Hardware-in-the-Loop Simulation

Hardware-in-the-loop (HIL) simulation is a technique for performing system-level testing of embedded systems in a comprehensive, cost-effective, and repeatable manner. This article provides an overview of the techniques of HIL simulation, along with hardware and software requirements, implementation methods, algorithms, and rules of thumb.

System level testing is one of the major expenses in developing a complex product that incorporates embedded computing. The need to minimize time to market while simultaneously producing a thoroughly tested product presents tremendous challenges. Increasing levels of complexity in system hardware and software are making this problem more severe with each new generation of products. Additionally, any significant changes made to an existing product's hardware or software must be thoroughly regression-tested to confirm that the changes do not produce unintended effects.

Embedded computing is becoming more pervasive in systems that are safety-critical, which makes the need for thorough testing even more acute. Clearly, there is an urgent need to accelerate and automate system-level testing as much as possible within the constraints of test system cost and development effort.

Hardware-in-the-loop (HIL) simulation is a technique for performing system-level testing of embedded systems in a comprehensive, cost effective, and repeatable manner. HIL simulation is most often used in the development and testing of embedded systems, when those systems cannot be tested easily, thoroughly, and repeatably in their operational environments.

HIL simulation requires the development of a real-time simulation that models some parts of the embedded system under test (SUT) and all significant interactions with its operational environment. The simulation monitors the SUT's output signals and injects synthetically generated input signals into the SUT at appropriate points. Output signals from the SUT typically include actuator commands and operator display information. Inputs to the SUT might include sensor signals and commands from an operator. The outputs from the embedded system serve as inputs to the simulation and the simulation generates outputs that become inputs to the embedded system. Figure 1 shows a high-level view of an example HIL simulation.

The HIL simulation test concept can be applied to a wide variety of systems, from relatively simple devices such as a room temperature controller to complex systems like an aircraft flight control system.

In the figure, the SUT is represented as having interfaces to sensors and actuators, as well as an operator interface that displays information and accepts command inputs. In this example, operator commands are synthetically generated by the simulation to stimulate the SUT during testing. The use of synthetically generated operator commands allows the automation of test sequences and permits precise repeatability of tests. It may also be possible to run the simulation in a mode in which humans observe the operator displays and generate inputs. This mode may be valuable for usability testing and operator training, but the ability to precisely repeat test sequences is lost.

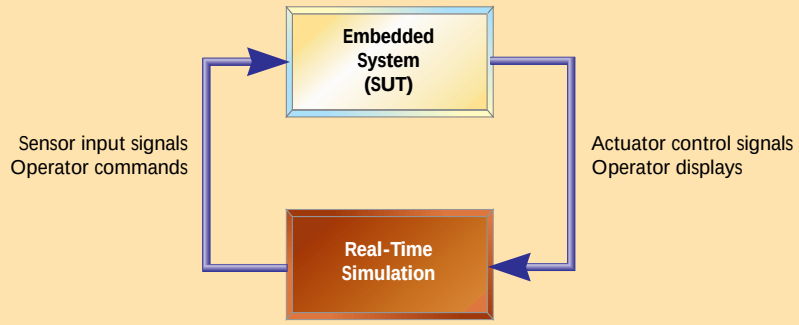
You can apply the HIL simulation test concept to a wide variety of systems, from relatively simple devices such as a room temperature controller to complex systems like an aircraft flight control system. HIL simulation has historically been used in the development and testing of complex and costly systems such as military tactical missiles, aircraft flight control systems, satellite control systems, and nuclear reactor controllers.

Ongoing performance advances and price reductions in computer hardware have made it possible to develop cost-effective HIL simulations for a much wider range of products. This article is intended to promote the application of HIL simulation to areas where this technique may not have been used in the past.

HIL simulation can help you to develop products more quickly and cost effectively with improved test thoroughness. Another benefit of HIL simulation is that it significantly reduces the possibility of serious problems being discovered after a product has gone into production. During the product development phase, HIL simulation is a valuable tool for performing design optimization and hardware/software debugging.

