30

Design technologies for embedded multimedia systems*

P.A.Subrahmanyam and Bryan D. Ackland
Bell Laboratories Research, Lucent Technologies
Room 4E-530, 101 Crawfords Corner Road, Holmdel, New Jersey, U.S.A. 07733
908-949-5812 (Phone); 908-949-9118 (FAX) e-mail: subra@research.bell-labs.com

Abstract. *A broad range of new wireless & multimedia products and services, such as set-top boxes and personal communicators, has been enabled by a melding of computing, communication, entertainment, and VLSI technologies. Unfortunately, the design effort needed to bring such products into the marketplace within a 6-9 month market window is currently a major bottleneck. This is because the design technology is growing at a slower pace than the underlying VLSI capabilities and system complexity. This paper summarizes the major tasks that need to be addressed when designing embedded multimedia systems, and highlights the challenges that need to be met. Some of the emerging design technologies that address these challenges are outlined, including hardware-software codesign methodologies, coverification and cosimulation frameworks. These improvements can better leverage the underlying technology to yield improved and more cost effective products for the consumer at home, in the office, or on the information superhighway.*

Keywords. Real-time, set-top boxes, VLSI, Hardware-software codesign, multiprocessors.

1. Introduction

A broad range of new multimedia/wireless products and services has been enabled by the fortuitous confluence of progress in four key areas. These are: (1) algorithms and standards in audio, video and graphics, e.g., for video conferencing (H.261, H.263), and digital video (MPEG 1, MPEG 2, MPEG 4); (2) semiconductor device and fabrication technology for high-speed, low cost CMOS VLSI digital and analog devices, including CMOS cameras; (3) increased capabilities of "end-user" computing devices such as personal computers, workstations, and personal digital assistants (PDAs); and (4) fast (10-100 Mb/s) local and wide area communication networks. Examples of such products include hand-held personal communicators, interactive video games using high quality graphics, and set-top boxes of varying sophistication, to name just a few. The underlying technologies include compressed digital audio and video, high quality 3-D graphics, speech and handwriting recognition, speech synthesis, wireless communication, and high speed modems. The spectrum of possibilities is further enriched when on chip cameras and display technologies can be cost-effectively integrated into this regime.

---

* Invited paper.

N. Terashima et al. (eds.), *Advanced IT Tools*

A significant —and rapidly increasing— number of these products and services have extremely high computational and communication requirements on the one hand, and low cost constraints on the other hand. In order to meet these dual requirements, such products increasingly exploit the notion of "system on a chip," where the increased levels of integration often yield significant cost savings. In 1996, many systems contain single chips with 5-10 million transistors, while single chips with 125 million transistors (including 50% memory) are projected within the next two years. A large component of the cost of delivering such systems to the marketplace tends to be the non-recurring design cost. As the life cycle of these products decreases, this design cost becomes an increasing, if not dominant, component of the product cost. Unfortunately, the technology for designing such systems is growing at a much slower pace than the underlying VLSI capabilities and system complexity. When design methodologies and tools are improved, then the processing technology can be better leveraged to yield more timely and cost efficient products for the customer. It also directly affects profitability, since the inability to inject a consumer electronics product into the marketplace during the right time window (6-9 months) negatively impacts profit margins.

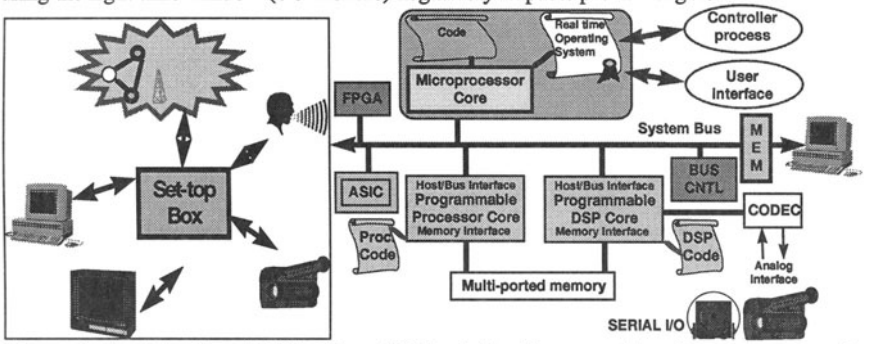

Figure 1: A set-top box (a) Interfaces (b) Typical software and hardware components

