

CONTROL AND SIGNAL PROCESSING: Can one processor do it all?

INTEGRATING CONTROL/SIGNAL PROCESSING IN ONE SYSTEM IS ABOUT MORE THAN TECHNICAL PERFORMANCE. TOOLS, CULTURE, LEGACY AND ENGINEERING EXPERIENCE ALSO PLAY BIG PARTS.

THE TREND TOWARD AGGREGATING multimedia applications into a single device supporting high-speed wired and wireless communication is driving a tighter coupling of control- and signal-processing requirements into contemporary system designs. On the other hand, primarily control-oriented applications, such as digital-

motor control, robotics, and hard-disk controllers, are benefiting from access to DSP operations that speed math-oriented processing and allow more precise and sensorless control designs. Although control-oriented applications typically need fast interrupt response and an ability to efficiently process bit-level control and status information, signal-processing applications require continuous number-crunching performance with some limited I/O capabilities. The proliferation of systems that can benefit from mixed control and signal processing have spurred numerous competing architectures to meet designers' needs.

These new system designs require quicker implementation of both real-

time control and signal processing at a lower cost than did previous designs. The potential advantages of a single device's performing real-time control and signal processing are reduced board space, lower system power consumption, lower system cost, more headroom for additional functions, and simplified system development. However, with device densities and resource integration increasing, a single device is not restricted to a single increasingly complex processing core. Rather, it can consist of multiple homogeneous or heterogeneous processing cores. Unfortunately, integrating multiple processors onto a single device does not greatly simplify software complexity. Therefore, it is important to consider

<i>At a glance</i>	64
<i>Different purposes</i>	65
<i>Optimizing your C</i>	66
<i>For more information</i>	68

how using single-threaded rather than multithreaded software will affect your design in mixed control/signal-processing applications.

Among the ways to implement both signal and control processing within your system, using a dedicated microcontroller or DSP for all your processing needs has, until recent years, been a widely available single-device option. Since then, many microcontroller manufacturers have added to or extended their architectures to include DSP functions, such as MAC (multiply-accumulate) instructions. Likewise, some DSP architectures have incorporated features such as integrated peripherals, programmable external chip-select lines, interrupt-driven I/O, timers, and larger external-memory ranges.

Viewing these extended processors as stopgap measures to address an industry-capability inflection point, processor vendors, such as Analog Devices with Blackfin, Infineon with TriCore, Micro-

AT A GLANCE

▶ Signal processing is finding its way into many applications, including traditionally control-oriented ones.

▶ Making C compilers efficient for DSPs and making engineers accept them will increase the number of applications relying on both control and signal processing.

▶ Your legacy code and the tools you are familiar with strongly influence your choice of unified or heterogeneous processors.

▶ Tools need to continue maturing to bridge engineering experience and the increase in mixed control/signal-processing applications.

chip with DSPic, Motorola with DSP-56800, and STMicroelectronics with ST100, have developed unified architectures, or microsignal processors, to bet-

ter optimize control and signal processing in one instruction engine. These architectures go beyond just tacking on a MAC-execution engine and include other DSP capabilities, such as multiple bus and memory structures and special-address generation. However, these blended processors must often yield to a pure DSP implementation for the highest performance signal-processing applications.

Until recently, integrating a dedicated microcontroller and a DSP into your design has meant using multiple devices or building your own integrated device. Standard dual-heterogeneous processor devices are now available; Texas Instruments' C5470 integrates a C5000 DSP and an ARM7 processor core into one device. Another way to address mixed control/signal processing is to integrate dedicated hardware accelerators and peripherals with your processor core. Each of these approaches address mixed control/signal processing to varying de-

