System Management Bus (SMBus) Specification

Version 2.0

August 3, 2000

SBS Implementers Forum

Revision No.	Date	Notes
1.0	2/15/95	General Release
1.1	12/11/98	Version 1.1 Release
2.0	8/3/00	Version 2.0 Release

Questions and comments regarding this specification may be forwarded to:
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

1. Introduction

1.1. Overview

The System Management Bus (SMBus) is a two-wire interface through which various system component chips can communicate with each other and with the rest of the system. It is based on the principles of operation of I^2C^*.

SMBus provides a control bus for system and power management related tasks. A system may use SMBus to pass 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.

1.2. Audience

The target audience for this document includes but is not limited to:

- 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 electrical characteristics, network control conventions and communications protocols used by SMBus devices. These can be thought as existing at the first three layers of the seven-layer OSI network model, that is, the physical, data link and network layers. Functions normally implemented at higher layers of the OSI model are beyond the scope of this document.

The original purpose of the SMBus 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, SMBus can also be used to connect a wide variety of devices including power-related devices, system sensors, inventory EEPROMs communications devices and more.

This version of the specification is a superset of previous versions, 1.0 and 1.1. All devices compliant with these previous versions are compliant with this version. Those features new to SMBus with this version of the spec are optional and are appropriate to the new environments enabled by those features. However, if implemented, these new features must be implemented in a manner compliant with this specification.

1.4. Organization of this document

This document is organized to first give the reader an overview of the SMBus and then to delve deeper into its actual working. The major technical discussion appears in three sections that treat the various aspects of the SMBus as they would appear in the first three layers of the OSI reference network model, the physical layer the data link layer and the network layer.

The section on the physical layer sets out SMBus electrical characteristics. The section on the data link layer specifies bit transfers, byte data transfers, arbitration and clock signals. The section on the link layer deals with the general usage model, the concept of addresses in SMBus, the Address Resolution Protocol and the bus data transfer protocol. All aspects of the SMBus proper may be described within the scope of the first three OSI layers.

The SMBus is a multiple attachment bus with no routing capability. Most communication occurs between and involves only two nodes, a master and a slave. Exceptions to this rule occur during and apply to devices that implement the Address Resolution Protocol as well as the Alert Response Address.

Appendixes at the end of this document contain additional information and guides to implementation that the reader may find useful.

1.5. 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.
- *ACPI Specification,* Version 1.0b, Intel Corporation, Microsoft Corporation, Toshiba Corp., February 2, 1999 (http://www.teleport.com/~acpi)
- *PCI Local Bus Specification*, revision 2.2, December 18, 1998, (http://www.pcisig.com)
- *SMBus Control Method Interface Specification*, Version 1.0, Smart Battery System Implementers Forum, December 1999

1.6. Definitions of terms

The following terms are defined with respect to this specification and may have other meanings in other contexts. Some of these terms are used throughout the specification while others have meaning only within limited portions. They are defined here so that the reader may be able to find their definitions in one place.

Address Resolution Protocol	A protocol by which SMBus devices with assignable addresses on the bus are enumerated and assigned non-conflicting slave addresses.
Address Resolved flag (AR)	A flag bit or state internal to a device that indicates whether or not the device's slave address has been resolved by the ARP Master.
Address Valid flag (AV)	A flag bit or state internal to a device that indicates whether or not the device's slave address is valid. This bit must be non-volatile for devices that support the Persistent Slave Address.
ARP	Address Resolution Protocol
SMBus ARP Enumerator	An SMBus master that uses a subset of the ARP for the purpose of discovering ARP-capable slave devices and their assigned slave addresses.
ARP Master	The SMBus master (hardware, software or a combination) responsible for executing the ARP and assigning addresses to ARP-capable slave devices. The SMBus Host will usually be the ARP Master but under some circumstances another SMBus master may assume the role. There is only one active ARP Master at any time.
Assigned Slave Address	The address assigned to a slave device by the ARP Master. This address is then used for accesses to the device's core function. Legal values are in the range 0010 000 to 1111 110 with some exceptions (associated with reserved addresses and those consumed by Fixed Slave Address devices).
Bus Master	Any device that initiates SMBus transactions and drives the clock.
Bus Slave	Target of an SMBus transaction which is driven by some master.
Fixed Slave Address	A slave address that cannot be changed. Non-ARP-capable SMBus devices fall into this category. The ARP Master must not assign a used

	(i.e. device is present on the bus) Fixed Slave Address to an ARP-capable device.
Master-receiver	A bus master in an SMBus transaction while it is receiving data from a bus slave during an SMBus transaction
Master-transmitter	A bus master in an SMBus transaction while it is transmitting data onto the bus during an SMBus transaction.
PEC	Packet Error Code
Persistent Slave Address (PSA)	An assigned slave address that is retained through loss of device power.
Reserved	Reserved fields and bits within any of the data structures in this document and in the SMBus ARP data structures in particular are expected to be unused and ignored by software. Reserved bits must written as 0 and read at 0 unless otherwise specified. These bits and fields are reserved for future use and may not be used by OEMs or IHVs.
Repeated Start	A repeated START is a START condition on the SMBus used to switch from write mode to read mode in a combined format protocol (e.g. Byte Read). The repeated START always follows an Acknowledge, and it always indicates that an address phase is beginning.
Self-powered device	A self-powered device is a device powered either by a battery or an external power source, but not by the system of which it is a part and not in any way by the SMBus.
Slave-receiver	A Slave-receiver is a device that acts as a bus slave in an SMBus transaction while it is receiving address, command or other data from a device acting as a bus master in the transaction.
Slave-transmitter	A Slave-transmitter is a device acting as a bus slave in an SMBus transaction while it is transmitting data on the bus in response to a bus master's request
SMBus Device Default Address	The address all ARP-capable slave devices must respond to. After a slave address has been assigned a device must still respond to commands at the SMBus Device Default Address for ARP management. This address is fixed at 1100 001.
SMBus Host Address	The address to which a host must always respond. This address is used by ARP-capable devices to address the host notify command. This address is fixed at 0001 000.
UDID	Unique Device Identifier. A 128-bit value that a device uses during the ARP process to uniquely identify itself.
Used Address Pool	The list of slave addresses known by the ARP Master that are either: • Reserved • Used by non-ARP-capable devices (Fixed Slave Addresses) • Already assigned to ARP-capable devices The ARP Master must not assign addresses from the first two categories to ARP-capable devices.

1.7. Conventions

Throughout this document, SMBus addresses are given in binary format. SMBus addresses are 7 binary bits long and are conventionally expressed as 4 bits followed by 3 bits followed by the letter 'b', for example, 0001 110b. These addresses occupy the high seven bits of an eight-bit field on the bus. The low bit of this field, however, has other semantic meaning that is not part of an SMBus address.

Diagrams of SMBus transaction data structures as they appear on the bus are of the form:

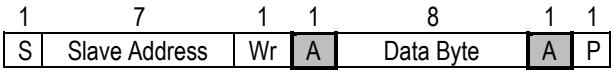


Figure 1-1: generic transaction diagram

In these diagrams, the un-shaded portions are supplied by the bus master of the bus transaction and the shaded portions are driven by the bus slave. The numbers across the top of the transaction diagram indicate the bit widths of each field. In some cases, values will be found below a field in a transaction diagram. When present, this indicates the actual value of the field. The semantics of the various sections of this example transaction are explained in the relevant part of this specification.

The specific SMBus START and STOP conditions, defined in section 4.1.2, are referred to in capital letters.

2. General Characteristics

SMBus is a two-wire bus. Multiple devices, both bus masters and bus slaves, may be connected to an SMBus segment. Generally, a bus master device initiates a bus transfer between it and a single bus slave and provides the clock signals. The one exception to this rule, detailed later, is during initial bus setup when a single master may initiate transactions with multiple slaves simultaneously. A bus slave device can receive data provided by the master or it can provide data to the master.

Only one device may master the bus at any time. Since more than one device may attempt to take control of the bus as a master, SMBus provides an arbitration mechanism that relies on the wired-AND connection of all SMBus device interfaces to the SMBus.

This specification defines two classes of electrical characteristics, low power and high power. The first class, originally defined in the SMBus 1.0 and 1.1 specifications, was designed primarily with Smart Batteries in mind, but could be used with other low-power devices. This version of the specification introduces an alternative higher power set of electrical characteristics. This class is appropriate for use when higher drive capability is required, for example with SMBus devices on PCI add-in cards and for connecting such cards across the PCI connector between each other and to SMBus devices on the system board.

Devices may be powered by the bus V_{DD} or by another power source, $V_{Bus,}$ (as with, for example, Smart Batteries) and will inter-operate as long as they adhere to the SMBus electrical specifications for their class.

The following diagram shows an example implementation of a 5 volt SMBus with devices powered by the bus V_{DD} inter-operating with self-powered devices.

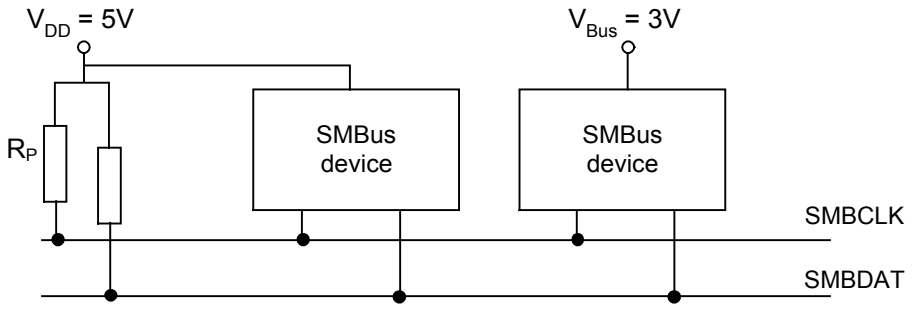


Figure 2-1: SMBus Topology

V_{DD} may be 3 to 5 volts +/- 10% and there may be SMBus devices powered directly by the bus V_{DD}. Both SMBCLK and SMBDAT lines are bi-directional, connected to a positive supply voltage through a pull-up resistor or a current source or other similar circuit. 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, i.e. powered-down devices must provide no leakage path to ground.

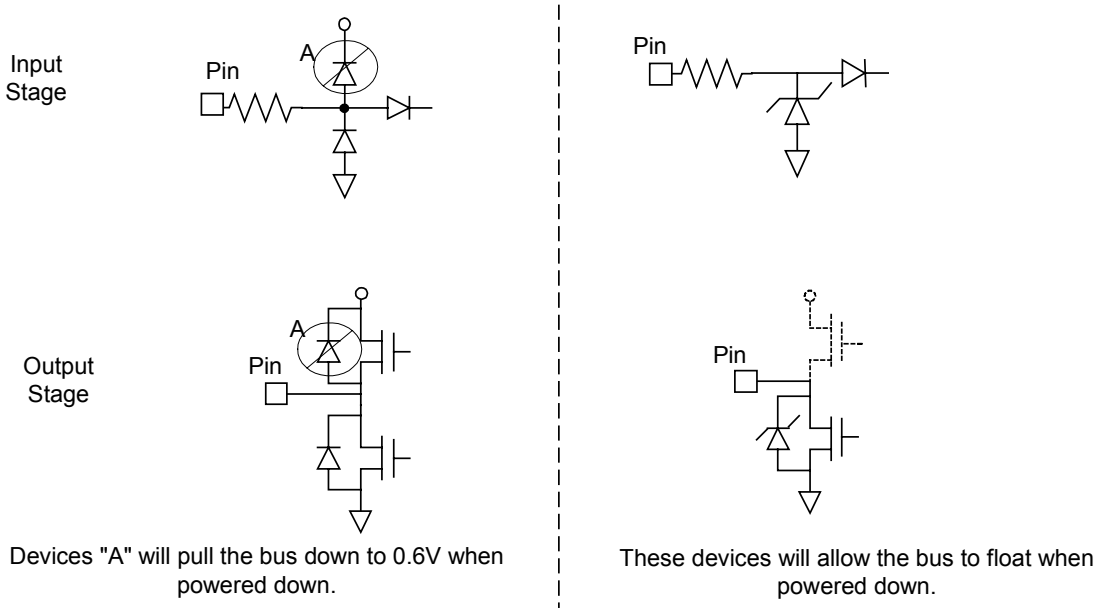


Figure 2-2: Example input and output stages of SMBus devices

A device that wants to place a ‘zero’ on the bus must drive the bus line to the defined logic low voltage level. In order to place a logic ‘one’ on the bus the device should release the bus line letting it be pulled high by the bus pull-up circuitry.

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

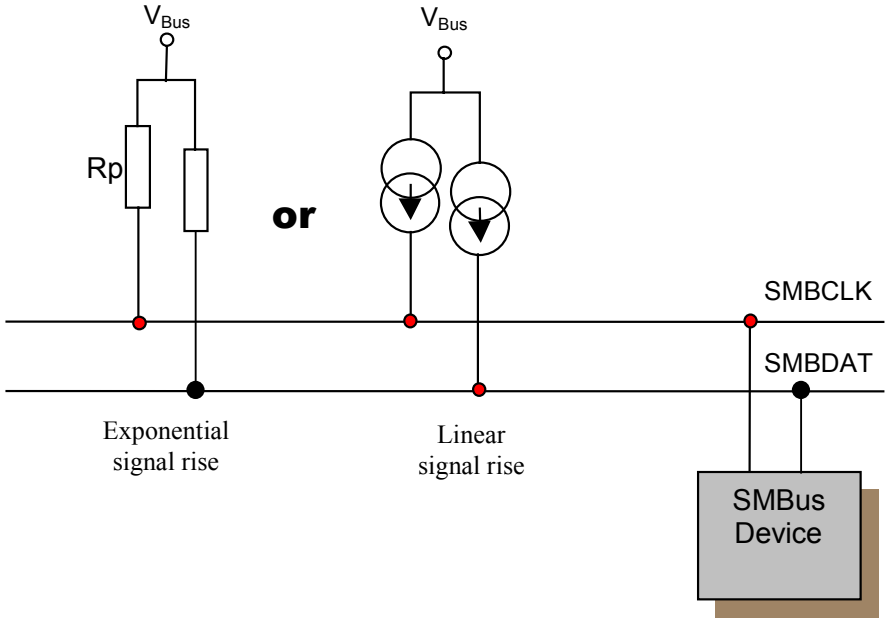


Figure 2-3: SMBus pull-up circuitry

3. Layer 1 – the Physical layer

3.1. Electrical characteristics of SMBus devices – two discrete worlds

SMBus 2.0 is expected to operate in at least two different mutually exclusive environments that have different electrical requirements. In one case, SMBus must operate reliably in the traditional low-power environment of the battery devices that are at its roots. It must also operate reliably in the higher-noise environment of the PCI connector and PCI add-in cards. This specification meets these needs by providing two classes of electrical characteristics as called out below. Most of these specifications deal with voltage levels, noise margins, etc. Of course, many specific items, including general AC specifications, are the same in both environments.

A low-power and a high-power bus may not be directly connected to each other. Low-power devices should not be expected to work on a high-power bus if the device's current sink capability is smaller than 4mA. . However, it is possible to combine a low-power bus segment and a high-power bus segment by connecting the two bus segments through a suitable bridge device.

See Appendix B for ways in which SMBus deviates from I^2C electricals.

3.1.1. SMBus common AC specifications

Both high-power and low-power SMBus electricals share a common set of AC specifications. The diagram below illustrates the various SMBus timings and sets the context for the specifications to follow.

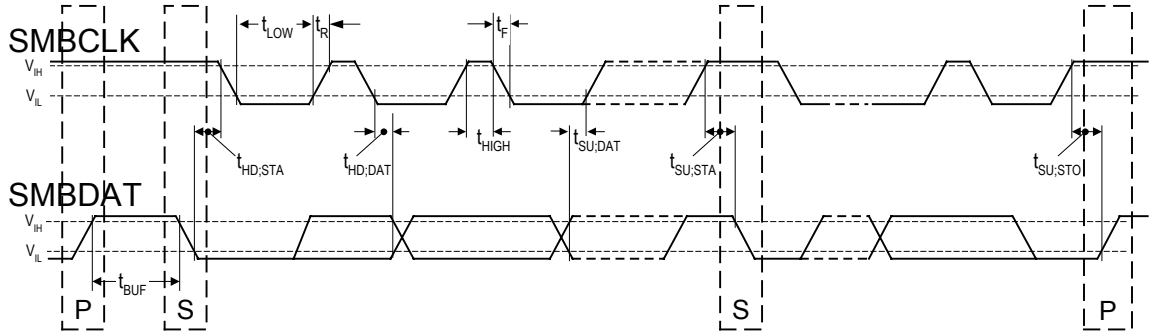


Figure 3-1: SMBus timing measurements

The table below applies to both high and low-power classes of electrical specifications.

