30-Aug-2012

MIPI® Alliance Specification for System Power Management Interface (SPMI)

Version 2.0 – 14 March 2012

* NOTE TO IMPLEMENTERS *

This document is a Draft MIPI Specification. MIPI member companies' rights and obligations apply to this Draft MIPI Specification as defined in the MIPI Membership Agreement and MIPI Bylaws.

An updated Frequently Asked Questions (FAQ) document has been prepared by the SPM Working Group for SPMI v2.0 Specification implementers. Also, a SPMI v2.0 Protocol Implementation Conformance Statement (PICS) document has been prepared by the SPM Working Group.

The SPMI v2.0 Specification contains the following major changes with respect to SPMIv1.0:

- Unified memory map for Slave Device.
- Command acknowledgement (ACK/NACK) for Write commands where required.
- System behavior upon detection of a parity error in an RCS Device-initiated Command Frame.

Backwards Compatibility

- SPMI v2.0 devices are not compatible with SPMI v1.0 Master and Request Capable Slave (RCS) devices.
- SPMI v2.0 Master devices are compatible with SPMI v1.0 Non-Request Capable Slave (NRCS) devices, if the Master ignores the value present on SDATA during the ACK/NACK cycle.

MIPI Alliance Specification for System Power Management Interface (SPMI)

Version 2.0 – 14 March 2012

MIPI Board Approved 28-Aug-2012

Further technical changes to this document are expected as work continues in the SPM Working Group.

NOTICE OF DISCLAIMER

The material contained herein is not a license, either expressly or impliedly, to any IPR owned or controlled by any of the authors or developers of this material or MIPI®. The material contained herein is provided on an "AS IS" basis and to the maximum extent permitted by applicable law, this material is provided AS IS AND WITH ALL FAULTS, and the authors and developers of this material and MIPI hereby disclaim all other warranties and conditions, either express, implied or statutory, including, but not limited to, any (if any) implied warranties, duties or conditions of merchantability, of fitness for a particular purpose, of accuracy or completeness of responses, of results, of workmanlike effort, of lack of viruses, and of lack of negligence.

All materials contained herein are protected by copyright laws, and may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of MIPI Alliance. MIPI, MIPI Alliance and the dotted rainbow arch and all related trademarks, tradenames, and other intellectual property are the exclusive property of MIPI Alliance and cannot be used without its express prior written permission.

ALSO, THERE IS NO WARRANTY OF CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THIS MATERIAL OR THE CONTENTS OF THIS DOCUMENT. IN NO EVENT WILL ANY AUTHOR OR DEVELOPER OF THIS MATERIAL OR THE CONTENTS OF THIS DOCUMENT OR MIPI BE LIABLE TO ANY OTHER PARTY FOR THE COST OF PROCURING SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA, OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, ARISING IN ANY WAY OUT OF THIS OR ANY OTHER AGREEMENT, SPECIFICATION OR DOCUMENT RELATING TO THIS MATERIAL, WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Without limiting the generality of this Disclaimer stated above, the user of the contents of this Document is further notified that MIPI: (a) does not evaluate, test or verify the accuracy, soundness or credibility of the contents of this Document; (b) does not monitor or enforce compliance with the contents of this Document; and (c) does not certify, test, or in any manner investigate products or services or any claims of compliance with the contents of this Document. The use or implementation of the contents of this Document may involve or require the use of intellectual property rights ("IPR") including (but not limited to) patents, patent applications, or copyrights owned by one or more parties, whether or not Members of MIPI. MIPI does not make any search or investigation for IPR, nor does MIPI require or request the disclosure of any IPR or claims of IPR as respects the contents of this Document or otherwise.

Questions pertaining to this document, or the terms or conditions of its provision, should be addressed to:

MIPI Alliance, Inc.
c/o IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08854
Attn: Board Secretary

Contents

161

Figures

Tables

Release History

Date	Release	Description
2008-10-27	v1.00.00	Initial MIPI Alliance Board-approved release.
2012-08-30	v2.0	Board-approved release.

MIPI Alliance Specification for SPMI

1 Introduction

The complexity and performance requirements of mobile phones and other portable electronic devices are increasing at an ever-accelerating rate. As the demand for new high performance, high data rate features increases system level power management is becomes more critical. The use of advanced power management techniques to reduce power consumption and improve battery life is becoming more important than ever before.

Reducing power consumption of digital processors in portable electronic devices can improve the battery life and increase the available power budget for features such as color screens and backlights. These are standard features on portable devices such as wireless handsets, handheld gaming consoles and portable media players.

To minimize power consumption of digital processors in portable electronic devices, system and IC designers are now using advanced power management techniques. Advanced hardware and software techniques are now being used to:

- Accurately monitor and control processor performance level required for a given workload or application
- Control various supply voltages based on the performance level

Rapid deployment of such advanced power management techniques requires interface standardization. This System Power Management Interface (SPMI) Specification addresses hardware interface standardization.

1.1 Scope

This document describes the low-level protocol, communication sequences and arbitration process including device IDs used to implement SPMI. In addition, the bus topology, I/O structures/physical layer and signal timing requirements are also within the scope of this document. Higher level protocols, software driver design and specific device-implementation details are out of scope of this document.

1.2 Purpose

The purpose of this document is to specify a standard interface between baseband or application processors and peripheral components. The SPMI reduces the time-to-market and design cost of mobile terminals by simplifying the interconnection of products from different manufacturers.

2 Terminology

The MIPI Alliance has adopted Section 13.1 of the *IEEE Standards Style Manual*, which dictates use of the words “shall”, “should”, “may”, and “can” in the development of documentation, as follows:

> The word *shall* is used to indicate mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).
>
> The use of the word *must* is deprecated and shall not be used when stating mandatory requirements; *must* is used only to describe unavoidable situations.
>
> The use of the word *will* is deprecated and shall not be used when stating mandatory requirements; *will* is only used in statements of fact.
>
> The word *should* is used to indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain course of action is deprecated but not prohibited (*should* equals *is recommended that*).
>
> The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted*).
>
> The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

All sections are normative, unless they are explicitly indicated to be informative.

Numbers are decimal unless otherwise indicated. A prefix of 0x indicates a hexadecimal number, while a prefix of 0b indicates a binary number.

This document uses the C/Verilog representation for operators where bitwise AND is represented by ‘&’, bitwise OR is represented by ‘|’, bitwise XOR is represented by ‘^’ and 1’s complement (negation) is represented by ‘~’. The modulo operator is represented by ‘%’.

2.1 Definitions

Address Frame: A series of nine bits with eight bits representing address information and a single parity bit.

Bus Arbitration: Bus Arbitration is the process of allocating the bus to one device for the purpose of sending a Sequence.

Bus Handover: The transfer of the bus ownership from the present BOM to the next BOM.

Bus Idle: The SPMI bus is idle when both the SCLK and SDATA are at a logic level zero between the end of a Sequence and the beginning of Bus Arbitration.

Bus Owner Master: The Master that sent the most recent Sequence on the bus and provides SCLK for the subsequent Bus Arbitration.

Bus Park Cycle: A single clock cycle that occurs when the SDATA signal control may change between devices during, or at the end of, a Sequence.

Command Frame: A series of thirteen bits with four bits representing a SPMI Master or Slave device address, eight bits representing a SPMI command and a single parity bit.

Command Sequence: Part of a Sequence including the Sequence Start Condition (SSC), Command and Data Frames and Bus Park Cycle.

Connected Master: A Master that has a Master Priority Level and that is able to initiate Command Sequences on an Initialized Bus.

Data Frame: A series of nine bits with eight bits of data and a single parity bit.

Disconnected Master: A Master that does not update its Master Priority Level and does not send or respond to Sequences on the bus.

Group Slave ID: A 4-bit number assigned to one or more Slave devices identifying them on the SPMI bus as a group.

Initialized Bus: The SPMI bus is considered initialized when there is a Bus Owner Master.

Master: A device on the SPMI bus that can drive the SCLK line, participate in multi-Master arbitration, and supports the transmission of all Master-specific Sequences.

Master Arbitration: A phase of Bus Arbitration between Master devices to allocate the SPMI bus to one Master.

Master ID: Unique 2-bit number assigned to a SPMI Master identifying it on the bus.

Master Priority Level: A number assigned to each Master that dictates the order in which bus ownership is granted.

Non-Request Capable Slave: A Slave device that cannot perform Slave Arbitration or send Sequences.

Sequence: A bus transaction on the SPMI bus that begins with Bus Arbitration, contains a SSC, a Command Frame, potentially Data and Address Frames and ends with a Bus Park Cycle. A Sequence may be transmitted by a Master or a Request Capable Slave device.

Slave: A device on the SPMI bus that is not capable of driving the SCLK line, i.e. not a Master.

Slave Arbitration: A phase of Bus Arbitration between Request Capable Slave devices to allocate the SPMI bus to one Request Capable Slave.

Slave ID: A 4-bit number assigned to a SPMI Slave. Can be either Unique Slave ID or Group Slave ID.

Request Capable Slave: A device on the bus that is not a Master, but is capable of requesting and participating in Bus Arbitration and sending Sequences.

Uninitialized Bus: The SPMI bus is considered uninitialized when there is no Bus Owner Master.

Unique Slave ID: Unique 4-bit number assigned to a SPMI Slave identifying it on the bus.

2.2 Abbreviations

High-Z	High impedance
SDATA	SPMI data

SCLK	SPMI clock
SCLKint	Internal serial clock used within a Master device

2.3 Acronyms

A	Alert
BOM	Bus Owner Master
C	Connect
EMI	Electromagnetic Interference
GSID	Group Slave Identifier
IC	Integrated Circuit
I/O	Input/Output
LSB	Least Significant Bit
MID	Master Identifier
MPL	Master Priority Level
MSB	Most Significant Bit
NRCS	Non-Request Capable Slave
PC	Power Controller
PMIC	Power Management Integrated Circuit
RCS	Request Capable Slave
SID	Slave Identifier
SoC	System-on-Chip
SR	Slave Request
SSC	Sequence Start Condition
SPM	System Power Management
SPMI	System Power Management Interface
TBO	Transfer Bus Ownership
USID	Unique Slave Identifier

3 References

[MIPI01] *MIPI Alliance Specification for Device Descriptor Block (DDB*), version 1.0, MIPI Alliance, 30 October 2008.

4 SPMI Overview

The System Power Management Interface (SPMI) is a two-wire serial interface that connects the integrated Power Controller (PC) of a System-on-Chip (SoC) processor system with one or more Power Management Integrated Circuits (PMIC) voltage regulation systems. SPMI enables systems to dynamically adjust the supply and substrate bias voltages of the voltage domains inside the SoC using a single SPMI bus.

Within the PC, the SPMI-related functions are referred to as the "Master". Within a PMIC, the SPMI related functions are referred to as the "Slave". There may be up to four Master devices and up to sixteen Slaves on a single SPMI bus. Multiple SPMI Masters and Slaves can reside on a single IC, on several separate ICs or any combination of the two.

A Slave device that can initiate Sequences on a SPMI bus is called a Request Capable Slave (RCS) device. A Slave device that can not initiate Sequences is called a Non-Request Capable Slave (NRCS) device.

This Specification defines the operating states, the command set, the physical interface, and the protocol for data communication between SPMI devices on a SPMI bus to insure the compatibility of command and data transfers. The SPMI Command Sequence set includes Slave and Master addressing, control of the Slave operating state, register read from and register write to Master and Slave devices, as well as commands supporting the use of MIPI Device Descriptor Block data read and bus management.

4.1 System Overview

The minimum system configuration consists of two SPMI devices one of which shall be a SPMI Master device.

A more complex system may include up to four separate SoCs with up to four SPMI Masters. Up to sixteen logical Slaves may be connected to these Master devices. The Slaves may reside on one or more physical devices. The Slave devices may provide supply voltages to the SoCs or perform other functions controlled via the SPMI. Figure 1 shows an example SPMI system.

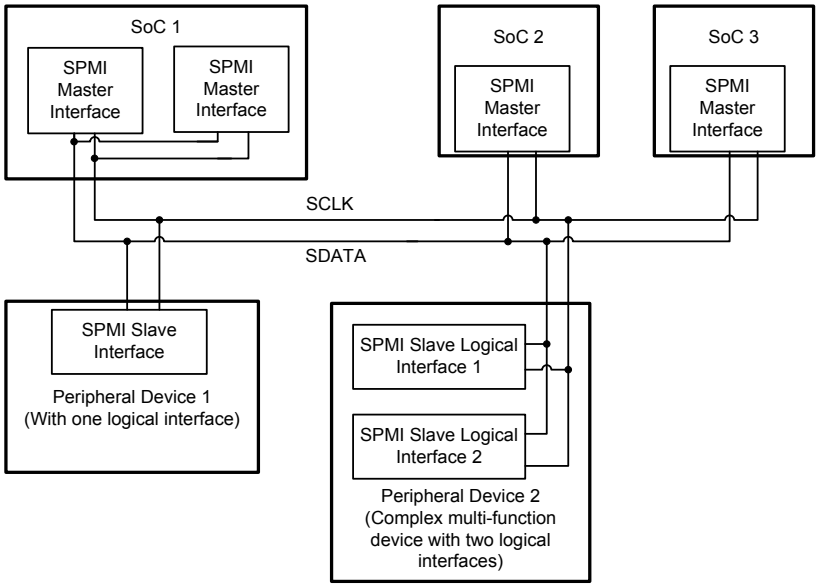


Figure 1 SPMI System Example

4.2 SPMI Protocol Overview

The SPMI protocol has the following features:

- Bus Arbitration –SPMI uses a Bus Arbitration process for allocating traffic on the bus between the Master devices and Request Capable Slave devices. See Section 8.
- Master Connection and Disconnection – A process for a Master to connect to, and disconnect from, an Initialized or an Uninitialized SPMI bus. See Section 9.
- Slave Initiated Communication – A process for a Request Capable Slave to initiate Sequences to Masters or other Slaves. See Section 10.
- ACK/NACK response for robust communication

The SPMI protocol has the following requirements:

- Enumeration – Each Master or Slave device needs a unique identifier to communicate on using SPMI. See Section 7.
- SPMI Master Requirements – A SPMI Master needs to support a minimum set of functions as defined in Section 11 and 13.
- SPMI Slave Requirements – A SPMI Slave needs to support a minimum set of functions as defined in Section 12 and 13.

5 Physical Interface

5.1 I/O Structures

The SPMI is a two-wire interface between SPMI devices that shall have two signals, one serial bi-directional data signal (SDATA) and one clock signal (SCLK) controlled by a SPMI Master. Figure 2 shows the I/O structures required for the data signal for both Master and Slave.

