Trademarks

AMD, the AMD Arrow logo, Agesa, and combinations thereof, are trademarks of Advanced Micro Devices, Inc. Microsoft and Windows are registered trademarks of Microsoft Corporation.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Disclaimer

The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice. AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to this document including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right. AMD shall not be liable for any damage, loss, expense, or claim of loss of any kind or character (including without limitation direct, indirect, consequential, exemplary, punitive, special, incidental or reliance damages) arising from use of or reliance on this document. No license, whether express, implied, arising by estoppel, or otherwise, to any intellectual property rights are granted by this publication. Except for AMD product purchased pursuant to AMD's Standard Terms and Conditions of Sale, and then only as expressly set forth therein, AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.
### Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Rev.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>April, 2012</td>
<td>3.00</td>
<td>• Becomes a public release.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated section 3.2.1.1 Special Locked Area in CMOS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Fixed typo in section 3.6 System Restart after Power Fail, where register 74h is corrected to 5Bh.</td>
</tr>
<tr>
<td>May, 2011</td>
<td>2.02</td>
<td>• Changed cover title – Replaced code name with marketing names.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added marketing names to the Hudson-1 family members in the Introduction.</td>
</tr>
<tr>
<td>February, 2011</td>
<td>2.01</td>
<td>• Added new Section 9 CIR Support.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added new section 10 SMI Programming.</td>
</tr>
<tr>
<td>November, 2010</td>
<td>2.00</td>
<td>• Added Hudson-E1 to the document.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated section 8 A-Link Bridge with the proper addresses for abRegBAR, AB_INDEX, and AB_DATA.</td>
</tr>
<tr>
<td>June, 2010</td>
<td>1.01</td>
<td>• Updated Figure 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Corrected typos 22h to 16h in Section 2.1 PCI Devices and Functions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added section 2.3.1 MMIO Programming for Legacy Devices.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated section 3.6 System Restart after Power Fail.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added new section 3.6.1 Power Fail and Alarm Setup.</td>
</tr>
<tr>
<td>Jan, 2010</td>
<td>1.00</td>
<td>First release.</td>
</tr>
</tbody>
</table>
# Table of Contents

1 **Introduction** ................................................................................................................................. 6
   1.1 About this Manual ........................................................................................................................... 6
   1.2 Block Diagram ............................................................................................................................... 7
   1.3 Internal PCI Devices ..................................................................................................................... 8

2 **Hudson-1 Programming Architecture** .......................................................................................... 9
   2.1 PCI Devices and Functions ............................................................................................................. 9
   2.2 I/O Map ......................................................................................................................................... 10
      2.2.1 Fixed I/O Address Ranges .................................................................................................... 10
      2.2.1.1 Fixed I/O Address Ranges – Hudson-1 Proprietary Ports ................................................... 10
      2.2.2 Variable I/O Decode Ranges ................................................................................................ 10
   2.3 Memory Map .................................................................................................................................. 11
      2.3.1 MMIO Programming for Legacy Devices .............................................................................. 11

3 **Hudson-1 Early-POST Initialization** ............................................................................................ 16
   3.1 512K/1M ROM Enable .................................................................................................................... 16
      3.1.1 PCI ROM ............................................................................................................................... 16
      3.1.2 LPC ROM ............................................................................................................................. 16
      3.1.3 LPC ROM Read/Write Protect ............................................................................................... 16
      3.1.4 SPI ROM Controller .............................................................................................................. 16
   3.2 Real Time Clock (RTC) ................................................................................................................... 18
      3.2.1 RTC Access ............................................................................................................................ 18
      3.2.1.1 Special Locked Area in CMOS .......................................................................................... 18
      3.2.1.2 Century Byte .................................................................................................................... 18
      3.2.1.3 Date Alarm ........................................................................................................................ 18
   3.3 BIOS RAM .................................................................................................................................... 19
   3.4 Serial IRQ ...................................................................................................................................... 19
   3.5 SubSystemID and SubSystem Vendor ID ....................................................................................... 20
   3.6 System Restart after Power Fail ..................................................................................................... 20
      3.6.1 Power Fail and Alarm Setup .................................................................................................. 21

4 **PCI IRQ Routing** ............................................................................................................................ 22
   4.1 PCI IRQ Routing Registers ............................................................................................................ 22
   4.2 PCI IRQ BIOS Programming ......................................................................................................... 23
   4.3 Integrated PCI Devices IRQ Routing ............................................................................................. 24
   4.4 PCI IRQ Routing for APIC Mode .................................................................................................. 24

5 **SMBus Programming** .................................................................................................................... 25
   5.1 SMBus Timing ............................................................................................................................... 25
   5.2 SMBus Host Controller Programming ......................................................................................... 25

6 **Serial ATA (SATA)** .......................................................................................................................... 28
   6.1 Device ID ....................................................................................................................................... 30
   6.2 SATA Controller Operating Modes .............................................................................................. 31

7 **APIC Programming** .......................................................................................................................... 32
# Table of Contents

7.1 Northbridge APIC Enable............................................................................................................. 32
7.2 FCH APIC Enable .......................................................................................................................... 32
7.3 IOAPIC Base Address..................................................................................................................... 32
7.4 APIC IRQ Assignment................................................................................................................... 32
7.5 APIC IRQ Routing ........................................................................................................................ 32

8 UMI Bridge ................................................................................................................................... 34
8.1 Programming Procedure ............................................................................................................... 35
8.2 UMI Configuration DMA Access ................................................................................................ 36
8.3 Enabling Non-Posted Memory Write .......................................................................................... 37

9 CIR Support .................................................................................................................................. 38
9.1 Southbridge CIR ........................................................................................................................... 38
  9.1.1 Host and CIR Communication ...................................................................................................... 38
  9.1.2 Logical Device Number 5 Registers ............................................................................................ 38
  9.1.3 Sample Code to Read and Write CIR Registers ........................................................................ 41
  9.1.4 Sample Code to Enable Tx0 and Program it as Push Pull Driver ............................................... 41
  9.1.5 Sample ASL Code for CIR ............................................................................................................. 42

10 SMI Programming ......................................................................................................................... 45
10.1 Workaround for SMI Command Port Status Byte ..................................................................... 45

11 Legacy BIOS Implementation for Chipset Integration Module Extensive (Hudson-1 CIMx) .......................................................................................................................... 47
11.1 Introduction ............................................................................................................................... 47
11.2 CIMx Interface Calls Environment ............................................................................................ 47
11.3 Interface Definition .................................................................................................................... 47
  11.3.1 FCH Power-On/Reset Initialization ......................................................................................... 47
  11.3.2 FCH BIOS POST Initialization ............................................................................................... 48
  11.3.3 S3 Resume Initialization ........................................................................................................... 48
  11.3.4 Callback Interface Definition .................................................................................................. 48

12 UEFI BIOS Implementation for Chipset Integration Module Extensive (Hudson-1 CIMx) ............................................................................................................................... 49
12.1 Introduction ............................................................................................................................... 49
12.2 CIMx Interface Calls Environment ............................................................................................ 49
  12.2.1 FCH PEI Module .................................................................................................................... 49
  12.2.2 FCH DXE Driver ..................................................................................................................... 50
1 Introduction

1.1 About this Manual

This manual provides guidelines for BIOS developers working with the AMD family of Fusion Controller Hub (FCH) codenamed Hudson-1. It describes the BIOS and software modifications required to fully support the Hudson-1 FCHs.

Note: The term Hudson-1 is used in this document to refer to the following Hudson-1 family members:

<table>
<thead>
<tr>
<th>Marketing Name</th>
<th>Codename</th>
</tr>
</thead>
<tbody>
<tr>
<td>A45</td>
<td>Hudson-D1</td>
</tr>
<tr>
<td>A50M</td>
<td>Hudson-M1</td>
</tr>
<tr>
<td>A55E</td>
<td>Hudson-E1</td>
</tr>
</tbody>
</table>

The information in this document applies to all members of the Hudson-1 family unless otherwise indicated. Other documents on the Hudson-1 are available at the AMD NDA site.

To help the reader to readily identify changes/updates in this document, changes/updates over the previous revision are highlighted in red.
1.2 Block Diagram

This section contains a diagram for the Hudson-1. Figure 1 below shows the Hudson-1 internal PCI devices and the major function blocks.

---

**Figure 1 Hudson-1 PCI Internal Devices and Major Function Blocks**

---

Notes:

* Flash controller function is not supported on Hudson-1 platforms.
** PCI controller function is not supported on Hudson-M1 platforms.
*** GbE MAC is not supported.
1.3 Internal PCI Devices

Note: The Hudson-1 internal PCI devices are listed in Figure 2 below. The sub-sections that follow provide descriptions of the PCI configuration space, the I/O space, and the memory space registers for each device. PCI configuration space registers are only accessible with configuration Read or configuration Write cycles and with the target device selected by setting its corresponding IDSEL bit in the configuration cycle address field.

Figure 2: Hudson-1 PCI Internal Devices
## 2 Hudson-1 Programming Architecture

### 2.1 PCI Devices and Functions

<table>
<thead>
<tr>
<th>Bus:Device:Function</th>
<th>Function Description</th>
<th>Dev ID</th>
<th>Enable/Disable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus 0:Device 14h:Function 0</td>
<td>SMBus Controller</td>
<td>4385h</td>
<td>Always enabled</td>
</tr>
<tr>
<td>Bus 0:Device 14h:Function 1</td>
<td>SATA (IDE) Controller</td>
<td>438Ch</td>
<td>PM_Reg: 0xDA bit3</td>
</tr>
<tr>
<td>Bus 0:Device 14h:Function 2</td>
<td>Azalia Controller</td>
<td>4383h</td>
<td>PM_Reg: 0xEB bit0</td>
</tr>
<tr>
<td>Bus 0:Device 14h:Function 3</td>
<td>LPC Controller</td>
<td>438Dh</td>
<td>PM_Reg: 0xEC bit0</td>
</tr>
<tr>
<td>Bus 0:Device 14h:Function 4</td>
<td>PCIB Bridge</td>
<td>4384h</td>
<td>Always enabled</td>
</tr>
<tr>
<td>Bus 0:Device 14h:Function 6</td>
<td>GEC Controller</td>
<td>1699h</td>
<td>PM_Reg: 0xF6 bit0 cleared</td>
</tr>
<tr>
<td>Bus 0:Device 12h:Function 2</td>
<td>USB #1EHCI Controller</td>
<td>4396h</td>
<td>PM_Reg: 0xEF bit1, bit3, bit5</td>
</tr>
<tr>
<td>Bus 0:Device 13h:Function 2</td>
<td>USB #2EHCI Controller</td>
<td>4396h</td>
<td></td>
</tr>
<tr>
<td>Bus 0:Device 16h:Function 2</td>
<td>USB #3EHCI Controller</td>
<td>4396h</td>
<td></td>
</tr>
<tr>
<td>Bus 0:Device 12h:Function 0</td>
<td>USB #1 OHCI Controller</td>
<td>4397h</td>
<td>PM_Reg: 0xEF bit 0, bit 2, bit 4, bit 6</td>
</tr>
<tr>
<td>Bus 0:Device 13h:Function 0</td>
<td>USB #2 OHCI Controller</td>
<td>4397h</td>
<td></td>
</tr>
<tr>
<td>Bus 0:Device 14h:Function 5</td>
<td>USB #4 OHCI Controller</td>
<td>438Bh</td>
<td></td>
</tr>
<tr>
<td>Bus 0:Device 16h:Function 0</td>
<td>USB #3 OHCI Controller</td>
<td>4397h</td>
<td></td>
</tr>
<tr>
<td>Bus 0:Device 11h:Function 0</td>
<td>Native/Legacy IDE Mode</td>
<td>4390h</td>
<td>PM_Reg: 0xDA bit0</td>
</tr>
<tr>
<td></td>
<td>AHCI Mode</td>
<td>4391h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Non-Raid-5 Mode</td>
<td>4392h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Raid5 Mode</td>
<td>4393h</td>
<td></td>
</tr>
</tbody>
</table>
2.2 I/O Map

The I/O map is divided into Fixed and Variable address ranges. Fixed ranges cannot be moved, but can be disabled in some cases. Variable ranges are configurable.

