A FULLY VIRTUAL MULTI-NODE 1553 BUS COMPUTER SYSTEM

Jakob Engblom, C.W. Mattias Holm
Virtutech AB, Norrtullsgatan 15, SE-11327 Stockholm, Sweden,
Email: jakob@virtutech.com, holm@virtutech.com

ABSTRACT

This paper describes a simulation setup consisting of multiple LEON-based boards connected by a 1553 bus, simulated using the Virtutech Simics simulator framework. The simulator is complete enough to run unmodified target binaries from the real target system, and fast enough to make simulation of very long runs feasible.

Such a simulation environment can be used as an alternative to development cards and prototype hardware to perform software development, software testing, fault-injection, and other software engineering and quality assurance tasks.

This paper discusses the simulation technology, target system, achieved performance, and uses of the technology.

1. INTRODUCTION

A *full-system simulator* is a software program simulating computer hardware with sufficient detail to run an unmodified software stack including device drivers and operating systems.

Our simulation framework, *Virtutech Simics* [1], is capable of simulating large systems and complete networks. The simulation approach used is full-system simulation, where the processor, memories, devices, and environment of a target computer system are all simulated in such detail that the target software cannot tell the difference to a real target system.

In this particular case, the target processor is a SPARC V8 processor, with the device set from an on-board computer system used in European satellite systems. The simulation of this system is run on top of a regular x86-based Linux or Windows PCs, or SPARC-based Solaris systems.

2. TARGET SYSTEM

The target system that we simulate is an on-board computer processor board from a commercial vendor, which has been used in several European space missions. For the purposes of this work, a subset of the functionality has been implemented, sufficient to boot an operating system and run some test programs for the 1553 bus.

As illustrated in Figure 1, the main components of the board is a LEON2-based unit and a custom IO unit (in the actual hardware, these two are implemented on a single chip). The LEON2 has hardware floating-point support.

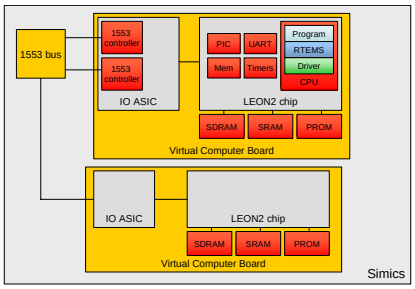

Figure 1: Overview of the simulated target system

The LEON2 processor core model used implements integer instructions, floating-point instructions, and the memory-management unit. Attached to the processor, there are some basic system functions like serial ports, timers, and the memory controller. The device set is sufficient to boot the RTEMS operating system [2].

The IO ASIC model is currently limited to the 1553 bus. There are other IOs of interest for simulation, for example Ethernet and SpaceWire. Simics has an Ethernet simulation available, but we need to implement the Ethernet controllers used on this particular ASIC to connect simulated Ethernet to the simulated boards.

The simulated boards are connected using a simulated 1553 bus, implemented on the message level. The bus is explicitly represented as an object in the Simics simulation system. This *bus object* makes it possible to inspect and modify the traffic on the bus without accessing the individual bus controllers attached to it.

Any number of the boards can be instantiated in the simulation and connected to the 1553 bus (limited by the addressing of the 1553 bus, but not by the simulator itself).

The operating system used in our setup is RTEMS. RTEMS uses the same board support layer as used with the real target, and there is no need to compile an application specifically for testing with the simulation.

The same device drivers for serial, 1553, etc. are used as with the real hardware. We have successfully run binaries provided to us by the board vendor, without any change to the binaries.

3. SIMULATION TECHNOLOGY

Simics is a full-system simulator designed for maximum execution speed, primarily intended to facilitate executing and developing software for the target system.

In order to achieve speed, a key simplification is the timing of instruction execution. In our basic model, processor instructions are assumed to have a fixed execution time, and device accesses provide simple time models for when transactions complete.

The functional results of instructions and device accesses is identical to a real machine (which is necessary in order to run real binaries), but the timing might be different. Simics is not intended to analyze or predict the precise cycle timing of code execution dependent on processor pipelining. Since building precise timing models of real hardware is very difficult, such detailed timing analysis has to be validated on the real target platform anyhow.

