Feldstein SPARC-V9 Simulator
Version 1.0
Software Requirements Specification

Revision number: 537 2010-07-13

Table of Contents

1. Problem Statement

1.1 Intentions

Enable cycle-based Boolean logic verification (i.e. functional verification), focusing on the pure digital portion of the SPARC-V9 design, and depending on other tools for the analog portions. Reduce the time and cost of verifying large SPARC-V9 designs, resulting in quicker time-to-market, lower development costs and higher revenue. Deliver the performance required for verifying high-density SPARC-V9 microprocessors in a competitive time frame.

Optimize the bug-discovery rate. Speed up simulation-debug-fix iterations.

Allow verification engineers to concentrate on their primary job of verifying the DUV, rather than concerning themselves with low-level simulator application interfaces and file-management chores.

1.1.1 High-Density Designs

Handle the high-density designs of today and the future. The direction computer architectures will take toward billion-transistor architectures has been debated since at least 1997. Assume a growth rate in transistor count on a chip of 55% per year.

1.1.1.1 *Example High Density Designs of Today and the Future*

1.1.1.1.1 Sun Microsystems, Inc.

1.1.1.1.1.1 UltraSPARC IV

Dual-thread UltraSPARC IV processor has a 66 million transistor count.

As part of its Throughput Computing initiative, Sun has started down the path to chip-level multithreading (CMT) with its UltraSPARC IV, revealed at Microprocessor Forum 2003. Sun's first UltraSPARC IV consists of two slightly enhanced UltraSPARC III processors that share a systems bus, DRAM memory controller, and off-die L2 cache. Built in the Texas Instruments 130nm process, the dual processors share one large off-die L2 cache, but when Sun migrates the UltraSPARC IV processor to 90nm, the company will add an on-die L2 cache, with the off-die cache becoming a large L3 cache.

1.1.1.1.1.2 Niagara

http://www.sun.com/processors/throughput/datasheet.html

In the 2005/2006 timeframe, Sun plans to dramatically improve performance while lowering price by delivering processors that provide 15x the application throughput of current processors.

Network facing Gen. 2 CMT

According to a 2004-02-11 article by Stephen Shankland, Staff Writer, CNET News.com:

[David Yen] said Sun plans to “tape out”, or complete the design, of ... Niagara this spring. It typically takes at least a year after tape-out to debug and test a chip and design new servers using it.

1.1.1.1.1.3 Rock

According to a 2003-02-25 article by Stephen Shankland, Staff Writer, CNET News.com:

Sun will merge Niagara lineage with the UltraSPARC lineage after 2005 in a processor 30 times faster than today's 1.2GHz UltraSPARC III, David Yen said. This new chip "will provide hardware features to do Java acceleration and will extensively utilize asynchronous circuitry," Yen said.

http://www.sun.com/processors/throughput/datasheet.html

Sun's upcoming (beyond 2005/2006 timeframe) multithreaded UltraSPARC processors are designed to boost application throughput up to 30 times.

Data facing Gen. 3 CMT

According to a 2004-04-26 open letter from David Yen, Executive Vice President, Sun Microsystems Scalable Systems Group:

“a commitment by Sun to bring our next-generation Chip Multithreading (CMT) technology processors and systems to market as quickly as possible”

“... unwavering in our commitment to the SPARC/Solaris platform. SPARC and Solaris continue to be our foundation for delivering premier solutions for secure, reliable, and powerful enterprise computing.”

“to better focus our SPARC development efforts, and to hasten the date at which we can deliver far more powerful SPARC/Solaris platforms based on new Chip Multithreading (CMT) technology.“

“we will focus our substantial development resources on ... Rock processor [family]. ... SPARC [design is] based on powerful CMT principles, and fully [supports] our revolutionary SPARC/Solaris Throughput Computing initiative. The ... Rock processors are already well along in development, and our intensified focus will enable us to get them to market as quickly as possible. These ... data facing processors are expected to deliver an order of magnitude gain in throughput performance. We will deliver [this] powerful new [generation] of CMT-based SPARC processors while continuing to protect our customers' investment in software by maintaining binary compatibility.”

