Report Information from ProQuest

July 17 2015 19:03

Table of contents

Does virtualization drive the future?

Author: Wilson, Ron

Abstract: Since electronics manufacturers first fabricated transistors using photolithography, the inexorable shrinking that process enabled has changed the world. Since it first became viable to put a stored-program computer on a few chips, the shift of functions from hardware to software has changed the world, as well. These forces have been obvious - with obvious results. But consider a more abstract and less obvious driving force that has, arguably, also been important, and in the future may unleash a revolution as great as those of the IC and microprocessor. You could call that underlying trend "virtualization." From at least the mid-1960s, engineers have used electronic systems to virtualize physical things--either components of the electronic system itself or objects in the outside world--and incorporate those models in place of the real objects. This virtualization has made it possible for electronic systems to behave as if they had hardware that they did not have. It has also allowed systems to behave as if they were interacting with a world from which they were isolated. Virtualization may be the concept that makes system-on-chips (SOC) with diverse computing sites usable in real applications. In fact, we are already seeing indications that this is the case.

Full text: The ability of electronic systems to simulate reality has made them more intelligent. Could it make them self-creating?

Over the 50 years since the first tabloid issue of*Electrical Design News* , underlying forces have driven the evolution of the electronics industry. Since manufacturers first fabricated transistors using photolithography, the inexorable shrinking that process enabled has changed the world. Since it first became viable to put a stored-program computer on a few chips, the shift of functions from hardware to software has changed the world, as well.

These forces have been obvious--with obvious results. But consider a more abstract and less obvious driving force that has, arguably, also been important, and in the future may unleash a revolution as great as those of the IC and microprocessor.

You could call that underlying trend "virtualization." The word is not susceptible to a one-line definition, so let's digress for a moment. When you use electrical quantities to perform physical work or release light, you say the system is electrical. When you use the same quantities--charge, current, or voltage--to convey information rather than merely to do work, you say that the system is electronic. Virtualization is, in this sense, a step beyond electronics. A system--be it a physical process, an object in the real world, or an imaginary person--is virtualized when it has undergone three key steps. First, a boundary must isolate the system from its environment. Second, designers identify the inputs and outputs that cross the boundary, along with the transforms that produce the outputs, thus modeling the system. Third, designers produce a functionally equivalent block--one that accepts the same inputs and produces the same outputs under the same circumstances--with an electronic system.

From at least the mid-1960s, engineers have used electronic systems to virtualize physical things--either components of the electronic system itself or objects in the outside world--and incorporate those models in place of the real objects. This virtualization has made it possible for electronic systems to behave as if they had hardware that they did not have. It has also allowed systems to behave as if they were interacting with a world from which they were isolated--either by distance or by the fact that that world didn't exist. These capabilities have accelerated the growth of electronics and in the future are likely to lend electronic systems capabilities that

in the past were the province of humans alone.

In the beginning

One of the earliest exercises in virtualization was--like many a breakthrough that later became standard practice in computer architecture--an advance in the IBM (www.ibm.com) 360 mainframe family. Before that time, machine-code instructions in computers referenced memory through physical addresses--numbers representing physical locations on the surface of a drum or, after the development of magnetic cores, physical locations within the array of small ferrite cores.

The development of virtual memory rested on the idea that the addresses that machine instructions created needn't be the last word. If arithmetic hardware were fast enough, it could translate addresses from the computer program on the fly into physical addresses using some prearranged mapping. This feature allowed a program that was assembled to run at one location in physical memory to execute in another location--even if the programmer had not made all the memory references relative to the contents of a base register. More important, it also meant that programs could run on a machine whose physical memory was many times smaller than the virtual-address range the program used. A portion of the operating system--a well-developed concept by this time--could allocate portions of the physical memory as necessary for the immediate needs of the program, swapping blocks of information onto and off of disk drives as necessary. By extension, this ability meant that a modest-sized machine could concurrently run a number of large programs, convincing each of them that it had access to all the physical memory it wanted when in fact it was borrowing small blocks of memory on an as-needed basis. Severing the link between the program's address space--which now became virtual--and the physical-address space was a huge and vital step in the creation of modern data processing. But it was far from the last one.

