

Incorporating Cost Modeling in Embedded-System Design

JAMES A. DEBARDELABEN
VIJAY K. MADISETTI
 Georgia Institute of
 Technology
 VP Technologies
ANTHONY J. GADIENT
 South Carolina Research
 Authority

System engineering constraints such as cost, time to market, and schedule are beginning to dominate product design. Incorporating cost modeling into the system-level design process yields a remarkable improvement in cost and quality of the manufactured product.

THE EMERGENCE OF MULTIMEDIA and wireless applications as growth leaders has created an increased demand for embedded systems. Examples of such applications range from digital cellular telephones to high-performance avionics systems. Along with the increased market share of these products, however, has come an increase in system complexity and cost. For example, the complexity of embedded avionics systems has quickly grown from millions to billions of operations per second. These systems, formerly implemented in hardwired or uniprocessor architectures, now must consist of programmable-multiprocessor arrays to meet performance requirements.

Time-to-market and life-cycle costs are key factors in the success of these products in the competitive electronics marketplace. These costs, therefore, should have a dominant influence on the design of embedded microelectronic systems. In addition, these systems must meet rigid form factor (such as size, power, and weight) constraints, which further complicate the design task. For designers of high-end embedded microsystems or large-volume consumer products, rapidly prototyping cost-effective implementations that meet stringent performance, functional, timing, and physical requirements is a formidable challenge.

Traditional design methodology

Figure 1 shows a typical design methodology for high-performance embedded avionics systems.¹ The approach suffers from numerous limitations:

- use of written requirements
- long prototyping times
- high design costs
- in-cycle silicon fabrication and test
- limited architectural exploration
- lack of systematic reuse
- lack of hardware-software codesign

Using written requirements to specify system functions and constraints fosters ambiguity in interpretation and does not facilitate customer interaction, thus leading to increased design iterations and low customer satisfaction. In addition, current practice predominantly relies on designer experience and ad hoc techniques to select an architecture and allocate algorithm functionality. This approach severely limits a designer's ability to explore the design space to improve the design. Furthermore, integration of hardware subsystems and application software does not occur until after hardware fabrication. Therefore, significant design flaws go undetected until the integration step, making design errors very costly. Flaws discov-

ered late in the design cycle incur significant rework and long delays. Under the design method shown in Figure 1, development costs range from \$20 million to \$100 million, and prototyping times range from 37 to 73 months.

To minimize overall system costs, traditional system-level design and test methodologies attempt to minimize hardware costs. However, these approaches overlook an important characteristic of software prototyping. Parametric studies based on historical project data show that designing and testing software is difficult if margins of slack for hardware CPU and memory resources are too restrictive.² Severe resource constraints may require software developers to interact directly with the operating system and/or hardware to optimize the code for system requirements. Such constrained architectures particularly increase the length of the integration and test phase because software is much trickier to write.

In systems whose hardware consists mainly of commercial, off-the-shelf (COTS) parts, the time and cost of software prototyping and design can dominate the schedule and budget. If physical constraints permit, designers can relax the hardware platform to achieve significant reductions in overall development cost and time. Figure 2 illustrates the software prototyping principle that software dominates system development cost when CPU and memory use are high. But as developers reduce resource use by adding extra hardware resources, software costs usually decrease drastically.

When form factor constraints on the system are severe, the designer can choose an advanced packaging technology such as the multichip module (MCM) to further exploit the software prototyping principle. The high packaging density and low power dissipation offered by advanced packaging allow more slack in the hardware architecture without violating system-level physical constraints. The designer must trade the reduction in software cost, however, for an increase in production cost due to MCM manufacturing.

While constraining the hardware-software architecture is

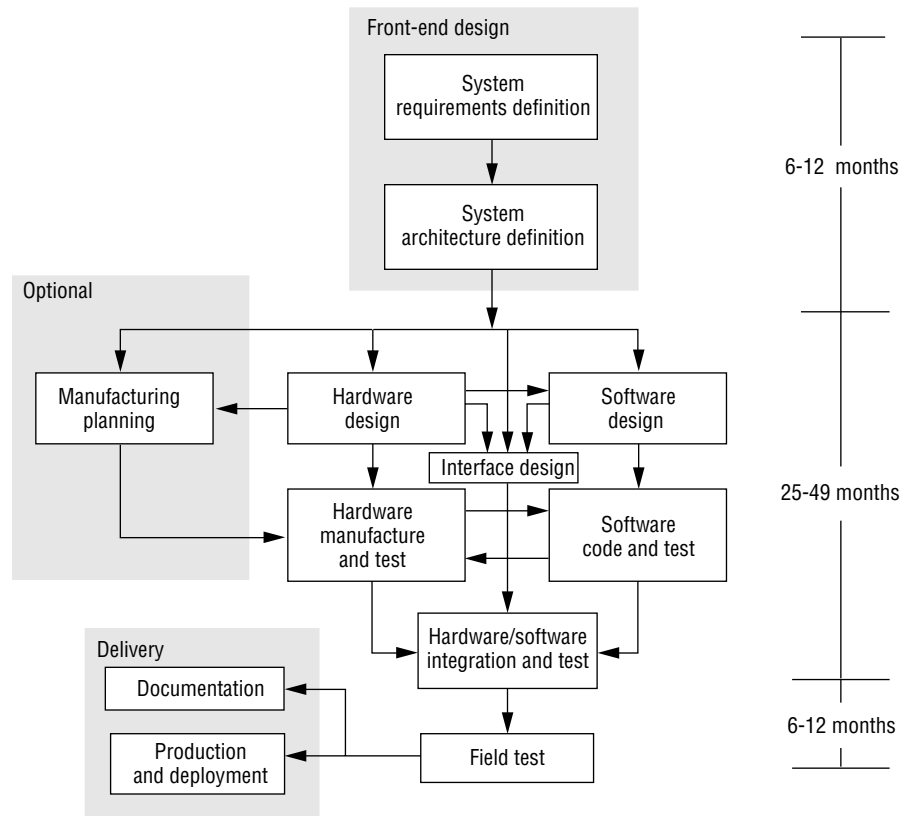


Figure 1. Typical design process for an embedded signal processor.

