

## Wind River Simics

### Table of Contents

- Full System Simulation ..... 1
- Solve Complex Problems ..... 1
- Significantly Improves Life Cycle..... 2
  - System Definition ..... 3
  - Board Bring-up ..... 3
  - Application Development ..... 4
  - Software Debugging ..... 4
  - System Integration ..... 5
  - System Deployment ..... 6
- Characteristics and Features ..... 6
  - Performance ..... 6
  - Fidelity ..... 6
  - Scalability ..... 6
  - Off-the-Shelf Model Library ..... 6
  - Full System Stop ..... 6
  - Checkpoints ..... 7
  - Run-to-Run Repeatability ..... 7
  - Reverse Execution..... 7

- System Visibility and Control ..... 7
  - System Panel..... 7
- Fault Injection..... 7
  - Scripting and Automation ..... 7
  - Source Code Debug..... 7
  - OS Awareness ..... 7
  - Persistent Assets ..... 7
- Wind River Simics Product Family ..... 8
  - Simics Hindsight..... 8
  - Simics Analyzer ..... 8
  - Simics Model Builder..... 9
  - Simics Extension Builder ..... 9
  - Simics Accelerator..... 9
  - Simics Ethernet Networking ..... 9
  - Simics Virtual Platforms..... 9
  - Simics Model Library ..... 9
- Professional Services ..... 10
- Education Services ..... 10
- Customer Support ..... 10

can run the complete, unmodified target software stack including the board support package (BSP), hypervisor, drivers, firmware, operating system, middleware, and application (see Figure 1).

Any computer system can be simulated with Simics, from a single processor to a complete board, a rack of boards, or a distributed system such as an avionics system, a data network, an industrial control system, or a mobile phone network (see Figure 2).

The core of a Simics simulation is a fast instruction set simulator for the target processors. Simics also simulates the peripheral devices, interconnects, and memories, 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.

Electronic systems have become increasingly more complex in recent years. They often contain multiple heterogeneous and multi-core processors running multiple software stacks and operating systems. In addition, they are usually part of a larger connected system, via a local bus, rack backplane, internal network, or the Internet. Engineering teams are routinely required to implement ever increasing numbers of features within fixed schedules. This challenge is multiplied when development teams are geographically distributed across the globe.

Wind River Simics is a full system simulator that enables developers to radically alter the way they develop, debug, integrate, and test even the most complex electronic systems to positively impact their business goals such as time-to-market, costs, and product quality.

### Full System Simulation

With Wind River Simics, developers can simulate their target hardware and use it in place of the physical hardware for software development, debugging, testing, system integration, system testing, and so on. This simulation, referred to as a Simics virtual platform,

### Solve Complex Problems

Simulation technology is used by many different industries and for many different purposes such as weather prediction, airframes design, weapons performance analysis, vehicle dynamics, bridge construction, and even for