However, for an HIL simulation (or any simulation, for that matter) to produce reliable outputs, it is critical to demonstrate that the simulated environment is an adequate representation of reality. This is where the concepts of simulation verification and validation must come into play.

FIGURE 1 Example HIL simulation



Simulation verification and validation

Before the HIL simulation can be used to produce useful results, there must first be a demonstration that the SUT and its simulated environment in the HIL simulation sufficiently represent the operational system and environment. The major steps in the process of demonstrating and documenting that the correctness and fidelity of the simulation are adequate are called verification and validation.

Verification is the process of demonstrating that the HIL simulation contains an accurate implementation of the conceptual mathematical models of the SUT and its environment. This step can be performed before a prototype of the SUT has even been developed. Verification is typically performed by comparing HIL simulation results with the results of analytical calculations or results from independently developed simulations of the SUT. Other techniques such as source code peer reviews can be effective verification tools.

Validation consists of demonstrating that the HIL simulation models the embedded system and the real world operational environment with acceptable accuracy. A standard approach to validation is to use the results of system operational tests for comparison against simulation results. This type of validation test involves running the embedded system in the HIL simulation through a test scenario that is identical to one that was performed by an actual system in an operational environment. The results of the two tests are compared and any differences are analyzed to determine if they represent a significant deviation between the simulation and the real world.

If the operational test results don't

BEN FISHMAN

accurately match the HIL simulation results, improving the fidelity of the simulation modeling in particular areas may be necessary. Defects in the simulation software or hardware interfacing may also become apparent when comparing simulation results to operational test results. If changes must be made to the HIL simulation to correct problems, you'll have to rerun any validation tests that have already been performed to confirm adequate simulation performance.

By carefully selecting the scenarios for operational system tests, you can validate the simulation for a limited number of specific test conditions and then proceed on the assumption that the simulation is valid over interpolations between those conditions. You must take care to demonstrate that the intervals over which interpolation is being performed are smooth and do not contain any hidden surprises.

After all deviations between the performance of the embedded system in the operational environment and in the HIL simulation have been reduced to an acceptable level and properly documented, the HIL simulation is considered validated. HIL simulation tests can then be used to replace or augment operational tests of the system as long as the environment being simulated falls within the validated range of the simulation.

When working with a validated simulation, you must always be careful to understand the range of conditions over which the simulation has been validated. If a proposed test is for a situation where the simulation has not been validated and an extrapolation from validated conditions cannot be justified, you must use the simulation only with extreme caution. In these cases, the simulation results should be accompanied with a statement of the assumptions that had to be made in using the simulation.

When making modifications to a validated simulation, you'll have to maintain adequate software configuration control and carry out regression tests to verify that simulation performance isn't adversely affected as changes are made. Restoring and running a previous version of a simulation is sometimes necessary. This situation can occur when a question arises as to whether an unexpected result is caused by actual behavior of the SUT,

Real-time integration algorithms

In an HIL simulation, the interactions between the SUT and the real-world environment are usually modeled as a continuous system defined by a set of nonlinear differential equations. This system must be arranged as a set of first-order differential equations, which are solved numerically in the real-time simulation.

Differential equations can be solved approximately by advancing time in small, discrete steps and approximating the solution at each step using a numerical integration algorithm. In a non real-time simulation, a variety of algorithms exists for varying the time step size so that a predefined degree of accuracy is maintained. These methods are usually unsuitable for use in a real-time environment due to requirements to sample inputs and update outputs at precise time intervals.

Some popular fixed-step size integration algorithms such as the fourth-order Runge-Kutta algorithm require inputs for their derivative sub-step calculations that occur at a time in the future relative to the time of the current sub-step. This makes them unsuitable for use in a real-time application.

