PowerPC 970MP Power On Reset Application Note

Version 1.0

Preliminary

July 28, 2006

© Copyright International Business Machines Corporation 2005, 2006

All Rights Reserved
Printed in the United States of America July-2006

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		

IEEE is a registered trademark in the United States, owned by the Institute of Electrical and Electronics Engineers.

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.

While the information contained herein is believed to be accurate, such information is preliminary, and should not be relied upon for accuracy or completeness, and no representations or warranties of accuracy or completeness are made.

Note: This document contains information on products in the design, sampling and/or initial production phases of development. This information is subject to change without notice. Verify with your IBM field applications engineer that you have the latest version of this document before finalizing a design.

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 Microelectronics home page can be found at **ibm.com**/chips

July 28, 2006

Table of Contents

List of Tables

List of Figures

1. PowerPC 970MP Power On Reset

1.1 Introduction

Earlier PowerPC designs had simplified power on reset requirements. They usually only required clocks and power to be stable followed by $\overline{\text{CP0\_HRESET}}$. Operational modes were selected by pin strapping at $\overline{\text{CP0\_HRESET}}$. Initialization of on-chip logic and arrays was handled by simple state machines triggered by the assertion of $\overline{\text{CP0\_HRESET}}$.

But the 970MP requires a more complicated power on reset sequence. Processor initialization is handled by the pervasive logic which is controlled either by I2C 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.

This application note is intended to serve as a guide to designers of PowerPC 970MP based systems. It will explain the sequence of events required to start executing code on a 970MP and provide debugging tips for power on reset code.

Note: The IBM PowerPC 970MP incorporates two complete microprocessors on a single chip, along with some common logic to connect these microprocessors to a system. The terms microprocessor, processor, and processing unit (PU) will be used interchangeably to describe each of the two individual processors. The term core refers to the instruction fetch and execution logic, including the L1 caches, but excluding the storage subsystem, of each processor. The term 970MP refers to the single chip module comprising the two processors and common logic.

1.2 Overview

The power on reset of a 970MP system goes through 8 separate phases, numbered IPL 0-7 as shown in *Table 1-5* on page 21. The sequence actually begins with the ramping of power and clocks prior to IPL 0. Once power supplies and clocks are stable, $\overline{\text{CP0\_HRESET}}$ and $\overline{\text{BYPASS}}$ should be asserted to reset the on-chip logic and PLL, respectively.

In addition to the initialization of the 970MP, the service processor (SPU) handles other functions via I2C. Initialization/configuration of the North Bridge, DIMMS, Clock chips, etc. must all be handled as part of the system POR sequence but initialization of those devices is beyond the scope of this document.

1.2.1 Differences between PowerPC 970MP and the PowerPC 970FX for Power On Reset

The 970MP POR sequence is very similar to that used for the 970FX. A PowerPC 970MP can be viewed as 2 970FX parts on a single package for the purposes of power on reset. Both cores require mode ring data to be initialized.

1.2.1.1 I2C addressing of PowerPC 970MP

The I2C address of the PowerPC 970MP is specified by the binary value 0b1000ppc where pp= the setting of the Processor ID bits via the PROCID pins (0:1), and c identifies the core, c=0 for core 0 and c=1 for core 1. For example, if the PROCID pins are both low (0, pulled to GND), the address for core 0 is 0b10000000, and the address of core 1 is 0b10000001.

1.2.1.2 Power On Reset SPU Hardware Considerations

It is assumed that 970MP systems will include a service processor, referred to herein as the SPU. The SPU usually consists of a low cost microcontroller. This microcontroller is responsible for hardware initialization of the 970MP 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 $\overline{\text{CP0\_HRESET}}$, $\overline{\text{CP1_HRESET}}$and $\overline{\text{BYPASS}}$ on the 970MP and should also have either a dedicated I2C bus master to initialize the system, or GPIO (General Purpose I/O Pins) that can be used to implement an I2C bus master.

The SPU should also have the ability to control I2CSEL, if the system is planned to also support use of a JTAG based debugger (for example, RISCWatch). This pin is used to force the internal logic to use I2C when high, or JTAG operation when low.

1.2.1.3 Bus Configuration

The larger L2 cache, the bus arbiter between the two cores, and the use of the EI2 receiver design combine to introduce additional cycles of delay in the path between the L2 cache and the North Bridge. In particular, the EI2 receiver adds one cycle of delay on incoming signals, and the bus arbiter adds two cycles of delay in each direction. This results in an additional five cycles of latency for a snoop response, for example. The programmable delay parameters, which are set to system dependent values during initialization, must account for these larger latencies in the 970MP. The range of values that can be specified for each of these parameters for the 970MP is shown in *Table 1-1*.

Table 1-1. Programmable Delay Parameters

Parameter	Description	Minimum	Maximum
COMPACE	Command pipeline delay	2	14
STATLAT[1]	Transfer handshake response latency	4	30
SNOOPLAT[2]	Processor snoop latency	6	12
SNOOPACC[2,3]	North Bridge snoop accumulation delay	9	24

1. Due to the additional staging of bus signals internally, setting a STATLAT value of 24 corresponds to a delay of 22 bus beats between the last beat of the ADO packet and the first beat of the transfer-handshake packet.
2. Similarly, a value two larger than the actual bus delay must be programmed into the SNOOPLAT and SNOOPACC fields.
3. SNOOPACC is a 4-bit field. When coded with a value of 1 - 15, the actual value is x + 8. For example, a one in the SNOOPACC field is actually a nine. When a zero is coded in this field, the actual value is 24.

1.3 Accessing Voltage Fuse Code (VFC) and Thermal Diode Calibration Data

The 970MP includes fuse data, called VFC for Voltage Fuse Code, on chip that indicates the recommended VDD setting and an on-chip thermal diode that can be used to monitor die temperature. This thermal diode should be used with an external100 µA current source. The voltage drop across the diode can be used to determine die temperature.

Each 970MP includes thermal diode calibration data stored in on-chip fuses. This calibration data indicates two test temperatures and two measured voltages. The voltages encoded in the fuse ring will provide the correct slope for measuring chip temperature.

The 970MP should first be started using the nominal voltage indicated in the 970MP Datasheet. This nominal voltage is used to scan out the encoded voltage and the thermal diode information. This operation can optionally done once during the first bring-up of the board, then the VFC and thermal diode calibration settings can be stored in nonvolatile memory for subsequent power on reset sequences.

Scanning the VFC and thermal diode data from the chip requires running part of the POR sequence in order to sample the fuses into the fuse ring, then scanning out the fuse ring, then restarting the POR sequence from the beginning (assertion of $\overline{\text{HRESET}}$).

1.3.1 Sampling on-chip fuses

The POR sequence should be followed from *Section 1.5.2* beginning on page 23 through *Section 1.5.5* on page 25. This includes ramping of power (using the nominal voltage) and clocks, releasing $\overline{\text{HRESET}}$, and completing the continue commands for IPL0 and IPL1. Once the service processor has completed the steps for IPL1, the fuses have been sampled and can be scanned out.

1.3.2 Scanning fuse data

Scanning the thermal diode data involves a series of TAP commands via I2C. TAP commands are used to emulate JTAG operations on the 970MP and are explained in more detail starting in *Appendix A.3.3 Primitive Command Summary* on page 40.

To begin the ring scan, send the I2C byte sequence 0x08, 0x40, 0x52, 0xDF, 0x00. This sets the JTAG to Shift-IR mode. Then set for scan-out and send the ring address by sending 0xDE, 0x40, 0x52, followed by the ring address which is 0xC00002, and is sent byte reversed as follows: 0x02, 0x00, 0xC0, then sending 0x12.

Then set the JTAG to Shift-DR mode by using the byte sequence 0x02, 0x40, 0x52, 0x03.

Now the JTAG is ready to scan the fuse ring out using 64 bit (8 byte) reads. Each 64 bit read operation begins by writing the preamble 0xBE, 0x40, 0x52, then reading 8 bytes of data from the 970MP. This scan of the fuse ring data proceeds from the highest numbered bit (2535) in the ring to bit 0, the first bit in the ring.

The first 64 bit read includes part of the VFC encoded setting. The 9PPC 70MP Datasheet shows this encoded data starting in bits 2470-2483. The first read includes bits 2471-2535.

Since the thermal diode calibration data is near the end of the ring in bits 2424-2469, the second 64 bit read includes all the thermal diode data. This data is encoded per *Table 1-3 Thermal Diode Data Encoding*.

1.3.2.1 VFC and Thermal Diode Data Example

Here is an example of a scan of the VFC and thermal diode information. Per the instructions in *Section 1.3.2 Scanning fuse data*, the scan begins with the sequence of writes as follows:

0x08, 0x40, 0x52, 0xDF, 0x00 (Shift IR Mode)

0xDE, 0x40, 0x52, 0x02, 0x00, 0xC0, 0x12 (Set scan out with ring address = 0xC00002)

0x02, 0x40, 0x52, 0x03 (Shift DR Mode)

0xBE, 0x40, 0x52 (Scan next 64 bits of data)

Then the service processor reads 8 bytes, for example: 0x04, 0x50, 0x22, 0x00, 0x00, 0x00, 0x00, 0x00.

Then another write of 0xBE, 0x40, 0x52 to scan the next 64 bits.

Then the service processor can read the next 8 bytes, for example: 0x88, 0xF1, 0x00, 0x18, 0x8E, 0x7F, 0x12, 0x2c. These two 8 byte blocks of data contain both the VFC and the thermal diode data which are ready to decode.

Table 1-2. Decoding Thermal and VFC Fuse Data Example

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Data Block 1	0x04	0x50	0x22	0x00	0x00	0x00	0x00	0x00
Bit swap	0x20	0x0A	0x44	0x00	0x00	0x00	0x00	0x00
Binary	00100000	00001010	01000100	00000000	00000000	00000000	00000000	00000000
Ring bits	2536:2529	2528:2521	2520:2513	2512:2505	2504:2497	2496:2489	2488:2481	2480:2473
Data Block 2	0x88	0xF1	0x00	0x18	0x8E	0x7F	0x12	0x2C
Bitswap	0x11	0x8F	0x00	0x18	0x71	0xFE	0x48	0x34
Binary	00010001	10001111	00000000	00011000	01110001	11111110	01001000	00110100
Ring Bits	2472:2465	2464:2457	2456:2449	2448:2441	2440:2433	2432:2425	2424:2417	2416:2409

The data bytes are bit reversed (bit0=bit7, bit 1=bit6, etc.) so we convert: 0x04, 0x50, 0x22, 0x00, 0x00, 0x00, 0x00, 0x00 to 0x20, 0x0A, 0x44, 0x00, 0x00, 0x00, 0x00, 0x00 and the second block from 0x88, 0xF1, 0x00, 0x18, 0x8E, 0x7F, 0x12, 0x2C to 0x11, 0x8F, 0x00, 0x18, 0x71, 0xFE, 0x48, 0x34. See *Table 1-2 Decoding Thermal and VFC Fuse Data Example* to get a clearer picture of this bit swapped data and the corresponding binary representation.

Notice that the binary version of the ring data, once it has been rearranged in this format is now the exact sequence of bits from the fuse ring. This makes it easier to extract numbered bits and determine their value.

The PowerPC 970MP Datasheet provides the VDD encoding information in the section headed VDD Fuse Coding. This section indicates that there are 4 possible 14 bit strings containing the VDD data: 2470:2483 (4th Pass) 2484:2497 (3rd Pass), 2498:2511 (2nd Pass), and 2512:2525 (1st Pass). That section also indicates that the 4th pass location should be read first, if those bits are all 0, proceed to the 3rd pass and so on until you come to the first test location with non-zero data.

The 4th-2nd Pass locations are all zero in this example, so we will use the 1st Pass location to determine the VFC bits. The datasheet shows bit 2525 is the VFC parity bit, with bits 2422:2424 containing the 3 bit VFC. Bit 2525 is the 5th bit from the right in byte 2, a zero. Bits 2422:2424 are the next three bits equal to 0b101. So this part has the VFC code of 101 with parity 0. The 970MP Datasheet will provide the correct VDD setting for this part based on this VFC code.

The thermal diode data is contained in bits 2424:2447. The first value encoded is the elevated test temperature. This value is in bits 2463:2469. These bits are in the 2nd block of 8 bytes, and “straddle” bytes 1 and 2. Starting with bit 2469 in byte 1 and running through 2463 in byte 2 (2 high order bits of byte 2) we get the binary field 0b1000110 or 0x46. Converting to decimal provides the elevated test temperature of 70C.

The next field to be extracted is the low test temperature in bits 2456:2462. They are in the 2nd block 6 lowest bits of byte 2 and the msb of byte 3. Extracting those 7 bits yields the binary value 0b0011110 or 0x1e. Converted to decimal this is equal to 30C. The low temperature value range includes an offset of -40C so removing that offset yields a low test temperature of -10C.

The next field from bits 2447:2436 contains the elevated temperature thermal diode reading in millivolts. Bit 2447 is in the 7 lsb bits of byte 4 in the second block, followed by 5 bits of byte 5 to get the 12 bit value 0b001100001110 or 0x30E or decimal 782. This is converted to a decimal value in millivolts by dividing by 2 and adding 300. So the thermal diode reading at elevated temperature equals 782 divided by 2 (391), adding 300 resulting in 691 millivolts.

The last field to be extracted from bits 2435:2424 provides the low temperature reading in millivolts. This value is in the 3 lsb bits of byte 5 followed by all of byte 6 and the msb of byte 7 yielding the 12 bit binary value 0b001111111100 or 0x3FC or decimal 1020. Again, converting to decimal millivolts requires dividing by 2 and adding 300. 1020 divided by 2 is 510 plus 300 equals 810.

So for our example part, the thermal diode measured from 691 up to 810 millivolts from 70C down to the final test temperature of -10C. This slope (lower temperature = higher voltage) is approximately 1.49 mV/degree C which is typical for these parts.

1.3.2.2 Scanning Thermal Diode Values for Core 1

The Thermal Diode values in Core 1 are scanned in the same way as for Core 0, but with the low order bit of the I2C address set to access the fuses in Core 1. Everything else except the I2C address should be handled and decoded in the exact same way as for Core 0.

Note: It is not necessary to scan out VDD values for Core 1, it is powered at the same voltage as Core 0.

Once you have scanned out the thermal diode data, it may be stored in system EEPROM for future use. The POR sequence can then be restarted from $\overline{\text{HRESET}}$ per the instructions in *Section 1.5.3* on page 24. Since the power and clocks have been already ramped, you can skip those steps and begin with asserting and releasing $\overline{\text{HRESET}}$ per the requirements of the Datasheet. It is not necessary to assert $\overline{\text{BYPASS}}$ unless you have also changed the PLL settings, in which case it should also be asserted to reset and relock the PLL.

