



Intel® 82801BA I/O Controller Hub (ICH2) AC '97

Programmer's Reference Manual (PRM)

June 2000

Order Number: 298238-001





Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products, including liability or warranties relating to fitness for a particular purpose, merchantability or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, lifesaving, or life-sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Intel® 82801BA (ICH2) I/O Controller Hub AC '97 may contain design defects or errors known as errata that may cause the product to deviate from published specifications. Current characterized errata are available upon request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

I²C is a 2-wire communications bus/protocol developed by Philips*. SMBus, a subset of the I²C bus/protocol, was developed by Intel. Implementations of the I²C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Alert on LAN is a result of the Intel-IBM Advanced Manageability Alliance and is a trademark of IBM*.

Copies of documents that have an order number and are referenced in this document or other Intel literature may be obtained from:

Intel Corporation

URL: www.intel.com

Phone: 1-800-548-4725

*Third-party brands and names are the property of their respective owners.

Copyright © Intel Corporation 1999, 2000

Contents

1.	Introduction	7
1.1.	Reference Documents and Information Sources	8
2.	AC '97 Controller's Theory of Operation	9
2.1.	AC'97 Initialization	9
2.1.1.	System Reset	9
2.1.2.	Codec Topology	10
2.1.3.	BIOS PCI Configuration	11
2.1.4.	Hardware Interrupt Routing	12
2.2.	DMA Engines	12
2.2.1.	Buffer Descriptor List	13
2.2.2.	DMA Initialization	14
2.2.3.	DMA Steady-State Operation	15
2.2.4.	Stopping Transfers	17
2.2.5.	FIFO Error Conditions	17
2.2.5.1.	FIFO Underrun	17
2.2.5.2.	FIFO Overrun	18
2.3.	Arbitration	18
2.4.	Data Buffers	18
2.4.1.	Memory Organization of Data	18
2.4.2.	FIFO Organization	18
2.5.	Multiple Codec/Driver Support	20
2.5.1.	Codec Register Read	20
2.5.2.	Codec Access Synchronization	21
2.5.3.	Data Request Synchronization in Audio Split Configurations	21
2.6.	Power Management	21
2.6.1.	Power Management Transition Maps	23
2.6.2.	Topology Detection	26
2.6.2.1.	Determining the Presence of a Secondary Codec	26
2.6.2.2.	Determining the Presence of a Modem Function	26
2.6.3.	Aggressive Power Management	26
2.6.3.1.	Primary Audio Requested to Transition to D3 State	27
2.6.3.2.	Secondary Modem Requested to Transition to D3 State	27
2.6.3.3.	Secondary Modem Requested to Transition to D0 State	28
2.6.3.4.	Audio Primary Requested to Transition to D0 State	28
2.6.3.5.	Using a Cold or Warm Reset	29
3.	Surround Audio Support	31
3.1.	Determine Codec's Audio Channels	31
3.2.	Enabling AC' 97 Controller Audio Channels	32
4.	AC '97 Audio Driver	35
4.1.	Win32 Driver Model	35
4.2.	Example Driver	36
4.3.	Plug and Play	39
4.4.	Power Management	39
4.4.1.	IAdapterPowerManagement	39
4.4.2.	IPowerNotify	40



5.	AC '97 Modem Driver	41
5.1.	Robust Host Based Generation of a Synchronous Data Stream	41
5.1.1.	Spurious Data Algorithm	42
5.1.2.	AC' 97 Spurious Data Implementation.....	42
6.	Appendix A: System BIOS Codec/Function Detection Algorithm	45
7.	Appendix B: Detail for AC '97 Controller Wake-Up Detection Circuitry.....	47

Figures

Figure 1-1.	Block Diagram of Intel® 8XX Chipset with ICH2 Component	7
Figure 1-2.	AC '97 Controller Connection to Its Companion Codec	8
Figure 2-1.	Possible Codec Configurations	10
Figure 2-2.	Generic Form of Buffer Descriptor (One Entry in the List)	13
Figure 2-3.	Buffer Descriptor List	14
Figure 2-4.	Compatible Implementation with Left and Right Sample Pair in Slots 3 and 4 Every Frame	19
Figure 2-5.	Compatible Implementation with Sample Rate Conversion Slots 3 and 4 Alternating over Next Frame	19
Figure 2-6.	Incompatible Implementation of Sample Rate Conversion with Repeating Slots over Next Frames	19
Figure 4-1.	WDM Audio Driver Hierarchy	35
Figure 4-2.	Driver Object Topology	36
Figure 4-3.	Hardware Initialization Call Sequence	37
Figure 4-4.	Stream Object Overview	38
Figure 4-5.	ICH2 Power Transition Process	40
Figure 7-1.	Wake-Up Circuitry Representation	47

Tables

Table 2-1.	Audio Registers (Device 31 Function 5 Audio)	11
Table 2-2.	Modem Registers (Device 31 Function 6 Modem)	12
Table 2-3.	BD Buffer Pointer (DWord 0: 00–03h)	13
Table 2-4.	D Control and Length (DWord 1: 04–07h)	13
Table 2-5.	Audio Descriptor List Base Address	14
Table 2-6.	Modem Descriptor List Base Address	15
Table 2-7.	Audio Last Valid Index	15
Table 2-8.	Modem Last Valid Index	15
Table 2-9.	FIFO Summary	20
Table 2-10.	Codec Topologies	22
Table 2-11.	Power State Mapping for Audio Single-Codec Desktop Transition	24
Table 2-12.	Power State Mapping for Modem Single-Codec Desktop Transition	24
Table 2-13.	Power State Mapping for Audio in Dual-Codec Desktop Transition	25
Table 2-14.	Power State Mapping for Modem in Dual-Codec Desktop Transition	25
Table 3-1.	Audio Codec Extended Audio ID Register	31
Table 3-2.	Multiple codec audio channel distribution	32
Table 3-3.	PCM 4/6 -channels capability bits	32
Table 3-4.	PCM 4/6 -channels enable bits	33
Table 7-1.	Wake-Up Condition Table for AC/MC Configuration	48
Table 7-2.	Wake-Up Condition Table for AMC Configuration	48
Table 7-3.	Wake-Up Condition Table for Single AC or MC Configuration	48



Revision History

Rev.	Draft/Changes	Date
-001	Initial Release	June 2000

1. Introduction

This document assists Independent Hardware Vendors (IHV) in supporting the feature set of the Intel® 82801BA I/O Controller Hub (ICH) AC '97 Digital Controller. General requirements for developing an audio mini-port driver that utilizes the AC '97 audio interface are described. The information in this document supplements the information provided in the *Intel® 82801BA (ICH2) I/O Controller Hub Datasheet*, and is intended for IHVs and Intel customers developing their own driver interface.

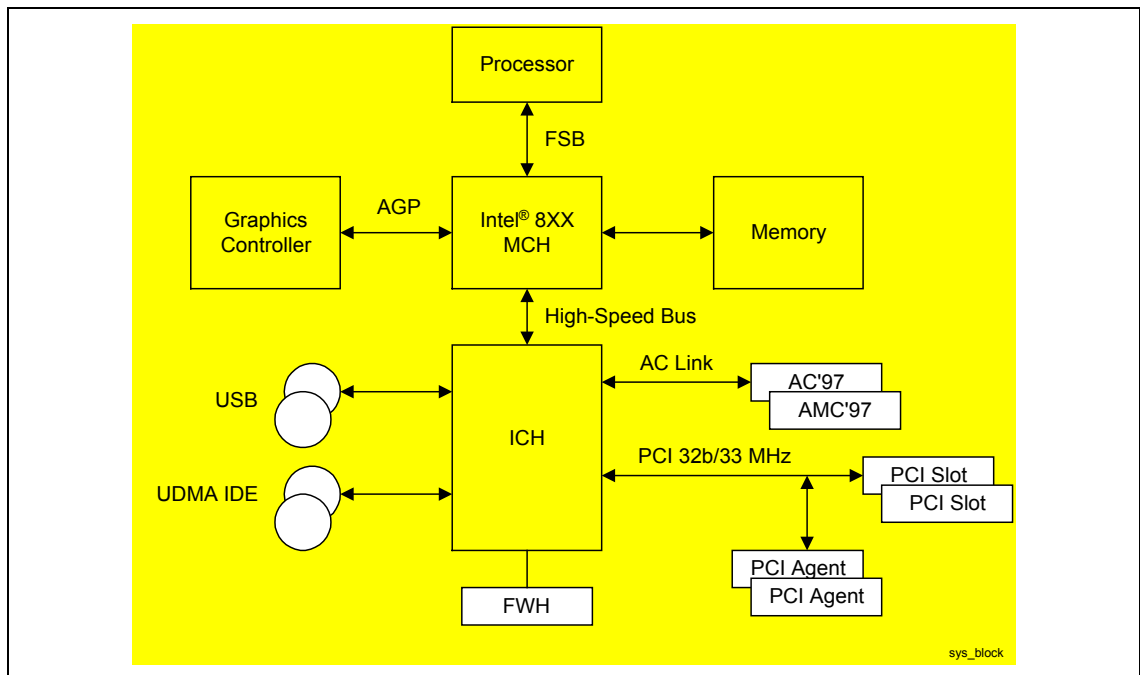
Functions that BIOS or Operating Systems (OS) must perform to ensure correct and reliable operation of the platform are described. Software specifications for the AC '97 digital controller are outlined. Details regarding the development of an audio device driver that is the baseline for a production driver already in the marketplace are provided.

It is assumed that the reader has a working knowledge of the AC '97 architecture and the Intel® ICH2 AC '97 controller implementation of the AC '97 specification. Also, the reader should understand the development of audio drivers for the target operating systems.

This document will be supplemented from time to time with specification updates that will contain information relating to the latest programming changes. Check with your Intel representative for the availability of specification updates.

Note: This document is based on the Intel® ICH2: AC '97 Software External Architecture Specification, Revision 0.95.

Figure 1-1. Block Diagram of Intel® 8XX Chipset with ICH2 Component

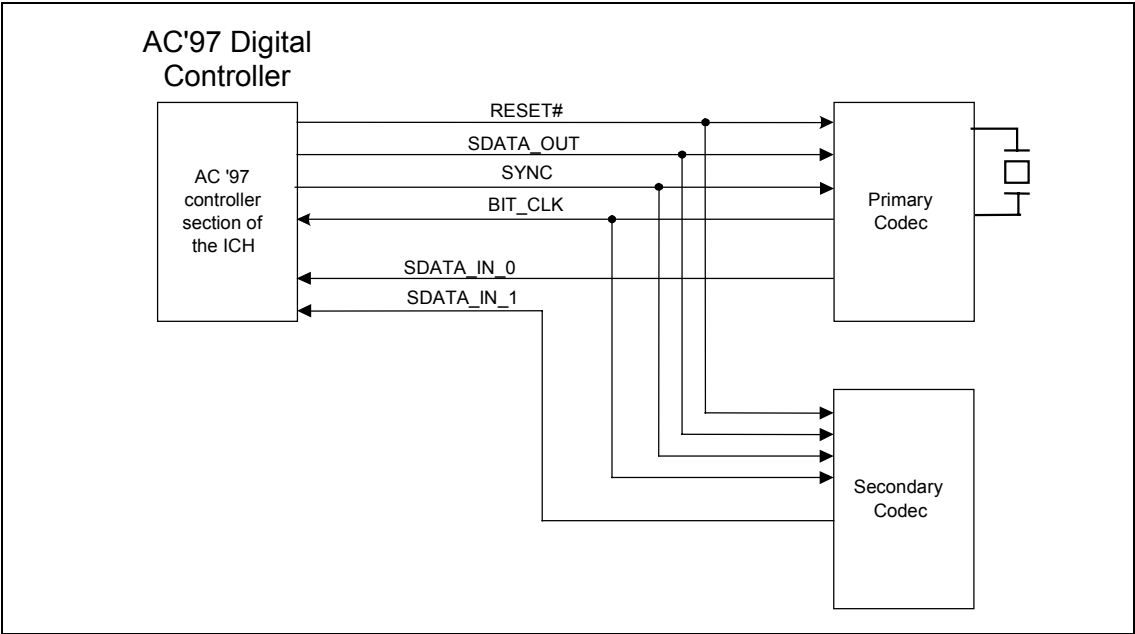


In this document, “ICH2” refers to the I/O Controller Hub 2. The ICH2 provides an AC '97-compliant controller. In this document, references to the AC '97 specification are to the AC '97 specification,



revision 2.1. The ICH2 AC '97 digital controller implementation interfaces to AC '97 2.1-compliant codecs. The ICH2 supports up to two AC '97-compliant codecs on the AC-link interface. The following figure shows the typical configuration of the ICH2 AC '97 controller and the companion codecs.

