Smart Battery System Specifications

System Management Bus BIOS Interface Specification

Revision 1.0
February 15, 1995

Copyright© 1996, Benchmarq Microelectronics Inc., Duracell Inc., Energizer Power Systems, Intel Corporation, Linear Technology Corporation, Maxim Integrated Products, Mitsubishi Electric Corporation, National Semiconductor Corporation, Toshiba Battery Co., Varta Batterie AG, All rights reserved.

Questions and comments regarding this specification may be forwarded to:

Intel Corporation
Intel Architecture Labs Technical Support
PHONE: (8am-5pm PST) (800) 628-8686
FAX: (916) 356-6100
INTERNATIONAL (916) 356-3551
Email: IAL_Support@ccm.hf.intel.com

THIS SPECIFICATION IS PROVIDED "AS IS", WITH NO WARRANTIES WHATSOEVER, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE.

IN NO EVENT WILL ANY SPECIFICATION CO-OWNER BE LIABLE TO ANY OTHER PARTY FOR ANY LOSS OF PROFITS, LOSS OF USE, INCIDENTAL, CONSEQUENTIAL, INDIRECT OR SPECIAL DAMAGES ARISING OUT OF THIS AGREEMENT, WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES. FURTHER, NO WARRANTY OR REPRESENTATION IS MADE OR IMPLIED RELATIVE TO FREEDOM FROM INFRINGEMENT OF ANY THIRD PARTY PATENTS WHEN PRACTICING THE SPECIFICATION.

Table of Contents

REVISION HISTORY

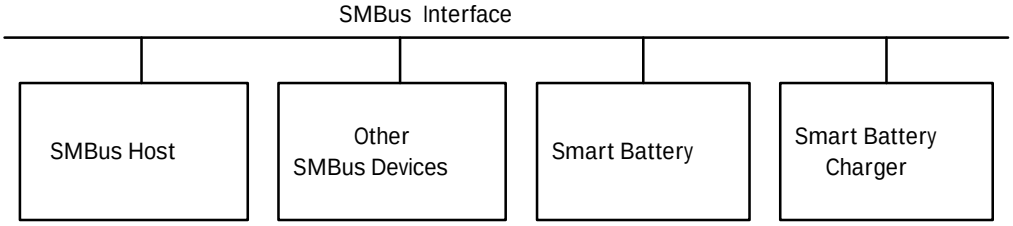
Revision Number	Date	Notes
1.0	2/15/95	General Release

DO WE NEED THIS BLANK PAGE ?

1. Introduction

The System Management Bus BIOS Interface Specification presents a standardized BIOS interface to devices on the SMBus. It exposes the SMBus to device drivers and application programs. In some instances, detailed information can be retrieved directly from a device like the Smart Battery, but in others, such as a Power Plane Controller (a device that selectively turns on and off power to devices on the system board), the BIOS may deny direct access.

The drawing below illustrates a typical SMBus system featuring an SMBus Host, Smart Battery, Smart Battery Charger and other SMBus devices such as an LCD Backlight Controller, Power Plane Controller etc.



1.1 Scope

This BIOS specification is designed to abstract the hardware implementation of the SMBus and expose the SMBus in a standardized way to higher layers of software. Software will not need to directly manipulate bits and bytes on the SMBus and is effectively isolated from needing to know about that process. System implementation of the SMBus and how the BIOS communicates with that implementation are NOT described by this specification. BIOS interaction with devices on the bus are also outside the scope of this document.

1.2 Audience

The audience for this document includes:

- Implementors of SMBus BIOS interface
- Designers of device drivers for SMBus devices such as Smart Batteries, Smart Battery Chargers, Smart Battery Selectors and other SMBus devices
- Designers of power management systems for portable electronic equipment powered by Smart Batteries and other SMBus Devices
- Application Programmers

2. References

- Advanced Power Management BIOS Interface Specification v1.1, Intel Corporation/Microsoft Corporation, September 1993
- Smart Battery Data Specification Revision v1.0, Duracell Inc./Intel Corporation, February, 1995
- System Management Bus Specification Revision v1.0, Intel Corporation, February, 1995

3. Definitions

- **APM:** Advanced Power Management. A BIOS interface defined to coordinate system-wide power management control via software.
- **Smart Battery:** A battery equipped with specialized hardware that can provide present state, calculated, and predicted information about the battery to an SMBus Host under software control. The access methodology to this data is described by this specification. The data content and protocol are defined in the Smart Battery Data Specification and the System Management Bus Specification.
- **Smart Battery Charger:** A battery charger that is designed to periodically communicate with and charge a Smart Battery. It is capable of dynamically adjusting its charging characteristics in response to the information provided by the Smart Battery.
- **Smart Battery Selector:** A SMBus device that selects which battery is connected to the charger, which battery is powering the system, and which battery is communicating with the SMBus host.
- **SMBus Device:** An electronic device or module that communicates via the SMBus with an SMBus Host and/or other SMBus Devices. For example, an LCD Backlight Controller in a notebook computer can be implemented as a SMBus Device.
- **SMBus**: The System Management Bus is a specific implementation of an I²C bus. The SMBus specification (see references) describes the data protocols, device addresses, and electrical requirements that are superimposed on the I²C bus specification. The SMBus is used to physically transport commands and information between the Smart Battery, SMBus Host, Smart Battery Charger, and other SMBus Devices.
- **SMBus Host:** A system that communicates with SMBus devices. It usually will embody some or all of a system's power management control. To accomplish this, it communicates with the Smart Battery and uses that information in executing its power management policy.

4. SMBus BIOS Interface

The SMBus BIOS interface provides a mechanism for device drivers, the OS, and application software to access some or all of the SMBus devices and in particular, the Smart Battery System. Exactly which devices are exposed and the extent of that exposure falls in the domain of the system BIOS designer and is beyond the scope of this specification.

4.1 SMBus Driver Connection and Calling Interface

The SMBus BIOS Interface provides both real and protected mode calling interfaces.

A real mode (Int 15H) SMBus BIOS Interface is required for all implementations. This default or not-connected interface uses the SMBus Access command code (53B0H) and the existing 15H BIOS interface. The SMBus BIOS Int 15H interface must operate in either real mode or virtual-86 mode on 80386 and later processors. Note: this document will refer to the real mode Int 15H interface as simply Int 15H. The Int 15H interface will be disabled except as noted whenever any connection exists.