2.2.1 Fixed I/O Address Ranges

2.2.1.1 Fixed I/O Address Ranges – Hudson-1 Proprietary Ports

<table>
<thead>
<tr>
<th>I/O Address</th>
<th>Description</th>
<th>Enable Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>C00h-C01h</td>
<td>IRQ Routing Index/Data register</td>
<td>PM_Reg: 0x00 [1]</td>
</tr>
<tr>
<td>C14h</td>
<td>PCI Error Control register</td>
<td>PM_Reg: 0x00 [20]</td>
</tr>
<tr>
<td>C50h-C51h</td>
<td>Client Management Index /Data registers</td>
<td>PM_Reg: 0x00 [27]</td>
</tr>
<tr>
<td>C52h</td>
<td>Gpm Port</td>
<td>PM_Reg: 0x00 [22]</td>
</tr>
<tr>
<td>C6Fh</td>
<td>Flash Rom Program Enable</td>
<td>PM_Reg: 0x00 [24]</td>
</tr>
<tr>
<td>CD0h-CD1h</td>
<td>PM2 Index/Data</td>
<td>Always enable</td>
</tr>
<tr>
<td>CD4h-CD5h</td>
<td>BIOS RAM Index/Data</td>
<td>PM_Reg: 0x20 [0] (Default enabled)</td>
</tr>
<tr>
<td>CD6h-CD7h</td>
<td>Power Management I/O register</td>
<td>PM_Reg: 0x00 [25]</td>
</tr>
</tbody>
</table>

2.2.2 Variable I/O Decode Ranges

<table>
<thead>
<tr>
<th>I/O Name</th>
<th>Description</th>
<th>Configure Register</th>
<th>Range Size (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM1_EVT</td>
<td>ACPI PM1a_EVT_BLK</td>
<td>PM_Reg: 0x60 &amp; 0x61</td>
<td>4</td>
</tr>
<tr>
<td>PM1_CNT</td>
<td>ACPI PM1a_CNT_BLK</td>
<td>PM_Reg: 0x62 &amp; 0x63</td>
<td>2</td>
</tr>
<tr>
<td>PM_TMR</td>
<td>ACPI PM_TMR_BLK</td>
<td>PM_Reg: 0x64 &amp; 0x65</td>
<td>4</td>
</tr>
<tr>
<td>P_BLK</td>
<td>ACPI P_BLK</td>
<td>PM_Reg: 0x66 &amp; 0x66</td>
<td>6</td>
</tr>
<tr>
<td>GPE0_EVT</td>
<td>ACPI GPE0_EVT_BLK</td>
<td>PM_Reg: 0x68 &amp; 0x67</td>
<td>8</td>
</tr>
<tr>
<td>SMI CMD Block*</td>
<td>SMI Command Block</td>
<td>PM_Reg: 0x6A &amp; 0x6B</td>
<td>2</td>
</tr>
<tr>
<td>Pma Cnt Block</td>
<td>PMa Control Block</td>
<td>PM_Reg: 0x6E &amp; 0x6F</td>
<td>1</td>
</tr>
<tr>
<td>SMBus Block</td>
<td>SMBus IO Space</td>
<td>PM_Reg: 0x2C &amp; 0x2D</td>
<td>16</td>
</tr>
</tbody>
</table>

Note:

- The SMI CMD block must be defined on the 16-bit boundary, i.e., the least significant nibble of the address must be zero (for example, B0h, C0h, etc.)
- The SMI CMD block consists of two ports – the SMI Command port at base address, and the SMI Status port at base address+1.
- The writes to SMI Status port will not generate an SMI. The writes to the SMI Command port will generate an SMI.
The SMI Command and SMI Status ports may be written individually as 8-bit ports, or together as a 16-bit port.

### 2.3 Memory Map

<table>
<thead>
<tr>
<th>Memory Range</th>
<th>Description</th>
<th>Enable Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000h-000D FFFFh</td>
<td>Main System Memory</td>
<td></td>
</tr>
<tr>
<td>0010 0000h-TOM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000E 0000h-000F FFFFh</td>
<td>Either PCI ROM or LPC ROM</td>
<td>PCI ROM: Enabled by ROM strap only, no register to program. LPC ROM: LPC_Reg: 0x68 &amp; LPC_Rom strap</td>
</tr>
<tr>
<td>FEC0 0000h-FEC0 00EFh</td>
<td>IOAPIC</td>
<td></td>
</tr>
<tr>
<td>FEC0 00F0h-FEC0 00F4h</td>
<td>Watch Dog Timer Base Address</td>
<td>PM_Reg: 0x48[0]</td>
</tr>
<tr>
<td>FED1 0000h-FED1 0100h</td>
<td>BIOS RAM base address</td>
<td>PM_Reg: 0x20[0]</td>
</tr>
<tr>
<td>FED4 0000h-FED4 3FFFh</td>
<td>TPM</td>
<td>Depends on configuration</td>
</tr>
<tr>
<td>FED6 1000h-FED6 1100h</td>
<td>GEC SHADOW ROM</td>
<td>LPC Reg9Ch [0]</td>
</tr>
<tr>
<td>FED8 0000h-FED8 0EFF</td>
<td>Acpi MMIO address</td>
<td>PM_Reg: 0x24[0]</td>
</tr>
<tr>
<td>FFC0 0000h-FFC7 FFFFh</td>
<td>FWH</td>
<td>LPC Reg: 0x70[3:0]</td>
</tr>
<tr>
<td>FF80 0000h-FF87 FFFFh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFC8 0000h-FFCF FFFFh</td>
<td>FWH</td>
<td>LPC Reg: 0x70[7:4]</td>
</tr>
<tr>
<td>FF88 0000h-FF8F FFFFh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFD0 0000h-FFD7 FFFFh</td>
<td>FWH</td>
<td>LPC Reg: 0x70[11:8]</td>
</tr>
<tr>
<td>FF90 0000h-FF97 FFFFh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFD8 0000h-FFDF FFFFh</td>
<td>FWH</td>
<td>LPC Reg: 0x70[15:12]</td>
</tr>
<tr>
<td>FF98 0000h-FF9F FFFFh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFE0 0000h-FFE7 FFFFh</td>
<td>FWH</td>
<td>LPC Reg: 0x70 [19:16]</td>
</tr>
<tr>
<td>FFA0 0000h-FFA7 FFFFh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFE8 0000h-FFE7 FFFFh</td>
<td>FWH</td>
<td>LPC Reg: 0x70[23:20]</td>
</tr>
<tr>
<td>FFA8 0000h-FFAF FFFFh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFF0 0000h-FFFF FFFFh</td>
<td>FWH</td>
<td>LPC Reg: 0x70[27:24]</td>
</tr>
<tr>
<td>FFF8 0000h-FFFF FFFFh</td>
<td>FWH</td>
<td>LPC Reg: 0x70[31:28]</td>
</tr>
<tr>
<td>FFB8 0000h-FFBF FFFFh</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 2.3.1 MMIO Programming for Legacy Devices

The Hudson-1 legacy devices LPC, IOAPIC, ACPI, TPM and Watchdog Timer require the base address of the Memory Mapped I/O registers to be assigned before these logic blocks are accessed. The Memory Mapped I/O register base address and its entire range should be mapped to non-posted memory region by programming the CPU register.

Below is a sample code for FCH MMIO Range calculation in System BIOS.
#include "SbPlatform.h"

//
// Declaration of local functions
//

typedef struct _OPTIMUM_FCH_MMIO_STRUCT {
    UINT16 Range0Base;
    UINT16 Range0Limit;
    UINT16 Range1Base;
    UINT16 Range1Limit;
    UINT16 Range2Base;
    UINT16 Range2Limit;
} OPTIMUM_FCH_MMIO_STRUCT;

/**
 * The FCH MMIO non-POST Range
 */
typedef struct _MMIO_RANGE_STRUCT {
    UINT16 Lpc0Base;
    UINT16 Lpc0Limit;
    UINT16 Lpc1Base;
    UINT16 Lpc1Limit;
    UINT16 SpiBase;
    UINT16 SpiLimit;
    UINT16 TmpBase;
    UINT16 TmpLimit;
    UINT16 HpetBase;
    UINT16 HpetLimit;
    UINT16 BiosRamBase;
    UINT16 BiosRamLimit;
    UINT16 WatchDogBase;
    UINT16 WatchDogLimit;
    UINT16 IoapicBase;
    UINT16 IoapicLimit;
    UINT16 AcpiMmioBase;
    UINT16 AcpiMmioLimit;
} MMIO_RANGE_STRUCT;

//
// Declaration of local functions
//

VOID fchMmioRangeCalculation (IN AMDSBCFG* pConfig, OUT OPTIMUM_FCH_MMIO_STRUCT* TempRange);

/**
 * fchMmioRangeCalculation - Calculatw FCH none-POST Mmio resource
 *
 * - Private function
 *
 * @param[in] pConfig - FCH configuration structure pointer.
 * @param[out] CFGMmioTableDescription - Optimum range for non-POST FCH MMIO range for IEV
 *
 */

©2012 Advanced Micro Devices, Inc.