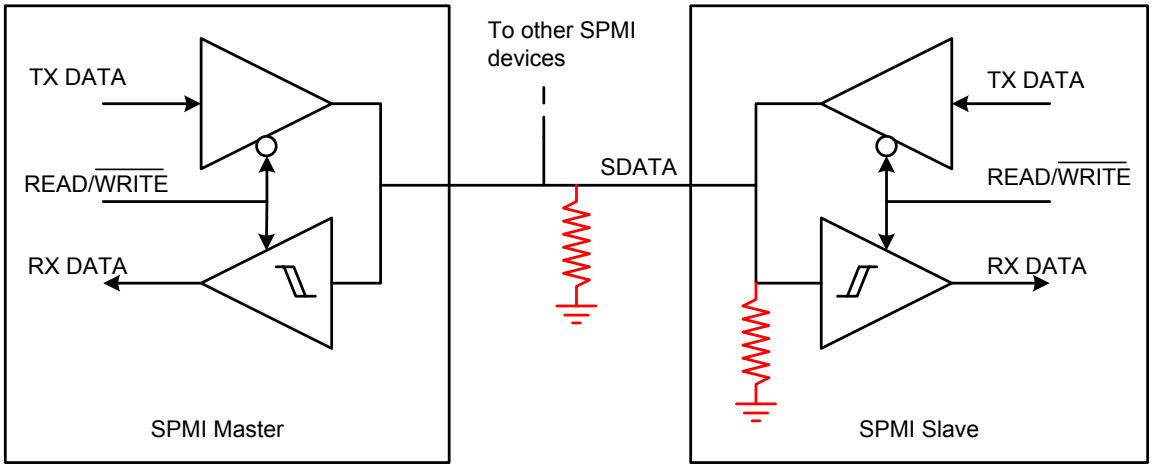


Figure 2 SDATA Master and Slave I/O Cells

Figure 3 shows the I/O structures required for the clock signal for both Master and Slave.

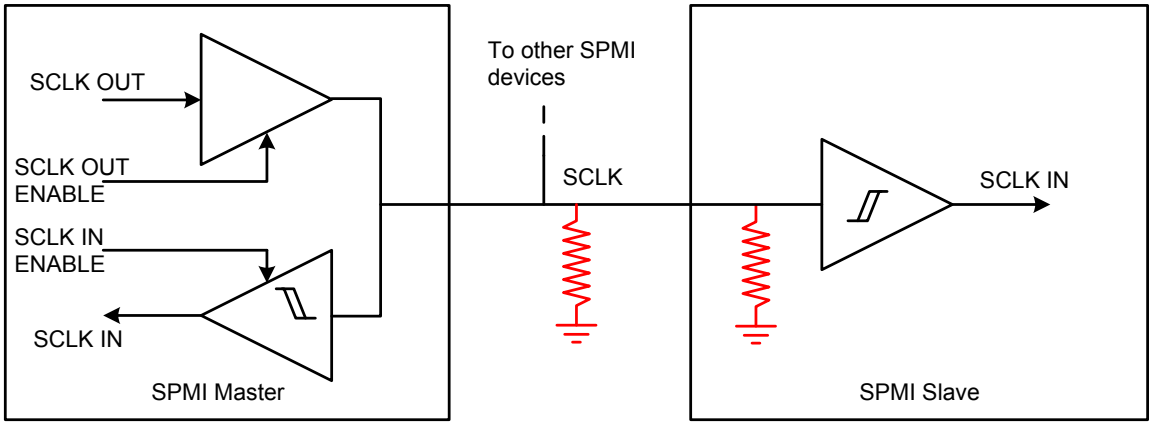


Figure 3 SCLK Master and Slave I/O Cells

SDATA and SCLK pull-down resistors may be implemented using external components instead of integrated resistors. The alternative implementations are shown in red in Figure 2 and Figure 3. A SPMI

Master device does not have pull-down resistors. See Table 1 for pull-down resistor specifications. Devices shall be implemented such that contention (devices driving different levels at the same time) on SCLK or SDATA does not cause irreversible damage to the line driver.

The I/O cells shall be implemented with high impedance input structures and output drivers that are high impedance when not active. I/O cells with typical CMOS structures usually provide these characteristics.

5.1.1 I/O Configuration with Multiple Slaves and Masters

The SPMI Specification supports up to sixteen logical Slave devices on a single or on multiple physical Slave ICs. Each physical Slave device has one SCLK input and one SDATA bidirectional interface. Each physical Slave device can have pull-down resistors in the I/O cells, or these pull-down resistors can be provided externally.

The pull-down resistances of a SPMI Slave and the total system are specified in Table 1. If more than four physical Slave devices are connected to the bus, then the number of individual Slave pull-down resistors shall be adjusted such that the total system pull-down resistance specification is met.

Table 1 SPMI Pull-down Resistance Specifications

	SDATA Pull-down		SCLK Pull-down		
	Min	Max	Min	Max	Units
SPMI Slave	0.5	2.0	0.5	2.0	MΩ
Total System	0.125	2.0	0.125	2.0	MΩ

The different physical Slave devices connect to the bus in parallel. Any logical Slave devices inside a physical Slave device share the common I/O structures of the physical device.

SPMI allows the use of up to four Master devices and up to sixteen Slave devices on a single bus with the use of Bus Arbitration.

5.2 I/O Voltage and Logic Levels

SCLK and SDATA are CMOS-like signals, i.e. single-ended, ground referenced, rail-to-rail, voltage mode signals. Therefore, electrical specifications in this document are given relative to the I/O supply voltage, VDD. The SCLK and SDATA terminals shall use the same signaling levels.

The signal levels of the external control and status signals (ENABLE, RESETN and PWROK), if they are present as external signals on Slave devices, are user defined.

5.2.1 Signaling Voltages

This document uses 1.2 V and 1.8 V signaling to describe various parameters. However, other signaling voltage levels can be used, but are out of scope of this document.

A Component using 1.8 V signaling shall meet the requirements given in Table 2.

Table 2 Parameters for Components using 1.8 V Signaling

Symbol	Description	Min	Max	Units
VDD	SPMI I/O Supply Voltage	1.65	1.95	V
V_{TP}	Positive Going Threshold Voltage	0.4*VDD	0.7*VDD	V

Symbol	Description	Min	Max	Units
V_{TN}	Negative Going Threshold Voltage	0.3*VDD	0.6*VDD	V
V_H	Hysteresis Voltage (V_{TP} - V_{TN})	0.1*VDD	0.4*VDD	V

Additionally, the Component shall have the static output electrical characteristics shown in Table 3.

Table 3 Static Electrical Characteristics for 1.8 V Signaling

Symbol	Description	Condition	Min	Max	Units
V_{OL}	Output Low Voltage	I_{OL} = 2mA	0	0.2*VDD	V
V_{OH}	Output High Voltage	I_{OH} = -2mA	0.8*VDD	VDD	V

A Component using 1.2 V signaling shall meet the requirements given in Table 4.

Table 4 Parameters for Components using 1.2 V Signaling

Symbol	Description	Min	Max	Units
VDD	SPMI I/O Supply Voltage	1.1	1.3	V
V_{TP}	Positive Going Threshold Voltage	0.4*VDD	0.7*VDD	V
V_{TN}	Negative Going Threshold Voltage	0.3*VDD	0.6*VDD	V
V_H	Hysteresis Voltage (V_{TP} - V_{TN})	0.1*VDD	0.4*VDD	V

Additionally, the Component shall have the static output electrical characteristics shown in Table 5.

Table 5 Static Electrical Characteristics for 1.2 V Signaling

Symbol	Description	Condition	Min	Max	Units
V_{OL}	Output Low Voltage	I_{OL} = 2mA	0	0.2*VDD	V
V_{OH}	Output High Voltage	I_{OH} = -2mA	0.8*VDD	VDD	V

5.2.2 SPMI I/O Voltage Supply

All Components on a single SPMI bus instance shall use the same I/O voltage level.

5.3 Device Classes

SPMI supports the concept of device classes by categorizing devices according to the range of supported SCLK frequencies. The two defined SPMI device classes are:

- High Speed (HS): 32.000 kHz to 26 MHz, with load up to 50 pF
- Low Speed (LS): 32.000 kHz to 15 MHz with load up to 50 pF

Physical devices from different device classes may be connected to the same SPMI bus instance. However, the combined bus loading shall not exceed the SPMI-specified value. In addition, the maximum bus clock speed shall not exceed the maximum speed supported by the slowest device on the bus. The slowest nominal bus clock frequency allowed for either device class is 32.000 kHz.

5.4 SPMI Clock (SCLK)

The BOM shall drive the SPMI clock signal. All clock Sequences shall start and end with the SCLK signal at logic level zero. A Slave device shall not drive the SCLK signal. A Slave device can hold the SCLK signal low with its pull-down resistor.

5.4.1 Electrical Specifications for the Master SCLK Driver

SCLK shall not toggle during idle and inactive time periods. SCLK shall only run while data is being transferred on the bus; otherwise SCLK is at logic level zero. The Master device may use an internal SCLK, SCLKint, for monitoring the bus activity in a multi-Master system. The SCLKint frequency is assumed to be the same as the SCLK frequency in this document.

To reduce active power consumption, the SCLK line is not terminated. To avoid reflections and over voltage problems on such a bus system, the SCLK line shall be transition time controlled for high speed operation. This constraint makes a conventional CMOS I/O unsuitable for driving the SCLK line at high speeds. The SCLK line may be driven at low speeds using a carefully matched CMOS driver with or without transition time control.

The bus diameter, i.e. the maximum physical distance between any transmitter and any receiver, is expected to be less than 15 cm. This constraint implies a minimum allowed transition time of 2.1 ns. To allow larger bus diameters, longer transition times or terminated bus topologies may be used, however such bus instances are out of scope for this document.

The minimum allowable transition time on the SCLK line is directly related to the bus diameter of the SCLK line. The longer this bus diameter, the longer the transition time needed to generate a reliable clock signal without reflections and voltage over- and undershoot.

When driving the test load specified in Section 5.4.1.1 the SCLK line driver shall conform to the timing characteristics shown in Figure 4.

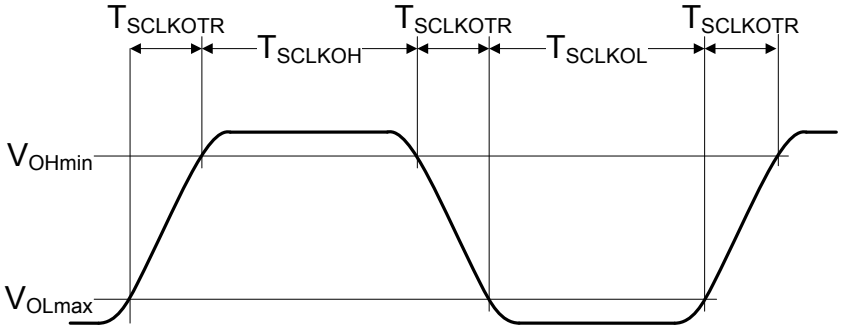


Figure 4 Clock Driver Output Waveform Constraints

Table 6 SCLK Output Timing Characteristics

Symbol	Description	Low Speed Device		High Speed Device		Units
		Min	Max	Min	Max	
T_{SCLKOH}	Clock Output High Time	22		12		ns
T_{SCLKOL}	Clock Output Low Time	22		12		ns
$T_{SCLKOTR}$	Clock Output Transition (Rise / Fall) Time	2.1	8	2.1	5.3	ns

The rise and fall times of the clock driver constrain both the maximum operating frequency and length of the SCLK line.

All timing characteristics are referenced to V_{OH} and V_{OL}.

5.4.1.1 Additional SCLK Driver Timing Requirements

The Bus Handover as shown in Figure 9 is a special bus condition that facilitates the change of bus ownership. The SDATA line is driven to a logic level zero while the SCLK is at a logic level one. The SDATA and SCLK lines are released on the falling edge of SCLK, or before another device assumes control of either line from the device driving it initially. The Bus Handover cycle is used at the end of the Transfer Bus Ownership Sequence.

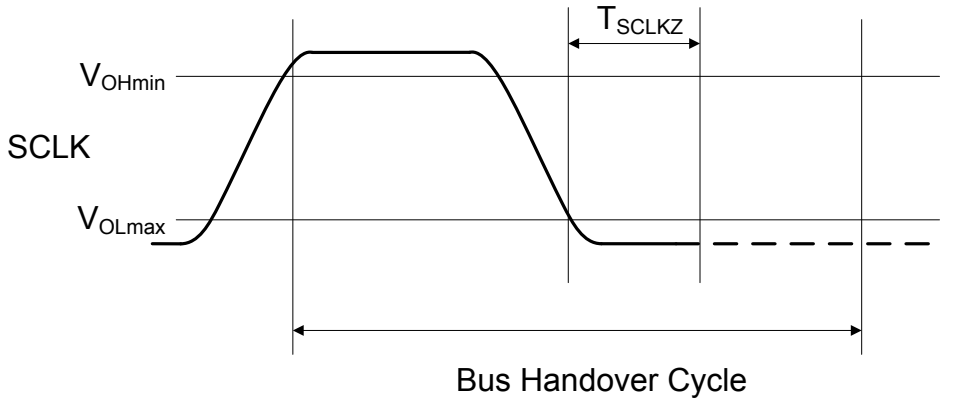


Figure 5 Bus Handover Timing

The SCLK signal shall also conform to the timing characteristics shown in Figure 5 and Table 7.

Timing is referenced to the V_{OH} and V_{OL} levels defined in Section 5.2.1.

The T_{SCLKZ} data signal specification is measured from the falling edge of SCLK.

Table 7 SCLK Bus Handover Timing Parameters

Symbol	Description	Low Speed Device		High Speed Device		Units
		Min	Max	Min	Max	
T_{SCLKZ}	Clock drive release time		18		10	ns

5.4.2 Electrical Specifications for SCLK Input

A SPMI device's set of timing requirements for correct operation shall meet the constraints given in Table 8. Its respective Min limits shall be no higher than the values in the table. A glitch rejection filter may be used on the SCLK input. The clock receiver shall be capable of receiving slowly changing edges without glitching. A SPMI device shall implement hysteretic input on the SCLK pin.

