AMD Bolton FCH
BIOS Developer's Guide

Publication #	**51205**	Revision:	**3.00**
Issue Date:	**August 2014**		

Revision History

Date	Rev.	Description
August	3.00	Initial public release.

.

Table of Contents

1 Introduction

1.1 About this Manual

This manual provides guidelines for BIOS developers working with the Bolton-M3, Bolton-D3, Bolton-D4, and Bolton-E4 FCH (Fusion Controller Hub). It describes the BIOS and software modifications required to fully support the device.

Note that the term Bolton is used throughout this document to refer to all members of the Bolton FCH family.

In general, the information provide in this document is applicable to all members of the family; however where information is applicable to only one particular ASIC, suitable indication will be given. For more details on features differentiating the different members, please refer to their respective databooks.

Other documents on the Bolton are available at AMD's NDA site or from your AMD FAE representative.

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 Diagrams

AMD's Bolton FCHs integrate the key I/O, communications, and audio features required in a state-of-the-art PC into a single device. These products are specifically designed to operate with AMD's APU (Accelerating Processing Unit) products in both desktop and mobile PCs.

Figure 1 below shows the Bolton internal PCI devices and the major function blocks. (Note: Some blocks are not supported and the number of USB and SATA ports supported may vary depending on the members of the family. Refer to individual databooks for details on supported features.)

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.

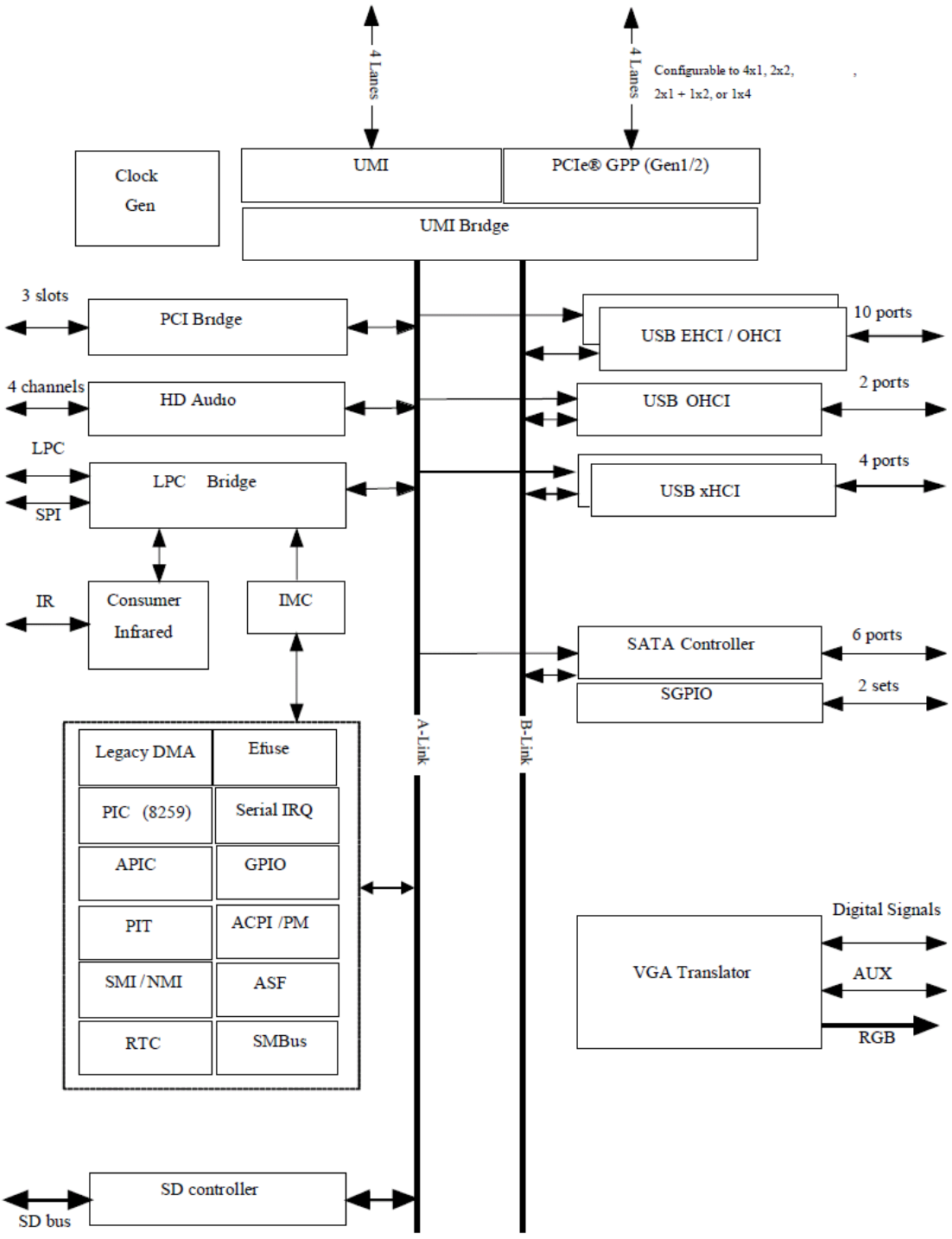

Figure 1 Bolton PCI Internal Devices and Major Function Blocks

2 Bolton Programming Architecture

2.1 PCI Devices and Functions

Bus:Device:Function	Function Description	Dev ID	Enable/Disable
Bus 0:Device 14h:Function 0	SMBus Controller	780Bh	Always enabled
Bus 0:Device 14h:Function 1	SATA (IDE) Controller	780Ch	PM_Reg: 0xDA bit3
Bus 0:Device 14h:Function 2	HD Audio Controller	780Dh	PM_Reg: 0xEB bit0
Bus 0:Device 14h:Function 3	LPC Controller	780Eh	PM_Reg: 0xEC bit0
Bus 0:Device 14h:Function 4	PCIB Bridge	780Fh	Always enabled
Bus 0:Device 14h:Function 6	GEC Controller	7811h	PM_Reg: 0xF6 bit0 cleared
Bus 0:Device 14h:Function 7	SD Flash Controller	7806h	PM_Reg: 0xE7 bit4 PM_Reg; 0xD3 bit 6
Bus 0:Device 10h:Function 0 Bus 0:Device 10h:Function 1	USB xHCI0 Controller USB xHCI1 Controller	7814h 7814h	Refer to Section 7 for further information.
Bus 0:Device 12h:Function 2 Bus 0:Device 13h:Function 2	USB #1 EHCI Controller USB #2 EHCI Controller	7808h 7808h	PM_Reg: 0xEF bit1, bit3,
Bus 0:Device 12h:Function 0 Bus 0:Device 13h:Function 0 Bus 0:Device 14h:Function 5	USB #1 OHCI Controller USB #2 OHCI Controller USB #4 OHCI Controller	7807h 7807h 7809h	PM_Reg: 0xEF bit 0, bit 2, bit 4
Bus 0:Device 11h:Function 0	Native/Legacy IDE Mode AHCI Mode (MS Driver) Non-Raid-5 Mode (Dot Hill) Raid5 Mode (Dot Hill) AHCI Mode (AMD Driver)	7800h 7801h 7802h 7803h 7804h	PM_Reg: 0xDA bit0
Bus0:Device 15h:Function 0 Bus0:Device 15h:Function 1 Bus0:Device 15h:Function 2 Bus0:Device 15h:Function 3	GPP Port 0 GPP Port 1 GPP Port 2 GPP Port 3	43A0h 43A1h 43A2h 43A3h	CIMx Input parameter PortPresent for each GPP port

