RAID Option ROM

Product Implementation Guide

Version 1.8
Date: 08/19/2009

Revision History

Version	Date	Author	Notes
1.8	2009/08/19	Adam Hsu	(1)To modify the description about AMD RAID / AHCI Common Command Interface function in Chapter 5. (2)To add "Interface Entry Point Format" V 2.0 description in Chapter 5.
1.7	2009/07/14	Adam Hsu	(1)To add description about AMD RAID / AHCI Common Command Interface function. ->Chapter 5 (2)Modify the description OEM Int13h / Ah=25h sub-function.
1.6	2009/01/07	Adam Hsu	(1)To add the OEM Int13h/Ah=25h sub-function description. -> Chapter 4, page 9,10,13
1.5	2008/02/13	Adam Hsu	(1)To change the search size from 32Kbytes to 1Kbytes during searching the MISC.BIN (2)To add the SBIOS PMM function using description during preparing the temp buffer for MISC.BIN by OpROM. (3)To add the summary memory using table during Init-time and Run-time.
1.4	2007/10/19	Adam Hsu	(1)How to reserve memory by Promise BIOS? (2)Where does the Promise BIOS search the misc.bin ? (3)Which memory does the Promise BIOS use to backup the conventional memory?
1.3	2007/10/12	Adam Hsu	(1) To change the implementation guide to 1.3 (2) To change the UI Part name to MISC.BIN and LoadTime Part name to SATA.BIN (3) To add the method and pictures to describe how to allocate the EBDA and search the misc.bin.
1.2	2007/10/05	Adam Hsu	Added load time part code will use DOS conventional area
1.1	2007/08/02	Grace Chang	Updated for SB650/SB700/SB750
1.0	2006/10/17	Sean Wang	Initial for SB600

Table of Contents

1. Scope

This guide assumes that the reader is familiar with the conventional INT 13h interface, the usage of the BIOS Device Parameter Table, Bootable CD-ROM Format Specification and the basic operation of mass storage devices. This guide describes in detail how to implement the option ROM BIOS and the INT13h functions.

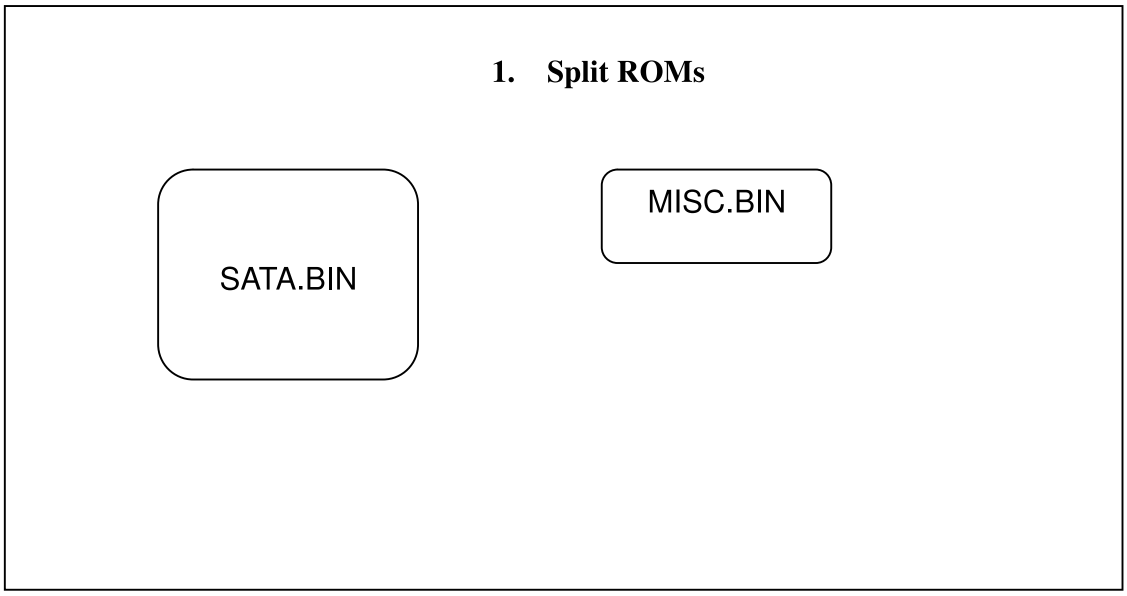
2. Binary Implementation

The binary file consists of two parts. One part is SATA.BIN which is for initialization and runtime use, another part is MISC.BIN which includes the user interface of the FastBuild (tm) Utility and other runtime code.

When the option ROM is called, the SATA.BIN will run first and search and copy the MISC.BIN to the conventional memory prepared by SBIOS PMM function or by the OpROM. After the initialization has been finished, it will prompt user to press “Ctrl+F” if user want to enter the configuration utility.

Besides, in order to reduce the Option ROM size in runtime stage, a part of MISC.BIN code will be put in EBDA area after load-time init stage.

The binary will be released in two Split ROMs.



