APPLICATION NOTE

How to Implement an SMBus Controller Using the 80C51SL KBC

Michael Camp/Yim Pun
Mobile & Home Products Group

Intel Corporation

Title: How to Implement an SMBus Controller using the 80C51SL KBC

Intel Ref. No:
FaxBack No:

Products Covered: MCS-51 Product Line
Date/Version: 11/08/94 Ver. 1.0
Related Info:
Keywords: SMBus, MCS-51, Smart Battery

Abstract

This application note describes how to add a Smart Battery to your system using the 80C51SL keyboard controller. The 80C51SL acts as the host for the System Management Bus (SMBus). The SMBus is used to communicate with the Smart Battery. It allows the host to query the battery's status and receive alarm messages. The schematic and functional description for the additional SMBus interface hardware is supplied. Software design considerations and suggestions for the keyboard BIOS extension that supports SMBus protocol are given.

Table of Contents

1. Introduction: System Level Overview

1.1 Smart Battery System Overview

The Smart Battery Specification presents an ideal solution for batteries used in portable electronic equipment such as laptop computer systems, cellular telephones or video cameras. Batteries presently pose a number of problems from both the user's and the equipment's perspective. First and foremost, they represent an unpredictable source of power. Typically a user has little advance knowledge that their battery is about to run out or how much operating time is left. Second, equipment powered by the battery cannot determine if the battery, in its present state, is capable of supplying adequate power for an additional load (such as spinning up a hard disk). Third, battery chargers must be individually tailored for use with a specific battery chemistry and may cause damage if used on another battery with a different chemistry.

This system, as depicted below in figure 1, shows how data flows on the SMBus between the Smart Battery, System Host, Smart Battery Charger and other SMBus devices. A more detailed description of the electrical interface and data protocols can be found in the supplementary documentation (refer to the "References" section).

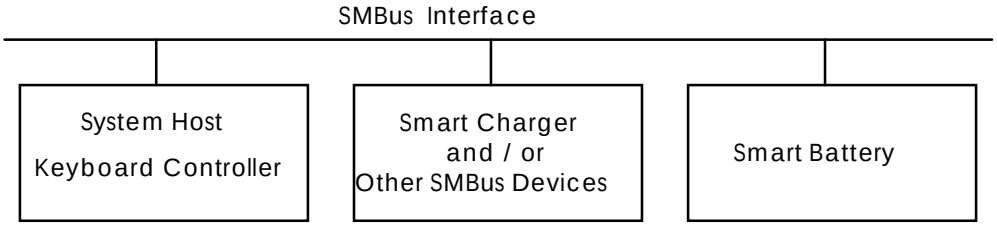


Figure 1. Typical Smart Battery System

1.2 SMBus Overview

The System Management Bus (SMBus) is a two-wire interface through which chips can communicate with the rest of the system. It uses I²C as its backbone. A system using SMBus passes messages to and from devices instead of using individual control lines which reduces pin count. Accepting messages ensures future expandability. With System Management Bus a device can

- Provide manufacturer information,
- Tell the system what its model/part number is,
- Save its state for a suspend event,
- Report different types of errors,
- Accept control parameters, and
- Return its status.

1.2.1 Difference between I^2C and SMBus.

These are the major differences between SMBus and I^2C:

- SMBus is based on fixed voltage levels, the I^2C levels are scaleable.
- SMBus specifies a minimum operational clock speed.
- SMBus specifies the protocols that a device is allowed to use when communicating with the host as a slave.

1.2.2 Electrical Characteristics

SMBus voltages and source currents deviate from the original I²C specification in order to allow today's chips to work on a future SMBus with a much lower operating voltage. These changes also make it easier for a battery manufacturer to build in ESD protection. The protocol deviates from the original I²C electrical characteristics in the following ways:

AC Specifications

Symbol	Parameter	Limits		Units	Test Conditions
		Min	Max		
FSMB	SMBus Operating Frequency	10	100	KHz	
TFREE	Bus free time between Start and Stop Condition	4.7		µs	
TSHLD	Hold time after Start Condition	4.0		µs	
TSRTSUP	Repeated Start Condition setup time	250		ns	
TSPSUP	Stop Condition setup time	4.0		µs	
TDHLD	Data hold time	0		ns	
TDSUP	Data setup time	250		ns	
THOG	Message Buffering Time		10	ms	
TLOW	Clock low period	4.7		µs	
THIGH	Clock high period	4.0		µs	
THL	Clock/Data Fall Time		300	ns	
TLH	Clock/Data Rise Time		1000	ns	

The minimum frequency specification is intended to prevent components from taking too long to complete a message. The bus may be at 0KHz when idle.

A device is allowed to extend one clock cycle up to THOG ms after the final slave address is acknowledged in any one message. In this way, a device cannot hold the bus for an extended period of time.

Every device must be able to resolve slave addresses at FSMB Max.