All the devices on the simulated board (and all interconnects between boards in a system) are implemented in a transaction-based style. This means that accesses to devices are handled as a synchronous unit, rather than simulating the actual bus traffic (on the processor and peripheral buses) required to perform the request in the real hardware.

The detailed function of the memory and system buses (the AMBA bus in the case of the LEON2) are not modeled explicitly in Simics. The modeling abstraction used is rather a functional memory map, where read and write operations to an address are directly routed to the memory areas or devices mapped at that address. This can be performed in a very efficient manner, ensuring high simulation performance. All memory operations complete immediately, which is natural in a transaction-level model .

Overall, using this level of abstraction makes it possible to reach simulation speeds of several hundred MIPS for a single board.

For a multi-board setup, it is important to note that Simics provides *a single global virtual time*. All processors and device models are synchronized to this time base, across all the boards in the simulated system. The progress of this time in relation to the real-world time is variable, and can be both faster and slower. Real-time clocks on the simulated systems report this virtual time, not the real time on the simulation host.

The virtual time also makes Simics a *deterministic* simulator. Each time a simulation is run from the same initial state, the same results will ensue, provided that all input and output to the simulator is controlled. It is possible to record interactions with a user or other entity external to the simulator to ensure this property in all circumstances. In the case where the system being simulated is self-contained (for example, driving tests using test scripts within the simulator or testing a boot of an operating system), determinism is automatic even without recording.

Thanks to determinism, it is possible to virtually reverse the execution of a simulated system, allowing for backwards debugging [4] – even for a multi-board, multiprocessor system. Such abilities have been available for a time for development boards, using hardware trace recorders from vendors like GreenHills, IAR, and Lauterbach. With Simics, it can also be done for virtual systems.

It is quite possible for a simulation to run many times faster than the real world. In particular, if a system is mostly idle, simulation can run very quickly. If a system is idling most of the time, simulation of very long executions (several days) can be run in a fraction of the time. Parallel simulation of many different test cases can also be used to reduce the overall execution time for a series of tests [3].

A Simics target system can be easily extended by developing new Simics device models. This can be done in either C/C++, Python or the Virtutech-provided domain-specific modeling language DML (Device Modeling Language).

4. 1553 BUS SUPPORT

The 1553 support is implemented with a standard Application Programming Interface (API) that all 1553 devices in Simics must confirm to. This allows any virtual 1553 device that utilizes the Simics 1553 bus API to connect and talk to any other device.

A generic bus setup is illustrated in Figure 2. Note that it is possible to attach 1553 bus devices which are not part of computer boards, for example independent instruments and ASICs. Also note that an instrumentation module can be attached to the “backside” of the bus, invisible to all connected devices. Such an instrumentation object can both inspect traffic and perform fault injection by modifying or deleting messages on the bus [6].

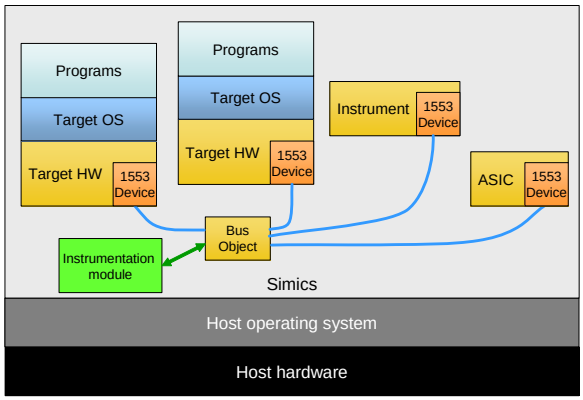

Figure 2: 1553 Bus Simulation

The devices are connected to a link object that distributes the messages to the right destination node. Devices can be bus masters, remote terminals, or bus monitors. When connecting to the bus object, devices state whether they want to see all messages or just messages intended for a particular or a particular set of addresses.

The bus simulation works by having devices pass messages to the bus-object where the messages are given a phase and a payload of data words. The phase has a direct mapping to the phases in the 1553 protocol (command, status, mode command and data word phases) and the payload is the data that is sent in the specified phase. This makes it possible to reconstruct the actual flow of bits over the bus when required, but still provides for fast simulation by working on complete messages. The abstraction into the message level does generate a number of differences between hardware and software, for example the simulation does not implement the Manchester encoding of messages, though it is possible to inject Manchester encoding errors via the link object, another difference is that the sync field is ignored as this is used to specify the phase the message is in.

