Theme Feature

Seeking Solutions in Configurable Computing

Configurable computing offers the potential of producing powerful new computing systems. Will current research overcome the dearth of commercial applicability to make such systems a reality?

William H. Mangione-Smith
University of California, Los Angeles

Brad Hutchings
Brigham Young University

David Andrews
University of Arkansas

André DeHon
University of California, Berkeley

Carl Ebeling
University of Washington

Reiner Hartenstein
University of Kaiserslautern

Oskar Mencer
Stanford University

John Morris
University of Western Australia

Krishna Palem
New York University

Viktor K. Prasanna
University of Southern California

Henk A.E. Spaanenburg
Lockheed Sanders

Configurable computing systems combine programmable hardware with programmable processors to capitalize on the strengths of hardware and software. Often these systems must also address the difficulties of both hardware and software, because they mix the technology. While the origins of configurable computing go back at least 30 years, the past eight years have brought about a significant increase in research activity.

Since at least 1989,[1] configurable computing systems[2] have demonstrated the potential for achieving high performance for a range of applications, including image filtering, convolution, morphology, feature extraction, and object tracking. Researchers have developed prototype systems that achieve performance an order of magnitude higher than more conventional approaches for a number of applications. However, realizing this potential outside of the laboratory has proven difficult because these systems rely on manipulating low-level abstractions—digital circuits, for example—and thus require highly skilled developers.

CURRENT STATE OF AFFAIRS

The earliest configurable computing machine was likely proposed, designed, and implemented by Gerald Estrin at UCLA in the early 1960s.[3] Estrin proposed the "fixed plus variable structure computer," which dedicated hardware to both an (inflexible) abstraction of a programmable processor and a (flexible) component that implemented digital logic. This basic architecture, which supports programmed hardware and software, is at the core of all subsequent configurable computing systems. Unfortunately, Estrin's architectural concepts were well ahead of the enabling technology, and he was only able to prototype a crude approximation of his vision. Many of the concepts that are now being discovered by the configurable computing community lie quietly unheeded in Estrin's early publications.

The enabling technology behind the renewed interest in configurable computing is the availability of high-density VLSI devices that use programmable switches to implement flexible hardware architectures. These chips contain memory cells that hold both configuration information for the programmable switches and state information for active computations. Before programming, the chips present a partial architecture, which is then refined according to the configuration information. The configured device provides an execution environment for a specific application.

The most common devices used for configurable computing are *field programmable gate arrays*. FPGAs present the abstraction of gate arrays, allowing developers to manipulate flip-flops, small amounts of memory, and logic gates.

Figure 1 illustrates the basic architectural components of all configurable computers. This highly abstracted model allows a wide range of design choices, all of which revolve around three main decisions.

- *Granularity of programmable hardware.* Most existing configurable computers use commercial FPGAs. Consequently, application development involves the use of traditional CAD tools, which were developed for application-specific integrated circuits (ASICs). Many application developers find this low-level abstraction difficult to work with, and the systems achieve poor circuit density for highly regular structures such as multipliers. To raise the level of abstraction, several configurable computing systems under development limit the programmable hardware to the interconnect, and in the place of gates and flip-flops they use components such as arithmetic logic units (ALUs) or multipliers.
- *Proximity of the CPU to the programmable hardware.* First-generation systems typically used peripheral buses like the Sparc SBus to provide a coprocessor-like structure. Recently, some researchers have argued that the programmable hardware must be much closer to the processor, perhaps even on the datapath, fed by processor registers. This issue affects hardware design as well as application development.
- *Capacity.* Different system designers have made drastically different choices about fundamental questions of system capacity. What is the best ratio of programmable hardware to memory size

0018-9162/97/$10.00 © 1997 IEEE

and bandwidth? Or processor communication bandwidth? How much programmable hardware is required: an unlimited amount for applications with unbounded parallelism or only so much?

Granularity of programmable hardware

The configurable computing community is divided into two camps, according to the level of abstraction provided by the programmable hardware. The majority of current research efforts use commercial FPGAs and manipulate digital circuits through logic gates and flip-flops. We will refer to these devices as *netlist computers*. As part of conventional CAD development of ASICs, digital circuits are translated into netlists, which are composed of logic gates and flip-flops.