DC Specifications

Symbol	Parameter	Limits		Units	Test Conditions
		Min	Max		
VIL	Data, Clock Input Low Voltage	-0.5	0.6	V	
VIH	Data, Clock Input High Voltage	1.4	5.5	V	
VOL	Data, Clock Output Low Voltage		0.4	V	@ ISINK MIN
ILEAK	Input Leakage		±1	µA	
ISINK	Device Sink Current	100	350	µA	
CSMB	Bus Capacitance		100	pf	@ 100 KHz

The System Management Bus is intended to typically run at 3.3V. The electrical specifications ensure that chips designed today will still work on a future bus implementation that is running at 2.0V or even lower.

Components attached to SMBus may be running at different voltages. The System Management Bus does not assume that all devices will know what V_{DD} is. The voltage levels are therefore fixed.

1.2.3 Protocol and Usage Model

The System Management Bus Specification refers to three types of devices:
- *slave* device that is receiving or responding to a command.
- *master* device that issues the command, generates the clocks, and terminates the transfer.
- *host* device, a specialized master that provides the main interface to the system's CPU.

There may be either no host or one host in a system. An example of a hostless system is a simple battery charging station. The station might sit plugged into a wall waiting to charge a smart battery.

A device can be designed so that it is never a master, only a slave. A device may act as a slave most of the time, but in special instances it may become a master or vice versa. The host, mostly a master, but in special cases becomes a slave to receive critical messages.

Device Identification -- Slave Address

Each device that uses the System Management Bus has an unique address called the *Slave Address*. Masters and the host have a slave address for those instances when another master wants to talk with them. For reference, the following Slave Addresses are reserved by the I²C specification and thus cannot be used by any of the devices on this particular interface:

Slave Address Bits 7-1	R/W bit Bit 0	Description
0000 000	0	General Call Address
0000 000	1	START byte
0000 001	X	CBUS address
0000 010	X	Address reserved for different bus format
0000 011	X	Reserved for future use
0000 1XX	X	Reserved for future use
0001 000 -- 1110 111	X	SMBus addressing (see next table)
1111 0XX	X	10-bit slave addressing
1111 1XX	X	Reserved for future use

In addition to the above reserved addresses, the following addresses are reserved for the System Management Bus.

Slave Address	Description
0001 000	SMBus Host
0001 100	SMBus Alert Response Address
0101 000	reserved for ACCESS.bus host
0110 111	reserved for ACCESS.bus default address
1001 0XX	Unrestricted Addresses

The SMBus Alert Response Address (0001100) can substitute for device master capability. See Appendix A in the System Management Bus Specification for details.

Unrestricted addresses (10010XX) are universally available. They are not intended for production parts and will never be assigned to any device. They are provided for prototyping and experimenting. Addresses not specified here or within the System Management Bus Specification are reserved for future use. All 10 bit slave addresses are reserved for future use. The host should be able to support access to 10-bit devices.
The host has the lowest address so that emergency messages going to the host have the highest priority. Emergency messages may carry the I²C General Call address if they pertain to more than one device.

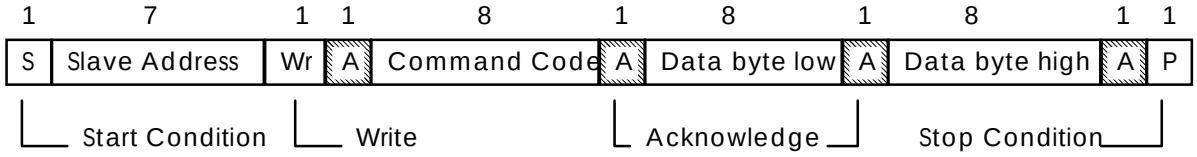
Using a SMB Device

A smart SMBus device will have a set of commands by which data can be read and written. Command arguments and return values can vary in length. Accessing a command that does not exist or is not supported causes an error condition. In accordance with the I²C specification, the Most Significant Bit (MSB) is transferred first.

There are eight possible command protocols for any given device. A slave device may use any or all of the eight protocols to communicate. The host device should be able to support all command protocols. The modes are Quick Command, Send Byte, Receive Byte, Write Word, Read Word, Process, Block Read, and Block Write. The Smart Battery only uses three of the eight commands they are Write Word , Read Word and Block Read. The formats for these commands are given below.

Write Word

The first byte of a Write Word access is the command code. The two bytes are the data to be written. In this example the master asserts the slave device address followed by the write bit. The device acknowledges and the master delivers the command code. The slave again acknowledges before the master sends the data word (low byte first). The slave acknowledges each byte according to the I²C specification, and the entire transaction is finished with a stop condition.

