Despite their popularity, idiosyncrasies in design tools and RTOSs (real-time operating systems) for DSPs interfere with users' needs and expectations. Unlike µCs, DSPs are plagued with hardware quirks. For example, RTOSs, which run on DSPs, fail to manage the most important DSP functions—acquiring, sampling, and processing data in real time while handling other housekeeping tasks. DSP systems are computationally intensive because they process massive streams of data. Unlike the control-flow processing that µPs and µCs perform, DSPs require that data is processed as it arrives, and, consequently, caching the data is impossible.

In addition to these problems, DSPs have small on-chip memory, which means you must minimize the code size and the time-critical code/data in the fastest processor memory. DSPs feature multiple internal and external buses that support parallel processing and many interrupts. Usually, a DSP data throughput can vary from 20 to 200 Mbytes/sec with a processing capability greater than 1000 MIPS. All DSPs have single-cycle MAC (multiply-accumulate) capability, and some DSPs have two multipliers, which allows them to process two MACs in one cycle. DSPs support circular buffers in hardware and bit-reversed addressing modes. Also, DSP memory architectures differ from memory architectures of conventional µPs. Most µPs allow only one memory access per clock cycle, but many DSPs allow the processor to perform several memory accesses during one clock cycle.

Programmable DSPs, unlike dedicated-hardware DSPs, need to manage their hardware resources so they can share them among DSP functions in an application. These resources include the CPU, on-chip memory, and peripherals. Now, DSP architectures are becoming more complex. For example, Texas Instruments' TMS320C80 packs four programmable DSPs, one RISC (reduced-instruction-set-computer) processor, special-purpose memory, and video controllers onto one chip. Programming the TMS320C80 is difficult because it has multiple CPUs, meaning that you have to decide how to grant and control access to shared memory. The TMS320C80 somewhat solves the problem by offering a crossbar-switch architecture linking all the CPUs with token-passing-arbitration circuitry. Lucent and Motorola's StarCore 140 DSP offers four MAC units and four ALUs on one chip and delivers 3000 MIPS at 300 MHz. Programming these complex DSPs can be a daunting task.

Typically, DSP functions need to execute in real time, and, therefore, the functions must time share the number of available CPUs. However, in most cases, simple time-sharing of the CPU may cause a time-critical DSP function to fail in a real-time scenario. For efficient algorithm execution, you can perform static memory allocation through explicit declarations in C language. But when the system gets complex with many DSP functions, you may have trouble statically allocating algorithm memory. Usually, you can use device drivers to handle a peripheral, but handling peripherals among DSP functions becomes a challenge.
One approach to the real-time processing problem involves separating DSP-based systems, which requires a mechanism that consistently distributes on-chip resources, such as memory, CPU, and functional tasks. A DSP RTOS must exploit the capabilities of the architecture and meet the critical requirements of these systems. You can use the RTOS to test and debug DSP applications in real time. A variety of scheduling options in an RTOS are mandatory for digital-signal processing. The requirement to stay within strict timing constraints mandates that a DSP RTOS include prioritized, pre-emptive scheduling.

**RTOSs are tailored to DSPs**

With Embedded Power Corp's royalty-free, scalable, multitasking RTXC DSP RTOS, you can develop applications using the C compiler of your choice. The vendor ports the RTOS kernel code to each supported processor and binds the code to a specific compiler for that processor. To get the combination you need for application development, you need only to specify the processor and the compiler tools. The RTXC DSP kernel supports StarCore SC140 processors as well as Motorola's DSP563xx and DSP566xx families.

To minimize the ROM footprint of the kernel, you can select the kernel object classes and services that are appropriate for your application. In addition, for each chosen class, you can select the class properties for your application, thus minimizing the amount of RAM the objects need. In applications that use the single- and multiple-stack components, the initialization procedure instantiates static objects before execution begins. In designs that use the multiple-stack component, dynamic kernel objects are available, and their definition occurs at runtime as part of the application code.

The standard RTXC DSP RTOS distribution package includes source code for demonstration applications. The vendor also provides application-conversion guides that help you make the transition from legacy-code schedulers or first-generation RTOS products to the RTXC DSP RTOS.

You can find support for managing Analog Devices' ADSP-2106x family of SHARC DSPs and Texas Instruments' TMS320C6x family of DSPs in ExpressLogic's ThreadX RTOS. Supplied in ANSI C source code, the ThreadX RTOS supports unlimited threads, queues, event flags, timers, semaphores, block pools, and byte pools. For both families, ThreadX optimizes context switching. When context switching occurs inside a ThreadX service call, it saves only the registers preserved across function calls as part of the thread's context-registers. Interrupt handling uses a similar technique. When an interrupt-service routine runs, only the compiler's scratch registers are saved initially. The full register set is saved only if it requires thread pre-emption. In the case of ADSP-21060 at 40 MIPS, the RTOS features a 6500- to 30,000-byte footprint.

Optimized for communications applications, Enea OSE Systems' OSE RTOS supports Lucent Technologies' and Motorola's StarCore SC140 DSP core. This RTOS offers a compact, high-performance kernel for DSPs, which can integrate within a distributed telecommunication system containing other types of CPUs, such as PowerPCs and ARM µCs. The OSE RTOS differs from other RTOSs because it adopts message-passing as its primary method of interprocess communication. The OSE RTOS for DSPs offers static, semidynamic, and interrupt processes. In addition, OSE for DSP RTOSs uses memory pools for memory protection, which is particularly useful in complex telecommunications systems that require high availability. (See sidebar "DSP RTOSs: state of the art and what to expect next.")