2. Ingredients of embedded multimedia systems

We will consider, as a representative example of an embedded multimedia system, a set-top box (Figure 1a). This is a box that is intended to complement a TV at home and provide a wide array of services. These can range from simple video-on-demand to more sophisticated services, e.g., video conferencing and interactive networked games. Such a set top box (STB) needs to interface with a backbone network. This entails the need for appropriate hardware interfaces for networks, e.g., cable, wireless, and ATM, as well as software interfaces for the appropriate network protocols, e.g., TCP/IP. The STB must be able to communicate with various I/O devices, such as audio microphones and speakers, video cameras and displays, game controllers, infrared remotes, serial I/O channels, etc. Such communication also needs the appropriate hardware interfaces and software drivers for these devices. In addition, the STB needs to support different kinds of processing require for "edutainment" (education and entertainment) when a consumer is watching movies, playing video games, or surfing the Internet. This entails the need for potentially sophisticated audio, video, and graphics processing capabilities in such a system. High end set-top boxes also need to interface with a PC, and interact with its host bus and memory subsystem.

Let's now take a peek and see what can lie under the hood of such a high-end set-top box (Figure 1b). While there can be substantial variation in the functionality and architecture details of any such system, the important common attribute to note is the considerable diversity in both the hardware and software ingredients.

The hardware components might include, for example, codecs for analog interfaces to devices such as cameras and displays, as well as cable and wireless modems, transceivers and ATM interfaces. The internal system bus must interface to memory and an external PC bus, e.g., a standard bus such as the PCI bus. Programmable computations are supported by one or more of processor cores, e.g., a standard microprocessor core, a DSP core, or perhaps an application specific processor core. The memory subsystem can consist of RAM, SRAM, DRAM, multi-bank DRAMs, and the like. Other modules include DMA controllers, and memory & bus interfaces, & I/O interfaces. These hardware components can be implemented using Field Programmable Gate Arrays (FPGAs), standard cells library components (ASICs), library cores, or custom-crafted circuits.

The software components can form an equally diverse mix. Examples include: software running on a microprocessor core, and written in a "high level language" e.g., C or C++; cooperative multi-tasking software running on one or more Digital Signal Processors (DSPs), typically written in assembly code; a real-time Operating system kernel running on the microprocessor core and/or DSP core; control and scheduler processes running on a micro-controller or processor; user interface modules running on the microprocessor; device drivers and interface protocols, etc. Such a mix results in a complex software architecture, which is increasingly becoming a critical bottleneck.

3. Computational requirements of multimedia applications

Multimedia signal processing places extreme demands on the underlying computing platform, with real-time video compression being one of the more demanding tasks. These requirements include: high throughput, on the order of hundreds to thousands of MIPS (millions of instructions per second); high input-output and memory bandwidth, on the order of tens to hundreds of megabytes per second (MB/s); low latency (on the order of milliseconds); real-time performance constraints; predictable throughput and delay; flexibility in the architecture to be able to adapt to rapidly evolving markets; and low cost (on the order of $100 or less for consumer electronic products).

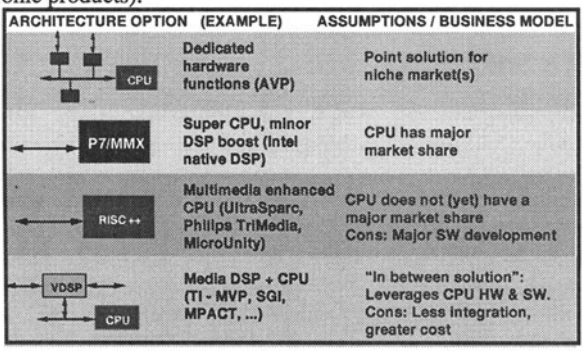

Figure 2: Emerging processor architectures for multimedia computing

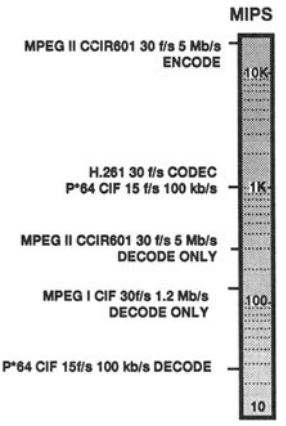

Figure 3 : Computational power in MIPS required to process compressed video

Some representative numbers for real time video compression are shown in Figure 3. For example, real time MPEG 2 encoding requires over 10 billion operations per second. Among the more compute intensive operations involved in video encoding are the computation of Discrete Cosine Transform (DCT) and motion estimation. These operations are typically performed on 8x8 pixel blocks that comprise a video frame. For a video conferencing application based on an image size of 352x288 pixels (CIF image) at 30 frames per second, the DCT computation requires 73 million multiply-adds per second. For an HDTV decoder, applying DCT over a 1280x720 image at 60 frames per second requires over 1.4 billion multiply-adds per second.