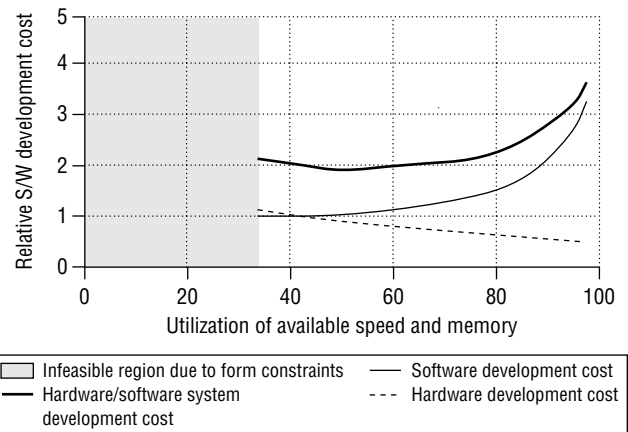


Figure 2. Effect of hardware constraints on hardware-software prototyping costs.

detrimental to software development cost, the corresponding effect on development time can be even more devastating. Time-to-market costs often outweigh design, prototyping, and production costs of commercial products. A recent survey showed that being six months late to market resulted in an av-

erage 33% profit loss, assuming a five-year product lifetime. Early market entry increases product yield, market share, and brand name recognition. Figure 3 shows a model of demand

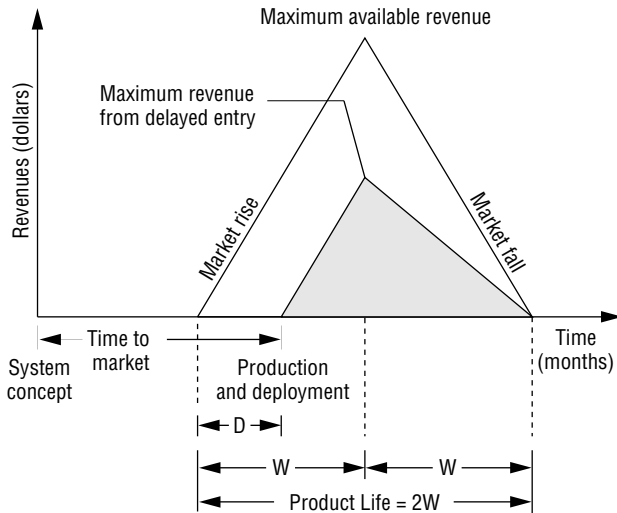


Figure 3. A typical time-to-market cost model. D: delay; W: market window.

and potential sales revenues for a new product (based on market research performed by Logic Automation, now owned by Synopsys). The unshaded region of the triangle signifies revenue loss due to late market entry. If the product life cycle is short, being late to market can spell disaster.

Cost-driven system-level design

To effectively address these system-level design challenges, product developers need a unified approach that considers the costs of both software and hardware options. This approach, which we call cost-driven system-level design, converges hardware and software design efforts into a methodology that improves cost, cycle time, and quality, and enhances design space exploration.

We have developed such a methodology at Georgia Tech’s Center for Signal and Image Processing under the auspices of the US Defense Advanced Research Projects Agency’s RASSP (Rapid Prototyping of Application-Specific Digital Signal Processors) program. Aimed at COTS-based embedded systems, the methodology uses parametric cost and development time estimation models to drive the design process. It seamlessly integrates a cost-driven architecture design engine (CADE) with a library-based cosimulation

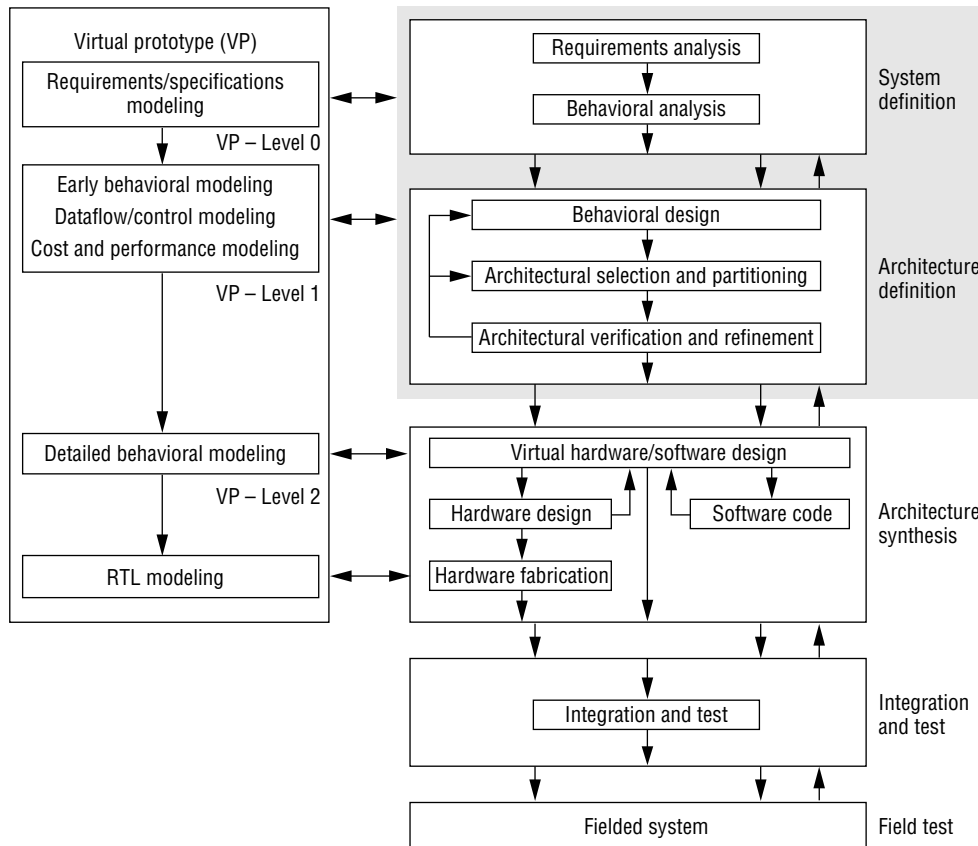


Figure 4. COTS-based embedded-system design methodology based on virtual prototyping.

and coverification environment for rapid prototyping. We use virtual prototypes³ to perform hierarchical design verification, with VHDL (VH-SIC Hardware Description Language) software models of the hardware executing a representation of the application code. Figure 4 diagrams the overall process flow. Our research focuses on demonstrating how to implement the shaded process steps (system definition and architecture definition) using virtual prototyping in an automated environment.

