IBM PowerPC 970FX RISC Microprocessor

User's Manual

Version 1.7

March 14, 2008

IBM ®

© Copyright International Business Machines Corporation 2003, 2008

All Rights Reserved
Printed in the United States of America March 2008

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both.

IBM	POWER	PowerPC
IBM Logo	Power Architecture	PowerPC Architecture
ibm.com		

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Other company, product, and service names may be trademarks or service marks of others.

All information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in applications such as implantation, life support, or other hazardous uses where malfunction could result in death, bodily injury, or catastrophic property damage. The information contained in this document does not affect or change IBM product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as an illustration. The results obtained in other operating environments may vary.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN “AS IS” BASIS. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

IBM Systems and Technology Group
2070 Route 52, Bldg. 330
Hopewell Junction, NY 12533-6351

The IBM home page can be found at **ibm.com**®
The IBM Semiconductor solutions home page can be found **ibm.com**/chips

Version 1.7
March 14, 2008

Table of Contents

List of Tables

List of Figures

About This Book

The primary objective of the *IBM PowerPC® 970FX RISC Microprocessor User's Manual* is to define the functionality of the PowerPC 970FX microprocessor for software and hardware developers.

The information in this book is subject to change without notice, as described in the disclaimers on the title page. As with any technical documentation, it is the readers' responsibility to be sure they are using the most recent version of the documentation. To locate any published errata or updates for this document, go to **www-01.ibm.com**/chips/techlib.

Note: Soft copies of many of the latest versions of the manuals and documents referred to in this manual that are produced by IBM can be accessed on the Web at **www-01.ibm.com**/chips/techlib.

Audience

This manual is intended for system software and hardware developers and application programmers who want to develop products for the 970FX microprocessor. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of reduced instruction set computer (RISC) processing, and details of the PowerPC Architecture™.

Organization

For ease in reference, the arrangement of topics in this book is similar to that of the *PowerPC Microprocessor Family: The Programming Environments Manual for 64-Bit Microprocessors* and the *PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual* (see *Related Documents* on page 18). Topics build upon one another, beginning with a description and summary of 970FX-specific registers and instructions and progressing to more specialized topics such as 970FX-specific details regarding the cache, exception, memory management models, and power management. Thus, chapters might include information from multiple levels of the architecture. For example, the discussion of the cache model uses information from both the virtual environment architecture (VEA) and the operating environment architecture (OEA).

A summary and a brief description of the major sections of this manual follows:

- *Chapter 1 PowerPC 970FX Overview* is useful for readers who want a general understanding of the features and functions of the PowerPC Architecture and the 970FX microprocessor. This chapter describes the flexible nature of the PowerPC Architecture definition, and provides an overview of how the PowerPC Architecture defines the register set, operand conventions, addressing modes, instruction set, cache model, exception model, and memory management model.
- *Chapter 2 Programming Model* is useful for software engineers who need to understand the 970FX-specific registers, operand conventions, and details regarding how the PowerPC instructions are implemented on the 970FX microprocessor. Instructions are organized by function.
- *Chapter 3 Storage Subsystem* discusses the storage subsystem as implemented on the 970FX microprocessor. The storage subsystem includes the core interface logic, the non-cacheable unit, the L2 cache and controls, and the bus interface unit.
- *Chapter 4 Exceptions* describes the exception model defined in the PowerPC OEA and the specific exception model implemented on the 970FX microprocessor.

- *Chapter 5 Memory Management* describes the 970FX implementation of the memory management unit specifications provided by the PowerPC OEA for PowerPC processors.
- *Chapter 6 Software Optimization Guidelines* describes key design characteristics of the 970FX microprocessor.
- *Chapter 7 Signal Description* describes the individual signals of the 970FX microprocessor.
- *Chapter 8 Processor Interconnect Bus* describes the processor interconnect (PI) which is a bus architecture providing high-speed, high-performance interconnections for processors, I/O devices, memory subsystems, and bridge chips.
- *Chapter 9 Power and Thermal Management* provides information about power saving and thermal management modes for the 970FX microprocessor.
- *Chapter 10 970FX Performance Monitor* describes the operation of the performance monitor diagnostic tool incorporated in the 970FX microprocessor and provides detailed event information.
- *Chapter 11 System Design* describes system-related features such as power-on reset, the I^2C bus, the scan communication (SCOM) facility that is used to access processor debug and diagnostic facilities, and reliability, availability, serviceability (RAS) considerations.
- *Chapter 12 Vector Processing Unit* provides a general understanding of the features and functions of the vector processing unit (VPU) used on the 970FX microprocessor.

Related Documents

Companion Manuals

This manual is intended as a companion to the following reference manuals:

- PowerPC Architecture[1] books:

  Note: The PowerPC Architecture books supersede the *PowerPC Programming Environments Manual* for the 970FX implementation. However, not all features available in the PowerPC Architecture are supported in the 970FX microprocessor (such as, logical partitioning).

  – *PowerPC User Set Architecture (Book I, Version 2.01)*. Covers the base user instruction set architecture (UISA), user-level registers, data types, memory conventions, memory and programming models, and related facilities available to the application programmer.

  – *PowerPC Virtual Environment Architecture (Book II, Version 2.01)*. Defines the storage model and related instructions and facilities available to the programmer, and the time-keeping facilities available to the application programmer. The VEA, which is the smallest component of the PowerPC Architecture, defines additional user-level functionality that falls outside typical user-level software requirements. The VEA describes the memory model for an environment in which multiple processors or other devices can access external memory and define aspects of the cache model and cache control instructions from a user-level perspective. The resources defined by the VEA are particularly useful for optimizing memory accesses and for managing resources in an environment in which other processors and other devices can access external memory.

    Implementations that conform to the PowerPC VEA also conform to the PowerPC UISA, but might not necessarily adhere to the operating environment architecture (OEA).

1. PowerPC Architecture refers to the instructions and facilities described in Books I, II, and III.

  – *PowerPC Operating Environment Architecture (Book III, Version 2.01)*. Defines the system (privileged) instructions and related facilities. The OEA defines supervisor-level resources typically required by an operating system. The OEA defines the PowerPC memory management model, supervisor-level registers, and the exception model.

    Implementations that conform to the PowerPC OEA also conform to the PowerPC UISA and VEA.

- *PowerPC Microprocessor Family: Programming Environments Manual for 64-Bit Microprocessors* (referred to as the *Programming Environments Manual*). Provides information about resources defined by the PowerPC Architecture that are common to PowerPC processors. This manual describes the functionality of the 64-bit architecture model.
- *PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual*. Describes how the vector/SIMD technology relates to both the 64-bit and the 32-bit portions of the PowerPC Architecture.
- *The PowerPC Architecture: A Specification for a New Family of RISC Processors* by C. May, E. Silha, R. Simpson, and H. Warren, Morgan Kaufman, May 1994. Defines the architecture from the perspective of the three programming environments and remains the defining document for the PowerPC Architecture.

Because the PowerPC Architecture is designed to be flexible in order to support a broad range of processors, these documents provide a general description of features that are common to PowerPC processors and indicate those features that are optional or that might be implemented differently in the design of each processor.

It is important to note that some resources are defined more generally at one level in the architecture and more specifically at another. For example, conditions that cause a floating-point unavailable exception are defined by the UISA, while the exception mechanism itself is defined by the OEA.

Additional Documentation

Some additional PowerPC documentation is available at **ibm.com**/chips/techlib through IBM Customer Connect at http://ibm.com/technologyconnect.

- *IBM PowerPC 970FX RISC Microprocessor Datasheet.* This datasheet provides specific data about bus timing, signal behavior, and ac, dc, and thermal characteristics, as well as other design considerations for the 970FX implementation.
- *PowerPC 970FX Power On Reset Application Note*. This document contains information about required power-on-reset design and initialization.
- *PowerPC Microprocessor Family: The Programmer’s Reference Guide* (MPRPPCPRG-01). This is a concise reference that includes the register summary, memory control model, exception vectors, and the PowerPC instruction set.
- *PowerPC Microprocessor Family: The Programmer’s Pocket Reference Guide* (SA14-2093-00). This foldout card provides an overview of the PowerPC registers, instructions, and exceptions for 32-bit implementations.
- Application notes. These short documents contain useful information about specific design issues useful to programmers and engineers working with PowerPC processors.

General Information

The following documentation provides useful information about the PowerPC Architecture and computer architecture in general:

Ferraiolo, F., E. Cordero, D. Dreps, M. Floyd, "Power4: Synchronous Wave-Pipelined Interface." *Hot Chips 1999*, Stanford, CA.

Hennessy, John L. and David A. Patterson. *Computer Architecture: A Quantitative Approach*. 2nd ed.

I^2C-Bus Specification. Version 2.1. Philips Semiconductors, 2000.

IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Std. 1149.1a-1993.

McClanahan, Kip. *PowerPC Programming for Intel Programmers.* Foster City, CA: IDG Books Worldwide, Inc.

Shanley, Tom. *PowerPC System Architecture*. Richardson, TX: Mindshare, Inc.

Conventions

This document uses the following notational conventions:

mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number.
0b0	Prefix to denote binary number.
rA, **r**B	Instruction syntax used to identify a source GPR.
rD	Instruction syntax used to identify a destination GPR.
frA, **fr**B, **fr**C	Instruction syntax used to identify a source FPR.
frD	Instruction syntax used to identify a destination FPR.
crfS	Instruction syntax used to identify a source CR field.
crfD	Instruction syntax used to identify a destination CR field.
REG[FIELD]	Abbreviations or acronyms for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets. For example, MSR[POW] refers to the power management bit in the machine state register.
vA, **v**B, **v**C	Instruction syntax used to identify a source VR.
vD	Instruction syntax used to identify a destination VR.
x	In certain contexts, such as a signal encoding, this indicates a don't care.
n	Used to express an undefined numerical value.

¬	NOT logical operator.
&	AND logical operator.
|	OR logical operator.
0 0 0 0	Indicates reserved bits or bit fields in a register. Although these bits may be written to as either ones or zeros, they are always read as zeros.

Acronyms and Abbreviations

Table i contains acronyms and abbreviations that are used in this document.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
ALU	Arithmetic logic unit.
ASI	Active source identifier.
ASR	Address space register.
AUE	Special UE to indicate altered UE.
BCM	Balanced coding method.
BIST	Built-in self test.
BHT	Branch history table.
BIU	Bus interface unit.
BPU	Branch processing unit.
BSDL	Boundary-scan description language.
CAM	Content-addressible memory.
CDF	Critical data forward.
CIU	Core interface unit.
CMOS	Complementary metal-oxide semiconductor.
COP	Common on-chip processor.
CR	Condition register.
CRA	Custom register array.
CQ	Completion queue.
CSI	Context synchronizing instruction.
CTR	Count register.
DABR	Data address breakpoint register.
DAR	Data address register.
DCMP	Data TLB compare.
DEC	Decrementer register.
DMISS	Data TLB miss address.
DPM	Dynamic power management.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
DSISR	Register used for determining the source of a DSI exception.
DTLB	Data translation lookaside buffer.
EA	Effective address.
EAR	External access register.
ECC	Error checking and correction.
ERAT	Effective to real address translation cache.
FIFO	First-in-first-out.
FIR	Fault isolation register.
FPECR	Floating-point exception cause register.
FPR	Floating-point register.
FPSCR	Floating-point status and control register.
FPU	Floating-point unit.
GPIO	General purpose I/O pins.
GPR	General purpose register.
HID*n*	Hardware implementation-dependent register.
IABR	Instruction address breakpoint register.
IAP	Initial alignment pattern.
IEEE ®	Institute for Electrical and Electronics Engineers.
IQ	Instruction queue.
ITLB	Instruction translation lookaside buffer.
JTAG	Joint Test Action Group.
L2	Secondary cache (Level 2 cache).
L2C	L2 cache controller.
LHR	Load hit reload. A load presented through a load/store port to the LMQ matches an existing entry which has already initiated a request to L2.
LHS	Load-Hit-Store. A load presented through a load/store port to the store reorder queue (SRQ) matches an existing entry. Store forwarding may be attempted. If the store contains all the data required by the load, store forwarding can occur. If the store does not contain all the data required by the load, store forwarding cannot occur and the load is rejected or flushed.
LIFO	Last-in-first-out.
LMQ	Load miss queue. An 8-entry queue which tracks loads that miss the L1 and are awaiting data from 970FX storage subsystem (STS). Each entry can handle two loads associated with a cache line.
LR	Link register.
LRU	Least recently used.
LSB	Least-significant byte.
lsb	Least-significant bit.
LSU	Load/store unit. The unit in the microprocessor which executes load and store instructions.
MERSI	Modified/exclusive/recent/shared/invalid—cache coherency protocol.
MMCR*n*	Monitor mode control registers.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
MMU	Memory management unit.
MRU	Most recently used.
MSB	Most-significant byte.
msb	Most-significant bit.
MSR	Machine state register.
NaN	Not a number.
NCU	Non-cacheable unit.
NIA	Next instruction address.
No-op	No operation.
NSA	Next sequential address.
NTC	Next to complete.
OEA	Operating environment architecture.
PID	Processor identification tag.
PFQ	Data Prefetch Filter Queue. Filter queue of 12 entries which detect data streams for prefetching.
PLL	Phase-locked loop.
PMC*n*	Performance monitor counter registers.
POR	Power-on reset.
POWER	Performance Optimized with Enhanced Reduced Instruction Set Computing (RISC) architecture.
PRQ	Data Prefetch Request Queue. A prefetch queue of eight streams which will be prefetched.
PTE	Page table entry.
PTEG	Page table entry group.
PVR	Processor version register.
RAW	Read-after-write.
RISC	Reduced instruction set computing.
RLM	Random logic macro.
RMCI	Real mode cache inhibited.
RTL	Register transfer language.
RWITM	Read with intent to modify.
RWNITM	Read with no intent to modify.
SCOM	Scan Communication (internal chip bus).
SDA	Sampled data address register.
SDQ	Store data queue.
SDR1	Register that specifies the page table base address for virtual-to-physical address translation.
SHL	Store-Hit-Load.
SHR	Store-Hit-Reload. A committed store ready to write to the L1 data cache (D-cache) line that matches an existing LMQ entry. The store is stalled until the reload is complete.
SIA	Sampled instruction address register.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
SIMM	Signed immediate value.
SLB	Segment lookaside buffer.
SPR	Special-purpose register.
SPU	Service processor unit.
SR*n*	Segment register.
SRC	System reference code.
SRQ	Store Reorder Queue. A 32-entry queue that tracks all stores active in the LSU.
SRR0	Machine status save/restore register 0.
SRR1	Machine status save/restore register 1.
SSB	Source synchronous bus.
STE	Segment table entry.
STS	970FX storage subsystem which includes core interface logic, non-cacheable unit, L2 cache and controls, and the bus interface unit.
SUE	Special UE to indicate a passed error.
TB	Timebase facility.
TBL	Timebase lower register.
TBU	Timebase upper register.
THRM n	Thermal management registers.
TLB	Translation lookaside buffer.
UE	Uncorrectable memory error.
UIMM	Unsigned immediate value.
UISA	User instruction set architecture.
UMMCR*n*	User monitor mode control registers.
UPMC*n*	User performance monitor counter registers.
USIA	User sampled instruction address register.
VA	Virtual address.
VALU	VPU arithmetic logic unit (ALU).
VEA	Virtual environment architecture.
VPU	Vector processing unit within the core.
VR	Vector register.
WAR	Write-after-read.
WAW	Write-after-write.
WIMG	Write-through/caching-inhibited/memory-coherency enforced/guarded bits.
XER	Integer exception register used for indicating conditions such as carries and overflows for integer operations.

Terminology Conventions

Table ii describes terminology conventions used in this manual and the equivalent terminology used in the PowerPC Architecture specification.

Table ii. Terminology Conventions

Architecture Specification	Current Manual
Data storage interrupt (DSI)	DSI exception
Extended mnemonics	Simplified mnemonics
Fixed-point unit (FXU)	Integer unit (IU)
Instruction storage interrupt (ISI)	ISI exception
Interrupt	Exception
Privileged mode (or privileged state)	Supervisor-level privilege
Problem mode (or problem state)	User-level privilege
Real address	Physical address
Relocation	Translation
Storage (locations)	Memory
Storage (the act of)	Access
Store in	Write back
Store through	Write through
Swizzling	Double-word swap

Table iii describes instruction field notation used in this manual.

Table iii. Instruction Field Conventions

Architecture Specification	Equivalent to:
BA, BB, BT	**crb**A, **crb**B, **crb**D (respectively)
BF, BFA	**crf**D, **crf**S (respectively)
D	d
DS	ds
FLM	FM
FRA, FRB, FRC, FRT, FRS	**fr**A, **fr**B, **fr**C, **fr**D, **fr**S (respectively)
FXM	CRM
RA, RB, RT, RS	**r**A, **r**B, **r**D, **r**S (respectively)
SI	SIMM
U	IMM
UI	UIMM
VA, VB, VT, VS	**v**A, **v**B, **v**D, **v**S (respectively)
VEC	VPU
/, //, ///	0...0 (shaded)

1. PowerPC 970FX Overview

The IBM PowerPC 970FX reduced instruction set computer (RISC) microprocessor is an implementation of the PowerPC Architecture. This chapter provides an overview of the features of the 970FX microprocessor and includes a block diagram showing the major functional components.

Note: In this document the IBM PowerPC 970FX RISC Microprocessor is abbreviated as PowerPC 970FX or 970FX microprocessor.

1.1 PowerPC 970FX Microprocessor Overview

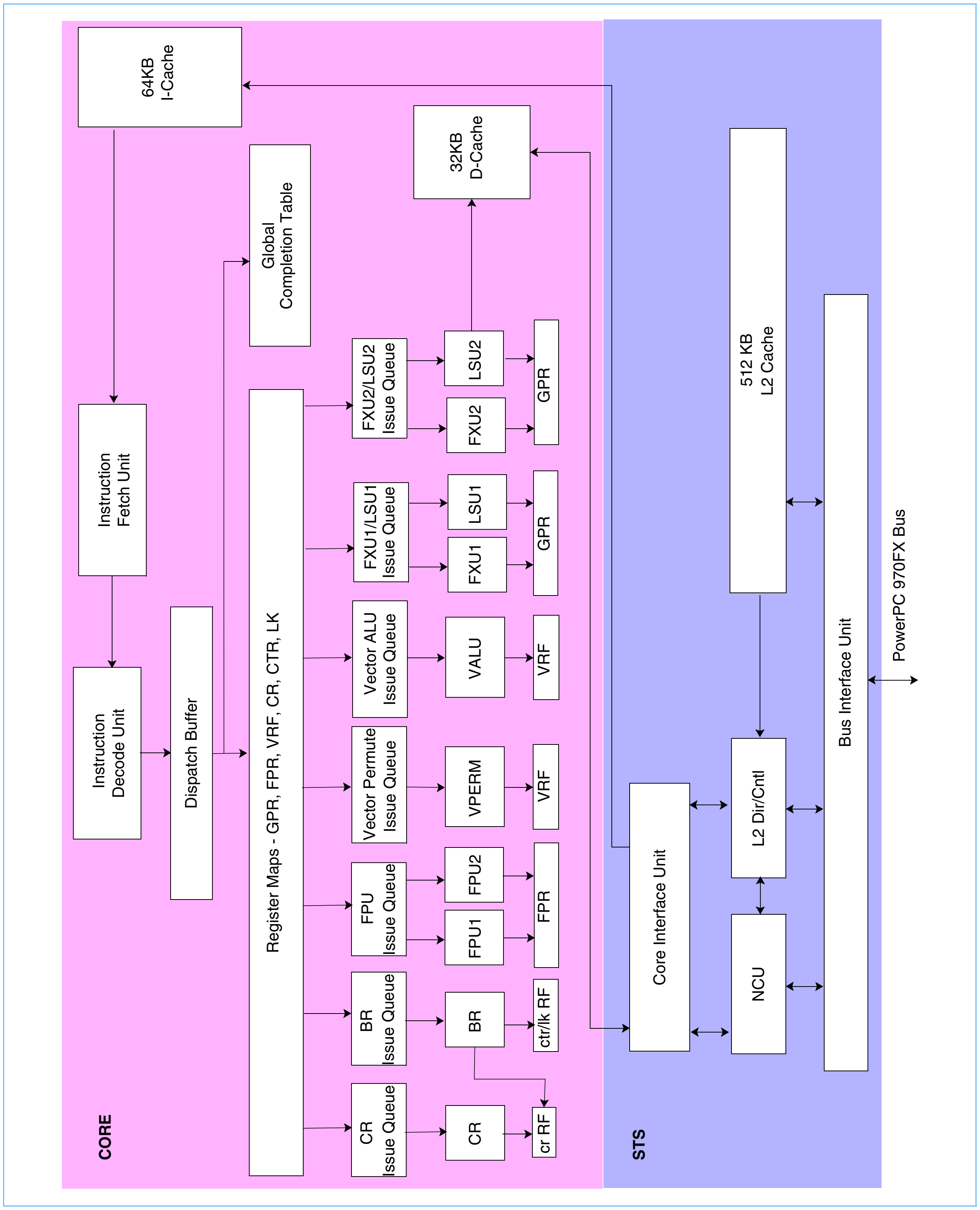
The 970FX is a 64-bit PowerPC RISC microprocessor with vector technology extensions–the single-instruction, multiple-data (SIMD) operations that accelerate data intensive processing tasks. This processor is designed to support multiple system configurations ranging from desktop and low-end server applications for uniprocessor up through 4-way symmetric multiprocessor (SMP) configurations.

The PowerPC 970FX RISC Microprocessor consists of three main components:

- The core which includes the vector processing execution units (VPU)
- The storage subsystem (STS), which includes the core interface logic, non-cacheable unit, L2 cache and controls, and the bus interface unit
- Pervasive functions

The block diagram in *Figure 1-1* on page 28 shows the major functional units comprising the core and storage subsystem. In the core, these units include instruction fetch, decode and dispatch units, plus the register files and execution units. The storage subsystem includes the second level (L2) cache and interface units.

Figure 1-1. 970FX Block Diagram



1.2 PowerPC 970FX Functional Units

1.2.1 Introduction

This section provides an overview of the 970FX microprocessor core, VPU, storage, and bus interface units. It includes a summary and details of key design fundamentals.

1.2.1.1 Key Design Fundamentals of the Microprocessor Core

- 64-bit implementation of the PowerPC® Architecture (version 2.01)
  - Binary compatibility with all PowerPC application level code (problem state)
  - Support for 32-bit operating system (O/S) bridge facility
  - Vector/SIMD multimedia extension
- Layered implementation strategy for very high frequency operation
  - Deeply pipelined design
    - 16 stages for most fixed-point register-to-register operations
    - 18 stages for most load and store operations (assuming an L1 D-cache hit)
    - 21 stages for most floating-point operations
    - 19 stages for fixed-point, 22 stages for complex-fixed, and 25 stages for floating-point operations in the vector arithmetic logic unit (VALU)
    - 19 stages for vector permute operations
  - Dynamic instruction cracking[1] for some instructions allows for simpler inner core dataflow
    - Dedicated dataflow for cracking one instruction into two internal operations
    - Microcoded templates for longer emulation sequences
- Speculative superscalar inner core organization
  - Aggressive branch prediction
    - Prediction for up to two branches per cycle
    - Support for up to 16 predicted branches in flight
    - Prediction support for branch direction and branch addresses
  - In-order dispatch of up to five operations into the distributed issue queue structure
  - Out-of-order issue of up to 10 operations into 10 execution pipelines
    - Two load or store operations
    - Two fixed-point register-register operations
    - Two floating-point operations
    - One branch operation
    - One Condition Register operation
    - One vector permute operation
    - One vector ALU operation
  - Register renaming on General Purpose Registers (GPRs), Floating-Point Registers (FPRs), Vector Registers (VRs), Condition Register (CR) fields, two bits of the Integer Exception Register (XER), Floating-Point Status and Control Register (FPSCR), Vector Save/Restore Register (VRSAVE), Vector Status and Control Register (VSCR), Link Register (LR), and Count Register (CTR)
- Large number of instructions in flight (theoretical maximum of 215 instructions)

1. Process by which some complex instructions are broken into two simpler, more RISC-like instructions.

- Up to 16 instructions in the instruction fetch unit (fetch buffer and overflow buffer)
- Up to 32 instructions in the instruction fetch buffer in the instruction decode unit
- Up to 35 instructions in three decode pipe stages and four dispatch buffers
- Up to 100 instructions in the inner-core (after dispatch)
- Up to 32 stores queued in the store queue (STQ) (available for forwarding)
- Fast, selective flush of incorrect speculative instructions and results

- Specific focus on storage latency management
  - Out-of-order and speculative issue of load operations
  - Support for up to eight outstanding L1 cache line misses
  - Hardware-initiated instruction prefetching from L2 cache
  - Software-initiated data stream prefetching with support for up to eight active streams
  - Critical word forwarding–critical sector first
  - New branch processing–prediction hints on branch instructions
- Power management
  - Static power management
    - Software initiated doze and nap low-power modes
  - Dynamic power management
    - Parts of the design stop their clocks when not in use under hardware control
  - Power tuning through frequency scaling
    - Software initiated slow down of the processor; selectable to half of the nominal operating frequency

1.2.1.2 Detailed Features of the Microprocessor Core

- Instruction fetching and branch prediction
  - 64 KB, direct-mapped instruction cache (I-cache)
    - 128-byte lines (broken into four 32-byte sectors)
    - Dedicated 32-byte read/write interface from L2 cache with a critical sector first reload policy
    - Effective-address index, real address tags
    - Cache supports one read or one write per cycle
    - Five additional predecode bits per word to aid in fast decoding and group formation
    - Parity protected with a force invalidate and reload on parity error
  - 128 total entries in the effective-to-real-address translation (ERAT) cache, 2-way set associative
    - Organization is 64 entries by two ways
    - Each entry translates 4 KB (no large page support; large pages take multiple entries)
  - 4-entry, 128-byte, instruction prefetch queue above the I-cache; hardware-initiated prefetches
  - Fetch a 32-byte aligned block of eight instructions per cycle
  - Branch prediction:
    - Scan all eight fetched instructions for branches each cycle
    - Predict up to two branches per cycle
    - Three-table prediction structure - global / local / selector (16K entries x 1-bit each)
    - 16-entry link stack for address prediction (with stack recovery)
    - 32-entry count cache for address prediction (indexed by the address of Branch Conditional to Count Register (**bcctr)** instructions)
- Instruction decode and preprocessing
  - 3-cycle pipeline to decode and preprocess instructions
    - Dedicated dataflow for cracking one instruction into two internal operations
    - Microcoded templates for longer emulation sequences of internal operations
    - All internal operations expanded into 86-bit internal form to simplify subsequent processing and explicitly expose register dependencies for all register pools
    - Dispatch groups (up to five instructions) formulated along with inter-instruction dependence masks
  - Cracked and microcoded instructions have access to four renamed emulation GPRs (eGPRs), one renamed emulation FPR (eFPR), and one renamed emulation CR (eCR) field (in addition to architected facilities)
  - 8-entry (16 bytes per entry) instruction fetch buffer (up to eight instructions in and five instructions out during each cycle)
  - Microcode patch facility allows most instructions other than branches to trap to microcode, which can be programmed to either emulate the effects of the instruction or cause an interrupt.
- Instruction dispatch, sequencing, and completion control
  - Four dispatch buffers, which can hold up to four dispatch groups when the global completion table (GCT) is full
  - 20-entry global completion table
    - Group-oriented tracking associates a 5-operation dispatch group with a single GCT entry
    - Tracks internal operations from dispatch to completion for up to 100 operations
    - Capable of restoring the machine state for any of the instructions in flight
      - Very fast restoration for instructions on group boundaries (that is, branches)
      - Slower for instructions contained within a group

- Supports precise exceptions (including machine check exceptions)
- Register renaming resources
  - 80-entry GPR rename mapper (32 architected GPRs plus four eGPRs and VRSAVE)
  - 80-entry FPR rename mapper (32 architected FPRs plus one eFPR)
  - 80-entry Vector Register file (VRF) rename mapper (32 architected VRFs)
  - 24-entry XER rename mapper (the XER is broken into mappable and non-mappable fields)
    - Two mappable fields: ov and ca
    - Non-mappable field: string-count
  - 16-entry LR/CTR rename mapper (one architected LR and one architected CTR)
  - 32-entry CR rename mapper (eight architected CR fields plus one eCR field)
  - 20-entry FPSCR rename mapper
  - VRSAVE
  - VSCR
- Instruction queuing resources:
  - Two 18-entry issue queues for fixed-point and load/store instructions
  - Two 10-entry issue queues for floating-point instructions
  - 12-entry issue queue for branch instructions
  - 10-entry issue queue for CR-logical instructions
  - 16-entry issue queue for vector permute instructions
  - 20-entry issue queue for vector ALU instructions and vector stores

- Fixed-point execution pipelines
  - Two fixed-point execution pipelines
    - Both capable of basic arithmetic, logical, and shifting operations
    - Both capable of multiplies
    - One capable of divides; the other capable of SPR operations
  - Out-of-order issue with bias towards oldest operations first
  - Symmetric forwarding between fixed-point and load/store execution pipelines
- Load/store execution pipelines
  - Two 6-stage load/store execution pipelines
  - Out-of-order issue with bias towards oldest operations first
    - Stores issue twice–an address generation operation (load/store), and a data steering operation (FXU/FPU/VPU)
  - 32 KB, 2-way, set-associative D-cache
    - Triple ported to support two reads and one write every cycle (no banking)
    - 2-cycle load-use penalty for FXU loads
    - 4-cycle load-use penalty for FPU loads
    - 3-cycle load-use penalty for loads to vector permute unit (VPERM)
    - 4-cycle load-use penalty for loads to VALU
    - Store-through policy; no allocation on store misses
    - 128-byte cache line
    - Least recently used (LRU) replacement policy
    - Dedicated 32-byte reload interface from L2 cache
    - Effective-address index, real address tags (hardware fix up on alias cases)
    - Parity protected; precise machine check interrupt on parity error; software fix if HID5[50] equals '1'. Otherwise, recovery is done by hardware (default).

- 128-entry total ERAT translation cache, 2-way, set-associative
  - Organization is 64 entries by two ways
  - Each entry translates 4 KB (no large page support; large pages take multiple entries)
- 32-entry store queue logically above the D-cache (real address based; content-addressable memory [CAM] structure)
  - Store addresses and store data can be supplied on different cycles
  - Stores wait in this queue until they are completed; then they write the cache
  - Supports store forwarding to inclusive subsequent loads (even if both are speculative)
- 32-entry load reorder queue (real address based; CAM structure)
  - Keeps track of out-of-order loads and watches for hazards
    - Previous store to the same address that gets executed after the load causes a flush
    - Previous load from the same address when a cross-invalidate has occurred causes a flush
- 8-entry load miss queue (LMQ) (real address based)
  - Keeps track of loads that have missed in the L1 D-cache
  - Allows a second load from the same cache line to merge onto a single entry

- Branch and Condition Register execution pipelines
  - One branch execution pipeline
    - Computes actual branch address and branch direction for comparison with prediction
    - Redirects instruction fetching if either prediction was incorrect
    - Assists in training/maintaining the branch table predictors, the link stack, and the count cache
  - One Condition Register logical pipeline
    - Executes CR logical instructions and the CR movement operations
    - Executes some Move to Special Purpose Register (**mtspr**) and Move from Special Purpose Register (**mfspr**) instructions also
  - Out-of-order issue with bias towards oldest operations first
- Floating-point execution pipelines
  - Two 9-stage floating-point execution pipelines (6-stage execution)
    - Both capable of the full set of floating-point instructions
    - All data formats supported in hardware (no floating-point assist interrupts)
  - Out-of-order issue with a bias towards oldest operations first
  - Symmetric forwarding between the floating-point pipelines
  - No support for the non-IEEE mode
- VPU execution pipelines
  - Two dispatchable units:
    - VALU contains three subunits:
      - Vector simple fixed: 1-stage execution
      - Vector complex fixed: 4-stage execution
      - Vector floating-point: 7-stage execution
    - VPERM: 1-stage execution
  - Out-of-order issue with bias towards oldest operations first
  - Symmetric forwarding between the permute and VALU pipelines

- Unified second-level memory management (address translation)
  - 1024-entry, 4-way, set-associative translation lookaside buffer (TLB)
    - Supports new large page architecture (16 MB large pages supported)
    - Hardware-based reload (from the L2 cache interface - no L1 D-cache impact)
    - Hardware-based update of the referenced (R) bit and the changed (C) bit
    - Parity protected; precise machine check interrupt on parity error (software fix-up)
  - 64-entry fully associative segment lookaside buffer (SLB)
    - SLB miss results in an interrupt and the software reload of the SLB
    - SLB can also be loaded using the 32-bit PowerPC Segment Register instructions
  - Supports a 65-bit virtual address and a 42-bit real address
- Data stream prefetch
  - Eight (modeable) data prefetch streams supported in hardware. Eight hardware streams are only available if vector prefetch instructions are disabled.
  - Four vector prefetch streams supported using four of the eight hardware streams. The vector prefetch mapping algorithm supports the most commonly used forms of vector prefetch instructions.

2. Programming Model

This chapter describes the 970FX programming model, emphasizing those features specific to the 970FX processor and summarizing those that are common to PowerPC processors. It consists of two major sections, which describe the following:

- Registers implemented in the 970FX
- 970FX instruction set

2.1 970FX Processor Register Set

This section describes the registers implemented in the 970FX. It includes an overview of registers defined by the PowerPC Architecture, highlighting differences in how these registers are implemented in the 970FX, and a detailed description of 970FX-specific registers.

Registers are defined at all three levels of the PowerPC Architecture—user instruction set architecture (UISA), virtual environment architecture (VEA), and operating environment architecture (OEA). The PowerPC Architecture defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions and reducing the number of instructions required for certain operations. Data is transferred between memory and registers with explicit load and store instructions only.

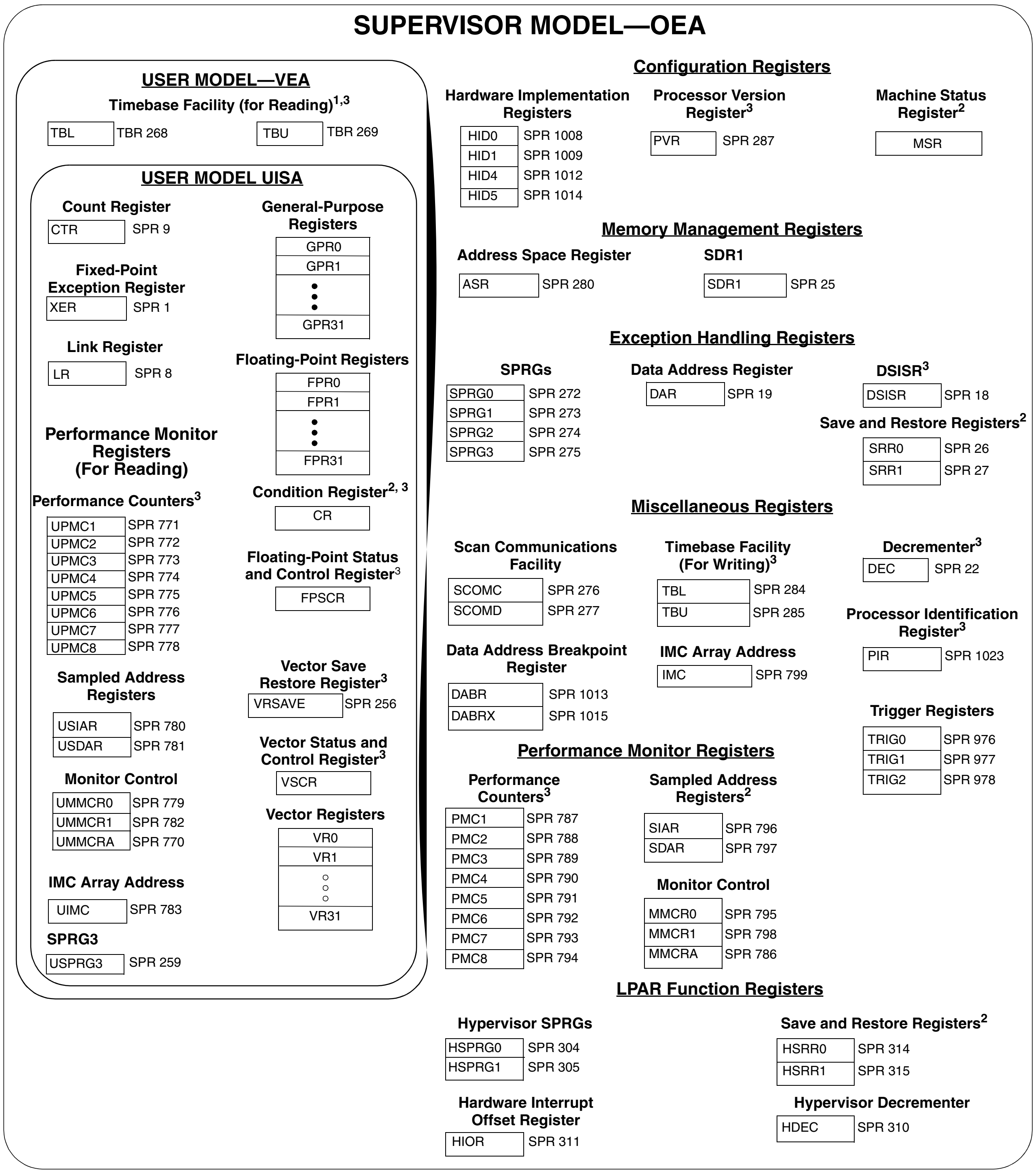
2.1.1 Register Set

PowerPC processors have two levels of privilege—supervisor mode of operation (typically used by the operating system) and user mode of operation (used by the application software, it is also called problem state). The programming models incorporate 32 GPRs, 32 FPRs, 32 VRs, special-purpose registers (SPRs), and several miscellaneous registers. Each PowerPC microprocessor also has its own unique set of hardware implementation-dependent (HID) registers.

While running in supervisor mode the operating system is able to execute all instructions and access all registers defined in the PowerPC Architecture. In this mode the operating system establishes all address translations and protection mechanisms, loads all processor state registers, and sets up all other control mechanisms defined on the 970FX processor. While running in user mode (problem state) many of these registers and facilities are not accessible and any attempt to read or write to these registers results in a program exception.

The registers implemented on the 970FX are shown in *Figure 2-1. PowerPC 970FX Microprocessor Programming Model—Registers*. The number to the right of the special-purpose registers (SPRs) indicates the number that is used in the syntax of the instruction operands to access the register (for example, the number used to access the integer exception register (XER) is SPR 1). These registers can be accessed using the **mtspr** and **mfspr** instructions. The inclusion of the VPU involves additional registers, and affects bit settings in some of the PowerPC registers (including MSR, SRR1, and CR) when the VPU is in use.

Figure 2-1. PowerPC 970FX Microprocessor Programming Model—Registers



Note:

1. TBR268 is read as a 64-bit value.
2. PowerPC registers affected by vector instructions
3. 32-bit registers.

The PowerPC UISA registers are user-level. General-purpose registers (GPRs), floating-point registers (FPRs), and vector registers (VRs) are accessed through instruction operands. Access to registers can be explicit (by using instructions for that purpose such as Move to Special Purpose register (**mtspr**) and Move from Special-Purpose register (**mfspr**) instructions) or implicit as part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

Implementation Note—The 970FX fully decodes the SPR field of the instruction. If the SPR specified is undefined, the illegal instruction program exception occurs. The PowerPC's user-level registers are described as follows.

- **User-level registers** (UISA)—The user-level registers can be accessed by all software with either user or supervisor privileges. They include the following registers.
  - General Purpose Registers (GPRs). The 32 GPRs (GPR0–GPR31) serve as data source or destination registers for fixed-point instructions and provide data for generating addresses.
  - Floating-Point Registers (FPRs). The 32 FPRs (FPR0–FPR31) serve as the data source or destination for all floating-point instructions.
  - Condition Register (CR). The 32-bit CR consists of eight 4-bit fields, CR0–CR7, that reflect results of certain arithmetic operations and provide a mechanism for testing and branching.
  - Floating-Point Status and Control Register (FPSCR). The FPSCR contains all floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits needed for compliance with the IEEE 754 standard.
  - Vector Registers (VRs). The vector register file consists of 32 VRs (VR0-VR31). The VRs serve as vector source and vector destination registers for all vector instructions.
  - Vector Status and Control Register (VSCR). The VSCR contains the non-Java control bit and the saturation status bit associated with vector operations.

  The remaining user-level registers are SPRs. Note that the PowerPC Architecture provides a separate mechanism for accessing SPRs (the **mtspr** and **mfspr** instructions). These instructions are commonly used to explicitly access certain registers, while other SPRs may be more typically accessed as the side effect of executing other instructions.

  - Integer Exception Register (XER). The XER indicates overflow and carries for integer operations and the number of bytes to be transferred by the load/store string indexed instructions.
    Implementation Note—The architecture defines XER[44:56] as reserved.
  - Link Register (LR). The LR provides the branch target address for the Branch Conditional to Link Register (**bclr***x*) instruction, and can be used to hold the logical address of the instruction that follows a branch and link instruction, typically used for linking to subroutines.
  - Count Register (CTR). The CTR holds a loop count that can be decremented during execution of appropriately coded branch instructions. The CTR can also provide the branch target address for the Branch Conditional to Count Register (**bcctr***x*) instruction.
  - Vector Save/Restore Register (VRSAVE). The VRSAVE assists the application and operating system software in saving and restoring the vector register architectural state across context-switching events.
  - User Performance Monitor Counter Registers (UPMC1-UPMC8). The UPMC1-UPMC8 provide user-level read access to PMC1-PMC8.
  - User Monitor Mode Control Registers (UMMCR0, UMMCR1, UMMCRA). These registers provide user-level read access to the MMCR0, MMCR1, MMCRA.

- User Instruction Match CAM registers (UIMC). The UIMC provides user-level read access to the IMC.
- User Sampled Instruction Address Register (USIAR). The USIAR provides user-level read access to the SIAR.
- User Sampled Data Address Register (USDAR). The USDAR provides user-level read access to the SDAR.
- User Software Use Special Purpose Registers (USPRG3). The USPRG3 provide an optional user-level read access to the SPRG3.

• **User-level registers** (VEA)—The PowerPC VEA defines the time base facility (TB), which consists of two 32-bit registers—time base upper (TBU) and time base lower (TBL). The time base registers can be written to only by supervisor-level instructions, but can be read by both user and supervisor-level software.

• **Supervisor-level registers** (OEA)—The OEA defines the registers that an operating system uses for memory management, configuration, exception handling, and other operating system functions. The OEA defines the following supervisor-level registers:

  - Configuration registers
    - Machine State Register (MSR). The MSR defines the state of the processor. The MSR can be modified by the Move to Machine State Register (**mtmsr**[**d**]), System Call (**sc**), and Return from Exception (**rfid**) instructions. It can be read by the Move from Machine State Register (**mfmsr**) instruction. When an exception is taken, the contents of the MSR are saved to the machine status save/restore register 1 (SRR1). See *Section 2.1.2.1. MSR Register (MSR)* for more information.
    - Processor Version Register (PVR). This is a read-only register that identifies the version (model) and revision level of the PowerPC processor. See the *IBM PowerPC 970FX RISC Microprocessor Datasheet* for details of the PVR.
  - Memory management registers
    - Address Space Register (ASR). In the 970FX, the address space register is supported. Due to the software reload of the SLBs on the 970FX, this register does not actually participate in any other specific hardware functions on the chip. It has been included as a convenience (and performance enhancement) for the SLB reload software.
    - Storage Description Register (SDR1). The SDR1 register specifies the page table base address used in virtual-to-physical address translation.
  - Exception-handling registers
    - Data Address Register (DAR). After a DSI or an alignment exception, DAR is set to the effective address (EA) generated by the faulting instruction.
    - Software Use Special Purpose Registers (SPRG0–SPRG3). The SPRG0–SPRG3 registers are provided for operating system use.
    - Data Storage Interrupt Status Register (DSISR). The DSISR register defines the cause of DSI and alignment exceptions.
    - Machine Status Save/Restore Register 0 (SRR0). The SRR0 register is used to save the address of the instruction at which execution continues when **rfid** executes at the end of an exception handler routine. See *Section 2.1.2.2. Machine Status Save/Restore Register (SRR1)* for more information.
    - Machine Status Save/Restore Register 1 (SRR1). The SRR1 is a 64-bit register used to save machine status on exceptions and restore machine status register when an **rfid** instruction is executed. See *Section 2.1.2.2. Machine Status Save/Restore Register (SRR1)* for more information.

Note: For information on how specific exceptions affect SRR1, see *Section 4.5. Exception Definitions*.

- Miscellaneous registers
    - Time Base (TB). This register is a 64-bit structure provided for maintaining the time of day and operating interval timers. The TB consists of two 32-bit registers—time base upper (TBU) and time base lower (TBL). The time base registers can be written to only by supervisor-level software, but can be read by both user and supervisor-level software. See *Section 2.1.2.3. Time Base and Decrementer (TB, DEC)* for more information.

      Implementation Note—In the 970FX, the time base register is incremented once every eight full frequency processor clocks. Alternatively, when HID0[19] is set to '1', the time base register is incremented at the timebase enable input pin (TBEN) input frequency.
    - Decrementer Register (DEC). This register is a 32-bit decrementing counter that provides a mechanism for causing a decrementer exception after a programmable delay. See *Section 2.1.2.3. Time Base and Decrementer (TB, DEC)* for more information.

      Implementation Note—In the 970FX, the decrementer register is decremented once every eight full frequency processor clocks. Alternatively, when HID0[19] is set to '1', the decrementer register is decremented at the TBEN input frequency.
    - Processor ID Register (PIR). The PIR register is used to differentiate between individual processors in a multiprocessor environment. See *Section 2.1.2.4. Processor ID Register (PIR)* for more information.
- Performance Monitor Registers. The following registers are used to define and count events for use by the performance monitor:
    - The Performance Monitor Counter Registers (PMC1–PMC8) are used to record the number of times a certain event has occurred. See *Section 2.1.3.6. Performance Monitor Registers (MMCR0, MMCR1, MMCRA, PMC1-8)* for more information.
    - The Monitor Mode Control Registers (MMCR0, MMCR1, MMCRA) are used to identify what events will be monitored and to enable various performance monitor interrupt functions. See *Section 2.1.3.6. Performance Monitor Registers (MMCR0, MMCR1, MMCRA, PMC1-8)* for more information.
    - The Sampled Instruction Address Register (SIAR) contains the effective address of an instruction executing at or around the time that the processor signals the performance monitor interrupt condition. See *Section 2.1.3.7. Sampled Instruction Address and Sampled Data Address Registers (SIAR, SDAR)* for more information.
    - The Sampled Data Address Register (SDAR) contains the effective address of the storage access instruction executing at or around the time that the processor signals the performance monitor interrupt condition. See *Section 2.1.3.7. Sampled Instruction Address and Sampled Data Address Registers (SIAR, SDAR)* for more information.

- **970FX-specific registers**—The PowerPC Architecture allows implementation-specific SPRs. The following are incorporated in the 970FX:

  Note: In the 970FX, these registers are all supervisor-level registers.

  – Hardware Implementation-Dependent Register 0 (HID0). This register controls various functions, such as enabling checkstop conditions, locking, enabling, invalidating the instruction and data caches, power modes, and others. See *Section 2.1.3.2. HID Registers (HID0, HID1, HID4, and HID5)* for more information.

  – Hardware Implementation-Dependent Register 1 (HID1). The HID1 register contains additional mode bits that are related to the instruction fetch and instruction decode functions in the 970FX. See *Section 2.1.3.2. HID Registers (HID0, HID1, HID4, and HID5)* for more information.

  – Hardware Implementation-Dependent Register 4 (HID4).The HID4 register contains bits related to the load-store function in the 970FX. See *Section 2.1.3.2. HID Registers (HID0, HID1, HID4, and HID5)* for more information.

  – Hardware Implementation-Dependent Register 5 (HID5).The HID5 register contains bits related to the load-store function in the 970FX. See *Section 2.1.3.2. HID Registers (HID0, HID1, HID4, and HID5)* for more information.

  – Data Address Breakpoint Register (DABR) and the data address breakpoint register extension (DABRX). The DABR controls the data address breakpoint mechanism, which provides a means of detecting load and store accesses to a designated doubleword. See *Section 2.1.3.3. Data Address Breakpoint Register (DABRX)* for more information.

  – Scan Communications Register (SCOMC). The SCOMC register is a control register that includes a command field, a destination field, and a set of status bits. See *Section 2.1.3.8. Scan Communication Registers (SCOMC and SCOMD)* for more information.

  – Scan Communications Register (SCOMD). The SCOMD register is an associated data register that acts as either a source of data or as a destination for data depending on the command placed into the SCOMC register. See *Section 2.1.3.8. Scan Communication Registers (SCOMC and SCOMD)* for more information.

  – Instruction Match Cam Registers (IMC). The IMC SPRs are used to access the IMC array which contains the mask values used for instruction matching. The Move To/From IMC (**mt**/**fimc)** instructions can be executed only in supervisor mode. See *Section 2.1.3.5. Instruction Match CAM Array Access Register (IMC)* for more information.

  – Trigger Registers (TRIG0-TRIG2). Writes to the trigger registers, named TRIG0, TRIG1, and TRIG2, can be inserted in the instruction stream to cause triggers to the on-chip trace array debug logic. These are intended to be used for debug and bring-up only and architecturally behave as a no-op. See *Section 2.1.3.9. Trigger Registers (TRIG0, TRIG1, TRIG2)* for more information.

  – Hardware Interrupt Offset Register (HIOR). The HIOR is used for interrupt vector relocation. See *Section 2.1.4.2. Hardware Interrupt Offset Register (HIOR)* for more information.

Note: While it is not guaranteed that the implementation of 970FX-specific registers is consistent among PowerPC processors, other processors may implement similar or identical registers.

2.1.2 Architected Registers in the 970FX Implementation

Several architected registers are implemented in the 970FX in a way that varies from, or extends, the definition in the PowerPC Architecture.

2.1.2.1 MSR Register (MSR)

The PowerPC Architecture describes the MSR bits [2], [4:47], [57], [60] and [63] as either *optional* or *reserved*. In the 970FX, bit [38] is used as the "VPU Available" enable and bit [45] is used as the power management (POW) enable; the other bits are not implemented and will return the value '0' when read.

Note: Little-endian mode is not supported (MSR[LE] and MSR[ILE] are treated as reserved).

Implementation Note—*Table 2-1* describes MSR bits that the 970FX implements which deviate from the PowerPC Architecture.

Table 2-1. MSR Bits

Bit	Name	Description
3	HV	Hypervisor state. 0 The processor is not in hypervisor state. 1 If MSR[PR] = '0' the processor is in hypervisor state; otherwise the processor is not in hypervisor state.
38	VP	Vector processor available. 0 The processor prevents execution of all vector instructions, including loads, stores, and moves. If such execution is attempted, a VPU unavailable exception is raised. 1 The processor can execute all vector instructions. The VRSAVE register is not protected by MSR[VP]. The data streaming family of instructions (**dst**, **dstt**, **dstst**, **dststt**, **dss**, and **dssall**) are not affected by the MSR[VP].
45	POW	Activates power management. The 970FX will clear the POW bit when it leaves a power saving mode. See *Chapter 9, Power and Thermal Management* for more information.
47	—	Reserved. The ILE bit is not implemented in the 970FX.
48	EE	External interrupt enable 0 The processor delays recognition of external interrupts and decrementer exception conditions. 1 The processor is enabled to take an external interrupt or the decrementer exception. **Note:** Setting MSR[EE] masks not only the architecture-defined external interrupt and decrementer exceptions, but also the 970FX-specific instrumentation, debug, performance monitor, and thermal exceptions.
49	PR	Problem state. 0 The processor is in privileged state. 1 The processor is in problem state.
63	—	Reserved. The LE bit is not implemented in the 970FX.

The MSR bits [HV, PR] determine the privledge level of the processor. Bit [3] is the HV state and bit [49] is the problem state. The following table describes the allowed processor states.

Table 2-2. Processor Allow States

HV	PR	Processor State
0	0	Priviledged
0	1	Problem
1	0	hypervisor
1	1	Problem

MSR[HV] can be set to ‘1’ only by the System Call instruction (**sc**) and some interrupts. It can be set to ‘0’ only by the **rfid** instruction and by some interrupts.

2.1.2.2 Machine Status Save/Restore Register (SRR1)

This register is used to save machine status during interrupts. In the 970FX, SRR1 bits [1:2], [4:32], [37], [39:41], [47], [56:57], [60], and [63] are treated as reserved. These bits are not implemented and will return the value ‘0’ when read. See *Section 4.3.2. Machine Status Save/Restore Register 1 (SRR1)* for additional information.

Table 2-3. Additional SRR1 Bit

Bit	Function
33	SIAR and SDAR contents synchronized.

2.1.2.3 Time Base and Decrementer (TB, DEC)

The time base counter and the decrementer are clocked at 1/8 (one-eighth) of the full frequency processor. The 970FX supports two modes of operation (controlled by HID0[19] and the time base enable input pin) for updating the time base and decrementer. When HID0[19] is zero, then the counters constantly update as long as the timebase_enable input pin (TBEN) is high (traditional mode of operation). When HID0[19] is one, the counters update only on the rising edge of the TBEN input pin.

When the processor is “stopped” (due to various breakpoint, debug and support processor functions), an additional mode bit (HID0[18]) determines whether or not the time base and the decrementer continue counting. Note that some support processor operations require the use of an alternate clocking mode for scan, and in these cases, the time base and the decrementer will not continue counting.

2.1.2.4 Processor ID Register (PIR)

The processor identification register (PIR) is a 32-bit register that holds a processor identification tag (PID). In the 970FX, this tag is in the three least significant bits (29:31). This tag is used to tag bus transactions and to differentiate processors in multiprocessor systems. The format of the register is as follows:

Figure 2-2. Processor Identification Register

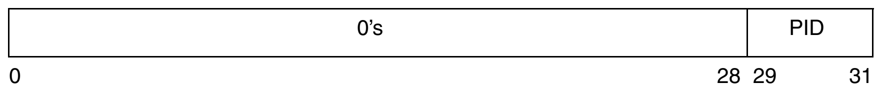


Table 2-4. PIR Register

Bits	Name	Description
0:28	–	Reserved (read as zeros)
29:31	PID	3-bit processor ID value

The PIR is a read only register. During power-on reset, *PID* is set to a unique value for each processor in a multi-processor system. For more information about the power-on reset configuration process, see *Chapter 11, System Design*.

2.1.3 PowerPC 970FX-Specific Registers

This section describes registers that are defined for the 970FX, but are not included in the PowerPC Architecture.

2.1.3.1 Move To/From System Register Instructions

The 970FX defines several new implementation specific system registers. Note that some of these registers are also user-mode readable through a second set of SPR encodings; and that some of these registers have special software synchronization requirements.

The encoded SPR values for these implementation specific registers are shown in *Table 2-5* on page 43. Note that the SPR is encoded in the **mfspr** and **mtspr** instructions such that bits 5:9 of the SPR field represent the five high-order bits of the SPR number, and bits 0:4 of the SPR field represent the five low-order bits of the SPR number.

Table 2-5. Implementation-Specific SPRs (Page 1 of 2)

SPR			Register Name	R/W	Synchronization Requirements		
Decimal (supervisor)	Decimal (user)	SPR(5:9) SPR(0:4)			Before Reads	After Writes	Before Writes
1023		11111 11111	PIR	R	none	N/A	N/A
1013		11111 10101	DABR	R/W	none	CSI	sync
1015		11111 10111	DABRX	R/W			
1008		11111 10000	HID0	R/W	none	Note 1	Note 1
1009		11111 10001	HID1	R/W	none	Note 2	Note 2
1012		11111 10100	HID4	R/W	none	Note 3	Note 3
1014		11111 10110	HID5	R/W	none	Note 4	Note 4
795	779	11000 n1011	MMCR0	R/W	none	Note 5	Note 5
798	782	11000 n1110	MMCR1	R/W	none	Note 5	Note 5
786	770	11000 n0010	MMCRA	R/W	none	Note 5	Note 5
787	771	11000 n0011	PMC1	R/W	sync	none	none
788	772	11000 n0100	PMC2	R/W	sync	none	none
789	773	11000 n0101	PMC3	R/W	sync	none	none
Note: For **mtspr**, n must be '1'. For **mfspr**, reading the SPR is privileged if and only if n = '1'.							

Table 2-5. Implementation-Specific SPRs (Page 2 of 2)

SPR			Register Name	R/W	Synchronization Requirements		
Decimal (supervisor)	Decimal (user)	SPR(5:9) SPR(0:4)			Before Reads	After Writes	Before Writes
790	774	11000 n0110	PMC4	R/W	sync	none	none
791	775	11000 n0111	PMC5	R/W	sync	none	none
792	776	11000 n1000	PMC6	R/W	sync	none	none
793	777	11000 n1001	PMC7	R/W	sync	none	none
794	778	11000 n1010	PMC8	R/W	sync	none	none
276		01000 10100	SCOMC	R/W	none	CSI	none
277		01000 10101	SCOMD	R/W	none	CSI	none
796	780	11000 n1100	SIAR	R/W	sync	none	none
797	781	11000 n1101	SDAR	R/W	sync	none	none
799	783	11000 n1111	IMC	R/W	none	CSI	none
976		11110 10000	TRIG0	W	N/A	none	none
977		11110 10001	TRIG1	W	N/A	none	none
978		11110 10010	TRIG2	W	N/A	none	none
256		01000 00000	VRSAVE	R/W	N/A	none	none
304		01001 10000	HSPRG0	R/W	none	none	none
305		01001 10001	HSPRG1	R/W	none	none	none
311		01001 10111	HIOR	R/W	none	none	none
314		01001 11010	HSRR0	R/W	none	none	none
315		01001 11011	HSRR1	R/W	none	none	none
Note: For **mtspr**, n must be '1'. For **mfspr**, reading the SPR is privileged if and only if n = '1'.							

Notes: (*for Table 2-5. Implementation-Specific SPRs*)

1. The following sequence must be used when modifying HID0:
   sync
   mtspr HID0,Rx
   isync
2. The following sequence must be used when modifying HID1:
   mtspr HID1,Rx
   mtspr HID1,Rx
   isync
   Executing two **mtspr** instructions is necessary to ensure that updates to all portions of HID1 will be complete before the **isync** instruction completes.
3. The following sequence must be used when modifying HID4:
   sync
   mtspr HID4,Rx
   isync
   When HID4[23] is changed, the above sequence should be preceded by a **mtsr** and **sync** which will cause the ERATs to be flushed.

4. The following sequence must be used when modifying HID5:
   sync
   mtspr HID5,Rx
   isync
   Whenever HID5[56] or HID5[57] is changed, the entire instruction cache must be flushed to insure that any succeeding **dcbz** is executed in the context of the new HID5 bit settings.
5. Although it is not necessary to use synchronizing instructions when modifying the MMCR(0,1,A) registers, it is recommended that the following sequence be used:
   sync
   mtspr MMCRz,Rx
   isync

Table 2-6 describes the 970FX's behavior for the **mtspr** and **mfspr** instructions.

Table 2-6. Move To / Move From SPR Behavior

Condition				Resulting Action
SPR		MSR[PR]	R/W	
SPR(0)	Register			
1	Any invalid SPR encoding	0	**mfspr**	No-op (target register is unchanged)
1	Any invalid SPR encoding	0	**mtspr**	No action (write is inhibited)
1	ACCR, ASR, CTRL, DABR, DAR, DEC, DSISR, HID0, HID1, HID4, HID5, IMC, SCOMC, SCOMD, SDR1, SDAR, SIAR, SRR0, SRR1, SPRG0, SPRG1, SPRG2, SPRG3, TBL, TBU, Performance Monitor Registers	0	**mfspr**	Returns value to GPR
		0	**mtspr**	Target SPR is updated
1	TRIG0, TRIG1, TRIG2	0	**mfspr**	Causes an illegal instruction type resulting in a program interrupt
		0	**mtspr**	Causes a trigger to trace array debug logic
1	PIR	0	**mfspr**	Returns value to GPR
		0	**mtspr**	Causes an illegal instruction type resulting in a program interrupt
1	Any SPR encoding (with SPR(0) = '1')	1	**mtspr** **mfspr**	Causes a privileged instruction type resulting in a program interrupt
0	Any invalid SPR encoding except: SPR(0:9) = 00000 00000 SPR(0:9) = 00100 00000 SPR(0:9) = 00101 00000 SPR(0:9) = 00110 00000	X	**mfspr**	No-op (target register is unchanged)
		X	**mtspr**	No action (write is inhibited)
0	SPR(0:9) = 00000 00000 SPR(0:9) = 00100 00000 SPR(0:9) = 00101 00000 SPR(0:9) = 00110 00000	X	**mtspr** **mfspr**	Causes an illegal instruction type resulting in a program interrupt

2.1.3.2 HID Registers (HID0, HID1, HID4, and HID5)

The 970FX includes many implementation dependent mode bits that allow various features of the chip to be enabled and disabled. These bits are included in the Hardware Implementation Dependent registers (HID0, HID1, HID4, and HID5). In general, HID0 attempts to line up the 970FX's modes with the relevant ones from earlier PowerPC implementations and then adds a few new ones. The HID1 register contains additional mode bits that are related to the instruction fetch and instruction decode functions in the 970FX. The HID4 and HID5 registers contain bits related to the load/store function in the 970FX. All of these registers are supervisor resources.

The state of each of the HID registers after a normal scan-based POR is all zeroes. The preferred state of these registers for optimal performance and function is also all zeroes, except where indicated.

Table 2-7. HID0 Bit Functions

Bits	Bit Name	Description
0	one_ppc	"One PowerPC instruction per dispatch group" mode (an instruction may span more than one group)
1	do_single	"Single group completion" mode Flush and refetch after the completion of each group or the completion of each microcoded instruction, if the instruction spans multiple groups.
2	isync_sc	Disable "isync scoreboard" optimization
3	ser-gp	Serialize Group Dispatch (next group not dispatched until previous group completes)
4:8	–	Reserved
9	nap	Nap
10	–	Reserved
11	dpm	Enable dynamic power management
12	–	Reserved
13	tg	Performance monitor threshold granularity control
14	hang_dis	Disable processor hang detection mechanism
15	nhr	"Not Hard Reset" (check after SRI to see if hard or soft)
16	inorder	"Serialized group issue" mode. The next group is not issued until the previous group completes. Does not include branch or CR-logical instructions.
17	–	Reserved
18	tb_ctrl	Enable time base counting when processor is "stopped"
19	ext_tb_en	External time base enable 0: Use TBEN input as enable (TB clocked at 1/8 full processor frequency) 1: Use TBEN input to clock time base (external clock)
20:21	–	Reserved
22	ciabr_en	Enable CIABR (Completion Instruction Address Breakpoint Register)
23	hdice_en	Enable hypervisor decrementer interrupt conditionally (HDICE). The initial reset value must be '0' and disables hypervisor interrupts.
24	en_therm	Enable external thermal interrupts
25:30	–	Reserved
31	en_attn	Enable support processor attention instruction
32	en_mck	Enable external machine check interrupts (preferred state = '1')
33:63	–	Reserved

Table 2-8. HID1 Bit Functions

Bits	Bit Name	Description
0:2	bht_pm	Branch history table (BHT) prediction mode 000 Static prediction 001 Unused (same as 000) 010 Global BHT prediction only 011 Global prediction with history compression 100 Local BHT prediction only 101 Unused (same as 100) 110 Full global/local prediction with global selection (gsel) 111 Full global/local prediction with gsel and history compression (preferred state)
3	en_ls	Enable link stack (preferred state = 0b1)
4	en_cc	Enable count cache (preferred state = 0b1)
5	en_ic	Enable instruction cache (preferred state = 0b1)
6	–	Reserved
7:8	pf_mode	Prefetch mode: 00 No instruction prefetch 01 Select next sequential address (NSA) instruction prefetch 10 Select NSA and NSA+1 instruction prefetch (preferred state) 11 Disable prefetch buffer
9	en_icbi	Enable “forced icbi match” mode
10	en_if_cach	Enable instruction fetch cacheability control 0 All instruction fetch accesses are treated as cache inhibited (regardless of the state of the page table I-bit). 1 Instruction fetch cacheability is controlled by the state of the page table I-bit. (preferred state)
11	en_ic_rec	Enable I-cache parity error recovery (preferred state = 0b1)
12	en_id_rec	Enable I-directory parity error recovery (preferred state = 0b1)
13	en_er_rec	Enable I-ERAT parity error recovery (preferred state = 0b1)
14	ic_pe	Force instruction cache parity error (error inject)
15	icd0_pe	Force instruction cache directory 0 parity error (error inject)
16	–	Reserved
17	ier_pe	Force I-ERAT parity error (error inject)
18	en_sp_itw	Enable speculative tablewalks. The ERAT is never loaded using a PTE if PTE[G] = ‘1’. (preferred state = ‘1’)
19:63	–	Reserved

Table 2-9. HID4 Bit Functions (Page 1 of 2)

Bits	Bit Name	Description
0	lpes1	LPAR environment selector bit [0]. LPES[0:1] are located in HID4[57, 0]. LPES[0:1] determine how MSR[HV] is set using interrupts and how memory access is performed when not in hypervisor mode. This is described in the PowerPC Architecture version 2.01.
1:2	rmlr(1:2)	LPAR real mode limit register (see HID4[58] for bit [0]).
3:6	lpid(2:5)	LPAR partition identity bits [2:5] (see also bits [62:63] for lpid(0:1)).
7:22	rmor(0:15)	LPAR real mode offset register [0:15].
23	rm_ci	The real mode caching inhibited bit can be used to permit a control register on an I/O device to be accessed without permitting the corresponding storage location to be copied into the caches. The bit should normally contain ‘0’. Software would set the bit to ‘1’ just before accessing the control register, access the control register as needed, and then set the bit back to ‘0’.
24	force_ai	Force alignment interrupt instead of microcode on unaligned operations
25	dis_pref	Disables data prefetching.
26	res_pref	Setting HID4[26] = ‘1’ resets the data prefetch mechanism, suppressing subsequent prefetch requests and clearing the stream detection logic so that stream detection will not be affected by accesses performed prior to setting the bit back to ‘0’.
27	en_sp_dtw	Enable speculative load tablewalk.
28	l1dc_flsh	L1 data cache flash invalidate 0 Normal operation 1 All sectors set to invalid and held invalid
29:30	dis_derpc	Disable D-ERAT parity checking [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]
31	dis_derpg	Disable D-ERAT parity generation (force parity to ‘0’ on EA[0:45] only)
32:33	dis_derat	Disable D-ERAT [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]; valid states 00,01,10
34:35	dis_dctpc	Disable data cache tag parity checking [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]
36	dis_dctpg	Disable data cache tag parity generation
37:38	dis_dcset	Disable data cache set [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]
39:40	dis_dcpc	Disable data cache parity checking [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]
41	dis_dcpg	Disable data cache parity generation
42:43	dis_dcrtpc	Disable data cache real address tag parity checking
44:47	dis_tlbpc	Disable TLB parity checking [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]
48	dis_tlbpg	Disable TLB parity generation
49:52	dis_tlbset	Disable TLB set [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]; valid states X‘0’,X‘7’,X‘B’,X‘D’,X‘E’
53	dis_slbpc	Disable SLB parity checking.
54	dis_slbpg	Disable SLB parity generation.
55	mck_inj	Machine check error inject enable
56	dis_stfwd	Disable store forwarding (cause reject)

Table 2-9. HID4 Bit Functions (Page 2 of 2)

Bits	Bit Name	Description
57	lpes0	LPAR environment selector bit. LPES[0:1] are located in HID4[57, 0]. LPES[0:1] determine how MSR[HV] is set using interrupts and how memory access is performed when not in hypervisor mode. This is described in the PowerPC Architecture version 2.01.
58	rmlr0	HID4 bits [58, 1:2] are real mode limit register bits [0:2]. 011 64 MB 111 128 MB 100 256 MB x10 1 GB x01 16 GB 000 256 GB
59	–	Reserved
60	dis_splarx	Disable speculative **lwarx**, **ldarx**
61	lg_pg_dis	Disable large page support; Large page (L) bit input to the SLB will be forced to zero (software will read a zero L-bit)
62:63	lpid(0:1)	LPAR partition identity bits [0:1]. HID4[62:63, 3:6] are LPID[0:5] respectively.

Table 2-10. HID5 Bit Functions (Page 1 of 2)

Bits	Bit Name	Description
0:31	–	Reserved
32:47	hrmor(0:15)	LPAR hypervisor real mode offset register
48:49	–	Reserved
50	DC_mck	Machine check enabled for data cache and data cache tag parity errors (software recovery enabled).
51	dis_pwrsave	L1 data cache, L1 D-cache Tag, D-ERAT power savings disable
52	force_G	Force guarded (G = '1') load
53	DC_repl	Data cache replacement algorithm: 0 default - Least Recently Used (LRU) 1 FIFO
54	hwr_stms	Number of available hardware prefetch streams: 0 4 hardware streams and 4 vector streams 1 8 hardware streams (HID5[55] must also be 0b1)
55	dst_noop	Data StreamTouch (DST) instructions no-op 0 DST's are enabled 1 DST's are a no-op and discarded in the Load Store Unit (LSU)
56	DCBZ_size	Makes **dcbz** a 32-byte store when **dcbz** instruction bit [10] = '0'
57	DCBZ32_ill	Makes a **dcbz** instruction with bit [10] = '0' an illegal instruction
58	tlb_map	TLB Mapping: 0 4 way set associative 1 direct mapped **Note:** When setting HID5[58] to setup the TLB to be direct mapped, HID4[49:52] (TLB set disable bits) must be cleared, otherwise translation will not work.

Table 2-10. HID5 Bit Functions (Page 2 of 2)

Bits	Bit Name	Description
59	lmq_port	Demand miss (LMQ to STS) 0 permit two per cycle 1 permit only one per cycle
60	lmq_size(0)	Number of outstanding requests to STS (970FX storage subsystem) HID5[60, 63] - maximum outstanding requests 00 8 01 1 10 2 11 4
61	–	Reserved
62	tch_nop	Make **dcbt** and **dcbtst** act like no-ops
63	lmq_size(1)	See description of HID5[60]

2.1.3.3 Data Address Breakpoint Register (DABRX)

The data address breakpoint facility provides a means of detecting load and store accesses to a designated doubleword. The address comparison is done on an effective address. The data address breakpoint facility is controlled by the architected data address breakpoint register (DABR) and the 970FX specific data address breakpoint register extension (DABRX).

Table 2-11. Data Address Breakpoint Register Extension (DABRX)

Bits	Name	Description
0:59	–	Reserved
60	BTI	Breakpoint translation ignore
61	HYP	Hypervisor state
62	PNH	Privileged but non-hypervisor state
63	PRO	Problem state (user mode)
Note: Bits [61:63] are termed the privileged mask (PRIVM).		

Data Address Compare

The 970FX supports the address compare control facility and the Address Compare Control Register (ACCR) as defined in the architecture. In addition, the 970FX includes support for the optional Data Address Breakpoint facility and associated DABR register described in the architecture. In either case, upon taking the data storage interrupt, the 970FX will set the DAR correctly.

The architecture allows some flexibility on whether or not an ACCR match and/or a DABR match actually occurs for certain conditions. More specifically, in the 970FX, store conditional instructions that are executed but not successful (that is, the store does not actually occur) will cause either an ACCR match or a DABR match if the appropriate match conditions are met. String instructions with zero length will not cause ACCR or DABR matches. The **dcbz** instruction will cause a DABR match if the appropriate match conditions are met.

As an alternative to causing an interrupt, a DABR match can be made to cause various forms of "hard stops" or "soft stops" for use as a debug aid (these controls are available through special SCOM commands). In general, this capability is not recommended for use in normal system operation since it may require the presence of an engineering support processor to restart the CPU.

2.1.3.4 Instruction Address Breakpoint Register (IABR)

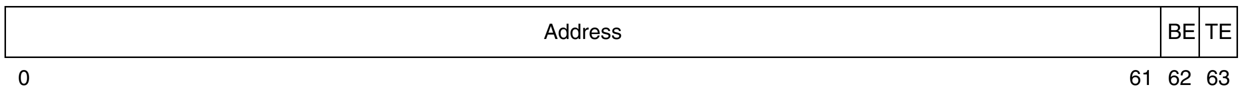
The Instruction Address Breakpoint Register (IABR), shown in *Figure 2-3*, can be used as a debug tool to trigger an event upon the fetch of a particular instruction address. The address in the IABR is compared to the Instruction Fetch Address Register (IFAR), which will also contain addresses of speculative instruction fetches. The IABR is set up as described in the *PowerPC Microprocessor Family: The Programming Environments* manual, except in the 970FX, the IABR is only available as a trigger to the debug logic. This trigger can be programmed to perform functions such as quiesce or checkstop. If the word specified in the IABR is fetched, the instruction breakpoint handler is invoked. The instruction that triggers the breakpoint does not execute before the handler is invoked.

CIABR can be enabled by either HID0[22] (software accessible) or scan/scom override.

The IABR uses the IFU fetch address, not the CIA (current instruction address that is executing). An IABR match occurs on the fetch of an instruction, even speculative.

Note: There can be multiple IABR matches for a single instruction before it is actually executed (or completed).

Figure 2-3. Instruction Address Breakpoint Register



The IABR bits are described in *Table 2-12. Instruction Address Breakpoint Register Bit Settings*. During power-on reset all bits are reset to '0'.

Table 2-12. Instruction Address Breakpoint Register Bit Settings

Bits	Name	Description
0:61	Address	Word address to be compared
62	BE	Breakpoint enabled. (Address match causes trigger to debug logic.)
63	TE	Translation enabled. An IABR match is signaled if this bit matches MSR[IR].

2.1.3.5 Instruction Match CAM Array Access Register (IMC)

The Instruction Match Cam (IMC) array facility is used for performance monitoring instrumentation . The array has privileged write access and user-level read access through this SPR. Writes to the register array are used to configure the IMC, and reads return information about the availability of registers within the facility.

2.1.3.6 Performance Monitor Registers (MMCR0, MMCR1, MMCRA, PMC1-8)

The performance monitor counter registers (PMC1 - PMC8), the performance monitor control registers (MMCR0, MMCR1, MMCRA), and the sampled address registers (SIAR, SDAR) are supported in the 970FX.

The performance monitor control registers MMCR0, MMCR1, and MMCRA are used in conjunction with the MSR and other SPRs to set up the performance monitor enable states, interrupt conditions, threshold values, match criteria, and selection of the events counted in each of the counter registers PMC1 - PMC8.

The MMCRx register bit assignments are shown in *Table 10-2. Performance Monitor Control Register MMCR0*; *Table 10-3. Performance Monitor Control Register MMCR1*; and *Table 10-4. Performance Monitor Control Register MMCRA*. All of the MMCRx and PMCx registers flush to zero unless otherwise noted in the MMCRx and PMCx tables.

The MSR bits that relate to performance monitor functions are shown in *Table 4-5. MSR Bit Settings* on page 93. The value of the SRR1 registers when a performance monitor interrupt is taken is shown in *Chapter 10, 970FX Performance Monitor*.

Table 2-13. Performance Monitor Count Registers (PMC1 - 8)

Bits	Bit Name	Bit Description
0	CTR_NEG	Counter negative bit
1:31	CTRDATA	Count data

2.1.3.7 Sampled Instruction Address and Sampled Data Address Registers (SIAR, SDAR)

The Sampled Instruction Address register (SIAR) and the Sampled Data Address register (SDAR) are used, respectively, to save the effective address of a sampled instruction and the effective address of a storage operand for a sampled instruction when the processor is in either trace-marking mode or performance-marking mode. The terms ‘sampled’ and ‘marked’ are used interchangeably in this manual.

Table 2-14. Sampled Instruction Address Register (SIAR) Performance Monitor Related

Bits	Bit Name	Bit Description
0:63	–	Sampled Instruction Address

Table 2-15. Sampled Data Address Register (SDAR) Performance Monitor Related Bits

Bits	Bit Name	Bit Description
0:63	–	Sampled Data Address

2.1.3.8 Scan Communication Registers (SCOMC and SCOMD)

The 970FX includes a pair of registers to aid in communicating with the Scan Communications facility (SCOM). The SCOMC register is a control register that includes a command field, a destination field, and a set of status bits. The SCOMD register is an associated data register that acts as either a source of data or as a destination for data depending on the command placed into the SCOMC register.

The SCOM facility contains an arbiter which serializes use of the facility among the bus masters (processor cores and core service processor), however there are very specific programming conventions associated with the use of this facility.

A detailed description of the SCOM facility can be found in *Section 11.6. SCOM Facility*.

2.1.3.9 Trigger Registers (TRIG0, TRIG1, TRIG2)

Writes to the trigger registers, named TRIG0, TRIG1, and TRIG2, can be inserted in the instruction stream to cause triggers to the on-chip trace array debug logic. These are intended to be used for lab debug and bring-up only and architecturally behave as a no-op.

2.1.4 Logical Partitioning Function Registers

2.1.4.1 Hypervisor Decrementer Interrupt Register (HDEC)

The Hypervisor Decrementer (HDEC) is a 32-bit decrementing counter that provides a mechanism for causing a Hypervisor Decrementer interrupt after a programmable delay.

The Hypervisor Decrementer is driven by the same frequency as the Time Base, in the same manner as the Decrementer. The Hypervisor Decrementer counts down, causing an interrupt and is implemented in SPR 310.

2.1.4.2 Hardware Interrupt Offset Register (HIOR)

The Hardware Interrupt Offset register, HIOR should be scanned (the HIOR is on the mode ring) to the system's starting address during initialization. Subsequently HIOR should be set to zero.

The real address of the interrupt vector is found using HIOR[22:43] combined with the 20-bit vector offset for the particular exception.

Table 2-16. Hardware Interrupt Offset Register (HIOR)

Bits	Bit Description
0:21	Reserved
22:43	Offset
44:63	Reserved

2.1.4.3 Hypervisor Real Mode Offset Register (HRMOR)

The Hypervisor Real Mode Offset Register (HRMOR) is implemented in HID5[32:47]. It is a hypervisor resource, which is used to relocate effective addresses in hypervisor real addressing mode. If MSR[HV] = ‘1’ and the effective address is ‘0’, the contents of HRMOR and the effective address are logically OR’d to form the address.

2.1.4.4 Hypervisor Decrementer Interrupt Conditionally Enable (HDICE)

The HDICE is located in HID0[23]. The value of zero must be the initial reset value and disables hypervisor interrupts.

2.1.4.5 Real Memory Limit Register (RMLR)

The Real Memory Limit Register, RMLR, is implemented in HID4[58,1:2]. The RMLR defines the memory limit for the logical partition above the offset defined by the RMOR. *Table 2-17* lists the supported limit values.

Table 2-17. Memory Limits (HID4[58,1:2])

Memory	Limit Value
64 MB	‘011’
128 MB	‘111’
256 MB	‘100’
1 GB	‘x10’
16 GB	‘x01’
256 GB	‘000’

Note: If MSR[HV] = ‘0’ and LPES1 = ‘1’, the access is controlled by the contents of the Real Mode Limit Register and Real Mode Offset Register.

2.1.4.6 Real Mode Offset Register (RMOR)

The Real Mode Offset Register (RMOR) is implemented in HID4[7:22]. If the access is permitted by the RMLR, the effective address for the access is ORed with the offset represented by the contents of the RMOR and the low-order bits of the result are used as the real address for the access.

Note: If MSR[HV] = ‘0’ and LPES1 = ‘1’, the access is controlled by the contents of the Real Mode Limit Register and Real Mode Offset Register.

2.1.4.7 Hypervisor Save/Restore Register (HSRRO, HSRR1)

The hypervisor machine status save/restore registers (HSRR0, HSRR1) are located in SPR 314 and 315 respectively. When a hypervisor decrementer interrupt occurs, the state of the machine is saved in the hypervisor machine status save/restore registers (HSRR0 and HSRR1). The effective address is storedin HSRR0 and the MSR in HSRR1. The contents of these registers is used to restore machine state when a **hrfid** instruction is executed.

2.1.4.8 Hypervisor SPRGs (HSPRG0, HSPRG1)

HSPRG0 and HSPRG1 are 64-bit registers provided for use by hypervisor programs at SPRs 304 and 305 respectively.

Note: Neither the contents of the HSPRGs, nor accessing them using **mtspr** or **mfspr**, has a side effect on the operation of the processor. One of more of the registers is likely to be needed by hypervisor interrupt handler programs (e.g., as scratch registers and/or pointers to processor save areas).

2.1.4.9 Logical Partitioning Identification Register (LPIDR)

The Logical Partition Identity Register (LPID) is implemented in LPID[0:5]. This register contains a value that identifies the partition to which the processor is assigned. The LPIDR is located in HID4[62,63,3,4,5,6].

For the 970FX implementation, moving a processor from the partition identified by the current contents of the LPIDR to a new partition (for example, altering the contents of the LPIDR) requires ensuring that no **tlbie** or **tlbsync** instruction sequences are being executed in either the current partition or the destination partition.

2.1.4.10 Real Mode Cache Inhibit Bit, RM_CI

The Real Mode Caching Inhibited bit can be used to permit a control register on an I/O device to be accessed without permitting the corresponding memory location to be copied into the cache. The bit should normally contain '0'. Software would set the bit to '1' just before accessing the control register, access the control register as needed, and then set the bit back to '0'. RM_CI is implemented in HID4[23].

2.1.4.11 Logical Partitioning Environment Selector Bits, LPES [0:1]

Logical Partitioning Environment Selector (LPES) bits [0:1] are located in HID4 bits [57,0]. These bits determine how MSR[HV] is set via interrupts and how memory access is performed when not in hypervisor mode.

2.2 Instruction Set Summary

This section describes instructions and addressing modes defined for the 970FX. These instructions are divided into the following execution unit categories:

- Fixed-Point Processor
- Floating-Point Processor
- Vector Processor
- Load and Store Processor
- Branch and Flow Control
- Storage Control
- Memory Synchronization

Fixed-point instructions operate on byte, half-word, word, and double-word operands. Floating-point instructions operate on single-precision and double-precision floating-point operands. The PowerPC Architecture uses instructions that are four bytes long and word-aligned. It provides for byte, half-word, word and double-word operand loads and stores between memory and a set of 32 general-purpose registers (GPRs). It provides for word and double-word operand loads and stores between memory and a set of 32 floating-point registers (FPRs). The vector processing unit extends the PowerPC Architecture and provides for quad-word operand loads and stores between memory and a set of 32 vector registers.

Arithmetic and logical instructions do not read or modify memory. To use the contents of a memory location in a computation and then modify the same or another memory location, the memory contents must be loaded into a register, modified, and then written to the target location using load and store instructions.

2.2.1 Classes of Instructions

The 970FX instructions belong to one of the following three classes.

- Defined
- Illegal
- Reserved

Note: While the definitions of these terms are consistent among the PowerPC processors, the assignment of these classifications is not.

The class is determined by examining the primary opcode and the extended opcode, if any. If the opcode, or combination of opcode and extended opcode, is not that of a defined instruction or of a reserved instruction, then the instruction is illegal. Instruction encodings that are now illegal may become assigned to instructions in the architecture or may be reserved by being assigned to processor-specific instructions.

2.2.1.1 Definition of Boundedly Undefined

If instructions are encoded with incorrectly set bits in the reserved fields, the results of execution can be said to be boundedly undefined. If a user-level program executes the incorrectly coded instruction, the resulting undefined results are bounded in that a spurious change from user to supervisor state is not allowed, and the level of privilege exercised by the program in relation to memory access and other system resources cannot be exceeded. Boundedly-undefined results for a given instruction may vary between implementations, and between execution attempts in the same implementation.

2.2.1.2 Defined Instructions

The 970FX provides support for the following optional instructions:

- **fsqrt** - Floating-point square root
- **fsqrts** - Floating-point square root single
- **fres** - Floating-point reciprocal estimate single
- **frsqrte** - Floating-point reciprocal square root estimate A-form
- **fsel** - Floating-point select
- **mfsr** - Move from segment register
- **mfsrin** - Move from segment register indirect
- **mtmsr** - Move to machine state register (32-bit)
- **mtsr** - Move to segment register
- **mtsrin** - Move to segment register indirect
- **slbie** - SLB invalidate entry
- **slbia** - SLB invalidate all
- **tlbie** - TLB invalidate entry
- **tlbsync** - TLB synchronize

The 970FX *does not* provide support for the following optional or obsolete instructions (or instruction forms). Attempted use of these will result in an *illegal instruction type program interrupt*.

- **bccbr** - Branch conditional to CBR (obsolete)
- **dcba** - Data cache block allocate (obsolete)
- **dcbi** - Data cache block invalidate (obsolete)
- **eciwx** - External control in word indexed
- **ecowx** - External control out word indexed
- **mcrxr** - Move to condition register from XER register (obsolete)
- **mtsrd** - Move to segment register doubleword (obsolete)
- **mtsrdin** - Move to segment register doubleword indirect (obsolete)
- **rfi** - Return from interrupt (obsolete)
- **tlbia** - TLB invalidate all
- **tlbiex** - TLB invalidate entry by index (obsolete)
- **slbiex** - SLB invalidate entry by index (obsolete)

2.2.1.3 Illegal Instructions

Illegal instructions can be grouped into the following categories:

- Instructions not defined in the PowerPC Architecture.The following primary opcodes are defined as illegal, but may be used in future extensions to the architecture: 1, 4, 5, 6, 56, 57, 60, 61
- Instructions defined in the PowerPC Architecture but not implemented in a specific PowerPC implementation. For example, instructions that can be executed on 64-bit PowerPC processors are considered illegal by 32-bit processors.
  The following opcodes are illegal in 32-bit implementations (such as the PowerPC 750), however they will be executed in 32-bit mode on a 64-bit processor such as the 970FX:
  2, 30, 58, 62
- All unused extended opcodes for instructions are illegal. Notice that extended opcodes for instructions defined only for 64-bit implementations are illegal in 32-bit implementations. The following primary opcodes have unused extended opcodes: 19, 30, 31, 56, 57, 59, 60, 61, 62, 63 (Primary opcodes 30 and 62 are illegal for 32-bit implementations, but as 64-bit opcodes they have some unused extended opcodes.)
- An instruction consisting entirely of zeros is illegal, and is guaranteed to be an illegal instruction. This increases the probability that an attempt to execute data or uninitialized memory invokes the system illegal instruction error handler (a program exception).

See *Section 4.5.9 Program Exception* on page 101 for additional information about illegal and invalid instruction exceptions. Except for an instruction consisting of binary zeros, illegal instructions are available for additions to the PowerPC Architecture.

2.2.1.4 Reserved Instructions

The PowerPC Architecture breaks the reserved instruction class down into several categories. The 970FX behaves in the following manner with respect to categorizing the following reserved instructions.

- Primary opcode equals zero. The 970FX will take an illegal instruction type of program interrupt for all cases except the Support Processor Attention (**attn)** instruction when HID0[31] is set to '1'.
- POWER Architecture instructions not in the PowerPC Architecture. The 970FX will take an illegal instruction type of program interrupt.
- Implementation specific instructions used to conform to the architecture. No action taken.
- Other instructions. The 970FX supports the implementation specific instruction **tlbiel**, (the processor local form of the TLB Invalidate entry used for managing TLB parity errors).

In addition, there are several implementation specific registers available for access through the **mtspr** and **mfspr** instructions. These are described in *Section 2.1.3.1 Move To/From System Register Instructions* on page 43.

2.2.2 Instruction Set Overview

The following sections provide a brief overview of the PowerPC instructions implemented in the 970FX microprocessor and highlight how a 970FX microprocessor implements a particular instruction. Note that the categories used in this section correspond to those used in *Chapter 4, "Addressing Modes and Instruction Set Summary"* in the *PowerPC Microprocessor Family: The Programming Environments* manual. These categorizations are somewhat arbitrary and are provided for the convenience of the programmer and do not necessarily reflect the PowerPC Architecture specification.

Note that some instructions have the following optional features:

- CR Update—The dot (**.**) suffix on the mnemonic enables the update of the CR.
- Overflow option—The **o** suffix indicates that the overflow bit in the XER is enabled.

2.2.3 Fixed-Point Processor

2.2.3.1 Fixed-Point Arithmetic and Compare Instructions

The architecture states that instructions that have the overflow exception [OE] bit set, or instructions that can set the carry [CA] bit, may execute more slowly than instructions that do not. In the 970FX the [SO] bit in the XER is not renamed. For instructions with the [OE] set, it is initially assumed that no overflow will occur and that the SO bit does not need to be changed. In the event that the instruction does cause an overflow and the SO bit was not set before the instruction executed (and therefore needs to be set), the machine will flush this instruction and those beyond this instruction, set the non-renamed [SO] bit, and then refetch and re-execute the instructions that follow. In general, if no overflow occurs or the [SO] bit has already been set, then this strategy will not have an adverse effect on performance.

On the other hand, most instructions that set and use the [CA] bit do not have any particular performance considerations. This field of the XER is renamed, and many of the common dependence hazards are minimized.

2.2.3.2 Fixed-Point Logical, Rotate, and Shift Instructions

The architecture defines the *preferred no-op* to be OR Immediate (**ori)** 0,0,0. In the 970FX microprocessor, this no-op form is recognized by the hardware and allowed to complete without taking any execution resources. This makes the instruction valuable for padding other instructions to achieve better alignment or better separation.

2.2.3.3 Move to and Move from System Register Instructions

The **mtspr** and **mfspr** instructions provide access to system registers using a GPR as the source or destination register, respectively. *Table 2-5. Implementation-Specific SPRs* lists the SPR numbers for both user-level and supervisor-level access to 970FX-specific registers.

2.2.3.4 Move to and Move from Machine State Register

The 970FX microprocessor supports both the 32-bit **mtmsr** instruction and the 64-bit **mtmsrd** instruction. The 970FX works to optimize the **mtmsr** instruction to help speed up the cases where little or no synchronization is required (for example, updates to the MSR[EE] bit).

2.2.3.5 Fixed-Point Invalid Forms and Undefined Conditions

The results of executing an invalid form of a fixed-point instruction or an instance of a fixed-point instruction for which the architecture specifies that some results are undefined are described below for the cases in which executing an instruction does not cause an exception. Only results that differ from those specified by the architecture are described in the following list.

- **Instruction with Reserved Fields**
  Bits in reserved fields are ignored; the results of executing an instruction in which one or more reserved bits are '1' is the same as if the bits were '0'.
- **divw**, **divwo**, **divwu, and divwuo Instructions**
  rD[0:31] is set to x'00000000'.
- **mulhw and mulhwu Instructions**
  Instruction target register, **r**D bits [0:31], contains the same result as **r**D[32:63].
- **Divide Instructions (divide by zero)**
  If the divisor is '0', **r**D is set to zero. If, in addition, the record bit (Rc) in the vector instruction field is '1', then CR0 is set to '0010'.
- **Move to and Move from Special Purpose Register Instructions**
  Table 2-6 on page 45 describes the results of specifying a SPR value that is not defined for the implementation.
- **Move From Time Base Instruction**
  The **mftb** instruction is treated as an alias for the **mfspr** instruction; the results are the same as for executing an **mfspr** instruction.
- **Move From Condition Register Instruction** (bit [11] = '1')
  One CR Field is copied into the associated bits of **r**D and the remaining bits of **r**D are set to zeros.
- **Move From Condition Register Instruction** (bit [11] = '1' and multiple bits of CRM set to '1')
  Source is CR(**n**), where **n** is the CR field specified by the bit in CRM that is set and has the smallest index value. If no bit in CRM is set to '1', the results will be the same as if CRM was set to '00000001'.
- **Move To Condition Register Fields Instruction** (bit [11] = '1' and multiple bits of CRM set to '1')
  CR(**n**) is updated where **n** is the CR Field specified by the bit in CRM that is set and has the smallest index value. If no bit in CRM is set to '1', executing the instruction does not modify the CR.

2.2.4 Floating-Point Processor

The 970FX contains two double precision floating-point units. Each of these units is optimized for fully pipelined double precision multiply-add functionality. In addition, each unit is capable of performing the floating-point divide and square root instructions.

The optional floating-point instructions (**fsqrt**, **fsqrts**, **fres**, **frsqrte**, and **fsel)** defined in the *PowerPC Microprocessor Family: The Programming Environments* manual are implemented.

Note: The 970FX does *not* support the *non-IEEE mode* that was defined in earlier versions of the architecture.

2.2.4.1 Floating-Point Arithmetic Instructions

The architecture requires operands to single-precision floating-point arithmetic instructions to be representable in single-precision format, and if they are not, then the results of the single-precision arithmetic instructions are undefined. In the 970FX, for most cases, a double precision operation will be performed but with the result rounded to single-precision. This will update the instruction target register (**fr**D), FPSCR and *cr1* field of the CR accordingly. For the single-precision divide and square-root instructions, **fdivs** and **fsqrts**, single-precision algorithms are executed on the double-precision data with the final results rounded to single-precision.

2.2.4.2 Floating-Point Invalid Forms and Undefined Conditions

The results of executing an invalid form of a floating-point instruction or an instance of a floating-point instruction for which the architecture specifies that some results are undefined are described below for the cases in which executing an instruction does not cause an exception. Only results that differ from those specified by the architecture are described in the following list.

- **Instruction with Reserved Fields**
  Bits in reserved fields are ignored; the results of executing an instruction in which one or more reserved bits are '1' is the same as if the bits were '0'.
- **Floating-Point Convert to Integer Word Instructions** (**fctiw** or **fctiwz**)
  frD(0:31) is set to x'FFF8 0000'.
- **Floating-Point Convert to Fixed-point Instructions** (**fctiw, fctiwz, fctid,** and **fctidz**)
  The contents of **FPSCR**(**FPRF**) are set to '00000'.
- **Move from FPSCR Instruction**
  frD(0:31) is set to x'FFF8 0000'.

2.2.5 Vector Processor

The 970FX contains two vector units, the vector arithmetic logical unit (VALU) and the vector permute unit (VPERM). The vector instructions and their implementation are described in *Chapter 12, Vector Processing Unit*.

2.2.6 Load Store Processor

2.2.6.1 Floating-Point Load and Store Instructions

Most forms of unaligned floating-point storage accesses are executed entirely in hardware (see *Section 3.4.2.1 Storage Access Alignment Support* on page 75).

2.2.6.2 Fixed-Point Load Instructions

Most forms of unaligned load operations are executed entirely in hardware. If a basic load operation crosses a page boundary, and either page translation signals an exception condition, then when the interrupt occurs, it will appear as though none of the load instructions have executed - this is not always the case for load multiple or load string instructions. For more information, see *Section 2.2.6.4. Fixed-Point Load and Store Multiple Instructions* and *Section 2.2.6.5. Fixed-Point Load and Store String Instructions*.

The Load Algebraic, Load with Byte Reversal, and Load with Update instructions may have greater latency than other load instructions. These instructions are implemented as a sequence of internal operations. Due to the dynamic scheduling and out-of-order execution capability of the processor, these effects are somewhat minimized. It should also be noted that although these instructions are broken up in this manner, the effects are never visible from a programming model perspective.

Any load from storage marked cache-inhibited that is not aligned will cause an alignment interrupt.

2.2.6.3 Fixed-Point Store Instructions

Most forms of unaligned store operations are executed entirely in hardware. If a store operation crosses a page boundary, and the second page translation signals an exception condition, then after the interrupt is taken, it will appear as though none of the storage updates have occurred to either page (this is not always the case for store multiple or store string instructions - see *Section 2.2.6.4. Fixed-Point Load and Store Multiple Instructions* and *Section 2.2.6.5. Fixed-Point Load and Store String Instructions* for more information).