In the second camp are the newer architectures, which are based on "chunky" function units such as complete ALUs and multipliers. These architectures limit the programmable hardware to the interconnect among the function units, but implement those units in much less IC area.

Netlist computers. A typical netlist computing device is an FPGA containing thousands of low-powered processing elements. For example, an FPGA cell might consist of a single flip-flop and a function generator that implements a Boolean function of four variables. FPGAs have a programmable interconnect that is manipulated as individual wires. Because of their fine granularity, netlist computers are the most flexible configurable computers; their elements can be used to implement state machines, datapaths, and nearly any digital circuit. This flexibility is purchased with additional silicon, and it results in lowered performance on certain classes of problems, compared to chunky architectures. The two best-known netlist computers are Splash[4] and DECPeRLe-1.[1]

Conceptually, Splash consists of a linear array of processing elements. This topology makes Splash a good candidate for linear-systolic applications, which stress neighbor-to-neighbor communications. Because of limited routing resources, Splash has not proven as effective at implementing multichip applications that are not linear systolic, though some progress has been made. The DECPeRLe-1 is organized as a two-dimensional mesh and consists of a 4×4 array of FPGAs. Each FPGA has connections to its nearest neighbors as well as to a column bus and a row bus.

The designers of Splash and the DECPeRLe-1 constructed them as attached accelerators alongside workstations. Neither Splash nor DECPerRLe-1 provide general-purpose routing networks between FPGAs. Instead they require the designer to manually partition the circuit during the design phase, ensuring that the available interconnect is used as efficiently as possible.

The netlist computer presents a number of serious challenges to application development. The developers must be concerned with the size and usage of the FPGA devices, the size and usage of the memories, and finally the overall interconnection of all devices on the platform during all phases of the design process. The design process is therefore more difficult and time-consuming. Furthermore, modifications can require a significant amount of CAD compilation time.

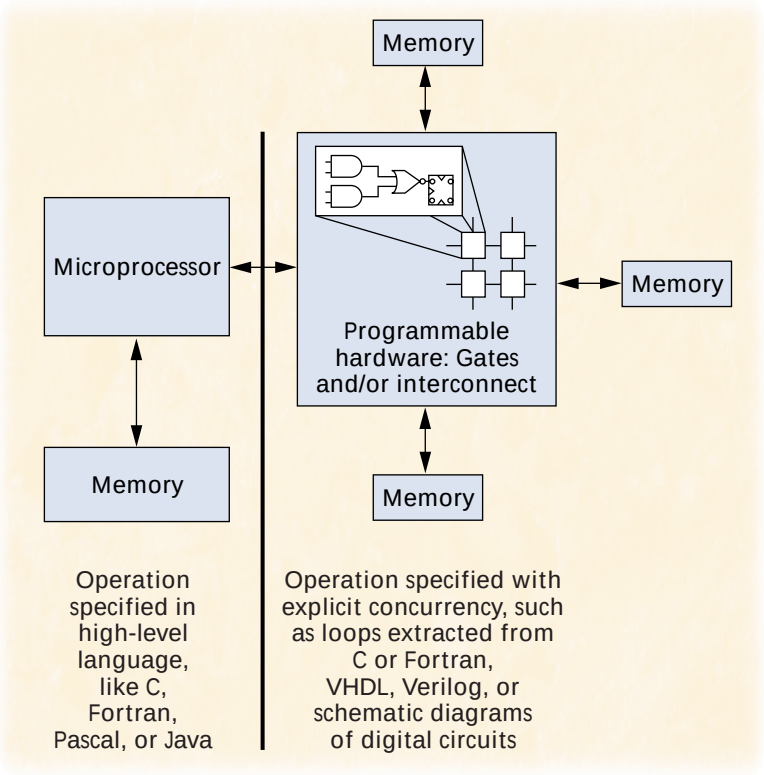

Figure 1. Architectural components of a configurable computer.

The challenge to the designers of netlist computers is to show that the increased flexibility presented by a low-level abstraction is essential to enable an important class of applications, thus compensating for the increased design difficulty. While a small number of netlist computing systems are available,[5] they have achieved little commercial success thus far.

Chunky function unit architectures. General-purpose processors (including digital signal processors) use optimized function units that operate in bit-parallel fashion on long data words. Compared with GPPs, FPGAs are inefficient for performing ordinary arithmetic and logic operations. Netlist computing has the advantage when it comes to nonstandard bit-oriented computations such as count-ones, find-first-one, or complicated masking and filtering.

