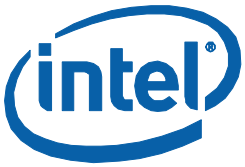


DCMI-HI

DCMI Host Interface Specification

October 7, 2010

Revision 1.0

Contents

Figures

Tables

Revision History

Revision Number	Description	Revision Date
1.0	Initial release	October 7, 2010

§

1 Introduction

This document includes a specification of the Intel DCMI Host Interface (DCMI-HI). DCMI-HI defines the PCIe Host interface and the messages that are used to set up and enable DCMI Messaging between host software and an Intel® Manageability Engine (management controller) that is running DCMI firmware. This information may be used to develop host driver software and utilities that support utilize DCMI in Intel platforms.

This specification defines the DCMI-HI is a logical interface that enables the PC BIOS and host software agents to transfer DCMI requests and responses with a management controller over a HECI physical interface. The DCMI-HI may also be used by BIOS prior to transferring control to the operating system.

Note that other interfaces may also be used by BIOS to communicate to the management controller during pre-boot and to support communication with an SMI Handler. Such interfaces are outside the scope of the specification. Thus, this specification does not require DCMI-HI to be used as the *only* interface for BIOS for DCMI communication with the management controller.

2 References and Terminology

2.1 Reference Documents

The following documents are companion and supporting specifications for the DCMI-HI:

[DCMI]	*– DCMI – Data Center Manageability Interface Specification v1.0, Revision 1.0,* May 1, 2008, http://www.intel.com/go/dcmi
[IPMI]	*- IPMI - Intelligent Platform Management Interface Specification Second Generation,* v2.0, Revision 1.0, February 12, 2004 with errata revision 3, February 15, 2006. http://developer.intel.com/design/server/ipmi/

2.2 Terminology

DCMI-HI	DCMI Host Interface Specification (this document)
CB	Circular Buffer
FW	Firmware
ME	Intel® Manageability Engine

3 HECI Hardware Interface

3.1 Overview

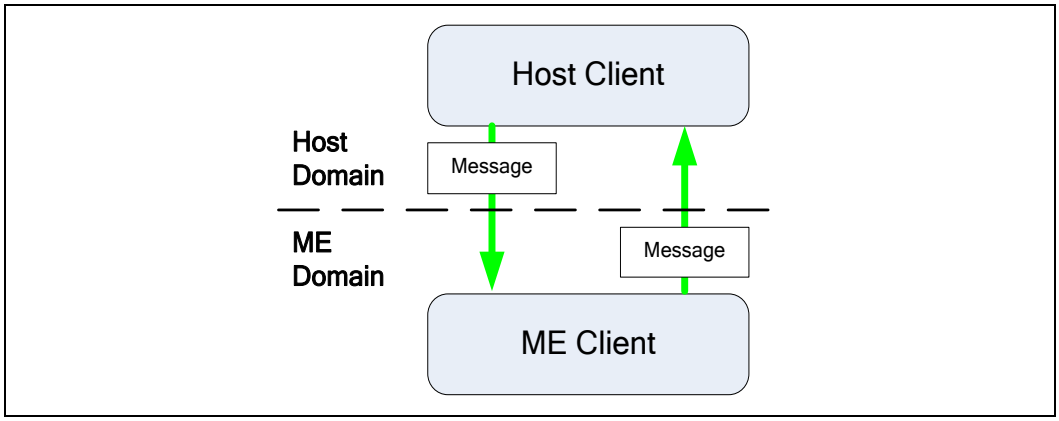
HECI is implemented as a PCI function that is enumerated by host software on the PCI bus. The HECI function provides a mechanism for host software to communicate with the Intel® Management Engine. HECI is a bi-directional fully asynchronous interface that passes messages between host software and ME firmware.

Host software and ME firmware exchange messages using a set of circular buffers and Control/Status registers (CSR's). There are two circular buffers (CBs), one for passing messages from the host to the ME and one for passing messages from the ME to the host.

Messages may be of variable length, but must consist of one or more 32-bit Dwords; message size info will be part of message. The HECI hardware interface provides CB (circular buffer) Read & Write Pointers and CB Depth fields for software / firmware to manage the circular buffers. CB Full and CB Empty conditions can be calculated from the CB Read & Write pointer and CB Depth information.

A basic diagram of the HECI register interface is shown in Figure 3-1 below.

Figure 3-1. HECI High Level Architecture



3.1 Register and Memory Mapping

The HECI Host PCI device is used by host software. Host software includes BIOS during the pre-OS boot time and OS driver during OS runtime. This HECI Host PCI device will be exposed to host software as a PCI device and used by an operating system driver. A basic diagram of the HECI register interface is shown in Figure 3-2 below.

Figure 3-2. HECI Register Interface

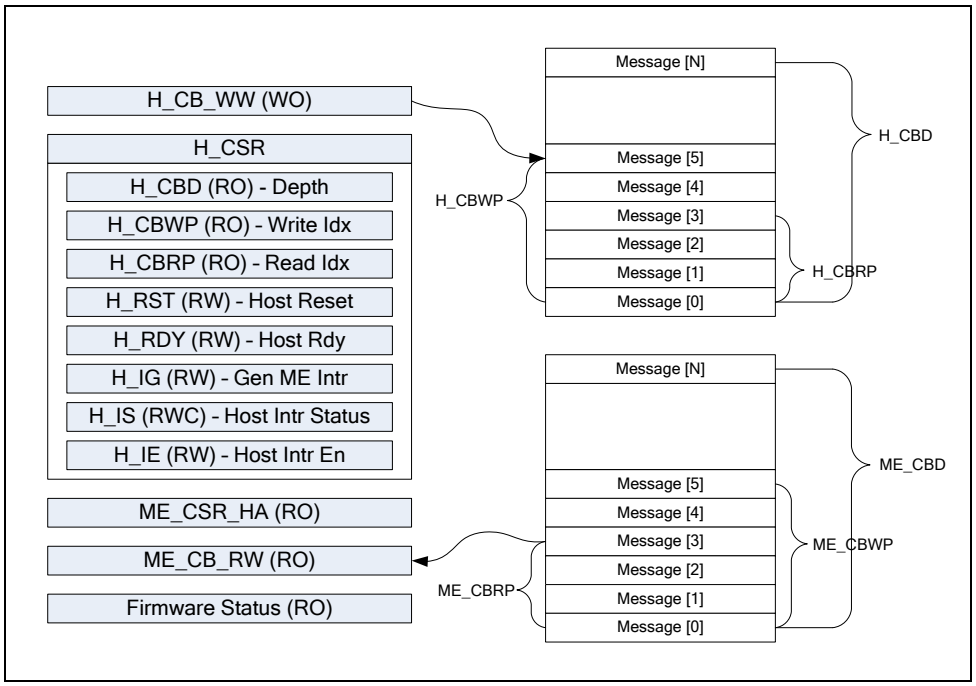


Table 3-1 provides a summary of the registers used in the HECI interface and their memory offsets and access. These registers are decoded as bus number X, device number Y, function number Z, depending on what chipset is used. All registers are reset by PLTRST# unless otherwise specified in a bit field description.

X/Y/Z – Vary from Chipset to Chipset. Please refer HECI device section in respective chipset documentation for the Bus number/Device number/Function number in B/D/F/Type.

Table 3-1 HECI PCI Configuration Register Summary

Register Name	Register Symbol	Register Start (hex)	Register End (hex)	Access
Identifiers	ID	0	3	RO;
Command	CMD	4	5	RO; RW;
Device Status	STS	6	7	RO;
Revision ID	RID	8	8	RO;
Class Code	CC	9	B	RO;
Cache Line Size	CLS	C	C	RO;
Master Latency Timer	MLT	D	D	RO;
Header Type	HTYPE	E	E	RO;
Built In Self Test	BIST	F	F	RO;
HECI MMIO Base Address	HECI_MBAR	10	17	RO; RW;
Sub System Identifiers	SS	2C	2F	RWO;
Capabilities Pointer	CAP	34	34	RO;
Interrupt Information	INTR	3C	3D	RO; RW;

Register Name	Register Symbol	Register Start (hex)	Register End (hex)	Access
Minimum Grant	MGNT	3E	3E	RO;
Maximum Latency	MLAT	3F	3F	RO;
Host Firmware Status	HFS	40	43	RO;
PCI Power Management Capability ID	PID	50	51	RO;
PCI Power Management Capabilities	PC	52	53	RO;
PCI Power Management Control And Status	PMCS	54	55	RWC; RO; RW;
Message Signaled Interrupt Identifiers	MID	8C	8D	RO;
Message Signaled Interrupt Message Control	MC	8E	8F	RO; RW;
Message Signaled Interrupt Message Address	MA	90	93	RW; RO;
Message Signaled Interrupt Upper Address (Optional)	MUA	94	97	RW;
Message Signaled Interrupt Message Data	MD	98	99	RW;
HECI Interrupt Delivery Mode	HIDM	A0	A0	RW;

3.1.1 ID - Identifiers

B/D/F/Type: X/Y/Z/PCI
Address Offset: 0-3h
Default Value: 00008086h
Access: RO;
Size: 32 bits

Bit	Access	Default Value	Description
31:16	RO	xxxxh	**Device ID (DID):** Device number assigned by Intel.
15:0	RO	8086h	**Vendor ID (VID):** 16-bit field identifies Intel as the vendor, assigned by the PCI SIG.

3.1.2 CMD - Command

B/D/F/Type: X/Y/Z/PCI
Address Offset: 4-5h
Default Value: 0000h
Access: RO; RW;
Size: 16 bits

Bit	Access	Default Value	Description
15:11	RO	00000b	**RSVD (RSVD):** Reserved
10	RW	0b	**Interrupt Disable (ID):** Disables this device from generating PCI line based interrupts. This bit does not have any effect on MSI operation.
9	RO	0b	**Fast Back-to-Back Enable (FBE):** Not implemented, hardwired to 0.
8	RO	0b	**SERR# Enable (SEE):** Not implemented, hardwired to 0.
7	RO	0b	**Wait Cycle Enable (WCC):** Not implemented, hardwired to 0.
6	RO	0b	**Parity Error Response Enable (PEE):** Not implemented, hardwired to 0.
5	RO	0b	**VGA Palette Snooping Enable (VGA):** Not implemented, hardwired to 0
4	RO	0b	**Memory Write and Invalidate Enable (MWIE):** Not implemented, hardwired to 0.
3	RO	0b	**Special Cycle Enable (SCE):** Not implemented, hardwired to 0.
2	RW	0b	**Bus Master Enable (BME):** Controls the HECI host controller's ability to act as a system memory master for data transfers. When this bit is cleared, HECI bus master activity stops and any active DMA engines return to an idle condition. This bit is made visible to firmware through the H_PCI_CSR register, and changes to this bit may be configured by the H_PCI_CSR register to generate an ME MSI. When this bit is '0', HECI is blocked from generating MSI to the host CPU. **Note:** DCMI-HI does not use DMA.
1	RW	0b	**Memory Space Enable (MSE):** Controls access to the HECI host controller's memory mapped register space.
0	RO	0b	**I/O Space Enable (IOSE):** Not implemented, hardwired to 0.

3.1.3 STS - Device Status

B/D/F/Type: X/Y/Z/PCI

Address Offset: 6-7h
Default Value: 0010h
Access: RO;
Size: 16 bits

Bit	Access	Default Value	Description
15	RO	0b	**Detected Parity Error (DPE):** Not implemented, hardwired to 0.
14	RO	0b	**Signaled System Error (SSE):** Not implemented, hardwired to 0.
13	RO	0b	**Received Master-Abort (RMA):** Not implemented, hardwired to 0.
12	RO	0b	**Received Target Abort (RTA):** Not implemented, hardwired to 0.
11	RO	0b	**Signaled Target-Abort (STA):** Not implemented, hardwired to 0.
10:9	RO	0b	**DEVSEL# Timing (DEVT):** These bits are hardwired to 00.
8	RO	0b	**Master Data Parity Error Detected (DPD):** Not implemented, hardwired to 0.
7	RO	0b	**Fast Back-to-Back Capable (FBC):** Not implemented, hardwired to 0.
6	RO	0b	**RSVD (RSVD):** Reserved
5	RO	0b	**66 MHz Capable (C66):** Not implemented, hardwired to 0.
4	RO	1b	**Capabilities List (CL):** Indicates the presence of a capabilities list, hardwired to 1.
	RO	0b	**Interrupt Status (IS):** Indicates the interrupt status of the device (1 = asserted).
2:0	RO	000b	**RSVD (RSVD):** Reserved

3.1.4 RID - Revision ID

B/D/F/Type: X/Y/Z/PCI
Address Offset: 8h
Default Value: 00h
Access: RO;
Size: 8 bits

Bit	Access	Default Value	Description
7:0	RO	00h	**Revision ID (RID):** Indicates stepping of the HECI host controller.

3.1.5 CC - Class Code

B/D/F/Type: X/Y/Z/PCI
Address Offset: 9-Bh
Default Value: 000000h
Access: RO;
Size: 24 bits

Bit	Access	Default Value	Description
23:16	RO	07h	**Base Class Code (BCC):** Indicates the base class code of the HECI host controller device.
15:8	RO	80h	**Sub Class Code (SCC):** Indicates the sub class code of the HECI host controller device.
7:0	RO	00h	**Programming Interface (PI):** Indicates the programming interface of the HECI host controller device.

3.1.6 CLS - Cache Line Size

B/D/F/Type: X/Y/Z/PCI
Address Offset: Ch
Default Value: 00h
Access: RO;
Size: 8 bits

Bit	Access	Default Value	Description
7:0	RO	00h	**Cache Line Size (CLS):** Not implemented, hardwired to 0.

3.1.7 MLT - Master Latency Timer

B/D/F/Type: X/Y/Z/PCI
Address Offset: Dh
Default Value: 00h
Access: RO;
Size: 8 bits

Bit	Access	Default Value	Description
7:0	RO	00h	**Master Latency Timer (MLT):** Not implemented, hardwired to 0.

3.1.8 HTYPE - Header Type

B/D/F/Type: X/Y/Z/PCI
Address Offset: Eh
Default Value: 80h
Access: RO;
Size: 8 bits

Bit	Access	Default Value	Description
7	RO	1b	**Multi-Function Device (MFD):** Indicates the HECI host controller is part of a multi-function device.

Bit	Access	Default Value	Description
6:0	RO	0000000b	**Header Layout (HL):** Indicates that the HECI host controller uses a target device layout.

3.1.9 BIST - Built In Self Test

B/D/F/Type: X/Y/Z/PCI
Address Offset: Fh
Default Value: 00h
Access: RO;
Size: 8 bits

Bit	Access	Default Value	Description
7	RO	0b	**BIST Capable (BC):** Not implemented, hardwired to 0.
6:0	RO	0000000b	**RSVD (RSVD):** Reserved

3.1.10 HECI_MBAR - HECI MMIO Base Address

B/D/F/Type: X/Y/Z/PCI
Address Offset: 10-17h
Default Value: 0000000000000004h
Access: RO; RW;
Size: 64 bits

This register allocates space for the HECI memory mapped registers defined in 3.1.

