# Intel® 82801EB (ICH5) and Intel® 82801ER (ICH5R) Serial ATA Controller

## Programmer's Reference Manual (PRM)

*July 2003*

Document Number: 252671-002

# Contents

# Figures



# Tables

# Revision History

| Revision Number | Description | Revision Date |
|---|---|---|
| -001 | Initial Release | April 2003 |
| -002 | Updated register/bit names to match Intel® 82801EB I/O Controller Hub 5 (ICH5) / Intel® 82801ER I/O Controller Hub 5 R (ICH5R) Datasheet | July 2003 |

This page is intentionally left blank.

# 1 Introduction

## 1.1 Overview

This document was prepared to assist BIOS software providers and Operating System (OS) providers in supporting the Intel® 82801EB (ICH5)/ 82801ER (ICH5R) SATA Controller feature set and programming interface.

It is assumed that the reader has a working knowledge of ATA/SATA architecture. Also, the reader should have an understanding of ATA, BIOS (including ACPI) and device driver development for the target operating systems.

This document also describes functions that the BIOS and the OS shall perform in order to ensure correct and reliable operation of the platform. This document will be supplemented from time to time with specification updates. The specification updates contain information relating to the latest programming changes. Check with your Intel representative for availability of specification updates.

This document does not cover any of the software requirements around configuration of the RAID controller in ICH5R.

The recommendations in this Programmers Reference Manual (PRM) apply to the following components:

**Table 1. Intel® ICH5 Device IDs**

| Intel® ICH Device | Device Function Number | PCI Device ID | PCI Vendor ID |
|---|---|---|---|
| Intel® 82801EB (ICH5) and Intel® 82801ER (ICH5R) | 02h | 24D1 | 8086h |

***Note:*** In this document, references to an ICHx device that has a corresponding ICHx-M/R device will include the ICHx-M/R part (e.g., reference to ICH5 includes ICH5R).

This page is intentionally left blank.

# 2 Conventions

## 2.1 Register Access

This document uses the following notation as related to register access: ***RegOffset.BitOffset***.

*Where:*

- ***RegOffset*** specifies the name of the register to be accessed (either in I/O or PCI configuration space)
- ***BitOffset*** specifies the name of a bit contained within ***RegOffset*** that is to be accessed.

*Example (Uses the Class Code register defined in the table below):*

Assumes the following standard PCI configuration register (Class Code) with

***CC.SCC*** refers to the Sub Class Code (**SCC -** bits 0:7) implemented within the Class Code (**CC** – offset 0Ah) register in the PCI Configuration space.

Offset 0Ah - Class Code Register (CC)

| Bits | Type | Reset | Description |
|---|---|---|---|
| 15:08 | RO | 01h | **Base Class Code (BCC):** Indicates that this is a mass storage device. |
| 07:00 | RO | 01h | **Sub Class Code (SCC):** Indicates that this is an IDE device. |

## 2.2 Keywords

- **Mandatory -** A keyword indicating items to be implemented as defined by this document.
- **System Software** – A keyword that refers to both BIOS and operating system software unless specifically stated otherwise.
- **Shall -** A keyword indicating a mandatory requirement. Equivalent to the term “must.”
- **Should -** A keyword indicating flexibility of choice with a strongly preferred alternative. Equivalent to the phrase “it is recommended.”

This page is intentionally left blank.

# 3 Intel® ICH SATA Controller Basic Attributes

The register set for the ICH5 SATA controller is basically identical to that of the integrated parallel ATA controller. Because the underlying SATA functionality is transparent to operating system software, it need not have any special knowledge about SATA or SATA devices. In addition to supporting the same programming interface, the SATA host controller can also be configured to use legacy ATA resources as well as native PCI resources.

## 3.1 Legacy Sub-Mode

A host controller (channel) configured for legacy sub-mode of operation has the following requirements:

- Has its Programming Interface register set for legacy mode
- Shall interrupt via IRQ14 (primary channel) and IRQ15 (secondary channel)
- Command and control block are accessed at fixed I/O locations:
  - **Command Block Offset:** 01F0h for primary and 0170h for secondary
  - **Control Block Offset**: 03F4h for primary and 0374h for secondary

## 3.2 Native Sub-Mode

A host controller (channel) configured for native sub-mode of operation has the following requirements:

- Has its Programming Interface register set for native mode.
- Shall interrupt via the INTA#.
- Command and control blocks are accessed via I/O space specified by the following BARs located by the following PCI configuration offsets:
  - Offset 10h – Primary Command Block Base Address
  - Offset 14h – Primary Control Block Base Address
  - Offset 18h – Secondary Command Block Base Address
  - Offset 1Ch – Secondary Control Block Base Address

## 3.3 Host Controller Configurations

The SATA host controller can function independently of, or in conjunction with the parallel ATA (P-ATA) host controller. The ICH5 can support a maximum of six ATA devices: four parallel ATA device plus two serial ATA devices.

The flexibility and software transparency of the SATA host controller presents an interesting problem for older operating systems that do not comprehend or support the native mode of operation (i.e., the ports/channels on both controller's could be configured to use legacy resources, which would result in hardware resource conflicts). To prevent this problem, ICH5 supports multiple SATA and P-ATA device configurations: compatible and enhanced. These configurations are discussed on the next section.

# 4 Theory of Operation

This section describes the proper usage and programming of the SATA host controller by BIOS and the operating system when the host controller is operating in compatible or enhanced configuration.

## 4.1 Compatible Configuration

The compatible configuration is for the express purpose of **maintaining backward-compatibility** with those operating systems that do not comprehend native mode of operation. In this configuration, a maximum of four ATA (serial and/or parallel) devices on the primary and/or secondary channels can be supported. This mode allows both the primary and secondary channels to be configured for legacy and/or native sub-modes of operation.

Since the ICH5 supports up to six ATA devices (parallel + serial), it is possible for an end-user to populate a platform with a combination of parallel and serial ATA devices. The fact that the ICH5 supports up to six ATA devices is not necessarily an issue for older operating systems; it is the fact that both the P-ATA controller and SATA controller can be configured to consume legacy resources (see Section: *3.1 Legacy Sub-Mode*) that causes the issues (i.e., the primary and secondary channel interfaces will attempt to share the same resources).

The compatible configuration solves this dilemma. The compatible configuration allows certain device configurations to be accessed by software without the issue of legacy resource conflicts and yet remain backward compatible with operating systems that don't implement native mode functionality.

The ICH5 supports three compatible configuration options. These options are summarized as follows and are discussed in detail in the following sections:

- P-ATA devices only (maximum of four) – Compatible Configuration Option 1
- SATA devices only (maximum of two) – Compatible Configuration Option 2
- P-ATA (maximum of two) and SATA devices (maximum of two) – Compatible Configuration Option 3

***Note:*** Proper support of these options requires that system BIOS provide a BIOS setup option (that is subsequently saved in nonvolatile memory) that allows the end user to select an option that is appropriate for their particular hardware/operating system configuration. To insure backward compatibility with existing software, BIOS is **not permitted** to dynamically select a configuration.

## 4.1.1 Additional Register Support

Support of certain Compatible configuration options requires that the ICH5 implement an additional hardware register that is configurable via BIOS. This register is located in the SATA function's PCI configuration space at offset 90h and is defined below. The usage model for this register is described in subsequent sections.

### 4.1.1.1 MAP – Address Map Register – Offset 90h

This register is set by BIOS during POST. The exact value programmed into this register is based on a BIOS setup option. Dynamic modifications of this register by BIOS as a result of POST or some other event (e.g., docking event) is not permitted, as the underlying operating system software may not comprehend the new device topology and may result in undefined behavior.

<table>
<tr><th>Bit</th><th>Type</th><th>Reset</th><th>Description</th></tr>
<tr><td>7:3</td><td>RO</td><td>0</td><td>Reserved</td></tr>
<tr><td>2:0</td><td>RW</td><td>000</td><td>Map Value (MV): The value in the bits below indicate the address range the SATA port responds to, and whether or not the P-ATA and SATA functions are combined.
<table>
<tr><th>Bits</th><th>Mapping</th></tr>
<tr><td>000</td><td>Non-combined, SATA Port 0 is primary master, SATA Port 1 is secondary master</td></tr>
<tr><td>001</td><td>Non-combined, SATA Port 0 is secondary master, SATA Port 1 is primary master</td></tr>
<tr><td>100</td><td>Combined, SATA Port 0 is primary master, SATA Port 1 is primary slave, P-ATA is secondary</td></tr>
<tr><td>101</td><td>Combined, SATA Port 0 is primary slave, SATA Port 1 is primary master, P-ATA is secondary</td></tr>
<tr><td>110</td><td>Combined, P-ATA is primary, SATA Port 0 is secondary master, SATA Port 1 is secondary slave</td></tr>
<tr><td>111</td><td>Combined, P-ATA is primary, SATA Port 0 is secondary slave, SATA Port 1 is secondary master</td></tr>
</table></td></tr>
</table>

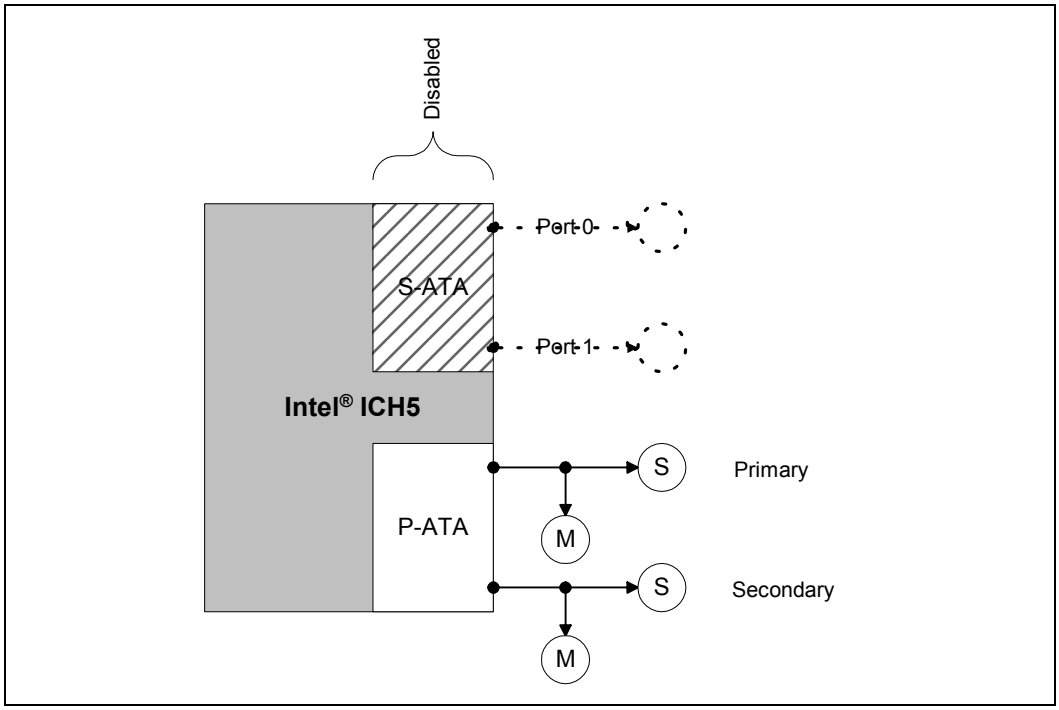
***Caution:*** Programming the ***MAP.MV*** register with values other than those specified above will result in undefined hardware behavior.

