Smart Battery System Specifications

System Management Bus Specification

Revision 1.1
December 11, 1998

Questions and comments regarding this specification may be forwarded to:
Email: smbus@sbs-forum.org
Or: questions@sbs-forum.org

For additional information on Smart Battery System Specifications, visit the SBS Implementer's Forum (SBS-IF) at:
www.sbs-forum.org

Table of Contents

Revision History

Revision Number	Date	Notes
1.0	2/15/95	General Release
1.1	12/11/98	Version 1.1 Release

1. Overview

1.1. What is System Management Bus?

The System Management Bus (SMBus) is a two-wire interface through which simple system and power management related chips can communicate with the rest of the system. It is based on the principals of operation of I²C.

SMBus provides a control bus for system and power management related tasks. A system using SMBus passes messages to and from devices instead of tripping individual control lines. Removing the individual control lines 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.

The System Management Bus may share the same host device and physical bus with I²C components provided that the electrical and timing specifications of this document are adhered to.

Intel conceived the System Management Bus originally, as the communication bus to accommodate Smart Batteries and other system and power management components. In 1994 SMBus became part of the On-board ACCESS.bus specifications. In January 1995 Philips announced in New York the royalty free status of ACCESS.bus devices including On-board ACCESS.bus compliant devices. In 1996 the Smart Battery System specifications were handed by Intel and Duracell to a group of 10 companies that formed the core group of the SBS. In 1997 the SBS Implementers Forum was formed and SMBus became part of the specifications handled by this group. The same year SMBus was incorporated into the ACPI specifications as the bus to communicate with the Smart Battery System and other system components, such as temperature sensors, etc. ACPI specifications also defined the SMBus host interface to the OS.

1.2. Audience

The target audience for this document includes:

- System designers implementing the System Management Bus Specification in their systems
- VLSI engineers designing chips to connect to the System Management Bus
- Software engineers writing support code for System Management Bus chips

1.3. Scope

This document describes the communications protocols available for use by devices on SMBus. Its original purpose was to define the communication link between an intelligent battery, a charger for the battery, and a microcontroller that communicates with the rest of the system. However, it can also be used to connect a wide variety of power-related devices.

The specification allows for multiple devices to attach to the System Management Bus. Information is exchanged through a simple index set specific to each device.

The SMBCLK and SMBDATA pins are similar to the clock and data pins found on an I²C bus. The SMBus electrical characteristics differ from those of I²C.

1.4. Supporting Documents

This specification assumes that the reader is familiar with or has access to the following documents:

- *The I²C-bus and how to use it*, Philips Semiconductors document #98-8080-575-01.
- *ACCESS.bus Specifications -- Version 2.2*, ACCESS.bus Industry Group, 370 Altair Way Suite 215, Sunnyvale, CA 94086 Tel (408) 991-3517
- *System Management Bus BIOS Interface Specification*, Revision 1.0, February 15, 1995
- *ACPI Specifications*, Version 1.0a, Intel Corporation, Microsoft Corporation, Toshiba Corp., July 1998 (http://www.teleport.com/~acpi)

2. General Characteristics

SMBus is a two-wire multi-master bus, meaning that more than one device capable of controlling the bus can be connected to it. A master device initiates a bus transfer and provides the clock signals. A slave device can receive data provided by the master or it can provide data to the master. Since more than one device may attempt to take control of the bus as a master, SMBus provides an arbitration mechanism, based on I2C and relying on the wired-AND connection of all SMBus interfaces to the SMBus.

If two or more masters try to place information on the bus, the first to produce a "ONE" when the other(s) produce a "ZERO" looses arbitration and has to release the bus. The clock signals during arbitration are wired-AND combination of all the clocks provided by SMBus masters. Bus clock signals from a master can only be altered by clock stretching or by other masters only during a bus arbitration situation.

In addition to bus arbitration, SMBus implements the I2C method of clock low extending in order to accommodate devices of different speeds on the same bus.

SMBus version 1.1 can be implemented at any voltage between 3 and 5 Volts +/- 10%. Devices can be powered by the bus VDD or by their own power source (such as Smart Batteries) and they will inter-operate flawlessly as long as they adhere to the SMBus electrical specifications.

The following diagram shows an example implementation of a 5 Volt SMBus with devices powered by the bus VDD inter-operating with devices powered by their own power supply.

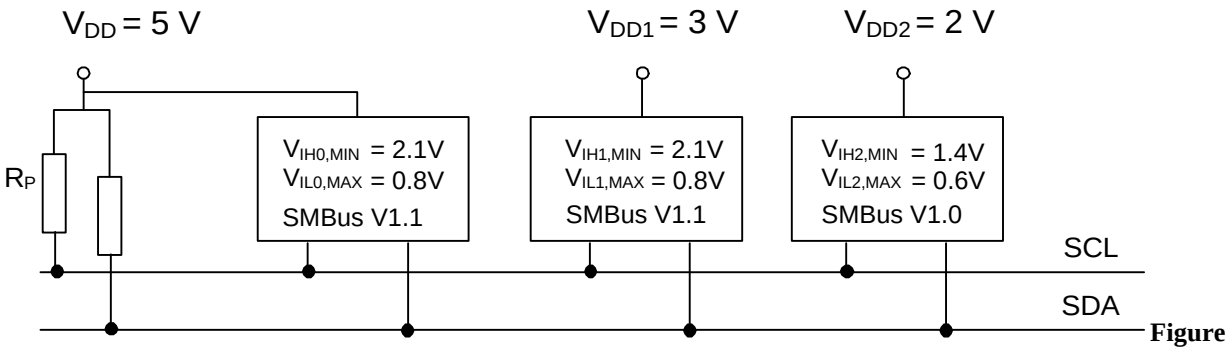


Figure 2.1: SMBus topology

In the specific example the device powered by VDD1=3 V is an SMBus Version 1.1 compliant device. The device powered by VDD2=2 V is an SMBus Version 1.0 compliant device. The VDD of the bus can be 3 to 5 Volts +/- 10% and there may be SMBus devices powered directly by the bus VDD. Both SCL and SDA lines are bi-directional, connected to a positive supply voltage through a pull-up resistor or a current source or other similar circuits. When the bus is free, both lines are HIGH. The output stages of the devices connected to the bus must have an open drain or open collector in order to perform the wired-AND function. Care should be taken in the design of both the input and output stages of SMBus devices, in order not to load the bus when their power plane is turned off.

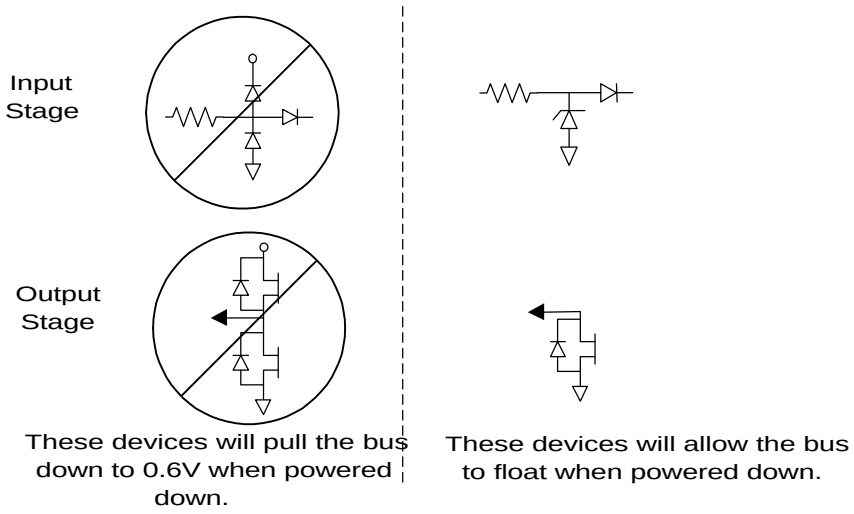


Figure 2.2: Input and output stage of SMBus devices

A device that wants to place a zero on the bus will have to drive the bus line to the defined logic low voltage levels. In order to place a logic "ONE" on the bus the device should release the bus line in order to let it be pulled high by the bus pull-up circuitry.

Devices adhering to version 1.0 of the SMBus specification will inter-operate with devices conforming to the SMBus Version 1.1 specification. Nevertheless, devices implementing the Version 1.1 $V_{IL,MAX}$ = 0.8 V will exhibit better noise margins.

The bus lines can be pulled high by a pull-up resistor or a current source or in cases that involve higher bus capacitance by a more sophisticated circuit that can limit the pulldown sink current while providing enough current during the low to high transition in order to maintain the rise time specifications of SMBus.

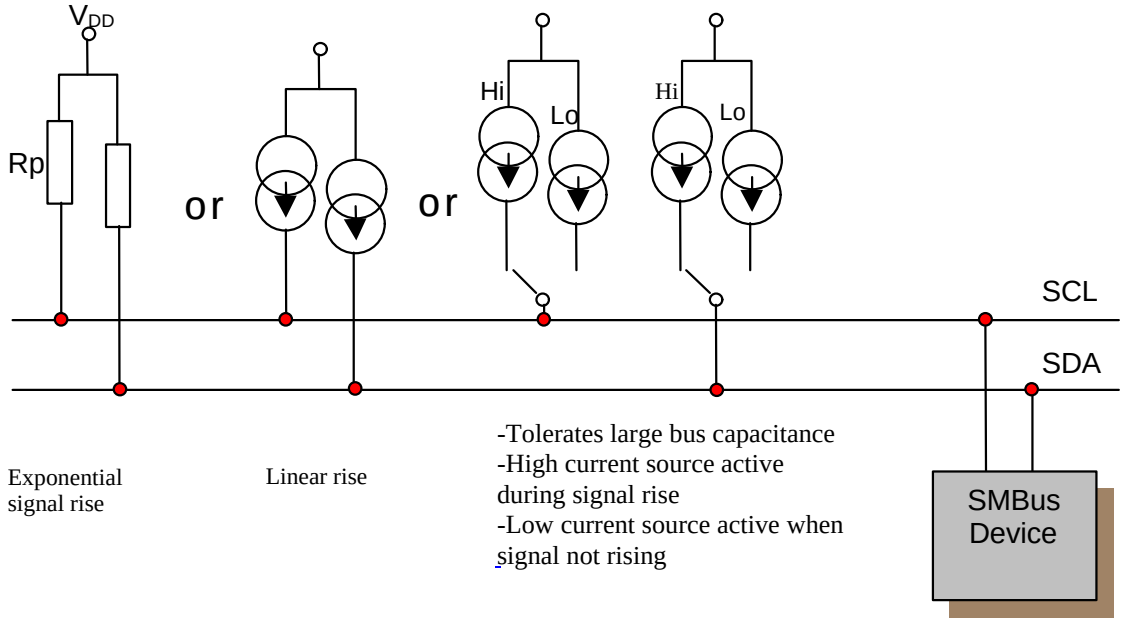


Figure 2.3: SMBus pullup circuitry

3. Bit transfers

SMBus uses fixed voltage levels of 0.8 and 2.1 Volts to define the logic “ZERO” and logic “ONE” on the bus respectively.

