Wind River® Simics creates a shared platform for software development by simulating your
full target system. A Simics simulation of a target system can run unmodified target software
from the physical target system (the same boot loader, BIOS, firmware, operating system,
board support package (BSP), middleware, and applications) while still inside a simulation
framework.

Simics targets are flexible and configurable, freeing developers from the restrictions of
physical hardware. With Simics, you can share work and collaborate between teams and
across continents, perfectly reproduce failures, and non-intrusively debug and inspect even
the most complex system.

FULL SYSTEM SIMULATION
Wind River Simics is capable of simulating any size or complexity of computer system, from
a single processor to a complete board, rack of boards, or distributed system such as an
avionics system, data network, industrial control system, or mobile phone network.

TABLE OF CONTENTS
Full System Simulation ................. 1
Usage Across the Product
Development Lifecycle ................. 3
Design .................................. 3
Develop ................................. 4
Integrate and Test ...................... 8
Deliver ................................. 10
Maintain ............................... 10
Characteristics and Features ....... 11
Performance .......................... 11
Fidelity ................................. 11
Scalability ............................. 12
Off-the-Shelf Model Library ....... 12
Full System Stop ...................... 12
Checkpoints ......................... 12
Run-to-Run Repeatability .......... 13
Reverse Execution .................. 13
System Visibility and Control ..... 13
System Panel ......................... 14
Fault Injection ....................... 14
Scripting and Automation ......... 14
Source Code Debug ................. 15
OS Awareness ....................... 15
Persistent Assets .................... 15
Wind River Simics Product Family . 15
Simics Hindsight ..................... 15
Simics Analyzer ...................... 16
Simics Model Builder ............... 17
Simics Extension Builder .......... 18
Simics Accelerator ................. 18
Simics Models and Virtual Platforms 18
The core of a Simics simulation is a fast instruction set simulator for the target processor. Simics also simulates the memories, peripheral devices, interconnects, disks, and networks, building a true full system model that is indistinguishable from the real thing as far as software is concerned. The simulation is fast enough to run real software loads and can scale to simulate very large systems using multi-core hosts and even clusters of hosts.

![Figure 1: Simulate your entire target system](image1)

![Figure 2: Simulate target systems at all levels of complexity](image2)
USAGE ACROSS THE PRODUCT DEVELOPMENT LIFECYCLE

Wind River Simics virtual platforms provide flexibility and acceleration for all phases of the product lifecycle (see Figure 3).

Activities such as board bring-up, system integration, and system testing can begin before physical hardware is available. And Simics offers unique capabilities such as early and continuous system integration; faster design prototyping by utilizing virtual prototypes instead of physical prototypes; and architectural analysis by running what-if scenarios and trying multiple hardware and software alternatives before committing to one. The result is shown in Figure 4: Projects take less time to run and run smoother, with less risk and higher final product quality.

Because Simics completely controls hardware and time, developers can adopt seemingly impossible debug tricks and approaches that yield substantial improvements in development and debug cycles.

**Design**

When system architects use traditional hardware-based approaches, it may take several weeks or months to determine the optimal configuration of the system. They rely on spreadsheet analysis and benchmarks run on various reference boards (assuming they are available from the manufacturer) but usually omit one critical step: an analysis using the legacy software that is often available. This omission usually contributes significant risks to the architectural analysis and is often not uncovered until late in the development process. Another risk of the traditional approach is that very new technologies are sometimes not considered because hardware prototypes are not yet available from the manufacturer.
When the same investigation is performed using Simics, these fundamental questions can be answered in days instead of weeks. Developers using virtual platforms have no need to redesign hardware, purchase and solder parts, apply power, and then connect them all together. Instead, they can quickly redefine the number of processors on each board, the number of boards, and even the type of board by simply modifying Simics configuration files or scripts. These new custom virtual platforms can be quickly networked together, and real system software loads can be executed and evaluated. Custom hardware devices can be prototyped and tested with driver code to make sure software can take advantage of the features it offers, getting the hardware/software interface right the first time in silicon.

Simics reduces the risks of this phase in several ways. First, architects can evaluate multiple options in the time that only one option could be evaluated using traditional methods. Second, by leveraging real software stacks, the what-if analysis becomes real and not just theoretical. Third, considering the impact of any change on the full system at the time of the architectural analysis reduces the risk of finding problems during system integration. Fourth, design integrates both hardware and software and allows collaboration across the hardware/software boundary.