Symbol	Parameter	Limits		Units	Comments
		Min	Max		
FSMB	SMBus Operating Frequency	10	100	KHz	See note 1
TBUF	Bus free time between Stop and Start Condition	4.7	-	µs	
THD:STA	Hold time after (Repeated) Start Condition. After this period, the first clock is generated.	4.0	-	µs	
TSU:STA	Repeated Start Condition setup time	4.7	-	µs	
TSU:STO	Stop Condition setup time	4.0	-	µs	
THD:DAT	Data hold time	300	-	ns	
TSU:DAT	Data setup time	250	-	ns	
TTIMEOUT	Detect clock low timeout	25	35	ms	See note 2
TLOW	Clock low period	4.7	-	µs	
THIGH	Clock high period	4.0	50	µs	See note 3

Symbol	Parameter	Limits		Units	Comments
		Min	Max		
$T_{LOW:SEXT}$	Cumulative clock low extend time (slave device)	-	25	ms	See note 4
$T_{LOW:MEXT}$	Cumulative clock low extend time (master device)	-	10	ms	See note 5
T_F	Clock/Data Fall Time	-	300	ns	See note 6
T_R	Clock/Data Rise Time	-	1000	ns	See note 6
T_{POR}	Time in which a device must be operational after power-on reset		500	ms	See section 3.1.4.2

Table 1: SMBus AC specifications

Note 1: The minimum frequency for synchronizing device clocks is defined in section 4.3.3. A master shall not drive the clock at a frequency below the minimum FSMB. Further, the operating clock frequency shall not be reduced below the minimum value of F_{SMB} due to periodic clock extending by slave devices as defined in section 4.3.3. This limit does not apply to the bus idle condition, and this limit is independent from the $T_{LOW:SEXT}$ and $T_{LOW:MEXT}$ limits.

For example, if the SMBCLK is high for $T_{HIGH,MAX}$, the clock must not be periodically stretched longer than $1/F_{SMB,MIN} - T_{HIGH,MAX}$. This requirement does not pertain to a device that extends the SMBCLK low for data processing of a received byte, data buffering and so forth for longer than 100us in a non-periodic way.

Note 2: Devices participating in a transfer can abort the transfer in progress and release the bus when any single clock low interval exceeds the value of $T_{TIMEOUT,MIN}$. After the master in a transaction detects this condition, it must generate a stop condition within or after the current data byte in the transfer process. Devices that have detected this condition must reset their communication and be able to receive a new START condition no later than $T_{TIMEOUT,MAX}$. Typical device examples include the host controller, and embedded controller and most devices that can master the SMBus. Some simple devices do not contain a clock low drive circuit; this simple kind of device typically may reset its communications port after a start or a stop condition.

A timeout condition can only be ensured if the device that is forcing the timeout holds the SMBCLK low for $T_{TIMEOUT,MAX}$ or longer.

Note 3: $T_{HIGH,MAX}$ provides a simple guaranteed method for masters to detect bus idle conditions. A master can assume that the bus is free if it detects that the clock and data signals have been high for greater than $T_{HIGH,MAX}$.

Note 4: $T_{LOW:SEXT}$ is the cumulative time a given slave device is allowed to extend the clock cycles in one message from the initial START to the STOP. It is possible that, another slave device or the master will also extend the clock causing the combined clock low extend time to be greater than $T_{LOW:SEXT}$. Therefore, this parameter is measured with the slave device as the sole target of a full-speed master.

Note 5: $T_{LOW:MEXT}$ 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. It is possible that a slave device or another master will also extend the clock causing the combined clock low time to be greater than $T_{LOW:MEXT}$ on a given byte. Therefore, this parameter is measured with a full speed slave device as the sole target of the master.

Note 6: Rise and fall time is defined as follows:

$$T_R = (V_{ILMAX} - 0.15) \text{ to } (V_{IHMIN} + 0.15)$$

$$T_F = (V_{IHMIN} + 0.15) \text{to } (V_{ILMAX} - 0.15)$$

3.1.1.1. 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 bus transactions and to maintain battery data, may require more time than might normally be expected. These specifications take such devices into account while maintaining relatively predictable communications. The following are general comments on the SMBus' timing:

- The bus is at 0 KHz when idle. The $F_{SMB,MIN}$ specification is intended to dissuade components from taking too long to complete a transaction.
- Every device must be able to recognize and react to a START condition at the fastest timings in Table 1: SMBus AC specifications, above.

3.1.1.2. Timeouts

Figure 3-2: Timeout measurement intervals illustrates the definition of the timeout intervals, $T_{LOW:SEXT}$ and $T_{LOW:MEXT}$.

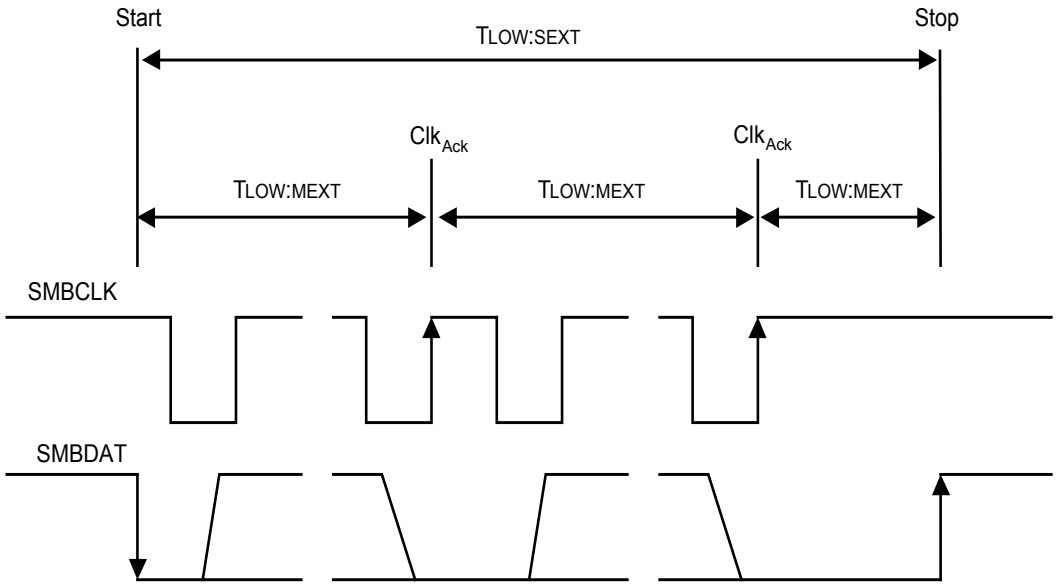


Figure 3-2: Timeout measurement intervals

3.1.1.3. Slave device timeout definitions and conditions

The $T_{TIMEOUT,MIN}$ parameter allows a master or slave to conclude that a defective device is holding the clock low indefinitely or a master is intentionally trying to drive devices off the bus. It is highly recommended that a slave device release the bus (stop driving the bus and let SMBCLK and SMBDAT float high) when it detects any single clock held low longer than $T_{TIMEOUT,MIN}$. Devices that have detected this condition must reset their communication and be able to receive a new START condition in no later than $T_{TIMEOUT,MAX}$.

Slave devices that violate $T_{LOW:SEXT}$ are not conformant with this specification. A Master is allowed to abort the transaction in progress to any slave that violates the $T_{LOW:SEXT}$ or $T_{TIMEOUT,MIN}$ specifications

3.1.1.4. 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 when caused by the combination of its clock extension with the clock extension from a slave device or another master.

A Master is allowed to abort the transaction in progress to any slave that violates the $T_{LOW:SEXT}$ or $T_{TIMEOUT,MIN}$ specifications. This can be accomplished by the Master issuing a STOP condition at the conclusion of the byte transfer in progress.[1]

3.1.2. Low-power DC specifications

3.1.2.1. DC parameters

The System Management Bus is designed to operate over a range of voltages between 3 and 5 Volts +/- 10% (2.7 V to 5.5 V).

Symbol	Parameter	Limits		Units	Comments
		Min	Max		
V_{IL}	Data, Clock Input Low Voltage	-	0.8	V	
V_{IH}	Data, Clock Input High Voltage	2.1	VDD	V	
V_{OL}	Data, Clock Output Low Voltage	-	0.4	V	at I_{PULLUP}, Max
I_{LEAK}	Input Leakage	-	±5	µA	Note 1
I_{PULLUP}	Current through pull-up resistor or current source	100	350	µA	Note 2
VDD	Nominal bus voltage	2.7	5.5	V	3V to 5V ±10%

Table 2: Low power SMBus DC specification

Note 1: Devices must meet this specification whether powered or unpowered. However, a microcontroller acting as an SMBus host may exceed I_{LEAK} by no more than 10 µA.

Note 2: 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.

Because of the relatively low pull-up current, the system designer must ensure that the loading on the bus remains within acceptable limits. Additionally, to prevent bus loading, any devices that remain connected to the active bus while unpowered (that is, their Vcc lowered to zero), must also meet the leakage current specification.

[1] A Master should take care when evaluating $T_{LOW:SEXT}$ violation during arbitration since the clock may be held low by multiple slave devices simultaneously. The arbitration interval may be extended for several bytes in the case of devices that respond to commands to the SMBus ARP address. If timeouts are handled at the driver level, the software may need to allow timeouts to be configured or disabled by applications that use the driver in order to support older devices that do not fully meet the SMBus timeout specifications. Devices that implement 'shared' slave addresses may also violate this specification due to combined clock stretching by the different devices sharing the address. $T_{TIMEOUT,MIN}$, however, does not increase due to combined clock stretching. Therefore, this is a safer timeout parameter for a Master to use when it knows it"s accessing SMBus 2.0 devices.

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.

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.

3.1.2.2. SMBus branch circuit model

The following diagram shows the electrical model of the SMBus. A series protection resistor can be used for ESD protection of components that can be hot-plugged to the system, such asa 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 low-power specification. This circuit model is equally valid for high-power components discussed in the next section. Due to significantly different bus loading, individual component values will change.

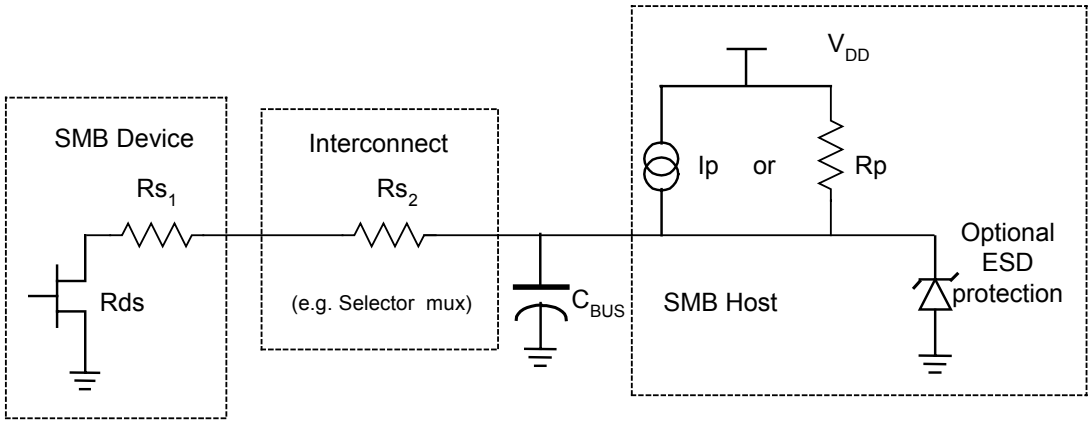


Figure 3-3: SMBus circuit model

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

The optional diode shown in the diagram above is for ESD protection. It may be necessary in systems where a removable SMBus device such as a Smart Battery is used. However, circuits as actually implemented must comply with the previously stated unpowered leakage current specification.

3.1.3. High-Power DC specifications

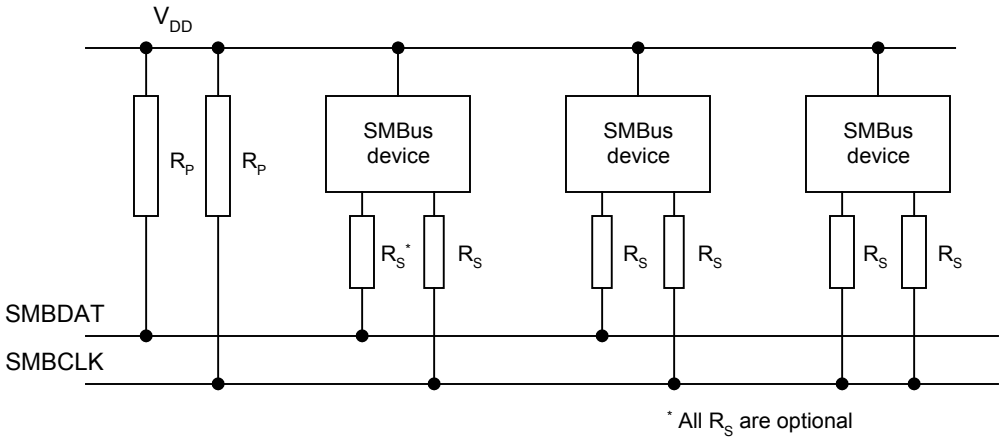


Figure 3-4: SMBus branch with multiple devices attached

High-power SMBus is specified below. These higher power specifications provide the robustness necessary, for example, to allow SMBus to cross the PCI connector, thus allowing SMBus devices on PCI add-in cards to communicate with other devices on both the system board and other PCI add-in cards in the same system. These higher power electrical specifications are an alternative to the lower power specifications stated above and may be used in environments where necessary. Some parameters are explained further in the sections below.

Symbol	Parameter	Limits		Units	Comments
		Min	Max		
V_{IL}	SMBus signal Input low voltage	-	0.8	V	
V_{IH}	SMBus signal Input high voltage	2.1	V_{DD}	V	
V_{OL}	SMBus signal Output low voltage	-	0.4	V	@ I_{PULLUP}
$I_{LEAK-BUS}$	Input Leakage per bus segment		±200	uA	
$I_{LEAK-PIN}$	Input Leakage per device pin		±10	uA	
VDD	Nominal bus voltage	2.7	5.5	V	3V to 5V ±10%
I_{PULLUP}	Current sinking, VOL = 0.4V	4		mA	
C_{BUS}	Capacitive load per bus segment		400	pF	
CI	Capacitance for SMBDAT or SMBCLK pin		10	pF	Recommended
V_{NOISE}	Signal noise immunity from 10 MHz to 100 MHz	300	-	mV p-p	This AC item applies to the high-power DC specification only

Table 3: High-power SMBus DC specification

3.1.3.1. Capacitive load of high-power SMBus lines

Capacitive load for each bus line includes all pin, wire and connector capacitances. The maximum capacitive load affects the selection of the R_P pull-up resistor or the current source in order to meet the rise time specifications of SMBus.

Pin capacitance (C_I) is defined as the total capacitive load of one SMBus device as seen in a typical manufacturer's data sheet. The value is a recommended guideline so that two such devices may, for example, be populated on an add-in card.

3.1.3.2. Minimum current sinking requirements for SMBus devices

While SMBus devices used in low-power segments have practically no minimum current sinking requirements due to the low pull-up current specified for low-power segments, devices in high-power segments are required to sink a minimum current of 4 mA while maintaining the $V_{OL,MAX}$ of 0.4 Volts. The requirement for 4 mA sink current determines the minimum value of the pull-up resistor R_P that can be used in SMBus systems.

3.1.4 Additional common low and high-power specifications

3.1.4.1 SMBus 'back powering' considerations

Unpowered devices connected to either a low-power or high-power SMBus segment must provide, either within the device or through the interface circuitry, protection against "back powering" the SMBus. Unpowered devices connected to high-power segments must meet leakage specifications in section 3.1.2.1.

3.1.4.2 Power-on reset

SMBus devices detect a power-on event in one of three ways:

- By detecting that power is being applied to the device,
- By an external reset signal that is being asserted or
- For self-powered or always powered devices, by detecting that the SMBus is active (clock and data lines have gone high after being low for more than 2 1/2 seconds).

An SMBus device must respond to a power-on event by bringing the device into an operational state within T_{POR}, defined in Table 1, after the device has been supplied power that is within the device's normal operating range. Self-powered or always-powered devices, such as Smart Batteries, are not required to do a complete power-on reset but must be in an operational state within 500 milliseconds after the SMBus becomes active.

4. Layer 2 – the Data Link layer

4.1. Bit transfers

SMBus uses fixed voltage levels to define the logic "ZERO" and logic "ONE" on the bus respectively.

4.1.1. Data validity

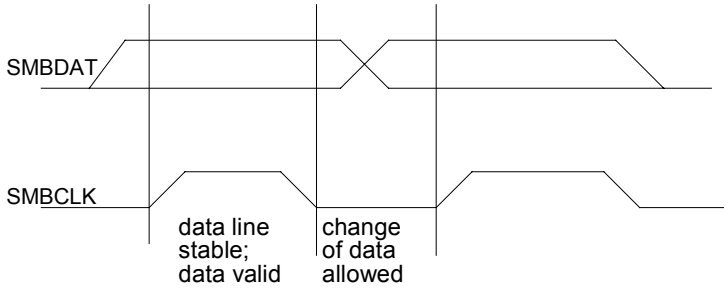


Figure 4-1: Data validity

The data on SMBDAT must be stable during the "HIGH" period of the clock. Data can change state only when SMBCLK is low. Figure 4-1 illustrates the relationships. See figure 3-1 and table 1 for actual specifications.

4.1.2. START and STOP conditions

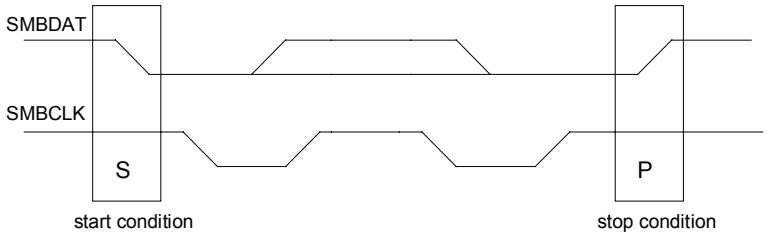


Figure 4-2: START and STOP conditions

Two unique bus situations define a message START and STOP condition.