Figure 1-2. AC '97 Controller Connection to Its Companion Codec



This document specifies only the software requirements and the driver interface for the Intel® ICH2 AC '97 digital controller.

1.1. Reference Documents and Information Sources

Document Name	Available From
Intel® 82801AA (ICH) or Intel® 82801AB (ICH0) I/O Controller Hub Datasheet	Order number: 290655
Intel® 82801BA (ICH2) I/O Controller Hub Datasheet	Order number: xxxxxx

2. AC '97 Controller's Theory of Operation

The ICH2 AC '97 Digital Controller (DC) interface is an implementation of the AC '97 Link, with additional features for supporting transaction and device power management. The AC '97 DC includes DMA engines for high-performance data transfer to memory, via a hub interface.

AC '97 DC and link supports isochronous traffic, which emphasizes data timing. This is critical for maintaining the data stream from the audio and/or modem codec.

2.1. AC'97 Initialization

2.1.1. System Reset

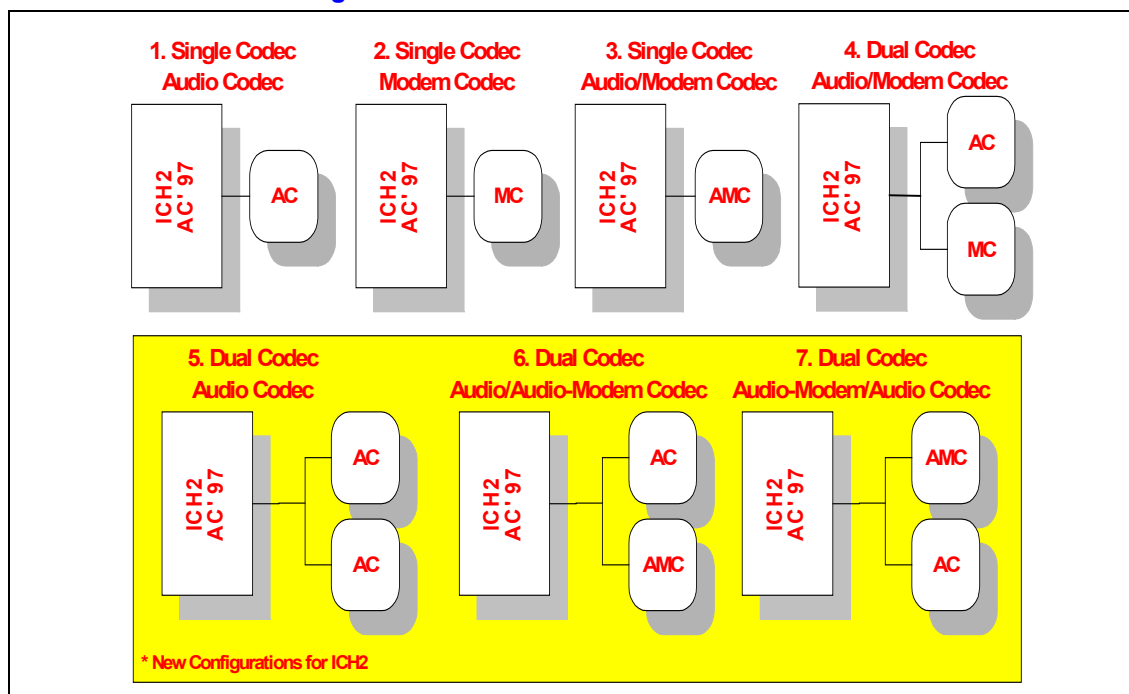
The AC '97 circuitry is reset upon power-up, by combining the PCIRST# signal with the AC Link RESET# signal. However, AC Link RESET# will not follow PCIRST# during a resume-from-sleep condition. During operation, the system can be reset by clearing the AC '97 Cold Reset bit in the Global Control/Status register (NABMBAR + 60h). This bit is maintained during the ICH2 sleep mode and can be used by the driver to select a warm or cold reset during a resume condition. If the codec is not present (i.e., AC '97 is not supported), codec ready will never be seen by the controller. Once the reset has occurred, a read to Mixer register 00h/80h will indicate the type of hardware residing in the codec(s).

Note: It is good practice to always check the Codec Ready bit before accessing the mixer register for the first time.

2.1.2. Codec Topology

The following figure shows the allowable codec configuration when attaching to the ICH2 AC '97 link. To avoid improper driver loading, the system BIOS should determine the presence/absence of the audio or modem codec attached on the AC link. (See Appendix A for the detailed procedure.)

Figure 2-1. Possible Codec Configurations



This information is used to disable (i.e., hide) the appropriate PCI function. To determine whether a codec or codecs are attached to the link, the system BIOS uses the following procedure.

2.1.3. BIOS PCI Configuration

As previously indicated, the AC '97 controller exposes two PCI functions in ICH2 device 31h. This allows for driver differentiation between these capabilities in the component.

- Function 5: AC '97 audio controller
- Function 6: AC '97 modem controller

As PCI devices, there are a number of registers that must be initialized in order to enable these functions. The following table summarizes these requirements.

Table 2-1. Audio Registers (Device 31 Function 5 Audio)

Offset	Register	Default	Initialize	Comments
04h–05h	Command (COM)	0000h	0005h	Bit 2: Bus Master Enable Bit 0: I/O Space Enable
10h–13h	Native Audio Mixer Base Address	00000001h	0000XX01h	0xXX00: Address in the 64-KB I/O space that allows 256 bytes of registers not in conflict with any other set
14h–17h	Native Audio Bus Mastering Base Address	00000001h	0000YY01h	0xYY00: Address in the 64-KB I/O space that allows 256 bytes of registers not in conflict with any other set
3Ch	Interrupt Line (INTLN)	00h	0Zh	A hardware interrupt (0-Fh) that follows the value assigned to PIRQB#. It has no effect on ICH2 and is used to indicate to software the IRQ value assigned to the device.

Table 2-2. Modem Registers (Device 31 Function 6 Modem)

Offset	Register	Default	Initialize	Comments
04h–05h	Command (COM)	0000h	0005h	Bit 2: Bus Master Enable Bit 0: I/O Space Enable
10h–13h	Native Audio Mixer Base Address	00000001h	0000XX01h	0xXX00: Address in the 64-KB I/O space that allows 256 bytes of registers not in conflict with any other set
14h–17h	Native Audio Bus Mastering Base Address	00000001h	0000YY01h	0xYY00: Address in the 64-KB I/O space that allows 256 bytes of registers not in conflict with any other set
3Ch	Interrupt Line (INTLN)	00h	0Zh	A hardware interrupt (0–Fh) that follows the value assigned to PIRQB#. It has no effect on ICH2 and is used to indicate to software the IRQ value assigned to the device.

A PnP-capable OS is responsible for initializing these PCI registers. If a PnP OS is not available in the system, then the BIOS is responsible for configuring all PCI devices, including these registers. A switch in the system setup usually is used to determine whether PnP is present. However, the final configuration and the existence/absence of this switch is implementation dependent.

2.1.4. Hardware Interrupt Routing

The audio and modem functions in the ICH2 internally share the same PCI IRQ (PIRQB#). The configuration software must take this into account and assign the same IRQ pin to both functions. Sharing IRQs increases the ISR latencies. Each ISR must determine if the interrupting device is the one serviced by the routine. If the device does not belong to the current servicing ISR, the ISR is responsible for calling the next ISR in the chain. PIRQB# also is exposed as a PCI interrupt on the PCI slots. Therefore, a device installed in a PCI slot may use the same IRQ assigned to the AC '97 functions. This further increases the ISR latencies.

In an environment where a high Quality of Service (QoS) is required, system designers must pay close attention to devices attached to the same PIRQ. Software-driven signal processing functions, such as in the case of software-driven modem and audio, require the maintenance of a low latency interrupt service, in order to maintain the proper functionality. Software driver programmers must pay close attention to the ISR latencies and make use of DPC, as much as possible.

2.2. DMA Engines

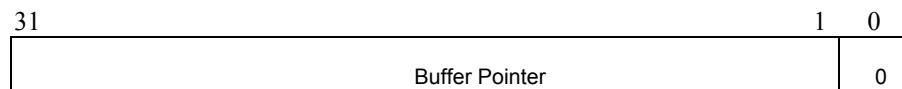
The ICH2 AC '97 controller uses the scatter/gather mechanism to access memory. There are three 16-bit DMA engines for audio PCM stereo in, PCM stereo out, and MIC mono. There are two 16-bit DMA engines for modem in and modem out. The audio and modem registers are located in two separate PCI functions in the ICH2 components, in order to allow for driver flexibility.

2.2.1. Buffer Descriptor List

The Buffer Descriptor list is an array of up to 32 entries, each of which describes a data buffer. Each entry contains a pointer to a data buffer, control bits, and the length of the buffer being pointed to, where the length is expressed as the number of samples. This, combined with the 16-bit sample size, gives the actual physical length of the buffer. The buffer length is restricted to 65536 samples. “0” in the buffer length indicates **no samples** to process. Each descriptor can point to a buffer of a different size. The samples are stored two per DWord (16-bit samples). In the case of audio PCM, these represent the left and right channels, respectively.

Figure 2-2. Generic Form of Buffer Descriptor (One Entry in the List)

(DWord 0 : 00–03h)



(DWord 1 : 04–07h)

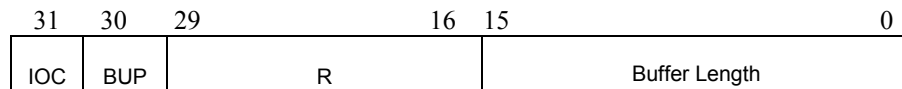


Table 2-3. BD Buffer Pointer (DWord 0: 00–03h)

Bit	Description
31:1	Buffer pointer. This field points to the location of the data buffer. The buffer must be DWORD aligned.
0	Reserved. Must be 0 when writing this field.