AMD A45/A50M/A55E BIOS Developer's Guide
VOID
fchMmioRangeCalculation ( 
    IN       AMDSBCFG*   pConfig, 
    OUT      OPTIMUM_FCH_MMIO_STRUCT*  TempRange
)
{
    MMIO_RANGE_STRUCT fchTemp;
    UINT16 TempRange1BaseH;
    UINT16 TempRange1BaseL;
    UINT8 Rang2Flag;

    // Fill all FCH Mmio range
    // Lpc ROM 1 Base read from FCH
    ReadPCI ((LPC_BUS_DEV_FUN << 16) + SB_LPC_REG68, AccWidthUint16, &fchTemp.Lpc0Base);
    // Lpc ROM 1 Limit read from FCH
    ReadPCI ((LPC_BUS_DEV_FUN << 16) + SB_LPC_REG6A, AccWidthUint16, &fchTemp.Lpc0Limit);
    // Lpc ROM 2 Base read from FCH
    ReadPCI ((LPC_BUS_DEV_FUN << 16) + SB_LPC_REG6C, AccWidthUint16, &fchTemp.Lpc1Base);
    // Lpc ROM 2 Limit read from FCH
    ReadPCI ((LPC_BUS_DEV_FUN << 16) + SB_LPC_REG6E, AccWidthUint16, &fchTemp.Lpc1Limit);
    // Spi Base Address read from FCH
    ReadPCI ((LPC_BUS_DEV_FUN << 16) + SB_LPC_REGA0 + 2, AccWidthUint16, &fchTemp.SpiBase);
    // Spi base address limit is less than 64K
    // Tpm Base Address read from FCH
    ReadPCI ((LPC_BUS_DEV_FUN << 16) + 0x86, AccWidthUint16, &fchTemp.TempBase);
    // Tpm Limit Address read from FCH
    ReadPCI ((LPC_BUS_DEV_FUN << 16) + 0x8A, AccWidthUint16, &fchTemp.TempLimit);
    // HPET Base Address read from FCH
    ReadMEM (ACPI_MMIO_BASE + PMIO_BASE + SB_PMIOA_REG50 + 2, AccWidthUint16, &fchTemp.HpetBase);
    // HPET base address limit is less than 64K
    // BIOS RAM base Address read from FCH
    ReadMEM (ACPI_MMIO_BASE + PMIO_BASE + SB_PMIOA_REG20 + 2, AccWidthUint16, &fchTemp.BiosRamBase);
    // BIOS RAM address limit is less than 64K
    // WatchDog base address read from FCH
    ReadMEM (ACPI_MMIO_BASE + PMIO_BASE + SB_PMIOA_REG48 + 2, AccWidthUint16, &fchTemp.WatchDogBase);
    // WatchDog address limit is less than 64K
    // IoApic base address read from FCH
    ReadMEM (ACPI_MMIO_BASE + PMIO_BASE + SB_PMIOA_REG34 + 2, AccWidthUint16, &fchTemp.IoapicBase);
    // IoApic address limit is less than 64K
    // ACPI Mmio base address read from FCH
    ReadMEM (ACPI_MMIO_BASE + PMIO_BASE + SB_PMIOA_REG324 + 2, AccWidthUint16, &fchTemp.AcpiMmioBase);
    // ACPI Mmio address limit is less than 64K
    // Reserved Range0Base for LPC ROM location for CPU specific ROM cycle.
    // In CMs usually set LPC ROM2 for LPC ROM base address
    TempRange->Range0Base = fchTemp.Lpc1Base;
TempRange->Range0Limit = fchTemp.Lpc1Limit;

// Intent all other field (except LPC) combine to one big MMIO range.
TempRangeBaseL = 0xFEC0; // FCH default value for Watchdog base address (lowest)
TempRangeBaseH = 0xFED8; // FCH default value for ACPI MMIO base address (highest)
TempRange->Range1Base = 0xFEC0;
TempRange->Range1Limit = 0xFED8;
TempRange->Range2Base = 0;
TempRange->Range2Limit = 0;
Rang2Flag = 0x00;

if  (( fchTemp.SpiBase < TempRange1BaseL ) || ( TempRange1BaseH < fchTemp.SpiBase)) {
    Rang2Flag = 1;
    TempRange->Range2Base = fchTemp.SpiBase;
    TempRange->Range2Limit = fchTemp.SpiBase;
}

if (( fchTemp.TempBase != 0) && ( Rang2Flag != 1)) {
    if  (( fchTemp.TempBase < TempRange1BaseL ) || ( TempRange1BaseH < fchTemp.TempBase)) {
        Rang2Flag = 1;
        TempRange->Range2Base = fchTemp.TempBase;
        TempRange->Range2Limit = fchTemp.TempLimit;
    }
}

if (( fchTemp.HpetBase != 0) && ( Rang2Flag != 1)) {
    if  (( fchTemp.HpetBase < TempRange1BaseL ) || ( TempRange1BaseH < fchTemp.HpetBase)) {
        Rang2Flag = 1;
        TempRange->Range2Base = fchTemp.HpetBase;
        TempRange->Range2Limit = fchTemp.HpetBase;
    }
}

if (( fchTemp.BiosRamBase != 0) && ( Rang2Flag != 1)) {
    if  (( fchTemp.BiosRamBase < TempRange1BaseL ) || ( TempRange1BaseH < fchTemp.BiosRamBase)) {
        Rang2Flag = 1;
        TempRange->Range2Base = fchTemp.BiosRamBase;
        TempRange->Range2Limit = fchTemp.BiosRamBase;
    }
}

if (( fchTemp.WatchDogBase != 0) && ( Rang2Flag != 1)) {
    if  (( fchTemp.WatchDogBase < TempRange1BaseL ) || ( TempRange1BaseH < fchTemp.WatchDogBase)) {
        Rang2Flag = 1;
        TempRange->Range2Base = fchTemp.WatchDogBase;
        TempRange->Range2Limit = fchTemp.WatchDogBase;
    }
}

if (( fchTemp.IoapicBase != 0) && ( Rang2Flag != 1)) {
    if  (( fchTemp.IoapicBase < TempRange1BaseL ) || ( TempRange1BaseH < fchTemp.IoapicBase)) {
        Rang2Flag = 1;
        TempRange->Range2Base = fchTemp.IoapicBase;
        TempRange->Range2Limit = fchTemp.IoapicBase;
    }
}
if ((fchTemp.AcpiMmioBase != 0) && (Rang2Flag != 1)) {
    if (((fchTemp.AcpiMmioBase < TempRange1BaseL) || (TempRange1BaseH < fchTemp.AcpiMmioBase)) {
        Rang2Flag = 1;
        TempRange->Range2Base = fchTemp.AcpiMmioBase;
        TempRange->Range2Limit = fchTemp.AcpiMmioBase;
    }
}
if ((Rang2Flag != 1) && (fchTemp.Lpc0Base != 0)) {
    TempRange->Range2Base = fchTemp.Lpc0Base;
    TempRange->Range2Limit = fchTemp.Lpc0Limit;
}
3 Hudson-1 Early-POST Initialization

The system BIOS must configure the Hudson-1 at the very beginning of POST. Some of the settings will change depending on the OEM design, or on the newer revision chipset.

3.1 512K/1M ROM Enable

With a Hudson-1 design, there can be two possible ROM sources: PCI ROM and LPC ROM. Two pin straps (UseLpcRom, FWHDisable) determine where the ROM is (refer to the Hudson-1 databook). Upon system power on, the Hudson-1 enables 256K ROM by default. The BIOS needs to enable 512K ROM or up to 1M for LPC ROM, if required.

3.1.1 PCI ROM

<table>
<thead>
<tr>
<th>Control Bit</th>
<th>Description</th>
<th>256K ROM Setting (Default)</th>
<th>512K ROM Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM_Reg: 0x04 [12]</td>
<td>When set to 1, the address between FFF80000h to FFFDFFFFh will be directed to the PCI ROM interface.</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>PM_Reg: 0x04 [13]</td>
<td>When set to 1, the address between 0E0000h to 0EFFFFFFh will be directed to the PCI ROM interface.</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

3.1.2 LPC ROM

To use the LPC ROM, the pin straps UseLpcRom, FWHDisable must be set accordingly.

<table>
<thead>
<tr>
<th>Control Bit(s)</th>
<th>Description</th>
<th>Default</th>
<th>512K ROM Setting</th>
<th>1M ROM Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPC PCI Reg: 0x68</td>
<td>16-bit starting &amp; end address of the LPC ROM memory address range 1.</td>
<td>000E0000h</td>
<td>000E0000h</td>
<td>000E0000h</td>
</tr>
<tr>
<td>LPC PCI Reg: 0x6C</td>
<td>16-bit starting &amp; end address of the LPC ROM memory address range 2.</td>
<td>FFFE0000h</td>
<td>FFF80000h</td>
<td>FFF00000h</td>
</tr>
<tr>
<td>LPC PCI Reg: 0x48 [4:3]</td>
<td>Enable bits for LPC ROM memory address range 1 &amp; 2. Note: with pins straps set to LPC ROM, these two bits have no effect on Reg68 &amp; Reg6C.</td>
<td>00b</td>
<td>11b</td>
<td>11b</td>
</tr>
</tbody>
</table>

3.1.3 LPC ROM Read/Write Protect

The Hudson-1 allows all or a portion of the LPC ROM addressed by the firmware hub to be read protected, write protected, or both read and write protected. Four dword registers are provided to select up to 4 LPC ROM ranges for read or write protection. The ROM protection range is defined by the base address and the length. The base address is aligned at a 2K boundary and the address length can be from 1K to 512K in increments of 1K.
Register 50h, 54h, 58h, 5ch of Device 14h, Function 3

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Address</td>
<td>31:11</td>
<td>ROM Base address. The most significant 21 bits of the base address are defined in this field. Bits[10:0] of the base address are assumed to be zero. Base address, therefore, is aligned at a 2K boundary.</td>
</tr>
<tr>
<td>Length</td>
<td>10:2</td>
<td>These 9 bits (0-511) define the length from 1K to 512K in increments of 1K.</td>
</tr>
<tr>
<td>Read Protect</td>
<td>1</td>
<td>When set, the memory range defined by this register is read-protected. Reading any location in the range returns FFh.</td>
</tr>
<tr>
<td>Write Protect</td>
<td>0</td>
<td>When set, the memory range defined by this register is write-protected. Writing to the range has no effect.</td>
</tr>
</tbody>
</table>

Example:

Protect 32K LPC ROM starting with base address FFF80000.