Bit	Access	Default Value	Description
63:4	RW	0000000000 00000h	**Base Address (BA):** Base address of register memory space.
3	RO	0b	**Prefetchable (PF):** Indicates that this range is not pre-fetchable
2:1	RO	10b	**Type (TP):** Indicates that this range can be mapped anywhere in 64-bit address space. Note that some chipsets may not support full 64-bit address space mapping.
0	RO	0b	**Resource Type Indicator (RTE):** Indicates a request for register memory space.

3.1.11 SS - Sub System Identifiers

B/D/F/Type: X/Y/Z/PCI
Address Offset: 2C-2Fh
Default Value: 00000000h
Access: RWO;
Size: 32 bits

Bit	Access	Default Value	Description
31:16	RWO	0000h	**Subsystem ID (SSID):** Indicates the sub-system identifier. This field should be programmed by BIOS during boot-up. Once written, this register becomes Read Only. This field can only be cleared by PLTRST#.
15:0	RWO	0000h	**Subsystem Vendor ID (SSVID):** Indicates the sub-system vendor identifier. This field should be programmed by BIOS during boot-up. Once written, this register becomes Read Only. This field can only be cleared by PLTRST#.

3.1.12 CAP - Capabilities Pointer

B/D/F/Type: X/Y/Z/PCI
Address Offset: 34h
Default Value: 50h
Access: RO;
Size: 8 bits

Bit	Access	Default Value	Description
7:0	RO	50h	**Capability Pointer (CP):** Indicates the first capability pointer offset. Itpoints to the PCI power management capability offset.

3.1.13 INTR - Interrupt Information

B/D/F/Type: X/Y/Z/PCI
Address Offset: 3C-3Dh
Default Value: 0100h
Access: RO; RW;
Size: 16 bits

Bit	Access	Default Value	Description
15:8	RO	01h	**Interrupt Pin (IPIN):** This indicates the interrupt pin the HECI host controller uses. The value of 01h selects INTA# interrupt pin, or equivalent internal signal.
7:0	RW	00h	**Interrupt Line (ILINE):** Software written value to indicate which interrupt line (vector) the interrupt is connected to. No hardware action is taken on this register.

3.1.14 MGNT - Minimum Grant

B/D/F/Type: X/Y/Z/PCI
Address Offset: 3Eh
Default Value: 00h
Access: RO;
Size: 8 bits

Bit	Access	Default Value	Description
7:0	RO	00h	**Grant (GNT):** Not implemented, hardwired to 0.

3.1.15 MLAT - Maximum Latency

B/D/F/Type: X/Y/Z/PCI
Address Offset: 3Fh
Default Value: 00h
Access: RO;
Size: 8 bits

Bit	Access	Default Value	Description
7:0	RO	00h	**Latency (LAT):** Not implemented, hardwired to 0.

3.1.16 HFS - Host Firmware Status

B/D/F/Type: X/Y/Z/PCI
Address Offset: 40-43h
Default Value: 00000000h
Access: RO;
Size: 32 bits

Bit	Access	Default Value	Description
31:0	RO	00000000h	**Firmware Status Host Access (FS_HA):** Indicates current status of the firmware for the HECI controller. Note: This register is not part of the DCMI-HI interface specification. Software should not use this register for DCMI-HI operation.

3.1.17 PID - PCI Power Management Capability ID

B/D/F/Type: X/Y/Z/PCI
Address Offset: 50-51h
Default Value: 8C01h
Access: RO;
Size: 16 bits

Bit	Access	Default Value	Description
15:8	RO	8Ch	**Next Capability (NEXT):** Indicates the location of the next capability item in the list. This is the Message Signaled Interrupts capability.
7:0	RO	01h	**Cap ID (CID):** Indicates that this pointer is a PCI power management.

3.1.18 PC - PCI Power Management Capabilities

B/D/F/Type: X/Y/Z/PCI
Address Offset: 52-53h
Default Value: C803h
Access: RO;
Size: 16 bits

Bit	Access	Default Value	Description
15:11	RO	11001b	**PME_Support (PSUP):** Indicates the states that can generate PME#.
10	RO	0b	**D2_Support (D2S):** The D2 state is not supported for the HECI host controller.
9	RO	0b	**D1_Support (D1S):** The D1 state is not supported for the HECI host controller.
8:6	RO	000b	**Aux_Current (AUXC):** Reports the maximum AUX current required when in the D3COLD state and armed for wake. Value of 000b is reported when HECI is an integrated PCI device in the chipset and already counted in system power supply budget by system designers. Additionally, in those implementations, HECI does not consume additional AUX current when armed for wake.
5	RO	0b	**Device Specific Initialization (DSI):** Indicates whether devicespecific initialization is required.
4	RO	0b	**RSVD (RSVD):** Reserved
3	RO	0b	**PME Clock (PMEC):** Indicates that PCI clock is not required to generate PME#.

Bit	Access	Default Value	Description
2:0	RO	011b	**Version (VS):** Indicates support for Revision 1.2 of the PCI Power Management Specification.

3.1.19 PMCS - PCI Power Management Control And Status

B/D/F/Type: X/Y/Z/PCI
Address Offset: 54-55h
Default Value: 0008h
Access: RWC; RO; RW;
Size: 16 bits

Bit	Access	Default Value	Description
15	RWC	0b	**PME Status (PMES):** The PME Status bit in HECI space can be set to '1' by ME FW performing a write into AUX register to set PMES. This bit is cleared by host CPU writing a '1' to it. ME cannot clear this bit. Host CPU writes with value '0' have no effect on this bit. This bit is reset to '0' by MRST#
14:9	RO	000000b	**RSVD (RSVD):** Reserved.
8	RW	0b	**PME Enable (PMEE):** This bit is read/write, under control of host SW. It does not directly have an effect on PME events. However, this bit is shadowed into AUX space so ME FW can monitor it. The ME FW is responsible for ensuring that FW does not cause the PME-S bit to transition to '1' while the PMEE bit is '0', indicating that host SW had disabled PME. This bit is reset to '0' by MRST#
7:4	RO	0000b	**RSVD (RSVD):** Reserved
3	RO	1b	**No_Soft_Reset (NSR):** This bit indicates that when the HECI host controller is transitioning from D3hot to D0 due to power state command, it does not perform and internal reset. Configuration context is pRSVD (RSVD): Reserved.
2	RO	0b	**RSVD (RSVD):** Reserved
1:0	RW	00b	**Power State (PS):** This field is used both to determine the current power state of the HECI host controller and to set a new power state. The values are: 00 - D0 state 11 - D3HOT state

Bit	Access	Default Value	Description
			The D1 and D2 states are not supported for this HECI host controller. When in the D3HOT state, the HBA's configuration space is available, but the register memory spaces are not. Additionally, interrupts are blocked. This field is visible to firmware through the H_PCI_CSR register, and changes to this field may be configured by the H_PCI_CSR register to generate an ME MSI.

3.1.20 MID - Message Signaled Interrupt Identifiers

B/D/F/Type: X/Y/Z/PCI
Address Offset: 8C-8Dh
Default Value: 0005h
Access: RO;
Size: 16 bits

Bit	Access	Default Value	Description
15:8	RO	00h	**Next Pointer (NEXT):** Indicates the next item in the list. This can be other capability pointers (such as PCI-X or PCI-Express) or it can be the last item in the list.
7:0	RO	05h	**Capability ID (CID):** Capabilities ID indicates MSI.

3.1.21 MC - Message Signaled Interrupt Message Control

B/D/F/Type: X/Y/Z/PCI
Address Offset: 8E-8Fh
Default Value: 0080h
Access: RO; RW;
Size: 16 bits

Bit	Access	Default Value	Description
15:8	RO	00h	**RSVD (RSVD):** Reserved
7	RO	1b	**64 Bit Address Capable (C64):** Specifies whether capable of generating 64-bit messages.
6:4	RO	000b	**Multiple Message Enable (MME):** Not implemented, hardwired to 0.
3:1	RO	000b	**Multiple Message Capable (MMC):** Not implemented, hardwired to 0.
0	RW	0b	**MSI Enable (MSIE):** If set, MSI is enabled and traditional interrupt

Bit	Access	Default Value	Description
			pins are not used to generate interrupts.

3.1.22 MA - Message Signaled Interrupt Message Address

B/D/F/Type: X/Y/Z/PCI
Address Offset: 90-93h
Default Value: 00000000h
Access: RW; RO;
Size: 32 bits

Bit	Access	Default Value	Description
31:2	RW	00000000h	**Address (ADDR):** Lower 32 bits of the system specified message address, always DW aligned.
1:0	RO	00b	**RSVD (RSVD):** Reserved

3.1.23 MUA - Message Signaled Interrupt Upper Address (Optional)

B/D/F/Type: X/Y/Z/PCI
Address Offset: 94-97h
Default Value: 00000000h
Access: RW;
Size: 32 bits

Bit	Access	Default Value	Description
31:0	RW	00000000h	**Upper Address (UADDR):** Upper 32 bits of the system specified message address. This register is optional and only implemented if MC.C64=1.

3.1.24 MD - Message Signaled Interrupt Message Data

B/D/F/Type: X/Y/Z/PCI
Address Offset: 98-99h
Default Value: 0000h
Access: RW;
Size: 16 bits

Bit	Access	Default Value	Description
15:0	RW	0000h	**Data (Data):** This 16-bit field is programmed by system software if MSI is enabled. Its content is driven onto the FSB during the data phase

Bit	Access	Default Value	Description
			of the MSI memory write transaction.

3.1.25 HIDM - HECI Interrupt Delivery Mode

B/D/F/Type: X/Y/Z/PCI
Address Offset: A0h
Default Value: 00h
Access: RW;
Size: 8 bits
BIOS Optimal Default 00h
This register is used to select interrupt delivery mechanism for HECI to Host CPU interrupts.

Bit	Access	Default Value	Description
7:2	RO	0h	**Reserved (RSVD)**
1:0	RW	00b	**HECI Interrupt Delivery Mode (HIDM):** These bits control what type of interrupt the HECI will send when ME writes to set the M_IG bit in AUX space. They are interpreted as follows: 00: Generate Legacy or MSI interrupt 01: Generate SCI 10: Generate SMI Note: Only 00b (Legacy or MSI interrupt) is supported for the DCMI-HI interface.

3.2 HECI Host Memory Mapped Registers

Table 3-1 below summarizes the memory mapped IO (MMIO) registers that are decoded in the memory range specified by HECI_BAR0. These registers are reset by PLTRST# unless otherwise specified in the register bit field descriptions.

Table 3-1. HECI_BAR0 MMIO Register Summary

Register Name	Register Symbol	Register Start (hex)	Register End (hex)	Default Value	Access
Host Circular Buffer Write Window	H_CB_WW	0	3	00000000h	W;
Host Control Status	H_CSR	4	7	02000000h	RO; RW; RWC;
ME Circular Buffer Read	ME_CB_RW	8	B	FFFFFFFFh	RO;

Register Name	Register Symbol	Register Start (hex)	Register End (hex)	Default Value	Access
Window					
ME Control Status Host Access	ME_CSR_HA	C	F	02000000h	RO;

3.2.1 H_CB_WW - Host Circular Buffer Write Window

B/D/F/Type: X/Y/Z/MMIO
Address Offset: 0-3h
Default Value: 00000000h
Access: W;
Size: 32 bits

This register is for host to write into its Circular Buffer (H_CB). The host's circular buffer is located at the ME subsystem address specified in the Host CB Base Address register.

Bit	Access	Default Value	Description
31:0	W	00000000h	**Host Circular Buffer Write Window Field (H_CB_WWF):** This bit field is for host to write into its circular buffer. The host's circular buffer is located at the ME subsystem address specified in the Host CB Base Address register. This field is write only, reads will return arbitrary data. Writes to this register will increment the H_CBWP as long as ME_RDY is 1. When ME_RDY is 0, writes to this register have no effect and are not delivered to the H_CB, nor is H_CBWP incremented.

3.2.2 H_CSR - Host Control Status

B/D/F/Type: X/Y/Z/MMIO
Address Offset: 4-7h
Default Value: 02000000h
Access: RO; RW; RWC;
Size: 32 bits

This register reports status information about the host circular buffer (H_CB) andallows host software to control interrupt generation. Note to software: reserved bits in this register must be set to 0 whenever this register is written.

Bit	Access	Default Value	Description
31:24	RO	02h	**Host Circular Buffer Depth (H_CBD):** This field indicates the maximum

Bit	Access	Default Value	Description
			number of 32 bit entries available in the host circular buffer (H_CB). Host software uses this field along with the H_CBRP and H_CBWP fields to calculate the number of valid entries in the H_CB to read or # of entries available for write. This field is programmed by the ME firmware during ME initialization. Programmer's note: This field is implemented with a "1-hot" scheme. Only one bit will be set to a "1" at a time. Each bit position represents the value n of a buffer depth of (2^n). For example, when bit# 0 is 1, the buffer depth is 1; when bit#1 is 1, the buffer depth is 2, etc. The allowed buffer depth values are 2, 4, 8, 16, 32, 64 and 128. **Note:** This is fixed to 64 for DCMI-HI
23:16	RO	00h	**Host CB Write Pointer (H_CBWP):** Points to next location in the H_CB for host to write the data. Software uses this field along with H_CBRP and H_CBD fields to calculate the number of valid entries in the H_CB to read or number of entries available for write.
15:8	RO	00h	**Host CB Read Pointer (H_CBRP):** Points to next location in the H_CB where a valid data is available for embedded controller to read. Software uses this field along with H_CBWR and H_CBD fields to calculate the number of valid entries in the host CB to read or number of entries available for write.
7:5	RO	000b	**RSVD (RSVD):** Reserved
4	RW	0b	**Host Reset (H_RST):** Setting this bit to 1 will initiate a HECI reset sequence to get the circular buffers into a known good state for host and ME communication. When this bit transitions from 0 to 1, hardware will clear the H_RDY and ME_RDY bits.
3	RW	0b	**Host Ready (H_RDY):** This bit indicates that the host is ready to process messages.
2	RW	0b	**Host Interrupt Generate (H_IG):** Once message(s) are written into its CB, the host sets this bit to one for the HW to set the ME_IS bit in the ME_CSR and to generate an interrupt message to ME. HW will send the interrupt message to ME only if the ME_IE is enabled. HW

Bit	Access	Default Value	Description
			then clears this bit to 0.
1	RWC	0b	**Host Interrupt Status (H_IS):** HW sets this bit to 1 when ME_IG bit is set to 1. Host clears this bit to 0 by writing a 1 to this bit position. H_IE has no effect on this bit.
0	RW	0b	**Host Interrupt Enable (H_IE):** Host sets this bit to 1 to enable the host interrupt (INTR# or MSI) to be asserted when H_IS is set to 1.

3.2.3 ME_CB_RW - ME Circular Buffer Read Window

B/D/F/Type: X/Y/Z/MMIO
Address Offset: 8-Bh
Default Value: FFFFFFFFh
Access: RO;
Size: 32 bits

This register is for host to read from the ME Circular Buffer (ME_CB). The ME's circular buffer is located at the ME subsystem address specified in the ME CB Base Address register.

