# PowerPC 970FX Power On Reset Application Note

## Version 0.6C

**Advance**

July 19, 2005


Printed in the United States of America July-2005



IBM Microelectronics Division
2070 Route 52, Bldg. 330
Hopewell Junction, NY 12533-6351

The IBM home page can be found at http://www.ibm.com

The IBM Microelectronics Division home page can be found at http://www.ibm.com/chips

970_por_title.fm.06C
July 19, 2005

# Contents

# List of Tables

# List of Figures

**This Page Intentionally Left Blank**

# 1. PowerPC 970FX 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 $\overline{\text{HRESET}}$. Operational modes were selected by pin strapping at $\overline{\text{HRESET}}$. Initialization of on-chip logic and arrays was handled by simple state machines triggered by the assertion of $\overline{\text{HRESET}}$.

But the 970FX requires a more complicated Power On Reset (POR) sequence. Processor initialization is handled by the pervasive logic which is controlled either by $I^2C$ 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 970FX based systems. It will explain the sequence of events required to start executing code on a 970FX and provide debugging tips for power on reset code.

## 1.2 Overview

The power on reset of a 970FX system goes through 7 separate phases, numbered IPL 0-6 as shown in *Figure 1-1*. The sequence actually begins with the ramping of power and clocks prior to IPL 0. Once power supplies and clocks are stable, $\overline{\text{HRESET}}$ and $\overline{\text{BYPASS}}$ should be asserted to reset the on-chip logic and PLL respectively.

In addition to the initialization of the 970FX, the service processor (SPU) handles other functions via $I^2C$. Initialization/configuration of the North Bridge, DIMMS, etc. must all be handled as part of the system POR sequence, however initialization of those devices is beyond the scope of this document.

## 1.3 Power On Reset SPU Hardware Considerations

It is assumed that 970FX 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 970FX and the North Bridge and can also be used to manage and supervise other system functions, like fans.

At a minimum, the SPU needs to be able to assert $\overline{\text{HRESET}}$ and $\overline{\text{BYPASS}}$ on the 970FX and should also have either a dedicated $I^2C$ bus master to initialize the system, or GPIO (General Purpose I/O Pins) that can be used to implement an $I^2C$ bus master.

### 1.3.1 $I^2C$ Bus Speed

The 970FX bus speed on $I^2C$ 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.3.2 SCOM (Register) Read/Write* on page 31.

**Note:** This SCOM write must occur after the second continue is sent during POR. Until this SCOM write occurs, the $I^2C$ bus speed will be limited to 50KHz. Once the SCOM has been written $I^2C$ bus speed may be increased to 100KHz.

### 1.3.2 Service Processor Firmware Bringup and Development

During early bringup and firmware development, use of a $I^2C$ controller/emulator is recommended. These tools are available from multiple sources and convert a PC's RS232 port or USB port to $I^2C$. Software on the PC can be used to communicate with the $I^2C$ bus in the system to develop and debug the power on reset sequence. Once the system has been brought up using the $I^2C$ 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.

## 1.4 Accessing Thermal Diode Calibration Data

The 970FX includes 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 970FX 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. Since this fuse ring will not change, it may optionally be scanned once during the first power up/bring up testing of the system, and flashed into non-volatile storage for later use.

Scanning the 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 $\overline{\text{HRESET}}$).

### 1.4.1 Sampling on Chip Thermal Diode Fuses

The POR sequence should be followed from *Section 1.6.1* beginning on page 15 through *Section 1.6.3* on page 17. This includes ramping of power 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.

### 1.4.2 Scanning Thermal Diode Data

Scanning the thermal diode data involves a series of TAP commands via $I^2C$. TAP commands are used to emulate JTAG operations on the 970FX and are explained in more detail starting in *Appendix A.3.3* on page 33.

To begin the fuse ring scan, first set the TAP controller to Shift-IR mode by writing the $I^2C$ byte sequence 0x08, 0x40, 0x52, 0xDF, 0x00 to the 970FX. Second, set the TAP controller for a scan-out of the fuse ring (ring address 0xC00002) by writing the $I^2C$ byte sequence 0xDE, 0x40, 0x52, 0x02, 0x00, 0xC0, 0x12. Note that the three byte ring address is sent in byte reverse order. Third, set the TAP controller to Shift-DR mode by writing the $I^2C$ byte sequence 0x02, 0x40, 0x52, 0x03.

Now the TAP controller 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 970FX. This scan of the fuse ring data proceeds from the highest numbered bit [1591] in the ring to bit [0], the first bit in the ring.

Since the thermal diode calibration data is at the end of the ring in bits [1546-1583], the first 64-bit read includes all the data. This data is encoded per *Table 1-1 Thermal Diode Data Encoding*.

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 1.6.1* on page 15. 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 $\overline{\text{BYPASS}}$.

*Table 1-1. Thermal Diode Data Encoding*

| Field name | Ring Position | Offset within First 64-Bit Read | Length in Bits | Adjustment | Expected Range |
|---|---|---|---|---|---|
| temp_low | 1546:1552 | 39:46 | 7 | Value - 40 | TBD |
| temp_high | 1553:1559 | 32:38 | 7 | None | TBD |
| voltage_low | 1560:1571 | 20:31 | 12 | Value / 2 + 300 | TBD |
| voltage_high | 1572:1583 | 8:19 | 12 | Value / 2 + 300 | TBD |

**Note:** All values are stored LSb:MSb (bit reversed). They should be bitswapped before applying the adjustments shown in this table. Temperatures are stored in degrees C, voltages are stored in milivolts.

*Figure 1-1. 970FX POR General Overview*

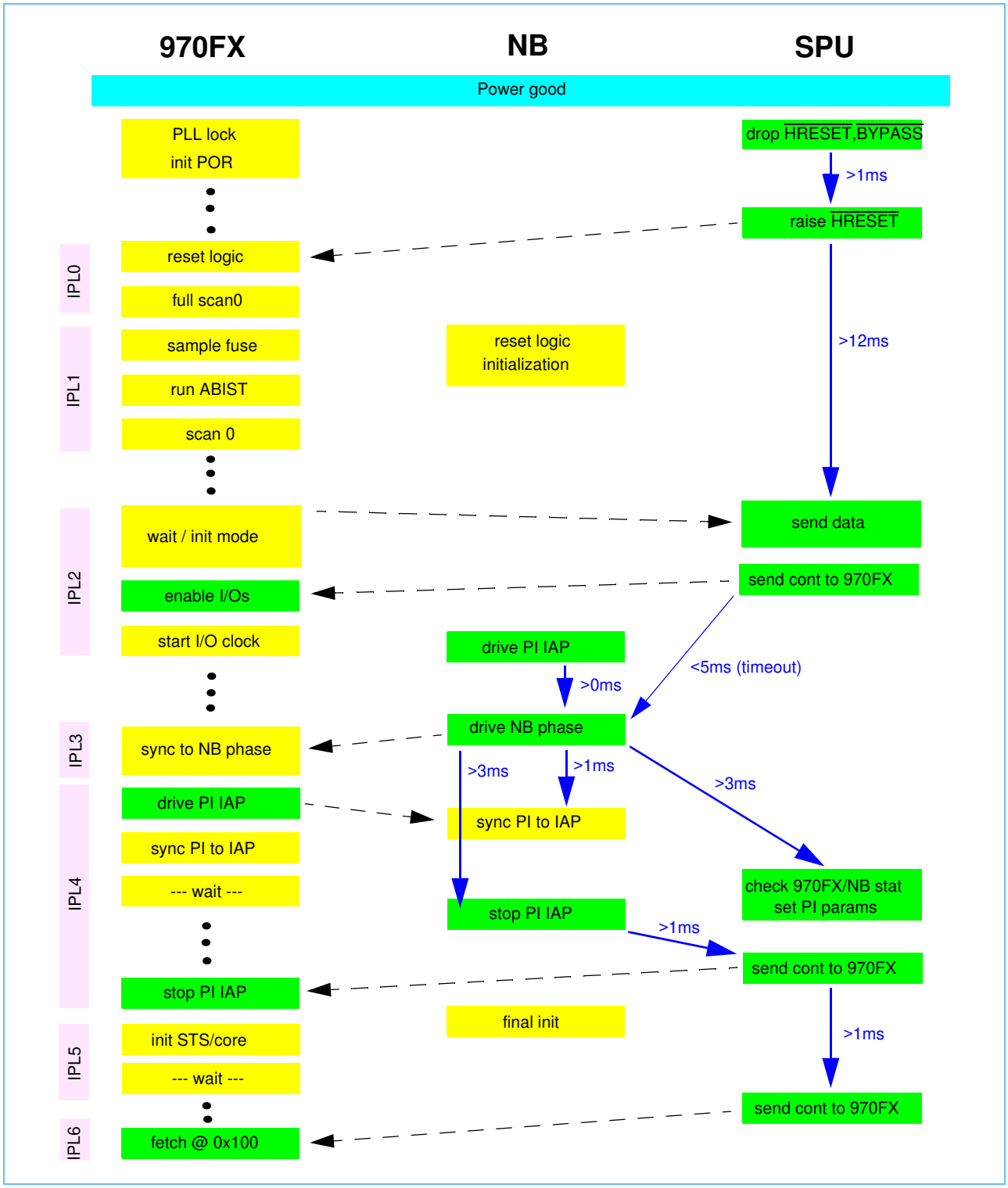


*Table 1-2. POR Procedure in Detail, Debug Mode (GPULDBG pin pulled high)*

| APC | IPL Step | 970FX | North Bridge | SPU |
|---|---|---|---|---|
| | | sync PLL | | drop $\overline{\text{HRESET}}$, $\overline{\text{BYPASS}}$ |
| | IPL | reset POR, high-Z I2CGO | reset NB | sample I2CGO, raise $\overline{\text{BYPASS}}$ |
| | | | | raise $\overline{\text{HRESET}}$ |
| | -1 | state PORWAIT | | |
| 00 | IPL0 | RSTFRL | | send continue |
| 01 | | [FULL] SCAN0 | | send continue |
| 02 | RASINIT | SAMPLEFUSE | | send continue |
| 03 | IPL1 | SCABISTINIT | | send continue |
| 04 | | SCAN0 | | send continue |
| 05 | | DABISTINITL1 | | send continue |
| 06 | | DABISTINITL2 | init NB | wait 1 us, send continue |
| 07 | CHIPINIT | SCAN0 | | send continue |
| 08 | | DABISTINITL2 | | send continue |
| 09 | | SCAN0 | | send continue |
| 10 | | WAIT | | send continue |
| | IPL2 | | start WIAP, start phase | send mode data |
| | CHIPCFG | | | send continue |
| 11 | | DRIVEIOS | | send continue |
| 12 | IPL3 | SYNCPHASE | | send continue |
| 13 | CHIPSYNC | STARTZIOCLK | | send continue |
| 14 | | TOGGLEWIAP | | send continue |
| 15 | | SYNCRIAP | | send continue |
| 16 | IPL4 | WAIT | | wait for 970FX PI synced |
| | IOSYNC | | | start NB PI sync |
| | | | sync RIAP | |
| | only in PI-mode | | stop WIAP | check EIs, set EIs params, stop NB WIAP., send continue. |
| 17 | | TOGGLEWIAP | | send continue |
| 18 | | STARTCORECLK | | send continue |
| 19 | | INITGUS | final init | wait for 1us, send continue |
| 20 | IPL5 | INITCORE | | send continue |
| 21 | COREINIT | WAIT | | send continue |
| | | | | final system init. send cont. |
| 22 | | STARTGUSCLK | | send continue |

*Table 1-2. POR Procedure in Detail, Debug Mode (GPULDBG pin pulled high)*

| APC | IPL Step | 970FX | North Bridge | SPU |
|---|---|---|---|---|
| 23 | IPL6 | INITGUS | | send continue |
| 24 | FETCHINIT | SRESET | | send continue |
| | | <fetch @ 0x100> | | |

## 1.5 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 $\overline{\text{HRESET}}$ is raised the program counter (APC) is cleared and the POR state machine first enters the PORWAIT state (see *Table 1-2* on page 13). In debug mode, the POR state machine will not advance until a continue is sent from the SPU. In normal (non-debug) mode the state machine will advance to the next cycle without a continue. The state machine then starts fetching the instruction from the POR sequence register pointed to by APC (latch state), decodes it (decode state) and starts the execution of the corresponding auxiliary state machine (go state). The state machine then waits until the auxiliary machine signals completion or a continue command is received through JTAG or $I^2C$. The POR program counter is then incremented and the state machine loops to the latch state.

There is also a “debug” mode available that is initiated by asserting “GPULDBG” high. In this mode, the 970FX will not automatically step through the preprogrammed sequence but instead will pause after each step to allow service procesor or JTAG intervention. In this mode, the service processor must initiate a continue after each POR step to continue the sequence.

