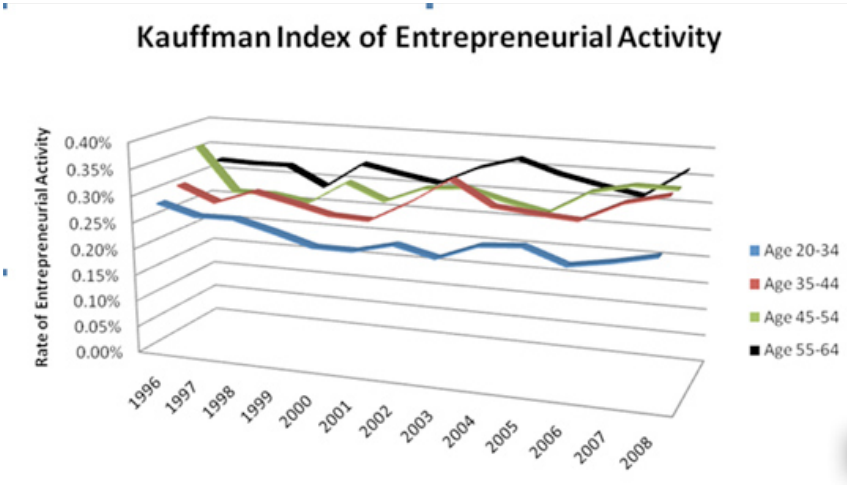
I think Silicon Valley has two unique attributes. Firstly, in the same way as Hollywood is the easiest place (but not the cheapest)

to make a movie since all the infrastructure is there, Silicon Valley has all the infrastructure for creating innovative companies. And secondly, Silicon Valley (and to a lesser extent other parts of the US) is good at sucking in the best talent from all over the world and integrating them into companies and making them productive. The company I (a Brit) ran last year was founded by an Israeli, had an Iranian CTO, a French lead engineer, Dutch and Russian engineers, Korean and a Columbian AEs to go with a couple of born in the US americans. The VC board members and investors were a Cypriot and a Vietnamese immigrant. Just look at the first two people there: an Israeli founder and an Iranian CTO. I don't see that happening in Tel Aviv or Tehran. They had to come here to do that.

Entrepreneurs ages

Entrepreneurs are all twenty-somethings straight out of college these days aren't they? Not so fast, it turns out that this is an illusion. It's probably true in some areas, such as social networking, where the young are the target audience too (at least initially).

But the Kauffman Foundation has done some research on the ages of entrepreneurs which they announced earlier this summer. Take a look at the chart below. First, I apologize for how hard it is to read, Edward Tufte would not be pleased (and if you don't know who Edward Tufte is then rush and buy "The visual display of quantitative information" immediately, and then perhaps his other books too).



Source: Robert W. Fairlie, Kauffman Index of Entrepreneurial Activity, at http://sites.kauffman.org/pdf/KIEA_041408.pdf.

The chart shows how recently, if anything, older entrepreneurs have been increasing. But at the very least is shows that there are plenty of entrepreneurs at all ages.

Of course there are entrepreneurs who are even younger too. My son works for YouTube and the most popular channel there is an annoying teenage kid called Fred who speeds up his voice. He is on track to be the first YouTube millionaire and is currently making over $50,000/month by selling ads and merchandize. There are 1.5M people subscribed to his channel.

Talking of being entrepreneurial, here's one of my ideas. Most sites on the internet are largely developed by people for people like themselves, at least initially. On this basis I think older retired people must be an underserved demographic. They are also the richest age-group (in most countries there is a vast transfer of money from young to old going on in ways that will not be sustainable for the current younger generation by the time they are old). And they have lots of time, which is a scarce commodity. And many of them, although not all, are online. So I haven't managed to think of a great idea but I firmly believe that this is a great place to look for an opportunity.

So entrepreneurs come in all ages although the kids that make it big seem to get all the publicity.

Designing a chip is like

You've probably tried to explain to somebody the unbelievable scale of what it takes to design a modern chip with hundreds of millions or billions of transistors. But even we have difficulty with numbers when they get that large, like when we hear that there are 500 billion galaxies in the universe. Large numbers just don't have that much impact. What's another trillion dollars on the national debt? One way to make that one clearer is that it is roughly the amount taken in annually in income tax. So $1T of debt means one year of everyone in the country paying double their tax.

I was talking to an architect yesterday evening who was familiar with AutoCAD ($3K/seat!) for 3D design and she was asking how similar that was to IC design.

The usual analogy I use for designing an integrated circuit is that it is like designing the Boeing 787 except doing it in 12 months using a manufacturing technology that has never been used before, on a design system that has never been used in production for that manufacturing technology. And by designing a 787 I mean all the parts, every part of every jet engine, every part of every seat, pump and instrument.

Of course some subassemblies might have been used before, such as the seats or the fuel-gauge (hey, IP-based design). But most things, such as the landing-gear, will need at least some change. Actually in terms of the count of parts this is underestimating things but it's not quite fair to compare a complex turbine blade with a single transistor and count both as one part.

But here's the thing I thought of last night that I've never articulated before. Having designed the 787 on the computer, you press a button and an amazing automated assembly plant take a couple of months to manufacture one. And then you put it on the end of the runway, put the throttles up to full and expect it to take off first time, using engines that have never run before and flight surfaces that have never flown before. Which it had better do, since it is already scheduled to come into service in November ready for the holiday market.

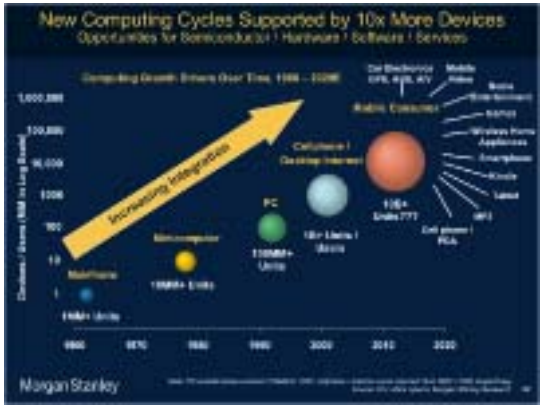
Then, unlike Boeing, the plane will be obsolete in 6 or 12 months. Next Christmas the 797 will be required, even bigger and more complex. But it will need to fly first time too.

EDA for the next 10 years

Last year at ICCAD, Jim Hogan and I led an discussion on the megatrends facing electronics and the implications going forward for EDA. Basically we took a leaf out of Scoop Nisker's book, who when he finished reading the news would sign off with "if you don't like the news go out and make some of your own." So we tried to.

Anyone whose being reading me regularly won't be surprised at the position that we took. I managed to find some interesting data from Morgan Stanley about how electronics is growing but it is also fragmenting. PCs ship in 100s of millions; cell-phones in billions (the world is expected to get to 100% penetration in a couple of years) and the fragmented consumer market in 10s of billions: car electronics, mobile video, home entertainment, games, kindles, iPods, smart-phones and so on.

So the end market is growing strongly but individual systems (with a few exceptions) are shipping in smaller individual volumes.



Meanwhile, over in IC-land the cost of design has been rising rapidly. For a 45nm chip it is now $50M. There are two problems with this for EDA. One is that the sticker price means that a lot fewer chips will be designed, and second that the fastest growing part of the cost is software (where EDA doesn't play much) now up to almost 2/3 the cost of design. But if a chip costs $50M to design then you'd

better be shipping it into a market of 250M+ units or the economics won't work.

So we have a mismatch: fragmented consumer market requiring low-cost low-volume designs. Semiconductor economics requiring high-cost high-volume designs.

The only way around this is aggregation at the silicon level, along with reconfigurability and reprogrammability.

The most basic form of aggregation is the FPGA, since the basic gates can be used for pretty much any digital design. It's not very efficient in terms of area or power, but it is completely flexible.

The second form of aggregation is the programmable SoC. This is something I've predicted for some time but I was surprised to discover recently that some manufacturers have been building these for several years. Indeed, Cypress gave me a chart showing that they are on track to ship 3/4 billion of this by the end of the year and should pass a billion next year. The programmable SoC doesn't have completely uncommitted gates like an FPGA, rather it has little building blocks for peripherals, both analog and digital, that can be reconfigured into a wide range of different devices. This can either be done one time to initialize the device, or it can be done dynamically under control of the on-board processor(s).

The third form of aggregation is the platform. This seems to be most successful in the wireless world, TI's OMAP being the most well-known. But it has also been happening in digital video. At some point it become more efficient to waste silicon by loading up a chip with everything you might ever want, and enable/disable by software, as opposed to eating the huge cost of masks and inventory of specializing each derivative to perfectly match the end customers needs.

Jim carried on to talk about which type of products make money in EDA. There is a range of types of tools from measurement, modeling, analysis, simulation and optimization. The further to the right on this list the more money customers are prepared to pay and the most likely it will be that you can create and sustain a competitive advantage for several years. Each tool needs to be better, faster or cheaper and preferably all three in order to be

successful. If you can only have two they'd better be better and faster. Cheaper in EDA has the same connotations as low-cost heart surgeon. With so much on the line that's not the place to economize.

Ultimately this is moving towards what I call software signoff, the inversion of the way about thinking about electronic systems. Instead of thinking of a complex SoC with some embedded software, a system is actually a big software system, parts of which need to be accelerated by some type of semiconductor implementation to make them economic (fast enough, low enough power). We don't have the tools today to take complex software and automatically build some parts in gates, assemble IP, assign the software to processors and so on. But that is the direction we need to move in.

The mismatch between fragmented end-markets and high costs of design is potentially disruptive and thus an opportunity to change the way that design is done. I return to Yoshihito Kondo of Sony's call to arms: “We don't want our engineers writing Verilog, we want them inventing concepts and transferring them into silicon and software using automated processes.”

The VHDL and Verilog story

VHDL is, of course, one of the two main hardware description languages dating back to the 1980s. The history of Verilog and VHDL is quite interesting. Verilog was originally created by Gateway Design Automation. Gateway was subsequently acquired by Cadence for what seemed like a very high valuation at the time, although of course it has probably been one of the most successful acquisitions Cadence did when you think of the sales of Verilog that they have made over the intervening years. VHDL, which is actually one of those nested acronyms since it stood for VHSIC Hardware Description Language, with VHSIC further parsed down into Very High Speed Integrated Circuit. The VHSIC program was run by the US DoD and VHDL looked for a time that it might become the dominant standard, since Verilog was a proprietary language owned by Cadence.

But Cadence opened Verilog up and let other people participate in driving the language standard. As Gordon Bell once said, the only justification for VHDL was to force Cadence to put Verilog into the public domain. But having two languages has been a major cost to the EDA industry for very little gain. VHDL was a very powerful language but in many ways was less practical than Verilog. For instance, you could define your own values for any signal. But that meant that gates from one library wouldn't necessarily interact properly with gates from another library (sounds like some of the problems with TLM models in SystemC that are finally being resolved). So that required a new standard, VITAL, so that gate-level signals were standardized. The richness of VHDL abstractions meant that it was and is used for some of the most complex communication chips. Model Technology (now part of Mentor) had probably the best VHDL simulator that they sold cheaply, and that helped to make VHDL more standard in the FPGA market than Verilog. Despite the fact that a Verilog simulator is easier to write than a VHDL simulator, it sold for a higher price for years. This has led to an odd phenomenon where some of the most advanced chips are done in VHDL, and many of the simpler ones.

Anyway, the dual language environment (and, of course, SystemVerilog has arrived to make a third) continues to exist. Almost all tools have, over the years, bitten the bullet and provided dual language support for both VHDL and Verilog. Often the front end for VHDL, which is a complex language to parse, comes from Verific (as does the VHDL front-end for Oasys's RealTime Designer).

Shake that EDA malaise

I have a sort of op-ed piece in Electronic Design that ended up being headlined “To Shake Its Malaise, EDA Must Look To Where Design Is Really Happening.” Journalists are constantly complaining about bad headlines being attached to their wonderful work, but in this case I think that the headline is a good summary of what I say.

The bottom line is that EDA, focused as it is on IC design in advanced processes, is focusing on a decreasingly important part of the overall electronic design process. Yes, you can't design a leading-edge chip without EDA so the market isn't going to go away. But most electronic systems use off-the-shelf chips rather than designing them from the ground up. There will always be a market for bespoke Saville Row tailoring of expensive suits, but the real market is at Macy's, Nordstrom's and Mens' Wearhouse.

Here's an example. The biometric company I work for has a fingerprint-protected USB drive product (that we got working the night before CES, it's not just taping out a chip that comes down to the wire). It contains some flash memory, a USB and hardware-encryption chip (standard product) and a programmable Luminary chip (now part of Texas Instruments). The whole system requires a fingerprint sensor and an OLED too, which obviously can't be integrated onto a custom chip in any case. Of course in volumes of hundreds of millions it would make sense to integrate the Luminary chip (which is an ARM processor with some standard peripherals) and the USB/encryption chip. But it will never ship in those volumes (I can dream) so I can't imagine that would ever make sense. Although, as a long-term IC guy, it upsets my sense of elegance to have two chips that clearly "should" be integrated, it is simply cheaper to use two separate chips. Most electronic products are like this: a handful of highly-integrated but standard chips on a little circuit board.

One theme that runs through this blog is that semiconductor economics drives everything. Semiconductor is a mass-production process that can deliver very cheap chips but only if the "mass" in mass-production is large enough. Otherwise the fixed costs overwhelm: the cost of design, the cost of masks and the fab setup times. The only alternative is to aggregate end-user systems so that the same chip is used in multiple designs. FPGAs

are obviously one form of aggregation, just buy raw gates and put them together later. The Luminary chip in the Biogy drive is another.

I certainly don't claim to have all the answers as to what the big EDA companies should do. But somebody needs to be the Mens' Wearhouse of EDA and serve the mainstream market, even though the unit price is lower. I guarantee it.

Chapter 2: Management

Three envelopes

Can there be any subject more boring than revenue recognition for software? If you listen to the conference calls of the public EDA companies, you'll either hear them discuss or get asked about how much of their business is ratable versus term. What does this mean? Should you care? Also, what does it matter how long the term is, isn't longer more money and so better?

When Jack Harding was CEO of Cadence, he lost his job because of these details. Cadence had been selling permanent licenses (for historical reasons I'll maybe go into at some point, EDA had a hardware business model). The sales organization had come up with the concept of a FAM, which stood for flexible access model. The basic idea was great. Instead of selling a permanent license valid forever, sell a license valid for only 3 years for not much less. Then, three years later sell the same license again. The lifetime of a permanent license had proved to be about 6 years in practice, so this was almost a doubling of the amount of money extracted per license. This was then scaled up into "buy all the licenses you will for the next three years today", with some flexibility built in by throwing extra licenses into the mix. This was done in a way that meant all the revenue, or most of it, could be recognized up-front.

There turned out to be two problems with this once it was scaled up. Firstly, the customers didn't really know what licenses they needed in year 3 although they had a pretty good idea about years 1 and 2. So to get them to go for this, the third year discount had to be huge. The second and bigger problem was that in two years the Cadence sales force closed three-year deals with every large account they had. The combination of these two things mean that every customer acquired all the licenses they needed for the next three years, but it was all booked in two years (and for not much more than two years' worth of money). Numbers looked great for two years but in year three there were no customers left. The

wheels came off and the numbers cratered. Jack Harding was CEO at the time and a couple of days after the quarterly conference call he was gone.

There's an old joke about a new CEO starting his first day and being left three envelopes by the outgoing CEO. He is told to open them when things get really bad. Things go OK for the new CEO for the first few months and then there is a downturn in business so he opens the first envelope. "Blame your predecessor" is on the card inside. So he makes speeches about how he inherited a company on the brink of ruin from the old CEO and the analysts and press give him a break. The second time things look bad, he opens the second envelope. "Reorganize." So the newish CEO takes all the business units and carves them up into functional divisions. That seems to fix business for a time. But eventually the future is not looking so bright any more and the now-not-so-new CEO opens the third envelope and read the card: "Prepare three envelopes".

So after Jack Harding left, Ray Bingham came in, opened the first envelope, and said Jack Harding and FAMs were bad, Cadence would henceforth book ratable business. Depending on details of the wording in the license, FASB (Financial Accounting Standards Board, a bunch of academic accountants from the east coast who've never run anything, but that's another story) forces the revenue to be recognized up front (like a FAM) or quarterly ("ratably") over the three years of the contract. With ratable business almost all of a quarters revenue comes out of backlog so it is very predictable, and there is a lot less pressure to close business at the end of a quarter (because only 1/12 will drop to the bottom line) which should lead to better discounting behavior.

However, there was pressure for Wall Street for growth and one way to provide that was to start to mix some term FAM-like business in with all the ratable stuff, since it dropped to the bottom line immediately. That was why smart analysts were so focused on what percentage of business was ratable. If you don't know that, you have no idea if the numbers are good or bad, or if they are sustainable.

[Full disclosure: Cadence acquired Ambit, where I was working, towards the end of Jack Harding's tenure. I then worked for Cadence for three years including working directly for Ray Bingham for a period. I left before Mike Fister came on board.]

Eventually the board brought in Mike Fister as CEO and Ray became Chairman. Mike Fister hadn't heard the joke, obviously, since he omitted to open the first envelope. He had a perfect opportunity to take a big loss, switch to ratable business and generally blame anything he wanted on Ray. Instead, he kept going on the same treadmill. To all of us observing from outside it was clear that Mike Fister wasn't going to make some new and interesting mistakes, he was going to make the same old mistake all over again. So it was no surprise when it turned out that the rate of growth was not sustainable, that they were booking ridiculously long-term deals of 5 or more years. The reason that this is bad is that a 5 year deal is not a green-field deal with a virgin customer, it is a 5 year deal with a customer who already has a 3 year deal, and customers don't pay much for a deal to buy software for years 4 and 5, let alone 6, 7, 8; they are not under any pressure. So eventually the wheels came off again. There was even some restatement of revenue associated with, surprise, whether some deals were correctly recognized as term or ratable.

So Mike Fister got to prepare his three envelopes and now we know it is Lip-Bu Tan who will open them. Watch for a big reset, blaming Mike for all the terrible deals he left behind, and lots of talk about starting with a clean sheet.

Semiconductor is not EDA

Executives from semiconductor companies regularly arrive in EDA companies convinced that their years of experience as customers mean that they understand the EDA business. Software people just need some of the discipline of semiconductor design, which the executive has plenty of, and a miraculous transformation will take place.

This view of the world makes the assumption that creating EDA software is just like creating a chip. After all, designing a chip is

done in Verilog, which is just a language, so how different can it be?

On the business side, it makes the assumption that selling software is just like selling silicon. After all, it is a technical sale, you take an order, you ship a product, how different can it be?

A lot.

On the engineering side, a chip has a definitive event when it tapes out. Software is never done. There are probably parts of Design Compiler shipping today that is code written in the late 1980s. Intel's latest microprocessor or TI's latest GSM chip or whatever doesn't contain stuff that old. Yes, IP blocks get reused, but not for decades. Even IP blocks have a different dynamic. If you need to cut a corner to get an IP block to work in your design, then you do it and tape out. If you find a bug in some software component then it needs to get fixed back at the canonical source. Otherwise, since the software lasts forever, there will forever be two versions, one containing your quick and dirty fix and one without. The result is that in software everything is much more inter-related than a semiconductor designer expects a lot of is it older than expected, and as a result there is also lots of code that works but is not well understood. Software development is just messier, and over time it gets worse.

There is also a different tradeoff in shipping a bug. Intel's cost to fix the floating point bug or nVidia's cost to fix their heating issues are hundred million dollar or billion dollar problems. While everyone has probably seen those tables showing that the cost of fixing a software bug once shipped is hundreds of times the cost of fixing it while the software is still in development, it is simply not a million dollar problem. Only products like the space shuttle guidance code can afford to spend astronomical(!) amounts on testing and have a long enough schedule to accommodate it. EDA software can't support that on either economical or schedule grounds. As a result, IC design really is more disciplined and spends a large amount, upwards of 60% of effort, on verification and almost no software can do that. When software is released it is not a bet the company event since bugs can be fixed.

The scale of software is also bigger. There may be billions of transistors on a chip, but many of them are in regular structures of one sort or another. No software is in regular structures or else it would have been further abstracted to get rid of the repetition. The number one rule in software development is to keep each thing in only one place. Cisco's IOS operating systems for routers is 25,000,000 lines of code. It is probably not clean but, by and large, there will not be a lot of duplication within it. It really is 25M lines of unique code. Chips do not consist of 25 million lines of Verilog.

On the business side there is an interesting difference between software and semiconductor. Firstly, semiconductor products typically have a lead time of the order of a quarter in length. This means that at the start of a quarter almost all the orders that will be produced that quarter are already in. Additional inventory might be built if there is spare capacity, in the hopes of selling it during the quarter (known as 'turns business'). Software really can receive an order at 11pm on the last day of the quarter and ship it for revenue before midnight.

However, the more interesting different dynamic is in negotiating. When a purchasing agent negotiates with an EDA salesperson they both know the marginal cost of the software: zero. It really doesn't cost any more to ship an additional copy of a software product. Semiconductor companies make sure that their salespeople do not know the manufacturing cost of the product (whether their cost models are good enough that they actually know it themselves is a different question). Marketing gives the salesman a price and perhaps some flexibility but neither the buyer nor the salesperson knows where the limits really are. Negotiations can be drawn out and nasty but there is a time aspect. If the buyer draws out the negotiations too long, they will not get their order submitted in time to get the product built. A software buyer knows that the biggest discount is likely as the quarter closes, and that the software company will still make incremental revenue no matter how big the discount.

There are probably other significant differences, but successful semiconductor experts can easily burn their fingers in the EDA business.

Finance

Finance is an area of business that is especially poorly understood by startup CEOs who tend to have engineering backgrounds, and underestimate the importance of everything else: account management, marketing and, of course, finance.

Let's start with how EDA companies report their results. If you listen to conference calls or read press releases you'll hear two sets of results. These are usually called GAAP (pronounced gap) and non-GAAP. GAAP stands for "generally accepted accounting principles" which actually doesn't mean generally accepted but means as mandated by FASB (pronounced fazz-bee), the financial accounting standards board. This ought to be a bunch of experienced company financial executives, or perhaps a bunch of experienced investors. But it is actually seven academics on the east coast who've never run a company, or even been in one.

Accounting is fundamentally about cash, and if you run a small business then you probably use cash accounting since it is the simplest. But in a larger company it does a poor job of matching income and expense flows together. For that you use accrual accounting. The first change is that revenue and expense recognition are separated from the receipt or expenditure of cash. This matters a lot in a true manufacturing business: you ship a customer a widget and a bill and eventually they pay. Much better to record the money on the same day as the widget went out since it is payment for that widget, and it is just a minor detail that the customer didn't pay for a few weeks. Similarly, you receive a widget and a bill and you record the expense that day, rather than worrying about whether you pay on 30 or 45 day terms. The second big change is that big-ticket items are not recorded as a single expense but are depreciated, recorded as a series of small expenses, over the life of the item (stepper, computer, fab). Again this does a good job of matching expenditure to use of the money. The stepper last for several years so it makes little sense to record a huge loss one quarter when you buy it, and unrealistically large profits for years while you use it. This is all pretty non-controversial although not so important for software businesses where a lot of money is not

tied up in manufacturing plant, inventory, work in process and so on. But GAAP doesn't stop there.

The trouble is that they have got so messed up with rules for depreciating goodwill, expensing stock options and so forth that they no longer really give a useful view of many companies' financial situations.

Two examples: a big EDA company buys as startup for $100M and the startup has assets on its balance sheet of just $5M. There are some wrinkles concerned with in-process R&D and capitalized software development, but most of the remaining $95M is called goodwill. It is essentially a plug number representing the difference between the price paid and all the tangible things anyone can find to assign to part of the purchase price. FASB (and so GAAP) used to insist that goodwill be depreciated over a certain period like 20 years, but now insists that each year the company evaluates the goodwill it is carrying on its books to see if it reflects a true assessment of the value of the acquisition and forces adjusting entries if not. That is why, for example, Ebay wrote down billions of dollars due to acquiring Skype when it became clear they paid too much and so had a big paper loss one quarter, that everyone ignored. However, changes like this are somewhat arbitrary and generate fictional gains and losses, not to mention assets on the balance sheet that aren't really assets (you can't do anything with them like sell them).

Second example: stock options. When options are granted, which at the time of grant has no effect whatsoever on the companies financial position, FASB (and so GAAP) insist that an expense be recognized. But there is no real expense in terms of money changing hands. So of course this theoretical expense is wrong, and later corrections will be required to bring it in line with what actually happened. If the stock price went down, the options might expire unexercised so it is just as if they were never issued, and the original expense will need to be reversed. If the stock price goes up, they will be exercised and the company will actually gain a certain amount of money from the exercise, and the number of shares outstanding will change. But the notional value will need to be reversed since in the EPS (earnings per share) calculation, option exercise affects the "per share" part and

not the “earnings” part, and pretending that it did messes up all the numbers.

Institutional investors ignore all this and focus on non-GAAP numbers, which take all that stuff back out again. In the case of a typical EDA company, non-GAAP numbers remove the depreciation of goodwill from startups acquired years ago, and also take out all the phantom value assigned to stock options. The non-GAAP numbers are much closer to what you need to assess how the business is doing. They are much closer to standard accrual accounting where cash payments are adjusted to do a better job of matching expenses and revenue to time.

For a really good summary of all that is wrong with FASB and GAAP I recommend T.J. Rogers, the CEO of Cypress Semiconductor, who wrote “Making financial statements mysterious”. It’s about 10 pages long. Here’s the opening paragraph:

I first noticed the misleading nature of Generally Accepted Accounting Principles a few years ago when an investor called to complain about the small amount of cash on our balance sheet. Since we had plenty of cash, I decided to quickly quote the correct figures from our latest financial report. But to my surprise, I could not tell how much cash we had either. With its usual—and almost always incorrect—claim of making financial reporting “more transparent,” the Financial Accounting Standards Board had made it difficult for a CEO to read his own financial report.

Of course I’m sure I’ve got some details wrong here. But that’s part of the point, finance is meant to summarize a business for executives and investors who are not deep finance experts.

Two million per salesperson

There is a rule of thumb that all EDA executives know (or have to learn expensively), which is that an EDA company thrives if its sales teams bring in $2M per salesperson. So a medium sized company with, say, 4 salespeople should have a booking forecast of around $8M and each salespersons quota should be about $2M.

For now let's ignore the difference between booking and revenue. Startups don't actually care about revenue that much, they care about cash. And cash comes a quarter later. The typical deal means that a startup must fund a sales team for the quarter, they close a deal in the last week, and the company receives cash in the middle of the following quarter. That time-lag, between the investment in the team and collecting the cash, is one of the main things for which series B investment money is needed. VCs have a phrase "just add water" meaning that the product is proven, the customer will buy at the right price. It should be a simple case of adding more money, using it as working capital to fund a bigger sales team and to cover the hole before the bigger sales team produces bigger revenue and pays for itself.

Where does this $2M rule come from? A successful EDA company should make about 20% profit and will require about 20% revenue to be spent on development. Of course it is more in the early stage of a startup, most obviously before the product is even brought to market but even through the first couple of years after that. Let's take another 20% for marketing, finance, the CEO and so on. That leaves 40% for sales and application engineers. The other rule of thumb is that a salesperson needs two application engineers, either a dedicated team or a mixture of one dedicated and one pulled from a corporate pool. If a salesperson brings in $2M then that 40% for sales and applications amounts to $800K, A fully loaded application engineer (salary, bonus,

benefits, travel) is about $250K. A fully loaded salesperson is about $300K (more if they blow away their quota). So the numbers add up. If the team brings in much less than $2M, say $1½M, then they don't even cover the costs of the rest of the company, let alone leave anything over for profit.

One consequence of the two million dollar rule is that it is hard to make a company work if the product is too cheap, at least in the early days before customers will consider large volume purchases. To make $2M with a $50K product, if you only sell two licenses at a time, is one order every two or three weeks. But, in fact all the orders come at the end of the quarter meaning that the salesperson is trying to close five deals with new customers at the end of each quarter, which will likely be impossible.

Of course, if a salesperson is new then they won't be able to achieve this. They have two strikes against them. Strike one, they don't know the product well enough to do an effective job of selling it. Strike two, they don't have a funnel of potential business as various stages of ripeness, from potential contacts, first meetings, evaluations and so on. So when a company is growing, hiring new people, the $2M quota is simply unrealistic. Even more money will be needed to cover the gap between starting to pay for a sales team until they are bringing in enough money to fund themselves.

I've put together no end of business models for software companies and the critical assumptions are always how long it takes a new salesperson to bring in any business, how fast they then ramp to the $2M level, and how many application engineers they need. You then can almost read the funding requirement off the spreadsheet.

Lady Windemere's FAM

In *Lady Windemere's Fan* Oscar Wilde wrote that a cynic is someone who knows the price of everything but the value of nothing. EDA companies are a bit like that. They only know the price of their tools.

How much money does Synopsys make on design compiler? Or Cadence sell of Virtuoso? The answer is that nobody really knows. Not even Synopsys and Cadence.

Of course the finance groups of EDA companies have their way of answering that question. They take the total number of licenses in a deal and add up all the list prices (for the appropriate time periods of course) and arrive at what is typically a very large number. They then take the actual value of the deal and from these two numbers (the deal size and the total value) arrive at a uniform access rate. Essentially they calculate a discount from list price assuming every tool received the same percentage reduction.

EDA companies didn't really plan this effect. They bundled large portfolios of tools (Cadence called them FAMs for flexible access model) as a way to increase market share, and for a time it was very effective. By the late 1990s, for example, Cadence roughly took in $400M per quarter and dropped $100M to the bottom line. Having difficulty in doing the accounting afterwards was just an unintended consequence.

However, the reason that this doesn't really work is that the list prices don't reflect value to the customer. The customer and the sales team don't really look at them. They think of the deal as delivering a certain design capability for a certain number of engineers, for a certain sum of money. Nobody wastes any time arguing that their Verilog simulation price is too high but they would be prepared to pay a bit more for synthesis, when the answer is going to be a wash in any case. That's both the strength and the weakness of bundling, or what is often but misleadingly called "all you can eat."

The biggest problem for EDA companies of this sort of accounting is that they lose price and market signals. Cadence didn't realize that it was losing its Dracula franchise to Mentor's Calibre until it was too late, since it never showed up in the numbers. Customers would simply refuse to pay so much for Dracula but the number of licenses in the deal wouldn't actually get adjusted, so the allocation of the portion of the deal to Dracula hid what was going on.

During the heyday of Synopsys's Design Compiler in the late 1990s, it was hard for them to know how much revenue to allocate to other products in the deal that might have been riding on its coattails. That's without even considering the fact that Synopsys would want to spread the revenue out as much as possible to look less like a one-product company to both customers and investors.

This problem is not unique to EDA. I talked to a VP from Oracle that I happened to meet and he told me that they have the same issue. Without getting signals from the market it is very hard to know where they should invest engineering resources. EDA has it slightly easier here since the march of process nodes guides at least some of the investment toward areas that everyone knows are going to become important. Technology as well as price determines the roadmap.

EDA companies fly somewhat blind as a result of all of this. If in every deal Verilog simulation is priced too high, and synthesis is priced too low, then this has implications for how much investment should go into synthesis versus simulation. But if nobody bothers to adjust them in each deal so that the price discrepancy eventually finds its way into the aggregate numbers, then investment will be misallocated. This is good neither for the EDA company nor for the customer, since both benefit from investment being in the places that the customer cares most about, as evidenced by their willingness to pay more for it.

Twelve-o-clock high

Three or four times in my life I've been given divisions or companies to run that have not been performing. Although it seems like an opportunity like that would be a poisoned chalice, it was actually a no-lose situation. If things went badly then I was drafted in too late. If things went well then I would be credited with the improvement. When expectations are so low it is not that hard to exceed them. Which is not at all the same thing as saying that improvement or success are easy.

When overnight I found myself as CEO of Compass Design Automation, one of my staff gave me the movie *Twelve o'clock*

high in which Gregory Peck takes over a bomber squadron during the second world war and turns it around. The previous commander had become too close to his men to be effective as a commander. It won some Oscars and still worth watching today.

It is a lot easier to make the changes to an organization as a newly-drafted boss than it is to makes those changes if you were the person responsible for the early decisions. Everyone is human and we don't like admitting that we made a mistake. We get emotionally attached to our decisions, especially to parts of the business that we rose up through or created. Nobody wants to kill their own baby. If you've ever fired someone that you hired or promoted, you probably discovered everyone around you thought, "what took you so long?" Reversing decisions that you made yourself tends to be like that.

As a newly drafted boss, morale will usually improve automatically just as a result of the change. Everyone knows lots of things that need to be changed and that were unlikely to be changed under the previous regime. It is a bit like the old joke about a consultant telling a manager something he already knows so that he can go ahead and do it. Just making some of those obvious changes fast creates a "things are going to be different" mentality.

The best example I know of the difficulty of reversing deeply ingrained decisions (without changing the leader) is in Andy Grove's book *Only the paranoid survive*. If you are less than a certain age you probably are unaware that Intel was a memory company, initially very successfully and then struggling against Japanese competition. Intel meant memories then in the same way as it means microprocessors today. Here's the scene. Andy Grove and Gordon Moore are in his office in 1985 discussing an upcoming board meeting. The business is going very badly:

> I turned to Gordon and asked, "If we got kicked out and the board brought in a new CEO what do you think he would do?" Gordon answered without hesitation, "He would get us out of memories." I stared at him numb then said "Why shouldn't you and I walk out the door, come back and do it ourselves?"

It was an extraordinarily brave decision, and laid the ground for what Intel has become today. Usually that type of wrenching change does require a new CEO who has no emotional attachment to the earlier decisions.

At the end of Twelve o'clock high the Gregory Peck character is removed from command. He identifies too closely with his men to be effective as a commander. Time for a new commander.

Startups and big companies: your end of the boat is sinking

I've worked at startups and I've worked at larger companies. I even worked at one company, VLSI Technology, where I joined it when it was a pre-IPO startup and left when it was thousands of people in tens of buildings. What is the difference? I think that the difference is best summed up in the jokey phrase "your end of the boat is sinking."

People talk about the "risk" of joining a startup, but the main risk, unless you are vice-president level or you are joining before the company is funded, is simply that you'll waste your time. You get paid pretty much the going rate for an engineer or a product marketing person or whatever you do. And you have some stock that will be worth a significant capital gain if the company is successful or nothing otherwise. If you are an executive, you get paid a lot less than the going rate in a big company. On the other hand, you have a lot of stock, 1-3% of the company for a vice-president, more for a hired-in CEO. Founders may have more than this depending on how much financing they end up needing to bring in. So for the senior people they really are losing something more than just time working for a startup.

Startups have two different dynamics from larger companies. The first is simply that they employ fewer people, pretty much by definition. Secondly, everyone's personal and financial success, especially the management, is bound up in the success or otherwise of the company.

Employing fewer people means that in a startup there is nowhere to hide. Everyone knows everyone else and it is clear who is performing and who, if anyone, is trying to free-ride on everyone else's efforts. In an environment like that, everyone is under pressure to perform. A startup can't afford much headcount and if you are not going to perform at a high level, or for some other reason are not a good match, then it is best for the startup to find someone else who will.

The second dynamic, that everyone's success is bound up with the company's success, means that people naturally are working towards the same goal. Startups often struggle as to what that goal should be, and different management teams do more or less well at communicating it, but it is not necessary on a daily basis to micromanage everyone's priorities. The natural DNA of a company that makes it operate in a particular way, which can be such a weakness in an Innovator's Dilemma situation, is a benefit here. If you don't tell people what to do there is a good chance they'll do what they should do anyway.

In a larger company, your success as an individual (unless you are in senior management) comes largely from doing what your boss expects you to do. This may or may not have something directly to do with the success of the company, but it is not your job to second-guess that. If you want a salary increase and promotion you must work within the system.

So in a startup you don't get "your end of the boat is sinking" behavior where people do what is good for their workgroup (division, site, product) at the expense of the good of the company. In a startup, where the boat is much smaller, everyone sees that the boat either floats or sinks and everyone is in it together. As a result, I find working in a startup is more fun than working in a larger company, at least unless you get senior enough to affect the large company strategy.

Ready for liftoff

I talked earlier about how it seems to take $6M to build a channel in EDA once you get to the "just add water" stage where all you need to do is to ramp up a salesforce and distribution. However,

typically you are not really ready for this when you first think you are. More EDA (and other) companies are killed by premature scaling than anything else. Ramping up a channel is very expensive and will burn a lot of money very fast for little return if the product is not ready, either killing the company completely or requiring an additional unexpected round of funding at an unimpressive valuation, diluting everyone's stock significantly.

When to scale is the most difficult decision a startup CEO faces. Too early and the company dies due to lack of financial runway. Too late and the company risks either missing the market window or losing out to competition during the land-grab phase of a new market.

There are two ways the product can be "not ready," and there will usually be a mixture of the two. The first, and most obvious, is that the first release won't include every feature that every user requires and so isn't ready to serve the entire market. This is probably even known and acknowledged; it's not as if engineering doesn't have a long list of stuff for version 2.

The more dangerous way the product is "not ready" is that you are not completely sure precisely which is the most important problem that it solves for the user, and which subsets of users will value this the most. Value it enough to consider engaging with you, an unproven startup with an immature buggy first release. For instance, you might have a product that you think serves the entire market, everyone will need one, you can't do 45nm designs without it. In fact, if you are just starting to engage with real customers, you might never have run a real 45nm design through your product, just doubled up 65nm designs and switched the library or something. It is often easier to get great results on those older designs. For example, in the last company where I worked, Envis, we got great power reduction results on 130nm designs, and public domain cores that had been around for even longer. After all, nobody cared that much about power back then so they didn't put much effort into designing to keep power under control. When we tried our tool on 90nm and 65nm designs, the results were initially less impressive. Designers had

already done many obvious things by hand meaning that we had to work harder to produce compelling incremental savings.

The reality is that your initial product certainly doesn't serve the whole market. If it does, you should have gone to market earlier with a less complete product. Worse, the precise feature set you have implemented might not serve any submarket completely either. But you need to have a product that serves at least some of the market 100%, even at the cost of being useless to a large part of the market, as compared to a solution that is 90% for everyone. At least in the first case there is at least one customer who might buy the tool; in the second case, nobody is going to buy the tool, everyone is going to wait for you to add the remaining 10%. A different 10%, perhaps, for every customer.

It is a fallacy to think that taking a product to market is a linear process. Do the engineering, prepare sales collateral, start selling. Taking a product to market is more of an iterative exploratory process. There is a phrase, "throwing mud against the wall to see what sticks" that sounds derogatory. But, in fact, the early stage of going to market should be like that. In the best of all worlds you'll have had one or two customer partners since the early stages of development, helping you spec the product and helping guide the early parts of development. But it doesn't always work out that way. Sometimes you don't have early partners, or your champion leaves, or sometimes those early partners turn out to be atypical in some important way, so that you are forced to choose between satisfying their unique requirements and developing features with wider applicability.

That leaves you with a product that you have to explore how to take to market, and to explore which if the many aspects of the product you should emphasize in your positioning. This is the City Slickers marketing problem, discovering the one thing that your customers value enough to buy the product and focusing your marketing and engineering on making that value proposition strong before worrying about other aspects of the product that might broaden the appeal to a larger segment of the whole market.

Customer support

Customer support in an EDA company goes through three phases, each of which actually provides poorer support than the previous phase (as seen by the long-term customer who has been there since the beginning) but which is at least scalable to the number of new customers. I think it is obvious that every designer at a Synopsys customer who has a problem with Design Compiler can't simply call a developer directly, even though that would provide the best support.

There is actually a zeroth phase, which is when the company doesn't have any customers. As a result, it doesn't need to provide any support. It is really important for engineering management to realize that this is actually happening. Any engineering organization that hasn't been through it before is completely unaware of what is going to hit them once the immature product gets into the hands of the first real customers who attempt to do some real work with it. They don't realize that new development is about to grind to a complete halt for an extended period. "God built the world in six days and could rest on the seventh because he had no installed base."

The first phase of customer support is to do it out of engineering. The bugs being discovered will often be so fundamental that it is hard for the customer to continue to test the product until they are fixed, so they must be fixed fast and new releases got to the customer every day or two at most. By fundamental I mean that the customer library data cannot be read, or the coding style is different from anything seen during development and brings the parser or the database to its knees. Adding other people between the customer engineer and the development engineer just reduces the speed of the cycle of finding a problem and fixing it, which means that it reduces the rate at which the product matures.

Eventually the product is mature enough for sales to start to ramp up the number of customers. Mature both in the sense that sales have a chance of selling it and the company has a chance of supporting it. It is no longer possible to support customers directly out of engineering. Best case, no engineering other than customer support would get done. Worst case, there wouldn't

even be enough bandwidth in engineering to do all the support. Engineering needs to focus on its part of the problem, fixing bugs in the code, and somebody else needs to handle creating test cases, seeing if bugs are fixed, getting releases to the customer and so on. That is the job of the application engineers.

During this second phase, a customer's primary support contact is the application engineer who they work with anyway on a regular basis. But as the company scales further, each application engineer ends up covering too many customers to do anything other than support them. Since their primary function is to help sales close new business, this is a problem. Also, AEs are not available 24 hours per day which can start to be a problem as real designs with real schedules enter crunch periods. So the third phase of customer support is to add a hotline.