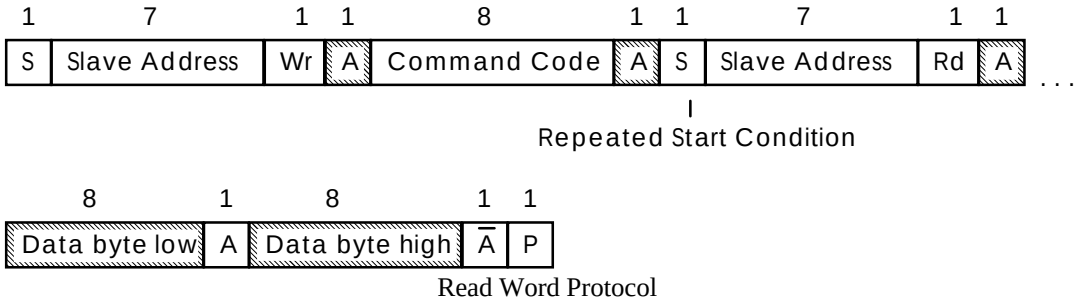


Write Word Protocol

Read Word

Reading data is slightly more complex than writing data. First the host must write a command to the slave device. Then it must follow that command with a repeated start condition to denote a read from that device's address. The slave then returns two bytes of data.

Note that there is not a stop condition before the repeated start condition, and that a "Not Acknowledge" signifies the end of the read transfer.

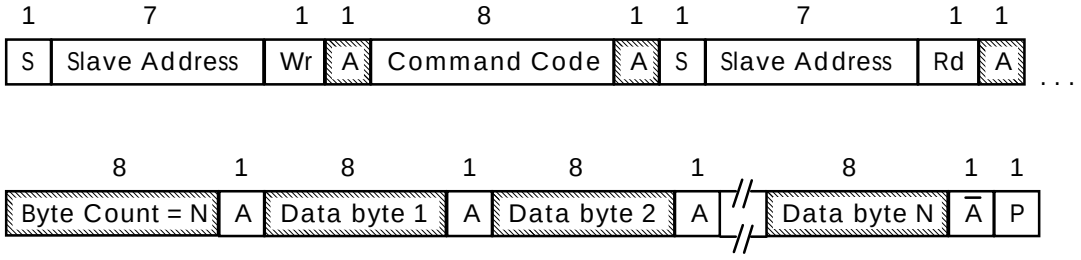


Read Word Protocol

Block Read

The Block Read begins with a slave address and a read condition. After the command code the host issues a byte count that describes how many more bytes will follow in the message. If a slave had 20 bytes to send, the first byte would be the number 20 (14h), followed by the 20 bytes of data. The byte count may not be 0. A Block Read can transfer a maximum of 255 bytes.

A Block Read differs from a block write in that the repeated start condition exists to satisfy the I²C specification's requirement for a change in the transfer direction.

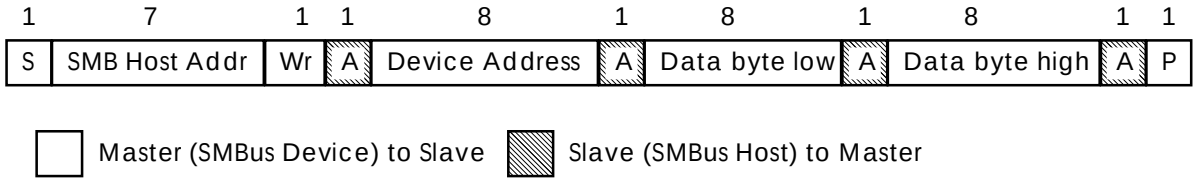


Block Read

Communicating with the Host

A message destined for the host could appear from an unknown device in an unknown format. To prevent possible confusion on the hosts part, only one method of communication is allowed -- a modified Write Word. This protocol is used when an SMBus device becomes a **master** to communicate with the SMBus host acting as a **slave**.

Device to Host communication will begin with the host address. The message's Command Code will actually be the initiating device's address. The host now knows the origin of the following 16 bits of device status.



7-bit Addressable Device to Host Communication

Reporting Errors

Any transfer can be aborted by either the slave or the master -- the master can issue a Stop Condition and the slave can withhold acknowledgment after any byte to terminate the transfer. This latter scenario is used for reporting errors.

If the device detects an error, it will withhold acknowledgment. The master can later visit the slave's Error Flag (if it is supported) to find out what went wrong. It is optional for the master to check and it is optional for the slave to provide the Error Flag.
Withholding acknowledgment is required for the last byte in a read operation under the I²C specification. This acknowledgment, or lack thereof, is generated by the master and therefore will not be interpreted as an error.

1.3 Keyboard Controller

The existing 80C51SL keyboard controller, which has the secondary power management ports can be used to emulate the SMBus protocols on many PC systems. Using the existing 80C51SL keyboard controller to interface to the SMBus to support the Smart Battery is the most efficient implementation since it requires relatively few hardware changes and doesn't affect the operation of the current keyboard controller design. With the SMB BIOS specification 1.0 and the SMB keyboard interface specification, all SMBus functions can be easily incorporated into the existing System and Keyboard BIOS's. This allows an existing design to be upgraded easily to take full advantage of all the features and capabilities that the SMBus and Smart Battery can offer. This application note shows a reference SMB implementation using the existing 80C51SL with the secondary power management ports.

2. Hardware Description

2.1 Design Considerations

To add a SMBus controller to a design, certain requirements need to be met. First of all, the 80C51SL must have four unused I/O pins and one available Interrupt pin. There must be enough board space to fit the required external logic as shown in Figure 3. The 80C51SL must have secondary power management ports, fifteen unused registers and enough room in the keyboard BIOS ROM for about 870 additional bytes of code.