Algorithms that are suitable for real time generally use fixed-time step sizes and only require inputs for derivative evaluation that are available at the current time or earlier. The simplest algorithm that meets these criteria is Euler integration. The formula for integrating a state x from the current time step (denoted by the subscript n) to a time h seconds into the future (subscript n+1) with a current derivative $\dot{x}_n$ is:

$$x_{n+1} = x_n + h\dot{x}_n$$

While it has the advantage of simplicity, Euler integration has poor performance characteristics in terms of the error in approximating the solution. It's a first-order algorithm, so its local truncation error (the error in the solution after a single step) is proportional to h.

An algorithm that provides much better accuracy while remaining suitable for real-time use is the Adams-Bashforth second order algorithm. It uses the current value of the derivative $\dot{x}_n$ as well as the previous frame's derivative $\dot{x}_{n-1}$:

$$x_{n+1} = x_n + h\left(\frac{3}{2}\dot{x}_n - \frac{1}{2}\dot{x}_{n-1}\right)$$

This is a second-order method with local truncation error proportional to h^2. It is the most commonly used integration method in real-time continuous system simulations. Its main drawback is that a significant transient error can occur if there is a discontinuity in the derivative function. It also requires an initial step using the Euler algorithm because $\dot{x}_{n-1}$ isn't available during the first simulation frame.

Many additional real-time integration algorithms are available at orders from second to fourth. Some, such as versions of the Runge-Kutta methods modified to be compatible with real-time inputs, provide additional features such as superior performance in dealing with discontinuous derivatives. This property is valuable when interfacing a model of a continuous system to a model of a discrete system.

In practice, the Adams-Bashforth second-order method is usually suitable. If this method doesn't work well, you may need to reduce the integration step time h.

or is the result of a change that has been made to the simulation software. You can quickly resolve these questions if the simulation software is under proper configuration management. Reasonably priced software version control tools are available that can make the configuration management process robust and relatively easy.

Verification and validation are indispensable parts of simulation development. If the simulation does not undergo an adequate verification and validation process, its results will lack credibility with decision makers and significant project risks may be created. To minimize these risks, you must allocate adequate resources for a verification and validation effort from the beginning of an HIL simulation development project.

HIL simulation hardware and software

To develop an HIL simulation, you'll need suitable computing and I/O hardware, as well as software to perform the real-time simulation modeling and I/O operations. Let's examine the requirements for these components in more detail.

Simulation computer hardware

In addition to the SUT, the hardware used in an HIL simulation must include:

- A computer system capable of meeting the real-time performance requirements of the simulation
- Facilities on the simulation computer (or on a connected host computer) to allow operator control of the simulation as well as simulation data collection, storage, analysis, and display
- A set of I/O interfaces between the simulation computer and the SUT

The real-time performance requirements for the simulation computer depend on the characteristics of the embedded system to be tested and its operational environment, such as:

- The SUT's I/O update rates and I/O data transfer speeds
- The bandwidth of the dynamic system composed of the SUT and its environment
- The complexity of the SUT elements and operational environment to be modeled in the simulation software

I/O devices

Many different categories of I/O devices are used in embedded systems.

In an HIL simulation, an I/O device must be installed in the simulation computer that connects to each SUT I/O port of interest. I/O interfaces are available from several sources that support signal types, such as:

- Analog (DACs and ADCs)
- Discrete digital (for example, TTL or differential)
- Serial (for example, RS-232 or RS-422)
- Real-time data bus (for example, MIL-STD-1553, CAN, or ARINC-429)
- Instrumentation bus (IEEE-488, for example)
- Network (Ethernet, for example)
- Device simulators (for simulating LVDT transducers, thermocouples, and the like)

For an SUT with low I/O rates and a simulated environment that isn't overly complex, an ordinary PC running a non real-time operating system such as Windows NT may be capable of running a valid and useful HIL simulation. For complex, high I/O rate SUTs, a high-performance computer system is essential. In these applications, you'll need more than just raw CPU speed. The simulation computer must also have well-defined and repeatable real-time performance characteristics. A high-performance simulation might require that the software update all the simulation models and perform I/O at precise intervals of a few hundred microseconds.

