

**A NEW APPROACH FOR WIRELESS DESIGN ALLOWS DESIGNERS TO INTEGRATE MULTIPLE, PACKET-BASED STANDARDS INTO RESOURCE-CONSTRAINED HANDSET HARDWARE.**

# Virtual machines and stochastic scheduling simplify wireless design

SYSTEM DESIGNERS sometimes refer to the traditional stack-development approach for user-equipment handsets as “silo-based” because of its extreme vertical integration between software and hardware and the lack of horizontal integration with other stacks (Figure 1). This silo approach breaks down when implementing multiple packet-based standards, because it assumes that the stack developer “owns” the underlying hardware resource and can therefore make assumptions about resources, such as scratch- and persistent-memory-buffer allocation. These availability assumptions are meaningless in a multimode environment, in which stacks that may “beat” against one another in their underlying timing are competitively acquiring resources, such as memory.

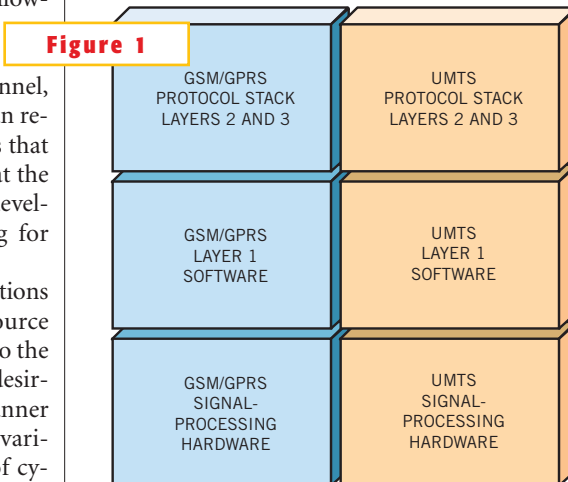
The silo approach assumes that you can configure the worst-case system loading at design time, allowing you to assign resources during system design rather than at runtime. However, the approach is essentially unworkable for multichannel, packet-based systems with a high peak-to-mean resource-loading profile. In addition, it assumes that one design group will code the system and that the standard will not change significantly during development. Both assumptions are likely wrong for modern communications systems.

Silo-based development often allows assumptions about each functional implementation’s resource usage, call format, and behavior to leak out into the rest of the design, leading to a number of undesirable design practices taking root under the banner of efficiency. For example, knowing how long various functions will take to execute (in terms of cycles) and how much scratch-memory function each will require, system designers can often write static schedules for scratch memory, allowing multiple routines that do not overlap in time to use a com-

mon buffer and thereby avoiding potentially expensive and nondeterministic calls to malloc() and free(). However, such a design also tends to be fragile. If you reimplement any of the engines, causing a modification in their resource profiles, timings, or both; if the underlying hardware should change; or, worse, if the stack shares those underlying resources with another stack altogether (the multimode problem), a ground-up redesign becomes a near certainty.

## AN ALTERNATIVE APPROACH

As with any high-complexity design problem, the best approach is to partition the issues into less complex blocks that you can tackle with some autonomy. The underlying conceptual model for this alter-



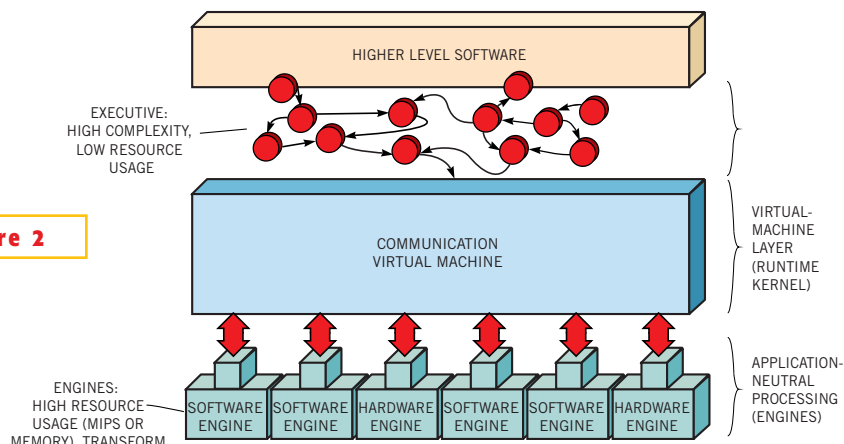
**A traditional silo-based development uses vertical implementation and lacks horizontal integration with other stacks.**

native approach, known here as the virtual-machine approach, assumes that Layer 1 of a communication stack decomposes into the executive, the virtual machine (runtime kernel), and the engines (Figure 2).