Any store to storage marked cache-inhibited that is not aligned will cause an alignment interrupt.

2.2.6.4 Fixed-Point Load and Store Multiple Instructions

The Load Multiple Word (**lmw**) instruction is executed in a manner such that up to two registers are loaded each cycle. Similarly, the Store Multiple Word (**stmw**) instruction is executed in a manner such that up to two registers are stored each cycle. The 32-entry *store queue* can accept up to two 8-byte stores per cycle; the cache can accept one 8-byte store per cycle. Because these instructions are emulated through the use of microcoded templates, after a small start-up penalty, they are processed at a rate of up to two registers per cycle.

Most forms of **lmw** and **stmw** instructions, even those that cross page and segment boundaries, are executed entirely in hardware. These instructions and the individual storage accesses associated with the instructions are not atomic. If a **stmw** crosses a page boundary, and the second page translation signals an exception condition, then after the interrupt is taken, it will appear as though none, some, or all of the accesses to the first page have occurred. It will also appear as though none of the accesses to the second page have occurred. On the other hand, for the **lmw** instruction that cross a page boundary where the second page translation signals an exception condition, all of the target registers will have an undefined value.

An attempt to execute a non-word aligned **lmw** or **stmw** will cause an alignment interrupt. An attempt to execute an **lmw** or **stmw** to storage marked cache-inhibited will cause an alignment interrupt.

The architecture allows these instructions to be interrupted by certain types of asynchronous interrupts (external interrupts, decrementer interrupts, machine check interrupts, and system reset interrupts). In these cases, for the load multiple instructions, all of the registers that were to be updated will have an undefined value, and the instruction must be completely restarted to achieve the full effect (that is, no partial restart capability is supported). For the store multiple instructions, some of the storage locations referenced by the instruction may have been updated. However, to guarantee full completion of the store multiple instruction, they must also be completely restarted.

2.2.6.5 Fixed-Point Load and Store String Instructions

The Load String Word **(lsw**) instruction is executed in a manner such that up to two registers are loaded each cycle. Similarly, the Store String Word (**stsw**) instruction is executed in a manner such that up to two registers are stored each cycle. The 32-entry store queue can accept up to two 8-byte stores per cycle; the cache itself can only accept one 8-byte store per cycle.

Because the immediate forms of these instructions are implemented using microcoded templates they incur a small start-up penalty, The X-form of the instructions contain a dependency on bits in the fixed-point XER register. Therefore, depending on when the last update to these bits has occurred, the instruction may be subject to a more expensive run-time flush and emulate sequence.

Most Load String and Store String instructions that cross page or segment boundaries are executed entirely in hardware. If a "store string" crosses a page boundary, and the second page translation signals an exception condition, then after the interrupt is taken, it will appear as though none, some or all of the accesses to the first page have completed. It will also appear as though none of the accesses to the second page have occurred. On the other hand, for "load string" instructions that cross a page boundary where the second page translation signals an exception condition, all of the target registers will have an undefined value.

If the storage operand of a Load String Word Immediate (**lswi**) is word aligned, then the accesses are performed in an optimal manner. If the operands are so aligned, the accesses are performed in an optimal manner if the operand resides entirely within a 64-byte block that is resident in the L1 D-cache or resides entirely within a 32-byte block. Although other "unaligned" string operations are supported in hardware, they may cause machine flushes and require long sequences of microcode. As a result, these types of "unaligned" string instructions may have significantly longer latencies.

An attempt to execute a **lswi**, **lswx, stswi,** or **stswx** instruction to storage marked cache-inhibited will cause an alignment interrupt.

The architecture allows these instructions to be interrupted by certain types of asynchronous interrupts (external interrupts, decrementer interrupts, machine check interrupts, and system reset interrupts). In these cases, for the load string instructions, all of the registers that were to be updated will have an undefined value, and the instruction must be completely restarted to achieve the full effect (i.e., no partial restart capability is supported). For the store string instructions, some of the storage locations referenced by the instruction may have been updated.

The architecture describes some preferred forms for the use of load and store string instructions. In the 970FX, these preferred forms have no effect on the performance of the instructions.

2.2.6.6 Load Store Invalid Forms and Undefined Conditions

The results of executing an invalid form of a load/store instruction for which the architecture specifies that some results are undefined are described below for the cases in which executing an instruction does not cause an exception. Only results that differ from those specified by the architecture are described in the following list.

- **Load with Update Instructions** (**r**A = '0')
  Effective address is placed into general purpose register 0 (R0).
- **Load with Update Instruction**s (**r**A=**r**D)
  Effective address is placed into **r**D. The storage operand addressed by EA is accessed, but the data returned by the load is discarded.
- **Load Multiple Instruction**s (**r**A in the range of registers to be loaded)
  If an exception (for example, data storage or external exception) causes the execution of the instruction to be interrupted, the instruction is restarted, **r**A has been altered by the previous partial execution of the instruction, and **r**A≠϶0', the new contents of **r**A are used to compute EA.
- **Load Multiple Instructions** (causing a misaligned access)
  For a Load Multiple Word instruction, if the storage operand specified by EA is not a multiple of four, an alignment exception is taken. For a Load Multiple Doubleword instruction, if the storage operand specified by EA is not a multiple of eight, an alignment exception is taken.
- **Load String Instructions** (zero length string)
  rD is not altered.

- **Load String Instructions** (**r**A and/or **r**B in the range of registers to be loaded)
  If **r**A and/or **r**B is in the range of registers to be loaded, the results are as follows:

  Indexed Form: If **r**A = '0' let Rx be **r**B; otherwise let Rx be the register specified by the smaller of the two values in instruction fields **r**A and **r**B. If **r**D=Rx no registers are loaded; otherwise registers **r**D through RX-1 are loaded as specified in the architecture (for example, only part of the storage operand is loaded).

  Immediate Form: If **r**A = '0', the instruction is executed as if it were a valid form. If **r**A=**r**D no registers are loaded; otherwise registers **r**D through **r**A-1 are loaded as if the instruction was a valid form but specifying a shorter operand length**.**
- **Store with Update Instructions** (**r**A = '0')
  Effective address is placed into general purpose register 0 (R0).
- **Load or Store Floating-Point with Update Instructions** (**r**A = '0')
  Effective address is placed into general purpose register 0 (R0).
- **Floating-Point Store Single Instructions** (exponent less than 874 and **fr**S[09:31] not equal to '0')
  The value placed in storage is a '0' with the same sign as the value in the register.

2.2.7 Branch Processor

2.2.7.1 Branch Processor Instructions

Support Processor Attention Instruction

The 970FX supports a special, implementation dependent instruction for signalling an attention to the support processor.

Figure 2-4. Processor Attention Instruction

attn

00	I	256	/
0	6	21	31

The immediate field (**I**) has no effect on the operation of this instruction in the 970FX microprocessor. If the support processor attention enable bit is set (HID0[31] = '1'), this instruction will cause all preceding instructions to run to completion, the machine to quiesce, and the assertion of the support processor attention signal. If the support processor attention enable is not set (HID0[31] = '0'), this instruction will cause an illegal instruction type of program interrupt.

2.2.7.2 Branch Processor Instructions with Undefined Results

The results of executing an invalid form of a branch instruction or an instance of a branch instruction for which the architecture specifies that some results are undefined are described below. Only results that differ from those specified by the architecture are described in the following list:

- **Instructions with Reserved Fields**
  Bits in reserved fields are ignored; the results of executing an instruction in which one or more of these bits are '1' is the same as if the bits were '0'.
- **bcctr and bcctrl Instructions**
  If branch-options (BO[2]) is set to '0', the contents of CTR, before any update, are used as the target address and for the test of the contents of the of CTR to resolve the branch. The contents of the CTR are then decremented and written back to the CTR.
- **System Call Instructions** (opcode 17)

Bits [30:31]	Description
'00'	**sc** instruction
'01'	illegal instruction exception
'10'	**sc** instruction
'11'	**sc** instruction

2.2.7.3 Move to Condition Register Fields Instruction (mtcrf)

The architecture warns that updating a subset of the **CR** fields on a **mtcrf** instruction may have worse performance than updating all of the fields. In the 970FX, both the **mtcrf** instruction and the **mfcr** instruction will be emulated through the use of microcode templates. For best performance, software should use the new single-field variants of these instructions as described in the architecture.

The 970FX supports the optional architecture extension that defines slight variations to the **mtcrf** and **mfcr** instructions to indicate that the movement of a single field of the condition register is desired. Since the performance of these instructions is better than their multiple field counterparts, use of these instructions is encouraged.

2.2.8 Storage Control Instructions

2.2.8.1 Overview of Key Aspects of Storage Control Instructions

In the 970FX, all cache control instructions operate on aligned 128-byte sections of storage. The following table summarizes many of the key aspects of the storage control instructions:

Table 2-18. Storage Control Instructions

Aspect	Cache Instructions					
	icbi	**dcbt**	**dcbtst**	**dcbz**	**dcbst**	**dcbf**
Granularity	128 bytes	128 bytes	128 bytes	32 or 128 bytes	128 bytes	128 bytes
Semantic checking	Load (DSI on storage exception)	Load (no-op on storage exception)	Load (no-op on storage exception)	Store (DSI on storage exception)	Load (DSI on storage exception)	Load (DSI on storage exception)
"r" bit update	Yes	Yes	Yes	Yes	Yes	Yes
"c" bit update	No	No	No	Yes	No	No
L1 I-cache effect	L1 I-cache and prefetch buffer	None	None	None	None	None
L1 D-cache effect	None	See *Section 2.2.8.4* on page 67	See *Section 2.2.8.4* on page 67	Invalidate	No-op	Invalidate
L2 Cache effect	None	See *Section 2.2.8.4* on page 67	See *Section 2.2.8.4* on page 67	See *Section 2.2.8.5* on page 67	See *Section 2.2.8.6* on page 68	See *Section 2.2.8.7* on page 68
TLB effect	Reload as required	Reload as required	Reload as required	Reload as required	Reload as required	Reload as required
SLB effect	Reload as required	None (no-op if miss)	None (no-op is miss)	Reload as required	Reload as required	Reload as required

2.2.8.2 Instruction Cache Block Invalidate (icbi)

The instruction cache block size for **icbi** on the 970FX is 128 bytes.

Execution of this instruction occurs in multiple phases. First, the effective address is computed and translated by the load/store execution pipeline. Next, the resulting real address is passed to the 970FX STS logic which broadcasts it onto the system bus. When the 970FX STS snoops this type of command on the system bus, it presents the command to the upstream instruction caches. As these invalidates are presented to the instruction cache, the associated real addresses are checked against all 16 possible locations in the effective addressed I-cache that could contain the particular real address. Only entries that actually match the real address will be invalidated. In addition, all entries in the instruction prefetch queue will be invalidated (independent of the address). As an aid for quickly flushing the entire contents of the I-cache, a special mode bit is provided (HID1[9]) that forces each of these 16 entries to be invalidated on an **icbi** (even if their address does not match the invalidate address).

The **icbi** instruction has no effect on the L2 cache.

To insure that the storage access caused by an **icbi** instruction has been performed with respect to the processor executing the **icbi** instruction, an **isync** instruction must be executed on that processor.

2.2.8.3 Instruction Cache Synchronize (isync)

As a performance optimization, the 970FX internally tracks and updates a scoreboard bit for instructions that change instruction-cache-oriented context that are required to be synchronized by the **isync** instruction. When the **isync** instruction is executed, this scoreboard bit is checked to see whether or not the machine must *flush and refetch* the instructions following the **isync**. In addition, the **isync** instruction is often used as a *load barrier* to prevent any subsequent load (or store) instructions from executing before previous load instruction have been completed. In these cases, the scoreboard bit will usually not be set, and the **isync** can complete without causing the flush.

2.2.8.4 Data Cache Block Touch (dcbt and dcbtst)

The data cache block size for **dcbt** and **dcbtst** on the 970FX is 128 bytes.

These instructions act as a touch for the D-cache hierarchy and the TLB. If data translation is enabled (MSR[DR] = 1), and an SLB miss results, then the instruction will be treated as a no-op. If a TLB miss results, then the instruction will reload the TLB (and set the reference bit). Once past translation, if the page protection attributes prohibit access, or the page is marked cache-inhibited, or the page is marked guarded, or the processors' D-cache is disabled (using the bits in the HID4 register), then the instruction will be finished as a *no-op* and will not reload the cache. Otherwise, the instruction will check the state of the L1 D-cache and if the block is not present, then it will initiate a reload. Note that this may also reload the L2 cache with the addressed block if it is not already present. If the cache block is already present in the L1 D-cache, the cache content is not altered. Note that if the **dcbt** or **dcbtst** instruction does reload cache blocks, it will affect the state of the cache replacement algorithm bits.

The 970FX does implement the optional extension to the **dcbt** instruction that allows software to directly engage a data stream prefetch from a particular address.

2.2.8.5 Data Cache Block Zero (dcbz)

The data cache block size for **dcbz** on the 970FX is 128 bytes. Support is also provided for a **dcbz** of 32 bytes in order to accommodate coding which assumed a 32-byte block size. The **dcbz** actions are listed in *Table 2-19*.

Note: The entire instruction cache must be flushed whenever HID5[56] or HID5[57] are changed.

*Table 2-19. **dcbz** Actions*

HID5[57]	HID5[56]	**dcbz** Instruction Bit [10]	Action
1	X	0	Illegal instruction
X	X	1	Cache block (128 bytes) zeroed
0	0	0	Cache block (128 bytes) zeroed
0	1	0	32 byte block zeroed

The function of **dcbz** is performed in the L2 cache. As a result, if the block addressed by the **dcbz** is present in the L1 D-cache, then the block will be invalidated before the operation is sent to the L2 cache logic for execution. The L2 cache will gain exclusive access to the block (without actually reading the old data), and will perform the zeroing function. For the 32-byte **dcbz**, the L2 cache may be required to read the line and then zero the 32 bytes.

If the cache block specified by the **dcbz** instruction contains an error, even one that is not correctable with error correction code (ECC), the contents of all locations within the block are set to zeros in the L2 Cache. If the specified block in the L2 Cache does not contain a hard fault, a subsequent load from any location within the cache block will return zeros and not cause a Machine Check interrupt.

If the block addressed by the **dcbz** instruction is in a memory region marked cache-inhibited, or if the L1 D-cache or L2 cache is disabled (using the bits in the HID registers), then the instruction will cause an alignment interrupt to occur.

Implementation Note–In order to emulate the behavior of the obsolete **dcba** instruction, a mode bit is provided that changes the behavior of **dcbz** as follows. When the mode bit is set to ‘1’, if the block addressed by the **dcbz** instruction is in a memory region marked cache-inhibited, the instruction is treated as a no-op. The **dcba** instruction was defined such that the referenced and changed bits need not be updated in this case. However, the 970FX will update these bits.

2.2.8.6 Data Cache Block Store (dcbst)

The data cache block size for **dcbst** on the 970FX is 128 bytes.

The **dcbst** instruction forces all preceding stores to the referenced block to become committed to the cache hierarchy, and then forces a clean operation in the L2 cache.

The **dcbst** instruction has no direct effect on the L1 D-cache (since it is store-through, it never contains modified data). The L2 cache updates and processor interconnect bus operations are performed as shown in *Table 3-5. 970FX L2 Cache Transitions Due to Processor Instructions* and *Table 3-6. 970FX L2 Cache State Transitions Due to Bus Operations* on page 82.

2.2.8.7 Data Cache Block Flush (dcbf)

The data cache block size for **dcbf** on the 970FX is 128 bytes.

The **dcbf** instruction forces all preceding stores to the referenced block to become committed to the cache hierarchy, and then acts like an invalidate to the L1 D-cache (since it is store-through, it never contains modified data). The L2 cache updates and processor interconnect bus operations are performed as shown in *Table 3-5. 970FX L2 Cache Transitions Due to Processor Instructions* and *Table 3-6. 970FX L2 Cache State Transitions Due to Bus Operations*.

Implementation Note–The **dcbf** will flush the 970FX L2 cache and/or any other CPU’s L2 cache in the system.

2.2.8.8 Load and Reserve and Store Conditional Instructions (lwarx/ldarx, stwcx/stdcx)

The reservation granularity for the 970FX is 128 bytes. The **lwarx** and **ldarx** instructions are sometimes executed speculatively. An attempt to execute a non-word aligned **lwarx** or **stwcx**, or a non-doubleword aligned **ldarx** or **stdcx** will cause an alignment interrupt. An attempt to execute a **lwarx, ldarx, stwcx,** or a **stdcx** instruction to storage marked cache-inhibited will cause a data storage interrupt.

2.2.9 Memory Synchronization Instructions

The 970FX design achieves high performance by exploiting speculative out-of-order instruction execution. The **sync** instruction, as defined in the architecture, acts as a serious barrier to this type of aggressive execution and therefore can have a dramatic effect on performance. Although the 970FX has optimized the performance of **sync** to some degree, care should be exercised in the use of this instruction. As a performance consideration, software should attempt to use the lightweight version of **sync** (**lwsync**) whenever possible.

The 970FX also supports the architected **ptesync** instruction for use in synchronizing page table updates. The 970FX implements **eieio** as described in the *PowerPC Virtual Environment Architecture (Book II)*.

In the 970FX storage subsystem logic, the store queues above the L2 cache attempt to gather both cacheable and cache inhibited store operations sequentially to improve bandwidth. A mode bit exists in the STS mode register (SCOM address x'043000') to disable store gathering of cache inhibited stores. Alternatively, if store gathering is not desired, software must insert between successive stores, either an **eieio** (preferable for performance) or a **sync** to prevent it. The **eieio** instruction is broadcast onto the system bus to allow ordering to be properly enforced throughout cache hierarchy and memory system (when detected on the system bus, these transactions have no direct effect on the processor).

2.2.10 Recommended Simplified Mnemonics

To simplify assembly language coding, a set of alternative mnemonics is provided for some frequently used operations (such as no-op, load immediate, load address, move register, and complement register). Programs written to be portable across the various assemblers for the PowerPC Architecture should not assume the existence of mnemonics not described in this manual.

For a complete list of simplified mnemonics, see the *PowerPC Architecture Books*.

3. Storage Subsystem

The 970FX storage subsystem (STS) encompasses the core interface unit (CIU), non-cacheable unit (NCU), the L2 cache control and L2 cache, and the bus interface unit (BIU).

This section provides an overview and a high-level block diagram of the storage subsystem. It summarizes key design fundamentals and the storage hierarchy. The functional units are described in detail.

The following key features are fundamental to the design:

- Store-through L1 D-cache.
- No castouts or snoop pushes by the core.
- Non-blocking snoop invalidates to the core (both instruction and data invalidates).
- Integrated L2 Controller.
- L2 controller handling of cacheable instruction fetches, loads and stores, and **dcbz** instructions.
- Non-cacheable unit handling of other storage type instructions.

3.1 Storage Hierarchy

Table 3-1. Storage Hierarchy Characteristics

Characteristic	L1 Instruction Cache	L1 Data Cache	L2 Cache
Data type	Instructions only	Data only	Instructions and data
Size	64 KB	32 KB	512 KB
Associativity (replacement policy)	Direct map	2-way set associative (LRU)	8-way set associative (LRU)
Line size (sector)	128 bytes (4 * 32-bytes)	128 bytes	128 bytes
Operation granularity	128 bytes	128 bytes	128 bytes
Index	Effective address	Effective address	Physical address
Tags	Physical address	Physical address	Physical address
Number of ports	1Read or 1Write (directory has 2Read or 1Write)	2Read and 1Write	1Read or 1Write
Inclusiveness	N/A	N/A	Inclusive of L1 D-cache not inclusive of L1 I-cache
Hardware coherency	No	Yes	Yes separate snoop ports
Store policy	N/A	Store-through no allocate on store miss	Store back allocate on store miss
Enable bit	Yes	Yes	No
Reliability, availability, serviceability (RAS)	Parity with invalidate on error for data and tags	Parity with invalidate on error for data and tags	ECC on data; parity on tags (recoverable with redundant tags)

3.2 Caches

The 970FX contains two levels of cache hierarchy: L1 and L2. The coherence block size for the 970FX is 128 bytes. For more information about 970FX cache characteristics, see *Table 3-1 Storage Hierarchy Characteristics* on page 71. For more information about the relative performance of these caches.

The 970FX automatically maintains the coherency of all data cached in these caches. Since some levels of the cache hierarchy contain both instructions and data, when the L2 cache services an instruction cache reload request, it does so in a coherent manner. This avoids the scenario where a line is reloaded into the L2 cache on behalf of a non-coherent instruction fetch, but then accessed by a load or store instruction with an aliased address that calls for correct coherency. However, the processor does not maintain instruction storage consistent with data storage and, as described in PowerPC Architecture, synchronization code is required to make the two consistent.

The L1 I-cache is indexed with an effective address. As a result, multiple copies of a particular physical block of memory can reside in multiple positions in the L1 I-cache (up to sixteen since four bits of the effective address are used in indexing the cache). The tag comparison associated with lookups in this cache is done using physical addresses, so there are no 'synonym' or 'alias' hazards that must be explicitly handled by the system software.

The L1 D-cache is indexed with an effective address. Only one copy of a particular real address block is kept in the cache at a time. On each access, a tag comparison is done with the physical address. On a cache miss, the cache reload mechanism searches the other three related sets to determine if the required real address block is located elsewhere in the cache. If so, it will appropriately eliminate these copies.

In addition to maintaining caches, the 970FX also includes several types of queues that act as logical predecessors and extensions to the caches. In particular, the machine contains store queues for holding store data "above the caches," cast-out queues for holding modified data that has been pushed out of the caches (by the replacement algorithm, cache control instructions, and/or snoop requests), and others. Hardware keeps all of these queues coherent, and in general neither software nor system hardware should be able to observe their presence.

3.2.1 Store Gathering

The 970FX performs gathering of cacheable stores in order to reduce the store traffic into the L2 cache. The gathering occurs in L2 store queues that sit above the L2 cache. The store queue is comprised of eight, 64-byte wide, fully-associative entries. Stores can be gathered while architecturally permitted (that is, there is no intervening barrier operation) and the matching address is valid in the store queue. The conditions for pushing the store queue data into the L2 cache are not visible to the programmer.

Gathering of cache-inhibited stores is also supported and can be disabled with a mode bit in the Non-Cacheable Unit (NCU) Configuration Register.

3.3 Cache Management

3.3.1 Flushing the L1 I-Cache

To help flush the entire contents of the I-cache efficiently, the 970FX implements a special mode of operation for the **icbi** instruction. This mode can be selected using a bit in the HID1 register. In this mode, all directory lookups on behalf of an **icbi** act as though there was a real address match. Therefore all lines looked at by the **icbi** will in fact be invalidated. As a result, the entire L1 I-cache can be invalidated by issuing a series of **icbi** instruction that step through each congruence class of the I-cache.

Note: Another way to clear the I-cache is to actually fill it with a set of known values by executing a piece of code that effectively touches each line of the cache. One way to write this code is to have a series of 512 branches to branches whose effective addresses are sequentially separated by 128 bytes (the line size of the I-cache). Many other possible code sequences can achieve the same effect.

3.3.2 Flushing the L1 D-Cache

The L1 D-cache is a store-through design, so it never holds modified data. As a result, to perform a flush of the L1 D-cache, the only instruction required is a **sync**. The **sync** instruction forces any pending stores in the store queue above the L1 cache to become globally coherent before the **sync** is allowed to complete.

To completely invalidate the L1 D-cache, use the l1dc_flsh mode bit located in the HID4 register to cause a flash invalidate of the D-cache. Software simply needs to set this bit and follow it with a **sync** instruction.

3.3.3 L2 Cache Disabling and Enabling

The L2 cache cannot be disabled.

3.3.4 L2 Cache Flush in Direct-Mapped Mode

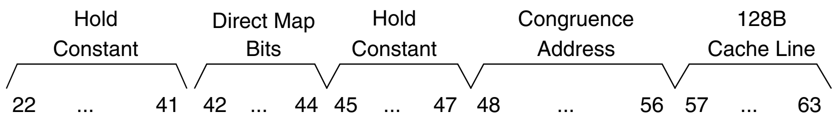
The STS Mode Register (at SCOM address x‘043000’) is set to x‘0000 0000 0000 8000’ to enter direct-mapped mode. In direct-mapped mode, victims are selected based on a simple address decode. *Table 3-2* shows the decode. The three tag address bits used for the mapping are real address bits [42:44].

Table 3-2. Simple Address Decode

Real Address (Bits [42:44])	Selected Victim
000	A
001	B
010	C
011	D
100	E
101	F
110	G
111	H

3.3.4.1 L2 Cache Flush Algorithm

Figure 3-1. L2 Address Map



The following sequence will flush the entire L2 cache to memory via software:

1. Disable interrupts.
2. Disable data address translation by setting MSR[DR] to '0'.
3. Disable instruction cache (I-cache) prefetch by setting HID1[7:8] to '00'.
4. Disable data cache (D-cache) prefetch by setting HID4[25] to '1'.
5. Flash invalidate the D-cache by setting HID4[28] to '1'.
6. Execute a **sync** instruction.
7. Disable the D-cache (set HID4[37:38] to '11'). This will guarantee that all loads are visible to the L2.
8. Set the L2 to direct mapped mode. This may be done by the service processor or through the SCOM control (SCOMC) and SCOM data (SCOMD) SPR interface.
9. Execute a **sync** instruction.
10. Initialize a register with the starting address of a 4 MB cacheable region of memory that is aligned on a 4 MB boundary (that is, bits [42:63] are all zeros).
11. Execute eight load instructions, incrementing the direct map field (bits [42:44]) of the load address between each load.
12. Increment the congruence address field (bits [48:56]) of the load address, and repeat step 9.
13. Repeat step 10 for all 1024 congruence address values.

To power down after performing this sequence, the processor executes an **attn** instruction to enter the quiescent state. To return to normal processing, set the L2 cache to set-associative mode. Enable the D-cache, prefetching, data address translation and interrupts, as desired.

3.4 Storage Model

3.4.1 Atomicity

The 970FX is fully compliant with the architectural requirement for single-copy atomicity on naturally aligned storage accesses.

3.4.2 Storage Access Ordering

The architecture defines a weakly ordered storage model for most types of storage access scenarios. For these cases, the 970FX takes advantage of this relaxed requirement to achieve better performance through out-of-order instruction execution and out-of-order bus transactions. As a result, if strongly ordered storage accesses are required, software must use the appropriate synchronizing instruction (**sync, ptesync, eieio** or **lwsync**) to enforce order explicitly, or perform these accesses to regions marked with attributes that require the hardware to enforce strong ordering (for example, stores to storage marked cache-inhibited and guarded, are required to occur in-order).

The 970FX performs load operations out-of-order internally to the processor; however, it also keeps track of these loads in a way that lets it know when an external request for exclusivity may lead to the appearance of non-sequential execution. For these cases, the 970FX has the capability to flush potentially bad results, and re-execute the code starting from the suspect load instruction.

3.4.2.1 Storage Access Alignment Support

Most storage accesses are performed without software intervention (that is, without an alignment interrupt). The relative performance of these accesses depends to some degree on their alignment. In many cases, unaligned storage accesses are handled with a performance equivalent to aligned accesses; however, in some cases the 970FX is forced to break unaligned accesses into multiple internal operations. Further, since effective address alignment for storage references cannot be determined until execution time, and 970FX's dataflow-oriented execution pipelines do not support iteration, some unaligned storage accesses actually cause a pipeline flush to allow a microcoded emulation of the instruction. For more information on the performance aspects of unaligned storage accesses.

The following list summarizes the cases in which the 970FX will engage a microcoded emulation of unaligned storage references:

- Any fixed-point load operation that crosses a 64-byte boundary (note 1)
- Any fixed-point load operation that misses in the L1 D-cache and crosses a 32-byte boundary (note 1)
- Any fixed-point store operation that crosses a 4 KB boundary (note 2)
- Any floating-point load double operation that is word aligned and crosses a 64-byte boundary (note 1)
- Any floating-point load double operation that is *word aligned*, misses in the L1 D-cache, and crosses a 32-byte boundary (note 1)
- Any floating-point store operation that is *word aligned* and crosses a 4 KB boundary (see note 1)

Notes:

1. If the instruction is not a multiple or string instruction, the access crosses a page boundary, and the access to either page causes an exception, appearing as though the load instruction has not been executed (i.e., **fr**D or **r**D is not modified).

2. If the access to the first page causes an exception, storage is not modified. Otherwise storage in the first page is updated even if the access to the second page causes an exception.

As an aid for software identification of these cases, the 970FX supports a debug-only mode, controlled by HID4[24], that will force an *alignment interrupt* in these scenarios. See *Section 4.5.8 Alignment Exception* on page 101 for a summary of cases in which the 970FX will take an *alignment interrupt.*

3.4.3 Atomic Updates and Reservations

The coherency granule size in the 970FX is 128 bytes.

The following events will affect the state of the reservation register:

- Execution of a **lwarx** or **ldarx** instruction (sets new reservation)
- Execution of a **stwcx** or **stdcx** instruction (successful or not, address match or not, the reservation is cleared)
- Snooped-RWITM bus operation that matches reservation address (clears the reservation)
- Snooped-DCLAIM bus operation that matches reservation address (clears the reservation)
- Snooped-Write w/ FLUSH bus operation that matches reservation address (clears the reservation)
- Snooped-Write w/ KILL bus operation that matches reservation address (clears the reservation)

When performing bus snooping, the 970FX will check the state of the internal caches *and* the state of the reservation register to formulate a snoop response. If a particular coherency block is not in any of the caches but the address is valid in the reservation register, then the processor/STS will act as though the coherency block is in the *shared* state for the snoop response (this will prevent another processor from taking the block as exclusive on a simple READ bus transaction).

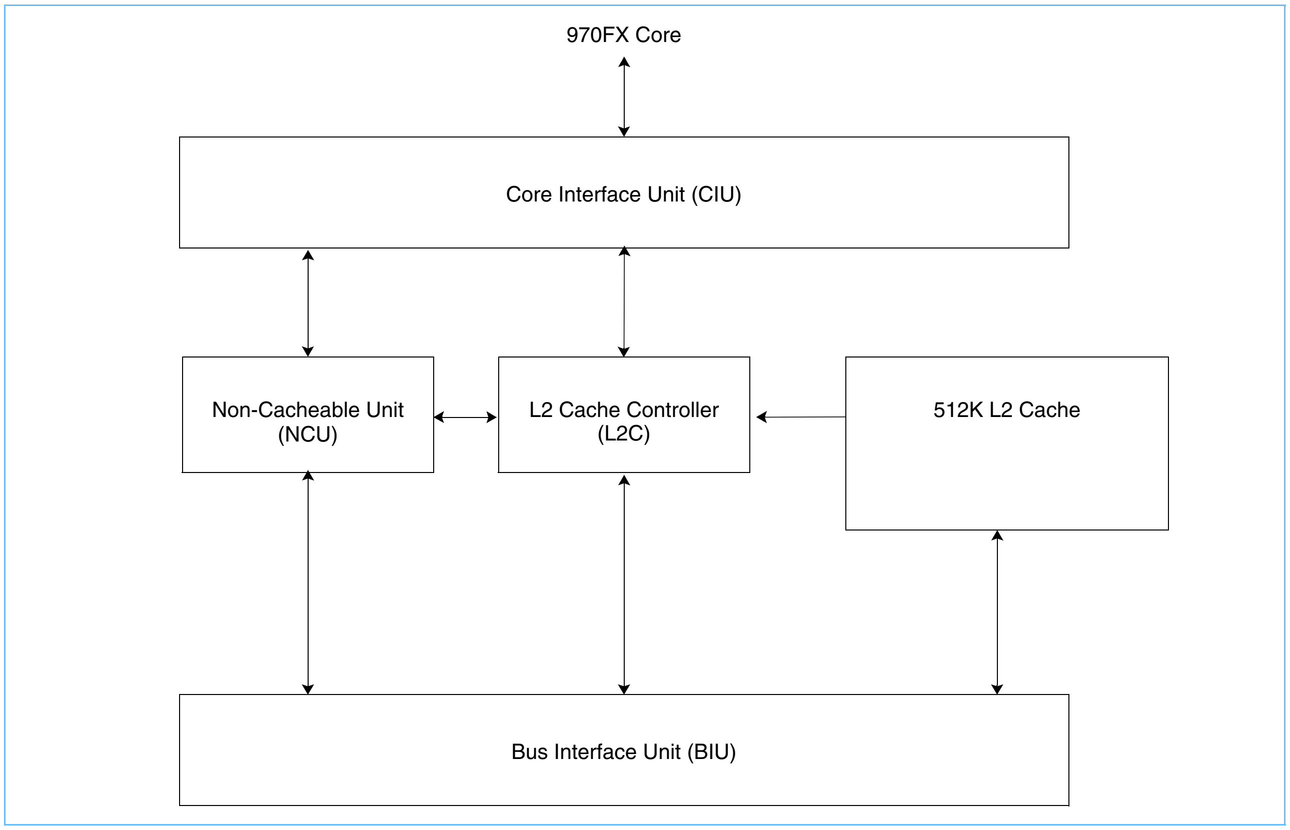
3.5 Functional Units

Table 3-3 lists the functional units within the storage subsystem and *Figure 3-2* shows how they interact. The noncacheable unit (NCU) handles all communications to and from the core that are not handled by the L2 cache. The core interface unit (CIU) and L2 cache controller are described in detail in the following sections.

Table 3-3. Storage Subsystem Functional Units

Unit	Mnemonic
Core Interface Unit	CIU
L2 Cache Controller	L2C
Non-Cacheable Unit	NCU
Bus Interface Unit	BIU

Figure 3-2. 970FX Storage Subsystem High-Level Diagram



3.5.1 Core Interface Unit

The core interface unit (CIU) is the interface block between the 970FX core and the rest of the storage subsystem. It contains the necessary pipeline buffers and queues to maintain the required transfer rates to and from the 970FX core.

The interfaces to the 970FX core include one instruction fetch unit (IFU) port, two load/store unit (LSU) ports, and one data prefetch and translate port. The CIU performs request arbitration, queueing, and flow control. It also maintains load/store ordering and provides prefetch support. In addition, the 970FX core interfaces include one store interface with the LSU. The CIU performs request queueing and flow control for this interface. It maintains store ordering and supports a 16-byte data path.

The CIU provides request flow control for the L2 cache interface. It dispatches operations to the L2 cache interface based on storage mode and operation type. The CIU also provides request flow control for the NCU interface. It dispatches operations to the NCU based on storage mode and operation type. It maintains cache-inhibited store ordering.

The reload/invalidate address interfaces include one IFU port, one LSU port, and one translate port. The CIU provides support for L1 cache invalidates. It also requests arbitration and flow control. The reload data bus is a 32-byte data path running at the CPU speed (1:1).

3.5.2 L2 Cache Controller

As shown in *Figure 3-2. 970FX Storage Subsystem High-Level Diagram*, the L2 cache controller (L2C) resides between the CIU and the BIU and also interfaces with the NCU. See *Table 3-1* for additional details of the L2 cache features.

L2 Cache Features
- 512 KB size, 8-way set associative
- Fully inclusive of the L1 data cache
- Unified L2 cache controller (combines entities such as instructions, data, and PTEs)
- Store-in L2 cache (store-through L1 cache)
- Fully integrated cache, tags, and controller
- Five-state modified/exclusive/recent/shared/invalid (MERSI) coherency protocol

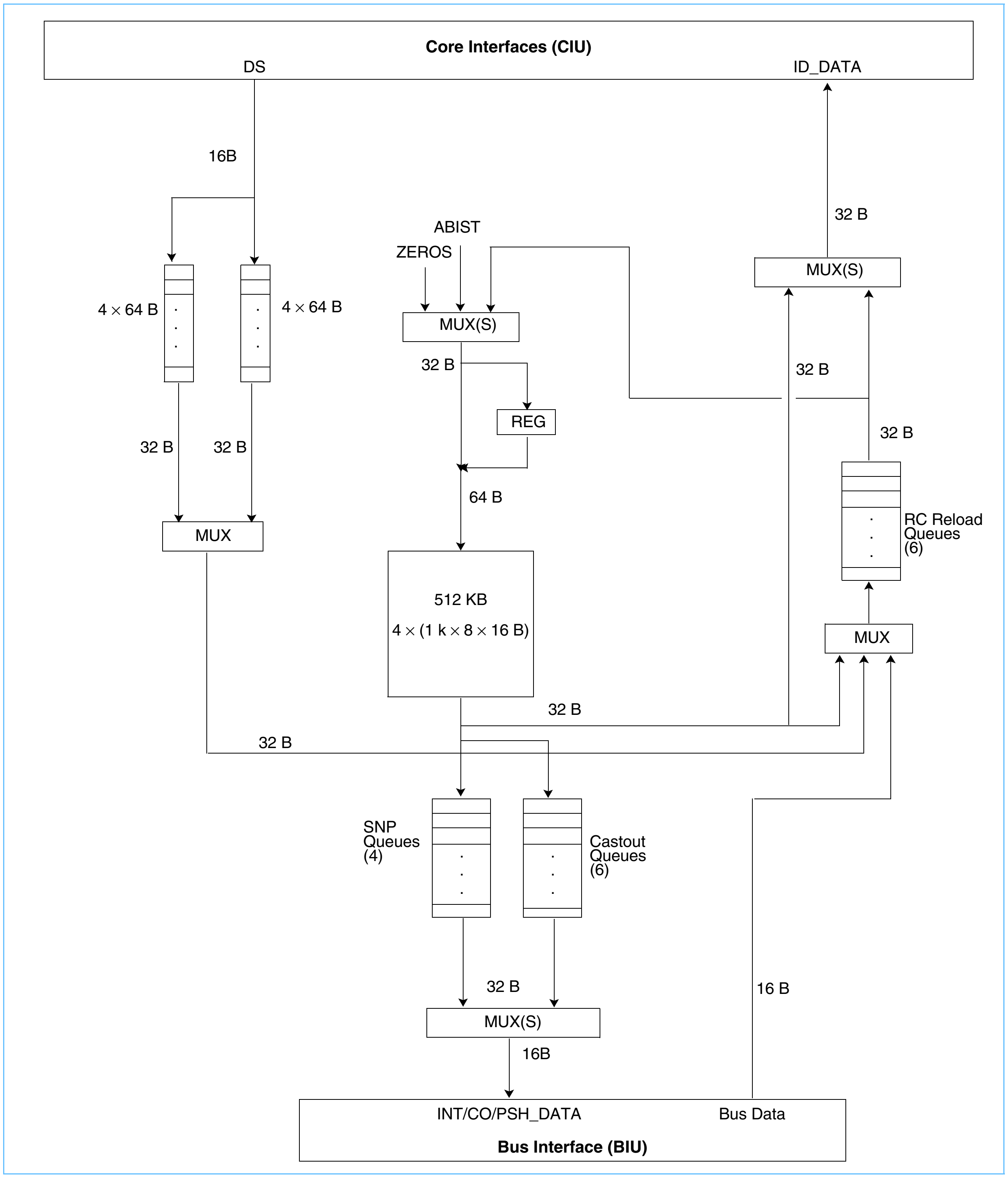
L2 Cache Controller Features
- Runs at core frequency (1:1)
- Handles all cacheable loads/stores (including **lwarx**/**stcwx**)
- Critical 32-byte forwarding on data loads
- Critical 32-byte forwarding in instruction fetches
- Six read/claim queues (RCQs)
- Eight (64-byte wide) store queues
- Store gathering supported
- Non-blocking L1 D-cache invalidates
- Recoverable single-bit directory errors (through redundant directory)

L2 Cache Snooper Features
- Separate directory for all system bus snoops
- Four snoop/intervention/push queues

L2 Controller Block Diagram

Figure 3-3 shows the dataflow of the L2 Cache Controller including the data queues.

Figure 3-3. 512 KB L2 Data Flow



3.5.2.1 Cache-Coherency

The cache-coherency protocol used in the L2 cache is standard MERSI as defined in *Table 3-4*.

Table 3-4. Cache-Coherency Protocol

Status Bit	Name	Meaning
M	Modified	The cache block is modified with respect to the rest of the memory subsystem.
E	Exclusive	The cache block is not cached in any other cache.
R[1]	Recent	The cache block is shared and this processor is the most recent reader of the cache block.
S	Shared	The cache block was (and still may be) cached by multiple processors.
I	Invalid	The cache block is invalid.

1. **Implementation Note** – The 970FX supports a cache-coherency mode in which the R state is not used. R is replaced with shared-last (SL).

3.5.2.2 Cache-Coherency Paradoxes

In the 970FX, some parts of cache-inhibited operations are handled by a special section of logic that does not access the caches as part of its normal operation. As a result, if data associated with cache-inhibited operations are present in the caches (causing a cache-coherency paradox), the 970FX will bypass some of the caches. This introduces the possibility of observing stale data is opened (more specifically, the 970FX will read from and write to the L1 D-cache if it hits, but it will bypass the L2 cache completely).

3.5.2.3 Cache State Transition Tables

Table 3-5 and *Table 3-6* show the cache state transitions that occur as a result of processor operations and snooped bus operations.

Table 3-5. 970FX L2 Cache Transitions Due to Processor Instructions

#	Instruction	Storage Mode	Coherency State	Bus Operation	A Resp In	Comment
1	**ld, dcb**	Ca	M, E, S, R			
2	**larx**	Ca	M, E, S, R			
3	**ld, dcbt, larx**	Ca	I → S*, E	Read Cache Line	S, Null	Atomic if LARX
4	**ld, larx,**	NonCa		Read Noncache Line		Atomic if LARX
5	**dcbt**	NonCa				No-op
6	**st, dcbtst, stcx**	Ca	M → M			
7	**st, stcx**	Ca	E → M			
8	**dcbtst**	Ca	E → E			
9	**st, dcbtst, stcx**	Ca	S, R → M	DClaim		Atomic if STCX
10	**st, dcbtst, stcx**	Ca	I → M	RWITM	RTY	Atomic if STCX
11	**st, stcx**	NonCa		Write with Flush		Atomic if STCX
12	Deallocate	Ca	M → I	Write with Kill		Copyback, W = ‘0’, M = ‘0’
13	Deallocate	Ca	E, S, I → I			
14	**dcbf**	Ca	M → I	Write with Kill		W = ‘1’, M = ‘0’
15	**dcbf**	Ca	E → I			
16	**dcbf**	Ca	I, S, R → I	Flush Block		
17	**dcbst**	Ca	M→S, E	Write with Clean		Cache > E
18	**dcbst**	Ca	E, R, S→ E, R, S			→ E, R
19	**dcbst**	Ca	I	Clean		
20	**dcbz**	Ca	E, M → M			
21	**dcbz**	Ca	I, S → M	DClaim		
22	**dcbz-32byte**	Ca				32 bytes, treated as store
23	**icbi**			IKill		

Note: Ca = cacheable, I = ‘0’; NonCa is I = ‘1’, S* means R if enabled.

Table 3-6. 970FX L2 Cache State Transitions Due to Bus Operations

Number	Bus Operation		Snooper State	Rsrv State	AResp Out	AResp In	Comments
1	Read Burst	N = '1', S = '0'	M →S		M	M	Causes C →M → C data-only operation. (Intervention)
2		N = '1', S = '1'	M → E		M	M	
3		N = '1', S = '0'	E,R → S		Shrl	Shrl	Causes C → C intervention
4		N = '1', S = '1'	E,R → E,R		Shrl	Shrl	Causes C → C intervention
5	Read Non Burst	N = '0', S = '0'	M → S		Retry	Retry	Causes Write with Clean (Push)
6		N = '0, S = '1'	M → E		Retry	Retry	Causes Write with Clean (Push)
7		N = '0', S = '0'	E,R → S		S	S	Reader may go R
8		N = '0', S = '1'	E,R → E,R		S	S	Reader will go S
9	Any Read		S		S	S	
10			I	R = '0'			
				R = '1'	S		
11	RWITM	N = '1'	M → I		M	M	Causes C -> C intervention
		N = '1'	E,R → I		Shdl	Shdl	Causes C -> C intervention
12		N = '0'	M → I		Retry	Retry	Causes Write w/Kill (Push)
13		N = '0'	E,R → I		Null		N says don't intervene
14			IS → I				
15	Write-With-Kill, DKill, DClaim		IESM → I				
16	Write-With-Flush		M →I		Retry	Retry	Causes Write w/Kill (Push)
17			ISE → I				
18	Flush		M → I		M		Causes Write w/Kill (Push)
19			ISER → I				
20	Clean		M →S,E		M		Causes Write with Clean M = '0' Cache → E
21	Clean		E,R,S → E,R,S		S		→ E,R
22	Clean		I → I		Null		
23	**sync**, **tlbsync**				Retry Null		Retry until done. Null when done.

3.5.3 Managing the Data Prefetch Hardware

Software can manage the data prefetch hardware by using special forms of the data cache block touch (**dcbt**) instruction. Two forms of **dcbt** variants are implemented in the 970FX, as described below.

3.5.3.1 Optional dcbt Variant

The architecture describes the first **dcbt** variant as optional. This version of the instruction includes a 2-bit Touch Hint (TH) field (instruction bits [9:10]), which permits a program to provide a hint regarding a sequence of data cache blocks. Such a sequence is called a "data stream". A **dcbt** instruction in which TH[0:1] does not equal '00' is called a "data stream variant" of **dcbt**.

Figure 3-4 shows the instruction format and interpretation of the TH field for this **dcbt** variant.

Figure 3-4. Data Cache Block Touch X-Form (Optional Variant)

dcbt **r**A,**r**B,TH

31	////	TH	RA	RB	278	/
0	6	9	11	16	21	31

Let the effective address (EA) be the sum (**r**A|0) + (**r**B).

TH	Description
00	The program will probably soon load from the block containing the byte addressed by EA.
01	The program will probably soon load from the data stream consisting of the block containing the byte addressed by EA and an unlimited number of sequentially following blocks (i.e., consisting of the blocks containing the bytes addressed by EA + n x block_size; where n = 0, 1, 2,...).
10	Reserved
11	The program will probably soon load from the data stream consisting of the block containing the byte addressed by EA and an unlimited number of sequentially preceding blocks (i.e., consisting of the blocks containing the bytes addressed by EA - n x block_size; where n = 0, 1, 2,...).

Restrictions

For the data stream variant cases (TH equals '01' or TH equals '11'), prefetching of the stream will start even if the first block of the stream is already in the L1 data cache.

3.5.4 Enhanced DCBT Variant

An additional variant of the **dcbt** instruction is implemented in the 970FX. In this version, the TH field is extended to four bits (instruction bits [7:10]) to provide the additional variant of **dcbt**. Note that the 2-bit optional variant of the software touch is a subset of the 4-bit extended version. A brief description of this variant is shown in *Figure 3-5*.

Figure 3-5. Data Cache Block Touch X-form (Enhanced Variant)

dcbt **r**A,**r**B,TH

31	/	TH	RA	RB	278	/
0	6	7	11	16	21	31

Let the effective address (EA) be the sum (**r**A|0) + (**r**B).

TH Description

0000 The program will probably soon load from the block containing the byte addressed by EA.

0001 The program will probably soon load from the data stream consisting of the block containing the byte addressed by EA and an unlimited number of sequentially following blocks (that is, consisting of the blocks containing the bytes addressed by *EA + n x block_size*, where n = 0, 1, 2,...).

0011 The program will probably soon load from the data stream consisting of the block containing the byte addressed by EA and an unlimited number of sequentially preceding blocks (that is, consisting of the blocks containing the bytes addressed by *EA - n x block_size*, where n = 0, 1, 2,...).

1000 The **dcbt** instruction provides a hint that describes certain attributes of a data stream, and optionally indicates that the program will probably soon load from the stream. The EA in this case, is interpreted as follows:

EA_TRUNC	D	UG	▨	ID
0 … 56	57	58	59	60 … 63

Bits	Name	Description
0:56	EA_TRUNC	High-order 57 bits of the effective address of the first unit of the data stream. The low order seven bits of that effective address are zero.
57	D	Direction 0 Subsequent units are the sequentially following units. 1 Subsequent units are the sequentially preceeding units.
58	UG	0 No information is provided by the UG field. 1 The number of units in the data stream is unlimited, the program's need for each block of the stream is not likely to be transient, and the program will probably soon load from the stream.
59	–	Reserved
60:63	ID	Stream ID to use for tihs data stream.

Note: All other TH decodes are reserved.
Restrictions – The TH = '1000' version of the **dcbt** instruction is not recognized when MSR[DR] = '0'.

Implementation Note— The extended **dcbt** variant implemented in the 970FX is a partial implementation of a more complete extension to be implemented on future PowerPC processors. In this partial implementation, the stream ID field selects which hardware prefetcher the stream is assigned to, but is otherwise unused. Unless the UG field is set to 1 by software – which initiates prefetch of an unlimited stream – the TH = 1000 form of the **dcbt** does nothing.

4. Exceptions

The operating environment architecture (OEA) portion of the PowerPC Architecture defines the mechanism by which PowerPC processors implement exceptions (referred to as interrupts in the architecture specification). Exception conditions may be defined at other levels of the architecture. For example, the user instruction set architecture (UISA) defines conditions that may cause floating-point exceptions; the OEA defines the mechanism by which the exception is taken.

The PowerPC exception mechanism allows the processor to change to supervisor state as a result of unusual conditions arising in the execution of instructions and from external signals, bus errors, or various internal conditions. When exceptions occur, information about the state of the processor is saved to certain registers and the processor begins execution at an address (exception vector) predetermined for each exception. Processing of exceptions begins in supervisor mode.

Although multiple exception conditions can map to a single exception vector, often a more specific condition can be determined by examining a register associated with the exception—for example, the DSISR and the Floating-Point Status and Control Register (FPSCR). The high-order bits of the Machine State Register (MSR) are also set for some exceptions. Software can explicitly enable or disable some exception conditions.

The PowerPC Architecture requires that exceptions be taken in program order. Therefore, although a particular implementation may recognize exception conditions out-of-order, they are handled strictly in-order with respect to the instruction stream. When an instruction-caused exception is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that have not yet entered the execute state, are required to complete before the exception is taken. For example, if a single instruction encounters multiple exception conditions, those exceptions are taken and handled based on the priority of the exception. Likewise, exceptions that are asynchronous and precise are recognized when they occur, but are not handled until all instructions currently in the execute stage successfully complete execution and report their results.

To prevent loss of state information, exception handlers must save the information stored in the Machine Status Save/Restore Registers, SRR0 and SRR1, soon after the exception is taken to prevent this information from being lost due to another exception being taken. Because exceptions can occur while an exception handler routine is executing, multiple exceptions can become nested. It is up to the exception handler to save the necessary state information if control is to return to the excepting program.

In many cases, after the exception handler returns, there is an attempt to execute the instruction that caused the exception (such as a page fault). Instruction execution continues until the next exception condition is encountered. Recognizing and handling exception conditions sequentially guarantees that the machine state is recoverable and processing can resume without losing instruction results.

In this book, the following terms are used to describe the stages of exception processing.

Recognition	Exception recognition occurs when the condition that can cause an exception is identified by the processor.
Taken	An exception is said to be taken when control of instruction execution is passed to the exception handler. That is, the context is saved and the instruction at the appropriate vector offset is fetched and the exception handler routine is begun in supervisor mode.
Handling	Exception handling is performed by the software linked to the appropriate vector offset. Exception handling is begun in supervisor mode (referred to as privileged state in the architecture specification).

Note: The PowerPC Architecture documentation refers to exceptions as interrupts. In this book, the term ‘interrupt’ is reserved to refer to asynchronous exceptions and sometimes to the event that causes the exception. The PowerPC Architecture also uses the word ‘exception’ to refer to IEEE-defined floating-point exception conditions that may cause a program exception to be taken, (see the *PowerPC Microprocessor Family: The Programming Environments* manual for more information). The occurrence of these IEEE exceptions may not cause an exception to be taken. IEEE-defined exceptions are referred to as IEEE floating-point exceptions or floating-point exceptions.

Note: Previous PowerPC microprocessors supported specifying the base real address by using the exception prefix field, MSR[IP]. The 970FX does not support this.

4.1 970FX Microprocessor Exceptions

As specified by the PowerPC Architecture, exceptions can be either precise or imprecise and either synchronous or asynchronous. Asynchronous exceptions are caused by events external to the processor’s execution; synchronous exceptions are caused by instructions. The types of exceptions are shown in *Table 4-1*.

Note: All exceptions except for the maintenance exception, thermal exception, and performance monitor exception are defined, at least to some extent, by the PowerPC Architecture.

Table 4-1. PowerPC 970FX Microprocessor Exception Classifications

Synchronous/Asynchronous	Precise/Imprecise	Exception Types
Asynchronous, nonmaskable	Imprecise	Machine check, system reset
Asynchronous, maskable	Precise	External interrupt, decrementer, maintenance exception, performance monitor exception, thermal exception
Synchronous	Precise	Instruction-caused exceptions

These classifications are discussed in greater detail in *Section 4.2*. For a better understanding of precise exceptions, see Chapter 6, “Exceptions” of the *PowerPC Microprocessor Family: The Programming Environments* manual. Exceptions implemented in the 970FX, and conditions that cause them, are listed in *Table 4-2 Exceptions and Conditions*.

Table 4-2. Exceptions and Conditions

Exception Type	Vector Offset (hex)	Causing Conditions
System reset	00100	Assertion of either soft reset input pin or by SCOM command sequence for “soft reset.” See *Section 4.5.1 System Reset Exception*.
Machine check	00200	There are many causes of a machine check exception, see *Section 4.5.2 Machine Check Exceptions*.
Data storage	00300	As specified in the PowerPC Architecture, if a page fault occurs. See *Section 4.5.3 Data Storage Exception*.
Data segment	00380	Data segment fault, as defined in the PowerPC Architecture. See *Section 4.5.4 Data Segment Exception*.
Instruction storage	00400	As defined by the PowerPC Architecture, if a page fault occurs. See *Section 4.5.5 Instruction Storage Exception*.
Instruction segment	00480	Instruction segment fault, as defined in the PowerPC Architecture. See *Section 4.5.6 Instruction Segment Exception*.
External interrupt	00500	An external interrupt is signaled by the assertion of the external interrupt input signal. See *Section 4.5.7 External Interrupt Exception*.
Alignment	00600	There are many causes of the alignment exception. See *Section 4.5.8 Alignment Exception*.
Program	00700	As defined by the PowerPC Architecture (e.g., instruction opcode error). See *Section 4.5.9 Program Exception*.
Floating-point unavailable	00800	As defined by the PowerPC Architecture. See *Section 4.5.10 Floating-Point Unavailable Exception*.
Decrementer	00900	As defined by the PowerPC Architecture, when the most-significant bit of the DEC register changes is ‘1’ and MSR[EE] = ‘1’. It is the responsibility of the decrementer exception service routine to clear DEC[0]. See *Section 4.5.11 Decrementer Exception*.
Hypervisor Decrementer	00980	The Hypervisor Decrementer is similar to the decrementer and is used to return control to the hypervisor. This interrupt is activated when no higher priority interrupt is active and MSR[EE] = ‘1’ or MSR[HV] = ‘0’ and the Hypervisor Decrementer is negative (HDEC[0] = ‘1’. This is a level sensitive interrupt and as such it is the responsibility of the interrupt service routine to clear HDEC[0].
System call	00C00	Execution of the System Call (**sc**) instruction. See *Section 4.5.12 System Call Exception*.
Trace	00D00	MSR[SE] = ‘1’ or MSR[BE] = ‘1’ and a trace-marked instruction successfully completes. See *Section 4.5.13 Trace Exception*.
Performance monitor	00F00	The performance monitor exception is signaled when the MSR[EE] bit is set, the MMCR0[PMXE] bit is set, and any of the performance monitor counters overflow. The performance monitor exception may also be triggered by the ‘0’ to ‘1’ transition of a particular time base bit. See *Section 4.5.14 Performance Monitor Exception*.
VPU unavailable	00F20	A VPU unavailable exception occurs when no higher priority exception exists, an attempt is made to execute an vector instruction, and MSR[VP] = ‘0’. See *Section 4.5.15 VPU Unavailable Exception*.
Instruction address breakpoint	01300	PowerPC 970FX does not support a visible form of the instruction address breakpoint facility. The instruction address breakpoint feature is accessible through the support processor interface. See *Section 4.5.16 Instruction Address Breakpoint Exception*.
Maintenance	01600	This exception can be signaled by a number of internal events, as well as by explicit commands from the support processor. See *Section 4.5.17 Maintenance Exception*.
VPU assist	01700	This exception occurs when operating in Java mode and the input operands or the result of an operation is denormalized. See *Section 4.5.18 VPU Assist Exception*.
Thermal	01800	A thermal exception is signaled by the assertion of the thermal interrupt input signal. See *Section 4.5.19 Thermal Exception*.

4.2 Exception Recognition and Priorities

Exceptions are roughly prioritized by exception class, as follows:

1. Nonmaskable, asynchronous exceptions have priority over all other exceptions. These are system reset and machine check exceptions. These exceptions cannot be delayed and do not wait for completion of any precise exception handling. (However, the machine check exception condition can be disabled so the condition causes the processor to go directly into the checkstop[1] state).
2. Synchronous, precise exceptions are caused by instructions and are taken in strict program order.
3. Imprecise exceptions (imprecise mode floating-point enabled exceptions) are caused by instructions and they are delayed until higher priority exceptions are taken.

   Note: Note that the 970FX does not implement an exception of this type.
4. Maskable asynchronous exceptions (external, decrementer, thermal, maintenance, performance monitor, and exceptions) are delayed if higher priority exceptions are taken.

The following list of exception categories describes how the 970FX handles exceptions up to the point of signalling the appropriate interrupt to occur. Note that a recoverable state is reached if the completed store queue is empty (drained, not cancelled) and any instruction that is next in program order and has been signaled to complete has completed. If MSR[RI] = 0, the 970FX is in a nonrecoverable state. Also, instruction completion is defined as updating all architectural registers associated with that instruction, and then removing that instruction from the completion buffer.

4.2.1 Exception Priorities

The following is a summary of the exception priorities for 970FX:

1. System reset exception
2. Machine check exception
3. Instruction dependent (as follows)
   - Fixed-point loads and stores
     – Mode dependent loads and stores
       (1) Illegal instruction type program exception
       (2) Privileged type program exception (ex: MSR[PR] = '1')
     – Data segment exception
     – Data storage exception
     – Alignment exception
     – Trace exception
   - Floating-point loads and stores
     – Floating-point unavailable exception
     – Data segment exception
     – Data storage exception (DSI)
     – Alignment exception
     – Trace exception
   - Other floating-point instructions
     – Floating-point unavailable exception

1. Hardware has detected a condition that it cannot resolve, and which prevents normal operation. It stops executing instructions, responding to interrupts, and so on.

  - Precise-mode floating-point enabled exceptions type program exception
  - Trace exception
- Vector instructions
  - VPU unavailable exception
  - Trace exception
- **rfid**, **mtmsr**[**d**]
  - Precise-mode FP enabled exceptions type program interrupt
  - Trace exception (for **mtmsr**[**d**] only)
- Other instructions
  - Exceptions that are mutually exclusive and the same priority:

    (1) Trap type program exception
    (2) System call
    (3) Privileged instruction type program exception
    (4) Illegal instruction type program exception

  - Trace exception
  - VPU assist exception
- Instruction segment exception
- Instruction storage exception

4. Maintenance exception
5. External interrupt
6. Performance monitor exception
7. Decrementer exception
8. Thermal exception

4.3 Exception Processing

When an exception is taken, the processor uses SRR0 and SRR1 to save the contents of the MSR for the current context, and to identify where instruction execution should resume after the exception is handled.

4.3.1 Machine Status Save/Restore Register 0 (SRR0)

When an exception occurs, the address saved in SRR0 determines where instruction processing should resume when the exception handler returns control to the interrupted process. Depending on the exception, this may be the address in SRR0 or at the next address in the program flow. All instructions in the program flow preceding this one will have completed execution and no subsequent instruction will have begun execution. This may be the address of the instruction that caused the exception or the next one (as in the case of a system call, trace, or trap exception). The SRR0 register is shown in *Figure 4-1*.

Figure 4-1. Machine Status Save/Restore Register 0 (SRR0)

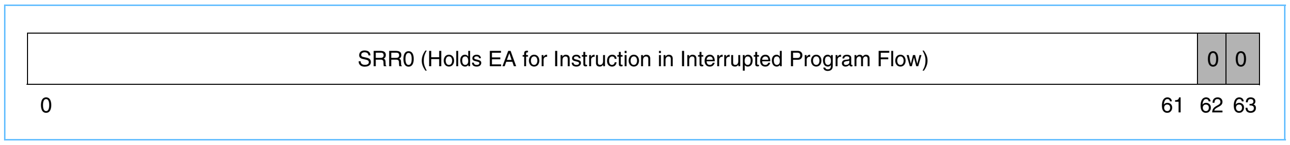


Table 4-3. SRR0 Bit Settings

Bit	Name	Description
0:61		Holds the effective address for the instruction in the interrupted program flow.
62:63	–	Reserved. Returns a zero when read.

4.3.2 Machine Status Save/Restore Register 1 (SRR1)

SRR1 is used to save machine status (selected MSR bits and possibly other status bits as well) on exceptions and to restore those values when an **rfid** instruction is executed. SRR1 is shown in *Figure 4-2*.

Figure 4-2. Machine Status Save/Restore Register 1 (SRR1)

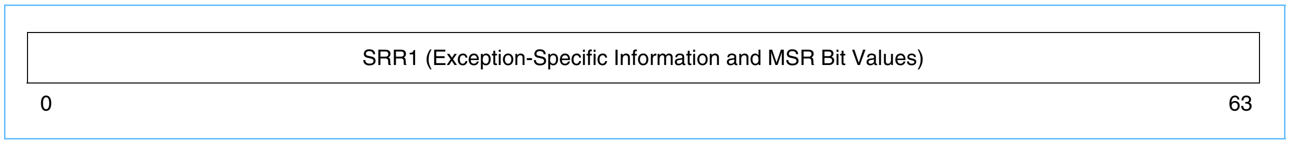


Table 4-4. SRR1 Bit Settings

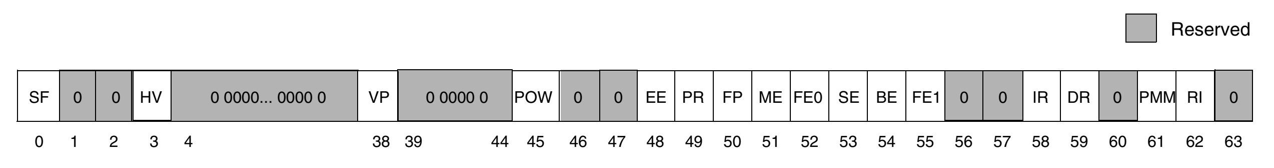
Bit	Name	Description
0:63	SRR1	Exception-specific information and MSR bit values. For most exceptions, bits [33:36] and [42:47] of SRR1 are loaded with exception-specific information. Bits [0:32], [37:41], and [48:63] of SRR1 are loaded with a copy of the corresponding bits of the MSR (prior to taking the exception).

Note: The function of the SRR1 is to save the current state of the machine (that is, the MSR) before a temporary state is invoked to service exceptions. After the servicing of the exception, the contents of SRR1 are returned to the MSR and the code stream can continue.

4.3.3 Machine State Register (MSR)

The 970FX’s MSR is shown in *Figure 4-3*.

Figure 4-3. Machine State Register (MSR)



The MSR bits are defined in *Table 4-5*.

Table 4-5. MSR Bit Settings

Bit	Name	Description
0	SF	Sixty-four bit mode. 0 Processor runs in 32-bit mode. 1 Default mode. Processor runs in 64-bit mode.
1:2	–	Reserved.
3	HV	Hypervisor mode. Set when running on a non-partitioned system or when “hypervisor code” is executing on a partitioned system. MSR[HV] can be set to ‘1’ only by the system call instruction and some interrupts. It can be set to ‘0’ only by the **rfid** and **hrfid** instructions.
4:37	–	Reserved.
38	VPU	VPU available. 0 The processor prevents execution of all vector instructions, including loads, stores, and moves. If such execution is attempted, a VPU unavailable exception is raised. 1 The processor can execute all vector instructions. The VRSAVE register is not protected by MSR [VP]. The data streaming family of instructions (**dst**, **dstt**, **dstst**, **dststt**, **dss**, and **dssall**) are not affected by the MSR[VP].
39:44	–	Reserved.
45	POW	Power management enable. 0 Power management disabled (normal operation mode). 1 Power management enabled (reduced power mode).
46:47	–	Reserved.
48	EE	External exception enable. 0 The processor delays recognition of external exceptions and decrementer exception conditions. 1 The processor is enabled to take an external exception or the decrementer exception. **Note:** Setting MSR[EE] masks not only the architecture-defined external exception and decrementer exceptions, but also the 970FX-specific debug, performance monitor, and thermal exceptions.
49	PR	Problem state. 0 The processor is privileged to execute any instruction. 1 The processor can only execute non-privileged instructions.
50	FP	Floating-point available. 0 The processor prevents dispatch of floating-point instructions, including floating-point loads, stores, and moves. 1 The processor can execute floating-point instructions and can take floating-point enabled program exceptions.

Table 4-5. MSR Bit Settings (Continued)

Bit	Name	Description
51	ME	Machine check enable. 0 Machine check exceptions are disabled. If one occurs system enters checkstop. 1 Machine check exceptions are enabled. Only **rfid** instructions can alter MSR[ME].
52	FE0	IEEE floating-point exception mode 0.
53	SE	Single-step trace enable. 0 The processor executes instructions normally. 1 The processor generates a single-step trace exception upon the successful execution of every instruction except **rfid, isync**, and **sc**. Successful execution means that the instruction caused no other exception.
54	BE	Branch trace enable. 0 The processor executes branch instructions normally. 1 The processor generates a branch type trace exception when a branch instruction executes successfully.
55	FE1	IEEE floating-point exception mode 1.
56:57	—	Reserved.
58	IR	Instruction address translation. 0 Instruction address translation is disabled. 1 Instruction address translation is enabled.
59	DR	Data address translation. 0 Data address translation is disabled. If data stream touch (**dst**) and data stream touch for store (**dstst**) instructions are executed when DR = '0', the results are boundedly undefined. 1 Data address translation is enabled. Data stream touch (**dst**) and data stream touch for store (**dstst**) instructions are supported when DR = '1'.
60	—	Reserved.
61	PMM	Performance monitor mode. This register bit is used to enable/disable performance monitor activity controlled by the process mark bit.
62	RI	Indicates whether system reset or machine check exception is recoverable. 0 Exception is not recoverable. 1 Exception is recoverable. The RI bit indicates whether from the perspective of the processor, it is safe to continue (that is, processor state data such as that saved to SRR0 is valid), but it does not guarantee that the interrupted process is recoverable. Exception handlers must look at SRR1[RI] for determination.
63	–	Reserved.

The 970FX provides precise floating-point exceptions whenever either of the floating-point enabled exception mode bits (MSR[FE0] and MSR[FE1]) are set. In all cases, the 970FX aggressively executes the floating-point instructions (even out-of-order as required), and sorts out any resulting exceptions at completion time. In some cases, due to the *group-oriented instruction tracking* scheme used by the 970FX, when an exception is detected, the hardware will flush the pipeline and re-dispatch the instructions individually in order to provide the precise exception. Since this only happens if an exception is to be taken, it does not represent a measurable decrease in performance.

Table 4-6. IEEE Floating-Point Exception Mode Bits

FE0	FE1	Mode
0	0	Floating-point exceptions disabled.
0	1	Imprecise nonrecoverable. For this setting, the 970FX operates in floating-point precise mode.
1	0	Imprecise recoverable. For this setting, the 970FX operates in floating-point precise mode.
1	1	Floating-point precise mode.

4.3.4 Enabling and Disabling Exceptions

When a condition exists that may cause an exception to be generated, it must be determined whether the exception is enabled for that condition.

- IEEE floating-point enabled exceptions (a type of program exception) are ignored when both MSR[FE0] and MSR[FE1] are cleared. If either bit is set, all IEEE enabled floating-point exceptions are taken and cause a program exception.
- Asynchronous, maskable exceptions (external, decrementer, performance monitor, thermal, and maintenance exceptions) are enabled by setting MSR[EE]. When MSR[EE] = ‘0’, recognition of these exception conditions is delayed. MSR[EE] is cleared automatically when an exception is taken to delay recognition of conditions causing those exceptions.
- A machine check exception can occur only if the machine check enable bit, MSR[ME], is set. If MSR[ME] is cleared, the processor goes directly into checkstop state when a machine check exception condition occurs.
- System reset exceptions cannot be masked.

4.3.5 Exception Processing Steps

After it is determined that the exception can be taken (by confirming that any instruction-caused exceptions occurring earlier in the instruction stream have been handled, and by confirming that the exception is enabled for the exception condition), the processor does the following:

1. Loads the SRR0 with an instruction address that depends on the type of exception. Normally, this is the instruction that would have completed next had the exception not been taken. See the individual exception description for details about how this register is used for specific exceptions.
2. Loads SRR1[33:36, 42:47] with information specific to the exception type.
3. Loads SRR1[0:32, 37:41, 48:63] with a copy of the corresponding MSR bits (prior to the exception).
4. Sets the MSR as described in *Section 4.5 Exception Definitions*. The new values take effect as the first instruction of the exception-handler routine is fetched.

   Note: MSR[IR] and MSR[DR] are cleared for all exception types. Therefore, address translation is disabled for both instruction fetches and data accesses beginning with the first instruction of the exception-handler routine.

Instruction fetch and execution resumes, using the new MSR value, at a location specific to the exception type. The location is determined by adding the exception’s vector offset (see *Table 4-2*) to the value in the Hardware Interrupt Offset Register (HIOR). For a machine check exception that occurs when MSR[ME] = ‘0’ (machine check exceptions are disabled), the checkstop state is entered (the machine stops executing instructions).

4.3.6 Setting the Recoverable Exception in the MSR

The recoverable exception (RI) bit in the MSR was designed to indicate to the exception handler whether the exception is recoverable. When an exception occurs, the RI bit is copied from the MSR to SRR1 and cleared in the MSR. All exceptions are disabled except machine check. If a machine check exception occurs while MSR[RI] is clear, a '0' value is found in SRR1[RI] to indicate that the machine state is definitely not recoverable. When MSR[RI] = '1', the exception is recoverable as far as the current state of the machine and all programs concerned including noncritical machine checks. Thus, in all exceptions, if SRR1[RI] is cleared, the machine state is not recoverable. If it is set, the exception is recoverable with respect to the processor and all programs. An operating system can handle MSR[RI] as follows:

- Use the Special Purpose Registers (SPRG0-SPRG3) to aid in saving the machine state. IBM suggests pointing SPRG0 to a stack save area in memory and saving three General Purpose Registers (GPRs) in SPRG1-3. Move SPRG0 into one of the GPRs that was saved. This GPR now points to the save area in memory. Move the GPRs, SRR0, SRR1, SPRG1-3, and other registers to be used by the exception routine into the stack save area. Update SPGR0 to point to a new save area. Set MSR[RI] to indicate that machine state has been saved. Also set MSR[EE] if you want to re-enable external exceptions.
- When exception processing is complete, clear MSR[EE] and MSR[RI]. Adjust SPRG0 to point to the stack saved area, restore the GPRs, SRR0 and SRR1, and any other register that you may have saved, execute **rfid**. This returns the processor to the interrupted program.

4.3.7 Returning from an Exception Handler

The Return from Interrupt Doubleword (**rfid**) instruction performs context synchronization by allowing previously-issued instructions to complete before returning to the interrupted process. In general, execution of the **rfid** instruction ensures the following:

- All previous instructions have completed to a point where they can no longer cause an exception.
- Previous instructions complete execution in the context (privilege, protection, and address translation) under which they were issued.
- The **rfid** instruction copies SRR1 bits back into the MSR and resets the MSR[POW] bit.
- Instructions fetched after this instruction execute in the context established by this instruction.
- Program execution resumes at the instruction indicated by SRR0.

For a complete description of context synchronization, see Chapter 6, "Exceptions" of the *PowerPC Microprocessor Family: The Programming Environments* manual.

4.4 Process Switching

The following instructions are useful for restoring proper context during process switching:

- The Synchronize (**sync**) instruction orders the effects of instruction execution. All instructions previously initiated appear to have completed before the **sync** instruction completes, and no subsequent instructions appear to be initiated until the **sync** instruction completes.
- The Instruction Synchronize (**isync**) instruction waits for all previous instructions to complete and then discards any fetched instructions, causing subsequent instructions to be fetched (or refetched) from memory and to execute in the context (privilege, translation, and protection) established by the previous instructions.
- The **stwcx./stdcx.** instruction clears any outstanding reservations, ensuring that an **lwarx/ldarx** instruction in an old process is not paired with an **stwcx./stdcx.** instruction in a new one.
- The Store Word Conditional Indexed/Store Doubleword Conditional Indexed (**stwcx./stdcx.**) instruction clears any outstanding reservations, ensuring that a Load Word and Reserve Indexed/Load Double Word and Reserve Indexed (**lwarx/ldarx)** instruction in an old process is not paired with an **stwcx./stdcx.** instruction in a new one.

The operating system should set MSR[RI] as described in *Section 4.3.6 Setting the Recoverable Exception in the MSR*.

4.5 Exception Definitions

When an exception/interrupt is taken, all bits in the MSR are set to 0, with the following exceptions:

- Exceptions always set MSR[SF] to 1.
- Only the machine check exception sets MSR[ME] to 0. All other exceptions leave MSR[ME] unchanged.

The following sections describe the implementation dependent aspects of the exceptions.

Note: If a description is not provided, the 970FX behaves as described in the *PowerPC Architecture Books*.

4.5.1 System Reset Exception

The system reset exception is a non-maskable, asynchronous exception that is caused by the assertion of either the soft reset input pin, or by the SCOM command sequence for soft reset.

The Not Hard Reset bit in HID0[15] can be used to help software distinguish between a hard reset and a soft reset. To use this capability, software should initially set this bit to a ‘1’. Later, when a system reset exception is taken, software can check the state of this bit to determine which type of reset occurred. If the bit is still set, then the reset was a soft reset, and if the bit is a zero, the reset was a hard reset.

4.5.2 Machine Check Exceptions

The are several possible causes of machine check exceptions in the 970FX, some of which are generally recoverable, and some of which are non-recoverable.

The following causes of machine check exceptions are precise and synchronous with the instruction that caused the operation that encountered the error (that is, SRR0 contains the address of the instruction that caused the operation).

1. The detection of a parity error in the L1 D-cache, the L1 D-cache Tag, the D-ERAT, the TLB, or the SLB during the execution of a load or store instruction. If the exception is caused by a soft error, then executing the appropriate sequence of instructions in the machine check handler program will clear the error condition without causing any loss of state, permitting the interrupted program to resume if MSR[RI] was a '1' when the instruction that encountered the error was executed.

   Note: The L1 D-cache and the L1 D-cache tag parity errors are recovered by hardware in the 970FX (default mode), without a machine check interrupt.
2. The detection of an uncorrectable error checking and correction (ECC) error in the L2 cache when a load instruction is executed.
3. The detection of an uncorrectable ECC error in the L2 cache while the page table is being searched in the process of translating an address.
4. The detection of erroneous data that is being returned to satisfy a load instruction for which the effective address specified a location in caching inhibited memory.

For *hard errors*, these characteristics cannot be reliably provided on a machine check, because it is likely that the failure will prevent reliable execution. Additionally, a machine check exception that occurs when MSR[ME] = '0' results in a checkstop.

In addition, there are a few possible sources for asynchronous machine check exceptions. A machine check exception is taken when the machine check input pin is asserted, (if enabled by setting HID0[32] = '1'). The Fault Isolation Register (FIR), debug logic, and hang recovery logic can also be programmed to induce machine check exceptions for various error conditions. Since these signals are asynchronous with respect to the executing program, asynchronous machine checks may or may not be recoverable. Software can use the MSR[RI] bit to help identify the cases where the machine check exception is recoverable.

Information about the suspected source of the error condition is logged into either the SRR1 register, the DSISR register, or both as defined in *Table 4-7* for synchronous machine checks.

Table 4-7. Register Settings for Machine Check Exception

Register	Bits	Setting
SRR0	0:63	Effective address of the next instruction that would have executed if the machine check exception was not taken. When this is a recoverable machine check due to a load that has surfaced an error, this will be the address of the load instruction itself (the 970FX allows the instruction to execute to surface the error, but inhibits the commitment of the results). When this is a recoverable machine check due to an instruction fetch surfacing an error, this will be the address of an instruction that initiated the memory/cache access.
SRR1	0:41	Loaded from MSR.
	42	Exception caused by Instruction Fetch Unit (IFU) detection of a hardware uncorrectable error (UE).
	43	Exception caused by load/store detection of error (see DSISR below).
	44:45	Exception cause indicated by the following encoding: 00 No error encoded. 01 Exception caused by an SLB parity error detected while translating an instruction fetch address. 10 Exception caused by a TLB parity error detected while translating an instruction fetch address. 11 Exception caused by a hardware uncorrectable error (UE) detected while doing a reload of an instruction-fetch TLB tablewalk.
	46:61	Loaded from MSR.
	62	Loaded from MSR[62] if recoverable. Otherwise, set to zero.
	63	Loaded from MSR.
DSISR	0:15	All zeros.
	16)	Exception caused by a UE deferred error (DAR is undefined).
	17)	Exception caused by a UE deferred error during a tablewalk (D-side).
	18	Exception was caused by a software recoverable parity error in the L1 D-cache.
	19	Exception was caused by a software recoverable parity error in the L1 D-cache tag.
	20	Exception was caused by a software recoverable parity error in the D-ERAT.
	21	Exception was caused by a software recoverable parity error in the TLB.
	22	Zero.
	23	Exception was caused by an SLB parity error (may not be recoverable). This condition could occur if the effective segment ID (ESID) fields of two or more SLB entries contain the same value.
	24:31	All zeros.
DAR	0:63	Effective address computed by a load or store instruction that caused the operation which encountered a parity error in the D-ERAT, TLB, or SLB, or that encountered an uncorrectable error (UE) while attempting to reload a TLB entry. Effective address computed by the load instruction that caused the operation that encountered a parity error in the L1 D-cache or L1 D-cache TAG arrays. For all other types of machine check exceptions, the DAR is undefined (including the case where the operand of the load instruction contains a UE).

Note: As mentioned above, the machine check exception handler is expected to help hardware recover from certain types of D-cache, D-cache directory, D-ERAT, and TLB errors detected by the hardware. In general terms, the exception handler should:

- Check whether or not the machine check exception is recoverable by looking at the state of the RI bit in SRR1
- Determine the type of error that caused the machine check by looking at the state of the SRR1 and DSISR Registers

- Flush the contents of the array that reported the detected error (this process is slightly different for each of the possible arrays)
- Return to the interrupted process.

If no error is encoded in SRR1[44:45], then the exception is likely caused by an asynchronous machine check, in which case the exception handler should access the Asynchronous Machine Check Register through the SCOMC facility.

4.5.3 Data Storage Exception

The 970FX implements the data storage exception as described in the PowerPC Architecture (OEA). A DSI exception occurs when no higher priority exception exists and an error condition related to a data memory access occurs. In case of a TLB miss for a load, store, or cache operation, a DSI exception is taken if the resulting hardware table search causes a page fault.

When this exception is taken, execution resumes at effective address x'00300'.

4.5.4 Data Segment Exception

The 970FX implements the data segment exception as described in the PowerPC Architecture (OEA). A data segment exception occurs when no higher priority exception exists and a data access cannot be performed because data address translation is enabled (MSR[DR] = '1') and the effective address of any byte of the storage location specified by a Load, Store, **icbi**, **dcbz**, **dcbst**, **dcbf**, **eciwx**, or **ecowx** instruction cannot be translated to a virtual address.

When this exception is taken, execution resumes at effective address x'00380'.

4.5.5 Instruction Storage Exception

The 970FX implements the instruction storage exception as described in the PowerPC Architecture (OEA). An ISI exception occurs when no higher priority exception exists and an attempt to fetch the next instruction fails.

When this exception is taken, execution resumes at effective address x'00400'.

4.5.6 Instruction Segment Exception

The 970FX implements the instruction segment exception as described in the PowerPC Architecture (OEA). An instruction segment exception occurs when no higher priority exception exists and next instruction to be executed cannot be fetched because instruction address translation is enabled (MSR[IR] = '1') and the effective address cannot be translated to a virtual address.

When this exception is taken, execution resumes at effective address x'00480'.

4.5.7 External Interrupt Exception

In the 970FX, an external interrupt is signaled by the assertion of the external interrupt input signal. The external interrupt signal is expected to remain asserted until the processor has actually taken the interrupt (failure to meet this requirement may lead the processor to not recognize the interrupt request).

4.5.8 Alignment Exception

An alignment exception is taken if any of the following conditions are detected:

- **lwarx**, **stwcx, lmw, stmw** instructions with non-word aligned addresses
- **ldarx** and **stdcx** instructions with non-doubleword aligned addresses
- **lmw** and **stmw** instructions to storage marked cache-inhibited
- **lswi**, **lswx**, **stswi**, and **stswx** instructions to storage marked cache-inhibited
- **dcbz** to storage marked cache-inhibited (a **dcbz** to cache-inhibited space is treated as a no-op instead of causing an alignment interrupt if the dcbz_ieq1_align bit in the mode ring is set to a 0)
- Any load or store to storage marked cache-inhibited that is not naturally aligned
- Floating-point load single instructions that are not word aligned and cross a 32-byte boundary
- Floating-point store instructions that are not word aligned and cross a 4 KB boundary
- When HID4[24] is set, some forms of unaligned storage accesses that are normally handled by the hardware are forced to take an alignment exception (to assist in debugging).

Table 4-8. Register Settings for Alignment Exception

Register	Bits	Setting
DSISR	(0:31)	Unchanged
DAR	(0:63)	Set to the effective address computed by the load or store instruction that caused the alignment exception. When the exception is caused by an unsupported access to cache-inhibited space, the DAR will be set to the effective address of the first access into the cache-inhibited space.

4.5.9 Program Exception

The 970FX implements the program exception as it is defined by the PowerPC Architecture (OEA). A program exception occurs when no higher priority exception exists and one or more of the exception conditions defined in the OEA occur.

The 970FX invokes the program exception for a system illegal instruction when it detects any instruction from the illegal instruction class. The 970FX fully decodes the SPR field of the instruction. If an undefined SPR is specified, a program exception is taken.

When this exception is taken, execution resumes at effective address x'00700'.

4.5.10 Floating-Point Unavailable Exception

The floating-point unavailable exception is implemented as defined in the PowerPC Architecture. When a floating-point unavailable exception is taken, instruction fetching resumes at the location determined by adding the offset x'00800' to the HIOR value.

4.5.11 Decrementer Exception

The decrementer exception is implemented as defined in the PowerPC Architecture. A decrementer exception occurs when no higher priority exception exists, the decrementer is negative (DEC[0] = ‘1’), and MSR[EE] = ‘1’. The decrementer exception is level sensitive. It is the responsibility of the interrupt service routine to clear DEC[0].

When this exception is taken, execution resumes at effective address x‘0000_0000_0000_0900’.

4.5.12 System Call Exception

The 970FX implements the system call exception as described in the PowerPC Architecture (OEA). A system call exception occurs when a system call (**sc**) instruction is executed.

When this exception is taken, execution resumes at effective address x‘00C00’.

4.5.13 Trace Exception

The trace exception is taken when the single-step trace enable bit (MSR[SE]) or the branch trace enable bit (MSR[BE]) is set and an instruction successfully completes. After a trace exception is taken, SRR0, SRR1, SIAR, and SDAR are set as shown in *Table 4-9*.

Table 4-9. Register Settings for Trace Exception

Register	Bits	Setting
SRR0	0:63	Set as specified in the architecture.
SRR1	0:32	Loaded from the MSR
	33:34	‘10’
	35	Set for a load instruction, otherwise cleared. Not set for a zero length **lswx**.
	36	Set for a store instruction, otherwise cleared. Not set for a zero length **stswx**.
	37:41	Loaded from the MSR
	42	Set for a **lwarx/ldarx** or **stwcx/stdcx** instruction; otherwise cleared
	43	Set to a ‘1’.
	44	Set to a ‘0’.
	45:47	Set to a ‘0’.
	48:63	Loaded from the MSR.
SIAR	0:63	Set to the effective address of the traced instruction.
SDAR	0:63	If the instruction that took the trace interrupt was a storage access instruction, the SDAR is set to the effective address of the storage access. SDAR is not set if an X-form Load String or Store String instruction specifies an operand length of zero.

If either MSR bits SE or BE is set to ‘1’ by a Return from Interrupt or Move to MSR instruction, the contents of SIAR and SDAR are undefined until a trace interrupt occurs.

4.5.14 Performance Monitor Exception

The performance monitor exception is signalled when the MSR[EE] bit is set, and a performance monitor exception condition occurs. See *Chapter 10. 970FX Performance Monitor* for a description of performance monitor exception conditions.

The following registers are set when a performance monitor exception occurs.

Table 4-10. Register Settings for the Performance Monitor Exception

Register	Bits	Setting
SRR0	0:63	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.
SRR1	0:32	Loaded from the MSR.
	33	Set to ‘1’ if the contents of the SDAR and the SIAR registers are associated with the same instruction.
	34:63	Loaded from the MSR.
SIAR	0:63	Set to the effective address of the marked instruction, where the marked instruction is an instruction that was executing, possibly out-of-order, at or around the time that the performance monitor exception occurred. The contents of the SIAR may be altered by the processor if and only if MMCR0[PMEE] = ‘1’. Thus, after a performance monitor exception occurs, the contents of SIAR are not altered by the processor until software sets MMCR0[PMEE] = ‘1’. After software sets MMCR0[PMEE] = ‘1’, the contents of SIAR are undefined until the next performance monitor exception occurs.
SDAR	0:63	Set to the effective address of the storage operand of an instruction that was executing, possibly out-of-order, at or around the time that the performance monitor exception occurred. This storage operand is called the marked data and may be, but need not be, the storage operand (if any) of the marked instruction. If the performance monitor exception causes a performance monitor interrupt, SRR1 indicates whether the marked data is in fact the storage operand of the marked instruction. The contents of the SDAR may be altered by the processor if and only if MMCR0[PMEE] = ‘1’. Thus, after a performance monitor exception occurs, the contents of SDAR are not altered by the processor until software sets MMCR0[PMEE] = ‘1’. After software sets MMCR0[PMEE] = ‘1’, the contents of SDAR are undefined until the next performance monitor exception occurs.

4.5.15 VPU Unavailable Exception

This exception occurs if there is an attempt to execute any vector instruction, including a vector load or store, with MSR[VP] negated. After this interrupt, execution resumes at offset x‘0000_0000_0000_0F20’. The register settings for this interrupt are shown in *Table 4-11*.

Note: A **mtspr** or **mfspr** instruction that references the VRSAVE register will not cause this interrupt.

Table 4-11. Register Settings for VPU Unavailable Interrupt

Register	Bits	Setting
SRR0	0:63	Set to the effective address of the instruction that caused the interrupt.
SRR1	0:32	Loaded from the MSR.
	33:36	Set to ‘0’.
	37:41	Loaded from the MSR.
	42:47	Set to ‘0’.
	48:63	Loaded from the MSR.

4.5.16 Instruction Address Breakpoint Exception

The PowerPC 970FX does not support a visible form of the instruction address breakpoint facility. The instruction address breakpoint feature is accessible through the support processor interface.

When this exception is taken, execution resumes at the effective address x‘01300’.

4.5.17 Maintenance Exception

The 970FX provides support for an implementation-dependent maintenance exception. This exception can be signaled by a number of internal events, as well as by explicit commands from the support processor.

When this exception is taken, execution resumes at the effective address x‘0000_0000_0000_1600’.

This exception is controlled by the MSR[EE] bit (in a manner similar to external interrupts). The register settings for this exception are shown in *Table 4-12*.

Table 4-12. Register Settings for Maintenance Exception

Register	Bits	Setting
SRR0	0:63	Set to the effective address of the next instruction that would have executed had the exception not been taken.
SRR1	0:32	Loaded from the MSR.
	33:36	Set to ‘0’.
	37:41	Loaded from the MSR.
	42:47	Set to ‘0’ (may be used later to distinguish various causes of exception).
	48:63	Loaded from the MSR.

4.5.18 VPU Assist Exception

This exception occurs when operating in Java mode and the input operands or the result of an operation are denormalized.

The register settings for this exception are shown in *Table 4-13*.

Table 4-13. Register Settings for VPU Assist Exception

Register	Bits	Setting
SRR0	0:63	Set to the effective address of the instruction that caused the exception.
SRR1	0:32	Loaded from the MSR.
	33:36	Set to ‘0’.
	37:41	Loaded from the MSR.
	42:47	Set to ‘0’.
	48:63	Loaded from the MSR.

When this exception is taken, execution resumes at offset x‘0000_0000_0000_1700’.

4.5.19 Thermal Exception

In the 970FX, a thermal exception is signaled by the assertion of the thermal interrupt input signal.

The register settings for this exception are shown in *Table 4-14*.

Table 4-14. Register Settings for Thermal Exception

Register	Bits	Setting
SRR0	0:63	Set to the effective address of the next instruction that would have executed had the exception not been taken.
SRR1	0:32	Loaded from the MSR.
	33:36	Set to ‘0’.
	37:41	Loaded from the MSR.
	42:47	Set to ‘0’.
	48:63	Loaded from the MSR.

When this exception is taken, execution resumes at offset x‘0000_0000_0000_1800’.

5. Memory Management

This chapter describes the 970FX microprocessor’s implementation of the memory management unit (MMU) specifications provided by the operating environment architecture (OEA) for PowerPC processors. The primary function of the MMU in a PowerPC processor is the translation of logical (effective) addresses to physical addresses (referred to as real addresses in the architecture specification) for memory accesses and I/O accesses (I/O accesses are assumed to be memory-mapped). In addition, the MMU provides access protection on a segment or page basis. This chapter describes the specific hardware used to implement the MMU model of the OEA in the 970FX. See the *PowerPC Operating Environment Architecture (Book III)* for a conceptual overview of the memory management model.

Two general types of memory accesses generated by PowerPC processors require address translation—instruction accesses and data accesses that are generated by load and store instructions. Generally, the address translation mechanism is defined in terms of the segment descriptors and page tables that the PowerPC processors use to locate the effective-to-physical address mapping for memory accesses. The segment information translates the effective address to an interim virtual address, and the page table information translates the interim virtual address to a physical address.

The segment descriptors, used to generate the interim virtual addresses, reside as segment table entries (STEs) in memory. The 970FX uses a segment lookaside buffer (SLB) on-chip that caches recently used segment table entries. In addition, a translation lookaside buffer (TLB) is implemented on the 970FX to keep recently-used page address translations on-chip.

The MMU, together with the exception processing mechanism, provides the necessary support for the operating system to implement a paged virtual memory environment and to enforce protection of designated memory areas. Exception processing is described in *Chapter 4 Exceptions*. Specifically, *Section 4.3 Exception Processing* describes the MSR, which controls some of the critical functionality of the MMUs.

5.1 MMU Overview

The 970FX implements the memory management specification of the PowerPC operating environment architecture for 64-bit implementations. The 970FX supports a 65-bit virtual address and a 42-bit physical (real) address.

Basic features of the MMU implementation in 970FX as defined by the OEA are:

- Support for real addressing mode—Effective-to-physical address translation can be disabled separately for data and instruction accesses.
- Segmented address translation—The 64-bit effective address is translated to a 65-bit virtual address. This 65-bit virtual address space is divided into 4 KB or 16 MB pages, each of which can be mapped to a physical page.

The 970FX also provides the following features that are not required by the PowerPC Architecture:

- Unified translation lookaside buffer (TLB)—The 1024-entry, 4-way set-associative TLB supports:
  - A new large page architecture (16 MB large pages supported).
  - Hardware-based reload (from the L2 cache interface in order to ensure no L1 D-cache impact).
  - Hardware-based update of the reference (R) and change (C) bits in a page table entry (PTE).
  - Parity protection; precise machine check interrupt on parity error (software fix-up).
  - Recently-used page address translations cached on-chip.
- Segment lookaside buffer (SLB)–The 64-entry, fully associative SLB supports:
  - Software reload of the SLB. An SLB miss results in an interrupt.
  - 32-bit PowerPC segment register instructions load the SLB.
- TLB invalidation:
  The 970FX implements the optional TLB Invalidate Entry (**tlbie**) and TLB Synchronize (**tlbsync**) instructions, which can be used to invalidate TLB entries.
- Little endian mode is not supported.

Figure 5-1 summarizes the 970FX MMU features, including those defined by the PowerPC Architecture (OEA) for 64-bit processors and those specific to the 970FX.

Table 5-1. MMU Feature Summary

Feature Category	Architecturally Defined/ 970FX-Specific	Feature
Address ranges	Architecturally defined	2^{64} bytes of effective address
	970FX-specific	2^{65} bytes of virtual address
		2^{42} bytes of physical address
Page size	Architecturally defined	4 KB
	970FX-specific	16 MB
Segment size	Architecturally defined	256 MB
Memory protection	Architecturally defined	Segments selectable as no-execute
		Pages selectable as user/supervisor and read-only or guarded
Page history	Architecturally defined	Referenced and changed bits defined and maintained
Page address translation	Architecturally defined	Translations stored as PTEs in hashed page tables in memory
		Page table size determined by mask in SDR1 register

Table 5-1. MMU Feature Summary (Continued)

Feature Category	Architecturally Defined/ 970FX-Specific	Feature
TLB	Architecturally defined	Instructions for maintaining TLBs (**tlbie** and **tlbsync** instructions in the 970FX)
	970FX-specific	1024-entry, 4-way, set-associative TLB (combined for both instruction and data).
Page table search support	970FX-specific	The 970FX performs the table search operation in hardware.
Segment descriptors	Architecturally defined	Stored as STEs in hashed segment tables in memory
	970FX-specific	64-entry fully associative SLB
Segment table search support	970FX-specific	The 970FX provides support for software reload of the SLB.

5.1.1 Speculative Storage Accesses

The 970FX is capable of executing load instructions to non-guarded, cacheable storage speculatively. This can occur when a load instruction is encountered on a predicted branch path, or when a logically preceding instruction causes an interrupt. As a result, it is possible for a speculative load that misses in the on-chip cache hierarchy to initiate an external storage request, even if that load instruction is not actually executed as part of the true instruction stream.

5.1.2 Storage Protection

When address translation is enabled, the protection mechanism is controlled by the following:

- MSR[PR] which distinguishes between supervisor (privileged) state and user (problem) state.
- K_S and K_P, the supervisor (privileged) state and the user (problem) state storage key bits in the SLB entry used to translate the effective address.
- For instruction fetches only:
  - the N (no-execute) value used for the access
  - the G (guarded) bit in the page table entry used to translate the effective address.

  Thus, for an instruction fetch, access is not permitted if the N value is '1', or if G equals '1'.

5.1.3 Storage Access Modes (WIMG Bits)

Storage access modes are controlled by the write-through/caching-inhibited/memory-coherency enforced/guarded bits (WIMG) bits. The 970FX does not support the optional W bit or the optional M bit. All accesses are treated as though W = '0' and M = '1' independent of the value of these bits in the page table. Furthermore, when the hardware is performing a *change bit* update, it will write the W and M bits as '0' and '1', respectively.