At the same time, much of the research in configurable computing has focused on parallel DSP applications. Examples include image morphology, sensor beam forming, and object recognition. These tasks usually process sensor data that is 8-12 bits wide. Some filtering might also be necessary, which typically requires

The primary challenge for the proponents of chunky architectures is to identify the essential set of features for the function units, the datapath width, and interconnect structure.

multipliers. Although FPGAs are capable of implementing these regular hardware structures, they are generally poor targets even when handcrafted libraries are used. FPGAs do provide a distinct opportunity for hardware specialization. For example, it is now a standard practice to use customized hardware when multiplying by a value that is constant over a long period of time.

Chunky function unit architectures address this problem of poor hardware efficiency by mixing highly optimized parallel function units—the sort found in programmable processors—with a programmable interconnect. For example, a collection of multipliers might be available along with a crossbar interconnect to efficiently support a wide range of infinite-impulse response (IIR) filters. This basic approach has been pursued by a number of recent research projects, including rDPA at Kaiserslautern,[6] Rapid at the University of Washington,[7] and Matrix at MIT.[8]

These architectures each present an abstraction that is much higher than logic gates and flip-flops. Consequently, highly regular applications that map well to a specific implementation will likely achieve high performance with low silicon cost. Many other regular applications can probably be mapped onto the same implementation, either by breaking wide-word operations into a composition of the narrower, native hardware function units or by simply wasting the upper bits of the fixed datapath to handle narrow-word operations. However, highly irregular computations will likely be a poor match for these architectures.

The primary challenge for the proponents of chunky architectures is to identify the essential set of features for the function units, the datapath width, and the necessary interconnect structure. Each of these issues presents a compromise between functional density and speed on the one hand and applicability on the other. Thus far, results for chunky function unit architectures have been limited to design evaluation and performance estimates. The immediate challenge is to demonstrate performance on a set of interesting applications in the laboratory.

Ultimately, frustration in both industry and research centers on the lack of a single unifying model that combines the netlist and chunky approaches. While the models are united by the use of programmable hardware and processors for executing applications, there currently is no effective unifying architecture either within or between the camps. Although it would be intellectually satisfying to have a single effective taxonomy, it is unlikely that one will emerge until a number of configurable computing architectures deliver real, commercially relevant applications to the market.

CONFIGURABLE COMPUTING MODELS

One way to characterize the differences among how configurable computing might be applied is to consider the rate of configuration. We are not concerned here with the abstractions that are configured, but rather with the abstraction of the configuration. The essential characteristic of all the computing models is that some amount of state is semistatic; that is, it changes frequently enough to take advantage of programmability but slowly enough to mask the hardware configuration time.

Static configuration involves hardware changes at a relatively slow rate: hours, days, or weeks. This classification trivially subsumes the case of a bug fix. Static configuration has been used to accelerate a number of applications, including automatic target recognition.[2]

Time-sharing is another approach. If an application can be pipelined, it might be possible to implement each phase in sequence on the programmable hardware. The DISC project at Brigham Young University involved compiling C code fragments to assembly language for a custom FPGA-based processor as well as configuration components for an FPGA accelerator.[9] The FPGA caches configuration components and executes a demand-driven fetch and reload in response to a fault. Time-sharing also has been used to improve performance for a number of applications, including a video communications system and image recognition.[2]

The most ambitious form of configuration suggested thus far involves dynamic generation of FPGA circuits. This technique has been proposed for both evolutionary systems that exhibit emergent behavior as well as more traditional applications through parameterized macro libraries and dynamic placement. However, it has not yet been shown how broadly this approach can be applied.

The challenge to the configurable computing research community is to refine these computing models into reusable frameworks. Thus far, the abstractions generally exist in the developer's mind and are implemented through ad hoc mechanisms based on available technology and local experience. The community must develop a set of application-programming interfaces (APIs) to support common interactions with any programmable hardware. Examples include hardware configuration, controlling programmable clocks, and debugging. At a higher level of abstraction, efficient application development frameworks must be created that understand these common modes of configuration and allow the developer to quickly and clearly express the desired structure. Any effective framework would need to leverage state-of-the-art research from the compiler and CAD worlds.