***Note:*** The P-ATA host controller does not implement this register.

## 4.1.2 Compatible Configuration - Option 1

This option is selected when one or more P-ATA devices are to be used. SATA device(s) **may or may not** be attached to the SATA ports, but will not be accessible to software. Figure 1 illustrates this configuration:

**Figure 1. Compatible Configuration - Option 1**




***Note:*** In the figure above, devices represented by dotted lines may be attached, but are not accessible to software.

***Note:*** This configuration requires no additional programming of the **Port Mapping** register, as in this configuration it has no effect on the P-ATA or SATA functions.

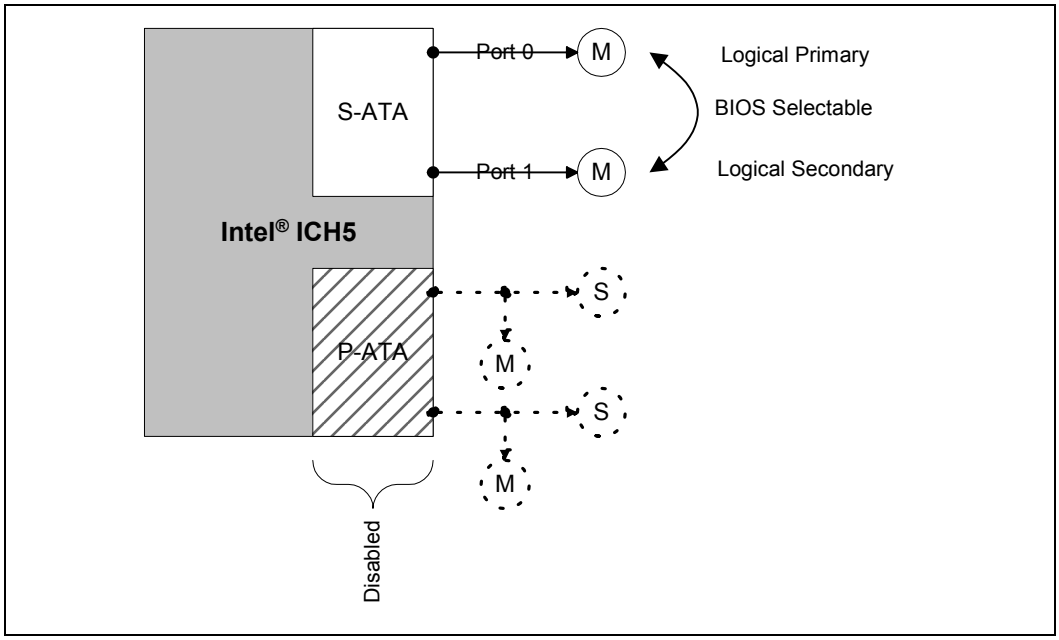
To enable this configuration, system BIOS:

1. Shall not program the SATA (Device 31, Function 2) controller's base address registers (Offsets 10h – 24h in PCI configuration space).
2. Shall disable access to the SATA controller's I/O space by programming the command register (PCI configuration, offset 04h, bit 0) with a 0.
3. Shall disable the SATA function by programming bit 2 (*D31_**F2_DISABLE***) of the ***Function Disable*** register (Device 31, Function 0, Offset F2h) with a 1. This will insure that the PCI configuration registers associated with the SATA function are not decoded and thus will insure that operating system configuration software does not enumerate and configure the SATA function.
4. Shall insure that the SATA ports are not enabled. This is accomplished by writing '0' to bits 1:0 in the ***Port Control and Status (PCS)*** register.
5. Shall program the P-ATA registers appropriately (the exact details are beyond the scope of this document).

## 4.1.3 Compatible Configuration - Option 2

This option is selected when one or more (maximum of two) SATA devices are to be used. P-ATA device(s) **may or may not** be attached to the P-ATA channels, but will not be accessible to software. Figure 2 illustrates this configuration:

**Figure 2. Compatible Configuration - Option 2**




***Note:*** In the figure above, devices represented by dotted lines may be attached, but are not accessible to software.

***Note:*** In this configuration, software reads and writes to the slave device registers will result in a master abort and as such reading from the slave device registers will return all 1. The Drive/Head register (offset 06h in the command block) is the exception.

To enable this configuration, system BIOS:

1. Shall not program the P-ATA (Device 31, Function 1) controller's base address registers (Offsets 10h – 24h in PCI configuration space).
2. Shall disable access to the P-ATA controller's I/O space by programming the command register (PCI configuration, offset 04h, bit 0) with a 0.
3. Shall disable the P-ATA function by programming bit 1 (*D31_**F1_DISABLE***) of the ***Function Disable*** register (Device 31, Function 0, Offset F2h) with a 1. This will insure that the PCI configuration registers associated with the P-ATA function are not decoded and thus will insure that operating system configuration software does not enumerate and configure the P-ATA function.
4. Shall program the ***MAP.MV*** register as follows:

***Note:*** These mappings represent a *master-master* device arrangement.

- ***MAP.MV*** == '000b'. This indicates that a SATA device on port 0 behaves as a *master* device on the logical primary channel and a device on port 1 behaves as a *master* device on the logical secondary channel.

The flexibility of the mapping register also allows the following configuration:

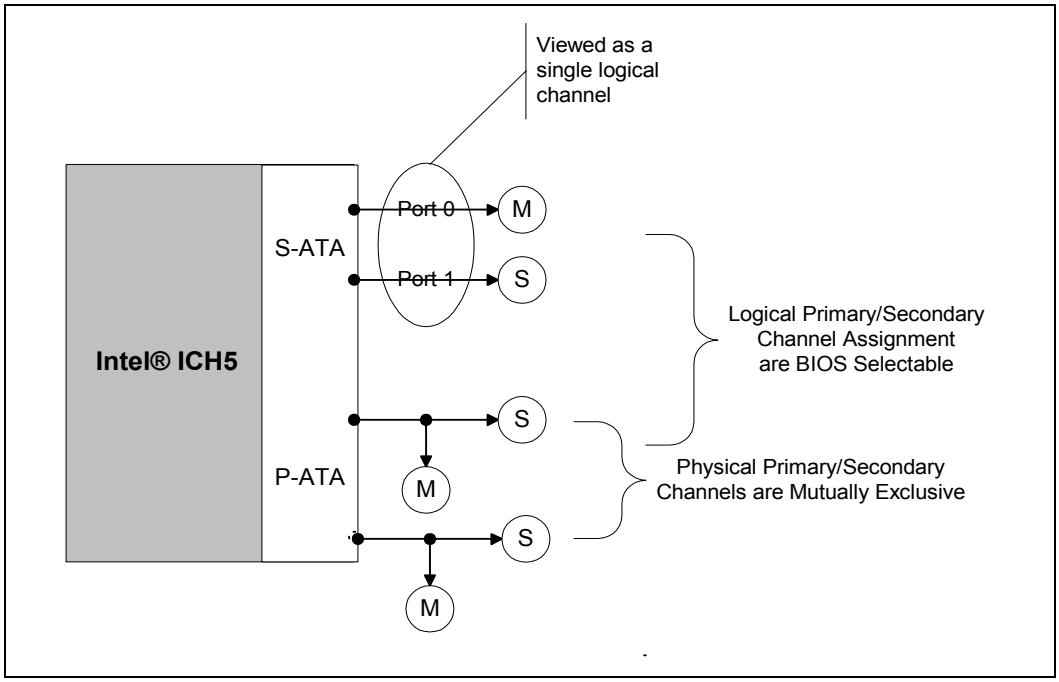
- ***MAP.MV*** == '001b'. This indicates that a SATA device on port 0 behaves as a *master* device on the logical secondary channel and a device on port 1 behaves as a *master* device on the logical primary channel.

5. Shall program the SATA registers appropriately. Because the programming interface is identical to that of P-ATA, BIOS should follow the same programming guidelines as used for the P-ATA (the exact details are beyond the scope of this document).

## 4.1.4 Compatible Configuration - Option 3 (Combined)

This option is selected when at least one SATA device (maximum of two) and at least one P-ATA device (maximum of two) are connected to both SATA and P-ATA host controllers and are both accessible by software. This configuration is referred to as the "combined" mode. In combined mode of operation, devices can be supported in legacy mode, native mode or both. In this mode, it is expected that the host controller will be programmed for legacy mode of operation. It is important to note that in this configuration, the SATA and P-ATA host controllers share functionality but appear as a single PCI function. In this case, the actual P-ATA function is hidden from system software (i.e., cannot be enumerated or accessed directly) but P-ATA devices connected to the function (primary channel only) may still be accessed. These devices are accessed via the standard, P-ATA compatible register set exposed by the SATA controller. Figure 3 illustrates this configuration:

**Figure 3. Compatible Configuration - Option 3**

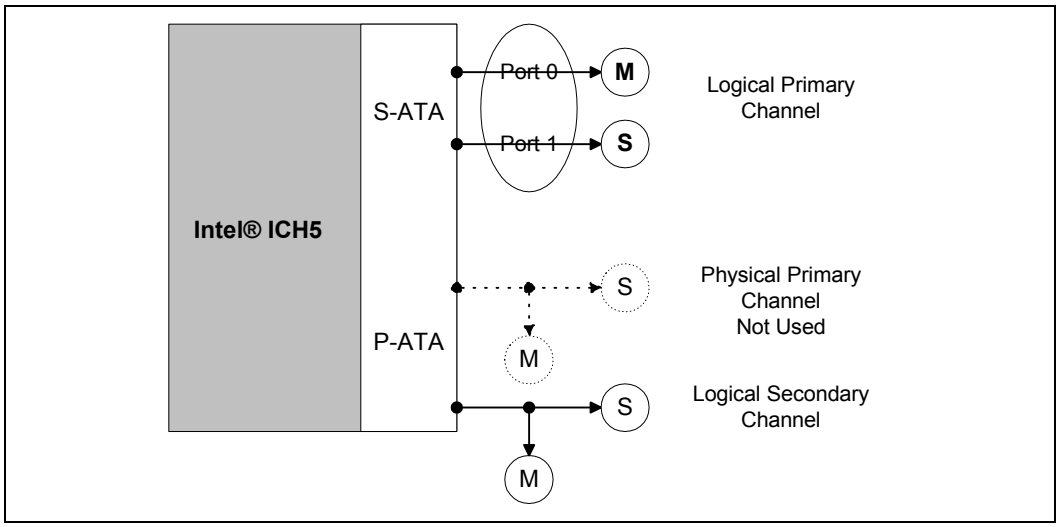


To enable this configuration, the system BIOS:

1. Shall not program the P-ATA (Device 31, Function 1) controller's base address registers (Offsets 10h – 24h in PCI configuration space).
2. Shall disable access to the P-ATA controller's I/O space by programming the command register (PCI configuration, offset 04h, bit 0) with a 0.
3. Shall disable the P-ATA function by programming bit 1 (*D31_**F1_DISABLE***) of the Function Disable register (Device 31, Function 0, Offset F2h) with a 1. This will insure that the PCI configuration registers associated with the P-ATA function are not decoded and thus will insure that operating system configuration software does not enumerate and configure the P-ATA function.
4. Shall program the ***MAP.MV*** register as follows:

   - If SATA is the **primary** channel and P-ATA is the **secondary** channel and Port 0 device is primary **master** and Port 1 device is primary **slave** then ***MAP.MV*** == '100b'. Figure 4 illustrates this configuration:

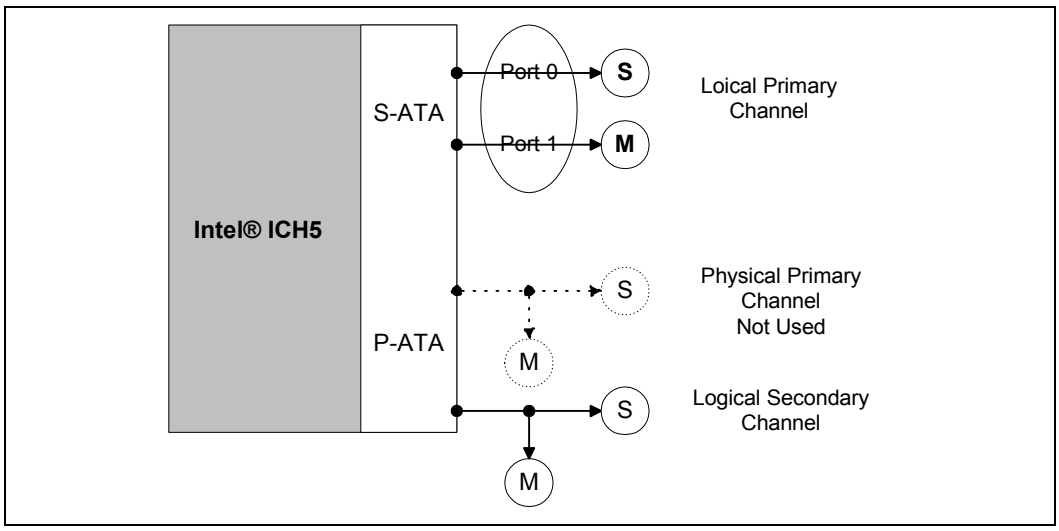
**Figure 4. Compatible Configuration - Option 3a**




***Note:*** In the figure above, devices represented by dotted lines may be attached, but are not accessible to software.

- If SATA is the **primary** channel and P-ATA is the **secondary** channel and Port 0 device is primary **slave** and Port 1 device is primary **master** then ***MAP.MV*** == '101b'. Figure 5 illustrates this configuration:

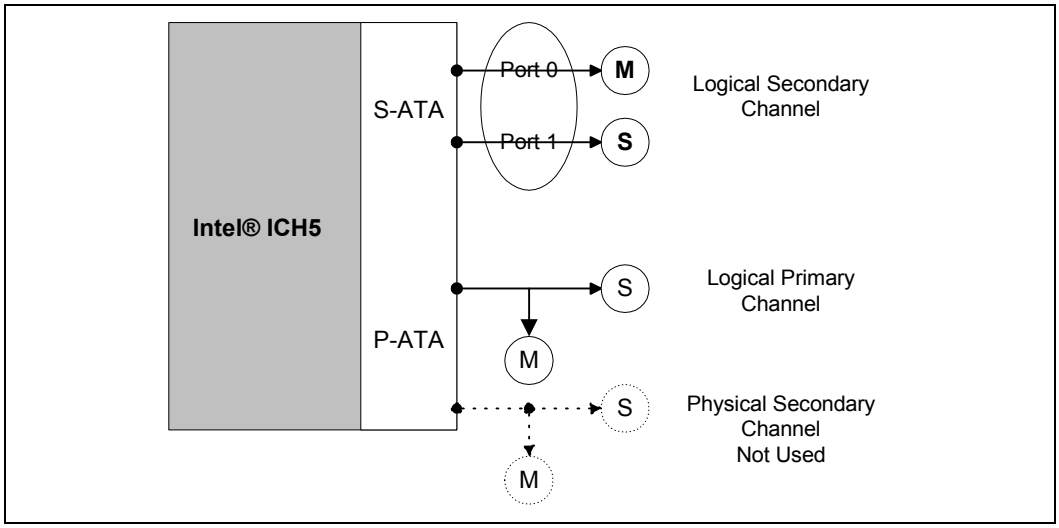
**Figure 5. Compatible Configuration - Option 3b**




***Note:*** In the figure above, devices represented by dotted lines may be attached, but are not accessible to software.

- If SATA is the **secondary** channel and P-ATA is the **primary** channel and Port 0 device is secondary **master** and Port 1 device is secondary **slave** then ***MAP.MV*** == '110b'. Figure 6 illustrates this configuration:

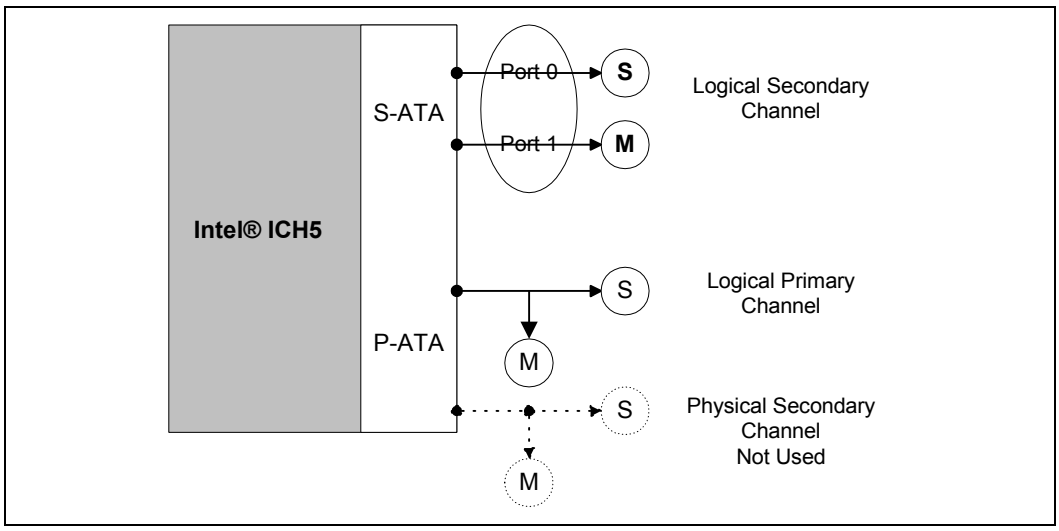
**Figure 6. Compatible Configuration - Option 3c**




***Note:*** In the figure above, devices represented by dotted lines may be attached, but are not accessible to software.

- If SATA is the ***secondary*** channel and P-ATA is the ***primary*** channel and Port 0 device is secondary ***slave*** and Port 1 device is secondary ***master*** then ***MAP.MV*** == '111b'. Figure 7 illustrates this configuration:

**Figure 7. Compatible Configuration - Option 3d**



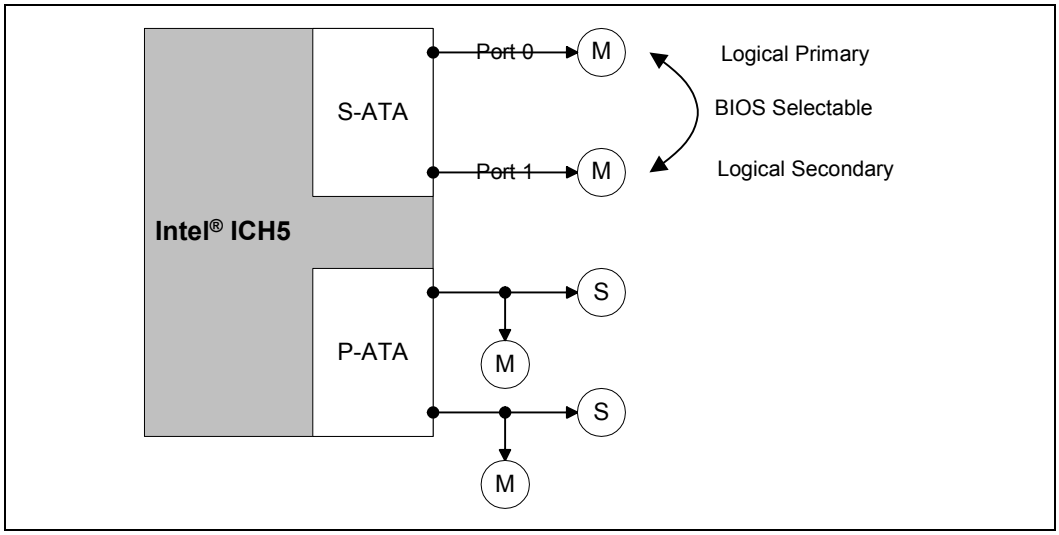

***Note:*** In the figure above, devices represented by dotted lines may be attached, but are not accessible to software.

## 4.2 Enhanced Configuration

The enhanced configuration is intended for those operating systems (e.g., Microsoft Windows* 2000, Windows* XP) that comprehend both legacy and native modes of operation. It is the preferred configuration for system software due to the maximum flexibility that it offers. Enhanced mode allows both the P-ATA and SATA host controllers to be used, providing a maximum of six ATA (4 P-ATA + 2 SATA) devices that can be used simultaneously.

***Note:*** In this configuration, the P-ATA controller must be programmed for legacy mode while the SATA controller must be programmed for native mode of operation. Figure 8 illustrates the enhanced mode configuration:

**Figure 8. Enhanced Configuration**




***Note:*** While the enhanced configuration can support a maximum of six ATA devices, Intel recommends that its customers limit their platforms to a four-device maximum configuration (applicable to both compatible and enhanced configuration), as this will provide for an easier transition to a four SATA device maximum configuration on future ICHs.