Note:

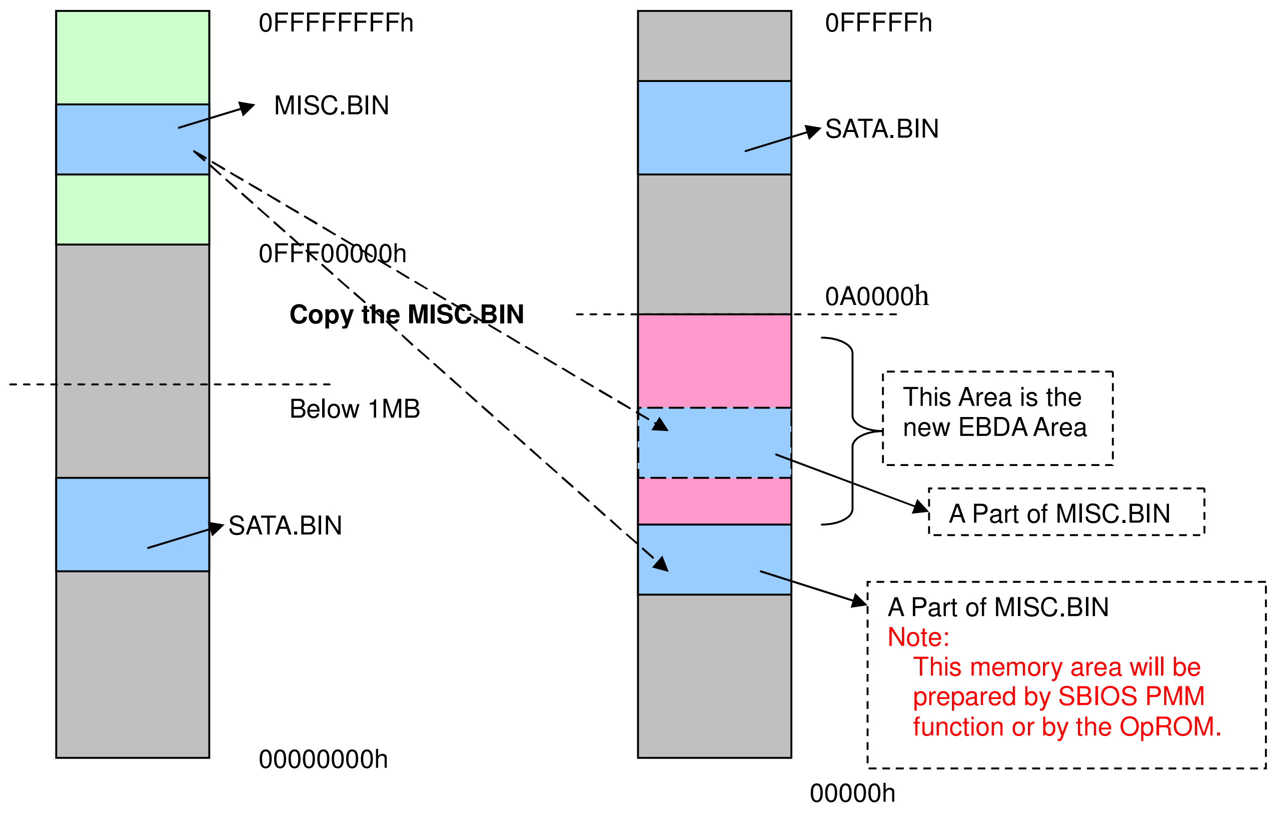
This dynamic UI loading mechanism requires at least the UI part being left uncompressed in the system BIOS. Otherwise the FastBuild (tm) Utility will unable to show UI properly.

Besides, the location of MISC.BIN is up to the system BIOS. For example, current AMD SB700/750 system bios will load MISC.BIN at around 0FFF00000h, SATA.BIN will search for "**MISC.SIG**" signature started from 0FFF00000h and ended at 0FFFFFFFFh by 1Kbytes.