At first glance, it might appear that this scenario has little to do with the definition of "virtualization." However, the process executes all of the steps in virtualization. IBM's designers isolated physical memory from the rest of the mainframe system. They identified the inputs--addresses and write data--and the outputs--timing signals and read data--that characterized the system. And they constructed a combination of hardware, which would become a memory-management unit, and software, which would become the virtual-memory manager that created a virtual main memory.

Virtualizing the IT world

Rich Lechner, vice president of IBM Virtualization (www-03.ibm.com/systems/ virtualization), defines the term as "the logical representation of resources not constrained by physical devices." He points out that, when you use virtualization in this way, it can "either treat one physical resource as if it were many or treat many, possibly dissimilar, resources as if they were one." Lechner traces the beginnings of virtualization to the 360 family's virtual-memory-system debut 40 years ago. But he says the practice has become far broader today than simply virtualization of memory, which is now a common practice even in microprocessors. "At one level, we can gather all of the storage assets available to a network into a single pool of storage," Lechner says. In this way, any program executing in the environment has access to all of the storage assets of the network through a single interface.

But if you are not careful, you will face chaos. The location of stored data does make a difference--in access time, cost, persistence, and coherence. So, for storage, virtualization has come to mean more than just providing a mapping from a single mass-storage API (application-programming interface) to a diverse set of storage devices. "The next level involves the routine cleansing and deduplicating of the storage system," Lechner explains. The virtualization system must make sure to remove stale copies of data, propagate updates, eliminate duplicate data sets, and keep data in a place that is most convenient to its clients. "This all by itself is a significant benefit," Lechner says. "Our field studies indicated that, before virtualization, the average midsized enterprise stores the same data in at least 20 places around the network."

The process does not stop with storage systems. In just the same way, system programmers can identify the

computing resources in a network, give them wrappers that present a common API, and hence virtualize them. In this way, the computing power available to an application becomes a slice of the entire computing community on the network, not simply the power of the machine on which the application happens to be running.

The next step in this process is even a bit more abstract. You can virtualize storage devices and servers, but what about applications? In exactly the same way, system programmers can draw a wrapper around each application, provide it with a set of APIs, and make it available to the network as a virtual application--say, a virtual database. Thus, when an application program executes in the network, it may be interacting with virtual-storage devices on a number of storage networks, running on a virtual processor when parts of the code are executing on a half-dozen servers around the network, and calling virtual applications that may be data-bases from different vendors with different organizations.

IT to embedded computing

All this virtualization might seem to be irrelevant to anything outside the IT (information-technology) world. But if you think of a multicore embedded system--based on the IBM Cell processor, for example--as a heterogeneous collection of computing resources, storage assets, and interconnects, perhaps the relevance becomes more clear. As has so often been the case, the IT solution of today is the SOC (system-on-chip) solution of tomorrow. Virtualization may be the concept that makes SOCs with diverse computing sites usable in real applications.

In fact, we are already seeing indications that this is the case. At IMEC (Interuniversity Microelectronics Center, www.imec.be) in Leuven, Belgium, researchers have created a virtual model of a runtime-configurable, multicore-computing system for software-defined radio. Antoine Dejonghe, a principal scientist at IMEC, describes the situation: "IMEC's M4 project is creating a radio system that can be agile in real time across wireless protocols and media types," he says. "We believe that technology scaling by itself won't be enough to bridge the energy gap between what our configurable architecture requires and what batteries will be able to provide. So, the viability of the system depends on being able to model the entire system, from analog front end through the media-access-controller software, in terms of its energy consumption. We will use this model to establish the optimum trade-off between energy and quality of user experience at runtime--perhaps as often as every 10 msec."