Bit	Access	Default Value	Description
31:0	RO	FFFFFFFFh	**ME Circular Buffer Read Window Field (ME_CB_RWF):** This bit field is for host to read from the ME Circular Buffer. The ME's circular buffer is located at the ME subsystem address specified in the ME CB Base Address register. This field is read only, writes have no effect. Reads to this register will increment the ME_CBRP as long as ME_RDY is 1. When ME_RDY is 0, reads to this register have no effect, all 1s are returned, and ME_CBRP is not incremented.

3.2.4 ME_CSR_HA - ME Control Status Host Access

B/D/F/Type: X/Y/Z/MMIO
Address Offset: C-Fh
Default Value: 02000000h
Access: RO;
Size: 32 bits

This register allows host software to read the ME Control Status register (ME_CSR). This register is reset by MERST#.

Bit	Access	Default Value	Description
31:24	RO	02h	**ME Circular Buffer Depth Host Read Access (ME_CBD_HRA):** Host read only access to ME_CBD.
23:16	RO	00h	**ME CB Write Pointer Host Read Access (ME_CBWP_HRA):** Host read only access to ME_CBWP.
15:8	RO	00h	**ME CB Read Pointer Host Read Access (ME_CBRP_HRA):** Host read only access to ME_CBRP.
7:5	RO	000b	**RSVD (RSVD):** Reserved
4	RO	0b	**ME Reset Host Read Access (ME_RST_HRA):** Host read access to ME_RST.
3	RO	0b	**ME Ready Host Read Access (ME_RDY_HRA):** Host read access to ME_RDY.
2	RO	0b	**ME Interrupt Generate Host Read Access (ME_IG_HRA):** Host read only access to ME_IG.
1	RO	0b	**ME Interrupt Status Host Read Access (ME_IS_HRA):** Host read only access to ME_IS.
0	RO	0b	**ME Interrupt Enable Host Read Access (ME_IE_HRA):** Host read only access to ME_IE.

4 Theory of Operation

4.1 Resets and Initialization

This section describes the reset and initialization flows for the HECI interface.

4.2 HECI Interface Reset Flows

The following sections describe the reset flows of the HECI interface to get the HECI interface to a known state from any unknown state and establish a starting point for HECI message communication. These reset flows describe a interface reset sequence performed by host software and firmware cooperation, and does not refer to a hardware reset sequence such as PLTRST# or MERST#.

4.3 Host initiating the reset of the HECI Interface

In this scenario, the host software driver or BIOS has loaded and desires to get the HECI message buffers in a known good state prior to initiating message transfers. Host software must perform this sequence after PLTRST# is asserted and before sending any messages through the HECI interface. Host software may also perform this sequence any time messages through HECI have become out of sync or the interface is in an unknown state.

Step	Peformed by
1. Perform PCI initialization (only needed after PLTRST#)	Host
2. Host terminates its own reads and writes to HECI, will not initiate any until this sequence completes. Not needed after PLTRST#.	Host
3. Host writes to the H_CSR register, setting both the H_RST bit and the H_IG bit to 1. Host software may also set H_IS and H_IE to 1 in order to receive an interrupt when ME FW sets the ME_RDY bit to 1.When H_RST transitions from 0 to 1, hardware clears the ME_RDY and H_RDY bit to 0. Note that for the first HECI device the setting of H_RST is not needed nor desired after PLTRST#. This will allow the ME firmware to make the HECI interface ready without needing to wait for the BIOS to set H_RST. If the BIOS sets H_RST during a POST sequence where the ME is also coming out of MOff, the BIOS may experience a long (>100ms) delay prior to the ME firmware being able to service the H_RST request. For the second HECI device, host software must set H_RST after a PLTRST#.	Host
4. Host reads the H_CSR once to ensure that the posted write to H_CSR in step 3 completes.	Host
5. If Host software did not set the H_IE to 1 in step 3, then host software now begins reading the ME_CSR_HA MMIO register, polling for ME_RDY bit to get set to 1	Host
6. The setting of the H_IG bit in step 3 causes hardware to set ME_IS and signal an interrupt to the ME if ME_IE is set to 1. If the interrupt is not enabled, then the firmware must periodically poll the H_RST_MERA bit to check for the host initiating a reset indicating it is ready to perform transactions over the HECI interface. ME	ME

Step	Peformed by
firmware should also initiate this sequence as a result of PLTRST# or MRST#.	
7. Servicing the setting of the ME_IS bit, the ME FW ISR reads H_CSR_MEA register and determines that the host has set its H_RST bit and is requesting an interface reset. The ME FW ISR then signals it has entered a host reset sequence to the main ME FW HECI reset handler.	ME
8. The ME FW HECI reset handler writes the ME_CSR to set ME_RDY and ME_IE bits to 0, and sets the ME_RST bit to 1.	ME
9. The ME FW HECI reset handler writes the PTR_RST bit in the ME CSR to clear all 4 pointers (H_CBRP, H_CBWP, ME_CBRP, ME_CBWP).	ME
10. Hardware will clear the PTR_RST bit after it has accomplished the reset of the pointers to all 0's. This interlocking is done to ensure that we account for any clock crossing delays prior to moving to the next step.	ME
11. FW reads PTR_RST bit until it observes it = 0. This should occur within 10 cycles. Note: see section 4.1.1.4 for special restrictions.	ME
12. At this point the HECI interface is idle and FW should write to the ME_CBD and H_CBD_MERWA fields to change the circular buffer depths if necessary.	ME
13. FW writes the ME_CSR to set ME_RDY, ME_IE, and ME_IG bits to 1, and clears the ME_RST bit to 0.	ME
14. Since step 5 the host has either been polling ME_RDY_HRA bit or waiting for a HECI host interrupt. Step 13 has set both the ME_RDY_HRA bit and the H_IS bit, so either of these two conditions will be satisfied now and host software will proceed.	Host
15. Host software writes the H_CSR to clear H_RST to 0 and set H_RDY and H_IG to 1. Host software is now ready to begin sending messages via the Host circular buffer.	Host
16. The setting of H_IG in step 15 causes the hardware to signal an interrupt to the ME.FW ISR reads H_CSR_MEA and observes that the H_RDY bit is now 1. ME firmware is now ready and permitted to begin sending messages via the ME circular buffer.	ME

NOTES:

1. If the ME FW code that was interrupted in step 6 was doing writes to HECI, then we are going to resume back into that code in step 7. This means that code will continue to do its writes to ME_CB_WW until it finished that message. This is OK since the ME_RDY bit is now 0 and writes are discarded by the hardware. An alternative is for FW to clear the ME_IE bit to 0 temporarily while FW is doing the HECI writes. Then, when the ISR code executes, we can do the reset sequence and know that we are not returning to a sequence of in progress writes that were interrupted.

2. If the ME FW code was doing HECI reads at the time of step 6, then FW was already in the HECI interrupt routine and step 6 will not proceed until those HECI reads have finished. Those reads should be discarded at the end of the HECI ISR when it reads the H_RDY_MERA bit and observes it is 0. See section 4.2.5.2 for ME FW HECI ISR flow.

4.4 ME Initiated reset of the HECI interface.

In this scenario the ME has been reset (either intentionally by some software /firmware request (like a firmware update) or unintentionally by a firmware watchdog timer time out.

Step	Peformed by
1. Perform ME register initialization to set up the CB base addresses (only needed after MERST#)	ME
2. ME FW terminates its own reads and writes to HECI, will not initiate any until this sequence completes. (Not needed after MERST#.)	ME
3. ME clears ME_RDY to 0 and sets the ME_RST and ME_IG bits to 1.	ME
4. When host software loads or if it is already loaded, it reads the ME_CSR_HA and observes ME_RST_HRA is set to 1. Host software now	Host

Step	Peformed by
initiates and follows the reset sequence described in section 4.3	

4.4.1 PTR_RST bit

The PTR_RST bit indicates a reset of both the host circular buffer pointers and the ME circular buffer pointers. The PTR_RST bit will only clear to 0 when the host is powered and not in reset.

4.4.2 HECI Interface Reset Timeout

A HECI interface reset can occur at any time during ME firmware initialization or normal operation. During the ME firmware boot there are periods of time when the firmware is in an initialing state where it will not respond to a HECI interface reset in a timely manner. It is recommended that host software implement a 15 second timeout on waiting for the ME_RDY bit to be set to 1 after initiating a HECI interface reset.

After initiating a HECI interface reset if the ME_RDY bit is not set to 1 within 15 seconds then host software should consider the HECI interface in an error state. It should then read the FW_STATUS register get the state of the ME firmware and take additional actions as required.

4.4.3 Programming the Circular Buffer Depth Fields

Host software and firmware must ensure that the HECI interface is idle (i.e. neither host nor ME firmware is reading from or writing to the circular buffers) prior to changing the circular buffer depth fields. Hardware does not enforce this restriction and if the HECI interface is not idle when either of the circular buffer depth fields are changed, the resulting behavior is unknown.

4.5 Driver Load or restart in the Host OS

A driver load/restart may be triggered by actions such as an add/remove in Device Manager. The host driver should not assume the HECI interface is in a known good state at the time of driver load. The driver should follow the Host initiated reset flow described in section 4.3

4.6 Driver shutdown, stop, or power management events.

Prior to any driver shutdown, stop or placing the device into a D-state other than D0, the driver should ensure that all pending messages are completed and stop sending new messages to the HECI interface according to the software specification for HECI messages, and then clear the H_RDY bit to 0 and set H_IG to let the firmware know it is no longer monitoring the HECI interface for messages.

4.7 ME Firmware initiated reset of ME partition, in S0

An MERST# is the signal used by ME firmware to internally to reset the ME system, but not the host system. MERST# will reset ME_CSR, clearing ME_RDY to 0, which causes hardware to discard all CB reads and writes, and setting the host circular buffer depth back to its default.

• MERST# will not affect the Host HECI PCI configuration registers in any way.

• MERST# will not affect the Host HECI MMIO registers except as listed above.

• H_CSR bits remain unchanged with the exception of H_CBD field as mentioned above.

ME FW will not generate or process any received messages until the ME initiated HECI interface reset sequence described in 4.4 completed.

4.8 Host Soft Reset (CF9h reset, still in S0)

HRST# is the signal used internally to reset the host system, but not the ME system. HRST# will reset all of the HECI PCI configuration registers. HRST# will also reset all of the HECI host MMIO registers, including:

- H_CSR, including:
  - H_RDY, H_CBWP, H_IS, H_IG
  - H_IE – HECI interrupt messages to the host are disabled.
  - H_CBD RO field is unchanged as it is a reflection of H_CBD_MERWA which is reset by MERST#.
- HECI ME MMIO ME_CSR ME_CBRP

Host SW will not generate or process any received messages until the HECI reset sequence described in section 4.3 is accomplished.

4.9 FW Status Register

The host visible HECI PCI device has a 32bit firmware status register located at offset 40h in its PCI configuration space. The ME firmware will write status information about the state of the HECI firmware into this register.

Note: This register is not part of the DCMI-HI interface definition and should not be used by host software. The definitions of the 32bit firmware status register vary from chipset to chipset.

4.10 Runtime Operation

This section describes the aspects of the runtime operation of the HECI interface. This section focuses on how host software (SW) and ME firmware (FW) will use the HECI hardware interface registers. This section does not describe the messaging formats and protocols used by SW and FW in the HECI messages passed through the circular buffers.

4.10.1 Calculation of Filled Slots in a HECI Circular Buffer

The procedure for calculating the number of filled slots (entries) in a HECI circular buffer is the same for both the host circular buffer and ME circular buffer. The following steps allow calculation of the number of filled slots in a circular buffer, where "X" can be replaced with either "H" for host circular buffer or "ME" for ME circular buffer.

1. Cast ReadPointer, WritePointer as signed 8 bit quantity (-128 to +127).
2. Cast BufferDepth as an unsigned 8 bit quantity.
3. Cast FilledSlots as an unsigned 8 bit quantity.
4. Read X_CSR.
5. Store X_CSR.X_CBRP in ReadPointer.
6. Store X_CSR.X_CBWP in WritePointer.
7. Store X_CSR.X_CBD in BufferDepth.
8. FilledSlots = (WritePointer – ReadPointer).

The following conditions may now be tested:

- If FilledSlots == 0, then the circular buffer is empty.
- If FilledSlots == BufferDepth, then the circular buffer is full.
- If FilledSlots > BufferDepth, then a circular buffer overflow occurred.

4.10.2 Calculation of Empty Slots in a HECI Circular Buffer

The procedure for calculating the number of empty slots (entries) in a HECI circular buffer is the same for both the host circular buffer and ME circular buffer. The following steps allow calculation of the number of empty slots in a circular buffer, where "X" can be replaced with either "H" for host circular buffer or "ME" for ME circular buffer.

1. Cast EmptySlots as an unsigned 8 bit quantity.

2. Perform steps listed in section 4.10.1

3. EmptySlots = (BufferDepth – FilledSlots).

To aid IA32 BIOS developers in correct implementation of empty and filled slots in IA32 assembly, the following is example code for these steps:

```
; Routine: HeciGetSlots
; Input: ES:EDI = MMIO address of either H_CSR or ME_CSR_HA
; Output: Carry Flag Set = Buffer overflow error
; Carry Flag Clear = No error
; AL = FilledSlots
; AH = EmptySlots
; Registers Modified: EAX
; Notes:
; * The caller should always check that the carry
; flag is not set before using either the
; EmptySlots or FilledSlots output values.
HeciGetSlots:
     mov eax, DWORD PTR es:[edi]
     shr eax, 8 ; AH = WritePointer
; AL = ReadPointer
     sub ah, al ; WP – RP
     shr eax, 8 ; AL = FilledSlots
; AH = BufferDepth
     sub ah, al ; AH = EmptySlots
; Set/Clear Carry Flag
     ret
```

4.10.3 Host Sending Message to ME Flow

The following flow describes the basic HECI operation of sending a message from the host to the ME. The message referred to in this flow may be a complete message that fits within the HECI circular buffer or a message packet, see section 6.1 for more details on message packets.

