picoJava-II™ Microarchitecture Guide

THE NETWORK IS THE COMPUTER™

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300

Part No.: 960-1160-11
March 1999

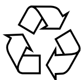
Copyright 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303 U.S.A. All rights reserved.

The contents of this document are subject to the current version of the Sun Community Source License, picoJava Core (“the License”). You may not use this document except in compliance with the License. You may obtain a copy of the License by searching for “Sun Community Source License” on the World Wide Web at http://www.sun.com. See the License for the rights, obligations, and limitations governing use of the contents of this document.

Sun, Sun Microsystems, the Sun logo and all Sun-based trademarks and logos, Java, picoJava, and all Java-based trademarks and logos are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.



Please
Recycle

Contents

Preface xix

1. Overview 1

1.1 Purpose 1

1.2 Components 2

1.2.1 Instruction Cache Unit (ICU) 3

1.2.2 Integer Unit (IU) 3

1.2.3 Floating Point Unit (FPU) 4

1.2.4 Data Cache Unit (DCU) 4

1.2.5 Stack Manager Unit (SMU) 5

1.2.6 Bus Interface Unit (BIU) 5

1.2.7 Powerdown, Clock, and Scan Unit (PCSU) 5

2. Instruction Cache Unit (ICU) 7

2.1 Structure 10

2.2 Functionalities 10

2.2.1 Instruction Cache Control (`ic_cntl`) 10

2.2.2 Instruction Buffer (I-Buffer) 11

2.2.3 I-Buffer Control (`ibuf_cntl`) 12

2.2.4 Instruction Cache Datapath (`icu_dpath`) 12

2.3 Cache Transactions 12
2.3.1 Cache Read Hits 12
2.3.2 Cache Read Misses 13
2.3.3 Noncacheable (NC) Reads 14
2.3.4 Cache Indexed Flushing 14
2.3.5 Error Transactions 14
2.3.6 Diagnostic Accesses 14
2.4 Power Management 15
2.5 Interface Signals 15

3. Integer Unit (IU) 19

3.1 Instruction Pipeline 20
3.2 Instruction Folding Unit (IFU) 21
3.2.1 I-Buffer Interface 22
3.2.2 Folding Logic 22
3.2.3 Folding Groups 23
3.3 Register Control Unit (RCU) 25
3.3.1 RCU Datapath 25
3.3.2 Interface Signals 27
3.4 Datapath 29
3.4.1 Functionalities 30
3.4.2 Interface Signals 31
3.5 Pipe Control Unit 32
3.5.1 Stall Types 33
3.5.2 Interface Signals 34
3.6 Microcode 35
3.6.1 Interface Protocol 36
3.6.2 Control Logic 37
3.6.3 Datapath 38

4. Floating Point Unit (FPU) 41

4.1 Structure 41

4.1.1 ALU 43

4.1.2 Multiply/Divide (MDIV) 43

4.1.3 Microcode Sequencer 43

4.2 Pipeline 43

4.3 Add and Subtract 44

4.4 Multiply and Divide 45

4.5 IEEE 754 Compliance 45

4.5.1 Deviation From IEEE 754 Specification 46

4.5.2 NaN Operations 46

4.6 Opcodes 47

4.6.1 `FADD` and `FSUB` Operations 47

4.6.2 `FMUL` and `DMUL` Operations 47

4.6.3 `FDIV`, `FREM`, `DDIV`, and `DREM` Operations 47

4.6.4 `FConvert` Operations 47

4.7 Power Management 48

4.8 Interface 48

4.8.1 Input Operations 48

4.8.2 Output Operations 50

4.8.3 Data Forwarding 51

4.9 Critical Paths 51

4.10 Signals 52

5. Data Cache Unit (DCU) 53

5.1 Dependencies 55

5.2 Data Cache 55

5.3 Functionalities 57

5.3.1 Arbiter 57

5.3.2 Address Control 57

5.3.3 Aligner Control 57

5.3.4 Miss Control 58

5.3.5 Writeback Control 58

5.3.6 Data Cache Datapath 58

5.4 Pipeline 58

5.4.1 Cache Reads 59

5.4.2 Cache Writes 59

5.4.3 Cache Fills 60

5.4.4 Writebacks 62

5.4.5 Noncacheable (NC) Loads 63

5.4.6 Noncacheable (NC) Stores 63

5.5 Cache Transactions 64

5.5.1 Arbitration of Requests 64

5.5.2 Replacements 64

5.5.3 Cache Compare Flushing 65

5.5.4 Cache Indexed Flushing 65

5.5.5 Cache Invalidate Flushing 65

5.5.6 Zeroing of Cache Lines 65

5.5.7 Nonallocating Writes 66

5.5.8 Nonfetching Allocates 66

5.5.9 Diagnostic Accesses 66

5.6 Interface Signals 66

5.7 Power Management 68

5.8 Critical Timing Path 68

6. Stack Manager Unit (SMU) 71

6.1 Functionalities 73

6.2 Dribbling Operations 74

6.2.1 Spills and Fills 75

6.2.2 Stack Overflow 75

6.2.3 Stack Underflow 76

6.2.4 Stack Cache Write Misses 76

6.3 Pipeline 77

6.4 Interface Signals 77

7. Bus Interface Unit (BIU) 79

7.1 Functionalities 79

7.2 Arbitration 80

7.3 Interfaces 81

7.4 Power Management 82

7.5 Interface Signals 83

8. Powerdown, Clock, Reset, and Scan Unit (PCSU) 85

8.1 Power Management 85

8.1.1 Normal Mode 85

8.1.2 Standby Mode 86

8.2 Clock Management 88

8.3 Reset Management 88

8.4 Scan and Test Features 89

8.5 Interface Signals 91

8.6 Debug and Trace Features 92

8.7 JTAG Support 93

8.7.1 Full Internal Scan 93

8.7.2 Breakpoints and External Halt Mode 93

8.7.3 Single-Stepping 94

8.7.4 Nonscannable Arrays 95

9. External Interface 97

9.1 Core Interface Signals 97

9.1.1 Processor Interface Signals 100

9.1.2 Memory Interface Signals 101
9.1.3 Trace and Debug Signals 103
9.2 picoJava-II Transactions 105
9.2.1 Boot Mode 105
9.2.2 Read-Type Transactions 106
9.2.3 Write-Type Transactions 108
9.3 Endianness and Cacheability 111
9.4 Customizable Features 111

10. Traps and Interrupts 113
10.1 Definitions 113
10.2 Traps 114
10.3 Trap Control 114
10.4 Interrupts 114
10.4.1 Nonmaskable Interrupts (NMI) 115
10.4.2 Maskable Interrupts 115
10.5 Interrupt Control 116

11. Megacells 117
11.1 Instruction Cache Data RAM 117
11.1.1 I/O Pins 120
11.1.2 I/O Signals 121
11.1.3 Timing 122
11.1.4 RAM Redundancy 124
11.1.5 Testing 124
11.2 Instruction Cache Tag RAM 125
11.2.1 I/O Pins 128
11.2.2 I/O Signals 130
11.2.3 Timing 130
11.2.4 Implementation 133

11.2.5 RAM Redundancy 133
11.2.6 Testing 133
11.3 Data Cache Data RAM 134
11.3.1 I/O Pins 136
11.3.2 I/O Signals 137
11.3.3 Timing 138
11.3.4 Implementation 140
11.3.5 RAM Redundancy 140
11.3.6 Testing 140
11.4 Data Cache Tag RAM 141
11.4.1 I/O Pins 144
11.4.2 I/O Signals 145
11.4.3 Timing 146
11.4.4 Implementation 149
11.4.5 RAM Redundancy 149
11.4.6 Testing 149
11.5 Stack Cache RAM 150
11.5.1 I/O Pins 150
11.5.2 I/O Signals 151
11.5.3 Timing 152
11.5.4 Testing 154
11.6 Floating Point Unit (FPU) ROM 154
11.6.1 I/O Pins 156
11.6.2 I/O Signals 157
11.6.3 Timing 158
11.6.4 Implementation 160
11.6.5 ROM Contents 160
11.6.6 RAM Redundancy 160
11.6.7 Testing 160

11.7 Integer Unit (IU) ROM 161
11.7.1 I/O Signals 162
11.7.2 Timing 162
11.7.3 Implementation 163
11.7.4 ROM Contents 163
11.7.5 RAM Redundancy 163
11.7.6 Testing 163

Index 165

Figures

FIGURE 1-1 picoJava-II Core 2

FIGURE 1-2 Basic Pipeline 3

FIGURE 2-1 Instruction Cache Unit (ICU) Interaction with Other Units 8

FIGURE 2-2 Aligners 9

FIGURE 2-3 Program Counter 9

FIGURE 2-4 Instruction Buffer 11

FIGURE 2-5 A Miss State Transaction 13

FIGURE 3-1 Integer Unit (IU) 20

FIGURE 3-2 Instruction Pipeline in the IU 21

FIGURE 3-3 Instruction Folding Unit (IFU) Datapath 24

FIGURE 3-4 Register Control Unit (RCU) Datapath (Register Access Part for RS1) 26

FIGURE 3-5 Integer Unit (IU) Datapath 30

FIGURE 3-6 `pipe_dpath` (`PC` Pipe and `OPTOP` Pipe) 33

FIGURE 3-7 Microcode Interface with the Main IU Datapath 35

FIGURE 4-1 Floating Point Unit (FPU) 42

FIGURE 4-2 Double-Precision Operation Using Two Cycles 44

FIGURE 4-3 Simplified ALU Section of the Floating Point Unit (FPU) 44

FIGURE 4-4 Multiply and Divide (MDIV) Section of the FPU 45

FIGURE 4-5 Floating Point Interface 48

FIGURE 4-6 FPU Timing 49

FIGURE 4-7 Critical Paths in the FPU 51

FIGURE 5-1 Data Control Unit (DCU) Interaction with Other Units 54

FIGURE 5-2 16-Kbyte Data Cache (512x256) 56

FIGURE 5-3 A Pipeline Read Transaction 59

FIGURE 5-4 A Cache Write Transaction 60

FIGURE 5-5 A Cache Fill Transaction 61

FIGURE 5-6 A Writeback Transaction 62

FIGURE 5-7 A Noncacheable Load 63

FIGURE 5-8 A Noncacheable Store 64

FIGURE 5-9 Critical Timing Path in the DCU 69

FIGURE 6-1 Stack Manager Unit (SMU) Interaction with Other Units 72

FIGURE 6-2 Stack Cache 74

FIGURE 7-1 Bus Interface Unit (BIU) 80

FIGURE 7-2 BIU Arbiter 80

FIGURE 7-3 Bus Interface Unit (BIU) Interaction with Other Units 82

FIGURE 8-1 Protocol for the PCSU Standby Mode 86

FIGURE 8-2 Test-Related Vendor Issue 90

FIGURE 8-3 Tracing the Instruction Flow Using the picoJava-II Core 92

FIGURE 8-4 Single-Step Timing Through Scan Mode 94

FIGURE 8-5 Single-Stepping Using Breakpoints 95

FIGURE 8-6 Single-Stepping Using `pj_halt` 95

FIGURE 9-1 The picoJava-II External Interface 98

FIGURE 9-2 An Example of a System Based on the picoJava-II Core 104

FIGURE 9-3 A Cached Read Transaction Followed by a Noncached Read Transaction 107

FIGURE 9-4 A Cached Write Transaction Followed by a Noncached Write Transaction 108

FIGURE 9-5 A Cached Load Miss 109

FIGURE 9-6 A Noncached Write 110

FIGURE 9-7 A Writeback and Cached Read Miss 110

FIGURE 9-8 An Example of How To Use the picoJava-II Core 111

FIGURE 10-1 Interrupt Control Mechanism 116

FIGURE 11-1 Instruction Cache Block (icram) 119

FIGURE 11-2 Timing Diagram A (Instruction Cache Data RAM) 123

FIGURE 11-3 Timing Diagram B (Instruction Cache Data RAM) 124

FIGURE 11-4 Instruction Cache Tag (itag) 127

FIGURE 11-5 Timing Diagram A (Instruction Cache Tag RAM) 132

FIGURE 11-6 Timing Diagram B (Instruction Cache Tag RAM) 133

FIGURE 11-7 Data Cache Data RAM (dcram) 135

FIGURE 11-8 Timing Diagram A (Data Cache Data RAM) 139

FIGURE 11-9 Timing Diagram B (Data Cache Data RAM) 140

FIGURE 11-10 Data Cache Tag RAM (dtag) 143

FIGURE 11-11 Timing Diagram A (Data Cache Tag RAM) 148

FIGURE 11-12 Timing Diagram B (Data Cache Tag RAM) 149

FIGURE 11-13 Stack Cache RAM Interface 150

FIGURE 11-14 Timing Diagram A (Read Cycle, Stack Cache RAM) 153

FIGURE 11-15 Timing Diagram B (Write Cycle, Stack Cache RAM) 153

FIGURE 11-16 Floating Point Unit (FPU) ROM 155

FIGURE 11-17 FPU ROM Interface 156

FIGURE 11-18 Assertion and Deassertion of rom_en and me 159

FIGURE 11-19 Timing Diagram A (FPU ROM) (Read Cycle, Normal Operation, Enabled, Nontest Mode) 159

FIGURE 11-20 Timing Diagram B (FPU ROM) (Read Cycle, Normal Operation, Disabled, Nontest Mode) 160

FIGURE 11-21 Integer Unit (IU) ROM 161

FIGURE 11-22 Timing Diagram (IU ROM) 162

Tables

TABLE 2-1 ICU Interface with the IU 15
TABLE 2-2 ICU Interface with the BIU 16
TABLE 2-3 ICU Interface with the Instruction Cache Tag 16
TABLE 2-4 ICU Interface with the Instruction Cache 16
TABLE 2-5 ICU Interface with the PCSU 17
TABLE 3-1 Instruction Types in the IFU 23
TABLE 3-2 IFU Groupings 23
TABLE 3-3 Block Interface with the IFU 27
TABLE 3-4 Block Interface with the IU 28
TABLE 3-5 Block Interface with Pipeline Control 28
TABLE 3-6 Block Interface with the SMU 29
TABLE 3-7 Block Interface of the IU Datapath 31
TABLE 3-8 Interface Signals for Pipe Control 34
TABLE 3-9 Block Interface of Microcode Control 37
TABLE 3-10 Block Interface of the Microcode Datapath 38
TABLE 4-1 Example Results of a NaN Two-Input Operation 46
TABLE 4-2 FPU Input Cycle Transactions 49
TABLE 4-3 FPU Signals 52
TABLE 5-1 Data Cache Fill Ordering 61

TABLE 5-2 DCU Interface with the Bus Interface Unit (BIU) 66
TABLE 5-3 DCU Interface with the Stack Manager Unit (SMU) 67
TABLE 5-4 DCU Interface with the Integer Unit (IU) 67
TABLE 5-5 Miscellaneous DCU Interfaces 68
TABLE 6-1 OPTOP Instructions That Trigger Underflows 76
TABLE 6-2 Pipeline Actions for Spills and Fills 77
TABLE 6-3 SMU Interface with the IU 77
TABLE 6-4 SMU Interface with the DCU 78
TABLE 7-1 BIU Interfaces with the Instruction Cache and the Data Cache 83
TABLE 7-2 BIU External Interface Signals 83
TABLE 8-1 PCSU Interface with JTAG 91
TABLE 8-2 PCSU Interface with Function Units 91
TABLE 8-3 PCSU Interface with External Systems 91
TABLE 9-1 Interface Signals 98
TABLE 9-2 Transaction Sizes 102
TABLE 9-3 Transaction Types 102
TABLE 9-4 acks 103
TABLE 11-1 test_mode and Enable Signals (Instruction Cache Data RAM) 118
TABLE 11-2 Input Pins (Instruction Cache Data RAM) 120
TABLE 11-3 I/O Signals (Instruction Cache Data RAM) 121
TABLE 11-4 Timing Specifications (Instruction Cache Data RAM) 122
TABLE 11-5 test_mode and Enable Signals (Instruction Cache Tag RAM) 126
TABLE 11-6 Input Pins (Instruction Cache Tag RAM) 128
TABLE 11-7 Output Pins (Instruction Cache Tag RAM) 129
TABLE 11-8 I/O Signals (Instruction Cache Tag RAM) 130
TABLE 11-9 Timing Specifications (Instruction Cache Tag RAM) 131
TABLE 11-10 test_mode and Enable Signals (Data Cache Data RAM) 135
TABLE 11-11 Input Pins (Data Cache Data RAM) 136

TABLE 11-12 Output Pins (Data Cache Data RAM) 137

TABLE 11-13 I/O Signals (Data Cache Data RAM) 137

TABLE 11-14 Timing Specifications (Data Cache Data RAM) 138

TABLE 11-15 test_mode and Enable Signals (Data Cache Tag RAM) 142

TABLE 11-16 Input Pins (Data Cache Tag RAM) 144

TABLE 11-17 Output Pins (Data Cache Tag RAM) 145

TABLE 11-18 I/O Signals (Data Cache Tag RAM) 145

TABLE 11-19 Timing Specifications (Data Cache Tag RAM) 147

TABLE 11-20 Input Pins (Stack Cache RAM) 150

TABLE 11-21 Output Pins (Stack Cache RAM) 151

TABLE 11-22 I/O Signals (Stack Cache RAM) 151

TABLE 11-23 Timing Specifications (Stack Cache RAM) 152

TABLE 11-24 Input Pins (FPU ROM) 156

TABLE 11-25 Output Pins (FPU ROM) 156

TABLE 11-26 Truth Table for ROM Output 157

TABLE 11-27 I/O Signals (FPU ROM) 157

TABLE 11-28 Timing Specifications (FPU ROM) 158

TABLE 11-29 I/O Signals (IU ROM) 162

TABLE 11-30 Timing Specifications (IU ROM) 162

Preface

This specification is the single depository of information that a logic designer or architect needs to gain a thorough understanding of the picoJava™-II microarchitecture—the second-generation Java™ processor.

For details about programming the picoJava-II architecture—

- Data types and runtime structure
- Instruction set
- Java method invocation and return
- Monitors
- Support of C Language
- Garbage collection
- System Management and Debugging

—see the *picoJava-II Programmer's Reference Manual*.

The Java Language and the Java Virtual Machine

The Java language is an object-oriented programming language developed by Sun Microsystems, Inc., in the early 1990s. Modeled after C and C++, it is designed to be simple and platform-independent at both the source and binary levels. The Java language was initially developed to address the problems of building software for networked consumer devices.

The Java virtual machine is the cornerstone of the Java programming language. It is the component of the Java technology responsible for the Java cross-platform delivery as well as for the small size of its compiled code. The Java virtual machine is an abstract computing machine. Like a real computing machine, it has an

instruction set and uses various memory areas. The Java virtual machine understands only a particular file format—the class file format. A class file contains Java virtual machine instructions, a symbol table, and other information. The Java virtual machine does not understand the Java programming language and does not require a specific underlying implementation.

Organization of This Book

There is no single best order for the chapters in this manual. You should read the sections that you are interested in and follow the references to the other sections if you need additional details.

Chapter 1, *Overview*, introduces the picoJava-II modules.

Chapter 2, *Instruction Cache Unit (ICU)*, describes the picoJava-II ICU, which fetches instructions from the instruction cache and provides them to the Integer Unit (IU).

Chapter 3, *Integer Unit (IU)*, describes the picoJava-II IU, which executes integer instructions, dispatch and tracking of integer and floating-point instructions, exception detection and handling, and other pipeline support functions.

Chapter 4, *Floating Point Unit (FPU)*, describes the picoJava-II FPU.

Chapter 5, *Data Cache Unit (DCU)*, describes the picoJava-II DCU, which manages all requests to the data cache.

Chapter 6, *Stack Manager Unit (SMU)*, describes the picoJava-II SMU, which stores and provides the necessary operands to the IU.

Chapter 7, *Bus Interface Unit (BIU)*, describes the picoJava-II BIU, which is the glue between the picoJava-II core and memory.

Chapter 8, *Powerdown, Clock, Reset, and Scan Unit (PCSU)*, describes the picoJava-II power management environment.

Chapter 9, *External Interface*, describes the picoJava-II core external interfaces.

Chapter 10, *Traps and Interrupts*, describes the picoJava-II trap handling, which is designed to support real-time systems.

Chapter 11, *Megacells*, contains specifications for the picoJava-II megacells.

At the end of this specification is an index.

Related Books and References

Three books form the documentation set for the picoJava-II release:

- *picoJava-II Programmers's Reference Manual*
- *picoJava-II Microarchitecture Guide* (this book)
- *picoJava-II Verification Guide*

The following publications are reference material for the subject matter:

- Lindholm, Tim and Frank Yellin: *The Java™ Virtual Machine Specification*. Addison Wesley, ISBN 0-201-63452-X.
- *IEEE Standard Test Access Port and Boundary-Scan Architecture, ANSI/IEEE Std. 1149.1-1990.*
- *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std. 754-1985.*
- Ungar, David: *ACM SIGPLAN Notices*, 19(5):157-167: *Generation Scavenging: A Non-disruptive High Performance Storage Reclamation Algorithm*, April 1984.
- Wilson P., and T. Moher: *ACM SIGPLAN Notices*, 24(10):23-35: *A Card-marking Scheme For Controlling Intergenerational References In Generation-based Garbage Collection On Stock Hardware*, 1989.
- Steele, Guy L.: *Communications of the ACM*, 18(9): *Multiprocessing Compactifying Garbage Collection*, September 1975.
- Hudson, R., and J. E. B. Moss: *Proceedings of International Workshop on Memory Management*: *Incremental Garbage Collection For Mature Objects*, St. Malo, France, September 16-18, 1992.

Typographic Conventions

TABLE P-1 describes the typographic conventions used in this book.

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, instructions, files, and directories; on-screen computer output; email addresses; URLs	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename*.
AaBbCc123	Book titles, section titles in cross-references, new words or terms, or emphasized words	Read Chapter 6 in *User's Guide.* These are called *class* options. You *must* be root to do this.
<>	A bit number or colon-separated range of bit numbers within a field	`WB_VECTOR`<15:0>

Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals using this program.

For a list of documents and how to order them, see the catalog section of the SunExpress™ Internet site at `http://www.sun.com/sunexpress`.

Sun Documentation Online

The `docs.sun.com` Web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com/`.

Disclaimer

The information in this manual is subject to change and will be revised from time to time. For up-to-date information, contact your Sun representative.

Feedback

Email your comments on this book to: `picojava-doc@sun.com`.

Acknowledgment

Many people—picoJava-II licensees, engineers, programmers, marketers—contributed to this book. We thank them for their input, feedback, and support.

CHAPTER 1

Overview

The picoJava-II core is a hardware implementation of the Java virtual machine instruction set. It supports all Java virtual machine instructions, including the `_quick` codes.

The core requires supporting software (such as a class loader, a bytecode verifier, a thread manager, and a garbage collector, and low-level microkernel) as part of the Java runtime environment, which is beyond the scope of this manual.

This manual focuses on the hardware only.

This chapter describes the picoJava-II core in the following sections:

- *Purpose* on page 1
- *Components* on page 2

1.1 Purpose

The core is targeted for emerging, networked, embedded applications, such as:

- Internet chip for network appliances
- Advanced cellular phone
- Traditional embedded products
- Global positioning systems
- Network computers

1.2 Components

The core includes the following units:

- Instruction Cache Unit (ICU)
- Integer Unit (IU)
- Floating Point Unit (FPU)
- Data Cache Unit (DCU)
- Stack Manager Unit (SMU)
- Bus Interface Unit (BIU)
- Powerdown, Clock, and Scan Unit (PCSU)

FIGURE 1-1 is a core block diagram; FIGURE 1-2 shows the processor's basic pipeline.

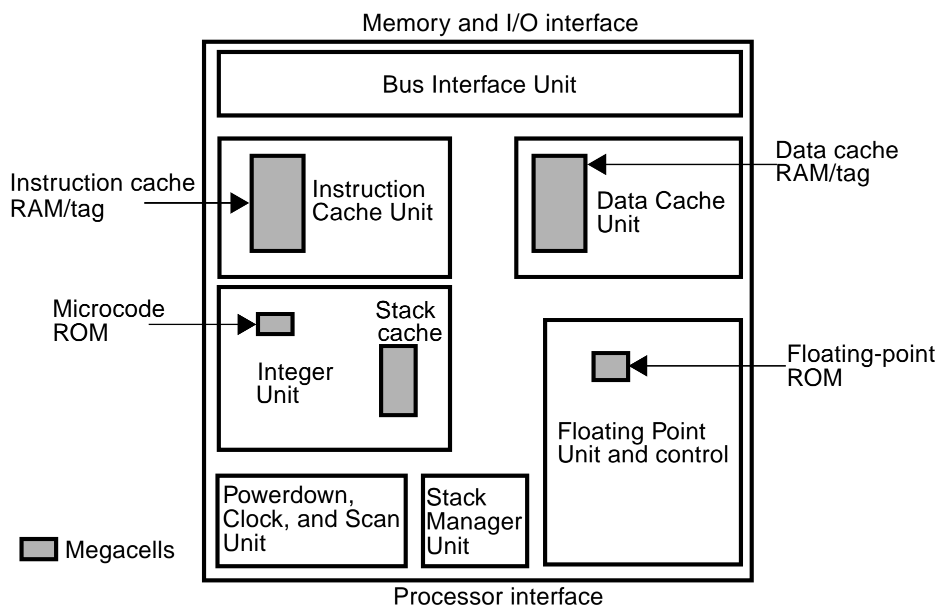


FIGURE 1-1 picoJava-II Core

Fetch	Decode	Register	Execute	Cache	Writeback
Fetch fixed-size cache lines from the instruction cache to the I-Buffer.	Precode and group instructions. Can send a maximum of four instructions.	Access the register file for operands.	Execute for one or more cycles. Micro-active.	Access the data cache.	Write results back into the operand stack. Forward all results before W.

FIGURE 1-2 Basic Pipeline

See *Instruction Pipeline* on page 20 for details.

1.2.1 Instruction Cache Unit (ICU)

The Instruction Cache Unit (ICU) fetches instructions and dispatches them to the decode unit of the Integer Unit (IU). The major blocks are:

- Instruction cache: Direct-mapped, 16-byte line size, configurable between 0-Kbyte, 1-Kbyte, 2-Kbyte, 4-Kbyte, 8-Kbyte and 16-Kbyte sizes
- Instruction buffer: 16 bytes

  The ICU can write 8 bytes and issue up to 7 bytes in one cycle to the IU.
- Instruction cache control logic
- Instruction buffer control logic
- Instruction cache datapath

The ICU supports low-power mode. See Chapter 2 for more information.

1.2.2 Integer Unit (IU)

The Integer Unit (IU) decodes the Java virtual machine instruction, which consists of an opcode that specifies the operation to be performed, followed by zero or more operands. See Chapter 3 for more information.

The instructions belong to the following groups:

Arithmetic/logical/shift	Branch	Load immediate	Method invocation and return
Integer multiply/divide	Object field manipulation	Load/store from local variables	Monitor enter and exit
Stack manipulation	Array management	Conversion	Trapping

The major blocks are:

- 32-bit ALU and shifter
- Microcode ROM
- Programmer-visible registers
- Multiply/divide unit
- Trap generation logic
- Dependency checking and forwarding logic
- Stack cache: 64-entry, 32-bit-wide array with three read and two write ports

Some instructions are executed using the microcode engine in the IU; other instructions cause a trap and must be emulated by trap software.

1.2.3 Floating Point Unit (FPU)

The Floating Point Unit (FPU) executes the floating-point instructions. It is optimized for single-precision performance to save on area. See Chapter 4 for more information.

The major blocks are:

- Microcode sequencer
- Input and output data registers
- Floating-point adder/ALU
- Floating-point multiply/divide unit

The FPU supports low-power mode. No floating-point exceptions are signalled as per *The Java Virtual Machine Specification.*

1.2.4 Data Cache Unit (DCU)

The Data Cache Unit (DCU) handles all the sourcing and sinking of data for load and store instructions. It interacts with the SMU, the IU, the ICU, and the BIU. See Chapter 5 for more information.

The data cache is configurable among 0-Kbyte, 1-Kbyte, 2-Kbyte, 4-Kbyte, 8-Kbyte, and 16-Kbyte sizes.

The major blocks are:

- Two-way, set-associative, write back, write allocate, 16-byte-line size cache.
- Arbiter logic
- Address control
- Aligner control
- Data cache control logic
- Copyback buffer (16 bytes)

The data cache also supports low-power mode.

1.2.5 Stack Manager Unit (SMU)

The Stack Manager Unit (SMU) stores and provides the necessary operands to the IU. It also handles the overflow and underflow conditions of the stack. See Chapter 6 for more information.

The major blocks are:

- Stack control logic
- Dribble manager logic
- Dribble datapath

1.2.6 Bus Interface Unit (BIU)

The Bus Interface Unit (BIU) implements the picoJava-II interface to the external world. It can generate requests to memory and I/O devices. Internal to the core, it interfaces with the ICU and the DCU. See Chapter 7 for more information.

The major blocks are:

- Arbitration logic
- Address and data muxes

1.2.7 Powerdown, Clock, and Scan Unit (PCSU)

The Powerdown, Clock, and Scan Unit (PCSU) integrates the clock generation and low power management. The processor core supports various powerdown modes, as described in Chapter 8.

CHAPTER **2**

Instruction Cache Unit (ICU)

The Instruction Cache Unit (ICU) fetches instructions from the instruction cache (I-Cache) and provides them to the decode block located in the Integer Unit (IU). To separate the rest of the pipeline from the fetch stage, the ICU uses an instruction buffer (I-Buffer) to hold instructions that are fetched from memory until they are consumed by the IU.

This chapter contains the following sections:

- *Structure* on page 10
- *Functionalities* on page 10
- *Cache Transactions* on page 12
- *Power Management* on page 15
- *Interface Signals* on page 15

To support the aggressive folding feature of the IU, the ICU provides the following data bandwidths:

- The I-Buffer gets 8 bytes from the I-Cache or 4 bytes from the Bus Interface Unit (BIU).
- The I-Buffer is 16 bytes deep.
- The IU consumes a maximum of 7 bytes from the I-Buffer in a cycle.
- The instruction cache line size is 16 bytes.
- The data bus to the BIU is 4 bytes.

In a cache miss, the ICU generates a memory request for the missed line. It takes one cycle in a cache hit. If the I-Buffer is empty and an instruction cache miss occurs, the decode unit is stalled. The line in the instruction cache can be invalidated by self-modifying code, however.

The ICU interacts with the IU and the BIU, as shown in FIGURE 2-1.

See FIGURE 2-2 and FIGURE 2-3 for an expansion of the aligners and program-counter-related datapath that are shown in FIGURE 2-1.

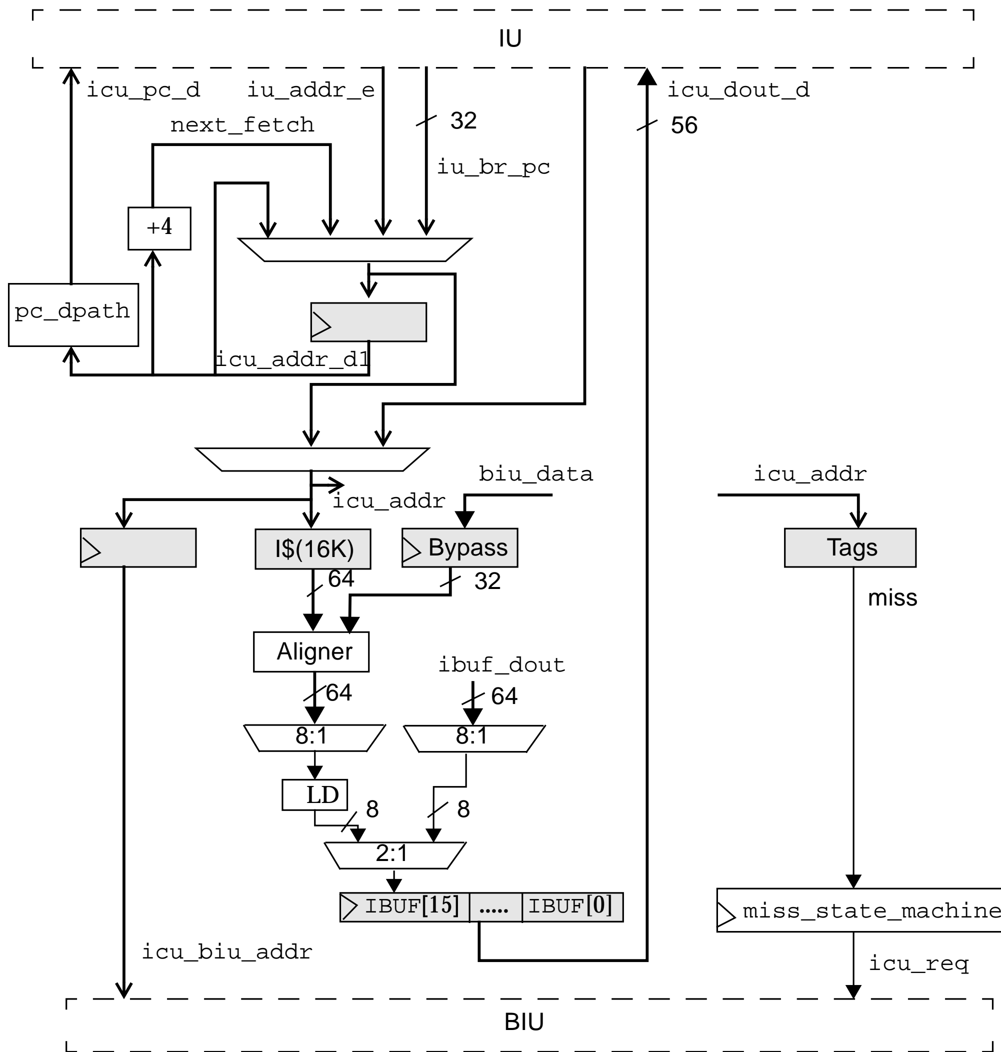


FIGURE 2-1 Instruction Cache Unit (ICU) Interaction with Other Units

Big endian ordering assigns the lowest address to the highest data byte:

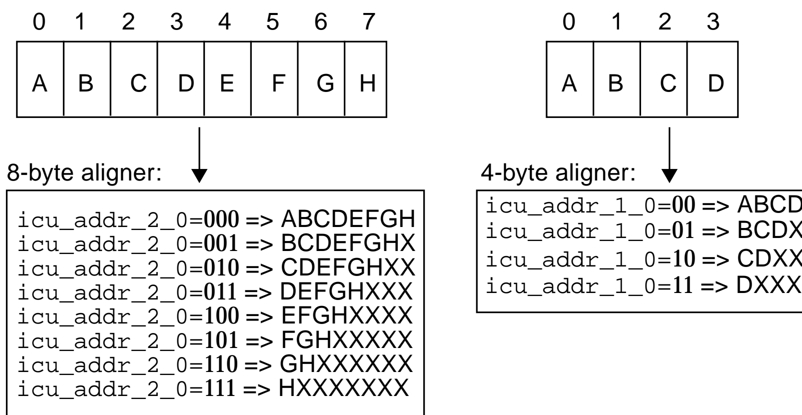


FIGURE 2-2 Aligners

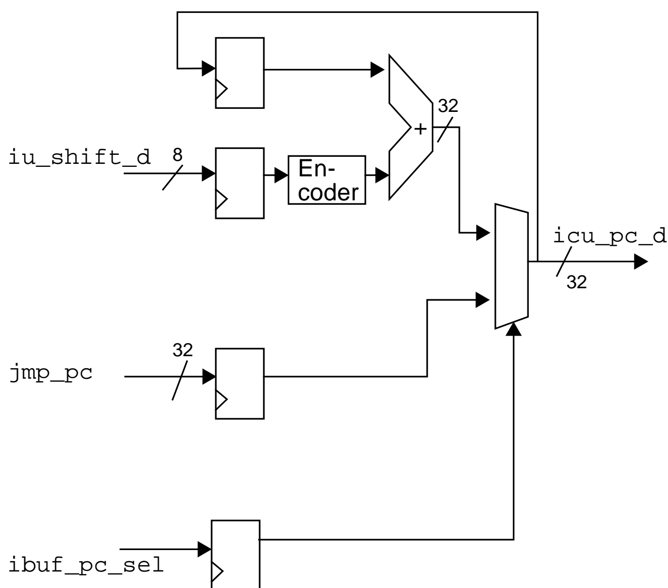


FIGURE 2-3 Program Counter

2.1 Structure

The instruction cache is a direct-mapped, 16-byte line size cache with a single-cycle latency. The cache size is configured to 16-Kbyte sizes (see FIGURE 2-1 on page 8).

Each line has a cache tag entry and an associated valid bit. On a cache miss, the ICU writes 16 bytes of data into the cache from main memory.

A cache tag contains a 18-bit address tag field and one valid bit for a 16-Kbyte size cache.

A 16-byte instruction buffer delinks the fetch stage from the rest of the pipeline for performance reasons. See *Instruction Buffer (I-Buffer)* on page 11 for details.

The process status register (`PSR`)contains an Instruction Cache Enable (`ICE`) bit, which, if disabled, causes instruction fetches to behave as if they had missed in the instruction cache—the ICU fetches the instructions from memory and forwards them to the I-Buffer but does not write them into the instruction cache.

2.2 Functionalities

The ICU does the following:

- Determines the next instruction to be fetched
- Generates address, data, and control signals for the data and the tag RAMs
- Aligns the data out of the RAM and feeds it to the I-Buffer
- Handles noncacheable instruction fetches
- Provides the datapath and control for handling cache misses

The I-Buffer tracks the valid instructions in the queue and updates the entries.

2.2.1 Instruction Cache Control (`ic_cntl`)

Instruction cache control (`ic_cntl`) determines which instruction in the instruction cache to access. Depending on the scenario, it uses the branch target `pc`, the trap `pc`, or the next `pc` for the access.

`ic_cntl` also provides mux selects for the data mux. It selects between the cache fill bus and the miscellaneous bus, which has a diagnostic role. It also keeps track of the current `pc`.

`ic_cntl` generates bus requests due to cache misses or noncacheable requests. On a cache miss, `ic_cntl` bypasses data from the ICU to the I-Buffer as soon as data are available from memory prior to completion of the cache fill. The ICU continues fetching sequential data until the I-Buffer is full or until a branch or trap occurs.

2.2.2 Instruction Buffer (I-Buffer)

As shown in FIGURE 2-4, the I-Buffer is a 16-byte-deep, first-in-first-out (FIFO) buffer.

In a cycle, the ICU adds 8 valid bytes to the I-Buffer from the instruction cache, starting from the first available position. When the I-Buffer is full, the transaction stops. The IU can read out a maximum of 7 valid bytes, also starting from position one.

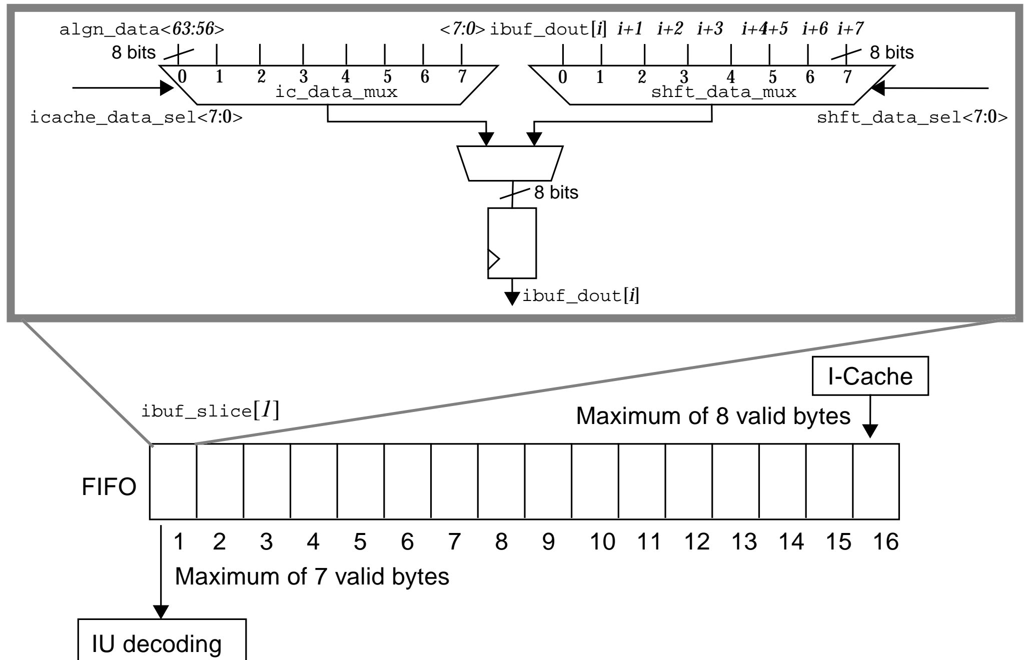


FIGURE 2-4 Instruction Buffer

When a branch or trap occurs, the ICU flushes the entries in the I-Buffer, causing the branch or trap data to move to the top of that buffer.

