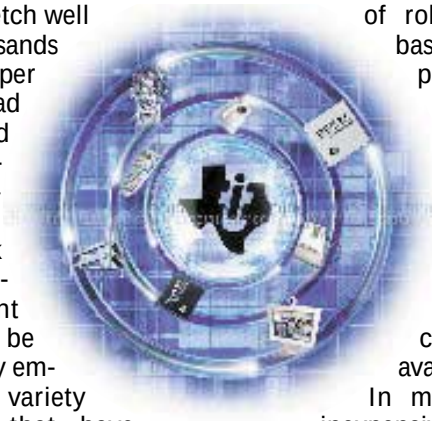
Open, Extensible DSP Systems: Leveraging Existing Software

By Steve Blonstein

DSPs are an essential element of many modern-day products like voice and telephony systems and even white goods. It wasn't that long ago that these systems were built from the ground up. With program sizes on the order of thousands of lines of code, that was an acceptable approach. However, as DSP program sizes stretch well past tens of thousands of lines of code per system—and head rapidly toward hundreds of thousands of lines—the old approach just won't work anymore. An alternative development strategy must be used. This strategy employs leveraging a variety of components that have already been built, debugged, and tested. It also means not rewriting any piece of code that can be purchased for a reasonable price on the open market. Your organization may not use this strategy and attempt to reinvent the wheel, but your competition won't hesitate to use it and leverage already existing components. The results are likely to favor your competition, especially in terms of time to market, robustness, and overall quality.



The most important question is how and where your organization can add unique value to the final system. Let's assume that it has DSP programming skills or can readily obtain them. Then the decision moves to where in the software "food chain" it should enter the process.

At the bottom of the chain is a kernel suitable for running and managing the overall DSP system. Virtually no organization can justify reinventing the wheel here. Using an off-the-shelf kernel is best in terms of robustness, installed base, simplicity, and proven performance.

At the next level are infrastructure components, such as drivers, math libraries, and communication stacks. Again, almost all of those components are available off the shelf. In most cases, they're inexpensive (or free), they're highly optimized for the architecture, and they make absolutely no sense to duplicate.

The next level up is the set of algorithms that the system needs to run. At this point, the choice becomes a little more interesting. There are cases in which the organization provides its own unique value and differentiation in the form of proprietary algorithms. These algorithms, therefore, must be created by the organization itself. However, there are also many "commodity" algorithms, such as modem, JPEG, and MPEG algorithms, that are commercially available. This point is where some of the tougher decisions must be made. Can the organization really save time and money by trying to build algorithms that are readily available? Does the organization add value or even truly have the skill set to build algorithms outside of its core competency area? Typically, it won't be able to add much value to something already written and available—so that means purchasing these components.

At the highest levels of the food chain are application-specific frameworks and the application code itself. Most likely, these levels are where an organization can, and probably should, add its unique system knowledge, which can't be bought on the open market.

Off-the-shelf components will work well only if they're designed to be modular and reusable—and if the system is designed to accept such components. For the system to accept such components easily, it too should be modular, scalable, and extensible—the very definition of today's open systems. To remain competitive, organizations must use and leverage open systems at every level of the process up to, and even including, the point at which they add their own unique value and differentiation.

Steve Blonstein is a technical director in Texas Instruments, Inc.'s Software Development Systems group in San Jose, Calif.