We believe that emphasizing cost-related issues benefits the cost-effectiveness of embedded microsystems more in the early design stages than in the later stages. Figure 5,⁴ which depicts costs committed versus costs incurred over the product life cycle, illustrates the rationale for our belief. Although the

front-end design process typically involves less than 10% of the total prototyping time and cost, it accounts for more than 80% of a system's life-cycle cost. For this reason, our research focuses on the front-end design process.

Our approach uses cost estimation models as well as performance estimation models to facilitate system-level design exploration early in the design cycle. We model the architecture selection process using mathematical programming formulations. We implement the models with commercial optimization packages, which efficiently solve complex problems, enabling the user to concentrate on problem-specific issues rather than data structures and implementation details. As output, CADE produces candidate architectures that we verify using VHDL performance-modeling technology.⁵

Methodology classification

We classify system-level design and test methodologies into the following broad categories. We will compare their results in an application case study of an avionic synthetic aperture radar (SAR) system later in the article.

1. *Standard COTS—minimum hardware cost.* The industry's most common approach tries to minimize hardware costs. Because control and diagnostic software design usually takes place after hardware fabrication, it is complicated by errors in hardware and tight constraints of the hardware platform.
2. *COTS with cost modeling—minimum system cost.* This methodology tries to minimize the sum of hardware and software costs, while maximizing profits.
3. *Full custom—custom hardware and software.* Another common practice, especially for designs with severe form factor or application-specific constraints, is to develop custom hardware and software. This approach is usually schedule- and cost-intensive.
4. *COTS with cost modeling and virtual prototyping.* This approach improves upon methodology 2 with the new technology of virtual prototyping, which uses hardware and software models to facilitate integration and test.
5. *COTS with cost modeling, virtual prototyping, advanced top-down programming, and reuse.* This methodology is the most advanced. Proposed as part of DARPA's RASSP program, it is likely to achieve the best cost objectives.

Georgia Tech's CADE environment supports methodologies 2, 4, and 5. We are adding new methodologies using cost modeling and life-cycle analysis to develop upgrades of current legacy systems efficiently.

Proposed methodology

Our front-end design and test methodology involves the development of executable requirements and specifica-

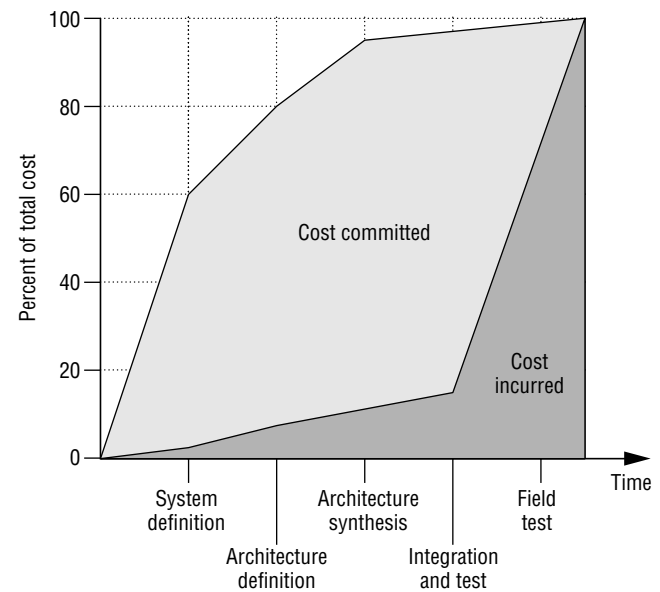


Figure 5. Cost commitment over the product life cycle.

tions,⁶ cost-driven architecture selection, task assignment, task scheduling, and VHDL performance modeling. Figure 6 (next page) illustrates the design process in detail. We start by translating written requirements into executable requirements and specifications, which we use to validate the original customer requirements and to remove ambiguities, thereby minimizing the requirements' volatility. In addition, detailed product functionality information in the executable requirements and specifications facilitates development of an accurate software size estimate, which is the primary software cost driver.

After we establish and verify the requirements, CADE automatically synthesizes a candidate hardware-software architecture. Inputs to this stage include system-level cost parameters, application requirements, and performance statistics. The cost parameters include software cost driver attributes (computer, personnel, product, and project),² COTS hardware procurement costs, product deployment deadlines, labor costs and constraints, and product market window attributes. Application requirements include form factor, precedence, and real-time constraints, as well as functional, memory, and communication requirements. Performance statistics consist of benchmark time measurements of common digital signal processing (DSP) primitives (fast Fourier transform, finite-length impulse response) executing on programmable processors in the VHDL COTS hardware reuse library. Common DSP functions tend to dominate the processing time for embedded signal processing applications. Therefore, by matching these algorithm statistics to the corresponding tasks in the functional speci-