**Early Customer Feedback**

Because virtual platforms can be created and shipped much more quickly than physical hardware, silicon and system vendors can use Simics to create a virtual platform model, build software, and then demonstrate the result to the customer. This approach allows the vendor to receive early feedback on the proposed features and implementation, allowing changes to be incorporated into the final shipping product.

**Develop**

**Low-Level Code**

During traditional development, the software teams writing low-level software like boot loaders, BIOS, and drivers are usually sitting idle until the first prototype boards are available. There may be some limited collaboration between the hardware and software teams before this point regarding what the capabilities of the hardware need to be. The hardware team may write a specification about what the hardware interfaces are, but many times this specification is incomplete or inaccurate. The risk of this approach is that several iterations of physical hardware prototypes and design changes may be needed, and the software team may waste time trying to figure out how the board works once they are in the critical crunch that follows hardware availability. In addition, development scaling out is dependent on the software developers’ access to target hardware.

With Simics, collaboration becomes far more effective. The hardware teams can provide the software teams with virtual platforms long before hardware prototypes are available. As the hardware design evolves, the software developers get access to progressively more complete models of the hardware. At each step, low-level software can be made to run on the system, enabling application development and even system integration long before the hardware system is completely designed. Thus, software and hardware development can proceed in parallel.
Figure 5: Iterative building of a model

Since Simics models are software, they can be provided in any amount, removing the hardware bottleneck even for low-level software development. An OS port to a new architecture or the creation of a BSP can be planned and executed without the risk of hardware availability interfering.

Figure 6: Single-step interrupt handler
Simics can also debug OS-level code in a way that no other tool can. Simics can single-step any code forward and backward (see Figure 6), break on hardware accesses and exceptions, and inspect the memory management unit (MMU) state and the precise state on any processor. Simics can investigate both virtual and physical memory contents. And Simics allows debug to be performed without access to hardware, which is normally a gating factor for low-level debug. Debug is possible before an ICE, JTAG, or serial connection to the target is available.

Platform Development

Once a target system boots its basic software, it is time for platform development, where a particular operating system stack is integrated with the hardware, middleware is added, and the whole hardware-software platform is made ready for application development. In this phase of a system development project, Simics provides a stable platform that is available in any amount to all developers. It is easy to stub parts of the platform or implement workarounds for hardware that is not yet quite done, or for software components which are yet to be ported properly to the new hardware. Simics also makes it possible to integrate pieces of an old platform with pieces from a new platform, and validate that the combination of different hardware and software versions works as intended.

For really new hardware, Simics can break the cyclical dependency between the OS kernel and the BSP. To get an OS kernel running on hardware, you need drivers. But to write the drivers, you need the OS kernel. Using Simics, the OS kernel can be brought up using very simple drivers that would never work on physical hardware, and using loading over a backdoor. It is even possible to start the operating system without basic services such as interrupts or timers.

Furthermore, by making it easy to trace the interaction between software and hardware in a system, Simics helps resolve conflicts between software and hardware teams over where the root cause of a nonfunctioning system is to be found—reducing the typical rounds of discussion about where a problem is located. With a trace, just compare the operations performed to the specification of the hardware and see if the hardware is misbehaving or the software is programming it in the wrong way. The virtual platform provides a common point of reference for the hardware and software designers to discuss the issues.

Virtual platform models can also issue warnings about suspicious activity that do not cause exceptions or hardware errors in the target system. Quite often, driver coding mistakes are masked by the behavior of the hardware. For example, although accessing nonexistent programming registers in a device is typically considered harmless, a virtual platform will warn about this, making the code cleaner and avoiding crashes when the next-generation hardware starts to use the previously reserved registers or bit fields for something unexpected. Device models can also warn about bad configuration values written to registers, or inconsistent values in different registers, which on hardware results in arbitrary behavior or silent failures.
Finally, a platform team can use Simics to deliver known-good platform setups to application and middleware developers. Using Simics target setups, scripts, and checkpoints, it is possible to deliver an identical ready-to-use setup to all software developers. It is even possible to deliver target systems that have already been booted and configured, so that all a software developer needs to do is to add their applications and run them.