3.1. Data validity

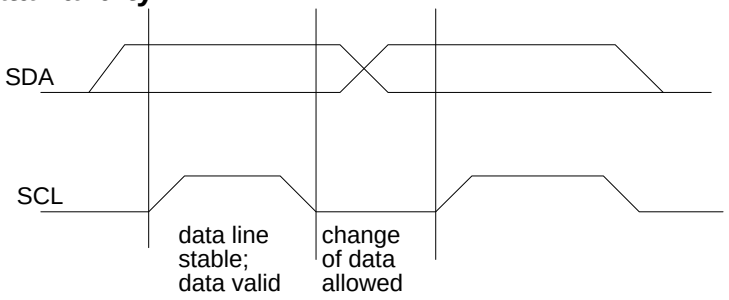


Figure 3.1: Data validity

The data on the SDA line must be stable during the “HIGH” period of the clock. Data can change state only when the SCL clock line is low.

3.2. Start and Stop condition

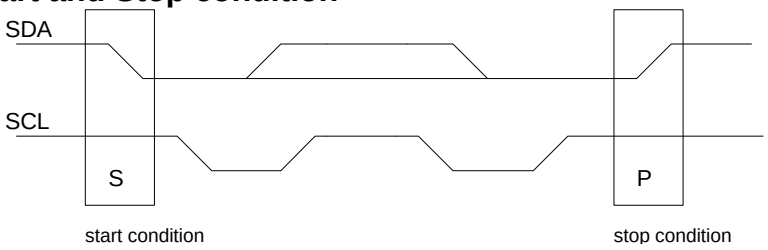


Figure 3.2: Start and Stop condition

As with I2C, two unique bus situations define a message START and STOP condition.
A HIGH to LOW transition of the SDA line while SCL is HIGH indicates a message START condition.
A LOW to HIGH transition of the SDA line while SCL is HIGH defines a message STOP condition.
START and STOP conditions are always generated by the bus master. After a START condition the bus is considered to be busy. The bus becomes free again after certain time following a STOP condition or after both the SCL and SDA lines remain high for more than 50 us.

4. Data transfers on SMBus

4.1. Byte format

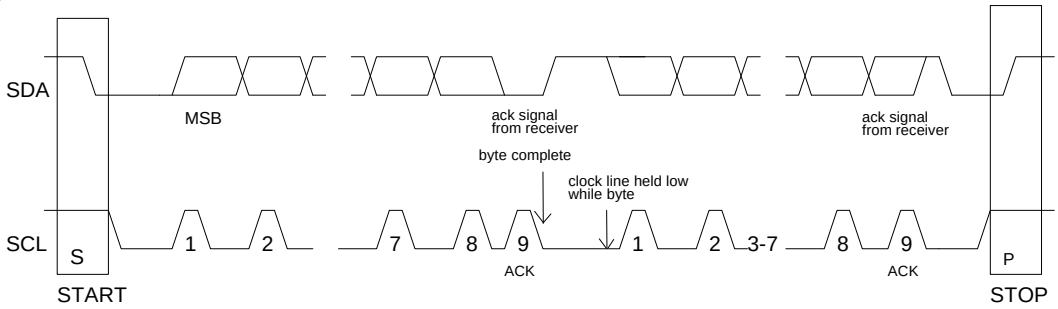


Figure 4.1: SMBus byte format

Every byte consists of 8 bits. Each byte transferred on the bus must be followed by an acknowledge bit. Bytes are transferred with the most significant bit (MSB) first.

4.2. Acknowledge (ACK) and not acknowledge (NACK)

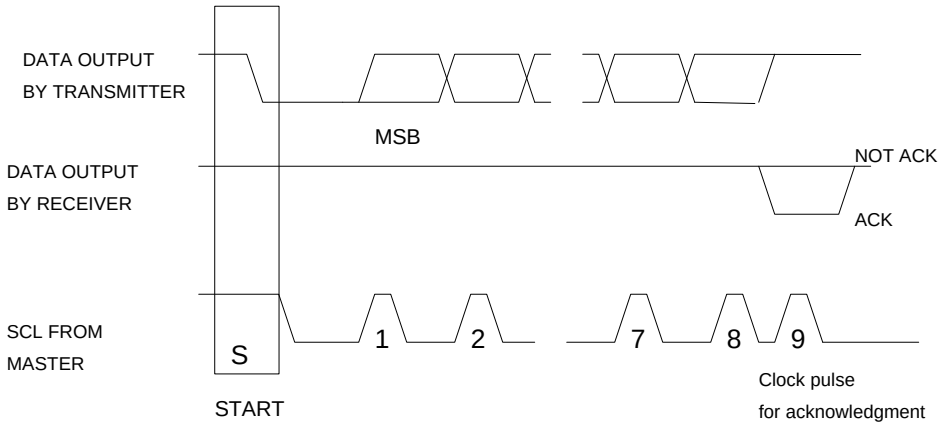


Figure 4.2: ACK and NACK signaling of SMBus

The acknowledge related clock pulse is generated by the master. The transmitter releases the SDA line (HIGH) during the acknowledge clock cycle. In order to acknowledge a byte, the receiver must pull the SDA line LOW during the HIGH period of the clock pulse according to the SMBus timing specifications. A slave device that wishes to not acknowledge a byte must let the SDA line remain HIGH during the acknowledge clock pulse.

An SMBus device has to acknowledge always its own address. SMBus uses this signaling to detect presence of detachable devices on the bus.

An SMBus slave device may decide to not acknowledge a byte in the following situations:

- The slave device is busy performing a real time task, or data requested are not available. The master upon detection of the NACK condition must generate a STOP condition to abort the transfer. Note that as an alternative, the slave device can extend the clock LOW period within the limits of this specification in order to complete its tasks and continue the transfer.

- The slave device detects an invalid command or invalid data. In this case the slave device must not acknowledge the received byte. The master upon detection of this condition must generate a STOP condition and retry the transaction.
- If a master-receiver is involved in the transaction it must signal the end of data to the slave-transmitter by not generating an acknowledge on the last byte that was clocked out by the slave. The slave-transmitter must release the data line to allow the master to generate a STOP condition.

SMBus is using the latter mechanism in order to detect whether a slave transmitter implements Packet Error Checking. In the case of a master-receiver, it will attempt to request more data from the slave transmitter by acknowledging the last data byte and continuing providing clocks requesting one more byte. If the slave transmitter implements Packet Error Checking, it will provide the Packet Error Code. The master receiver will check the validity of the Packet Error Code and if it is valid it will register the device as implementing Packet Error Checking. If the slave transmitter does not implement Packet Error Checking, it will provide either invalid data or no data at all. The master receiver will check the data validity and if data are not valid it will register the device as not implementing Packet Error Checking.

5. Arbitration and clock generation

5.1. Synchronization

A situation may occur that more than one master is trying to place clock signals on the bus at the same time. The resulting bus signal will be the wired AND of all the clock signals provided by the masters.

It is important for the bus integrity that there is a clear definition of the clock, bit by bit for all masters involved during an arbitration process.

A HIGH to LOW transition on the SCL line should cause all devices involved to start counting off their LOW period. As soon as a device finishes counting its LOW period it will release the SCL line. Nevertheless, the actual signal on the SCL may not transition to the HIGH state if another master with longer LOW period keeps the SCL line LOW. In this situation the master that released the SCL line will enter the SCL HIGH wait period. When all devices have counted off their LOW period, the SCL line will be released and go HIGH. All devices concerned at this point will start counting their HIGH periods. The first device that completes its HIGH period count will pull the SCL line LOW and the cycle will start again.

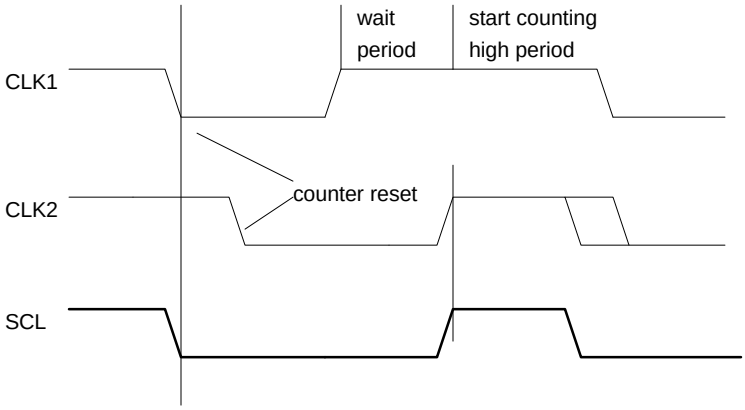


Figure 5.1: SMBus clock synchronization

This way a synchronized clock is provided for all devices, where the SCL LOW period is determined by the slowest device and the SCL HIGH period is determined by the fastest device.

5.2. Arbitration

A master may start a transfer only if the bus is free. The bus is free after a STOP condition or after the SCL line remains high for more than $t_{HIGH, MAX}$. Two or more devices may generate a START condition within the minimum hold time ($t_{HOLD:STA}$) resulting in a defined START condition on the bus.

Since the devices that generated the START condition may not be aware that other masters are contending for the bus, arbitration takes place on the SDA line while the SCL is HIGH. A master that transmits a HIGH level, while the other(s) master is transmitting a LOW level on the SDA line looses the arbitration and it is required to give up the bus.

The master that lost the arbitration may continue to provide clock pulses until the completion of the byte that he lost the arbitration. Arbitration in the case of two masters trying to access the same device may continue past the address byte. In this case arbitration will continue with the remaining transfer data. This mechanism requires that all SMBus devices are monitoring the actual state of the SDA line during every bus transaction.

If a master also incorporates a slave function and looses the arbitration during the address stage, it should check the actual address placed on the bus in order to determine whether another master is trying to access it. In this case the master that lost the arbitration must switch immediately to its slave receiver mode in order to receive the rest of the message.