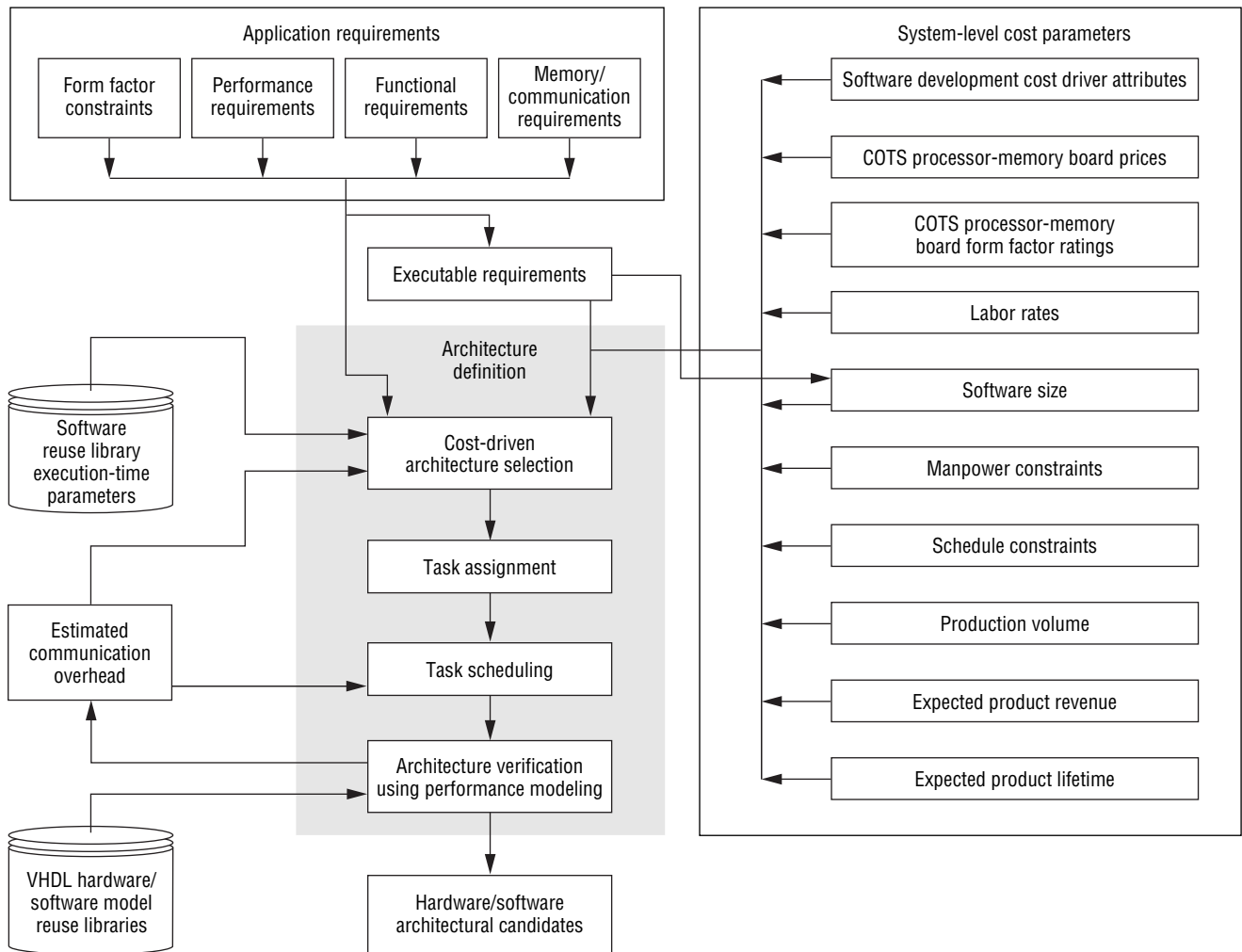


Figure 6. Incorporating cost modeling in embedded-system design.

fication, we can obtain a reliable performance estimate.

To provide for maximum architectural scalability, we assume a distributed memory architecture. The target architecture consists of multiple programmable processors, DRAM, and I/O devices connected over a high-performance interconnect network. CADE's goal is to determine the number and type of COTS programmable processor boards, the memory capacity, and the packaging technology. At the same time, it must map the data/control flowgraph nodes to the architectural elements and schedule them so as to optimally meet design constraints and economic objectives.

We then simulate the architectural candidates generated by CADE, using VHDL performance models to verify that an architecture meets system-level requirements. After performance modeling, we feed updated communication overhead parameters such as communication queuing delays (and bottlenecks) back to the architecture selection stage.

CADE then synthesizes a new candidate architecture with the updated model parameters. We repeat this process until the candidate architecture meets performance requirements and no longer changes significantly between successive iterations. The resulting candidate architecture then moves to the detailed architecture synthesis stage for detailed behavioral modeling.

Cost-driven architecture selection

CADE models the architectural design problem as a constrained optimization problem. Its objective is to generate a hardware-software architecture that minimizes system life-cycle costs and maximizes potential profits, within performance and form factor constraints. This approach mathematically models the architecture selection process by formulating it as a mixed-integer nonlinear programming (MINLP) problem.

The objective function of the optimization problem consists of cost estimates for hardware production, time to market, and software development and maintenance. In industry sectors where these objectives may not apply, each organization can implement its own objective function, without affecting the quality of the approaches we propose. Currently, CADE derives its software development and maintenance cost estimates from the US Air Force's embedded-mode Revised Intermediate Cocomo (REVIC) development effort and time equations. In the REVIC model, the development cycle runs from the contract award through hardware-software integration and testing. The REVIC software cost models are well suited for avionics systems such as SAR because their calibrations reflect actual Air Force experience. In practice, REVIC produced estimates close to actual costs for a variety of military signal processor projects.⁷

REVIC's development effort, development time, and annual maintenance effort equations, derived from Boehm's pioneering work,² are of the form shown in Equations 1–3 ("Cost models and functions" box). The units of the effort equations are in person-months, and development time is in calendar months. The software size estimate KSLOC, which includes application code, OS kernel services, control and diagnostics, and support software, is expressed as the number of source lines of code (thousands). However, this estimate should first be adjusted for reuse of COTS software components. The effort adjustment factors f_i and m_i model the effect of personnel, computer, product, and project attributes on software development and maintenance cost (see the REVIC reference manual for detailed descriptions⁸).

The hardware resource constraint effort adjustment factors f_E and f_M denote the effects of execution time and main storage margins on development cost. Annual change traffic ACT corresponds to the fraction of the software product's source code that undergoes change during a typical year, either through addition or modification. Model parameters A , B , C , and D capture the software project's multiplicative effects and economies and diseconomies of scale. These parameters should be calibrated to organization-specific and application-specific requirements and project data.

To minimize time to market for the system, we assume that the design is using only COTS hardware components. Therefore, we extrapolate the hardware production cost estimates from price quotes from vendors of COTS multi-processor cards. In CADE, we use the time-to-market cost model shown in Figure 3 to estimate the effect of delivering the product late. We can express this loss in revenue mathematically as $r_L = R_0 * d(3W-d)/(2W^2)$. R_0 refers to the expected product revenue. W is half the product life cycle (months). Delay d is the amount (months) by which the system's time to market (development time) exceeds the product deployment deadline.

Cost models and functions

Development effort, development time, annual maintenance effort:

$$s_E = A(KSLOC)^B f_E \cdot f_M \cdot \prod_{i=1}^{17} f_i \quad (1)$$

$$s_T = C(s_E)^D \quad (2)$$

$$m_E = ACT \left(A(KSLOC)^B f_E \cdot f_M \cdot \prod_{i=1}^{16} m_i \right) \quad (3)$$

Multiobjective cost function:

$$Cost = C^S s_E + V \left[\sum_i \sum_j (C_{ij}^p x_{ij} + C^M y) \right] + C^{MAINT} T^{MAINT} m_E + r_L \quad (4)$$