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 – Bolton Proprietary Ports

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

2.2.2 *Variable I/O Decode Ranges*

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

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

Memory Range	Description	Enable Bit
0000 0000h-000D FFFFh 0010 0000h-TOM	Main System Memory	
000E 0000h-000F FFFFh	LPC ROM	LPC ROM : LPC Reg68h & LPC_Rom strap
FEC0 0000h-FEC0 00EFh	IOAPIC	
FEC0 00F0h-FEC0 00F4h	Watch Dog Timer Base Address * Recommended	PM_Reg: 0x48[0]
FED1 0000h-FED1 0100h	BIOS RAM base address * Recommended	PM_Reg: 0x20[0]
FED4 0000h-FED4 3FFFh	TPM	Depends on configuration
FED6 1000h-FED6 1100h	GEC SHADOW ROM	LPC Reg9Ch [0]
FED8 0000h-FED8 0EFF	Acpi MMIO address * Recommend	PM_Reg: 0x24[0]
FFC0 0000h-FFC7 FFFFh FF80 0000h-FF87 FFFFh	FWH	LPC Reg: 0x70[3:0]
FFC8 0000h-FFCF FFFFh FF88 0000h-FF8F FFFFh	FWH	LPC Reg: 0x70[7:4]
FFD0 0000h-FFD7 FFFFh FF90 0000h-FF97 FFFFh	FWH	LPC Reg: 0x70[11:8]
FFD8 0000h-FFDF FFFFh FF98 0000h-FF9F FFFFh	FWH	LPC Reg: 0x70[15:12]
FFE0 0000h-FFE7 FFFFh FFA0 0000h-FFA7 FFFFh	FWH	LPC Reg: 0x70 [19:16]
FFE8 0000h-FFEF FFFFh FFA8 0000h-FFAF FFFFh	FWH	LPC Reg: 0x70[23:20]
FFF0 0000h-FFF7 FFFFh FFB0 0000h-FFB7 FFFFh	FWH	LPC Reg: 0x70[27:24]
FFF8 0000h-FFFF FFFFh FFB8 0000h-FFBF FFFFh	FWH	LPC Reg: 0x70[31:28]

2.3.1 *MMIO Programming for Legacy Devices*

The Bolton 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 IBV
 *
 */
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 then 64K

  // Tpm Base Address read from FCH
  ReadPCI ((LPC_BUS_DEV_FUN << 16) + 0x86, AccWidthUint16, &fchTemp.TmpBase);
  // Tpm Limit Address read from FCH
  ReadPCI ((LPC_BUS_DEV_FUN << 16) + 0x8A, AccWidthUint16, &fchTemp.TmpLimit);
  // 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 then 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 then 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 then 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 then 64K

  // ACPI Mmio base address read from FCH
  ReadMEM (ACPI_MMIO_BASE + PMIO_BASE + SB_PMIOA_REG24 + 2, AccWidthUint16, &fchTemp.AcpiMmioBase);
  // ACPI Mmio address limit is less then 64K

  // Reserved Range0Base for LPC ROM location for CPU specific ROM cycle.
  // In CIMx usually set LPC ROM2 for LPC ROM base address
  TempRange->Range0Base = fchTemp.Lpc1Base;
  TempRange->Range0Limit = fchTemp.Lpc1Limit;

  // Intent all other filed (except LPC) combine to one big MMIO range.
  TempRange1BaseL = 0xFEC0;        // FCH default value for Watchdoag base address (lowerest)
  TempRange1BaseH = 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.TmpBase != 0) && ( Rang2Flag != 1)) {
  if  (( fchTemp.TmpBase < TempRange1BaseL ) || ( TempRange1BaseH < fchTemp.TmpBase)) {
    Rang2Flag = 1;
    TempRange->Range2Base = fchTemp.TmpBase;
    TempRange->Range2Limit = fchTemp.TmpLimit;
  }
}
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 Bolton Early-POST Initialization

The system BIOS needs to configure the Bolton 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 the Bolton design, there can be two possible ROM sources: PCI ROM and LPC ROM. Two pin straps (UseLpcRom, FWHDisable) decide where the ROM is (see the Bolton databooks). Upon system power on, the Bolton 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*

Control Bit	Description	256K ROM Setting (Default)	512K ROM Setting
PM_Reg: 0x04 [12]	When set to 1, the address between FFF80000h to FFFDFFFFh will be directed to the PCI ROM interface.	0	1
PM_Reg: 0x04 [13]	When set to 1, the address between 0E0000h to 0EFFFFh will be directed to the PCI ROM interface.	0	1

3.1.2 *LPC ROM*

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

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

3.1.3 *LPC ROM Read/Write Protect*

The Bolton 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. The address length can be from 1K to 512K in increments of 1K.

Register 50h, 54h, 58h, 5ch of Device 14h, Function 3

Field Name	Bits	Description
Base Address	31:11	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.
Length	10:2	These 9 bits (0-511) define the length from 1K to 512K in increments of 1K.
Read Protect	1	When set, the memory range defined by this register is read-protected. Reading any location in the range returns FFh.
Write Protect	0	When set, the memory range defined by this register is write-protected. Writing to the range has no effect.

Example:

Protect 32K LPC ROM starting with base address FFF80000.

Base address bits 31:11 1111 1111 1111 1000 0000 0 b

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 = FFF8007F h

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 Bolton. Refer to the *AMD Bolton 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 Bolton 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 function (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 (registers 0Eh to FFh) should be 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).