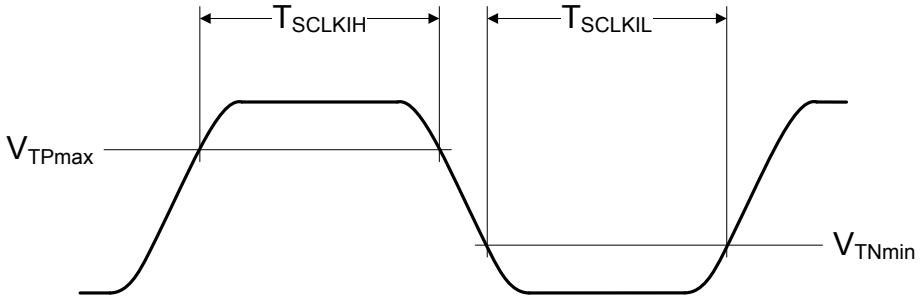


Figure 6 Received Clock Signal Constraints

Timings are referenced to V_{TPmax} and V_{TNmin}.

Table 8 Clock Input Timing Requirements

Symbol	Description	Low Speed Device		High Speed Device		Units
		Min	Max	Min	Max	
T_{SCLKIH}	SCLK Input High Time	22		12		ns
T_{SCLKIL}	SCLK Input Low Time	22		12		ns

5.5 SPMI Data (SDATA)

The SPMI data signal is bi-directional. Data shall be written on the rising edge (transition from logical level zero to logical level one) of the SCLK signal by both Master and Slave devices. Data shall be read on the falling edge (transition from logical level one to logical level zero) of the SCLK signal.

5.5.1 Electrical Specifications for the SDATA Driver

The same signal integrity issues, maximum distance between any transmitter to any receiver device and signal transition time factors, affect the SDATA line as well as the SCLK line as described in Section 5.4. For this reason, the SDATA driver of a High Speed device shall have transition time control to meet transition time specifications listed in Table 9.

The minimum slew time, $T_{SDATAOTRmin}$, is specified for a single device driving the test load described in Section 5.6. The slew time observed on the SDATA line may be smaller than $T_{SDATAOTRmin}$ when multiple devices are simultaneously driving the bus, e.g. during Bus Request.

The output driver of a SDATA terminal shall drive the test loads specified in Section 5.6 with the dynamic specifications shown in Figure 7 and Table 9.

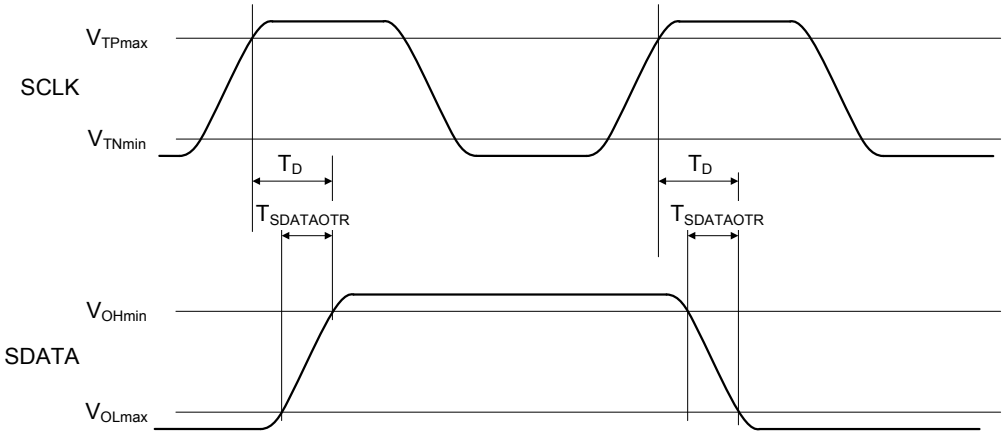
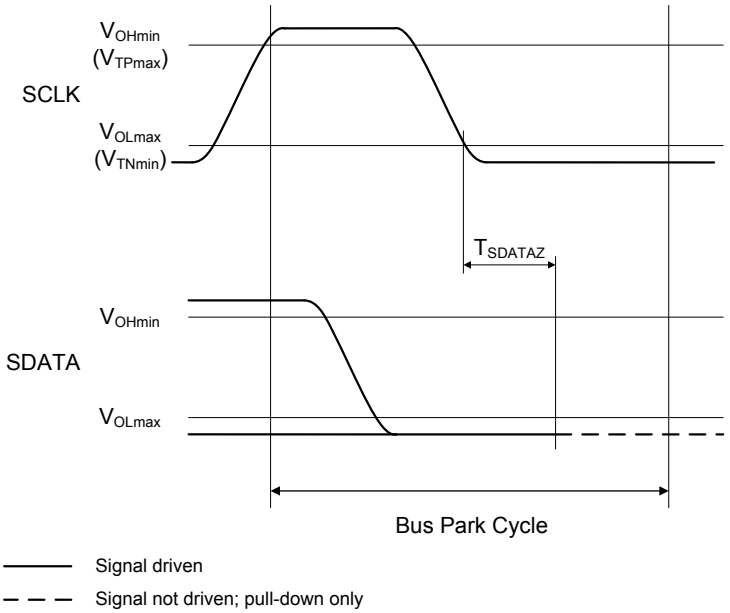


Figure 7 Bus Active Data Transmission Timing Specification

Table 9 SDATA Output Timing Characteristics

Symbol	Description	Low Speed Device		High Speed Device		Units
		Min	Max	Min	Max	
T_D	Time for Data Output Valid from SCLK rising edge	0	20	0	11	ns
$T_{SDATAOTR}$	SDATA Output Transition (Rise/Fall) Time	2.1	8	2.1	5.3	ns

Timing is referenced to V_{TPmax}, V_{OHmin} and V_{OLmax}.



T_{SDATAZ} is measured from SCLK VOL level for the master device driving SCLK and SDATA line
T_{SDATAZ} is measured from SCLK VTN level for a device receiving SCLK and driving SDATA line

Figure 8 Bus Park Cycle Timing

The Bus Park Cycle as shown in Figure 8 is a special bus condition that facilitates the change of SDATA control for bus turnaround purposes. The SDATA line is driven to a logic level zero while SCLK is at a logic level one. The SDATA line is released on the falling edge of SCLK. The Bus Park Cycle is also used at the end of all Sequences with the exception of the Transfer Bus Ownership Sequence.

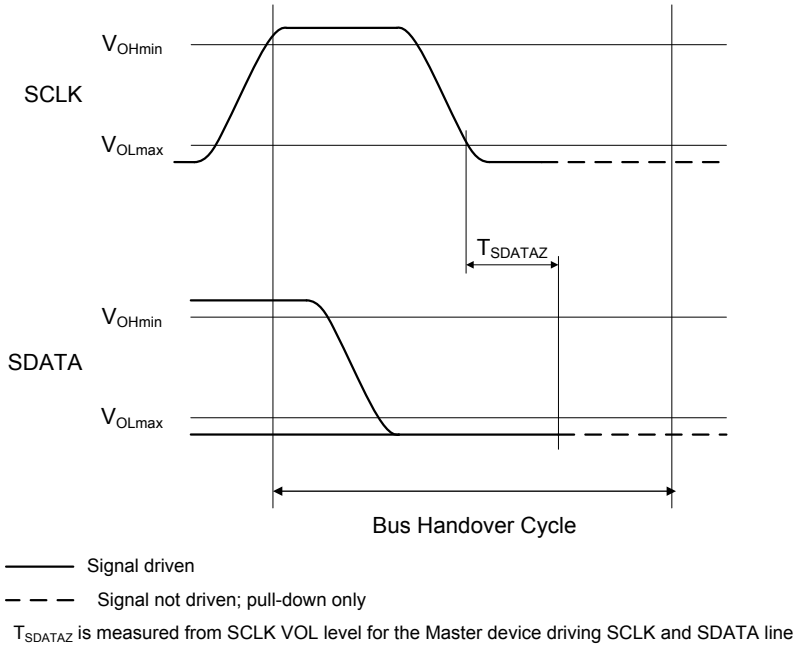


Figure 9 Bus Handover Timing

The Bus Handover as shown in Figure 9 is a special bus condition that facilitates the change of bus ownership. The SDATA line is driven to a logic level zero while the SCLK is at a logic level one. The SDATA and SCLK lines are released on the falling edge of SCLK. The Bus Handover cycle is used at the end of the Transfer Bus Ownership Sequence.

Table 10 SDATA Release Timing Parameters

Symbol	Description	Low Speed Device		High Speed Device		Units
		Min	Max	Min	Max	
T_{SDATAZ}	Data drive release time		18		10	ns

T_{SDATAZ} timing is referenced to V_{OLmax} and V_{TNmin} in Figure 9 and Figure 8, respectively.

T_{SDATAZ} data signal specification is measured from the falling edge of SCLK (either from V_{OLmax} when the device is driving SCLK and SDATA lines, or from V_{TNmin} when the device is receiving SCLK and driving the SDATA line).

5.5.2 Electrical Specifications for the SDATA Receiver

The SDATA receiver may use a glitch rejection filter on the SDATA input. The SDATA receiver shall be capable of receiving slowly changing edges without glitching. A SPMI component shall implement hysteretic input on the SDATA pin. Data receiver timing shall follow the timing of Table 11.

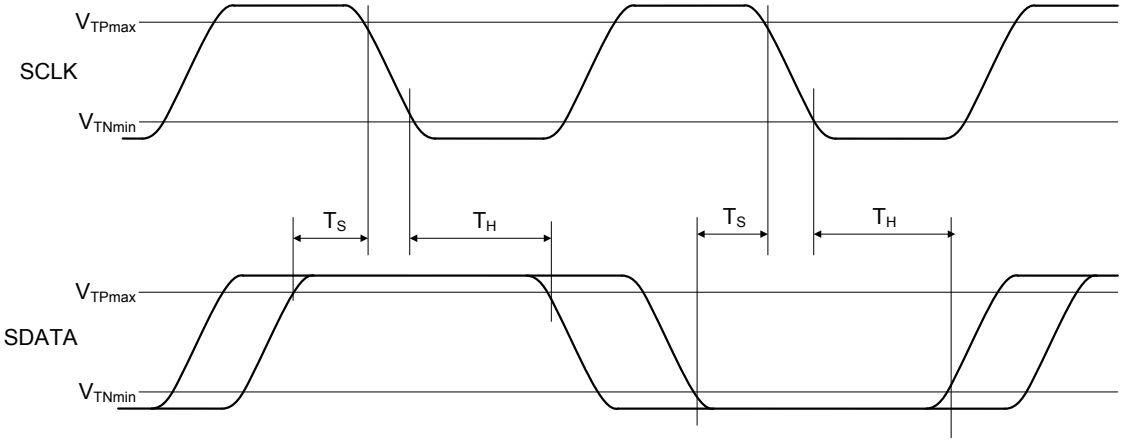


Figure 10 Bus Active Data Receiver Timing Requirements

Table 11 Data Receiver Timing Requirements

Symbol	Parameter	Low Speed Device		High Speed Device		Units
		Min	Max	Min	Max	
T_S	Data setup time	2		1		ns
T_H	Data hold time	5		5		ns

Timings are referenced to V_{TPmax} and V_{TNmin} and are measured at the input of the device.

5.6 Device Characterization

The device electrical characteristics for SCLK and SDATA lines listed in Section 5 shall be guaranteed using the device characterization circuit shown in Figure 11. The characterization trace impedance in the figure is only for characterization purpose, and does not reflect real trace impedance in an application that may be different.

The slew times ($T_{SCLKOTRmin}$ and $T_{SCLKOTRmax}$ for the SCLK line, $T_{SDATAOTRmin}$ and $T_{SDATAOTRmax}$ for the SDATA line), transient voltage, and transition time specifications of a device shall be characterized at both the smallest and largest loads, represented by 15 pF and 50 pF, respectively, through a transmission line of 75 Ω impedance and 15 cm physical length.

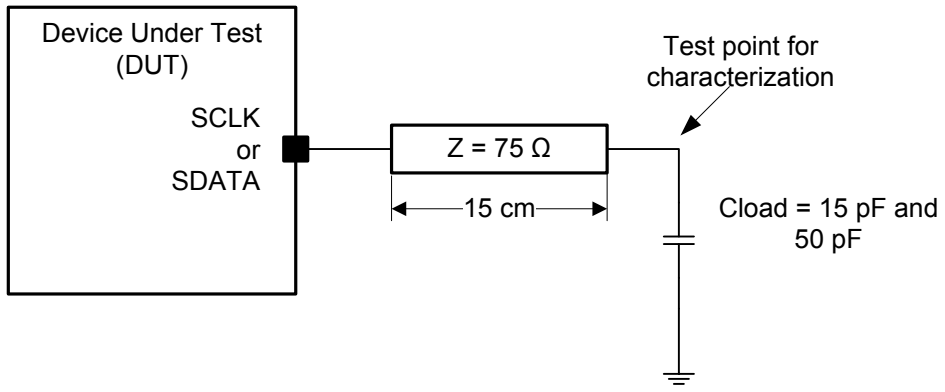


Figure 11 Device Characterization Circuit

5.7 Electromagnetic Interference

Electromagnetic Interference (EMI) should be taken into account when designing or selecting I/O drivers for SPMI implementation. The SPMI Specification does not define EMI requirements. The product developer is responsible for managing EMI characteristics and susceptibility to EMI according to the needs of the application.

6 SPMI Constructs

The SPMI Specification constructs all Sequences on the interface using individual bits. Bits are organized into Frames and Sequences. See Section 6.1 through Section 6.3 for information on Sequence elements and Section 13 for descriptions of the various SPMI Sequences.

6.1 Bit Ordering

Bits shall be sent on the bus MSB first. In all figures, bits and Frames are shown such that both individual bits and Frames are represented, in left-to-right, top-to-bottom reading order, as they would transfer across the interface.

6.2 Sequences

Sequences shall be comprised of the following five events that occur in order:

- Bus Arbitration
- Transmission of the Sequence Start Condition (SSC)
- Transmission of Frames (Command Frame and possibly one or more Data Frames)
- Transmission of ACK/NACK for applicable Command Sequences
- Transmission of a Bus Park Cycle

The last four events, SSC, Frames, ACK/NACK and Bus Park Cycle, form the Command Sequence.

6.2.1 Bus Arbitration

Bus Arbitration is the process where the Bus shall be allocated to one Master or Request Capable Slave among the devices that may simultaneously request to send a Command Sequence on the bus. See Section 8 for details on Bus Arbitration.

6.2.2 Sequence Start Condition

The Sequence Start Condition shall be a unique condition on the bus identified by a rising edge followed by a falling edge on SDATA while SCLK remains at a logic low level. The SSC is used by a Slave or Master to identify the start of a Command Sequence.

The Bus Owner Master shall generate the SSC by driving SDATA to logic level one for one SCLKint period, then to logic level zero for one SCLKint period while holding SCLK at logic level zero as shown in Figure 12.