“confident that the future of the SPARC/Solaris is better and brighter than it has ever been in its already extraordinary 17 year history”

1.1.1.1.2 Fujitsu Limited

1.1.1.1.2.1 SPARC64 V

According to a 2002-12 white paper on “PRIMEPOWER & Consolidation”:

The open SPARC specification permits implementation choices, for example, incorporating circuitry that increases performance or enhances reliability. For SPARC64 V, Fujitsu has increased the instruction-level-parallelism beyond that of other SPARC implementations. Specialized instruction-fetch hardware alalyses data dependencies and potential instruction parallelism for code segments allowing multiple execution units to remain busy in SPARC64's out-of-order superscalar design.

Many more details on this processor are given in this white paper.

According to a 2003-09-03 Fujitsu Siemens Computers press release:

Fujitsu Siemens Computers, the leading European vendor of secure Busin3ess Critical Computing solutions, has announced a record-breaking data warehousing benchmark result for its Solaris/SPARC-based PRIMEPOWER 2500 server running 64 SPARC64 V processors. Fujitsu Siemens Computers achieved a TPC-H benchmark of 34,345.4QphH@3000GB, setting another world record in industry-leading standards.

1.1.1.1.2.2 SPARC64 VI

According to a 2003-10-15 article by Stephen Shankland, Staff Writer, CNET News.com:

Fujitsu's Takumi Maruyama, manager of the company's E Processor development work, described the next-generation SPARC64 VI, code-named Olympus and due to arrive in the second half of 2005.

Maruyama said the SPARC64 VI is a dual-processor design that will debut at speeds faster than 2.4GHz. It will be built by way of a manufacturing process that uses electronics with features that measure 90 nanometres. (A nanometre is a billionth of a metre; today's high-end chips are built with 130-nanometre features. New manufacturing processes with smaller feature sizes enable chip designers to pack more circuitry onto a chip.)

The SPARC64 VI will also have the ability to run two threads in each processor, Marayuma said. Overall, he predicted performance would increase by a factor of four over the current SPARC64 V generation, which runs at 1.35GHz.

The SPARC64 VI is working in Fujitsu's lab today at 2GHz, Maruyama said.

1.1.1.1.2.3 SPARC64 VII

Further down the road Fujitsu will debut the SPARC64 VII chip, which will have four processors on a single chip and a clock speed that ranges from 5GHz to 6GHz, according to the current roadmap.

1.1.1.1.3 Fujitsu-Sun Integration of High-Performance Servers Operations

A statement from Fujitsu 2003-10-23 said as follows: "Throughout the years, Fujitsu has enjoyed a close partnership with Sun Microsystems and the two companies have had a number of discussions about the benefits of working together to deliver the best solutions to our customers. Those discussions are continuing. At the present time, however, nothing has been decided with respect to expanding the scope of our current relationship with Sun."

1.1.2 Regression Testing

1.1.2.1 150,000 tests could be run overnight

For static verification (i.e. as opposed to random), libraries of architectural verification programs typically would be developed. These libraries are to provide broad coverage of both the architectural and the implementation-specific features of a DUV. A partial set of these tests would be used to regress each model to assess the quality of the DUV delivered. This subset would run overnight.

1.1.2.2 500,000 tests could be run in 3 days

A complete static regression could consist of 500,000 tests. Such a regression would be run against major DUV deliveries.

1.1.3 Average over a billion cycles per day

1.1.4 Rapid Isolation Down to a Single Unit for Most Failing Test Cases

1.1.5 Computational and Non-Computational Instruction Execution Fully Exposed

1.2 System Requirements

A document is required, describing the system requirements for executing the simulator.

1.2.1 Java Virtual Machine (JVM)

1.2.1.1 Comment (not a requirement)

Compilers that translate Java bytecodes (or in some cases the Java source code) into the native machine code of the client's machine have been written for most popular platforms. These compiled programs perform comparably to compiled C or C++ code. However, there are not bytecode compilers for every Java platform, so Java programs will not perform at the same level on all platforms.

1.3 Free Trial Download from WWW

1.3.1 http://www.alanfeldstein.com/

1.3.2 Feature-Limited

1.3.3 End-User License Agreement

1.3.4 Installation instructions

on http://www.alanfeldstein.com download page

1.3.5 tar.gz

1.4 Buy from WWW

1.4.1 http://www.alanfeldstein.com/

Link to http://www.bmtmicro.com/

1.4.2 Full-Featured

1.4.3 End-User License Agreement

1.5 Application

Consists of (simulator + SPARC-V9 Standard Reference Model), (view), and (controller).

1.5.1 Implemented Entirely in Software

Down to the JVM, the application is implemented in software. Implementation of the JVM is beyond the scope of this document.

