



Economics of Software-rich Chips

October 2002

This white paper shows that the trend towards increasing numbers of systems being implemented in the form of embedded processors running software is a consequence of the economic drive of Moore's Law and is thus driven irresistibly by the mass-production economies of scale of the semiconductor industry. The paper is intended for senior managers in the system, embedded software and semiconductor industries. It examines the economic trends and looks at some approaches to how to take advantage of them.

"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."
Gordon Moore

Electronic Systems

Electronic systems are all around us in our homes, our cars, and our offices. The cost of the semiconductors that are the heart of these systems have dropped dramatically. A system such as a video-game console was unimaginable a decade ago, easily outperforming the best then-available systems in the form of multi-million dollar aircraft flight-simulators. Video-game consoles are now so cheap that not only can we afford them, our children can buy them out of their allowances.

The primary driver for this is the continual march of semiconductor process technology, which allows larger and larger systems to be implemented on a single silicon chip, and at the same time reduces the cost of the next generation of any existing silicon-based systems. This was first noticed by Gordon Moore over 30 years ago and is now known as Moore's law.

At the high-end, this means that it becomes possible to implement previously unfeasibly complex systems. For smaller chips, where feasibility is not the issue, a given piece of electronics can be implemented at a price that is always declining until it reaches the level at which it can enable new markets or dramatically expand existing ones.



Performance Increases or Cost Decreases

As semiconductor technology delivers increasingly higher performance and lower cost, it can be used for two rather different purposes.

Firstly, it can be used to develop higher performance chips and thus deliver systems with a higher performance than was previously possible. For example, dedicated microprocessors for PCs, high-end graphics-chips or high-end router chips all allow higher-performance PCs, video-games and routers respectively. The technology simply allows systems to be constructed with more performance than was possible before.

Most systems are only partially performance limited in this way since they are based on some sort of standard or performance point (or both). For example, a cell-phone takes advantage of semiconductor technology by continuously driving down physical size, increasing battery life and decreasing cost. Only occasionally when the standard is updated (such as future 3G cell-phones) is it possible to build a genuinely higher-performance system. As another example, to compare to the high-end router discussed above, consider a mid-range router with a given packet-processing rate. The advance in semiconductor technology allows this to be delivered at a lower cost. Of course, it also allows an alternative way to build higher performance router, as opposed to cost-reducing the previous higher-performance chip. This will be attractive if the implementation is much simpler to deal with.

So over time, in some segments, higher-performance silicon technology allows new higher performance chips to be built (if there is a market for that in the segment), and allows other performance points to be cost-reduced and to use simpler implementations.

Software-rich Chips

One particular implication of the inexorable march of process technology has been the availability of processors as part of a chip. When semiconductor-based processors were first created they occupied an entire chip, so any system incorporating a processor was inevitably a multi-chip printed-circuit board design. As processors have become smaller and hence cheaper, it has become cost-effective to implement functionality of all or parts of a system using processors and an associated program, known as embedded software.

Most silicon structures are not general-purpose and only a few markets can capture the value of the increase in silicon performance to deliver a higher performance, higher priced system. For other structures, the increase in performance shows up as a reduced duty cycle since the specialized structure cannot be used for other purposes. A processor is different, being general-purpose by design. As processors get cheaper and higher performance they can subsume more and more of the specialized functionality of the chip onto a single (or few) processors. Thus inexorably there is a wave washing the specialized semiconductor structures into the processors and hence moving from a hardware implementation to a software implementation.

Increasingly with the availability of powerful specialized signal and media processors, whole systems are constructed where almost the entire functionality is implemented in embedded software of one type or another. For example, a current (second-generation) digital cellular phone typically contains a general purpose microprocessor to handle the user interface and protocol details, and a digital signal processor to handle the complex aspects of the radio interface and also encoding and decoding the voice. Apart from memory for the processors and buses to interconnect them, there is little else on the chip.

System Design

System design is the design in-the-large of electronic systems, a step that is then followed by more detailed implementation of the various components. The most advanced groups create an abstract executable specification and then through successive refinement steps reduce it to implementation, eventually to chip layout and software source code.

