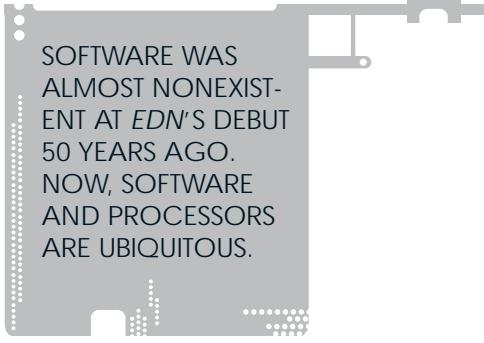


ROB MAGIERA / NOUMENA DIGITAL

SOFTWARE AND PROCESSORS: HERE, THERE, AND EVERYWHERE



SOFTWARE WAS ALMOST NONEXISTENT AT *EDN*'S DEBUT 50 YEARS AGO. NOW, SOFTWARE AND PROCESSORS ARE UBIQUITOUS.

D

uring *EDN*'s inaugural publication year in 1956, hardly any software existed. Although the University of Cambridge's EDSAC—the first stored-program computer—became operational in 1949, computers and programmers remained rare until 1954, when IBM (www.ibm.com) introduced the model 704. In 1956, the first PC (“personal” computer) and the first significant HLL (high-level-language) compiler appeared, helping to spark the explosive growth of computers and software. Systems based on the DNA-like intertwining of software and stored-program processors have now covered the planet and ventured into interstellar space (see

sidebars “Software-ization” on pg 144 and “Hardware morphs into software” at www.edn.com/50th).

THE FIRST COMPUTER PROGRAM

Manhattan Project physicists Nick Metropolis and Stanley Frankel created and ran the world's first computer program on ENIAC (electronic numerical integrator and computer)—the first programmable electronic computer—two months after the end of World War II (**Reference 1**). ENIAC lacked stored-program abilities; plug wires, patch-panel configurations, and switch settings comprised its programs. During most of the war, Frankel ran a computing-service bureau at Los Alamos, NM (birthplace of the atomic bomb), staffed with calculator operators (called “hand computers”) and punch-card tabulating-machine operators. Frankel and Metropolis developed complicated numerical-analysis algorithms for tough nuclear-physics problems and “ran” these algorithms on the tabulators and hand computers. Frankel became so obsessed with the automated tabulating machines' abilities that he ignored his managerial duties. In January 1945, management transferred Frankel to Enrico Fermi's F Division to work with Edward Teller on the thermonuclear “super” bomb. That move took Frankel to ENIAC.

Metropolis and Frankel arrived at ENIAC's home in the Moore School at the University of Pennsylvania with 1 million punch cards containing their initial program data. The "Los Alamos Problem" that ran on ENIAC in late 1945 and early 1946 was a 1-D nuclear-fusion simulation. Details remain classified, but this calculation was the most complex that anyone in the history of science ever attempted when it ran as ENIAC's "shakedown" program, flushing out many hardware bugs. ENIAC's limited numeric storage—20 10-digit accumulators—forced the physicists to oversimplify their equations, so the numeric result itself wasn't especially meaningful. The real goal of the exercise was to see whether ENIAC or a much bigger version of ENIAC could help develop Teller's super bomb. The conclusion? Yes.

After the war, Frankel consulted for defense contractors but lost his security clearance in 1949. California Institute of Technology (www.caltech.edu) hired Frankel to run a computing-service bureau, where he spent most of his time studying the nascent fields of digital logic and computer design. He dreamt of small, affordable computers that schools could purchase. Frankel designed and bread-boarded a computer he called MINAC (minimal automatic computer). It used only 15 flip-flops and stored its register values and memory contents on a rotating-drum memory that Caltech physics major James Cass built by hand.

THE FIRST PC

MINAC was slow, cheap, reliable, and marketable. Librascope—a nearby military contractor wanting entry into the commercial computer market—licensed MINAC from Caltech, hired Cass, and engaged Frankel as a consultant. Cass' engineering group "productized" MINAC. The resulting LGP-30 computer—a compact, desk-sized machine that employed a Flexowriter (an electromechanical typewriter with paper-tape reader and punch) for a programmer's console—required only 115 vacuum tubes and 1450 germanium diodes. It fit into a small room and needed no special climate control. A programmer could enter the room, close the door, and be alone with the machine.