Table 2-4. D Control and Length (DWord 1: 04–07h)

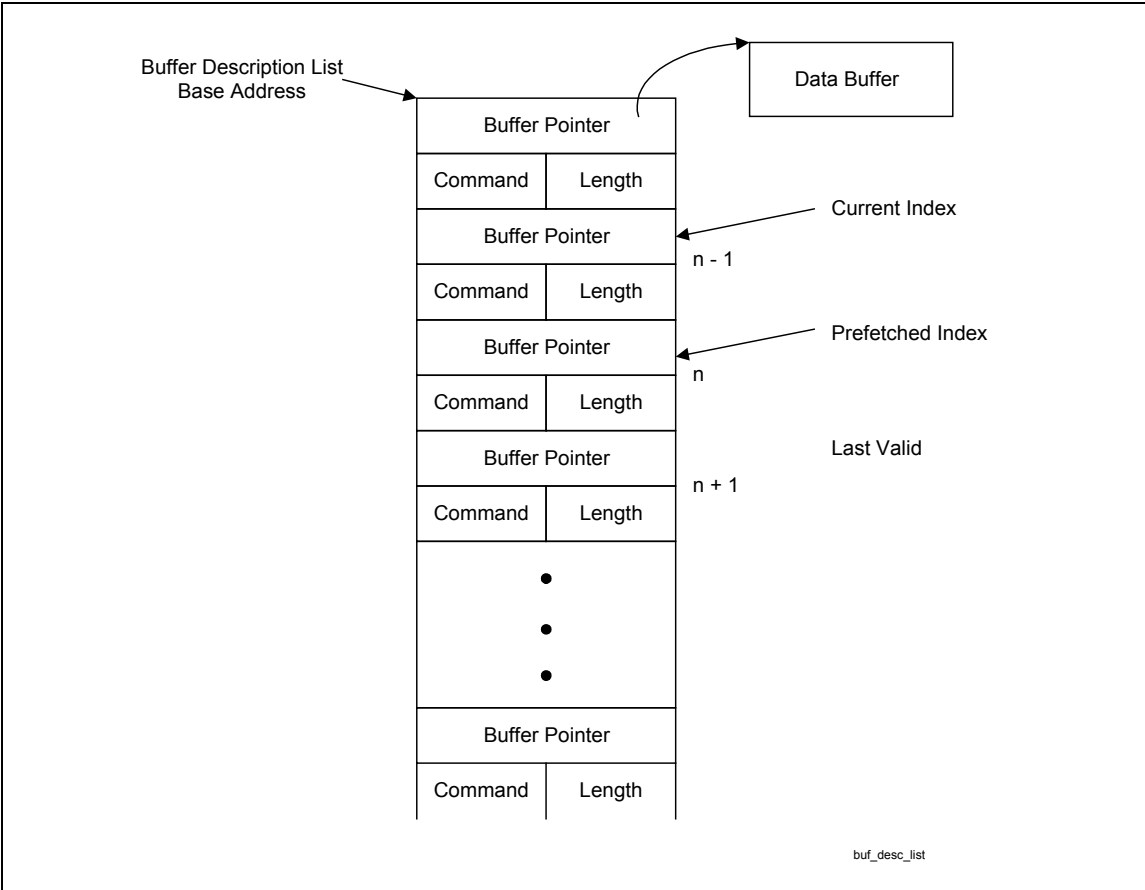
Bit	Description
31	Interrupt On Completion (IOC). 1 = Enabled. When this is set, it means that the controller should issue an interrupt upon completion of this buffer. It should also set the IOC bit in the status register. 0 = Disabled.
30	Buffer Underrun Policy (BUP). 0 = When this buffer is complete, if the next buffer is not yet ready (i.e., the last valid buffer has been processed), then continue to transmit the last valid sample. 1 = When this buffer is complete, if this is the last valid buffer, transmit zeros after this buffer has been processed completely. This bit typically is set only if this is the last buffer in the current stream.
29:16	Reserved. Must be 0 when writing this field.
15:0	Buffer length. This is the length of the data buffer, in number of samples. The controller uses this data to determine the length of the buffer, in bytes. “0” indicates <u>no sample</u> to process.



2.2.2. DMA Initialization

The maximum length of the buffer descriptor list is fixed at 32. (This is limited by the size of the index registers.) The figure below shows the organization of the Buffer Descriptor List.

Figure 2-3. Buffer Descriptor List



The following steps describe the driver initialization process for a single DMA engine. The same process should be repeated for each DMA engine.

1. Create the buffer descriptor list structure in memory (non-paged poll).
2. Write the Buffer Descriptor List Base Address register with the base address of the buffer descriptor list.

Table 2-5. Audio Descriptor List Base Address

Audio Buffer Descriptor List Base Address	I/O Address
PCM IN	NABMBAR + 00h (PIBDBAR)
PCM OUT	NABMBAR + 10h (POBDBAR),
MIC	NABMBAR + 20h (MCBDBAR)

Table 2-6. Modem Descriptor List Base Address

Modem Buffer Descriptor List Base Address	I/O Address
Line IN	MBAR + 00h (MIBDBAR)
Line OUT	MBAR + 10h (MOBDBAR),

- Set up the buffer descriptors and their corresponding buffers. Buffers are passed to the mini-port driver as Memory Descriptor Lists (MDL). These MDLs contain the physical page address of the virtual audio buffer. Multiple buffer descriptors may be required to represent a single virtual buffer passed to the mini-port driver. PCM buffers always must be of even length, since the number of channels is always a multiple of two.
- Once buffer descriptors have been set in memory, the software writes the Last Valid Index (LVI) register.

Table 2-7. Audio Last Valid Index

Audio Last Valid Index (LVI)	I/O Address
PCM IN	NABMBAR + 05h (PILVI)
PCM OUT	NABMBAR + 15h (POLVI)
MIC	NABMBAR + 25h (MCLVI)

Table 2-8. Modem Last Valid Index

Modem Last Valid Index (LVI)	I/O Address
Line IN	MBAR + 05h (MILVI)
Line OUT	MBAR + 15h (MOLVI)

- After the LVI registers have been updated, the software sets the run bit in the control register, in order to execute the descriptor list.

2.2.3. DMA Steady-State Operation

Software has two concurrent activities to perform during normal operation: Preparing new buffers/buffer descriptors and marking as free the processed buffer descriptors and buffers. Once the run bit has been set in bus master control register bit 0, the bus master fetches the buffer descriptor.

- The bus master starts processing the current buffer. Once current buffer has been processed, depending upon the bits set in the command field, the interrupt is asserted and the interrupt bit is set.
- The bus master increments the current and prefetch indices. It then starts executing the current buffer and schedules the next buffer to be prefetched.
- The buffer service routine maintains a variable that points to the head of the list of descriptors to be processed. The descriptor list service routine performs the following activities:

```

// Update head of descriptors to be processed
While (head != current_index)
{
    Mark head free ;
    // Check for end of descriptor list
    If head == base_address + (31 * 8);
        // Last entry on the list, set head to top of
list
        head = base_address;
    Else
        // Still inside list, increment head to next
entry
        head++;
}

```

Note: This algorithm needs to be optimized in order to reduce the number of memory accesses during execution. The While statement could translate to several memory accesses, if this code is not executed after each buffer descriptor update.

Also, the routine that prepares buffers maintains a variable that points to the entry *after* the tail of the list. This value is always the next entry after the Last Valid Index register. This routine utilizes the following algorithm:

```

// Update tail of descriptor list ready for execution
// and audio buffers when available for processing
While ((tail == free) && (buffers_available > 0))
{
    Prepare buffer descriptor indexed by tail;
    buffers_available--;
    //Assign tail to Last Valid Index
    LVI = Tail;
    // Check for end of descriptor list
    If (tail == base_address + 31 * 8);
        // Last entry on the list, set tail to top of
list
        tail = base_address;
    Else
        // Advance tail to next value
        tail++;
}

```


2.2.4. Stopping Transfers

There are two ways to stop transfers:

1. Simply turn off the Bus Master run/pause bit. This will immediately halt the current DMA transfer. Data in the output FIFOs will be read out until they empty. The registers will retain their current values and the AC link's corresponding slots will be invalidated. Setting the run/pause bit will resume DMA activity.
2. Software can stop creating new buffers and hence not update the Last Valid Index register. The bus master will stop once the last valid buffer has been processed. All register information is maintained. During this condition, the controller will transmit the last valid sample or zeros, depending on the status of the Buffer Underrun Policy (BUP) bit in the buffer descriptor entry. If the run/pause bit remains set, then any future update to the Last Valid Index register will cause the bus master operation to resume.

Note: Software must ensure that the DMA controller halted bit is set before attempting to reset registers.

2.2.5. FIFO Error Conditions

Two general conditions could cause FIFO error bit 4 in the status register to be set. Depending on the status of bit 3 in the control register, this also causes an interrupt.

2.2.5.1. FIFO Underrun

FIFO underrun will occur when the AC '97 controller FIFO is drained.

1. This results from system congestion. The DMA read transaction could still be pending, as data has not returned from memory. In this case, the controller will repeat the last sample until new data is available in the FIFO.
2. As a result of the DMA engine reaching the Last Valid Index, there is no further access to memory. Therefore, the FIFO will drain. In this case, the controller will transmit the last valid sample or zeros, depending on the status of the Buffer Underrun Policy (BUP) bit in the buffer descriptor entry. This condition is an error unless the software is able to update the descriptor list before the DMA engine reaches the Last Valid Index. However, this condition could result from the completion of the processing of the last buffer. It is up to the software driver to determine the final status of this condition. Also see the preceding Stopping Transfers section.

2.2.5.2. FIFO Overflow

FIFO overflow occurs when valid data is transmitted in proper AC link slots and the DMA FIFO remains full. Two conditions could cause FIFO error bit 4 in the status register to be set. Depending on the status of bit 3 in the control register, this also will cause an interrupt.

1. This results when the DMA engine is unable to update system memory with the contents of the FIFO, as a result of system congestion. In this case, all new samples received from the AC Link will be lost.
2. When the DMA engine reaches the last valid index, there is no further access to memory. Therefore, the FIFO will not drain. This condition is an error if the software is unable to update the descriptor list before the DMA engine reaches the last valid index. However, this condition could result naturally when the last buffer entry has been processed. It is up to the software driver to determine the final status of this condition. Also see the preceding Stopping Transfers section.

2.3. Arbitration

Up to five AC '97 DMA channels can be enabled at one time: PCM in, PCM out, Mic in, Modem in, and Modem out. A round-robin arbitration scheme is used to arbitrate among the five channels.

2.4. Data Buffers

2.4.1. Memory Organization of Data

The 16-bit samples are packed in, with two samples per DWord. The buffers are always DWord aligned.

2.4.2. FIFO Organization

The ICH2 AC '97 controller supports 16-bit samples on all channels.

Data is written to the FIFO in sample blocks, according to the order of valid slots in a channel. For example, for audio PCM in, the controller checks the first valid slot and adds it to the FIFO first entry as a word (16 bits). The next valid slot is added as the second word entry in the FIFO, in order to create the PCM stereo sample pair. This procedure assumes that the first valid slot always is the left channel (slot 3), followed by the right channel in slot 4 in the same or subsequent frame. If the codec transmits data repeating the slot, this will cause the controller to misplace the sample in the FIFO. Codecs compatible with the ICH2 AC '97 implementation should always maintain the indicated order, and should never use the same slot twice in order to transmit samples to the controller. The figures below show ICH2-compatible and ICH2-incompatible implementations.

Figure 2-4. Compatible Implementation with Left and Right Sample Pair in Slots 3 and 4 Every Frame

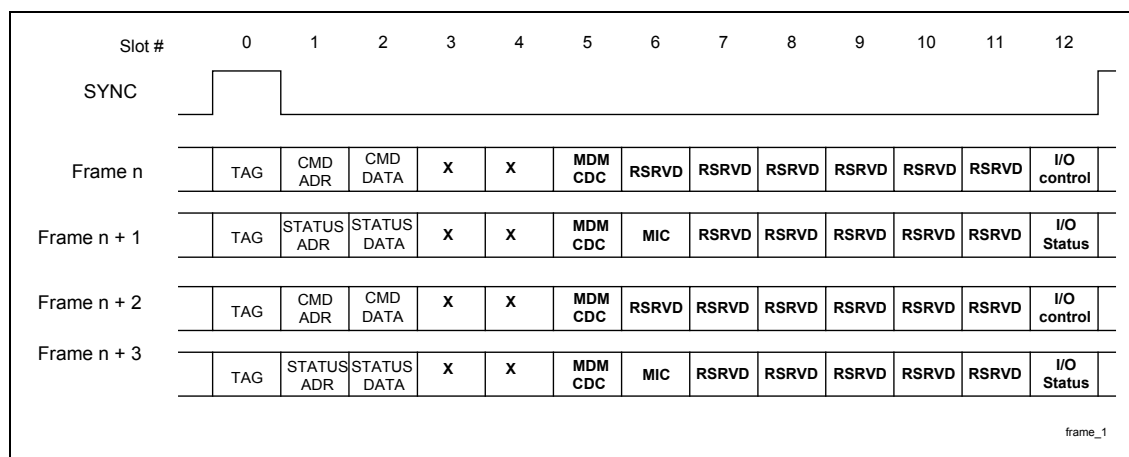


Figure 2-5. Compatible Implementation with Sample Rate Conversion Slots 3 and 4 Alternating over Next Frame