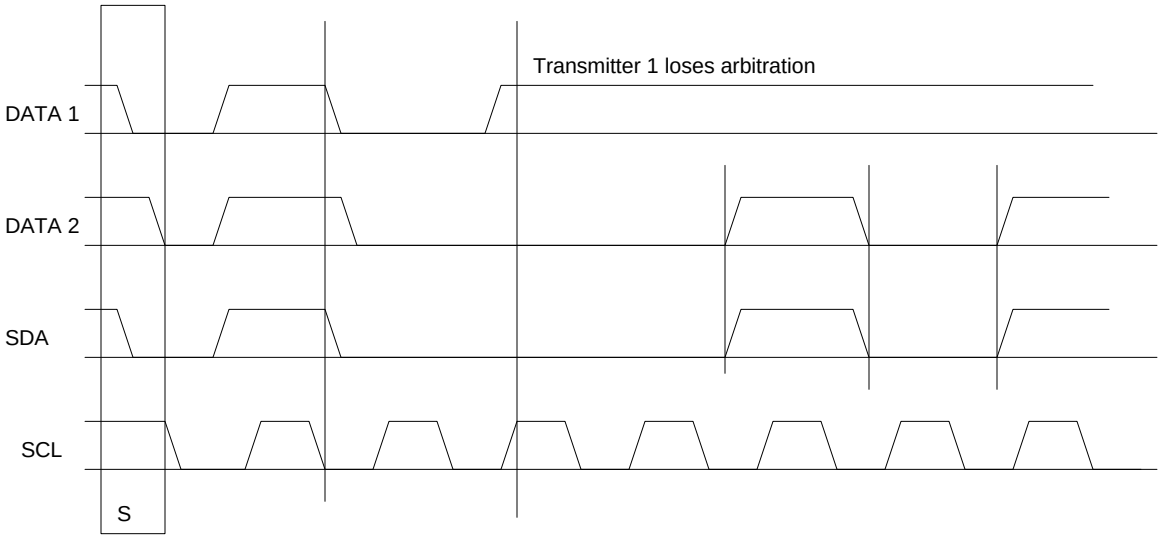


Figure 5.2: SMBus arbitration

During each bus transaction masters are still required to be able to recognize a repeated START condition on the bus. A device that detects a repeated START condition must quit the transfer.

Arbitration is not allowed between:

- A repeated START condition and a data bit
- A STOP condition and a data bit
- A repeated START condition and a STOP condition.

5.3. Clock low extending

SMBus provides a clock synchronization mechanism, similar to I2C, in order to accommodate devices of different speeds to co-exist on the bus. In addition to the bus arbitration procedure the clock synchronization mechanism can be used during a bit or a byte transfer in order to allow slower slave devices to cope with faster masters.

On the bit level a device can slow down the bus by extending periodically or whenever needed the clock LOW period.

Devices are allowed to stretch the clock during the transfer of one message up to the maximum limits described in the AC specifications of this document. Nevertheless, devices designed to stretch every clock cycle periodically should maintain the $f_{SMB,MIN}$ frequency of 10 KHz ($T_{SMB,MAX}$ = 100µs) in order to preserve the SMBus bandwidth.

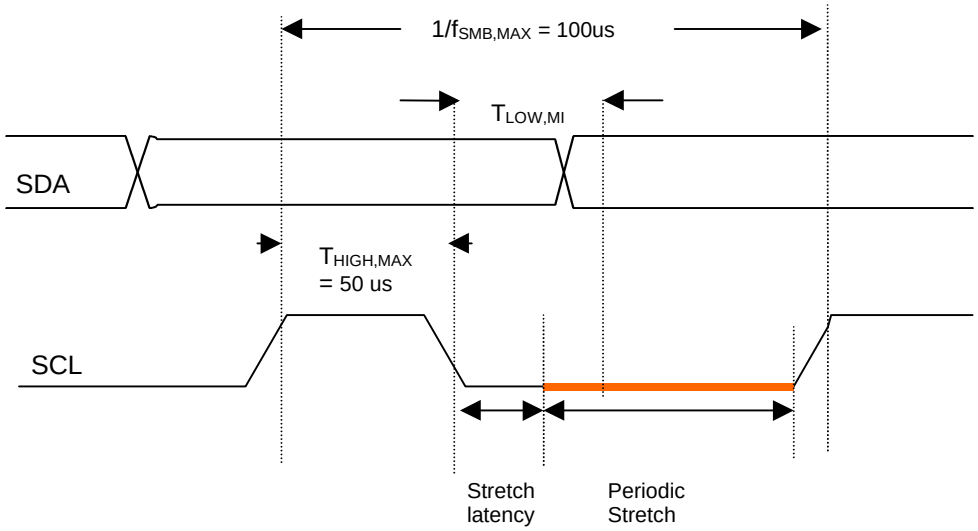


Figure 5.3: Periodic clock stretching by a slave SMBus device

Clock LOW extension or stretching , if necessary, must start before the $T_{LOW:MIN}$ of the bus has expired. Devices designed to stretch the clock periodically on every bit transfer should maintain the minimum bus frequency $f_{SMB,MIN}$ of 10 KHz. A slave device may select to stretch selectively the clock line during a specific bit transfer in order to process a real time task or check the validity of a byte. In this case the slave device must adhere to the $T_{TIMEOUT}$ and $T_{LOW:SEXT}$ specifications. Clock LOW extension may occur during each bit transfer including the clock provided prior to the ACK clock pulse.

A slave device may select to stretch the clock LOW period between byte transfers on the bus, in order to process received data or prepare data for transmission. In this case the slave device will hold the clock line LOW after the reception and acknowledgement of a byte. Again the slave device is responsible for not violating the $T_{LOW:SEXT}$ specification of SMBus.

During a bus transaction the master also can select to extend the clock LOW period between bytes or at any point in the byte transfer, including the clock LOW period after the byte transfer and before the acknowledgement clock. The master may need to extend the clock LOW period selectively in order to process data or serve a real time task. In doing so, the master must not exceed the $T_{LOW:MEXT}$ specification.

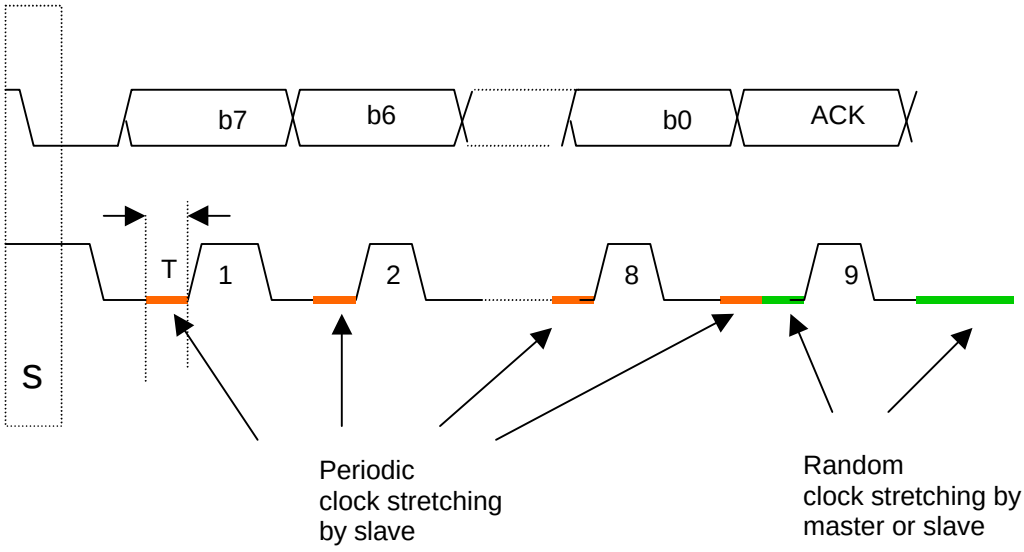


Figure 5.4: Periodic and random clock stretching

Both master and slave devices must adhere to the SMBus $T_{TIMEOUT}$ specification in order to maintain bus bandwidth and recovery from fatal bus conditions.

6. Data transfer formats

SMBus data transfers follow the format shown in the following figure.

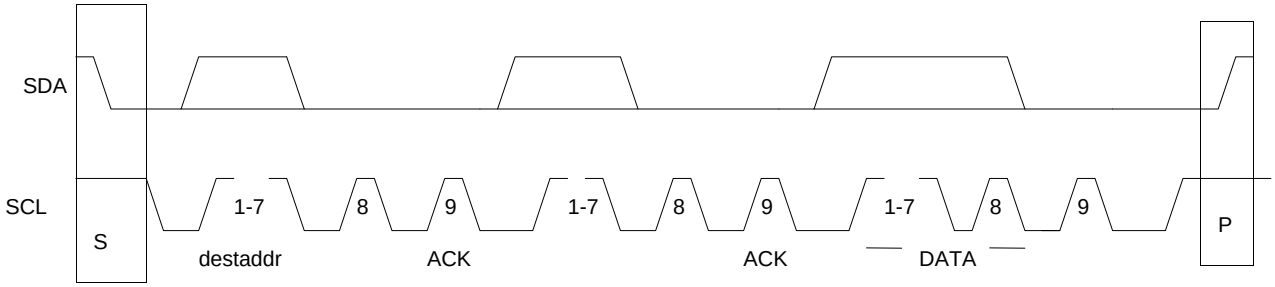


Figure 6.1: Data transfer over SMBus

After the START condition (S) the master places the 7-bit address of the slave device it wants to address on the bus. The address is 7 bits long followed by an eighth bit indicating the direction of the data transfer (R/_W); a ZERO indicates a transmission (WRITE) while a ONE indicates a request for data (READ). A data transfer is always terminated by a STOP (P) condition generated by the master. The SMBus implements several communication formats that are a subset of the communication formats of I2C.

Specific SMBus protocols require the master to generate a repeated START condition followed by the slave device address without first generating a STOP condition.

The data formats implemented by SMBus are:

- Master transmitter transmits to slave-receiver. The transfer direction in this case is not changed.
- Master reads slave immediately after the first byte. At the moment of the first acknowledgment (provided by the slave receiver) the master-transmitter becomes a master-receiver and the slave-receiver becomes a slave transmitter.
- Combined format. During a change of direction within a transfer, the master repeats both a START condition and the slave address but with the R/_W bit reversed. In this case the master receiver terminates the transfer by generating a NACK on the last byte of the transfer and a STOP condition.

7. Protocol

7.1. Usage Model

The System Management Bus Specification refers to three types of devices. A *slave* is a device that is receiving or responding to a command. A *master* is a device that issues commands, generates the clocks, and terminates the transfer. A *host* is a specialized master that provides the main interface to the system's CPU. There may be at most one host in a system. One 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 may 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. It can also work the other way around as in the case of the host, where a device is mostly a master, but in special cases it might become a slave.

7.2. Device Identification -- Slave Address

Each device that uses the System Management Bus has a 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
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
1100 001	SMBus Device Default Address
0101 000	reserved for ACCESS.bus host
0110 111	reserved for ACCESS.bus default address
1001 0XX	Unrestricted Addresses

All other addresses are reserved for formal assignment by the SMBus address coordinating committee.