The hotline staff are typically not tool users, they are more akin to 911 dispatchers. Customers hate them since they are not as knowledgeable as they are themselves. Their job is to manage the support process, ensure that the problem is recorded, ensure that it eventually gets fixed, and that the fix gets back to the customer and so on. It is not to fix anything except the most trivial of problems themselves.

However, it turns out that one problem the hotline can do a lot to help with, and that is problems with licenses, license keys and the license manager. In every EDA company I've been involved with this has represented almost half of all support calls. EDA product lines are very complex and as a result there are a lot of calls about licenses that don't require the intervention of engineering to get fixed.

At each phase of support, the quality (and knowledge) of the engineer directly interfacing to the customer goes down but the bandwidth of available support increases. Engineering can only directly support a handful of customers themselves. Each AE can only directly support a handful of customers but more AEs can easily be added as sales increase. A hotline can scale to support a huge number of customers 24 hours per day, and it is easy to add more hotline engineers. The hotline can also be located in an area where it is cheaper to staff, since it doesn't need to be in Silicon Valley.

This isn't specifically an EDA problem. I'm sure we've all had experience of calling customer support for Comcast or our wireless router, and been told to do all the things we've already tried. It's frustrating, but it's also obvious that they can't simply put us through to the guy who wrote the code in the cable modem or our router.

Test cases

I talked recently about customer support and how to handle it. One critical aspect of this is the internal process by which bugs get submitted. The reality is that if an ill-defined bug comes in, nobody wants to take the time to isolate it. The AEs want to be out selling and that if they just throw it over the wall to engineering then it will be their job to sort it out. Engineering feels that any bug that can't easily be reproduced is not their problem to fix. If this gets out of hand then the bug languishes, the customer suffers and, eventually, the company too. As the slogan correctly points out, "Quality is everyone's job."

The best rule for this that I've ever come across was created by Paul Gill when we were at Ambit. To report a bug, an application engineer must provide a self-checking test case, or else engineering won't consider it. No exceptions. And he was then obstinate enough to enforce the "no exceptions" rule.

This provides a clear separation between the AE's job and the development engineers job. The AE must provide a test case that illustrates the issue. Engineering must correct the code so that it is fixed. Plus, when all that activity is over, there is a test case to go in the regression suite.

Today, most tools are scripted with TCL, Python or Perl. A self-checking test case is a script that runs on some test data and gives a pass/fail test as to whether the bug exists. Obviously, when the bug is submitted the test case will fail (or it wouldn't be a bug). When engineering has fixed it, then it will pass. The test case can then be added to the regression suite and it should stay fixed. If it fails again, then the bug has been re-introduced (or another bug with similar symptoms has been created).

There are a few areas where this approach won't really work. Most obviously are graphics problems: the screen doesn't refresh correctly, for example. It is hard to build a self-checking test case since it is too hard to determine whether what is on the screen is correct. However, there are also things which are on the borderline between bugs and quality of results issues: this example got a lot worse in the last release. It is easy to build the test case but what should be the limit. EDA tools are not algorithmically perfect so it is not clear how much worse should be acceptable if an algorithmic tweak makes most designs better. But it turns out that for an EDA tool, most bugs are in the major algorithms under control of the scripting infrastructure and it is straightforward to build a self-checking test case.

So when a customer reports a bug, the AE needs to take some of the customer's test data (and often they are not allowed to ship out the whole design for confidentiality reasons) and create a test case, preferably small and simple, that exhibits the problem. Engineering can then fix it. No test case, no fix.

If a customer cannot provide data to exhibit the problem (the NSA is particularly bad at this!) then the problem remains between the AE and the customer. Engineering can't fix a problem that they can't identify.

With good test infrastructure, all the test cases can be run regularly, and since they report whether they pass or fail it is easy to build a list of all the failing test cases. Once a bug has been fixed, it is easy to add its test case to the suite and it will automatically be run each time the regression suite is run.

That brings up another aspect of test infrastructure. There must be enough hardware available to run the regression suite in reasonable time. A large regression suite with no way to run it frequently is little use. We were lucky at Ambit that we persuaded the company to invest in 40 Sun servers and 20 HP servers just for running the test suites

A lot of this is fairly standard these days in open-source and other large software projects. But somehow it still isn't standard in EDA which tends to provide productivity tools for designers, without using state of the art productivity tools themselves.

On a related point, the engineering organization needs to have at least one very large machine too. Otherwise inevitably customers will run into problems with very large designs where there is no hardware internally to even attempt to reproduce the problem. This is less of an issue today when hardware is cheap than it used to be when a large machine was costly. It is easy to forget that ten years ago, it cost a lot of money to have a server with 8 gigabytes of memory; few hard disks were even that big back then.

And with perfect timing, here's yesterday's XKCD on test-cases:

Strategic errors

In the time I was at VLSI, we made a couple of strategic errors relating to EDA. It is perhaps unfair to characterize them this way since it is only with hindsight that the view is clear.

First a bit of history. VLSI was created in the early 1980s to do what came to be called ASIC designs. To do that we had internal tools and they made VLSI pretty successful, first in ASIC and later standard product lines for PC chipsets and GSM phones. VLSI was a pre-IPO startup when I joined and it grew to a $600M company that was eventually acquired by Philips Semiconductors (now called NXP) in a $1B hostile takeover. In 1991 VLSI spun out its internal software as a new company, Compass Design Automation, which never really achieved success. It grew to nearly $60M and eventually (by then I was CEO of Compass) was sold to Avant! for about $90M depending on how you count in 1997.

But let's go back a bit. In the mid 1980s, VLSI had a major problem. It didn't have enough money. It didn't have enough money to build a 1um fab, and it didn't have enough money to fund TD (technology development, meaning development of the semiconductor process itself) for a state-of-the-art 1um process. So they did major strategic deals with Hitachi and Philips Semiconductors that brought in process technology, patent licenses and money. This meant that VLSI was in business in 1um as a semiconductor company with a new fab in San Antonio and the Hitachi 1um process up and running there.

The really clever decision would have been to foresee that the profitable part of the business was going to be EDA, and VLSI was one of the leaders, if not the leader, at that time. If they forgot about all this fab stuff, they wouldn't need the money, they wouldn't need the process, and they could be a very profitable software company. They could have been Cadence, which was just starting to get going at the time. The trouble was that they had semiconductor management whose deep operational experience of running fabs would have been pretty useless for running a software company.

In effect, this would have been spinning out the Design Technology group from VLSI to become Compass, and leaving VLSI as a semiconductor company to die or become an early version of an eSilicon type of fabless ASIC company (since it would have limited money, no process and no fab). But, in any case, eventually it became really obvious that combining a semiconductor company and an EDA company was not a good idea and it was time to split into two viable companies.

The second strategic error was waiting too late to do this. By 1991 when VLSI did it, their technology was no longer way out ahead of the competition, and the industry was not yet looking for the integrated solutions that Compass had (since it had not grown by acquisition). This meant that Compass always struggled to both acquire customers and to acquire library support from other semiconductor companies. This would have been helped if VLSI had sold part of Compass to a VC or someone independent, since there would have been at least part of the ownership of the company that didn't care about VLSI and only cared about the

value of the Compass stock. That would act as a guarantee of independent arm-length behavior by VLSI, which wasn't there when VLSI owned 100% of Compass (which it did until it was sold to Avant! in 1997). When LSI wanted to license our datapath technology, it was apparently vetoed by Wilf Corrigan because, if Compass's resources got tight, VLSI would get them and LSI would not.

Interestingly, a couple of years earlier in 1988 Daisy had made a hostile bid for Cadnetix, and had merged the companies to create Daisix. For a number of reasons, this never worked and eventually in 1990 they filed for bankruptcy. VLSI turned out to be owed a lot of money by Daisix (or one of its parents, I don't remember the details). Daisix was offered in settlement of the debt, but VLSI wasn't interested and it went to Intergraph instead. If they had taken Daisix, up and running as an EDA company with huge breadth of 3rd party library support, and merged it with Compass's technology then there was certainly the possibility for Compass hitting the ground running, rather than struggling to earn library party support from other vendors which eventually limited their market largely to people licensing Compass's libraries.

Emotional engineers

People sometimes say that salespeople are emotional, unlike engineers. I think what they mean is that salespeople are (stereotypically) extrovert so if you mess with them they'll make a noise about it. Whereas engineers are introvert and will just brood ("How can you tell if an engineer is extrovert? He stares at *your* shoes"). But actually salespeople are coin-operated. Change their commission plans and they'll cancel everything they were doing and do something else. They'll complain loudly but they'll do it. Engineers are much more emotional and have a lot invested in their products. Cancel their product and it takes a long time for them to become productive again. Unlike sales, they won't complain but they won't do it. They'll waste a lot of time talking amongst themselves about how management shortsightedly delayed the salvation of mankind instead.

Every EDA startup, at least the ones that get that far, goes through a difficult emotional transition with engineering when the first product finally starts to ship.

In the early days of a startup, almost the entire company (maybe everyone other than the CEO) is in engineering. The focus of every review meeting is engineering schedules. The focus of any HR activity is hiring that next great engineer. Everyone is waiting for that first release of the product. Every small slip of the product, every minor change of specification, is minutely analyzed.

Finally the big day arrives and the focus of the company switches very quickly to sales. How is the funnel? What is the booking forecast? How are we doing hiring that critical application engineer? How long to cash-flow neutral? To the extent that management pays attention to engineering it is more focused on when showstopper bugs that are impacting sales will be fixed. Nobody seems to care nearly as much about release 2.0 as they did about release 1.0.

Anyone who has more than one child has seen something similar. Their first child is an only child, the center of their parents' attention. Until that second baby arrives and suddenly they are no longer the center of attention. Firstly, there are now two children so attention would naturally be halved. But secondly, babies have an extremely effective strategy for getting all the attention they need: make an unpleasant noise and don't stop until their needs are satisfied.

When sales start, engineering is like the first child. They go from having all the attention to having to share it. And to make it worse, the second child, sales, has a very effective strategy for getting all the attention they need: explain the reasons they are not closing business until their needs are satisfied. To make things worse still, the reason they are not closing business is probably related to deficiencies in the early immature product, which means that what little attention engineering does get is negative.

This is a very tough emotional transition. Engineering is on the start of a path from being almost 100% of the company declining

to 20% of the company as it moves towards maturity. Engineering will hold headcount relatively flat as other parts of the company seem to explode. Engineering goes from being the star of the show to a supporting role.

The most important thing to do about handling this is to make sure everyone understands that it is going to happen, like telling your 4 year-old about the new baby. And, what is more, make sure everyone realizes that it is a symptom of success, a rite of passage. When all that anyone cares about is engineering, it means that the company isn't selling anything. When management cares about other things, that's the first taste of victory. It's engineering's job to get out of glare of attention as quickly as they can, and let sales start taking the heat.

After all, how much fun was it when the CEO was analyzing engineering's embarrassingly inaccurate schedules in great detail. Every day.

CEO: a dangerous job

Why do so few startup CEOs last the distance? The Bill Gates, Michale Dell and Scott McNealys who take their companies all the way from the early days as a tiny startup all the way up to enormous multi-division companies are very exceptional. I think that it is obvious that running a little engineering organization developing a technical product requires very different skills from running a large company. Engineering skills dominate in the first; people and strategic management skills dominate in the second. A CEO has to grow a lot along with the organization to be successful at each stage of the company's growth.

What is less obvious is that the skills getting a company going are very different from running it once the engineering phase is drawing to a close, or in some case just getting started. I've read various statistics, but something like 75-80% of startup CEOs are replaced before their company gets acquired (or merges, or goes public etc).

Getting a company started, and raising the first money to fund it, requires a level of focus and obsession that is abnormal. The

"doing whatever it takes" attitude is necessary in those very early days, but it tends to leave a trail of turds to be sorted out later. Further, some people like this have difficulty making the transition to being a team-player once the key hires have been made. Nothing will alienate high-performers more than trying micromanage them, or treating them without integrity, or generally not regarding them as close to equals. A startup is more like a jazz-band than a military organization. It is interesting that the highest performing small-scale parts of the military, Navy SEALs or the British SAS, abandon a lot of the military trappings (SAS officers famously are often called by their first names).

I've been in several startups where I've come in later, well after founding, and had to sort out problems that are left over from getting the company founded. Complete inequities in salary or, especially, stock seem to be the natural debris of getting people out of their current organization and into startups. But not getting them into the company is probably a worse problem.

There are no hard divisions between different stages in the life-cycle of a startup, but roughly speaking there are four. Getting the company founded along with the other initial founders; getting the engineering development solidly under way with a competent team; getting initial sales and starting to ramp up a channel; growth to a more mature organization with an industry standard breakdown of headcount. There is a fifth (and probably more) stages as it become more and more difficult to manage larger organizations. The largest organization I've run had about 600 people, and that is like sailing a supertanker. You think you spin the wheel but nothing happens.

At each of those four stages, the CEO may or may not make the transition. VCs are famously ruthless if they think that the CEO is not the best person to look after their investment. The old CEO, no matter how important he or she was in earlier days, is off to "spend some more time with their family" and a new person is at the helm overnight.

The most dangerous phase for many CEOs is the transition from engineering to starting to ship the product. Founding CEOs are often very technical, effectively the primary architect of the product. Like many engineers, they overestimate the importance

of technology and they underestimate the importance of marketing and account management. By their nature as founders, they may be much better at driving over objections than at listening. So they fail at the business side since they are out of their natural comfort-zone and they compound the problem because they won't listen to people who know what they are doing wrong. This happens so often that VCs see it coming from afar and don't even wait to see if the CEO can handle it before hitting the eject button. They knew when they founded the company that they would change the CEO. Sometimes they even make it a condition of funding, to make the process less traumatic when it happens.

Channel choices

Should a separate product be sold through a separate channel? If a new product is pretty much more of the same then the answer is obviously "no." If the new product is disruptive, sold to a different customer base, or requires different knowledge to sell then the answer is less clear. There seem to be several main inputs into the decision. Cost, reach, conflict, transition and disruption.

First, cost. Each channel costs money. Obviously a separate direct sales force, as Cadence once had for the Alta Group (its system level tools), is expensive. Less obviously, even a distributor or reseller has cost too: upfront cost in training them and ongoing cost in supporting them and in the portion of each sale that they retain. At the very least the separate channel needs to be more productive than it would be to simply sell through the existing channel. By productive, I mean delivering more margin dollars. Sales might be higher with the separate channel, but sales costs might be even higher still making it unattractive. That is one reason that typically when an acquisition is made, the sales force from the acquired company is folded into the sales force for the acquiring company (usually with some of the lower performers being surplus to requirements) rather than being ramped up aggressively as a separate channel.

The second issue is reach. The existing sales force sells to certain customers, and in fact to certain groups within those customers. It will be hard for an existing sales force to sell a new product if it has different customers or even completely different groups within those customers. Their "rolodex" (or CRM system) isn't any use. They are not already on the right aircraft, they are not already going to the right meetings. In this case, that militates for having a separate channel.

The third issue is conflict. So-called "channel conflict" occurs when a customer might be able to purchase the same product through more than one channel, specifically more than one type of channel, such as direct from the company or via some sort of reseller. This has impact on pricing in a way that might have downsides. For example, go up to Napa Valley and visit a winery. For sure, the winery will be very happy to sell you a few bottles of wine. Since they don't have any middlemen and have a huge amount of inventory (they don't just have the few bottles in the store, they have hundreds of barrels of the stuff in the back) then surely they will sell you the wine for less than anyone else. But, in general, they will sell you the wine for the highest price anywhere. If they sold it for less, they would make more money at the winery but they would risk having distributors and restaurants refuse to carry it. In EDA, if there is a product available through distribution and direct, then the direct channel cannot routinely undercut the distribution or the distributor will soon stop actively selling.

The fourth reason to have a separate channel is when the market demands, or the company decides, that it must transition its sales from one channel to another. Maybe they decide to move from direct sales to only doing telesales or only taking online orders. Or perhaps they decide that the day of a standalone product has gone, and they will only be able to sell integrated with a partner going forward. The channel must switch from however they sold before, to simply relying on the partner to sell their product and getting their share of those sales (and, presumably, enlarging their partners market in some way or else the partner wouldn't be interested). I've talked before about how in EDA OEMs only work when the product is actually a component, since otherwise the customer will always want a direct relationship with the real

seller. But if you do have a component, rather than a product, you must sell through an OEM type of license (as do companies like Verific or Concept Engineering).

Finally, disruption. If you have a product that is disruptive you have to have a separate channel. Disruptive, in the Innovator's Dilemma sense, means (usually) that it is sold at a low price point to people who are not served by existing products, and where the low price point product is expected to improve fast and gradually swallow most of the market. Think early PCs versus minicomputers or teeth-whitening strips versus dentist's providing whitening service. The existing channel is threatened by the disruptive technology, may not even be able to cover its channel cost (think of your dentist selling you teeth whitening strips you could just pick up in Longs) and will be unenthusiastic about selling it compared to more profitable lines. If you are going to be brave enough to try and kill your own baby, then you need separate organization for the baby and the killers. Sometimes, moreover, the disruption *is* the channel itself (Amazon and bn.com or, for a historical example, Sears starting to sell by catalog as well as department stores). This means a new channel by definition.

EDA has rarely had a separate sales force, since it is just too expensive. One that I mentioned above was Cadence's Alta Group. For a period that had its own sales team and was successfully growing revenue. But it was expensive and Cadence decided to fold it back into the main sales force. Sales declined and Cadence ended up "selling" that part of the business to CoWare (which you can regard as one way of going back to a separate channel).

You comp plan is showing

I talked recently about setting up separate channels and when it made sense to do it, and about some aspects of channel conflict. One area where separate channels are usually required is when a business is global. Most EDA products, even from quite small companies, have business in Japan, Taiwan, Korea and Europe as well as the US. Most of these cannot be serviced with a direct

sales organization until the company is pretty sizeable and maybe not even then. But customers don't always view the world the way your sales compensation structure does. It is really important not to let the way you structure and compensate your internal organisations, especially sales, show through and limit how customers can do business with you.

When I was at VLSI Technology, we wanted to standardize on a single Verilog simulator and we had decided that it would be ModelSim. So we wanted to negotiate a deal for using ModelSim throughout the company. At the time, Mentor had already acquired ModelSim but it was still sold partially through the old ModelSim channels, which were distributors and VARs (value-added-resellers). I don't think ModelSim ever had any direct sales force. We met with our Mentor account manager.

Mentor basically refused to do any sort of global deal because they felt unable to go around their distributors in each region; we would have to do a separate deal with each region. Also, licenses sold in one region would not be usable in other regions since the distributor/VARs provided first line support. The US alone was several different regions so this wasn't very attractive.

Part of the reason for doing a global deal was that we could get better pricing, we thought, since Mentor's costs would also be lower if we wrote one contract for a large amount, as opposed to negotiating lots of smaller contracts with each region. We also didn't want to have to worry about where a license was used, we wanted a certain amount of simulation capacity for a certain number of dollars. Internally we didn't even track where tools were used. There is always some issues about using licenses in regions other than the one where they were sold. Firstly, the salespeople get annoyed if someone in region A sells a lot of software that is largely used in region B, especially when the salespeople for region B starts to get lots of calls from "their" customer. Even if the customer promises that all support will go through region A, this usually doesn't stick, especially once different languages are involved. It is just not credible that all Japanese customers will be supported through, say, Dallas, whatever the software license says.

It can be a major problem when the internal scaffolding of the sales organization shows through to customers like that. “I can’t sell you that because I won’t get any commission” is not a very customer-focused response. You get the same problem, on a smaller scale, in many restaurants if you ask someone who is not your waiter for another glass of wine. The server won’t ignore you totally but they won’t bring the wine either, just tell your server if they remember.

Whenever possible, you want your channel to look as unified as possible to the customer, no matter what battles are going on internally. Like a swan, serene on top and paddling like hell underneath.

At the other extreme, my girlfriend works for a medical education company. It’s largely grown by acquisition but has the (to me insane) strategy of keeping each company’s sales force and branding intact. So any given hospital may have half-a-dozen people calling on it, selling them different products under different brand names, but from the same company. The financial inefficiency of doing this is huge, and as more and more of their business moves into the electronic space and is integrated into electronic medical record systems, more and more of their business will be through indirect channels in any case. But they don’t see this as either inevitable nor a good thing (since it is less profitable) and worry a lot about channels that conflict with their own salespeople. Some of their competitors have bitten the bullet, got rid of their direct sales force and only sell indirectly. Lower costs, one brand name, and no channel conflict. The straps of their compensation scheme aren't showing.

As for VLSI and ModelSim, we ended up doing a deal with another company, Cadence I think. It's not just a minor inconvenience to let your sales compensation drive the business. It can drive it away.

Board games

I talked earlier about changing the CEO in startups. The board in any company really has two main functions. One is to advise the CEO since the board often has complementary experience. For

example, older venture capital investors have probably seen before something very similar to any problem that may come up, or board members with industry experience may have a more "realistic" view on how taking a product to market is likely to turn out than the spreadsheets describing the company's business plan.

The second, and most important, job of the board is to decide when and whether to change the CEO. In one way of looking at things, this is really the only function of the board. The CEO can get advice from anywhere, not just the board. But only the board can decide that the company leadership needs to change. It is the rare CEO that falls on his own sword, and even then it is the board that decides who the new CEO is going to be.

Usually there is some controversy that brings a crisis to a head. The CEO wants to do one thing. There is some camp, perhaps in the company, or perhaps outside observers, or perhaps on the board itself, that thinks that something else should be done. The issues may be horribly complicated. But in the end the board has a binary choice. It can either support the CEO 100%, or it can change the CEO. It can't half-heartedly support the CEO ("go ahead, but we don't think you should do it"). It can't vote against the CEO on important issues ("let's vote down making that investment you proposed as essential for the future").

I was involved in one board level fight. I was about to be fired as a vice-president even though the board supported my view of what the company needed to do and told me that they wouldn't let the CEO fire me. But in the end, they only had those two choices: support the CEO, or fire the CEO. The third choice, don't fire the CEO but don't let him fire me, didn't actually exist. So I was gone. And the new CEO search started that day and the old CEO was gone within the year.

Boards don't always get things right, of course. I don't know all the details, but there is certainly one view of the Carly Fiorina to Mark Hurd transition at H-P that Carly was right, and Mark has managed to look good since all he had to do was manage with a light hand on the wheel as Carly's difficult decisions (in particular the Compaq merger) started to bear fruit. If she had been allowed to stay, she'd have got the glory in this view.

Almost certainly, Yahoo's board got things wrong with the Microsoft acquisition offer. Jerry Yang wanted (and did) refuse it. The board supported him. Their only other choice was to find a new CEO, which they eventually did.

When Apple's board fired Gil Amelio and brought Steve Jobs back, hindsight has shown that it was a brilliant decision. But in fact it was extraordinarily risky. There are very few second acts in business, where CEOs have left a company (and remember, an earlier Apple board had stripped Steve Jobs of all operational responsibility effectively driving him out of the company) and then returned to run them successfully later. Much more common is the situation at Dell or Starbucks, where the CEO returns when the company is struggling and the company continues to struggle.

Hiring and firing in startups

Startups have unique problems in human resources. For a start, they don't have human resource departments or even, in the earliest days, anyone to even do the mechanical stuff of making sure the right forms are filled out. You have to do that yourself.

There's some obvious stuff that is unlikely to trip anyone up: people need to have a legal right to work at the company, meaning be US citizens or permanent residents. In the earliest days you are not likely to want to have to go through the visa application process so that is probably the end of the list of people you'd want to bring on board. One exception might be someone who has an H-1 (or other appropriate) visa already; it is fairly straightforward to reassign it to a new company and doesn't run into any quota issues and only takes a few weeks.

One thing that is incredibly important is to make sure to create a standard confidentiality disclosure agreement and make sure that, without fail, every employee signs it. This binds the employees to keep company confidential information confidential (and survives their quitting), and also assigns to the company copyright and patent rights in the code (or whatever) they create. If an employee leaves to go to a competitor, that is not the moment to discover that the employee never signed his or her

employment agreement and that it is legally murky what rights they have to ideas they came up with on your watch.

But the most difficult challenge is building the right team. This is probably not a problem with the first handful of hires. They are likely either to be founders or else already known to the founders from "previous lives," working together in a similar company.

One small point to be aware of is if any of the founders was packaged out from a previous company (as part of a layoff, for example) and signed a release. Almost certainly the release will explicitly prohibit the ex-employee from recruiting people from the old company for a period of time. However, that doesn't mean you can't hire them; they have a right to work where they want (at least in California, ymmv). The best way to play safe is for such ex-employees not to interview the candidate. That way they can't be accused of "recruiting."

The first problem about hiring, especially if the founders are doing their first startup, is the deer-in-headlights phenomenon of not being able to make a decision about who to hire. Most of the time a candidate will never want to work for you more than right after the interview, having heard the rosy future, seen the prototype, met the team and everything. The quicker you can get them an offer, the more likely they are to accept. Firstly, they won't have had time to interview with anyone else equally attractive and secondly they won't have had time to start to get to the sour-grapes stage of rationalizing why you haven't given them an offer already. One advantage startups have over bigger companies is that they can make people an offer very fast. It can make a big difference: when I first came to the US I was promised an offer from Intel and H-P. But VLSI Technology gave me an offer at the end of the day I interviewed, so I never even found out what the others might have offered (Intel had a hiring freeze before they'd have been able to get me an offer, as it happened). Don't neutralize the fast offer advantage that startups have by being indecisive.

The second problem about hiring is hiring the wrong people. Actually, not so much hiring them. It goes without saying that some percentage of hires will turn out to be the wrong person however good your screening. The problem comes when they

start work. They turn out to be hypersmart, but think actually delivering working code is beneath them. They interview really well but turn out to be obnoxious to work with. They don't show up to work. They are really bright but have too much still to learn. Whatever. Keeping such people is one of the reason startups fail or progress grinds to a halt.

Firing people is an underrated skill that rarely gets prominence in books or courses on management. Even in large companies, by the time you fire someone, everyone around you is thinking, "what took you so long?" In a startup, you only have a small team. You can't afford to carry deadweight or, worse, people who drag down the team. It doesn't matter what the reason is, they have to go. The sooner the better. One thing to realize is that it is actually good for the employee. They are not going to make it in your company, and the sooner they find a job at which they can excel, the better. You don't do them any favors by keeping them on once you know that they have no future there.

It may be the first time that you've fired someone in your life, which means that it will be unpleasant and unfamiliar for you. Whatever you do, don't try and make that point to the employee concerned. No matter how uncomfortable you might feel, he or she is going to be way more uncomfortable. It doesn't get much easier with experience. It will always be more fun to give someone a bonus than to terminate them.

Make sure to have someone else with you when you terminate someone. In a big company that will be someone from HR, in a small company you just want someone to be a witness in case of a lawsuit ("he told me he fired me because I was a woman"). In California you must give them a check for all pay due there and then (actually I think you have until the end of the day) so make sure you have it ready. Normally you will want the employee to sign a release saying that they won't sue you and so on. If you give the employee severance (a good idea to give at least a little so the other employees feel that they work for a company that is fair) then the severance is actually legally structured as payment for that release. So don't give them the check until they sign (and if they are over 40, there is a waiting period during which they

have the right to rescind their signature, so don't give them the check until that expires).

Application Engineers

Application engineers are the unsung heroes of EDA. They have to blend the technical skills of designers with the interpersonal skills of salespeople. Most AEs start out as design engineers (or software engineers for the embedded market). But not all design engineers make it as AEs, partially because, as I'm sure you've noticed, not all design engineers have good interpersonal skills! There's also another problem, memorably described to me years ago by Devadas Varma: "they've only been in the restaurant before; now they're in the kitchen they're not so keen on what it takes to prepare the food." Being an AE means cutting more corners than being a design engineer, and some people just don't have that temperament. An AE usually has to produce a 95% solution quickly; a design engineer has to take whatever time it takes to produce a 100% solution.

AEs have a lot of options in their career path. As they become more senior and more experienced they have four main routes that they can take. They can remain as application engineers and become whatever the black-belt AEs are called in that company, be the guy who has to get on a plane and fly to Seoul to save a multi-million dollar renewal. They can become AE managers, and run a region or a functional group of AEs. They can move into product marketing, which is always short of people who actually know the product. Or they can move into sales and stop resenting the fact that when the deal closes, for which they feel they did all the work, the salesperson makes more than they do (and usually discover sales is harder than they expected).

In a startup, in particular, the first few AEs hired can be the difference between success and failure. The first release of a product never works properly, never quite matches what the market need is and is simply immature. The AE has to keep the customer happy by substituting their own expertise for the deficiencies of the tool while at the same time conveying back to engineering the improvements that are required. Most startups are

attacking some sort of walled city, in the sense that there is an incumbent tool/methodology that is already in use, and the startup has to prove that they are better. In fact, not just better, compellingly better. The initial value proposition for most startups, when you look from the 10,000 foot level, is that it is riskier to stick with the existing methodology rather than trust the startup and try the new technology. Getting the customer decision-maker to that point is a mixture of technology (it has to work well enough) and trust in the AE (whatever happens, this guy is going to be there for me). Both factors have to be there to close those so-important initial orders because no matter how good the technology looks, the customer knows that the tool is not mature and might fail at any moment.

It's been interesting looking at the downsizing of GM and Chrysler's dealer network. It seems that part of the reason that car companies sell through independent dealers is that in the early days, nobody would buy a car from halfway across the country without a local guy in-town they trusted (and the situation got locked in place because those local guys became the richest people in town and got the states to pass laws that they could never be designed out; it almost every state it is illegal for GM to sell you a car directly). But that trust issue is just like the AE issue. Customers wouldn't buy a car (tool) from a startup without a dealer (AE) too. It didn't matter how good Ford's car appeared to be in the showroom; in 1930, nobody trusted it not to break frequently (a good assumption) and they needed to trust that their investment was going to continue to be good.

AEs are really hard to find for a startup. Good AEs are pretty highly compensated, and so it is hard to match their salary, so it takes a lot of stock to makeup the difference. I did some consulting for a semiconductor equipment company once and they had an EDA product but they failed to hire a good AE since their own AEs were paid a lot less than a black-belt EDA AE and their salary policies were too inflexible. Good AEs are like gold and if you don't have them you don't get any gold.

The career path train doesn't stop every day

When I lived in France there was a program called "La piste de Xapatan" in which contestants had to negotiate a series of challenges before whizzing down a zipline and running up a hill to catch a train. But "le train de Xapatan part toujours à l'heure" (the train from Xapatan always leaves on time) which meant that usually the contestant would arrive just after the train started to move and either just catch it or just miss it by seconds.

The career path train, however, isn't like that. It doesn't stop at the station every day and when it does stop you have to decide whether or not to get on. When you want a change of job for some reason, there doesn't seem to be a train. And when you aren't really looking for anything the train shows up and you have the opportunity to board while it is in the station. But it won't be in the station again tomorrow; you have to decide right now.

It's especially hard to decide if the opportunity takes you out of the comfort zone of what you have been used to in your career so far, or if it involves relocating. Two times the career path train stopped for me were "would you like to go to France and open up an R&D center for us?" and "would you like to return to California and run all of R&D?" There's always some sort of tradeoff in a promotion, not just more money for doing what you are already doing.

Big companies usually have dual career ladders for engineers, with a management track and a technical track. However, it's a bit of an illusion since only the strongest technical contributors really have a sensible option of staying completely technical and continuing to advance. I think dual career ladders are mostly important because they institutionalize the idea that a senior technical person might be paid more than their manager, sometimes a lot more. In more hierarchical eras that didn't happen.

But the fact that only the strongest engineers can keep advancing as engineers means that at some point most of them will have to

transition into management or into some other role that makes use of their engineering background along with other skills to do something that is more highly leveraged than simply doing individual contributor engineering. It's a big step that will require you to learn stuff you've not had to do before.

But people are often not keen to take that critical step out of their comfort zone. I've sometimes been surprised at how reluctant people are to step up and take on new challenges when I've offered them what is essentially a significant promotion. Funnily, one of the biggest issues is always the salary review process. Everybody wants to avoid the work and responsibility of reviewing people that work for them, and sometimes this hate is so visceral that people refuse to have anyone report to them. I'll be the first to admit that reviewing people's performance is not the most enjoyable part of management, but it is just a few days per year for the formal part. Actually the trick is to make sure that nothing in a review is a surprise because you've already been communicating feedback both good and bad throughout the year. It is bad management on your part of you surprise someone with a really bad review when they didn't think anything was wrong.

I'm sure that there are anecdotes to the contrary, but in general I think most people are best to take opportunities when they are offered. This is especially true in startups and small companies. They tend to grow downwards, in the sense that people there early get to bring in the lower levels. There are more opportunities for responsibility early (a plus) but less formal training (a negative) unlike large companies that often have in-house management training.

So when the career path train stops, you'd better have a very good reason not to get on.

How do you get a CEO job?

How do you get to be CEO? I've done it a couple of times now and I'd be happy to do it again. I assume we are talking about a startup of some kind rather than a large company. But if a private equity fund takes a semiconductor private they face many of the same issues in their choice of CEO.

The reality is that the number one criterion that anyone is going to look for if they have a free choice is that you have been CEO before. Better still, is that when you were CEO before, you had a good exit, selling the company you ran for a good price or at least generally having left the company in better shape than you found it.

So if you've been CEO before and not made too much of a mess of it, you can get to be CEO again. So how do you get your first gig? Well, you are not going to get headhunted to run a class-A company. Before anyone is going to trust you with a class-A company you have to have taken a mess and made something of it. Everyone's first CEO job is to take a company with little hope and try and turn it around. This isn't as bad as it sounds since expectations are low and so the standards that the board will judge you by are not as demanding as if you took a company with great potential. This type of CEO job typically comes along because you are in the right place at the right time. My two stints as CEO came about this way. At Compass, I was "on the bench" in the finance division doing M&A when VLSI decided Compass needed new leadership. I was someone who knew the company and could take over instantly without needing to do a CEO search. At Envis, I was VP marketing (actually only working part-time as a consultant) when I was asked to take over.

If the board has time to do a proper search for a CEO, probably the most important criterion is that you are "fundable." By that they mean that investors are going to view you as CEO as an asset not a liability. The best proof of fundability is that you have raised money successfully in a previous CEO job, but a substitute is the right combination of business savviness and track record. You've probably heard that VCs invest first in the market, then the team and only then in the technology. So the CEO is really important. The perfect CEO can raise money simply on his name (imagine if Marc Andreeson decided to start another company). More mortal CEOs are regarded as an asset to the company, a CEO who isn't going to need to get swapped out later. Lower down are people who are at least OK for the current stage of the company, with a question mark over whether they will make it long term. That sounds bad, but in fact 75% of founding CEOs don't make it so it's not as disparaging as it sounds.

Of course, the guaranteed way to be CEO is to found your own company. You get to choose the CEO and can pick yourself. But whether you make a good CEO and whether you can get funded are not questions that go away. You just have to answer them yourself.

Managing your boss

There are shelves of management books about how to manage people that work for you. I don't know of any management books about another very important skill: how to manage your boss. Or, if you are CEO, how to manage your board.

Instead of thinking of your boss as someone who tells you what to do (they'll obviously do some of that) think of them as someone that you are going to tell what you are doing and how they can help you accomplish your goals.

This is not about sucking up to your boss and being a yes-man. You boss is probably not so vain and stupid as to regard that as A-team behavior. You can't always get what you want using your own personal charisma, sometimes you actually need your boss to do some tackling for you to leave the field clear.

One rule I've always tried to follow is not to produce big surprises. Of course things can go wrong and, say, schedules can slip. But they don't go from being on time to being 6 months late overnight, without the slightest earlier hint of trouble. It is better to produce a small surprise and warn your boss that things might be getting off track (and have a reputation for being honest) than to maintain the image of perfection until the disaster can no longer be hidden. Just like the salesman's mantra of "underpromise and overdeliver" your boss is a sort of customer of yours and should be treated the same way.

Lawyers are advised never to ask a witness a question that they don't already know the answer to. Getting decisions that cut across multiple parts of a company can be a bit like that too. Never ask for a decision when you don't already know exactly what everyone on the decision making panel thinks. Ideally they all buy into your decision, but in the middle of a meeting is not

the time to find out who is on your side and who isn't. Your boss can be invaluable in helping to get his peers on-board and finding out what they think in a way that you, being more junior, perhaps cannot.

In some ways this sounds like office politics, but actually I'm talking getting the company to make the correct decision. Often someone junior is the best-placed person to know the right technical solution, but they are not senior enough to drive the implementation if it requires other groups to co-operate. That's when managing your boss comes into the picture.

If you are CEO you have some of the same issues managing your board. But your board is not one person and they all have different capabilities that you can take advantage of. But, just as in the decision committee scenario above, if you need a decision from the board make sure that everyone is bought into it already, or at least have some of the board ready to counterbalance any skeptics.

Integration and differentiation

EDA acquisitions are very tricky to manage in most cases. This is because most acquisitions are acquiring two things: a business and a technology.

In the long run the technology is usually the most important aspect of the acquisition but the business is important for two separate reasons. Firstly, the revenue associated with the standalone business, ramped up by some factor to account for the greater reach of the acquiring company's sales channel, is the way that the purchase price is usually justified. It is too hard to value technology except as a business. That's why the venture capital euphemism for selling a company for cents on the dollar is "technology sale." But more importantly, the business is the validation of the technology. Nobody can tell whether a startup's technology is any good except by looking to see if anyone is buying it.

However, once the acquisition is done there is an immediate conflict. There is a running business to be kept going. After all

that was the justification for the purchase price. It took the whole company to do that before acquisition, so presumably it will take the whole company afterwards. In the short term, the differentiation of the technology rests on its continuing to sell well. But the real reason for the acquisition is often to acquire the base technology and incorporate it into the rest of the product line. The only people who know the technology well enough to do this are the acquired company's engineering organization. Suddenly they are double booked, developing the product to the plans that underpinned the forward bookings forecast, and working with the acquiring company's engineers to do the integration.

An example. When I was at Cadence we acquired Cadmos for their signal integrity product SeismIC. plus other stuff in development. Googling back at the press release, I see that a person whose name sounds strangely familiar said:

"Adding CadMOS signal integrity analysis engines to established Cadence analog and digital design solutions provides us with the best correct-by-design timing and signal integrity closure capabilities in the industry," said Paul McLellan, corporate vice-president of custom integrated circuit (IC) products at Cadence.

Except, of course, to realize that vision required the Cadmos engineering team to work full-time on integration. Meanwhile, there is a business going full blast selling the SeismIC standalone. I believe there was also an earnout (part of the acquisition price depends on how much was sold) based on the standalone business only. A difficult balancing act for the engineering managers and myself.

We had similar issues when Cadence acquired Ambit. We needed to integrate the Ambit timing engine (and later the underlying synthesis technology itself) into Cadence's whole digital product line at the same time as we were trying to give Synopsys a run for their money in the standalone synthesis business. Both of those goals were really important strategically but there was only one set of engineers.

Balancing these two conflicting requirements is probably the hardest aspect to manage of a typical EDA acquisition. It is really

important, not just for financial reasons, to maintain the leadership position of the technology in the marketplace. At the same time, integrate that leadership technology so that it is available under-the-hood in other parts of the product line which, in the end, is probably how it will mostly get into customer's hands. Preserve the differentiation while doing the integration.

Big company guys don’t do small

Big company guys think that they can run startups because they’ve run small divisions of big companies. So that must be the same, right?

Actually the two things are very different and not many people seem to be good at making the transition once they have got used to how a big company works, with their assistants, and finance organization, and HR department and all the rest.

When I was at VLSI and the fab was not running effectively, the company would hire a VP from TI or Motorola (where the CEO had previously worked and so knew good people he’d worked with before). These guys were used to running a fab that was running smoothly, with a large organization around them. They were not used to sorting out a dysfunctional fab with very few people to support them. When they didn't work out, they were doubly expensive because they needed big severance packages to get rid of them.

When you become CEO of a startup, you have do everything yourself. Especially if the startup is attempting to run very lean with minimal cash burn, and conserve most of that cash for engineering. You want to put together a business plan? Fire up Excel. There’s at most a part-time accountant in the finance department and you can’t delegate it to them. Even if you have a “CFO for a day” part-time senior finance consultant, they don’t understand the business intimately like you should because that’s bound up with strategy which is not just something financial. They can help review the plan but they can’t do it for you.

If you’ve not got a very good engineering manager then you can’t rely on the current schedules. And you don’t have enough money

to do what you would in a big company and hire a good engineering manager, or even a really good product management specialist. You have to do that yourself too. In a typical startup, as CEO, you will probably be the only person who isn't writing code or designing chips.

Another problem with big companies is that people don't really know how successful their business really is, since it is often very bound up in company-wide financial measures that are not closely enough tied to reality. So it is easy to look good when you aren't, or looks undeservingly bad. If you are in a big EDA company, nobody knows how to really allocate revenue from big volume purchases to product lines. If you are in a semiconductor company, the cost model is rarely as accurate as necessary, and fab variances (because the fab is overloaded, or underloaded, or not yielding as expected) distort it again.

If your company has a few hugely profitable product lines (think Intel or Synopsys) then the smaller product lines may look good or not depending on how the overhead of the company is handled, and whether the profitable lines eat a lot of overhead leading to everyone else looking good (margin bleed-through), or the opposite, leading to everyone else looking worse than reality. It is too expensive to do full activity-based costing (ABC) and so overhead is often misleading. If cost of sales is a fixed percentage of revenue, that assumes all products and all order sizes are equally easy to sell, which is clearly not true. But this may make some product lines look great (hire that manager) and others look poor (and he looked so promising) even though it purely an artifact of the underlying management accounting.