1. A HIGH to LOW transition of the SMBDAT line while SMBCLK is HIGH indicates a message START condition.
2. A LOW to HIGH transition of the SMBDAT line while SMBCLK 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 idle again after certain time following a STOP condition or after both the SMBCLK and SMBDAT lines remain high for more than $T_{HIGH:MAX}$.

4.1.3. Bus idle condition

Bus idle is the conditon during which the SMBCLK and SMBDAT lines are both high, without any state transitions, for a time period specified as the minimum of the following:

- T_{BUF} (4.7 µS) from the last detected STOP condition, or
- $T_{HIGH:MAX}$ (50 µS)

The latter timing parameter covers the condition where a master has been dynamically added to the bus and may not have detected a state transition on the SMBCLK or SMBDAT lines. In this case, the master must wait long enough to ensure that a transfer is not currently in progress.

4.2. Data transfers on SMBus

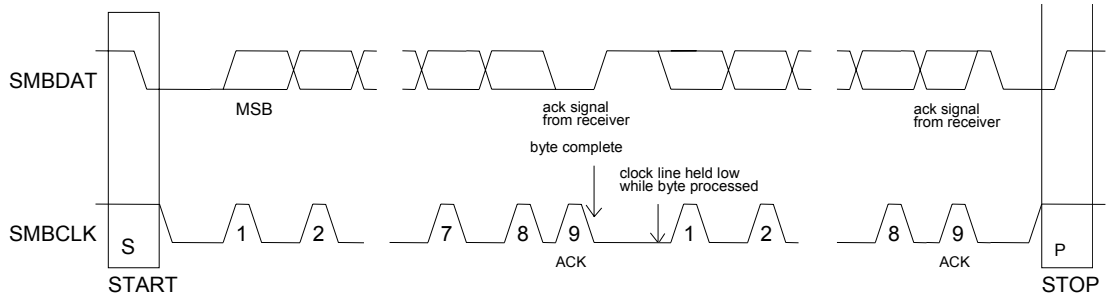


Figure 4-3: SMBus byte format

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.

The diagram below, figure 4-4 illustrates the positioning of acknowledge (ACK) and not acknowledge (NACK) pulses relative to other data

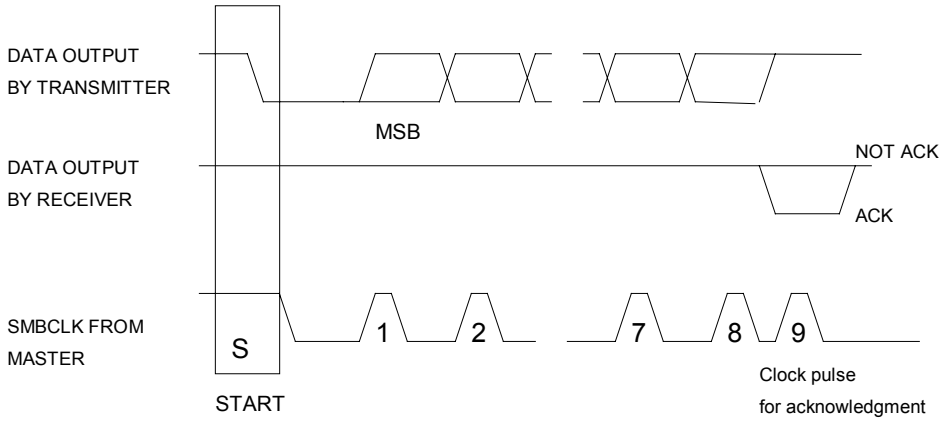


Figure 4-4: ACK and NACK signaling of SMBus

The acknowledge-related clock pulse is generated by the master. The transmitter, master or slave, releases the SMBDAT line (HIGH) during the acknowledge clock cycle. In order to acknowledge a byte, the receiver must pull the SMBDAT line LOW during the HIGH period of the clock pulse according to the SMBus timing specifications. A receiver that wishes to NACK a byte must let the SMBDAT line remain HIGH during the acknowledge clock pulse.

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

An SMBus slave device may decide to NACK a byte other than the address 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 NACK 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 generating an NACK 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.

The latter mechanism is one way for a device to detect whether a slave transmitter implements Packet Error Checking. See section 5.4 for more information on Packet Error Checking.

4.3. Clock generation and arbitration

4.3.1. Synchronization

A situation may occur in which 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 SMBCLK line will cause all devices involved to start counting off their LOW period and start driving SMBCLK low if the device is a master. As soon as a device finishes counting its LOW period it will release the SMBCLK line. Nevertheless, the actual signal on the SMBCLK may not transition to the HIGH state if another master with longer LOW period keeps the SMBCLK line LOW. In this situation the master that released the SMBCLK line will enter the SMBCLK HIGH wait period. When all devices have counted off their LOW period, the SMBCLK 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 SMBCLK line LOW and the cycle will start again.

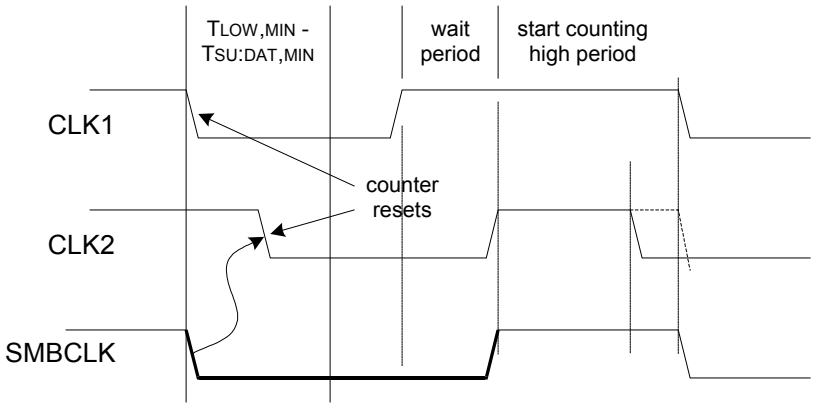


Figure 4-5: SMBus clock synchronization

In Figure 4-5, the interval between the first high-to-low transition of CLK1 and CLK2 must be less than ($T_{LOW:MIN} - T_{SU:DAT}$). This way, a synchronized clock is provided for all devices, where the SMBCLK

LOW period is determined by the slowest device and the SMBCLK HIGH period is determined by the fastest device.

4.3.2. Arbitration

A master may start a transfer only if the bus is idle. One 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 SMBDAT line while the SMBCLK is HIGH. A master that transmits a HIGH level, while the other(s) master is transmitting a LOW level on the SMBDAT line loses control of the bus in the arbitration cycle.

The master that lost the arbitration may continue to provide clock pulses until the completion of the byte on which it 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. In the event that two masters are arbitrating and the first master wants to put a repeated START on the bus while the second master wants to put a ZERO data bit on the bus, the first master must recognize that it cannot cause the start and lose arbitration. If the first master wants to put a repeated START on the bus while the second master wants to put a ONE data bit on the bus, the second master will see the repeated start and lose arbitration. If both masters put a repeated START on the bus in the same bit position, arbitration should continue at each data bit.

This mechanism requires that SMBus master devices participating in an arbitration cycle monitor the actual state of the SMBDAT line during arbitration.

If a master also incorporates a slave function and loses control of the bus in the arbitration cycle during the address stage, it must 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.

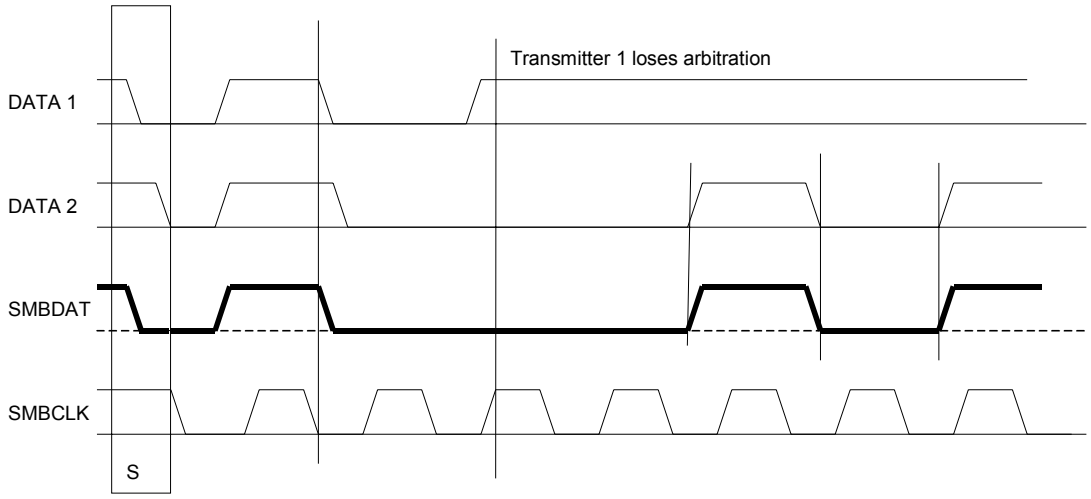


Figure 4-6: SMBus arbitration

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 that it didn't generate must quit the transfer.

Once a master has won arbitration, arbitration is disallowed until the bus is again idle.

4.3.3. Clock low extending

SMBus provides a clock synchronization mechanism to allow 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.

At the bit level, a device may slow down the bus by periodically extending the clock low interval.

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 must maintain the $f_{SMB,MIN}$ frequency of 10 KHz ($f_{SMB,MIN}^{-1}$= 100µs) in order to preserve the SMBus bandwidth.

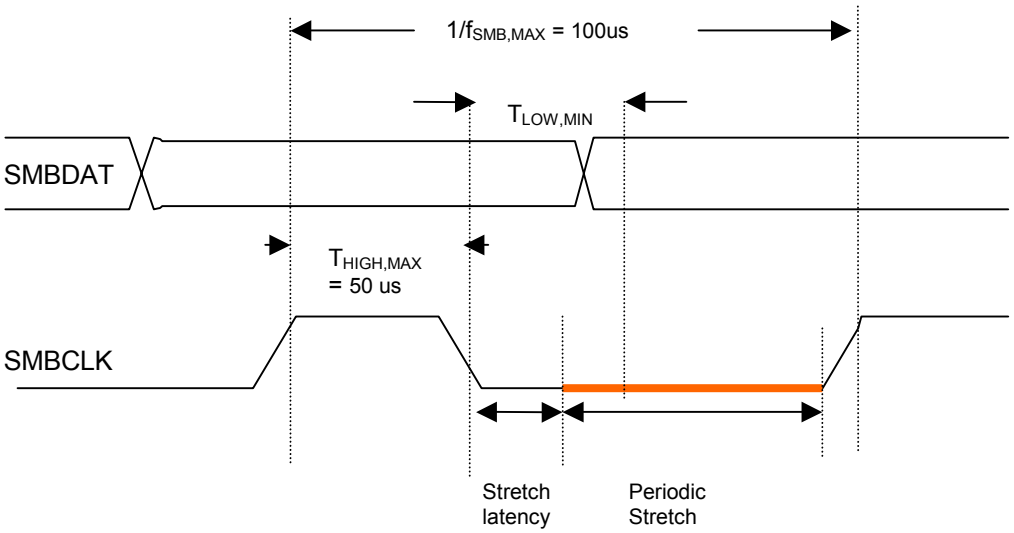


Figure 4-7: Periodic clock stretching by a slave SMBus device

Clock LOW extension, or stretching , if necessary, must start after the SMBCLK high-to-low transition within a $T_{LOW:MIN}$ – $T_{SU:DAT}$ interval. Devices designed to stretch the clock on every bit transfer must maintain the minimum bus frequency $f_{SMB,MIN}$ of 10 KHz. A slave device may opt to stretch 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 any 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.

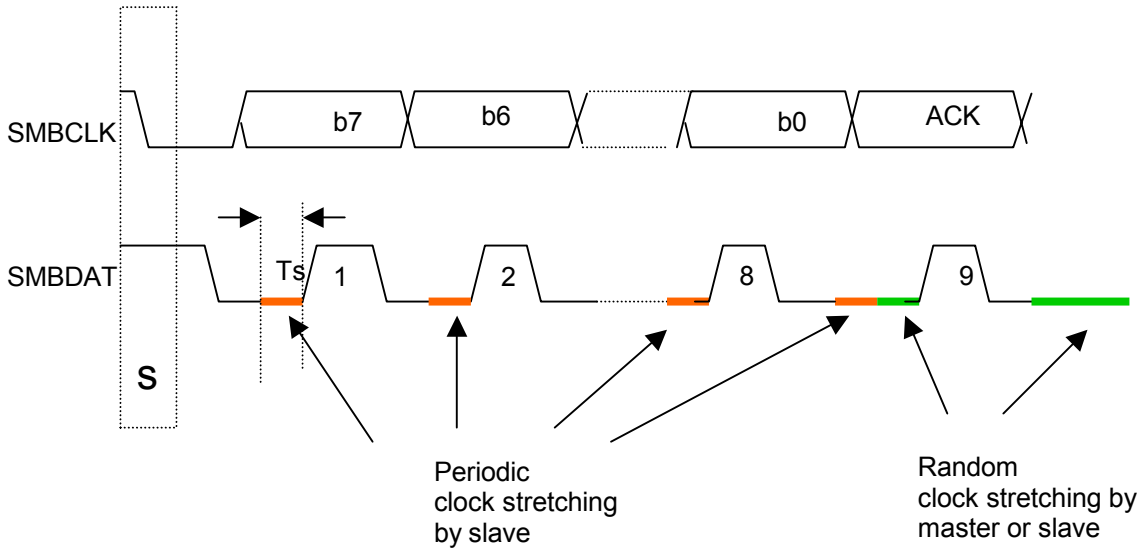


Figure 4-8: Periodic and random clock stretching

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.

4.4. Data transfer formats

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

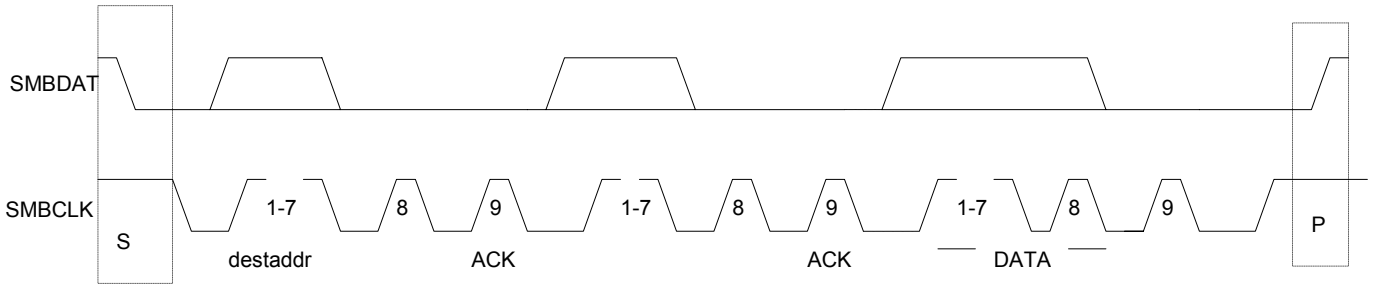


Figure 4-9: Data transfer over SMBus

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.

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

5. Layer 3 – Network layer

5.1. 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. A host must be a master-slave and must support the SMBus host notify protocol. 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 and always a slave. A device may act as a slave most of the time, but in special instances it may become a master. A device can also be designed to be a master only. A system host is an example of a device that acts as a host most of the time but that includes some slave behavior..

5.2. Device identification – slave address

Any device that exists on the System Management Bus as a slave has a unique address called the *Slave Address*. For reference, the following addresses are reserved and must not be used by or assigned to any SMBus slave device unless otherwise detailed by this specification.

Slave Address Bits 7-1	R/W# bit Bit 0	Comment
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
0101 000	X	Reserved for ACCESS.bus host
0110 111	X	Reserved for ACCESS.bus default address
1111 0XX	X	10-bit slave addressing
1111 1XX	X	Reserved for future use
0001 000	X	SMBus Host
0001 100	X	SMBus Alert Response Address
1100 001	X	SMBus Device Default Address

Table 4: Reserved SMBus Addresses

All other addresses are available for address assignment for dynamic address devices and/or for miscellaneous devices. Miscellaneous device addresses are discussed in Section 5.2.1.4.

The SMBus Alert Response Address (0001 100) can be a substitute for device master capability. See Appendix A for details.

The SMBus Device Default Address is reserved for use by the SMBus Address Resolution Protocol, which allows addresses to be assigned dynamically. See section 5.6 for details

Addresses not specified here or within the appendices are reserved for future use. All 10-bit slave addresses are reserved for future use and are outside the scope of this specification.

The host has the lowest legitimate address so that messages going to the host have the highest priority with respect to bus arbitration.

5.2.1. SMBus address types

Several SMBus 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 in actual SMBus systems.

5.2.1.1. Reserved addresses

SMBus, Access.bus and I^2C reserve several addresses for specific bus functions as defined in Table 4.

5.2.1.2. Purpose-assigned addresses

These addresses are assigned by the SMBus Working Group to specific types of devices. Each device type that obtains an assigned address has to have an SMBus specification associated with it. Some systems using SMBus assume that if a device exists at a purpose-assigned address then the device complies with the associated specifications for that address. For example, SMBus application to Smart Battery implementations assume that Smart Battery devices and controllers are at their purpose-assigned addresses. Thus, devices that do not meet the purpose-assigned address specifications for Smart Battery devices cannot be used in Smart Battery applications.