1.5.2 Implementation language is Java

1.5.2.1 64-bit integer primitive type

1.5.2.2 64-bit floating-point primitive type

1.5.2.3 WORA

Write Once, Run Anywhere

1.5.2.4 JHDL

1.5.2.5 Using a value of type long as an array index results in a compilation error

SPARC-V9 includes a linear address space with 64-bit addressing.

1.5.3 Menu

1.5.3.1 Help

1.5.3.1.1 Help Contents

Describes how to use the simulator.

1.5.3.1.2 About Feldstein SPARC-V9 Simulator

1.6 Simulator

This is the simulator logic (which represents the operation of the SPARC-V9 system).

For the purposes of this section of the SRS, the "simulator" is the encapsulation of the simulator kernel, the HWSystem and its SparcSimulation subclass, and the test bench. The "simulator" does not include any models.

1.6.1 Phase-Based

FSS is like a cycle-based simulator except that, instead of a cycle, it is based on a phase (also known as a step). A step is the time from one changing clock edge to the next, or if there are multiple clocks the time from one changing clock edge until any of the clocks changes.

FSS uses algorithms that eliminate unnecessary calculations to achieve huge performance gains in verifying functionality:

- Evaluates the state of the logic at the end of each phase.
- Zero-delay evaluation of the Boolean logic gates between state elements
- Intra-phase timing is ignored (i.e. complete separation of functional verification from timing verification).

By limiting the calculations, phase-based simulators can provide very large increases in performance over conventional event-driven simulators.

Phase-based simulators only focus on the functionality of the design and therefore can highly optimize the calculations for that purpose.

The sizes of designs that need to be verified are growing rapidly and debug complexity is climbing exponentially. The speed and low memory usage of phase-based simulators offer relief for this increase in

complexity.

1.6.1.1 Performance and memory requirements scale at most linearly with the problem size

1.6.2 Two-Value

Only two logic states (0 and 1) are computed.

Making a JHDL x-assignment to a signal would tell the simulator to treat the signal as having an unknown value and would tell the synthesis tool to treat the signal as a don't care. The synthesis tool would build a gate-level design using *optimized* gates that will not drive an unknown output on the signal. This means there would be a mismatch between pre-synthesis and post-synthesis simulations for all x-assigned signals.

Since this simulator does not allow x-assignment to a signal, the synthesis tool loses a degree of freedom, and as a result cannot build a gate-level design using optimized gates.

1.6.3 Configurable

Can simulate a variety of SPARC-V9 implementations, each one a Design Under Verification. Supports parameterized DUVs.

1.6.4 Simulator Kernel

This is the repository of global circuit information. It implements the basic simulation kernel. It can be accessed outside of the byucc.jhdl.base package by the SparcSimulation.

1.6.4.1 Rank-Ordered Logic Evaluation

1.6.5 HWSystem

This JHDL class defines the top-level node for a circuit. All circuit-level issues, such as cycling, are taken care of in this class.

1.6.6 SparcSimulation

This Feldstein class is a subclass of HWSystem.

1.6.7 Test Bench

An instance of a Feldstein class that implements the TestBench JHDL Java interface is the child of the SparcSimulation instance.

This is the interface between the simulator and the DUV (i.e. the full-chip). Implements all possible system configurations that a SPARC-V9 microprocessor supports. Represents the system and drives the pins of the DUV, providing inputs and checking certain outputs.

1.6.7.1 Models external logic such as main memory, caches, and I/O devices

1.6.7.2 Models external agents such as other processors or system processes (called daemons in UNIX)

1.6.7.3 Models basic pin interface functions such as reset and clocking.

1.6.7.4 Comment (not a requirement)

Writing a test bench in C can be typically much faster than doing the same task in Verilog or VHDL. C has other advantages: the test bench can be portable across Verilog and VHDL; and C programmers who don't have HDL experience can help write and debug models. Transition to C for performance and feature reasons.

1.6.7.5 Automated Verification

Test bench monitors various interfaces or components of the system during simulation. Some automated verification may be performed at simulation time. Produces trace files of other relevant traffic. Trace files are input for post-simulation tools outside the scope of this SRS.