## 4.2.1 PI - Programming Interface Register – Offset 09h

As stated previously, the P-ATA and SATA channels can be configured for either native mode only or a combination of legacy and native modes. This is controlled via the **Programming Interface** register. The programming interface register is found in both the SATA and P-ATA functions and can be modified by both BIOS (during POST) and operating system software.

| Bit | Type | Reset | Description |
|---|---|---|---|
| *7* | *RO* | *1* | *Indicates the SATA controller supports bus master operation.* |
| *6:4* | *RO* | *0* | *Reserved* |
| 3 | RO | 1 | **SOP_MODE_CAP:** Indicates that the secondary controller supports both legacy and native modes. |
| 2 | RW | 0 | **SOP_MODE_SEL:** Determines the mode that the secondary channel is operating in.<br>0 = Legacy PCI mode (Default)<br>1 = Native PCI mode |
| 1 | RO | 1 | **POP_MODE_CAP:** Indicates that the primary controller supports both legacy and native modes. |
| 0 | RW | 0 | **POP_MODE_SEL:** Determines the mode that the primary channel is operating in.<br>0 = Legacy PCI mode (Default)<br>1 = Native PCI mode |

Table 2 illustrates the valid values that system BIOS can use for the programming interface register when in enhanced mode.

***Note:*** The ICH5 does permit the SATA and P-ATA host controllers to simultaneously operate in native mode if they are programmed to do so by the operating system. Due to potential operating system incompatibilities, it is a requirement (when in enhanced mode) that the system BIOS programs the P-ATA and SATA host controllers exactly as described in Table 2.

***Caution:*** Improper programming **could** result in undefined behavior.

Some Microsoft operating systems have specific platform support requirements when operating on systems capable of native mode of operation. Refer to paper titled *BIOS Settings for Native-Mode-Capable ATA Controllers*, available from Microsoft Corporation at: http://www.microsoft.com/hwdev/storage/, for additional operating system-related details.

**Table 2. Valid BIOS Option for the Programming Interface Register**

| SATA – Device 31, Function 2 | | P-ATA – Device 31, Function 1 | |
|---|---|---|---|
| POP_MODE_SEL | SOP_MODE_SEL | POP_MODE_SEL | SOP_MODE_SEL |
| 1 | 1 | 0 | 0 |

***Note:*** The SATA and P-ATA host controllers do not support the programming of the primary and secondary channels differently (i.e., primary as native and secondary as legacy). Programming the controllers with values other than those specified above is illegal and will result in undefined hardware behavior.

Table 3 illustrates an illegal programming combination. System BIOS programming of all the channels (both the SATA and P-ATA controllers) for legacy mode or native will result in undefined behavior. Operating system software requiring legacy only mode capability is required to find the SATA host controller configured for compatible mode.

**Table 3. Illegal BIOS Options for the Programming Interface Register**

| SATA – Device 31, Function 2 | | P-ATA – Device 31, Function 1 | |
|---|---|---|---|
| POP_MODE_SEL | SOP_MODE_SEL | _ SOP_MODE_SEL | SOP_MODE_SEL |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |

***Note:*** Following a reset, the SATA and P-ATA controllers will both be configured for legacy mode. Therefore it is very important that system BIOS program these registers with legal values as defined in Table 2. Valid BIOS Option for the Programming Interface Register

### 4.2.2 MAP Register Programming

As shown in Figure 8. Enhanced Configuration, enhanced mode configures the SATA so each SATA port is viewed as individual logical channels with a single master device. Using the **MAP** register, the logical SATA channels can be configured so that port 0 is the logical primary channel and port 1 is the logical secondary channel or vice-versa. The programming for this register is as follows:

- ***MAP.MV*** == '000b'. This indicates that a SATA device on port 0 behaves as a *master* device on the logical primary channel and a device on port 1 behaves as a *master* device on the logical secondary channel.
- ***MAP.MV*** == '001b'. This indicates that a SATA device on port 0 behaves as a *master* device on the logical secondary channel and a device on port 1 behaves as a *master* device on the logical primary channel.

When in Enhanced mode, programming the MAP register to values other than '000b' or '001b' could result in undefined hardware behavior. The MAP register shall be programmed by the BIOS only during POST.

## 4.3 PCS - Port Control and Status Register – Offset 92h

To provide better power management and device presence capabilities, the SATA host controller implements a **Port Control and Status (PCS)** register. The PCS register provides improved power savings in that it allows system software to disable individual SATA ports. Device presence detection is beneficial to system software as this can greatly reduce boot times and resume times (from S3 and below).

| Bit | Type | Reset | Description |
|---|---|---|---|
| *7:6* | *RO* | *0* | *Reserved* |
| 5 | RO | 0 | **Port 1 Present (P1P):** When set, the SATA host has detected the presence of a device on port 1. It may change at any time. This bit is cleared when the port is disabled via the P1E bit (bit 1 of this register). |
| 4 | RO | 0 | **Port 0 Present (P0P):** When set, the SATA host has detected the presence of a device on port 0. It may change at any time. This bit is cleared when the port is disabled via the P0E bit (bit 0 of this register). |
| *3:2* | *RO* | *0* | *Reserved* |
| 1 | RW | 0 | **Port 1 Enabled (P1E):** When set, the port is enabled. When cleared, the port is disabled. When enabled, the port is in the "on" state and can detect devices. When disabled, the port is in the "off" state and cannot detect any devices. |
| 0 | RW | 0 | **Port 0 Enabled (P0E):** When set, the port is enabled. When cleared, the port is disabled. When enabled, the port is in the "on" state and can detect devices. When disabled, the port is in the "off" state and cannot detect any devices. |

#### 4.3.1.1 Port Enabling/Disabling

By default, the SATA ports are set (by hardware) to the disabled state as a result of a D3 to D0 power state transition (due to initial power-on reset or resume from suspend).

System software may choose to (keep) disable a port as a result of a device being disconnected from a port(s). Overall power consumption can be reduced if system software only enables those ports that have SATA devices attached. This is especially beneficial to mobile systems.

The following are general guidelines to be used for determining when the SATA ports **shall** be enabled and when they **should** be disabled:

*Note:* ***IMPORTANT –*** A port **shall not** be enabled if power is not applied to the attached SATA device. Power shall be applied to the device for a period of at least 30ms before its associated port is to be enabled by system software. System software **shall** insure that a port is disabled prior to removing power to the associated SATA device.

- A port on the ICH5 SATA host controller shall be enabled in order for the ***PCS.PxP*** (where ***x*** is 0 for Port 0 or 1 for Port 1) bits to be accurate. See Section *4.4.1 Hardware and Software Considerations.*
- As part of a robust power conservation strategy, a port on the ICH5 SATA host controller should be disabled when a SATA device is not physically present or is unusable.

Ports may be enabled and disabled via the following methods:

- Operating system device driver (either directly or through ACPI)
- ACPI power management control method (e.g., _PSx)

##### 4.3.1.1.1 BIOS Considerations

To insure that SATA devices are functional following certain system power state transitions, system BIOS shall enable the SATA ports under the following conditions:

- Any APM supported system state transition where the ICH5 SATA host controller is reset.
- Device power state transitions where the operating system **cannot** enable the SATA ports through the use of a native (non-ICH5 SATA host controller aware) device driver or through the use of ACPI control methods.

These considerations do not apply if **both** of the following conditions are true:

- The SATA devices are not crucial to operating system boot or for resuming from hibernation

**AND**

- The operating system implements an ICH5 SATA host controller aware device driver.

Failure by the system BIOS to provide the required support will result in device inaccessibility and/or loss of operating system functionality.

#### 4.3.1.1.2 Enabling/Disabling a SATA Port from an Operating System Driver

To disable or enable a SATA port, system software need only program the ***PCS.PxE*** (where ***x*** is 0 for Port 0 or 1 for Port 1) bit(s) with a 0 or a 1. Note that the SATA host controller hardware allows the ***PCS.PxE*** bits to be written to individually or simultaneously. A WDM driver may program the ***PCS.PxE*** bits directly (through the PCI driver) or indirectly (through an ACPI control method – see *A.4 Device Presence Check – Using ACPI* for an example of how to call an ACPI control method from a WDM driver).

*A.1 Enabling/Disabling SATA Ports from a WDM Driver* illustrates how a WDM driver could read/write the ***PCS.PxE*** bits for enabling or disabling the SATA port(s).

***Note:*** Because the SATA host controller is designed with backward compatibility in mind, it is not expected that existing operating system software designed for P-ATA host controllers should ever have to modify the ***PCS.PxE*** bits directly; this shall be done by the system BIOS and/or ACPI control methods (see Section: *4.3.1.1.3 Enabling/Disabling a SATA Port from ACPI*).

#### 4.3.1.1.3 Enabling/Disabling a SATA Port from ACPI

Enabling and disabling of the SATA ports is also a function of the standard ***PSx*** (_PS0 or _PS3 – these control methods can be used to supplement the device power management and are executed whenever the operating system wishes to place the SATA into the D0 and D3 power states, respectively) control methods found in the ACPI namespace for the SATA controller.

By enabling the SATA ports when the host controller transitions to the D0 power state (via ***_PS0***) and disabling the SATA ports prior to the host controller transitioning to the D3 power state (via ***_PS3***), backward compatibility with system software designed for non-ICH5 SATA aware ACPI operating systems will be maintained.

Using ACPI can also be beneficial to device drivers that comprehend the ICH5 SATA in that the ACPI implementation abstracts the actual hardware implementation and location of the ***PCS*** register. This is especially useful if hardware designers require a change in the location in PCI configuration space of the ***PCS*** register due to silicon stepping requirements. Using ACPI could prevent a re-spin of the device driver and any subsequent re-validation.

*A.2 Enabling/Disabling SATA Ports in the _PSx Control* Method provides an example of how the SATA ports are enabled and disabled via the ***_PS0*** and ***PS3*** control methods.

***Note:*** For those platforms that support hot insertion/removal of SATA devices through specialized hardware, the associated ACPI control methods (e.g., _EJx) shall also implement ASL (ACPI Script Language) code that disables a SATA port (for device removal events) and enables a SATA port (for device insertion events).

## 4.4 Device Presence Detect

The ICH5 SATA host controller provides bits in the ***PCS*** register (***P0P*** and ***P1P***) that can be used by system software to detect the presence of (or lack of) SATA device(s) connected to the SATA host controller.

Device presence detection has two main benefits:

- Can **potentially** assist in quicker boot times (both BIOS POST and operating system) by eliminating the need for time-consuming device detection algorithms (as required for P-ATA devices). Limitations to this feature are addressed in the next section.
- Assists in providing better power management. A port with no device present can be disabled as a disabled SATA port consumes less power than one that is enabled (See Section: *4.3.1.1 Port Enabling/Disabling)*

### 4.4.1 Hardware and Software Considerations

The value of the port presence bits is valid only under the following conditions:

- The SATA host controller is in the D0 power state.
- The port to be checked is enabled.

