Embedded System Design Issues (the Rest of the Story)

Philip Koopman
Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213
koopman@cs.cmu.edu
http://www.cs.cmu.edu/~koopman/

Abstract

Many embedded systems have substantially different design constraints than desktop computing applications. No single characterization applies to the diverse spectrum of embedded systems. However, some combination of cost pressure, long life-cycle, real-time requirements, reliability requirements, and design culture dysfunction can make it difficult to be successful applying traditional computer design methodologies and tools to embedded applications. Embedded systems in many cases must be optimized for life-cycle and business-driven factors rather than for maximum computing throughput. There is currently little tool support for expanding embedded computer design to the scope of holistic embedded system design. However, knowing the strengths and weaknesses of current approaches can set expectations appropriately, identify risk areas to tool adopters, and suggest ways in which tool builders can meet industrial needs.

1. Introduction

Approximately 3 billion embedded CPUs are sold each year, with smaller (4-, 8-, and 16-bit) CPUs dominating by quantity and aggregate dollar amount [1]. Yet, most research and tool development seems to be focussed on the needs of high-end desktop and military/aerospace embedded computing. This paper seeks to expand the area of discussion to encompass a wide range of embedded systems.

The extreme diversity of embedded applications makes generalizations difficult. Nonetheless, there is emerging interest in the entire range of embedded systems (*e.g.*, [2], [3], [4], [5], [6]) and the related field of hardware/software codesign (*e.g.*, [7]).

This paper and the accompanying tutorial seek to identify significant areas in which embedded computer design differs from more traditional desktop computer design. They also present "design challenges" encountered in the course of designing several real systems. These challenges are both opportunities to improve methodology and tool support as well as impediments to deploying such support to embedded system design teams. In some cases research and development has already begun in these areas — and in other cases it has not.

The observations in this paper come from the author's experience with commercial as well as military applications, development methodologies, and life-cycle support. All characterizations are implicitly qualified to indicate a typical, representative, or perhaps simply an anecdotal case rather than a definitive statement about all embedded systems. While it is understood that each embedded system has its own set of unique requirements, it is hoped that the generalizations and examples presented here will provide a

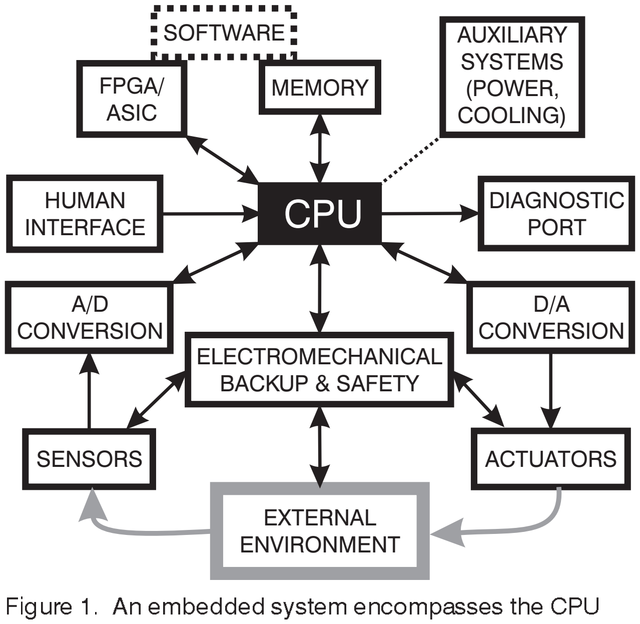

Figure 1. An embedded system encompasses the CPU as well as many other resources.

broad-brush basis for discussion and evolution of CAD tools and design methodologies.

2. Example Embedded Systems

Figure 1 shows one possible organization for an embedded system. In addition to the CPU and memory hierarchy, there are a variety of interfaces that enable the system to measure, manipulate, and otherwise interact with the external environment. Some differences with desktop computing may be:

- The human interface may be as simple as a flashing light or as complicated as real-time robotic vision.
- The diagnostic port may be used for diagnosing the system that is being controlled — not just for diagnosing the computer.
- Special-purpose field programmable (FPGA), application specific (ASIC), or even non-digital hardware may be used to increase performance or safety.
- Software often has a fixed function, and is specific to the application.