1.6.7.5.1 Random External Interrupts

1.6.7.5.2 Instruction-by-Instruction-Level Checking

1.6.7.5.3 Uniprocessor

1.6.7.5.4 CMP

1.6.7.5.4.1 MP Cache Coherence

Run-time coherency monitor/protocol checker

1.6.7.5.4.2 Memory and Cache Consistency

1.6.7.5.4.3 Synchronization

1.6.7.5.4.4 Reservations

1.6.7.5.4.5 Ordering and Propagation of I/O Traffic at Several Levels of the I/O Hierarchy

1.6.7.6 Behavioral JHDL

The test bench is implemented as behavioral JHDL for speed.

1.6.8 Effectively optimizes DUV for simulation speed.

1.6.8.1 Avoid placing value on wire if wire already carries that value

Before a simulation run, it is possible to determine that certain assignments in behavioral models would be redundant. Such assignments should be ignored during a simulation run. Effect during simulation run should be as if the redundant assignment had been removed from the behavioral model source code before compilation.

1.6.8.2 Comment (not a requirement)

If there is a necessary selection statement in a behavioral model, and a particular action is associated with

every guard condition of that selection statement, then it may be desirable to perform that action before the selection statement. That is, it may not be necessary to wait for evaluation of the condition before performing an action that does not depend on the result of that evaluation. Since the selection statement is necessary, there is not necessarily a simulation speedup possible here. The same amount of work must be done regardless of the order of execution. However, there is a possible synthesis optimization (i.e. one that could result in a higher-performance SPARC-V9 implementation). Synthesis is beyond the scope of this SRS.

1.7 Model

1.7.1 User-supplied full-chip behavioral/RTL DUV

1.7.1.1 SPARC-V9 Compliant Design Under Verification

Design Under Verification is required to comply with the SPARC-V9 Architecture as defined in *The SPARC Architecture Manual Version 9*. Design Under Verification could be certified by SPARC International at Level 2. Simulator is not required to simulate any other type of logic design.

SPARC-V9 is an instruction set architecture (ISA) with 32- and 64-bit integer and 32-, 64- and 128-bit floating-point as its principal data types.

SPARC-V9 defines general-purpose integer, floating-point, and special state/status register instructions, all encoded in 32-bit-wide instruction formats. The load/store instructions address a linear, 2^{64}-byte address space.

1.7.1.1.1 Floating-Point

1.7.1.1.1.1 IEEE Standard for Binary Floating-Point Arithmetic (754-1985)

The 32- and 64- bit floating point types conform to IEEE Std 754-1985.

1.7.1.1.1.2 IEEE Standard for Shared-Data Formats Optimized for Scalable Coherent Interface (SCI) Processors (1596.5-1993)

The 128-bit floating-point type conforms to IEEE Std 1596.5-1993.

1.7.1.2 Behavioral model

1.7.1.2.1 Design Under Verification

1.7.1.2.1.1 Comment (not a requirement)

A behavioral model captures a designer's real intent. It captures the design at the microarchitectural level. Formal verification is improved because of the smaller model size and because of easy separation of control logic. Coverage analysis is improved because there are obvious structures to instrument for coverage models. Test program generation is improved because a behavioral model can be used as an implementation-specific reference model for focused test generation. Enables linking detailed performance modeling with high-level behavioral specification

1.7.1.2.1.2 Simulation speed is improved at the behavioral level, as compared with the RTL

1.7.1.2.1.3 machine-readable, executable specification

1.7.1.2.1.4 Runs benchmarks for performance prediction

1.7.1.3 RTL model

1.7.1.3.1 Complies with IEEE Std 1364.1-2002.

1.7.1.3.1.1 JHDL

Complies with IEEE Std 1364.1-2002 as much as is possible for JHDL. JHDL RTL is not required to be synthesizable.

register transfer level of abstraction

pre-synthesis simulation

1.7.1.4 JHDL

It is possible to write a JHDL design that can be simulated on this CBS.

JHDL describes itself as a "set of FPGA CAD tools". However, for pre-synthesis cycle-based simulation, the choice of digital circuit implementation approach (e.g. custom circuit design, cell-based design methodology, or array-based implementation approaches) need not be considered.

The abstract public TechMapper class is a package of commonly used components. All concrete classes that inherit from TechMapper implement the same set of circuits: gates, muxes, flip flops, adders, and wires. To abstract the details of TechMappers away for the user, an API of function calls was created (the Logic class API) to provide a uniform way to access TechMappers. Thus, a designer would call the and() function in the Logic API and get an AND gate. The net result is that the Logic API provides a technology-independent way for a user to create simple logic function.