The simulation computer must provide system-level software that supports real-time computing and doesn't allow code execution to be blocked in inappropriate ways. Most general-purpose operating systems don't provide sufficient support of real-time features to be useful in anything other than a low I/O rate HIL simulation. This condition may necessitate the use of an RTOS or a dedicated real-time software environment on the simulation computer.

A summary of requirements for a high-performance real-time simulation computer system includes:

- High-performance CPUs
- Support for real-time operations
- High data transfer rates
- Support for a variety of I/O devices

In years past, these requirements were met by specially designed, extremely expensive simulation computers. While more recently, many HIL simulation developers have turned to the VME bus as the basis for their simulation computer systems. In the future, newer buses such as CompactPCI may provide a lower cost foundation for developers of real-time simulations.

Simulation software structure

The simulation software contains sections of code that perform the tasks needed during real-time simulation. A diagram of the basic software flow of an HIL simulation is shown in Figure 2.

As examination of the flow diagram reveals, the HIL simulation software can be separated into three basic parts:

- Initialization of the simulation software and external hardware
- A dynamic loop that includes I/O, simulation model evaluation, and state variable integration
- Shutdown of the simulation software and external hardware

At the bottom of each pass through the dynamic loop, an interval timer must expire before execution of the next frame begins. The length of this interval, known as the simulation frame time, is a critical parameter for the HIL simulation. The frame time

must be short enough to maintain simulation model accuracy and numerical stability. At the same time, it must be long enough to tolerate the worst-case time to complete all the calculations and I/O operations in the dynamic loop. A shorter frame time requirement implies higher performance requirements for the simulation computer hardware. Alternatively, a shorter frame time may require simplification of the simulation models so that their calculations can complete in the available time. As the frame time is lengthened, simulation accuracy degrades. At a certain point, the numerical integration algorithm becomes unstable. The following formula is a rule of thumb for determining the longest acceptable frame time for a simulation model:

$$h \leq \frac{\tau_s}{20}$$

In this formula, τ_s is the shortest time constant in seconds of the simulated dynamic system and h is the frame time in seconds. As h is increased above the range given by the formula, the accuracy of the simulation will begin to suffer and eventually it will become numerically unstable.

Multiframing

Some subsystems modeled in a simulation may have time constants that vary significantly from those of other subsystems. When this occurs, improving simulation performance is possible by using the technique of multiframing. A multiframe simulation has more than one frame rate. The frame times h_1, h_2, and so forth are generally set at integer multiples of a common factor, with the fastest frame time called h_f. The simulation updates each model at the times appropriate for that model's frame rate.

Faster frame rate models that use values computed by slower models may need to use interpolation or extrapolation techniques to compute input values appropriate for the current time. State variables in a slow frame are suitable for interpolation, and algebraic variables must be extrapolated. The interpolation or extrapolation may be done using methods of first or higher order.

If the simulation computer supports multithreading, a scheduling technique such as Rate Monotonic Scheduling can manage the execution of the models and their I/O operations at the different frame rates (see the sidebar on Rate Monotonic Scheduling). Appropriate inter-thread communication techniques must be used to allow data to be passed among the models in a controlled manner. A multiframe simulation can provide comparable accuracy to a single frame rate simulation that updates all its models at the h_f rate while requiring far less CPU power.

Simulation programming languages and environments

Over the years, a number of specialized programming languages and environments have been developed to ease the task of developing simulations of dynamic systems. Some simulation languages are text-based while other development environments are graphical. Graphical tools allow the user to construct diagrams describing the system to be simulated. Many of these tools are intended for use only in non real-time applications, and therefore aren't suitable for developing HIL simulations. In addition, many of these languages and tools are either proprietary or are supported by only a single company.