The SMBus Alert Response Address (0001100) can be a substitute for device master capability. See Appendix A for details.

The SMBus Device Default Address is reserved for future use by SMBus devices which may allow assignable addresses.

Unrestricted addresses (10010XX) are up for grabs. 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 appendices 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.

7.2.1. SMBus address contention

Several SMBus and I2C devices can be used simultaneously in an actual system. In case of device address contention the designer may use either programmable features implemented in SMBus devices to resolve such contention or/and multiple SMBus branches within the same system to spread devices that use the same address.
There are several type of addresses currently in use into actual SMBus systems.

1. Reserved addresses
   These addresses are reserved either by SMBus or I2C for specific bus functions
2. Assigned addresses
   These addresses are assigned by the SMBus WG to specific type of devices. Each device type that obtains an assigned address has to have an SMBus specification associated with it. Assigned addresses must not be used for any other type of device.
3. Registered addresses
   Manufacturers have in the past and may continue in the future producing SMBus compatible devices for specific system purposes that they do not need a complete SMBus specification or do not require explicit support from the OS. Such devices for example may be port expanders, D/A circuits, etc. The SMBus WG maintains a list of such devices registered to the SMBus committee and it will place effort to coordinate manufacturers in order to avoid address conflicts with devices likely to co-exist on the same bus. Typically these devices provide some address programmability through dedicated device pins, that can be used by the system designer to configure specific system implementations.
4. Programmable addresses
   Version 1.1 of SMBus does not provide a programmable address mechanism. Future versions of the SMBus specification will address this issue.

7.3. Using a Device

A smart SMBus device will have a set of commands by which data can be read and written. All commands are 8 bits (1 byte) long. Command arguments and return values can vary in length. Accessing a command that does not exist or is not supported provokes an error condition. In accordance with the I²C specification, the Most Significant Bit 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 must be able to support all command protocols. The protocols are Quick Command, Send Byte, Receive Byte, Write Byte/Word, Read Byte/Word, Process, Block Read, and Block Write.

Commands may be thought of as register accesses (although devices are not guaranteed that will implement a linear register space).

7.4. Packet Error Checking

Version 1.1 of SMBus introduces a Packet Error Checking mechanism to improve reliability and communication robustness. Implementation of Packet Error Checking by SMBus devices is optional. SMBus devices that implement Packet Error Checking must be capable to communicate with devices that do not implement the Packet Error Checking mechanism.

Pakcet Error Checking, whenever applicable, is implemented by appending a Packet Error Code (PEC) at the end of each message transfer. Each protocol (except for the modified Write, the slave-to-host Write described in a later Section) has two variants: one with the Packet Error Code (PEC) byte and one without. The PEC is a CRC-8 error-checking byte, calculated on all the message bytes (including addresses and read/write bits). The PEC is appended to the message by the device that supplied the last data byte.

7.4.1. Packet Error Checking implementation

The SMBus must accommodate devices that support Packet Error Checking and devices that do not. A device that acts as a slave and supports the PEC must always be prepared to perform the slave transfer with or without a PEC, verify the correctness of the PEC if present, and issue a NAK if the PEC is present but not correct.

Master implementation

A device that acts as a master receiver and supports PEC must first positively identify the capabilities of any device it accesses.

Positive identification of the Slave device capabilities can be achieved through the following method:

1. The Master performs a Read transfer without PEC to the device register containing the device version (whenever applicable).
2. If the device version indicates support of PEC the Master repeats the read operation with PEC.
3. If the PEC received is correct the Master registers the device as PEC compliant.

For positive identification it is recommended for the master to verify more than once the validity of the PEC. It is the responsibility of the master receiver to register the capabilities of the slave transmitter device for all subsequent communications.

A master that has identified a slave as capable of supporting PEC can use Packet Error Checking for subsequent communications with this specific device both when acting as a master-receiver as well as a master transmitter.

Slave implementation

A slave device that implements Packet Error Checking must be prepared to receive and transmit data with or without a PEC. During a slave receive transfer, after the device has identified the protocol and command must accept and check the additional PEC for validity.

During a slave transmit transfer, the slave transmitter must respond to additional clocks after the last byte transfer and furnish a PEC to the master receiver requesting it.

7.4.2. Packet Error Code calculation by CRC-8

Each bus transaction requires a Packet Error Code (PEC) calculation by both the transmitter and receiver within each packet. Use an 8-bit cyclic redundancy check (CRC-8) of the each read or write bus transaction to calculate a Frame Check Sequence (FCS). The FCS is calculated in any way that conforms to a CRC-8 represented by the polynomial, $C(x) = x^8 + x^2 + x^1 + 1$.

Calculating the FCS for transmission or reception is implemented in a method chosen by the device manufacturer. It is possible to perform the check with low-cost hardware, or firmware algorithm that could process the message bit-by-bit or with a byte-wise look-up table. The SMBus web page will show some example CRC-8 methods in the near future.

The FCS is appended to the message as dictated by the protocol in the earlier section. The FCS calculation includes all bytes in the transmission, including address, command and data. The FCS calculation does not include ACK, NAK, START, STOP nor Repeated START bits.

Device implementation of this error method is determined by the specification revision code that is present in SpecificationInfo() command for a Smart Battery, Smart Battery Charger or Smart Battery Selector. See these individual specifications for exact revision coding identities.

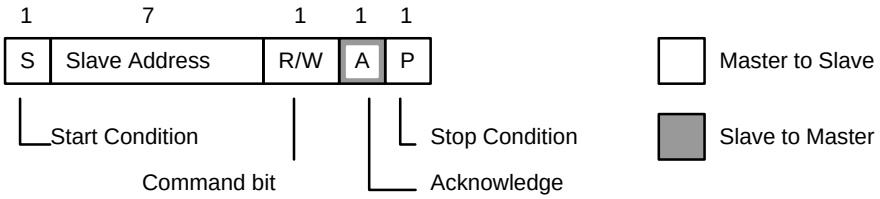
7.5. Bus Protocols

Following is a description of the various SMBus protocols with and without a Packet Error Code.

7.5.1. Quick Command

Here, part of the slave address denotes the command -- the R/W bit. The R/W bit may be used to simply turn a device function on or off, or enable/disable a low-power Standby mode. There is no data sent or received.

The quick command implementation is good for very small devices that have limited support for the SMBus specification. It also limits data on the bus for simple devices.

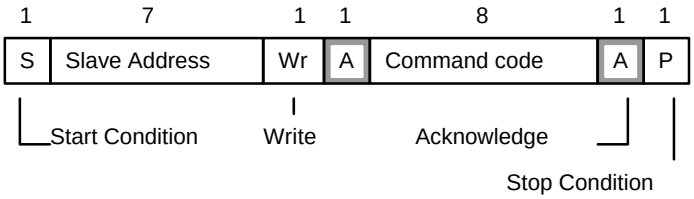


Quick Command Protocol

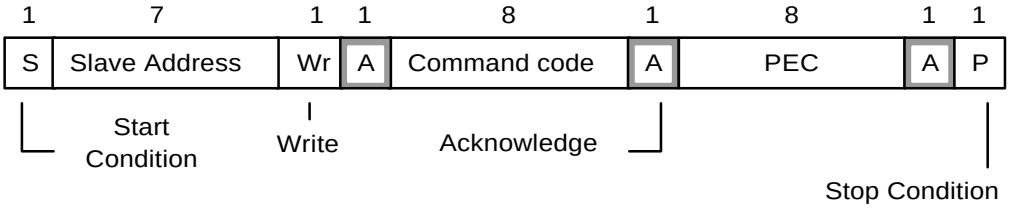
7.5.2. Send Byte

A simple device may recognize its own slave address and accept up to 256 possible encoded commands in the form of a byte that follows the slave address.

All or parts of the Send Byte may contribute to the command. For example, the highest 7 bits of the command code might specify an access to a feature, while the least significant bit would tell the device to turn the feature on or off. Or, a device may set the "volume" of its output based on the value it received from the Send Byte protocol.



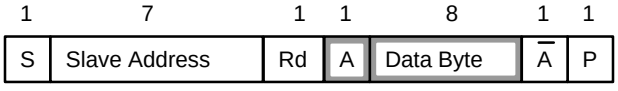
Send Byte Protocol



Send Byte Protocol with PEC

7.5.3. Receive Byte

The Receive Byte is similar to a Send Byte, the only difference being the direction of data transfer. A simple device may have information that the host needs. It can do so with the Receive Byte protocol. The same device may accept both Send Byte and Receive Byte protocols. A "Not ACKnowledge" signifies the end of a read transfer according to the I²C specification.

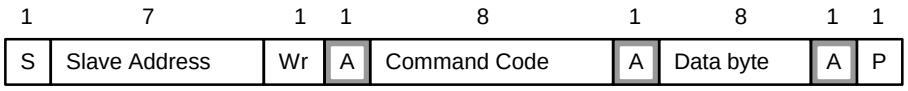


Receive Byte Protocol

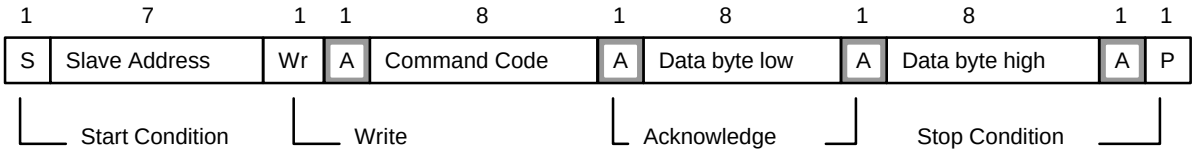
Receive Byte Protocol with PEC

7.5.4. Write Byte/Word

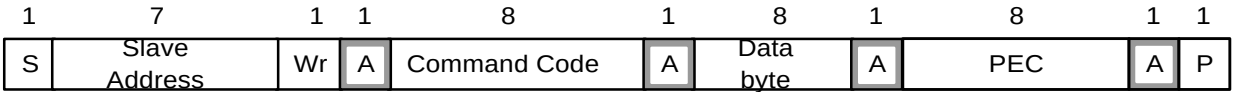
The first byte of a Write Byte/Word access is the command code. The next 1 or 2 bytes are the data to be written. In this example the master asserts the slave device address followed by the write bit. The device acknowledges and the master delivers the command code. The slave again acknowledges before the master sends the data byte or word (low byte first). The slave acknowledges each byte according to the I²C specification, and the entire transaction is finished with a stop condition.