VirtexTechMapper is the default TechMapper. All library primitive elements are behaviorally modelled, and the choice of TechMapper determines which behavioral model will be used. JHDL allows users to override the default to get a different TechMapper. This simulator will hide that capability.

byucc.jhdl.Logic.TechMapper is public in byucc.jhdl.Logic; can be accessed from outside package. TechMapper could be extended with a new concrete class that creates technology-specific implementations of logic functions created by the user, using a digital circuit implementation approach more typical of high-performance microprocessors. That is not necessary for pre-synthesis cycle-based simulation. However, simulation speedup may be found by extending TechMapper with a new concrete class.

Extend TechMapper with a new concrete class for simulation speedup, choose a nondefault concrete class that comes with JHDL for simulation speedup, or let VirtexTechMapper be the TechMapper. In any case, the user will think in terms of the Logic API.

1.7.1.5 Chip-Level Simulation

1.7.1.5.1 Uniprocessor

1.7.1.5.2 Chip Multiprocessor (CMP)

1.7.1.5.2.1 Multiple instantiations of base design

For example, simulate something like an 8 processor Symmetric MultiProcessor (SMP) design.

An SMP platform offers the opportunity to take a single design and break it up so segments of it are simulating on different processors. The result is that this segmented, single-program simulation will run much faster across the multiple processors than it would on a single processor. Inter-processor traffic and cache invalidation rates are minimized because results are only examined at the end of clock cycles. Cleanly partition a design to get optimal segmentation.

1.7.1.6 Implementation language is Java

1.7.1.6.1 64-bit integer primitive type

Type long is 64-bit signed two's-complement integer.

1.7.1.6.2 Built-in integer operators do not indicate overflow in any way

1.7.1.6.2.1 Carry-Lookahead

While Java handles 64-bit integer addition nicely, the carry bit is another matter. To assist users who wish to write RTL code, provide a carry-lookahead method. Implementation makes method easy to parallelize.

1.7.1.6.3 64-bit floating-point primitive type

1.7.1.6.4 WORA

Write Once, Run Anywhere

1.7.1.6.5 JHDL

1.7.2 SPARC-V9 Standard Reference Model

Also known as the Instruction Set Processor, the SPARC-V9 Standard Reference Model can be assumed by the user to comply with the SPARC-V9 Architecture at Level 2.

1.7.2.1 Non-implementation-specific

1.7.2.2 Set of correctness properties

1.7.2.3 Comment (not a requirement)

The reference model re-implements the function of the DUV, usually in a high level programming language or an HVL.

1.7.2.4 Emphasis on software speed

1.8 View

This is the display of the simulator on screen (so that the user may view it graphically).

1.8.1 Runs in separate thread from simulator

1.8.2 Waveform

1.8.2.1 External signal activity

1.8.2.2 Internal signal activity

1.8.3 Registers

1.8.4 Address Trace

View address trace of the instruction and data references.

1.8.4.1 Locality

Graphically view temporal and spatial locality.

1.8.5 Cache

1.8.5.1 Contents

1.8.5.2 Cache Hits and Misses

1.8.5.2.1 View which instruction and data references hit and which miss

1.8.5.2.2 Report miss rate

1.8.6 Memory

1.8.7 Instruction Trace

View actual instructions executed.

1.8.8 Device-vs.-Cycle-Time Plot of all System Activity Captured by the Test Bench

1.8.9 Pipeline

1.8.10 MP Utilization

Graphically view a quantitative comparison of each processor's utilization.

1.8.11 Instructions Per Clock (IPC)

For a given simulation run of a program, what was the average number of instructions that were completed per clock cycle?

1.8.11.1 Integer

1.8.11.2 Floating-Point

1.8.11.3 Overall

1.8.12 Instructions Per Program

For a given simulation run of a program, how many instructions were executed?

1.8.12.1 Integer

1.8.12.2 Floating-Point

1.8.12.3 Overall

1.8.13 Clock Cycles Per Program

For a given simulation run of a program, how many clock cycles were required to execute the program?

1.8.14 Class Browser

Gives a hierarchical view of the DUV.

1.8.15 Schematic

1.8.16 JHDL's HWSystem class methods

This class defines a the top-level node for a circuit. All circuit-level issues, such as stepping and cycling, are taken care of in this class.