Base address bits 31:11  1111 1111 1111 1000 0000 0b
Length 32K bit 10:2 = 31h = 000 0111 11 b
Read protect bit 1 = 1
Write protect bit 0 = 1
Register 50h = 1111 1111 1111 1000 0000 0000 0111 1111 b = FFF8007Fh

Note:

1. Registers 50h ~ 5Fh can be written once after the hardware reset. Subsequent writes to them have no effect.
2. Setting sections of the LPC ROM to either read or write protect will not allow the ROM to be updated by a flash programming utility. Most flash utilities write and verify ROM sectors, and will terminate programming if verification fails due to read protect.

3.1.4 SPI ROM Controller

The SPI ROM interface is a new feature added to the Hudson-1. Refer to the AMD Hudson-1 Register Reference Manual for more information on this feature. AMD will provide reference code for this feature.

Note: The LPC ROM Read/Write Protect mentioned in the previous paragraph also applies to SPI. Two strap pins, PCICLK0 and PCICLK1, determine the Hudson-1 boot up from LPC ROM or SPI ROM. There is no register status to reflect whether the current ROM interface is LPC or SPI.
3.2 Real Time Clock (RTC)

3.2.1 RTC Access

The internal RTC is divided into two sections: the clock and alarm functions (registers 0h to 0Dh), and CMOS memory (registers 0Eh to FFh). The clock and alarm functions must be accessed through I/O ports 70h/71h. The CMOS memory is accessed through I/O ports 72h/73h.

3.2.1.1 Special Locked Area in CMOS

Some CMOS memory locations may be disabled for read/write. PM_Reg: 0x56 defines the bits to disable these CMOS memory locations. Once set, the area is protected. It can only be disabled by cycling the system from S0 to G3 to S0 (RSM_RST# toggled) or by doing a system cold reset (SYS_Reset# toggled).

<table>
<thead>
<tr>
<th>RtcControl - RW – 8 bits - [PM_Reg: 56h]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Name</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>RTCProtect</td>
</tr>
<tr>
<td>RTCProtect</td>
</tr>
<tr>
<td>RTCProtect</td>
</tr>
<tr>
<td>RTCProtect</td>
</tr>
<tr>
<td>RTCProtect</td>
</tr>
</tbody>
</table>

3.2.1.2 Century Byte

The RTC has a century byte at CMOS location 32h. Century is stored in a single byte and the BCD format is used for the century (for example, 20h for the year 20xx). This byte is accessed using I/O ports 70h and 71h. The BIOS must set PMIO register 56h bit 12 to 1 to use this century byte at CMOS location 32h.

3.2.1.3 Date Alarm

The RTC has a date alarm byte. This byte is accessed as follows:

1. Set to 1 the RTC register 0Ah, bit 4, using I/O ports 70h and 71h.
2. Write Date Alarm in BCD to register 0Dh using I/O ports 70h and 71h.
3. Clear to 0 the RTC register 0Ah bit 4 using I/O ports 70h and 71h.

Note: It is important to clear RTC register 0Ah bit 4 to zero. Otherwise, the CMOS memory may not be accessed correctly from this point onward.
3.3 BIOS RAM

The Hudson-1 has 256 bytes of BIOS RAM. Data in this RAM is preserved until RSMRST# or S5 is asserted, or until power is lost.

This RAM is accessed using index and data registers at CD4h/CD5h.

3.4 Serial IRQ

The Hudson-1 supports serial IRQ which allows a single signal to report multiple interrupt requests. The Hudson-1 supports a message for 21 serial interrupts which include 15 IRQs, SMI#, IOCHK#, and 4 PCI interrupts.

**PM_Reg: 54h** is used for setting serial IRQ.

<table>
<thead>
<tr>
<th>Bits in PM_Reg: 54h</th>
<th>Description</th>
<th>Power-on Default</th>
<th>Recommended Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1 – Enables the serial IRQ function</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0 – Disables the serial IRQ function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1 – Active (quiet) mode</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 – Continuous mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5:2</td>
<td>Total number of serial IRQs = 17 + NumSerIqBits</td>
<td>0</td>
<td>0100b</td>
</tr>
<tr>
<td></td>
<td>0 – 17 serial IRQs (15 IRQs, SMI#, IOCHK#)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 – 18 serial IRQs (15 IRQs, SMI#, IOCHK#, INTA#)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15 - 32 serial IRQ's</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Hudson-1 serial IRQ can support 15 IRQs, SMI#, IOCHK#, INTA#, INTB#, INTC#, and INTD#.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1:0</td>
<td>Number of clocks in the start frame</td>
<td>0</td>
<td>00b</td>
</tr>
</tbody>
</table>

Note: BIOS should enter the continuous mode first when enabling the serial IRQ protocol so that the Hudson-1 can generate the start frame.
3.5 SubSystemID and SubSystem Vendor ID

SubSystem ID and SubSystem Vendor ID can be programmed in various functions of Hudson-1 register 2Ch. These registers are write-once registers. For example, to program a SubSystem vendor ID of 1002h and SubSystem ID of 4341h in AC97 device 14h, function 5, use the following assembly language sample code:

```
mov eax,8000A52Ch
mov dx,0CF8h
out dx,eax
mov dx,0CFCh
mov eax,43411002h
out dx,eax
```

3.6 System Restart after Power Fail

The way the system restarts following a power-fail/power-restore cycle depends on the setting of PMIO register 5Bh [bits 1:0].

<table>
<thead>
<tr>
<th>PMIO Register 5Bh bits[1:0]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00b or 10b</td>
<td>The system will remain off until the power button is pressed.</td>
</tr>
<tr>
<td>01b</td>
<td>The system will always restart after the power is restored.</td>
</tr>
<tr>
<td>11b</td>
<td>At power-up the system will either restart or remain off depending on the state of the system at power failure. If the system was on when the power failed, the system will restart at power-up. If the system was off when the power failed, the system will remain off after the power is restored. Pressing the power button is required to restart the system.</td>
</tr>
</tbody>
</table>

Notes on programming the PMIO register 5Bh:

1. Bits[3:0] should be used for programming. Bits[7:4] are read-only bits and reflect the same values as bits[3:0].

2. Bit 2 is used by the hardware to save the power on/off status. This bit should not be modified during software/BIOS programming, however, its value should be restored (effectively setting a value of ‘1’ as will be the case after every power up) upon every access into this register. The BIOS programmer should always read PMIO register 5Bh, modify bit3 and bits[1:0] as required, and write back the PMIO register 5Bh.

3. PM_Reg: 5Bh is initialized on every cold boot (G3->S5->S0 transition); however, it is also required to restore the settings in bits [7, 5:4] to bits [3, 1:0] following any SYS_RST# or RSMRST# assertion.
3.6.1 Power Fail and Alarm Setup

The state of the machine after the power-fail/power-restore cycle is controlled by PMIO register 5Bh bits[1:0] as described above. This programming can be over-ridden for the special case when the alarm is set. When both the alarm and the PMIO register 5Bh bit3 are set, the system will restart after the power is restored, regardless of how register 5Bh bits [1:0] are defined.
## 4 PCI IRQ Routing

### 4.1 PCI IRQ Routing Registers

The Hudson-1 uses one pair of I/O ports for PCI IRQ routing. The ports are at C00h/C01h.

<table>
<thead>
<tr>
<th>Address</th>
<th>Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C00h</td>
<td>PCI_Intr_Index</td>
<td>PCI interrupt index. Selects which PCI interrupt to map</td>
</tr>
<tr>
<td>0h</td>
<td>INTA#</td>
<td></td>
</tr>
<tr>
<td>1h</td>
<td>INTB#</td>
<td></td>
</tr>
<tr>
<td>2h</td>
<td>INTC#</td>
<td></td>
</tr>
<tr>
<td>3h</td>
<td>INTD#</td>
<td></td>
</tr>
<tr>
<td>4h</td>
<td>INTE#</td>
<td></td>
</tr>
<tr>
<td>5h</td>
<td>INTF#</td>
<td></td>
</tr>
<tr>
<td>6h</td>
<td>INTG#</td>
<td></td>
</tr>
<tr>
<td>7h</td>
<td>INTH#</td>
<td></td>
</tr>
<tr>
<td>8h</td>
<td>Misc</td>
<td></td>
</tr>
<tr>
<td>9h</td>
<td>Misc0</td>
<td></td>
</tr>
<tr>
<td>Ah</td>
<td>Misc1</td>
<td></td>
</tr>
<tr>
<td>Bh</td>
<td>Misc2</td>
<td></td>
</tr>
<tr>
<td>Ch</td>
<td>INTA from serial irq</td>
<td></td>
</tr>
<tr>
<td>Dh</td>
<td>INTB from serial irq</td>
<td></td>
</tr>
<tr>
<td>Eh</td>
<td>INTC from serial irq</td>
<td></td>
</tr>
<tr>
<td>Fh</td>
<td>INTD from serial irq</td>
<td></td>
</tr>
<tr>
<td>10h</td>
<td>SCI</td>
<td></td>
</tr>
<tr>
<td>11h</td>
<td>SMBUS0</td>
<td></td>
</tr>
<tr>
<td>12h</td>
<td>ASF</td>
<td></td>
</tr>
<tr>
<td>13h</td>
<td>HD audio</td>
<td></td>
</tr>
<tr>
<td>14h</td>
<td>FC</td>
<td></td>
</tr>
<tr>
<td>15h</td>
<td>GEC</td>
<td></td>
</tr>
<tr>
<td>16h</td>
<td>PerMon</td>
<td></td>
</tr>
<tr>
<td>20h</td>
<td>IMC INT0</td>
<td></td>
</tr>
<tr>
<td>21h</td>
<td>IMC INT1</td>
<td></td>
</tr>
<tr>
<td>22h</td>
<td>IMC INT2</td>
<td></td>
</tr>
<tr>
<td>23h</td>
<td>IMC INT3</td>
<td></td>
</tr>
<tr>
<td>24h</td>
<td>IMC INT4</td>
<td></td>
</tr>
<tr>
<td>25h</td>
<td>IMC INT5</td>
<td></td>
</tr>
<tr>
<td>30h</td>
<td>Dev18 (USB) IntA#</td>
<td></td>
</tr>
<tr>
<td>31h</td>
<td>Dev18 (USB) IntB#</td>
<td></td>
</tr>
<tr>
<td>32h</td>
<td>Dev19 (USB) IntA#</td>
<td></td>
</tr>
<tr>
<td>33h</td>
<td>Dev19 (USB) IntB#</td>
<td></td>
</tr>
<tr>
<td>34h</td>
<td>Dev22 (USB) IntA#</td>
<td></td>
</tr>
<tr>
<td>35h</td>
<td>Dev22 (USB) IntB#</td>
<td></td>
</tr>
<tr>
<td>36h</td>
<td>Dev20 (USB) IntC#</td>
<td></td>
</tr>
<tr>
<td>40h</td>
<td>IDE pci interrupt</td>
<td></td>
</tr>
<tr>
<td>50h</td>
<td>GPPInt0</td>
<td></td>
</tr>
<tr>
<td>51h</td>
<td>GPPInt1</td>
<td></td>
</tr>
<tr>
<td>52h</td>
<td>GPPInt2</td>
<td></td>
</tr>
<tr>
<td>53h</td>
<td>GPPInt3</td>
<td></td>
</tr>
<tr>
<td>C01h</td>
<td>PCI_Intr_Data</td>
<td>0 ~ 15 : IRQ0 to IRQ15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IRQ0, 2, 8, 13 are reserved</td>
</tr>
</tbody>
</table>
4.2  PCI IRQ BIOS Programming

PCI IRQs are assigned to interrupt lines using I/O ports at C00h and C01h in index/data format. The register C00h is used as the index as written with index number 0 through 0Ch as described in section 4.1. Register C01h is written with the interrupt number as data.