2.2 SMB System

Figure 2 shows a block diagram of the SMBus system. The 80C51SL communicates to the host CPU over the ISA bus through the secondary power management ports 68h and 6Ch. All SMB keyboard commands are accessed through the power management ports instead of the conventional keyboard controller port to avoid conflicts. An additional SMB keyboard interface logic is required between the 80C51SL and the SMBus to resolve the contention among SMB devices and to detect alarm messages from other SMB masters. The additional SMBus logic consists of two 74HC74's, one 74LS02 and one 74LS05. The schematics for the SMB keyboard interface logic are as shown in Figure 3.

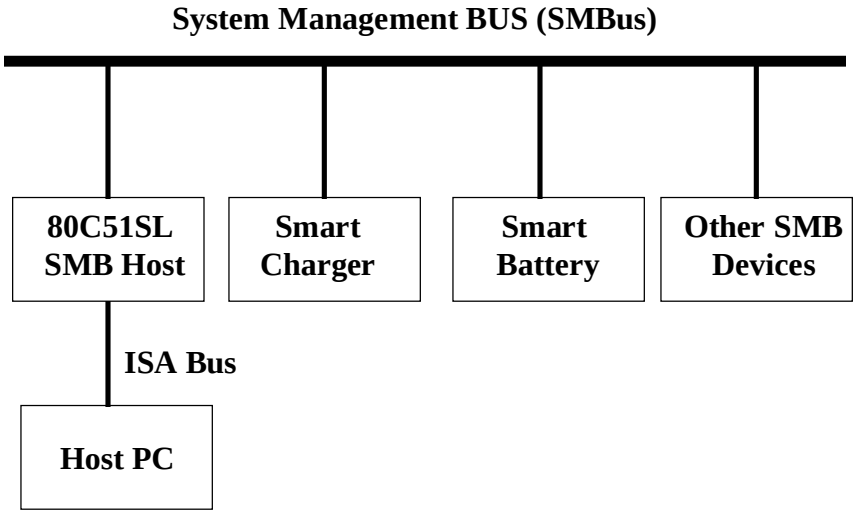


Figure 2. Block Diagram of Smart Battery System

2.2.2 SMB Keyboard Controller Interface Implementation

The implementation for the SMB external logic interfacing to 80C51SL is shown in figure 3. This logic resolves the contention among SMB devices and to detect alarm messages from other SMB masters. The implementation meets the SMBus electrical specifications by providing open collector outputs for the SMBus Clock and Data signals and level shifted signals for the 8051SL inputs.