Table 1-3. Thermal Diode Data Encoding

Field name	Ring Position	Length in bits	Adjustment	Expected Range
temp_low	2424:2435	7	Value - 40	-40 to 88C
temp_high	2463:2469	7	None	0 to 128C
voltage_low	2424:2435	12	Value / 2 + 300	300 to 2348 mV
voltage_high	2436:2447	12	Value / 2 + 300	300 to 2348 mV

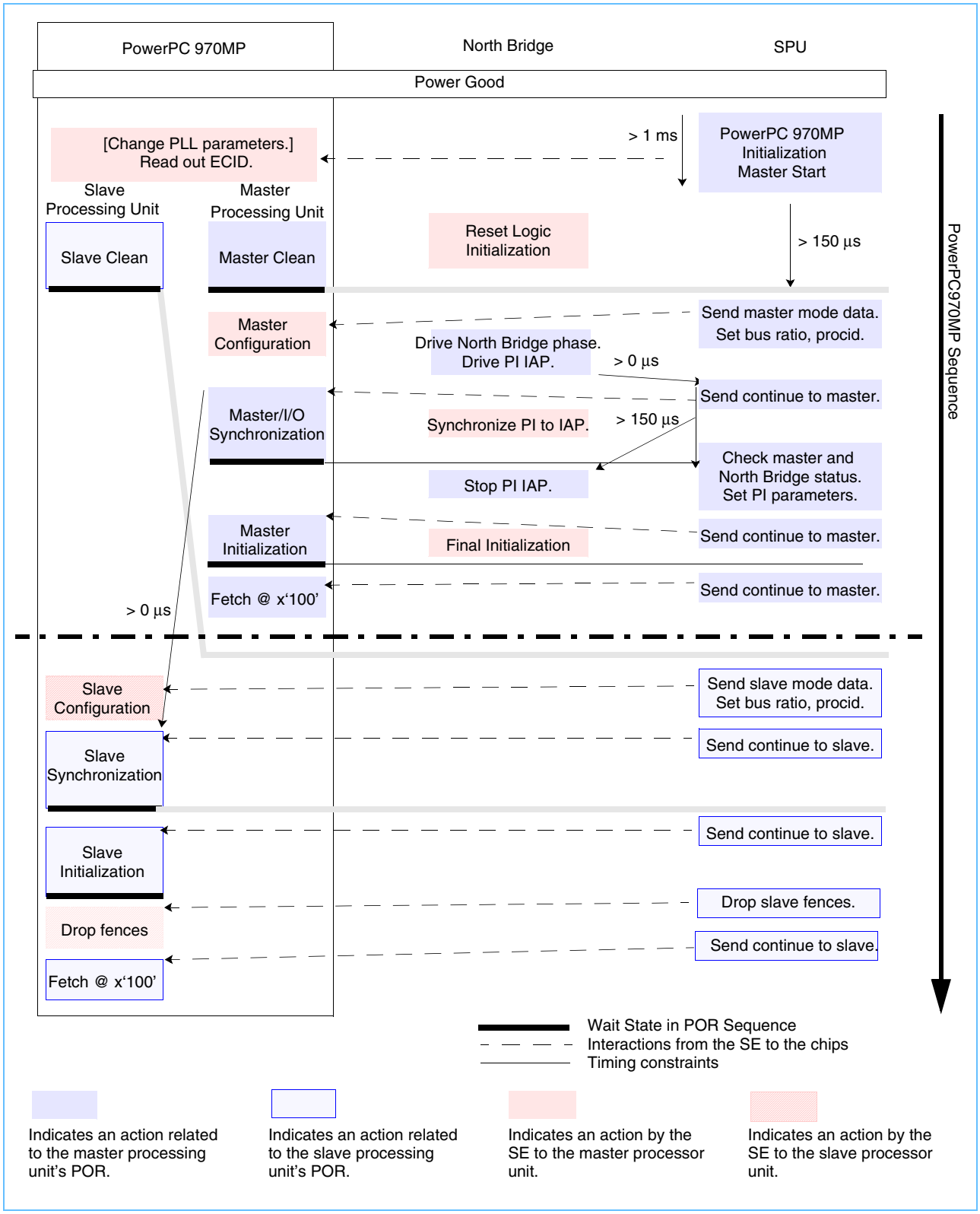
Note: All values are stored lsb:msb (bit reversed). They should be bit swapped before applying the adjustments shown in this table. Temperatures are stored in degrees C, voltages are stored in millivolts.

1.3.3 Service processor firmware bringup and development

During early bringup and firmware development, use of a I2C controller/emulator is recommended. These tools are available from multiple sources and convert a PC's RS232 port or USB port to I2C. Software on the PC can be used to communicate with the I2C bus in the system to develop and debug the power on reset sequence. Once the system has been brought up using the I2C 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 1-1. 970MP POR General Overview



1.4 POR 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 wait for them to complete.

After $\overline{\text{CP0\_HRESET}}$ is raised the program counter (APC) is cleared and the POR state machine first enters the PORWAIT state (see *Table 1-5* on page 21). The POR state machine will not advance until a continue is sent from the SPU. 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 JTAG or I2C. The POR program counter is then incremented and the state machine loops to the latch state.

The instructions used by the POR state machine are preset inside the 970MP. It is not expected that users will need to change this built in instruction sequence.

1.4.1 The 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 (e.g. 0x0000.0000) to SCOM address 0x40.0101. For more information on SCOM writes, see *Appendix A.4.2* on page 41

Note: The first continue command written to the 970MP after $\overline{\text{CP0\_HRESET}}$ will **not** generate an I2C acknowledge. The 2nd and all subsequent continue commands will receive the normal I2C acknowledge. This issue is included in the PowerPC 970MP Errata.

1.4.2 The POR Status Register

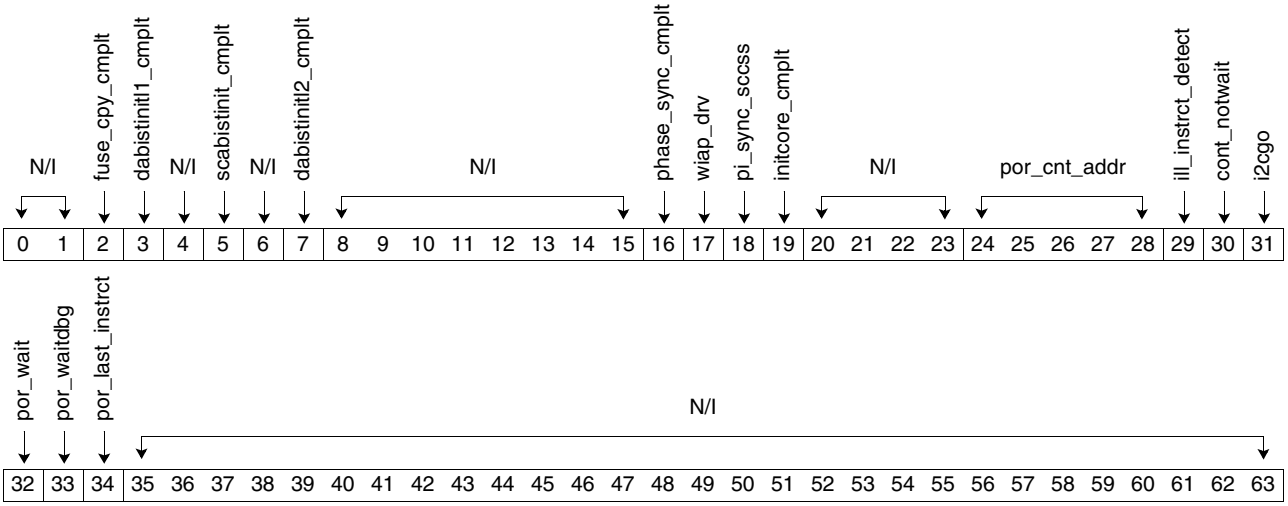
The POR status register is located at SCOM address 0x40.0000. This register indicates the current POR program counter in bits 24:28 and whether or not the current instruction has completed. For more information on SCOM read operations see *Appendix A.4.2* on page 41.

Address x‘400000’

Name Status Register

Type RO/WO
Clear entries are marked with an asterisk (*).

Reset Reset to all zeros during POR (inactive).



Bits	Field Name	Clear Entry	Description
0:1	N/I		Not implemented.
2	fuse_cpy_cmplt	*	Status is active after fuse copy completion.
3	dabistinitl1_cmplt	*	Status is active after DABISTINITL1 completion.
4	N/I		Not implemented.
5	scabistinit_cmplt	*	Status is active after SCABISTINIT completion.
6	N/I		Not implemented.
7	dabistinitl2_cmplt	*	Status is active after DABISTINITL2 completion.
8:15	N/I		Not implemented.
16	phase_sync_cmplt	*	Status is active after phase synchronization completion.
17	wiap_drv		Status is active while the writer initial alignment pattern (WIAP) is driven by POR.
18	pi_sync_sccss	*	Status is active after successful processor interface synchronization.
19	initcore_cmplt	*	Status is active after INITCORE completion.
20:23	N/I		Not implemented.
24:28	por_cnt_addr		Contains the current POR program counter address.

Bits	Field Name	Clear Entry	Description
29	ill_instrct_detect	*	Status is active after an illegal instruction is detected.
30	cont_notwait	*	Status is active if a continue was received while not in the Wait state.
31	i2cgo		Status is active after an I2CGO command until the next CONT command.
32	por_wait		Status is active when the POR state machine is in the Wait state.
33	por_waitdbg		Status is active when in the POR state machine is in the Waitdbg state.
34	por_last_instrct		Status is active when the POR state machine has reached the last instruction.
35:63	N/I		Not implemented.

1.4.3 Mode Rings

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 may need to be modified.

Most systems will use mode rings set to the default values provided in *Table 1-4* on page 19; 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 PowerPC 970MP boot ROM, plus the reset vector (0x100) should equal the physical address of the first instruction to be executed by the 970MP.

Initializing the mode ring is handled by the following procedure, also described in more detail in *Section 1.5.5.1* beginning on page 25. There are 2 basic steps:

1. A sequence of writes to the TAP controller. First the IR (Instruction Register), then the Data Register (DR) is loaded with the bits of the mode ring. Using I2C, you can send up to 8 bytes at a time.
2. Then a TAP Reset is sent to write the mode ring from the TAP controller.

The details of these steps, and their relationship to equivalent JTAG operations is documented in *Appendix A.3.2* on page 40.

The mode ring scan from I2C consists of a sequence of 64 bit writes to match the I2C TAP controller logic. Each block of 64 bits is written with a preamble command to the TAP controller to convert the I2C bytes to JTAG clock and data bits.

Table 1-4. Mode-Ring Content for Master and Slave Processing Units (for I2C) (Page 1 of 2)

64 Bit Burst	Bytes 1:8 for I2C	Notes
1	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	
2	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	
3	0x00, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	
4	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	
5	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, ***p1, p2***	Bytes Indicated by ***p1-p5*** should be replaced per *Table 1-7* on page 26 according to Bus Ratio and PLL Settings
6	***p3, p4, p5***, 0x01, 0x00, 0x00, 0x00, 0x00	Bytes Indicated by ***p1-p5*** should be replaced per *Table 1-7* on page 26 according to Bus Ratio and PLL Settings
7	0x00, 0x01, 0x00, 0x00, 0x00, 0x82, 0x82, 0x00	
8	0x00, 0x80, 0xE1, 0x00, 0x00, 0x04, 0x00, 0x04	
9	0x00, 0x65, 0xD8, 0x10, 0x00, 0x00, 0x00, 0x00	
10	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	
11	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	
12	0x00, 0x00, 0x00, 0x1F, 0x00, 0x00, 0x00, 0x00	
13	0x00, 0x00, 0x00, 0x00, 0xFC, 0xFF, 0xFF, 0xFF	
14	0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	

Table 1-4. Mode-Ring Content for Master and Slave Processing Units (for I2C) (Page 2 of 2)

64 Bit Burst	Bytes 1:8 for I2C	Notes
15	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00	
16	0x08, 0x00, 0x00, 0x00, 0x00, ***h1, h2, h3***	Bytes indicated by ***h1-h9*** should be replaced per *Table 1-6* on page 25 according to desired HIOR
17	***h4, h5, h6, h7, h8, h9,*** 0xFF, 0xFF	Bytes indicated by ***h1-h9*** should be replaced per *Table 1-6* on page 25 according to desired HIOR
18	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	
19	0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	
20	0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00	
21	0x00, 0x00, 0x54, 0x30, 0xC2, 0xA1, 0xAE, 0x00	
22	0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x40	
23	0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xEA	
24	0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	
25	0x00, 0x00, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00	
26	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x09, 0x48	
27	0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	
28	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	
29	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00	
30	0x00, 0x00, 0x00, 0x84, 0xA3, 0x1F, 0x00, 0x00	
31	0x00, 0x00, 0x00, 0x00, 0x60, 0x07, 0x00, 0x00	

1.5 Detailed view

Table 1-5 on page 21 shows the POR procedure in detail.The sequence shows a complete POR of Core 0 Followed by Core 1. Note that only Core 0 requires IAP. The POR instructions for TOGGLEWIAP and SYNCRIAP have no effect on Core 1 and behave like a NOP.

The sequence shown in *Table 1-5* on page 21 shows a typical scenario for starting Core 1 but others are possible. Core 1 may even be left unpowered until needed. Core 1 startup is usually much simpler than Core 0 - it doesn't require IAP since Core 0 actually handles that process. It is usually handled by CP1_HRESET followed by a series of continues until IPL2, then loading the slave mode ring, then another series of continues until Core 1 executes the Sreset fetch at HIOR + 0x100.

Note: For the purposes of this document, Master is assumed to refer to Core 0, and Slave is assumed to refer to Core 1.

Table 1-5. POR Procedure in Detail

POR Program Counter SCOM 0x40.0000 Bits 24:28	IPL Step	PowerPC 970MP Master (Core 0)	PowerPC 970MP Slave (Core 1)	North Bridge	SPU (GPULDBG must be pulled High)
--	IPL0 CHIP START	Power up PU0; detect chip type; initialize PLL;			Toggle CP0 and CP1 $\overline{\text{HRESET}}$, $\overline{\text{BYPASS}}$. Set PLL parameters;
00	IPL1 MASTER CLEAN	RSTFRL	IPL1 SLAVE CLEAN	Initialize North Bridge	Send continue, Core 0
01		[FULL] SCAN0			Send continue, Core 0
02		SAMPLEFUSE			Send continue, Core 0
03		SCABISTINIT			Send continue, Core 0
04		SCAN0			Send continue, Core 0
05		DABISTINITL1			Send continue, Core 0
06		DABISTINITL2			Send continue, Core 0
07		SCAN0			Send continue, Core 0
08		DABISTINITL2			Send continue, Core 0
09		SCAN0			Send continue, Core 0
10	IPL2 MASTER CFG	WAIT			
				Start WIAP, *psync*.	Send mode ring data and bus ratio to Core 0
					Send continue Core 0
11		DRIVEIOS			Send continue, Core 0
12	IPL3 MASTER SYNC	SYNCPHASE			Send continue, Core 0
13		STARTZIOCLK			Send continue, Core 0
14	IPL4 MASTER IOSYNC	TOGGLEWIAP			Send continue, Core 0
15		SYNCRIAP		Synchronize RIAP.	Start North Bridge processor interface synchronization.
16		WAIT			
				Stop WIAP.	Check or set processor interfaces; send continue, Core 0
17		TOGGLEWIAP			Send continue, Core 0

POR Program Counter SCOM 0x40.0000 Bits 24:28	IPL Step	PowerPC 970MP Master (Core 0)	PowerPC 970MP Slave (Core 1)	North Bridge	SPU (GPULDBG must be pulled High)
18	IPL5 MASTER INIT	STARTCORECLK		Final initialization.	Send continue, Core 0
19		INITGUS			Wait for 1 µS, send continue, Core 0
20		INITCORE			Send continue, Core 0
21		WAIT			Send continue, Core 0
					Final system initialization. Send continue. Core 0
22	IPL6 MASTER FETCH	STARTGUSCLK			Send continue, Core 0
23		INITGUS			Send continue, Core 0
24		SRESET			Send continue, Core 0
		<fetch @ x‘100’>			
--			IPL2 SLAVE CFG		Send 10 continues to Core 1. Send mode data and bus ratio to Core 1. Send continue, Core 1
--			IPL3 SLAVE SYNC		Send 5 continues, Core 1.
--			IPL4 SLAVE NOP		Send continue, Core 1
--			IPL5 PSLAVE INIT		Send 5 continues, Core 1
--					Unfence slave; send continue to Core 1
--			IPL6 SLAVE FETCH		Send 3 continues, Core 1

1.5.1 I2C Bus Speed

The 970MP bus speed on I2C will be limited to approximately 50Khz unless the service processor can write the value 0x0083F000.00000000 to the SCOM address 0x60.0400. SCOM writes are covered in detail in *Appendix A.4.2 SCOM Register Read/Write* on page 41.