Although it is possible to make the transition from a big company to a startup, but both EDA and fabless semiconductor are littered with people who failed to do that. They were very successful at running a division of a big company, but were unable to translate that skill into success at either founding or coming into a startup and getting it to run well.

Being CEO

I talked earlier about how you get to be CEO (basically, luck the first time; track record after that). But what does being a CEO entail?

I think all senior management jobs consist of two separate dimensions that have two separate skill sets. I call these management and leadership. Some people are good at both, some are good at only one.

Management is the basic operational management of the company. Presumably you already know how to do this, at least in your own domain (engineering, marketing, sales, etc) or you probably wouldn't have been promoted. When you get more senior you have a new challenge: you have to manage people not from your own domain. If you are an engineer, it's like salespeople are from another planet and you don't understand what makes them tick. If you are a salesperson you may think the same about engineering. If the company is medium sized things are not so bad since you'll have a sales manager and an engineering manager to insulate you. But if the company is small then you'll have to manage the aliens directly. My recommendation is to get some advice. If you've never set up a sales commission plan before, don't assume that because you are a smart engineer who knows Excel that you can just wing it. If you don't know a friendly VP sales who can give you free advice, find a consultant and pay them. It's a lot cheaper than making major mistakes.

As CEO you may have only an accountant (or maybe nobody) to support you in finance. I think it makes sense to get a "CFO for a day" consultant to help you unless you are very comfortable with all the finance issues and already have a good feel for how to put together a business plan, how to turn a sales plan into a cash-flow forecast and so on. If your eyes glaze over when you read my blog postings on finance, you need someone to help you. Whatever you do in finance, don't treat it as a problem that will go away if you ignore it. You'll need to get a financial audit done at some point, sooner than you expect, and cutting corners will then come to light.

If you are not an engineer by background, you can't manage engineering. That's not to say that you aren't capable of managing engineering but just like salespeople won't respect you unless you've carried a bag, engineering people want to be managed by someone who understands technology and development and knows what it takes to get a product out. If you don't have an engineering manager you'll at least need to trust one of the senior engineers to be feeding you the unvarnished truth.

The second leg of being a CEO or a senior manager is leadership. The most important aspect of this is to get everyone in the company committed to moving the company in the same direction. Unless you make truly stupid decisions, it is more important that everyone is aligned than that the decision is ideal. As General Patton famously said, "A good plan executed violently now is better than the perfect plan next week." In business, "violently" is the wrong adverb but the sentiment is the same.

Having said that, it is also important that the overall strategy of the company is good and represents the best that the management team can come up with. It is also important to be flexible. If something isn't working then you'll need to try something else and preferably while you still have enough money in the bank to find out whether that new approach is good. Remember, most successful startups end up doing something somewhat or completely different from what they set out to do initially.

A general rule about management and especially being a CEO: if something good happens in the company, everyone will tell you about it. If something bad happens, nobody will tell you. Despite the proverb that "bad news travels fast," inside a company bad news travels really slowly so you need to make a special effort to discover it. In the early stages it is good to have someone in engineering who is a personal friend who will not hide bad news. Later on, you need someone in sales like that who'll tell you what is really happening when the company tries to sell the product. You can't sit in your CEO office and believe everything that you are told. You have to get out and dig.

Getting out of EDA

Over the last year I've had lots of meetings with people who used to work in EDA and have lost their jobs, or, in some cases, still have a job but want to make a longer term change. The subject that comes up all the time is "How do I get out of EDA?"

This is not unreasonable. EDA has shrunk its employment over the last couple of years and it is unlikely to come back again to its previous level. So some people will need to find jobs in new industries.

If you are an engineer in EDA then you know how to do very technical programming. You could certainly do other forms of technical programming. But the sweet spot in the job market is in internet companies and there is a lot of specialized stuff there that you probably don't have experience of. If someone wants to get an internet startup going quickly then you want people who already know Ruby on Rails, mySQL or the iPhone developer kit or whatever. Not someone really smart who could probably learn that stuff eventually. Personally, I think this is silly. A smart programmer can suck up a language in no time and will run rings around someone less good even with a lot of domain experience. Good programmers are not 30% better than average ones, they are 10 times better. But even if you get hired, you don't get paid for all that deep knowledge of, say, placement that you've spent years acquiring.

If you are in marketing or management it is even more difficult. At one level you have experience of running business to business (b2b) marketing for a software company. But you have years of understanding of IC design and none of relational databases or whatever, which makes it hard to make that transition. Furthermore, most internet companies are business to consumer (b2c) or internet-based business to business which is very similar.

I interviewed over a year ago with a b2c company and I was amazed that they seemed interested in me. It was bit like the Groucho Marx joke about not wanting to be a member of any club that would have him. The fact that they seemed interested in hiring me for a job that I was so manifestly unqualified for

(although it would have been interesting to learn) made me doubt their competence.

If you are in some other domains you get stuck in those domains. I have a friend who is in finance. That allows you to work in all sorts of different companies, but always in finance where all companies look very similar. She wants to get out of finance, which is a similar problem. She's smart enough to do all sorts of jobs but the only jobs that will pay anything close to what she is used to are ones that value all that financial experience.

It's a tough transition to make. Your experience is what makes you valuable in EDA (or finance or whatever). If you go somewhere where that is not valued it is very hard to make anything close to what you made in EDA. After all, EDA pays pretty well so long as you have a job.

Hunters and farmers: EDA salesforces

I wrote recently about mergers in the EDA space, mainly from the point of view of engineering which tends to end up being double booked keeping the existing standalone business going while at the same time integrating the technology into the acquiring companies product line.

The business side of the acquired company has a different set of dynamics. They only have to cope with running the existing business since any integration won't be available for sale for probably a year after the acquisition. The basic strategy is to take the existing product that has presumably been selling well, and make it sell even better by pumping it through the much larger salesforce of the acquiring company.

The big question is what to do about the salesforce of the acquired company. A big problem is that there are really two types of salespeople that I like to call hunters and farmers. A startup salesforce is all hunters. A big company salesforce is all farmers. Some individuals are able to make the transition and

play both roles, but generally salespeople are really only comfortable operating as either a hunter or a farmer.

Hunters operate largely as individuals finding just the right project that can make use of the startup's technology. Think of a salesperson trying to find the right group in Qualcomm or the right small fabless semiconductor company. Farmers operate usually in teams to maximize the revenue that can be got out of existing relationships with the biggest customers. Think of Synopsys running its relationship with ST Microelectronics.

Given that most of the hunters are not going to become good farmers, or are not going to want to, then most of the acquired company's salesforce will typically not last all that long in the acquired company. But they can't all go immediately since they are the only resource in the world that knows how to sell the existing product, that has a funnel of future business already in development and probably have deals in flight on the point of closing. One typical way to handle things is to keep some or all of the existing salesforce from the acquired company, and create an overlay salesforce inside the acquired company specifically to focus on helping get the product into the big deals as they close.

The challenge is always that the existing salesforce doesn't really want a new product to introduce into deals that are already in negotiation. They have probably already been working on the deal for six months, and they don't want to do anything to disrupt its closing. Adding a new product, even though it might make the deal larger, also adds one more thing that might delay the deal closing. The new, unknown or poorly known product, might not work as advertised. As I've discussed before, big company salesforces are very poor at selling product where the customer isn't clamoring for it.

So the typical scenario goes like this: the small acquired company salesforce is sprinkled into the big acquiring company salesforce for a quarter or two to make sure that initial sales happen and so that the farmers learn how to sell the product. After a quarter or two, the hunters will either drift away because they find a new startup opportunity, make the transition to being farmers in their own right (they may have been at some point in their career

anyway), or else they fail to make the transition and end up being laid off.

Running a salesforce

If you get senior enough in any company then you'll eventually have salespeople reporting to you. Of course if you are a salesperson yourself this won't cause you too much problem; instead, you'll have problems when an engineering organization reports to you and appears to be populated with people from another planet.

Managing a salesforce when you've not been a salesperson (or "carried a bag" as it is usually described) is hard when you first do it. This is because salespeople typically have really good interpersonal skills and are really good negotiators. You want them to be like that so that they can use those skills with customers. But when it comes to managing them, they'll use those skills on you.

When I first had to manage a salesforce (and, to make things more complicated, this was a European salesforce with French, German, English and Italians) I was given a good piece of advice by my then-boss. "To do a good job of running sales you have to pretend to be more stupid than you are."

Sales is a very measurable part of the business because and order either comes in or doesn't come in. Most other parts of a business are much less measurable and so harder to hold accountable. But if you start to agree along with the salesperson why an order really slipped because engineering missed a deadline, then you start to make them less accountable. They are accountable for their number, and at some level which business they choose to pursue, and how it interacts with other parts of the company, is also part of their job. So you just have to be stupid and hold them to their number. If an order doesn't come for some reason, they still own their number and the right question is not to do an in-depth analysis with them about why the order didn't come (although you might want to do that offline), but to ask them what business they will bring in to compensate.

Creating a sales forecast is another tricky skill, again because an order either comes or doesn't come. One way of doing it is to take all the orders in the pipe, along with a percentage chance they'll close. Multiply each order by the percentage and add them all up. I'm not a big believer in this at all since the chance of a 10% order closing in the current period is probably zero and it's easy to fool yourself. Yes, the occasional blue bird order comes out of nowhere, sometimes so much out of nowhere it wasn't even on the list. I've never run a huge salesforce with hundreds of salespeople; the law of averages might start to work a bit better then, but typically a forecast is actually build up with the judgment of the various sales managers up the hierarchy.

Another rule I've learned the hard way is that an order than slips from one quarter to the next is almost never incremental. You'd think that if the forecast for this quarter is $500K, and the forecast for next quarter is $500K, then if a $100K order slips that you have a bad $400K quarter now but you've got a good $600K quarter coming up. No, it'll be $500K. Somehow the effort to finally close the slipped order comes out of the effort available to close other orders and you are wise not to count on a sudden blip in sales productivity.

Salespeople are a pain to hire because you have to negotiate with them and they are at least as good, if not better, negotiators than you are. It's even worse in Europe where, if you don't simply lay down the law, you can spend days negotiating about options for company cars ("I insist on the 8-CD changer"). At least in the US most of the negotiation is over salary and stock, which are reasonable things to spend some time on.

Another thing I've discovered is that salespeople really only respect sales managers who have themselves been salespeople in the field. Not marketing people who have become sales managers, not business development people who've become salespeople. It's probably partly camaraderie but sales seems to be something that you have to have done to really understand. You want your sales manager to be respected by the salespeople because you want them to bring him into difficult sales situations to help close them, and they won't if they don't trust and respect him.

How long should you stay in a job?

How long should you stay in a job? The answer will depend a bit on your personality. But I think a job is interesting so long as you're learning a lot and that seems to mean that you should stay in a job about three years. The first year you don't know how to do the job and your are learning a lot, the second year you are getting the hang of it and by the third year you have become good at the job. But being good at the job typically means that you don't have much more to learn from the job by continuing to do it. It's time to move on.

When I say it's time to move on I don't mean that you need to move company, although that is certainly one option. If you move to work on a new product you'll be learning stuff again. If you relocate to Japan you'll be learning stuff again. If you move from application engineering to product marketing you'll be learning again.

In particular, if you get promoted your job will change and you'll be learning stuff again. This is especially acute the first time you are promoted into management. Typically you are the best engineer or salesperson or whatever on the team and so you get promoted. Now you have to learn about management, a subject that previously you may not have taken much interest in. It is an especially difficult transition since your comfort zone is not to do management at all, just do everyone's jobs for them (after all, you were the best on the team so you are better than they are). It is a hard lesson to learn that as a manager your output is not what you do personally, it is the output of your group. It is not a positive that you did a lot of the work yourself, that means you are not doing a good job of nurturing the people in your group, not training them to be as good as you are.

People will often move on to another company anyway if they are bored since there might not be an appropriate position to move into, or a promotion to be had. This is especially true of new graduates who get fed up with some aspects of the company bureaucracy or culture and move to a new company to escape. However, the new company is typically the same (although different in details). It's just the nature of companies that they

don’t always do just what you think they ought to. The result of this phenomenon is that I think the best value people you can possibly hire are people who have already worked for at least one company and have 3-5 years experience. At that point they are enormously more productive than a brand new graduate, not about to leave because of company bureaucracy, and although they are paid more they are not paid a correct premium. The new graduates are probably overpaid and the 3-5 year people underpaid.

I know mostly about engineering and a good engineer is not 30% better than a poor one, they are ten times more productive. So 3-5 year guy is not 50% better than a new graduate, which may reflect the pay differential, they may be 5 times better.

Spending money effectively

People die because they run out of oxygen. It doesn’t matter what the reason is—trauma, cancer, heart attack—lack of oxygen is what finally kills us. In the same way, startups die because they run out of cash. It doesn’t matter what the reason is—engineering never finished the product, the customers wouldn’t buy it, it wasn’t possible to raise another round—running out of cash is what finally kills us.

So obviously cash is so important in a startup that it should never be spent? Well, not so fast. I’ve seen some really silly decisions about how to save money in startups over the years.

Most of the cash being burned in a software startup goes on engineers’ salaries. Consequently it makes sense to do everything to make their work environment as productive as possible. Do not force them to use old computers because they are already around. Computers are pretty cheap these days, a few days of an engineer’s salary will buy you something really high end. Do not equip the servers with so little disk space that they have to delete old data that will eventually turn out to be useful. Terabyte disk drives are under a hundred dollars. And don't forget to make it easy for your engineers to work from home, by having good VPN and paying for them to have good internet connections. You pay

your engineers more in an hour than their internet connection costs for a month.

One thing we discovered at Envis was that companies that provide PCs for gamers deliver the most bang for the buck. They overclock the designs, add special fast memory, have custom motherboards and so on. For a lot less than Dell will sell you a machine, you can get one that is half as fast again, with lots of cores. And a bonus, they look really cool.

Do not hire a consultant and then, for egalitarian reasons, give them a day on which it is their job to clean the kitchen at $150/hour. In fact, don't make your engineers clean the kitchen anyway. That's pretty pricy labor. And don't annoy your engineers by charging for coffee or soda.

In a semiconductor company, there is an additional significant cost, namely design tools. Sometimes this is a cost in a software company too since they need tools for quality assurance and integration purposes. This is a hard balance to get right since too few tools again means that what looks like saving money on tools is really burning extra money on engineers' salaries. Too many tools obviously wastes money more directly. When I was at Cadence we had a venture investment program where we would provide almost unlimited tools to startups for a mixture of cash and an equity position. We'd discovered that most startups underinvested in tools because they were so expensive but that this jeopardized their success.

Benefits, especially medical, are another area where startups can be penny wise and pound foolish. The most cost effective way to handle medical benefits, given that usually everyone is young and fit, is a combination of catastrophic coverage and a health savings account (HSA). In fact this is probably the best way to handle medical period, but that's a political hot potato right now. It is what Whole Foods does and what John Mackey, the CEO, has recently got into trouble with the left for recommending as better than what congress is attempting to put together.

Bottom line: remember Gordon Bell's line that cash is more important than your mother. But remember that engineers'

salaries are your biggest investment, and it is foolish not to do everything to make that investment as effective as possible.

Interview questions

A friend of mine is interviewing for a marketing position at an EDA startup. I'd better leave everything anonymous to protect the innocent. He (or maybe it was she) asked me what good questions to ask would be.

There are two reasons for asking questions in an interview, when you are the candidate. One is that the type of questions you ask reveal that you are already thinking about the important issues affecting the company. And the other is that you genuinely want to know. In most cases, the questions serve both ends. In fact most questions you ask should help you decide if the company is going to be successful and whether you have the right skillset to improve those chances.

When you interview for a position at a startup, it is important to realize that you are interviewing the company as much as they are interviewing you. The point of working for a startup is that the stock they give you will be valuable (otherwise go do something else) and they need to convince you of that. When you interview at a big successful company it is much more of a case of them interviewing you. After all, if you've done your homework, you should know what makes them successful. Most of that information is in the public domain.

The most important question I like to ask is why the senior people in the company believe it will be successful. Since they work there, presumably they do but sometimes that have a hard time articulating why. The answer needs to be more than just having good people or good technology. The market that they sell into needs to be large enough and homogenous enough for their (or any) product strategy to have the possibility of being successful.

Another thing I like to ask are: what is the one reason people buy your product? Of course if they don't have a good answer then there is all the more upside from doing a great job at marketing

(if you are interviewing for a marketing position). But typically, if most of the company is engineers, they'll have too many answers to this question rather than too few. Avoid the fine art and bicycles problem. City Slickers marketing is finding out the "one thing" and becoming focused on delivering that. If customers are all buying for different reasons, it is not possible to build a repeatable sales process.

A third question is to ask, which is good in non-startups too, is "If I got the job and was starting tomorrow morning at 9am, what would be the most important things to get working on?" They may not be the most important strategic things long-term, but if there hasn't been any marketing before there is usually a backlog of urgent stuff: the customer presentation is hopeless, the website hasn't been updated in ages, the company logo sucks, engineering needs a decision about which standard to support, or whatever.

Acquistions: cull the managers

When a company acquires another one, not just in EDA, there is often an internal group already doing something similar. For example, Intuit has just acquired mint.com and they already have a product, Quicken Online that competes in pretty much the same space. So how to merge the companies and the products?

Be ruthless and cull all the director-level management of the existing product (Quicken Online in this case). Put the managers of the acquired product in charge.

This is one thing that I learned at Cadence (you might have noticed that Cadence has done a fair number of acquisitions over the years, to say the least). The first thing to do is to lay off all the managers responsible for the internal competing product. They will inevitably try and sabotage the acquisition in more or less devious ways, worry too much about users of the existing product and so on. The junior worker-bee programmers or designers can be reassigned; they are much less emotionally invested in the failed internal product and have the knowledge to merge any parts of the old product that make sense.

In the Quicken case they seem to be doing something different, based on what they have said anyway. The correct thing to do, in my opinion, is to put the mint.com guys in charge of everything. Not just their own product but also the Quicken Online product. And the managers of Quicken Online need to go. They probably weren't in favor of the acquisition and will subtly try and show that it was a mistake and try and ensure as much as possible of their own work survives going forward. But it is the mint.com product where as much as possible must survive going forward, and the best way to ensure that is to put those guys in charge.

Steve Jobs did just this when he returned to Apple along with the operating system from Next (internally Mac code is still littered with classes that start NS for NextStep). He put the Next software managers in charge and pushed out the managers who had been responsible for the failed strategy that Apple had been pursuing. The Next managers could implement their strategy much more easily if they didn't have another set of managers arguing with them about every decision.

Everybody knows that the big time sink in mergers is where products overlap. But the best way to handle this is to make sure that the managers of the successful, acquired, product are in charge of those decisions and not the managers of the failed product. This doesn't make the problem go away completely, after all the customers of the existing product cannot typically simply be upgraded painlessly to the new product, but at least it means that the winning product will be the acquired one, which is essentially the decision that senior management had already determined is what they wanted to have happen when they decided to do the acquisition.

Not all mergers are like this, of course. Sometimes the new product line is completely complementary with no overlap. But often, under the hood, there is more overlap than is obvious. When Cadence acquired Ambit, they were already ahead of the curve because their internal synthesis product, Synergy, was doing so badly that they had killed it off six months before they acquired us. But one reason for acquiring Ambit was for its timing engine, which seemed to be the best in existence at that time, but the existing timing team at Cadence still controlled

timing strategy. It took months to arrive at the foregone conclusion that the Ambit timing engine should "win" and become the Cadence timing engine, a decision that would have taken 5 minutes if Ambit's timing team had been put in charge on day 1.

It is very difficult to keep innovation going after an acquisition, especially if it is done at a high price so that many individuals have made significant money and are really hanging around largely to vest the rest of their stock. Keeping a competing team around, and one that already is better connected politically, almost guarantees that innovation will stop and that the acquisition will be much less successful than it could have been.

Chapter 3: Marketing

City Slickers Marketing

I have done a fair number of consulting projects for EDA startups and a lot of them start out with what I like to call "City Slickers marketing", named after the movie City Slickers. For those of you who have not seen it, there is an old cowboy, Curly (played by Jack Palance) and a young advertising account manager Mitch (played by Billy Crystal). The marketing is named for the following conversation:

> Curly: Do you know what the secret of life is? [holds up one finger] This.
> Mitch: Your finger?
> Curly: One thing. Just one thing. You stick to that and the rest don't mean shit.
> Mitch: But, what is the "one thing?"
> Curly: [smiles] That's what *you* have to find out.

When I arrive at startups where the CEO is the key technologist, or even just an engineer by background, I tend to have a conversation like this:

"What's the one reason people should buy your product?" I ask.

"One reason, I can give you twenty," the CEO replies.

Technical people (and I am one, so this is a lesson I had to learn the hard way too) tend to overvalue technical features in a product and assume that if the technology is good then the product will sell itself. And if one feature is a good reason to buy, then lots of features are even more of a reason.

But the world doesn't work that way.

A colleague recently reminded me about a store in San Jose that sold "Fine art and bicycles". Presumably a fine art dealer who also was a keen cyclist. This is an extreme example of how multiple features are not necessarily additive and how you have

to take account of the way the customers look at the world. There simply isn't a "fine art and bicycle" market that you can be the leader of. In fact, even if you are the best art shop in the area, selling bicycles too isn't a plus, it detracts from your message.

A lot of early marketing in a startup is working out what the single compelling reason is for a customer to engage with you. And it has to be just one (or maybe a couple if you can segment the market a bit). The early stage of engaging with customers is sometimes referred to as throwing mud against the wall and seeing what sticks. You can't find the single compelling reason as an intellectual exercise, you have to get out and engage with customers and work out where your technology solves problems the customer cares about.

When I arrived at Ambit, I learned that our value was that we produced faster circuits than Synopsys. And we had better time-budgeting. And we could run top-down. And we ran faster. And our pricing was bundled. And physical synthesis was in development. And…and…and.

It was once we realized that we could handle large million gate designs in one gulp that we really started to get traction. The other things were all true, but none of them was compelling enough to get a company to engage with a startup. But if you had a million gate design and you couldn't get it through Design Compiler, then you were calling us to take your money.

This is different from the elevator pitch for investment purposes. Ambit was "Design Compiler only better" but that isn't focused enough for marketing a product or driving a detailed development roadmap. You have to find out the *one thing*.

Intel only needs one copy

It is obvious that companies make money in EDA only if they sell enough software. One rule of thumb is that EDA companies thrive if each salesperson brings in $2M per year, and they don't if they only bring in less.

But enough software really means enough hours of use of the software. For a large EDA company, most of the money comes from a relatively small number of large customers, and they optimize their use of licenses in server farms, sharing licenses world-wide and so on.

But enough hours of use of software in turn really means that either the software must run for a long time (like place and route or RET decoration) or else that customer engineers must sit in front of it for a long time (like a layout editor).

Other tools suffer from what I call the "Intel only needs one copy" problem. They have a hard time building license demand naturally. This is less of a problem in a startup, who are quite happy in the early days to sell one copy to everyone, but to get a good growth trajectory it is necessary to build on the beach-head of those first licenses and proliferate widely into at least some of the accounts.

If license demand isn't built naturally then it becomes necessary to attempt to do unnatural things like try and charge per tapeout, or try and license on a per-named-user basis, or try and charge a royalty. These are all possible but at the very least the sales cycle will stretch out for a startup, and it will run out of cash, or for a large company it becomes too complex to include a weirdly licensed tool into a large contract (which, incidentally, is also one reason that OEM deals never work in EDA).

This is one of the big challenges of the ESL market. The tools are only needed occasionally, don't run for a very long time and don't require users to run them interactively for long periods.

Bottom line: it is really hard to sell a tool with an unexpected business model, which for EDA means some sort of floating license for a period of time. A nice analogy is the restaurant business. When you go to a restaurant you expect to pay depending on what dishes you order. That's how restaurants work. But in fact most of a restaurant's costs are fixed: the rent, the employees' salaries, utilities, advertising. So rationally a restaurant might charge by the minute no matter what you eat. That changes things a bit (caviar is cheap, that espresso after dinner is really expensive) but even so I suspect you'd have a

hard time running a business that way. It's just not what the customers expect.

Super models

Xxxxx

I about open source software in EDA, or rather about the lack of it. One area where there is some free and open source software, as well as closed source software, is on that boundary between EDA for chips and software tools for embedded systems, namely what seem to be called virtual platforms or virtual prototypes (I hate the name "virtual prototype" since it is a chip-centric view of the world implying that the platform is useless once the chip shows up).

Virtual platforms, while they have some utility for chip development, are largely sold to software developers to allow them to do software development more productively and earlier than would be the case if they had to use the real hardware, which comes along too late and is too opaque. The performance of the virtual platforms is almost unbelievably high, running ARM or PowerPC code binaries at hundreds of MIPS on an off-the-shelf PC, often similar to the performance on the actual hardware.

I have worked for both VaST Systems Technology and Virtutech who both supply tools into this market. They charge per seat in the region of $5-25K/seat/year. In the IC design world these price point are low; in the software development world they are very high. Synopsys with its Virtio acquisition is also in this market. Imperas is a startup founded by Simon Davidmann in the UK to enter this market specifically to address the difficulty of programming multicore chips. Driven by a mixture of lack of funding but also a deliberate change of strategic direction, about a year ago they made their environment free and created Open Virtual Platforms (OVP).

I met Simon at DATE last year where this was announced, and asked him why he did it. Firstly, he said that he is a big fan of Kim and Mauborgne's book *Blue Ocean Strategy*, changing the rules and competing where the competition isn't. But the thing

that really brought it home was discovering a fact about QEMU. QEMU is a similar type of simulator developed largely by one person, Fabrice Bellard, and distributed free (and open source). The fact Simon discovered is that QEMU has more Google hits than Synopsys.

Think about that for a moment: a single free product that most of you have never heard of in a neighboring space to IC design has more web references than the EDA market leader has for all their products put together (they are almost identical at around 1.7M apiece when I looked just now).

Simon also realized that companies made more money from verification tools around simulators than selling the simulators themselves. So for Imperas the key would be to get people using the simulator so that there was a base into which to sell higher value tools. It is too soon to tell whether the strategy is working fully, but MIPS and Tensilica are both distributing models on the OVP foundation.

When I was at VaST and Virtutech it was clear to me that the market would be limited so long as models were not being supplied by the component vendors, either at the same time as or in advance of silicon. I always used the analogy of Synopsys in the early days. At first Synopsys themselves developed the ASIC vendor libraries necessary for synthesis. Bob Dahlberg, who ran the group, told me that at one point he had well over 100 people doing this. Then the ASIC vendors realized that it was their job if they wanted the job done how they wanted it done when they wanted it done. A year later Synopsys disbanded the group completely since ASIC vendors had completely taken over the task.

This is starting to happen in the automotive industry around VaST's technology, For some time the main suppliers into the automotive industry (NEC, Renasas, Infineon, Freescale and others) have supplied processor models for VaST's environment. NEC America is announcing today that they will be distributing complete virtual platforms on VaST's foundation technology into the automotive industry, going beyond simply providing processor models. Software engineers in tier-1 suppliers (automotive-speak for people like Delphi, Visteon and Denso)

and OEMs (automotive-speak for car companies like GM, BMW and Toyota) will be able to develop their software without having to wait for silicon to be available and in a much more productive environment than the real electronic control unit that will eventually ship in the cars.

However, I think that component suppliers will continue to remain reluctant to develop models for the virtual platform ecosystem while there are limited standards for interoperability or, as an alternative, a de facto winner in the same way as Synopsys was clearly the early winner in synthesis. Even a company like Freescale, which distributes VaST models into the automotive industry also distributes Virtutech models into the communication (think router and base-station) industry, which is clearly not optimally productive.

The situation where OVP, VaST, Virtutech, Virtio, QEMU, Bochs and others all have incompatible virtual platform environments is not really sustainable. SystemC provides some standardization around modeling of peripheral devices where performance is not critical, but processor models depend heavily on the underlying simulation technology to get their blazing performance.

The other alternative is native cross-compilation environments. If you develop software for the iPhone Apple supplies a Mac-based iPhone simulator. It is fast but people complain about its accuracy especially for graphics. But presumably Apple decided it was not worth using true virtual platform to get the accuracy at some loss of performance, or maybe they didn't even know just how fast simulation technology can be. Also, it is not so much iPhone application software but the call processing and low level software that absolutely requires a high accuracy platform.

It will be interesting to see how this all plays out.

Why does EDA have a hardware business model?

EDA really started back in the 1970s (late 60s in fact) with companies like Calma and Applicon. They drove the first EDA transition from cutting rubilith, red sticky plastic that was physically cut with X-acto knives, to digitizing the input and generating pattern-generation tapes for automated mask-making equipment. One legacy of this era remains with us today: Calma's system was called the Graphical Design System or GDS, and the second generation (32-bit!) was called GDS-II. Normally it stored its data on disk but it had a format, called "stream format" for writing the data out onto magnetic tape. The disks were too small to keep the designs there permanently. This format, GDS-II stream format (or often just GDS-II or GDS) was the standard for decades for moving layout data between systems and from design environment to mask shop. It is still not dead although it is definitely coughing up blood. The transition to Oasis or other formats has gone much slower than expected. Everyone supports GDS-II and so it is the least-common denominator format.

Calma and Applicon initially thought of themselves as hardware companies. They sold computers. Calma was actually a Data-General minicomputer (it's nothing directly to do with EDA but if you've never read it, you must read the book *Soul of a New Machine* about development of the 32-bit version). The business model was the same business model as most hardware was sold: you bought the hardware, digitizers, screens and so on. And you paid an annual maintenance contract for them to keep it all running which was about 15-20% of the hardware cost per year. The software was simply bundled into the price. This was before the days of a separate pure software industry; almost all software only ran on one brand of computer the way cell-phone software or digital camera software is sold today: you can't buy it separately, it's bundled with the hardware.

The next generation of EDA was also hardware-based. Gate-level design was dominated by the DMV: Daisy, Mentor, Valid. Daisy built the Daisy Logician, Valid built the Scald-station (I think

that was the name) and Mentor OEMed Apollo workstations instead of building their own. The business model remained the same: buy the hardware and pay an annual maintenance. I don't know if the software was even a separate line item.

It is hard to believe, but back in that period there was a worry in EDA that as the hardware costs came down then software costs would have to come down too. It was hard to believe that someone might pay more for software than the hardware on which it ran. After all, they never had before. Today, when you can run millions of dollars of software on a box costing a few thousand dollars this seems comical. But go back to the cell-phone software I mentioned earlier. Maybe one day we'll be paying $500 for good cell-phone software from an independent market, and then buying a cheap phone for $10 on which to run it. After all, that's where the value increasingly is.

The next generation of EDA software, VLSI Technology (where I worked), SDA, ECAD (that together became Cadence), Silicon Compilers, SDL and other companies that I'm sure I've forgotten, wrote software that was more hardware independent. They would run on Vax (always) and one or more of those new-fangled workstation thingies from Apollo (or Sun once they made it to production). They would usually sell you the hardware if you wanted, but you could just buy the software and run it on your own hardware. The business model was the same old hardware business model though: pay an upfront license and annual maintenance of 15-20%. This was how we ended up at first with a hardware business model for a software business.

The final change was the switch to the time-based license, initially 3 year, that we largely have today. Essentially this is a software lease. Gerry Hsu[1] is usually credited with this. He told me that he noticed that people like to lease cars so that they get a new car every 3 years or so, and decided to see if you could sell software the same way. It turned out to be a good idea and the financial side of the business liked it since it gave very predictable revenue. In any given quarter, most of the revenue was business booked in the three years before, and only a small amount from the new business booked that quarter.

Business is still done as a mixture of time-based licenses (recognized over the period) and term business (recognized up-front). But, as I said earlier, the mix is open to abuse and only the savviest followers of the industry realize how critical the percentage of ratable business is in trying to decide if an EDA company is doing well or not. It remains tempting for an EDA company to persuade a company to do that $10M deal as a term license instead of a time-based license. The $10M is recognized immediately and can fill an embarrassing $10M hole caused by the lack of $100M of time-based bookings.

[1] By the way, Gerry Hsu is sometimes portrayed as a bit of a buffoon. But he was certainly extremely smart, and very perceptive. Just somewhat ethically challenged. I worked for him for 8 hours!

The arrogance of ESL

ESL, or electronic system level design, is a catchall term for tools above the level of RTL. There are two primary aspects to this: synthesis and verification of IC designs from representations higher than RTL (usually untimed C or System-C); and tools that do something to address development of the software component of electronic systems.

I have no problem with the term ESL for the first of these segments, synthesis and verification. There are several EDA companies (Mentor with Catapult, Forte, Synfora, CriticalBlue, AutoESL, Cadence C-to-silicon) providing synthesis and one (Calypto) providing formal verification of this level of design. Getting design productivity up higher than pumping out RTL Verilog is necessary and these companies, despite their limited success, are probably part of the solution.

But when EDA companies turn to the software space they look at everything through their IC spectacles and assume that ESL methodologies in the chip design world will have some part to play in development of the software that runs on the chips. They have IC bias. But the software component of electronic systems is much larger and much longer-lived than the hardware (chip)

part. ESL thinking that it will impact software development is the tail trying to wag the dog.

I was at a keynote by the CTO of Cisco a couple of years ago. He revealed that IOS, Cisco's router software and operating system, is 25 million lines of code and there are an additional 35 million lines of scripts for testing it. Consequently the number one priority for any chip being sold to go into a Cisco router is "don't break the software." This is way ahead of anything to do with chip area, performance, power dissipation and so forth. Indeed, I heard (anecdotally, so this is hearsay) that Cavium, who have a 16 core MIPS processor, were unable to penetrate Cisco since IOS isn't multi-threaded enough to take advantage of all those cores. The chip has no problems and is probably desirable in all sorts of other dimensions but it breaks the software so game-over.

I once asked some embedded software developers at an electronic system company what they thought about ESL. This was fairly soon after I had joined VaST and still suffered myself from IC bias. I was expecting them to say it was promising, or they hated SystemC or something like that. Instead, they could only think of 'English as a second language' and had never even heard of ESL. Almost no software runs directly on the bare chip in any case, it is all intermediated by a real-time operating system such as Wind River's VxWorks, Green Hills's Integrity or, increasingly, some flavor of Linux (which includes OS-X on iPhone and Android on the Google-phone). This makes direct software-hardware co-design, where some of the code is optionally implemented in hardware, much more complex. Pulling out a block of software for synthesis into custom hardware (or for implementation on a special data-plane processor) requires the stubbed out software to make operating system calls to access a custom device driver that can talk to the custom hardware directly. Automating that process requires building not just the hardware, but the device driver and other operating system scaffolding, as well as the stub back in the original source code. Of course, that is completely operating system dependent and so requires multiple implementations.

Software people simply don't care how the chip was designed. The models created as part of the hardware design process are too slow by factors of thousands or even millions to be useful as part

of the software development process. But most importantly, the bulk of the software payload for the chip already exists in the form of previous versions of the product. Even a brand new product like iPhone carried over a lot of software from the Mac that was simply cross compiled to run on the iPhone's ARM processor.

Ferrari vs Formula 1

It used to be received wisdom that the way to get a good design flow was for a semiconductor company to purchase best-in-class point tools and then integrate them together themselves. I think there were two reasons for this. First, the EDA companies had grown from a lot of acquisitions so that's what they had for sale: good point tools that were poorly integrated. Second, they were selling to CAD groups in an era when semiconductor was doing well and CAD groups liked to justify their existence by doing lots of evaluation (which point tool is best?) and then integrating them (need lots of people).

For most people, this was actually not the best way to get a productive environment matched to their needs. It is as if we all had to buy cars the way a Formula-1 team does, buying the best engine, the best brakes, the best gearbox and making everything work well together ourselves at great expense. If you really need to win a Formula-1 race then this is the only way to go. Even a top of the line Ferrari is simply way too slow. But for most of us, a Honda Accord is just fine, easier to use, cheaper to acquire, and orders of magnitude less expensive to get and keep on the road.

Back in that era I was at VLSI Technology. When we spun out Compass we had a Honda Accord in a marketplace where people thought they wanted to build their own Formula-1 racecar. Potential customers only wanted to benchmark point tools and wouldn't even attempt to benchmark an entire design flow. I'm not even sure how you would. I don't know how much better the design flows that CAD groups assembled out of Cadence and Synopsys point tools (along with a seasoning of stuff from startups) really were. And neither does anyone else. They were certainly incredibly expensive in comparison. Before the spinout,

I made several visits to semiconductor companies whose CAD groups were bigger than VLSI's Design Technology group. But Design Technology developed all the tools, wrote all the source code for synthesis, simulation, timing analysis, place and route, physical verification, designed all the standard cell libraries, created the memory compilers and the datapath compiler. Soup to nuts. I think the only external tool in wide use was for gate-array place and route, an area where VLSI was never that competitive anyway (if you really wanted a gate-array, you went to LSI Logic).

Magma was the first and only EDA company to build an integrated environment. A CAD manager friend of mine told me that they used Magma for everything they could. For the most difficult designs they used Cadence's Silicon Ensemble but they could train someone on Magma in a day (and they weren't immediately hired away by the competition once they'd been expensively put through training).

At the EDAC forecast meeting a couple of weeks ago, Aart de Geus said he has been preaching that an integrated flow is important for years. One difference he is noticing in the current downturn, he said, is that this time executives are listening. Chi-Ping Hsu of Cadence told me the same thing about the Cadence PFI initiative which was well-received by power-sensitive customers (is there another sort of customer?). PFI's main thread, the CPF standard, pulled together tools from across Cadence's product line along with standards that allowed external tools to play in the flow too. Synopsys UPF does the same thing on their side of the standard wars trench. People had managed to put together power-aware flows before, lashing together point tools with lots of their own scripts. But they were very buggy and many chips failed due to trivial things like missing isolators or not taking getting the timing right in multi-voltage blocks. This seems to be a thing of the past now, although most designs are still on the basic end of power saving (fixed voltage islands, power-down) and not yet attempting the really tricky things like dynamic voltage and frequency scaling (lowering the voltage and slowing the clock when there is not much to do).

In the current hyper-cost-sensitive environment I think that the pendulum will swing back the other way towards these more pre-integrated flows and away from the integrate-your-own-point-tools approach. It is also the only way that complex factors like power, that cut across the whole design flow, can be accommodated. The slowing of startup acquisitions by the majors feeds into this, giving them time to put the effort into integration without constantly gaining more things to integrate. The integration has enormous value despite the fact that customers have been historically reluctant to pay vendors for it. When I was at Cadence we had some research showing customers spent $3 or so on integration for every $1 that Cadence got. So customers were paying for it, just not externally.

He who goes first loses

There's a big debate about whether innovation occurs most in small or large companies. I've always maintained that the problem is a different one. I think it is clear that the engineering groups of large companies are capable of creating leading edge technology. Look at any franchise product like Design Compiler, Virtuoso or Verilog simulation and see how it has advanced over many generations spread over a decade or more in ways that involve large amounts of innovation.

Where large companies have a problem is that they are very poor at introducing new products into their channels. They have large efficient sales organizations but those organizations are geared up to closing deals with customers for products that the customer already knows it wants. Unfortunately, when a brand new product is introduced, there is an attitude among the salespeople that "he who goes first loses." But just as the Luddites really were right that automatic looms would put them out of business, the first person in a large company to sell a new product really does lose. There will be problems with the product that will tie up their application engineering resources for months, and potentially a large multi-million dollar deal will be held hostage to problems in a single copy of a hundred-thousand dollar tool. Better simply not to sell the product until enough other sales have

been made for it to be mature. But with every salesperson taking this attitude, no sales occur.

This can extend even to products that are acquired. When Cadence purchased Ambit's synthesis product line, it was obviously very strategic for Cadence salespeople to sell it aggressively. If they were successful, it would start to cut off money flowing to Synopsys and even if they were less successful, they would force Synopsys to circle the wagons to protect its Design Compiler franchise and so have less effort available to put into threatening Cadence's huge place and route franchise. But Cadence salespeople would not. They had big quotas at big semiconductor companies to close, and their focus was to let Synopsys have synthesis and try and close a deal to supply everything else. Selling synthesis against Synopsys required extra effort and the payback of a few experimental licenses would not move the needle on their quota.

Another product from my time at Cadence was called Heck (at least internally, I forget what unmemorable name it got given externally). It was a formal verification tool built on some technology developed at Cadence Berkeley Labs. To tell the truth, I've no idea whether it was any good or not, but since the salespeople refused to try and sell it we never found out. In the end Cadence acquired Verplex and the Conformal product line that customers were already starting to adopt.

Very few products have been successfully introduced by large EDA companies (once they have become large). By successful I mean built up into $100M per year businesses. And by product I mean a genuinely new product line, not a new version of an existing product. The only one I can think of is Calibre. This was developed over the years inside Mentor and somehow survived being canceled for almost a decade before coming to dominate physical verification. Cadence helped by making a huge misstep. They tried to protect their Dracula franchise by making their hierarchical DRC Vampire require incompatible rule decks. Mentor had no such qualms and as a result the obvious upgrade path from Dracula was to Calibre not Vampire.