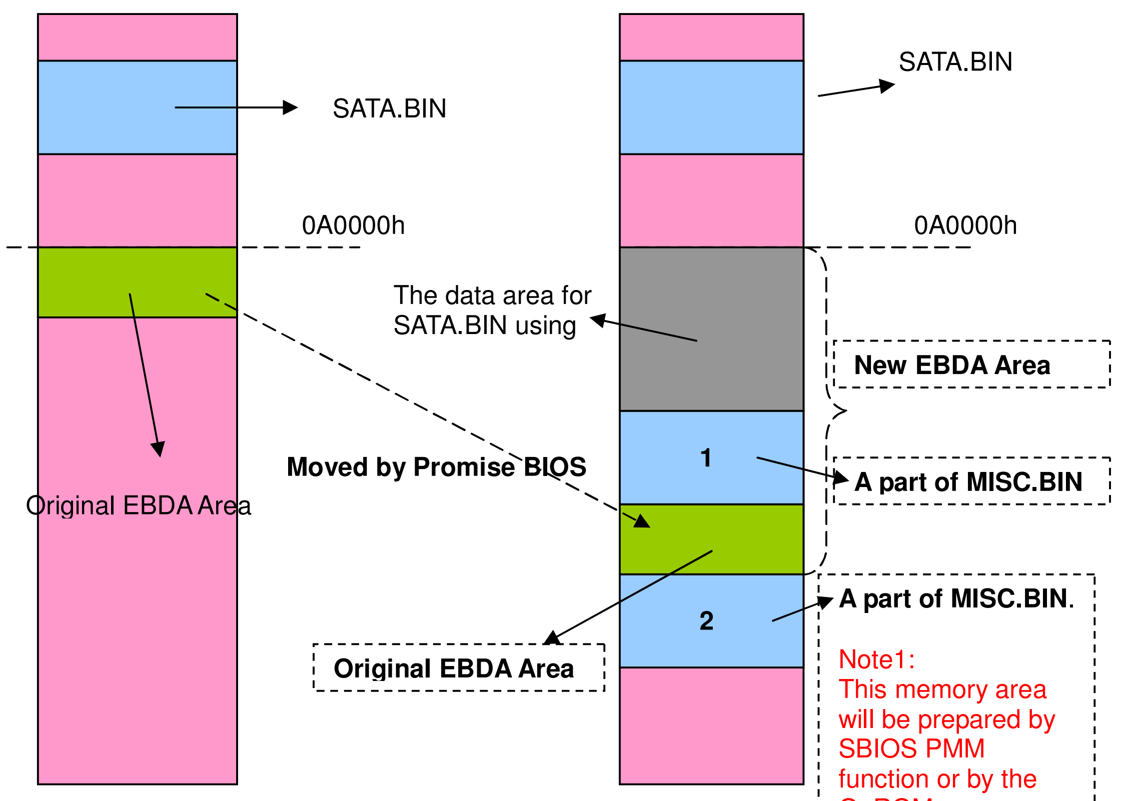
However, for current Promise BIOS design mechanism, the SATA.BIN will search the MSIC.BIN from address 0 ~ 7FFFFh first, if the MISC.BIN can't be found, the SATA.BIN will try to search again from 0FFF00000h to 0FFFFFFFFh.

If the MISC.BIN was found, the SATA.BIN will backup one conventional memory area and then copy the MISC.BIN into it. After Option ROM Init is finished, the SATA.BIN will restore the conventional memory area that was backup by SATA.BIN before.

The principle of backup the conventional memory is the Promise BIOS will copy the conventional memory to the high memory address that is 06400000h and this copied range is up to the part of MISC.BIN file size.



How to allocate the EBDA area by SATA.BIN



Note1:
This memory area will be prepared by SBIOS PMM function or by the OpROM.

Note2:
If the SBIOS PMM function fails, this area will be backup first before the SATA.BIN use. Then, it will be restored after SATA.BIN Init stage.

1. The gray, green and blue areas are used by Promise BIOS
2. The mark 2 of blue area will be copied to high memory 06400000h first before SATA.BIN copy the MISC.BIN.
3. This start address of this mark 2 of blue area is up to EBDA segment address.
   For example, to suppose the EBDA segment address is 00920000h and the size of mark 2 of blue area is 2000h, so the start address of mark 2 of blue area is the EBDA segment address subtract the 2000h.

FIG .1

The summary of memory range using by OpROM :

Init-Time EBDA Segment Address : **X** (Linear address)

Init-Time EBDA Size (Got from byte 0) : **Y**

Run-Time EBDA Segment Address : **W** (Linear address)