Based on the discussion above, it is not surprising that one key decision in system design is which functionality should be implemented in hardware (specially designed semiconductor structures) and which in embedded software. In general to implement a function in software will be slower than a dedicated implementation, consume more power, but consume no (or negligible) chip-area since the processor is already on the chip. However, it is not straightforward to determine if the implementation consumes too much power, or is too slow, since this depends on very detailed modeling of the processors under a realistic body of software.

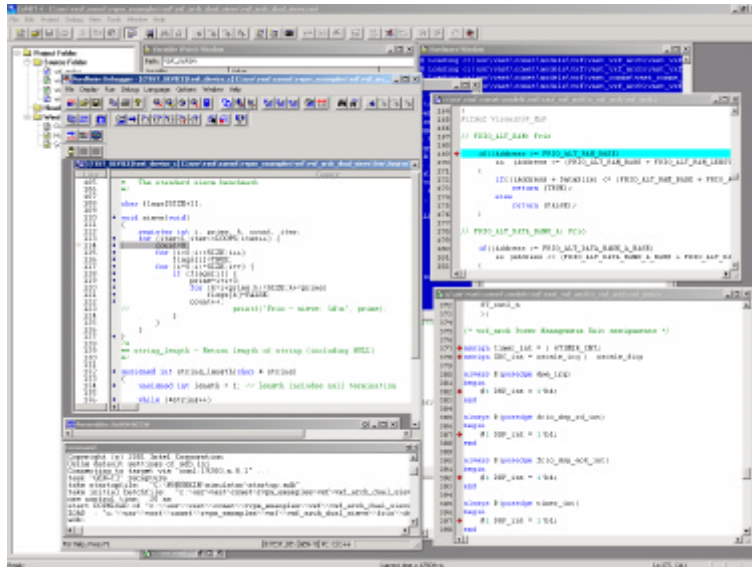
There are two separate areas of interest in system design. One is what the system should do, its behavior. And the other is how the system will be implemented, its architecture. To some extent these are independent concerns since there are many possible ways to implement a given piece of functionality.

In principle given some behavior and a target architecture it is possible to evaluate the match and decide how the architecture will perform. In practice, this has been difficult to do since it has not been possible to model the processors running the software fast enough with good accuracy to give useful quantitative results. Previous software simulation techniques have either given up too much speed or too much accuracy to be useful, and this early in the design cycle it is not practical to build hardware mockups for each architecture being considered. The process of evaluating an architecture has become so lengthy that in practice it is only possible to guess at the goodness of an architecture, and then build the system with crossed fingers.

The alternative to accuracy is always pessimism, so it is always possible to have a good enough architecture by using faster processors and wider buses than necessary. But this may end up with a system that dissipates too much power, and certainly renders the solution vulnerable to price competition from chip designs that more closely match the required performance to the silicon.

VaST modeling technology

VaST Systems Technology breaks this bottleneck. VaST can model processors and the associated infrastructure of buses, memories, interrupt-controllers at close to actual silicon speeds, orders of magnitude faster than any other software technology available. This allows architectural exploration to take place and optimize the tradeoff between cost, power and performance.



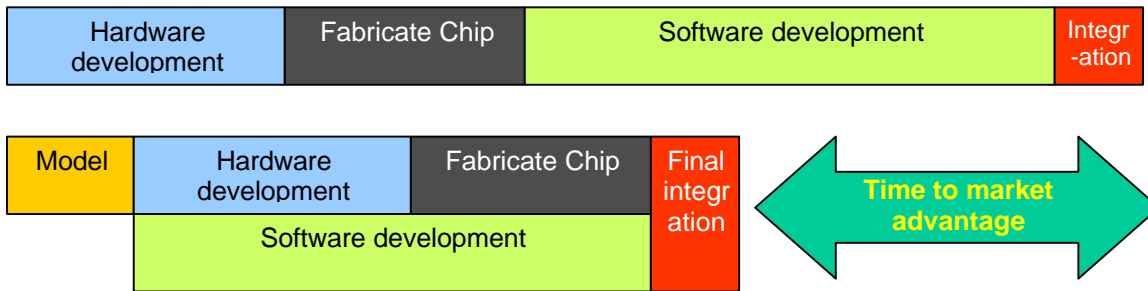
VaST's processor and bus modeling technology allows processors to be simulated at speeds of 25-120 MIPs. This is significantly above the 10 MIP threshold that is considered the minimum to be useful for architects and software developers. Even in a multi-processor system where the host PC has to divide its cycles among two or more processors, acceptable speeds are achieved.