Most developers will choose to use the debug mode to develop the code for their Service Processing Unit. It requires a few more milliseconds to send each continue command but provides greater visibility to the machine state from the Service Processor.

### 1.5.1 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 all 0's to SCOM address 0x400101. For more information on SCOM writes, see *Appendix A.3.2 SCOM (Register) Read/Write* on page 31.

**Note:** The first continue command written to the 970FX after HRESET will not generate an $I^2C$ acknowledge and has to be sent a second time! The second and all subsequent continue commands will receive the normal $I^2C$ acknowledge. This issue is included in the *IBM PowerPC 970FX RISC Microprocessor Errata List for DD 3.X*.

### 1.5.2 POR Status Register

The POR status register is located at SCOM address 0x400000. This register indicates the current POR program counter and whether or not the current instruction has completed. For more information on SCOM read operations see *Appendix A.3.2 SCOM (Register) Read/Write* on page 31.

### 1.5.3 Mode Ring

The mode ring is a bitstream that customizes the processor. Most systems can use the default mode ring provided in *Table 1-3* on page 18, however, some customization may have to take place based on the selected bus mode and the memory range of the device where the initial processor fetch will get data from.

Initializing the mode ring is handled by the following procedure, also described in more detail in *Section 1.6.4.2* beginning on page 18. There are 3 basic steps:

1. Initializing the Phase Sync Control Register (SCOM register 0x800006) with the value 0x0000.02F2.8000.0000.
2. 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 $I^2C$, you can send up to 8 bytes at a time.
3. Then a TAP Reset will return the TAP controller to the idle state.

The details of these steps, and their relationship to equivalent JTAG operations is documented in *Appendix A.3.3 Non-Register Commands* on page 33.

## 1.6 Detailed view

This detailed description of the 970FX power on reset sequence will assume the “debug” mode (GPULDBG pin held high) to show each individual step in the POR state machine sequence.

### 1.6.1 Power and Clocks

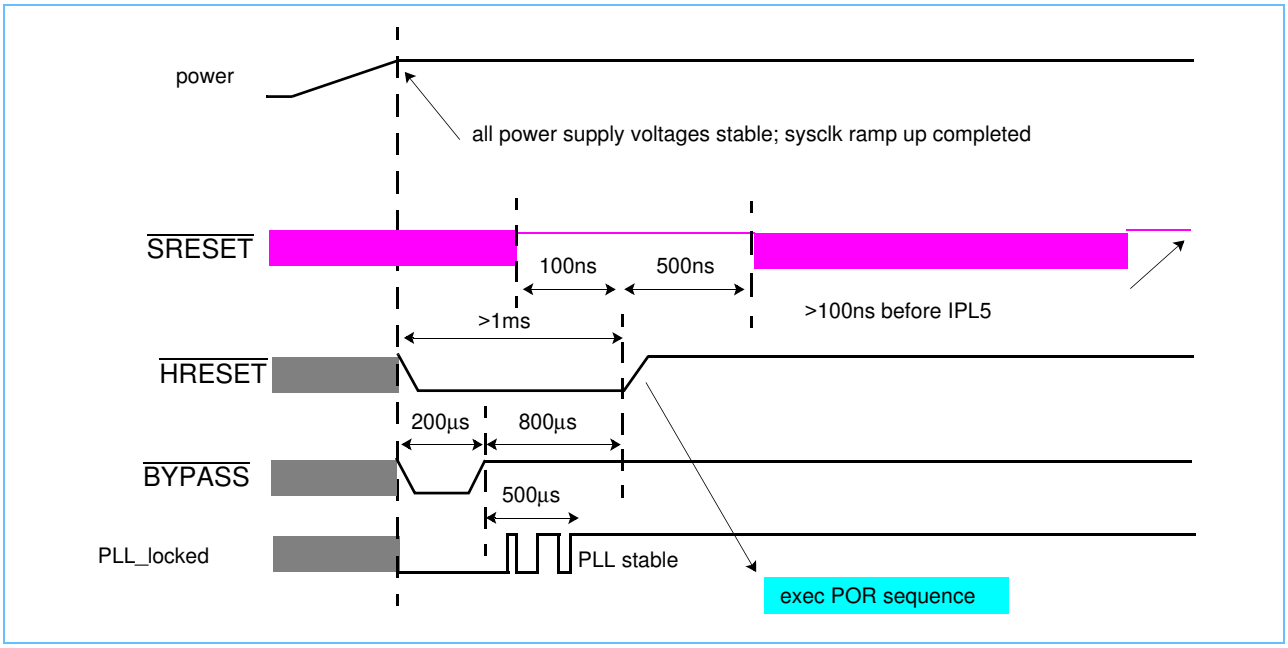
#### *1.6.1.1 Power and Clock Ramping for 970FX*

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

If for bring-up and debug purposes, the PLL was to be used in bypass-mode, (for example, the processor frequency equals the system clock input frequency), then the $\overline{\text{BYPASS}}$ signal should be held low and the $\overline{\text{SRESET}}$ forced low around the $\overline{\text{HRESET}}$ low to high transition (see *Figure 1-2*). Note that in bypass-mode the BUS interface of the chip will not train or work properly, and hence the chip will never fetch. This is only intended for debugging the chip at a very low power (for example, during assembly test).

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 $\overline{\text{HRESET}}$ for 1ms. This will reset the PLL and allow it to lock during the remaining 800uSec required for $\overline{\text{HRESET}}$ to complete. After $\overline{\text{HRESET}}$ goes high the automatic power-on sequence begins with IPL0.

*Figure 1-2. $\overline{HRESET}$, $\overline{SRESET}$, $\overline{BYPASS}$ timing for the PowerPC 970FX*




### 1.6.2 IPL0

#### *1.6.2.1 Release $\overline{HRESET}$*

Once $\overline{HRESET}$ is released and the first continue command is issued to the 970FX, 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 1-2.

In debug mode, 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 1-2.

**Note:** The first continue command written to the 970FX after $\overline{HRESET}$ will **not** generate an $I^2C$ acknowledge. Subsuquent continue commands will get the normal $I^2C$ acknowledge. This is listed in the 970FX Errata.

#### *1.6.2.2 Scanning of Boundary Scan Latches*

One of the rings initalized by step 1 is the boundary scan latches. During this step, some of the 970FX 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 $\overline{QREQ}$. This pin is used to indicate a request to sleep. The SPU should avoid creating a situation where the potential toggling of $\overline{QREQ}$ 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 1-2. Once the fuses are copied the POR sequence automatically moves to IPL1.

#### *1.6.2.3 Increasing I²C Clock Speed*

At this point in the POR sequence, you may (optionally) write SCOM 0x600400 with the value 0x0083F000.00000000 to increase I²C speed to 100 KHz. Without this SCOM initialization I²C bus speed will be limited to 50 KHz.

**Note:** All I²C accesses including this one, must use 50KHz I²C clock rate. The I²C clock speed may only be increased after this SCOM write is complete. Using a higher I²C clock rate may cause unreliable I²C operation.

### 1.6.3 IPL1

#### *1.6.3.1 Chip Initialization*

IPL1 begins by initializing the arrays. This is done using the ABIST engine to initialize the core, VPU, STS, and pervasive array. Finally the latches are initialized.

As a result of the latch initialization, the fence between the core and the storage subsystem and between the storage subsystem and the elastic interfaces are raised. The core is initialized.

From debug mode the SPU should issue 8 more continue commands to complete all the steps in IPL1. This should leave the POR program counter at instruction 10 (WAIT).

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.

#### *1.6.3.2 Verifying Chip initialization in IPL1 is complete*

The SPU can check the status of IPL1 by reading the POR status register at SCOM 0x400000. When bits [24:28] of this register (the POR APC) contains the value 0x0a, the chip is ready to start IPL2.

### 1.6.4 IPL2

#### *1.6.4.1 Initalization of the Phase Sync Control Register*

Before the mode ring can be loaded you must first initialize the Phase Sync Control Register (SCOM register 0x800006) by writing the value 0x0000.02F2.8000.0000.

**Note:** This register **must** be initialized with this value before loading the mode ring, otherwise the mode ring load will fail.

### *1.6.4.2 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 procedure to load the mode ring is shown in *Table 1-3*. The lines listed in that table should be sent via $I^2C$ to the 970FX with a stop command at the end of each line. The ***p0-p3*** and ***h1-h4*** customization values are described in *Section 1.6.4.4 Mode Ring Customization for PLL Multiplier and Bus Ratio (p0, p1, p2, p3)* and *Section 1.6.4.3 Mode Ring Customization For HIOR* respectively.

*Table 1-3. Sample Mode Ring Loading Procedure*

| Content (Hex) |
|---|
| 08 40 52 DF 00 |
| DE 40 52 00 80 C0 0F |
| 02 40 52 03 |
| BE 40 52 00 00 00 00 00 00 00 02 |
| BE 40 52 00 00 00 00 00 02 00 00 |
| BE 40 52 00 00 05 01 00 00 C3 01 |
| BE 40 52 00 08 00 08 00 DE B0 21 |
| BE 40 52 00 00 04 00 00 00 01 00 |
| BE 40 52 00 00 00 00 00 00 00 00 |
| BE 40 52 00 ***p0 p1 p2 p3*** 00 00 00 |
| BE 40 52 00 00 00 00 00 00 00 00 |
| BE 40 52 00 00 00 00 00 00 00 00 |
| BE 40 52 00 00 00 C0 07 00 00 00 |
| BE 40 52 00 00 00 00 00 00 00 00 |
| BE 40 52 00 00 00 00 00 00 00 00 |
| BE 40 52 00 00 00 00 00 00 00 00 |
| BE 40 52 00 01 00 00 00 80 04 00 |
| BE 40 52 ***h1 h2 h3 h4*** 00 00 00 00 |
| BE 40 52 00 00 00 00 00 00 00 00 |
| BE 40 52 00 00 00 00 00 00 00 00 |
| BE 40 52 00 00 40 00 00 00 00 00 |
| BE 40 52 00 00 c0 70 00 50 C8 10 |
| BE 40 52 00 00 00 01 00 00 00 00 |
| BE 40 52 14 00 00 00 00 00 00 a0 |
| BE 40 52 7E 00 00 00 00 00 00 00 |
| BE 40 52 00 00 18 00 00 00 00 00 |
| BE 40 52 00 00 00 00 00 00 08 40 |
| BE 40 52 00 10 00 00 00 20 00 7B |
| BE 40 52 03 00 00 00 00 00 00 00 |
| BE 40 52 00 00 00 00 00 00 00 00 |
| BE 40 52 00 00 00 00 1C FC 00 00 |
| DE 40 52 00 00 00 00 00 3B 00 00 |
| 03 40 52 1F |
| Bytes Indicated by ***p0-p3*** should be replaced per *Table 1-4* according to Bus Ratio and PLL Settings.<br>Bytes indicated by ***h1-h4*** should be replaced per *Section 1.6.4.3* beginning on page 19 according to desired HIOR. |

### *1.6.4.3 Mode Ring Customization For HIOR*

Certain fields in the mode ring may need to be customized for your application.The HIOR register is a 64-bit register that defines the base physical address for the interrupt vectors, including the reset vector. The processor will begin executing code at the physical address of the HIOR register plus the reset vector, 0x100. For example, if the HIOR is initialized to 0x0000.0000.FFF0.0000 in the mode ring, the first fetch will be made at the physical address 0x0000.0000.FFF0.0100.

The HIOR register is included in the mode ring shown in *Table 1-3*. The bytes to be replaced by the desired HIOR value are indicated by ***h1-h4***. The correct value for these bytes is derived from the HIOR in the following manner:

- h1 = HIOR[43] * 128
- h2 = HIOR[35:42]
- h3 = HIOR[27:34]
- h4 = HIOR[22:26]

For example, an HIOR of 0x0000.0000.FFF0.0000 as shown above, will map to h1=0x00, h2=0xF8, h3=0x01, h4=0x00.

### *1.6.4.4 Mode Ring Customization for PLL Multiplier and Bus Ratio (p0, p1, p2, p3)*

The mode ring must also be modified to match the system bus ratio and PLL Mode. Use *Table 1-4* to determine the correct byte sequence to insert into the mode ring. The bytes in that table replace the ***p0 - p3*** bytes in *Table 1-3*.

**Note:** These mode ring settings do not affect or override the PLL multiplier or bus ratio settings. They are used to support correct frequency scaling for the power tuning features. Incorrect configuration of these bytes will not affect full frequency operation but will cause bus errors during power tuning (for example, f/2) operation.