Motion estimation is used to achieve better compression by using the correlation between 16x16 macroblocks in two consecutive video frames. The motion estimation operation requires computing a figure of merit that measures the match (or lack thereof) between a macroblock in one frame, and a displaced macroblock in the previous frame; typically, this figure of merit is the sum of the absolute differences of the pixel values in the two macroblocks. Motion estimation using a search range of 32 pixels on a (352x288) CIF image at 30 frames/second requires over 12.5 billion compare-adds per second, albeit at reduced precision, e.g., 8-16 bits. A more detailed review of video coding techniques can be found in [Aravind et al. 93].

4. Emerging multimedia processor & system architectures

The processing power needed to enable real-time execution of high-end multimedia applications is beyond the capability of conventional general purpose processors. Consequently, additional hardware support is needed for some of the key operations. These include: very high performance integer vector processing; hardware assisted multiply accumulate operations, low precision arithmetic (8, 16 and 32-bit arithmetic), general purpose (non-DSP) control tasks, multi-threaded real-time applications, as well as flexible integer and floating point processing support for audio and graphics processing.

To address these requirements, several new processor architectures are emerging. Some examples of these, and their pros and cons are depicted in
Figure 2, along with comments on the associated business model. Key attributes of a few of these architectures are summarized in Table 1. All of these processors include some form of support for SIMD (vector) instructions that operate 8 and 16-bit numbers. The Microunity Media Processor [Hansen 96] accommodates other bit widths as well, and offers considerable support for manipulating sequences of bits within 64 and 128 bit words.

Table 1: Some features of emerging multimedia processor architectures

	TI MVP	Chromatic MPACT	Philips Trimedia	IBM MFAST	Intel P55C
Architectural features	4 x 64 bit DSPs + 32bit RISC + cross bar interconnect	VLIW/SIMD 4 ALUs ME engine 792bit bus	VLIW 25 functional execution units +VLD engine	VLIW 4x4 folded array, SIMD columns, 32b mesh	Micro-processor with "multimedia" (low precision vector) instructions
Clock rate	40 MHz	62 MHz	100 MHz	50 MHz	200 MHz
Peak Performance	1.2 Gops	2 Gops	4 Gops	20 Gops	800 Mops
Memory type/ Bandwidth	DRAM 400 MB/s	RAMBUS 500 MB/s	SDRAM 400 MB/s	SDRAM 800 MB/s	DRAM 100 MB/s
Volume Prod	1995	2Q96	4Q96	2Q97	4Q96

5. Designing multimedia systems: Challenges & emerging technologies

5.1 Designing embedded multimedia systems

The major sub-tasks in designing an embedded system, such as a set-top box, involve:

- *Modeling* the system to be designed, and experimenting with the algorithms needed;
- *Partitioning* the function to be implemented into smaller, interacting *modules*;
- *Allocating* the elements in the partition to execute on hardware units, or software running on custom hardware or a general microprocessor;
- *Scheduling* the times at which the functions are executed. This is important multiple modules in the partition share a single hardware unit;
- *Implementing* a functional description in either (1) software that runs on a processor or (2) a collection of custom, semi-custom, or commodity hardware parts.

Modeling, analysis & simulation.

The components of a design need to be modeled, analyzed and simulated, often at multiple levels of abstraction. Examples include system level modeling (to experiment with algorithms and behavioral models), and implementation level modeling (using register-transfer level VHDL descriptions and C/machine code for software). The prevalent approach is to write models in C/C++ at the system level, and VHDL or Verilog at the behavioral level and below. Digital Signal Processing is one of the few application domains where mature tools such as MATLAB exist that enable experiments with variations in algorithms and bit-precisions of the data types used.

Once the system components are assembled together, s*imulation* is commonly used for validating behavior and performance. Some mixed-level cosimulation frameworks allow varying levels of abstraction in the design to be mixed. Design space exploration at the system level not currently supported by any tools, and relies on insights gained from prior experience with similar designs, back of the envelope calculations, and informal (non-quantitative) arguments. System level design "analysis" is usually done by simulating the system model written in C/C++. No formal analysis is feasible because C/C++ programs are not amenable to any such analysis.