Librascope announced the LGP-30 in mid-1956. It was the first PC. The com-



pany built more than 500 LGP-30s—a phenomenal number in an era in which most computers were one-of-a-kind machines. The pool of LGP-30 programmers grew large enough to form the Pool user group, and one-third of the LGP-30s went into business applications. Librascope even transformed three LGP-30s into early embedded systems for industrial-process control.

Librascope shipped the first LGP-30 in September 1956. One month later, IBM published the first Fortran programmer's reference manual for the model 704, IBM's first mass-produced computer with magnetic-core memory. IBM programmer John Backus had assembled an HLL-development team shortly before the machine's introduction in May 1954. He sought a more abstract, algebraic language that would make programming faster, cheaper, and more reliable to stem rising assembly-language-programming costs—which were already approaching hardware costs. Backus unveiled Fortran in a paper he presented at the Western Joint Computer Conference in February 1957.

As Frankel had foreseen, many colleges and universities purchased LGP-30s. Dartmouth College (www.dartmouth.edu) acquired an LGP-30 in 1959. John

Kemeny, a hand computer at Los Alamos, NM, during the war; Thomas Kurtz; and their students developed many simplified HLLs on the LGP-30, including Algol 30 and Scalp (self-contained Algol processor), which led to the development of Basic on a time-shared GE-225 computer in 1964.

Computer and software development exploded throughout the 1950s and 1960s. Large-scale software development became practical as HLLs, including Fortran and Basic, spread. Minicomputers, starting with the PDP-1, which DEC (Digital Equipment Corp) introduced in 1959 (www.computerhistory.org/pdp-1), lowered the cost of computing. Fortran II, III, IV, and more powerful programming languages, such as Algol 60 and Pascal, fueled a growing army of programmers. However, computers and software still failed to proliferate widely. Hardware costs remained too high, but the situation was about to change.

THE KITCHEN COMPUTER

A glimpse of the near future appeared in an unlikely place. The 1969 Neiman Marcus catalog featured the \$10,600 "kitchen computer"—a Honeywell H316 minicomputer souped up in a futuristic, bright-red fiberglass cabinet, supposedly to increase its appeal to women. Honeywell built only a prototype. Neiman Marcus sold none of the units, but falling hardware costs would soon allow computers to penetrate low-cost markets, including the home. Honeywell advertised its H316 minus the glitzy cabinet as the first mini-

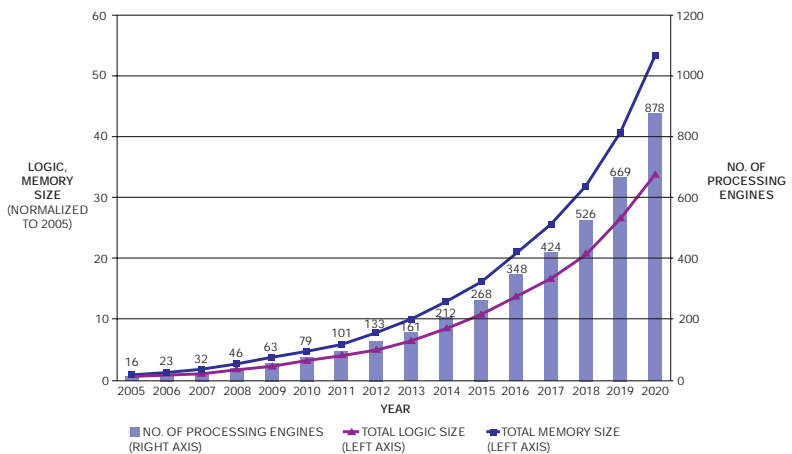


Figure 1 The ITRS (International Technology Roadmap for Semiconductors) predicts that future SOCs will incorporate dozens and then hundreds of processors.

computer to cost less than \$10,000. For computers to truly proliferate, industry needed another breakthrough.

In 1969, the same year that the kitchen computer appeared, Japanese calculator manufacturer Basicom sought a semiconductor company to fabricate chips for a family of calculator-based products. Basicom's Osaka and Tokyo factories separately approached Mostek and fledgling Intel Corp (www.intel.com) because only these vendors had high-density silicon-

gate-MOS processes. The Tokyo factory struck a deal with Intel for the more complex calculator-chip set. The chip set, which Basicom's Masatoshi Shima designed, was based on ROM-driven decimal state machines. Intel President Bob Noyce assigned the job of Basicom liaison to Ted Hoff, the company's application-research manager (references 2 and 3).