Table 5-2 summarizes the treatment of the WIMG bits in the 970FX:

Table 5-2. Treatment of WIMG Bits in the 970FX

WIMG	Description
x1xx	Treated as WIMG = '0111', for loads
	Treated as WIMG = '011x,' for stores
x0x1	Treated as WIMG = '0011'
x0x0	Treated as WIMG = '0010'

5.1.4 Support for 32-Bit Operating Systems

The 970FX supports most of the optional bridge facilities and instructions for 64-bit implementations.

The bridge facility can be used to ease the transition to the PowerPC software-managed segment lookaside buffer (SLB) architecture, from either the segment register architecture provided by the 32-bit PowerPC implementation or the hardware-accessed segment table architecture provided by the 64-bit PowerPC implementations. The bridge facility permits the operating system to continue to use the 32-bit PowerPC implementation's segment register manipulation instructions and to continue to use the address space register (ASR).

Associated with this support, the following optional instructions are supported:

- **mtsr** - *Move to segment register*
- **mtsrin** - *Move to segment register indirect*
- **mfsr** - *Move from segment register*
- **mfsrin** - *Move from segment register indirect*
- **mtmsr** - *Move to machine state register (32-bit)*

These instructions allow software to associate effective segments 0 through 15 with any of the virtual segments 0 through 2^{37}-1. SLB entries 0:15 serve as virtual segment registers, with SLB entry *i* used to emulate segment register *i*. The **mtsr** and **mtsrin** instructions move 32 bits from a selected GPR to a selected SLB entry. The **mfsr** and **mfsrin** instructions move 32 bits from a selected SLB entry to a selected GPR.

5.2 Real Addressing Mode

If address translation is disabled (MSR[IR] = ‘0 ‘or MSR[DR] = ‘0’) for a particular access, the effective address is treated as the physical address and is passed directly to the memory subsystem. These MSR bits are forced to '1' when running in user mode.

The WIMG bits for storage access in real addressing mode are determined as follows. The W and M bits are not supported in the 970FX, and are considered to always have values of W = ‘0’ and M = ‘1’. The G-bit is always asserted in real addressing mode. For data accesses, HID4[23] determines the value of the I bit in real addressing mode. For instruction accesses, HID1[10] can be used to force the value of the I bit to '1', although this value applies to address translation mode, as well as to real addressing mode.

6. Software Optimization Guidelines

This section highlights some 970FX microprocessor characteristics and conditions that should be considered when developing software.

6.1 Design Characteristics

The 970FX microprocessor has long pipelines with the following characteristics:

- There are six cycles from the instruction fetch to dispatch (dispatch is the sixth cycle).
- Complex instructions are broken down into sequences of simple internal operations.
- Some instructions stall in dispatch until certain interlocks are released.
  - The primary interlock is called the “non-rename scoreboard” bit.
  - Only one scoreboard bit exists for all scoreboarded resources.
  - Instructions that write a non-renamed resource set the non-rename scoreboard bit when dispatched and reset this bit when complete.
  - All SPRs are scoreboarded except: LR, CTR, and the following bits in XER: CA, OV.
  - Instructions that use or read from the non-renamed registers stall in the dispatch unit until the flag clears.
- Instructions that set the scoreboard also typically end a dispatch group and are completion serialized (wait until next-to-complete before eligible for execution).
- Dispatch receives groups, which are a unit of tracking.
  - Up to 20 groups active after dispatch (80 - 100 PowerPC instructions).
  - Four to seven cycles from dispatch to finish.

The 970FX microprocessor has multiple execution units:

- Two load/store units (LSU)
- Two floating-point units (FPU)
- Two fixed-point units (FXU) (that are symmetric except that FX1 does divides and FX0 does SPR access)
- One branch unit (BRU)
- One condition register unit (CRU)

The 970FX microprocessor utilizes out-of-order execution:

- Execution is in-order until dispatch has placed instructions into issue queues.
- Instructions issued from queues to execution units are out-of-order.
- Instructions complete in order.

The 970FX microprocessor has the following load/store unit characteristics:

- Complicated loads and stores are broken up by decode unit.
  - **lmw** and **stmw** are converted to a stream of single-register loads and stores. String instructions generate a similar stream, except that X-form string instructions cause generation of internal operations to read the byte count field from the XER, causing a dispatch stall if the XER setting instruction has not executed.
- Problems are handled by flush, refetch group, or dispatch as single-instruction groups.
- Loads that are dependent on a store in the same group cause a flush if forwarding is not possible. This is because the load must wait until the store has updated the cache, but the cache update must be non-speculative and can only be done after the store completes. Completion is done on a group basis, and can only be done when all internal operations (IOPs) in the group have finished. Therefore, the entire group is flushed. When decoded, the load is forced into a separate group.
- Any load with data that crosses a 64-byte boundary (32-byte boundary if a load misses in the L1 cache) causes flush and microcode expansion. If the offending load is an IOP generated by the microcode expansion of a string instruction, the entire PowerPC instruction is flushed and re-expanded such that each register's data is processed by two loads/stores and a merge.
- Loads dependent upon a store, but executed early (load executes before store), cause a flush.
- Flush and refetch costs about 20 cycles. Misaligned loads usually are flushed twice; once to get the load isolated in a dispatch group, and the second time to generate the microcoded sequence of IOPs to fetch the data and splice it together.
- The data prefetch engine can prefetch eight active streams.

The 970FX microprocessor uses the following memory hierarchy for data:

- The L1 data cache is a 32 Kb, 128-byte line with a 2-cycle latency.
  - The L1 D-cache is store-through.
  - A store miss in the L1 data cache does not establish a line in the L1 D-cache.
  - Cache reloads are 32 bytes per cycle.
- The L2 cache is a 1 Mb, 128-byte line.

The 970FX microprocessor decode unit has the following features:

- Processes a stream of PowerPC instructions and forms dispatch groups.
  - Branches always force an end of current group.
  - Some instructions are forced to be first in a group. For example: **divw**, CR logical.
- Cracking generates two IOPs from one PowerPC instruction. For example:
  - All update forms (load/store + add(i) to update register)
  - X-form fixed-point stores (add + store)
  - Load algebraic (load + extend sign)
  - Many record forms (basic arithmetic + compare immediate)
  - Fixed-point divides
  - All CR-logicals except destructive forms (**r**D = **r**B)

  - Both IOPs of the cracked instruction must be in the same group. This forces the cracked pair to start a new group if the original instruction was last in the previous group (and there was no room for the second IOP).
- Microcode: generate three or more IOPs from a single PowerPC instruction.
  - Microcoded instructions generate one or more groups, thus forcing an end of the previous group. For example:
    - **lmw** and **lswi** (all multiples and string instructions)
    - **mtcrf** (more than one target field)
    - **mtxer** and **mfxer**
- Some instructions are forced to be first in a group. For example:
  - Fixed-point divide (also cracked)
  - **addc**/**subfc** (also cracked)
  - **mtspr**/**mfspr** (to satisfy FX0/LSU0/CRU execution requirement)
  - CR-logicals (can also be cracked)

The instruction fetch unit (IFU) has the following characteristics:

- Fetches are aligned on 8-word blocks
- It takes three cycles to redirect a fetch from Next-Sequential. For example, there are two dead cycles between the last fetch of a block containing a branch and fetching the branch target.
- The fetcher cannot handle a new fetch block until all branches in the current block have been recorded in the branch instruction queue (BIQ) for future resolution. Only branches between the branch target address and the end of the block are significant. These branches are recorded two per cycle, so the maximum time required is four cycles.

Branch prediction has the following characteristics:

- Predicts both direction (conditional) and address (to Link or Count).
- Highly accurate (95%) for most codes.
- Accuracy can be improved with hint bits.
- About 11 cycles are needed to correct a wrong guess.
- Replacing conditional branches with alternative code is likely to be a win (some fixed-point maximum, minimum, select).

Dispatch, issue, and issue queues have the following characteristics:

- Dispatch performs register renaming (mapping), scoreboard dependency checking, and distribution to correct the issue queue.
- Six instruction queues
  - FPQ0 (10 IOPs) feeding FPU0
  - FPQ1 (10 IOPs) feeding FPU1
  - FXQ0 (18 IOPs) feeding FXU0 and LSU0
  - FXQ1 (18 IOPs) feeding FXU1 and LSU1
  - BRQ (12 IOPs)
  - CRQ (10 IOPs)

- There is a fixed relationship between the dispatch group slot and the target instruction queue.
  - Slot 0: FPQ0, FXQ0, CRQ
  - Slot 1: FPQ1, FXQ1, CRQ
  - Slot 2: FPQ1, FXQ1
  - Slot 3: FPQ0, FXQ0
  - Slot 4: BRQ
- In addition, the FX, FP, and CR queues are subdivided into even and odd subqueues. The attached execution units can obtain IOPs from either subqueue, but IOPs always stay in the subqueue to which they were initially dispatched. Each subqueue has half the total capacity of the queue. Thus:
  - Slot 0: FXQ0-O or FPQ0-O or CRQ-O
  - Slot 1: FXQ1-O or FPQ1-O or CRQ-E
  - Slot 2: FXQ1-E or FPQ1-E
  - Slot 3: FXQ0-E or FPQ0-E
- IOPs are issued from the queues when all operands are ready, and there is an execution unit available; IOPs can be issued the next cycle after dispatch.
- Dependent IOPs cannot be issued back-to-back. That is, dependent instructions can be issued only every other cycle (assuming that they execute in one cycle)
- IOPs can be artificially serialized by being dispatched to the same FX queue. Thus, suboptimal scheduling might cause underutilization of one of two symmetric execution units.

6.2 Software Considerations for the 970FX Microprocessor

Software for the 970FX microprocessor needs to consider the following conditions:

- XER has non-renamed fields.
- X-form string instructions are slowed down; therefore, it is best to avoid these instructions.
- **mtxer** drains the functional units.
- SPRs are not renamed except for CTR, LR, and some XER fields. Referencing non-renamed SPRs causes pipeline drain.
- There is a scoreboard interlock between an **mtspr** and the next subsequent **mfspr** such that the **mfspr** is held in the dispatch until the scoreboard goes off (when the last **mtspr** completes).
- The **mtsr** instruction is not recommended, because it is scoreboarded and forces execution serialization.
- The L1 data cache is write-through, and stores the miss in the L1 cache that does not establish the line in the L1 cache, but establishes only the line in the L2 cache.
- Loads dependent upon previous stores can be slow, and can trigger a flush and refetch. They should be scheduled, so that they are dispatched in separate groups.
- Store forwarding: If the store data is in the store-reorder queue (SRQ), then the data can be forwarded to the load (as if the load hit the L1 cache).

  This is possible only when the data loaded is completely contained in the data from the store.
  For example:

  - **lw** following an **stw** to the same address
  - **lh** following an **stw** to the same address

  - **lbz** from any byte in the word stored by an **stw**
  - **lw** from one of the words stored by an **stfd**

  If the bytes loaded overlap the bytes stored, then no forwarding can be done, and the load appears to stall until the store data has been written to the cache. For example:

  - **lfd** following an **stw**
  - **lw** following an **sth** to the same address

  If the store and load are in the same dispatch group, then a flush and refetch is done so that they will be in different groups to permit completion of the store.

  If the load executes before the store address is computed, a flush and refetch occurs. The first re-executed instruction is the "load/next" after the store. To prevent this, schedule the dependent load four instructions (or more) after the store.

- Because instructions are tracked internally in groups, dependent instructions must be arranged so that they are in separate groups. This minimizes the length of time the individual instructions are in the execution section of the machine.
- Use instructions that minimize cracking or microcode expansion. This maximizes utilization of the dispatch buffer. For example:
  - Use update forms, which are always cracked, if the cracked pair does not cause early group termination. Using update forms helps to reduce the code footprint in the instruction cache.
  - Do not use X-form fixed-point stores (always cracked and sometimes microcoded)
- The granularity of reservations (**lwarx**/**stwcx.**) is the data cache line, which is 128 bytes.
  - Any store by another processor to the same cache line causes the reservation to be lost.
  - Atomically updated variables should be carefully placed, because the atomic-update sequences treat the variable as a reservation cell.
  - Lock cells and atomically updated variables must be the sole occupant of a cache line. Read-only data in same line is refetched from other L2 if any datum has been modified.
- Instructions are fetched from the I-cache in aligned 8-word blocks.
  - Branch targets must be aligned on 8-word (32-byte) boundaries, where feasible. At a minimum, they must be aligned on a 4-word (16-byte) boundary, to maximize fetch and decode efficiency.
- Use **mfspr**(sprg0) as a high performance method to validate privileged mode.

7. Signal Description

This chapter describes the external signals of the 970FX microprocessor. It contains a concise description of individual signals, showing behavior when the signal is asserted and negated and when the signal is an input and an output.

Note: A bar over a signal name indicates that the signal is active low. For example, $\overline{\text{CHKSTOP\_B}}$ (checkstop in/out) and $\overline{\text{BYPASS\_B}}$ (PLL bypass). Active-low signals are referred to as asserted (active) when they are low and negated when they are high. Signals that are not active low, such as ADIN[0:43] (address bus signals) are referred to as asserted when they are high and negated when they are low.

The 970FX microprocessor's signals are grouped as follows:

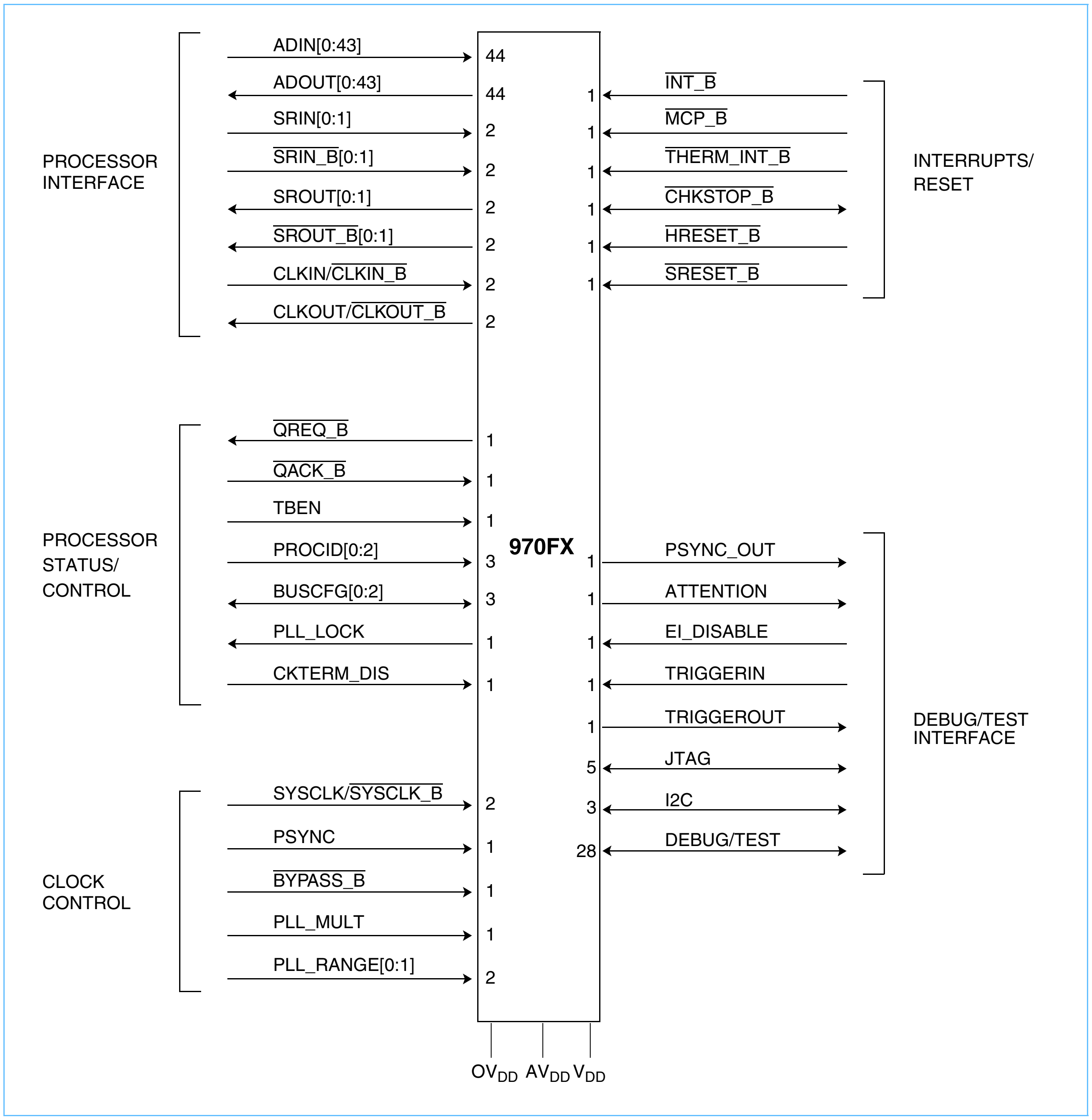
- Processor interface – These signals are used to transfer address, data, and control information between the 970FX and a companion chip to provide coherent access to memory and access to memory-mapped I/O.
- Processor status and control – These signals are used to monitor and provide external control of various processor facilities, including the external bus and power management.
- Clock control – These signals determine the system clock frequency. They can also be used to synchronize multiprocessor systems.
- Interrupts/resets – These signals include the external interrupt signal, checkstop[1] signals, and both soft reset and hard reset signals. They are used to interrupt and to reset the processor under various conditions.
- Debug/test interface – The debug/test interface provides a serial interface to the system for performing debug, bring-up, and manufacturing tests. The JTAG (IEEE 1149.1a-1993) interface and the I^2C interface provide a serial interface to the system for performing board-level boundary-scan interconnect tests.

1. Hardware has detected a condition that it cannot resolve and which prevents normal operation. It stops executing instructions, responding to interrupts, and so on.

7.1 Signal Configuration

Figure 7-1 illustrates the configuration of the 970FX microprocessor signals, showing how the signals are grouped. A pinout showing pin numbers is included in the *IBM PowerPC 970FX RISC Microprocessor Datasheet*.

Figure 7-1. 970FX Signal Groups



7.2 Signal Descriptions

This section describes individual signals on the 970FX, which are grouped as shown *Figure 7-1 970FX Signal Groups*. In the following section, “cycle” or “clock” refers to a single bus clock period, which may correspond to one or more internal processor clocks depending on the clock mode programmed for the 970FX.

Note: In PLL-bypass mode, the SYSCLK input signal clocks the internal processor directly, the PLL is disabled, and the bus mode is set for 1:1 mode operation. This mode is intended for factory use only.

7.2.1 Processor Interface

The processor interface provides a high-speed, source synchronous, point-to-point connection between the 970FX and a companion chip. It consists of two unidirectional sets of signals, one to carry outgoing information from the 970FX, the other to carry incoming information to the 970FX. Each of these two sets of signals consists of a 44-bit bus to transfer logical data with redundancy, a differential clock (2 signals), and a 2-bit differential snoop response (4 signals).

Chapter 8 provides detailed information on the format and timing of these signals as they are used in the Processor Interconnect protocol implemented in the 970FX.

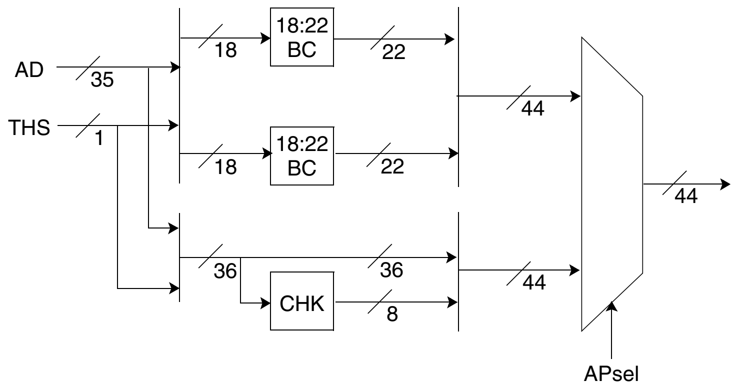
7.2.1.1 Address/Data In (ADIN[0:43])–Input

The address/data input signals carry address, data, and control information from the companion chip to the 970FX. The 44 bits of ADIN carry 36 bits of address/data (AD) and transfer handshake (TH) information plus 8 bits of redundancy.

There are two defined formats for encoding the 36 AD and THI signal lines onto the 44 source synchronous bus (SSB) signal lines, see *Figure 7-2 Encoding and Selection Logic for the Drive Side of a 970FX Processor Interconnect SSB*. The first format exploits a balanced coding method (BCM) to maintain an equal number of zeros and ones on the signal lines. During any valid state of the bus, exactly 22 of the signals lines are high and 22 are low. The BCM advantage is that it dramatically improves the signal-to-noise robustness of the bus for high-speed operation at the cost of a few extra signal lines. The BCM can inherently detect a single bit error from any of the 44 signal lines.

The second mode uses 36 of the 44 SSB signal lines for the data transfer. The remaining eight SSB signal lines are used to encode an 8-bit parity value that has sufficient redundancy to detect up to two bit errors across any of the 44 SSB signal lines and correctly identify the bit position of any single bit error.

Figure 7-2. Encoding and Selection Logic for the Drive Side of a 970FX Processor Interconnect SSB



In BCM mode, the 36 inputs are partitioned between two 18-bit balanced code (BC) encoders. In the alternate mode, the 36 bits pass straight through and 8 parity bits are added to the output. A select signal line, called APsel, is programmed through the I^2C interface to select which mode is used.

Timing: The processor interface is source synchronous, meaning that the same clock that launches data on the sending end is transferred with the data and used at the receiving end to capture the data. The interface is run in double data rate (DDR) fashion, with a data transfer on every rising and falling edge of the clock. Since there is no arbitration on this interface, valid data can be transferred on any clock edge. ADIN uses CLKIN as its reference.

7.2.1.2 Snoop Response In (SRIN[0:1], $\overline{\text{SRIN\_B}}$[0:1])–Input

The snoop response input signals carry a 2-bit code from the companion chip to the 970FX, indicating the coherency response of the system to an earlier command sent on the ADOUT bus. SRIN and $\overline{\text{SRIN\_B}}$ represent a differential pair, such that SRIN carries the snoop response in an asserted high signal level at the same time that $\overline{\text{SRIN\_B}}$ carries the same snoop response in an asserted low signal level.

Timing: Same as ADIN.

7.2.1.3 Clock In (CLKIN/$\overline{\text{CLKIN\_B}}$)–Input

The CLKIN signal originates in the companion chip and is sent synchronously with the data (ADIN and SRIN) for use in data capture at the receivers in the 970FX. This clock is transmitted as a differential pair.

Timing: The clock in signal is derived from the on-chip PLL on the companion chip and synchronized to the PSYNC signal which provides a periodic global reference event. During the initial alignment procedure (IAP) for the processor interface, a rising edge of the clock in signal is identified as corresponding to time zero. Every other rising edge thereafter is a time zero, delimiting the basic unit of time on the bus, in which four beats of data can be transferred.

7.2.1.4 Address Data Out (ADOUT[0:43])–Output

The address/data output signals carry address, data, and control information from the 970FX to the companion chip. The 44 bits of ADOUT carry 36 bits of address/data (AD) and transfer handshake (TH) information plus 8 bits of redundancy, similarly to ADIN.

Timing: Same as ADIN, except that ADOUT uses CLKOUT as its reference.

7.2.1.5 Snoop Response Out (SROUT[0:1], $\overline{\text{SROUT\_B}}$[0:1])–Output

The snoop response output signals carry a 2-bit code from the 970FX to the companion chip, indicating the coherency response of the processor to an earlier reflected command sent on the ADIN bus. SROUT and $\overline{\text{SROUT\_B}}$ represent a differential pair, such that SROUT carries the snoop response in an asserted high signal level at the same time that $\overline{\text{SROUT\_B}}$ carries the same snoop response in an asserted low signal level.

Timing: Same as ADOUT.

7.2.1.6 Clock Out (CLKOUT/$\overline{\text{CLKOUT\_B}}$)–Output

The clock out signal originates in the 970FX and is sent synchronously with the data (ADOUT and SROUT) for use in data capture at the receivers in the companion chip. This clock is transmitted as a differential pair.

Timing: The clock out signal is derived from the on-chip PLL on the 970FX and synchronized to the PSYNC signal which provides a periodic global reference event. During the initial alignment procedure (IAP) for the processor interface, a rising edge of the clock out signal is identified as corresponding to time zero. Every other rising edge thereafter is a time zero, delimiting the basic unit of time on the bus, in which four beats of data can be transferred.

7.2.2 Processor Status and Control

7.2.2.1 Quiescent Request ($\overline{\text{QREQ\_B}}$)–Input

The $\overline{\text{QREQ\_B}}$ signal, along with $\overline{\text{QACK\_B}}$, is used for power management on the 970FX. It has two distinct uses. When a power tuning frequency shift procedure is not in progress, assertion of $\overline{\text{QREQ\_B}}$ indicates that the 970FX has entered Doze mode, and is prepared to go into Nap mode. This signal remains asserted until the 970FX returns to Run mode.

When a power tuning frequency shift procedure is in progress, assertion of $\overline{\text{QREQ\_B}}$ indicates that the 970FX is prepared to perform the frequency shift itself. This signal remains asserted until the 970FX has completed the frequency shift procedure. See *Chapter 9* for more information on power tuning frequency shifting.

Timing : The $\overline{\text{QREQ\_B}}$ signal may be asserted or negated by the processor at any time.

7.2.2.2 Quiescent Acknowledge ($\overline{\text{QACK\_B}}$)–Input

The $\overline{\text{QACK\_B}}$ signal, along with $\overline{\text{QREQ\_B}}$, is used for power management on the 970FX. It has two distinct uses. When a power tuning frequency shift procedure is not in progress, assertion of $\overline{\text{QACK\_B}}$ indicates that all bus activity that requires snooping has stopped and that the 970FX may enter Nap mode. This signal must be negated whenever bus activity requiring snooping is resumed, or the 970FX negates $\overline{\text{QREQ\_B}}$.

When a power tuning frequency shift procedure is in progress, assertion of $\overline{\text{QACK\_B}}$ indicates that the rest of the system is prepared to perform the frequency shift itself. This signal remains asserted until the companion chip has completed the frequency shift procedure. See *Chapter 9* for more information on power tuning frequency shifting.

Timing: The $\overline{\text{QACK\_B}}$ signal is asserted in response to assertion of the $\overline{\text{QREQ\_B}}$ signal by the 970FX. It may be asserted any time $\overline{\text{QREQ\_B}}$ is asserted, and may be negated at any time.

7.2.2.3 Time Base Enable (TBEN)–Input

The TBEN input signal can be used in one of two ways, as determined by the value of HID0[19]. When HID0[19] = ‘0’, the time base register is incremented, and the decrementer register is decremented, at 1/8th the full processor frequency, whenever TBEN is asserted. These two timer registers maintain their value when TBEN is negated in this mode.

When HID0[19] = ‘1’, the time base register is incremented, and the decrementer register decremented, on every rising edge of the TBEN input signal. In this externally clocked mode, the TBEN frequency must not exceed 1/16th the full processor frequency in order to guarantee sufficient sampling of this external signal.

Timing: The TBEN input is asynchronous to the SYSCLK and processor clocks, and may change at any time, subject to the previously stated frequency restriction.

7.2.2.4 Processor Id (PROCID[0:2])–Input

The 3-bit processor ID input is used to assign a unique ID to this 970FX in a multiprocessor system that can have up to eight processors. The PROCID signals are sampled during power-on-reset and the 3-bit value is placed in the low order bits of the PIR register.

Timing: These signals should be permanently tied to V_{DD} or GND, as appropriate for the desired ID value.

7.2.2.5 Bus Configuration Select (BUSCFG[0:2])–Input

The 3-bit BUSCFG input encodes the processor clock to bus clock ratio. It is used to select the appropriate clock dividers in the 970FX in order to generate the desired bus clock frequency. The interpretation of the BUSCFG values can be found in the *IBM PowerPC 970FX RISC Microprocessor Datasheet*.

Timing: These signals should be permanently tied to V_{DD} or GND, as appropriate for the desired bus configuration value.

7.2.2.6 PLL Locked (PLL_LOCK)–Output

The PLL_LOCK signal is asserted when the PLL has achieved lock, or when running in bypass mode. The signal is negated otherwise.

Timing: The PLL_LOCK signal can change at any time. The initial maximum latency for the PLL to achieve lock is specified in the *IBM PowerPC 970FX RISC Microprocessor Datasheet*.

7.2.2.7 Clock Receiver Termination (CKTERM_DIS)–Input

The CKTERM_DIS signal allows the internal termination on the CLKIN and $\overline{\text{CLKIN\_B}}$ signals to be disabled. When CKTERM_DIS is negated, the clock in signals are terminated. When the CKTERM_DIS signal is asserted, the termination of the clock in signals is removed from the receiver circuit.

Timing: This signal should be permanently tied to V_{DD} or GND, as appropriate for the desired clock configuration.

7.2.3 Clock Control

7.2.3.1 System Clock (SYSCLK/SYSCLK_B)–Input

The SYSCLK inputs provide the reference clock from which the on-chip PLL develops the processor mesh clock, as well as the bus clock. The system clock is provided to the processor as a differential pair. The mesh clock frequency is determined by this reference clock and the value of the PLL_MULT input. The bus clock frequency is determined by the mesh clock frequency and the value of the BUSCFG input. See the *IBM PowerPC 970FX RISC Microprocessor Datasheet* for the correspondence between these inputs and the clock frequency ratios.

Timing: See the *IBM PowerPC 970FX RISC Microprocessor Datasheet* for clock specifications.

7.2.3.2 Phase Synchronization (PSYNC)–Input

The PSYNC signal provides a synchronization pulse to all processors and companion chips in the system, providing the basis for identifying a periodic time zero event in each chip.

Timing: See the *IBM PowerPC 970FX RISC Microprocessor Datasheet* for clock specifications.

7.2.3.3 PLL Bypass ($\overline{\text{BYPASS\_B}}$)–Input

The $\overline{\text{BYPASS\_B}}$ signal indicates to the processor that the system clock input should be fed directly to the PLL output, bypassing the PLL. This mode of clocking the processor can be used for debugging, as well as during initial power-on.

Timing: To bypass during debug, this signal should be tied to GND.

7.2.3.4 PLL Multiplier (PLL_MULT)–Input

The PLL_MULT signal is used to specify the ratio of the full processor mesh frequency to the system clock frequency. See the *IBM PowerPC 970FX RISC Microprocessor Datasheet* for the correspondence between the value of this signal and the clock ratio.

Timing: This signal should be permanently tied to V_{DD} or GND, as appropriate for the desired clock configuration.

7.2.3.5 PLL Range Select (PLL_RANGE[0:1])–Input

The PLL_RANGE signal is used to identify the desired frequency range of the processor mesh clock. See the *IBM PowerPC 970FX RISC Microprocessor Datasheet* for the correspondence between the value of this signal and the desired frequency range.

Timing: This signal should be permanently tied to V_{DD} or GND, as appropriate for the desired clock configuration.

7.2.4 Interrupts and Resets

Most system status signals are input signals that indicate when exceptions are received, when checkstop conditions have occurred, and when the 970FX must be reset.

7.2.4.1 Interrupt ($\overline{\text{INT\_B}}$)–Input

The $\overline{\text{INT\_B}}$ signal provides a means for raising an external interrupt. This exception can be masked by the MSR[EE] bit. When MSR[EE] = '0', the processor will not respond to the assertion of $\overline{\text{INT\_B}}$.

7.2.4.2 Machine Check Interrupt ($\overline{\text{MCP\_B}}$)–Input

The $\overline{\text{MCP\_B}}$ signal provides a means for raising a machine check exception. This exception can be masked by two control bits. If HID0[32] = '0', the assertion of $\overline{\text{MCP\_B}}$ is ignored. If HID0[32] = '1', and MSR[ME] = '1', machine checks are enabled, and the assertion of $\overline{\text{MCP\_B}}$ will result in a machine check exception being taken. If HID0[32] = '1', and MSR[ME] = '0', machine checks are disabled, and the assertion of $\overline{\text{MCP\_B}}$ will cause the processor to enter checkstop state.

Timing: This signal can be asserted at any time, asynchronously to the system clock. Once asserted, the $\overline{\text{MCP\_B}}$ signal must remain asserted for at least two bus clock cycles to insure that it is recognized.

7.2.4.3 Thermal Interrupt ($\overline{\text{THERM\_INT\_B}}$)–Input

The $\overline{\text{THERM\_INT\_B}}$ signal provides an external means for raising a thermal management interrupt. This exception can be masked by the MSR[EE] bit. When MSR[EE] = '0', the processor will not respond to the assertion of $\overline{\text{THERM\_INT\_B}}$.

Timing: This signal can be asserted at any time, asynchronously to the system clock.

7.2.4.4 Checkstop ($\overline{\text{CHKSTOP\_B}}$) –Bidirectional

The checkstop signal is both an input and an output signal on the 970FX.

Checkstop ($\overline{\text{CHKSTOP\_B}}$) –Input

The Checkstop input signal provides a means for external initiation of a checkstop.

Timing: This signal can be asserted at any time, asynchronously to the system clock.

Checkstop ($\overline{\text{CHKSTOP\_B}}$) –Output

The Checkstop output signal indicates that the processor has entered the checkstop state.

Timing: Might be asserted at any time.

7.2.4.5 Hard Reset ($\overline{\text{HRESET\_B}}$)–Input

The $\overline{\text{HRESET\_B}}$ signal provides a means for resetting the processor and initiating the power-on-reset sequence.

Timing: May be asserted at any time, asynchronously to the system clock.

7.2.4.6 Soft Reset ($\overline{\text{SRESET\_B}}$)–Input

The $\overline{\text{SRESET\_B}}$ signal provides a means for external initiation of the soft (or warm) reset. When this signal is asserted, the processor responds by taking a system reset exception.

Timing: May be asserted at any time, asynchronously to the system clock.

7.2.5 Debug/Test Interface

7.2.5.1 Phase Synchronization Out (PSYNC_OUT)–Output

The PSYNC_OUT signal allows monitoring of the internal psync phase during bring-up/debug.

7.2.5.2 Attention (ATTENTION)–Output

ATTENTION is an output signal from the 970FX to the JTAG debugger, used in debug mode. When Attention is active, only primitive test access port (TAP) commands will be acknowledged with the standard I^2C ACK pulse.

7.2.5.3 Procesor Interface Disable (EI_DISABLE)–Input

Turns off elasticity in the processor interface bus.

7.2.5.4 Trigger In (TRIGGERIN)–Input

TRIGGERIN is an input signal that initiates a trace collection from outside.

7.2.5.5 Trigger Out (TRIGGEROUT)–Output

TRIGGEROUT is an output signal used to indicate that an internal trace collection has begun.

7.2.5.6 JTAG Signals

The IEEE 1149.1 defines a five-wire interface called a test access port (TAP) for communicating with the boundary scan architecture. The five JTAG signals are: TDI, TDO, TMS, TCK, $\overline{\text{TRST\_B}}$.

Test Clock (TCK)–Input

TCK is a JTAG test clock which is separate from the system mesh clock. The TCK controls all test access port functions. The rising edge causes TMS and TDI to be sampled by Access.

Test Data In (TDI)–Input

TDI is a JTAG serial input used to feed test data and test access port instructions.

Test Data Out (TDO)–Output

TDO is a JTAG serial output used to extract data from the chip under test control.

Test Mode Select (TMS)–Input

TMS is a JTAG select signal used to control the operation of the JTAG state machine. The value of TMS during a rising edge of TCK causes a state transition in the TAP controller.

Test Logic Reset ($\overline{\text{TRST\_B}}$*)–Input*

Test reset ($\overline{\text{TRST\_B}}$) is an asynchronous JTAG signal used to reset the JTAG state machine. The $\overline{\text{TRST\_B}}$ signal ensures that the JTAG logic does not interfere with the normal operation of the chip, and must be asserted and deasserted coincident with the assertion of the $\overline{\text{HRESET\_B}}$ signal.

7.2.5.7 I^{2}C Signals

The 970FX I^{2}C bus conforms to the standard-mode timing specification and does not support the high-speed (Hs-mode) or fast-mode timing. The 970FX has the following I^{2}C signals:

- I^{2}C Signal Clock ($\overline{\text{I2CCK\_B}}$)–I^{2}C signal clock is both an input and output signal pin.
- I^{2}C Interface Data ($\overline{\text{I2CDT\_B}}$)–I^{2}C interface data is both an input and output signal pin.
- I^{2}C Interface Go (I2CGO)–I2CGO is an asynchronous open drain output signal used to prevent access collisions between JTAG and I^{2}C. If the level of the interface is low, only JTAG should access the 970FX. I^{2}C can make use of the interface if the level is high.

7.2.6 Voltage and Ground

The 970FX provides the following connections for power and ground:

- OV$_{DD}$—The OV$_{DD}$ signal provides the supply voltage connection for the system interface drivers.
- AV$_{DD}$—AVDD is a power signal which drives the analog sections of the PLL. See the *IBM PowerPC 970FX RISC Microprocessor Datasheet* for information on how to use this signal.
- V$_{DD}$—The V$_{DD}$ signal provides the supply voltage connection for the processor core.

8. Processor Interconnect Bus

The IBM PowerPC 970FX RISC Microprocessor Processor Interconnect is a bus architecture providing high-speed, high performance interconnections for processors, I/O devices, memory subsystems, and bridge chips. This bus architecture provides a forward-looking, general-use, yet cost-effective solution for designing high performance IBM PowerPC systems.

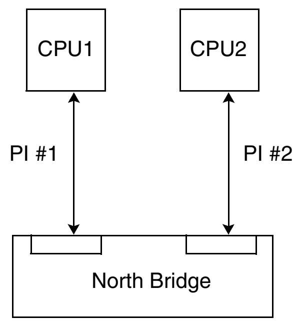
At the heart of the processor interconnect (PI) bus is a set of unidirectional, point-to-point bus segments, a new design selected to achieve maximum data transfer rates. The bus segments include two 35-bit address/data segments (one in each direction), two 1-bit transfer handshake segments, and two 2-bit snoop response segments. New features include:

- Pipelined transactions for reading and writing data and maintaining cache coherency
- Packet protocols for data sharing, data synchronization, and cache snooping
- True split transactions, enabling the master and slave to simultaneously conduct different transactions with each other
- Wave pipelining to exploit maximum data bandwidth at the electrical interface

The unidirectional segments are the basis for supporting the features above. These buses are point-to-point connections, carry their own local clock signal (source synchronous), and require no arbitration. Error detection mechanisms exist for all bus segments.

There are many possible configurations that incorporate different numbers of processors, I/O interfaces, memory bandwidth, different speed, cost and power requirements. *Figure 8-1* shows an example of a two processor configuration.

Figure 8-1. PI Two-Processor Configurations



The remainder of this section specifies the processor interconnect architecture, targeting a dual processor, dual-ported North Bridge configuration, as shown in *Figure 8-1*. Using two PI ports on the North Bridge enables direct connection of two processors.

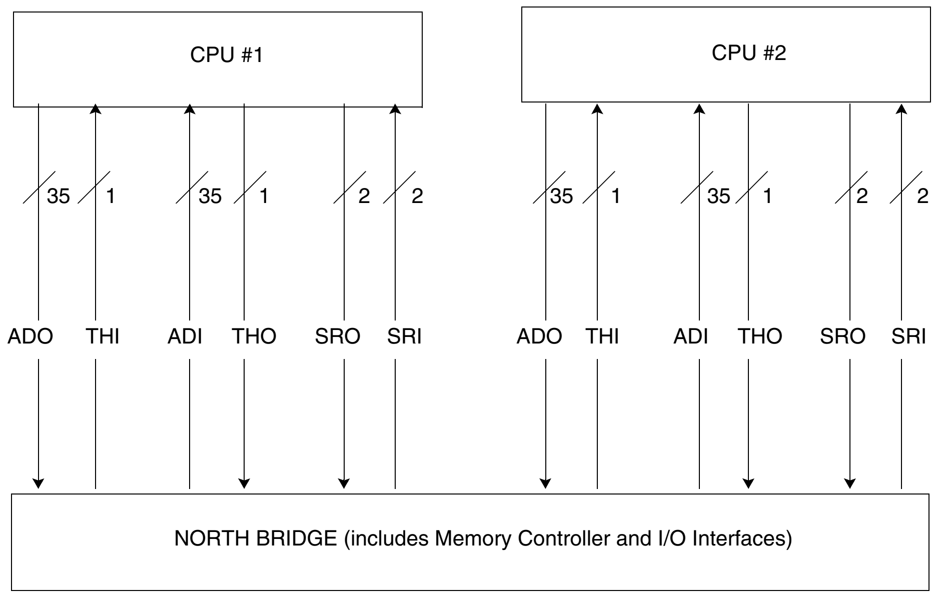
8.1 Overview

The processor interconnect bus consists of a set of unidirectional, point-to-point bus segments for maximum data transfer rates. No bus-level arbitration is required. An Address/Data (AD) bus segment, a Transfer Handshake (TH) bus segment, and a Snoop Response (SR) bus segment exist in each direction, outbound and inbound. *Figure 8-2* shows two processors connected to a North Bridge using two PI buses.

This section frequently uses the terms: *packet, beat*, *master*, and *slave*. The usage conventions of these terms are as follows:

- Data is transferred across a bus in *beats* from master to slave. A *beat* is a timing event relative to the rising or falling edge of the clock signal. Nominally there are two beats per clock cycle (one for the rising edge and one for the falling edge).
- A *packet* is the fundamental protocol data unit for the PI bus. A non-null packet consists of an even number of data elements that are sequentially transferred across a source-synchronous bus at the rate of one element per bus beat. The number of bits in each data element equals the width of the bus. Packets are used for sending commands, reading and writing data, maintaining distributed cache coherency, and transfer-protocol handshaking.
- A sender or source of packets for a bus segment is called a *master* and a receiver or recipient is called a *slave*. For example, on an outbound processor bus segment, the North Bridge is the slave and the processor is the master. On an inbound processor bus segment, the North Bridge is the master and the processor is the slave.

Figure 8-2. Logical Representation of Two Processors Connected to a North Bridge



8.1.1 Packets

Four basic packet types are defined: null packets, command packets, data packets, and transfer-handshake packets. Non-null packet lengths are always an even number of beats.

Null packets are sent across the address/data bus. For the null packet all bits are zero. Null packets are ignored by slave devices.

Command packets are sent across the address/data bus. There are three types of command packets: read-command packets, write-command packets, and coherency-control packets.

Data packets are also sent across the address/data bus. There are two types of data packets: read-data packets and write-data packets. A write-data packet immediately follows a write-command packet. A read-data packet is sent in response to a read-command packet or a cache-coherency snoop operation. A data read header contains the address of the command, the command type, and transfer details.

Transfer-handshake packets are sent across the transfer-handshake bus. This packet is issued to confirm receipt and indicate the condition of the received command packet or data packet. Condition encoding includes Acknowledgement, Retry, Parity Error, or Null/Idle. A transfer-handshake packet is two beats in length.

See *Section 8.2 Packet Transfer Protocol* for a detailed description of these four packet types.

8.1.2 Bus Segments

An AD bus segment, a TH bus segment, and a SR bus segment exist in each direction, outbound and inbound. *Table 8-1* and the following subsections further describe these signals.

Table 8-1. PI Signal Description

Signal Names	Signal Lines	Mnemonic	Description
Address/Data Out	35	ADO	Address or data and control information
Transfer-Handshake Out	1	THO	Acknowledgment packet for command and data packets received on the address/data in bus
Snoop Response Out	2	SRO	Snoop coherency response from the processor
Address/Data In	35	ADI	Address or data and control information
Transfer-Handshake In	1	THI	Acknowledgment packet for command and data packets received on the address/data out bus
Snoop Response In	2	SRI	Accumulated snoop coherency response from the North Bridge

8.1.2.1 Address/Data Bus Segment

The address/data bus is used to transfer both command packets (containing control information) and data packets (containing the data to be transferred). The address/data bus consists of one 35-bit outbound address/data (ADO) bus segment and one 35-bit inbound address/data (ADI) bus segment.

Commands are issued to the bus as 2-beat packets. A read-data packet consists of a 2-beat header followed by the data payload. The number of beats issued with a data transfer depends on the size of the total transfer. Data payload is issued to the bus in even multiples of 4-byte wide data beats. Included in the packet is a bit for special system support and a data error bit.

8.1.2.2 Transfer-Handshake Bus Segment

The transfer-handshake bus sends transfer-handshake packets which confirm command or data packets were received on the address/data bus. The transfer-handshake bus consists of one 1-bit outbound bus segment (THO) and one 1-bit inbound bus segment (THI). Every device issuing a command packet, data packet, or reflected command packet to the Address/Data bus receives a transfer-handshake packet through the transfer-handshake bus some fixed number of beats after issuing the command or data packet.

Each transfer-handshake bus segment sends transfer packets for command and data packets transferred in the opposite direction. That is, the outbound transfer-handshake bus sends acknowledgment packets for the command and data packets received on the inbound AD bus. There is no dependency or relationship between packets on the outbound address/data bus and the outbound transfer-handshake bus.

A transfer-handshake packet might result in a command packet being reissued to the bus because a data buffer in the command queue is full. IBM suggests that the North Bridge implement queues that are deep enough to minimize the impact of command packet retries on system performance.

A transaction remains active until it has passed all response windows. For write transactions, this includes the last beat of the data payload. Since commands might be retried for queue or buffer full conditions, transactions that must be ordered cannot be simultaneously in the active state.

A write transaction issued by the processor can be retried. The slave issues two transfer-handshake packets for a write transaction. The first packet is for the write-command packet and the second for the write-data packet.

For read transactions, the processor will not retry inbound (memory to processor) transfers. Reflected commands (that is, snoop requests inbound from the North Bridge to the processor) cannot be retried. This is necessary to ensure a fixed snoop window is maintained.

8.1.2.3 Snoop Response Bus Segment

The Snoop Response bus supports global snooping activities to maintain cache coherency. This bus is used by a processor to respond to a reflected command packet received on the ADI bus. The Snoop Response bus consists of one 2-bit outbound snoop response (SRO) bus segment and one 2-bit inbound snoop response (SRI) bus segment. The bus segments can detect single bit errors.

A snoop response begins when a processor receives a reflected command packet on the ADI bus. The processor provides a snoop response reporting the coherency status of the request received on the ADI bus segment. The North Bridge gathers snoop responses from all processors and sends the accumulated snoop response on the SRI bus segments concurrently to all processors.

8.1.3 Transactions

Three transaction types are defined: read, write, and command-only. *Section 8.4 Bus Transactions* describes the transactions in detail. The following subsections show the sequence of operations for these transaction types.

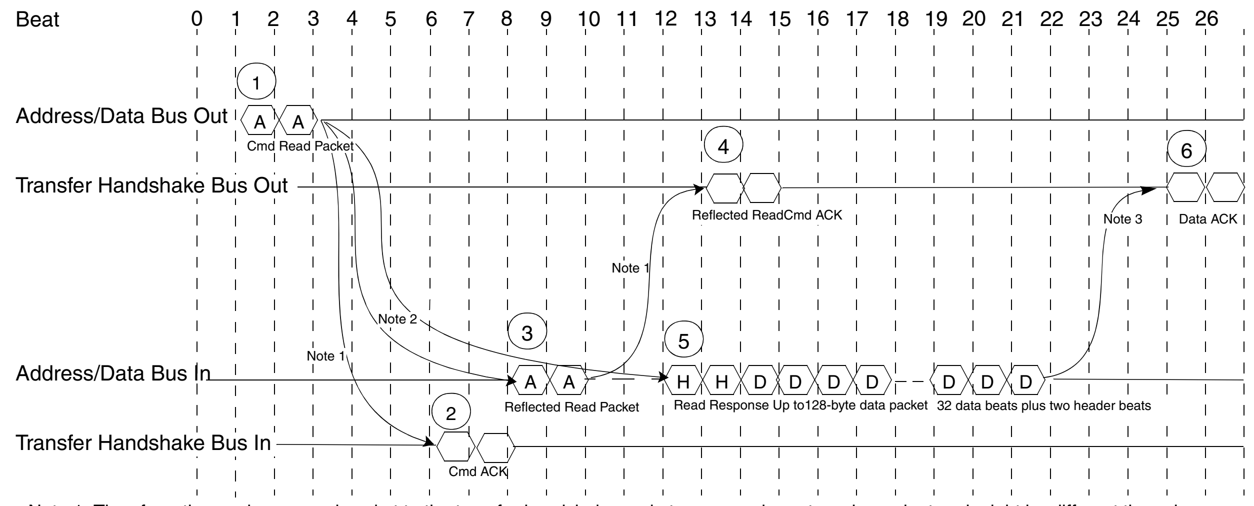
8.1.3.1 Read Transaction

Figure 8-3 shows the sequence of operations for a read transaction.

1. The master (requesting processor) issues a read-command packet on the ADO bus segment to request a full or partial cache line of data from the slave (North Bridge).
2. The slave sends a transfer-handshake packet to the master on the THI bus segment.
3. For cache-coherency purposes, the slave reflects the read-command packet on the ADI bus segment to all processors.
4. Each processor sends a transfer-handshake packet on the THO bus segment to the slave in response to the reflected read-command packet.
5. The slave sends the read-data packet on the ADI bus segment to the master.
6. The master sends a transfer-handshake packet on the THO bus segment to the North Bridge in response to the read-data packet.

The read-data packet transfer ranges from 4 to 34 beats. The first two beats transferred are a header containing the master's tag and data packet size. The data payload portion must be transmitted in sequence with the critical word first. A command packet might then be interjected into the data payload portion on an even-beat boundary.

Figure 8-3. Read Transaction Timing Diagram



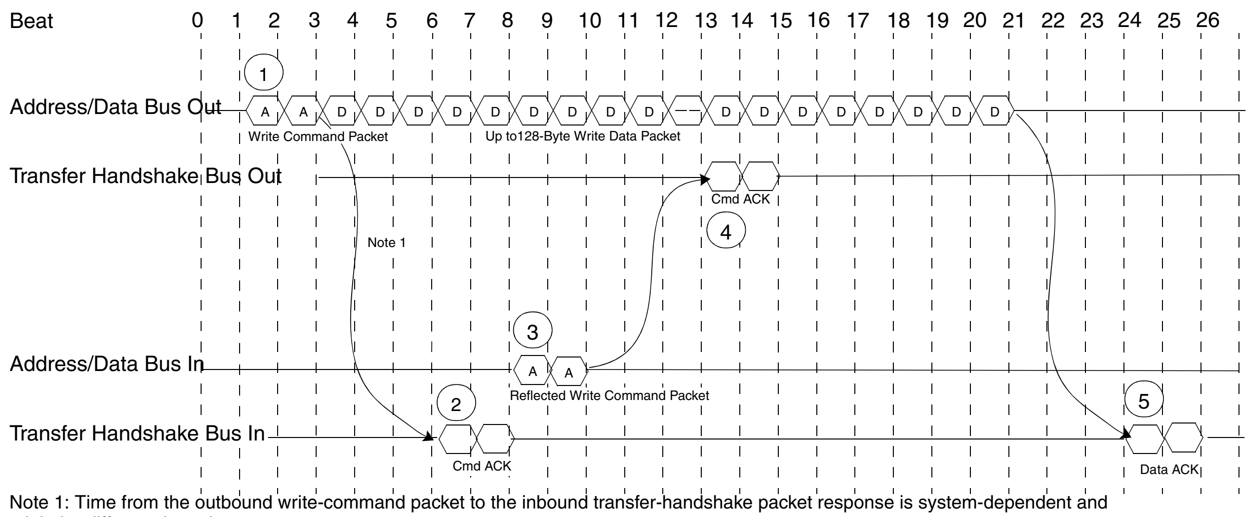
Note 1: Time from the read-command packet to the transfer-handshake packet response is system-dependent and might be different than shown.
Note 2: Time from the read-command to the reflected read request or to read-data packet response is not a fixed value.
Note 3: Time from inbound data packet to outbound transfer-handshake packet response is system-dependent and might be different than shown.

8.1.3.2 Write Transaction

A processor initiates a write transaction to store either a full or partial cache line of data to memory or to an I/O device. A write transaction consists of a command packet immediately followed by a data packet on the master's ADO bus segment. The data must be issued to the address/data bus segment in consecutive beats, but can be paused on an even beat to issue a command packet for a read operation. A write-command packet cannot be interjected into a write-data packet transfer. *Figure 8-4* shows the sequence of operations for a write transaction.

1. The master (requesting processor) issues a write-command packet on the ADO bus segment to write a full or partial cache line of data. The write-command packet is immediately followed by a write-data packet.
2. The slave (North Bridge) sends a transfer-handshake packet on the THI bus segment in response to the write-command packet.
3. For cache-coherency purposes, the slave reflects the write-command packet on the ADI bus segment to all processors.
4. Each processor sends a transfer-handshake packet on the THO bus segment to the slave in response to the reflected write-command packet.
5. The slave sends an acknowledgment packet on the THI bus segment to the master in response to the write-data packet.

Figure 8-4. Write Transaction Timing Diagram



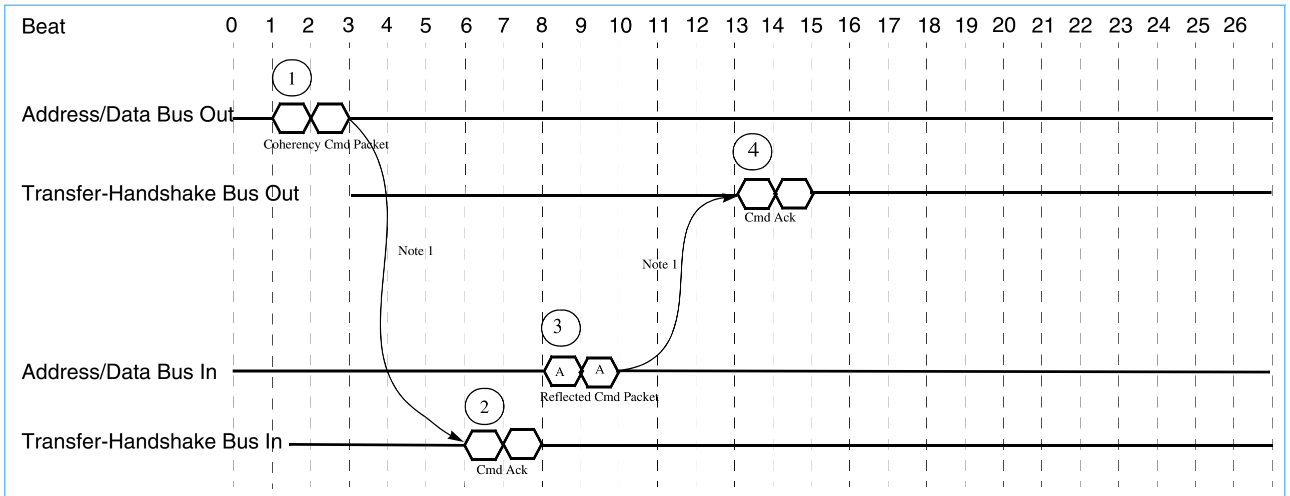
Note 1: Time from the outbound write-command packet to the inbound transfer-handshake packet response is system-dependent and might be different than shown.

8.1.3.3 Command-Only Transaction

Figure 8-5 shows the sequence of operations for a command-only transaction.

1. The master (requesting processor) issues a command packet to the slave (North Bridge) on the ADO bus segment.
2. The slave sends a transfer-handshake packet to the master on the THI bus segment in response to the command packet.
3. For cache-coherency purposes, the slave reflects the command packet on the THI bus segment to all processors.
4. Each processor sends a transfer-handshake packet on the THO bus segment in response to the reflected command packet.

Figure 8-5. Command-Only Transaction Timing Diagram



Note: Time from the coherency-command packet to inbound transfer-handshake packet is system-dependent and might be different than shown.

8.1.4 Memory and Cache Coherency

8.1.4.1 Physical Memory Size

The PowerPC Architecture supports a maximum physical address bus of 64 bits. The PI specification limits the memory addressing to 42 bits. This allows for a maximum address space of 4 terabytes (TBytes).

8.1.4.2 Coherency Protocol

Coherency is maintained using global snoops of all command packets by reflecting command packets from the North Bridge to the processor. The snoop response bus is used exclusively for this purpose. This bus consists of two unidirectional 2-bit bus segments per processor port, and is used to source response out and receive response in. Responses are sourced at a configurable time after the global snoop. The response in is sampled at a later time, also configurable. The snooping protocol is detailed in *Section 8.3*.

8.1.4.3 Coherency Block Size

The cache line is the smallest increment of memory over which coherency information is maintained. This bus can support 32-byte, 64-byte, and 128-byte coherency block sizes. The coherency block size is determined by the target processor. All bus attachments must support this coherency block size for uniform operation. The I/O must be capable of transferring less than or equal to, but not greater than, the coherency block size during DMA transfers to and from coherent memory.

8.2 Packet Transfer Protocol

This section defines packet protocols for data sharing, data synchronization, and cache snooping. The PI defines four basic packet types: null packets, command packets, data packets, and transfer-handshake packets.

8.2.1 Command Packet Definition

The command packet transfer protocol specifies how addresses are passed between bus devices. Due to the narrow width of the bus, this transfer takes two bus beats to complete, thereby allowing one command packet every two beats.

The command packet consists of a memory address, command type, command size, and command tag. The command packet is identified on the Address/Data bus by the detection of the packet start signal and a packet-type encoding for a command packet. *Table 8-2* shows the bit definitions for the Address/Data bus during a command-packet transfer.

Table 8-2. Command Packet Description

Beat	Bits	Description
1	0:1	‘10’ (Address valid decode)
1	2:6	Transfer Type (0:4)
1	7:15	Transfer Tag (0:8)
1	16:17	Address Modifiers I/S,M (1:2)
1	18:34	Address (42:58) 17 bits of the 42-bit address.
2	0:1	‘10’ (Address valid decode)
2	2	Address Modifier W/N (0)
2	3:6	Transfer Size (0:3)
2	7:26	Address (22:41) most significant 20 bits of the 42 address bits
2	27:29	Address Modifiers G,R,P (3:5)
2	30:34	Address (59:63) least significant 5 bits of the 42 address bits
Note: W: write through, M: memory coherent, N: intervention, A: atomic, R: rerunning, I: cache-inhibited, S: noncacheing coherent read, P: pipelined snoops, G: guarded read.		

8.2.1.1 Address Modifiers

Bits [16:17] of beat one, bit [2] and bits [27:29] of beat two of a command packet contain the address modifier bits. These bits further describe the type of command packet. In some cases they must be decoded along with the Transfer Type bits to determine the operation.

Table 8-3 shows when these bits are used to modify transactions, what the modification is, and what the values are when they are hard coded. Under certain conditions some bits might be sourced from the page table WIM bits (“W” stands for write through, “I” for cache inhibit, “M” for memory coherence).

Address Modifier[0]

The AM[0] bit, when indicated as a W, means that write through is desired. When the bit is '1', it means the data for a write transaction is to be forwarded all the way to system memory or a memory-mapped device. When the bit is '0', the data must be forwarded at least one cache level toward memory. This bit is normally, but not always, sourced from the page table.

On a read operation when indicated as an N, this bit indicates whether the master can support intervention on this request. If intervention is enabled (N = '1'), then the transfer size must be the coherency block size. (A snooper might not intervene if this bit is reset, and might intervene if it is asserted.)

Address Modifier[1]

The AM[1] bit, when indicated as an I, means Cache-Inhibit status. If the bit is '1' in a write-command packet, it means that the data should not be cached downstream from this processor. When indicated as an S and the bit is a '1' in a read-command packet, it means that the requesting processor will not cache the data when received, and memory (or an intervening cache) might still retain the current coherency status.

Address Modifier[2]

The AM[2] bit, when indicated as an M, is always used as the memory coherent indicator or snoop request signal. If this bit is '0', the horizontal coherency snoopers ignore this transaction, meaning memory is not coherent or this is a transaction that snoopers do not need to look at (vertical caches need to snoop all snoop response (SResp) enabled transactions regardless of the M-bit).

Note: This bit should be defined consistently for future transactions that might be architected, as snoopers will not see any transaction where M = '0'. This bit is frequently sourced from the page table WIM bits when indicated as an M, but at other times it is hard coded so snoopers see the transaction, for example, it might be set by an I/O adapter for coherent I/O.

Address Modifier[3]

The AM[3] bit is used to further define operations. For example, it is used to indicate a write-with-kill versus a write-with-clean. When indicated as a G, it is used to indicate a guarded read.

Address Modifier[4]

The AM[4] bit, when indicated as an R, means that this transaction has already been issued to the bus once, and is now being reissued.

Implementation Note – The bit should be set to zero in current implementations of the architecture, to remain compatible with potential architecture extensions.

Address Modifier[5]

The AM[5] bit when indicated as P, means that this transaction can be pipelined for snoop requests and responses. If P is '0' then command packets are reflected one at a time after the snoop response for previous command packets are seen by all processors.

8.2.1.2 Transfer Type Field

The transfer type (TType) field indicates the type of command packet that was issued to the bus. The valid transfer types defined by the PI bus are shown in *Table 8-3*. Both the processor and the North Bridge must support all commands listed in *Table 8-3*. I/O devices support only a limited subset of the commands.

Table 8-3. Transfer Type Encoding

Address Modifiers (WIMGRP)	TType Binary	Hex	Bus Operation	Code	Address Format	Data Payload	Comments
XXMXRP	00000	00	Clean	CL	Mem	N	M = ‘1’ normally
WIMXRP	00010	02	Write with Flush	WNB	Mem	Y	M = ‘1’ normally
XXMXRP	00100	04	Flush	FL	Mem	N	M = ‘1’ normally
WXM0RP	00110	06	Write with Kill	WBK	Mem	Y	W = X if from a I/O bridge
WXM1RP	00110	06	Write with Clean	WBC	Mem	Y	W = ‘1’, I = X, M = ‘0’
XXMXRP	01000	08	SYNC	SY	Tag	N	
NSMGRP	A1010	0A,1A	Read	RD	Mem	N	S = ‘1’ means RWNITC
XXMXRP	01100	0C	DKill	DK	Mem	N	M = ‘1’
NXMXRP	A1110	0E,1E	RWITM	RWITM	Mem	N	I = X, normally M = ‘1’
XXMX0P	10000	10	EIEIO	EI	Tag	N	M = ‘0’
XXMXXX	10100	14	Reserved				M = ‘0’
XXMX0P	11000	18	TLBIE	TI	Tag	N	M = ‘0’, P = ‘0’
XXMXXX	11100	1C	Reserved				M = ‘0’
XXMXRP	00001	01	LARX-Reserve	LR	Mem	N	M = ‘0’
XXMXRP	A0011	03,13	DClaim	DC	Mem	N	M = ‘1’
XXMXXX	001X1	05,07	Reserved				M = ‘0’
XXMXRP	01001	09	TLBSYNC	TS	Tag	N	M = ‘0’
XXMXXX	01X11	0B,0F	Reserved				M = ‘0’
XXMX0P	01101	0D	IKill	IK	Mem	N	M = ‘1’ normally, P = ‘0’
XXMXXX	10001	11	Reserved			A	M = ‘0’
XXMXXX	10010	12	Reserved			N	M = ‘0’
XXMXRP	10101	15	Deallocate Dir Tag	DDT	Mem	A	M = ‘0’
XXMXXX	1011X	16,17	Reserved for customers				M = ‘0’
XXMXXX	110X1	19,1B	Reserved				M = ‘0’
XXMX0P	11111	1F	Null	NUL	None	N	M = ‘0’
Note: W = write through, M = memory coherent, N = intervention, A = Atomic, R = ReRunning, I = Cache-inhibited, S = Non Caching coherent read, P= Pipelined Snoops, G = Guarded Read, X = drive ‘0’ when driving signal and don't care when receiving the signal.							

Address Field

The address field contains the address associated with the command packet. This field is defined to be 42-bits wide.

Transfer Size Field

The transfer size field indicates the size of the data packet associated with the command packet. For command packets that do not have a data packet associated with them, this field is undefined. *Table 8-4* defines the encoding for the transfer size field for the commands that require a data packet.

Table 8-4. Transfer Size Encoding

Transfer Size	Description	Number of Data Beats
0000	8 Bytes	2
0001	1 Byte	2
0010	2 Bytes	2
0011	3 Bytes	2
0100	4 Bytes	2
0101	5 Bytes	2
0110	6 Bytes	2
0111	7 Bytes	2
1000	128 Bytes	32
1001	16 Bytes	4
1010	32 Bytes	8
1011	Reserved	
1100	64 Bytes	16
1101	Reserved	
1110	Reserved	
1111	Reserved	

Transfer Tag

Command packets contain a 9-bit transfer tag used to link a command with data. This field is valid for all transactions to the bus and contains a number (generated by the processor) to identify the read-data packet on a read transaction and the write-data packet for a write transaction. Explicit tagging of command and data packets allows a bus device to have multiple concurrent outstanding transactions that require a data packet. This means that read-data packets can appear out-of-order on the bus so that transactions can complete when data is available as opposed to returning all data packets in the order the commands were issued. In addition the tag can be used to reference the response back to a command in an internal queue of a bus device. There must only be one outstanding transaction referred to by a tag at any time.

Tag Deallocation For Read Operations

Read transactions use the tag field to identify incoming read data packets that are associated with the transaction. Once a tag is assigned to a read transaction it cannot be reissued until all the read data has been received.

Tag Deallocation For Store, Castout, and Push Operations

Address/data command tags remain active until a clean global snoop response is received.

8.2.1.3 Tag Definition

Table 8-5 defines the 9-bit tag that is sent with a command or read-data packet.

Table 8-5. Tag Definition

Bits	Field	Description
0:3	Master number	Master number (one must be reserved for the North Bridge)
4:8	Master tag	Tag (one of 32) assigned to the master's requesting resource

Interjecting Command Packets

Data transfers on the bus are either write-data packets issued with a write-command packet, or read-data packets. These transfers consist of multiple data beats. When a transfer contains multiple beats of data payload transfer, a command packet might be interjected on an even-beat boundary. This feature allows new transactions to be started without having to wait for a long multi-beat data transfer to complete. This protocol allows read-command packets and coherency-control packets to be interjected. Write, castout, push, partial write operations, or other data packets cannot be interjected into a multiple-beat data transfer.

8.2.1.4 Command Pacing

It is possible for the processor to issue command packets at a rate faster than the slave can accept. The slave must then retry the packets so the commands are not lost. This is undesirable because of the additional bus bandwidth consumed for the retried commands. The North Bridge should implement queues that are sufficiently deep to minimize the impact of command packet retries on system performance. This scenario assumes the slaves can handle consecutive data packets, which requires the data buffering to be run at least at the bus clock speed. To avoid this situation, a command pipeline delay parameter, COMPACE, is defined for the bus.

The command pipeline delay parameter is a 4-bit field that is programmed into each bus master to indicate the number of bus beats of delay that must be placed between each command packet on the bus. The delay is in bus beats (assumed to be even). *Table 8-6* list the allowable range of values for COMPACE and related processor delay parameters along with North Bridge delay parameters and typical values that these parameters might take on. See *Section 11.3.2 Configurable Parameters* for additional information on these configurable delay parameters.

Note: This does not restrict the use of intervening bus beats for data packets.

Table 8-6. Programmable Delay Parameters

Parameter	Processor	North Bridge	Range in Bus Beats			Description
			Minimum	Typical	Maximum	
COMPACE	Y	N	4		14	Command pipeline delay. (See *Section 8.2.1.4*)
STATLAT	Y	N	4		24	Transfer handshake response latency. (See *Section 8.2.3* and *Section 8.4.2*)
STATLAT	N	Y		22		
SNOOPWIN	N	Y		4		Snoop window pacing. (See *Section 8.3.1* and *Section 8.4.2*)
SNOOPLAT	N	Y		25		North Bridge Snoop Latency. (See *Section 8.3.1*)
PAAMWIN	N	Y		23		
SNOOPLAT	Y	N	5		15	Processor snoop latency. (See *Section 8.3.1*)
SNOOPACC	Y	N	0		15	North Bridge snoop accumulation delay. (See *Section 8.3.1* and *Section 8.4.2.3*)

8.2.2 Data Packet Definition

The data packet transfer protocol specifies how data is passed between bus devices. A data packet is defined as an even numbered beat transfer on the address/data bus. A write-data packet immediately follows a write-command packet. It is identified on the bus by the data valid decode. A read-data packet has a 2-beat header that includes the tag and the data size. Typically, read data packets are sent from the North Bridge to a processor. However, during intervention, a processor can send a read-data packet to the North Bridge.

A data packet of minimum size consists of 8 bytes of data and the data error signal (DERR) to validate the data. Up to 16 pairs of data beats are used to transfer a cache line. *Table 8-7* shows the bit definitions for the read-data packet header on the address/data bus. *Table 8-8* shows the bit definitions for the address/data bus during a data transfer.

Table 8-7. Read-Data Packet Header Description

Beat	Bits	Description
1	0:1	‘11’ (Data and address valid decode)
1	2:6	Reserved
1	7:15	Transfer Tag (0:8)
1	16:18	Reserved
1	19:22	Responder or Intervener ID
1	23:34	Reserved
2	0:1	‘11’ (Data and address valid decode)
2	2	Reserved
2	3:6	Transfer Size (0:3)
2	7:34	Reserved

Table 8-8. Data Beat Description

Beat	Bits	Description
1	0:1	'01' (Data valid decode)
1	2:33	Next consecutive four bytes of the data packet
1	34	Data error signal (DERR) indicates an off-bus data error, full data transfer is invalid
2	0:1	'01' (Data valid decode)
2	2:33	Next consecutive four bytes of the data packet
2	34	Reserved

8.2.2.1 Two-Beat Transfers

The PI supports data transfers of varying lengths. Since the payload portion of all data packets must be at least two beats, a transfer of less than 8 bytes must be padded with additional data to fill the 8-byte minimum transfer size. The data on the bus must be address aligned so a request must be separated into two requests if an 8-byte address boundary is crossed. A master can transfer from 1 to 8 bytes of data during this operation. Data is returned in the original memory order. *Table 8-9* shows the address restrictions for transfers of 1 - 8 bytes.

Table 8-9. Two-Beat Data Transfers

Starting Address[61:63]	Byte Lanes																Data Size
	00	01	02	03	04	05	06	07									
000 – 111	X	X	X	X	X	X	X	X									1 Byte
000, 010, 100, 110	X		X		X		X										2 Byte
000	X																3 Byte
000, 100	X				X												4 Byte
000	X																8 Byte

Note:
1. 'X' is a valid starting position.
2. The operand may not cross a doubleword boundary.

8.2.2.2 Multi-Beat Transfers

The PI supports multiple-beat data transfers that are 16, 32, 64, and 128 bytes in length. All such requests for writes and reads that are less than a full coherency block (128 bytes) must be aligned to an address boundary equal to the size of the transfer. For read data transfers that are a full coherency block, data is returned with the critical 16 bytes first, followed by the remaining data in an interleaved burst order. The resulting data transfer is a block of data that is aligned to the size of the request.

Data Transfer Format for 128 Byte Transfers

On read data packet transfers that are a full coherency block, the order of the returned data words depends on the address that was specified inside the command packet. Each block of the read data packet is transferred in a sequence of 32-byte data beats. Data ordering is based on the block size. Within a word, data is always transferred in-order starting with the most significant byte and ending with the least significant byte.

Partial write commands with transfer sizes less than 8 bytes cannot cross an 8-byte boundary. All write commands (including write, castout, push, and partial write) with transfer sizes of 8 bytes or more must be aligned on an address boundary equal to the size of the transfer.

Table 8-10. Packet Ordering for 128-Byte Interleaved Packets on 32-Byte Boundaries

Address (57:59) 128-Byte	Packet Order Viewed at 16-byte Read Data Transfer	Packet Order Viewed at 32-byte Read Data Transfer
000	0 1 2 3 4 5 6 7	0 1 2 3
001 (Not Valid)	~~1 0 3 2 5 4 7 6~~	
010	2 3 0 1 6 7 4 5	1 0 32
011 (Not Valid)	~~3 2 1 0 7 6 5 4~~	
100	4 5 6 7 0 1 2 3	2 3 0 1
101 (Not Valid)	~~5 4 7 6 1 0 3 2~~	
110	6 7 4 5 2 3 0 1	3 2 1 0
111 (Not Valid)	~~7 6 5 4 3 2 1 0~~	

Interjecting Command Packets

Data transfers on the bus are either write-data packets issued with a write-command packet, or read-data packets. These transfers consist of multiple data beats. When a transfer contains multiple beats of data payload, a command packet can be interjected on an even-beat boundary. This feature allows new transactions to be started without having to wait for a long multi-beat data transfer to complete. This specification allows read-command packets and coherency-control packets to be interjected. Write, castout, push, and partial write operations, or other data packets cannot be interjected into a multiple-beat data transfer.

8.2.3 Transfer-Handshake Packets

The transfer-handshake bus is used to acknowledge command or data packets that were received on the inbound bus. Every command and data packet that is received on the inbound bus is acknowledged by a transfer-handshake packet on the associated outbound transfer-handshake bus. The transfer-handshake packet occurs a fixed number of beats later. Each transfer-handshake packet is two beats in length. *Table 8-11* shows the handshake encoding for the bus.

Table 8-11. Transfer-Handshake Definition

Response Beat 0, Beat 1	Description
0 0	Null/Idle
1 0	Acknowledge (command/data accepted)
0 1	Retry (command/data rejected, reissue command)
1 1	Parity error (parity error detected on bus)

The slave sends this acknowledge packet to the bus *n* beats after receipt of the last beat of the command or data packet, (*n* is the minimum number of beats necessary for the slave to receive the data from the bus, check the command and address, and generate the response). This time is implementation-dependent and may vary from one device to the next. The master samples the response STATLAT beats after the last beat of the command or data packet. STATLAT is the number of bus beats *between* the last beat of the command or data packet and the first beat of the acknowledge packet. For example, if the last beat of a command packet was on beat *j* and the first beat of the acknowledge packet occurred on beat *k*, then the value for STATLAT would be *k-j-1 (see Figure 11-11)*. The STATLAT beat count includes the time required by the slave to generate the response plus the time that it takes for the packet to be sent and the acknowledge to be returned. For consistency in design of the processors that attach to this bus, an upper limit is defined for the time between the master issuing the last beat of the command or data packet to the bus to when it receives the first beat of the acknowledge packet (see *Table 8-6* and the *IBM PowerPC 970FX RISC Microprocessor Datasheet*). This time should be minimized to eliminate unnecessary delays on commands in the pipeline that have ordering requirements with the current command. The STATLAT parameter is configured by the I^2C interface during the bus initialization phase.

8.2.3.1 Null Transfer Handshake

The null transfer handshake is the default response from a slave device. If the slave does not drive the transfer handshake with either an acknowledge, retry, or parity error, then the response is, by default, null. The null response can occur due to a slave device timeout or a terminated transaction under certain, special conditions defined below.

Master:
For the master this transfer-handshake response from the slave indicates that the command or data packet that the master sent was not accepted by the slave. Based on the address/data packet type, the master actions are as follows:

> **Command packet:** The processor responds by going into checkstop.[1] If the command was a write command and the master detects this response before it has completed the full data transfer of the write-data packet, it can either complete the full data transfer or discard the remaining even-numbered data beats for the transfer.
>
> **Read-data packet:** The processor responds by going into checkstop. A master always transmits full packets on the bus. The handshake is received after the end of the read-data packet (see *Figure 8-3*). Note the error might also result from a time-out waiting for data or an incorrect transfer size by the slave.
>
> **Write-data packet:** The processor responds by going into checkstop, if the write-command packet associated with this packet had received an acknowledge transfer handshake from the slave. Otherwise the null response is ignored. If the write command is retried by the slave, then the null response is the correct response for the data packet associated with that write command.

Slave:
The slave issues the null transfer handshake response for the non-error condition: Data packet for a write command that was retried. The command or data packet is discarded and status is logged in the slave for the error case.

1. Hardware has detected a condition that it cannot resolve, and which prevents normal operation. It stops executing instructions, responding to interrupts, and so on.

8.2.3.2 Transfer Handshake Acknowledgment

The acknowledge response indicates that the addressed slave accepted this command or data packet. If a bus agent[1] accepts (acknowledges) a command packet to send to a remote bus, it is responsible for completing the transaction back to the bus master for the case where the remote bus does not accept the command packet. For a read transaction this implies returning data to the master with the data error signal activated. The data error is signalled by asserting the 35th bit (DERR signal) of the even data beats. For writes the command and data packets are discarded. The device must also have a mechanism to signal a machine check that the error occurred.

Master:
For the master, the acknowledge response indicates that the command or data packet was accepted and that it might complete execution of the packet transfer. Based on the packet type, the master acts as follows:

> **Command packet:** For a command packet that results in data being returned by the slave, the acknowledge response indicates that the command has been accepted and need not be reissued to the bus. Inbound data packets to complete the transaction may be received starting in the beat following the response. For a write-data packet, the acknowledge response indicates that the command has been accepted. The slave may not retry the data packet after it accepts the command. That is an acknowledge response for the command packet indicates that the slave has set aside buffer space for the write data packet. For command packets, this response indicates that the command is complete.
>
> **Data packets:** This response indicates to the master that the data being sent was accepted by the slave without errors.

Slave:
The slave issues this acknowledge response when:

- The slave received the command packet with a valid transfer type, transfer size, and address.
- For write transactions, there is queue space for the command and data.

The slave stores command packets in a command queue and stores data packets in data buffers.

8.2.3.3 Transfer Handshake Retry

A handshake retry may be issued to flow control the command packets for cases where the slave does not have space for the command packet or the data packet associated with the command. Any command packet may be retried by the slave, except for reflected command packets. Data packets may not be retried.

Master:
For the master the retry response indicates that the command was rejected by the slave for lack of space in the command queue or the data buffers. Based on the packet type the master acts as follows:

> **Command packet**: When the master receives a retry response for a command packet it reissues the packet to the AD bus. If the command was a write command and the master detects this response before it has completed the full data transfer, then it can either complete the full data transfer, or discard the remaining even-numbered data beats for the transfer before reissuing the command packet.
>
> **Data packet**: Retry responses are not valid for write data packets and read data packets.

1. Bus agents are devices such as the North Bridge, but not switches that might be used to relay command and data packets.

Slave:
The slave issues this response when:

- The slave received the command packet but the command queue was full.
- If the packet was a write command packet, there is no space for the command or the data.

To properly detect termination of a partial write-data packet, the slave must examine the Address Valid decode bits (see *Table 8-2 Command Packet Description*) on a per even-beat basis.

Note: The retry transfer handshake cannot be issued for write data packets.

8.2.3.4 Transfer Handshake Parity Error

This response is optionally issued whenever a single bit error is detected during any bus beat. It is an unrecoverable error that results in a machine check to the processor with all command and data packets in the pipeline being discarded.

Master:
For the master, the response is a hard error indicating that the bus is no longer functional. The processor responds by going into checkstop.

Command packet: When the master receives a parity error response for a command packet, it reports the failure back to the system. The bus must be reinitialized before it can be used again.

Data packet: Same as command packet errors.

Slave:
If the slave issues this response (optional), it should be within the normal packet response timings. (This packet error may make this timing determination imprecise.) For the slave, this condition is a hard error and the bus is no longer functional. The slave logs the error and reports it to the system. The error reporting mechanism is system-dependent.

8.3 Snoop Responses

Cache coherency is maintained using a global snoop method, where a memory controller device (the North Bridge) reflects command packets to all processors at the same time. Snooping is supported by dedicated snoop response bus segments, consisting of one 2-bit SRO and one 2-bit SRI.

A snoop response begins when a processor receives a reflected command packet on the ADI bus. The processor starts a programmable timing chain that determines when the processor's SRO is driven and when the processor's SRI will be sampled.

The snoop response from each processor is transmitted on the SRO response bus in two beats (see *Table 8-12*). The North Bridge gathers the snoop responses from all processors and performs a logical-OR operation on the accumulated responses. The North Bridge sends the logical-OR of the snoop response back to all processors on the SRI bus.

Table 8-12. Snoop Response Bit Definition

Beat	Bits	Description
1	SR[0]	Intervention
1	SR[1]	Modified
2	SR[0]	Retry
2	SR[1]	Shared

Table 8-13. Allowed Snoop Responses

Retry	Intervention	Modified	Shared	Description
0	0	0	0	Null (exclusive for reads)
0	0	0	1	Shared
0	0	1	x	Modified
0	1	0	0	Invalid
0	1	0	1	Shared intervention
0	1	1	x	Modified intervention
1	x	x	x	Retry

8.3.1 Snoop Response Bus Implementation

Each snoop response bus is controlled by two configurable parameters: SNOOPLAT and SNOOPACC. For all parameters, time is measured in bus beats from the final locally clocked flip-flop or latch output to the first locally clocked input.

The processor SNOOPLAT parameter defines the number of bus beats between receiving the last beat of the reflected command packet and driving the first beat of the snoop response. SNOOPLAT does not need to be programmed for the processors, since the processors are assumed to be identical. The North Bridge SNOOPLAT value is the sum of the transfer time of the reflected command packet from the North Bridge to the processor, the processor SNOOPLAT value, and the transfer time of the snoop response bus from the processor to the North Bridge (see *Figure 11-12 North Bridge Configurable Timing Parameters* on page 287).

On the North Bridge, the SNOOPACC parameter defines the delay between the time a processor sends the last beat of an individual snoop response to the time it receives the first beat of the accumulated snoop response from the North Bridge (see *Figure 11-13 Processor Configurable Timing Parameters* on page 287). SNOOPACC includes the time required by the North Bridge to gather the responses from all of the processors. The North Bridge reflects all incoming command packets at a pace determined by the SNOOPWIN parameter. SNOOPWIN sets the snoop window, which is the minimum distance between two consecutive snoop requests (see *Figure 11-13*).

An address collision occurs if the current address is the same as a previously received snoop requested address. If this occurs the current snoop request is delayed until the conflicting previous request is concluded. This condition is called Previous Adjacent Address Match (PAAM). The PAAMWIN parameter indicates the number of bus beats a request is active during which a conflicting snoop request cannot be issued. An unrelated snoop request can be sent during the PAAM window. *Figure 11-12. North Bridge Configurable Timing Parameters* shows the timing of the PAAMWIN parameter.

For a snoop request to be issued, the following conditions must be satisfied:

1. At least SNOOPWIN beats have transpired since the previous snoop request was issued.
2. There is at least one non-active PAAM address slot available.
3. No active PAAM address conflicts with the request.

The number of PAAM address slots on the North Bridge is implementation-dependent, but ranges from 2 to 4. A snoop request activates a PAAM address slot when it is issued. After PAAMWIN beats the slot is deactivated and can be reassigned to another request. The number of address bits used to detect conflict is also implementation-dependent.

There is no requirement that all snoop requests fall in exact modulo SNOOPWIN beats. Even-numbered idle bus beats can be used beyond SNOOPWIN between two subsequent snoop requests. The PAAMWIN value is not required to be a multiple of the SNOOPWIN value.

The I^{2}C interface is used to program all programmable delay parameters (see *Section 11.2 I^{2}C Interface* on page 278).

8.3.2 Snoop Response Descriptions

8.3.2.1 SResp Retry Response Code (Priority 1 - highest)

SResp Retry is issued for the following reasons:

- Lost reservation*:* A master that has a reservation will retry an atomic write/flush itself if the reservation has been lost since the write was issued.
- Push condition: A snooper will retry a transaction if a push is needed for a read or write-with-flush.
- Resource conflict: A snooper will retry a transaction due to collision with a resource that has ownership of the line.
- Memory and intervention buffer full: A North Bridge can retry a read transaction that might cause intervention, if it determines it temporarily cannot receive the intervention data. It is typically more efficient to use the transfer handshake retry on the intervening data packet for this case.

SResp Retry ramifications:

- Master: May reissue this operation use a different tag, or may reissue a different operation instead of or before the operation is reissued. Any data transfer aborted by this retry may be terminated prior to the data packet completion.
- Target: Any operation that has completed a SResp Retry may take a variable amount of time to clean up resources and therefore may cause future retries due to resources being tied up by this operation. Guarded cache-inhibited write operations need to be ordered with respect to each other. The processor cannot proceed and cannot issue the next operation until the SResp window with the null response has passed.
- Snooper: Any operation that has completed a SResp Retry is aborted by the snooper and leaves the cache state unmodified, except when Intervention is disabled on a read request and the snooper has modified data. The snooper will then push the data back to memory and clean or invalidate the line.

8.3.2.2 SResp Modified - Intervention Response Code (Priority 2)

The Modified coding is activated when a snooper detects the address of a cache line on a read operation that is contained in its own cache and is modified (dirty). The snooper then provides the data by using intervention.

SResp Modified-Intervention is asserted if a snooper asserts SResp Modified-Intervention on a Read or RWITM when bus intervention is enabled (N = '1'), snooping is enabled (M = '1') and a cache line is snooped modified. If SResp Retry is sampled instead of SResp Modified-Intervention, then the snooper can either push the block to memory or leave the cache state unmodified.

The ramifications of a SResp Modified - Intervention for bus devices are:

- Master and Read or RWITM:
  This tells the master that its request is satisfied by the cache holding the modified data.
- Memory and Read or RWITM:
  This tells the North Bridge to cancel its read request. If the command was read, the North Bridge looks for the tagged data and copies the block to memory.

8.3.2.3 SResp Shared - Intervention Response Code (Priority 3)

The Shared-Intervention coding is activated when a snooper detects the address of a cache line on a reflected read command packet that is contained in the snooper's own cache and is the owner (most recent recipient) of the data. This signal can only be asserted by one bus device, since there is only one owner of data. Since SResp Retry is a higher priority than SResp Shared, the snooper must wait until the snoop response is received before beginning the intervention push.

A snooper using this code must accommodate the option on burst reads whereby the requestor indicates intervention is not desired. In these cases, the response must be SResp Shared.

The ramifications of a SResp Shared - Intervention are:

- A master receiving this SResp code looks for intervention data.
- The North Bridge treats SResp Shared - Intervention as SRespRetry.

8.3.2.4 SResp Modified Response Code (Priority 4)

SResp Modified is asserted for the following reasons:

- A snooper asserts SResp Modified on a Read or RWITM when bus intervention is not enabled (N = '0'), snooping is enabled (M = '1') and a cache line is snooped modified. If SResp Retry is sampled instead of SResp Modified, then the snooper can either push the block to memory or leave the cache state unmodified.
- A snooper asserts SResp Modified for the flush or clean bus operations if the addressed block is modified. If SResp Modified is sampled in this case, then the snooper pushes the block to memory and marks the cache Invalid (flush), or Shared/Exclusive (clean). If SResp Retry is sampled instead of SResp Modified then the snooper can either push the block to memory or leave the cache state unchanged.

8.3.2.5 SResp Shared Response Code (Priority 5)

Snooper:
The Shared response is encoded when a snooper inspects the address of a cache line on a read transaction that is contained in its own cache and has not been modified, marking the block shared if the block was marked exclusive. This signal can be asserted by more than one snooper, and the snooper will retain a copy of the block.

I/O Snooper:
I/O devices that do not cache data Exclusive or Modified (shared only) are allowed to assert without having the block cached (for example, they might snoop at a larger granularity than the block address).

Master:
This tells the bus master that the data on a read, when returned, must be marked shared and not exclusive.

8.3.2.6 SResp Null or Clean Response Code (Priority 6 - lowest)

The null or clean response is encoded to indicate one of the following:

1. There is no local (or remote) device presently caching this line.
2. A synchronize type transaction has been completed by all snoopers (for example, **sync**, **tlbsync**).
3. The line is cached, but the null response is allowed (for example, the null for a clean transaction that hits on an exclusive line).

8.4 Bus Transactions

This section provides details of the following processor interconnect bus transactions:

- Memory read transactions (general)
- Memory write transactions (general)
- Command only transactions

8.4.1 Terms

Each of the transactions in this section uses the following terms to define the parameters of the transaction.

Reservation	A reservation is an address location held by the processor. It is used for emulating atomic operations using the PowerPC **larx** and **stcx** type instructions. A processor has at most one reservation at any time. A reservation is established by executing a **lwarx** or **ldarx** instruction. It is normally lost when the corresponding **stwcx.** or **stdcx.** instruction is performed. A reservation might also be lost if the data at the address is modified by another processor or bus device.
Snooper	A bus device that inspects inbound reflected command packets and uses the Snoop Response bus to keep cached data coherent with other system caches. A bus adapter or I/O bridge might contain a cache and if so will act like a snooper.
Memory	The bus device that responds to a memory read or write, and handles positive acknowledgment for coherent operations. If some portion of memory is attached to a remote bus then the bus adapter also acts like memory for memory accesses to that remote memory space.
I/O Bridge	An I/O bridge device is a gateway to an I/O bus that cannot cache data in the Exclusive or Modified state. The bridge does not forward snoops to the I/O bus. If an I/O device has shared cache data, it is necessary to implement a directory for the cached data in the shared state.

8.4.2 Memory Read Transactions (General)

A master (processor) reads I/O or memory data by sending a read command to the memory controller of the North Bridge. The processor drives the ADO bus provided it was not in the midst of sending another command packet, and there was no higher priority transaction ready to be sent. After a programmable number of beats (STATLAT), the master reads the transfer handshake from the THI bus to ascertain the status of the transfer. The slave (North Bridge) sends a positive acknowledge on the THI bus if no parity error was detected and there was a slot to queue the read request. If no queuing space is available, a retry status is returned.

The North Bridge dequeues the request after internal arbitration and decodes the command packet. It issues a read request to the North Bridge for the indicated block size and reflects the command packet to all processors for snooping purposes. The North Bridge paces new snoop requests based on the programmable parameter, SNOOPWIN. The North Bridge will detect address collisions (transactions to the same cache line) and will delay the second conflicting transaction until PAAMWIN bus beats have transpired since the original conflicting transaction was issued for snooping. In addition, processors can request that transactions be handled one at a time, by setting the pipelined snoop (P) address modifier bit low.

Each processor drives their outbound SRO bus during the snoop window that is seen by all processors and the North Bridge at the same time. The processor may request that the transaction be retried with a retry snoop response. Otherwise, if a processor has a clean copy in its cache, the shared response code is returned. If the requested cache line is modified inside a processor cache, that processor signals the intervention snoop response, which is a promise to send to the North Bridge the modified copy in the form of a processor-to-memory read-data packet. The North Bridge accumulates the combined (logical-OR) snoop responses from all of the attached processors. Depending upon the combined response the North Bridge may abort, delay, or send the memory data or the intervened data to the original requester. The intervened data is also written to memory for regular read transactions (no intention to modify).

When the North Bridge responds with the read data it sends a read-data packet, which consists of a 2-beat header and 2 to 32 beats of payload data. The header contains the original tag and the data size. The payload data is sent immediately after the header. The DERR bit is asserted if the data contains an error.

8.4.2.1 Read Transaction

A read command is issued to get data that is not immediately going to be modified. The modifier bits that are valid are N (intervention) and S (non-caching). The M and I modifiers are sourced from the page table entry, hardwired, or set by the I/O.

Master:
A read burst is issued by the processor to satisfy a load, tablewalk access, **dcbt** or other data prefetch, or I-fetch to a cacheable page that misses the cache. A read non-burst is caused by a non-cacheable load or I-fetch.

Atomic:
The Atomic modifier (TType <0>) is set along with the M-bit when the read is to satisfy a **lwarx** or **ldarx**.

S-Bit:
The S-bit is set along with the M-bit when the master will not cache the data but wants the latest copy. If S is set, a snooper is allowed to clean up dirty data in its cache by pushing it to memory, but keeping it marked exclusive afterwards.

N-Bit:
The N-bit is set when the master and memory are capable of intervention, intervention is desired, and the read data size is the coherency block size for the system. All non-block size reads must have the N-bit set to zero. In addition, the processor is capable of setting N = '0' for all reads in case it is attached to memory that does not support intervention.

G-Bit:
The G-bit is set when the read is the result of a load to cache-inhibited guarded storage. When set, the system implementation knows this read might only complete once.

P-Bit:
The P-bit is set when snoop pipelining is allowed (default for reads). This bit can be cleared for reads if the processor requires the transaction snoop response be resolved before another independent transaction is issued. When an address collision is detected, the North Bridge automatically delays the colliding transaction until the previous transaction is resolved.

Snooper:
If the address contained in the reflected command packet is in the cache and marked Modified, the snooper performs a push or intervention.

Memory:
Memory can provide the addressed data no earlier than the end of the snoop window for that transaction. The North Bridge examines the Snoop Response bus and if it was SResp Retry or SResp Intervention, the North Bridge will terminate the operation and deallocate the tag. If the SResp response is Modified and because the North Bridge supports intervention, the North Bridge captures the line as it is transferred to the requestor and stores the line to memory.

I/O Bridge:
If the G-bit is set, an I/O bridge cannot issue the read to any memory mapped I/O devices more than once. This means waiting until the previous guarded read is committed (no retry from the transfer handshake) before sending the next request.

8.4.2.2 Read with No Intent to Cache Transaction

Read with no Intent to Cache (RWNITC) is another name for a read transaction with the S-bit and M-bit set (see above). It is a coherent read; that is, the master wants the latest data, but does not cache it, therefore the snooper can keep caching the data as Exclusive after it provides the data through a push or an intervention.

8.4.2.3 Read with Intent to Modify Burst Transaction

Master:
The read with intent to modify (RWITM) transaction is issued by a master to bring an entire block into a cache for the purpose of writing to it. It is always a block sized read. It is triggered by a store, **stwcx.**, **stdcx.**, or **dcbtst** to a cacheable page that misses in the cache. The master should mark its cache Exclusive if the SResp is not Retry.

Snooper:
The snooper invalidates any line cached at the same physical block address and asserts SResp Null if marked Invalid, Shared, or Exclusive. If the request hits its cache, and it is marked Modified, the snooper performs either a push or an intervention. If the system supports Shared-Intervention and the request was marked N = '1', then the snooper can respond Shared-Intervention and push the data.

Memory:
The memory can provide the addressed data no earlier than after SNOOPACC. The North Bridge must examine the snoop response code, and if it was Retry or Intervention, the North Bridge should terminate the operation and deallocate the tag.

Atomic:
The Atomic modifier (TType <0>) is set along with the M-bit when the read is to satisfy a cacheable copy-back **stwcx.** or **stdcx.**.The master SResp retries its own RWITM-A if the reservation is subsequently cleared after issuing the RWITM but before SResp and does not reissue the RWITM-A. If a processor does not support any cache levels below it (for example, it sees all the system coherency traffic, then the A-bit need not be set on RWITM).

N-Bit:
The N-bit is set when the master and memory are capable of intervention, intervention is desired, and the read data size is the coherency block size for the system. All non-block size reads must have the N-bit set to zero. In addition, the processor is capable of setting N = '0' for all reads, in case the memory does not support intervention.

G and S-Bits:
These bits are not defined for RWITM.

8.4.2.4 LARX-Reserve Transaction

Master:
The LARX-Reserve transaction is an address-only transaction that sets the reservation for every cache level below the level serviced by a read atomic operation. If the reservation at one level is already set to the same address as the LARX or a LARX-Reserve being propagated, then it should not be propagated further because this causes a bus operation each time the LARX is executed and might be part of a program loop.

Snooper:
Does not see the LARX-Reserve for M = '0'.

Memory:
Ignores this operation.

8.4.3 Memory Write Transactions (General)

A master sends a write command to write data to memory or to an I/O device. The write-command packet is immediately followed by a write-data packet. The slave (North Bridge) checks the command to see if there is buffer space to store the write-data packet. The slave responds with a retry transfer-handshake packet if there is insufficient buffer space. The master can then terminate sending the write-data packet on an even beat. It may then try again to send the write-command packet and write-data packet at a later time.