1.8.16.1 Return the status of system-wide uniquification

1.8.16.2 Get the list of ExternallyUpdateableCells

1.8.16.3 Get the list of all LargeExternallyUpdateableCells

1.8.16.4 Get the list of all CheckpointableCells

1.8.16.5 Access TestBench child of this system

1.8.16.6 Get the total number of clock cycles executed in the simulator up to this point

1.8.16.7 Get the current clock value

In a design with multiple clocks, the user specifies a particular clock to look at (other than the CPU clock). Return the clockdriver's value at the end of the current CPU clock cycle.

1.8.16.8 Check that the circuit is valid for netlisting

1.8.16.9 Return the unscheduled cells

1.8.16.10 Search the circuit graph for a node or wire by name

1.8.16.11 Return the number of floating or multiple driver errors on the last simulation run

1.8.16.12 Cause the ExternalUpdate Manager to dump the current state of the system to a StateObject

1.8.16.13 Get the usingImplicitClock flag

1.8.16.14 See what the current schedule is for the default clock

1.8.16.15 Return whether the circuit is in multi-clock mode

1.8.16.16 Return whether transition counts have been enabled

1.8.16.17 Print out all of the transition counts for every wire in the current testbench and it's corresponding children

1.8.16.18 Print out all of the transition counts for every wire in the target Cell and it's children

1.8.16.19 Print the status of the simulator

1.8.16.20 Find the propagated wire in the global schedule

1.8.16.21 Return whether the simulator is initialized

1.9 Controller

This is the graphical user interface (which allows the user to control the simulation).

1.9.1 JHDL's HWSystem class methods

This class defines a the top-level node for a circuit. All circuit-level issues, such as stepping and cycling, are taken care of in this class.

1.9.1.1 Force an initialize of all of the internal data structures of the HWSystem at the appropriate time (it may not happen immediately)

1.9.1.2 Set the status of system-wide uniquification

1.9.1.3 Access TestBench child of this system

1.9.1.4 Add an Observable to the circuit

1.9.1.5 Add a SimulatorCallback to the circuit

1.9.1.6 Remove a SimulatorCallback from the circuit

1.9.1.7 Optimize the circuit for simulation

1.9.1.8 Advance the simulator by the specified number of CPU clock cycles

1.9.1.9 Force a call to the sim_update of all SimulatorCallbacks and Observables

1.9.1.10 Force a simulator (re)initialization

1.9.1.11 Reset the simulation

1.9.1.12 Cause the ExternalUpdate Manager to load the state of system from the State object corresponding to the given cycle count

1.9.1.13 Declare a wire to be an external clock

1.9.1.14 Set the usingImplicitClock flag

1.9.1.15 Modify the current default clock schedule

1.9.1.16 Enable or disable the transition counts in the current HWSystem

1.9.1.17 Set a flag that causes the simulator to break out of it's run() loop

1.9.2 Effective pseudo-random testing of a particular section of a chip

1.9.3 Interactive Debugging

If the designer finds the output patterns/waveforms during simulation do not match what he expects, the design needs to be debugged.

1.9.3.1 Step through SPARC-V9 assembly language

1.9.3.2 Set breakpoint on any SPARC-V9 assembly language instruction

Execution will stop at that breakpoint.

1.9.3.3 Start execution from any point at which it is stopped

1.9.4 Register Contents

1.9.4.1 Change contents of any register prior to simulation run

1.9.4.2 Change contents of any register while execution is stopped at a breakpoint

1.9.5 Cache Contents

1.9.5.1 Change contents of any block prior to simulation run

1.9.5.2 Change contents of any block while execution is stopped at a breakpoint

1.9.6 Memory Contents

1.9.6.1 Change contents of any memory address prior to simulation run

1.9.6.2 Change contents of any memory address while execution is stopped at a breakpoint

1.9.7 Signal interface

Makes it possible to dump binary traces of internal signal activity to a file for later processing.

1.9.8 Call subroutines internal to the model

1.9.9 Loader

Until the DUV can boot an operating system, the controller must act as the loader.

Reads an absolute program (called a load module in UNIX) from a file on the host and places it into simulated primary memory, where the DUV can decode and execute each instruction.

Before a program can be executed, simulated primary memory must be allocated to the process. Once the system (who?) knows which simulated primary memory locations are going to be used to execute the program, it can then map the address space into the allocated simulated primary memory addresses. The executable program is then translated into its final executable form—the form expected by the DUV—and loaded into the simulated primary memory at the proper location. When the PC (program counter) is set to the simulated primary memory address of the first executable instruction—the main entry point—for the program, the DUV begins to execute the program. The link editor (a tool beyond the scope of this SRS) will have identified the entry point when it created the absolute program.