*Table 1-4. Mode-Ring Content Dependent on the Bus Ratio and PLL Settings (p0, p1, p2, p3)*

| Bus Ratio | PLL x8 | PLL x12 |
|---|---|---|
| 2:1 | 0x98, 0xB0, 0x00, 0x00 | Not Recommended |
| 3:1 | Not Recommended | 0x90, 0x57, 0x83, 0x00 |
| **Note:** The combinations of Bus Ratio and PLL Mode shown as Not Recommended are likely to cause performance degradation caused by internal synchronization delays. For best performance use only the combinations shown. | | |

### *1.6.4.5 PSYNC Fix Needed for DD 3.X*

Some DD 3.X 970FX hardware is affected by an errata requiring resynchronization of the PSYNC input to the internal “Time 0” signal. (Also referred to as CAP, Clock Alignment Procedure).

After the mode ring has been scanned into a DD 3.X part, the SPU should write SCOM 0x800006 with bit [1] = 1, then write SCOM 0x800006 with bit [1] = 0. This will cause the CAP to resynchronization the internal Time0 signal with the PSYNC input.

**Note:** Failure to implement this SCOM write on DD 3.X hardware may result in parity errors or transfer handshake null errors on the first processor instruction fetch.

#### *1.6.4.6 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 970FX, this pattern is documented in the *PowerPC 970FX RISC Microprocessor User Manual* in Section 11.3 “Bus Initialization, Configuration, Power Management and Test.”

### 1.6.5 IPL3

#### *1.6.5.1 Synchronization of all PowerPC 970FXs to PSYNC*

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

#### *1.6.5.2 Send Continues*

At this point the service processor sends 4 “continue” commands to the 970FX to start IPL4. The continue command is triggered by writing any 64-bit value (for example, all ‘0’s) to SCOM address 0x400101. At this point the POR status register bits [24:28] should read a value of 0x10, indicating instruction 16.

### 1.6.6 IPL4

#### *1.6.6.1 Wait for IAP to Complete*

The 970FX(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.

#### *1.6.6.2 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 *IBM PowerPC 970FX RISC Microprocessor User’s Manual* in Section 11.3 “Bus Initialization, Configuration, Power Management and Test.”

#### *1.6.6.3 Stop IAP Pattern*

With the Processor Interface bus initalized and ready for operation, another SCOM write to Processor Interface Mode Register (SCOM address 0x046A00, *IBM PowerPC 970FX RISC Microprocessor User’s Manual* Section 11.6.3.9) will halt the IAP pattern and quiesce the bus. This should also be done for the North Bridge.

#### *1.6.6.4 Send Continue*

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

### 1.6.7 IPL5

#### *1.6.7.1 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 970FX.

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 southbridge, or may require copying the contents of a service processor ROM into the system DRAM.

Once this final initialization is complete, three more continues will take the 970FX to IPL6.

#### *1.6.7.2 IPL6*

#### *1.6.7.3 Start STS Clock*

The last step Fetch Init (IPL6) starts the STS clock, resets the storage interface of the 970FX and starts fetching instructions at address HIOR + 0x100. The first continue command will start the GUSCLK. This continue must be completed before setting the PI parameters.

#### *1.6.7.4 Initialize PI Parameters*

Once the IAP has been completed successfully, the service processor must initialize the PI bus for correct operation by configuring the STATLAT, SNOOPLAT, SNOOPACC, and APSEL parameters. These parameters are documented in the *IBM PowerPC 970FX RISC Microprocessor User's Manual* in section 11.3 "Bus Initialization, Configuration, Power Management and Test."

#### *1.6.7.5 STS Init and SRESET*

Once the PI parameters have been set, one more continue command will initiate the $\overline{\text{SRESET}}$ and cause the processor to begin fetching instruction.

## 1.7 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.

### 1.7.1 Automatic Array Recovery

The 970FX 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 1-5* on page 22 and *Table 1-6* on page 23 allows the 970FX 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 970FX should be set to checkstop.

*Table 1-5. Enabling Automatic Array Recovery Modes, By Array*

| Array/Error Source | HID Register Setting | Error Mask Register (SCOM 0x030400) | Machine Check Register (SCOM 0x030901) | Checkstop Register (SCOM 0x030800) |
|---|---|---|---|---|
| I-Cache / ITAG | HID1[11:12] = '11' | 0x030400[0:2] = '000' | 0x030901[0:2] = '000' | 0x030800[0:2] = '000' |
| I-ERAT | HID1[13] = '1' | 0x030400[3] = '0' | 0x030901[3] = '0' | 0x030800[3] = '0' |
| L2 UE | N/A | 0x030400[4] = '0' | 0x030901[4] = '0' | 0x030800[4] = '1' |
| Logic UE | N/A | 0x030400[5] = '0' | 0x030901[5] = '0' | 0x030800[5] = '1' |
| D-Cache | HID5[50] = '0'<br>HID4[39:41] = '000' | 0x030400[6] = '0' | 0x030901[6] = '0' | 0x030800[6] = '0' |
| DTAG | HID4[34:36] = '000' | 0x030400[7] = '0' | 0x030901[7] = '0' | 0x030800[7] = '0' |
| D-ERAT | HID4[29:31] = '000' | 0x030400[8] = '0' | 0x030901[8] = '0' | 0x030800[8] = '0' |
| LSU-TLB | HID4[44:48] = '00000' | 0x030400[9] = '0' | 0x030901[9] = '0' | 0x030800[9] = '0' |
| LSU-SLB | HID4[53:54] = '00' | 0x030400[10] = '0' | 0x030901[10] = '0' | 0x030800[10] = '0' |

*Table 1-6. L2 Array Recovery Details*

| L2 Error | STS Mode Register (SCOM 0x043000) | Error Mask Register (SCOM 0x040401) | Checkstop Register (SCOM 0x040801) | Notes |
|---|---|---|---|---|
| L2 CE | 0x43000[50] = '0' | 0x040401[42] = '0' | 0x040801[42] = '0' | |
| L2 UE | 0x43000[50] = '0' | 0x040401[43] = '0' | 0x040801[43] = '1' | Uncorrectable Error |
| L2 Special UE | | 0x040401[44] = '0' | 0x040801[44] = '1' | Error outside L2, but detected in L2 |
| L2Dir | | 0x040401[45] = '0' | 0x040801[45] = '0' | |
| L2Dir Checkstop | | 0x040401[46] = '0' | 0x040801[46] = '1' | |
| L2 Hang Detect | | 0x040401[47] = '1' | 0x040801[47] = 'X' | |
| L2 STQ | | 0x040401[48] = '0' | 0x040801[48] = '1' | Store queue error |
| Logic UE | | 0x040401[49:51] = '0' | 0x040801[49:51] = '1' | |
| L2$ Quad CE Threshold | | 0x040401[52:56] = '00000' | 0x040801[52:56] = '00000' | |
| L2Dir multiple errors within 2 hang pulses | | 0x040401[57] = '0' | 0x040801[57] = '0' | |

## 1.8 Debugging Tips

### 1.8.1 Verifying I$^2$C Operation

Some microcontrollers used as SPUs for 970FX will require level shifters to avoid levels on the 970FX pins beyond OVdd

> **Note:** I$^2$C pins should never exceed voltages beyond the OVdd tolerances provided in the *IBM PowerPC 970FX RISC Microprocessor Datasheet*. Devices may be permanently damaged if inputs exceed allowable maximum voltages.

Proper I$^2$C operation can be confirmed by attempting to read the POR status register at SCOM address 0x400000. This register should be initialized to '0' after $\overline{\text{HRESET}}$ is released and should increment after sending a continue command.

Most problems with I$^2$C communications are caused by incorrect pullup resistor values, or I$^2$C clock frequency. Note that 970FX I$^2$C clock frequency should not exceed 50KHz unless your SPU code implements the SCOM initialization described in *Section 1.3.1* beginning on page 9 and *Section 1.6.2.3* beginning on page 17.

### 1.8.2 SCOM Access to Uninitilized Units

If SCOM access is attempted to units before clocking is enabled, the SCOM engine will fail and the SCOM state machine will hang. Before attempting any SCOM access, 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 $\overline{\text{HRESET}}$.

### 1.8.3 Use of $\overline{\text{SRESET}}$

A POR sequence that causes an error is likely to require a complete restart from $\overline{\text{HRESET}}$ and $\overline{\text{BYPASS}}$. Unlike earlier PowerPC processors, the 970FX 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.

#### *1.8.3.1 Diagnosing IAP Errors*

Errors and warnings during IAP are recorded in the Processor Interface Status Register, which is documented in *Table 1-7* on page 25. Note that bit [36] does not indicate that IAP was successful, only that the IAP state machine ran to completion. The IAP state machine will not complete if any fatal error occurs (bits [39, 42 or 45]). The other bits must be interpreted per the following sections.

*Clock period too large, Bit [38]:*

This error would be expected for Processor Interface busses at the slowest possible speeds (below approximately 233MHz). It indicates that the system is using the highest possible delay count for deskew. The error is not fatal, but should be corrected by running at higher bus frequency if possible.

*No first valid10 found, Bit [39]:*

This error is fatal, the IAP state machine never found the leading 10 edge of the IAP. If the connection between the 970FX and the North Bridge is valid, the processor or North Bridge may be bad, or the termination may be set incorrectly.

*No first valid 01 found Bit [40]:*

This error may occur for very slow bus speeds (below approximately 233MHz). Although it is not considered a fatal error, no per bit deskew was performed, so POR cannot be completed. Check termination settings or try a higher bus ratio.

*Data bit deskewed to maximum delay, Bit [41]:*

This warning error indicates that at least one bit on the bus has been deskewed to the maximum number of delay elements. It may be tolerable on systems with slow bus speeds. Check termination settings, trace length violations, or signal integrity issues.

*No flag 0 found, Bit [42]:*

This is a fatal error, no common data eye could be found. May be caused by using balanced coding for IAP. Check termination settings, signal integrity.

*No flag 1 found, Bit [43] and/or no flag 2 found, Bit [44] and/or Final Delay too large, Bit [47]:*

This error is critical, it may be cause if the clock period is too large. Check signal integrity.

*Final values fail, Bit [45]:*

This is a fatal error indicating that no data pattern could be found after bus was deskewed. Check termination settings, signal integrity, clocking.

*Clock Delay negative, Bit [46]:*

This is a warning indicating that some data bits are arriving too early (prior to the clock). Check termination settings, trace length violations, or signal integrity. You may be able to workaround this warning error by rerunning IAP with a larger value in the data-windage field of the Processor Interface Mode register.

*Table 1-7. Processor Interconnect Status Register*

| Address: | x046B01 (PI inputs PII)<br>x'0F6B00' (PI not used)<br>x'0F7B01' (PI outputs PIO) |
|---|---|
| Type: | RO |
| Reset: | Reset to '0's by Power-On Reset |
| Bit(s) | Description |
| 0:35 | Not implemented |
| 36 | Reduced IAP (RIAP) done (0b1 = done; clears when RIAP is dropped); status only |
| 37 | RESTPASS (valid when ESTMODE='1'); status only |
| 38 | Clock period too large (greater than delay line); critical |
| 39 | No first valid '10' found; fatal |
| 40 | No first valid '01' found; critical |
| 41 | Data bit deskewed to maximum; warning |
| 42 | No valid flag – 0; fatal |
| 43 | No valid flag – 1; critical |
| 44 | No valid flag – 2; critical |
| 45 | Final values fail; fatal |
| 46 | Final delay value is negative; warning |
| 47 | Final delay value is too large; critical |
| 48:55 | Calculated clock period (8 bits; in delay elements); status only |
| 56:63 | Inserted I/O clock delay (8 bits; in delay elements); status only |

Status indication key:
Warning – Indicates a possible system design problem.
Critical – Reduced IAP was performed, meaning interface sampling point is suboptimal.
Fatal – Interface cannot perform even a reduced IAP.

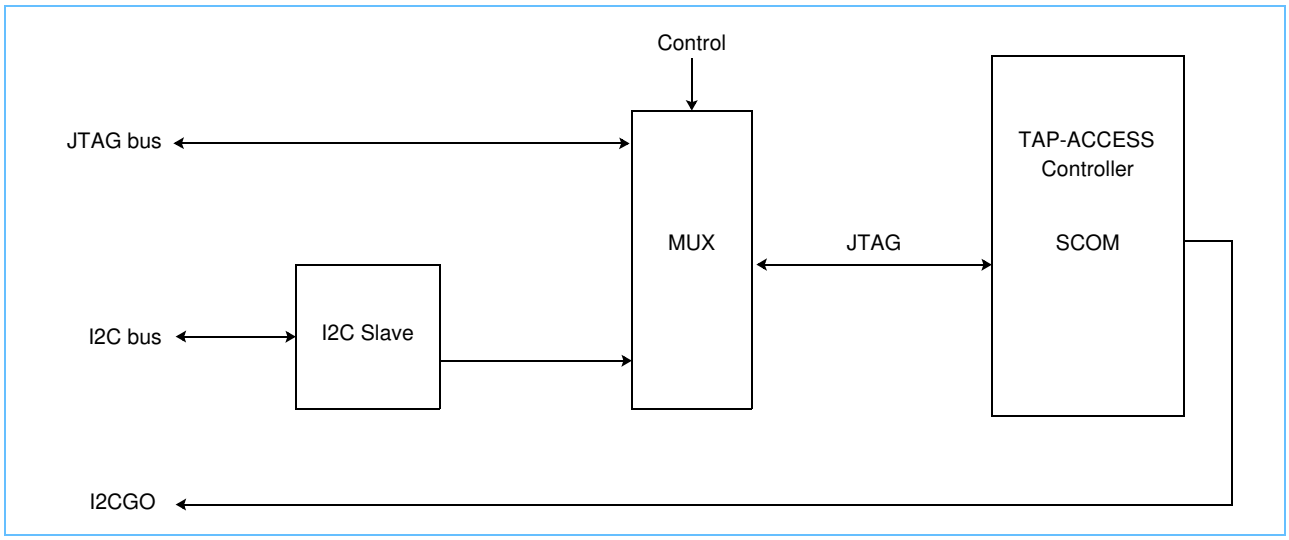
# Appendix A. I2C Details

## A.1 I2C Purpose

I2C (Interconnect for Integrated Circuits) 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 tap controller that interfaces with Access or other IEEE 1149.1 compatible device in order to perform reading, writing, and scanning of registers within a chip.