Step	Peformed by
1. Host software reads the H_CSR register to determine the number of empty slots available in its circular buffer, as described in section 4.2.2.	Host
2. If there is not enough empty slots available in the host circular buffer, the host queues the request in an internal software queues and waits until there is room in the host circular buffer.	Host
3. Once there is enough empty slots available in the host circular buffer, the host writes the message into the host circular buffer. To accomplish this, the host sequentially writes each dword of the message to the H_CB_WW (Host Circular Buffer Write Window) MMIO register. Note that the host is permitted to write more than one message / message packet into the host circular buffer in this step, as long as there is enough room in the host circular buffer. This reduces the interrupt overhead of the HECI interface.	Host
4. Host sets the H_IG (Host Interrupt Generate) bit in the H_CSR register to 1, preserving the state of other bits, to generate an interrupt to the ME.	Host
5. Host reads the ME_CSR_HA register and checks that the ME_RDY bit is set to 1. If the ME_RDY bit is clear (0), a fatal error occurred during transmission of the message. The host must re-enter the initialization	Host

Step	Peformed by
flow, see section 4.1.1.2.	
6. Upon the setting of the H_IG bit in step 4, hardware sets the ME_IS and generates an interrupt to the ME firmware. The hardware then clears the H_IG bit.	ME
7. FW clears the ME_IS bit in the ME_CSR by writing a 1 to the ME_IS bit position in the ME_CSR. The remainder of the FW actions may be done at interrupt service context or in a thread depending upon FW implementation.	ME
8. FW reads the H_CSR_MEA and determines the number of filled slots in the circular buffer available for reading (HostFilledSlots), as described in section 4.2.1.	ME
9. If the host circular buffer is empty, the host has not sent a message. This interrupt is for another reason. See section 4.2.5 for HECI interrupt causes.	ME
10. If the host circular buffer has overflowed, the host has written too many slots in the circular buffer. This is an error on the part of host software. The ME firmware should initiate a reset of the HECI interface as described in section 4.1.1.2.	ME
11. FW reads the H_CBRW (Host Circular Buffer Read Window) register HostFilledSlots times to get the current message data and moves it to a temporary buffer for dispatch to the appropriate FW client.	ME
12. FW reads the H_CSR_MEA. If the H_RDY bit is 0, then a host reset occurred during the transaction and the message should be discarded as bad data may have been retrieved from the host's circular buffer.	ME
13. FW decodes the message header and signals the appropriate FW client that a message is ready for it to retrieve.	ME
14. FW sets the ME_IG (ME Interrupt Generate) bit in the ME_CSR register to 1, preserving the state of other bits. This is done to signal the host that the ME has consumed HostFilledSlots from the host circular buffer.	ME

4.10.4 ME Sending Message to Host Flow

The following flow describes the basic HECI operation of sending a message from the ME to the host. The message referred to in this flow may be a complete message that fits within the HECI circular buffer or a message packet, see section 6.1 for more details on message packets.

Step	Peformed by
1. FW reads the ME_CSR register to determine the number of empty slots available in its circular buffer, as described in section 4.2.2.	ME
2. If there is not enough empty slots available in the ME circular buffer, the FW queues the request in an internal FW queues and waits until there is room in the ME circular buffer.	ME
3. Once there is enough empty slots available in the ME circular buffer, the FW writes the message into the ME circular buffer. To accomplish this, the FW sequentially writes each dword of the message to the ME_CB_WW (ME Circular Buffer Write Window) AUX register. Note that the FW is permitted to write more than one message / message packet into the ME circular buffer in this step, as long as there is enough room in the ME circular buffer. This reduces the interrupt overhead of the HECI interface.	ME
4. FW sets the ME_IG (ME Interrupt Generate) bit in the ME_CSR register to 1, preserving the state of other bits, to generate an interrupt to the host.	ME
5. FW reads the H_CSR_MEA register and checks that the H_RDY bit is set	ME

Step	Peformed by
to 1. If the H_RDY bit is clear (0), a fatal error occurred during transmission of the message. The ME must re-enter the initialization flow, see section 4.1.1.2.	
6. Upon the setting of the ME_IG bit in step 4, hardware sets the H_IS and generates an interrupt to the host software. The hardware then clears the ME_IG bit.	ME
7. Host software (SW) clears the H_IS bit in the H_CSR by writing a 1 to the H_IS bit position in the H_CSR. The remainder of the FW actions may be done at interrupt service context or in a thread depending upon SW implementation.	Host
8. SW reads the ME_CSR_HRA and determines the number of filled slots in the circular buffer available for reading (MEFilledSlots), as described in section 4.2.1.	Host
9. If the ME circular buffer is empty, the ME has not sent a message. This interrupt is for another reason. See section 4.2.5 for HECI interrupt causes.	Host
10. If the ME circular buffer has overflowed, the ME has written too many slots in the circular buffer. This is an error on the part of FW. The SW should do initiate a HECI interface reset as described in section 4.1.1.1.	Host
11. SW reads the ME_CBRW (ME Circular Buffer Read Window) register MEFilledSlots times to get the current message data and moves it to a temporary buffer for dispatch to the appropriate SW driver.	Host
12. SW reads the ME_CSR_HA. If the ME_RDY bit is 0, then a ME reset occurred during the transaction and the message should be discarded as bad data may have been retrieved from the ME's circular buffer.	Host
13. SW decodes the message header and signals the appropriate SW driver that a message is ready for it to retrieve.	Host
14. SW sets the H_IG (Host Interrupt Generate) bit in the H_CSR register to 1, preserving the state of other bits. This is done to signal the FW that the SW has consumed MEFilledSlots from the host circular buffer.	Host

4.10.5 Interrupt Handling

This section describes the expected operation of the SW and FW interrupt service routines. For the HECI interface, there are several possible reasons the HECI interrupt has been asserted which are the same for both host software and ME firmware ISRs:

- A message has just been written to the circular buffer.
- A message has just been read from the circular buffer.
- The host or ME has initiated an interface reset sequence.
- The host or ME has completed its portion of the interface reset sequence and normal operation can begin.

4.10.5.1 Host ISR Flow

This section describes the flow for the host software interrupt service routine.

Step	Peformed by
1. Upon FW setting the ME_IG bit in the ME_CSR, hardware sets the H_IS and generates an interrupt to the host software if H_IE is set to 1. The hardware then	ME

Step	Peformed by
clears the ME_IG bit.	
2. Host software (SW) reads the H_CSR register and writes the value read back to the H_CSR register. This clears the H_IS bit in the H_CSR to 0 because a value of 1 was written to the H_IS bit position in the H_CSR.	Host
3. SW reads the ME_CSR_HA	Host
4. If ME_RDY_HRA = 0 then the FW is requesting a HECI interface reset. a. Signal the main HECI driver to initiate a HECI interface reset sequence as described in section 4.1.1.2. b. Exit the ISR	Host
5. Determine the number of filled slots in the ME circular buffer available for reading (MEFilledSlots), as described in section 4.2.1.	Host
6. If the ME circular buffer is empty, the ME has not sent a message, this interrupt indicates the ME has completed reading the message(s) last transmitted by the host. a. The host software ISR should signal the main host SW HECI driver that more slots are available (empty) in the host HECI buffer for additional message transmission. b. Exit the ISR	Host
7. If the ME circular buffer has overflowed, the host has written too many slots in the circular buffer. This is an error on the part of SW. a. The SW should enter the HECI interface initialization flow, see section 4.1.1.1. b. Exit the ISR	Host
8. SW reads the ME_CBRW (ME Circular Buffer Read Window) register MEFilledSlots times to get the current message data and moves it to a temporary buffer for dispatch to the appropriate SW driver.	Host
9. SW decodes the message header and signals the appropriate SW driver that a message is ready for it to retrieve.	Host
10. SW sets the H_IG (Host Interrupt Generate) bit in the H_CSR register to 1, preserving the state of other bits. This is done to signal the FW that the SW has consumed MEFilledSlots from the host circular buffer.	Host
11. The host software ISR should signal the main host SW HECI driver that more slots may be available (empty) in the host HECI buffer for message transmission.	Host
12. SW exits the interrupt service routine.	Host

4.11 Power Management

4.11.1 Power State Support

The HECI device supports D0 and D3 power states for compliance with PCI Power Management and ACPI specifications. As a integrated PCI device in the GMCH, there is no difference in the power consumption in the S0 operational state between D0 and D3, therefore there is little value to implementing an idle timeout in the host driver software to put the device in D3 while system is in S0 running state.

5 Link Layer

5.1 Host Link Ready State

From the Host side link is ready only when all the following conditions are true:

1. Host Memory Space Enable bit is 1.
2. Host Power State bits are 0.
3. Host BAR is set.
4. Host Reset (H_RST) bit is 0.
5. Host Ready (H_RDY) bit is 1.
6. ME Ready (ME_RDY) bit is 1.

5.2 ME Link Ready State

From the ME side link is ready only when all the following conditions are true:

1. Circular Buffer Depth bits have one bit set.
2. ME Reset (ME_RST) bit is 0.
3. ME Ready (ME_RDY) bit is 1.
4. Host Ready (H_RDY) bit is 1.

5.3 Host Interrupt Generation Rules

Host must set Host Interrupt Generate (H_IG) bit every time when one of the following events happen:

1. Host changes H_RST bit.
2. Host changes H_RDY bit.
3. Host completes reading of data.
4. Host completes writing of data.

5.4 ME Interrupt Generation Rules

ME must set ME Interrupt Generate (ME_IG) bit every time when one of the following events happen:

1. ME changes ME_RST bit.
2. ME changes ME_RDY bit.
3. ME completes reading of data.
4. ME completes writing of data.

5.5 Link Layer Errors

When in the Link Ready state, the following table lists link errors that can occur, and their resultant actions.

Error	Action Taken
ME or Host Circular Buffer overflow.	Detecting agent (Host or ME) resets HECI interface.
ME clears ME_RDY or Host clears H_RDY	Detecting agent (Host or ME) resets HECI interface.

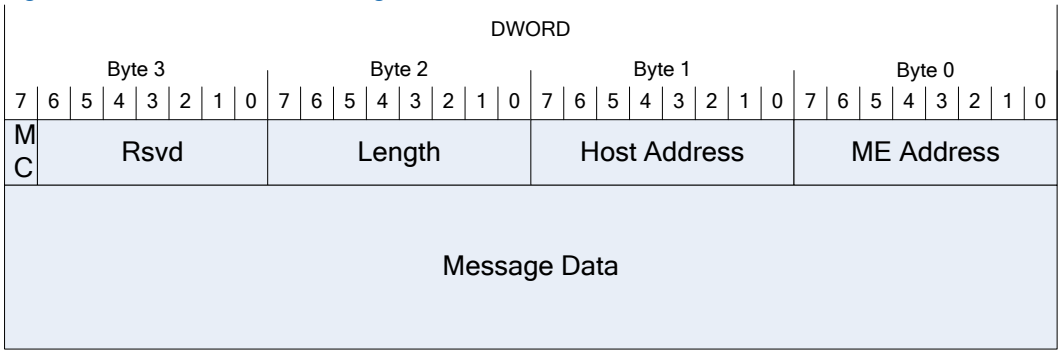
6 Transaction Layer

6.1 General Message Format

A general message format is used to pass data in the HECI circular buffers. The same message format is used for messages from the host to the ME and from the ME to the host. The message consists of a generic header and a data payload. The format and contents of the data payload are opaque to the host and firmware HECI drivers, and are only interpreted by the associated clients in the host and firmware domains.

Figure 6-1 shows the general message format of a message placed in a HECI circular buffer. The message consists of a message header (the first dword) and the message data. Messages that are longer than the HECI circular buffer are broken up into a series of "message packets."

Figure 6-1 - General HECI Message Format



The HECI message header is the 1st dword of any HECI message. It is described with the following C prototype:

Prototype

```
typedef struct _HECI_MESSAGE_HEADER
{
    UINT32 MEAddress:8;
    UINT32 HostAddress:8;
    UINT32 Length:9;
    UINT32 Reserved:6;
    UINT32 MessageComplete:1;
} HECI_MESSAGE_HEADER;
```

7 Bus Message Layer

The host and ME HECI driver implement a message protocol between the two drivers that assists in overall HECI messaging operation. This bus messaging layer performs functions such as HECI driver start and stop messages, flow control, and power management messages.

7.1 HECI Bus Message Command Summary

Table 7-1 summarizes the messages in version 0x0001 of the HECI Bus Message Protocol.

Table 7-1. HECI Bus Message Command Summary, Version 0x0001

Code	Name	Initiator	Response Required	Timeout
0x01	Host Version Request	Host	Yes	Standard
0x02	Host Stop Request	Host	Yes	Standard
0x03	ME Stop Request	ME	No	Standard
0x04	Host Enumeration Request	Host	Yes	Standard
0x05	Host Client Properties Request	Host	Yes	Standard
0x06	Client Connect Request	Host	Yes	Standard
0x07	Client Disconnect Request	Host or ME	Yes	Standard
0x08	Flow Control	Host or ME	No	Standard
0x09	Client Connection Reset Request	Host	Yes	Standard

7.2 General Format of HECI Bus Messages

All HECI bus messages are implemented using the HECI message format described in section 6.1; The HostAddress and MEAddress fields in the HECI_MESSAGE_HEADER will be 0x0 and 0x0, respectively.

HECI bus messages are formatted according the following C prototype:

Prototype

```
typedef struct _HBM_COMMAND
{
    UINT8 Command:7;
    UINT8 IsResponse:1;
} HBM_COMMAND;
typedef struct _HECI_BUS_MESSAGE
{
    HBM_COMMAND Command;
    UINT8 CommandSpecificData[];
} HECI_BUS_MESSAGE;
```

Element	Description
Command	The Command element describes the major function of this HECI bus message. The commands are enumerated in the sections below.
IsResponse	This indicates whether this HECI bus message is a direct response to a previously received HECI bus message.
CommandSpecificData	This is the command specific data, whose length is defined by the command code.

All HECI bus message structure prototypes are prefixed with "HBM_"

7.3 Sequencing of HECI Bus Messages

HECI bus messages are not necessarily responded to in order by ME firmware. For example, the host may send a Client Connect Request message to ME client X, followed immediately by a Host Enumeration Request message. The Host Enumeration Response message may be sent by the ME to the host prior to the Client Connect Response message for ME client X.

7.4 Timeouts for HECI Bus Message Responses

Unless otherwise specified, all HECI bus request messages must be responded to within 15 seconds, if a response message is required. This is the standard timeout referenced in Table 7-1.

7.5 HECI Bus Message Protocol Versioning

The HECI bus message protocol has a versioning mechanism to allow the host HECI driver and ME HECI driver to arbitrate to the proper level of protocol interface versions. This versioning mechanism is described in the Host Version Request message and Host Version Response message. The HECI bus message protocol version number consists of a major version number and a minor version number, defined with the following C prototype:

Prototype

```
typedef struct _HBM_VERSION
{
    UINT8 MinorVersion;
    UINT8 MajorVersion;
} HBM_VERSION;
```

Element	Description
MinorVersion	The minor version of the HECI bus message protocol definition. This field will be incremented whenever a change is made to the HECI bus message protocol that is backwards compatible with previous versions' definition.
MajorVersion	The major version of the HECI bus message protocol definition. This field will be incremented whenever a change is made that breaks backwards compatibility for host software or ME firmware using this protocol.

7.6 Host Version Request

This message is typically the first message sent over the HECI interfaces by host software after the HECI interface reset sequence described in section 4.1.1 is completed. It is recommended that operating system drivers always send this message after a HECI interface reset sequence. Platform BIOS firmware, however, may opt to not send this message is it is shipped with the platform along with the ME firmware. This message includes the HECI bus message interface version supported by the host driver.

After the HECI interface reset sequence described in section 4.1.1 is completed, but prior to receiving this message, the HECI bus message protocol will operate at version 0x0001.

Response Message Required: Host Version Response

Prototype

```
typedef struct _HBM_HOST_VERSION_REQUEST
{
    HBM_COMMAND Command;
    UINT8 Reserved;
    HBM_VERSION HostVersion;
} HBM_HOST_VERSION_REQUEST;
```

Element	Description
Command	0x01
Reserved	Reserved for future use.