2.2.3 I-Buffer Control (`ibuf_cntl`)

I-Buffer control (`ibuf_cntl`) provides mux selects and other control signals for the I-Buffer functions. It also keeps track of the valid bits. The I-Buffer is full if the valid entries exceed 8 bytes.

Each I-Buffer entry is associated with:

- A valid bit that indicates that the entry has a valid byte
- A dirty bit that indicates an error in the data fetch
- The length of the instruction that corresponds to that opcode

2.2.4 Instruction Cache Datapath (`icu_dpath`)

The instruction cache datapath (`icu_dpath`) consists of the address and datapath logic (see FIGURE 2-1 on page 8). The data RAM and tag RAMs are not included in the datapath.

The datapath does the following:

- Provides appropriate addresses and data to the RAMs
- Stores valid instructions into the I-Buffer
- Sends appropriate addresses on the memory buses for cache fill transactions
- Supports diagnostic reads and writes into the cache

`icu_dpath` includes the datapath for generating the program counter of the next instruction that is to be decoded.

2.3 Cache Transactions

There are four types of cache transactions: read hits, read misses, noncacheable reads, and indexed flushing.

2.3.1 Cache Read Hits

On a read hit, the ICU does the following:

1. **Fetches 8 bytes of data from the instruction cache and stores them in the I-Buffer in one cycle.**

   The ICU can write a maximum of 8 bytes into the I-Buffer.

2. **Decodes the instruction-length information associated with every byte of the data from the instruction cache into 4 bits per byte and writes them into the length entry of the I-Buffer.**

2.3.2 Cache Read Misses

FIGURE 2-5 illustrate a cache read miss.

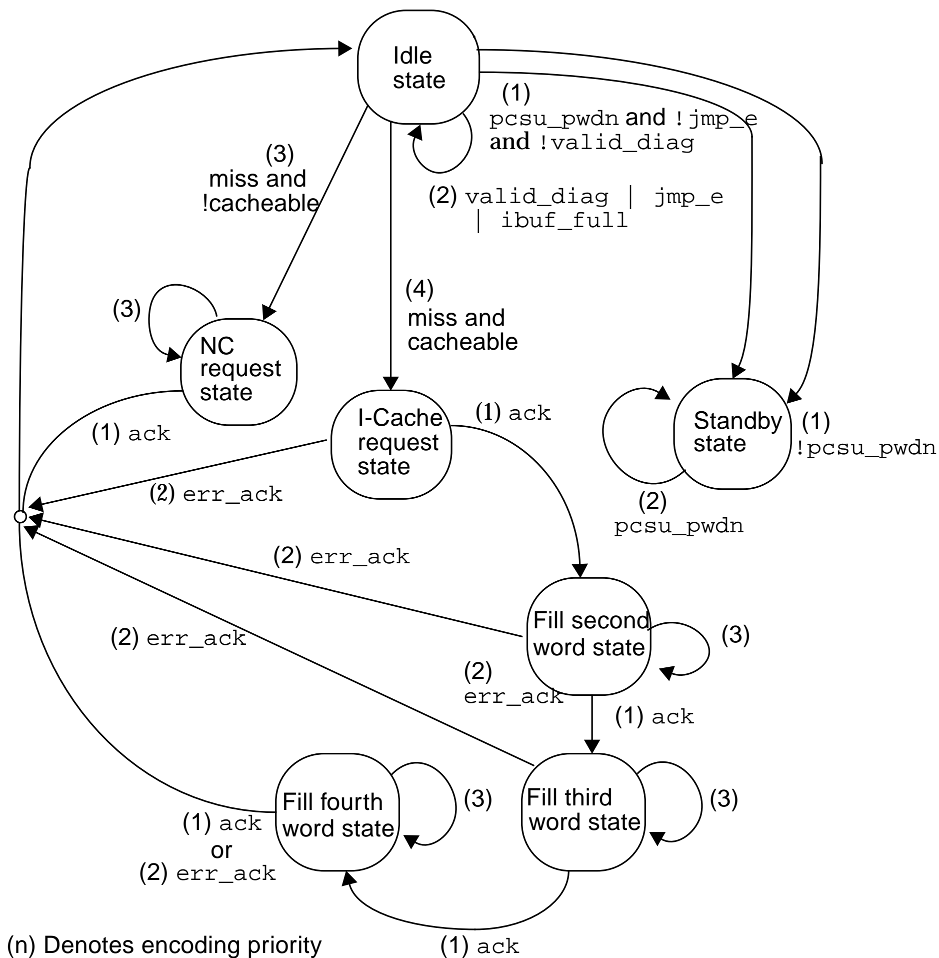


FIGURE 2-5 A Miss State Transaction

On a cache miss:

1. **Miss control generates a cache fill request to the BIU.**
2. **On receiving data from memory, the ICU writes the data into the cache and bypasses them into the I-Buffer.**

   Branches or traps occur only *after* completion of the process.

2.3.3 Noncacheable (NC) Reads

The ICU sends a noncacheable request to the BIU for noncacheable instructions.

Once the data are available on the cache fill bus, the ICU bypasses them into the I-Buffer and does not write them into the instruction cache.

2.3.4 Cache Indexed Flushing

The picoJava-II core supports self-modifying code, and the IU maps `cache_flush` and `cache_index_flush` into a single indexed flush signal, `iu_flush_e`, to the ICU. Thus, the ICU can invalidate an instruction cache line without comparing its tag against the address provided by the IU.

During a flush, the ICU accesses the tag for the indexed line and resets the valid bit.

2.3.5 Error Transactions

If an error occurs during a cache fill transaction, the ICU generates a fault bit and stores it in the I-Buffer along with the instruction. It does not write the line into the instruction cache.

Hence, except for setting the fault bit, an error transaction acts like a noncacheable transaction. The IU takes a trap when it decodes the instruction.

2.3.6 Diagnostic Accesses

The ICU supports diagnostic access instructions, which directly read or write the contents of the tag and data RAMs. The core reads or writes the ICU RAMs for such instructions in the E stage. For details on diagnostic access instructions, see pico*Java-II Programmer's Reference Manual.*

2.4 Power Management

The instruction cache receives a `power_down` signal that turns off the cache when not in use. The cache is also off while waiting for data to return from memory during a cache fill transaction.

For more information on power management, see Chapter 8.

2.5 Interface Signals

The following tables define the ICU interface signals.

TABLE 2-1 ICU Interface with the IU

Signal	Type	Description
iu_br_pc<31:0>	Input	Branch or trap PC
iu_addr_e<31:0>	Input	The address for diagnostic reads and writes into RAMs
misc_din<31:0>	Input	Data bus for diagnostic writes
iu_shift_d<7:0>	Input	The number of bytes data that should be shifted
iu_brtaken_e	Input	Branch instruction in the E stage has been taken
iu_ic_diag_e	Input	Diagnostic read or write
iu_flush_e	Input	Flush instruction in E stage
iu_psr_ice	Input	I-Cache enable bit in the PSR register
iu_psr_bm8	Input	8-bit boot code enable bit from PSR register
diag_ld_cache_c	Output	Diagnostic read to RAM
icu_hold	Output	IU pipe hold for diagnostic access when there is an outstanding transaction in the ICU
icu_dout_d<55:0>	Output	The first 7 bytes of the I-Buffer
ibuf_oplen<27:0>	Output	The opcode length from the I-Buffer
icu_drty_d<6:0>	Output	Dirty output from the first 7 bytes of the I-Buffer
icu_lvd_d<6:0>	Output	Valid output from the first 7 bytes of the I-Buffer
icu_pc_d<31:0>	Output	PC of the first byte of the I-Buffer
misc_dout<31:0>	Output	Data bus for diagnostic reads

TABLE 2-2 ICU Interface with the BIU

Signal	Type	Description
biu_data<31:0>	Input	Data from the BIU for cache fill
icu_biu_addr<31:0>	Output	The address to the BIU for cache fill transaction
biu_icu_ack<1:0>	Input	Acknowledgment from the BIU that data on biu_data bus are available
icu_req	Output	Instruction cache fill request: tv (transaction valid)
icu_size<1:0>	Output	The size of data requested
icu_type<3:0>	Output	The instruction cache transaction type

TABLE 2-3 ICU Interface with the Instruction Cache Tag

Signal	Type	Description
itag_vld	Input	Valid bit from the tag RAM
ic_hit	Input	Hit from the tag comparator
itag_dout	Input	Tag data from tag RAM
icu_tag_in<17:0>	Output	Data to be written into the tag RAM
icu_tag_vld	Output	Valid bit to be written into the tag RAM
icu_tag_we	Output	Write enable for tag array
icu_tag_addr	Output	The address for accessing tag
icram_powerdown	Output	Tag RAM cell's powerdown pin

TABLE 2-4 ICU Interface with the Instruction Cache

Signal	Type	Description
icram_dout<63:0>	Input	Data from the instruction RAM
icu_din<31:0>	Output	Data to the instruction RAM
icu_addr<13:3>	Output	Address for accessing the instruction RAM
icram_powerdown	Output	RAM cell's powerdown pin
icu_ram_we	Output	Write enable

TABLE 2-5 ICU Interface with the PCSU

Signal	Type	Description
pcsu_powerdown	Input	PCSU request for powerdown
icu_in_powerdown	Output	ICU notification to the PCSU that it is ready for clock shutoff

CHAPTER 3

Integer Unit (IU)

The Integer Unit (IU) executes all the instructions that are defined in *The Java Virtual Machine Specification*, except for the floating-point-related instructions, which are executed in the Floating Point Unit (FPU).

This chapter contains the following sections:

- *Instruction Pipeline* on page 20
- *Instruction Folding Unit (IFU)* on page 21
- *Register Control Unit (RCU)* on page 25
- *Datapath* on page 29
- *Pipe Control Unit* on page 32
- *Microcode* on page 35

At the front-end, the IU interacts with:

- The Instruction Cache Unit (ICU) to fetch instructions
- The Floating Point Unit (FPU) to execute floating point instructions
- The Data Cache Unit (DCU) to execute load-related and store-related instructions

After fetching new instructions from the ICU, the IU does the following:

- Groups the instructions in the Instruction Folding Unit (IFU)
- Accesses registers and provides bypass signals in the Register Control Unit (RCU)
- Decodes the operation in the decode unit and executes the instruction in the execution and microcode datapaths

FIGURE 3-1 illustrates the IU.

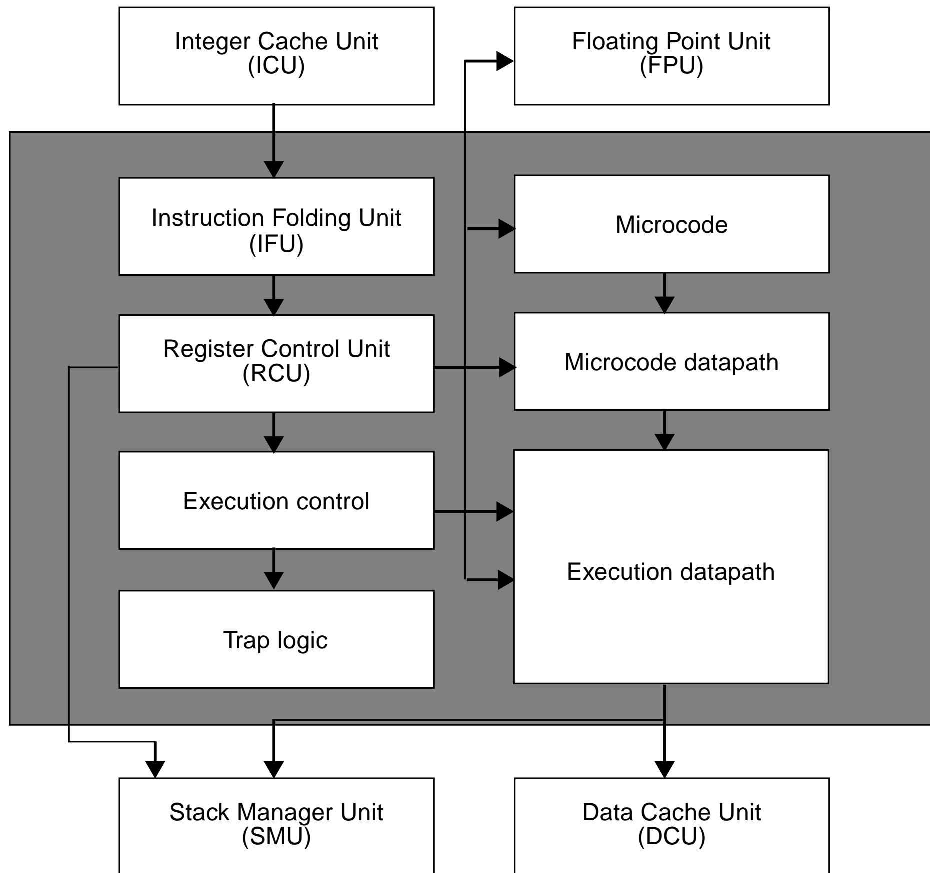


FIGURE 3-1 Integer Unit (IU)

3.1 Instruction Pipeline

The instruction pipeline consists of six stages, as shown in FIGURE 3-2.

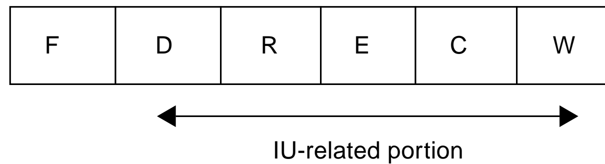


FIGURE 3-2 Instruction Pipeline in the IU

Here are the pipeline stages in the sequence in which they occur:

- **Fetch (F) stage** — The ICU fetches instructions from either the instruction cache in the ICU or from external memory.
- **Decode (D) stage** — The IU groups and precodes instructions in the Instruction Folding Unit (IFU).
- **Register (R) stage** — The IU fetches the operands from the stack cache and determines load-use conditions, bypass conditions, and stack cache miss conditions. The logic for this stage is in the Register Control Unit (RCU).
- **Execution (E) stage** — The IU uses the ALU to either compute arithmetic or calculate the address of a load or store from the DCU. All multicycle instructions use the microcode datapath for execution.
  - In case of a control flow instruction, the IU calculates the branch address and the condition upon which the branch is dependent.
  - In case of a floating-point instruction, the IU provides the operands to the FPU for execution.
- **Cache (C) stage** — The IU accesses data from the data cache, prioritizes, and takes traps at the end of the cycle.
- **Write (W) stage** — The IU writes back results to the stack cache.

3.2 Instruction Folding Unit (IFU)

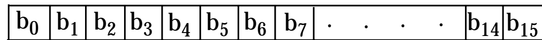
The Instruction Folding Unit (IFU) does the following:

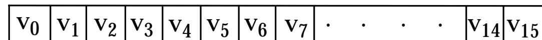
- Examines the top 7 bytes in the instruction buffer (I-Buffer) to determine how many instructions can be folded (up to a maximum of four)
- Decodes the instructions and provides the result to the R stage and sends the shift signal, which indicates the number of bytes consumed, to the I-Buffer

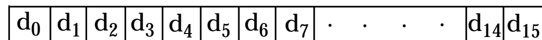
3.2.1 I-Buffer Interface

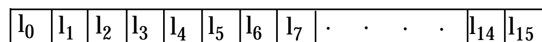
The I-Buffer is a 16-byte-deep prefetch buffer. Each I-Buffer byte, b_i, contains:

- A valid bit, v_i, which indicates whether that byte is valid
- A dirty bit, d_i, which indicates whether that byte is dirty
- A 4-bit length information, l_i, which indicates the length of the instruction that corresponds to that opcode.









3.2.2 Folding Logic

The folding logic determines how many instructions can be dispatched in a cycle according to the instructions and length information from the ICU. The core can dispatch two operand loads (one operation and a result store) and emulate RISC-like execution.

Folding is disabled if one of the following conditions is true:

- The `PSR.FLE` bit is set to zero.
- Instructions that cause a memory exception are present in the I-Buffer.
- The local variables are not in the stack cache.

A folded group of instructions is reexecuted after folding is disabled if an instruction in a folded group traps.

3.2.3 Folding Groups

The IFU classifies instructions into six types and groups them for the opcodes, as listed in TABLE 3-1.

TABLE 3-1 Instruction Types in the IFU

LV	A local variable load or load from global register or push constant
OP	An operation that uses the top two entries of stack and that produces a one-word result
BG2	An operation that uses the top two entries of the stack and breaks the group
BG1	An operation that uses only the topmost entry of stack and breaks the group
MEM	A local vars store, global register store, and memory load
NF	A nonfoldable instruction

TABLE 3-2 lists the groupings; each line represents a group.

TABLE 3-2 IFU Groupings

LV	LV	OP	MEM
LV	LV	OP	
LV	LV	BG2	
LV	OP	MEM	
LV	BG2		
LV	BG1		
LV	OP		
LV	MEM		
OP	MEM		

The IFU takes the top 7 bytes of the I-Buffer and arranges the instructions according to a RS1, RS2, OP, RD format, as shown in FIGURE 3-3.

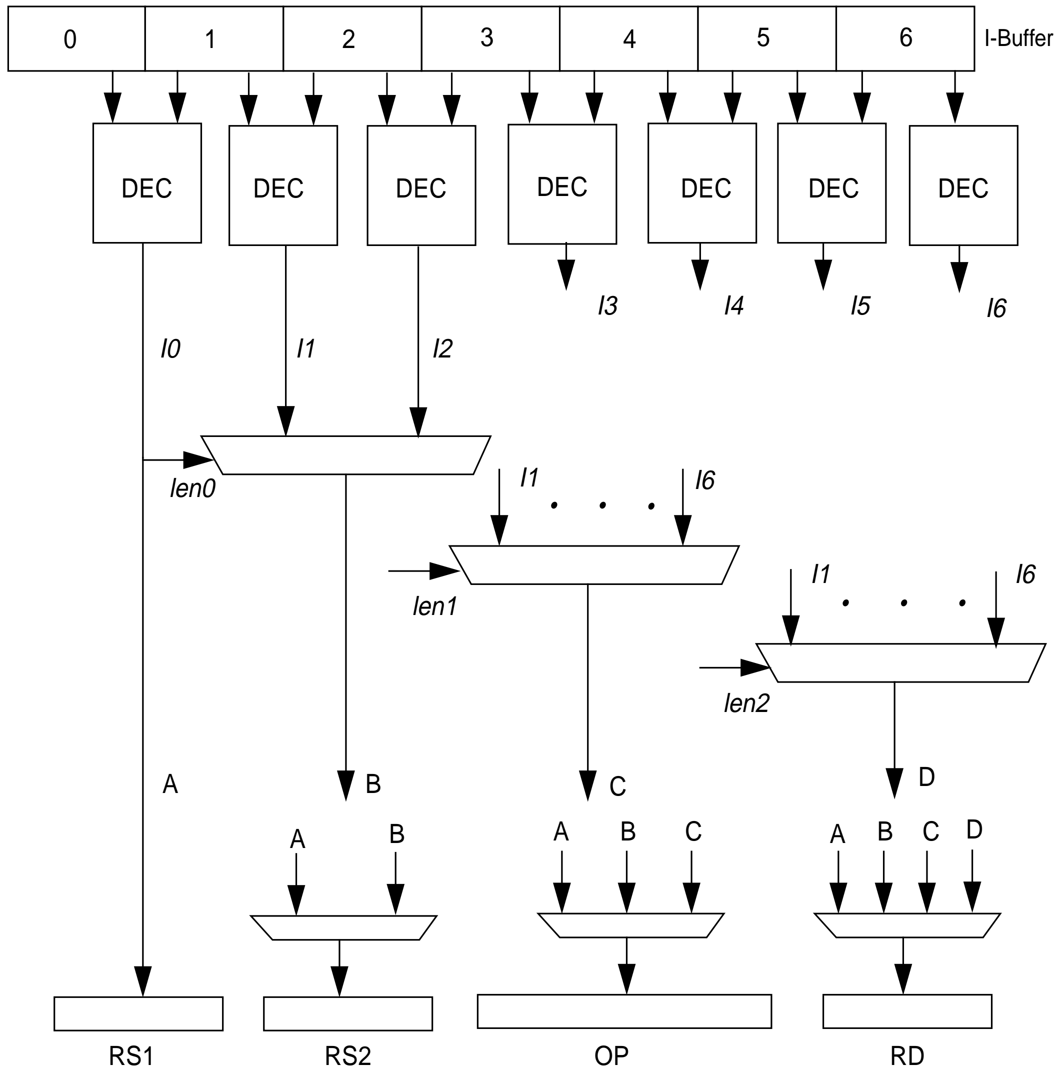


FIGURE 3-3 Instruction Folding Unit (IFU) Datapath

3.3 Register Control Unit (RCU)

The Register Control Unit (RCU) consists of:

- The register access logic, which generates the constants, fetching operands from the stack and determining stack cache hits or misses
- The bypass logic, which generates the control for bypass signals that are used by the IU to bypass data from older stages in the E stage. It also generates the load use case, during which the IU stalls the D and R stages of the pipe.
- The destination logic, which determines the destination address for each operation in the R stage, keeps track of the destination address down the pipe, and writes to the stack cache.

3.3.1 RCU Datapath

The register access datapath contains:

- The stack cache
- The adder for computing the address of the operands in the stack
- Comparators for the bypass logic
- The destination pipe to keep track of destination addresses

The RCU computes the local variable addresses to access the stack cache. Some instructions push constants onto the stack, in which case the RCU generates the constants in the runit.

By default, the RCU accesses the top two entries of the stack. If an operation uses more than two entries of the stack, however, the RCU uses a help bit, generated by the IFU, which signals for four operands over two cycles.

FIGURE 3-4 shows the runit datapath for RS1. RS1 outputs operand 1 for the ALU. Similarly, an RS2 datapath provides operand 2.

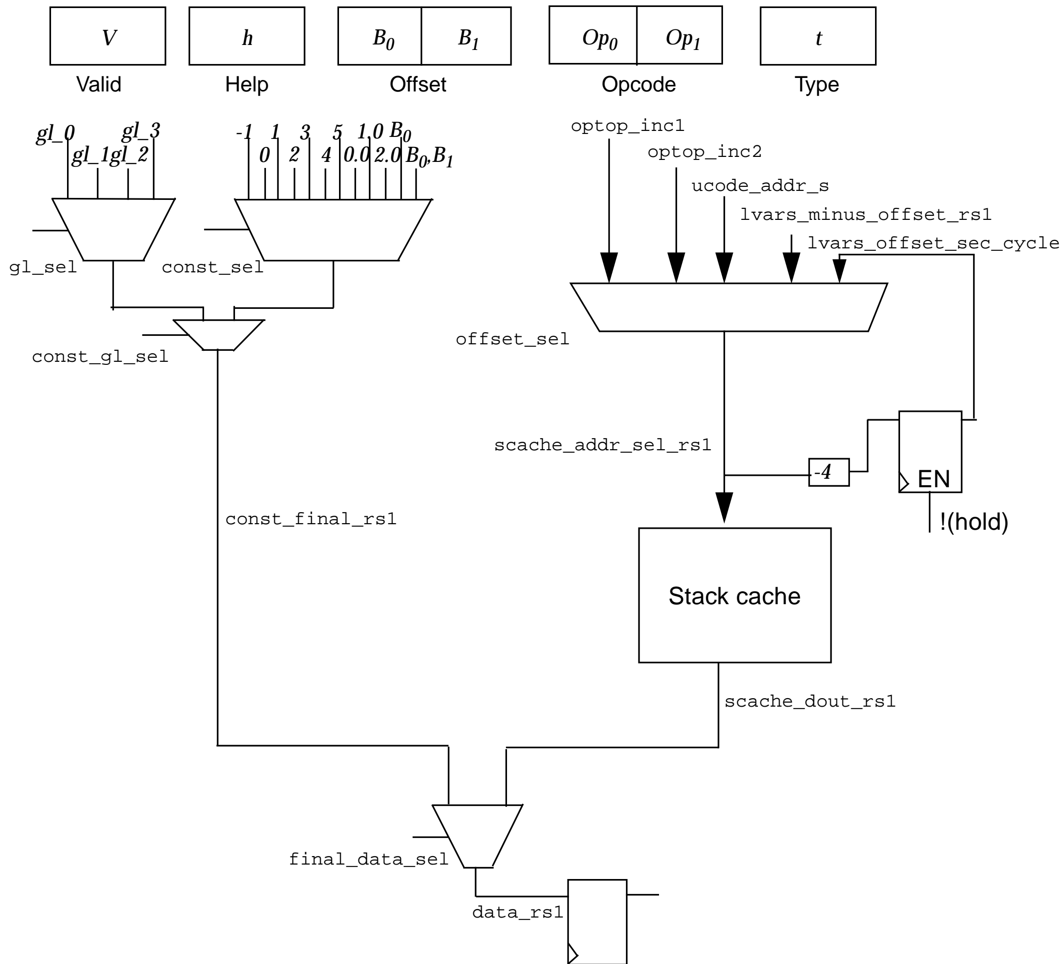


FIGURE 3-4 Register Control Unit (RCU) Datapath (Register Access Part for RS1)

To determine bypassing, the RCU compares the operand addresses with the destination addresses in the E, C, and W stages of the pipeline. Actual data bypassing occurs in the E stage because of timing.

The RCU also compares the C stage destination addresses with the stack bottom to determine stack cache hits or misses. In a hit, the RCU writes the data into the stack cache; in a miss, it forwards the data to the SMU, to be written out to memory.

The RCU datapath has the four global registers, which it writes to only in the W stage and reads in the R stage. The bypass logic also keeps track of writes to global registers.

3.3.2 Interface Signals

TABLE 3-3 through TABLE 3-6 define the interface signals.

TABLE 3-3 Block Interface with the IFU

Signal	Type	Description
opcode_1_rs1_r<7:0>	Input	First byte of the opcode in RS1 stage
opcode_2_rs1_r<7:0>	Input	Second byte of the opcode in RS1 stage
opcode_1_rs2_r<7:0>	Input	First byte of the opcode in RS2 stage
opcode_2_rs2_r<7:0>	Input	Second byte of the opcode in RS2 stage
opcode_1_op_r<7:0>	Input	First byte of the opcode in OP stage
opcode_2_op_r<7:0>	Input	Second byte of the opcode in OP stage
opcode_1_rsd_r<7:0>	Input	First byte of the opcode in RSd stage
opcode_2_rsd_r<7:0>	Input	Second byte of the opcode in RSd stage
valid_rsd_r	Input	RD is of MEM type
offset_1_rs1_r<7:0>	Input	First byte of offset in RS1 stage
offset_2_rs1_r<7:0>	Input	Second byte of offset in RS1 stage
offset_1_rs2_r<7:0>	Input	First byte of offset in RS2 stage
offset_2_rs2_r<7:0>	Input	Second byte of offset in RS2 stage
valid_rs1_r	Input	Indication that the opcode in RS1 is valid
valid_rs2_r	Input	Indication that the opcode in RS2 is valid
type_rs1_r<7:0>	Input	Indication of the type of long operation in RS1
lvars_acc_rs1_r	Input	Assertion for all operations that access the stack cache using the lvars register in RS1
group_1_r	Input	Indication of a group 1 folding
group_2_r	Input	Indication of a group 2 folding
group_3_r	Input	Indication of a group 3 folding
group_4_r	Input	Indication of a group 4 folding
group_5_r	Input	Indication of a group 5 folding
group_6_r	Input	Indication of a group 6 folding
group_7_r	Input	Indication of a group 7 folding

TABLE 3-3 Block Interface with the IFU *(Continued)*

Signal	Type	Description
group_8_r	Input	Indication of a group 8 folding
group_9_r	Input	Indication of a group 9 folding
no_fold_r	Input	Indication of no folding

TABLE 3-4 Block Interface with the IU

Signal	Type	Description
rs1_data_e<31:0>	Output	Operand 1
rs2_data_e<31:0>	Output	Operand 2
rs1_forward_mux_sel<3:0>	Output	Bypass controls for RS1
rs2_forward_mux_sel<3:0>	Output	Bypass controls for RS2
iu_lvars<31:0>	Input	Architectural VARS register
iu_sc_bottom<31:0>	Input	Architectural SC_BOTTOM
iu_data_w<31:0>	Input	Data to be written to the stack
ucode_areg0<31:0>	Input	Address from microcode for data to stack
ucode_addr_s	Input	Microcode address to access the stack cache
rs1_forward_mux_sel	Output	Mux selects for data forwarding

TABLE 3-5 Block Interface with Pipeline Control

Signal	Type	Description
scache_rd_miss_e	Output	Stack cache read miss
optop_e<31:0>	Input	OPTOP from the E stage
inst_vld<2:0>	Input	Indication of a valid instruction in the W, C, and E stages
iu_trap_r	Input	Indication of a trap in the R stage
hold_e	Input	Indication of a hold in the E stage
hold_c	Input	Indication of a hold in the C stage
optop_offset	Output	Indication of an OPTOP net change

TABLE 3-6 Block Interface with the SMU

Signal	Type	Description
smu_rf_addr<5:0>	Input	Address for read/write from dribbler
smu_data<31:0>	Input	Data from SMU for fills
iu_smu_data<31:0>	Output	Read port of SMU
dest_addr_w<31:0>	Output	Stack cache write miss address
scache_wr_miss_w	Output	Stack cache write miss request
smu_we	Input	Write enable signal from the dribbler

3.4 Datapath

The IU datapath of the E stage pipeline is a one-cycle execution engine used by simple instructions that take only one cycle, as well as by the microcode for multiple-cycle instructions and calculation of addresses.

The major functional units in the datapath are:

- Main adder
- Address adder
- Comparator, shifter
- Bit-wise operator
- Integer converter

The IU also maintains most architecture registers inside the datapath, except for:

- PC and OPTOP, which it monitors in pipe control
- GLOBAL0, GLOBAL1, GLOBAL2, and GLOBAL3, which it monitors in the R stage

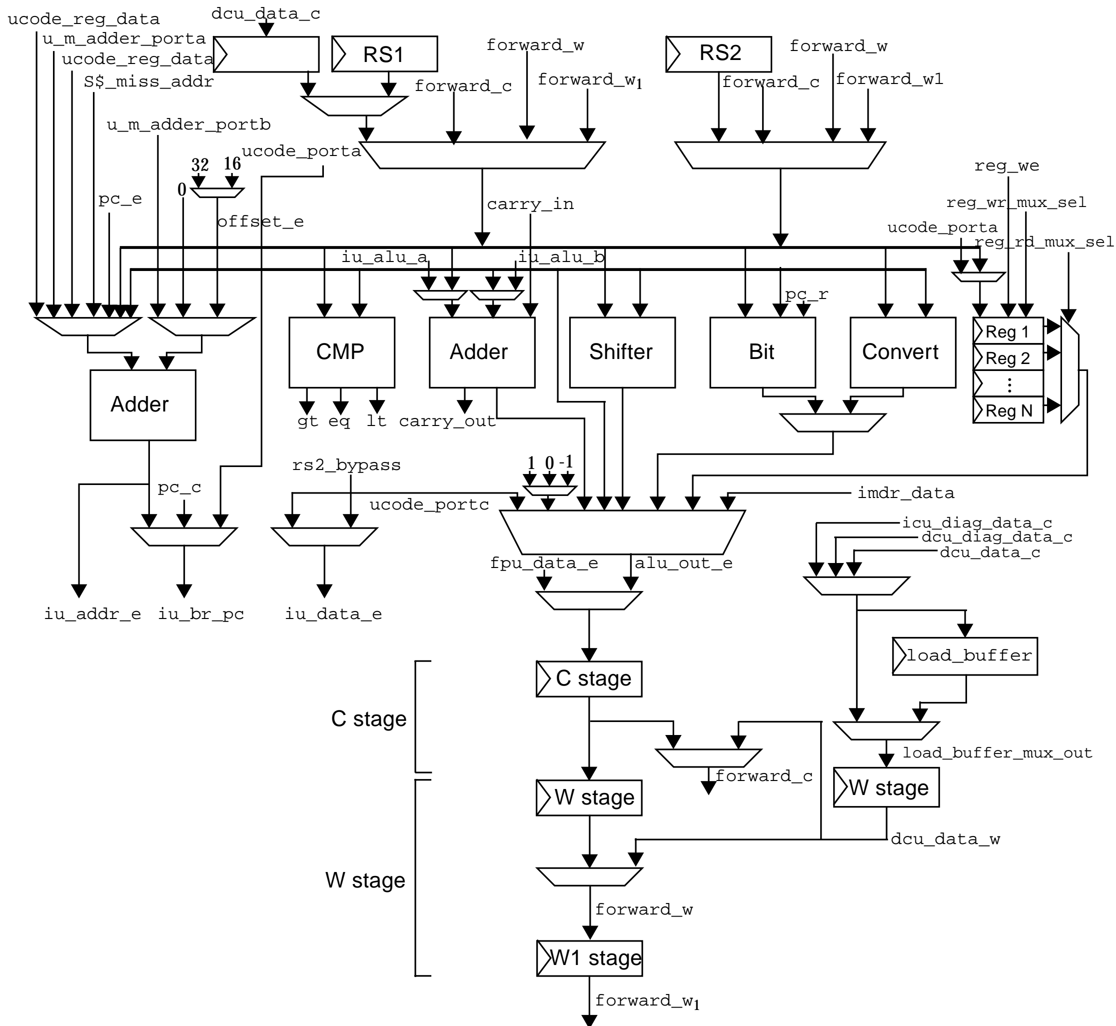


FIGURE 3-5 Integer Unit (IU) Datapath

3.4.1 Functionalities

The IU datapath has the following functionalities:

- **Add, sub, neg** — Use the main adder. carry_in and carry_out are the interface signals. For 64-bit arithmetic operations, use carry_out as the carry_in for the next stage.

- **Compare, compare with zero** — Use the comparator, which supports both signed and unsigned comparisons. There are three output signals: `gt`, `eq`, and `lt`.
- **Left and right shift** — Use the shifter, which supports the left shift, the right logical shift, and the right arithmetic shift.
- **AND, OR, XPR** — Use the bit-wise operator.
- **i2b, i2s, i2c, i2l** — Use the integer converter, which supports sign extensions in 8-bit and 16-bit conversions.
- **RS1 value** — This value goes directly to the output for constant push instructions and local var store instructions.
- **RS2 value** — In an extended store, the IU bypasses this value to `iu_data_e`.
- **Address adder** — This adder calculates the addresses for:
  - Microcode
  - Branch targets
  - Stack cache misses

3.4.2 Interface Signals

TABLE 3-7 defines the interface signals for the IU datapath.

TABLE 3-7 Block Interface of the IU Datapath

Signal	Type	Description
ru_rs1_e<31:0>	Input	src1 operand from the stack cache
ru_rs2_e<31:0>	Input	src2 operand from the stack cache
scache_miss_addr_e<31:0>	Input	Address of stack cache access miss
dcu_data_c<31:0>	Input	Data of stack cache access miss (from the data cache)
ucode_porta_e<31:0>	Input	src1 operand from microcode
ucode_portc_e<31:0>	Input	Microcode result
u_m_adder_porta_e<31:0>	Input	Address 1 of microcode address adder
u_m_adder_portb_e<31:0>	Input	Address 2 of microcode address adder
pc_e<31:0>	Input	PC value from the E stage
pc_c<31:0>	Input	PC value from the C stage
imdr_data_e<31:0>	Input	Multiply-divide-remainder result
fpu_data_e<31:0>	Input	FPU result
icu_diag_data_c<31:0>	Input	Diagnostic read from the ICU
dcu_diag_data_c<31:0>	Input	Diagnostic read from the DCU

TABLE 3-7 Block Interface of the IU Datapath *(Continued)*

Signal	Type	Description
carry_out_e	Output	Carry output of the main adder
iu_addr_e<31:0>	Output	Output of address adder to the data cache and the instruction cache
iu_data_e<31:0>	Output	Bypassed IU data to the data cache and the instruction cache
iu_br_pc_e<31:0>	Output	Branch target PC
alu_out_w<31:0>	Output	Output of ALU

3.5 Pipe Control Unit

The Pipe Control Unit (PCU) contains a decode unit, which provides the control for the IU, and pipe logic, which keeps track of PC and OPTOP registers (as shown in FIGURE 3-6). It also generates pipeholds for the various stages.

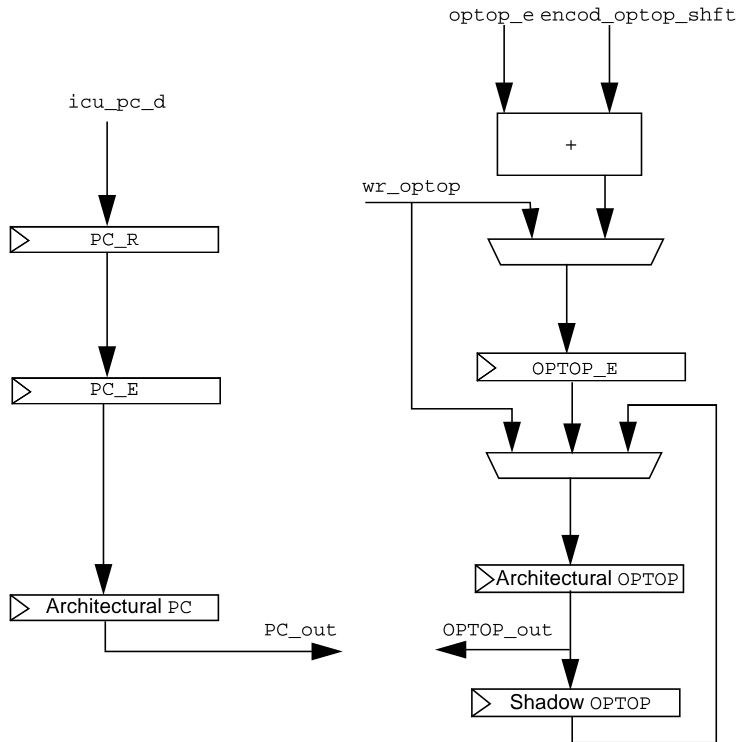


FIGURE 3-6 pipe_dpath (PC Pipe and OPTOP Pipe)

The decode unit decodes all the single-cycle operations for the IU. Microcode handles multicycle instructions.

The different holds from different blocks on the core are resolved in the pipe logic to generate a pipeline hold. Maintenance of PC pipe and OPTOP registers takes place in the PCSU.

3.5.1 Stall Types

All state changes and therefore all OPTOP changes occur only at the end of the E stage.

Following are the various types of pipeline stalls in the core:

- dcu_hold — This stall holds the pipeline DREC due to a load miss. Once the data are available, iu_data_vld latches the data into the cache register.
- smu_hold — This stall holds the pipeline DREC in an underflow or overflow. All writes to stack cache take place in the W stage.

- pj_halt — The pipeline DREC is on hold when this external signal is asserted.
- pj_in_halt — The pipeline DREC is on hold when this signal is asserted at instruction and data breakpoints.
- ucode busy — This stall holds the pipeline DR when the microcode is active. OPTOP updates cannot occur in the last cycle of a microcode operation.
- imdr busy — This stall holds the pipeline DR when the multiply/divide/remainder unit is busy.
- fpu busy — This stall holds the pipeline DR when the FPU is active.
- scache hold — This stall holds the pipeline DR on a local variable load miss in the R stage when the instruction moves to the E stage. The hold is deasserted once data are available from the data cache.
- icu_hold — This stall holds the pipeline DR during diagnostic reads and writes to the I-Cache or I-Tag.
- multicycle hold — This stall holds the pipeline D during the fetch of operands for long and double operations.
- lduse hold — This stall holds the pipe for one cycle in a load use—a load in the E stage and a use in R stage.

3.5.2 Interface Signals

TABLE 3-8 defines the interface signals for pipe control.

TABLE 3-8 Interface Signals for Pipe Control

Signal	Type	Description
icu_pc_d<31:0>	Input	PC of the first instruction in the I-Buffer
inst_vld_r	Input	Valid operation
fold_r	Input	Folded group in the R stage
pc_offset_r<2:0>	Input	Offset that calculates end PC
arch_optop<31:0>	Output	Architectural OPTOP
arch_pc<31:0>	Output	Architectural PC
wr_optop_e	Input	Update of the architectural OPTOP register
iu_data_e<31:0>	Input	Data bus to write to OPTOP
iu_trap_c	Input	Trap taken in the C stage
reissue_c	Output	Flush pipe and reissue instruction in the C stage
squash_fold	Output	No folding: used when folding group traps

3.6 Microcode

Microcode implements the multicycle instructions. It has its own datapath, which consists mainly of eight temporary registers and a ROM (284x80), but shares two adders in the IU.

The microcode interface with the main IU datapath is shown in FIGURE 3-7.