Synopsys made PrimeTime a big success, but the story is complicated by the fact that they acquired Viewlogic and with it

Motive, the market leader in static timing. They then shut down Motive and transferred all its customers to PrimeTime. But undeniably they did manage to get their salesforce to sell it.

So I think that it is not so much that large EDA companies are incapable of innovation. They do it all the time. But their salesforces are reluctant to sell any product for which there is not already strong market pull. Marketing in EDA is unable to create that demand either, which is a different story.

However, startups are different. The salesforce will sell new products because the salesforce typically has precisely one product to sell, and it is new. They are not really the same sort of salesperson either. Startup salespeople are more like hunters whereas large company salespeople are farmers. It seems to take that combination of single mindedness in the salesforce and an entire company whose success depends on getting those initial customers to adopt the product. Once customers start to clamor for the product, it is the moment for a large EDA company to acquire the startup and the huge machine of their salesforce can drive the bookings number up very rapidly.

All purpose EDA keynote

I've given lots of keynote speeches about EDA over the years. You too can give your own keynote if you follow these simple secret guidelines.

Ladies and gentlemen…

Moore's law…blah, blah, blah. Show generic Moore's law slide. New challenges. Scary.

Design gap…blah, blah, blah. Show generic design gap slide. Must close the gap. Scary.

Chips are getting bigger, more physical effects are becoming important, wavelength used for lithography is not changing, engineering productivity must increase.

The three mega-trends: drive up the level of abstraction for greater productivity, drive down the level of detail since second-

order effects are becoming first-order, and increase integration to improve productivity.

So far everything has been completely generic. You could have given the same speech a decade ago. If you did, it is a good idea to at least update the years on your generic slides so they don't finish five years in the past. Now it's time to get vaguely specific. You'll need to update the rest of the keynote at least every process node. That's only every couple of years so not too much work.

Talk about big issues of the day that affects everyone. Power is hot (or perhaps that should be cool) or how about process variability, or impact of new lithography restrictions. If you talk about power, talk about how power format standards (or at least the one you support) will make everything straightforward. Don't forget how committed you are to standards.

Drive up level of abstraction so that front-end designers are more productive. Talk about the architectural level; nobody is quite sure what it is but it is big picture so wave your hands a lot. Maybe talk unconvincingly about need to take embedded software into account. The audience knows nothing about it but they have whole groups doing it, and they are bigger than the IC groups, so it must be important. Talk about importance of IP and doing design using much larger blocks. This is a good time to talk about standards again and how committed you are to them. System-C and transactional-level modeling are good names to drop. Verification is 60% of cost of design. Tradeoffs need to be done at architectural level for greatest effect, later in the design cycle is too, uh, late.

Drive down level of detail so that we take into account new physical and manufacturing effects we used to be able to ignore. "You can't ignore the physics any more" makes it sound like you didn't forget all the physics you learned in college. Designers need to worry about process variability and will need statistical timing tools to worry with. And after thirty years of pretty much putting what we want onto masks we are not going to be able to do that any more. Good moment to have scary pictures of the difference in how layout looks on the screen to the mask to the silicon.

Need for greater productivity. Next generation databases. If yours is open, argue about why this is public spirited, sustainable and green. If yours is closed, argue about how that enables your tools to be more optimized and efficient. Everyone needs more integrated tools. Nothing is fast enough so your tools will all be multi-threaded one day. Soon. You hope. Flows are important. Unless you only have point tools in which case talk about how best-in-class point tools are even better than flows.

You are short on time so slip in a quick mention of manufacturing test. Who knows anything about it? But chips have to be tested so talk about scan. Or BIST. Or ScanBIST. Then there's packaging and printed circuit boards. They are probably important too, but everyone in the audience is a chip designer. Best not to think too much about them.

They don't design FPGAs either, but good to mention them to show you understand how widely they are used. But there's no money in EDA for FPGAs so best to gloss over exactly what capabilities you have.

Wrap it up and get off the stage. We are working hard on all these areas. We are your partner for the future.

No sex before marriage in EDA

In most businesses, every company doesn't feel the need to make every product that it sells. When you buy a car from General Motors, they don't make the ABS system themselves, they buy it from Delphi or from Bosch. When DEC came out with the Vax, they didn't feel the need to make their own graphics terminals, they bought them from Tektronix and re-badged them.

This is known as an OEM deal. OEM stands for "original equipment manufacturer" and refers to the fact that General Motors is the manufacturer of the original equipment (the car) and the other parts are treated by regulation as if GM had made them themselves. Indeed, they may even badge the part with their own logo and make it hard to find out just who is the real manufacturer.

OEM in other industries has come simply to mean re-selling stuff created by another company. In EDA software, for example, almost everyone's schematic viewer is actually a product from Concept Engineering in Germany.

But this sort of deal, where a component of the product is incorporated from an external company, seems to be the only sort of OEM deal that works. Once the deal moves up to the level of a whole tool then OEM deals almost never work in EDA. There seem to be two reasons for this, one on the customer side and one on the vendor side.

On the customer side, if you are buying a product from bigEDA and you know that it really comes from littleEDA, then why would you not want to deal with littleEDA directly? If you have a problem, you know that bigEDA is just going to pass the question onto littleEDA anyway, and even before you buy it there may be some channel conflict when both bigEDA and littleEDA are competing for your business, and for sure the littleEDA sales team knows much more about the product. It just doesn't make too much sense to flow your dollars to littleEDA through bigEDA, and flow their support back through the bigEDA support channel.

On the vendor side, bigEDA wants to do big deals with their major customers. They'll give you all your EDA software, or a good part of it, for all your EDA budget, or a good part of it. OEM deals usually require a per license payment from bigEDA to littleEDA but that doesn't fit well with a deal where technically the semiconductor company may be getting unlimited or a large number of licenses for a bundled sum. There is simply no way to calculate an appropriate number of license fees to pay littleEDA, and the need to do so makes the deal more complex and so the salesperson simply drops the OEM product as not worth the usually minimal increment in bookings.

Finally, there is a strategic reason that makes OEM deals unattractive. You'd think that an OEM deal would be sex before marriage. If the deal works well then bigEDA can buy smallEDA. The trouble is, if the deal works well then another big EDA company might make a move. And either way, bigEDA is going to have pushed up the price of smallEDA and is going to

have to buy back their own revenue. There's no sex before marriage in EDA. If smallEDA is the right company then marry them immediately before they get more expensive.

Standards

I was once at a standardization meeting many years ago when a friend of mine leaned over and said, "I tend to be against standards, they just perpetuate other people's mistakes." I think this is really a criticism of standardizing too early. You can only standardize something once you already know how to do it well.

In many businesses, the winner needs to be clear before the various stakeholders will move. Standards are one way for a critical mass of companies to agree on the winner. For example, Philips and Sony standardized the CD for audio and since it was the only game in town it was adopted immediately by vendors of CD players, the record labels knew which format to put discs out in, the people building factories to make the CDs knew what to make. A few years earlier there had been the first attempt to make videodiscs, but there were three or more competing formats. So everyone sat on their hands waiting for the winner to emerge, so in the meantime everything failed. When everyone tried again a few years later, the DVD standard was hammered out, it was the winner before it shipped a single disk, and the market took off. This was a lesson that seemed to have been lost in the HD-DVD vs BlueRay wars, although by then disks were starting to be irrelevant and downloading and streaming movies is clearly going to be the long-term winner.

EDA is an interesting business for standards. Since you can only standardize something you already know how to do, standards are useless for anything leading edge. By the time we know how to do something, the first batch of tools is out there using whatever interfaces or formats the initial authors came up with. Standardization, of the IEEE variety, lags far behind and serves to clean up the loose ends on things where there are already de facto standards. Also, EDA market expansion is not going to be driven by standards in the way that CDs were. Synopsys won synthesis (as opposed to Trimeter, Silc, Autologic and others)

and so .lib and sdc became the standards, not the other way round. If all the other EDA companies had created a competing standard to .lib, nobody would have cared. It is the winningness not the standardization that is important.

Once the first tools are out there for some new technology, all using incompatible formats, then standard wars begin. The market leader wants its standard to become the de facto standard adopted by everyone. It is cheap for them since they don't need to make changes; it is expensive for everyone else since they need to change their software to read the standard and probably make some internal changes so that their tool's semantics match those implicit in the standard. Even if an IEEE-style standardization effort takes place, it is too slow. By the time the standard comes out it has often already been superseded by upgrading of the formats by the market leader to accommodate the realities of the process nodes that have come along in the meantime.

Customer behavior is very two-faced too. Every semiconductor vendor will talk about the importance of standards with a long solemn face. Especially their CAD managers. But, at least for their leading edge chips, they won't put any money behind those statements and they will buy the best tool for the job whatever standards it does and does not support. Designing leading-edge chips is hard enough without worrying about whether some abstract standard is open enough.

Of course, once a market matures then supporting the de facto standard is an important part of "best tool for the job". When I first started in EDA, Calma still maintained that GDSII was a proprietary standard that nobody else was allowed to read. However, every Calma system shipped with a file describing the format, so I took the legally dubious step of reading that file, and a couple of days later we could read chips into the VLSI Technology layout editor. A layout editor that didn't read GDSII wasn't really a layout editor no matter how good it was at editing layout.

So expect customers and EDA vendors going forward to talk a lot about how important standards are. But expect them to produce

and buy the best tool for the job and the standard to emerge from the competition for that honor.

Semi equipment and EDA

I had lunch with Lance Glasser a couple of weeks ago. He used to run about half of KLA-Tencor's semiconductor equipment business (and I did some consulting for him back then). We got to discussing why EDA and semiconductor equipment are so different.

At first glance, there are a lot of parallels with EDA. Most notably the same customers, the same technology treadmill and a small number of large companies without a lot of differentiation in their product offerings. But there are big differences. For equipment, the innovation often comes in the big companies, which have shown themselves capable of both developing innovative technology (involving not just optics and hardware but also a huge amount of complex software—60% of the engineers at a typical equipment company are software and algorithms) and also getting that technology successfully into their channel. Big EDA companies are not good at that. Why the difference?

Semiconductor companies know that they need both new equipment for the fab and new design tools for their design groups in order to bring a new process node online. In general, the most advanced fabs (such as Intel or TSMC) work very closely with the equipment vendors on the spec of new equipment and then on ensuring that the equipment works properly in the new environment. If you think it is hard to get your hands on a netlist for a next generation design, try getting your hands on some test wafers when most of the equipment does not yet exist. And when the equipment is ready for production, the fabs have no expectation that they will get it for free in return for this work, though they will certainly drive for deep discounts. As Lance said, sometimes the customers think "JDP" stands for "jumbo discount program."

One big difference is the way equipment is sold. Of course it is hardware not software, which means that neither the salesperson nor the buyer know the exact incremental cost and so what the

profit margin is at any particular price, although Intel actually invests in a "should cost" program to work out what they think a piece of equipment should cost to give them better negotiating leverage.

Another big difference about hardware is that it has lead-time. If you want to open your fab by such-and-such date then the equipment needs to be ordered by a much earlier deadline. This makes the negotiation much more balanced: the equipment vendor can delay knowing that the clock is ticking. Yes, they want the order but the fab absolutely has to close a deal by a given day. The only time a similar situation would exist in EDA is if a big semiconductor company were stupid enough to leave negotiating a new deal until right up to the last day of the old deal when all its existing licenses would expire. Then the EDA company could just delay too. This advantage had decreased in recent years as the customers place a larger percentage of their orders within lead time (to try to transfer the inventory risk to the vendor), but it is still not a bad as with software.

The other difference about equipment is that it really is a one-time buy, a true "permanent license." You buy a piece of equipment this year and you pay for it this year. Next process generation you don't "rebuy" all your existing equipment with just a soupcon of new stuff such as better optics. But with software you do. So even though a new piece of equipment may contain a lot of the previous generation in its design, the semiconductor company doesn't expect to get that bit for free on the basis that they already paid for it in the previous generation.

The way EDA works, even the old days when EDA still had a hardware business model and sold permanent licenses, there was always a debate as to how much of a new product was incremental (thus expected to be included as part of maintenance) or was a new tool (thus required a new permanent license). Today, with time-based licenses, much of a salesperson's quota may be "re-selling" the existing capability. When so much is riding on just keeping the customer on-board using the existing tools, the salesperson becomes very risk averse about selling new products. Unless the customer insists on buying, it is only a small amount of incremental revenue for possibly a large amount of

incremental problems. From the salesperson's perspective better not to include it in the deal at all. For the EDA company as a whole, in the short term and looking at just that one deal, this is rational. It is only in the longer term and in the aggregate that not getting new products into the channel is a slow death. Equipment companies often structure their sales incentives around penetration, share, and adoption of new products. More insidiously, this style of business (all your money for all your needs satisfied) means that EDA does not attempt to sell to value, does not attempt to increase the meaning of "all your money." Customer companies, who know the value, make it hard discover for the EDA company. For example, it is hard to find out how heavily individual tools are used. Equipment for 45nm is harder to engineer than it was for 180nm and so everyone expects it might cost more. (It is not all one-sided, EDA companies don't have to worry about wafer size changes—the equipment industry still hasn't made back the cost of changing from 200 to 300 mm.)

An equipment salesperson is more like an EDA startup salesperson. If he or she doesn't sell new equipment, there isn't anything else to sell. Very little ramping of production goes on except in the latest processes. There is almost no market for new 90nm steppers today, for example (there's probably a second-hand market though, they used to advertise that sort of thing on billboards along 101 between San Jose and San Francisco).

Little differences in the details seem to have a huge effect of the business. The fact that there is no concept of a software upgrade in equipment, the fact that hardware is solid and has real cost, that it has lead-time, has meant that equipment companies cannot go to zero on pricing, have to increase prices since their costs increase, and have to work closely with early adopters to mature the product. EDA companies have given up trying to sell the value of new products and so have given up trying to grow their customers budgets. So they don't grow, and EDA is probably smaller than it was five years ago (if we exclude IP).

It's like football only with bondage

Woodrow Wilson once said "If I am to speak ten minutes, I need a week for preparation; if an hour, I am ready now." Being succinct is really important when trying to close some sort of deal, whether it is a CEO trying to convince and investor or a salesperson trying to convince a customer. And as the Wilson quote shows, it is really hard.

Analogies are a great way of explaining things. You probably heard that movies are often pitched in a ten-second bite "It's like xxx only yyy." For instance *Alien*: "It's like *Jaws*, only in space." Or *Chicken Run*: "It's like *The Great Escape* only with clay chickens."

Investors can be pitched this way too. They typically don't really understand the technology they are investing in so it's no good talking about how great your modifications to Kernighan-Lin are for next generation 32nm placement in a restricted design rule environment. Better to say "It's like Silicon Perspective but taking modern process limitations into account."

When I was at Ambit, we had a product called PKS (physically knowledgeable synthesis) which was the first synthesis tool that took physical layout into account in timing. But it was hard to explain to people why this was important back then, everyone was used to synthesis with wire-models and didn't really understand the limitations. I found that the best way to explain it was that it was like trying to find the distance you'd have to travel to visit 4 cities in the US. It clearly makes a big difference if you know the cities are in LA, Miami and Seattle, as opposed to LA, Phoenix and Las Vegas. If you know nothing about where they are, which is the wireload model case, all you can do is use some sort of average and say it is 1500 miles. Always. This analogy also served to overcome the objection that we were not using the precise placement that would end up after physical design. If the cities are LA, Miami and Seattle, it doesn't matter that much that the Seattle visit was actually to Portland; it's close enough and a lot better than assuming Portland, Maine. I found that with this analogy people would immediately understand the

reason for what we were doing and the limitations in the old approach.

Another analogy I like is in multi-core. Forget all the programming but just focus on the infrastructure. Everything assumes, or rather assumed, a certain model of programming: the programming languages, the hardware, the operating systems., the way programmers wrote code assuming that future computers would be more powerful not less It's like containerization. The whole shipping infrastructure of the world is built on a standard sized container. Multi-core is as if someone suddenly said that you couldn't have container trucks any more, for each big truck you used to have you now get a dozen FedEx delivery vans. In fact you can have millions of them, they are so cheap and getting cheaper. The trouble is that the infrastructure doesn't work like that. The carrying capacity of millions of FedEx trucks might be much more than the container trucks, but the legacy stuff all comes in containers. It just doesn't do to look only at the total carrying capacity.

A company I'm on the board of, Tuscany Design Automation, has a product for structured placement. In essence, the design expert gives some manual guidance. But people are worried at how difficult this is since they've never used a tool that made it easy. It really is hard in other tools where all you get is to edit a text file and don't get any feedback on what you've done. The analogy I've come up with is that it is like computer typesetting before Macs and PageMaker and Word. You had text-based systems where you could put arcane instructions and make it work but it was really hard and best left to specialists. Once the whole desktop publishing environment came along it turned out that anyone (even great aunt Sylvia) could produce a newsletter or a brochure. It was no longer something that had to be left to typesetting black-belts. And so it is with structured placement. Once you make it easy, and give immediate feedback, and people can see what they are doing then anyone can do it.

Pricing. Vases and coffee pots

My father-in-law was an executive at Wedgwood and responsible, among other things, for pricing every piece of china they made. Wedgwood has recently been run into the ground by Waterford Crystal who had acquired it, and it is now in administration (roughly chapter 11), but that is another story.

Anyway, a Wedgwood coffee pot sold for $80 (I'm guessing these numbers). It is a complicated piece to make consisting of a body, a handle, a spout, a lid and a little piece that goes inside to stop all the coffee grounds going down the spout. However, if we throw away the lid, don't bother to put on the spout, forget the handle and the little filter piece then we have a much simpler item. It's a vase. It sells for $100. Paradoxically, the vase, which is much simpler to manufacture, has higher value to the consumer and so sells for a higher price.

It is important not to assume that the value to a customer of every product is a fixed industry markup over its cost. That will be true in a competitive mature market since any differentiation that leads to a higher price will get copied and competed away, but it is not true when products are differentiated. That is why it is so important to have differentiation, otherwise you are stuck selling silicon at a small markup to cost and you probably are not the lowest cost supplier. You either want to be Walmart (lowest cost supplier) or Whole Foods (lots of differentiation), not Safeway.

When I was at VLSI Technology in the late 1990s, one of the things I did was help run the strategic planning process for VLSI's communication business (mostly GSM chips). We had a relationship with a French company called Wavecom that had a GSM software stack and a GSM radio design. VLSI made the baseband chip. At the time, understanding the GSM standard well enough to build a baseband chip was a big differentiation, and we were one of a couple of semiconductor companies with a standard product, which gave us some pricing power. But it was easy enough to see that, just as had happened in the PC chipset business, lots of competitors would enter the market and it would become a cutthroat cost-plus business. Digital design, no matter how complex, is not defensible for long. I told our

communication group that we had better have a plan for acquiring Wavecom (which didn't want to be acquired, certainly at any price we could afford) or else we should have a plan for finding a new software/RF partner since that is where the differentiation would move. Otherwise we would eventually get pushed out of the market. In the end Philips Semiconductors acquired VLSI in a hostile takeover, and they already had a software stack and RF and so the problem got solved that way. But it wasn't a message the communication division wanted to hear since building those chips was so hard they wanted to believe that it would continue to be differentiation for a long time.

Software has a disadvantage over hardware in that the manufacturing cost is known. It is basically zero. All the cost of software is really an R&D cost being amortized over product sales. It is like a pharmaceutical business in that sense, pills that are incredibly expensive to formulate but incredibly cheap to manufacture. But unlike the pharmaceutical business where IP protection works, there is very little that can be done in practice to keep a product proprietary. It is thus hard to keep differentiation and pricing power for long. And hard even when there is no competition. Hardware accelerators such as Cadence's Palladium or Eve's products have a large hardware cost and sell for a high price. But despite that proven value, you know that if a software product had the same performance it would not sell at the same premium price point.

Of course these things are relative. In most software businesses, the prices we get for EDA tools are prices that other companies dream about. This is a challenge as the ESL market grows and starts to meld with the embedded software market. We may think a Verilog simulator is cheap, but it is a lot more expensive than a compiler or a debugger (even ignoring open source where the price is often zero). But that is a topic for another day.

The main implication for EDA is that the value of a product is determined by the customer not by the EDA company. There is plenty of pressure from customers to reduce prices on non-differentiated products, but very little from the EDA companies

to get more return from the strongly differentiated products. Too many coffee pots and not enough vases.

A real keynote: move up to software

I gave a dinner keynote at the Electronic Design Process 2009 meeting in Monterey last week. However, I'd already made the mistake of giving the secret recipe for any keynote speech (and, by way of confirmation, I received an email from Aart assuring me that his keynotes had rigorously followed the outline for the last fifteen years!). I would have to come up with something different.

So this is my current keynote. I focused on what I thought were the four big opportunities right now. In fact, in a funny sort of way, they are different facets of the same opportunity.

The first is that semiconductor companies now ship a lot of software along with their silicon, but by and large have not found a way to turn that into premium margins. They still ship margined up silicon and regard software as a marketing expense that is required to be in place for anyone to buy it. Semiconductor companies now have more software engineers than design engineers so this is backwards. It is the silicon that has little value and should be thrown into the deal.

The second opportunity is what I call "Coore's law." Just as with Moore's law described how the number of components on a chip was increasing exponentially, the number of cores on a chip is increasing exponentially. We are still at the fairly flat part of the curve so it's not that obvious yet. But, as I've said before, the semiconductor industry has taken their power problem and dumped it on the software industry in the form of multi-core. But they completely underestimated the impossibility of software solving this problem in a reasonable timeframe, and some people contend it will never be solved (the existence of our brains as a counter-example notwithstanding). And that is for new code written in new languages with new tools. Most code is legacy code, and legacy code is often what I've heard called "stiff ware." Technically it is software and malleable. In practice, nobody understands it well enough to make extensive changes

(and in the worst cases, not all the source code has been properly preserved). Anyway, incremental improvements to solving the multicore challenge are the next opportunity.

The third opportunity is that EDA business models (sell lots of expensive licenses) don't scale into the software world, or even the ESL world for that matter. In the same way, IBM had to learn the hard way that mainframe business models don't scale into the world with ubiquitous computing and ubiquitous networking. A lot of ESL has the problem of "Intel only needs one copy" and the software world suffers from open source killing innovation. Open source is clearly the most effective approach to software development, but it does best at copying things and has a poor track record of true innovation (if you are writing for yourself or copying, then the spec is easy). The obvious analogy here is with music. There will never be another Michael Jackson (and there's some upside to that too, of course) and Thriller will forever be the best selling album. Nobody will make serious money selling music itself and they can only make money by selling stuff associated with music that is harder to copy: clothing, concert performances and so on. In just the same way, hardware companies ride on products like Linux, recovering any development they do for the community (if any) through their hardware margin. Nonetheless, the opportunity is to move EDA from just plain IC design up to these higher levels and find a business model that makes it work.

Finally, the fourth opportunity is to look still further afield and take in the entire design process, in a similar way as PLM companies like IBM, PTC and Dassault do for mechanical, but with considerably less technology on the design side. Take the "E" out of "EDA". By taking the entire design problem, the business model issues associated with software might be side-stepped. And all four challenges are really about software.

In summary, the challenge is to expand from EDA as IC design (which is the most complex and highest priced part of the market) to design in general, in particular to take in the growing software component of electronic systems. It's a technology problem for multicore, but most of the rest is a business challenge

Competing with free EDA software

Chris Anderson (editor of Wired, owner of TED, author of The Long Tail) has a new book called Free coming out in July. One thing that he emphasizes (at least in his articles on the subject, I've not seen the book itself) is that "free" is very different from "really cheap." If you are an Amazon Prime subscriber, whereby you get free 2-day shipping once you've paid an annual fee, or if you are an iPhone user whereby you get unlimited data access, you know that this changes your behavior. People instinctively know this when they sign up for monthly gym membership; most people would be better just paying the one-time fee each time they go but they know (or at least hope) that "free" will change their behavior and they will use the gym more.

Websites are increasingly "free," meaning either that they haven't yet found a business model, like Twitter; or that they are advertising supported, like EDN or plentyOfFish; or that they don't attempt to make money on the website and exist for some other reason, like Wikipedia or Moveon.org.

Alternatively, many websites use what has become known as the "freemium" business model. A large part of the website is free, but if you want to get to the best stuff or want more then you have to pay. For example, flickr is free but if you want a lot more storage you have to pay; almost all games have some initial levels free so that you can start playing and only need to pay once you get in deeper.

One challenge in EDA is that the big companies bundle a lot of tools together for a single price, so effectively all of the tools are free (in the same way that going to the gym is free once you've paid the subscription). As planned, this makes it hard for small EDA companies to compete. Their tools haveto be so much better that customers will pay for similar tools that they already own.

I had lunch with Paul Estrada (a.k.a. Pi) a couple of weeks ago. He is COO of Berkeley Design Automation (which is obviously located in…Santa Clara). They produce a SPICE-accurate circuit simulator AFS that is 5 to 10 times faster and has higher capacity than the big company SPICE tools. For designers with really big

simulations, that is a pretty compelling value proposition (over lunch instead of overnight). But for designers with smaller simulations and access to unlimited big company SPICE simulators, it is harder to convince them to even take a look, never mind open their wallets. However those slow big company simulators still tie up hardware (and circuit simulators are both CPU and memory intensive, so need the good stuff) and they keep expensive designers busy waiting.

So Berkeley recently introduced a block-level SPICE tool, AFS Nano, that sells for only $1,900. This literally saves customers enough in hardware to justify the purchase, even if they have a pile of big company SPICE simulators stacked up on the shelf. Oh yeah, and those expensive designers can get back to work. It is not quite the freemium business model (which would require giving AFS Nano away) but it is close. Like with the other models, Berkeley hopes the near-freemium AFS Nano will get customers interested in their big tools.

Another interesting book is *What Would Google Do?* by Jeff Jarvis. He examines lots of businesses and wonders what they would look like if you largely gave away everything to make the user experience as good as possible, and then found alternative ways to monetize the business.

EDA software is notoriously price-inelastic. It doesn't matter how cheap your tool is, it has a relatively small number of potential users. You might steal some from a competitor, but overall the number of customers is not driven by the price of the tools in the same way as, say, iPods. So a free business model is unlikely to work unless there is a strong payment stream from somewhere else such as a semiconductor royalty. There is also a high cost to adoption in terms of training, setting up technology files and so forth meaning even "free" EDA software isn't really free once you get it into use. So it is unclear what Google would do in the EDA space, other than not enter it since it is too small to be interesting to them.

It’s turtles all the way down

According to Steven Hawking, Bertrand Russell once gave a public lecture on astronomy. He described how the earth orbits around the sun and how the sun, in turn, orbits around the center of a vast collection of stars called our galaxy. At the end of the lecture, a little old lady at the back of the room got up and said: "What you have told us is rubbish. The world is really a flat plate supported on the back of a giant turtle." The scientist gave a superior smile before replying, "What is the turtle standing on?" "You're very clever, young man," said the old lady. "But it's turtles all the way down!"

Electronic systems are a bit like that. What is a system depends on who you talk to, and a system to one person is built out of components that are themselves systems to someone else. Pierre Paulin neatly defined system-level as “one level above whatever level you are working at.”

In the EDA and semiconductor world we are used to talking about systems-on-chip or SoCs. But the reality is that almost no consumer product consists only of a chip. The closest are probably those remote sensing transport fare-cards like Translink now creeping around the bay area (finally, well over 10 years after Hong Kong’s Octopus card which was probably the first). They are self-contained and don’t even need a battery (they are powered by induction). Even a musical birthday card requires a battery and a speaker along with the chip to make a complete system.

Most SoCs require power supplies, antennas and a circuit board of some sort, plus a human interface of some sort (screen, buttons, microphones, speakers, USB…) to make an end-user product. Nonetheless, a large part of the intelligence and complexity of a consumer product is distilled into the primary SoC inside so it is not a misnomer to refer call them systems.

However, when we talk about ESL (electronic system level) in the context of chip design, we need to be humble and realize that the chip goes into something larger that some other person considers to be the system. Importantly from a business

perspective, is that the people at the higher level have very little interest in how the lower level components are designed and it is technically hard to take advantage of in any case. The RTL designer doesn't care much about how the library was characterized; the software engineer doesn't care much about how the language used for the RTL and so on.

At each level some model of the system is required. It seems to be a rule of modeling that it is very difficult to improve (automatically) the performance of a model by much more than a factor of 10 or 20 by throwing out detail. Obviously, you can't do software development on an RTL model of the microprocessor; too slow by far. Less obviously, you can't create a model on which you can develop software simply by taking the RTL model and reducing its detail and speeding it up. At the next level down, the RTL model itself is not something that can be created simply by crunching the gate-level netlist, which in turn is very different from the circuit simulation model. The process development people model implants and impurities in semiconductors but those models are not much use for analog designers; they contain too much of the wrong type of detail making them too slow.

When I was at Virtutech, Ericsson was a customer and they used (and still do, as far as I know) Virtutech's products to model 3G base stations, which is what the engineers we interfaced with considered a system. A 3G base station is a cabinet sized box that can contain anything from a dozen up to 60 or so large circuit boards, in total perhaps 800 processors all running their own code. Each base station is actually a unique configuration of boards so each had to be modeled to make sure that that collection of boards operated correctly, which was easiest to do with simulation. Finding all the right boards and cables would take at least a couple of weeks.

I was at a cell-phone conference in the mid-1990s where I talked to a person in a different part of Ericsson. They had a huge business building cell-phone networks all over the world. He did system modeling of some sort to make sure that the correct capacity was in place. To him a system wasn't a chip, wasn't even a base-station. It was the complete network of base-stations along with the millions of cell-phones that would be in

communication with them. He thought on a completely different scale to most of us.

His major issues were all at the basic flow levels. The type of modeling he did was more like fluid dynamics than anything electronic. The next level down, at the base-station, the biggest problem was getting the software correctly configured for what is, in effect, a hugely complex multi-processor mainframe with a lot of radios attached. Even on an SoC today, more manpower goes into the software than into designing the chip itself.

And most chips are built using an IP-based methodology, some of which is complex enough to call a system in its own right. So it's pretty much "turtles all the way down".

Don't listen to your customers

There is a train of thought that the route to success in a business is giving a customer what they say they want. At some level this is obviously good advice. But there are two problems with it. Firstly, the customer always wants incremental improvement on what they already have, and rarely is imaginative enough to ask for what they really need. And secondly, this can lead to design by committee producing a product that has too many features to be usable.

A nice example of this is Apple's design of the iPhone. Nobody knew they wanted it. In a vague sort of way they probably wanted a phone from Apple knowing it would be Mac and iPod-like. Luckily Apple didn't simply go and ask all the carriers what they wanted, they designed what they wanted to and then found a carrier willing to take it largely unseen. Of course lots of people were involved in the iPhone design, not just CEO Steve Jobs and chief designer Jonathan Ive (another Brit, by the way, referring back to my post about the benefits of easier immigration) but it was designed with a conceptual integrity rather than a list of tick-the-box features. The first version clearly cut a lot of corners that might have been fatal: no 3G data access, no GPS, no cut-and-paste, no way to send photos in text messages, only a couple of applications honored landscape mode. The second version came with 3G and GPS. Most of the rest of the initial peeves are now

fixed in the 3.0 version of the operating system (which, as a registered iPhone developer, I already have installed). But the moral is that they didn't ask their customers to produce a feature list, and they didn't make an attempt to implement as much of that list as possible.

When I was at Cadence we were falling behind in place and route. So we decided to build a next generation place and route environment including everything the customers wanted. It was to be called Integration Ensemble. We asked all our customers what the requirements should be. So, of course, it ended up as a long list of everything every group had ever wanted, with little conceptual integrity. In particular, for example, customers insisted that integration ensemble should provide good support for multiple voltages, which were just going mainstream at that time, or they wouldn't even consider it. We specced out such a product and started to build it. With so many features it would take longer to build than customers would want to wait but customers were insistent that anything less than the full product would be of no use. Then these same customers all purchased Silicon Perspective since what they really needed was good placement and fast feedback, which was not at the top of their list. Silicon Perspective did not even support multiple voltage supplies at that point. The end of that story was that Cadence expensively acquired Silicon Perspective and Integration Ensemble was quietly dropped. The customers got what they wanted even though they never asked for it.

One area where marketing is especially easy is when the developer is the customer for the product, when they are "eating their own dogfood" as the saying goes. This is one of the factors driving success in open source software: the developers are usually their own customers. iPhone and other Apple products are also like this; the designers all will use the product. EDA software (and many other products from jet-engines to heart-pacemakers) are generally not like this, so marketing in the sense of product definition is required. Generally in this type of environment open source has been unsuccessful and a lot of the intellectual property (in the most general sense) of the product is not so much in the implementation but in the compromises around what to put in and what to leave out. Doing a good job of

specifying products is one of the hardest parts of marketing, requiring a deep understanding of the customer problems and a deep enough understanding of the technology and engineering involved to deliver a solution.

The art of presentations

As a marketing guy, and even when I was an engineering manager, I make a lot of presentations. I've also been on a couple of presentation courses over the years. The most recently by Nancy Duarte, whose biggest claim to fame is doing Al Gore's slides for his Inconvenient Truth presentation. The most amazing thing about that was not the course itself but the location: a whole building of professional slide designers doing nothing but presentations for large companies for tens of thousands of dollars a time.

Most problems with presentations come about from making the presentation serve too many purposes. They are what will be on the screen for the audience to see, they may be your own way of keeping track of what you need to say, and they may be a handout that is meant to stand on its own for people who missed the presentation. The problem is that the first function, adding to what you are saying, requires different content from the other two, reminding you what to say or serving as a substitute for what you say.

The reality is that your audience can only concentrate on one verbal thing at a time. If you put a lot of text on your slide then your audience will be reading it and not listening to you. You need to decide which is going to win. You cannot have it both ways and make a detailed content-rich speech accompanied by a detailed content-rich presentation. If the content is identical in both places, it is very boring. If it is different, it is very confusing. There are even studies that show that if what you say is all on the slides, then you are better either giving a speech (without slides), or handing out the slides (without saying anything).

The rest of this entry assumes that you are doing the most common form of hi-tech presentation, where a good part of the

content is on the slides. When you deliver it you should emphasize the key points but don't go over every line. Instead, tell anecdotes that back up the dry facts on the screen. Personalize them as much as you can to make them more powerful and memorable. This approach works well for presentations that you are not going to rehearse extensively, or where someone else may be the presenter. If it's not on the slide it doesn't exist.

When putting together a presentation, like any sort of writing, the most important thing is to have a clear idea in your own mind of what you want to say. So the first rule is to write the one slide version of the presentation first. If you can't do this then you haven't decided what point you are trying to make, or what your company's value proposition is, or how to position your product. Until you get this right, your presentation is like a joke where you have forgotten the punch line. Once you have this, then this should be very close to the first slide of your eventual presentation. After all, it is the most important thing so you should open with it; and probably close with it too.

When you have the one slide version worked out you can go to 3 or 4 slides. Get that right before you go to the full-length presentation. When you expand the few points from those few slides to a full-length presentation, make sure that you presentation "tells a story". Like a good story, it should have a theme running through it, not just be a collection of random slides. How many slides? No more than one every 2 minutes max. If you have 20 minutes to speak, 10 slides or so.

In the consulting work I do, I find that not getting these two things right are very common. Presentations where the basic message is not clear, and presentations that do not flow from beginning to end. Not to mention people trying to get through 20 slides in 10 minutes.

If you are presenting to foreigners who don't speak good English, you must make sure that everything important is on the slides since you can assume they will not catch everything that you say (maybe *anything* you say). You will also need to avoid slang that non-Americans might not understand (although you'd be surprised how many baseball analogies Europeans use these days

without knowing what they really mean in a baseball context). I remember the people at a Japanese distributor being confused by "low-hanging fruit." They thought it must have some sort of sexual connotation!

So make sure you know the main point, and make sure that the presentation tells a story that starts from and finishes with the main point.

Oh, and here is another rule of thumb. Print out your slides. Put them on the floor. Stand up. If you can't read them the type is too small. Or go with Guy Kawasaki's rule of using a minimum font size at least half the age of the oldest person in the room.

Swiffering new EDA tools

Why isn't a new EDA tool like Swiffer?

One point that I've made before is that big EDA companies suffer from being unable to get new products into their channel. As I said elsewhere:

> "When so much is riding on just keeping the customer on-board using the existing tools, the salesperson becomes very risk averse about selling new products."

The effect of this is that big EDA companies can only sell to customers once there is market demand. But that is the same problem as Proctor and Gamble faced with, say, Swiffer. Nobody was demanding mop with replaceable sheets, nobody knew one was available. So traditional marketing showed how useful it could be and that it was available at your local supermarket and now Swiffer is on track to be a billion dollar business.

Why can't marketing do much to create demand in EDA? I don't entirely know, but here are some plausible relevant things.

Firstly, the EDA market (for IC design, not for FPGA or embedded software) is inelastic. No matter how much advertising is done, no matter how low the price, no matter how appealing the packaging, the market for EDA tools is fixed. Sure, we can steal market share from each other, maybe we can increase ASPs,

we can expand the definition of EDA. But there is no untapped market of people out there who never knew they wanted to design a chip, in the same way as we all turned out to be a market of people who never knew we needed a post-it note. So we are only marketing to people who already know they are designers.

EDA is not even like other software industries. It values different things because it moves so fast. All users complain, with justification, about the bugginess of EDA software, but they can't get by with the old solid version in the same way as in slower moving software industries. In books like *Crossing the Chasm* and *The Innovator's Dilemma*, marketers are told to worry about the job that the customer hires you to do. The customer doesn't want a drill, they want a hole. The job is holes. But when the EDA engineer goes to Home Depot, he's not looking for ways to make a hole. He's already decided that he wants an 18V cordless drill with two gear ratios. Maybe he'll pick between DeWalt and Bosch but he's not looking at those ads for explosive nail-guns.

Next, the design engineer has been burned before. Because the technology treadmill moves so fast, tools don't always work well (or sometimes at all) but the purchaser doesn't have the luxury of waiting for code to mature, for standards to be in place, for the landscape of winners and losers to be clear. But a lot of IC design is about reducing risk (because we can't just fab the chip repeatedly in the equivalent way to a software engineer compiling and testing the code). One component of risk is using a new tool so there is always a push of potential advantage of the new tool against the pull of potential disaster if it fails. So designers have learned to evaluate new tools in enormous detail, to understand not just what they should do, but what they actually do and how they work internally to do it. Other people don't take the cylinder-head off the engine before buying a car.

Brand name counts for very little in EDA. To the extent it counts for anything in this context, it stands for a large organization of application engineers who can potentially help adoption. It certainly doesn't stand for rock-solid reliability. The speed of development means that every large EDA company has had its share of disastrous releases that didn't work and products that never made it to market. There are no Toyotas and Hondas in

EDA with a reputation for unmatched quality. I don't think anyone knows how it would be possible to create one without it also having a reputation for the unmatched irrelevance of many of its products due to lateness.

So there are a few theories. Like all stories after the fact, they are plausible but it is not clear if they are the real reason. But the facts are clear: traditional marketing, such as advertising, doesn't work for EDA products.

Presentations without bullets

I talked earlier about the typical hi-tech presentation where the content is largely on the slides. In that case you must add color by what you say rather than simply reading what is on the slides.

The alternative approach is essentially to make a speech. The real content is in what you say. The slides then should be graphical backup (pictures, graphs, key points) to what you are saying. Watch a Steve Jobs keynote from MacWorld to see this type of presentation done really well, or presentations from TED (but beware, not all of them have slides at all).

But just like Steve Jobs or the TED presenters, to carry this off well you need to rehearse until you have your speech perfect, either basically memorizing it or doing it from notes. Whatever you do, don't write it out word for word and read it. The slides are not going to help you remember what to say, they are another complication for you to make sure is synchronized with your speech. So rehearse it without the slides until you have that perfect. Then rehearse it with the slides. Then rehearse it some more. Like a good actor, it takes a lot of repetition to make ad libs look so spontaneous.

This approach will not work presenting to foreigners who don't speak fluent English. There is simply not enough context in the visuals alone, and your brain has a hard time processing both visuals and speech in a second language. If you know a foreign language somewhat, but are not bilingual, then watch the news in that language. It is really hard work, and you already know the

basic story since they cover the same news items as the regular network news.

If you are giving a keynote speech, then this is the ideal style to use. You don't, typically, have a strong "demand" like you do when presenting to investors (fund my company) or customers (buy my product). Instead you might want to intrigue the audience, hiding the main point until late in the presentation. So instead of opening with a one-slide version of the whole presentation, you should try and find an interesting hook to get people's interest up. Preferably not that Moore's Law is going to make our lives harder since I think we've all heard that one.

I find the most difficult thing to achieve when giving speeches to large rooms of people is to be relaxed, and be myself. If I'm relaxed then I'm a pretty good speaker. If I'm not relaxed, not so much. Also, my natural speed of speaking is too fast for a public speech, but again if I force myself to slow down it is hard to be myself. This is especially bad if presenting to foreigners since I have to slow down even more.

I also hate speaking from behind a fixed podium. Sometimes you don't get to choose, but when I do I'll always take a wireless lavalier (lapel) mike over anything else, although the best ones are not actually lapel mikes but go over your ear so that the mike comes down the side of your head. That leaves my hands free, which makes my speaking better. Must be some Italian blood somewhere.