It is required that support for 16-bit protect mode and 32-bit protect mode be provided; support for the real mode connection is optional. The SMBus BIOS interface, when connected, is accessed by calling through the real mode or protected mode entry points returned by the SMBus Real Mode Connect, 16-Bit Connect, and SMBus 32-Bit Connect calls. Note: this document will refer to the connected modes collectively as the Connected Mode.

Note: The contents of the resisters used by BOTH the successful and unsuccessful return may be altered. The contents of the registers used to call a function may also be altered. The contents of all other registers will remain unaltered.

4.1.1 SMBus Installation Check (01H)

This call allows the SMBus caller to determine if a system's BIOS supports the SMBus BIOS Interface and if so, which version of the specification it supports. The values passed in BL and CX are required to uniquely identify a legitimate caller to the SMBus BIOS Interface and, if not present, will result in an SMBus invalid signature error.

The version number returned by this call is the highest level of SMBus BIOS Interface specification supported by the SMBus BIOS.

The vendor-specified hardware code may be optionally used to identify the SMBus host hardware. If this feature is not used, it must return zero. This return code may be used by operating systems that do not want to use the BIOS services, but rather want to identify and communicate directly with the hardware.

Call With

AX	=	53B0H	SMBus Access
BH	=	01H	SMBus Installation Check
BL	=	72H	
CH	=	61H	
CL	=	64H	

Returns

If function successful:

Carry	=	0	SMBus is supported by BIOS
AH	=	01H	SMBus BIOS Interface Specification major version number (in BCD format)
AL	=	00H	SMBus BIOS Interface Specification minor version number (in BCD format)
BL	=	Number of SMBus Devices Present	
CH	=	ASCII "i" character (69H)	
CL	=	ASCII "A" character (41H)	
DX	=	Vendor Specified SMBus Hardware Code 0000H indicates undefined hardware	

If function unsuccessful:

Carry	=	1
AH	=	Error code 0AH SMBus invalid signature 86H SMBus not supported

Supported modes

Int 15H only (this function is always available)

4.1.2 SMBus Real Mode Connect (02H)

This call allows an SMBus caller to connect to the SMBus BIOS Interface in real mode. This function returns the appropriate entry points into the SMBus BIOS.

The SMBus BIOS rejects an interface connect request if a connection already exists. When the SMBus BIOS interface is connected in any mode, the default Int 15H interface will be disabled.

Call With

AX = 53B0H SMBus Access
BH = 02H SMBus Real Mode Connect
CH = ASCII "i" character (69H)
CL = ASCII "A" character (41H)

Returns

If function successful:

Carry = 0
AX = SMBus 16-bit code segment (real mode segment base address)
BX = Offset of entry point into the SMBus BIOS Interface
CX = SMBus 16-bit data segment (real mode segment base address)

If function unsuccessful:

Carry = 1
AH = Error code
01H SMBus connect failed
02H SMBus already connected
0AH SMBus invalid signature
86H SMBus not supported
AL = If Error Code = 02H
01H Real mode connect already established
02H 16-bit connect already established
03H 32-bit connect already established

Supported modes

Int 15H only (this function is always available)

4.1.3 SMBus 16-Bit Connect (03H)

This call allows an SMBus caller to connect to the SMBus BIOS Interface in 16-bit protect mode. This function returns the appropriate entry points into the SMBus BIOS.

The SMBus BIOS rejects an interface connect request if a connection already exists. The default SMBus BIOS Interface Int 15H real mode interface will be disabled when a protected mode connection is established.

Call With

AX	=	53B0H SMBus Access
BH	=	03H SMBus 16-bit Connect
CH	=	ASCII "i" character (69H)
CL	=	ASCII "A" character (41H)

Returns

If function successful:

Carry	=	0
AX	=	SMBus 16-bit code segment (real mode segment base address)
BX	=	Offset of entry point into the SMBus BIOS Interface
CX	=	SMBus 16-bit data segment (real mode segment base address)
SI	=	CSeg length in bytes
DI	=	DSeg length in bytes

If function unsuccessful:

Carry	=	1
AH	=	Error code 01H SMBus connect failed 02H SMBus already connected 0AH SMBus invalid signature 86H SMBus not supported
AL	=	If Error Code = 02H 01H Real mode connect already established 02H 16-bit connect already established 03H 32-bit connect already established

Supported modes

Int 15H only (this function is always available)

Comments:

To use the SMBus BIOS 16-bit protected mode interface after the connection is in effect requires the caller to set up two selector descriptor entries in either the GDT or LDT. These selectors must be in consecutive order and must come in the sequence: 16-bit code segment, then data segment. The calling code must build the descriptors using the corresponding segment base and segment length information returned from this call. At the time the SMBus BIOS is called, these descriptors must be valid and CPL=0. The code descriptors must also specify a ring-0 privilege level.

The caller builds a far pointer to the SMBus BIOS 16-bit entry point using the 16-bit code selector and the offset returned in BX from this call. Subsequent calls to the SMBus BIOS are performed by loading the appropriate registers and doing a far call to this constructed 16-bit entry point. The calling software must provide a 16-bit stack that is large enough to handle any use by the BIOS plus handle any possible interrupts that occur.

4.1.4 SMBus 32-Bit Connect (04H)

This call allows an SMBus caller to connect to the SMBus BIOS Interface in 32-bit protected mode. This function returns the appropriate entry points into the SMBus BIOS.

The SMBus BIOS rejects an interface connect request if any real or protected mode connection already exists. The default SMBus BIOS Interface Int 15H real mode interface will be disabled when a protected mode connection is established.

Call With

AX	=	53B0H SMBus Access
BH	=	04H SMBus 32-bit Connect
CH	=	ASCII "i" character (69H)
CL	=	ASCII "A" character (41H)

Returns

If function successful:

Carry	=	0
AX	=	SMBus 32-bit code segment (real mode segment base address)
EBX	=	Offset of the 32-bit code entry point into the SMBus BIOS
CX	=	SMBus 16-bit code segment (real mode segment base address)
DX	=	SMBus data segment (real mode segment base address)
SI	=	32-bit CSeg length in bytes
DI	=	DSeg length in bytes

If function unsuccessful:

Carry	=	1
AH	=	Error code 01H SMBus connect failed 02H SMBus already connected 0AH SMBus invalid signature 86H SMBus not supported
AL	=	If Error Code = 02H 01H Real mode connect already established 02H 16-bit connect already established 03H 32-bit connect already established

Supported modes

Int 15H only (this function is always available)

Comments:

To use the SMBus BIOS 32-bit protected mode interface after the connection is in effect requires the caller to set up three selector descriptor entries in either the GDT or LDT. These selectors must be in consecutive order and must come in the sequence: 32-bit code, 16-bit code, then data segment. The calling code must build the descriptors using the corresponding segment base and segment length information returned from this call. At the time the SMBus BIOS is called, these descriptors must be valid and CPL=0. The code descriptors must also specify ring-0 privilege level. A 16-bit code segment is also given, so that the SMBus BIOS may call 16-bit code from the 32-bit interface if required.