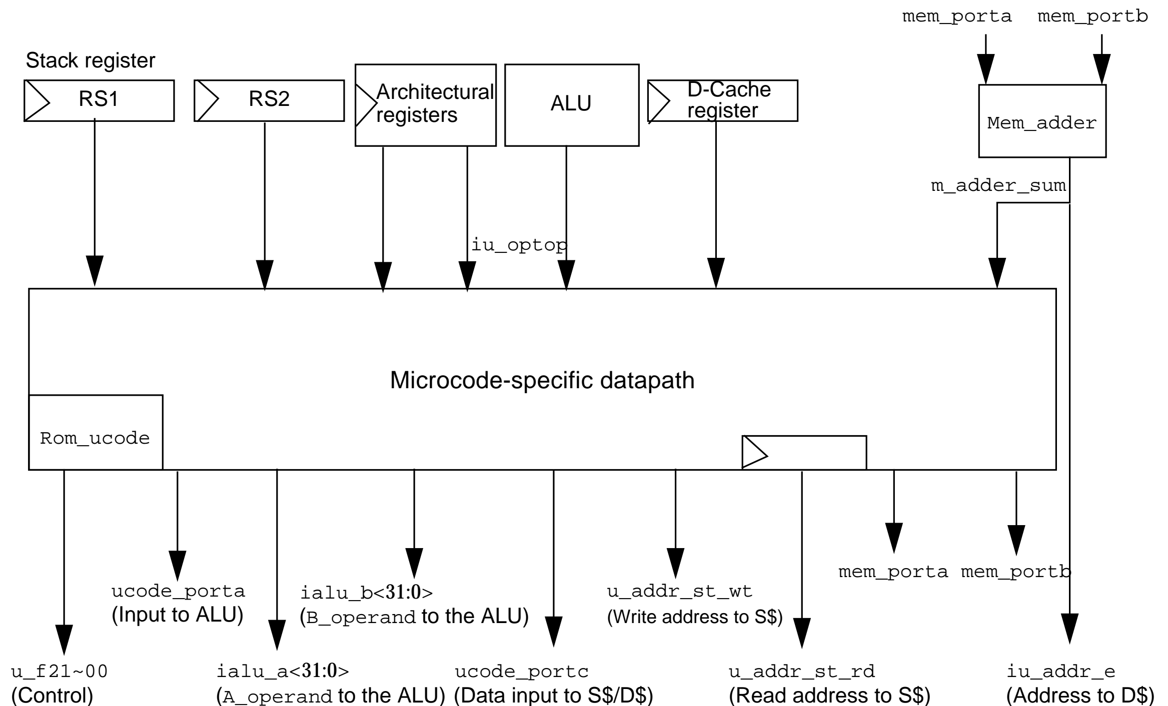


FIGURE 3-7 Microcode Interface with the Main IU Datapath

3.6.1 Interface Protocol

The microcode interface protocol is as follows:

- In the stack cache, only one read port and one write port are available to the microcode; it *cannot* read and write at the same stack cache address.
- In the architectural registers, only one read port and one write port are available to the microcode; it *can* read and write to the same register.
- At the beginning of a microcode operation, the top two entries of the stack are available in the `rs1` and `rs2` registers. If microcode makes a data cache read request, the `rs1` value changes. The `rs2` value stays the same during a microcode operation.
- `U_addr_st_rd`, the `read_address` pointer to the stack cache, must be an early signal.
- `U_addr_st_wt`, the `write_address` pointer to the stack cache, can be a late signal. `ucode_portc` is the data input to the stack cache.
- `ucode_addr_d`, the address pointer to the data cache, is the output of the `Mem_adder`. `ucode_portc` is the data input to the `data_cache`.
- If microcode issues a stack cache read in cycle X, data are available to use in cycle X + 1 in the `rs1` stack register.
- If microcode issues a data cache read in cycle X, data are available to use in cycle X + 2 in the data cache register (`dreg`).
- Microcode can access both the stack cache and the data cache in a given cycle.
- Stack cache and data cache misses are transparent to microcode. In case of a miss, `ie_stall_ucode` stalls microcode.
- If data from the IU to microcode are invalid, the IU issues `ie_stall_ucode` to hold the microcode process.
- If microcode issues a stack cache read request and then a data cache request in the next cycle, a stack cache read miss and data cache request can occur in the same cycle. Thus, microcode must complete the stack cache miss first and then process the data cache request. Also, the `iu_addr` mux muxes between the load miss and the ALU output.
- Microcode can access the data cache and stack cache on a back-to-back basis.
- Microcode can request the IU to copy data from the data cache to the stack cache by providing addresses for both caches.
- Microcode can cancel a data cache request a cycle late and can therefore issue data cache requests prior to determining an exception.
- Microcode can cancel a data cache or stack cache request a cycle late when it asserts the `u_abt_rdwt` signal. Unlike an exception, however, the microcoded instruction continues to execute.
- Microcode asserts the `u_done` signal in the last cycle of an operation.

- Microcode cannot update OPTOP or VARS in the last cycle.
- Once microcode has changed the VARS, FRAME, or CONST_P register, it cannot make data cache read requests.
- Synchronous interrupt can only take place during the first cycle of a microcode operation. The IU ensures no state changes even if microcode has made requests.
- Asynchronous interrupts can occur at any time. Microcode may have made state changes, however, which are not recoverable.
- Only ie_stall_ucode stalls microcode.
- iu_trap_r and ifu_op_valid_r are never active at the same time. Microcode does not decode soft_trap instructions.
- To support the OPLIM check in the IU, microcode first updates OPTOP for all invoke_*_quick instructions, then builds the new frame on the stack and changes the related architecture registers.
- If a microcoded instruction completes in one cycle, the u_done signal stays high.
- The IU does not allow a microcode read of stack_cache if the address is either OPTOP + 4 or OPTOP + 8.

3.6.2 Control Logic

The microcode control logic picks up the instruction in the R stage. The ROM address is generated for all multicycle operations. By default, the address points to location 0x0, which has the default control values. The address accesses a 300x80-entry asynchronous ROM. The microcode control also has branching mechanism for handles and exceptions (see TABLE 3-9).

TABLE 3-9 Block Interface of Microcode Control

Signal	Type	Description
opcode_1_op_r<7:0>	Input	Operation with an immediate operand
opcode_2_op_r<7:0>	Input	Operation with an immediate operand
opcode_3_op_r<7:0>	Input	Operation with an immediate operand
valid_op_r	Input	Validation of an operation
iu_trap_r	Input	Issue of a trap operation (R stage) from the IU
iu_hold_e	Input	Holding of microcode in the IU (multiple E stages)
iu_psr_gce	Input	Enabling of garbage collection
u_f23~00<79:0>	Output	Control decode for the E stage
u_done	Output	Completion of microcode (not busy)

TABLE 3-9 Block Interface of Microcode Control *(Continued)*

Signal	Type	Description
u_ref_null	Output	Null array_ or object_ref
u_ary_ovf	Output	Indication that the index in an array is out of bounds
u_ptr_un_eq	Output	Two not-equal class pointers
u_gc_notify	Output	Detection of a condition for garbage collection
ie_stall_ucode	Input	An IU stall of a microcode operation
ie_kill_ucode	Input	An IU termination of a microcode operation
iu_hold_e	Input	IU hold of the E stage
iu_psr_gce	Input	Garbage collection enable bit from the PSR
ie_alu_cryout	Input	Carry-out of the ALU adder
ie_comp_a_eq0	Input	Signal that indicates that operand A equals to 0

3.6.3 Datapath

The datapath consists of temporary registers and muxes to control the five read ports and three write ports to the microcode-specific datapath (see TABLE 3-10).

TABLE 3-10 Block Interface of the Microcode Datapath

Signal	Type	Description
rs1<31:0>	Input	Top of stack, also used for reading stack
rs2<31:0>	Input	The second operand from the stack
dreg<31:0>	Input	Data from data_cache
alu_out<31:0>	Input	Output of ALU
archi_data<31:0>	Input	Output of architectural registers
iu_optop<31:0>	Input	OPTOP register
ucode_porta<31:0>	Output	Port A of microcode to ALU
ucode_portb<31:0>	Output	Port B of microcode to ALU
ialu_a<31:0>	Output	A_operand to the ALU of the IU
ucode_portc<31:0>	Output	Port C of microcode, data to stack_ or data_cache
u_areg0<31:0>	Output	Signal for calculating the write address to the stack cache
u_addr_st_rd<31:0>	Output	Signal for reading the address port to the stack cache

TABLE 3-10 Block Interface of the Microcode Datapath *(Continued)*

Signal	Type	Description
u_f01_wt_stk	Output	Microcode write to the stack cache request
u_f02_rd_stk	Output	Microcode read from the stack cache request
u_m_adder_porta<31:0>	Output	A_operand input to the memory adder
u_m_adder_portb<31:0>	Output	B_operand input to the memory adder
m_adder_sum<31:0>	Input	Output of the mem_adder(=ucode_addr_d)
u_abt_rdwt	Output	Termination of the access to the data cache or stack cache

CHAPTER 4

Floating Point Unit (FPU)

The Floating Point Unit (FPU) resolves all mathematical instructions.

This chapter contains the following sections:

- *Structure* on page 41
- *Pipeline* on page 43
- *Add and Subtract* on page 44
- *Multiply and Divide* on page 45
- *IEEE 754 Compliance* on page 45
- *Opcodes* on page 47
- *Power Management* on page 48
- *Interface* on page 48
- *Critical Paths* on page 51
- *Signals* on page 52

4.1 Structure

The FPU is organized into four basic sections, as shown in FIGURE 4-1.

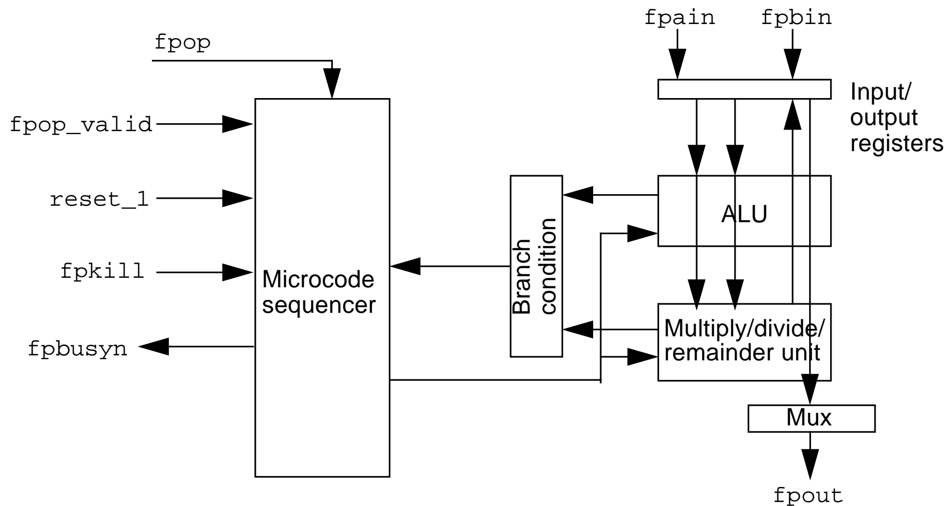


FIGURE 4-1 Floating Point Unit (FPU)

The four basic sections are:

- **Microcode sequencer** — Controls the microcode flow and microcode branches.
- **Input-output registers** — Controls input-output data transactions. This section also provides the input data loading and output data unloading registers for intermediate result storage.
- **Floating point adder-ALU** — Includes the combinatorial logic that performs the floating-point adds, floating-point subtracts, and conversion operations.
- **Floating point multiply/divide unit** — Contains the hardware for performing multiply, divide, and remainder operations.

The FPU module is organized as a microcoded engine with a 32-bit datapath, which is often reused many times during computation of the result.

Double-precision (DP) operations require approximately four times the number of cycles as single-precision (SP) operations.

During typical operation, the microcode sequencer presents a new microword to the Datapath Unit. It monitors the returning branch condition to determine the next microword.

The FPU datapath consists of two main sections: ALU and multiply/divide (MDIV).

4.1.1 ALU

The ALU contains a 32-bit datapath with compute elements that perform floating point add, subtract, compare, and conversion operations, as well as all normalization calculations.

4.1.2 Multiply/Divide (MDIV)

The MDIV contains a multiply array logic section, which can perform array calculation for SP floating-point multiply in two cycles. Therefore, the entire single-precision multiply requires four cycles, except in special cases.

For each double-precision floating-point multiply, the picoJava-II core uses this multiply array logic section eight times, thus requiring approximately 15 cycles for double-precision floating-point multiply, depending on the data.

4.1.3 Microcode Sequencer

The microcode sequencer (MSEQ) requires two 160-entry ROMs. Other control structures include the microcode `PC` and the next microcode calculation logic. This logic contains many incoming branch signals that control the microcode flow. A given field in each microcode word selects the appropriate branch condition.

When an `fpop` command is issued, the FPU uses hardware to map it to the correct microcode starting address.

4.2 Pipeline

Output data are available for reading one cycle after the `fpbusyn` signal is deasserted. For double-precision output data, two read cycles are required to obtain both output data words, as shown in FIGURE 4-2.

The `fpbusyn` signal is asserted low in the cycle following the detection of a valid opcode. It remains asserted until the cycle before the first data is output.

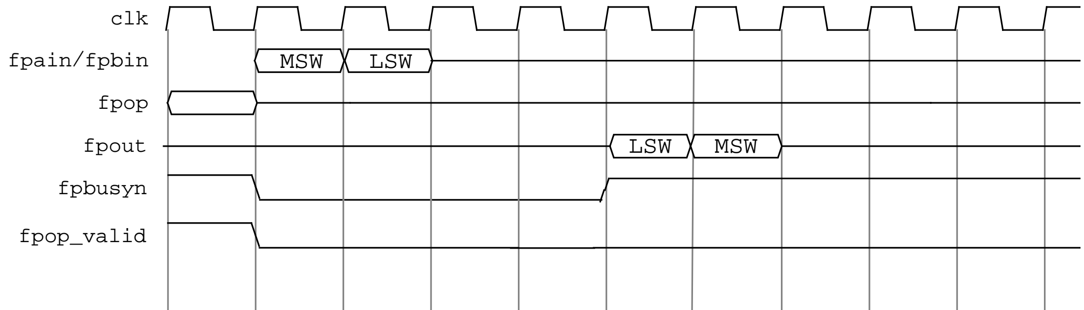


FIGURE 4-2 Double-Precision Operation Using Two Cycles

4.3 Add and Subtract

A branch condition might be asserted after each *primitive* microcycle operation, which can then result in a change in program flow. FIGURE 4-3 is a simplified block diagram of the ALU section of the FPU.

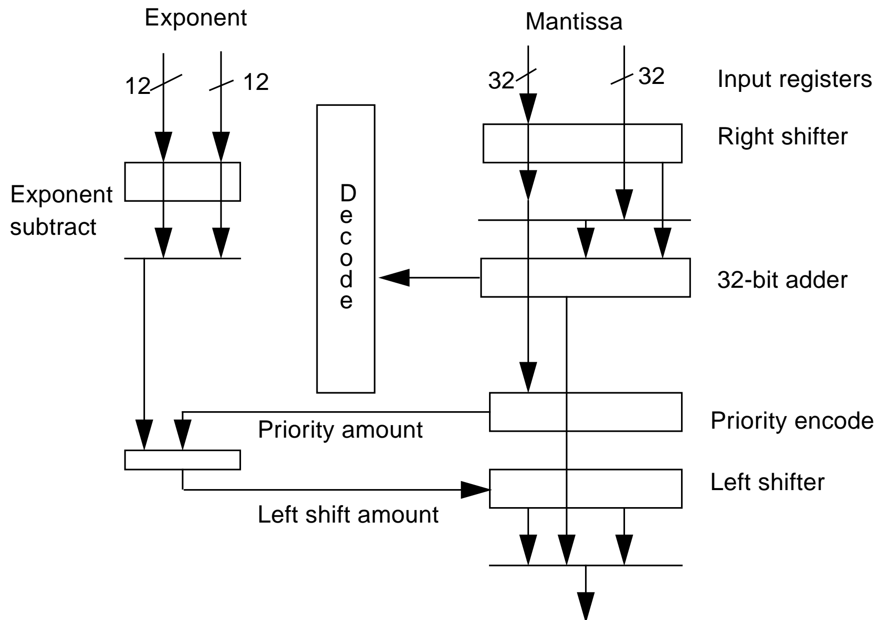


FIGURE 4-3 Simplified ALU Section of the Floating Point Unit (FPU)

4.4 Multiply and Divide

FIGURE 4-4 shows the MDIV section of the FPU.

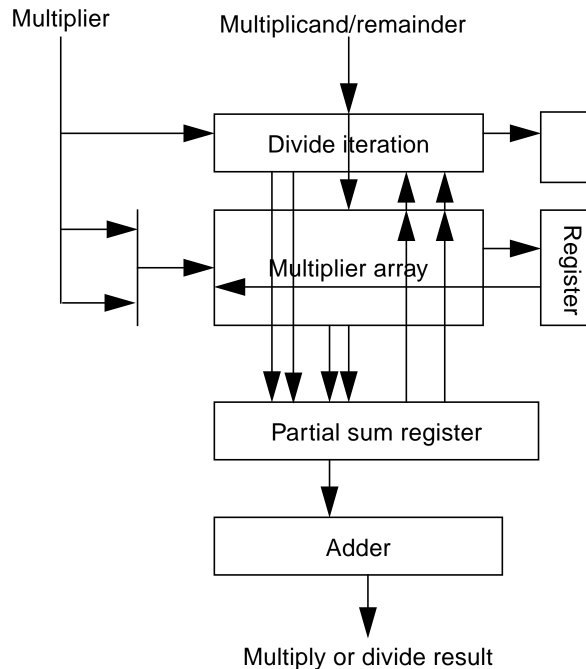


FIGURE 4-4 Multiply and Divide (MDIV) Section of the FPU

4.5 IEEE 754 Compliance

The FPU conforms to IEEE-754 compatibility, which requires support for *gradual underflow* that allows the mantissa to become unnormalized, leading to smaller errors in handing very small numbers. The FPU implements this gradual underflow exactly as defined in the IEEE specification.

4.5.1 Deviation From IEEE 754 Specification

For differences from IEEE 754, see *The Java Virtual Machine Specification* by Tim Lindholm and Frank Yellin.

These differences include providing special case outputs when an input is a NAN. Also, *The Java Virtual Machine Specification* allows only one type of conceptual NAN value compared to the IEEE specification, which allows several variations of NAN.

4.5.2 NaN Operations

The core's NaN operation applies to any two input operations—except compares, which have their own rules for NaNs.

For example, in a two-input operation—

```
out = a op b; /* op = {+,-,*,/,%} */
```

—TABLE 4-1 shows the results.

TABLE 4-1 Example Results of a NaN Two-Input Operation

out	A	B	Description
AnanP	Anan	X	Copies ANan to output (with positive sign)
BnanP	Not_nan	Bnan	Copies BNan to output (with positive sign)
INaN	Invalid combination		Produces the Inan result (positive always)

Where the single- and double-precision values of INaN for TABLE 4-1 are:

INaN = 7fff 0000	For single-precision operations
INaN = 7fff e000 0000 0000	For double-precision operations

Invalid combinations include some of the following for each opcode, `A_operand` and `B_operand`:

- `FADD`, `DADD`: {+INF,-INF}
- `FMUL`, `DMUL`: {0, +-INF}
- `FDIV`, `DDIV`, `FREM`, `DREM`: {+-INF,+-INF} {any number, 0}

The core treats all NaNs as quiet NANs: They do not signal an exception.

Here, the INaN (indefinite NaN) output acts as the signal for an invalid combination.

4.6 Opcodes

FPU opcodes perform a variety of operations, as described in this section.

4.6.1 FADD and FSUB Operations

Most single-precision FADD/FSUB operations require four cycles. Certain rare cases that involve DENORM, NaN, or INFINITY input operands may require up to seven cycles. Most DADD/DSUB operations require eight cycles plus extra cycles for loading and unloading, for a total of 12 cycles.

4.6.2 FMUL and DMUL Operations

Single precision requires four cycles. Double precision requires 15 cycles, except for cases where the multiply operation underflows into the DENORM range.

4.6.3 FDIV, FREM, DDIV, and DREM Operations

An FDIV operation typically requires 31 cycles.

A DDIV operation typically requires 61 cycles.

An FREM/DREM operation typically requires approximately x number of cycles, where x equals to DIVIDEND EXPONENT minus DIVISOR EXPONENT. The number could be as high as 2,000 cycles for DREM and is data dependent. To support applications with very high interrupt rates, the DREM opcode traps and is emulated by software if the PSR.DRT bit is set.

4.6.4 FConvert Operations

Fconvert converts a single-precision or double-precision floating-point value to a 32-64 INT value, or vice versa.

On an overflow, the maximum positive or negative number is provided. These operations require a variable number of cycles.

4.7 Power Management

When the FPU is in an idle state, such as when the `fpop` is not a valid FPU opcode, it assumes a lower power mode by preventing unnecessary toggling of logic (thrashing) when no calculation is necessary. In this mode, all states remain unchanged to ensure that the large combinatorial data paths remain quiet.

4.8 Interface

The interface to the FPU consists of two 32-bit input buses (`fpain`<31:0>, `fpbin`<31:0>), as shown in FIGURE 4-5. These inputs are directed to one or more of the FPU internal input registers based upon the type of `fpop` operation.

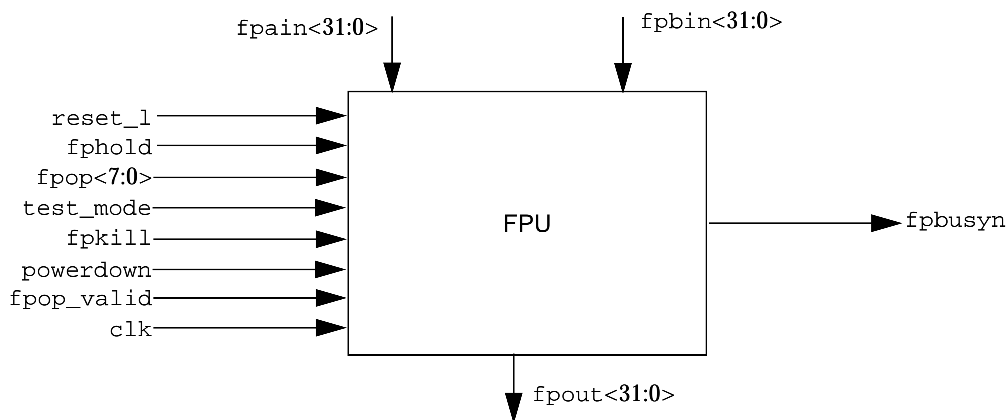


FIGURE 4-5 Floating Point Interface

4.8.1 Input Operations

Input loading of the FPU is through the following:

- Two 32-bit input buses, `fpain`<31:0> and `fpbin`<31:0>
- An 8-bit `fpop`<7:0> opcode command bus

An input transaction begins when the core detects a valid FPU opcode on the fpop input. In the next cycle (cycle 0) and optionally the following cycle (cycle 1), input data are transferred over these two buses.

TABLE 4-2 lists the data type and the number of operands for the FPU input transactions, and FIGURE 4-6 illustrates FPU timing.

TABLE 4-2 FPU Input Cycle Transactions

Operand Number and Type	Cycles to Load Input	Cycle 0: FPAIN, FPBIN	Cycle 1: FPAIN, FPBIN	cyc0_type
Two doubles	2	a.d.m b.d.m	a.d.l b.d.l	0
Two singles	1	a.f b.f	None	1
One long or double	1	a.x.m a.x.l	None	2 3
One integer or single	1	a.x 0 .m =MS word	None .l = LS word	4 5

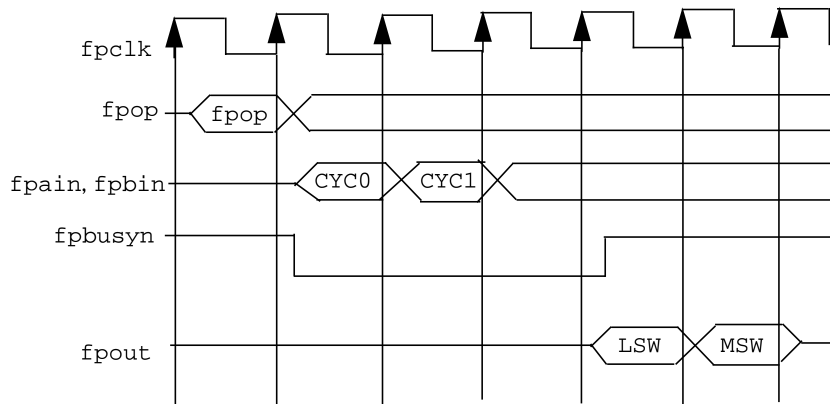


FIGURE 4-6 FPU Timing

For example, a DSUB operation (two double-precision operands) transfers the MS words A_operand and B_operand in cycle 0. fpop is valid one cycle before cycle 0. Since this input transaction requires two cycles, the LS word of the A and B operands is transferred in the following cycle (cycle 1). In single-cycle input transactions, the cycle 1 transaction is absent.

The B_operand is the second operand in the equation. Therefore, DSUB performs:

```
R.d = A.d - B.d.
```

Note – The double-precision value is indicated by `.d`.

4.8.2 Output Operations

The `fpbusyn` output signal is asserted (low) in the cycle following a valid `fpop` detection. It remains asserted until the first output transaction cycle, as shown in FIGURE 4-2 on page 44.

Once the FPU calculation begins, the FPU acts as a master by starting the output transaction as soon as the calculation is complete, as indicated by deasserting the `fpbusyn` signal.

The FPU ignores `fpop`s that are presented on:

- Cycles when `fpbusyn` is asserted (asserted low)
- The cycle that follows the deassertion of `fpbusyn`
- The first output transaction cycle of a two-cycle output (double or long)

The output transaction consists of one or two cycles (if the result is long or double) over the 32-bit `fpout`<31:0> bus:

- The first cycle transfers the `LS` word.
- The second cycle (if needed) transfers the `MS` word.

`fpkill`

The `fpkill` input signal terminates any FPU transaction on the next cycle and returns the FPU to the low-power idle state.

`fphold`

When asserted by the IU, the `fphold` input signal freezes the FPU operation at its current cycle and causes that cycle to repeat. Completion of the cycle occurs only when the IU deasserts `fphold` at the end.

The FPU is sensitive to `fphold` at all times; only `fpkill` overrides the hold function.

`fpop_valid`

The `fpop_valid` signal indicates that the incoming `fpop` byte is valid. Here, the `fpop` must also be a valid floating point operation for the FPU to proceed.

4.8.3 Data Forwarding

The FPU cannot forward data. Thus, an FPU operation (always writes to top-of-stack), followed immediately by another operation (ALU or FPU) that has a source as the top-of-stack, must stall the pipeline for one additional cycle.

4.9 Critical Paths

There are two primary critical paths in the FPU, both located in the ALU section.

The first critical path requires taking the incoming right shift amount from the SA register, performing the 32-bit wide shift and then performing a 32-bit add. The overflow (OVF) from this add may then be selected to control a mux, which picks the next microcode word into the microword register.

The second path takes a registered mantissa value and performs a 32-bit leading zero detection (LZD). Its output is then added to another 12-bit value. FIGURE 4-7 shows these two critical paths.

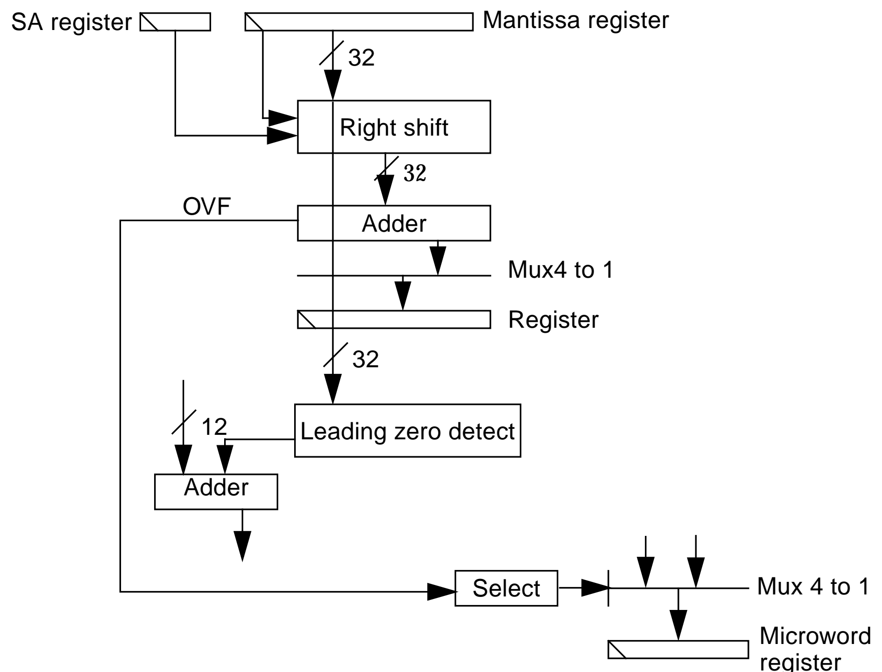


FIGURE 4-7 Critical Paths in the FPU

4.10 Signals

TABLE 4-3 defines the FPU signals.

TABLE 4-3 FPU Signals

Signal	Type	Definition
`fpain`<31:0>	Input	32-bit loading channel
`fpbin`<31:0>	Input	32-bit loading channel
`fpout`<31:0>	Output	32-bit output channel
`fpop_valid`	Input	Valid `fpop` input
`fpop`<7:0>	Input	8-bit floating-point operation command
`fpkill`	Input	Termination of the `fpop` and return of the FPU to idle
`fpbusyn`	Output	An active FPU operation
`reset_l`	Input	A reset of the FPU
`powerdown`	Input	A shutdown of the FPU ROMs
`fphold`	Input	A freeze of an FPU operation

CHAPTER **5**

Data Cache Unit (DCU)

The Data Cache Unit (DCU) arbitrates between requests to the data cache, which can come from the Stack Manager Unit (SMU) or the pipeline.

This chapter contains the following sections:

- *Dependencies* on page 55
- *Data Cache* on page 55
- *Functionalities* on page 57
- *Pipeline* on page 58
- *Cache Transactions* on page 64
- *Interface Signals* on page 66
- *Power Management* on page 68
- *Critical Timing Path* on page 68

The DCU interacts with the SMU, the Integer Unit (IU), the Instruction Cache Unit (ICU), and the Bus Interface Unit (BIU). FIGURE 5-1 shows the interaction among these units.

Note – Data cache requests from the pipeline use bit 30 to determine endianness and bits 29:28 to determine noncacheability. The DCU assumes data cache requests from the SMU to be cacheable and big endian. For details, see *Endianness and Cacheability* on page 111.

Note – You must not write and read the status RAM in the same cycle.

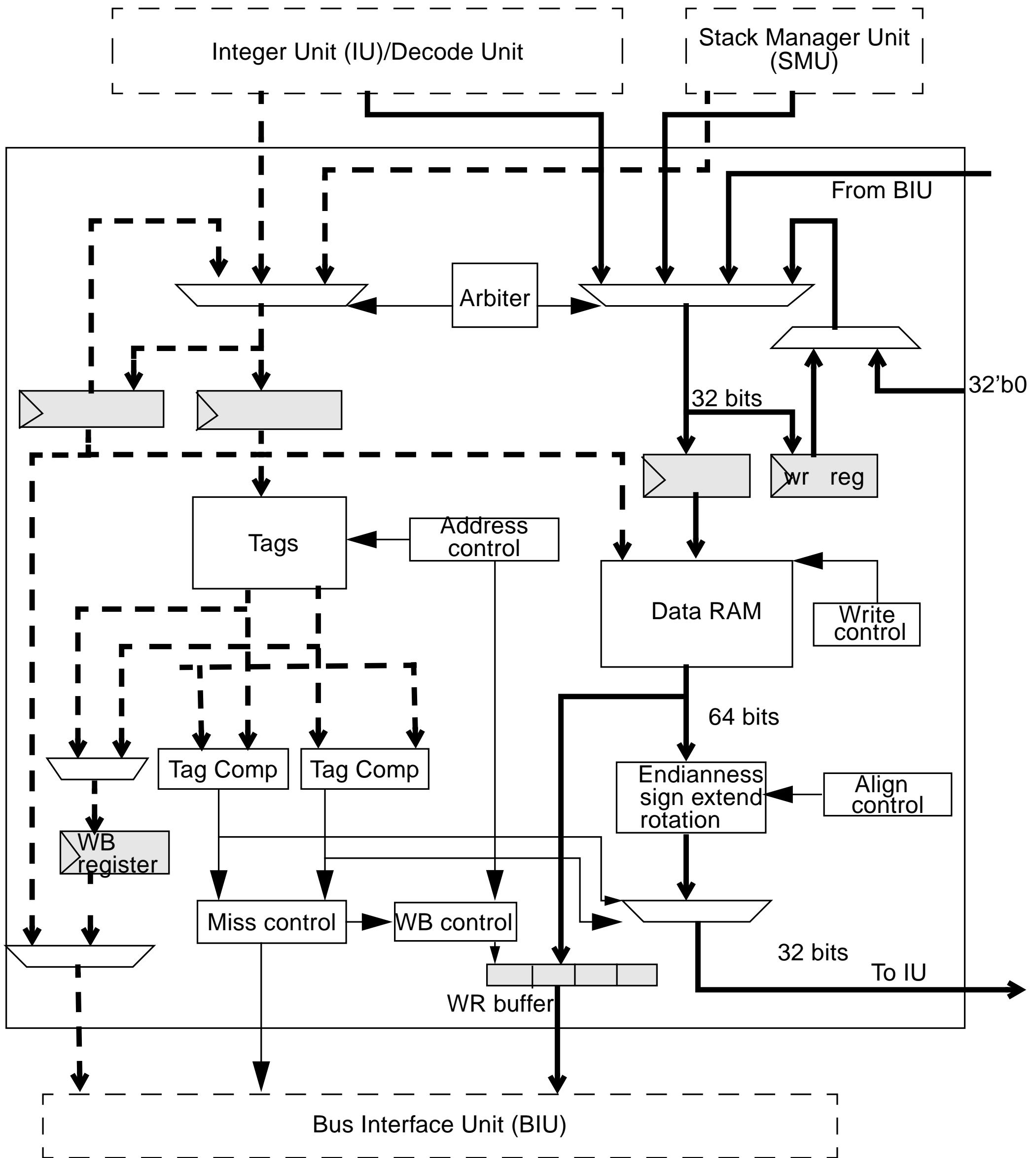


FIGURE 5-1 Data Control Unit (DCU) Interaction with Other Units

5.1 Dependencies

DCU actions depend upon the following:

- **Cache hits** — In cache hits, the DCU returns data to the IU or SMU in one cycle for loads. It also takes one cycle in case of write hits.
- **Cache misses** — In cache misses, the DCU stalls the pipeline until the requested data are available from main memory.
- **Noncacheable loads and stores** — The IU or SMU bypasses the DCU when a load or store is noncacheable and sends a request to the BIU.
- **Nonaligned loads and stores** — Nonaligned loads and stores trap in the software.

The DCU also generates requests to the bus in cache misses and noncacheable loads and stores.

5.2 Data Cache

The data cache is a two-way set associative, write-back, write-allocate, 16-byte line cache. The cache size is configurable to 0-Kbyte, 1-Kbyte, 2-Kbyte, 4-Kbyte, 8-Kbyte, and 16-Kbyte sizes. The default is 16 Kbytes. Each line has a cache tag store entry associated with it. On a cache miss, the DCU writes 16 bytes of data into the cache from main memory.

FIGURE 5-2 shows a 16-Kbyte data cache.

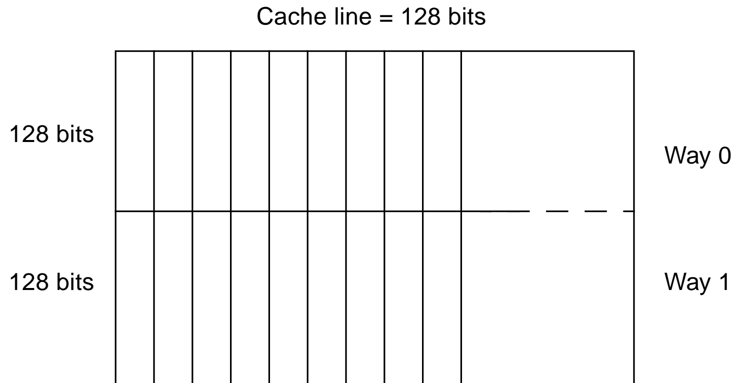


FIGURE 5-2 16-Kbyte Data Cache (512x256)

The cache tags contain a 19-bit address tag field, one valid bit, and one dirty bit. Each set also has one Least Recently Used (LRU) bit, which indicates the line in that set that was updated least recently. To support multiple cache sizes, the width of tag fields can change easily.

The DCU data RAMs consist of two RAM banks, each with 2,048 entries of 32 bits each (for a 16-Kbyte data cache). The lines are organized such that alternate entries in the same cache line are mapped to different RAM blocks to allow simultaneous access to the following:

- Two consecutive words in a single line, which is useful for performing entire line fills or evictions
- The same word in both ways of a set, which is useful for lookups during which the required data may be present in either way of the set

If the PSR bit for cache-enable is not set, the DCU handles loads and stores as if they were noncacheable instructions.

A single 16-byte writeback buffer contains dirty cache lines that must be replaced. In a single cycle, the cache can provide a maximum of 4 bytes on a read or a write into the cache.

Note – When the core is in reset state, the RAMs are in an unknown state. Reset software must use diagnostic accesses to initialize the RAMs explicitly.

The DCU can also perform diagnostic reads and writes on the cache.

5.3 Functionalities

The DCU does the following:

- Arbitrates data cache requests from the SMU and the pipeline
- Generates address, data, and control signals for the data and tag RAMs
- Reorders the data RAM output to provide the right data in a cache hit
- Provides datapath and control logic for processing noncacheable requests
- Provides the datapath and datapath control functions for cache misses

5.3.1 Arbiter

The arbiter does the following:

- Arbitrates data cache requests from the pipeline and the SMU
- Selects address and data inputs for the tag and data RAMs

5.3.2 Address Control

Address control updates the following:

- The address field of the tags while writing to the caches
- The `LRU` bit during cache access, invalidating the entry in case of flushes
- The dirty bit while writing to the caches

5.3.3 Aligner Control

Aligner control provides the following:

- Write-enable signals to the data RAM
- Control signals to align data from the data RAM during cache hits

5.3.4 Miss Control

Miss control interfaces with the writeback control and the read buffer control and does the following:

- Handles all cache misses
- Generates a stall signal to the pipeline as well as the bus requests for cache line fills
- Provides the handshaking signals for these transactions

5.3.5 Writeback Control

Writeback control determines which line to replace, depending on the `LRU` bit, and verifies if the dirty bit is set. If so, it moves that line into the write buffer. Once the cache fill transaction started by miss control is complete, writeback control starts a writeback cycle.

5.3.6 Data Cache Datapath

The data cache datapath consists of all the address and datapath logic, as shown in FIGURE 5-1 on page 54. The data RAM and tag RAMs are not included in the datapath.

The datapath does the following:

- Provides appropriate address and data to the RAMs
- Postprocesses data before sending them to the pipeline or the register file
- Sends appropriate address and data on the memory buses for cache fill and writeback transactions

The datapath also contains a single-entry, 16-byte size store buffer, thus accommodating a cache line that is being replaced.

5.4 Pipeline

This section describes the cache transactions in the pipeline.

5.4.1 Cache Reads

A cache read takes three stages, as illustrated in FIGURE 5-3.

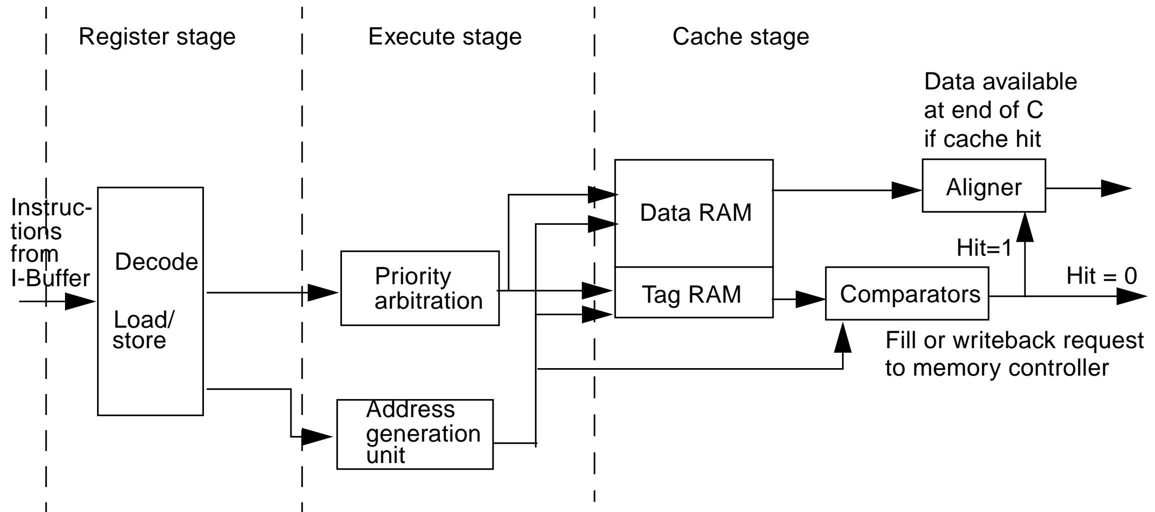


FIGURE 5-3 A Pipeline Read Transaction

- **Register (R) stage**, during which the instruction is decoded to determine if it is a load or store.
- **Execute (E) stage**, during which the address is computed using a separate 32-bit adder for loads and stores. In parallel, a request is sent to the data cache. Depending on the priority, the address and request are ready before the end of the E stage.
- **Cache (C) stage**, during which the tags and data RAMs are accessed with the address in this clock. Only the tags are accessed in this stage for stores.