The North Bridge reflects every command packet to all processors. The snoopers ignore the reflected command packet if M = '0'. Only the original processor needs to see the address inside the command packet to deallocate the tag after the transaction is completed. At that time, the North Bridge takes responsibility of snooping for the pushed (castout) data. The transaction must be propagated all the way to memory if the W-bit is asserted.

8.4.3.1 Write-With-Kill Transaction

Master:
The write-with-kill transaction is a burst operation used to tell all snoopers to invalidate any copies of this line in their caches, while also storing the line to memory.

Table 8-14. Write-With-Kill Types Supported

WIM Bits for Write-With-Kill	W-Bit	M-Bit
Copyback due to load, store, or **dcbz**	0	0
I/O Write[1]	0 or 1	1
Flush due to **dcbf**	1	0
Push due to snoop	1	0

1. An I/O write is a full cache line write from a memory address.

Snooper:
If M = '0' the snooper ignores this operation. If M = '1' the snooper treats this operation as a Data Line Kill (DKill) to the same address block, marking it Invalid (this includes any store buffers) and the operation is passed to any higher level cache.

Memory:
Memory must not update storage if the transfer-handshake packet indicates Retry or the SResp value (if applicable) is Retry.

8.4.3.2 Write-With-Clean Transaction

Master:
A write-with-clean transaction is a burst operation caused by a processor executing a ***dcbst*** instruction or a bus snoop read or clean to a modified block. It is used to tell all lower level caches that a copy still remains in this level, while updating memory (or I/O). Since no snooper has the line, it sets M = '0' so horizontal snooping is avoided. The block is written all the way to memory.

Snooper:
Snoopers should not see this operation since M = '0'.

Memory:
Memory must not update storage if the SResp is Retry.

8.4.3.3 Write-With-Flush Transaction

Master:
A write-with-flush transaction is a partial-block write to memory and can be a sub-block burst operation from the I/O. It is used for cache-inhibited, or write through writes from a processor (sub-block writes). The processor sources the M-bit from the page table entry. I/O masters can also use this for DMA writes to a cache block without getting ownership first. The processor will set M = '1' for this transaction.

Snooper:
If M = '1' and the line is cached Modified, this operation is SResp Retried. The line is pushed back to memory with a write-with-kill, then invalidated (this includes any store back buffers [SBBs]). The only appropriate SResp response is Retry by a snooper (other than SResp Null, which is the default response).

Memory:
Memory must not update storage if the transfer-handshake indicates retry or the SResp value (if applicable) is Retry. A bus agent cannot pass a write-with-flush to an I/O bus that might contain memory mapped devices or memory that can be reserved without first successfully passing the response window on the PI bus.

8.4.4 Command-Only Transactions

8.4.4.1 DClaim Transaction (Invalidate others)

Master:
A master issues a Data Line Claim (DClaim) to service a **dcbtst**, **dcbz**, or store instruction. The DClaim is used to attempt to take a coherent block from the shared (or with **dcbz** invalid) state to the modified state and all other horizontal caches to the invalid state. It differs from DKill in that the DClaim does not invalidate the master's copy in a lower level (higher number) cache.

Snooper:
Snoopers must invalidate their data cache blocks if there is a hit on this address.

Memory:
Ignores this operation.

8.4.4.2 Flush Transaction

Master:
A flush transaction is caused by a **dcbf** that hits on a memory coherent cache block and is marked shared or invalid. It is sent to other snoopers that might have a copy of the line.

Snooper:
If M = '1' snoopers snoop their caches, and if the line is cached, it is marked invalid. If the line was marked as modified, it is pushed back to memory. A snooper might respond modified, or might respond Null. A SResp Retry response should only be used if the command cannot be accepted or a pipeline address collision occurs.

Memory:
Memory might ignore this operation, even if a snooper responds SResp Modified, since intervention is not supported on the flush operation itself. The flushed data is sent to memory on a separate write-with-kill operation.

8.4.4.3 Clean Transaction

Master:
A clean transaction is caused by a **dcbst** that hits on a memory coherent cache block and is marked shared or invalid. It is sent to other snoopers that might have a modified copy of the line.

Snooper:
If the line is cached, then it is marked shared (or exclusive if it is the lowest cache in the hierarchy). If it was marked modified, the line is pushed back to memory. A snooper might respond modified, or might respond Null. A SResp Retry response should only be used if the command cannot be accepted or a pipeline address collision occurs.

Memory:
Memory might ignore this operation, even if a snooper responds SResp Modified, since intervention is not supported on the clean operation itself. The cleaned data is sent to memory on a separate write-with-clean operation.

8.4.4.4 IKill Transaction

Master:
The intent of the Instruction Line Kill (IKill) transaction is to invalidate entries in any instruction-only caches in the system. Data only or combined caches are invalidated with other coherency operations. An IKill block is caused by an **icbi** instruction that hits on an instruction cache block that is marked as memory coherent. In order to prevent bus livelocks, this command should be issued with the P-bit set to 0.

Snooper:
Snoopers must invalidate their instruction cache blocks if there is a hit on this address. Any unified data or data only cache does not need to be snooped. A snooper might respond Retry, or might respond Null. A SResp Retry response should only be used if the command cannot be accepted due to resource conflicts.

Memory:
Memory can ignore this operation.

8.4.4.5 TLBIE Transaction

Master:
TLBIE is caused by a processor executing a **tlbie** instruction.

Snooper:
Snoopers accept this transaction regardless of the M-bit and invalidate any TLBs in the congruence class.

Memory:
Memory might ignore this operation.

8.4.4.6 TLBSYNC Transaction

The intent of the TLBSYNC transaction is to act as a barrier that forces all previous operations using invalidated TLBs to complete before the TLBSYNC completes.

Master:
The master issues the TLBSYNC transaction in response to a processor **tblsync** instruction.

Snooper:
Snoopers must SResp Retry the TLBSYNC until all previous loads or stores and I-fetches that used any TLBs have been flushed or performed and any snooped TLBIEs are completed. A snooped TLBSYNC has the same effect on a processor that a **sync** would have if it were executed on that processor.

Memory:
Memory can ignore this operation.

8.4.4.7 SYNC Transaction

Master
A master issues a SYNC transaction when a processor executes a **sync** instruction. The master stops processing all future instructions until all previous instructions have been completed. Then the SYNC transaction is issued to the bus, and the **sync** instruction is not completed until the SYNC transaction completes on the bus. SResp Retry will cause the operation to be repeated, SResp Null signals completion. To prevent bus livelocks, this command should be issued with the P-bit set to '0', if the snooper implementation would cause resource conflict retries.

Snooper:
A snooper drives SResp Retry if there are any snoop operations pending, or cache pushes or snoop operations pending from previously snooped bus operations. Otherwise it responds SResp Null.

Memory:
Memory signals SResp Retry until stores are performed if they can be reordered within the memory.Otherwise responds SResp Null. Memory can also respond SResp Null. The SYNC is used as a store barrier.

8.4.4.8 EIEIO Transaction

Master:
The intent of the EIEIO transaction is to act as a barrier for all non-cacheable loads or stores that follow it. It forces all previous non-cacheable operations to complete before any non-cacheable operation issued after the EIEIO. EIEIO is caused by a processor executing an **eieio** instruction.

Snooper or Memory:
Ignore this operation.

I/O Bridge or Bus Adapter:
Accept and propagate toward memory-mapped I/O storage and do not allow any cache-inhibited storage access to bypass (if they can be reordered).

8.4.4.9 Null Transaction

Master:
A Null transaction is used by the processor to break cyclic deadlocks or prescheduled transactions that are no longer needed.

Snooper, Bus Adapter, or Memory:
Ignore.

9. Power and Thermal Management

The PowerPC 970FX power management design is optimized to achieve high performance whenever it is needed, while minimizing the operating power during both active and idle periods.

9.1 Definitions

9.1.1 Full Power Mode

Full Power (Full Run mode) is the default power mode of the processor. After initialization or reset, the processor will always be in this mode. All internal units are clocked at full clock speed and are fully operational.

9.1.2 Doze Mode

This mode is entered from full power mode after the processing core has been quiesced, and instruction fetch and data prefetch have ceased. This mode is a power saving mode, since only the circuitry needed to provide bus snooping capability and maintain memory coherency is active. An interrupt condition such as external interrupts, decrementer, hreset, sreset, thermal management, or machine check is required to return to full power.

9.1.3 Nap Mode

Nap mode provides additional power savings beyond Doze mode. In general clocks to all internal units are switched off. Only the timer/decrementer facility, the I/O circuitry, and part of the pervasive unit are clocked and operating. The phase-locked loop (PLL) is running and stays locked to the global system clock (SYSCLK). The clock mesh is operating, as is the bus clock.

To enter Nap mode, the HID0[NAP] and then MSR[POW] must be set. The processor will then gate its core clocks and enter Doze mode. In Doze mode, the processor will continue to snoop, but will assert its QREQ signal, to indicate to the chipset that it is prepared to go into Nap mode if snooping is not required. If the chipset determines that there is no memory activity requiring the processor to snoop, it asserts QACK. Once the processor detects the assertion of QACK, it transitions to Nap mode. While in Nap mode, QACK is monitored constantly. If it is dropped, the processor transitions back to Doze mode.

If the processor has to act upon an incoming snoop, the BIU becomes active and QREQ is deasserted. However the processor stays in Doze mode waiting for the BIU to become idle again. As soon as the BIU is idle, QREQ is issued again. QACK can be reactivated when snoop is completed (after snoop response time). The processor switches back to Nap mode after QACK is received.

If QACK is received without QREQ being sent (for example, the BIU is not idle), the processor enters an error state. If QACK is deactivated while the processor is switching to Nap mode, the transfer to Nap mode completes before the processor is brought back to Doze mode. Any external interrupt, reset, or check condition transfers from Nap mode back to full power mode.

9.1.4 North Bridge Considerations

The PI participates in the system power management through two asynchronous control signals called QREQ and QACK. QREQ is a processor output signal that is asynchronously sampled by the local clock of the North Bridge. QACK is a North Bridge output signal that is asynchronously sampled by the local clock of the processor and other bus masters.

Figure 9-1 is a flowchart of the sequence of steps for the processor to enter Doze or Nap mode. *Figure 9-2* is a flowchart of the sequence of complementary steps taken by the North Bridge in response to the assertion or negation of QREQ by the processor. In Doze mode, the processor must be capable of snooping all reflected command packets from the North Bridge. In Nap mode, the processor is not required to snoop transactions, although it must be capable of returning to Doze mode for the purpose of snooping if QACK is negated.

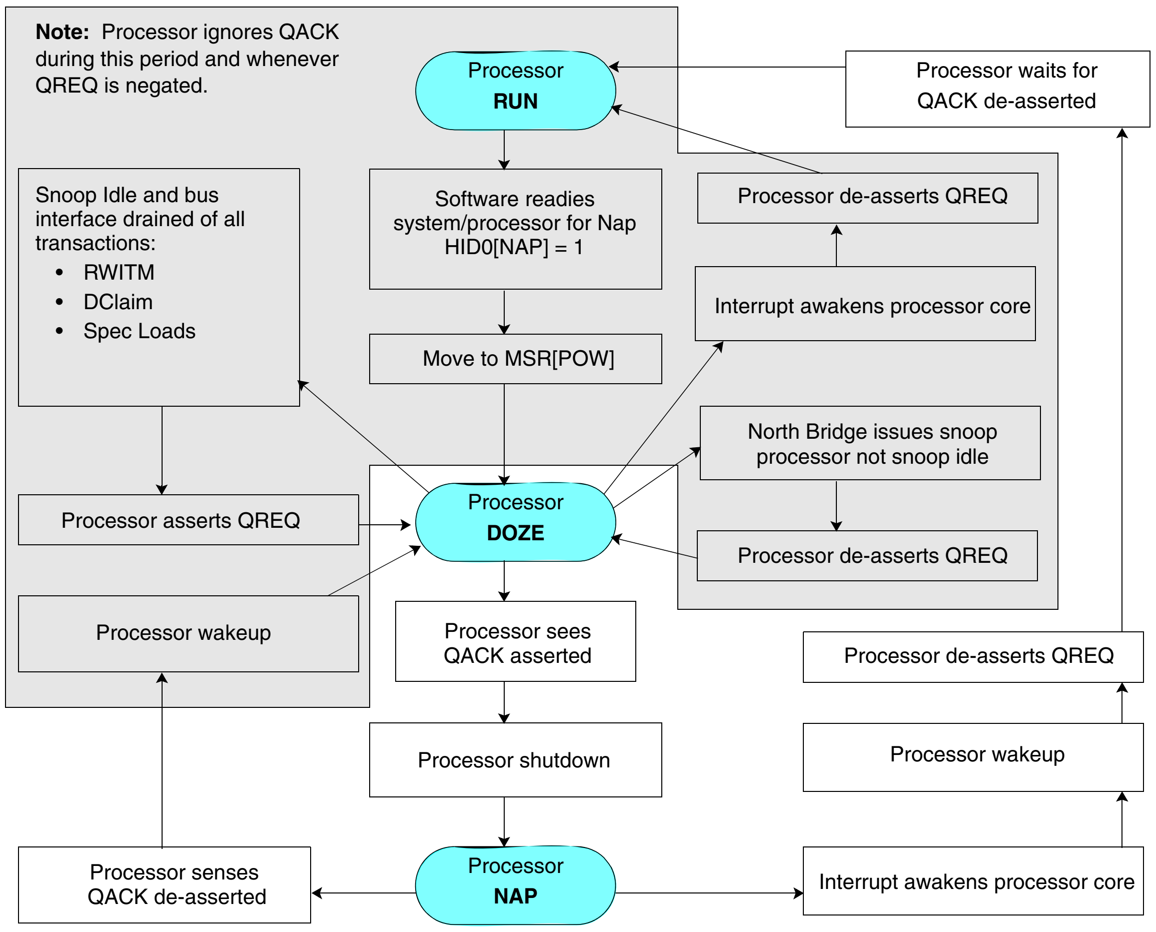
In the normal (or desired) sequence of events, the processor and North Bridge observe a 4-phase handshake for QREQ and QACK. The processor first asserts QREQ after the processor has quiesced, the snoopers are idle, and all outstanding PI bus transactions have completed. The processor then waits for the North Bridge to assert QACK. While the processor is waiting for the assertion of QACK, it is in an intermediate mode called Doze. Once the North Bridge asserts QACK the processor enters Nap mode. To exit Nap mode the processor negates QREQ and then waits for the North Bridge to negate QACK before returning to the Run state.

There are a few scenarios in which the four-phase handshake is preempted:

1. While in Doze mode the North Bridge reflects command packet snooping. The action taken by the processor is to negate QREQ while snooping the reflected command packet while staying in Doze mode.
2. While in Doze mode the processor receives an interrupt. The action taken by the processor is to negate QREQ and return to the Run state.
3. While in Nap mode the North Bridge negates QACK while the processor has QREQ asserted. The processor must then return to Doze mode within 64 bus clocks so that it can return to snooping reflecting command packets from the North Bridge.

As shown in *Figure 9-2*, the North Bridge normally negates QACK when QREQ is negated by any of the attached processors. However, it might also negate QACK if there is bus activity from any of the other attached bus devices that can be a bus master.

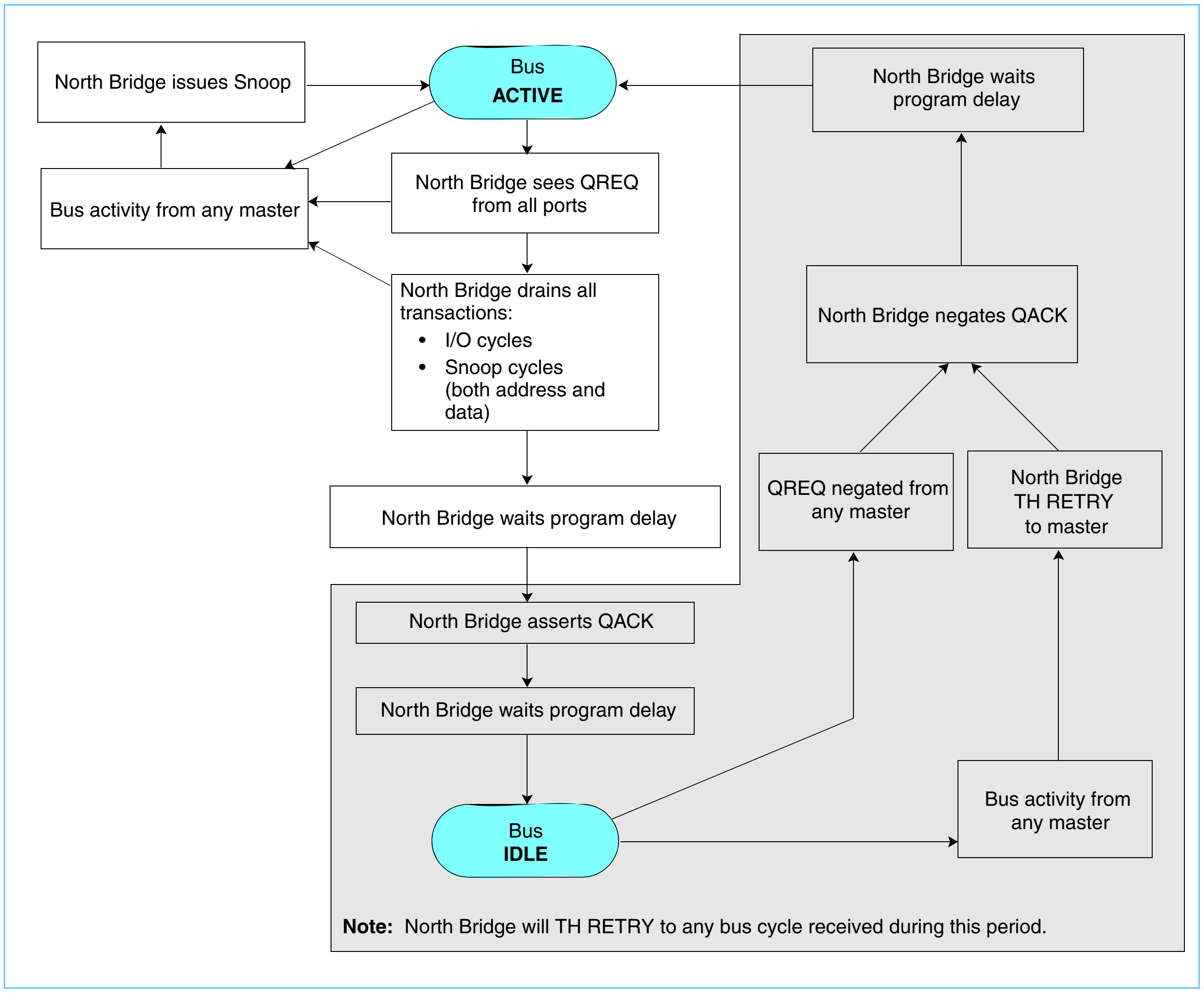
Figure 9-1. Processor Sequence to Enter Doze or Nap Mode


Note: Processor ignores QACK during this period and whenever QREQ is negated.
Processor RUN
Processor waits for QACK de-asserted
Snoop Idle and bus interface drained of all transactions:
• RWITM
• DClaim
• Spec Loads
Software readies system/processor for Nap HID0[NAP] = 1
Processor de-asserts QREQ
Interrupt awakens processor core
Move to MSR[POW]
North Bridge issues snoop processor not snoop idle
Processor asserts QREQ
Processor DOZE
Processor de-asserts QREQ
Processor wakeup
Processor sees QACK asserted
Processor de-asserts QREQ
Processor shutdown
Processor wakeup
Processor senses QACK de-asserted
Processor NAP
Interrupt awakens processor core

Processor State Transition Delay

1. Delay from "Processor senses QACK de-asserted" to "DOZE" is 25 to 72 processor cycles
2. Delay from "Interrupt awakens processor core" to "Processor de-asserts QREQ" is (25 to 72 processor cycles) + 48
3. Delay from "Processor de-asserts QREQ" to "RUN" is less than a maximum of 256 cycles

Figure 9-2. North Bridge QREQ/QACK Signalling


North Bridge issues Snoop
Bus
ACTIVE
North Bridge waits program delay
Bus activity from any master
North Bridge sees QREQ from all ports
North Bridge drains all transactions:
• I/O cycles
• Snoop cycles (both address and data)
North Bridge negates QACK
QREQ negated from any master
North Bridge TH RETRY to master
North Bridge waits program delay
North Bridge asserts QACK
North Bridge waits program delay
Bus
IDLE
Bus activity from any master
Note: North Bridge will TH RETRY to any bus cycle received during this period.

9.2 Power Management Support

System software manages power dissipation in a variety of ways, using a number of hardware facilities.

9.2.1 Power Management Control Bits

Dynamic power management (DPM) refers to the cycle-by-cycle control of clocks as hardware facilities are used for computation, and then go idle for some cycles. This gating of clocks while circuits are idle saves power with no reduction in performance. In normal operation, DPM should be enabled. It can be disabled, however, by negating HID0[DPM]. To enter Nap mode, software must first set a bit HID0[NAP] and then set MSR[POW] to trigger the transition to that mode. The power management control bits are summarized in *Table 9-1*.

Table 9-1. Power Saving Mode

Bit	Name	Power Saving Mode
HID0[9]	NAP	Nap
HID0[11]	DPM	Dynamic Power Management (enable)
MSR[45]	POW	POW bit

9.2.2 Interrupts

The only way to get from a power saving mode back into the full power mode is by asserting one of the following interrupts:

- External interrupt
- Thermal management interrupt
- Decrementer interrupt.

Before entering a power saving mode, the MSR[EE] bit must be set to enable these interrupts. When an interrupt is taken, it automatically resets the MSR[POW] bit, so software must set it once again to reenter a power saving mode.

9.2.3 Bus Snooping

After the HID0[NAP] bit is set and then the MSR[POW] bit is set, the processor enters Doze mode and asserts its QREQ signal. In this mode, core clocks are gated to reduce power, but clocks in the STS are still active to support bus snooping. The PLL, timers, and interrupt logic are also active in all the idle modes. The processor must remain in this Doze mode for as long as the system determines that snooping is required. The assertion of QREQ indicates to the system the processor's readiness to go into Nap mode. Once the system determines that snooping is not currently required, it can assert QACK. When the processor receives this signal, it will complete the transition to Nap mode.

If snooping is required again, the system can negate QACK, signalling to the processor that it must transition back to Doze mode and begin snooping the bus. After a sufficient delay, activity can be initiated on the bus. If this bus activity once again ceases, the system can assert QACK and the processor will go back into Nap mode.

9.2.4 Bus States while in Power Saving Modes

When serving snoops, the BIU is active and drives the outputs as required. When in Nap mode, there is no snooping.

- Data Out Bus, Transfer Handshake Out and,Coherence Response are driven to an idle mode.
- Clock Out is always be driven with the proper clock signal.
- Clock In expects to receive a clock signal.

9.3 Software Considerations for Power Management

9.3.1 Entering Power Saving Mode

The following code sequence should be used to enter a power save mode.

```
.......
.......
mthid0 (NAP)
.......
.......
.......
.......
loop:         dssall  (VPU prefetching stop)
sync
mtmsr (POW)
isync
br   loop
.......
.......
```

The Data Stream Stop All (**dssall**) instruction is needed to stop the prefetch engines started in behalf of the VPU prefetches. Only the above sequence will bring the processor into the power save mode. Switching the Move to HID0 (**mthid0**) and the Move to Machine Status Register (**mtmsr**) in the above sequence does not result in a switch to a power saving mode. When an interrupt is taken, it resets the MSR[POW] bit.

9.3.2 External Interrupt Enable

Only an external interrupt or timer interrupt will bring the processor back from the power save mode. Therefore note that MSR[EE] **must** be set before entering the above loop. Failing to set MSR[EE] and applying an external interrupt or a timer interrupt will result in unpredictable behavior by the processor.

9.4 Power Tuning Overview

Power tuning allows for operation of the PowerPC 970FX at a reduced power level. This is accomplished by frequency control of the on-chip (core) and off-chip clock frequencies. Presently, the power tune functionality provides for reduction from the maximum frequency to 1/2 of the maximum.

Table 9-2. Power Management Modes

Static Power Management Modes	Frequency Scaling
Full, Doze, Nap	f
Full, Doze, Nap	f/2

Note:

1. See the *IBM PowerPC 970FX RISC Microprocessor Datasheet Supplement* for actual power dissipation specifications.

In the system, all processor units (PUs) and the PU interfaces in the North Bridge change the power tuning mode concurrently. Any processor unit can request the mode change. This information is then transmitted to the North Bridge via the PI bus as a special request. The North Bridge grants the requests and mirrors this special request to all PUs and waits for all PUs to signal that they have quiesced the bus and are ready to switch mode. The North Bridge then triggers the mode switch for the supported bus ratios of 2:1 and 3:1.

The frequency scaling on the PI bus requires changing of the RoundTripDelay and the TargetTime parameters. Since the I/O voltage is not changed, an initial alignment procedure (IAP) is not required. The new parameters are sent along with the power tuning command; they overwrite the old parameters when the frequency switch occurs. No parameter change is required for the power save frequency, since no bus activity is active in this mode.

When switching power tuning modes consideration must be given to the following items:

- Switching from high to low frequency will result in a loss of accuracy and resolution of the decrementer counter and will slow reaction on interrupts. The operating system has to set the decrementer counter in order to prevent event and interrupt misses or queue overflow on external devices.
- Some interfaces must be running at a constant speed and voltage independent of the internal frequency and voltage (for example, I^{2}C interface, SDRAM interface, and PCI interface).

9.4.1 Power Tuning Definitions

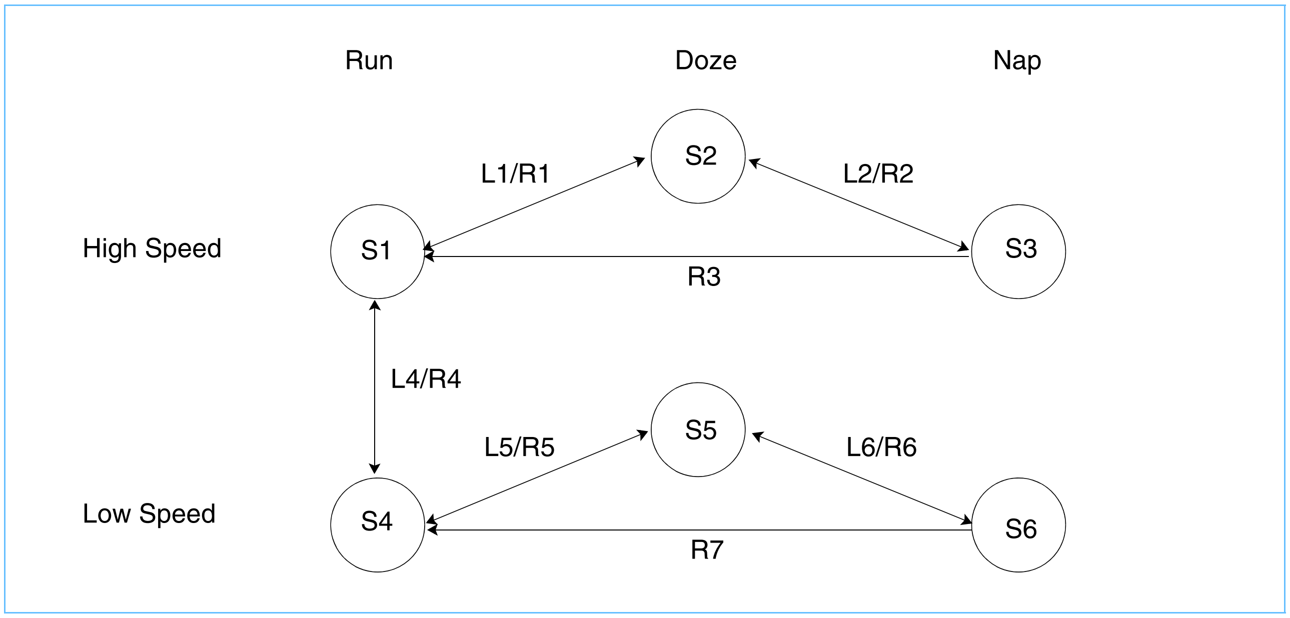
Bus clock (Bclk)	The external bus clock has half the frequency of the data clock due to the double data rate transmission mode on the PI.
Data clock (Dclk)	The bus data clock has a frequency of *1/n-th* of the mesh clock, where *n* is the bus ratio. Valid values for the bus ratio are 2 and 3; with 8 and 16 supported only for test purposes.
Local clock (lclk)	Full frequency clock as delivered by the PLL, but with the same analog delay as the mesh. Every rising edge on mclk has a concurrent rising edge on lclk with a small skew.
Mesh clock (mclk)	Logic behind the PLL generates full, half, or quarter frequency of the PLL clock and sends it on the mesh. The PLL also guarantees that some rising edge of the mesh clock at a latch is aligned to some rising edge of the SYSCLK when using full frequency.
PI	Processor interconnect bus (processor interface).
PLL/Full frequency clock	Frequency is either 8 or 12 times the SYSCLK.
PSYNC	A signal provided by the North Bridge which is active for one rising edge of SYSCLK every 24 SYSCLK cycles.
PSYNC edge	A special mesh clock rising edge, aligned with a rising edge of the SYSCLK while the external PSYNC is active.
SYSCLK	This is the system clock as provided on the board.
Time0	Time0 mark. A special rising edge on the bus clock, which is either concurrent to the PSYNC edge or *4 x* bus clocks edges (i.e., two external bus clock phases) away.

Note: Not all bus ratios are valid for all frequency modes, when taking the following into consideration: the external PSYNC is a 24:1 SYSCLK signal, the PLL multiplication factors, and the Time0 definition.

9.4.2 Power Modes

Figure 9-3 is a state diagram showing the various power modes supported in the 970FX and the transitions between them. *Table 9-3* identifies the power states in that diagram, and *Table 9-4* identifies the transitions labeled in the diagram.

Figure 9-3. 970FX microprocessor Power Modes



There are 5 transitions that *lower* power dissipation, indicated as L*x*, corresponding to left to right or top to bottom transitions in the diagram. Each of these has a reciprocal transition that *raises* power dissipation, indicated as R*x.* In addition, there are 2 transitions which raise power (R3 and R7) and do not have reciprocal transtions.

Table 9-3. Power Mode States

State	Description
S1	Full Run, High Speed
S2	Doze, High Speed
S3	Nap, High Speed
S4	Full Run, Low Speed
S5	Doze, Low Speed
S6	Nap, Low Speed

The *two Full Run modes, (S1, S4)*, one each for high and low speed, correspond to all operating processor functions. The *two Doze modes, (S2, S5),* involve limited functionality, which include bus snooping, but not instruction execution. The timers (decrementer and time base) continue to run during Doze modes, as does the logic for responding to interrupts. The *two Nap modes, (S3, S6),* correspond to a level of functionality below Doze, in which snooping is not supported, but timer and interrupt logic is still active.

The state transitions between Run, Doze, and Nap at a given frequency are triggered as in the prior PowerPC 970. For full frequency, these are transitions L1, R1, L2, R2, R3; and for low frequency L5, R5, L6, R6, R7.

Table 9-4. Transitions between Power Modes

Transition	From	To	Trigger
L1	Run, High	Doze, High	MSR[POW] with HID0[NAP] = ‘1’
R1	Doze, High	Run, High	Interrupt
L2	Doze, High	Nap, High	QACK asserted
R2	Nap, High	Doze, High	QACK negated
R3	Nap, High	Run, High	Interrupt
L4	Run, High	Run, Low	Power tuning command
R4	Run, Low	Run, High	Power tuning command
L5	Run, Low	Doze, Low	MSR[POW] with HID0[NAP] = ‘1’
R5	Doze, Low	Run, Low	Interrupt
L6	Doze, Low	Nap, Low	QACK asserted
R6	Nap, Low	Doze, Low	QACK negated
R7	Nap, Low	Run, Low	Interrupt

The transition from a Full Run mode to a corresponding Doze mode is initiated by the software setting the MSR[POW] bit to a ‘1’, when the HID0[NAP] bit is a ‘1’. This will trigger the normal idle mode sequence that causes I-fetch to quiesce, the BIU to quiesce, the clocks to the core to be gated, and QREQ to be asserted. At this point, the processor is in Doze mode. If or when QACK is asserted, the clocks driving the snoop logic are gated, and the processor enters Nap mode. Once in Nap mode, if QACK is negated, the snoop logic is reactivated and the processor returns to Doze mode. From either Doze mode or Nap mode, an interrupt (External, Decrementer, System Management, Performance Monitor, or Reset) will reactivate all the clocks, returning the processor to Full Run mode, where it will execute instructions starting at the corresponding interrupt vector. This brief description of the transitions among corresponding Full Run, Doze, and Nap modes applies to both processing speeds (high and low).

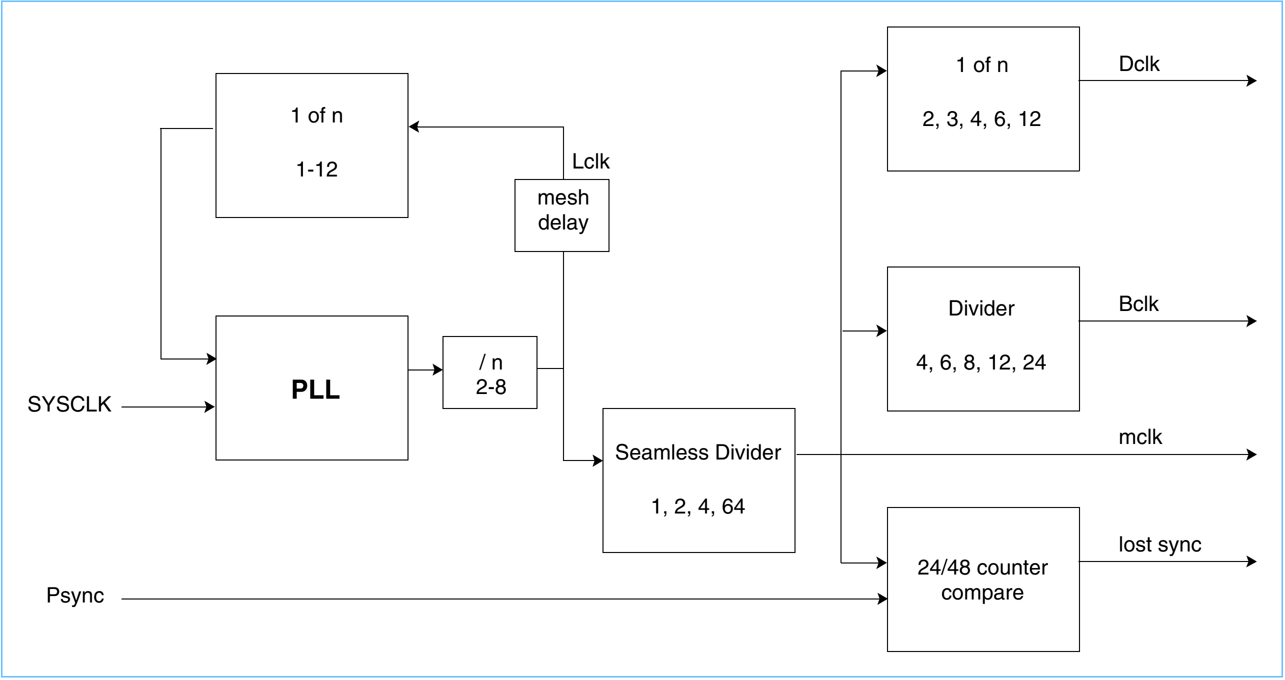
The power tuning transitions are initiated by one processor writing to the Power Control Register (PCR), but are then mediated by a special bus transaction to the North Bridge, which reflects that transaction to all processors in the system. Power tuning transitions involve changing the mesh frequency while in Run mode. These transitions are initiated by software changing the Power Control Register (PCR) in one of the processors, which causes a special bus transaction to the North Bridge. This transaction is reflected by the North Bridge to all processors in the system, which causes those processors to begin the transition process. Processors indicate their readiness to make the frequency switch itself by asserting QREQ to the North Bridge, and the North Bridge responds when it is ready and after it has received all the QREQs from the processors by asserting QACK to all processors. The time it takes to get to this point in the procedure varies. It depends on activity levels in the processors and North Bridge.

9.5 PLL Design

The PLL is designed to support the frequency scaling capability of the 970FX microprocessor. Both the processor clock and the bus clock are derived from the reference clock input to the chip in the PowerPC 970FX design. For frequency scaling it is assumed that the reference clock, SYSCLK , and the related synchronizing clock, Psync run at a constant frequency.

The PLL uses a fixed divider in the feedback path, but a variable, seamlessly switched divider in the forward path. The fixed feedback path allows the PLL to constantly run at a fixed frequency, avoiding the need to relock when switching frequencies. The processor clock (mclk) and bus clock (Bclk) frequencies can be changed seamlessly, while maintaining the ratio between these two clocks at a fixed value. *Figure 9-4* shows this design. Note that the processor interface (PI) supports a double data rate bus. Therefore, the data rate clock (Dclk) is twice the Bclk frequency, and is constrained by the processor design to be no more than half the mclk frequency.

Figure 9-4. PLL Design



The PLL is designed to allow a feedback divider value ranging from 1 to 12, in series with an additional divide by 2 to 8 in the feedback path. The forward divider is also in series with the divide by 2 to 8. To generate mclk from the PLL output frequency, it has selectable values of 1, 2, 4, and divide by 64 mode. The forward divider can then generate the data rate clock from mclk with selectable values of 2, 3, 4, 6, and 12 (values of 8 and 16 are also available for debug, but are not supported on the processor interface bus, nor by the frequency scaling facility). Despite these many possible configurations, one constraint that limits the combinations of frequencies that can be used in the PowerPC 970FX is imposed by the psync counter.

Note: The seamless divider currently supports values of 1 and 2.

The psync counter in the PowerPC 970FX continuously counts 24 mclks and then resets to zero, except when Dclk values of 4 and 12 are used. In these cases, the counter will count to 48. This psync counter is used to generate PI control signals that are synchronized with the North Bridge (NB) drivers and receivers, as mediated by the Psync signal. Whenever a Psync pulse is detected, the psync counter value is checked to be sure that synchronization is maintained. Since the Psync pulse occurs once every 24 SYSCLK cycles, the mclk frequency is constrained to be a multiple of the SYSCLK frequency (an even multiple in the case of a Dclk divider of 4 or 12). The frequency scaling capability on the PowerPC 970FX further constrains the clock configuration values, since this psync counter constraint applies to the reduced frequency, as well as the high frequency clock rates.

To meet the psync counter constraint the allowable divider values in the feedback path are multiples of four. With a feedback value of eight, for example, using a forward divider value of '1' yields the high frequency mclk that is eight times the SYSCLK. Using a divider value of '2' then yields the medium frequency mclk that is four times the SYSCLK, and using a divider value of '4' yields the low frequency mclk that is two times the SYSCLK.

There are several constraints on frequency configurations for the PowerPC 970FX besides that imposed by the psync counter. See the *IBM PowerPC 970FX RISC Microprocessor Datasheet* for information on the allowed frequency range of the processor.

9.6 Time Base and Decrementer

The time base and decrementer registers will run at a constant frequency, independent of changes to the processor and bus frequencies. The default operation of these timers is to run at 1/8 the full processor frequency, even when the processor itself is running at a lower frequency. When TBEN is configured to clock these timers (HID0[NAP] = '1'), the timers will run at the TBEN frequency. When the external clock input mode is used, the TBEN input frequency must not exceed the value specified in the *IBM PowerPC 970FX RISC Microprocessor Datasheet*.

Since the mesh clock frequency can be lowered to 1/64th of the full-speed, the time base and decrementer might be increased or decreased by more than one at a time. Therefore testing that the decrementer has reached the value of zero in order to generate an internal interrupt is not sufficient. The logic detects that the counter has wrapped around. Additionally, the time resolution of the counters cannot exceed the mesh clock frequency.

9.7 I^{2}C Bus Interface

The I^{2}C bus interface operates at a constant speed independent of the current processor frequency.

9.8 Frequency Scaling

Whenever an application requires less than the maximum performance available from the processor, active power can be reduced linearly by reducing the processor clock frequency. Frequency scaling on the PowerPC 970FX involves changing the bus frequency along with the processor frequency, due to the high speed of the PI bus, and the constraint that the processor frequency be at least twice the bit rate of the bus. In order to support frequency scaling in a multiprocessor system, the North Bridge must be involved in initiating the sequence. That sequence is summarized as follows:

1. The operating system, running on one of the processors, detects a change in idle time, or predicts a change in processing requirements.
2. That processor writes a configuration value to the power control register (PCR). The write to this register initiates a bus transaction which has high priority in the STS, and goes out with a newly defined transaction type (0x05). The configuration value written to the register is used as the *address* of the bus transaction. See *Section 9.8.4 Power Adjust Bus Transaction*.
3. The North Bridge responds by negating QACK to any napping processors and reflecting the special transaction and address to all processors.
4. Each processor snoops this bus transaction, captures the address, and negates its QREQ signal if it is in DOZE mode. Once it acknowledges the bus transaction, the processor sets a bit in the power status register (PSR) and starts its frequency change state machine.
5. When the North Bridge receives the command acknowledgement, it waits at least 12 bus beats to insure that all QREQs associated with NAP/DOZE have been negated, then kicks off its frequency change state machine.
6. Each state machine oversees the process of achieving a quiescent bus state that is sufficient to allow the frequency change. This quiescent state corresponds to completing currently active transactions to the point where no timing constraints exist on the bus between what has been sent, and what will be sent next.
7. When each processor has its bus in a quiescent state (sending null transactions, acknowledging received transactions, accepting incoming data), it asserts QREQ.
8. When the North Bridge has its bus in a quiescent state (sending null transactions, acknowledging received transactions), and all QREQs from all processors are now asserted, it no longer has to accept or respond to incoming transactions, and so it will assert QACK to all processors.
9. In response to this QACK, each processor can now stop responding to incoming transactions and so will stop its bus clocks at the beginning of the next time0 cycle. Meanwhile, the North Bridge is executing its frequency change.
10. After stopping its bus clocks, each processor executes the frequency change, by changing the forward divider on the PLL. At this time, the PI configuration values are also changed.
11. After the frequency has been changed, each processor updates its (core speed) psync counter from its full-speed psync counter.
12. At the beginning of the next time0 cycle, each processor starts its bus clocks.
13. Each processor then negates QREQ, indicating it is ready to accept incoming transactions.
14. When the North Bridge has detected that all QREQs are negated, and the North Bridge has finished its frequency change sequence, it negates QACK, indicating to the processors that they can commence outgoing bus activity.

15. In each processor, the status bit in the PSR indicating a frequency switch in progress is reset. The state machine goes idle and the frequency switch is complete.

9.8.1 Initiating a Frequency Change

Software initiates a frequency change by writing to the PCR. The value written to the PCR frequency field determines the target frequency being switched to. The values in the parameter fields must correspond to this new frequency. Similarly, if the voltage field is used, the voltage requested must correspond to the frequency requested. The North Bridge is responsible for changing the voltage before the frequency change when raising voltage, and after the frequency change when lowering the voltage.

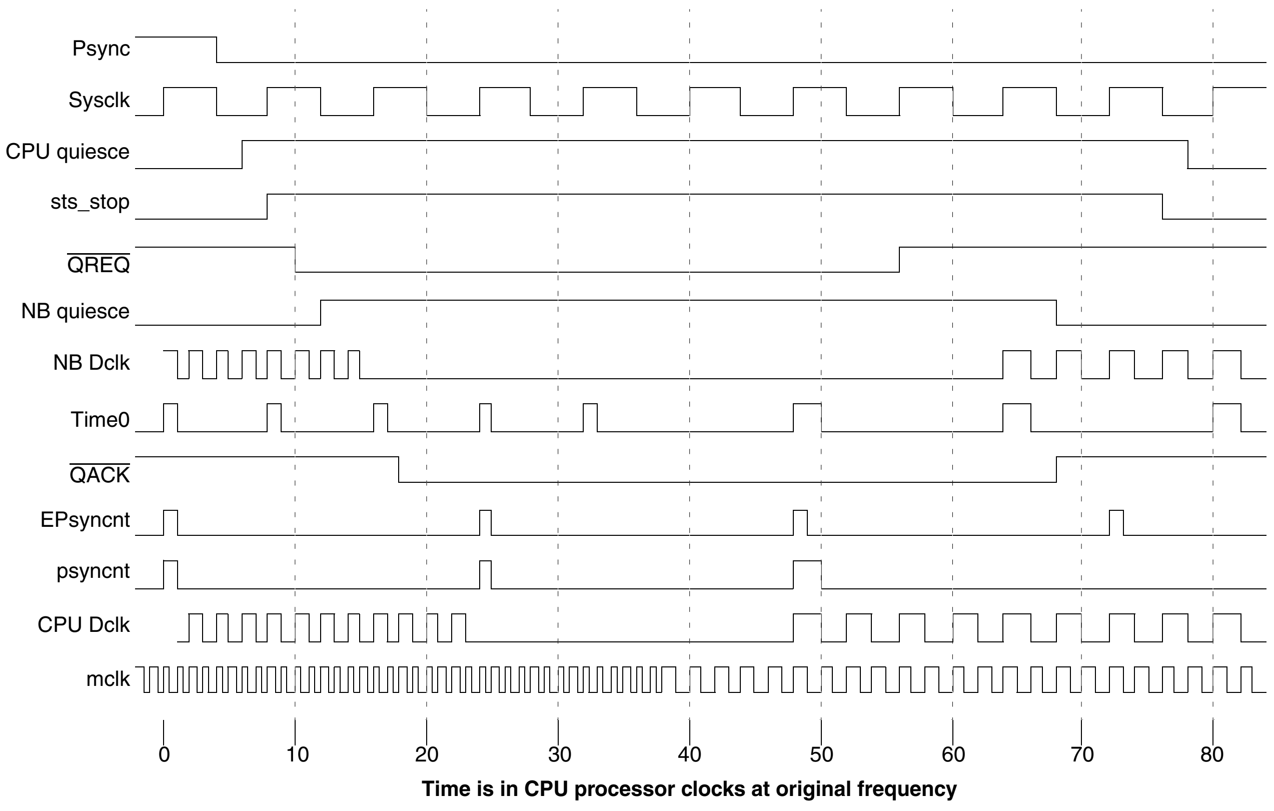
The QREQ and QACK signals have been overloaded to provide handshaking during the frequency change procedure. Therefore, these signals are not available for their normal use (handshaking for Nap mode) during the procedure. System hardware or software must enforce the negation of these signals at the beginning of the procedure. If a processor puts itself into Nap mode during the frequency change procedure, the processor blocks assertion of the QREQ signal for Nap signalling until after the frequency change is complete.

The waveforms in *Figure 9-5* show the ordering of events on the CPU to North Bridge interface during a frequency change in which the clocks are slowed to half speed. The time shown at the bottom of the figure is in CPU processor clocks at the original frequency. However, this figure is intended to show the ordering of events, and not actual latencies between events. Latencies are discussed in *Section 9.8.6 Frequency Scaling Latencies*.

The sequence in *Figure 9-5* starts at the point after a CPU has sent the change request to the North Bridge and the North Bridge has reflected that request to all the processors. Each CPU then completes any bus transactions in progress, and reaches a quiescent state. The CPU quiesce signal shown in *Figure 9-5* is intended to indicate that the quiescent state is reached at cycle 6. Two cycles later, the CPU asserts its internal sts_stop signal. At this point, the core no longer has access to the L2 cache or bus. Two cycles later, the CPU asserts QREQ. During this time, the North Bridge has also been progressing toward a quiescent state. The North Bridge quiesce signal indicates that this state is reached at cycle 12, though it might occur before QREQ is asserted.

The combination of QREQ asserted and North Bridge quiescent causes the North Bridge to stop its bus clocks on a Time0 boundary, which occurs at cycle 16. The North Bridge may instead continue to run its bus clock, as long as it drives null transactions during the period that the bus clocks would otherwise be stopped. This is followed by the North Bridge asserting QACK, shown at cycle 18. Once QACK is asserted, the CPU stops its bus clocks on the next internal psync boundary (psyncnt), shown at cycle 24. With its bus clocks stopped, the CPU changes the frequency of its processor (and therefore bus) clock, shown at cycle 38. Once the frequency change occurs, the CPU starts its bus clocks on the next psync boundary, shown at cycle 48. After starting its bus clocks, the CPU negates QREQ, shown at cycle 56. The North Bridge then starts its bus clocks on a Time0 boundary (cycle 64) after which it negates QACK (cycle 68). Internal to the CPU, the negation of QACK leads to the negation of sts_stop (cycle 76) allowing core access to the L2 cache and activity to proceed on the bus.

Figure 9-5. Frequency Scaling Event Ordering



9.8.2 Power Control Register

Software writes to the PCR bits to indicate that a frequency change is desired, and to pass the information corresponding to that frequency change to all the processors in the system. Writing to the PCR initiates the frequency change process, by generating a special bus transaction that is sent to the North Bridge and eventually reflected to all the processors. The address bits of this special transaction are copied from PCRH[22:31] and PCR[0:31], as described in the following section.

Note: The special bus transaction is generated when the PCR register is written, so the Power Control Register High (PCRH) must be updated as needed prior to writing the PCR.

The PCR is implemented as a SCOM register at address 0x0AA001. The PCRH is implemented as a SCOM register, at the same address as the PCR. The high order bit in the register is used to indicate which register is being written (PCR high order bit = '1', PCRH high order bit = '0'). The 32-bit PCR and PCRH registers are written using **mtspr** instructions that target the SCOMD and SCOMC SPRs, such that the low order 32 bits (bits [32:63]) of the source register are moved to the target PCR or PCRH.

For example, before initiating a frequency change set the following

- Appropriate values in the low order bits of GPR3 to indicate the desired settings for the PCR, (including bit [32] = '1')
- Appropriate values for PCRH in GPR4 (including bit [32] = '0')
- The SCOM address placed in GPR5 (64-bit value of x'00000000 0AA00100')

The following sequence initiates a frequency change:

```
.set SCOMD 277          # SPRN for SCOMD
.set SCOMC 276          # SPRN for SCOMC
mtspr SCOMD, GPR4
isync
mtspr SCOMC, GPR5
isync
mtspr SCOMD, GPR3
isync
mtspr SCOMC, GPR5
```

Note: For the PowerPC 970FX, each frequency change should be preceded by a write to the PCR in which Gd contains all '0's. Not clearing the PCR will prevent further frequency scale commands from being issued by the bus, even though the instruction sequence will complete within the processor.

Table 9-5 describes the bits in the Power Control Register.

Table 9-5. Power Control Register (PCR)

Bit	Description
0	Must be '1'
1:7	Reserved
8:12	Spare field
13:14	Frequency field 00 full frequency 01 half frequency 10 quarter frequency (not supported) 11 illegal
15	Frequency request valid
16	Reserved
17:18	Target time
19:23	STATLAT is the number of bus beats between the last beat of the address/data (AD) packet and the first beat of the transfer-handshake (TH) packet.
24:27	SNOOPLAT is the number of bus beats between the last beat of a reflected command packet to the first beat of the individual snoop responses from each of the processors received at the North Bridge.
28:31	SNOOPACC is the number of bus beats between the last beat of the individual snoop response sent from a processor to the first beat of the accumulated snoop response received from the North Bridge.

The Power Control Register High (PCRH) contains the high order address field, as described in *Table 9-6*.

Table 9-6. Power Control Register High (PCRH)

Bit	Description
0	Must be 0
1:21	Reserved
22:31	High order address field

9.8.3 Power Status Register

The status of the power tuning facility is available in the Power Status register (PSR). This register consists of read-only bits, indicating the current voltage (if supported by software) and the current frequency.

When the power adjust special transaction reflected from the North Bridge is received by the processor, it sets PSR[2] to indicate that a frequency change is in progress. Shortly after the North Bridge has asserted QACK to start the frequency scale, the new frequency field is reflected in PSR[6:7]. Once the frequency scaling has completed PSR[3] is also set to ‘1’.

A SCOM read of the PSR once bit [2] and [3] are set will automatically clear both bits. The PSR is implemented as a SCOM register at address 0x408001 as described in *Table 9-7*.

Table 9-7. Power Status Register (PSR)

Bit	Description
0:1	Reserved
2	Power tuning command has been received
3	Power tuning command has completed
4:5	Reserved
6:7	Current frequency
8:63	Reserved

9.8.4 Power Adjust Bus Transaction

The processor sends a power adjust transaction to the North Bridge to initiate the frequency and voltage scaling sequence in the system. This is a command-only transaction that contains information encoded in a subset of the address bits to indicate the desired target frequency, and the corresponding parameter information. The transaction type and related bus signals for this transaction are as follows:

Table 9-8. Power Adjust Transaction

Bus Operation	Power Adjust
Transaction type	00101 (0x05)
Address modifiers (WIMGRP)[1]	001000
Tag field	11011

1. W = write through, I = cache inhibited, M = memory coherent, G = guarded read, R = rerunning, P = pipelined snoop.

The encoding of the address bits for this transaction is as follows:

Table 9-9. Power Adjust Transaction

Bit	Description
0:21	Not implemented
22:31	High order address bits
32:39	Reserved
40:44	Spare field
45:46	Frequency field
47	Frequency request valid
48	Reserved
49:50	Target time
51:55	STATLAT is the number of bus beats between the last beat of the address/data (AD) packet and the first beat of the transfer-handshake (TH) packet.
56:59	SNOOPLAT is the number of bus beats between the last beat of a reflected command packet to the first beat of the individual snoop responses from each of the processors received at the North Bridge.
60:63	SNOOPACC is the number of bus beats between the last beat of the individual snoop response sent from a processor to the first beat of the accumulated snoop response received from the North Bridge.

These 42 low order address bits are copied from the corresponding fields in the Power Control Register. Software can set bits [22 to 31] to any desired value to make the address fall within some desired range.

The North Bridge uses the frequency field to determine what new frequency is being requested. Only the maximum rated or full frequency and one-half of full frequency are currently supported. The frequency request valid bit must be asserted if a frequency change is being requested. The presence of this bit allows the option of a voltage change only request. If the frequency request valid bit is negated, the North Bridge will not reflect this transaction to the processors. If the frequency request bit is asserted, the North Bridge has the option of reflecting the transaction to the processors in cases where the frequency field itself does not correspond to a valid change. This includes the cases where the frequency field value indicates no change, change to the current value, or change to a value that is not allowed from the current value.

Once the transaction is reflected to the processors, each processor responds as follows:

- If the frequency field indicates no change, the processor does nothing.
- If the frequency field indicates a change to the current frequency, or a change to a new frequency, then the processor must execute the frequency change procedure.

In addition to the four parameters passed in the power adjust transaction, the PI also depends on the values of the programmable bit line and clock delays that are determined during the IAP at power-on. To support frequency scaling, this IAP must be run at the high frequency, high voltage setting for the processor. Then, the effect of running at lower frequencies is to widen the signal eye, while the effect of lowering the core voltage (the I/O voltage remains constant) is to increase all the bit and clock delays.

9.8.5 Clock Dithering

Input current to the processor can change significantly during transitions of the power tuning frequency. These current changes must be controlled to avoid over and under-voltages that a high di/dt might cause due to inductance in the power distribution network. A clock dithering mechanism included in the the power tuning facility enables gradually transitioning between frequencies.

The power tuning facility supports frequency scaling with a constant-frequency PLL that feeds multiple frequency dividers. The outputs of these dividers are fed to a frequency multiplexer, from which one divider output is selected as the processor mesh clock at any given time. Toggling this multiplexer-selection signal during a transition from frequency A to frequency B accomplishes clock dithering. Thus, most clocks are at frequency A at the beginning of the transition. Gradually, more and more frequency-B clocks are introduced in the dithering pattern.

Figure 9-6. Clock Dithering Block Diagram

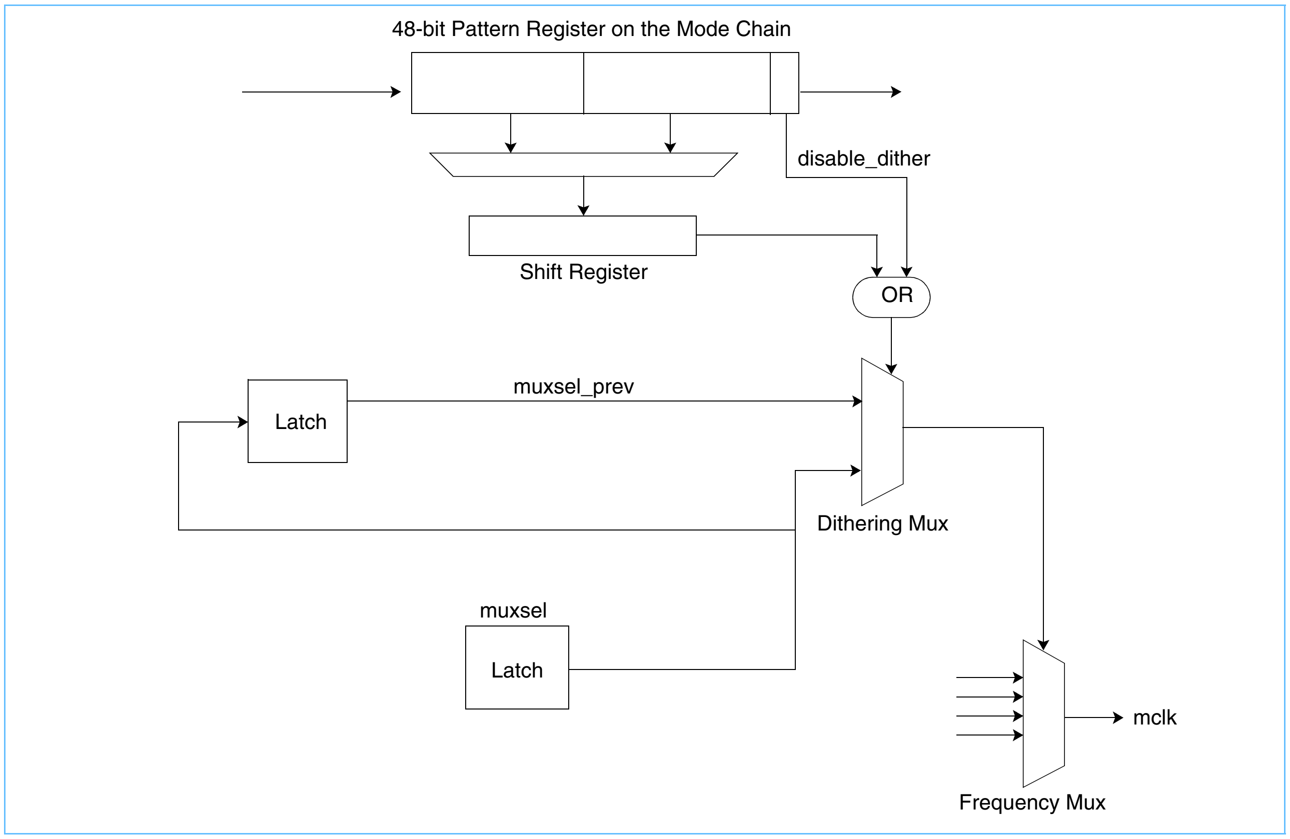
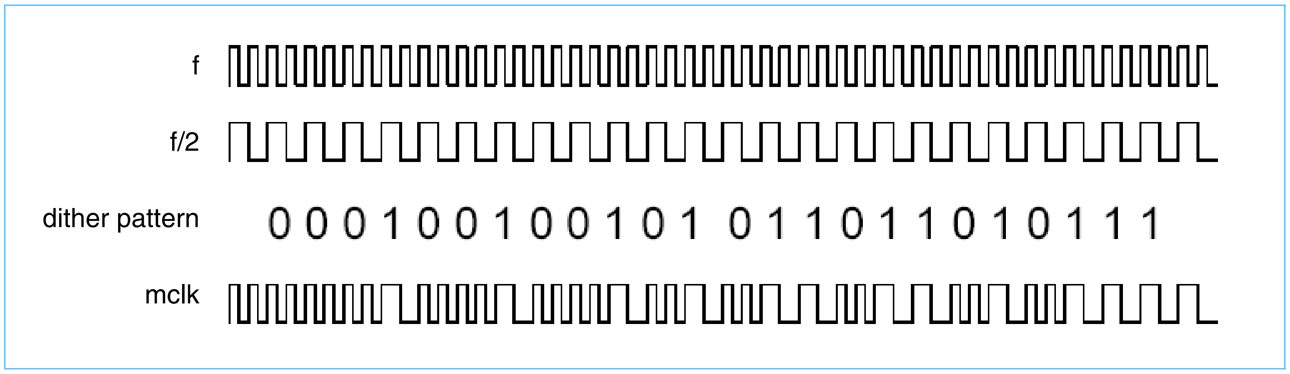


Figure 9-6 shows the components controlling the dithering of the clock. Two, 24-bit dithering patterns are provided in the mode ring. They support distinct dithering patterns for transitions between the high and medium frequencies and the transitions between the medium and low frequencies. When a frequency shift is initiated, the appropriate mode ring pattern is selected using a multiplexer for transfer to a 24-bit shift register. At the same time, the multiplexer select pattern for the previous frequency is saved in the muxsel_prev latch, while the new frequency is loaded into the muxsel latch.

Clock dithering involves a 2-level multiplexer selection process. The shift register is clocked at the lower of the previous and new frequencies. Starting on the rising edge of the mclk/4, it shifts the pattern one bit to the right every cycle, and applies the right most bit to the dithering mux to select a multiplexer select pattern. That pattern is then applied to the frequency multiplexer to select the mesh clock frequency. A '1' bit in the shift pattern selects the new frequency, a '0' bit selects the old frequency. At the end of the shift pattern, a '1' bit is forced, to continuously select the new frequency. A separate mode ring bit can be used to disable clock dithering, by forcing this control bit to always be a '1' via the OR circuit shown in *Figure 9-6*.

As an example of a shift pattern for achieving a gradual transition from high to medium frequency might be '1110 1011 0110 1010 0100 1000' (see *Figure 9-7*). These bits are shifted at the medium frequency. Each '1' corresponds to one cycle of medium frequency. Each '0' corresponds to two cycles of high frequency (since the shift register is clocked at medium frequency). Thus reading the pattern from right to left, the pattern specifies six fast clocks, followed by one medium clock, followed by four fast clocks, followed by one medium clock, and so on, as indicated in the *Figure 9-7*.

Figure 9-7. Sample Shift Pattern



9.8.6 Frequency Scaling Latencies

The sequence for raising the frequency has a latency, (from the time the operating system writes the configuration value in the PCR to the time when the status bit in the PSR indicates that the change is complete), that has the following components:

- Time to signal North Bridge
- Time to signal processors
- Time for North Bridge and processors to quiesce
- Time for North Bridge and processors to handshake
- Time for one psync (1:24) cycle
- Time to handshake and reset the status bit

While the processor signals the frequency change to the North Bridge and until the North Bridge reflects the power adjust command back to the processor, it proceeds normally. Once the processor begins to quiesce the bus, the processor core will no longer be able to access data and instructions from the L2 or bus interface. As long as the processor is able to execute with data and instructions in the L1 caches, it can continue to run. In the best case, the processor will only stall for about a cycle when the mesh clock frequency itself is switched. More specifically, the processor will be unable to respond to interrupts while the bus interface is in a quiescent state, unless the instructions and data needed to handle the interrupt are in the L1 caches prior to the frequency change. This means the interrupt response might be delayed due to a frequency switch. See the *IBM PowerPC 970FX RISC Microprocessor Datasheet* for latency values.

9.8.6.1 Reducing Clock Mesh Power

There are two power saving modes defined for the PowerPC 970FX, Nap and Doze. In Nap mode, the clocks to the core are turned off, while the timers, PLL, and part of the pervasive unit continue to operate. Doze mode is similar, except that snoop logic is also active. Doze mode is entered from full power mode by setting HID0[NAP], and then the MSR[POW] bit. This also causes the $\overline{\text{QREQ\_B}}$ signal to be asserted, requesting that the North Bridge put the bus in a quiescent state. When the North Bridge complies, it asserts $\overline{\text{QACK\_B}}$, causing the processor to transition into Nap mode. Whenever $\overline{\text{QACK\_B}}$ is negated, the processor must return to Doze mode to process snoop transactions.

Figure 9-8 shows the event ordering for frequency scaling. The minimum requirements for the delays noted in the figure are shown in the *Table 9-10* and *Table 9-11*.

Figure 9-8. Frequency Scaling Event Ordering

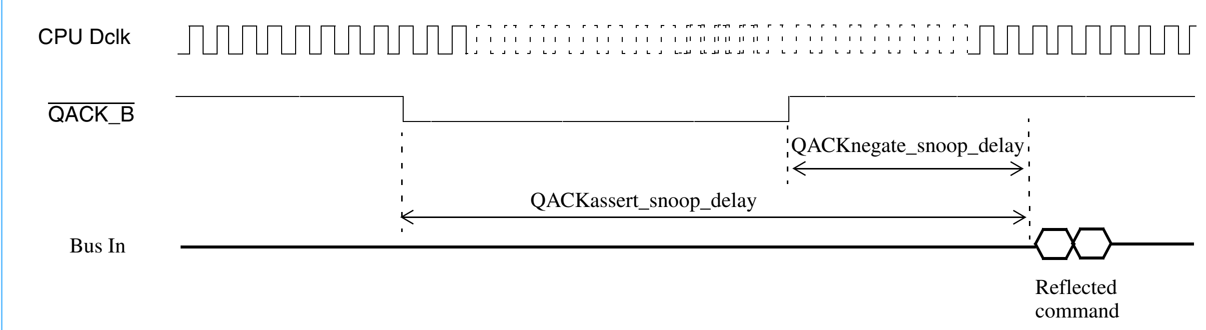


Table 9-10 provides the minimum QACKassert_snoop_delay required for two different bus ratios and two different mesh clock frequencies.

Table 9-10. Minimum QACKassert_snoop_delay Requirement in Bus Clocks for the 970FX

Mesh Clock Frequency	Bus Ratio	
	2:1	3:1
f	90	60
f/2	84	56

Table 9-11 provides the minimum QACKnegate_snoop_delay required for two different bus ratios and two different mesh clock frequencies.

Table 9-11. Minimum QACKnegate_snoop_delay Requirement in Bus Cocks for the 970FX

Mesh Clock Frequency	Bus Ratio	
	2:1	3:1
f	55	37
f/2	52	35

9.9 Additional Dynamic Power Management

The PowerPC 970FX implements dynamic power management (DPM) – the gain of clocks to idle circuits while in an operational mode – in a number of functional units, including the VPU, IDU, STS, and RAS. For example, there are two levels of clock control for the VPU, a coarse level and a fine level. The coarse control is essentially a static form of clock gating control, making use of the VPU available bit (MSR[VP]). When this bit is a zero, the latches in all VPU stages from issue to writeback are gated off. The fine level control is much more dynamic. It occurs on a stage by stage basis within each execution pipeline, starting with the latches following the RF2 stage. When this fine level of control is enabled, all clocks in all of the VPU stages from the register access to the writeback stages are gated off at all times, except for the cycles when that stage has an active instruction in it.

Dynamic power management can be disabled in the RAS units by asserting bit[0] in the JTAG register with modifier address 0x000800. DPM in the rest of the processor can be disabled by negating HID0[DPM].

10. 970FX Performance Monitor

The 970FX microprocessor has a complex, speculative, out-of-order execution core coupled with an equally complex multilevel storage hierarchy. Users concerned with performance analysis and system optimization have access to performance monitoring features, which support a wide range of tasks including the following:

- Profiling memory hierarchy behavior and tuning system algorithms to optimize scheduling, partitioning, and structuring for tasks and data
- Tuning applications for the target system
- Debugging, analyzing, and optimizing processor architecture features

The performance monitor facility provides information for a wide variety of activities and is part of the facilities that are collectively referred to as instrumentation facilities. Instrumentation facilities include matching/sampling, tracing, and thresholding.

Note: The 970FX performance monitor should only be used as a debug facility until characterization of its features and functions is complete.

10.1 Performance Monitoring Facilities Overview

The 970FX performance monitoring facility is an extension to that of earlier PowerPC processors. There are eight Performance Monitor Counter Registers (PMC1-8). They can count a variety of events, many of which are relevant to performance analysis. As before, the counters support user or supervisor and marked or unmarked filtering of events. A marked instruction is one that is eligible for sampling as determined by the instruction fetch unit (IFU) and instruction dispatch unit (IDU) instruction matching facilities.

The most-significant change introduced by the 970FX performance monitor is the concept of indirect events. A subset of the normally selected direct Performance Monitor Counter (PMC) events are multiplexed so that there is a larger number of total available events. Unlike event selection on previous PowerPC processors (which had only direct events), indirect events cannot be configured entirely independently (setting a multiplexer affects the indirect events on more than one PMC). Some indirect events can also be summed together by the hardware. This feature is most often used to sum the performance event counts of a functional unit pair (for example, floating-point unit 0 [FPU0] and floating-point unit 1 [FPU1]).

10.1.1 Performance Monitor Facilities

The instrumentation performance monitor (perfmon) on the 970FX microprocessor includes the following functions:

- Counts up to eight concurrent software selected events in individual 32-bit counters. The counting of events can be enabled by software under several conditions such as user (problem) or supervisor (privileged) state, and Run or Wait state.
- Generates a maskable exception when an event counter overflows (triggering).
- Freezes the contents of the event counters until a selected trigger occurs and then begin counting (triggering).
- Increments the event counters until a selected trigger occurs and then freezes counting (triggering).
- Monitors classes of instructions selected by the instruction matching facility.

- Randomly chooses an instruction for detailed monitoring (sampling).
- Counts start/stop event pairs that exceed a selected timeout value (thresholding).

10.1.2 Performance Monitor Event Selection

One event per counter can be selected for monitoring at a given time. The event to be monitored is selected by setting the appropriate value in the Monitor Mode Control Register (MMCR) bit field for that counter. The events counted might be the number of cycles that the event occurs or the number of occurrences of the event depending on the particular event selected.

10.1.3 Machine States and Enabling the Performance Monitor Counters

Performance monitor counting can be enabled or disabled under several machine states, which are selected using the counting control bit fields in the MMCRs and the state bits in other Special Purpose Registers (SPRs).

10.1.4 Trigger Events and Enabling the Performance Monitor Counters

Certain kinds of conditions and events, called trigger events, can be used to control performance monitor activities such as starting or stopping the counters and causing performance monitor exceptions. These scenarios are selected using the condition/event enable bits fields and the exception enable bits of the MMCRs in conjunction with control bits in other SPRs.

10.1.5 Performance Monitor Exceptions

Trigger events can cause performance monitor exceptions to occur based on the values of the exception enable bits in the MMCRs. An enabled exception might cause a performance monitor exception to occur if the exception is enabled in other SPRs.

10.1.6 Sampling

The 970FX microprocessor can be configured to sample instructions for detailed monitoring. The 970FX microprocessor instrumentation facilities support setting mask values for matching particular instructions or kinds of instructions that are then eligible to be sampled (that is, they are marked for sampling). The performance monitor includes events for counting marked instructions at each stage of the pipeline and in certain other situations. Instruction sampling is a useful facility for gathering both detailed and statistical information for particular instructions.

Note: Instruction marking is entirely separate from thread marking with the performance monitor mode bit in the Machine State Register (MSR[PMM]).The state of the MSR[PMM] bit is only relevant for event counting in order to determine when counters should be frozen (MMCR0[FCM1, FCM0] fields).

10.1.7 Thresholding

Unlike previous PowerPC processors, which implemented thresholding only on load instructions, the 970FX processing unit monitors the pipeline stage progression of sampled instructions and can detect when the stage-to-stage cycle count for a selected start/stop pair of pipeline stages exceeds a specified threshold value.

10.1.8 Trace Support Facilities

The 970FX microprocessor supports both the single step and the branch trace modes as defined by the PowerPC Architecture.

10.2 Instruction Sampling Facilities

10.2.1 Special Purpose Registers and Fields Associated with Instrumentation

The 970FX microprocessor instrumentation facilities and associated 970FX microprocessor components include several SPRs used for or associated with performance monitoring, matching, sampling, and tracing. Unless otherwise noted, the Special Purpose Registers described below and listed in *Table 10-1* on page 189 can be read in user (problem) and supervisor (privileged) state and written in supervisor state by using the Move From Special Purpose Register (**mfspr**) and Move To Special Purpose Register (**mtspr**) instructions, respectively. The MSR Register is read and written by the Move From Machine State Register (**mfmsr**) and Move To Machine State Register (**mtmsr**) instructions.

The 970FX microprocessor instrumentation facilities include the following Special Purpose Registers and register bit fields (also listed in *Table 10-1* on page 189):

- Performance Monitor Mode Control Registers (MMCRx)
  These registers include both counting control and event select bit fields.
- Performance Monitor Counter Registers (PMCx)
  These registers increment each time (or cycle, depending on the selected event) that an event occurs while the counter is enabled. These registers also have the control function for the counter overflow condition.
- Machine State Register [EE] (MSR[EE])
  This register bit is used to enable or disable external interrupts. The performance monitor exception is considered an external interrupt.
- Machine State Register [PMM] (MSR[PMM])
  This register bit is used to enable or disable performance monitor activity controlled by the process mark bit.
- Machine State Register [PR] (MSR[PR])
  This register bit is used to establish user (problem) or supervisor (privileged) mode and the performance monitor counting activity controlled by this bit.
- Machine State Register [SE] (MSR[SE])
  This register bit is used to enable or disable the trace exception after each instruction is completed.
- Machine State Register [BE] (MSR[BE])
  This register bit is used to enable or disable the branch trace exception and after a branch instruction is completed.
- Hardware Implementation-Dependent Register0[13] (HID0[TG])
  This register bit is used to determine the granularity the thresholder uses for counting cycles.
- Control Register[31] (CNTL[31])
  This register bit is used to determine the Wait or Run state and the performance monitor activity controlled by this bit.

- Scan Communication Register x'240' [0:15] (SCOM x'240' [0:15])
  These register bits are used to establish the timeout and resume delays used by the performance monitor to coordinate the matching and sampling facility.
- Scan Communication Register x'340' [11:12] (SCOM x'340' [11:12])
  These register bits are used to establish the matching and sampling filter mode used by the matching and sampling facility to produce marked instructions that can be counted by the performance monitor.
- Instruction Match Content-Addressable Memory (CAM) Registers (IMC)
  The IMC SPRs are used to access the IMC array that contains the mask values used for instruction matching. The Move To IMC (**mtimc**) and Move From IMC (**mfimc**) instructions can be executed only in supervisor mode.
- Time-Base Register [47, 51, 55, 63] (TB[47, 51, 55, 63])
  These register bits are used to enable or disable the time-base events that can be used to enable or disable performance monitor counting.
- Sample Address Registers (SxAR)
  The Sampled Instruction Address Register (SIAR) and Sampled Data Address Register (SDAR) contain the address and data, respectively, relating to a marked instruction. The registers can be read in supervisor (privileged) or user (problem) state, but are modified only by the hardware. The values written to these registers by the hardware depend on the processing state and on the kind of instruction that is being marked for sampling.
- Machine Status Save/Restore Register (SRRO, SRR1)
  These registers are used to save machine status during exception handling. In addition, SRR1[33] is used to determine when the contents of the SIAR and SDAR registers are synchronized, so that they refer to the same marked instruction.

Table 10-1. 970FX Performance Monitor and Trace-Related Special Purpose Registers

Register Name	SPR Address Bits[1]		Function
	5:9	0:4[2]	
MMCR0	‘11000’	‘n1011’	Performance Monitor Mode Control Register 0
MMCR1	‘11000’	‘n1110’	Performance Monitor Mode Control Register 1
MMCRA	‘11000’	‘n0010’	Performance Monitor Mode Control Register A
PMC1	‘11000’	‘n0011’	Performance Monitor Counter Register 1
PMC2	‘11000’	‘n0100’	Performance Monitor Counter Register 2
PMC3	‘11000’	‘n0101’	Performance Monitor Counter Register 3
PMC4	‘11000’	‘n0110’	Performance Monitor Counter Register 4
PMC5	‘11000’	‘n0111’	Performance Monitor Counter Register 5
PMC6	‘11000’	‘n1000’	Performance Monitor Counter Register 6
PMC7	‘11000’	‘n1001’	Performance Monitor Counter Register 7
PMC8	‘11000’	‘n1010’	Performance Monitor Counter Register 8
MSR[61]	Use **mtmsr**, **mfmsr** instructions (supervisor [privileged] mode only)		Machine State Register [Performance Monitor Mark]
MSR[48]			Machine State Register [External Interrupt]
MSR[49]			Machine State Register [User (Problem)/Supervisor (Privileged) State]
MSR[53]			Machine State Register [Single-Step Trace Enable]
MSR[54]			Machine State Register [Branch Trace Enable]
HID0[13]	‘11111’	‘10000’	Hardware Implementation-Dependent Register 0 [Threshold Granularity]
CTRL[31]	‘00100	‘n1000	Control Register [Run Bit]
SCOMC	Use **mtscomc/d** and **mfscomc/d** instructions		Scan Communication Control
SCOMD			Scan Communication Data
IMC	Use **mtimc**, **mfimc** instructions (supervisor mode write, user and supervisor mode read)		Instruction Match CAM Register
TBL [47,51,55,63]	‘01000	‘n1100	Time-base bits used for performance monitor time-base events
SIAR	‘11000’	‘n1100’	Sampled Instruction Address Register
SDAR	‘11000’	‘n1101	Sampled Data Address Register
SRR1	‘00000’	‘n1011’	Machine Status Save/Restore Register 1

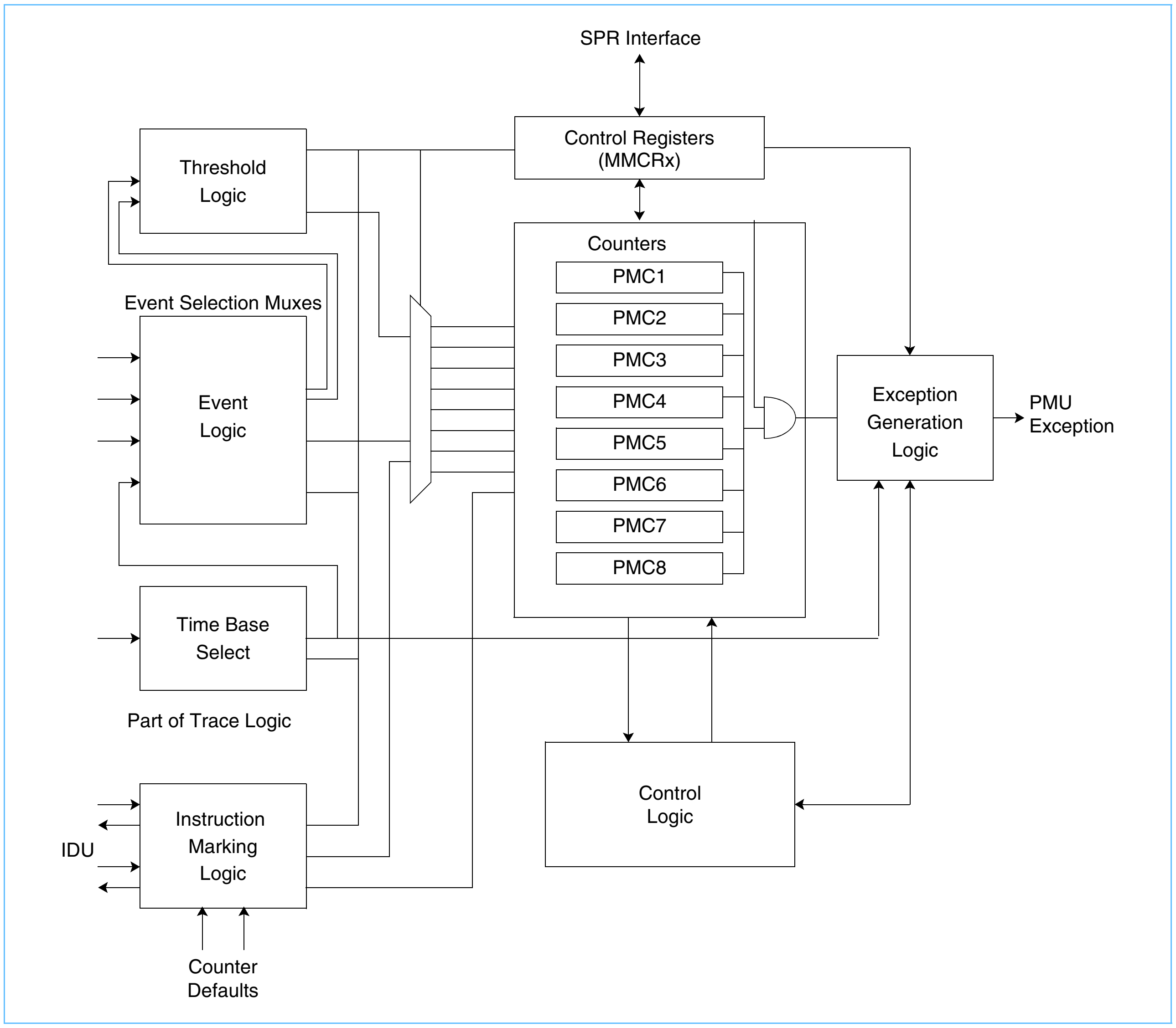
Note:

1. In a **mtspr**/**mfspr** instruction, the instruction SPR field of bits [11:15] hold SPR address bits [0:4] and bits [16:20] hold SPR field bits [5:9].
2. When n is set to ‘1’, it indicates an SPR address value for a supervisor mode **mtspr** or **mfspr** instruction.
   When n is set to ‘0’, it indicates an SPR address value for a user mode **mfspr** instructions.
   For **mfspr**, the instruction is supervisor mode if and only if SPR[0] is set to ‘1’.

10.3 Performance Monitor Components

A schematic overview of the components that make up the 970FX performance monitor is shown in *Figure 10-1*. These components and their use are described in the following sections.

Figure 10-1. Performance Monitor Architecture



10.4 Performance Monitor Control Registers

The Performance Monitor Control Registers, MMCR0, MMCR1, and MMCRA, are used in conjunction with the MSR and other SPRs to set up the performance monitor enable states, exception conditions, threshold values, match criteria, and selection of the events counted in each of the Counter Registers, PMC1 - PMC8.

The MMCRx Register bit assignments are shown in *Section 10.4.1 Performance Monitor Control Register MMCR0* on page 191, *Section 10.4.2 Performance Monitor Control Register MMCR1* on page 194, and *Section 10.4.3 Performance Monitor Control Register MMCRA* on page 197. The MSR bits that relate to performance monitor functions are shown in *Table 10.4.5 Performance Monitor and Trace Related Bits in the Machine State Register (MSR)* on page 200.

For all of the Performance Monitor Control Register fields, it is always understood that the counter is incremented if that action is not prohibited by some other control condition. All of the MMCRx and PMCx Registers flush to zero unless otherwise noted in the following MMCRx and PMCx tables.

10.4.1 Performance Monitor Control Register MMCR0

Figure 10-2. Performance Monitor Control Register MMCR0

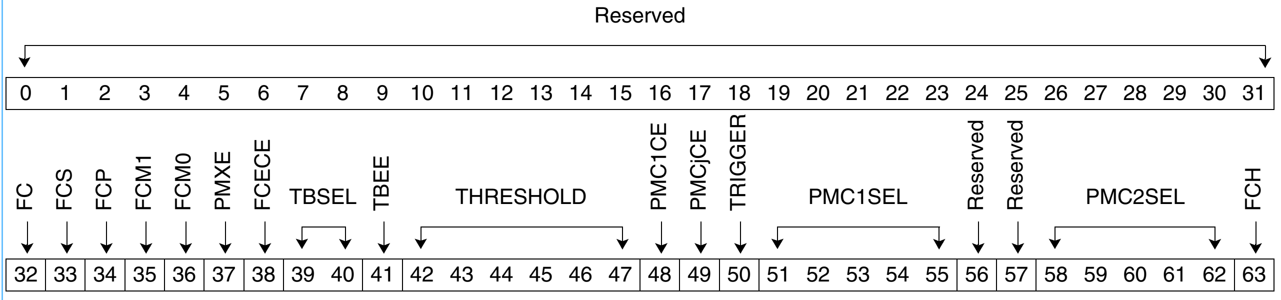


Table 10-2. Performance Monitor Control Register MMCR0

Bits	Field Name	Description
0:31	—	Reserved.
32	FC	Freeze counters. 0 The PMCs are incremented. 1 The PMCs are not incremented. The processor sets this bit to '1' when an enabled condition or event occurs *and* the "freeze counters on enabled condition or event" bit is '1' (MMCR0[FCECE] = '1').
33	FCS	Freeze counters when in supervisor state. 0 The PMCs are incremented. 1 The PMCs are not incremented in supervisor state (MSR[PR] = '0').
34	FCP	Freeze counters when in user (problem) state. 0 The PMCs are incremented. 1 The PMCs are not incremented in user (problem) state (MSR[PR] ='1').
35	FCM1	Freeze counters when performance monitor mark bit (MSR[PMM]) is set to '1'. 0 The PMCs are incremented. 1 The PMCs are not incremented when the MSR mark bit is '1' (MSR[PMM] = '1').

Table 10-2. Performance Monitor Control Register MMCR0

Bits	Field Name	Description
36	FCM0	Freeze counters when performance monitor mark bit (MSR[PMM]) is set to ‘0’. 0 The PMCs are incremented. 1 The PMCs are not incremented when the MSR mark bit is ‘0’ (MSR[PMM = ‘0’).
37	PMXE	Performance monitor exception enable. 0 Performance monitor exceptions are disabled. 1 Performance monitor exceptions are enabled until a performance monitor exception occurs, at which time the hardware disables the performance monitor exception (MMRC0[PXME] is set to ‘0’). For implementations that do not provide a performance monitor exception, software can set PXME to ‘1’ and then poll the bit to determine whether an enabled condition or event has occurred.
38	FCECE	Freeze counters on enabled condition or event. 0 The PMCs are incremented. 1 The PMCs are incremented until detection of an enabled counter negative condition *or* detection of an enabled time-base transition event occurs *and* the trigger bit enables the detected event (MMCR0[TRIGGER] equals ‘0’). At that time the counters are frozen (MMCR0[FC] is set to ‘1’) until the condition is reset by software. If the enabled condition or event occurs when MMCR0[TRIGGER] equals ‘1’, then the FCECE bit is treated as if it were ‘0’.
39:40	TBSEL	Time-base selector. 00 Time-base bit 63 is selected. 01 Time-base bit 55 is selected. 10 Time-base bit 51 is selected. 11 Time-base bit 47 is selected. When the selected time base transitions from ‘0’ to ‘1’ *and* the time-base event is enabled (MMCR0[TBEE] equals ‘1’) *and* the performance monitor exception is enabled, a performance monitor exception occurs and the performance monitor exception is disabled (MMRC0[PXME] is set to ‘0’). In multiprocessor systems with the Time-Base Registers synchronized among the processors, time-base transition events can be used to correlate the performance monitor data obtained by the several processors provided that software has specified the same TBSEL value for all of the processors in the system. The frequency of the time base is implementation dependent, and a system service routine should be invoked to obtain the frequency before a value for TBSEL is chosen.
41	TBEE	Time-base exception enable. 0 Disable time-base transition events. 1 Enable time-base transition events.
42:47	THRESHOLD	Threshold value. When a threshold event is selected, counting occurs only for those of the selected event occurrences whose duration in number of cycles exceeds the value in the THRESHOLD field.
48	PMC1CE	PMC1 count enable. This bit determines whether the counter negative condition due to a negative value in PMC1 is enabled. 0 Disable PMC1 counter negative condition. 1 Enable PMC1 counter negative condition.
49	PMCjCE	PMCj count enable (where j represents any counter from 2 to 8). This bit determines whether the counter negative condition due to a negative value in PMCj ($2 \leq j \leq 8$) is enabled. 0 Disable PMCj ($2 \leq j \leq 8$) counter negative condition. 1 Enable PMCj ($2 \leq j \leq 8$) counter negative condition.
50	TRIGGER	Trigger enable. 0 The PMCs are incremented. 1 PMC1 is incremented. The PMCjs ($2 \leq j \leq 8$) are not incremented until PMC1 is negative *or* an enabled condition or event occurs. At that time, the PMCj counters ($2 \leq j \leq 8$) resume counting and the trigger is disabled (MMCR0[TRIGGER] set equal to ‘0’).

Table 10-2. Performance Monitor Control Register MMCR0

Bits	Field Name	Description
51:55	PMC1SEL	PMC1 event selector. The value in this bit field combined with MMCR1[0:31] determines which event will be counted by PMC1.
56	—	Reserved.
57	—	Reserved.
58:62	PMC2SEL	PMC2 event selector. The value in this bit field combined with MMCR1[0:31] determines which event will be counted by PMC2.
63	FCH	Freeze counters in hypervisor mode.

10.4.2 Performance Monitor Control Register MMCR1

Figure 10-3. Performance Monitor Control Register MMCR1

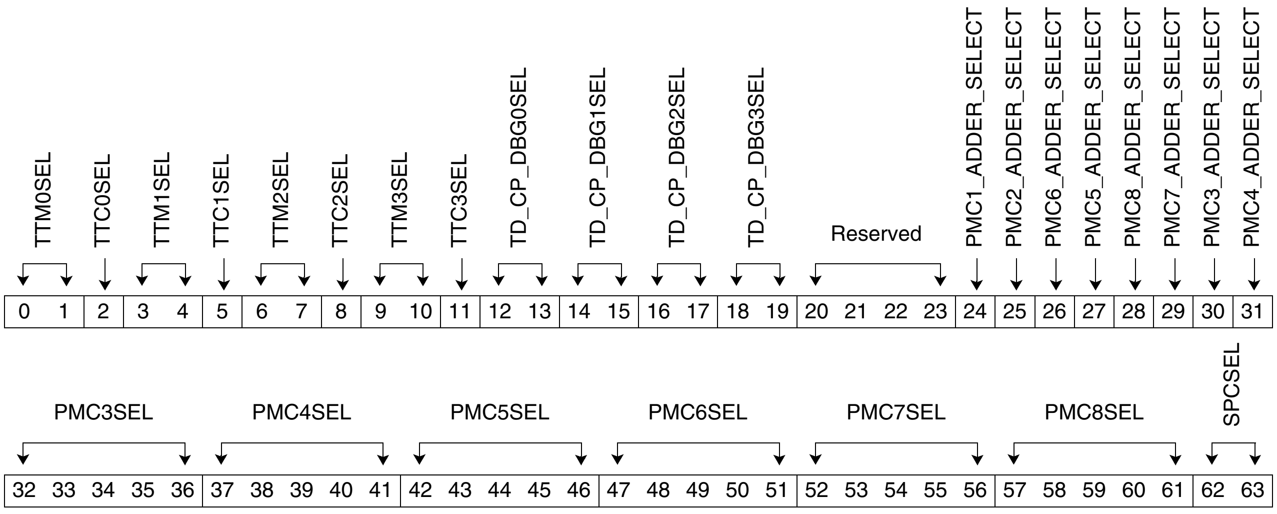


Table 10-3. Performance Monitor Control Register MMCR1

Bits	Field Name	Description
0:1	TTM0SEL	FPU/ISU/IFU/VPU unit select. 00 FPU 01 Instruction sequencer unit (ISU) 10 IFU 11 Vector processing unit (VPU)
2	TTC0SEL	Reserved.
3:4	TTM1SEL	IDU/ISU/STS unit select. 00 IDU 01 Undefined 10 ISU 11 Storage subsystem (STS)
5	TTC1SEL	Reserved.
6:7	TTM2SEL	Reserved.
8	TTC2SEL	Reserved.
9:10	TTM3SEL	Load/store unit 1 (LSU1) select. 0x Lane 2 is LSU1 upper. 1x Lane 2 is LSU1 lower. x0 Lane 3 is LSU1 upper. x1 Lane 3 is LSU1 lower.
11	TTC3SEL	Reserved.

Table 10-3. Performance Monitor Control Register MMCR1

Bits	Field Name	Description
12:13	TD_CP_DBG0SEL	Byte lane 0 unit select. 00 Unit from TTM0. 01 Unit from TTM1. 10 LSU0, byte 0. 11 LSU1, byte 0.
14:15	TD_CP_DBG1SEL	Byte lane 1 unit select. 00 Unit from TTM0. 01 Unit from TTM1. 10 LSU0, byte 1. 11 LSU1, byte 1.
16:17	TD_CP_DBG2SEL	Byte lane 2 unit select. 00 Unit from TTM0. 01 Unit from TTM1. 10 LSU0, byte 2. 11 LSU1, byte 2 or byte 6 (controlled by TTM3SEL[0]).
18:19	TD_CP_DBG3SEL	Byte lane 3 unit select. 00 Unit from TTM0. 01 Unit from TTM1. 10 LSU0, byte 3. 11 LSU1, byte 3 or byte 7 (controlled by TTM3SEL[1]).
20:23	—	Reserved.
24	PMC1_ADDER _SELECT	PMC1 event adder lane select. 0 Byte lane 0: Add 0 + 4. 1 Byte lane 2: Add 0 + 4.
25	PMC2_ADDER _SELECT	PMC2 event adder lane select. 0 Byte lane 0: Add 1 + 5. 1 Byte lane 2: Add 1 + 5.
26	PMC6_ADDER _SELECT	PMC6 event adder lane select. 0 Byte lane 0: Add 2 + 6. 1 Byte lane 2: Add 2 + 6.
27	PMC5_ADDER _SELECT	PMC5 event adder lane select. 0 Byte lane 0: Add 3 + 7. 1 Byte lane 2: Add 3 + 7.
28	PMC8_ADDER _SELECT	PMC8 event adder lane select. 0 Byte lane 1: Add 0 + 4. 1 Byte lane 3: Add 0 + 4.
29	PMC7_ADDER _SELECT	PMC7 event adder lane select. 0 Byte lane 1: Add 1 + 5. 1 Byte lane 3: Add 1 + 5.
30	PMC3_ADDER _SELECT	PMC3 event adder lane select. 0 Byte lane 1: Add 2 + 6. 1 Byte lane 3: Add 2 + 6.
31	PMC4_ADDER _SELECT	PMC4 event adder lane select. 0 Byte lane 1: Add 3 + 7. 1 Byte lane 3: Add 3 + 7.
32:36	PMC3SEL	PMC3 event selector. The value in this bit field combined with MMCR1[0:31] determines which event will be counted by PMC3.

Table 10-3. Performance Monitor Control Register MMCR1

Bits	Field Name	Description
37:41	PMC4SEL	PMC4 event selector. The value in this bit field combined with MMCR1[0:31] determines which event will be counted by PMC4.
42:46	PMC5SEL	PMC5 event selector. The value in this bit field combined with MMCR1[0:31] determines which event will be counted by PMC5.
47:51	PMC6SEL	PMC6 event selector. The value in this bit field combined with MMCR1[0:31] determines which event will be counted by PMC6.
52:56	PMC7SEL	PMC7 event selector. The value in this bit field combined with MMCR1[0:31] determines which event will be counted by PMC7.
57:61	PMC8SEL	PMC8 event selector. The value in this bit field combined with MMCR1[0:31] determines which event will be counted by PMC8.
62:63	SPCSEL	Speculative count event selector. 00 Reserved 01 Event A1x 10 Event A2x 11 Event A3x See *Table 10-20* on page 213 for definitions of the events.