External Logic for SMB

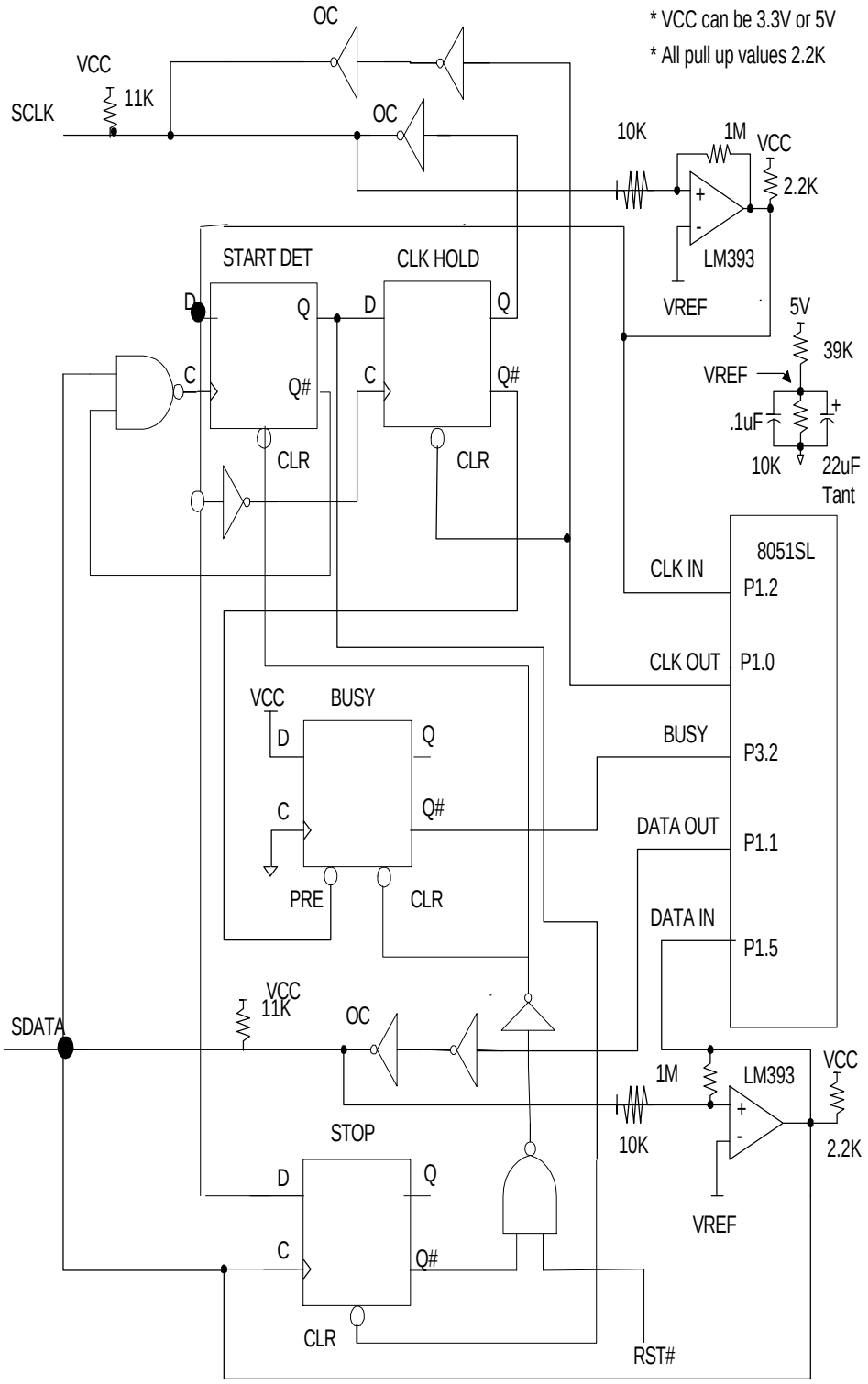


Figure 3. External Logic for the SMBus

The external logic circuit functions as follows: A start condition on the SMBus, SDATA goes low while SCLK is high, is captured by the flip-flop labeled START DET. The output of START DET goes into the CLK HOLD flip-flop holding SCLK low until the 8051SL drives CLK OUT low and then high. This keeps a SMBus device from sending more data before the KBC is ready to handle it. The output of CLK HOLD goes into the BUSY flip-flop that generates an interrupt to the 8051SL. This interrupt is used to notify the KBC that a SMBus message is starting. The interrupt is disabled when the KBC is acting as the SMBus host and directly controlling SCLK and SDATA. The fourth flip-flop labeled STOP is checking the SMBus for the Stop condition, SDATA goes high while SCLK is high. The STOP flip-flop clears the BUSY flip-flop to signal the end of a SMBus message. The LM393 contains two comparators that are used to shift the SMBus voltages to valid logic signals for the 8051SL. The trip point for the comparators is set at 1V. This appears as VREF on the schematic. A parts list for the External SMBus logic is given in Table 1.

Table 1. Parts List for External SMBus Logic

Qty	Part	Package
1	80C51SL Keyboard Controller	100 pin PQFP
1	LM393 Dual Comparator	8 pin dip
1	74LS05 Hex Open Collector Inverter	14 pin dip
1	74LS00 Quad 2 input NAND Gate	14 pin dip
2	74LS74 Dual D flip flop with pre/clr	14 pin dip
2	2.2k resistors	smt or sip
2	1 M resistors	
1	22uF Tantalum Capacitor	
1	100pf Ceramic Capacitor	
3	10 resistors	

This external hardware supports all of the Smart Battery capabilities. If a laptop design doesn't need to process Smart Battery Alarm messages then this hardware can be reduced. For instance a low capacity alarm message can be implemented by periodically polling the remaining capacity. The hardware for the "no alarms" design requires only two chips, the 74LS05 and the LM393. The 74LS05 Hex Open Collector Inverter is used to buffer the Data Out and Clock Out signals from the 80C51SL. The LM393 level shifts the SMBus signals into Data In and Clock In signals to the 80C51SL. If the SMBus signals from the Smart Battery can properly drive the 80C51SL inputs directly then the LM393 can be removed as well.

3. Software Description

3.1 Design Considerations

The keyboard BIOS was designed to support the three required SMBus commands needed for a Smart Battery. The other five SMBus commands can be added later to support different SMB devices. The BIOS also handles alarm messages coming from the Smart Battery to the SMB Host. The alarm message sets an alarm status bit. This bit is polled periodically to check for alarm messages. It is also possible to have the 80C51SL KBC to notify the host CPU with an interrupt.

3.2 SMB Architecture

Figure 4. shows a block diagram of all software layers required to support the SMBus. As shown in the diagram, four layers of software, namely, keyboard BIOS supporting SMB commands, SMB BIOS 1.0 Extension, SMB.386 VxD, and SMB applications, are required to support the SMB system. The keyboard BIOS supports SMB commands to communicate between the host system and the SMB devices having the 80C51SL as their SMB host. The SMB BIOS Extension hides the SMB hardware implementation from the OS and applications by providing a user-friendly BIOS interface. SMB VxDs facilitates Windows programming by providing a standard SMB API to all Windows applications.

In the diagram, each of the ellipses represent an interface specification. The SMB Keyboard BIOS Specification can be found in Appendix A of this document and the other specifications are available in the Smart Battery Binder.