If a Request Capable Slave device (see Section 8) wants to send the Command Sequence following the SSC then the BOM shall send a Bus Park Cycle (see Section 6.2.6) on the last SCLKint period of the SSC. The Bus Park Cycle releases the SDATA line so the Slave device can send the Command Sequence. If the BOM is sending the Command Sequence there is no need for a Bus Park Cycle as the BOM drives the SDATA line during the Command Frame.

A Command Frame shall start on the next SCLKint rising edge.

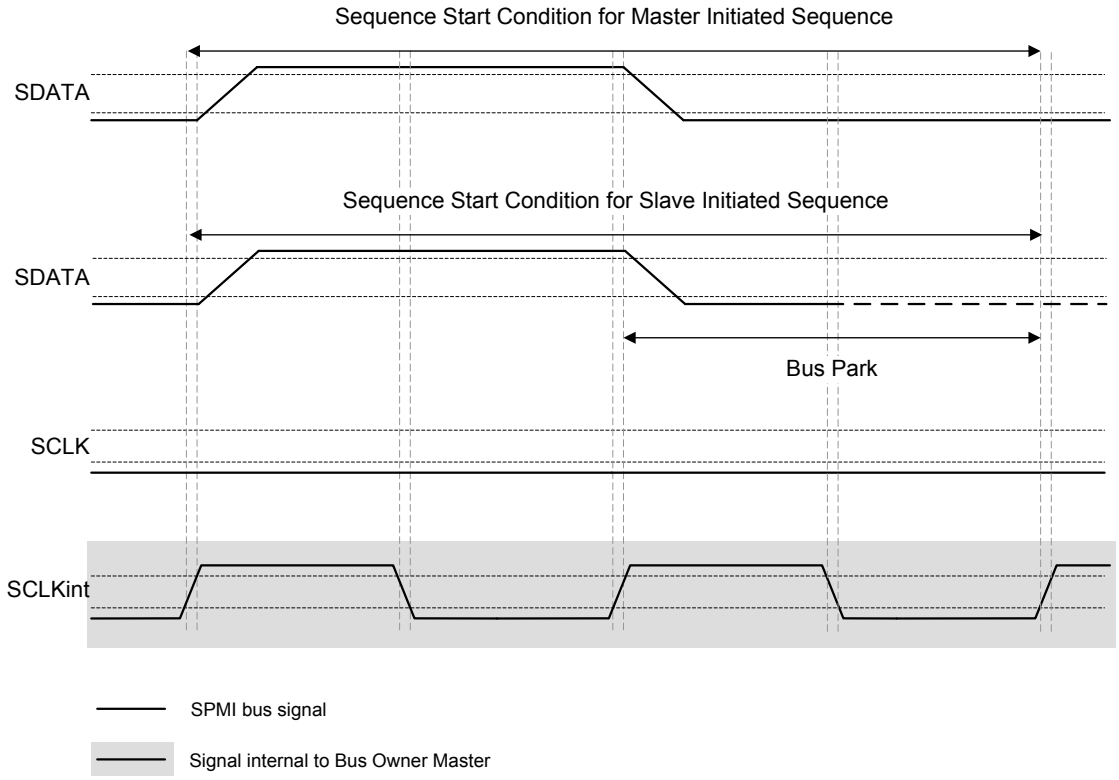


Figure 12 Sequence Start Condition

6.2.3 Frames

There are three basic types of Frames:

- Command Frame, thirteen bits. See Section 6.2.3.1.
- Data or Address Frames, nine bits. See Section 6.2.3.2.
- No Response Frame, nine or thirteen bits. See Section 6.2.3.3.

All Frames consist of data, address or command bits and a parity bit.

6.2.3.1 Command Frame

A Command Frame shall consist of a 4-bit address field, an 8-bit command field, and a single parity bit. The first two bits of the address field are always zero when sending a master address or MID field. The Command Frame structure is shown in Figure 13.

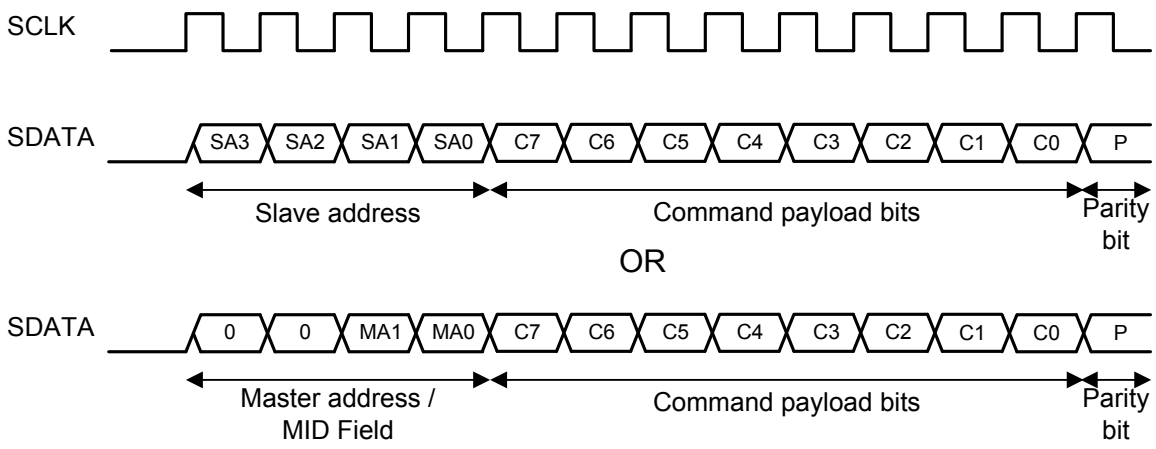


Figure 13 Command Frame

6.2.3.2 Data and Address Frames

A Data or Address Frame shall consist of eight data bits or eight address bits, respectively, and a single parity bit. The Frame structure is shown in Figure 14. The Frame is called an Address Frame when the payload bits carry address information and a Data Frame when the payload bits carry data. The type of data being carried is defined by the position of the Frame within the Sequence. See Section 13 for more information.

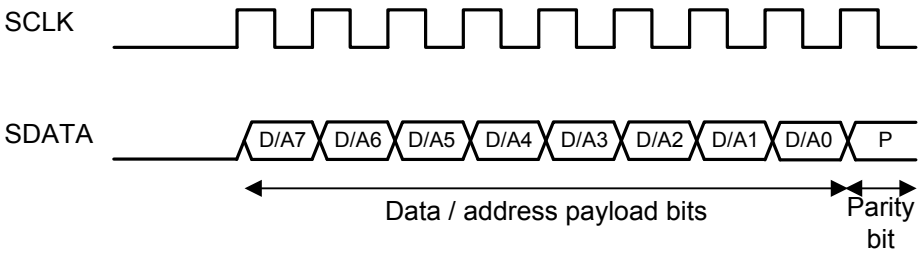
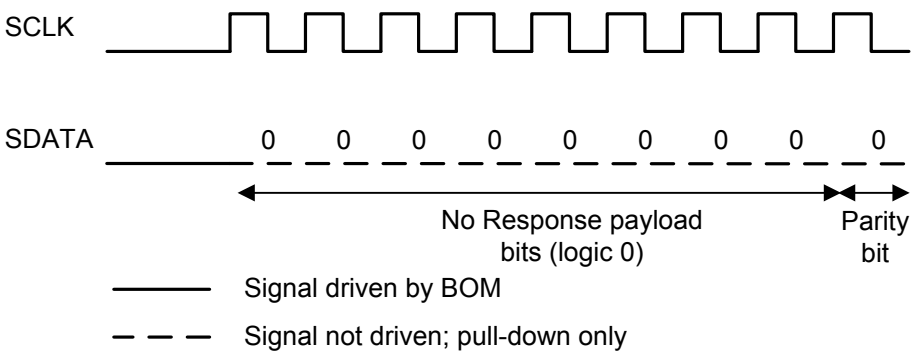


Figure 14 Data Frame

6.2.3.3 No Response Frame

All bits, including the parity bit, of a No Response Frame shall be zero. This Frame may be generated by driving SDATA to logic level zero or passively allowing the pull-down resistor to pull SDATA low for the duration of the Frame. A No Response Frame may be nine bits long if it is a Data Frame or thirteen bits long if it is a Command Frame.



641

642 **Figure 15 No Response Data Frame**

643 ### 6.2.4 Parity Bit

644 A Frame shall end with a single parity bit. The parity bit shall be driven such that the total number of bits in
645 the Frame that are driven to logic level one, including the parity bit, is odd. For example, a Data Frame
646 with data 0x63 (0b01100011) has a '1' parity bit and a Data Frame with data 0x4C (0b01001100) has a '0'
647 parity bit.

648 ### 6.2.5 ACK/NACK

649 ACK/NACK consists of two cycles that are used to confirm correct completion of a Write Command
650 Sequence and certain other Command Sequences. ACK/NACK shall be used on all Command Sequences,
651 except Read Command Sequences, the Authenticate Command Sequence and the Transfer Bus Ownership
652 Command Sequence. ACK/NACK is not used on Command Sequences where the correctness of the
653 Command Sequence can be confirmed based upon Data Frames received by the Device generating the
654 Command Sequence. ACK/NACK shall be generated as described in Section 13.

655 If a Command Sequence that includes ACK/NACK is performed, the Device generating the Command
656 Sequence shall generate a Bus Park Cycle to transfer control of the SDATA line to the accessed Devices.
657 The Bus Park Cycle permits SDATA turnaround in preparation of the ACK/NACK.

658 After the Bus Turnaround Cycle, the BOM shall provide one clock cycle on the SCLK line. A Device
659 controlling the SDATA line shall return the ACK/NACK value on SDATA.

660 If a Command Sequence that includes ACK/NACK is addressed to one Device using USID or MID, the
661 addressed Device shall respond with an ACK/NACK value of 0b1 if the Command Sequence was received
662 correctly, otherwise it shall respond with an ACK/NACK value of 0b0.

663 If a Command Sequence that includes ACK/NACK is addressed to a group of Devices using GSID, an
664 addressed Device shall keep its SDATA driver in high-Z state if the Command Sequence was received
665 correctly. Only if an error was detected shall a Device respond with ACK/NACK value of 0b1. This
666 procedure avoids race conditions between multiple receiving Devices on SDATA.

667 The Device generating the Command Sequence that includes ACK/NACK shall interpret the ACK/NACK
668 value based on whether it used USID/MID or GSID to send the Command Sequence.

669 After the ACK/NACK, a Device in control of SDATA shall perform a Bus Park Cycle as explained in
670 Section 6.2.6. An example of ACK/NACK using the Register Write Command Sequence is shown in
671 Figure 16.

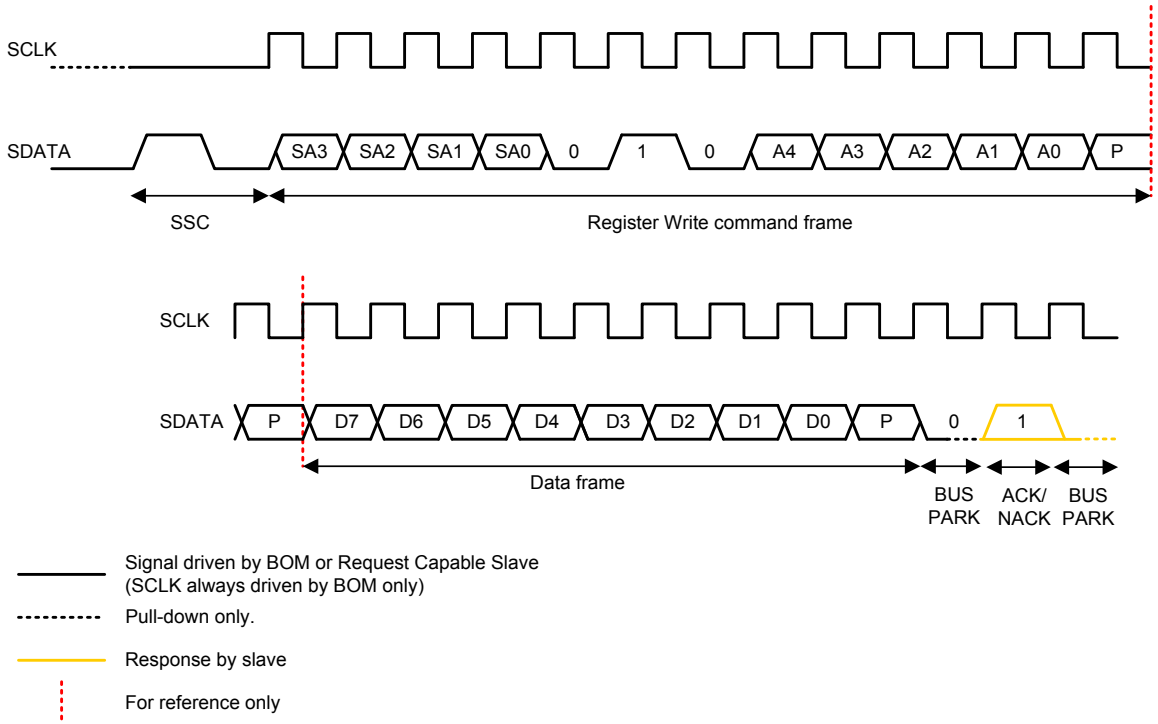


Figure 16 Example ACK/NACK using Register Write Command Sequence

6.2.6 Bus Park Cycle

A Device with ownership of SDATA shall initiate a Bus Park Cycle on SDATA (see Figure 8) at the end of a Sequence, or when the device transfers control of SDATA to another device. The purpose of the Bus Park Cycle is to put the SPMI bus into a known state in preparation for an SDATA signal control change, the start of Bus Arbitration, the start of a Command Sequence, or the start of the bus idle condition.

During the Bus Park Cycle, the device releasing SDATA shall drive the SDATA signal to a logic level zero during the first half of the SCLK clock cycle. Thereafter, the device releasing SDATA shall put its SDATA driver into the high impedance state. The BOM controlling the Sequence shall provide a normal SCLK during the Bus Park Cycle except when the Bus Park Cycle occurs during the SSC (see Section 6.2.2).

6.3 Bus Idle Condition

The SPMI bus is in the “Idle” condition when both SCLK and SDATA signals are at logic level zero and all devices connected to the bus other than the Bus Owner Master (BOM) have their bus driver outputs in a high impedance state. The BOM shall be the only device on the bus that drives the SCLK line, maintaining a strong (low-impedance) low level on that line. A pull-down resistor on SDATA maintains a weak (high impedance) low level on that line. The bus is always in idle condition between the end of a Command Sequence and the beginning of Bus Arbitration. See Figure 17 for an illustration of the bus idle condition.

