Sun microsystems

picoJava I

December 1997

# picoJava™ I

DATA SHEET

Java Processor Core

## DESCRIPTION

picoJava™ I is a uniquely designed processor core which natively executes Java bytecodes as defined by the Java Virtual Machine (JVM).

Most processors require the JVM to be interpreted by them. This requires that the JVM bytecodes be interpreted or dynamically compiled using a Just-In-Time (JIT) compiler for a specific processor. picoJava I eliminates the need for this. JIT compilers or interpreters and the overhead that accompany them are eliminated by the picoJava I processor core.

picoJava I accelerates the JVM runtime environment. The picoJava I processor core features thread synchronization and a variety of garbage collection methods. It also supports method invocations and the hiding of loads from local variables, thus streamlining object oriented programming. The picoJava I processor core can be optimized for power, die size, or speed. This flexibility gives the price/performance necessary for a range of target applications. picoJava I's architecture is composed of the following basic units: integer execution unit, floating point unit, instruction cache, data cache, stack manager, and bus interface unit.

The picoJava I processor core executes the most commonly used instructions in hardware. Complex instructions are microcoded and the most complex instructions are trapped and emulated in software. Based on a four stage pipeline with a 64 entry stack cache, top-of-stack operations are accelerated by instruction folding, a process of loading and executing an instruction in a single cycle. In addition to the standard JVM bytecodes, picoJava I implements a set of extended bytecodes. These bytecodes can only be executed by the operating system and are not available to the user. Extended bytecodes support arbitrary load/store, cache management, and internal register access. Both data and instruction caches are configurable from 0-16 kilobytes (two-way, set associative). The interface to I/O and memory is managed by the bus interface unit. Single and double precision floating point data are enabled by a floating point unit if the target application design requires it.

### *Features*

- Customer configurable core
- Executes Java bytecodes natively
- Hardware stack architecture
- Instruction folding
- 4 stage RISC pipeline
- Extended instruction set
- Hardware support of runtime environment
- Complete static core

### *Benefits*

- Market differentiation
- Most efficient execution of Java applications
- Supports JVM execution stack efficiently
- Better cycles per instruction (CPI)
- Simplicity
- Support for system level functions
- Efficient JVM
- Low power capability