An analysis of Layer 1 software in operation reveals that it spends 80 to 90% of the execution time servicing the computation-intensive signal-processing transforms associated with the wireless application. These high-resource functions include Fourier transforms, vector multiplying, FIR filters, and decimators. In fact, these transforms exhibit a high degree of commonality across different wireless applications. You implement these high-resource, largely application-neutral components in either dedicated hardware or highly platform-optimized software. The proposed architecture treats high resource functions in a special way to create “engines.” Specifically, the architecture conformance-tests engines against their behavioral equivalents and profiles their performance, enabling a unique type of resource simulation.

The remaining 10% of processing resources, the executive, is essentially a controlling state machine that invokes the transforms as the specific wireless standard requires. The executive is usually highly complex but requires relatively few processing resources. These low-resource, largely application-specific components rarely contain resources that inherently bind them to an underlying hardware substrate. In fact, they are portable to any other design using the same virtual-machine runtime kernel regardless of whether they implement the engines in hardware or software. Examples of executive functions include overall data-flow expression of a data plane, acquisition and tracking logic, and channel creation and deletion of planes in a third-generation design.

The alternative approach bases the architecture around a thin virtual-machine runtime kernel that separates the executive and the high MIPS engines. At its simplest level, it provides semiconductors and RTOSs with abstraction layers that enable portable baseband software. This function does not replace the RTOS; the system still uses RTOS services where the engines reside on a processor, usual-



**Figure 2**

**An alternative development approach bases the Layer 1 software architecture on a virtual machine.**

ly in a DSP. The virtual-machine runtime element needs to identify common threading, interrupt, memory, and resource-management models, which the designer then maps onto the available primitives for any third-party RTOS by its runtime implementation. The virtual machine also includes sophisticated resource-management functions that are critical to solving the Layer 1 wireless-development bottleneck (Figure 3).

All this effort would be for nothing, from a resource-scheduling point of view, if the high-level code were to directly call the engines. Direct calls would more or less determine the underlying execution sequence and the threading model, which is critical for an efficient implementation. Even worse, the caller would be responsible for setting up the appropriate memory for the underlying engine, effectively leading to explicit resource scheduling. In this approach, only a middleware service, namely the virtual-machine runtime kernel, may call an engine. Specifically, the runtime kernel includes a scheduler that effectively exists as a single instance across all executive-process and logical threads. It uses a plug-in scheduling policy to decide which of these tasks to submit for execution to the underlying RTOS, how many RTOS threads it will use, and at what priority and logical time step.

**SIMULATION IS KEY**

Although you must ensure that your system is bit-correct, it is equally necessary to make appropriate judgments about resource management. Certainly,

worst-case analysis is inappropriate and can give vastly overpessimistic and, hence, expensive results for multimode designs. However, using performance simulation may reveal a dependency on how the various stacks “beat” against one another in time. Virtual-machine resource simulation does not run the engines themselves when the executive calls them but rather simply updates a “resource-use scoreboard.” You capture information on the resource usage of each engine separately during a cycle-accurate simulation or during actual hardware execution by a performance-profiling process. The profile holds records of the use of each resource type (memory, cycles, bus bandwidth, and others) for a range of values of an appropriate independent variable, such as vector size or bit width. It can stochastically express the engine-resource profiles, whereby, for example, the number of cycles a task requires is not simply a deterministic function of the dimensions of its inputs. (For example, a turbo coder takes more cycles to process a more corrupted input vector.) A parametric form stores the resource costs for each engine as part of the engine’s componentized description. The simulator executes the “real” executive code but substitutes execution of the engines for robust estimates of their resource usage.

Because resource simulation is so much faster than cycle-accurate simulation, it allows you to place a vast amount of traffic through a candidate system design and examine many scenarios of interaction between two stacks. This ben-

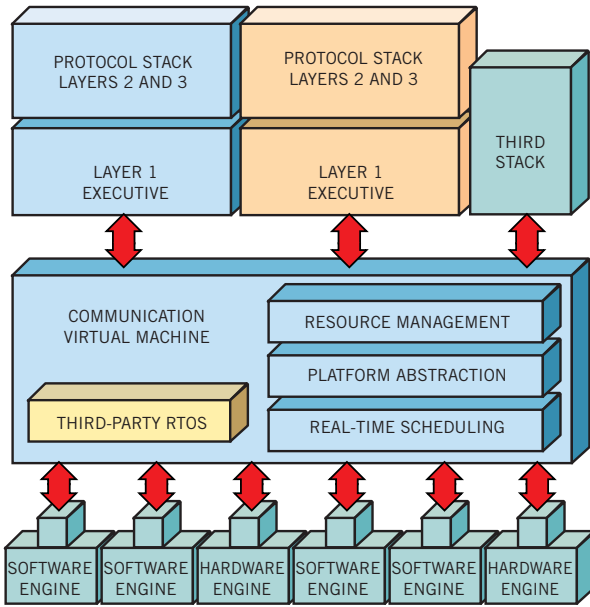
efit aids designers of complex multi-mode and packet-based systems. Resource simulation creates a representative estimate of system behavior over a range of operating conditions. Once you compute this data, you can use it to both make design-time decisions, such as partitioning the software across a hardware substrate, and as one input for a runtime predictive (stochastic) scheduler.