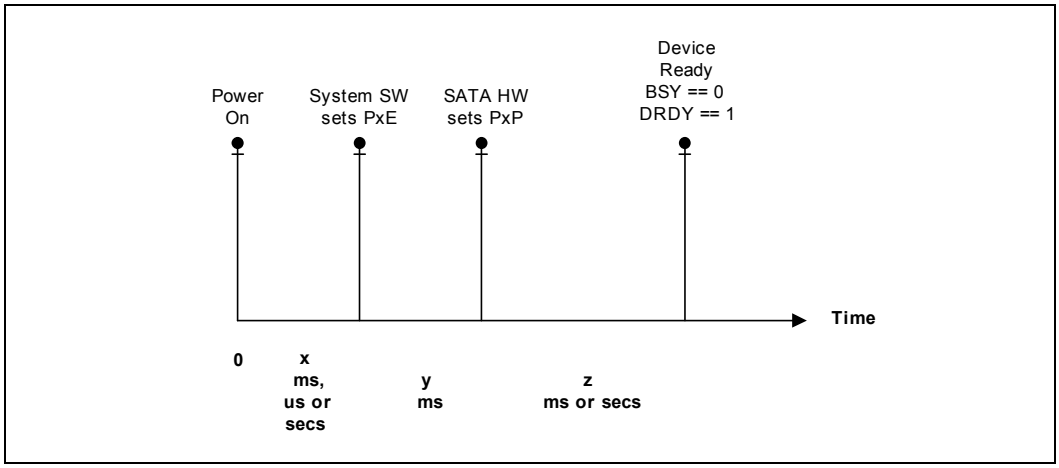
***Note:*** There is no requirement that both ports are enabled when checking the ***PxP*** bit; only the port being examined needs to be enabled.

At some point in time, after a port has been enabled (***PCS.PxE== 1)***, system software will want to determine two things:

1. Is a SATA device is attached to the port?
2. If a SATA device is attached to a port, is it ready to receive commands?

These two items can be determined by software in succession (detect for device, then check readiness) or simultaneously (check for readiness, as a ready device must be present). Device detection can be accomplished through an examination of a port's ***PxP*** bit (via polling). **However, the amount of time that may elapse between when system software enables a port and when the ICH5 SATA HBA detects a device (or not) is SATA device specific and is not specified.**

### Figure 9. Power-on to Device Ready Elapsed Time




In Figure 9, the total elapsed time from power on until a device attached to a SATA port is "ready." is the sum of the amount of elapsed time shown above (x + y + z). In the figure above, the time value '*x*' is the amount of time that elapses prior to the setting of the ***PxE*** bit by system software. This value may measure in microseconds, milliseconds or seconds – its value is dependent on when system software sets the PxE bit.

The time value '*y*' can be up to 10ms plus an unspecified amount of time (as per the *Serial ATA: High Speed Serialized AT Attachment v1.0* specification - this document is available from the Serial ATA Working Group at: http://www.serialata.org/). This duration is highly dependent on the manufacturer of the SATA device.

The time value '*z*' is bound to an upper value and is defined by the *AT Attachment with Packet Interface – 6 (ATA/ATAPI-6) Specification*, available from the T13 Technical Committee at: http://www.t13.org/.

If the ***PxP*** bit is not set by the ICH5 SATA PHY at some interval within the time denoted by x + y + z, then system software shall assume that no device is present on the port.

As noted in the figure above, the ***PxP*** bits only indicate device presence; device readiness (e.g., able to receive commands) shall be determined by an examination of the Status register (contained within the appropriate channel control register). The device indicates 'readiness' when the device is 1) Not busy (BSY == 0) and 2) is ready (DRDY == 1). This is identical to how device presence/readiness is determined for P-ATA devices.

The accuracy of the ***PxP*** bits cannot be guaranteed if the SATA PHY and associated device is in a slumber state. System software shall first disable the port and then re-enable it. This will cause the SATA PHY and associated device to wake, thus allowing the SATA hardware to properly detect and report the port connect status.

***Note:*** In the above figure (Figure 9), because the cumulative amount of time '*y*' is not bound to any specified value and can vary from device manufacturer to manufacturer. Because this time can very short (e.g., within 10 ms) or very long (10+ ms), system software may choose to exclusively use device "readiness" as an indicator of device presence instead of relying on the setting of ***PxP***. Selecting a mechanism for use by system software is beyond the scope of this paper.

### 4.4.2 Device Detection – Software Examples

System software may examine the port presence bits using two different methods:

1. Direct reads to the SATA host controller I/O space.
2. ACPI control methods.

The sample code in *A.3 Device Presence Check – Using I/O* illustrates how system software could directly read the port presence bits to determine the connect status of the SATA ports.

The sample code (for Windows WDM drivers) in *A.4 Device Presence Check – Using ACPI* illustrates how to execute an ACPI control method that returns the connection status for the SATA port(s) associated with a logical SATA channel. This code assumes that a PDO (*Physical Device Object*) exists for both the logical primary and secondary channels. WDM device drivers can only execute control methods associated with their corresponding _ADR object in ACPI namespace (of which there is a direct relationship between the _ADR object and the PDO).

Note that in these two examples, the ACPI control method handles the complexities of determining whether the SATA host controller is configured for Compatible mode and which SATA ports are the logical master and logical slave (if applicable). This sample driver code uses the ACPI sample code illustrated in *A.5 ACPI Control Method (GSPS)*.

## 4.5 ATA Swap Bay Support

While the ICH5 SATA host controller does not support surprise (removal of a SATA device while transactions are active on the SATA bus) device removals, it does provide basic ATA Swap Bay Support. Using the ***PCS*** register configuration bits (***PxE*** and ***PxP***) and power management flows, a device can be powered down by system software, and the port can then be powered off, allowing removal and insertion of a new device.

How system software handles device swapping is very dependent on the target platform. For example, mobile platforms may continue to use a combination of specialized hardware, interrupts and ACPI control methods (especially useful for swap bays that could support traditional P-ATA devices in addition to SATA devices) for communicating device insertions and removals to the operating system. Other platform types (e.g., desktop, workstations) traditionally do not implement additional hardware (interrupts) and ACPI control methods and may require that the operating system (device drivers) implement specific software based device detection algorithms. How this is implemented is beyond the scope of this document.

## 4.6 Implementing the Intel® ICH5 SATA Host Controller in ACPI Namespace

See *Appendix B – Example ACPI Namespace* for an example of an ACPI namespace for the ICH5 SATA host controller. This example supports the ICH5 SATA host controller operating in enhanced, combined and non-combined modes.

This page is intentionally left blank.

# Appendix A – Coding Examples

## A.1 Enabling/Disabling SATA Ports from a WDM Driver

```
.
.
.
//
// Function Proto for Reading and Writing to device configuration space
//
NTSTATUS ReadWriteConfigSpace( PDEVICE_OBJECT pDO, BOOLEAN fRead, PVOID pBuf,
                               ULONG Offset, ULONG Length);
//
//
// This example illustrates how a WDM driver could read/write the PCS.PxE bits for
// enabling or disabling the SATA port(s).
//
// This example assumes direct access to the SATA PCI Configuration space
//

//
// Disable the ports. For simplicity, we don't do any status checking
//
BYTE val = 0; // to disable both ports
ReadWriteConfigSpace( pDO, TRUE, &val, 0x92, sizeof( val));
.
.
.
//
// Enable the ports. For simplicity, we don't do any status checking
//
BYTE val = 3; // to enable both ports
ReadWriteConfigSpace( pDO, FALSE, &val, 0x92, sizeof( val));
.
.
.
//
// Enable only port 0. For simplicity, we don't do any status
// checking
//
BYTE val;
```

```
//
// need to perform a read/write modified in order to do this
// properly
// Read the current settings
ReadWriteConfigSpace( pDO, TRUE, &val, 0x92, sizeof( val));
val |= 1;  // set bit 0 to enable Port 0
ReadWriteConfigSpace( pDO, FALSE, &val, 0x92, sizeof( val));
.
.
.

//
// ReadWriteConfigSpace – Read or write PCI config space
//
// Entry:
//     pDO → ptr to our device object
//     fRead → TRUE if reading, FALSE if writing
//     pBuf → ptr to buffer that contains/receives the data
//     Offset → Offset into PCI config space to read/write
//     Length → size of pBuf
// Exit:
//     returns NT_SUCCESS if no errors.
//
NTSTATUS
ReadWriteConfigSpace( PDEVICE_OBJECT pDO, BOOLEAN fRead, PVOID pBuf, ULONG Offset,
                      ULONG Length)
{
   .
   .
   .
   KEVENT  event;
   PIRP  pIrp;
   NTSTATUS status;

   pIrp = IoAllocateIrp( pDO->StackSize, FALSE);
   if( pIrp == NULL)
      return STATUS_INSUFFICIENT_RESOURCES;
   KeInitializeEvent( &event, NotificationEvent, FALSE);
   // The PnP IRPs need the Status field initialized to STATUS_NOT_SUPPORTED.
   pIrp->IoStatus.Status = STATUS_NOT_SUPPORTED;
   nextStack = IoGetNextIrpStackLocation( pIrp);
   nextStack->MajorFunction= IRP_MJ_PNP;
   nextStack->MinorFunction= fRead ? IRP_MN_READ_CONFIG : IRP_MN_WRITE_CONFIG;
   nextStack->Parameters.ReadWriteConfig.WhichSpace = PCI_WHICHSPACE_CONFIG;
```

```
    nextStack->Parameters.ReadWriteConfig.Buffer = pBuf;
    nextStack->Parameters.ReadWriteConfig.Offset = Offset;
    nextStack->Parameters.ReadWriteConfig.Length = Length;
    status = IoCallDriver( pDO, irp);

    if( status == STATUS_PENDING) {
        //
        // Request did not complete. Need to wait until it does
        //
        KeWaitForSingleObject( &event, Suspended, KernelMode, FALSE, NULL);
        status = pIrp->IoStatus.Status;
    }
    return( status);
}
```