Other SMBus implementations do not rely solely on the device address to identify a device's functionality. In these applications, devices may have addresses that overlap with the purpose-assigned addresses.

The device manufacturer is forewarned that this may preclude use of that device in other applications. In general, purpose-assigned addresses should be avoided except for devices that are intended to meet the specification for the corresponding address(es). The device manufacturer should consult the SMBus WG to get the latest information on purpose-assigned addresses as a guide to whether their address assignment is disallowed in certain SMBus applications.

5.2.1.3. Prototype Addresses

Slave Address	Description
1001 0XX	Prototype Addresses

Table 5: Prototype addresses

The Prototype addresses (1001 0XX) are reserved for device prototyping and experimenting in applications that utilize purpose-assigned addresses. They are not intended for production parts and should never be assigned to any device.

5.2.1.4. Miscellaneous device addresses

Manufacturers have produced and may continue producing SMBus compatible devices for specific system purposes, for which they do not need to implement the complete SMBus specification, or for which they do not require explicit support from the OS. Such devices, for example, may be port expanders, D/A circuits, etc. The SMBus web site (http://www.smbus.org) includes a page where SBS-IF member companies can share address information for such devices. This page may aid manufacturers to avoid address conflicts with devices likely to co-exist on the same SMBus segment.

5.2.1.5. Dynamically assigned addresses

Version 2.0 introduces the concept of dynamically assigned addresses. This is detailed in section 5.6.

5.3. Using a device

A typical 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 may cause an error condition. In accordance with this specification, the Most Significant Bit is transferred first.

There are eleven possible command protocols for any given device. A slave device may use any or all of the eleven protocols to communicate. The protocols are Quick Command, Send Byte, Receive Byte, Write Byte, Write Word, Read Byte, Read Word, Process Call, Block Read, Block Write and Block Write-Block Read Process Call.

5.4. Packet error checking

Version 1.1 of SMBus introduced a Packet Error Checking mechanism to improve reliability and communication robustness. Implementation of Packet Error Checking by SMBus devices is optional for SMBus devices but is required for devices participating in and only during the ARP process. SMBus devices that implement Packet Error Checking must be capable to communicate with the host and other devices that do not implement the Packet Error Checking mechanism.

Packet Error Checking, whenever applicable, is implemented by appending a Packet Error Code (PEC) at the end of each message transfer. Each protocol (except for Quick Commandand the host notify protocol 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.

5.4.1. Packet error checking implementation

The SMBus must accommodate any mixture of 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 only process the message if the PEC is correct. Implementations are encouraged to issue a NACK if the PEC is present but not correct.

5.4.1.1. ACK/NACK and Packet Error Checking

The ACK/NACK bit in an SMBus byte is as susceptible to corruption as any other bit in an SMBus packet. Hence, an ACK at the end of a PEC is not a guarantee that the PEC is correct. A master-transmitter receiving an ACK at the end of a write should not necessarily assume that the write was successfully carried out at the slave-receiver of the write, although it is highly likely that it was.

A NACK received after a PEC by a master-transmitter indicates that the link layer of the slave-receiver became aware of an error with the transmission in time to supply a NACK at the end of the PEC byte. This may be due to an incorrect PEC or any other error. Errors discovered above the data link layer may also be indicated with a NACK if the device is fast enough to discover and indicate the error when the NACK is due.

An ACK received after a PEC by a master-transmitter means that no error could be determined by the link layer in the slave-receiver in time to supply a NACK. This might be because the receiver is not able to check the validity of the PEC in real time.

If a master transmitter wants to be sure that a write-operation is performed correctly at the target device, it must use some higher-layer mechanism to verify this. This might take the form of a read-with-PEC of the data; receipt of a correct PEC would reliably indicate that that the original write occurred without error.

When a master-receiver reads data from a slave-transmitter, the ACK/NACK supplied by the master-receiver at the end of the transaction has little meaning other than to mark the end of the last byte. The slave-transmitter is supplying the data, and if the PEC supplied by the slave-transmitter is correct, the master-receiver may assume that the data was received as the slave transmitted it. If not, it is up to the master-receiver to take any appropriate remedial action.

5.4.1.2. Master implementation

A master may use PEC on any transaction. It is required that the master have either a priori knowledge of whether or not the target slave supports PEC or a way to determine whether the target slave supports PEC. The use of PEC is governed by upper layer protocols (e.g. device drivers), specifications (e.g.requirements of the SMBus ARP protocol) or detection algorithms for a given class of devices (e.g. smart batteries).

5.4.1.3. 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.

Each bus transaction requires a Packet Error Code (PEC) calculation by both the transmitter and receiver within each packet. The PEC uses an 8-bit cyclic redundancy check (CRC-8) of each read or write bus transaction to calculate a Packet Error Code (PEC). The PEC may be calculated in any way that conforms to a CRC-8 represented by the polynomial, $C(x) = x^8 + x^2 + x^1 + 1$ and must be calculated in the order of the bits as received.

Calculating the PEC 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 provides some example CRC-8 methods.

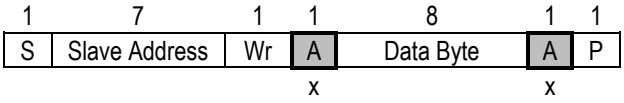
The PEC is appended to the message as dictated by the protocols in section 5.5. The PEC calculation includes all bytes in the transmission, including address, command and data. The PEC calculation does not include ACK, NACK, START, STOP nor Repeated START bits. This means that the PEC is computed over the entire message from the first START condition.

Whether a device implements packet error checking may be determined by the specification revision code that is present in the SpecificationInfo() command for a Smart Battery, Smart Battery Charger or Smart Battery Selector. See these individual specifications for exact revision coding identities. It may also be discovered in the UDID, defined in section 5.6.1, for other devices.

5.5. Bus Protocols

Following is a description of the various SMBus protocols with and without a Packet Error Code. Compliant devices need not support all the protocols defined in this section. The results returned by such a device to a protocol it does not support are undefined.

Below is a key to the protocol diagrams in this section. Not all protocol elements will be present in every command. For instance, not all packets are required to include the Packet Error Code.



S	Start Condition
Sr	Repeated Start Condition
Rd	Read (bit value of 1)
Wr	Write (bit value of 0)
x	Shown under a field indicates that that field is required to have the value of 'x'
A	Acknowledge (this bit position may be '0' for an ACK or '1' for a NACK)
P	Stop Condition
PEC	Packet Error Code
	Master-to-Slave
	Slave-to-Master
…	Continuation of protocol

Figure 5-1: SMBus packet protocol diagram element key

A value shown below a field in the following diagrams is a mandatory value for that field.

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.

Examples of these formats will be seen in the SMBus protocols below.

5.5.1. 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.

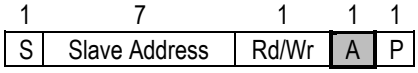


Figure 5-2: Quick command protocol

5.5.2. 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.

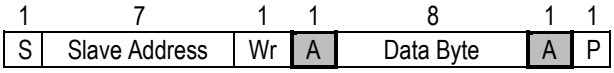


Figure 5-3: Send byte protocol

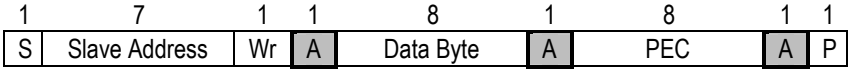


Figure 5-4: Send byte protocol with PEC

5.5.3. 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 NACK (a '1' in the ACK bit position) signifies the end of a read transfer.

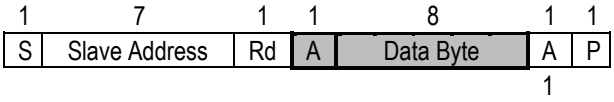


Figure 5-5: Receive byte protocol

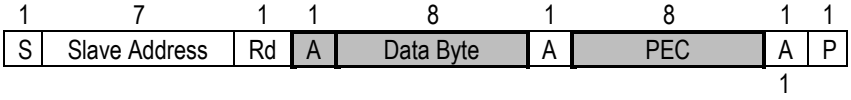


Figure 5-6: Receive byte protocol with PEC

5.5.4. Write byte/word

The first byte of a Write Byte/Word access is the command code. The next one or two bytes, respectively, 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, and the entire transaction is finished with a STOP condition.

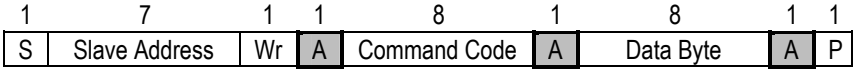


Figure 5-7: Write byte protocol

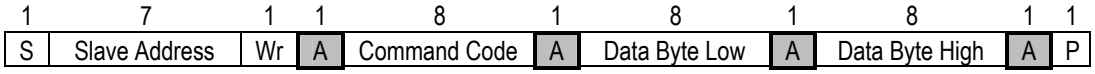


Figure 5-8: Write Word Protocol

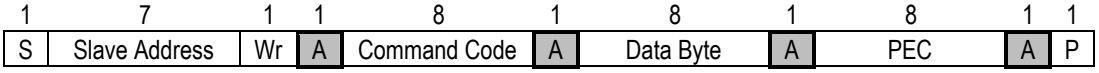


Figure 5-9: Write byte protocol with PEC

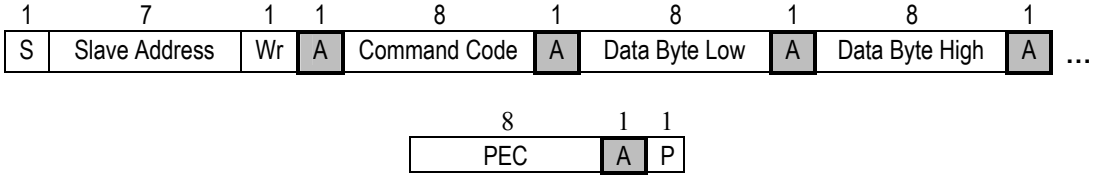


Figure 5-10: Write Word Protocol with PEC

5.5.5. 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 one or two bytes of data.

Note that there is no STOP condition before the repeated START condition, and that a NACK signifies the end of the read transfer.

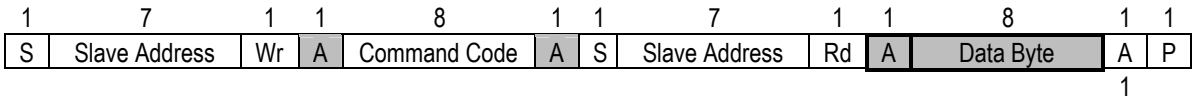


Figure 5-11: Read Byte Protocol

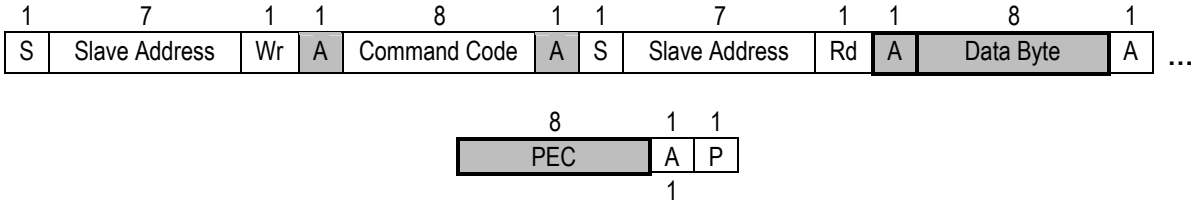


Figure 5-12: Read byte protocol with PEC

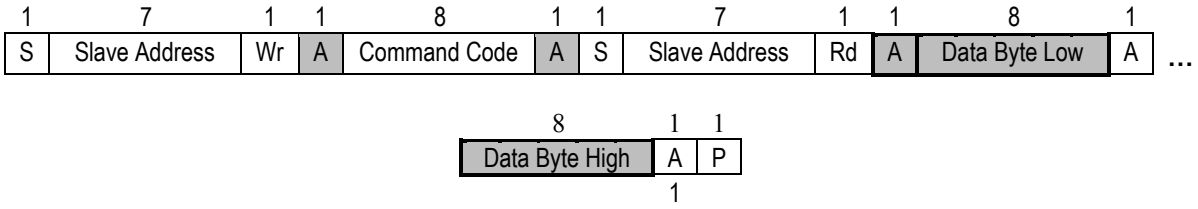


Figure 5-13: Read word protocol

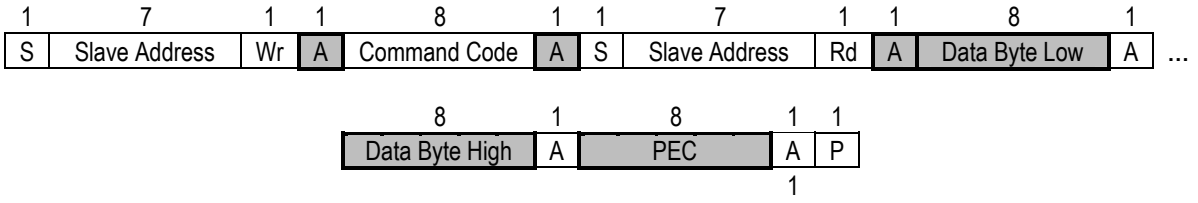


Figure 5-14: Read word protocol with PEC

5.5.6. 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 without the Read-Word command field and the Write-Word STOP bit.

Note that there is no STOP condition before the repeated START condition, and that a NACK signifies the end of the read transfer.

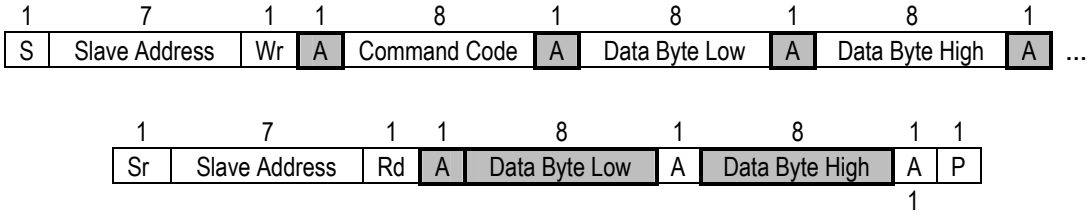


Figure 5-15: Process Call

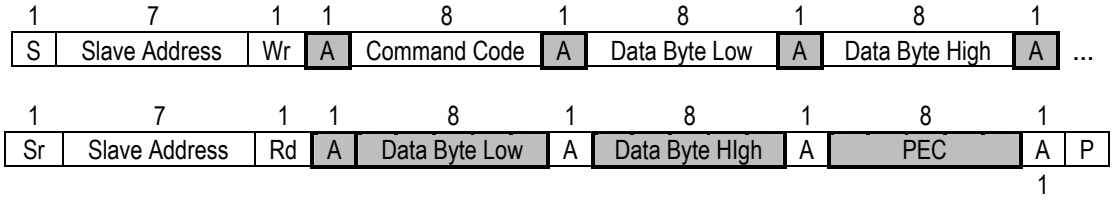


Figure 5-16: Process Call with PEC

5.5.7. Block write/read

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 has 20 bytes to send, the byte count field will have the value 20 (14h), followed by the 20 bytes of data. The byte

count does not include the PEC byte. The byte count may not be 0. A Block Read or Write is allowed to transfer a maximum of 32 data bytes.

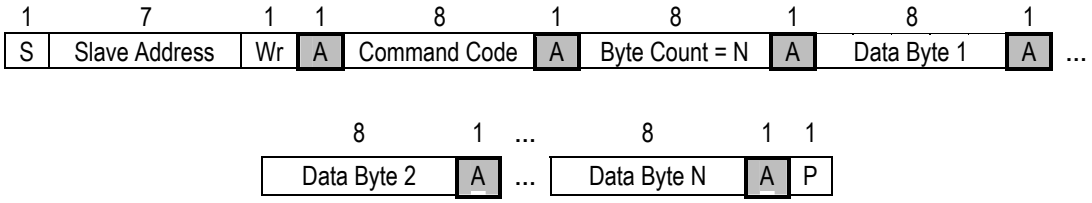


Figure 5-17: Block Write

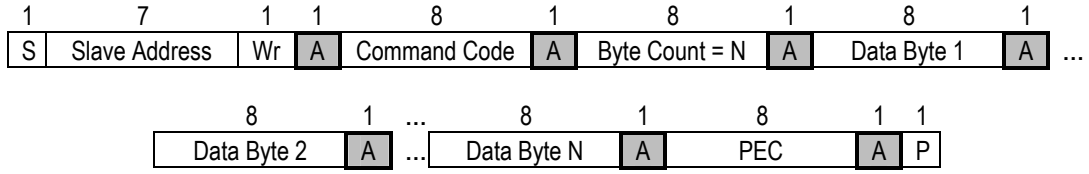


Figure 5-18: Block Write with PEC

A Block Read differs from a block write in that the repeated START condition exists to satisfy the requirement for a change in the transfer direction. A NACK immediately preceding the STOP condition signifies the end of the read transfer.