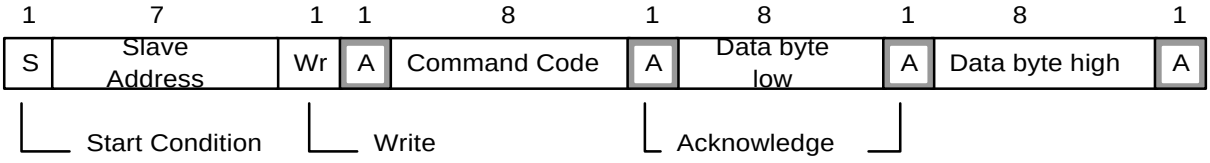
Write Byte Protocol



Write Word Protocol



Write Byte Protocol with PEC

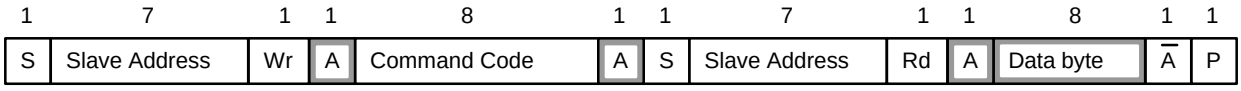


Write Word Protocol with PEC

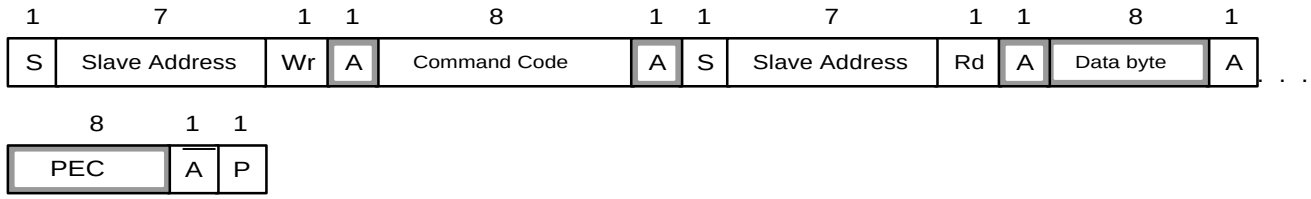
7.5.5. Read Byte/Word

Reading data is slightly more complicated 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 1 or 2 bytes of data.

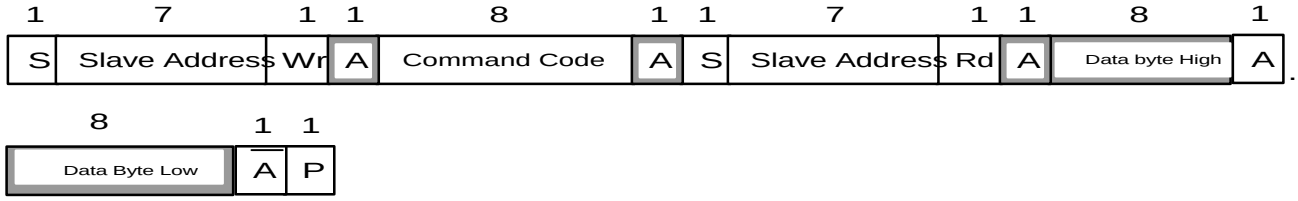
Note that there is no stop condition before the repeated start condition, and that a "Not ACKnowledge" signifies the end of the read transfer.



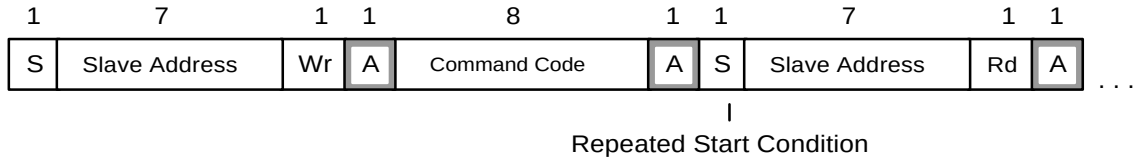
Read Byte Protocol



Read Byte Protocol with PEC



Read Word Protocol

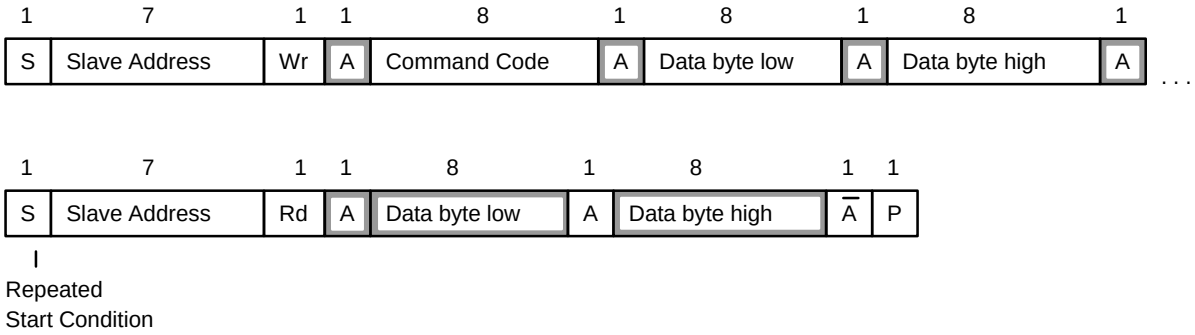


Read Word Protocol with PEC

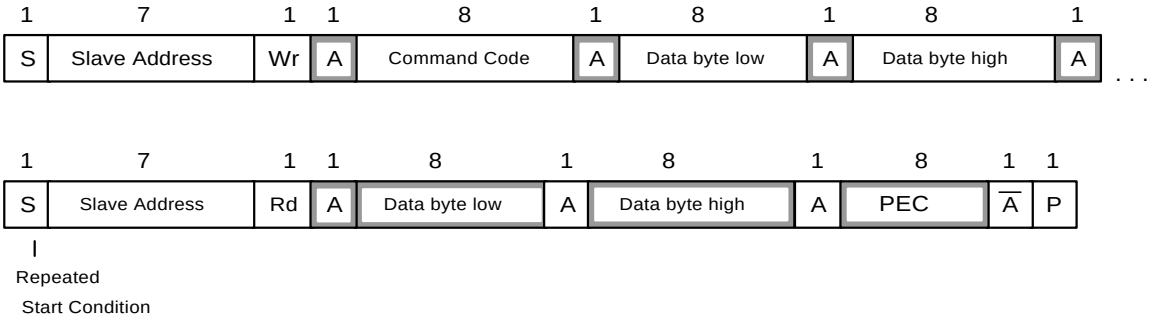
7.5.6. Process Call

The process call is so named because a command sends data and waits for the slave to return a value dependent on that data. The protocol is simply a Write Word followed by a Read Word, but without a second command or stop condition.

The slave can perform any calculations or lookups during the time it takes to transmit the repeated start condition and slave address.



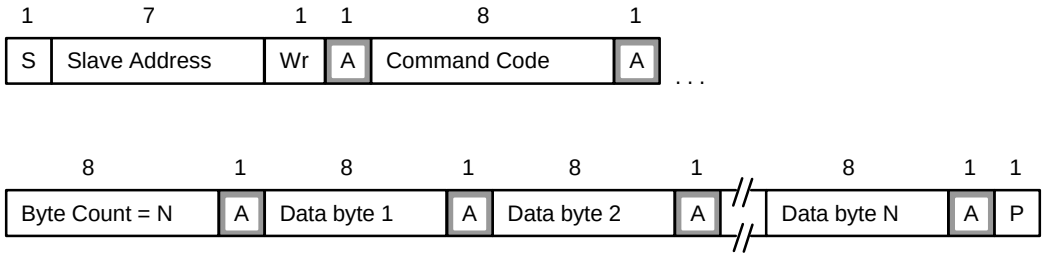
Process Call



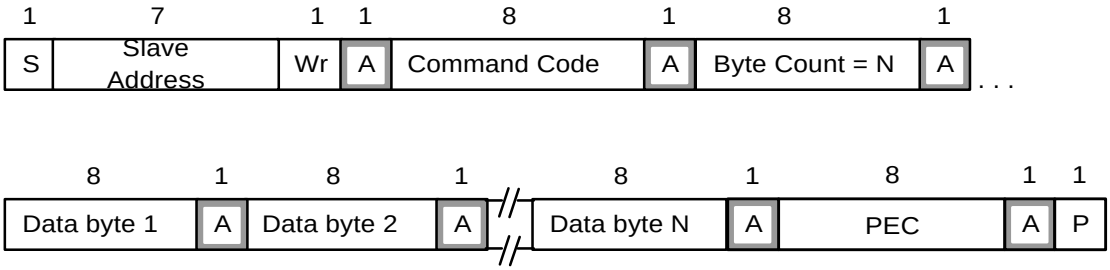
Process Call with PEC

7.5.7. Block Read/Write

The Block Write begins with a slave address and a write condition. After the command code the host issues a byte count which 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 or Write is allowed to transfer a maximum of 32 data bytes.

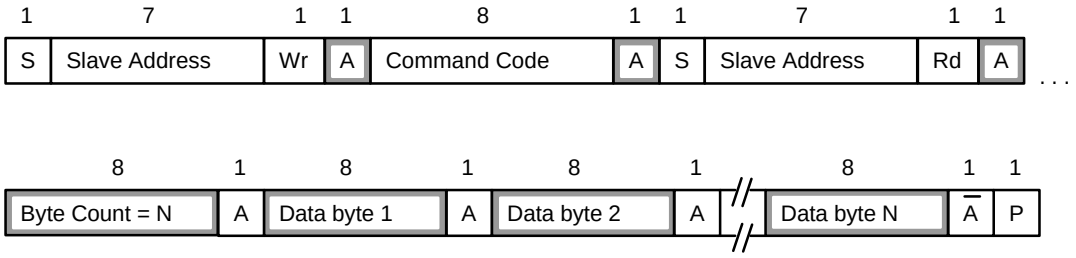


Block Write

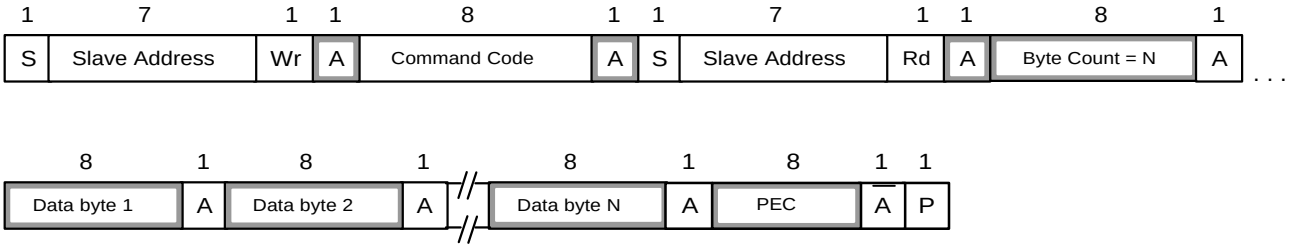


Block Write with PEC

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

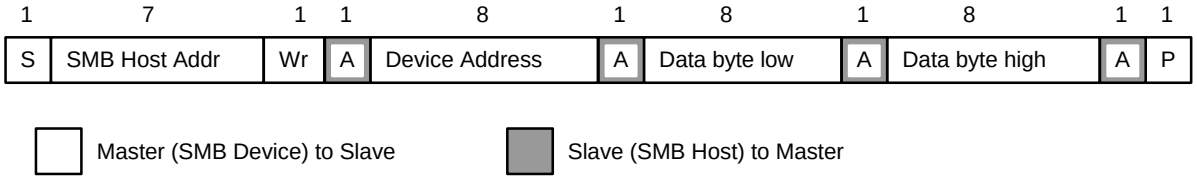


Block Read with PEC

7.6. 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 host's part, only one method of communication is allowed, a modified Write Word. The standard Write Word protocol is modified by replacing the command code with the calling device's address. This protocol is used when an SMBus device becomes a **master** to communicate with the SMBus host acting as a **slave**.