## BLOCK DIAGRAM AND TYPICAL APPLICATION

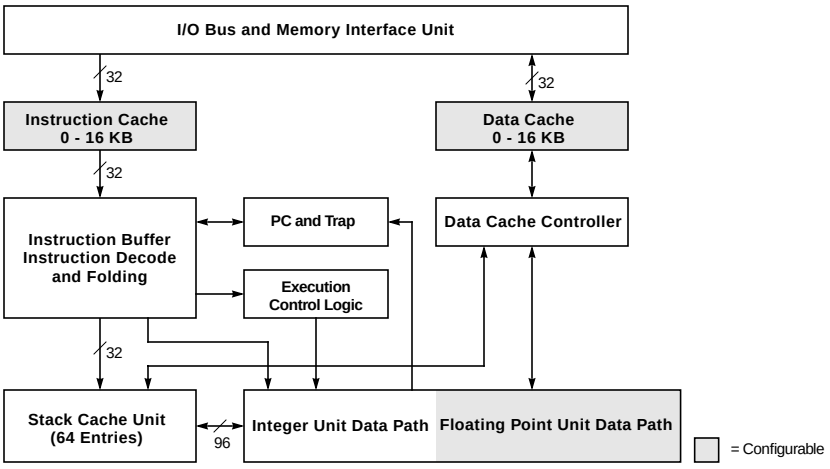


**Figure 1. picoJava I Block Diagram**

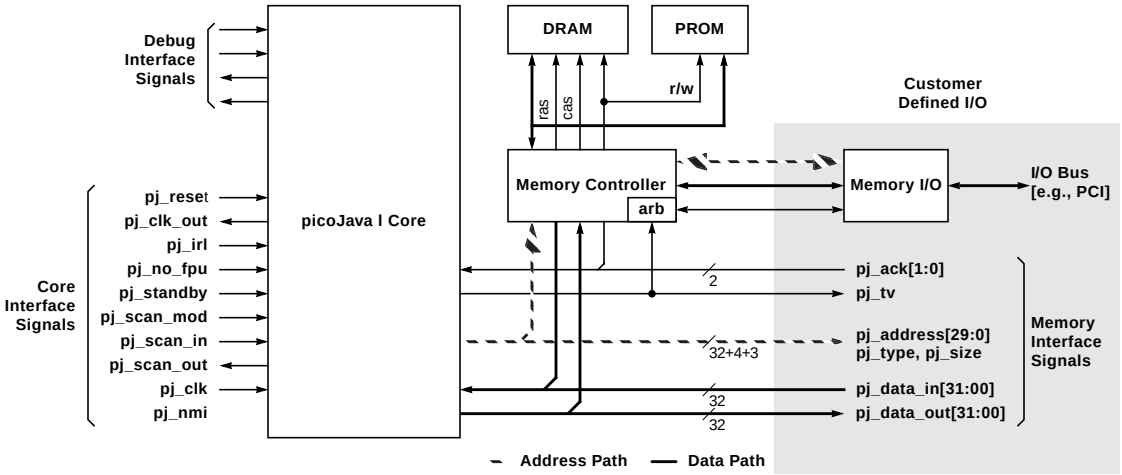


**Figure 2. Example of picoJava I Chip-Based Design**

## TECHNICAL OVERVIEW

### *Instruction Fetch/Cache Unit (ICU)*

The Instruction Cache Unit (ICU) fetches instructions from the Instruction cache (ICache) and provides them to the decode block located in the integer unit. In order to separate the rest of the pipeline from the fetch stage, a twelve byte Instruction buffer (IBuffer) is used to hold any instructions fetched from memory until they are consumed by the integer unit.

The Instruction cache is a direct-mapped, eight byte line size cache with single cycle latency. The cache size is configurable to 0KByte, 1KByte, 2KByte, 4KByte (default), 8KByte and 16KByte sizes.

### *Integer Execution Unit (IEU)*

The Integer Execution Unit (IEU) executes all the non floating point instructions defined in JavaSoft's *The Java Virtual Machine Specification*. In addition to the JVM defined instructions, the IEU implements a set of instructions called EXTENDED BYTE CODES. These extra opcodes are necessary to support system level functions such as arbitrary load / store, cache management, and internal register access. These instructions can only be executed by the operating system and are not available to the user.

The IEU is the gate-keeper for the execution of all instructions. It fetches instructions from the instruction cache unit (ICU), and if floating point, they are directed to the Floating Point unit (FPU) for execution. The IEU also fetches data from and stores it back to the data cache unit (DCU).

Since picoJava I is a stack based architecture, the IEU does not contain any general purpose registers visible to the programmer. Rather it contains a cache of the operating stack area, known as the Stack Cache (SC). A microcode ROM is also contained in the IEU which executes certain JVM instructions.

### *Floating Point Unit (FPU)*

The Floating Point Unit (FPU) executes all mathematical instructions. Optimized for single precision performance and smaller core area the FPU is a microcoded engine with a 32-bit data path. Double precision (DP) operations can be implemented but require approximately two to four times the number of cycles as single precision (SP) operations. During typical operation, the microcode sequencer will present a new microword to the data path unit and will monitor the returning branch condition to determine the next microword.

### *Data Cache Unit (DCU)*

The Data Cache Unit (DCU) arbitrates requests coming from the dribble manager or the pipeline. Request priority is given to the pipeline.

### *Stack Manager Unit (SMU))*

The Stack Manager Unit (SMU) consists of a three-read, two-write port stack cache, Stack Control Unit, and the Dribble Manager.

The Stack Control Unit provides the necessary control signals for the IEU (two read, one write port to stack cache) to retrieve operands. It also controls the reading of data from the data cache into the stack Cache and writing back from the stack cache to data cache.

The Dribble Manager dribbles data between the stack and memory whenever there is an overflow or underflow in the stack cache. The Dribble Manager provides the necessary control signals for a single read-write port of the stack cache used exclusively for dribbling purposes.

### *Bus Interface Unit (BIU)*

The Bus Interface Unit (BIU) is the interface between the picoJava core, external memory, and other I/O devices. Interfacing directly to the instruction cache and the Data Cache Units, the BIU is a simple single master memory I/O controller. If multimaster operation is required, an external bus controller/arbitrator is used.

### *Powerdown, Clock, Scan Unit (PCSU)*

The Powerdown-Clock-Scan Unit (PCSU) integrates power management, internal clock generation, system reset, scan, and test. The picoJava processor core supports two levels of low-power operation and provides the basic MUX scan facility that can be connected to a JTAG controller.