Note: This SCOM write must occur after the 2nd continue is sent during POR. Until this SCOM write occurs, the I2C bus speed will be limited to 50Khz. Once the SCOM has been written I2C bus speed may be increased to 100Khz.

1.5.2 Power and Clocks

1.5.2.1 Power and Clock Ramping for PowerPC 970MP

The voltage regulator for the 970MP should be set to the value indicated by the PowerPC 970MP Datasheet for the encoded VFC values scanned out per the process detailed in *Section 1.3.2* beginning on page 11.

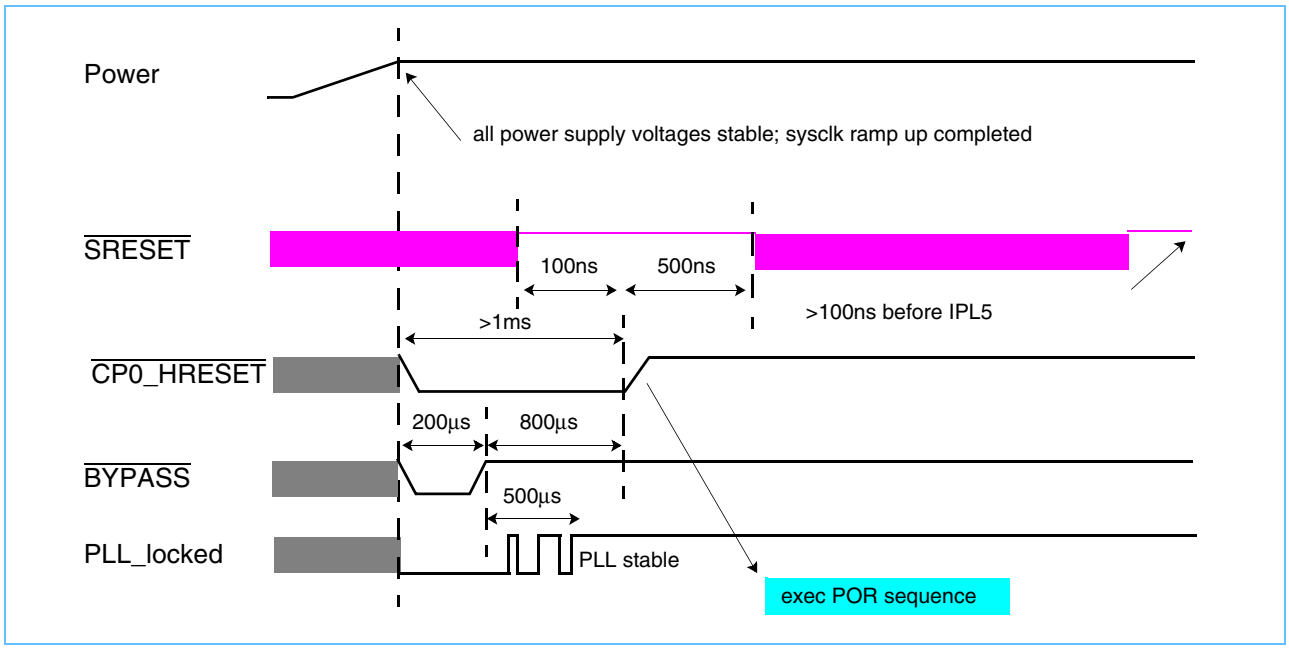
In the PowerPC 970MP the $\overline{\text{SRESET}}$ pin should be held high around the release of $\overline{\text{CP0\_HRESET}}$ as depicted in *Figure 1-2*. Failing to conform to this specification will result in asynchronous clocks on the 970MP Bus when doing frequency switching using power tuning.

When the PLL is in bypass-mode, where the $\overline{\text{BYPASS}}$ pin is held low, the $\overline{\text{SRESET}}$ pin should also be held low instead of high around the release of $\overline{\text{CP0\_HRESET}}$. This prevents the initial clock alignment procedure (CAP).

On bring-up boards, the polarity of $\overline{\text{SRESET}}$ around the release of $\overline{\text{CP0\_HRESET}}$ should be selectable to allow CAP skipping in case of hardware problems.

Once the power supply is stable and the clock generator is running, the service processor should assert $\overline{\text{BYPASS}}$ for 200 microseconds, while also asserting $\overline{\text{CP0\_HRESET}}$ for 1ms. This will reset the PLL and allow it to lock during the remaining 800uSec required for $\overline{\text{CP0\_HRESET}}$ to complete. After $\overline{\text{CP0\_HRESET}}$ goes high the preprogrammed power-on sequence begins with IPL0.

Figure 1-2. $\overline{\text{CP0\_HRESET}}$, $\overline{\text{SRESET}}$, $\overline{\text{BYPASS}}$ timing for the PowerPC 970MP



1.5.3 IPL0

1.5.3.1 Release $\overline{\text{CP0\_HRESET}}$

Once $\overline{\text{CP0\_HRESET}}$ is released and the first continue command is issued to the 970MP, IPL0 starts. During IPL0 all the free running logic is initialized by scanning 0s into all of the RAS latches. This corresponds to Step 0 (RSTFRL) in *Table 1-5* on page 21.

Issuing a continue command will move the instruction pointer to step 1, which will begin scanning 0s into all the rings. This corresponds to Step 1 ([FULL] SCAN0) in *Table 1-5* on page 21.

Note: The first continue command written to the 970MP after $\overline{\text{CP0\_HRESET}}$ will **not** generate an I2C acknowledge. Subsequent continue commands will get the normal I2C acknowledge. This is listed in the PowerPC 970MP Errata.

1.5.3.2 Scanning of Boundary Scan Latches

One of the rings initialized by step 1 is the boundary scan latches. During this step, some of the 970MP 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 initialization most of the North Bridge logic is probably still being initialized so this should not pose a problem.

It's important to note that one of the pins that may become active is $\overline{\text{QREQ}}$. This pin is used to indicate a request to sleep. The SPU should avoid creating a situation where the potential toggling of $\overline{\text{QREQ}}$ or other output pins may 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 will be impossible to predict.

After another continue command is issued, the fuses are copied into their latches. This corresponds to step 2 in *Table 1-5* on page 21.

1.5.4 IPL1

1.5.4.1 Chip Initialization

IPL1 begins by initializing the arrays. This is done using the ABIST engine to initialize the core, gus, vector unit, and pervasive arrays. 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.

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).

This phase takes approximately 12 ms. During this phase the service processor can also begin initializing the North Bridge and getting it ready to start the IAP after the mode rings have been scanned in.

1.5.4.2 Verifying Chip initialization in IPL1 is complete

The SPU can check the status of IPL1 by reading the POR status register at SCOM 0x40.0000. When bits 24:28 of this register (the POR Program Counter) contains the value 0x0a, the chip is ready to start IPL2.

1.5.5 IPL2

1.5.5.1 Load Mode Ring

Now that the chip has been initialized in IPL1, we are ready to start configuration. The POR state machine enters the WAIT instruction until the mode ring has been loaded. The mode ring contents are documented in *Table 1-4* on page 19 for DD 1.x.

1.5.5.2 Mode Ring Customization For HIOR

Certain fields in the mode ring may need to be customized for your application. In particular, the HIOR must be initialized correctly. The HIOR 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 1-4 Mode-Ring Content for Master and Slave Processing Units (for I2C)*. The bytes to be replaced by your desired HIOR value are indicated by ***h1-h9*** as shown in *Table 1-6* on page 25. Each byte must also be bitswapped (bit0 = bit7, bit 1= bit6 etc.) for I2C. The least significant 2 bits of HIOR are stored in the most significant 2 bits of ***h1***, followed by the rest of HIOR bits from lsb to msb in ***h9***.

Table 1-6. Mode Ring Customization for HIOR

HIOR Byte	Details
h1	0b'xx111111' - xx= HIOR(0:1)
h2	HIOR(2:9)
h3	HIOR(10:17)

Table 1-6. Mode Ring Customization for HIOR

HIOR Byte	Details
h4	HIOR(18:25)
h5	HIOR(26:33)
h6	HIOR(34:41)
h7	HIOR(42:49)
h8	HIOR(50:57)
h9	0b'11xxxxxx' - xxxxxx=HIOR(58:63)
Note: All bytes of HIOR must be bitswapped (bit7=bit0, bit6 = bit1 etc.) for I2C format. h1 and h9 should be formatted with lsb in msb position.	

1.5.5.3 Mode Ring Customization for PLL Multiplier and Bus Ratio

Systems planning to use the power tuning F/2 and F/4 modes must also modify the mode ring to match the system bus ratio and PLL mode. Use *Table 1-7* on page 26 to determine the correct byte sequence to insert into the mode ring. The bytes in that table replace the bytes in *Table 1-4* on page 19.

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 tune features. Incorrect configuration of these bytes will not affect full frequency operation but will cause bus errors during power tune (F/2, F/4) operation.

Table 1-7. Mode-Ring Content Dependent on the Bus Ratio and PLL Settings

Bus Ratio	PLL x8	PLL x12	PLL x24
1:1	Not Supported	Not Supported	Not Supported
2:1	0x00, 0xA0, 0x00, 0xC1, 0x00	0x00, 0xA0, 0x00, 0xC1, 0x00	Not Supported
3:1	0x00, 0xA0, 0x22, 0x60, 0x00	0x00, 0xA0, 0x22, 0x60, 0x00	Not Supported
4:1	0x00, 0xA0, 0x00, 0xC1, 0x00	0x00, 0xB0, 0x00, 0xC1, 0x00	Not Supported
6:1	0x00, 0xB0, 0x22, 0xE3, 0x01	0x00, 0xB0, 0x22, 0xE3, 0x01	Not Supported
8:1	Not Supported	Not Supported	Not Supported
12:1	0x00, 0xA0, 0x08, 0xC9, 0x04	0x00, 0xB0, 0x08, 0xC9, 0x04	Not Supported
24:1	Not Supported	Not Supported	0x00, 0xA0, 0x34, 0xC9, 0x04

1.5.5.4 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 I2C writes. TAP commands take the form of writes to the reserved scom address 524xxx. The format of TAP commands is covered in *Appendix A.4.3* on page 44.

Writing the mode ring begins by setting the JTAG logic to SHIFT-IR mode. This is accomplished by writing the I2C byte sequence 0x08 0x40, 0x52, 0xDF, 0x00.

The next TAP command sets the ring address of the mode ring, for 970MP this is 0xC08000. This is accomplished by writing the I2C byte sequence 0xDE, 0x40, 0x52, 0x00, 0x80, 0xC0, 0x0F.

The next TAP command sets up the SHIFT-DR, this is handled by the I2C byte sequence 0x02, 0x40, 0x52, 0x03.

1.5.5.5 Writing Mode Ring Data

Generally, the mode ring data is then written in a series of 64 bit bursts using the TAP controller via I2C. Each 64 bit write begins with the preamble 0xBE, 0x40, 0x52, then is followed by 8 bytes of mode ring data.

The mode ring contents are listed in sequential byte order in *Table 1-4 Mode-Ring Content for Master and Slave Processing Units (for I2C)*. This table contains the bytes to be sent via I2C for both cores.

Send the mode ring data to each core by writing the preamble bytes above, (0xBE, 0x40, 0x52) followed by 8 bytes from *Table 1-4* on page 19. Continue sending these 64 bit bursts of data until you have sent every byte in the mode ring. Then send the TAP Reset as shown in *Section 1.5.5.6* on page 27.

1.5.5.6 Issuing Reset to the TAP Controller

A TAP Reset is sent to write the mode ring from the TAP controller by writing the I2C sequence 0x03, 0x40, 0x52, 0x1F. Once this reset has been issued, the entire mode ring has been transferred into the 970MP. This must be done at the end of each ring.

1.5.5.7 Start IAP and Phase

Once the mode ring has been loaded, then North Bridge should begin sending WIAP pattern. At this point the North Bridge starts sending the synchronization pattern to the 970MP.

1.5.6 IPL3

1.5.6.1 Sync of all PowerPC 970MPs to psync

In this phase the 970MP begins synchronizing to the external PSYNC signal driven by the system to indicate the correct “time zero” reference for bus operation and snooping.

1.5.6.2 Send Continues

At this point the service processor sends 4 “continue” commands to the 970MP to start IPL4. The continue command is triggered by writing any 64 bit value (for example, all 0s) to SCOM address 0x40.0101. At this point the POR status register bits 24:28 should read a value of 0x10, indicating instruction 16.

1.5.7 IPL4

1.5.7.1 Wait for IAP to Complete

The 970MP(s) start driving the elastic interface alignment pattern (TOGGLEWIAP) and proceed to the synchronization of their PI receivers (SYNCRIAP). The North Bridge can then start synchronization of its PI receivers. This process is expected to complete within 20 ms.

1.5.7.2 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. These registers are documented in the 970MP System Manual in the I/O SCOM Registers section.

1.5.7.3 Initialize PI Parameters

Once the IAP has been completed successfully, the service processor will then initialize the PI bus for correct operation by configuring the STATLAT, SNOOPLAT, COMPACE and SNOOPACC parameters. The Target cycles should also be configured for the North Bridge. These registers are documented in the 970MP System Manual in the I/O SCOM Registers section.

Table 1-8. Recommended STATLAT, SNOOPLAT, SNOOPACC, COMPACE for 970MP *with CPC925*

970 MP Target Cycle	CPC 925 Target Cycle	970MP STATLAT	970MP SNOOPLAT	970MP COMPACE	970MP SNOOPACC
1	0	19	6	4	15
Note: SNOOPACC values are bus values, not register encodings. To get correct register encodings, subtract 8					

Table 1-9. Recommended STATLAT, SNOOPLAT, SNOOPACC, COMPACE for 970MP *with CPC945*

970MP Target Cycle	CPC 945 Target Cycle	970MP STATLAT	970MP SNOOPLAT	970MP COMPACE
TBD	TBD	TBD	TBD	TBD

1.5.7.4 Stop IAP Pattern

With the Processor Interface bus initialized and ready for operation, another SCOM write to Processor Interface Mode Register (SCOM address 0x04.6A00) will halt the IAP pattern and quiesce the bus. This should also be done for the North Bridge.

1.5.7.5 Send Continue

At this point the service processor sends 3 “continue” command to the 970MP to start IPL5.

1.5.8 IPL5

1.5.8.1 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. Then the SPU finishes the system initialization and sends 7 more continue commands to the 970MP.

Some system implementation will also need to make the boot ROM available to the processor’s memory map. This may 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 970MP to IPL6.