Effort adjustment and utilization:

$$f_E = 1 + 0.55u_2^p + 1.267u_3^p + 3.6u_4^p \quad (5)$$

$$f_M = 1 + 0.3u_2^m + u_3^m + 3.5u_4^m \quad (6)$$

where

$$u_p = \sum_{i=1}^4 u_i^p \quad (7)$$

$$u_m = \sum_{i=1}^4 u_i^m \quad (8)$$

Utilization, processors, and memory capacity:

$$u_p = \frac{T_R}{T_A} \quad (9)$$

$$u_m = \frac{M_R}{y} \quad (10)$$

where

$$T_R = \sum_j \left(\sum_{k \in T1} W_1 T_{jk}^{exe} q_j + \sum_{k \in T2} W_2 T_{jk}^{exe} q_j + \dots \right) + COMM \quad (11)$$

$$T_A = LCM(RT1, RT2, \dots) \sum_i \sum_j N_j^p x_{ij} \quad (12)$$

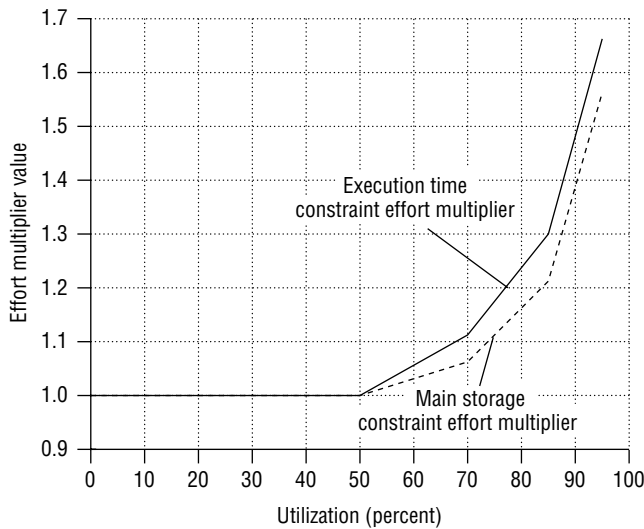


Figure 7. The effect of hardware constraints on execution time and main storage constraint effort multipliers. (Source: B. Boehm²)

Equation 4 states the multiobjective function quantitatively. The model's primary decision variables are x_{ij} and y . Variable x_{ij} denotes the number of processor boards of type i (Analog Devices' Sharc, Intel's i860) and configuration j (two-processor card, four-processor card). Variable y refers to the number of 4-Mbyte memory chips. The cost function parameters include the following: C^S , software labor cost per person-month; V , production volume; C_{ij}^P , procurement cost of a processor board of type i and configuration j ; C^M , procurement cost per 4 Mbytes of DRAM board; C^{MAINT} , software maintenance labor cost per person-month; and T^{MAINT} , length of time the product is maintained (years).

The software development and maintenance efforts s_E and m_E are functions of the number of processors and the memory capacity. More specifically, these software effort quantities are directly related to the execution time and main storage effort multipliers f_E and f_M , which in turn are related to the design variables. Figure 7 shows the relation between these effort adjustment factors and the CPU and memory utilization factors.² As processor and memory utilization increase above 50%, the execution time and main storage constraint effort multipliers begin to increase drastically. Increases in effort multiplier values are directly proportional to increases in software development effort and time. Equations 5–8 describe this relationship mathematically. Processor and memory utilization factors u_p and u_m consist of the aggregate total of corresponding utilization increments u_i^p and u_i^m .

Processor utilization and memory utilization are inversely proportional to the number of processors and memory capacity, as shown in Equations 9–12. T_R is the total time re-

quired for execution of application tasks, as well as for the accompanying OS and communication overhead. T_A is the total execution time available for processing. We calculate this value by multiplying the total number of processors by the least common multiple of real-time task deadlines. M_R is the total memory required by the application and support software. $COMM$ is the estimated communication overhead. We arrive at the initial communication overhead estimate by making an educated guess. In CADE, we initially estimate the overhead as 30% of the worst-case communication time. The worst-case communication overhead is the overhead that would be incurred if every task were assigned to separate processors.

Static analytical techniques for modeling detailed communication estimation can be computationally intensive as well as inaccurate due to dynamic effects such as queuing delays and network congestion. For this reason, we iteratively update the communication overhead estimate with data obtained from the architecture verification stage, which uses simulation-based performance modeling for increased accuracy. Other auxiliary variables and parameters are the following: q_j , a binary variable that signifies the processor type used in the architecture; T_{jk}^{exe} , a benchmark measurement for estimating the time to execute task k on a processor of type j ; $T1, T2, \dots$, the set of tasks that have real-time constraints $RT1, RT2, \dots$; W_1, W_2, \dots , the percent of computation time spent executing tasks with $RT1, RT2, \dots$; N_j^p , the number of processors on a board with configuration j .

Time-to-market cost r_L is also a function of the amount of hardware resources allocated for the architecture. This is due to the fact that time-to-market cost is a function of delay in delivering the product to market. In turn, delay is a function of development time, and as defined in Equation 2, development time is a function of software effort.

In addition to minimizing cost, avionics systems usually must meet stringent form factor constraints. In CADE, we model these constraints as linear functions of the number of processor boards and the amount of memory allocated to the architecture.

We are currently using the General Algebraic Modeling System (GAMS) optimization package—more specifically, the GAMS/Dicopt MINLP package—to solve models of this form. Dicopt uses Minos, a nonlinear programming package, and the Optimization Subroutine Library (OSL), a mixed-integer programming package, to rigorously solve these problems. Dicopt generates a locally optimal solution to the MINLP model. We use linearization techniques to transform the nonconvex constraints in our optimization architecture selection model into convex constraints, thereby ensuring the locally optimal solution obtained will also be the globally optimal solution. MINLP software packages are suitable for solving the architecture selection problem

because it is evident from our formulation of the problem that the variables are relatively few. The MINLP method allows a quantitative description of the architecture selection problem as well as an efficient solution process.

Case study

A case study of a polarimetric SAR system, which we used as a benchmark in the RASSP demonstration studies, shows how the proposed approaches compare with standard industry practice. Our results were very close to those actually reported by a leading defense contractor for the minimum cost approaches using COTS and custom methodologies.

SAR benchmark. SAR is an important tool for the collection of high-resolution, all-weather image data and is applicable to tactical military and civilian remote-sensing systems. In addition, SAR can identify man-made objects on the ground or in the air. Such object identification typically requires real-time processing by means of an embedded signal processor.

In our case study, we consider the design of an SAR processor that must form images in real time on board an aircraft. Figure 8 shows a block diagram of the SAR image formation algorithm.