The following assembly language example assigns INTB# line to interrupt 10 (0Ah).

```assembly
mov dx,0C00h ; To write to IO port C00h
mov al,02h  ; Index for PCI IRQ INTB# as defined in section 4.1
out dx,al  ; Index is now set for INTB#
mov dx,0C01h ; To write interrupt number 10 (0Ah)
mov al,0Ah  ; Data is interrupt number 10 (0Ah)
out dx,al  ; Assign IRQB# to interrupt 10
```
4.3 Integrated PCI Devices IRQ Routing

In the Hudson-1, AC'97 and USB require a PCI IRQ. Internally they are routed to different PCI INT#s.

<table>
<thead>
<tr>
<th>Device</th>
<th>Reg3Dh of PCI Device</th>
<th>PCI INT#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus 0:Device 14h:Function 1</td>
<td>02</td>
<td>INTB#</td>
<td>IDE Controller*</td>
</tr>
<tr>
<td>Bus 0:Device 14h: Function 2</td>
<td>01</td>
<td>INTA#</td>
<td>High Definition Audio</td>
</tr>
<tr>
<td>Bus 0:Device 14h: Function 5</td>
<td>03</td>
<td>INTC#</td>
<td>USB #4 OHCI Controller</td>
</tr>
<tr>
<td>Bus 0:Device 12h:Function 0</td>
<td>01</td>
<td>INTC#</td>
<td>USB #1 OHCI Controller #0</td>
</tr>
<tr>
<td>Bus 0:Device 12h:Function 2</td>
<td>02</td>
<td>INTB#</td>
<td>USB #1 EHCI Controller</td>
</tr>
<tr>
<td>Bus 0:Device 13h: Function 0</td>
<td>01</td>
<td>INTC#</td>
<td>USB #2 OHCI Controller #0</td>
</tr>
<tr>
<td>Bus 0:Device 13h: Function 2</td>
<td>02</td>
<td>INTB#</td>
<td>USB #2 EHCI Controller</td>
</tr>
<tr>
<td>Bus 0:Device 16h: Function 0</td>
<td>01</td>
<td>INTC#</td>
<td>USB #3 OHCI Controller #0</td>
</tr>
<tr>
<td>Bus 0:Device 16h: Function 2</td>
<td>02</td>
<td>INTB#</td>
<td>USB #3 EHCI Controller</td>
</tr>
<tr>
<td>Bus 0:Device 11h:Function 0</td>
<td>01</td>
<td>INTD#</td>
<td>SATA Controller #2</td>
</tr>
</tbody>
</table>

* This is implemented in current CIMx module reference code

4.4 PCI IRQ Routing for APIC Mode

<table>
<thead>
<tr>
<th>PCI IRQ</th>
<th>APIC Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTA#</td>
<td>16</td>
</tr>
<tr>
<td>INTB#</td>
<td>17</td>
</tr>
<tr>
<td>INTC#</td>
<td>18</td>
</tr>
<tr>
<td>INTD#</td>
<td>19</td>
</tr>
<tr>
<td>INTE#</td>
<td>20</td>
</tr>
<tr>
<td>INTF#</td>
<td>21</td>
</tr>
<tr>
<td>INTG#</td>
<td>22</td>
</tr>
<tr>
<td>INTH#</td>
<td>23</td>
</tr>
</tbody>
</table>
5 SMBus Programming

The Hudson-1 SMBus (System Management Bus) complies with SMBus Specification Version 2.0.

5.1 SMBus Timing

The SMBus frequency can be adjusted using different values in an 8-bit I/O register at the SMBus base + 0Eh location.

The SMBus frequency is set as follows:

\[ \text{SMBus Frequency} = \frac{\text{Primary Alink Clock}}{(\text{Count in index 0Eh} \times 4)} \]

The power-up default value in register 0Eh is A0h, therefore, the default frequency is \((66\text{MHz})/(160 \times 4)\), or approximately 103 KHz.

The minimum SMBus frequency can be set with the value FFh in the register at index 0Eh which yields the following: \((66\text{MHz})/(255 \times 4) = 64.7\text{ KHz.}\)

5.2 SMBus Host Controller Programming

<table>
<thead>
<tr>
<th>Step</th>
<th>Descriptions</th>
<th>Register in SMBus I/O Space</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wait until SMBus is idle.</td>
<td>Reg00h[0]</td>
<td>0 – Idle 1 – Busy</td>
</tr>
<tr>
<td>2</td>
<td>Clear SMBus status.</td>
<td>Reg00h[4:1]</td>
<td>Write all 1’s to clear</td>
</tr>
<tr>
<td>3</td>
<td>Set SMBus command.</td>
<td>Reg03h</td>
<td>The command will go to SMBus device.</td>
</tr>
<tr>
<td>4</td>
<td>Set SMBus device address with read/write protocol</td>
<td>Reg04h</td>
<td>Bit7:1 – address Bit0 – 1 for read, 0 for write</td>
</tr>
<tr>
<td>5</td>
<td>Select SMBus protocol</td>
<td>Reg02h[4:2]</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Do a read from Reg02 to reset the counter if it’s going to be a block read/write operation</td>
<td>Reg02h</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Set low byte when write command</td>
<td>Reg05h</td>
<td>Byte command – It is the written data Word command – It is the low byte data Block command – It is block count Others – Don’t care</td>
</tr>
<tr>
<td>8</td>
<td>Set high byte when write command</td>
<td>Reg06h</td>
<td>Word command – It is the high byte data Others – Don’t care</td>
</tr>
<tr>
<td>9</td>
<td>Write the data when block write</td>
<td>Reg07h</td>
<td>Block write – write data one by one to it Others – Don’t care</td>
</tr>
<tr>
<td>10</td>
<td>Start SMBus command execution</td>
<td>Reg02h[6]</td>
<td>Write 1 to start the command</td>
</tr>
<tr>
<td>11</td>
<td>Wait for host not busy</td>
<td>Reg00h[0]</td>
<td></td>
</tr>
<tr>
<td>Step</td>
<td>Action</td>
<td>Register</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>---------------------------------------------</td>
<td>----------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>12</td>
<td>Check status to see if there is any error</td>
<td>Reg00h[4:2]</td>
<td>With 1 in the bit, there is error</td>
</tr>
<tr>
<td>13</td>
<td>Read data</td>
<td>Reg05h</td>
<td>Byte command – It is the read data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Word command – It is the low byte data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Block command – It is block count</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Others – Don’t care</td>
</tr>
<tr>
<td>14</td>
<td>Read data</td>
<td>Reg06h</td>
<td>Word command – It is the high byte data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Others – Don’t care</td>
</tr>
<tr>
<td>15</td>
<td>Read the data when block write</td>
<td>Reg07h</td>
<td>Block read – read data one by one.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Others – Don’t care</td>
</tr>
</tbody>
</table>

The following flow chart illustrates the steps in programming the SMBus host controller.
Fig 3: Programming the SMBus Host Controller
6 Serial ATA (SATA)

The Hudson-1 SATA controller has two main features:

1. Supports six SATA ports.

2. Supports a unique architecture that allows the user to configure the SATA controller to work in conjunction with the IDE (PATA) controller to provide configurations that cannot be supported with the SATA controller alone. This feature is referred to as “Combined Mode” in this document.

In Combined Mode, the SATA controller can be configured as either AHCI mode or RAID mode and supports up to four SATA ports. Ports 0:3 are assigned for this configuration. The other two SATA ports will be configured as PATA ports and function in IDE mode. Two SATA ports (port 4 and port 5) share one IDE channel (could be either Primary or Secondary channel) from the IDE (PATA) controller.

Alternatively, the SATA controller can be configured as IDE mode supporting up to six IDE channels. In this configuration the SATA ports will be assigned to the Primary / Secondary channels as defined in the table below. The configuration for six IDE ports can also be achieved in two modes simultaneously by using the combined mode, i.e., two IDE ports can be configured to work in Legacy mode while the other four ports can be configured to work in Native or Compatibility mode.

### SATA Port Assignment in Combined IDE Mode

<table>
<thead>
<tr>
<th>Port Number</th>
<th>Primary, Secondary, Master / Slave Assignment</th>
<th>SATA Drive Controlled by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port 0</td>
<td>Primary master</td>
<td>SATA controller</td>
</tr>
<tr>
<td>Port 1</td>
<td>Secondary master</td>
<td>SATA controller</td>
</tr>
<tr>
<td>Port 2</td>
<td>Primary slave</td>
<td>SATA controller</td>
</tr>
<tr>
<td>Port 3</td>
<td>Secondary slave</td>
<td>SATA controller</td>
</tr>
<tr>
<td>Port 4</td>
<td>Primary (Secondary) master</td>
<td>PATA controller</td>
</tr>
<tr>
<td>Port 5</td>
<td>Primary (Secondary) slave</td>
<td>PATA controller</td>
</tr>
</tbody>
</table>

The following figure shows the various combined mode configurations.
Fig 4: Combined Mode Configurations

Note: In the IDE combined mode, the Microsoft® inbox driver will control all PATA drives showing all devices under two physical IDE controllers.
6.1 Device ID

The Hudson-1 SATA will have different device IDs for different drivers as these are distinct devices from the driver point of view. In a non-fresh installed condition, Windows® will match the 4 IDs (vendor ID, device ID, sub-system ID and sub-vendor ID) first, and if they are matched, it will load the driver and will not check the sub-class code. This will result in a blue screen in Windows XP if the SATA RAID driver is loaded with the SATA controller in IDE mode, and device ID is shared.