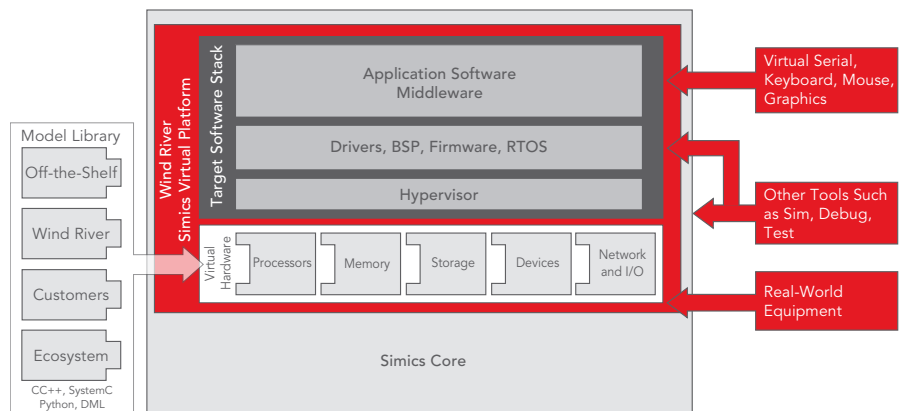


Figure 1: Simulate your entire target system

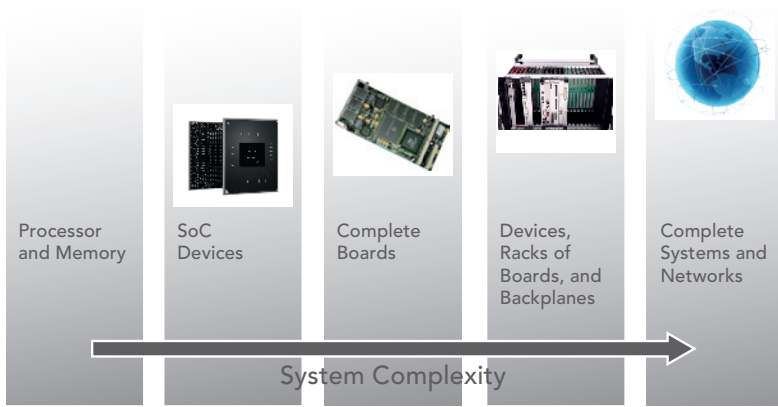


Figure 2: Simulate target systems at all levels of complexity

practice targets for surgeons. Simics brings simulation to embedded systems and software development.

Simulation offers the following benefits:

- **Goof-proof:** Virtual environments are very forgiving of mistakes.
- **Accessible:** There is no need to build, ship, configure, or touch physical hardware.
- **Customizable:** There is easy access to the parameters that determine the details of target behavior.
- **Archivable and restorable:** Models and system states can be archived like any other software or file.
- **Repeatable:** Operational and test scenarios can be precisely duplicated from run to run.
- **Economical:** It is usually less expensive than a physical counterpart and more readily available.

Simics enables product development teams to leverage the power of simulation for the purpose of architectural analysis, virtual prototyping, software development, debugging, testing, and system integration. By using Simics,

product teams can realize the following benefits:

- **Reduce time-to-market:** Optimize the product development life cycle by reducing the following: the time to debug software and system problems; the impact of show-stopper bugs; bottlenecks caused by hardware availability; the number of iterations of hardware prototypes; and the time wasted by inefficient communication across teams.
- **Decrease costs to develop products:** Reduce capital expenditure associated with target hardware and lab equipment costs by replacing physical hardware with Simics virtual hardware. Decrease operating expenses by providing a more efficient debug platform. Reduce the amount of time needed to develop a product by parallelizing many development activities.
- **Improve product quality:** Provide all developers and testers with a platform that is like the real system. Simics can be used to stress the software by forcing hardware, software, and system faults. Virtual platforms can be easily shared by geographically dispersed teams, ensuring consistent setups and facilitating the

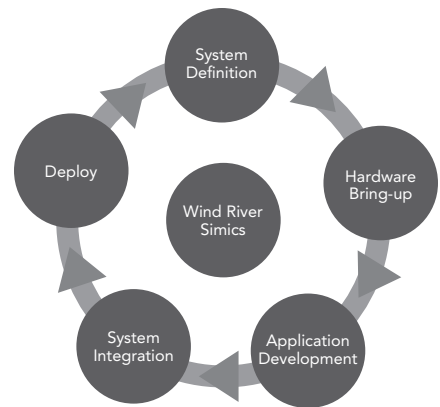


Figure 3: Accelerate product development

exchange of system setups and bug reports. Simics allows automated and parallel testing of systems, cutting test time and thus improving test coverage.

### Significantly Improves Life Cycle

Many of the benefits Simics offers are realized when developers leverage the flexibility the virtual platforms can provide to all phases of the product life cycle (see Figure 3).