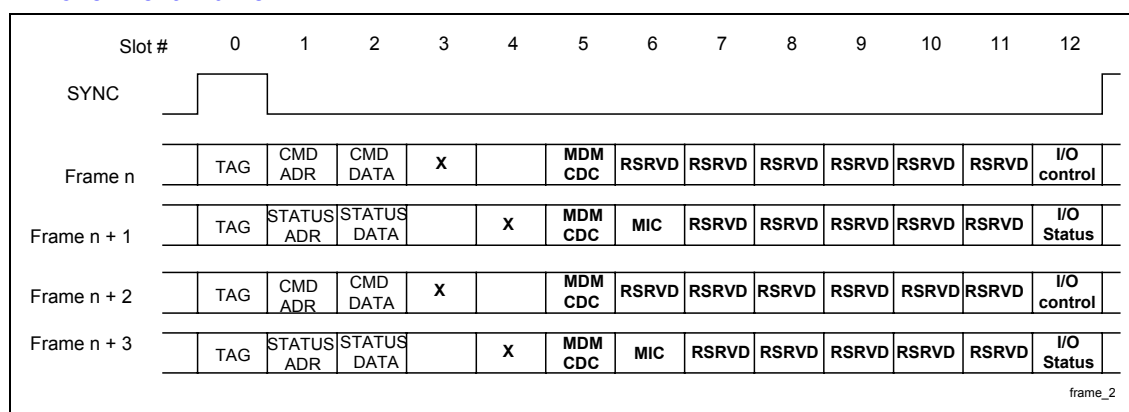


Figure 2-6. Incompatible Implementation of Sample Rate Conversion with Repeating Slots over Next Frames

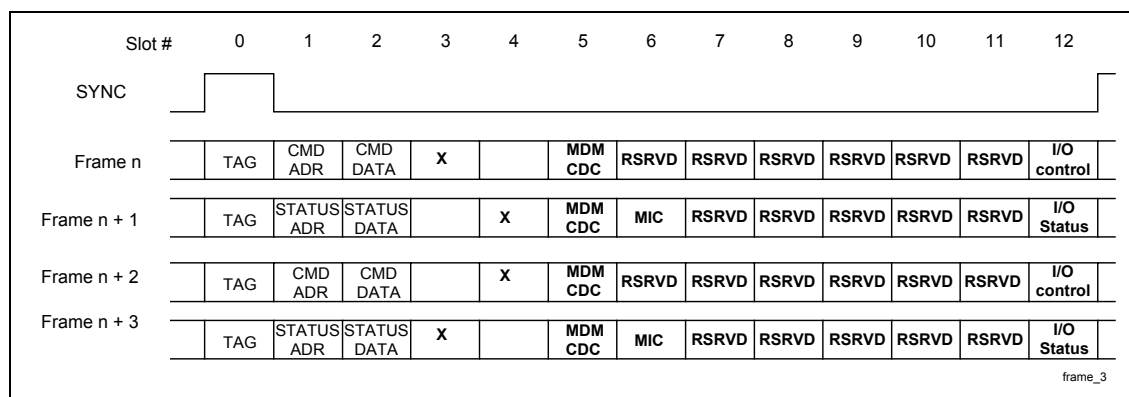


Table 2-9. FIFO Summary

Channel	No. of Samples	FIFO Depth	FIFO Width	Comments
Mic In	1	2	32 bits	Two samples per entry (DWord)
PCM In	2	4	32 bits	Left and right for stereo in the same FIFO. Two samples per DWord
PCM Out	2	4	80 bits	Left, right, surround left, surround right, center, and LFE channels in the same FIFO.
Modem In	1	2	32 bits	Two samples per entry (DWord)
Modem Out	1	2	32 bits	Two samples per entry (DWord)

2.5. Multiple Codec/Driver Support

The ICH2 AC '97 controller is capable of supporting a two-codec implementation. Under this implementation, both codecs share the SDATA_OUT signal, while independent SDATA_IN[0:1] are used by the codec to supply data to the controller. Even when two SDATA_IN are used, these two signals are logically OR'ed inside the digital controller, effectively creating one digital input data stream. In the ICH, this configuration precludes the simultaneous use of two similar codecs. Therefore, only one audio and one modem function are allowed to operate concurrently in ICH based configurations.

The ICH2, though, can have two audio codecs in what is termed a “split-audio” configuration, where one codec is decoding some of the PCM out slots (Left and Right, for instance) and a second audio codec decodes additional PCM out slots, such as Surround Left and Surround Right. PCM In data must still only be supplied by one audio codec, however.

2.5.1. Codec Register Read

Codec register reads are presented in the AC link in the next available frame after the I/O transaction is received by the controller. Data is returned to the controller, depending on codec availability. In the meantime, the processor waits for the transaction to complete, thereby stalling further software execution. To avoid longer-than-necessary latencies, the codec must return data in the next-available frame. Multiple frame transactions impose large system latencies, to the detriment of system performance.

Even when data is returned in the frame immediately after the read request is presented in the AC link, the minimum latency is still on the order of 40 μ s. To minimize the effect on the system caused by long latencies in the AC link, the software drivers **must** maintain a copy of the codec register in memory (i.e., shadow) and must use this data instead of accessing the codec.

Shadowing in memory is effective as long as the codec does not change the register values themselves. Therefore, the status of the GPIOs configured as inputs on the latest frame is accessible to software, by reading the register at offset 54h in the modem codec I/O space. Only the 16 MSBs are used to return GPI status. Reads from 54h will not be transmitted across the link. Instead, data received in slot 12 is stored internally in the controller, and the data from the most recent slot 12 is returned on reads from offset 54h.

Power-down in the codec offset 26h and 3Eh status registers is not supported by an automatic shadowing mechanism, as is the case for offset 54h. However, these registers are used sparingly and are read only during power-down status determination.

Finally, the codec ready status is required during system initialization. It is automatically reflected in the Global Status Register at NABMBAR + 30h (*MBAR* + 40h) bit 8 for the primary codec and at bit 9 for the secondary codec. These two bits need not be saved in memory.

2.5.2. Codec Access Synchronization

All codec register writes are posted transactions in the AC '97 controller. The AC '97 controller indicates transaction completion to the host processor immediately following the request, even when the transaction is actually pending completion in the AC link. This is done to improve system performance. However, it also restricts the operation of the driver(s). Also, register reads present synchronization issues.

Before a codec register access is initiated, the driver must check the status of the codec access in Progress (CAIP) bit 0, in the Codec Access Register at *NABMBAR* + 34h (*MBAR* + 44h). If no write is in progress, this bit will be 0, and the act of reading the register sets this bit to 1. This reserves the right to perform I/O read or write access. Once the write is completed, hardware automatically clears the bit. The driver also must clear this bit, if it decides not to perform a codec I/O write after reading this bit. If the bit has already been set, it indicates that another driver is performing a codec I/O write across the link, so the driver should try again later.

2.5.3. Data Request Synchronization in Audio Split Configurations

To support more than 2 channels of audio output, the AC '97 Specification 2.1 allows for a configuration where two audio codecs work concurrently to provide surround capabilities (refer to AC'97 Specification Rev. 2.1 Appendix C.) To maintain data on demand capabilities the AC '97 controller, when configured for 4 or 6 audio channels, will wait for all the appropriate Slot Request bits to be set before sending data in the *SDATA_OUT* slots. This allows for a simple FIFO synchronization of the attached codecs.

It is assumed that system software has properly programmed both audio codecs to the same sample rate and the codes have identical or at least compatible FIFO depth requirements. It is highly recommended that the codecs are provided by the same vendor upon certification of their interoperability in an audio channel configuration.

2.6. Power Management

Power management of the driver/codecs interaction requires careful sequencing in the AC '97 environment. In the ICH2 AC '97 environment, it is possible for two drivers to share the same AC link interface with two separate codecs. If a driver forces an aggressive sleep state in the link, it could have functional repercussions on the pairing codec. The D3 state is the deep sleep state in a device that abides by ACPI compliance requirements. When a driver is requested to set its device to the D3 state, the driver should enter the most aggressive power-saving mode possible. The D3 state also is often the precursor to a system-wide core power removal. Therefore several considerations must be taken into account, in order to maintain the device functionality and wake-up capability.

The procedure followed by an AC '97 device driver varies according to the system configuration. The following table lists the possible codec combinations supported by the ICH2 AC '97 controller.



The Intel® ICH2 audio/modem controller supports a maximum of one audio and one modem device. The following system implementations are possible. (For details, also see Figure 2-1. Possible Codec Configurations.)

Table 2-10 Codec Topologies

Config.						
1	AC	(Primary)				
2	MC	(Primary)				
3	AMC	(Primary)				Possible D3 state interactions
4	AC	(Primary)	+	MC	(Secondary)	Possible D3 State interactions
5	AC	(Primary)	+	AC	(Secondary)	Driver interaction concern
6	AMC	(Primary)	+	AC	(Secondary)	Possible D3 State Interactions
7	AC	(Primary)	+	AMC	(Secondary)	Possible D3 State Interactions

Note: These configurations could be limited further by the AC '97 riser card configuration and loading. For details, refer to the Audio/Modem I/O Riser Specification.

It is evident that configurations 1 and 2 above require no driver synchronization among between AC' 97 codecs. Configuration 1 and 2 are single codec topologies, and therefore an aggressive power saving mode is possible including disabling of the actual AC link without concern of affecting paired codec functionality.

Configuration 3 is a single codec topology that provides both functions audio and modem. In this configuration driver interaction is also critical if a separate set of drivers are in control of the audio and modem functions.

Configuration 4, however, is a two-codec topology. In this configuration 4 an aggressive power saving mode requires detailed attention to cross interactions and the effect on AC link functionality.

Configuration 5, is a two-codec audio topology. In this configuration concerns are on the proper power down sequence. However, no driver interaction is expected as only the audio driver executedriver executes power management functions.

Configuration 6 is also a two-codec topology with split audio and integrated modem support. This is the most complex interaction, as two different sets of driver will be operating in a complex topology.

Configuration 7 is identical to configuration 6, with the primary and secondary codecs switched.

In order to power manage AC' 97 codecs, there are two sets of PR bits that drivers need to be concerned aboutmanaged. One set is at offset NAMBAR + 26h in the audio function, mapped to offset 26h in the primary codec, and the and a second set is at MMBAR + 3Eh, mapped to offset 3Eh in the modem function. Notice that register 3Eh does not provide link down functionality, this is provided in register 56h bit 12, (MLNK) modem link.

2.6.1. Power Management Transition Maps

The following paragraphs discuss power management transition maps, within the constraints of an ACPI system environment. The following tables map a codec's PR bit transitions to specific ACPI D states for the device.

The following points were taken into consideration when generating the following tables:

- Power management is defined within the framework of a desktop system. Further power savings are possible by implementing more aggressive power management typical of mobile environment policies. (See the following *Aggressive Power Management* section.) However, these power savings are a trade-off between the driver complexity and the functional restrictions.
- The selection of a specific power policy depends on the proper identification of the topology by the driver(s).
- The secondary codec is provided with an external clocking mechanism and is not dependent on BIT_CLK to drive internal state machines, when in the power-down mode.
- After a warm or cold reset, the device driver brings all PR(x) bits to the D0 state.
- The transition from/to any Dx state is accomplished by simultaneously setting/resetting all appropriate PR(x) bits. The codec should not limit the PR(x) bit transition sequence discussed previously.
- Audio Codec Reg. 26h D15 EAPD (formerly, the PR<7> enable/disable function) is newly defined as the control for an external audio power amp. The audio codec should provide an audio amp output pin (GPO) that provides off/on capability according to this bit's set/reset status.
- The modem tables assume caller-ID capability during wake-up-on-ring, so Vref is ON during D3.
- The modem D3 configuration is dependent upon wake-up-on-ring event enable. If wake-up-on-ring is enabled, the GPIO cannot go down in D3.

Note: When a codec section is powered back on, the Powerdown Control/Status register (index 26h) should be read to verify that the section is ready, before attempting any further operations.