DIFFERENT PURPOSES

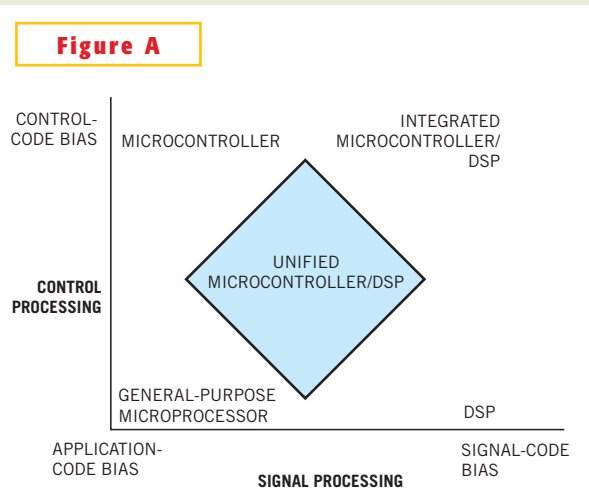
To understand why basic processor architectures coexist and whether they will converge or remain viably separate, look at

the characteristics of each (Figure A). General-purpose microprocessors sport the fastest clock rates and rely heavily on

large external memories and controllers. They run large, complex operating systems supported by a mature development-tool infrastructure, and significant reusable legacy software is available. These processors are central to desktop computers and strong for general data manipulation. Because they rely on external devices to interact with the real world, they are stronger for user interfaces and application hosting that can communicate with and manage external components for control and signal processing. They are performance-biased and less sensitive to size, heat dissipation, power consumption, and cost than embedded processors.

Microcontrollers are control-biased processors that integrate on-chip standardized peripherals, device controllers, and memory to drive down system cost for a class of applications. General-purpose processors

exclude these components to maintain higher system flexibility. Microcontrollers are specialized to use interrupt processing to quickly sense and react to external events appropriate for real-time control applications. They differentiate themselves through features such as processing performance, integrated peripheral sets, on-chip memories, memory management, packaging, power management, and development support. They are more cost-efficient than general-purpose microprocessors because they optimize board space, code size, and power dissipation for their target embedded applications. These primarily control-oriented systems may have DSP operations, such as MAC (multiply-accumulate) instructions, but they view DSPs as math coprocessors in otherwise-microcontroller-based architectures. They can run RTOSs that are smaller and better tuned for



A conceptual mapping of processor architectures, based on the scale of control- and signal-processing requirements for an application, illustrates which applications a general-purpose controller, signal-processor, unified control/signal-processor, or dual-heterogeneous processors is best for. Those applications furthest out on an axis may require multiple processors or arrays of processors.

grees and with different trade-offs (see sidebar “Different purposes”).

As software development and maintenance consumes an increasing share of a design’s life-cycle time and budget costs, software complexity directly drives a design’s time to market and ability to adjust to market shifts. The speed of robust software development, ease of maintenance, and simplified reuse in succeeding design generations are critical considerations in choosing a processor approach. Using a single instruction engine is simpler than using multiple instruction threads, especially heterogeneous ones. But other considerations beyond application-performance requirements—such as tool maturity, code legacy, and your engineering team’s experience with an architecture—can drive your processor decision.

LOOKING FOR COMMON GROUND

Designers tend to choose components they are familiar with to minimize a pro-

ject’s risk. An engineer will favor a microcontroller that has the necessary features and memory to perform control tasks and provide enough headroom to satisfy the signal-processing requirements, especially if he or she is familiar with the development tools or the processor. Likewise, a DSP-software engineer will include control code in a DSP instruction stream as long as it does not compromise the signal processing. A designer will consider an unfamiliar unified processor or a complementary DSP or controller only if the project’s control/signal-processing requirements exceed the preferred architecture’s capabilities. What makes a DSP or microcontroller unfamiliar to a software engineer?