CAD and compilation tools

The issues of development models and tools are inextricably linked. The underlying device hardware presents an architecture that the tools manipulate and refine. The tools then make this refined architecture

available to the system developer. Because the space between netlist and chunky architectures is large, current configurable computing systems leverage a wide range of development models. However, a number of consistent problems exist in all current tool sets, and so we address the problems here as a whole.

Tools for configurable computing systems must manage resources through time as well as space. Some of these same scheduling problems occur in VLSI CAD design systems, where independent hardware blocks are active concurrently and physical placement can have a dominant effect on performance. Configurable computing systems open up the opportunity to migrate functionality from one time segment to another, or possibly into multiple time segments in different physical locations. This opportunity presents a new challenge, as CAD software often introduces intolerable inefficiencies even when dealing with the subset of issues that arise during ASIC design.

FPGAs are the most commonly available component for configurable computing systems. This choice encourages the designer to use the existing FPGA development tools. Thus, applications are specified and designed by manipulating digital logic components, usually through schematic capture or synthesis. Unfortunately, digital circuits are not abstract enough to enable most programmers to effectively use the technology. At the same time, the highest level abstractions available—Verilog and VHSIC Hardware Description Language—often result in low performance. It is clear that a development model that manipulates higher level abstractions, and yet lends itself to highly efficient CAD tools, will be required before configurable computing becomes accessible to a large body of developers.

A few efforts have been made to produce a unified development environment for configurable computing systems.[10,11] Thus far, these efforts have focused on providing a single language that can be effectively mapped to either software or hardware. This structure could then be used in conjunction with an automatic partitioning tool that would guide the compilation process toward the most appropriate target for each code block. Furthermore, the model of computation is processor and ASIC, which happens to be implemented in reprogrammable logic. Few of the current experimental development systems incorporate any models of dynamic configuration.

One promising avenue of research derives from existing work on high-performance compilation for very long instruction word (VLIW) processor architectures.[12] VLIW architectures provide a number of parallel function units, which are often connected through flexible but restricted networks. Most existing compilers are parameterized to study the benefit of various function unit combinations and interconnect structures. This computing model is a degenerate case of the chunky architecture, in which a linear array of function units is supported by a rich network. It seems likely that this compiler technology base could be adapted to support chunky configurable computing architectures.

The challenge to tool developers for configurable computing systems is to produce high-performance development systems that capture the complete application specification (hardware and software) and allow the developer to express how the system adapts through time. This challenge is particularly daunting in that it requires leveraging existing tools for ASIC development, compilation, and hardware-software codesign along with capturing those aspects of application specification that are unique to configurable computing.

Benchmarking and metrics

Because configurable computing systems are developed to satisfy the demand for higher performance systems, there must be some objective method of benchmarking their success. Without objective performance measures, the entire community runs the risk of losing credibility. However, we do not hold out much hope that a meaningful broad-based benchmark suite will be developed for configurable computing systems, for a number of reasons.

First, high performance is typically achieved by fine-tuning and tailoring a circuit description to a specific application. Most of the time these circuits are used only once; circuits might even be optimized for a specific data set. It is unlikely that any general benchmark will apply to these circuits in a meaningful way. When implementations are highly optimized, basic system designs (such as netlists) are not even truly portable across different FPGA architectures. Performance measures often evaluate the skill of the designers rather than the system architecture or the technical approach.

Second, configurable computing systems try to overlap I/O and computation to increase throughput. It is difficult to incorporate I/O requirements in a broad benchmark. Furthermore, I/O characteristics can change dramatically based on the available hardware ports. This factor can complicate the task of performing a deep and thorough performance evaluation.

We are not suggesting that it is impossible to measure performance and compare results. On the contrary, application-specific benchmarks are an effective means of evaluation, and ultimately they are the only meaningful way to compare systems.

Of course, application-specific benchmarks are no panacea. They tell us only what is possible and do not always shed light on how the performance was achieved; this tends to be true of all benchmarks. However, in these early stages as researchers experiment with a variety of implementation techniques and architectures, application-specific benchmarks can be

Because configurable computing systems are developed to satisfy the demand for higher performance systems, there must be some objective method of benchmarking their success.

It is not surprising that many of the most pressing needs for configurable computing systems can be lumped into the category of FPGA-related technology.

useful both as a means of comparison (for the same problem) and as a demonstration of feasibility.