Configuration 1 single audio codec - primary:

Table 2-11. Power State Mapping for Audio Single-Codec Desktop Transition

PR<0:5> + (EAPD)								+12	+5 from +12	+3.3 Digital	+3.3 Vaux Digital	Comments
	EAPD	CLK	AC-Link	Mixer Vref.	Mixer	DAC	ADC					
Device State	7	5	4	3	2	1	0					
D0	0	0	0	0	0	0	0	On	On	On	On	All on
D1	0	0	0	0	0	1	1	On	On	On	On	DAC, ADC
D2	1	0	0	0	1	1	1	On	On	On	On	Mix, Amp
D3	1	1	1	1	1	1	1	Off	Off	Off	On	Clock, Vref

Configuration 2 single-modem codec - primary:

Table 2-12. Power State Mapping for Modem Single-Codec Desktop Transition

PR<A:D> + MLNK (other power control (PRx) bits do not apply for ICH2 implementation)						+12	+5 from +12	+3.3 Digital	+3.3 Vaux Digital	Comments
	Sdata_In	DAC1	ADC1	Vref	GPIO					
Device State	MLNK	D	C	B	A					
D0	0	0	0	0	0	On	On	On	On	All on
D1	0	1	1	0	0	On	On	On	On	DAC, ADC
D2	0	1	1	0	0	On	On	On	On	Same as D1
D3 (wake-up on ring)	1	1	1	0	0	Off	Off	Off	On	Sdata_In
D3	1	1	1	1	1	Off	Off	Off	On	Sdata_In, Vref, GPIO

Configurations 3 and 4 dual-function, single- or dual-codec configuration:

Table 2-13. Power State Mapping for Audio in Dual-Codec Desktop Transition

PR<0:5> + (EAPD)								+12	+5 from +12	+3.3 Digital	+3.3 Vaux Digital	Comments
	EAPD	CLK	AC-Link	Mixer Vref.	Mixer	DAC	ADC					
Device State	7	5	4	3	2	1	0					
D0	0	0	0	0	0	0	0	On	On	On	On	All on
D1	0	0	0	0	0	1	1	On	On	On	On	DAC, ADC
D2	1	0	0	0	1	1	1	On	On	On	On	Mix, Amp
D3	1	0	0	1	1	1	1	Off	Off	Off	On	Clock, Vref

1. PR(4) link-down and PR(5) internal clocks disable are NOT recommended for desktop configuration. Setting these to power control bits could affect modem operation in an AC + MC configuration.
2. In a mobile system configuration, PR(4) and PR(5) could be used to provide further power savings. Driver designers should use D3 state codec semaphores in the ICH2 AC '97 controller, in order to determine the audio or modem codec power status before setting the PR(4) and PR(5) bits. For details, refer to the Intel® ICH2: AC '97 External Architecture Specification. The mini-port driver developed for the Intel® ICH2 AC '97 controller does not provide this capability.

Table 2-14. Power State Mapping for Modem in Dual-Codec Desktop Transition

PR<A:D> + MLNK (other power control (PRx) bits do not apply for ICH2 implementation)						+12	+5 from +12	+3.3 Digital	+3.3 Vaux Digital	Comments
	Sdata_In	DAC1	ADC1	Vref	GPIO					
Device State	MLNK	D	C	B	A					
D0	0	0	0	0	0	On	On	On	On	All on
D1	0	1	1	0	0	On	On	On	On	DAC, ADC
D2	0	1	1	0	0	On	On	On	On	Same as D1
D3 (wake-up on ring)	1	1	1	0	0	Off	Off	Off	On	Sdata_In
D3	1	1	1	1	1	Off	Off	Off	On	Sdata_In, Vref, GPIO

Table 2-10 and Table 2-11 show the recommended power transition tables for a desktop system. The preceding tables eliminate the need for a driver to provide codec topology detection, thereby simplifying the initialization sequence. These tables do not provide the maximum power saving. However, they are believed to provide sufficient power saving for desktop applications. The OEM and IHV are free to differentiate their products further by enabling the deeper power savings obtained by identifying the codec topology.

Note: For further details on the device to system power states mappings refer to the Intel® ICH2 Programmer's Reference Manual.

2.6.2. Topology Detection

A set of drivers could always assume the preceding configurations 3 and 4 and establish their power management policy based on Table 2-10 and Table 2-11. These are the safest configurations, with a semi-aggressive power management style consistent with a desktop environment. However, even in a desktop environment, further power savings are possible when in single-codec configurations 1 and 2. In order to implement the preceding tables, the audio driver must be able to predetermine the AC link topology configuration.

2.6.2.1. Determining the Presence of a Secondary Codec

To determine whether or not a secondary codec is present, the driver must check the secondary codec ready bit located in the Global Status Register at:

Secondary Codec Ready: I/O Address: NAMBAR + 30h (MBAR + 40h), bit 9

If this bit is set to 1, it indicates that a secondary codec is active in the AC link.

2.6.2.2. Determining the Presence of a Modem Function

In the case of an AMC configuration, only the primary codec ready bit is indicated. In order to determine the proper power-down configuration, the audio driver must determine the presence/absence of modem functionality in the codec. The audio driver could check the Extended Modem ID Register at:

Extended Modem ID: I/O Address: NAMBAR + 3Ch

The content of this register is FFh, if no modem function is present.

2.6.3. Aggressive Power Management

As indicated in previous sections, it is possible to go into a more-aggressive power-saving mode by carefully synchronizing the audio and modem driver interactions over the AC link. This aggressive power saving usually is found in mobile environments, where battery power is critical.

Driver synchronization is required in a dual-codec configuration, where the audio driver could cause a link-down power condition, by setting the PR4 and PR5 bits in the audio codec register. When PR4 and PR5 are set, the AC link base clock BIT_CLK is stopped. If this action occurs while the modem codec is still in the operating mode, it will cause malfunctions and possibly hang the system.

To avoid this and similar situations, the audio and modem driver could follow a protocol using the provided audio and modem D3 state bit semaphores: AD3 for audio and MD3 for modem. These bits are located at:

Codec Write Semaphore Registers:

NABMAR + 30h audio I/O space and MBAR + 40h modem I/O space

Bit 16 for audio (AD3)

Bit 17 for modem (MD3)

The AC '97 drivers should set the appropriate bit after setting the codec in the D3 state. The audio codec could use this semaphore to determine if the modem codec is already in the D3 state and to shut down the link by also asserting PR4 and PR5 in the power management register in the audio function/codecs. The following sections review in detail the sequence of events for drivers/codecs entering the D3 state and resuming the D0 state.

2.6.3.1. Primary Audio Requested to Transition to D3 State

The audio power management procedure attempts to get the audio codec to transition to the D3 state.

```
If MD3 == true           // (sleeping?)
    {
        Audio_Power_Manage_Reg = D3 + PR4 + PR5;
                                // yes, sleep plus AC link down
    }
Else
    {
        Audio_Power_Manage_Reg = D3;    // No, sleep keeping link up
    }
AD3 = true;                // Set to "audio sleeping"
// Setting the flag last avoids race condition during D0->D3 transit.
```

2.6.3.2. Secondary Modem Requested to Transition to D3 State

The modem power management procedure tries to get the modem codec to transition to the D3 state.

```
Secondary_codec = D3 + MLNK // Yes, sleep plus SDATA_IN1 low
MD3 = true
// Setting the flag last avoids race condition during D0->D3 transit.
// MLNK corresponds to register 56h, bit 12 (D12).
```

2.6.3.3. Secondary Modem Requested to Transition to D0 State

The modem power management procedure tries to get the modem codec to transition to the D0 state.

```
MD3 = false           // Set to "modem awake"
//Setting the flag first avoid race condition during D3->D0 transit.
If Modem_ready == True
{
    Modem_Power_Manage_Reg = D0 // Bring back to fully awake.
}
If AD3 == true        // (audio sleeping?)
{
    Link_reset()      // Cause a warm or cold reset.
    While (!Modem_ready) // Wait for modem ready.
    {
        read modem codec ready bit every 400 ms
    }
    Modem_Power_Manage_Reg = D0 // Bring back to
awake.
}
```

2.6.3.4. Audio Primary Requested to Transition to D0 State

The audio power management procedure attempts to get the audio codec to transition to the D0 state.

```
AD3 = false           // set to "audio awake"
//Setting the flag first avoid race condition during D3->D0 transit.
If Audio_ready == True
{
    Audio_Power_Manage_Reg = D0; //Bring back to fully awake.
}
If MD3 == true;        // (modem sleeping?)
{
    Link_reset();      // Cause a warm or cold reset.
    While (!Audio_ready); // Wait for modem ready.
    {
        read audio codec ready bit every 100ms;
    }
    Audio_Power_Manage_Reg = D0; // Bring back to
awake.
}
```

Appendix B provides a schematic representation of the wake-up circuitry. This should be used as the reference for ACPI and APM wake-up code, since it relates to the preceding paragraph.

2.6.3.5. Using a Cold or Warm Reset

In the preceding pseudo code, there are several references to resetting the AC link “Link_reset()”. Before deciding whether to execute a cold or warm reset, drivers must determine whether or not the system enters a suspend event where core power is removed from the system. A device is in a “D3 hot” state after the device is set in the lowest power consumption mode and the core power is maintained. A device is in a “D3 cold” state when the device is set in the lowest power consumption mode and the core power is removed.

In the ICH2 AC '97 implementation, when core power is removed, the cold reset bit is reset to 0. This bit is located at:

NABMBAR + 2Ch and MBAR + 3Ch

Bit 1 AC'97 Cold Reset#

A driver requested to resume the D0 state from the D3 state must check the status of the AC '97 Cold Reset bit. If this bit = 0, the driver sets it to 1 in order to de-assert the AC_RESET# signal in the link, thus completing a cold reset. If the Cold Reset bit is set to 1, then a warm reset is required if the AC link is down according to the procedures indicated under aggressive power management. To execute an AC '97 warm reset, the driver must set to 1 the AC '97 Warm Reset bit located at:

NABMBAR + 2Ch and MBAR + 3Ch

bit 2 AC'97 Warm Reset#

A pseudo code representation is as follows:

```
void Link_reset(void)
{
    If Cold_Reset# == True // AC_RESET# asserted, D3 when cold!
    {
        Cold_Reset# = False; // De-assert AC_RESET# Wake-up!
    }
    Else
    {
        Warm_reset = True; // D3 is Hot! Do warm reset.
    }
}
```



This page left intentionally blank

3. Surround Audio Support

The AC'97 specification allows for up to 6 channel of audio supported in the AC link. The audio device driver must determine the number of audio channels available in the codec(s) and properly enable the AC'97 controller to support those channels in the link.

3.1. Determine Codec's Audio Channels

Upon determination of the link topology, system software must proceed to resolve the total number of audio channels available in the codec(s). The Appendix A of the AC' 97 sSpecification Rev. 2.1 in appendix A, uses defines codec registers index 28h to indicate the presence of surround, center and LFE channels as follows:

Extended Audio ID Register (Index 28h)

Table 3-1. Audio Codec Extended Audio ID Register

Bit D6:	CDAC=1 indicates optional PCM Center DAC is supported
Bit D7:	SDAC=1 indicates optional PCM Surround DAC is supported
Bit D8:	LDAC=1 indicates optional PCM LFE DAC is supported
Bit D9:	AMAP=1 indicates optional slot/DAC mappings based on Codec ID (Refer to AC'97 2.1 Appendix D for description)
Bit D15-D14:	ID1, ID0 is a 2-bit field which indicates the Codec configuration: Primary is 00; Secondary is 01, 10, or 11 (Refer to AC'97 2.1 Appendix C for details)

The AC' 97 Specification 2.1 aAppendix D, section D.3.2.1, provides detail information on the usage model for multiple audio channels. This specification assumes the proper use of the AMAP bit describes the partition of the channels exposing critical functionality. If AMAP bit is not set to "1" a generic driver will not be able to determine the proper codec to slot distribution for a split audio codec configuration.

Base on table 44 of the AC' 97 Specification Rev. 21. The following table describes the available codec topology to audio channel distribution for ICH2 AC' 97 controller.