Basicom's engineers planned several chips, each to come in expensive (for the time) 40-pin DIPs. Once he fully

understood Basicom's plan, Hoff knew Intel couldn't deliver the chips at the agreed price. The logic required large chips, and the 40-pin packages were expensive. Hoff alerted Noyce and suggested an alternative: a CPU-like logic chip with ROM-based software. Noyce, a mathematician and device physicist, didn't understand how a computer on a chip running software could replace logic, but he got the drift and encouraged Hoff to pursue the concept. Busi-

"SOFTWARE-IZATION"

By James Truchard, PhD, National Instruments

"Software-ization" may be a made-up word, but it accurately describes the trend in design. Today, hardware is what is left over when you finish with the software. With the growing popularity of reprogrammable hardware, such as FPGAs, and software techniques, such as digital-filter design and modulation, the software logic is becoming the most important aspect of a design, making the world increasingly "software-ized."

The increased focus on software design is evident in the design teams of today. According to Venture Development Corp (www.vdc-corp.com), the average software-design team now numbers 3.9 members versus 2.3 for hardware-design teams (Reference A). The trend toward software-ization does not end there: The need for programmable hardware is evident in the increasing number of programmable devices, such as FPGAs (Reference B). The need is clear: Designers want to reprogram their applications, which makes economic and efficiency sense. Managers prefer this approach because it means that you get more out of the hardware you purchase because you can reprogram it as specifications change, efficiently develop prototypes, and add features to deployed systems.

Another important angle emphasizing the need for

software-ization is related to the "long-tail" business model (Figure A). The long tail describes how future business focus should be on "odd jobs" or millions of niche markets versus a few high-volume vertical markets. Examining vertical markets, such as cellular-telephone or plasma-television designs, you find plenty of reference designs specific to those markets. Those working in the long tail on applications from laser wrinkle removers to underwater surface crawlers have no access to reference designs; nevertheless, the expectations of products delivered in the long tail do not diminish. You still have the same pressure to deliver high quality and innovation but without the tools or reference designs available to help. At this point, software-ization couples with an integrated platform to help design engineers. Software-ization dynamically adjusts to the changing world of odd jobs. You need an integrated, off-the-shelf platform, so that when you implement the design in hardware, you need not hire any more hardware engineers (Figure B).

Another benefit of using software-ization with off-the-shelf hardware platforms becomes evident when examining test challenges with products in lower volumes. Tests on high-volume cell phones are thorough and costly, but high sales volume offsets this time and expense. However, with lower volumes of, say, 10 to 100 annually, it no longer makes economic sense to perform corner-case

continued on pg 146

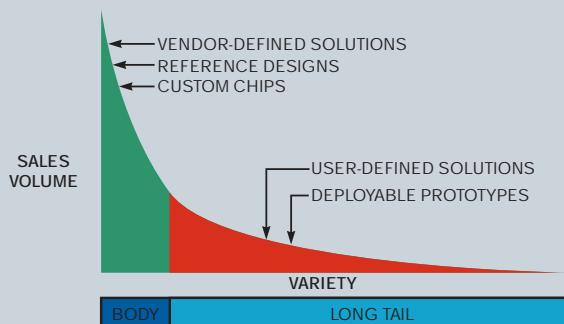


Figure A The long tail describes how future business focus should be on "odd jobs" or millions of niche markets versus a few high-volume vertical markets.

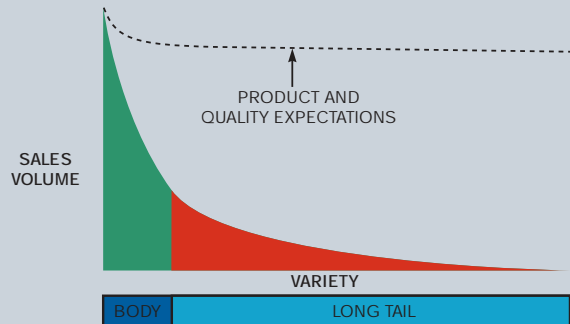


Figure B You need an integrated, off-the-shelf platform, so that when you implement the design in hardware, you need not hire any more hardware engineers.

com eventually agreed to this approach.