In addition to the emphasis on interaction with the external world, embedded systems also provide functionality specific to their applications. Instead of executing spreadsheets, word processing and engineering analysis, embedded systems typically execute control laws, finite state machines, and signal processing algorithms. They must often detect and react to faults in both the computing and surrounding electromechanical systems, and must manipulate application-specific user interface devices.

In order to make the discussion more concrete, we shall discuss four example systems (Table 1). Each example portrays a real system in current production, but has been slightly genericized to represent a broader cross-section of applications as well as protect proprietary interests. The four examples are a Signal Processing system, a Mission Critical control system, a Distributed control system, and a Small consumer electronic system. The Signal Processing and Mission Critical systems are representative of traditional

An example of:	**Signal Processing**	**Mission Critical**	**Distributed**	**Small**
Computing speed	1 GFLOPS	10 - 100 MIPS	1-10 MIPS	100,000 IPS
I/O Transfer Rates	1 Gb/sec	10 Mb/sec	100 Kb/sec	1 Kb/sec
Memory Size	32 - 128 MB	16 - 32 MB	1 - 16 MB	1 KB
Units Sold	10 - 500	100 - 1000	100 - 10,000	1,000,000+
Development Cost	\$20M - \$100M	\$10M - \$50M	\$1M - \$10M	\$100K - \$1M
Lifetime	15 - 30 years	20 - 30 years	25 - 50 years	10 - 15 years
Environment	Vibration, Heat	Heat, Vibration, Lightning	Dirt, Fire	Over-voltage, Heat, Vibration
Cost Sensitivity	\$1000	\$100	\$10	\$0.05
Other Constraints	Size, weight, power	Size, weight	Size	Size, weight, power
Safety	—	Redundancy	Mechanical Safety	—
Maintenance	Frequent repairs	Aggressive fault detection/maintenance	Scheduled maintenance	“Never” breaks
Digital content	Digital except for signal I/O	~½ Digital	~½ Digital	Single digital chip; rest is analog/power
Certification authorities	Customer	Federal Government	Development team	Customer; Federal Government
Repair time goal	1-12 hours	30 minutes	4 min. - 12 hours	1-4 hours
Initial cycle time	3-5 years	4-10 years	2-4 years	0.1-4 years
Product variants	1-5	5-20	10-10,000	3-10
Engineering allocation method	Per-product budget	Per-product budget	Allocation from large pool	Demand-driven daily from small pool
Other possible examples in this category:	Radar/Sonar Video Medical imaging	Jet engines Manned spacecraft Nuclear power	High-rise elevators Trains/trams/subways Air conditioning	Automotive auxilliaries Consumer electronics “Smart” I/O

Table 1. Four example embedded systems with approximate attributes.

military/aerospace embedded systems, but in fact are becoming more applicable to general commercial applications over time.

Using these four examples to illustrate points, the following sections describe the different areas of concern for embedded system design: computer design, system-level design, life-cycle support, business model support, and design culture adaptation.

Desktop computing design methodology and tool support is to a large degree concerned with initial design of the digital system itself. To be sure, experienced designers are cognizant of other aspects, but with the recent emphasis on quantitative design (*e.g.*, [8]) life-cycle issues that aren't readily quantified could be left out of the optimization process. However, such an approach is insufficient to create embedded systems that can effectively compete in the marketplace. This is because in many cases the issue is not whether design of an immensely complex system is feasible, but rather whether a relatively modest system can be highly optimized for life-cycle cost and effectiveness.

While traditional digital design CAD tools can make a computer designer more efficient, they may not deal with the central issue — embedded design is about the system, not about the computer. In desktop computing, design often focuses on building the fastest CPU, then supporting it as required for maximum computing speed. In embedded systems the combination of the external interfaces (sensors, actuators) and the control or sequencing algorithms is or primary importance. The CPU simply exists as a way to implement those functions. The following experiment should serve to illustrate this point: ask a roomful of people what kind of CPU is in the personal computer or workstation they use. Then ask the same people which CPU is used for the engine controller in their car (and whether the CPU type influenced the purchasing decision).