**Application Development**

Many application developers use substitutes for their software development, debugging, and testing tasks because of bad experiences using target hardware that is flaky or unavailable, or working with temperamental embedded electronics. Taking this approach is fraught with risks, especially during system integration. Since in such a case application developers use a different toolchain (including compiler/linker/libraries) and run their code on a different target architecture, many problems do not appear until the code is compiled for the real target and integrated onto the target system. Because this activity occurs late in the development process, problems found at this stage can have huge repercussions on the overall schedule and product reliability.

Simics enables application developers to develop using the real target from the start, with the same toolchain, libraries, and operating system API. In addition to eliminating the risks of late integration, development time is significantly reduced. Simics saves developers the time and effort traditionally spent getting access to hardware, and eliminates the problems that cause application developers to move off of the target hardware. For example, with traditional development it might take time to set up a certain complex configuration in a lab; there is likely a shortage of boards to go around; and as hardware is being developed, it is rare that all developers have identical setups. Furthermore, shipping hardware to a geographically distributed development team is expensive and can take inordinate amounts of time (e.g., for customs clearance). As developers switch between targets, they often have to find the next type of board they need.

With Simics, developers can work more efficiently because they spend less time on unnecessary friction and more time on the actual target. There is an unlimited amount of target hardware for everyone, it is reliable, and it doesn’t require a lab setting to use. In case all that is needed is the right processor architecture with the right operating system, the Simics Quick Start Platforms provide ready-to-use platforms with known-good software stacks to get application developers productive without any dependence on hardware availability.

**Software Debugging**

Debugging is a very time-consuming part of software development, often consuming more than half the development time. Debugging is difficult on modern systems, since there are many moving pieces and concurrent actions, and bugs often only manifest themselves occasionally. Once a bug has been found, reasoning backward to the root cause can be laborious. Simics removes luck and variability from the equation by making it trivial to repeat a particular broken system execution (bug) and providing much more powerful insight into and control over the target system. Some Wind River customers have struggled for months trying to repeat and isolate bugs on physical hardware only to find them in hours with Simics.
Once a bug has been observed in an execution on a Simics virtual platform, it can be reproduced any number of times at any time or in any place in the world. Simics makes it possible to transport bugs with guaranteed replication, saving developer time and ensuring that more rare bugs actually get fixed. As shown in Figure 7, Simics checkpoints can capture both the state of a system at a certain point in time or its entire execution history, and developers can add annotations to the system execution to make it easier for the recipient to understand what is going on. This ability eliminates the repetitive back-and-forth between bug reporters and developers in their attempts to understand the precise configuration of the target system where a bug hits, and makes sure that the developers see exactly what the bug reporter saw.

Once a bug is captured, developers can use reverse debugging to work backward from the manifestation of an issue to its root cause. This technique is much more efficient than traditional cyclic debugging, where the target is restarted and the failing scenario re-run over and over again in order to build an understanding of a bug. Simics can step backward and trigger breakpoints in reverse, making it possible to stop at the previous time a variable or memory location changed its value. Simics allows debuggers to continue from finding the bug directly to debugging and unearthing the cause of it.

**Integrate and Test**

While its usefulness for OS porting, device drivers, and application stacks is obvious, Simics also provides important benefits to system integration.

Once the basic operating system and software layers are up and running on a virtual platform, system integration and testing can begin. This means that integration testing can start very early in a project, long before hardware or a complete software stack exists. User applications can be developed for the first iterations of an operating system and integrated with other applications, legacy code, and third-party binaries on a virtual platform early in the development cycle.

*Figure 7: Checkpoints capture state and history*

Once a bug is captured, developers can use reverse debugging to work backward from the manifestation of an issue to its root cause. This technique is much more efficient than traditional cyclic debugging, where the target is restarted and the failing scenario re-run over and over again in order to build an understanding of a bug. Simics can step backward and trigger breakpoints in reverse, making it possible to stop at the previous time a variable or memory location changed its value. Simics allows debuggers to continue from finding the bug directly to debugging and unearthing the cause of it.

**Integrate and Test**

While its usefulness for OS porting, device drivers, and application stacks is obvious, Simics also provides important benefits to system integration.

