

A person is holding a white sign in a boxing ring. The sign has the text "ROUND 2" in large, light blue, outlined letters, with "ROUND" arched over the number "2". Below that, the text "HANDS-ON PLATFORMS" is written in smaller, solid blue, sans-serif capital letters. The person's hands are visible at the bottom of the sign. In the background, a boxing ring with red and blue ropes is visible, along with a metal truss structure and several spotlights hanging from the ceiling.

ROUND
2

**HANDS-ON
PLATFORMS**

EMBEDDED PLATFORMS: A SECOND LOOK

PROCESSING PLATFORMS AND THEIR SOFTWARE-DEVELOPMENT ECOSYSTEMS ARE BECOMING MORE IMPORTANT FOR THE SUCCESS OF DESIGNING INCREASINGLY COMPLEX APPLICATIONS.

BY ROBERT CRAVOTTA • TECHNICAL EDITOR

During the last quarter of 2002, I did a hands-on project with Texas Instruments' OMAP (Open Mobile Application Processor) platform using the Innovator development kit. Although the article that resulted from that effort shared my experiences with the platform and the development ecosystem that TI had built around the platform, the article also touched on the issues of working with preproduction silicon and software; in essence, it was an early-adopter story ([Reference 1](#)).

Fast-forward to October 2005; TI announced its Davinci technology products and announced working product demonstrations within months of the rollout of the technology. The Davinci devices share a similar architecture with the OMAP devices in that they both integrate an ARM core with a TI DSP core in a single device. Doing a hands-on project with the new platform presented an opportunity to give a second try at developing on a similar complex-processing platform and to see how the development ecosystem that supports it had evolved over the past few years. Complex-processing platforms are not unique to TI, so, for the project, I also looked at platform offerings from other semiconductor companies to identify emerging trends for developing with these complex-processing platforms.

One thing that I learned from this exercise is that, although the silicon integration is a big part of the marketing focus for these platforms, the software assets and the tools that the platform provider provides are overshadowing or will overshadow the silicon to gain that much-sought-after design win. This statement is not an attempt to diminish the value and complexity of the hardware integration but an acknowledgment of the growing importance of the software assets and tools. This software simplifies the growing complexity of the software effort as a key differentiator between silicon offerings that provide increasingly similar integration and capabilities.

According to iSuppli (www.isuppli.com), by 2010, the total semiconductor market, excluding memory, analog, and

display drivers, will reach \$163 billion from a bit more than \$108 billion today, and about 88% of this market will be in processor-based offerings (Reference 2). All of those processor-based designs will need software. Gene Frantz, principal fellow and business-development manager for DSPs at TI, proposed in a recent blog entry that “the new twist on SOCs (systems on chips) is that the S is more likely to stand for “software” than “system.”

According to Jay Gould, product-marketing manager for embedded processing at Xilinx, “The number of software engineers on a given project may outnumber the hardware engineers by a factor of two to five or more.” Ata Khan, director of product innovation for the standard-IC-business line of NXP Semiconductors, and D Chris Baumann, director of BiCMOS products at Atmel, both point out: “For complex-processing applications, our customers’ software development makes up 70 to 80% of their project’s total schedule and budget resources.” This percentage translates to a factor of 2.5 to four software-development resources for each hardware-development resource for these complex-processing applications.

The increasing focus on software is partly due to the fact that, with each advancement in processor technology, more software-instruction cycles are available to perform new value-added processing per sample period for a given application. For example, according to Frantz, in 1982, telecom systems could leverage 625 instruction cycles per sample period with a 5-MIPS processor. Today’s modern DSPs can deliver hundreds of thousands of instruction cycles per sample period. He goes on to point out similar progressions, albeit with fewer instruction cycles per sample period, for audio and video applications.

The flexibility that software affords is a primary mechanism that these platforms offer to developers to add differentiating features and capabilities to their designs. Another contributor to the increasing resource allocation of software for embedded systems is based on an axiom I learned and adopted as an embedded-software and -system designer. For any problem, regardless of its root cause, if the software can detect the condition, can adjust for it, can minimize the effects of it, or even correct the condition, then, it is by defi-

AT A GLANCE

- Complex-processing platforms are relying more on their development-support ecosystem for their adoption and success.
- Each iteration of platform offerings incorporates the lessons developers learned from previous iterations.
- Providing a development ecosystem is a huge effort, involving the growing pains of providing software assets for previously unsupported operating systems.

