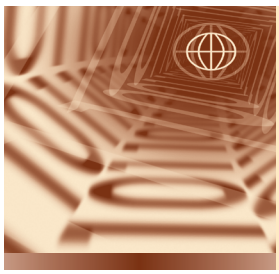
What Is Embedded Computing?

Wayne Wolf, Princeton University



Evolving from a craft to an engineering discipline over the past decade, embedded computing continues to mature.

For those who think a lot about embedded computing, as well as the uninitiated, it's important to define exactly what the term means. In brief, an embedded system is any computer that is a component in a larger system and that relies on its own microprocessor.

But is embedded computing a field or just a fad? The purpose of this new bimonthly column is to give researchers as well as practitioners an opportunity to demonstrate that embedded computing is an established engineering discipline with principles and knowledge at its core.

A LONG HISTORY

People have been building embedded systems—including complex systems—for decades. The early microprocessors were so limited that calling what they did "computing" is generous; their primary function was to manage input-output devices, and squeezing performance out of these systems required more craft than science.

By the early 1980s, however, designers were using 16-bit microprocessors to create very sophisticated applications. One example is the 68000, which car designers used to build automobile engine controls that relied on sophisticated algorithms to manage fuel and spark. Using the 68000's computational power improved fuel efficiency and reduced emissions while making the engine easier to start in various environments. Some car design algorithms used numerical methods like Kalman filters; others were more complex and used multiple modes too difficult to implement in mechanical controllers.

Laser and inkjet printers also emerged in the 1980s. Print engines require computational support for both typesetting and real-time control. First, users generate characters and lines that a computation must convert into pixels. Translating page description languages requires executing a great deal of code that performs a wide range of functions. Second, the pixels must be delivered to the printer at just the right moment; this is particularly true of inkjets.

Cell phones use coprocessors for digital-signal processing and microprocessors for button control and protocol processing. By the early 1990s, a typical cell phone contained five or six DSPs and CPUs. The telephone infrastructure also uses embedded DSPs and microprocessors extensively.

NEW APPLICATIONS

Products ranging from the multibillion-dollar B-2 bomber to home appliances or one-dollar novelties now rely on embedded computers. Personal digital assistants must support devices, operating systems, and user-loaded applications, just like PCs, but with more stringent cost and power constraints. PDA design requires careful attention to both hardware and software.

In the next decade, some microprocessors, largely invisible to users, will be used for signal processing and control—for example, to enable home networking across noisy, low-quality media such as power lines. Others will be used to create advanced user interfaces—for example, for the entire cluster of home entertainment devices.

Microprocessors have also enabled new categories of portable devices that will assume roles and perform functions yet to be determined. The cell phone and PDA combinations that hit the market in 2001 are a major step in the evolution of handheld devices. As third-generation communication systems deploy, wireless networks will incorporate multimedia—many cell phones used in Japan already have large screens and cameras, and wireless handsets will soon feature video.

AN EMERGING FIELD

Although we all use embedded devices on a daily basis, should embedded computing have the status of a separate discipline such as integrated engineering or software management? The answer is a resounding yes.

Already central to most electronic systems, embedded computers perform increasingly numerous functions that once would have been implemented in random logic and that require many algorithms too complex to build in hardwired logic. Embedded computer hardware and software are on the crit-

ical design path for many types of electronic systems.

It's tempting to believe that adding CPUs makes it easier to design and change a system, but the opposite is true. Any number of problems in an embedded computing system can result in disaster. For example,

- using an undersized hardware platform causes software design difficulties;
- bad software architecture can lead to software, performance, and power problems; and
- underestimating power consumption can reduce the entire system's effective lifetime.

Software design is always challenging, particularly when it needs to meet performance and power requirements as well as achieve functional correctness. In addition, programmability creates a temptation that entices designers to write ever more complex specifications. As our ability to fabricate and design complex systems improves, the goal posts keep moving—we must perform better just to stay even.

Without techniques to reliably bring complex embedded systems to fruition, on time and on budget, we face problems keeping pace with very large-scale integration. In conformance with Moore's law, advances in VLSI manufacturing over the next decade will require using embedded computers to build huge chips.

An embedded CPU is a predesigned block of intellectual property that can be reused many times. Embedded CPUs require memory, another reusable IP block. This reusability facilitates design management, making it possible to at least partly decouple hardware and software designs. If they can characterize IP block performance and understand IP interactions, software designers can use IP blocks as their models.

GOALS

A field is defined by its goals and matures as it articulates ways to achieve those goals. Embedded computing has five primary objectives.

Architectural design

Embedded computing seeks to design architectures that can execute a particular application's functions—protocols, signal processing, user interface, and so on—while meeting all performance and power requirements. The theoretical basis for custom software architectures evolved with real-time systems theory in the 1970s; during the past decade, hardware-software codesign developed design algorithms.

Embedded computer hardware and software are on the critical design path for many types of electronic systems.

Analysis

Embedded system designers also strive to characterize both hardware and software for performance, power, and size. This goal is clearly related to the first—we can't design an architecture until we understand the components' characteristics. Much progress has been made in the last decade on various aspects of measurement including timing analysis of embedded software and power analysis of architectures.

Modeling

To analyze embedded systems at an appropriate level of abstraction, designers must create a hierarchy of models. For example, analyzing a microprocessor's power consumption entirely at the gate level—assuming the manufacturer provides the gate-level design—is less desirable than having an abstract model that relates to the behavior of the programs running on the microprocessor. Researchers are making progress in understanding how to analyze a program without going down to the instruction level, but more work remains to be done.

Verification

Embedded system designs must meet their functional specifications as well as nonfunctional specs such as speed and power consumption. Hardware-software cosimulation tools developed in the 1990s can simulate systems at multiple levels of abstraction, which greatly speeds up simulation throughput. However, we need more advanced formal methods to complement simulation.

Application orientation

Developing architectures for common applications—wireless, video, networking, and so on—is another function of embedded computing. This involves adjusting to design tradeoffs that result from the ability to fit more processing power onto a single chip. Also, knowing just what state-of-the-art technology is can be difficult because developers often do not disclose novel designs until long after they create them. We are making substantial progress in understanding the basic shape of architectures suitable for many of the major application areas in embedded computing.

This column will consider both the diversity and unity of embedded computing in hardware, software, and applications. We'll look at particular domains—control, communications, and so on—to discuss the problems that embedded system designers encounter, but we'll also step back and examine broader principles.

I have my own opinions on embedded computing, but various guest columnists from both industry and academia will also share their expertise in this column. I look forward to engaging in a dialogue with you in future issues of *Computer*. ■

Wayne Wolf *is a professor of electrical engineering at Princeton University and CTO of MediaWorks Technology. Contact him at wolf@princeton.edu.*