## A.2 Enabling/Disabling SATA Ports in the _PSx Control Method

```
//
// This sample ASL code demonstrates how to enable and disable the SATA port(s)
// from the _PS0 and _PS3 control methods
//
OperationRegion(IDEC,PCI_Config,0x90,3)
Field(IDEC,ByteAcc,NoLock,Preserve)
{
     MAP, 8, // SATA Map register – Offset 90h
        , 8, // skip 8 bits
     PCS, 8 // SATA Port status and control register – Offset 92h
}
Device( IDE1) {  // SATA controller
    Name(_ADR, 0x01f0002)  // Device 31, Function 2
    //
    // EPRT – Enable the SATA ports. This assumes the controller is in combined
    // mode. This would be need to modified to handle non-combined modes.
    // Entry:
    //  arg0 ➔ bit map indicating which port(s) to enable:
    //   bit 0 set, enable Port 0
    //   bit 1 set, enable Port 1
    //   all other bits must be zero
    //

    //
    // Since the ports operate independently, we can enable both
```

```
// simultaneously if necessary
//
Method( EPRT, 1) {
    Store( 1, Local0)           // Set max attempts
    Store( Arg0, Local1)
    While( LNotEqual( Local0, 0)){
        Or( Arg0, PCS, PCS)     // enable Port(s)
        Sleep( 500)             // Wait 500ms. Some devices respond
                                // very quickly, some longer. This loop will
                                // account for worse case. This is an
                                // example and could be better optimized
        Decrement( Local0)      // account for this attempt
        Store( PCS, Local2)     // fetch port presence bits
        // Check if we are enabling Port 0
        If( LAnd( Arg0, 0x01) {
            If( LAnd( Local2, 0x10)) {
                Decrement( Local1)    // Port 0 is enabled
            }
            Else {
               //
               // Since a device detect failed, we disable the port. This will
               // insure that the port remains disabled - this is not
               // required, but is part of a good power conservation policy.
               //
               And( PCS, 0x02, PCS)   // disable Port 0
            }
       }
        // Check if we are enabling Port 1
        If( LAnd( Arg0, 0x02) {
            If( LAnd( Local2, 0x20)) {
                Decrement( Local1)      // Port 1 is enabled
            }
            Else {
                And( PCS, 0x01, PCS)      // disable Port 1
            }
        }
        If( LEqual( Local1, Zero)) {  // all ports enabled?
           Store( Zero, Local0)      // terminate loop
        }
    } // end while
}
Device( PRID) {  // Primary channel
    Name( _ADR, 0)  // Logical primary channel (Port 0 or 1, BIOS selectable).
    // Assumes BIOS populated the SATA timing registers appropriately
```

```
 // depending on which port is configured as primary and secondary
  Device( DRV0)  // Logical primary master
{
   Name( _ADR, 0)
   …
}
//
// Handle transitions to D0 power state
//
Method(_PS0,0)
{
   //
   // make sure the OS drivers finds the ports in an enabled state as they
   // (the device drivers) may have been designed for P-ATA and 'know'
   // nothing about the PCS register
   //
   // Since enhance mode implements a master-master scheme, only 1 port
   // would be enabled here (dependent on the MAP settings). In Combined
   // mode, both SATA ports are viewed as a single logical channel
   // implementing a master-slave configuration in which case both ports are
   // enabled.
   //

   //
   // Enable Power to the device - Set the GPIO(s) bit corresponding to the
   // power plane control. This shall be done before
   // the port(s) are enabled. This is platform specific
   //
   …        // power plane control is platform specific
   // Must wait 30ms before we can enable the ports
   Sleep( 30)
   EPRT( 0x03)  // enable the ports – assumes combined mode
   //
   // Check ports and disable device power plane if port(s) not enabled.
   //
   …        // power plane control is platform specific
}
//
// Handle transitions to D3 power state
//
Method(_PS3,0)
{
   //
   // Disable the ports. Since a non-SATA aware driver could be in use,
```

```
        // we need to disable the SATA ports here
        //
        // Since enhance mode implements a master-master scheme, only 1 port
        // would be disabled here (dependent on the MAP settings). In Combined
        // mode, both SATA ports are viewed as a single logical channel
        // implementing a master-slave configuration in which case both ports are
        // disabled.
        //
        Store( Zero, PCS) // disable ports 0 and 1 – assume combined mode
        //
        // Disable Power to the device - Set the GPIO bit corresponding to the
        // power plane control
        //
        …   // platform specific
    }
}
Device( SECD) {  // Secondary channel
    Name( _ADR, 1)  // Logical secondary channel (Port 0 or 1, BIOS selectable)
    Device( DRV0) // Logical secondary master
    {
        Name(_ADR, 0)
        …
    }
    //
    // Handle transitions to D0 power state
    //
    Method(_PS0,0)
    {
        //
        // make sure the OS drivers finds the ports in an enabled state as they
        // (the device drivers) may have been designed for P-ATA and 'know'
        // nothing about the PCS register
        //
        // Since enhance mode implements a master-master scheme, only 1 port
        // would be enabled here (dependent on the MAP settings). In Combined
        // mode, both SATA ports are viewed as a single logical channel
        // implementing a master-slave configuration in which case both ports
        // are enabled.
        //

        //
        // Enable Power to the device - Set the GPIO(s) bit corresponding to the
        // power plane control. This shall be done before
        // the port(s) are enabled. This is platform specific
```

```
            //
             …        // power plane control is platform specific
            // Must wait 30ms before we can enable the ports
            Sleep( 30)
            EPRT( 0x03)   // enable the ports, assumes combined mode
            //
            // Check ports and disable device power plane if port(s) not enabled.
            //
            …       // power plane control is platform specific
        }
        //
        // Handle transitions to D3 power state
        //
        Method(_PS3,0)
        {
            //
            // Disable the ports. Since a non-SATA aware driver could be in use,
            // we need to disable the SATA ports here
            //
            // Since enhance mode implements a master-master scheme, only 1 port
            // would be disabled here (dependent on the MAP settings). In Combined
            // mode, both SATA ports are viewed as a single logical channel
            // implementing a master-slave configuration in which case both ports
            // are disabled.
            //
            Store( Zero, PCS) // disable ports 0 and 1 – assume combined mode.
           //
           // Disable Power to the device - Set the GPIO bit corresponding to the
           // power plane control, platform specific
           //
            …
        }
    }
} // Device( IDE1)
```

## A.3 Device Presence Check – Using I/O

```
//
// This sample code illustrates how system software could directly read the port
// presence (PxP) bits to determine the connect status of the SATA ports.
//
//  Note: This sample code does not apply to any specific operating system. As
// such, the function, OS_ReadPCI() is a hypothetical, OS provided function that
// allows a device's PCI configuration space to be read.
```

```
//
BYTE bMAP = OS_ReadPCI( 0x90); // read the map register
//
// It is assumed that the port is enabled and the device and PHY are not in a
// slumber state as this is required in order for the port presence bits to be
// accurate.
//
BYTE bPCS = OS_ReadPCI( 0x92); // read the port/status control register
//
// check the status of port 0
//
INT iPort0Status, iPort1Status, iAllStatus = 0;
//
// make sure the P-ATA is not the primary channel
//
if( bMAP < 0x06) {
   //
   // Not P-ATA device. Must be SATA.
   //
   if( bPCS & 0x10)
      iPort0Status = 1;  // Port 0 device present
   if( bPCS & 0x20)
      iPort1Status = 1; // Port 1 device present
   // Now we need to figure out which is master and slave (if applicable)
   if( !(bMAP & 0x01)) {  // is Port 0 master?
      // Port 0 is master
      iAllStatus = iPort0Status;
      // check if combined mode
     if( bMAP & 0x04)  // Is combined mode?
     // yes, set the slave device status (Port 1)
        iAllStatus |= iPort1Status << 1;
   }
    else {
     // Port 1 is master
     iAllStatus = iPort1Status;
     if( bMAP & 0x04) // Is combined mode?
        // yes, set the slave device status (Port 0)
        iAllStatus |= iPort0Status << 1;
   }
 }
 else
   // P-ATA does not support this. Indicate connect status unknown
   iAllStatus = -1;
 //
```

```
// At this point, iAllStatus will contain:
// 0 ==> no devices present
// 1 ==> master device present
// 2 ==> slave device present
// -1 ==> Not supported or unknown (P-ATA)
//
return( iAllStatus); // done
```

## A.4 Device Presence Check – Using ACPI

```
//
// The following sample code (for Windows WDM drivers) can be used to execute a
// control method that will return the status of the SATA port(s) associated with
// a SATA channel. It also illustrates how to execute an ACPI control method that
// returns the connection status for a logical channel. This code assumes that a
// PDO (Physical Device Object) exists for both the logical primary and secondary
// channels. WDM device drivers can only execute control methods associated with
// their corresponding _ADR object in ACPI namespace (of which there is a direct
// relationship between the _ADR object and the PDO).
// This sample driver code uses the ACPI sample code illustrated
// in A.5 ACPI Control Method (GSPS).
//
// Function Protos
NTSTATUS SATA_CheckPortStatus( IN PDEVICE_OBJECT pDO, PDWORD pdwStatus);
NTSTATUS SendIRP( IN PDEVICE_OBJECT Pdo, IN ULONG Ioctl, IN PVOID InputBuffer,
                  IN ULONG InputSize, IN PACPI_EVAL_OUTPUT_BUFFER OutputBuffer,
                  IN ULONG OutputSize);
VOID ProcessConnectStatus( DWORD dwPortStatus);
{
      .
      .
      .
      NT_STATUS status;
      DWORD dwPortStatus;
      Status = SATA_CheckPortStatus( pDO, &dwPortStatus);
      if( !NT_SUCCESS( status))
          //
          // Not an ACPI OS or control method not supported. Set the
          // status to indicate that port status is unknown
          //
          dwPortStatus = -1;
      ProcessConnectStatus( dwPortStatus);  // TODO – implementation specific
```

```
        .
        .
        .
}
#define SATA_GET_PORT_STATUS 'SPSG' // Control method to execute
//
// SATA_CheckPortStatus - Get the SATA port status
//
// Entry:
//    pDO   ==> pointer to our physical device object
//    pdwStatus ==> ptr to receive port status value
//
// Exit:
//    Returns status code.
//    If success, pdwStatus has the SATA port status:
//     0 ==> No devices connected
//     1 ==> Master device present
//     2 ==> Slave device present
//     3 ==> Master and slave device present
//    -1 ==> Device presence unknown
//
NTSTATUS
SATA_CheckPortStatus( IN PDEVICE_OBJECT pDO, PDWORD pdwStatus)
{
    NTSTATUS status = STATUS_SUCCESS;
    PACPI_METHOD_ARGUMENT pArgBuf;
    ACPI_EVAL_INPUT_BUFFER InputBuffer;
    ACPI_EVAL_OUTPUT_BUFFER OutputBuffer;

    //
    // setup the request structure
    //
    InputBuffer->MethodNameAsUlong = SATA_GET_PORT_STATUS;
    InputBuffer->Signature = ACPI_EVAL_INPUT_BUFFER_SIGNATURE;
    //
    // finally, do the request
    //
    status = SendIRP( pDO,              // this logical channel's PDO
                      IOCTL_ACPI_EVAL_METHOD,
                      pInputBuffer,     // ptr to input buffer
                      sizeof( ACPI_EVAL_INPUT_BUFFER), // its size
                      pOutputBuffer,    // ptr to output buffer
                      sizeof( ACPI_EVAL_OUTPUT_BUFFER); // its size
    //
```

```
    // if not successful, the control method probably does not
    // exists or APM mode is being used
    //
    if( NT_SUCCESS( status)) {
       //
       // make sure we received the correct data type back
       //
       if( pOutputBuffer->Argument[ 0].Type != ACPI_METHOD_ARGUMENT_INTEGER)
          status = STATUS_DATA_ERROR;
       else
          // Port status is here
          *pdwStatus = pOutputBuffer->Argument[ 0].Argument;
    }
    return( status);
}
//
// SendIrp - Send Irp to ACPI
//
// Entry:
//    Pdo   ==> target of the request
//    Ioctl  ==> the request
//    InputBuffer  ==> ptr to input parameters
//    InputSize  ==> size of InputBuffer
//    OutputBuffer  ==> ptr to receive results
//    OutputSize  ==> size of OutputBuffer
//
// Exit:
//    Returns status code.
//    OutputBuffer <== indeterminent
//
NTSTATUS
SendIRP( IN PDEVICE_OBJECT Pdo, IN ULONG Ioctl, IN PVOID InputBuffer,
         IN ULONG InputSize, IN PACPI_EVAL_OUTPUT_BUFFER OutputBuffer,
         IN ULONG OutputSize)
{
    IO_STATUS_BLOCK ioBlock;
    KEVENT evIoctl;
    NTSTATUS status;
    PIRP Irp;
    PIO_STACK_LOCATION  pIrpStack;

    // Initialize an event to wait upon
    KeInitializeEvent( &evIoctl, SynchronizationEvent, FALSE);
    // Build the request
```

```
    Irp = IoBuildDeviceIoControlRequest( Ioctl, Pdo, InputBuffer, InputSize,
                                         OutputBuffer, OutputSize, FALSE,
                                        &evIoctl, &ioBlock);
    if( !Irp)
       status = STATUS_INSUFFICIENT_RESOURCES;
    else {
       pIrpStack = IoGetNextIrpStackLocation( Irp);
       pIrpStack->MajorFunction = IRP_MJ_DEVICE_CONTROL;
       // Pass the request to the Pdo, always wait for the completion
       // routine
       status = IoCallDriver( Pdo, Irp);
       if( status == STATUS_PENDING)
          // Wait for the IRP to be completed, then grab the real status code
       KeWaitForSingleObject( &evIoctl, Executive, KernelMode, FALSE, NULL);
       status = ioBlock.Status;
       // Sanity check the data
       if( OutputBuffer != NULL) {
          if( OutputBuffer->Signature != ACPI_EVAL_OUTPUT_BUFFER_SIGNATURE
          || OutputBuffer->Count == 0)
          status = STATUS_DATA_ERROR;
       }
    }
    return( status);
}
```