RtcControl - RW – 8 bits - [PM_Reg: 56h]			
Field Name	**Bits**	**Default**	**Description**
RTCProtect	0	0h	When set, RTC RAM index 38h:3Fh will be locked from read/write.
RTCProtect	1	0h	When set, RTC RAM index F0h:FFh will be locked from read/write.
RTCProtect	2	0h	When set, RTC RAM index E0h:EFh will be locked from read/write.
RTCProtect	3	0h	When set, RTC RAM index D0h:DFh will be locked from read/write.
RTCProtect	4	0h	When set, RTC RAM index C0h:CFh will be locked from read/write.

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 Bolton 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 Bolton supports serial IRQ, which allows one single signal to report multiple interrupt requests. The Bolton 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.

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

Note**:** BIOS should enter the continuous mode first when enabling the serial IRQ protocol, so that the Bolton can generate the start frame.

3.5 SubSystemID and SubSystem Vendor ID

SubSytem ID and SubSystem Vendor ID can be programmed in various functions of Bolton register 2Ch. These registers are write-once registers. For example, to program a SubSytem 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 manner in which the system will restart following a power-fail/ power-restore cycle depends on the setting of PMIO register 5Bh [1:0].

PMIO Register 5Bh bits[1:0]	Description
00b	Reserved.
01b	The system will always restart after the power is restored.
10b	The system will remain off until the power button is pressed.
11b	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.

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. The BIOS programmer should always read the PMIO register 5Bh, modify bit[6] and bits[3:2] as required, and write back the PMIO register 5Bh. This is required for every boot cycle and after any RSMRST# or SYS_RST# 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 Bolton uses one pair of I/O ports for PCI IRQ routing. The ports are at C00h/C01h.

Address	Register Name	Description
C00h	PCI_Intr_Index	PCI interrupt index. Selects which PCI interrupt to map 0h: INTA# 1h: INTB# 2h: INTC# 3h: INTD# 4h: INTE# 5h: INTF# 6h: INTG# 7h: INTH# 8h: Misc 9h: Misc0 Ah: Misc1 Bh: Misc2 Ch: INTA from serial irq Dh: INTB from serial irq Eh: INTC from serial irq Fh: INTD from serial irq 10h: SCI 11h: SMBUS0 12h: ASF 13h: HD audio 14h: SD 15h: GEC 16h: PerMon 20h: IMC INT0 21h: IMC INT1 22h: IMC INT2 23h: IMC INT3 24h: IMC INT4 25h: IMC INT5 30h: Dev18 (USB) IntA# 31h: Dev18 (USB) IntB# 32h: Dev19 (USB) IntA# 33h: Dev19 (USB) IntB# 34h: Dev22 (USB) IntA#/xHCI0 35h: Dev22 (USB) IntB#/xHCI1 36h: Dev20 (USB) IntC# 40h: IDE pci interrupt 50h: GPPInt0 51h: GPPInt1 52h: GPPInt2 53h: GPPInt3
C01h	PCI_Intr_Data	0 ~ 15 : IRQ0 to IRQ15 IRQ0, 2, 8, 13 are reserved