The basic concept of merging the off-chip I2C and JTAG busses is shown in *Figure A-1*.

*Figure A-1. Merged JTAG and I2C Interfaces*




### A.1.1 Uses for I2C Slave

SCOM reads/writes: The I2C Slave can translate data sent across the I2C bus in order to read or write registers via native JTAG commands and then return the data back on the I2C bus if requested. The I2C slave has a 12-byte buffer that holds data sent from the master to the slave, as well as data retrieved from registers during a read sequence. All data is 8 byte aligned due to the implementation of the TAP controller.

Native JTAG Function: The I2C slave can also be used to interpret I2C data and send native JTAG commands to a TAP controller. The I2C slave monitors the Attention signal sent from the TAP and optionally uses this as part of the decision to acknowledge data or the slave address depending upon the read/write bit of the I2C address. This can be enabled/disabled for testability/lab functions to make communication easier.

Through the use of primitive decodes the I2C slave will return the data in its 12-byte register without doing any JTAG activity or compromising the previous data from capturing TDO (see also IR Status Read example).

1. I2C standard (IIC) for a serial bus. For more information see: http://www-us2.semiconductors.philips.com/i2c/.

### A.1.2 I2CGO Pin

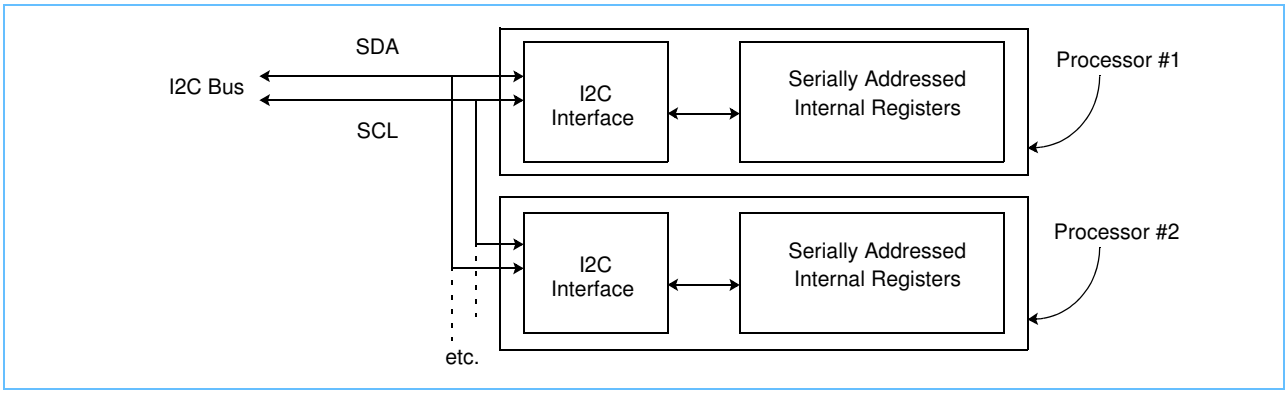
The I2CGO open collector pin is used to prevent access collisions between the JTAG and $I^2C$. If the level of the pin is low, only JTAG should access the 970FX. The $I^2C$ can make use of the interface if the level is high.

The I2CGO can be set by either the JTAG or $I^2C$ through SCOM write commands. When set by an SCOM write, the pin will hold its old value until the TAP controller enters either the RESET or the IDLE state.

**Note:** The I2CGO pin is only a signal between the JTAG and I2C interfaces. To change the state of I2CGO, write to SCOM 0x400200.

### A.1.3 $I^2C$ Overview

*Figure A-2. $I^2C$ Protocol*

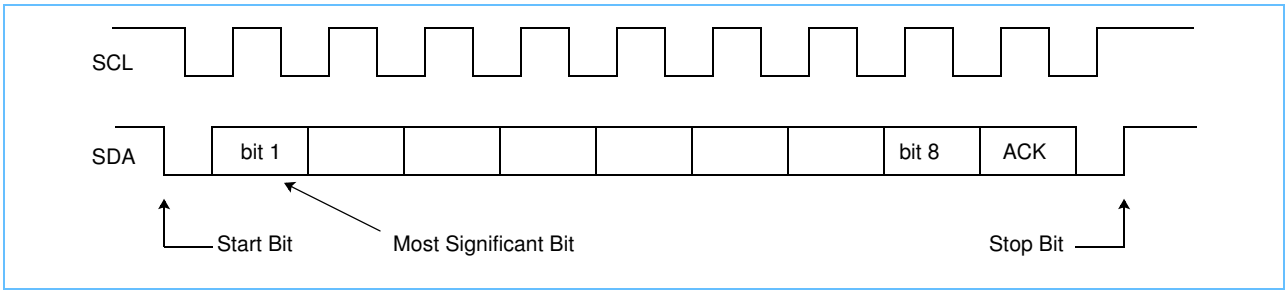



$I^2C$ protocol, as defined, supports only register read and write operations. This specification will deal only with the 7-bit slave addressing mode, allowing $2^7$ slaves to be directly addressed. The protocol implies that an addressed slave responds to a read/write by addressing successive bytes serially, starting in memory where it is defined by the particular slave.

**Implementation note** – $I^2C$ to JTAG slave implementation extends direct-addressing to $2^{23}$, operating on 8-byte aligned registers. Additionally, operations other than register read/write, which are supported by the access implementation of JTAG, are available.

#### *A.1.3.1 $I^2C$ Bus Operation*

*Figure A-3. $I^2C$ Bus Operation*



The $I^2C$ bus consists of two wires: SDA (serial data) and SCL (serial clock). These rest at ‘1’ while the bus is quiescent. Any bus master might initiate a message transfer with a start bit (SDA 1->0 while SCL=1). $I^2C$ operates on packets, each one-byte wide, each followed by an acknowledge bit (0=good ack). When the message begins, the master owns driving both SCL and SDA until the ACK bit when the slave owns driving SDA (on writes only see $I^2C$ Spec. for further details). Each SDA data/ack bit must remain stable ‘1’ or ‘0’ while SCL clocks (0->1->0); i.e., SDA setup prior to SCL rising edge, held after falling edge, transition while SCL is low.

The slave may temporarily 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.

Packet bits are transmitted with the most significant bit (MSb, bit 1) first. The start byte bits 1:7 address a particular slave chip. Bit 8 of the start byte is a read/write bit. Writes progress with the master sending bytes and the slave acknowledging. A read begins as the master writes the start byte, but the dataflow reverses after the slave acknowledges the start byte. Now the slave is sending packets and the master is Acking. The slave continues to send until the master Ack=’1’. All transmissions end with a stop bit (SCL=’1’ while SDA 0->1). The bus is quiescent again following the stop bit.

#### *A.1.3.2 Deviations from $I^2C$ Standard*

Following is a list of deviations, not limitations, of the current design.

- Does not support 10-bit addressing mode.
- Does not support general call address.
- Does not support auto-increment of slave byte address. The POR-default byte address is x‘000000’ (an SCOM address).

### A.1.4 JTAG Overview

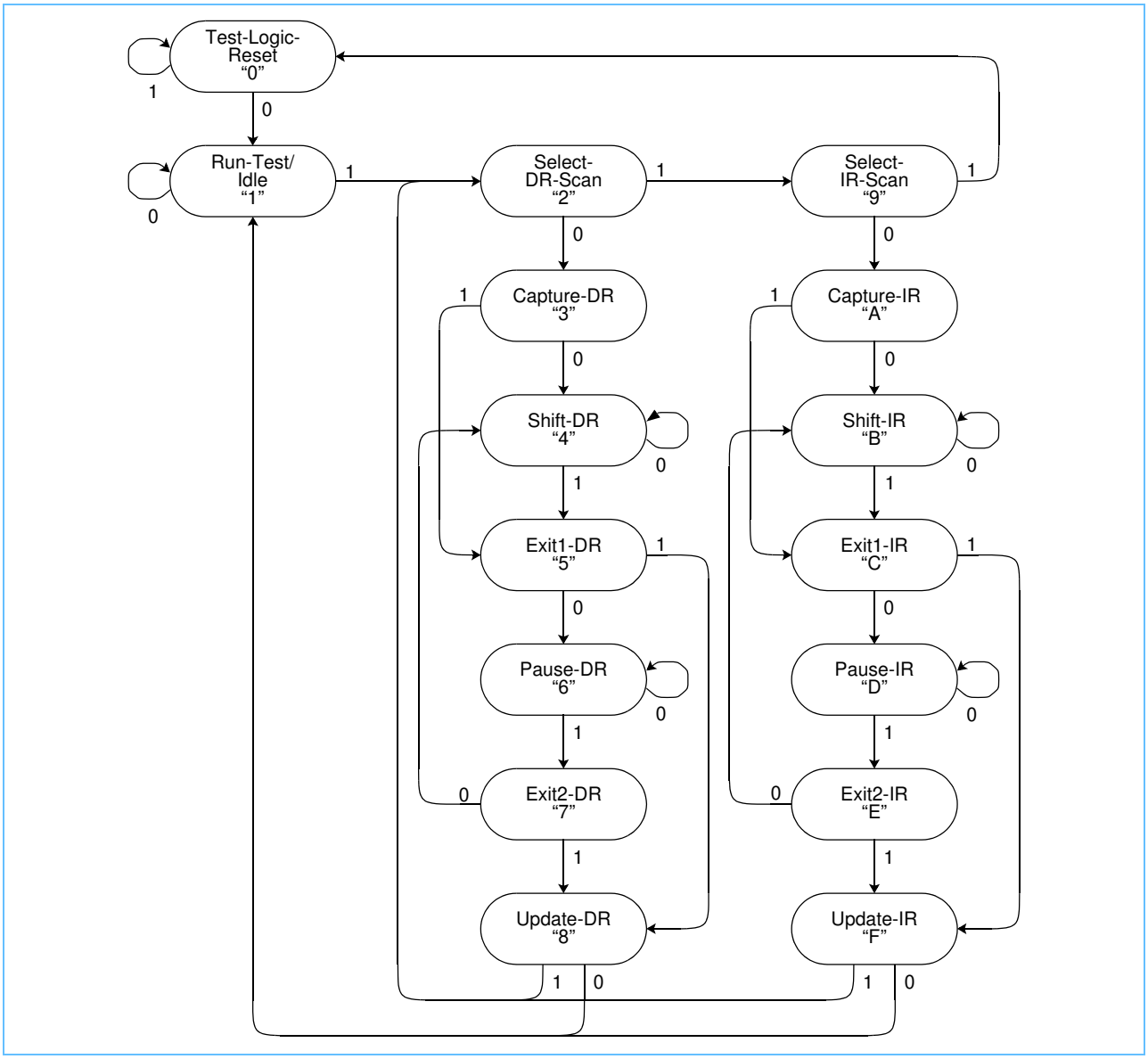
The IEEE 1149.1 defines a 5-wire interface called a Test Access Port (TAP) for communicating with boundary scan architecture.

- TCK “test clock,” rising edge causes TMS and TDI to be sampled by Access.
- TMS “test mode select,” the value of TMS during the rising edge of TCK causes a state transition in the TAP controller.
- TDI “test data in,” is the serial data input to Access.
- TDO “test data out,” is the serial data output from Access.
- TRST “test logic reset,” causes an asynchronous reset of the test logic (TAP->TestLogicRst).

#### *A.1.4.1 TAP Controller*

Access is a specific implementation of the IEEE spec. TMS and TCK control the TAP controller. It is necessary to understand ShiftIR and ShiftDR states for this discussion.