The caller builds a far pointer to the SMBus BIOS 32-bit entry point using the 32-bit code selector and the offset returned in EBX from this call. Subsequent calls to the SMBus BIOS are performed by loading the appropriate registers and doing a far call to this constructed 32-bit entry point. The calling software must provide a 32-bit stack that is large enough to handle any use by the BIOS plus handle any possible interrupts that occur.

4.1.5 SMBus Disconnect (05H)

This call allows the connected SMBus caller to disconnect from the SMBus BIOS Interface. The connection may be disconnected only by the connected driver calling with these defined values into the SMBus BIOS interface. The default SMBus BIOS Int 15H real mode interface will be re-enabled when the connection is terminated.

Call With

AX = 53B0H SMBus Access
BH = 05H SMBus Disconnect
CH = ASCII "i" character (69H)
CL = ASCII "A" character (41H)

Returns

If function successful:

Carry = 0
AH = 00H SMBus OK

If function unsuccessful:

Carry = 1
AH = Error code
04H SMBus not connected
05H SMBus Int 15H Disabled
0AH SMBus invalid signature
86H SMBus not supported

Supported modes

Connected Mode only

4.1.6 SMBus Device Address (06H)

This call allows the SMBus caller to create a complete list of the SMBus device addresses present in the system. This call does NOT return the reserved SMBus device addresses. It will return the addresses of all SMBus devices, including those that the BIOS restricts access to. This call is used repetitively until the caller gets the addresses of all the SMBus devices that are present in the system. Note: This call will only return the one address per device. The least significant bit of the address is a read/write bit; so for example, if 0x16 is returned, then the device may respond at both addresses 0x16 AND 0x17.

This function may be used by systems that share a common I^2C controller for both the SMBus and ACCESS.bus. Since ACCESS.bus dynamically assigns addresses, it needs to know which addresses are already in use. This function provides the ACCESS.bus host a list of addresses to pre-load its "address-in-use" table.

Call With

AX	=	53B0H	SMBus Access
BH	=	06H	SMBus Device List
BL	=	xxH	Address at position (0...n-1)
CH	=	ASCII “i” character (69H)	
CL	=	ASCII “A” character (41H)	

Returns

If function successful:

Carry	=	0
AH	=	00H SMBus OK
BH	=	Number of SMBus Devices
BL	=	SMBus Device Address at list position xxH

If function unsuccessful:

Carry	=	1
AH	=	Error code 06H SMBus device address request out of range 0AH SMBus invalid signature 86H SMBus not supported

Supported modes

Int 15H (this function is always available) and Connected Mode

4.1.7 SMBus Critical Messages (07H)

This call allows the SMBus caller to retrieve SMBus device-to-SMBus host messages. These messages are stored in a queue that is at least five (three-byte) messages long. Each successful invocation of this call returns and removes the oldest remaining message from the message queue.

If the message queue overflows, the oldest message will be lost and the call will return a Message List Overflow error. The caller still needs to retrieve the remaining messages.

Call With

AX = 53B0H SMBus Access
BH = 07H SMBus Critical Messages
CH = ASCII “i” character (69H)
CL = ASCII “A” character (41H)

Returns

If function successful:

Carry = 0
AH = 00H SMBus OK
AL = SMBus Device Address
BX = SMBus Device Message

If function unsuccessful:

Carry = 1
AH = Error code
05H SMBus Int 15H Disabled
07H SMBus unknown failure
08H SMBus message list empty
09H SMBus message list overflow
0AH SMBus invalid signature
86H SMBus not supported

Supported modes

Int 15H and Connected Mode

4.2 SMBus Device Command Protocols

The following are commands that are used to read data from or write data to SMBus devices. The SMBus Host acts as a master and the target SMBus Device as a slave.
Note: In the following descriptions, xxH means a value is required when calling the SMBus BIOS and nnH means the BIOS returns a value.

The SMBus BIOS access is a two-phase process. Separating the request from the response allows drivers access to the relatively slow SMBus devices without seriously impacting system performance. For example, a Smart Battery may take over 30 ms to respond to a command. Waiting for a BIOS return during this time might seriously impact system performance.

4.2.1 SMBus Request (10H)

This command requests access to a device on the SMBus. This command, in conjunction with the SMBus Request Continuation command, is used to request access to the SMBus for all SMBus protocols. The SMBus Command Complete call is (repeatedly) used to determine if and when the SMBus command has been completed and to gather the results of the pending SMBus transaction. Note: for the SMBus protocol BlockWrite, DH (MSB) will contain the block length and DL (LSB) will contain the first byte in the block. Refer to tables 4.2.5 and 4.2.6.

Call With

AX	=	53B0H	SMBus Access
BH	=	10H	SMBus Request Command
BL	=	xxH	SMBus Protocol
CH	=	xxH	SMBus Device Address
CL	=	xxH	SMBus Device Command
DH	=	xxH	MSB data
DL	=	xxH	LSB data

Returns

If function successful:

Carry = 0

AH = 00H, 80H SMBus OK - the high bit set indicates a previously unreported SMI has taken place.

If function unsuccessful:

Carry = 1

AH = Error code
- 05H SMBus Int 15H Disabled
- 10H SMBus Device Address Not Acknowledged
- 11H SMBus Device Error Detected
- 12H SMBus Device Command Access Denied
- 13H SMBus Unknown Error
- 14H SMBus Transaction Pending
- 17H SMBus Device Access Denied
- 19H SMBus Protocol not Supported
- 1AH SMBus Busy
- 86H SMBus not supported

Supported modes

Int 15H and Connected Mode

4.2.2 SMBus Request Continuation (11H)

This command is used to continue requesting access to the SMBus for the SMBus write block protocol. Refer to tables 4.2.5 and 4.2.7 for specific values. See the examples in Appendix C for more details.