<table>
<thead>
<tr>
<th>Device ID</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>4390</td>
<td>SATA in IDE mode</td>
</tr>
<tr>
<td>4391</td>
<td>SATA in AHCI mode with Microsoft® driver</td>
</tr>
<tr>
<td>4392</td>
<td>SATA in RAID mode with Promise non-Raid 5 driver</td>
</tr>
<tr>
<td>4393</td>
<td>SATA in RAID mode with Promise Raid 5 driver</td>
</tr>
</tbody>
</table>
6.2 SATA Controller Operating Modes

Whenever SATA is set to any of the IDE modes (native IDE, legacy IDE, IDE→AHCI) and the Combined Mode is set to OFF, only four ports (0-3) can be supported by the SATA controller, while the other two ports (4-5) cannot be used.

When the Combined Mode is ON, ports 4, 5 will always be connected through the PATA controller, meaning that any device connected to this port will be shown as a PATA IDE device.
7 APIC Programming

With the AMD integrated chipset solution, the BIOS needs to program both the Northbridge (in the APU) and the FCH in order to support APIC.

7.1 Northbridge APIC Enable

There are three bits in the Northbridge that the BIOS should set before enabling APIC support.

- Reg4C bit[1] - This bit should be set to enable. It forces the CPU request with address 0xFECx_xxxx to the FCH.
- Reg4C bit[18] - This bit should be set to enable. It sets the Northbridge to accept MSI with address 0xFEEx_xxxx from the FCH.

7.2 FCH APIC Enable

There are two bits in the PM_Reg that the BIOS should set before enabling APIC support.

- Reg34 bit[0] = 1 to enable the APIC function.
- Reg34 bit[1] = 1 to enable the xAPIC function. It is only valid if bit[0] is being set.

7.3 IOAPIC Base Address

The IOAPIC base address can be defined at PM_Reg: 34h bit[5-31]. The power-on default value is FEC00000h.

Note: This register is 32-bit access only. The BIOS should not use the byte restore mechanism to restore its value during S3 resume.

7.4 APIC IRQ Assignment

Hudson-1 has IRQ assignments under APIC mode as follows:

- IRQ0~15 – Legacy IRQ
- IRQ 16 – PCI INTA
- IRQ 17 – PCI INTB
- IRQ 18 – PCI INTC
- IRQ 19 – PCI INTD
- IRQ 20 – PCI INTE
- IRQ 21 – PCI INTF
- IRQ 22 – PCI INTG
- INT 23 – PCI INTH
- IRQ 09 – ACPI SCI

SCI is still as low-level trigger with APIC enabled.

7.5 APIC IRQ Routing

During the BIOS POST, the BIOS will do normal PCI IRQ routing through port C00h/C01h. Once
APIC is fully enabled by the operating system, routing in C00h/C01 must be all cleared to zero.

The following is a sample ASL code that may be incorporated into the BIOS:

```
OperationRegion (PIRQ, SystemIO, 0xC00, 0x2)
Field (PIRQ, ByteAcc, NoLock, Preserve) {
    PIDX, 8, // Index port
    PDAT, 8  // Data port
}
IndexField (PIDX, PDAT, ByteAcc, NoLock, Preserve) {
    PIRA, 8, // INT A
    PIRB, 8, // INT B
    PIRC, 8, // INT C
    PIRD, 8, // INT D
    PIRE, 8, // INT E
    PIRF, 8, // INT F
    PIRG, 8, // INT G
    PIRH, 8, // INT H
    Offset (0x10),
    PIRS, 8, // SCI
    Offset (0x13),
    HDAD, 8, // HD Audio
    Offset (0x15),
    GEC_, 8, // GEC
    Offset (0x30),
    USB1, 8, // USB1
    USB2, 8, // USB2
    USB3, 8, // USB3
    USB4, 8, // USB4
    USB5, 8, // USB5
    USB6, 8, // USB6
    USB7, 8, // USB7
    Offset (0x40),
    IDE_, 8, // IDE
    SATA, 8, // SATA
    Offset (0x50),
    GPP0, 8, // GPP0
    GPP1, 8, // GPP1
    GPP2, 8, // GPP2
    GPP3, 8, // GPP3
}
```
8 UMI Bridge

The registers are accessed using an address-register/data-register mechanism. The address register is AB_INDEX [31:0], and the data register is AB_DATA [31:0].

<table>
<thead>
<tr>
<th>31:30</th>
<th>29:17</th>
<th>16:2</th>
<th>1:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegSpace[1:0]</td>
<td>Reserved</td>
<td>Register address[16:2]</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**AB_INDEX [31:0]**

<table>
<thead>
<tr>
<th>31:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data[31:0]</td>
</tr>
</tbody>
</table>

**AB_DATA [31:0]**

<table>
<thead>
<tr>
<th>RegSpace[1:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>00b</td>
</tr>
<tr>
<td>01b</td>
</tr>
<tr>
<td>10b</td>
</tr>
<tr>
<td>11b</td>
</tr>
</tbody>
</table>

**Definition of RegSpace[1:0]**

In order to read or write a particular register, the software will write the register address and the register space identifier to AB_INDEX and then do a read or write to AB_DATA. This is analogous to how PCI configuration reads and writes work through I/O addresses CF8h/CFCh.

The location of AB_INDEX in the I/O space is defined by the abRegBaseAddr register located at PMIO, register 0E0h. The AB_DATA register address is offset 4h from the AB_INDEX address. The address of the AB_INDEX must be 8 byte aligned.

<table>
<thead>
<tr>
<th>31:3</th>
<th>2:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseAddr[31:3]</td>
<td>Rsv</td>
</tr>
</tbody>
</table>

**abRegBAR [31:0] at PMIO, Register 0E0h**

AXCFG and ABCFG registers are accessed indirectly through AB_INDEX/AB_DATA. To read or write a particular register through AB_INDEX/AB_DATA, the register address and the register space identifier is first written to AB_INDEX. The specified register is then accessed by doing a read or write to AB_DATA (see the example below).

Access to AXINDC and AXINDP registers requires a second level of indirection. Registers in these spaces are addressed through the following indirection registers: AX_INDEXC/AX_DATAC and AX_INDEXP/AX_DATAP.
### Register Indirect Address

<table>
<thead>
<tr>
<th>Register</th>
<th>Indirect Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX_INDEXC</td>
<td>30h</td>
</tr>
<tr>
<td>AX_DATA</td>
<td>34h</td>
</tr>
<tr>
<td>AX_INDEXP</td>
<td>38h</td>
</tr>
<tr>
<td>AX_DATA_P</td>
<td>3Ch</td>
</tr>
</tbody>
</table>

**Example:** To write to register 21h in the INDEXC space with a data of 00, the following steps are required:

1. Out 30h to AB_INDEX. This will prepare to write register from INDEXC
2. Out 21h to AB_DATA. This will set register 21h of INDEXC
3. Out 34h to AB_INDEX. This will prepare to write data to register defined in steps 1 and 2 above
4. Out 00 to AB_DATA. This will write the data to the register defined in steps 1 and 2 above.

### 8.1 Programming Procedure

Indirect access is required to access both UMI Configuration and UMI Bridge Configuration register space. The programming procedure is as follows:

**Write:**

1. Set the UMI bridge register access address. This address is set at PMIO, register 0E0h. This is an I/O address and needs to be set only once after power-up. The I/O address must be on an 8-byte boundary (i.e., 3 LS bits must be zeroes).

**Example:** To set CD8h as an UMI bridge register access address:

```asm
mov dx,0CD6h  ;
mov al,0E0h ; Index
out dx,al
mov dx,0CD7h  ;
mov al,0D8h ; Data
out dx,al
mov dx,0CD6h  ;
mov al,0E1h ; Index
out dx,al
mov dx,0CD7h  ;
mov al,00Ch ; Data
out dx,al
```

**Note:** Although the 32-bit I/O address is set for the UMI bridge (e.g.,
00000CD8h), the bridge may be accessed by a 16-bit address (i.e., 0CD8h). The MS word is set to 00 by default (see the example below).

2. Write the register address in the AB_INDEX.

   **Example:** To write to the UMI Bridge Configuration register space at 90h:
   
   ```
   mov dx, 0CD8h  ; I/O address index assigned to A-Link
   mov eax, 0C0000090h ; Bits[31:30] = 11 for A-Link Bridge register
   out dx, eax ; Register index is set
   mov dx, 0CDCh  ; I/O address for data
   mov eax, 00000001h  ; Power down 2 lanes to save power
   out dx, eax
   ```

   **Read:**
   
   Use a similar indirect procedure to read out the register value inside AB and BIF.

8.2 UMI Configuration DMA Access

To enable UMI Configuration DMA access, a specific register space needs to be configured first. This register is in the UMI register space that refers to port-specific configuration registers (see section 8 above for a description of the AB_INDEX register). When configuring the register, bit2 of byte 4 needs to be set to “1” to enable the DMA access.

Follow these steps to initialize UMI configuration DMA access (this initialization has to be performed during S3 wakeup also):

1. Issue an I/O write to AB_INDEX. The write data’s bit [31:29] should be 100b (binary). The register to be written is in the port-specific configuration register space, and bit [16:0] should be 0x4 (hex).

2. Issue an I/O write to AB_Data. This write data’s bits[31:0] should be 0x4h (i.e., 32b0000_0000_0000_0100 binary).

```
mov dx, 0CD8h  ; ALINK_ACCESS_INDEX
in eax, dx
and eax, NOT (0C001FFFFh)
or eax, 080000004h
out dx, eax
mov dx, 0CDCh  ; ALINK_ACCESS_DATA
mov eax, 04h
out dx, eax
```

;Write AB_INDEX 0x30
;Write AB_DATA 0x21
;Write AB_INDEX 0x34
;Write AB_DATA 0x00
mov      dx, 0CD8h  ; ALINK_ACCESS_INDEX
mov      eax, 30h
out dx, eax
mov      dx, 0CDCh  ; ALINK_ACCESS_DATA
mov      eax, 21h
out dx, eax
mov      dx, 0CD8h  ; ALINK_ACCESS_INDEX
mov      eax, 34h
out dx, eax
mov      dx, 0CDCh  ; ALINK_ACCESS_DATA
mov      eax, 00h
out dx, eax

8.3 Enabling Non-Posted Memory Write

The register index 10h of AXINDC bit9 should be set to 1.

mov      dx,AB_INDX  ; AB index register
mov      eax,30h  ; Address of AXINDC
out dx,eax  ; Set register address
mov      dx,AB_DATA  ; To write register address
mov      eax,10h  ; Write register address
out dx,eax
mov      dx,AB_INDX
mov      eax,34h  ; To write data portion of the AXINDC
out dx,eax
mov      dx,AB_DATA
in eax,dx  ; Read the current data
or al,200h  ; Set bit 9
out dx,eax  ; Write data back.
9 CIR Support

The integrated micro controller (IMC) in Hudson-1 provides the interface to connect two IR receivers and two IR transmitters. The two IR receivers are for Learning IR Data and Receiving IR Data and the two IR transmitters are for Transmitting IR Data. The controller has the ability to support wake from S5, S4, S3 or S1 states and is capable of storing and returning the wake identification code to the IR driver.