Common attributes of full-featured dynamic simulation languages and graphical development environments include:

- Numeric integration of state variables

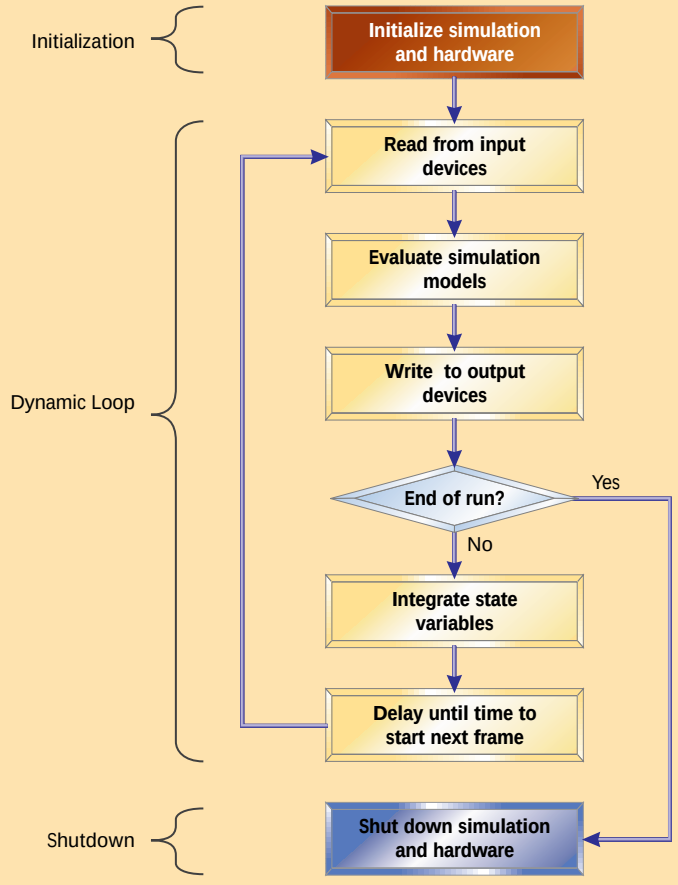
- Support for multidimensional interpolation functions
- Support for matrix and vector data types and mathematical operations
- A library of simulation component models such as transfer functions, limiters, or quaternions

For a simulation language or graphical programming tool to be useful for HIL simulation, it must be able to generate code optimized for use on a real-time simulation computer. It should also provide a simulation operator interface that allows the user to:

- Examine and modify simulation variables
- Perform simulation runs
- Collect and store simulation data
- Analyze, display, and print simulation results
- Debug and optimize the simulation

In addition to the specialized simulation languages and graphical development tools, it is common to use general purpose programming languages such as Fortran and C to develop HIL simulations. Many HIL simulations have been written from scratch, using only the tools and libraries supplied with the target system compiler. While this approach can succeed, it requires a great deal of effort developing efficient and correct methods for numerical integration, interpolation function generation, coordinate transformation, and so forth.

FIGURE 2 HIL simulation software flow



Developing simulation models

Developing simulation models is a field unto itself. For a relatively simple application, it may be possible to develop a model that consists of a few equations describing the physics of the subsystem or phenomenon being modeled. For a more complex system, a model might be based on a large number of complex mathematical algorithms and voluminous tables of experimentally measured data.

A real-time simulation model of a continuous dynamic system is usually implemented as a set of simultaneous, nonlinear, first-order ordinary differential equations. These equations are supplemented by algebraic equations that are used to compute intermediate and other miscellaneous values. This entire set of equations is used to compute the derivatives of the state variables during each simulation frame. After evaluating the derivatives, the state variables are integrated numerically to produce the state variable values that will be valid at the start of the next frame.

For subsystems that operate in a discrete-time manner, models can be developed in terms of difference equations. In a difference equation, the next-frame value of a state is computed during the evaluation of the current frame. At the end of the frame, the discrete states are updated to their next-frame values. Discrete states can be used to implement algorithms such as digital filters and to model subsystems such as DAC output signals. Continuous-system models and discrete-system models can be combined in a simulation as long as integration algorithms are used for the continuous system that can accommodate the discontinuities introduced by the discrete time model.