Tags are compared. If there is a cache hit, the load's data are available at the end of C stage after going through aligner muxes to obtain the requested 32 bits of data. If there is a cache miss, a request is made to the bus. In case of stores, data are written into the cache in the C+1 stage on a cache hit. In case of a miss, a request is made to the bus in the background.

5.4.2 Cache Writes

A cache write takes three stages, as illustrated in FIGURE 5-4.

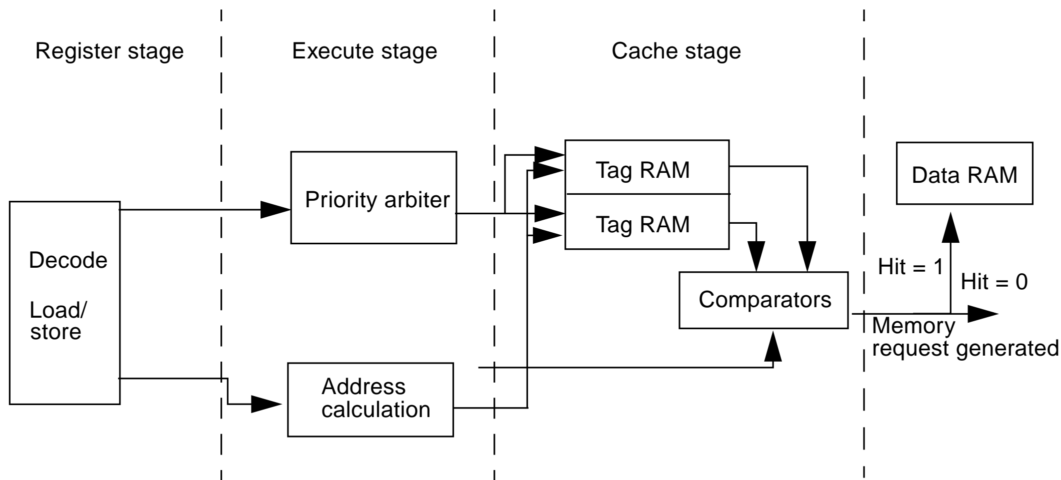


FIGURE 5-4 A Cache Write Transaction

The data cache is a write-allocate-writeback cache. When a write miss occurs, the DCU fetches a line from memory and writes it into the D-Cache. However, since the SMU writes sequential data, if the write to the end of a cache line misses the D-Cache, the DCU writes the missed data directly into the D-Cache instead of reading from memory. This protocol saves memory cycles and context switch time as well as improves the interrupt latency.

The DCU does not stall the pipeline but cannot accept additional requests until the store transaction is complete. The IU or SMU can dispatch back-to-back stores for the data cache. Stores can take two or more cycles to complete, depending on the cache hit or miss, replacing dirty lines, and so on. The IU assumes this to be a one-cycle operation and continues the execution flow.

Note – There is a bubble between a store and an immediate load.

5.4.3 Cache Fills

On a cache miss, a cache fill request is sent to the memory controller at the end of the C stage. A cache fill transaction starts whenever there is a cache miss and there are no outstanding bus requests. It waits until it receives a `pj_ack` from the memory controller, then writes the data from the memory bus onto the cache.

FIGURE 5-5 is a state diagram for the cache fill transaction.

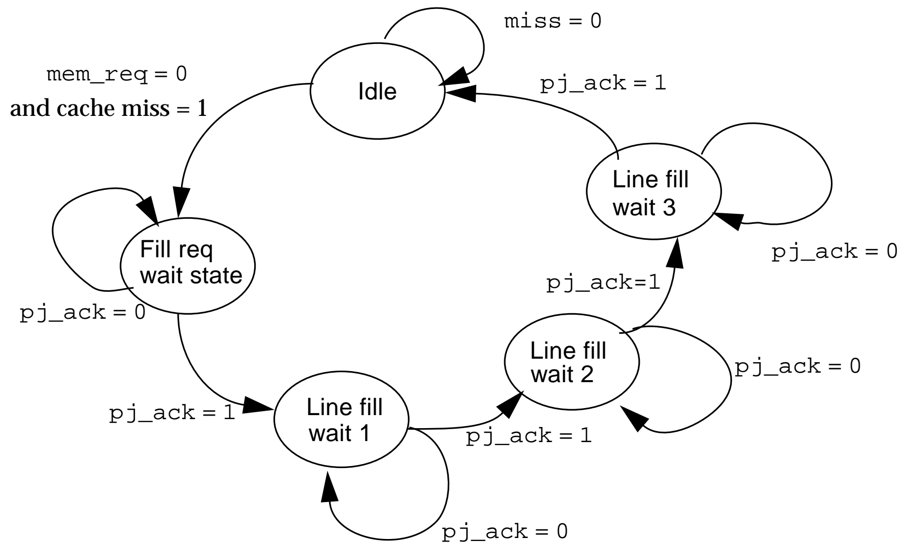


FIGURE 5-5 A Cache Fill Transaction

The transaction also forwards the right data to the IU or SMU after sign extension of data, if required. On the second `pj_ack`, the second part of the line is written into the cache. The same process is repeated for the remaining two `ack`s.

If an error `ack` is received during cache fill, the state memory controller is immediately reset and goes back to the idle state. FIGURE 5-5 does not show that transaction.

TABLE 5-1 lists the order of filling.

TABLE 5-1 Data Cache Fill Ordering

Requested Data Word	Order of Fill Data Returned
0	0,1,2,3
1	1,2,3,0
2	2,3,0,1
3	3,0,1,2

5.4.4 Writebacks

FIGURE 5-6 is a state diagram for the writeback transaction.

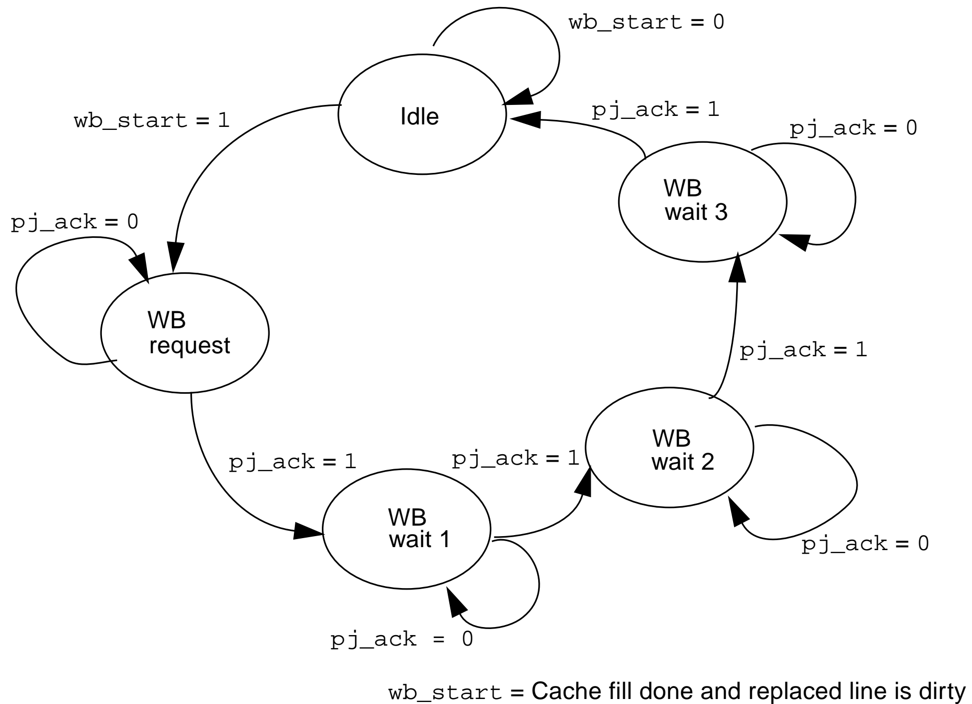


FIGURE 5-6 A Writeback Transaction

A writeback (WB) transaction takes place whenever a cache miss occurs and there is a dirty line to be replaced. In case of cacheable loads and stores, a writeback transaction starts after a cache line fill cycle is complete.

The WB transaction takes place in the background. Thus, the pipeline stall is deactivated once the cache fill transaction is complete. In case there is another miss while the WB transaction is in progress, the cache fill transaction waits for the WB transaction to finish before it starts its process.

During this transaction, if an error ack is received, the state machine is reset to its idle state. To simplify the logic, if there is a request for the line being written back during the WB cycle, the data are not bypassed. Instead, the state machine signals a cache miss.

5.4.5 Noncacheable (NC) Loads

Noncacheable loads force a cache miss, and an NC request is made to the bus. Once the data are available, they are bypassed to the pipeline, as shown in FIGURE 5-7.

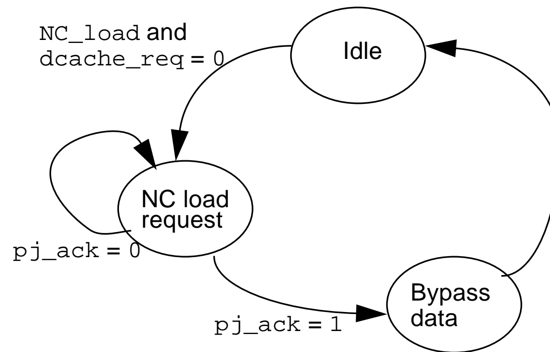


FIGURE 5-7 A Noncacheable Load

5.4.6 Noncacheable (NC) Stores

Noncacheable stores also force a cache miss and an NC request is made to the bus. The data are written into the writeback buffer and the pipeline is not stalled. The transaction is very similar to the WB transaction once the store data are written into the write buffer.

If the write buffer contains a cache line, the noncacheable store occurs after the cached store is retired. This step maintains strong write ordering.

FIGURE 5-8 illustrates a noncacheable store.

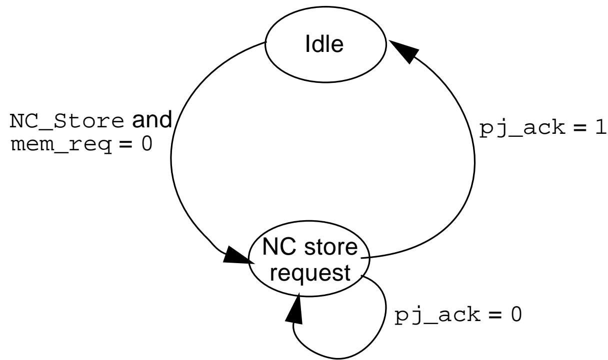


FIGURE 5-8 A Noncacheable Store

5.5 Cache Transactions

This section defines the various cache transactions.

5.5.1 Arbitration of Requests

The data cache receives requests from both the SMU and the pipeline. Usually, the pipeline has a higher priority than the SMU; however, in rare cases where the SMU holds the pipe, it has a priority over the IU pipe.

The requests from the SMU are single 32-bit loads or stores.

5.5.2 Replacements

The core uses the pseudo Least Recently Used (LRU) replacement policy. There is one bit for every two lines; the bit is updated whenever data are written into the cache. This policy does not penalize read latency. An LRU update occurs only on a cache fill or a write hit in the cache.

Once a miss is detected, the `LRU` bit determines the line to be replaced. If the dirty bit of that line is set, the line is moved into the writeback buffer. Cache miss is detected in the C stage. Moving the replaced line into the write buffer takes two additional cycles. Once the cache fill transaction is complete, a writeback request is sent to the memory controller. Writebacks take place in the background.

5.5.3 Cache Compare Flushing

In a cache compare flushing operation, the corresponding line tags are accessed and compared to determine a hit. If it is a hit and the dirty bit is not set, the tag's valid bit is turned off. If the dirty bit is set, a writeback transaction starts. The `LRU` bit is also updated.

This protocol accommodates self-modifying code.

5.5.4 Cache Indexed Flushing

Cache indexed flushing is similar to cache compare flushing, except that the tags are not compared. The cache line to be invalidated is written back to memory if it is dirty. The valid bit is reset after moving the dirty line into memory.

The cache indexed flush instruction has the same address semantics as a diagnostic read or write.

5.5.5 Cache Invalidate Flushing

In a cache invalidation, the corresponding line tags are accessed and compared to determine a cache hit. If it is a hit, the line is invalidated. There is no writeback transaction. On a cache miss, there is no state change.

5.5.6 Zeroing of Cache Lines

When the core executes the `zero_line` instruction, the DCU receives a signal from the IU to zero out a particular cache line. `zero_line` takes five cycles to complete—one to determine hit or miss and four to write. The IU assumes this to be a one-cycle operation and continues the execution flow unless there is an instruction following `zero_line` that uses the data cache. This instruction is then stalled until `zero_line` completes.

If the core executes `zero_line` when the data cache is absent or turned off, it generates an exception (`ZeroLineEmulationTrap`).

The tags are checked to see if the cache line is a valid dirty line. If not, the cache line is filled with zeroes. If it is a dirty line, a writeback transaction is done and then the cache line is zeroed out. In case of cache line zeroing, it is a nonallocate store. The `LRU` bit is updated and the dirty bit is set.

5.5.7 Nonallocating Writes

The DCU supports nonallocating writes to handle the `nastore_word_index` instruction and processes them like cacheable stores on a cache hit. However, on a cache miss, the DCU converts `nastore_word_index` into a noncacheable store and sends it to memory.

The DCU also uses nonallocating writes for SMU requests to avoid fetching new lines into the cache, hence improving the performance of dribble stores.

5.5.8 Nonfetching Allocates

The DCU uses nonfetching allocates for SMU requests to write out data at the last word in a line. For such writes, it allocates the address for the line in the cache, evicting and writing back another line if necessary, but does *not* fetch the contents of the new line from memory; instead, it anticipates the SMU to overwrite that entire line.

5.5.9 Diagnostic Accesses

The DCU supports diagnostic access instructions, which directly read or write the contents of the tag and data RAMs. The core reads or writes the DCU RAMs for such instructions in the E stage. For details on diagnostic access instructions, see *picoJava-II Programmer's Reference Manual*.

5.6 Interface Signals

TABLE 5-2 through TABLE 5-5 define the DCU interfaces.

TABLE 5-2 DCU Interface with the Bus Interface Unit (BIU)

Signal	Type	Description
dcu_biu_data<31:0>	Output	Data bus for writeback/NC store transactions
biu_data<31:0>	Input	Data bus for cache fill/NC load transactions
dcu_addr_out<31:0>	Output	Address bus for memory transactions
dcu_size<1:0>	Output	Size of data requested

TABLE 5-2 DCU Interface with the Bus Interface Unit (BIU) *(Continued)*

Signal	Type	Description
dcu_type<3:0>	Output	Type of transaction
dcu_req	Output	Transaction valid/request
iu_psr_dce	Input	Enable the data cache
biu_dcu_ack<1:0>	Input	Acknowledgment that data are available on data bus

TABLE 5-3 DCU Interface with the Stack Manager Unit (SMU)

Signal	Type	Description
smu_data<31:0>	Input	The SMU data bus for stack spills
smu_addr<31:0>	Input	The SMU address used for spills and fills
smu_ld	Input	A load request from the SMU
smu_st	Input	A store request from the SMU
smu_data_vld	Output	An acknowledgment from the DCU that valid data are available on the dcu_data bus
smu_stall	Output	A signal that stalls the SMU because of a busy or miss state of the D-Cache
smu_prty	Input	A signal that indicates the SMU priority over the IU in the DCU

TABLE 5-4 DCU Interface with the Integer Unit (IU)

Signal	Type	Description
iu_addr_e<31:0>	Input	The address that corresponds to the IU request
iu_data_e<31:0>	Input	Data bus used for writes to the DCU
iu_inst_e<7:0>	Input	Load and store instruction from the IU
iu_stall	Output	Pipeline stall
kill_inst_e	Input	Termination of instruction execution in the E stage
iu_trap_c	Input	Termination of outstanding DCU operations because of a trap
iu_diag_e	Input	Diagnostic read or write commands to the DCU
iu_zero_e	Input	A zero_line instruction to the DCU

TABLE 5-5 Miscellaneous DCU Interfaces

Signal	Type	Description
`clk`	Input	Clock
`dcu_si`	Input	Scan-in port
`dcu_so`	Output	Scan-out port
`pcsu_powerdown`	Input	Powerdown
`dcu_data`	Output	Data bus used for data cache reads to the IU, the SMU, and the external unit

5.7 Power Management

The data cache receives a `power_down` signal to turn off the cache when not in use. This signal turns off the address decoders and the sense amplifiers. Before the caches are turned off, all outstanding transactions are completed.

The data cache is also turned off during the idle cycles when the cache is not accessed. It is turned off during the dead cycles of a cache fill transaction.

5.8 Critical Timing Path

The most critical path in the DCU is the tag access-tag comparison path, shown in FIGURE 5-9. This path is for a 16-Kbyte, two-way set associative cache configuration.

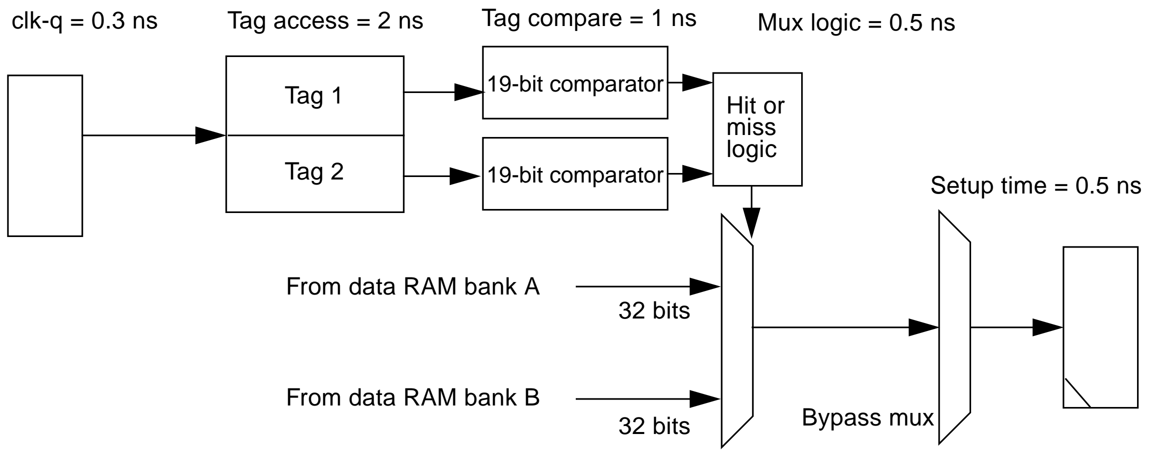


FIGURE 5-9 Critical Timing Path in the DCU

CHAPTER **6**

Stack Manager Unit (SMU)

The Stack Manager Unit (SMU) handles overflow and underflow conditions of the stack cache. For details on how the core uses the stack cache, see Chapter 3, “Memory System and Caches,” in *picoJava-II Programmer’s Reference Manual*.

This chapter contains the following sections:

- *Functionalities* on page 73
- *Dribbling Operations* on page 74
- *Pipeline* on page 77
- *Interface Signals* on page 77

The SMU stalls the pipeline upon detecting an overflow or underflow. FIGURE 6-1 shows its interaction with the Integer Unit (IU) and the Data Cache Unit (DCU).

Note – See *Dribbling Operations* on page 74 for the algorithm that defines an SMU operation.

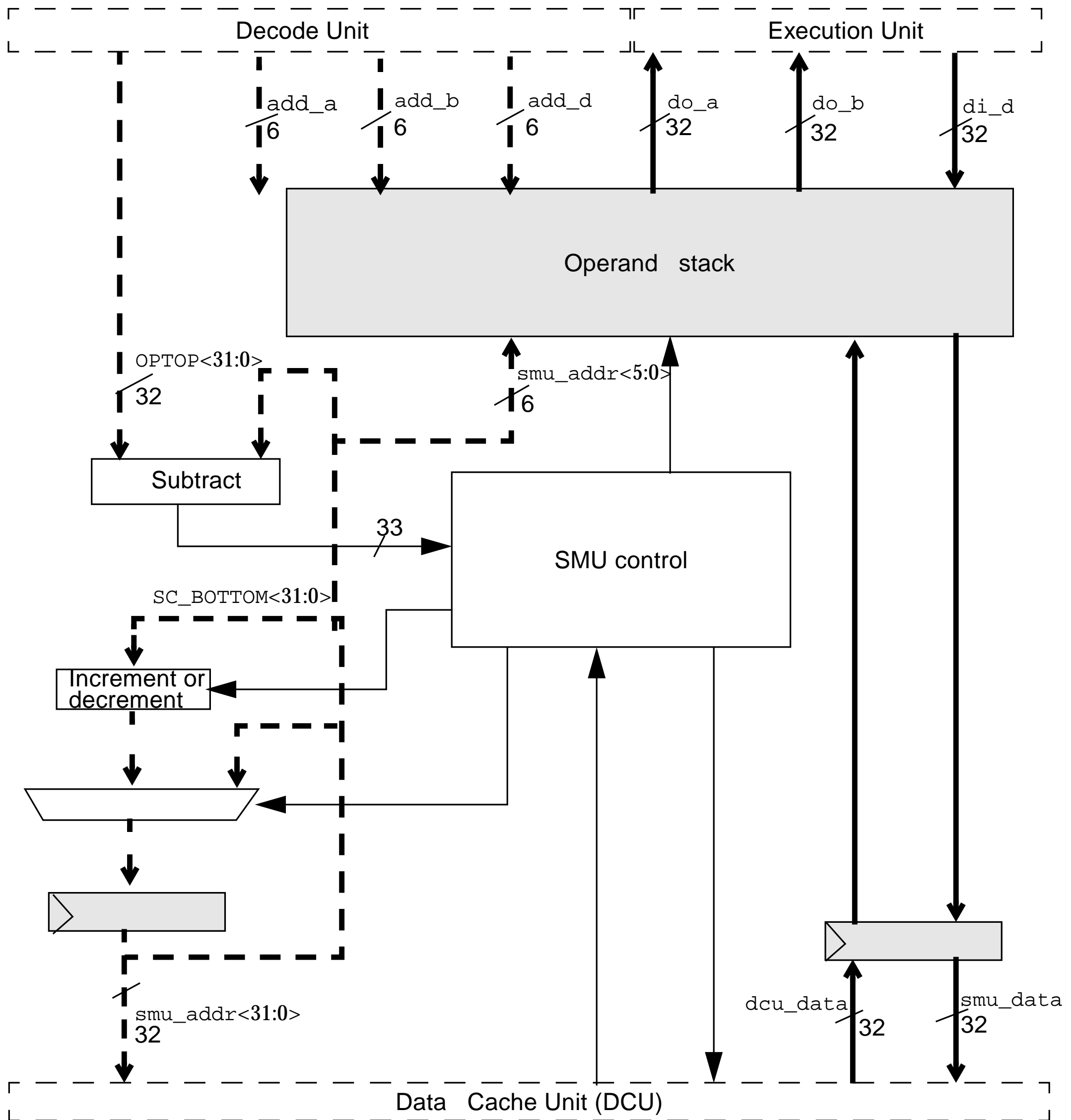


FIGURE 6-1 Stack Manager Unit (SMU) Interaction with Other Units

6.1 Functionalities

The SMU performs the following functions:

- Moves data in and out of the stack into memory in an overflow or underflow in the stack cache. It also provides the control signals for one read port and one write port of the stack cache, which are used for dribbling only.
- Handles spills and fills of the stack cache by speculatively dribbling the data in and out of the stack cache from and to the data cache.
- Generates a pipeline stall signal to stall the pipe when a stack cache overflow or underflow condition is detected.
- Keeps track of requests sent to the data cache. A single request to the data cache consists of a 32-bit consecutive load or store request.
- Handles stack cache write misses of the IU.

The stack cache is a 64-entry, three-read, two-write port register file. It caches the top few entries of the operand stack. Two read ports and one write port fetch and store operands to and from the IU. Exclusive read and write ports perform background dribbling of data to and from the data cache, depending on whether the condition is an overflow or underflow.

Note the following points about the stack cache:

- All memory accesses generated as offsets from `OPTOP`, `VARS`, or `FRAME` are stack accesses. The IU first checks whether the address exists in the stack cache. All other memory accesses go directly to the data cache.

  The condition for deciding whether an address is a hit in the stack cache is the address in the interval (`OPTOP` + 4, `SC_BOTTOM` inclusive).

  Memory accesses generated as offsets from `OPTOP` must *always* hit the stack cache. The locations that correspond to the four top words in the stack and the two empty words above the top of the stack are present in the stack cache.

  Accesses to these locations are assumed to hit directly in the stack cache because `OPTOP` offsets are always between 4 and –1, inclusive, and the SMU *always* ensures that the stack cache has between 6 and 60 entries at the top of the stack.

- The SMU must be on (`PSR.DRE` = 1).

  This setting ensures correct program behavior. If the SMU is disabled (`PSR.DRE`=0), it does not hold the pipe, the requirement of 6–60 entries in the stack cache is not met, and the stack cache underflow or overflow operation is not handled correctly. Therefore, operations may use erroneous data.

Note – The SMU is *not* a performance feature that can be switched off unless the program understands the stack cache behavior. The stack cache operation is not transparent to the program if the SMU is off.

- An address that hits the stack cache is stored at the entry in the stack cache that corresponds to the 6 least significant bits of the address.
- Do not set `fill_mark` to 0 because this setting causes incorrect program behavior.
- The word pointed to by `SC_BOTTOM` is assumed to be present in the stack cache.
- The SMU datapath keeps track of the memory address for loading or storing data from and to the stack cache. It also moves data from the stack cache to memory, and vice versa. During initialization, a 32-bit address is stored in `SC_BOTTOM`. From then on, only the SMU manipulates the `SC_BOTTOM` register and software must *not* write to it. The SMU increments `SC_BOTTOM` on a fill and decrements it on a spill.

6.2 Dribbling Operations

The stack cache caches part of the stack. See FIGURE 6-2.

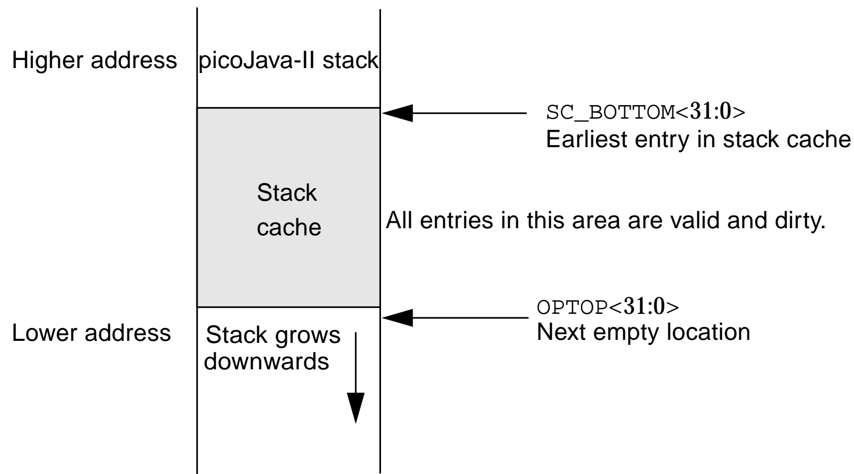


FIGURE 6-2 Stack Cache

OPTOP is on the stack cache. The entries between SC_BOTTOM and OPTOP are dirty data and must be saved in memory in a stack overflow.

6.2.1 Spills and Fills

During a cycle, the SMU compares the number of entries against a high watermark and a low watermark:

- If the result exceeds the high watermark, the SMU starts a spill transaction and sends a write request to the data cache, along with the address and data, and receives an acknowledgment in return.

  The SMU sends another request if a spill condition continues to exist.

- If the result is less than the low watermark, the SMU starts a fill transaction and sends a read request to the data cache, along with the address, and receives an acknowledgment in return.

  The SMU then writes the data on the DCU bus into the stack cache. It sends another request if a fill condition continues to exist.

Note – The low watermark must be at least eight entries and less than the high watermark. Both watermarks must *not* match.

Spill and fill transactions take place in the background while the pipeline continues to execute instructions.

In case of a drastic jump in the OPTOP value so that no entries are on the stack cache, the SMU stalls the pipeline and ensures that adequate free space is available on the stack cache.

Note – In a write of a new OPTOP value (OPTOP') to the OPTOP register, if OPTOP < OPTOP' <= OPTOP + 64, the contents of the stack cache between OPTOP and OPTOP' may not be saved.

6.2.2 Stack Overflow

A stack overflow occurs if OPTOP < SC_BOTTOM – 60, causing OPTOP to fall off the stack. The SMU then stalls the pipeline and activates a state machine. The state machine sends a series of write requests to the data cache.

Once the dirty entries have been written into the data cache, the stack cache verifies that the six topmost entries of the stack (based on the new OPTOP location) are present in the stack cache, reading them from the data cache or from memory if they are not already present.

When there are six entries in the stack cache, the SMU deasserts the stall; execution of instructions continues.

Note – The stack overflow condition also occurs if, after `write_optop` or `priv_update_optop` writes to OPTOP, the new OPTOP exceeds SC_BOTTOM. Overflows do not occur from returns.

6.2.3 Stack Underflow

A stack underflow occurs if OPTOP exceeds SC_BOTTOM (for example, upon a return from a large method), causing OPTOP to fall off the top of stack cache. The SMU then stalls the pipeline and activates a state machine. The state machine sets SC_BOTTOM to the new OPTOP value and sends read requests until there are six entries in the stack cache.

Once the SMU has written the data into the stack cache, it deasserts the stall. Any entries beyond OPTOP are not used and therefore not saved.

The SMU can activate an underflow only in response to changes in OPTOP caused by the return instructions in TABLE 6-1.

TABLE 6-1 OPTOP Instructions That Trigger Underflows

areturn	freturn	lreturn	return	return1
dreturn	ireturn	priv_ret_from_trap	return0	return2

Note – If you use the `priv_ret_from_trap` instruction to facilitate context switching, software must ensure that the stack cache contents in the original context are written back to the data cache.

6.2.4 Stack Cache Write Misses

When the IU has a stack cache miss on a write, it issues to the SMU the store address and data. The SMU gives this request higher priority over spills and fills and holds the pipe until the store has been issued to the data cache.

On a stack cache miss request, the SMU clears its pipe, issues the miss address, and stores the miss data to the data cache.

6.3 Pipeline

TABLE 6-2 details the computations that take place in the pipeline for spills and fills.

TABLE 6-2 Pipeline Actions for Spills and Fills

E Stage	C Stage	W Stage
Computes a new OPTOP.	Determines overflow or underflow.	Sends a data cache read request in a fill or underflow and a data cache write request in a spill or overflow.
	Determines spill or fill.	Determines a new valid pointer.

6.4 Interface Signals

TABLE 6-3 and TABLE 6-4 define the SMU interfaces.

TABLE 6-3 SMU Interface with the IU

Signal	Type	Description
smu_rf_addr<5:0>	Output	The stack cache address for reads and writes
iu_rf_dout<31:0>	Input	The spill data from the IU
smu_rf_din<31:0>	Output	The fill data from the SMU to the IU
smu_hold	Output	A stall of the pipeline data
iu_optop_in<31:0>	Input	iu_optop_c value at input of flop
iu_optop_int_we	Input	iu_optop_c write enable signal
iu_data_in<31:0>	Input	sc_bottom value from the IU for write_scbottom operations
iu_sbase_we	Input	Write enable signal for iu_data_in
smu_sbase<31:0>	Output	The updated SC_BOTTOM value
smu_sbase_we	Output	Write enable signal for smu_sbase
iu_pse_dre	Input	Dribbler enable bit from the PSR register
iu_int	Input	An indication of an interrupt to the SMU
smu_we	Input	Write enable signal for writes to the stack register file
low_mark<5:0>	Input	Low watermark value from the PSR register

TABLE 6-3 SMU Interface with the IU *(Continued)*

Signal	Type	Description
`high_mark<5:0>`	Input	High watermark value from the `PSR` register
`smu_data_vld`	Input	Valid DCU data
`smu_st_c`	Input	SMU store in C stage
`iu_smu_flush`	Input	A write to stack locations between addresses `sb` – 2 and `sb` + 2
`ret_optop_update`	Input	An update of the `OPTOP` because of return instructions.
`iu_smiss`	Input	An indication of an instruction store miss
`iu_data<31:0>`	Input	Instruction store miss data from the IU
`iu_address<31:0>`	Input	Instruction store miss address from the IU
`smu_na_st`	Output	A nonallocate store indication to the DCU
`dcu_data`	Input	Data from the D-Cache

TABLE 6-4 SMU Interface with the DCU

Signal	Type	Description
`smu_addr<31:0>`	Output	The address used for a data cache transaction
`smu_data<31:0>`	Output	The data for data cache writes
`smu_ld`	Output	A data cache load request during fills
smu_st	Output	A data cache store request during spills
`smu_stall`	Input	A stall for the next request due to a cache miss

CHAPTER 7

Bus Interface Unit (BIU)

The Bus Interface Unit (BIU) implements the picoJava-II interface to the external world. It generates requests to memory and I/O devices. However, the memory bus is a dedicated bus between the picoJava-II core and the memory controller. Requests to I/O devices go through the memory controller, which maintains an address map of the entire system. Hence, the core is the only master on the memory bus and does not have to arbitrate to use it.

This chapter contains the following sections:

- *Functionalities* on page 79
- *Arbitration* on page 80
- *Interfaces* on page 81
- *Power Management* on page 82
- *Interface Signals* on page 83

7.1 Functionalities

The BIU contains the arbitration logic for internal requests and performs the following tasks:

- Generates read and write requests to memory
- Provides acknowledgment to the instruction cache and data cache controllers so they can sink data
- Handles errors generated on the memory bus

FIGURE 7-1 is a block diagram of the BIU.

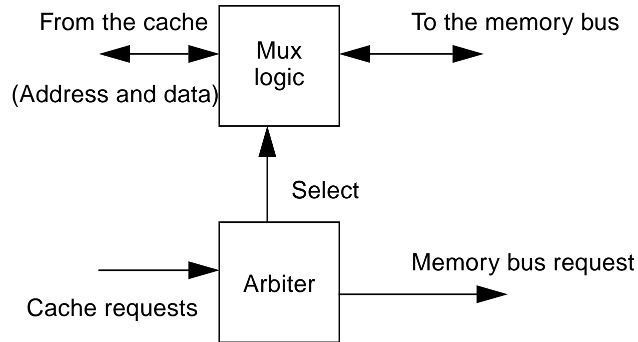


FIGURE 7-1 Bus Interface Unit (BIU)

7.2 Arbitration

FIGURE 7-2 shows the BIU arbitrator.

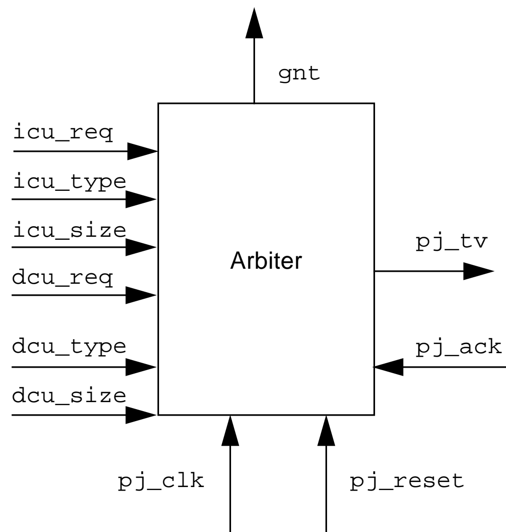


FIGURE 7-2 BIU Arbiter

The BIU arbitrates internal requests from the instruction cache and data cache and generates an external request on the bus. It handles dribble requests from the data cache as data cache requests.

The instruction cache and data cache controls ensure that only one instruction cache request and one data cache request can be active.

The arbitration algorithm is as follows:

- Data cache requests have a higher priority than instruction cache requests.
- *_type* and *_size* indicate to the arbiter the number of acknowledgments to expect for a transfer.
- On an external reset (`pj_reset`), the arbiter resets its state machine, clearing all pending requests, and becomes idle.
- In back-to-back requests, the arbiter can initiate the second request in the next clock after the first transfer is complete or after the last acknowledgment.
- An error acknowledgment on a cycle terminates a transfer.

7.3 Interfaces

The BIU interfaces with the instruction cache and data cache controllers to the external (memory) interface.

To request data from memory, the instruction cache or data cache controllers send a request, along with such information as the request address, data, and size, to the BIU. The BIU then arbitrates between the pending requests and initiates a transfer on the memory bus.

The BIU acknowledges read cycles to the requesting controller (instruction cache or data cache) so the DCU or ICU can latch the data. (See Chapter 9, *External Interface*, for descriptions on the various acknowledgments.) In write cycles, the acknowledgments act for flow control so the instruction cache and data cache controllers do not generate new requests for pending ones.

If an error acknowledgment occurs on the data transfer, the BIU terminates the current cycle and sends an error acknowledgment to the requesting controller, which then ignores the data and takes the appropriate action. The error acknowledgment can occur on any cycle. Any pending requests are initiated on the bus.

FIGURE 7-3 illustrates how the BIU interfaces with other units.

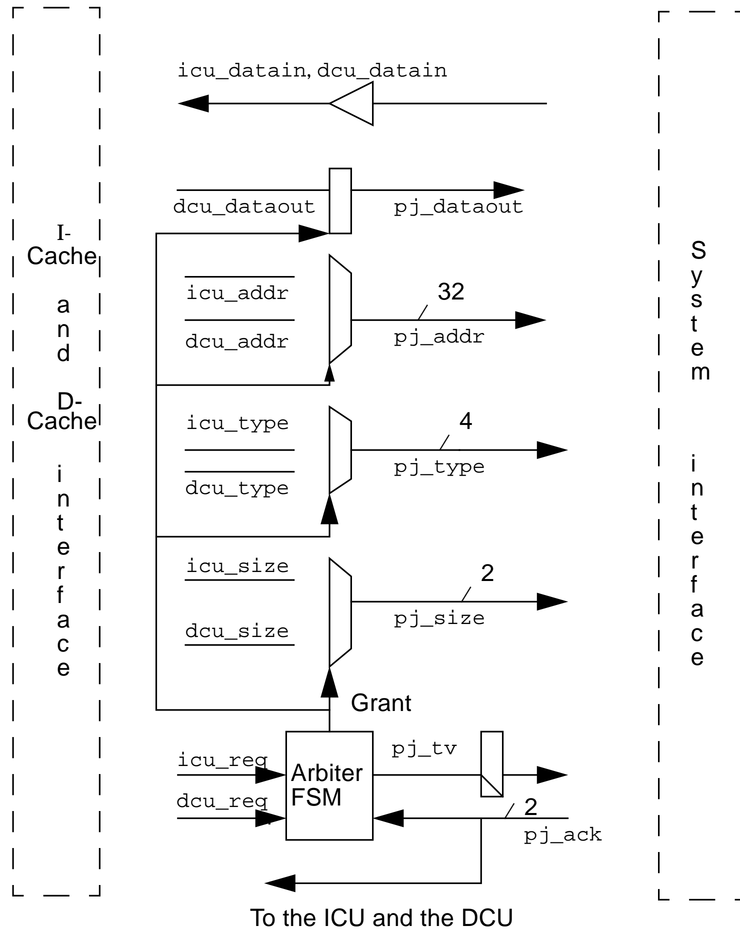


FIGURE 7-3 Bus Interface Unit (BIU) Interaction with Other Units

7.4 Power Management

Power management is inherently handled because the instruction cache and data cache controllers stop generating requests to the BIU when they are in powerdown mode.

7.5 Interface Signals

TABLE 7-1 and TABLE 7-2 define the BIU interface signals.

TABLE 7-1 BIU Interfaces with the Instruction Cache and the Data Cache

Signal	Type	Definition
icu_req	Input	A signal from the instruction cache that indicates a request
icu_addr <31:0>	Input	The physical address for an instruction cache request
icu_type<3:0>	Input	The type of an instruction cache transaction
icu_size<1:0>	Input	The size of an instruction cache transfer
biu_icu_ack<1:0>	Output	The bus acknowledgment for an instruction cache transfer
biu_data<31:0>	Output	Data delivered to the instruction cache or the data cache from external device
dcu_req	Input	A signal from the data cache that indicates a request
dcu_addr<31:0>	Input	The physical address for a data cache request
dcu_type<3:0>	Input	The type of a data cache transaction
dcu_size<1:0>	Input	The size of a data cache transfer
dcu_dataout<31:0>	Input	The data cache data to be written to the external device
biu_dcu_ack<1:0>	Output	The bus acknowledgment for the data cache transfer

TABLE 7-2 BIU External Interface Signals

Signal	Type	Description
pj_clk	Input	Clock
pj_reset	Input	Reset
pj_address<31:0>	Output	The address of a data request
pj_data_out<31:0>	Output	Data written to an external device
pj_data_in<31:0>	Input	Data read from an external device
pj_tv	Output	Indication of a valid cycle on the bus
pj_size<1:0>	Output	The size of a transfer

TABLE 7-2 BIU External Interface Signals *(Continued)*

Signal	Type	Description
pj_type<3:0>	Output	The type of a transfer
pj_ack<1:0>	Input	The bus acknowledgment for a transaction
pj_ale	Output	Enabling of the picoJava-II address latch

CHAPTER 8

Powerdown, Clock, Reset, and Scan Unit (PCSU)

The Powerdown, Clock, Reset, and Scan Unit (PCSU) controls the following:

- The powerdown functions (see the next section, "*Power Management*")
- The internal clock functions (see *Clock Management* on page 88)
- The system reset function (see *Reset Management* on page 88)
- The scan and test functions (see *Scan and Test Features* on page 89)

In addition to the sections listed above, the chapter contains these sections:

- *Interface Signals* on page 91
- *Debug and Trace Features* on page 92
- *JTAG Support* on page 93

8.1 Power Management

The picoJava-II core offers two low-power modes: normal mode and standby mode.