Today, the models are under construction--using the same sequence of steps--to create a virtual software-defined radio. Designers have theoretically modeled the analog front end, for instance, with approximately 11 control bits as inputs, along with theoretically calculating output power, distortion, and energy consumption. The designers are now calibrating these models against actual silicon measurements. They will construct similar models to predict the performance and energy consumption of the configurations of IMEC's ADRES (architecture-for-dynamically reconfigurable-embedded-system)-computing cores in the radio project.

The result will be a virtualization of the radio that should execute in software, consuming less than 5% of an ARM9 and continually optimizing the radio for current traffic, link quality, and application variables (**Figure 1**). This process, Dejonghe believes, can bridge the gap between what batteries can deliver and what battery life users will demand.

Creating a virtual world

IMEC's work creates a virtualization of a physical system. However, designers are virtualizing larger and more complex systems. Another easy place to find excitement in virtualization is in the world of electronic games. Traditionally, that virtualization did not exist. Most video games today have a lot more in common structurally with a Walt Disney cartoon than with a simulation of the physical world. Games, like cartoons, have story lines. The choices of the player do not directly interact with the world of the game; players merely choose story lines, and the game proceeds down one of the paths the scriptwriters designed for it.

This scenario is true even at the macro level. When you slip around the corner and level the four-eyed alien from the planet *Grr* with your super halitosis blaster, you unleash a sequence of animation frames, in which the Grrien satisfyingly rips open, tumbles through the air, and lands as a puddle of unfamiliar proteins. This

sequence is almost the reverse of motion estimation in video-compression algorithms: It begins with a set of key frames and animates incremental motions of objects against a relatively fixed back ground. To the player, the result is much the same no matter which way he triggers the sequence, unless he misses.

As games become richer and players become more sophisticated, this situation causes a problem, according to Manju Hegde, chief executive officer and chairman of gaming "physics-engine" developer Ageia. He has a vested interest in this problem because Ageia supplies software and hardware accelerators for performing the dynamic computations necessary for eliminating the animation sequences and computing the trajectories of the spinning Grrien bits.

No one--least of all, the animators--would dispute that it is better to do a dynamics-based simulation of the objects in the game world than to rely on animation sequences. However, the greatest benefit would be to the game architects. Because of the labor involved in producing animation sequences from key frames, a game can have only a limited number of animation sequences. So, architects must confine the action of the game so that only a limited number of outcomes are possible at any time. The art of game design today is to make the game feel rich and unscripted without causing an exponential explosion in the number of key frames that developers must prepare.

Hegde illustrates with a rather less violent example: a basketball game. "If you want a realistic slam-dunk sequence in your game, you start out by rigging lights all over your star player and then you record him doing some dramatic dunks," Hegde explains. "Then, back in the lab, you extract from the recording key frames and interpolate the movement of the image edges to produce animation. This animation can look natural on the screen, but, any time you push the slam-dunk button in the right context, you are going to see the same sequence."

The alternative approach is to build a physics-based model of our hero--that is, to virtualize him. "We create a physics-based model of the body," Hegde says. "Today, it can be as detailed as having approximately 200 bones connected by six kinds of joints. We then model each of these bones and joints according to the laws of physics. You apply forces to them, and they respond. Now, the dunk becomes the dynamics of the individual bones and joints in the model person. If you view a game from a distance during fast action, a game might use just 20 bones to reduce the calculations, but it looks 'right.'" Hegde thus describes a scenario that fits this article's definition of virtualization--in this case, of a basketball forward.

This virtualization has so many advantages over conventional animation that manufacturers would--except for a couple of issues--produce all games in this way. One of these issues goes back to scripting. If the outcome of the player's input depends on both physics and the game script, the sequence of play can quickly become unmanageable. What if physics dictates that the player breaks his wrist on the rim of the basket and leaves the court writhing in pain? Game architects who employ physics-based models often intervene in the simulation to direct it to allowable outcomes, so that the game stays within its script. This delicate business blends physical simulation and animation.