In high-end embedded systems, the tools used for desktop computer design are invaluable. However, many embedded systems both large and small must meet additional requirements that are beyond the scope of what is typically handled by design automation. These additional needs fall into the categories of special computer design requirements, system-level requirements, life-cycle support issues, business model compatibility, and design culture issues.

3. Computer Design Requirements

Embedded computers typically have tight constraints on both functionality and implementation. In particular, they must guarantee real time operation reactive to external events, conform to size and weight limits, budget power and cooling consumption, satisfy safety and reliability requirements, and meet tight cost targets.

3.1. Real time/reactive operation

Real time system operation means that the correctness of a computation depends, in part, on the time at which it is delivered. In many cases the system design must take into account worst case performance. Predicting the worst case may be difficult on complicated architectures, leading to overly pessimistic estimates erring on the side of caution. The Signal Processing and Mission Critical example systems have a significant requirement for real time operation in order to meet external I/O and control stability requirements.

Reactive computation means that the software executes in response to external events. These events may be periodic, in which case scheduling of events to guarantee performance may be possible. On the other hand, many events may be aperiodic, in which case the maximum event arrival rate must be estimated in order to accommodate worst case situations. Most embedded systems have a significant reactive component.

Design challenge:

- Worst case design analyses without undue pessimism in the face of hardware with statistical performance characteristics (*e.g.*, cache memory [9]).

3.2. Small size, low weight

Many embedded computers are physically located within some larger artifact. Therefore, their form factor may be dictated by aesthetics, form factors existing in pre-electronic versions, or having to fit into interstices among mechanical components. In transportation and portable systems, weight may be critical for fuel economy or human endurance. Among the examples, the Mission Critical system has much more stringent size and weight requirements than the others because of its use in a flight vehicle, although all examples have restrictions of this type.

Design challenges:

- Non-rectangular, non-planar geometries.
- Packaging and integration of digital, analog, and power circuits to reduce size.

3.3. Safe and reliable

Some systems have obvious risks associated with failure. In mission-critical applications such as aircraft flight control, severe personal injury or equipment damage could result from a failure of the embedded computer. Traditionally, such systems have employed multiply-redundant computers or distributed consensus protocols in order to ensure continued operation after an equipment failure (*e.g.*, [10], [11])

However, many embedded systems that could cause personal or property damage cannot tolerate the added cost of redundancy in hardware or processing capacity needed for traditional fault tolerance techniques. This vulnerability is often resolved at the system level as discussed later.

Design challenge:
- Low-cost reliability with minimal redundancy.

3.4. Harsh environment

Many embedded systems do not operate in a controlled environment. Excessive heat is often a problem, especially in applications involving combustion (*e.g.*, many transportation applications). Additional problems can be caused for embedded computing by a need for protection from vibration, shock, lightning, power supply fluctuations, water, corrosion, fire, and general physical abuse. For example, in the Mission Critical example application the computer must function for a guaranteed, but brief, period of time even under non-survivable fire conditions.

Design challenges:
- Accurate thermal modelling.
- De-rating components differently for each design, depending on operating environment.

3.5. Cost sensitivity

Even though embedded computers have stringent requirements, cost is almost always an issue (even increasingly for military systems). Although designers of systems large and small may talk about the importance of cost with equal urgency, their sensitivity to cost changes can vary dramatically. A reason for this may be that the effect of computer costs on profitability is more a function of the proportion of cost changes compared to the total system cost, rather than compared to the digital electronics cost alone. For example, in the Signal Processing system cost sensitivity can be estimated at approximately $1000 (*i.e.*, a designer can make decisions at the $1000 level without undue management scrutiny). However, with in the Small system decisions increasing costs by even a few cents attract management attention due to the huge multiplier of production quantity combined with the higher percentage of total system cost it represents.

Design challenge:
- Variable “design margin” to permit tradeoff between product robustness and aggressive cost optimization.

4. System-level requirements

In order to be competitive in the marketplace, embedded systems require that the designers take into account the entire system when making design decisions.

4.1. End-product utility

The utility of the end product is the goal when designing an embedded system, not the capability of the embedded computer itself. Embedded products are typically sold on the basis of capabilities, features, and system cost rather than which CPU is used in them or cost/performance of that CPU.