The algorithm's two principal stages are range processing and azimuth compression processing. First, range processing transforms each SAR data pulse consisting of 4,064 real video samples into a compressed range pulse consisting of 2,048 complex samples. Then, azimuth compression, using cross-range convolution filtering, places compressed range pulses in time sequence into a 2D (2,048 × 1,024) processing array and convolves each row of the array with a row-specific reference kernel. The convolution outputs are saved in an image array, which becomes the output strip-map image of the SAR.

At its maximum PRF (pulse repetition frequency) of 556 Hz, the radar delivers 512 pulses with 4,064 data samples for each of the four polarizations in 920 milliseconds. The processor must form a 512-pulse image for each of three polarizations in real time with a latency not greater than 3 seconds. Given a maximum PRF of 556 Hz for the radar, the real-time constraint on the range-processing tasks is 1.8 milliseconds, and the real-time constraint on the azimuth-processing tasks is 920 milliseconds. The computational requirement of the algorithm is about 1.1 Gflop/s. The memory requirement is approximately 77 Mbytes.⁹

Results. We performed a detailed cost analysis comparison of the architectures generated by our CADE environment and those generated by standard design methodologies. Specifically, from the list presented earlier, we compared methodologies 2, 4, and 5 with 1 and 3.

Table 1 (next page) lists the system-level cost parameters

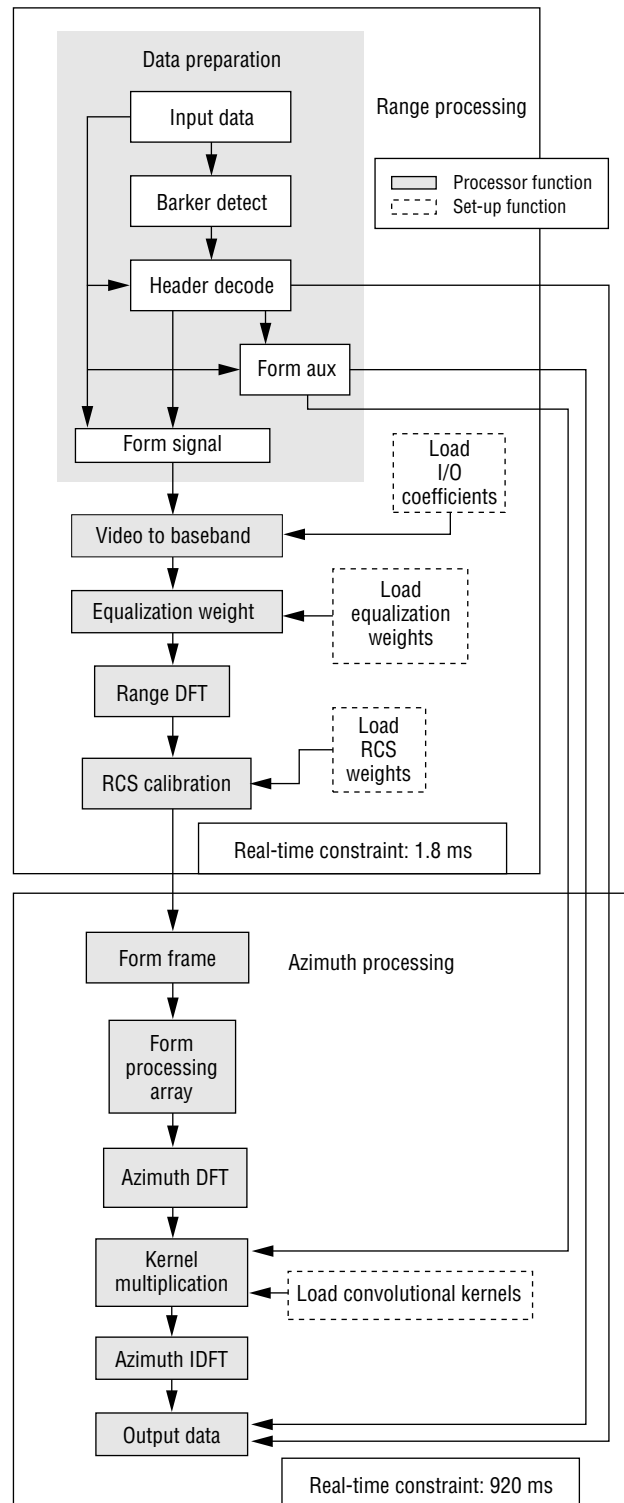


Figure 8. SAR image processing.

affecting the design of the SAR processor. These parameters remain constant in all experiments unless explicitly stated

Table 1. SAR system-level cost parameters.

Parameter	Value
Sharc 2-processor board price	\$8,000
Sharc 4-processor board price	\$15,000
Sharc 6-processor board price	\$25,000
Sharc 8-processor board price	\$40,000
i860 2-processor board price	\$12,000
i860 4-processor board price	\$22,000
i860 6-processor board price	\$37,000
i860 8-processor board price	\$60,000
Sharc 2-processor board power	12 W
Sharc 4-processor board power	20 W
Sharc 6-processor board power	25 W
Sharc 8-processor board power	28 W
i860 2-processor board power	16 W
i860 4-processor board power	28 W
i860 6-processor board power	35 W
i860 8-processor board power	42 W
DRAM price per 4 Mbytes	\$1,400
4-Mbyte DRAM power dissipation	4 W
Estimated KSLOC	8.75 KSLOC
Maintenance period	15 years
Software labor cost per person-month	\$15,000
Software maintenance labor cost	\$15,000
Maximum full-time software personnel	3
Annual change traffic	15%
SAR processor unit price	\$1 million
Expected revenue	\$1 million × production volume
Product life cycle	36 months
Product deployment deadlines	24 and 32 months

otherwise. We derived most of these parameters from a manual cost analysis of the SAR benchmark performed by RASSP participants at MIT Lincoln Labs.⁷

In our experiments, we assumed that the interconnect topology would consist of six-port Mercury Raceway crossbars connected in a fat tree network. We derived the prices of two- and four-processor cards from the costs of two or four COTS processors, crossbar ASICs, printed circuit board area, and other elements. The two- and four-processor boards contain single-chip packaging technology only. The six-processor card consists of three COTS MCM-L modules, each module consisting of two COTS processor chips. The eight-processor card consists of two COTS MCM-D modules, each module consisting of three COTS processor chips. Each board can contain up to 64 Mbytes of DRAM. The DRAM price includes the DRAM

chip, the required printed circuit board area, and associated interconnect. We adjusted the software size estimate for reuse of DSP software library elements. We used REVIC default values for the multiplicative constants and scale factors in the software development and maintenance equations.