Emerging codesign techniques employ formally based computational models and specification languages that support the high level specification of heterogeneous systems, and that are more amenable to formal analysis and synthesis. This includes codesign finite state machines [Chiodo 94]; and models that combine event-based models with data flow based models for multimedia domains [Subrahmanyam 94]. It is important to note that formal analysis techniques are best used to complement simulation when appropriate, rather than replace it. For instance, subjective measures of system performance, such as "good picture quality" cannot be formally expressed and analyzed.

Multimedia applications require enormous amounts of data to be simulated. As a consequence, emulation platforms are often used for speeding up system level simulation. Representative speeds of some MPEG-2 encoder simulations at different levels of abstraction, and on different hardware platforms (logic accelerators and hardware emulators) are shown in Table 2. The main challenge here is to reduce the time and effort it takes to set up an In Circuit Emulation (ICE) platform, and the limited capacity of the emulation engines (measured by the number of usable gates). The higher end of current emulator capacity is on the order of 2-6 million available gates.

Table 2: Relative speed (slowdown) of encoder simulation

Platform/Tool	Time (relative)
Chip (100 MHz)	1
Hardware Emulator using FPGAs (1 MHz)	10^2
Software emulator	10^3-10^4
Hardware simulator (gate level)	10^5-10^7
Software simulator (RTL)	10^6-10^8
Software simulator (gate level)	10^7-10^9

Following system simulation, the hardware and software components have to be implemented using partitioning, scheduling and synthesis tools. Software is a major and growing component of the cost of embedded systems, and often a bottleneck in the design process. We elaborate on this in Section 5.3.

5.2 Core-based design: reusable hardware and software modules

One of the challenges when designing consumer electronics products is being able to rapidly engineer a *range* of products, each targeted to address a different cost-performance point. If the basic primitives are at the level of transistors (for hardware) and machine instructions (for

software), then obtaining an optimal design to meet a given set of functional and performance constraints involves searching a very large design space, and is practically impossible.

An effective design technique that reduces the overall design space is to build a system using a domain-specific set of reusable hardware and software modules at a much higher level of abstraction than transistors and machine instructions. This is often feasible because each of the primitive building blocks is domain specific, and can be optimized individually. Examples of modules typically implemented in hardware include parameterized processor cores, floating point units, vector coprocessors, etc. Examples of the functionality of domain specific modules used in video encoding include: motion estimation, DCT, variable length encoding (VLE) and decoding (VLD), filtering, etc. Such modules can be mapped into either software or hardware implementations. Differentiating features in a product line are often introduced via software rather than hardware changes, although different hardware configurations are also employed towards this end.

Although the use of reusable hardware and software cores reduces the overall design space, the range of architectural alternatives is still quite large, and automated techniques and tools can assist in its exploration. This remains an open challenge; [Subrahmanyam & Kalavade 95] discuss a focused strategy for the codesign of multithreaded applications.

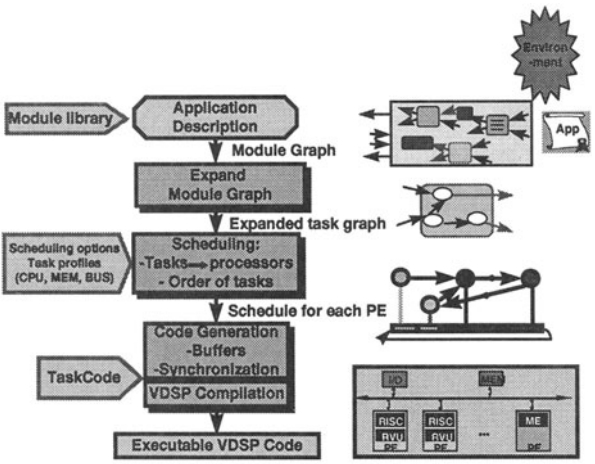

Figure 4: Application development: Tools support partitioning, allocation & scheduling

5.3 Software Architecture: Real-time Distributed/Parallel processing

In the remainder of this section, we will focus on the design of a software architecture that provides an infrastructure for implementing complex applications on a heterogeneous multiprocessor targeted at media processing applications. This supports ease of programming, and enables the application to extract and exploit MIMD & SIMD parallelism. The scheduling support partitions resources among multiple applications efficiently, while low-level software communication primitives that exploit the hardware features enable high-level programmability without compromising performance. The software architecture supports the development and execution of a potentially diverse mix of applications. This includes:

- continuous media applications that have real-time constraints, e.g., video conferencing and video on demand;
- interactive applications that need fast response, e.g., interactive video games and infrared mouse clicks;
- background applications, e.g., surfing the world wide web on the Internet.