## A.5 ACPI Control Method (GSPS)

```
//
// Define where the SATA MAP and Status/Control registers reside in PCI Config
// space
//
OperationRegion(IDEC,PCI_Config,0x90,3)
Field(IDEC,ByteAcc,NoLock,Preserve)
{
   MAP, 8, // SATA Map register - Offset 90h
       , 8, // skip 8 bits
   PCS, 8 // SATA Port status and control register - Offset 92h
}
Device( IDE1) {  // SATA controller
   Name(_ADR, 0x01f0002)  // Device 31, Function 2
   Device( PRID) {  // Primary channel
      Name( _ADR, 0)  // Logical primary channel (Port 0 or 1, BIOS selectable).
      // Assumes BIOS populated the SATA timing registers appropriately
      // depending on which port is configured as primary and secondary
      //
      // This control method determines if device(s) are present on the logical
      // primary channel.  Because this control method is shared with P-ATA
      // devices and because P-ATA does not
      // have equivalent registers, this control will indicate a status of
      // unknown (P-ATA only)
      //
      // Entry:
      //     Nothing
      // Exit:
      //     DWORD value representing the channel status:
      //           0 ==> no devices present
      //           1 ==> master device present
      //           2 ==> slave device present
      //          -1 ==> Not supported or unknown (P-ATA)
      //
      Method( GSPS, 0) // Get the port status for the device(s)
      {
        // need to figure out which port(s) to look at (e.g. are we in a
        // master-master configuration or master-slave)
        //
        Store( Zero, Local0)
```

```
Store( MAP, Local1)
Store( PCS, Local2)
// make sure the P-ATA is not the primary channel
If( LLess( Local1, 0x06))
{
   // Not P-ATA device. Must be SATA.
   Store( Zero, Local3)
   Store( Zero, Local4)
   Store( Zero, Local5)
   If( LAnd( Local2, 0x10))
   {
      Store( One, Local3) // Port 0 device present
    }
   If( LAnd( Local2, 0x20))
   {
      Store( One, Local4) // Port 1 device present
   }
   // Now we need to figure out which is master and slave (if
   // applicable)
   If( LNot( LAnd( Local1, 0x01))) // is Port 0 master?
   {
      // Port 0 is master
      Store( Local3, Local5)
      If( LAnd( Local1, 0x04)) // Is combined mode?
      {
         // yes, set the slave device status (Port 1)
         Or( ShiftLeft( Local4, 1, Local4), Local5)
      }
   }
   Else
   {
      // Port 1 is master
      Store( Local4, Local5)
      If( LAnd( Local1, 0x04)) // Is combined mode?
      {
          // yes, set the slave device status (Port 0)
          Or( ShiftLeft( Local3, 1, Local3), Local5)
      }
   }
   Return( LocaL5)
}
Else
{
   // P-ATA is primary. Indicate device status unknown
```

```
            return( 0xffffffff)
         }
      }
      Device( DRV0)  // Logical primary master
      {
         Name( _ADR, 0)
      }
   }
   Device( SECD)   // Secondary channel
   {
      Method( GSPS, 0) // Get the port status
      {
         // Similar to GSPS for PRID
      }
      Name( _ADR, 1)  // Logical secondary channel (Port 0 or 1, BIOS selectable)
      Device( DRV0)  // Logical secondary master
      {
         Name(_ADR, 0)
      }
   }
}
```

This page is intentionally left blank.