**MULTIMODE SYSTEMS**

The multimode case includes a number of independent executives that you must schedule over a single physical thread. Designers may assume that although engine-resource profiles and engine-call sequence maps may be available or, in any case, may

be derived, explicit deadline information is rarely available. Your problem then becomes deriving a valid serialized schedule for such a system at a specified loading, which you express as a set of system parameters, such as the number of active channels or the maximum throughput bit rate. Limits of 100% on each of the resources constrain the efficiency of any such schedule on the upper boundary. (For example, any schedule that uses 120% of the available memory is, at some point, invalid or at least requires further work to clarify its starvation behavior.) But, below this point, some weighting determines the “goodness of fit.” For example, the designer may consider desirable a serialized schedule that keeps memory allocation at less than 50% and so appropriately weight the overall metric.

The solution to the multimode problem requires more sophisticated handling than simple scheduling policies, such as first-come, first-served. Specifically, a more sophisticated scheduling policy is necessary. Current policies neither use the known engine-resource costs to any significant extent nor attempt to look forward. Furthermore, the use of deadlines in a real-time system, particularly on an engine-level basis, is often unrealistic due to the event-driven nature of processing. A more sophisticated re-



**Figure 3** The virtual-machine runtime provides resource management and scheduling, decoupling executives from engines.

source-allocation policy is necessary, and you must treat it as an integral part of the scheduler.

**STOCHASTIC SCHEDULING**

Designers may process data that they gather during resource simulation to construct an engine-request probability matrix. The matrix is based upon the calls from the executive to the engines via the virtual-machine runtime kernel (Figure 4). The derived matrix is sparse, with many “zero” transitions and a number of

**LOOKING “INFINITELY” INTO THE FUTURE WOULD CAUSE A COMBINATORIAL EXPLOSION IN THE CONSIDERED STATE SPACE.**

“one” transitions. However, a typical stack with branching produces some probabilities between zero and one, which is the first introduction of stochastic behavior into the system.

At any given time, various logical threads in the controlling executive present the runtime scheduler with only the very next deterministic-engine request to consider for execution. Sched-

ulers can use the aforementioned transition matrices and the costs of engine execution available from the engine-resource usage profiles to derive a number of possible forward scenarios for evaluation.

However, even if it were possible, looking “infinitely” into the future would cause a combinatorial explosion in the considered state space. It is also undesirable to look a uniform “fixed” number of hops ahead, because some schedules may be more promising than others. The problem here is cognate to the one that chess-playing software faces; it must consider the possible future consequences of various moves. It does not consider all possible outcomes (even within

the constraints of, for example, a two-move look ahead) but rather uses a set of heuristics to determine which scenarios to further expand. Stochastic scheduling policy faces a cognate challenge.

With the scenario-generation heuristics in place, the next required step is to provide a set of planning metrics. You use these metrics to analyze the merits of each of the candidate scenarios that the generation heuristics produce and, ultimately, to allow you to represent each scenario with a scalar “goodness” value.

The overall domain for these planning metrics usually spans some or most of the following “objective” measures, evaluated on a per-time slice and per-group basis: The measures are overall memory usage, overall group usage, proximity to deadlines when known, and power usage.

You may also employ a number of other heuristic metrics. Referring back to the chess-software analogy, the objective metrics would be cognate to valuing outcome positions based on piece values, and the heuristics cognate to rules such as “Bishops placed on open diagonals are worth more than bishops that command fewer free squares.”

System designers can set the transfer-function curvature—determining, in effect, whether the system responds early or late to resource shortages. In addition, system designers can determine the rel-

ative weights to assign to each of the planning metrics whose sum gives the final single scalar value.

The approach that this virtual-machine paradigm uses—stochastic scheduling—provides a solution to the multimode problem. It uses tables that predict the likelihood of engine-request transitions, constructed during simulation runs, together with the engines' resource-usage profiles, to allow the scheduling policy at runtime to look forward in time and dynamically balance the requirements of multiple concurrent stacks.