Another completely different approach, difficult to carry off, is what has become known as the Lawrence Lessig presentation style, after the Stanford law professor who originated it. Look, for example, for presentations where he talks about copyright and gets through 235 slides in 30 minutes, or watch a great presentation on identity with Dick Hardt using the same approach. Each slide is on the screen for sometimes just fractions of a second, maybe containing just a single word. I've never dared to attempt a presentation like this. The level of preparation and practice seems daunting.

Creating demand in EDA

I talked earlier about how EDA marketing can't create demand. Small companies cannot afford much marketing and large companies are in the vicious cycle of not being able to get innovation into the channel since they can't create demand even though they could have money to spend if it were effective.

EDA used to be rich enough that it would advertise anyway, at least to get the company name out in front of people (remember all those in-flight magazine ads for Cadence and the "curfew key" and suchlike). But as times got tighter, EDA stopped advertising since it was ineffective. In turn, the books that used to cover EDA, like EE Times and EDN, cut back their coverage and laid off their specialist journalists like Richard Goering and Mike Santarini. To be honest, I think for some time before that the major readers of EDA coverage were the other EDA companies, not the customers. I don't have any way to know, but I'm sure the readership of this blog is the same.

Trade shows seem to be a dying breed too, and not just in EDA. DATE seems to be dead, as a tradeshow, with almost no exhibitors any more. I wouldn't be surprised if this year it has almost no visitors any more either, and gives up next year. EDA seems like it can support one real tradeshow, which is DAC. It is mainly for startups for whom it is really the only way to get discovered by customers outside of having a half-reasonable website. The large EDA companies run their own tradeshows in an environment that leverages their costs better than paying a ridiculous rate for floor space, paying rapacious convention center unions to set up the booth, and putting up with whatever restrictions show management has chosen for this year ("you can't put a car on the booth, just because" was one memorable one that I ran into once).

The large EDA companies, with some justification, feel that a big presence at DAC is subsidizing their startup competitors as well as not being the most cost-effective way to reach their customers to show them the portfolio of new products. The best is to avoid the political noise by at least showing up, but the booths with 60

demo suites running continuously with a 600 employee presence are gone.

That leaves websites and search engines as the main way that customer engineers discover what is available. So you'd think that EDA company websites, especially for startups who have no other channels, would be good. But there are very few websites that do a good job of explaining the product line, the focus of the company and so on in a way that is customer-oriented.

If you talk to PR agencies, they'll tell you that the new thing is using social networks and blogging to reach customers. But they don't really seem to know just how that would work. I mean I'm reaching you through a blog because you are reading this. But if every other blog item were a thinly disguised regurgitated press release you'd soon give up reading. But it's not really possible to do anything more technically in-depth. I don't have the knowledge to be the Roger Ebert of EDA even if I had the time to go to all the "screenings". But that leaves the problem that there isn't an easy way to find out what is coming soon to a theatre, sorry, server-farm, near you and whether it is worth the investment of time to take a serious look.

Finger in the nose

It's interesting how certain phrases catch the popular imagination and almost overnight become clichés, appearing in all sorts of writing. The best of these phrases have the twin benefits that they are both memorable and also immediately communicate the point you are making. The first time you come across them, they may even seem brilliant. The thousandth time, rather less so. Do we really need to "rearrange the deck chairs on the Titanic" any more, to imply that we are addressing minor tactical issues while the major strategic issues remain unaddressed? Or rather, we are "ignoring the elephant in the living room."

I don't recall exactly when "change" didn't seem to imply big enough, well, change. So we had to have "sea change" which sounds bigger, even though I'm not sure exactly what a sea change is. It sounds very nautical, and, as the son of a naval officer, I ought to know what it means but I don't and neither

does my father. Wikipedia tells me that it comes from Shakespeare's *Tempest* but that hardly explains the recent change in its popularity. Or should I say sea change in its popularity.

It was George Orwell, in *Politics and the English Language*, an essay that anyone who does any writing should read, who pointed out that most of these clichés are simply ways of avoiding thinking through exactly what we mean.

In Britain, there are a lot of American television shows, so people there get very accustomed to American ways of saying things. The oddest is the way that British people know a lot of American sports terminology used in an everyday sense, without knowing anything about the underlying sports feature that the phrase is meant to conjure up. Some phrases are obvious: “in the ballpark” for example. But British people may well know that something “out of left field” is a surprise, without really knowing where left field is and what might be coming out of it surprisingly. Or know that a “Hail Mary pass” is a last desperate attempt, without ever having seen such a pass on the football field (er, that would be American football to the British, since football is what Americans call soccer). British cricket terminology doesn’t do so well in the opposite direction. Few Americans know what “batting on a sticky wicket” means.

When you get into other countries where English is a foreign language, you need to be very wary of using such imagery. It may simply be unknown even to people who are bilingual, and without sometimes knowing anything about the underlying image being conjured up, not something easy to guess at. I mentioned recently that some Japanese wondered what “low hanging fruit” meant, and guessed at some sort of sexual metaphor. Much better are metaphors that work immediately in any language, like “herding cats.”

When I lived in France, we had great fun translating colloquial phrases word for word from French into English or vice-versa. For example, the French have a phrase “doigt dans le nez” used to imply that something is trivially easy. Literally translated it’s “finger in the nose”, the implication being that it is so easy you could do it with a finger in your nose. I suppose we might say we could do it with one hand behind our backs. Or more fun, we

could say it is a “piece of cake” and translate that word for word as a “tranche de gateau.”

So write something on weird phrases. Finger in the nose!

Corporate CAD cycle

Many things in business go in cycles. One in EDA is what I call the “corporate CAD cycle”. It goes like this. I’m sure a similar dynamic plays out in other industries too.

A large multidivisional semiconductor company has dozens of products in development. They have management who decide that the best way to hold groups responsible is to give them complete control of their destiny. Each group decides what its design methodology is going to be, purchases tools and is generally pretty successful at getting their products out. Life is good. Then one day someone in finance or corporate CAD notices two things: the amount of money that the company is spending on tools is very high, and secondly the various groups have made different tool/methodology decisions and so it is much more difficult than it should be to move around people (need re-training) and pieces of designs (wrong formats, wrong standards). The first phase of the cycle has ended.

A decree goes out. Corporate CAD will make corporate-wide volume tool purchase and decide what should be the standard flow. The standard flow will be mandated throughout the corporation, no exceptions. In a fairly short time the big problems are fixed: the tool budget is slashed now that experienced purchasing agents are negotiating very large deals with just a few EDA vendors; moving people and blocks around is simpler since everyone has the same base knowledge. Life is good. The second phase of the cycle has ended.

But management notices that many product groups are being a lot less successful at getting their products out than they used to be. This is a huge multi-million dollar problem. Management takes a look at the most important bet-the-company product. The product group says they are forced to use the wrong tools, that they are spending too much time getting blocks into corporate formats

that they don't use themselves, that they have too few licenses. But this chip is critical, so just this once, since the group is really expert, they are allowed to buy whatever tools they need to get the job done. Of course when corporate CAD said there were no exceptions to the standard flow, they weren't that stupid. Of course there were exceptions for RF. And the advanced group over there that is doing a pilot project in 45nm obviously needs some tools that are different. Oh, and we just acquired a little fabless semiconductor company who uses something non-standard, we'd better let them at least get their chip out before we force them to use the standard flow, it's hard enough for them switching to our process. Gradually the iron grip of corporate CAD relaxes. Corporate CAD makes big purchases, but a lot of those tools are sitting on shelves. Design groups are largely making their own decisions about tools. Chips are coming out successfully again. Life is good. Then someone in finance…

This cycle seems to re-play itself in many areas where two contradictory goals collide. In the CAD case above it is the need to have a standard flow, but also allow exceptions when the standard flow is not enough. Of course I exaggerated the story a bit, but the basic cycle between giving too much freedom and not enough seems to be real and I've seen a version of the cycle play out in many semiconductor companies.

One other area with this sort of cycle is whether to manage functionally or along product lines. Should a large semiconductor or EDA company (or anyone else for that matter) run engineering as an organization, and then have project managers for each product? Or should a product be responsible for all its own groups? Engineering, marketing, finance (probably not), sales (maybe). There are problems with both structures. If product groups are self-contained (even if they don't have dedicated salesforces) then there is no control of corporate brand image, little standardization of development processes and so on. At once point when I was in Cadence years ago, we had 6 or 7 full-page ads in EE times from different groups, all product-oriented but with a different look and feel and brand image. However, if engineering, marketing and so on are all functionally managed then they can be very unresponsive to urgent needs in the product groups.

The way this seems to play out is to be functionally organized for a few years, then when that seems ossified, become structured into business units or product groups for a few years. One reason that I think that this can actually be effective, not just management churning, is that the relationships from the previous era endure, at least for a couple of years. If you were previously in marketing for a product group, and now you are in the company-wide marketing organization, you still know all the engineers that work on the products you care most about; after all, only a year ago you were all in the same organization. When eventually you get blown part back into product groups, you still know everyone in marketing, you know in your bones the corporate look-and-feel, branding and so on, and it takes a couple of years to gradually forget (as an organization, with people coming and going, more than individuals actually forgetting).

So there really is something to the old management canard that, when you don't know what to do, centralize everything that is distributed and distribute everything that is centralized.

Licensed to bill

As I've said before, in every sizeable EDA company that I've worked, a huge percentage, 30-50%, of all calls to the support hotline are to do with license keys. Why is this so complicated? Are EDA software engineers incompetent?

Most of these problems are not directly with the license key manager (the most common, almost universal, one is FlexLM). Sometimes there are direct issues because customers want to run a single license server for all the EDA tools they have from all their vendors, something that the individual EDA companies have a hard time testing since they don't have access to everyone else's tools. More often license problems are simply because licenses are much more complicated than most people realize.

All sorts of license problems can occur, but here is a typical one. The customer wants some capability and discusses with the salesperson who provides a quote for a particular configuration. Eventually an order gets placed and a license key is cut for that configuration. At this point, and only then, it turns out that the

configuration doesn't actually deliver the capability that the customer thought he'd asked for, and that the salesperson thought she'd provided. Something is missing. The customer calls support to either to report a bug or, if they realize what is going on, to try and get the specific license added. Often an option has been omitted from the configuration (such as a special parser) that everyone assumed was included, or assumed that it wasn't needed, or that turned out to be bundled with some other capability in a mysterious way.

Digital Equipment, in the heyday of the Vax, actually had an AI program XCON salespeople had to use to configure Vax computers since otherwise they always had similar problems, although in the hardware domain. The order omitted a required cable, or overloaded a power supply or left out a software driver. Without this error being corrected, the delivered system could not be assembled in a way that would run. This is worse still in the hardware world since it takes from a couple of days to a couple of weeks to get a missing cable to the customer site. It can't simply be fixed over the phone.

The fundamental problem is that it is hard to map capabilities that marketing wants to sell and price, into the actual control points in the software that permit or deny certain activities, and the ways in which the different components interact. Few people have a good understanding of this, and there is no correct answer to many of the questions.

Here's an example. Should a long-running tool claim a license when it starts for an optional feature that might be required later? Or should it wait until it has run for hours and then fail if a license is not then available? Which inconveniences the user less? There are pressures on the vendor side to want to claim licenses as early as possible (so the customer needs to buy more licenses) which at least means that if a tool is going to fail due to lack of licenses, it does so immediately without having done a lot of wasted work, and in a part of the code where it is easy to handle. There are pressures from the customer side to want to claim licenses as late as possible (so they don't get held for long periods when they are not being truly used) but also to expect that the tool will behave gracefully when their paucity of licenses

comes to light and the run is deep in the innards of the tool when it finds out it cannot continue.

Interactive tools are worse still. Do you claim a license in order to show the capability on a menu? Or do you show a menu item that may fail due to lack of a license when you click it? Do you behave the same if the customer has licenses but all are currently in use, versus the customer not having any licenses to that product at all?

None of these problems typically affect the engineers developing the product or their AEs. Usually all employees have a "run anything" license. The licenses issues often only come to light when customers run into problems. After all, they may be the only site in the world running that particular configuration. Some testing can be done easily, but exhaustive testing is obviously impossible.

EDA companies want to create incremental revenue for new capabilities, so they don't want to simply give them to all existing customers even though they may want to make sure that all new customers are "up to date." This drives an explosion of license options that sometimes interact in ways that nobody has thought of.

Until some poor engineer, in the middle of the night, tries to simulate a design containing two ARM processors. That's when they discover that nobody thought about whether two ARM simulations should require two licenses or one. The code claims another license every time an ARM model is loaded, in effect it says two. Marketing hadn't considered the issue. Sales assured the customer that one license would be enough without asking anyone. Nobody had ever tried it before. "Hello, support?"

DAC

Xxx

The design automation conference (DAC) is later this month in San Francisco. Trade shows in general are probably gradually dying. I doubt we'll be going to them in ten years time. But

rumors of their death are somewhat exaggerated. DAC will probably be in San Francisco longer than the Chronicle.

Marketing in EDA these days is very difficult since the channels don't exist in the way that they used to. Both EETimes and EDN have laid off their seasoned EDA journalists (Richard Goering from EETimes, now at Cadence, and Mike Santarini from EDN, now at Xilinx). The big EDA companies stopped advertising, which meant that the books couldn't financially justify covering the industry. This was an unwanted side-effect of what was probably a reasonable decision. It's never been clear whether advertising in the print edition was ever a good financial decision, but as more and more eyeballs went online it certainly got worse. Online advertising rates are just not as high, and there is a limit to how much a publication can annoy its customers with flashing adds, peel-back corners and stealing the screen for an enforced video.

So DAC is left standing as really the only marketing channel that works. It works in the sense that the major decision-makers from the EDA customers all come to DAC. DATE will presumably continue as a great conference but I doubt its tradeshow will recover as a must-attend event (and since it costs about the same as attending DAC, even a European EDA company will go to DAC if it can do only one). Japan still has its local shows too.

The big EDA companies are all going this year. I think it is foolish when they don't attend. This is partially because they need to be seen to be good corporate citizens and not attending is unnecessary insulting to everyone else in the industry. They also tend to generate unnecessary bad publicity if they stay away. However, it is simply uneconomic for them to come in the strength that they used to (when I was at Cadence we'd have 5-600 people at DAC, running 50 demo suites). The "tax" from DAC itself, the conference center, the unions and everyone makes it a lot more expensive than running their own one-company shows.

Clearly, for the small companies, DAC is their one opportunity to get noticed other than people falling onto their website from a search engine. Not attending is tantamount to admitting you either no longer exist or are about to die. Envis, where I was

recently interim CEO, has apparently pulled out of DAC. I don't have an inside scoop on what it means but it doesn't seem like it would be good.

If you work in marketing, DAC gives an interesting insight into the problems of GM or United Airlines. It is the one time when us silicon valley types have to get involved with union work rules. It costs more to get your equipment from the loading dock in the conference center to your booth space, than it does to ship it there (and this is true in Las Vegas or Annaheim, not just when DAC is local in San Francisco). You have to use a certain number of hours of labor whether you need it or not. You have to pay hundreds of dollars for someone to vacuum your carpet each day, since you are not allowed to do it yourself.

I said above that all the major decision makers from EDA customers are at DAC. Depending on your definition of "decision maker" that is anywhere from a few dozen to a few hundred people. The whole of DAC is really for them. Whether the junior engineers and academics show up simply doesn't matter that much (for the tradeshow, the accompanying conference would be nothing without academic participation).

The Denali party

As everyone in EDA knows, Denali has thrown a party every DAC for what seems like forever.

I had lunch last week with Mark Gogolowski and I asked him how the party came about. It started 11 years ago in 1999 at DAC in New Orleans. Denali wanted to have a party for their customers, but they faced a couple of constraints. They couldn't compete with the big Cadence and Synopsys parties of that era, but on the other hand they knew that parties weren't much fun unless they felt crowded. So they'd better invite more than just their (few) customers, especially since they needed to partner with all the simulation vendors, which meant all the big guys anyway. So invite everyone. Denali was under 10 employees in this era, not well-known, so they were more worried about holding a party and nobody coming than the opposite. But never underestimate the gravitational attraction of an open bar.

They expected about 100, maybe 150 people, would attend. One thing that they hadn't anticipated was that the AEs from the big guys weren't able to get into their own parties (the execs and sales guys went with their customers; AEs need not apply). So they showed up in large numbers. In the end well over 500 people came for at least some of the evening. At midnight the venue management told them they had to stop the party since the entire night's alcohol budget was already gone. So they gulped, wrote a large check, and kept the party going for another hour. Shutting down a party as early as midnight in New Orleans and throwing their customers out didn't laissez les bons temps roulez.

They realized that the party had been something special, and not just for their customers. The entire EDA community had shown up since Denali was neutral ground. Nobody from Cadence went to the Synopsys party and vice versa. But Denali, as the Switzerland of EDA, welcomed everyone. So next year, it seemed like it would be a good idea to do it again. And so it has been for many years.

I think it has turned out, somewhat fortuitously, to have been a great way to market themselves. We are in an era when it is really hard to get your name out in front of customers and partners. Denali doesn't have that problem, plus it has a lot of goodwill from the entire EDA community since the Denali party isn't exclusive. You don't have to be a customer of Denali to get in; you can even be a competitor.

EDA idol is back again this year, along with a new "Community Superhero" contest. Another new thing this year is that they will be presenting an award for "EDA's Next Top Blogger." Of course, I have my own idea of who that should be. When I know how you can vote I'll let you know!

So here we are a decade later. Everyone knows who Denali is, and they are a much bigger company now. They are still private, so just how big is largely a guess. But nobody cares about their revenue, the financial answer everyone wants to know is "how much does the Denali party cost?" I slipped a shot of vodka into Mark's Diet Coke but he still wasn't talking.

Value propositions

I spent some time earlier this week giving someone a bit of free consulting about value propositions in EDA. If you take the high-level view then there seem to be three main value propositions in EDA: optimization, productivity and price.

Optimization means that your tool produces a better result than alternatives. A place and route tool that produces smaller designs. A synthesis tool that produces less negative slack. A power-reduction tool that reduces power. This is the most compelling value proposition you can have since the result from using your tool as opposed to sticking with the status quo shows through in the final chip affecting its price, performance or power. The higher the volume the chip is expected to run at, the higher the value of optimizing it.

Productivity means that your tool produces an equivalent result to the alternatives but does it in less time. My experience is that this is an incredibly difficult value proposition to sell unless the productivity difference is so large that it is a qualitative change: 10X not just 50% better. Users are risk-averse and just won't move if they have "predictable pain." It may take an extra week or an extra engineer, but it is predictable and the problem is understood and well-controlled. A new tool might fail, causing unpredictable pain, and so the productivity gain needs to be enormous to get interest. Otherwise the least risky approach is to spend the extra money on schedule or manpower to buy predictability.

The third value proposition is that you get the same result in the same time but the tool is cheaper. For something mission-critical this is just not a very interesting value proposition, sort of like being a discount heart surgeon. Only for very mature product spaces where testing is easy is price really a driver: Verilog simulation for example. The only product I can think of that strongly used price as its competitive edge was the original

ModelSim VHDL simulator, and even then it was probably simply the best simulator and the low price simply left money on the table.

Another dimension of value proposition is whether the tool is must-have or nice-to-have. By must-have I don't mean that customers must buy your tool (nice work if you can get it) but that they must buy either from you or one of your competitors or roll their own. Nice-to-have means that a chip can be designed without a tool in that space, doing stuff by hand, creating custom scripts, having a longer schedule or whatever. It is almost impossible to build a big business on a nice-to-have tool.

Moore's law makes must-have a moving target. Signal integrity analysis ten years ago was, perhaps, nice-to-have. Then for designers in leading edge processes it became must-have. Eventually the technology got rolled into place and route tools since everybody needed it.

That is actually a fairly typical route for technology. Some new wrinkle comes on the scene and somebody creates a verification tool to detect the handful of fatal wrinkles that can then be fixed by hand. A couple of process generations later, there are 100,000 fatal wrinkles being detected and so it is no longer adequate to have just a verification tool. It becomes necessary to build at least some wrinkle avoidance into the creation tools so that fatal wrinkles are not created, or are only created in manageable numbers again. So the tool goes from nice-to-have, to must-have to incorporated into the main flow.

Being too early to market

Startups have a singular focus on getting their product to market as quickly as possible. Given that focus, you'd think that the primary mode of failure for a startup would be being too late to market, but it's actually hard to think of startups that fail by being too late. Some startups fail because they never manage to get a product shipped at all, which I suppose is a sort of special case of being too late to market: you can't be later than never. But try and think of a startup that failed because, by the time it got to market, a competitor had already vacuumed up all the

opportunities. Monterey in place and route, I suppose, simply too far behind Magma and the big guys re-tooling.

On the other hand, many startups fail because they are too early to market. In EDA, technologies tend to be targeted at certain process nodes which we can see coming down the track. There's little upside in developing technologies to retrofit old design methodologies that, by definition, already work. Instead, the EDA startup typically takes the Wayne Gretsky approach of going where the puck is going to be. Develop a technology that is going to be needed and wait for Moore's law to progress so that the world does need it. The trouble with this is that it often underestimates the amount of mileage that can be got out of the old technologies.

Since process nodes come along every couple of years, and even that is slowing, getting the node wrong can be fatal. If you develop a technology that you believe everyone needs at 45nm but it turns out not to be needed until 30nm then you are going to need an extra two years of money. And even then, it may turn out not to be really compelling until that 22nm node, after you've gone out of business. All the OPC (optical proximity correction) companies were too early to market, supplying technology that would be needed but wasn't at that point in time. Even companies that had good exits, like Clearshape, were basically running out of runway since they were a process generation ahead of when their technology became essential.

The windows paradigm was really developed at Xerox PARC (yes, Doug Englebart at SRI had a part to play too). Xerox is often criticised for not commercializing this but in fact they did try. They had a computer, the Xerox Star, with all that good stuff in. But it was way too expensive and failed because it was too early. The next attempt was Apple. Not Macintosh, Lisa (pictured above). It failed. Too early and so too expensive. One can argue the extent to which the first Macs were too early, appealing only to hobbyists at first until the laser printer (also invented at PARC) came along. There are other dynamics in play than just timing but Microsoft clearly made the most money out of commercializing those Xerox ideas, coming along after everyone else.

Another means of being too early is simply having an initial product that it turns out nobody needs yet because it's not good enough yet. Semiconductor development processes are all about risk-aversion, and any change has to mean that the risk of changing is less than the risk of not changing. For a startup with an early product in a process generation where the technology might be only nice-to-have this is a high barrier to cross. The startup might just serve as a wakeup call to everyone else that a product is required in the space, and eventually another startup executes better (having seen the first company fail) or the big EDA companies copy the technology into their own product line.

Overall, I think more startups fail by being too early to market than fail by being too late. Remember, it's the second mouse that gets the cheese.

Barriers to entry

When I looked around at DAC last month (well, the month before last, what happened to August?) one thing that is in some ways surprising is that, given the poor growth prospects of the EDA industry, there are so many small EDA companies.

If you are a technologist of some sort then it seems like the challenge of getting an EDA company going is insurmountable. After all, there are probably only a couple of dozen people in the world who have deep enough knowledge of the esoteric area of design or semiconductor to be able to create an effective product. That seems like it should count as a high barrier.

But, in fact, technology is the lowest of barriers if you are in a market where technology counts for something. Designing and building chips is something that races along at such a breakneck pace that the whole design ecosystem is disrupted every few years and new technology is required. It has to come from somewhere. As a result, brand-name counts for very little and small companies with differentiated important technology can be successful very quickly.

Other industries are not like that nowhere else does technology move so fast. What was the last big innovation in automotive?

Probably hybrid powertrains. Most cars still don't have them and it is now ten year old technology.

Let's think of an industry with just about the least amount of technology, so pretty much at the other end of the scale from EDA and semiconductor: bottled water. Do you think that your bottled water startup is going to do well because you have better water technology? Do you think that the customer who chose Perrier rather than Calistoga could actually taste the difference anyway? Bottled water is selling some sort of emotional aspirational dream.

You've obviously noticed that if you go to bar and get upscale water then you typically end up with something from Europe (San Pellegrino, Perrier, Evian) and not something from California (Crystal Geyser, Calistoga). It has to be bottled in Europe and shipped here. Why don't they ship it in bulk and bottle it here? For the same reason as wine is bottled before it is shipped: nobody would trust what was in the bottle. One thing that surprised me when I was in Japan a couple of years ago is that the Crystal Geyser water we turn down as being insufficiently upscale is what they drink over there. It comes from California, the other side of the Pacific, how exotic is that? I don't know if the third leg of the stool exists, people in Europe drinking water from Asia: bottled from a spring on Mount Fuji, how zen is that?.

In between are lots of companies and industries where there is obviously a technical component, and an emotional component. BMW may be the ultimate driving machine, but most people who buy one couldn't tell you what a brake-horsepower is, even if they know how many their car has. And almost nobody actually uses all that horsepower, running their car at the redline on the tacho all the time. Yes, there's technology but mostly it's an emotional sell.

In the commercial world, think of Oracle. Do you think you are going to displace Oracle because you're little startup has some superior relational database technology? No, there's a whole ecosystem around Oracle, they largely sell to people who don't understand technology (CFOs) and so brand-name counts for

something. They are partially making an emotional decision and buying peace of mind.

Brand name still counts for a lot in the consumer market space, even if less than it used to. This is measured by the increase in price for the brand name that consumers will pay compared to the no-name. Many of the top brand names in the world (Coca-Cola, Kellogs, Colgate) are very old going back a century or so but the premium, especially in the current downturn, that people will pay to get a Sony rather than Best Buy own-brand is shrinking.

So brand name or ecosystem are really high barriers to entry. Technology not much. A few smart guys and a two or three years of writing code is a lot easier than recreating the ecosystem around ARM, never mind making your cola as well known as Coke.

Chapter 4: Engineering

Where is all open source software?

There is a big cultural difference between tools for IC design and tools for software design. A difference in the way they are developed, the way they are sold, the way they are deployed. I think there are two reasons.

Firstly, IC designers are not software designers (duh!) and so are not generally capable of writing or extensively modifying their own tools, so it doesn't cross their mind that it might be a good use of their time to do so. After all, it wouldn't be.

Secondly, the culture has arisen that the studliest IC designer (or whatever the equivalent is for a woman) has the most money spent on their tools. Companies capitalize IC designers with large amounts of software to increase their productivity. But it is not really productivity, it is the ability to get the job done at all. Only in the most technical sense is improving the time taken to design a chip from a millennium to six months merely a productivity increase. The EDA industry has improved engineering productivity by a factor of maybe 100,000 in the last thirty years.

Neither of these are true in the world of tools for software engineers. Firstly, they *are* software engineers and so can develop software fast by either just going and developing it on whatever is available (without requiring centuries) or they can write themselves some productivity tools and then use those tools to produce their product with higher efficiency. They are their own customers in that sense.

Open source development has also been shown pretty conclusively to be more efficient the closed source. Eric Raymond's book "*The Cathedral and the Bazaar*" is a little dated but still the best survey of this. But there is a problem. Open source software is also known as "free software". Originally this meant "free as in freedom not free as in beer" but has come to mean "free as in beer" too. Or at least very cheap.

Open source means that if you buy the product, and perhaps even if you don't, you also get the source code and can do what you want with it. Of course, since you can do what you want, it becomes really hard to sell a second copy since you can always build that yourself from the source, so the second copy had better be really cheap or that's what you'll do. For example, Wind River used to have a proprietary royalty-bearing operating system called VxWorks but the world is going to Linux and so Wind River supplies their own version of Linux. But it is hard to charge much and impossible to charge a royalty on this since a customer can get Linux from many places and even hire a few engineers and build their own version. Or a competitive company can take the same source and customize their own product. It is hard to see what Sun has got from Java to justify its investment, and even harder to see why it recently bought MySQL for about a billion dollars when the price for a copy of MySQL is exactly zero.

In that way, open source is a bit like Craigslist. Craigslist didn't steal all the money from newspaper classified advertising. It took a billion dollar business and made it into a million dollar business, making it impossible for anybody, even Craigslist themselves, to make real money in classified advertising.

So the result of all of this in the world of tools for software development is that all the best tools are open source, but nobody can make any money selling them. This works fine as long as enough people like Sun and IBM pay their developers to do open source development on the basis they make money on the hardware, or enough programmers do this in their spare time for fun (and because if you want a job at somewhere like Google, one of the things they'll take a look at is what open source projects you work on after hours). But there are no Microsofts of open source, no Oracles nor Adobes. Not even Intuits or Mathworks.

IC design tools are all closed source, apart from a few bits of infrastructure like openAccess. Synopsys isn't about to give you the source code for Design Compiler just because you bought a license, and they certainly aren't going to put it up on the web so Cadence can grab themselves a copy too. It is arguable whether the quality would even improve that much if they did so since

most of the users are not itching to get into the millions of lines of source code and add a few enhancements.

Now that electronic systems contain large amounts of software content, these two worlds are starting to collide a bit. The investment for a system design team in tools for the IC part is maybe in the millions of dollars, and for the software part it is essentially zero. And not because they are forgoing the best tools because they are too cheap; those free tools *are* the best tools.

The number of chip designers is fairly small, perhaps 20,000 engineers. There are ten million software engineers. The price of EDA tools has to be high to recover the development cost over a small base; software tools have a much larger base. After all, Bill Gates got rich selling copies of MS-DOS and Windows for less than $100 each. But he sold a lot.

Open source again

The blog entry on open source seems to have generated more comments than anything else. Maybe it's because all the EDA users want software to be free, and all the EDA producers are worried that it might head in that direction. Everyone has an opinion.

In a seemingly off-topic thought, let me recommend Econtalk which is a weekly interview by Russ Roberts (a prof at GMU and Stanford) with someone knowledgeable on some aspect of economics interpreted in a wide sense. Last week it was Keynesian economics and the week before it was building schools in Africa. I typically listen while commuting.

One week these two disconnected items tie together since Russ's guest is Eric Raymond talking about open source and the nature of the open source process. Recommended.

One thing about open source that I think people misunderstood is that I was not predicting that there would or should be open source EDA tools, or that the market was not big enough. I think open source is successful when the programmer and the user are the same person so there is no need to try and reduce the

requirements to a specification. Or where the project is an open source copy such as creating open source Flash, or even Linux (open source Unix), so that the original serves as the specification. I've even seen people claim that if you need a specification the project is already off the rails. It is really hard to write good software for an application that you don't understand well yourself, where you are not going to be your own user. EDA software is largely like that. Designers are not (good) programmers and programmers know scarily little about chip design

There seems to be a similar dynamic about many websites: Facebook, eBay, Yahoo, mySpace and many others were created to serve a need that the founders felt they needed filled for themselves, and then were smart enough to seize the moment. On the other hand I think there is lots of opportunity on the net for sites serving older people. The people who found web companies are young and as a result older people are underserved. But old people are on the web, they have money, they have time and they are a fast growing demographic. What's not to like? Like in the open source case, the people who create such companies and write the code are unlikely themselves to be in their 60s and 70s so creating something successful is much harder.

Why is EDA so buggy?

I have sat through numerous keynote speeches by CTOs of semiconductor companies berating the EDA industry for shipping tools that are full of bugs and that are late, not ready enough in advance of the appropriate process node. Of course this is true, and nothing I am going to say is to imply that improvement is impossible. But it is an intrinsic problem, not just laziness or incompetence on the part of EDA vendors.

In an informal setting, that is to say over a beer rather than in front of a large audience, I tell such CTOs that if they want more reliable software then they can simply use an old version of the tools that has been shipping for years. Tools like that get pretty solid. Of course that is simply a glib response since I know that they can't design 65nm designs with 130nm tools.

So my next suggestion is that the EDA industry could delay shipping new tools for an extra year or so to allow extensive testing. Another glib response since we all know that the tools are already late and even if it were possible to test them extensively without shipping them (what would we use for test data?) the delay would be unacceptable.

There are two reasons for this state of affairs, one technical and one economic.

The technical reason is that it simply isn't possible to know everything necessary about a new process node far enough ahead of time to allow for a robust development cycle. For example, even with a huge team of people ready to go it is impossible to develop a full suite of technology for 22nm design. We just don't know everything we need to know to get started and neither do the semiconductor companies and their equipment suppliers. Inevitably the tools will be later than desired and customers would rather have buggy tools now than better tools in six months. It is the EDA version of MacArthur's dictum that a good plan violently executed now is far better than a pefect plan executed next week. In fact, whatever schedule is chosen, there are always customers lining up to be beta sites, so that they can get their hands on technology earlier, and pressure to ship even earlier even though the tools will be buggier still.

There is also the fact that it is not really possible to develop an EDA tool in a vacuum. There need to be libraries and designs in the process node to be used to test and wring out the code. A new tool is often rock solid on old designs, it is the new bigger more complex designs that break the tools in new ways.

The economic argument is that EDA has to support several process nodes at once and recoup its investment in a timely manner. Those same CTOs who berate EDA companies for not being aggressive enough, work at the same companies that have CAD managers who insist that resources be diverted to back-patching bugs. They want fixes that are already available added back into obsolete (and no longer officially maintained) versions of the software because their design groups haven't got round to switching. And those CTOs have finance managers who don't

want to increase their budget for leading edge tools only used by the small number of advanced groups.

The effect of all of this is that EDA companies make their money and recoup their investment on processes only when they become mainstream. They cannot afford to make that investment too far ahead of the mainstream for economic reasons as well as technical ones. The fact that the mainstreaming of the most advanced processes is slowing is already starting to strain this model somewhat since it delays the time to payback.

One way to improve the quality of EDA software would be open source. But if undertaken by the major suppliers, this would also destroy their business model and probably thus result in no software at all. However, EDA moves too fast for open source to simply clone the successful products in parallel. The Econtalk podcast with Eric Raymond pointed out another industry that moves fast and where open source has little impact: games. By the time a game is clearly a hit it is too late to start an open source project to clone it. The gaming community will have got bored with that game and moved onto something else by the time the open source free version is complete and widely available.

Other industries don't have this problem because they don't move so fast. Technologies in automotive are adopted in decades not a year or two. CAD for automotive has a lot of time to adapt. But EDA is stuck with a very short reaction cycle and even if the ROI was richer, it is not clear that much would change.

Groundhog Day

You've probably seen the movie *Groundhog Day* in which the Bill Murray self-centered weatherman character is stuck in a time warp, waking up every morning to exactly the same day until, after re-examining his life, he doesn't. Taping out a chip seems to be a bit like that, iterating trying to simultaneously meet budgets in a number of dimensions: area, timing and power. And, of course, schedule. Eventually, the cycle is broken and the chip tapes out.

There is a lot of iteration in chip design. The goal of EDA is to move as much of the iteration under the hood as possible. It is obviously not possible to manually try the type of iteration the synthesis or place and route tools do millions of times as they produce their results. To the user this seems like a linear process: read in the design, churn away, and write out the answer.

But when the user doesn't get the result they wanted the Groundhog Day feeling begins. EDA tools are not all that easy to drive and most of the controls are somewhat indirect. Years ago I once drove an old car with a steering wheel mounted manual gear-shift. The linkage necessary to make the lever actually engage the shafts in the gearbox probably worked OK when the car had been new, but by the time I got to try, it was a secret art to move the lever just the right way to engage the gears. Controlling an EDA tool is like that only harder. There are many parameters with very poorly defined results that are not even really understood by the programmers who added them. Some of them are even documented!

The internal iterations of EDA tools are inevitably hard to control. The algorithms are all exponential and so rules-of-thumb need to be used to make them terminate at all. One complex algorithm really needs another inside its inner loops since a good placement is onethat routes well. But there obviously isn't time to try millions of routes while finalizing a placement. And a good route is one that doesn't cause timing problems, or crosstalk problems, or create features that can't be manufactured and so on. It is amazing that anything works at all.

Each process node the problem seems to get harder since we add a new wrinkle. We used to have simple timing models that didn't even worry about resistance or signal slew rate. We didn't have to worry about crosstalk. We didn't worry about power. There was no need for resolution enhancement technologies since we were using light with a shorter wavelength than the feature sizes.

The latest Groundhog Day wrinkle is process variability along with the sheer difficulty of closing so many budgets simultaneously. The black-belt groups whose job is to get chips out where other groups are struggling are finding that they have

to do more manual intervention than they are used to: more floorplanning, manual placement, structured placement of datapaths and so on. This seems to be the way of the future. There is so much knowledge about the design that is needed for success than is captured through the design process. With that knowledge designers can find out whether it is still Groundhog Day or whether it is finally February 3rd and time to tape out.

Power is the new timing

In the 1980s, chip design was focused on layout: cramming all those gates into as few chips as possible, trying make use of every square millimeter of silicon. The 1990s were the decade of timing, when all the tools became timing driven with a completely synchronous design methodology. Of course area was still important but the biggest headache for designers was closing timing. The 2000s seem to be the decade of power, where the biggest headache is now meeting the power budget.

In the past, each process generation was accompanied by a reduction in power supply voltage so that it was possible to push up the frequency. Especially since voltage is squared in the power equation. However, that game has come to an end since reducing the voltage takes it too close to the threshold voltage and transistors will not turn off properly. That is why, in particular, microprocessors have gone multi-core rather than having 10GHz frequencies. Their power density would be the same as in the core of a nuclear reactor, not too suitable for a server never mind a laptop.

Later, I'll summarize the techniques available for power reduction. Having recently been interim CEO of a startup company in the power reduction business, I know a lot more than I used to. But a fundamental problem is that almost any technique requires changes to a large number of tools. For example, if the chip has two power supply voltages, a gate may have two different performances depending on which block it is used in. The simulator needs to know that to get the timing right. But V_{dd} and V_{ss} don't occur explicitly in the netlist. This is mainly for historical reasons since they didn't occur explicitly in schematics

either. Besides, back then there was only one of each so there wasn't the possibility for ambiguity.

The CPF and UPF standards were the most recent EDA standard war. It looks like another Verilog/VHDL standoff where both sides sort of win, and tools will need to be agnostic and support both. Both standards are really a way of documenting power intent for the techniques for power reduction that advanced design groups have struggled to do manually. CPF (common power format, but think of the C as Cadence, although it is officially under SI2 now) seems slightly more powerful than UPF (universal power format, but think of the universal as Synopsys, Magma and Mentor, although it is officially under Accelera now and is on track to becoming an IEEE standard P1801). CPF and UPF attempt to separate the power architecture from everything else so that changes can be made without requiring, in particular, changes to the RTL.

Both standards do a lot of additional detailed housekeeping, but one important thing that they do is to define for each group of gates which power supply they are attached to so that all tools can pick the correct performance, hook up the correct wires, select the right library elements during synthesis, know when a block is turned off and so on.

The detailed housekeeping that the standard formats take care of acknowledge that the netlist is not independent of the power architecture. If two blocks are attached to power supplies with different voltages, then any signals between the two blocks need to go through level shifters to ensure that signals switch properly. But they don't appear explicitly in the netlist. Since those level shifters will eventually be inserted at place and route, any earlier tools that analyze the netlist need to consider them too or they will be confused.

If a block is powered down, then output signals need to be tied to either V_{dd} or V_{ss} since otherwise they will drift to an intermediate value creating a partially active path from V_{dd} to V_{ss} through both the P and N transistors of gates in the fanout. This will dissipate power: not good. But again, these cells, which don't appear in the netlist, will eventually be inserted and so will affect timing. During powerdown, it is also possible that some register values

need to be preserved, meaning that special retention registers that take a third always-on power supply must be used.

The purpose of the CPF and UPF formats is to make it explicit what these changes to the netlist are so that all tools in the flow make the same decision and are not surprised to find, say, an isolation cell in the layout that doesn't correspond to anything in the input netlist. Or, indeed, an isolation cell missing in the layout, which should have been inserted despite the fact that it doesn't appear in the input netlist either.

You can learn a lot about low-power techniques by reading the tutorial documents and presentations on the various websites associated with these two important standards.

Power again

Earlier I promised an overview of what power reduction techniques are out there. First, a disclosure: I was interim CEO of Envis for about a year and I've done some consulting for Nanochronous.

Firstly, there are two kinds of power: dynamic and static. Dynamic power is used in switching signals inside the circuit. It is affected by operating frequency and voltage. Static power is dissipated whether the circuit is doing anything or not, and mostly is leakage power through transistors that are supposedly off but in fact leak a little current. This was not a problem above 100nm or so, but below transistors are not so much on and off, as bright and dim.

The most common way to control leakage is to use special libraries that have two versions of each gate (or most gates). One is slow but has low leakage. One is fast but leaks since it never truly turns completely off. On the critical path the fast leaky gates are used; off the critical path the non-leaky slow gates are used. Synthesis tools will choose the cells automatically based on the timing constraints.

Taking this technique a little further was Blaze DFM whose tool would make tiny adjustments to the mask data for transistors off

the critical path, lowering their performance but making them leak a lot less. TSMC licensed this technology and Tela announced recently that it was acquiring them.

The most common dynamic technique is clock gating. The old rules used to be to do purely synchronous design, and clock every flop on every clock cycle. If a register was only loaded with a new value sometimes, then a multiplexor was added to recirculate the old value back to the input so that when the flop was clocked it would re-latch the same value as it was already holding. The simplest form of clock gating is to replace those multiplexors with a clock gating element (CGE) that inhibits clocking the flop when the value doesn't change. This doesn't win you anything on a single flop, but if it is, for example, a 32-bit register then 32 muxes can be replaced with a single CGE saving on area, and, because the effective clock rate of the register is reduced, power. By clever circuit analysis it is possible to find more complex circumstances under which registers can be suppressed either combinationally (the value really wasn't going to change) or sequentially (the value might bave been going to change but no output from the circuit would noticed the change). All the synthesis tools, most notably Synopsys Power Compiler, do the mux replacement. Calypto automates the more extensive gating approaches.