BIOS Requirements:
1. IMC firmware / CIMx program IMC, CIR base address and IRQ assignment of the IR.
2. CIMx initializes the ASL code with the exception of CIR pin configuration information.
3. Platform BIOS programs the CIR pin configuration in IMC logical Device 5.
4. Platform BIOS then add the information to the ASL code to declare the hardware capabilities (transmitter, receiver, jack, etc.) to the IR driver.

Driver Requirements:
The CIR driver is available on the AMD NDA site. This driver should be installed for enabling the function of the CIR.

9.1 Southbridge CIR

9.1.1 Host and CIR Communication

The platform BIOS needs to configure the CIR pins based on the platform requirement. Platform BIOS will program registers in logical device # 5 of the southbridge.

9.1.2 Logical Device Number 5 Registers

The CIR is designed as a logical device in the southbridge that complies with the Plug and Play ISA Specification. The logical device number of the CIR device is 5 and it contains the following registers that need to be updated by the Platform BIOS.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR_OpenDrain</td>
<td>4</td>
<td>0</td>
<td>R/W</td>
<td>Transmitter output pin type. 0: TX are push-pull driver. 1: TX are open-drain.</td>
</tr>
<tr>
<td>IR_Tx1</td>
<td>3</td>
<td>0</td>
<td>R/W</td>
<td>0: TX1 emitter output pin held high 1: TX1 pin outputs emitter signal</td>
</tr>
<tr>
<td>IR_Tx0</td>
<td>2</td>
<td>0</td>
<td>R/W</td>
<td>0: TX0 emitter output pin held high 1: TX0 pin outputs emitter signal</td>
</tr>
<tr>
<td>IR_Enable</td>
<td>1:0</td>
<td>0</td>
<td>R/W</td>
<td>00: IR Function will not control the IR Rx and Tx pins. 01: IR Function controls the RX and TX0 pins. 10: IR Function controls the RX and TX1 pins. 11: IR Function controls the RX and both TX pins.</td>
</tr>
</tbody>
</table>

IR_OpenDrain (bit 4) - Program this bit to 1 or 0 depending on the how it is connected to the CIR emitter.

IR_Tx1 and IR_Tx0 (bits 2 and 3) - Program these bits to ‘1’ to enable the transmit signal for the
corresponding Tx that is used on the platform.

**IR_Enable** (bits 0 and 1) - Program these bits to ‘1’ to enable the transmit signal for the corresponding Tx that is used on the platform.

<table>
<thead>
<tr>
<th>FieldName</th>
<th>Bits</th>
<th>Default</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR_TXINV</td>
<td>1</td>
<td>0</td>
<td>Read/Write</td>
<td>Invert the Transmit signal</td>
</tr>
<tr>
<td>IR_RXIN</td>
<td>0</td>
<td>0</td>
<td>Read/Write</td>
<td>Invert the Received signal</td>
</tr>
</tbody>
</table>

IR TX or RX signal can be inverted by programming bit 1:0. Depending on the CIR emitter requirements these bits can be programmed accordingly.

Note that the IMC firmware will set IMC_PortActive to 0x6E once the IMC is enabled because the power-up default value of 0x2E is usually used by most Super IO controllers. Therefore, an I/O conflict may occur when IMC and the Super IO controller are used together. If the value picked by the IMC firmware is still not suitable for the design, platform BIOS developers can change it by programming the IMC_PortActive field of the Southbridge LPC ISA bridge PCI configuration register A4h to the desired address.

LPC ISA Bridge (Device 20, Function 3):

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Bits</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMC_PortActive</td>
<td>0</td>
<td>1b</td>
<td>When set to 1, LPC can decode the address specified in IMC_PortAddress, otherwise LPC ignores it.</td>
</tr>
<tr>
<td>Addr15_1</td>
<td>15:1</td>
<td>0017h</td>
<td>When Addr15_1 is non-zero, and if an IO cycle from host has address[15:1] = Addr15_1, the cycle will be routed to IMC instead of to LPC bus. By default, address[15:0] = 002Eh or 002Fh will be routed. Read-only to host if IMC_PortHostAccessEn = 0.</td>
</tr>
</tbody>
</table>

Since the activation of Logical Device Number 5 and the address assignment of the registers are handled by CIMx during the IMC enabling process, the platform software needs only to program the value of the above registers to enable the CIR (refer to the Get_IMC_MsgReg_Base_Addr function in the sample code below).
; Find the IMC Config port address to access the configuration registers
; Input: None
; Output: DX = IMC Config port address

Get_IMC_Config_Port:
  mov  eax, 8000A3A4h ; LPC bridge configuration register A4h
  mov  dx, OCF8h
  out  dx, eax
  mov  dx, OCFCh
  in   eax, dx ; Read IMC_PortAddress register
  and  ax, 0FEh ; Mask out bit 0 to retain LPCCfg_A4[15:1]
  mov  dx, ax ; DX = Index Port address
  ret

; Find CIR base address
; Input: None
; Output: DX = CIR register base address

Get_IMC_CIR_Base_Addr:
  call  Get_IMC_Config_Port ; Get IMC Config port address
  mov  al, 5Ah ; Enter configuration state
  out  dx, al ; Write Index Port
  mov  al, 07h ; Select Logical Device Number register
  out  dx, al ; DX = Data Port
  inc  dx ; DX = Data Port
  mov  al, 05h ; Select logical device 5, CIR Registers
  out  dx, al
def dx ; DX = Index Port
  mov  al, 60h ; Read high byte of CIR register base address
  out  dx, al
  inc  dx ; DX = Data Port
  in   al, dx
  mov  bh, al ; BX = CIR base address bit[15:8]
def dx ; DX = Index Port
  mov  al, 61h ; Read low byte of CIR register base address
  out  dx, al
  inc  dx ; DX = Data Port
  in   al, dx
  mov  bl, al ; BX = CIR base address bit[7:0]
def dx ; DX = Index Port
  mov  al, 0A5h ; Exit configuration state
  out  dx, al
  mov  dx, bx ; DX = CIR register base address[15:0]
  ret
9.1.3 Sample Code to Read and Write CIR Registers

; Write an 8-bit value to a CIR register  
; Input:  AH = value to write  
; AL = message register index value  
; Output: None

Write_CIRReg:
    push ax  ; Save input parameters  
    call Get_IMC_CIR_Base_Addr; Get IMC CIR register base address  
    pop ax  ; Restore input parameters  
    out dx, al  ; Write index to CIR index register  
    mov al, ah  
    inc dx  ; DX = message data register  
    out dx, al  ; write data to message data register  
    dec dx
    ret

; Read from a CIR register  
; Input:  AL = message register index value  
; Output: AL = value read

Read_CIRReg:
    push ax  ; Save input parameters  
    call Get_IMC_CIR_Base_Addr; Get IMC CIR register base address  
    pop ax  ; Restore input parameters  
    out dx, al  ; Write index to message index register  
    inc dx  ; DX = message data register  
    in al, dx  ; AL = value read from message register requested
    ret

9.1.4 Sample Code to Enable Tx0 and Program it as Push Pull Driver

mov al, ABh ; AL = CIR pin config register index value  
call Read_CIRReg

mov ah, 05h ; AH = value to write  
and al, E0h ; clear bits 4:0  
or al, ah ; ah = contents to write back
mov al, ABh ; AL = CIR register index value  
call Write_CIRReg

9.1.5 Sample ASL Code for CIR

//Hudson-1 Embedded Controller IR device
Device(ECIR)
{
    Name(_HID,EISAID("AMDC001"))

    Method(_STA)
    {
        Store(0x00, Local1)
        If (LEqual(OSTY, 6)) //We need to use IR only for Vista
        {
            Store(STA(0x05), Local1)
        }
        Return(Local1)
    }

    Method(_CRS)
    {
        Name(RSRC,ResourceTemplate()
        {
            IO(Decode16,0x00,0x00,0x08,0x08)
            IRQNoFlags() {}
        })
        CreateByteField(RSRC,0x02,IO1) //IO port low
        CreateByteField(RSRC,0x03,IO2) //IO port high
        CreateByteField(RSRC,0x04,IO3) //IO port low
        CreateByteField(RSRC,0x05,IO4) //IO port high
        CreateWordField(RSRC,0x09,IRQV) //IRQ mask
        Acquire(ECMU, 5000)
        CFG(0x05)
        If (ACT)
        {
            Store(IOBL,IO1)
            Store(IOBH,IO2)
            Store(IOBL,IO3)
            Store(IOBH,IO4)
            Store(0x01,LOCAL0)
            ShiftLeft(LOCAL0,INT,IRQV)
        }
        XCFG()
        Release(ECMU)
        Return(RSRC)
    } // Method(_CRS)

    Name(_PRS,ResourceTemplate()
    {
        StartDependentFn(0,0)
        {
            IO(Decode16,0x550,0x550,0x1,0x8)
            IRQ(Edge,ActiveHigh,Shared) {0x5}
        }
        StartDependentFnNoPri()
        {
            IO(Decode16,0x650,0x650,0x1,0x8)
            IRQ(Edge,ActiveHigh,Shared) {0x5}
        }
        StartDependentFnNoPri()
        {
            IO(Decode16,0x550,0x550,0x1,0x8)
            IRQ(Edge,ActiveHigh,Shared) {0x3}
        }
    }
}
StartDependentFnNoPri()
{
  IO(Decode16,0x650,0x650,0x1,0x8)
  IRQ(Edge,ActiveHigh,Shared) (0x3)
}

EndDependentFn()
})

Method(_SRS,1)
{
  //Arg0 = PnP Resource String to set
  CreateByteField(Arg0,0x02,IO1)
  CreateByteField(Arg0,0x03,IO2)
  CreateWordField(Arg0,0x09,IRQV)
  Acquire(ECMU, 5000)
  CFG(0x05)
  Store(IO1,IOBL)
  Store(IO2,IOBH)
  FindSetRightBit(IRQV,Local0)
  Subtract(Local0,1,INT)
  Store(0x01,ACT)
  XCFG()
  Release(ECMU)
} // Method(_SRS)

Method(_PRW, 0) { Return(GPRW(0x17, 4)) } // can wakeup from S4 state

Method(_PSW, 1)
{
  If(Arg0)
  {
    Store(1, IRWF)
  }
  Else
  {
    Store(0, IRWF)
  }
}

//AMD specific control method to return board related IR info
Method(IRCF, 2)
{
  //Bit 0 - 7
  //Version number of silicon (Supports up to 16 version numbers).
  //For example
  //0x39 for A11, 0x3A for A12
  //Bit 8 v 10(Supports up to 7 transmitters. (AMD IR device supports 2 transmitters)
  //Number of TX ports
  //For example
  //0 for no emitters, 1 for one emitter
  //Bit 11 v 12(Supports up to 3 receivers. (AMD IR device supports 2 receivers)
  //Number of receivers
  //For example
  //0 for no receivers, 1 for one receiver
  //Bit 13 v 15
  //Receiver number for Learn.
  //0 for no learning receiver.