Element	Description
HostVersion	The highest version of the HECI bus message protocol supported by the host driver. The host driver must be compatible with all versions of the HECI bus message protocol definition that are less than HostVersion

7.7 Host Version Response

This message returns the HECI bus message interface version supported by the ME firmware.

Response Message Required: None

Prototype

```
typedef struct _HBM_HOST_VERSION_RESPONSE
{
    HBM_COMMAND Command;
    UINT8 HostVersionSupported;
    HBM_VERSION MEMaxVersion;
} HBM_HOST_VERSION_RESPONSE;
```

Element	Description
Command	0x81
HostVersionSupported	A value of 0x01 (TRUE) indicates whether the ME firmware can support and will now operate at HostVersion supplied by the Host Version Request message. A value of 0x00 (FALSE) indicates the ME firmware can not support the version requested by the host.
MEVersion	Indicates the highest HECI bus message protocol definition version supported by the ME firmware.

If the ME firmware can not support an older version of the HECI bus message protocol, the ME firmware will indicate so by filling in the HostVersionSupported field with the value of FALSE. This may occur because either the ME firmware does not support an earlier version of the interface or the ME firmware does not have support for a newer version of the protocol supported by the host driver. The ME firmware will always fill in the MEVersion element with the highest HECI bus message protocol supported by the ME firmware.

If the HostVersionSupported field contains FALSE, host software should do the following:

1. If host software can support the MEVersion, it should proceed with the HECI bus messaging normal operation at the specified MEVersion.

2. If host software can not support the MEVersion (i.e. MEVersion is greater than HostVersion), host software should send a Host Stop Message and terminate HECI messaging. It is recommended that the driver request the administrator of the system for a driver update.

The desired behavior of the Host Version Request message exchange can be summarized in Table 7-2.

Table 7-2. Table 7-2 - HECI Bus Message Protocol Version Selection

Host Version vs. ME Version	HostVersion Supported	Resultant Version	Comments
HostVersion = MEVersion	TRUE	HostVersion	
HostVersion > MEVersion	FALSE	MEVersion	Host software is required to be backwards compatible with older HECI interfaces, and therefore must support MEVersion.
HostVersion.Major = MEVersion.Major HostVersion.Minor < MEVersion.Minor	TRUE	HostVersion	The ME firmware can use the HostVersion of the HECI protocol as its MEVersion of the HECI protocol is backwards compatible with HostVersion.
HostVersion.Major < MEVersion.Major	TRUE / FALSE	HostVersion / Stop	If the ME firmware supports the older version of the HECI protocol, it can return TRUE in the HostVersionSupported field and operate at HostVersion. If the ME firmware does not support the older version of the HECI protocol, it can return FALSE in the HostVersionSupported field. In this case, host software must send a HBM_HOST_STOP_REQUEST message and terminate communication over the HECI interface.

7.8 Host Stop Request

This message must be the last message sent by the host driver over the HECI interfaces to ME firmware in a orderly shutdown of the host interface. Unorderly shutdown or reset events may still occur due to asynchronous events such as CPU triple fault reset events. The host driver should send this message as part of its normal low power state (D1, D2, or D3) entry sequence, and as part of the driver's shutdown / stop sequences.

Response Message Required: Host Stop Response

Prototype

```
typedef struct _HBM_HOST_STOP_REQUEST
{
    HBM_COMMAND Command;
    UINT8 Reason;
    UINT8 Reserved[2];
} HBM_HOST_STOP_REQUEST;
```

Element	Description
Command	0x02
Reason	This element describes the reason the host is requesting the HECI interface be stopped. The enumeration of possible Reason values is described in Table 7-3 Reserved Reserved for future use. Must be zero for this version of the HECI bus messaging protocol.

Table 7-3. Host Stop Reason Codes

Reason Value	Description
0x00	Driver stop request.
0x01	Device D1 entry request
0x02	Device D2 entry request
0x03	Device D3 entry request
0x04	System S1 entry request
0x05	System S2 entry request
0x06	System S3 entry request
0x07	System S4 entry request
0x08	System S5 entry request
0x09	System restart request

7.9 Host Stop Response

This message must be the last message sent by the ME firmware HECI driver over the HECI interfaces to the host HECI driver. The ME firmware sends this message in response to the Host Stop Request message after all ME firmware clients have confirmed they are ready to stop. This message must be send within TBD seconds of the Host Stop Request message.

Response Message Required: None

Prototype

```
typedef struct _HBM_HOST_STOP_RESPONSE
{
    HBM_COMMAND Command;
    UINT8 Reserved[3];
} HBM_HOST_STOP_RESPONSE;
```

Element	Description
Command	0x82
Reserved	Reserved for future use. Must be zero for this version of the HECI bus messaging protocol.

7.10 ME Stop Request

This message is sent by the ME as a request for the host system to stop using the HECI interface. For example, this message may be sent as a result of a firmware update process that is sent remotely OOB from the IT console.

Response Message Required: Host Stop Request

Prototype

```
typedef struct _HBM_ME_STOP_REQUEST
{
HBM_COMMAND Command;
UINT8 Reason;
UINT8 Reserved[2];
} HBM_ME_STOP_REQUEST;
```

Element	Description
Command	0x03
Reason	Reason This element describes the reason the ME firmware is requesting the HECI interface be stopped. The enumeration of possible Reason values is described in Table 7-4.
Reserved	Reserved for future use. Must be zero for this version of the HECI bus messaging protocol.

Table 7-4. ME Stop Reason Codes

Reason Value	Description
0x00	Firmware Update.

7.11 Host Enumeration Request

This message is sent by the host HECI driver to enumerate the clients present in the ME. This message requests the list of valid addresses that are assigned to clients in

the ME. The assumption is that all ME firmware clients have been registered with the ME HECI driver prior to the ME HECI driver returning the Host Enumeration Response message.

Response Message Required: Host Enumeration Request

Prototype

```
typedef struct _HBM_HOST_ENUMERATION_REQUEST
{
    HBM_COMMAND Command;
    UINT8 Reserved[3];
} HBM_HOST_ENUMERATION_REQUEST;
```

Element	Description
Command	0x04
Reserved	Reserved for future use. Must be zero for this version of the HECI bus messaging protocol.

7.12 Host Enumeration Response

This message is sent by the ME HECI driver in response to the Host Enumeration Message to enumerate the clients present in the ME. This message returns the list of valid addresses that are assigned to clients in the ME. The assumption is that all ME firmware clients have been registered with the ME HECI driver prior to the ME HECI driver returning the Host Enumeration Response message.

Response Message Required: None

Prototype

```
typedef struct _HBM_HOST_ENUMERATION_RESPONSE
{
    HBM_COMMAND Command;
    UINT8 Reserved[3];
    UINT8 ValidAddresses[32];
} HBM_HOST_ENUMERATION_RESPONSE;
```

Element	Description
Command	0x84
Reserved	Reserved for future use. Must be zero for this version of the HECI bus messaging protocol.
ValidAddresses	This is an array of 256 bits. Each bit position indicates whether the corresponding address has been assigned to a ME firmware client or not. A value of 1 indicates the corresponding address has been assigned to a ME firmware client, while a value of 0 indicates it is not assigned. For example if bit 3 of the ValidAddresses bit array is set 1, then address 3 has been assigned to a ME firmware client. Address 0 is used by the HECI bus message protocol and therefore this address is reserved and bit 0 of this array will always be 0.

Note: ME firmware should not respond to the Host Enumeration Request message until the firmware is initialized and all ME clients are registered.

7.13 Host Client Properties Request

This message is sent by the host HECI driver to obtain a client's properties in the ME firmware.

Response Message Required: Host Client Properties Response

Prototype

```
typedef struct _HBM_HOST_CLIENT_PROPERTIES_REQUEST
{
    HBM_COMMAND Command;
    UINT8 Address;
    UINT8 Reserved[2];
} HBM_HOST_ CLIENT_PROPERTIES _REQUEST;
```

Element	Description
Command	0x05
Address	The address of the client for which the properties are being requested.
Reserved	Reserved for future use. Must be zero for this version of the HECI busmessaging protocol

7.14 Host Client Properties Response

This message is sent by the ME HECI driver in response to the Host Client Properties Request message to enumerate the clients present in the ME. This message returns the properties of the client requested in the Host Client Properties Request message. The assumption is that all ME firmware clients have been registered with the ME HECI driver prior to the ME HECI driver returning the Host Enumeration Response message.

Response Message Required: None

Prototype

```
typedef struct _HECI_CLIENT_PROPERTIES
{
    GUID ProtocolName;
    UINT8 ProtocolVersion;
    UINT8 MaxNumberOfConnections;
    UINT8 FixedAddress;
    UINT8 SingleReceiveBuffer;
    UINT32 MaxMessageLength;
} HECI_ CLIENT_PROPERTIES;

typedef struct _HBM_HOST_CLIENT_PROPERTIES_RESPONSE
{
    HBM_COMMAND Command;
    UINT8 Address;
    UINT8 Status;
    UINT8 Reserved[1];
    HECI_CLIENT_PROPERTIES ClientProperties;
} HBM_HOST_ CLIENT_PROPERTIES _RESPONSE;
```

Element	Description
Command	0x85
Address	The address of the client for which the properties are being requested.
Status	The result of the Host Client Properties Request message. 0x00 Success. 0x01 No ME FW Client is registered for Address.
Reserved	Reserved for future use. Must be zero for this version of the HECI bus messaging protocol.
ProtocolName	This GUID defines the name of client message protocol used by this client. See associated client messaging specification for a definition of GUID and client message formats.
ProtocolVersion	The version of the client protocol definition identified by ProtocolName. This field will be incremented whenever a change is made to the client's protocol that is backwards compatible with previous versions' definition. If a change is required to the client's protocol that is not backwards compatible, a new protocol must be defined with a new value for the ProtocolName field.
MaximumNumberOfConnections	This is the maximum number of connections that a client allows. If set to 0, the ME client is "connectionless", meaning that no connection is required to send and receive messages to the ME client and the HostAddress field is reserved and must be 0.
FixedAddress	If this client has a fixed address, the fixed address will be specified here. If the client is a dynamic address client, this field will be 0. Valid values for this field are 0 to 0x1F. See section **7.23** for more information on client addressing.
SingleReceiveBuffer	For connection based clients, set to 0x01 if this ME firmware client supports only a single receive buffer that is shared across multiple client connections. In this mode When Flow Control messages are sent from the ME to the host for this type of ME FW client, the HostAddress field of the Flow Control message must be set to 0x0. The host HECI driver should use this as a signal to send the next message from any of the host clients that are waiting to send a message to that ME FW client. For fixed address clients are connectionless and therefore will support only a single receive buffer.
MaxMessageLength	This is the largest message size that is supported by the client as a transmitter or recipient of HECI messages. The corresponding transmitting client should never attempt to transmit a message larger than MaxMessageLength; doing so may result in overflowing the receiving client's receive message buffer. MaxMessageLength is in units of bytes whose maximum value is 0xFFFFFFFF (4GB-1).

Note: If the address requested in Host Client Properties Request message was not valid (i.e. there was no firmware client registered for that address, then all of the client properties information (ProtocolName, ProtocolVersion, etc.) is indeterminate.

Note: ME firmware should not respond to the Host Client Properties Request message until the firmware is initialized and all ME clients are registered. Table 7-5 TBD below lists valid combinations of the MaximumNumberOfConnectios, FixedAddress, and SingleReceiveBuffer HECI_CLIENT_PROPERTIES fields.

Table 7-5 below lists valid combinations of the MaximumNumberOfConnections, FixedAddress, and SingleReceiveBuffer HECI_CLIENT_PROPERTIES fields.

Table 7-5. HECI_CLIENT_PROPERTIES Field Truth Table

FixedAddress	MaximumNumber OfConnections	SingleReceiveBuffer	Comments
0x0	0x0	Any value	INVALID. All dynamic address clients must support connections.
0x0	!= 0x0	0x0	Dynamic address ME client that supports a receive buffer per connection with each host client. Flow control is performed in messages sent in the connection.
0x0	!= 0x0	0x1	Dynamic address ME client that supports a single receive buffer that is used to receive messages from all host clients. Flow control is performed in messages sent in the connection.
!= 0x0	0x0	0x0	INVALID. All fixed address ME clients are connectionless and therefore must support only a single receive buffer.
!= 0x0	0x0	0x1	Fixed address ME client. Client connection, disconnection and flow control is not used to this type of ME client.
!= 0x0	!= 0x0	Any value	INVALID. All fixed address ME clients must be connectionless.

7.15 Client Connect Request

This message is sent to indicate that a client has registered with the host HECI driver and wishes to begin communication with the associated ME firmware client. This message provides a signal for the corresponding HECI driver to start flow control messages for the client. **Note:** The host driver can only send one Client Connect Request message per MEAddress at a time. The host driver must wait for a Client Connect Response message before putting another Client Connect Request message to the same MEAddress into the Host to ME HECI Buffer. Failure to do so may result in the ME not recognizing the second Client Connect Request.

Response Message Required: Client Connect Response

Prototype

```
typedef struct _HBM_CLIENT_CONNECT _REQUEST
{
    HBM_COMMAND Command;
    UINT8 MEAddress;
    UINT8 HostAddress;
    UINT8 Reserved;
} HBM_CLIENT_CONNECT_REQUEST;
```

Element	Description
Command	0x06
MEAddress	The address of the ME client the host is requesting to connect to. HostAddress The address of the Host client connecting to the ME client.
Reserved	Reserved for future use. Must be zero for this version of the HECI bus messaging protocol.

7.16 Client Connect Response

This message is sent in response to a Client Connect Request.

Response Message Required: None

Prototype

```
typedef struct _HBM_CLIENT_CONNECT_RESPONSE
{
    HBM_COMMAND Command;
    UINT8 MEAddress;
    UINT8 HostAddress;
    UINT8 Status;
} HBM_CLIENT_CONNECT_RESPONSE;
```

Element	Description
Command	0x86
MEAddress	The address of the ME client of the originating Client Connect Request message.
HostAddress	The address of the host client that originated the Client Connect Request message.
Status	Result of the Client Connect Request: 0x00 Success 0x01 Not Found. The client is no longer registered. 0x02 Already Connected. The host / ME client pair is already connected. 0x03 Out of resources, no additional connections allowed. 0x04 Invalid parameter. The specified MeAddress belongs to a FixedAddress connectionless client or is 0x0.
Reserved	Reserved for future use. Must be zero for this version of the HECI bus messaging protocol.

7.17 Client Disconnect Request

This message is sent to indicate that a host or ME client wishes to end communication with the associated host / ME firmware client. This message provides a signal for the corresponding HECI driver to stop flow control messages for the client. After sending the Client Disconnect Request message, the sender should process incoming messages as usual until Disconnect Response is received.