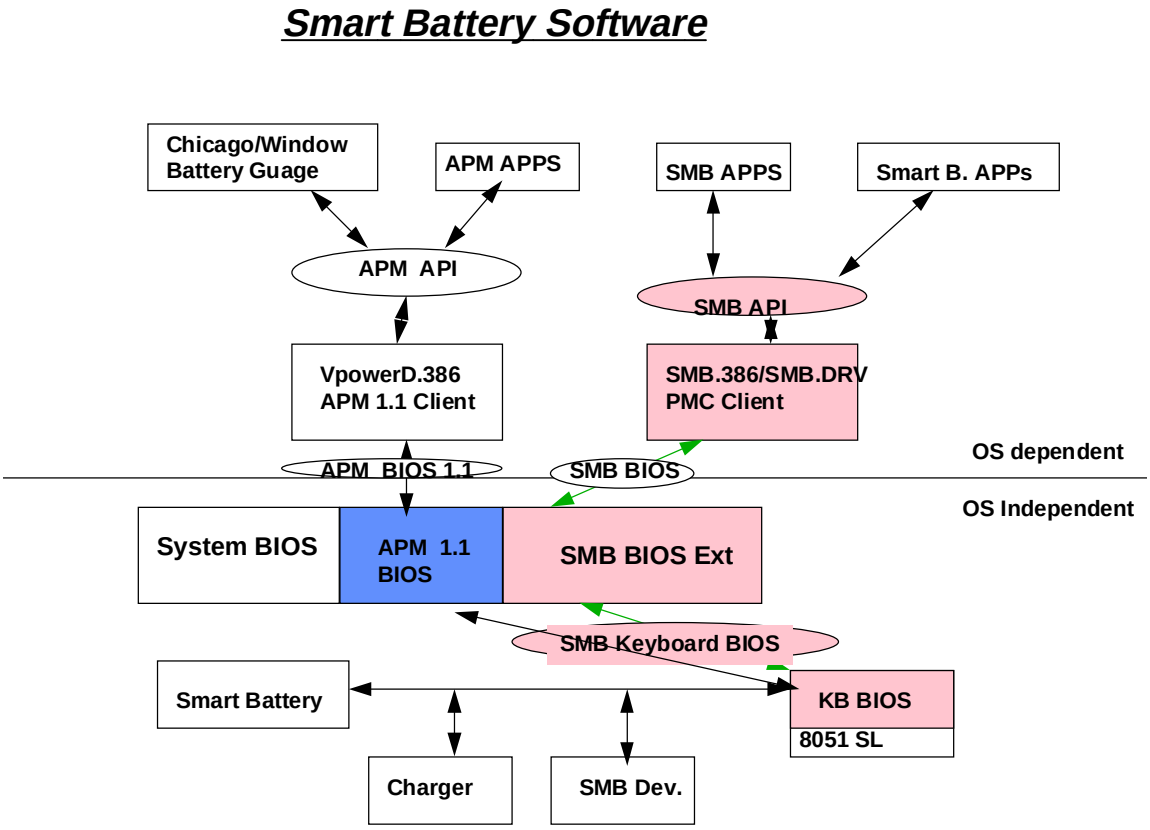


Figure 4. Smart Battery Software Block Diagram

Note: An 80C51 source code listing which implements the interface described in appendix A will be included in the final version of this Application Note.

4. Conclusion

Adding a Smart Battery to a laptop design requires only minimal hardware and software changes. The hardware requirements can be scaled to meet different price points. The external hardware replaces all of the temperature sensors and low voltage detection circuits that were needed before the Smart Battery. All of the software for the Smart Battery system is contained in the keyboard BIOS and SMB.386 VxD. The software provides the platform for new power aware applications. The user will be able to let the laptop know how long they want to work and the system will take over automatically adjusting the power savings to meet the goal.

The advantages of a Smart Battery system for the user include: greater confidence in the battery, longer battery life and chemistry independence. The standardization of battery intelligence will also enable the standardization of battery packs, which in turn will lead to new distribution channels, multiple sources for batteries and reduced manufacturing cost. All these factors will eventually translate into lower costs and greater convenience for end users. The advantages of a smart battery standard for OEMs include: reduced system design costs, faster time to market with new battery chemistries and flexible designs, capable of accommodating multiple battery chemistries. Because the charging algorithms for a standard smart battery are contained in the battery itself, an improvement in battery chemistry or charging algorithms doesn't require any changes in the laptop's hardware or software. Lastly, by using standard battery packs incorporating the battery intelligence standard, OEMs will also benefit from being able to eliminate costly inventories of custom batteries from single suppliers.

Appendix A Keyboard BIOS Interface Specification

This specification defines the Keyboard BIOS commands that are needed to support the Smart Battery System. It assumes that a 80C51SL KBC is being used to provide software emulation of the SMBus host. The design philosophy followed was to come up with the minimum set of SMBus commands that are needed to fully support the smart battery. Also, a conscious effort was made to reduce the computational burden on the 80C51SL as much as possible.