*Figure A-4. IEEE/Access TAP Controller*



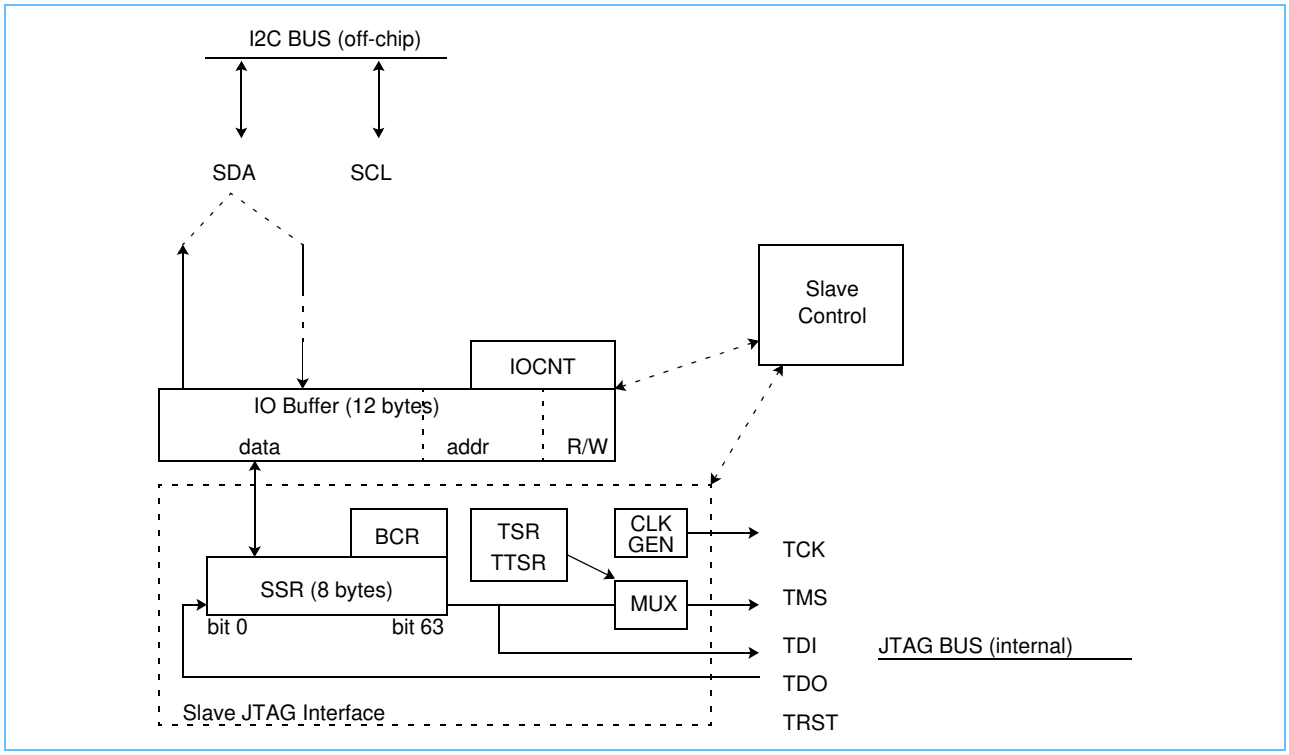

- ShiftIR: The instruction register inside Access is serially connected between TDI and TDO. While TMS is held '0', each TCK clock causes another bit to be shifted into the instruction register from TDI and IR status to be shifted out TDO.
- ShiftDR: One of many data registers (TDRs) is serially connected between TDI and TDO. While TMS is held '0', each TCK clock causes another bit to be shifted into the register from TDI and (old) data to be shifted out TDO.

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

## A.2 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.2.1 Slave Data Flow Overview

The physical interface handles $I^2C$ protocol and timing, providing a one-byte wide data interface along with associated handshaking signals. SDA is the name of the bi-directional Serial Data signal defined by $I^2C$. SCL is the name of the serial clock signal defined by $I^2C$.

The I/O Buffer (IOBUF) buffers successive bytes (to be) transferred to (from) the chip addressed as the slave chip by the $I^2C$ start byte. Data in the IOBUF is transferred in parallel one byte at a time to/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 $I^2C$ to TDI. During read operations the dataflow is from TDO to $I^2C$.

The Serial Shift Register (SSR) can receive or supply up to 8 bytes in parallel from/to the IOBUF. This data is then serially shifted to/from the JTAG following IEEE 1149.1 protocol. Test Data In (TDI) is name of the serial in data to JTAG. Test Data Out (TDO) is the name of the serial data out of JTAG.

## A.3 Programming and Examples

### A.3.1 Considerations for Concurrent Resource Use by I$^2$C and JTAG

When the on-chip scan resource (Access) will be shared by the I$^2$C and the chip-level JTAG I/O, special considerations are required to prevent either from interfering with the other:

- There is only one copy of the scan resource, the semaphore pin I2CGO is needed to quiesce the off-chip JTAG or I$^2$C slave while the other is active.
- Neither I$^2$C nor JTAG have the concept of “atomic” streams of commands, some operations might fail if the command stream is interlaced.
- The I$^2$C slave logic which is used to drive the on-chip JTAG MUX gives priority to the I$^2$C and can lock-out JTAG in certain circumstances if the “Enable Attention” TAP command is not used.

The Enable Attention TAP command is shown in *Section A.3.4.1 Primitive TAP Command (TAPCMD Addr(11:8)=”0000”)*.

### A.3.2 SCOM (Register) Read/Write

*Table A-1. SCOM Register Read/Write* (bit numbering, LSb=0)

| Start Byte | | SCOM Addr | | | SCOM Data (IO buffer) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I$^2$C Slave Addr (7 bits) | R/$\overline{W}$ | LSByte | Mid | MSByte | 8 bytes (LSByte,MSbit first) | | | | | | | |
| 7..4 970FX base addr<br>3..1 Proc ID | ‘0’ | 7..0<br>bit 0=odd Pty | 15..8 | 23..16 | 7..0 | | | | | | | 63..56 |
| | ‘1’ | | | | | | | | | | | |

This command format will perform a 64-bit SCOM read/write.

An SCOM operation begins with the start byte address (970FX I2C slave base address), optionally followed by up to 3 bytes of the SCOM address (LSByte, high-order bit first). Note that the SCOM addresses must be byte aligned. If a stopbit is received at this point only the portion of the SCOM address written will be altered, no other activity is initiated.

**Notes:**

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. I$^2$C slave implements a 24-bit SCOM address (23 down to 0, 0=LSb). Access/JTAG defines the LSb as a parity bit. The remaining 23 bits (23 down to 1) represent a byte-aligned address. The SCOM transfers are always 8-bytes wide, but each chip usually architects registers of varying widths that might not necessarily be aligned on 8-byte boundaries. The specific alignment of data within the I$^2$C SCOM read/write data field is chip implementation dependent.

**Write**
Data for a write operation follows, starting with the fourth byte (LSByte, high-order bit first). A stop or overflow condition following one or more data bytes kicks off the SCOM operation. The SCOM address is not auto-incremented. Writing more than 8 bytes of data will result in the data at the same SCOM address being over-written. 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.

**Notes:**

1. An overflow/underflow occurs when more than 8 bytes of data are written/read.
2. SCOM data (IOBUF) contents are unpredictable prior to the first R/W operation. If the same address is written successively, the data remains in the buffer – modified only by the data byte(s) rewritten.

**Read:**
For an SCOM read operation, a restart (a stop followed by a start) is then issued on the $I^2C$ interface with the R/W bit = 0b1. This triggers an SCOM read to be kicked off inside of the $I^2C$ hardware. The dataflow then reverses and the slave returns one data byte for each acknowledge (LSByte first). The SCOM address is not auto-incremented. If greater than 8 bytes are read, then the data will be repeated starting with the least significant byte.

**TAPCMD Example 1: SCOM READ of Register Address x'010203'**
The following delineates the sequence of $I^2C$ Commands needed to perform an SCOM read of register address x‘010203’, assuming the $I^2C$ slave address is b'1000000-' (the last bit (bit 8) is the R/W bit) and the data returned is x'AABBCCDD':

| | |
|---|---|
| wait for $I^2C$ bus idle (SCL=‘1’, SDA=‘1’) | |
| master start bit | |
| master byte | x‘80’ (slave ACK=‘0’, indicating recognition of its own address, bit 8=‘0’ indicates a WRITE to slave) |
| master byte | x‘03’ (slave ACK=‘0’ address is written to slave, least significant byte first) |
| master byte | x‘02’ (slave ACK=‘0’ address is written to slave, middle byte) |
| master byte | x‘01’ (slave ACK=‘0’ address is written to slave, most significant byte) |
| master stop bit | |
| wait for $I^2C$ bus idle | |
| master start bit | |
| master byte | x‘81’ (slave ACK=‘0’, bit 8='1' indicates a READ from slave) |
| slave byte | x‘DD’ (master ACK=‘0’, indicating it wants slave to send another byte) |
| slave byte | x‘CC’ (master ACK=‘0’, indicating it wants slave to send another byte) |
| slave byte | x‘BB’ (master ACK=‘0’, indicating it wants slave to send another byte) |
| slave byte | x‘AA’ (master ACK=‘1’, indicating it wants slave to stop sending data) |
| master stop bit | |

In the examples in this note “master” is an external $I^2C$ controller, “slave” is the I2CMasterSlave hardware.

**TAPCMD Example 2: SCOM WRITE of Register address x‘040506’ with Data**

The following delineates the sequence of $I^2C$ commands needed to perform an SCOM write of register address x‘040506’, assuming the data to write is x‘0011223344556677’:

| | |
|---|---|
| master start bit | |
| master byte | x‘80’ (slave ACK=‘0’, indicating recognition of its own address, bit 8='0' indicates a WRITE to slave) |
| master byte | x‘06’ (slave ACK=‘0’ address is written to slave, least significant byte first) |
| master byte | x‘05’ (slave ACK=‘0’ address is written to slave, middle byte) |
| master byte | x‘04’ (slave ACK=‘0’ address is written to slave, most significant byte) |
| master byte | x‘77’ (slave ACK=‘0’ data is written to slave, least significant byte) |
| master byte | x‘66’ (slave ACK=‘0’ data is written to slave) |
| master byte | x‘55’ (slave ACK=‘0’ data is written to slave) |
| master byte | x‘44’ (slave ACK=‘0’ data is written to slave) |
| master byte | x‘33’ (slave ACK=‘0’ data is written to slave) |
| master byte | x‘22’ (slave ACK=‘0’ data is written to slave) |
| master byte | x‘11’ (slave ACK=‘0’ data is written to slave) |
| master byte | x‘00’ (slave ACK=‘0’ data is written to slave, most significant byte) |
| master stop bit | |

### A.3.3 Non-Register Commands

*Table A-2. TAP Command* (bit numbering, LSb=0)

| Start Byte | | SCOM Address / TAP Command | | | Serial Shift Register Data (I/O Buffer) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $I^2C$ Slave Addr (7 bits) | R/$\overline{W}$ | LSByte | MIDbyte | MSByte | 8 bytes (LSByte,MSbit first) | | | | | | | |
| 7..4 970FX base addr,<br>3..1 Proc ID | ‘0’ | 7:TSR<br>6:TTSR<br>5..0:BCR | 15..12 x‘4’<br>11..8:x‘0’=TAPCMD<br>11..8:x‘1’=Null TAPCMD<br>11..8:x‘2’=Enable Attn<br>11..8:x‘3’=Disable Attn[1] | 23..16:<br>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 TAP CMD bits [10:8]. Bit [11] is reserved for future use.

### A.3.4 TAPCMD - TAP Command Bus

The tap command (TAPCMD) signal is 9 bytes wide. The first byte is a primitive “command” to be performed by the JTAG Scan Logic.

**TMS Select Register** (TSRL2: TAPCMD bit 0) determines whether MUX “C” will select SSR data to be shifted out to TMS, or a “constant value.” A ‘0’ indicates that the data in the SSR is a stream of TMS control information. A ‘1’ directs MUX “C” to select 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 Select Register** (TTSRL2: TAPCMD bit 1) provides a value on TMS during the Nth (terminal) TCK pulse. Intended for long scanning operations when the 64-bit width of the SSR is insufficient to complete the scan. TTSRL2=’0’ will bridge primitive operations, allowing the scan to continue with another primitive command. TTSRL2=’1’ terminates the operation by moving the JTAG TAP state inside Access out of the shift-DR or shift-IR 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 deglitch TRST.

**Bit Count Register** (BCRL2: TAPCMD bit 2:7) defines the number of TCK pulses to send, along with a sequence of information on TMS or TDI. A BCR value of b‘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=b‘111111’ is the terminal count, indicating that one TCK pulse is to be released.

#### *A.3.4.1 Primitive TAP Command (TAPCMD Addr(11:8)=”0000”)*

In this format, the LSByte of the SCOM address is interpreted as a TAP command causing bits to be shifted out to the TAP port on TMS or TDI and capturing TDO. This enables the $I^2C$ interface to perform any action supported by the JTAG logic.