Next there is a whole spectrum of techniques that depend on voltage islands. A voltage island is an area of the chip with its own power supply. Obviously this has a major impact on physical design since the island must correspond to a particular region of the die. The first thing that can be done with voltage islands is simply to power them with different supply voltages. Those on a lower voltage will have lower performance, of course, but they will also consume lower power, both static and dynamic.

Power down is another common technique. Voltage islands which are not being used are turned off completely by turning off their power supply. When you are not making a call on your cell-phone, the gates used to process transmit and receive data are not required and can be turned off. This needs to be done carefully, or else the current inrush when the island is turned back on can cause the voltage to drop elsewhere on the chip. Typically this

means that the island must be powered up slowly using small transistors and then finally brought up to operational level by turning on much larger transistors. Powering down blocks is always done under software control but the powered down block needs to be isolated from the rest of the circuit so that its output signals do not drift and cause crowbar current and waste power elsewhere. There are no tools for automatically finding areas to power down. The software not the netlist would be the place to look. The CPF and UPF formats have extensive support for power down.

As we get deeper below 100nm, the variability of processes gets much wider. This means that the typical chip and the worst case chip are getting further and further apart and so the penalty of designing to worst case design, given that most chips are typical by definition, gets larger and larger. Adaptive voltage scaling is a way to handle this. Use on-chip circuitry to measure the actual performance, and then lower the voltage (saving both dynamic and static power) just the right amount that the chip still runs at the correct speed.

One adaptive solution involving off-chip voltage regulators is National Powerwise. They have put this in the public domain since they make their money selling the off-chip voltage regulators. Nanochronous builds copies of critical paths and uses these to adapt the clocking to the environment (process corner, voltage, temperature) so that the chip will automatically consume less power but still run to spec as the voltage is lowered. Elastix does something similar, adapting the performance of the chip as the voltage is altered, while taking the process corner into account. Handshake removes the clock completely and runs asynchronously with whatever performance is appropriate given the power supply voltage. Nanochronous is in Greece, Elastix is in Spain, and Handshake is in Netherlands; it must be something in the wine.

The next approach is to vary the voltage to islands while the chip is being used, rather than having fixed, but different, power supply voltages for each island. When the voltage is changed under software control it is known as dynamic voltage and frequency scaling. This is a technique that is talked about a lot

and used only a little, as far as I can tell. The idea is that if your microprocessor (or whatever) is not doing anything very important, why not run it slowly. And when it is in heavy computation mode run it flat out. To do this is tricky though. To slow it down the frequency must be lowered, and then (and only then) the voltage can be lowered. To speed up, the voltage must be raised, which takes time if it is not going to create a lot of power-supply noise, and then the frequency can be bumped up.

A lot of power gets consumed in the clock tree itself. Certainly 30% and sometimes 50% of the total power. Azuro works on laying this out and placing the gates more sensibly than is typically done by the clock tree synthesis built into every place and route tool.

Cyclos has another approach to reducing the 30% consumed in the clock. They think that clocks are the wrong shape, being square waves. If the clock was a sine wave then it could be resonant if we added some inductors, and would not consume power in the clock tree. That would be nice but the price is that every clocked element needs to be adapted so it can work with a sinusoidal clock rather than the usual rising-edge, falling-edge square wave we are all used to.

No list of all companies in the power area would be complete without Sequence, now a division of Apache, some of whose ancestral companies were around for over 15 years. Their primary focus is on measuring power, with or without vectors, at netlist or RTL level. They are pretty much the standard tool for this.

Multicore

As most people know, power is the main reason that PC processors have had to move away from single core chips with increasingly fast clock rates and towards multi-core chips. Embedded chips are starting to go in the same direction too; modern cell-phones often contain three or more processors even without counting any special purpose ones used for a dedicated purpose like mp3 decode. The ARM Cortex is multicore.

Of course this moves the problem from the IC companies, how to design increasingly fast processors, to the software side, how to write code for multi-core chips. The IC companies have completely underestimated the difficulty of this.

The IC side of the house has assumed that this is a problem that just requires some effort for the software people to write the appropriate compilers or libraries. But in fact this has been a research problem for over forty years: how do you build a really big powerful computer out of lots of small cheap ones? It is unlikely to be solved immediately, although clearly a lot more research is going on in this area now.

There are some problems, traditionally known as "embarrassingly parallel," which are fairly easy to handle. Far from being embarrassing, the parallelism is so simple that it is easy to make use of large numbers of processors at least in principle. Problems like ray-tracing, where each pixel is calculated independently, are the archetypal example. In fact nVidia and ATI graphics processors are essentially multi-core processors for calculating how a scene should be rendered (although they don't use ray-tracing, they use cheaper polygon-based algorithms). In the EDA world, design rule checking or RET decoration are algorithms where it is (fairly) easy to parallelize them: divide the chip up into lots of areas, run the algorithm on each one in parallel and take a lot of care on stitching the bits back together again at the end.

But most problems are more like Verilog simulation. It is hard to get away from having a global timebase, and then the processors have to run in lock-step and the communication overhead is a killer. Yes, in limited cases processors can run ahead somewhat without violating causality (such as simulating fast processors connected by slow Ethernet) and so reduce the amount of required synchronization but the typical chip is not like that.

Years ago Gene Amdahl noted that the amount of speedup that you can get by building a parallel computer of some sort is limited not by what can be made parallel but what cannot. If, say, 10% of the code cannot be parallelized, then even if we take the limiting case that the parallel code finishes instantaneously, the maximum speedup is just 10 times. This has come to be known

as Amdahl's law. So that is the first limitation on how to use multi-core. To use hundreds of cores effectively then the amount of code that cannot be completely parallelized must be tiny.

The next problem is that it is not possible to divide up the problem at compile time and capture that decision in the binary. If you have a loop that you are going to go around 1000 times to calculate something for 1000 elements, then one way is to unroll the loop, spawn the calculation simultaneously on 1000 threads on 1000 processors and accumulate the results. If the amount of calculation is very large compared to the overhead of spawning and accumulating, this might be good. But if you only have two processors, then the first two threads will go ahead and the next 998 will block waiting for a processor to become available. All the overhead of spawning and accumulation and blocking is just that, overhead that slows down the overall computation. How to map computation to processors must be done partially at run-time when the resources available are known.

The other big problem is that most code already exists in libraries and in legacy applications. Even if a new programming paradigm is invented, it will take a long time to be universally used. Adding a little multi-threading is a lot simpler than completely re-writing Design Compiler in a new unfamiliar language, which is probably at least a hundred man-years of effort even given that the test suites already exist.

There are some hardware issues too. Even if it is possible to use hundreds of cores, the memory architecture needs to support enough bandwidth of the right type. Otherwise most of the cores will simply be waiting for relatively slow access to the main memory of the server. Of course it is possible to give each processor local memory, but if that is going to be effective those local memories cannot be kept coherent. And programming parallel algorithms in that kind of environment is known to be something only gods should attempt.

I've completely ignored the fact that it is known to be a hard problem to write parallel code correctly, and even harder when there really are multiple processors involved not just the pseudo-parallelism of multiple threads or processes. As it happens, despite spending my career in EDA, I've got a PhD in operating

system design so I speak from experience here. Threads and locks, monitors, message passing, wait and signal, all that stuff we use in operating systems is not the answer.

Even if the programming problem is solved with clever programming languages, better education and improved parallel algorithms, the fundamental problems remain. Amdahl's law limiting speedup, the bottleneck moving from the processor to the memory subsystem, and the need to dynamically handle the parallelism without introducing significant overhead. They are all hard problems to overcome. Meanwhile, although the numbers are small now, the number of cores per die is increasing exponentially; it just hasn't got steep yet.

Our brains manage to be highly parallel though, and without our heads melting, so there is some sort of existence proof of what is possible. But, on the downside, we are really slow at calculating most things and also pretty error-proon.

Internal development

One potential change to the way chips are designed is for EDA to become internal to the semiconductor companies. In the early days of the industry it always was.

Until the early 1980s there wasn't really any design automation. There were companies like Calma and Applicon that sold polygon level layout editors (hardware boxes in those days) and programs like Spice and Aspec that were used for circuit simulation (and usually run on mainframes). Also there were a couple of companies supplying DRC software, also typically run on mainframes.

In the early 1980s, companies started to develop true design automation internally. This was implemented largely by the first set of students who'd learned how to design chips in college as part of the Mead and Conway wave. Hewlett-Packard, Intel and Digital Equipment, for example, all had internal development groups. I know because I interviewed with them. Two startups from that period, VLSI Technology (where I ended up working when I first came to the US) and LSI Logic had ambitious

programs because they had a business of building chips for other people. Until that point, all chips were conceived, designed and manufactured internally within semiconductor companies. VLSI and LSI created what we initially called USICs (user specific integrated circuits) but eventually became known, less accurately, as ASICs (application specific integrated circuits). It was the age of democratizing design. Any company building an electronic product (modems, Minitel, early personal computers, disc controllers and so on) could design their own chips. At this stage a large chip was a couple of thousand gates. The EDA tools to accomplish this were supplied by the semiconductor company and were internally developed.

First front-end design (schematic capture and gate-level simulation) moved out into a 3rd party industry (Daisy, Mentor, Valid) and then more of design with companies like ECAD, SDA, Tangent, Silicon Compilers, Silicon Design Labs and more moved out from the semiconductor companies into the EDA industry.

At first the quality of the tools was almost a joke. I remember someone from the early days of Tangent, I think it was, telling me about visiting AT&T. Their router did very badly set against to the internal AT&T router. But there was a stronger focus and a bigger investment behind theirs and it rapidly overtook the internal router. Since then almost all EDA investment moved into the 3rd party EDA industry. ASIC users, in particular, were very reluctant to use tools that tied them to a particular silicon manufacturer since they didn't want to get locked-in for their next design. Since every semiconductor company wanted to get into ASIC (even Intel had an ASIC group) and the ASIC flow was pretty much standard (gate-level handoff and back-annotation) the market exploded.

ASIC, in the sense of designs done by non-semiconductor companies, has declined as levels of integration have gone up (what was 5 chips is now 1) and as most designs that are not power-sensitive have moved to FPGAs. So once again most designs are done inside semiconductor companies where being "locked-in" to in-house tools would not be an issue.

The EDA industry invests approximately 20% revenue in R&D. Maybe even 35% if past acquisitions were properly accounted for. So there is somewhere around a 3 to 5 times cost disadvantage. Also, it is generally accepted that producing a generalized supported software product is at least 3 times (and maybe much more) expensive than just developing a product for internal use. With approximately 3 serious competitors in each tool segment, the EDA industry needs to take about 30 times as much money from the semiconductor industry as it would cost a semiconductor company to develop a tool internally. That is 3 tools being developed, each at a cost 3 times the internal development, with selling price of 3 times the cost of development. This is significant since the number of large semiconductor companies purchasing tools is also declining as they consolidate and/or run into financial trouble. It is too early to call predict exactly how that will pan out.

There is today no market for specialized tools for microprocessor design. The tools are all internally developed. It is certainly arguable whether it would be possible to produce a general tool but the economics would not work in any case. There simply are too few microprocessor design groups to pay the tax of the EDA industry generality, overhead and profit.

There is no real market today for tools for FPGA design. The tools are all (OK, mostly) internally developed. But the economics wouldn't work when there are only 2 or 3 FPGA vendors. It is more economic for each vendor to develop their own suite (not to mention that it better fits their business model).

One future scenario is that all semiconductor design becomes like microprocessor design and FPGA design. Too few customers to justify an external EDA industry, too specialized needs in each customer to make a general solution economic. Design moves back into the semiconductor companies. I don't have much direct knowledge of this happening, but Gary Smith is always pointing out that it is an accelerating trend, and he sees much better data than I do.

One other issue is that for any design tool problem (such as synthesis or simulation) there is only a small number of experts in the world and, by and large, they are not in the CAD groups of

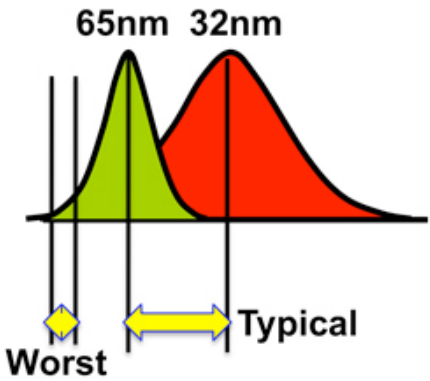
semiconductor companies, they are in the EDA companies. I predicted earlier that the world is looking towards a day of 3 semiconductor clubs. In that environment it is much more like the FPGA world and so it is not far-fetched to imagine each club needing to develop their own tool suite. Or acquiring it. Now how many full-line EDA companies are there for the 3 clubs? Hmm.

Process variation: you can't ignore statistics any more

I like to say that "you can't ignore the physics any more" to point out that we have to worry about lots of physical effects that we never needed to consider. But "you can't ignore the statistics any more" would be another good rallying cry.

In the design world we like to pretend that the world is pass/fail. If you don't break the design rules your chip will yield. If your chip timing works at the worst case corner then your chip will yield (yes, you need to look at other corners too).

But manufacturing is actually a statistical process and isn't pass/fail at all. One area that is getting worse with each process generation is process variability especially in power and timing. If we look at a particular number such as the delay through a nand-gate then the difference between worse-case and typical is getting larger. The standard-deviation about the mean is increasing. This means that when we move from one process node to the next, the typical time improves by a certain amount but the worst-case time improves by much less. If we design to worst-case timing we don't see much of the payback from the investment in the new process.



An additional problem is that

we have to worry about variation across the die in a way we could get away with ignoring before. In the days before optical proximity correction (OPC) the variation on a die were pretty much all due to things that affected the whole die: the oxide was slightly too thick, the reticle was slightly out of focus, the metal was slightly over-etched. But with OPC, identical transistors may get patterned differently on the reticle, depending on what else is in the neighborhood. This means that when the stepper is slightly out of focus it will affect identical transistors (from the designer's point of view) differently.

Treating worst-case timing as an absolutely solid and accurate barrier was always a bit weird. I used to share an office with a guy called Steve Bush who had a memorable image of this. He said that treating worse case timing as accurate to fractions of a picosecond is similar to the way the NFL treats first down. There is a huge pile of players. Somewhere in there is the ball. Eventually people get up and the referee places the ball somewhere roughly reasonable. And then they get out chains and see to fractions of in inch whether it has advanced ten yards or not.

Statistical static timing analysis (SSTA) allows some of this variability to be examined. There is a problem in static timing of handling reconvergent paths well, so that you don't simultaneously assume that the same gate is both fast and slow. It has to be one or the other, even though you need to worry about both cases.

But there is a more basic issue. The typical die is going to be at a typical process corner. But if we design everything to worst case then we are going to have chips that actually have a much higher performance than necessary. But now that we care a lot about power this is a big problem: they consume more power than necessary giving us all that performance we cannot use. There has always been an issue that the typical chip has performance higher than we guarantee, and when it is important we bin the chips for performance during manufacturing test. But with increased variability the range is getting wider and when power rather than timing is important, too fast is a big problem.

One way to address this is to tweak the power supply voltage to slow down the performance to just what is required, along with a commensurate reduction in power. This is called adaptive voltage scaling (AVS). Usually the voltage is adjusted to take into account the actual process corner, and perhaps even the operating temperature as it changes. Once this is done then it is possible to bin for power as well as performance. Counterintuitively, the chips at the fastest process corner may well be the most power thrifty since we can reduce the supply voltage the most.

CDMA tales

It is very rare for a company to develop a new standard and establish it as part of creating differentiation. Usually companies piggy-back their wares on existing standards and attempt to implement them better than the competition in some way. There were exceptions with big companies. When AT&T was a big monopoly it could simply decide what the standard would be for, say, the modems of the day or the plug you phone would use. IBM, when it was an effective monopoly in the mainframe world, could simply decide how magnetic tapes would be written. I suppose Microsoft can just decide what .NET is and millions of enterprise programmers jump.

Qualcomm, however, created the basic idea of CDMA, made it workable, owned all the patents, and went from being a company nobody had heard of to being the largest fabless semiconductor company and have even broken into the list of the top 10 largest semiconductor companies.

The firs time I ran across CDMA it seemed unworkable. CDMA stands for code-division multiple access, and the basic technique relies on mathematical oddities called Walsh functions. These are functions that everywhere take either the value 0 or 1 and are essentially pseudo-random codes. But they are very carefully constructed pseudo-random codes. If you encode a data stream (voice) with one Walsh function and process it with another at the receiver you get essentially zero. If you process it with the same Walsh function you recover the original data. This allows everyone to transmit at once using the same frequencies, and only

the data stream you are trying to listen to gets through. It is sometimes explained as being like at a nosiy party, and being able to pick out a particular voice by tuning your ear into it.

Years ago I had done some graduate work in mathematics, so I'd actually come across Walsh functions and so the idea of CDMA was very elegant. However, my experience of very elegant ideas is that they get really messy when they meet real-world issues. Force-directed placement, for example, seems an elegant concept but it gets messier once your library cells are not points and once you have to take into account other constraints that aren't easily represented as springs. So I felt CDMA would turn out to be unworkable in practice. CDMA has its share of complications to the basic elegant underpinning: needing to adjust the transmit power every few milliseconds, needing to cope with multiple reflected, so time-shifted, signals and so on.

At the highest level what is going on is that GSM (and other TDMA/FDMA standards) could get by with very simple software processing since they put a lot of complexity in the air (radio) interface and didn't make optimal use of bandwidth. CDMA has a very simple radio interface (ignore everyone else) but requires a lot of processing at the receiver to make it work. But Moore's law means that by the time CDMA was introduced, 100 MIPS digital signal processors were a reality and so it was the way of the future.

Of course, my guess that CDMA was too elegant to be workable was completely wrong. Current and future standards for wireless are largely based on wide-band CDMA, using a lot of computation at the transmitter and, especially, receiver to make sure that bandwidth is used as close to the theoretical maximum as possible.

But before CDMA turned out to be a big success Qualcomm was struggling. In about 1995 VLSI tried to license CDMA to be able to build CDMA chips as well as the GSM chips that they already built. Qualcomm had "unreasonable" terms and were hated in the industry since they charged license fees to people who licensed their software, people who built phones (even if all the CDMA was in chips purchased from Qualcomm themselves) and people who built chips (even if they only sold them to people who

already had a Qualcomm phone license). They were hated by everyone. Now that's differentiation. The royalty rates were too high for us and we ended up walking from the deal.

I was in Israel 2 days from the end of a quarter when I got a call from Qualcomm. They wanted to do a deal. But only if all royalties were non-refundably pre-paid up front in a way they could recognize that quarter. Sounds like an EDA license deal! We managed to do a deal on very favorable terms (I stayed up all night two nights in a row, after a full day's work, since I was 10 hours different from San Diego, finally falling asleep before we took off from Tel Aviv and having to be awakened after we'd landed in Frankfurt). The license was only about $2M or so in total I think, but that was the relatively tiny amount Qualcomm needed to avoid having a quarterly loss and impacting their stock price and so their ability to raise the funds that they would need to make CDMA a reality. Which they proceeded to do.

Another look at internal development

In the early 1980s VLSI design techniques were being disseminated outside a handful of semiconductor companies where the priestly knowledge had previously been secreted. VLSI Technology and LSI Logic (primarily) invented ASIC design, whereby customers did part of the design and the semiconductor company (we called them foundries then, but they were not exactly the same as what we call a foundry today) did the rest. A lot of design tool development was internal. There was a good reason for that, namely that there was no 3rd party EDA industry providing the necessary tools, and so no tools meant no product to fill the fab. Remember that if you have a fab you are like an hotel; every wafer start slot with no actual wafer to start is like a plane with an empty seat. Once the slot has passed or the plane takes off, it's gone for ever.

So VLSI Technology and LSI Logic (and HP, Intel, TI and everyone else) had a large amount of internal CAD. It was differentiation to some extent, and there wasn't really an alternative. The CAD teams were staffed with top rate engineers,

many of them M.Sc. and Ph.D. students from the first cohort of people from universities starting to teach and research VLSI design.

Then came the first wave of EDA companies, the DMV—Daisy, Mentor and Valid. They provided systems for doing some front end design, basically schematic capture and simulation. The standard ASIC methodology was to do design to netlist, ship the netlist to the semiconductor vendor who would do place and route (either as a standard-cell design or a gate-array) and provide back annotation of capacitance values (we didn't worry about resistance back then, timing was dominated by capacitance) for resimulation and signoff.

This design flow didn't work very well at first. Wilf Corrigan, CEO of LSI Logic famously complained that the EDA industry took all the profit from ASIC. Customers would buy tools but the semiconductor company would only get their money once a design could get through the flow. So much of the heavy lifting to mature the flows and make them workable was done by the semiconductor companies not the EDA companies. The next generation of EDA companies was SDA and ECAD who merged to form Cadence (this was long before Synopsys). The Japanese semiconductor companies adopted 3rd party EDA vigorously, since they had very limited internal development and this allowed them to get into the new markets that ASIC was opening up.

The writing was on the wall for internal EDA, at least in the long term.

So the EDA part of the business split into an external part, the EDA industry, and an internal part, the CAD groups. By and large the CAD groups were training grounds for entry level engineers, some engineers with deep design experience and, usually, first rate management (often drafted in from the design side of the company to keep the internal politics calm). They knew how to use tools but not how to create them. The internal tool developers migrated from the semiconductor companies into the EDA companies.

The two exceptions to this, companies that kept large internal development groups, were Intel and IBM who still, to this day,

develop significant amounts of EDA software for their own internal use. But it is very expensive to do this, and even they don't know how useful it is. I once asked someone in Intel's Design Technology (their internal CAD/development group) whether their routers were better than the EDA industries. He admitted they didn't have a clue; they had no bandwidth to even take a look at what was available externally.

So I don't see internal development being a major force since the economics don't work very well. However, that could change with consolidation of both EDA and semiconductor companies, and especially if a semiconductor company jump-started internal development by acquiring one or more EDA companies.

Chapter 5: Finance and Investment

Venture capital for EDA is dead

xx

There have been a many recent articles about venture capital and how it is changing. They have focused on venture capital being broken since many funds are losing money and even that venture capital, in aggregate, is investing more than it is taking out in exits. There are short-term problems with the current downturn too. For example, I've heard of limited partners (the actual investors) refusing capital calls when a fund wants to make an investment. As a result, VCs have essentially suspended new investments for Q4 of 2008 and Q1 of 2009. Of course one of the biggest problem of all is that the market for IPOs is closed completely.

In the EDA world venture capital has been broken for some time. VC funds are simply too large and too risk averse. In effect they have become private equity banks. The reason size matters is that a $500M or $1B fund simply can't make $5M or $10M investments. A partner can only serve on a limited number of boards and funds of this size would need to make hundreds of investments. Instead, they need to focus on making fewer big investments. However, early stage investments simply don't require that much money, and the amount of money that they do require is continuing to decrease in the software world. A cynic might look at the enormous VC investment in cleantech, especially solar, as being attractive simply because they require a lot of money to get to manufacturing. On the other hand, Web 2.0 and EDA startups require little money, often less than $10M, to get to revenue.

The herd dynamic means that VCs all either want to invest in a sector or don't. A VC told me once that "venture capitalists make

sheep look like independent thinkers". So if the big guys like Sequoia or Kleiner-Perkins are avoiding a sector (perhaps just because their funds are too simply big for the sector or the limited partners have directed them) then everyone else seems to avoid it too. EDA has been out of favor for some time since each startup does not address a "billion dollar market" nor is it not going to be a phenomenon (although without much revenue) like Facebook or mySQL.

In the software world, the cost of hardware is basically zero (we all have a computer already and even buying a new one is something we can afford without a VC) and software productivity is improving all the time. In the web world this is driven by languages like Python and Ruby, along with environments like Django or Rails. And for web infrastructure there is Amazon S3 and Google App Engine. All the costs are variable so big upfront investment isn't needed even to scale to millions of users. In the EDA world this is driven by the same languages, along with infrastructure like Open Access that mean that a startup doesn't need to spend its first year or so building its underlying scaffolding, it can focus immediately on code that adds real value to users.

Paul Graham of Y-combinator thinks that VCs have become redundant in the internet space. In EDA the amount of money required is low enough that personal investment and private investors are sometimes enough to get to positive cash-flow without any venture capital at all. Altos and Apache are both profitable and were funded entirely privately. Denali has been private and reportedly very profitable for a long time; they certainly throw a good party.

How long before a venture capital fund decides to buy a company, removing any semblance of being genuinely different from private equity banks?

Venture capital for your grandmother

Like many of us in Silicon Valley, I often encounter people (hi Dad!) who don't understand venture capital. I don't mean all the details, I mean just the basic way investment is done in a startup. Often, even employees in startups don't understand it either. Here's how I explain it.

Let's say you've got a good idea for a company, and you have done some work on it to produce a prototype. Maybe at this stage there are two of you. You have the prototype, the team (the two of you) and a business plan. Let's say everyone agrees it is worth $1M at this point. We'll ignore how that valuation was arrived at, although it is somewhat like buying a house, you (seller) want a higher valuation and the VC (buyer) wants lower so the valuation depends on the going rate for that sort of company and somewhat on how desperate you are to sell and how enthusiastic the VC is to buy. So you and your partner each have stock in your brand new company worth $500K. But you can't just sell the company at this stage, companies like that don't have any buyers at all. You need to make it more successful first.

So you decide you need some investment money, so you can pay yourselves and hire some more employees. You convince a venture capitalist that your company is going places and he or she wants to put in $500K. Everyone agreed that the company is worth $1M before this happens. This is called the pre-money valuation. The VC wires $500K to your bank account and you give them stock for 1/3 of the company. Suddenly your company is worth $1.5M, consisting of $1M for the company as it was the day before, plus another $500K sitting in the company bank account. This is called the post-money valuation. So you and your partner each own 1/3 of the company and the VC owns 1/3 of the company. But the valuation is higher so your 1/3 is worth $500K, exactly the same as your 1/2 was worth the day before. You've neither lost nor gained anything.

So what have you given up? A share of the future gains. You used to own 100% of the company with your partner, now you

only own 2/3 of the company. If the company suddenly becomes worth $60M then you each have $20M and the VC has $20M (the VC has preferred stock, which is different from what you and your partner probably have, so this might not be precisely accurate but it is close enough). What you gave up was that if the company was suddenly worth $60M before, you and your partner would have $30M each. But realistically, that wasn't going to happen because you didn't have enough money on your own to fund the company over time. So if this scenario plays out you get a nice payout. But, of course, if the company becomes worthless then everyone's share goes to zero. 1/3 of zero is zero.

You might assume that if the company nearly goes bankrupt and is sold for just, say, $300K that you'd have 1/3 of it, namely $100K. But that is where the biggest difference between preferred stock and your stock comes to light. The preferred stock is so called because it gets preferential treatment and in this scenario the VC gets all of the money. You have a loss of $500K but then you never put in any real money so it is a paper loss. The VC has a very real loss of $200K since they put in $500K, you spent it, and the company pretty much failed.

Next, full-ratchet anti-dilution provisions and piggy-back rights. Well, maybe not.

EDA: not boring enough

EDA is fun. Innovation is fun and not many businesses require as much innovation as EDA. Working in an EDA startup in particular was (and still can be) a lot of fun because the ratio of innovation to meetings, company politics and the rest is much higher.

But one effect of this has been that too many people want to start EDA companies. It is not as bad as Web 2.0 companies, and with the current freeze in EDA investment it is over for the time-being and maybe forever.

One piece of advice I remember seeing, I forget where, is never to do a job that has significant non-monetary compensation for doing it. Too many people will want to do it for those other

reasons. Everyone wants to open a restaurant, write a book, and be an actor.

The company where my son works in San Francisco advertised for a graphic designer on craigslist. They took the ad down again after over 200 people had applied for the job. They took the ad down after…four hours. Too many people want to be graphic designers because they think it is cool, or arty, rather than because it is a profitable business to which they are especially well suited.

The person sitting next to me on a flight to Chicago once told me that he was in the concrete business. He had a dozen concrete plants in towns you've never heard of in unfashionable parts of the mid-West. The economics were simple. A town can support one concrete plant but not two. Consequently the owner of a concrete plant has a sort of monopoly. Sure, a contractor can buy concrete from another plant, but that is one town over, perhaps an additional 50 miles round trip for the concrete truck, a cost that makes it non-competitive. His plants returned over 30% of their capital every year. Concrete is far more profitable than EDA and partly because it is so boring.

If that guy was our Dad and we inherited the business, I'm sure we could all run it. But we don't even consider businesses like that because technology is more exciting. EDA is not badly paid by any means, but considering just how hard it is and how much training and knowledge is required it is not that well-paid either.

I've read (but not verified) that one very well paid group of consultants are people who do Cobol programming. Everyone wants to program next generation web applications using AJAX and Python, not some crusty programming language designed in the 1950s. How much further from the trendy cutting edge can you get.

Bill Deegan, a friend of mine, does the equivalent in the EDA world. Not the sexy EDA algorithms for him, he creates and maintains the build and Q/A systems without which the programmers don't have a product. Usually his clients bring him in when the build system has been ignored by the hot-shot programmers for so long that they can barely build their product

never mind release it to a customer. He describes it as like garbage collection (the kind with a truck, not recovering unused program memory). It's not glamorous but it needs to be done, and done well, and just like garbage collection, things get really messy if it isn't. You won't be surprised to know that he is rarely idle.

One hit wonders

Venture capitalists have the concept of a zombie. Just like in the movies, a zombie company is one of the living dead. It is a company that is not burning through cash, and so is not going to go bankrupt if starved of further investment. On the other hand, it is not doing well enough that it has any exit possibilities. Venture capitalists have a fuse burning on their funds though, and generally 7 to 10 years after they first raised money for the fund they want to be able to close it down and do the final accounting: so many companies were sold for nice gain, so many ran out of money and so on.

But zombies make this difficult since they are not dead yet and could even go on for years growing slowly, successfully funding operations out of revenue but never achieving a growth rate that is going to interest another company in a merger or acquisition, never throwing off enough profit to make a merger with anyone accretive.

In this scenario, VCs will push companies to try something, anything, that might create success, even with the attendant risk of total failure. VCs like the answer to be clear even though the employees would rather simply have a job for a long time. Simply winding up the company is unattractive since, say, $1M for a technology sale is so close to zero as to be the same thing, so if there is any risky chance of quickly making the company genuinely successful that is more attractive.

Public companies can get into this state too, not doing well enough to go anywhere but not doing badly enough to die. Their stock price languishes since there is no chance that anyone is going to try and acquire the company as for its running business,

and little chance that the company is going to break out and become a star performer.

For example, at one point soon after I left VLSI Technology their market cap (their share price times the number of shares outstanding) was not only less than their book value (the value of all their capital equipment, buildings, investments and cash) but less than the cash they had in the bank. In theory it should be possible to buy the company using its own cash (ignoring any premium).

This is not just like buying a house with no money down, it is like buying a house for $500K when you know there is $600K in the master-bedroom closet. It is a vote of no-confidence by the shareholders, an acknowledgement that the company is in the value destroying business. Of course VLSI at that cheap price was attractive and Philips Semicondutors (now NXP) bought it for its wireless business with Ericsson and people who knew how to get an process node into production a year or two faster than NXP was able to do with their conservative approach.

A company that is currently in this sort of shape is California Micro Devices (CAMD). Their stock price today is $2.18 with 23.55M shares outstanding. So their market cap is $51.35M. Their last four quarters of revenue totaled just over $60M on which they lost less than $1M. They can go on for a long time like that.

But they have $51.6M of cash and $66.9M of current assets (accounts receivable, inventory, short-term investments) and only $10.3M of debt. So their market cap is equal to their cash, and about half the value of simply winding up the company (probably not all the current assets would be realizable in this scenario though). It is like the house with the money in the closet.

Everyone knows that if they run the business as usual they will simply waste that cash. There is really no reason not to simply wind up the company and return the money to the shareholders, giving them about a 100% premium. But at the same time everyone knows they are not going to do that which is why the stock price does not reflect the break-up value.

Fabless semiconductor companies often have a single hit and make a lot of money on that first chip. Portal Player, who made the sound chip in the first iPods are a good example. But, if they are not acquired, they sometimes go on to burn that money trying to follow up their first hit with a second only to discover like Little Eva (Locomotion) or Norman Greenbaum (Spirit in the sky) that they are a one-hit-wonder.

Will you greenlight my chip

I already took a cursory look at the fact that the semiconductor industry is going to restructure, partially driven by the current economic downturn but mainly by the fact that almost all semiconductor companies are going to become completely fabless. What we have got used to calling IDMs (integrated device manufacturers) are just going to be large semiconductor companies that used to have fabs. This trend is driven by two things: the economic size of fab has got so large that it exceeds most semiconductor companies' needs; and the cost of process development has got too high for any single semiconductor company (except Intel, TSMC and some of the DRAM guys) to be able to afford it.

Having a fab to fill means that a semiconductor company has a huge fixed cost that has to be amortized over all the wafers actually manufactured. This puts a huge premium on having the fab filled. Just like a hotel cannot inventory rooms, they are either occupied tonight or not, a fab cannot inventory wafer starts. Either a wafer was started or it was not, and a wafer not started is one that doesn't carry its share of the overhead of depreciating and staffing the fab. So semiconductor companies have grown up to contain collections of divisions that together require all the wafers a fab can produce. If there are not enough then it is attractive to acquire further product lines.

Once a semiconductor company has no fab, then the particular collection of businesses that make it up have very little reason to be grouped into the same company. Further, it makes very little sense for a semiconductor company to pay a big premium to acquire a new product by buying a fabless semiconductor

company since it no longer has a fab to fill and so doesn't really have any economies of scale. Sometimes, as with TI and digital signal processing, there is company-wide expertise that cuts across a many products. But often not. For instance, it is interesting that TI was attempting to sell its wireless business (it gave up because it couldn't get a good price) despite wireless having a significant DSP component.

One possible future scenario is that many of the semiconductor companies of today will disintegrate since they don't have a lot of reason to keep product lines together. In fact the whole idea of a product line may start to be obsolete since so much of a chip is now externally sourced IP, both semiconductor IP and software libraries. I also looked at how some semiconductor companies are one-hit-wonders, with a successful chip that fills their cash position, that they then gadually burn through.

An unlikely place to look for parallels to chip design is the movie industry. Back in the middle of the last century, the studios were like IDMs. They had an entire infrastructure for making movies that had to be amortized by making lots of movies (to fill the studio, like filling the fab). Today, movies are not made like that. They are made by virtual companies that are put together expressly to make a single movie, almost everyone is a subcontractor not an employee of the movie, and the company is disbanded when the movie has been made and the profits have (or, often, have not) been distributed to the investors.

Chip design could go like that, with an individual chip being built by a team of subcontractors assembled for just that purpose and manufactured by a foundry, probably TSMC. If the chip makes a lot of money the investors get a return; otherwise not. associated with keeping a company together just because it had a hit product and no guarantee that the next product will be another hit. Better to distribute the profits and fund the next chip as a completely independent project. Every chip is a one-hit-wonder by design.

So, do you want to green-light my chip, Mr Spielberg?

Crushing fixed costs

There is a trend that the current downturn is only going to accelerate: to turn fixed costs into variable costs. Often this is what is behind outsourcing of some capability. Sometimes it is driven purely by either cost (let's do it in China) or core-competence considerations (do we really need to run our own cafeteria?) but often it is driven by a desire to switch an inflexible fixed cost for a variable cost. Instead of owning a fab (fixed cost) then let's just buy wafers from TSMC (variable cost).

There are two big problems with a large expensive fixed cost. One is just that it is expensive and so it ties up a lot of capital (or a lot of expense budget for a "fixed" cost like employees) for which there may well be more profitable uses. Second, the fixed cost usually puts in place a fixed capacity of some sort, and that capacity risks always being either more than the market need is, or less than the market need is.

TSMC makes money as a foundry, of course (well, maybe not right now). It's scale is enormous so it may well be able to make money selling wafers for the same price as you can get wafers out of your own fab, even if you have one running at capacity. But that's the point. Your fab is never running at capacity. It is either below capacity, in which case wafers cost more than the "standard price" because all that depreciation needs to be spread over fewer wafers. Or else it is above capacity, meaning that there are wafers that you could sell profitably that are not being built (if your planning is poor, you may even have orders for them, and commitment dates that you are going to miss). Even if you pay a price higher than your standard price for wafers, it is worth a lot to avoid having to absorb fab variances when the fab is not full, and to gain the capability to sell more than capacity when you have a strong order book.

In the web space, you no longer need to build your own high-capacity server farm. Amazon, Google and others will sell you server time and disk space on a purely variable cost basis. If you website becomes a big hit then scaling should be much more straightforward.

In some ways you can look at Amazon S3 or TSMC as companies that are in the business of making the up-front investment in fixed cost assets and then charging you a variable cost to use them. Lots of other companies do the same. It doesn't cost an airline anything (well, not much) extra to fly an extra passenger; it is basically in the job of taking airplanes (fixed cost) and working out good business models to sell trips (variable cost). Cell-phone companies largely have a network of base-stations (fixed cost) and work out how to charge each customer for using them (variable cost). It's not always obvious what the best model is for making the cost variable: do you charge data per megabyte, or unlimited data for a month? How does the money get split when you are roaming on other people's networks? Is data the same price as the digitized data underlying a voice-call?

When supply chains disaggregate, usually one thing that happens is that non-core areas, especially ones involving fixed costs such as equipment or full-time employees, are divested. New companies spring up to specialize in providing that non-core activity as their core competence. Ross Perot made his fortune at EDS taking companies' IT departments off their hands and created a big specialist company to provide those services. Semiconductor companies get rid of their EDA groups and an EDA industry comes into existence (Cadence, Synopsys, Mentor etc). Semiconductor companies get rid of some of their fabs and a foundry industry comes into existence (TSMC, UMC, Chartered etc). Semiconductor companies get rid of their technology development (TD) groups and rely on the foundry industry for that too. One interesting area of debate right now is whether design is next, and how much of design. Nokia already moved its chip development group into ST. eSilicon, according to Jack Harding its CEO, is doing very well. Faraday is (or at least was) doing about 200 designs a year.

When semiconductor companies design chips about as often as they reconfigure buildings, does it make any more sense to have their own not-very-expert employees floor-planning their chips than their own building architects floor-planning their offices.

Technology of SOX

Sarbanes-Oxley, often abbreviated to SOX, is a set of accounting rules that were introduced by congress in response to the accounting scandals of Enron, Worldcom and their like during the dotcom boom. It is a mixture of different regulations, some concerned with how companies are audited, some concerned with liability a CEO and CFO have for irregularities in their companies, and so on. Many provisions are completely uncontroversial.

But the biggest problem, especially for startups, comes about from section 302 and 404. Section 302 says that companies must have internal financial controls, that the management of the company must have evaluated them in the previous 90 days. Section 404 says that management and the auditors must report on the adequacy and effectiveness of the internal controls.

In practice this means that auditors must repeatedly go over every minute piece of data, such as every cell in a spreadsheet, every line on every invoice, before they can sign off. For a small company, the audit fees for doing this are a minimum of $3M per year. For larger companies the amount grows, of course, but slowly so that it is much less burdensome for a large established company (where it might be 0.06% revenue) than for a small one.

Only public companies are required to comply with SOX so you could argue that it doesn't matter that much for a small venture funded startup. At one level that is true. But it has also meant that a company has to be much larger to go public.

In the past, an EDA company with $20M in revenue and $3M in profit (with good growth) could go public. But now a private company like that must comply with SOX to go public, so that $3M cost suddenly hits and the company has $20M in revenue and no profit at all. It must wait until it is, perhaps, $80M in revenue with $12M in profit (which would have been $15M without SOX). In EDA, in particular, this is extremely difficult to achieve with a single product since most sub-markets are not that large. In effect, EDA companies can no longer go public.

This applies to many venture-backed startups in whatever industry. Since the introduction of SOX most IPOs have taken place in either London or Hong-Kong. It is controversial just how much of that is directly due to SOX but clear that a lot of companies that could have gone public in the past in the US have not done so, and as a result have been acquired for lower valuations that would otherwise have been the case.

In the early 2000s (SOX was passed in 2002) the stock market was not friendly to IPOs as it recovered from the downturn. But venture backed IPOs in 2005 and 2006 were way below what they were in the 1990s, and Q2 2008 was the first quarter in 30 years in which no venture-backed IPOs took place in the US. This has been another reason that VCs are rarely willing to invest in EDA companies.

EDA and startups: $7M to takeoff

A startup EDA company needs about $7M in bookings to become self-sustaining and not require another round of external funding. Curiously, it doesn't seem to depend all that much on the product provided there is really a market out there, which, of course is by no means a given. And more funding can always be an accelerator to growth even if slow growth would have been possible without it.

The R&D team should be about 10 people. It will be less in the early days but it shouldn't really be more unless the company truly must develop a range of products in parallel. With more than 10 people, engineering will be off developing a range of products even if that isn't the plan!

With a CEO and another "person" in the form of an accountant, an office manager, a little marketing (they may be one person or more likely a few people part-time) that makes a total of 12-13 people, which is a fixed cost of around $2.5-3M per year. A single sales team is around $800K-1M per year. With that headcount in place it takes about $3.5-4M to break even.