Porting the same RTOS onto different DSPs with minimum code modifications is a highly desirable feature. Eonic Systems offers the Virtuoso 4.2 that includes the Virtuoso 4.2 RTOS, a project manager, a multithreaded asynchronous network-host server, and a suite of graphical analysis and debugging tools. Eonic Systems claims that its Virtuoso CSP (Communicating Sequential Processor)-
based multitasking allows you to write code for one processor system but execute that code on other processors without regard for pointers. When a more powerful architecture is available, you can add processors to enhance performance or remove them without rewriting any code. Jovian's Pegasus graphical DSP design environment is seamlessly integrated with Virtuoso 4.2. Pegasus and you create and simulate DSP designs by graphically dragging and dropping predefined blocks of functions into a block diagram. Once you are satisfied with the simulation, Pegasus automatically generates the required C code. Its DSParCer separates the code into individual tasks and generates the DSP-specific code that runs in parallel on one or more DSP processors. Pegasus also creates a make file that runs the DSP compiler, linker, and Virtuoso RTOS configuration tools.

Virtuoso 4.2 provides a range of scheduling options, including round robin with prioritization, time-slicing, and prioritized and pre-emptive scheduling. Virtuoso also supports priority "inheritance" to protect tasks from being blocked during resource sharing. Two interrupt-handling levels can disable or enable global interrupts and can process an ISR (interrupt-service routine) in less than 1 µsec. The Virtuoso microkernel provides more than 90 kernel services that include task, memory, resource, and timer management, as well as queue, mailbox, semaphore, and event services.

Virtuoso supports DSPs and high-end mPs from Analog Devices (ADSP-21020, ADSP-2106x, and ADSP-21160 SHARC DSP families), Texas Instruments (TMS320C3X, 320C4X, 320C54X, 320C62XX, and 320C67XX), Advanced RISC Machines (ARM, ARM7T, and ARM9), IBM/Motorola (PowerPC), and Infineon (Carmel). Virtuoso 4.2 supports most of the COTS (commercial off-the-shelf) boards but runs best on Eonic's Atlas Universal Digital-Signal computer board built to support the advanced features in the Virtuoso RTOS. If you use the Atlas computer and Virtuoso, you can develop your software before or during DSP-hardware selection.

Virtuoso offers seamless integration with third-party debugging tools, such as Texas Instruments' Code Composer Studio and Analog Devices' Visual DSP software-development tools. Virtuoso 4.2 is available now and costs $10,000 for single-processor targets and $20,000 for virtual-single-processor, multiprocessor systems.

**Real-time DSP debugging with RTOSs**

One important benefit of an RTOS is that you can use the same RTOS to verify your application code. In the case of the RTXCDSP tool, you can test and debug your code during various stages of application development. During initial development phases, you can invoke the diagnostic mode to verify the passed parameter's kernel services by the application code. The diagnostic-mode extensions report such parametric or usage errors to a user-defined error handler. By invoking the trace mode, you can enable the kernel to capture system events, such as changes of context, interrupts, and event occurrences and store the time-stamped event information in a user-defined trace buffer. After developing your application code, you can collect operational statistics to help you verify correct operation by monitoring the interaction between the application code and the RTXCDSP kernel. The statistics mode allows you to extend the RTXCDSP kernel to capture operational information so that you can access it with kernel-awareness extensions to your compiler's debugger or other utilities.

Texas Instruments' DSP/BIOS II is a scalable, real-time kernel for the C5000 and C6000 TMS320 DSPs with pre-emptive multithreading, real-time analysis, and system-configuration tools. Using DSP/BIOS, you can instrument your applications to be probed, traced, and monitored as they execute applications in real time. Alternatively, the kernels can trace programs in real time without any explicit calls to the DSP/BIOS application-programming interfaces. In addition, DSP/BIOS is integrated within the vendor's Code Composer studio IDE (integrated development environment) and requires no runtime license fees. The DSP BIOS II kernel object viewing CCS (common-channe-
-signaling) plug-in provides visibility into kernel objects during program execution. You can use the tool to view the state of the execution threads or the properties of kernel objects used within the application.

Virtuoso also provides a graphical-design-analysis and-debugging capability for DSP-based systems. Virtuoso 4.2's real-time tracing monitor allows you to analyze all system events with clock-cycle precision in an interactive, graphical format. For any node, the real-time tracing monitor graphically displays the scheduling of the tasks on the node. Clicking on any point on the time trace launches a status window that shows the precise details at that time; these details include the tracer-entry number, node identification, active task, event system name, timer value, relative time required, kernel services in use, and event description. By examining the pattern of task execution, you can determine whether the scheduling is correct.

What can you expect to happen in this industry in the future? According to Vijaya Sarathy, software manager at Texas Instruments, Bangalore, India, the difference between a general-purpose µP and a DSP is becoming increasingly vague. Engineers will soon focus more on how dynamically an OS can adapt to scenarios and how well it can handle multiple-processor applications and multiple chip-level resources.

Author info

You can reach Technical Editor NS Manju Nath at +852-2965-1555, fax +852-2976-0706, e-mail nsmanjunath@cahners.com.hk.