It is also possible to connect the simulated bus to a real-world 1553 bus, using a 1553 interface card in the host computer. This allows simulated processors to be interfaced to real hardware. However, connecting real hardware to the 1553 bus does at the moment mean that reverse execution is not possible. Though, adapting the real 1553 network connection for reverse execution should pose no problems as this has previously been done for other real network connections.

The current implementation of our simulator only includes computer boards, but we expect that users of the simulation will add models of application-specific hardware to the buses. Long term, it is possible to construct simulations of entire satellite systems, including the computers, navigation instruments, payload, and the environment. Indeed, the 1553 bus model is already in industrial use in such settings, outside of this particular project.

5. PERFORMANCE

Simics uses an on demand dynamic recompilation technique in order to achieve very high performance for the simulated target.

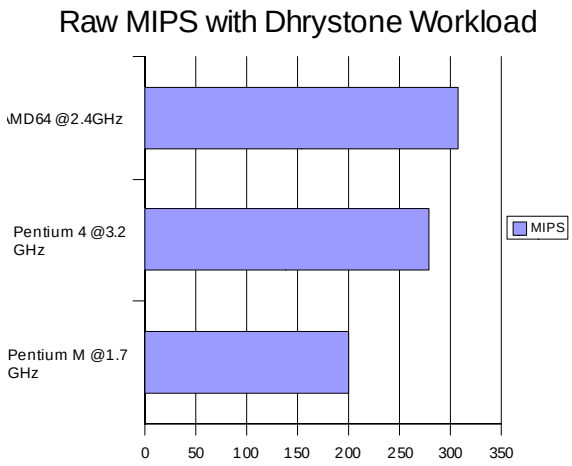

Figure 3: Simics/LEON2 Performance

Programs that put the processor into sleep mode the greater part of the time can execute at a speed of several GIPS. Integer-intense code that puts a 100% load on the processor achieves a performance of over 300 MIPS on a 2.4GHz AMD64 based host machine. On the same host, performance of up to 74 GIPS has been observed on a mostly idle simulated system where basically only the timer interrupts resulted in code execution. This corresponds to about 300 times faster than real-time execution, and means that simulating an hour of real time would take about 12 seconds.

The performance of Simics executing the Dhrystone benchmark program on the RTEMS operating system, in raw MIPS is shown for several hosts in Figure 3. Note that this is the actual number of instructions executed and not the VAX MIPS rating that Dhrystone produce. The VAX MIPS rating is notably higher, since each VAX MIPS requires less than 1 million target instructions to be computed (this is a common phenomenon with Dhrystone [5]).

Floating point performance is typically lower, this come from the fact that the Simics LEON2 processor simulator aims to be bit-accurate in the Floating Point Unit (FPU) simulation, regardless of the host machine. This differs from a number of other simulators that utilize the host FPU in order to simulate floating point operations. Using the host FPU to simulate the target FPU can result in rounding errors and incorrect handling

of floating point exceptions. Since Simics' intentions are to be an instruction level simulator whose behavior is to be impossible to distinguish from real hardware by the software running on it, bit exactness have not been sacrificed for performance.

Note however, that if the operating system running in Simics use software emulation of floating point instructions, the floating point performance will depend on the integer simulation code.

6. USES OF VIRTUAL HARDWARE

Simulated computer systems can be used for many different tasks. The obvious use is to complement the usage of development boards with simulated boards. Simulation usually runs target code faster than the real hardware, and the turn-around time to load new code is shorter. This means that simulation can be used instead of development hardware, without making development cycles longer.

As a system development tool, full-system simulation offers the possibility to build complete simulated space systems, where not only the computer boards but also the rest of the space-bound system and possibly models off the operating environment are included.