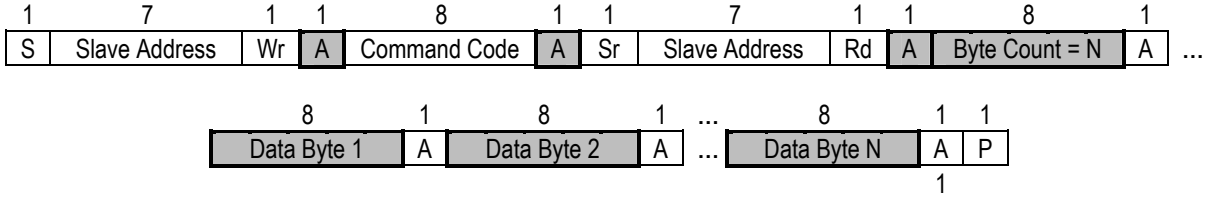


Figure 5-19: Block Read

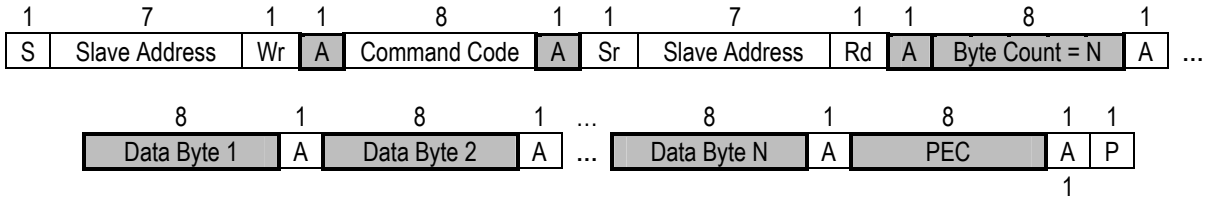


Figure 5-20: Block Read with PEC

5.5.8. Block write-block read process call

The block write-block read process call is a two-part message. The call begins with a slave address and a write condition. After the command code the host issues a write byte count (M) that describes how many more bytes will be written in the first part of the message. If a master has 6 bytes to send, the byte count

field will have the value 6 (0000 0110b), followed by the 6 bytes of data. The write byte count (M) cannot be zero.

The second part of the message is a block of read data beginning with a repeated start condition followed by the slave address and a Read bit. The next byte is the read byte count (N), which may differ from the write byte count (M). The read byte count (N) cannot be zero.

The combined data payload must not exceed 32 bytes. The byte length restrictions of this process call are summarized as follows:

- M ≥ 1 byte
- N ≥ 1 byte
- M + N ≤ 32 bytes

The read byte count does not include the PEC byte. The PEC is computed on the total message beginning with the first slave address and using the normal PEC computational rules. It is highly recommended that a PEC byte be used with the Block Write-Block Read Process Call.

Note that there is no STOP condition before the repeated START condition, and that a NACK signifies the end of the read transfer.

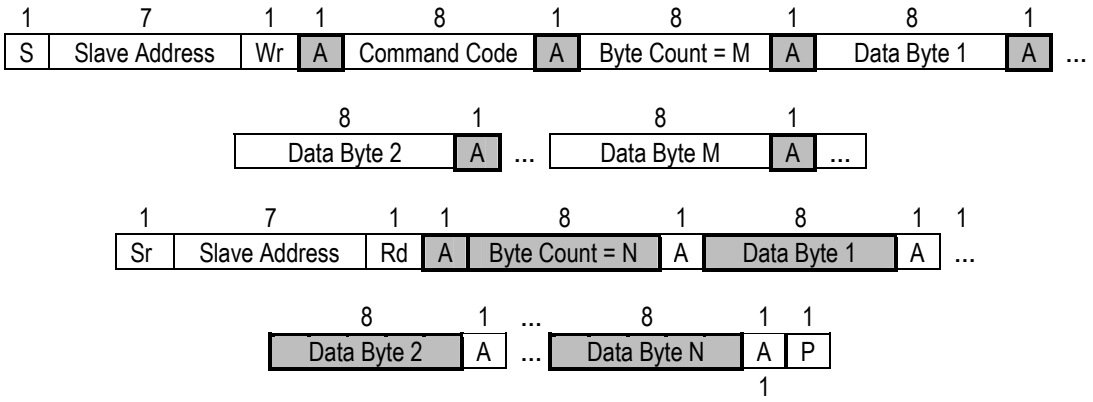


Figure 5-21: Block Write - Block Read Process Call

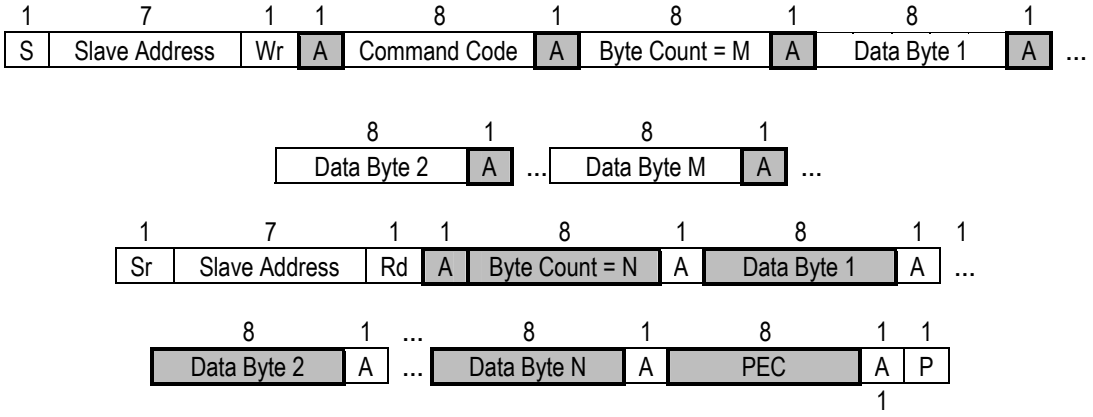


Figure 5-22: Block Write - Block Read Process Call with PEC

5.5.9 SMBus host notify protocol

To prevent messages coming to the SMBus host controller from unknown devices in unknown formats only one method of communication is allowed, a modified form of the Write Word protocol. The standard Write Word protocol is modified by replacing the command code with the alerting device's address. This protocol MUST be used when an SMBus device becomes a **master** in order to communicate with the SMBus host acting as a **slave**.

Communication from SMBus Device to SMBus Host begins with the SMBus Host address (0001 000b). The message's Command Code is the initiating SMBus device's address. From this, the SMBus Host knows the origin of the following 16 bits of device status. The contents of the status are device specific.

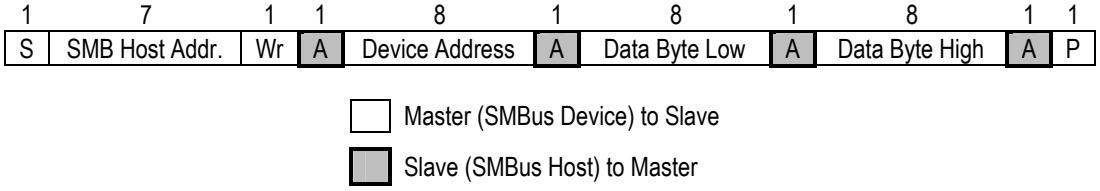


Figure 5-23: 7-bit Addressable Device to Host Communication

SMBus hosts must support the host notify protocol. Hosts may implement the optional #SMBALERT line if devices in the system use it.

5.6. SMBus Address resolution protocol

SMBus slave address conflicts can be resolved by dynamically assigning a new unique address to each slave device. The Address Resolution Protocol (ARP) possesses the following attributes:

- Address assignment utilizes the standard SMBus physical layer arbitration mechanism
- Assigned addresses remain constant while device power is applied; address retention through device power loss is also allowed
- No additional SMBus packet overhead is incurred after address assignment. (i.e. subsequent accesses to assigned slave addresses have the same overhead as accesses to fixed address devices.)
- Any SMBus master can enumerate the bus

5.6.1. Unique Device Identifier (UDID)

In order to provide a mechanism to isolate each device for the purpose of address assignment each device must implement a unique device identifier (UDID). This 128-bit number is comprised of the following fields:

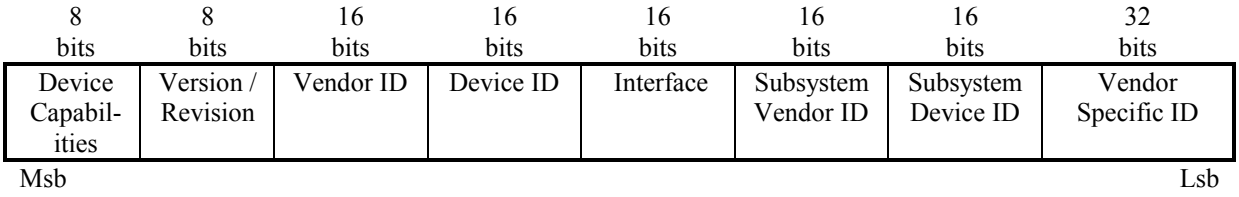


Figure 5-24: UDID

Device Capabilities Describes the device's capabilities. See detail below.

Version / Revision	UDID version number, and a silicon revision identification. See detail below.
Vendor ID	The device manufacturer's ID as assigned by the SBS Implementers' Forum or the PCI SIG.
Device ID	The device ID as assigned by the device manufacturer (identified by the Vendor ID field).
Interface	Identifies the protocol layer interfaces supported over the SMBus connection by the device. For example, ASF and IPMI.
Subsystem Vendor ID	This field may hold a value derived from any of several sources: • The device manufacturer's ID as assigned by the SBS Implementers' Forum or the PCI SIG. • The device OEM's ID as assigned by the SBS Implementers' Forum or the PCI SIG. • A value that, in combination with the Subsystem Device ID, can be used to identify an organization or industry group that has defined a particular common device interface specification.
Subsystem Device ID	The subsystem ID identifies a specific interface, implementation, or device. The Subsystem ID is defined by the party identified by the Subsystem Vendor ID field.
Vendor-specific ID	A unique number per device. See detail below.

5.6.1.1. Device capabilities field

The Device Capabilities field serves multiple purposes:

1. Reports generic SMBus capabilities.
2. Guarantees order of ARP resolution. Because a 'zero' bit wins arbitration over a '1' bit, and the Address Type bits are the first two bits presented when a device presents its UDID, all fixed address devices on the bus are detected during ARP first, followed by devices with dynamic and persistent addresses, and so on. The bits in the brackets below show the highest two bits, the address type field, within the Device Capabilities field:

- `[00]` Fixed Address devices are identified first.
- `[01]` Dynamic and Persistent Address devices are identified next.
- `[10]` Dynamic and Volatile Address devices are identified next.
- `[11]` Random Number devices are identified last.

7	6	5	4	3	2	1	0
Address Type		Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	Reserved (0)	PEC Supported
Msb							Lsb

Figure 5-25: 8-bit device capabilities field

Address Type	These two bits describe the type of address contained in the device: 00 Fixed Address device 01 Dynamic and persistent address device 10 Dynamic and volatile address device 11 Random number device
PEC Supported	This bit is set if the device supports Packet Error Code on all commands supported at the device's SMBus address associated with this UDID. If this bit is not set, the

ability of the device to support PEC is unknown.

Reserved Reserved bits are for future extendibility and must be returned as 0b and ignored when read.

5.6.1.2. Version/Revision field

The version/revision field serves multiple purposes:

1. Identifies a UDID version to allow for future extendibility.
2. Identifies the silicon revision level.

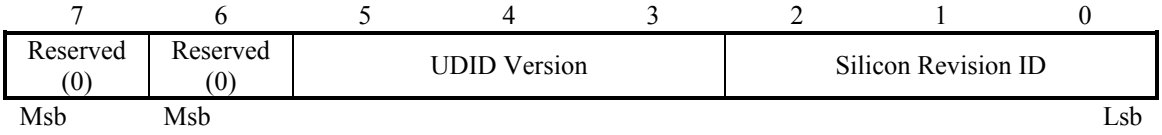


Figure 5-26: Version/Revision field

Reserved Reserved bits are for future extendibility and must be a 0b.

UDID Version These bits define the UDID version as defined here:

- 000b Reserved
- 001b UDID version 1 (defined for SMBus 2.0 release)
- 010b – 111b Reserved for future use

It is expected that the version will increment as the bit definitions or protocols in this section change

Silicon Revision ID These bits are used to designate the silicon revision level. The vendor determines this value. The vendor is encouraged to increment this value when silicon changes are made that will/might affect the software interface (e.g. new features, changed interface, etc.). In the event that all 8 encoded values are exhausted, the vendor is encouraged to use a different Device ID for the next revision.

5.6.1.3. Interface

The Interface field defines the SMBus version and the Interface Protocols supported.

The least significant nibble is used to identify the SMBus version:

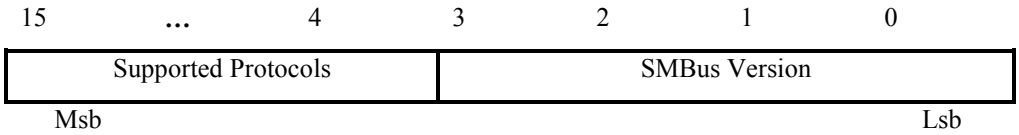


Figure 5-27: Interface field

SMBus Version These bits define the SMBus version as defined here:

- 0000b SMBus 1.0 – do not use in ARPable devices
- 0001b SMBus 1.1– do not use in ARPable devices

- 0010b Reserved
- 0011b Reserved
- 0100b SMBus Version 2.0
- All other values reserved

Note: The values 0000b and 0001b in the field definition above support use of the UDID definition in other specifications.

The most significant bits of the interface field are used to identify the protocols supported by the device:

Protocols	Meaning
bit 15	Reserved for future definition under the SMBus specifications.
bit 14	Reserved for future definition under the SMBus specifications.
bit 13	Reserved for future definition under the SMBus specifications.
bit 12	Reserved for future definition under the SMBus specifications.
bit 11	Reserved for future definition under the SMBus specifications.
bit 10	Reserved for future definition under the SMBus specifications.
bit 9	Reserved for future definition under the SMBus specifications.
bit 8	Reserved for future definition under the SMBus specifications.
bit 7	Reserved for future definition under the SMBus specifications.
bit 6 IPMI	Device supports additional interface access and capabilities per IPMI specifications
bit 5 ASF	Device supports additional interface access and capabilities per ASF specifications
bit 4 OEM	Device supports vendor-specific access and capabilities per the Subsystem Vendor ID and Subsystem Device ID fields returned by discoverable SMBus devices. The Subsystem Vendor ID identifies the vendor or defining body that has specified the behavior of the device. The Subsystem Device ID is used in conjunction with the System Vendor ID to specify a particular level of functional equivalence for the device.

5.6.1.4. SubSystem IDs

The SubSystem Vendor ID can be specified as 0000h if the SubSystem fields are unsupported. If the SubSystem Vendor ID is 0000h, the SubSystem Device ID must also be 0000h. These fields may not be supported for inexpensive or generic type sensors that do not require subsystem identification/differentiation. If these fields are supported, it is required that the values be stored in some form of non-volatile storage.

5.6.1.5. Vendor-specific ID

This field is used to provide a unique ID for functionally equivalent devices. This is for devices that would otherwise return identical UDIDs for the purpose of address assignment. This field is defined by the device manufacturer (as specified by the Vendor ID field) who may employ a central numbering scheme or a random number scheme for dynamic address devices. The data in this field is irrelevant for devices that do not support dynamic addressing.

The rules of this field are stated here for clarity:

1. Devices that support an assigned device address must support a unique ID in this field.

2. If a pre-assigned unique ID is used, at least 16-bits must be unique. However, 32-bits is recommended.
3. If a random number is implemented in this field, Random Number Requirements must be met.
4. Devices that support a fixed device address must still implement this field but not uniquely.

Uniqueness is important to guarantee that two like devices are identified discretely. It is the responsibility of the device/system manufacturer to determine the possibility of like devices, and the mechanism for providing uniqueness via the UDID and slave address fields.

5.6.1.6. Random number requirements

If a random number is utilized the following requirements must be met:

1. It must be at least 16 bits in length.
2. The device is not allowed to support a persistent slave address.
3. The device is not allowed to support fixed addresses. (If the device has a fixed address mode, the Vendor Specific ID should be a constant, and therefore not random – this is required to guarantee that the SMBus ARP resolution order is maintained)
4. The random number must be retained while the device has power with the exceptions described in items 5 and 6.
5. The random number must be regenerated when the device receives the Reset Device command.
6. The random number must be regenerated when the device senses a bus collision during a read operation directed to its Assigned Slave Address. When this happens, the device must issue a host notify protocol if the device supports it.

5.6.2. Power-on reset

Power-on reset is described in section 3.1.4.2. In the case of ARP-capable devices, 'operational state' implies the ability to respond to ARP commands as required in this section.