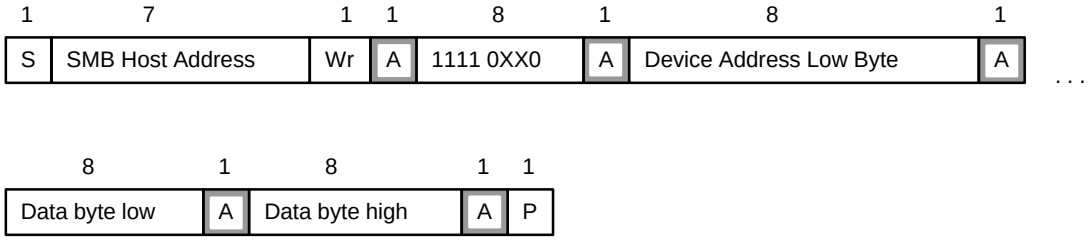
Device to Host communication will begin with the host address. The message's Command Code will actually be the initiating device's address. In the case of 7 bit address, the address bits occupy the 7 most significant bits of the byte. The eighth least significant bit can either be a zero or one. The host now knows the origin of the following 16 bits of device status.



7-bit Addressable Device to Host Communication

The Write Word protocol will be modified slightly for 10-bit addressing. If the device has a 10-bit address, it sends the I²C reserved address for 10-bit addressing (1111 0XX) followed by a 0 to make it 8 bits, the undefined bits being the 2 most significant bits of the 10-bit address. The next byte completes the address. 16 bits of device status follow.

The low byte of the device message precedes the high byte, just as in a Write Word.



10-bit Addressable Device to Host Communication

7.7. Reporting Errors

Any transfer may 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 or cause a timeout to occur thus terminating the transfer.

If the device detects an error, it may signal it to the master by not acknowledging the byte or the in the case of the last byte the whole message. The master can retry to send the message. If the error persists the master can 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.
If the devices interacting support PEC, the PEC code can be used to check for data integrity in a transfer. Upon reception of wrong PEC or NACK during the transmission of the PEC byte, the master must retry the transmission as described previously.
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.

A device may decide to generate an error indication for one or more of the following reasons:

- Device is not ready to process the request for data (either read or write)
- Device does not recognize the command code or function requested
- Device does not permit the command code or function requested
- Overflow or underflow condition
- Incorrect size of data in a block read/write transfer
- Unrecognized or unsupported data transfer protocol used in transaction
- Wrong PEC if implemented
- Any other known or unknown error condition

The error may be generated in order to stop the transaction and indicate that any data already transferred is not reliable.

A device may signal an error by:

- Not ACKnowledge signal at the end of a byte transfer. This method is used when the SMBus device is acting as a slave-receiver and is receiving data from a master-transmitter. The master-transmitter, usually the SMBus Host device, will look for an ACKnowledge bit from the slave after every byte is transmitted

- Hold the SMBCLK line low for longer than $T_{TIMEOUT}$ to cause a device timeout to occur. When a device is acting as a slave-transmitter, the ACKnowledge bit is generated from the master-receiver when the preceding byte has been received correctly. A slave-transmitter may signal an error condition by holding the SMBCLK line low for longer than the $T_{TIMEOUT}$ period. Doing so will cause a timeout condition and the SMBus will then be restored to an idle state (both SMBCLK and SMBDATA returned high.)

In either case, the master device must attempt to generate a Stop Condition on the SMBus to end the transaction.

8. Electrical Characteristics of SMBus devices

The protocol deviates from the original I²C electrical characteristics in the following ways:

8.1. AC Specifications

Symbol	Parameter	Limits		Units	Comments
		Min	Max		
F_{SMB}	SMBus Operating Frequency	10	100	KHz	
T_{BUF}	Bus free time between Stop and Start Condition	4.7		µs	
$T_{HD:STA}$	Hold time after (Repeated) Start Condition. After this period, the first clock is generated.	4.0		µs	
$T_{SU:STA}$	Repeated Start Condition setup time	4.7		µs	
$T_{SU:STO}$	Stop Condition setup time	4.0		µs	
$T_{HD:DAT}$	Data hold time	300		ns	
$T_{SU:DAT}$	Data setup time	250		ns	
$T_{TIMEOUT}$	Clock low time-out	25	35	ms	see note 1
T_{LOW}	Clock low period	4.7		µs	
T_{HIGH}	Clock high period	4.0	50	µs	see note 2
$T_{LOW:SEXT}$	Cumulative clock low extend time (slave device)		25	ms	see note 3
$T_{LOW:MEXT}$	Cumulative clock low extend time (master device)		10	ms	see note 4
T_F	Clock/Data Fall Time		300	ns	See note 5
T_R	Clock/Data Rise Time		1000	ns	See note 5

Note 1: Devices participating in a transfer will timeout when any clock low exceeds the value of $T_{TIMEOUT,MIN}$ of 25 ms. Devices that have detected a timeout condition must reset the communication no later than $T_{TIMEOUT,MAX}$ of 35 ms. The maximum value specified must be adhered to by both a master and a slave as it incorporates the cumulative stretch limit for both a master (10 ms) and a slave (25 ms).

Note 2: T_{HIGH} Max provides a simple guaranteed method for devices to detect bus idle conditions.

Note 3: $T_{LOW:SEXT}$ is the cumulative time a slave device is allowed to extend the clock cycles in one message from the initial start to the stop. If a slave device exceeds this time, it is expected to release both its clock and data lines and reset itself.

Note 4: $T_{LOW:MEXT}$ is the cumulative time a master device is allowed to extend its clock cycles within each byte of a message as defined from start-to-ack, ack-to-ack, or ack-to-stop.

Note 5: Rise and fall time is defined as follows:

- $T_R = (V_{ILMAX} - 0.15)$ to $(V_{IHMIN} + 0.15)$
- $T_F = 0.9V_{DD}$ to $(V_{ILMAX} - 0.15)$

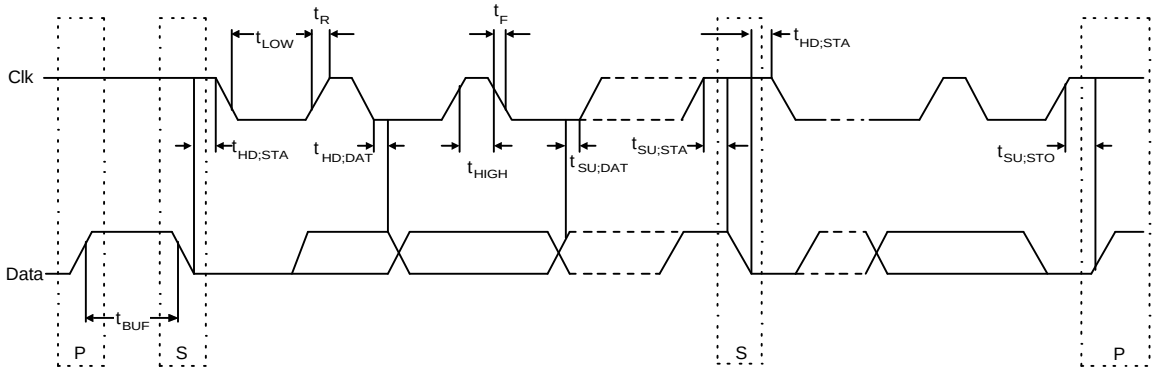


Figure 8.1: Timing Measurements

8.1.1. General timing conditions

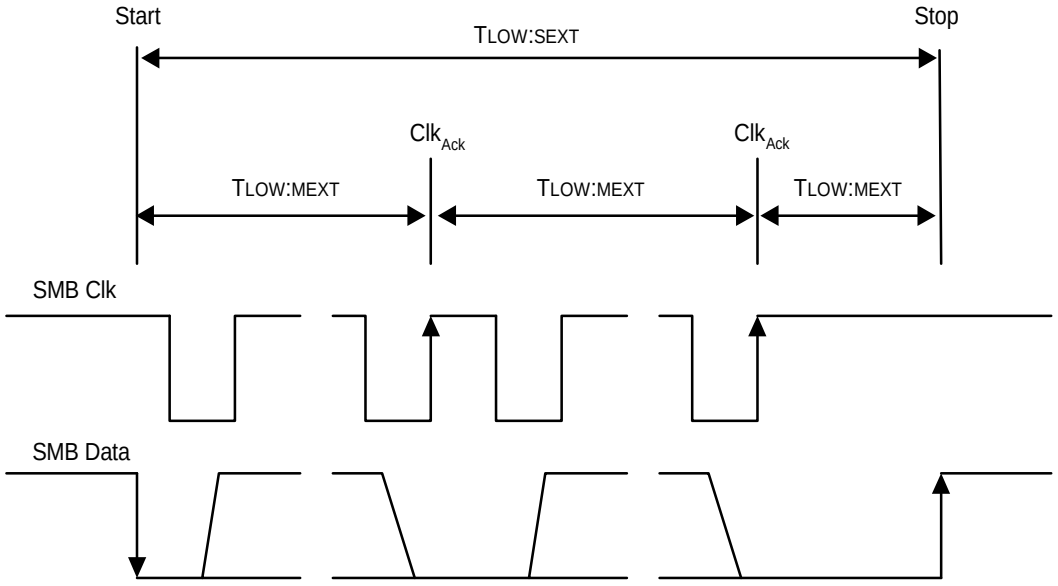
The SMBus is designed to provide a predictable communications link between a system and its devices. However some devices, such as a Smart Battery using a microcontroller to support both bus and maintain battery data, may require more time than might normally be expected. These specifications take such devices into account while maintaining a relatively predictable communications. The following are general comments on the SMBus' timing:

- The bus may be at 0 KHz when idle.
- The FSMB Min is intended to dissuade components from taking too long to complete a transaction.
- An idle bus can be detected by observing that both the clock and data remain high for longer than THIGH Max.
- Every device must be able to recognize and react to a start condition at FSMB Max.