This specification will vary according to the specific implementation. For example, in this design the host polls the keyboard controller to see if an Alarm Message has occurred. In a system that doesn't poll the GET_ALARM command would not be needed and some other method of notifying the host would be used.

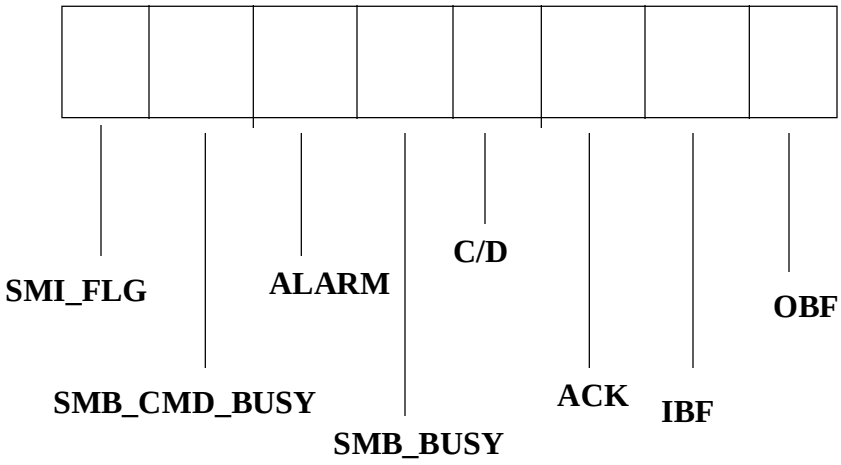
Command	Input	Output
1) WRITE_WORD	KEY_CMD, SLV_ADDR SMB_CMD, DATA WORD	Command Status
2) READ_WORD	KEY_CMD, SLV_ADDR SMB_CMD	Data Word + Command Status
3) WRITE_BLOCK	KEY_CMD, SLV_ADDR, SMB_CMD, DATA_LENGTH Block of Data Length Bytes	Command Status
4) READ_BLOCK	KEY_CMD, SLV_ADDR SMB_CMD, DATA_LENGTH	Block Data of Data Length Bytes + Command Status
5) GET_ALARM	KEY_CMD	Alarm Address, Alarm Data Word
6) SET_SMI_FLAG	KEY_CMD	None
7) CLR_SMI_FLAG	KEY_CMD	None

The **KEYBOARD_CMD** is a unique 1 byte command that is written to the Power Management Command Buffer at port 6CH to inform the 80C51SL which of the 8 SMB commands to execute. All other SMBus data is transferred through the Power Management Data Buffer at port 68H. All of the KEYBOARD_CMD codes are defined in the following table. Some of the codes are not used by the Smart Battery. These commands maybe used by other SMBus devices so they are defined here for completeness.

	Name	KEY_CMD	PURPOSE
1)	QUICK_CMD	80h	Not needed for Smart Battery
2)	SEND_BYTE	81h	Not needed for Smart Battery
3)	RCV_BYTE	82h	Not needed for Smart Battery
4)	WRITE_BYTE	83h	Not needed for Smart Battery
5)	READ_BYTE	84h	Not needed for Smart Battery
6)	WRITE_WORD	85h	Writes a Word of Data to a SMBus device
7)	READ_WORD	86h	Reads a Word of Data from a SMBus device
8)	WRITE_BLOCK	87h	Writes a Block of Data to a SMBus device
9)	READ_BLOCK	88h	Reads a Block of Data from a SMBus device
10)	PROC_CALL	89h	Not needed for Smart Battery
11)	GET_ALARM	8Bh	Gets the Alarm Address and Alarm Data Word
12)	SET_SMI_FLAG	8Ch	Sets the SMI flag
13)	CLR_SMI_FLAG	8Dh	Clears the SMI flag

SMB Status is one byte long and has the following flags:

Power Management Status Port (6CH)



The **SMI_FLG** bit indicates that the system management mode has intercepted the prior SMB requests.
The **SMB_CMD_Busy** bit indicates whether a SMB command is in progress.
The **ALARM** bit indicates that a alarm message has been received.
The **SMB_BUSY** bit indicates that the SMBus is busy.
The **ACK** bit indicates if the previous SMB command is acknowledged

SLV_ADDR is the Address of the Slave device on the SMBus to which this command is sent. The SMBus address assigned to the Smart Battery is 16H. A SMBus read command adds 1 to the SLV_ADDR to indicate that it is a read.

SMB_CMD is a 1 Byte command that informs the SMB device which function to perform. The Smart Battery has functions 00h through 23h. These functions are defined in the Smart Battery Interface - Communications Specification. Other SMB devices at different addresses will have other functions defined by SMB_CMD.

For each SMB requests, a command status byte is always returned by 80C51SL at the end of SMB commands to indicate the status of completion. The format of the command status byte is as shown below.

DATA_LENGTH is one byte long and it represents the number of bytes in the READ_BLOCK command. The current implementation limits the block size to between one and eight bytes. A lack of available on chip registers is the cause of this.