Software engineers working with microcontrollers develop an expertise in sequential processes, usually program in C, work with mature and complete development tools, rely on an RTOS to abstract and manage the low-level interaction with the hardware, and have access

to a variety of reusable software modules. In contrast, software engineers working with DSPs develop an expertise in algorithmic processes, usually program in assembly language, work with younger tool sets than microcontrollers, are intimately familiar with the low-level hardware, cannot afford inefficiencies from using an RTOS, and generate the signal-processing code on a project-by-project basis. DSPs have multiple buses and memories and many specialized registers that complicate compiler-code generation; microcontrollers have more general-purpose, register-friendly architectures. Control processing is characterized by burst processing; signal processing is continuous and sustained. Additionally, signal processing requires more knowledge of mathematical techniques, such as DFTs, FFTs, digital filtering, and convolution. These differences were tolerable while signal processing remained a niche discipline, but signal processing is moving into the scope of many more appli-

real-time response than general-purpose operating systems. RTOSs provide a level of hardware abstraction that supports robust and mature development tools and a pool of reusable legacy software.

DSPs continuously, repetitively, and as quickly as possible perform as many arithmetic operations as possible to satisfy the specialized processing requirements of world signals, where microcontrollers and conventional analog components are less cost-effective. DSPs differ from microcontrollers in that they include support for fractional number types; multiple bus and memory structures supporting simultaneous memory operations for single-cycle MAC execution; specialized address generation for arrays, circular buffers, and bit-reversed algorithms; specialized registers to minimize memory accesses; and zero-overhead looping. This specialized structure keeps the com-

putational units fed with new data to process but sacrifices the code size, efficiency of context switching, and branch/flow processing that is stronger in microcontrollers. DSPs differentiate themselves by number-crunching performance, footprint size, power consumption, and development support. Unlike microprocessors, RTOS support is practically nonexistent, because the abstraction of the low-level hardware is anathema to a programmer trying to squeeze the highest performance out of a DSP. Because of pervasive low-level assembly-programming practices, much less legacy software is available for reuse on DSPs than on microcontrollers. Contemporary DSP architectures support more efficient compiler-code generation and help encourage a growing body of reusable legacy code.

Using a microcontroller for signal processing or a DSP for control-oriented applications is

less than optimal, but devices that incorporate features of both architectures lend themselves to light duty in the other segments. Unified control/signal processors have cores that use a register-based programming model of a microcontroller, can execute and scale to multiple single-cycle MAC operations, support specialized address generation, support multilevel interrupt priorities, and stress C-compiler efficiency for both control- and signal-processing code generation. Using a single-instruction thread simplifies programming for tightly coupled control- and signal-processing streams. These processors rely on RTOSs for task priority and latency management. Legacy control code is available, but it is still unclear how unified instruction threads affect the reusability of intermixed control/signal code.

Integrating a separate DSP and microcontroller on the same device yields significant system

savings, but it does not eliminate the need for interprocessor communications that a unified processor simplifies. Deriving interprocessor communications has been a project-to-project exercise; however, these dual-core devices provide standardized links and shared resources that can lead to reusable interprocessor-communication code. This configuration is strong when the control- and signal-processing requirements are both significant and loosely coupled. Generally, the microcontroller initializes and supervises the DSP, directs host communications, and manages the user interface, and the DSP concentrates on the signal processing. Each processor in the device can operate at the appropriate clock speed and can tap into its own available legacy code. Although microcontrollers can benefit from RTOSs, there is still little RTOS support for DSPs.

cations that will require more software engineers to be familiar with signal processing. Unfortunately, most software engineers are more familiar with microcontrollers than with DSPs.

Many DSP vendors are investing significant resources in their contemporary DSP architectures to bring DSP development tools on par with microcontroller tool sets and expand the number of engineers that can effectively program DSPs (Reference 1). Although these architectures prioritize the need for efficient compiler-code generation, technical, business, and cultural barriers minimize the adoption of C coding for signal processing.

