Position statement for panel: "Wrong Assumptions and Neglected Areas in Embedded Research" International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC) May 6, 2008 Orlando, FL, USA

Time is a Resource, and Other Stories

Edward A. Lee *
EECS, UC Berkeley, eal@eecs.berkeley.edu

Position statement for panel: "Wrong Assumptions and Neglected Areas in Embedded Research"

I examine some misleading aphorisms.

"Computing takes time"

This phrase, though appealing, is not using the commonly accepted meaning of the word "computing." What it is really trying to say is that "computing is an abstraction of a physical process that takes time." But every abstraction omits some details (or it wouldn't be an abstraction), and one of the details that computing omits is time.

The term "computing" refers to the abstraction, not to the physical process. Were this not true, then a program in a programming language would not define a computation. One could only define a computation by describing the physical process. In my introductory programming classes (dating well back into the last millenium), I was taught that a program can be executed by stacking coke cans. The computation is the same regardless of how it is executed. This is, in fact, the essence of the abstraction.

When considering embedded systems, it is arguable that we (as a community) have chosen a rather inconvenient abstraction. Moreover, the fact that the physical process takes time is only one of the reasons that the abstraction is inconvenient. It would still be inconvenient if computing were infinitely fast. In order for computations to interact meaningfully with physical processes, they must include time in the domain of discourse.

"Time is a resource"

Computation, as expressed in modern programming languages, obscures many resource management problems. Memory is provided without bound by stacks and heaps. Power and energy consumption are not the concern of a programmer. Even when these resource management problems are important, there is no way to talk about them within the semantics of a programming language.

Time, however, is not quite like these other resources. First, barring metaphysical discourse, it is genuinely unbounded. To say that "the available time per unit time is bounded" is tautological, yet this is effectively what people say when they manage it as a bounded resource. Second, time gets expended whether we use it or not. It cannot be conserved and saved for later. This is true up to a point with, say, battery power. Batteries leak, so their power cannot be indefinitely conserved, but designers rarely optimize a system to use as much battery power before it leaks away as they can. Yet that is what they do with time.

If time is a resource, it is a rather unique resource. To lump together the problem of managing time with the problems of managing other more conventional resources will inevitably lead to the wrong solutions. Use of conventional resources is optimized. Using fewer resources is always better than using more. As a consequence, management of conventional resources is an optimization problem, not a correctness problem. Hence, there is no need to make energy consumption a semantic property of computing. This is not true of time.

"Time is a non-functional property"

What is the "function" of a program? In computation, the function is a mapping from input bits to output bits. The Turing-Church thesis defines "computable functions" to be those that can be expressed by a terminating sequence of such bits-to-bits functions, or mathematically by a finite composition of functions whose domain and codomain are the set of sequences of bits.

In embedded systems, the function of a computation is defined by its effect on the physical world. This is no less a function than a mapping from bits to bits. But as a function, the domain and codomain are not sequences of bits. Why are we insisting on the wrong definition of "function"?

"Real time is a quality of service problem"

Everybody wants quality. Higher quality is always better than lower quality (at least, under constant resource usage, creating a paradox with "time is a resource"). Time in embedded systems is not like that. Less time is not better than more time. That would imply that it is better for an engine controller to fire the spark plugs earlier than later.

Precision and *variability* in timing are quality of service problems, but time itself is much more than that. If time is not present in the semantics of programs, then no amount of "quality of service" will adequately address timing properties of embedded systems.

*This work is supported by the National Science Foundation (CNS-0647591 and CNS-0720841).