Further, VaST models are cycle-accurate, modeling pipeline stalls/interlocks, branch prediction, cache hits/misses and memory management. The quantitative results obtained from using a VaST virtual platform thus directly translate into real performance numbers for the system. It is thus possible to use the virtual platform to examine time critical and safety critical responsiveness, such as whether an airbag will be deployed sufficiently fast by an automotive electronic control unit.

No volume shipments until software is complete

No matter what development approach is used, development of embedded software is at the tail of the process, and obviously the system product cannot ship until the embedded software development is complete. This is a very important point: nobody in the value chain makes any money until the software development is complete and the product ships in volume. This remains true even for a chip company shipping to a customer system company responsible for the software. No chips will ship in volume until the customer completes software development.

Much effort and investment has been focused in the past on chip development. But in a world in which an increasing percentage of the system is implemented in embedded software, myopic focus on the chip in isolation is not warranted. It has meant that almost the entire tools budget has usually gone to chip development, and software development



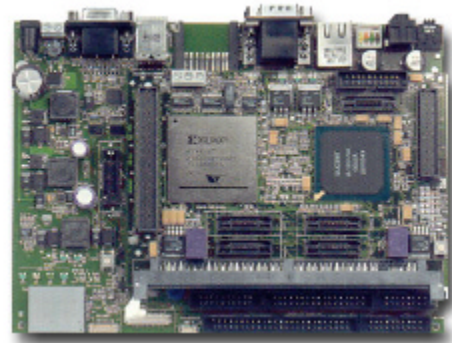
has been dismissed as unimportant and straightforward. To be fair, until now there has been a shortage of tools that would accelerate the software schedule. Another symptom is that hardware designers have been reluctant to provide a behavioral level model of the chip, which would be the biggest accelerator of the software development schedule, since it is work that does not accelerate the chip schedule. Chip developers are measured only on the chip development, and their management finds it counterintuitive that a longer chip development schedule might lead to a shorter time to money. This is especially so when the software development is being done by a customer of the chip company.

This attitude will change in the future as it is realized that the software schedule is more important than the hardware schedule, since it contains more of the effort and is always the critical path. Anything that pulls in the software schedule, from investment in better tools, to adding functionality to the chip to increase the software engineers view into the chip, will become more and more attractive as chips contain increasing amounts of embedded software.

VaST for Software Development

In addition to architectural exploration discussed above, VaST’s modeling technology allows software to be developed on a software model of the platform. The model is cycle accurate, so that it delivers the correct answers even for complex multi-processor architectures. And the processors are modeled at 25-120MIPS so the normal edit-compile-debug cycle that is the bread and butter of a programmer’s day, can be done directly on the model.

The current de facto way of doing this is to use a single board computer. There are several disadvantages to this approach, but far and away the biggest is that the FPGAs on the single board computer (or the board if it is a custom design) cannot be programmed until hardware design is complete, so it forces a serial development where software and hardware development cannot be overlapped much. Using the VaST approach to architectural exploration and software development, a provisional architecture can be



encapsulated by the architects and given to the software developers so that they can get started. This allows architectural exploration, hardware development and software development to proceed in parallel, pulling in the final schedule significantly. Remember, nobody makes any money until the software development is complete, so the only milestone that really matters is the completion of software development.

For more complex systems with multiple processors there is another serious problem with the hardware-based approach: when one processor hits a breakpoint the other processors do not stop. In a software-based solution the entire system freezes when one processor hits a breakpoint and it is possible for the engineer to investigate at leisure any aspect of the suspended system.