8.1.2. Timeouts

The following diagram illustrates the definition of the timeout intervals, TLOW:SEXT and TLOW:MEXT.

TTIMEOUT Measurement Intervals



8.1.3. Slave device timeout definitions and conditions

A slave device must always timeout when any clock is held low longer than $T_{TIMEOUT}$ maximum.

8.1.4. Master device timeout definitions and conditions

$T_{LOW:MEXT}$ is defined as the cumulative time a master device is allowed to extend its clock cycles within one byte in a message as measured from:

- start to ack
- ack to ack
- ack to stop.

A system host may not violate $T_{LOW:MEXT}$ except while forcing a slave device timeout.

8.2. DC Specifications

8.2.1. Parameters

The System Management Bus is designed to operate over a range of voltages between 3 and 5 Volts +/- 10%.

Symbol	Parameter	Limits		Units	Comments
		Min	Max		
V_{IL}	Data, Clock Input Low Voltage	-0.5	0.8	V	
V_{IH}	Data, Clock Input High Voltage	2.1	5.5	V	
V_{OL}	Data, Clock Output Low Voltage		0.4	V	@ $I_{PULLUP,MAX}$
I_{LEAK}	Input Leakage		±5	µA	
I_{PULLUP}	Current through pullup resistor or current source	100	350	µA	Note 1

Note 1: The $I_{PULLUP,MAX}$ specification is determined primarily by the need to accommodate a maximum of 1.1K Equivalent Series Resistor of removable SMBus devices, such as the Smart Battery, while maintaining the $V_{OL,MAX}$ of the bus. Alternative circuits that can accommodate higher bus capacitances can be implemented as shown on page 5 of this specification.
In cases where a microcontroller is used as the SMBus host, the parameter I_{LEAK} may be exceeded. However, because of the relatively low pullup current, the system designer must ensure that the loading on the bus remains within acceptable limits. Additionally, to prevent bus loading, any components that remain connected to the active bus while unpowered (that is, their V_{CC} lowered to zero), MUST also meet the leakage current specification while unpowered.

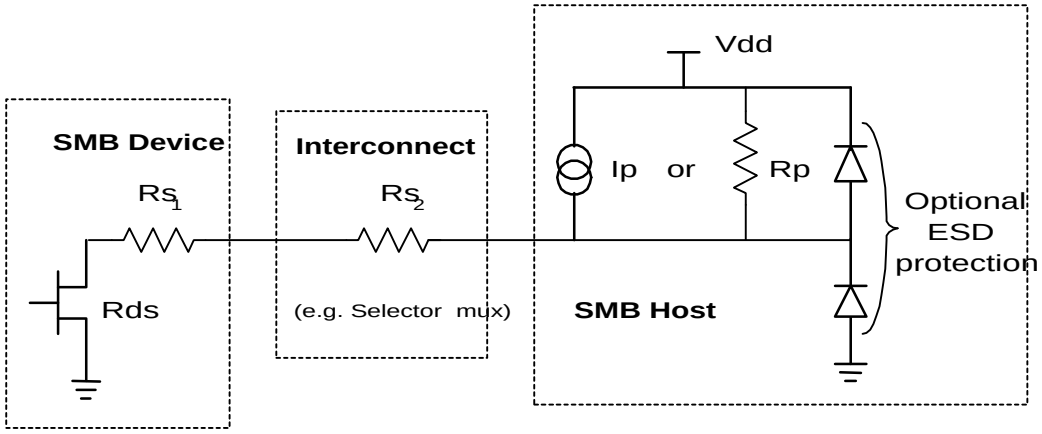
Systems can be designed today using CMOS components, such as microcontrollers. It is the responsibility of the system designer to ensure that all SMBus components comply with the SMBus timing requirements, and are able to operate within the voltage requirements of the specific system.

The I²C bus references its electrical characteristics to V_{DD}. Components attached to SMBus may operate at different voltages. Therefore the SMBus cannot assume that all devices will share a common V_{DD}, hence fixed voltage logic levels.

8.2.2. SMBus branch Circuit model

The following diagram shows the electrical model of an SMBus branch serving Smart Battery System components. A high value series protection resistor can be used for ESD protection of components that can be hot-plugged to the system, such as the Smart Battery. The Equivalent Series Resistor (ESR) of the device and interconnect must not exceed 1.1K in order to maintain the $V_{OL,MAX}$ of the SMBus specification.

SMBus Circuit Model



$$Rds + Rs_1 + Rs_2 < V_{OL\text{-}max} \div I_{pullup\text{-}max} = 1143 \text{ ohms}$$

The value of the pullup resistors (Rp) will vary depending on the system's V_{DD} and the bus's actual capacitance. Current sources (Ip) offer better performance but with increased cost.

The optional diodes, shown in the diagram above, are for ESD protection. They may be necessary in systems where removable SMBus devices such as the Smart Battery are used.

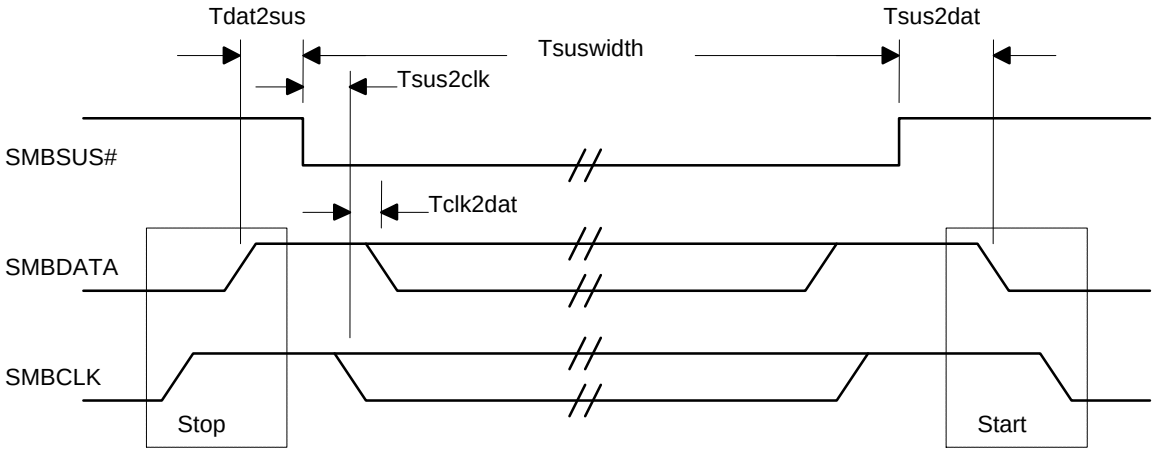
Appendix A: Optional SMBus Signals

SMBSUS#

An **optional** third signal, SMBSUS#, goes low when the system enters the Suspend Mode. Suspend Mode refers to a low-power mode where most devices are stalled or powered down. Upon resume, the SMBSUS# returns high. The system then returns all devices to there operational state.

The SMBSUS# signal is included for clarity and completeness since many of the functions served by the System Management Bus relate to suspend and resume of the system.

The system can use the SMBCLK and SMBDATA lines to program device behavior. During normal operating mode the system may issue configuration commands to different devices. Those commands may tell the device how it is supposed to behave whenever the SMBSUS# line goes active. For example, the system might tell a power plane switcher to turn off power to the hard drive but keep the keyboard controller on when the system goes into suspend mode.



SMBus During Suspend

Timing	Min	Typical
$T_{DAT2SUS}$	0ns	tens of ms
$T_{sus2clk}$	0ns	tens of ns
$T_{clk2dat}$	0ns	0ns
$T_{suswidth}$		minutes, hours, weeks
$T_{sus2dat}$	0ns	hundreds of ms

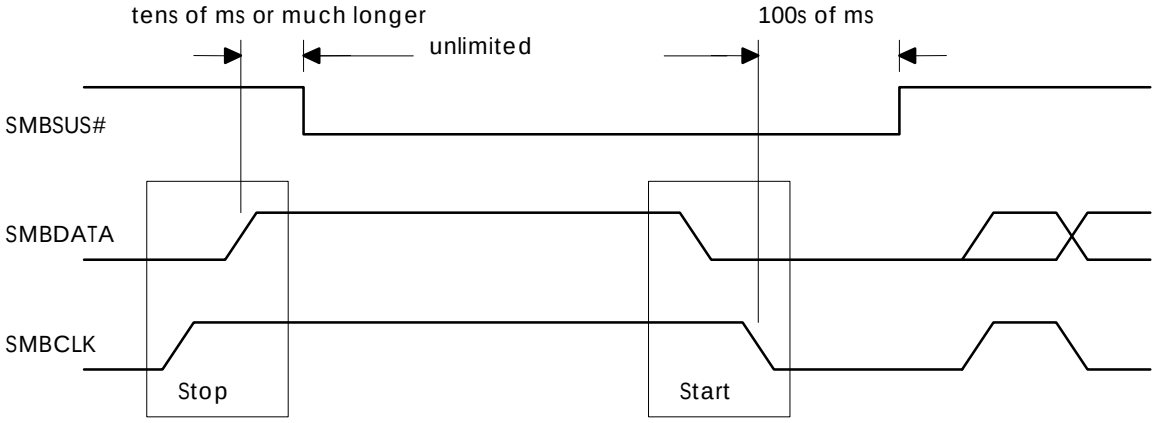
SMBSUS# is not a wired-OR signal. It is an output from the device that controls system management functions, and an input to everything else.

During suspend there is no activity on the System Management Bus unless the SMBus is used to resume from suspend mode. Activity resumes after coming out of suspend.

Anytime after a stop condition the SMBSUS# signal may go active low signifying the system is going into Suspend Mode. This can happen immediately (min = 0ns), but will probably take much longer. In fact, the final SMBus message might terminate minutes or hours before SMBSUS# goes low. Suspend Mode could last a couple of seconds, minutes, hours, or weeks. Before the System Management Bus can send another message the system must come out of Suspend Mode, a process known as Resume. The resume process

probably has to supply voltage to the System Management Bus anyway, although the SMBus may be awake during suspend. The resume process can take a long time, perhaps hundreds of milliseconds. Careful power-down sequencing of the SMBCLK and SMBDATA pullups will prevent devices from falsely seeing a start condition on the bus.

If power is supplied to the System Management Bus during suspend, a device may use it to awaken the system. The host or another device will watch for a start condition on the bus. That device initiates the resume sequence. Communication on the bus resumes when the system is out of suspend.