Hoff, Shima, and Stan Mazor, newly arrived from Fairchild, developed a four-chip set—the CPU, a 320-bit RAM, a 256-byte masked ROM, and a 10-bit shift register for I/O—linked by a multiplexed 4-bit bus. (The RAM and ROM chips included 4-bit I/O ports.) This approach reduced IC-die cost and package size, sharply cutting the cost of the chip set. Intel packaged the RAM and ROM in 16-pin plastic DIPs and the CPU in a 16-pin ceramic DIP. These plastic and ceramic

packages cost 18 cents and less than \$1, respectively. Intel snagged Federico Faggin from Fairchild in March 1970 to translate the design into silicon. With Shima's help, he'd developed working silicon for the four chips by January 1971—just nine months later.

Busicom, hurt by falling calculator prices, renegotiated chip prices during 1971. Noyce exchanged a price cut for the right to sell the chips into noncompeting applications, and Busicom's CPU debuted as Intel's 4004 microprocessor at the Fall

Joint Computer Conference in November 1971. Since then, system designers have incorporated billions of microprocessors, microcontrollers, and DSPs from dozens of vendors into many applications.

RISE OF THE COMPILERS

The microprocessor's debut allowed engineers to exploit software's problem-solving abilities in a vast array of electronic products. At first, machine code and assembly language were the only microprocessor-programming alterna-

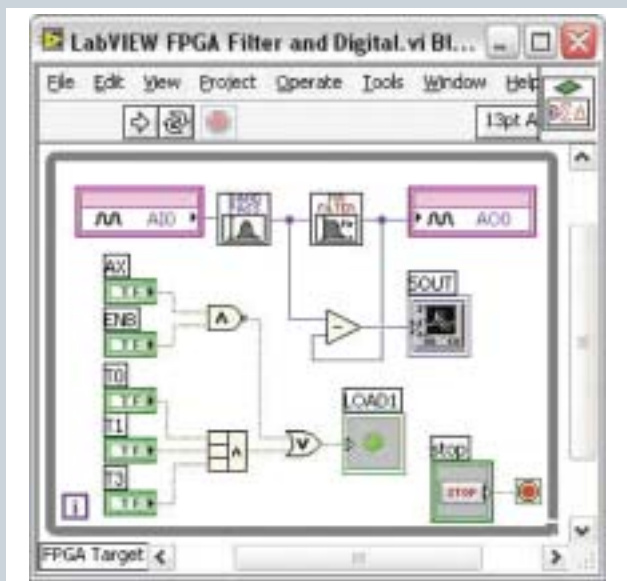


Figure C You implement a software-defined filter, such as National Instruments' LabView, in hardware.

continued from pg 144

testing, but you still need a high-quality product. The more efficient approach is reuse. Tested, off-the-shelf hardware platforms often come with industrial certifications, including international safety, EMC (electromagnetic-compatibility), shock, vibration, and environmental ratings. So, you can reuse this tested equipment and incorporate certified products into your design process for a much lower cost.

What does software-ization look like, and what does it mean to have software-program hardware? Software to program hardware must include syntax and semantics with explicit notations for expressing time and concurrency—primary attributes of hardware. Traditional, sequential programming languages do not exhibit these behaviors, making these applications difficult to develop.

Using the right software tool to program hardware is

happening in many applications, including custom-test digital protocols with FPGAs; communications logic running in a DSP; and remote updating of deployed, embedded systems on a real-time microprocessor.

One of the most common design tasks—digital-filter modeling and design—takes advantage of software-ization. The use of digital filters eliminates a number of problems that their analog counterparts face. In software, you can attenuate unwanted signal elements such as noise caused by electrical components and environmental effects, apply antialiasing algorithms to test data, and reduce sample sets with decimation. Digital filters find use in a variety of applications ranging from machine-condition monitoring and animal-vocalization detection to seismic signal decomposition and audio special effects (Figure C). Once you design your filter with an integrated hardware platform, you can implement the filter without knowing the hardware details—another example of the beauty of software-ization.

Software-ization is a good thing for everyone. It means more manageable projects, lower hardware costs due to more reuse, and an easier path to more innovation due to the inherent ability to more quickly experiment, iterate, and implement.