Activities such as board bring-up, system integration, and system testing can begin before physical hardware is available. Even though this in itself can improve time-to-market, more substantial benefits are available when developers take advantage of the unique capabilities that Simics offers. These benefits include early and continuous system integration, faster prototyping by utilizing virtual prototypes instead of physical prototypes, and architectural analysis by running what-if scenarios and trying multiple hardware/software alternatives before committing to one. The result is shown in Figure 4:

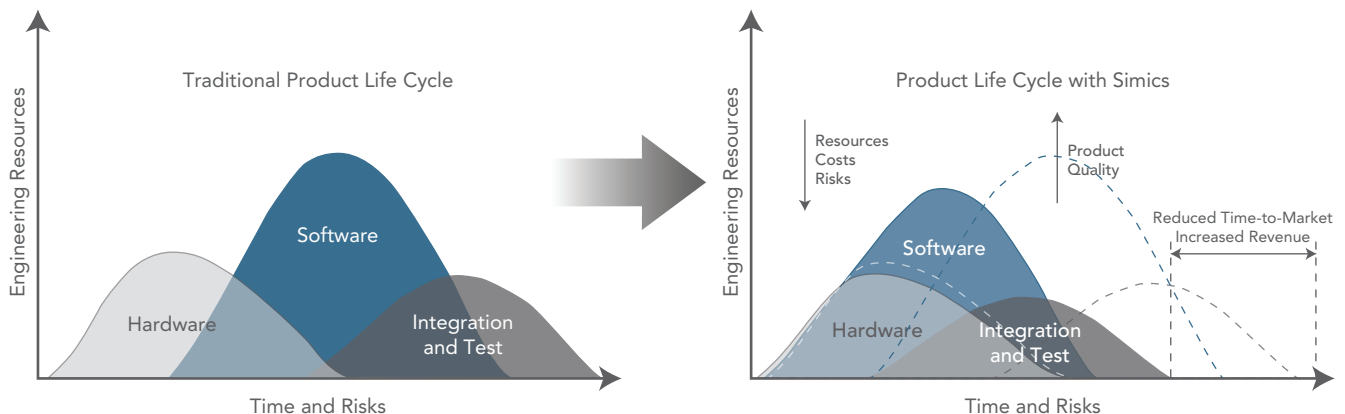


Figure 4: Reduce risks, time-to-market, and costs during product development

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.

### System Definition

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 that 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. Simics also enables engineers to investigate the system impact of low-level hardware or software parameters such as cache, memory speed, buffer, or packet size.

Simics reduces 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.

### Early Customer Enablement

Because virtual platforms can be so quickly created relative to physical hardware, silicon or 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.

### Board Bring-up

During traditional development, the software team writing the low-level software is 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 as to 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, the picture becomes very different. 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 can 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 even completely designed. Thus, software and hardware development can proceed in parallel.

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

Simics can break the cyclical dependency between OS kernel and the BSP. To get an OS kernel running on hardware, you need drivers. But to write the drivers, you