1.5.9 SCOM Accesses required to workaround prefetch filter queue overrun errata

All versions of 970MP have an errata related to the use of certain forms of the **dcbt** instruction. This errata can cause a system hang unless these instructions are serialized. The instruction match CAM is used to implement this serialization as follows:

Write SCOM address 0x023300 with 0x82FFFDD3.F00001D6, then write the same address with 0x7FFFFA2D.F00001D7. This initializes the instruction match CAM to serialize the affected **dcbt** instructions. Then write SCOM address 0x23401 with the value 0x00000000.00000002. This sets the Patch Map register to indicate to performance monitoring software that the instruction match CAM 6/7 should not be used.

1.5.10 SCOM Accesses required to workaround Larx spin livelock errata

All versions of 970MP have an errata related to the use of the lwarx instruction. This errata can cause a live-lock condition between processor cores unless this workaround is implemented.

Set bit 55 of SCOM 0xA8000 as 0b'1'. This will back-off retries for a longer period before driving out a new command on the bus. The performance impact is minimal.

1.5.11 SCOM Accesses required to workaround false BIU FIR error DD 1.0x

DD 1.0x of 970MP may generate a false bus FIR If the cores are allowed to nap independently. This bus FIR can be masked by setting bit 55 of SCOM 0xA0400 to 0b'1'. This workaround is not required for DD 1.1x or later revisions.

1.5.12 IPL6

1.5.12.1 Start Gus Clock

The last step, Fetch Init (IPL6), starts the STS clock, resets the storage interface of the 970MP and starts fetching instructions at address HIOR + 0x100. This requires 2 more continue commands to complete IPL6.

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 will start execution with memory translation off (real address mode, effective address = physical address), and with caches disabled. Some system configurations will also need other prefetching or superscalar features disabled to allow correct operation.

1.6.1 Initial HID and MSR State

The processor's HID registers and MSR will be initialized by the mode ring as shown in *Table 1-10* on page 30. Boot code can modify the HIDs and MSR as needed. When modifying HID registers, follow the recommended process as described in *Table B.1* on page 48

Table 1-10. Initial HID and MSR State after POR

Register	Initial Value	Notes
HID0	0x0000.0000.8008.8000	Can be modified via mode ring See *Appendix C-1 PowerPC 970MP Mode Ring Data Table in I2C Format (DD 1.x)* on page 59, Burst 21
HID1	0x0000.0000.000F.D3C2	Can be modified via mode ring See *Appendix C-1 PowerPC 970MP Mode Ring Data Table in I2C Format (DD 1.x)* on page 59, Burst 30
MSR	0x0000.0000	Can be modified via mode ring. See *Appendix C-1 PowerPC 970MP Mode Ring Data Table in I2C Format (DD 1.x)* on page 59, Burst 16.
HID4	0x0000.0000.0000.0000	Can be modified via mode ring. See *Appendix C-1 PowerPC 970MP Mode Ring Data Table in I2C Format (DD 1.x)* on page 59, Bursts 27, 28.
HID5	0x0000.0000.0000.0000	Can be modified via mode ring. *Appendix C-1 PowerPC 970MP Mode Ring Data Table in I2C Format (DD 1.x)* on page 59, Burst 28.

1.6.2 Automatic Array Recovery

The 970MP 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 1-11* on page 31 and *Table 1-12* on page 31 allows the 970MP to recover from array soft errors automatically.

Note: The checkstop enables for the L2 UE (Uncorrectable Error) 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 970MP should be set to checkstop.

Table 1-11. Enabling Automatic Array Recovery Modes, By Array

Array/Error Source	HID Register Setting	Error Mask Register (SCOM 0x30400)	Machine Check Register	Checkstop Register
I-Cache / ITAG	HID1(11:12)=11	0x030400(0:2)=000	0x030901(0:2)=000	0x30800(0:2)=000
I-ERAT	HID1(13)=1	0x030400(3)=0	0x030901(3)=0	0x30800(3)=0
L2 UE	N/A	0x030400(4)=0	0x030400(4)=0	0x030400(4)=1
Logic UE	N/A	0x030400(5)=0	0x030400(5)=0	0x030400(5)=1
D-Cache	HID5(50)=0 HID4(39:41)=000	0x30400(6)=0	0x030901(6)=0	0x30800(6)=0
DTAG	HID4(34:36)=000	0x30400(7)=0	0x030901(7)=0	0x30800(7)=0
D-ERAT	HID4(29:31)=000	0x30400(8)=0	0x030901(8)=0	0x30800(8)=0
LSU-TLB	HID4(44:48)=00000	0x30400(9)=0	0x030901(9)=0	0x30800(9)=0
LSU SLB	HID4(53:54)=00	0x30400(10)=0	0x030901(10)=0	0x30800(10)=0

Table 1-12. L2 Array Recovery Details

L2 Error	SCOM (0x43000)	Error Mask Register (SCOM 0x40401)	Checkstop Register (SCOM 0x40801)	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)=0b000	0x040801(49:51)=0b111	
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	

1.7 Debugging Tips

1.7.1 Verifying I2C Operation

Some microcontrollers used as SPUs for 970MP will require level shifters to avoid levels on the 970MP pins beyond OVDD.

Note: I2C pins should never exceed voltages beyond the OVDD tolerances provided in the PowerPC 970MP Datasheet. Devices may be permanently damaged if inputs exceed allowable maximum voltages.

Proper I2C operation can be confirmed by attempting to read the POR status register at SCOM address 0x40.0000. This register should be initialized to 0 after $\overline{\text{CP0\_HRESET}}$ is released and should increment after sending a continue command.

1.7.2 SCOM access to Uninitialized Units

If SCOM access is attempted to units before clocking is enabled, the SCOM hardware can be disabled accidentally. Before attempting any SCOM access, make sure that you have completed 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).

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 $\overline{\text{CP0\_HRESET}}$.

1.7.3 Use of $\overline{\text{SRESET}}$

A POR sequence that causes an error is likely to require a complete restart from $\overline{\text{CP0\_HRESET}}$ and $\overline{\text{BYPASS}}$. Unlike earlier PowerPC processors, the 970MP cannot usually be restarted via $\overline{\text{SRESET}}$. The $\overline{\text{SRESET}}$ 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.

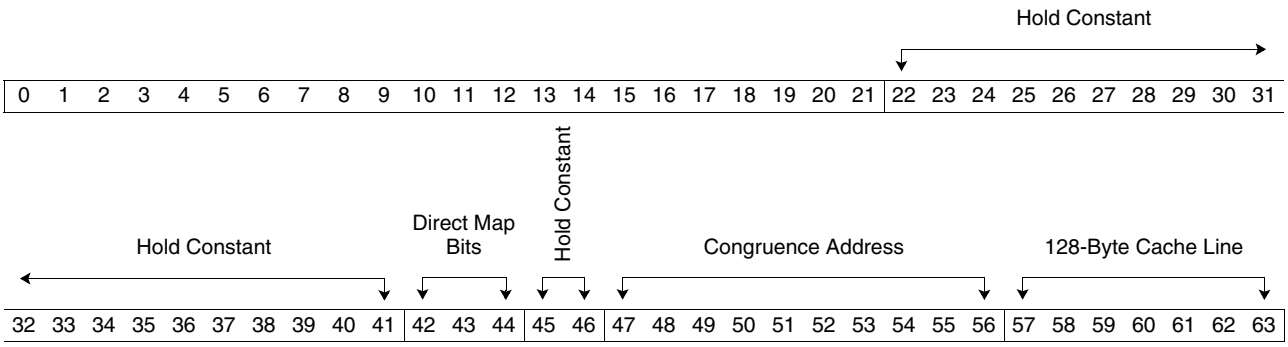
However, once a system is running it should be restartable via assertion of $\overline{\text{SRESET}}$. For example, you should be able to recover from most software crashes or kernel panics by asserting $\overline{\text{SRESET}}$, provided the boot code is still intact in memory.

1.7.4 Diagnosing IAP errors

Refer to the PowerPC 970MP System Manual section on I/O SCOM registers to help diagnose problems with the IAP portion of power on reset.

1.8 L2 Cache Flush Algorithm

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 (HID1[7:8] = ‘00’).
4. Disable data cache (D-cache) prefetch (HID4[25] = ‘1’).
5. Flash invalidate the D-cache (HID4[28] = ‘1’).
6. Execute a **sync** instruction.
7. Disable the D-cache (HID4[37:38] = ‘11’). This will guarantee that all loads are visible to the L2.
8. Set the L2 to direct-mapped mode. This can be done by the service element or through the SCOM control (SCOMC) and SCOM data (SCOMD) special purpose register (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 47 - 56) of the load address, and repeat step 11.
13. Repeat step 12 for all 1024 congruence address values.

To power down after performing this sequence, the processor executes an ATTN instruction to enter the quiescent state. Once a processing unit has been flushed, it should be fenced if the intent is to power down that processing unit. This helps avoid snooping and hang problems.

To return to normal processing after performing this sequence, set the L2 to set-associative mode. Enable the D-cache, prefetching, data address translation, and interrupts, as desired.

Appendix A. I2C Interface

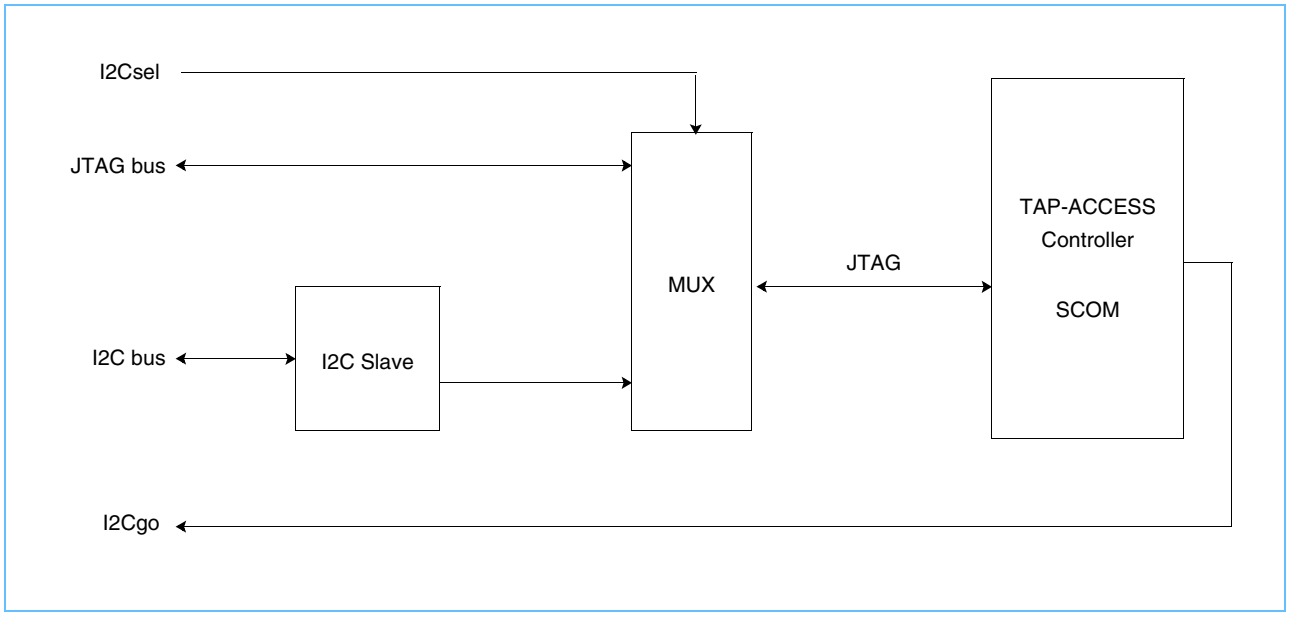
In the PowerPC 970MP, each processing unit has its own independent inter-integrated circuit (I2C) interface. The base address of each interface is derived from the processor ID of the processing unit (the 3-bit ProcID field) and is ‘1000’.

A.1 I2C Purpose

The I2C is a standard bus developed by Philips Electronics.[1] The I2C slave described in this documentation 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 A-1 shows the basic concept of merging the off-chip I2C and JTAG buses.

Figure A-1. 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 data 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 functions: 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 TAP and optionally uses this as part of the decision to acknowledge data or as the slave address depending upon the read/write

1. For more information, see http://www.semiconductors.philips.com/

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). (For more information, see the IR Status Read example on page 45.)

A.1.1 *I2Csel* and *I2Cgo* Pins

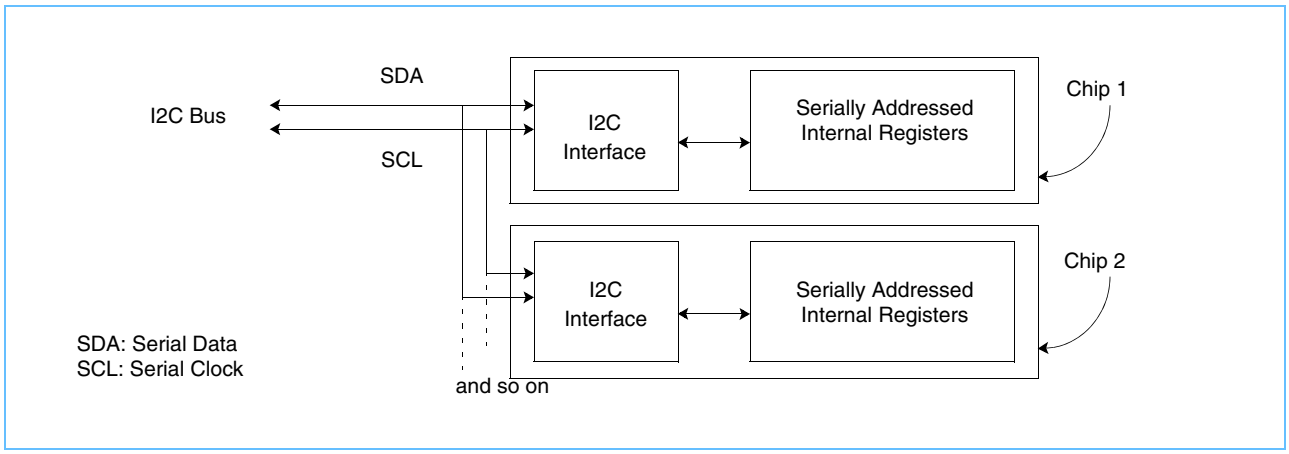
The *I2Csel* input pin is used to select either JTAG access to the processor, if low, or I2C access to the processor, if high. The *I2Cgo* open collector pin can be used to prevent access collisions between JTAG and I2C as described in section 2.5.2 of the PowerPC 970MP System Manual.

A.1.2 I2C Protocol

The I2C protocol, as defined, supports only register read and write operations. This specification only deals with 7-bit slave addressing mode, which allows 2^7 slaves to be directly addressed. The protocol, illustrated in *Figure A-2*, implies that an addressed slave responds to a read or write by addressing successive bytes serially, starting in memory where it is defined by the particular slave.