## INTERNAL SIGNAL DESCRIPTION

### TABLE 1: Internal Signal Definition

| Signal | Delay[1] | Type | R# | A/S | Definition |
|---|---|---|---|---|---|
| pj_reset | 5 | setup | R1 | S | Reset and start the processor at address 0x00000000 |
| pj_reset_out | 2 | valid | R2 | | Indicates the reset Extended Byte code was executed |
| pj_clk | - | In | R0 | CLK | picoJava cores clock |
| pj_clk_out | * | valid | R1 | | picoJava's clock to external interfaces |
| pj_irl [3:0] | - | In | | A | Interrupt Exception Signals |
| pj_nmi | - | In | | A | Non-maskable interrupt input to picoJava |
| pj_boot8 | - | static | R0 | S | Controls size of Instruction Cache fetches |
| pj_standby | - | static | R0 | S | Standby pin to control power consumption 90% |
| pj_standby_out | 4 | valid | R1 | | Notify system that processor is in Standby Mode |
| pj_no_fpu | - | static | R0 | S | Disable internal Floating Point Unit |
| pj_scan_out | 2 | valid | R0 | | Provide basic scan facility |
| pj_scan_mode | 5 | setup | R1 | S | Switch flip-flops in core to serial shift |
| pj_scan_in | 3 | setup | R1 | S | Input to the processor core's scan chain |
| pj_data_in[31:0] | 5 | setup | R1 | S | Used during reads |
| pj_data_out[31:0] | 4 | valid | R1 | S | Used during writes |
| pj_address[31:0] | 10 | valid | R1 | | Used to interface with non-multiplexed 32-bit address buss |
| pj_size[1:0] | 6 | valid | R1 | | Indicates the size of requested data |
| pj_type[3:0] | 6 | valid | R1 | | Indicates the type of transaction requested by Integer Unit |
| pj_tv | 5 | valid | R1 | | Asserted to start new transaction to the Memory Controller |
| pj_ack[1:0] | 12 | setup | R0 | | Indicates that data will be driving in same cycle on pj_data_in |
| pj_ale[1:0] | 6 | valid | R0 | | Enables address latching |
| pj_halt | 2 | setup | R0 | S | Halt instruction fetching |
| pj_resume | 2 | setup | R0 | S | Resume instruction fetching |
| pj_brk1_sync | 10 | valid | R2 | S | Breakpoint 1 detected by the Core |
| pj_brk2_sync | 10 | valid | R2 | S | Breakpoint 2 detected by the Core |
| pj_in_halt | 1 | valid | R0 | S | Processor is in halt mode (not fetching instructions) |
| pj_inst_complete | 10 | valid | R2 | S | An instruction was just retrieved (when this signal is high) |

1. Numbers are equivalent to two input NAND gate delays.

## TIMING OF SIGNALS

The picoJava I processor core is used with a variety of memory controllers: DRAM, SDRAM, EDO, SRAM, FLASH. It can also be interfaced to various I/O Controllers, such as PCI, USB, PCMCIA. To achieve this flexibility without loss in memory latency, the picoJava I processor core memory interfaces with a VIRTUAL memory controller.

Transactions fall into two main types: “Read-Type Transactions” and “Write-Type Transactions”. The transaction protocol is one of request and accept. This simplifies the design of the memory controller - which can be designed by a third party. This protocol achieves the minimum possible read latency making it easy to increase the number of memory controller ports and still support high bandwidth memory, such as SDRAM.

The memory bus is 32-bit wide. Sub-32-bit accesses are big-endian ordered. Memory access byte enable is indicated by interface signals. Sub 32-bit devices require swap logic at their interface for reads and writes. Access to devices greater than 32-bit wide will need data buffering and routing the data to the proper memory module.