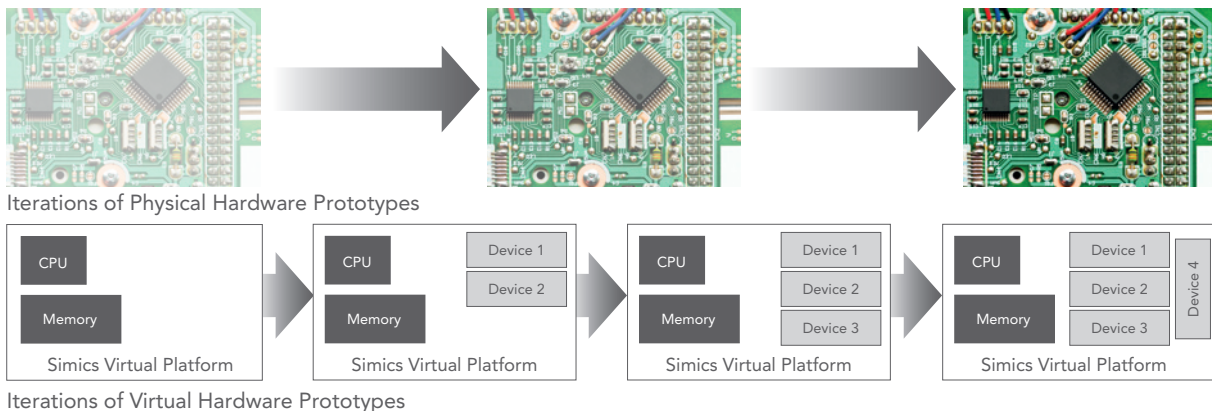


Figure 5: Iterative building of a model

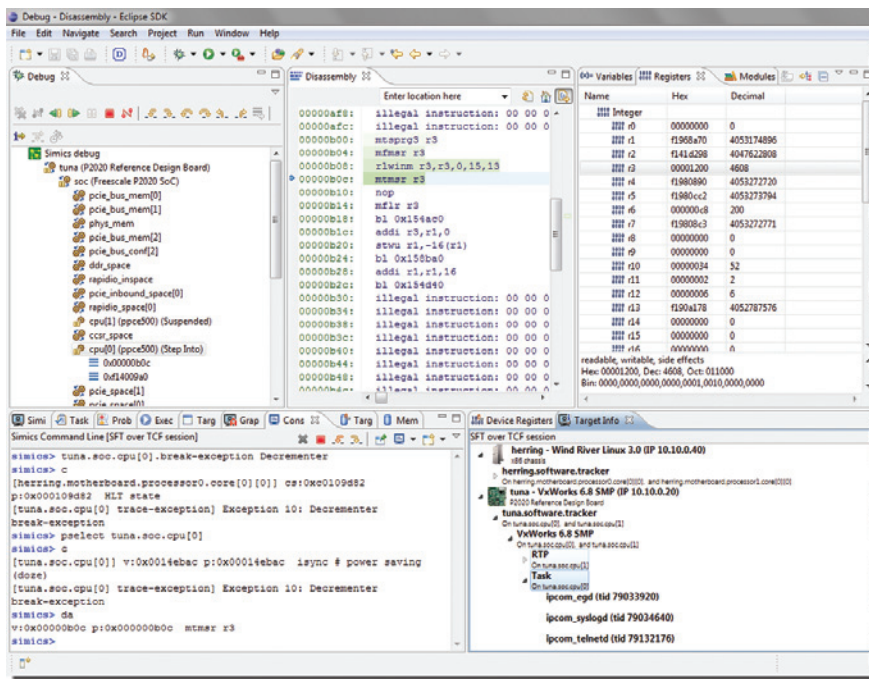


Figure 6: Single-step interrupt handler

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. Another example is integrating different pieces of prototype code with each other because Simics offers a more forgiving environment.

Simics can 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. Debug can also be performed without any access to hardware, which is normally a gating factor for low-level debug. Debug is possible before an ICE, JTAG, or even serial connection is available.

Simics makes it easy to trace the interaction between software and hardware in a system. This makes it simple to resolve a conflict 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 issue warnings about suspicious activity that is not bad enough to force an exception or error in the target system. Quite often errors in code are masked by the behavior of the hardware. For example, accessing nonexistent programming registers in a device is typically harmless because the hardware just ignores accesses it does not understand. A virtual platform, on the other hand, can and 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.

### 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 application developers would use a different toolchain (including compiler/linker/libraries) and run their code on a different target architecture, many problems would 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 and libraries. In addition to eliminating the risks of late integration, time is significantly reduced. Simics eliminates the problems that cause application developers to move off of the target hardware. There is an unlimited amount of target hardware for everyone; it is reliable, and doesn't require a lab setting to use.