8.1.1 Normal Mode

In normal mode, the function units conserve power as much as possible. They perform the following operations:

- The Instruction Cache Unit (ICU) generates the powerdown signal `icram_powerdown` while waiting for noncacheable or missed data. This signal remains asserted until the ICU resumes an operation.

  Alternately, the ICU can connect `icram_powerdown` to a powerdown pin of the instruction cache RAM megacell.

- The Data Cache Unit (DCU) generates the dcu_pwrdown signal if all of the following conditions are met:
  - There are no DCU-related instructions in the E or C stage.
  - There are no active cache fill cycles.
  - The DCU is not replacing dirty lines.
  - The DCU is not executing the zero_line instruction.
  - There are no outstanding requests when a stalled instruction is present.
  - There are no pending stores.

  The dcu_pwrdown signal remains asserted until the DCU resumes an operation.

8.1.2 Standby Mode

The core can assume standby mode by executing the powerdown instruction, priv_powerdown. The PCSU then does the following:

1. **Asserts the internal pcsu_powerdown signal to all function units and waits until they acquiesce.**
2. **Enters standby mode and indicates the standby state to the system with the pj_standby_out pin.**

Protocol

FIGURE 8-1 shows the protocol for standby mode.

]

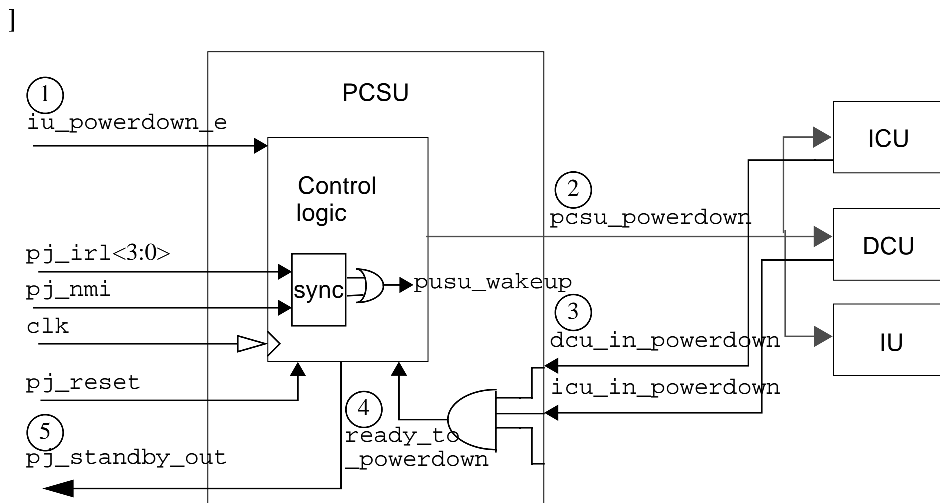


FIGURE 8-1 Protocol for the PCSU Standby Mode

Where:

① The core executes a powerdown instruction and causes assertion of iu_powerdown_e.

② The PSCU asserts the pcsu_powerdown signal to the ICU, the DCU, and the IU. See the next section for details.

③ The ICU and the DCU assert the dcu_in_powerdown and icu_in_powerdown.

④ When these signals are combined, the PCSU issues the ready_to_powerdown signal to the control logic unit.

⑤ The PCSU issues the pj_standby_out signal.

And, external to the core, pj_standby_out can be used to gate the clock.

Transition

Here is what takes place when the functional units enter standby mode.

- The ICU enters standby mode after completing the pending accesses and after the PCSU has asserted the pcsu_powerdown signal. It then asserts the icu_in_powerdown signal to indicate to the PCSU that it is ready for the PCSU to shut off the clock.
- The DCU enters standby mode after the PCSU has asserted the pcsu_powerdown signal, where it waits until there are no instructions in the E stage or beyond. Finally, the DCU asserts the dcu_in_powerdown signal to the PCSU to indicate that it is ready for the PCSU to shut off the clock.

Exit from Standby Mode

The core wakes up from standby mode if there is an interrupt on pj_irl or pj_nmi. In this case, resumption of clocks to the core takes *four* clock cycles, including the two clocks that are used in the synchronizer stage.

A reset turns off standby mode for the core.

Cache Flushing

It is not necessary to flush the internal cache as long as the caches are in static state in standby mode. Otherwise, software must push out all dirty entries to external memory before executing the powerdown instruction.

Note – Enter standby mode through `priv_powerdown`.

8.2 Clock Management

The core runs off a single clock source, `clk`. There is no clock gating inside the core; however, the `pj_standby_out` signal can be used to gate the clock to the core.

During the scan, `clk` serves as a scan clock after the `sm` signal is asserted. Muxing the JTAG clock and the system clock into `clk` must be external to the core.

The core user implements a systemwide clock distribution scheme. The system-level clock generator distributes clocks to the entire system at various frequencies.

The source that drives `clk` can be the JTAG clock, a PLL, or a systemwide clock controller and is outside of the core.

8.3 Reset Management

When a reset operation occurs in the processor core, the registers inside the core reset themselves. The RAM blocks in the stack cache, the instruction cache, and the data cache remain unchanged. A reset operation causes the processor to start executing the next instruction at address 0x00000000. For the values of the registers after reset, refer to *picoJava-II Programmer's Reference Manual.*

The core can reset itself in one of two ways:

- If the `pj_reset` signal to the core is active, the processor enters reset mode on the next rising edge of `clk`. A low-high transition is assumed to be asynchronous to the clock; however, a high-low transition must meet the setup requirements for `clk`, which must remain asserted for at least eight processor clocks before it is deasserted. The processor may or may not latch this signal prior to using it.

Note – Reset is active high.

Note – As part of the reset trap sequence, we recommend that you invalidate the caches explicitly with diagnostic write instructions prior to enabling them.

- Software can also cause the processor to trigger a reset. The core supports the `priv_reset` instruction, which executes a system-level reset sequence.

See *Processor Interface Signals* on page 100 for more information.

8.4 Scan and Test Features

It is difficult to control and observe the internals of a core and its I/O when it is embedded within a larger design. Since the `clk` signal is accessible (a JTAG controller can mux the JTAG clock into `clk`), the core provides the basic mux scan facility that interfaces with three signals: `si`, `so`, and `sm`. A JTAG controller can connect to the core with these signals.

Due to JTAG limitations, the scan chain is split. The input pins can be driven by the JTAG TAP controller or directly by the tester. As shown in FIGURE 8-2, you must supply the JTAG block for single-stepping core operations for debugging.

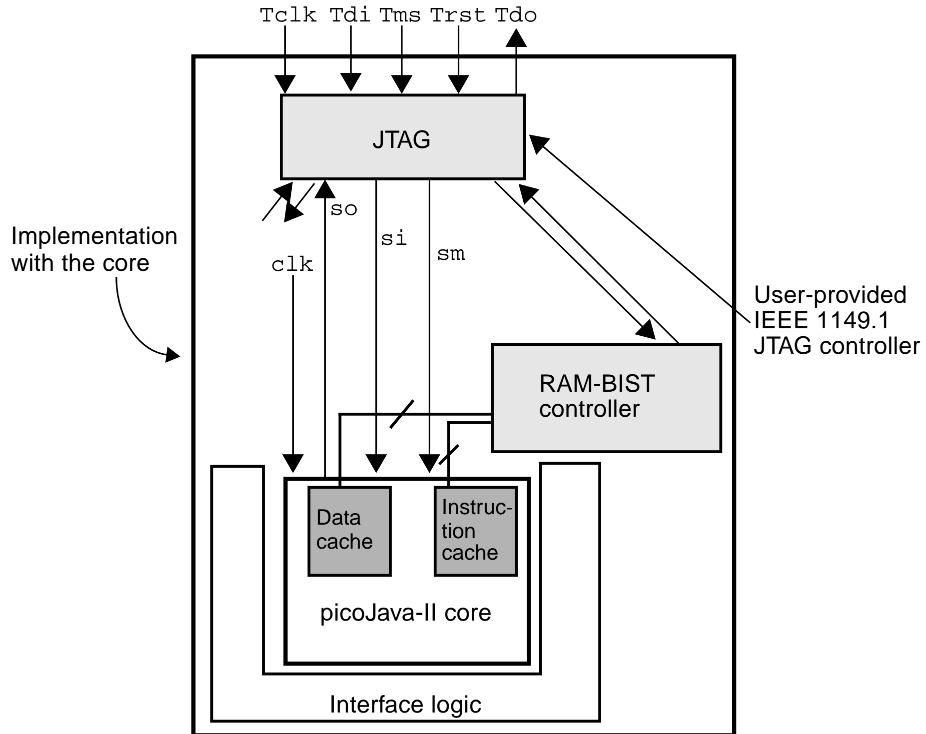


FIGURE 8-2 Test-Related Vendor Issue

Since the core is a high-level RTL model, regular scan-based testing is difficult to implement. However, the core contains a few BIST (built-in-self-test) hooks that increase the observability of the core when it is embedded in a larger chip. These features are selectable and are *not* present in the baseline core. The main emphasis is on testing the core memory.

You can select an option to get a version of the picoJava-II netlist with an embedded BIST controller, which tests the instruction and data caches.

Note – This step may reduce the core's operating speed slightly, typically a mux delay. Also, the BIST controller contains the hooks to interface to a JTAG controller, which may already exist in the chip.

8.5 Interface Signals

TABLE 8-1, TABLE 8-2, and TABLE 8-3 define the PCSU signals.

TABLE 8-1 PCSU Interface with JTAG

Signal	Type	Description
clk	Input	The core clock, a mux output of the external system clock and scan clock
si	Input	Scan input
sm	Input	Scan mode enable
so	Output	Scan output

TABLE 8-2 PCSU Interface with Function Units

Signal	Type	Description
icu_in_powerdown	Input	The ICU is ready for standby mode
dcu_in_powerdown	Input	The DCU is ready for standby mode
iu_powerdown_e	Input	The IU executes priv_powerdown
pcsu_powerdown	Output	A standby mode request to the units
pj_irl_sync<3:0>	Output	Synchronized pj_irl to the IU
pj_nmi_sync	Output	Synchronized pj_nmi to the IU

TABLE 8-3 PCSU Interface with External Systems

Signal	Type	Description
pj_irl<3:0>	Input	Interrupt request lines
pj_nmi	Input	A nonmaskable interrupt
pj_reset	Input	Reset
pj_clk_out	Output	Free-running clk output to the external system
pj_standby_out	Output	Standby mode

8.6 Debug and Trace Features

The core supports data breakpoint and instruction breakpoint traps. It contains three 32-bit registers that control the debug features:

- BRK1A, which contains the instruction PC to be trapped on
- BRK2A, which contains the mask bits for the comparison
- BRK12C, which controls the various matches

For more details, see *picoJava-II Programmer's Reference Manual*.

You cannot set breakpoints on the size of the data load or store. Rather, you must account for them in the software that uses this feature. The address match for the incoming address is based on the address in the breakpoint address registers and the corresponding enable bits in the breakpoint control register.

The core provides control signals and PC address to the external system for you to trace the instruction stream dynamically. You must provide a trace FIFO in which to collect traces, along with a mechanism for reading the contents of the trace buffer. See FIGURE 8-3.

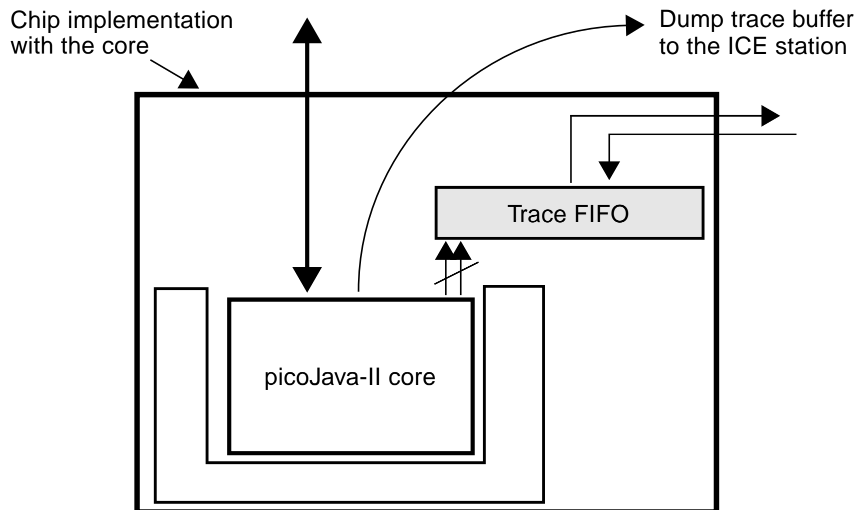


FIGURE 8-3 Tracing the Instruction Flow Using the picoJava-II Core

8.7 JTAG Support

The core supports IEEE 1149.1 JTAG scan capabilities through JTAG support pins in the core.

The picoJava-II JTAG that supports the pin interface to the JTAG controller is:

`clk`	`sm`
`si`	`so`

The JTAG controller is not part of the core and must be supplied externally.

8.7.1 Full Internal Scan

All internal flip-flops are accessible in scan mode. The CPU state can be extracted from the scan chain with JTAG. Programmer-visible registers are part of the scan chain and are accessible through JTAG.

8.7.2 Breakpoints and External Halt Mode

Breakpoints can force the core into halt mode. To force halt mode, set the `halt` bit in the breakpoint control register (`BRK12C`[31]); this setting stalls the CPU pipeline as the clock continues to run.

The core input pin `pj_halt` forces the core into halt mode due to external events, such as a logic analyzer. While asserted, the CPU remains in halt mode. Normal operation resumes when `pj_halt` is deasserted.

To use breakpoints, the external agent must correctly set up the breakpoint control register by turning on the `halt` bit, then run the program until the breakpoint hits. The processor halts as soon as the breakpoint is reached.

To resume the processor, the external agent must issue a single pulse on the `pj_resume` input port, which restarts the processor. If at that point the external agent wants to single-step the processor, it must follow the `pj_halt` and `pj_inst_complete` sequencing steps (see the following section). The `pj_in_halt` signal is asserted by the core to indicate that the processor has reached the halt state through the breakpoint, which was set by the user.

The processor asserts the pj_brk1_sync and pj_brk2_sync signals if the halt mode bit in the breakpoint control register is not set *and* the breakpoint is detected. It does *not* halt in this case, however. The external agent must halt the processor explicitly (for example, by stopping the clock) if it wants to do so or just observe the event.

8.7.3 Single-Stepping

The JTAG controller can be designed to enter scan mode when the CPU state is in halt mode by observing the pj_brk1_sync, pj_brk2_sync, and pj_in_halt signals.

The JTAG controller must take complete control over the clock during single-stepping. This control includes bringing the CPU out of the halt mode by inputting the pj_resume pin signal into the core.

Single clock mode is the picoJava-II single-step scan mode. The JTAG controller issues one single TCK clock, after which the scan chain is accessed. The pj_inst_complete signal is a core output signal.

To single-step the processor, the user of the core must use the following two signals: pj_halt and pj_inst_complete. The external agent must deassert the pj_halt signal until it sees the pj_inst_complete signal going high. In the *same* cycle, the external agent must also assert the pj_halt signal to prevent the next instruction from completing. The external agent can then scan values or observe signals in the core.

See FIGURE 8-4, FIGURE 8-5, and FIGURE 8-6 for details.

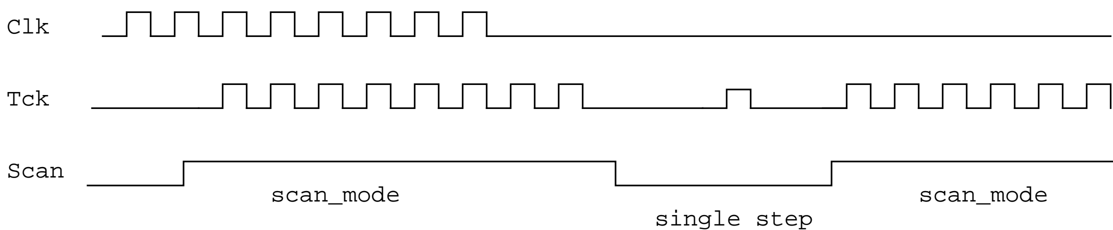


FIGURE 8-4 Single-Step Timing Through Scan Mode

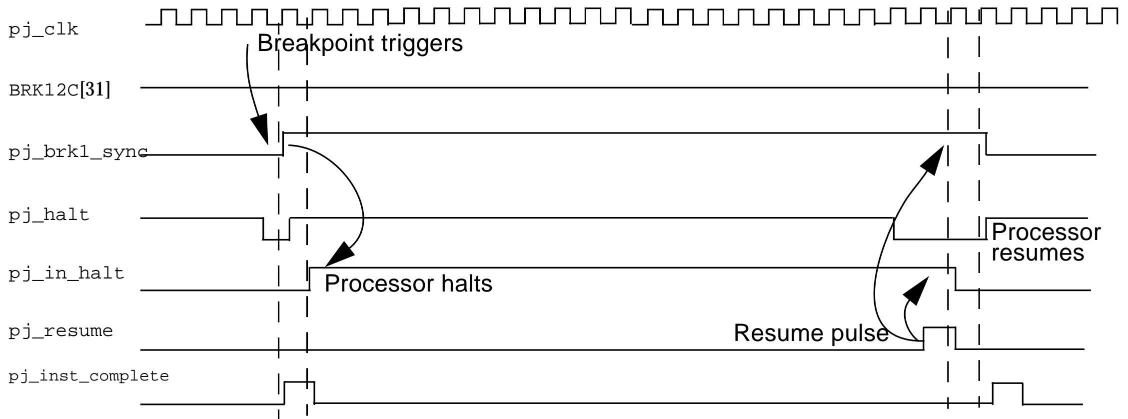


FIGURE 8-5 Single-Stepping Using Breakpoints

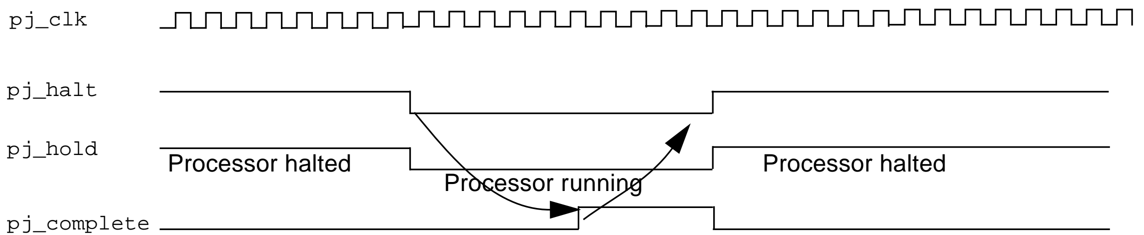


FIGURE 8-6 Single-Stepping Using `pj_halt`

8.7.4 Nonscannable Arrays

Cache and tag arrays are accessible only through extended bytecode instructions. You can use JTAG in single-step mode to extract cache contents by scanning in the appropriate cache address and control signals, applying a clock, and scanning out the flip-flops that contain the accessed data.

This method also applies to the stack cache.

CHAPTER 9

External Interface

This chapter defines the interface between the picoJava-II core and all external devices.

The core includes both the CPU (`cpu.v`) and the Bus Interface Unit (BIU). The BIU is, however, an independent module to provide flexibility for merging with the memory controller.

As reference for implementation, see the simulation environment (`sys.v`) for details on how the CPU, the BIU, and the memory module are instantiated.

This chapter contains the following sections:

- *Core Interface Signals* on page 97
- *picoJava-II Transactions* on page 105
- *Endianness and Cacheability* on page 111
- *Customizable Features* on page 111

9.1 Core Interface Signals

The core interface supports three types of signals:

- *Processor Interface Signals* (see page 100)
- *Memory Interface Signals* (see page 101)
- *Trace and Debug Signals* (see page 103)

FIGURE 9-1 illustrates the picoJava-II external interface and the three types of signals.

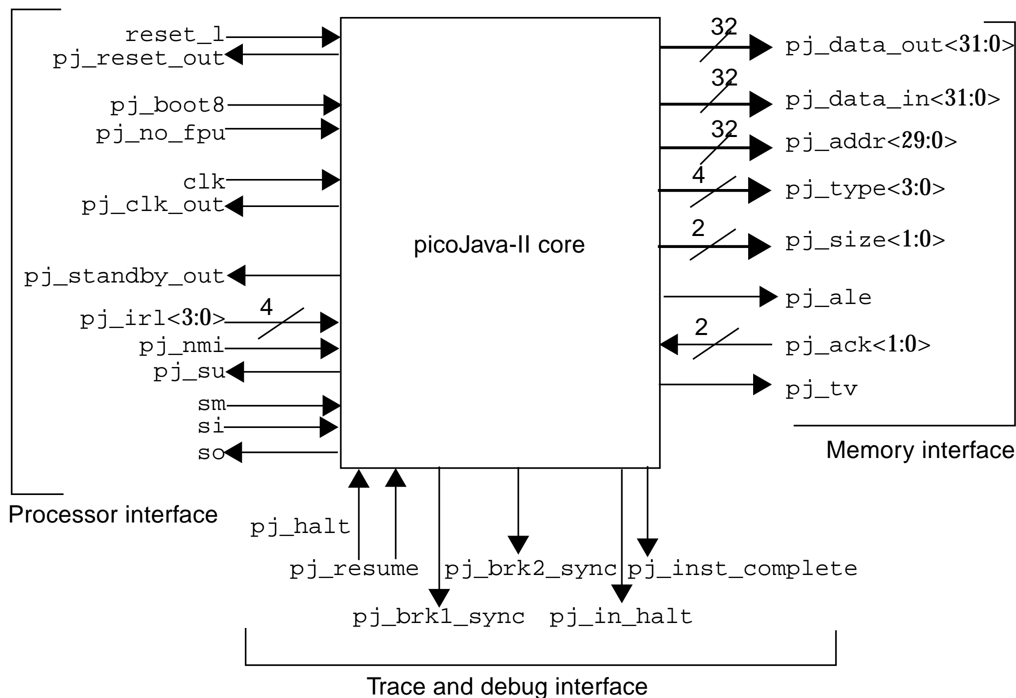


FIGURE 9-1 The picoJava-II External Interface

The triad R0, R1, or R2 denotes the timing requirement for each of the core interface signals.

- R0 is a signal from a flip-flop.
- R1 is a signal generated during the first phase of `clk`.
- R2 is a signal generated during the second phase of `clk`.

TABLE 9-1 shows the core interface signals, the equivalent gate delays, the signal type, the R number, whether a signal is synchronous or asynchronous, and the signal definitions.

TABLE 9-1 Interface Signals

Signal	Delay[1]	Type	R No.	A/S	Definition
`reset_l`	5	Setup	R1	S	Reset and startup of the processor at address 0x00000000
`pj_reset_out`	2	Valid	R2	S	Execution of the reset extended bytecode
`clk`	-	In	R0	CLK	The core clock
`pj_clk_out`	*	Valid	R1	S	The picoJava-II clock to external interfaces

TABLE 9-1 Interface Signals *(Continued)*

Signal	Delay[1]	Type	R No.	A/S	Definition
pj_irl<3:0>	-	In		A	Interrupt of exception signals
pj_nmi	-	In		A	Nonmaskable interrupt input to the core
pj_boot8	-	Static	R0	S	Size control of instruction cache fetches
pj_standby_out	4	Valid	R1	S	Notification to the system that the processor is in standby mode
pj_no_fpu	-	Static	R0	S	Disabling of the Floating Point Unit (FPU)
so	2	Valid	R0	S	Basic scan facility
sm	5	Setup	R1	S	Switch of flip-flops in the core to serial shift
si	3	Setup	R1	S	Input to the processor core scan chain
pj_data_in<31:0>	5	Setup	R1	S	Data reads
pj_data_out<31:0>	4	Valid	R1	S	Data writes
pj_addr<31:0>	10	Valid	R1	S	Interfaces with nonmultiplexed 32-bit address bus
pj_size<1:0>	6	Valid	R1	S	The size of the requested data
pj_type<3:0>	6	Valid	R1	S	The transaction type requested by the Integer Unit
pj_tv	5	Valid	R1	S	Assertion to start new transaction to the memory controller
pj_ack<1:0>	12	Setup	R0	S	Indication that data are driving in the same cycle on pj_data_in
pj_ale	6	Valid	R0	S	Enabling of address latching
pj_halt	2	Setup	R0	S	Termination of instruction fetches
pj_resume	2	Setup	R0	S	Resumption of instruction fetches
pj_brk1_sync	10	Valid	R2	S	Detection of Breakpoint 1
pj_brk2_sync	10	Valid	R2	S	Detection of Breakpoint 2
pj_in_halt	1	Valid	R0	S	The processor is in halt mode (not fetching instructions).
pj_inst_complete	10	Valid	R2	S	Retrieval of an instruction (when this signal is high)
pj_su	0	Valid	R0	S	The processor is in supervisor mode

[1]The number is in terms of equivalent two-input NAND gate delays.

9.1.1 Processor Interface Signals

The processor interface signals are:

- `reset_l` (input) [R1] — An input to the core. A low-to-high transition on this pin resets the processor and starts the processor at address 0x00000000. The low-high transition is asynchronous to the clock, but the high-low transition should be set up to the `clk` signal. The processor can latch this signal prior to its use, which must remain asserted for at least eight processor clocks before it is deasserted.
- `pj_reset_out` (output) [R2] — An output from the core. A low-to-high transition on this signal indicates that the processor has executed the reset extended bytecode. The core can use this signal to execute a system-level reset sequence.

  The core asserts this signal for only one clock on encountering `priv_reset`. The external agent must register it in time and reset the core with the `reset_l` signal.
- `clk` (input) [R0] — A single clock source. In case of a need to increase the core frequency, decouple it from the system frequency at the memory controller interface. This step makes the core easy to design (no synchronizers) and avoids the synchronization penalty for cores, which run in lock-step with a high-speed memory clock. If necessary, the memory controller can use a separate clock. (The memory controller can use wait state pins as input to accommodate DRAM access variations, but the pins are not an input to the core.)
- `pj_clk_out` (output) [R1] — The core clock to the external interface. This signal is necessary during powerdown when the core stops the internal clock to the system, but keeps the `pj_clk_out` clock running. Use this signal to refresh the DRAMs, or leave it unconnected.
- `pj_irl` <3:0> (input) — Interrupt signals to the core, which latches these asynchronous signals with a synchronizer. A system that is based on the core can selectively disable or mask the input combinations by setting the appropriate bits in the core status register. A nonmaskable interrupt (`pj_nmi`), which consists of asynchronous pins, inputs separately to the core.

  To generate an interrupt, the system must assert the relevant input until the processor responds by jumping to the addressed location and starts execution. The interrupt controller register that generates the interrupt must be writable from the core.

  See *picoJava-II Programmer's Reference Manual* for the exception vectors and their addresses and priorities. All these interrupts are maskable in two ways: by setting the `PSR.PIL` bit to the proper value or by setting the `PSR.IE` bit to 0.
- `pj_nmi` (input) — An asynchronous signal and the only nonmaskable interrupt input to the core. The timing and behavior are similar to those of the `pj_irl`<3:0> lines, except that this is a nonmaskable interrupt. The trap type value is 0x30. Disable this interrupt by setting the `PSR.IE` bit to 0.

- pj_boot8 (input) [R0] — A signal that controls the instruction fetch mode during bootup and allows the system designer to interface with the core to an 8-bit boot PROM interface. Setting this bit to high fetches the instruction in bytes. The location of the byte on pj_data_in<31:0> is pj_data_in<31:24>. This pin should not change its value as reflected in the PSR once the processor has come out of reset. Writing 0 to the PSR.BM8 bit clears the bit, taking it out of the 8-bit boot mode.
- pj_su (output) [R0] — An output from the core, this signal indicates the privilege level of the processor. A high setting indicates that the processor is in supervisor mode; otherwise, it is in user mode.
- pj_standby_out (output) [R1] — A signal for the standby mode. The system can use this signal to power down other units outside the core. When the core enters powerdown mode due to execution of a powerdown instruction, it asserts the pj_standby_out signal.
- pj_no_fpu (input) [R0] — A signal that disables the internal FPU of the core. It can reduce the area and power in certain implementations. The value of this pin, as reflected in the processor status register, should not change once the processor has come out of reset.

  If the clk signal is accessible, the core can provide basic scan facility with the three other signals: si, so, and sm. A JTAG controller connects to the core with these signals.
- so (output) [R0] — With this signal, the scan chain is split due to JTAG limitations. The tester or an external JTAG test access port control can drive the input pins.
- sm (input) [R1] — This signal switches the scannable flip-flops in the core into a serial shift register.
- si (input) [R1] — This signal is the input to the scan chain of the processor core.

9.1.2 Memory Interface Signals

The memory interface signals are:

- pj_data_in<31:0> (input) [R1] — A 32-bit wide, unidirectional data bus to the core uses for read-type transactions. Typically, the memory controller drives this bus and arbitrates for more than two masters since core interface allows the memory controller to define multiple bus masters. Data on this bus are valid when the bus drive (usually the memory controller) asserts the signal pj_ack<1:0>.
- pj_data_out<31:0> (output) [R1] — A 32-bit wide, unidirectional data bus for the write-type transaction.
- pj_addr<29:0> (output) [R1] — A 30-bit address bus for the core memory interface.

- pj_size<1:0> (output) [R1] — A signal that indicates the data size in the request phase, as defined in TABLE 9-2. *picoJava-II Transactions* on page 105 explains the request phase.

TABLE 9-2 Transaction Sizes

pj_size**<1:0>**	**Transaction Size**
0x0	1 byte
0x1	2 bytes (half-word)
0x2	4 bytes (word)
0x3	Reserved

Note – When pj_type equals 2 and pj_size equals 0, the instruction fetch is in 8-bit boot mode, a mode *not* allowed with the instruction cache on.

- pj_type<3:0> (output) [R1] — A signal that indicates the data type requested by the IU in the request phase, as defined in TABLE 9-3. *picoJava-II Transactions* on page 105 explains the request phase.

TABLE 9-3 Transaction Types

pj_type**<3:0>**	**Transaction Type**	**Transaction Size** pj_size**<1:0>**
0x2	Noncached instruction fetch request	0x0 - 8-bit boot, 0x2 - 32-bit boot
0x0	Instruction cache fill request (16 bytes)	See footnote 1
0x4	Data cache fill request (16 bytes)	See footnote 1
0x5	Data cache writeback	See footnote 1
0x6	Noncached data cache load request	0x0, 0x1, 0x2
0x7	Noncached data cache store request	0x0, 0x1, 0x2
0xc	SMU-initiated cache fill request	See footnote 1
0xd	SMU-initiated writeback	See footnote 1
0xe	Noncacheable SMU load request	0x2
0xf	Noncacheable SMU store request	0x2

[1]This field is irrelevant.

- pj_tv (output) [R1] — A signal the core asserts to start a new transaction to the memory controller. This signal also indicates that the pj_addr, pj_type, and pj_size signals are valid. The core keeps this signal on until the cycle in which it samples the pj_ack signal is active.

- pj_ale (output) [R1] — A signal that stands for "address latch enable," which the core asserts to start a new transaction to the memory controller. This signal also indicates that the pj_addr, pj_type, and pj_size signals are valid. A low pulse on this signal indicates that the memory controller can latch the pj_addr, pj_type, and pj_size signals in that cycle.
- pj_ack<1:0> (input) [R0] — A signal that the memory controller uses to indicate to the core that data are driven in the *same* cycle as that on pj_data_in<31:0> in the case of a read cycle.

  The core must understand how many acks follow and then wait for the requisite number of acks. In a write cycle, this signal indicates that the core should switch the data bus to the next portion of a burst write request. See TABLE 9-4 for the four types of acks.

TABLE 9-4 acks

Operation	pj_ack**<1:0>**
Idle cycle	0x0
Valid data	0x1
ERROR1 (memory error)	0x2
ERROR2 (I/O error)	0x3

There is no retry mechanism. Since the core uses pj_ack as a timing-critical signal in many places directly without prior registration, it should drive it from a flip-flop in the external system. The memory controller can indicate an error ack in any part of a burst request.

Note – When an error occurs in a burst transaction, the memory controller terminates the transaction.

9.1.3 Trace and Debug Signals

Following are the trace and debug signals.

- pj_halt (input) [R0] — An external agent uses this signal to stop the processor from fetching instructions.
- pj_resume (input) [R0] — An external agent uses this signal to start the processor in fetching instructions.
- pj_brk1_sync (output) [R2] — An external agent uses this signal to indicate that the core has detected breakpoint1.
- pj_brk2_sync (output) [R2] — An external agent uses this signal to indicate that the core has detected breakpoint2.

- pj_in_halt (output) [R0 — An external agent uses this signal to indicate that the processor is in halt mode and is not fetching instructions.
- pj_inst_complete (output) [R2] — An external agent uses this signal to indicate the retrieval of an instruction.

FIGURE 9-2 is an example of a system based on the core.

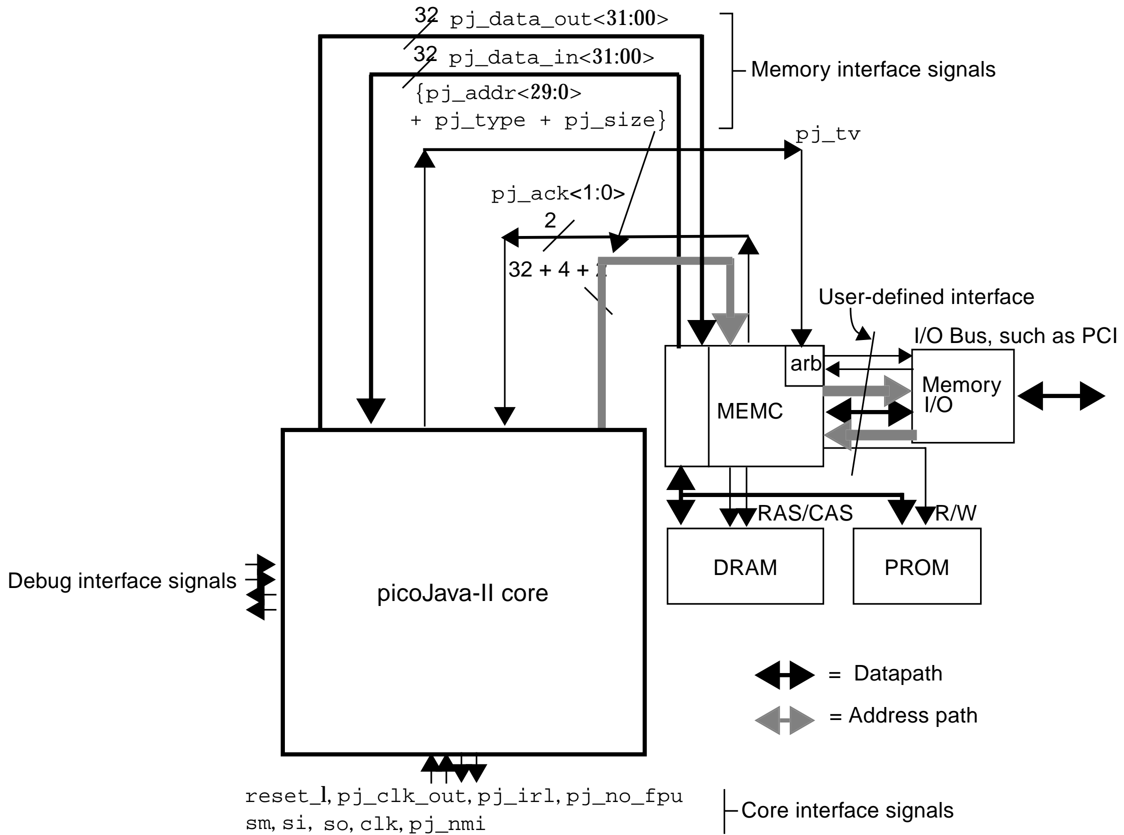


FIGURE 9-2 An Example of a System Based on the picoJava-II Core

9.2 picoJava-II Transactions

The core can interface with various memory controllers, such as SDRAM, EDO, SRAM, DRAM, and FLASH, as well as I/O controllers like PCI, USB, and PCMCIA. To achieve this flexibility without loss in memory latency, the core memory interface is a *virtual* memory controller.

There are two main transaction types. See *Read-Type Transactions* and *Write-Type Transactions* on page 108.

Note – There is no provision for the memory controller to snoop into the core to probe the caches or to access the internal registers.

The transaction protocol, one of request and accept, eases the design of the memory controller, which can be done by a third party. This protocol also achieves the minimum possible latency for reads and enables the memory controller to add more ports to itself while supporting a high-bandwidth memory, such as synchronous DRAMs.

The memory bus is a 32-bit wide bus with sub-32-bit accesses in big-endian ordering. There are no byte-enables; instead, the core exports the address and the access size. Sub-32-bit devices must incorporate swap logic at their end for reads and writes. Accesses to devices greater than 32-bits must also buffer and route the data to the proper memory module. The picoJava-II bus is optimized for 32-bit transfers.

9.2.1 Boot Mode

During bootup, the `pj_boot8` signal controls the instruction fetch mode and allows the software to interface with the core with an 8-bit boot PROM interface. Setting this signal to high at boot time causes instruction fetches to be in single-byte sizes, as reflected in the `PSR.BM8` bit of the `PSR`.

While the processor is in boot mode, the location of the byte on `pj_data_in`<31:0> is `pj_data_in`<31:24>. This signal should not change its value (as reflected in the `PSR`) once the processor comes out of reset.

To perform boot mode correctly, do the following:

1. **Use the `priv_write_psr` instruction *after* the branch to the nonboot memory location has occurred; the execution of this instruction disables the boot mode.**
2. **Issue `priv_read_psr` prior to `priv_write_psr` if you wish to execute `priv_write_psr` without modifying the contents.**

3. **Start the pair at word boundary.**
4. **Insert four `nop`s after `priv_write_psr` to recover from boot mode to word access.**
5. **Tie the `pj_boot8` pin to high.**

CODE EXAMPLE 9-1 summarizes the above steps.

```
(8-bit PROM)
 Boot_Start:  Instruction_1
              Instruction_2
              ...
              goto Application    // Application program must
                                // begin at word boundary.

(Nonboot memory)
Application:         // At word boundary
            priv_read_reg_psr   // To avoid modifying PSR, push
                              // its content.
         sipush 0xBFFF  // Zero bit 14 (PSR.BM8).
         iand
         priv_write_reg_psr  // Disable boot mode by
                             // popping the content to PSR.

         nop      // Insert four nops to recover
         nop      // from boot mode to word access.
         nop      // Word access.
         nop
```

CODE EXAMPLE 9-1 Sample Boot Sequence Code

9.2.2 Read-Type Transactions

The instruction cache, data cache, and the SMU use read-type transactions to read from either the memory or from I/O space and further classify these transactions, depending on the `pj_type<2:0>` pins of the core, as shown in TABLE 9-3 on page 102.

FIGURE 9-1 shows a read-type transaction.

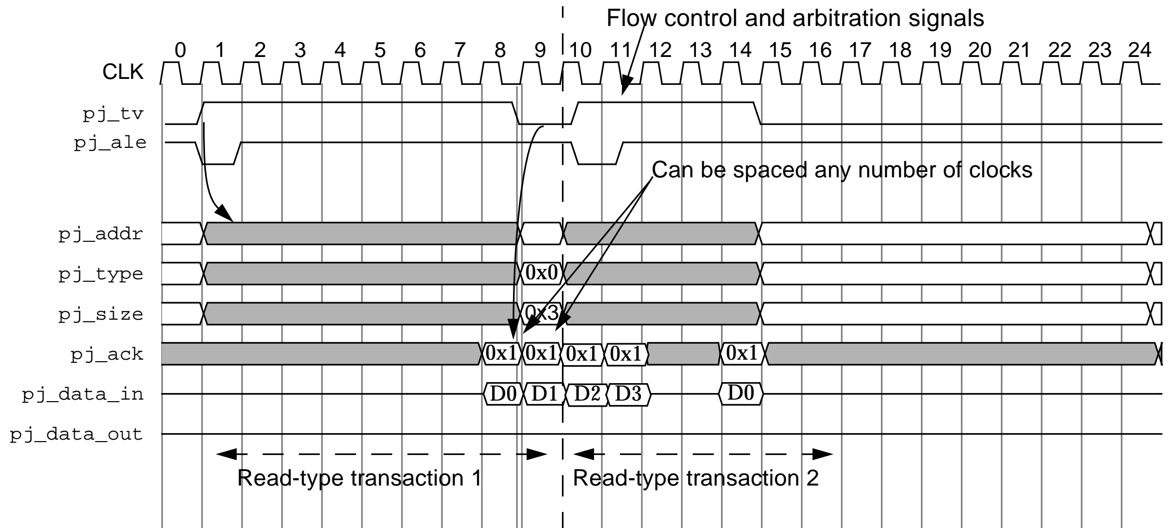


FIGURE 9-3 A Cached Read Transaction Followed by a Noncached Read Transaction

In the request phase of the transaction, the core makes a request to the memory controller by asserting the pj_tv signal along with the pj_type, pj_size, and pj_addr signals. It then waits for acceptance from the memory controller and samples its pj_ack signal for the timing of the acceptance.