In general, simulation models are not precise representations of their

physical counterparts. Assumptions and simplifications must be made to develop simulation models. The fewer assumptions made, the more complex a model must be. Some major limitations on model complexity are:

- The time it takes to design, develop, verify, and validate the simulation model
- The effort required to collect and analyze experimental data needed to develop the model
- The execution time and memory space that the model's implementation consumes on the simulation computer

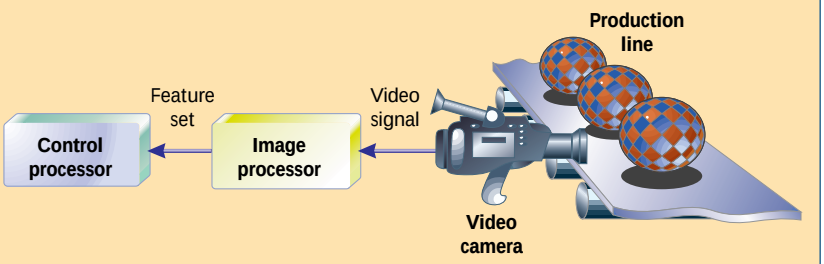
The developer must design and implement a model that has adequate fidelity for its intended purpose while remaining within these constraints.

Many techniques have been employed in HIL simulations to implement models of adequate fidelity while remaining within simulation computer execution time and memory limits. Some of these techniques include:

- Precompute as much as possible during program initialization. Although the compiler's optimizer can help with techniques such as common subexpression elimination, you can enhance performance further by removing calculations that produce unchanging values from the simulation dynamic frame
- Use interpolation functions to approximate the evaluation of complex expressions. Doing a multidimensional lookup and interpolation may be faster than evaluating an expression containing several transcendental function calls, floating point divisions, or other time-consuming operations
- In a real-time simulation, the worst-case execution time for a model is all that matters, rather than the average time. Therefore, focus on optimizing only those execution paths that contribute to the worst-case execution time. Good profiling and debugging tools are invaluable during this process

Although developing and validating a simulation model that meets the requirements discussed above is challenging, the model can then be used in a variety of applications in addition to real-time simulation. For example, the model can be used by product

FIGURE 3 Example production line control system



developers to perform their own system analyses, and they can modify it to examine the effects of proposed product changes and enhancements. These changes can then be implemented in the HIL simulation for further testing. This sharing of simulation models among different groups can result in tremendous benefits to the product development process.

Implementation issues

Several issues in the development and operation of an HIL simulation can be problematic. Some areas that have created difficulties in past HIL simulations are examined in this section.

Selection of appropriate interface points. When you are interfacing an embedded system to a simulation computer, it isn't always obvious what the best point in the system is at which to implement the interface. As an example, consider the application shown in Figure 3, in which a video camera monitors items moving along a production line. The signal from the camera is captured by an image processor, which extracts a feature set from the image. This feature set is then processed by the production line control processor.

An HIL simulation used for development of the production line control system would require an interface to the information coming in from the video camera. Where should this interface be located? One possibility is that the simulation computer simulates the operation of the production line, video camera, and image processor, and synthesizes feature sets for transmission to the control processor. This method would probably have the lowest cost because it would be done entirely in the digital domain and the feature set data is relatively low-bandwidth as compared to the video signal. However, in this approach the video camera and image processor aren't tested in the HIL simulation. If the operational details of the camera and image processor aren't thoroughly understood and accurately modeled, this approach may be inadequate.

A more costly alternative would be to have the simulation computer synthetically generate a scene in real time and transfer it to the image processor with the same format and timing the video camera uses. This would require additional processing and I/O hardware in the simulation computer, along with software to control them. With this technique, the image processor is tested in the HIL simulation, and testing can be performed repeatably. However, the video camera still isn't being tested.