10.4.3 Performance Monitor Control Register MMCRA

Figure 10-4. Performance Monitor Control Register MMCRA

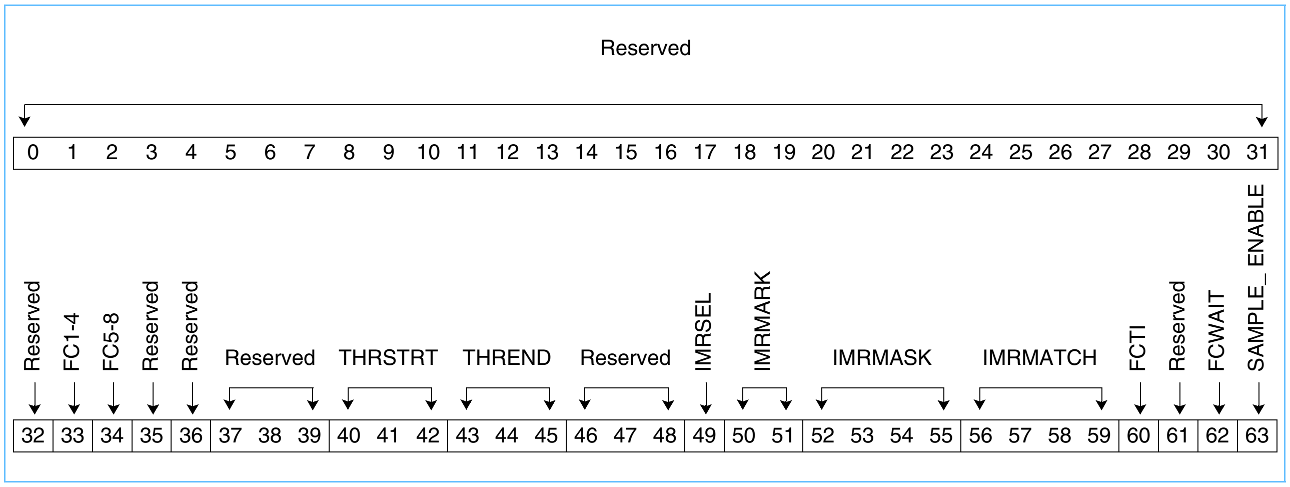


Table 10-4. Performance Monitor Control Register MMCRA

Bits	Field Name	Description
0:31	—	Reserved.
32	—	Reserved.
33	FC1-4	Freeze counters 1 - 4. 0 PMC1 - 4 are incremented. 1 PMC1 - 4 are not incremented.
34	FC5-8	Freeze counters 5 - 8. 0 PMC5 - 8 are incremented. 1 PMC5 - 8 are not incremented.
35	—	Reserved.
36	—	Reserved.
37:39	—	Reserved.
40:42	THRSTRT	Threshold start event.
43:45	THREND	Threshold end event.
46:48	—	Reserved.
49	IMRSEL	Instruction mark (IMR) select. IMR select interacts with IMR mark to determine stage 1 eligibility as described in *Section 10.11 IDU Instruction Sampling Facility* on page 234. 0 Stage 1 eligible instructions are determined through predecode bits from the IFU combined with the IMRMATCH and IMRMASK fields as described in *Section 10.11* on page 234. This is useful if the IMR mark equals '00'. 1 The instruction mark bit (IMR bit) from the IFU IMC match array is used to determine Stage 1 eligibility.

Table 10-4. Performance Monitor Control Register MMCRA

Bits	Field Name	Description
50:51	IMRMARK	IMR Mark. Chooses the mark mode for which instructions are Stage 2 eligible. 00 All Stage 1 eligible internal operations (IOPs) 01 Only Stage 1 eligible IOPs that resulted from microcode expansion 10 Only one IOP per eligible PowerPC instruction 11 First IOP that goes to the LSU for every eligible PowerPC load/store (ld/st) instruction
52:55	IMRMASK	IMR Mask. A mask ANDed with the predecode bits before using the IMRMATCH field.
56:59	IMRMATCH	IMR Match. The value that the result of the IMRMASK ANDed with the predecode bits must match to be Stage 2 eligible. All 4 bits of the result must match the IMRMATCH exactly. To match ALL IOPs (that is, the match will always succeed) set IMRSEL equals '0', IMRMASK equals '0000', and IMRMATCH equals '0000'.
60	FCTI	Freeze Counters. 0 The PMCs are incremented. 1 The PMCs are not incremented.
61	—	Reserved
62	FCWAIT	Freeze Counters in Wait State (implies that CNTL[31] equals '0'). 0 The PMCs are incremented. 1 The PMCs (except those counting cycles) are not incremented when CNTL[31] equals '0'.
63	SAMPLE_ ENABLE	0 Sampling is disabled 1 Sampling is enabled

10.4.4 Performance Monitor Count Registers PMC1 - 8

Figure 10-5. Performance Monitor Count Registers PMC1 - 8

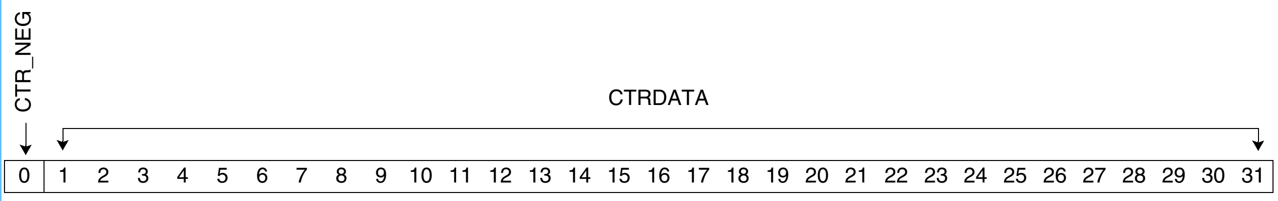


Table 10-5. Performance Monitor Count Registers PMC1 - 8

Bits	Field Name	Description
0	CTR_NEG	Counter negative bit.
1:31	CTRDATA	Count data.

10.4.5 Performance Monitor and Trace Related Bits in the Machine State Register (MSR)

Figure 10-6. Performance Monitor and Trace Related Bits in the MSR

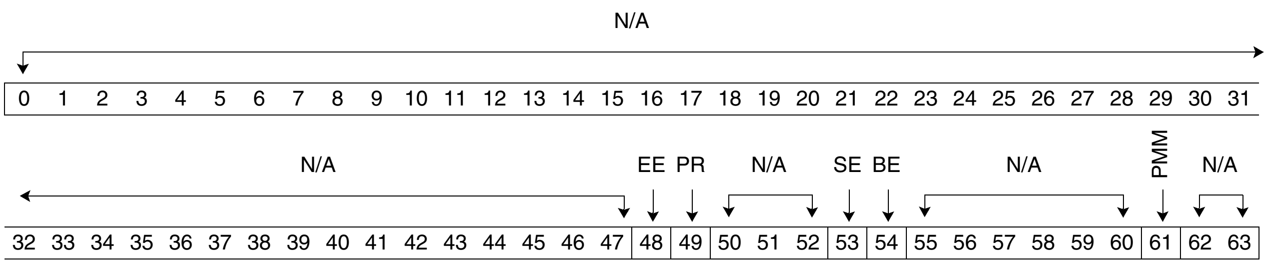


Table 10-6. Performance Monitor and Trace Related Bits in the MSR

Bits	Field Name	Description
0:47	N/A	Not applicable.
48	EE	External interrupt enable. 0 The processor is disabled for external, decrementer, and performance monitor exceptions. 1 The processor is enabled for external, decrementer, and performance monitor exceptions.
49	PR	Problem (user) state. 0 The processor is privileged to execute any instruction. 1 The processor can execute only non-privileged instructions.
50:52	N/A	Not applicable.
53	SE	Single step trace enable. 0 The processor does not generate a trace exception after instruction completion. 1 The processor generates a trace exception after successfully completing the execution of the next instruction unless that instruction is an Return from Exception Doubleword (**rfid**), which is never traced.
54	BE	Branch trace enable. 0 The processor does not generate a trace exception after branch instruction completion. 1 The processor generates a trace exception after successfully completing the execution of a branch instruction whether or not the branch is taken.
55:60	N/A	Not applicable.
61	PMM	Performance monitor mode enable. 0 The currently executing process is not marked. 1 The currently executing process is marked. This bit is used to mark a process for the performance monitor. Several performance monitor MMCR0 control bits can then be set to enable counting based on the value of the PMM bit. When an exception occurs, this bit is saved, set to '0' for the duration of the exception processing, and then restored when the **rfid** instruction is executed. If this bit is changed with an **mtmsr** or Move to Machine State Register Doubleword (**mtmsrd**) instruction, the change is not guaranteed to have taken effect until after a subsequent context-synchronizing instruction has completed execution.
62:63	N/A	Not applicable.

10.4.6 Performance Monitor Related Bits in Hardware Implementation-Dependent Register 0 (HID0)

Figure 10-7. Performance Monitor Related Bits in HID0

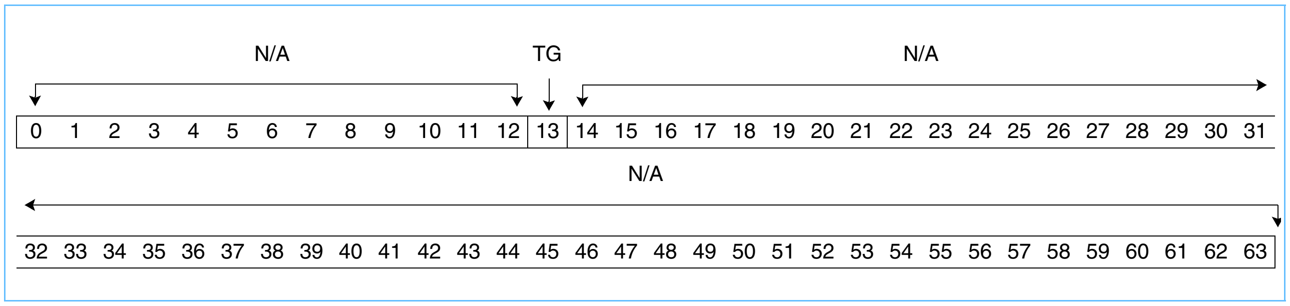


Table 10-7. Performance Monitor Related Bits in HID0

Bits	Field Name	Description
0:12	N/A	Not applicable.
13	TG	Performance monitor threshold granularity. 0 The thresholder counts every processor cycle. 1 The thresholder counts every 32 processor cycles.
14:63	N/A	Not applicable.

10.4.7 Performance Monitor Related Bits in the Control Register (CTRL)

Figure 10-8. Performance Monitor Related Bits in the CTRL

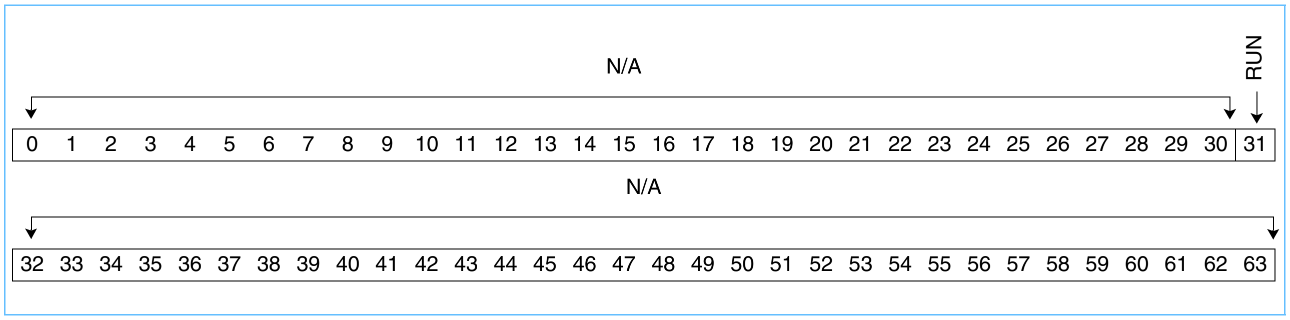


Table 10-8. Performance Monitor Related Bits in the CTRL

Bits	Field Name	Description
0:30	N/A	Not applicable.
31	RUN	Wait state bit.
32:63	N/A	Not applicable.

10.4.8 Performance Monitor Related Bits in the SCOM0240, 1240 Register (SCOM x‘240’)

Figure 10-9. Performance Monitor Related Bits in the SCOM x‘240’ Register

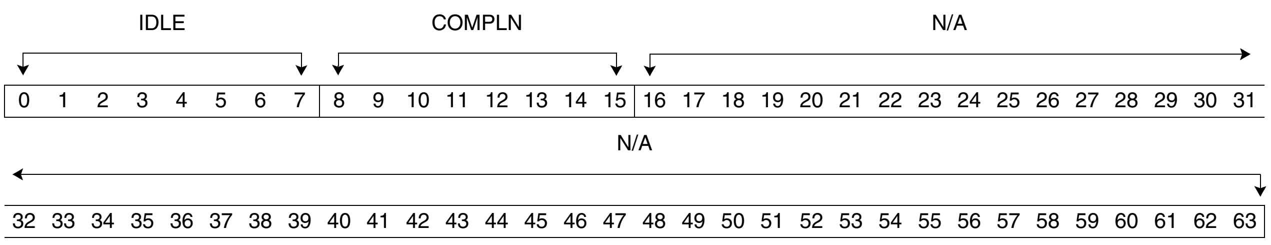


Table 10-9. Performance Monitor Related Bits in the SCOM x‘240’ Register

Bits	Field Name	Description
0:7	IDLE	Sampling logic idle delay.
8:15	COMPLN	Sampling logic completion delay.
16:63	N/A	Not applicable.

10.4.9 Performance Monitor Related Bits in the SCOM0360,1360 Register (SCOM x'360')

Figure 10-10. Performance Monitor Related Bits in the SCOM x'360'

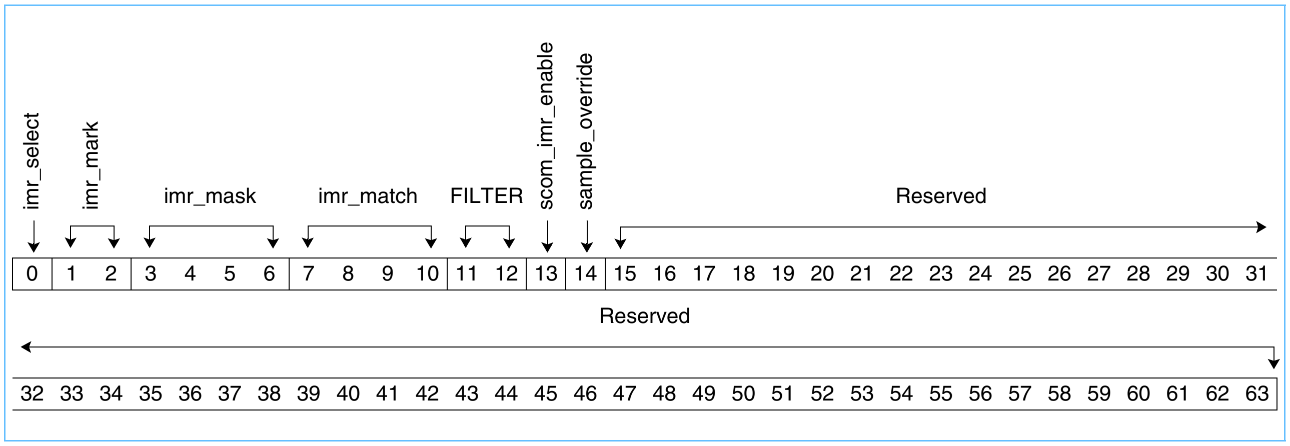


Table 10-10. Performance Monitor Related Bits in the SCOM x'360'

Bits	Field Name	Description
0	imr_select	Same as MMCRA[imr_sel].
1:2	imr_mark	Same as MMCRA[imr_mark] and overrides MMCRA if bit 13 equals '1'.
3:6	imr_mask	Same as MMCRA[imr_mask] and overrides MMCRA if bit 13 equals '1'.
7:10	imr_match	Same as MMCRA[imr_match] and overrides MMCRA if bit 13 equals '1'.
11:12	FILTER	IMR filter random/all and first/all. These two bits form a 2-step filtering operation on the eligible bits associated with the instructions in the group. Bit 11 first determines whether instruction eligibility bits pass the first filter step based on either a random pass/nopass (bit 11 equals '1') choice or an all pass (bit 11 equals '0') choice for each instruction. Bit 12 determines how microcoded instructions are sampled (and has no effect on non-microcoded instructions): 00 No filtering (OR) 01 No filtering (AND) 10 Use Good_Address mode of sampling microcode expansions. 11 Use More_Hits mode of sampling microcode expansions. In Good_Address mode, there is at most one IOP in any microcode expansion that is eligible for sampling. This is (a) the first load/store IOP if there are any load/store IOPs in the expansion, or (b) the first IOP in the final group of the expansion. If the random filter suppresses marking this IOP, then no IOP will be marked for the microcode expansion. In More_Hits mode, multiple IOPs in a microcode expansion are eligible for sampling. These are (a) the first load/store IOP in any group, or (b) the first IOP of the final group. If the random filter suppresses marking the first of these IOPs, a subsequent one might still be sampled. (However, at most one will be marked in a single microcode expansion.)
13	scom_imr_enable	0 Performance monitor fields are used for mark, mask, match. 1 SCOM fields are used for mark, mask, match.
14	sample_override	0 Performance monitor "ok_to_sample" indication is used. 1 Overrides performance monitor "ok_to_sample" indication.
15:63	—	Reserved.

10.4.10 Performance Monitor Related Bits in the IMC Array (IMC)

Figure 10-11. Performance Monitor Related Bits in the IMC

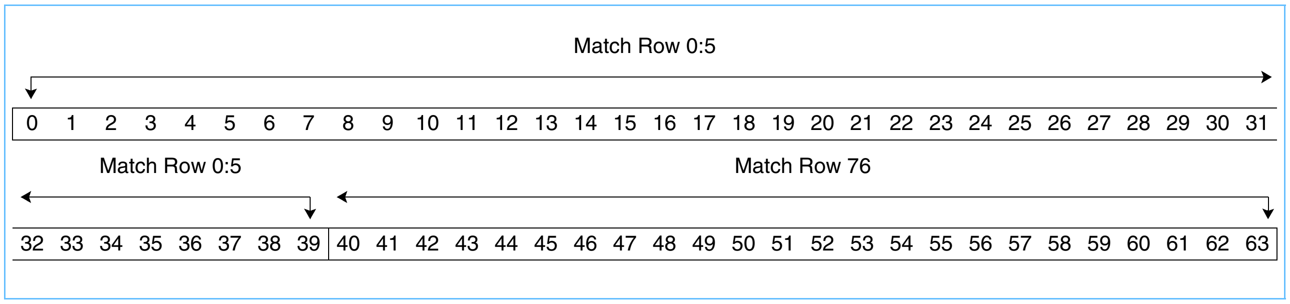


Table 10-11. Performance Monitor Related Bits in the IMC

Bits	Field Name	Bit Description
0:39	Match Row 0	Opcode/extended opcode match.
0:39	Match Row 1	Opcode/extended opcode match.
0:39	Match Row 2	Opcode/extended opcode match.
0:39	Match Row 3	Opcode/extended opcode match.
0:39	Match Row 4	Opcode/extended opcode match.
0:39	Match Row 5	Opcode/extended opcode match.
0:63	Match Row 76	Full instruction match.

10.4.11 Performance Monitor Related Bits in the Sampled Instruction Address Register (SIAR)

Figure 10-12. Performance Monitor Related Bits in the SIAR

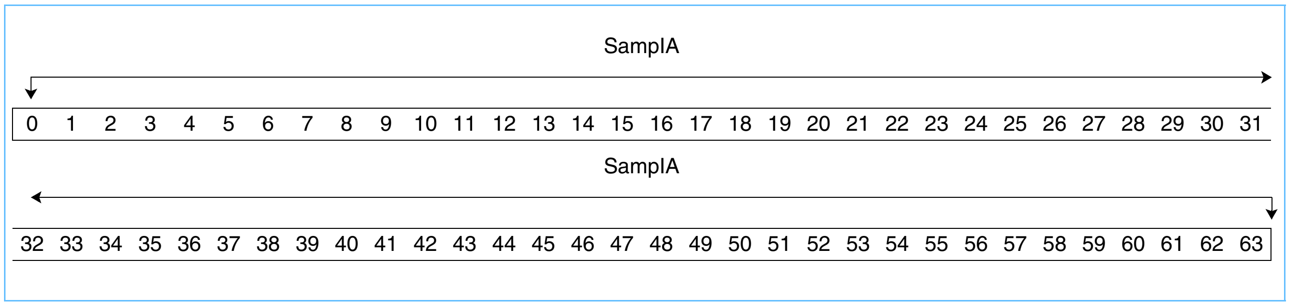


Table 10-12. Performance Monitor Related Bits in the SIAR

Bits	Field Name	Description
0:63	SamplA	Sampled instruction address.

10.4.12 Performance Monitor Related Bits in the Sampled Data Address Register (SDAR)

Figure 10-13. Performance Monitor Related Bits in the SDAR

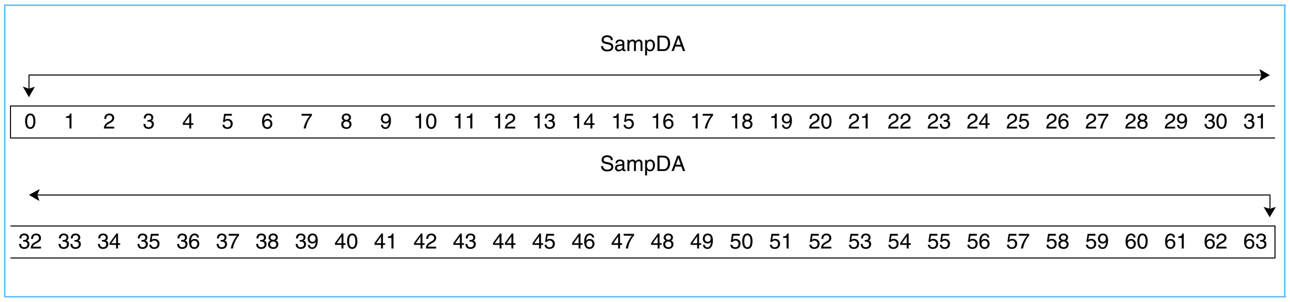


Table 10-13. Performance Monitor Related Bits in the SDAR

Bits	Field Name	Description
0:63	SampDA	Sampled Data Address

10.4.13 Performance Monitor Related Bits in the SRR1 (SRR1)

Figure 10-14. Performance Monitor Related Bits in the SRR1

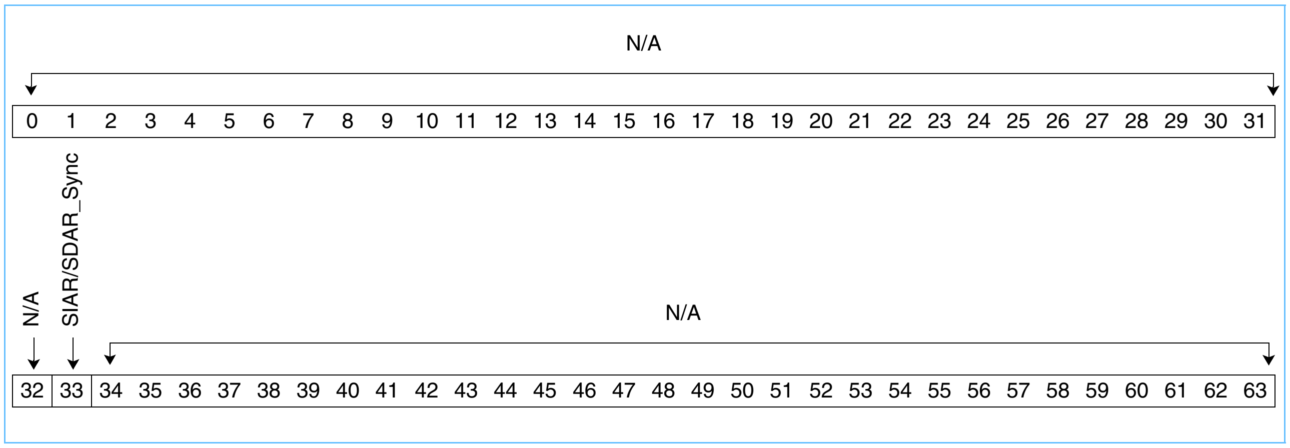


Table 10-14. Performance Monitor Related Bits in the SRR1

Bits	Field Name	Description
0:32	N/A	Not applicable.
33	SIAR/SDAR_Sync	SIAR and SDAR contents synchronized.
34:63	N/A	Not applicable.

10.4.14 Performance Monitor Related Bits in the Time-Base Register (TB)

Figure 10-15. Performance Monitor Related Bits in the TB Register

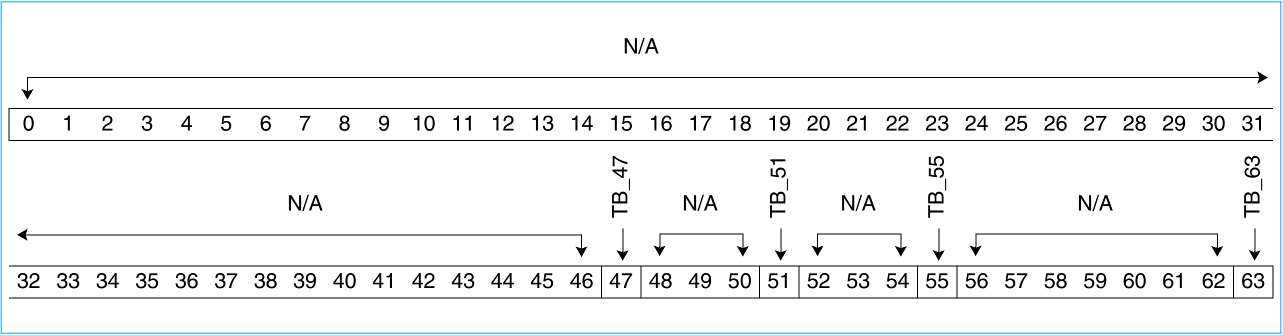


Table 10-15. Performance Monitor Related Bits in the TB Register

Bits	Field Name	Description
0:46	N/A	Not applicable.
47	TB_47	Time-Base Register bit [47].
48:50	N/A	Not applicable.
51	TB_51	Time-Base Register bit [51].
52:54	N/A	Not applicable.
55	TB_55	Time-Base Register bit [55].
56:62	N/A	Not applicable.
63	TB_63	Time-Base Register bit [63].

10.5 Performance Monitor Event Selection

Event signals are routed from the functional units through the processor performance monitor buses. These signals are multiplexed and divided into byte lanes 0 - 3. A smaller number of events are routed directly to the performance monitor unit (PMU); these events are referred to as direct events. Each PMC can be configured to count a subset of the direct events or one of two possible byte lanes. Counters 1, 2, 5, and 6 can be configured to count events on byte lane 0 or 2, counters 3, 4, 7, and 8 can be configured to count events on byte lane 1 or 3. The selection of event source (direct, byte lane) is controlled by the PMCxSEL field in MMCR1 (where x is the PMC number). *Figure 10-16* shows this selection. *Table 10-16* shows how the PMCxSEL field is used to select which events are monitored.

Performance monitor events fall into three categories:

- Direct: All the information is hardwired to the PMU.
- Bus: All the information is routed over the hierarchical event bus.
- Combined: Some information comes from the event bus; the PMU does additional processing on it.

Figure 10-16. Event Selection

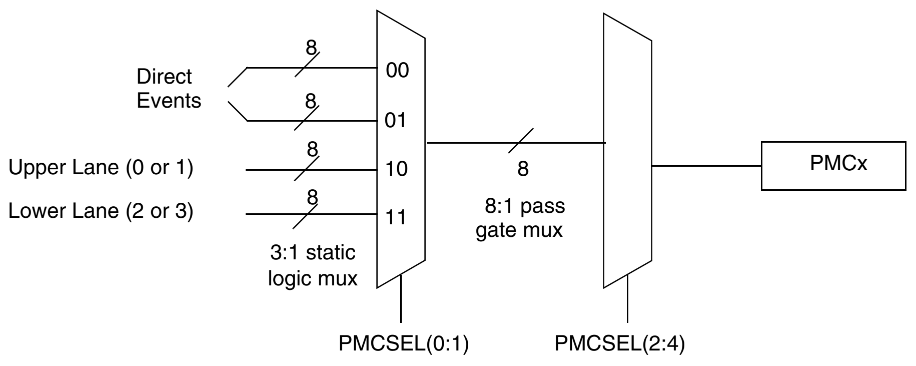


Table 10-16. Performance Monitor Internal Multiplexer PMCxSEL[0:4] Bit Values

PMCSEL[0:1]	PMCSEL[2:4]	Counted Event
00	000-111	Direct Events
10	000	None. When count_en is '0', turn off counter.
10	111	Cycles
01	000-111	Direct Events
10	000-111	Select smaller byte lane
11	000-111	Select larger byte lane

10.5.1 Direct Events

As shown in *Table 10-16*, direct events are selected with PMCxSEL[0:1] set to '0x'. When PMCxSEL[0:1] equals '10' and PMCxSEL[2:4] equals '111', the counter is configured to count cycles. When PMCxSEL[0:1] equals '10' and PMCxSEL[2:4] is '000', the counter is off (counts nothing). The direct events that can be counted are shown in *Table 10-19* on page 210.

Some direct events, such as events that add two other events or interpret the memory source encodes for data or instruction fetches, also require data from the performance monitor events. Although they are listed in *Table 10-19 Direct Events*, they rely on a meaningful configuration of the performance monitor event selections to produce meaningful results.

10.5.1.1 Combined Events

Each PMC can add similar events to produce a single, combined count. For example, each load store unit provides a data-cache miss event, which can be added to produce the total data-cache miss count. The added events are considered direct events, but they rely on the performance monitor bus being configured properly to produce meaningful results. Since each PMC can receive event signals from two byte lanes on the performance monitor bus, the added events can be configured to add events on one of the two byte lanes. Events cannot be added from different byte lanes. The PMCx_ADDER_SELECT fields in MMCR1 control which byte lanes are used.

10.5.1.2 Source-Encoded Events

Source-encoded events (direct event 7 [PMCxSEL equals '00111'] for data and event 6 [SEL equals '00110'] for instructions) are combined events that count events from a specific source as shown in *Table 10-17* and *Table 10-18* on page 209.

Note: Intervention event sources are only meaningful on multiprocessor systems.

Table 10-17. Event Data Source Encodings

Encoding (0:3)	Event Source
0000	L2 cache
0010	Memory
1000	Shared Intervention (another L2 cache)
1010	Modified Intervention (another L2 cache)
All Others	Reserved

To count data source-encoded events, the performance monitor event bus must be configured as follows:

1. Route LSU1 byte 3 data to the PMU (the "L1 reload data source" LSU1 indirect event) by setting the TD_CP_DBG3SEL field in MMCR1 to '11'.
2. Select the direct event that decodes the desired data source. To count L1 data reloads from the L2, for example, PMC1, direct event 7 (the PMC1SEL field in MMCR0 set to '00111') should be used.

Table 10-18. Event Instruction Source Encodings

Encoding (0:3)	Event Source
1001	I-cache
1010	Prefetch buffer
0000	L2 cache
0001	Memory
1111 or 1011	No instructions on bus

To count instruction source-encoded events, the performance monitor event bus should be similarly configured:

1. Route IFU byte 2 data to the PMU (the "iL1 cache data source" IFU indirect event) by setting the TD_CP_DBG2SEL field in MMCR1 to '00' and TTM0SEL to '10'.
2. Select the direct event that decodes the desired data source. To count L1 instruction reloads from memory, for example, PMC3, direct event 6 (the PMC3SEL field in MMCR0 set to '00110') should be used.

10.5.1.3 Instruction Counts

Two types of instruction and IOP counting are available with the 970FX performance monitor:

- Direct event 1 (SEL equals '00001') on PMC1, PMC4, PMC6, PMC7, and PMC8 counts instructions according to the IMRMARK field of the MMCRA Register:
  - 00 All stage 1 eligible IOPs
  - 01 Stage 1 eligible IOPs from microcode expansion
  - 10 One IOP per eligible PowerPC instruction
  - 11 First IOP to LSU per eligible PowerPC load/store instruction
- Direct event 9 (SEL equals '01001') on PMC1 - PMC8 always counts PowerPC instructions independent of the IMRMARK field of the MMCRA Register (see *Table 10-19* on page 210).

Table 10-19. Direct Events (Page 1 of 2)

SEL(0:4)	PMC1	PMC2	PMC3	PMC4	PMC5	PMC6	PMC7	PMC8
00 000 plus	Add 0 + 4	Add 1 + 5	Add 2 + 6	Add 3 + 7	Add 3 + 7	Add 2 + 6	Add 1 + 5	Add 0 + 4
MMCR1(24:31) = '0', '1'	MMCR1[24] = '0' byte lane 0	MMCR1[25] = '0' byte lane 0	MMCR1[30] = '0' byte lane 1	MMCR1[31] = '0' byte lane 1	MMCR1[27] = '0' byte lane 0	MMCR1[26] = '0' byte lane 0	MMCR1[29] = '0' byte lane 1	MMCR1[28] = '0' byte lane 1
	MMCR1[24] = '1' byte lane 2	MMCR1[25] = '1' byte lane 2	MMCR1[30] = '1' byte lane 3	MMCR1[31] = '1' byte lane 3	MMCR1[27] = '1' byte lane 2	MMCR1[26] = '1' byte lane 2	MMCR1[29] = '1' byte lane 3	MMCR1[28] = '1' byte lane 3
00 001	number of instructions complete	work held	stop completion	number of instructions complete	dispatch_success	number of instructions complete	number of instructions complete	number of instructions complete
00 010	marked group dispatch	LSU empty (load miss queue [LMQ] and store reorder queue [SRQ] empty)	LSU empty (LMQ and SRQ empty)	Fixed-point unit 0 (FXU0) idle and FXU1 busy	FXU0 idle and FXU1 idle	FXU0 busy and FXU1 busy	FXU0 busy and FXU1 idle	external interrupt
00 011	marked store complete	threshold timeout event	marked store with interrupt complete	SRQ empty	one or more PowerPC instructions completed	marked store sent to STS	group completed	group dispatch reject
00 100	global completion table (GCT) empty	group dispatch	cycles in supervisor mode	marked group complete	group marked in IDU	FXU marked instr finish	FPU marked instr finish	LSU marked instr finish
00 101	run_cycles; that is, # cycles when CNTL[31] = '1'	branch unit (BRU) marked instr finish	VPU marked instr finish	condition register unit (CRU) marked instr finish	marked group complete time out	marked group issued	marked instr finish any unit	time base event
00 110	Inst src encode 0000	Inst src encode 0001	Inst src encode 0010	Inst src encode 0011	Inst src encode 0100	Inst src encode 0101	Inst src encode 0110	Inst src encode 0111
00 111	Data src encode 0000	Data src encode 0001	Data src encode 0010	Data src encode 0011	Data src encode 0100	Data src encode 0101	Data src encode 0110	Data src encode 0111
01 000	Counter OFF	Counter OFF	Counter OFF	Counter OFF	Counter OFF	Counter OFF	Counter OFF	Counter OFF
01 001	number of instructions complete	number of instructions complete	number of instructions complete	number of instructions complete	number of instructions complete	number of instructions complete)	number of instructions complete	number of instructions complete
01 010	Overflow from counter 8	Overflow from counter 1	Overflow from counter 2	Overflow from counter 3	Overflow from counter 4	Overflow from counter 5	Overflow from counter 6	Overflow from counter 7
01 011	reserved	GCT empty by SRQ full	reserved	reserved	—/A1a/A2a/A3a (*1) (See *Table 10-20* on page 213)	reserved	—/A1b/A2b/A3b (*1) (See *Table 10-20* on page 213)	reserved
01 100	reserved	reserved	reserved	reserved	—/A1c/A2c/— (*1) (See *Table 10-20* on page 213)	reserved	—/A1d/A2d/—(*1) (See *Table 10-20* on page 213)	reserved

Table 10-19. Direct Events (Page 2 of 2)

SEL(0:4)	PMC1	PMC2	PMC3	PMC4	PMC5	PMC6	PMC7	PMC8
01 101	Instruction source decode 1000	Instruction source encode 1001	Instruction source encode 1010	Instruction source encode 1011	Instruction source encode 1100	Instruction source encode 1101	Instruction source encode 1110	Instruction source encode 1111
01 110	Byte 3 decode 1000	Data source encode 1001	Data source encode 1010	Data source encode 1011	Data source encode 1100	Data source encode 1101	Data source encode 1110	Data source encode 1111
01 111	Cycles	Cycles	Cycles	Cycles	Cycles	Cycles	Cycles	Cycles

10.5.2 Over 32-Bit Count

The 970FX PMU can chain together multiple 32-bit PMCs to create up to a 256-bit wide PMC register when used in conjunction with overflow counting. This is useful for performance measurement on high clock-rate machines. The maximum count value depends on the following settings:

- PMCn can count over 32-bits when PMC*n+1*SEL(0:4)(where *n* is 1 - 7) is set to '01010'.
- PMC8 can count over 32-bits when PMC1SEL(0:4) is set to '01010'.
- When PMCn + 1 uses this overflow counting function, PMCn is prohibited from asserting an exception signal when a negative condition occurs (PMC1CE(PMCjCE) equals '1' and PMCn[0] is '1').

10.5.2.1 Examples of Over Bit Count

Example 1

When PMC1 is set to '00100' (GCT empty) and PMC2 is set to '01010' (overflow function), then PMC2 works as the upper 32 bits of PMC1. In this case, the overflow exception is only asserted by PMC2 (never by PMC1) when PMCjCE equals '1' (don't care PMC1CE) and PMC2[0] is '1'.

Example 2

When PMC8 is set to '00001' (number of instructions complete) and PMC1 are set to '01010' (overflow function), then PMC1 functions as the upper 32 bits of PMC8. In this case, the overflow exception is only asserted by PMC1 (never by PMC8) when PMC1CE equals '1' (don't care PMCjCE) and PMC1[0] is '1'.

Example 3

When PMC1 is set to '00100' (GCT empty) and PMC2, PMC3, and PMC4 is set to '01010' (overflow function), then PMC4, PMC3, and PMC2 function as the upper 96 bits of PMC1. In this case, the overflow exception is only asserted by PMC4 (never by PMC1, PMC2, or PMC3) when PMCjCE equals '1' (don't care PMC1CE) and PMC4[0] is '1'.

10.5.3 Speculative Count

PMC5 and PMC7 support the speculative count function with a backup register. This is enabled when MMCR1[62:63] is set to '01', '10', or '11' and a speculative event is selected (PMC[5,7]SEL equals '01011' or '01100'). The PMC starts counting speculatively whenever a next-to-complete (NTC) group completion stops (or GCT empty happens). The PMC then stores the counts to itself and its backup register if the last finished event matches what the PMC initially set up. If there is no match, the PMC restores the old count value from the backup register. This allows the PMU to establish a cycles per instruction (CPI) breakdown for various categories (CPI contribution due to an instruction-cache [I-cache] miss, data-cache [D-cache] miss, LSU, FXU, FPU, and so on).

A negative condition exception only occurs when the count value is not speculative and a negative condition occurs. (When PMC1CE[PMCjCE] is set to '1' and the backup register's negative bit is '1'.)

Table 10-20 lists the speculative count events.

Table 10-20. Speculative Count Events

PMC Number		SEL(0:4)	MMCR1 Condition		Count Events	See Note
			bit 62	bit 63		
	5, 7	01011	0	0	Reserved	
A1a	5	01011	0	1	Completion stall by LSU instruction	
A2a	5	01011	1	0	Completion stall by FXU instruction	
A3a	5	01011	1	1	Completion stall by D-cache miss	
A1b	7	01011	0	1	Completion stall by FPU instruction	
A2b	7	01011	1	0	Completion stall by FXU long instruction	
A3b	7	01011	1	1	Completion stall by reject	
	5, 7	01100	0	0	Reserved	
A1c	5	01100	0	1	Completion stall by FPU long instruction	
A2c	5	01100	1	0	GCT empty by I-cache miss	1
A1d	7	01100	0	1	Completion stall by reject (ERAT miss)	
A2d	7	01100	1	0	GCT empty by branch miss predict	1
	5, 7	01100	1	1	Reserved	

Note:

1. This count event also requires MMCR1 bits. They should be set as follows:
   Bit 1 = '1' and bits 0, 16, and 17 = '0'

or

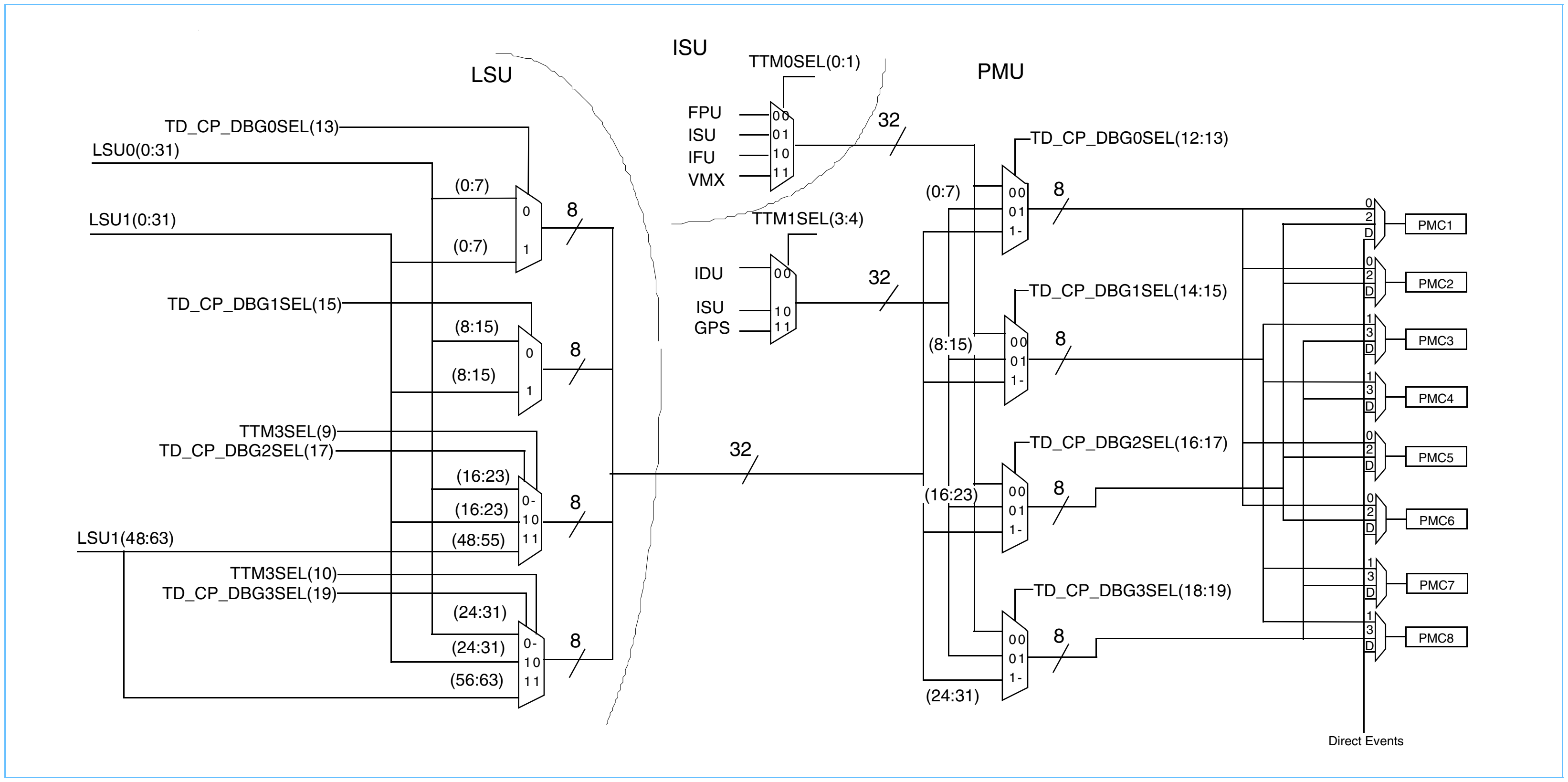
   Bits 3 and 17 = '1' and bits 4 and 16 = '0'

Speculative events are also able to count in the over 32-bit count mode. In this case, the overflow value is carried to the upper PMC only when the counting value is not speculative. For this reason, the upper PMC does not require a backup register to copy and restore the count value.

10.6 Configuring the Performance Monitor Bus

The 970FX performance monitor bus is configured through a series of hierarchal multiplexers, as shown in *Figure 10-17* on page 214. This diagram also shows that all unit event buses are 32 bits, except for the LSU1 that sources an extra 16 bits, denoted as LSU1[48:63]. The LSU0 and LSU1 event buses are multiplexed into a single 32-bit LSU event bus, using the multiplexers shown at the left of *Figure 10-17*. Basically, the multiplexers, on a byte basis, select either the LSU0 or the LSU1 events. The extra LSU1 events are selected using MMCR1[9:10], the TTM3SEL select signals.

Figure 10-17. 970FX Performance Monitor Bus Configuration



The rest of the first level of selection is controlled by the TTM0 and TTM1 multiplexers. These control from which unit the non-LSU event signals are selected. The ISU can be selected by more than one multiplexer (TTM0 and TTM1). The TTM multiplexers are controlled by the TTMxSEL fields in MMCR1. MMCR1[0:1] control TTM0 and MMCR1[3:4] control TTM1.

The three 32-bit outputs of the LSU, TTM0, and TTM1 multiplexers are sent to the TM_DEBUG multiplexers, which are controlled by the 2-bit TD_CP_DBGxSEL fields in the MMCR1; bits [12:13] control TM_DEBUG0, bits [14:15] control TM_DEBUG1, bits [16:17] control TM_DEBUG2, and bits [18:19] control TM_DEBUG3.

After the performance monitor bus is configured, individual events can be selected, as described at the beginning of this section. *Table 10-21* shows the events available through the performance monitor bus and the TTM and TM_DEBUG multiplexer used to select them.

Table 10-21. Performance Monitor Bus Assignments (Page 1 of 8)

	Bits [0:31]	Byte Lane	Bits [0:7]	Event Description
FPU: TTM0 = ‘00’, TD_CP_DBGxSEL = ‘00’				
	0	0	0	FPU0 divide
	1	0	1	FPU0 mult-add
	2	0	2	FPU0 square root
	3	0	3	FPU0 add, mult, sub, compare, fsel
	4	0	4	FPU1 divide
	5	0	5	FPU1 mult-add
	6	0	6	FPU1 square root
	7	0	7	FPU1 add, mult, sub, compare, fsel
	8	1	0	FPU0 move, estimate
	9	1	1	FPU0 round, convert
	10	1	2	FPU0 estimate
	11	1	3	FPU0 finished and produced a result
	12	1	4	FPU1 move, estimate
	13	1	5	FPU1 round, convert
	14	1	6	FPU1 estimate
	15	1	7	FPU1 finished and produced a result

Table 10-21. Performance Monitor Bus Assignments (Page 2 of 8)

	Bits [0:31]	Byte Lane	Bits [0:7]	Event Description
	16	2	0	FPU0 denormalized operand
	17	2	1	FPU0 stall 3
	18	2	2	FPU0 store
	19	2	3	FPU0 single precision
	20	2	4	FPU1 denormalized operand
	21	2	5	FPU1 stall 3
	22	2	6	FPU1 store
	23	2	7	FPU1 single precision
	24	3	0	Floating-Point Status and Control Register (FPSCR)
	25	3	1	FPU0 multiply
	26	3	2	FPU0 compare
	27	3	3	FPU0 select
	28	3	4	FPU1 multiply
	29	3	5	FPU1 compare
	30	3	6	FPU1 select
	31	3	7	Floating-point stall store
IFU:	TTM0 = '10', TD_CP_DBGxSEL = '00'			
	0:15	0:1	0:7	Nothing
	16:19	2	0:3	L1 I-cache data source
	20	2	4	Valid instruction available
	21	2	5	Cycles IFU is held by pipeline hold
	22	2	6	Instruction prefetch installed in prefetch buffer
	23	2	7	L2 prefetch request
	24	3	0	I-ERAT write
	25	3	1	Branch execution issue valid
	26	3	2	Branch miss predict due to Condition Register (CR) value
	27	3	3	Branch miss predict due to target address predict
	28	3	4	Cycles L1 I-cache write active
	29:31	3	5:7	Nothing

Table 10-21. Performance Monitor Bus Assignments (Page 3 of 8)

	Bits [0:31]	Byte Lane	Bits [0:7]	Event Description
IDU: TTM1 = '00', TD_CP_DBGxSEL = '01'				
	0	0	0	Instruction queue has three slots full
	1	0	1	Instruction queue has one slots full
	2	0	2	Instruction queue has six slots full
	3	0	3	Instruction queue has zero slots full
	4	0	4	Instruction queue has four slots full
	5	0	5	Instruction queue has two slots full
	6	0	6	Instruction queue has seven slots full
	7	0	7	Instruction queue has eight slots full
	8	1	0	Instruction queue has five slots full
	9:15	1	1:7	Instruction queue full
	16:31	2:3	0:7	Nothing
LSU0: (See *Figure 10-17 970FX Performance Monitor Bus Configuration* on page 214) TD_CP_DBGxSEL = '1x'				
	0	0	0	Instruction translation lookaside buffer (TLB) miss
	1	0	1	Instruction segment lookaside buffer (SLB) miss
	2	0	2	Data ERAT (D-ERAT) miss side 0
	3	0	3	Snoop TLB Invalidate Entry (**tlbie**)
	4	0	4	Data TLB miss
	5	0	5	Data SLB miss
	6	0	6	D-ERAT miss side 1
	7	0	7	Tablewalk duration
	8	1	0	Marked flush unaligned load side 0
	9	1	1	Marked flush unaligned store side 0
	10	1	2	Marked flush from load reorder queue (LRQ) store-hit-load (SHL), load-hit-load (LHL) side 0
	11	1	3	Marked flush SRQ load-hit-store (LHS) side 0
	12	1	4	Marked flush unaligned load side 1
	13	1	5	Marked flush unaligned store side 1
	14	1	6	Marked flush from LRQ store-hit-load (shl), load-hit-load (lhl) side 1
	15	1	7	Marked flush SRQ LHS side 1

Table 10-21. Performance Monitor Bus Assignments (Page 4 of 8)

	Bits [0:31]	Byte Lane	Bits [0:7]	Event Description
	16	2	0	Marked L1 D-cache load miss side 0
	17	2	1	Store conditional (**stcx**) failed
	18	2	2	Marked IMR reload
	19	2	3	Marked L1 D-cache store miss
	20	2	4	Marked L1 D-cache load miss side 1
	21	2	5	**stcx** passed
	22	2	6	Marked **stcx** fail
	23	2	7	Load and reserve indexed (**larx**) executed 0
	24	3	0	Floating-point load side 0
	25	3	1	L1 cache prefetch request
	26	3	2	Out of streams
	27	3	3	L2 cache prefetch
	28	3	4	Floating-point load side 1
	29	3	5	VPU type L2 prefetch
	30	3	6	Data stream touch (DST) stream start
	31	3	7	New stream allocated
LSU1: (See *Figure 10-17 970FX Performance Monitor Bus Configuration* on page 214) TD_CP_DBGxSEL = '1x'				
	0	0	0	Flush unaligned load side 0
	1	0	1	Flush unaligned store side 0
	2	0	2	Flush from LRQ SHL, LHL side 0
	3	0	3	Flush SRQ LHS side 0
	4	0	4	Flush unaligned load side 1
	5	0	5	Flush unaligned store side 1
	6	0	6	Flush from LRQ SHL, LHL side 1
	7	0	7	Flush SRQ LHS side 1
	8	1	0	L1 D-cache load side 0
	9	1	1	L1 D-cache store side 0
	10	1	2	L1 D-cache load miss side 0
	11	1	3	L1 D-cache store miss
	12	1	4	L1 D-cache load side 1
	13	1	5	L1 D-cache store side 1
	14	1	6	L1 D-cache load miss side 1
	15	1	7	L1 D-cache entries invalidated from L2

Table 10-21. Performance Monitor Bus Assignments (Page 5 of 8)

	Bits [0:31]	Byte Lane	Bits [0:7]	Event Description
	16	2	0	SRQ store forwarding side 0
	17	2	1	SRQ slot 0 valid
	18	2	2	LRQ slot 0 valid
	19	2	3	LSU0 reject
	20	2	4	SRQ store forwarding side 1
	21	2	5	SRQ slot 0 allocated
	22	2	6	LRQ slot 0 allocated
	23	2	7	LSU1 reject
	24:27	3	0:3	L1 cache reload data source
	28	3	4	L1 cache reload data valid
	29	3	5	LMQ slot 0 valid
	30	3	6	LMQ slot 0 allocated
	31	3	7	LMQ full
	32:47	0:1	0:7	Nothing
	48	2	0	SRQ reject 0 - load hit store
	49	2	1	LMQ reject 0 - LMQ full or missed data coming
	50	2	2	LSU0 reject - reload critical data forward (CDF) or tag update collision
	51	2	3	LSU0 reject - ERAT miss
	52	2	4	SRQ reject 1- load hit store
	53	2	5	LMQ reject 1- LMQ full or missed data coming
	54	2	6	LSU1 reject - reload CDF or tag update collision
	55	2	7	LSU1 reject - ERAT miss
	56:58	3	0:3	L1 cache reload data source
	60	3	4	Marked L1 cache reload data source valid
	61	3	5	LMQ load-hit-reload merge
	62	3	6	Marked SRQ valid
	63	3	7	Nothing

Table 10-21. Performance Monitor Bus Assignments (Page 6 of 8)

	Bits [0:31]	Byte Lane	Bits [0:7]	Event Description
ISU: TTM0 = ‘01’, TD_CP_DBGxSEL = ‘00’ TTM1 = ‘10’, TD_CP_DBGxSEL = ‘01’				
	0	0	0	Completion table full
	1	0	1	Floating-Point Register (FPR) mapper full
	2	0	2	Integer Exception Register (XER) mapper full
	3	0	3	FPU0 issue queue full
	4	0	4	CR mapper full
	5	0	5	BR issue queue full
	6	0	6	Link Register/ Counter Register (LR/CTR) mapper full
	7	0	7	FPU1 issue queue full
	8	1	0	FXU0/LSU0 issue queue full
	9	1	1	CR issue queue full
	10	1	2	LRQ full
	11	1	3	SRQ full
	12	1	4	FXU1/LSU1 issue queue full
	13	1	5	Flush originated by LSU
	14	1	6	Flush originated by branch miss predict
	15	1	7	Flush (includes LSU, branch miss predict)
	16:18	2	0:2	Instructions dispatched count
	19	2	3	Dispatch valid
	20	2	4	Dispatch reject
	21	2	5	Nothing
	22	2	6	Group experienced a branch redirect
	23	2	7	Group experienced a branch miss predict
	24	3	0	Nothing
	25	3	1	Dispatch blocked by scoreboard
	26	3	2	FXU0 produced a result
	27	3	3	Duration of the external interrupt enable in the Machine State Register (MSR[EE] = ‘0’)
	28	3	4	Nothing
	29	3	5	General Purpose Register (GPR) mapper full
	30	3	6	FXU1 produced a result
	31	3	7	MSR(EE) equals ‘0’ and interrupt pending

Table 10-21. Performance Monitor Bus Assignments (Page 7 of 8)

	Bits [0:31]	Byte Lane	Bits [0:7]	Event Description
Vector: TTM0 = '11', TD_CP_DBGxSEL = '01'				
	0	0	0	Arithmetic logic unit (ALU) issue queue full
	1	0	1	VPERM issue queue full
	2	0	2	ALU issue marked instruction
	3	0	3	VPERM issue marked instruction
	4	0	4	Saturation zero to one
	5	0	5	VPU mapper full
	6	0	6	Store issue marked instruction
	7	0	7	Nothing
	8:15 below selected by OVMA.USADEC2.CHICKEN1.IO.L2(7) = '0' (default)			
	8	1	0	Finish with IMR
	9	1	1	Generic forward
	10	1	2	Vector ALU issue count
	11	1	3	Denormalized traps
	12	1	4	Saturation bit set
	13	1	5	Finish contention cycle
	14	1	6	Nothing
	15	1	7	Nothing
	16:19 below selected by OVMP.RPRPM.MODE_PMON_MISC.IO.L2 = '0' (default)			
	16	2	0	Instruction finish with IMR
	17	2	1	Forwarding occurred from PERM or ALU or load
	17	2	2	Issue valid
	19	2	3	Saturation count for valid instruction
STS: TTM1 = '11', TD_CP_DBGxSEL = '01'				
	0	0	0	L2 cache access collision with L2 prefetch (Data Stream Touch [DST])
	1	0	1	L2 cache access collision with L2 prefetch (non-DST)
	2	0	2	L2 cache access for store
	3	0	3	L2 cache miss on store access (recent [R], shared [S], or invalid [I])
	4	0	4	L2 cache miss; bus response is modified intervention
	5	0	5	L2 cache miss; bus response is shared intervention
	6	0	6	I = '1' store operation (before gathering)
	7	0	7	I = '1' store operation completed on bus

Table 10-21. Performance Monitor Bus Assignments (Page 8 of 8)

	Bits [0:31]	Byte Lane	Bits [0:7]	Event Description
	8	1	0	I = '1' load operation completed on bus
	9	1	1	Cacheable store operation (before gathering)
	10	1	2	Master bus transactions completed
	11	1	3	Master bus transactions retried
	12	1	4	Master L2 cache store transaction on bus was retried
	13	1	5	Master L2 cache read transaction on bus was retried
	14	1	6	Master SYNC operation competed
	15	1	7	Master SYNC operation retried
	16	2	0	Load or store dispatch retries due to castout (CO) conflicts
	17	2	1	Load or store dispatch retries due to snoop conflicts
	17	2	2	Load or store dispatch retries
	19	2	3	All read/claim state machines busy
	20	2	4	All CO state machines busy
	21	2	5	All snoop state machines busy
	22	2	6	Cacheable store queue full
	23	2	7	I = '1' store queue full
	24	3	0	Snoop (external)
	25	3	1	Snoop state machine dispatched
	26	3	2	Snoop retried due to any conflict
	27	3	3	Snoop retried due to all snoop state machines busy
	28	3	4	Snoop caused cache transition from modified (M) to exclusive (E) or shared (S)
	29	3	5	Snoop caused cache transition from E to S
	30	3	6	Snoop caused cache transition from E or S to recent (R) or invalid (I)
	31	3	7	Snoop caused cache transition from M to I

10.7 Enabling the Performance Monitor Counters

10.7.1 Machine States

The eight counters in the 970FX performance monitor can be enabled to count events as a result of a number of machine state conditions and triggering events. The machine state conditions and triggering events are enabled by the settings of the MMCR0, MMCR1, and MMCRA register control fields, combined with the values of the performance monitor-related bits in the MSR and other SPRs. While machine states and triggering events are closely related in their effect on performance monitor behavior, it is easier to understand them if the two are first considered separately, as described in this section for the machine states and in *Section 10.7.2* on page 224 for the triggering events.

The term machine state condition as it is used here includes:

- Supervisor versus user (problem) state (MSR[PR], MMCR0[FCS, FCP])
- Marked versus unmarked process state (MSR[PMM], MMCR0[FCM1, FCM0])
- Conditional versus unconditional counting state (MMCR0[FC], MMCRA[FC1:4], FC5:8], CTRL[31], MMCRA[FCWAIT])
- Wait state versus non-wait state (CTRL[31], MMCRA[FCWAIT])

By combining the state of the machine with the events selected for counting, many different aspects of performance can be obtained for a given program.

For example, a programmer might want to gather statistics on only a particular process. This can be done by doing the following:

1. Set the appropriate bits in MMCR0 that enable counting only for a marked process.
2. Before each run of the selected process begins, set the performance monitor mode bit in the Machine State Register (MSR[PMM]) to the value that marks that process.
3. After each run of the selected process ends, set the performance monitor mode bit to the value that unmarks that process.

Another example follows:

The performance monitor can be set up to count only when the machine is in the supervisor state by ensuring that the MMCR0 bits that specify counting are enabled only when the machine is in the supervisor (privileged) state and are disabled when the machine is in the user (problem) state.

Table 10-22 on page 224 illustrates several different counting scenarios.

Table 10-22. Examples of Event Counter Enabling States

Counting State	MMCR0[Bit] = Value	MSR[Bit] = Value
Disable all counting	FC = ‘1’	Does not count for all values of PR, PMM
Enable all counting	FC = ‘0’	Counts[1] for all values of PR, PMM
Enable counting in supervisor state only	FCP = ‘1’, FCS = ‘0’	Counts when PR = ‘0’
Disable counting in supervisor state only	FCS = ‘1’, FCP = ‘0’	Counts when PR = ‘1’
Enable counting in user (problem) state only	FCS = ‘1’, FCP = ‘0’	Counts when PR = ‘1’
Disable counting in user (problem) state only	FCS = ‘0’, FCP = ‘1’	Counts when PR = ‘0’
Enable counting for marked processes only	FCM0 = ‘1’, FCM1 = ‘0’	Counts when PMM = ‘1’
Disable counting for marked processes only	FCM0 = ‘0’, FCM1 = ‘1’	Does not count when PMM = ‘1’
Enable counting for unmarked processes only	FCM0 = ‘0’, FCM1 = ‘1’	Counts when PMM = ‘0’
Disable counting for unmarked processes only	FCM0 = ‘1’, FCM1 = ‘0’	Does not count when PMM = ‘0’
Enable counting for marked processes in supervisor state only	FCP = ‘1’, FCS = ‘0’, FCM0 = ‘1’, FCM1 = ‘0’	Counts when PR = ‘0’ and PMM = ‘1’
Enable counting for unmarked processes in supervisor state only	FCP = ‘1’, FCS = ‘0’, FCM0 = ‘0’, FCM1 = ‘1’	Counts when PR = ‘0’ and PMM = ‘0’
Enable counting for marked processes in user (problem) state only	FCP =‘0’, FCS = ‘1’, FCM0 = ‘1’, FCM1 = ‘0’	Counts when PR = ‘1’ and PMM = ‘1’
Enable counting for unmarked processes in user (problem) state only	FCP = ‘0’, FCS = ‘1’, FCM0 = ‘0’, FCM1 = ‘1’	Counts when PR = ‘1’ and PMM = ‘0’

Note:
1. All enables are based on whether the other MMCRx and/or MSR bits permit this action.

10.7.2 Trigger Events

Machine states that determine counter activity have been presented in *Section 10.7.1* on page 223. Several examples of states and their corresponding counter behaviors were shown in order to illustrate some of the more common uses. In addition to counting enable/disable for various machine states, there are certain kinds of performance monitor conditions and events that can affect performance monitor counting activity. The occurrence of these conditions and events, which together are called trigger events, combined with the controls that enable the trigger events, can be used together with machine states to further control performance monitor activity.

The term trigger event as it is used for the 970FX performance monitor includes the following:

- The time-base transition event can occur when a selected time-base bit changes from ‘0’ to ‘1’. The time-base event setup uses the following fields: TB_REG[47, 51, 55, 63], HID0[13], and MMCR0[TBSEL]. The time-base event enable uses the following field: MMCR0[TBEE]. The possibility of side effects when an enabled time-base event occurs uses the following fields: MMCR0[FCECE, TRIGGER].
- The counter negative condition for PMC1 can occur when the value in PMC1 is negative. The PMC1 counter negative condition setup uses the following field: PMC1[0]. The PMC1 counter negative condition enable uses the following field: MMCR0[PMC1CE]. The possibility of side effects when the PMC1 counter negative condition occurs uses the following fields: MMCR0[FCECE,TRIGGER].
- The counter negative condition for PMCj ($2 \leq j \leq 8$) occurs when the value in any PMCj is negative. The PMCj counter negative condition setup uses the following field: PMCj[0]. The PMCj counter negative con-

dition enable uses the following field: MMCR0[PMCjCE]. The possibility of side effects when the PMCj counter negative condition occurs uses the following fields: MMCR0[FCECE, TRIGGER].

Note: The three kinds of trigger events can occur independently of each other and independently of whether the condition or event is enabled. For example, a counter can go negative regardless of whether the counter negative condition for that counter is enabled. However, the side effects of that counter going negative will be seen only if the counter goes negative, the counter negative condition for that counter is enabled, and the side effects are also enabled.

By combining the trigger events and their respective enables with the time-related values obtained in the counters, performance profiles of different kinds of events can be obtained for a given program.

10.7.2.1 Time-Base Transition Events

Time-base events occur when the selected time-base bit (TB_REG[47, 51, 55, 63], HID0[13], MMCR0[TBSEL]) changes value from ‘0’ to ‘1’. If the time-base transition event is enabled (MMCR0[TBEE]), then any performance monitor action that is started by the occurrence of a trigger event (MMCR0[TRIGGER]) will be initiated. Any performance monitor action that is stopped by the occurrence of a trigger event (MMCR0[FCECE]) will be terminated. In multiprocessor systems with the Time-Base Registers synchronized among the processors, time-base transition events can be used to correlate the performance monitor data obtained by the several processors provided that software has specified the same TBSEL value for all of the processors in the system.

The frequency of the time base is implementation dependent, and a system service routine should be invoked to obtain the frequency before a value for TBSEL is chosen.

10.7.2.2 PMC1 Counter Negative Condition Events

The PMC1 counter negative condition occurs when PMC1[0] equals ‘1’, which can occur either through counting from ‘0’, counting from a positive value greater than ‘0’, or through setting the PMC1[0] bit to ‘1’ in an interrupt or service routine. If the PMC1 negative count condition is enabled (MMCR0[PMC1CE]), any performance monitor action that is started by the occurrence of a trigger event (MMCR0[TRIGGER]) will be initiated, and any performance monitor action that is stopped by the occurrence of a trigger event (MMCR0[FCECE]) will be terminated when PMC1[0] becomes negative.

For example, if the PMC1 negative count condition is to be used to start the other PMCj counters after a designated number of cycles has elapsed, the set up would be as follows:

1. PMC1 is set to the value (x‘8000 0000’ - <number of cycles>).
2. PMC1SEL is set up to count cycles.
3. MMCR0[PMC1CE] is set to enable the PMC1 negative counter condition.
4. TRIGGER is enabled.

In this case, it is not necessary to enable the PMC1 counter negative condition because the TRIGGER uses either PMC1 negative or an enabled trigger event to start the enabled PMCjs counting.

10.7.2.3 PMCj (2 ≤ j ≤ 8) Counter Negative Condition Events

The PMCj (2 ≤ j ≤ 8) counter negative condition event occurs when PMCj[0] equals ‘1’ (2 ≤ j ≤ 8), which can occur through counting from ‘0’, counting from a positive value greater than ‘0’, or through setting the PMCj[0] (2 ≤ j ≤ 8) bit to ‘1’ in an interrupt or service routine. If the PMCj (2 ≤ j ≤ 8) counter negative condition event is enabled (MMCR0[PMCjCE]), any performance monitor action that is started by the occurrence of a trigger event (MMCR0[TRIGGER]) will be initiated, and any performance monitor action that is stopped by the occurrence of a trigger event (MMCR0[FCECE]) will be terminated, when any of the PMCj (2 ≤ j ≤ 8) counters become negative.

10.7.3 Method for Enabling or Disabling Performance Monitor Counting

This section describes the fundamental mechanism that should be used to place the selected values into the Performance Monitor Registers and other SPRs to initiate and terminate counting.

Once all of the control and event selection choices have been made, there are 32-bit and 64-bit values that must be placed into each of the registers associated with performance monitor counting. These values are placed in the registers with the **mtspr** instruction, which may be executed only in supervisor mode.

Note: If the Performance Monitor Counter Register values are changed while the performance monitor is enabled for counting, then the resulting performance monitor state is undefined.

The basic steps for enabling the performance monitor counting activity are as follows:

1. Enter supervisor mode.
2. Execute a synchronizing instruction.
3. Execute all **mtspr** instructions that place values to enable counting into the performance monitor and other Special Purpose Registers except for MMCR0.
4. Execute all **mtspr** instructions to initialize the performance monitor counters to the appropriate values.
5. Execute the **mtspr** instructions that place the value to enable counting into MMCR0.
6. Execute a synchronizing instruction.
7. Exit supervisor mode.
8. Start the program for which counting is to be done.

When the program being counted completes, the following steps are used to disable performance monitor counting:

1. Enter supervisor mode.
2. Execute a synchronizing instruction.
3. Execute the **mtspr** instructions that places the value to disable counting into MMCR0.
4. Execute all **mfspr** instructions to read the values from the performance monitor counters.
5. Execute a synchronizing instruction.
6. Exit supervisor mode.

The performance monitor counters contain either the number of times the selected event has occurred or the number of cycles during which the monitored event occurred after the performance monitor was enabled for counting.

Note: In either case, any counted events that occur after the performance monitor counting is enabled in supervisor mode, but before the program under study is entered, will be included in the overall count value. In the same way, any counter events that occur after the program under study is exited, but before the performance monitor counting is disabled, will also be included in the overall count value.

10.8 Performance Monitor Exceptions

The three trigger events described in *Section 10.7.2* beginning on page 224 can cause a performance monitor exception to occur and the subsequent performance monitor exception to be generated if the following sequence of actions occurs:

1. The trigger event occurs.
2. The trigger event is enabled.
3. The performance monitor exception is enabled.
4. External interrupts are enabled.

Note: This is the highest priority interrupt.

A performance monitor exception can be disabled for a given trigger event by disabling that trigger event (MMCR0[TBEE, PMC1CE, PMCjCE]). Performance monitor exceptions can be disabled for all of the trigger events collectively by disabling the performance monitor exception (MMCR0[PMXE]). The performance monitor exception, which is classified as an external interrupt, can be disabled either by disabling the performance monitor exception (MMCR0[PMXE]) or by disabling the external interrupts (MSR[EE]).

When an enabled condition or event occurs and a performance monitor exception is taken, the performance monitor exception is disabled by the hardware so that the SIAR and SDAR will contain the address and data information for an instruction that was executing at or around the time of the exception. Because the contents of the SIAR and SDAR can be altered if and only if MMCR0[PMXE] equals '1', the contents of those registers can change only if software re-enables the performance monitor exception. If such a re-enable is done and multiple performance monitor exceptions occur before the performance monitor exception is taken, then the exception reflects the most recently occurring such exception. Data from the previous exceptions are lost.

If a performance monitor exception is pending and the value of MSR[EE] is changed from '0' to '1', then the performance monitor exception will occur before the next instruction is executed provided no higher priority exception exists. The occurrence of the performance monitor exception cancels the performance monitor exception.

In summary, the following registers are set when a performance monitor exception occurs:

- SRR0[0:63] is set to the effective address of the instruction that the processor would have attempted to execute next if no interrupt conditions were present.
- SRR1[33] is set to '1' if the contents of the SDAR and the SIAR are associated with the same instruction.
- Other SRR0 and SRR1 bits are set as described in *Chapter 4 Exceptions*.
- SIAR is set to the effective address of the marked instruction, where the marked instruction is an instruction that was executing, possibly out-of-order, at or around the time that the performance monitor exception occurred. The contents of the SIAR may be altered by the processor if and only if MMCR0[PMEE] is set to '1'. Thus, after a performance monitor exception occurs, the contents of the SIAR is not altered by the processor until software sets MMCR0[PMEE] to '1'. After software sets MMCR0[PMEE] to '1', the contents of SIAR is undefined until the next performance monitor exception occurs.

- SDAR is set to the effective address of the storage operand of an instruction that was executing, possibly out-of-order, at or around the time that the performance monitor exception occurred. This storage operand is called the marked data and may be, but need not be, the storage operand (if any) of the marked instruction. If the performance monitor causes a performance monitor exception, the SRR1 indicates whether the marked data is in fact the storage operand of the marked instruction. The contents of the SDAR may be altered by the processor if and only if MMCR0[PMEE] is set to ‘1’. Thus, after a performance monitor exception occurs, the contents of the SDAR is not altered by the processor until software sets MMCR0[PMEE] to ‘1’. After software sets MMCR0[PMEE] to ‘1’, the contents of SDAR are undefined until the next performance monitor exception occurs.
- MSR is set the same as for other external interrupts.

10.9 Instruction Matching and Sampling

The 970FX performance monitor provides a facility for the detailed analysis of instruction flow by sampling particular instructions or classes of instructions. Instructions must pass through three stages of eligibility to be marked for sampling. The contents of the SIAR/SDAR reflect the marked instruction that is currently executing.

10.9.1 Stage 1 Eligibility

There are two instruction marking facilities for stage 1:

- **IFU** - Uses the IMC array to either set or clear the mark (imr) bit associated with any matching instruction. This imr bit, along with the branch (B), first (F), split (S), and last (L) predecode bits, are retained in the L1 instruction cache along with each instruction. All instructions with the imr bit set are eligible for stage 2 of marking.
- **IDU** - Uses the BFSL predecode bits (set independent of any instruction matching in the IMC array) along with the imr_match and imr_mask fields in the MMCRA. All instructions with predecode bits (BFSL) matching imr_match when ANDed with imr_mask are eligible for stage 2 of marking.

Which facility is used depends on the MMCRA imr_select field. If imr_select equals ‘0’, the IDU facility will be used. Otherwise, if imr_select equals ‘1’, the IFU facility will be used.

10.9.2 Stage 2 Eligibility

Any eligible instructions from stage 1 are further filtered by the imr_mark field in the MMCRA:

‘00’:	All IOPs
‘01’:	Only IOPs resulting from microcode expansion
‘10’:	Only one IOP per PowerPC instruction
‘11’:	First IOP to go to the LSU for every PowerPC load/store instruction

10.9.3 Stage 3 Eligibility

Any eligible instructions from stage 2 are marked if the internal *ok_to_sample* performance monitor signal is asserted. This results in at most one marked instruction in the pipeline at a time. Another eligible instruction will be marked after a marked instruction completes or the previous marking cycle has timed out (set by the SCOM x‘240’ IDLE field).

10.10 IFU Instruction Matching Facility

The PowerPC instruction matching by opcode or extended opcode is performed by the IFU Instruction Matching facility implemented in hardware through the use of an instruction match CAM (IMC) array. When a PowerPC instruction is fetched from memory, the IFU instruction matching facility compares the instruction with the opcode/extended opcode mask values in each of the IMC array rows. If a PowerPC instruction matches one or more IMC array row masks, the IFU predecode bits associated with the marked instruction are set based on the value of the IMC function bit in each of the matched IMC array rows. The IMC function bits and descriptions of the subsequent processing for the PowerPC instruction matched for each function bit are as follows:

- Force only (fo) forces the IDU to place the PowerPC instruction in a group by itself by setting the predecode bits for the instruction as 'first' and 'last' in the group. The IMC fo function bit is used for hardware debug and workaround. It is only accessible by using scan.
- Instruction marking (imr), which is used for performance monitor instruction marking, causes the IDU to recognize that the instruction is IFU-eligible for marking. The IMC imr function bit is used by the performance monitor for marked instruction events and threshold event counts. It is accessible to the user by using the supervisor mode **mtimc**/**mfimc** instruction.

In addition to the IFU predecode bits associated with the IMC function bits, other IFU predecode bits—based on the instruction type—are bundled with each instruction in the IFU instruction cache.

Note: As long as an instruction resides in the Level 1 Instruction cache, its match bit will remain unchanged. If the match condition for an instruction changes, then the Level 1 Instruction cache should be flushed to ensure proper setting of the match bits for all instructions.

10.10.1 Overview of the IFU Instruction Matching Facility

Each processor core includes an IFU instruction matching facility, implemented as the IMC array, which is used to maintain the kinds of IFU instruction matching requests and the mask values used for each IFU matching request. The method of reserving an IMC array row differs depending on whether the row is being requested for use by the hardware debug facility (using scan only) or for the performance monitor marked instruction or threshold event facility (using the supervisor **mtimc** and supervisor or user **mfimc** instructions). All IMC array rows required for hardware debug operations (fo reservations) are reserved during the 970FX power-on reset (POR) scan sequence. Any IMC array rows that have not been fo-reserved during system initial program load (IPL) may be requested by an executing program for use by the performance monitor.

Before an executing program requests an IMC array row, the program must determine which IMC array rows are available to software; that is, which rows are not fo-reserved. This information is available by reading the IMC Special Purpose Register (SPR) with the **mfimc** instruction, which may be executed only in supervisor mode. An executing program may request any IMC array rows that are not fo-reserved by writing the IMC SPR with the **mtimc** instruction, which can be executed only in supervisor mode.

10.10.2 IMC Array

The IMC array, which is contained in the IFU, consists of eight row entries as follows:

- Six rows (0 through 5) support a 17-bit partial instruction match of the opcode and extended opcode fields in instruction bits [0:5] and instruction bits [21:31], respectively.
- Two rows (6 and 7) combined support a full 32-bit instruction match of all the instruction fields in instruction bits [0:31].

Each IMC array row includes fields for:

- The imr function bit (bit [60], called the Mark bit) that determines the predecode tag value sent to the IDU. The imr bit is programmable through the Instruction Match facility.
- The two mask values that together encode the instruction match criteria (called v0 and v1).
- The machine configuration this match applies to (PR, FP Available, VPU Available). Note that the PR bit in the MSR determines the Privileged state if a '0' and the Problem state if a '1'.
- Optional replacement field for the BFSL (predecode) bits to replace for a matched instruction (SPR cannot access theses fields).

The programmer's model view of the IMC array is shown in *Table 10-23* and *Table 10-24*.

Table 10-23. Partial Match Rows in the IMC Array

0:16	17	18	19	20	21:37	38	39	40	41	42:52	53:58	59	60	61:63
v0(17)	N/A	MSR [PR]	MSR [FP]	MSR [VPU]	v1(17)	N/A	MSR [PR]	MSR [FP]	MSR [VPU]	N/A	BFSL Replacement	Replace	Mark	Index
														0
														1
														2
														3
														4
														5

Table 10-24. Complete Match Rows in the IMC Array

0:31	32	33	34	35	36:52	53:58	59	60	61:63
v0 (Row 6) / v1 (Row 7)	N/A	MSR [PR]	MSR [FP]	MSR [VPU]	N/A	BFSL Replacement	Replace	Mark	Index
									6
									7

The following are some rules and general facts about the IMC array features and use:

- The IMC array row with full 32-bit v0 and v1 masks can be used to match the opcode, the extended opcode, and all other fields for any PowerPC instruction.
- The IMC array rows with 17-bit mask values cannot be used to match branch instructions.
- The 17-bit v0 and v1 mask values can be used to match only those bits of a PowerPC instruction (but not branch instructions) that represent the opcode and extended opcode bit fields of that instruction and cannot be used to match any other instruction bit fields such as the register fields, shift fields, mask fields, reserved fields, immediate fields, or the rc bit.
- The bits of v0 and v1 that correspond to any fields other than the opcode and extended opcode bit fields of instruction bits [21:31] should be set to the 'don't care' v0 and v1 value as is explained in *Section 10.10.5 The v0 and v1 Mask Criteria* on page 233.
- The 17-bit v0 and v1 bits [0:5] and v0 and v1 bits [6:16] correspond to instruction bits [0:5] and instruction bits [21:31].
- The 3 mode bits (PR, FP, VP) correspond to MSR[PR], MSR[FP], and MSR[VP]

10.10.3 Reading the IMC SPR with the mfimc Instruction

The IMC SPR is read in order to determine which IMC array rows are fo-reserved. The image of the IMC SPR that is obtained by executing the **mfimc** instruction (which can be executed in supervisor and user mode) is referred to as the **patch map**. The programmer's model view of the patch map is shown in *Figure 10-18*.

Figure 10-18. Patch Map

Patch Map Bit Number (IMC Array Row Address)								
0:55	56 (0)	57 (1)	58 (2)	59 (3)	60 (4)	61 (5)	62 (6)	63 (N/A)
Reserved[1]								Reserved

1. The designation 'reserved' is used both to indicate bits in the patch map that are not used for this implementation, as well as to identify the fields that are not accessible when using the **mfimc** instruction.

A patch map status bit value of '1' indicates that the associated IMC array row is fo-reserved and cannot be altered by the executing program. If the status bit value is '1', an **mtimc** to the associated IMC array row is treated as a no-op.

The facts summarized below emphasize what information is and is not available from the patch map:

- The only information that the patch map provides about the IMC array rows is whether or not a row is fo-reserved.
- The patch map provides no information about reservations made by the performance monitor IMR facility.
- An IMC array row that is reserved by the performance monitor IMR facility will not show a patch map status bit of '1' if the patch map is read with the **mfimc** instruction.
- There is no **mfimc**-like instruction that will help an executing program determine what v0 and v1 values were used when a performance monitor IMR facility request was made.
- All information about IMC array use must be maintained by the executing program.
- If an IMR reservation is made for an available IMC array row and the v0, v1 mask used for the IMR reservation is the same as a v0,v1 mask that is already being used for an fo reservation, the instruction selected by the v0, v1 mask will be correctly processed for both the imr request and the fo request.

10.10.4 Writing the IMC SPR with the mtimc Instruction

After an executing program determines which IMC array rows are fo-reserved (by doing a **mfimc** and seeing which patch map bits are set to '1'), the program should initialize an IMC_reservation data structure, which it should then use to track all fo/imr-reservations.

The program can make an IMR reservation of any available IMC array row through use of the **mtimc** instruction (which can be executed only in supervisor mode). The **mtimc** instruction is used to select the IMC array row, provide the v0 and v1 mask values, and set the imr bit. After a performance monitor IMR request is successfully completed, the requesting program should update its IMC_reservation data structure to record the reservation and the v0, v1 mask values.

A performance monitor IMR request remains in effect on a processor until it is:

- Canceled by an **mtimc** instruction that sets the imr bit to '0',
- Changed by an **mtimc** instruction that changes the IMC array row fields v0, v1 and writes the imr bit to '1',
- Cleared or replaced by a system reboot, or
- Overwritten by the service processor using scan or SCOM.

When an existing performance monitor IMR request is changed or canceled by a subsequent **mtimc** instruction, the executing program must update its IMC reservation data structure and invalidate the I-cache in order to reset the match bits set by a previous IMR reservation. Otherwise, stale instruction marks from the previous IMC setup might make the performance measurements unreliable, meaning old marks might still be encountered and new marks might not always occur depending on the state of the I-cache.

The programmer's model of the IMC SPR for the **mtimc** instruction differs slightly for requests of the 32-bit and the 17-bit instruction match IMC array rows. The IMC array rows for 17-bit matches, which are at IMC array row addresses 0 - 5, are written with a single **mtimc** instruction. It specifies the IMC array row address, the 17-bit opcode and extended opcode instruction mask values for v0 and v1, the machine mode mask values for v0 and v1, and it sets the imr bit to '1'. The image of the IMC SPR that is used when executing the **mtimc** instruction for IMC array row addresses 0 - 5 is shown in *Table 10-25* on page 232.

Table 10-25. IMC SPR for a 17-Bit Match

IMC Row Bit Numbers (IMC Array Row Fields)								
0:16	17	18:20	21:37	38	39:41	42:59	60	61:63
$v0_{[0-5,21-31]}$	Res	$V0_{PR,FP,VP}$	$v1_{[0-5,21-31]}$	Res	$v1_{PR,FP,VP}$	Reserved[1]	IMR1	IMC Row Address

1. The designation 'reserved' is used both to indicate bits in the IMC SPR that are not used for this implementation as well as to identify the fields that are not accessible when using the **mtimc** instruction.

The IMC array row for 32-bit matches, which is at IMC array row address 6 and 7, is written with two **mtimc** instructions. Each specifies an IMC array row address, a 32-bit instruction mask value, and an imr bit value as follows:

- The first **mtimc** instruction sets the IMC SPR bits [61:63] = '110', the IMC SPR bits [0:31] = $v0_{instruction[0-31]}$, the IMC SPR bits [33:35) = $v0_{PR,FP,VP}$, and the IMC SPR bit [60] = '0'.
- The second **mtimc** instruction sets the IMC SPR bits [61:63] = '111', the IMC SPR bits [0:31] = $v1_{instruction[0-31]}$, the IMC SPR bits [33:35] = $v1_{PR,FP,VP}$, and the IMC SPR bit [58] = '1'.

The image of the IMC SPR that is used when executing the first **mtimc** instruction (for IMC array row address 6) is shown in *Table 10-25* and the image of the IMC SPR that is used when executing the second **mtimc** instruction (for IMC array row address 7) is shown in *Table 10-26*.

*Table 10-26. IMC SPR Used when Writing the Second **mtimc** Instruction for a 32-Bit Match*

IMC Row Bit Numbers (IMC Array Row Fields)							
0:31	32	33:35	36:59	60 imr	61:63 IMC Row Address		
$v1_{[0\text{-}31]}$	Reserved	$v1_{PR,FP,VP}$	Reserved	1	1	1	1

10.10.5 The v0 and v1 Mask Criteria

The mask criteria used for matching the values in the instruction bit fields and the machine state are based on the values in the v0 and v1 fields. Each pair of bits, v0(n) and v1(n); where n is 0 – 16, or n is 0 – 31, or n equals PR, FP; is interpreted as an encoded 4-value criterion. It determines how the corresponding instruction bit (m) is to be matched, where either m is 0 - 31 for a full instruction match or m is 0 - 5, 21 - 31 for an opcode/extended opcode match. Bit correspondences between v0, v1 bit numbers and instruction numbers depend on whether v0,v1 is a 17-bit or a 32-bit mask and are as follows:

- 17-bit mask opcode bits:
  v0,v1 (0:5) corresponds to instruction bits[0:5]
- 17-bit mask extended opcode bits:
  v0,v1(6:16) corresponds to instruction bits[21:31]
- 32-bit mask full instruction bits:
  v0.v1(0:31) corresponds to instruction bits[0:31]

The bit match criteria established by the four values of v0(n), v1(n) are shown in *Table 10-27* on page 233.

Table 10-27. Encoding Bits v0 and v1 of the IMC Array Mask

v0 Value	v1 Value	Meaning
0	0	Never match (disable all)
0	1	Match a one (1)
1	0	Match a zero (0)
1	1	Always match (don't care)

10.10.6 Instruction Matching Examples

```
17-bit match using only instruction opcode (bits [0:5])
Load Doubleword: opcode = 58, extended opcode = N/A (don't care); PR=0; FP=1, VP=0/1
v0 = 0b00010111111111111 1101
v1 = 0b11101011111111111 1011

17 bit match using instruction opcode and extended opcode
Load Word and Zero Indexed: opcode = 31, extended opcode = 23; PR=1; FP=0/1, VP=0/1
v0 = 0b10000011111010001 1011
v1 = 0b01111100000101110 1111
```

10.11 IDU Instruction Sampling Facility

Another level of sampling activity—performed in the IDU—includes further processing for the imr and fo reservations made using the IFU Instruction Matching facility. Because of the way an instruction is processed through the two IDU selection stages, it is possible that the IDU instruction sampling processing can override previous IFU IMR marking.

The IDU instruction sampling facility that produces marked instructions for the instruction pipeline consists of two independent selection stages. In each of the two selection stages, the selection criteria for that stage determines which instructions pass out of that IDU selection stage as eligible to be marked. Because an instruction passes through each of the two IDU selection stages after it is processed for IFU IMR marking, it is possible that the IDU instruction sampling eligibility criteria can override previous IFU IMR marking. It is also possible that the IDU stage 2 processing can override IDU stage 1 processing if the criteria for each of the two selection stages are not set up correctly.

An eligible instruction that is marked by the IDU is referred to as a marked (sampled) instruction.

10.11.1 Overview of the IDU Instruction Sampling Facility

The IDU instruction sampling facility uses the IFU imr and predecode bits from the instruction cache, together with PMU/SCOM data and control fields, to determine which instructions are eligible to be marked (sampled) and when an instruction can actually be marked (sampled).

Operation of the IDU instruction sampling facility to determine which instructions are eligible for marking occurs continuously when performance monitor mode sampling is enabled (MMCRA[63] equals ‘1’). The choice of which instructions are eligible to be marked is based on the values of the IFU imr and predecode bits combined with the values of the select, mask, match, mark, and filter fields. The IDU continuously designates instructions as eligible to be marked based on the above fields, but the IDU only marks instructions when sampling is enabled and the performance monitor signals the IDU that it is *ok_to_sample*. Only one marked group flows through the instruction pipeline at a time.

IDU processing of an instruction based on the fo IMC function bits is independent of IDU processing based on the IFU imr function bit, so a given instruction might be processed by the IDU for any or all of the fo, and imr functions.

The IDU eligibility stages continuously produce eligible instructions. The sample_enable field combined with the performance monitor signal *ok_to_sample* control final marking as outlined below. The information in parenthesis corresponds to the annotations on the left in *Figure 10-19* on page 239.

- The imr_select field (MMCRA[49]) determines the method used for stage 1 instruction eligibility (eligibility stage 1 - method).
- Depending on the imr_select field value, the imr_mask field (MMCRA[52:55]) and the imr_match field (MMCRA[56:59]) can be used to determine the type of the stage 1 eligible instructions that pass through to stage 2 (eligibility stage 1 - type).
- The imr_mark field (MMCRA[50:51]) determines what type of stage 1 eligible instructions are to be considered for stage 2 eligibility (eligibility stage 2 - type).
- The imr_filter field (SCOM x‘34’ [11:12]) determines which and how many of the stage 2 eligible instructions actually become marked instructions in the pipeline (eligibility stage 2 - method).
- The performance monitor signal *ok_to_sample* determines whether marking is blocked or might resume (mark/no mark stage).

The IFU matching and the IDU instruction sampling activities occur continuously regardless of whether instructions are being marked by the IDU. The IDU can mark an instruction only when the performance monitor signals that marking is enabled. After an instruction is marked by the IDU, the performance monitor disables marking until either a marked instruction completes, the instruction is a store that is sent to the STS, or the performance monitor completion delay timer indicates that the marked instruction has been flushed.

Note: As long as an instruction resides in the Level 1 Instruction cache, its match bit will remain unchanged. If the match condition for an instruction changes, the Level 1 Instruction cache should be flushed to ensure proper setting of the match bits for all instructions.

10.11.2 Stage 1 Eligibility

The IDU uses the IFU predecode bits for branch, first, split, and last (shown in *Table 10-28* on page 236) and the imr bit stored with an instruction in the instruction cache to establish eligibility for marking.

Note: As long as an instruction resides in the Level 1 Instruction cache, its imr match bit will remain unchanged. If the match condition for an instruction changes, the Level 1 Instruction cache should be flushed to ensure proper setting of the imr match bits for all instructions.

The method used to choose IDU stage 1 eligible instructions is based on the value of the imr_select field (MMCRA[49]). Depending on the value of the imr_select field, a second decision point may be required to choose the type of stage 1 eligible instructions using the imr_mask field (MMCRA[52:55]) and the imr_match field (MMCRA[56:59]). These two scenarios, based on the value of the imr_select field value, are as follows:

- imr_select equals '1':
  The IFU imr bit is used to determine stage 1 eligibility. All instructions with the IFU imr bit set are passed through to stage 2 as eligible for marking.
- imr_select equals '0':
  The IFU BFSL predecode bits are used to determine stage 1 eligibility.

If imr_select equals '1', so that the IFU imr bit determines stage 1 eligibility, it is possible to choose values for imr_mark (IDU eligibility stage 2 - type) that will cancel the IMR eligibility created in stage 1.

If imr_select equals '0' and the IFU BFSL predecode bits are used to determine stage 1 eligibility, there are two further stages of processing to establish the type of instructions that are eligible:

1. The BFSL predecode bits for the instruction are ANDed with the imr_mask field to produce a 4-bit intermediate result, and
2. The 4-bit intermediate result is compared with the imr_match field. All instructions with an exact 4-bit match between the intermediate result and the imr_match field are passed through to stage 2 as eligible for marking.

To match all instructions, and thus pass all instructions through to stage 2 as eligible for marking, set the following values for the stage 1 method/type decision variables:

- imr_select: '0'
- imr_mask: '0000'
- imr_match: '0000'

The IFU BFSL predecode will be used, the mask will result in all zeros for the intermediate result, and the match will always succeed. The eligibility method chosen at stage 1 can determine what kind of instruction is counted for the performance monitor count event called "number of instructions completed," depending on how the eligibility criteria is set up in stage 2.

:

Table 10-28. IFU BSFL Predecode Bit Definitions

B(ranch)	S(plit)	F(irst)	L(ast)	Classification	Description
0	0	0	0	Simple	One IOP formed from one instruction with restrictions.
0	0	0	1	Simple-Last	IOP formed will be the last in the resultant dispatch group.
0	0	1	0	Simple-First	IOP formed will be the first in the resultant dispatch group.
0	0	1	1	Simple-Only	IOP formed will be the only in the resultant dispatch group.
0	1	0	0	Split	Two IOPs formed from one instruction; both must be in the same dispatch group.
0	1	0	1	Split-Last	Two IOPs formed from one instruction; the second IOP must be the last IOP in the resultant group.
0	1	1	0	Split-First	Two IOPs formed from one instruction; the first IOP must be the first IOP in the resultant dispatch group.
0	1	1	1	Split-Only	Two IOPs formed from one instruction; no other IOPs are present in the resultant dispatch group.
1	0	0	0	Branch - Conditional	IOP formed from a conditional branch instruction.
1	0	0	1	Branch - Unconditional	IOP formed from an unconditional branch instruction.
1	0	1	0	Illegal Opcode	Not a valid instruction.
1	0	1	1	Reserved	
1	1	0	0	Microcode - Hard	A nonprogrammable microcode sequence must be generated.
1	1	0	1	Reserved	
1	1	1	0	Microcode - Conditional (otherwise: Split-Last)	A nonprogrammable microcode sequence must be generated if certain conditions are not met.
1	1	1	1	Microcode - Conditional (otherwise: Split-Only)	A non-programmable microcode sequence must be generated if certain conditions are not met (for example, the FXM[1] field is not singular).

1. Field mask used to identify the CR fields to be updated by the **mtcrf** instructions.

10.11.3 Stage 2 Eligibility

The imr_mark field (MMCRA[50:51]) value and then the imr_filter field (SCOM x'340'[11:12]) are used by the IDU to establish stage 2 eligibility for marking. The first decision point for IDU stage 2 eligibility is the type of stage 1 eligible instructions that can be stage 2 eligible; this determines final stage 2 eligibility. The second decision point for IDU stage 2 eligibility is the method of passing the eligible information to the mark/no mark stage; this determines what eligible instructions actually get marked.

The type of stage 1 eligible instructions that will be stage 2 eligible is based on the value of the imr_mark field (MMCRA[50:51]). The imr_mark field value determines stage 2 eligibility of instructions as follows:

00	All stage 1 eligible IOPs are stage 2 eligible for marking.
01	Only stage 1 eligible IOPs that resulted from microcode expansion are stage 2 eligible for marking.
10	Only one IOP per stage 1 eligible PowerPC instruction is stage 2 eligible for marking (use this mode to make all PowerPC instructions eligible).
11	For every stage 1 eligible PowerPC instruction, the first IOP that goes to the LSU is stage 2 eligible for marking (use this mode to make all load/store instructions eligible).

The stage 1 eligibility method combined with the stage 2 eligibility type determines what kind of instruction is counted for the performance monitor count event called "number of instructions completed" as shown in *Section 10.11.6 Examples of Instruction Sampling Scenarios* on page 242.