Simics saves developers time and effort in getting access to hardware. It might take time to set up a certain complex configuration in a lab. There is likely a shortage of boards to go around. As hardware is being developed, it is rare that all developers have identical setups. Shipping hardware to a geographically distributed development team is expensive and can take inordinate amounts of time (e.g., customs clearance). As developers switch between targets, they often have to find the next type of board they need. With Simics they can work more efficiently because they will spend less time on unnecessary friction and more time on the actual target.

### Software Debugging

Software debugging involves three fundamental activities: provoking the bug, isolating the bug, and fixing the bug. Traditionally, successful debugging requires a high degree of developer skill and experience, often combined with patience and luck.

Simics removes luck from the equation by trivializing efforts to repeat and isolate the bug. Several of Wind River's customers previously struggled for months to repeat and isolate bugs on physical hardware only to find them in hours with Simics.

### ***Provoking and Transporting the Bug***

To repeat the bug on physical hardware, developers may have to restart the system or application hundreds or thousands of times, using a new set of input parameters, data streams, or operator actions each time, or hoping for some random fluctuation that will provoke the bug.

Simics virtual platforms are different. They operate in a virtual world where the entire system state and all inputs are controllable and recordable. As a result, any simulation can be trivially reproduced. Once a bug is seen inside a Simics simulation, it can be reproduced any number of times at any time or any place in the world. Thus, Simics makes it possible to transport bugs with guaranteed replication.

### ***Isolating the Bug***

Once the bug can be reliably repeated, the developer must find the source of the bug. Traditional hardware-centric debug methods require an iterative approach where breakpoints are set, the system is run, registers are reviewed, and the application is restarted or execution is resumed to the next breakpoint. Using this technique, developers can eventually find the precise offending lines of source code.

With Simics, developers can run the system in reverse, watching the sequence of steps that led up to an issue. Simics will trigger breakpoints in reverse, making it possible to stop at the previous change to a variable or memory location. Such an approach does away with the need to start the debug session over and over again and try to reproduce a bug and plant different sets of breakpoints. Instead, Simics allows debuggers to continue from finding the bug directly to debugging and unearthing the cause of it.

Simics can observe all parts of the system state and trace all interactions without disturbing the target execution, which means that it is easy to understand just what the system is doing.

### ***Fixing the Bug***

Once a bug has been repeated and isolated, the effort to resolve it may range from trivial to extensive. With Simics, developers may apply standard features such as check-pointing, reverse execution, run-to-run repeatability, and full-system visibility and control while finding the precise bug fix.

### ***System Integration***

While the use of Simics for OS porting, device drivers, and application stacks is rather obvious, it also provides large 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.

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.

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.

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 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.

### ***Variation Testing***

Simics allows developers and integrators to experiment with many more configurations than 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.

### ***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 corner-case testing for any hardware or I/O 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. When bugs are detected during these tests, the test engineer has only to freeze the system and send the checkpoint file to the software engineering team for guaranteed bug duplication and correction.

## System Deployment

In the deployment phase, 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.

### *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, all without any physical hardware present. Once a specific (virtual) hardware configuration has been produced for Customer Y, it is trivial to save, restore, and run that same configuration.

### *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.

### *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 and scripting can be used to set up training scenarios that users have to handle.

### *Improving Documentation*

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.

## Characteristics and Features

Simics provides several characteristics and key features that support the product development life cycle from definition through development and deployment.

### 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 simulation power 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 many times bigger than host memory. Using idle-time optimization, idle parts of the target have minimal impact on 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 of systems and complex network topologies.

### 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 customized 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, with no part running away.

## 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, memory management units (MMUs), peripheral devices, and network connections. It also stores some core Simics state, such as the current time and events queued for later execution, allowing the virtual platform to continue its execution seamlessly from a checkpoint.

## 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. If there are asynchronous inputs, they are typically scripted or recorded to ensure they occur the same way each time.

## 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 to what you can access and modify, and all accesses are nonintrusive. Simics makes it possible to inspect normally invisible state 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.