A more brutal problem is the amount of computing requirements. "Today, an AMD FX-62 dual-core CPU running at 2.8 GHz may be able to handle a couple of characters with extensive bone models," Hegde says. "But you couldn't do physics-based simulation with a large number of characters on the screen at once. The bursts of computation necessary to support a number of characters all running, for instance, would overwhelm the processors."

Microsoft's Robotics Studio (www.microsoft.com/robotics) is developing a similar application, also using Ageia's technology. Rather than playing games, the Microsoft group provides a virtual-development environment for the programming--and, eventually, the design--of robots. Tandy Trower, general manager of the group, says that the need for such an environment is obvious across the spectrum--from industry to education. On one end, with a KUKA (Keller und Knappich Augsburg) Robotics (www.kuka.com) robotic arm selling for more than $100,000, industrial developers need a low-cost environment in which to develop and test programs. At the other extreme,

robotics has proved to be one of the few endeavors that can attract US students to engineering and mathematics. Even simple robots, however, are far beyond the reach of most secondary schools, even though students have proved capable of programming--and even designing--them. So an affordable virtualization of a robot and its surroundings would be a big win, and Microsoft is attempting to achieve that goal.

Building a virtual world behind the Direct-X graphics environment, the company has provided libraries of models for popular robots; primitives for constructing physical objects, machines, and other robots; and a physics-driven simulation engine to animate them. "There are two ways of creating entities," Trower explains. "Developers can write them directly as code--in C, Visual Basic, Python, and the like--through a managed code interface or assemble them from basic geometric shapes and assign them physical characteristics, such as mass, hardness, and so on" (**Figure 2**). Once you set them in motion, the robots, which are themselves entities, can interact with other entities, including other robots, in a world that physical laws govern.

Trower points out that, although Microsoft's work in this area is on one level similar to the physics-based modeling that is starting to appear in games, it differs dramatically at another level. "Games take place in a well-defined environment," he says. "Robot simulation does not. You have to work out everything that happens based on physics, because there is no script." Trower says that the virtualization technology could move from the development tool into the robots themselves. For example, you can blend the simulation with real-world sensor data. A robot that can include an optional--and expensive--laser range finder, for example, might instead have a virtualized range finder; fusing other sensor inputs to generate the range data. In the future, you may see the next step: robots virtualizing the world around them, a scenario that Brooke Williams, DSP-automotive-vision-marketing manager at Texas Instruments (www.ti.com), sees before his own eyes.

"We are starting to see automobile-safety systems fusing sensor data to create a virtual model of the car's surroundings," Williams says. "This model can either be presented to the driver as warning information, or it can be used to take control of the vehicle." For example, TI is combining radar, a good tool for detection and ranging but poor for forming images, with machine-vision systems, which are great at finding edges and bearings but poor at ranging or detection. This combination of tools will allow the creation of virtual models of the objects surrounding a car. "The object is to predict a crash; prepare the vehicle by tightening seat belts, arming air bags, and closing the windows, for instance; and attempting to take evasive action," Williams says.

But to achieve this goal, simple proximity warning is insufficient. The tools must identify objects from their surroundings and track and categorize them; the insurance industry must know that the system can distinguish between pedestrians and shrubbery, for instance. You want to neither turn away in panic from a car that can't physically reach your trajectory nor mow down a pedestrian to avoid destroying a hedge. "Manufacturers are even talking about external air bags that could deploy to protect pedestrians in collisions," Williams says. Such decisions require not just a measurement, but also an understanding of the car's surroundings.

Therein lies a possible endpoint for the trend of virtualization: electronic systems that can not only sense, but also model and predict their environment. Such systems exhibit artificial intelligence and also express virtualization. These systems, protecting drivers from their own folly, exploring the otherwise-inaccessible reaches of the physical world, and using their virtual models to reason about their surroundings, represent another major expansion in the role of electronics.