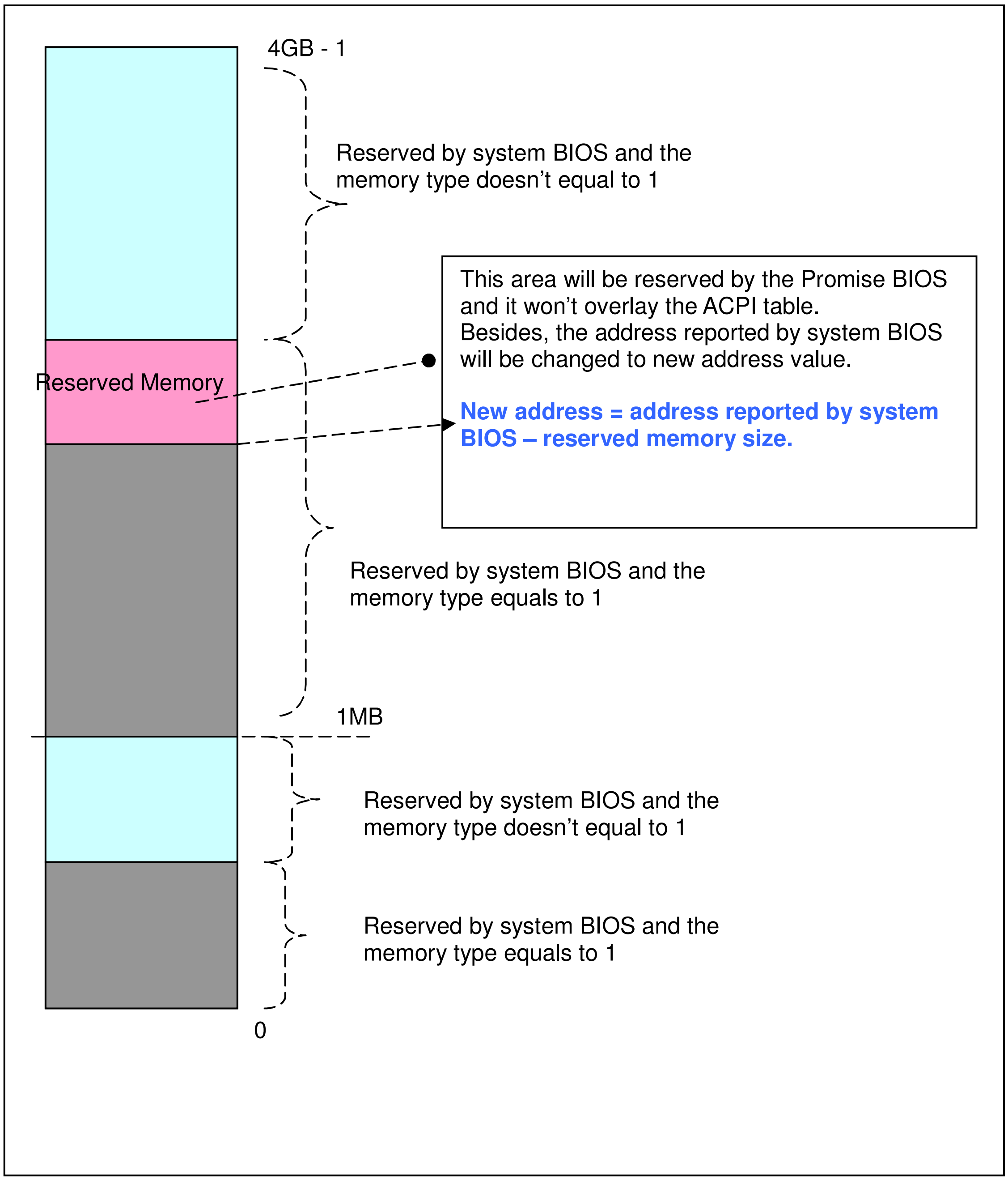
MISC.BIN (CAM Module) Size : **Z**

MISC.BIN (UI Module and other modules) Size : **P**

	PMM OK		PMM Fail	
	Init-Time Start Address	Run-Time Start Address	Init-Time Start Address	Run-Time Start Address
MISC.BIN (CAM Module)	**X + Y**	**W + Y**	**X + Y**	**W + Y**
MISC.BIN (UI Module and other module)	Depend on SBIOS PMM function	None	**X - P**	None
SATA Bin Data Area	**X + Y + Z**	**W + Y + Z**	**X + Y + Z**	**W + Y + Z**

Besides, in order to improve the driver performance, the Promise BIOS will use the reserved memory mechanism to do it.

The principle of this mechanism relates to INT15h / E820h. In the runtime stage, the Promise BIOS will hook the system BIOS INT15h / E820h function. If any user calls this function, the Promise BIOS will handle it and pass the address reported by system BIOS to used. The Promise BIOS just ONLY handle the OS memory area (memory type = 1).


4GB - 1
Reserved by system BIOS and the memory type doesn't equal to 1
This area will be reserved by the Promise BIOS and it won't overlay the ACPI table.
Besides, the address reported by system BIOS will be changed to new address value.
New address = address reported by system BIOS – reserved memory size.
Reserved Memory
Reserved by system BIOS and the memory type equals to 1
1MB
Reserved by system BIOS and the memory type doesn't equal to 1
Reserved by system BIOS and the memory type equals to 1
0

3. Array Detect

In RAID mode, the last logical cylinder of a hard drive is in the MDD area and will be protected from read/write by system BIOS or user. In NONE-RAID mode, system BIOS can read this flag to check whether the attached hard drive has been configured to RAID mode.

4. INT 13h Function Definitions

The option ROM will follow the *BIOS Boot Specification* V1.01 to implement BCV entries and the INT13h service routine which is complaint with *BIOS Enhanced Disk Drive Services 2.0* or *2.1* will be provided as Table 1.

Moreover, the Promise RAID option ROM (starts from 3.0.1540.50 version) also provides OEM Int13h function call except for standard Int13h.

Currently, the option ROM ONLY supports OEM Int13h / AH=25h to get each physical hard disk information on each port by ATA identify command.

Those *Reserved functions* will always return a success while any other function number beyond Table 1 will be regarded as an *Invalid function* and always returns with an error code 01h.