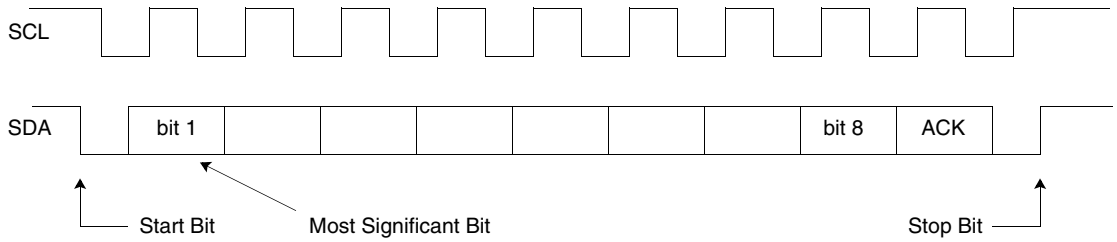
Implementation Note: This I2C-to-JTAG slave implementation extends direct addressing to 2^{23}, operating on 8-byte aligned registers. In addition, operations other than register reads and writes that are supported by the access implementation of JTAG are available.

Figure A-2. I2C Protocol



A.2 I2C Bus Operation

Figure A-3. I2C Bus Operation



As shown in *Figure A-3*, the I2C bus consists of two wires: serial data (SDA) and serial clock (SCL). These stay 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 I2C operates on packets; each packet is 1 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 *I2C Bus Specification* from Philips 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 temporally pace the operation by holding SCL low following the ACK bit. The SCL clock rate slows to the speed of the slowest master/slave attached. For further information on how this is accomplished, see the *I2C Bus Specification* from Philips Semiconductors.

Packet bits are transmitted most significant bit (msb) first (that is, bit 1 is transmitted first). Bits 1 - 7 of the start byte address a particular slave chip. Bit 8 of the start byte is a read/$\overline{\text{write}}$ 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 data flow reverses after the slave ACKs 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.

A.2.0.1 I2C Bus Speed

The 970MP 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. SCOM writes are covered in detail in *Appendix A.4.2 SCOM Register Read/Write* on page 41.

Note: This SCOM write must occur after the 2nd continue is sent during POR. Until this SCOM write occurs, the I2C bus speed will be limited to 50Khz. Once the SCOM has been written I2C bus speed may be increased to 100Khz.

A.2.1 Deviations from I2C Standard

These are deviations, not limitations. If necessary, these features could be implemented.

- Does not support 10-bit addressing mode.
- Does not support general call address.
- Does not support auto-increment of the slave byte address. The power-on reset (POR) default byte address is x‘000000’ (an SCOM address, as implemented by Access).

A.2.2 JTAG Overview

The Boundary Scan/JTAG, formally known as IEEE standard 1149.1, is a set of design rules that facilitate testing, device programming, and debug at the chip, 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 the 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 output from Access.
test logic reset (TRST)	TRST causes an asynchronous reset of the test logic (TAP transitions to TestLogicRst).

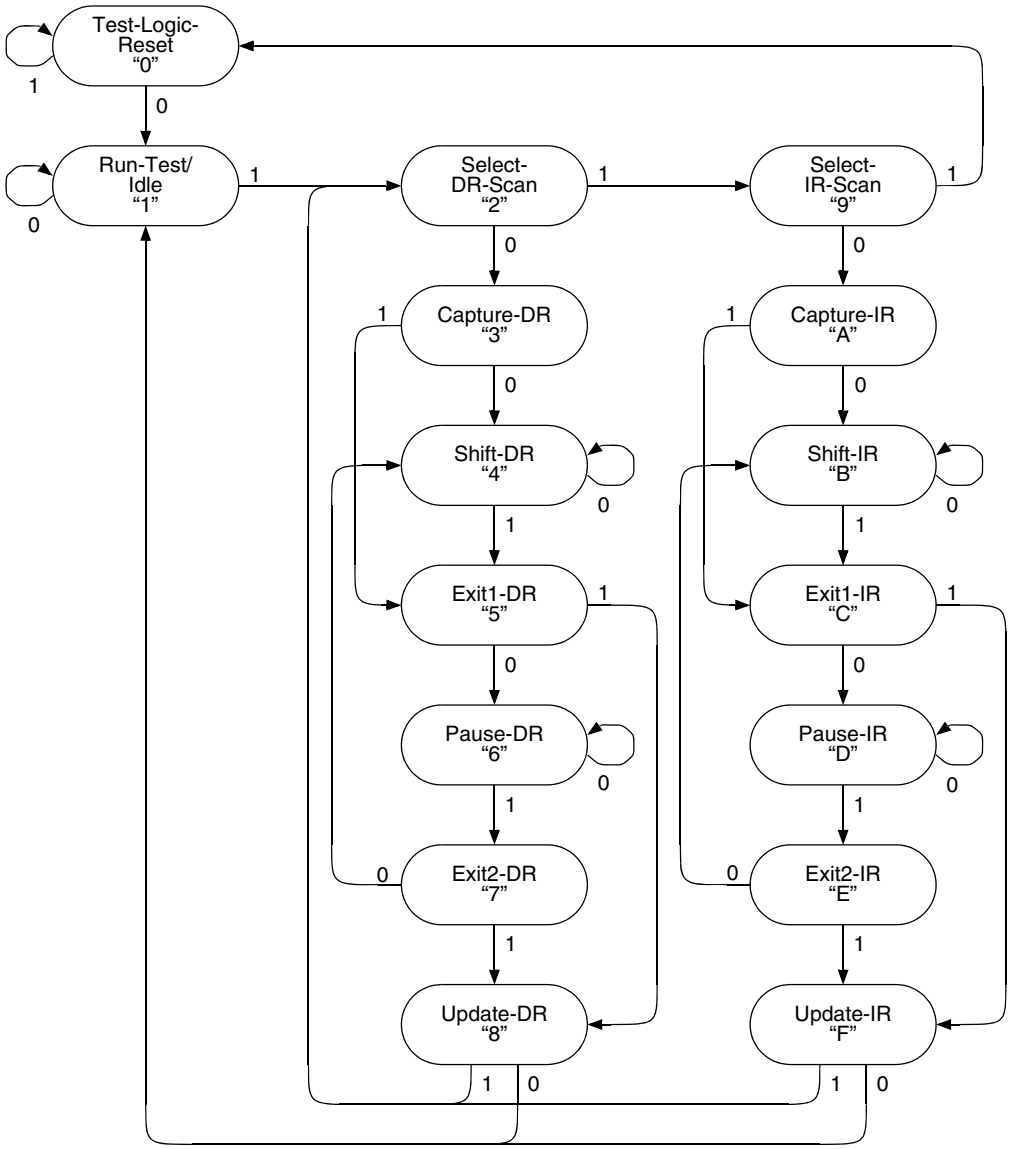
A.2.2.1 TAP Controller

Figure A-4 on page 38 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 the 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 TCK causes another bit to be shifted into the Instruction Register from TDI and 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 to TDO.

The contents of the Instruction Register determine the specific TDR addressed, or non-register operation to be performed.

Figure A-4. IEEE/Access TAP Controller

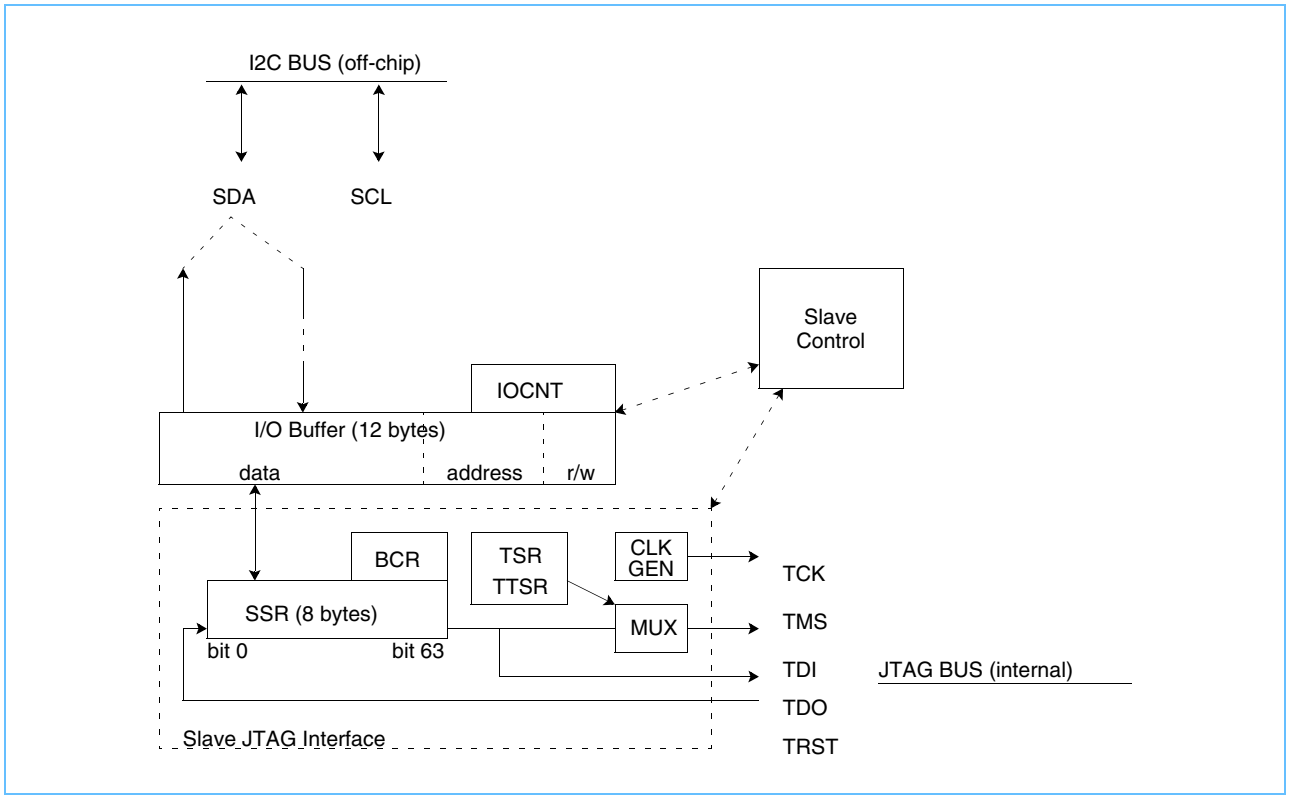


Note: The numbers in quotation marks are the state numbers used in the state machine implementation. The numbers on the arrows represent the state of the TMS line leading to the transition.

A.3 Slave Implementation

Figure A-5 outlines the major registers involved in data and control flow.

Figure A-5. Major Registers Involved in Data and Control Flow



A.3.1 Slave Data Flow Overview

The physical Interface handles I2C protocol and timing. It provides a 1 byte wide data interface along with associated handshaking signals. *SDA* is the bidirectional serial data signal defined by I2C. *SCL* is the serial clock signal defined by the I2C. Detailed operation of the physical interface is not discussed in this specification, since it is closely related to the implementation of the I2C standard.

The I/O buffer (IOBUF) buffers successive bytes to be transferred to or from the chip addressed as the slave chip by the I2C 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 I2C to the TDI. During read operations, the dataflow is from the TDO to the I2C.

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 the IEEE 1149.1 protocol. TDI is the serial data in to JTAG. TDO is the serial data out of JTAG.

A.3.2 TAP Command Bus

The TAP command (*TAPCMD*) signal is 9 bytes wide. The first byte, described below, is a primitive command to be performed by the JTAG scan logic.

TMS Register (*TSRL2*: TAPCMD bit 0) determines whether SSR data or a constant value will be shifted out to TMS. A '0' indicates that the data in SSR is a stream of TMS control information. A '1' selects a static '0' value driven on TMS of the first "N-1" TCK clock pulses. The TTSRL2 value is then driven on TMS during the Nth (terminal) TCK pulse.

Terminal TMS Register (*TTSRL2*: TAPCMD bit 1) provides a value on TMS during the Nth (terminal) TCK pulse. It is intended for long scanning operations when the 64-bit width of the SSR is insufficient to complete the scan. TTSRL2, when set to '0', will bridge primitive operations, allowing the scan to continue with another primitive command. TTSRL2, when set to '1', terminates the operation by moving the JTAG TAP state inside Access out of the ShiftDR or ShiftIR state.

Test Logic Reset (*TRST*) is formed by gating TAPCMD bits 0 - 1 in a combination not otherwise used for primitive commands. This output is latched to de-glitch TRST.

Bit Count Register (*BCRL2*: TAPCMD bits 2:7) defines the number of TCK pulses to send, along with a sequence of information on TMS or TDI. A Bit Count Register (BCR) value of '111110' indicates that 64 TCK pulses are to be sent. The entire SSR register will be right-shifted out of TDI or TMS, shifting TDO back into the SSR on the left. BCR set to '111111' is the terminal count, indicating that one TCK pulse is to be released.

A.3.3 Primitive Command Summary

Table A-1 summarizes the *TAPCMD* signal. Bits 8 - 71 of the *TAPCMD* signal represent data to be loaded into the SSR in preparation for shifting out TDI or TMS.

Table A-1. TAP Primitive Command Interface (Page 1 of 2)

Bit	Signal	Description
0	TSRL2	TMS Select Register 0 TMS ≤ (SSR(63), shift right) × (BCRL2 + 2) 1 TMS ≤ '0' while BCRL2 ≠ '111111' TTSRL2 when BCRL2 = '111111'
1	TTSRL2	TMS Terminal Select Register Value driven on TMS during the last bit of BCRL2 count when TSRL2 = '1'.
	TRST	TRST ≤ (TSRL2 = '0') and (TTSRL2 = '1') This is a way to get TRST function without using more bits.
2:7	BCRL2 (6 bits)	Bit Count Register Load with the number of bits to shift minus two: 111110 = 64 TCK pulses (SSR shifted 63 times) 111101 = 63 TCK pulses (SSR shifted 62 times) • • • 111111 = one TCK pulse (SSR will not shift) **Note:** There is no encoding for zero pulses.

Table A-1. TAP Primitive Command Interface (Page 2 of 2)

Bit	Signal	Description
8:71	SSRL2 (64 bits)	Simple Scan Register Data to be shifted out of TDI (also out of TMS if TSRL2 = '1'). TDO ≥ (0)SSRL2(63) ≥ TDI/TMS TDO is captured on the rising edge of TCK. TDI and TMS are launched (SSR is shifted) on the falling edge of TCK.

A.3.4 CMDDONE - TAPCMD done, JTAG Scan Logic idle

When the BCR reaches '111111', the command is done. The ONECLKL2 latch returns to '0', and the TAPCMD state machine rests in idle. The internal JTAG bus is inactive. This signal is fed back to the I2C slave state machine as an indicator that a new primitive command can be initiated.

A.4 Programming and Examples

A.4.1 Considerations for Concurrent Resource Use by I2C and JTAG

When the on-chip scan resource (Access) will be shared by the I2C 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 I2C slave while the other chip is active.
- Neither I2C nor JTAG have the concept of atomic streams of commands; some operations may fail if the command stream is interlaced.
- The *I2Csel* pin is used to select if the I2C macro or the external JTAG interface has control over the internal JTAG engine of the chip.

A.4.2 SCOM Register Read/Write

Table A-2 describes the command format that will perform a 64-bit SCOM read/write.

Table A-2. SCOM Register Read/Write (bit numbering, lsb = '0')

Start Byte		SCOM Address			SCOM Data (I/O buffer)							
I2C Slave Address (7 bits)	R/$\overline{W}$	LSByte	MidByte	MSByte	8 bytes (LSByte, MSbit first)							
7:4 970MP base address 3:1 ProcID	'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 (I2C slave base address), optionally followed by up to 3 bytes of SCOM address (LSByte, high-order bit first).[1] Note that 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 initiates the SCOM operation.[3] The SCOM address is not auto-incremented.[4] Writing more than 8 bytes of data will result in the data at the same SCOM address being overwritten.[5] 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.