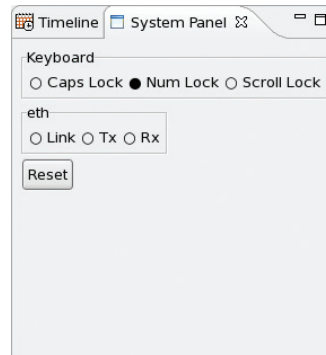


Figure 7: Wind River Simics system panel

## System Panel

Wind River Simics provides the ability for users to create virtual representations of their physical hardware's user interface including switches and displays and connect it to the Simics virtual platform. This provides familiar user interaction with the virtual platform. Figure 7 shows an example of this.

## 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 boot ROM, low-level firmware, device drivers, operating system kernels, 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

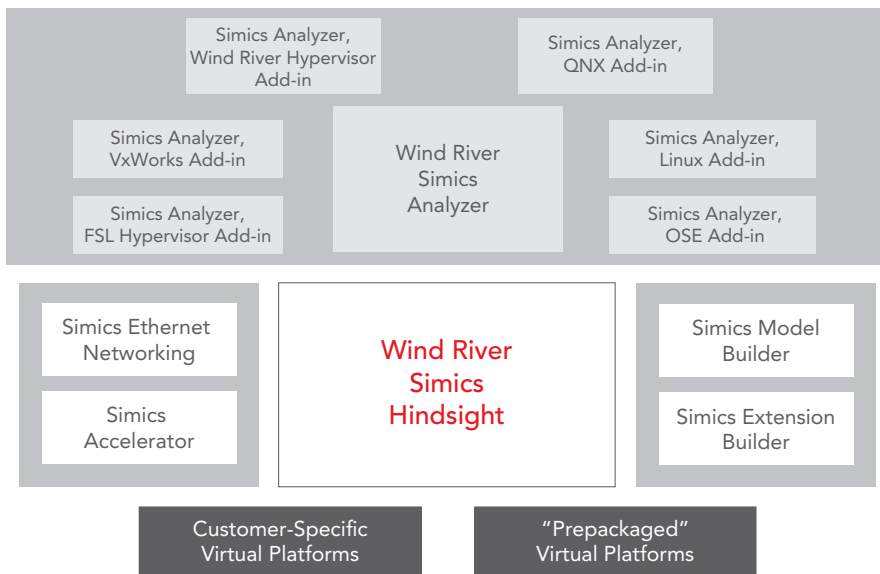


Figure 8: Wind River Simics product family

it can be brought in to 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 setup 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

Wind River Simics Hindsight is the core element of Simics. It provides the user interface to Simics to instantiate and control the virtual system environment

consisting of anything from individual boards, backplane connected chassis, or subsystems, all the way up to a complex, network-connected system. Simics Hindsight instantiates the virtual system environments in which virtual platforms will run, themselves executing real operating systems and application stacks.

Simics Hindsight allows developers to load virtual platforms, functional scripts, or checkpoints to start/stop/resume model execution and run the system forward or in reverse. With Simics Hindsight, developers can set breakpoints and watchpoints, inspect any hardware element, and use the Performance Monitor.

With support for standard networking capabilities, developers can communicate with standard network-connected debug agents. This capability enables developers to use the Simics Hindsight debugger in tandem with their preferred development tools (e.g., standard Eclipse, Wind River Workbench, and many others).

### Simics Analyzer

Wind River Simics Analyzer provides analysis and debugging capabilities 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 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 time line, 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, OSEs, and processes and their status. OS awareness currently supports Wind River's operating systems as well as a number of other commercial and open source offerings. The system execution time line 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

Wind River Simics Model Builder allows users to create, modify, configure, and control 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

Wind River Simics Extension Builder lets users adapt and extend Simics with new modules, allowing for customizing, expanding use cases, and integrating into other tooling environments. Included is a processor API that can be used to integrate other simulators with Simics such as instruction set simulators (ISS).

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

Wind River 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 Ethernet Networking

Wind River Simics Ethernet Networking provides connectivity between virtual systems running inside a simulation as well as the ability to connect the virtual systems to physical equipment that is accessible via the Simics host computer.