One way of looking at an embedded system is that the mechanisms and their associated I/O are largely defined by the application. Then, software is used to coordinate the mechanisms and define their functionality, often at the level of control system equations or finite state machines. Finally, computer hardware is made available as infrastructure to execute the software and interface it to the external world. While this may not be an exciting way for a hardware engineer to look at things, it does emphasize that the total functionality delivered by the system is what is paramount.

Design challenge:
- Software- and I/O-driven hardware synthesis (as opposed to hardware-driven software compilation/synthesis).

4.2. System safety & reliability

An earlier section discussed the safety and reliability of the computing hardware itself. But, it is the safety and reliability of the total embedded system that really matters. The Distributed system example is mission critical, but does not employ computer redundancy. Instead, mechanical safety backups are activated when the computer system loses control in order to safely shut down system operation.

A bigger and more difficult issue at the system level is software safety and reliability. While software doesn’t normally “break” in the sense of hardware, it may be so complex that a set of unexpected circumstances can cause software failures leading to unsafe situations. This is a difficult problem that will take many years to address, and may not be properly appreciated by non-computer engineers and managers involved in system design decisions ([12] discusses the role of computers in system safety).

Design challenges:
- Reliable software.
- Cheap, available systems using unreliable components.
- Electronic *vs.* non-electronic design tradeoffs.

4.3. Controlling physical systems

The usual reason for embedding a computer is to interact with the environment, often by monitoring and controlling external machinery. In order to do this, analog inputs and outputs must be transformed to and from digital signal levels. Additionally, significant current loads may need to be switched in order to operate motors, light fixtures, and other actuators. All these requirements can lead to a large computer circuit board dominated by non-digital components.

In some systems “smart” sensors and actuators (that contain their own analog interfaces, power switches, and small CPUS) may be used to off-load interface hardware from the central embedded computer. This brings the additional advantage of reducing the amount of system wiring and number of connector contacts by employing an embedded network rather than a bundle of analog wires. However, this change brings with it an additional computer design problem of partitioning the computations among distributed computers in the face of an inexpensive network with modest bandwidth capabilities.

Design challenge:

- Distributed system tradeoffs among analog, power, mechanical, network, and digital hardware plus software.

4.4. Power management

A less pervasive system-level issue, but one that is still common, is a need for power management to either minimize heat production or conserve battery power. While the push to laptop computing has produced “low-power” variants of popular CPUs, significantly lower power is needed in order to run from inexpensive batteries for 30 days in some applications, and up to 5 years in others.

Design challenge:

- Ultra-low power design for long-term battery operation.

5. Life-cycle support

Figure 2 shows one view of a product life-cycle (a simplified version of the view taken by [13]). First a need or opportunity to deploy new technology is identified. Then a product concept is developed. This is followed by concurrent product and manufacturing process design, production, and deployment. But in many embedded systems, the designer must see past deployment and take into account support, maintenance, upgrades, and system retirement issues in order to actually create a profitable design. Some of the issues affecting this life-cycle profitability are discussed below.

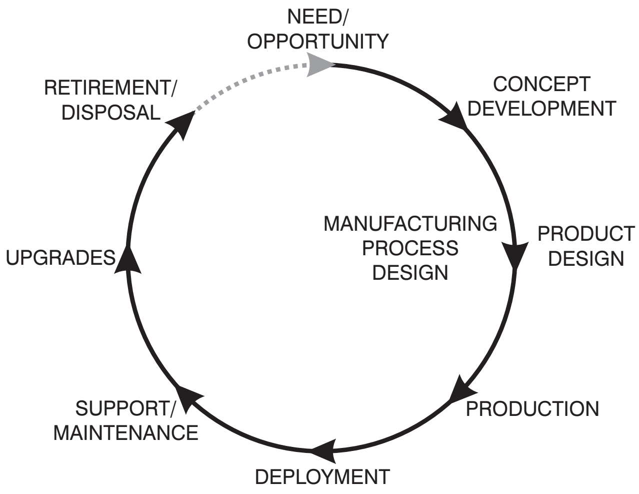

Figure 2. An embedded system lifecycle.

5.1. Component acquisition

Because an embedded system may be more application-driven than a typical technology-driven desktop computer design, there may be more leeway in component selection. Thus, component acquisition costs can be taken into account when optimizing system life-cycle cost. For example, the cost of a component generally decreases with quantity, so design decisions for multiple designs should be coordinated to share common components to the benefit of all.