Read: For an SCOM read operation, a restart (a stop followed by a start) is then issued on the I2C interface with the R/W bit set to ‘1’. This triggers the initialization of an SCOM read within the I2C hardware. The data-flow then reverses, and the slave returns 1 data byte for each acknowledgment (LSByte first). The SCOM address is not auto-incremented. If more than 8 bytes are read, the data will be repeated starting with the LSByte.

TapCmd Example 1: SCOM READ of register address x'010203'

The following sequence of I2C commands is needed to perform an SCOM read of register address x'010203', assuming the I2C slave address is ‘1000000-' (the last bit, bit 8, is the R/W bit) and the data returned is x‘AABB CCDD':

wait for I2C bus idle	SCL equals '1', SDA equals '1'	
master start bit		
master byte	x'80'	The slave ACK equals '0'; indicating recognition of its own address; bit 8 equals '0', indicating a write to the slave.
master byte	x'03'	The slave ACK equals '0'; the address is written to the slave, least significant byte first.
master byte	x'02'	The slave ACK equals '0'; the address is written to the slave, middle byte.
master byte	x'01'	The slave ACK equals '0'; the address is written to the slave, most significant byte. The master stop bit waits for the I2C bus idle master start bit.
master byte	x‘81’	The slave ACK equals '0'; bit 8 equals '1' indicating a read from the slave.
slave byte	x‘DD’	The master ACK equals '0'; indicating it wants the slave to send another byte.
slave byte	x‘CC’	The master ACK equals '0'; indicating it wants the slave to send another byte.

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 I2C 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 may 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 or read.
4. Optionally, the SCOM address could be auto-incremented to write or read successive SCOM addresses for writes and reads of more than 8-bytes. This option has not been implemented at this time.
5. 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 bytes rewritten.

slave byte	x‘BB’	The master ACK equals '0'; indicating it wants the slave to send another byte.
slave byte	x‘AA’	The master ACK equals '1'; indicating it wants the slave to send another byte.
master stop bit		

Note: In the examples, the master is an external I2C controller; the slave is the I2CMasterSlave hardware.

TapCmd Example 2: SCOM WRITE of register address x'040506' with data

The following sequence of I2C commands is needed to perform an SCOM write of register address x'040506', assuming the data to write is x‘BADC 0FFE EBAD C0FF’:

master start bit		
master byte	x'80'	The slave ACK equals '0'; indicating recognition of its own address; bit 8 equals '0', indicating a write to the slave.
master byte	x'06'	The slave ACK equals '0'; the address is written to the slave, least significant byte first.
master byte	x'05'	The slave ACK equals '0'; the address is written to the slave, middle byte.
master byte	x'04'	The slave ACK equals '0'; the address is written to the slave, most significant byte.
master byte	x'FF'	The slave ACK equals '0'; data is written to the slave, least significant byte.
master byte	x'C0'	The slave ACK equals '0'; data is written to the slave.
master byte	x'AD'	The slave ACK equals '0'; data is written to slave.
master byte	x'EB'	The slave ACK equals '0'; data is written to slave.
master byte	x'FE'	The slave ACK equals '0', data is written to slave.
master byte	x'0F'	The slave ACK equals '0'; data is written to slave.
master byte	x'DC'	The slave ACK equals '0'; data is written to slave.
master byte	x'BA'	The slave ACK equals '0'; data is written to slave, most significant byte.
master stop bit		

A.4.3 Non-Register Commands

Table A-3. TAP Command (bit numbering, LSb = '0')

Start Byte		SCOM Address / TAP Command			Serial Shift Register Data (I/O buffer)							
I2C Slave Address (7 bits)	R/$\overline{W}$	LSByte	MidByte	MSByte	8 bytes (LSByte, MSbit first)							
7:4 970MP base address, 3:1 ProcID	'0'	7 TSR 6 TTSR 5:0 BCR	15:12 x'4' 11:8 x'0' = TapCmd 11:8 x'1' = Null TapCmd 11:8 x'2' = Enable Attn 11:8 x'3' = Disable Attn[1]	23:16: x'52'	7:0							63:56
	'1'											

1. Attention checking is disabled by default.

This command format is identified by the reserved SCOM address (23:12) = x'524' (this address is relocatable). There are eight modes of operation, selected by the value of TapCmd bits 10:8. Bit 11 is reserved for future use (it has no effect at this time).

A.4.3.1 Primitive TAP Command (TapCmd Addr(11:8) = '0000')

In this format, the LSByte of the SCOM address is interpreted as a TAP command (see *Section A.3.3 Primitive Command Summary* on page 40) causing bits to be shifted out to the TAP port on TMS or TDI and capturing TDO. This enables the I2C interface to perform any action supported by the JTAG logic.

A stop or overflow condition initiates the BCR with its current value for a write operation. An overflow condition, stop, or start with the R/W Address bit set to '1' initiates the BCR with its current value for a read operation.

Writing or reading more than "BCR_value mod 8" (+ 1 if the remainder is non-zero) bytes of data causes a BCR reload to continue the data stream (that is, to write or read successive bytes in the scan ring).

The data represents either a TMS sequence, an IR command/modifier, or plain TDR data. Start or stop clocks, wire test enable, and so on are usually SCOM operations, but TAP commands are required to scan a ring (through a greybox). The Access scan out command supports TDO to TDI data wrap for a non-destructive scan ring read.

TapCmd Example 3: 1000 Bit Scan-IN

master start	x'80084052DF00' stop	The master sends the slave (JTAG) to Shift-IR.
master start	x'80DE40524100080F' stop	The master loads the '0F800041' command into JTAG. See the ACCESS specification for more information.
master start	x'8002405203' stop	The master sends JTAG to Shift-DR.
master start	x'80BE4052'	The master sets up the slave to operate in 64-bit bursts.
	x'DATADATADATADATA'	1000 bits / 64 = 15 (8-byte bursts) + 40 bit remainder
	...	

	...	
	x‘DATADATADATADATA’ stop	This is the last 8-byte burst.
master start	x‘80E64052’	The master sets up the slave to expect 40 bits of data.
	x‘0123456789’ stop	Final 40 bits of data.
master start	x‘800340521F’ stop	Return slave/JTAG to TestLogicReset.

The final example assumes that the TAPCMD primitive address is x'5240' (this is relocatable).

A.4.3.2 Null TAP Command (TapCmd Addr(11:8) = ‘0001’)

JTAG activity on TCK/TMS/TDI is suppressed. This mode is intended, for example, to read back IR status information immediately following a TapCmd sequence that scans a new command into the IR.

Note: An I2C write to the SSR data (I/O buffer) has no effect. An I2C read of the SSR data will return the most recent JTAG/TDO data scanned out.

TapCmd Example 4: Read IR status immediately after instruction scan

1. Load primitive command: Instruct the slave to bring the TAP state machine first to Test Logic Reset and then to Shift-IR to scan in an instruction.

   I2C Write Start with R/W Bit = ‘0’, Length = 5 Bytes, STOP

Byte 1	08x:	10 TCK pulses (data after address/command bytes is TMS sequence)
Byte 2	40x:	LSB of primitive command address
Byte 3	52x:	MSB of primitive command address
Byte 4	DFx:	TMS sequence of values (taken from rightmost LSB to MSB)
Byte 5	00x:	TMS sequence of values continued (gets TAP state machine to Shift-IR)

2. Load primitive command and instruction data. Instruct the slave to scan the following into the TAP Controller Instruction Register: 0F 80 00 41. This is a scan of the internal ring. Then, take the TAP state machine to Exit1-IR.

   I2C Write Start with R/W Bit = ‘0’, Length = 7 Bytes, STOP

Byte 1	DEx:	32 TCK pulses
Byte 2	40x:	LSB of primitive command address
Byte 3	52x:	MSB of primitive command address
Byte 4	41x:	LSB of instruction
Byte 5	00x:	Next byte of instruction
Byte 6	80x:	Next byte of instruction
Byte 7	0Fx:	MSB of instruction (scan out of internal scan ring)

3. Load primitive command: Instruct slave to read what was last scanned out of the scan controller (this will be the IR-STATUS since an Instruction Register scan was the last operation).

   I2C Write Start with R/W Bit = ‘0’, Length = 3 Bytes, STOP

Byte 1	NNx	Number of bytes to read minus 2
Byte 2	41x	LSB of primitive command address
Byte 3	52x	MSB of primitive command address

4. Get return data for primitive read command. Return data that was scanned out.

   I2C Read Restart with R/W Bit = ‘1’), Length = 4 Bytes, STOP

   Byte 1 -x: LSB of IR status
   Byte 2 --x: Next byte of IR status
   Byte 3 --x: Next byte of IR status
   Byte 4 --x: MSB of IR Status
   ...where “--” represents data returned.

5. Load next primitive command: Instruct slave to bring TAP state machine to Test Logic Reset State.

   I2C Write Start with R/W Bit = ‘0’, Length = 4 Bytes, STOP

   Byte 1 03x: 5 TCK pulses
   Byte 2 40x: LSB of primitive command address
   Byte 3 52x: MSB of primitive command address
   Byte 4 1Fx: Sequence of TMS values from rightmost bit to leftmost (only 5 bits used to return TAP state machine to reset)

A.4.3.3 Enable Attention Command (TapCmd Addr(11:8) = ‘0010’)

The I2C uses the following method to enable an attention raised from an Access action. An I2C write to the SSR data (I/O buffer) enables attention checking by the I2C slave. JTAG activity is suppressed.

Note: When this sequence is written, the slave then acknowledges SCOM read/write operations only when attention is non active (for more information, see the Access specification). When Attention is active, only primitive TAP commands will be acknowledged with the standard I2C ACK pulse.

TapCmd Example 5: Re-enabling attention checking internally

I2C Write Start with R/W Bit = ‘0’), Length = 4 Bytes, STOP

Byte 1 --x: This byte is a don’t care for this command.
Byte 2 42x: LSB of primitive command address
Byte 3 52x: MSB of primitive command address

Issue I2C stop sequence.

A.4.3.4 Disable Attention Command (TapCmd Addr(11:8) = '0011')

The I2C uses the following method to disable an attention raised from Access action. An I2C write to the SSR data (I/O buffer) disables attention checking by the I2C slave.

Note: When this sequence is written, the slave will acknowledge primitive commands as well as SCOM read/write sequences even if Attention is active.

This method is used to override issues such as a machine check being active, which would in turn cause an attention to be raised. This does not correct an attention being raised from an SCOM error and the data received after an SCOM error should be disregarded (see the SCERR section in the Access documentation). By default, attention is masked off.

TapCmd Example 6: Masking attention internally

I2C Write Start with R/W Bit = '0', Length = 4 Bytes, STOP

Byte 1	--x:	- TCK pulses
Byte 2	43x:	LSB of primitive command address
Byte 3	52x:	MSB of primitive command address

Issue I2C stop sequence.

Appendix B. HID Registers and SPRs

B.1 Move To/From System Register Instructions

The 970MP 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 B-1*. 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 B-1. Implementation-Specific SPRs (Page 1 of 2)

SPR			Register Name	R/W	Synchronization Requirements		
Decimal (privileged)	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
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

Note: For **mtspr**, n must be 1. For **mfspr**, reading the SPR is privileged if and only if n=1.

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

SPR			Register Name	R/W	Synchronization Requirements		
Decimal (privileged)	Decimal (user)	SPR(5:9) SPR(0:4)			Before Reads	After Writes	Before Writes
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
311		01001 10111	HIOR	R/W			
Note: For **mtspr**, n must be 1. For **mfspr**, reading the SPR is privileged if and only if n=1.							

Notes: (*for Table B-1*)

1. The following sequence must be used when modifying HID0:
   sync
   mtspr HID0,Rx
   mfspr Rx,HID0
   mfspr Rx,HID0
   mfspr Rx,HID0
   mfspr Rx,HID0
   mfspr Rx,HID0
   mfspr Rx,HID0
   After modifying HID0, executing six **mfspr** instructions specifying HID0 as the source and specifying the same target General Purpose Register (GPR) (Rx) in all six instructions is necessary to ensure that the modification is effective and that the processor is in a valid state to continue executing subsequent instructions.
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 Instruction Cache Synchronize (**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 Move to Segment Register (**mtsr**) and Synchronize (**sync**) instruction**,** which will cause the effective-to-real-address translations (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 ensure that any succeeding Data Cache Block Set to Zero (**dcbz**) instruction 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 B-2 describes the 970MP's behavior for the **mtspr** and **mfspr** instructions.

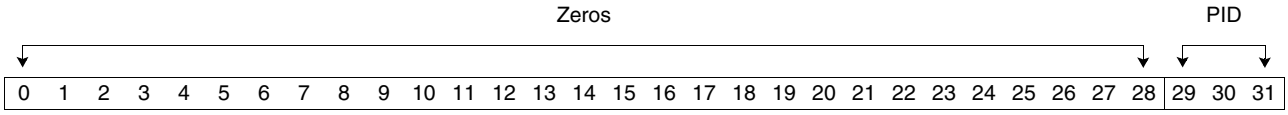
Table B-2. Move To / Move From SPR Behavior

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, perfmon regs	0	**mfspr**	Returns value to GPR
		0	**mtspr**	Target SPR is updated
1	TRIG0, TRIG1, TRIG2	0	**mfspr**	Causes illegal instruction type program interrupt
		0	**mtspr**	Causes trigger to trace array debug logic
1	PIR	0	**mfspr**	Returns value to GPR
		0	**mtspr**	Causes illegal instruction type program interrupt
1	Any SPR encoding (w/ SPR(0)=1)	1	**mtspr** **mfspr**	Causes privileged instruction type 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 illegal instruction type program interrupt

B.1.1 Processor ID Register (PIR)

The processor identification register (PIR) is a 32-bit register that holds a processor identification tag in the three least significant bits (29:31). This tag is used for tagging bus transactions and for processor differentiation in multiprocessor systems. The form of the register is as follows:

Table B-3. Processor Identification Register Bit Functions



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

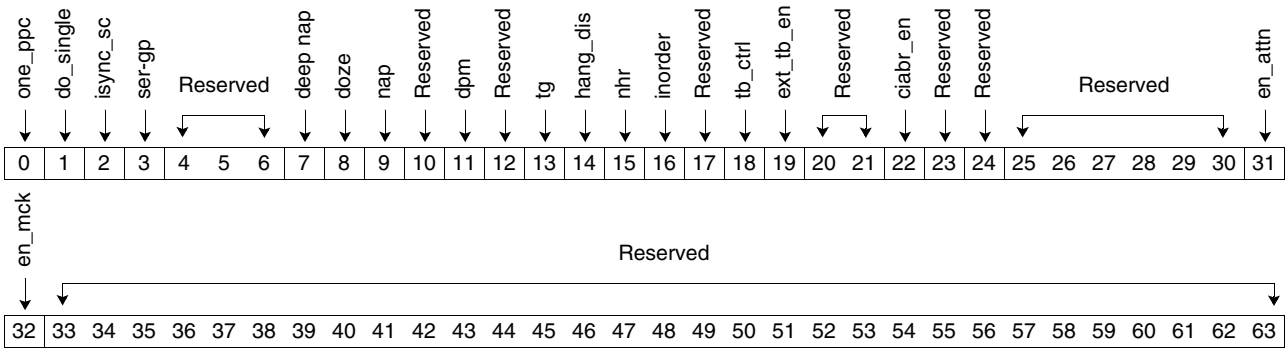
The PIR is a *read only* register. During *power-on reset*, PID is set to a unique value for each processor in a multi-processor system.