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).

```
mov     dx,0C00h        ; To write to IO port C00h
mov     al,01h          ; 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 Bolton, AC'97 and USB require a PCI IRQ. Internally, they are routed to different PCI INT#s.

Device	Reg3Dh of PCI Device	PCI INT#	Description
Bus 0:Device 14h:Function 1	02	INTB#	IDE Controller*
Bus 0:Device 14h: Function 2	01	INTA#	High Definition Audio
Bus 0:Device 14h: Function 5	03	INTC#	USB #4 OHCI Controller
Bus 0:Device 12h:Function 0	01	INTC#	USB #1 OHCI Controller #0
Bus 0:Device 12h:Function 2	02	INTB#	USB #1 EHCI Controller
Bus 0:Device 13h: Function 0	01	INTC#	USB #2 OHCI Controller #0
Bus 0:Device 13h: Function 2	02	INTB#	USB #2 EHCI Controller
Bus 0:Device 10h: Function 0	01	INTC#	xHCI Controller 0
Bus 0:Device 10h: Function 1	02	INTC#	xHCI Controller 1
Bus 0:Device 11h:Function 0	01	INTD#	SATA Controller #2
* This is implemented in the current CIMx module reference code.			

4.4 PCI IRQ Routing for APIC Mode

PCI IRQ	APIC Assignment
INTA#	16
INTB#	17
INTC#	18
INTD#	19
INTE#	20
INTF#	21
INTG#	22
INTH#	23

5 SMBus Programming

The Bolton 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:

SMBus Frequency = (Primary Alink Clock)/(Count in index 0Eh * 4)

The power-up default value in register 0Eh is A0h, therefore the default frequency is (66MHz)/(160 * 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: (66MHz)/(255*4) = 64.7 KHz.

5.2 SMBus Host Controller Programming

Note: Before accessing SMBus, the program needs to check if SMBus Auto Polling is enabled. If yes, it should be paused/stopped before any SMBus operation. After the SMBus operation, SMBus Auto Polling status should be restored.

Step	Descriptions	Register in SMBus I/O Space	Comments
1	Wait until SMBus is idle.	Reg00h[0]	0 – Idle 1 – Busy
2	Clear SMBus status.	Reg00h[4:1]	Write all 1's to clear
3	Set SMBus command.	Reg03h	The command will go to SMBus device.
4	Set SMBus device address with read/write protocol	Reg04h	Bits[7:1] – address Bit0 – 1 for read, 0 for write
5	Select SMBus protocol	Reg02h[4:2]	
6	Do a read from Reg02 to reset the counter if it's going to be a block read/write operation	Reg02h	
7	Set low byte when write command	Reg05h	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
8	Set high byte when write command	Reg06h	Word command – It is the high byte data Others – Don't care
9	Write the data when block write	Reg07h	Block write – write data one by one to it Others – Don't care
10	Start SMBus command execution	Reg02h[6]	Write 1 to start the command
11	Wait for host not busy	Reg00h[0]	

12	Check status to see if there is any error	Reg00h[4:2]	With 1 in the bit, there is error
13	Read data	Reg05h	Byte command – It is the read data Word command – It is the low byte data Block command – It is block count Others – Don't care
14	Read data	Reg06h	Word command – It is the high byte data Others – Don't care
15	Read the data when block write	Reg07h	Block read – read data one by one. Others – Don't care

The following flow chart illustrates the steps in programming the SMBus host controller.

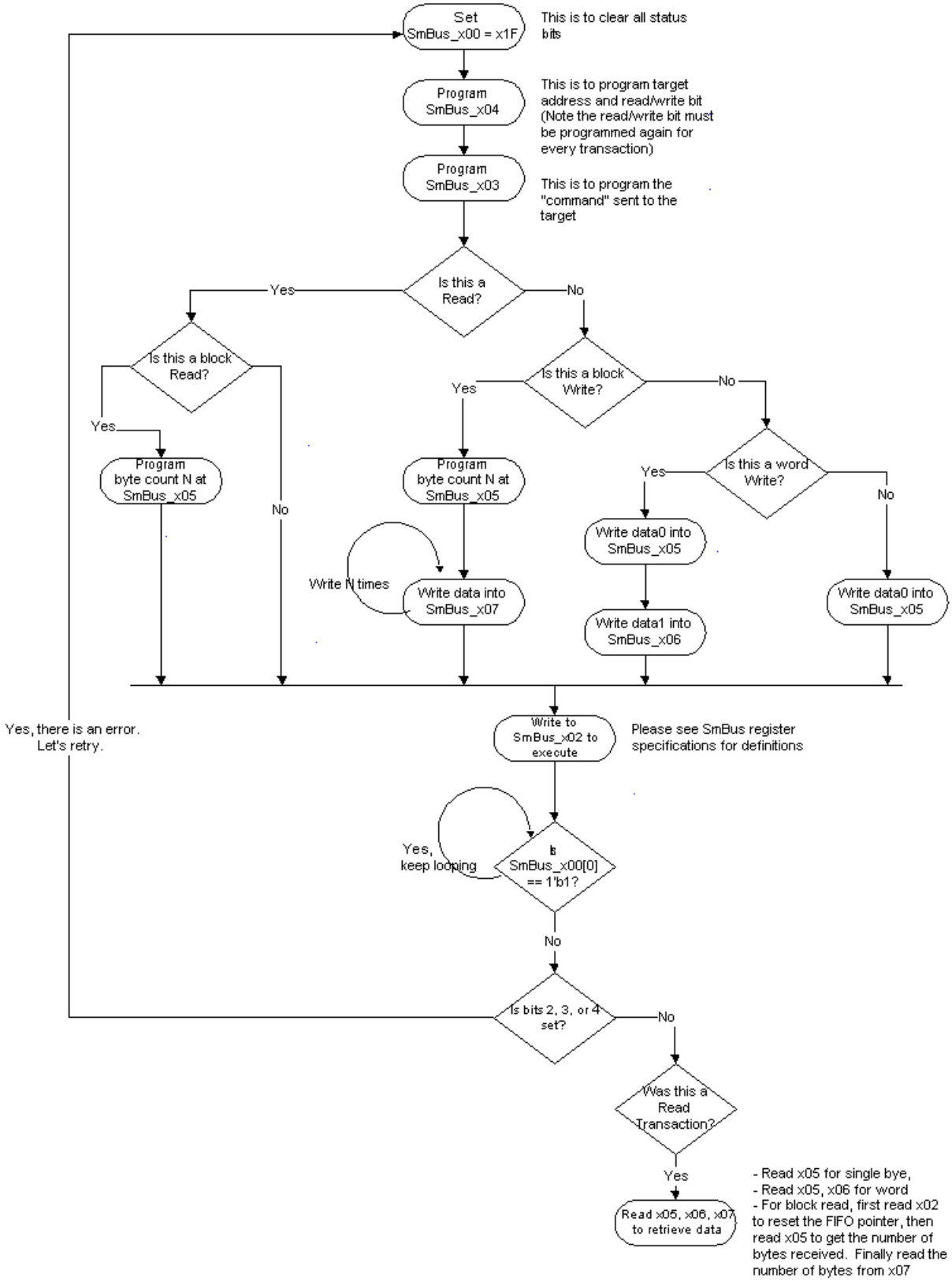

Figure 2: Programming the SMBus Host Controller

5.3 Sample ASL Code Required to Support Loading of SMBUS Device Drivers

The following ASL code is required to support OS in loading the SMBUS 0 and SMBUS 1 device drivers. This ASL code is also required to support Synaptics Touch Pad device supporting the Synaptics InterTouch over SMBus.

Sample ASL code for Smbus0, ASF.
SMB0001 is also used by OS to load device driver for ASF HC.
SMB0000 is also used by OS to load device driver for SMBus0 HC.

```
Device (SMB1)
{
   Name(_HID, "SMB0001")
   Name(_CRS, ResourceTemplate() {
     Io(Decode16,
        0x0B20,
        0x0B20,
        0x20,
        0x20)
     IRQ(Level, ActiveLow, Shared, ) {7}
   }