### Simics Virtual Platforms

Wind River offers prepackaged, prebuilt virtual platforms that are useful for evaluating new target architectures as well as providing a jump-start to creating a custom model. Wind River also provides the ability for customer-specific virtual platforms to be created.

### Simics Model Library

Wind River 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 a customer using Simics Model Builder.

Here is a partial list of devices that have been modeled for Simics.

#### Target Devices

- Memory and system controllers
- Interrupt controllers
- DMA controllers
- Ethernet controllers
- Serial ports
- Timers
- I2C controllers and devices
- PCI and PCI Express controllers, bridges, and devices
- RapidIO controllers and devices
- USB devices and disks
- SCSI controllers and devices
- FireWire controller and devices
- MIL-STD-1553 masters and remote terminals

## Target Processor Architectures

- Power Architecture
  - Freescale e300
  - Freescale e500
  - Freescale e500mc
  - Freescale e600
  - Freescale e5500
  - Freescale MPC603e
  - Freescale MPC750, MPC755 ("G3")
  - Freescale MPC74xx ("G4")
  - IBM PowerPC 403 core
  - IBM PowerPC 405 core
  - IBM PowerPC 440 core
  - IBM PowerPC 464 core
  - IBM PowerPC 476FP core
  - IBM PowerPC 750FX/GX
  - IBM PowerPC 970, 970MP
- SPARC
  - SPARC-V8
  - SPARC-V9
- Renesas
  - H8S microcontroller
  - SuperH SH-4
- Texas Instruments
  - TMS320C64x
  - TMS320C64x+
- Motorola 68000
  - Freescale 68040
- Intel and AMD
  - Intel 80386
  - Intel 80486
  - Intel Pentium
  - Intel Pentium MMX
  - Intel Pentium Pro
  - Intel Pentium II
  - Intel Pentium III
  - Intel Pentium 4
  - Intel Pentium 4E
  - Intel Pentium M
  - Intel Core
  - Intel Core 2
  - Intel Core i7 (Nehalem)
  - Intel Xeon variants
  - Intel "Sandy Bridge"
  - Intel Atom

- AMD Athlon
- AMD Athlon 64
- AMD Opteron
- MIPS
  - MIPS 4K
  - MIPS 5K
  - PMC RM7000
  - PMC E9000
  - Cavium cnMIPS64
  - RMI XLR MIPS64
- ARM
  - StrongARM
  - XScale
  - ARM7 (v4)
  - ARM9 (v5)
  - ARM11 (v6)
  - ARM Cortex-A9 (v7)
- Tensilica
  - Xtensa

## Professional Services

Wind River Professional Services, a CMMI Level 3–certified organization, enables you to reduce risk and focus on development activities that add value and differentiate your design. Our team delivers device software expertise within structured engagements that directly address key development challenges and contribute to the success of our clients. Our track record of timely delivery and in-depth understanding of market and technology dynamics makes Wind River a valuable implementation partner for clients worldwide.

Wind River Professional Services offers a variety of services around Wind River Simics to meet your needs, including installation and configuration, tool integration, deployment best practices, and virtual platform modeling.

## Education Services

Education is fundamentally connected not only to individual performance but to the success of your project or your company. Lack of product knowledge can translate into longer development schedules, poor quality, and higher costs. The ability to learn—and to convert that learning into improved performance—creates extraordinary value for individuals, teams, and organizations. To help your team achieve that result, Wind River offers flexible approaches to delivering Wind River Simics product education that best fits your time, budget, and skills development requirements.

## Customer Support

Wind River provides full technical support for Wind River Simics at centers worldwide. Support is also available 24 hours a day at our Online Support website ([www.simics.net](http://www.simics.net)) or by email at [support@windriver.com](mailto:support@windriver.com).

For more information about Wind River Simics, visit [www.windriver.com/products/simics](http://www.windriver.com/products/simics).