But a sales team only brings in $2M so $3.5M is more than one sales team can bring in so we need a second, at another $800K-

1M pushing the breakeven up to $4.5M. This is just about doable but more likely a third sales team will be required, pushing revenue to $6M and expenses to $5.5M. Add in some inefficiencies in training salespeople, filling the funnel and the rule of thumb is that you need to get to about $7M to become cash-flow netural and the company start to be able to fund its own growth, albeit slowly.

But breakeven isn't the end goal, being profitable enough to have options is. Then we can be acquired, or continue to grow the business or even just pay our shareholders nicely out of the profits. This means getting the business up to about $10-11M, which means about 5 sales teams. The 5 sales teams will cost about $4-5M, leaving $6M. That will pay the $3M original (non-sales) fixed cost with $1M for some additions to the corporate team: a marketing person, maybe some non-bag-carrying sales management, and after a couple of years somebody might want a pay raise. That leaves $2M to either take as profit or use to fund further growth, start a second product and so on.

All of this makes one big assumption. That the product is really ready at the point that the channel expenses are ramped up. It assumes that each salesperson rapidly makes it to the $2M per sales team level. This is where companies die though. If the sales teams are added too early then they will burn all the cash. If the product is not ready for the mainstream then the sales guys will not make it to the $2M level and burn all the cash. But if everything is in place then the company can get to $10M rapidly. The first year I was at Ambit we did $840K in revenue; the second year, $10.4M.

This is the point at which a company is very attractive for acquisition. It has traction in the market ($10M in sales and growing), the technology is proven (people are using lots of it; look, $10M in sales), the acquisition price hasn't got too expensive yet (only $10M in sales), it is probably the market leader in its niche ($10M in sales and growing). Of course if the company continues to grow it will typically take in more investment at this point in order to grow even faster. Value of a software company is some multiple of forward earnings, and the greater the growth the greater the multiple.

EDA startups: channel costs $6M

I've put together several business plans for EDA startups once the product is ready for market. One question is always how much money needs to be raised. The answer always turns out to be about $6M.

When you put together a spreadsheet to show how the bookings will build up, there are two factors to which the amount of money turns out to be very sensitive:

How long after you hire a salesperson before they start to produce revenue?

How fast does a salesperson ramp up to "full power"?

The answer to these two questions governs how much you need to invest in a sales team before they are a net positive for the company, and the total and timing of that investment governs how big a hole you have to cover and thus how much money you need to raise.

You might think that how big you plan to get is a critical variable, but in fact the answer is about a $50M run-rate after 5 years. If you don't have a plan like this then nobody will fund you (they probably won't anyway at present, but let's leave that to one side). You almost certainly won't grow that fast, and everyone knows it, but they will "take a haircut" to any numbers you give them anyway, so you'd better play along and give them big ones.

Other assumptions you'd better bake in. Any bookings you have will come in the last week of the quarter. This means that any cash associated with that order will not come until the following quarter. So every quarter, for every sales team, you need to pay all their salaries without the cash from the business they are generating that quarter to offset those expenses.

Each salesperson needs two application engineers to be effective. Or at least 1½. This means that a sales team costs approximately $800K per year in salaries, travel and so on, which is $200K per quarter, perhaps a little less if you don't have the full 2 AEs per salesperson.

As for sales productivity, at capacity a sales team brings in $2M/year. If you put in much more than this then you are simply being unrealistic. If you put in much less you'll find that the business never gets cash-flow positive.

EDA tends to have a 6 month sales cycle. So normally a new salesperson won't close business in less than 6 months, and probably 9 months (3 months to understand the product and set up initial meetings, 6 months of sales cycle). I like to use a ramp of $0, $0, $250, $250, $500 for the first 5 quarters, which assumes a salesperson sells nothing for two quarters, is at half speed for two quarters and then hits the full $2M/year rate. Later this may be conservative since a new salesperson can inherit some funnel from other existing salespeople in the same territory and so hit the ground if not running then at least not at a standing start. In the early days it might be optimistic since I've assumed that the product really is ready for sale and it is just a case of adding sales teams. But realistically it probably isn't.

So those are the variables. In 5 years you need to be at $50M which means about 25 sales teams at the end of year 4 (because only those sales teams really bring in revenue in year 5). Some may be through distribution, especially in Asia, but it turns out not to make all that much difference to the numbers.

In the meantime, the rest of the company has to be paid for and don't directly bring in revenue. So if you ramp sales too slowly, the rest of the company will burn more money in the meantime. This makes the model less sensitive than you would expect to the rate at which you hire sales people, within reason.

If you hire people too fast on day one, then the hole gets huge before your first teams start to bring in any money to cover the cost of the later guys. You need to get to about $7M of bookings before things get a bit easier and the early salespeople are bringing in enough to cover the costs of the rest of the company. However, if you bring in people too slowly then you will not get to a high enough number in the out years. The trick is to hire in a very measured way early and then accelerate hiring later. This will give a hole of about $4-5M meaning you should raise about $6M to give yourself some cushion to cover all the inevitable delays.

FPGA software

Why isn't there a large thriving FPGA software market? After all, something like 95% of semiconductor designs are FPGA so there should be scope for somebody to be successful in that market. If the big EDA companies have the wrong cost structure, then a new entrant with a lower cost structure, maybe.

In the early 1980s, if you wanted to get a design done then you got tools from your IC vendor. But gradually the EDA market came into being as a separate market, driven on the customer side by the fact that third-party tools were independent of semiconductor vendor and so avoided the threat of paying excessive silicon prices due to being locked into a software tool flows. Once the 3rd party EDA industry captured enough of the investment dollars then they could advance their tools much faster than any single company and the captive tool market was largely doomed.

For FPGAs that is not the case. If you want to do a design with Xilinx, then you get tools from Xilinx. With Altera, tools from Altera and so on. Yes, there are some tools like Synplicity (now part of Synopsys) and Mentor's FPGA suite, but they are focused only on the high end. But it is hard to make money only at the high end. When, over time, the high end goes mainstream then the FPGA vendors produce good-enough tools at free/low prices. So the R&D costs need to be recovered from just those few high-end users since the tools never become cash cows like happens with IC design tools for any particular process node as time progresses. Like the red queen in *Alice through the Looking Glass*, it takes all the running one can do to stay in the same place.

There may be change coming as more and more FPGA designs are actually prototypes for ASIC designs, or might want to be cost-reduced into ASIC designs and so on. This means that people want to use the same tools for ASIC and FPGA design, and on the surface is one reason Synopsys acquired Synplicity.

One other issue is that it is FPGA architectures and their tools are more intimately bound together than with IC designs. It is a dirty

secret of synthesis (and maybe place and route) that despite the lower price point, FPGA synthesis is harder, not easier, than mainstream synthesis for standard-cell libraries. Solving the general problem for all the different varieties of FPGA architectures seems to be extremely costly. By contrast, Xilinx only has to build tools that work with Xilinx and can ignore that its algorithms might not work with Altera's arrays.

But probably the biggest factor is that there are not enough FPGA companies. If there were a dozen FPGA companies then enough of them would compete by supporting a 3rd party FPGA tool industry and eventually the investment there would overpower the companies that tried to keep it internal. This is just what happened when the Japanese and everyone else tried to get into ASIC. They had no internal tools so they leveled that playing field by making Daisy/Mentor/Valid and subsequently Cadence then Synopsys successful. Eventually companies like VLSI Technology and LSI Logic felt they should try and spin out their tools and adopt EDA industry tools.

It is unclear whether this was good financially. I told Al Stein, CEO of VLSI, when we spun out Compass that he shouldn't expect his tool bill to go down. He would pay more to the EDA industry that Compass was entering (some of it to Compass) than he had been paying just to fund the division that became Compass. And this turned out to be a true prediction.

For ASIC designs today, IBM's backend tools are the only ones internally developed. But they are #1 in cell-based design so it is hard to argue that the strategy of keeping that development internal is demonstrably bad.

And Xilinx and Altera are doing OK keeping their tools internal.

Wall Street Values

Wall Street does a terrible job of valuing investment. I've talked earlier about how financial accounting standards do a poor job of capturing the value of many modern companies in their balance sheet. But Wall Street is driven by people who only know how to read a balance sheet (and a P&L which is just an explanation of

changes to the balance sheet) and they act as if the balance sheet is the truth.

As a result, Wall Street loves acquisitions and hates investment, even if the investment is much cheaper than an acquisition.

Assume bigCo acquires startupCo for $100M. FASB considers the acquisition to have something to do with the business going forward as if it were a piece of land just purchased that needs to be carried on the books and have its value adjusted from time to time. But in reality it should adjust the prior quarters' P&Ls to reflect the fact that all that R&D that was done should really have been set against revenue back then. When we were allowed to account for acquisitions through pooling of assets, it was closer to this but still got stuck with the goodwill which really also should be partially set against prior quarters. Anyway, Wall Street loves this sort of deal, whatever the price, since it is seen as a write-off (purchase price) leaving a leaner cleaner company to make more profit going forward. It doesn't care about prior quarters anyway.

By contrast, if bigCo instead spent $1M per quarter for the previous couple of years, which is much less than the $100M it acquired startupCo for, then Wall Street would have penalized it by lowering its stock price due to the lower profitability. Since the investment doesn't show on the balance sheet it is a pure loss with no visible increase in anything good. Of course it is hard for anyone, especially financial types on Wall Street, to know if the investment is going to turn out to be Design Compiler (good) or Synergy (not good), if it is Silicon Ensemble (good) or Route66 (not good). But the same could be said about any other investment: is that expensive factory for iPods (good) or Segways (bad).

When a company goes public, it sells some shares for cash, so ends up with lots of cash in the bank. But it then finds that it is hard to spend that cash except on acquisitions. If it invests it in R&D, then the net income will be lower than before and so the share price will decline from the IPO price due to the reduced profitability. If it uses it to acquire companies, then prior to the acquisition its profit is high (no investment) so its stock price is high. After the acquisition its profit is high (new product to sell

with largely sunk costs). At the acquisition, Wall Street doesn't care because it is a one-time event and Wall Street never cares about one-time events. Even if, like emergency spending in Congress, they happen every year.

I think it is bad when accounting rules in effect force a company to make tradeoffs that are obviously wrong. It is obviously better to develop a tool for $10M than buy a company that builds the same tool for $100M. Yet Wall Street prefers it, so that's what happens. Of course there is a difference between developing a product that might succeed and buying a company that is already succeeding, and I've talked elsewhere about the issues of getting products into the channel. So the acquisition might make sense anyway. But accounting skews those decisions too much.

Royalties

Venture capitalists love royalties. They love royalties because they think that they might get an unexpected upside since they are hoping that a customer, in effect, signs up for a royalty and sells far more of their product than they expected and thus has to pay much more royalty than expected. Since a normal successful EDA business is predictable (license fees, boring) it doesn't have a lot of unlimited upside.

My experience is that you need to be careful how to structure royalty deals to have any hope of that happening. At VLSI I remember we once had a deal to build a chip for one of the Japanese game companies if we could do it really fast (we were good at that sort of thing). It needed lots of IP so we just signed everyone's deal with no license fees for as long as possible, but which all had ridiculous royalty rates. We probably had a total of about 25% royalty on the part, more than our margin. But we reasoned as follows: "One in three, the project gets canceled and never goes into production (it did); one in three it goes into production, but we never ship enough volume to reach the point we pay more in royalties than we would in license fees; one in three it is a big success and we tell everyone the royalty they are going to get, and if they don't accept, we design them out."

IP is more mature now, so the royalty rates and contracts are more realistic. Back then the lawyers could consume the whole design cycle negotiating terms and there wasn't enough time to wait. Everyone thought their non-differentiated IP was worth a big royalty of a few percent, even though a big SoC (even then) might have dozens of IP blocks that size. So perhaps the problem has gone away. If you were on a short time to market, you simply didn't have time to negotiate terms or royalty rates. You had to get going. Hence our strategy of accepting everything on the basis we'd renegotiate if it became important.

The best form of royalty is one that is paid out elsewhere in the value chain. If TSMC pays a royalty to Artisan (now ARM) or Blaze (now Tela) then the customer ends up paying since TSMC adds it to their bill. But if the chip is a success then the customer doesn't get to re-negotiate from a position of strength. If the overall deal is a huge success then TSMC has probably negotiated percentage breaks anyway (I don't know any details so I'm guessing) and can choose either to reduce the cost it charges on or take a bigger profit.

However, when Artisan was bought by ARM for $900M every VC wanted a deal like that. And they saw royalties as the secret sauce. But royalties only work really well when they are much higher with unexpected success, otherwise they are just moving payments around in time. Plus they only work well with unexpected success if they can't be renegotiated as a result.

The reality is that they are often disappointing. Mike Muller, CTO of ARM once told me that "royalties are always later and less than you expect." It took Artisan 15 years to get to the stage it was sold to ARM, those royalties were largely licenses fees foregone in the early years. ARM was, in effect, paying for money pushed into the future that it would then collect.

At one level, many people say that it is a pity that EDA doesn't get a percentage of semiconductor revenue. But in a way, they do. EDA has been around 1.5-2% of semiconductor revenue for years. Of course EDA would like that number to be 4% but it's unlikely that semiconductor companies would have signed up for the deal of no upfront money and big royalties, even though they would probably have been served well by it. They would have

avoided any of the business impact that many companies suffer due to having inadequate numbers of licenses.

SaaS for EDA

SaaS, or software as a service, is the capability to deliver software over the net. In web 1.0 years this was called the ASP model, for application service provider. The archetypal company doing this is Salesforce.com, which provides customer relationship management (CRM) software for businesses, initially small and medium sized. This was in comparison with companies like Siebel (now part of Oracle) who used the traditional installed software model, basically the same model as almost all EDA companies use.

So will SaaS take over EDA? The advertised advantages of SaaS are a lower cost of sales, faster update cycle for the software, and that it can be incrementally adopted one seat at a time. And several EDA companies, some small, one called Cadence, have announced SaaS offerings already.

I think the attraction of SaaS for users comes from a misunderstanding: that with SaaS, which is one kind of "metered use" of tools, the tool bill would go down. This seems unlikely. One problem with all kinds of metered use for EDA is that the large users have licenses that run 24 hours per day, and small users just use tools occasionally (for example, during tapeout). If the tool/hour is priced so that the heavy users pay a little less than before, then occasional users pay almost nothing. If the tool/hour is priced so that the occasional users pay a little less than before, then the heavy users prices go up by multiples of the current cost. There is no good in-between price either. SaaS doesn't get around this issue. And if many people don't pay less than before they are not going to adopt metered use, SaaS or otherwise.

I think that the dynamics of the business process are very different for EDA. One assumption in SaaS is that by lowering the barrier to entry to a single seat bought over the net, as opposed to a corporate deal, the market can be expanded to people who corporate deals don't reach, or at the very least it will steal significant market share from the big guys. This was

salesforce.com's base, initially selling to people who would not have bought a CRM system, and then stealing users from the big guys. It is classic Innovator's Dilemma disruption, starting not by stealing business from the established market leaders, but going where the competition is "non-use."

But most EDA is not like that. There is no crowd of unsatisfied IC designers just waiting to build chips if only place & route were cheaper. And as to undercutting the big guys, any innovative business model that turns EDA into a smaller market is likely to reduce investment in EDA, which gets problematic very quickly. Stealing a market segment Craigslist style, by turning a billion dollar market into a million dollar market and dominating it, will not be able to sustain the R&D needed. The reality is that you can't compete much on price in EDA: there is no market to expand into, and if you succeed in stealing a lot of market at a low price then you had better genuinely have lower costs (especially in R&D) to be able to do it again for the next process node. It is a similar problem to the one in pharmaceuticals. Drugs cost a lot less to make than they sell for, but if you artificially (by fiat or excessive discounting) reduce the prices too much then current drugs are cheap but no new ones will be forthcoming. Unlike with drugs not developed, if there are no workable tools for the next process node then we will all know what we are missing; it is not just a profit opportunity foregone, it is a disaster.

The next problem with EDA is that you can't get the job done with tools from only one vendor. So if you use SaaS to deliver all your EDA tools, you will repeatedly need to move the design from one vendor to another. But these files are gigabytes in size and not so easily moved. So it seems to me that if SaaS is going to work, it has to be through some sort of intermediary who has all (or most) tools available, not just the tools from a single vendor. If you use a Cadence flow but you use PrimeTime (Synopsys) for timing signoff and Calibre (Mentor) for physical verification then this doesn't seem workable unless all are available without copying the entire design around.

Another problem is that SaaS doesn't work well for highly interactive software. Neither Photoshop nor layout editors seem

like they are good candidates for SaaS since the latency kills the user experience versus a traditional local application. Yes, I know Adobe has a version of Photoshop available through SaaS, but go to any graphics house and see if anyone uses it.

There are some genuine advantages in SaaS. One is that software update is more painless since it is handled at the server end. You don't normally notice when Google tweaks its search algorithm. But designers are rightly very wary of updating software during designs: better the bugs you know than some new ones. So again, EDA seems to be a bit different, at least has been historically.

The early part of the design process and FPGA design are a better place for SaaS perhaps. The files are smaller even if they need to be moved, the market is more elastic (not everyone is already using the best productivity tools). But this part of the market already suffers from difficulty in extracting value from the market and SaaS risks reducing the price without a corresponding true reduction in cost. Walmart is not the low price supplier because it has everyday low prices; it has everyday low prices because it has got its costs lower than anyone else's. Perhaps the ideal market is FPGA design where the main tool suppliers are not really trying to make money on the tools directly, and where few of the designs are enormous.

So if SaaS is going to succeed in EDA, my guess it that it will either be a virtual CAD organization offering tools from many vendors, or else in the FPGA world where single-vendor flows are common.

Why are VCs so greedy?

Why are venture capitalists so greedy? Why do they want a 20-30X return on their money? Why doesn't investing $5M and selling the company for $15M a couple of years later make them happy? After all, when I've bought a stock and sold it for three times what I paid, I'm pretty pleased with myself.

To understand the reasons, you need to know a little about how venture capital funds work. There are two lots of people involved, the general partners, who are the people who work for

the venture capital company; and then there are limited partners (LPs), the people (insurance companies, pension funds, whatever) that put up the money to be invested. The general partners may pay themselves 2% of the value of the fund per year a management fee, plus 20% of any profits.

One critical factor is that the fund has a lifetime, maybe 10 years. A fund would like return at least 20% per year. After all, the stock market has returned nearly 10% since 1900, including a great depression and the current downturn, and with a lot less risk. VCs should do at least twice as well as that.

The money isn't actually all put into the fund by the LPs on day one, and taken out on the tenth anniversary. As the VCs find companies to invest in, they make capital calls on the LPs to get the money. If and when there are successful "exits", meaning that portfolio companies are sold or go public, then money is returned to the LPs. To keep the math simple, though, let's calculate returns as if all the money were invested early, and all the payouts arrive late.

There is one thing about VC investments that is different from you making an investment in the stock market but that is not often explicitly talked about: the venture fund (normally) only gets to invest the money once. This is a big difference from other types of investors, and is one of the reasons that you are happy if your stock triples in a couple of years and you sell it, and a VC is not. You can do something else with the money for the next 8 years and make more money. The VC typically cannot, it is returned to the LPs.

So let's do a bit of math. Let's say there is a $100M fund with a lifetime of 10 years. To keep things really simple, let's ignore the management fees and the carry, the percentage of profits that the VCs retain to buy their Ferraris. A return of 20% per year means that the $100M fund needs to return about $600M in total (that's simply 20% per year compounded for 10 years).

But not all investments will turn out to be wise. VCs, by definition, are investing in risky companies and at most 15-20% will make money, and often fewer ("fund 20, pray for 2"). So the

$20M that turns out to be invested in great companies needs to generate $600M, meaning a 30X multiple.

That's why VCs are so greedy. They have to get a 30X return on the good investments to make their numbers. Getting a 3X return in 2 years doesn't do much to help them, even though it might be great for the founders, early investors and employees. If they have a company that is doing well enough to get an acquisition offer yielding a 3X return in 2 years, they will prefer to keep on being independent, and hope the company continues to do well and can generate a 30X return (or more) during the remanining lifetime of the fund. They are all like Barry Bonds was: home-run or strikeout. Getting on first base is just not that interesting.

Term sheets

What is a term sheet? If you raise money from a venture capitalist (or an experienced angel) then the most important conditions for the investment will be summarized in a term-sheet. It sounds like this should be a simple document of a single sheet of paper, but in fact these days it is dozens of pages of legalese that is a good way towards the final legal documents. In fact, it is so complex that typically the really really important stuff will indeed be summarized in a few bullet points ln a different piece of paper (or an email).

The most important bullet point, of course, is the pre-money valuation. This is the amount that the investor values your company before putting in the money. The post-money valuation consists of this amount plus the money that they invest.

One item to make sure you understand is the employee option pool. There will be a pool of options created for future employees. The normal way is to carve this out from the previous round before the new round money goes in. This means you pay for the option pool, not the new investors. At one level this sounds unfair, but in fact it just feeds into the valuation. The pre-money valuation is thus usually the valuation of the previous round's stock, plus the value of option pool. So the previous round's stock is not worth as much as it sounds. Of course this can be done other ways, but that will affect the valuation

differently. Make sure you understand it since not all "$8 million" pre-money valuations are created equal.

The next most important item is probably whatever is agreed about liquidation preferences. The VC will have preferred stock, and preferred means that it gets a better deal than common stock in the event of an acquisition (or wind-up) of the company. Just how much better a deal is important. At the very least, usually a VC wants to get their money back before the common stock gets any. But it can be a multiple of their money they get back, or their money plus a dividend (interest). And then they get their share of the common, which there are a number of ways to do.

Another really important area, especially when there are multiple rounds, is who has to approve change-of-control decisions (essentially acquisitions). Can the VCs over-ride the common stock? Can series B veto series A? In one acquisition I was involved with, each round of preferred stock voting had to approve the acquisition, not just the preferred stockholders voting together as a group, which is not unusual. Sounds innocuous? But one round of investment was a corporate investor (a customer) who thus had veto rights over the acquisition (that they leveraged into a very good deal for themselves since they were also a customer of the eventual acquirer).

An excellent book on term sheets is *Term Sheets and Valuations* which goes through the most important items on the term sheet and gives you guidance for each item as to what is middle-of-the-road, what is aggressively in favor of the venture capitalist, and what is aggressively in favor of the company (or common stockholders). I've purchased this book several times since my copy always goes missing when someone forgets to give it back. If you are going to raise money and have not done it dozens of times before then you should read this book.

Another really good resource is that Wilson-Sonsini, the biggest name in legal for most silicon valley stuff, has put a term sheet generator online. You fill in various pages with the data, and it spits out the legal document. If you are really raising money, you should at minimum have your lawyer go over the document, it's still just a document generator on the web. But if you are thinking about raising money, or want to know more about term-sheets,

then you can play with the tool. Even going through the experience of filling out the forms will force you to understand a whole lot of issues that you might not have come across.

There's also lots of stuff on various venture capitalist blogs. This will often give you an insight into why a VC might care about a particular issue, when it would be important, how big a concession it would be for them to give you more favorable terms and so on.

As always, remember that the VCs have usually done this lots of times before. You, not so much. Get good advice.

The antiportfolio

You have to be pretty brave to be a venture capitalist and keep an "anti-portfolio" page on your website. This lists the deals that you were offered but turned down. Bessemer Ventures is the only VC I know that does this. They've had some great exits over the years (such as Skype, Hotjobs, Parametric Technology or going back, Ungermann-Bass). But they also turned down FedEx (7 times), Intel and Paypal. And here's their description of another one that got away:

Cowan's college friend rented her garage to Sergey and Larry for their first year. In 1999 and 2000 she tried to introduce Cowan to "these two really smart Stanford students writing a search engine". Students? A new search engine? In the most important moment ever for Bessemer's anti-portfolio, Cowan asked her, "How can I get out of this house without going anywhere near your garage?"

Most of us who've been in Silicon Valley for a long time also have our own sort of anti-portfolio: the companies that we could have worked for and didn't. I've never interviewed and then turned down a job that turned out to be a really huge mistake, but I have been invited to come and interview and refused.

When SDA (the fore-runner of Cadence) was founded, a friend of mine, Graham Wood, actually someone I shared an office with doing my PhD at Edinburgh, relocated to California from Bell

Labs to join them. He was about employee number 20 and went on to be the creator of SKILL, still significant today in the battle over Cadence's Virtuoso franchise. He asked me to come and interview. But at the time I was happy at VLSI Technology and thought we were going to change the world. In a way, of course, we did. But I wasn't smart enough to see that the real money in ASIC was not in building the chips but in building the software to build the chips. Plus, if you are going to write EDA software, you are may as well do it at an EDA company where you are regarded as valuable, rather than at a semiconductor company where you are regarded as a weird item on the expense line.

Wes Patterson ran VLSI's design centers and he left to be CEO of Xilinx. Somebody, not Wes himself, invited me over to interview but I was too blind to see that they would become a big success. "RAM-programmable gate-arrays? Who'd use one of those? How big a market is that?" Again I failed to realize that the cost structure might not have been great at first, but it was only going to get better. If you only want a handful of parts then FPGAs are the only sensible solution. For high volume, ASIC is the way to go. But over time, the cutover point creeps up and up and FPGAs serve more and more of the market.

Many years later I was headhunted by Xilinx to come and interview to run their entire software business and make it profitable. I wasn't sure how feasible this was. Based on my experience at VLSI you can't run a real software P&L inside a semiconductor company since you are an enabler for silicon and, if a customer is important, the company will just give everything for free, but not compensate your P&L with some of the silicon profits you enable. I interviewed with Wim Reolandts, the CEO who'd recently joined from HP. They asked me to come back in but that same week I was asked to become CEO of Compass. Who knows which would have been the better opportunity? Being a CEO is really hard unless you have the one qualification that everyone wants: you've been a CEO before. So it is always a good idea to take it when fate offers you that opportunity.

After Ambit was acquired by Cadence, Al Stein, VLSI's CEO, tried to interest me in running a venture capital portfolio for VLSI. It was all the rage then for companies to take some of their

cash and try and use their specialized inside knowledge to pick some winners for investment. I went and talked to the guy who ran the equivalent fund for Adobe, who'd started the trend. He'd invested in Netscape and other early internet successes and been wildly successful. But in the end I stayed at Cadence since I wasn't convinced I'd be that good a venture capitalist. In retrospect I should probably have taken the job. I'm sure it would have been really interesting irrespective of how successful I turned out to be, and jobs are really interesting when you are learning a lot. In fact, if I'd taken it, it would have been fairly short lived since soon after Philips Semiconductors (now NXP) bought VLSI.

One job I did interview for and eventually turn down wisely, was to help run the software arm of European Silicon Structures. This was a company set up in Europe (duh!) to use e-beam technology to do very small runs of wafers cost-effectively. I turned the job down when the CEO couldn't convince me of a good reason to have a large well-funded software division since clearly the company stood or fell based on how good the e-beam technology turned out to be. By then I'd got smart enough to know that you don't want to be in an "expense" department in a semiconductor company. It turned out the e-beam technology didn't work that well and the company failed. I think Cadence picked over the bones of the software division.

I was never offered a single digit badge number job at Google or anything like that. But it is always hard to tell which jobs are going to be with companies that turn out to be wildly successful. I asked a friend of mine who worked for me briefly as my finance guy before going on to be the CFO of Ebay and lead the most successful IPO of all time what was the most important criterion for success: "Luck."

CEO pay

If you are an investor, what do you think the best predictors for success for a startup are? If you could pick only one metric, which one would you use?

Peter Thiel, who invested in both PayPal and Facebook so seems to know what he is doing, reckons it is to examine how much the CEO is paying him or herself.

Thiel says that "the lower the CEO salary, the more likely it is to succeed."

A low CEO salary has two effects, both of them important. It means that the CEO is focused on making the equity of the company valuable, rather than attempting to make the company last as long a possible to collect a paycheck.

The second effect is that the CEO's salary is pretty much a ceiling on the salaries of all the other employees and it means that they are similarly aligned.

The effect of those two things together means that the cash burn-rate of the company is lower, perhaps much lower, and as a result either extra engineers can be hired or the runway to develop and get the product to market is longer.

When Thiel was asked what was the average salary for CEOs from funded startups he came up with the number $100-125K. For an EDA startup, this seems pretty low since it is much lower than good individual contributor engineers. I have seen a report that an EDA or semiconductor startup CEO should be paid around $180K (plus some bonus plan too). On the other hand, maybe Peter Thiel is right. How many EDA and semiconductor startups have been that successful recently?

A good rule of thumb in a startup is that the more junior you are then the closer to normal market salary you should get. There are two reasons for this: you can't afford it and you don't get enough equity to make up for it. If you are on a $100K/year salary at market, you probably can't afford to work for $50K/year. If you are an executive at a big EDA company making $400K/year you can afford to work for under $200K/year. If the company makes it, the vice-presidents in the company will have 1-2% equity, which is significant. The more junior people typically not so much (at least partially because they are that much more numerous) unless the company managed to bootstrap without any significant investment.

Thiel has a company, younoodle, that (among other things) attempts to predict a value a startup might achieve 3 years from founding. It is optimized for internet companies that have not yet received funding, so may not work very well for semiconductor or EDA startups. And guess one of the factors that it takes into account when assessing how successful the company will be: CEO pay.

Chapter 6: Books

Innovator's dilemma

The Innovator's Dilemma is a book by Harvard business school professor Clayton Christensen. I highly recommend the book both as one of the most stimulating and best-written business books (I know that is an oxymoron, but this is a book you will really enjoy reading). The basic thesis is that there are two kinds of innovation, sustaining (giving high-end customers what they want) and disruptive (giving a new set of less demanding customers something less that state-of-the-art). Sustaining innovation eventually gives people more than they want at a premium price point, but disruptive innovation often improves faster and eventually steals the main market from below when its basic capability addresses the mainstream at a lower price.

Here's an example: Digital Equipment Corporation (DEC) built Vax computers in the 80s. Customers wanted more and more powerful Vaxen and had no use for the IBM PC when it came out, a low-powered machine that didn't even have a floppy disk as standard, let alone a hard disk, when it came out. But eventually the PC destroyed DEC's business (and it will almost certainly destroy Sun's) as it got more powerful. The dilemma is that it is unclear what a company like DEC (or Sun today) should have done. They were not stupid, they could see the PC coming, and they even made several attempts to enter the PC market themselves. But it was of no use initially to their primary customers and they didn't really have the capability to sell to the people who could make use of early PCs. By the time the PC was powerful enough to be of interest to the scientific and business computing segments, where DEC sold most of its kit, it was too late. Other companies (Compaq, Dell etc) were already established as the leaders and DEC was eventually dismembered with part going to Intel and part going to Compaq and so ending up inside HP. It is not that it was or is impossible to built a computer more powerful than a high-end PC, it is that the cost-

differential is so large that very few applications justify paying such an enormous premium.

Clayton's book has some other lovely examples: cable driven earth-moving equipment being driven out by hydraulic; steel mini-mills making rebar and gradually working up to high-grade sheet steel and so on.

When I was at Cadence we had an annual engineering conference, a mixture of presentations of papers that could not be presented externally due to confidentiality, social getting together of engineers from dispersed sites and an opportunity to address a lot of engineering in one place (I think about a third of all Cadence engineers attended). Professor Christensen was one of the keynote speakers at one meeting and he was also a fascinating speaker.

One thing he discussed a bit was the end of Moore's law. He predicted that Moore's law would end because it would deliver more capability than the mainstream required at a price that was higher than the mainstream wanted to pay. This was already happening in the PC marketplace where for some time microprocessors have been "fast enough" for almost all applications (whereas through most of the 1980s and early 1990s people would upgrade their PC regularly simply because the old ones lacked oomp).

I think it is clear now that the mainstream PC market in its own turn is going to be disrupted from below by iPhone like devices. iPhones will get more powerful until most of what a PC is used for can be done on an iPhone (or a Google Android-based phone or a Nokia one; I'm just using iPhone as shorthand). Of course they don't have big screens or keyboards but if my office and home had them, then my powerful future iPhone would simply work from my pocket when I was nearby. Or maybe it will project onto my retina or sense the muscles in my fingers or something. Who knows?

For many systems, FPGAs are disrupting ASIC from below in traditional innovator's dilemma style. Nobody does an ASIC unless they absolutely have to, which either means an enormous amount of integration, enormous volumes, or low-power

requirements (which is the Achille's heel of FPGAs). If you can use an FPGA then you will.

Moore's law has been driven for decades by semiconductor economics. It was always 30% or more cheaper to use the new process generation than stick with the old one. But it is not clear whether 28nm (and 22nm or whatever comes next) will have such a cost reduction. Maybe 22nm is going to be the mainframe of semiconductor processing, very expensive and delivering more capability than the mainstream market can take advantage of. The mainstream will hold back in older processes and use clever software to get what they want; after all, most chips these days are just substrates for running the software of the system.

The book that changed everything

Until 1979, IC design was done by specialists who understood every aspect of the design from semiconductor fabrication, transistor characteristics, all the way up to small blocks of a maybe a thousand gates which was the limit of chip fabrication in that era. In the late 1970s this "tall thin man" approach started to break down. Design was getting too complex for people who understood the process to do it, and the process was getting sufficiently complex to become the realm of its own specialists.

Everything changed thirty years ago with the publication of Mead and Conway's book "*Introduction to VLSI systems*." It is out of print but it was the most influential book in semiconductor design and design automation ever.

Mead and Conway separated design from manufacturing by creating simplified design rules for layout, and a simplified timing model suitable for digital design. No longer was it necessary to understand every nuance of the fabrication process, no longer was it necessary to consider every transistor as an analog device. The most important aspect of this is that it meant that computer scientists could design digital chips since they no longer needed deep electrical engineering knowledge.

I was at Edinburgh University at the time, finishing up my PhD (in computer science, not electrical engineering). John Gray, who

had run Carver Mead's silicon structures project at Caltech on sabbatical from Edinburgh returned carrying galley proofs of the yet-to-be-published Mead and Conway book. He ran a course based on this, one of the first on IC design in the UK I presume, heavily attended not just by the students who were meant to be on the course but by many of us faculty too.

Design became the province of computer scientists who understood enough about layout, enough about timing, enough about architecture and enough about test to successfully create state-of-the-art chips. Indeed, they could do so more effectively than the electrical engineers since chips were getting to be too large to do entirely by hand, and computer scientists already knew how to deal with complexity. They also started to create the first EDA tools, simple layout editors, simple simulators, rudimentary design rule checkers, because their natural instincts were never to do anything by hand if you could create a program to automate it.

Mead and Conway's book created a cohort of IC-literate computer scientists who went on to populate the CAD groups of the semiconductor companies and, eventually, the EDA industry once it got going.

To see how big a difference it made, look at analog design versus digital design today. Analog design is largely done today the way digital design was done until Mead and Conway: deeply expert designers with the raw process models, raw design rules and polygons.

The next big change would be the invention of Verilog and RTL synthesis that meant that computer scientists could design complex chips with almost no knowledge of how chips worked, what a transistor was, how a chip was made. This new layer meant that front end designers and back end designers were different people with different skill-sets.

We seem to be on the cusp of another such layer with ESL tools starting to become much more widely used, allowing designers with very little hardware knowledge at all to create complex systems. The layer above that is software, already well-understood and with its own culture and tools.

Relevance lost

If you are at all interested in accounting I recommend the book *Relevance Lost* by Thomas Johnson and Robert Kaplan. I think it is a fascinating background to how we ended up with the kind of finance departments we have, but I admit it might be a minority interest. I had a girlfriend once who was in finance and I couldn't even interest her in reading my copy.

Although published in 1991 it is still in print. It covers how accounting used to be useful to managers, starting with New England mill owners in the 19th century. However, as the accounting rules and processes were hijacked by financial accounting they have become steadily more and more useless for managing the business. Nobody wants to keep multiple sets of books so managers try and manage using accounts put together for financial accounting reasons on a timescale driven by financial accounting deadlines.

The situation is even more disconnected in the case of a software or design company. Much of the real value of the company is bound up in partially or completed software products (or designs). The rules for capitalizing development are so strict that it must only be done when the product is pretty much released. Almost all the development is written off as an expense as if it were part of the utility bill, as opposed to an investment building up value in the company. From a point of view of keeping the tax paid by the company low this may be desirable; from the point of view of the balance sheet giving a useful assessment of the company not so much. Design Compiler is clearly a major asset of Synopsys but you won't find it on the balance sheet anywhere, either as an estimate of its value as a forward looking business or even as a rollup of the cost of development over the years.

Other intangible assets, such as an effective high-skilled development team, appear nowhere. If a key employee leaves the value of the company may well have changed in a meaningful way but this is nowhere reflected. It is completely unclear how one would actually account for this in any sensible way, of course, but it sort of happens anyway. Look at the change in market cap of Apple when Steve Jobs is thought to be sick or not,

which is actually the value of the asset of having Jobs as CEO that in principle should be on the balance sheet somewhere.

Software companies seem to have very lax financial controls in my experience. I worked for over ten years at VLSI Technology, a semiconductor company. That is a business in which a lot of money flows around but the margins are thin. Fabs cost (today) billions of dollars so getting the accounting right is important. The financial controls and forecasting in a semiconductor company are generally very good. When we spun Compass out we were still consolidated into VLSI's books and we kept the finance we were used to. Every manager did an expense forecast for 6 months ahead, and monthly we looked at variances to that and were expected to explain them. Startups are small enough that their financial controls, at least for cash, are usually pretty good. But when I got to Cadence I was surprised that even as the manager of the custom IC business unit (then a $250M/year business) I wasn't expected to forecast my expenses, it was hard to even find out what they were, and as a result they were pretty much whatever they turned out to be. The concept of over-spending didn't exist. I assume that has changed somewhat now that the financial outlook is less rosy, but that sort of thing is part of the DNA of a company and is actually quite hard to change.

Crossing the chasm

The most influential book on hi-tech marketing of the last twenty years or so has to be Geoffrey Moore's *Crossing the Chasm*. I doubt that there is anyone in marketing reading this blog who has not read it. In fact everyone in hi-tech should read it since it affects not just how products are marketed, but how they are developed, where investment needs to be made and how, and generally what is required for a hi-tech product such as an EDA tool, software product, semiconductor chip or a system. If you are in engineering wondering why your product marketing manager is insisting that you stop work on the new whizzy feature for the next version in order to make sure that the current version reads

some obsolete format of library then this book makes it clear why.

The key insight of the book is that the mainstream buys for different reasons than early adopters. As a result, it is much harder than you would expect to turn success with early adopters into success with the mainstream. Getting from this early success to the nirvana of mainstream adoption is crossing the chasm, the chasm being the fact that you can burn all of your money trying to get across unsuccessfully if you ignore what is necessary for success.

The big idea in *Crossing the Chasm* actually comes from an earlier book by Bill Davidow published way back in 1986 *Marketing High-technology* (still in print), which first introduced the idea of the "whole product." However, Geoffrey Moore did a much better job of explaining it and the chasm metaphor was a much more viral image.

Early adopters will do their own work to make up for deficiencies in your product, especially tailoring it to work in their environments, adding missing scripts or libraries and generally working out how to get the most value out of your product. Mainstream customers will not do that. You need to deliver them everything that you need, the whole product. You may not need to deliver all this yourself, but you need to create an ecosystem so that everything is available.

A good example is the early days of Synopsys. You can sell a synthesis product like Design Compiler (DC) to a few early adopters on the basis that they will do their own work to take existing simulation cell libraries and manually create the .lib libraries necessary for synthesis. However, the mainstream will not. The mainstream wants the whole product, one that they can use from day one. This means not just DC but also .lib libraries for whichever library they happen to want to use for fabrication. So in the early days Synopsys had a huge group of engineers creating these libraries for the ASIC vendors. I think Bob Dahlberg, who ran the group, told me that it peaked at 200 people. Within a year the ASIC vendors realized that they needed to do this job themselves since they didn't want Synopsys's

library group to be on their critical path to revenue from a new process node.

This shows another point, that once you start to achieve success in the mainstream, you become part of someone else's whole product and they need to support you to be successful themselves.

The whole product becomes a barrier to entry too. Once Synopsys had all the ASIC vendors on-board, they were not likely to want to create libraries for other synthesis tools. So Mentor's Autologic, Compass's ASIC Synthesizer, Trimeter, SILC and all the other struggled not just because Synopsys could invest more in developing synthesis but also because nobody else could get the whole product together easily.

Chris Wilson, the CTO of NuSym, gets all this but isn't sure what to do about it. He complains that there is now so much infrastructure required in a simulator (3 versions of the API, several testbench languages, Verilog, VHDL, SystemVerilog, C) that it takes all their effort just to do that and very little is used to deliver the core differentiated technology. Of course it would be convenient for him if someone else provided all that so that they can focus on their core technology, but nobody does. Synopsys didn't want to develop ASIC libraries either. But he knows he won't be successful without full compatibility.

Coincidentally, both Bill Davidow and Geoffrey Moore both ended up in the same venture capital firm, Mohr-Davidow Ventures (MDV). When we finally got MDV over the finishing line to invest in VaST while I was there, we ended up being invited to a half-day meeting with Geoffrey Moore. This lead to a dysfunctional conversation since I knew that the Mohr in the name of the company was spelled differently so I figured that somebody was confused about who we were meeting. But I was wrong: Geoffrey Moore was (and is) a partner of MDV. At that point VaST was having some early success in a handful of companies, mostly in Japan, and so we spent a very interesting afternoon brainstorming how we could create an ecosystem of models which we all knew was the main barrier to getting across the chasm.