## BUS TRANSACTION WAVEFORMS

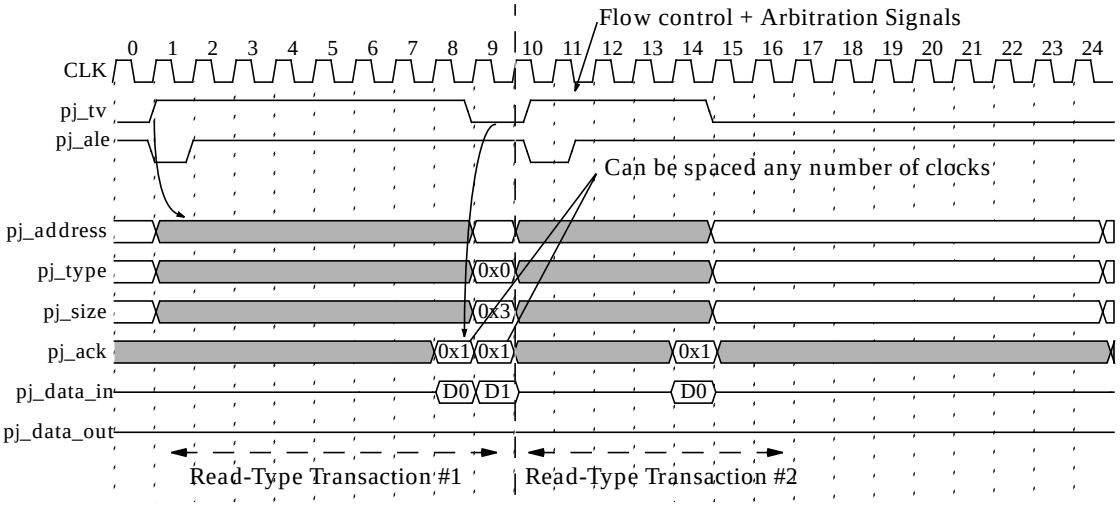


**Figure 3. Cached Read Transaction Followed by a Non-Cached Read Transaction**

***Note:*** *On back-to-back pending transactions from different devices, the pj_tv signal will be continuously asserted. It does not de-assert on these transactions.*

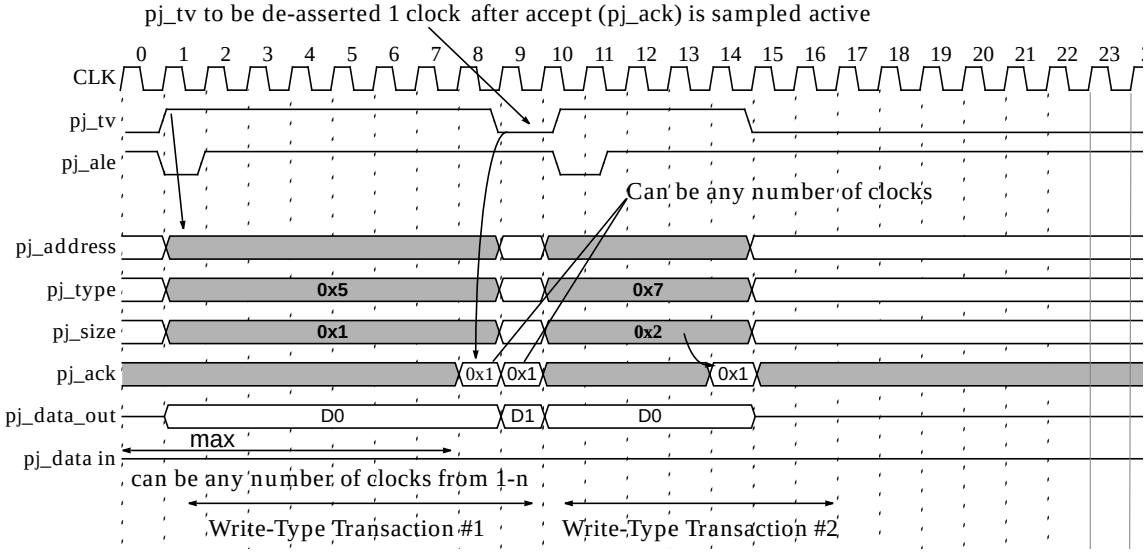


**Figure 4. Cached Write Transaction Followed by a Non-Cached Write Transaction**

***Note:*** *On back-to-back pending transactions from different devices, the pj_tv signal will be continuously asserted. It does not de-assert on these transactions.*