REFERENCES

- A** Lanfear, Chris, "Embedded software strategic market intelligence program, 2005, Volume 8: Embedded Systems Market Statistics," Venture Development Corp, www.vdc-corp.com/Purchase.asp?viewtype=detail&id=1823.
- B** Bartos, Frank, "ASICs Versus FPGAs," *Control Engineering*, June 2005, www.manufacturing.net/ctl/article/CA607224.
- C** Anderson, Chris, "The Long Tail," *Wired Magazine*, October 2004, www.wired.com/wired/archive/12.10/tail.html.

AUTHOR'S BIOGRAPHY

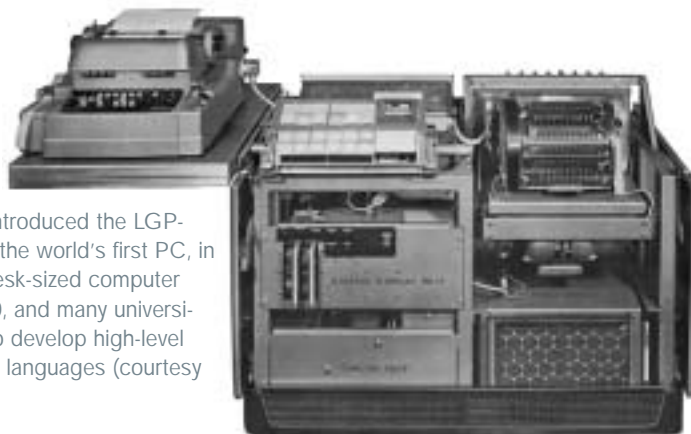
James Truchard, PhD, is co-founder, president, and chief executive officer of National Instruments.

tives. Early HLL compilers for microprocessors appeared, but they generated slow, memory-hungry object code rather than well-written, handcrafted assembly code. Early microprocessor-based systems were already slow and memory-starved, so early compilers were nearly useless, but they evolved.

The same year that Intel introduced the 4004 microprocessor, Dennis Ritchie started extending Bell Labs' B programming language for the PDP-7 minicomputer by adding a character type (**Reference 4**). Ritchie called this new language NB (new B). NB was an embryonic version of C, and the language had become recognizable as C by 1973. Ritchie continued to work on C and published *The C Programming Language* with Brian Kernighan in 1978 (**Reference 5**). This book served as a de facto C standard throughout the 1980s. The embedded-development world dabbled with many HLLs, including C, Basic, Forth, Pascal, and a microprocessor version of PL/I called PL/M, but superior performance and memory efficiency kept assembly-language coding in the lead.

Throughout the 1980s, embedded systems became increasingly complex, compilers improved, and the cost of 32-bit microprocessors decreased. Rising software costs and complexity eventually curtailed assembly-language programming on microprocessors, mirroring events that spurred Backus to develop Fortran for mainframes. C's popularity as an embedded programming language climbed. ANSI established the X3J11 committee to produce a C standard in the summer of 1983. When C became both an ANSI and an ISO standard in 1990, C became king.

By 1990, microprocessor-centric design was the first choice of board-level-system designers. Logic design became a last resort, which designers used only when microprocessors were too slow. Around 1995, microprocessors became cores—just part of an IC—transforming ASICs into SOCs (systems on chips). Early SOCs incorporated only one



Librascope introduced the LGP-30, arguably the world's first PC, in 1956. The desk-sized computer cost \$27,000, and many universities used it to develop high-level programming languages (courtesy Bob Lilley).

microprocessor; then, two; then, many. The ITRS (International Technology Roadmap for Semiconductors) predicts that this trend will continue (**Figure 1**). Like board-level-system design before it, SOC design became processor-centric (**Reference 6**). Many SOCs today incorporate dozens or hundreds of intercon-

nected processors. Microprocessors and software are the very fabric of contemporary electronic design. That fabric now covers the planet, as many embedded-microprocessor applications demonstrate (**Table 1**). In fact, processors and software now reach beyond the earth.