}


Device (SMB0)
{
   Name(_HID, "SMB0000")
   Name(_CRS, ResourceTemplate() {
     Io(Decode16,
        0x0B00,
        0x0B00,
        0x10,
        0x10)
     IRQ(Level, ActiveLow, Shared, ) {11}
   }
```

Note: In addition to the above mapping in ASL, the Platform BIOS should enable the ASF controller via CIM-X call-back function and assign I/O and Interrupt resources to ASF.

6 Serial ATA (SATA)

The Bolton SATA controller compared to the previous generation FCHs differs in two areas:

1. Supports two additional ports providing a total of 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 the 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 Table 1 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.

Table 1 SATA Port Assignment in Combined IDE Mode

Port Number	Primary , Secondary , Master / Slave Assignment	SATA Drive Controlled by
Port 0	Primary master	SATA controller
Port 1	Secondary master	SATA controller
Port 2	Primary slave	SATA controller
Port 3	Secondary slave	SATA controller
Port 4	Primary (Secondary) master	PATA controller
Port 5	Primary (Secondary) slave	PATA controller

The following figure shows the various combined mode configurations.

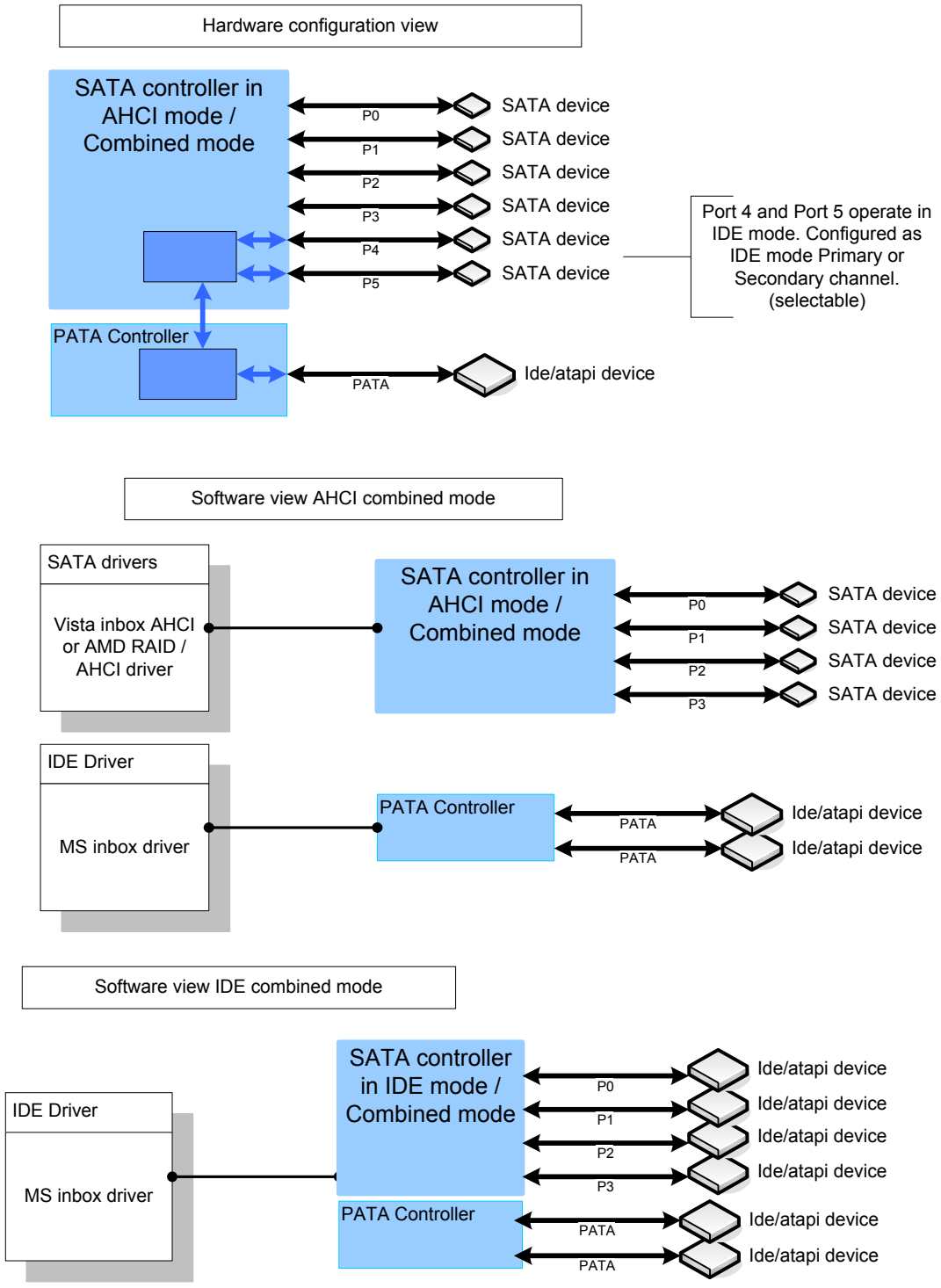

Figure 3: Combined Mode Configurations

Note: In the IDE combined mode, the MS inbox driver will control all PATA drives showing all devices under two physical IDE controllers.

6.1 Device IDs and Drivers

The Bolton 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 various 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.

SATA Mode	OS Driver Support				Device ID	Comments
	Windows XP*	Vista	Windows 7	Windows 8		
IDE	MS inbox IDE	MS inbox IDE	MS Inbox IDE	MS Inbox IDE	7800h	
AHCI	Dot Hill OTB	MS inbox AHCI	MS inbox AHCI	MS inbox AHCI	7801h	AMD Win 7 OTB AHCI drivers can be loaded for both device IDs
	Dot Hill OTB**	MS inbox AHCI	AMD inbox AHCI	AMD inbox AHCI	7804h	
RAID	Dot Hill OTB	Dot Hill OTB RAID 0/1/10	Dot Hill inbox RAID 0/1/10	Dot Hill inbox RAID 0/1/10	7802h	Dot Hill OTB drivers can be loaded for RAID 0/1/10.
	Dot Hill OTB	Dot Hill OTB RAID 5	Dot Hill OTB RAID 5	Dot Hill OTB 0/1/10	7803h	Dot Hill OTB drivers can be loaded for RAID 5.

*Windows XP driver supported only with FS1 and FM1 Family 12 APU platforms.
* *Device ID 7804h to be supported in post Q3 2011 releases of Windows XP drivers.

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 and 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.

When the Combined Mode is ON, there is no need to program PIO timing parameters on the IDE (PATA) controller.

7 USB

Note:
- Several register descriptions here are copied from the Register Reference Guide (RRG) for illustrative purposes. Please refer to the RRG as the definitive register reference.

7.1 Enable

When the system powers up from G3->S5->S0, the USB3.0/xHCI controllers are disabled and held in reset. An EHCI/OHCI controller pair is enabled to control USB2.0 ports 10-13, which are used by the USB3.0/xHCI controller when it is enabled. This OHCI/EHCI controller pair must first be disabled and the ports routed to the xHCI controller. These controls are in an ACPI PMIO register.

- Set UsbEnable.USB3 OHCI Enable = 0
- Set UsbEnable.USB3 EHCI Enable = 0
- Set USBEnable.Port Routing Select = 1

UsbEnable – RW – 8 bits – [PM_Reg:Efh]			
Field Name	**Bits**	**Default**	Description
USB1 OHCI Enable	0	1b	Enable bit for USB1 OHCI controller (device-18, function-0)
USB1 EHCI Enable	1	1b	Enable bit for USB1 EHCI controller (device-18, function-2)
USB2 OHCI Enable	2	1b	Enable bit for USB2 OHCI controller (device-19, function-0)
USB2 EHCI Enable	3	1b	Enable bit for USB2 EHCI controller (device-19, function-2)
USB3 OHCI Enable	4	1b	Enable bit for USB3 OHCI controller (device-22, function-0)
USB3 EHCI Enable	5	1b	Enable bit for USB3 EHCI controller (device-22, function-2)
USB4 OHCI Enable	6	1b	Enable bit for USB4 OHCI controller (device-20, function-5)
port routing select	7	0b	Port10-port13 routing select bit 0: port10-port13 are routed to USB3 1: port10-port13 are routed to XHC
Note: xHCI Enable is in xHC ACPI MMIO space at ACPI_USB3_REG: 00h			

The xHCI enables and the resets are both controlled by register fields in the ACPI MMIO space allocated for USB3.0 registers. The base address for these registers is obtained from the following ACPI register:

AcpiMmioEn - RW – 8/16/32 bits - [PM_Reg: 24h]			
Field Name	**Bits**	**Default**	**Description**
AcpiMMioDecodeEn	0	0b	Set to1 to enable AcpiMMio space.
AcpiMMIoSel	1	0b	Set AcpiMMio registers to be memory-mapped or IO-mapped space. 0: Memory-mapped space 1: I/O-mapped space

AcpiMmioEn - RW – 8/16/32 bits - [PM_Reg: 24h]			
Field Name	**Bits**	**Default**	**Description**
AcpiMMioAddr	31:13	FED8_00h	SBResourceMMIO_Base register Offset 000:0FF SM PCI configuration registers 100:1FF GPIO 200: 2FF SMI 300: 3FF PMIO 400: 4FF PMIO2 500: 5FF BIOS_RAM 600: 6FF CMOS_RAM 700: 7FF CMOS 800: 8FF ACPI 900: 9FF ASF registers A00: AFF Smbus registers B00: BFF WatchDog registers C00: CFF HPET (new) D00: DFF IoMux (new) E00: EFF Misc (new) 1000:10FF Serila Debug bus 1400:14FF Gfx Dac 1800:18FF CEC **1C00:1CFF Usb3**

- Set USB3.0_ACPI_MMIO_REG00.xHCI0_Enable = 1
- Set USB3.0_ACPI_MMIO_REG00.xHCI1_Enable = 1

USB3.0_ACPI_MMIO_REG00 - RW - 32 bits - [ACPI_USB3.0_REG: 00h]			
Field Name	**Bits**	**Default**	**Description**
XHCI0_Enable	0	0b	XHCI0 Enable 1: Enable XHCI0.
XHCI1_Enable	1	0b	XHCI1 Enable 1: Enable XHCI1

7.2 Firmware Load

Each xHCI controller contains a micro-controller. The micro-controller executes firmware from an on-chip rom as well as an instruction ram. The firmware to be loaded to the instruction RAM is stored in the platform BIOS ROM. Before reset is released to the xHCI controller, the firmware must be loaded on-chip to the instruction RAM and certain registers. This loading process is done via a series of steps described in the following sections.

7.2.1 *FW_Load_Mode*

- Set USB3.0_ACPI_MMIO_REG00.FW_Load_Mode = 1

USB3.0_ACPI_MMIO_REG00 - RW - 32 bits - [ACPI_USB3.0_REG: 00h]			
Field Name	**Bits**	**Default**	**Description**
FW_Load_Mode	28	0b	Firmware Load Mode 0: xHCI embedded CPU will execute bootstrap routine to load firmware to instruction ram. 1: xHCI embedded CPU will skip instruction ram loading routine (instruction ram must be preloaded)

7.2.2 *SPI Data Block*

Certain special sections of the firmware are loaded to a set of 16 SPI Data Block Registers.

SPI Data Block N – R/W- 32 bits – [ACPI_USB3.0_REG: C0h/C4h/C8h/CCh/D0h/D4h/D8h/DCh/E0h/E4h/E8h/ECh/F0h/F4h/F8h/FCh]			
Field Name	**Bits**	**Default**	**Description**
XHC_SPI_Data_Block	31:0	0h	SPI Data Block

The special sections of firmware are:
- Header
- BCD: Boot Configuration Data
- ACD: Application Configuration Data

Firmware is stored in the BIOS rom at XHC_FIRMWARE_START_ADDR. It is organized as shown in the table below:

Byte Addr (hex)	Content	Comment
0	Startup Code	Header
1		
2	BCD Addr	
3		
4	BCD Size	
5		
6	FWID Addr	
7		

8	FWID Size	
9		
A	ACD Addr	
B		
C	ACD Size	
D		
BCD_ADDR	BCD	Boot Configuration Data
BCD_ADDR+1		
…		
BCD_ADDR+BCD_SIZE		
ACD_ADDR	ACD	Application Configuration Data
ACD_ADDR+1		
…		
ACD_ADDR+ACD_SIZE		
FWID_ADDR	FW Version	Firmware Version
FWID_ADDR+1		
FWID_ADDR+2	FW Image Data	Firmware Image
…		
FWID_ADDR+FWID_SIZE		

The micro-controller fetches data from the SPI Data Block using a set of four base-address-registers (BARs) such as the one below:

SPI BAR0 – R/W- 32 bits – [ACPI_USB3.0_REG: A0h]			
Field Name	**Bits**	**Default**	**Description**
XHC_SPI_data_type0_addr	15:0	0h	Type0 data rom address.
XHC_SPI_data_type0_size	31:16	0h	Type0 data size.

SPI Valid Base – R/W- 32 bits – [ACPI_USB3.0_REG: B0h]			
Field Name	**Bits**	**Default**	**Description**
SPI_BAR0_Vld	0	0h	SPI BAR0 Valid. 1: SPI_BAR0 register is valid
SPI_Base0	7:2	0h	SPI Base0 Offset in SPI data block of type0 data.

SPI_BAR0.XHC_SPI_data_type0_addr and SPI_BAR0.XHC_SPI_data_type0_size define the address and size of a piece of firmware (e.g. FWID, BCD, or ACD). SPI_Valid_Base.SPI_BAR0_Vld indicates if SPI_BAR0 is valid. SPI_Valid_Base.SPI_Base0 is the offset into the SPI Data Block where this piece of firmware is stored.

7.2.2.1 Header

The header contains the following two byte fields:

Byte 00: Start Up Code
Byte 02: BCD Addr

Byte 04: BCD Size
Byte 06: FWID Addr
Byte 08: FWID Size
Byte 10: ACD Addr
Byte 12: ACD Size

- Set SPI_BAR0.XHC_SPI_data_type0_addr = 0000h
- Set SPI_BAR0.XHC_SPI_data_type0_size = 14d
- Set SPI_Valid_Base.SPI_BAR0_Vld = 1
- Set SPI_Valid_Base.SPI_Base0 = 0x00
- Load Header to SPI Data Block:

```
for (i=0; i<14; i++) {
  byte_addr = XHC_FIRMWARE_START_ADDR + i;
  byte_data = BIOS_ROM[byte_addr];
  byte_shift = i%4;
  data_block = i/4;
  SPI_DATA_Block_<data_block>[(8*byte_shift+7) : 8*byte_shift] = byte_data;
}
```

7.2.2.2 BCD

- Read BCD Addr (bytes 2, 3) from the start address of the xHCI firmware in the BIOS rom and write it to SPI_Bar1.XHC_SPI_type1_addr.
- Read BCD Size (bytes 4, 5) from the start address of the xHCI firmware in the BIOS rom and write it to SPI_Bar1.XHC_SPI_type1_size.
- Set SPI_Valid_Base.SPI_BAR1_Vld = 1
- Set SPI_Valid_Base.SPI_Base1 = 0x00 + 14d = 0x0E
- Load BCD to SPI Data Block:

```
for (i=0; i<BCD_SIZE; i++) {
  byte_addr = XHC_FIRMWARE_START_ADDR + BCD_ADDR + i;
  byte_data = BIOS_ROM[byte_addr];
  byte_shift = (SPI_Base1 + i)%4;
  data_block = (SPI_Base1 + i)/4;
  SPI_DATA_Block_<data_block>[(8*byte_shift+7) : 8*byte_shift] = byte_data;
}
```

7.2.2.3 ACD

- Read ACD Addr (bytes 10, 11) from the start address of the xHCI firmware in the BIOS rom and write it to SPI_Bar2.XHC_SPI_type2_addr.
- Read ACD Addr (bytes 10, 11) from the start address of the xHCI firmware in the BIOS rom and write it to SPI_Bar2.XHC_SPI_type2_size.
- Set SPI_Valid_Base.SPI_BAR2_Vld = 1
- Set SPI_Valid_Base.SPI_Base2 = 0x0E + BCD SIZE
- Load ACD to SPI Data Block:

```
for (i=0; i<ACD_SIZE; i++) {
  byte_addr = XHC_FIRMWARE_START_ADDR + ACD_ADDR + i;
  byte_data = BIOS_ROM[byte_addr];
  byte_shift = (SPI_Base2 + i)%4;
  data_block =(SPI_Base2 + i)/4;
```

SPI_DATA_Block_<data_block>[(8*byte_shift+7) : 8*byte_shift] = byte_data;
}

7.2.2.4 FWID

- Read FW_Version (two bytes) at XHC_FIRMWARE_START_ADDR + FWID_ADDR.
- Write FW_Version to SPI_Misc.XHC_SPI_FW_ID.

SPI Misc – R/W- 32 bits – [ACPI_USB3.0_REG: B4h]			
Field Name	**Bits**	**Default**	**Description**
XHC_SPI_FW_ID	15:0	0h	SPI Firmware ID Firmware Version ID

7.2.3 *Instruction RAM Load*

The USB3.0/xHCI controller contains an engine to read the remainder of the firmware from BIOS ram and load it to the instruction ram. This engine is programmed by the following steps.

- Read FWID_ADDR (two bytes) at XHC_FIRMWARE_START_ADDR + 6.
- Write FWID_ADDR + 2 to USB3.0_ACPI_MMIO_REG04.xHCI_FW_Address_Offset
- Read FWID_SIZE (two bytes) at XHC_FIRMWARE_START_ADDR + 8
- Write FWID_SIZE to USB3.0_ACPI_MMIO_REG04.xHCI_FW_Size

USB3.0_ACPI_MMIO_REG04 - RW - 32 bits - [ACPI_USB3.0_REG: 04h]			
Field Name	**Bits**	**Default**	**Description**
XHCI_FW_Address_Offset	15:0	0h	XHCI Firmware Address Address of the first byte of application firmware in the external ROM.
XHCI_FW_Size	31:16	0h	XHCI Firmware Size Size of the application firmware in the external ROM.

- Set USB3.0_ACPI_MMIO_REG08.xHCI_Inst_Ram_Start_Addr = 0

USB3.0_ACPI_MMIO_REG08 - RW - 32 bits - [ACPI_USB3.0_REG: 08h]			
Field Name	**Bits**	**Default**	**Description**
XHCI_Inst_Ram_Start_Addr	15:0	0h	XHCI Instruction RAM Start Address Instruction RAM address where first byte of application firmware is to be loaded.
reserved	31:16	0h	

- Set USB3.0_ACPI_MMIO_REG00.Inst_Ram_Preload_Start = 1
- Wait for Inst_Ram_Preload_Complete == 1
  while (Inst_Ram_Preload_Complete == 0) {
  ;

}

- Set USB3.0_ACPI_MMIO_REG00.Inst_Ram_Preload_Start = 0

USB3.0_ACPI_MMIO_REG00 - RW - 32 bits - [ACPI_USB3.0_REG: 00h]			
Field Name	**Bits**	**Default**	**Description**
Inst_Ram_Preload_Start	29	0b	Instruction RAM Preload Start When set from 0->1, facilitator will load instruction ram. Upon completion of instruction ram load sequence (see Inst_Ram_Preload_Complete), Inst_Ram_Preload_Start should be set to 0.
Inst_Ram_Preload_Complete	30	0b	Instruction RAM Preload Complete (Read Only) Hardware will set this bit when it has completed instruction ram preload. Hardware will clear this bit when “Instr_Ram_Preload_Start” is cleared by software.

7.3 Release Reset

- Set USB3.0_ACPI_MMIO_REG00.U3P_PLL_Reset = 0
- Wait for PLL to lock.
  while (USB3.0_ACPI_MMIO_REG00.U3P_Lock == 0) {
  ;
  }
- Set USB3.0_ACPI_MMIO_REG00.U3P_PHY_Reset = 0
- Set USB3.0_ACPI_MMIO_REG00.U3_Core_Reset = 0

USB3.0_ACPI_MMIO_REG00 - RW - 32 bits - [ACPI_USB3.0_REG: 00h]			
Field Name	**Bits**	**Default**	**Description**
U3P_Lock	7	0b	USB3.0 PHY PLL Lock (Read Only) 1: USB3.0 PHY PLL Locked.
U3P_PLL_Reset	8	1b	USB3.0 PHY PLL Reset 1: Reset USB3.0 PHY PLL and SSPHYIF power control logic in LFPS clock domain.
U3P_PHY_Reset	9	1b	USB3.0 PHY Reset 1: Reset SSPHYIF power control logic in 125MHz clock domain.
U3_Core_Reset	10	1b	USB3.0 Core Reset 1: Reset the USB3.0/XHCI (XHC0 + XHC1) core logic.

7.4 _UPC and _PLD Support for ACPI 3.0

It is required by WHQL test to use ACPI 3.0 interfaces _UPC (USB port capabilities) and _PLD (physical location description) for accurately defining the USB port configuration.

USB Port Status	Usage in System	_UPC.PortIsConnectable Byte	_PLD.UserVisible bit (bit 64)
Port is visible and the user can freely connect and disconnect devices.	External port	Set (0xFF)	Set (1)
Port is not user visible and user cannot freely connect and disconnect devices.	Connect to onboard devices	Set (0xFF)	Cleared
Port is physically implemented by the USB host controller, but is not used.	Not used	Cleared (0x00)	Cleared

The following is a sample ASL code:

```
Device(XHC0)
{
   Name(_ADR, 0x00100000)

   Device (RHUB)
   {
    Name (_ADR, 0x00000000)   // Root HUB always has a value of 0

    Device (PRT1)
    {
     Name (_ADR, 0x00000001) // Port 1 under xHCI0 = USB3 Port 0

     Name(_UPC, Package() {
       0xFF,                 // Port is connectable
       0x03,                 // Connector type - USB 3 Standard-A
       0x00000000,           // Reserved 0 - must be zero
       0x00000000})          // Reserved 1 - must be zero

     Name(_PLD, Package() {
       Buffer (0x14) {
       0x82, 0x00, 0x00, 0x00,     // Revision 2, Ignore color
       0x00, 0x00, 0x00, 0x00,
       0x31, 0x1C, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00,
       0xFF, 0xFF, 0xFF, 0xFF}
     })
    }

    Device (PRT2)
    {
     Name (_ADR, 0x00000002) // Port 2 under xHCI0 = USB3 Port 1

     Name(_UPC, Package() {
       0xFF,                 // Port is connectable
       0x03,                 // Connector type - USB 3 Standard-A
       0x00000000,           // Reserved 0 - must be zero
       0x00000000})          // Reserved 1 - must be zero
```

```
        Name(_PLD, Package() {
          Buffer (0x14) {
          0x82, 0x00, 0x00, 0x00,     // Revision 2, Ignore color
          0x00, 0x00, 0x00, 0x00,
          0x30, 0x1C, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00,
          0xFF, 0xFF, 0xFF, 0xFF}
        })
      }
    } // Device(RHUB)
  } // Device(XHC0)
```

8 APIC Programming

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

8.1 Northbridge APIC Enable

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

- Enable Local APIC in CPU. (Set bit[11] in APIC_BASE MSR(001B) register.)
- 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.

8.2 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.

8.3 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.

8.4 APIC IRQ Assignment

Bolton 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.

8.5 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
}
```

9 UMI Bridge

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

31:30	29:17	16:2	1:0
RegSpace[1:0]	Reserved	Register address[16:2]	Reserved

AB_INDX [31:0]

31:0
Data[31:0]

AB_DATA [31:0]

RegSpace[1:0]	
00b	AXINDC Index/Data Registers. (AX_INDXC)
01b	AXINPD Index/Data Registers (AX_INDXP)
10b	Alink Express Configuration (AXCFG)
11b	Alink Bridge Configuration (ABCFG)

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_INDX 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_INDX in the I/O space is defined by the abRegBaseAddr register located at Device 14h, function 0, register 0F0h. The AB_DATA register address is offset 4h from the AB_INDX address. The address of the AB_INDX must be 8 byte aligned.

31:3	2:0
BaseAddr[31:3]	Rsv

abRegBAR [31:0] at Bus 0, Device 14h, Function 0, Register 0F0h

AXCFG and ABCFG registers are accessed indirectly through AB_INDX/AB_DATA. To read or write a particular register through AB_INDX/AB_DATA, the register address and the register space identifier is first written to AB_INDX. 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
AX_INDXC	30h
AX_DATAC	34h
AX_INDXP	38h
AX_DATAP	3Ch

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

1. Out 30h to AB_INDX. This will prepare to write register from INDXC
2. Out 21h to AB_DATA. This will set register 21h of INDXC
3. Out 34h to AB_INDX. 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 n steps 1 and 2 above.

9.1 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 device 14h, function 0, register 0F0h. This is an I/O address and needs to be set only once after power-up. The I/O address must be on a 8-byte boundary (i.e., 3 LS bits must be zeroes).

   Example: To set C80h as an UMI Bridge register access address:

   ```
   mov     dx,0CF8h            ; To access device 14h, function 0
   mov     eax,8000A0F0h       ;
   out     dx,eax
   mov     dx,0CFCh
   mov     eax,00000C80h       ; UMI Bridge register access address
   out     dx,eax
   ```

   Note: Although the 32-bit I/O address is set for the UMI Bridge (e.g., 00000C80h), the bridge may be accessed by a 16-bit address (i.e., 0C80h). The MS word is set to 00 by default (see the example below).

2. Write the register address in the AB_INDX.

   Example: To write to the UMI Bridge configuration register space at 90h:

   ```
   mov     dx,0c80h            ; I/O address index assigned to A-Link
   mov     eax, 0C0000090h     ; Bits[31:30] = 11 for UMI Bridge register
                               ; space
   out     dx,eax              ; Register index is set
   mov     dx,0c84h            ; 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.

9.2 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 beginning of section 9 for a description of the AB_INDX 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_INDX. The write data's bit [31:30] should be 10 b(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., 32'b0000_0000_0000_0100 binary).

```
mov     dx, _0C80h              ; ALINK_ACCESS_INDEX
in  eax, dx
and eax, NOT (0C001FFFFh)
or  eax, 080000004h
out dx, eax
mov     dx,  0C84h              ; ALINK_ACCESS_DATA
mov     eax, 04h
out dx, eax

        ;Write   AB_INDX   0x30
        ;Write   AB_DATA  0x21
        ;Write    AB_INDX  0x34
        ;Write   AB_DATA  0x00

mov     dx,  0C80h              ; ALINK_ACCESS_INDEX
mov     eax, 30h
out dx, eax
mov     dx, 0C84h               ; ALINK_ACCESS_DATA
mov     eax, 21h
out dx, eax
mov     dx,0C80h                ; ALINK_ACCESS_INDEX
mov     eax, 34h
out dx, eax
```

```
mov     dx, 0C84h           ; ALINK_ACCESS_DATA
mov     eax, 00h
out dx, eax
```

9.3 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.
```