Simulation has some practical advantages and even unique capabilities compared to using real hardware, when seen as a software development tool:

- The simulated environment offers a very convenient and powerful software debugging environment, with superior visibility into the state and behavior of the target system, including the device states.
- Reversible debugging (setting breakpoints and executing backwards in time) on the system level is a unique feature not possible without simulation technology.
- Fault injection is very easy to perform in simulation, as simulation provides perfect control over processors, memories, devices, and bus traffic [6][7].
- Regular engineer workstations can be used to run test cases for many different hardware platforms, increasing flexibility and avoiding the need to distribute particular hardware cards.
- The time to execute each test case can be reduced, as configuring and simulated system and loading code onto it is easier and faster than configuring and loading code on a real hardware board.
- A cluster of workstations or a server can be used to execute tests in parallel, which shortens the time to execute test runs and shortens the turn-around time for a new software version.

From a risk-reduction and project-management perspective, simulation has some interesting benefits.

- Money is saved by not having to invest in hardware systems, rather using regular development workstations for target test runs.
- Since virtual hardware can be made available before the real hardware is available (even in prototype form), it is possible to start development of software such as device drivers and operating systems long before actual silicon exist of the target machine [8].
- Simulated hardware is easy to distribute to subcontractors and among geographically distributed development teams. There is no need to ship fragile physical boards around to supply subcontractors with hardware to work on.
- Simulated computers have a very long lifespan, as they are "just software". The simulation environment will not rot or break due to age. This is illustrated in Figure 4. As long as Simics is ported to new architectures, the simulation will keep working, and software maintenance and development work for the target can continue [3].

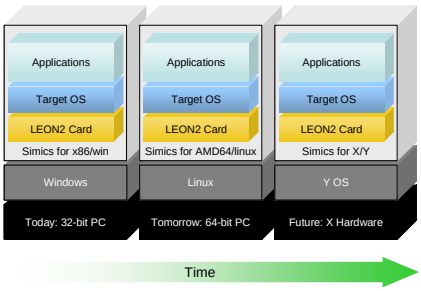

Figure 4: Longevity support with Simics

- Simulated complete systems can also be used to model the faults that accrue in a deployed system over time. This makes it possible to test workarounds and new software versions on the ground, in an realistic environment corresponding to a partially failed system already in place in space [9].

7. FUTURE WORK

Future work on the target system discussed in this paper includes adding additional devices and further optimizations to performance in the core LEON2 system. The 1553 link object can also be modified so that it stores messages coming from physical devices. This would allow for reverse execution, even when the simulation is

connected to a real 1553 bus with real hardware attached.

8. SUMMARY

This paper has presented a virtual LEON2-based on-board computer system capable of running binaries from a real board. The simulator is based on Virtutech Simics, with custom device models corresponding to the real board. It runs the RTEMS operating system and some test code, at a speed of up to 300 MIPS on a regular development PC. Virtual boards have many potential benefits for the development of space-bound embedded computer systems.

9. REFERENCES

[1] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hållberg, Johan Högberg, Fredrik Larsson, Andreas Moestedt, Bengt Werner. "Simics: A Full System Simulation Platform", *IEEE Computer*, Feb 2002.

[2] RTEMS website, www.rtems.org

[3] Jakob Engblom, Guillaume Girard, and Bengt Werner. Testing Embedded Software using Simulated Hardware, *ERTS 2006*, Toulouse, January 2006.

[4] Clive Maxfield. Cool EDA Products light up 2005, *EETimes*, March 13, 2005. (http://www.eetimes.com/showArticle.jhtml?articleID=159403011)

[5] Tom Halfhill. Embedded Benchmarks Grow Up, *Microprocessor Report*, June 21, 1999.

[6] Jonas Myhrman and Pierre Svärd: *Studying Fault Injection in WCDMA Base Station Processors Using Simics Simulator*, MSc Thesis, Chalmers Institute of Technology, Department of Computer Science and Engineering, Göteborg, Sweden, 2005.

[7] Bertrand Bastien: *A Technique for Performing Fault Injection in System Level Simulations for Dependability Assessment*, MSc Thesis, University of Virginia, School of Applied Science, January 2004.

[8] Nick Flaherty. Models solve the software/hardware challenge, *ElectronicsWeekly.com*, May 12, 2006. (http://www.electronicsweekly.com/Articles/2006/05/12/38526/Modelssolvethehardwaresoftwarechallenge.htm)

[9] John Waters. Iridium Simulates Space Software with Simics, *Application Development Trends*, October 31, 2005. (http://www.adtmag.com/article.aspx?id=12020)