The first microprocessor in space, an

TABLE 1 EVERYDAY USES FOR EMBEDDED MICROPROCESSORS AND SOFTWARE

Office and retail	Home
Telephones and PBXs	Conventional and microwave ovens
Printers	Food processors, mixers, and blenders
Copiers and faxes	Refrigerators and dishwashers
Postal scales and shipping management	Climate and lighting control
Fire and intrusion alarms	TVs, cable and satellite boxes, VCRs, and DVRs
Lighting and HVAC controls	Home-entertainment systems
Elevator and automatic-door controls	DVD, CD, and MP3 player/recorders
Bar-code and RFID readers	Clothes washers, dryers, and irons
Video security and monitoring	Corded and mobile telephones
Energy and utilities monitoring and billing	Toys and games
Record keeping and management	Digital cameras and camcorders
Inventory management	Barbecue grills
Civil ground transportation	Space, aviation, Naval, and military
Drive-train control	Engine control and management
Passenger-cabin climate control	Guidance and attitude control
Entertainment systems	Onboard systems monitoring
GPS and compass navigation	Radar and collision avoidance
Collision-avoidance systems	Global and celestial navigation
Mobile-phone and satellite communications	Radio and satellite communications
Fare collection (public transport)	Passenger-cabin climate control
Traction control and automatic braking	Fuel management
Active shock absorbers	Weapons management and control
Manufacturing	Medical
Process control	Diagnostic, imaging, and treatment systems
Robotic assembly and transport	Patient record keeping
Inventory management and tracking	Robotic surgery
Energy and load management	Pharmaceutical dispensing and inventory control
Security, safety, and access management	Therapeutic and rehabilitation systems
The Internet	
Routers and switches	
Network interfaces and bridges	
Cable/DSL modems and gateways	
Web, mail, search, and storage servers	
Firewalls	
Wireless access points and repeaters	

THE CASE FOR APPLICATION-SPECIFIC PROGRAMMABLE LOGIC

By Alex Lidow, International Rectifier

The “software-ization” of an enormous range of applications has brought more features and lower product costs to end users and shorter development times and lower product-lifetime-management costs to manufacturers. By and large, these applications exploit the processing power of microprocessors, microcontrollers, and DSPs. They depend on the availability of off-the-shelf programmable logic that economically provides greater processing power than the application’s peak processing requirements demand.

Real-time applications as disparate as motion control, communications, and image processing test the limits of software implementations running on general-purpose programmable hardware. In these applications, complex processing requirements combine with high data rates or signal bandwidths to drive hardware usage and code complexity to uneconomic extremes.

In motion control, for example, high-speed control loops depend on real-time vector-transform calculations to determine power-switching times within a rotating frame of reference. A second, inverse, set of transforms operates on the motor’s feedback signal. The complexity of these calculations exceeds the processing bandwidth of low-cost processors based on traditional architectures. The “bigger hammer” approach—using more powerful general-purpose processors fabricated on faster semiconductor processes—doesn’t sufficiently improve computational efficiency and can drive both silicon and software-development costs to noncommercial levels.

Still, the allure of programmable loops and the accompanying benefits to OEMs of reduced development cost and time savings merits the rethinking necessary to achieve high bandwidth and low cost. Though traditional microprocessor, microcontroller, and DSP architectures may not be best for these applications, custom-hardware implementations often poorly support rapid product-development cycles or easy design maintenance.

As high-speed communications and image processing have done before them, motion-control applications are taking advantage of computational engines—application-specific logic—that serve as a programmable interface between an application-management layer and a real-time process. Computational engines offer a middle ground between software-ization and fixed-topology hardware that benefits OEM designers by reducing both development time and project risk in these computationally intensive real-time applications.

At the real-time-process interface, computational engines efficiently support high-bandwidth control

loops through application-specific hardware peripherals, low-level firmware, and vendor-provided algorithms. OEM-configurable computational resources allow product developers to customize the engine’s resources in ways that parallel the high-level coding of general-purpose programmable logic. In the case of the engine-based implementation, however, the development environment can support programming in application-relevant terms.

Visual-programming environments can go a step further, allowing product developers to specify signal flows between computational blocks without coding per se. Such high-level constructs do not preclude OEMs from customizing the lower level operating algorithms or developing new algorithms from scratch, but they do provide for significantly shorter development and substantially improved code reuse. This balance between hardware and software resources promotes rapid prototyping, quick development of derivative products, and low-cost design maintenance.