The Upcoming Power-struggles

There are two major battles that will play out over the coming years, driven by the trend towards software-rich chips.

The first battle is between hardware and software groups. In the past, the hardware groups designing chips have been larger than the software groups. They have had most of the budget and most of the management attention. Indeed, getting a system-on-chip to market was almost synonymous with designing the hardware of a chip. As a result, most decisions about design tools, modeling and architecture were taken by the hardware group looking largely at their own issues. Software groups were expected to develop the embedded software, but were not expected to require powerful tools nor to require any support from the hardware designers.

However, as software becomes an increasingly large percentage of the design of a system, the software groups are going to become larger and more influential. Designing a system-on-chip will shift towards designing the complex embedded software and, in parallel, building a silicon platform to run it efficiently. As a result, decisions about methodology will start to be driven towards approaches that address the issues of software developers since they will be the main drivers of cost and time-to-market. For example, it will be more common to add hardware to the chip to facilitate software debugging. And in the area covered by this paper, the chip modeling will need to be done in a way that makes embedded software development easier, and not just in a way that makes development of the chip easier. The biggest difference here is performance. During hardware design, at least part of the chip is being modeled at the hardware RTL level. The performance of the rest of the simulation is then less important since the RTL simulation will consume almost all the simulation cycles. But for embedded software it is important that the performance of the modeling is high enough to allow for productive edit-compile-debug cycles, while at the same time preserving accuracy so that the performance of the system can be measured. This need not have any negative consequences for the hardware designers, but does mean that the low-performance models that have typically satisfied hardware designers will not be adequate.

The other battleground is for supremacy in the supply chain. In the supply-chain discussed earlier, the system companies write most of the software. But increasingly the semiconductor companies are realizing that they have to write at least some of the software if they are not going to be marginalized, in much the same way as 10 years ago they realized that they had to bring system-design expertise in-house if they were going to be able to compete on anything other than silicon price. In a similar way, the processor companies are starting to configure platforms to increase their value in the supply chain.

There are two trends going on here. One is that any work done by a supplier can be amortized across all its customers. So if a semiconductor company writes some software for its platform, it can sell that software to all its customers and thus potentially have an economy of scale that is unavailable to each of its customers writing the software in isolation. The other is that anything that is purchased by a company from a supplier does not form useful differentiation since it is also available to its competitors from the same supplier. As a result, the more standard something is the more it moves to the left in the supply chain. On the other hand, in areas where differentiation is important, companies will do their best to stop the expertise moving to their left and thus reducing their differentiation.

Since in a software-rich chip it is the software that provides most of the differentiation, ultimately this battle will play out as a battle for control of the software. Whoever supplies the software will take the lion's share of the added value to be had. To the extent that semiconductor companies can ship both silicon and its software they will be able to command a price for the software that is above their cost to develop it (spread over their broader customer base) but below what a system company would need to invest to go it alone. They will thus present a better value proposition to their system company customers than other semiconductor companies who do not have software available. To the extent that system companies can stop this happening and keep important software differentiation in-house, they will be able to retain more of the added value and keep the barriers to entry into their business higher, and play the semiconductor suppliers off against each other.

All the same arguments apply to the processor companies who will be forced to supply silicon platforms and then software too, to the extent that it is standard enough for them to spread the cost over all their semiconductor company licensees and thus a whole system segment.

Summary

Moore's law and the compelling semiconductor economics that it drives are driving the growth of software-rich chips, systems implemented using increasingly large quantities of embedded software for their differentiation.

VaST's unique simulation technologies allow architectural exploration and software development to take place using a software model of the chip. This is the only technology

that is both fast enough to allow software development to be done on such a model, and accurate enough that measurements of performance and power can be made from the model that reflect how the eventual system will behave.

This technology enables the transition from a chip-centric view of system design, where the software is little more than an afterthought, to a software-centric view of system design, where chip design is regarded as the creation of an efficient platform for running the software.