DSP compiler code is considered good if it can reach 80% efficiency when compared with hand-optimized assembly (see sidebar “Optimizing your C”). This practice conflicts with DSP-programming practices that push the mathematical processing to the processor’s limits and cannot afford such inefficiency. Complicating compiler optimizations, the standard C specification does not accommodate key DSP architectural features, such as fixed-point arithmetic, divided memory spaces, and circular buffers. Compilers can support some DSP characteristics, such as hardware loop registers and constraints on the register set, without inclusion in the C specification, but examining three approaches to implement fixed-point arithmetic in C illustrates the trade-offs and impact to compiler-code efficiency and the software engineer’s effort. One method—mapping fixed-point types onto integer

types—prevents the compiler from optimizing fractional arithmetic and places the burden on the programmer to keep the notation unambiguous and to manually manage the scaling. Another method uses abstract data types to encapsulate fixed-point types, but the function-call syntax can complicate arithmetic expressions and incurs non-value-added-procedure overhead. Last and critical for the compiler to provide optimizations, a language extension can add the fixed-point data type to the specification.

Proprietary intrinsic functions provide a mechanism for language extensions, but they complicate portability issues because they are architecture- and compiler-specific. The ISO (International Organization for Standardization) is considering the DSP-C specification to standardize language extensions, and a number of DSP vendors have adopted the specification for their own processors and tools (Reference 2). Unfortunately, specialized DSP architectural optimizations are not universal, and standardized extensions will not eliminate the use of proprietary intrinsics to accommodate them.

Aside from whether a compiler can produce efficient code, a more subjective but significant barrier to adoption of compilers for DSP programming is the inertia of the prevailing engineering culture. Assembly programming is the status quo, and compiled code needs to prove itself against the perceptions and prejudice that earlier DSP compilers and tools acquired. If signal processing were

not expanding into so many diverse applications, the cultural barrier would be insurmountable. Fortunately, 20% of the embedded code often accounts for 80% of the execution time. Focusing your low-level, assembly-language talent at this critical 20% of the code opens the door for compilers and development tools to address the remaining 80%, helping to balance available experienced DSP software engineers, time to market, ease of use, and maintainability.

TO SINGLE-THREAD OR MULTITHREAD?

Software is a critical-path item in embedded designs. Software engineers view the system differently from hardware engineers in that they target an instruction set, an RTOS, or both. Using different languages, or even just variants of the same language, adds to the software complexity and affects function partitioning and software reuse. With software making up an increasing share of development time and budget, it is critical to consider whether a project should use one or two software-development teams. You also need to consider whether one development tool set is sufficient how the control and signal-processing tasks communicate and synchronize how the project can leverage legacy code; and the challenges to system integration and debugging.

Generally, but especially for systems with no legacy code to contend with, single-threaded software development is easier than multithreaded software development. Unified architectures present an incremental increase in processor fea-

OPTIMIZING YOUR C

With many contemporary DSPs, the designers considered architectural features to make acceptable code from a compiled C source code more realistic. C provides better portability between hardware platforms and permits more code reuse, resulting in shorter development times. Using language extensions and intrinsic functions goes a long way in assisting compiler efficiency, but intrinsics can negatively affect portability. Here are some sug-

gestions for how you can structure your signal-processing C code to assist the compiler efficiency.

Look at memory as a predominant performance bottleneck. Although processor performance increases annually by 40 to 50%, memory performance increases closer to 5% in the same time. Locate the most frequently executed code and data in on-chip memory. Explicitly declare frequently used variables as “register.” Reuse

local variables declared as register for multiple nonconflicting variables; doing so reduces stack operations but makes the code less readable. Declare local variables as global or as static so they are located in the heap and avoid slow, indirect accesses to and from the stack.