Design challenge:

- Life-cycle, cross-design component cost models and optimization rather than simple per-unit cost.

5.2. System certification

Embedded computers can affect the safety as well as the performance the system. Therefore, rigorous qualification procedures are necessary in some systems after *any* design change in order to assess and reduce the risk of malfunction or unanticipated system failure. This additional cost can negate any savings that might have otherwise been realized by a design improvement in the embedded computer or its software. This point in particular hinders use of new technology by resynthesizing hardware components — the redesigned components cannot be used without incurring the cost of system recertification.

One strategy to minimize the cost of system recertification is to delay all design changes until major system upgrades occur. As distributed embedded systems come into more widespread use, another likely strategy is to partition the system in such a way as to minimize the number of subsystems that need to be recertified when changes occur. This is a partitioning problem affected by potential design changes, technology insertion strategies, and regulatory requirements.

Design challenge:

- Partitioning/synthesis to minimize recertification costs.

5.3. Logistics and repair

Whenever an embedded computer design is created or changed, it affects the downstream maintenance of the product. A failure of the computer can cause the entire system to be unusable until the computer is repaired. In many cases embedded systems must be repairable in a few minutes to a few hours, which implies that spare components and maintenance personnel must be located close to the system. A fast repair time may also imply that extensive diagnosis and data collection capabilities must be built into the system, which may be at odds with keeping production costs low.

Because of the long system lifetimes of many embedded systems, proliferation of design variations can cause significant logistics expenses. For example, if a component design is changed it can force changes in spare component inventory, maintenance test equipment, maintenance procedures, and maintenance training. Furthermore, each design change should be tested for compatibility with various system configurations, and accommodated by the configuration management database.

Design challenge:

- Designs optimized to minimize spares inventory.
- High-coverage diagnosis and self-test at system level, not just digital component level.

5.4. Upgrades

Because of the long life of many embedded systems, upgrades to electronic components and software may be used to update functionality and extend the life of the embedded system with respect to competing with replacement equipment. While it may often be the case that an electronics upgrade involves completely replacing circuit boards, it is important to realize that the rest of the system will remain unchanged. Therefore, any special behaviors, interfaces, and undocumented features must be taken into account when performing the upgrade. Also, upgrades may be subject to recertification requirements.

Of special concern is software in an upgraded system. Legacy software may not be executable on upgraded replacement hardware, and may not be readily cross-compiled to the new target CPU. Even worse, timing behavior is likely to be different on newer hardware, but may be both undocumented and critical to system operation.

Design challenge:

- Ensuring complete interface, timing, and functionality compatibility when upgrading designs.

5.5. Long-term component availability

When embedded systems are more than a few years old, some electronic components may no longer be available for production of new equipment or replacements. This problem can be especially troublesome with obsolete processors and small-sized dynamic memory chips.

When a product does reach a point at which spare components are no longer economically available, the entire embedded computer must sometimes be redesigned or upgraded. This redesign might need to take place even if the system is no longer in production, depending on the availability of a replacement system. This problem is a significant concern on the Distributed example system.

Design challenge:

- Cost-effectively update old designs to incorporate new components.

6. Business model

The business models under which embedded systems are developed can vary as widely as the applications themselves. Costs, cycle time, and the role of product families are all crucial business issues that affect design decisions.

6.1. Design *vs.* production costs

Design costs, also called Non-Recurring Engineering costs (NRE), are of major importance when few of a particular embedded system are being built. Conversely, production costs are important in high-volume production. Embedded systems vary from single units to millions of units, and so span the range of tradeoffs between design *versus* production costs.

At the low-volume end of the spectrum, CAD tools can help designers complete their work with a minimum of effort. However, at the high-volume end of the spectrum the designs may be simple enough and engineering cost such a small fraction of total system cost that extensive hand-optimization is performed in order to reduce production costs.