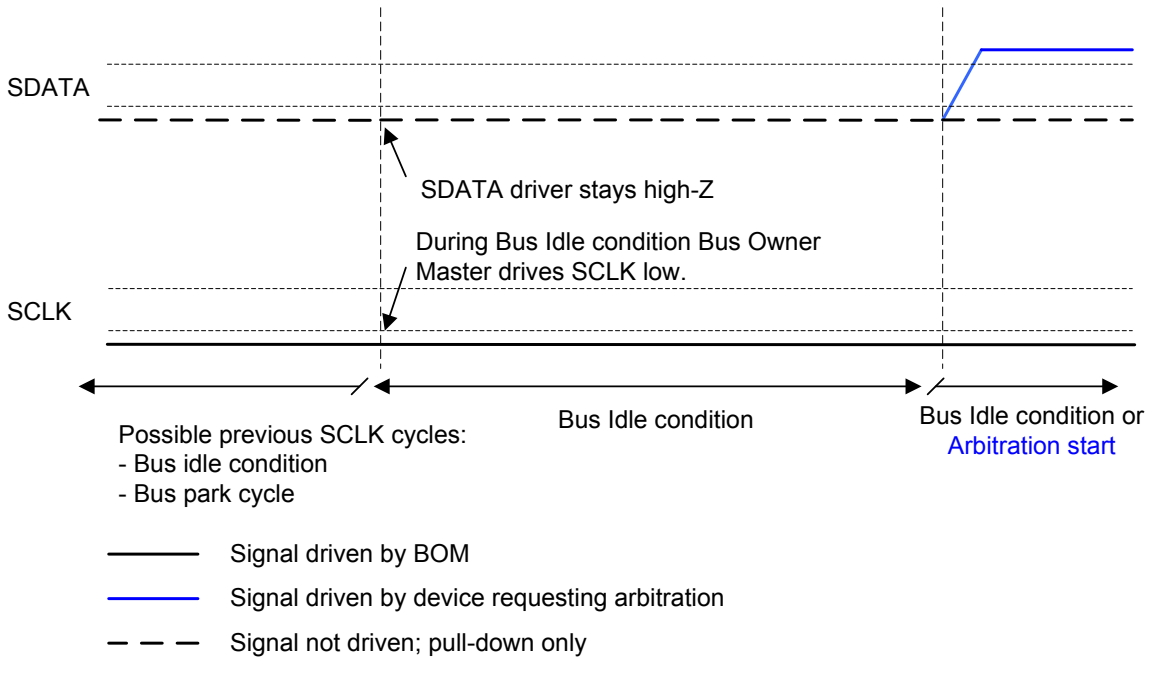


Figure 17 Bus Idle Condition

6.4 Sequence Priority Classes

SPMI bus Sequences shall belong to one of two priority classes: Priority or Secondary Sequences. Priority Sequences are Sequences that are considered by the transmitting device to be of highest urgency and importance, and therefore must be sent as soon as possible. Priority Sequences are always transacted on the bus before any secondary traffic, whether initiated by a Master or a Request Capable Slave. Secondary Sequences are transacted on the bus whenever there is bandwidth left over from the priority Sequences.

See Section 8 for information about Bus Arbitration and how the SPMI bus is allocated to the different Master devices and Request Capable Slave devices based on their Sequence classes.

7 Device Enumeration

Devices on the SPMI bus are Master or Slave devices. To be able to address devices on the bus, SPMI uses Master Identifier (MID) numbers and Slave Identifier (SID) numbers to identify specific devices or groups of slave devices.

Each Master device on a SPMI bus shall have a unique identifier called the Master Identifier (MID). The Master ID number shall be statically assigned by the system integrator.

Each Slave device on the bus shall have a unique identifier called the Unique Slave Identifier (USID). The Unique Slave ID shall be statically assigned by the system integrator.

If there are less than sixteen Slave devices connected to the SPMI bus, the system integrator may assign more than one Slave ID to a Slave device, provided that the device supports such an assignment. The additional Slave Identifier numbers are called Group Slave IDs (GSID).

Unique Slave Identifier numbers and Group Slave Identifier numbers are collectively referred to as Slave Identifiers (SID).

7.1 Master Identifier

A SPMI system may have up to four Master devices. Each Master in a system shall be assigned a unique MID of 0b00, 0b01, 0b10 or 0b11. MIDs do not have to be sequential. For example, in a system with two Master devices, one Master may have a MID of 0b00 and the second Master may have a MID of 0b10.

7.2 Unique Slave Identifier

A SPMI system may have up to sixteen Slave devices. Each Slave in a system shall be assigned a USID between 0b0000 and 0b1111, inclusive. USIDs do not have to be sequential. For example, in a system with four Slave devices, one Slave device may have a USID of 0b0000, the second Slave device may have a USID of 0b1010, the third Slave may have a USID of 0b1110 and the fourth Slave may have a USID of 0b0011.

7.3 Group Slave Identifier

A Group Slave ID is a SID assigned to a Slave device in addition to the USID it already has. The GSID can be the same for any number of Slave devices, and each Slave device may have multiple GSIDs. GSIDs do not need to be sequential, and it is not mandatory for a Slave device to have a GSID.

The use of GSIDs limits the number of USIDs available to the system. For example, in a system that uses two GSIDs, no more than fourteen Slaves devices can exist.

8 Bus Arbitration

SPMI supports the existence of one to four Master devices and up to sixteen Slave devices on the bus at the same time. The Master and Slave devices share the bus using a Bus Arbitration process to determine which Master or Slave device has access to the bus at a given time. The ability to share the SPMI bus between multiple Master and Slave devices allows sharing common peripheral resources as well as direct Master-to-Master, Master-to-Slave, Slave-to-Slave or Slave-to-Master communications via the bus.

A Slave device that can arbitrate for SPMI bus access is called a Request Capable Slave. A Slave device that does not arbitrate for bus access is called a Non-Request Capable Slave.

SPMI is an asynchronous bus and thus, when the bus is idle, multiple devices may request access to the bus via Bus Arbitration. A specific Master, called the Bus Owner Master (BOM), monitors the Bus Arbitration requests and subsequently facilitates the Bus Arbitration process to grant the bus to one requestor.

8.1 Bus Arbitration Levels

There are four different Bus Arbitration levels on the SPMI bus as shown in Table 12.

Table 12 Bus Arbitration Levels

Priority	Bus Arbitration Level	Description
highest	1	Slave arbitration using the Alert bit (A-bit) for priority Sequences from Request Capable Slave devices
	2	Master Priority Arbitration for priority Sequences from Master devices
	3	Slave arbitration using the Slave Request bit (SR-bit) for secondary Sequences from Request Capable Slave devices
lowest	4	Master Secondary Arbitration for secondary Sequences from Master devices

The different Bus Arbitration levels are exposed after arbitration request in the order indicated. The lower levels of arbitration only occur when a high level arbitration mechanism has not previously allocated the bus to a specific device.

The different Bus Arbitration levels allow the devices sharing the SPMI bus to transmit timing critical and urgent Sequences with minimal latency while Sequences that can tolerate more latency are transmitted when unused bus bandwidth is available. All priority Sequences from Request Capable Slave devices are arbitrated at Bus Arbitration level 1. All priority Sequences from Master devices are arbitrated at Bus Arbitration level 2. Secondary Sequences are arbitrated at Bus Arbitration level 3 and Bus Arbitration level 4.

Slave arbitration for both Slave Arbitration methods is based on the SID of the Slave devices on the SPMI bus. See Section 8.4.

Master arbitration is based on Master Priority Levels that are dynamically rotated among Master devices to provide equal access to the bus. See Section 8.5.

8.2 Conditions

A Master shall operate according to the Master Arbitration process described in Section 8.5 even when all other Master devices are disconnected from the SPMI bus. This requirement makes it possible for a disconnected Master or a Request Capable Slave device to access the SPMI bus.

Note, each Master on a single SPMI bus instance may generate SCLK at a different frequency as long as the frequency meets the requirements specified in Section 5.3. In addition, each Master may have a different SCLKint frequency.

Request Capable Slave devices shall follow the Slave Arbitration process as described in Section 8.4.

In a system design with only one Master and no Request Capable Slave devices, Bus Arbitration is not required.

Bus Arbitration shall be executed before every Command Sequence on the SPMI bus except in the case of the sole Master just mentioned.

Before Bus Arbitration, the SPMI bus shall be in the Idle State, and the BOM shall be present on the bus. In case a BOM is not present, Bus Initialization is needed. Refer to Section 9.3 for a description of SPMI Bus Initialization.

8.3 Bus Arbitration Overview

Initial Conditions

The Bus Owner Master has completed the current Sequence transmitting the last bit of the final Frame. Thereafter, the bus is considered Idle by all Master devices and Request Capable Slave devices on the bus. SDATA is maintained at a logic level zero due to the pull-down resistors. SCLK is driven to a logic level zero by the BOM.

Bus Ownership Request

During bus Idle, a Master (already active or waking-up) or Request Capable Slave may initiate a Bus Request by asserting SDATA at a logic level one. A Master or Request Capable Slave shall not request the bus within a time period T_{SDATAZ} from the last falling edge of SCLK of the previous Sequence. Master devices and Request Capable Slave devices may issue Bus Requests at the same time. The Bus Arbitration process results in only one Master or Request Capable Slave device having bus ownership.

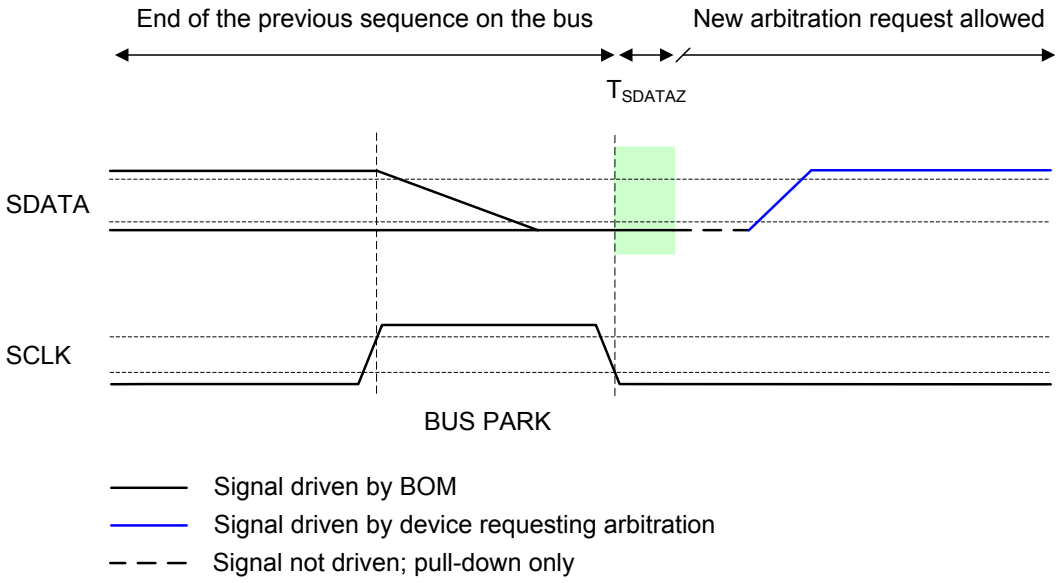


Figure 18 New Arbitration Delay

Bus Arbitration Steps

After the BOM has detected SDATA is logic level one, the BOM shall start the SCLK signal based on its internal clock frequency within 64 µs of the Bus Arbitration Request. All other devices participating in the Bus Arbitration process shall thereafter operate based on this clock signal.

A device disconnected from the SPMI bus shall ignore the Bus Arbitration and maintain its SDATA and SCLK drivers in a high-Z state.

The Bus Arbitration process consists of the following steps:

Step 1: Responding to Arbitration Request

After SDATA has been asserted to logic level one, the BOM shall produce a SCLK pulse. On the falling edge of the SCLK pulse, a device that is driving SDATA shall put its SDATA driver into a high-Z state. The BOM shall produce a second SCLK pulse on the bus and execute a Bus Park Cycle on SDATA during this SCLK cycle.

Step 2: Connecting Master

The BOM shall produce a third SCLK pulse. A Connecting Master shall drive SDATA to logic level one during this SCLK pulse and shall put its SDATA driver into a high-Z state on the falling edge of SCLK. If no Master devices are connecting to the SPMI bus, SDATA remains at logic level zero due to the internal pull-down resistors of the devices connected to the bus.

If the BOM detects that SDATA is logic level one on the falling edge of SCLK, it shall execute the Connect Sequence before proceeding with Bus Arbitration from Step 3. See Section 8.3.1 for a description of the Connect Sequence.

Step 3: Slave Arbitration through Alert-bit

The BOM shall provide one SCLK pulse during which a Request Capable Slave shall drive SDATA to logic level one if the device has a priority Sequence to send. If the BOM detects that SDATA is logic level one on the falling edge of SCLK, it shall execute Slave Arbitration process as described in Section 8.4. If

the BOM does not detect logic level one on SDATA on the falling edge of SCLK it shall execute the Master Priority Arbitration process.

Step 4: Master Priority Arbitration

The BOM shall provide four SCLK pulses for Master Priority Arbitration. A Connected Master with a Priority Sequence to send shall participate in the Master Priority Arbitration as explained in Section 8.5.

Step 5: Slave Arbitration using SR-bit

The BOM shall provide one SCLK pulse during which a Request Capable Slave device shall drive SDATA to logic level one if it has a Sequence to send. If the BOM detects that SDATA is logic level one on the falling edge of SCLK, it shall execute Slave Arbitration as described in Section 8.4. If the BOM does not detect logic level one on SDATA on the falling edge of SCLK it shall execute the Master Secondary Arbitration.

Step 6: Master Secondary Arbitration

The BOM shall provide four SCLK pulses for Master Secondary Arbitration. A Connected Master with any Sequences to send shall participate in the Master Secondary Arbitration as described in Section 8.5. This shall end arbitration.

If arbitration passes through Step 6 without any device winning the bus, the bus is Idle again.

If during Step 3 through Step 6 a device wins arbitration, the arbitration shall be stopped and a Sequence shall be transmitted on the SPMI bus.

The flow diagram in Figure 19 illustrates the SPMI Bus Arbitration process.