We next summarize three aspects of such a software architecture: the high-level methodology for developing applications, the associated programming model, and the underlying operating system and run-time support. The overall software environment for such a system needs to include other elements, such as support for debugging concurrent real-time systems.

The goals of the SW architecture are to support 2 slightly different categories of users. The first is to support application programmers in developing complex applications, e.g., video conferencing or Video-on-demand, abstracted from architectural details. The second is to support DSP library module developers in developing fine-tuned reusable domain-specific building blocks, e.g., DCT & motion estimation that exploit the HW architecture. The role of the SW architecture is provide a bridge between the abstractions that are relevant to an application and module writers and the underlying HW resources of the machine, such as the CPUs, memory, bus, and various I/O ports.

Application development paradigm.
The paradigm for application development is shown in Figure 4. An *application* description constructed using either library or user developed modules. Examples of modules are motion estimation and DCT at the level of a video frame. This yields what is termed *a module graph.* The computation in a module can be optionally broken up into smaller units called *tasks.* An example of a task is the computation of DCT on a block of 8x8 pixels. For the purpose of making efficient use of the hardware resources, the module graph is automatically elaborated to yield an *expanded task graph,* where tasks represent an atomic unit of computation that is scheduled for execution.

Both an application and a module are associated with certain parameters that are used in generating a schedule. Application level parameters include: constraints on deadlines, the number and nature of processing elements that are used; event characteristics, e.g., periodicity of arrival of input frames; and constraints on responses to events, e.g., deadlines, jitter tolerance, etc. Examples of parameters associated with modules include: a measure of the total computation that is performed by the module, the number of processing elements that the computation is farmed out to (in any specific instantiation of the module); and the grain size of the computation unit (task) that is dispatched.

Once the expanded task graph is generated from the module graph, it is scheduled onto the chosen set of heterogeneous processing elements. The resulting is used to generate a code skeleton that is then compiled and linked with the code for the individual tasks. The resulting code can be run on the hardware target. The system software architecture is designed so that it allows the user to specify details pertaining to partitioning, allocation, and schedules as desired; default options are supplied if no specification is provided.

Programming Model

We adopt a multi-threaded model for programming, An application is specified as a collection of interacting processes, each of which forms an independent thread of execution. The intent is that one thread executes on a single PE. The basic synchronization mechanism uses semaphores that are implemented in hardware.

There are two reasons for choosing such a multithreaded programming model. First, from a pragmatic viewpoint, it allows us to leverage both existing sequential code and compiler technology at the level of computations executing on a single processor. Second, it accommodates 3 levels of parallelism: coarse grain parallelism at the level of multiple applications and multiple modules executing at any one time on the hardware; medium grain parallelism at the level of multiple processes and tasks executing on the different PEs; and fine grain parallelism at the level of a single processing element, e.g., multiple-issue processors, vector coprocessors, VLIW processors, etc. The coarse grain parallelism is either explicit in the set of applications, or specified by the user; the medium grained parallelism is automatically derived by computer-aided software design tools, or can optionally be explicitly specified by an application programmer; the low level parallelism is often supported by the hardware, although it may on occasion be programmed using assembly code or microcode.

The rationale here is that fine grain parallelism has typically been tedious and error prone for humans to exploit, and easier for machines implement. Conversely, it has been quite difficult for a program to automatically identify medium & coarse grain parallelism that is often apparent to a human familiar with the application domain. This approach leaves open the possibility of incorporating, if & when it becomes mature, the appropriate compiler technology: both for coprocessors, media signal processors, as well as for multiprocessors.

6. Summary

Embedded multimedia systems are important from a business standpoint and interesting from a research standpoint. They offer enormous opportunities to innovate new products and services for the consumer, both at home and in the office. We reviewed the existing design processes, their competencies, and drawbacks. We have argued that an improvement in some aspects of current design methodologies can help leverage the underlying software and VLSI technologies even more fruitfully, and enable further efficiencies and cost reduction. Some of the key challenges include: improving the speed, cost, and ease-of-use of emulation platforms; providing efficient compilers for media and digital signal processors (and, more generally, application specific instruction set processors); enabling the generation of customized application specific operating system kernels; and improving the system-level design methodologies and tools. In our experience, design methodologies that employ *tightly coupled, domain specific* reusable software and hardware modules have proven very useful as a basis for designing such systems.