B.1.1.1 HID Registers (HID0, HID1, HID4, and HID5)

The PPC 970MP microprocessor 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 970MP microprocessor modes with the relevant ones from earlier PowerPC implementations and then adds a few new ones. HID1 contains additional mode bits that are related to the instruction fetch and instruction decode functions in the PPC 970MP microprocessor. HID4 and HID5 contain bits related to the load/store function in the PPC 970MP microprocessor. All of these registers are supervisor resources.

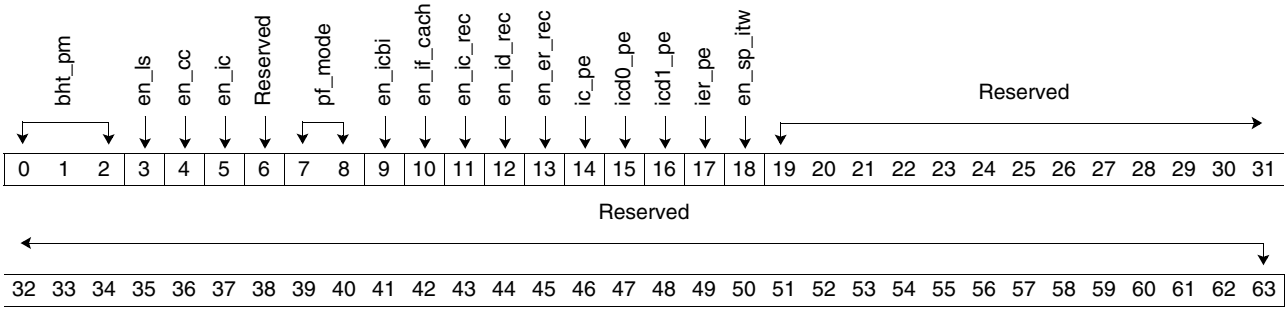
The state of each of the HID registers after a normal scan-based POR is all zeros. The preferred state of these registers for optimal performance and function is also all zeros, except where indicated.

Table B-4. HID0 Bit Functions



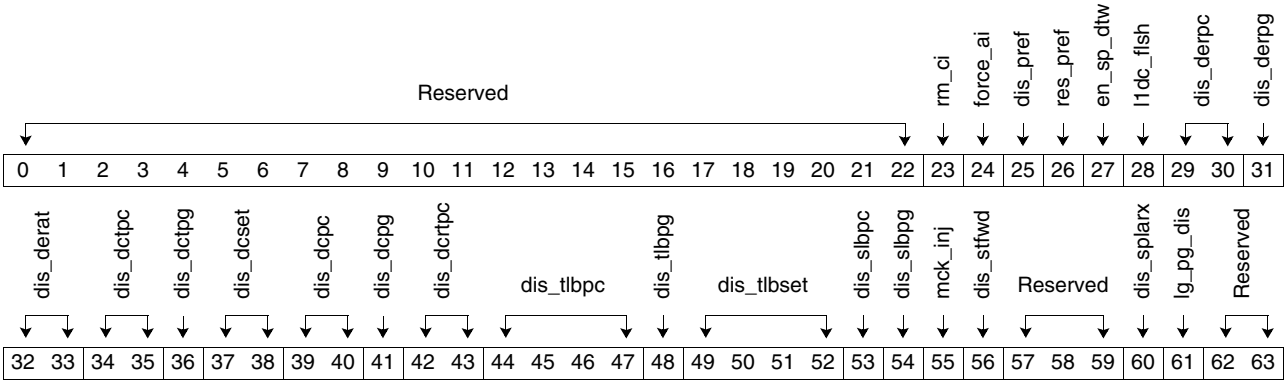
Bits	Field Name	Description
0	one_ppc	One PowerPC AS instruction per dispatch group mode. An instruction may span more than one group.
1	do_single	Single group completion mode. Flush and refetch after the completion of each group or the completion of each microcoded instruction, if the instruction spans multiple groups.
2	isync_sc	Disable isync scoreboard optimization.
3	ser-gp	Serialize group dispatch. The next group is not dispatched until the previous group completes.
4:6	—	Reserved
7	deep nap	Deep nap
8	doze	Doze
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 snoop response in (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 the processor is stopped.
19	ext_tb_en	External time-base enable. 0 Use TBEN input as enable. TB is clocked at 1/8 of the full processor frequency. 1 Use TBEN input to clock time base (external clock).
20:21	—	Reserved
22	ciabr_en	Enable Completion Instruction Address Breakpoint Register (CIABR).
23	—	Reserved
24	—	Reserved
25:30	—	Reserved
31	en_attn	Enable support processor attention instruction.
32	en_mck	Enable external machine check interrupts (preferred state equals ‘1’).
33:63	—	Reserved

Table B-5. HID1 Bit Functions



Bits	Field 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 equals ‘1’).
4	en_cc	Enable count cache (preferred state equals ‘1’).
5	en_ic	Enable instruction cache (must be ‘1’ for proper functioning).
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 Instruction Cache Block Invalidate (**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 I bit in the page table 1 Instruction fetch cacheability is controlled by the state of the I bit in the page table (preferred state)
11	en_ic_rec	Enable I-cache parity error recovery (preferred state equals ‘1’).
12	en_id_rec	Enable I-directory parity error recovery (preferred state equals ‘1’).
13	en_er_rec	Enable instruction ERAT (I-ERAT) parity error recovery (preferred state equals ‘1’).
14	ic_pe	Force instruction cache parity error (error inject).
15	icd0_pe	Force instruction cache directory 0 parity error (error inject).
16	icd1_pe	Force instruction cache directory 1 parity error (error inject).
17	ier_pe	Force I-ERAT parity error (error inject).
18	en_sp_itw	Enable speculative tablewalks. The ERAT is never loaded using a page table entry (PTE) if PTE[G] is set to ‘1’ (preferred state equals ‘1’).
19:63	—	Reserved

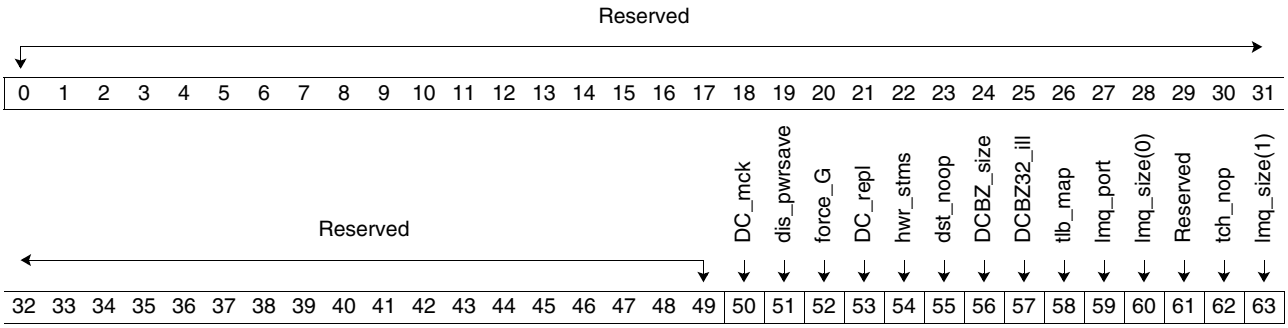
Table B-6. HID4 Bit Functions



Bits	Field Name	Description
0:22	—	Reserved
23	rm_ci	Data accesses in real mode are treated as cache-inhibited.
24	force_ai	Force alignment interrupt instead of microcoding unaligned operations (that is instead of breaking unaligned operations into multiple smaller operations).
25	dis_pref	Disables data prefetching.
26	res_pref	Setting HID4[26] to ‘1’ resets the data prefetch mechanism, suppressing subsequent prefetch requests and clearing the stream detection logic so that stream detection is not affected by accesses performed before 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 data ERAT (D-ERAT) parity checking (one bit for each set).
31	dis_derpg	Disable D-ERAT parity generation (force parity to ‘0’ on EA[0:45] only).
32:33	dis_derat	Disable one or more ways of the 4-way set associative D-ERAT (one bit per set); valid states are 00, 01, and 10.
34:35	dis_dctpc	Disable data cache tag parity checking (one bit for each set).
36	dis_dctpg	Disable data cache tag parity generation.
37:38	dis_dcset	Disable data cache set (one bit for each set).
39:40	dis_dcpc	Disable parity checking in one or more ways of the 4-way set-associative data cache (one bit per set).
41	dis_dcpg	Disable data cache parity generation.
42:43	dis_dcrtpc	Disable parity checking on the physical address tag of the data cache (one bit per set).
44:47	dis_tlbpc	Disable parity checking in one or more ways of the 4-way set-associative translation lookaside buffer (TLB) (one bit per set).
48	dis_tlbpg	Disable TLB parity generation.
49:52	dis_tlbset	Disable set in one or more ways of the 4-way set associative TLB (one bit per set); valid states are x‘0’, x‘7’, x‘B’, x‘D’, and x‘E’.
53	dis_slbpc	Disable SLB parity checking.
54	dis_slbpg	Disable SLB parity generation.
55	mck_inj	Enable machine-check error injection.
56	dis_stfwd	Disable store forwarding (cause reject).

Bits	Field Name	Description
57:59	—	Reserved
60	dis_splarx	Disable speculative Load Word and Reserve Indexed (**lwarx**) and Load Double Word and Reserve Indexed (**ldarx**) instructions.
61	lg_pg_dis	Disable large page support. The large page (L) bit input to SLB will be forced to zero (software will read a zero L bit).
62:63	—	Reserved

Table B-7. HID5 Bit Functions



Bits	Field Name	Description
0:49	—	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 (D-cache), L1 D-cache tag, D-ERAT power savings disable.
52	force_G	Force guarded (G equals ‘1’) load.
53	DC_repl	Data cache replacement algorithm. 0 Least recently used (LRU) (default) 1 First-in-first-out (FIFO)
54	hwr_stms	Number of available hardware prefetch streams. 0 Four hardware streams and four VPU streams 1 Eight hardware streams (HID5[55] must also be ‘1’)
55	dst_noop	Data Stream Touch (DST) instructions no-op. 0 DSTs are enabled. 1 DSTs are a no-ops and discarded in the load/store unit (LSU).
56	DCBZ_size	Makes **dcbz** a 32-byte store when bit 10 of the **dcbz** instruction is set to ‘0’.
57	DCBZ32_ill	Makes a **dcbz** instruction with bit 10 equal to ‘0’ an illegal instruction.
58	tlb_map	TLB mapping. 0 4-way set associative 1 Direct mapped **Note:** When setting HID5[58] to make the TLB direct mapped, the TLB set disable bits, HID4[49:52], must be cleared; otherwise, translation will not work.
59	lmq_port	Demand miss (load miss queue [LMQ] to 970MP storage subsystem [STS]). 0 Permit two per cycle. 1 Permit only one per cycle (this setting is not currently supported)

Bits	Field Name	Description
60	lmq_size(0)	Number of outstanding requests to STS. HID5 [60, 63] — Maximum outstanding requests 00 — 8 01 — 1 (this setting is not currently supported) 10 — 2 11 — 4
61	—	Reserved
62	tch_nop	Make the Data Cache Block Touch (**dcbt)** and Data Cache Block Touch for Store (**dcbtst**) instructions act like no-ops.
63	lmq_size(1)	See description of HID5[60].

B.1.2 SCOM Registers (SCOMC and SCOMD)

The 970MP 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 this facility.

B.1.3 Trigger Registers

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 bringup only and architecturally behave as a no-op.

B.1.4 IMC Array Access Register

The Instruction Match Cam (IMC) array facility is used for performance monitoring instrumentation and for the softpatch of instructions (this latter use is restricted for the support processor and is not available through the SPR access to this register array). The array has privileged write access and user-level read access via 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.

B.1.5 Performance Monitor Registers

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 970MP.

Appendix C. Additional Mode Ring Customization

C.1 Mode Ring Table Introduction

The *Table C-1* on page 59 in this appendix provides the complete recommended mode ring for both cores of a 970MP in I2C format. For most applications, only minimal mode ring customization is required as documented in *Section 1.5.5.2* beginning on page 25.

Some applications may need to customize the mode ring beyond the normal recommended ring that is provided in that section. This appendix documents some of the bits and their function as a guide to further customization.

Note: Mode ring changes should be carefully evaluated. Some changes may cause checkstops or other system exceptions if used improperly. Others may degrade performance or system stability. Customers should generally avoid changing mode ring bits without carefully evaluating any potential negative effects.

C.1.1 Correctly formatting mode ring bits for I2C

This appendix assumes that the mode ring will be sent to the 970MP in I2C format. When modifying individual bits in the mode ring it's important to remember that the mode ring bits are sent from the highest numbered bit working down to the last bit in the ring which is numbered 0. So although the mode ring is 1984 bits, the first bit is actually numbered 1983.

The bits within each byte are also sent in bitswapped format (i.e. msb of each byte is actually the lsb and vice versa). In order to convert each byte into a normal binary format you would have to swap the bits around to get the highest order bit in the left most position.

Also, pay close attention to the designation **inverted**. It's used to indicate when mode ring bits need to be provided in one's complement (logical inverted) form.

C.1.1.1 Identifying and modifying mode ring bits in the master table

To corrrectly identify a given bit in the mode ring, start with the bit number, then consult the ring start/end columns in the table to figure out which 64 bit burst contains the byte.

Subtract the bit number(s) from the ring start bit, then divide by 8 to figure out which byte contains the bit. The bytes are organized from the highest bit in byte 1 to the lowest order bits in byte 8.

The last step is bit swapping the affected byte to get the bits in msb-lsb order.

Note: Don't forget to pay attention to bits marked **inverted**. Those bits must be put into the ring by their logical inversion (one's complement) form.