Assertion of the pj_ack signal to the core indicates that the memory controller has accepted the transaction from the picoJava-II port for a write-type instruction.

In a read-type transaction, the core waits until it detects the assertion of the pj_ack<1:0> signals after an arbitrary length of clock cycles. Afterward, if there are no outstanding transactions, deassertion of the pj_tv signal occurs.

Note – On a burst-type request, pj_tv need not be deasserted if there is a pending request.

Continuous assertion of pj_tv occurs on back-to-back transactions on the same device, but it must put the values for the next transaction on the pj_type, pj_size, and pj_addr lines. pj_ale has a pulse that indicates the start of a new transaction.

Next, the memory controller responds to the data by driving the bus pj_data_in in the same cycles as the assertion of the pj_ack<1:0> signals. The pj_ack signal is a type R0 signal. The read-type transaction is now complete and the next transaction can begin immediately.

The arrows in FIGURE 9-3 indicate the transactions. Also shown in this figure are the cause-and-effect relationships between the read-type transaction signals at various times.

Note – Continuous assertion of the `pj_tv` signal occurs on back-to-back pending transactions from different devices. No deassertion takes place.

9.2.3 Write-Type Transactions

The data cache and the SMU use write-type transactions to store data to memory or to I/O space and further classify these transactions, depending on the `pj_type<2:0>` pins of the core, as shown in TABLE 9-3 on page 102.

FIGURE 9-4 shows a write-type transaction.

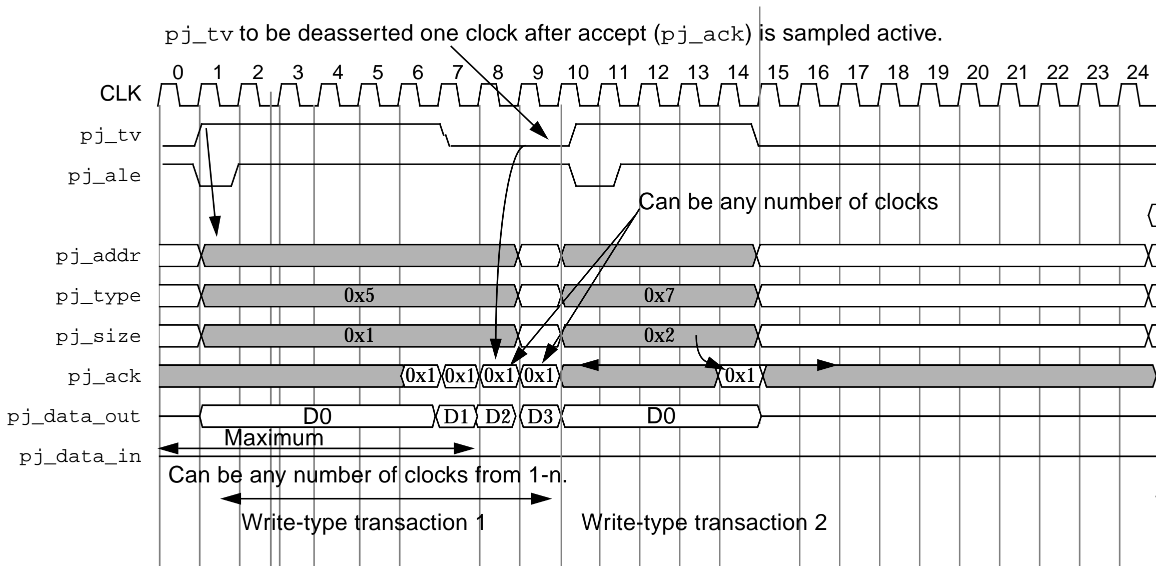


FIGURE 9-4 A Cached Write Transaction Followed by a Noncached Write Transaction

The core makes a request to the memory controller by asserting the `pj_tv`, `pj_type`, `pj_size`, `pj_data_out`, and `pj_addr` signals. After the `pj_ack` signal to the core is active, the core switches the data on the `pj_data_out` bus in the case of a burst transaction. (The memory controller uses the `pj_ack` signal for flow control.) No turnaround time is necessary because the data bus is a point-to-point bus, not a tri-state bus.

With this scheme, the core achieves near-optimal read-write bandwidth. The core bus is an on-chip bus only, hence these simplifications.

Note – Continuous assertion of the `pj_tv` signal occurs on back-to-back pending transactions from different devices. No deassertion takes place.

The memory controller scans `pj_size` and returns only one `ack` to fulfill the original load-store request. It ignores the `pj_type` field for this transaction. In all cases, as soon as a transaction is complete, a new one starts, thus utilizing almost the full bandwidth of the 32-bit data bus.

FIGURE 9-5, FIGURE 9-6, FIGURE 9-7 illustrate three types of write transactions.

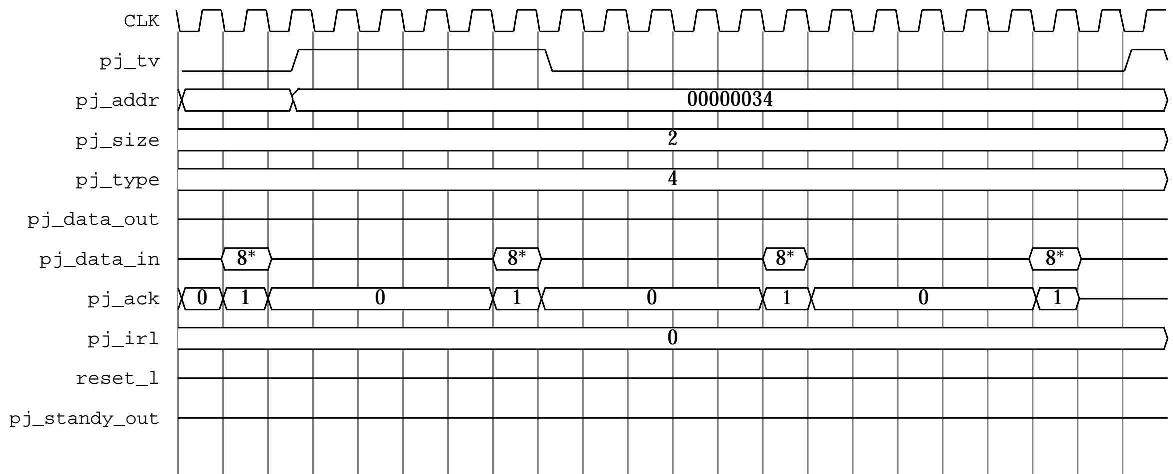


FIGURE 9-5 A Cached Load Miss

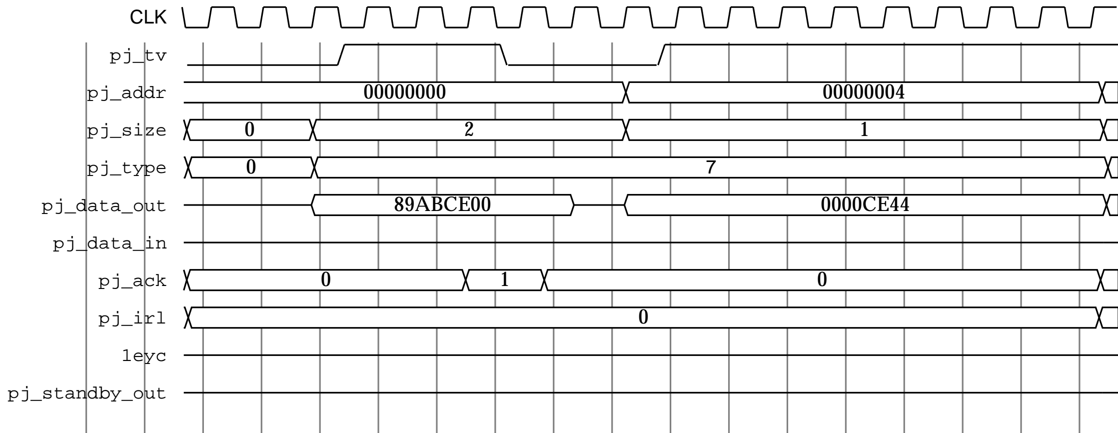


FIGURE 9-6 A Noncached Write

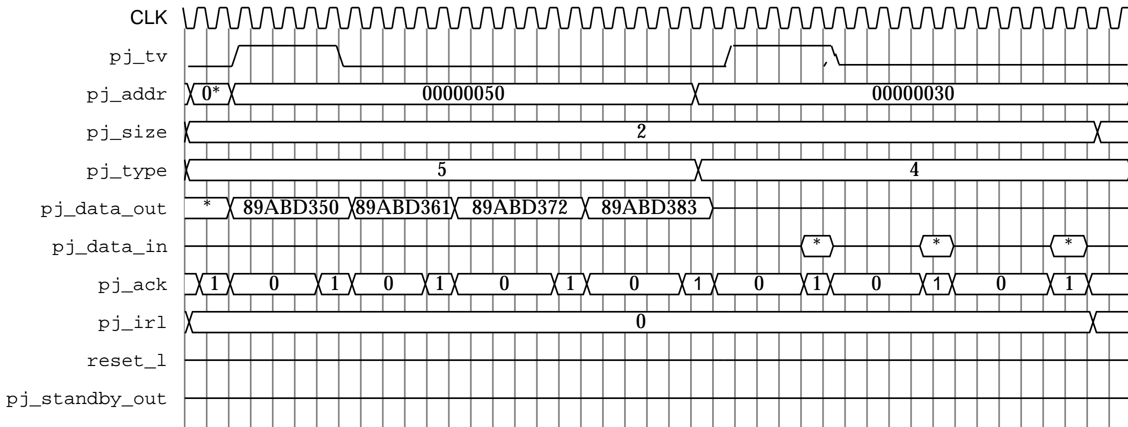


FIGURE 9-7 A Writeback and Cached Read Miss

9.3 Endianness and Cacheability

The core supports little-endian and big-endian transactions as well as cacheable and noncacheable transactions with either of the following:

- Special instructions
- Normal instructions that access a special address space

For details, refer to *picoJava-II Programmer's Reference Manual*.

9.4 Customizable Features

You can use the core in various applications, from network computers to low-power embedded mechanisms. The core is flexible and can meet the needs of different designs.

FIGURE 9-8 shows an example.

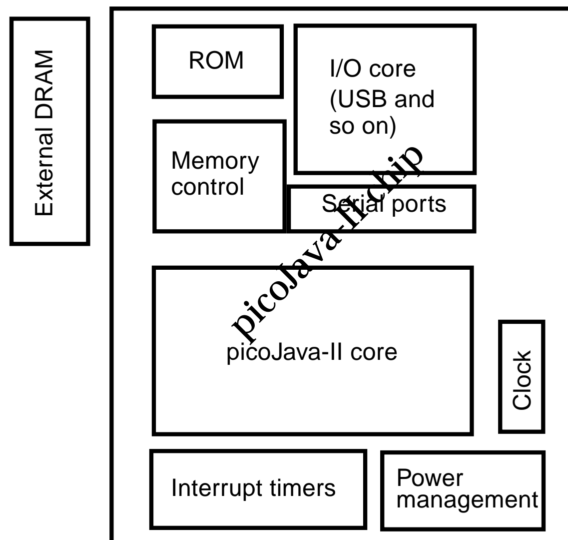


FIGURE 9-8 An Example of How To Use the picoJava-II Core

During implementation, you can configure the following three features of the core:

- Instruction cache sizes: 0 Kbytes, 1 Kbytes, 2 Kbytes, 4 Kbytes, 8 Kbytes, and 16 Kbytes.
- Data cache sizes: 0 Kbytes, 1 Kbytes, two Kbytes, 4 Kbytes, 8 Kbytes, and 16 Kbytes.
- FPU: You can include or exclude the FPU from your design. If you exclude the FPU, you must use software trap handlers to emulate floating-point instructions.

  This requirement allows a modular design for the core—you can add or remove modules and save chip area, power, or both.

CHAPTER 10

Traps and Interrupts

This chapter describes the trap and interrupt control mechanism in the picoJava-II core. It contains the following sections:

- *Definitions* on page 113
- *Traps* on page 114
- *Trap Control* on page 114
- *Interrupts* on page 114
- *Interrupt Control* on page 116

10.1 Definitions

Traps and interrupts are vectored transfers of control to the supervisor state through a trap table. The trap table's base address resides in the `TRAPBASE` register.

The offset within the trap table to which `TRAPBASE` points is determined by the trap type (`TT`). For details, see *picoJava-II Programmer's Reference Manual*.

When the core takes a trap, it creates a trap frame in which to save its current state. It then branches to the location of the trap handler and resumes execution. Trap execution continues on the stack and, when complete, returns to the interrupted method.

10.2 Traps

Three types of events can interrupt an operation in the core:

- Emulation traps, which are a subset of the Java virtual machine instructions (see Appendix A, Opcodes, in *picoJava-II Programmer's Reference Manual*)
- Traps that are generated during execution of instructions, such as runtime errors, exceptions, and hardware errors
- Interrupts that occur outside of the core

10.3 Trap Control

All traps are resolved and executed by the Integer Unit (IU) in the C stage of the pipeline. The IU prioritizes trap conditions, including trapping instructions, exceptions, and interrupts.

If multiple trap conditions occur in the same clock cycle, the core takes only the highest-priority trap. Priority is not relevant for trap conditions that occur at different times.

Since there is a clean interface through the stack mechanism for trap invocations, the core supports nested trap levels. There is no hardware limit to the number of nested levels; however, software must keep track of these levels.

10.4 Interrupts

There are two types of external interrupts: nonmaskable interrupts and maskable interrupts.

All interrupts are taken in the C stage of the pipeline. The `PC` of the next unexecuted instruction is saved on the stack.

10.4.1 Nonmaskable Interrupts (NMI)

A nonmaskable interrupt (`pj_nmi`) signals an event requiring immediate service. This interrupt can be disabled by setting the `IE` bit to 0; however, the core ignores the `PIL` check when servicing this interrupt. From the time the NMI pulse begins until the NMI handler is activated, the core ignores all other pulses on the NMI line. Once the core calls the NMI handler, another NMI can occur.

The core creates a trap frame to handle the NMI, the priority for which is higher than that of the highest maskable interrupt. It generates a trap frame similar to that of a trap call. If the software needs to access additional machine registers, it must use the extended bytecode instructions.

Using the NMI handler for debugging purposes to re-create a failure depends on the implementation. Optionally, the NMI handler can generate a reset with the `priv_reset` instruction.

Note – NMI is a level-sensitive, asynchronous signal. The period in which the core accepts NMI is directly related to the time in which the `IE` bit is not asserted.

10.4.2 Maskable Interrupts

A level-sensitive, encoded external maskable interrupt is presented to the core on the `pj_irl`<3:0> signal; the core accepts it if `PSR.IE`=1 and `IRL` > `PSR.PIL`. Interrupts can occur when the core is in either user mode or privilege mode.

When the core invokes a trap, the `PSR.IE` bit is set to 0, disabling further interrupts. The processor then performs the normal operations for a trap, that is, it sets up the trap frame, enters privilege mode, fetches the location of the interrupt handler from the trap table, and jumps to the first instruction of the interrupt handler. The interrupt handler can turn the `PSR.IE` bit back on if it wishes to enable further interrupts.

Interrupt servicing is a functionality of the software-defined interrupt service routines.

10.5 Interrupt Control

Interrupts conceptually enter the pipeline in the D stage of the pipeline, which the IU resolves and executes in the C stage like all other traps. The values of the Interrupt Enable (IE) and the Processor Interrupt Level (PIL) bits in the PSR determine if an interrupt has occurred.

An interrupt occurs if the following conditions are met:

- The PSR.IE bit is set to 1. When it is set to 0, the core ignores all interrupts, including the nonmaskable interrupt on the pj_nmi signal.
- With PSR.IE = 1, the interrupt request level (IRL) on the pj_irl signals is greater than the value in the PSR.PIL field, or the pj_nmi signal is asserted.

FIGURE 10-1 illustrates the mechanism for interrupt control.

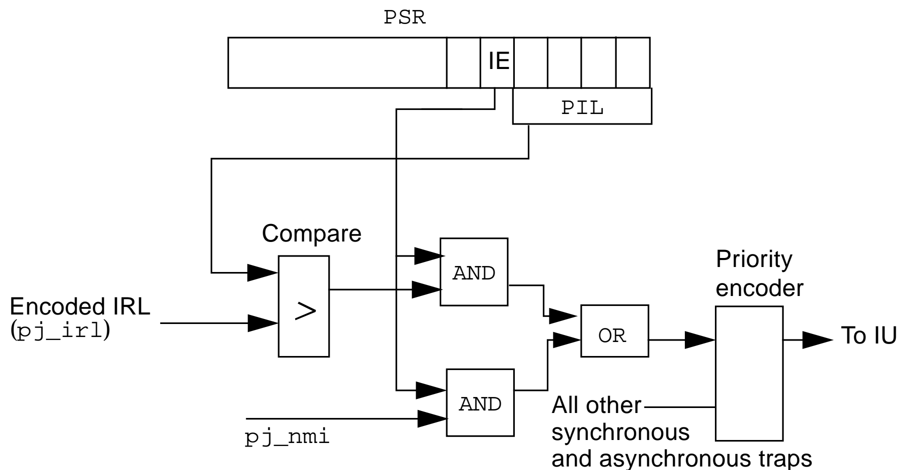


FIGURE 10-1 Interrupt Control Mechanism

CHAPTER 11

Megacells

The picoJava-II core has several megacells that you can customize for improved timing. This chapter describes the specifications for these megacells.

The timing statistics in this chapter assume a 200-MHz core with a 16-Kbyte data cache and a 16-Kbyte instruction cache. You can derive from these statistics corresponding timing statistics for other frequency and cache sizes.

This chapter contains the following sections:

- *Instruction Cache Data RAM* on page 117
- *Instruction Cache Tag RAM* on page 125
- *Data Cache Data RAM* on page 134
- *Data Cache Tag RAM* on page 141
- *Stack Cache RAM* on page 150
- *Floating Point Unit (FPU) ROM* on page 154
- *Integer Unit (IU) ROM* on page 161

11.1 Instruction Cache Data RAM

The instruction cache (icram) is logically organized as 2,048 entries by 64 bits. The cache line size is four words.

An 11-bit address bus (icu_addr) accesses the RAM for reading or writing data. All input to the RAM is internally latched to provide synchronous reads and writes.

In any given cycle, 64 bits of data (icram_dout) are read out or 32 bits of input data (icu_din) are written in from or to the cache. A 2-bit write enable (icu_ram_we) provides control to write data into the upper 32 bits or lower 32 bits position. Input for built-in self-test (bist*) is required at the pinout. A test mode pin (test_mode) is used for the cache to select taking either the normal data, address, and enable input (icu_din, icu_addr, and icu_ram_we) or BIST address, data, and enable

input (bist_icu_din, bist_icu_addr, and bist_icu_ram_we). The enable pin (enable) puts the instruction cache in disable mode when it is at low to conserve dynamic power consumption.

When the instruction cache is disabled, the value of output data (icram_dout) should remain unchanged as before regardless of address input and write enable changes. All input should be synchronously latched at the block boundary.

TABLE 11-1 lists and describes test_mode and the enable signals.

TABLE 11-1 test_mode and Enable Signals (Instruction Cache Data RAM)

Pin	enable **= 0**	enable **= 1**
test_mode = 0	The instruction cache is disabled. The value of the output data is maintained as before regardless of the address input and write enable changes.	The instruction cache is enabled. The normal address, data, and write enable input (icu_addr, icu_din, and icu_ram_we) are passed to the RAM.
test_mode = 1	Same as above.	The instruction cache is enabled and enters built-in test mode. The BIST address, data, and write enable input (bist_icu_addr, bist_icu_din, and bist_icu_ram_we) are passed to the RAM.

FIGURE 11-1 illustrates the instruction cache block.

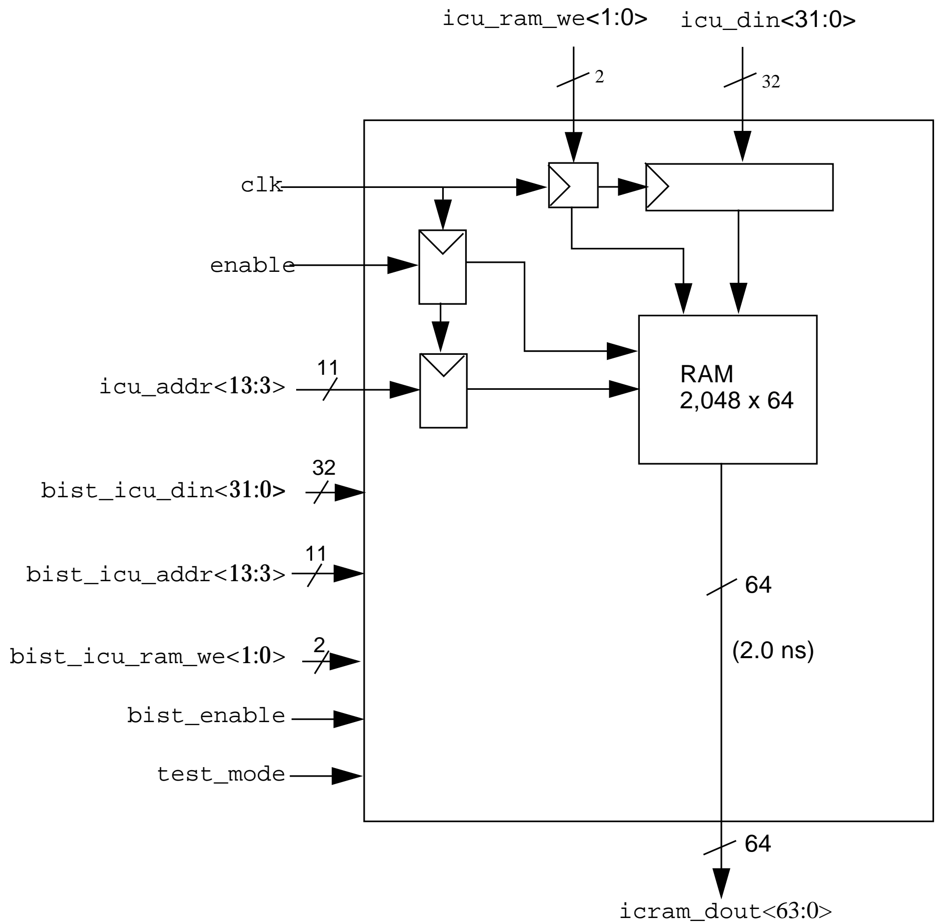


FIGURE 11-1 Instruction Cache Block (icram)

11.1.1 I/O Pins

TABLE 11-2 lists and describes the I/O pins.

TABLE 11-2 Input Pins (Instruction Cache Data RAM)

Pin	Description
clk	Clock input pin.
icu_addr<13:3>	Instruction cache address input. The word at the location addressed by icu_addr is accessed for either reading or writing its contents. icu_addr is 11 bits wide.
icu_din<31:0>	Instruction cache data input. The value of icu_din is written to the higher or lower 32 bits of the location addressed by the contents of icu_addr, depending on the value of icu_ram_we<1:0>.
icu_ram_we<1:0>	Instruction cache RAM write enable. The RAM performs a read-write from or to the location addressed by icu_addr, depending on the value of icu_ram_we<1:0>. Read access is performed when icu_ram_we<1:0> = 00. When icu_ram_we<1:0> = 10, the value of icu_din is written to the *higher* 32 bits of the location addressed by the contents of icu_addr. When icu_ram_we<1:0> = 01, the value of icu_din is written to the *lower* 32 bits of the location addressed by the contents of icu_addr.
enable	Megacell enable. This signal enables or disables the RAM to conserve dynamic power consumption when it is not being used. When enable = 0, the RAM is disabled and the value of all data output (icram_dout<63:0>) is maintained as it was before regardless of address input and write enable changes. Normal operations are performed when enable = 1.
bist_icu_addr<13:3>	The instruction cache address input for built-in self-test (BIST). The word at the location addressed by bist_icu_addr is accessed for either reading or writing its contents.
bist_icu_din<31:0>	The instruction cache data input for BIST. The value of bist_icu_din is written to the higher or lower 32 bits of the location addressed by the contents of bist_icu_addr, depending on the value of bist_icu_ram_we<1:0>.

TABLE 11-2 Input Pins (Instruction Cache Data RAM) *(Continued)*

Pin	Description
bist_icu_ram_we<1:0>	The instruction cache RAM write enable for BIST. The RAM performs a read-write from or to the location addressed by bist_icu_addr, depending on the value of bist_icu_ram_we<1:0>. Read access is performed when bist_icu_ram_we<1:0> = 00. When bist_icu_ram_we<1:0> = 10, the value of bist_icu_din is written to the higher 32 bits of the location addressed by the contents of bist_icu_addr. When bist_icu_ram_we<1:0> = 01, the value of bist_icu_din is written to the lower 32 bits of the location addressed by the contents of bist_icu_addr.
bist_enable	Megacell enable pin for BIST. When bist_enable = 0, the RAM is disabled and the value of all data output (icram_dout<63:0>) is maintained as before regardless of address input and write enable changes. Normal operations are performed when bist_enable = 1.
test_mode	Megacell test mode enable pin. When this pin is high, the megacell test mode is entered and BIST address, data, and write enable input (bist_icu_addr, bist_icu_din and bist_icu_ram_we) are taken to the RAM. The normal data, address, and write enable input (icu_din, icu_addr, and icu_ram_we) are passed to the RAM when this pin is low.

The output pin is icram_dout<63:0>, which is accessed using the address icu_addr (or bist_icu_addr if test_mode = 1).

11.1.2 I/O Signals

TABLE 11-3 describes the input and output pins of the instruction cache (icram) and their input loading and output drive strengths.

TABLE 11-3 I/O Signals (Instruction Cache Data RAM)

Signal	Description	I/O	Cin/Cload 200 MHz
clk	Clock input pin	Input	0.3 pf
icu_addr<13:3>	Address input for read/write access	Input	0.02 pf
icu_din<31:0>	Data input for write access	Input	0.02 pf
icu_ram_we<1:0>	Instruction cache RAM write enable	Input	0.02 pf
enable	RAM enable	Input	0.02 pf

TABLE 11-3 I/O Signals (Instruction Cache Data RAM) *(Continued)*

Signal	Description	I/O	Cin/Cload 200 MHz
`bist_icu_addr`<13:3>	Address input for BIST	Input	0.02 pf
`bist_icu_din`<31:0>	Data input for BIST	Input	0.02 pf
`bist_icu_ram_we`<1:0>	Instruction cache RAM write enable for BIST	Input	0.02 pf
`bist_enable`	RAM enable for BIST	Input	0.02 pf
`test_mode`	Megacell test mode enable	Input	0.02 pf
`icram_dout`<63:0>	Data output	Output	0.35 pf

11.1.3 Timing

The specifications are listed below and in TABLE 11-4.

Process	Slow
Temperature	105 degrees C junction
Operating voltage	2.20v

TABLE 11-4 Timing Specifications (Instruction Cache Data RAM)

Symbol	Description	Condition	Spec 200 MHz
`Tcyc`	`clk` cycle time	Minimum	4.9 ns
`Tpwh`	`clk` high- level pulse width	Minimum	2.1 ns
`Tpwl`	`clk` low-level pulse width	Minimum	2.1 ns
`Tads`	`icu_addr` setup to `clk` high	Minimum	0.5 ns
`Tadh`	`icu_addr` hold from `clk` high	Minimum	0.3 ns
`Twes`	`icu_ram_we` setup to `clk` high	Minimum	0.3 ns
`Tweh`	`icu_ram_we` hold from `clk` high	Minimum	0.3 ns
`Tdis`	`icu_din` setup to `clk` high	Minimum	0.3 ns
`Tdih`	`icu_din` hold from `clk` high	Minimum	0.3 ns
`Tdxor`	`clk` high to read `icram_dout` valid	Maximum	2.0 ns[1]

TABLE 11-4 Timing Specifications (Instruction Cache Data RAM) *(Continued)*

Symbol	Description	Condition	Spec 200 MHz
`Tdow`	`clk` high to data written into the RAM	Maximum	1.8 ns[2]
`Tens`	Setup time for enable to powerdown	Minimum	1.0 ns
`Tenh`	Hold time for enable to powerdown	Minimum	0.3 ns

[1]Critical path.

[2]Timing requirement for writing data into the RAM (cells), measured from the rising edge of the `clk` from `icu_din`.

FIGURE 11-2 and FIGURE 11-3 illustrate the timing.

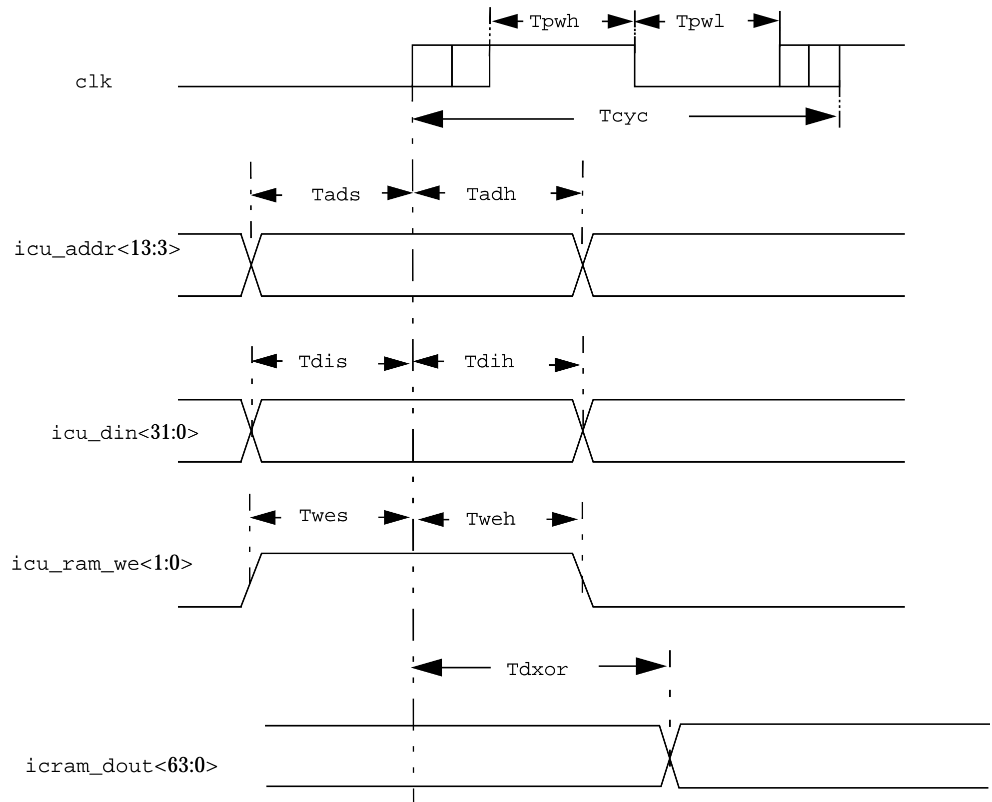


FIGURE 11-2 Timing Diagram A (Instruction Cache Data RAM)

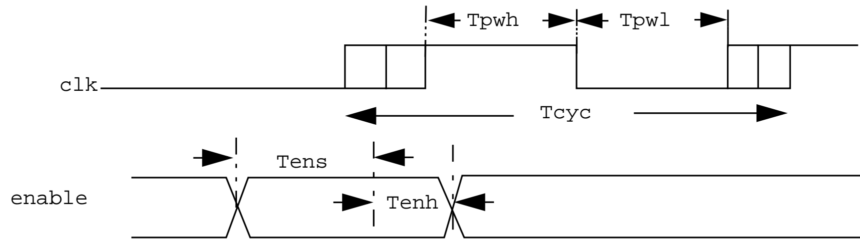


FIGURE 11-3 Timing Diagram B (Instruction Cache Data RAM)

11.1.4 RAM Redundancy

No redundancy is required for this block due to its relatively small RAMs.

11.1.5 Testing

Built-in Self-Test (BIST) is used for testing in this block.

The instruction cache RAM block is tested as embedded memory along with the rest of the chip silicon. There are no plans to test the block individually. A Verilog test bench and test pattern are available for verification.

Note the following:

- All timing specifications are for the Sun® SSLH corner operation conditions:
  - Process: Slow
  - Temperature: 105 degrees C junction
  - Vdd: 2.20v
- The test conditions for all timing specifications are:
  - Input Vil/Vih switching levels: 0.0v/2.20v
  - Input rise/fall times: 0.2 ns (10%~90% Vdd)
  - Input output timing reference levels: 2.20v/2

11.2 Instruction Cache Tag RAM

The instruction cache tag (itag.v) is logically organized as 1,024 entries by 19-bit RAM. Each entry corresponds to its associated line of four words in the instruction cache data RAM.

The instruction cache tag (itag) has one 10-bit address input (icu_tag_addr), one tag_in bus of 18 bits (icu_tag_in), and one valid bit (icu_tag_vld). A 19-bit comparator (cmp19_e) is included inside the block to provide faster hit or miss result. The 18-bit tag data output along with the a 1-bit valid output (itag_vld) from the RAM (tag RAM bit[0]) are compared internally against the 18 bits latched in tag data input (icu_tag_in) and a 1-bit logic "1" (at the 19th bit position) to produce the tag hit output (itag_hit).

The tag data output (itag_dout) and the valid bit output (itag_vld) should be buffered to ensure adequate drive strength to accommodate the heavy wire loading. Input for built-in self-test (bist*) is provided at the pinout. A test mode pin (test_mode) is used for the RAM to select taking either the normal data, address, and enable input (icu_tag_in, icu_tag_vld, icu_tag_we, and icu_tag_addr) or the BIST address, data, and enable input (bist_icu_tag_in, bist_icu_tag_vld, bist_icu_tag_we, and bist_icu_tag_addr).

The enable pin (enable) puts the instruction cache tag in disable mode when it is at low to conserve dynamic power consumption.

When the instruction cache tag is disabled, the value of the output data (itag_dout, itag_vld, and itag_hit) should remain unchanged as they were before, regardless of address input and write enable changes. All input should be synchronously latched at the block boundary.

TABLE 11-5 lists and describes test_mode and the enable signals.

TABLE 11-5 test_mode and Enable Signals (Instruction Cache Tag RAM)

Pin	enable **= 0**	enable **= 1**
test_mode = 0	The instruction tag RAM is disabled. The value of the output data is maintained as before regardless of the address input and write enable changes.	The instruction tag is enabled. The normal address, data, and write enable input (icu_tag_in, icu_tag_vld, icu_tag_addr, and icu_tag_we) are passed to the RAM.
test_mode = 1	Same as above.	The instruction tag is enabled and enters built-in test mode. The BIST address, data, and write enable input (bist_icu_tag_in, bist_icu_tag_vld, bist_icu_tag_addr, and bist_icu_tag_we) are passed to the RAM.

FIGURE 11-4 illustrates the instruction cache tag.

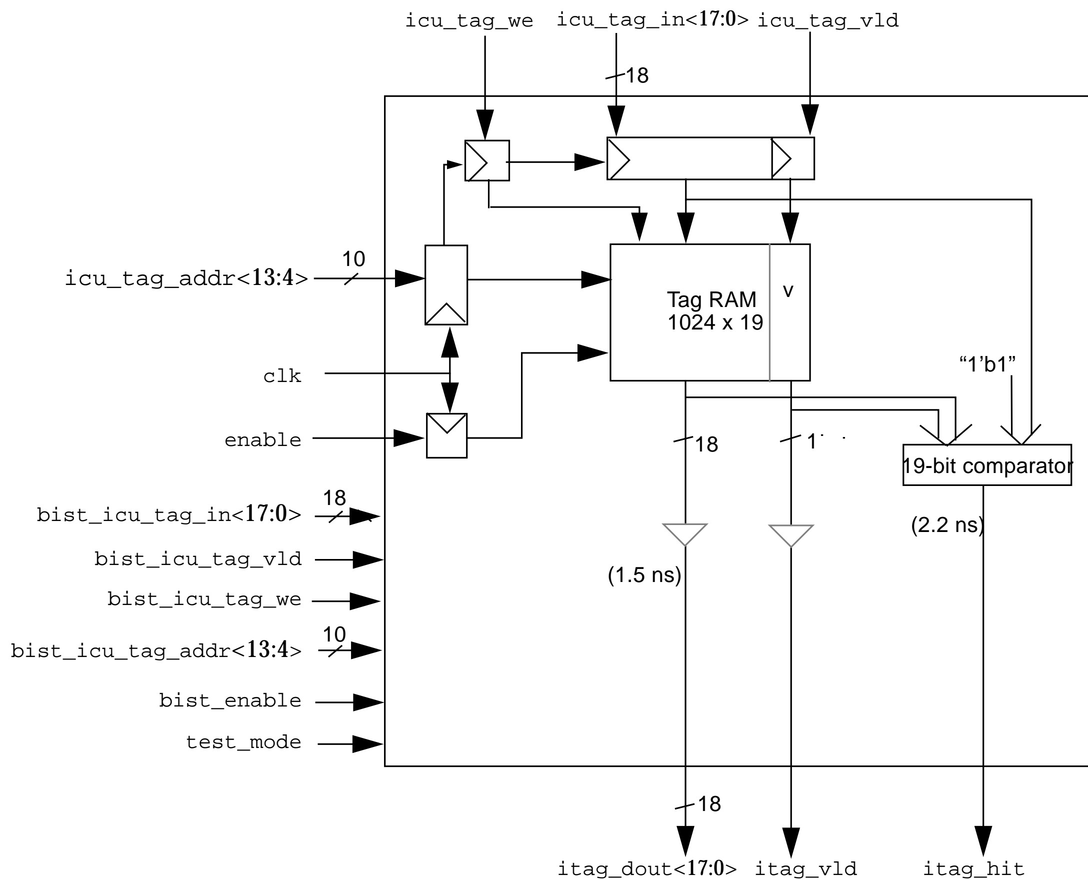


FIGURE 11-4 Instruction Cache Tag (itag)

11.2.1 I/O Pins

TABLE 11-6 and TABLE 11-7 list and describe the input and output pins, respectively.

TABLE 11-6 Input Pins (Instruction Cache Tag RAM)

Pin	Description
clk	Clock input pin.
icu_tag_addr<13:4>	Instruction tag address input. The word at the location addressed by icu_tag_addr is accessed for either reading or writing its contents. icu_tag_addr is 10 bits wide.
icu_tag_in<17:0>	Instruction tag data input. The value of icu_tag_din is written to the location addressed by icu_tag_addr.
icu_tag_vld	Instruction tag valid bit input. The value of icu_tag_vld is written to the location addressed by icu_tag_addr.
icu_tag_we	Instruction tag RAM write enable. The RAM performs a read-write from or to the location addressed by icu_tag_addr.
enable	Megacell enable. This signal is used to enable or disable the RAM to conserve dynamic power consumption when it is not being used. When enable = 0, the RAM is disabled and the value of all data output (itag_dout and itag_vld) is maintained as it was before regardless of address input and write enable changes. Normal operations are performed when enable = 1.
bist_icu_tag_addr<13:4>	Instruction tag address input for BIST. The word at the location addressed by bist_icu_tag_addr is accessed for either reading or writing.
bist_icu_tag_in<17:0>	Instruction tag data input for BIST. The value of bist_icu_din is written to the location addressed by bist_icu_tag_addr.
bist_icu_tag_vld	Instruction tag valid bit input for BIST. The value of bist_icu_tag_vld is written to the location addressed by bist_icu_tag_addr.
bist_icu_tag_we	Instruction tag RAM write enable for BIST. The RAM performs a read-write from or to the location addressed by bist_icu_tag_addr.

TABLE 11-6 Input Pins (Instruction Cache Tag RAM) *(Continued)*

Pin	Description
bist_enable	Megacell enable pin for BIST. When bist_enable = 0, the RAM is disabled and the value of all data output (itag_dout and itag_vld) is maintained as it was before regardless of the address input and write enable changes. Normal operations are performed when bist_enable = 1.
test_mode	Megacell test mode enable pin. When this pin is high, the megacell test mode is entered; the BIST address, data, and write enable input (bist_icu_tag_addr, bist_icu_tag_in, and bist_icu_tag_we) is taken to the RAM. When this pin is low, normal data, address, and write enable input (icu_tag_addr, icu_tag_in, and icu_tag_we) are passed to the RAM.