Once the basic operating system and software layers are up and running on a virtual platform, system integration and testing can begin. This means that integration testing can start very early in a project, long before hardware or a complete software stack exists. User applications can be developed for the first iterations of an operating system and integrated with other applications, legacy code, and third-party binaries on a virtual platform early in the development cycle.
Simics makes it trivial to automate hardware and software setups, including configuring multiple target systems or boards in a rack. Using Simics scripts, any number of variants can be automatically created with possibly infinite variation. Loading software, running tests, and collecting test results is much easier with a virtual platform than on hardware. Resetting a target that has crashed or hung is very simple, and there is no need to perform clean-up between tests. Instead, all tests can start from a known-good configuration stored as a checkpoint.

When bugs are detected during testing, test engineers can simply pass a checkpoint with a recording to the developers to reproduce the issues found precisely and unambiguously.

**Early and Agile Integration**

With Simics, system integration can begin solely on virtual platforms, expanding to a combination of virtual and physical hardware, and finally to fully physical hardware as those models, software, and hardware elements become available. Throughout the process, the same toolchain used on physical hardware can be used with Simics.

To quickly get to integration, Simics allows users to implement stubs or dummy models for parts of the system that are not yet ready. For example, a board in a rack can be replaced by a simple simulation that tells the rest of the system it is there and everything is fine, even though it does not really do anything yet.

Because integration is performed so much earlier in the development process compared to traditional hardware-based approaches, the “integration” moves from a single high-risk, high-pressure task that begins largely after all hardware is delivered, to a much lower-risk activity that progresses in parallel with both software and hardware development—that is, continuous integration.

By starting integration testing as early as possible, integration and system testing teams can provide feedback and test results to the software developers early on, just as the software developers are able to provide feedback to the hardware designers. In this way, virtual platforms enable an agile iterative work process that encompasses hardware design, platform software development, application development, and system-level integration and test.

**Corner-Case Testing and Fault Injection**

Because Simics provides complete system visibility with the ability to stop and inspect the system at any time, engineers can perform comprehensive testing for any hardware or I/O corner case or fault condition, such as response to corrupt packets, out-of-range readings, bad disk sectors, or any conceivable hardware failure. Faults can be actively injected into the system, provoking behavior that otherwise would be difficult to detect and analyze and allowing testing of fault handling and recovery code.

Simics also allows developers and integrators to experiment with many more configurations than are possible with physical hardware, to find configuration-related bugs and explore performance scalability long before physical hardware is changed. With a virtual platform, it is simple to change the speed and number of processors and see how software scales. Memory sizes can be scaled up and down. Another board can be added to a system, or networks restructured and reconfigured.
Deliver
In the delivery phase of a project, Simics provides internal and external users with access to a target platform and its software. Depending on the nature of the system and its customers, Simics can be used in many ways.

Enabling a Partner Ecosystem
For large projects and programs, it is common for several different companies to share and work on the same hardware and software platforms as part of subcontractor chains and as providers of distinct parts of a system. For such projects, Simics provides an efficient way to share hardware platforms, software platforms, and integrated systems across company boundaries by enabling subcontractors and partners with virtual models of target hardware, along with any software integrated on it. Software integrations performed in the ecosystem can be delivered back to the prime contractor in the same way, using Simics virtual platforms to quickly communicate both hardware and software setups.

Encapsulating Customer-Specific Configurations
Most complex modular systems are configured according to customer-specific needs. Some customers might need 15 routers, 20 switches, and 10 gateways. Other customers might have just five routers, four switches, and one gateway. Although each of these deliveries uses the same components, the quantity and distribution of subsystems can vary.

While the modular approach provides superior product scalability and customization potential, it can cause support problems, often requiring a dedicated lab of equipment that must be reconfigured to address specific customer issues. Virtual platforms, however, can be quickly reconfigured and scaled without any physical hardware present. Once a specific (virtual) hardware configuration has been produced for Customer Y, it is easy to save, restore, and run that same configuration.

Delivering Customer Training Programs
Simics virtual platforms are an excellent vehicle for the delivery of system-specific training. There is no need to send groups of users to particular labs or to ship special hardware to training sessions. Simics makes it possible to prepare course scenarios in the form of checkpoints. If users manage to break or misconfigure a system, resetting to a clean state is trivial because Simics does not save state changes unless specifically told to. Fault injection, scripting, and checkpoints can be used to set up training scenarios that users have to handle. Scripts can also be used to watch a user and automatically detect when things go really bad or complete correctly.