Function Number(AH)	**Function**
00h	Reset Disk Subsystem
01h	Get Status of Last Operation
02h	Read Sectors into Memory
03h	Write Sectors from Memory
04h	Verify Sectors
05h	*Reserved.*
08h	Get Drive Parameters
09h	*Reserved.*
0Ch	*Reserved.*
0Dh	*Reserved.*
10h	*Reserved.*
11h	*Reserved.*
14h	*Reserved.*
15h	Get Disk Type
25h	Get physical hard disk information by ATA identify command
41h	Check Extensions Present
42h	Extended Read
43h	Extended Write
44h	Extended Verify Sectors
47h	*Reserved.*
48h	Get Device Parameters
4Bh	Terminate Disk Emulation

Table 1

4.1. Reset Disk Subsystem

Entry:

AH - 00h
DL - BIOS device number

Exit:

carry clear - Reset successful
AH - 0

This function will always return success.

4.2. Get Status of Last Operation

Entry:

AH - 01h
DL - BIOS device number

Exit:

carry clear - Last command was successful
AH - 0
AL - Status of last INT 13h operation

This function shall be used to return the status of the last INT 13 command executed. This function will always return success.

4.3. Read Sectors into Memory

Entry:

AH - 02h
AL - Number of sectors to read, shall be greater than 0 and less than 128
CH - Low order 8 bits of the cylinder number
CL - Bits 0-5 specify the sector number, bits 6-7 are the high order 2 bits of the cylinder
DH - Head number
DL - BIOS device number
ES:BX - Pointer to destination buffer in memory

Exit:

carry clear - Read was successful
AH - 0
AL - Number of sectors read
ES:BX - Buffer filled with read data
carry set
AH - Error code
ES:BX - Pointer to buffer partially filled with read data

This function shall be used to read data from the device into host memory buffer pointed to by ES:BX.

4.4. Write Sectors from Memory

Entry:

AH - 03h
AL - Number of sectors to write, shall be greater than 0 and less than 128
CH - Low order 8 bits of the cylinder number
CL - Bits 0-5 specify the sector number, bits 6-7 are the high order 2 bits of the cylinder
DH - Head number.
DL - BIOS device number
ES:BX - Pointer to source buffer in memory

Exit:

carry clear - Read was successful
AH - 0
AL - Number of sectors written
carry set
AH - Error code

This function shall be used to transfer data from the host buffer pointed to by ES:BX to the device.

4.5. Verify Sectors

Entry:

AH - 04h
AL - Number of sectors to verify, shall be greater than 0 and less than 128
CH - Low order 8 bits of the cylinder number
CL - Bits 0-5 specify the sector number, bits 6-7 are the high order 2 bits of the cylinder
DH - Head number.
DL - BIOS device number

Exit:

carry clear - Read was successful
AH - 0
AL - Number of sectors verified
carry set
AH - Error code

This function shall be used to check the sectors in the specified range on the device for errors. No data is transferred between the host and device by this command.

4.6. Get Drive Parameters

Entry:

AH - 08h
DL - BIOS device number

Exit:

carry clear - Get Drive Parameters was successful
AH - 00h
BL - Vendor Specific
CH - Low order 8 bits of the maximum cylinder number
CL - Bits 0-5 specify the maximum sector number, bits 6-7 are the high order 2 bits of the maximum cylinder number
DH - Maximum head number
DL - Total number of INT 13h devices with an INT 13h device number greater than 7Fh
carry set
AH - Error code

This function shall be used to find the CHS geometry used by INT 13 functions 2, 3, and 4 to access the drive.

4.7. Get Disk Type

Entry:

AH - 15h
DL - BIOS device number

Exit:

carry clear - Request was successful
AH - 03h = Hard Drive present
CX:DX - Number of sectors on the media
carry set
AH - Error code

This function shall be used to find the device type. This function is called by some versions of DOS during the boot process.

4.8. Get each physical hard disk information by ATA identify command

Entry:

AH - 25h

ECX - “AMD_” (Signature)

DL – Drive Number

DH – Low 4-bit (SATA port number)

– High 4-bit (Target ID if PM exists)

ES: BX - Transfer buffer (512 bytes)

Exit:

carry clear

ES: BX – Transfer buffer (512 bytes)

carry set

AL - 0FFh (Drive doesn’t exist on current port)

AH - Error code (80h, Drive fail to respond)

This function can be used to get each physical hard disk information by ATA identify command. Moreover, this sub-function call can be executed on init time or runtime stage as long as the caller prepare the suitable input parameter and the option ROM Int13h had been hooked.

However, the user still need to care two keys if want to use this sub-function call.
(1) If want to use this sub-function, first, the caller needs to find the suitable drive number (DL) value which can be accepted by OPROM and pass it to the OPROM to get the identify data from device.

(2) If there is no any hard disks and ONLY SATAPI device on the platform, this sub-function call can NOT work cause of the option ROM won’t treat the SATAPI as logical drive (80h,81h…).

4.9. Check Extensions Present

Entry:

AH - 41h

BX - 55AAh

DL - BIOS device number

Exit:

carry clear

AH - Version of extensions = 20h/21h

AL - Internal use only

BX - AA55h/55AAh

CX - Interface support bit map

carry set

AH - Error code (01h, Invalid Command)

This function shall be used to check for the presence of INT 13h extensions. If CF=1b, the extensions are not supported for the requested device. If CF=0b, BX shall be checked to confirm that it contains the value AA55h indicating that the extensions are present. If BX = AA55h, the value of CX shall be checked to determine what subsets of this interface are supported for the requested device. At least one subset shall be supported. The version of the extensions shall be 20h or 21h. This indicates that which INT 13h extensions this function is compliant with.