A stop or overflow condition kicks off the BCR with its current value for a write operation and an overflow condition, stop or start with R/W address bit set to ‘1’ kicks off the BCR with its current value for a read operation.

Writing or reading more than “BCR_value mod 8” (+1 if remainder is non-zero) bytes of data causes a BCR reload to continue the data stream (for example, Write/Read successive bytes in scan ring).

The “data” represents either a TMS sequence, IR command/modifier, or plain TDR data. Start/Stop clocks, wiretest enable, etc. are usually SCOM operations, but TAP commands are required to scan a ring (via a greybox). Access Scanout command supports TDO->TDI data wrap for non-destructive scan ring read.

### A.3.5 Primitive Command Summary

The definition of the first byte of the TAPCMD signal is summarized in *Table A-3*.

*Table A-3. TAP "Primitive Command" Interface*

| Bit | Signal | Description |
|---|---|---|
| 0 | TSRL2 | TMS Select Register<br>0: TMS ≤ (SSR(63), shift right)*(BCRL2+2)<br>1: TMS ≤ '0' when BCRL2 ≠ 111111<br>TTSRL2 when BCRL2 = 111111 |
| 1 | TTSRL2 | TMS Terminal Select Register<br>Value driven on TMS during last bit of BCRL2 count when TSRL2='1' |
| | TRST | TRST ≤ (TSRL2='0') and (TTSRL2='1')<br>this is a way to get TRST function without using more bits |
| 2:7 | BCRL2<br>(6 bits) | Bit Count Register<br>Load with number of bits to shift minus two:<br>b111110 = 64 TCK pulses (SSR shifted 63 times)<br>b111101 = 63 TCK pulses (SSR shifted 62 times)<br>etc.<br>b 111111 = one TCK pulse (SSR will not shift)<br>Note: there is no encoding for zero pulses. |
| 8:71 | SSRL2<br>(64 bits) | Simple Scan Register<br>Data to be shifted out TDI (also out TMS if TSRL2='1')<br>TDO ≥ (0)SSRL2(63) ≥ TDI/TMS<br>TDO is captured on the rising edge of TCK,<br>TDI/TMS are launched (SSR is shifted) on the falling edge of TCK. |

The remaining bits of the TAPCMD signal represent "data" to be loaded into the SSR in preparation of shifting out TDI or TMS.

### A.3.6 CMDDONE - TAPCMD Done, JTAG Scan Logic Idle

When the BCR reaches b111111 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 $I^2C$ slave state machine as an indicator that a new primitive command may be initiated.

### TAPCMD Example 3: 1000-bit Scan-IN Example

| | |
|---|---|
| master start x80084052DF00 stop | master sends slave (JTAG) to Shift-IR |
| master start x80DE40524100080F stop | master loads 0F800041 command into JTAG |
| master start x8002405203 stop | master sends JTAG to Shift-DR |
| master start x80BE4052 | master set-up slave to operate in 64-bit bursts |
| xDATADATADATADATA | 1000 bits / 64 = 15 (8-byte bursts) + 40-bit remainder |
| ... | |
| ... | |
| xDATADATADATADATA stop | this is the last 8-byte burst |
| master start x80E64052 | master set-up slave to expect 40 bits of data |
| x0123456789 stop | final 40 bits of data |
| masterstart x800340521F stop | return slave/JTAG to Test Logic Reset |

The final example assumes that the TAPCMD primitive address is x5240 (this is relocatable).

### *A.3.6.1 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, which scans a new command into the IR.

**Note:** An $I^2C$ write to the SSR data (I/O Buffer) has no effect. An $I^2C$ 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 slave to bring tap state machine first to Test Logic Reset and then to Shift-IR to scan in an instruction

   $I^2C$ Write Start with R/W Bit = 0, Length = 5 Bytes, STOP
   - Byte 1: 08x : 10 TCK Pulses (Data after ADDR/CMD 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 SM to Shift-IR)

2. Load Primitive Command and Instruction Data: Instruct 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 TAP S/M to Exit1-IR

   $I^2C$ 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 scan controller (This will be the IR-STATUS since an instruction register scan was the last operation)

   $I^2C$ Write Start w/R/W Bit = 0, Length = 3 Bytes, STOP
   - Byte 1: 1Ex : Number of bits to read (=32) 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

   $I^2C$ 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

   $I^2C$ 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.3.6.2 Enable Attention Command (TAPCMD Addr(11:8)=0010)*

$I^2C$ method of enabling attention raised from Access action: An $I^2C$ write to the SSR data (I/O buffer) enables attention checking by the $I^2C$ slave. JTAG activity is suppressed.

**Note:** When this sequence is written, the slave will then acknowledge SCOM read/write operations only when attention is non active.

When Attention is active, only primitive TAP commands will be acknowledged with the standard $I^2C$ ACK pulse.

**TAPCMD Example 5: Re-Enabling attention checking internally**

1. $I^2C$ Write Start w/ R/W Bit = 0), Length = 4 Bytes, STOP
2. 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
3. Issue I2C stop sequence

#### *A.3.6.3 Disable Attention Command (TAPCMD Addr(11:8) = 0011)*

$I^2C$ method of disabling attention raised from Access action: An $I^2C$ write to the SSR data (I/O buffer) disables attention checking by $I^2C$ 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 for overriding issues such as 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. By default attention is masked off.

**TAPCMD Example 6: Masking attention internally**

1. $I^2C$ Write Start with R/W Bit = 0, Length = 4 Bytes, STOP
2. Byte 1: --x : TCK pulses
   Byte 2: 43x : LSB of primitive command address
   Byte 3: 52x : MSB of primitive command address
3. Issue $I^2C$ Stop sequence

### A.3.7 POR Unit SCOM Registers

The registers available as interface to the POR unit are listed in *Table A-4* on page 38 along with their access mode and functionality.

*Table A-4. POR Unit SCOM Registers*

| Name | Address | Mode | Functionality |
|---|---|---|---|
| STATUS | 0x400001 | R/W | Status register of POR unit (write access clears reg.) |
| CONT | 0x400100 | Write | Continue/restart command to POR fsm |
| I2CGO | 0x400200 | Write | Assert I2CGO pin |
| PSEQ0 | 0x401400 | Write | aPOR sequence register. 12 first entries (60 MSB of data) |
| PSEQ1 | 0x402401 | Write | aPOR sequence register. 12 next entries (60 MSB of data) |
| PSEQ2 | 0x404401 | Write | aPOR seq. reg. 8 last entries (40 MSB of data) |
| RSV | 0x40xxxx (other) | R/W | Reserved |

# Appendix B. HID Registers and SPRs

## B.1 Move To/From System Register Instructions

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

The encoded SPR values for these implementation specific registers are shown in *Table 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 |
| 304 | | 01001 10000 | HSPRG0 | R/W | none | none | none |
| 305 | | 01001 10001 | HSPRG1 | R/W | none | none | none |
| 311 | | 01001 10111 | HIOR | R/W | none | none | none |
| 314 | | 01001 11010 | HSRR0 | R/W | none | none | none |
| 315 | | 01001 11011 | HSRR1 | R/W | none | none | none |
| **Note:** For **mtspr**, n must be 1. For **mfspr**, reading the SPR is privileged if and only if n=1. | | | | | | | |

**Notes:** (*for Table B-1*)

1. The following sequence must be used when modifying HID0:
   **sync**
   **mfspr** ,HID0,Rx
   **isync**
2. The following sequence must be used when modifying HID1:
   **mtspr** HID1,Rx
   **mtspr** HID1,Rx
   **isync**
   Executing two **mtspr** instructions is necessary to ensure that updates to all portions of HID1 will be complete before the **isync** instruction completes.
3. The following sequence must be used when modifying HID4:
   **sync**
   **mtspr** HID4,Rx
   **isync**
   When HID4[23] is changed, the above sequence should be preceded by a **mtsr** and **sync** which will cause the ERATs to be flushed.
4. The following sequence must be used when modifying HID5:
   **sync**
   **mtspr** HID5,Rx
   **isync**
   Whenever HID5[56] or HID5[57] is changed, the entire instruction cache must be flushed to insure that any succeeding **dcbz** is executed in the context of the new HID5 bit settings.

5. Although it is not necessary to use synchronizing instructions when modifying the MMCR(0,1,A) registers, it is recommended that the following sequence be used:
   **sync**
   **mtspr** MMCRz,Rx
   **isync**

*Table B-2* describes the 970FX'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**<br>**mfspr** | Causes privileged instruction type program interrupt |
| 0 | Any invalid SPR encoding except:<br>SPR(0:9) = 00000 00000<br>SPR(0:9) = 00100 00000<br>SPR(0:9) = 00101 00000<br>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<br>SPR(0:9) = 00100 00000<br>SPR(0:9) = 00101 00000<br>SPR(0:9) = 00110 00000 | X | **mtspr**<br>**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:

*Figure B-1. Processor Identification Register*

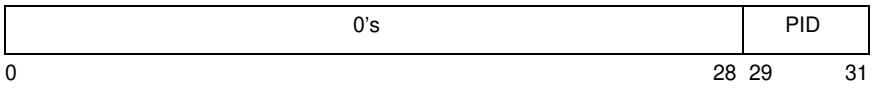



*Table B-3. PIR Register*

| Bits | Name | Description |
|---|---|---|
| 0:28 | – | Reserved (read as zeros) |
| 29:31 | PID | Three 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.1.2 HID Registers (HID0, HID1, HID4, and HID5)

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

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

*Table B-4. HID0 Bit Functions* (Page 1 of 2)

| Bits | Bit 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<br>Flush and refetch after the completion of each group or the completion of each microcoded instruction, if the instruction spans multiple groups. |
| 2 | isync_sc | Disable "isync scoreboard" optimization |
| 3 | ser-gp | Serialize Group Dispatch (next group not dispatched until previous group completes) |
| 4:8 | – | Reserved |
| 9 | nap | Nap |
| 10 | – | Reserved |
| 11 | dpm | Enable dynamic power management |
| 12 | – | Reserved |

*Table B-4. HID0 Bit Functions* (Page 2 of 2)

| Bits | Bit Name | Description |
|---|---|---|
| 13 | tg | Performance monitor threshold granularity control |
| 14 | hang_dis | Disable processor hang detection mechanism |
| 15 | nhr | “Not Hard Reset” (check after SRI to see if hard or soft) |
| 16 | inorder | “Serialized group issue” mode. The next group is not issued until the previous group completes. Does not include branch or cr-logical instructions. |
| 17 | – | Reserved |
| 18 | tb_ctrl | Enable time base counting when processor is “stopped” |
| 19 | ext_tb_en | External time base enable<br>0: Use TBEN input as enable (TB clocked at 1/8 full processor frequency)<br>1: Use TBEN input to clock time base (external clock) |
| 20:21 | – | Reserved |
| 22 | ciabr_en | Enable CIABR (Completion Instruction Address Breakpoint Register) |
| 23 | hdice_en | Enable hypervisor decrementer interrupt conditionally (HDICE). The initial reset value must be ‘0’ and disables hypervisor interrupts. |
| 24 | en_therm | Enable external thermal interrupts |
| 25:30 | – | Reserved |
| 31 | en_attn | Enable support processor attention instruction |
| 32 | en_mck | Enable external machine check interrupts (preferred state = 0b1) |
| 33:60 | – | Reserved |
| 61 | lg_pg_dis | Large page disable |
| 62:63 | – | Reserved |

*Table B-5. HID1 Bit Functions* (Page 1 of 2)

| Bits | Bit Name | Description |
|---|---|---|
| 0:2 | bht_pm | Branch history table (BHT) prediction mode<br>000 Static prediction<br>010 Global BHT prediction only<br>100 Local BHT prediction only<br>110 Full global/local prediction with global selection (gsel)<br>001 – unused – (does same as 000)<br>011 Global prediction with history compression<br>101 – unused – (does same as 100)<br>111 Full global/local prediction with gsel and history compression (preferred state) |
| 3 | en_ls | Enable link stack (preferred state = ‘1’) |
| 4 | en_cc | Enable count cache (preferred state = ‘1’) |
| 5 | en_ic | Enable instruction cache (preferred state = ‘1’) |
| 6 | – | Reserved |
| *7:8* | pf_mode | Prefetch mode:<br>00 No instruction prefetch<br>01 Select next sequential address (NSA) instruction prefetch<br>10 Select NSA and NSA+1 instruction prefetch (preferred state)<br>11 Disable prefetch buffer |
| 9 | en_icbi | Enable “forced icbi match” mode |