Call With

AX	=	53B0H	SMBus Access
BH	=	11H	SMBus Request Continuation Command
BL	=	xxH	SMBus Protocol
CH	=	xxH	SMBus Device Address
CL	=	xxH	number of valid bytes in DX (1 or 2)
DH	=	xxH	MSB data (CL = 1 or 2)
DL	=	xxH	LSB data (CL = 2)

Returns

If function successful:

Carry = 0

AH = 00H SMBus OK

CL = 00H SMBus Hardware not ready for more data
01H SMBus Hardware ready for 2 more data bytes

If function unsuccessful:

Carry = 1

AH = Error code
05H SMBus Int 15H Disabled
11H SMBus Device Error Detected
13H SMBus Unknown Error
15H SMBus No Transaction Pending
16H SMBus Request does not Match Pending Transaction
18H SMBus Timeout
1BH SMBus SMI detected
86H SMBus not supported

Supported modes

Int 15H and Connected Mode

4.2.3 SMBus Request Abort (12H)

This command stops the current SMBus request. This command is used normally to terminate a pending request after an SMBus SMI Detected error (1BH) is noted. However, it may be used to terminate any pending request. Refer to tables 4.2.5 and 4.2.8 for specific values. See the examples in Appendix C for more details.

Call With

AX	=	53B0H	SMBus Access
BH	=	12H	SMBus Request Abort Command
BL	=	xxH	SMBus Protocol
CH	=	xxH	SMBus Device Address
CL	=	xxH	SMBus Device Command

Returns

If function successful:

Carry = 0

AH = 00H SMBus OK

If function unsuccessful:

Carry = 1

AH = Error code
- 05H SMBus Int 15H Disabled
- 13H SMBus Unknown Error
- 15H SMBus No Transaction Pending
- 16H SMBus Request does not Match Pending Transaction
- 86H SMBus not supported

Supported modes

Int 15H and Connected Mode

4.2.4 SMBus Request Data and Status (13H)

The SMBus Request Data and Status call is used to determine if and when a SMBus transaction has been completed. When this call is made, the BIOS will go out to the hardware and if any data is available, gather it one or two bytes at a time from the SMBus interface device before returning. This command will be repeated until all the data available is retrieved or an error detected. Note: for all SMBus protocols except BlockRead, the data is atomic and must be returned in one call. (For example, a ReadWord command must return both bytes in the same call.) For the BlockRead protocol, the first call to this service will return the block length in DH and the first byte of the block in DL, if CL=2. For subsequent calls the next most significant data byte will be returned in DH and least significant data byte in DL (if CL = 2). Refer to tables 4.2.5 and 4.2.9 for specific values. See the examples in Appendix C for more details.

Call With

AX	=	53B0H	SMBus Access
BH	=	13H	SMBus Request Complete Command
BL	=	xxH	SMBus Protocol
CH	=	xxH	SMBus Device Address
CL	=	xxH	SMBus Device Command

Returns

If function successful:

Carry	=	0	
AH	=	00H	SMBus OK
CH	=	nnH	00H No data pending, transaction complete
			01H No data pending, transaction continues
			02H Data pending
CL	=	nnH	Number of valid bytes in DX (0 ... 2)
DH	=	nnH	MSB data
DL	=	nnH	LSB data

If function unsuccessful:

Carry	=	1
AH	=	Error code 05H SMBus Int 15H Disabled 10H SMBus Device Address Not Acknowledged 11H SMBus Device Error Detected 13H SMBus Unknown Error 14H SMBus Transaction Pending 15H SMBus No Transaction Pending 16H SMBus Request does not Match Pending Transaction 18H SMBus Timeout 1BH SMBus SMI detected 86H SMBus Not Supported

Supported modes

Int 15H and Connected Mode

4.2.5 SMBus Protocol Codes

The following table lists the valid SMBus protocol codes:

SMBus Protocol	Code
Quick Command	00H
Send Byte	01H
Receive Byte	02H
Write Byte	03H
Read Byte	04H
Write Word	05H
Read Word	06H
Block Write	07H
Block Read	08H
Process Call	09H
reserved	0AH ...FFH

4.2.6 SMBus Request - Protocol Register Requirements

The following table lists the registers used by an SMBus Request call for each SMBus protocol:

SMBus Protocol	AX	BH	BL	CH	CL	DH	DL
Quick Command	53B0H	10H	00H	Addr	n/a	n/a	n/a
Send Byte	53B0H	10H	01H	Addr	n/a	n/a	data
Receive Byte	53B0H	10H	02H	Addr	n/a	n/a	n/a
Write Byte	53B0H	10H	03H	Addr	Cmd	n/a	data
Read Byte	53B0H	10H	04H	Addr	Cmd	n/a	n/a
Write Word	53B0H	10H	05H	Addr	Cmd	MSB	LSB
Read Word	53B0H	10H	06H	Addr	Cmd	n/a	n/a
Block Write	53B0H	10H	07H	Addr	Cmd	len	db1
Block Read	53B0H	10H	08H	Addr	Cmd	n/a	n/a
Process Call	53B0H	10H	09H	Addr	Cmd	MSB	LSB

4.2.7 SMBus Request Continuation - Protocol Register Requirements

The following table lists the registers used by an SMBus Request Continuation for each SMBus protocol call:

SMBus Protocol	AX	BH	BL	CH	CL	DH	DL
Quick Command	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Send Byte	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Receive Byte	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Write Byte	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Read Byte	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Write Word	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Read Word	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Block Write	53B0H	11H	07H	Addr	1 or 2	db n	db n+1
Block Read	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Process Call	n/a	n/a	n/a	n/a	n/a	n/a	n/a

4.2.8 SMBus Request Abort - Protocol Register Requirements

The following table lists the registers used by an SMBus Abort call for each SMBus protocol:

SMBus Protocol	AX	BH	BL	CH	CL
Quick Command	53B0H	12H	00H	Addr	n/a
Send Byte	53B0H	12H	01H	Addr	n/a
Receive Byte	53B0H	12H	02H	Addr	n/a
Write Byte	53B0H	12H	03H	Addr	Cmd
Read Byte	53B0H	12H	04H	Addr	Cmd
Write Word	53B0H	12H	05H	Addr	Cmd
Read Word	53B0H	12H	06H	Addr	Cmd
Block Write	53B0H	12H	07H	Addr	Cmd
Block Read	53B0H	12H	08H	Addr	Cmd
Process Call	53B0H	12H	09H	Addr	Cmd