Explore declaring functions as inline while balancing between accessing memory and reducing processing overhead. Try to avoid test and branch operations in a loop by splitting the

loop into multiple instances for each separate condition. Check your compiler’s performance for loop efficiencies when using the “do,” “do-while,” and “for” loop constructs. Many DSPs have hardware loop registers that the compiler may use with one construct but not another. Try to replace division operations with multiplication by the reciprocal to take advantage of hardware multipliers, but beware of losing precision in the conversion; it may impact your algorithm.

tures that make it easier for designers to ease into signal processing with formerly control-only applications. Unified instruction engines give engineers immediate access to both control and DSP functions under a single memory map and a single set of tools and reduce asynchronous threads that simplify inter-process communication. Data and time dependencies are easier to see in a single-threaded implementation. An RTOS plays an important role in single-threaded implementations, managing scheduling for high-latency/low-priority through low-latency/high-priority tasks. Unified instruction architectures simplify system integration, because all of the software targets the same instruction set, and one tool set supports all debugging.

Dual-heterogeneous core systems are more complex than unified architectures. Designers must obtain and learn two languages or variants as well as the idiosyncrasies of two tool sets. Some development environments help simplify this situation by providing a “shell” around the two tool sets and imposing a consistent interface and command structure. Unlike unified architectures, in heterogeneous core implementations, no immediate access exists between the microcontroller and DSP; developers must build the interprocessor-communication mechanisms and protocols. These mechanisms can employ semaphores through shared memory or link ports designed

for interprocessor communication. Processing resources are separate and exclusive between the controller and the DSP with some shared resources, such as small blocks of memory. The controller may still employ an RTOS, but the DSP will usually gain no direct benefit from it. It is more difficult to use and even more challenging to emulate conventional debugging techniques to simulate coupled heterogeneous-processor architectures. A dual-processor configuration enjoys a degree of autonomy between the instruction threads so that software changes on one processor may have little or no effect on the other processor. Changing the user-interface code in a dual-configuration system rarely impacts the signal-processing thread; this situation is less true for single-threaded implementations.

Will simpler, single-threaded, unified architectures replace dual-heterogeneous processor implementations? Realize that unified architectures must balance features and make trade-offs to deliver good performance for both control- and signal-processing capabilities. Unified architectures require higher clock rates, because they must do the work of two processors. In a dual-processor implementation, you can independently clock each core at the rate the application requires, and you can select each core for best of class for each type of processing. Single-instruction-issue, unified archi-

tectures, by definition, leave the computational engine idle while the control code is executing. Therefore, the clock rate must be higher than just the sum of the dual-clock approach. Higher clock rates affect power consumption and may be inappropriate for power-sensitive applications. For high-performance signal processing, the bus and memory architecture must be able to continuously feed the computational engine so there are no stalls. In this situation, just adding a MAC to a microcontroller becomes a weaker option as performance thresholds move forward. Give attention to your preferred processor’s road map and how the architecture will accommodate the higher signal-processing rates your application demands. If you are adding signal processing to a control-oriented application and a unified processor can grow with your performance requirements, then it makes sense to leverage your company’s legacy microcontroller code and the control-coding experience of your engineers with a unified control/signal processor. Likewise, if your application pushes the edge of DSP performance, and the signal and control processing are loosely coupled, you will probably better benefit from a dual-threaded configuration.

LOOK TO THE TOOLS

Unified processors do not eliminate the need for integrated multithreaded

FOR MORE INFORMATION...

For more information on products such as those discussed in this article, go to www.ednmag.com/info and enter the reader service number. When you contact any of the following manufacturers directly, please let them know you read about their products in *EDN*.