*Table B-5. HID1 Bit Functions* (Page 2 of 2)

| Bits | Bit Name | Description |
|---|---|---|
| 10 | en_if_cach | Enable instruction fetch cacheability control<br>0 All instruction fetch accesses are treated as cache inhibited (regardless of the state of the page table I-bit).<br>1 Instruction fetch cacheability is controlled by the state of the page table I-bit. (preferred state) |
| 11 | en_ic_rec | Enable I-cache parity error recovery (preferred state = ‘1’) |
| 12 | en_id_rec | Enable I-directory parity error recovery (preferred state = ‘1’) |
| 13 | en_er_rec | Enable I-ERAT parity error recovery (preferred state = ‘1’) |
| 14 | ic_pe | Force instruction cache parity error (error inject) |
| 15 | icd0_pe | Force instruction cache directory 0 parity error (error inject) |
| 16 | — | Reserved |
| 17 | ier_pe | Force I-ERAT parity error (error inject) |
| 18 | en_sp_itw | Enable speculative tablewalks. The ERAT is never loaded using a PTE if PTE[G] =’1’. (preferred state = ‘1’) |
| 19:63 | — | Reserved |

*Table B-6. HID4 Bit Functions* (Page 1 of 2)

| Bits | Bit Name | Description |
|---|---|---|
| 0 | lpes1 | LPAR environment selector bit [0]. LPES[0:1] are located in HID4[57, 0].<br>LPES[0:1] determine how MSR[HV] is set using interrupts and how memory access is performed when not in hypervisor mode. This is described in the PowerPC Architecture version 2.01. |
| 1:2 | rmlr(1:2) | LPAR real mode limit register (see HID4[58] for bit [0]). |
| 3:6 | lpid(2:5) | LPAR partition identity bits [2:5] (see also bits [62:63] for lpid(0:1)). |
| 7:22 | rmor(0:15) | LPAR real mode offset register [0:15]. |
| 23 | rm_ci | The real mode caching inhibited bit can be used to permit a control register on an I/O device to be accessed without permitting the corresponding storage location to be copied into the caches. The bit should normally contain ‘0’. Software would set the bit to ‘1’ just before accessing the control register, access the control register as needed, and then set the bit back to ‘0’. |
| 24 | force_ai | Force alignment interrupt instead of microcode on unaligned operations. |
| 25 | dis_pref | Disables data prefetching. |
| 26 | res_pref | Setting HID4[26] = ‘1’ resets the data prefetch mechanism, suppressing subsequent prefetch requests and clearing the stream detection logic so that stream detection will not be affected by accesses performed prior to setting the bit back to ‘0’. |
| 27 | en_sp_dtw | Enable speculative load tablewalk. |
| 28 | l1dc_flsh | L1 data cache flash invalidate.<br>0 Normal operation.<br>1 All sectors set to invalid and held invalid. |
| 29:30 | dis_derpc | Disable D-ERAT parity checking [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]. |
| 31 | dis_derpg | Disable D-ERAT parity generation (force parity to ‘0’ on EA[0:45] only). |
| 32:33 | dis_derat | Disable D-ERAT [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]; valid states 00,01,10. |

*Table B-6. HID4 Bit Functions* (Page 2 of 2)

| Bits | Bit Name | Description |
|---|---|---|
| 34:35 | dis_dctpc | Disable data cache tag parity checking [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]. |
| 36 | dis_dctpg | Disable data cache tag parity generation. |
| 37:38 | dis_dcset | Disable data cache set [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]. |
| 39:40 | dis_dcpc | Disable data cache parity checking [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]. |
| 41 | dis_dcpg | Disable data cache parity generation. |
| 42:43 | dis_dcrtpc | Disable data cache real address tag parity checking. |
| 44:47 | dis_tlbpc | Disable TLB parity checking [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]. |
| 48 | dis_tlbpg | Disable TLB parity generation. |
| 49:52 | dis_tlbset | Disable TLB set [disable parity checking in one or more ways of the 4-way set associative TLB (one bit per way)]; valid states X‘0’,X‘7’,X‘B’,X‘D’,X‘E’. |
| 53 | dis_slbpc | Disable SLB parity checking. |
| 54 | dis_slbpg | Disable SLB parity generation. |
| 55 | mck_inj | Machine check error inject enable |
| 56 | dis_stfwd | Disable store forwarding (cause reject). |
| 57 | lpes0 | LPAR environment selector bit. LPES[0:1] are located in HID4[57, 0].<br>LPES[0:1] determine how MSR[HV] is set using interrupts and how memory access is performed when not in hypervisor mode. This is described in the PowerPC Architecture version 2.01. |
| 58 | rmlr0 | HID4 bits [58, 1:2] are real mode limit register bits [0:2].<br>011 64 MB<br>111 128 MB<br>100 256 MB<br>x10 1 GB<br>x01 16 GB<br>000 256 GB |
| 59 | – | Reserved. |
| 60 | dis_splarx | Disable speculative **lwarx**, **ldarx**. |
| 61 | lg_pg_dis | Disable large page support; Large page (L) bit input to the SLB will be forced to zero (software will read a zero L-bit) |
| 62:63 | lpid(0:1) | LPAR partition identity bits [0:1]. HID4[62:63, 3:6] are LPID[0:5] respectively. |

*Table B-7. HID5 Bit Functions (Page 1 of 2)*

| Bits | Bit Name | Description |
|---|---|---|
| 0:31 | – | Reserved |
| 32:47 | hrmor(0:15) | LPAR hypervisor real mode offset register |
| 48:49 | – | Reserved |
| 50 | DC_mck | Machine check enabled for data cache and data cache tag parity errors (software recovery enabled). |

*Table B-7. HID5 Bit Functions (Page 2 of 2)*

| Bits | Bit Name | Description |
|---|---|---|
| 51 | dis_pwrsave | L1 data cache, L1 D-cache Tag, D-ERAT power savings disable |
| 52 | force_G | Force guarded (G='1') load |
| 53 | DC_repl | Data cache replacement algorithm:<br>0 default - Least Recently Used (LRU)<br>1 FIFO |
| 54 | hwr_stms | Number of available hardware prefetch streams:<br>0 4 hardware streams and 4 vector streams<br>1 8 hardware streams (HID5[55] must also be 0b1) |
| 55 | dst_noop | Data StreamTouch (DST) instructions no-op<br>0 DST's are enabled<br>1 DST's are a no-op and discarded in the Load Store Unit (LSU) |
| 56 | DCBZ_size | Makes **dcbz** a 32-byte store when **dcbz** instruction bit [10] = '0' |
| 57 | DCBZ32_ill | Makes a **dcbz** instruction with bit [10] = '0' an illegal instruction |
| 58 | tlb_map | TLB Mapping:<br>0 4 way set associative<br>1 direct mapped<br>**Note:** When setting HID5[58] to setup the TLB to be direct mapped, HID4[49:52] (TLB set disable bits) must be cleared, otherwise translation will not work. |
| 59 | lmq_port | Demand miss (LMQ to STS)<br>0 permit two per cycle<br>1 permit only one per cycle |
| 60 | lmq_size(0) | Number of outstanding requests to STS (970FX storage subsystem)<br>HID5[60, 63] - maximum outstanding requests<br>00 8<br>01 1<br>10 2<br>11 4 |
| 61 | – | Reserved |
| 62 | tch_nop | Make **dcbt** and **dcbtst** act like no-ops |
| 63 | lmq_size(1) | See description of HID5[60] |

### B.1.3 SCOM Registers (SCOMC and SCOMD)

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

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

### B.1.4 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.1.5 IMC Array Access Register

The Instruction Match Cam (IMC) array facility is used for performance monitoring facility. 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.1.6 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 970FX.

# Appendix C. Using I2C and JTAG

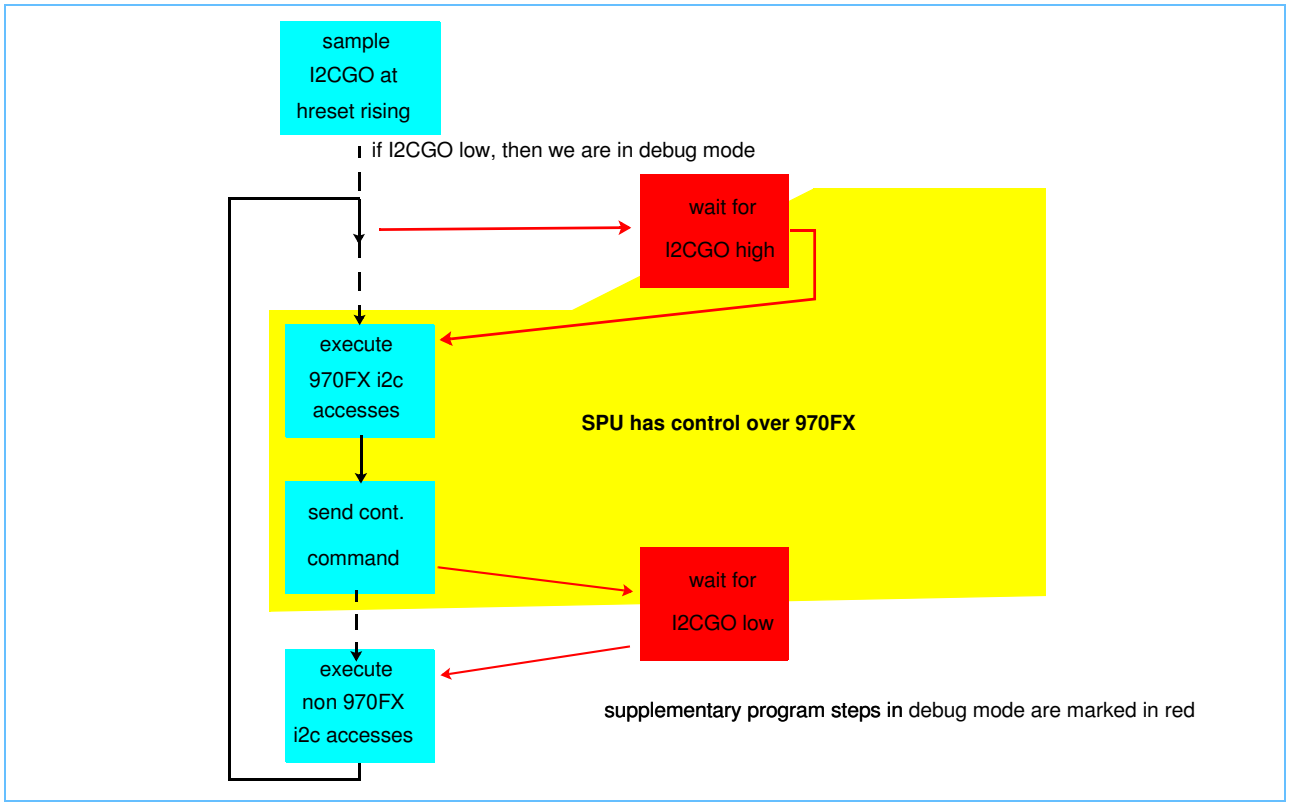
## C.1 I2C/JTAG Hand Shaking

At HRESET low the 970FX samples its GPULDBG pin and if asserted immediately drive its I2CGO line low, giving initial control to the JTAG debugger. The SPU samples I2CGO when raising HRESET. If low it knows that it is in debug mode and that it must check for I2CGO to be high before accessing the 970FXs. In non-debug mode the line will be held in high-Z constantly by the corresponding 970FX.

After the JTAG debugger has finished accessing the 970FX it will execute a JTAG command to return the I2CGO pin of the 970FX to high-Z. The SPU will wait for this pin to become high (external pull-up resistor) to proceed through the normal POR procedure until it issues a continue command to the 970FX. This command will in debug mode result in the 970FX driving its I2CGO signal low, indicating that the SPU has lost control of the 970FX and that the JTAG debugger may now access the processor again. The SPU should then wait for the I2CGO signal to be driven low by the 970FX in order to prevent the SPU from reaccessing the 970FX before it had time to drive the I2CGO signal low. While the I2CGO signal is low the SPU may continue to access all I2C devices on the bus except the 970FX(s) that have invalidated their I2CGO line. It has to wait for the I2CGO pin to be asserted again before reaccessing the 970FX. Similarly the JTAG debugger monitors the I2CGO and may only access the 970FX when it is low. This signal is open drain and can be dot-ored across the 970FXs to minimize the number of debug pins on the SPU.

The corresponding changes in the SPU program flow for debugging are shown below in red. Before the first access to the 970FX or before access to the 970FX following a continue command, the SPU must wait for the I2CGO signal to go high.