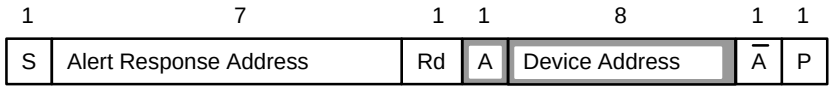
Using SMBus to Resume from Suspend

Since the SMBSUS# is an optional signal, some system devices may not know if the system is in suspend mode or not. Such a device may assume that if both SMBCLK and SMBDATA lines are high that the bus is alive and active. The possibility exists that this device may try to send a critical message to another device which accepts the SMBSUS# signal and is therefore asleep. Therefore it is important that a system is able to resume on a start condition if a non-suspendable master resides on the System Management Bus and that master can send critical messages to suspended devices.

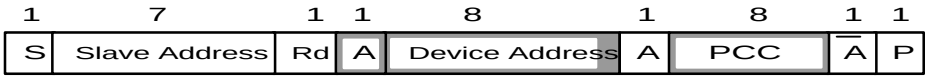
SMBALERT#

Another **optional** signal is an interrupt line for devices that want to trade their ability to master for a pin. SMBALERT# is a wired-AND signal just as the SMBCLK and SMBDATA signals are. SMBALERT# is used in conjunction with the SMBus General Call Address. Messages invoked with the SMBus are 2 bytes long.

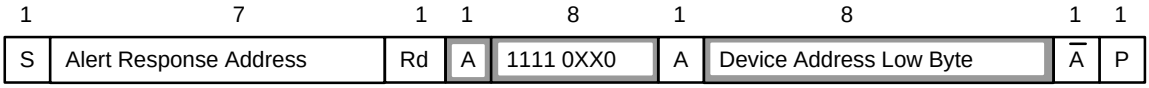
A slave-only device can signal the host through SMBALERT# that it wants to talk. The host processes the interrupt and simultaneously accesses all SMBALERT# devices through the Alert Response Address (ARA). Only the device(s) which pulled SMBALERT# low will acknowledge the Alert Response Address. The host performs a modified Receive Byte operation. The 7 bit device address provided by the slave transmit device is placed in the 7 most significant bits of the byte. The eighth bit can be a zero or one.



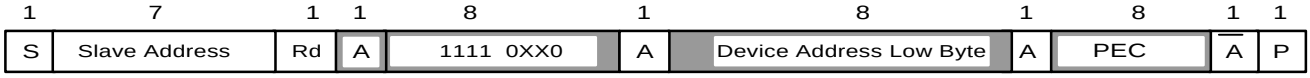
A 7-bit Addressable Device Responds to an ARA



A 7-bit Addressable Device Responds to an ARA with PEC



A 10-bit Addressable Device Responds to an ARA



A 10-bit Addressable Device Responds to an ARA with PEC

If more than one device pulls SMBALERT# low, the highest priority (lowest address) device will win communication rights via standard I²C arbitration during the slave address transfer.

After acknowledging the slave address the device must disengage its SMBALERT# pulldown. If the host still sees SMBALERT# low when the message transfer is complete, it knows to read the ARA again.

A host which does not implement the SMBALERT# signal may periodically access the ARA.

Appendix B

Main Differences Between System Management Bus and I2C

The major differences between I2C and SMBus fall into several categories including electrical, timing, protocols and operating modes.

DC Specifications for SMBus and I2C

Both I2C and SMBus are capable of operating with mixed devices that either have fixed input levels (such as Smart Batteries) or their input levels are related to VDD. When mixing devices, the I2C specification defines the VDD to be 5.0 Volt +/- 10% and the fixed input levels to be 1.5 and 3.0 Volts.

Version 1.1 of SMBus instead of relating the bus input levels to VDD, it defines them to be fixed at 0.8 and 2.1 Volts. This SMBus specification allows for bus implementations with VDD ranging from 3 to 5 Volts +/- 10%. SMBus has relaxed in this version the initial requirement for fixed input levels of 0.6 and 1.4 Volts, in order to reduce the cost of SMBus compliant devices. Devices compliant with the 1.0 specification of SMBus will still operate with a version 1.1 SMBus. In addition they will be ready to operate with 2 Volt SMBus implementations in the future.

A second difference in the DC parameters between I2C and SMBus is in the power consumption of the bus. SMBus was designed to accommodate extremely low power consumption devices, such as the control circuitry within a Smart Battery. These devices have limited current sinking capabilities and a low power consumption bus is essential for maintaining communications without draining the battery of a mobile computer. As a result, SMBus sets more stringent DC requirements than I2C. One of the main differences is the IOL specification for VOL = 0.4 Volts. SMBus devices are required to sink a minimum of 100 uA as opposed to 3mA specified for I2C devices for the same VOL.

A third difference is in the specification of the maximum leakage current for each device connected to the bus. I2C specifies the maximum leakage current to be 10 uA. SMBus version 1.0 specified maximum leakage current of 1 uA. Version 1.1 of the SMBus specification relaxes the leakage requirements to 5 uA, in order to reduce the cost of testing of SMBus devices.

Finally, SMBus does not specify a maximum bus capacitance. Instead it specifies the $I_{PULLDOWN}$ maximum of 350 uA. Bus capacitance can be calculated taking into consideration the maximum rise time and $I_{PULLDOWN}$.

The following table lists the main differences among the DC parameters for I2C and SMBus.

DC parameter comparison between Standard I2C, Fast I2C and SMBus devices								
Symbol	**Parameter**	**Std I2C mode device**		**Fast I2C mode device**		**SMBus device**		**Units**
		MIN	**MAX**	**MIN**	**MAX**	**MIN**	**MAX**	
V_{IL}	Fixed input level	-0.5	1.5	-0.5	1.5	-0.5	0.8	V
	V_{DD} related input level	-0.5	$0.3V_{DD}$	-0.5	$0.3\ V_{DD}$	N/A	N/A	V
V_{IH}	Fixed input level	3.0	V_{DD}max+ 0.5	3.0	V_{DD}max+0 .5	2.1	5.5	V
	V_{DD} related input level	$0.7V_{DD}$	V_{DD}max+ 0.5	$0.7V_{DD}$	V_{DD}max+0 .5	N/A	N/A	V
V_{HYS}	V_{IH}-V_{IL}	N/A	N/A	$0.05V_{DD}$	-	N/A	N/A	V
V_{OL}	V_{OL} @ 3mA	0	0.4	0	0.4	N/A	N/A	V
	V_{OL} @ 6mA	N/A	N/A	0	0.6	N/A	N/A	
	V_{OL} @ 350uA	N/A	N/A	N/A	N/A	-	0.4	
I_{PULLUP}		N/A	N/A	N/A	N/A	100	350	uA
I_{LEAK}		-10	10	-10	10	-5	5	uA

Timing specifications differences of I2C and SMBus

There are differences in the timing specifications between I2C and SMBus. As in the case of DC specification, proper understanding of the parameters is needed in order to combine reliably I2C with SMBus devices.

1. SMBus defines a minimum bus clock frequency F_{SMB} of 10 KHz. I2C does not specify any minimum bus frequency. Besides maintaining effective bus throughput, this SMBus specification parameter can be used as a simple way to detect a bus idle condition (in addition or in lieu of detecting each STOP condition) as well as to implement bit timeout.
2. Maximum clock frequency for SMBus is defined at 100 KHz. I2C provides two modes of operation. The STANDARD MODE up to 100 KHz and the FAST-MODE up to 400 KHz.
3. SMBus defines a clock low time-out, $T_{TIMEOUT}$ of 35 ms. I2C does not specify any timeout limit.
4. SMBus specifies $T_{LOW:SEXT}$ as the cumulutative clock low extend time for a slave device. I2C does not have a similar specification.
5. SMBus specifies $T_{LOW:MEXT}$ as the cumulative clock low extend time for a master device. Again I2C does not have a similar specification.
6. SMBus defines both rise and fall time of bus signals. I2C does not.

The SMBus time-out specifications do not preclude I2C devices co-operating reliably on the SMBus. It is the responsibility of the designer to ensure that I2C devices are not going to violate these bus timing parameters.

Other differences

ACK and NACK usage

There are the following differences in the use of the NACK bus signaling:

- In I2C, a slave receiver is allowed not to acknowledge the slave address, if for example is unable to receive because it's performing some real time task. SMBus requires devices to acknowledge their own address always, as a mechanism to detect a removable device's presence on the bus (battery, docking station, etc.)
- I2C specifies that a slave device, although it may acknowledge its own address, some time later in the transfer it may decide that it cannot receive any more data bytes. The I2C specifies, that the device may indicate this by generating the not acknowledge on the first byte to follow.
  Besides to indicate a slave device busy condition, SMBus is using the NACK mechanism also to indicate the reception of an invalid command or data. Since such a condition may occur on the last byte of the transfer, it is required that SMBus devices have the ability to generate the not acknowledge after the transfer of each byte and before the completion of the transaction. This is important because SMBus does not provide any other resend signaling. This difference in the use of the NACK signaling has implications on the specific implementation of the SMBus port, especially in devices that handle critical system data such as the SMBus host and the SBS components.

SMBus protocols

Each message transaction on SMBus follows the format of one of the defined SMBus protocols. The SMBus protocols are a subset of the data transfer formats defined in the I2C specifications. I2C devices that can be accessed through one of the SMBus protocols are compatible with the SMBus specifications. I2C devices that do not adhere to these protocols cannot be accessed by standard methods as defined in the SMBus and ACPI specifications.

Appendix C: SMBus Device Address Assignments

The following table represents the SMBus device assignments as of February 15, 1995.

Slave Address	Description	Specification
0001 000	SMBus Host	System Management Bus Specification[1] v 1.0 February 1995
0001 001	Smart Battery Charger	Smart Battery Charger Specification[1] v 0.95a February 1995
0001 010	Smart Battery Selector	Smart Battery Selector Specification[1] v 0.9 March 1995
0001 011	Smart Battery	Smart Battery Data Specification[1] v 1.0 February 1995
0001 100	SMBus Alert Response	System Management Bus Specification[1] v 1.0 February 1995
0101 000	ACCESS.bus host	
0101 100	LCD Contrast Controller	TBA
0101 101	CCFL Backlight Driver	TBA
0110 111	ACCESS.bus default address	
1000 0XX	PCMCIA Socket Controllers (4 addresses)	TBA
1000 100	(VGA) Graphics Controller	TBA
1001 0XX	Unrestricted addresses	System Management Bus Specification[1] v 1.0 February 1995
1100 001	SMBus Device Default Address	System Management Bus Specification[1] v 1.0 February 1995

Notes
[1] - Available from the SBS-IF at www.sbs-forum.org

###