SET_SMI_FLAG is used by the SMI handler to indicate to the SMBus command that was interrupted that it was aborted. The interrupted command then knows to use the CLR_SMI_FLAG command to clear the SMI flag and restart the command.

CLR_SMI_FLAG is used by the interrupted SMBus command to clear the SMI flag before it restarts the SMBus command.

Appendix B SMBus Semaphore Solution for 80C51SL implementation.

A semaphore checking scheme is needed to correctly handle the multiple SMB requests from the APM BIOS and SMB BIOS interfaces. For example, a SMB request was initiated by a SMB driver through the SMB BIOS interface. Before the SMB command was completed, a time-out interrupt occurred , interrupting the SMB and generating a SMB request through the APM BIOS interface. As SMI has the highest priority, SMM handler will take over the control and start a new SMB command. This will confuse the 80C51SL in the way that it can only handle one SMB request at a time. The following described semaphore scheme will ensure that only one SMB request is handle by 80C51SL at any time and all SMB commands are serialized. Figure 5 shows the flow chart describing how the semaphore scheme works.

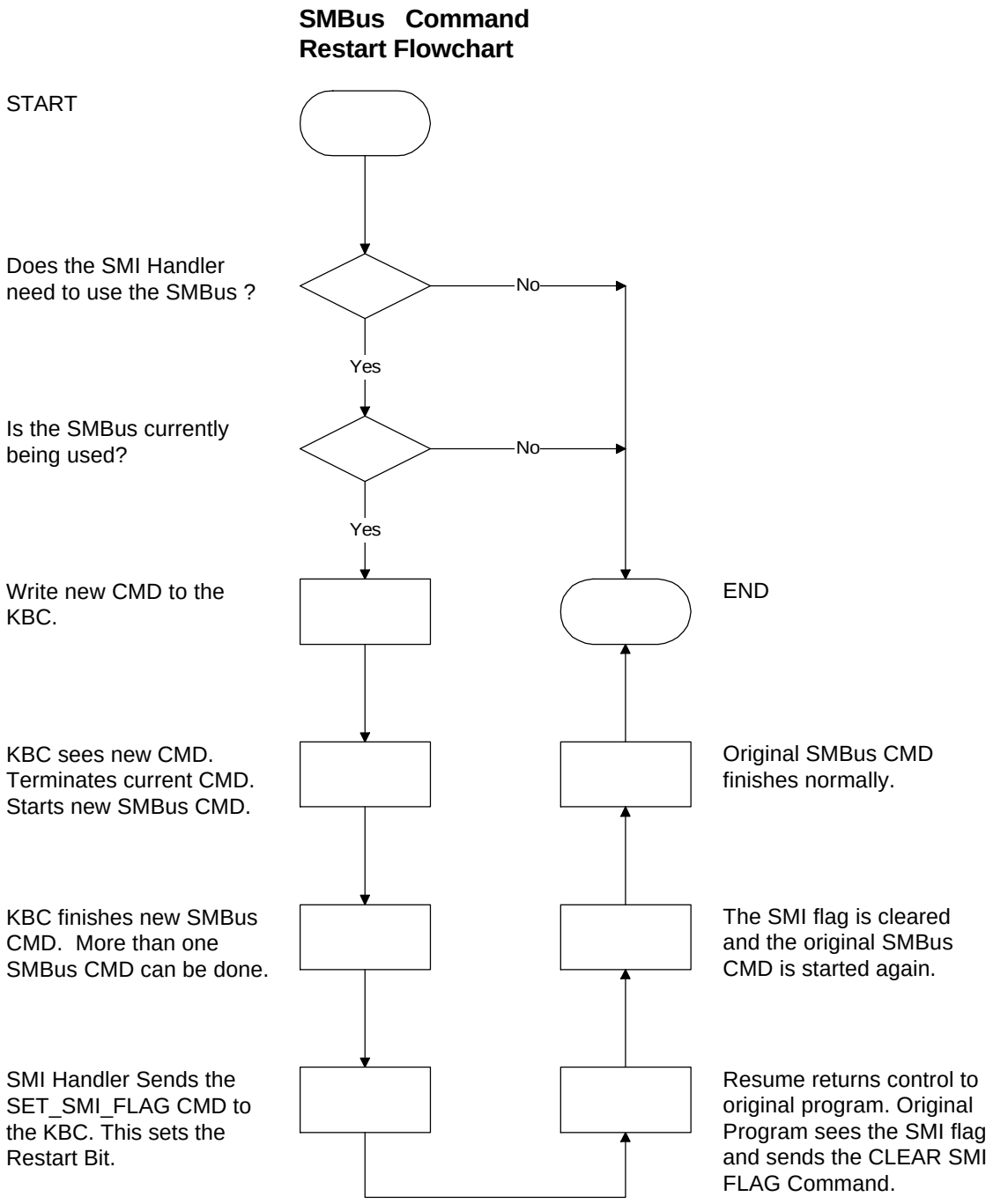


Figure 5. SMBus Command Restart Flowchart