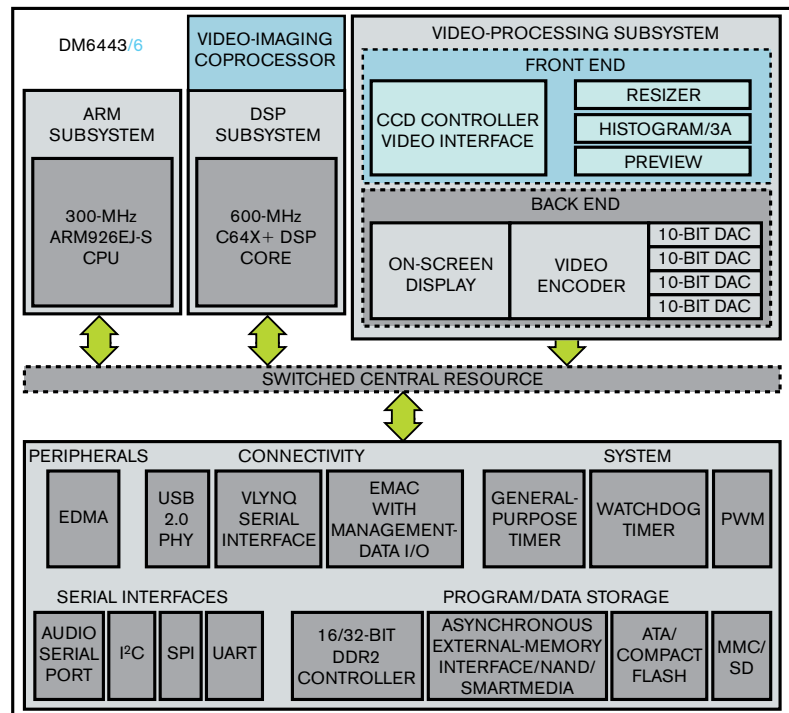
nition a software problem. This axiom derives its truth from the fact that, in addition to being more efficient at performing complex serial operations than a pure-hardware implementation, software is almost always easier, faster, and cheaper to change than hardware, once the hardware exists.

The consequence of this axiom is that most faults that manifest themselves during the late stages of a design project directly flow into—and negatively impact—the software-development effort.

This fact increases the complexity of the software far beyond just meeting the original product specifications, because the software must now accommodate unanticipated environmental and manufacturing variables without the need for changing the hardware if possible. The software-development team’s success at accommodating the challenges of this axiom often represents the difference between successful and unsuccessful designs.

DAVINCI VERSUS OMAP

The Davinci and OMAP devices share some similarities. They both employ a heterogeneous-processing architecture that combines an ARM core with a TI’s DSP core. They both support standard APIs (application-programming interfaces) that encapsulate and isolate the development and execution of the code residing on each core. For both device families, application programmers can use tools they are familiar with that target the ARM core; application programmers need not become DSP experts to take advantage of the integrated DSP core. Likewise, DSP



NOTE: ■ = DM6446.

Figure 1 The Davinci devices have two processor cores and many storage, connectivity, and media-related peripherals.

programmers can use tools they are familiar with for the DSP core without learning about the ARM core. The Davinci and OMAP devices both use the eXpressDSP-compliant algorithms that are part of TI's third-party network, and they both support a primary execution model that places the ARM code in a master role and the DSP code in a co-processor or slave role.

However, the Davinci technology differs from the OMAP platform in that Davinci targets digital-video and -audio applications rather than mobile applications, and it is an evolutionary step beyond OMAP with regard to software-development support. By virtue of their different target applications, the two platforms use different DSP cores. The OMAP employs power-efficient C55x DSP cores, whereas the Davinci employs high-performance C64x DSP cores. Also, OMAP devices use an ARM925 core, whereas the Davinci devices use an ARM926 core. Both sets of devices integrate different peripheral sets that are appropriate to their target-application domains. The integrated peripheral set in Davinci devices includes video-processing subsystems, serial and connectivity interfaces, and several memory and storage interfaces (Figure 1). The differences in the target-application domains drive these differences between the two platforms.