Each slave device must fit into only one of these categories and must obey the power on reset state:

Device Type	AR Flag	AV Flag	SMB Address	UDID
PSA (Persistent Slave Address)	CLEAR	Read from NVR	Read from NVR; undefined if AV Flag is CLEAR	NO CHANGE
Non-PSA / Non-Random Number	CLEAR	CLEAR	undefined	NO CHANGE
Non-PSA / Random Number	CLEAR	CLEAR	undefined	Generate Random Number

Table 6: Internal state of ARP-capable devices on power-on reset

5.6.3. ARP commands

The ARP Master can issue general commands and directed commands. A general command is targeted at all devices and is required for the address resolution process. A directed command is targeted at a single device. All packets originated by the ARP Master use Packet Error Checking (See Section 7.5) and begin with the basic format:

<SMBus Device Default Address> <command>

<SMBus Device Default Address>	1100 001 (the R/W# bit completes the byte)
<command>	*General* (0x00 through 0x1F) 0x00 = Reserved 0x01 = Prepare to ARP 0x02 = Reset Device (general) 0x03 = Get UDID (general) 0x04 = Assign Address 0x05 = Reserved • • • 0x1F = Reserved *Directed* • Odd numbered commands denote Get UDID from a specific slave and are of the form yyyy yyy1 where yyyy yyy is the 7-bit targeted slave address. For example, a command of 0x21 is a directed Get UDID to slave address 0010 000. • Even numbered commands denote Reset Device to a specific slave and are of the form yyyy yyy0 where yyyy yyy is the 7-bit targeted slave address. For example, a command of 0x5C is a directed Reset Device to slave address 0101 110. • Values of 0xFE and 0xFF are reserved

Table 7: ARP command number scheme

An ARP Enumerator is allowed to issue the “Prepare to ARP”, “Get UDID” (general and directed), and “Assign Address” commands; it is not allowed to issue the Reset Device commands. It must execute the “Assign Address” command for each device in the general “Get UDID” command using the same address that is returned by the “Get UDID” command. An SMBus Enumerator is not allowed to re-assign addresses and it is not allowed to assign an address to a device with an invalid/uninitialized address.

Devices can optionally support the “Notify ARP Master” command that is used to notify the ARP Master that the device requires address resolution. If the ARP Master supports this command, it must respond as a slave to this command and provide a software indication that the ARP needs to be executed.

5.6.3.1. Device categorization

SMBus devices are categorized as follows:

ARP-capable	Device supports all ARP commands with the exception of the optional host notify command. Slave address is assignable. Device supports both Reset commands.
Fixed and Discoverable	Device supports the Prepare to ARP, directed Get UDID, general Get UDID and Assign Address commands. Slave address is fixed; device will accept the Assign Address command but will not allow address reassignment. Device supports both Reset commands.
Fixed, not Discoverable	Device supports the directed Get UDID command. Slave address is fixed.
Non-ARP-capable	Device does not support any ARP commands. Slave address is fixed.

5.6.3.2. Prepare to ARP

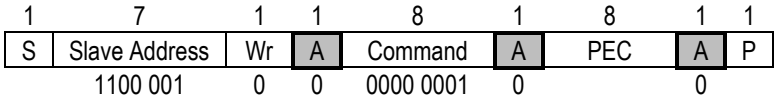
Action: always ACK/PROCESS

AR Flag: CLEAR

AV Flag: NO CHANGE

This command informs all devices that the ARP Master is starting the ARP process. All ARP-capable devices must acknowledge all bytes in this SMBus packet and clear their Address Resolved (AR) flag. They must also cancel any pending "Notify ARP Master" commands. If the ARP Master detects that any of the bytes have not been acknowledged then it can assume that there are no ARP-capable devices present on the bus. Retries are recommended in case bus noise causes an erroneous NACK.

This command utilizes the standard SMBus Send Byte Protocol with PEC as illustrated below.



5.6.3.3. Reset device (general)

Action: always ACK/PROCESS

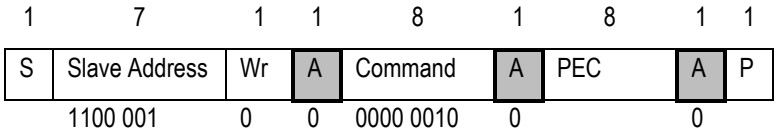
AR Flag: CLEAR

AV Flag: if (non-PSA) then CLEAR; else NO CHANGE

This command forces all non-PSA, ARP-capable devices to return to their initial state. That is, they must clear their AR (Address Resolved) and AV (Address Valid) flags; those devices that support the Persistent Slave Address must clear their AR flag. An ARP-capable device that implements a random number as part of its UDID must regenerate its random number upon receipt of this command. All ARP-capable devices must acknowledge all bytes in this SMBus packet. If the ARP Master detects that any of the bytes have not been acknowledged then it can assume that there are no ARP-capable devices present on the bus.

This reset is just for the ARP functions, and is not intended as a general device reset.

This command utilizes the standard SMBus Send Byte Protocol with PEC as illustrated below.

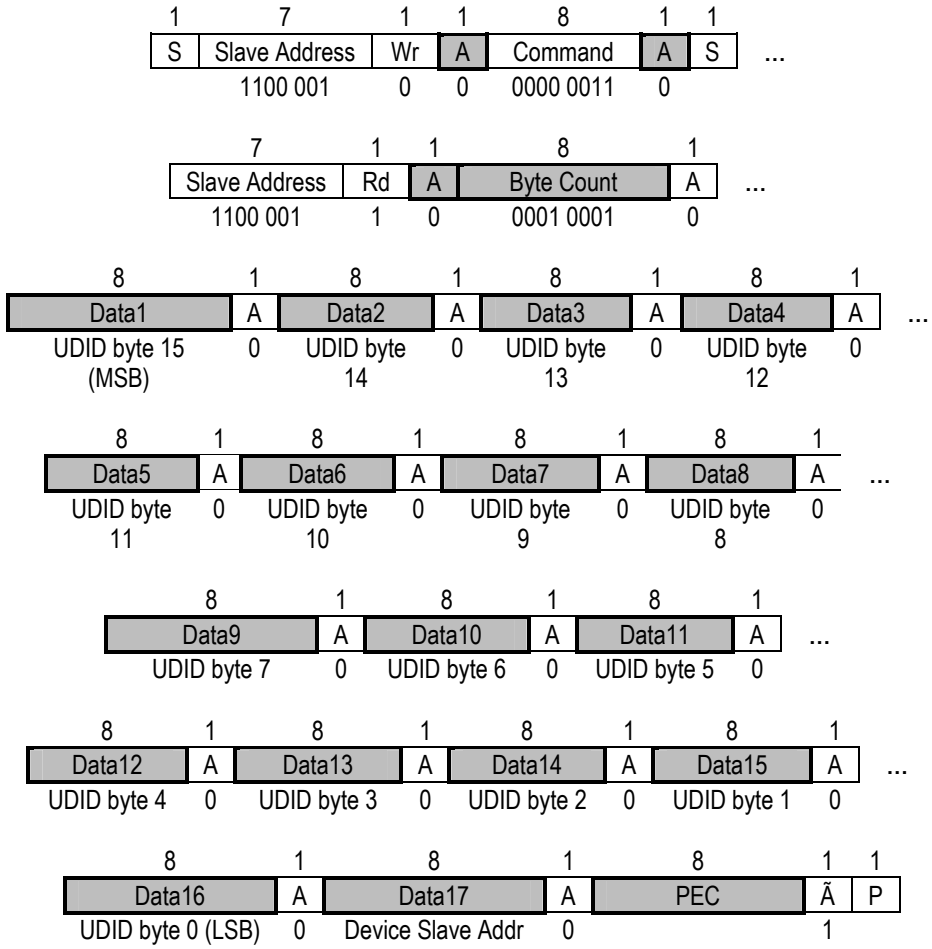


5.6.3.4. Get UDID (general)

Action: if (AR=0) then ACK/PROCESS; else NACK/REJECT.
AR Flag: NO CHANGE
AV Flag: NO CHANGE

This command requests ARP-capable and/or Discoverable devices to return their slave address along with their UDID. If the ARP Master detects that any of the first three bytes have not been acknowledged then it can assume that there are no ARP-capable or Discoverable devices present on the bus or all ARP-capable devices have valid assigned slave addresses.

This command utilizes the standard SMBus Block Read Protocol with PEC as illustrated below.



NOTES

- Bit 0 (lsb) in the Data17 field must be returned as 1.
- If a device has its AV flag clear then it must return 1111 111 for the remaining bits in the Data17 field.

5.6.3.5. Assign address

Action: always ACK; if (UDID match) then PROCESS.

AR Flag: SET if UDID matches.

AV Flag: SET if UDID matches.

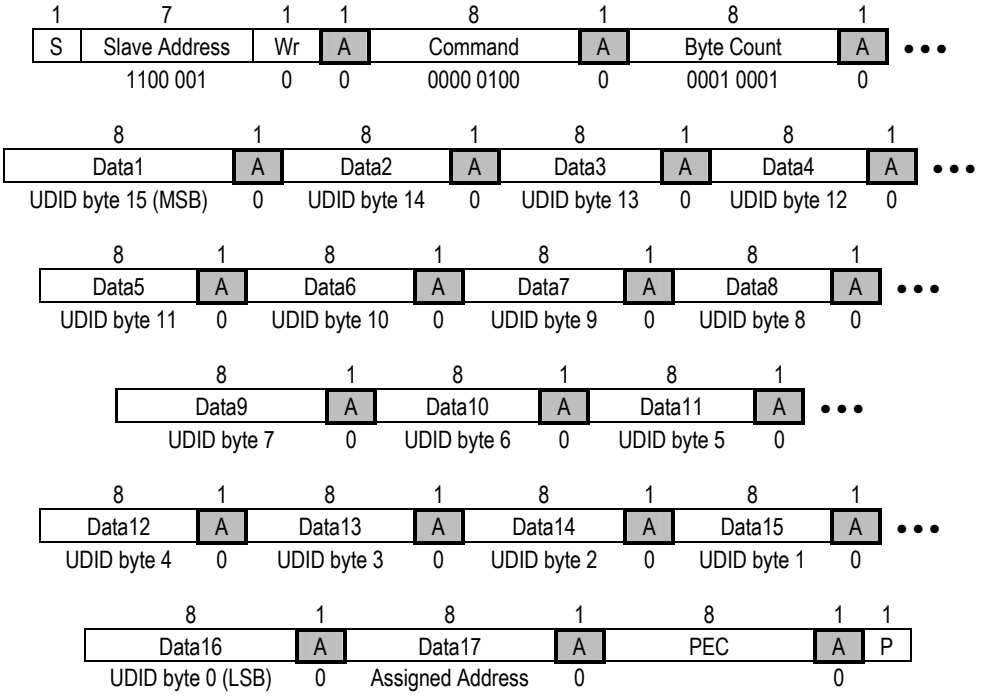
The ARP Master assigns an address to a specific device with this command. Since this command utilizes a particular device's UDID only that device will adopt the new address. All ARP-capable devices must monitor the UDID bytes in this packet (all bytes except the assigned address byte) . Once a device determines that it is not the target of the command (due to a UDID bit or byte mismatch) it must NACK the current byte, if possible, or the next byte. A slave device matching all but the last UDID byte has the choice to NACK the last UDID byte or the subsequent assigned address byte. If the ARP Master detects a NACK for any byte then it must assume that the target device is no longer present. It is suggested that the ARP Master consider retrying the command in order to guard against noise causing a present device to be ignored.

A slave device that matches the entire UDID must immediately adopt the new slave address. It must reprogram its Persistent Slave Address, if applicable. Bit 0 (lsb) of the Assigned Address field must be ignored.

NOTES:

1. A slave device must respond to this command even if its AR flag is SET.
2. The slave device only ACKs the PEC byte if it matches the value calculated on data it received, if not it must NACK the PEC byte AND ignore the "Assign Address" command. This behavior allows the host to determine that the slave device successfully accepted the address without any further bus activity.

This command utilizes the standard SMBus Block Write Protocol with PEC as illustrated below.



5.6.3.6. Get UDID (directed)

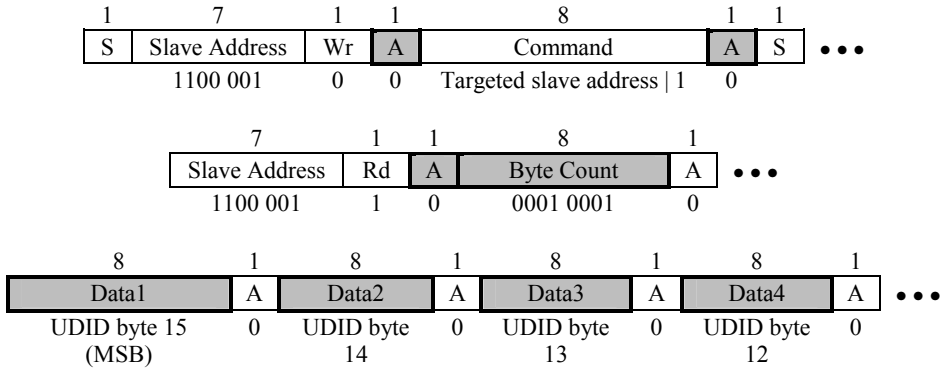
Action: if (AV=1) then ACK/PROCESS; else NACK/REJECT.

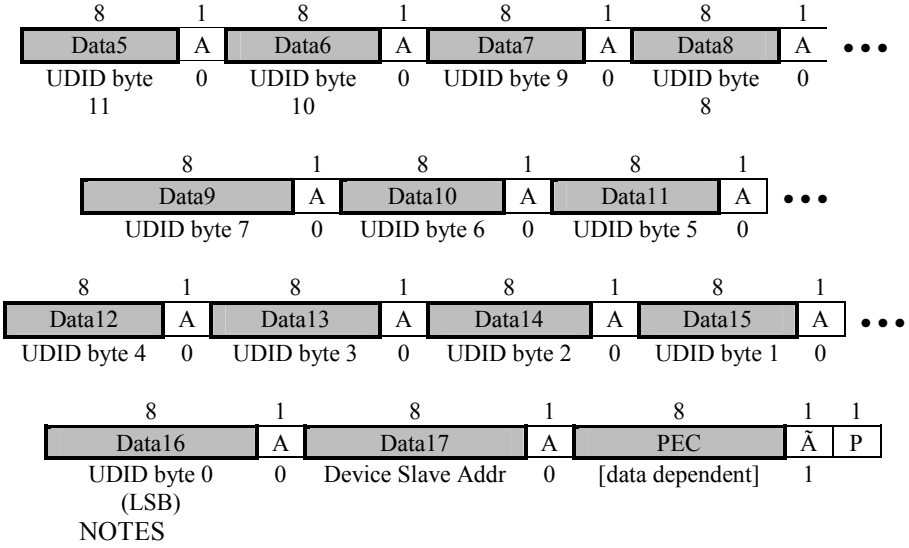
AR Flag: NO CHANGE

AV Flag: NO CHANGE

This command requests a specific ARP-capable device to return its Unique Identifier. If the ARP Master detects that any of the first three bytes have not been acknowledged then it can assume that no ARP-capable device is present at the targeted slave address.

This command utilizes the standard SMBus Block Read Protocol with PEC as illustrated below.





NOTES

Bit 0 (lsb) in the Data17 field must be returned as 1.

5.6.3.7. Reset device ARP (directed)

Action: if (AV=1) then ACK/PROCESS; else NACK/REJECT.

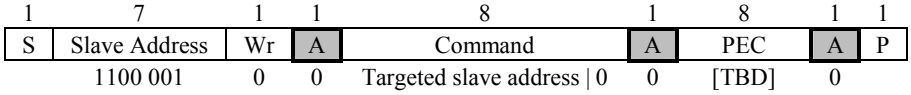
AR Flag: CLEAR

AV Flag: if (non-PSA) then CLEAR; else NO CHANGE

This command forces a specific non-PSA, ARP-capable device to return to its initial state. That is, it must clear its AR and AV flags; if the device supports the Persistent Slave Address it must clear its AR flag. An ARP-capable device that implements a random number as part of its UDID must regenerate its random number upon receipt of this command. If the ARP Master detects that any of the bytes have not been acknowledged then it can assume that no ARP-capable device is present at the targeted slave address.

This reset is just for the ARP functions, and is not intended as a general device reset.

This command utilizes the standard SMBus Send Byte Protocol with PEC as illustrated below.

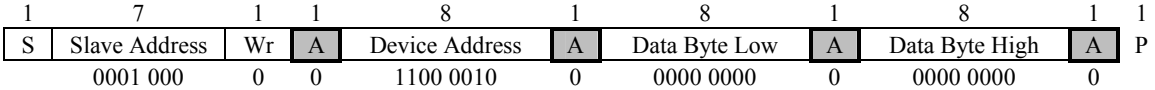


5.6.3.8. Notify ARP master

A device may use this command to notify the ARP Master that the device requires address resolution. The device may execute this command under the following circumstances:

- Device power up
- When the device senses a bus collision during a read operation directed to its Assigned Slave Address.

This command utilizes the standard SMBus protocol to communicate with the Host (at the SMBus Host Address as defined in 7.3.1) as illustrated below.



Note: The value of 0x0000 in the data field means that the device wishes to be ARPed. All other values are reserved for future use.

5.6.3.9. Implementation notes

An SMBus ARP Master in a Hot Plug System will typically not require the host notify command as it gets asynchronous indication of a device added or removed via other means, though there's no restriction in this specification against using this notification in those types of applications.

A mobile system that has addition and removal of devices on the SMBus can benefit from this command if the SMBus ARP Master and the removable devices support this command. (Note, there could be one device in the system that performs the notify on behalf of other devices. System software must always run the FULL ARP process.)