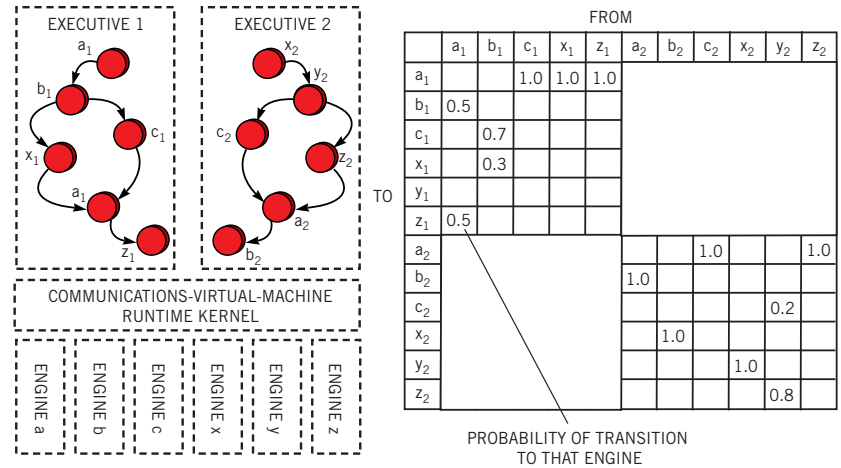
Clearly, it is critical that the benefits of stochastic scheduling exceed its cost in cycles and memory in implementation. You can apply many proprietary techniques, to minimize the recalculation requirements at each time slice. You establish the efficiency of each putative implementation by further examining the system performance using the resource simulator with the appropriate plug-in scheduling policy installed in the runtime kernel.

Stochastic-scheduling policies deliver superior efficiency over general-purpose approaches, such as "earliest deadline first," because they have additional information available and are more appropriate to communication applications than, say, static, RMA (rate-monotonic-analysis) techniques.

**RESOURCE STARVATION**

Starvation occurs when necessary system processing does not occur in a timely fashion, because inappropriate resources were available to schedule it. For a number of cases, a "smarter" scheduling policy can produce significantly better performance, but, as loadings increase, there ultimately comes a point at which even the most sophisticated policies cannot cope. At this point, the system must be able to systematically fail some of the requests. Such failure might actually be part of the envisioned and accepted behavior of the system—a necessary cost of existing in a bursty environment. The important thing is that the scheduler takes action and gracefully degrades the system performance, rather than invoking a catastrophic failure.

Stochastic scheduling under the virtual-machine approach successfully generates valid serialized schedules for a significant class of multimode scenarios, in



**Figure 4** You can derive a conceptual model of an engine-state transition-probability matrix from simulation data.

which silo-based approaches fail. Furthermore, it gives better performance than "simple-RTOS" scheduling approaches. Specifically, stochastic scheduling provides significant benefits through its use of additional information not available to conventional approaches to balance inherently bursty requirements at runtime between multiple competing stacks. The virtual-machine paradigm of resource-profiled engines (high-resource basic transforms) is central to this endeavor, because it provides additional information to the scheduler about the most significant resource consumers a priori and because such an approach is necessary for efficiently performing large-scale Monte Carlo traffic simulation of multimode systems.

The virtual-machine resource simulator is necessary, because it provides the engine-request-transition probability matrix for each executive. A static schedule is inappropriate for bursty and multimode systems, because they tend to have high peak-to-mean resource-usage profiles, and static schedules at design time tend to focus on worst-case analysis, leading to inefficient or impractical designs. The virtual-machine runtime scheduler, by separating executives from the engines that they wish to invoke, is a necessary step that prevents developers falling into the "silo-mode" trap and enables them to share resources. All significant resources, not simply cycles, require scheduling; therefore, you must also schedule memory. Memory schedulers allow the end system to approach the ef-

ficiency of silo-mode techniques that fix all or most buffers at design time and still allows for burstiness and beating multimode, multivendor implementations.

This alternative approach to the problematic silo-development method provides a design methodology and tool set based on the deployment of a virtual-machine runtime kernel. It removes the executive software's dependence on the underlying semiconductor hardware and in doing so commoditizes the underlying engine implementations. Innovative simulation and optimization techniques maximize the performance and cost efficiency of resulting products. The architecture allows separate wireless executives to safely share underlying high-MIPS engines through the virtual-machine kernel. It offers a significant advantage by creating multistandard options providing the necessary flexibility and cost efficiency for the sophisticated user equipment of the future. □

**AUTHOR'S BIOGRAPHY**  
 Mike Woodward is a technical manager at RadioScape Ltd (Scottsdale, AZ), where he has worked since 1999. He is the author of an award-winning Digital Audio Broadcasting system and the founder of the RadioLab Group, which produced the award-winning RadioLab 3G product. He holds a BS in physics from Leicester University (UK), an MS in microwave solid-state physics from Portsmouth University (UK), and a master's of philosophy in the properties of selenium-doped MODFETs from Sheffield University (UK).