Maintain
As a system goes into maintenance, the Simics platforms used to develop it still provide value. They make sure that there is always target hardware available, and that configurations can be reliably reproduced. When physical hardware starts to get scarce, Simics can provide a virtually infinite supply of systems.
Aiding Customer Support
With Simics, it is easy to set up a virtual lab and equip a geographically distributed support team with the virtual equivalent of the physical end product. When a customer problem is reported from the field, support engineers can identify the problem in the virtual system and easily collaborate and share both the system and its state with other teams within the support and development organizations.

The final virtual platform models can be saved and archived where they will serve as a template for future product designs, complementing traditional design and user documentation, source code, and more. The documentation archive can contain a virtual version of the full system that is capable of executing the software within the same archive collection.

System Evolution
A Simics platform can be used to evolve a system into its next generation. Starting with the existing system, designers can start to swap out individual components and build up a new system. The existing system thus becomes the starting point for the design of the next generation, looping the development cycle back from the Maintain phase to the Design phase.

For systems with very long hardware life spans, Simics provides ample access to development hardware and test systems. While physical boards tend to be scarce for ten-year-old hardware, Simics can provide any number of virtual boards, and do so for the foreseeable future. The software-visible behavior of a Simics virtual platform is the same today as it was ten years ago, and it will be the same in ten years’ time. This makes Simics ideal to support software-based system upgrades to existing hardware modules, as well as the maintenance and testing of successive releases of the system.

Characteristics and Features
Wind River Simics provides several characteristics and key features that support the product development lifecycle from design to maintenance.

Performance
Simics runs models and virtual platforms fast enough to satisfy software developers who are used to running software on physical hardware. The speed and full system simulation capability of Simics differentiates it from most simulation tools provided by the electronic design automation (EDA) industry. Although these EDA tools are extremely accurate from a hardware perspective, and they can be used to develop low-level initialization and test code, they are too slow to be practical for OS, application, or systems software.

Fidelity
The fidelity and accuracy of the model is sufficient to ensure that the software running on top of that model is unable to distinguish the virtual platform from physical hardware.
Scalability
Simics can use multi-core hosts and multiple host machines to scale up the simulation as the target system complexity increases. Simics memory simulations use page-based swapping, lazy allocation, and zero-page and shared-page elimination to simulate target memories that can be orders of magnitude larger than the host memory. Using idle-time optimization, idle parts of the target system have minimal impact on overall simulation performance.

Simics can model nearly any electronic system, from a single processor with memory and rudimentary I/O all the way up to complex, heterogeneous, multi-board, multiprocessor, multi-core systems consisting of hundreds of processors, devices, and network connections. Simics can simulate systems and complex network topologies. The Simics framework has a hierarchical component system that makes arbitrarily complex systems easy to manage and set up, using the same hierarchy and encapsulation as the physical hardware.

Off-the-Shelf Model Library
Wind River provides a wide range of off-the-shelf models, speeding the time to create a model of any particular target system. There are models of individual hardware components as well as complete machines. The models can be used as is or as the basis for larger or more complex virtual platforms.

Full System Stop
Unlike physical hardware, virtual systems can be completely and synchronously stopped by using a single command. With virtual platforms, the stop includes not just processor cores but peripheral devices and even data in-flight on networks and buses. In this frozen state, developers have full access and visibility to all hardware and software variables. Simics allows the whole system to be single-stepped and reverse-stepped, with no component running and suffering time outs or diverging the system state.

Checkpoints
Simics checkpoints store the complete state of the virtual platform to the host computer disk. When the checkpoint is loaded into Simics, the result is the same target system state as when it was saved. The checkpoint includes the hardware setup (boards, networks, plug-in cards, and other configuration aspects), hardware state, and software state. It contains the contents of memories and disks, the state of processor registers, MMUs, peripheral devices, and network connections. As shown in Figure 7, checkpoints can also contain a recording of the execution of a virtual platform, capturing a slice of time and not just a particular instance. As shown in Figure 8, to help the recipient of a checkpoint understand the context and state of a checkpoint, Simics lets users annotate with comments both the checkpoint file itself, and the execution of a system, documenting how the current system state was arrived at.
Run-to-Run Repeatability

Simics virtual platforms are fully repeatable. This means that any simulation run can be repeated identically at any point in time, on any host, anywhere in the world. Typically this means starting from a checkpoint and re-executing forward. If there are asynchronous inputs needed to drive the execution along a certain path, they are typically scripted or part of the history recording, to ensure they occur the same way each time. The Simics debugger is designed not to interfere with the target execution or target state, and thus allows completely non-intrusive debugging that does not affect the repeatability of the target execution.