Response Message Required: Client Disconnect Response

Prototype

```
typedef struct _HBM_CLIENT_DISCONNECT_REQUEST
{
    HBM_COMMAND Command;
    UINT8 MEAddress;
    UINT8 HostAddress;
    UINT8 Reserved[1];
} HBM_CLIENT_DISCONNECT_REQUEST;
```

Element	Description
Command	0x07
MEAddress	The address of the ME client the host is requesting to disconnect from.
HostAddress	The address of the Host client disconnecting from the ME client.
Reserved	Reserved for future use. Must be zero for this version of the HECI bus messaging protocol.

Note: Both ME and Host could send a Client Disconnect Request message simultaneously. This should be fine; both sides should process the message and return the Client Disconnect Response message.

7.18 Client Disconnect Response

This message is sent in response to a Client Disconnect Request.

Response Message Required: None

Prototype

```
typedef struct _HBM_CLIENT_DISCONNECT_RESPONSE
{
    HBM_COMMAND Command;
    UINT8 MEAddress;
    UINT8 HostAddress;
    UINT8 Status;
} HBM_CLIENT_DISCONNECT_RESPONSE;
```

Element	Description
Command	0x87
MEAddress	The address of the ME client in the originating Client Disconnect Request message.
HostAddress	The address of the Host client in the originating Client Disconnect Request message.

Element	Description
Status	Result of the Client Disconnect Request: 0x0 Success

7.19 Flow Control

This message is sent by either the Host or ME whenever a client is ready to receive a message. See section 7.27 for a description of the flow control mechanism.

Response Message Required: None

Prototype

```
typedef struct _HBM_FLOW_CONTROL
{
    HBM_COMMAND Command;
    UINT8 MEAddress;
    UINT8 HostAddress;
    UINT8 Reserved[5];
} HBM_FLOW_CONTROL;
```

Element	Description
Command	0x08
MEAddress	The address of the ME client that is ready to receive a message.
HostAddress	The address of the Host client that is ready to receive a message.
Reserved	Reserved for future use. Must be zero for this version of the HECI bus messaging protocol.

Note: ME Clients may support only a single receive buffer, as indicated by the SingleReceiveBuffer field of HECI_CLIENT_PROPERTIES. When Flow Control messages are sent from the ME to the host for this type of client, the HostAddress field of the Flow Control message will be set to 0x0. The host HECI driver should use this as a signal to send the next message from any of the host clients that are waiting to send a message to that ME client.

7.20 Client Connection Reset Request

This message is sent to indicate that a host client wishes to reset communication with the associated host / ME firmware client. This message is only originated by the Host and is not originated by the ME. This message provides a signal for the corresponding HECI driver to reset flow control credits and re-issue flow control messages for the clients. This message is typically sent due to a host client canceling an I/O request that is in process. It causes the ME firmware HECI driver to do the following:

1. Stop and terminate any in progress large message receive operation. The next message between these two clients will be the start of a new message.

2. Signal the ME client that the connection is being reset.

3. Wait for the ME client to clean up and complete any outstanding transactions the ME client is processing for the Host Client. After the ME client has completed this, it will acknowledge the connection reset event to the ME HECI driver.

4. The ME firmware HECI driver will now reset its flow control credit count for this connection.

5. The ME firmware HECI driver then sends the Client Connection Reset Response message to the host HECI driver.

6. The host HECI driver now resets its flow control credit count for this connection and then sends any flow control messages for any receive buffers it still has outstanding for this client connection.

Response Message Required: Client Connection Reset Response

Prototype

```
typedef struct _HBM_CLIENT_CONNECTION_RESET_REQUEST
{
    HBM_COMMAND Command;
    UINT8 MEAddress;
    UINT8 HostAddress;
    UINT8 Reserved[1];
} HBM_CLIENT_CONNECTION_RESET_REQUEST;
```

Element	Description
Command	0x09
MEAddress	The address of the ME client for the connection.
HostAddress	The address of the Host client for the connection.
Reserved	Reserved for future use. Must be zero for this version of the HECI bus messaging protocol.

7.21 Client Connection Reset Response

This message is sent in response to a Client Connection Reset Request message

Response Message Required: None

Prototype

```
typedef struct _HBM_CLIENT_CONNECTION_RESET_RESPONSE
{
    HBM_COMMAND Command;
    UINT8 MEAddress;
    UINT8 HostAddress;
    UINT8 Status;
} HBM_CLIENT_CONNECTION_RESET_RESPONSE;
```

Element	Description
Command	0x89
MEAddress	The address of the ME client in the originating Client Connection Reset Request message.
HostAddress	The address of the Host client in the originating Client Connection Reset Request message.
Status	Result of the Client Connection Reset Request: 0x0 Success

7.22 Client Addressing

This section describes how addresses numbers are assigned to ME Client (MEAddress) and to Host Clients (HostAddress). These addresses are used in the MEAddress and HostAddress fields of the HECI message header. The address is used to link a host software client to a ME firmware client for the purposes of communication through the HECI interface.

7.22.1 ME Client Addressing

ME Clients can be of two address types: Fixed Allocation and Dynamic Allocation

7.22.1.1 Fixed Allocation Address

On a platform family basis, ME firmware clients integrated into the firmware that ships with the platform may have a fixed address. This means that for that platform family, the client will always have the same MEAddress value. The fixed address is useful for very early host firmware/BIOS code for which it is impractical to perform client address enumeration. See section 9 for a list of platform specific client address assignments.

Fixed address clients do not utilize the connection model. These clients are connectionless. These clients also do not utilize the flow control messages and

therefore may require timeouts and retries if the message does not successfully reach the ME firmware client.

The address range of 0x1 to 0x1F is reserved for fixed address clients. It is recommended that the operating system HECI driver not assume a fixed client address mapping; it should always perform enumeration of ME clients in the ME firmware. This makes the operating system HECI driver more portable from platform family to platform family.

7.22.1.2 Dynamic Allocation Address

These ME firmware clients are dynamically assigned addresses when they register with the firmware HECI driver during firmware initialization. To ensure that all ME firmware clients are registered with the firmware HECI driver, firmware should not respond to Host Enumeration Request or Host Client Properties Request messages until the firmware has completed its initialization process and all ME FW clients are registered with the ME HECI driver.

7.22.1.3 Host Client Addressing

Host software clients are assigned addresses when they register with the HECI driver in the host. The address is used in the HostAddress field of the HECI message header. The address is used to tell ME firmware which Host HECI client has sent the message. This allows several host clients to connect to a single ME FW client.

7.22.1.4 Addressing Fixed Addressed ME Clients

When addressing fixed address ME clients, the HostAddress field is reserved and must be set to 0 for this version of the HECI Bus message protocol.

7.22.1.5 Addressing Dynamic (Connection based) ME Clients

When addressing dynamic address ME clients, the HostAddress field must not be set to 0. The HostAddress field value of 0 is reserved for HECI bus messages.

7.23 Client Identification

The clients can be enumerated at any time through the Host Enumeration Request / Response and the Host Client Properties Request / Response messages.

7.24 Power Management

Prior to the host driver entering any low power state (either device low power D1-D3 state or system low power S1-S5 state), the host operating system software is expected to notify all host software clients (applications and/or drivers) that are using the HECI interface. This notification indicates that the HECI interface is about to be stopped. Then host operating system software will notify the HECI driver that all host software has stopped using the HECI interface, the host HECI driver will send the Host Stop Request message to the ME firmware. After the ME firmware sends the Host Stop

Response message, the HECI driver can put the host HECI PCI device into a lower power state (D3).

7.25 Resets

The HECI interface may experience both asynchronous and synchronous reset scenarios. If a ME reset occurs that causes the HECI interface to go into a reset state, host software should notify the Host Clients (applications) of the ME reset if possible.

7.26 Flow Control

The HECI messaging layer implements a simple flow control mechanism in the HECI bus message layer that assists the clients with flow control. This flow control communicates to the HECI driver that a client's receive buffer for a specific host / ME client address pair message is ready to receive a new message. This flow control communication is between the host and firmware HECI drivers. The HECI driver will not put (transmit) a client's message in the HECI circular buffer until the corresponding client is ready to receive a message. This flow control works in both directions: from host client to ME client and from ME client to host client.

There can only be one outstanding flow control message per host / ME Client connection pair at a time. For example, once a flow control message has been sent to signal that ME client X is ready to receive a message from host client Y, then another flow control message to signal that ME client X is ready to receive another message from host client Y will not be sent until host client Y sends message to ME client X.

Flow control between clients is initialized whenever a new connection is made by the host, and when a disconnect between two clients is made, any outstanding flow control credits are discarded. For connection based clients where the ME client has only a single receive buffer, the flow control is only reset when the HECI interface is reset, and not on connect or disconnect messages. Note that if the connection based client has a single receive buffer and the host starts sending a message to the ME and then disconnects the connection before the message is complete, the flow control credit was consumed by the host; the ME must free the current receive buffer and reissue the flow control credit.

7.27 Firmware Recovery Operation

When a FW image update fails (for example, due to a power failure during a FW update operation) the firmware will enter a recovery mode of operation. In the recovery mode of operation, the FW_STATUS register Current State field will be set to "Recovery". In this mode of operation, the HECI interface will operate with the following restrictions:

1. All HECI Bus Messages as specified in this specification are supported as defined.

2. A limited set of HECI messages are supported for BIOS. BIOS writers should consult the product specific BIOS specification for details on HECI messages not supported when the FW is in Recovery mode.

3. A single Recovery client is supported and enumerated by the operating system. This client is used to send the FW update image to the ME to install in the flash and complete the FW update process.

7.28 Bus Message Layer Errors

When in the Link Ready state, the following table lists errors that can occur at the bus message layer and their resulting actions.

Error	Action Taken
Length field in HECI_MESSAGE_HEADER defined bus messages does not equal size specified for the HBM_COMMAND.	Detecting agent (Host or ME) resets HECI interface.
Unknown HBM_COMMAND	Detecting agent (Host or ME) resets HECI interface.
Client message received for a connection that does not exist	Ignore and discard message.
Client message received for a connection when receive buffer is not ready.	Close the connection.
Client message received with the length exceeding MaxMessageLength in HECI_CLIENT_PROPERTIES.	Close the connection.
Connect Request for already existing connection	Send Host Client Properties Response message with Status = Already Connected.
Connect Request when the previous one to the same MeAddress is not responded yet	Ignore and discard the Client Connect Request message.
Disconnect Request/Response and FlowControl message to non-existing connection	Ignore and discard the message

8 Flows

8.1 Inter-Object Flows

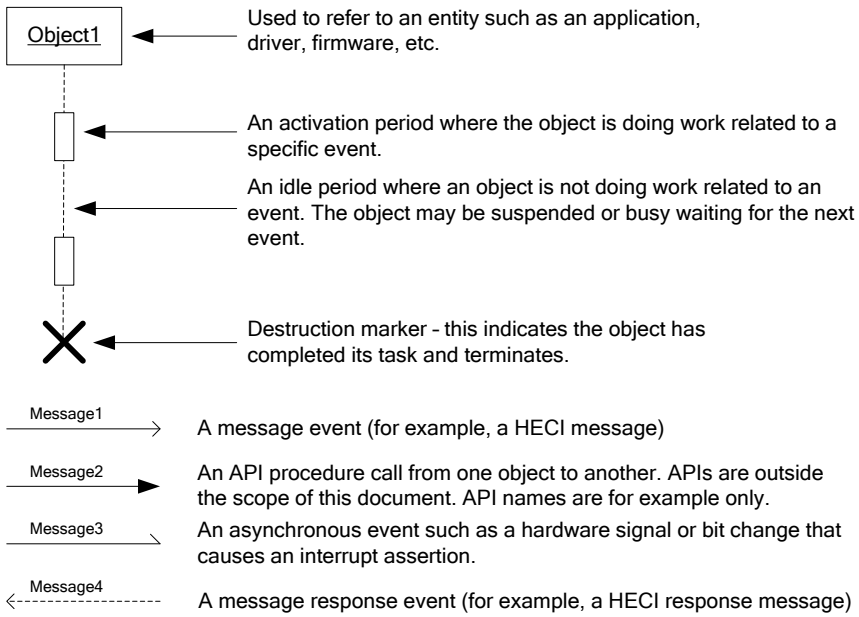
This section describes interactions between the various objects in the system that interact with the HECI subsystem. The term object is used as a general term to refer to any of the following:

Object Name	Description
Host HECI Client	Host software application, service or driver that interacts directly with the host HECI driver. This software sends and receives messages to an associated ME HECI Client via the host HECI driver.
Host HECI Driver	The driver layer software running on the host processor that is responsible for programming the host HECI hardware interface registers. This object name is also used synonymously with BIOS / firmware "drivers" for the HECI interface.
ME HECI Driver	The driver layer firmware running on the Intel® Management Engine (ME) that is responsible for programming the ME HECI hardware interface registers.
ME HECI Client	ME firmware that interacts directly with the ME HECI driver. This firmware sends and receives messages via to an associated Host HECI Client via the host HECI driver.

This section is intended to only describe the interactions between objects and actions taken based upon those interactions. This section does not describe the details of what each object does in great depth; that level of detail can be found in the associated flow sections for that object. The diagrams used in this section follow the UML sequence diagram format; Figure 8-1 provides a key of the formatting description of elements in a UML sequence diagram.

Note: These flows are intended to show the HECI message flows only and are not intended to document host driver or ME firmware driver APIs.

Figure 8-1. UML Sequence Diagram Key

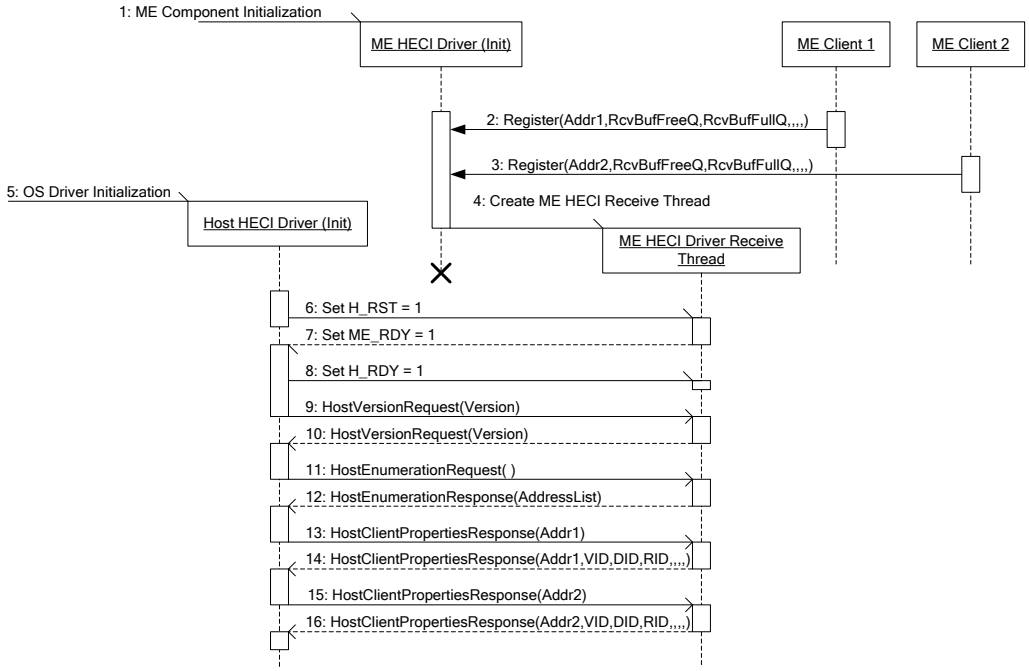


8.2 Driver Startup Flow

Figure 8-2 describes the startup flow from main ME firmware initialization to the point where the host driver / BIOS is ready to send and receive messages over the HECI interface. The diagram assumes a normal startup process, i.e. errors in the start up process are not covered here. Figure 8-2 shows a high level overview of the interactions of the host and ME HECI drivers during their associated initialization. The primary items accomplished in the flow are:

- The HECI hardware is initialized from a pre-boot state (ROM or early bring up firmware may have initialized the HECI interface already for the purposes of capturing BIOS progress codes).
- The ME firmware HECI clients register with the ME HECI driver.
- The host HECI driver and ME HECI driver pass startup messages between each other and an initial version checking mechanism is employed to ensure both agree on the HECI bus message interface version
- The host HECI driver enumerates all the ME firmware clients registered with the ME HECI driver.