TABLE 11-7 Output Pins (Instruction Cache Tag RAM)

Pin	Description
itag_dout<17:0>	Instruction tag data output, accessed by the icu_tag_addr address (or bist_icu_tag_addr if test_mode = 1). This signal should be buffered up to drive the output loading specified in TABLE 11-6.
itag_vld	Instruction tag valid bit output accessed by icu_tag_addr (or bist_icu_tag_addr if test_mode = 1). This signal should be buffered up to drive the output loading specified in TABLE 11-6.
itag_hit	Instruction tag hit output, which is the result from an equal tag comparator that compares the RAM output data and the valid bit with the latched data input (icu_tag_in).

11.2.2 I/O Signals

TABLE 11-8 summarizes the input and output pins of the instruction tag and their input loading and output drive strengths.

TABLE 11-8 I/O Signals (Instruction Cache Tag RAM)

Signal	Description	I/O	Cin/Cload 200 MHz
clk	Clock input pin	Input	0.3 pf
icu_tag_addr<13:4>	Address input for read and write accesses	Input	0.02 pf
icu_tag_in<17:0>	Tag data input for write access	Input	0.02 pf
icu_tag_vld	Valid data input for write access	Input	0.02 pf
icu_tag_we	Instruction tag RAM write enable	Input	0.02 pf
enable	RAM enable	Input	0.02 pf
bist_icu_tag_addr<13:4>	Address input for BIST	Input	0.02 pf
bist_icu_tag_in<17:0>	Tag data input for BIST	Input	0.02 pf
bist_icu_tag_vld	Valid bit input for BIST	Input	0.02 pf
bist_icu_tag_we	Instruction tag RAM write enable for BIST	Input	0.02 pf
bist_enable	RAM enable for BIST	Input	0.02 pf
test_mode	Megacell test mode enable	Input	0.02 pf
itag_dout<17:0>	Tag data output	Output	0.3 pf
itag_vld	Tag valid bit output	Output	0.3 pf
itag_hit	Tag hit output	Output	0.5 pf

11.2.3 Timing

The specifications are listed below and in TABLE 11-9.

Process	Slow (G11p)
Temperature	105 degrees C junction
Operating voltage	2.20v

TABLE 11-9 Timing Specifications (Instruction Cache Tag RAM)

Symbol	Description	Condition	Spec 200 MHz
Tcyc	clk cycle time	Minimum	4.9 ns
Tpwh	clk high-level pulse width	Minimum	2.1 ns
Tpwl	clk low-level pulse width	Minimum	2.1 ns
Tads	icu_tag_addr set up to clk high	Minimum	0.5 ns
Tadh	icu_tag_addr hold from clk high	Minimum	0.3 ns
Twes	icu_tag_we set up to clk high	Minimum	0.3 ns
Tweh	iu_tag_we hold from clk high	Minimum	0.3 ns
Tdis	icu_tag_in/icu_tag_vld set up to clk high	Minimum	0.3 ns
Tdih	icu_tag_in/icu_tag_vld hold from clk high	Minimum	0.3 ns
Ttag	clk high to itag_dout<17:0>	Maximum	1.5 ns
Thit	clk high to itag_hit valid	Maximum	2.2 ns[1]
Tvldor	clk high to itag_vld	Maximum	1.5 ns
Tens	Setup time for enable to power down	Minimum	1.0 ns
Tenh	Hold time for enable to power down	Minimum	0.3 ns

[1]Critical path

FIGURE 11-5 and FIGURE 11-6 illustrate the timing.

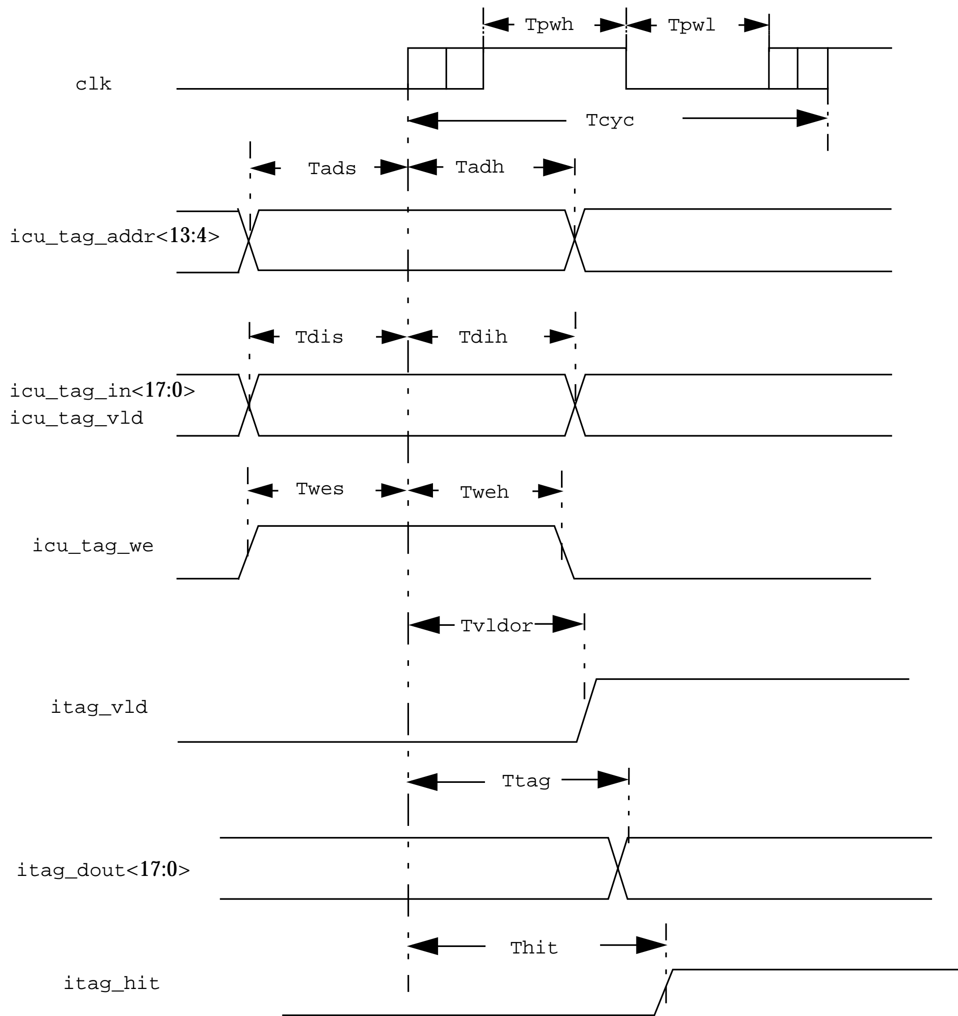


FIGURE 11-5 Timing Diagram A (Instruction Cache Tag RAM)

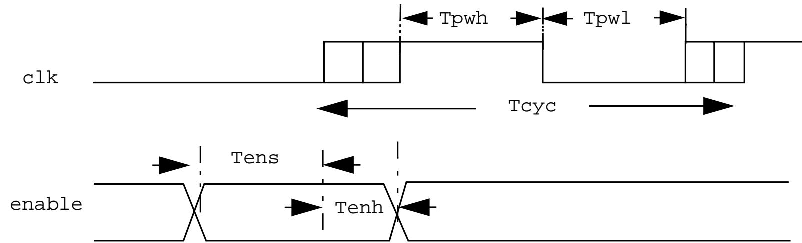


FIGURE 11-6 Timing Diagram B (Instruction Cache Tag RAM)

11.2.4 Implementation

The critical path of the instruction tag (itag) is from clk to tag RAM data output itag_dout through the 19-bit comparator to generate the itag_hit signal. The goal is to design and place the tag RAM and its comparator such that the itag_hit signal becomes valid within the timing specifications.

The icu_tag_in register is used for both RAM write and tag hit comparisons. Circuit design must ensure that the different kinds of logic do not interfere with each other. The nomenclature of the module (icram) and its input and output pins must be the same as that in FIGURE 11-4 on page 127.

11.2.5 RAM Redundancy

No redundancy is required for this block due to its relatively small RAMs.

11.2.6 Testing

Built-in Self-Test (BIST) is used for testing in this block.

The tag RAM block is tested as embedded memory along with the rest of the chip silicon. There are no plans to test the block individually. A Verilog test bench and test patterns are provided for verification.

Note the following:

- All timing specifications are for the Sun SSLH corner operation conditions:
  - Process: Slow
  - Temperature: 105 degrees C junction
  - Vdd: 2.20v

- The test conditions for all timing specifications are:
  - Input Vil/Vih switching levels: 0.0v/2.20v
  - Input rise/fall times: 0.2 ns (10%~90% Vdd)
  - Input output timing reference levels: 2.20v/2

11.3 Data Cache Data RAM

The data cache block is logically organized as two RAMs with 2,048 entries of 32 bits each. Since the data cache line size is four words, each line occupies four entries in the RAMs. The lines are organized such that alternate entries in the same cache line are mapped to different RAM blocks to allow simultaneous access to two consecutive words in a single line as well as simultaneous access to the same word in both ways of a set.

Way select is determined by the `addr[2]` bit, as follows:

- For even-word addresses, `addr[2]` = 0; RAM bank A contains data for Way 0 and bank B for Way 1.
- For odd-word addresses, `addr[2]` = 1; RAM bank A contains data for Way 1 and bank B for Way 0.

The RAMs are addressed with one 10-bit address bus and two 2-bit address buses.

A bypass function is built in to route incoming data directly to the output mux when a bypass signal is asserted. Bypass data override both data output ports. One 64-bit `data_out` bus is provided for reading out data from both RAM banks in the event of a writeback and flush cache. All input is synchronously latched in at block boundary.

A `test_mode` pin is used to select between normal input and BIST input to the RAM. A powerdown pin (`enable`) is asserted to shut off the sense amplifier to save power if feasible.

When the data cache is disabled, the value of the output data should remain unchanged regardless of address input and write enable changes.

TABLE 11-10 lists and describes `test_mode` and the enable signals.

TABLE 11-10 `test_mode` and Enable Signals (Data Cache Data RAM)

Pin	`enable` = 0	`enable` = 1
`test_mode` = 0	The data cache is disabled. The value of the output data is maintained as before regardless of address input and write enable changes. `data_out` is valid only if `bypass` is asserted.	The data cache is enabled. The normal address, data, and write enable input are passed to the RAM.
`test_mode` = 1	Same as above.	The data cache is enabled and enters built-in test mode. The BIST address, data, and write enable input are passed to the RAM.

FIGURE 11-7 illustrates the timing.

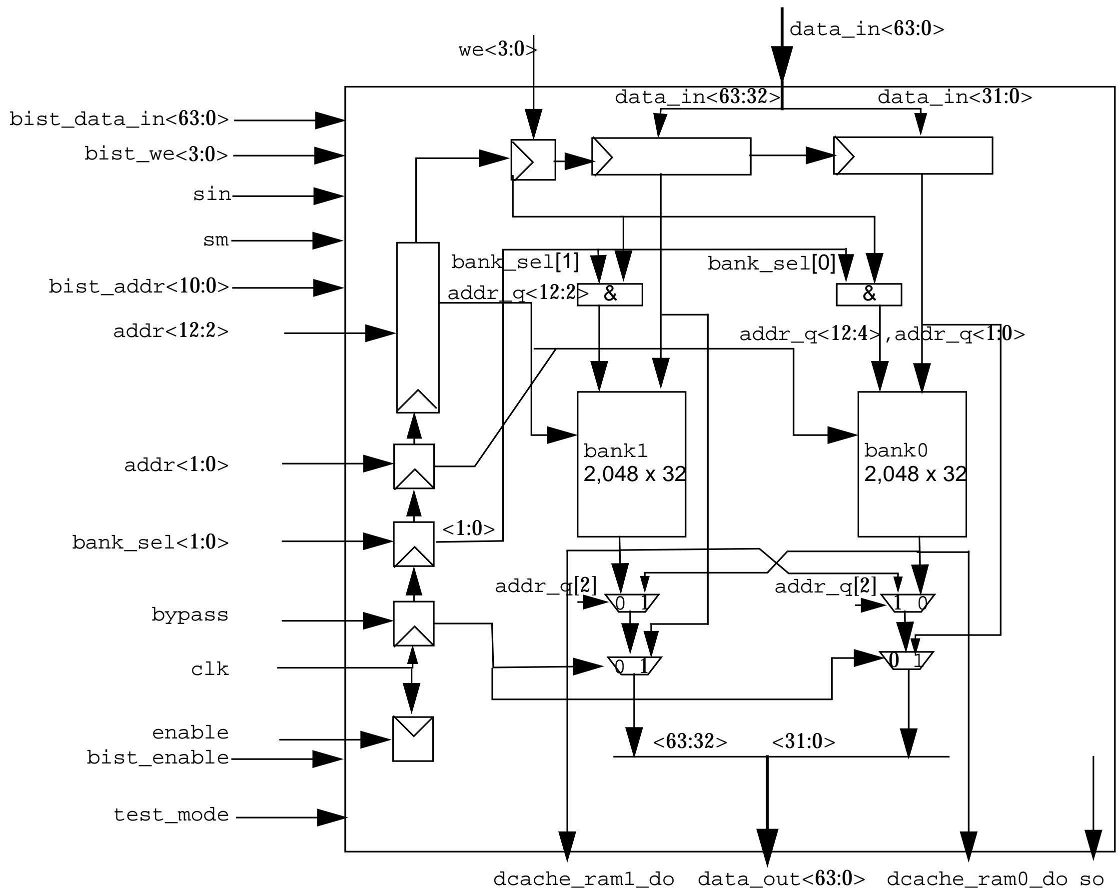


FIGURE 11-7 Data Cache Data RAM (`dcram`)

11.3.1 I/O Pins

TABLE 11-11 and TABLE 11-12 list and describe the input and output pins, respectively.

TABLE 11-11 Input Pins (Data Cache Data RAM)

Pin	Description
clk	Clock input pin
addr<12:2>	Address input for data RAM bank 1 reads and writes. Clocked into RAM with the rising edge of clk.
addr{<12:4><1:0>}	Address input for data RAM bank 0 reads and writes. Replaces addr<3:2>. Clocked into RAM with rising edge of clk.
data_in<63:32>	Cache data bank 1 input clocked into RAM with rising edge of clk
data_in<31:0>	Cache data bank 0 input clocked into RAM with rising edge of clk
we<3:0>	Byte write enable control input for cache RAM. Active high. Writes data_in data into selected set. Clocked in with rising edge of clk.
bank_sel<1:0>	Sets select enable control input asserted high selects, asserted low not selects. Clocked in with rising edge of clk.
bypass	Bypasses function enable input. The data on data_in bypasses the RAM and is available at the output. This functionality is valid even if the enable signal is 0.
enable	Powerdown mode input pin. Megacell enable. Active high. This signal is used to enable or disable the RAM to conserve dynamic power consumption when it is not being used. When enable=0, the RAM is disabled and the value of all data output is maintained as before regardless of address input and write enable changes.
bist_addr<10:0>	Pin for BIST. Same for the rest of the input. All bist_* signals are selected when the test_mode pin is high.
bist_data_in<63:0>	
bist_we<3:0>	
bist_enable	This signal has the same functionality as the enable signal when the test_mode pin is asserted (BIST mode).
test_mode	Test mode enable pin. When this pin is high, the megacell test mode is entered and the bist_* input is taken into the RAM. The normal address, data, and write_enable input are passed to the RAM when the test_mode = 0.
sin, sm	Scan in and scan mode signals

TABLE 11-12 Output Pins (Data Cache Data RAM)

Pin	Description
data_out<63:32>	Cache data output of set_1 (bank 1). Latched between read and write operations.
data_out<31:0>	Cache data output of set_0 (bank 0). Latched between read and write operations.
dcache_ram0_do<31:0>	Raw data out from Bank 0 RAM
dcache_ram1_do<31:0>	Raw data out from Bank 2 RAM
so	Scan-out signal

11.3.2 I/O Signals

TABLE 11-13 summarizes the input and output pins of the data cache block and their input loading and output drive strengths.

TABLE 11-13 I/O Signals (Data Cache Data RAM)

Signal	Description	I/O	Cin/Cload 200 MHz
clk	Clock input pin	Input	0.30 pf
addr<12:2>	Address input for RAM bank 1	Input	0.02 pf
addr<1:0>	Address input for RAM bank 0 in place of addr<3:2>	Input	0.02 pf
data_in<63:0>	data_bus input for data cache writes	Input	0.02 pf
we<3:0>	Data cache RAM write enable signal, byte writable	Input	0.02 pf
bank_sel<1:0>	Selects which bank to write	Input	0.02 pf
bypass	Bypasses the function enable signal	Input	0.02 pf
enable	Powerdown mode input active low	Input	0.02 pf
test_mode	BIST enable input (test input is selected)	Input	0.02 pf
bist_addr<10:0>	bist_addr<10:0> pin for BIST	Input	0.02 pf
bist_data_in<63:0>	data_in<63:0> for BIST	Input	0.02 pf
bist_we<3:0>	we<3:0> for BIST	Input	0.02 pf
bist_enable	enable signal for BIST	Input	0.02 pf

TABLE 11-13 I/O Signals (Data Cache Data RAM) *(Continued)*

Signal	Description	I/O	Cin/Cload 200 MHz
data_out<63:0>	Data output	Output	0.25 pf
dcache_ram0_do<31:0>	Bank 0 data out	Output	0.25 pf
dcache_ram1_do<31:0>	Bank 1 data out	Output	0.25 pf

11.3.3 Timing

The specifications are listed below and in FIGURE 11-14.

Process	Slow
Temperature	105 degrees C junction
Operating voltage	2.20v

TABLE 11-14 Timing Specifications (Data Cache Data RAM)

Symbol	Description	Condition	Spec 200 MHz
Tcyc	clk cycle time	Minimum	4.9 ns
Tpwh	clk high-level pulse width	Minimum	2.1 ns
Tpwl	clk low-level pulse width	Minimum	2.1 ns
Tads	Address setup to clk high	Minimum	0.5 ns
Tadh	Address hold from clk high	Minimum	0.3 ns
Tbps	Bypass setup to clk high	Minimum	0.4 ns
Tbph	Bypass hold from clk high	Minimum	0.3 ns
Tstsls	bank_sel setup to clk high	Minimum	0.4 ns
Tstslh	bank_sel hold from clk high	Minimum	0.3 ns
Twes	we setup to clk high	Minimum	0.5 ns
Tweh	we hold from clk high	Minimum	0.3 ns
Tdis	di setup to clk high	Minimum	0.25 ns
Tdih	di hold from clk high	Minimum	0.4 ns
Tidor	clk high to internal read data valid	Maximum	1.9ns

TABLE 11-14 Timing Specifications (Data Cache Data RAM) *(Continued)*

Symbol	Description	Condition	Spec 200 MHz
`Tdxor`	`clk` high to `data_out` valid	Maximum	2.2 ns[1]
`Tens`	Setup time for `enable` to power down the SRAM	Minimum	1.0 ns
`Tenh`	Hold time for `enable` to power down the SRAM	Minimum	0.3 ns

[1]Cache RAM data output; critical path

FIGURE 11-8 and FIGURE 11-9 illustrate the timing.

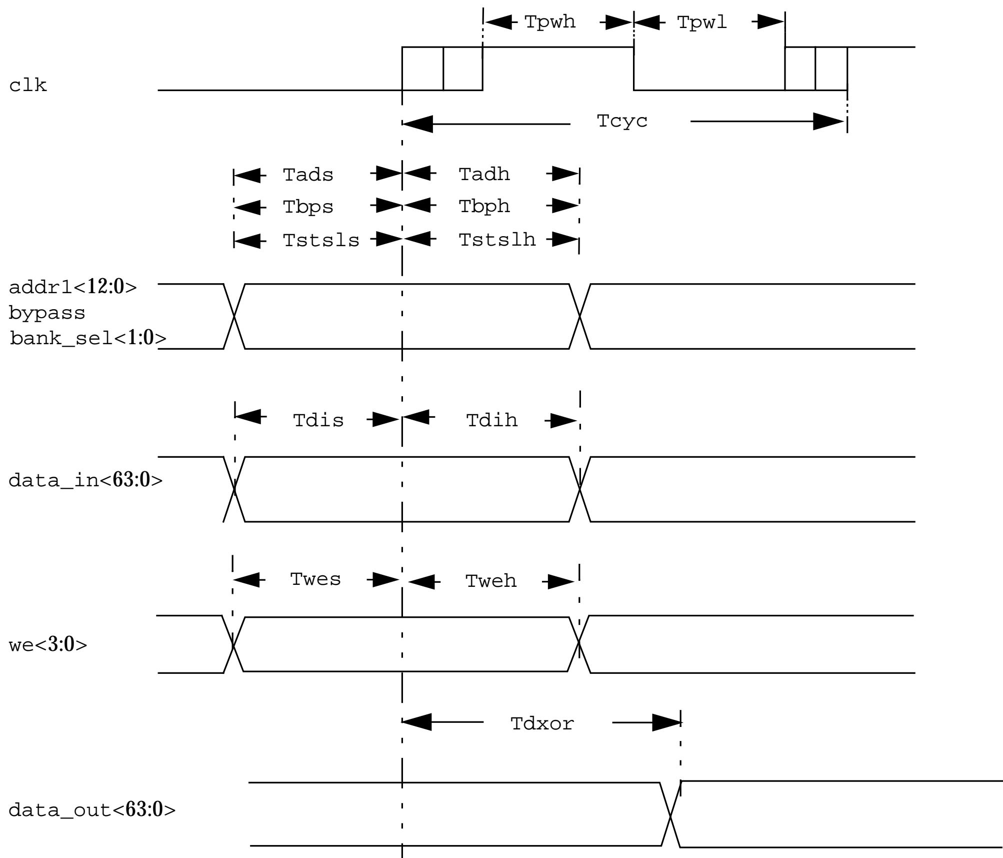


FIGURE 11-8 Timing Diagram A (Data Cache Data RAM)

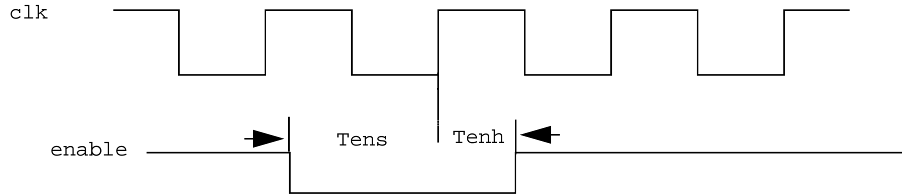


FIGURE 11-9 Timing Diagram B (Data Cache Data RAM)

11.3.4 Implementation

The critical path of this block is `data_out` through the bypass and swap muxes. Consider the layout of this group first to ensure that all 32-bit data have even delay numbers.

RAM 1 and RAM 0 should be symmetrical to ensure that both banks have the same delay numbers.

Note – `data_in` registers are used for both RAM writes and for data bypassing.

11.3.5 RAM Redundancy

No redundancy is required for this block due to its relatively small RAMs.

11.3.6 Testing

Built-in Self-Test (BIST) is used for testing in this block.

The data cache RAM block is tested as embedded memory along with the rest of the chip silicon. There are no plans to test the block individually. A Verilog test bench and test pattern are available for verification.

Note the following:

- All timing specifications are for the Sun SSLH corner operation conditions:
  - Process: Slow
  - Temperature: 105 degrees C junction
  - Vdd: 2.20v

- The test conditions for all timing specifications are:
  - Input Vil/Vih switching levels: 0.0v/2.20v
  - Input rise/fall times: 0.2 ns (10%~90% Vdd)
  - Input output timing reference levels: 2.20v/2

11.4 Data Cache Tag RAM

The data cache tag block is logically organized as a two-way associative set, with each tag RAM set consisting of 512 entries by 19 bits plus a status RAM of 512 entries by 5 bits. Each entry corresponds to its associated set of one line of four words in the data cache.

The status RAM is dual ported; both ports can be accessed simultaneously for reads and writes.

All input to RAMs is internally latched to have synchronized reads and writes.

One 9-bit address bus drives all three RAMs for read. A second 9-bit address bus drives the status RAM's write port to update its contents. One `tag_data_in` bus of 19 bits is used for both sets of tag RAM for data input in write tag mode and for tag comparison in tag read mode. All input is registered internally.

A `set_select` line selects a set to write to and to read out the `tag_data` for diagnostic reads and `write_back` mode. Internally, a 2:1 mux is built in to select `tag_out` from one of the two sets of tag RAMs.

Two 20-bit comparators are included in the block to provide faster hit or miss results. Both `tag_data` outputs are compared internally against the latched `cmp_addr_in` bits along with valid 1 and valid 0 to get fast hit 1 and hit 0. An MSB of the compare operand is used for valid bit comparisons—it is compared with logic 1 to determine hit or miss of each set. The hit 1 and hit 0 output is internally buffered individually to give strong drives.

Status RAM read address1[`addr`] shares the same address as tag RAMs internally and uses address2 [`stat_addr`] as its write port address. These two ports are totally independent of each other and can be accessed simultaneously.

Reading and writing into the same location (and the same field within the address) can result in undefined read data. However, you can be writing to one field (dirty) and reading from the other (valid) *at the same address at the same time*. The written data are valid. The status RAM has only one read port and one write port.

All the input is synchronously latched in at a block boundary. The same enable signal (pin) drives the status RAM and both banks of the tag RAM. The enable signal is latched on the falling edge of the clock before going to the status RAM. However, the enable pin is clocked with a flip-flop before going to the tag RAMs. This pin is asserted to shut off the sense amplifier to save power when needed.

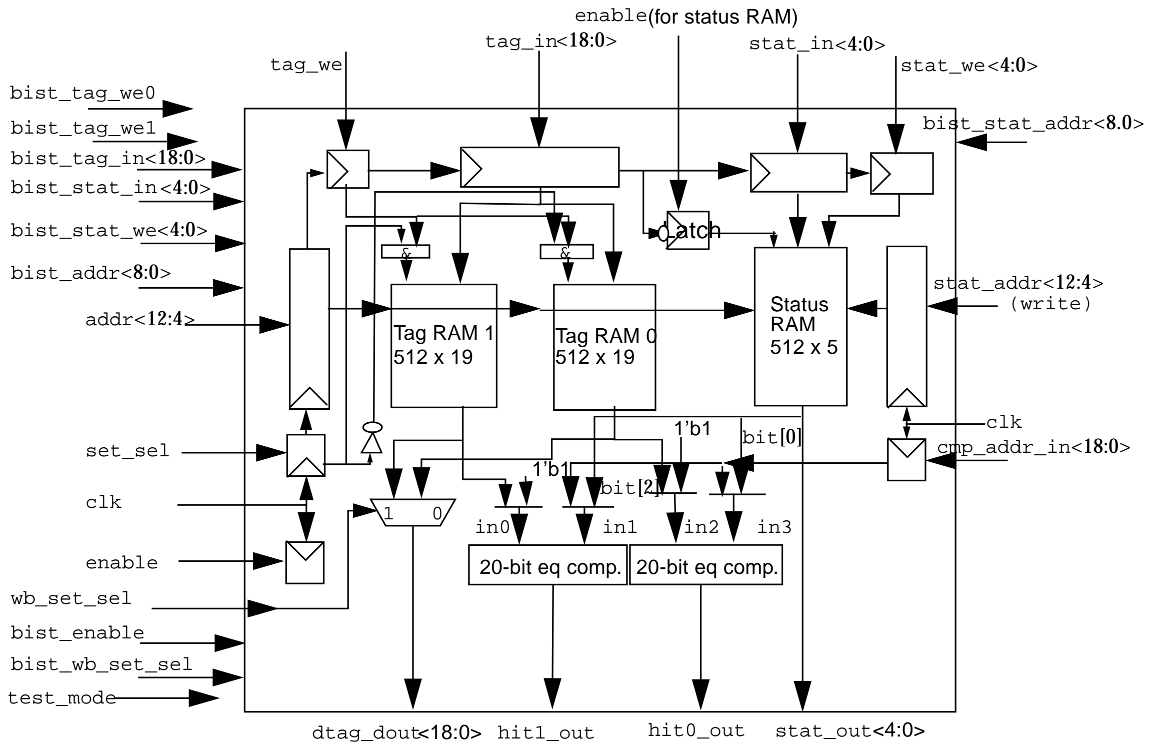
When the data cache is disabled, the value of output data remains unchanged as before regardless of address input and write enable changes.

TABLE 11-15 lists and describes `test_mode` and the enable signals.

TABLE 11-15 `test_mode` and Enable Signals (Data Cache Tag RAM)

Pin	`enable` **= 0**	`enable` **= 1**
`test_mode` = 0	The data tag is disabled. The value of the output data is maintained as before regardless of address input and write enable changes.	The data tag is enabled. The normal address, data, and write enable input are passed to the RAM.
`test_mode` = 1	Same as above.	The data tag is enabled and enters built-in test mode. The BIST address, data, and write enable input are passed to the RAM.

FIGURE 11-10 illustrates the data cache tag RAM.



in0<19:0> = {dou1<18:0>, 1'bl}
in1<19:0> = {cmp_addr_q<18:0>, stat_out[2]}
in2<19:0> = {dou1<18:0>, 1'bl}
in3<19:0> = {comp_addr_q<18:0>, stat_out[2]}

FIGURE 11-10 Data Cache Tag RAM (`dtag`)

11.4.1 I/O Pins

TABLE 11-16 and TABLE 11-17 list and describe the input and output pins, respectively.

TABLE 11-16 Input Pins (Data Cache Tag RAM)

Pin	Description
clk	Clock input pin
addr<12:4>	Address input for tag RAM reads and writes and status reads. clocked into RAM with rising edge of clk.
stat_addr<12:4>	Address input for status RAM writes. Clocked into RAM with rising edge of clk.
tag_in<18:0>	Tag data input clocked into RAM with rising edge of clk.
cmp_addr_in<18:0>	Address to compare against for the generation of hit or miss signals. Clocked into RAM with the rising edge of clk.
stat_in<4:0>	Status data input clocked into RAM with the rising edge of clk.
set_sel	Sets select enable control input to select which tag RAM to write to. Asserted high, selects set 1; asserted low, selects set0.
wb_set_sel	Selects which tag RAM output comes on the dtag_dout pin. High selects set 1 tag; low selects set 0 tag.
tag_we	Write enable control input for tag RAM. Active high. Writes tag_in data into selected set.
stat_we<4:0>	Write enable control input for status RAM. Active high. Writes status_in data into status RAM. Each bit is a write_enable for the corresponding stat_in bits.
enable	Powerdown mode input pin. Megacell enable. High for regular operation; low for powerdown mode. This signal is used to enable or disable the RAM to conserve dynamic power consumption when it is not being used. When enable = 0, the RAM is disabled and the value of all data output is maintained as before regardless of address input and write enable changes. For the status RAM, the enable is taken through a negative edge-triggered latch. The clock is the input to the latch_enable.
bist_addr<8:0>	Pins for BIST. Same for the rest of input pins. All bist_* signals are selected when the test_mode pin is asserted.
bist_stat_addr<8:0>	
bist_tag_in<18:0>	
bist_stat_in<4:0>	
bist_enable	
bist_wb_set_sel	

TABLE 11-16 Input Pins (Data Cache Tag RAM) *(Continued)*

Pin	Description
bist_tag_we0	
bist_tag_we1	
bist_stat_we<4:0>	
test_mode	Selects between regular and BIST mode signals. When this pin is high, the megacell test mode is entered and the bist_* input is taken into the RAM. The normal address, data, and write_enable input are passed to the RAM when test_mode = 0.

TABLE 11-17 Output Pins (Data Cache Tag RAM)

Pin	Description
dtag_dout<18:0>	Tag data output of selected set
stat_out<4:0>	Status data output
hit1_out	Output of tag 1 equals to compare_address input; active high signal
hit0_out	Output of tag 0 equals to compare_address input; active high signal
bist_tag_dout1	BIST data out
bist_tag_dout2	BIST data out

11.4.2 I/O Signals

TABLE 11-18 summarizes the input and output pins of the data cache tag block and their input loading and output drive strengths.

TABLE 11-18 I/O Signals (Data Cache Tag RAM)

Signal	Description	I/O	Cin/Cload 200 MHz
clk	Clock input pin	Input	0.34 pf
addr<12:4>	Address input for tag reads and status reads	Input	0.02 pf
stat_addr<12:4>	Address input for status writes	Input	0.02 pf
tag_in<18:0>	tag_data input for writes and comparisons	Input	0.02 pf
cmp_addr_in<18:0>	Address for comparison with tag output	Input	0.02 pf
stat_in<4:0>	tag_status data input	Input	0.02 pf

TABLE 11-18 I/O Signals (Data Cache Tag RAM) *(Continued)*

Signal	Description	I/O	Cin/Cload 200 MHz
tag_we	Tag RAM write enable signal	Input	0.02 pf
stat_we<4:0>	Status RAM write enable signal	Input	0.02 pf
set_sel	Sets select signal	Input	0.02 pf
wb_set_sel	Selects tag address to wb_address_reg	Input	0.02 pf
enable	Powerdown mode input active low for powerdown	Input	0.02 pf
dtag_dout<18:0>	Sets selected tag data output	Output	0.15 pf
stat_out<4:0>	Status data output	Output	0.45 pf
hit1_out	Tag1 compare equals active high	Output	0.50 pf
hit0_out	Tag0 compare equals active high	Output	0.50 pf
test_mode	BIST enable input. Test input is selected when high.	Input	0.02 pf
bist_tag_in<18:0>	Pin for BIST. Same for the following signals.	Input	0.02 pf
bist_stat_in<4:0>	stat_in<4:0> for BIST	Input	0.02 pf
bist_tag_we0	tag_we for BIST	Input	0.02 pf
bist_tag_we1	tag_we for BIST	Input	0.02 pf
bist_stat_we<4:0>	stat_we<4:0> for BIST	Input	0.02 pf
bist_enable	enable for BIST	Input	0.02 pf
bist_wb_set_sel	wb_set_sel for BIST	Input	0.02 pf
bist_addr<8:0>	addr<12:4> for BIST	Input	0.02 pf
bist_stat_addr<8:0>	stat_addr<12:4> for BIST	Input	0.02 pf

11.4.3 Timing

The specifications are listed below and in FIGURE 11-19.

Process	Slow
Temperature	105 degrees C junction
Operating voltage	2.20v

TABLE 11-19 Timing Specifications (Data Cache Tag RAM)

Symbol	Description	Condition	Spec 200 MHz
Tcyc	clk cycle time	Minimum	4.9 ns
Tpwh	clk high-level pulse width	Minimum	2.1 ns
Tpwl	clk low-level pulse width	Minimum	2.1 ns
Tads	Address setup to clk high	Minimum	0.5 ns
Tadh	Address hold from clk high	Minimum	0.3 ns
Tstsls	set_sel setup to clk high	Minimum	0.4 ns
Tstslh	set_sel hold from clk high	Minimum	0.3 ns
Twes	we setup to clk high	Minimum	0.3 ns
Tweh	we hold from clk high	Minimum	0.3 ns
Tdis	di setup to clk high	Minimum	0.25 ns
Tdih	di hold from clk high	Minimum	0.4 ns
Ttgor	clk high to internal tag read data valid	Maximum	1.6 ns[1]
Ttag	clk high to tag_out valid	Maximum	2.3 ns
Thit	clk high to hit1_out/hit0_out valid	Maximum	2.3 ns[2]
Tstor	clk high to status_out valid	Maximum	1.8 ns[3]
Twb_set	clk high to wb_set_sel valid	Maximum	2.0 ns
Tcmp_addr_val	clk high to cmp_addr_in valid	Maximum	1.2 ns
Tens	enable setup to clk high	Minimum	1.0 ns
Tenh	enable hold from clk high	Minimum	0.3 ns
Tcmps	Setup time for cmp_addr_in	Minimum	0.25 ns
Tcmph	Hold time for cmp_addr_in	Minimum	0.4 ns

[1]Internal tag RAM data output valid point. Use this number only as a reference to measure the tag RAM speed.
[2]Critical path. This path should have a higher priority over other paths.
[3]Internally, valid 1 and valid 0 should be available by 1.85 ns at comparator input.

FIGURE 11-11 and FIGURE 11-12 illustrate the timing.

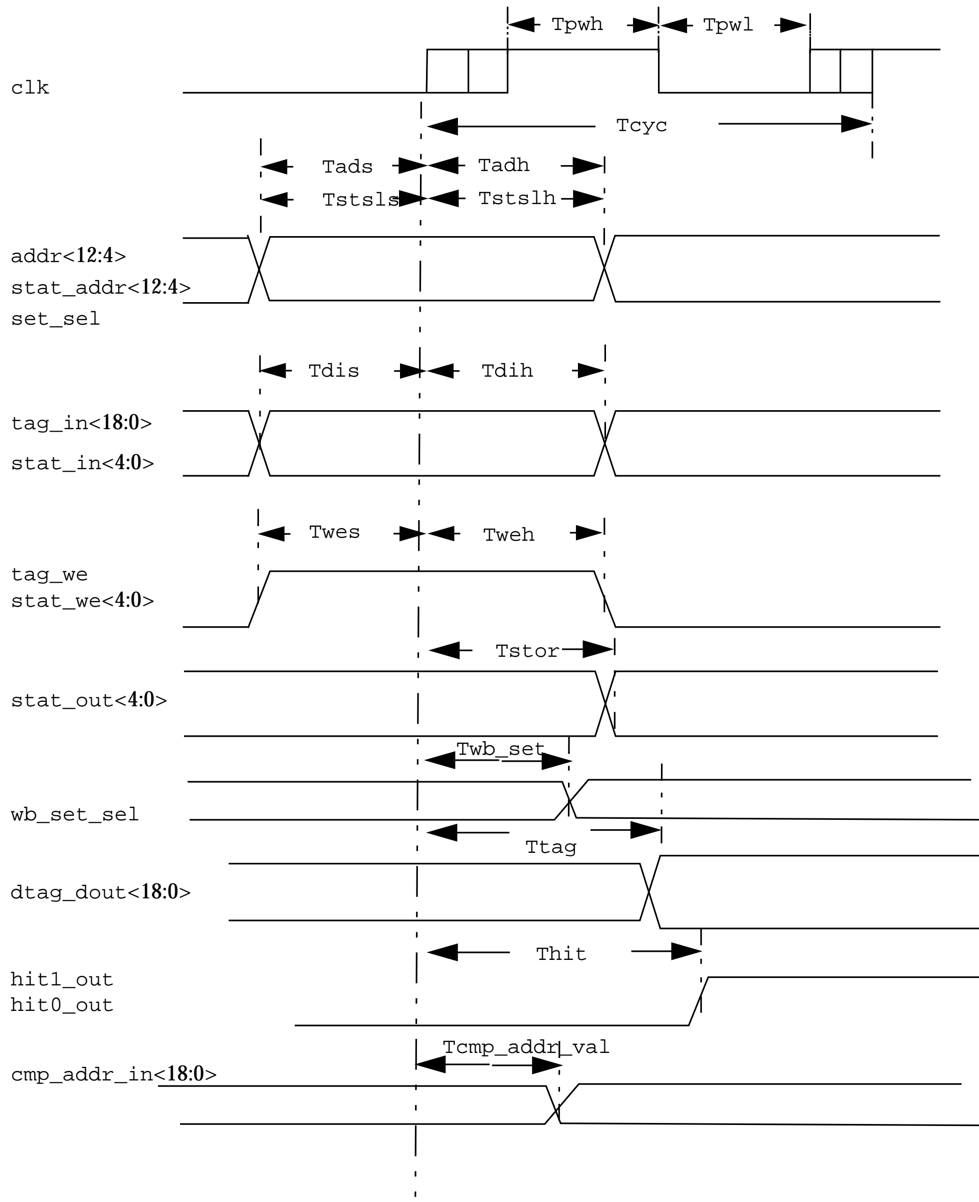


FIGURE 11-11 Timing Diagram A (Data Cache Tag RAM)

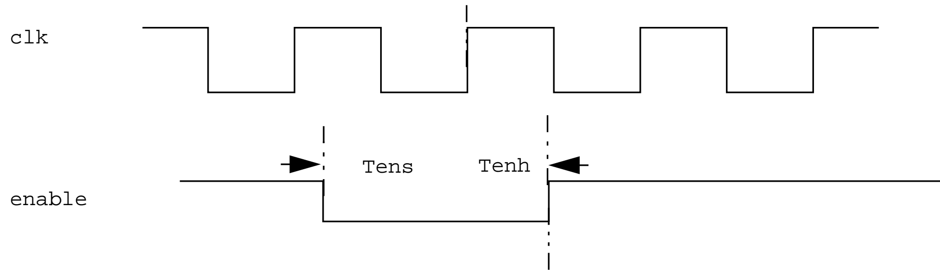


FIGURE 11-12 Timing Diagram B (Data Cache Tag RAM)

11.4.4 Implementation

The critical path of the data cache tag block is through the tag RAM and the 20-bit comparator to generate `hit1_out` or `hit0_out` signals. The goal is to design and place the tag RAMs and the comparator of both sets such that both `hit1_out` and `hit0_out` signals can meet the timing specifications.