Reverse Execution

Reverse execution complements system repeatability and checkpoints by allowing developers to run their systems backward. Breakpoints will trigger as Simics reverses, making it possible to stop on the previous access to a variable or execution of a line of code, just like regular breakpoints stop at the next execution.

System Visibility and Control

Simics provides backdoor access to every register, memory location, or other part of the target state. There are no limitations on what you can access and modify, and all accesses are non-intrusive. Simics makes it possible to inspect normally invisible states such as supervisor-level registers, MMU translation lookaside buffers (TLBs), and internal registers in hardware devices. Memory can be investigated from both virtual and physical addresses.
System Panel
Simics enables users to create a virtual representation of their physical hardware’s user interface, including switches and displays, and connect it to the Simics virtual platform. This ability allows familiar user interaction with the virtual platform. Figure 9 shows an example of this.

![Figure 9: Simics system panel provides a way to expose I/O from the target being simulated, in a manner consistent with the I/O of the physical target](image)

Fault Injection
Virtual platforms can inject faults at any place in a system, at any point during a run. Thanks to scripting and the inherent control over time in the simulator, faults can be programmed to be completely repeatable, removing any random factors from fault testing. Virtual platform fault injection is completely nondestructive, since the only thing affected is a simulation of the target. Faults can change the contents of memory, registers, sensor readings, or network packets. It is possible to cut network connections and virtually pull boards out of racks.

Scripting and Automation
Virtual platforms are excellent automation tools. Since they control all inputs and outputs of the target system, scripting can perform interaction with the system such as reading the serial console output and entering appropriate commands and text according to programmed instructions. Scripts can be used to load software, change the target state, inject faults, set up debug contexts and debug information, set breakpoints, and anything else that can be done interactively.

This ability to automate the system to such a fine level of detail enables the detection and duplication of bugs, system corner case testing, integration and test, creation of advanced training scenarios, and sales demonstrations.
Source Code Debug
Source code debug in Simics can be applied to any code, not just user-level applications. You can debug a BIOS, boot ROM, low-level firmware, device drivers, operating system kernels, hypervisors and their guests, and other code that is usually hard to get a good grip on in a debugger. Using OS awareness, it is also easy to debug individual user-level tasks.

OS Awareness
With OS awareness, Simics supports working with OS-level abstractions such as processes, tasks, and threads. This allows debugging individual threads in isolation and seeing what is running where in the system. Breakpoints and stepping inside a thread will step through the execution of that thread, skipping times when the thread is calling into the OS or is inactive due to OS scheduler decisions. Instrumentation and tracing can be restricted to a single execution context, allowing developers to focus on a small part of the system. Scripts can use OS awareness to automate debug tasks and analyze the execution of a system with full knowledge of the tasks that are running and debug information for these tasks.

Persistent Assets
As Simics is used in a system development project, a series of assets with persistent value will be created. The model itself encapsulates a lot of knowledge about the target system, and it can be brought into use at any point in time, even many years down the line. For hardware with very long lifetimes, such as control systems and aerospace and defense systems, Simics models offer a solution that will be around until the end of the project. Simics models will not rot or break due to wear and tear.

Simics setups can be archived for easy and instant retrieval when a particular version of the hardware and software of a product is needed. Simics checkpoints can capture interesting system setups and states. Simics scripts can be built into very powerful tools that operate on target hardware and software. Simics can turn procedures that otherwise exist in the heads of programmers into repeatable, explicit scripted setups.

WIND RIVER SIMICS PRODUCT FAMILY
Wind River Simics includes a number of products that complement each other to provide a modular solution to meet every customer’s precise needs.