Sidebar

Software I/O, virtual I/O, or software-assisted I/O?

By David Fotland, Ubicom Inc

Typical microprocessor families have dedicated hardware for each I/O function. This difference leads to families of chips with the same CPU but different I/O mixes. Cost is higher because the semiconductor company must make many chip versions, and mask costs are high in state-of-the-art process technology. The alternative is to create an SOC (system on chip) with all of the I/O hardware on the same chip. This approach also leads to higher cost, because the customer is paying for silicon to implement I/O that he won't use in his application.

The solution to this problem is software I/O. Some 8-bit microcontrollers use this technique, called "bit-banged" I/O. If the microcontroller has on-chip memory and deterministic execution, the software can directly control I/O pins to implement the I/O protocol. A simple example is a UART. The start bit causes an interrupt, and software reads the input pin to receive the data. While the data is arriving, the CPU cannot do anything else, so this technique is useful only for I/O that is infrequent or intermittent. The interrupt-response time limits the use of this technique to low-speed I/O.

Some 32-bit processors, such as those from ARM (www.arm.com) or MIPS (www.mips.com), can't use software I/O because code execution is far from deterministic. Pipeline hazards and cache misses make it impossible to use instructions for accurately timed external events. Operating systems such as Linux turn off interrupts for milliseconds at a time, making real-time I/O response impossible.

Ubicom (www.ubicom.com) has the only 32-bit CPU that uses software I/O. The multithreaded CPU has a hardware scheduler that can select a thread for execution during every clock. Real-time threads have a fixed schedule and deterministic execution, even if other threads have mispredicted branches or cache misses. The unit has 10 threads, so it can allocate one real-time thread to each I/O port to manage that port.

The instruction set supports software I/O and packet processing. An instruction can move data between memory and I/O. MIPS and ARM CPUs, in comparison, need two instructions: a load and a store. Single instructions can set, clear, or test any I/O bit. Interrupt-response time from an I/O event to scheduling instructions in the managing thread takes only a few CPU clocks. When an I/O port is idle, it suspends its managing thread, using no CPU resources.

The high-performance, 32-bit CPU can use software-I/O for functions more complex than a UART. Ubicom has implemented a full PCI bus at 27 MHz, MPEG Transport Stream, IDE, and Utopia in software. It has also implemented MII (media-independent interface) for 10/100-Mbps Ethernet, USB, SPI, GPSI (graphics-processor software interface), and other serial interfaces with a combination of hardware and software. By spending 10 to 20% of the CPU throughput on software I/O, the company dramatically reduced the die area necessary for I/O, resulting in a flexible single chip to cover a wide range of applications.

Author's biography

David Fotland is chief technology officer of Ubicom, where he led the architecture of the Ubicom multithreaded processor for packet processing and software I/O. He is responsible for all aspects of architecture and definition of processors and solutions.

Previously, Fotland spent 21 years at Hewlett-Packard Co (www.hp.com) as lead engineer, project manager, and system architect. He was a key participant in the PA-RISC instruction-set definition and was lead designer of the first PA-RISC processor and system. He was a leader in the development of the HP-Intel (www.intel.com) Itanium instruction set.

Sidebar

Immersed in engineering, advanced 3-D visualization promotes insight

By Jeff Brum, Fakespace Systems

As electronics speed into an era in which manufacturers fabricate not just circuits, but also physical structures themselves in nanoscale geometries, the role of computer-based simulation as a design tool is increasingly important. Correspondingly, the benefits of visualization in the review and analysis of simulations play a growing role. Looking to the future, immersive stereoscopic display tools will amplify the power of visualization.

The adoption of immersive visualization in electronics design revolves around several factors. Atomic-scale phenomena, which are major concerns as the semiconductor road map extends beyond the 65-nm-process node, have been major players in advanced visualization techniques. Scientists at NIST (National Institute for Standards and Technology, www.nist.gov), for example, use a stereoscopic display--two walls and a floor--to gain insight into the molecular bonding of "smart-gel" polymers (Figure A and Reference A).