The link editor will have operated in static mode. Without this restriction, operating system support would be required (e.g. a runtime linker for dynamic objects).

The absolute program is an ELF executable object file.

SPARC-V9 is designed to be binary upward-compatible from SPARC-V8 for applications running in nonprivileged mode that conform to the SPARC-V8 ABI.

1.9.9.1 Programs

Programs provide broad coverage of both the architectural and the implementation-specific features of the DUV. The controller need not create these programs, but it must be able to handle the various types.

1.9.9.1.1 Uniprocessor

1.9.9.1.2 MP

1.9.9.1.3 Absolute program is a 32-bit object

1.9.9.1.4 Absolute program is a 64-bit object

1.9.10 Control whether or not the DUV will have its state compared with that of the SPARC-V9 Standard Reference Model

1.9.11 Control which automated verification is performed at simulation time and which will be perfomed after simulation has completed

Automated verification that is performed after simulation is performed by other tools outside the scope of this SRS.

1.9.12 Specify events to be monitored during simulation run time

1.9.13 Boot operating system

Execute Solaris on the model.

1.9.13.1 Comment (not a requirement)

Booting a version of the operating system, which has routines like memory array tests removed, is feasible with cycle-simulation.

1.9.14 Feed stimulus with a data file

1.9.15 Start execution of new tests

1.9.16 Regression Testing

1.9.16.1 Subset would run overnight

for example, 150,000 tests

1.9.16.2 Complete set would run against major DUV deliveries

for example, 500,000 tests

1.9.16.3 Load and start execution of architectural verification programs

1.9.16.4 Load and start execution of implementation verification programs

for example, specific sequences that Solaris developers identify as interesting, and sequences that are identified by lead architects and designers

1.9.16.5 Load and start execution of up to 2,147,483,647 cycles of test patterns on a

new DUV

1.9.16.6 Load and start execution of all previously completed sections of the regression tests

These tests need to be run over and over again.

1.9.16.7 Load and start execution of tests that found previous bugs

1.9.17 Direct simulation to run benchmarks for performance prediction

1.9.18 Runs in separate thread from simulator

2. Glossary

ABI	Application Binary Interface
API	Application Programming Interface
Absolute program	A data structure that includes the translated program, the addresses in which each instruction will be stored, and a description of how variables should be set up when the program runs. In UNIX systems, the absolute program is called the load module (and may also be referred to by its default name of `a.out`).
CBS	Cycle-Based Simulator
CMP	Chip multiprocessor
CMT processor	Chip multithreading processor; a single processor capable of executing multiple threads simultaneously. While one thread waits for memory, the processor remains busy executing other threads.
CPU clock cycle	That which is counted by the counter field of the TICK register.
DUV	Design Under Verification
Exception	A condition that makes it impossible for the processor to continue executing the current instruction stream without software intervention.
FPU	Floating-Point Unit; a processing unit that contains the floating-point registers and performs floating-point operations, as defined by *The SPARC Architecture Manual Version 9*
GUI	Graphical User Interface

ILP	Instruction-Level Parallelism; potential overlap among instructions
IPC	Instructions Per Clock
ISA	Instruction Set Architecture
IU	Integer Unit; a processing unit that performs integer and control-flow operations and contains general-purpose integer registers and processor state registers, as defined by *The SPARC Architecture Manual Version 9*
Interrupt request	A request for service presented to the processor by an external device.
JVM	Java Virtual Machine (also called the Java interpreter); loads Java applications into memory and executes.
Link editor	A translation tool that combines relocatable object modules with library modules to produce an absolute program that is suitable for loading.
Loader	Also called the absolute loader. A tool to retrieve an absolute program module from secondary memory, translate it into a format suitable for execution, and place the resulting executable image into primary memory.
Logic Synthesis	The process of conversion from HDL to logic gates
MP	Multiprocessor; parallel processors
Memory stall cycles	The number of cycles during which the processor is stalled waiting for a memory access.
Miss rate	Cache misses per memory access.
Privileged mode	The processor mode when PSTATE.PRIV = 1.
Processor	The combination of integer unit and floating-point unit.
RTL	Register Transfer Level
SCI	Scalable Coherent Interface
SMP	Symmetric (shared-memory) Multiprocessor; multiprocessor with single main memory that has a symmetric relationship to all processors and a uniform access time from any processor
SRS	Software Requirements Specification
Simulator kernel	The part of the simulator that is a subclass of JHDL's abstract Simulator class. byucc.jhdl.base.Simulator is not public in byucc.jhdl.base; cannot be accessed from outside package