Acknowledgements. We would like to acknowledge the members of the MSP/VDSP team for their contributions and insights, and Dr. Altman for his patience with this manuscript.

References

[Ackland et. al. 93] Ackland, B. D., et al., "A video codec chip set for multimedia applications", *AT&T Technical Journal*, Vol 72, No. 1, pp. 50-66, Jan 1993.

[Aravind et. al. 93] Aravind R., et. al., "Image and Video Coding Standards", *AT&T Technical Journal*, Vol 72, No. 1, pp. 67-89, Jan 1993.
[Chiodo et al. 94] Chiodo, M. et al., A formal methodology for hardware/software codesign of embedded systems. *IEEE Micro,* August 94.
[Endo 96] Endo, K. "EDA for MPEG-2 System Design", Proc. APCHDL , pp. 45-53, Jan 96.
[Guttag 93] Guttag K., "The multiprocessor video processor, MVP", *Proc. IEEE Hot Chips V,* Stanford CA., August 1993.
[Foley 96] Foley, P., "The Mpact Media Processor Redefines the multimedia PC", pp. 311-318, Proc. IEEE Compcon, Feb. 96.
[Rathnam & Slavenburg 96] Rathnam, S., and G. Slavenburg, "An architectural overview of the Programmable Multimedia Processor, TM-1", pp. 319-326, Proc. IEEE Compcon, Feb. 96.
[Hansen 96] Hansen, C., "Architecture of a Broadband MediaProcessor", pp.334-340, Proc. IEEE Compcon, Feb. 96.
[Goldman & Tirumalai 96] Goldman, G. and P. Tirumalai, "UltraSPARC-II: The advancement of UltraComputing", pp.417-423, Proc. IEEE Compcon, Feb. 96.
[Subrahmanyam et al. 95] Subrahmanyam, P.A., et al. "Software Architecture of a Video Signal Processor", Bell Laboratories, November 95.
[Subrahmanyam 94] Subrahmanyam, P.A., "Combining event-based and data flow models for multimedia applications", Internal memorandum, Bell Labs, November 94.
[Subrahmanyam & Kalavade 95] Subrahmanyam,P.A., and A. Kalavade, "Hardware-Software Codesign of Multithreaded Applications", Bell Laboratories, Internal Memo-randum, Nov 95.
[Sackinger, Muller & Subrahmanyam 95] Sackinger, E., U. Muller & P.A.Subrahmanyam, "A Parallel Processor Chip for Image Processing and Neural Networks", *Proc. Intl. Conf. on Applications of Neural Networks*, Paris 95.
[Wolf 94] Wolf, W.H., "Hardware-Software Codesign of Embedded Systems'", *Proceedings of the IEEE*, 82(7), pp. 967-991, July 1994.

BIOGRAPHY.

P.A.Subrahmanyam is with the VLSI Systems Research Department in the Wireless & Multimedia Communications Systems Research Center at Bell Laboratories. His current research interests include: multimedia systems and applications, formal methods in system design, CAD tools for VLSI design and verification, hardware-software codesign, parallel architectures and programming environments. He is the author of numerous journal and conference papers in these and other areas, and the editor/co-editor of over 4 books related to Formal Aspects of VLSI design and multimedia. Dr. Subrahmanyam is a founding Editor-in-Chief of the International Journal on Formal Methods in System Design, a member of IFIP Working Group 10.5/10.2 on VLSI/System Specification and Design, a member of the IEEE Design/Multimedia Segment Committee, and the Chairperson of the IEEE Technical Committee on VLSI.

Bryan D. Ackland received the BSc in Physics from Flinders University in 1972, and the BE and PhD EE from the University of Adelaide, Australia, in 1975 and 1979. In 1979 he joined AT&T Bell Labs, and is currently Head of the VLSI Systems Research Department at Bell Laboratories. His interests included raster graphics, symbolic layout and verification tools for full-custom VLSI, MOS timing simulation, and VLSI layout synthesis. His interests now are VLSI architectures and design tools for high-performance signal processing and communications, particularly multimedia. He is a Fellow of the IEEE.