Similarly, researchers at LANL (Los Alamos National Laboratory, www.lanl.gov) use a range of immersive

environments--from wall-sized to a 43-million-pixel, five-walled projected room--to view terascale data sets (Figure B). Bob Green, visualization specialist at LANL, notes that the researchers "are viewing simulations based on computations that generate more data than is contained in the entire print collection of the Library of Congress in one calculation."

In engineering, visualization has had its largest impact to date in macroscale CAD programs, such as automotive and aerospace design, and the evaluation of complex structures for interferences that are not readily apparent in simpler graphical representations. As simulation and visualization data accumulate in MEMS (microelectromechanical-system) design, this type of modeling and simulation will grow in importance. The ability to visualize and "fly through" transistor-scale structures and even large segments of a complex microcircuit design will also benefit from advanced 3-D visualization that blends multiple streams of data, including device characteristics, interconnects, and material behavior.

Wall- or even room-sized immersive displays support a more collaborative, more intuitive, style of work than do graphics processors. A major benefit in the review of complex structures is the ability to "move" freely along all three axes of an image and still maintain the visual acuity of high-resolution desktop processors. (Today's projectors support resolution of 1400×1050 pixels or higher.) Viewing nanoscale design can feel like taking a helicopter tour over a dense city center with the ability to identify and zoom into landmarks. Improved insight and collaboration lead to faster and better decisions, speeding time to market and reducing development costs. Although researchers are just beginning to measure the benefits of immersive visualization (Reference B), they generally agree that they bring new levels of insight to engineers and scientists. In immersive visualizations, groups of users can view the inner workings of devices and gain deeper understanding of electromagnetic effects and the relationships between elements of a design. With the availability of dual PC clusters and advanced graphics cards, these types of virtual environments no longer require specialized graphics supercomputers. The increased accessibility of immersive visualization makes its addition to the tool kit of electronic engineers practical and inevitable.

References

"Recipe for a Shake Gel," National Institute for Standards and Technology, Aug 27, 2003, www.nist.gov/public_affairs/newsfromnist_smartgels.htm .

Kraak, Menno Jan, "Beyond Geovisualization,"*IEEE Computer Graphics and Applications* , May/June 2006, Volume 26, No 3, pg 6.

Author's biography

Jeff Brum is a vice president at Fakespace Systems (www.fakespace.com), a division of Mechdyne Corp with a 15-year history of developing and supporting advanced systems for immersive visualization in science, engineering, and public-exhibition applications.

AuthorAffiliation

You can reach Executive Editor Ron Wilson at 1-408-345-4427 and ronald.wilson@reedbusiness.com.

Subject: Electronics industry; Research & development; R & D; Statistical data; Information technology; Embedded systems;

Location: United States, US

Classification: 8650: Electrical & electronics industries; 5400: Research & development; 9140: Statistical data; 9190: United States

Publication title: EDN

Volume: 51

Issue: 20

Pages: 124-136

Publication year: 2006

Publication date: Sep 28, 2006

Section: Features; Design Feature

Publisher: UBM LLC

Place of publication: Boston

Country of publication: United States

Publication subject: Engineering--Electrical Engineering

ISSN: 00127515

CODEN: EDNSBH

Source type: Trade Journals

Language of publication: English

Document type: Cover Story

Document feature: Illustrations Graphs Photographs

ProQuest document ID: 222434637

Document URL: http://search.proquest.com/docview/222434637?accountid=9253

Last updated: 2014-02-05

Database: ABI/INFORM Complete

Bibliography

Citation style: IEEE - Institute of Electrical and Electronics Engineers

[1] R. Wilson. Does virtualization drive the future? EDN 51(20), pp. 124-136. 2006. Available: http://search.proquest.com/docview/222434637?accountid=9253.