Mr Rodgers goes to Washington

T.J. Rodgers, the CEO of Cypress Semiconductor, is also a critic of government intervention in the economy, especially that of Silicon Valley. Whatever you think of the strategic decisions that he made at Cypress, most of which look pretty good in hindsight, he is a great writer. I wish I could write as clearly and interestingly as he does.

If you haven't read it then I highly recommend reading every word of his 1993 testimony to Congress in the Clinton era, "*Free Market or Government Subsidies?*" It is especially worth reading in the light of the current extensive intervention in the economy by the government in all sorts of arbitrary ways. Luckily they aren't intervening much yet in technology. Of course, on one level, it would be nice to get some stimulus money, but without the interference that comes with it.

One area that technology that does have a large government dimension, at the very least in competing for the same VC money, is the environment. I regard most of the current venture-capital investments in "green" technologies largely as bets on governments subsidizing them whether they are economic or not. For instance, did you know that Germany, that famously sunny country with its enormous deserts, is the biggest installer of photovoltaic solar power, accounting for almost half the world market? And with the senate dominated by states that have few people but lots of agriculture, there are no prizes for which country has the most subsidies for turning expensive food into ethanol, a nasty fuel that corrodes pipes, attracts water and produces lots of aldehyde pollution at the tailpipe. Not to mention uses roughly as much energy to produce as it generates when used.

T.J. Rodgers's plea to Congress in 1993 for a balanced budget (given that the budget was eventually in surplus on Clinton's watch) looks absurd today, given the way we are racking up an unpayable tab. But his testimony from 15 years ago stands up really well today.

Even more recommended is T.J. Rodgers's piece for the Cato Institute, "*Why Silicon Valley Should Not Normalize Relations with Washington*," that does a great job of contrasting what he, T.J. Rodgers, worries about on a daily basis, with what Dwayne Andreas, then-CEO of Archer Daniels Midland worries about. T.J. Rodgers worries about semiconductor technology, borrowing money, how much to invest in solar, which products to design. Andreas worries about…well, making sure that Congress passes the right laws to ensure that he can farm the subsidies. ADM is the largest beneficiary, even before the last changes of the law, of the policy of insisting that a certain amount of ethanol gets added to gasoline.

Early exits

I came across the book *Early Exits* recently. It is definitely worth a read, especially for anyone having anything to do with EDA startups. An early exit is one after a relatively small number of years at a relatively small multiple to the original investment. As VCs don't like this sort of deal in general. They need big returns on at least a few of their investments and they don't care that much about the rest. Early exits don't interest them.

There are a number of reasons that EDA startups are not getting funded by venture capitalists any more. The most obvious is that large EDA companies have stopped buying them at high valuations. This is for a mixture of reasons but one is that standalone tools are harder to ramp up profitably without tightly integrating them into the main body of pre-existing tools. For example, standalone statistical static timing is interesting, but much more important is integrating statistical static timing into the synthesis, place and route flow. Don't just find the errors after the fact, stop them occurring in the first place.

But a second reason that EDA startups are unattractive is that they don't require enough money. Venture funds are growing larger and it is a fact of life that being on the board of a company looking after a $3M investment is about the same amount of work as looking after an investment of $30M. If a fund is large, it can't afford to dole it out $3M at a time; that requires too many

investments. Instead, fewer but larger investments are required. This means that the size of investment is too large for an EDA company. Too large in two ways: too large since EDA tools don't require that much capital to develop, and too large since the exit price required to make the investment successful is higher than is likely to happen.

I've been somewhat involved with several startups recently who are looking about how to raise a little money. Relatively small amounts are needed and venture capitalists are simply not the place to go looking. Individual investors (angels) and corporate investors (customers) are much more likely. New technology continues to be needed and this type of investor can live with the likely return on a successful company.

The new rules are raise only a tiny amount of money, run the company on a shoestring, validate the technology with some initial sales and exit earlier rather than later. If you wait, you will need more money to build a big channel, and any acquirer will have to tear it down anyway. EDA startups spend more money building sales channels than technology, and one thing Cadence and Synopsys don't need more of is channel.

One thing that the book points out is something I'd not really thought about. The sales cycle for an EDA tool is about 9 months. What do you think the sales cycle for an EDA company is? More, a year or two from first contact to closed deal. If you are going for an early exit, the sales cycle for the company is about the same as the time you need to develop the product, so you need to start selling the company before you found it!

Four steps to the epiphany

There's a book on how to bring a product to market that is almost a samizdat document in the marketing world. It's a privately published book originally intended to accompany a course at Berkeley and Stanford. It's not the most readable of prose so don't expect the Innovator's Dilemma or Crossing the Chasm. However it is packed with good stuff for any startup, and especially for EDA startups who embody all the problems that

the book addresses. It's called *Four Steps to the Epiphany* by Steve Blank.

The heart of the idea of the book is that you don't know what the customer wants. So in addition to developing a product (preferably the minimum shippable product, since how do you know the customer even wants that?) you need to develop customers. You have a product development process. You need a customer development process. And hiring a VP of sales and a VP of business development and waiting around for engineering to ship doesn't count.

A secondary idea is that the customer development process is very different if you are creating a brand new market, entering an existing market or re-segmenting an existing market (producing a product that only serves part of the market, usually but not always either creaming off the high-end or disrupting the low-end).

It is hard to summarize an entire book in one blog post and I don't intend to try. You'll have to invest in the book yourself and I guarantee that you will find plenty of thoughtful ideas that are immediately applicable to almost any product launch, whether in a startup or a large company.

If you only take one idea away from the book it would be this: get out of the building. Startups don't fail for lack of technology, they fail for lack of customers. Heed Steve's words: "In a startup, no facts exist inside the building, only opinions." You have to go and talk to potential customers and even talking won't be enough. You'll have to ship them early product, burn them when it doesn't do what they needed, and correct your course. If you scale the company before you have the product right, you'll run out of money (and in the current climate you're not getting any more).

The idea of listening to customers is not to find out everything that they want and build a laundry list. It is to attempt to narrow the product down to the minimum shippable product, one that at least a few customers can get value from even if it doesn't do everything they want. Saint-Exupery's quote that "A designer knows he has achieved perfection not when there is nothing left

to add, but when there is nothing left to take away," summarizes the goal for the earliest stage of customer development. Then start iterating as your understanding increases.

If you read newspaper articles on startups, you often get the idea that a couple of guys in a garage really understood something deep and took down some huge corporation that was too dumb to notice. The reality is that almost all successful startups end up doing something different from what they first intended when they were founded, sometimes dramatically so. Look at Paypal (originally doing beamed payments from Palm Pilots) or more recently Twitter (part of a podcasting company). Or even Google (originally just doing search without a clue about how to monetize it). In EDA the changes are less dramatic but very few business plans survive after contact with the market.

Chips and Change

I've been reading an interesting book on the semiconductor industry. It's called *Chips and Change* by Clair Brown and Greg Linden. I got sent a review copy (there are some tiny advantages to being a blogger) and I'm not sure whether it is truly available. Amazon shows it as having a publishing date of 9/30 but also being in-stock with a delay. Anyway, if you have anything to do with semiconductors I recommend you buy a copy immediately.

The book looks at semiconductor as an economic issue rather than from a technological point of view (although this is not ignored) which fits in with my view of the world. Semiconductor process transitions are driven by economics (cheaper transistors) more than technology (better transistors) especially now where leakage and other considerations make it unclear whether you are getting better transistors or only more of them.

The book examines how the semiconductor industry has lurched between major crises that has driven both its success and its restructuring over time. It starts back in the 1980s when the US, having essentially invented the integrated circuit, started to lose the quality war to Japan. It examines 8 crises in total.

First, losing the memory quality war to Japan that eventually drove most US memory supplies (Intel, for example, remember they were a memory supplier) out of the market. Most readers probably don't remember when HP announced how much better the quality of Japanese memories was compared to American, and how it shook the industry to the core (they had lots of data).

The second crisis was the rising cost of fabrication. The result of this in the US (but not elsewhere) was the creation of fabless semiconductor companies that used TSMC, UMC and Chartered to manufacture. Also the creation of clubs of companies sharing the cost of process development.

The third crisis was the rising cost of design. This meant that low volume products just were not economically viable. Chips used to be consumed by big corporations largely insensitive to price to consumers who were hyper-sensitive to price. This was the fourth crisis. Somewhere in here the FPGA started to play a role.

The fifth crisis was the limits to Moore's law, in particular limitations in lithography (Moore's law is more about lithography than any other aspect of semiconductor manufacture). This has been an ongoing issue forever, of course, but has started to become the fundamental limitation on progress.

their cost, increased out of control there was a rush to find new talent in India and China. Partially for cost reasons but also because there were too few designers available without looking globally.

But fabs got more and more expensive, and price pressure on end-products got more intense leading to the current situation where most companies cannot afford to build a fab nor develop a leading edge process to run within it. There are just 4 or 5 groupings now that can do this (Intel, Samsung/IBM/ST, TSMC, Japan, UMC/TI) and there is likely to be further consolidation. Even with tapping into low cost Asian labor , semiconductors are not getting the share they feel they should of the electronics value chain.

The 8th challenge is the new level of global competition. Japan is clearly, for example, losing out as a "Galapagos market" with lots of internal competition but, as I've said before, turning their back on the world, just like how the Galapagos produced giant tortoises. But also there is governmental competition with states attempting to join the industry keeping global competition feverish.

The book has a great graphic that summarizes the change in the basis of competition over time. If you read from left to right you see the problems come up chronologically. The vertical scale splits them into technological problems, economic problems, and competitive/globalization issues. This single graphic pulls together all of the issues facing the semiconductor industry, and how it got here, in a single simple chart.

As I said earlier, if you are involved in the challenges of the semiconductor industry, this is a book you should read (and, in case anyone is suspicious, I'm have no relation with the publisher other than receiving a free copy).

The Flaw of Averages

I've been reading a very interesting book called "*The Flaw of Averages*" by Sam Savage. It looks at why using average data only produces the correct answers in very limited circumstances. The flaw of averages is that plans based on average assumptions are, on average, wrong.

For example, assume you are a manager deciding how big a factory (or fab) to build. Your marketing manager tells you he is

certain that you'll sell between 80,000 and 120,000 per year. But you insist on a number and get given the average of 100,000 and you build a factory with a capacity for 100,000. Let's assume that the marketing manager nailed the numbers precisely (don't we always?). On average how much money will you make? Well, the number will be somewhere between 80,000 and 120,000. If the number is less than 100,000 you make less money than you expected. If the demand is greater than 100,000 you don't make more money because your capacity is maxed out. So, on average, you make less money than you expected even though your factory has average capacity.

There are other fascinating things. You may have heard of Simpson's paradox. One of the most famous examples of this was a 1986 kidney stone study where treatment A was more effective than treatment B. But if you looked at only small kidney stones, then treatment B was better than treatment A. And if you looked at only large kidney stones, then again treatment B was better than treatment A. But when the two were combined, A was better than B. WTF?

Another example: in each of 1995, 1996 and 1997 David Justice had a higher baseball batting average than Derek Jeter. But taking the three years together, Derek Jeter had a higher average than Justice. WTF?

A lot of what you learned in school about statistics (means, variance, correlation etc) is really not very relevant now that we can run large numbers of investigations as to what is really going on in seconds. Means and standard deviations were an attempt to get at something important before this capability existed, what Sam Savage calls "steam era" statistics. Now we can use computation to make sure we don't fall into traps.

There's also lots of stuff about options and how to price them depends on thinking (or computing) this sort of thing properly. If a stock is $20 today and on average will be $20 in 12 months time, how much should you pay for an option to buy it for $21 in a year. If you'd succeeded in answering this a few decades ago you'd have won the Nobel prize. You may have heard about Black-Scholes option pricing, which does the math to work this out. Even though at the average stock price ($20) an option to

purchase at $21 is worth nothing (because you'd simply not exercise your option) it clearly is worth something since there is some chance that the stock will end up above $21 and you can make money exercising your option and selling it at the market price.

Some of these ideas are important in thinking about business plans and formalizes some of the sensitivity analysis that it is always good to do (how much more money do we need to raise if the first orders come 6 months later than expected? if the product costs 30% more to develop?).

Consider a drunk walking down the middle of a highway. His average position is in the center of the road on the yellow line. But on average where is he. Dead.

And don't forget, almost everyone has more than the average number of legs.

Chapter 7: Off-topic

What color is a green card?

Most American don't know the answer, which is today off-white. Mine is alongside. Presumably it was green once.

I'm an immigrant. I was born and brought up in the UK and came to the US in 1982 on an H-1 visa. After four years I got a green card (which was pink in those days) and became a permanent resident. I can live here indefinitely (provided I don't commit major felonies). Actually, the real timeline is that after 2 years I applied for a green card but it took another couple of years for the government to process the paperwork. There is no upside in good immigration processing since immigrants don't vote. Imagine the outcry if it took two years to get a passport.

It is one of the strengths of the US that people like me could easily come here and contribute, along with other more significant immigrants such as Sergey Brin, Jerry Yang, Albert Einstein, Bob Hope, John Muir, Carlos Santana and many others. That doesn't happen much in China or Mexico. The mayor of Vienna is not an American immigrant; Arnold Schwarzenegger came in the other direction.

On a personal note, I'm very grateful for the opportunity that the US gave me.

Most discussion of immigration centers on illegal immigration of poorly educated Mexicans, but all the evidence seems to be that while poor Americans may lose slightly through increased competition for low-paid jobs, they gain even more from things like lower cost food. But as a strategic issue for the US I don't think this is all that big a deal. The US economy doesn't stand or fall on the basis of how many Mexicans work here.

Much more important is the idiotic legal immigration policy we have for educated people. The most insane part is allowing students to come here for PhDs (55% of engineering PhDs are foreign-born) and expelling them when they are done, since there is no automatic route to stay here. Plus we make it harder than necessary to come here to study in the first place. First loss, these are just the kind of people that we need here in the US to drive technology businesses. Second loss, even if students go back to their home countries, they go back with a positive image of the US to counter the negative views of people who know little about the country.

The H-1 visa quota for each year opens up on 1st of April and closes immediately since twice as many applications are received that day as are available for the entire year. But those are for visas starting October 1st. When I came to the US either there was no quota or it was much higher than the number of applicants. If a company wanted to hire a qualified candidate from overseas (me) then it applied for a visa, waited about 6 weeks and got it, then the person could start. Today it is impossible to hire someone on that basis since the delay is about 9 months on average until the next October 1st after the next April 1st, and then there is only a 50:50 chance of getting a visa anyway. Companies can't be bothered with such a lengthy uncertain process.

The result is that H-1 visas have become a way for overseas consulting companies, especially Indian, to apply for large numbers of visas knowing some will get through and their employees can then come here months later. This is not necessarily bad but it also squeezes out everyone else, every talented person that an American company wants to hire from overseas, every student who wants to stay on once they have their doctorate and so on. The best solution if it is politically unacceptable to do the sensible thing and remove the cap, would be to 'auction off' the visas. But I don't mean by paying bids to the government but by using the salary that the employee would receive. The higher the salary paid the easier to get a visa for that employee. The Indian job shops would be 'outbid' by PhDs.

I can do no better than to quote James Fallows, an editor at Atlantic Monthly who currently lives in China (and used to live

in Japan during its heydey in the late 80s). Here he is talking about an Irishman who lived in southern California but had to move to China because he couldn't get a visa to remain here:

> "I might as well say this in every article I write from overseas: The easier America makes it for talented foreigners to work and study there, the richer, more powerful, and more respected America will be. America's ability to absorb the world's talent is the crucial advantage no other culture can match—as long as America doesn't forfeit this advantage with visa rules written mainly out of fear."

China and India

Let's look at China and India. They are both enormous, with over a billion people each. Both are making huge strides towards modernity having opened up to the outside world in the last twenty years or so after having very protectionist planned economies with the usual unimpressive results.

Both China and India have huge disparities in income and standards of living, with a strong growing middle class but a large population of subsistence farmers and people living on the margins in cities. And there are lots of cities, over 100 cities of over 1 million people in China so most of them you've never heard of. You are much more conscious of the poverty in India because everything is mixed up there. You can be in, say, the Cadence buildings in Noida and you could be in California except that some of the women are in jeans and some are in saris. But walk outside and there are cattle wandering around, people drying dung and beggars everywhere. In China, the eastern cities are prosperous and the marginalized mainly live in the rural west so if you visit Beijing and Shanghai you won't see them.

The other interesting major difference is the system of government. India is a bit like California with a thriving private sector and dysfunctional public administration. For example, every company has to have its own generators since the power companies are micromanaged by the politicians, can't force anyone to pay their bills, have controlled prices and so can't

make any money and thus can't invest. As the largest democracy in the world, those rural and illiterate poor are able to force bad policies in many areas.

By contrast, China has a non-democratic government. But as a result it is perhaps easier for it to pursue sensible policies that would not necessarily be popular. So they have managed, originally in the south far from prying eyes in Beijing (which is in the north for those of you with no Chinese geography: bei is the Chinese for north) to have business friendly policies that have ignited a boom that has lifted more people out of poverty faster than anything in history. Deng Xiaoping, who is credited with orchestrating the change in policy, is a hero in that sense, despite the unacceptable blot of Tiananmen Square. To read a current viewpoint on various aspects of China, I recommend James Fallows's book *Postcards From Tomorrow Square*.

Because of their populations and a strong cultural emphasis on education as a means to advancement, both China and India produce a lot of well-trained people in any area such as engineering and computer science. India, of course, has the legendary Indian Institutes of Technology (IITs). Almost every successful Indian you meet in Silicon Valley is from one of the IIT (even Asok, the intern in Dilbert). China also graduates a huge number of engineers, There is some debate about the actual number and whether many of what Asia calls an engineer are really what we would call technicians. But it is pretty clear that both countries are graduating more real engineers than the US. This is not necessarily bad for the US. It is the dirty secret of Silicon Valley that so much of the engineering is done by people born outside of the US. When I ran a 200 person engineering group in the 1990s I estimated that over half were immigrants, starting with me. So it is all the more important that, as I said last week, the US have sensible immigration policies to make it easy for such people to come here (or stay here when they finish their advanced degrees).

Of course the current downturn will have a major impact on both countries. China, in particular, has to keep growing fast enough to make the country rich before it becomes old due to its inverted demographic that will be created by the one-child policy. It will

be interesting to compare China and India and whether Lee Kuan Yew's view that you need to liberate economically before liberating politically is proven right or wrong. China (and Singapore, of course) are exhibit A; India is exhibit B.

Visa. Priceless

The current downturn has lead to renewed focus in the H-1B visa cap, not to mention xenophobic restrictions slipped into the TARP bills to make the US even less welcoming. I think we have the worst of all worlds right now. The caps are so low that companies cannot use H-1 visas to hire talented people from overseas to work for them, they have become only a way for Asian subcontractors to get people in the to country and nothing much else. The entire year's supply of visas goes in a day so the old model no longer works. It is no longer possible to find a talented person overseas, hire him or her, get a visa and set the start date a few weeks later. That is how I came to the US in the early 1980s. Now, the only model that works for a person like that is to hire them onto your overseas subsidiary (so don't be a startup or you won't have one) and after they have worked for a year it is possible to transfer them on an L-1 visa.

But people always tend to focus on the lowest level people and debate whether or not a person with an H-1 visa is taking a job away from an equally qualified American. In the old days the answer was certainly "no", but now I'm not so sure. They are for sure taking a job away from an almost certainly more talented overseas employee who cannot get hired under the current visa system and who would be an unquestionable gain to the US as an immigrant.

However, immigrants create a lot of jobs for Americans too by their skill at founding or managing companies. In EDA, for example, Aart de Geus (CEO of Synopys) came from Switzerland, Lip-Bu Tan (CEO of Cadence) came from Singapore, Rajeev Madhavan (CEO of Magma) came from India. As far as I know, Wally Rhines (CEO of Mentor) is American born and bred. Some other sizeable EDA companies with immigrant CEOs are Attrenta (Ajoy Bose from India), Apache

(Andrew Yang from China), Sequence (Vik Kulkarni from India), VaST (Alain Labatt from France), Virtutech (John Lambert from England).

xxx

I'm guessing that most of the immigrants originally came to this country either as students (so on an F-1 visa) or on an H-1 visa. Today we make it much too hard for the next generation of talented individuals overseas to come here and stay.

I think that over the next few years the problem with the US just as likely to be immigrants leaving the country, especially to return to India or Taiwan/China. This is already happening to some extent. Growth there is more attractive than here, and the infrastructure in the US for starting a business, thought better, is no longer so superior to everywhere else.

I think that the US's capability to absorb talented individuals and make them successful is a competitive advantage no other country has. Everyone else must love the way we are handicapping ourselves these days. We are our own April fool joke, but not even mildly humorous.

Downturn

Superficially, the present downturn is similar to the "technology" crash of 2001. I put technology in quotes since very little of that first internet boom involved true technology, and many people who called themselves programmers were writing plain HTML. As somebody, I forget who, said to me at the time: "surely one day technology will count again." Of course some companies, like Amazon, Webvan, eBay or Napster, had a differentiated technology foundation to go with what was mainly a business model play but most did not.

But undeniably the boom of investment created a huge number of jobs. When the crash finally came, large numbers of them were destroyed. A lot of those people had come to the bay area attracted by the boom, and when their jobs went away they went home again. The SoMa warehouses in San Francisco emptied out

as fast as they had filled and apartment rents came back down. Many people who had left the EDA industry to make their fortune returned to a place where their knowledge was still valued. As is often the case, the people in EDA (at least the ones I know) who made it big in the internet companies were people who left early, before it was obvious that it was a good idea. People who joined Yahoo before it was public, who formed eBay's finance organization or founded small companies that built important pieces of the plumbing.

This downturn seems different. Many of the people being laid off (and I don't just mean in EDA, in silicon valley in general) are people who have been here for decades, not people who came here in the last few years as part of the gold rush. Of course, veterans have been laid off before and then immediately re-hired when the eventual upturn came.

But again this downturn seems different. I don't think that many of these jobs are coming back again. Ever. EDA in particular is undergoing some sort of restructuring, as is semiconductor. We can argue about precisely what we will see when the dust settles, but I don't think many of us expect to see the 2007 landscape once again.

I've pointed out before that it is obvious that EDA technology is required since you can't design chips any other way. But the EDA industry as it was configured will not be the way that tools continue to be delivered. It is hard to imagine that Cadence will employ 5000 people again any time soon, to pick the most obvious example.

The many dozens of EDA startups that used to employ significant numbers of people in aggregate aren't coming back either. Any startups that do get formed will be extremely lean with just a handful of people. Partially this is driven by technology: with modern tools and open infrastructure, it doesn't take an EDA startup half a dozen people and a year or two to build (duplicate) the infrastructure they need on which to create differentiated technology. It takes a couple of guys a couple of months. Partially size is driven by investment. With public markets closed to EDA companies (to everyone right now but to small software companies probably forever) then the only investments in EDA

that makes sense are ones that still make sense with $25M as a target acquisition price, not $250M.

A recent report by Accenture (it's called "How Semiconductor Companies Can Achieve High Performance by Simplifying Their Businesses" but you have to pay lots of $ to read it) reveals that some semiconductor engineers are "disenchanted" in their work, and "fearful of losing their jobs." That's the kind of revelation that really makes you want to reach for your wallet.

Facetiousness aside, the report also points out that as semiconductor companies go fabless (or at least fab-lite in the meantime) then the dynamics of what is valuable change. Most obviously if you are in technology development (i.e. semiconductor process development), which is no longer required. And as I've pointed out, once you don't have a fab, there is often not a lot of justification for the particular combination of businesses that find themselves in the same semiconductor company. The weak nuclear force has gone to zero and all those nuclei are going to fly apart.

Silicon Valley is a unique ecosystem, the center of the unverse for technology. But it is changing in form in ways that are not yet clear.

Old standards

About 12 years ago I went on a three-day seminar about the wireless industry presented by the wonderfully named Herschel Shosteck (who unfortunately died of cancer last year although the company that bears his name still runs similar workshops). It was held at an Oxford college and since there were no phones in the rooms, they didn't have a way to give us wake-up calls. So we were all given alarm clocks. But not a modern electronic digital one. We were each given an old wind-up brass alarm clock. But there was a message behind this that Herschel had long espoused: old standards live a lot longer than you think and you can't ignore them and hope that they will go away.

In the case of the wireless industry, he meant that despite the then-ongoing transition to GSM (and in the US to CDMA and

US-TDMA) the old analog standards (AMPS in the US, a whole hodge-podge of different ones in Europe) would be around for a long time. People with old phones would expect them to continue to work and old base stations would continue to be a cheap way of providing service in some areas. All in all it would take a lot longer than most people were predicting before handset makers could drop support for the old standard and before base stations would not need to keep at least a few channels reserved for the old standard. Also, in particular, before business models could fold in the cost saving from dropping the old standard.

My favorite old standard is the automobile “cigarette lighter” outlet. According to Wikipedia it is actually a cigar lighter receptacle (hence the size, much larger than a cigarette). The current design first started appearing in vehicles in 1956. Originally, they were simply intended to be a safer way for drivers to light their cigars than using matches. After all “everyone” smoked back then. Since cars had no other power outlet, anything that needed power used that socket as a way of getting it without requiring any special wiring. Who knew that in an age where few of us smoke, and where we can’t smoke on planes, that we’d be plugging our computers into outlets on (some) planes that are designed to match that old design. If you’d told some engineer at GM in the 1950s that the cigarette lighter socket would be used by people like him to power computers on planes, he’d have thought you insane. Computers were million dollar room-sized things that only a handful of big companies used, and planes were too expensive for ordinary people. Talking of planes, why do we always get on from the left-hand side? Because it is the "port" side that ships would put against the port for loading, unobstructed by the steering-oar that was on the right-hand side before the invention of the rudder, hence steer-board or "starboard". The first commercial planes were sea-planes, so they naturally followed along. Another old standard lives on, a thousand years after steering-oars became obsolete.

We see some of the same things in EDA. OK, the 1970s weren't a thousand years ago, but in dog years it seems like it. For physical layout, it is still the case that a lot of designs are moved around in what is basically the Calma system tape backup format, a standard that dates back to the mid 1970s. Verilog is not going

away any time soon to be replaced with something more "modern." Sometimes new standards come along but it is rare for the old ones to die completely. We can probably drop Tegas netlist support, I suppose, but I'm sure somebody somewhere has a legacy design where that is the only representation available.

So new standards come along all the time, but the old standards simply don't die. At least not for a lot longer than you would expect. Rrrrrnnnngggggg.

San Francisco: silicon valley's dormitory

San Francisco is a dormitory town for Silicon Valley. Not completely, of course. But unless you regularly drive between Mountain View and San Francisco you probably aren't aware of the huge fleet of buses that now drives people from San Francisco to other cities: Google in Mountain View, Yahoo all over, Genetech in South San Francisco, Ebay in San Jose. I have a friend who knows Gavin Newsom, the mayor, and keeps trying to get him to come and stand on a bridge over the freeway one morning to see just what is happening where lots of people (me included) largely work in Silicon Valley but live in the city. The traffic is still more jammed entering the city than leaving but it's getting close. Bauer, who used to just run limos I think, now has a huge fleet of buses with on-board WiFi that they contract out to bring employees down to the valley from San Francisco. They cram the car-pool lane between all those Priuses making the not-so-green 40 mile trip.

San Francisco seems to have a very anti-business culture. Anything efficient and profitable is bad. So if, like me, you live in San Francisco you have to drive for 15 minutes and give your tax dollars to Daly City if you want to go to Home Depot. They finally gave up trying to open a store in San Francisco after 9 years of trying. Of course a Walmart, Ikea or Target is unthinkable. And even Starbucks has problems opening new stores since they (big) compete too effectively against local coffee shops (small, thus good by definition). The reality is that some small coffee shops (like Ritual Roasters) are among the best

in the US, and a Starbucks next door wouldn't do well; and for some a Starbucks in the area would be an improvement. But in any case it makes more sense to let the customers of coffee shops decide who is good rather than the board of supervisors trying to burnish their progressive credentials.

Those two things together—much commerce is out of the city, many inhabitants work outside the city—are warnings that San Francisco is not heeding. San Francisco already has one big problem (as do many cities) that housing is really expensive (at least partially due to economically illiterate policies like rent control and excessive political interference in the planning process making it difficult to build any new housing) and the public schools are crappy. So when a resident has a family, they have to be rich to afford a large enough house and a private school, or they move out. So every year San Francisco can close some schools since there are ever fewer children in the city; famously there are more dogs than kids.

The trend, which is not good, is for San Francisco to depend increasingly on three things: out of town rich people who live elsewhere (often in Nevada due to California's taxes) but like to keep an apartment in San Francisco (about 20% of the people in the building where I live are like that); people who live in San Francisco and work somewhere else; and tourism. Two of those three groups are spending a lot of money and generating a lot of tax that San Francisco doesn't get to see, but it does have a lot of the costs associated with them. Of course, tourism brings dollars in from outside but most of the employment it creates is not at the high valued added end of the scale: restaurants, hotels and retail largely generate low-productivity low-pay jobs.

Busboys for San Francisco; on-chip buses in Silicon Valley; wi-fi equipped buses in between.

Patent trolls

CDMA is also another interesting oddity from a patent point of view. Most patents are tiny pieces of incremental innovation that form the many little pieces you need to build complex technological products. You can't build a semiconductor without

violating thousands if not millions of patents. For example, Motorola (Freescale now, I suppose) owned a patent on the idea of filtering photoresist which surprisingly passed the non-obvious test. This used to be a minor annoyance since the patents were owned by other semiconductor companies, and the problem could be resolved with a manageable payment or royalty and a cross-license. After all, you don't need to be in the business for long before they can't build anything without *your* patents. Now that a huge number of patents are owned by so-called patent trolls, people who have purchased patents for the explicit purpose of trying to generate disproportionate licensing revenue, the cross-licensing approach won't always work and, as a result, the patent system is effectively broken for technologies like semiconductor (and EDA for that matter) that stand on the shoulders of those who went before in ways too numerous to even take time to examine.

Patents were a problem for GSM phone manufacturers since companies like Philips and Motorola managed to design their own patents into the standard. GSM had the concept of essential and non-essential patents. An essential patent was one that you couldn't avoid: if you were compliant with GSM you were violating the patent, something that "shouldn't happen." However, the essential patent owners preferred to keep their heads down for political reasons (don't want those European governments telling us off in public) and keep quiet about what patents they owned until businesses were rich enough to be worth suing. For example, Philips owned the patent on the specific vocoder (voice encoder) used in GSM. Not the general idea of a vocoder, or that type of vocoder, just the specific parameters used in GSM, for which they would like about $1/phone. It was as if Ford owned the patent on the order of the pedals in a car. Not the idea of an accelerator, clutch and brake but the specific configuration of the clutch to the left of the brake to the left of the accelerator. And then got some car standardization authority to mandate that order for all vehicles. Come to think of it, that's pretty much what GM did when they got the US government to mandate catalytic converters for all cars, which required all car manufacturers to license catalytic converter patents from a certain car manufacturer beginning with G. And there was some of this with Qualcomm too, since "everybody" knew that the main US

carriers would choose GSM, the almost world-wide standard, until someone from the President's office apparently told them at the last minute that it really ought to be a US standard.

People are smarter these days about making sure that patents don't get designed into standards. Look at the fuss over Rambus. However, it is still a grey area. After all, nobody knows what even their own company's patent portfolio really covers. If you've read a patent, you know how hard it is to tell what it really says. You can only read the word "plurality" a limited number of times before your eyes glaze over. And at the company level, nobody knows the whole portfolio. If you are the representative from, say, Nokia on some standardization committee, then you can't really guarantee that any particular standard doesn't violate any Nokia patents, and you are certainly not going to sign up for guaranteeing never to sue, say, Samsung over a patent violation. Especially as you are not the corporate counsel, you are some middle level engineer assigned to a standardization committee that may or may not turn out to be strategically important.

But CDMA was a complete patent-protected technology more like a blockbuster drug formula. You couldn't do anything in CDMA without licensing a portfolio of patents from Qualcomm on whatever terms they felt like giving you. They invented the entire technology and patented it before anyone else really knew it was feasible. They sued Broadcom, they sued Ericsson, they sued everyone and pretty much established that there was no way around this no matter what. In 2G this wasn't a big issue since GSM doesn't depend in any way on CDMA. But W-CDMA and all the later technologies use various aspects of CDMA and so Qualcomm is in the happy position of having a tax on every cell phone.

Patents

The basic "tradeoff" in having a patent system is that without the promise of some sort of state-sanctioned monopoly innovation would be a something that would be underprovided. Let's not argue about that dubious point, and just take it as a given.

Another positive for the system is that requiring the inventor receiving the monopoly to disclose the details of the invention, means that once the monopoly period ends then the details are freely available for everyone to copy.

Let's see how that seems to work in practice in the two industries I know well, EDA and semiconductors.

I knew nothing about patents until the mid-1980s. I was at VLSI Technology and we didn't bother patenting stuff since we were small and patenting was expensive. Once VLSI reached about $100M in revenue, other semiconductor companies with large patent portfolios (IBM, Motorola, TI, AT&T, Philips, Intel and so on) came knocking on our door with a suitcase of patents, saying we probably infringed some of them and would we please pay several million dollars in licensing fees. We probably were infringing some of them, who was even going to bother to try and find out, so that first year the only negotiation was how much we would pay. VLSI started a crash program to patent everything we could, especially in EDA where we were ahead of the work going on inside other semiconductor companies. When the patent licenses came up for renewal we were in a much stronger position. They were infringing our patents and how much were they going to pay us. Well, how about we license your patents and you license ours and no money (or at least a lot less) needs to change hands? No lawyers on either side had any intention of actually reading the patents, or disturbing their own engineers to find out if they were infringed. It was patent licensing by the ton.

To me, in these industries patents seem to be entirely defensive created purely on the basis that other people have patents and therefore might seek license revenue. If there were no patent system, both EDA and semiconductor would proceed exactly as they do today. There may be the occasional patent that is so valuable that it is created to attempt to get monopoly licensing out of the rest of the industry (Rambus, Blueray) but these seem to be mainly political issues around trying to get proprietary technology into standards. Most patents are incremental improvements on existing technology that are created only for defensive reasons, with no expectation of ever truly licensing anyone or even going looking for infringement. Every company

needs a portfolio of patents so that when other players in the industry come seeking license royalties, the "victim" has a rich portfolio that the licensor is probably violating and so the resolution is some sort of cross-license pact. There is some genuine licensing of patents in semiconductor, but none that I know of in EDA.

As to patents being a way of disseminating information, there are two problems. The first is that in semiconductor and EDA, waiting 20 years for a patent to expire and then implementing the protected invention using the patent as a guideline is laughable. The timescales are just too long to matter in this industry, and secondly, have you read a patent? There is no way you can really discern what it even covers, let alone use it as a blueprint for implementation. For example, Kernighan and Lin's patent from 1969 on their well-known partitioning algorithm. My guess is that every placement tool in every EDA suite violated this patent, but was written without ever looking at the patent. It's standard graduate level graph optimization and has probably been independently invented several times.

Patent law provides for damages in the event of patent infringement. But willful patent infringement, when you know that the patent exists, carries punitive triple damages. So the advice I've always been given by lawyers is to tell my engineers never to read any patents. That way, even if a patent is infringed it is not being *willfully* infringed since there is no way for whoever wrote the code, or whatever, to know that it was violating that particular patent.

So the situation comes down to this: companies patent inventions in order to have currency to negotiate with other companies with patent portfolios and not to disclose important techniques to the general public, and not because without the protection of a patent, innovation in semiconductor and EDA would grind to a halt. It is like mutual assured destruction in with nuclear weapons. The purpose of all that effort and investment in nuclear weapons was purely to ensure that they other guy's weapons weren't a threat.

Companies that purchase a few patents simply to demand licensing fees, so-called patent trolls, violate this game. They are like a terrorist with a nuclear bomb. No matter how many

missiles we have to “cross-license” the terrorist isn’t interested. At least when it was just companies threatening each other and then cross-licensing the game wasn’t played with real money. The shakedown of RIM (Blackberry) a year or so ago was a complete indictment of the ridiculous situation we have got reached.

So in EDA and semiconductor, patents are largely a joke. If they didn’t exist, people would not be clamoring for them. There was plenty of innovation in software in the 1960s when software was not even patentable. Nobody cares about patents except for defense, so for our industry patents are a cost not a benefit, a distraction for engineers who could better be spending their time engineering. In fact, I'd go further. If patents were actually enforced, in the sense of requiring a license to be negotiated to every patent actually violated, then innovation would grind to a halt.

Where does everybody come from?

Where does all the brainpower that drives Silicon Valley come from? The answer, by and large, is not from round here.

A good analogy I saw recently was with Hollywood. Where do all those pretty young actresses come from? By and large, not from Los Angeles. If you are pretty enough with some acting talent living in a small town in the mid-West, Hollywood is potentially your route to advancement. The odds aren’t that great, of course, since pretty women aren’t a vanishingly small percentage, and Hollywood doesn’t want all its actresses to look like supermodels anyway.

Silicon Valley draws in intellectual firepower in the same way. In fact in an even bigger way since we don’t care what race you are and whether your English is perfect. We draw in many of the smartest people from all over the world, in many cases have them do Masters degrees or PhDs here, and then employ them, to the extent that our grandstanding politicians will allow (which is not the topic for today).

I remember studying a 240 person engineering group I was responsible for and I estimated that over half of them were born outside of the US: a lot of Indians, of course, Vietnamese and Chinese. But also French, English, South American, Egyptian. Pretty much everywhere. Of the people who were brought up in the US, a big percentage seemed to be from the mid-West just as in the Hollywood example above. That was a surprise.

This isn't meant to be a criticism of California's K-12 education system, although there is certainly plenty of criticism to go round, especially for the bureaucrats and the venal teacher's unions. But if you are brought up around here (or in New York, Boston and so on), you have lots of options and working really hard in high-school so that you can go to college and work on a really hard engineering degree might not be that attractive. But if you are in a small town in the middle of an agricultural state, or a large town in India, with little in common with your peers due to your geekiness, then this seems like the a good way to escape. It's probably not that deliberate a plan, teenagers are notorious for acting in the moment, but, as Sam Lewis and Joe Young put it: "How you going to keep them down on the farm after they've seen Paree?" Well, Mountain View isn't quite Paris but it's not Nowhereville either, and the weather is a lot nicer.

Politicians all over the world look at Silicon Valley and say "we want one too." But Silicon Valley is really self-sustaining, sucking in intellectual talent from wherever it is found. Those politicians want a film industry too.

But Silicon Valley and Hollywood both got started in another era through a serious of chance events like Shockley preferring California to New Jersey, and the early film industry wanting to get as far away from Edison and his patent lawyers . Getting a Silicon xxx or a film industry going in your state requires more than just a few adjustments to the state tax code. The best talent, even from your state, is going to California (so long as California's appaling political leaders don't sell the entire state to the public sector unions).

Silicon Valley, the Hollywood of the North.

Public affluence private squalor

California. Unions actually behave just the way that you would expect them to, to maximize their own power. The California democratic politicians, meanwhile, have gerrymandered themselves a permanent incumbent majority and they are supported by those same unions. So they go along with expansion of spending year after year, and expansion of salaries and, especially, benefits. Everyone gains except the private sector taxpayers, who get screwed.

I think that the political-union complex is completely out of control in the public sector, and is a major threat to Silicon Valley's continued long-term success. We are on a path to private squalor and public affluence in California. The private sector will have to fund all the promises that the politicians made over the years, in a massive transfer of wealth from the poor (retirees living on their savings and people making average salaries) to the rich (public sector retirees).

The cities of Vallejo went bankrupt recently, entirely due to firefighter and police salaries and benefits, especially retirees where they have lots of retired employees on six-figure salaries and unlimited medical benefits for life. Vallejo has 120,000 residents but $850M of unfunded retiree commitments to the police and firefighters. That's around $25,000 per household. Those people probably also owe at least that much in unfunded public sector retiree benefits at the state level too. Many other cities are predicted to go bankrupt in the current downturn, since it's the only way they have a chance to re-negotiate those gold-plated contracts.

A friend of friend works in finance for the Santa Clara school district. Every teacher they employ costs $180,000 per year. About a third of that is salary and medical benefits but the rest is their retirement benefits, which Santa Clara is smart enough not to leave unfunded to create a future disaster. I wish someone was putting away over $100,000 per year for my retirement.

There's not really any good way to measure prison guard productivity, but in education there is. Over the last 20 years,

adjusted for inflation, California's spending on education has doubled. But standards are exactly where they were 20 years ago according to the state's tests. In what other industry has productivity halved? The best part of California's education system is actually the universities and community colleges. But that part is now threatened because all the money is going to the inefficient K-12 segment where, in principle, we could cut spending by 50% with no effect on outcomes. Stanford is a private university, but Berkeley, UCLA and so on are not. If they lose their stars then it will certainly affect Silicon Valley.

Since Arnold Schwarzenegger was elected governor in 2003, California's spending has increased by 40% because so much of the budget is on autopilot, driven by various propositions or by existing contracts. What that means is that we could reduce California's budget by a bout a third and it would be something like 2001 again. It didn't seem bad back then after all.