Simics Hindsight
Simics Hindsight is the Simics base product. It is needed to run simulations, and provides the user interface to setup, launch, and control Simics simulations. Simics Hindsight allows developers to load virtual platforms, functional scripts, and checkpoints to start, stop, or resume model execution and run the system forward or in reverse. Simulation session comments and execution recordings are part of the Simics Hindsight product.

Simics Hindsight includes the Simics Quick Start Platforms. A Quick Start Platform is a completely synthetic, simulation-only Simics virtual platform that includes BSPs for popular
operating systems (such as Linux and VxWorks®). Quick Start Platforms enable users to immediately obtain the benefits of simulation for application software development, without needing to wait for a specific virtual platform to become available. They are also extremely flexible and configurable, making them suitable for experiments into hardware configurations and multi-core scaling.

Simics Hindsight includes Ethernet network support, allowing virtual platforms to be virtually networked together. It also includes support for Ethernet real-network connections, allowing virtual platforms to connect to the physical world.

This capability allows developers to communicate with standard network-connected debug agents running on the target system, enabling the use of Simics tools alongside existing development tools (e.g., standard Eclipse, Wind River Workbench, and many others).

**Simics Analyzer**

Simics Analyzer enables analysis and debugging of software applications for any Simics model, from a basic virtual platform running a single processor and memory to a large distributed and heterogeneous system.

While Simics Hindsight provides a hardware-centric view of the system, Simics Analyzer provides a software-context view of the software applications running above the target operating system. All information within Simics Analyzer is given in the context of processes, threads, and software functions. Unlike similar tools on the market that operate against physical hardware, Simics Analyzer requires no instrumentation or other changes to the target code.
The primary functionalities of Simics Analyzer are full system process list, system execution timeline, code coverage, and source code debugger.

The full system process list is built upon Simics OS awareness to provide a “what runs where” view of the system, including an easy-to-understand-and-use list of machines, CPUs, operating systems, and processes and their status. OS awareness currently supports the Wind River operating systems as well as a number of other commercial and open source offerings. The system execution timeline provides a “what runs where and when” view by adding timing information to the full system process list and presenting all information within an intuitive time graph format.

The code coverage function provides “statement coverage” information on the application code running on the virtual platform. This coverage data is written to an HTML file for easy access by any web browser, or to plain text for integration with a different tool or spreadsheet.

The built-in Eclipse source code debugger is a fully featured C/C++ debugger integrated with unique Simics features such as reverse execution, multi-core, and heterogeneous target support.

**Simics Model Builder**

Simics Model Builder allows users to create, modify, and configure any size model or virtual platforms ranging from a simple board with just a CPU and memory to complex systems containing multiple networked, heterogeneous multi-core platforms. Using Simics Model Builder, engineers can quickly create and use virtual hardware prototypes, run what-if scenarios, and investigate system architecture issues.

Although Simics Model Builder allows developers to integrate functional models from almost any language into Simics Hindsight, explicit support is provided for models written in C, C++, SystemC, and Python. IP-XACT artifacts can also be used with an import and export capability.

Simics Model Builder provides DML, a specialized modeling tool that simplifies the creation of fast, functional models of hardware devices. Using a specialized C-based syntax, DML allows device models to be created quickly and efficiently by simplifying the definition of a device’s programming registers while ensuring proper support for device state inspection, check-pointing, and reverse execution. DML allows device models to be created quickly and efficiently in C by allowing developers to ignore how a device is physically implemented and instead concentrate on the behavior of the device.

To protect proprietary models within what would otherwise be an open system, Simics Model Builder allows developers and silicon vendors to integrate their custom models as either source files or object files.
**Simics Extension Builder**

Simics Extension Builder lets users adapt and extend the Simics interface and environment, allowing for expanded use cases, integration into other tooling environments, and customized user workflows. Included is a processor API that can be used to integrate other simulators with Simics such as instruction set simulators (ISSes). The rich simulator and Eclipse APIs provided by Simics Extension Builder allow users to create nearly any type of new extension to Simics. Examples of such extensions include hyper-simulators, connections to new debuggers, text and graphical consoles, real network connections, trace generators, new OS awareness modules (i.e., process trackers), GUI extensions, and statistics or analysis tools.

**Simics Accelerator**

Simics Accelerator ensures that regardless of model complexity, the simulation can run fast enough to satisfy software and systems developers who are accustomed to working with physical hardware.