After the type of stage 2 eligible instructions is established, the method of passing the stage 2 eligibility information to the mark/no mark stage is determined in two steps using the imf_filter[11] bit value and then using the imr_filter[12] bit value, which have the following functions:

imr_filter[11:12]

0x	No filtering
1x	Randomly sample eligible IOPs

Bit [12] selects one of two behaviors in sampling from microcode expansions (and has no effect on sampling from non-microcode groups):

10	Use the Good_Address mode of sampling microcode expansions
11	Use the More_Hits mode of sampling microcode expansions

In Good_Address mode, there is at most one IOP in any microcode expansion that is eligible for sampling. This is either the first load/store IOP if there are any load/store IOPs in the expansion, or the first IOP in the final group of the expansion. If the random filter suppresses marking this IOP, then no IOP will be marked for the microcode expansion.

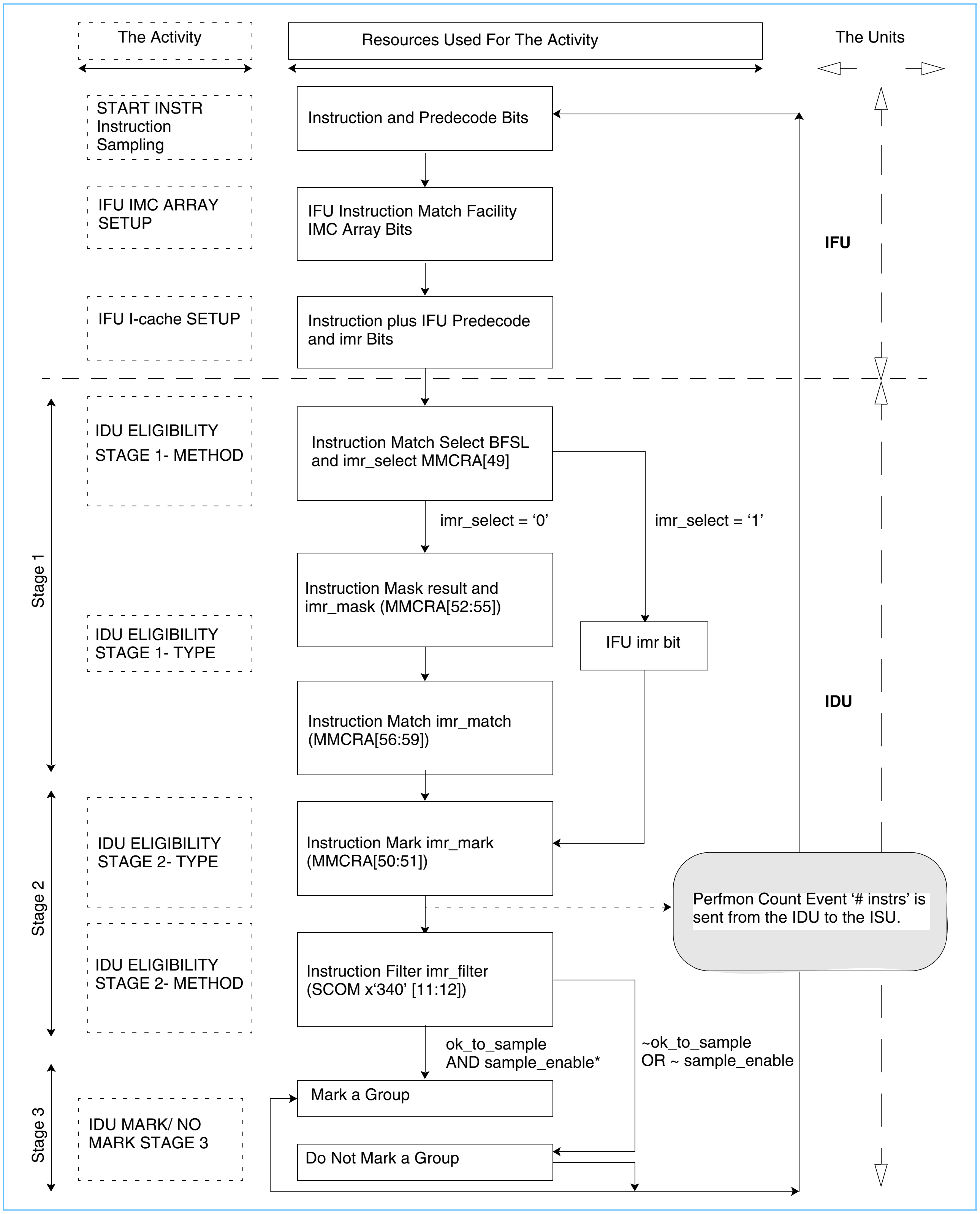
In More_Hits mode, multiple IOPs in a microcode expansion are eligible for sampling: the first load/store IOP in any group, or the first IOP of the final group. If the random filter suppresses marking the first of these IOPs, a subsequent one might still be sampled. (However, at most one will be marked in a single microcode expansion.)

The suggested mode for imr_filter[11:12] is '10'.

10.11.4 Stage 3 Mark/No Mark

The sample_enable field (MMCRA[63]) value and the *ok_to_sample* signal state are used by the IDU to complete the mark/no mark stage for instruction sampling. If marking is disabled with the sample_enable bit (MMCRA[63] equals ‘0’), then no group is marked by the IDU regardless of stage 2 eligibility. If marking is enabled with the sample_enable bit (MMCRA[63] equals ‘1’), then marking depends on the state of the handshake protocol between the IDU and the performance monitor signal ok_to_sample. The ok_to_sample signal is sent by the performance monitor to the IDU when the performance monitor determines that the previous marking cycle has completed successfully or has timed out. The handshake and synchronization mechanism are explained in *Section 10.11.5 Complete Masking, Matching, and Marking Cycle* on page 240.

Figure 10-19. IFU and IDU Instruction Sampling Flow


The Activity
Resources Used For The Activity
The Units
START INSTR Instruction Sampling
Instruction and Predecode Bits
IFU IMC ARRAY SETUP
IFU Instruction Match Facility IMC Array Bits
IFU
IFU I-cache SETUP
Instruction plus IFU Predecode and imr Bits
IDU ELIGIBILITY STAGE 1- METHOD
Instruction Match Select BFSL and imr_select MMCRA[49]
imr_select = '0'
imr_select = '1'
Stage 1
Instruction Mask result and imr_mask (MMCRA[52:55])
IDU ELIGIBILITY STAGE 1- TYPE
IFU imr bit
Instruction Match imr_match (MMCRA[56:59])
IDU
IDU ELIGIBILITY STAGE 2- TYPE
Instruction Mark imr_mark (MMCRA[50:51])
Stage 2
Perfmon Count Event '# instrs' is sent from the IDU to the ISU.
IDU ELIGIBILITY STAGE 2- METHOD
Instruction Filter imr_filter (SCOM x'340' [11:12])
ok_to_sample AND sample_enable*
~ok_to_sample OR ~ sample_enable
Mark a Group
Stage 3
IDU MARK/ NO MARK STAGE 3
Do Not Mark a Group

10.11.5 Complete Masking, Matching, and Marking Cycle

Instruction marking is set up by initializing the MMCRx[imr_xxx] fields, the SCOMyy[imr_filter] fields, and possibly the SCOMxx[xx_delay] for the kind of marking to be done. *Section 10.11.7 Enabling and Disabling Marking* on page 244 describes the procedure for initializing these registers plus any other registers to support performance monitor counted events. There are several examples at the end of this section that show the values of the imr_xxx fields for common marking and counting scenarios.

Once the instruction marking cycle is set up and enabled, instructions are continuously processed for eligibility as is described in *Section 10.11.2 Stage 1 Eligibility* on page 235, and in *Section 10.11.3 Stage 2 Eligibility* on page 237. The actual marking of a group described in *Section 10.11.4 Stage 3 Mark/No Mark* on page 238. The steps that occur from when the performance monitor *ok_to_sample* signal initiates marking until the next *ok_to_sample* signal initiates the next marking cycle are described in this section.

The overall timing of the marking cycle is driven by two performance monitor timers called the completion_delay counter and the idle_delay counter. The completion_delay counter is first initialized at the start of a marking cycle from the value in the sampling logic completion delay field (SCOM x'240' [COMPLN]). The idle_delay counter is initialized at the end of a marking cycle from the value in DELAY field (SCOM x'240' [DELAY]).

The purpose of the completion_delay counter is to predict the situation that the marked instruction has been flushed from the instruction pipeline before it can complete. The purpose of the idle_delay counter is to introduce a period of time between the end of a marking cycle (through either instruction completion or flush) and the start of the next marking cycle. The completion_delay and the idle_delay values must be greater than zero whenever matching or marking is enabled. Otherwise, the processor activity will be undefined.

A marking cycle begins when the performance monitor asserts the *ok_to_sample* signal for one cycle. The assertion of this signal (assuming that sample_enable equals '1') causes the IDU to mark the next group that enters stage 3 if it has passed all the stage 2 eligibility tests. After a group is marked in the IDU (this is a performance monitor count event), the IDU continues to process instructions for eligibility but does not mark another group until the next *ok_to_sample* signal is received from the PMU.

When the signal indicating that the group is marked in the IDU is sent from the IDU to the performance monitor (this is also a performance monitor count event), the performance monitor begins decrementing the completion_delay counter by one each cycle that the signal group_completed is asserted. If group_completed is not asserted, the completion_delay counter is not decremented.

This use of the completion_delay counter is intended to model a marked_group_flushed situation. The underlying assumption of this model and the default value of COMPLN equals 20 is as follows: if no marked group event occurs in any functional unit during a full wrap of the 20-entry completion buffer, then the marked group has been flushed. The completion_delay value must be greater than zero whenever instruction sampling is enabled or processor activity will be undefined.

If the completion_delay counter does not time out between when the signal indicating that the group is marked in the IDU is received by the performance monitor and when the next marked event signal is received by the performance monitor, the completion_delay timer is reset to the value in COMPLN. It resumes decrementing for each cycle that group_completed is asserted. At each stage of the marking cycle, the completion_delay counter is initialized, counts down whenever the signal group_completed is asserted, and either times out or is re-initialized when the next marked event occurs. Thus, at each stage of the marking cycle, the marked group is allowed the COMPLN number of cycles while groups are completing from the completion buffer before the marked group is considered flushed.

The sequence of events for a complete marking cycle, where a marked group moves through all of the instruction pipeline stages without being flushed, is as follows:

1. The performance monitor signal *ok_to_sample* is asserted for one cycle.
2. A group is marked when the IDU transfers the group to the ISU (performance monitor count event direct5[4]). The completion_delay counter is initialized to the value COMPLN and begins to decrement on each cycle that group_completed is asserted.
3. A marked group is dispatched (performance monitor count event direct1[2]). The completion_delay counter is reinitialized to the value COMPLN and begins to decrement on each cycle that group_completed is asserted.
4. A marked group is issued (performance monitor count event direct6[5]). The completion_delay counter is reinitialized to the value COMPLN and begins to decrement on each cycle that group_completed is asserted.
5. A marked group finishes (FPU: performance monitor count event direct7[4], FXU: performance monitor count event direct6[4], CRU: performance monitor count event direct4[5], BRU: performance monitor count event direct2[5], LSU: performance monitor count event direct8[4], any unit: performance monitor count event direct7[5]). The completion_delay counter is reinitialized to the value COMPLN and begins to decrement on each cycle that group_completed is asserted.
6. A marked group completes (performance monitor count event direct4[4]).
7. The idle_delay counter is initialized to the value DELAY and is then decremented to '0'.
8. The *ok_to_sample* signal is asserted for one cycle and the marking cycle begins again at step 2.

When the completion_delay counter times out (performance monitor count event direct5[5]), the performance monitor enters the marking cycle state where the idle_delay counter is set to the value DELAY and is then decremented to '0' (step 7 above). When the idle_delay counter times out, the performance monitor asserts the *ok_to_sample* signal for one cycle, and the marking cycle begins again.

The default idle_delay value is set to four. The idle_delay value must be greater than zero whenever matching/marking is enabled or processor activity will be undefined.

If a marked store is sent to the STS (performance monitor count event direct6[3]), the event called "marked store sent to STS" stops the marking cycle described above and causes the performance monitor to enter the idle state. The performance monitor stays in the idle state until one of the signals "sampled store complete" (performance monitor count event direct1[3]) or "sampled store complete with intervention" (performance monitor count event direct3[3]) is received. At that time, the performance monitor resumes the marking cycle at marking cycle step 7 above.

When sampling_enable is set to zero, the performance monitor enters the idle state of the marking cycle until sampling is enabled again.

10.11.6 Examples of Instruction Sampling Scenarios

Follow the procedure in *Section 10.11.7 Enabling and Disabling Marking* on page 244 to place the values for instruction sampling into the appropriate Special Purpose Register fields. *Section 10.4 Performance Monitor Control Registers* on page 191 describes the performance monitor related registers and fields. To count marked events, the appropriate PMCxSEL fields should also be set up and the enable counting bit must be set to the enabled value. In each of the examples below, only the values of the Instruction Sampling Register fields are shown.

Example 1: Set up the instruction sampling facility to count PowerPC instructions as the performance monitor count event called "number instructions completed."

The values given here are those that set up the instruction sampling stages so that the performance monitor count event called "number instructions completed" will be the count for PowerPC instructions completed. If sampling is enabled, instructions will be randomly sampled. This is the recommended setting. The required field values are as follows:

imr_mark	'10'	Only one IOP per PowerPC instruction is stage 2 eligible for marking.
imr_select	'0'	
imr_mask	'0000'	
imr_match	'0000'	
imr_filter[11:12]	'10'	If sampling is enabled, randomly sample.

Example 2: Set up the instruction sampling facility to count IOP instructions as the performance monitor count event called "number instructions completed."

The values given here are those that set up the instruction sampling stages so that the performance monitor count event called "number instructions completed" will be the count for IOP instructions completed. The required field values are as follows:

imr_mark	'00'	All stage 1 eligible IOPs are stage 2 eligible for marking.
imr_select	'0'	
imr_mask	'0000'	
imr_match	'0000'	

Example 3: Set up the instruction sampling facility to count load/store instructions as the performance monitor count event called "number instructions completed."

The values given here are those that set up the instruction sampling stages so that the performance monitor count event called "number instructions completed" will be the count for load/store instructions completed. The required field values are as follows:

imr_mark	'11'	For every PowerPC load/store instruction, the first IOP that goes to the LSU is stage 2 eligible for marking.
imr_select	'0'	
imr_mask	'0000'	
imr_match	0000	

Example 4: Set up the instruction sampling facility to match or mask all PowerPC instructions that are BFSL-Split, and then sample eligible instructions that get through the random filter.

The values given here are those that set up the instruction sampling stages so that the performance monitor count event called “number instructions completed” will be the count for PowerPC BFSL-Split instructions completed. The required field values are as follows:

imr_mark	‘00’	All stage 1 eligible IOPs are stage 2 eligible for marking.
imr_select	‘0’	
imr_mask	‘0100’	
imr_match	‘0100’	
imr_filter[11:12]	‘10’	If sampling is enabled, randomly sample.

Example 5: Set up the instruction sampling facility to match or mask all PowerPC instructions that are BFSL-Hard microcoded, and then sample all eligible instructions.

The values given here are those that set up the instruction sampling stages so that the performance monitor count event called “number instructions completed” will be the count for PowerPC BSFL-Hard microcoded instructions completed. The required field values are as follows:

imr_mark	‘00’	All stage 1 eligible IOPs are stage 2 eligible for marking.
imr_select	‘0’	
imr_mask	‘1100’	
imr_match	‘1100’	
imr_filter[11:12]	‘00’	Pass all stage 1 eligible bits in the group.

Example 6: Set up the instruction sampling facility to IFU IMR match or mask all PowerPC add instructions, and then sample all eligible instructions.

The values given here are those that set up the instruction sampling stages so that the performance monitor count event called “number instructions completed” will be the count for IMC-marked PowerPC instructions completed. The required field values are as follows:

imr_mark	‘10’	
imr_select	‘1’	
imr_mask	‘0000’	
imr_match	‘0000’	
imr_filter[11:12]	‘00’	If sampling is enabled, randomly sample.

10.11.7 Enabling and Disabling Marking

The processor comes out of reset with instruction marking disabled (MMCRA[63] equals '0') and with all of the MMCRA/SCOM[imr_xxx] fields set to zero. To set up the performance monitor for marking, follow these steps:

1. Enter supervisor mode.
2. Execute a synchronizing instruction or wait for any previous activity to complete.
3. Execute all **mtspr** instructions that place values to set up for marking (and counting if that is to be done) into the Performance Monitor Registers *except* for the counting enable in MMCR0 and the sample_enable in MMCRA.
4. Wait for all previous **mtspr** instructions to complete, and then execute the **mtspr** instruction to enable counting in MMCR0.
5. Wait for the previous **mtspr** instruction to complete, and then execute the **mtspr** instruction that enables marking in MMCRA.
6. Execute a synchronizing instruction or wait for the last **mtspr** instruction to complete.
7. Exit supervisor mode.
8. Start the program for which marking (and counting) is to be done.

Note: Any marked or counted events that occur after the performance monitor counting is enabled in supervisor mode, but before the program under study is entered, will be included in the overall mark or count activity. In the same way, any counter events that occur after the program under study is exited, but before the performance monitor marking or counting is disabled, will also be included in the overall mark/count activity.

When the program being marked or counted completes, the following steps disable performance monitor marking or counting:

1. Enter supervisor mode.
2. Wait for the previous **mtspr** instructions to complete, and then execute the **mtspr** instruction that disables marking in MMCRA.
3. Wait for the previous **mtspr** instruction to complete, and then execute the **mtspr** instruction to disable counting in MMCR0.
4. Execute a synchronizing instruction or wait for the last **mtspr** instruction to complete.
5. Wait for the counting operations that are in flight to complete and then execute the **mfspr** instructions to read the values from the performance monitor counters.
6. Execute a synchronizing instruction or wait for the last **mfspr** instruction to complete.
7. Exit supervisor mode.

Notes:

If marking is disabled while a marked instruction is still active, the performance monitor will finish that marking operation in the usual way. The marking state machine for the performance monitor will then go to idle until marking is again enabled.

Instruction marking is disabled while in Single Step or Branch trace mode.

10.12 SIAR and SDAR Registers

The Sampled Instruction Address Register (SIAR) and the Sampled Data Address Register (SDAR) are used, respectively, to save the effective address of a sampled instruction and the effective address of a storage operand for a sampled instruction when the processor is in either trace-marking mode or instruction-sampling mode. The terms "sampled" and "marked" are use interchangeably.

The processor is in instruction-sampling mode whenever MMCRA[63] equals '1' and MSR[SE] and MSR[BE] equal '0'. The processor is in a well-defined trace-marking mode whenever MMCRA[63] equals '0' and either MSR[SE] equals '1' or MSR[BE] equals '1', or MSR[SE] and MSR[BE] equal '1'.

Note: If instruction sampling is not disabled during trace-marking by setting MMCRA[63] to '0', results are undefined.

The contents of the SIAR and SDAR depend on the marking modes that the processor is in, as explained in the following sections.

10.12.1 Instruction Sampling

When the processor is not in trace-marking mode and instruction-sampling mode is enabled, instruction-sampling mode is active regardless of whether the performance monitor is enabled for any counting activity. In instruction-sampling mode, the performance monitor interacts with the IDU to initiate the instruction-sampling cycle. It then monitors the progress of the sampled instruction as it moves through the instruction pipeline. Each instruction-sampling cycle ends either when the marked instruction completes or when the performance monitor determines that the marked instruction has been flushed from the pipeline.

10.12.1.1 Performance Monitor Exceptions

Performance monitor exceptions occur when a performance monitor counter becomes negative and the counter negative exception is enabled, or when a time-base event occurs and the time-base exception is enabled. When a performance monitor exception occurs, SIAR and SDAR have the following values:

- The SIAR contains the effective address of the last sampled instruction.
- The SDAR is set to the effective address of the storage operand of the last sampled instruction issued to the LSU.
- The effective address of the storage operand contained in the SDAR might be, but need not be, associated with the SIAR instruction as explained above.
- If single step or branch trace (SE/BE) tracing is inactive, the contents of the SIAR and the SDAR are frozen when a performance monitor exception is raised, at which time the hardware sets MMCR0[PMXE] to '0' (locking the SIAR and SDAR).
- If SE/BE tracing is active, the contents of the SIAR, the SDAR, and SRR1[33] as used by the performance monitor exception facility are undefined and can change even when performance monitor exceptions are disabled (MMCR0[PMXE] equals '0').
- If SE/BE tracing is inactive, the contents of SIAR and SDAR remain frozen until software sets MMCR0[PMXE] to '1'. The contents of SIAR and SDAR can be altered by the processor if and only if MMCR0[PMXE] equals '1' provided SE/BE tracing is inactive.
- If the performance monitor exception is enabled and MSR[EE] equals '1', the performance monitor exception condition causes a performance monitor exception to be taken and the value of SRR1[33] indicates whether the contents of the SIAR and SDAR refer to the same instruction.

- If SRR1[33] equals '1', and SE/BE tracing is inactive, and there was only one sampled instruction in the machine, the SDAR and SIAR contents are associated with the same instruction.
- If SE/BE tracing is inactive, when SRR1[33] equals '0' it indicates either that the SIAR and SDAR contents are not associated with the same instruction or that the SIAR instruction had no storage operand.
- After software sets MMCR0[PMXE] to '1' and if SE/BE tracing is inactive, the contents of SIAR and SDAR are undefined with respect to performance monitor exception processing until the next performance monitor exception occurs.
- After software sets MMCR0[PMXE] to '1' and if SE/BE tracing is inactive, the contents of SIAR and SDAR are again available for use by the performance monitor instruction-sampling facility as described above.

10.12.2 Single Step and Branch Trace Marking Mode

When the processor is in SE or BE trace mode, instruction-sampling activity on the performance monitor is disabled, but all other performance monitor activities (except sampling) can be active. In particular, performance monitor count events for marked instructions will still be processed if counting is enabled. The marked events counted when trace mode is active will be for trace-marked events, not for sampled events.

In BE mode, the performance monitor count event called "number of instructions completed" is not accurate because of the way the BE mode trace facility marks instructions to force SIAR updates. It is possible for software to calculate the number of instructions in a basic block by capturing the starting address of the basic block from the SRR0 value of the previous branch and using it together with the ending address of the basic block from the SIAR value.

10.12.2.1 Single Step Trace Mode

If MSR[SE] equals '1', the processor is in single step trace mode. Every instruction is trace-marked. The processor is forced into single instruction mode regardless of the value of the IFU predecode bits. An instruction is trace-marked when it is transferred from the IDU to the ISU and SRR1[33] is reset to zero. If the PowerPC instruction spans multiple groups, the first load/store IOP or the first IOP in the last group is the one marked.

The SIAR is updated at dispatch to contain the address for the trace-marked instruction. The SDAR is updated by the LSU if the PowerPC instruction includes one or more load/store operation. If the SDAR is updated by the LSU, it contains the address of the storage operand for the first load/store IOP. If the SDAR is updated by the LSU, the LSU also causes SRR1[33] to be set to '1' to indicate that the contents of the SIAR and the SDAR are associated with the same trace-marked instruction.

When the trace-marked instruction completes, the processor generates a trace exception. During trace exception processing, the SIAR value is that for the trace-marked instruction that has just successfully completed. During trace exception processing, the SDAR value—if it was updated for this instruction—is that of the first load/store operation of the trace-marked instruction that has just successfully completed. A global flush is performed after the return from trace exception processing.

In the interval between the **rfid** for the trace exception processing for the last completed instruction and the dispatch of the next instruction, the SIAR and SDAR contents represent the last instruction completed and not the instruction that is moving through the IFU to the IDU for dispatch. The next instruction that will be dispatched after the global flush is trace-marked when it is transferred from the IDU to the ISU, SRR1[33] is reset to zero ('0'), and the SIAR value is set for that next marked instruction. The single instruction trace-marking cycle continues as described above.

10.12.2.2 Branch Trace Mode

If MSR[BE] equals '1', the processor is in branch trace mode. Every branch instruction is trace-marked. A branch instruction is trace-marked when it is transferred from the IDU to the ISU.

The SIAR is updated at dispatch to contain the address for the trace-marked branch instruction. The SDAR contents are undefined. When the trace-marked branch instruction completes, the processor generates a trace exception. During trace exception processing, the SIAR value represents the trace-marked branch instruction that has just completed successfully. A global flush is performed after the return from trace exception processing.

In the interval between the **rfid** for the trace exception processing for the last completed branch instruction and the dispatch of the next branch instruction, the SIAR contents represent the last branch instruction completed. The next branch instruction decoded by the IDU after the global flush is trace-marked when it is transferred from the IDU to the ISU and the SIAR value is set for that next marked branch instruction. The branch trace-marking cycle continues as described above.

10.12.3 Comparison to Previous PowerPC Processors

According to the PowerPC Architecture, the SIAR contains the effective address of an instruction that was executing *around* the time of a performance monitor exception. On previous processors that had relatively short pipelines and few instructions in flight, the sampled instruction was at most 20 or so instructions away from the instruction that caused the exception. On the 970FX microprocessor, with the potential for over a hundred instructions in flight, that distance grows. The theoretical maximum is once every 200 - 250 instructions, while the likely distance is 50 - 75 instructions.

Performance profiling tools that use performance monitor events to determine when the SIAR and SDAR are read (for example, read SIAR every 100 L1 D-cache misses) can profile based on any performance monitor event. To assure accuracy, however, only *sampled* events should be profiled. These are a subset of all events that are caused by sampled instructions. The SIAR is set by a sampled instruction, so you can be fairly sure that when an exception caused by a sampled event (a counter overflowing for example), the SIAR is pointing to the *exact* instruction that caused it. In this case, the 970FX microprocessor is more accurate than previous processors. If you profile on non-sampled events, you cannot be sure that the exception was caused by the instruction (group actually) pointed to by the SIAR. The offending instruction was executing *around* the sampled instruction, depending on the event, probably within 50 - 100 instructions.

10.13 Thresholding

Thresholding can be used to obtain counts of the number of marked instructions for which the execution time between a designated start/end pipeline event pair exceeds a specified threshold value. Only one marked instruction is active in the 970FX processing unit pipeline at a time, and only one threshold value can be used for comparison with the selected start/end event pair. The start and end events that can be used for thresholding are shown in *Figure 10-20 Performance Monitor Threshold Logic* on page 249. The values of the respective threshold start/end bit fields in MMCRA[THRSTRT,THREND] are shown in *Table 10-29 Start and End Event Select Bits and the Performance Monitor Threshold Logic* on page 250. The threshold value is specified in the MMCR1[THRSHOLD] field. The threshold value can be further scaled by HID0[13]. If HID0[13] equals ‘0’, it causes the thresholder to count every processor cycle. If HID0[13] equals ‘1’, it causes the thresholder to count every 32 processor cycles. For a marked instruction moving through the 970FX processing unit pipeline, the events that can be used for threshold start/end measurement occur in the following order: marked in IDU, dispatch, issue, finish, complete.

Once a pair of start/end threshold events is selected and the start event occurs, the threshold facility begins decrementing from the threshold value and continues to decrement until either the decrementer times out or the end event occurs. If the decrementer times out and if the threshold logic event is selected for counting, the threshold logic event counter is incremented. Both the thresholder time out and the occurrence of the end event cause the threshold decrementer to be reset to the threshold value. The thresholder begins decrementing when the next start event occurs.

Threshold start/end pairs must be selected in a manner that represents a reasonable scenario. For example, a start event that is the same as the end event will not provide useful threshold event count information regardless of the threshold value selected. A start event that occurs later in the pipeline than the end event will not give a useful measure of the transit time of a marked instruction through the pipeline. The results of unreasonable threshold start/end event selections may produce undefined results.

Figure 10-20. Performance Monitor Threshold Logic

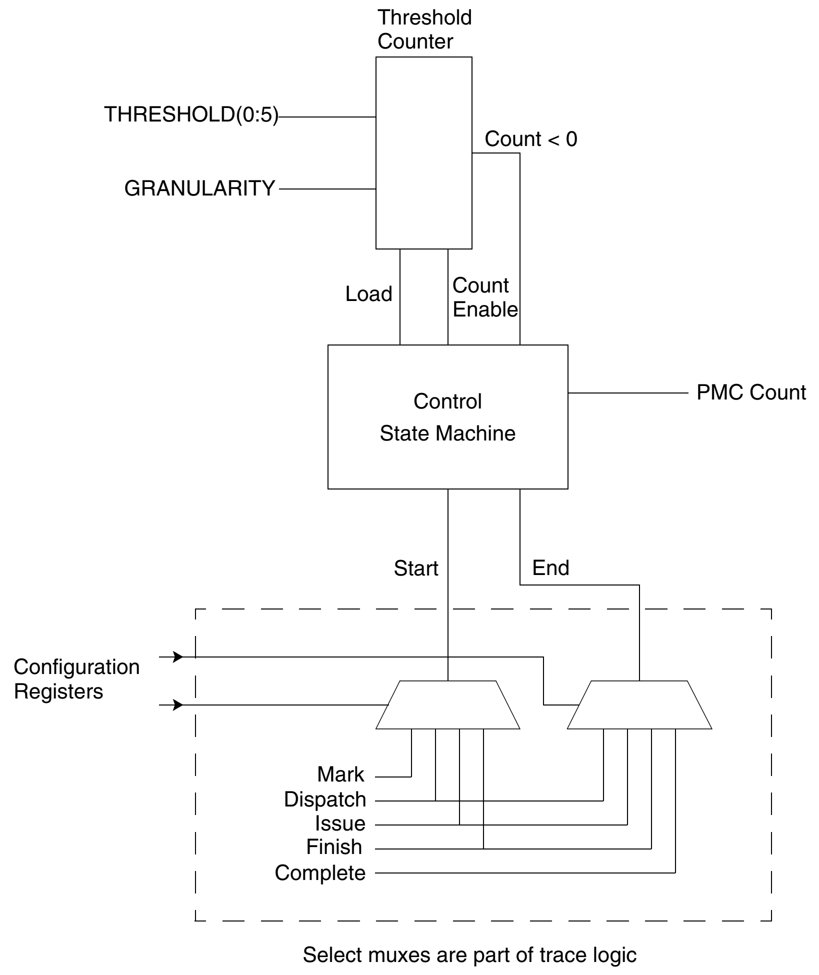


Table 10-29. Start and End Event Select Bits and the Performance Monitor Threshold Logic

MMCR1 [THRSTRT] Value	Threshold Start Event Selected	MMCR1 [THREND] Value	Threshold End Event Selected
000	No start event	000	No end event
001	Group marked in IDU	001	No end event
010	Marked group dispatched	010	Marked group dispatched
011	Marked group issued	011	Marked group issued
100	Marked group finish[1]	100	Marked group finish
101	Marked group complete[1]	101	Marked group complete
110	No start event	110	No end event
111	No start event	111	No end event

1. An instruction that has finished but not completed has gone all the way through the pipeline, and the renamed registers have been updated with new values. However, it is still sitting in the completion queue. When an instruction completes, the architected registers are updated with the values from the renamed registers.

10.14 Detailed Event Information

Table 10-30. Detailed Event Descriptions (Page 1 of 8)

Event Description	Detailed Description
CLB has x where x is 0 to 8	The cache line buffer (CLB) is a 4-instruction wide by 8-instruction deep buffer between the fetch unit and the dispatch unit. This signal indicates how many entries, each of which is 4-instructions wide, are occupied at any given time.
Valid instructions available, but IFU held by BIQ or IDU	This signal is asserted each time either the IDU is full or the branch instruction queue (BIQ) is full.
Branch execution issue valid	This signal is asserted each time the ISU issues a branch instruction.
Branch miss predict due to CR value	This signal is asserted when the branch execution unit detects a branch mispredict because the CR value is the opposite of the predicted value. This signal is asserted after a branch issue event and results in a branch redirect flush if not overridden by a flush of an older instruction.
Branch miss predict due to target address prediction	This signal is asserted each time the branch execution unit detects an incorrect target address prediction. This signal is asserted after a valid branch execution unit issue and causes a branch mispredict flush unless a flush is detected from an older instruction.
CR mapper full	The ISU sends a signal indicating that the CR mapper cannot accept any more groups. Dispatch is stopped. **Note:** This condition indicates that a pool of the mapper is full but the entire mapper may not be.
Tablewalk duration	This signal is asserted every cycle when a tablewalk is active. While a tablewalk is active, any request attempting to access the TLB is rejected and retried.
L1 D-cache entries invalidated from L2	A D-cache invalidated was received from the L2 because a line in L2 was castout.
Out of streams	A new prefetch stream was detected, but no more stream entries were available.
D-SLB miss	An SLB miss for a data request occurred. When there is a miss in the SLB, the operating system must reload the buffer with the information needed for a hit so that the transaction can proceed. Therefore, an SLB miss causes an interrupt (trap) to indicate to the operating system that it needs to resolve the problem.
D-TLB miss	A TLB miss for a data request occurred. Requests that miss the TLB may be retried until the instruction is in the next-to-complete group (unless HID4 is set to allow speculative tablewalks). This may result in multiple TLB misses for the same instruction.
Duration MSR[EE] equals '0'	The ISU sends the MSR[EE} bit to the PMU. It is up to the performance monitor to count the cycles while this bit is '0'.
MSR[EE] equals '0' and interrupt pending	The ISU sends the MSR[EE] bit and a signal indicating that an interrupt is pending to the PMU. It is up to the performance monitor to count the cycles while MSR[EE] equals '0' and the interrupt is pending.
FPR mapper full	The ISU sends a signal indicating that the FPR mapper cannot accept any more groups. Dispatch is stopped. **Note:** This condition indicates that a pool of mappers is full but the entire mapper may not be.
FPU0 add, mult, sub, compare, fsel	This signal is active for one cycle when FPU0 is executing an add, mult, sub, compare, or fsel kind of instruction. The instruction could be **fadd***, **fmul***, **fsub***, **fcmp****, or **fsel**.
FPU0 denormalized operand	This signal is active for one cycle when one of the operands is denormalized.
FPU0 divide	This signal is active for one cycle at the end of the microcode executed when FPU0 is executing a divide instruction. The instruction could be **ffdiv**, **fdivs**, **fdiv.**, or **fdivs.**
FPU0 estimate	This signal is active for one cycle when FPU0 is executing one of the estimate instructions. The instruction could be **fres*** or **frsqrte*** where **xyz*** means **xyz** or **xyz.**
FPU0 finished and produced a result	This signal only indicates finish, not completion.
FPU0 mult-add	This signal is active for one cycle when FPU0 is executing a multiply-add kind of instruction. The instruction could be **fmadd***, **fnmadd***, **fmsub***, or **fnmsub*** where **xyz*** means **xyz**, **xyzs**, **xyz.**, **xyzs.**

Table 10-30. Detailed Event Descriptions (Page 2 of 8)

Event Description	Detailed Description
FPU0 move, estimate	This signal is active for one cycle when FPU0 is executing a move kind of instruction or one of the estimate instructions. The instruction could be **fmr***, **fneg***, **fabs***, **fnabs***, **fres***, or **frsqrte*** where **xyz*** means **xyz** or **xyz.**.
FPU0 FPSCR	This signal is active for one cycle when FPU0 is executing an FPSCR move-related instruction. The instruction could be **mtfsfi***, **mtfsb0***, **mtfsb1***, **mffs***, **mtfsf***, or **mcrsf*** where **xyz*** means **xyz**, **xyzs**, **xyz.**, or **xyzs.**.
FPU0 round, convert	This signal is active for one cycle when FPU0 is executing **frsp** or a convert kind of instruction. The instruction could be **frsp***, **fcfid***, or **fcti*** where **xyz*** means **xyz**, **xyzs**, **xyz.**, or **xyzs.**.
FPU0 square root	This signal is active for one cycle at the end of the microcode executed when FPU0 is executing a square root instruction. The instruction could be **fsqrt*** where **xyz*** means **xyz**, **xyzs**, **xyz.**, **xyzs.**.
FPU0 issue queue full	The issue queue for FPU 0 cannot accept any more instructions. Issue is stopped.
FPU0 single precision	This signal is active for one cycle when FPU0 is executing a single-precision instruction.
FPU0 stall 3	This signal indicates that FPU0 has generated a stall in pipe 3 due to overflow, underflow, massive cancel, convert to integer (sometimes), or convert from integer (always). This signal is active during the entire duration of the stall.
FPU0 store	This signal is active for one cycle when FPU0 is executing a store instruction.
FPU1 add, mult, sub, compare, fsel	This signal is active for one cycle when FPU1 is executing an add, mult, sub, compare, or fsel kind of instruction. The instruction could be **fadd***, **fmul***, **fsub***, **fcmp****, or **fsel** where **xyz*** means **xyz**, **xyzs**, **xyz.**, **xyzs.** and **xyz**** means **xyzu** and **xyzo**.
FPU1 denormalized operand	This signal is active for one cycle when one of the operands is denormalized.
FPU1 divide	This signal is active for one cycle at the end of the microcode executed when FPU1 is executing a divide instruction. The instruction could be **fdiv**, **fdivs**, **fdiv.**, or **fdivs.**
FPU1 estimate	This signal is active for one cycle when FPU1 is executing one of the estimate instructions. The instruction could be **fres*** or **frsqrte*** where **xyz*** means **xyz** or **xyz.** .
FPU1 finished and produced a result	This signal only indicates finish, not completion.
FPU1 mult-add	This signal is active for one cycle when FPU1 is executing a multiply-add kind of instruction. The instruction could be **fmadd***, **fnmadd***, **fmsub***, or **fnmsub*** where **xyz*** means **xyz**, **xyzs**, **xyz.**, and **xyzs.**.
FPU1 move, estimate	This signal is active for one cycle when FPU1 is executing a move kind of instruction or one of the estimate instructions. The instruction could be **fmr***, **fneg***, **fabs***, **fnabs***, **fres***, or **frsqrte*** where **xyz*** means **xyz** or **xyz.**.
FPU1 round, convert	This signal is active for one cycle when FPU1 is executing **frsp** or convert kind of instruction. The instruction could be **frsp***, **fcfid***, or **fcti*** where **xyz*** means **xyz**, **xyzs**, **xyz.**, **xyzs.**.
FPU1 square root	This signal is active for one cycle at the end of the microcode executed when FPU1 is executing a square root instruction. The instruction could be **fsqrt*** where **xyz*** means **xyz**, **xyzs**, **xyz.**, **xyzs.**.
FPU1 issue queue full	The issue queue for FPU 1 cannot accept any more instructions. Issue is stopped.
FPU1 single precision	This signal is active for one cycle when FPU1 is executing a single-precision instruction.
FPU1 stall 3	This signal indicates that FPU1 has generated a stall in pipe 3 due to overflow, underflow, massive cancel, convert to integer (sometimes), or convert from integer (always). This signal is active during the entire duration of the stall.
FPU1 store	This signal is active for one cycle when FPU1 is executing a store instruction.
FXU0/LSU0 issue queue full	The issue queue for FXU/LSU unit 0 cannot accept any more instructions. Issue is stopped.
FXU1/LSU1 issue queue full	The issue queue for FXU/LSU 1 cannot accept any more instructions. Issue is stopped.
FXU0 produced a result	The FXU0 finished an instruction and produced a result.

Table 10-30. Detailed Event Descriptions (Page 3 of 8)

Event Description	Detailed Description
FXU1 produced a result	The FXU1 finished an instruction and produced a result.
GPR mapper full	The ISU sends a signal indicating that the GPR mapper cannot accept any more groups. Dispatch is stopped. **Note:** This condition indicates that a pool of mapper is full but the entire mapper may not be.
Dispatch blocked by scoreboard	The ISU sends a signal indicating that dispatch is blocked by the scoreboard.
Dispatch reject	Dispatch successful equals *dispatch_valid* and one cycle later ~*dispatch_reject*.
Dispatch valid	The ISU sends *dispatch_valid* and *dispatch_reject* signals to the PMU. It is up to the performance monitor to look at these signals to count the number of dispatch groups.
Instruction prefetch installed in prefetch buffer	This signal is asserted when a prefetch buffer entry (line) is allocated but the request is not a demand fetch.
Instruction prefetch request	Asserted when a non-canceled prefetch is made to the CIU.
Translation written to I-ERAT	This signal is asserted each time the I-ERAT is written. This indicates that an ERAT miss has been serviced. ERAT misses will initiate a sequence resulting in the ERAT being written. ERAT misses that are later ignored will not be counted unless the ERAT is written before the instruction stream is changed, This should be a fairly accurate count of ERAT missed (best available).
Instructions dispatched count	The ISU sends the number of instructions dispatched.
Valid instruction available	Asserted each cycle when the IFU sends at least one instruction to the IDU.
I-SLB miss	An SLB miss for an instruction fetch has occurred.
I-TLB miss	A TLB miss for an Instruction fetch has occurred.
L1 reload data source valid	The data source information is valid.
L1 prefetches	A request to prefetch data into the L1 was made.
L2 Prefetch	A request to prefetch data into the L2 was made.
larx executed side 0	An **larx** (**lwarx** or **ldarx**) was executed on side 0 (there is no corresponding unit 1 event since **larx** instructions can only execute on unit 0).
L1 D-cache load miss side 0	A load, executing on unit 0, missed the D-cache.
L1 D-cache store side 1	A store executed on unit 1.
L1 D-cache load miss side 1	A load, executing on unit 1, missed the D-cache.
L1 D-cache load side 0	A load executed on unit 0.
LR/CTR mapper full	The ISU sends a signal indicating that the LR/CTR mapper cannot accept any more groups. Dispatch is stopped. **Note:** This condition indicates that a pool of the mapper is full but the entire mapper may not be.
LMQ full	The LMQ was full.
LMQ LHR merge	A D-cache miss occurred for the same real cache line address as an earlier request already in the load miss queue and was merged into the LMQ entry.
LMQ slot 0 allocated	The first entry in the LMQ was allocated.
LMQ slot 0 valid	This signal is asserted every cycle when the first entry in the LMQ is valid. The LMQ has eight entries that are allocated on a FIFO basis.
LRQ full	The ISU sends this signal when the LRQ is full.
LRQ slot 0 allocated	LRQ slot zero was allocated.
LRQ slot 0 valid	This signal is asserted every cycle that slot zero of the store request queue is valid. The SRQ is 32 entries long and is allocated on a round-robin basis.

Table 10-30. Detailed Event Descriptions (Page 4 of 8)

Event Description	Detailed Description
SRQ full	The ISU sends this signal when the SRQ is full.
SRQ slot 0 allocated	SRQ slot zero was allocated.
SRQ slot 0 valid	This signal is asserted every cycle that slot zero of the store request queue is valid. The SRQ is 32 entries long and is allocated round-robin.
SRQ sync duration	This signal is asserted every cycle when a sync is in the SRQ.
LSU busy side 0	LSU 0 is busy rejecting instructions.
D-ERAT miss side 0	A data request (load or store) from LSU 0 missed the ERAT. Requests that miss the D-ERAT are rejected and retried until the request hits in the ERAT. This may result in multiple ERAT misses for the same instruction.
Flush from LRQ SHL, LHL side 0	A load was flushed by unit 1 because a a younger load executed before an older store executed and they had overlapping data. Alternatively, two loads executed out-of-order, they had byte overlap, and there was a snoop in between to an overlapped byte.
Flush SRQ LHS side 0	A store was flushed because a younger load hits an older store that is already in the SRQ or in the same group.
Flush unaligned load side 0	A load was flushed from unit 1 because it was unaligned (crossed a 64-byte boundary, or a 32-byte boundary if it missed the L1).
Flush unaligned store side 0	A store was flushed from unit 1 because it was unaligned.
Floating-point load side 0	A floating-point load was executed from LSU unit 0.
SRQ store forwarding side 0	Data from a store instruction was forwarded to a load on unit 0.
LSU busy side 1	LSU 0 is busy rejecting instructions.
D-ERAT miss side 1	A data request (load or store) from LSU 1 missed the ERAT. Requests that miss the D-ERAT are rejected and retried until the request hits in the ERAT. This may result in multiple ERAT misses for the same instruction.
Flush from LRQ SHL, LHL side 1	A load was flushed by unit 1 because a a younger load executed before an older store executed and they had overlapping data. Alternatively, two loads executed out-of-order, they had byte overlap, and there was a snoop in between to an overlapped byte.
Flush SRQ LHS side 1	A store was flushed because younger load hits an older store that is already in the SRQ or in the same group.
Flush unaligned load side 1	A load was flushed from unit 1 because it was unaligned (crossed a 64-byte boundary, or a 32-byte boundary if it missed the L1).
Flush unaligned store side 1	A store was flushed from unit 1 because it was unaligned (crossed a 4KB boundary).
Floating-point load side 1	A floating-point load was executed from LSU 1.
Marked IMR reload	A Data L1 cache reload occurred due to marked load.
Marked L1 reload data source valid	The source information is valid and is for a marked load.
Marked L1 D-cache load miss side 0	A marked load, executing on unit 0, missed the D-cache.
Marked L1 D-cache load miss side 1	A marked load, executing on unit 1, missed the D-cache.
Marked SRQ valid	This signal is asserted every cycle when a marked request is resident in the store request queue.
Marked flush from LRQ SHL, LHL side 0	A marked load was flushed by unit 0 because a a younger load executed before an older store executed and they had overlapping data. Alternatively, two loads executed out-of-order, they have byte overlap, and there was a snoop in between to an overlapped byte.
Marked flush SRQ LHS side 0	A marked store was flushed because a younger load hits an older store that is already in the SRQ or in the same group.
Marked flush unaligned store side 0	A marked store was flushed from unit 0 because it was unaligned.

Table 10-30. Detailed Event Descriptions (Page 5 of 8)

Event Description	Detailed Description
Marked flush unaligned load side 0	A marked load was flushed from unit 0 because it was unaligned (crossed a 64-byte boundary, or a 32-byte boundary if it missed the L1).
LSU side 0 finished IMR	LSU unit 0 finished a marked instruction.
Marked flush from LRQ SHL, LHL side 1	A marked load was flushed by unit 1 because a a younger load executed before an older store executed and they had overlapping data. Alternatively, two loads executed out-of-order, they had byte overlap, and there was a snoop in between to an overlapped byte.
Marked flush SRQ LHS side 1	A marked load was flushed by unit 1 because a a younger load executed before an older store executed and they had overlapping data. Alternatively, two loads executed out-of-order, they had byte overlap, and there was a snoop in between to an overlapped byte.
Marked flush unaligned load side 1	A marked load was flushed from unit 1 because it was unaligned (crossed a 64-byte boundary, or a 32-byte boundary if it missed the L1).
Marked flush unaligned store side 1	A marked store was flushed from unit 1 because it was unaligned (crossed a 4K page boundary).
LSU side 1 finished IMR	LSU unit 1 finished a marked instruction.
Marked L1 D-cache store miss	A marked store missed the D-cache.
Marked **stcx** fail	A marked **stcx** (**stwcx** or **stdcx**) failed.
Snoop **tlbie**	A **tlbie** was snooped from another processor.
L1 D-cache store miss	A store missed the D-cache.
L1 D-cache store miss	A store missed the D-cache.
L1 D-cache store side 0	A store executed on unit 0.
L1 D-cache load side 1	A load executed on unit 1.
stcx failed	An **stcx** (**stwcx** or **stdcx**) failed.
stcx passed	An **stcx** (**stwcx** or **stdcx**) instruction was successful.
XER mapper full	The ISU sends a signal indicating that the XER mapper cannot accept any more groups. Dispatch is stopped. **Note:** This condition indicates that a pool of the mapper is full but the entire mapper may not be.
No instructions fetched	No instructions were fetched this cycle (due to IFU hold, redirect, or I-cache miss).
One or more PowerPC instruction completed	A group containing at least one PowerPC instruction completed. For microcoded instructions that span multiple groups, this will only occur once.
BR issue queue full	The ISU sends a signal indicating that the issue queue that feeds the IFU BR unit cannot accept any more groups (the queue is full of groups).
CR issue queue full	The ISU sends a signal indicating that the issue queue that feeds the IFU CR unit cannot accept any more groups (the queue is full of groups).
Processor cycles	Processor cycles.
Data loaded from L2	Data L1 cache was reloaded from the local L2 due to a demand load.
Data loaded from memory	Data L1 cachewas reloaded from memory due to a demand load.
New stream allocated	A new prefetch stream was allocated.
External interrupts	An external interrupt occurred.
FPU executed add	This signal is active for one cycle when FPU0 is executing an add, mult, sub, compare, or fsel kind of instruction. The instruction could be **fadd***, **fmul***, **fsub***, **fcmp****, or **fsel** where **xyz*** means **xyz**, **xyzs**, **xyz.**, **xyzs.** and **xyz**** means **xyzu** and **xyzo.** Combined Unit 0 + Unit 1.

Table 10-30. Detailed Event Descriptions (Page 6 of 8)

Event Description	Detailed Description
FPU received denormalized data	This signal is active for one cycle when one of the operands is denormalized. Combined Unit 0 + Unit 1.
FPU executed FDIV instruction	This signal is active for one cycle at the end of the microcode executed when FPU0 is executing a divide instruction. The instruction could be **fdiv**, **fdivs**, **fdiv**. **fdivs**. Combined Unit 0 + Unit 1.
FPU executed FEST instruction	This signal is active for one cycle when executing one of the estimate instructions. The instruction could be **fres*** or **frsqrte*** where **xyz*** means **xyz** or **xyz.** Combined Unit 0 + Unit 1.
FPU produced a result	FPU finished and produced a result. This only indicates finish, not completion. Combined Unit 0 + Unit 1.
FPU executed multiply-add instruction	This signal is active for one cycle when FPU0 is executing a multiply-add kind of instruction. The instruction could be **fmadd***, **fnmadd***, **fmsub***, or **fnmsub*** where **xyz*** means **xyz**, **xyzs**, **xyz.**, **xyzs.**. Combined Unit 0 + Unit 1.
FPU executing FMOV or FEST instructions	This signal is active for one cycle when executing a move kind of instruction or one of the estimate instructions. The instruction could be **fmr***, **fneg***, **fabs***, **fnabs***, **fres***, or **frsqrte*** where **xyz*** means **xyz** or **xyz.**. Combined Unit 0 + Unit 1.
FPU executed FRSP or FCONV instructions	This signal is active for one cycle when executing **frsp** or a convert kind of instruction. The instruction could be **frsp***, **fcfid***, **fcti*** where **xyz*** means **xyz**, **xyzs**, **xyz.**, **xyzs.**. Combined Unit 0 + Unit 1.
FPU executed FSQRT instruction	This signal is active for one cycle at the end of the microcode executed when FPU0 is executing a square root instruction. The instruction could be **fsqrt*** where **xyz*** means **xyz**, **xyzs**, **xyz.**, **xyzs.**. Combined Unit 0 + Unit 1.
Cycles FPU issue queue full	Cycles when one or both FPU issue queues are full.
FPU executed single-precision instruction	FPU is executing a single-precision instruction. Combined Unit 0 + Unit 1.
FPU stalled in pipe 3	FPU has generated a stall in pipe 3 due to overflow, underflow, massive cancel, convert to integer (sometimes), or convert from integer (always). This signal is active during the entire duration of the stall. Combined Unit 0 + Unit 1.
FPU executed store instruction	FPU is executing a store instruction. Combined Unit 0 + Unit 1.
Cycles FXLS queue is full	Cycles when one or both FXU/LSU issue queues are full.
FXU busy	FXU0 and FXU1 are both busy.
FXU produced a result	The fixed-point unit (Unit 0 + Unit 1) finished a marked instruction. Instructions that finish may not necessarily complete.
FXU idle	FXU0 and FXU1 are both busy.
FXU0 busy FXU1 idle	FXU0 is busy while FXU1 is idle.
FXU1 busy FXU0 idle	FXU0 is idle while FXU1 is busy.
Cycles GCT empty	The global completion table is completely empty.
Completion table full	The ISU sends a signal indicating that the GCT is full.
Group completed	A group completed. Microcoded instructions that span multiple groups will generate this event once per group.
Group dispatches	A group was dispatched.
Group dispatch rejected	A group that previously attempted dispatch was rejected.
Group dispatch success	Number of groups successfully dispatched (not rejected).
Group marked in IDU	A group was sampled (marked).
Instructions completed	Number of eligible instructions that completed.

Table 10-30. Detailed Event Descriptions (Page 7 of 8)

Event Description	Detailed Description
Instructions fetched from L1	An instruction fetch group was fetched from L1. Fetch groups can contain up to eight instructions.
Instructions fetched from L2	An instruction fetch group was fetched from L2. Fetch groups can contain up to eight instructions.
Instructions fetched from memory	An instruction fetch group was fetched from memory. Fetch groups can contain up to eight instructions.
Instructions fetched from prefetch	An instruction fetch group was fetched from the prefetch buffer. Fetch groups can contain up to eight instructions.
Cycles is L1 write active	This signal is asserted each cycle a cache write is active.
larx executed	An **larx** (**lwarx** or **ldarx**) was executed. This is the combined count from LSU0 + LSU1, but these instructions only execute on LSU0.
L1 D-cache load misses	Total Data L1 cache load references that miss the Data L1 cache.
L1 D-cache load references	Total Data L1 cache load references.
LSU busy	LSU (unit 0 + unit 1) is busy rejecting instructions.
D-ERAT misses	Total D-ERAT misses (Unit 0 + Unit 1). Requests that miss the D-ERAT are rejected and retried until the request hits in the ERAT. This may result in multiple ERAT misses for the same instruction.
LRQ flushes	A load was flushed because a younger load executed before an older store executed and they had overlapping data. Alternatively, two loads executed out-of-order, they had byte overlap, and there was a snoop in between to an overlapped byte.
SRQ flushes	A store was flushed because a younger load hits an older store that is already in the SRQ or in the same group.
LRQ unaligned load flushes	A load was flushed because it was unaligned (crossed a 64-byte boundary, or a 32-byte boundary if it missed the L1).
SRQ unaligned store flushes	A store was flushed because it was unaligned.
LSU executed floating-point load instruction	
Cycles LMQ and SRQ empty	Cycles when both the LMQ and SRQ are empty (LSU is idle).
Cycles SRQ empty	The store request queue is empty.
SRQ store forwarding side 1	Data from a store instruction was forwarded to a load on unit 1.
Marked instruction BRU processing finished	The branch unit finished a marked instruction. Instructions that finish may not necessarily complete.
Marked instruction CRU processing finished	The condition register unit finished a marked instruction. Instructions that finish may not necessarily complete.
Marked data loaded from L2	Data L1 cache was reloaded with modified (M) data from the L2 of a chip on this MCM due to a marked load.
Marked data loaded from memory	Data L1 cache was reloaded with modified (M) data from the L2 of another MCM due to a marked load.
Marked instruction FPU processing finished	One of the floating-point units finished a marked instruction. Instructions that finish may not necessarily complete.
Marked instruction FXU processing finished	One of the fixed-point units finished a marked instruction. Instructions that finish may not necessarily complete.
Marked group completed	A group containing a sampled instruction completed. Microcoded instructions that span multiple groups will generate this event once per group.
Marked group dispatched	A group containing a sampled instruction was dispatched.

Table 10-30. Detailed Event Descriptions (Page 8 of 8)

Event Description	Detailed Description
Marked group issued	A sampled instruction was issued.
Marked group completion timeout	The sampling timeout expired indicating that the previously sampled instruction is no longer in the processor.
Marked instruction finished	One of the execution units finished a marked instruction. Instructions that finish may not necessarily complete.
Marked L1 D-cache load misses	
Marked instruction LSU processing finished	One of the load/store units finished a marked instruction. Instructions that finish may not necessarily complete.
Marked LRQ flushes	A marked load was flushed because a younger load executed before an older store executed and they had overlapping data. Alternatively, two loads executed out-of-order, they have byte overlap, and there was a snoop in between to an overlapped byte.
Marked SRQ flushes	A marked store was flushed because a younger load hits an older store that is already in the SRQ or in the same group.
Marked unaligned load flushes	A marked load was flushed because it was unaligned (crossed a 64-byte boundary, or a 32-byte boundary if it missed the L1).
Marked unaligned store flushes	A marked store was flushed because it was unaligned.
Marked store instruction completed	A sampled store has completed (data home).
Marked store completed with intervention	A marked store previously sent to the memory subsystem completed (data home) after requiring intervention.
Marked store sent to storage subsystem	A sampled store has been sent to the memory subsystem.
Run cycles	Processor cycles gated by the run latch.
L1 D-cache store references	Total Data L1 cache store references.
Completion stopped	The RAS unit has signaled completion to stop.
Time-base bit transition	Occurs when the selected time-base bit (as specified in MMCR0[TBSEL]) transitions from ‘0’ to ‘1’.
Threshold timeout	The threshold timer expired.
Work held	The RAS unit has signaled completion to stop and there are groups waiting to complete.

11. System Design

11.1 Power-On Reset

The 970FX requires a more complicated power-on reset sequence than earlier PowerPC processors. The processor initialization is handled by the pervasive logic which is controlled either by the I^{2}C or JTAG from an external service processor. This service processor, usually a microcontroller, must initiate and monitor on-chip initialization and test sequences to ensure proper operation.

The power-on reset (POR) unit was developed to allow a power-on reset of the 970FX processor with almost no external support. This procedure is called automatic power-on reset (aPOR). After power ramping and PLL synchronization a predefined set of instructions initializes the processor, runs the array tests, sets up the configuration switches, and synchronizes the PI interfaces. Progress through the aPOR sequence is saved in the POR Status Register accessible by the service element (SE) controlling the power-on reset.

For more detailed information, see the *IBM PowerPC 970FX Power-On Reset Application Note*.

11.1.1 Overview

The power-on reset of a 970FX system goes through seven separate phases, numbered IPL0-6 as shown in *Figure 11-1*. The sequence actually begins with the ramping of power and clocks prior to IPL0. Once power supplies and clocks are stable, $\overline{\text{HRESET\_B}}$ and $\overline{\text{BYPASS\_B}}$ should be asserted to reset the on-chip logic and PLL respectively.

In addition to the initialization of the 970FX, the service processor (SPU) handles other functions via the I^{2}C. Initialization/configuration of the North Bridge, DIMMS, etc. must all be handled as part of the system power-on reset sequence, however initialization of these devices is beyond the scope of this section.

The sequence shown in*Table 11-1* assumes the automatic or "normal" mode of power-on reset. In this mode each step in the sequence is preprogrammed and only minimal intervention by the service processor is required to move the sequence along.

There is also a "debug" mode available that is initiated by asserting "GPULDBG" high (see *Table 11-3*). In this mode, the GPUL will not automatically step through the preprogrammed sequence, but instead will pause after each step to allow service procesor or JTAG intervention. In this mode, the service processor must initiate a continue after each power-on reset step to continue the sequence.

Most developers will choose to use the debug mode to develop the code for their Service Processing Unit. It requires a few more milliseconds to send each continue command, but provides greater visibility to the machine state from the Service Processor.

11.1.2 Power-On Reset SPU Hardware Considerations

It is assumed that 970FX systems will include a service processor (SPU) which generally consists of a low cost microcontroller. This microcontroller is responsible for hardware initialization of the 970FX and the North Bridge and can also be used to manage and supervise other system functions, like fans.

At a minimum, the SPU needs to be able to assert $\overline{\text{HRESET\_B}}$ and $\overline{\text{BYPASS\_B}}$ on the 970FX and should also have either a dedicated I^{2}C bus master to initialize the system, or general purpose I/O pins (GPIO) that can be used to implement an I^{2}C bus master.

Most system bring-up and debug processes should allow the SPU to control the GPULDBG pin to allow the power-on reset sequence to run in debug mode. An alternative would be to provide a jumper to allow that pin to be strapped in debug mode during bring-up.

11.1.2.1 I^{2}C Bus Speed

The 970FX bus speed on I2C will be limited to approximately 50Khz unless the service processor can write the value 0x0083F000.00000000 to the SCOM address 0x600400 (Clock Ratio Register).

Note: This SCOM write must occur after the second continue is sent during power-on reset. Until this SCOM write occurs, the I^{2}C bus speed will be limited to 50KHz. Once the SCOM has been written the I^{2}C bus speed may be increased to 100KHz.

11.1.2.2 Service Processor Firmware Bring-up and Development

During early bring-up and firmware development, use of an I^{2}C controller/emulator is recommended. These tools are available from multiple sources and convert a PC's RS232 port or USB port to I^{2}C. Software on the PC can be used to communicate with the I^{2}C bus in the system in order to develop and debug the power-on reset sequence. Once the system has been brought up using the I^{2}C interface, this information can be used to develop firmware for the service processor.

The SPU firmware development will be easier if the system design provides some mechanism for easily modifying the firmware in the system.

Figure 11-1. 970FX Power-On Reset General Overview

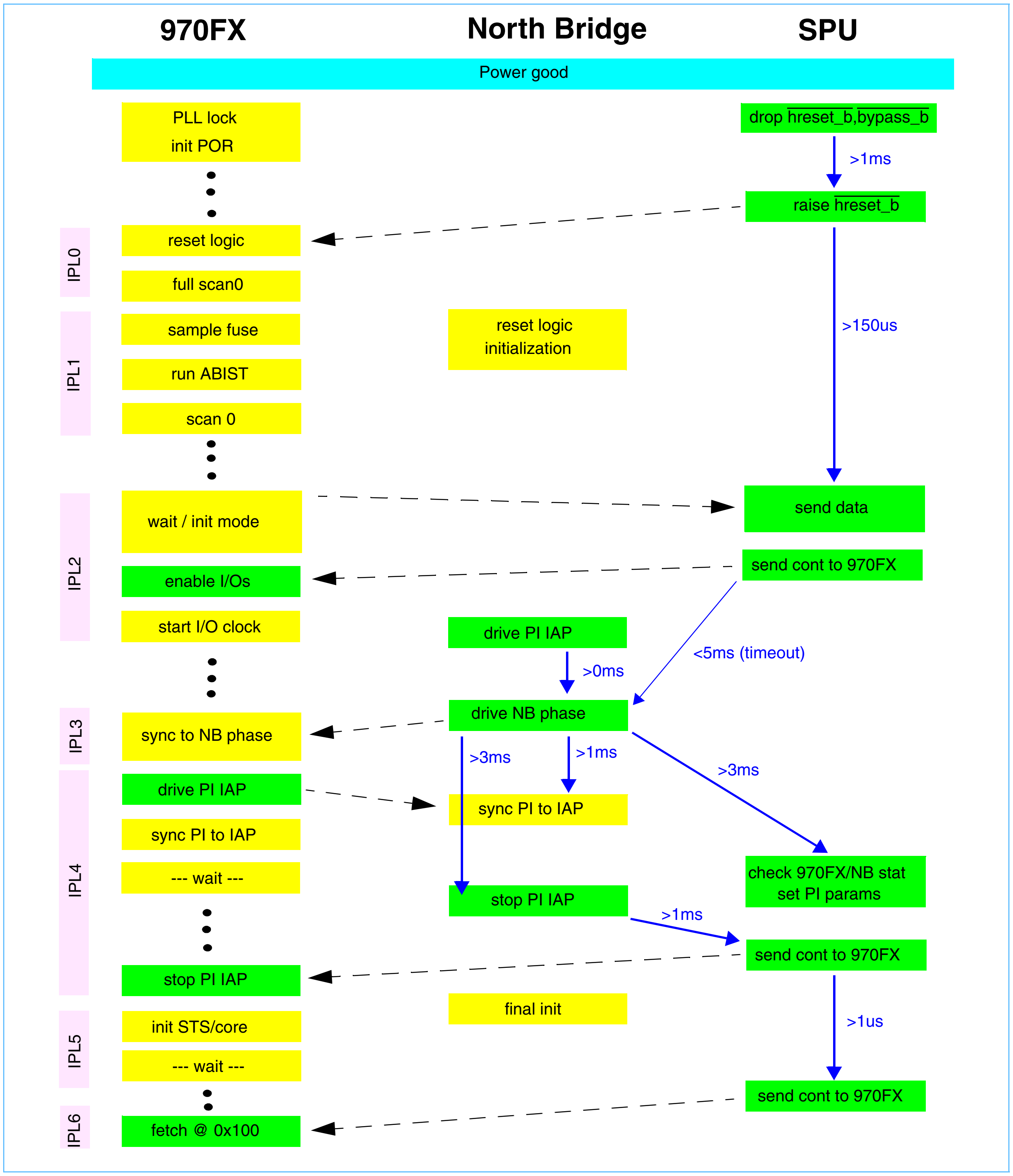


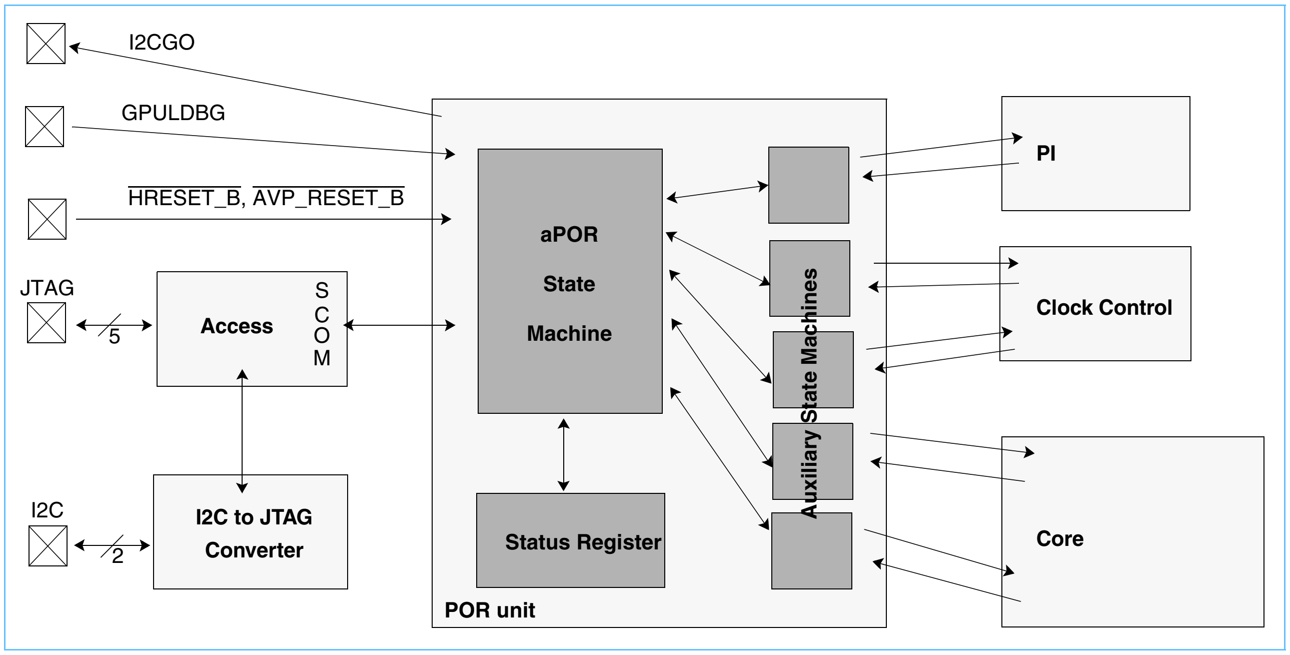
Table 11-1. Power-On Reset Procedure in Normal Mode

APC	IPL Step	970FX	North Bridge	SPU
		sync PLL		drop hreset_b, bypass_b
	IPL	reset POR, high-Z I2CGO	reset NB	sample I2CGO, raise bypass_b
				raise hreset_b
	-1	state PORWAIT		
00	IPL0	RSTFRL		
01		[FULL] SCAN0		
02	RASINIT	SAMPLEFUSE		
03	IPL1	SCABISTINIT		
04		SCAN0		
05		DABISTINITL1		
06		DABISTINITL2	init NB	wait 150us (970FX)
07	CHIPINIT	SCAN0		
08		DABISTINITL2		
09		SCAN0		
10		WAIT		
	IPL2		start WIAP, start phase	send mode data
	CHIPCFG			send continue
11		DRIVEIOS		
12	IPL3	SYNCPHASE		
13	CHIPSYNC	STARTZIOCLK		
14		TOGGLEWIAP		
15		SYNCRIAP		
16	IPL4	WAIT		wait for 970FX PI synced
	IOSYNC			start NB PI sync
			sync RIAP	
	only in PI-mode		stop WIAP	check EIs, set EIs params, stop NB WIAP., send cont.
17		TOGGLEWIAP		
18		STARTCORECLK		
19		INITSTS	final init	wait for 1us
20	IPL5	INITCORE		
21	COREINIT	WAIT		
				final system init. send cont.
22		STARTGUSCLK		
23	IPL6	INITSTS		
24	FETCHINIT	SRESET		
		<fetch @ 0x100>		

11.1.2.3 Interface Description

The interfaces of the Power-On Reset unit are shown in *Figure 11-2*.

Figure 11-2. Power-On Reset Interfaces



The POR unit can be accessed through both the *JTAG* and the *I²C*. The I²C macro used in the 970FX is only an I²C to JTAG protocol converter. Both JTAG Test Access Ports (TAPs) are muxed to a single TAP connected to the JTAG macro (Access) and therefore should not be used concurrently. An open drain pin I2CGO is used to arbitrate between both interfaces. The $\overline{\text{HRESET\_B}}$ input pin initializes the power-on reset, and the GPULDBG input pin is used to indicate to the 970FX operation in debug mode.

Table 11-2. External Power-On Reset Related I/O Pins

Pin Name	Polarity	Description
$\overline{\text{AVP\_RESET\_B}}$	low	Initiates power-on reset for functional pattern testing.
$\overline{\text{BYPASS\_B}}$	low	Used to bypass the PLL.
$\overline{\text{HRESET\_B}}$	low	(Power-on reset) Asserted after power supply and system clock have stabilized for 10ms to initialize power-on reset. After invalidation of this signal the power-on reset process will start. It requires some interaction with an external service element controlling the other chips of the motherboard.
GPULDBG	high	(Power-on reset debug mode) Used to bring the POR machine to a WAIT state directly after power-on reset process start, allowing to change to the POR sequence b.
I2CGO	low	(I2C Go Token pin) This pin is connected to a JTAG register and can be set by a JTAG command. It is used to arbitrate between I^2C and JTAG access to single step though the POR sequence.
SYSCLK SYSCLK_B	–	(Reference clock) This pin feeds the PLL input. The PLL deliver a x6/x8 clock which is distributed internally through the clock mesh: mclk
PSYNC	–	(Phase Sync In) This pin is driven by the North Bridge and identifies a time-zero mclk clock edge every 48.
PSYNC_OUT	–	(Phase Sync Out) This signal should match the Phase Sync In signal and is used to check phase alignment while debugging.
PROCID(0:2)	–	(Processor ID) Used to set the processor ID in a multi-processor environment. This is also used to obtain a unique I^2C address for the 970FX.
$\overline{\text{I2CDT\_B}}$	–	I^2C serial data
$\overline{\text{I2CCK\_B}}$	–	I^2C serial clock
TCK	–	JTAG test clock
TMS	–	JTAG test mode select
TDI	–	JTAG test data in
TDO	–	JTAG test data out
$\overline{\text{TRST\_B}}$	low	JTAG test reset

11.1.3 Power-On Reset Unit Description

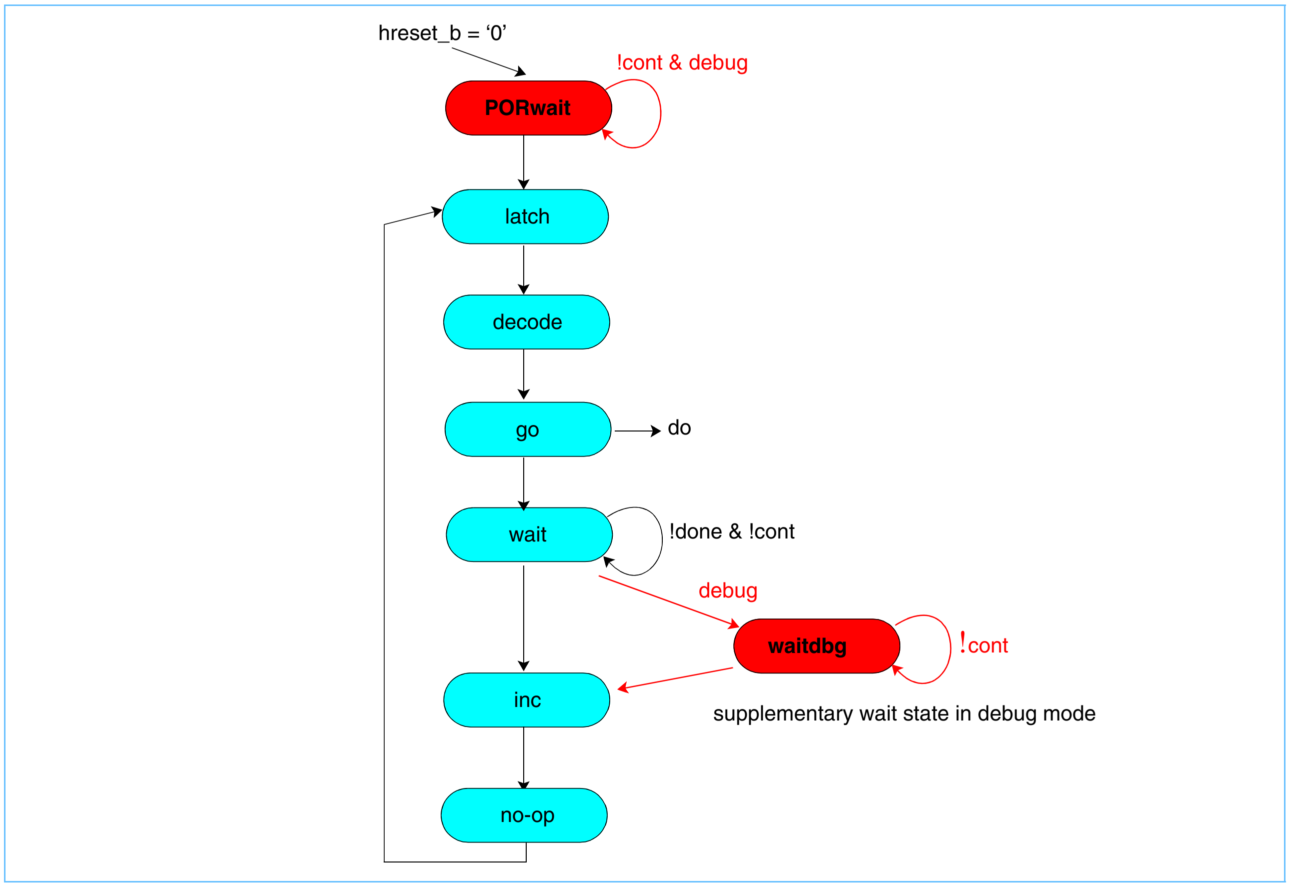
The POR unit can be divided into three main parts: The POR state machine that runs through the power-on reset sequence and decodes the power-on reset instructions; the auxiliary state machines that execute one particular instruction each and interface with the other chip units; and the status register that keeps track of the current processor and POR state.

11.1.3.1 Power-On Reset State Machine

The POR state machine consists of a POR Sequence Register containing 32 instructions, an instruction decoder, a program counter, and a control state machine that starts the execution of each instruction by auxiliary state machines and waits for them to complete.

After $\overline{\text{HRESET\_B}}$ is raised, the program counter (APC) is cleared and the POR state machine first enters the PORWAIT state. In debug mode, the POR state machine will not advance until a continue is sent from the SPU. In normal (non-debug) mode the state machine will advance to the next cycle without a continue. The state machine then starts fetching the instruction from the POR sequence register pointed to by APC (latch state), decodes it (decode state) and starts the execution of the corresponding auxiliary state machine (go state). The state machine then waits until the auxiliary machine signals completion or a continue command is received through the JTAG or I^2C. The POR program counter is then incremented and the state machine loops to the latch state.

Figure 11-3. State Diagram



11.1.3.2 Continue Command

To move the POR state machine from step to step, the service processor issues a “continue” command. The continue command is triggered by writing any 64 bit value (for example, all ‘0’s) to the Power-On Reset Continue Register (SCOM address 0x400101).

Note: The first continue command written to the 970FX after HRESET will not generate an I^2C acknowledge. The second and all subsequent continue commands will receive the normal I^2C acknowledge. This issue is included in the *IBM PowerPC 970FX RISC Microprocessor Errata List*.

11.1.3.3 POR Status Register

The POR Status Register is located at SCOM address 0x400000 and indicates the current POR program counter and whether or not the current instruction has completed.

11.1.3.4 Mode Ring

The mode ring is a bitstream that initializes critical mode latches in the processor cores. Most applications can use the recommended default settings, but some must be configured based on system design details. The Hypervisor Interrupt Offset Register (HIOR), which selects the physical base address of the reset vector and other interrupt vectors might need to be modified.

Most systems will use mode rings set to the default values provided in *Table 11-4*, however, some systems will need to customize the mode ring to use the correct HIOR value. This register provides the base address for the first fetch at the end of the POR sequence. The HIOR value, usually the base address of the 970FX boot ROM, plus the reset vector (0x100) should equal the physical address of the first instruction to be executed by the 970FX.

Note: The reset vector offset of 0x100 can not be changed.

Initializing the mode ring is handled by the following procedure, also described in more detail in *Section 11.1.5.4 IPL2*. There are 3 basic steps:

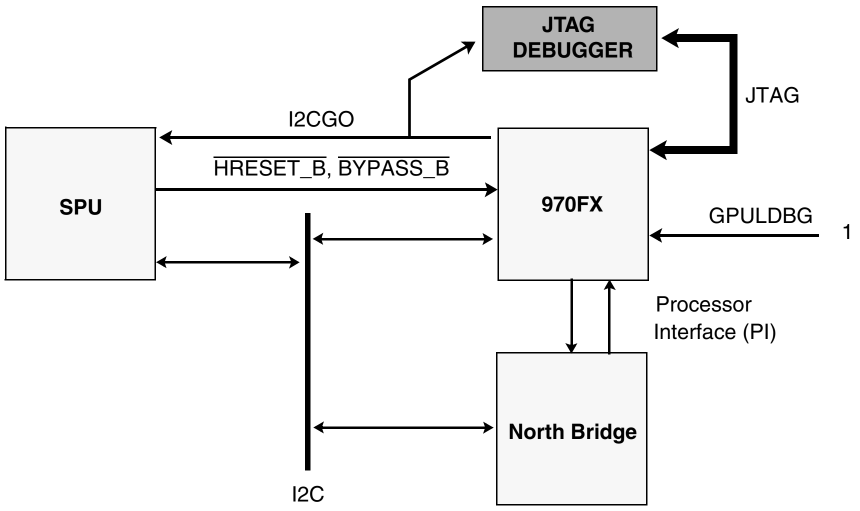
1. Initializing the Phase Synchronization Control Register (SCOM register 0x800006) with the value 0x0000.02F2.8000.0000.
2. A sequence of writes to the TAP controller. First the Instruction Register (IR), then the Data Register (DR) is loaded with the bits of the mode ring. Using I^2C, you can send up to 8 bytes at a time.
3. Then a TAP Reset is sent to write the mode ring from the TAP controller.

Note: See the *IBM PowerPC 970FX Power-On Reset Application Note* for additional details on initializing the Mode Ring.

11.1.4 Debugging Support

Figure 11-4 shows the system environment with the additional debugging environment for bring-up in red. The GPULDBG pin is driven high and the JTAG debugger (for example, the IBM RISCWatch) bring-up vehicle is connected to each 970FX via dedicated JTAG busses.

Figure 11-4. System Environment with Debugging Environment



In debug mode the POR state machine will always pause after the execution of each POR instruction as if a WAIT instruction has been inserted after each instruction. A continue command must then be issued to proceed to the next instruction. This also applies to the WAIT instruction such that a total of two continue commands must be issued to proceed in debug mode. This allows the SPU power-on reset procedure to remain unchanged and the JTAG debugger to access the 970FX after each POR step.

11.1.5 Power-On Reset Procedure in Debug Mode (Detail View)

This description of the 970FX power-on reset sequence will assume the “debug” mode (GPULDBG pin held high) to show each individual step in the POR state machine sequence. This mode would typically be used during system bring up, but once the power-on reset sequence has been correctly developed and debugged, the firmware would probably be modified to use the automatic mode.

The POR procedure in debug mode is listed in*Table 11-3 POR Procedure in Detail, Debug Mode*. The JTAG debugger should be able to single step the POR sequence in order to be able to check the 970FX state and make changes to the logic. This is necessary in the debugging environment, at least to correctly setup copies of the unblown fuses, run LBIST, or change the initial value of the latches/mode ring without changing the SPU code or iterating the 970FX hardware.

Table 11-3. POR Procedure in Detail, Debug Mode (GPULDBG pin pulled high)

APC	IPL Step	970FX	North Bridge	SPU
		sync PLL		drop hreset_b, bypass_b
	IPL	reset POR, High Z i2cgo	reset North Bidge	sample i2cgo, raise bypass_b
				raise hreset_b
	-1	state PORWAIT		
00	IPL0	RSTFRL		send continue
01		**[FULL] SCAN0**		send continue
02	RASINIT	SAMPLEFUSE		send continue
03	IPL1	SCABISTINIT		send continue
04		SCAN0		send continue
05		DABISTINITL1		send continue
06		DABISTINITL2	initialize North Bidge	wait 1us, send continue
07		SCAN0		send continue
08	CHIPINIT	DABISTINITL2		send continue
09		SCAN0		send continue
10	IPL2	WAIT		send continue
			start WIAP, start phase	send mode data
	CHIPCFG			send continue
11		DRIVEIOS		send continue
12	IPL3	SYNCPHASE		send continue
13		STARTZIOCLK		send continue
14	IPL4	TOGGLEWIAP		send continue
15		SYNCRIAP		send continue
16	IO SYNC	WAIT		Wait for 970FX PI synced
				start North Bridge PI sync
			sync RIAP	
	only in PI mode		stop WIAP	check PIs, set PIs parameters, stop North Bridge WIAP., send continue
17		TOGGLEWIAP		send continue
18	IPL5	STARTCORECLK	final init	send continue
19		INITSTS		send continue
20		INITCORE		send continue
21	COREINIT	WAIT		send continue
				final system init, send continue

Table 11-3. POR Procedure in Detail, Debug Mode (GPULDBG pin pulled high)

APC	IPL Step	970FX	North Bridge	SPU
22	IPL6	STARTGUSCLK		send continue
23		INITSTS		send continue
24	FETCHINIT	SRESET		send continue
		<fetch @ 0x100>		

11.1.5.1 Power and Clock Ramping for the 970FX

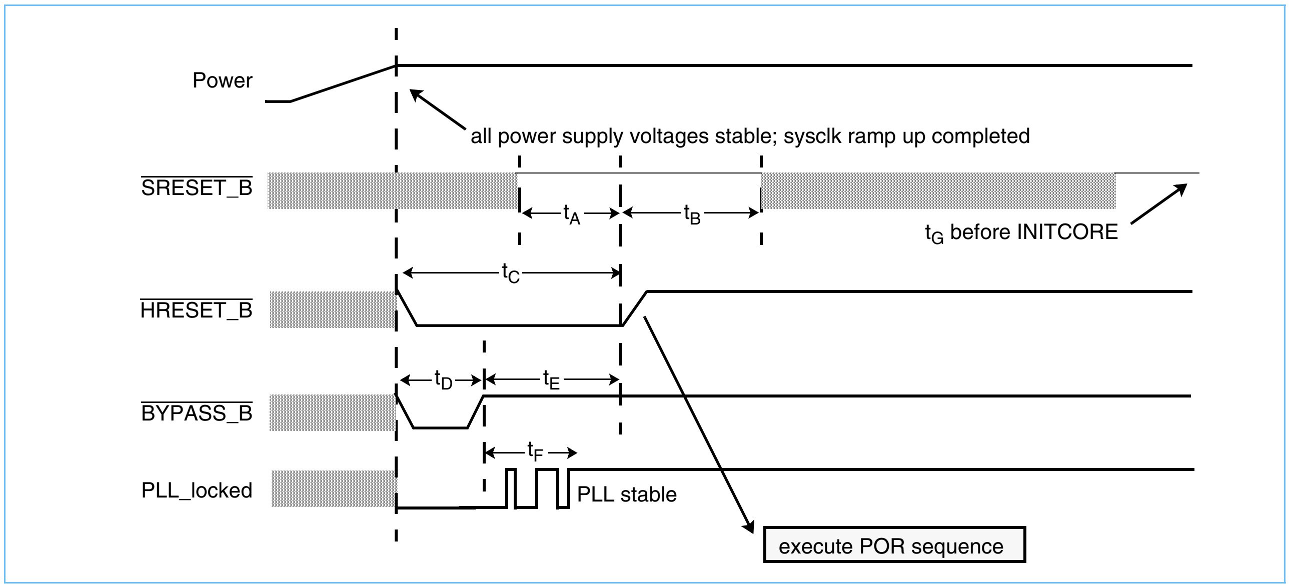
In the 970FX the $\overline{\text{SRESET\_B}}$ pin should be held high around the release of $\overline{\text{HRESET\_B}}$ as shown in *Figure 11-5*. Failing to conform to this specification will result in asynchronous clocks on the 970FX bus when doing frequency switching using power tuning.

When the PLL is in bypass-mode, where the $\overline{\text{BYPASS\_B}}$ pin is held low, the $\overline{\text{SRESET\_B}}$ should also be held low instead of high around the release of $\overline{\text{HRESET\_B}}$. This prevents the initial clock alignment procedure (CAP).

On bring-up boards, the polarity of the $\overline{\text{SRESET\_B}}$ around the release of $\overline{\text{HRESET\_B}}$ should be selectable to allow CAP skipping in the event of hardware problems.

Once the power supply is stable and the clock generator is running, the service processor should assert $\overline{\text{BYPASS\_B}}$ for t_D microseconds, while also asserting $\overline{\text{HRESET\_B}}$ for t_C milliseconds. This will reset the PLL and allow it to lock during the remaining time t_E required for $\overline{\text{HRESET\_B}}$ to complete. After $\overline{\text{HRESET\_B}}$ goes high the automatic power-on sequence begins with IPL0. (See the *IBM PowerPC 970FX RISC Microprocessor Datasheet* for timing values.)

Figure 11-5. $\overline{\text{HRESET\_B}}$, $\overline{\text{SRESET\_B}}$, $\overline{\text{BYPASS\_B}}$ Timing for the 970FX



11.1.5.2 IPL0

Release $\overline{\text{HRESET\_B}}$

Once $\overline{\text{HRESET\_B}}$ is released and the first contiue command is issued to the 970FX microprocessor, IPL0 starts. During IPL0 al the free running logic is initialized by scanning '0's into all of the RAS latches. This corresponds to Step 0 (RSTFRL) in *Table 11-3*.

In debug mode, issuing a continue command will move the instruction pointer to step 1, which will begin scanning '0's into all the rings. This corresponds to Step 1 ([FULL] SCAN0) in *Table 11-3*.

Note: The first continue command written to the 970FX after HRESET will not generate an I^2C acknowledge. Subsequent continue commands will get the normal I^2C acknowledge. This issue is included in the *IBM PowerPC 970FX RISC Microprocessor Errata List*.

Scanning of Boundary Scan Latches

One of the rings initalized by Step 1 is the boundary scan latches. During this step, some of the 970FX microprocessor output pins may be toggled as the boundary scan ring is initialized. System logic should ignore these output pin toggles. At this stage of the system intialization, most of the North Bridge logic is still being intialized, so this should not pose a problem.

It is important to note that one of the pins that might become active is $\overline{\text{QREQ\_B}}$. This pin is used to indicate a request to sleep. The SPU should avoid creating a situation where the potential toggling of $\overline{\text{QREQ\_B}}$ or other output pins might put the North Bridge logic into an undesired state. Since the scanning of the boundary scan latches depends on their uninitialized state at power up, this behavior is impossible to predict.

After another continue command is issued, the fuses are copied into their latches. This corresponds to step 2 in *Table 11-3*. Once the fuses are copied, the POR sequence automatically moves to IPL1.

Increasing I^2C Clock Speed

At this point in the POR sequence, you may (optionally) write to the Clock Ratio Register (SCOM 0x600400) with the value 0x0083F000.0000000 to increase I^2C speed to 100KHz. Without this SCOM initialization the I^2C bus speed will be limited to 50KHz.

Note: All I^2C accesses including this one, must use a 50KHz I^2C clock rate. The I^2C clock speed may only be increased after this SCOM write is complete. Using a higher I^2C clock rate may cause unreliable I^2C operation.

11.1.5.3 IPL1

Chip Initialization

IPL1 begins by initializing the arrays. This is done using the ABIST engine to intialize the Core, VPU, STS, and Pervasive Array. Finally the latches are initialized. As a result of the latch initialization, the fence between the core and the storage subsystem and between the storage subsystem and the processor interfaces are raised. The core is initialized.

From debug mode the SPU should issue 8 more continue commands to complete all the steps in IPL1. This should leave the POR program counter at instruction 10 (WAIT). During this phase the service processor can also begin intializing the North Bridge and getting it ready to start the IAP after the mode rings have been scanned in.

Verifying Chip Initialization in IPL1 is Complete

The SPU can check the status of IPL1 by reading the POR status register (SCOM 0x400000). When bits [24:28] of this register (the POR APC) contains the value 0x0A, the 970FX is ready to start IPL2.

11.1.5.4 IPL2

Initialization of the Phase Sync Control Register

Before the mode ring can be loaded you must first initialize the Phase Sync Control Register (SCOM register 0x800006) by writing the value 0x0000.02F2.8000.0000.

Note: This register must be initialized with this value before loading the mode ring, otherwise the mode ring load will fail.

Load Mode Ring

Now that the processor has been initialized in IPL1, configuration can begin. The POR state machine enters the WAIT instruction until the mode ring has been loaded. The mode ring contents are documented in *Table 11-4*.

Table 11-4. Sample Mode-Ring Content (for I^2C chip address of 0x40)

Content (Hexadecimal)
`08 40 52 DF 00`
`DE 40 52 00 80 C0 0F`
`02 40 52 03`
`BE 40 52 00 00 00 00 00 00 00 02`
`BE 40 52 00 00 00 00 00 02 00 00`
`BE 40 52 00 00 05 01 00 00 C3 01`
`BE 40 52 00 08 00 08 00 DE B0 21`
`BE 40 52 00 00 04 00 00 00 01 00`
`BE 40 52 00 00 00 00 00 00 00 00`
`BE 40 52 00` ***p0 p1 p2 p3*** `00 00 00`
`BE 40 52 00 00 00 00 00 00 00 00`
`BE 40 52 00 00 00 00 00 00 00 00`
`BE 40 52 00 00 00 C0 07 00 00 00`
`BE 40 52 00 00 00 00 00 00 00 00`
`BE 40 52 00 00 00 00 00 00 00 00`
`BE 40 52 00 00 00 00 00 00 00 00`
`BE 40 52 00 01 00 00 00 80 04 00`
`BE 40 52` ***h1 h2 h3 h4*** `00 00 00 00`
`BE 40 52 00 00 00 00 00 00 00 00`
`BE 40 52 00 00 00 00 00 00 00 00`
`BE 40 52 00 00 40 00 00 00 00 00`
`BE 40 52 00 00 c0 70 00 50 C8 10`
`BE 40 52 00 00 00 01 00 00 00 00`
`BE 40 52 14 00 00 00 00 00 00 a0`
`BE 40 52 7E 00 00 00 00 00 00 00`
`BE 40 52 00 00 18 00 00 00 00 00`
`BE 40 52 00 00 00 00 00 00 08 40`
`BE 40 52 00 10 00 00 00 20 00 7B`
`BE 40 52 03 00 00 00 00 00 00 00`
`BE 40 52 00 00 00 00 00 00 00 00`
`BE 40 52 00 00 00 00 1C FC 00 00`
`DE 40 52 00 00 00 00 00 3B 00 00`
`03 40 52 1F`

Notes:

1. Bytes Indicated by ***p0-p3*** should be replaced per *Table 11-5* according to Bus Ratio and PLL Settings.
2. Bytes indicated by ***h1-h4*** should be replaced per *Section Mode Ring Customization for HIOR* according to desired HIOR.

Table 11-5. Mode Ring Content Dependent on the Bus Ratio and PLL Settings

Bus Ratio	PLL x8	PLL x 12
2:1	0x98, 0xB0, 0x00, 0x00	Not Recommended
3:1	Not Recommended	0x90, 0x0F, 0x83, 0x00

Note: The combinations of Bus Ratio and PLL Mode shown as Not Recommended are likely to cause performance degradation caused by internal synchronization delays. For best performance use only the combinations shown.

Mode Ring Customization for HIOR

Certain fields in the mode ring may need to be customized for the application. The HIOR register is a 64-bit register that defines the base physical address for the interrupt vectors, including the reset vector. The processor will begin executing code at the physical address of the HIOR register plus the reset vector, 0x100. For example, if the HIOR is initialized to 0x0000.0000.FFF0.0000 in the mode ring, the first fetch will be made at the physical address 0x0000.0000.FFF0.0100.

The HIOR register is included in the mode ring shown in *Table 11-4 Sample Mode-Ring Content (for I^{2}C chip address of 0x40)*. The bytes to be replaced by the desired HIOR value are indicated by ***h1-h4***. In this example, ***h1*** = HIOR[43] * 128; ***h2*** = HIOR[35:42]; ***h3*** = HIOR[27:34]; ***h4*** = HIOR[22:26].

Mode Ring Customization for PLL Multiplier and Bus Ratio

The mode ring must also be modified to match the system bus ratio and PLL Mode. Use *Table 11-5* to determine the correct byte sequence to insert into the mode ring. The bytes in *Table 11-5* replace the ***p1-p5*** bytes in *Table 11-4*.

Note: These mode ring settings do not affect or override the PLL multiplier or bus ratio settings. They are used to support correct frequency scaling for the power tuning features. Incorrect configuration of these bytes will not affect full frequency operation, but will cause bus errors during the power tuning operation.

TAP Commands to Start Mode Ring Scans

The mode ring is written by a series of TAP commands. TAP commands provide a way to emulate JTAG functionality via a series of I^{2}C writes.

Writing the mode ring begins by setting the JTAG logic to SHIFT-IR mode. This is accomplished by writing the I^{2}C byte sequence 0x08, 0x40, 0x52, 0xDF, 0x00.

The next TAP command sets the ring address of the mode ring, for 970FX this is 0xC08000. This is accomplished by writing the I^{2}C bytes 0xDE, 0x40, 0x52, 0x00, 0x80, 0xC0, 0x0F. The next TAP command sets up the SHIFT-DR, this is handled by the I^{2}C byte sequence 0x02, 0x40, 0x52, 0x03.

Writing Mode Ring Data

The mode ring data is then written in a series of 64-bit bursts using the TAP controller via I^{2}C. Each 64-bit write begins with the preamble 0xBE, 0x40, 0x52, followed by 8 bytes of mode ring data.

The mode ring contents are listed in sequential byte order in *Table 11-4*. This table contains the bytes to be sent via I^{2}C.

Send the mode ring data to the core by writing the preamble bytes above, (0xFE, 0x40, 0x52) followed by 8 bytes from *Table 11-4*. Continue sending these 64-bit bursts of data until you have sent every byte in the mode ring. Then send the TAP Reset as described in *Return TAP Controller to Idle State*.

Return TAP Controller to Idle State

A TAP reset is sent to write the mode ring from the TAP controller by writing the I^{2}C sequence 0x03, 0x40, 0x52, 0x1F. Once this reset has been issued, the entire mode ring has been transferred into the 970FX.

PSYNC Requirement for 970FX DD3.X

Note: See the *IBM PowerPC 970FX Power-On Reset Application Note* for details on the resynchronization of the PSYNC input to the internal "Time 0" signal (also known as Clock Alignment Procedure) on some DD3.X hardware.

Start IAP and Phase

Once the mode ring has been loaded, the North Bridge should begin sending the WIAP pattern. At this point the North Bridge starts sending the synchronization pattern to the 970FX.