The Davinci platform builds on the lessons TI learned, both technically and businesswise, from the OMAP systems. For example, the Davinci platform extends the xDAIS (eXpressDSP Algorithm Interoperability Standard) with the xDM or xDAIS-xDM (xDAIS for digital media) interfaces. This interface affects each programmer's role—an application role for the ARM and a signal-processing role for the DSP—for a Davinci or an OMAP system; however, the interface is the mechanism that application programmers use to load, unload, and use the algorithms loaded on the DSP core. An important goal of xDM is to enable a plug-and-play architecture for multimedia codecs across various classes of algorithms and sources, or vendors. It defines common status and parameters for each class of multimedia codecs to facilitate replacing one codec with another without requiring a change to the application source code;



Figure 2 The Davinci evaluation module includes a camera, an LCD, an IR remote, speakers, and a hard-disk drive, as well as demonstration and development software.

only the configuration code for the codec needs to change.

The xDM standard defines and adds a uniform set of APIs for four codec classes—video, image, speech, and audio—to the xDAIS specification. The xDM's eight generic interfaces consist of an encoder and a decoder for each of the four codec classes. The xDM also supports extensions for requirements such as metadata parsing, file format, and custom processing. The xDM APIs add two new functions, `process()` and `control()`, to the xDAIS-defined set of functions. The call order of the interface-function calls is similar in the xDAIS and the xDM components except that xDM removes the `algControl()` method. When you use the xDAIS algorithms with xDM algorithms, the xDM `control()` method substitutes for the `algControl()` method.

Unlike the rollout of the OMAP, the rollout of the Davinci evaluation module includes three prewired DSP/codec combinations with which developers can work. The inclusion of the prewired codecs allows developers to evaluate the functions of the system without waiting for the implementation of production-supported codecs. The evaluation module comprises a kit that includes a camera, an LCD, an IR remote, speakers, and a hard-disk drive (Figure 2).

The kit includes working demonstrations already stored on the hard drive to provide immediate access to exercising the encoding, decoding, and networking capabilities of the system.

Another difference between the rollout of the Davinci and the OMAP platforms was the immediate availability of chip- and board-level development resources for the ARM core. The Davinci evaluation module also included development tools and documentation to get started and work with the demonstrations. TI was able to include these components in part by limiting the development environment to Linux. Development tools for the DSP core are part of the Code Composer Studio tool suite. TI recently released two tools to support development with Davinci devices in late September 2006. The eXpressDSP configuration tool assists developers in integrating xDM codecs into the codec engine; formerly, the system integrator manually performed this process. The other new tool, the minimally invasive TMS320DM644x SOC visual analyzer, captures and displays data for both the ARM and the DSP cores on a single time line for a system view of the application behavior.

No amount of documentation can substitute for working directly with these complicated platforms: This proj-

ect illustrated some disconnections between expectations and reality. Using a lesson I learned from the previous platform hands-on project, I went to a Davinci technical seminar. I thought the full-day session would be a hands-on workshop, but I was wrong. (The first hands-on technical workshop for Davinci would not occur until after I had finished this project and just before the print date of this article.) The technical seminar presented the technical and business aspects of the platform, but there was no hands-on portion. I was surprised by the number of Linux-implementation details the seminar presented. Later, when I had a Davinci evaluation module in my office to work with, I figured out why.

The first half of the seminar presented a nice overview of the system components and each of the peripherals, particularly some coding examples the video-processing subsystem uses. We learned about the Davinci Framework API and the high-level portions of each of the processing layers: application, signal processing, codec engine, and third-party software. The overview also covered the internal and third-party development tools that support each programming layer. The rest of the day consisted of third-party authorized-software-provider presentations.

I received a Davinci evaluation module to work with a few weeks later. It was a quick and easy process to connect all of the parts of the system and get the demonstrations operating correctly. My ability to get the demonstrations running quickly was due in no small part to the fact that all of the software resided on the hard drive. The nice thing, though, was that I could confirm that each component and interface was properly connected and operating correctly before setting up my workbench.