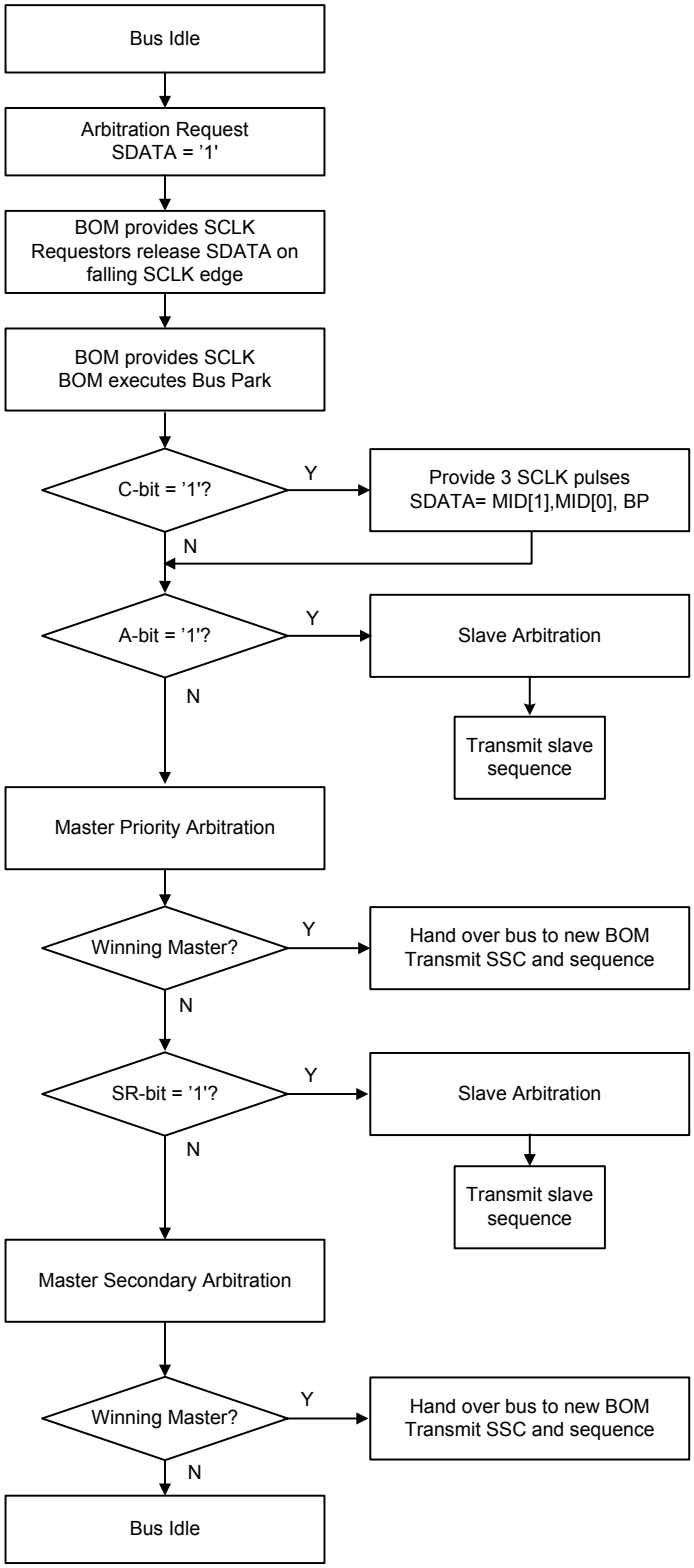


Figure 19 Bus Arbitration Flow Diagram

8.3.1 Connect Sequence

The Connect Sequence facilitates connecting new Master devices to the SPMI bus. Master connection is explained in Section 9.

The Connect Sequence (C-Sequence) is used to provide the MID of the BOM to a Connecting Master so that the Connecting Master can compute its Master Priority Level (MPL). The MPL is required to participate in Bus Arbitration. A Connecting Master shall initiate the C-Sequence by asserting the C-bit to logic level one during Bus Arbitration. After detecting the C-bit value as logic level one, the BOM shall insert three SCLK cycles into the normal Bus Arbitration process and drive the SDATA line with the following bit Sequence: MID[1], MID[0] and Bus Park Cycle (see Figure 20). After the C-Sequence, the Bus Arbitration shall continue from the Alert bit (A-bit).

If the C-bit is logic level zero, the Bus Arbitration shall immediately continue from the A-bit on the next SCLK cycle.

Figure 20 illustrates the Bus Arbitration with and without the Connect Sequence.

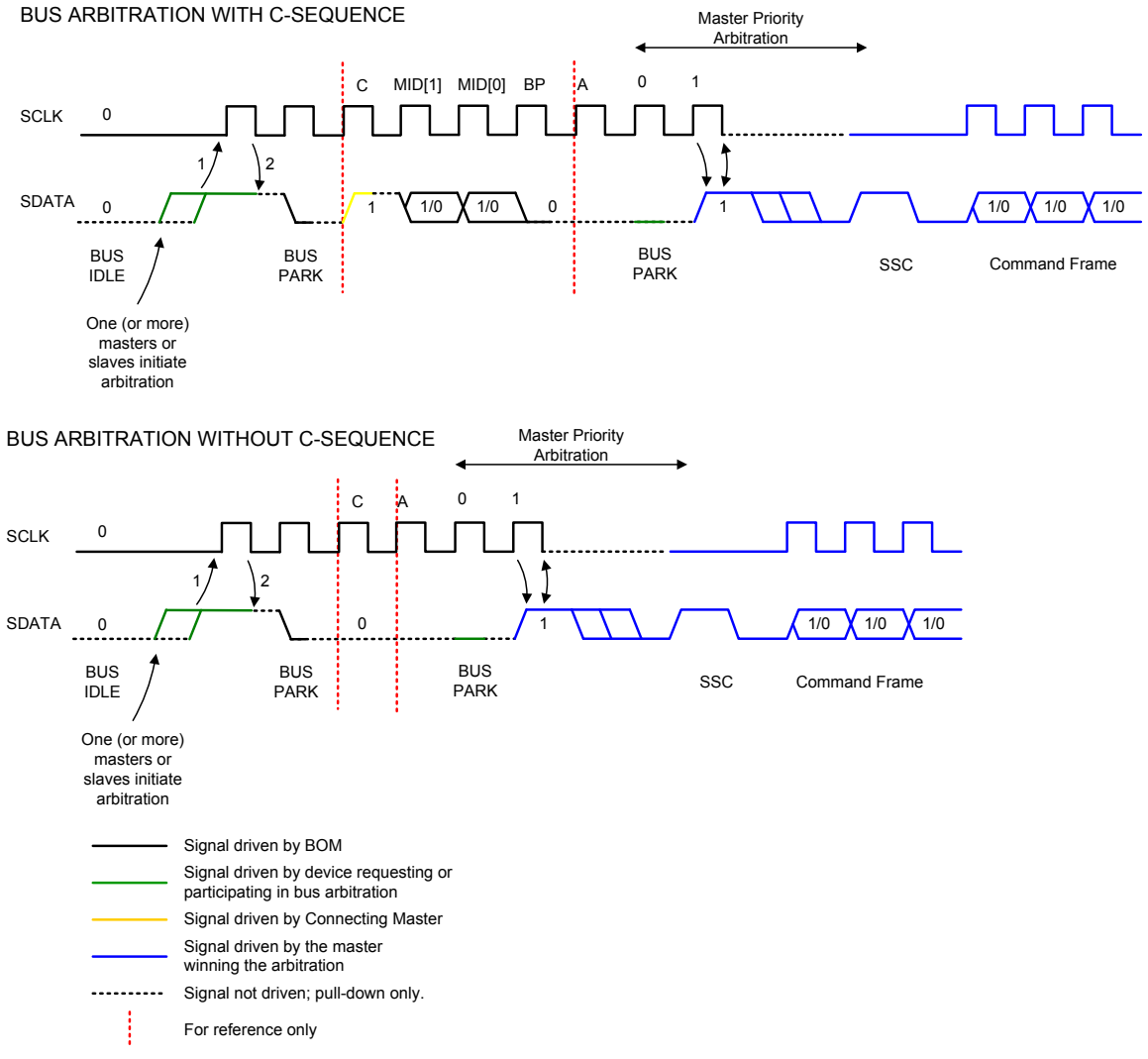


Figure 20 Bus Arbitration with and without C-Sequence

8.4 Slave Arbitration

A Request Capable Slave shall implement the Slave Arbitration process. A Non-Request Capable Slave device shall ignore the Slave Arbitration process and maintain its SDATA driver in a high-Z state throughout the process.

A Request Capable Slave that does not have a Sequence to send, and therefore does not need to participate in the arbitration, shall ignore the Slave Arbitration process and maintain its SDATA driver in a high-Z state throughout the process.

A Request Capable Slave shall initiate the Slave Arbitration process by asserting the A-bit or the SR-bit to logic level one during the Bus Arbitration process. The Slave Arbitration process shall be the same in both cases. It is important for the Request Capable Slave to know whether it requests Slave Arbitration using the A-bit or the SR-bit.

A Request Capable Slave shall use the A-bit to send a Sequence with Bus Arbitration level 1. A Request Capable Slave shall use the SR-bit to send a Sequence with Bus Arbitration level 3.

A Master shall not set the A-bit nor the SR-bit.

Slave Arbitration is based on the fixed Slave Address of each Request Capable Slave device. The BOM shall provide eight SCLK pulses during the Slave Arbitration process.

During the first SCLK pulse a Request Capable Slave device that requested Slave Arbitration shall drive SDATA to logic level one if the Slave Address bit SA[3] is ‘1’. The Request Capable Slave device shall put its SDATA driver to a high-Z state on the falling edge of SCLK. If SA[3] is ‘0’ the Slave device shall maintain its SDATA driver in a high-Z state throughout the SCLK period.

A Request Capable Slave device participating in the arbitration shall read SDATA on the falling edge of SCLK. If SDATA does not match the Slave Address bit SA[3] value for the Slave device then the Slave device shall stop participating in the arbitration process and shall maintain its SDATA driver in a high-Z state for the remainder of the Slave Arbitration process.

The BOM shall execute Bus Park Cycle during the second SCLK pulse. A Request Capable Slave shall maintain its SDATA driver in a high-Z state during the Bus Park Cycle.

This Sequence is repeated for Slave Address bit SA[2] during the third and fourth SCLK periods.

This Sequence is repeated for Slave Address bit SA[1] during the fifth and sixth SCLK periods.

This Sequence is repeated for Slave Address bit SA[0] during the seventh and eighth SCLK periods.

Figure 21 shows the Slave Arbitration process using the A-bit Slave Arbitration request.

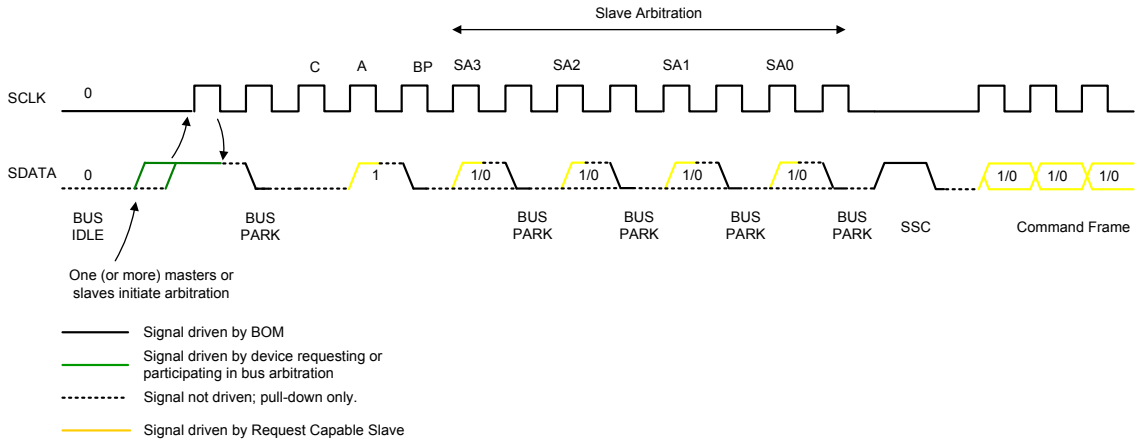


Figure 21 Slave Arbitration using the A-bit

Figure 22 shows the Slave Arbitration process using the SR-bit Slave Arbitration request.

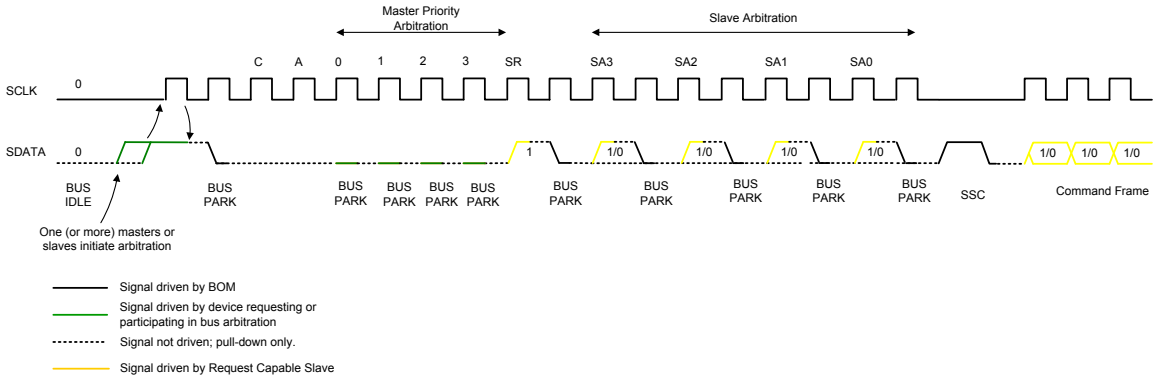


Figure 22 Slave Arbitration using the SR-bit

A Request Capable Slave device that reads its own address during the Slave Arbitration process wins Slave Arbitration and access to the SPMI bus. All other Request Capable Slave devices lose Slave Arbitration.

The BOM shall generate an SSC that completes within T_{BT} (see Section A.2.3) after the end of the Slave Arbitration process. After the SSC, the BOM shall put its SDATA driver in a high-Z state and shall generate thirteen more SCLK pulses. During the thirteen SCLK pulses, the Request Capable Slave device that won Slave Arbitration shall drive the Command Frame of its Sequence on the SPMI bus. The BOM shall also provide any additional SCLK pulses required by the RCS device to complete its Sequence. At the end of the Sequence the SPMI bus is Idle.

Unlike Master Arbitration (Section 8.5), Slave Arbitration does not result in a change to the Master Priority Level. Therefore, after Slave Arbitration, and the subsequent Sequence, the BOM shall be the same as the BOM before Slave Arbitration.