11.4.5 RAM Redundancy

No redundancy is required for this block due to its relatively small RAMs.

11.4.6 Testing

Built-in Self-Test (BIST) is used for testing in this block.

Note the following:

- All timing specifications are for the Sun SSLH corner operation conditions:
  - Process: Slow
  - Temperature: 105 degrees C junction
  - Vdd: 2.20v
- The test conditions for all timing specifications are:
  - Input Vil/Vih switching levels: 0.0v/2.20v
  - Input rise/fall times: 0.2 ns (10%~90% Vdd)
  - Input output timing reference levels: 2.20v/2

11.5 Stack Cache RAM

The stack cache RAM (rf.v) is a five-port asynchronous RAM or register file. All five ports, three for reads and two for writes, can be accessed independently.

The stack cache RAM (rf) is used for the IU, which gets data from the stack and stores them back into the stack through two read ports and one write port. Writing into the same location of reading has a higher priority; write data are valid and read data are undefined during writes. Writing into the same location through both write ports at the same time is not defined. FIGURE 11-13 illustrates the interface.

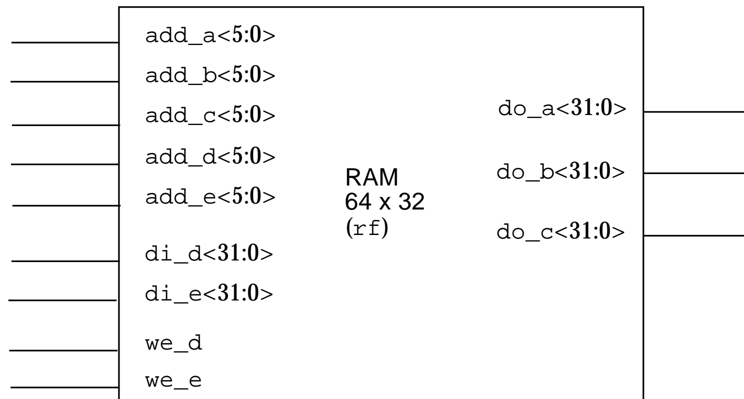


FIGURE 11-13 Stack Cache RAM Interface

11.5.1 I/O Pins

TABLE 11-20 and TABLE 11-21 list and describe the input and output pins, respectively.

TABLE 11-20 Input Pins (Stack Cache RAM)

Pin	Description
add_a<5:0>	Address input, port A (read only)
add_b<5:0>	Address input, port B (read only)
add_c<5:0>	Address input, port C (read only)
add_d<5:0>	Address input, port D (write only)
add_e<5:0>	Address input, port E (write only)

TABLE 11-20 Input Pins (Stack Cache RAM) *(Continued)*

Pin	Description
di_d<31:0>	Data input, port D
di_e<31:0>	Data input, port E
we_d	Write enable control input, port D, active high
we_e	Write enable control input, port E, active high

TABLE 11-21 Output Pins (Stack Cache RAM)

Pin	Description
do_a<31:0>	Data output, port A
do_b<31:0>	Data output, port B
do_c<31:0>	Data output, port C

11.5.2 I/O Signals

TABLE 11-22 summarizes the input and output pins of the stack cache RAM and their input loading and output drive strengths.

TABLE 11-22 I/O Signals (Stack Cache RAM)

Signal	Description	I/O	Cin/ Cload 200 MHz
add_a<5:0>	Address input for read port A	Input	0.02 pf
add_b<5:0>	Address input for read port B	Input	0.02 pf
add_c<5:0>	Address input for read port C	Input	0.02 pf
add_d<5:0>	Address input for write port D	Input	0.02 pf
add_e<5:0>	Address input for write port E	Input	0.02 pf
di_d<31:0>	Data input for write port D	Input	0.02 pf
di_e<31:0>	Data input for write port E	Input	0.02 pf
we_d	Write enable input for port D	Input	0.06 pf
we_e	Write enable input for port E	Input	0.06 pf
do_a<31:0>	Read port A data output	Output	0.25 pf

TABLE 11-22 I/O Signals (Stack Cache RAM) *(Continued)*

Signal	Description	I/O	Cin/ Cload 200 MHz
do_b<31:0>	Read port B data output	Output	0.25 pf
do_c<31:0>	Read port C data output	Output	0.25 pf

11.5.3 Timing

The specifications are listed below and in TABLE 11-23.

Process	Slow
Temperature	105 degrees C junction
Operating voltage	2.20v

TABLE 11-23 Timing Specifications (Stack Cache RAM)

Symbol	Description	Condition	Spec 200 MHz
Trc	Read cycle time	Minimum	2.9 ns
Taa	Read address access time	Maximum	1.7 ns[1]
Toh	Output hold time from address change	Minimum	0.2 ns
Twc	Write cycle time	Minimum	3.2 ns
Taw	Address valid to end of write	Minimum	1.1 ns
Tasw	Address valid to start of write	Minimum	0.2 ns
Twp	Write minimum pulse width	Minimum	0.9 ns
Twr	Write recovery time	Minimum	0.1 ns
Tdw	Data valid to end of write	Maximum	0.6 ns
Tdh	Data hold time	Maximum	0.2 ns
Twd	Data output delay time from write	Maximum	2.6 ns[2]
Twh	Data output hold time from write	Maximum	1.7 ns
Twdd	Data input to data output access time	Maximum	2.1 ns[2]
Twdh	Data input to data output hold time	Minimum	1.4 ns

[1]Critical path. This path should have the highest timing priority over other paths.
[2]No write-throughs.

FIGURE 11-14 and FIGURE 11-15 illustrate the timing.

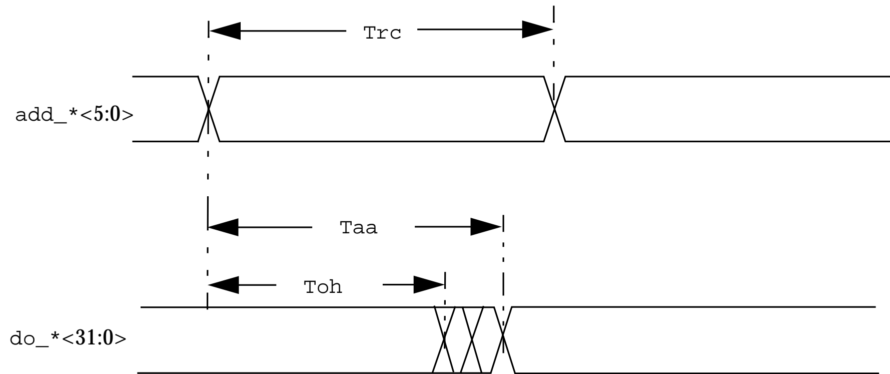


FIGURE 11-14 Timing Diagram A (Read Cycle, Stack Cache RAM)

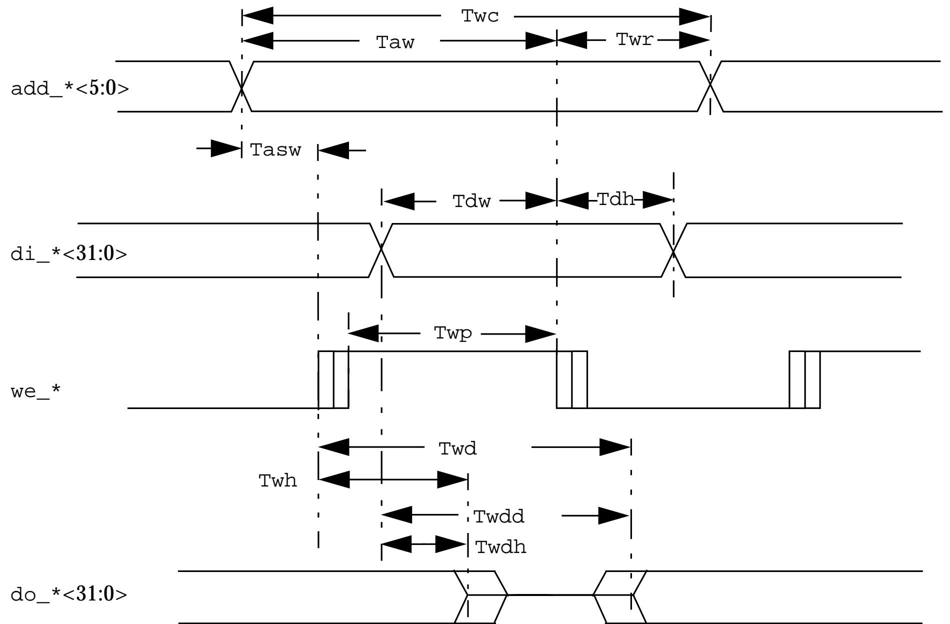


FIGURE 11-15 Timing Diagram B (Write Cycle, Stack Cache RAM)

11.5.4 Testing

All timing specifications are for the Sun SSLH corner operation conditions:

- Process: Slow
- Temperature: 105 degrees C junction
- Vdd: 2.20v

The test conditions for all timing specifications are:

- Input Vil/Vih switching levels: 0.0v/2.20v
- Input rise/fall times: 0.2 ns (10%~90% Vdd)
- Input output timing reference levels: 2.20v/2

Writing into the same location through both ports at the same time is not defined.

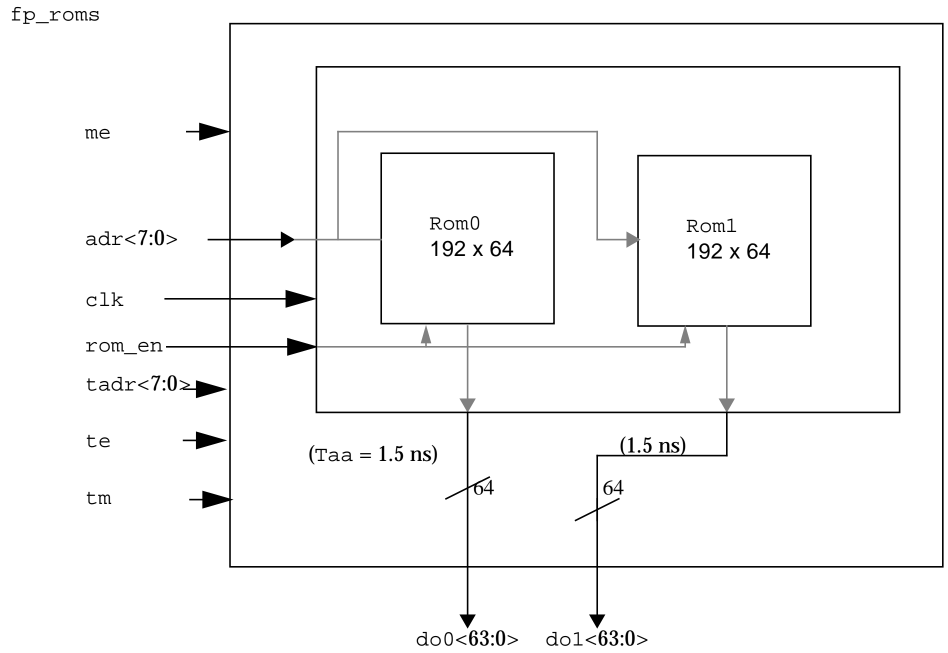
11.6 Floating Point Unit (FPU) ROM

The `fp_roms` module (`fp_roms.v`) contains two ROMs (192 x 64 each). Both ROMs have the same design and configurations, but they store different data.

The microcode ROM is a low-power, high-density, synchronous, read-only memory.

An 8-bit address selects one of the memory entries and the value is output on the ROM output. Input registers should be built in to latch the address on the rising edge of the clock.

FIGURE 11-16 illustrates the FPU ROM.



Generation of internal me (rom enable) and enable

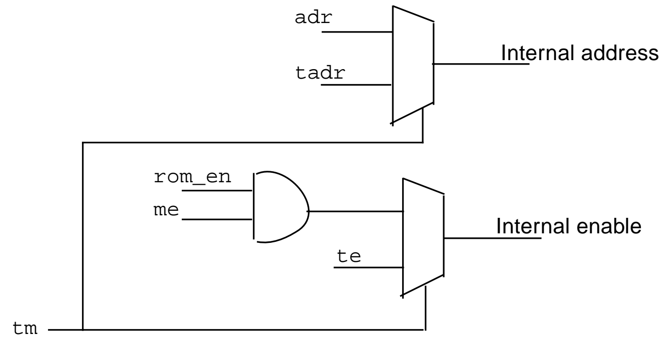


FIGURE 11-16 Floating Point Unit (FPU) ROM

FIGURE 11-17 illustrates the FPU ROM interface.

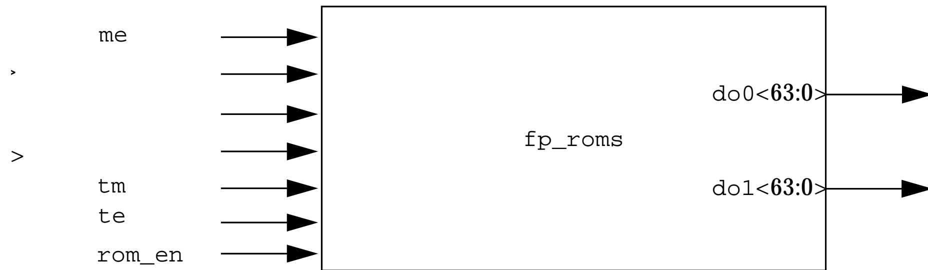


FIGURE 11-17 FPU ROM Interface

11.6.1 I/O Pins

TABLE 11-24 and TABLE 11-25 list and describe the input and output pins, respectively.

TABLE 11-24 Input Pins (FPU ROM)

Pin	Description
`adr<7:0>`	Address input.
`clk`	Clock input.
`tadr<7:0>`	Test address input.
`tm`	Test model; overrides `rom_en` and `me` input.
`te`	Test enable input.
`rom_en`	When deasserted (`rom_en` = 0), the output `do0` and `do1` remain unchanged regardless of address changes.
`me`	When deasserted (`me` = 0), the output `do0` and `do1` remain unchanged regardless of address changes.

TABLE 11-25 Output Pins (FPU ROM)

Pin	Description
`do0<63:0>`	Data output
`do1<63:0>`	Data output

TABLE 11-26 is the truth table for the ROM output.

TABLE 11-26 Truth Table for ROM Output

te	tm	rom_en	me	ROM Output
X	0	0	0	Disabled
X	0	0	1	Disabled
X	0	1	0	Disabled
X	0	1	1	Enabled
1	1	X	X	Enabled
0	1	X	X	Disabled

11.6.2 I/O Signals

TABLE 11-27 summarizes the input and output pins of the FPU ROM and their input loading and output drive strengths.

TABLE 11-27 I/O Signals (FPU ROM)

Signal	Description	Type	C_{in}/C_{load}
adr<7:0>	Address input for ROM.	Input	0.074 pf
clk	Clock input.	Input	0.076 pf
do0<63:0>	Addressed data output from rom0.	Output	0.25 pf
do1<63:0>	Addressed data output from rom1.	Output	0.25 pf
me	When deasserted (me = 0), the do0 and do1 output remains unchanged.	Input	0.029 pf
rom_en	When deasserted (when rom_en = 0), do0 and do1 output remains unchanged.	Input	0.029 pf
te	Test enable input.	Input	0.022 pf
tadr<7:0>	Test address input.	Input	0.008 pf
tm	Test mode input.	Input	0.015 pf

11.6.3 Timing

The specifications are listed below and in TABLE 11-28.

Process	Slow
Temperature	105 degrees C junction
Operating voltage	2.20v

TABLE 11-28 Timing Specifications (FPU ROM)

Symbol	Description	Condition	Spec 200 MHz
`Trc`	Read cycle time	Minimum	4.9 ns
`Tpwh`	`clk` high-level pulse width	Minimum	2.1 ns
`Tpwl`	`clk` low-level pulse width	Minimum	2.1 ns
`Tads`	Read address setup time	Maximum	0.35 ns
`Tadh`	Read address hold time	Minimum	0.2 ns
`Taa`	Data read access time from `clk` high	Maximum	1.5 ns
`Toh`	Output hold time from `clk` high	Minimum	0.5 ns
`Tmes`	Setup time for `rom_en` and `me` signals	Maximum	0.4 ns
`Tmeh`	Hold time for `rom_en` and `me` signals	Minimum	0.3 ns
`Ttes`	Setup time for `te`	Maximum	0.6 ns
`Tteh`	Hold time for `te`	Minimum	0.65 ns
`Ttms`	Setup time for `tm`	Maximum	0.6 ns
`Ttmh`	Hold time for `tm`	Minimum	0.65 ns

Note – As long as `Tmeh` and `Tmes` are met, `rom_en` and `me` can be asserted and deasserted in adjacent cycles, as shown in FIGURE 11-18.

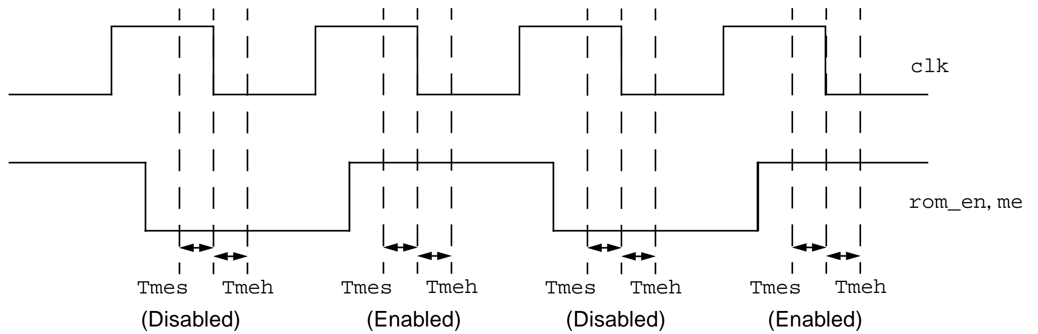


FIGURE 11-18 Assertion and Deassertion of `rom_en` and `me`

FIGURE 11-19 and FIGURE 11-20 illustrate the timing.

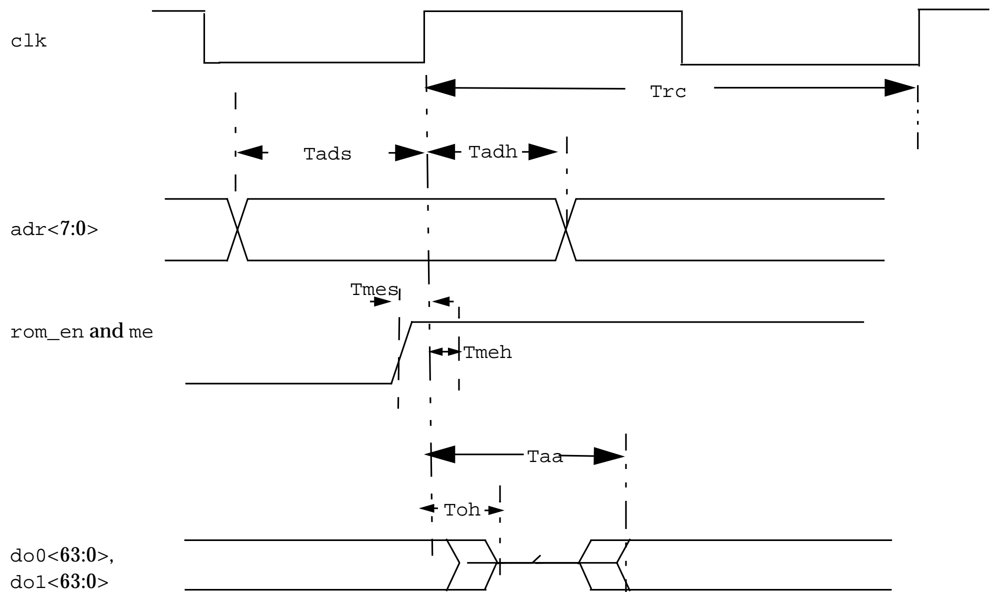


FIGURE 11-19 Timing Diagram A (FPU ROM) (Read Cycle, Normal Operation, Enabled, Nontest Mode)

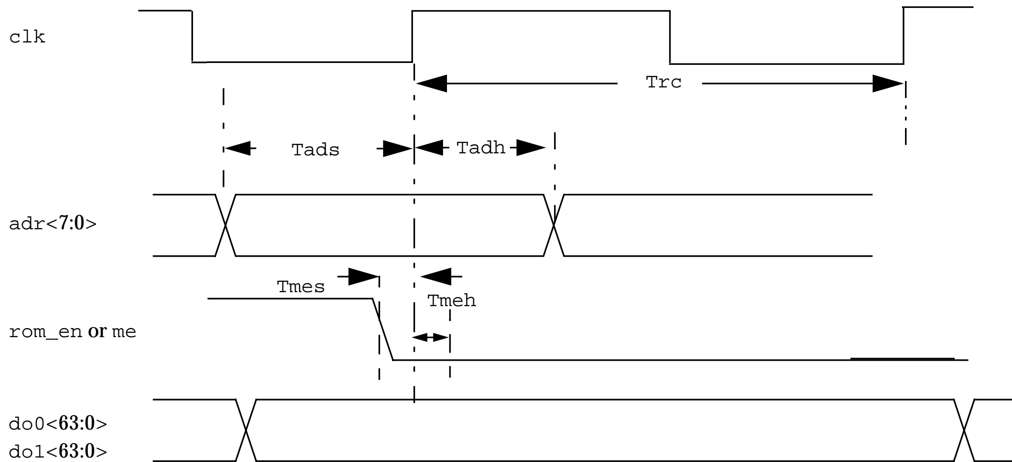


FIGURE 11-20 Timing Diagram B (FPU ROM) (Read Cycle, Normal Operation, Disabled, Nontest Mode)

11.6.4 Implementation

See FIGURE 11-16 on page 155 for details of the implementation.

11.6.5 ROM Contents

ROM contents are programmed at the time of manufacturing. See the Verilog model `fp_roms.v` for details.

11.6.6 RAM Redundancy

No redundancy is required for this block due to its relatively small ROM.

11.6.7 Testing

The ROM block is tested as embedded memory along with the rest of the chip silicon.

Note the following:

- All timing specifications are for the Sun SSLH corner operation conditions:
  - Process: Slow
  - Temperature: 105 degrees C junction
  - Vdd: 2.20v
- The test conditions for all timing specifications are:
  - Input Vil/Vih switching levels: 0.0v/2.20v
  - Input rise/fall times: 0.2 ns (10%~90% Vdd)
  - Input output timing reference levels: 2.20v/2

11.7 Integer Unit (IU) ROM

The IU ROM (`ieu_rom.v`) is a low-power, asynchronous, read-only memory (284 entries by 80 bits). A 9-bit address selects one of the memory entries, and the value is output on the `do` bus.

FIGURE 11-21 illustrates the IU ROM.

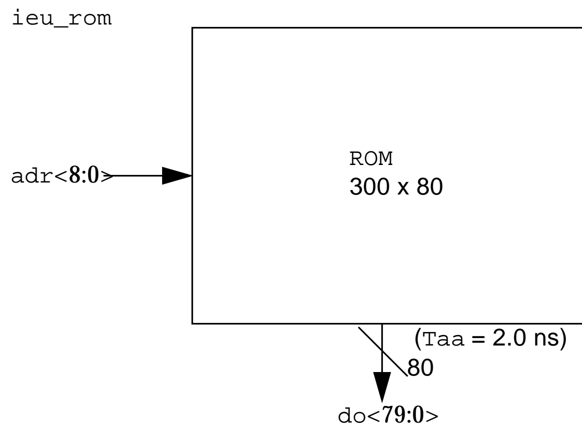


FIGURE 11-21 Integer Unit (IU) ROM

11.7.1 I/O Signals

TABLE 11-29 summarizes the input and output pins of the IU ROM and their input loading and output drive strengths.

TABLE 11-29 I/O Signals (IU ROM)

Signal	Description	Type	C_{in}/C_{load}
adr<8:0>	Address input for ROM	Input	0.02 pf
do<79:0>	Addressed data output	Output	0.25 pf

11.7.2 Timing

The specifications are listed below and in TABLE 11-30.

Process	Slow
Temperature	105 degrees C junction
Operating voltage	2.20v

TABLE 11-30 Timing Specifications (IU ROM)

Symbol	Description	Condition	Spec 200 MHz
Taa	Data read access time from the negative edge of clk	Maximum	2.0 ns

TABLE 11-22 illustrates the timing.

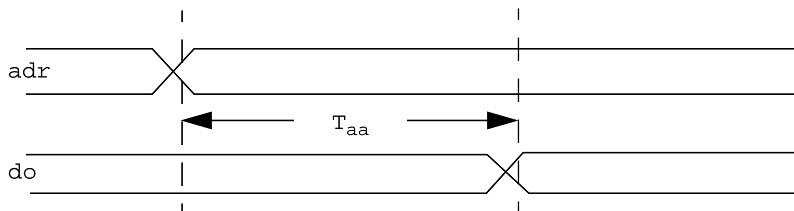


FIGURE 11-22 Timing Diagram (IU ROM)

11.7.3 Implementation

There are no special requirements for the IU ROM, except for meeting the timing specifications. See FIGURE 11-21 on page 161 for details of the implementation.

11.7.4 ROM Contents

ROM contents are programmed at the time of manufacturing.

For the format of the `ieu_rom` bit pattern, you can choose one of the following in `ieu_rom.v`:

- The Verilog `case` statement, which is the default
- The `megacell` format, which uses `ieu_rom.data`

11.7.5 RAM Redundancy

No redundancy is required for this block due to its relatively small ROM.

11.7.6 Testing

The ROM block is tested as embedded memory along with the rest of the chip silicon.

Note the following:

- All timing specifications are for the Sun SSLH corner operation conditions:
  - Process: Slow
  - Temperature: 105 degrees C junction
  - Vdd: 2.20v
- The test conditions for all timing specifications are:
  - Input Vil/Vih switching levels: 0.0v/2.20v
  - Input rise/fall times: 0.2 ns (10%~90% Vdd)
  - Input output timing reference levels: 2.20v/2

Index

A

add_a 150, 151
add_b 150, 151
add_c 150, 151
add_d 150, 151
add_e 150
addr 136, 137, 144, 145
adr 156, 157, 162
alu_out 38
alu_out_w 32
arch_optop 34
arch_pc 34
archi_data 38
areturn 76

B

bank_sel 136, 137
BIST hooks 90
bist_addr 136, 137, 144, 146
bist_data_in 136, 137
bist_enable 121, 122, 129, 130, 136, 137, 144, 146
bist_icu_addr 120, 122
bist_icu_din 120, 122
bist_icu_ram_we 121, 122
bist_icu_tag_addr 128, 130
bist_icu_tag_in 128, 130
bist_icu_tag_vld 128, 130
bist_icu_tag_we 128, 130
bist_stat_addr 144, 146
bist_stat_in 144, 146
bist_stat_we 145, 146
bist_tag_dout1 145
bist_tag_dout2 145
bist_tag_in 144, 146
bist_tag_we0 145, 146
bist_tag_we1 145, 146
bist_wb_set_sel 144, 146
bist_we 136, 137
BIU
  arbitration 80
  error transactions 81
  interface 82
    description 81
    pins 83
  overview 79
  power management 82
biu_data 16, 66, 83
biu_dcu_ack 67, 83
biu_icu_ack 16, 83
boot mode 105
BRK12C 92
Bus Interface Unit, *See* BIU
bypass 136, 137

C

C stage, *See* cache stage
cache
  flushing, ICU 14
  line invalidating 65
  read hits, ICU 12
  read misses, ICU 13
  stage (C stage) 21
cacheability of transactions 111
carry_out_e 32

clk 68, 89, 91, 93, 98, 100, 120, 121, 128, 130, 136, 137, 144, 145, 156, 157
clock management 88
cmp_addr_in 144, 145
control logic in microcode 37
customizable features 111

D

D stage, *See* decode stage
data breakpoint traps 92
Data Cache Unit, *See* DCU
data_in 136, 137
data_out 137, 138
datapath
    FPU 42
    microcode 38
    RCU 25
dcache_ram0_do 137
dcache_ram1_do 137
DCU
    address control 57
    aligner control 57
    arbiter 57
    arbitration of requests 64
    cache
        description 55
        fills 60
        flushing 65
        invalidate flushing 65
        line zeroing 65
        reads 59
        transactions 64
        writes 59
    critical timing path 68
    data RAMS 56
    D-Cache datapath 58
    dependencies 55
    diagnostic reads and writes 66
    interactions with other units 53
    interface pins 66
    miss control 58
    nonallocating writes 66
    noncacheable (NC)
        loads 63
        stores 63
    nonfetching allocates 66
    overview 53
    power management 68
    replacements 64
    writeback
        buffer 56
        control 58
        transactions 62
    zeroing of cache lines 65
dcu_addr 83
dcu_addr_out 66
dcu_biu_data 66
dcu_data 68, 78
dcu_data_c 31
dcu_dataout 83
dcu_diag_data_c 31
dcu_in_powerdown 91
dcu_req 67, 83
dcu_si 68
dcu_size 66, 83
dcu_so 68
dcu_type 67, 83
decode (D) stage 21
dest_addr_w 29
di_d 151
di_e 151
diag_ld_cache_c 15
do 162
do_a 151
do_b 151
do_c 151
do0 156, 157
do1 156, 157
dreg 38
dreturn 76
dtag_dout 145, 146

E

E stage, *See* execution stage
enable 120, 121, 128, 130, 136, 137, 144, 146
endianness of transactions 111
execution (E) stage 21

F

F stage, *See* fetch stage
features, customizable 111
fetch (F) stage 21

Floating Point Unit, *See* FPU
folded_r 34
folding
  groups, IFU 23
  logic, IFU 22
fpain 52
fpbin 52
fpbusyn 43, 50, 52
fphold 50, 52
fpkill 50, 52
fpop 52
fpop_valid 50, 52
fpout 52
FPU
  add and subtract 44
  ALU 43
  critical paths 51
  data forwarding 51
  datapath 42
  double precision 42
  FADD/FSUB operations 47
  FConvert operations 47
  FDIV/FREM/DDIV/DREM operations 47
  floating point
    adder-ALU 42
    multiply/divide unit 42
  FMUL/DMUL operations 47
  gradual underflow 45
  IEEE 754 compliance 45
  INaN 46
  input
    buses 48
    operations 48
    registers 42
  interface 48
  invalid combinations in NAN 46
  MDIV 43
  microcode sequencer 42, 43
  multiply and divide 45
  NAN 46
  opcodes 47
  output
    operations 50
    registers 42
  overview 41
  pipeline 43
  power management 48
  quiet NANs 46
  signals 52
  single precision 42
  structure 41
fpu_data_e 31
freturn 76

G

group_1_r 27
group_2_r 27
group_3_r 27
group_4_r 27
group_5_r 27
group_6_r 27
group_7_r 27
group_8_r 28
group_9_r 28

H

high watermark 75
high_mark 78
hit0_out 145, 146
hit1_out 145, 146
hold_c 28
hold_e 28

I

ialu_a 38
ibuf_cntl 12
ibuf_oplen 15
I-Buffer
  control 12
  interface in IFU 22
ic_cntl 10
ic_hit 16
I-Cache datapath 12
icram_dout 16, 122
icram_powerdown 16
ICU
  cache
    flushing 14
    read hits and misses 12
    transactions 12
  diagnostic reads and writes 14
  error transactions 14

functionalities 10
I-Cache control 10
interaction with other units 8
interface pins 15
noncacheable reads 14
overview 7
power management 15
structure 10
icu_addr 16, 83, 120, 121
icu_biu__addr 16
icu_diag_data_c 31
icu_din 16, 120, 121
icu_dout_d 15
icu_dpath 12
icu_drty_d 15
icu_hold 15
icu_in_powerdown 17, 91
icu_lvd_d 15
icu_pc_d 15, 34
icu_ram_we 16, 120, 121
icu_req 16, 83
icu_size 83
icu_tag_addr 16, 128, 130
icu_tag_in 16, 128, 130
icu_tag_vld 16, 128, 130
icu_tag_we 16, 128, 130
icu_tsize 16
icu_type 16, 83
IE field 116
ie_alu_cryout 38
ie_comp_a_eq0 38
ie_kill_ucode 38
ie_stall_ucode 38
imdr_data_e 31
inst_vld 28
inst_vld_r 34
instruction
breakpoint traps 92
buffer (I-Buffer) 11
pipeline 20
Instruction Cache Unit, *See* ICU
Instruction Folding Unit (IFU) 21
Integer Unit, *See* IU
interface
protocol in microcode 36
signals
core 97
IFU, IU, pipeline control, and SMU 27
IU datapath 31
memory 101
Interrupt Enable (IE) bit 116
interrupts
control of 116
definitions 114
maskable 115
nonmaskable
ireturn 76
itag_dout 16, 129
itag_hit 129
itag_vld 16, 129
IU
datapath
functional units 29
interface signals 31
functional descriptions 30
interface signals 34
overview 19
Pipe Control Unit (PCU) 32
ROM size 35, 37
stalls in the PCU 33
iu_addr_e 15, 32, 67
iu_address 78
iu_br_pc 15
iu_br_pc_e 32
iu_brtaken_e 15
iu_data 78
iu_data_e 32, 34, 67
iu_data_in 77
iu_data_w 28
iu_diag_e 67
iu_flush_e 15
iu_hold_e 37, 38
iu_inst_e 67
iu_int 77
iu_lvars 28
iu_optop 38
iu_optop_in 77
iu_optop_int_we 77
iu_powerdown_e 91
iu_pse_dre 77
iu_psr_bm8 15
iu_psr_dce 67

iu_psr_gce 37, 38
iu_psr_ice 15
iu_rf_dout 77
iu_sbase_we 77
iu_shift_d 15
iu_smiss 78
iu_smu_data 29
iu_smu_flush 78
iu_stall 67
iu_test_rw_e 15
iu_trap_c 34, *67*
iu_trap_r 28, 37
iu_zero_e 67

J

JTAG support 93

K

kill_inst_e 67

L

latency, read 105
Least Recently Used (LRU)
  bit 56, 65
  replacement policy 64
low watermark 75
low_mark 77
low-power modes 85
lreturn 76
LRU, *See* Least Recently Used (LRU)
lvars_acc_rs1_r 27

M

m_adder_sum 39
maskable interrupts 115
me 156, 157
megacells
  data cache
    data RAM 134–141
    tag RAM 141–149
  FPU ROM 154–161
  instruction cache
    data RAM 117–124
    tag RAM 125–134
  IU ROM 161–163
  stack cache RAM 150–154
  timing statistics 117
  types 117
memory
  controllers 105
  interface signals 101
microcode 35
misc_din 15
misc_dout 15

N

nastore_word_index 66
NMI, *See* nonmaskable interrupts
no_fold_r 28
nonmaskable interrupts (NMI) 115
nop 106

O

offset_1_rs1_r 27
offset_1_rs2_r 27
offset_2_rs1_r 27
offset_2_rs2_r 27
opcode_1_op_r 27, 37
opcode_1_rs1_r 27
opcode_1_rs2_r 27
opcode_1_rsd_r 27
opcode_2_op_r 27, 37
opcode_2_rs1_r 27
opcode_2_rs2_r 27
opcode_2_rsd_r 27
opcode_3_op_r 37
OPTOP 75
optop_offset 28

P

pc_c 31
pc_e 31
pc_offset_r 34

PCSU
cache flushing 87
clock management 88
dcu_in_powerdown signal 87
dcu_pwrdown signal 86
debug and trace features 92
exit from standby mode 87
icram_powerdown signal 85
icu_in_powerdown signal 87
normal mode 85
pj_irl signal 87
pj_nmi signal 87
pj_standby_out signal 88
reset management 88
scan and test features 89
signals 91
standby mode 86
pcsu_powerdown 17, 68, 91
PCU stalls 33
picoJava-II
core diagram 2
overview 1
PIL field 116
Pipe Control Unit (PCU) 32
pj_ack 60, 84, 99, 103, 107
pj_addr 99, 101, 107
pj_address 83
pj_ale 84, 99, 103
pj_boot8 99, 101, 105, 106
pj_brk1_sync 94, 99, 103
pj_brk2_sync 94, 99, 103
pj_clk 83
pj_clk_out 91, 98, 100
pj_data_in 83, 99, 101
pj_data_out 83, 99, 101
pj_halt 93, 99, 103
pj_in_halt 93, 99, 104
pj_inst_complete 93, 99, 104
pj_irl 91, 99, 100, 116
pj_irl_sync 91
pj_nmi 91, 99, 100, 115, 116
pj_nmi_sync 91
pj_no_fpu 99, 101
pj_reset 83, 88, 91
pj_reset_out 98, 100
pj_resume 94, 99, 103
pj_size 83, 99, 102, 107
pj_standby_out 88, 91, 99, 101
pj_su 99, 101
pj_tv 83, 99, 102, 107
pj_type 84, 99, 102, 106, 107
powerdown 52
Powerdown, Clock, Reset, and Scan Unit, *See* PCSU
priv_read_psr 105
priv_reset 89, 115
priv_ret_from_trap 76
priv_write_psr 105
Processor Interrupt Level (PIL) bit 116
PROM interface 105
PSR.BM8 105
PSR.IE 100
PSR.PIL 100

R

R stage, *See* register stage
read latency 105
read-type transactions 106
register (R) stage 21
Register Control Unit (RCU) 25
reissue_c 34
reset_l 52, 98, 100
ret_optop_update 78
return 76
return0 76
return1 76
return2 76
ROM size, IU 35, 37
rom_en 156, 157
rs1 38
rs1_data_e 28
rs1_forward_mux_sel 28
rs1_foward_mux_sel 28
rs2 38
rs2_data_e 28
rs2_forward_mux_sel 28
ru_rs1_e 31
ru_rs2_e 31

S

SC_BOTTOM 74, 75
scache_miss_addr_e 31
scache_wr_miss_w 29
set_sel 144, 146
si 89, 91, 93, 99, 101

sin 136
sm 89, 91, 93, 99, 101, 136
SMU
  datapath 74
  dribbling operations 74
  fill transactions 75
  functionalities 73
  interaction with other units 72
  interface pins 77
  overview 71
  spill transactions 75
  stack
    cache
      structure 73
      write misses 76
    overflow 75
    underflow 76
smu_addr 67, 78
smu_data 29, 67, 78
smu_data_vld 67, 78
smu_hold 77
smu_ld 67, 78
smu_na_st 78
smu_prty 67
smu_rf_addr 29, 77
smu_rf_din 77
smu_sbase 77
smu_sbase_we 77
smu_st 67, 78
smu_st_c 78
smu_stall 67, 78
smu_we 29, 77
so 89, 91, 93, 99, 101, 137
squash_fold 34
stack
  overflow 75
  underflow 76
Stack Manager Unit, *See* SMU
stat_addr 144, 145
stat_in 144, 145
stat_out 145, 146
stat_we 144, 146

T

Taa 152, 158, 162
Tadh 122, 131, 138, 147, 158
tadr 156, 157
Tads 122, 131, 138, 147, 158
tag_in 144, 145
tag_we 144, 146
Tasw 152
Taw 152
Tbph 138
Tbps 138
Tcmp_addr_val 147
Tcmph 147
Tcmps 147
Tcyc 122, 131, 138, 147
Tdh 152
Tdih 122, 131, 138, 147
Tdis 122, 131, 138, 147
Tdow 123
Tdw 152
Tdxor 122, 139
te 156, 157
Tenh 123, 131, 139, 147
Tens 123, 131, 139, 147
test_mode 118, 121, 122, 126, 129, 130, 135, 136, 137, 142, 145, 146
Thit 131, 147
Tidor 138
tm 156, 157
Tmeh 158
Tmes 158
Toh 152, 158
Tpwh 122, 131, 138, 147, 158
Tpwl 122, 131, 138, 147, 158
traps
  control of 114
  definitions 113
  types of 114
Trc 152, 158
Tstor 147
Tstslh 138, 147
Tstsls 138, 147
TT (trap type) 113
Ttag 131, 147
Tteh 158
Ttes 158
Ttgor 147
Ttmh 158
Ttms 158
Tvldor 131
Twb_set 147
Twc 152
Twd 152

Twdd 152
Twdh 152
Tweh 122, 131, 138, 147
Twes 122, 131, 138, 147
Twh 152
Twp 152
Twr 152
type_rs1_r 27

U

u_abt_rdwt 39
u_addr_st_rd 38
u_areg0 38
u_ary_ovf 38
u_done 37
u_f01_wt_stk 39
u_f02_rd_stk 39
u_f23~00 37
u_gc_notify 38
u_m_adder_porta 39
u_m_adder_porta_e 31
u_m_adder_portb 39
u_m_adder_portb_e 31
u_ptr_un_eq 38
u_ref_null 38
ucode_addr_s 28
ucode_areg0 28
ucode_porta 38
ucode_porta_e 31
ucode_portb 38
ucode_portc 38
ucode_portc_e 31

V

valid_op_r 37
valid_rs1_r 27
valid_rs2_r 27
valid_rsd_r 27

W

W stage, *See* write stage
watermarks 75
wb_set_sel 144, 146
we 136, 137
we_d 151
we_e 151
wr_optop_e 34
write
  stage (W stage) 21
  type transactions 108

Z

zero_line 65, 86
ZeroLineEmulationTrap 65