Figure 8-2. High Level HECI Interface Startup Flow



Each communication event in Figure 8-2 is numbered. Table 8-1 provides a brief description of the event, the initiator of the event, the intended target of the event, and the subsequent action taken by the target.

Table 8-1. High Level HECI Interface Startup Flow

#	Initiator	Target	Dependencies	Action
1	Kernel	ME HECI Driver		The ME Kernel invokes the ME HECI driver's initialization function(s). The ME HECI driver initializes the HECI AUX registers and provides an interface for ME HECI firmware clients to register with the ME HECI driver.
2,3	ME HECI Client(s)	ME HECI Driver	1	The ME HECI clients invoke a function named "Register" of the ME HECI driver set up in step 1. The ME HECI clients are registering their client properties information with the ME HECI driver, as well as receiving a logical address that will be used to send messages to and from its corresponding host client.
4	ME HECI Driver	ME HECI Driver	2,3	The ME HECI driver will create and start a HECI message

#	Initiator	Target	Dependencies	Action
		Receive Thread		receive thread once it has been initialized with all clients property information. The ME Kernel's initialization model supports the ordering of these events.
5	OS PnP Manager	Host HECI Driver		The operating system plug and play manager starts the Host HECI driver initialization function(s). The host software, which may included one or more driver components including the host HECI driver, will initialize the PCI configuration space of the host HECI PCI device. The host HECI driver will then initialize the HECI PCI device's MMIO space, which will include setting the H_RST bit to 1.
6	Host HECI Driver	ME HECI Driver	4,5	Once the ME HECI driver has initialized its ME HECI driver receive thread, it is ready to begin communication with the host HECI driver. The ME HECI driver receive thread will service the H_RST assertion from the Host HECI driver. The ME HECI driver will reset the HECI circular buffer pointers and configure the HECI interface to prepare for the receipt of the first message from the host.
7	ME HECI Driver	Host HECI Driver	6	Once the ME HECI driver completes the ME HECI interface configuration and is ready to receive the first message from the host, the ME HECI driver sets the ME_RDY bit to 1. This activates the host HECI driver to continue the HECI PCI device's MMIO space initialization.
8	Host HECI Driver	ME HECI Driver	7	Once the Host is ready to send its first message to the ME, the host sets the H_RDY bit to 1. No action is needed from the ME HECI driver other than to log that the HECI interface is now in a ready state.
9	Host HECI Driver	ME HECI Driver	8	The host sends the Host Version Request bus message to the ME HECI driver. This message includes a version number. The ME HECI driver examines the version number to determine that it is a HECI bus message interface version that the firmware can support.

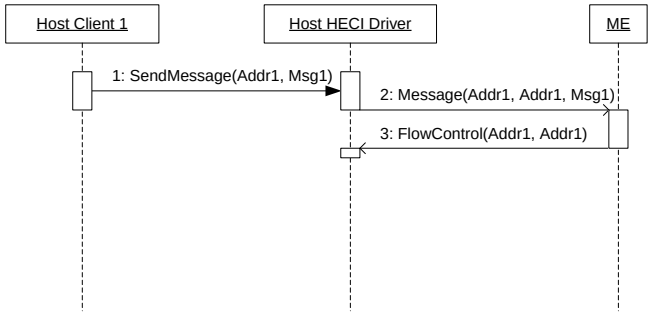
#	Initiator	Target	Dependencies	Action
10	ME HECI Driver	Host HECI Driver	9	The ME HECI driver sends the Host Version Response bus message to the Host HECI driver; the message includes an indication of whether the ME driver can support the host HECI driver's version number as well as the version number of the HECI bus message interface that the ME HECI driver supports. This results in the host HECI driver starting its HECI interface enumeration process.
11	Host HECI Driver	ME HECI Driver	10	The host HECI driver sends the HostEnumerationRequest bus message to the ME HECI driver to get the list of the addresses for registered clients. The ME HECI driver will take the list of addresses for ME firmware clients that registered in steps 2 and 3 and provide those to the Host HECI driver through the HostEnumerationResponse bus message.
12	ME HECI Driver	Host HECI Driver	11	Upon receipt of the HostEnumerationResponse bus message, the Host HECI driver can now begin querying for client properties information. Since the current HECI bus message interface does not support hot plugging of new firmware clients, it is expected that the host HECI driver only has to do the enumeration process once.
13, 15	Host HECI Driver	ME HECI Driver	12	The host driver now begins querying for client properties by sending HostClientPropertiesRequest bus messages to the ME HECI driver for each address found valid in the HostEnumerationResponse message. The ME HECI driver will validate the address in HostClientPropertiesRequest message is for a registered ME firmware client and if so then return the client's HECI_CLIENT_PROPERTIES information in a HostClientPropertiesResponse message. If the client is not present (address not valid), then the HsotClientPropertiesResponse is also sent, but all fields in the

#	Initiator	Target	Dependencies	Action
				HECI_CLIENT_PROPERTIES structure are filled with -1.
14, 16	ME HECI Driver	Host HECI Driver	13,15	For each HostClientPropertiesRequest received, the ME HECI driver returns the HostClientPropertiesResponse. The Host HECI driver then records each client's properties for future use in the Client start up flow.

8.3 Host Client Small Message Send Flow

Figure 8-3 describes a flow of a host client sending a small message using HECI. "Small" messages are those messages whose HECI message header and body size is less than the free HECI message buffer space and can be transferred as a single message through the HECI circular buffer interface.

Figure 8-3. Host Client Small Message Send Flow



Each communication event in Figure 8-3 is numbered. Table 8-3 provides a brief description of the event, the initiator of the event, the intended target of the event, and the subsequent action taken by the target.

Table 8-2. Host Client Small Message Send Flow

#	Initiator	Target	Dependencies	Action
1	Host HECI Client 1	Host HECI Driver	Client startup flow complete d	Client 1 sends a message targeted to ME Client 1.
2	Host HECI Driver	ME HECI Driver	1	The Host HECI driver accepts this message and when a flow control packet indicates there is a free buffer ready to receive the message on the

#	Initiator	Target	Dependencies	Action
				ME, it sends the message. The ME HECI driver extracts the message from the Host to ME circular buffer and writes it to client 1's awaiting empty buffer.
3	ME HECI Driver	Host HECI Driver	2	The ME HECI driver sends a FlowControl message to the Host HECI driver indicating that it is now ready to receive another message from the Host HECI client.

A similar flow is described for a ME client sending a message to the host, shown in Figure 8-4

Figure 8-4. ME Client Small Message Send Flow

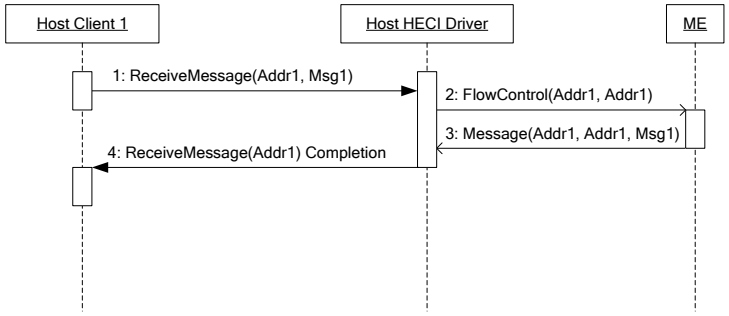


Table 8-3. ME Client Small Message Send Flow

#	Initiator	Target	Dependencies	Action
1	Host HECI Client 1	Host HECI Driver	Client startup flow completed	When the Host client is ready to receive a message from the ME it invokes a function such as "ReceiveMessage". The host client may be expecting a response from the FW to a message it is about to send to the ME firmware.
2	Host HECI Driver	ME HECI Driver	1	The host HECI driver now sends a FlowControl message to the ME HECI driver to inform it that the host client is ready to receive a message.

#	Initiator	Target	Dependencies	Action
3	ME HECI Driver	Host HECI Driver	2	The ME HECI client now decides to send a message to the host HECI client. Perhaps this message is a response to an earlier request message from the host. If the ME HECI driver has a flow control credit for the address requested, it puts the message in the ME to Host HECI buffer.
4	Host HECI Driver	Host HECI Client 1	3	The host HECI driver removes the message from the ME to Host HECI buffer and returns the message to Host HECI Client 1 by completing the originating "receiveMessage" function.

9 DCMI-HI

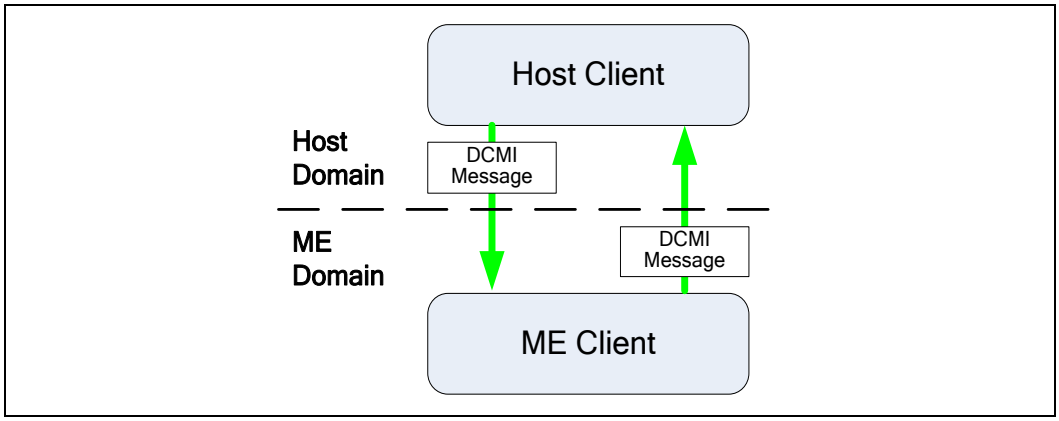
This section describes the specific usage of the HECI interface for transferring DCMI messages between host software and the ME.

The HECI (Host Embedded Controller Interface) provides a bi-directional fully asynchronous communications interface between host software and management controller or "management engine" (ME) firmware, as shown in the following figure.

This specification uses the term “Client” to refer to the logical software endpoints that send and receive messages via HECI. Clients can be either drivers, services, or applications depending upon the architecture of the software stack interfacing with HECI. The goal of the DCMI-HI messaging interface is to provide a transport between a host client and a ME client for the purpose of transferring DCMI messages.

Figure 9-1 depicts the simplification of DCMI-HI architecture.

Figure 9-1. DCMI-HI High Level Architecture



9.1 Comparision with KCS

In many DCMI solutions, the host SW communicates with the ME over a Keyboard Controller Style (KCS) interface, as defined by [IPMI]. In order to reduce the changes in host software, the DCMI-HI capabilities can be used in the same manner as KCS and to encapsulate the same messages as communicated over the KCS.

Table 9-1. DCMI-HI and KCS Comparison

Characteristic	KCS	DCMI-HI
Handshake	Per Byte	Per Message

Characteristic	KCS	DCMI-HI
Handshake interrupt support	Optional	Mandatory
Asynchronous Responses from ME to Host	Notify then transfer. KCS provides status/optional interrupt indicating that message data is available. Multiple KCS byte-level transactions must then be issued transfer the data.	Transfer then Notify. The DCMI-HI puts the response message in the HECI buffer whenever there is buffer space available and then notifies the host via status bit or interrupt.
Asynchronous Messages from ME to Host	Pull model using the IPMI *Get Message* command. KCS provides status/optional interrupt indicating that message data is available. Any message to the host that is not a direct response to a host-issued request must be delivered via the KCS Message Buffer and retrieved by a Get Message command.	Same as KCS. The DCMI-HI protocol can also support the asynchronous delivery of messages directly without going through the receive message queue, should that capability be required in the future.
Polled operation supported?	Yes	Yes
Abortable Transactions	Yes. A transaction can be stopped midway and a new transaction started.	Yes. The last value in a message transaction indicates whether the transfer should be committed or ignored.
Interruptable Transactions	Yes. The KCS definition includes intrinsic support that enables another party, such as an SMI Handler, to interrrupt and abort a transaction in progress, perform its own transactions, then return to the interrupt handler/driver. The interrupted party can detect that this occurred and restart the interrupted transaction.	No. This capability is not supported by DCMI-HI.

9.2 Request / Response Protocol

DCMI Messages utilize a request / response protocol where a request message (also called a command) is used to request an operation and a response message is returned. A request message contains fields that identify the type of request and zero or more parameters (request data) that are associated with the particular request.

Under normal operation, every request message will receive a corresponding response message.

The response message is used to convey several things, as defined by [DCMI]:

Normal completion - a value called the completion code is returned in all responses. If this code indicates "Command Completed Normally" it typically means the command was accepted and successfully acted on. In some cases, the response may be sent before the command has been fully executed. In this case, "Command Completed Normally" only means the command was accepted.

Error Status - The completion code may also return an error value if there were errors in the parameters of the request, or if the particular DCMI command is not supported by the implementation. The completion code is at the DCMI Messaging level. Errors at the transport (HECI) interface level are reported through separate HECI error indications.

Response data - in addition to the completion code, the response message may also carry zero or more bytes of response data that is specific to the request.

9.3 Request / Response Symmetry

The DCMI-HI protocol allows both requests and responses to be generated by the host or by the ME. Typically, the host software will generate requests and the ME will provide responses. The definition of the requests and responses messages that are supported across the interface is given by [DCMI].

9.4 Asynchronous Notification

The KCS interface definition did not require an asynchronous notification (interrupt) mechanism to notify the host that a message was available from the ME. This meant that some implementations only supported being polled by host software.

DCMI-HI requires that Asynchronous Notification via interrupt is available for all implementations. DCMI-HI also requires support that allows a driver or BIOS to poll the interface if desired.

9.5 HECI Level Error reporting

In some cases, the ME may not be able to return a response at the DCMI level. In those cases, the FW will either return an empty response buffer with a HECI success indication, or will return an empty response with HECI error indication.