For methodology 3, however, some of these system-level cost parameters are slightly different. Using this methodology, we assume that the software size estimate will increase to 12,000 source lines of code due to the customization of software routines. In addition, custom designing the processor board leads to a six-month increase in overall development time over the minimum COTS hardware cost approach, methodology 1. We traded this increase in development time and cost for a 40% reduction in unit production cost resulting from customizing the SAR implementation.

In some cases, the values of software cost driver effort multipliers f_i differ, depending on the design and test methodology used. For methodologies 1, 2, and 3, we assume for simplicity that all effort multipliers have a nominal value of 1.0 (except the hardware resource constraint effort multipliers f_E and f_M). However, for methodologies 4 and 5, the requirements volatility effort multiplier has a low rating of 0.92 due to the use of executable requirements. In addition, RASSP demonstration studies have shown that virtual prototyping provides a twofold improvement in development productivity.³ Methodology 5's routine use of advanced top-down design and programming methodologies and reuse leads to modern programming practices and software tool usage effort multiplier values of 0.82 and 0.62, respectively.

Assuming a tight product deployment deadline of 24 months and a relaxed deadline of 32 months for the SAR application, we analyzed costs vis-à-vis schedule for the five methodologies. Tables 2 and 3 show the detailed results. The tight power constraint is a system power dissipation limit of 150 watts.

Table 2 shows that for a production volume of 100 radar systems, methodology 2 produced a processor architecture consisting of eight 4-processor Sharc boards and 124 Mbytes of memory, thus resulting in processor and memory utilizations of 73% and 62%. Methodology 2 met the stringent schedule deadline at a cost of \$20.6 million. Due to the low cost of DRAM with respect to COTS processor cost, we extend memory margins more than execution time margins when we attempt to reduce software development time and cost.

Table 3 (p. 34) shows that methodology 1 failed to meet the 24-month deadline (prolonging the schedule to 30.5 months) and incurred a \$67.3 million cost for 100 systems. The minimum-hardware-cost system implementation produced by methodology 1 consisted of one 2-processor Sharc board, six 4-processor Sharc boards, and 84 Mbytes of memory. The resulting processor and memory utilizations were 95% and 91%.

When we relaxed the constraint on time to market to 32

Table 2. SAR architectural profiles produced by system-level design and test methodologies supported by CADE (2, 4, and 5) over a range of production volumes.

Design methodology	Volume	Processor architecture (boards per configuration)				No. of proc.	Memory allocation (Mbytes)	Utilization (%)		
		Processor type	2-proc.	4-proc.	6-proc.			8-proc.	Proc.	Memory
2 (Tight schedule)	10	Sharc	0	9	0	0	36	144	69	53
	50	Sharc	1	8	0	0	34	124	73	62
	100	Sharc	1	8	0	0	34	124	73	62
2 (Relaxed schedule)	10	Sharc	0	9	0	0	36	144	69	53
	50	Sharc	1	7	0	0	30	108	83	71
	100	Sharc	0	7	0	0	28	108	89	71
2 (Tight schedule and power)	10	Sharc	0	0	0	4	32	152	78	50
	50	Sharc	0	0	0	4	32	152	78	50
	100	Sharc	0	0	0	4	32	152	78	50
4 (Tight schedule)	10	Sharc	0	9	0	0	36	112	69	68
	50	Sharc	0	7	0	0	28	100	89	77
	100	Sharc	0	7	0	0	28	92	89	83
5 (Tight schedule)	10	Sharc	1	7	0	0	30	108	83	71
	50	Sharc	0	7	0	0	28	92	89	83
	100	Sharc	0	7	0	0	28	88	89	87

months, methodology 1 (with minimum cost of hardware) met the requirement, and the cost decreased to \$19.8 million. Surprisingly, the cost was less than in the case of a 24-month time-to-market window because there was no loss of revenue penalty for missing the market window. Table 3 shows how cost, schedule, and related components of these approaches vary as a function of production volume. Methodology 2's cost improvement over methodology 1 for the stringent schedule was $67.3/20.6 = 3.3$.

In our next set of experiments, we limited the radar system's power constraint to 150 watts. The product deployment deadline was 24 months. Methodology 2 generated a system architecture composed of four 8-processor Sharc boards and 152 Mbytes of memory, resulting in processor and memory utilizations of 78% and 50%. Methodology 2's cost increased slightly as a result of the use of MCM technology at a higher hardware cost. However, the MCM technology's increased density allowed us to obtain significant cost savings by relaxing the hardware architecture to reduce software cost and development time. The cost improvement of methodology 2 over methodology 1 was still substantial at 2.5 for production runs of 100 radar systems. See Tables 2 and 3 for a more detailed analysis.

Methodology 3 (using custom hardware and software)

lengthened the schedule (as we expected), but it reduced the hardware system cost 40%. The development costs were large: \$114.2 million and 40.4 months (missing the 24-month deadline). This implies that methodology 2 improves upon methodology 3 by a factor of 5.5. Table 3 presents a detailed cost-sensitivity analysis of methodology 3 over a range of production volumes.

Methodology 4 (which is methodology 2 with virtual prototyping added) resulted in a lower cost and quicker schedule: \$16.2 million and 21.9 months. This implies that improvements of methodology 4 over methodologies 1 and 3 are 4.2 and 7.0, respectively. Again, see Tables 2 and 3.

Finally, methodology 5 (methodology 4 plus top-down design methodologies and advanced prototyping techniques) resulted in a cost of \$15 million and a schedule of 18.1 months. The improvements over methodologies 1 and 3 are thus 5.2 and 8.6, for a production run of 10 radar systems (Table 3).

Thus, we found that cost modeling and related methodological enhancements allow us to design and test systems at minimum system cost. In contrast, the minimum-hardware-cost approach (which is current industry practice) results in excessive costs in time and dollars. Consistent with results from the RASSP benchmarking study, we observed that fac-

Table 3. Cost and schedule of SAR implementations generated by the design and test methodologies.