CAD tools may be able to outperform an average engineer at all times, and a superior engineer on very large designs (because of the limits of human capacity to deal with complexity and repetition). However, in small designs some embedded computer designers believe that a superior human engineer can outperform CAD tools. In the Small system example a programmer squeezed software into a few hundred bytes of memory by hand when the compiler produced overly large output that needed more memory than was available. It can readily be debated whether CAD tools or humans are “better” designers, but CAD tools face skepticism in areas that require extraordinary optimization for size, performance, or cost.

Design challenge:

- Intelligently trade off design time versus production cost.

6.2. Cycle time

The cycle time between identification of a product opportunity and product deployment (also called Time to Market) can be quite long for embedded systems. In many cases the electronics are not the driving force; instead, product schedules are driven by concerns such as tooling for mechanical components and manufacturing process design. Superficially, this would seem to imply that design time for the electronics is not an overriding concern, but this is only partially true.

Because the computer system may have the most malleable design, it may absorb the brunt of changes. For example, redesign of hardware was required on the Mission Critical example system when it was found that additional sensors and actuators were needed to meet system performance goals. On the Small example system, delays in making masked ROM changes in order to revise software dominate concerns about modifications (and programmable memory is too expensive). So, although the initial design is often not in the critical path to product deployment, redesign of the computer system may need to be done quickly to resolve problems.

Design challenge:

- Rapid redesign to accommodate changing form factors, control algorithms, and functionality requirements.

6.3. Product families

In many cases embedded system designs are not unique, and there are a variety of systems of various prices and capabilities forming a product family. To the extent that system designers can reuse components, they lower the total cost of all systems in the product family.

However, there is a dynamic tension between overly general solutions that satisfy a large number of niche requirements, and specifically optimized designs for each point in a product family space. Also, there may be cases in which contradictory requirements between similar systems prevent the use of a single subsystem design. In the Mission Critical and Small examples different customers require different interfaces between the embedded system and their equipment. In the Distributed example regulatory agencies impose different safety-critical behavior requirements depending on the geographic area in which the system is deployed.

Design challenge:

- Customize designs while minimizing component variant proliferation.

7. Design culture

Design is a social activity as well as a technical activity. The design of desktop computers, and CPUs in particular, has matured in terms of becoming more quantitative in recent years. With this new maturity has come an emphasis on simulation and CAD tools to provide engineering tradeoffs based on accurate performance and cost predictions.

Computer designers venturing into the embedded arena must realize that their culture (and the underlying tool infrastructure) are unlike what is commonly practiced in some other engineering disciplines. But, because embedded system design requires a confluence of engineering skills, successful computer designers and design methodologies must find a harmonious compromise with the techniques and methodologies of other disciplines as well as company management. Also, in many cases the engineers building embedded computer systems are not actually trained in computer engineering (or, perhaps not even electrical engineering), and so are not attuned to the culture and methodologies of desktop computer design.

7.1. Computer culture *vs.* other cultures

A specific problem is that computer design tools have progressed to the point that many believe it is more cost-effective to do extensive simulation than build successive prototypes. However, in the mechanical arena much existing practice strongly favors prototyping with less exhaustive up-front analysis. Thus, it may be difficult to convince project managers (who may be application area specialists rather than computer specialists) to spend limited capital budgets on CAD tools and defer the gratification of early prototype development in favor of simulation.

Design challenge:

- Make simulation-based computer design accessible to non-specialists.

7.2. Accounting for cost of engineering design

One area of common concern is the effectiveness of using engineers in any design discipline. But, some computer design CAD tools are very expensive, and in general organizations have difficulty trading off capital and tool costs against engineering time. This means that computer designers may be deprived of CAD tools that would reduce the total cost of designing a system.

Also, in high-volume applications engineering costs can be relatively small when compared to production costs. Often, the number of engineers is fixed, and book-kept as a constant expense that is decoupled from the profitability of any particular system design, as is the case in all four example systems. This can be referred to as the “Engineers Are Free” syndrome. But, while the cost of engineering time may have a small impact on product costs, the unavailability of enough engineers to do work on all the products being designed can have a significant opportunity cost (which is, in general, unmeasured).

Design challenge:

- Improved productivity via using tools and methodologies may be better received by managers if it is perceived to increase the number of products that can be designed, rather than merely the efficiency of engineers on any given product design effort. This is a subtle but, in practice, important distinction.