/Otherwise it is the mask value of learning receiver (3 bits to support maximum 3
receivers).

//bit16 to represent the presence of receiver LED on the hardware

Name(CONN, 0)
If (LEqual(ARG0, 0x01))
{   //Query platform configuration
    Or(RVID, CONN, CONN)
    Or(0x13200, CONN, CONN)
    Return(CONN)
}
If (LEqual(ARG0, 0x02))
{   //Emitter Jack connection info
    ShiftLeft(GP64, 1, CONN)
    Or(GP51, CONN, CONN)
    Return(CONN)
}
If (LEqual(ARG0, 0x03))
{   //Flash LED function
    If (LEqual(ARG1, 0x00))
    {    //Don't flash the LED
        Store(1, \G31O)
    }
    else
    {    //Flash the LED
        Store(0, \G31O)
        Sleep(100)
        Store(1, \G31O)
    }
}
If (LEqual(ARG0, 0x04))
{   //Set EC-ACPI Interrupt Config
    If (LEqual(ARG1, 0x00))
    {    //Set as edge triggered
        Store(0, \ACIR)
    }
    else
    {    //Set as level triggered
        Store(1, \ACIR)
    }
}
If (LOr (LEqual(ARG0, 0x00), LGreater(ARG0, 0x05)))
{   Store(0xFFFFFFFF, CONN)   //Return error is invalid function
    Return(CONN)
}   //_IRS
} //Device(ECIR)
10 SMI Programming

10.1 Workaround for SMI Command Port Status Byte

This workaround is required as per erratum # 20 in the Hudson-1 FCH Family Product Errata (PID # 47792).

If an IMC-enabled platform is using the SMI command port base +1 with byte (8 bit) access, all such access should be changed to use word (16 bit) access starting at the SMI command port base address.

For platforms that use the SMI command port status register, typically software will write to the command port status byte (base +1) and then write to the command port base consecutively to set up the software SMI. The following code example shows how to use word access instead of byte access:

```
// Issue command port SMI

if (*ArgumentBufferSize == 2) {
    //WriteIO (ACPI_SMI_DATA_PORT, AccWidthUint8, &bData);
    wValue = bIndex + (bData << 8);
    WriteIO (ACPI_SMI_CMD_PORT, AccWidthUint16, &wValue);
} else {
    WriteIO (ACPI_SMI_CMD_PORT, AccWidthUint8, &bIndex);
}
```

Reading from the command port status register will also require using word access:

```
VOID
SwGetContext(
    IN  DATABASE_RECORD    *Record,
    OUT SMM_CONTEXT        *Context
)
{
    EFI_STATUS  Status;
    UINT8       ApmCnt;

    Status = mSmst->SmmIo.Io.Read (
        &mSmst->SmmIo,
        SMM_IO_UINT16,
        Smi_CmdPort,
        0,
        &ApmCnt
    );

    ASSERT_EFI_ERROR (Status);

    Context->Sw.SwSmiInputValue = ApmCnt;
}
```
Note: The above examples assume that the setting up of SMI is done by writing to the Command port and Command port+1 consecutively when setting up a software SMI. If the Status byte is read independently, then care must be taken to disable the SMI command port before doing a word access to command port base address in order to prevent an SMI interrupt from being generated. The SMI command port can be enabled after the write is completed. The SMIControl75 field of the SmiControl4 register (bit [23:22] of SMI_Reg: B0h) is used to enable/disable SMI.
11 Legacy BIOS Implementation for Chipset Integration Module Extensive (Hudson-1 CIMx)

11.1 Introduction

CIMx-Hudson-1 introduces a similar interface and distribution model as the CIMx-SB7xx to help quickly integrate Hudson-1 FCH family support in customer products. It differs from the CIMx-SB7xx by using a new Push High interface which is the same as the AMD AGESA™ module used for the V5 series model. Since the APU/FCH use the same interface file, it helps to reduce the coding size of the interface file of the IBV (i.e., individual BIOS company or BIOS vendor).

11.2 CIMx Interface Calls Environment

Prior to calling any CIMx interface, the following is required:

1. Place CPU into 32-bit protected mode.
2. Set CS as 32-bit code segment with Base/Limit – 0x00000000/0xffffffff.
3. Set DS/ES/SS as 32-bit data segment with Base/Limit – 0x00000000/0xffffffff.

11.3 Interface Definition

All interface calls to CIMx-Hudson-1 binary are C-like calls to the Entry Point of the binary image.

\[
\text{void (*ImageEntryPointPtr)(void* Config)}
\]

11.3.1 FCH Power-On/Reset Initialization

Upon system power-on, or cold reset, there is minimal initialization required such as SMBus base address programming and enabling the legacy IO to bring the system to a working state. The BIOS should call this entry to B1 module at a very early stage during power on initialization.

CIMx has assumed a set of default values for all the build parameters such as BIOS size, SMBus base address, power management base addresses, etc. In order to use your own set of values for these configurable options, define the buildparameters structure and give the 32-bit physical pointer as input to the CIMx module. If the pointer is NULL or set to all 1, then the CIMx module uses the default parameters. Also the BIOS developer has the flexibility of changing the oem.h file to redefine the values, however, in this case, the CIMx module needs to be rebuilt for the new values to be included in the binary.

**Parameters**

Please refer to Hudson-1 CIMx implementation note.chm for further description.
11.3.2 FCH BIOS POST Initialization

Hudson-1 BIOS POST initialization is divided into 3 stages; “Before PCI Enumeration”, “After PCI Enumeration” and “Late POST Initialization”. Each of these needs the same inputs except that the function ID must be set up properly. The “Before PCI Enumeration” routine should be called before PCI devices are enumerated and resources are assigned. It is recommended to call the “Before PCI Enumeration” routine at very early POST after memory detection. The “After PCI Enumeration” routine should be called after the resources are assigned to the PCI devices since this routine initializes MMIO spaces of some of the devices. The “Late POST Initialization” routine should be called at the end of BIOS POST just before giving control to the operating system.

Parameters

Please refer to Hudson-1 CIMx implementation note.chm for further description.

11.3.3 S3 Resume Initialization

CIMx will do the necessary programming to prepare the FCH to resume from the sleep state. It is highly recommended to use the same copy of AMDSBCFG for BIOS post and S3 resume initialization. This will allow CIMx to exchange data between interface calls if necessary. There are two calls necessary during S3 resume time to restore the FCH to the previous state: The first call should be done before PCI devices configuration spaces are restored and the other call should be done after PCI devices configuration spaces are restored.

Parameters

Please refer to Hudson-1 CIMx implementation note.chm for further description.

11.3.4 Callback Interface Definition

Callback functions are supported in the CIMx module to enable the OEMs to hook at a specific location in the CIMx module. This will allow OEMs to do some specific initialization in between the CIMx functions. OEMs who require some callbacks can request AMD to add this call in CIMx.

Parameters

Please refer to Hudson-1 CIMx implementation note.chm for further description.
12 UEFI BIOS Implementation for Chipset Integration
Module Extensive (Hudson-1 CIMx)

12.1 Introduction

The platform interface and configuration driver (AMD platform interface) is the interface between
the UEFI core and the AMD chipset driver. It has two purposes: (1) to allow the platform to
configure the chipset driver behavior (device enabling/disabling, internal configuration change),
and (2) to provide a means to the platform to abstract some platform specific behavior (GPIO
toggling, hardware location, hardware support, etc.) and allow the driver to request interaction
and information from the platform. This driver is written both by AMD and the UEFI core provider.

The goal is to provide a driver that can be added to any code base with the minimum amount of
work and modification. Most calls are done through the entry points or callbacks and all the
platform code is centralized in one location.

The driver is built using the TianoCore tool chain (EDK) and verified in a variety of platform BIOS.
The EDK build files will be provided as part of the driver package to make it easy to integrate it in
any codebase.

12.2 CIMx Interface Calls Environment

The AMD chipset driver is composed of four parts:

1. PEI driver
2. DXE driver
3. AMD UEFI driver library
4. Core chipset driver

The first three are unique to the AMD UEFI chipset driver. The core chipset driver is shared with
the legacy driver and is not covered by this document. It is integrated in the UEFI driver as a
library since both PEI and DXE need to access its functions.

12.2.1 FCH PEI Module

The FCH is controlled by the AmdSbPei PEI. This PEI is responsible for the power-on
initialization of the FCH including recovery support, boot-mode update, SMBus control and the
production of a reset service.

The boot mode is only updated if the boot mode seen by the FCH has a higher priority (as
defined in the PI specification) than the current mode. Since the boot mode might not be final
when the FCH PEIm is entered, a function is provided in AMD_PEL_FCH_INIT_PPI to allow the
platform to query the module for the FCH boot mode. The resulting configuration is used by the
FCH PEI module for power-on initialization and then passed to DXE through a HOB.
Parameters

Please refer to Hudson-1 CIMx implementation note.chm for further description.

12.2.2 FCH DXE Driver

The FCH is initialized in three phases:

1. Before PCI initialization
2. After PCI initialization
3. Late FCH Init

The first phase is done on entry to the AmdSbDxe driver and this is assumed to happen before the PCI sub-system is initialized. This allows the driver to perform some early initialization on peripheral controllers before they are discovered by the core and publish the protocols supported by the FCH. The runtime FCH driver to support is also initialized and the necessary callbacks for the subsequent phases are registered.

The second phase occurs once PCI has been initialized by the core. It is performed upon the first installation of the PciIo protocol (EFI_PCI_IO_PROTOCOL_GUID) by the core, through a callback on the protocol installation with a TPL of TPL_NOTIFY. At that point the peripheral controllers have their resources assigned and the FCH driver can start using them. Based on platform hardware, procedures such as USB MMIO assignment, SATA port enumeration and audio configuration are performed.

The final phase occurs when the system is ready to boot and the ready to boot (EFI_EVENT_SIGNAL_READY_TO_BOOT) event has been signaled. A callback on the event with a TPL of TPL_NOTIFY is used to initiate this phase. The final FCH configuration, including HPET and SATA, is performed at this point. The FCH driver also publishes an "end of FCH init" protocol at the end of its initialization to allow other drivers to trigger on it.

Parameters

Please refer to Hudson-1 CIMx implementation note.chm for further description.