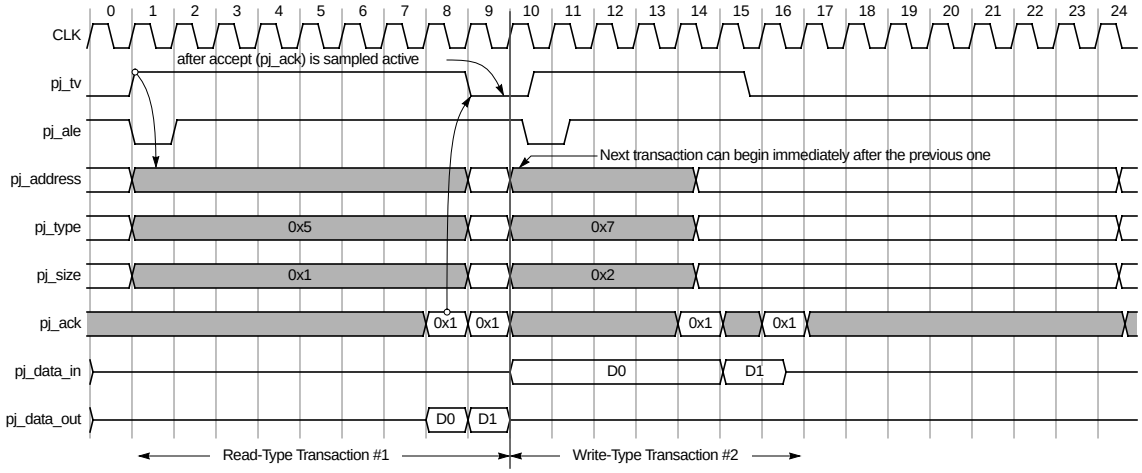


**Figure 5. Cached Read Transaction Followed by a Cached Write Transaction**

***Note:*** *On back-to-back pending transactions from different devices, the pj_tv signal will be continuously asserted. It does not de-assert on these transactions.*