On the application-control end of the design, the engine can present an intelligent interface to either a co-integrated or co-embedded microcontroller. The small microcontroller can efficiently manage the user interface—panel switches and displays—and direct the outermost application-control loop. The advantage of this functional segmentation between the microcontroller and the real-time engine is that the interface between them can use physical quantities instead of coded coefficients. Again using the motion-control example, this interface can express shaft speeds in rpm or torque limits in newton-meters. This layer of abstraction separates the application-control development from all of the details of managing the motor in real time.

Integrated design platforms, comprising the real-time engine, vendor-provided routines, hardware peripherals, and development environment, capture highly specialized hardware-engineering expertise. They also allow the OEM to build upon the resource by customizing or developing in-house functional and algorithm libraries. Most important, this approach reduces the OEM’s need to reinvent the technological wheel and instead promote a focus on those design areas that best differentiate the product.

AUTHOR’S BIOGRAPHY

Alex Lidow is chief executive officer of International Rectifier.



Neiman Marcus unveiled the Honeywell “kitchen computer,” which it based on a stock H316 minicomputer, in Neiman’s 1969 Christmas catalog. Linux kernel developer Val Henson re-created the original model’s catalog pose in this photo taken at the Computer History Museum in Mountain View, CA. Although the \$10,000 computer failed as a product because it was ill-conceived and far too expensive, it predicted the successful and pervasive penetration of processors and software into many home products (courtesy Val Henson, more at <http://infohost.nmt.edu/~val/kitchen.html>).



Intel introduced its 4004 microprocessor—a calculator chip originally designed for Busicom—with this ad in the November 15, 1971, issue of *Electronic News*. The lower right corner includes an invitation to readers to see the new CPU on a chip in the company’s hospitality suite at the Fall Joint Computer Conference in Las Vegas, held later that month. This ad, minus the invitation, ran in *EDN* the following month.

RCA Cosmac (CDP1802), launched into polar orbit 30 years ago on Sept 11, 1976, aboard the DMSP’s (Defense Meteorological Satellite Program) 5D-1 F1 satellite (Reference 7). A high-pressure nitrogen-supply-line leak caused the spacecraft to tumble. The leak depleted the satel-

lite’s thrusters and saturated its reaction-wheel attitude-control system. The satellite lost all attitude control. Uploading new on-orbit software allowed direct ground control of the satellite’s onboard magnetic torquing coils, which eventually stabilized the satellite and saved the

mission. A similar upload fixed another attitude problem on the next satellite in the series, the DMSP 5D-1 F2.

Spacecraft designers favored RCA’s Cosmac because it was available in radiation-hardened CMOS on SOS (silicon on sapphire). Many other spacecraft, including Viking, Galileo, the US space shuttle, and the two Voyager interplanetary probes, employed Cosmacs. When NASA launched them in 1977, the Voyagers each contained three Cosmacs, which have far outlived RCA Semiconductor. The two Voyagers have taken their microprocessors and software several billion miles from earth, beyond the solar system, and into interstellar space where they continue to gather and transmit data three decades after launch.^{EDN}

REFERENCES

- 1 Leibson, Steve, *Stanley P. Frankel: Unrecognized Genius*, www.hp9825.com/html/stan_frankel.html.
- 2 Aspray, William, “Masatoshi Shima, An Interview,” IEEE History Center, May 17, 1994.
- 3 Berlin, Leslie, *The Man Behind the Microchip: Robert Noyce and the Invention of Silicon Valley*, Oxford University Press, 2005.
- 4 Ritchie, Dennis, “The Development of the C Language,” <http://cm.bell-labs.com/cm/cs/who/dmr/chist.html>.
- 5 Kernighan, Brian W and Dennis M Ritchie, *The C Programming Language*, ISBN 0-13-110370-9 and 0-13-110362-8, Prentice Hall, 1978.
- 6 Leibson, Steve, *Designing SOCs with Configured Cores: Unleashing the Tensilica Xtensa and Diamond Cores*, Morgan Kaufmann, 2006.
- 7 Strom, Steven R and George Iwanaga, “Overview and History of the Defense Meteorological Satellite Program,” *Crosslink*, The Aerospace Corp, Winter 2005, www.aero.org/publications/crosslink/winter2005/02.html.