7.3. Inertia

In general, the cost of change in an organization is high both in terms of money and organizational disruption. The computer industry can be thought of as being forced to change by inexorable exponential growth in hardware capabilities. However, the impact of this growth seems to have been delayed in embedded system development. In part this is because of the long time that elapses between new technology introduction and wide-scale use in inexpensive systems. Thus, it may simply be that complex designs will force updated CAD tools and design methodologies to be adopted for embedded systems in the near future.

On the other hand, the latest computer design technologies may not have been adopted by many embedded system makers because they aren't necessary. Tool development that concentrates on the ability to handle millions of transistors may simply not be relevant to designers of systems using 4- and 8-bit microprocessors that constitute the bulk of the embedded CPU market. And, even if they are useful, the need for them may not be compelling enough to justify the pain and up-front expense of change so long as older techniques work.

That is not to say that new tools aren't needed, but rather that the force of cultural inertia will only permit adoption of low-cost tools with significant advantages *to the problem at hand*.

Design challenge:

- Find/create design tools and methodologies that provide unique, compelling advantages for embedded design.

8. Conclusions

Many embedded systems have requirements that differ significantly both in details and in scope from desktop computers. In particular, the demands of the specific application and the interface with external equipment may dominate the system design. Also, long life-cycles and in some cases extreme cost sensitivity require more attention to optimization based on these goals rather than maximizing the computational throughput.

The business and cultural climates in many embedded system design situations are such that traditional simulation-based computer design techniques may not be viable in their current form. Such methodologies may not be cost-effective given constraints on categories of expenditures, may not be seen as worthwhile by non-computer-trained professionals, or may simply be solving the wrong problems.

Recent interest in hardware/software codesign is a step in the right direction, as it permits tradeoffs between hardware and software that are critical for more cost-effective embedded systems. However, to be successful future tools may well need to increase scope even further to include life-cycle issues and business issues.

The tutorial slide presentation presented at the conference augments this paper, and may be found at: http://www.cs.cmu.edu/~koopman/iccd96/

Acknowledgements

This work was sponsored, in part, by DARPA contract DABT63-95-C-0026, and ONR contract N00014-96-1-0202.

References

[1] Bernard Cole, "Architectures overlap applications", *Electronic Engineering Times*, March 20, 1995, pp. 40,64-65.

[2] Stephanie White, Mack Alford & Julian Hotlzman, "Systems Engineering of Computer-Based Systems." In: Lawson (ed.), *Proceedings 1994 Tutorial and Workshop on Systems Engineering of Computer-Based Systems*, IEEE Computer Society, Los Alamitos CA, 1994, pp. 18-29.

[3] *Design Automation for Embedded Systems: an international journal*, Kluwer Academic, ISSN 0929-5585.

[4] *Embedded Systems Programming*, Miller Freeman, San Francisco, ISSN 1040-3272.

[5] Daniel D. Gajski, Frank Vahid, Sanjiv Narayan & Jie Gong, *Specification and Design of Embedded Systems*, PTR Prentice Hall, Englewood Cliffs NJ, 1994.

[6] Jack Ganssle, *Art of programming Embedded Systems*, Academic Press, San Diego, 1992.

[7] Don Thomas & Rolf Ernst (eds.), *Proceedings: Fourth International Workshop on Hardware/Software Co-Design*, IEEE Computer Society, Los Alamitos CA, 1996.

[8] David Patterson & John Hennessy, *Computer Architecture: a Quantitative Approach*, Morgan Kaufmann, San Mateo CA, 1990.

[9] Philip Koopman, "Perils of the PC Cache", *Embedded Systems Programming*, May 1993, **6**(5) 26-34.

[10] Shem-Tov Levi & Ashok Agrawala, *Fault Tolerant System Design*, McGraw-Hill, New York, 1994.

[11] Daniel Siewiorek & Robert Swarz, *Reliable Computer Systems: design and evaluation (2nd edition)*, Digital Press, Burlington MA, 1992.

[12] Nancy Leveson, *Safeware: system safety and computers*, Addison-Wesley, Reading MA, 1994.

[13] Georgette Demes *et al.*, "The Engineering Design Research Center of Carnegie Mellon University," *Proceedings of the IEEE*, **81**(1) 10-24, January 1993.