11.1.5.5 IPL3

Synchronization of all PowerPC 970FX RISC Microprocessors to PSYNC

In this phase the 970FX begins synchronizing to the external PSYNC signal driven by the system to indicate the correct "time zero" reference for bus operation and snooping.

Send Continues

At this point the service processor sends four "continue" commands to the 970FX to start IPL4. The continue command is triggered by writing any 64-bit value (for example, all '0's) to Power-on Reset Continue Register (SCOM address 0x400101). At this point the POR Status Register bits[24:28] should read a value of 0x10, indicating instruction 16.

11.1.5.6 IPL4

Wait for IAP to Complete

The 970FX starts driving the elastic interface alignment pattern (TOGGLEWIAP) and proceeds to the synchronization of the PI receivers (SYNCRIAP). The North Bridge can then start synchronization of its PI receivers.

Verify IAP Complete without Errors

The SPU finally checks correct synchronization of all chips in the system. The PI error condition registers can be checked at this point to determine if IAP was able to complete without error.

Initialize PI Parameters

Once the IAP has successfully completed, the service processor will initialize the PI bus for correct operation by configuring the STATLAT, SNOOPLAT, SNOOPACC, and APSEL parameters. The target cycles should also be configured for the North Bridge.

Stop IAP Pattern

With the processor interface bus initalized and ready for operation, another SCOM write to the Processor Interface Mode Register (SCOM address 0x046A00) will halt the IAP pattern and quiesce the bus. This should also be done for the North Bridge.

Send Continue

At this point the service processor sends 3 “continue” command to the 970FX to start IPL5.

11.1.5.7 IPL5

Start Core Clock

The Core Init step (IPL5) finishes the Core initialization. The core clock is started, the array fences are dropped (GUSINIT without STS clocks), the Core CAM arrays are set up, and the Core quiesced. The SPU then finishes the system initialization and sends 7 more continue commands to the 970FX microprocessor.

Some system implementations will also need to make the boot ROM available to the processor's memory map. This can be done by initializing a South Bridge, or may require copying the contents of a service processor ROM into the system DRAM.

Once this final initialization is complete, 3 more continues will take the 970FX to IPL6.

11.1.5.8 IPL6

Start STS Clock

The last step Fetch Init (IPL6) starts the STS clock, resets the storage interface of the 970FX, and starts fetching instructions at address HIOR + 0x100. 0x100. The first continue command will start the GUSCLK. This continue must be completed before setting the PI parameters.

Initialize PI Parameters

Once the IAP has been completed successfully, the service processor must initialize the PI bus for correct operation by configuring the STATLAT, SNOOPLAT, SNOOPACC, and APSEL parameters. These parameters are documented in*Section 11.3 Bus Initialization, Configuration, Power Management, and Test*.

STS Init and SRESET

Once the PI parameters have been set, one more continue command will initiate the SRESET and cause the processor to begin fetching instructions.

11.1.6 Processor Initialization

The processor will begin executing code at the physical address of the HIOR Register plus the reset vector, 0x100. For example, if the HIOR is initialized to 0x0000.0000.FFF0.0000 in the mode ring, the first fetch will be made at the physical address 0x0000.0000.FFF0.0100.

The processor starts execution with memory translation off (real address mode, effective address = physical address), and with the caches disabled. Some system configurations will also need other prefetching or superscalar features disabled to allow correct operation.

11.1.6.1 Automatic Array Recovery

The 970FX microprocessor has built in recovery mechanisms to protect array reliability. An array soft error would normally cause a checkstop condition that requires service processor intervention. Enabling the recovery modes in *Table 11-6* and *Table 11-7* allows the 970FX microprocessor to recover from array soft errors automatically.

Note: The checkstop enables for the L2 Uncorrectable Error (UE) and Logic UE should be set to ‘1’ (Enabled). In the rare event either of these errors occur, no automatic recovery is possible and the 970FX microprocessor should be set to checkstop.

Table 11-6. Enabling Automatic Array Recovery Modes, by Array

Array/Error Source	HID Register Setting	Error Mask Register (SCOM 0x030400)	Machine Check Register (SCOM 0x030901)	Checkstop Register (SCOM 0x030800)
I-Cache / ITAG	HID1[11:12] = ‘11’	0x030400[0:2] = ‘000’	0x030901[0:2] = ‘000’	0x030800[0:2] = ‘000’
I-ERAT	HID1[13] = ‘1’	0x030400[3] = ‘0’	0x030901[3] = ‘0’	0x030800[3] = ‘0’
L2 UE	N/A	0x030400[4] = ‘0’	0x030901[4] = ‘0’	0x030800[4] = ‘1’
Logic UE	N/A	0x030400[5] = ‘0’	0x030901[5] = ‘0’	0x030800[5] = ‘1’
D-Cache	HID5[50] = ‘0’ HID4[39:41] = ‘000’	0x030400[6] = ‘0’	0x030901[6] = ‘0’	0x030800[6] = ‘0’
DTAG	HID4[34:36] = ‘000’	0x030400[7] = ‘0’	0x030901[7] = ‘0’	0x030800[7] = ‘0’
D-ERAT	HID4[29:31] = ‘000’	0x030400[8] = ‘0’	0x030901[8] = ‘0’	0x030800[8] = ‘0’
LSU-TLB	HID4[44:48] = ‘00000’	0x030400[9] = ‘0’	0x030901[9] = ‘0’	0x030800[9] = ‘0’
LSU-SLB	HID4[53:54] = ‘00’	0x030400[10] = ‘0’	0x030901[10] = ‘0’	0x030800[10] = ‘0’

Table 11-7. L2 Array Recovery Details

L2 Error	STS Mode Register (SCOM 0x043000)	Error Mask Register (SCOM 0x040401)	Checkstop Register (SCOM 0x040801)	Notes
L2 CE	0x43000[50] = ‘0’	0x040401[42] = ‘0’	0x040801[42] = ‘0’	
L2 UE	0x43000[50] = ‘0’	0x040401[43] = ‘0’	0x040801[43] = ‘1’	Uncorrectable Error
L2 Special UE		0x040401[44] = ‘0’	0x040801[44] = ‘1’	Error outside L2, but detected in L2
L2Dir		0x040401[45] = ‘0’	0x040801[45] = ‘0’	
L2Dir Checkstop		0x040401[46] = ‘0’	0x040801[46] = ‘1’	
L2 Hang Detect		0x040401[47] = ‘1’	0x040801[47] = ‘X’	
L2 STQ		0x040401[48] = ‘0’	0x040801[48] = ‘1’	Store queue error
Logic UE		0x040401[49:51] = ‘0’	0x040801[49:51] = ‘1’	
L2$ Quad CE Threshold		0x040401[52:56] = ‘00000’	0x040801[52:56] = ‘00000’	
L2Dir multiple errors within 2 hang pulses		0x040401[57] = ‘0’	0x040801[57] = ‘0’	

11.1.7 Debugging Tips

11.1.7.1 Verifying I^{2}C Operation

Some microcontrollers used as SPUs for 970FX will require level shifters to avoid levels on the 970FX pins beyond OV_{DD}.

Note: I^{2}C pins should never exceed voltages beyond the OV_{DD} tolerances provided in the *IBM PowerPC 970FX RISC Microprocessor Datasheet*. Devices might be permanently damaged if inputs exceed allowable maximum voltages.

Most problems with I^{2}C communications are caused by incorrect pullup resistor values, or I^{2}C clock frequency. Note that 970FX I^{2}C clock frequency should not exceed 50KHz unless the SPU code implements the SCOM initialization described in *I^{2}C Bus Speed on page 260* and*Increasing I^{2}C Clock Speed on page 270*.

11.1.7.2 SCOM Access to Uninitilized Units

If an SCOM access is attempted to units before clocking is enabled, the SCOM engine will fail and the SCOM state machine will hang. Before attempting any SCOM access, ensure completion of the required IPL step to enable clocks to that unit (for example, STS SCOMs cannot be accessed until POR step 22, which actually enables the STS clocks).

Notes: If the SCOM hardware is disabled by attempting access to a unit that is not yet clocked, the only recovery is to start over from HRESET.

11.1.7.3 Use of SRESET

A POR sequence that causes an error is likely to require a complete restart from HRESET and BYPASS. Unlike earlier PowerPC processors, the 970FX usually cannot be restarted via SRESET. The $\overline{\text{SRESET\_B}}$ signal can only be used in limited circumstances, where the core can be quiesced before it will be recognized. Errors in POR initialization may make quiescing the core impossible. For this reason, it is usually easier to just begin the entire sequence over rather to attempt to quiesce the core and restart from the fetch at 0x100.

However, once a system is running it should be able to restart via assertion of the $\overline{\text{SRESET\_B}}$. For example, recovery should be possible from most software crashes or kernel panics by asserting $\overline{\text{SRESET\_B}}$, provided the boot code is still intact in memory.

11.1.7.4 Diagnosing IAP Errors

Errors and warnings during IAP are recorded in the Processor Interface Status Register (*Section 11.6.8.18* on page 318). Note that bit [36] does not indicate that IAP was successful, only that the IAP state machine ran to completion. The IAP state machine will not complete if any fatal error occurs (bits [39, 42, or 45]). See the *IBM PowerPC 970FX Power-On Reset Application Note* for interpretation of the other bits.

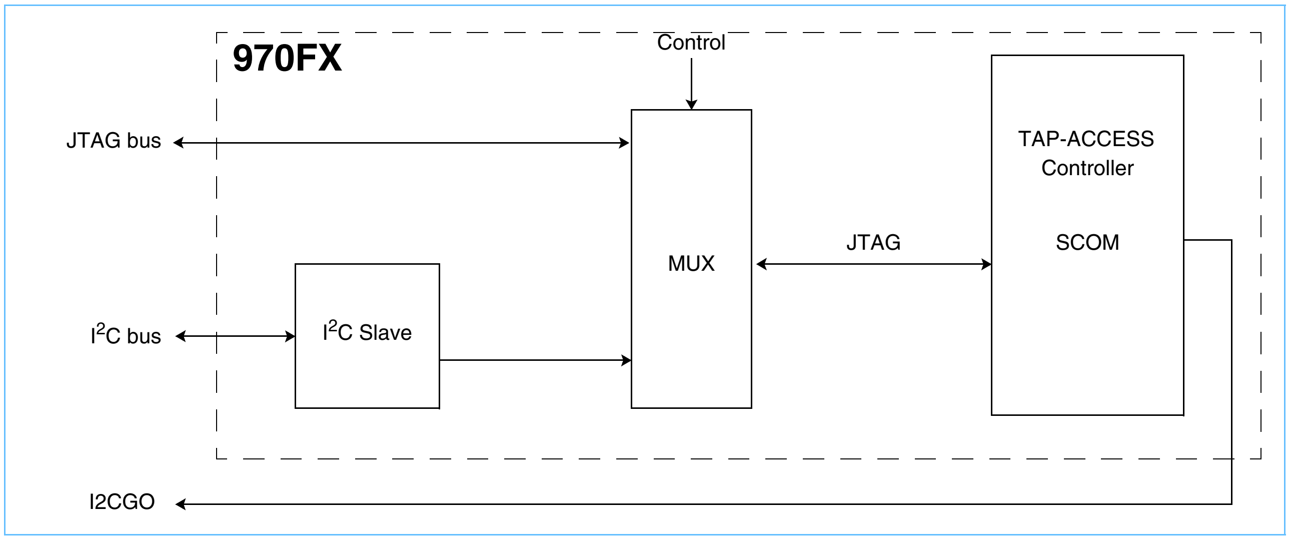
11.2 I²C Interface

11.2.1 I²C Purpose

I²C (Interconnect for Integrated Circuits) is a standard bus developed by Philips Electronics.[1] The I²C Slave described in this manual converts data sent across an I²C bus into native JTAG commands. The I²C slave can be used as a test access port (TAP) controller that interfaces with the Access macro or with other IEEE 1149.1 compatible devices in order to read, write, and scan registers within a chip.

Figure 11-6 shows the basic concept of merging the off-chip I²C and JTAG busses.

Figure 11-6. Merged JTAG and I²C Interfaces



The I²C slave has two basic functions:

- SCOM reads and writes: The I²C slave can translate data sent across the I²C bus in order to read or write registers using native JTAG commands and then return the on the I²C bus if requested. The I²C slave has a 12-byte buffer that holds data sent from the master to the slave, as well as data retrieved from registers during a read sequence. All data is 8-byte aligned due to the implementation of the TAP controller.
- Native JTAG function: The I²C slave can also be used to interpret I²C data and send native JTAG commands to a TAP controller. The I²C slave monitors the Attention signal sent from the TAP and optionally uses this as part of the decision to acknowledge data or as the slave address depending upon the read/write bit of the I²C address. This can be enabled or disabled for testability and lab functions to make communication easier.

Through the use of primitive decodes the I²C slave returns the data in its 12-byte register without performing any JTAG activity or compromising the previous data from capturing test-data output (TDO).

1. I²C standard (IIC) for a serial bus. For more info see: http://www-us2.semiconductors.philips.com/i2c/.

11.2.1.1 I2CGO Pin

The I2CGO open collector pin is used to prevent access collisions between the JTAG and I^{2}C. If the level of the pin is low, only JTAG should access the 970FX. The I^{2}C can make use of the interface if the level is high.

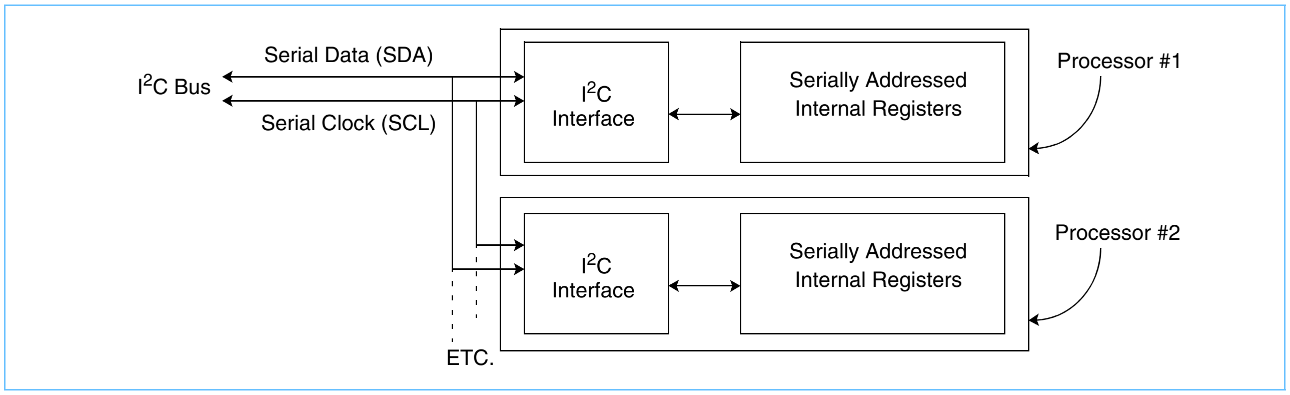
The I2CGO can be set by either the JTAG or I^{2}C through SCOM write commands. When set by an SCOM write, the pin will hold its old value until the TAP controller enters either the RESET or the IDLE state.

11.2.1.2 I^{2}C Protocol

The I^{2}C protocol, as defined, supports only register read and write operations. This manual only deals with the 7-bit slave addressing mode, allowing 2^7 slaves to be directly addressed. The protocol, illustrated in *Figure 11-7*, implies that an addressed slave responds to a read or a write by addressing successive bytes serially, starting in memory where it is defined by the particular slave.

Implementation note – The I^{2}C-to-JTAG slave implementation extends direct-addressing to 2^{23}, operating on 8-byte aligned registers. In addition, operations other than register reads and writes that are supported by the access implementation of JTAG are available.

Figure 11-7. I^{2}C Protocol



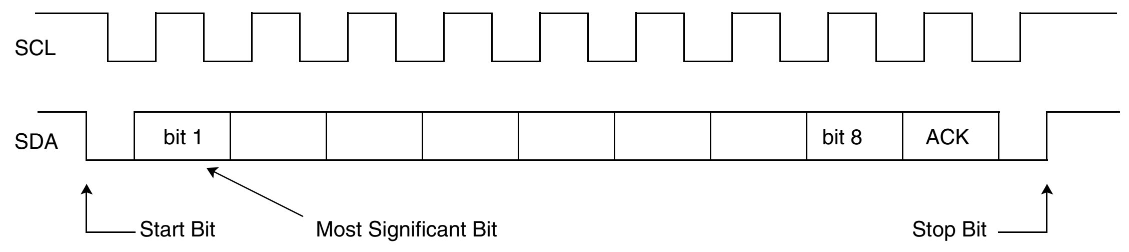
11.2.1.3 I^{2}C Bus Operation

As shown in *Figure 11-8*, the I^{2}C bus consists of two wires: SDA (serial data) and SCL (serial clock). These rest at '1' while the bus is quiescent. Any bus master can initiate a message transfer with a start bit (SDA transitions from '1' to '0' while SCL equals '1'). The I^{2}C operates on packets, each one-byte wide, each is followed by an acknowledgment bit (0 indicates a good ACK). When the message begins, the master owns driving both SCL and SDA until the ACK bit is received; then the slave owns driving the SDA (on writes only, see the *I^{2}C Bus Specification* from Phillips Semiconductors for further details). Each SDA data/ACK bit must remain stable at '1' or '0' while the SCL clocks transition from '0' to '1' to '0'. That is, the SDA is set up prior to the SCL rising edge, held after the falling edge, and transitions while SCL is low.

The slave can temporarily pace the operation by holding the SCL low following the ACK bit. The SCL clock rate slows to the speed of the slowest master/slave attached.

Packet bits are transmitted with the most significant bit (MSb, bit [1]) first. Bits [1:7] of the start byte address a particular slave chip. Bit [8] of the start byte is a read/write bit. The least significant byte is always transmitted first, followed by successively more significant bytes (this applies whether the byte is interpreted as address or data information). Writes progress with the master sending bytes and the slave acknowledging. A read begins as the master writes the start byte, but the dataflow reverses after the slave acknowledges the start byte. Now the slave is sending packets and the master is acknowledging. The slave continues to send until the master ACK equals '1'. All transmissions end with a stop bit (SCL equals '1' while SDA transitions from '0' to '1'). The bus is quiescent again following the stop bit.

Figure 11-8. I^2C Bus Operation



11.2.1.4 Deviations from the I^2C Standard

These are deviations, not limitations, of the current design.

- Does not support 10-bit addressing mode.
- Does not support general call address.
- Does not support auto-increment of slave byte address. The power-on reset (POR) default byte address is x'000000' (an SCOM address, as implemented by Access).

11.2.1.5 JTAG Overview

The Boundary Scan/JTAG formerly known as IEEE standard 1149.1, is a set of design rules that facilitate testing, device programming,and debug at the processor, board, and system levels. IEEE 1149.1 defines a 5-wire interface called a Test Access Port (TAP) for communicating with boundary scan architecture. The wires in the interface are:

Test Clock (TCK)	The rising edge causes TMS and TDI to be sampled by Access macro.
Test Mode Select (TMS)	The value of TMS during the rising edge of TCK causes a state transition in the TAP controller.
Test Data In (TDI)	TDI is the serial data input to Access.
Test Data Out (TDO)	TDO is the serial data input to Access.
Test Logic Reset (TRST)	TRST causes an asynchronous reset of the test logic (TAP transitions to TestLogicRst).

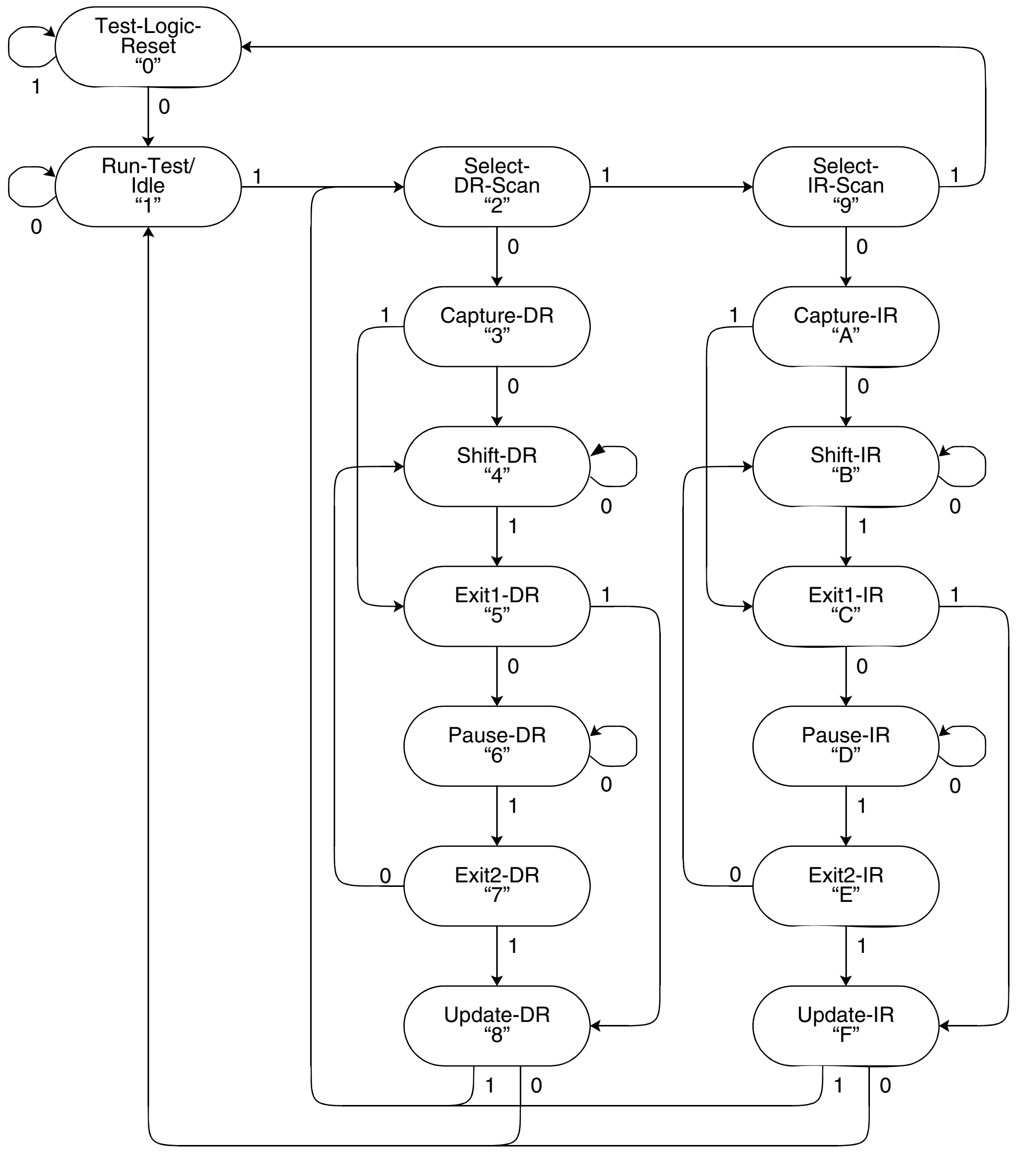
11.2.1.6 TAP Controller

Figure 11-9 on page 282 illustrates the operation of the TAP controller and the Access macro, which is a specific implementation of the IEEE specification. TMS and TCK control the TAP controller. It is necessary to understand ShiftIR and ShiftDR states for this discussion.

- ShiftIR: The instruction register (IR) inside the Access macro is serially connected between TDI and TDO. While TMS is held '0', each test clock (TCK) causes another bit to be shifted into the Instruction Register from TDI and the IR Status to be shifted out to TDO.
- ShiftDR: One of many test data registers (TDRs) is serially connected between TDI and TDO. While TMS is held '0', each TCK causes another bit to be shifted into the register from TDI and (old) data to be shifted out TDO.

The contents of the Instruction Register determine the specific TDR addressed or nonregister operation to be performed.

Figure 11-9. IEEE/Access TAP Controller

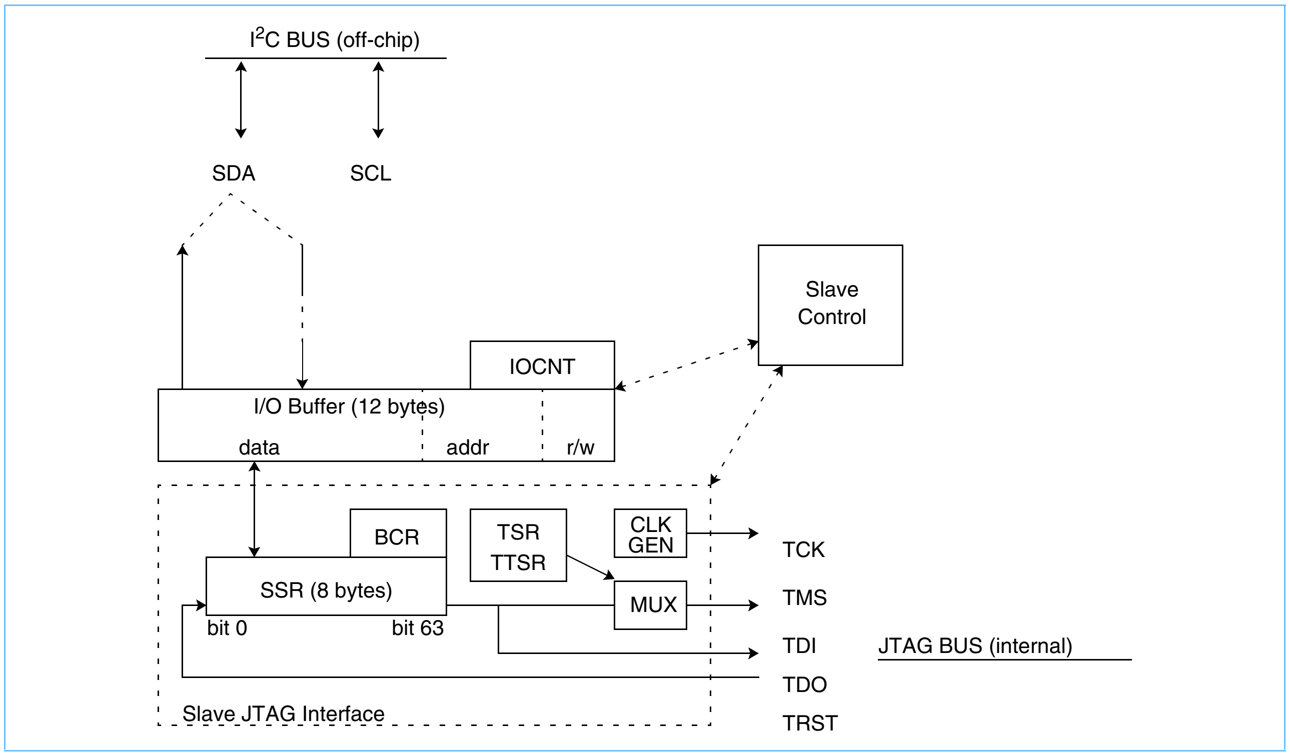


Note: The numbers in quotation marks are the state numbers used in the state machine implementation. The numbers on the arrows represent the state of the TMS line leading to the transition.

11.2.2 Slave Implementation

Figure 11-10 outlines the major registers involved in data and control flow.

Figure 11-10. Major Registers Involved in Data and Control Flow



11.2.2.1 Slave Data Flow Overview

The physical interface handles I^{2}C protocol and timing. It provides a one-byte wide data interface along with associated handshaking signals. SDA is the bi-directional serial data signal defined by the I^{2}C. SCL is the serial clock signal defined by the I^{2}C.

The I/O Buffer (IOBUF) buffers successive bytes to be transferred to or from the chip addressed as the slave chip by the I^{2}C start byte. Data in the IOBUF is transferred in parallel one byte at a time to or from the physical interface. The entire start byte is held in the IOBUF, but only the read/write bit is further used by the slave logic. During write operations, dataflow is from the I^{2}C to the TDI. During read operations the dataflow is from TDO to I^{2}C.

The Serial Shift Register (SSR) can receive or supply up to 8 bytes in parallel from or to the IOBUF. This data is then serially shifted to or from the JTAG following IEEE 1149.1 protocol. Test Data In (TDI) is the serial data in to JTAG. Test Data Out (TDO) is the serial data out of JTAG.

11.2.3 Programming

11.2.3.1 Considerations for Concurrent Resource Use by I^{2}C and JTAG

When the on-chip scan resource (Access) will be shared by the I^{2}C and the chip-level JTAG I/O, special considerations are required to prevent either from interfering with the other:

- There is only one copy of the scan resource, the semaphore pin I2CGO is needed to quiesce the off-chip JTAG or I^{2}C slave while the other is active.
- Neither I^{2}C nor JTAG have the concept of "atomic" streams of commands, some operations might fail if the command stream is interlaced.
- The JTAGINPROG output from the I^{2}C slave logic which is used to drive the on-chip JTAG MUX gives priority to the I^{2}C and can lock-out JTAG in certain circumstances if the "Enable Attention" TAP command is not used.

11.2.3.2 SCOM Register Read/Write

Table 11-8 describes the command format that will perform a 64-bit SCOM read/write.

Table 11-8. SCOM Register Read/Write (bit numbering, LSb = '0')

Start Byte		SCOM Address			SCOM Data (I/O buffer)							
I^{2}C Slave Addr ess (7 bits)	$R/\overline{W}$	LSByte	Mid	MSByte	8 bytes (LSByte,MSbit first)							
7..4 970FX base address 3..1 Proc ID	'0'	7..0 bit 0=odd Parity	15..8	23..16	7..0							63..56
	'1'	▨	▨	▨								

An SCOM operation begins with the start byte address (I^{2}C slave base address), optionally followed by up to 3 bytes of the SCOM address (LSByte, high-order bit first).[1] Note that the SCOM addresses must be byte aligned.[2] If a stop bit is received at this point ,only the portion of the address written will be altered; no other activity is initiated.

Write: Data for a write operation follows, starting with the fourth byte (LSByte, high-order bit first). A stop or overflow condition following one or more data bytes kicks off the SCOM operation.[3] The SCOM address is not auto-incremented. Writing more than 8 bytes of data will result in the data at the same SCOM address being overwritten.[4] Writing less than 8 bytes of data will result in the SCOM address being written with a full 8 bytes, the most significant bytes of data will be written with whatever was left in the SCOM data I/O buffer from the previous operation.

1. A write must start with all 3 bytes of the SCOM address; a read may use a previously written address or modify one, two, or all three bytes of the address before the restart/read byte.
2. The I^{2}C slave implements a 24-bit SCOM address (23 down to 0, 0=LSb). Access/JTAG defines the LSb as a parity bit. The remaining 23 bits (23 down to 1) represent a byte-aligned address. The SCOM transfers are always 8-bytes wide, but each chip usually architects registers of varying widths that might not necessarily be aligned on 8-byte boundaries. The specific alignment of data within the I2C SCOM read/write data field is chip implementation dependent.
3. An overflow or underflow occurs when more than 8 bytes of data are written/read.
4. SCOM data (IOBUF) contents are unpredictable prior to the first read/write operation. If the same address is written successively, the data remains in the buffer–modified only by the data byte(s) rewritten.

Read: For an SCOM read operation, a restart (a stop followed by a start) is then issued on the I^2C interface with the R/W bit set to ‘1’. This triggers the initialization of an SCOM read within the I^2C hardware. The dataflow then reverses and the slave returns one data byte for each acknowledgment (LSByte first). The SCOM address is not auto-incremented. If greater than 8 bytes are read, the data will be repeated starting with the least signficant byte.

Note: See the *IBM PowerPC 970FX Power-On Reset Application Note* for additional information on I^2C programming examples.

11.2.3.3 Power-on Reset and Processor Interconnect SCOM Registers

Table 11-9 lists the registers available as interface to the power-on reset (POR) unit along with their access mode and functionality.

Table 11-9. POR Unit SCOM Registers

Name	SCOM Address	Mode	Functionality	Page
STATUS	0x400000	R/W	Status register of POR unit (write access clears register)	322
CONT	0x400101	Write	Continue command to POR fsm	323
I2CGO	0x400201	Write	Assert I2CGO pin	323
PSEQ0	0x401400	Write	aPOR sequence register. 12 first entries (60 LSB)	325
PSEQ1	0x402400	Write	aPOR sequence register. 12 next entries (60 LSB)	325
PSEQ2	0x404400	Write	aPOR sequence register 8 last entries (40 LSB)	325
RSV	0x40xxxx	R/W	Reserved	–

Table 11-10 lists the registers containing the status and configuration of the Processor Interconnect Bus and of the BUS unit (PI protocol layer). The processor interconnect (PI) registers are logically part of the pervasive unit.

Table 11-10. PI and STS SCOM Registers

Name	SCOM Address	Mode	Functionality	Page
PISTAT	x046B01	Read	PI inbound status register	318
PICONF	x046A01	Write	PI inbound configuration register (delay)	317
BUSCONF	x0A8000	Write	STS bus unit configuration register	321

11.3 Bus Initialization, Configuration, Power Management, and Test

11.3.1 Bus Initialization

The bus devices use a physical layer initialization sequence to initialize the bus. A specific pattern is sent across the bus, which initializes the processor interconnect in slave devices. This sequence is described in *Section 11.4.1 Initialization at Power-On Reset*.

11.3.2 Configurable Parameters

The Processor Interconnect (PI) defines multiple configurable parameters for efficient operation of the bus. The values that can be programmed into these parameter registers are technology and implementation-dependent. During the initialization process at system start-up, the service processor unit (SPU) identifies the system configuration and programs the individual devices attached to the bus (that is, the North Bridge and the processors) with the appropriate values using the I^{2}C device interfaces. All values are in bus beats. For parameters that cross the processor interconnect, the values are from the final locally clocked flip-flop or latch output to the first locally clocked flip-flop or latch input, after deskewing has taken place through the processor interconnect.

Figure 11-11 shows the configurable timing parameters COMPACE and STATLAT. COMPACE is the minimum number of bus beats between command packets issued from the processor. STATLAT is the number of bus beats between the last beat of the address/data (AD) packet and the first beat of the transfer-handshake (TH) packet.

Figure 11-11. Configurable Timing Parameters

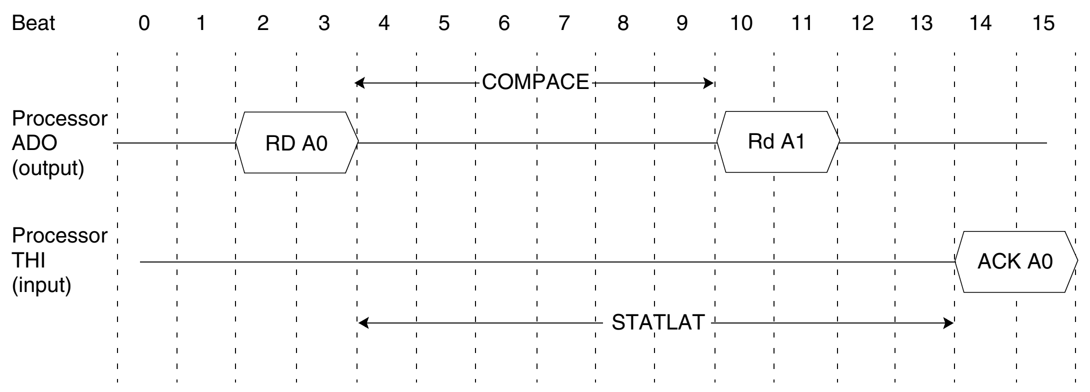


Figure 11-12 shows the North Bridge configurable timing parameters SNOOPWIN, SNOOPLAT, and PAAMWIN. SNOOPWIN is the minimum number of idle bus beats between reflected command packets. PAAMWIN is the minimum number of bus beats between command packets reflected from the North Bridge to the processors when there is an address collision (shown as A0 in *Figure 11-12 North Bridge Configurable Timing Parameters*). SNOOPLAT is the number of bus beats between the last beat of a reflected command packet to the first beat of the individual snoop responses from each of the processors received at the North Bridge. SNOOPLAT includes the time of flight across the interface and any switch devices interposed between the North Bridge and a processor.

Figure 11-12. North Bridge Configurable Timing Parameters

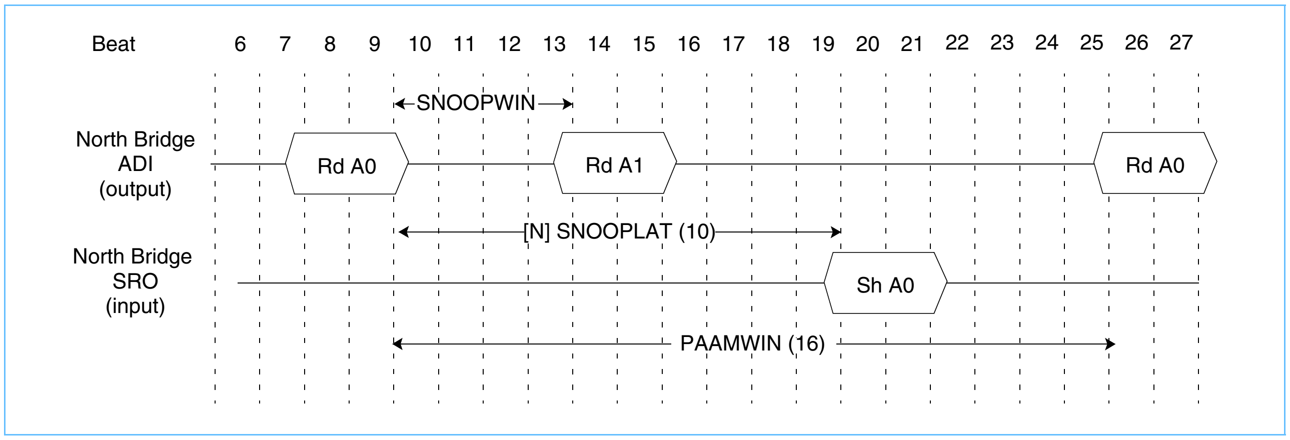
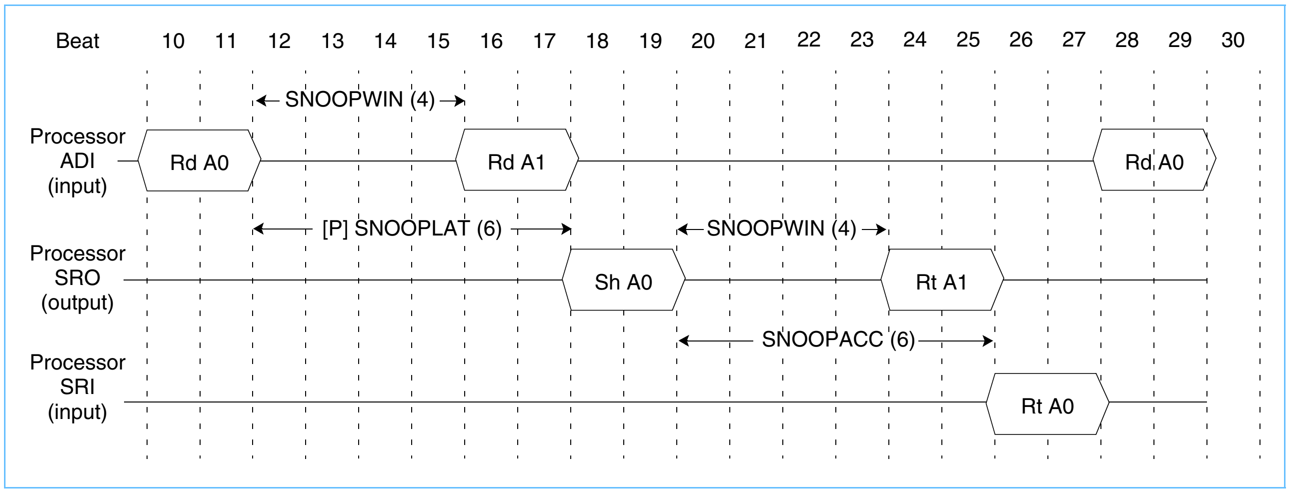


Figure 11-13 shows the processor configurable timing parameters SNOOPLAT and SNOOPACC. SNOOPLAT is defined above. SNOOPACC is the number of bus beats between the last beat of the individual snoop response sent from a processor to the first beat of the accumulated snoop response received from the North Bridge. SNOOPACC includes the time of flight across the interface and any switch devices interposed between a processor and North Bridge.

Figure 11-13. Processor Configurable Timing Parameters



11.3.3 Configuration Interface (I^2C Interface)

An I^2C interface is used to program the processor interconnect bus parameters. The interface complies with the *I^2C Bus Specification*.

The I^2C Interface consists of two bi-directional signals, $\overline{\text{I2CCK\_B}}$ and $\overline{\text{I2CDT\_B}}$. Both signals use open-drain drivers that require external pull-up resistors. Multiple devices can be connected to the same signals. The PROCID[0:1] inputs are used to address a specific processor.

Table 11-11. I^2C Interface Signals

Signal	Polarity	Name
$\overline{\text{I2CCK\_B}}$	Active Low	I^2C interface clock input
$\overline{\text{I2CDT\_B}}$	Active Low	I^2C interface data input/output
PROCID[0:1]	Active High	Processor identification input

Table 11-12. I^2C Registers used by the 970FX Processor Interconnect Bus

Name	Mode	Address	Description
PI STATUS REG	Read	x046B01	PI inbound status register. See *Section 11.6.8.18 Processor Interconnect Status Register* on page 318.
PI MODE REG	Read/Write	x046A00	PI inbound configuration register (delay). See *Section 11.6.8.17 Processor Interconnect Mode Register* on page 317.
BUSCONF	Read/Write	x0A8000	Configurable timing delay parameters. See *Section 11.6.8.22 Bus Configuration Register* on page 321.

11.3.4 Reliability, Availability, and Serviceability (RAS) Requirement

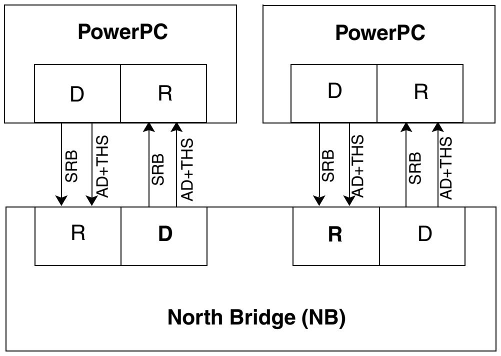
All devices attached to the Processor Interconnect (PI) bus must implement three registers that capture errors and assist in the isolation of failures. RAS circuitry ensures a PI implementation which targets a particular processor will remain stable across various target processors.

A Fault Isolation register (FIR), a Fault Isolation Capture register (FICR), and a Fault Isolation Mask register (FIMR) must be implemented. The FIR captures all failures that occur. The register is not frozen on the first error, but continues to accumulate all detected errors. The FICR is used to log the first detected error. This register is a masked copy of the FIR register (output of the FIR is masked with the FIMR) and is frozen once the first error is detected.

11.4 Processor Interconnect Electrical Interface

The processor interconnect uses high-speed source-synchronous buses (SSBs) to transfer data between the PowerPC and North Bridge chips, and to support the cache-coherency "snooping" protocols for multiprocessor configurations. The SSBs are unidirectional point-to-point connections between a drive side (D) and a receive side (R). SSBs are put into pairs to form a bidirectional channel between a PowerPC and a North Bridge chip as shown in *Figure 11-14*.

Figure 11-14. Bus Diagram of a Dual-processor 970FX PI-Based System



Source synchronous bus (SSB) data is transferred on every bus clock edge, that is, double the data rate (DDR) of the bus-clock frequency. There are 50 signal lines per SSB. Two lines are used for the differential bus clock lines, 44 signal lines are used to communicate 36 bits of logical data, and four signal lines are used for the differential Snoop Response bus. The 36 data bits consist of 35 bits of the address/data (AD) channel and a single bit for the Transfer Handshake bus (TH).

The SSBs achieve high-speed operation using low-cost packaging solutions by exploiting four features:

1. **Source synchronous signalling**. The differential bus clocks are bundled with the single-ended data signals.
2. **Far-end (parallel) termination**. The single-ended data signals use parallel termination at the far end of the signal line to absorb signal reflections and maintain a quasi-constant current loading for each data signal line.
3. **Balanced coding**. The application of balanced coding to the SSB maintains a quasi-constant current loading across the entire SSB interface. Within the SSB there is no net current flow across the power planes. This dramatically reduces noise problems due to power-supply rail collapse (i.e., Ldi/dt noise) and current voltage offsets between the chips.
4. **Point-to-point unidirectional signalling**. Restricting the signal fan-out to a single point and keeping the signal flow unidirectional mitigates problems associated with high-frequency signal attenuation.

11.4.1 Initialization at Power-On Reset

The receive-side circuitry for the SSBs inside a PI system might require initialization at power-on to deskew data signal lines, align the bus clocks, and synchronize the receive-side FIFOs to the local clock domains of the ASICs and processors. Within a PI system there is the concept of "time zero" which is globally established across all the chips. In the processor interconnect, time zero is derived from the phase synchronization (PSync) and global system clock (SYSCLK) signals (see *Section 11.4.2 Target Cycle*).

The purpose of the initial alignment pattern (IAP) is to establish the settings for the delay lines of the per bit deskew circuitry and optimize the positioning of the sampling clocks on the receive side. During IAP each drive side transmits a bit pattern sequence across each SSB. This pattern is repeated by the drive side for as many bit times (for example, 500000) as needed by the initialization sequential circuitry on the receive side. The I^2C interface controls the duration for how long the pattern is repeated. Upon IAP completion, the receive-side reports its status through the Processor Interconnect Status Register (PISTAT) which is accessible from the I^2C interface. An all-zero result stored in PISTAT indicates that the IAP completed without error. A non-zero pattern indicates that there was an error. The bit fields and their meaning are shown in *Table 11-34 Processor Interconnect Status Register* on page 318.

The sequence of the power-on reset steps is:

1. Stabilize and lock the clocks to the globally distributed SYSCLK.
2. The drive side of each SSB begins transmission of the test patterns for receive-side calibration and optimization. This step is initiated from the I^2C interface by a sequence that is system dependent.
3. Wait for a completion signal from each receive side SSB that the IAP has completed. The completion signal is registered and can be accessed from the I^2C interface. The location of the register and how it is accessed through the I^2C interface is implementation dependent.

The transmission of the test patterns is terminated once the completion signal has been detected. The results of the initialization can be read out from the I^2C interface and the bus is ready for general system use.

In addition to the Processor Interconnect Status Register (PISTAT), there is also the Processor Interconnect Mode Register (PICONFIG) for configuring each SSB that can be accessed from the I^2C interface. The register bit fields and their meanings are shown in *Table 11-33 Processor Interconnect Mode Register* on page 317.

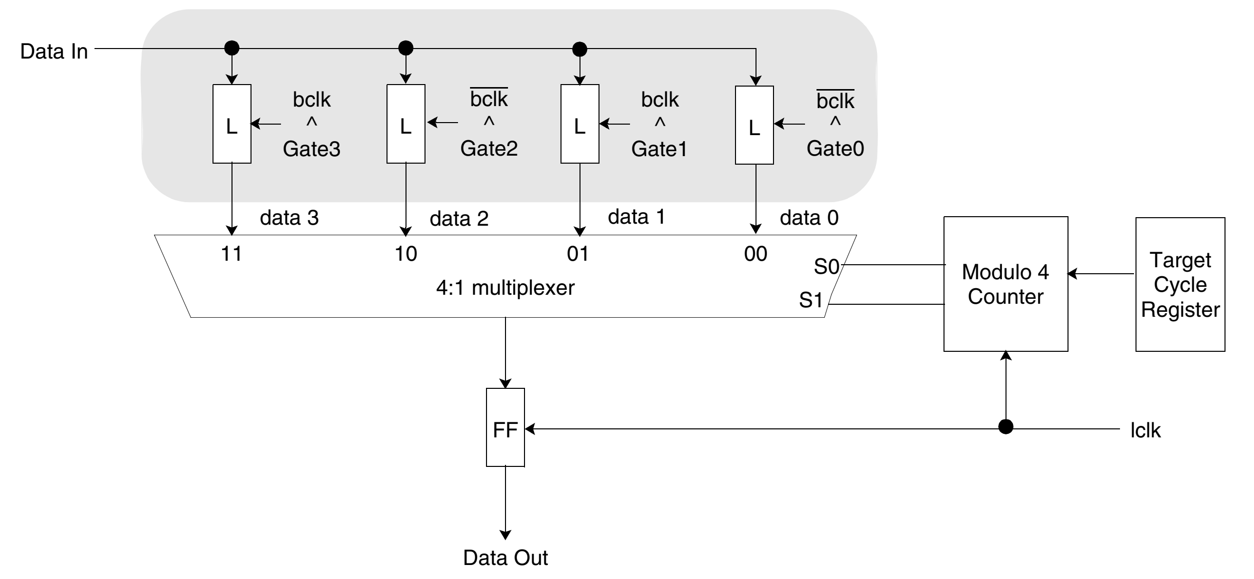
11.4.2 Target Cycle

The flight time of a data signal from the drive side to the receive side of an SSB can extend beyond the period of a single bus clock. The principles of the processor interface allow data and clocks signals to take multiple bus clock cycles to travel from one side to the other.

Furthermore, each receive side can be programmed to transfer SSB data across the time-domain boundary on the same "target beat" relative to time zero, which is the globally synchronized time domain for all of the processors in a processor interconnect system.

This synchronization can be accomplished using a FIFO type circuit such as the one in *Figure 11-15*. The four gate signals (Gate0 through Gate3) are derived from the incoming bus clock (bclk) of the SSB. These signals are half the frequency of the bus clock, have a 50% duty cycle, and are 90 degrees out of phase from each other (See *Figure 11-16*). During the IAP, the gate signals are shifted one bit time at a time until the "1" in the IAP pattern is aligned into the rightmost latch (data 0) and the "0" is captured in the leftmost latch (data 3). This alignment procedure occurs in the shaded box of *Figure 11-15*.

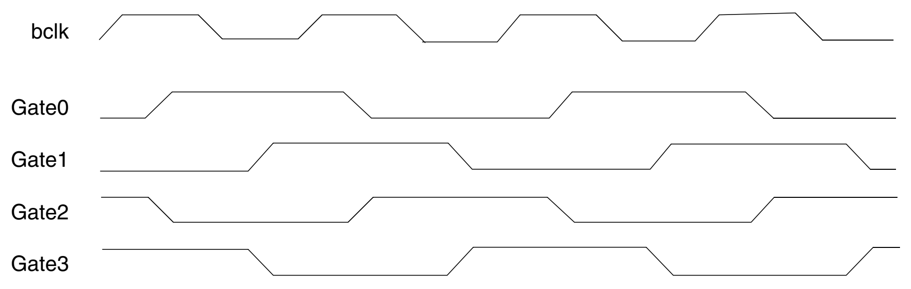
Figure 11-15. Receive-Side FIFO Circuit



The 4:1 multiplexor and the modulo 4 counter are used to cross the time domain from that of the SSB bus clock to the local clock (Lclk) of the chip. A static timing analysis determines the worst case aggregate latency from the drive side through the receive side up to the input point of the data out flip-flop (see *Figure 11-15*).

The combination of the 4:1 multiplexer and modulo 4 counter establishes four possible target cycles for transferring data between the two time domains. The duration of each target cycle equals one bit time. Depending on the results of the worst case analysis, it might be determined that SSB input data cannot be clocked into the data out flip-flop in the same target cycle that it arrives. This will occur if the timing violates the set-up and hold time requirements of the data out flip-flop. In this case, one of the other three target cycles is selected. For example, the following cycle would allow the shortest safe latency, but later cycles would provide larger set-up times. The target cycle is programmed through the I^{2}C interface by loading a 2-bit value into the Target Cycle register, which in turns initializes the modulo 4 counter relative to time zero.

Figure 11-16. Timing Diagram Showing Relationship Between Bclk and the Four Gate Signals



11.5 Processor Interconnect Bus Error Detection and Correction

11.5.1 Error Detection for Balanced Encoding

The PI bus protocol defines an encoding of each beat of information on the inbound address/data (ADI) and the outbound address/data (ADO) bus, so that half the signals carry a '1' bit and half the signals carry a '0' bit on each beat. This is done by converting the 36 bits of information on each bus to the 44-bit pattern that is transferred, in a scheme called "balanced coding." This balanced coding scheme implicitly provides parity checking of the bus signals, in that an unequal number of '1's and '0's in any beat indicates an error. This balanced coding bus mode is selected by setting BUSCONF[49] to '0'. See *Section 11.6.8.22 Bus Configuration Register*.

11.5.2 Error Detection for Alternative Encodings

The 970FX design supports two additional "unencoded" bus modes, in which bits [0:35] of the ADI and ADO bus carry the address and data information, while bits [36:43] carry checking information.

11.5.2.1 Single-Error and Double-Error Detection

The first of these unencoded bus modes implements the following 10-input parity functions to generate the eight check bits:

```
b36 = P(  b0,  b6,  b7,  b29, b30, b31,  b32,  b33,  b34,  b35 )
b37 = P(  b0,  b1,  b6,  b23, b24, b25,  b26,  b27,  b28,  b35 )
b38 = P(  b0,  b1,  b2,  b18, b19, b20,  b21,  b22,  b28,  b34 )
b39 = P(  b1,  b2,  b3,  b14, b15, b16,  b17,  b22,  b27,  b33 )
b40 = P(  b2,  b3,  b4,  b11, b12, b13,  b17,  b21,  b26,  b32 )
b41 = P(  b3,  b4,  b5,  b9,  b10, b13,  b16,  b20,  b25,  b31 )
b42 = P(  b4,  b5,  b7,  b8,  b10, b12,  b15,  b19,  b24,  b30 )
b43 = P(  b5,  b6,  b7,  b8,  b9,  b11,  b14,  b18,  b23,  b29 )
```

where P(0 to 7) computes even parity over its input signals.

In the receiver, the error syndrome is computed by exclusive-ORing the received and generated check bits. A syndrome of 0x00 results when the received and generated check bits match, indicating that no error occurred. A non-zero syndrome indicates that an error occurred. This check bit implementation will detect any single or double bit error over the 44-bit pattern. This first unencoded bus mode is selected by setting BUSCONF[49, 56, 57] to '100'.

Note: Single bit errors that occur using this bus mode yield syndromes that allow the failing bit to be identified. However, some double bit errors yield those same single-bit error syndromes. For this reason, this mode can be used to detect all single and double bit errors, but cannot be safely used to correct single bit errors.

11.5.2.2 Single-Error Correct, Double-Error Detection

The second unencoded bus mode implements the following functions to generate the eight check bits:

b36 = P(b23, b24, b25, b26, b27, b28, b29, b30, b31, b32, b33, b34, b35)
b37 = P(b9, b10, b11, b12, b13, b14, b15, b16, b17, b18, b19, b20, b21, b22)
b38 = P(b3, b4, b5, b6, b7, b8, b18, b19, b20, b21, b22, b33, b34, b35)
b39 = P(b2, b5, b6, b7, b8, b15, b16, b17, b22, b29, b30, b31, b32)
b40 = P(b1, b3, b4, b8, b12, b13, b14, b17, b21, b26, b27, b28, b32, b35)
b41 = P(b0, b1, b2, b4, b7, b10, b11, b14, b20, b24, b25, b28, b31)
b42 = P(b0, b1, b3, b6, b9, b11, b13, b16, b19, b23, b25, b27, b30, b34)
b43 = P(b0, b2, b5, b9, b10, b12, b15, b18, b23, b24, b26, b29, b33)

In the receiver, the error syndrome is computed by exclusive-ORing the received and generated check bits. A syndrome of 0x00 results when the received and generated check bits match, indicating that no error occurred. A non-zero syndrome indicates that an error occurred. *Table 11-13* lists the syndromes from all single-bit errors, along with which failing bit causes that syndrome.

All non-zero syndromes that are not listed in *Table 11-13* indicate double-bit errors.

This check bit implementation can be used to correct any single-bit error and to detect any double-bit error over the 44-bit pattern. This second unencoded bus mode is selected by setting BUSCONF[49, 56, 57] to '101'.

Table 11-13. Bit Error Position Identifier

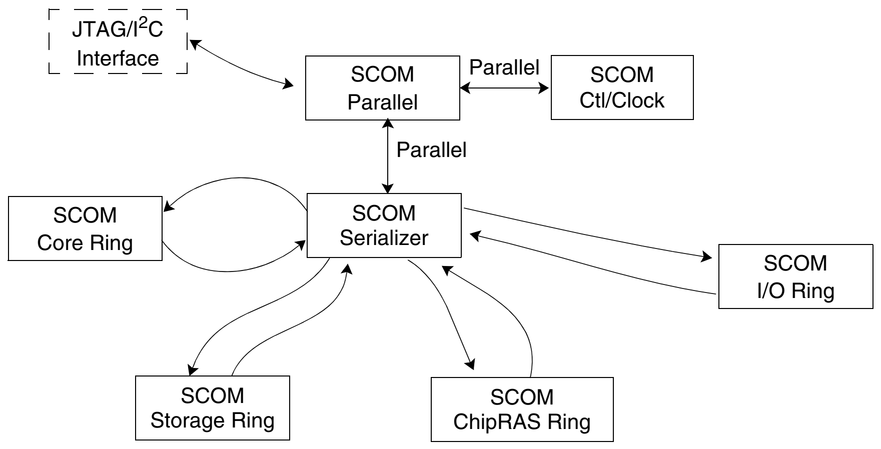
Syndrome	Failing Bit
0x07	0
0x0e	1
0x15	2
0x2a	3
0x2c	4
0x31	5
0x32	6
0x34	7
0x38	8
0x43	9
0x45	10
0x46	11
0x49	12
0x4a	13
0x4c	14
0x51	15
0x52	16
0x58	17
0x61	18
0x62	19
0x64	20
0x68	21
0x70	22
0x83	23
0x85	24
0x86	25
0x89	26
0x8a	27
0x8c	28
0x91	29
0x92	30
0x94	31
0x98	32
0xa1	33
0xa2	34
0xa8	35
0x80	36
0x40	37
0x20	38
0x10	39
0x08	40
0x04	41
0x02	42
0x01	43

11.6 SCOM Facility

Scan communication (SCOM) is used to access vital chip debug and diagnostic facilities while the chip is running without stopping clocks. It is implemented as address and data serial rings running through the chip to limit the wiring. Each facility has a unique address on this ring, which is used to address it.

Figure 11-17 illustrates the SCOM topology. The serial ring is split into independent rings running in the clock domains, so that a clock stop in the one domain will not break the SCOM. A small number of facilities that control the SCOM configuration and the chip clocks are addressed directly without using the serial ring.

Figure 11-17. Processor Unit SCOM Topology



11.6.1 Processor Core SCOM SPR Access

Each processor (core) has two special purpose registers (SPRs) used to access the SCOM interface: SCOMC and SCOMD. SCOMC and SCOMD are both 64-bit read/write SPRs and are used for SCOM Control and SCOM Data respectively. The interface is implemented as a direct connection to the parallel-to-serial converter, which handles the arbitration between the core and service processor.

11.6.2 Operating System Protocol to Access SCOM SPRs

In the PowerPC 970FX, SCOMC and SCOMD are complete operations. They do not require a software protocol in order to function properly except to disable external (asynchronous) interrupts. Software must check the error bits after performing an SCOMC to ensure that the command successfully completed. *Table 11-14 Operating System Code to Access SCOM* outlines a general software protocol for using these registers.

Table 11-14. Operating System Code to Access SCOM

For SCOM READ	For SCOM WRITE
set MSR[EE] = '0' MTSCOMC MFSCOMD MFSCOMC if Error = '1', branch to SCOM error routine set MSR[EE] = '1'	set MSR[EE] = '0' MTSCOMD MTSCOMC MFSCOMC if Error = '1' branch to SCOM error routine set MSR[EE] = '1'

Asynchronous interrupts must be disabled during these blocks. Otherwise, an interrupt could arrive and make the SCOM port busy. If that occurs between the ‘move from’ (MF) and ‘move to’ (MT) instructions that cause the reads and writes, then the SCOM interface out of the core could malfunction (or at least not perform as software intended).

11.6.3 SCOMD Format

The SCOMD is simply a 64-bit register. The interpretation of the contents of this register is determined by the SCOMC Status and Control bits. It is the source for outgoing data during an SCOM write access (SCOMC[RW] = ‘1’ when **mtscomc** is issued). It is the destination for incoming data after an SCOMC read access (SCOMC[RW] = ‘0’ after **mtscomc** completes).

11.6.4 SCOMC Format

The SCOMC is divided into four 16-bit fields, as shown in *Table 11-15* and *Figure 11-18*:

Figure 11-18. SCOMC SPR Format

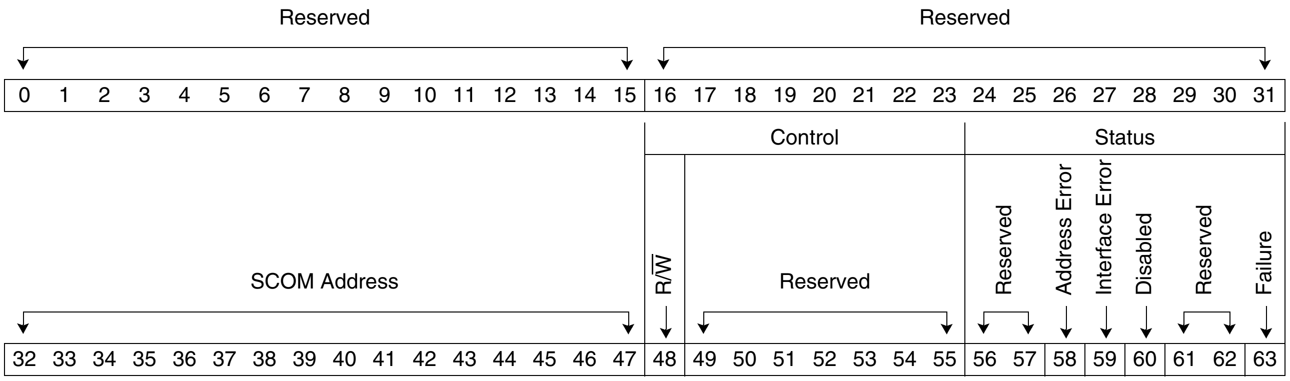


Table 11-15. SCOMC SPR Format

SCOMC Bits	Type	Usage	Description
0:31	Unused	Reserved	Unused
32:47	Write-Only	Control	SCOM Address (0:15)
48	Write-Only	Control	SCOM Read/Write Request Bit 0 Write request 1 Read request
49:55	Unused	Reserved	Unused Control Bits
56	Read-Only	Reserved	Unused Status bit
57	Read-Only	Status	SCOM Protocol Error
58	Read-Only	Status	SCOM Address Error
59	Read-Only	Status	SCOM Interface Error
60	Read-Only	Status	SCOMC disabled by service processor
61	Read-Only	Status	Reserved (Zero)
62	Read-Only	Status	Reserved (Zero)
63	Read-Only	Status	Failure (SCOMC disabled or Interface Error [formerly Busy])

The reserved fields should be written to zeros by the software on an **mtscomc** and return zeros on an **mfscomc**. The Address and Control fields are undefined while Failure equals '1'.

All SCOMC Status bits will be cleared by the hardware upon an **mtscomc** with the exception of Failure, which is set to indicate to the operating system that the SPR SCOM access is active. Additional status bits will be set depending on the status of the SCOM operation:

- **Protocol Error:** The SCOM hardware has violated a basic protocol, such as giving a grant when not asked or returning a data packet when expecting an address packet. This error bit is *not* cleared on the next **mtscomc**

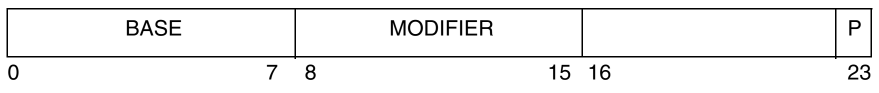
  Note: This bit will probably cause a checkstop to occur and sets a corresponding bit in the Fault Isolation Register (FIR).

  This bit indicates a problem has occurred in the SCOM hardware, and this interface can no longer be trusted.
- **Address Error:** The SCOM address was not recognized by an SCOM satellite. This indicates that the write did not happen or that the read returned no data (depending on the R/W bit). This error bit is cleared on the next **mtscomc**. This indicates a probable software error.
- **Interface Error:** If the SCOM logic in the arbiter detects an error condition, such as a timeout on the SCOM interface, or if the core hang recovery engages while SCOMC is active, it sends a SCOM reset (screset). This causes the SCOMC operation to be terminated and the logic to record an error. The error bit is cleared on the next **mtscomc** and is recoverable (the command must be retried).
- **SCOMC Disabled:** The service processor has the ability to disable a core from becoming an SCOM master, causing the core to treat **mtscomc** as a no-op. **MFSCOMC** will set this bit, along with Failure and Interface Error to ensure the software realizes this condition.
- **Failure:** Summary indicating if there were any errors since the last **mtscomc**. Formerly the "Busy" bit, which indicated if the SCOMC interface was in use.

11.6.5 SCOM Address Allocation

The SCOM supports a 23-bit address with the 24th bit being a parity bit. This is the address that would be sent to the SCOM through the JTAG port. The internal SCOM bus, the part that is serialized, needs no more than 16 bits of addressing. So, to simplify the logic, addresses sent internally are truncated to16 bits. Thus, bits [0:15] are sent to the parallel SCOM controller and bits [16:22] must be zero for the processor SCOM addresses.

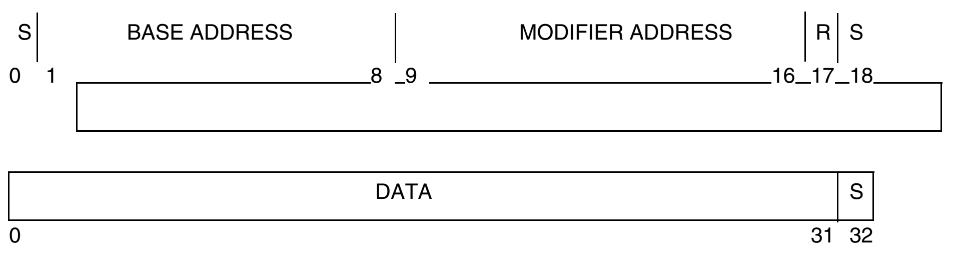
Figure 11-19. SCOM Addressing



For the STS, each unit's SCOM serial address and data are daisy chained together. For example, the STS SCOM address and SCOM data out connect to the SCOM address and SCOM data in of the L2 cache. The SCOM address and SCOM data out of the L2 cache connect to the SCOM address and SCOM data in of the bus. The SCOM address and SCOM data out of the bus connect to the SCOM address and SCOM data in of the storage subsystem (STS).

Inside the STS, the serial SCOM address bit [0] is the start bit. Bits [1:8] are the SCOM base address for STS SCOM registers. Bits [9:16] are the actual SCOM address for each register. Bit [17] is for SCOM read or write operations ('1' = READ, '0' = WRITE) and bit [18] is the stop bit. For the internal serial SCOM data bus, each SCOM register in the STS is 32 bits wide. Bit [32] of the SCOM data bus is the stop bit.

Figure 11-20. Serial SCOM Address and SCOM Data Bus



11.6.6 SCOM Register Descriptions

Table 11-16 lists the subset of SCOM registers which are described in this section.

Table 11-16. Subset of SCOM Registers

Modifier SCOM Address [9:16]	Register	Domain	Page
x021001	Core RAS Control (Pulsed) Register	Core	300
x021100	Core RAS Mode Register	Core	301
x021200	Core RAS Status Register	Core	304
x021301	Core Hang Recovery Control Register	Core	307
x021400	Core Power Down and Idle Status Register	Core	309
x022001 / x022100 / x022200	Service Processor Special Attention Register / And Mask / Or Mask)	Core	311
x022601 / x022700 / x022800	Asynchronous Machine Check Source Register / And-Mask / Or-Mask	Core	310
x023000	Instruction Address Breakpoint Register (IABR)	Core	310
x023101	Hardware Implementation Dependent Register 0 (HID0). See *Table 2-7 HID0 Bit Functions*.	Core	46
x023201	Hardware Implementation Dependent Register 1 (HID1). See *Table 2-8 HID1 Bit Functions*.	Core	47
x030001 / x031000/ x032000	Core Fault Isolation Register / And-Mask / Or-Mask	Core	312
x030400 / x031401 / x032401	Core Fault Isolation MASK Register / And-Mask / Or-Mask	Core	313
x030800	Core Checkstop Enable Register	Core	313
x030901	Core Machine Check Enable Register	Core	313
x040000 / x041001 / x042001 x040801	L2 Fault Isolation Register / And-Mask / Or-Mask / Checkstop	STS/BIU	314
x040401 / x041400 / x042400	L2 Error Mask Register / And Mask / Or Mask	STS/BIU	314
x043000	STS Mode Register	STS/BIU	316
x046A00 / x0F7A00	Processor Interconnect Mode Register	STS/BIU	317
x046B01	Processor Interconnect Status Register	STS/BIU	318
x0A0001 / x0A1000 / x0A2000	970FX Bus Fault Isolation Register / And Mask / Or Mask	STS/BIU	318

Table 11-16. Subset of SCOM Registers

Modifier SCOM Address [9:16]	Register	Domain	Page
x0A0400	970FX Bus Error Mask / And Mask / Or Mask	STS/BIU	319
x0A0800	Checkstop Enable Register	STS/BIU	320
x0A8000	Processor Configurable Timing Delay Parameter Register (BUSCONF)	STS/BIU	321
x0A9000	970FX Bus Status Register	STS/BIU	322
x400000	Power-On Reset Status Register	ChipRAS	322
x400101	Power-On Reset Continue Register	ChipRAS	323
x400201	Power-On Reset I^2C/JTAG Arbitration Register	ChipRAS	323
x400801	Power Management Control Register	ChipRAS	324
x401400	Power-On Reset Sequence Register 0	ChipRAS	325
x402400	Power-On Reset Sequence Register 1	ChipRAS	325
x404400	Power-On Reset Sequence Register 2	ChipRAS	325
x408001	Power Tuning Status Register. See *Table 9-7 Power Status Register (PSR).*	ChipRAS	178
x500001	Global Fault Isolation for Checkstop Conditions (Global FIR)	ChipRAS	326
x500400	Error Enable Mask	ChipRAS	327
x500601	Mode Register for Fault Isolation Registers	ChipRAS	327
x500700	Debug Mode Register	ChipRAS	328
x504101	Chip ID Register	ChipRAS	330
x503001	Hang Pulse Generation	ChipRAS	329
x600001	SCOM Mode Register	ChipRAS	330
x600100	SCOM Controller Error Register	ChipRAS	331
x600400	Clock Ratio Register	ChipRAS	332
x800000	Clock Command Register	Always available	333
x800003	Status Register	Always available	334
x800006	Phase Synchronization Control Register	Always available	335
x800009	Clock Command Control Register	Always available	336
x80000A	Energy Star Register	Always available	338
x80000C	Status Register Mask	Always available	339
x80000F	I/O Control Register	Always available	340
x820004	ABIST Status Register	Always available	340
x840002	LBIST Options Register	Always available	341
x840008	LBIST Channel Length Register	Always available	342
x84000B	LBIST Test Length Register	Always available	342
x84000D	Clock Ramping Configuration Register	Always available	343

11.6.7 Register Description Conventions

The following terms are used in the following SCOM register descriptions.

Reserved	Indicates that the latch might be implemented in the future. Engineering software should write zeros and expect unknown data.	
Not Implemented	Indicates that the latch is not implemented. Engineering software should write zeros and expect zeros.	
Type	SCOM request type	
	RW	Read/Write
	RO	Read Only. Write requests to a Read Only SCOM register will be treated as a no-op and the data will be discarded.
	WO	Write Only. Read requests to a Write Only SCOM register will result in an error condition.
	RWor	Read/Write OR. The write operation is a special type that will OR into the specified bits in the register. An associated Write Only AND mask is provided to allow the engineering software to clear bits in this type of SCOM register.

11.6.8 SCOM Error Handling

If an error occurs while servicing an SCOM command, the ANY_SCATTN bit in the Access Status register is raised. The engineering software should issue an SCOM reset using the instruction register (IR) opcode of x'1B', and then read the SCOM Control Error register (x'600100') to view the fault. To clear ANY_SCATTN, first write all zeroes to the SCOM Control Error register (SCOM register x600100), then write all zeroes to the JTAG SCOM Status register (x'000080'). Finally write all zeroes to the JTAG Access Status register (x'000002').

The following error indications in the SCOM Controller Error register, (x'600100'), bits [0:23], might be due to programming errors. Bits [24:47] contain the failing SCOM address.

x'010000'	Invalid address. Does not match any known address ranges.
x'001041'	The address was decoded as a serial type and sent onto the SCOM serial ring, but no SCOM satellite accepted ownership of the address. This is most likely due to the use of an invalid address.
x'001040'	The address was decoded as a serial type read, but no data was returned. This is most likely due to an invalid read request being sent on the serial SCOM bus. For example, this error will happen if the enineering software does a read to a write only register.

11.6.8.1 Processor Core RAS Control (Pulsed) Register

Table 11-17. Core RAS Control (Pulsed) Register

Address:	x021001
Type:	WO
Reset:	N/A
Bit	Description
0:39	Not implemented
40	Enter core power down mode (works without setting MSR[POW], see core RAS mode register bits [5:6])
41	Break hold on the IFU/LSU status after core hang detect due to external source (core RAS status register bits [59:63])
42	Cancel pending requests (quiesce, SRESET, machine check, core step) – and accept pending mode changes
43	Force next machine check or SRESET (and all exceptions) to be marked non-recoverable
44	Force core quiesce manually (RAS logic override)
45	Force core maintenance mode (RAS logic override)
46	Force quiesce state machine to quiesce state (RAS logic override)
47	Force quiesce state machine to run state (RAS logic override)
48	Inject fetch hang for testing hang recovery logic
49	Inject force reject hang for testing hang recovery logic
50	Inject dispatch hang for testing hang recovery logic
51	Inject completion hang for testing hang recovery logic
52	Core running test – use to see if core is running (clears core RAS status register [15] until a group completes) **Note:** Do not do this if in Maintenance Single Step mode (Core RAS Status[54] MUST = ‘0’)
53	Inject NTC (Next To Complete, all instructions in the machine); flush manually without the before and after waits (do not attempt while running)
54	Inject NTC+1 (Next To Complete Plus One, all but the oldest instruction); flush manually without the before and after waits (do not attempt while running)
55	Clear instruction stop due to checkstop. Also clears hang detection, hang history, and miscellaneous status latches.
56	Not implemented
57	Not implemented
58	Inject SRESET manually (no auto quiesce); RAS logic override only • Do not attempt if processor is running • ISU may OR 0x0100 with another interrupt vector (if they occur simultaneously)
59	Inject machine fheck manually (no auto quiesce); RAS logic override only • Do not attempt if processor is running • ISU may OR 0x0200 with another interrupt vector (if they occur simultaneously)
60	SRESET request • This causes the core to first quiesce then vector to 0x0100 and start instructions • If quiesce is unsuccessful, SRESET will not occur and a special attention will be sent to the Service Processor indicating timeout on quiesce request (assuming hang pulses are activated) • After request, verify core RAS status[15] = ‘1’ to ensure SRESET was successful (core has started)
Note: Core RAS Status[15] will be cleared for [60:62] until a group completes.	

Table 11-17. Core RAS Control (Pulsed) Register

Address:	x021001
Type:	WO
Reset:	N/A
Bit	Description
61	Instruction Step (core step) **Note:** Core must be in maintenence mode (quiesced) – see Core RAS Status[12] Single Group Completion mode is active, meaning a core flush and refetch occurs between each step. This allows the NIA to be changed (via scan) between steps if desired. Behavior is determined by (Core RAS Mode[43] and not (Core RAS Mode[54] OR HID0[0])) 0: PowerPC instruction step (default) Completes more than one group if the PowerPC instruction is a microcoded multi-group sequence 1: Group step (for debug); completes a single group or microcode group sequence after request, verify that Core RAS Status[15] = '1' to ensure step was successful (core has started)
62	Instruction Start (core resume) **Note:** Core must be in maintenance mode (quiesced) – see Core RAS Status[12] After request, verify that the Core RAS Status[15] = '1' to ensure start was successful (core has started)
63	Instruction Stop (core stop); causes core quiesce and leaves core in maintenance mode
Note: Core RAS Status[15] will be cleared for [60:62] until a group completes.	

11.6.8.2 Core RAS Mode Register

Table 11-18. Core RAS Mode Register

Address:	x021100
Type:	RW
Reset:	Reset to all '0's during Power-On Reset
Bit	Description
0:4	Debug Throttle Modes – periodically alter instruction flow based on *Debug Throttle Control* 00000 : None 01xxx : Stop fetch (both IFU and LSU prefetch) 0x1xx : Stop dispatch 0xx1x : Stop completion 0xxx1 : Force LSU reject (stops issue from ISU to LSU) 11xxx : One PowerPC per group (might quiesce depending on bit [44]) 1x1xx : Single group completion (will quiesce to change) 1xx1x : Serialized group issue (will quiesce to change) 1xxx1 : Serialized group dispatch (does NOT quiesce to change)
5	Power down Nap/Doze mode enable override (when set ignores HID0[9]), see bit [6] for mode select **Note:** To enter power down, either software must set MSR[POW] or use Core RAS Control register[40]
6	Nap mode override selector 0: Force Nap mode when bit [5] is set 1: Force Doze mode when bit [5] is set
7	Inorder issue select 0: Serialized group execution when selected will serialize at dispatch. This is the most powerful form of serialization, only allowing one group in flight in the machine at a time. This includes branch and CR-logical instructions and also prevents groups from sharing ISU resources such as mappers and renames. 1: Serialized group execution when selected will serialize at issue. This does not include branches and CR-logicals and does not completely serialize the ISU.

Table 11-18. Core RAS Mode Register

Address:	x021100
Type:	RW
Reset:	Reset to all ‘0’s during Power-On Reset
Bit	Description
8	Block machine check exception or maintenance exception injection based on debug triggers when one is already set in the Machine check source accumulation register. This will prevent multiple triggers from causing a checkstop with MSR[ME] = 0, allowing the interrupt handler to ignore spurious triggers until it has a chance to clear the source register.
8:11	Spare
12:19	Not implemented
20	Initialize the ISU global completion table
21	Initialize the ISU SB, SX, and SF Arrays
22	SCOMC disable. Disables core SCOM master accessible via core SPR bus. MTSCOMC in the core is treated as a no-op by RAS logic
23	Performance inhibit (debug only)
24	Enable CIABR
25	Programmable **mtspr** TRIG0 mode (SPR 976 mtspr_data(0:63) all zeroes always cause TRIG0) 0: SPR 976 mtspr_data is not used to form TRIG1 and TRIG2 1: SPR 976 mtspr_data(63) causes TRIG1 and mtspr_data(62) causes TRIG2
26	Do not stop fetch, dispatch, and completion on checkstop. Set this when the processor is set to clock stop on checkstop in order to prevent core scan rings from being altered when a checkstop occurs, since it takes several cycles to clock stop after checkstop occurs.
27	Do not attempt to quiesce then machine check or SRESET during core hang recovery **Note:** This will cause checkstop if the first two attempts are unsuccessful
RAS trigger sources to debug logic	
28:30	000: No internal trigger selected 001: External trigger causes internal trigger 010: Decrementer exception causes internal trigger 011: External exception causes internal trigger 100: Machine check exception causes internal trigger 101: Problem state hang detect causes internal trigger 110: Bad hang state detect causes internal trigger 111: Quiesce causes internal trigger
31	Internal trigger does not cause external trigger
32	Core debug trigger does not cause external trigger
RAS activities based on debug trigger	
33	External trigger causes core quiesce. Bit [37] determines behavior after quiesce.
34:36	Core debug trigger causes: 000: No action selected 001: Core quiesce. (Bit [37] determines behavior after quiesce) 010: NTC flush (see Core Hang Recovery Control register bits [46:48]) 011: NTC+1 flush (see Core Hang Recovery Control register bits [46:48]) 100: Machine check exception (see bit [8] for mode to prevent checkstop) 101: Maintenance exception 110: Enter reduced execution mode for specified number of group completions (see Core Hang Recovery Control register bits [46:48]) 111: Pause (quiesce and hold prefetch) until core idle

Table 11-18. Core RAS Mode Register

Address:	x021100
Type:	RW
Reset:	Reset to all '0's during Power-On Reset
Bit	Description
37	Auto core resume on core quiesce due to debug trigger. When debug or external trigger causes quiesce, and this bit is: 0: Quiesce, set Special ATTN, and enter core maintenance mode. 1: Auto-start core immediately after the ISU completes the quiesce (implicitly causes a flush). Special ATTN is not set and core resumes normal instruction execution.
RAS control logic modes	
38:39	00: No hang pulse induced random periods 10: Enter reduced execution mode for random period after every hang pulse 01: Quiesce for random period after every hang pulse 11: Undefined
40	STS Toad mode enable Throttles back core execution when the STS detects a memory subsystem hang
41	Return to good hang state immediately after meeting the completion criteria. Do not wait on hang pulse to leave reduced throughput mode
42	Use any group completed (instead of PowerPC) to debug logic (trace trigger and to indicate progress for hang detection logic 0: PowerPC group completed 1: Any group completed (including intermediate multi-group sequences)
43	Group step instead of PowerPC instruction step; causes instruction step not to enter one PowerPC mode automatically during maintenance mode
44	Do not quiesce when changing one PowerPC mode
45	Do not wait for SRQ empty before auto-restarting after a quiesce due to mode change, debug trigger, or when in single_group_completion mode.
46	Machine check on external checkstop
47	Quiesce on external or FIR checkstop (default is hold completion). **Note:** Causes Special Attention to service processor.
48:49	Action to perform for NTC Flush Hang Recovery when LSU is not in safe (LMQ not empty) 00: Do not attempt flush (This still holds completion, dispatch, and force reject for the same time. It will eventually transition to the next hang recovery state if the LSU is not safe) 01: Stop dispatch and completion (sets Special Attention register) 10: Checkstop 11: Do flush anyway (Unsafe – debug only) Use Unsafe NTC+1 Mode (Core Hang Recovery Control register [17]) for NTC+1 flushes.
Core toolbox – hooks for service processor	
50	Manually force LSU reject
51	Manually hold fetch (both IFU and LSU)
52	Manually stop dispatch
53	Manually stop completion
Core toolbox – behavior mode controls **Note:** Must quiesce core before changing these (corresponding HID0 change invokes state machine to do automatic quiesce, mode change, and resume)	

Table 11-18. Core RAS Mode Register

Address:	x021100
Type:	RW
Reset:	Reset to all '0's during Power-On Reset
Bit	Description
54	One PowerPC per group mode; OR'd with HID0[0]. No more than one PowerPC instruction will be placed in a group. Some instructions get expanded into a multiple group microcoded sequence.
55	Single group completion mode; OR'd with HID0[1] which causes automatic group step. Only one group (or microcoded group sequence) will be allowed to complete at a time. A core quiesce (involving an instruction flush and refetch) occurs between completion of each group. Subsequent groups of instructions will not be allowed to complete, but will be allowed to execute at the same time unless mode bit [56] = 1.
56	Serialized group execution mode (in-order dispatch OR issue of groups) • OR'd with HID0[3] when inorder_issue_select = 0 • OR'd with HID0[16] when inorder_issue_select = 1 A group must be completed before the next one may be issued (OR dispatched), effectively serializing execution of the groups in the processor. Subsequent groups of instructions will not be allowed to execute at the same time. See inorder_issue_select, bit [7] of this register for more details.
Core maintenance modes	
57	Run Decrementer/Timebase while instruction stepping (take ISU quiesced into account during maint_mode). Also allows DEC exceptions (only) while instruction stepping. **Note:** DEC/Timebase stopping only takes effect if HID0[18] = 0
58	Keep asynchronous exceptions during quiesce or instruction stepping (decrementer, external, external machine check, and performance exceptions). If kept, present only after completion of maintenance activity.
59:60	Hang recovery logic test modes 00: None 01: Allow problem hang recovery to break test hang 10: Allow bad hang recovery to break test hang 11: Allow machine check hang recovery to break test hang
61:62	STS source core hang recovery modes 00: None (only attempt hang recovery if nothing is pending to the STS) 01: Special ATTN if ambiguous source or the IFU/LSU indicates STS pending transactions 10: Attempt core hang recovery if ambiguous source (there are possibly STS pending transactions) 11: Always attempt core hang recovery (even if ambiguous source or IFU/LSU indicates STS pending transactions) **Note:** To cause SP-ATTN for core source hang detect (no STS pending) use hang recovery control register bit [55].
63	Disable hang detection and recovery based on hang pulse

11.6.8.3 Core RAS Status

Table 11-19. Core RAS Status Register

Address:	x021200
Type:	RO
Reset:	All 0's except bits 7, 12, 20, 26
Bit	Description
0	Entire core is quiesced. (status[1] AND status[8:10])
1	ISU is quiesced for reasons listed below
2	Checkstop caused quiesce request

Table 11-19. Core RAS Status Register

Address:	x021200
Type:	RO
Reset:	All 0's except bits 7, 12, 20, 26
Bit	Description
3	Quiesce request was made to ISU by debug logic
4	ISU completed quiesce as requested (by service processor or debug logic)
5	ISU quiesced by ATTN instruction
6	ISU quiesced by LSU DABR debug mode (Note: will not be set by CIABR but set by SP-ATTN[2])
7	POR state (scan flush indicator); forces a stop dispatch. When set, it indicates a quiesce must be done first before starting the core.
8	ISU GCT is empty (quiesce indication)
9	ISU SRQ is empty (quiesce indication)
10	LSU/IFU is not waiting for a STS transaction (NOR of unheld but masked status sourcing bits [59:63])
11	LSU is quiesced (and STS is idle) • LMQ is idle (no outstanding load misses including speculative) • SRQ has no stores past completion Service processor should poll this bit to verify both STS and core are quiesced
12	Core is in maintenence mode (ISU is quiesced or being stepped) • Enter and stay in maintenance mode when the ISU quiesces (includes during instruction stepping) • Leave maintenance mode when the core resumes, SRESET, or external machine check occurs (See Core RAS Mode[59])
13	Completion is stopped by checkstop
14	Quiesce request pending
15	Instruction (or group) completed since last maintence operation (for example, instruction step, start, SRESET; for core running the test use Core RAS Control register (x021001) bit[55] first).
16	NTC flush not safe and caused a checkstop (core hang)
17	NTC not safe and caused stop (ATTN)
18	Hang detect/recovery caused a stop (ATTN) on problem, bad, or STS hang detect as programmed
19	Demand instruction fetch has been pending since last hang pulse
20:25	Hang state (Good || Problem || Bad || Quiesce/MCHK || Wait || Fail)
26:31	Hang state history (Good || Problem || Bad || Quiesce/Mchk || Wait || Fail)
32:39	Completion count LFSR (8 bits) – matched against completion count limit
40	Seen programmed number of groups complete after hang state (clears on return to good state)
41	Fetch (both the IFU and LSU) is being held by core RAS logic
42	Dispatch is stopped by core RAS logic
43	Completion is stopped by core RAS logic
44	Instruction completion is being held due to core RAS debug throttle hooks
45	Performance exception is being held (pending completion of core maintenance operation)
46	Decrementer exception is set
47	External exception is set

Table 11-19. Core RAS Status Register

Address:	x021200
Type:	RO
Reset:	All 0’s except bits 7, 12, 20, 26
Bit	Description
48	Thermal exception is set
49:51	Quiesce state machine (Quiesced ǁ Running ǁ AsyncInt)
52:53	Quiesce state machine (Chmod ǁ Pause)
54	Maintenance single stepping mode is in effect. The core is in serial dispatch mode (and automatically one PowerPC mode if Mode[43] = 0)
55	STS status [59:63] being held due to core-STS hang detect for debug/fault isolation
56	Instruction start (core resume) pending or instruction (or group) step pending
57	Mode change pending
58	Machine check pending
Note: Bits [59, 60:63] can be masked by scan latches (TDR.IFU_STAT_DIS and TDR.LSU_STAT_DIS) and normally follow the masked IFU/LSU signals, but are held when a core-STS hang detect occurs to aid in debug/fault isolation (as indicated by bit [55]). Use Core RAS Control register bit [41] to break the hold after an STS hang detect occurs. Note that the hold is also broken during hang recovery as soon as a single group completes, but may take more than one group complete to deem hang recovery successful in which case if the processor is still hung these status indicators follow the current status.	
59	ifu_demand_pend – IFU has outstanding demand instruction fetches to STS
60	lmq_not_empty – LSU has outstanding loads to STS (also indicates not safe to NTC flush)
61	srq_pend_busy – LSU has entries in SRQ being held due to STS busy on store port
62	srq_pend_ugly – LSU has outstanding ops to the STS waiting for response (**icbi**, **stwcx**, or **sync** is outstanding to STS)
63	xlate_pending – LSU has outstanding translation request to the STS

11.6.8.4 Core Hang Recovery Control Register

Table 11-20. Core Hang Recovery Control Register

Address:	x021301
Type:	RW
Reset:	0:7 – Reset to 0x9F = 3 hang pulses 8:15 – Reset to 0x2D = 100 hang pulses 16:17 – Reset to 00 18:27 – Reset to 0x1FF = 1*255 default 36:43 – Reset to 0xFE = 255 = max 44:63 – Reset to all '0's during Power-On Reset except bit [52] and [60]
Bit	Description
0:7	Core hang limit (8-bit LFSR match value) • Number of hang pulses without a PowerPC instruction (or group) completion used to detect a core hang • Used when lsu_safe indicates no outstanding load/store instructions to the STS • This should be set to a minimum of 3 hang pulses (0x9F)
8:15	Core-STS Hang Limit (8-bit LFSR match value) • Number of hang pulses without a PowerPC instruction (or group) completion used to detect a Core-STS hang • Used when lsu_safe indicates outstanding load/store instructions to the STS • Must be set to a value greater than the core hang limit
16	After flush control 0: Wait 255 cycles after attempting hang recovery flush 1: Use Before Flush Wait Time as the After Wait Time as well
17	Unsafe NTC+1 mode 0: Do not attempt NTC+1 flushes if not safe 1: Attempt NTC+1 even if LSU indicates not safe (LMQ is not empty)
18:27	Wait time (# * 255 cycles) before hang recovery flush (10-bit LFSR match value). Stops dispatch and completion (and LSU forces a reject for NTC) during this wait before attempting flush. **Note:** 3FF (count of zero) is invalid and will cause undefined results. 1FF = 1*255 = 255 cycles minimum wait. 3FE = 1022 * 255 = 260610 cycles of maximum wait.
28	High resolution flush counter (before and after wait times are single cycle instead of having a 255 multiplier)
29	Spare
30:35	Not implemented
36:43	Completion count limit (8-bit LFSR match value); number of group completions to wait before returning to a good hang state after a hang recovery is attempted
Random pulse hang buster vontrols	
44:45	Random buster controls (use before and after wait times specified above) 00: Random pulse has no effect (disabled) 01: Random pulse injects NTC+1 flush 10: Random pulse injects NTC flush (debug only, see note below) 11: Random pulse stalls instruction dispatch and completion with no flush (only waits the before time) **Note:** Bit [17] of this register and bits [48:49] of the core RAS mode register determine the behavior of these flushes when the LSU indicates not safe. The recommended setting is to not flush when the LSU is not safe if using any of these random busters that are enabled. **Note:** Random NTC flush is a severe error recovery mechanism. Recovery may depend on actual error conditions.
Degraded mode during random period (either random pulse or hang pulse induced) **Note:** 1. Core RAS Mode register [38] must be set in order to enter these random periods 2. These are also set until the programmed number of completions (above) have occurred after the flush on trigger 3. inorder_issue_select (bit [7] of Core RAS Mode register) affects the meaning of bit [46, 49]	

Table 11-20. Core Hang Recovery Control Register

Address:	x021301
Type:	RW
Reset:	0:7 – Reset to 0x9F = 3 hang pulses 8:15 – Reset to 0x2D = 100 hang pulses 16:17 – Reset to 00 18:27 – Reset to 0x1FF = 1*255 default 36:43 – Reset to 0xFE = 255 = max 44:63 – Reset to all '0's during Power-On Reset except bit [52] and [60]
Bit	Description
46	Serialized group execution mode during random period (or after flush on trigger)
47	Single group completion mode during random period (or after flush on trigger)
48	One PowerPC per group mode during random period (or after flush on trigger)
Degraded mode during first level (problem) hang recovery (hang pulse based only)	
49	Serialized group execution mode during problem hang recovery
50	Single group completion mode during problem hang recovery
51	One PowerPC per group mode during problem hang recovery
First level (problem) hang state recovery actions (hang pulse based only)	
52	Attempt NTC+1 flush for first hang detect (problem hang state) – defaults to '1'
53	Perform SRESET instead of machine check for third level hang recovery, unless disabled via Core RAS Mode register bit [27]
54	Cause maintenance exception to be taken after problem hang recovery
55	Stop completion and cause Special Attn on first-level hang detect (sets SP-ATTN register)
Degraded mode during second level ("Bad") hang recovery	
56	Serialized group execution mode during bad hang recovery
57	Single group completion mode during bad hang recovery
58	One PowerPC Per group mode during bad hang recovery
Second level ("Bad") hang state recovery actions	
59	Stop completion and cause Special Attn on third level hang (sets SP-ATTN register) **Note:** Should also set bit [27] of core RAS mode register
60	Attempt NTC Flush on bad hang - defaults to '1'
61	Cause maintenance exception to be taken after bad hang recovery
62	Stop completion and cause Special Attn on bad hang (sets SP-ATTN register)
Non-recoverable machine check for hang recovery control	
63	Make all hang recovery machine checks or SRESETs non-recoverable (only has effect if quiesce was successful)

11.6.8.5 Core Power Down and Idle Status Register

Table 11-21. Core Power Down and Idle Status Register

Address:	x021400
Type:	RO
Reset:	N/A
Bit	Description
0:6	Power Down State Machine (0:6)
7	IFU fetch and LSU prefetch stopped
8	Power down wait timer
9:15	Interrupt status (Thermal exception active, external exception active, external machine check active/held OR machine check pending, external SRESET active/held OR SRESET pending, performance monitor exception held, decrementer exception active)
16	Asynchronous interrupt pending in ISU
17	Core idle indication
18	Wait for idle from ISU
19	Idle count match
20:27	Idle LFSR value
28:63	Not implemented

11.6.8.6 Asynchronous Machine Check Source Register / And-Mask / Or-Mask

Table 11-22. Asynchronous Machine Check Source Register / And-Mask / Or-Mask

Address:	x022601 / x022700 / x022800
Type:	RW / WO (and) / WO (or)
Reset:	Reset to all '0's during Power-on reset
Bit	Description
This Register indicates the source of all asynchronous machine check events. When any source occurs: • quiesce, then vector to 0x0200 and restart instruction execution • if quiesce is unsuccessful, machine check will NOT occur and a special attention will be sent to the Service Processor indicating timeout on quiesce request (assuming hang pulses are activated). Service Processor can then check Core RAS Status[58] to see that machine check was not accepted (still pending) **Note:** This is a history accumulation register and must be cleared after each interrupt to absolutely determine the new source. The AND mask should be used to clear bits in the register and the OR-mask should be used by software to set new bits in order to prevent an interrupt from another source from being missed.	
0	External Machine Check (from chip C4 pin)
1	FIR induced Machine Check – See FIR
2	Hang Recovery Machine Check Interrupt Attempt
3	Debug Logic Trigger Machine Check Interrupt
4	Machine Check instead of external checkstop (see Core RAS mode register)
5:7	Software/operating system programmable
8:63	Not implemented

11.6.8.7 Instruction Address Breakpoint Register

Table 11-23. Instruction Address Breakpoint Register (IABR)

Address:	x023000
Type:	WO
Reset:	Reset to all '0's during power-on reset
Bit	Description
Usage Note: This uses the IFU FETCH address, not the CIA (current instruction address that is executing) IABR match occurs on fetch of the instruction, even speculative. **Note:** There can be multiple IABR matches for a single instruction before it is actually executed (or completed) By default the IABR matches if the address is in the current fetch group equal to or AFTER the current IFAR. There is an additional mode bit accessible through scan only that forces only exact matches which will occur only in the three cases enumerated above (GCP.PIFU.IFBC.ASSSCANONLYNEW(7)). Writing this SCOM register sets latches in the IFU only (may also be set via scan): GCP.PIFU.IFBT.IABR.L2(0:63)	
0:61	Word address to be compared
62	Breakpoint enabled. (Address match causes trigger to debug logic)
63	Translation Enabled. An IABR match is signaled only if MSR[IR] matches this bit.