I received an unexpected shock when I went to set up my work space. I have done plenty of cross-platform development. For this project, I knew I would be developing to a Linux target. However, I didn't know that I would be working on a Linux-development host; I had assumed that I would be able to work on a Windows-development host. All of the initial tools for the evaluation module operate only on a Linux host. Montavista supplied the develop-

EMULATING LINUX WAS NOT A DEBILITATING EXPERIENCE, BUT THERE ARE DIFFERENCES IN ITS BEHAVIOR FROM WINDOWS APPLICATIONS.

ment tools, so I checked Montavista's Web site to confirm that it supported host environments besides Linux. The company's development tools support Linux-, Solaris-, and Windows-development hosts.

TI's support personnel told me that the Davinci tools supported only a Linux-development host. I did not want to install Linux on my computer. This issue had nothing to do with a dislike of Linux; after all, Linux was to be the target operating system. Rather, it had everything to do with having to expend time, energy, and thought on a low-value effort for the project. I had no plans to continue using Linux on my host system, and I needed to fully understand the Linux configuration for the target. However, I lacked the time to relearn my host computer's operating system. Working in a Linux host environment increased my learning curve.

Fortunately, TI support shipped me a Red Hat Linux image and granted me use of a license for this project, so that I could emulate Linux on my computer with VMware player. This help allowed me to avoid manually setting up my computer for dual boot and to more quickly continue with the project.

I know how to find the data files and applications on my computer running Windows. The applications behave and interoperate as I have come to expect. I have years of lessons learned in how those applications behave. Emulating Linux was not a debilitating experience, but there are differences in its behavior from Windows applications. As an example, I used the supplied Gnome Ghostview to read the supplied Adobe Acrobat documents. I never could figure out whether I could perform a simple text-string search within the browser. I could have searched the Internet for a different browser, but why should I have to do that when I already have a perfectly acceptable browser?

The evaluation module supports only a Linux host because the Davinci-tools team wanted the tools to be available

at the same time the evaluation module became available. Also, TI expected most of the early adopters to use a Linux target and would have experience with Linux as a development-host environment. The TI tool group acknowledged that support for other host-development operating systems is a next step. As it turns out, using Linux was also a significant learning curve for the TI support team because it had not worked with it before and because the team needed to be proficient with the system before the company started shipping the Davinci evaluation module to customers.

The project plan was to use the encoding-, decoding-, and networking-demonstration code to make a crude video phone. One problem was that I had only one Davinci evaluation module, so the demonstration could not perform in real time; I would have to simulate one end or the other of the system on each run. Obtaining the first evaluation module was a challenge; TI had only a limited supply. And there was not enough time to obtain a second module. Another problem was that the encoding and decoding examples that came with the module processed either speech or video data, but not both, so I had no example of how to synchronize the two. In my conversations with TI's tools group, I learned that the company was looking at the GStreamer media-processing library as a framework to help with audio and video synchronization and other capabilities. GStreamer sits above the level of a normal library. Late in the project, I started working with Ittiam using its video-phone demonstration.

For this project, I also tried out Green Hills Software's Probe and Multi development environment for Davinci. The Probe is a hardware-debugging device that connects to Davinci. Even though Green Hills Software's Multi normally supports development on Windows, Linux, Solaris, and HP-UX host systems, the Davinci tools operated only under Linux during this project. The Multi integrated development envi-

ronment supports the needs of each of the programming layers: application, DSP, and system. With the Probe, Multi provides visibility to both the ARM and the DSP cores, as well as supports Linux-kernel awareness. I was able to trace directly into the Linux kernel by building the Linux-kernel image with debugging information turned on and then translating, with Green Hills' dblink tool, the dwarf debugging information that GCC (GNU Compiler Collection) generated to a debugging format that Multi understands.