Agere Systems
1-800-372-2447
www.agere.com
Enter No. 301

ARM
1-408-579-2200
www.arm.com
Enter No. 305

BOPS
1-888-890-BOPS
www.bops.com
Enter No. 308

Intel
1-408-765-8080
www.intel.com
Enter No. 312

STMicroelectronics
1-718-861-2650
www.st.com
Enter No. 316

Texas Instruments
1-800-477-8924-x4500
www.ti.com
Enter No. 318

Altium/Tasking
1-858-521-4280
www.altium.com
Enter No. 302

Associated Compiler Experts
+31-20-6646416
www.ace.nl
Enter No. 306

DSP Architectures
1-360-573-4084
www.dsparchitectures.com
Enter No. 309

LSI Logic
1-866-574-5741
www.lsillogic.com
Enter No. 313

Tensilica
1-408-986-8000
www.tensilica.com
Enter No. 317

3DSP
1-949-435-0600
www.3dsp.com
Enter No. 319

Analog Devices
1-800-262-5643
www.analog.com
Enter No. 303

AXYS Design Automation
949-341-1900
www.axysdesign.com
Enter No. 307

Green Hills Software
1-805-965-6044
www.ghs.com
Enter No. 310

Microchip
1-480-792-7200
www.microchip.com
Enter No. 314

ARC Cores
1-408-361-7800
www.arccores.com
Enter No. 304

Infineon Technologies
1-888-463-4636
www.infineon.com
Enter No. 311

Motorola
1-512-895-2000
www.motorola.com
Enter No. 315

SUPER INFO NUMBER

For more information on the products available from all of the vendors listed in this box, enter no. 320 at www.ednmag.com/info.

software-development tools. Using multiple unified processors in a design can address some of the clock-rate and performance constraints by assigning one of the processors as the signal processor and the other as the host controller. Because they are identical devices executing the same instruction set, the processors present an opportunity to rebalance software loading that does not exist with heterogeneous processors. Another example of using a unified processor in a multiple-system is to initialize and supervise an array of DSPs. In this case, the unified processor can act as the host controller with some light but tightly coupled signal processing, whereas the DSP array handles the autonomous, heavy-duty signal processing.

The only way software can continue to provide more functions in less development time is through heavy code reuse, more analysis, and more assisted decision-making from tools earlier in the design cycle. Your ability to divide a problem into manageable parts depends on your ability to glue solutions together, and development tools can assist if they mature to encompass a broader view as systems continue to increase in complexity. Today's software-development tools are becoming more multiprocessor-aware for debugging, but that awareness does not extend deeper into the development cycle. Today's multiprocessor-aware debuggers are incomplete; they still have trouble with intermixing trace buffers for each processor in the system. When a problem exists in the system, it is essential to have a coherent and synchronized snapshot of each system component to track down the origin of the trouble. Multithreaded software development, even for homogeneous processors, is still a technically demanding, manual process. Getting to tool-assisted (versus automated) multithreaded software development or multiple-heterogeneous RTOSs is a huge challenge that is still an academic exercise.

Whether the software is single-threaded or multithreaded, there is only one system program. Managing the software complexity is an issue of system integration. Tools address DSP-software devel-

opment; controller-software development; system integration, including shared resource arbitration; verification tools; and production-testing tools. Little integration and two-way interaction exist between these tools. The efforts to make C programming useful for signal processing is a step toward a consistent high-level language that tomorrow's tools can leverage. Academic discussions say that the way to meet tomorrow's complex design requirements is to reveal the details of the hardware resources. The reasoning is that, as clock cycles get faster, you can no longer reach every corner of a chip in a single cycle, and it grows increasingly difficult to hide that latency from the software.

It is hard to see these processor architectures converging on a single approach when you recognize that they exist in response to widely varying application-specific constraints of performance, power, size, cost, and ease of use. A system- rather than component-level perspective is a key driver for design decisions. As an example, deciding whether to go with single-threaded or multithreaded programming depends heavily on your application requirements for performance, time to market, software maintainability and reusability; your company's engineering experience and development infrastructure; and system cost. □

REFERENCES

1. Cravotta Robert, "Software development tools are growing up," *EDN*, July 19, 2001, pg 65.
2. "DSP-C Specification," July 2001 Revision.

AUTHOR'S BIOGRAPHY



Technical Editor Robert Cravotta's experience with multithreaded control/signal-processing development includes dynamically tunable laser-sensor applications and autonomous spacecraft that uses vision processing to close the guidance and navigation-control loop. You can reach him at 1-661-296-5096, fax 1-661-296-1087, e-mail rcravotta@cahners.com.