The benchmarking challenge to the configurable computing community is straightforward: to develop a benchmark that is useful for understanding and evaluating the performance of a range of configurable computing systems. The path to this goal is unclear.

Commercial development

Despite published research reports, no companies are known to use configurable computing for a competitive advantage. (FPGA vendors have suggested in public forums that some current customers do use configurable computing techniques but consider these practices trade secrets. It seems unlikely that the number of such systems could be large.) A number of vendors have produced systems to enable application development,[5,13] though these do not constitute applications themselves. While FPGAs have become a common component in a wide range of commercial products, the vast majority of these uses are for replacing gate arrays to reduce development cost and time.

Even though there are no shipping products, a number of companies have announced configurable computing systems. In particular, Metalithic is pursuing digital recording, and Sundance is developing a system for image processing through automatic hardware/software codesign. The earliest commercial successes will likely involve signal processing, particularly image processing. Many important image-processing applications are well matched to existing FPGA architectures, and a number of research papers have reported impressive performance achievements.

Nonetheless, the lack of any visible market use for configurable computing continues to cast a suspicious shadow across the entire field.

LIMITATIONS OF CONFIGURABLE COMPUTING

It is not surprising that many of the most pressing needs for configurable computing systems can be lumped into the category of FPGA-related technology, given the fact that so many configurable computing systems are based on FPGAs. A number of issues are being addressed by commercial vendors, but we feel these currently limit configurable computing systems.

- *Equivalent gate capacity.* By most measures, available FPGAs have generally provided the equivalent of 10K to 50K gates. These devices are often large enough to experiment with the basic strategies for configuration, but have limited the scope of the designs and forced application developers to turn to pared-down systems. The source of these capacity limits is the use of conservative semiconductor processes, particularly with limited metal layers. Recent devices have moved to leading-edge processes technology, resulting in FPGAs with over 100K equivalent gates. The problem of gate density will likely decrease in the future as on-chip delays drive designers toward partitioned designs that can be implemented efficiently on multiple chips.
- *Configuration speed.* Most existing FPGAs use relatively slow serial-shift paths for device configuration, even when a parallel interface is presented through the I/O pins. The configuration time is important for many, but not all, models of computation. In particular, those using fast design swapping might end up limited by reconfiguration time. The industry is moving toward parts that reconfigure faster, partly in response to these needs as well as customer desires for shorter testing cycles.
- *Memory structures and interface.* Most FPGAs have no external memory interface that can be accessed from the active circuit, which forces system designers to sacrifice some programmable resources to build an application-specific memory interface. A similar problem occurs on a chip, where blocks of RAM tend to be scarce and hard to access. A more efficient approach would be for FPGA vendors to implement standard memory interfaces on their devices in dedicated hardware, which can be optimized to the problem.

Configurable computing is a rich area of active research. Unfortunately, no system to date has yet proven attractive or competitive enough to establish a commercial presence. We believe that ample opportunity exists for work in a broad range of areas. In particular, the configurable computing community should focus on refining the emerging architectures, producing more effective software/hardware APIs, better tools for application development that incorporate the models of hardware reconfiguration, and effective benchmarking strategies. ❖

References

1. P. Bertin, D. Roncin, and J. Vuillemin, "Introduction to Programmable Active Memories," in *Systolic Array Processors*, J. McCanny, J. McWhirther, and E. Swartslander, eds., Prentice Hall, Englewood Cliffs, N.J., 1989, pp. 300-309.
2. J. Villasenor and W.H. Mangione-Smith, "Configurable Computing," *Scientific American*, June 1997, pp. 66-71.
3. G. Estrin et al., "Parallel Processing in a Restructurable Computer System," *IEEE Trans. Electronic Computers*, Dec. 1963, pp. 747-755.
4. M. Gokhale et al., "Building and Using a Highly Parallel Programmable Logic Array," *Computer*, Jan. 1991, pp. 81-89.