Single Audio Codec Configuration (Refers to Figure 3, item 1)	
Primary	Total Audio Channels
L/R	2
L/R; S-L/R	4
L/R; S-L/R; C/LFE	6

Split Audio Codec Configuration (Refers to fFigure 3 items 5 and 6)		
Primary	Secondary	Total Audio Channels
L/R	S-L/R	4
L/R	S-L/R; C/LFE	6

Table 3-2. Multiple codec audio channel distribution**Legend:**

L/R: Left Stereo Channel (Slot 3); Right Stereo Channel (Slot 4)

S-L/R: Surround Left Channel (Slot 7); Surround Right Channel (Slot 8)

C/LFE: Center Channel (Slot 6); Low Frequency Enhancement “subwoofer” (Slot 9)

N/A: Not Applicable for this configuration

3.2. Enabling AC' 97 Controller Audio Channels

The ICH2 indicates how many channels is capable to supportsupports in its Global Status Register. The AC' 97 controller defaults to PCM Stereo support after system reset or suspend-resume from S3, S4 or S5. Audio drivers can determine the total number of channels and re-configure the controller to support either 4 or 6 -channels audio for PCM Out only. PCM In is not configurable and always is set for 2 -channels input. PCM MIC In is also not configurable and is always single channel (monaural.) The total number of audio channels supported are indicated at:

NABMBAR + 30h: Global Status Registers.

Table 3-3. PCM 4/6 -channels capability bits

21:20	PCM 4/6 Capability: These read-only bits indicate the capability to support more than 2 channels for the PCM OUT. These bits will both be 1 to indicate that the ICH2 supports both 4 and 6-channel PCM output.
-------	--

System software enables the number of channels it intends to provide to the codec at:

NABMBAR + 2Ch: Global Control Registers.

Table 3-4. PCM 4/6 -channels enable bits

21:20	PCM 4/6 Enable: Enables the PCM Output to be in 4 channel or 6-channel mode.	AC Link Slots Enable:
	00 = 2 channel mode (default).	3, 4 (L and R)
	01 = 4 channel mode.	3, 4, 7, 8 (L, R, RL and RR)
	10 = 6 channel mode	3, 4, 7, 8, 6, 9 (L, R, RL, RR, C and LFE)
	11 = Reserved	(undefined)



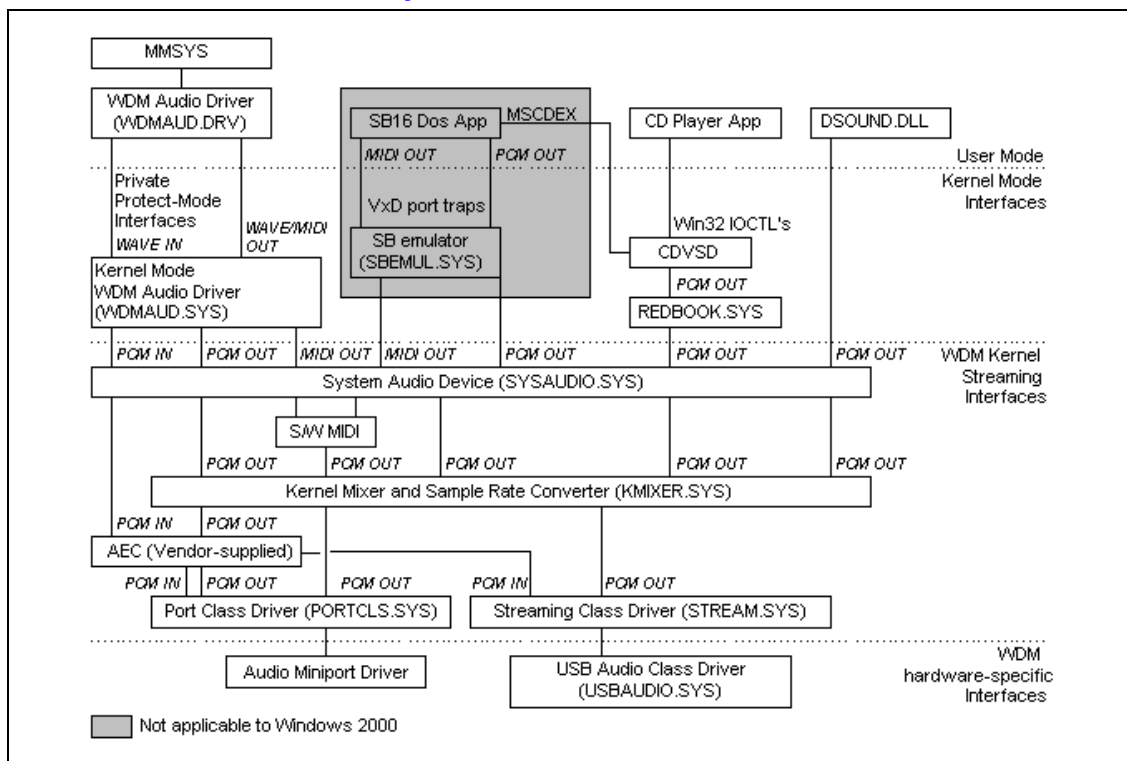
This page left intentionally blank

4. AC '97 Audio Driver

4.1. Win32 Driver Model

The AC' 97 DC software interface is targeted to be implemented as a Win32 Driver Model (WDM) miniport driver. WDM allows for a common set of binaries for device classes and buses to be shared by the Windows* platforms that support this model, currently Windows* 98 Second Edition and Windows* 2000 operating systems.

Figure 4-1. WDM Audio Driver Hierarchy



The AC' 97 DC interface under WDM should be implemented as an audio miniport. Figure 4-1 illustrates how the different driver layers are organized and relate to each other. The port class driver is the highest driver in the chain that the miniport driver will communicate with. The PCI bus driver is responsible for enumerating the AC' 97 codec, providing the device's resources, and loading the miniport driver. The bus driver is supplied solely by Microsoft. The miniport driver, which is device dependent, is responsible for interpreting the commands received from the port class driver into the device specific functions. For details on the audio miniport please refer to:

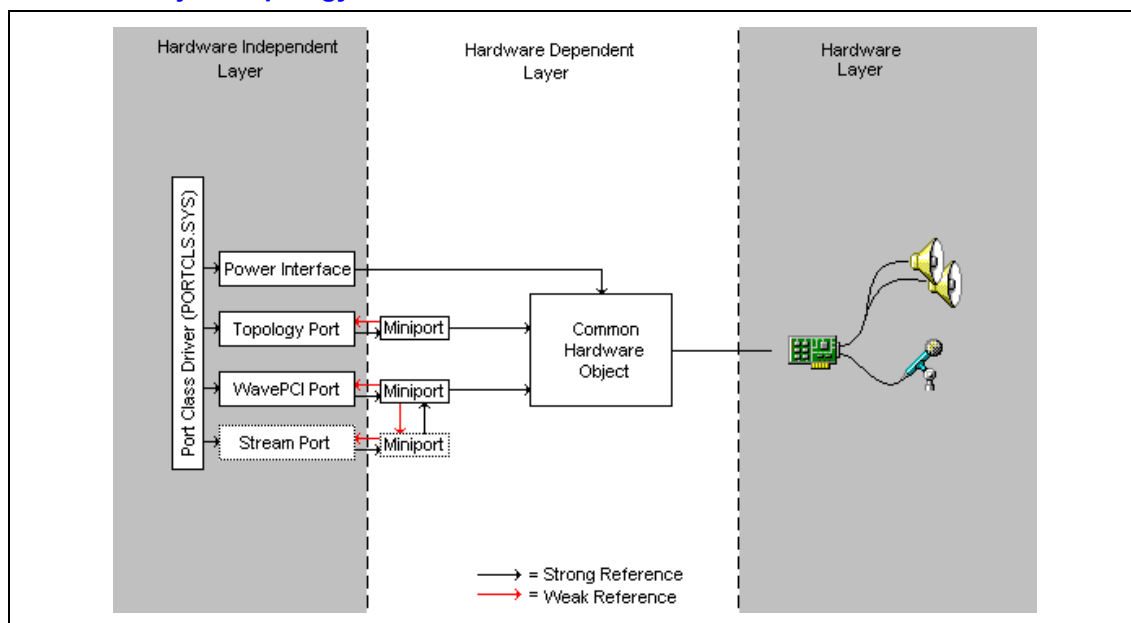
Microsoft Corporation, WDM Streaming Miniport Driver Model Specification Rev 0.1

The Intel® ICH2 AC' 97 driver supports methods for controlling the streaming clients of the port class driver and the analogue mixing capabilities of the AC' 97 codec. In order to support the streaming clients, the driver creates a stream miniport, which allocates a buffer descriptor list for the scatter/gather DMA engine. The list is allocated from locked memory using `HalAllocateCommonBuffer` as the stream is initialized. Later, when the stream is started, the stream miniport requests the initial packets from the operating system. The operating system provides both the physical and virtual addresses of the packets as well as the length of each packet in bytes. The driver then places the physical addresses and the lengths of the packets into the buffer descriptor list and starts the DMA. The driver also requests new packets during its interrupt service routine or when the operating system calls the stream miniport's `MappingAvailable` method to let it know that more packets are available. The analogue mixing capabilities of the AC' 97 codec are exposed as a topology miniport. The operating system calls the miniport's property handlers in response to user requests.

4.2. Example Driver

The following description is provided as reference material to support an Intel® ICH2 AC' 97 miniport driver. The driver was developed from the Creative Labs SoundBlaster 16 example driver in the Microsoft Windows® 2000 Device Driver Kit (DDK). As of Windows® 2000 Beta3, the Microsoft DDK also contains a sample AC' 97 ICH2 miniport driver (`src\wdm\audio\adapters\ac97`).

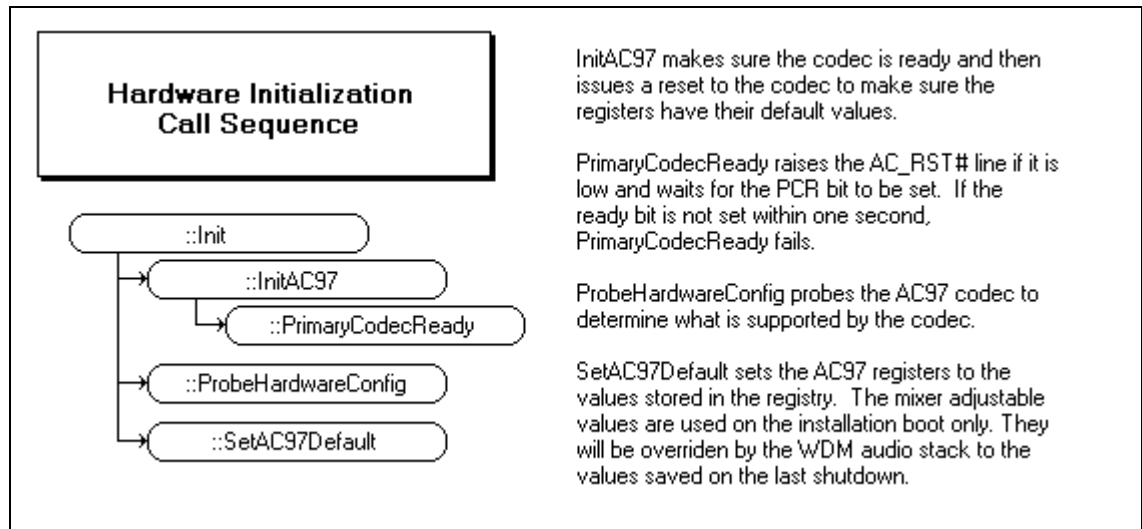
Figure 4-2. Driver Object Topology



The example driver contains four major C++ COM classes. Figure 4-2 shows their relationship to one another and to the port class driver. This entire hierarchy is replicated for each enumerated device that references the driver. This is not to be confused with split codec configurations. The stream port/miniport pair is only instantiated when a new logical channel is created (e.g. every time we play a wave file).

ADAPTER.CPP contains the initial driver entry points. Once the resources (I/O ports and IRQ line) are claimed, the hardware object is instantiated. Further calls are made to the port driver to define the miniports' entry points.