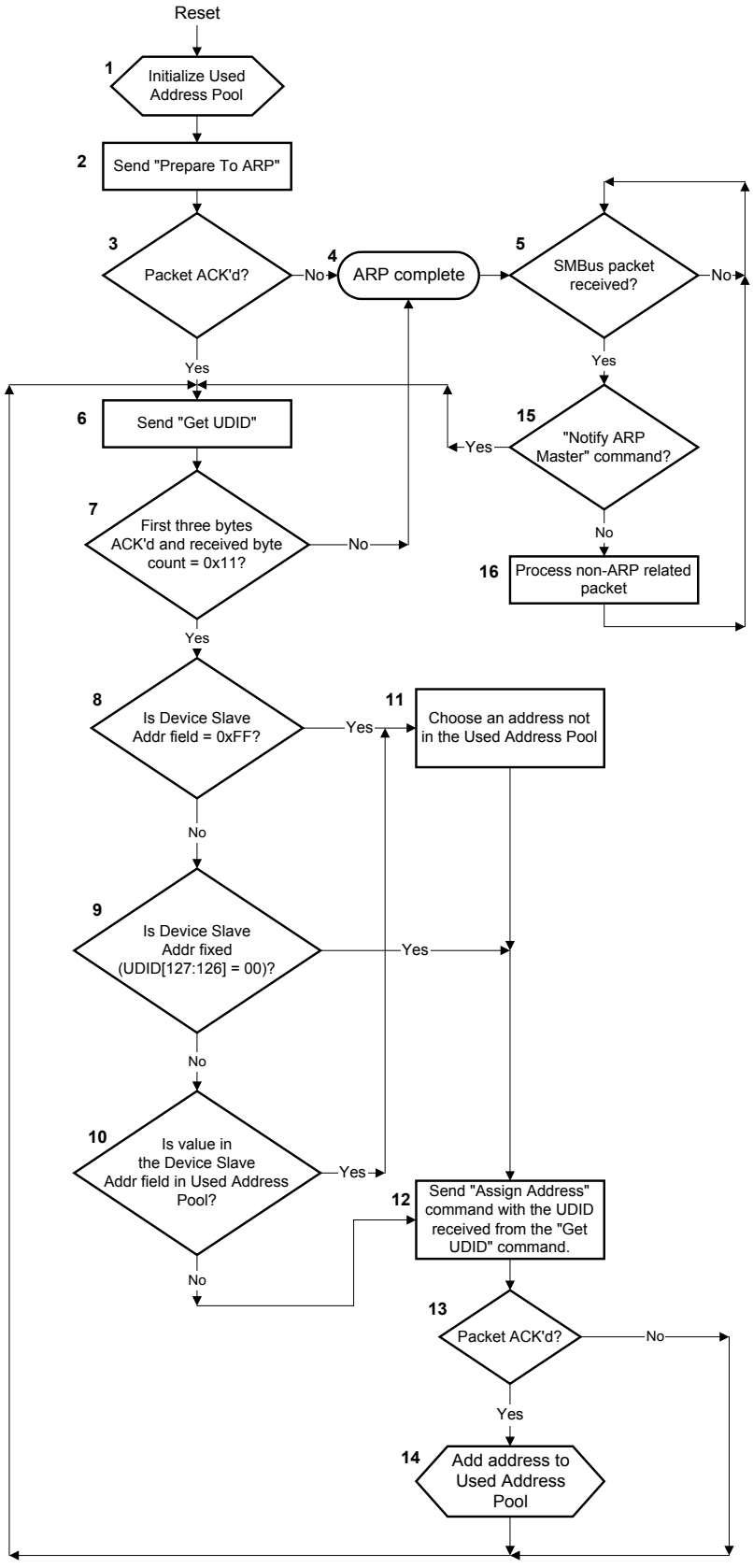
5.6.3.10. ARP execution

The ARP Master must always execute the ARP when it enters the working state and anytime it receives an SMBus status change indication (device added or removed – e.g. hot plug). The process begins with the ARP Master issuing the "Prepare To ARP" command. In all cases the ARP Master must be able to resolve addresses when it receives the "Notify ARP Master" command.

If the possibility exists that SMBus devices may join the system without a corresponding system reset (for example in hot-plug-capable systems), the ARP Master may optionally choose to issue the Get UDID (General) command at least once every 10 seconds in order to discover newly added devices that require address resolution but that don't support the "Notify ARP Master" command. No device whose AR flag is clear will respond to this command. However, a newly added device will enter the system with a power-on reset, which will reset its AR flag. It will respond to a Get UDID (General) command with its UDID. The host may choose to assign such a newly added device a non-conflicting address or it may choose to re-ARP the entire bus.

The ARP Master or any other SMBus master can perform device "discovery" or "enumeration" by executing a subset of the ARP. This is accomplished by issuing the "Prepare To ARP" command followed by repeatedly issuing the general "Get UDID" and "Assign Address" commands until no device acknowledges. Until the ARP process is complete the ARP Master must not wait more than two seconds before issuing a general "Get UDID" command after issuing the previous general "Get UDID" command. This restriction is important to allow another SMBus master to determine when it is safe to do an enumeration of the bus.

5.6.3.11. ARP master behavior



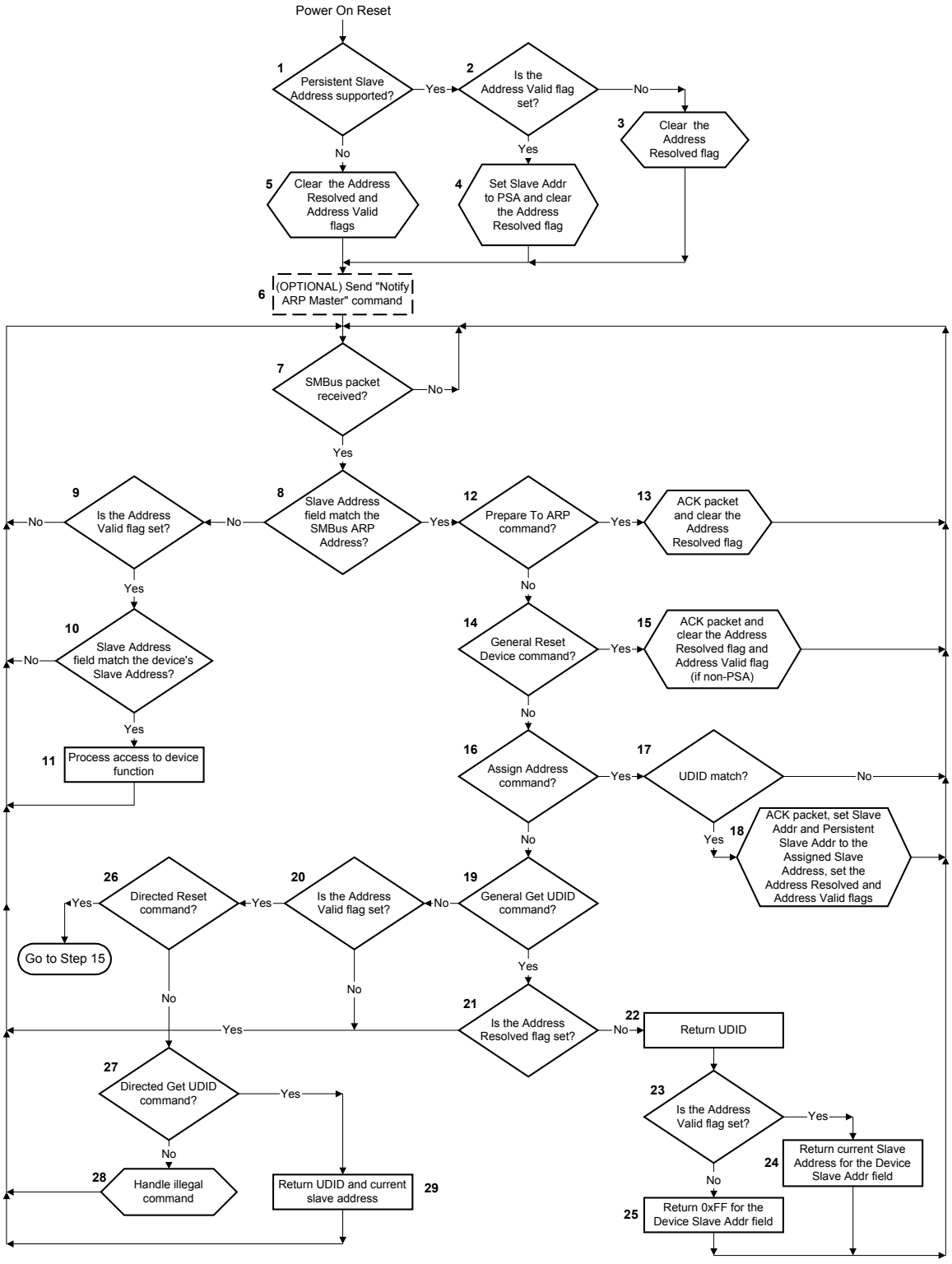
Referring to the previous flow diagram the ARP Master operates as follows:

1. Upon starting, the ARP Master will initialize its Used Address Pool. Initially this will consist of the slave addresses of fixed SMBus devices known to the ARP Master and reserved addresses (as defined in Appendix C).
2. Send the "Prepare To ARP" command.
3. Check for an acknowledgement for all bytes in the previous packet. If any bytes were not acknowledged then the ARP Master can assume that no ARP-capable devices are present and may therefore consider the ARP process complete and Proceed to step 4. If all bytes were acknowledged then go to step 6.
4. The ARP Master found no response to the "Prepare To ARP" command so it can assume that no ARP-capable devices are present in the system at this time. The ARP Master may periodically re-issue the "Prepare To ARP" command to discover any ARP-capable devices added. Proceed to step 5.
5. Wait for an SMBus packet. If a packet is received proceed to step 15.
6. Send the "Get UDID" command.
7. Check for an acknowledgement for the first three bytes and verify that the byte count value received is 0x11. If not, then the ARP Master can assume that an ARP-capable device(s) is no longer present and may therefore consider the ARP process complete and Proceed to step 4. Otherwise proceed to step 8.
8. Check the value of the Device Slave Address received. If 0xFF then proceed to step 11 since this device does not possess a valid slave address. Otherwise proceed to step 9.
9. Determine if this device has a fixed slave address. If bits 127 and 126 of the UDID are 00b than it has a fixed address, so procede to step 12. Otherwise proceed to step 10.
10. The device possesses a valid slave address. However, the ARP Master must check this address against the Used Address Pool to insure that no other device has already been assigned the same address. If the received Device Slave Address is found in the Used Address Pool then proceed to step 11. If not, then the device can keep its current slave address but needs acknowledgement from the ARP Master so proceed to step 12.
11. Select a slave address that is not in the Used Address Pool and proceed to step 12.
12. Send the "Assign Address" command with the UDID returned by the device in the "Get UDID" command packet.
13. Check for acknowledgement of all bytes in the "Assign Address" command packet. If any byte was not acknowledged then the ARP Master assumes the device is no longer present; proceed to step 6 to determine if there are more devices requiring address resolution. If all bytes were acknowledged then the ARP Master assumes that the device has accepted the address assignment; proceed to step 13.
14. The device now has a valid slave address. The ARP Master must add this address to the Used Address Pool. Proceed to step 6 to determine if there are more devices requiring address resolution.
15. The ARP Master checks to see if the received packet was the "Notify ARP Master" command. If so, then it must execute the ARP to resolve the address for the newly added device(s); proceed to step 6. If not, then proceed to step 16.
16. The ARP Master received a non-ARP related packet. Process it accordingly and proceed to step 5.

The ARP Master behavior flow diagram cover the case when the ARP Master has come out of a reset state. The ARP supports "hot-plug" devices so the ARP Master must be prepared to execute the ARP at any time; step 15 covers the case when a device a device issues the "Notify ARP Master" command. Since that command is optional the ARP Master cannot rely on a notification from the device upon its appearance on the SMBus. As such the ARP Master should periodically issue the "Prepare To ARP" command if the SMBus implementation supports "run-time" device addition. Doing so will also allow the ARP Master to determine if any ARP-capable devices have been removed from the SMBus. In this case the ARP Master can remove addresses from the Used Address Pool if it doesn't detect a response from a device previously assigned an address.

The flow diagram also does not cover consideration for bus timeout mechanisms or retries. These should be implemented in order to comply with the bus timing specifications.

5.6.3.12. ARP-capable device behavior



Referring to the previous flow diagram an ARP-capable device operates as follows:

1. After exiting the power on reset state, a device that supports the Persistent Slave Address (PSA) will go to step 2 to see if it is valid. If the device does not support the PSA, it will proceed to step 5.
2. A device supporting PSA must check its Address Valid flag which is a non-volatile. If that flag is set then it has previously received an assigned slave address and proceeds to step 4. If the Address Valid flag is cleared then it must proceed to step 3
3. Although the device supports the PSA the value is currently invalid. The device must clear the Address Resolved flag indicating that it has not had its slave address assigned. Proceed to step 6.
4. The device has a valid PSA so it assumes that slave address for now. However, this address has not been resolved by the ARP Master so the device must clear its Address Resolved flag. Proceed to step 6.
5. The device does not support the PSA so it must clear its Address Valid and Address Resolved flags. Proceed to step 6.
6. If supported, the device will master the SMBus and send the "Notify ARP Master" command. This will inform the ARP Master that a new device is present. Proceed to step 7.
7. The device waits for an SMBus packet.
8. Upon receipt of an SMBus packet the device must first check the received slave address against the SMBus Device Default Address. If there is a match then it proceeds to step 12, otherwise it proceeds to step 9.
9. The received address is not the SMBus Device Default Address so the packet is potentially addressed to the device's core function. The device must check its Address Valid bit to determine whether or not to respond. If the Address valid bit is set then it proceeds to step 10, otherwise it must return to step 7 and wait for another SMBus packet.
10. Since the device has a valid slave address it must compare the received slave address to its slave address. If there is a match then it proceeds to step 11, otherwise it must return to step 7 and wait for another SMBus packet.
11. The device has received a packet addressed to its core function so it acknowledges the packet and processes it accordingly. Proceed to step 7 and wait for another SMBus packet.
12. The device detected a packet addressed to the SMBus Device Default Address. It must check the command field to determine if this is the "Prepare To ARP" command. If so, then it proceeds to step 13, otherwise it proceeds to step 14.
13. Upon receipt of the "Prepare To ARP" command the device must acknowledge the packet and make sure its Address Resolved flag is clear in order to participate in the ARP process. Proceed to step 7 and wait for another SMBus packet.
14. The device checks the command field to see if the "Reset Device" command was issued. If so, then it proceeds to step 15, otherwise it proceeds to step 16.
15. Upon receipt of the "Reset Device" command the device must acknowledge the packet and make sure its Address Valid (if non-PSA) and Address Resolved flags are cleared. This will allow the ARP Master to re-assign all device addresses without cycling power. Proceed to step 7 and wait for another SMBus packet.
16. The device checks the command field to see if the "Assign Address" command was issued. If so, then it proceeds to step 17, otherwise it proceeds to step 19.
17. Upon receipt of the "Assign Address" command the device must compare its UDID to the one it is receiving. If any byte does not match then it must not acknowledge that byte or subsequent ones. If all bytes in the UDID compare then the device proceeds to step 18, otherwise it must return to step 7 and wait for another SMBus packet.

18. Since the UDID matched, the device must assume the received slave address and update its PSA, if supported. The device must set its Address Valid and Address Resolved flags that means it will no longer respond to the "Get UDID" command unless it receives the "Prepare To ARP" or "Reset Device" commands or is power cycled. Proceed to step 7 and wait for another SMBus packet.
19. The device checks the command field to see if the "Get UDID" command was issued. If so, then it proceeds to step 21, otherwise it proceeds to step 20.
20. The device may be receiving a directed command. Directed commands must only be acknowledged by slaves with a valid address. If the address is not valid then ignore the packet and return to step 7 and wait for another SMBus packet. If the address is valid then proceed to step 26.
21. Upon receipt of the "Get UDID" command the device must check its Address Resolved flag to determine whether or not it should participate in the ARP process. If set then its address has already been resolved by the ARP Master so the device proceeds to step 7 to wait for another SMBus packet. If the AR flag is cleared then the device proceeds to step 22.
22. The device returns its UDID and monitors the SMBus data line for collisions. If a collision is detected at any time the device must stop transmitting and proceed to step 7 and wait for another SMBus packet. If no collisions were detected then proceed to step 23.
23. The device must now check its Address Valid flag to determine what value to return for the Device Slave Address field. If the AV flag is set then it proceeds to step 24, otherwise it proceeds to step 25.
24. The current slave address is valid so the device returns this for the Device Slave Address field (with bit 0 set) and monitors the SMBus data line for collisions (i.e. another device driving a "0" when this device is "driving" a "1." Proceed to step 7 and wait for another SMBus packet.
25. The current slave address is invalid so the device returns a value of 0xFF and monitors the SMBus data line for collisions. If the ARP Master receives the 0xFF value it will know that the device requires address assignment. Proceed to step 7 and wait for another SMBus packet.
26. Is this a directed "Reset Device" command? If so then proceed to step 15. Otherwise proceed to step 27.
27. Is this the "Directed Get UDID" command? If so, then proceed to step 29. Return the UDID information. If not, then proceed to step 28
28. The device has not received a valid command so it must handle the illegal command in accordance with SMBus rules for error handling. Proceed to 7 and wait for another SMBus packet.
29. Return the UDID information and current slave address, then proceed to 7 and wait for another SMBus packet.

The flow diagram does not cover consideration for bus timeout mechanisms. These should be implemented in order to comply with the bus timing specifications. If the device supports the "Notify ARP Master" command it may wish to consider implementing a timeout mechanism. This mechanism could cause the device to re-issue the "Notify ARP Master" command if the ARP Master doesn't respond within a particular time period.