4.2.9 SMBus Request Data and Status - Protocol Register Requirements

The following table lists the registers used by an SMBus Data_Status call for each SMBus protocol:

SMBus Protocol	AX	BH	BL	CH	CL
Quick Command	53B0H	13H	00H	Addr	n/a
Send Byte	53B0H	13H	01H	Addr	n/a
Receive Byte	53B0H	13H	02H	Addr	n/a
Write Byte	53B0H	13H	03H	Addr	Cmd
Read Byte	53B0H	13H	04H	Addr	Cmd
Write Word	53B0H	13H	05H	Addr	Cmd
Read Word	53B0H	13H	06H	Addr	Cmd
Block Write	53B0H	13H	07H	Addr	Cmd
Block Read	53B0H	13H	08H	Addr	Cmd
Process Call	53B0H	13H	09H	Addr	Cmd

4.3 SMBus Device-to-SMBus Host

In the case where an SMBus Device wants to communicate with the SMBus Host, the SMBus Device temporarily becomes a master and treats the SMBus Host as a slave. These messages are *always* sent using the SMBus Write Word protocol, where the command code is the sending SMBus Device's slave address followed by a word of data that constitutes the message. The SMBus Host is expected to queue up five of these messages. They are accessed by an entity that periodically polls for their presence. (refer to the SMBus Device List function for details about how to access these messages)

4.4 SMBus Error Detection and Signaling

The SMBus uses a simple system for signaling errors. This system is designed to minimize the amount of traffic on the SMBus while allowing either the SMBus Host or any SMBus Device to signal an error condition has been detected.

4.4.1 SMBus Device Address Not Acknowledged Error

When the target SMBus Device fails to acknowledge its SMBus slave address, the SMBus controller is obliged to generate a stop condition on the SMBus. The SMB BIOS then signals the caller by returning 10H in AH and setting the carry flag.

4.4.2 SMBus Device Signals An Error

When the target SMBus Device detects an error condition, it signals the SMBus controller by failing to acknowledge any data byte. The SMBus controller then generates a stop condition on the SMBus and the SMB BIOS signals the caller by setting the carry flag and returning 11H in AH. Note: Because SMBus errors are device-specific, there is no standard method to return the cause of the error. For example, the Smart Battery has a function that returns the reason for the error but the Smart Charger does not have an error reporting mechanism.

4.4.3 SMBus Controller Detected Error

When the SMBus Controller detects an error, it generates a stop condition on the SMBus and terminates the data transfer. The SMB BIOS then signals the error condition by setting the carry flag and returning the appropriate error code in AH.

4.4.4 SMB-BIOS Detected Error

When the SMB BIOS detects an error, if necessary, it causes the SMBus controller to generate a stop condition on the SMBus and terminates the data transfer. The SMB BIOS then signals the error condition by setting the carry flag and returning the appropriate error code in AH.

Appendix A - SMB Extension Function Summary

This table summarizes the SMBus driver support functions.

SMBus Function	Int 15	Real Mode	16 Bit	32 Bit
reserved (00H)	n/a	n/a	n/a	n/a
SMBus Installation Check (01H)	Yes	No	No	No
SMBus Real Mode Connect (02H)	Yes	No	No	No
SMBus 16-bit Connect (03H)	Yes	No	No	No
SMBus 32-bit Connect (04H)	Yes	No	No	No
SMBus Disconnect (05H)	No	Yes	Yes	Yes
SMBus Device Address (06H)	Yes	Yes	Yes	Yes
SMBus Critical Message List (07H)	Yes	Yes	Yes	Yes
reserved (08H .. 0FH)	n/a	n/a	n/a	n/a

This table summarizes the SMBus data protocols.

SMBus Function	Int 15	Real Mode	16 Bit	32 Bit
SMBus Request (10H)	Yes	Yes	Yes	Yes
SMBus Request Continuation (11H)	Yes	Yes	Yes	Yes
SMBus Request Abort (12H)	Yes	Yes	Yes	Yes
SMBus Request Data and Status (13H)	Yes	Yes	Yes	Yes
reserved (14H .. FFH)	n/a	n/a	n/a	n/a

Appendix B - Firmware Error Codes

SMB Error Descriptions

(00H or 80H) SMBus OK

This error code, in conjunction with the carry flag clear, is returned to indicate that the call has been successfully completed. The most significant bit will be set whenever an SMI that has accessed the SMBus has occured and an SMBus BIOS transaction is not in progress. This bit will be automatically cleared by an SMBus Request, SMBus Request Continuation, SMBus Request Abort or SMBus Request Data and Status command. This bit allows device drivers to detect if a sequence of commands that must completed without intervening commands was interrupted by an SMI. For example, the Smart Battery's AtRateTimeRemaining command is dependent upon the AtRate value. If this value was altered by a command issued in SMM code, the return value could be incorrect.

(01H) Real Mode Connection Already Established

This error code, in conjunction with the carry flag set, is returned to indicate that the connect call failed because a real mode connection already exists.

(02H) 16-bit Protected Mode Connection Already Established

This error code, in conjunction with the carry flag set, is returned to indicate that the connect call failed because a 16-bit protected mode connection already exists.

(03H) 32-bit Protected Mode Connection Already Established

This error code, in conjunction with the carry flag set, is returned to indicate that the connect call failed because a 32-bit protected mode connection already exists.

(04H) SMBus Not Connected

This error code, in conjunction with the carry flag set, is returned to indicate that the disconnect call failed because no connection exists.

(05H) SMBus Int 15H Disabled

This error code, in conjunction with the carry flag set, is returned to indicate that the a caller tried to access the function via an Int 15H call when the Int 15H interface was disabled.

(06H) SMBus Address Request Out Of Range

This error code, in conjunction with the carry flag set, is returned to indicate that the device address call failed because the requested device was out of range. The requested device must be between 0 and one less than the number of SMBus devices returned by the installation check call or the device address call.

(07H) SMBus Unknown Failure

This error code, in conjunction with the carry flag set, is returned to indicate that the device address call failed because of an unknown SMBus error.

(08H) SMBus Message List Empty

This error code, in conjunction with the carry flag set, is returned to indicate that the device address call failed because the message queue was empty. No action required.