Figure 4-3. Hardware Initialization Call Sequence



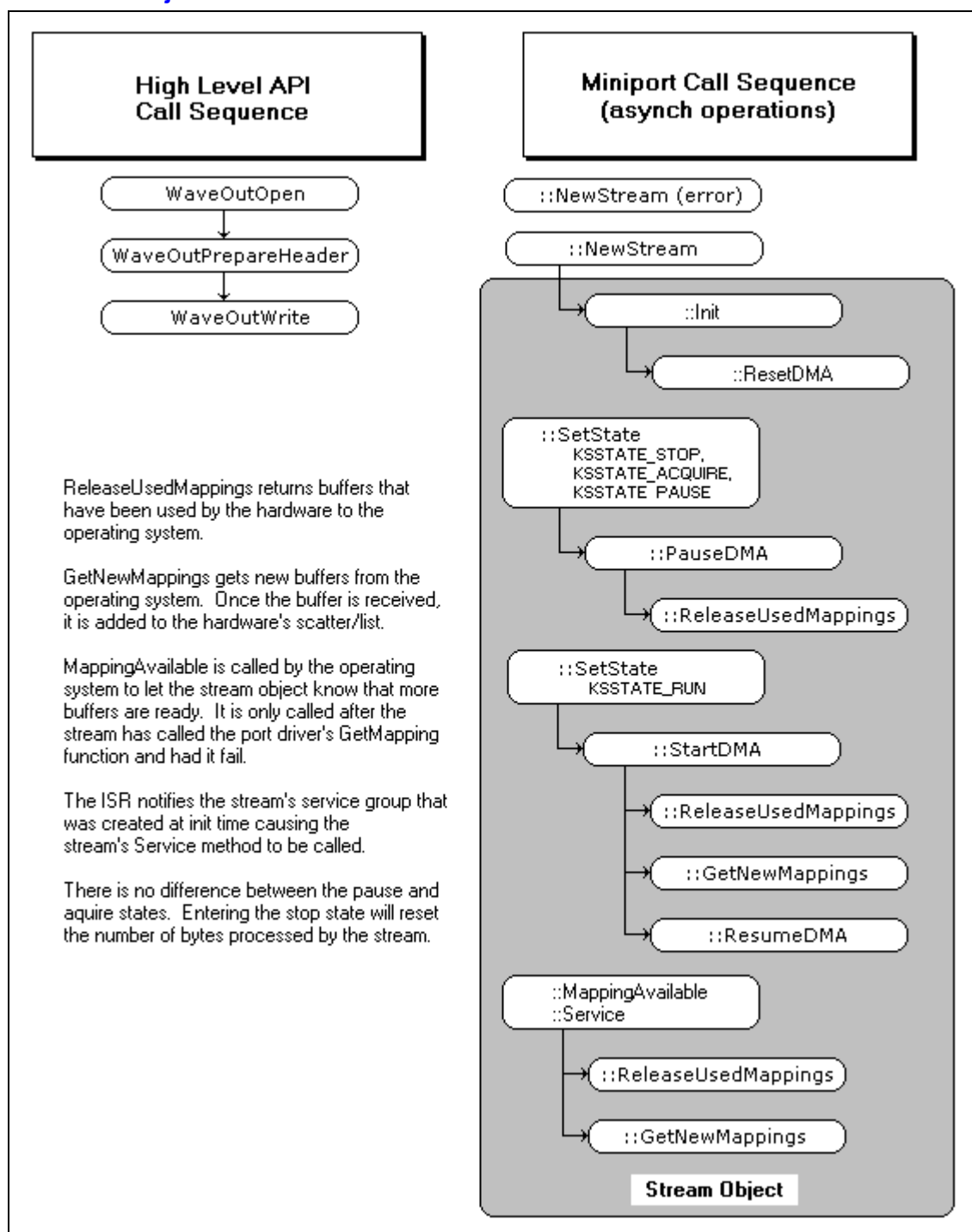
COMMON.CPP contains the definition of the common hardware object. This object is used to control all access to the physical hardware. Initialization of the physical hardware, is accomplished by this object when it is initialized. The hardware initialization call sequence is outlined in Figure 4-3. The object also implements the IAdapterPowerManagement interface which responds to ACPI messages from the operating system by saving and restoring the state of the codec.

MINTOPO.CPP contains the definition of the mixer topology object and implements the IMiniportTopologyICH interface (which includes the IMiniportTopology port driver interface). This object dynamically creates the mixer topology based on the results of probing the codec and returns the mixer property handlers' entry points to the operating system.

PROPHND.CPP contains the code responsible for translating mixer property change requests into the AC' 97 register values.

MINWAVE.CPP contains the definition of the WavePCI miniport object and implements the IMiniportWavePci and IPowerNotify port driver interfaces. This object is responsible for stream object creation, routing interrupts, and notifying the stream objects of changes to the device's power state. When an interrupt occurs, the ISR determines the source of the interrupt and queue a deferred procedure call (DPC) to service the interrupt. The DPC will call the Service method of the stream object associated with the interrupt. A separate stream object is created for each DMA channel in the ICH2. This object also determines the current position within the stream. *Care must be taken since the position is dependent on the current index and the current position in the buffer indicated by that index. These can not be read in a single read and the current index may change before the buffer position is read.*

Figure 4-4. Stream Object Overview



ICHWAVE.CPP contains the definition of the stream object which implements the `IMiniportWaveICHStream` interface (which includes the `IMiniportWavePciStream` port driver interface). This object is controls the DMA of the ICH2. This includes the responsibility of saving/restoring the state of the DMA in response to changes in the power state of the device. The interaction of the stream miniport with the operating system is outlined in Figure 4-4.

4.3. Plug and Play

The port driver filters all I/O request packets (IRPs) for the driver, including PnP IRPs. The device is started when the port driver is notified that the PCI bus driver has completed the `IRP_MN_START_DEVICE` IRP for the device. Each time the device is started, the driver will create the driver object hierarchy. WDM audio drivers should minimize global data as this inhibits multi-device scenarios (this is not an issue with this hardware design). It is also good practice to allocate device instance data in response to starting the device as opposed to when the driver's `AddDevice` is called. This is because the driver must filter PnP IRPs (instead of letting the port driver filter them) if it wishes to be notified of when the device is stopped or removed.

When the device is stopped or removed, the port driver will release its pointers to the object interfaces. This results in a chain reaction, which will destroy the object hierarchy. As the destructors for the objects are called, all resources claimed by the device are released and any memory allocated by the driver is freed.

4.4. Power Management

Power management is supported under WDM through two optional port driver interfaces. The `IAdapterPowerManagement` interface may be used for general notification of changes in the devices power state. The OS may query the device prior to changing the power state of the device through this interface. This object that implements this interface is registered with the port driver when the device is started. The second interface is `IPowerNotify`. This interface may be implemented by miniport drivers, if they wish to be notified independently of changes in the power state of the device. On transitions to lower power states, the port driver will call the methods of the `IAdapterPowerManagement` interface prior to calling the methods of the `IPowerNotify` Interface. The reverse is true for transitions to higher power states. In the AC' 97 example driver, both interfaces are implemented.

Since only one power management IRP is present in the system at any one time, the aggressive power management approach may be safely implemented and is encouraged. Multithreaded driver conflicts that would otherwise result are not an issue.

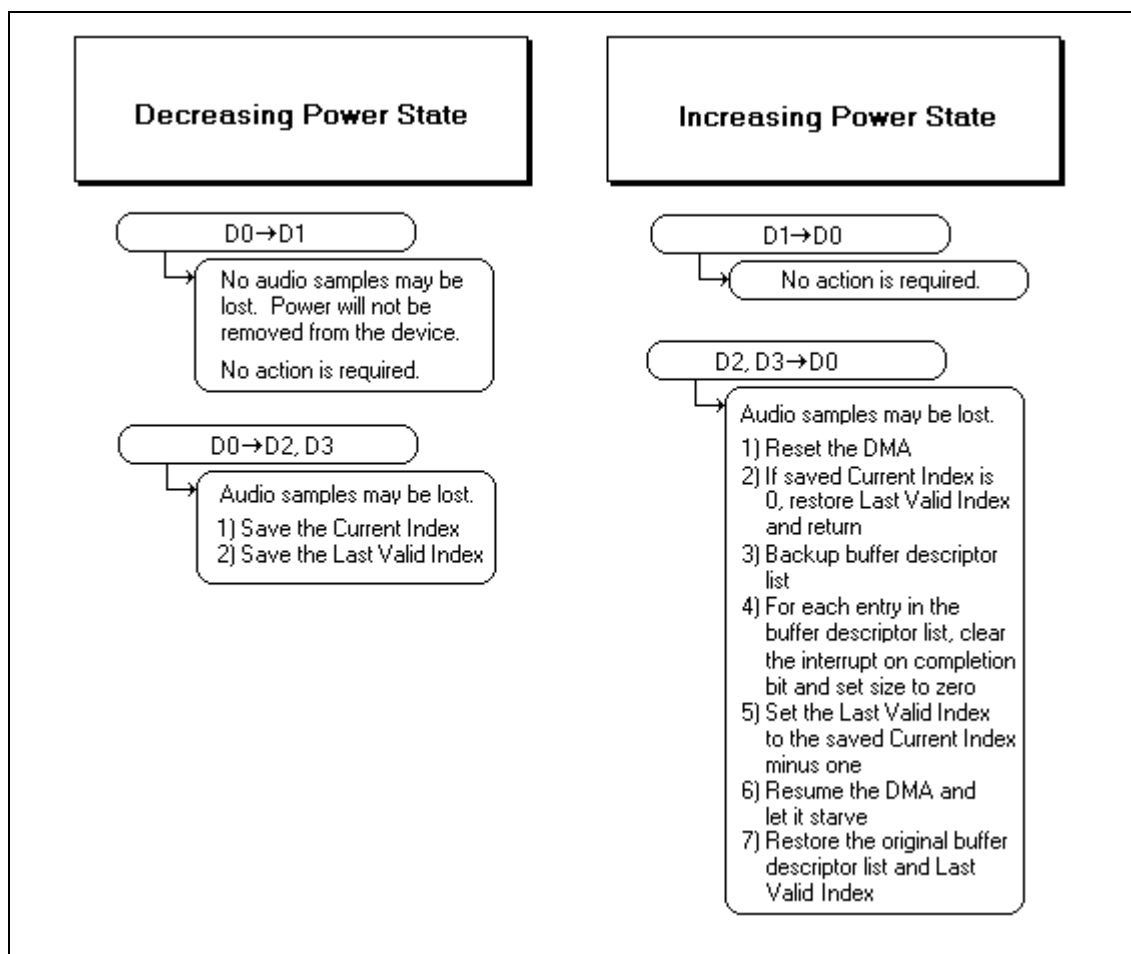
4.4.1. IAdapterPowerManagement

This interface is implemented by the common hardware object. Queries to change the power state of the device are never refused for any reason. On a transitions to different power states, this driver will change the power state of the AC' 97 codec as outlined in the Power Management Transition Maps. There is no need to save the current state of the AC' 97 codec upon entering the D3 state as all codec writes have been cached by the driver to maximize performance. The registers of the ICH2 are not saved by the hardware object as knowledge of the buffer descriptor list would be required. This responsibility is passed onto the WavePCI miniport object.

4.4.2. IPowerNotify

This interface is implemented by the WavePCI miniport object. The operating system is responsible for pausing any running streams prior to changing the device from the D0 state. On transitions to different power states, the miniport will use the weak reference to any stream objects to notify the stream of changes in the power state of the device. If no streams have been created, then the transition continues.

Figure 4-5. ICH2 Power Transition Process



However, if streams are instantiated, the stream object must save/restore the state of its associated DMA channel. This is a complicated process as many of the ICH2 DMA registers are read-only. Figure 4-5 outlines the process followed by the driver for saving/restoring the ICH2.

An alternative method would be to rearrange the buffer descriptor list if power is lost from the device. The chosen method attempts to restore the hardware to its previous state. The only register that is incorrect will be the position pointer within the current buffer. However, the amount is inconsequential and this limitation is not solved by rearranging the buffer descriptor list.

5. AC '97 Modem Driver

The AC' 97 specification allows for a modem codec to be connected to the AC link interface. This allows for the development of a software stack that provides modem functionality, i.e. a soft modem. Currently there is no single definition of how a soft-modem should be implemented under Microsoft operating systems as is the case for audio in a WDM environment. Soft modem vendors have developed a variety of implementations for Windows* 95, Windows* 98, Windows* NT* 4.0 and Windows* NT* 5.0 operating systems. The design problems are not trivial for the soft-modem developer. This document does not attempt to describe solutions for each of these environments, but focuses instead on facilitating the development of the driver/hardware interface. At the time of this specification, Microsoft and Intel are engaging with the industry to define a common interface for the WDM environment.