# Appendix B – Example ACPI Namespace

```
//
// The following illustrates the sample ASL code for a Combined
// and non-combined mode configuration.
//
// The SATA controller supports several configurations. Combined
// mode and non-Combined mode. If in non-Combined mode (P-ATA and
// SATA are separate PCI functions), then:
//    Port 0 == logical primary master
//    Port 1 == logical secondary master
//           or (optionally selectable via BIOS setup)
//    Port 0 == logical secondary master
//    Port 1 == logical primary master
//
// Note that a separate P-ATA device namespace is required in the
// event that non-Combined mode is used.
//
// If in Combined mode (P-ATA and SATA combined as a single PCI
// function) then:
// Assumes SATA Port 0 and Port 1 are a single logical channel
// (primary/secondary BIOS selectable)
//
// Assumes P-ATA (physical primary or secondary channel) is a
// logical channel.(primary/secondary BIOS selectable).
//
// Possible Combined mode configurations:
//      Port 0 == primary master
//      Port 1 == primary slave
//      P-ATA  == Secondary master/slave
//             or
//      Port 0 == primary slave
//      Port 1 == primary master
//      P-ATA  == secondary master/slave
//             or
//     P-ATA  == primary master/slave
//     Port 0 == secondary master
//     Port 1 == secondary slave
```

```
//            or
//     P-ATA  == primary master/slave
//     Port 0 == secondary slave
//     Port 1 == secondary master
//
OperationRegion(IDEC,PCI_Config,0x90,3)
Field(IDEC,ByteAcc,NoLock,Preserve)
{
    MAP, 8, // SATA Map register - Offset 90h
       , 8, // Skip 8 bits
    PCS, 8  // SATA Port status and control register
}
Device( IDE1) {   // SATA controller
Name(_ADR, 0x01f0002)  // Device 31, Function 2
   //
   // EPRT – Enable the SATA ports. This assumes the controller is in combined
   // mode. This would be need to modified to handle non-combined modes.
   // Entry:
   //  arg0 ➔ bit map indicating which port(s) to enable:
   //   bit 0 set, enable Port 0
   //   bit 1 set, enable Port 1
   //   all other bits must be zero
   //

   //
   // Since the ports operate independently, we can enable both simultaneously
   //
   Method( EPRT, 1) {
      Store( 1, Local0)           // Set max attempts
      Store( Arg0, Local1)
      While( LNotEqual( Local0, 0)){
         Or( Arg0, PCS, PCS)      // enable Port(s)
         Sleep( 500)              // Wait 500ms. Some devices respond
                                  // very quickly, some longer. This loop will
                                  // account for worse case. This is an
                                  // example and could be better optimized
         Decrement( Local0)       // account for this attempt
         Store( PCS, Local2)      // fetch port presence bits
         // Check if we are enabling Port 0
         If( LAnd( Arg0, 0x01) {
            If( LAnd( Local2, 0x10)) {
               Decrement( Local1)    // Port 0 is enabled
            }
            Else {
```

```
                //
                // Since a device detect failed, we disable the port. This is not
                // required, but is part of a good power conservation policy.
                //
                And( PCS, 0x02, PCS)   // disable Port 0
            }
        }
        // Check if we are enabling Port 1
        If( LAnd( Arg0, 0x02) {
            If( LAnd( Local2, 0x20)) {
                Decrement( Local1)      // Port 1 is enabled
            }
            Else {
                And( PCS, 0x01, PCS)      // disable Port 1
            }
        }
        If( LEqual( Local1, Zero)) {  // all ports enabled?
           Store( Zero, Local0)      // terminate loop
        }
    } // end while
}

//
// CTYP - This method determines the type of device being
// managed by the specified logical channel
//
// Entry:
//  Arg0 ==> specifies which channel is to be checked (0 =
//    Primary, 1 = Secondary)
// Exit:
//  Returns 0 if the channel is hosting P-ATA device(s)
//  Returns 1 if the primary channel is hosting SATA device(s)
// (Combined mode)
//  Returns 2 if the secondary channel is hosting SATA
//  device(s) (Combined Mode)
//  Returns 3 if port 0 is primary master SATA device (non-
// Combined mode)
//  Returns 4 if port 1 is primary master SATA device (non-
// Combined)
//  Returns 5 if port 0 is secondary master SATA device(s)
//(non-Combined Mode)
//  Returns 6 if port 1 is secondary master SATA device(s)
//(non-Combined Mode)
//
```

```
Method( CTYP, 1)
{
    Store( Zero, Local0)
  If( Arg0)
   {
        //
        // Check the Primary Channel
        //
        // Check if combined mode and if this device is a P-ATA
       // device MAP == 100b or 101b Combined mode, P-ATA is
       // secondary
       //
        If( LAnd( LGreater( MAP, 0x1), LLess( MAP, 0x6)))
        {
            Store( 0x1, Local0)  // SATA is primary, combined
        }
        Else {
            If( LEqual( MAP, Zero))
            {
                Store( 3, Local0)// port 0 is primary master
            }
            If( LEqual( MAP, One)
            {
                Store( 4, Local0)// port 1 is primary master
           }
        }
    }
    Else {
        //
        // Check the secondary Channel
        //
        // Check if combined mode and if this device is a P-ATA
       // device MAP == 110b or 111b Combined mode, P-ATA is
       // primary
        //
        If( LGreater( MAP, 0x5))
        {
            Store( 0x2, Local0) // SATA is Secondary, combined
        }
        Else {
            If( LEqual( MAP, Zero))
            {
                Store( 5, Local0)// port 0 is secondary master
            }
```

```
            If( LEqual( MAP, One)
            {
                Store( 6, Local0)// port 1 is secondary master
            }
        }
    }
     Return( Local0)
}
//
// Logical Primary channel
//
// Physical SATA Port 0 == logical primary master
//                              or
// Physical SATA Port 1 == logical primary master
//
// In Combined mode, the following must be supported by PRID:
//  Physical SATA Port 0 == logical primary master
//  Physical SATA Port 1 == logical primary slave
//                               or
//  Physical SATA Port 0 == logical primary slave
//  Physical SATA Port 1 == logical primary master
//                               or
//  P-ATA == Primary master/slave
//
Device( PRID) {
    Name( _ADR, 0)  // Logical primary channel (Port 0/1, BIOS
                    // selectable or P_ATA)
    Method( _GTM) {}// similar to current P-ATA implementations
    Method( _STM, 3) {} // similar to current P-ATA implementations
    //
    // Like the _GTF methods, these control methods may require
    // additional checks as the power sequences (registers, etc)
    // may be different for P-ATA and SATA channels.
    //
    Method(_PS0,0)
    {
        // Handle PM duties based on device type
        Store( CTYP( 0), Local0)
        If( Local0) {
            //
            // Not P-ATA device. Must be SATA
            // make sure the OS drivers finds the ports in an
           // enabled state as they (the device drivers) may have
           // been designed for P-ATA and 'know' nothing about the
```

```
            // PCS register
             //
             // Since Enhance mode implements a master-master
            // scheme, only 1 port would be enabled here (dependent
            // on the MAP settings). In Combined mode, both SATA
            // ports are viewed as a single logical channel
            // implementing a master-slave configuration in which
            // case both ports are enabled.
            //
            Store( Zero, Local1)
            If( LEqual( Local0, 1))
             {
                 ...          // Enable Power to the device, platform specific
                 Sleep( 30)  // power must be applied for at least 30ms
               Store( 0x03, Local1)  // is combined, enable both ports
             }
             If( LEqual( Local0, 3)
             {
                 ...          // Enable Power to the device, platform specific
                 Sleep( 30)  // power must be applied for at least 30ms
                 Store( 0x01, Local1)   // only enable port 0
             }
             If( LEqual( Local0, 4)
             {
                 ...          // Enable Power to the device, platform specific
                 Sleep( 30)  // power must be applied for at least 30ms
                 Store( 0x02, Local1)  // only enable port 1
             }
             EPRT( Local1)      // enable the Port(s)
             ...               // Disable Power to the device(s) if the port(s)
                              // were left disabled, platform specific
        }
        Else
        {
            // Is Combined mode and is a P-ATA device
           ....
        }
    }
    Method(_PS3,0)
    {
        // Handle PM duties based on device type
       Store( CTYP( 0), Local0)
       If( Local0)
       {
```

```
            //
            // Not P-ATA device. Must be SATA
            // make sure the OS drivers finds the ports in an enabled
           // state as they (the device drivers) may have been
           // designed for P-ATA and 'know' nothing about the PCS
           // register
           //
           // Since Enhance mode implements a master-master scheme,
           // only 1 port would be disabled here (dependent on the
           // MAP settings). In Combined mode, both SATA ports are
           // viewed as a single logical channel implementing a
           // master-slave configuration in which case both ports
           // are disabled.
           //
           If( LEqual( Local0, 1))
            {
                Store( 0x0, PCS)// is combined, disable both ports
            }
            If( LEqual( Local0, 3)
            {
                NAnd( PCS, 0x01, PCS) // only disable port 0
            }
            If( LEqual( Local0, 4)
            {
                NAnd( PCS, 0x02, PCS) // only disable port 1
            }
            // Disable Power to the device - Set the GPIO bit
           // corresponding to the power plane control, platform
           // specific
           //
            ...   // platform specific
        }
       Else
       {
           // Is Combined mode and is a P-ATA device
           ....
       }
   }
   Device( DRV0)  // Logical primary master (SATA Port 0/1, or P-ATA device 0)
   {
      Name( _ADR, 0)
      //
      // similar to current P-ATA implementations. Since this
     // device node can represent either a P-ATA device or a SATA
```

```
      // device, a check may need to be added since the task file
      // could be different.
      //
     Method( _GTF,0)
      {
          // return task file info based on device type
        If( CTYP( 0))
        {
            // Is Combined mode and is a P-ATA device
           ....
        }
        Else
        {
            // Not P-ATA device. Must be SATA
           ....
        }
      }
   }
//*****************************************************************
// DRV1 is only accessed when configured for Combined mode. In
// non-Combined mode SATA devices use a master-master
// arrangement.
//
Device( DRV1)  // Logical primary slave (SATA Port 0/1, or P-ATA device 1)
{
    Name( _ADR, 1)
     //
     // similar to current P-ATA implementations. Since this device
     // node can represent either a P-ATA device or a SATA device,
     // a check may need to be added since the task file could be
     // different.
     //
     Method( _GTF,0)
     {
         // return task file info based on device type
       If( CTYP( 0))
       {
           // Is Combined mode and is a P-ATA device
          ....
       }
       Else
       {
           // Not P-ATA device. Must be SATA
          ....
```

```
    }
   }
 }
 //*****************************************************
}
//
// Logical Secondary channel
//
// Physical SATA Port 0 == logical secondary master
//                              or
// Physical SATA Port 1 == logical secondary master
//
// In Combined mode, the following must be supported by SECD:
//  Physical SATA Port 0 == logical secondary master
//  Physical SATA Port 1 == logical secondary slave
//                               or
//  Physical SATA Port 0 == logical secondary slave
//  Physical SATA Port 1 == logical secondary master
//                               or
//  P-ATA == logical secondary master/slave
//
 //
 Device( SECD) {
    Name( _ADR, 1)
    Method( _GTM) {}// similar to current P-ATA implementations
    Method( _STM, 3) {} // similar to current P-ATA implementations
    //
    // Like the _GTF methods, these control methods may require
    // additional checks as the power sequences (registers, etc)
    // may be different for P-ATA and SATA channels.
    //
   Method(_PS0,0)
    {
        // Handle PM duties based on device type
        Store( CTYP( 1), Local0)
       If( Local0)
       {
           //
           // Not P-ATA device. Must be SATA
           // make sure the OS drivers finds the ports in an enabled
          // state as they (the device drivers) may have been
          // designed for P-ATA and 'know' nothing about the PCS
          // register
          //
```

```
        // Since Enhance mode implements a master-master scheme,
       // only 1 port would be enabled here (dependent on the
       // MAP settings). In Combined mode, both SATA ports are
       // viewed as a single logical channel implementing a
       // master-slave configuration in which case both ports
       // are enabled.
        //
        If( LEqual( Local0, 2))
        {
            ...          // Enable Power to the device, platform specific
            Sleep( 30)  // power must be applied for at least 30ms
            Store( Local1, 0x03, Local1)  // is combined, enable both ports
        }
        If( LEqual( Local0, 5)
        {
            ...          // Enable Power to the device, platform specific
            Sleep( 30)  // power must be applied for at least 30ms
            Store( 0x01, Local1)  // only enable port 0
       }
        If( LEqual( Local0, 6)
        {
            ...          // Enable Power to the device, platform specific
            Sleep( 30)  // power must be applied for at least 30ms
            Store( 0x02, Local1() // only enable port 1
        }
        EPRT( Local1)    // enable the port(s)
        ...              // Disable Power to the device(s) if the port(s)
                        // were left disabled, platform specific
    }
    Else
    {
        // Is Combined mode and is a P-ATA device
        ....
    }
}
Method(_PS3,0)
{
    // Handle PM duties based on device type
    Store( CTYP( 0), Local0)
    If( Local0)
    {
        //
        // Not P-ATA device. Must be SATA
        // make sure the OS drivers finds the ports in an
```

```
            // enabled state as they (the device drivers) may have
            // been designed for P-ATA and 'know' nothing about the
            // PCS register
            //
             // Since Enhance mode implements a master-master scheme,
            // only 1 port would be disabled here (dependent on the
            // MAP settings). In Combined mode, both SATA ports are
            // viewed as a single logical channel implementing a
            // master-slave configuration in which case both ports

            // are disabled.
            //
             If( LEqual( Local0, 2))
             {
                 Store( 0x0, PCS)      // is combined, disable both ports
             }
             If( LEqual( Local0, 5)
             {
                 NAnd( PCS, 0x01, PCS) // only disable port 0
             }
             If( LEqual( Local0, 6)
             {
                 NAnd( PCS, 0x02, PCS) // only disable port 1
             }
             // Disable Power to the device - Set the GPIO bit
            // corresponding to the power plane control, platform
            // specific
             //
             ... // platform specific
        }
        Else
        {
            // Is Combined mode and is a P-ATA device
           ....
        }
    }
Device( DRV0)  // Logical secondary master (SATA Port 0/1, or P-ATA device 0)
{
    Name( _ADR, 0)
   //
   // similar to current P-ATA implementations. Since this
   // device node can represent either a P-ATA device or a SATA
   // device, a check may need to be added since the task file
   // could be different.
```

```
    //
    Method( _GTF,0)
     {
        // return task file info based on device type
       If( CTYP( 1))
       {
           // Is Combined mode and is a P-ATA device
          ....
       }
       Else
       {
           // Not P-ATA device. Must be SATA
          ....
       }
   }
}
//*******************************************************
// DRV1 is only accessed when configured for Combined mode. In
// non-Combined mode SATA devices use a master-master
// arrangement.
//
Device( DRV1)  // Logical secondary slave (SATA Port 0/1, or P-ATA device 1)
{
     Name( _ADR, 1)
     //
     // similar to current P-ATA implementations. Since this
     // device node can represent either a P-ATA device or a SATA
     // device, a check may need to be added since the task file
     // could be different.
     //
     Method( _GTF,0)
     {
         // return task file info based on device type
        If( CTYP( 1))
        {
            // Is Combined mode and is a P-ATA device
           ....
        }
        Else
        {
            // Not P-ATA device. Must be SATA
           ....
        }
   }
}
```

```
  //*****************************************************
 }
}
```