Table C-1. PowerPC 970MP Mode Ring Data Table in I2C Format (DD 1.x)

64 Bit Burst	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Ring Start Bit	Ring End Bit
1	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	1983	1920
Notes Burst 1	The first 46 bits of this burst are “padding”. They are used to provide a convenient way to initialize both mode rings in 64 bit bursts without resorting to smaller sections that would have to be sent differently. The “Actual” mode ring starts in byte 6 with the 2 lowest order bits. Those 2 lowest bits correspond to Bits 1937 and 1936. The padding bits are discarded by the final 31st 64 bit burst of I2C data.									
2	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	1919	1856
Notes Burst 2										
3	0x00	0x06	0x00	0x00	0x00	0x00	0x00	0x00	1855	1792
Notes Burst 3	Bits 1800:1811 FIR masks - Default 0xFFF - **Inverted -** Override with write to SCOM 0x80401 bits 48:59									
4	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	1791	1728
Notes Burst 4	Bits 1778:1789 are used for dither control. Dithering controls how powertuning switches from full to 1/2 or 1/4 speed. Bits 1730:1777 are the dither slow pattern. - Default 0x0000.0000.0000 Bits 1682:1729 are the dither fast pattern. - Default 0x0000.0000.0000									
5	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0xA0	1727	1664
Notes Burst 5	Bits 1682:1729 are the dither fast pattern. - Default 0x0000.0000.0000 Bits 1644:1675 are power tune settings for F/2 and F/4 bus operation - See *Table 1.5.5.3* on page 26 for customization guide									
6	0x00	0xC1	0x00	0x01	0x00	0x00	0x00	0x00	1663	1600
7	0x00	0x01	0x00	0x00	0x00	0x82	0x82	0x00	1599	1536
	Bits 1538:1541 VSQCTL.MODE_BITS.SCMSLATCH.SCB.SC.L2[6:9) = 0b0000 to workaround L2 store queue overflow errata									
8	0x00	0x80	0xE1	0x00	0x00	0x04	0x00	0x04	1535	1472
	Bit 1492 MCSQC.MODE_SWITCHES.SCMSLATCH.SCB.SC.L2[0] = 0b’1’ altermate fix for L2 store queue overflow errata (Since it is fixed in burst 7 above the fix is not needed here - It would change byte 6 from 0x04 to 0x0c)									
9	0x00	0x65	0xD8	0x10	0x00	0x00	0x00	0x00	1471	1408
10	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	1407	1344
11	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	1343	1280
12	0x00	0x00	0x00	0x1F	0x00	0x00	0x00	0x00	1279	1216
Notes Burst 12	Bits 1251:1255 Debug and Trace control - Default 0b000000 - **Inverted** Bits 1214:1245 checkstop enable (0:31) - Default 0x0000.0000 - Override with write to SCOM 0x30800									
13	0x00	0x00	0x00	0x00	0xFC	0xFF	0xFF	0xFF	1215	1152
Notes Burst 13	Bits 1214:1245 Checkstop enable (0:31) - Default 0x0000.0000 -Override with write to SCOM 0x30800 Bits 1182:1213 Machine Check Enable (0:31) - Default 0x0000.0000 - Override with write to SCOM 0x30901 Bits 1150:1181 Core Fir Mask Register - Default 0xFFFF.FFFF - Override with write to SCOM 0x30400 - **Inverted**									

Table C-1. PowerPC 970MP Mode Ring Data Table in I2C Format (DD 1.x)

64 Bit Burst	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Ring Start Bit	Ring End Bit
14	0x03	0x00	0x00	0x00	0x00	0x00	0x00	0x00	1151	1088
Notes Burst 14	Bits 1150:1181 Core Fir Mask Register - Default 0xFFFF.FFFF - Override with write to SCOM 0x30400 - **Inverted**									
15	0x00	0x00	0x00	0x00	0x00	0x00	0x40	0x00	1087	1024
	Bit 1034 SCU_SPARE_MSFF(13) = 0b'1' to workaround potential hang with external interrupt errata									
16	0x08	0x00	0x00	0x00	0x00	0x00	0x00	0x00	1023	960
Notes Burst 16	Bits 978:1009 MSR (0:31) Initial value - Default 0x0000.0000 - **Inverted** Bits 914:977 HIOR (0:63) initial value - Default 0x0000.0000.0000.0000 See ***Table 1-6*** on page 25									
17	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	959	896
Notes Burst 17	Bits 914:977 HIOR (0:63) initial value - Default 0x0000.0000.0000.0000 See ***Table 1-6*** on page 25									
18	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	895	832
19	0x00	0x80	0x00	0x00	0x00	0x00	0x00	0x00	831	768
	Bit 816 SCX.SPARE_MSFF(2) = 0b1' to workaround potential hang with external interrupt errata									
20	0x00	0x00	0x02	0x00	0x00	0x00	0x00	0x00	767	704
21	0x00	0x00	0x54	0x30	0xC2	0xA1	0xAE	0x00	703	640

Table C-1. PowerPC 970MP Mode Ring Data Table in I2C Format (DD 1.x)

64 Bit Burst	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Ring Start Bit	Ring End Bit
Notes Burst 21	Bit 687 HID1 bit 14 shadow - Default 0b'0' - Force instruction cache parity error (error inject) Bit 685 HID1 bit 12 shadow - Default 0b'1' - Enable I-directory parity error recovery Bit 684 HID1 bit 15 shadow -Default 0b'0' - Force ICache directory 0 parity error (error inject) Bit 683 HID1 bit 7 shadow - Default 0b'1' - Prefetch mode (select NSA and NSA+1 instruction prefetch - preferred state) Bit 682 HID1 bit 16 shadow - Default 0b'0' - Force I Cache directory 1 parity error (error inject) Bit 681 HID1 bit 18 shadow - Default 0b'1' - Enable speculative tablewalks (preferred state) Bit 679 HID1 bit 17 shadow - Default 0b'0' - Force I-ERAT parity error (error inject) Bits 676:678 HID1 bits 8:10 shadow - Default 0b'000' - bit 8 isPrefetch mode (select NSA and NSA+1 instruction prefetch - preferred state), bit 9 is Enable forced icbi match mode, bit 10 is I fetch cacheability control (preferred state) Bit 675 HID1 bit 11 shadow - Default 1 - Enable I Cache parity error recovery (preferred state) Bit 674 HID1 bit 13 shadow - Default 1 - Enable I ERAT parity error recovery (preferred state) Bit 673 HID0 bit 9 - Default 0b'0' - Nap Mode Bit 671 HID0 bit 32 - Default 0b'0' - Enable external machine check interrupts Bit 669 HID0 bit 8 - Default 0b'0' - Doze mode Bit 667 HID0 bit 14 - Default 0b'0' - Disable processor hang detect mechanism Bit 666 HID0 bit 18 - Default 0b'0' - Enable time-base counting when processor is stopped Bit 665 HID0 bit 31 - Default 0b'1' - Enable Attention Bit 664 HID0 bit 19 - Devault 0b'1' - TBEN input enable or external clock input, 0 = TBEN enable 1 = external clock input Bit 663 HID0 bit15 - Default 0b'1' - Not hard reset Bit 662 HID0 bit 16 - Default 0b'0' - Serialized group issue mode Bit 661 HID0 bit 0 - Default 0b'0' - One PPC instruction per dispatch group Bit 660 HID0 bit 3 - Default 0b'0' - Serialize group dispatch Bit 659 HID0 bit 7 - Default 0b'0' - Deep Nap Bit 657 HID0 bit 11 - Default 0b'0' - Enable Dynamic Power Management Bit 656 HID1 bit 1 - Default 0b'1' - Branch history prediction mode (preferred state) Bit 654 HID1 bit 5 - Default 0b'1' - Enable I Cache (must be 1 for proper function) Bit 653 HID1 bit 0 - Default 0b'1' - Branch history prediction mode (preferred state) Bit 652 HID1 bit 4 - Default 0b'1' - Enable count cache (preferred state) Bit 651 HID0 bit 2 - Default 0b'0' - Disable isync scoreboard optimization Bit 650 HID1 bit 3 - Default 0b'1' - Enable link stack (preferred state) Bit 649 HID0 bit 22 - Default 0b'0' - Enable CIABR Bit 648 HID1 bit 2 - Default 0b'1' - Branch history prediction mode (preferred state) Bit 645 HID0 bit 1 - Default 0b'0' - Single group completion mode Bit 644 HID0 bit 13 - Default 0b'0' - Performance monitor threshold granularity control									
22	0x00	0x00	0x10	0x00	0x00	0x00	0x00	0x40	639	576
23	0x01	0x00	0x00	0x00	0x00	0x00	0x00	0xEA	575	512
24	0x07	0x00	0x00	0x00	0x00	0x00	0x00	0x00	511	448
Notes Burst 24	Bits 479:482 Live lock prevention - default 0b'1111' - **Inverted** Bit 483 Live lock prevention - Default 0b'0' **Note:** Bits 479-483 should not be all 0 - LFSR implementation - one bit must be on to prevent performance issue									
25	0x00	0x00	0x03	0x00	0x00	0x00	0x00	0x00	447	384
26	0x00	0x00	0x00	0x00	0x00	0x00	0x09	0x48	383	320
27	0xFF	0xFD	0x07	0x00	0x00	0x00	0x00	0x00	319	256

Table C-1. PowerPC 970MP Mode Ring Data Table in I2C Format (DD 1.x)

64 Bit Burst	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Ring Start Bit	Ring End Bit
Notes Burst 27	Bits 237:300 HID4 (63:0) Default 0x0000.0000.0000.0000 - **NOTE: HID4 is stored in the ring MSB:LSB**									
28	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	255	192
Notes Burst 28	Bits 237:300 HID4 (63:0) Default 0x0000.0000.0000.0000 - **NOTE: HID4 is stored in the ring MSB:LSB** Bits 173:236 HID5 (0:63) -Default 0x0000.0000.0000.0000									
29	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	191	128
	Bits 173:236 HID5 (0:63) - Default 0x0000.0000.0000.0000									
30	0x00	0x00	0x00	0x84	0xA3	0x1F	0x00	0x00	127	64
	Bits 83:102 HID1 (0:19) Default HID1 0x0000000.000FD3C2									
31	0x00	0x00	0x00	0x00	0x60	0x07	0x00	0x00	63	0

Appendix D. Glossary

ABIST	array built-in self test
Acc_Opt	Access Options Register
ACK	acknowledgment
AD	address/data
adc	almost dc
ADO	array data output
AET	all event trace
aPC	program counter
ASI	active source identifier
AUE	special UE to indicate an altered UE
AVP	architectural verification program
Bclk	bus clock
BCR	Bit Count Register
BHT	branch history table
BIST	built-in self test
BIU	bus interface unit
CAM	content-addressable memory
CAP	clock alignment procedure
CE	correctable data error
CIA	current instruction address
CIABR	Completion Instruction Address Breakpoint Register
CIU	core interface unit
COM	common area
COP	common on-chip processor
CR	Condition Register
CRA	custom register array
CRC	cyclic redundancy check
CRESP	combined response

CTR	Count Register
DABR	Data Address Breakpoint Register
D-cache	data cache
DCBZ	data cache block set to zero
Dclk	data rate clock
DEC	decrementer
Derr	data error
di/dt	rate of current change
DPM	dynamic power management
DR	Data Register
DSI	data storage interrupt
dssall	data stream stop all
E	exclusive
EA	effective address
EC	engineering change
ECC	error correction code/error checking and correction
ECCCK	error correction code check
ECID	electronic chip ID
EE	enable exception
eFuses	electrical fuses
eGPR	emulation GPR
EI2	elastic interface 2
EICAL	elastic interface recalibration
EIO	elastic I/O
EPS	event processing sequencer
ERAT	effective-to-real address translation
EST	electrical shorts test
f	functional frequency
F1	frequency one

F2	frequency two
FAR	Fault Address Register
FIFO	first-in-first-out
FIR	Fault Isolation Register
fo	force only
FP	floating point
FPR	Floating Point Register
FPSCR	Floating Point Status and Control Register
FPU	floating point unit
FRU	field replaceable unit
FXU	fixed point unit
GBD	guard band
GCT	global completion table
GPR	General Purpose Register
GUI	graphical user interface
HID	Hardware Implementation Dependent Register
hreset	hard reset
I^2C	inter-integrated circuit
IABR	Instruction Address Breakpoint Register
IAP	initial alignment pattern
ICA	initial clock alignment
I-cache	instruction cache
ICBI	instruction cache block invalidate
IDDQ	quiescent supply current
IDU	instruction decode unit
IEEE	Institute of Electrical and Electronics Engineers
IFAR	Instruction Fetch Address Register
IFU	instruction fetch unit
IMC	instruction match CAM

imr	instruction mark
INIT	initialization
IOBUF	I/O buffer
IOP	internal operation
IPL	initial program load
IR	Instruction Register
ISU	instruction sequencer unit
ISV	instruction sequencer for VPU
ITR	interrupt
JTAG	Joint Test Action Group
L1	Level 1 cache
L1D	L1 data cache
L2	Level 2 cache
LBIST	logic built-in self test
lclk	local clock
LD/ST	load/store
LFSR	Linear Feedback Shift Register
LMQ	load/miss queue
LPAR	logical partitioning
LR	Link Register
LRU	least recently used
LSb	least significant bit
LSSD	level sensitive scan design
LSU	load-store unit
LTC	learned target cycle
MC	multi-cycle
MCHECK	machine check signal
mclk	mesh clock
ME	machine check enable

MERSI	modified/ exclusive/ reserved/ shared/ invalid cache-coherency protocol
MISR	Multiple Input Signature Register
MMCR2	Monitor Mode Control Register 2
MSb	most significant bit
MSB	most significant byte
MSR	Machine State Register
MTFPSCR	Move To Floating Point Status and Control Register
mthid0	move to HID0
mtmsr	move to Machine State Register
mtspr	move to Special Purpose Register
N/I	not implemented
ncctl	noncacheable control
NCF	nominal core frequency
NCU	noncacheable unit
NIA	next instruction address
NTC	next to complete
NUMA	nonuniform memory access
PAAM	previous adjacent address match
PBD	per-bit deskew
PCI	peripheral component interconnect
PCR	Power Control Register
PCRH	Power Control Register High
PI	processor interface
PI2	processor interface 2
PID	processor identification
PLL	phase-locked loop
PMU	performance monitor unit/power controller
POR	power-on reset
POW	power management

PR	problem
PRPG	pseudo-random test pattern generator
PSR	Power Status Register
psync	synchronizing clock
psyncnt	internal *psync* boundary
PU0	processing unit 0
PU1	processing unit 1
PVR	Processor Version Register
QACK	quiescent acknowledgment
QREQ	quiescent request
RA	real address
raddr	read address
RAS	reliability, availability, serviceability
RC	read/claim
RCFSM	receive state machine
RDT	random data test
RECERROR	recoverable error signal
refclk	system clock
RIAP	receiver IAP
RISC	reduced instruction set computer
RTAS	run time abstraction software
S	shared
SBE/CE	single bit error
SC	sequencer completion
SCL	serial clock
SCOM	scan communications
SCOM	scan communication (internal chip bus)
SCOMC	SCOM control
SCOMD	SCOM data

SCSCAN	SCOM scan
SD	sequencer dispatch
SDA	serial data
SDAR	Sampled Data Address Register
SDRAM	static dynamic random access memory
SE	service element/step enable
SEC/DEC	single error correct/double error detect
SEM	sequential execution model (program order instruction sequence)
SIAR	Sampled Instruction Address Register
SICR	Scan-In Check Register
SI	shared invalid
SLB	segment lookaside buffer
SLIC	System Licensed Internal Code
sp	soft patch
SPECATT	special-attention signal
SPR	special purpose register
SRC	system reference code
sreset	soft reset
SRL	shift register latch
SRQ	store reorder queue
SRR1	Save/Restore Register 1
SSR	Serial Shift Register
STS	storage subsystem
STWCX	store word conditional
SUE	special UE
SYNCRIAP	synchronize receiver initial alignment pattern
SYSCLK	system clock
TAP	test access port
TAPCMD	TAP command

TBEN	time base enable
TCK	test clock
TDI	test-data input
TDO	test-data output
TDR	Test Data Register
TH	transfer handshake
TLB	translation lookaside buffer
TLBI	translation lookaside buffer invalidate entry
TMS	test mode select
TO	test only
TR	transaction retry
TRST	test reset
TSRL2	TMS Register
TTSRL2	Terminal TMS Register
UE	uncorrectable memory error
VMA	vector multimedia arithmetic
VMP	VMX permute unit
VPA	VPU arithmetic
VPP	VPU permute
VPU	vector processing unit
VRF	vector register file
waddr	write address
WIAP	writer initial alignment pattern
WIMGRP	W = write through, I = cache inhibited, M = memory coherent, G = guarded read, R = rerunning, P = pipelined snoop
XER	Exception Register

Revision Log

Revision Date	Contents of Modification
July 28, 2006	Version 1.0 Initial public version released.