5.1. Robust Host Based Generation of a Synchronous Data Stream

This section presents a method for synchronous modem data to be reliably generated on the host processor of a computer system where the host processor is running a non real-time operating system whose maximum response latency (interrupt, thread, etc.) exceeds the period at which the host processor generates consecutive buffers of modem data. For the purposes of this discussion we will assume that the host processor periodically, in response to interrupts, generates a buffer of modem data in memory which is then utilized or consumed in a synchronous fashion by hardware. This modem data would comprise a sequence of digital representations of the analog signal to be transmitted over a phone line in accordance with one of a variety of modem protocols, baud rates, etc., and could be transmitted to the AC' 97 DMA engines via the buffer descriptor list as described in section 2.2.

We will also assume, for the sake of simplicity, that the data are double buffered, so that failure to generate new data before the next period will result in stream underflow from the hardware's viewpoint, but other scenarios including multiply buffered designs, as well as non periodic processing models can be accommodated. The algorithm works by providing good data followed by spurious data which is chosen or computed so as to be adequate to maintain connection with the other modem by, for example, transitioning seamlessly with respect to the phase of the carrier frequency and the baud rate, thereby avoiding a retrain. This enables the datapumps of the two modems to maintain synchronization in the face of infrequent hold-offs from processing experienced by the datapump of the host-based transmitting modem. The spurious data will, of course, cause a packet retransmission or other action by the controller, but to the receiving modem the incoming data signal will be indistinguishable from one corrupted by line conditions.

The first invocation of the host based modem task provides an initial buffer and one or more buffers of spurious data (henceforth, spurious buffers). The task chooses or computes each of the spurious buffer(s) based on signal state at end of immediately preceding buffer. Note that these buffers do not have to be computed on the fly but can be precomputed and indexed into at run time. Subsequent invocations overwrite the previously provided spurious data with good data so that under normal conditions the spurious data is never used or consumed by the DMA engine. In the event that the host based modem task does not generate the next buffer in time for the DMA engine to begin consuming it the DMA engine is able to begin consuming the spurious buffer and thereby maintain seamless connection with the other modem's datapump.

5.1.1. Spurious Data Algorithm

The following pseudo code presents a conceptual view the algorithm. `LastState()` is a function of which returns a unique integer as a function of, for example, the carrier phase and baud position of the *last* sample in the buffer. In an actual implementation this value would be computed during the course of generating the buffer. The `SpuriousBufferList` is an array of precomputed spurious buffers.

```
while (1)
{
    compute next buffer;
    pNextBuffer = &buffer;
    pSpuriousBuffer = &(SpuriousBufferList [LastState(buffer)]);
    wait for timer interrupt;
}
```

In this simplified scenario the device grabs the `pNextBuffer` address and stores it locally, using it to request the samples in the buffer one at a time. At the same time the device copies the `pSpuriousBuffer` into `pNextBuffer` so that when it is done with the current buffer it will get the spurious buffer unless the host software runs and overwrites `pNextBuffer` with a pointer to good data. In the next section we show how to implement the spurious data algorithm within the context of the AC' 97 buffer descriptor interface to hardware.

5.1.2. AC' 97 Spurious Data Implementation

The following pseudo code presents a modified version of the routine that prepares buffers and inserts them into the AC' 97 buffer descriptor list. In contrast to the version of this routine given in section 2.2.3, in this version `tail` points to the last good (i.e., non-spurious) buffer in the list. Furthermore, because the AC' 97 DMA engine prefetches the next buffer descriptor we split the buffer generated by the datapump into two parts, with the second as small as practical and denote this size as `MinBufferLength` (here assumed to be 8 samples = 4 DWORDS = 500 us. at 16 KHz.). For simplicity we assume that only a single buffer is generated by the datapump at a time and we ignore checking for the end of the descriptor list (i.e., the addition is implicitly modulo 32).

```
while (tail <= Prefetched_Index)
{
    tail++; // Happens IFF Spurious Data was used
}
if (((tail <= LastValidIndex) || (tail == free)) &&
    (((tail+1) <= LastValidIndex) || ((tail+1) == free)))
{
    Descriptor.BufferPtr[tail] = &buffer;
    Descriptor.BufferLength[tail] = length(buffer) - MinBufferLength;
```

```

        Descriptor.BufferPtr[tail+1] =
            &buffer + length(buffer) - MinBufferLength;
        Descriptor.BufferLength[tail+1] = MinBufferLength;
        tail += 2;
    }
else
{
    ;    //error: no space for this data buffer
}
if ((tail <= LastValid index) || (tail == free))
    Descriptor.BufferPtr[tail] =
        &(SpuriousBufferList[LastState(buffer)]);
    Descriptor.BufferLength[tail] =
        SpuriousBufferLength[LastState(buffer)];
    LastValidIndex = tail;
    //Note: tail is NOT incremented, so next time we'll overwrite this
    //      descriptor, which is the whole point of this algorithm
}
else
{
    LastValidIndex = tail-1;//warning: no space for spurious data
    buffer
}

```

The above implementation can be improved upon in a number of ways. Firstly, rather than adding a single (large) spurious buffer, a number of smaller ones could be chained together. In this way the amount of spurious data actually transmitted would be reduced while still maintaining a given level of protection against long latencies for the host-based software. Additionally, the implementation could be extended to handle multiple buffers at once, inserting several buffers in a row and only splitting the last one and then appending a spurious buffer or buffers. Finally, the descriptor list is a circular buffer and a real implementation would have to check `tail` and `tail+1` against `base_address + 31 * 8`.



This page left intentionally blank

6. **Appendix A: System BIOS Codec/Function Detection Algorithm**

The System BIOS is responsible for configuring the AC'97 Link topology. The *Communications and Networking Riser Specification 1.0* (available at <http://developer.intel.com/technology/cnr/>) defines a riser specification with plug-and-play capabilities for the AC Link. Please check the CNR specification and the CNR website for the latest information.

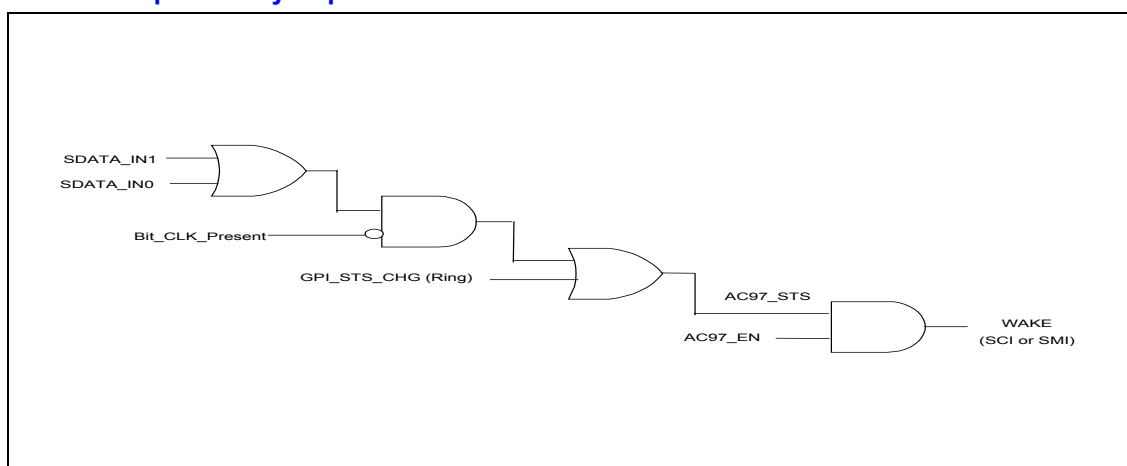


This page left intentionally blank

7. Appendix B: Detail for AC '97 Controller Wake-Up Detection Circuitry

The following diagram illustrates the wake-up detection circuitry for the AC '97 controller:

Figure 7-1. Wake-Up Circuitry Representation



SDATA_IN[0] : AC '97 serial data input 0 attached to the primary codec

SDATA_IN[1] : AC '97 serial data input 1 attached to the secondary codec

Bit_CLK_Present: Represents the presence of AC '97 bit clock signal.

GPI_STS_CGH: Indicates the codec GPI status change reported in slot 12.

AC '97_EN: Indicates that the AC '97 wake-up circuitry has been enabled.

AC '97_STS: Indicates that the wake-up status was requested.

WAKE: Indicates a system request for a wake-up event. (Translates to SCI or SMI, depending on the power management procedure.)

The following tables list the possible codec combinations, the possible power states, and how they resolve for a wake-up event.

Table 7-1. Wake-Up Condition Table for AC/MC Configuration

Codec Config.	System State	Audio Sleep State	Modem Sleep State	Status	Wake on
Audio 1 & Modem 2	S0	D0/D2 (link active)	D3 wake enabled	Bit_CLK = 1 AC97_EN = 1 GPI_STS = 1	OK (modem SDATA_IN active)
Audio 1 & Modem 2	S0	D3 (link inactive)	D3 wake enabled	Bit_CLK = 0 AC97_EN = 1 GPI_STS = 1	OK (modem Ring GPI active)
Audio 1 & Modem 2	S1–S5*	D3 (link inactive)	D3 wake enabled	Bit_CLK = 0 AC97_EN = 1 GPI_STS = 1	OK (modem Ring GPI active)

Table 7-2. Wake-Up Condition Table for AMC Configuration

	System State	Audio Sleep State	Modem Sleep State	Status	Wake on
AMC 1	S0	D0/D2 (link active)	D3 wake enabled	Bit_CLK = 1 AC97_EN = 1 GPI_STS = 1	OK (modem SDATA_IN active)
AMC 1	S0	D3 (link inactive)	D3 wake enabled	Bit_CLK = 0 AC97_EN = 1 GPI_STS = 1	OK (modem ring GPI active)
AMC 1	S1–S5*	D3 (link inactive)	D3 wake enabled	Bit_CLK = 0 AC97_EN = 1 GPI_STS = 1	OK (modem ring GPI active)

Table 7-3. Wake-Up Condition Table for Single AC or MC Configuration

	System State	Audio Sleep State	Modem Sleep State	Status	Issue
Audio 1	S0–S5	D3 (link inactive)	N/A	Bit_CLK = 0 AC97_EN = 0	OK (modem SDATA_IN active)
Modem 1	S0	N/A	D3 wake enabled	Bit_CLK = 0 AC97_EN = 1 GPI_STS = 1	OK (modem ring GPI active)
Modem 1	S1–S5	N/A	D3 wake enabled	Bit_CLK = 1 AC97_EN = 1 GPI_STS = 1	OK (modem ring GPI active)

Intel around the world

United States and Canada

Intel Corporation
Robert Noyce Building
2200 Mission College Boulevard
P.O. Box 58119
Santa Clara, CA 95052-8119
USA
Phone: (800) 628-8686

Europe

Intel Corporation (UK) Ltd.
Pipers Way
Swindon
Wiltshire SN3 1RJ
UK

Phone:
England (44) 1793 403 000
Germany (49) 89 99143 0
France (33) 1 4571 7171
Italy (39) 2 575 441
Israel (972) 2 589 7111
Netherlands (31) 10 286 6111
Sweden (46) 8 705 5600

Asia-Pacific

Intel Semiconductor Ltd.
32/F Two Pacific Place
88 Queensway, Central
Hong Kong, SAR
Phone: (852) 2844 4555

Japan

Intel Kabushiki Kaisha
P.O. Box 115 Tsukuba-gakuen
5-6 Tokodai, Tsukuba-shi
Ibaraki-ken 305
Japan
Phone: (81) 298 47 8522

South America

Intel Semicondutores do Brazil
Rue Florida, 1703-2 and CJ22
CEP 04565-001 Sao Paulo-SP
Brazil
Phone: (55) 11 5505 2296

For more information

To learn more about Intel Corporation, visit our site
on the World Wide Web at www.intel.com