Simics Accelerator allows the full complement of host-processing and host-memory resources to be applied to running a Simics model. Simics Accelerator distributes the simulation workload across multiple processor cores on a single host platform, or across the processor cores of several network-connected hosts. When Simics Accelerator is employed, the result is a dramatic increase in speed for the simulation of large target systems.

**Simics Models and Virtual Platforms**

In addition to the Quick Start Platforms included in Simics Hindsight, Wind River offers Pre-configured Virtual Platforms (PCVPs), and Customer Specific Virtual Platforms (CSVPs). PCVPs are prepackaged, prebuilt virtual platforms that model silicon vendor reference boards or standard commercial boards. PCVPs are pre-integrated with Wind River operating environments to jumpstart development. CSVPs are customer-specific packaging of standard models into a virtual platform. Both PCVPs and CSVPs can serve as either ready-to-run virtual platforms or as the starting point for customization.

**Simics Model Library**

Simics provides an extensive model library that makes creating a custom model of virtually any hardware system quick and easy. Even if a model of your particular device does not exist, a model of it can be created either by Wind River or by the customer using Simics Model Builder.
Following is a partial list of devices that have been modeled for Simics:

**Target Devices**
- ARINC 429 controllers and devices
- CAN buses and controllers
- Crypto accelerators
- Display adaptors
- EEPROMs
- Flash memories
- Ethernet controllers
- I2C controllers and devices
- Interrupt controllers
- GPIO controllers
- Graphics display units
- Memory controllers, DDR, SRAM
- MIL-STD-1553 controllers and devices
- NOR and NAND flash
- PCI and PCIe controllers, bridges, and switches
- Pattern-matching accelerators
- RapidIO controllers and switches
- Real-time clocks
- SATA and IDE disks and controllers
- SCSI controllers and devices
- SD/MMC flash disks
- Serial devices
- Spacewire controllers and devices
- System controllers
- Temperature sensors
- Timers and watchdogs
- UARTs
- USB devices and disks

**Target Processor Architectures**
- Intel® 8051
- Freescale 68020, 68040
- ARM architecture
  - StrongARM
  - XScale
  - ARM7 (v4)
  - ARM9 (v5)
  - ARM11 (v6)
  - ARM Cortex-A9 (v7)
  - ARM Cortex-A15 (v7)
  - ARM Cortex-R4F (v7)
- H8/300, H8/2000
- Intel architecture
  - Intel 80186
  - Intel 80386
  - Intel 80486
  - Intel Pentium/MMX
  - Intel Pentium Pro
  - Intel Pentium II
  - Intel Pentium III
  - Intel Pentium 4/E
  - Intel Pentium M
  - Intel Core
  - Intel Core 2
  - Intel Nehalem
  - Intel Sandy Bridge
  - Intel Atom
  - Intel Xeon variants
- MIPS architecture
  - MIPS 3K
  - MIPS 4K
  - MIPS 5K/f
  - PMC RM7000
  - PMC E9000
  - Cavium cnMIPS64
  - Netlogic XLR
  - Netlogic ec4400
- Power architecture
  - Freescale e300
  - Freescale e500
  - Freescale e600
  - Freescale e5500
  - Freescale e6500
  - Freescale MPC603e
  - Freescale MPC750 ("G3")
  - Freescale MPC755
  - Freescale MPC74xx ("G4")
  - IBM PPC403
  - IBM PPC405
  - IBM PPC440
  - IBM PPC464
  - IBM PPC476
  - IBM PPC750/FX/GX
  - IBM PPC970/
  - SuperH SH-4
  - SPARC architecture
    - LEON II (V8)
    - UltraSparc II, III (V9)
  - Tensilica Xtensa
  - Texas Instruments C64
    - TMS320C64x
    - TMS320C64x+
  - X86 AMD
    - AMD Athlon
    - AMD Athlon 64
    - AMD Opteron

Wind River is a world leader in embedded software for intelligent connected systems. The company has been pioneering computing inside embedded devices since 1981, and its technology is found in nearly 2 billion products. To learn more, visit Wind River at www.windriver.com.

2015 Wind River Systems, Inc. The Wind River logo is a trademark of Wind River Systems, Inc., and Wind River and VxWorks are registered trademarks of Wind River Systems, Inc. Rev. 01/2015