8.4.1 Slave Request Hold

Slave Request Hold is a mechanism that allows a Request Capable Slave device to access the bus despite the fact that Slave Address 0b1111 always win Slave Arbitration if a Request Capable Slave with that address participates in Slave arbitration. The Slave Arbitration process is not fair based on the Slave Addresses alone.

To guarantee other Request Capable Slave devices access to the SPMI bus a Request Capable Slave shall obey the following rules:

- If a Request Capable Slave device requests and wins Slave Arbitration using the A-bit, it shall not request Slave Arbitration again using the A-bit until it has observed at least one arbitration process where the A-bit is logic level zero. While waiting for this condition to occur, the Request Capable Slave device may request Slave Arbitration without the A-bit set to logic level one as many times as necessary.
- If a Request Capable Slave requests and wins Slave Arbitration using the SR-bit, it shall not request Slave Arbitration again using the SR-bit until it has observed at least one arbitration process where the SR-bit is logic level zero. While waiting for this to occur, the Request Capable Slave device may request Slave Arbitration without the SR-bit set to logic level one as many times as necessary

Figure 23 illustrates the state machines that implement these rules.

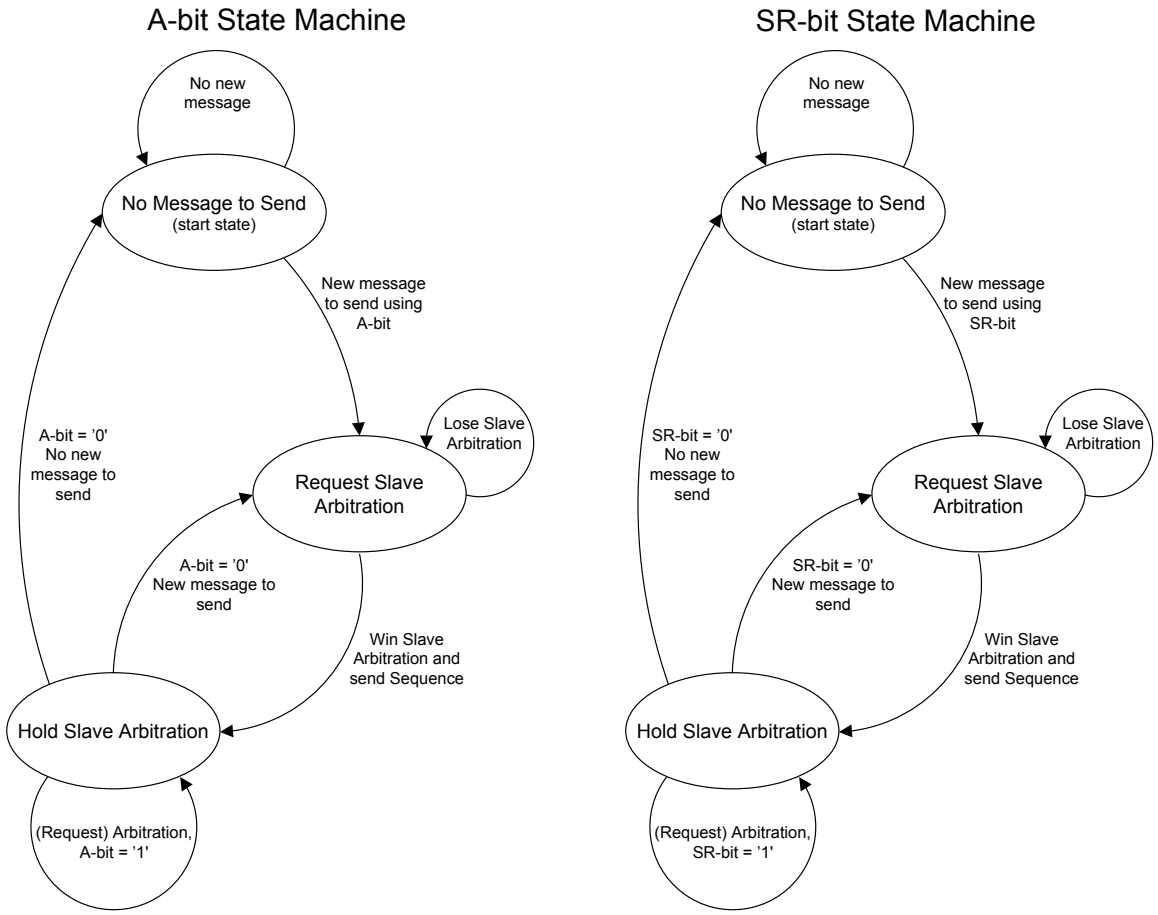


Figure 23 Request Capable Slave Hold State Machines for A-bit and SR-bit Slave Arbitration Requests

8.5 Master Arbitration

The Master Arbitration process enables up to four Master devices to share the bus. SPMI Master Arbitration takes traffic class into account by allocating the bus for priority traffic before allocating the bus for secondary traffic. The arbitration also takes into account the Master Priority Level for each traffic class. The Master with the highest MPL requesting arbitration is granted the bus first.

Master Arbitration can occur twice during Bus Arbitration, first as Master Priority Arbitration if no Request Capable Slave requested Slave Arbitration using the A-bit, and again as Master Secondary Arbitration after the SR-bit if that was logic level '0'. These two Master Arbitration rounds arbitrate bus for priority traffic and secondary traffic.

For the arbitration process, each active Master shall support the following features:

- Priority Sequence queue
- Secondary Sequence queue
- Master Priority Level (see Table 13)

Note that in the following sections, SCLKint signal refers to the internal clock signal of a Master. There is no phase or frequency relationship required between the different SCLK frequencies in separate Master devices. The SCLK bus line is driven by the Bus Owner Master SCLKint signal, thus the frequency of SCLK bus line changes when the ownership of the bus is changed from one Master to another one. An example Master Priority Arbitration Sequence is shown in Figure 24.

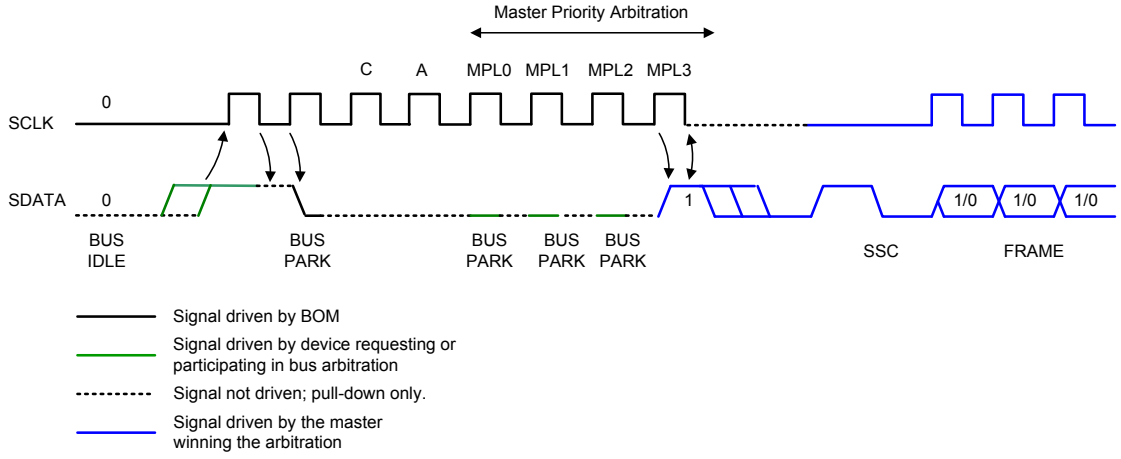


Figure 24 Master Priority Arbitration Example

Master arbitration shall be between one and four SCLK cycles long. Each Master is assigned an SCLK cycle depending upon its Master Priority Level. The Master with MPL=0 shall assert a logic level one during the rising edge of the first SCLK cycle if it is requesting to send a Sequence. A Master with MPL=1, 2, or 3 shall assert a logic level one during its respective SCLK cycle if a Master with a higher priority has not already asserted a logic level one. A Master not requesting to send a Sequence should initiate a Bus Park Cycle during its SCLK cycle.

During Master Priority Arbitration only Master devices with priority Sequences shall participate. If no Master devices are granted the bus ownership during the Master Priority Arbitration the Bus Arbitration continues with the SR-bit as explained in Section 8.3.

If SR-bit is logic level zero Master secondary arbitration shall be performed. Master secondary arbitration is executed in the same way as the Master Priority Arbitration, with the exception that Master devices with Sequences of any traffic class shall participate.

An example of Master secondary arbitration is shown in Figure 25. A Master with MPL=1 needs to send a secondary Sequence. The Master with MPL=1 requests Bus Arbitration. The BOM responds by providing Bus Arbitration clock. No other devices have any Sequences to send so the Bus Arbitration proceeds to Master secondary arbitration (as explained in Section 8.3) where Bus Arbitration results in the bus being granted to the Master.

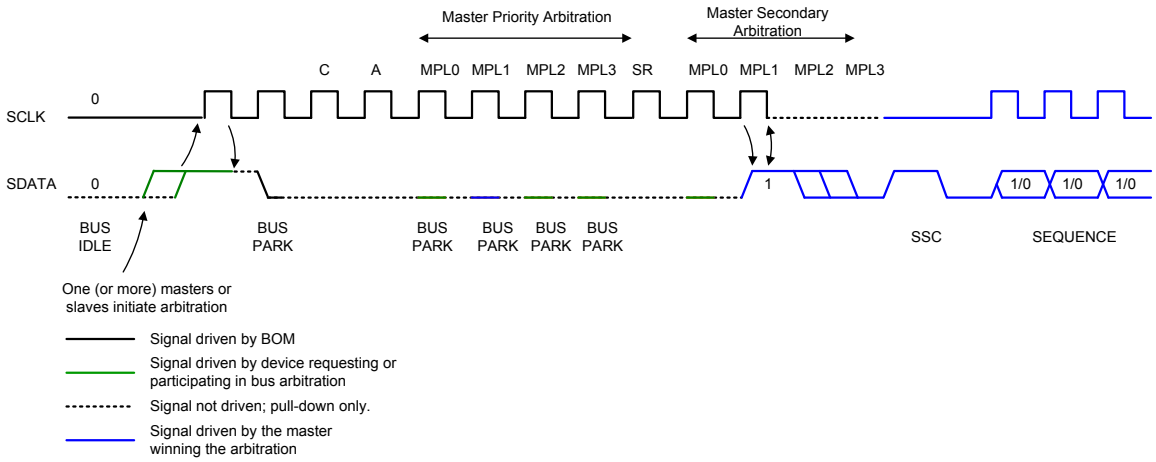


Figure 25 Master Secondary Arbitration Example

During the SSC, each connected Master shall calculate its MPL for the next arbitration round, using the round-robin arbitration algorithm, as described in Section 8.6.

The outcome of this protocol is that all Master priority Sequences are transacted on the bus before any Master secondary Sequences are transacted. All Master devices take turns controlling the bus until all Master devices run out of Master priority traffic. Thereafter, the same Sequence is repeated for Master secondary traffic as long as there are Master Sequences remaining or until Master priority traffic is generated again.

If no Master has any Sequences to transmit the bus goes idle with both SCLK and SDATA signals at logic level zero; SDATA held at a logic level zero by the passive pull-down resistors and SCLK driven to a logic level zero by the BOM. The bus is then Idle.

8.5.1 SCLK Handover

When a Master wins arbitration the SCLK line control shall be transferred from the present BOM to the winning Master, which then becomes the new BOM.

The present BOM shall detect Master Arbitration end by sampling the SDATA on the falling edge of SCLK. If SDATA is logic level one the present BOM shall release the SCLK line.

The new BOM shall wait for one of its SCLKint clock cycles before driving the SDATA line to logic level zero and then sending the Sequence Start condition. SSC shall be completed within T_{BT} of SDATA being driven to logic level zero. At the start of the Sequence Start Condition the new BOM shall drive the SCLK line to logic level zero. After the SSC the new BOM shall start the Command Frame immediately. See Figure 26.

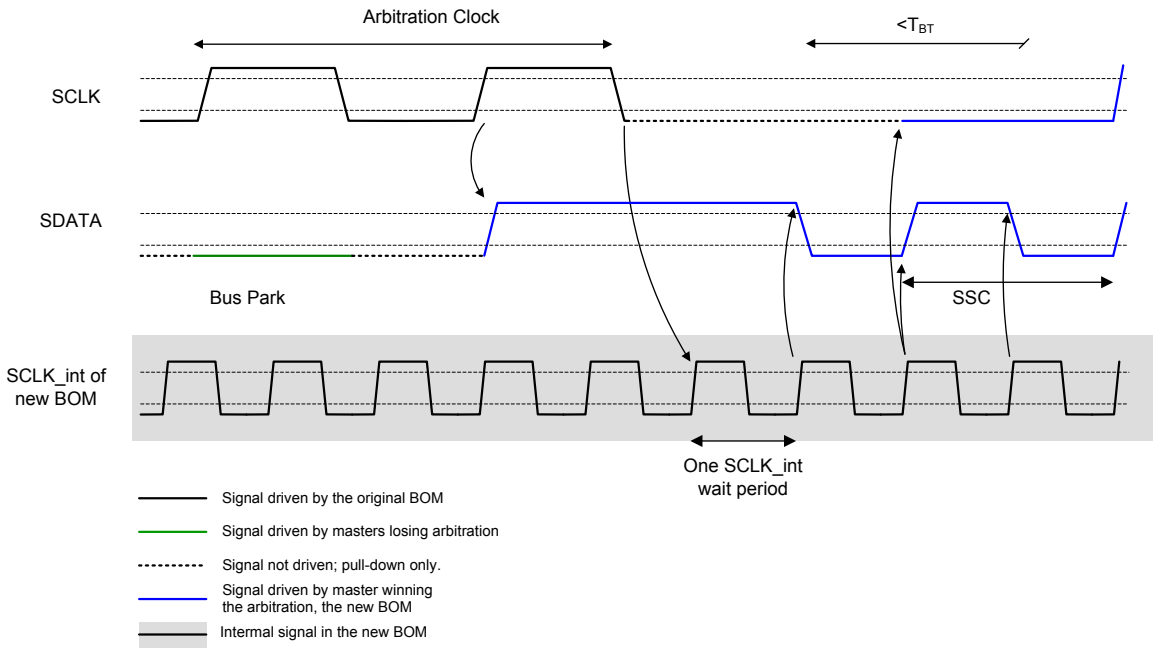


Figure 26 SCLK Handover Sequence

The SSC allows all Master devices and Slaves connected, or connecting, to the bus to synchronize to the bus activity. See Section 6.2.2 for information about the SSC.