Another even more expensive alternative would be to set up a video projection system that places the real-time, synthetically generated image on a screen in front of the video camera.

This allows the entire subsystem to be tested, including the video camera, image processor, and production line control computer. Drawbacks to this approach, besides the hardware cost and development expense, include the need to align and calibrate the projected image so that it appears realistic to the system. Although this technique may seem farfetched for this application, it is used effectively in the test of some tactical missile sensors.

Finally, you could use a simulated or real production line to create the scene viewed by the video camera in real time. This method may or may not be feasible, depending on considerations such as whether such production lines exist, how complex it would be to build a simulated line, and so forth.

This example demonstrates how important it is to identify the requirements for an HIL simulation and select the appropriate points at which to interface the system under test to the simulation computer. Selection of the proper interface point depends on the simulation purpose, the adequacy of models of subsystems that might be simulated rather than implemented as real hardware in the simulation, and funding available to develop the HIL simulation.

Budget realistically for simulation development and ongoing operations. An HIL simulation can be an expensive proposition. Its value will be realized, though, if overall project cost and risk can be reduced compared to the cost if the simulation had not been used. The initial development of an HIL simulation can be quite expensive and this must be understood during project planning. Continuing operation of the simulation will also require competent technical staffing to implement upgrades and troubleshoot problems.

The project phase at which the HIL is brought on-line can have a significant impact on overall product development cost. The availability of an HIL simulation early in the project can be invaluable to system designers who need a tool for studying alternatives and refining their designs.

Consider in advance how to deal with unexpected results. Inevitably, there will be times when a simulation doesn't produce the expected results. If this occurs while a deadline is creating severe time pressures, these results may be ignored. This situation is especially likely to occur if the simulation hasn't gone through a thorough validation process and won the support of all interested parties.

Some preplanning should be done to decide what actions to take in these situations. Ideally, sufficient resources should be devoted to determining the reason for the discrepancy. If it turns out that there is a system problem, the HIL simulation will have demonstrated its value.

Rate Monotonic Scheduling in real-time simulations

Rate Monotonic Scheduling (RMS) is a method for scheduling real-time tasks so that each task's deadlines will always be met. The technique is applicable to run-time environments that support prioritized, preemptive multitasking or multithreading. Here, we use the term *task* to mean either a task or a thread. The mathematical basis of RMS has been rigorously developed and analyzed (visit the Software Engineering Institute at *www.sei.cmu.edu* and search for "Rate Monotonic Analysis").

To use RMS, the execution frequency of each task must first be identified. In our HIL simulation application, this frequency will be the inverse of the task's integration step size, *h*. The scheduling priority of the tasks is then assigned so that the highest frequency task has the highest priority on down to the lowest frequency task, which has the lowest priority. This assignment of priorities as a monotonic function of task execution rate is what gives RMS its name.

The execution time of each task cannot be so long that it causes itself or any lower-priority tasks to miss deadlines. Upper bounds for each task's execution time can be determined such that the entire system will be guaranteed to not miss any deadlines as long as all tasks stay within their allotted execution time.

Applying RMS to real-time simulation must involve consideration of the following elements:

- Task switch overhead must be accommodated in developing timing requirements
- Data must be passed among the tasks in a properly timed manner
- Extrapolation and interpolation may need to be performed on data passed between different-rate tasks
- I/O operation timing must be considered when selecting integration time step sizes
- State variable integration must be coordinated with the intertask data transfers

If these items are properly handled, RMS can provide significant benefits in multiframe real-time simulations. Multiframe simulation is used when subsystems must be simulated at different frame rates. Using RMS allows these different frame rates to run on a single CPU while I/O operations occur at the correct times for each different frame rate.

If the RMS-based multiframe simulation is carefully designed, it should be possible to move tasks to different CPUs with relative ease as new CPUs are added to the system. Tasks can be relocated among CPUs to provide optimization through load balancing.