(09H) SMBus Message List Overflow

This error code, in conjunction with the carry flag set, is returned to indicate that the device address call failed because the message queue overflowed. This bit will be set when the BIOS detects that the message list queue has overflowed and will be reset when the error is returned.

(0AH) SMBus Invalid Signature

This error code, in conjunction with the carry flag set, is returned to indicate that the value passed in CX was not "iA" (6941H). The call will fail unless a valid signature is is present in CX when the call is made.

(10H) SMBus Device Address Not Acknowledged

This error code, in conjunction with the carry flag set, is returned to indicate that the SMBus call failed because the slave device address was not acknowledged.

(11H) SMBus Device Error Detected

This error code, in conjunction with the carry flag set, is returned to indicate that the SMBus call failed because the slave device signaled an error condition. The caller can read the device's error register if one is available.

(12H) SMBus Device Command Access Denied

This error code, in conjunction with the carry flag set, is returned to indicate that the SMBus call failed because the SMBus host will not allow the specific command for the device being addressed. For example, the SMBus host may not allow a caller to adjust the Smart Battery's low capacity alarm value.

(13H) SMBus Unknown Error

This error code, in conjunction with the carry flag set, is returned to indicate that the SMBus call failed because the SMBus host encountered an unknown error.

(14H) SMBus Transaction Pending

This error code, in conjunction with the carry flag set, is returned to indicate that the SMBus call failed because the SMBus BIOS already has a pending transaction.

(15H) SMBus No Transaction Pending

This error code, in conjunction with the carry flag set, is returned to indicate that the SMBus call failed because the SMBus BIOS was expecting a pending transaction.

(16H) SMBus Request does not Match Pending Transaction

This error code, in conjunction with the carry flag set, is returned to indicate that the SMBus call failed because the calling information does not match that for the pending transaction.

(17H) SMBus Device Access Denied

This error code, in conjunction with the carry flag set, is returned to indicate that the SMBus call failed because the SMBus host will not allow access to the device addressed. For example, the SMBus host may not allow a caller to communicate with an SMBus device that controls the system's power planes.

(18H) SMBus Timeout

This error code, in conjunction with the carry flag set, is returned to indicate that the SMBus call failed because the SMBus host detected an SMBus Timeout The timeout value is expected to be in the range of 25 - 30 ms (> THog in the SMBus Specification), but its actual value will be system-specific.

(19H) SMBus Host Unsupported Protocol

This error code, in conjunction with the carry flag set, is returned to indicate that the SMBus call failed because the SMBus host does not support the requested protocol. The SMBus BIOS is not required to support all protocols, only those required by the devices present in the system.

(1AH) SMBus Busy

This error code, in conjunction with the carry flag set, is returned to indicate that the SMBus call failed because the SMBus host reports that the SMBus is presently busy with some other transaction. For example, the Smart Battery may be sending charging information to the Smart Charger.

(1BH) SMBus SMI Detected

This error code, in conjunction with the carry flag set, is returned to indicate that the pending SMBus transaction was interrupted by an SMI. The caller should issue and SMBus Request Abort, then repeat the SMBus transaction or sequence of transactions as appropriate.

(86H) SMBus BIOS Interface Not Supported

This error code, in conjunction with the carry flag set, is returned to indicate that the SMBus call failed because the host does support the SMBus BIOS interface.

SMB Error Code Groupings

SMBus Errors	00H-09H, 10H-1BH, 80H
SMBus BIOS Interface Not Present	86H
All other values reserved	

SMB Specific Errors

Error Number	SMBus Extension Function Error Messages	Applicable SMBus Calls
00H, 80H	SMBus OK	SMBus Disconnect (05H) SMBus Device Address (06H) SMBus Critical Message List (07H) SMBus Request (10H) SMBus Request Continuation (11H) SMBus Request Abort (12H) SMBus Request Data and Status (13H)
01H	Real mode connection already established	SMBus Real Mode Connect (02H) SMBus 16-bit Connect (03H) SMBus 32-bit Connect (04H)
02H	16-bit connection already established	SMBus Real Mode Connect (02H) SMBus 16-bit Connect (03H) SMBus 32-bit Connect (04H)
03H	32-bit connection already established	SMBus Real Mode Connect (02H) SMBus 16-bit Connect (03H) SMBus 32-bit Connect (04H)
04H	SMBus not connected	SMBus Disconnect (05H)
05H	SMBus Int 15H Disabled	SMBus Disconnect (05H) SMBus Critical Message List (07H) SMBus Request (10H) SMBus Request Continuation (11H) SMBus Request Abort (12H) SMBus Request Data and Status (13H)
06H	SMBus device address request out of range	SMBus Device Address (06H)
07H	SMBus unknown failure	SMBus Critical Message List (07H)
08H	SMBus message list empty	SMBus Critical Message List (07H)
09H	SMBus message list overflow	SMBus Critical Message List (07H)
0AH	SMBus invalid signature	SMBus Real Mode Connect (02H) SMBus 16-bit Connect (03H) SMBus 32-bit Connect (04H) SMBus Disconnect (05H) SMBus Device Address (06H) SMBus Critical Message List (07H)

SMB Specific Errors (cont)

Error Number	SMBus Extension Function Error Messages	Applicable SMBus Calls
10H	SMBus Device Address Not Acknowledged	SMBus Request (10H) SMBus Request Data and Status (13H)
11H	SMBus Device Error Detected	SMBus Request (10H) SMBus Request Continuation (11H) SMBus Request Data and Status (13H)
12H	SMBus Device Command Access Denied	SMBus Request (10H)
13H	SMBus Unknown Error	SMBus Request (10H) SMBus Request Continuation (11H) SMBus Request Abort (12H) SMBus Request Data and Status (13H)
14H	SMBus Transaction Pending	SMBus Request (10H) SMBus Request Data and Status (13H)
15H	SMBus No Transaction Pending	SMBus Request Continuation (11H) SMBus Request Abort (12H) SMBus Request Data and Status (13H)
16H	SMBus Request does not Match Pending Transaction	SMBus Request Continuation (11H) SMBus Request Abort (12H) SMBus Request Data and Status (13H)
17H	SMBus Device Access Denied	SMBus Request (10H)
18H	SMBus Timeout	SMBus Request Continuation (11H) SMBus Request Data and Status (13H)
19H	SMBus host unsupported protocol	SMBus Request (10H)
1AH	SMBus Busy	SMBus Request (10H)
1BH	SMBus SMI Detected	SMBus Request Continuation (11H) SMBus Request Data and Status (13H)
86H	SMBus BIOS interface not supported	SMBus Installation Check (01H) SMBus Real Mode Connect (02H) SMBus 16-bit Connect (03H) SMBus 32-bit Connect (04H) SMBus Disconnect (05H) SMBus Device Address (06H) SMBus Critical Message List (07H) SMBus Request (10H) SMBus Request Continuation (11H) SMBus Request Abort (12H) SMBus Request Data and Status (13H)