11.6.8.8 Service Processor Special Attention Register

Table 11-24. Service Processor Special Attention Register / And Mask / Or Mask)

Address:	x022001 / x022100 / x022200
Type:	RW / WO (and) / WO (or)
Reset:	Reset to all ‘0’s during Power-On Reset
Bit	Description
Any bit set in this register will drive Special Attention to the service processor. It may be used by the processor core to implement a mail-box function to communicate to the service processor. **Note:** This is an accumulation register and each source must be cleared to deassert the the Special Attention. The AND mask should be used to clear bits in the register and the OR-mask should be used by software to set new bits in order to prevent an attention from another source from being missed.	
0	Core to service processor special attention request
1	SP-ATTN instruction – ISU quiesced for maintenance due to software breakpoint
2	Debug DABR ‘softstop" (or ISU CIABR) caused core quiesce
3	Core hang detected – Core RAS help request to service processor – see Core RAS Status register • Moded by Core RAS Mode register[41] and Core RAS Hang Recovery Mode register[55,62] • Core dispatch and completion is stopped while this bit is set
4	RAS trigger caused core quiesce
5	Core FIR induced Special ATTN
6	Timeout on quiesce request (or pending change needing a quiesce) – two hang pulses and still pending
7:9	Software/operating system programmable
10:63	Not implemented

11.6.8.9 Hardware Implementation Dependent Register 0 (HID0)

Table 11-25. Hardware Implementation Dependent Register 0 (HID0)

Address:	x023101
Type:	RW
Reset:	Reset to all ‘0’s during Power-On Reset
Bit	Description
0:63	See *Table 2-7 HID0 Bit Functions* on page 46 for bit definitions.

11.6.8.10 Hardware Implementation Dependent Register 1 (HID1)

Table 11-26. Hardware Implementation Dependent Register 1 (HID1)

Address:	x023201
Type:	RW
Reset:	Reset to all ‘0’s during Power-On Reset
Bit	Description
0:63	See *Table 2-8 HID1 Bit Functions* on page 47 for bit definitions.

11.6.8.11 Core Fault Isolation Register / And-Mask / Or-Mask

Table 11-27. Core Fault Isolation Register / And-Mask / Or-Mask

Address:	x030001 / x031000 / x032000
Type:	RW / WO (and) / WO (or)
Reset:	Reset to all ‘0’s during power-on reset
Bit	Description
0	I-cache parity error
1:2	I-cache tag parity error
3	I-ERAT parity error
4	IFU fetched an Uncorrected Error (UE) from L2 cache
5	IFU detected a checkstop condition
6	D-cache parity error
7	D-cache tag parity error
8	D-ERAT parity error
9	LSU TLB parity error
10	LSU SLB parity error
11	LSU fetched an L2 UE
12	Load/Store unit detected checkstop condition
13	FP Unit 0 OR Unit 1 detected a checkstop condition
14	Checkstop on Trigger (debug only)
15	IDU detected a checkstop condition
16	ISU detected a checkstop condition
17	Reserved
18	FXU Unit 0 OR Unit 1 detected a checkstop condition
Note: For Processor Hang detects, reference Core RAS Status SCOM register x‘021200’ bits [26:31] to determine extent of hang recovery that was required. In addition, in the case of a Processor Hang with either an external or inknown source, the specific LSU and IFU status is available in the Core RAS Status SCOM register (x‘021200’ bits [59:63]). Write to Core Ras Control register SCOM x‘021001’ bit [55] and bit [41] respectively when clearing these Processor Hang FIR bits in order to clear the recovery and status information (if the hang was successfully recovered) for future diagnostics. Check Core RAS Mode register (SCOM x‘021100’ bits [61:63]) to see what types of hang recovery are enabled (these settings affect how the XStop Enable mask should be set for these bits).	
19	Processor Hang Detected with unknown Source (could be either internal or external)
20	Processor Hang Detected due to Internal Source
21	Processor Hang Detected due to External Source
22	Processor Hung beyond repair
23	Machine check and MSR[ME] = ‘0’
24	Core SPR bus has violated protocol OR SCOMC arbiter error
25	SCOMC abend
26	Invalid software access to IMC
27	Invalid SCOMC address
28	Reserved

Table 11-27. Core Fault Isolation Register / And-Mask / Or-Mask

Address:	x030001 / x031000 / x032000
Type:	RW / WO (and) / WO (or)
Reset:	Reset to all '0's during power-on reset
Bit	Description
29	Service processor attempted an invalid debug throttle setting.
30	Reserved
31	CIU detected checkstop condition
32:63	Not implemented

11.6.8.12 Core Fault Isolation Mask Register / And-Mask / Or-Mask

Table 11-28. Core Fault Isolation MASK Register / And-Mask / Or-Mask

Address:	x030400 / x031401 / x032401
Type:	RW / WO (and) / WO (or)
Reset:	Reset to all '1's during power-on reset
Bit	Description
0:31	Mask ('1' means FIR bit masked OFF)
32:63	Not implemented

11.6.8.13 Core Checkstop Enable Register

Table 11-29. Core Checkstop Enable Register

Address:	x030800
Type:	RW
Reset:	Reset to all '0's during power-on reset
Bit	Description
0:31	Causes checkstop to occur when corresponding FIR bit is set.
32:63	Not implemented

11.6.8.14 Core Machine Check Enable Register

Table 11-30. Core Machine Check Enable Register

Address:	x030901
Type:	RW
Reset:	Reset to all '0's during power-on reset
Bit	Description
0:31	Causes machine check to occur when corresponding FIR bit is set.
32:63	Not implemented

11.6.8.15 L2 Fault Isolation Register / And-Mask / Or-Mask / Checkstop Enable

Table 11-31. L2 Fault Isolation Register / And-Mask / Or-Mask / Checkstop

FIR Address:	x040000 / x041001 / x042001		Xstop Enable Address: 040801	
Error Mask Address:	x040401 / x041400 / x042400			
Type:	RW (FIR) / WO (and) / WO (or) / WR (Xstop)			
Reset:	Reset to all '0's during power-on reset		Default Mask Settings	
Bit	Description	Action	Error Mask	Checkstop Enable
0:41	Not implemented		0	0
42	L2 Cache CE	L2 correctable data error on processor request (I or D) • Data is corrected before forwarding to the processor • Corrected data is written back to the L2 L2 correctable data error on castout (flush, RWITM, or result of a replacement) • Data is corrected before forwarding to the BUS unit • (Threshold = 32)	0	0
43	L2 Cache UE	Uncorrectable error (UE) detected in L2 in response to processor load request. • UE response is sent to requesting core Store hits in the L2 line which contains a UE • Store is merged into line (timing does not permit discardin the store) write AUE ECC to distinguish between passed error from other source and local UE that has been altered Uncorrectable error (UE) detected in L2 in response to a cast out operation • Special UE (SUE) sent to memory	0	1
44	L2 Cache Special UE	Same as cache UE (PASSED ERROR)	1	0
45	L2 Directory parity error	Failing directory is refreshed with content of the other directory (Threshold = 32)	0	0
46	L2 Directory checkstop	L2 Directory parity error detected on both halfs of the directory. System Checkstop	0	1
47	L2 hang detect	No response to memory read request. The same action is take regardless of whether some other component acknowledged the request and then never returned the data, or no component on the fabric acknowledged the request. System checkstop	0	1
48	L2 store queue parity error	Any of the four store queues have a parity error. System Checkstop	0	1
49	RC machines data timeout error	RC machine timeout error Internal control error. System checkstop.	0	1
50	Non-cachable control or RC machine data tag error	ncctl or rcfsm0-3 detected valid data tag match when they were not expecting data Internal control error. System checkstop.	0	1
51	Store queue store error	A store command was interleaved between the 1st and 2nd of a **stqw** (stqw_seq_err) or detected a store queue request when the 4-entry store queue was full (stq_overflow_err).	0	1

Table 11-31. L2 Fault Isolation Register / And-Mask / Or-Mask / Checkstop

FIR Address:	x040000 / x041001 / x042001		Xstop Enable Address: 040801	
Error Mask Address:	x040401 / x041400 / x042400			
Type:	RW (FIR) / WO (and) / WO (or) / WR (Xstop)			
Reset:	Reset to all ‘0’s during power-on reset		Default Mask Settings	
Bit	Description	Action	Error Mask	Checkstop Enable
52	Quadrant0 CE threshold	Multiple cache CEs have been detected during a hang pulse duration.	0	0
53	Quadrant1 CE threshold	Multiple cache CEs have been detected during a hang pulse duration.	0	0
54	Quadrant2 CE threshold	Multiple cache CEs have been detected during a hang pulse duration.	0	0
55	Quadrant3 CE Threshold	Multiple cache CEs have been detected during a hang pulse duration.	0	0
56	Multiple cache CEs	Multiple cache CEs have been detected during a hang pulse duration from more that one quadrant.	0	0
57	Multiple Directory Parity errors	Multiple directory parity errors were detected within a period of two hang pulses. This indicates that the array error is not an intermittent fault. Since the STS cannot make forward progress with a stuck fault in the directory the system will be checkstopped.	0	1
58	llegal CRESP seen in RCFSM	RC machine detected a illegal cresp operation	0	1
59	Invalid MERSI detected	RC Machine detected or wrote an invalid MERSI state in the directories.	0	1
60	NCU Snoop PAAM Error Checkstop	NCU snooper detects another **icbi**/**tlbi** request while the previous request is still active. For example, cresp has not returned yet. The illegal PAAM request will be lost since the snooper does not register it and hence NO snoop response	0	1
61	NCU snoop Cresp Error Checkstop	When NCU snooper detects illegal values on cresp(0:3) “0000”=good; “xxx1”=retry; other=illegal. Errors are registered and snooper will hang waiting for good or retry response. Secondary PAAM error may occur as a result. A configuration switch cfg_snp_cresp_ckstp_en_b is provided to turn error detection off. In this case, the snooper will hang and the result is undermined	0	1
62	NCU CRESP Illegal Error	NCU receives from the BUS a cresp_valid with a cresp(0:3) value that is illegal based on the transaction sent.	0	1
63	NCU CRESP Timeout Error	Any of the state machines sourcing a transaction (store, load, or uop) to the bus times out while waiting for a valid and good cresp. Timeout occurs upon the second hang pulse seen while waiting for the good cresp.	0	1

11.6.8.16 STS Mode Register

Table 11-32. STS Mode Register

Address:	x043000
Type:	RW
Reset:	Reset to all ‘0’s during Power-On Reset
Bit	Description
0:47	Not implemented.
48	When set to ‘1’, it puts the L2 cache into direct mapped mode such that the receipient selection is based on a straight decode of address bit [42:44].
49	Disables shared last state in C2
50	Disables viewing of any ECC check error detection.
51	Forces intervention bit off.
52	Forces all E directory states to S. 0 Default allows E state
53	Reserved.
54	Enables livelock warning signal sent to cerrs where it can cause checkstop.
55	Reserved.
56	NCU store gather control disable.
57	Force g = ‘0’ st to go out to bus in order; hence, one has to complete with no retry before the other can go out.
58:60	Set the value for the time-out counter for the gather logic. The time-out count is made programmable for gather fine tuning. By setting these bits to ‘0’s, it in effect discourages gathering, though not totally disabling it. Its value is (^bit0 and ^bit1 and bit2 and 0 and 0). The timeout is set as default ‘11000’ pclk count when the latches are set to "000". Its range is (0-56 pclks, step by 4).
61	Turn off gather timer. No time out for gather.
62:63	Spare.

11.6.8.17 Processor Interconnect Mode Register

Table 11-33. Processor Interconnect Mode Register

Address:	x046A00 (PI inputs - PII) x0F7A00 (PI outputs - PIO)
Type:	RW
Reset:	Reset to '0's by Power-On Reset
Bit	Description
0:31	Not implemented.
32	WIAP (PIO, starts writing the IAP pattern on the output side)
33	PRECOMP_EN (PIO); enable = '1'
34	REFFORWARD (PII) 0: ½ OV_{DD} 1: average of CLKIN/ $\overline{\text{CLKIN\_B}}$
35	HIGH_SKEW (PII)
36	RIAP (PII, starts reading the IAP pattern on the input side)
37	LO_OHM_MODE (PIO) 0: 40 ohms 1: 20 ohms
38	Not implemented
39	FIAP(PII) (not supported)
40:41	TARGET_TIME(0:1) (PII)
42	RESET (PII, controls the IAP state machine and associated logic)
43	BYPASS (PII, there is no differential clock)
44:48	WIND(0:4) (PIO, PII, this is the windage for the clock) ; bit [0]=sign bit 00000 Clock is centered
49:52	VREF_WIND(0:3) (PII); bit[0] = sign bit; refers to REFFORWARD (bit [34]) 0000: Centerpoint
53:55	DATA_WIND (PII, windage for data increments); adds delay
57	ESTMODE (PII,PIO) 0: PI controls are for IAP (default) 1: PI controls are for EST (not supported)
58	ESTONE (PII,PIO) 0: Drive and expect all zeroes, walk a one across the interface when ESTMODE = '1' (default) 1: Drive and expect all ones, walk a zero across the interface when ESTMODE = '1' (not supported)
59	RDTMODE(PII,PIO) 0: PI controls are for IAP or EST (default) 1: PI controls are for Random Data Test (RDT, also known as IOBIST) and RDTPASS (not supported)
60:63	Not implemented.

11.6.8.18 Processor Interconnect Status Register

Table 11-34. Processor Interconnect Status Register

Address:	x046B01 (PI inputs PII)
Type:	RO
Reset:	Reset to ‘0’s by Power-On Reset
Bit	Description
0:35	Not implemented
36	Reduced IAP (RIAP) done (0b1 = done; clears when RIAP is dropped); status only
37	RESTPASS (valid when ESTMODE = ‘1’); status only
38	Clock period too large (greater than delay line); critical
39	No first valid ‘10’ found; fatal
40	No first valid ‘01’ found; critical
41	Data bit deskewed to maximum; warning
42	No valid flag – 0; fatal
43	No valid flag – 1; critical
44	No valid flag – 2; critical
45	Final values fail; fatal
46	Final delay value is negative; warning
47	Final delay value is too large; critical
48:55	Calculated clock period (8 bits; in delay elements); status only
56:63	Inserted I/O clock delay (8 bits; in delay elements); status only
Status indication key: Warning – Indicates a possible system design problem. Critical – Reduced IAP was performed, meaning interface sampling point is suboptimal. Fatal – Interface cannot perform even a reduced IAP	

11.6.8.19 970FX Bus Fault Isolation Register / And Mask / Or Mask

Table 11-35. 970FX Bus Fault Isolation Register / And Mask / Or Mask

Address:	x0A0001 / x0A1000 / x0A2000			
Type:	RW (FIR)/ WO (and) / WO (or)			
Reset:	Flushed to all ‘0’s during power-on reset			
Bit	Description	Action	Error Mask	Checkstop Enable
0:31	Not implemented	–	–	–
32:44	Spare	–	–	–
45	Parity error for input from NCU	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex	0	1
46	Parity error for input from L2	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex	0	1

Table 11-35. 970FX Bus Fault Isolation Register / And Mask / Or Mask

Address:	x0A0001 / x0A1000 / x0A2000			
Type:	RW (FIR)/ WO (and) / WO (or)			
Reset:	Flushed to all '0's during power-on reset			
Bit	Description	Action	Error Mask	Checkstop Enable
47	Parity error for input from encoded data (both bits [47, 52] will be set for non-encoded errors)	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex.	0	1
48	BUS slave report parity error via transaction response	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex.	0	1
49	Transaction response return null	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex.	0	1
50	Bus slave returns transaction response retry on a data packet	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex.	0	1
51	Bus slave returns intervention without share or mod or returns shared, modified, or intervention on castout	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex.	0	1
52	Error on bus input data in non-encoded mode	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex	0	1
53	Command packet is odd number of beats	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex.	0	1
54	Data header packet is odd number of beats	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex.	0	1
55	Data packet is odd number of beats	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex.	0	1
56	Header packet appears inside data packet	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex.	0	1
57	Cresp PAAM Cresp PAAM window violationwindow violation	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex.	0	1
58	Spare	–	–	–
59:63	Tag associated with transaction response in reported errors	Processor execution is halted. System checkstop is raised which will halt the rest of the processors in the complex.	0	1

11.6.8.20 970FX Bus Error Mask / And Mask / Or Mask

Table 11-36. 970FX Bus Error Mask / And Mask / Or Mask

Address:	x0A0400
Type:	RW / WO (and) / WO (or)
Reset:	Flushed to '1's during power-on reset
Bit	Description
0:44	Not Implemented
45:63	Mask (1 = FIR bit masked OFF)

11.6.8.21 Checkstop Enable Register

Table 11-37. Checkstop Enable Register

Address:	x0A0800
Type:	RW
Reset:	Flushed to ‘0’s during Power-On Reset
Bit	Description
0:44	Not implemented
45:63	Mask (‘1’ = checkstop)

11.6.8.22 Bus Configuration Register

Table 11-38. Processor Configurable Timing Delay Parameter Register (BUSCONF)

Address:	x0A8000	
Type:	RW	
Reset:	Reset to all '0's during power-on reset	
Bits	Bit Name	Description
0:31	–	Reserved
32:35	COMPACE	Command Pipeline Delay
36:40	STATLAT	Transfer Handshake Response Latency
41:44	SNOOPLAT	Processor Snoop Latency
45:48	SNOOPACC	Snoop Accumulation delay
49	APSEL	Bus encode disable. ('1' = disable)
50	bcfg_en_rww_reg	Enable commands during data writes. ('0' = enable)
51:52	bcfg_align_cmd	Align command 00 Command out on even or odd 01 Command out on even 10 Command out on odd
53	bcfg_cp_aresp_ena	Disables wait for cresp for castouts and pushes. ('1' = disable)
54	bcfg_en_cmpc_tms_8	Enables a longer wait period before "back-off"
55	bcg_en_cmpc_ajst	Enables bus "back-off" of sending command
56	grs_bfb_mode_reg(24)	Sets bus mode to no encode with single error correct some double error detect
57	grs_bfb_mode_reg(25)	Set bus mode to no encode with single error correct and double error detect
58	grs_bfb_mode_reg(26)	Power tuning facility disable
59:63	–	Reserved

11.6.8.23 Bus Status Register

Table 11-39. 970FX Bus Status Register

Address:	x0A9000
Type:	RW
Reset:	Reset to all ‘0’s during power-on reset
Bit	Description
0:53	Not implemented
54	RC is active Op. (bfb_status_decode[0:3] = ‘0000’ or ‘1000’)
55	Co is active Op. (bfb_status_decode[0:3] = ‘0001’ or ‘1001’)
56	PSH is active Op. (bfb_status_decode[0:3] = ‘ 0010’ or ‘1010’)
57	NCUST is active OP. (bfb_status_decode[0:3] = ‘0011’ or ‘1011’)
58	NCULD is active OP. (bfb_status_decode[0:3] = ‘0100’ or ‘1100’)
59	INT is active OP. (bfb_status_decode[0:3] = ‘0101’ or ‘1101’)
60	Snoop CMD is active Op. (bfb_status_decode[0:3] = ‘0110’ or ‘1110’)
61	Read Data is active Op. (bfb_status_decode[0:3] = ‘0111’ or (‘1000 ‘to ‘1110’))
62	STS core idle
63	STS snoop idle

11.6.8.24 Power-On Reset Status Register

Table 11-40. Power-On Reset Status Register

Address:	x400000
Type:	RO / WO (clear entry marked with *)
Reset:	Reset to inactive by Power-On Reset 0x000000000000000
Bit	Description
0:1	Not implemented
2	* **Status:** Active after fuse copy completion
3	* **Status:** Active after DABISTINITL1 completion
4	Not implemented
5	* **Status:** Active after SCABISTINIT completion
6	Not implemented
7	* **Status:** Active after DABISTINITL2 completion
8:15	Not implemented
16	* **Status:** Active after phase synchronization completion
17	**Status:** Active while WIAP is driven by power-on reset
18	* **Status:** Active after successful PI synchronization
19	* **Status:** Active after INITCORE completion
20:23	Not implemented

Table 11-40. Power-On Reset Status Register (Continued)

Address:	x400000
Type:	RO / WO (clear entry marked with *)
Reset:	Reset to inactive by Power-On Reset 0x000000000000000
Bit	Description
24:28	**Status:** Current POR program counter address
29	*** Status:** Active high
30	*** Status:** Active if continue was received while not in the WAIT state (active high)
31	**Status:** Active after an I2CGO command until the next CONT command
32	**Status:** Active when in POR state machine is in the WAIT state
33	**Status:** Active when in POR state machine is in the WAITDBG state
34	**Status:** Active when in POR state machine has reached the last instruction.
35:63	Not implemented

11.6.8.25 Power-On Reset Continue Register

Table 11-41. Power-On Reset Continue Register

Address:	x400101
Type:	WO
Reset:	Reset to inactive by Power-On Reset
Bit	Description
0	**Command bit:** Send continue command to the POR state machine. This command also invalidates the I2CGO pin next time the JTAG state machine enter the test-mode-reset or run-idle state.
1:63	Not implemented.

11.6.8.26 Power-On Reset I^{2}C/JTAG Arbitration Register

Table 11-42. Power-On Reset I^{2}C/JTAG Arbitration Register

Address:	x400201
Type:	WO
Reset:	Reset to inactive by power-on reset
Bit	Description
0	Command Bit: Assert and hold the I2CGO pin when the JTAG state machine enters the Test-Mode-Reset or Run-Idle state.
1:63	Not implemented.

11.6.8.27 Power Management Control Register

Table 11-43. Power Management Control Register

Address:	x400801
Type:	Bits 0:3, 24:26(RW) Bits 4:22 (RO)
Reset:	Reset to all ‘0’s by power-on reset
Bit	Description
0	Real doze mode enable **Note:** Must be set to ‘1’ during power-on reset to enable doze.
1	Advance on QACK drop timeout enable
2	Reset power management control state
3	Pulse degate pending
Power management control status bits	
4:7	State latches
8	Nap_ready received
9	Doze_ready received
10	Biu_snoop_idle received
11	QACK received
12	Interrupt_pending received
13	Clock_ramp_done received
14	Doze_ramp_done received
15	QREG sent
16	Stop_req_nap sent
17	Stop_req_doze sent
18	Degate_pending sent
Power management control error bits	
19	State machine error (undefined state)
20	BIU Snoop Idle dropped while qack active
21	Timeout error on clock ramp done
22	Timeout error on qack not dropped
Miscellaneous	
23	Inhibit power save frequency in NAP mode
24	Select PMC Tracing
25	Force time base enable
26	Force local clock backup mode for repsync and time base counter
27:63	Not implemented

11.6.8.28 Power-on Reset Sequence Register 0

Table 11-44. Power-On Reset Sequence Register 0

Address:	x401400
Type:	WO
Reset:	Initialized by power-on reset.
Bit	Description
0:4	**Data:** POR Instruction #0 (first instruction)
5:54	Not implemented
55:59	**Data:** POR Instruction #11
60:63	Not implemented

11.6.8.29 Power-On Reset Sequence Register 1

Table 11-45. Power-On Reset Sequence Register 1

Address:	x402400
Type:	WO
Reset:	Initialized by power-on reset.
Bit	Description
0:4	**Data:** POR Instruction #12
5:54	Not implemented
55:59	**Data:** POR Instruction #23
60:63	Not implemented

11.6.8.30 Power-On Reset Sequence Register 2

Table 11-46. Power-On Reset Sequence Register 2

Address:	x404400
Type:	WO
Reset:	Initialized by power-on reset.
Bit	Description
0:4	**Data:** POR instruction #24
5:34	Not implemented
35:39	**Data:** POR instruction #31 (last instruction)
40:63	Not implemented

11.6.8.31 Power Tuning Status Register

Table 11-47. Power Tuning Status Register

Address:	x408001
Type:	RW: 2-3, 7-8,16-19 WO: 15
Reset:	Initialized by power-on reset to 0000000 00000000
Bit	Description
0:63	See *Table 9-7 Power Status Register (PSR).*

11.6.8.32 Global Fault Isolation for Checkstop Conditions Register

Table 11-48. Global Fault Isolation for Checkstop Conditions (Global FIR)

Address:	x500001
Type:	RO
Reset:	Reset to all ‘0’s by power-on reset
Bit	Description
0:31	Not implemented.
32	Processor Core
33	L2
34	BIU
35	Power management control
36	Synching error in TC_CCINTF
37	Checkstop pin
38	Radiation detection error
39	Checkstop on trigger
40:63	Not implemented

11.6.8.33 Error Enable Mask

Table 11-49. Error Enable Mask

Address:	x500400
Type:	RW
Reset:	Reset to all '0's by power-on reset
Bit	Description
0:31	Not implemented
32	Physical Core
33	L2
34	BIU
35	Power management control
36	Synchronization error in TC_CCINTF
37	Checkstop pin
38:63	Not implemented

11.6.8.34 Mode Register for Fault Isolation Registers

Table 11-50. Mode Register for Fault Isolation Registers

Address:	x500601
Type:	RW
Reset:	Reset to all '0's by Power-On Reset
Bit	Description
0:31	Not implemented.
32	Hold recoverable fault isolation when error is detected. Behaves like the checkstop FIR.
33	Checkstop indications from checkstop pin or checkstop trigger will set the checktop attention.
34	Processor machine check will set recoverable attention.
35	Processor machine check will set special attention.
36	Processor machine check will set checkstop attention.
37	Global checkstop signal will signal checkstop to processor core.
38	Local checkstop signal will signal checkstop to processor core.
39:63	Not implemented.

11.6.8.35 Debug Mode Register

Table 11-51. Debug Mode Register

Address:	x500700
Type:	RW
Reset:	Reset to all ‘0’s by Power-On Reset
Bit	Description
0:31	Not implemented.
Note: When setting any of the bits [32:35], the core RAS mode bit [26] should also be set to prevent checkstop from altering core rings when checkstop holds fetch, dispatch, and completion before the clock stop has had time to take effect.	
32	Force phase aligned clockstop when global checkstop signal is active
33	Force phase aligned clockstop when local checkstop signal is active
34	Force immediate clockstop when global checkstop signal is active
35	Force immediate clockstop when local checkstop signal is active
36:37	Not implemented
38	Core Special Attention will cause checkstop
39	Increase hang pulse rate by 100x
40:47	Not implemented.
48	Disable STS hang pulses
49:51	Not implemented
52	Disable core hang pulses
53	Disable core and STS hang pulses when recoverable attention is set
54	Disable core and STS hang pulses when special attention is set
55	Disable core and STS hang pulses when checkstop attention is set
56	Disable early hang pulse to core
57:60	Not implemented
61	Disable early hang pulses when recoverable attention is set
62	Disable early hang pulses when special attention is set
63	Disable early hang pulses when checkstop attention is set

11.6.8.36 Hang Pulse Generation Register

Table 11-52. Hang Pulse Generation

Address:	x503001
Type:	RW
Reset:	Reset to all ‘0’s by power-on reset
Bit	Description
0:31	Not implemented
32:39	STS Hang Mask x‘01’ - Hang Pulse generated every *Counter x 2* x‘02’ - Hang Pulse generated every *Counter x 4* x‘04’ - Hang Pulse generated every *Counter x 8* x‘08’ - Hang Pulse generated every *Counter x 16* x‘10’ - Hang Pulse generated every *Counter x 32* x‘20’ - Hang Pulse generated every *Counter x 64* x‘40’ - Hang Pulse generated every *Counter x 128* x‘80’ - Hang Pulse generated every *Counter x 256*
40:47	Core Hang Mask x‘01’ - Hang Pulse generated every *Counter x 2* x‘02’ - Hang Pulse generated every *Counter x 4* x‘04’ - Hang Pulse generated every *Counter x 8* x‘08’ - Hang Pulse generated every *Counter x 16* x‘10’ - Hang Pulse generated every *Counter x 32* x‘20’ - Hang Pulse generated every *Counter x 64* x‘40’ - Hang Pulse generated every *Counter x 128* x‘80’ - Hang Pulse generated every *Counter x 256*
48:51	Not implemented
52:63	Hang Counter limit mask (12-bit LFSR) x‘4FC’ = 100 Cycles x‘CF0’ = 200 Cycles x‘DAE’ = 300 Cycles x‘E89’ = 400 Cycles x‘D1A’ = 500 Cycles x‘0A3’ = 600 Cycles x‘F8C’ = 700 Cycles x‘FAA’ = 800 Cycles x‘8F3’ = 900 Cycles x‘6C8’ = 1000 Cycles x‘5D1’ = 2000 Cycles x‘668’ = 3000 Cycles x‘D3E’ = 4000 Cycles

11.6.8.37 Chip ID Register

Table 11-53. Chip ID Register

Address:	x504101
Type:	RW (Bit 35 is WO)
Reset:	Reset to '0's by power-on reset
Bit	Description
0:31	Not implemented
32:34	Processor ID (0:2)
35	Capture values from PID primary C4 inputs (Work Only)
36:63	Not implemented

11.6.8.38 SCOM Mode Register

Table 11-54. SCOM Mode Register

Address:	x600001
Type:	RW
Reset:	Set to '00000FFE 00000000' by Power-On Reset or SCRESET
Bit	Description
0	Enable state machine error checking
1	Enable data wrap test
2	Enable address wrap check
3	Enable valid address checking
4	Enable SCOM global hang checking
5:6	Not implemented
7	Enable modifier parity checking
8	Enable valid clock SCOM address checking
9	Enable STS SCOM ring (set to '0' before scanning STS)
10	Enable virtual core0 (enables SCOM ring and functional fences) (set to '0' before scanning core)
11	Enable chip trace array SCOM ring (set to '0' before scanning I/O)
12:17	Not implemented
18	Fence cores on checkstop **Note:** Core checkstop is controlled by Mode register for fault isolation registers bits[44:45] (SCOM x500601)
Note: Scan Communications Reset (SCRESET) resets all sequencers that communicate over the Scan Communications bus to an idle state. This terminates LBIST or ABIST if in progress, but will not halt BIST or functional clock sequences or clear the scan rings. If a BIST was running, software should also stop the system clocks and clear the scan rings. This instruction is intended to reset a "hung" sequence and re-establish JTAG control of the SCOM interface.	

Table 11-54. SCOM Mode Register

Address:	x600001
Type:	RW
Reset:	Set to ‘00000FFE 00000000’ by Power-On Reset or SCRESET
Bit	Description
19	Raise Core and STS fences on clock stop. **Note:** Clock stop is seen by free-running logic several cycles before the clocks are actually stopped. This bit should be set to ‘1’ by software by default, but must be set to ‘0’ before cycle stepping for debug.
20:31	SCOM hang detection delay match in processor clocks. 500: xD1A 1000: x6C8 2000: x5D1 3000: x668 4095: xFFE
32:63	Not implemented.
Note: Scan Communications Reset (SCRESET) resets all sequencers that communicate over the Scan Communications bus to an idle state. This terminates LBIST or ABIST if in progress, but will not halt BIST or functional clock sequences or clear the scan rings. If a BIST was running, software should also stop the system clocks and clear the scan rings. This instruction is intended to reset a “hung” sequence and re-establish JTAG control of the SCOM interface.	

11.6.8.39 SCOM Controller Error Register

Table 11-55. SCOM Controller Error Register

Address:	x600100
Type:	RO (Write zeros to clear after SCRESET)
Reset:	Reset to all ‘0’s by Power-On Reset
Bit	Description
0	Modifier parity error
1	Parity error in parallel SCOM state machine
2	Parity error in serial SCOM state machine
3	Parity error in main SCOM controller state machine
4	Parity error in SCOM arbiter state machine
5	Dataav and Useadr were both on
6	Bad address in SCOM ring
7	SCOM address was not recognized
8	SCOM address for clock command was invalid
9	SCOM collision
10	Not implemented.
11	SCOM hang (See Active Source Indicator (ASI) in [15:23] to see what is pending)
12:14	Reserved
15	(ASI) Waiting for the address to return
16	(ASI) Waiting for grant from arbiter

Table 11-55. SCOM Controller Error Register

Address:	x600100
Type:	RO (Write zeros to clear after SCRESET)
Reset:	Reset to all '0's by Power-On Reset
Bit	Description
17	(ASI) Waiting for read data to return
18	(ASI) Parallel to serial state machine waiting for Useaddr (SCOM Read) to drop
19	(ASI) Parallel to serial state machine waiting for Dataav (SCOM Write) to drop
20	(ASI) Parallel to parallel state machine waiting for Useaddr to drop
21	(ASI) Parallel to parallel state machine waiting for Dataav to drop
22	(ASI) Waiting for read shifter to empty
23	(ASI) Address was returned and not accepted
24:47	SCOM address that was pending at the time of the error
48:63	Not implemented

11.6.8.40 Clock Ratio Register

Table 11-56. Clock Ratio Register (N:1 phase hold control)

Address:	x600400
Type:	RW
Reset:	ratio: Reset to all '0's by power-on reset encode_iap: Reset to '1' by power-on reset count1us_factor: Reset to all '1's by power-on reset
Bit	Description
0	Not implemented
1:3	Bus clock ratio (for N:1 phase_hold generation) 000: 2:1 (supported) 001: 3:1 (supported) 010: 4:1 (not curently supported) 011: 6:1 (not curently supported) 100: 8:1 (not curently supported) 101: 12:1 (not curently supported) 110: 16:1 (not curently supported) 111: 1:1
4	Capture values from BUS_CFG primary C4 inputs (WO)
5:7	I^2C clock ratio – This counter is asynchronous to the mod48 counter independent of powertuning. 000: 1/16 the frequency of the core at full speed 001: Not supported 010: 1/8 the frequency of the core at full speed 011: Not supported 100: Not supported 101: Not supported 110: Not supported 111: 1:1 **Note:** This clock rate is independent of th epowertuning frequency.

Table 11-56. Clock Ratio Register (N:1 phase hold control)

Address:	x600400
Type:	RW
Reset:	ratio: Reset to all '0's by power-on reset encode_iap: Reset to '1' by power-on reset count1us_factor: Reset to all '1's by power-on reset
Bit	Description
8	Encode IAP pattern
9:11	Reserved
12:19	Reset value for µs-counter in I^{2}C macro.
20:63	Not implemented

11.6.8.41 Clock Command Register

Table 11-57. Clock Command Register

Address:	x800000
Type:	RW
Reset:	Set to '0000000 00000000' by Power -On Reset
Bit	Description
0:10	Unused
11	SYNC clock sekect: must be programmed to logic level zero ('0')
12:15	Clock command (bits[14:15] are not implemented) x4: Pulse selected clocks x8: Start selected clocks xC: Stop selected clocks
Domain Select	
16	Core
17:18	Unused
19	STS
20	I/O
Array Clock Commands (applies for start, pulse, and stop)	
21	Apply command to RAM C2 (not valid for I/O)
22	Apply command to C2star clocks
23	Reserved
Latch Clock Commands (applies for start or pulse, and stop)	
24	Apply command to C1
Free Running Clock Section	
25	Domain select: Chipras
26:63	Not implemented

11.6.8.42 Status Register

Table 11-58. Status Register

Address:	x800003
Type:	RW 2, 4, 7-9, 31. RO 20-21, 23-24
Reset:	Set to ‘0000040 00000000’ by Power -On Reset
Bit	Description
0:1	Unused.
2	Invalid Read/Write Address Error. An attempt was made to read or write to a clock interface address that does not exist. Raises SCATTN unless blocked.
3	Unused.
4	JTAG/BIST Collision Error. JTAG attempted to initiate a write to a valid register while Event Processor or ABIST was in progress. Raises SCATTN unless blocked.
5:6	Unused.
7	Register Write Error. A write to options register with address x800009 while the core0, core1, or STS clocks are not all off; or a write to miscellaneous register with address x80000F while I/O clocks are running, will raises SCATTN unless blocked.
8	PLL_LOCK state. If ‘1’, PLL unlock detected. Raises SCATTN unless blocked.
9	Bad psync detected. The ccintf mod48 counter does not match the tpt mod48 counter.
10:19	Unused.
20:27	Clock state bits, CLK_STATE(0:7) These correspond one to one with the domains defined in the x‘800000’ register bits 16 to 20. A value of 1 indicates the clocks (c1) are running. These bits are not writeable. Bit [20] - Core Bit [21] - Free running Bit [22] - Unused Bit [23] - STS Bit [24] - I/O Bits [25:27] - Unused
28:30	Unused.
31	tcl_tc_ortho_clk_chk: Indicates that an attempt was made to scan a domain while the clocks were not stopped.
32:63	Not implemented.

11.6.8.43 Phase Synchronization Control Register

Table 11-59. Phase Synchronization Control Register

Address:	x800006
Type:	RW: 0-2,16-17; WO: 22:27
Reset:	Set to ‘00000000 00000000’ by Power -On Reset
Bit	Description
0	Mode bit: Allow Resync: If active will force resynchronization if an asynchronious psync is detected. **Note:** Debug use only.
1	Command bit: Run Phase-Synchronization.
2	Configuration bit: Do not Checkstop on bad_sync.
3:15	Not implemented.
16	Mask Clock Freeze: When equal to ‘1’, the clock-freeze signal from ts_glob is ignored.
17	Enable Blank Clock Command: Used in conjunction with x‘80_00_00’ Pulse Clock Command causing BLANKING of one system clock for chip clocks selected by the domain selects in the command to the x‘80_00_00’ register.
18:21	Not implemented.
22:27	Psync initial load counter value (initial value: 0x0F)
28:63	Not implemented.

11.6.8.44 Clock Command Control Register

Table 11-60. Clock Command Control Register

Address:	x800009
Type:	RW, writing this register while core or STS is running will raise an attention
Reset:	Set to ‘0000000 0314001A’ by power-on reset
Bit	Description
0:27	Event Register (Four Fields of Five Control Bits per Field) 21 - 27 First Event Field 14 - 20 Second Event Field 7 - 13 Third Event Field 0 - 6 Fourth Event Field Field Bit Definitions Bit 0/7/14/21: Run SYSTEM C1 Clock Bit 1/8/15/22: Run SCAN SC1 Clock Bit 2/9/16/23: Run RAM_C2 Clock Bit 3/10/17/24: Run C2STAR Clock Bit 4/11/18/25: Ignored Bit 5/12/19/26: Enable Clock-rate Divide Counter (slows down the application of clock events). Bit 6/13/20/27: Enable Generic Counter (specifying the number of times to loop on event field). **Notes:** 1. System C2 clock is always running in functional mode. 2. Domain selects (bit [38:42]) only control the scope of the C1, RAM_C2, C2STAR. 3. Bit [52] initiates this sequence, which performs the events in order. Bits[55:57] control looping on these events. SCOM x‘84000B’ must be used to configure the TEST LENGTH count. 4. If SCAN SC1 is the **only** clock selected for an event, then the CHANNEL LENGTH count is used for that event. SCOM x‘840008’ must be used to configure the CHANNEL LENGTH count. If SC1 is selected in addition to either a SYSTEM or RAM clock, then only a single clock is given.
28:31	Generic Counter (0:3). Programmed Count Value: Down Counter
32:35	Lower-order Phase Hold Counter (for event processing.) Programmed Count Value: Up Counter - 0 to 11. **Note:** May not be used while the PSYNC POR instruction is active.
36:37	Upper-order Phase Hold Counter (for event processing). Programmed Count Value: Up Counter – 0 to 3. **Notes:** 1. May not be used while the PSYNC POR instruction is active. 2. This counter counts the carry out of the lower-order phase hold counter (for event processing).
38	Cmd_reg(0): CORE select.
39	Cmd_reg(2): STS select.
40	Cmd_reg(3): I/O select.
41	Cmd_reg(4) Free running select.
42	Cmd_reg(5): Domain select for hard stop: Free running domain.
43	Cmd_reg(6) Global Array Inhibit: This signal gets ORed with ABIST_EN and Local_Latch OR ESP_RAMSCAN and NOT Local_Lt OR LSSD/GSD Scan-Active. If the scan 0 has not occurred through the scan-abist section the arrays will be enabled functionally when the Global Array Inhibit is inactive. Set by power-on reset. Need to clear by writing.
44	Cmd_reg(7) SCAN 0 command (similar to the FLUSH 0 Access Command). Scan in 0 to all selected domains specified by Cmd_reg(0:3) except FUSE and NOT_BIST (timing chain) rings.

Table 11-60. Clock Command Control Register

Address:	x800009
Type:	RW, writing this register while core or STS is running will raise an attention
Reset:	Set to '0000000 0314001A' by power-on reset
Bit	Description
45	Cmd_reg(8) FULL SCAN 0 Command. Used in conjunction with the SCAN 0 Command, causes the FUSE and NOT_BIST(timing chain) to also scan to 0. Power-on reset sets it to 1 (clean notbist chain).
46	Cmd_reg(9) LBIST. Set for running LBIST, will cause LBIST ring-selects and MISR connections to be established and LBIST logic to be fenced out in the free running domain. No effect on EPS engine. Bit [11] in x'80000F' will automatically be set to fence then I^{2}C (not reset when resetting LBIST bit).
47	Cmd_reg(10) CAM BIST. Set for testing CAMs with scanned ABIST.
48	Cmd_reg(11) CENTRAL SCANNED ABIST CORE. Set to test core arrays covered by scanned ABIST.
49	Cmd_reg(12) CENTRAL SCANNED ABIST NOT CORE. Set to test non-core arrays covered by scanned ABIST.
50	Cmd_reg(13) Decentral ABIST. Set to test arrays covered by dedicated ABIST engines: L2 Cache, L2 Directory, L2 LRU, IFU Cache, IFU Directory, IFU BHT, L1 D-cache
51	Cmd_reg(14) Decentral ABIST RAM_SELECT. 0 for first half of arrays, 1 for second half.
52	Cmd_reg(15) Event Processing Command. Initiates a clock event process.
53	Cmd_reg(16) Phase Load Control(0). Load the phase counter with programmed value at Event Processor State 2.
54	Cmd_reg(17) Phase Load Control(1). Load the phase counter with programmed value with Event Complete. Normally, for LBIST this bit would be set. By not programming this bit the capability to change the RUN_CLOCK pulse to PHASE_HOLD allignment is achieved.
55:57	Cmd_reg(18:20) Runn_Command Control. Causes the Event Processor to loop back to different event fields. This provides the capability of an initialization setup. '000' loop on all events (branch is back to first event) '100' loop on fourth event '101' branch back to second event '110' branch back to third event
58	Cmd_reg(21) Event Processing Override. Provides the means to interrupt an event process that is in progress.
59:61	Cmd_reg(22:25) Domain Select for Hard Stop. bit [22]: CORE 0 ***ACTIVE High*** bit [23]: Reserved bit [24]: STS ***ACTIVE High*** bit [25]: I/O ***ACTIVE High***
62	Cmd_reg(26) IO MESH CLOCK SELECT. The Processor Interconnect runs off an INPUT clock called the IO_CLOCK. For LBIST and Scanning of the I/O domain, this bit must first get set. Power-on reset sets it to '1'.
63	Cmd_reg(27) LBIST AC MODE. Cause 8:1 logic clocks off.

11.6.8.45 Energy Star Register

Table 11-61. Energy Star Register

Address:	x80000A
Type:	RW
Reset:	Set to ‘xx000x20 00000000’ by power-on reset. Value marked as x are read only. At power-on reset these are initialized to ‘0’.
Bit	Description
0:5	Clock Rate Divide Counter (0 to 5) Programmed Count Value: Down Counter. **Note:** Moved from Clock Control Register to here due to space limitation.
6	Enable Trace Arrays / Debug Logic On. Initial value is ‘0’.
7	Fuse clock control (rcha_reg(14)). Initial value is ‘0’.
8:17	Not implemented.
18:23	Shadow Counter *(Read-Only).* The master counter values at the time of the clock freeze on immediate stop are in the shadow counter. This shadow counter value can be read and then loaded into the master counter in x‘800009’ order to start the clocks using RUNN in the event processor.
24	Enter Energy Star Mode.
25	Exit Energy Star Mode.
26	**Config. Shadow stop enable.** Allow starting/pulsing clocks based on the saved value for the master phase hold counter. **Note:** In order for this mode to work, this bit must be written while both STS and I/O clocks are running. From the POR state, the following sequence must be programmed: 1. x‘80_00_00’ x‘0008_1800_0000_0000’ start BOTH STS AND I/O clocks. 2. x‘80_00_0A’ x‘0000_0020_0000_0000’ enable shadow stop. 3. x‘80_00_00’ x‘000C_1800_0000_0000’ stop EITHER STS OR I/O clocks. 4. x‘80_00_09’ x‘0000_0000_0F38_1002’ scan0. 5. x‘80_00_09’ x‘0000_0000_0000_0000’ clear array inhibit, receiver inhibit. 6. x‘80_00_00’x‘0004_D800_0000_0000’ pulse clocks (repeat as may be needed). 7. x‘80_00_00’ x‘0008_D800_0000_0000’ run clocks. Clocks resume at the next, subsequent phase_hold allignment after the last pulse clock.
27	UNCONDITIONAL IMMEDIATE EXIT from Energy Star Mode.
28	If ‘0’: will stop ABIST clocks in Core and VPU.
29	If ‘0’: will stop ABIST clocks in STS.
30	If ‘0’: will stop ABIST clocks in scanned ABIST machines and prevents scanning through the LCLK domain.
31:63	Not implemented.

11.6.8.46 Status Register Mask

Table 11-62. Status Register Mask

Address:	x80000C
Type:	RW
Reset:	Set to ‘0080000 00000000’ by power-on reset
Bit	Description
0:1	Unused
2	Block Invalid Read/Write Address Error (BLKRWAERR)
3	Unused
4	Block JTAG/BIST Collision Error (BLKJBCERR)
5:6	Unused
7	Block Options Register Write Error (BLKORWERR)
8	Block PLL Lock Error (BLKPLLERR)
9:30	Unused
31	Block clock orthoganility from generating attention
32:63	Not implemented

11.6.8.47 I/O Control Register

Table 11-63. I/O Control Register

Address:	x80000F
Type:	RW; writing this register while I/O is running will raise an attention
Reset:	Set to '2200000 00000000' by power-on reset
Bit	Description
Note: The tristate control bits(0:6) determine whether or not drivers are placed in a high-impedance state. A bit value of '1' places the appropriate drivers in the high-impedance state. These bits should be set prior to executing LBIST, and are flushed to all '1's by a power-on reset.	
0-1	Not implemented.
2	Drivers Inhibit.
3-5	Not implemented.
6	Receivers Inhibit.
7	Not implemented.
8	KILL_DYN_CLK DISABLE. This bit if asserted stops the kill_dyn_clk. For example, it asserts the kill_dyn_clk signal. This bit must be 0 at all times.
9	STOP_PIPE DISABLE. This bit disables the global "stop_pipe" signal, which is used to enabled the stop_ctl to dynamic logic. Stop_pipe is an asynchronous signal. This control turns on the clocks to the dynamic non-scan L1 latches in order to initialize them after SCAN 0 initialization. This bit must be 0 at all other times.
10	WrapIOCmd. Prevent Driver Inhibit during wrap IO test.
11	I^{2}C Inhibit. If set the I^{2}C outputs to the JTAG macro are fenced. This bit will be set when bit [46] (LBIST) in SCOM x'800009' is set (not reset when reseting LBIST bit).
12:63	Not implemented.

11.6.8.48 ABIST Status Register

Table 11-64. ABIST Status Register

Address:	x820004
Type:	RW, bits [21:26] are unspecified after a LBIST,SCAN0.SCAN and should be reinitialized before ABIST ABIST fail bits are only valid if the corresponding ABIST done bit is asserted. **Note:** Bits in this register are only updated during eps controlled operation. Turn ccintf_local_psav_dis if using ABIST with functional clocks.
Reset:	Set to '0000000 00000000' by power-on reset
Bit	Description
0	EPS Running or Decentral Abist in progress (RO).
1	Any EventComplete. (Cleared with running an event. Set and held after the completion of the event processing.) Cleared when bit [52] of x'800009' is zero.
2:19	Not implemented.
20	Physical Core0 (without VPU) ABIST done.
21	VPU scanned ABIST done.
22	STS scanned ABIST done.
23	STS dedicated ABIST done.

Table 11-64. ABIST Status Register

Address:	x820004
Type:	RW, bits [21:26] are unspecified after a LBIST,SCAN0.SCAN and should be reinitialized before ABIST ABIST fail bits are only valid if the corresponding ABIST done bit is asserted. **Note:** Bits in this register are only updated during eps controlled operation. Turn ccintf_local_psav_dis if using ABIST with functional clocks.
Reset:	Set to '0000000 00000000' by power-on reset
Bit	Description
24	Physical Core ABIST Fail. ABIST OUT from the core evaluated whenTest DONE asserted, are conditions that must be met before this bit is cleared. Also evaluated if bit [26] of x'840002' is a '1'. Fail is held after being set. To clear with a SCOM write this register, the abist_out must be clear.
25	VPU scanned ABIST fail.
26	STS scanned ABIST fail.
27	STS dedicated ABIST fail.
28:63	Not implemented.

11.6.8.49 LBIST Options Register

Table 11-65. LBIST Options Register

Address:	x840002
Type:	RW
Reset:	Set to '000E0005 00000000' by power-on reset
Bit	Description
0:11	Not implemented.
12	PHYSICAL CORE 0 ABIST DONE MASK: Mask ABIST DONE from CORE 0. (ABIST PASS is also a mask as a result of masking the done).
13	STS + VPU SCANNED ABIST DONE MASK.
14	STS ABIST DONE MASK: Mask ABIST DONE from STS. (ABIST PASS is also a mask as a result of masking the done).
15	Not implemented.
16:21	Not implemented.
22	CONTINUAL LBIST: Set this in order to override the TEST LENGTH register. Useful for power and noise measurements. Continual LBIST is stopped with a write to the x'800009' register, eps_override bit.
23:25	Not implemented.
26	OVERRIDE ABIST DONE MASK. The abist pass bit is gated by not abist_done. If abist_done is not asserted then the abist pass status cannont be determined; however, by setting this OVERRIDE bit, you can observe the abist pass status. **Notes:** 1. The ABIST_FAIL is '0' until ABIST_DONE is set. Once ABIST_DONE is set, then ABIST_FAIL is alllowed to be set. 2. This override affects all three domains: Core, Scanned ABIST not CORE, and STS. **PROGRAMMING NOTE:** Because the gus_scan_abist requires x'FFFFF' + 1 number of tester loops to complete, where each tester loop is 131 times Scan_Clock_Rate number of mesh-clocks, only x'4FFFF' tester loops are run. Therefore the ABIST_DONE status bit will not be set, and the OVERRIDE ABIST DONE MASK should be set in order to validate the ABIST_FAIL status. The ABIST_DONE signal from the GUS_SCAN_ABIST engine (HTBC RLM) should be checked.

Table 11-65. LBIST Options Register

Address:	x840002
Type:	RW
Reset:	Set to '000E0005 00000000' by power-on reset
Bit	Description
27	ENABLE TDO DURING RUN_TEST_IDLE (for ABIST real time fail observation). The bit fail mapping and array diagnostics for debugging ABIST fails on the tester often have the need to observe the ABIST fail on a real time basis. Because the ABIST fail information is sent to ACCESS for status, it is conveniently routed to the TDO, typically a burnin pin as well. Special TDO_ENABLE is required in order to observe this signal. **Note:** Normal IEEE JTAG Specification has the TDO in high-impedance during run-test-idle. This state is used then to enable the TDO_EN during abist operations when bit 27 is asserted.
28:31	SCAN CLOCK SPEED**.** Set to the divide-by value to limit scan clock frequency during scan events of the event processor, in order to accommodate relaxed timing on the scan paths. Typical applications are flush0, lbist, scanned abist. This has no effect on system scan ring access using the '0F' access command, which occurs with TCK frequency when in the SHIFT_DR state. Bit [31] is the LSb. The nominal value is '0101' (for example, 1/6th of the system clock). A value of '0000' is supported, and can be taken advantage of for increaed simulation throughput. **Note:** The value x'0001' is invalid and must not be used.
32:63	Not implemented.

11.6.8.50 LBIST Channel Length Register

Table 11-66. LBIST Channel Length Register

Address:	x840008
Type:	RW
Reset:	Set to '00000000 00000000' by power-on reset
Bit	Description
0:15	CHANNEL LENGTH**.** Contains the 'number of cycles minus on' the EVENT PROCESSOR is to loop on a particular event that has the SCAN_CTL bit asserted in the micro-coded instructions, except for the case when RAMSTOP_CTL is set coincident with SCAN_CTL. For this exception case, only one scan clock pulse / ramc2 clock pulse are generated together after a delay of the SCAN_SPEED value from the beginning of the event.
16:63	Not implemented.

11.6.8.51 LBIST Test Length Register

Table 11-67. LBIST Test Length Register

Address:	x84000B
Type:	RW
Reset:	Set to '4000000 00000000' by power-on reset
Bit	Description
0:19	TEST LENGTH**.** Contains the 'number of cycles minus one' that the EVENT PROCESSOR is to loop on the micro-coded instructions. For LBIST, it would be the number of PRPG Loads/system clock loops to run. If loaded to all '0's, then ONE cycle PRPG Load/System Clock cycle is run. MISR CLEAR has no overriding effect on the test length. **Note:** Do not attempt less than 2 loops in this register while doing a RUN_N using the last event due to logic implementation (pipelining). Example: to loop 4 times, program 3 into this register ('00000000000000000011').
20:63	Not implemented.

11.6.8.52 Clock Ramping Configuration Register

Table 11-68. Clock Ramping Configuration Register

Address:	x84000D
Type:	RW
Reset:	Set to ‘2D75E300 00000000’ by power-on reset
Bit	Description
0:5	Ramp up: mod48 counter value to start c2 clock on every other cycle
6:11	Ramp up: mod48 counter value to fully turn on c2 clock
12:17	Ramp down: mod48 counter value to start c2 clock on every other cycle
18:23	Ramp down: mod48 counter value to fully turn on c2 clock
24:63	Unused

12. Vector Processing Unit

The Vector/SIMD technology, referred to in this document as the vector processing unit (VPU), provides a software model that accelerates the performance of various software applications and runs on reduced instruction set computing (RISC) microprocessors. This is a short vector parallel architecture that extends the instruction set architecture (ISA) of the PowerPC Architecture. It is based on separate vector/SIMD[1]-style execution units that have high data parallelism. This parallelism allows it to perform on multiple data elements in a single instruction. The term vector refers to the spatial parallel processing of short, fixed-length, one-dimensional matrices performed by an execution unit. It should not be confused with the temporal parallel (pipelined) processing of long, variable-length vectors performed by classical vector machines. High degrees of parallelism are achievable with simple in-order instruction dispatch and low-instruction bandwidth. However, the ISA is designed so as not to impede additional parallelism through superscalar dispatch to multiple execution units or multithreaded execution unit pipelines.

12.1 970FX Vector/SIMD Multimedia Overview

The VPU expands the current PowerPC Architecture through the addition of a 128-bit vector execution unit, which operates concurrently with the existing scalar integer and floating-point units. This new engine provides for highly parallel operations, allowing for the simultaneous execution of up to four 32-bit floating operations or sixteen 8-bit fixed-point operations in one instruction. All VPU data paths and execution units are 128 bits wide and are fully pipelined.

12.1.1 VPU Implementation

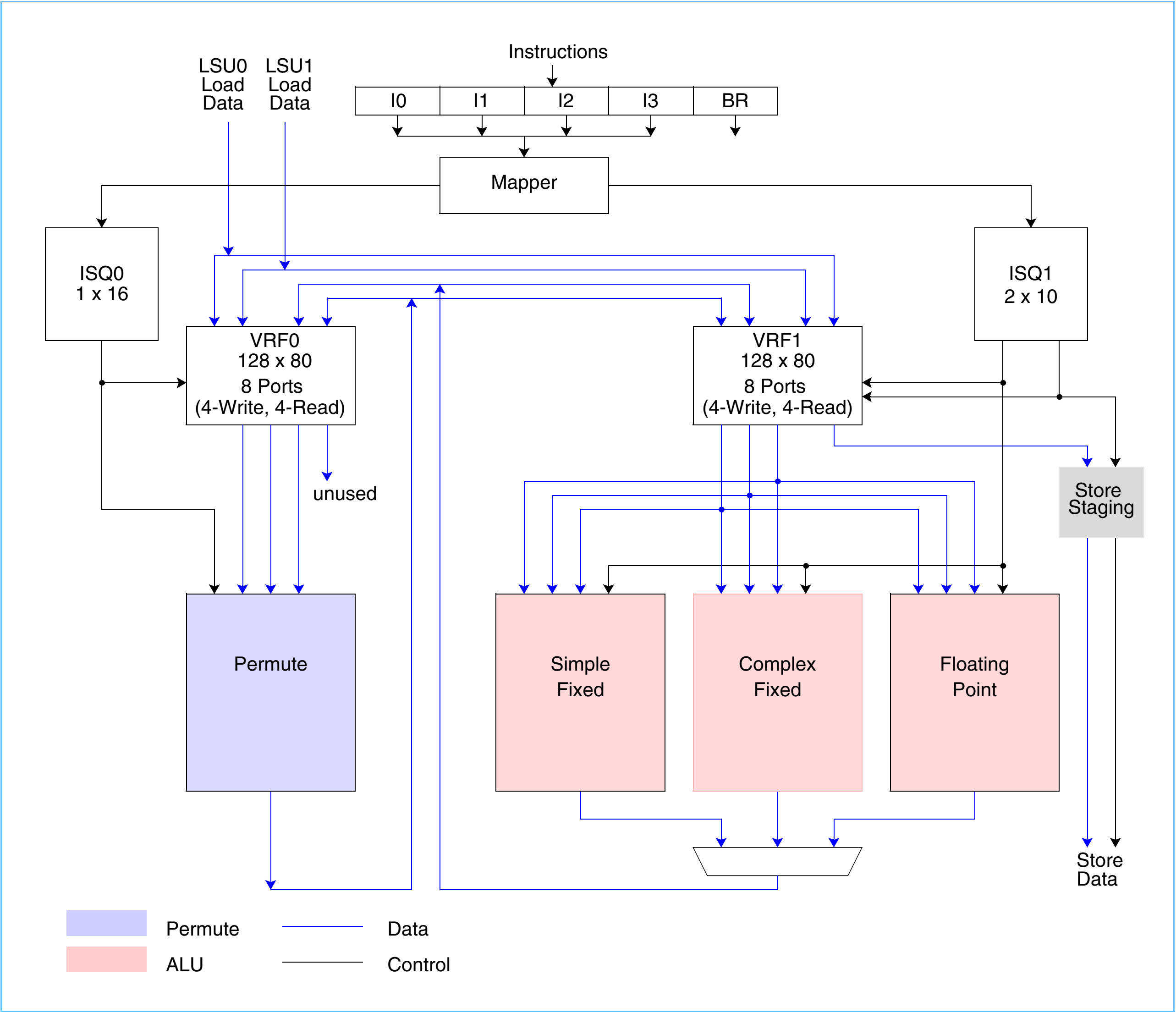
The 970FX implements many aspects of a preferred implementation as described in the *PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual*. The key features of a preferred implementation include:

- All data paths and execution units are 128 bits wide
- Two independent vector processing sub-units, one for all arithmetic logic unit (ALU) instructions and one for permute operations

The VPU is divided into two dispatchable units: vector ALU and vector permute. The vector ALU unit is further subdivided into a vector floating-point unit, a vector simple-fixed unit, and a vector complex-fixed unit. The vector ALU and permute units receive predecoded instructions from the issue queue in the instruction sequencer unit for the VPU (ISV). Vector instructions are issued to the appropriate vector unit when all of the source operands are available. Vector loads, stores, and data stream touch (DST) instructions are executed in the load/store unit (LSU) pipes. There are two copies of the Vector Register files; one provides operands for the vector permute unit, and one provides operands for the vector ALU. *Figure 12-1* provides a high-level view of the instruction sequencer unit (ISU) interaction with the VPUs.

1. Single instruction stream, multiple data streams

Figure 12-1. VPU Block Diagram



12.1.2 Vector ALU

Conceptually, the vector unit ALU is capable of operating on three source vectors and producing a single result vector on each instruction. The ALU is an SIMD-style arithmetic unit, where an instruction performs the same operation on all the data elements that comprise each vector. The ALU is partitioned into four separate ALUs for 32-bit integers and for single-precision floating-point operands. For 16-bit integers, the ALU is partitioned into 8 ALUs, and for 8-bit integers it is partitioned into 16 separate ALUs. No arithmetic is performed on elements larger than 32 bits. The largest adder in the vector ALU is 32 bits wide, and the largest multiplier array is 24 bits wide for the single-precision floating-point mantissa.

12.2 Vector Registers

12.2.1 VRSAVE Register

This 32-bit register is maintained and managed by software only. Each bit in the VRSAVE Register corresponds to a Vector Register and indicates whether the corresponding register is currently being used by the executing process. Therefore, the operating system needs to save and restore only those Vector Registers (VRs) when an exception occurs. The register is handled as a renamed register within the General Purpose Register (GPR) file in the 970FX microprocessor.

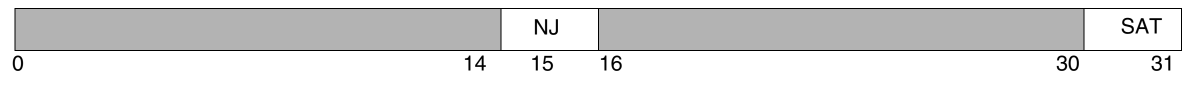
Note: If this approach is taken, it must be applied rigorously. If a program fails to indicate that a given vector register is in use, software errors can occur that will be difficult to detect and correct because they are timing-dependent. Some operating systems save and restore VRSAVE only for programs that also use other vector registers.

The VRSAVE Register can be accessed only by the Move From Special Purpose Register (**mfspr**) or Move To Special Purpose Register (**mtspr**) instructions. The **mfspr** instruction copies VRSAVE to the low-order 32 bits of a GPR; the **mtspr** instruction copies the low-order 32 bits of a GPR to VRSAVE.

12.2.2 Vector Status and Control Register (VSCR)

The Vector Status and Control Register is a special 32-bit register (not an SPR) that is read and written in a manner similar to the Floating-Point Status and Control Register (FPSCR) in the scalar floating-point unit. Two special instructions, Move From Vector Status and Control Register (**mfvscr**) and Move To Vector Status and Control Register (**mtvscr**), are provided to move the VSCR from and to a Vector Register. When moved to or from a Vector Register, the 32-bit VSCR is right justified in the 128-bit Vector Register. When moved to a Vector Register, the upper 96 bits (0:95) of the Vector Register are cleared (set to zeros).

Figure 12-2. VSCR Format



The VSCR has two defined bits, the non-Java mode (NJ) bit (VSCR[15]) and the saturation (SAT) bit (VSCR[31]). The remaining bits are reserved. VSCR bit settings are shown in *Table 12-1*.

The VSCR bits, after being moved to a Vector Register, are shown in *Figure 12-3*.

Figure 12-3. VSCR Moved to a Vector Register

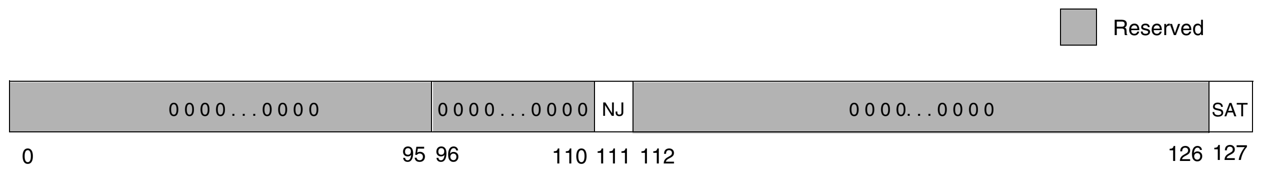


Table 12-1. VSCR Field Descriptions

Bits	Name	Description
0:14	—	Reserved.
15	NJ	Non-Java. A mode control bit that determines whether vector floating-point operations are performed in a mode that is compliant with Java, IEEE, and C9X or possibly in a faster noncompliant mode. 0 The Java-IEEE-C9X–compliant mode is selected. Denormalized values are handled as specified by the Java, IEEE, and C9X standard. 1 The non-Java/non-IEEE–compliant mode is selected. If an element in a source vector register contains a denormalized value, the value '0' is used instead. If an instruction causes an underflow exception, the corresponding element in the target vector register (VR) is cleared to '0'. In both cases, the '0' has the same sign as the denormalized or underflowing value.
16:30	—	Reserved.
31	SAT	Saturation. A sticky status bit indicating that some field in a saturating instruction became saturated since the last time SAT was cleared. In other words, when SAT is set to '1', it remains set to '1' until it is cleared to '0' by a **mtvscr** instruction. 0 Indicates no saturation has occurred since this bit was last cleared by a **mtvscr** instruction. 1 The vector saturate instruction implicitly sets when saturation has occurred on the results of one of the vector instructions having saturate in its name. (See *Table 12-4* on page 353.)

The **mtvscr** instruction is context synchronizing. This implies that all vector instructions logically preceding an **mtvscr** in the program flow will execute in the architectural context (NJ mode) that existed before completion of the **mtvscr**. It also implies that all instructions logically following the **mtvscr** will execute in the new context (NJ mode) established by the **mtvscr**.

After an **mfvscr** instruction executes, the result in the target Vector Register is architecturally precise. It reflects all updates to the SAT bit that could have been made by vector instructions logically preceding it in the program flow. Further, it will not reflect any SAT updates that can be made to it by vector instructions logically following it in the program flow. Reading the VSCR can be much slower than typical vector instructions, and therefore care must be taken in reading it to avoid performance problems.

12.3 Effects on Existing PowerPC Facilities

12.3.1 Control Flow

Vector instructions can be freely intermixed with existing PowerPC instructions to form a complete program. Vector instructions provide a vector compare and select mechanism to implement conditional execution as the preferred mechanism to control data flow in VPU programs. Vector compare instructions can update the Condition Register, thus providing the communication from the vector execution units to the PowerPC branch instructions necessary to modify program flow based on vector data.

12.3.1.1 Condition Register

The Condition Register (CR) is affected by the VPU architecture. The CR is a 32-bit register, divided into eight 4-bit fields, CR0-CR7 (see *Figure 12-4*), that reflect the results of certain arithmetic operations and provide a mechanism for testing and branching. For the VPU ISA, the CR6 field can optionally be used. If the record bit (Rc) of a vector instruction field is set in a vector compare instruction, then the CR6 field is updated according to *Table 12-2*.

Figure 12-4. Condition Register (CR)

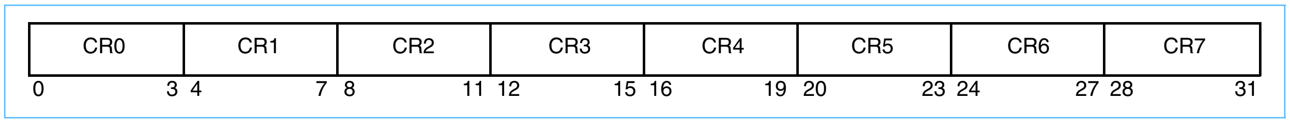


Table 12-2. CR6 Field Bit Settings for Vector Compare Instructions

Bits	Description
	Vector Compare
0	1 Comparison successful for all fields 0 Comparison failed for at least one field
1	Always zero
2	1 Comparison failed for all fields 0 Comparison successful for at least one field
3	Always zero
	Vector Compare Bounds (vcmpbfp)
0	Always zero
1	Always zero
2	1 All values within bounds 0 Not all values within bounds
3	Always zero

The Rc bit should be used sparingly. As for other PowerPC instructions, in some implementations, instructions with the Rc bit set to '1' could have a longer latency or be more disruptive to the instruction pipeline flow than instructions with the Rc bit set to '0'. Therefore, techniques of accumulating results and testing infrequently are advised.

12.3.1.2 Machine State Register

Certain bits in the Machine State Register (MSR) affect instructions in the vector data stream. MSR[VP] indicates whether the vector processor is available. *Table 12-3* defines the VP, PR, and DR bits.

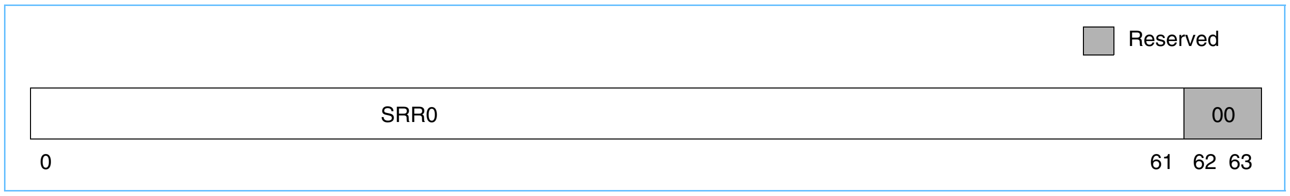
Table 12-3. MSR Bit Settings Affecting the VPU

Bits	Name	Description
38	VP	VPU available 0 The processor prevents execution of all vector instructions, including loads, stores, and moves. If such execution is attempted, a VPU unavailable exception is raised. 1 The processor can execute all vector instructions. **Note:** The VRSAVE Register is not protected by MSR[VP]. The data streaming family of instructions (**dst**, **dstt**, **dstst**, **dststt**, **dss**, and **dssall**) are not affected by the MSR[VP].
49	PR	Problem (user) state 0 The processor is privileged to execute any instruction. 1 The processor can only execute non-privileged instructions. **Note:** Care should be taken if data-stream prefetching is used in a supervisor (privileged) state (MSR[PR] = '0'). For each existing data stream, prefetching is enabled if (a) MSR[DR] is set to '1' and (b) MSR[PR] has the value it had when the **dst** or **dstst** instruction that specified the data stream was executed. Otherwise, prefetching for the data stream is suspended.
59	DR	Data address translation 0 Data address translation is disabled. If data stream touch (**dst**) and data stream touch for store (**dstst**) instructions are executed when DR is set to '0', the results are boundedly undefined. 1 Data address translation is enabled. Data stream touch (**dst**) and data stream touch for store (**dstst**) instructions are supported when DR is set to '1'.

12.3.2 Machine Status Save/Restore Registers (SRR0, SRR1)

SRR0 holds the effective address (EA) for the instruction that caused the VPU unavailable exception, and SRR1 holds the machine state status as described in *Chapter 4 Exceptions*.

Figure 12-5. Machine Status Save/Restore Register 0 (SRR0)



12.4 Exceptions

There are three exceptions that can result from the execution of a vector instruction:

- VPU unavailable exception
- VPU assist exception
- Data storage exception

12.4.1 VPU Unavailable Exception

This interrupt is described in *Section 12.3.1.2 Machine State Register*.

12.4.2 VPU Assist Exception

The VPU assist exception happens when operating in Java mode and either the input operands or the result of an operation are denormalized. After this exception, execution resumes at offset 0x0000_0000_0000_1700. See *Section 4.5.18 VPU Assist Exception* on page 104 for more information.

12.4.3 Data Storage Exception

Load vector indexed and store vector indexed instructions transfer quadword vectors between memory and vector registers. Load vector element indexed and store vector element indexed instructions transfer byte, halfword, and word scalar elements between memory and vector registers. All vector loads and vector stores use the index (**r**A|0 + **r**B) addressing mode to specify the target memory address. No update forms are provided. A load vector element indexed instruction transfers a scalar data element from memory into the destination vector register, leaving other elements in the vector with boundedly-undefined values. A store vector element indexed transfers a scalar data element from the source vector register to memory leaving other elements in the quadword unchanged. No data alignment occurs, that is, all scalar data elements are transferred directly on their natural memory byte-lanes to or from the corresponding element in the vector register. Quadword memory accesses made by load vector indexed and store vector indexed are not guaranteed to be atomic.

12.5 Optional Instructions

The 970FX microprocessor implements all vector instructions as listed in the *PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual*. See *Table 12-4 Supported Vector Instructions*.

12.5.1 Java Mode Instruction Handling Implementation.

The 970FX VPU implementation handles certain instructions differently based on the Java mode setting in the VSCR register. Java compliance does require compliance with certain aspects of the IEEE Standard including:

- Support of denorms as inputs and results (gradual underflow) for arithmetic operations.
- Not a number (NaN) results for invalid operations.
- NaNs compare unordered with respect to everything, so that the result of any comparison of any NaN to any data type is always false.
- NaNs are handled the same way in both the Java or non-Java mode for the 970FX implementation.

For some instructions, denormalization produces the exact result without trapping. The 970FX implementation of the VPU handles most denorms by trapping at interrupt vector 0x0000_0000_0000_1700 (VPU assist interrupt).

12.5.2 Least Recently Used (LRU) Instructions

The VPU architecture suggests that load vector indexed LRU (**lvxl**) and store vector indexed LRU (**stvxl**) are handled differently than the regular load/store instructions in that they leave cache entries in the least recently used (LRU) state instead of a most recently used (MRU) state. This supports efficient processing of data that is known to have little reuse and poor caching characteristics.

The 970FX microprocessor will treat **lvxl** and **stvxl** as a regular load and store with respect to the replacement algorithm. That is, the cache entry will be set as MRU.

12.5.3 Data Stream Instructions

Data stream (**dst**) instructions are broken up into two internal instructions (IOPs), one for the effective address and one for the prefetch information. They are marked serialized and are not executed until they are the next instruction to complete.

12.6 Vector Instruction Set

Table 12-4 lists the supported vector instructions.

Table 12-4. Supported Vector Instructions (Page 1 of 6)

Number	Mnemonic	Operands	Execution Unit	Description
Load Vector Element Indexed				
1	**lvebx**	vD,rA,rB	LOAD	Load Vector Element Byte Indexed
2	**lvehx**	vD,rA,rB	LOAD	Load Vector Element Halfword Indexed
3	**lvewx**	vD,rA,rB	LOAD	Load Vector Element Word Indexed
4	**lvx**	vD,rA,rB	LOAD	Load Vector Indexed
5	**lvxl**	vD,rA,rB	LOAD	Load Vector Indexed LRU
Store Vector Element Indexed				
6	**stvebx**	vS,rA,rB	STORE	Store Vector Element Byte Indexed
7	**stvehx**	vS,rA,rB	STORE	Store Vector Element Halfword Indexed
8	**stvewx**	vS,rA,rB	STORE	Store Vector Element Word Indexed
9	**stvx**	vS,rA,rB	STORE	Store Vector Indexed
10	**stvxl**	vS,rA,rB	STORE	Store Vector Indexed LRU
Load Vector for Shift				
11	**lvsl**	vD,rA,rB	LOAD	Load Vector for Shift Left
12	**lvsr**	vD,rA,rB	LOAD	Load Vector for Shift Right
Move To and Move From Vector Status and Control Register				
13	**mtvscr**	vB	Simple	Move To Vector Status and Control Register
14	**mfvscr**	vD	Simple	Move From Vector Status and Control Register
Data Stream				
15	**dst**	rA,rB,tag	LSU	Data Stream Touch
16	**dstt**	rA,rB,tag	LSU	Data Stream Touch Transient (treated as dst)
17	**dstst**	rA,rB,tag	LSU	Data Stream Touch for Store (treated as dst)
18	**dststt**	rA,rB,tag	LSU	Data Stream Touch for Store Transient (treated as dst)
19	**dss**	tag	LSU	Data Stream Stop
20	**dssall**		LSU	Data Stream Stop All
Vector Add				
21	**vaddubm**	vD,vA,vB	Simple	Vector Add Unsigned Byte Modulo
22	**vaddubs**	vD,vA,vB	Simple	Vector Add Unsigned Byte Saturate
23	**vaddsbs**	vD,vA,vB	Simple	Vector Add Signed Byte Saturate
24	**vadduhm**	vD,vA,vB	Simple	Vector Add Unsigned Halfword Modulo
25	**vadduhs**	vD,vA,vB	Simple	Vector Add Unsigned Halfword Saturate
26	**vaddshs**	vD,vA,vB	Simple	Vector Add Signed Halfword Saturate
27	**vadduwm**	vD,vA,vB	Simple	Vector Add Unsigned Word Modulo

Table 12-4. Supported Vector Instructions (Page 2 of 6)

Number	Mnemonic	Operands	Execution Unit	Description
28	**vadduws**	vD,vA,vB	Simple	Vector Add Unsigned Word Saturate
29	**vaddsws**	vD,vA,vB	Simple	Vector Add Signed Word Saturate
30	**vaddfp**	vD,vA,vB	Float	Vector Add Float
Vector Add and Write Carry-Out				
31	**vaddcuw**	vD,vA,vB	Simple	Vector Add and Write Carry-Out Unsigned Word
Vector Subtract				
32	**vsububm**	vD,vA,vB	Simple	Vector Subtract Unsigned Byte Modulo
33	**vsububs**	vD,vA,vB	Simple	Vector Subtract Unsigned Byte Saturate
34	**vsubsbs**	vD,vA,vB	Simple	Vector Subtract Signed Byte Saturate
35	**vsubuhm**	vD,vA,vB	Simple	Vector Subtract Unsigned Halfword Modulo
36	**vsubuhs**	vD,vA,vB	Simple	Vector Subtract Unsigned Halfword Saturate
37	**vsubshs**	vD,vA,vB	Simple	Vector Subtract Signed Halfword Saturate
38	**vsubuwm**	vD,vA,vB	Simple	Vector Subtract Unsigned Word Modulo
39	**vsubuws**	vD,vA,vB	Simple	Vector Subtract Unsigned Word Saturate
40	**vsubsws**	vD,vA,vB	Simple	Vector Subtract Signed Word Saturate
41	**vsubfp**	vD,vA,vB	Float	Vector Subtract Float
Vector Subtract and Write Carry-Out				
42	**vsubcuw**	vD,vA,vB	Simple	Vector Subtract and Write Carry-Out Unsigned Word
Vector Multiply Odd Integer				
43	**vmuloub**	vD,vA,vB	Complex	Vector Multiply Odd Unsigned Byte
44	**vmulosb**	vD,vA,vB	Complex	Vector Multiply Odd Signed Byte
45	**vmulouh**	vD,vA,vB	Complex	Vector Multiply Odd Unsigned Halfword
46	**vmulosh**	vD,vA,vB	Complex	Vector Multiply Odd Signed Halfword
Vector Multiply Even Integer				
47	**vmuleub**	vD,vA,vB	Complex	Vector Multiply Even Unsigned Byte
48	**vmulesb**	vD,vA,vB	Complex	Vector Multiply Even Signed Byte
49	**vmuleuh**	vD,vA,vB	Complex	Vector Multiply Even Unsigned Halfword
50	**vmulesh**	vD,vA,vB	Complex	Vector Multiply Even Signed Halfword
Vector Multiply-Add				
51	**vmhaddshs**	vD,vA,vB,vC	Complex	Vector Multiply-High and Add Signed Halfword Saturate
52	**vmhraddshs**	vD,vA,vB,vC	Complex	Vector Multiply-High Round and Add Signed Halfword Saturate
53	**vmladduhm**	vD,vA,vB,vC	Complex	Vector Multiply-Low and Add Unsigned Halfword Modulo
54	**vmaddfp**	vD,vA,vC,vB	Float	Vector Multiply-Add Float
Vector Multiply-Sum Integer				
55	**vmsumubm**	vD,vA,vB,vC	Complex	Vector Multiply-Sum Unsigned Byte Modulo
56	**vmsummbm**	vD,vA,vB,vC	Complex	Vector Multiply-Sum Mixed-Sign Byte Modulo

Table 12-4. Supported Vector Instructions (Page 3 of 6)

Number	Mnemonic	Operands	Execution Unit	Description
57	**vmsumuhm**	vD,vA,vB,vC	Complex	Vector Multiply-Sum Unsigned Halfword Modulo
58	**vmsumuhs**	vD,vA,vB,vC	Complex	Vector Multiply-Sum Unsigned Halfword Saturate
59	**vmsumshm**	vD,vA,vB,vC	Complex	Vector Multiply-Sum Signed Halfword Modulo
60	**vmsumshs**	vD,vA,vB,vC	Complex	Vector Multiply-Sum Signed Halfword Saturate
Vector Sum Across Signed Integer Saturate				
61	**vsumsws**	vD,vA,vB	Complex	Vector Sum Across Signed Word Saturate
Vector Sum Across Partial (1/2) Signed Integer Saturate				
62	**vsum2sws**	vD,vA,vB	Complex	Vector Sum Across Partial (1/2) Signed Word Saturate
Vector Sum Across Partial (1/4) Integer Saturate				
63	**vsum4ubs**	vD,vA,vB	Complex	Vector Sum Across Partial (1/4) Unsigned Byte Saturate
64	**vsum4sbs**	vD,vA,vB	Complex	Vector Sum Across Partial (1/4) Signed Byte Saturate
65	**vsum4shs**	vD,vA,vB	Complex	Vector Sum Across Partial (1/4) Signed Halfword Saturate
Vector Average Integer				
66	**vavgub**	vD,vA,vB	Simple	Vector Average Unsigned Byte
67	**vavgsb**	vD,vA,vB	Simple	Vector Average Signed Byte
68	**vavguh**	vD,vA,vB	Simple	Vector Average Unsigned Halfword
69	**vavgsh**	vD,vA,vB	Simple	Vector Average Signed Halfword
70	**vavguw**	vD,vA,vB	Simple	Vector Average Unsigned Word
71	**vavgsw**	vD,vA,vB	Simple	Vector Average Signed Word
Vector Logical				
72	**vand**	vD,vA,vB	Simple	Vector Logical AND
73	**vor**	vD,vA,vB	Simple	Vector Logical OR
74	**vxor**	vD,vA,vB	Simple	Vector Logical XOR
75	**vandc**	vD,vA,vB	Simple	Vector Logical AND with Complement
76	**vnor**	vD,vA,vB	Simple	Vector Logical NOR
Vector Rotate Left Integer				
77	**vrlb**	vD,vA,vB	Simple	Vector Rotate Left Integer Byte
78	**vrlh**	vD,vA,vB	Simple	Vector Rotate Left Integer Halfword
79	**vrlw**	vD,vA,vB	Simple	Vector Rotate Left Integer Word
Vector Shift Left Integer				
80	**vslb**	vD,vA,vB	Simple	Vector Shift Left Integer Byte
81	**vslh**	vD,vA,vB	Simple	Vector Shift Left Integer Halfword
82	**vslw**	vD,vA,vB	Simple	Vector Shift Left Integer Word
83	**vsl**	vD,vA,vB	Simple	Vector Shift Left

Table 12-4. Supported Vector Instructions (Page 4 of 6)

Number	Mnemonic	Operands	Execution Unit	Description
Vector Shift Right Integer				
84	**vsrb**	vD,vA,vB	Simple	Vector Shift Right Byte
85	**vsrab**	vD,vA,vB	Simple	Vector Shift Right Algebraic Byte
86	**vsrh**	vD,vA,vB	Simple	Vector Shift Right Halfword
87	**vsrah**	vD,vA,vB	Simple	Vector Shift Right Algebraic Halfword
88	**vsrw**	vD,vA,vB	Simple	Vector Shift Right Word
89	**vsraw**	vD,vA,vB	Simple	Vector Shift Right Algebraic Word
90	**vsr**	vD,vA,vB	Simple	Vector Shift Right
Vector Compare Greater-Than				
91	**vcmpgtub**[.]	vD,vA,vB	Simple	Vector Compare Greater-Than Unsigned Byte [Record]
92	**vcmpgtsb**[.]	vD,vA,vB	Simple	Vector Compare Greater-Than Signed Byte [Record]
93	**vcmpgtuh**[.]	vD,vA,vB	Simple	Vector Compare Greater-Than Unsigned Halfword [Record]
94	**vcmpgtsh**[.]	vD,vA,vB	Simple	Vector Compare Greater-Than Signed Halfword [Record]
95	**vcmpgtuw**[.]	vD,vA,vB	Simple	Vector Compare Greater-Than Unsigned Word [Record]
96	**vcmpgtsw**[.]	vD,vA,vB	Simple	Vector Compare Greater-Than Signed Word [Record]
97	**vcmpgtfp**[.]	vD,vA,vB	Simple	Vector Compare Greater-Than Float [Record]
Vector Compare Equal-To				
98	**vcmpequb**[.]	vD,vA,vB	Simple	Vector Compare Equal-To Unsigned Byte [Record]
99	**vcmpequh**[.]	vD,vA,vB	Simple	Vector Compare Equal-To Unsigned Halfword [Record]
100	**vcmpequw**[.]	vD,vA,vB	Simple	Vector Compare Equal-To Unsigned Word [Record]
101	**vcmpeqfp**[.]	vD,vA,vB	Simple	Vector Compare Equal-To Float [Record]
Vector Compare Greater-Than-or-Equal-To				
102	**vcmpgefp**[.]	vD,vA,vB	Simple	Vector Compare Greater-Than-or-Equal-To Float [Record]
Vector Compare Bounds Float				
103	**vcmpbfp**[.]	vD,vA,vB	Simple	Vector Compare Bounds Float [Record]
Vector Conditional Select				
104	**vsel**	vD,vA,vB,vC	Simple	Vector Conditional Select
Vector Pack				
105	**vpkuhum**	vD,vA,vB	Permute	Vector Pack Unsigned Halfword Unsigned Modulo
106	**vpkuhus**	vD,vA,vB	Permute	Vector Pack Unsigned Halfword Unsigned Saturate
107	**vpkshus**	vD,vA,vB	Permute	Vector Pack Signed Halfword Unsigned Saturate
108	**vpkshss**	vD,vA,vB	Permute	Vector Pack Signed Halfword Signed Saturate
109	**vpkuwum**	vD,vA,vB	Permute	Vector Pack Unsigned Word Unsigned Modulo
110	**vpkuwus**	vD,vA,vB	Permute	Vector Pack Unsigned Word Unsigned Saturate
111	**vpkswus**	vD,vA,vB	Permute	Vector Pack Signed Word Unsigned Saturate
112	**vpkswss**	vD,vA,vB	Permute	Vector Pack Signed Word Signed Saturate

Table 12-4. Supported Vector Instructions (Page 5 of 6)

Number	Mnemonic	Operands	Execution Unit	Description
113	**vpkpx**	vD,vA,vB	Permute	Vector Pack Pixel32
Vector Unpack High				
114	**vupkhsb**	vD,vB	Permute	Vector Unpack High Signed Byte
115	**vupkhsh**	vD,vB	Permute	Vector Unpack High Signed Halfword
116	**vupkhpx**	vD,vB	Permute	Vector Unpack High Pixel16
Vector Unpack Low				
117	**vupklsb**	vD,vB	Permute	Vector Unpack Low Signed Byte
118	**vupklsh**	vD,vB	Permute	Vector Unpack Low Signed Halfword
119	**vupklpx**	vD,vB	Permute	Vector Unpack Low Pixel16
Vector Merge High				
120	**vmrghb**	vD,vA,vB	Permute	Vector Merge High Byte
121	**vmrghh**	vD,vA,vB	Permute	Vector Merge High Halfword
122	**vmrghw**	vD,vA,vB	Permute	Vector Merge High Word
Vector Merge Low				
123	**vmrglb**	vD,vA,vB	Permute	Vector Merge Low Byte
124	**vmrglh**	vD,vA,vB	Permute	Vector Merge Low Halfword
125	**vmrglw**	vD,vA,vB	Permute	Vector Merge Low Word
Vector Splat				
126	**vspltb**	vD,vB,UIM	Permute	Vector Splat Byte
127	**vsplth**	vD,vB,UIM	Permute	Vector Splat Halfword
128	**vspltw**	vD,vB,UIM	Permute	Vector Splat Word
Vector Splat Immediate Signed Integer				
129	**vspltisb**	vD,SIM	Permute	Vector Splat Immediate Signed Byte
130	**vspltish**	vD,SIM	Permute	Vector Splat Immediate Signed Halfword
131	**vspltisw**	vD,SIM	Permute	Vector Splat Immediate Signed Word
Vector Permute				
132	**vperm**	vD,vA,vB,vC	Permute	Vector Permute
Vector Shift Left Double by Octet Immediate				
133	**vsldoi**	vD,vA,vB,SH	Permute	Vector Shift Left Double by Octet Immediate
Vector Shift by Octet				
134	**vslo**	vD,vA,vB	Permute	Vector Shift Left by Octet
135	**vsro**	vD,vA,vB	Permute	Vector Shift Right by Octet
Vector Maximum				
136	**vmaxub**	vD,vA,vB	Simple	Vector Maximum Unsigned Byte
137	**vmaxsb**	vD,vA,vB	Simple	Vector Maximum Signed Byte
138	**vmaxuh**	vD,vA,vB	Simple	Vector Maximum Unsigned Halfword

Table 12-4. Supported Vector Instructions (Page 6 of 6)

Number	Mnemonic	Operands	Execution Unit	Description
139	**vmaxsh**	vD,vA,vB	Simple	Vector Maximum Signed Halfword
140	**vmaxuw**	vD,vA,vB	Simple	Vector Maximum Unsigned Word
141	**vmaxsw**	vD,vA,vB	Simple	Vector Maximum Signed Word
142	**vmaxfp**	vD,vA,vB	Simple	Vector Maximum Float
Vector Minimum				
143	**vminub**	vD,vA,vB	Simple	Vector Minimum Unsigned Byte
144	**vminsb**	vD,vA,vB	Simple	Vector Minimum Signed Byte
145	**vminuh**	vD,vA,vB	Simple	Vector Minimum Unsigned Halfword
146	**vminsh**	vD,vA,vB	Simple	Vector Minimum Signed Halfword
147	**vminuw**	vD,vA,vB	Simple	Vector Minimum Unsigned Word
148	**vminsw**	vD,vA,vB	Simple	Vector Minimum Signed Word
149	**vminfp**	vD,vA,vB	Simple	Vector Minimum Float
Vector Estimate Float				
150	**vrefp**	vD,vB	Float	Vector Reciprocal Estimate Float
151	**vrsqrtefp**	vD,vB	Float	Vector Reciprocal Square Root Estimate Float
152	**vlogefp**	vD,vB	Float	Vector Log 2 Estimate Float
153	**vexptefp**	vD,vB	Float	Vector 2 Raised to the Exponent Estimate Float
Vector Negative Multiply-Subtract Float				
154	**vnmsubfp**	vD,vA,vC,vB	Float	Vector Negative Multiply-Subtract Float
Vector Round to Floating-Point Integral Value				
155	**vrfin**	vD,vB	Float	Vector Round to Floating-Point Integer Nearest
156	**vrfiz**	vD,vB	Float	Vector Round to Floating-Point Integer toward Zero
157	**vrfip**	vD,vB	Float	Vector Round to Floating-Point Integer toward Positive Infinity
158	**vrfim**	vD,vB	Float	Vector Round to Floating-Point Integer toward Minus Infinity
Vector Convert To Fixed-Point				
159	**vctuxs**	vD,vB,UIM	Float	Vector Convert to Unsigned Fixed-Point Word Saturate
160	**vctsxs**	vD,vB,UIM	Float	Vector Convert to Signed Fixed-Point Word Saturate
Vector Convert From Fixed-Point				
161	**vcfux**	vD,vB,UIM	Float	Vector Convert From Unsigned Fixed-Point Word
162	**vcfsx**	vD,vB,UIM	Float	Vector Convert From Signed Fixed-Point Word

Revision Log

Each release of this document supersedes all previously released versions. The revision log lists all significant changes made to the document since its initial release. In the rest of the document, change bars in the margin indicate that the adjacent text was modified from the previous release of this document.

Revision Date	Page	Contents of Modification
March 14, 2008		Version 1.7
	19	• Added reference to *PowerPC 970FX Power On Reset Application Note* in *Additional Documentation* on page 19.
	77	• Revised *Section 3.5 Functional Units*.
	113	• Added a new section on software optimization (see *Section 6 Software Optimization Guidelines*).
	349	• Simplified the CR6 Field Bit Settings in *Table 12-2 CR6 Field Bit Settings for Vector Compare Instructions*.
		• Removed *Section 2.2.1.3 Invalid Forms.* • Removed *Section 3.5.3 Non-Cacheable Unit.* • Removed *Section 3.5.4.1 Overview of the Hardware-Controlled Data Prefetch.* • Removed *Section 3.5.4.2 Hardware Prefetch Engine Implementation.* • Removed *Section 3.5.4.4 Vector Prefetch Instruction Support.* • Removed *Section 3.5.4.5 Programmability.* • Removed *Section 5.3 Memory Segment Model.*
December 19, 2005	36, 46, 66, 74, 98	Version 1.6 • Corrected HID0[61]. • Clarified the HIOR definition. • Edited *Section 3.3.4.1 L2 Cache Flush Algorithm*. • Edited *Section Software Touch Instruction*, changed FIFO replacement algorithm to LRU replacement algorithm. • Added SPRG3 (User level) added to *Figure 2-1* and tables.

Revision Date	Page	Contents of Modification
June 19, 2005	38, 48, 53, 55, 60, 111, 222, 223, 224, 267, 341, 370, 375, 400, 403, 427	Version 1.5 • Corrected *Section 2.2.1.4 Illegal Instructions*, regarding illegal opcodes. • Updated *Chapter 10. 970FX Performance Monitor.* • Added cross-references to the corresponding register descriptions *in Table 11-12 I2C Registers used by the 970FX Processor Interconnect Bus*. • Moved *Table 11-9. Processor Configurable Timing Delay Parameter Register (BUS-CONF)* to *Section 11.6.8.22 Bus Configuration Register.* • Updated definition of bit[47] in *Table 11-35 970FX Bus Fault Isolation Register / And Mask / Or Mask*. • Removed references to Enhanced Trace Facility (Chapter 11). • Updated *Table 8-6 Programmable Delay Parameters* - COMPACE, SNOOPLAT, and SNOOPACC values. • Updated *Section 8.2.2.1 Two-Beat Transfers* and *Section 8.2.2.2 Multi-Beat Transfers*. • Updated *Section 11.1 Power-On Reset*. • Removed references to the soft patch facility in*Section 2.1.3.5 Instruction Match CAM Array Access Register (IMC)*, *Chapter 4. Exceptions*, *Chapter 10. 970FX Performance Monitor*, *Chapter 11. System Design*, and *Chapter 12. Vector Processing Unit*. • Revised HID0[7] in *Table 2-7 HID0 Bit Functions*. • Changed register names of EISTAT to PI STAT REG; EICONF to PI MODE REG in *Table 11-12 I2C Registers used by the 970FX Processor Interconnect Bus*. • Added Logical Partitioning Registers to *Section 2.1.4 Logical Partitioning Function Registers* and *Figure 2-1 PowerPC 970FX Microprocessor Programming Model—Registers*. • Changed the second ECC mode to use BUSCONF[49, 56, 57] = '111'.
November 10, 2004		Version 1.41 • Initial version.