Multi separates process debugging into windows, each with its own background color; this feature is useful when using process-context breakpoints, which stops execution of the code only when the breakpointed line of code is executing as part of the specified process. I was also able to use the Time Machine capability; after you stop the processor core, it lets you step the instructions backward and forward. Green Hills recently added an always-on feature to Time Machine, which greatly increases its usability and helps developers because they no longer need to remember to turn on the Probe to capture an event.

BEYOND THE PLATFORM

Another difference between the ecosystems for the Davinci and OMAP platforms is that TI now offers to act as a first-tier point of contact for technical support and licensing of third-party hardware and software IP (intellectual property). This move simplifies the design team's efforts to acquire and use third-party IP and enables the support team to track problems and issues and more quickly share relevant information among those groups with similar issues. It also enables TI to more easily see trends in technical and business challenges, as well as requests for feature support, so that the company can react quickly to emerging opportunities.

The Davinci evaluation module includes another less obvious difference from the OMAP development kit that is consistent with a growing trend in embedded designs. Both platforms are heterogeneous, multicore systems. However, the Davinci module includes yet a third different processor core on the board to perform support functions: an

MORE AT EDN.COM

+ For a related article on using reference-design resources, go to www.edn.com/article/CA512130.

+ Find another article by Technical Editor Robert Cravotta at www.edn.com/article/CA608178.

+ Learn about abstraction tools at www.edn.com/article/CA426084.

ultralow-power, 16-bit MSP430 RISC mixed-signal microcontroller, which operates with and controls the nine LEDs, IR interface, and real-time clock on the system board.

Employing multiple processing architectures is a growing trend for complex embedded-system applications. Examples include NXP's Nexperia platform, which combines MIPS cores with TriMedia cores, along with hardware accelerators and media-processing peripherals. The Nexperia platform employs an API analogous to the Davinci and OMAP platforms. NEC Electronics' EMMA (Enhanced Multimedia Architecture) platform features devices with as many as 16 processor cores, including 32-bit RISC, 32-bit RISC with DSP, and 64-bit RISC architectures, along with hardware accelerators and stream processors in the same device. An increasing number of tools simplify or help automate the creation and use of custom hardware accelerators and coprocessors for use alongside application processors and DSPs (references 3 and 4).

The continuing growth of complex embedded designs using multiple heterogeneous-processing architectures in a single design represents an area of opportunity for software-development tools to assist in system-level partitioning, identifying and implementing concurrency in software, and verifying operation and interoperability of all the processing engines. It also represents a glimpse at how the industry might be able to finally build and distribute reusable software components because developers could design each processing architecture or core to limit interaction with software on other cores. Platform providers can more safely provide commodity functions implemented as software on dedicated processors by locking

access to those dedicated processors for all but those customers willing to risk breaking the encapsulation.

Mechanisms that support easier reuse and reliable interoperability of software components are critical capabilities for abstracting and scaling software complexity. To date, efforts to accomplish this goal have met with limited success, but domain-specific organizations are trying to specify and create a working model for interoperability between software components. Examples of such organizations are CE Linux Forum, the Digital Living Network Alliance, and the SDR Forum.

As the complexity of these processing platforms continues to increase, the support ecosystem for them will be more critical to the success of both the platform provider and partners and the design teams that use these platforms. A key concept that these early ecosystems are demonstrating is how a wide audience of application-level designers can rapidly and safely leverage and incorporate the work of a few expert users of a processing architecture in the context of a domain problem. **EDN**

REFERENCES

- 1 Cravotta, Robert, "Forge Ahead?" *EDN*, March 6, 2003, pg 50, www.edn.com/article/CA278834.
- 2 "Market Forecast Database," iSuppli, June 2006, www.isuppli.com.
- 3 Cravotta, Robert, "EDN hands-on project: Accelerate your performance," *EDN*, Nov 11, 2004, pg 50, www.edn.com/article/CA476908.
- 4 Cravotta, Robert, "EDN hands-on project, part 2: Automate your acceleration," *EDN*, Dec 7, 2004, pg 55, www.edn.com/article/CA484490.

FOR MORE INFORMATION

You can find a list of relevant vendors and organizations at the Web version of this article at www.edn.com/061109cs.

You can reach
Technical Editor
Robert Cravotta
at 1-661-296-5096
and rcravotta@edn.com.