5. J. McHenry, "The WILDFIRE Custom Configurable Computer," *Proc. Int'l Soc. Optical Engineering: Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing,* J. Schewel, ed., Philadelphia, 1995, pp. 189-199.
6. R. Hartenstein and R. Kress, "A Datapath Synthesis System for the Reconfigurable Datapath Architecture," *Proc. Asia and South Pacific Design Automation Conference '95,* 1995, pp. 479-484.
7. C. Ebeling, D.C. Cronquist, and P. Franklin, "Rapid—Reconfigurable Pipelined Datapath," *Proc. Field Programmable Logic,* Springer-Verlag, Heidelberg, 1996, pp. 126-135.
8. E. Mirsky and A. DeHon, "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources," *Proc. FPGAs Custom Computing Machines,* IEEE CS Press, Los Alamitos, Calif., 1996, pp. 157-166.
9. M.J. Wirthlin and B.L. Hutchings, "A Dynamic Instruction Set Computer," *Proc. FPGAs for Custom Computing Machines,* IEEE CS Press, Los Alamitos, Calif., 1995, pp. 99-107.
10. S. Guo and W. Luk, "Compiling Ruby into FPGAs," *Field-Programmable Logic and Applications,* W. Moore and W. Luk, eds., Springer, Oxford, UK, 1995, pp. 188-197.
11. W. Luk, N. Shirazi, and P. Cheung, "Compilation Tools for Run-Time Reconfigurable Designs," *Proc. FPGAs for Custom Computing Machines,* IEEE CS Press, Los Alamitos, Calif., 1997, pp. 56-65.
12. J.C. Dehnert and R.A. Towel, "Compiling for the Cydra 5," *J. Supercomputing,* 1993, pp. 181-229.
13. S. Casselman, "Virtual Computing and the Virtual Computer," *Proc. FPGAs for Custom Computing Machines,* IEEE CS Press, Los Alamitos, Calif., 1993, pp. 43-48.

William H. Mangione-Smith is an assistant professor in the Electrical Engineering Department at the University of California, Los Angeles. He received a BSE in electrical engineering, and an MSE and a PhD in computer science and engineering, all from the University of Michigan, Ann Arbor.

Brad Hutchings is an associate professor of electrical and computer engineering at Brigham Young University. He received a PhD in computer science from the University of Utah.

David Andrews is an associate professor of computer systems engineering at the University of Arkansas. He received a BSEE and an MSEE from the University of Missouri, Columbia, and a PhD from Syracuse University. He is a member of the IEEE Computer Society.

André DeHon is a visiting postdoctoral research engineer at the University of California, Berkeley. He received an SB, SM, and a PhD from the Massachusetts Institute of Technology. He is a member of the ACM, the IEEE Computer Society, Eta Kappa Nu, Tau Beta Pi, and Sigma Chi.

Carl Ebeling is a professor of computer science and engineering at the University of Washington. He received a PhD from Carnegie Mellon University. Ebeling received the NSF Presidential Investigator Award and served as program chair for the Conference on Advanced Research in VLSI and the International Symposium on FPGAs.

Reiner Hartenstein is a professor of computer engineering at the University of Kaiserslautern. He founded the PATMOS workshop series on low-powered VLSI and cofounded the Euromicro and the German E.I.S.-Project on VLSI design. Hartenstein is a member of the GI, ITG, and GME German professional societies, the ACM, and is a senior member of the IEEE.

Oskar Mencer is a PhD candidate in electrical engineering at Stanford University. Mencer received a BSc in computer engineering from Technion, Israel Institute of Technology, and an MS degree in electrical engineering from Stanford University. He is a student member of the IEEE Computer Society and ACM.

Krishna Palem is an associate professor of computer science in the Courant Institute of Mathematical Sciences at New York University. Palem cofounded PARCON—The Symposium on New Directions in Parallel and Concurrent Computing.

Viktor K. Prasanna (V.K. Prasanna Kumar) is a professor of electrical engineering systems at the University of Southern California and serves as director of the school's Computer Engineering Division. Prasanna received a BS in electronics engineering from Bangalore University, an MS from the School of Automation at the Indian Institute of Science, and a PhD in computer science from Pennsylvania State University.

Henk A.E. Spaanenburg is the director of software and computing architectures technology at Sanders, a Lockheed Martin company in Nashua, New Hampshire. He also heads the Software and Computing Architectures Technology Directorate at the Signal Processing Center of Excellence in Sanders' Advanced Technology Division. He received an MSEE from Delft University of Technology in the Netherlands and a PhD in electrical and computer engineering from Syracuse University.

Contact Mangione-Smith at billms@ucla.edu.