4.9. Extended Read

Entry:

AH - 42h

DL - BIOS device number

DS:SI - Device address packet

Exit:

carry clear

AH - 0

carry set

AH - Error code

This function shall transfer sectors from the device to memory.

4.10. Extended Write

Entry:

AH - 43h

AL - 0 or 1, write with verify off; 2, write with verify on

DL - BIOS device number

DS:SI - Device address packet

Exit:

carry clear

AH - 0

carry set

AH - Error code

This function shall transfer sectors from memory to the device.

4.11. Extended Verify Sectors

Entry:

AH - 44h

DL - BIOS device number

DS:SI - Device address packet

Exit:

carry clear

AH - 0

carry set

AH - Error code

This function verifies sectors without transferring data between the device and system memory.

4.12. Get Device Parameters

Entry:

AH - 48h

DL - BIOS device number

DS:SI - address of result buffer

Exit:

carry clear

AH - 0

DS:SI - address of result buffer

carry set

AH - Error code

This function returns default device parameters. It shall be mandatory regardless of the interface subset that is supported.

4.13. Terminate Disk Emulation

Entry:

AH - 4B

AL = 00, Return Status and Terminate Emulation

AL = 01, Return Status only, Do Not Terminate Emulation

DL = Drive number to terminate, 7F = Terminate all

DS:SI - Empty Specification Packet

Exit:

carry clear - System released

AH - 0

DS:SI - Completed Specification Packet

carry set - System not in emulation mode

AH - Error code

DS:SI - Completed Specification Packet

This function returns the system to a configuration that matches a normal floppy or hard disk boot.

5. AMD RAID / AHCI Common Command Interface

5.1-What is this function?

This RAID / AHCI common command interface is like the ATA / ATAPI command pass-through.

In the other words, in the RAID or AHCI mode, if the caller wants to submit the ATA or ATAPI command to the device like SMART or READ / WRITE, caller will prepare one packet and this packet describes how the ATA / ATAPI command be set. Then, to pass this packet to OPROM and the OPROM will return the necessary information to caller finally.

Especially, this interface is helpful for some customers who own their specific SBIOS feature.

5.2-How caller prepares the packet and pass to OPROM?

First, the OPROM will prepare one entry point for this function. The entry point in OPROM is located at 0x48 – 0x4B then caller can prepare the necessary packet and pass it to the entry point provided by OPROM.

However, the format version of “Interface Entry Point” has V1.0 and V2.0.The difference between them is V2.0 version will support one “Device Mapping Table” at 0x4D-0x4E and it will be used to tell caller what / where the device is on current platform. Moreover, the caller can refer to this table to submit ATA / ATAPI command to any device via AMD AHCI / RAID Common Command Interface. So, caller need to make sure the “Interface Entry Point” format version is V2.0 before using AMD AHCI / RAID Common Command Interface.

Interface Entry Point Format

Locate	Description
0x4B-0x4A	**ODD_AccessEntry_Seg:** Word, Entry Point Segment, locate at Oprom segment:offset 0x4B-0x4A.
0x49-0x48	**ODD_AccessEntry_Off:** Word, Entry Point Segment, locate at Oprom segment:offset 0x49-0x48.

Locate	Description
0x4C	Version 1.00 Format ODD Bit Map: Byte, Record ODD exist position. Locate at Oprom segment:offset 0x4C. Each bit set means ODD exist, otherwise ODD doesn't exist at this position.

ODD Position Bitmap	Input ODD channel value
BIT0	0 (if BIOS wants read "BIT0" odd, the input value must 0)
BIT1	1 (if BIOS wants read "BIT0" odd, the input value must 1)
BIT2	2 (if BIOS wants read "BIT0" odd, the input value must 2)
…	…
BIT7	**Must 0 (our chip only MAX support 6 ports) (SB700 only)**

Version 2.00 Format

BIT0-BIT6	**Reserved**
BIT7	**Must 1 indicate match spec version 2.00 (SB700 or later)**

Locate	Description
0x4E-0x4D	**Version 2.00 Using** **Device Mapping Table offset : Word, offset of the Exist Device Mapping Table. Max supports 10 devices.**

Input :

ecx	: '_AMD'
edx	: '_IRB'
es:bx	: irb structure

Output:

CY : Error
NC : Successful

```
Irb Structure definition
struct{
       UINT8 FeaturesReg;
       UINT8 Features_Exp;
       UINT8 SectorCountReg;
       UINT8 SectorCount_Exp;
       UINT8 SectorNumberReg;
       UINT8 SecNum_Exp;
       UINT8 CylLowReg;
       UINT8 CylLow_Exp;
       UINT8 CylHighReg;
       UINT8 CylHigh_Exp;
       UINT8 DriveHeadReg;
       UINT8 CommandReg;
       UINT8 bReserved[4];
} IDE_TASK_FILE_;

Receive_FIS_buffer definition
typedef struct _AHCI_D2H_REGISTER_FIS
{
       UINT8 FisType;        // 0x34
       UINT8 PM:4 ;
       UINT8 Reserved1 :2;
       UINT8 I:1;
       UINT8 Reserved2 :1;
       UINT8 Status;
       UINT8 Error;

       UINT8 SectorNumber;
       UINT8 CylLow;
       UINT8 CylHigh;
       UINT8 Dev_Head;

       UINT8 SectorNum_Exp;
       UINT8 CylLow_Exp;
       UINT8 CylHigh_Exp;
       UINT8 Reserved;

       UINT8 SectorCount;
       UINT8 SectorCount_Exp;
       UINT8 Reserved3[2];

       UINT8 Reserved4[4];
} AHCI_D2H_REGISTER_FIS, *PAHCI_D2H_REGISTER_FIS;



struct{
       UINT8 Irb_Function;
       UINT8  Irb_Status;
       UINT8  Channel;              // physical port
       UINT8  TargetId;             // Port Multiplier port
       UINT32 Irb_Flags;
       UINT32 Receive_FIS_buffer;     //physical address, word boundary
       UINT32 DataTransferLength;     //buffer length, length must be even,
                                 //less than 64k
       UINT32 DataBuffer;            //input/output buffer, physical address
                                 //word boundary
       union {

       //
       // ATA Task file register contents
       //
       IDE_TASK_FILE_ IdeTaskFile;

       //
       // CDB for ATAPI devices
```

```
        //
        UINT8 Cdb[16];

        };

} IRB;

//
// irb flags
//
#define IRB_FLAGS_DATA_IN_                    0
#define IRB_FLAGS_DATA_OUT_                   1
//
// irb status
//
#define IRB_STATUS_PENDING_                   0x0
#define IRB_STATUS_SUCCESS_                   0x1
#define IRB_STATUS_DATALENGTH_MISMATCH_         0x2
#define IRB_STATUS_DEVICE_ERROR_               0x3
#define IRB_STATUS_INVALID_REQUEST_             0x4
#define IRB_STATUS_BUS_RESET_                  0x5
#define IRB_STATUS_SELECTION_TIMEOUT_           0x6
#define IRB_STATUS_BUSY_                      0x7
#define IRB_STATUS_ABORT_                     0x8
//
// irb function
//
#define IRB_FUNCTION_ATA_COMMAND_               0
#define IRB_FUNCTION_ATAPI_COMMAND_             1
```

*note: This item depends on COMMAND type. If it is CDB formats defined in SCSI documents, using IRB_FUNCTION_ATAPI_COMMAND_. If commands defined in ATA/ATAPI specification, using IRB_FUNCTION_ATA_COMMAND_.

//Ex for Identify Cmd.

```
// clear irb buffer
        zero_irb();
//identify is ATA cmd
        irb->Function = IRB_FUNCTION_ATA_COMMAND_;
//initial irb status
        irb->IrbStatus = IRB_STATUS_PENDING_;
// it is read data command
        irb->IrbFlags = IRB_FLAGS_DATA_IN_
// identify data is 256 words
        irb->DataTransferLength = 512
// return data buffer
        irb->DataBuffer = 0x12345678
// match AT/ATAPI spce
        irb->IdeTaskFile.bDriveHeadReg = 0xa0;
        irb->IdeTaskFile.bCommandReg = 0xEC or 0xA1;
// which port / target you want to read
        irb->Channel = portnumber;
        irb->TargetId = TargetID;
// jump into entry_point
        call ENTRY_point
```

//Ex for S.M.A.R.T

```
        irb->Function = IRB_FUNCTION_ATA_COMMAND_;
        irb->IrbStatus = IRB_STATUS_PENDING_;
        irb->IrbFlags = IRB_FLAGS_DATA_OUT_;
        irb->DataTransferLength = 0;
        irb->DataBuffer = 0x12345678;
        irb->IdeTaskFile.bCommandReg = 0xB0;
        irb->IdeTaskFile.bFeaturesReg = 0xD8;   // enable S.M.A.R.T.
```

```
       irb->IdeTaskFile.bCylLowReg = 0x4F;
       irb->IdeTaskFile.bCylHighReg = 0xC2;
       irb->IdeTaskFile.bDriveHeadReg = 0xa0;
       irb->Channel = portnumber;
       irb->TargetId = TargetID;
```