Appendix C - Example Code Fragments

This section contains example code fragments intended to illustrate the use of selected SMBus BIOS Interface functions. In typical use, an SMBus driver will:

1. Test for the presence of an SMBus BIOS Interface
2. If one is present, will attempt to connect to the BIOS interface
3. Poll for events
4. Provide user services as required
5. Disconnect from the BIOS interface

SMBus Installation Check

```
        mov     ax,     53b0h   ; SMBus access code
        mov     bh,     01h     ; SMBus installation check command
        mov     bl,     72h     ; an identifier to make sure we don't run into an
                                ; unexpected int 15h  function
        mov     ch,     61h
        mov     cl,     64h
        int     15h
        jc      command_failed          ; carry flag is set indicating an error
                                        ; check al for error code

        cmp     ch,     'i'
        jnc     install_chk_fail        ; Bad return value - command failed
        cmp     cl,     'A'
        jnc     install_chk_fail        ; Bad return value - command failed
        ; installation check succeeded - proceed
```

Real Mode Connect

```
smb_off         dw      0
smb_seg         dw      0

        mov     ax,     53b0h   ; SMBus access code
        mov     bh,     02h     ; real mode connect command
        mov     ch,     'i'
        mov     cl,     'A'
        int     15h
        jc      command_failed ; carry flag is set indicating an error
                               ; check al for error code

        mov     smb_off, bx     ; we now have a pointer to the BIOS entry point
        mov     smb_seg, ax
        ; real mode connect succeeded - proceed
```

SMBus 16-Bit Connect

```
SMB16Entry     dd      0

SMB16CodeSel   dw      0
SMB16DataSel   dw      0

SMB16CodeSeg   dw      0
SMB16SegOff    dw      0
SMB16DataSeg   dw      0
SMB16DataSeg   dw      0
SMB16DataLen   dw      0

               ; perform SMB 16-bit protected mode connect
               mov     ax,53B0H
               mov     bh,04H
               mov     ch,'i'
               mov     cl,'A'
               int     15h
               jc      connect_failed

               ; save off SMB Info
               mov     [SMB16CodeSeg],ax
               mov     [SMB16SegOff],bx
               mov     [SMB16DataSeg],cx
               mov     [SMB16CodeLen],si
               mov     [SMB16DataLen],di

               ; allocate two new consecutive LDT descriptors; one for data, one for code
               ; order MUST be code first, then data
               .....
               .....
               mov     [SMB16CodeSel],new code selector
               mov     [SMB16DataSel],new data selector

               ; initialize 16-bit protected mode code descriptor
               ;    selector start at [SMB16CodeSeg]:0
               ;    segment length of [SMB16CodeLen]
               .....
               .....

               ; initialize 16-bit protected mode data descriptor
               ;    selector start at [SMB16DataSeg]:0
               ;    segment length of [SMB16DataLen]
               .....
               .....

               ; build 16-bit entry point into SMB BIOS
               mov     ax,[SMB16codeSel]
               mov     [SMB16entry+2],ax
               mov     ax,[SMB16SegOff]]
               mov     [SMB16entry],ax
```

SMBus 32-Bit Connect

```
SMB32Entry     dd      0

SMB32CodeSel   dw      0
SMB32DataSel   dw      0

SMB32CodeSeg   dw      0
SMB32CodeLen   dw      0
SMB16CodeSeg   dw      0
SMB32Off       dd      0
SMB32DataSeg   dw      0
SMB32DataLen   dw      0

               ; perform SMB 32-bit protected mode connect
               mov     ax,53B0H
               mov     bh,03H
               mov     ch,'i'
               mov     cl,'A'
               int     15h
               jc      connect_failed

               ; save off SMB Info
               mov     [SMB32CodeSeg],ax
               mov     [SMB32Off],ebx
               mov     [SMB16CodeSeg],cx
               mov     [SMB32DataSeg],dx
               mov     [SMB32CodeLen],si
               mov     [SMB32DataLen],di

               ; allocate three new consecutive GDT descriptors; one for data, one for code
               ; order MUST be 32-bit code first, then 16-bit code, followed by data segment
               .....
               .....
               mov     [SMB32CodeSel],new 32-bit code selector
               mov     [SMB16CodeSel],new 16-bit code selector
               mov     [SMB32DataSel],new data selector

               ; initialize 32-bit protected mode code descriptor
               ;    selector start at [SMB32CodeSeg]:0
               ;    segment length of [SMB32CodeLen]
               .....
               .....

               ; initialize 16-bit protected mode code descriptor
               ;    selector start at [SMB16CodeSeg]:0
               ;    segment length of [SMB32CodeLen]
               .....
               .....

               ; initialize 32-bit protected mode data descriptor
               ;    selector start at [SMB32DataSeg]:0
               ;    segment length of [SMB32DataLen]
               .....
               .....

               ; build 32-bit entry point into SMB BIOS
               mov     ax,[SMB32codeSel]
               mov     [SMB32entry+2],ax
               mov     ax,[SMBSegOff]]
               mov     [SMB32entry],ax
```