Supervisor software	Software that executes when the processor is in privileged mode.
TLP	Thread-Level Parallelism; a level of parallelism higher than ILP, logically structured as separate threads of execution
Test bench	A wraparound on a design that injects stimulus into the "design under verification" to verify the functionality of the design.
Thread	**1 :** A set of software instructions that can execute independently. **2 :** A unit of computation that contains minimum internal state and resources. It is associated with a normal, heavyweight operating system process.
Throughput	The aggregate amount of work done in a given amount of time.
Trap	The action taken by the processor when it changes the instruction flow in response to the presence of an exception, a Tcc instruction, or an interrupt. The action is a vectored transfer of control to supervisor software through a table, the address of which is specified by the privileged Trap Base Address (TBA) register.
Uniprocessor	One processor

3. References

Brigham Young University Configurable Computing Laboratory [2003]. *JHDL API Documentation, www.jhdl.org/documentation/api/index.html*.

Burger, D., and J. Goodman [2004]. "Billion-Transistor Architectures: There and Back Again," *IEEE Computer* 37:3 (March).

Darringer, J., E. Davidson, D. Hathaway, B. Koenemann, M. Lavin, B. Lee, J. Morrell, S. Ponnapalli, K. Rahmat, W. Roesner, E. Schanzenbach, and L. Trevillyan [2000]. "EDA in IBM: Past, Present and Future," *IEEE Trans. Computer-Aided Design, Integrated Circuits & Systems* 19:12 (December), 1476-1497.

Deitel, H., and P. Deitel [2003]. *Java™: How to Program*, 5th ed., Prentice Hall, Upper Saddle River, N.J.

Hennessy, J., and D. Patterson [2003]. *Computer Architecture: A Quantitative Approach*, 3rd ed., Morgan Kaufmann, San Francisco.

IEEE [1985]. *IEEE Std 754-1985: IEEE Standard for Binary Floating-Point Arithmetic*.

IEEE [1993]. *IEEE Std 1596.5-1993: IEEE Standard for Shared-Data Formats Optimized for Scalable Coherent Interface (SCI) Processors*.

IEEE [2002]. *IEEE Std 1364.1-2002: IEEE Standard for Verilog Register Transfer Level Synthesis*.

Lee, W. [2003]. *Verilog Coding for Logic Synthesis*, Wiley, Hoboken, N.J.

Lindholm, T., and F. Yellin [1999]. *The Java Virtual Machine Specification*, second edition, Addison-

Wesley, Reading, Mass. Also available online at *java.sun.com/docs/books/vmspec/*.

Ludden, J., W. Roesner, G. Heiling, J. Reysa, J. Jackson, B. Chu, M. Behm, J. Baumgartner, R. Peterson, J. Abdulhafiz, W. Bucy, J. Klaus, D. Klema, T. Le, F. Lewis, P. Milling, L. McConville, B. Nelson, V. Paruthi, T. Pouarz, A. Romonosky, J. Stuecheli, K. Thompson, D. Victor, B. Wile [2002]. "Functional verification of the POWER4 microprocessor and POWER4 multiprocessor systems," *IBM J. Research and Development* 46:1 (January).

Nutt, G. [1997]. *Operating Systems: A Modern Perspective*, Addison-Wesley, Reading, Mass.

Rabaey, J. [1996]. *Digital Integrated Circuits: A Design Perspective*, Prentice Hall, Upper Saddle River, N.J.

SPARC International, Inc. [1994]. *The SPARC Architecture Manual Version 9*, PTR Prentice Hall, Englewood Cliffs, N.J.

Sun Microsystems, Inc. [2003], "Introduction to Throughput Computing", [Online], Available: http://www.sun.com/processors/whitepapers/throughput_whitepaper.pdf (February).

Taylor, S., M. Quinn, D. Brown, N. Dohm, S. Hildebrandt, J. Huggins, and C. Ramey [1998]. "Functional verification of a multiple-issue, out-of-order, superscalar Alpha processor – The DEC Alpha 21264 microprocessor," *Proc. 35th Design Automation Conference* (June), San Francisco, 638-644.

Yen, D. [2004], "An Open Letter from David Yen", [Online], Available: http://www.sun.com/aboutsun/media/features/vos_yen.html (April).