AUTHOR’S BIOGRAPHY

Steve Leibson spent the first 10 years of his career as an engineer, then spent 15 years as an award-winning journalist, publishing more than 200 articles and serving as editor in chief of both *EDN* (1993 to 1996) and *Microprocessor Report* (2000 to 2001). Now he is the technology evangelist for Tensilica.

HARDWARE MORPHS INTO SOFTWARE

By Jack Ganssle, The Ganssle Group

In the 1950s, the computer went from laboratory curiosity to a hugely expensive but useful business tool. The '60s saw the mainframe's role grow tremendously; meanwhile, minicomputers from Digital Equipment, Honeywell, Raytheon, Data General, and many others put computing into the reach of new kinds of customers who didn't possess the deep pockets that made IBM salesmen smile. But minis were still so expensive that they rarely replaced dedicated circuitry except in specialized applications. Even by 1970, DEC's PDP-8/E had a base price of \$6500, which is equivalent to approximately \$34,000 in today's dollars.

In 1972, *EDN* reported on the advent of the first microprocessor, Intel's famous 4004, which debuted at \$100 for just the chip itself (Reference A). Though a useful computer needed a tremendous amount of support circuitry, clever engineers realized that this new pricing node made it reasonable to dedicate processors to specific tasks. The embedded-system industry was born.

Better technology and the magic of volume manufacturing pushed prices even lower. By 1974, Texas Instruments' TMS1000, the first microcontroller, nearly delivered on the promise of a "computer on a chip." Two years later, Intel's 8048 truly shrunk a complete machine into a single IC—one whose legacy persists today in the 8051 family. In 5 billion years, when the sun grows dark, someone, somewhere, will be writing 8051 code.

Managers bought into the notion that software was free. Couple free—other than astronomical development and support costs—code with nearly free hardware, and it makes sense to substitute processors for dedicated circuits. Early embedded systems replaced industrial-automation equipment such as PLCs (programmable-logic controllers), which used big arrays of expensive cam timers and relays to control factory equipment. These mechanical contraptions were programmable; a technician could alter the position of cams to change timing—for instance, to select when to dispense liquid from a nozzle. Infinitely malleable software was a natural and cheaper substitute.

In the '60s, EEs often used analog circuits to perform calculations that were prohibitively expensive using logic. Some instruments, such as colorimeters, had to compute the classic "distance" equation: the square root of the sum of the squares of various parameters. It's not hard to put a transistor into the feedback loop of an op amp to cheaply compute square roots, but doing a

square root with TTL components is tough. Yet, these analog circuits had poor resolution and were subject to drift. The marketplace found that a software approach was more economical and stable and could output as much precision as an application demanded.

Instrumentation vendors found that they could generate better data by applying software-smoothing algorithms to perpetually noisy analog inputs. These algorithms had similar effects on outputs. Old-timers remember using grease pencils on the faces of oscilloscopes' CRTs to "log" waveforms. Though analog storage tubes offered some relief, today's smart scopes not only offer plenty of storage options, but also perform math on the data to figure frequency, rms amplitude, and more. Some run an FFT to rotate time data into the frequency domain.

Onboard software replaced ever-vigilant ground controllers so spacecraft operating many light-hours away could autonomously perform complex maneuvers.

Legislation influenced design. Automotive carburetors yielded to fuel-injection systems whose software monitors many inputs to meet emission standards. Energy Star appliances use smart code to reduce power consumption, and intelligent controllers at power plants limit coal consumption by calculating optimum burn ratios.

Software is no longer just a substitute for analog and digital circuits. Entire classes of features and products just couldn't exist without the complexity that code enables. Cell phones, iPods, digital cameras, and more would be as large as city blocks if designers implemented them in hardware.

Ironically, a new trend is now replacing software with hardware. FPGA vendors offer embedded processors whose instruction sets you can fine-tune to meet an application's needs. At least one vendor offers a tool that converts snippets of slow code into speedy FPGA hardware.

REFERENCE

A "One-chip CPU available for low-cost dedicated computers," *EDN*, Jan 15, 1972, pg CH 21.

AUTHOR'S BIOGRAPHY

Jack Ganssle has written more than 500 articles and four books about embedded systems, as well as a book about his sailing fiascos. He now gives seminars to companies worldwide about better ways to develop embedded systems. Contact him through www.ganssle.com.