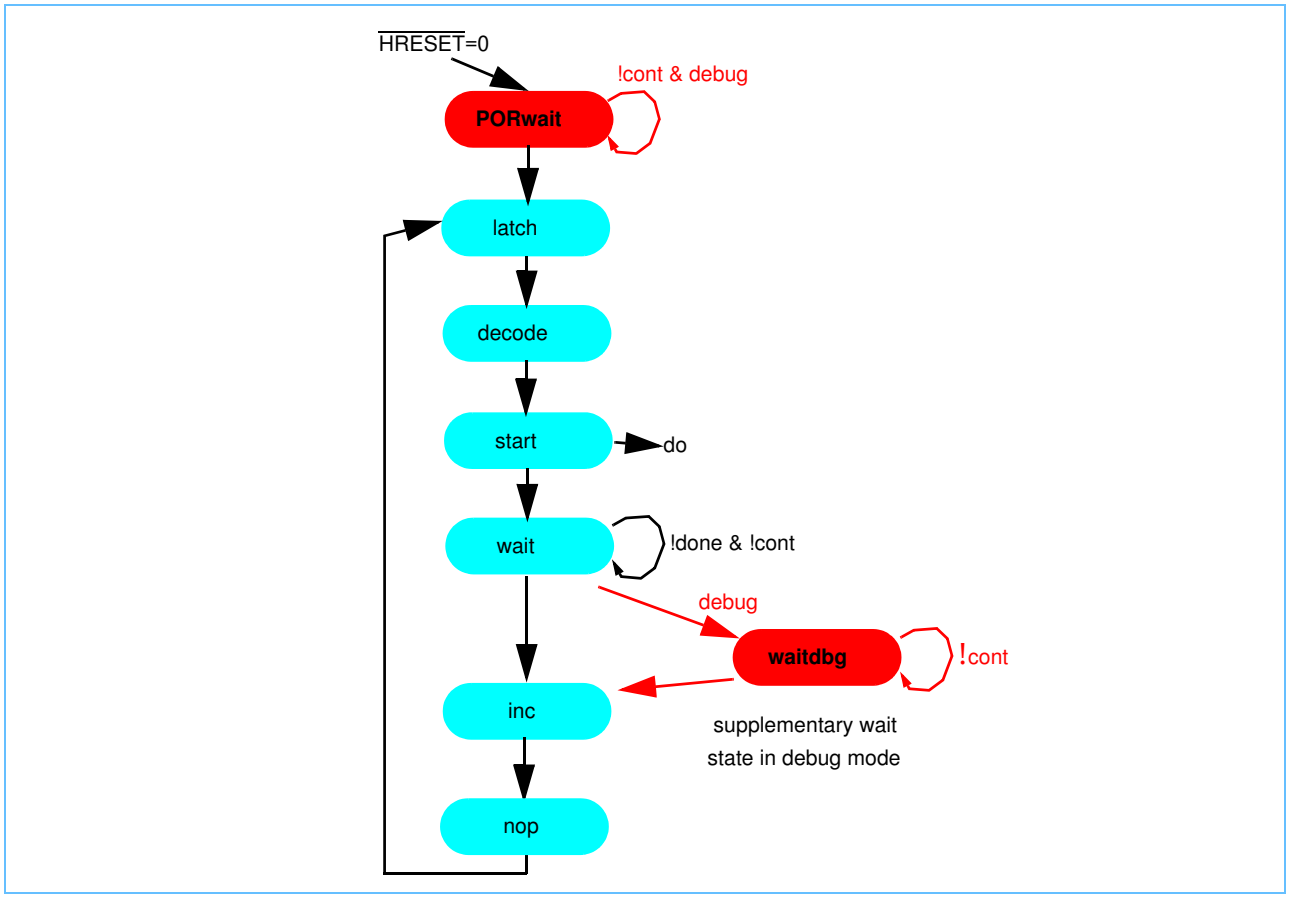
*Figure C-1. PCU Program Flow*



# Appendix D. Operational Mode Description

After $\overline{\text{HRESET}}$ is raised the program counter PC is cleared and the POR state machine first enters the POR WAIT state (see *Figure D-1* on page 49). This state will not be left in debug mode to allow reprogramming of the POR sequence. In non-debug mode it will be left in the next cycle. The state machine then starts fetching the instruction from the POR sequence register pointed to by PC (*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 $I^2C$. The POR program counter is then incremented and the state machine loops to the *latch* state.

*Figure D-1. State Diagram*




Upon entering the WAIT state a watch-dog counter is started for all but the WAIT instruction. If this counter reaches the value $2^{22}$ (approximately *5ms* at 2GHz clock frequency) then a timeout bit will be set in the status register. In debug mode a supplementary WAIT state (*waitdbg*) is entered after the nominal WAIT state to allow intervention during the POR procedure. A continue command from JTAG or $I^2C$ is required to leave this state.

After completion of the instruction in the last entry of the POR sequence register the *last instruction* bit in the status register will be set and the state machine will move to *waitdbg* instead of the *inc* state to allow reloading of the POR sequence register through JTAG or $I^2C$ if a longer sequence is required.

## D.1 Supported POR Instructions

*Table D-1* lists the instructions supported by the POR state machine along with their coding and functionality. Coding values not listed in *Table D-1* default to the WAIT instruction and set the illegal instruction bit in the 970FX POR status register, SCOM 0x400000.

*Table D-1. POR Unit Instructions*

| Coding | Name | Functionality |
|---|---|---|
| 00000 | NOP | No operation |
| 00001 | WAIT | Wait for SE intervention |
| 00010 | RSTFRL | Reset Free Running Logic |
| 00011 | SAMPLEFUSE | Copy the fuses content to latches |
| 00100 | SCAN0 | Flush the scan rings with 0 |
| 00101 | DABISTINITL1 | Run ABIST to init L1 arrays |
| 00110 | SCABISTINIT | Run ABIST to init scanned array (all) |
| 00111 | DABISTINITL2 | Run ABIST to init L2 arrays |
| 01000 | DRIVEIOS | Start driving external interfaces outputs |
| 01001 | STARTZIOCLK | Start clocks in the ZIO unit |
| 01111 | STARTGUSCLK | Start clocks in the STS unit |
| 10011 | STARTCORECLK | Start clocks in the CORE+VPU |
| 01010 | STOPCLKS | Stop all clocks in ZIO+STS+CORE+VPU |
| 01011 | SYNCPHASE | Sync to the external phase signal |
| 01100 | TOGGLEWIAP | Start/Stop driving the IAP pattern |
| 01101 | SYNCRIAP | Sync to the IAP pattern |
| 01110 | INITGUS | Perform bus unit (STS) initialization |
| 10000 | INITCORE | Perform core initialization |
| 10001 | SRESET | Start the core |

### D.1.1 Instruction NOP

This instruction does nothing. The POR state machine will immediately advance to the next instruction.

### D.1.2 Instruction WAIT

This instruction does nothing. The WAIT state will be indicated in the status register. The POR state machine will stay in this state until a continue instruction is sent by JTAG or $I^2C$. Limitation: this instruction may not be used in the last entry of the POR sequence register.

### D.1.3 Instruction RSTFRL

This instruction resets the free running logic. As a result all clock domain (except the free running clock domain) are stopped; the c2-clock domain is started and the free running logic state initialized. After completion, the next instruction is issued.

### D.1.4 Instruction SAMPLEFUSE

This instruction copies the fuse information (384 fuses) to the corresponding latches and continues to the next instruction.

### D.1.5 Instruction SCAN0

This instruction scans zeros along all the scan chains of the chip to initialize the latches (except chipras, fuse and notbist chains). Inverters along the chain take care of setting latches to one if required. A total of 2049 scan clocks (sc1/c2) are generated. After completion, the next instruction is executed. Notice that due to the initialization of the SCOM clock control command register at power on, the first SCAN0 command issued by the POR engine is equivalent to a FULLSCAN0. For example, it also cleans up the fuse and notbist chains.

### D.1.6 Instruction DABISTINITL1, DABISTINITL2, SCABISTINIT

These instructions run an automatic build-in self initialization of the chip arrays using the ABIST engines. SCAN0 must be run prior to these instructions. The arrays are left in a clear (all zero) state upon completion. After completion the next instruction is fetched. DABISTINITL1 initialize the L1I and L1D core arrays. The second is used for the L2 arrays. Due to L2 array implementation, it must be run twice with a SCAN0 in between to clean the whole L2 cache. SCABISTINIT initialize all of the arrays using scanned ABIST engines.

*Table D-2. ABISTINIT Command Description*

| DABISTINITL1 | DABISTINITL2 | SCABISTINIT |
|---|---|---|
| Set core_dabist_init<br>Set core_dabist_en<br>Unset core_wr_ary_inh | Set gus_dabist_init,<br>Set gus_dabist_en<br>Unset gusdedicated_wr_ary_inh | Set core/notcore_scabist_init,<br>Set core/notcore_scabist_en<br>Unset core_wr_ary_inh, sc_wr_ary_inh |
| Wait 64 cycles | Wait 64 cycles | Wait 64 cycles |
| Apply 4x4097 c1/ramc2/c2star/c2 to core | Apply 4x4097 c1/ramc2/c2star/c2 to STS | Repeat 2049 times:<br>1. Apply 128 sc1/c2 (ratio 6:1)<br>2. Apply 1 ramc2/c2<br>3. Apply 1 c2<br>4. Apply 1 c1/ramc2/c2start/c2 |
| Wait 64 cycles | Wait 64 cycles | Wait 64 cycles |
| Set core_wr_ary_inh<br>Unset core_dabist_init, core_dabist_en | Set STS_wr_ary_inh<br>Unset gus_dabist_init, gus_dabist_en | Set core_wr_ary_inh, sc_wr_ary_inh<br>Unset core/notcore_scabist_init,<br>Unset core/notcore_scabist_en |

### D.1.7 Instruction DRIVEIOS

After this instruction was issued the external interfaces outputs will be driven by the chip. The POR state machine moves there after the next instruction.

### D.1.8 Instruction STARTZIOCLK, STARTGUSCLK, STARTCORECLK

These instructions start the functional clock in the I/O, STS, and CORE clock domains on the chip. Upon completion the next instruction is fetched. Clocks are always started on a psync boundary to prevent timing problems in n:1 clock domains.

### D.1.9 Instruction STOPCLKS

This instruction stops all the clock domains of the chip on a psync boundary. After completion the POR state machine starts the next instruction.

### D.1.10 Instruction SYNCPHASE

This instruction is used to synchronize to the phase signal asserted by the North Bridge every 48 processor clocks for 1 system clock.

### D.1.11 Instruction TOGGLEWIAP

TOGGLEWIAP instructs the processor interface (PI) sender (writer) to start/stop driving the writer initial alignment pattern (WIAP). This is achieved through an auxiliary state machine consisting of a single set/reset flip-flop. At power-on this flip-flop is reset, so that no IAP pattern is driven out. The state machine moves within 1 cycle to the next instruction.

### D.1.12 Instruction SYNCRIAP

This instruction is used to start the synchronization of the PI receiver (reader). It should only be issued after the WIAP is driven by the sender. An auxiliary state machine will take care of raising the appropriate signal to the PI. The state machine will then wait for the PI to signal completion before moving to the next instruction. Status of the PI receiver can be checked through SCOM.

### D.1.13 Instruction INITGUS

This instruction will unfence the arrays and initialize the BUS unit inside STS. This instruction should only be issued after the PI has been initialized and valid data are received/sent. This instruction should be issued a first time prior to turning the STS clock domain on to unfence the array and reissued after turning the STS clock domain on to reset the storage system. After completion the state machine moves to the next instruction.

### D.1.14 Instruction INITCORE

This instruction finishes the initialization of the core. It will first initialize the CAM arrays, bring the core to quiesce and turn on attentions and checkers. After completion the state machine fetches the next instruction. This instruction also drop the fence around chipras (*tcl_tx_gateios*). It should only be issued after core, STS, and I/O have been initialized, the I/O receiver inhibits have been unasserted.

### D.1.15 Instruction SRESET

This instruction takes the processor out of quiesce and initializes the fetching (standard fetch address is 0x100 is not set overwise). This instruction should be the last instruction in a normal POR sequence since the processor is running on its own afterwards. Therefore after completion of this instruction the POR state machine stops and waits indefinitely (the timeout bit in the status register will be set). A continue command can be sent to force the POR state machine to advance to the next instruction in the POR sequence register.

### D.1.16 SCOM Access

*Table D-3* lists the POR registers accessible via SCOM.

*Table D-3. POR Unit SCOM Registers*

| Name | Address | Mode | Functionality |
|---|---|---|---|
| STATUS | 0x400001 | R/W | Status register of POR unit (write access clears reg.) |
| CONT | 0x400100 | Write | Continue/restart command to POR fsm |
| I2CGO | 0x400200 | Write | Assert i2CGO pin |
| PSEQ0 | 0x401400 | Write | POR sequence register. 12 first entries (60 MSB of data) |
| PSEQ1 | 0x402401 | Write | POR sequence register. 12 next entries (60 MSB of data) |
| PSEQ2 | 0x404401 | Write | POR seq. reg. 8 last entries (40 MSB of data) |
| RSV | 0x40xxxx (other) | R/W | Reserved |

### D.1.17 Status Register

The status register reflects the current state of the POR procedure of the chip.

# Revision Log

| Revision Date | Contents of Modification |
|---|---|
| July 19, 2005 | Version 0.6C<br>• Initial version |