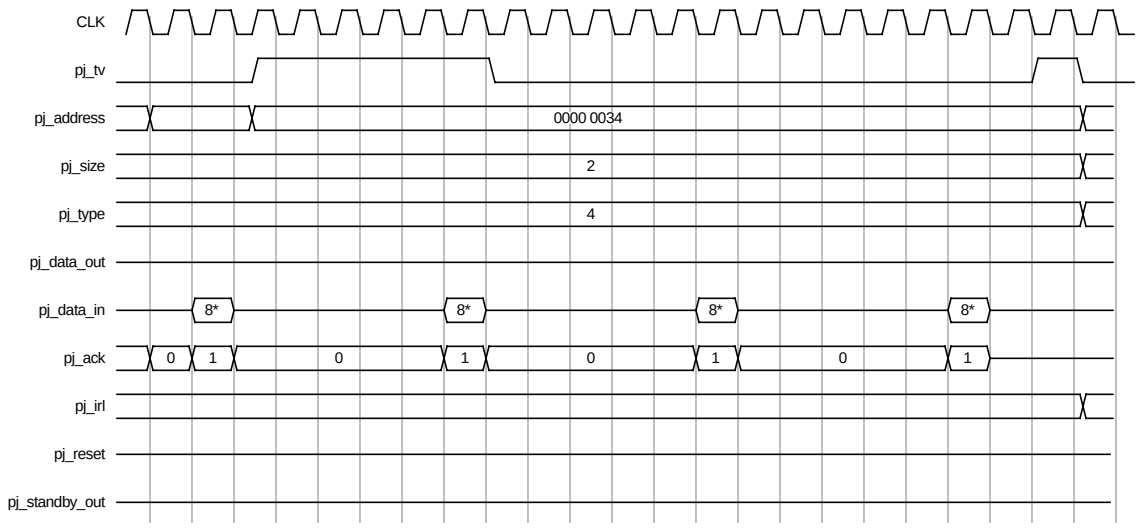


**Figure 6. Cached Load Miss**

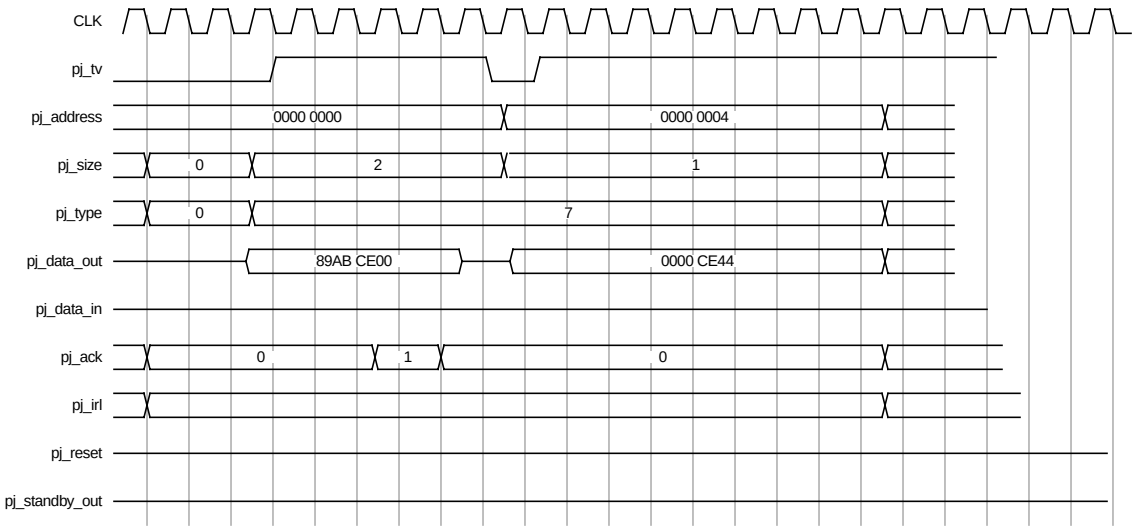


**Figure 7. Non-Cached Write**

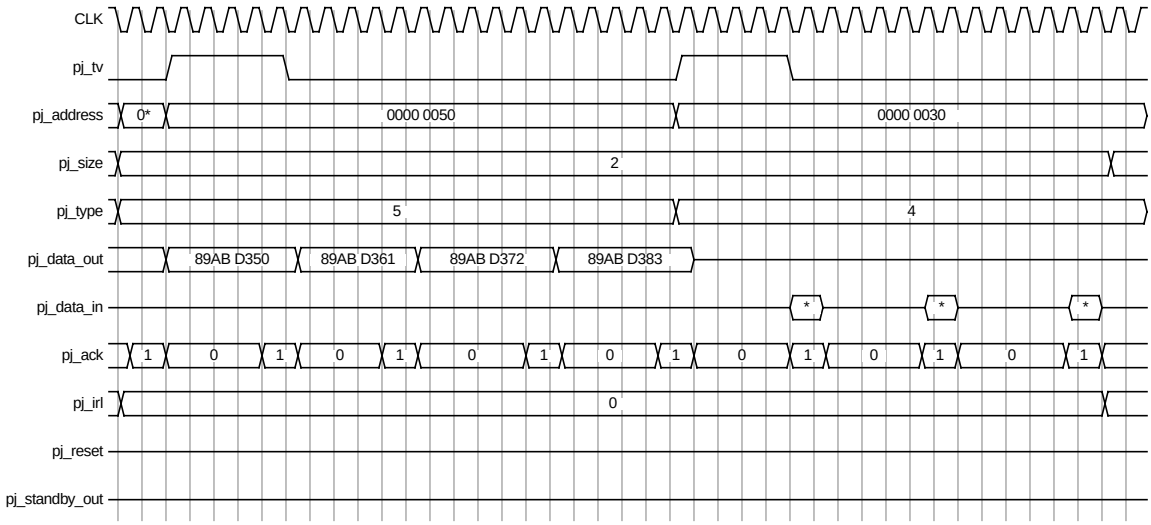


**Figure 8. WriteBack and Cached Read Miss**

picoJava I

THE NETWORK IS THE COMPUTER™

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 USA
Telephone: 800-681-8845
Internet: www.sun.com / microelectronics

©1997 Sun Microsystems, Inc. All Rights reserved.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT ANY EXPRESS REPRESENTATIONS OF WARRANTIES. IN ADDITION, SUN MICROSYSTEMS, INC. DISCLAIMS ALL IMPLIED REPRESENTATIONS AND WARRANTIES, INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTURAL PROPERTY RIGHTS.

This document contains proprietary information of Sun Microsystems, Inc. or under license from third parties. No part of this document may be reproduced in any form or by any means or transferred to any third party without the prior written consent of Sun Microsystems, Inc.

Sun, Sun Microsystems and the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The information contained in this document is not designed or intended for use in on-line control of aircraft, aircraft navigation or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses.

**Part Number: 805-2990-01**