RMS is a tool that developers can use to implement high-performance, real-time simulations that make the best possible use of available resources.

It isn't unusual for an HIL simulation to be capable of generating and storing 100K of data per second of operation. What do you do with all this data?

Simulations can produce massive amounts of data. It isn't unusual for an HIL simulation to be capable of generating and storing 100K of data per second of operation. If this simulation is in operation for a full day, it will produce several gigabytes of data. What do you do with all this data?

A plan must be in place for performing data reduction, analysis, and archiving. Users who request test runs of the simulation must specify what types of data output they require. All data that may be of interest at a later date must be archived by the simulation operation staff.

Data analysis and plotting programs are the most general tools used to examine the raw simulation output data. More specialized automated tools can be a great help for generating summary reports and other tasks, such as scanning the data to make sure tolerances aren't violated.

Configuration management must be part of the process from the beginning. The software that is used in the HIL simulation must be maintained under adequate configuration management at all times. Making any changes to the software in such a way that the change cannot be undone is simply an unacceptable scenario.

Sometimes an anomaly will occur in a test situation that didn't occur during a similar test in the past. Of course, in the time since the previous test, both the system under test and the simulation have most likely undergone significant changes. If the simulation software has been under configuration control, it will be possible to restore the software version used in the previous test and run it again. In attempting this, a problem may arise if the hardware interfaces or embedded system communication protocols have changed between the two simulation versions. In this case, additional effort will be required to identify the source of the anomaly.

Reasonably priced software configuration management tools are available that make the process relatively painless. There really is no excuse for not having your simulation under configuration control.

Consider having the users of the results provide a simulation accreditation. As previously mentioned, verification and validation of the simulation are critical steps in the simulation's development. Performing an additional step called accreditation often makes sense. An accredited simulation has had its verification and validation tests developed based partially on inputs from the ultimate users of the simulation results. Those users then examine the results of the verification and validation tests and have authority to approve the simulation as demonstrating acceptable fidelity.

Involving the expert users in the verification and validation process can provide powerful feedback that results in an improved simulation and more extensive simulation usage than would otherwise occur. Accrediting the simulation can provide crucial "buy-in" from people who might otherwise be dismissive when presented with simulation results that don't match their expectations.

A reasonable simulation?

The development of an HIL simulation can be a complex and time-consuming process. Getting system experts to accept the results of an HIL simulation as valid is often a formidable obstacle. A simulated operational environment will never be a perfect representation of the real thing. Given the costs and potential difficulties involved, when does it make economic and technical sense to develop and use an HIL simulation?

An HIL simulation is a cost-effective and technically valid approach in the following situations:

- When the cost of an operational system test failure may be unacceptably high, such as when testing aircraft, missiles, and satellites
- When the cost of developing and operating the HIL simulation can be saved by reductions in the number of operational tests. For example, this may be the case with an automotive antilock brake control system that must be tested under a wide variety of operator input and road surface conditions
- When it is necessary to be able to duplicate test conditions precisely. This allows comprehensive regression testing to be performed as changes are made to the SUT
- When it would be valuable to perform development testing on system component prototypes. This allows subsystems to be thoroughly tested before a full system prototype has even been constructed

HIL simulation is a valuable technique that has been used for decades in the development and testing of complex systems such as missiles, aircraft, and spacecraft. By taking advantage of low-cost, high-powered computers and I/O devices, the advantages of HIL simulation can be realized by a much broader range of system developers. A properly designed and implemented HIL simulation can help you develop your products faster and test them more thoroughly at a cost that may be significantly less than the cost of using traditional system test methods. esp

Jim A. Ledin, PE, is an electrical engineer in Camarillo, CA. He has worked in the field of HIL simulation for the past 14 years. He is a principal developer of HIL simulations for several U.S. Navy tactical missile systems at the Naval Air Warfare Center in Point Mugu, CA. He can be reached by email at jledin@ix.netcom.com.