Design methodology	Volume	Cost breakdown (% of total)			Total cost (\$ millions)	Schedule (months)	
		Time to market	Software development	Software maintenance			Hardware production
1 (Tight schedule)	10	36.3	16.6	37.4	9.7	13.1	30.5
	50	63.9	5.9	13.2	17.1	37.2	30.5
	100	70.6	3.2	7.3	18.9	67.3	30.5
2 (Tight schedule)	10	0	19.6	44.2	36.2	5.1	23.8
	50	0	9.0	20.0	71.0	12.1	24.0
	100	0	5.2	11.7	83.1	20.6	24.0
1 (Relaxed schedule)	10	0	26.1	58.7	15.3	8.3	30.5
	50	0	16.2	36.4	47.4	13.4	30.5
	100	0	11.0	24.7	64.3	19.8	30.5
2 (Relaxed schedule)	10	0	19.6	44.2	36.2	5.1	23.8
	50	0	10.8	23.9	65.3	11.5	27.6
	100	0	7.6	16.6	75.8	18.8	31.7
2 (Tight schedule and power)	10	4.0	18.5	41.5	36.1	5.9	24.3
	50	7.7	7.1	15.9	69.3	15.4	24.3
	100	8.7	4.0	9.0	78.3	27.2	24.3
3 (Tight schedule)	10	44.1	14.7	33.1	8.2	21.6	40.4
	50	75.9	5.1	11.4	7.7	62.8	40.4
	100	83.4	2.8	6.3	7.6	114.2	40.4
4 (Tight schedule)	10	0	15.2	34.3	50.5	3.5	19.3
	50	0	7.8	17.6	74.6	9.4	21.5
	100	0	4.8	10.7	84.5	16.2	21.9
5 (Tight schedule)	10	0	12.4	27.8	59.8	2.5	16.4
	50	0	4.8	10.9	84.3	8.1	17.6
	100	0	2.8	6.4	90.8	15.0	18.1


tors of improvement lie between 3 and 9, which are quite significant. In addition, we could use CADE to design systems very quickly for a shorter time to market at a fixed cost.

As production volume increases, development costs for design and test decrease relative to production costs. The use of advanced methodologies will also reduce life-cycle maintenance costs.

WE DON'T RECOMMEND that all sectors of the electronics industry use the same cost model or objective function. Each sector must develop and refine its models and functions on

the basis of its own historical information. We strongly argue, however, for an effective system-level design methodology that incorporates cost modeling. Seamlessly integrated into the electrical engineering aspects of the process, cost modeling can produce an order-of-magnitude improvement in cost-related objective functions. At Georgia Tech and VP Technologies, we have shown that modern design and test methodologies can benefit from the automated use of cost models early in the design process, with very impressive schedule and cost savings results. Indeed, industry has recently used parametric cost estimators with a great deal of success. For example, Motorola's Iridium project, one of the

largest and most complex software projects ever, used SEER, a parametric software cost modeling tool. The tool reportedly estimated costs within 3% of actual project costs.¹⁰

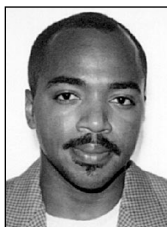
An interface between systems engineering and electrical engineering is essential to making this technology widely accessible and easily applicable. Much exciting research awaits us in this relatively new area coupling architectural design and test with economics. 

Acknowledgments

DARPA's RASSP program under contract F33615-94-C-1493, 1994-1997, partially supported the research described here.

References

1. V.K. Madisetti, "Rapid Digital System Prototyping: Current Practice, Future Challenges," *IEEE Design & Test of Computers*, Vol. 13, No. 3, Fall 1996, pp. 12-22.
2. B. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
3. V.K. Madisetti and T.W. Egolf, "Virtual Prototyping of Embedded Microcontroller-Based DSP Systems," *IEEE Micro*, Vol. 15, No. 5, Oct. 1995, pp. 9-21.
4. W. Fabrycky and B. Blanchard, *Life-Cycle Cost and Economic Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 1991.
5. L. Dung and V.K. Madisetti, "Conceptual Prototyping of Scalable Embedded DSP Systems," *IEEE Design & Test*, Vol. 13, No. 3, Fall 1996, pp. 54-65.
6. A.H. Anderson et al., "VHDL Executable Requirements," *Proc. First Annual RASSP Conf.*, DARPA, US Dept. of Defense, Washington, D.C., 1994, pp. 87-90.
7. J. Anderson, "Projecting RASSP Benefits," *Proc. Second Annual RASSP Conf.*, DARPA, US Dept. of Defense, 1995.
8. *REVIC Software Cost Estimating Model User's Manual Version 9.2*, US Air Force Analysis Agency, Dec. 1994.
9. B. Zuerndorfer and G. Shaw, "SAR Processing for RASSP Application," *Proc. First Annual RASSP Conf.*, DARPA, US Dept. of Defense, 1994, pp. 253-268.
10. D. Lyons, "SEER Helps Keep Satellite Cost Estimates Down to Earth," *Infoworld*, Nov. 27, 1995.



James A. DeBardelaben is a PhD candidate in the School of Electrical and Computer Engineering at Georgia Institute of Technology. He is also a technology specialist at VP Technologies. His research interests include hardware-software codesign, VHDL-based rapid

prototyping, system-level cost modeling, and mathematical programming. He received an BSc degree in electrical engineering from Brown University and an MSE degree in computer engineering from Princeton University.



Vijay K. Madisetti is an associate professor of electrical and computer engineering at Georgia Tech and president and CEO of VP Technologies, Marietta, Georgia, a subcontractor in system-level modeling and design. He is the technical director of the RASSP Education and

Facilitation Program and leads Georgia Tech's signal processing efforts in the Army Research Laboratory's Federated Sensors Laboratory Program. Madisetti obtained his PhD in electrical engineering and computer sciences from the University of California, Berkeley. He is a member of the IEEE and the Computer Society.



Anthony J. Gadiant is the director of research at the South Carolina Research Authority in Charleston, South Carolina. He is also the principal investigator in the RASSP Education and Facilitation Program. His research interests include environments for distributed collaboration, distance learning, design tool interoperability, and rapid prototyping of embedded electromechanical systems. Gadiant obtained a BSEE from the University of Virginia, an MBA from Wright State University, and a PhD in electrical and computer engineering from Carnegie Mellon University.

Send questions and comments about this article to Vijay K. Madisetti, ECE-0250, Georgia Tech, Atlanta, GA 30332-0250; vkm@ee.gatech.edu.

Moving?

Please notify us four weeks in advance.

Name (Please print)

New Address

City

State/Country

Zip

Mail to:
IEEE Computer Society
Circulation Department
PO Box 3014
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314

- List new address above.
- This notice of address change will apply to all IEEE publications to which you subscribe.
- If you have a question about your subscription, place label here and clip this form to your letter.

**ATTACH
 LABEL
 HERE**