The device decodes the two internal flags as described in the following table:

Address Valid (AV Flag)	Address Resolved (AR Flag)	Meaning
Cleared	Cleared	The device does not have a valid slave address and will participate in the ARP process. This is the POR state for a device that doesn't support the PSA or if it does it has not previously been assigned a slave address.
Cleared	Set	Illegal state!
Set	Cleared	The device has a valid slave address but must still participate in the ARP process.
Set	Set	The device has a valid slave address that has been resolved by the ARP Master. The device will not respond to the "General Get UDID" command. However, it could subsequently receive an "Assign Address" command and would change its slave address accordingly.

Table 8: Device decodes of AV and AR flags

5.6.3.13. Enumeration rules

Any device may enumerate the bus provided that the device is intelligent and capable of doing so. Additionally, the enumerating device must provide "snooping" capabilities to guarantee that multiple devices aren't enumerating/ARPing at the same time. An enumerator must adhere to the following rules:

1. If an enumerator sees a "Prepare to ARP" or "Get UDID" command it must immediately stop enumerating.
2. An enumerator must monitor the bus for ARP commands for at least 2 seconds before beginning the enumeration process with the "Prepare to ARP" command.
3. If an enumerator sees an unassigned address then it must issue a host notify command and stop enumerating.

5.6.3.14. Example scenarios

The ARP can be illustrated by the following examples. Note that the UDID values are simple examples for illustrative purposes. They do not necessarily represent legal values.

Example 1

In this scenario assume the following:

- The ARP Master has already exited its reset state and has run the ARP. The Used Address Pool does NOT contain the values 1001 000, 1001 001, … 1001 111.
- New Device A has a UDID of 0x8123456789ABCDEF0000000000000000, does support the Persistent Slave Address and was previously assigned the slave address 1001 001. Its Address Valid flag is set but its Address Resolved flag is cleared.
- New Device B has a UDID of 0xF123456789ABCDE00000000000000000 and does not support the Persistent Slave Address. Its Address Valid and Address Resolved flags are cleared.
- New Device C has a UDID of 0xF123456789ABCDE10000000000000000 and does not support the Persistent Slave Address. Its Address Valid and Address Resolved flags are cleared.
- All devices exit their power on reset state at the same time.

The ARP will proceed as follows:

1. When the devices exit their power on reset state they will optionally attempt to issue the "Notify ARP Master" command. Assume for this example that all three devices do so.
2. All devices will transmit all bytes of the "Notify ARP Master" command without collision.
3. The ARP Master having received the "Notify ARP Master" command will issue the "Get UDID" command.
4. After detecting the repeat Start condition and receiving the SMBus Device Default Address with the R/W# bit set the devices will transmit the byte count for this command without collision. The devices will then begin to transmit their UDIDs as follows:

Device A: 1000…	1	0	
Device B: 1111…	1	1	
Device C: 1111…	1	1	
Seen on the bus:	1	0	Device A wins the bus arbitration

Device A wins the bus arbitration since it is transmitting a "0" as the 2nd msb in the most significant byte of the UDID whereas Devices B & C are transmitting "1." As such Devices B & C will cease transmission of this packet. Device A will complete its transmission.

1. The ARP Master will send the "Assign Address" command using Device A's slave address and UDID. Device A will then set its Address Resolved flag (the Address Valid flag was already set). Device A will no longer respond to the "Get UDID" command until it receives the "Prepare To ARP" or "Reset Device" commands or is power cycled.
2. The ARP Master will add slave address 1001 001 to the Used Address Pool since Device A acknowledged the packet.
3. The ARP Master will issue the "Get UDID" command again.
4. Devices B & C having lost the previous arbitration will respond and will transmit the byte count for this command without collision.
5. The devices will begin to transmit their UDIDs. Since the UDIDs are equal through the first seven most significant bytes there will be no bus collisions. The eighth UDID byte will be transmitted as follows:

Device B: 1110 0000	1	1	1	0	0	0	0	0	
Device C: 1110 0001	1	1	1	0	0	0	0	1	
Seen on the bus:	1	1	1	0	0	0	0	0	Device B wins the bus arbitration

Device B wins the bus arbitration since it is transmitting a "0" as the last bit in the eighth byte of the UDID whereas Device C is transmitting "1." As such Device C will cease transmission of this packet. Device B will complete its transmission by sending the remaining eight bytes of the UDID and an 0xFF for the Device Slave Address field since its Address Valid bit is cleared.

1. Device B will await an assigned address since its Address Valid flag is cleared.
2. The ARP Master recognizes that the returned slave address field was 0xFF. It captures the returned UDID and selects an address (e.g. 1001 000) not in the Used Address Pool and issues the "Assign Address" command.

3. All devices will monitor the "Assign Address" command looking for a UDID match. Since Device B will match its UDID it will acknowledge the packet and adopt the slave address assigned by the ARP Master. Device B will also set its internal Address Resolved and Address Valid flags and will no longer respond to the "Get UDID" command until it receives the "Prepare To ARP" or "Reset Device" commands or is power cycled.
4. The ARP Master will add slave address 1001 000 to the Used Address Pool since Device B acknowledged the packet.
5. The ARP Master will issue the "Get UDID" command again.
6. Device C having lost the previous arbitration will respond and will transmit the byte count for this command without collision. Since it is now the only device responding all remaining bytes will be transmitted without collision.
7. Device C will await an assigned address since its Address Valid flag is cleared.
8. The ARP Master recognizes that the returned slave address field was 0xFF. It captures the returned UDID and selects an address (e.g. 1001 010) not in the Used Address Pool and issues the "Assign Address" command.
9. All devices will monitor the "Assign Address" command looking for a UDID match. Since Device C will match its UDID it will acknowledge the packet and adopt the slave address assigned by the ARP Master. Device C will also set its internal Address Resolved and Address Valid flags and will no longer respond to the "Get UDID" command until it receives the "Prepare To ARP" or "Reset Device" commands or is power cycled.
10. The ARP Master will add slave address 1001 010 to the Used Address Pool since Device C acknowledged the packet.
11. The ARP Master will issue the "Get UDID" command again.
12. Since all three devices have their internal Address Resolved flag set they will not respond.
13. The ARP Master will detect that no device has acknowledged the packet and will terminate the ARP.

Example 2

In this scenario assume the following:

- The system is in the S5 state.
- The system does not contain any devices with slave address values 1001 000, 1001 001, … 1001 111.
- Device A has a UDID of 0x0123456789ABCDEF0000000000000000, does support the Persistent Slave Address and was previously assigned the slave address 1001 001. Device A was present in the system before it entered the S5 state. Its Address Valid flag is set but its Address Resolved flag is cleared.
- New Device B has a UDID of 0xFEDCBA98765432100000000000000000, does support the Persistent Slave Address and was previously assigned the slave address 1001 001 when present in another system. Device B was added to the system while it was in the S5 state. Its Address Valid flag is set but its Address Resolved flag is cleared.
- Both devices exit their power on reset state at the same time.

The ARP will proceed as follows:

1. The system transitions to the S0 state (assume the user pressed the power button).
2. The ARP Master will exit the reset state and will initialize its Used Address Pool.
3. When the devices exit their power on reset state they may attempt to issue the "Notify ARP Master" command; assume that they do for this example. The ARP Master will attempt to issue the "Prepare To ARP" command.

4. If the ARP Master receives the "Notify ARP Master" command before it can issue the "Prepare To ARP" command it will simply ignore it.
5. The ARP Master will issue the "Get UDID" command since presumably the "Prepare To ARP" command was acknowledged.
6. After detecting the repeat Start condition and receiving the SMBus Device Default Address with the R/W# bit set the devices will transmit the byte count for this command without collision.
7. The devices will begin to transmit the most significant byte of their UDID as follows:

Device A: 0000 0001	0	
Device B: 1111 1110	1	
Seen on the bus:	0	Device A wins the bus arbitration

Device A wins the bus arbitration since it is transmitting a "0" as the msb in the most significant byte of the UDID whereas Device B is transmitting "1." As such Device B will cease transmission of this packet. Device A will complete its transmission.

1. The ARP Master will send the "Assign Address" command using Device A's slave address and UDID. Device A will then set its Address Resolved flag (the Address Valid flag was already set). Device A will no longer respond to the "Get UDID" command until it receives the "Prepare To ARP" or "Reset Device" commands or is power cycled.
2. The ARP Master will add slave address 1001 001 to the Used Address Pool since Device A acknowledged the packet.
3. The ARP Master will issue the "Get UDID" command again.
4. Device B having lost the previous arbitration will respond and will transmit the byte count for this command without collision. Since it is now the only device responding all remaining bytes will be transmitted without collision
5. The ARP Master recognizes that the returned slave address was already in the Used Address Pool. It captures the returned UDID and selects an address (e.g. 1001 000) not in the Used Address Pool and issues the "Assign Address" command.
6. All devices will monitor the "Assign Address" command looking for a UDID match. Since Device B will match its UDID it will acknowledge the packet and adopt the new slave address assigned by the ARP Master. Device B will keep its internal Address Valid flag set and will set its Address Resolved flag so that it will no longer respond to the "Get UDID" command until it receives the "Prepare To ARP" or "Reset Device" commands or is power cycled.
7. The ARP Master will add slave address 1001 000 to the Used Address Pool since Device B acknowledged the packet.
8. The ARP Master will issue the "Get UDID" command again.
9. Since both devices have their internal Address Resolved flag set they will not respond.
10. The ARP Master will detect that no device has acknowledged the packet and will terminate the ARP.

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 SMBDAT 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.

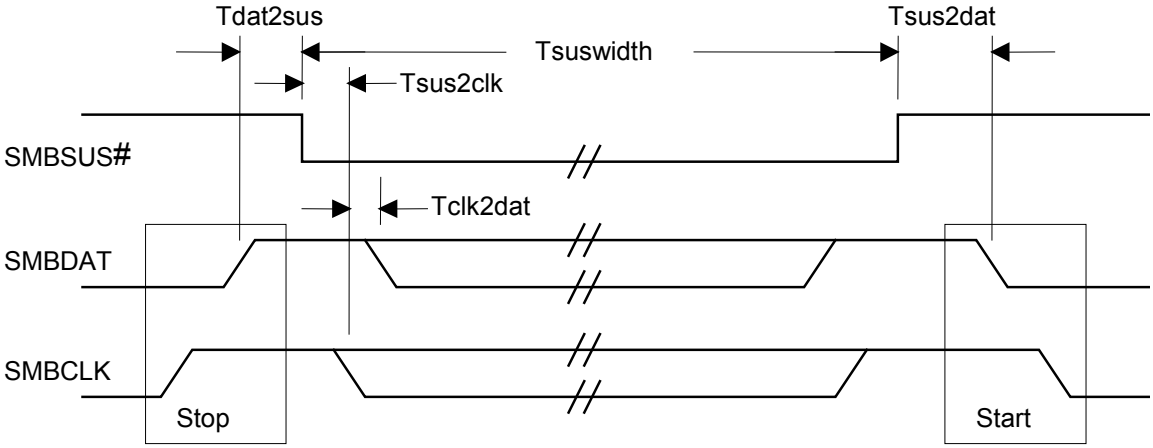


Figure A- 1: SMBus During Suspend

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

Table 9: SMBus suspend parameters

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 SMBDAT pull-ups 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 the suspend state.

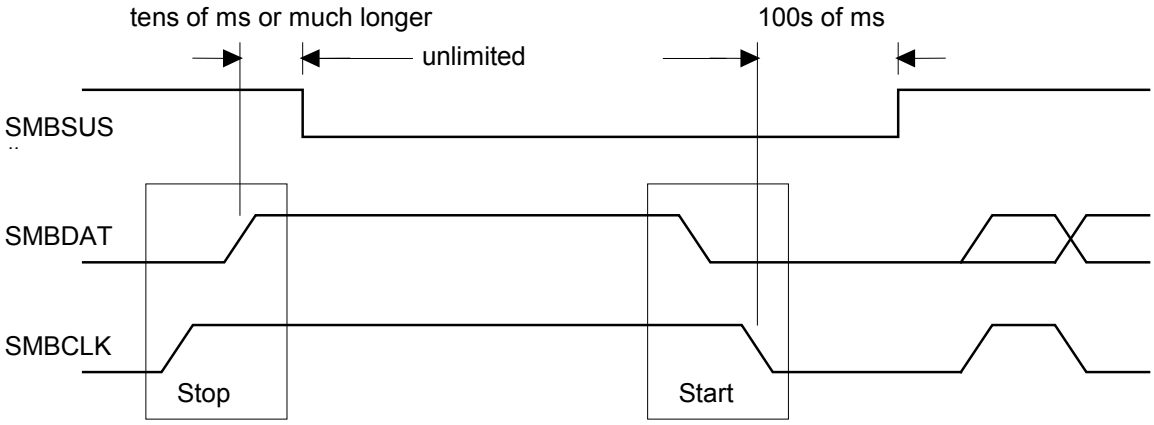


Figure A- 2: Using SMBus to Resume from Suspend

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 SMBDAT 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 that 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 SMBDAT 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.

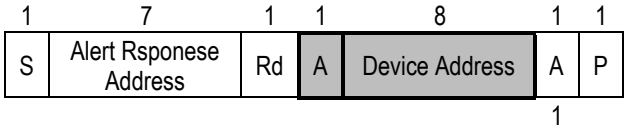


Figure A- 3: A 7-bit-Addressable Device Responds to an ARA

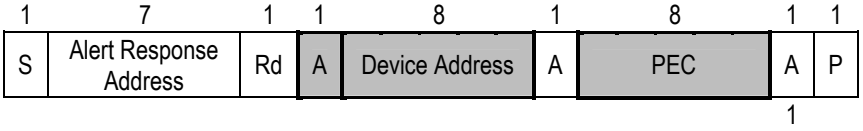


Figure A- 4: A 7-bit-Addressable Device Responds to an ARA with PEC

If more than one device pulls SMBALERT# low, the highest priority (lowest address) device will win communication rights via standard 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 – Differences between SMBus and I^2C

While SMBus is derived from I^2C, there are several major differences between the specifications of the two busses in the areas of electricals, timing, protocols and operating modes.

DC specifications for SMBus and I2C

Both I^2C and SMBus are capable of operating with mixed devices that have either fixed input levels (such as Smart Batteries) or input levels related to V_{DD}. When mixing devices, the I^2C specification defines the V_{DD} to be 5.0 Volt +/- 10% and the fixed input levels to be 1.5 and 3.0 Volts.

Instead of relating the bus input levels to V_{DD}, SMBus defines them to be fixed at 0.8 and 2.1 Volts. This SMBus specification allows for bus implementations with V_{DD} ranging from 3 to 5 Volts +/- 10%. SMBus has relaxed 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 higher versions of SMBus.

A second difference in the DC parameters between I^2C and SMBus is in the power consumption of the low-power version of the bus. SMBus low-power electricals are 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 I^2C. One of the main differences is the IOL specification for VOL = 0.4 Volts. SMBus low-power devices are required to sink a minimum of 100 uA as opposed to 3mA specified for I^2C devices for the same VOL.

A third difference is in the specification of the maximum leakage current for each device connected to the bus. I^2C 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 I^2C and SMBus.

DC parameter comparison between Standard I^2C, Fast I^2C and SMBus devices								
Symbol	**Parameter**	**Std I^2C mode device**		**Fast I^2C mode device**		**SMBus device**		**Units**
		MIN	**MAX**	**MIN**	**MAX**	**MIN**	**MAX**	
V_{IL}	Fixed input level	-0.5	1.5	-0.5	1.5	-	0.8	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

Table 10: DC parameter differences between SMBus and I2C

Timing specification differences between SMBus and I²C

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.

SMBus defines a minimum bus clock frequency FSMB 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.

SMBus defines a data hold time, the time during which SMBDAT must remain valid from the falling edge of SMBCLK, of 300 nS. I²C defines this hold time as zero.

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.

SMBus defines a clock low time-out, TTIMEOUT of 35 ms. I²C does not specify any timeout limit.

SMBus specifies TLOW: SEXT as the cumulative clock low extend time for a slave device. I²C does not have a similar specification.

SMBus specifies TLOW: MEXT as the cumulative clock low extend time for a master device. Again I²C does not have a similar specification.

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 March 31, 1999.

Slave Address	Description	Specification
0001 000	SMBus Host	System Management Bus Specification[1] v 1.1 December 1998
0001 001	Smart Battery Charger	Smart Battery Charger Specification[1] v 1.1 December 1998
0001 010	Smart Battery Selector Smart Battery System Manager	Smart Battery Selector Specification[1] v 1.1 December 1998 Smart Battery System Manager Specification[1] v 1.0B August 1999
0001 011	Smart Battery	Smart Battery Data Specification[1] v 1.1 December 1998
0001 100	SMBus Alert Response	System Management Bus Specification[1] v 1.1 December 1998
0101 000	ACCESS.bus host	
0101 100	Reserved by previous versions of the SMBus specification for LCD Contrast Controller. This address may be reassigned in future versions of the SMBus specification.	
0101 101	Reserved by previous versions of the SMBus specification for CCFL Backlight Driver. This address may be reassigned in future versions of the SMBus specification.	
0110 111	ACCESS.bus default address	
1000 0XX	Reserved by previous versions of the SMBus specification for PCMCIA Socket Controllers (4 addresses) . These addresses may be reassigned in future versions of the SMBus specification.	
1000 100	Reserved by previous versions of the SMBus specification for (VGA) Graphics Controller. This address may be reassigned in future versions of the SMBus specification.	
1001 0XX	Unrestricted addresses	System Management Bus Specification[1] v 1.1 December 1998
1100 001	SMBus Device Default Address	System Management Bus Specification[1] v 1.1 December 1998

Table 11: Assigned SMBus addresses

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