SMBus ReadWord/WriteWord (SB AtRate function)

```
//
// Read Smart Battery At Rate Value
//
   //
   // Issue the request
   //
   IssueRequest
   ErrorCode = SMBusRequest(ReadWordProtocol, SmartBatteryAddress,
                            AtRateFunctionCode)
      if SMBusOK
         // This is the normal exit
         // Schedule GetResults to get the results at later time ...
      end if

   Case (ErrorCode)
      SMBus Busy :
         // Re-schedule IssueRequest for a later time.
         // Other devices are using the bus.  For example, the Smart Battery
         // is communicating with a Smart Charger

      SMBus Transaction Pending :
         // Re-schedule IssueRequest for a later time.  Another transaction
         // is in progress.

      SMBus Unknown Error :
         // Retry as appropriate or giveup

      SMBus Device Access Denied :
         // The response is device specific
         // The BIOS will not allow access to this device

      SMBus Device Address Not Acknowledged :
         // The response is device specific
         // The device is not acknowledging - e.g., battery removed so it can not
         // acknowledge its address

      SMBus Device Error Detected :
         // The response is device specific
         // The device has signaled an error

      SMBus Device Command Access Denied :
         // The response is device specific
         // The BIOS will NOT allow this specific command to the device
         // For example - The BIOS may not allow a Smart Battery's alarm values to
         // be changed

      SMBus Protocol Not Supported :
         // Do not use this protocol
         // The BIOS MUST support all the protocols used by devices in the system

      SMBus Not Supported :
         // Don't try to access the BIOS again - it doesn't support SMB
   EndCase

   End IssueRequest

      //
   // Get the results
   //
   GetResults
   ErrorCode = SMBusDataAndStatus(ReadWordProtocol, SmartBatteryAddress,
                                  AtRateFunctionCode, &at_rate)
      if SMBus OK
         Return OK
         // were done - everything went OK
      end if
```

```
   case (ErrorCode)
      SMBus Transaction Pending :
         // Re-schedule GetResults for a later time.
         // A transaction is in progress, but is not complete enough to return data

      SMBus Device Address Not Acknowledged :
         // The response is device specific
         // The device is not acknowledging

      SMBus Device Error Detected :
         // The response is device specific
         // The device has signaled an error

      SMBus No Transaction Pending :
         // Do not call again until another request is made
         // No transaction is in progress - we shouldn't be here

      SMBus Timeout :
         // Try the request again
         // The bus has timed out and the pending transaction killed

      SMBus Busy :
         ErrorCode = SMBusRequestAbort(ReadWordProtocol, SmartBatteryAddress,
                                       AtRateFunctionCode)
         IssueRequest
         // Abort the request we think is pending then try the request again

      SMBus SMI Detected :
         ErrorCode = SMBusRequestAbort(ReadWordProtocol, SmartBatteryAddress,
                                       AtRateFunctionCode)
         return Interrupted by SMI to caller
         // We need to tell the BIOS to cancel the pending transaction
         // and return error to the caller so they can repeat the previous request
   end case
   end GetResults

// AtRate now contains the Smart Battery's AtRate value
```

SMBus BlockRead (SB ManufacturerName function)

```
//
// Read Smart Battery Manufacturer Name
//
   global first time flag
   //
   // Issue the request
   //
   IssueRequest
   ErrorCode = SMBusRequest(BlockReadProtocol, SmartBatteryAddress,
                            ManufacturerNameFunctionCode)
      if SMBusOK
         first time flag = true
         // This is the normal exit
         // Schedule GetResults to get the results at later time ...
      end if

   Process Errors as in previous example
   End IssueRequest


   //
   // Get the results
   //
   GetResults
   ErrorCode = SMBusDataAndStatus(BlockReadProtocol, SmartBatteryAddress,
                                  ManufacturerNameFunctionCode, &data)
      if SMBus OK
         if first time flag
            buffer length = data.msb (DH)
            copy data.lsb (DL) into the buffer
            first time flag = false
         else

            copy data.msb (DH) into the buffer
            copy data.lsb (DL) into the buffer
         end if
         decrement buffer length by the byte count (1 or 2)

         if buffer length = 0
            return OK
         else
            re-schedule GetResults for a later time to get more data
         end if
      end if

   case (ErrorCode)
      SMBus Transaction Pending :
         // Re-schedule GetResults for a later time.
         // A transaction is in progress, but is not complete enough to return data

      SMBus Busy :
         ErrorCode = SMBusRequestAbort(BlockReadProtocol, SmartBatteryAddress,
                                       ManufacturerNameFunctionCode)
         IssueRequest
         // Abort the request we think is pending then try the request again

      SMBus SMI Detected :
         ErrorCode = SMBusRequestAbort(BlockReadProtocol, SmartBatteryAddress,
                                       ManufacturerNameFunctionCode)
         return Interrupted by SMI to caller
         // We need to tell the BIOS to cancel the pending transaction
         // and return error to the caller so they can repeat the previous request

       Other Errors : see previous example
   end case
   end GetResults
```

SMBus BlockWrite

```
//
// Write Block
//
   //
   // Issue the request
   //
   IssueRequest
   ErrorCode = SMBusRequest(BlockWriteProtocol, DeviceAddress,
                            FunctionCode, BlockLength (DH) , DataBlock[0](DL) )
      if SMBusOK
         // This is the normal exit
         // Schedule Continue to continue the block write transfer at a later time ...
      end if

   Process Errors as in previous example
   End IssueRequest


   //
   // Continue the block write transfer
   //
   Continue
   ErrorCode = SMBusContinuation(BlockWriteProtocol, DeviceAddress,
                                 OneOrTwoBytes, DataBlock[N] (DH), DataBlock[N+1] (DL) )
      if SMBus OK
            if more data in block then
                adjust N index to point to next byte to transfer
                re-schedule Continue for a later time to send more data
            end if
      end if

   case (ErrorCode)
      SMBus SMI Detected :
         ErrorCode = SMBusRequestAbort(BlockWriteProtocol, DeviceAddress,
                                       FunctionCode)
         return Interrupted by SMI to caller
         // We need to tell the BIOS to cancel the pending transaction
         // and return error to the caller so they can repeat the previous request

       Other Errors : see previous example
   end case
   end Continue
```

SMBus Disconnect

```
        mov     ax,     53b0h   ; SMBus access code
        mov     bh,     05h     ; SMBus disconnect command
        mov     ch,     'i'
        mov     cl,     'A'
        call    far [smb_off]  ; use the pointer from the connect call to access the BIOS
        jc      command_failed ; carry flag is set indicating an error
                               ; check al for error code

        ; disconnect succeeded - proceed
```

SMBus Device Address (Get a list of the SMBus devices present)

```
count           db      0
                ; read all device addresses

loop1:          mov     ax,53B0H
                mov     bh,06H
                mov     bl,[count]
                mov     ch,'i'
                mov     cl,'A'
                int     15h
                jc      error_or_maybe_no_more_devices

                ;;; save off bl here (which contains this device's address)

                mov     bl,[count]
                inc     bl
                mov     [count],bl

                jmp     loop1
```

###