// Check SMART COMMAND STS

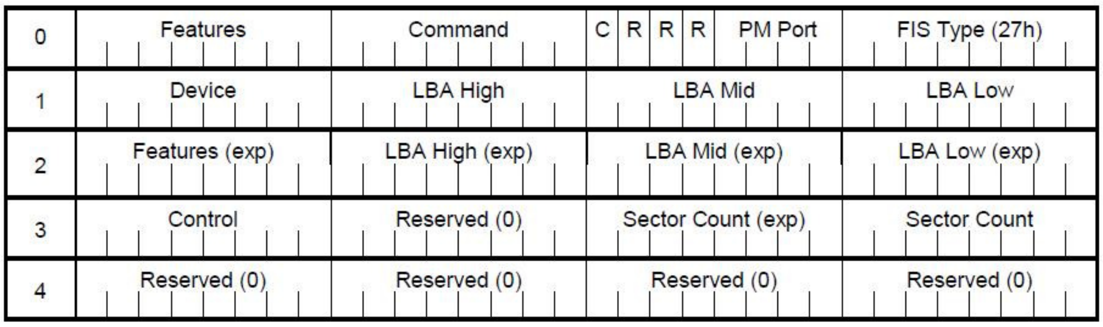
```
       PAHCI_D2H_REGISTER_FIS pD2H = irb->Receive_FIS_buffer;
       if (pD2H->CylHigh == 0xC2 &&
          pD2H->CylLow  == 0x4F)
       {
              // smart ok
       }
       else
              // smart fail
```

//Ex for ATAPI Read

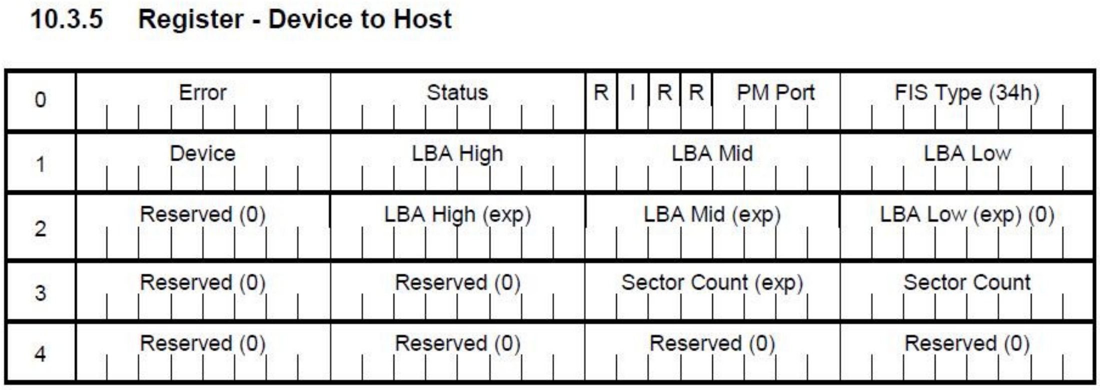
```
       irb->Function = IRB_FUNCTION_ATAPI_COMMAND_;
       irb->IrbStatus = IRB_STATUS_PENDING_;
       irb->IrbFlags = IRB_FLAGS_DATA_IN_;
       irb->DataTransferLength = 2048;
       irb->DataBuffer = 0x12345678;
       irb->Cdb[0] = 0x28; //ATAPI_READ
// read block 17
       irb->Cdb[2] = 0;       //
       irb->Cdb[3] = 0;       //
       irb->Cdb[4] = 0;       //
       irb->Cdb[5] = 0x11;     //
// we read 1 logical block.
       irb->Cdb[7] = 0x0;
       irb->Cdb[8] = 0x1;
       irb->Channel = portnumber;
       irb->TargetId = TargetID;
```

FIS defination

10.3.4 Register - Host to Device



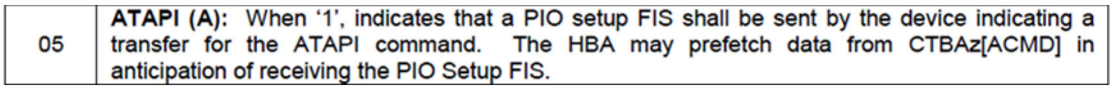
0	Features	Command	C	R	R	R	PM Port	FIS Type (27h)
1	Device	LBA High	LBA Mid					LBA Low
2	Features (exp)	LBA High (exp)	LBA Mid (exp)					LBA Low (exp)
3	Control	Reserved (0)	Sector Count (exp)					Sector Count
4	Reserved (0)	Reserved (0)	Reserved (0)					Reserved (0)



10.3.5 Register - Device to Host

0	Error	Status	R I R R PM Port	FIS Type (34h)
1	Device	LBA High	LBA Mid	LBA Low
2	Reserved (0)	LBA High (exp)	LBA Mid (exp)	LBA Low (exp) (0)
3	Reserved (0)	Reserved (0)	Sector Count (exp)	Sector Count
4	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)

Command Head BIT 5



05	**ATAPI (A):** When '1', indicates that a PIO setup FIS shall be sent by the device indicating a transfer for the ATAPI command. The HBA may prefetch data from CTBAz[ACMD] in anticipation of receiving the PIO Setup FIS.

Device Mapping Table Format (V2.0)
Format:

15	9	8	7 4	3 0
Reserved	Reserved	D	Target	Port

Table End Value: 0XFFFF

D flag: 0 – Hdd / 1 – Odd

Port: Indicate current port, Max value is 6

Target: If PM exist, it indicate the which port on PM, Otherwise it should be zero. Max value is 0xE.

Input parameter, orginal is port number, byte, now the byte high 4 bit, target id. Low 4 bit, port number.