The FW will return an empty buffer with an error indication in the following cases:

1. FW Host i/f is not ready to process command
2. Length error - input length is too short or the message is otherwise unable to be positively identified as a DCMI-HI request or response

The FW will return an empty buffer with success indication in the following cases:

1. Invalid DCMI-HI version
2. Response bit is on
3. DCMI-HI messaging has not been initialized with the software

9.6 Multiple Outstanding Requests

The FW implementation of DCMI-HI is allowed to support 'command queuing' where host software can issue more than one request without having to wait for a response.

Unlike the KCS interface, the DCMI-HI protocol includes IPMI sequence number (Seq field) support to allow the interface to support multiple outstanding requests and more than one logical software requester.

9.7 Message Integrity

DCMI-HI does not provide additional integrity checks on the message transfer, other than verifying the transaction length.

9.8 Request Retries

The DCMI-HI is considered fundamentally reliable. If a correctly formatted request does not receive a response, it is assumed to be for reasons other than data corruption in the transmission of the request or response across the interface. However, the DCMI-HI does include the use of the IPMI Seq field that can be used to differentiate a retried request from an original request. This provides a mechanism that can enable a non-idempotent command to be retried without re-executing it. Refer to the specification of the "BT" interface in [IPMI] for more information on the Seq field and its use.

9.9 Aborting Transactions

Host-to-ME abort by Host: The bytes for a host-to-ME message transfer are written into the HECI HOST Circular Buffer. The HECI interface does not provide host control of the write pointer, which means the host cannot set the pointer back to a starting point in order to restart loading a message.

There is a HOST RESET capability that resets the Host Circular Buffer, but this affects the whole interface, including the ME Circular Buffer (ME to Host). Unless a HOST RESET is issued, a Host-to-ME message transfer must always be completed.

The Host can direct the ME to ignore a message by using a field called the 'commit' field. The commit field is the last field in a message transaction. Based on the value of the field, the message will either be accepted for further processing or will be ignored. Once the commit field has been written, it cannot be recalled. The only options available at that point are to complete the transaction or perform a HOST RESET.

ME-to-Host abort by Host: Similarly, there is no provision for the host to abort an ME-to-host transfer once the bytes have begun to be written into the circular buffer. The host side must therefore perform the entire transaction, after which it can discard the message from ME if desired. The ME can also cause the host to ignore the message by placing the appropriate value in the commit field.

9.10 Dropped Messages

The only cause for dropped messages at the DCMI-HI transport level is if the commit field is set to 'Drop'. Otherwise, refer to [DCMI] and [IPMI] for causes of dropped messages at the DCMI Message level, and to [HECI] for dropped messages at the HECI transport level.

9.11 Out-of-order Responses and Asynchronous Messages

If a driver or host software issues a request before the response for a previous command has been returned it is possible that temporal ordering of responses will not be preserved. Additionally, the HECI interface is capable of supporting other messaging protocols besides DCMI-HI. Therefore, host software must check the ME Address information in the HECI Message Header to verify the that HECI message is a DCMI-HI message (see **Error! Reference source not found.**, **Error! Reference source not found.**), and then check the NetFn/LUN, Command, and Seq fields (9.23, DCMI-HI Message Formats) to verify that a response matches up with a given request. This is described further in [IPMI].

9.12 Circular Buffer Depths

The host and ME Circular Buffer depths shall be set to at least 16 uint32 (64 bytes) in the ME.

9.13 HECI Message Length and Byte Ordering

The Length field in the HECI Message Header provides the message length in bytes, excluding the HECI_MESSAGE_HEADER. A value of 0 indicates no additional bytes are part of the message. Note that HECI interface transfers in units of DWORDs. Within a DWORD, bytes are little endian ordered, i.e. byte 0 is bits 0 to 7, byte 1 is bits 8 to 15, byte 2 is bits 16 to 23 and byte 3 is bits 24 to 31.

9.14 HECI Bus Messages

Table 9-2 presents a summary of the HECI Bus Messages. This list is provided to illustrate the types of messages that are used for generic HECI interface initialization and setting up messaging connections between the host client and the ME client. Refer to [HECI] for more information.

Table 9-2 - HECI Bus Message Command Summary, HECI Version 0x0001

Code	Name	Initiator	Response Required
0x01	Host Version Request	Host	Yes
0x02	Host Stop Request	Host	Yes
0x03	ME Stop Request	ME	No
0x04	Host Enumeration Request	Host	Yes
0x05	Host Client Properties Request	Host	Yes
0x06	Client Connect Request	Host	Yes
0x07	Client Disconnect Request	Host or ME	Yes
0x08	Flow Control	Host or ME	No
0x09	Client Connection Reset Request	Host	Yes

9.15 DCMI-HI Discovery and Connection

DCMI-HI uses Dynamic Allocation Address as described in [HECI]. This means that the address (ME Address) that is associated with the ME Client for DCMI-HI is not fixed and must be discovered. The following steps provide a quick overview of the process a host uses to discover whether the ME contains a HECI Client that supports DCMI-HI and how it connects to the Client to get the ME Address for subsequent communication.

1. Host software issues a Host Enumeration Request. ME responds with a 256-bit array where each bit position identifies an UINT8 'address' value from 0 to 255. The address is a handle that is used for identifying an ME Client that services a given protocol. If the array has N bits that are set, it means that N different Client connections are supported.

2. Host software iterates through the supported addresses using the Host Client Properties Request to get information on which protocols are supported by each ME Client. A GUID is used as the primary identifier for a protocol. Each Host Client Properties Response returns the GUID for the protocol, plus additional information such as the supported version of the protcol, how many connections the client supports, and so on.

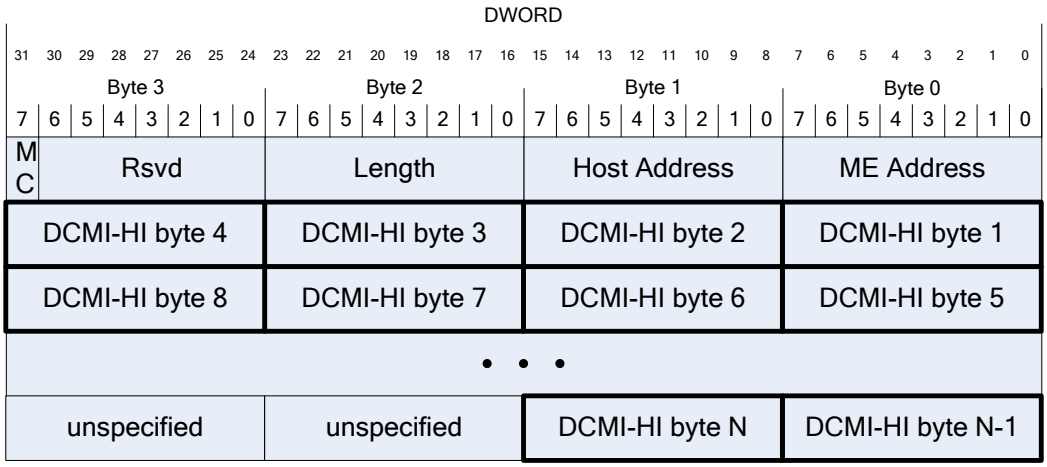
3. Host software initiates a connection using the Client Connect Request. The request includes a host address value is used to identify the Host Client. A connection binds a Host Client address and an ME Client address. A given connection is subsequently identified by the combination of the Host Client address and ME Client address.

9.16 DCMI-HI Message Encapsulation

DCMI Messages are carried in the Message Data portion of a HECI Message (see Figure 9-2. DCMI-HI Byte Offsets to HECI Message Data Mapping).

By convention, DCMI Interfaces are specified as a sequential byte stream with byte labelling starting from 1. The following figure shows the relationship of the DCMI byte offsets to the DWORDs used to transfer the Message Data across the HECI Interface.

Figure 9-2. DCMI-HI Byte Offsets to HECI Message Data Mapping



9.17 DCMI-HI GUID

The following GUID identifies the DCMI-HI Message Protocol. This GUID is used for discovery and connection establishment as described in the 9.15, DCMI-HI Discovery and Connection.

GUID: {7519B383-48FC-43e5-A5EB-5959CB581000}

9.18 Channel Number

Each interface to a BMC in [IPMI] has a channel number that is used when configuring the channel and for routing messages between channels. DCMI-HI uses the System Interface channel number (0x0F) as specified in [IPMI].

DCMI-HI Channel Number: System Interface, 0x0F

9.19 Channel Protocol Type

[IPMI] defines a Channel Protocol Type number that is used to identify the format of messages that are used on the channel. This number is returned in the response to the *Get Channel Info* command. The DCMI-HI message format matches the KCS message format. The Channel Protocol does not match up exactly with any of the pre-existing IPMI protocols, therefore the Channel Protocol Type shall be returned as "OEM Type 1" (0x1C).

DCMI-HI Channel Protocol Type: OEM 1 (0x1C)

9.20 Channel Medium Type

The *Get Channel Info* command also returns a value for the Channel Medium Type. The DCMI-HI shall return 0Ch [System Interface (KCS, SMIC, or BT)] as the Channel Medium Type. While [IPMI] presently only calls this out for KCS, SMIC, or BT system interfaces, it's interpreted that this value is useable a general identifier for any system interface.

DCMI-HI Channel Medium Type: 0Ch

9.21 Channel Vendor IANA

The *Get Channel Info* command also requires a Vendor "IANA" number to be returned. This number identifies the OEM or organization that has defined the protocol for the channel. Since Intel is the author of the DCMI-HI specification, the Vendor IANA is set to the IANA for Intel Corporation, 343 (decimal).

DCMI-HI Channel Vendor IANA: Intel (343)

9.22 SMS and Event Message Buffer Interrupt Type

The *Get Channel Info* command also requires an SMS Interrupt Type and Event Message Buffer Interrupt Type to be returned as follows:

For Channel = Fh (System Interface) :

byte 1: SMS Interrupt Type

00h-0Fh = IRQ 0 through 15, respectively

10h-13h = PCI A-D, respectively

14h = SMI

15h = SCI

20h-5Fh = system interrupt 0 through 63, respectively

60h = assigned by ACPI / Plug 'n Play BIOS

FFh = no interrupt / unspecified

all other = reserved

byte 2: Event Message Buffer Interrupt Type

see values for byte 1

Since HECI is implemented as a PCIe device, there is little value in obtaining the particular interrupt assignment, therefore it is acceptable to return unspecified (0xFF) for the SMS Interrupt Type. The Event Message Buffer is not a feature of the DCMI-HI.

DCMI-HI SMS Interrupt Type: unspecified (0xFF) or the PCI (0x10-0x13) or system interrupt (0x20-0x5F) being used.

DCMI-HI Event Message Buffer Interrupt Type: unspecified (0xFF)

9.23 DCMI-HI Message Formats

The following figures define the ordering of fields for DCMI-HI Request and Response messages.

Figure 9-3, DCMI-HI Request Message Format

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5:N-1	Byte N
RsSWID or RsSA	NetFn/LUN	Seq	Cmd	Data	Commit

Figure 9-4, DCMI-HI Response Message Format

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6:N-1	Byte N
RqSA or RqSWID	NetFn/LUN	Seq	Cmd	Completion Code	Data	Commit

Where:

Field	Defined by	*Description*
RsSA, RsSWID	**IPMI, DCMI-HI**	*Responder's Slave Address or Responder's Software ID (SWID). This is a provision to support the possible future extension of DCMI to deliver asynchrous messages directly to the Host software without going through the receive message queue.* *For this version of DCMI-HI, Host-to-ME requests can use any value for SWID.*

		ME-to-Host asynchronous requests are not supported. *BMC address 0x20 should be used as RsSA for messages targeted to ME. Responses return the same RqSA or RqSWID that was used to deliver the request.*
RqSA, RqSWID	**IPMI, DCMI-HI**	*Requester's Slave Address or Requester's Software ID (SWID). This is a provision to support the possible future extension of DCMI to deliver asynchrous messages directly to the Host software without going through the receive message queue.* *For this version of DCMI-HI, ME to Host responses use the SWID value from the original request.* *Host-to-ME responses are not supported.*
LUN	**IPMI**	*Logical Unit Number. This is a sub-address that allows messages to be routed to different 'logical units' that reside behind the same physical interface. The LUN field occupies the least significant two bits of the first message byte.*
NetFn	**IPMI**	*Network Function code. This provides the first level of functional routing for messages received by the BMC via the KCS Interface. The NetFn field occupies the most significant six bits of the first message byte. Even NetFn values are used for requests to the BMC, and odd NetFn values are returned in responses from the BMC.*
Seq	**IPMI**	*Used for matching responses up with requests. The DCMI-HI interface can support interleaved 'multi-threaded' communications if the underlying DCMI implementation elects to support it. There can be multiple simultaneous outstanding requests from the Host with responses returned asynchronously (and in any order).* *The Requester sets the value for this field. The Responder returns the value in the corresponding response. The Seq field must be used in combination with the NetFn and Command fields to uniquely identify a request or response. I.e. the same Seq value is allowed to be used in multiple outstanding requests, as long as the combinations of Seq value, NetFn, and Command are unique.*
Cmd	**IPMI, DCMI**	*Command code. This message byte specifies the operation that is to be executed under the*

		specified Network Function.
Data	**IPMI, DCMI**	*Zero or more bytes of data, as required by the given command. The general convention is to pass data LS-byte first, but check the individual command specifications to be sure.*
Commit	***DCMI-HI***	*Indicates whether the message should be accepted for further processing or should be ignored.* *0x01 = Commit.* *A value of 0x01 indicates that the message should be accepted for further parsing and processing.* *0x00 = Drop.* *A value of 0x00 indicates that the message must be ignored (dropped).* *All other values are reserved.*

9.24 DCMI-HI -Specific Messages

There are no messages that are specific to the configuration or operation of the DCMI-HI transport. All messages are either DCMI or IPMI messages as specified by [DCMI] or [IPMI], respectively, or are HECI Bus messages as defined by [HECI].

10 Timing Specifications

This section presents the timing specifications for DCMI Messaging using the DCMI-HI.

Figure 10-1, DCMI-HI Timing Specifications

Symbol	Min	Typ.	Max	*Description*
T1	**-**	30 ms	2 s	*Host Request to ME Response timeout. Measured from the time that the Host Request transaction transfer is initiated (I.e. the time when the Host launches the transfer after it has finished loading the request message data into the circular buffer) to the time that the Response transaction become available.* *Unless otherwise specified by [DCMI], if this timeout occurs on a legitimate request message (i.e. a request message that is correctly formatted) it is an error condition. If the interface continues to timeout on successive requests, a HECI HOST RESET may be attempted.*
T2	**-**	30 ms	2 s	*ME Request to Host Response timeout. Measured from the time that the ME Request transaction transfer becomes available to the Host to the time when the Host has initiated the Response transaction (I.e. the time when the Host launches the transfer after it has finished loading the response message data into the circular buffer).* *Unless otherwise specified by [DMCI] the ME may consider it an error condition if the Host does not respond within this interval. The ME may elect to retry the request.*
T3	**-**	-	*5 s*	*Seq field expiration. The time that the Seq field value remains valid for a given message. See [IPMI] for more information on Seq field expiration.*

§

§