8.6 Determining Master Priority Level for Master Arbitration

The MPL of a Master device changes after each Sequence that is sent on the bus using a Round-Robin algorithm. The Round-Robin algorithm has several advantages:

- Provides access to the bus for all Master devices
- The worst-case latency for a Master requesting the bus is easily predicted: it is always proportional to the number of Master devices connected on the bus minus one.
- Knowing the MID of the Bus Owner Master is sufficient for all Master devices, connected or connecting ones, to set their MPL without ambiguity since MPL=3 for the Bus Owner Master. See Section 9.2 for information about connecting to SPMI bus.

A Master shall calculate its new MPL according to the original MPL of the winning Master. For example, if the winning Master originally had MPL=0, a Connected Master would add three (see Table 13) to its current MPL (modulo 4) to determine its new MPL. Similarly, if the winning Master originally had a MPL=1, 2, or 3, all connected Master devices would add 2, 1, or 0, respectively, to their current MPL (modulo 4) to determine their new MPL.

Table 13 New Master Priority Level Calculation

Priority	Winning Master Original MPL	Master Priority Level Adder
highest	0	+3
	1	+2
	2	+1
lowest	3	+0

Formula for calculating the new MPL is:

MPL(n+1) = (MPL(n) + MPLA) modulo 4

Where: MPL(n+1) = new Master Priority Level

MPL(n) = existing Master Priority Level

MPLA = Master Priority Level Adder from Table 13

A Connected Master device shall change its MPL only upon detection of the SSC. The Master Priority Level shall not be changed as a result of Slave Arbitration.

Table 14 shows an example of the Round-Robin arbitration Sequence. In this example, it is not necessary that all Master devices are connected on the bus.

Table 14 Round-Robin Arbitration Sequence

Master ID	MPL, Sequence n	New Bus Request for Sequence n+1	MPL, Sequence n+1	New Bus Request for Sequence n+2	MPL, Sequence n+2
0	MPL = 0		MPL = 2		MPL = 0
1	MPL = 1	Requester[1]	MPL = 3	Requester	MPL = 1
2	MPL = 2		MPL = 0		MPL = 2
3	MPL = 3	Requester	MPL = 1	Requester[2]	MPL = 3

1. *Master ID1 wins arbitration*
2. *Master ID3 wins arbitration*

8.7 Bus Arbitration Error Handling

The SPMI protocol is designed to be robust against Electro-Magnetic Interference (EMI) that might induce noise spikes on the SCLK and/or SDATA lines. This section describes the behavior of SPMI Bus Arbitration when there is a noise spike on SDATA while the bus is in the Idle condition. Noise spikes are not expected on SCLK since the BOM drives a logic level zero on the SCLK during Idle.

If a noise spike occurs on SDATA before arbitration starts, a false bus request is detected. In this case, there are no Master devices or Request Capable Slave devices requesting the bus, thus C-, A- and SR-bit shall be logic level zero as shall be all Master arbitration bits. The arbitration shall end with no device winning the arbitration and the bus shall be Idle.

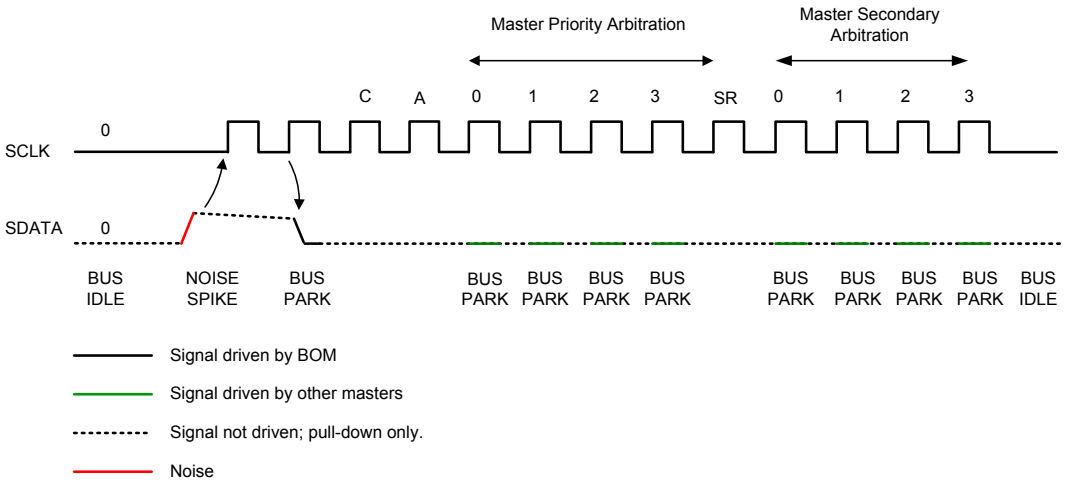


Figure 27 False Bus Request Detection

If a noise spike occurs on SDATA during either the Master Priority or Secondary arbitration, it would appear that a Master is requesting the bus when in reality, it is not. In this situation, there would be no SSC. If the SSC is not detected on the bus for at least the Bus Timeout Period T_{BT} (see Figure 28), the BOM shall retain ownership of the bus and Master Priority Levels shall remain the same.

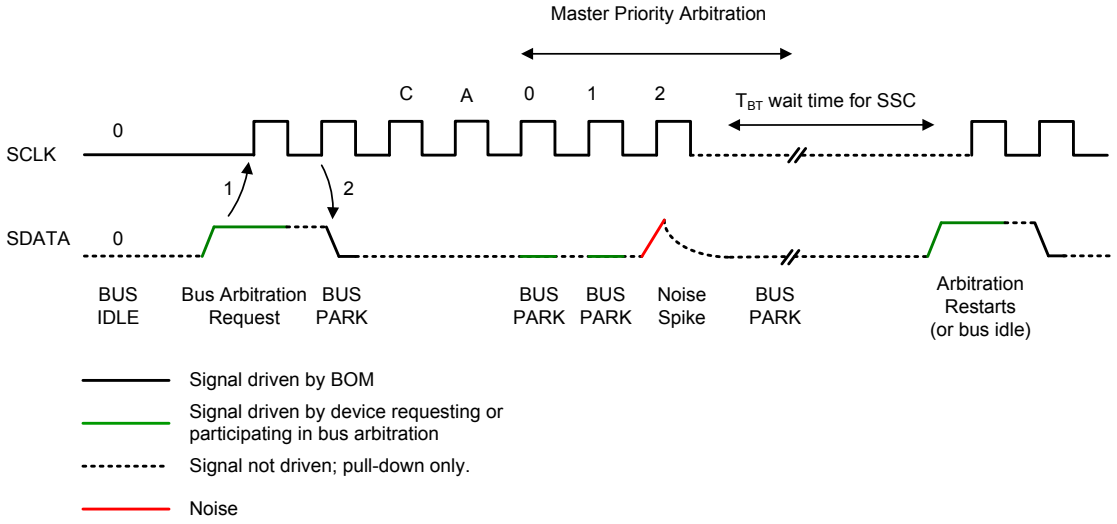


Figure 28 Noise During Priority Slots of the Bus Arbitration

If a Master detects another Master has used the Master Priority Level that it was assigned, the Master shall disconnect from the bus and connect again.

In case noise during arbitration causes accidental Slave Arbitration, the BOM shall assume there is a Slave that won the arbitration in any case. Since no slave requested arbitration, a No Response Frame will be seen on the Bus during the Command Frame. The BOM and any other connected device on the bus shall detect a corrupt Command Frame. As a result of this the BOM shall stop SCLK and the bus shall be idle. Figure 29 shows an example of noise induced Slave Arbitration.

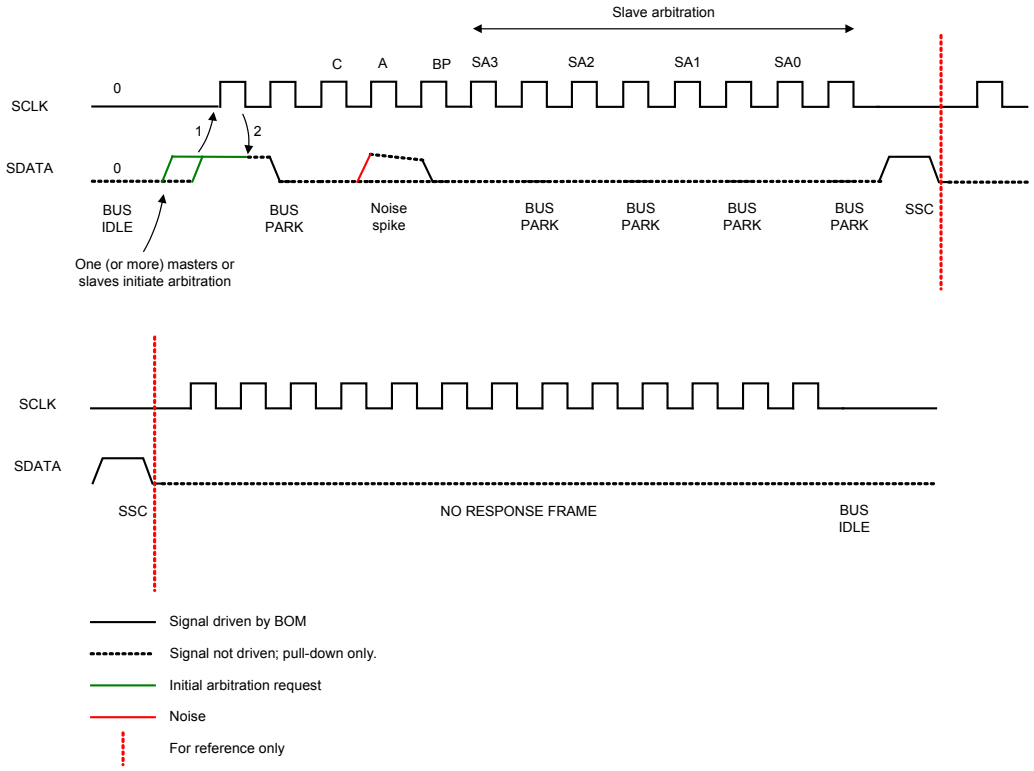


Figure 29 False Slave Arbitration due to Noise

9 Master Connection and Disconnection

9.1 Definitions

Connecting to the SPMI bus is the process where a Master device gains access to the Bus Arbitration process after the bus is initialized.

If the bus is uninitialized then the process of accessing the bus is called Bus Initialization as explained in Section 9.3.

Disconnecting from the SPMI bus is the process where a Master device ceases to participate in the Bus Arbitration process. After disconnecting from the bus, a Master device shall not send Sequences on the bus without first connecting to, or initializing, the bus.

The SPMI bus is Initialized when there is at least one active Master (the Bus Owner Master).

The SPMI bus is Uninitialized when there is no BOM on the bus. This state results when all Master devices, including the Bus Owner Master, disconnect from the bus or when the BOM disconnects from the bus without passing bus ownership to another Connected Master.

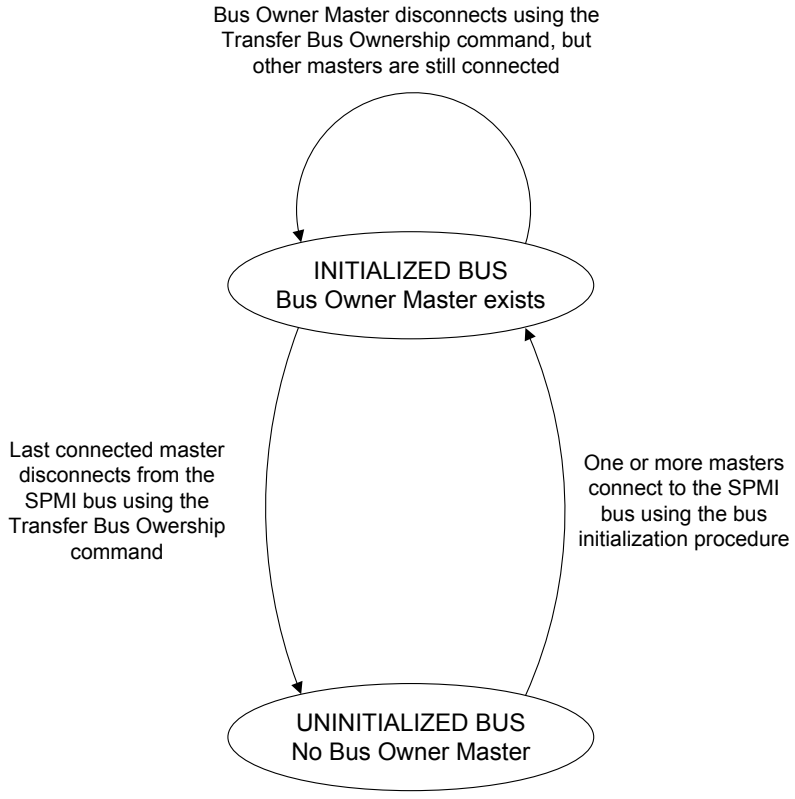


Figure 30 SPMI Bus States

9.2 Master Connecting on the Bus

When connecting to the SPMI bus, a Master device attempts to acquire a MPL so it may participate in the Master Arbitration process. In order to get a MPL the Master device needs to know the MID of the BOM or, if the bus is uninitialized, become the BOM.

A Connecting Master can use any of three different procedures to connect to the SPMI bus. Since the Connecting Master does not know if the bus is initialized, i.e. if there is a BOM connected to the bus, it shall simultaneously monitor the interface for SCLK activity (procedure 2) and look for a SSC (procedure 1) to determine the appropriate procedure to use to connect to the bus. The three procedures can be summarized as follows:

1. If the Connecting Master detects a SSC it uses the procedure described in Section 9.2.1 to connect to the bus. This procedure is guaranteed to work even if the bus is busy.

2. If the Connecting Master detects the Bus Idle state (both SCLK and SDATA are logic level zero), it uses the procedure described in Section 9.2.2 to connect to the bus.

3. If the Connecting Master detects Bus Arbitration (SCLK is logic level zero and SDATA is logic level one), it uses the procedure described in Section 9.2.3 to connect to the bus.

Together these procedures ensure that a Connecting Master is either notified of the MID of the BOM or the Connecting Master initializes the SPMI bus and becomes the BOM. Any number of Master devices may try to connect simultaneously or with arbitrary timing in relation to each other.

9.2.1 Connecting by Detecting SSC

Whenever the Connecting Master detects an SSC, it shall connect to the bus using the following procedure.

Step 1: The Connecting Master shall decode the Frames following the SSC to determine the Command Sequence end. This is possible as Command Sequence length is determined by the Command Frame in the Command Sequence.

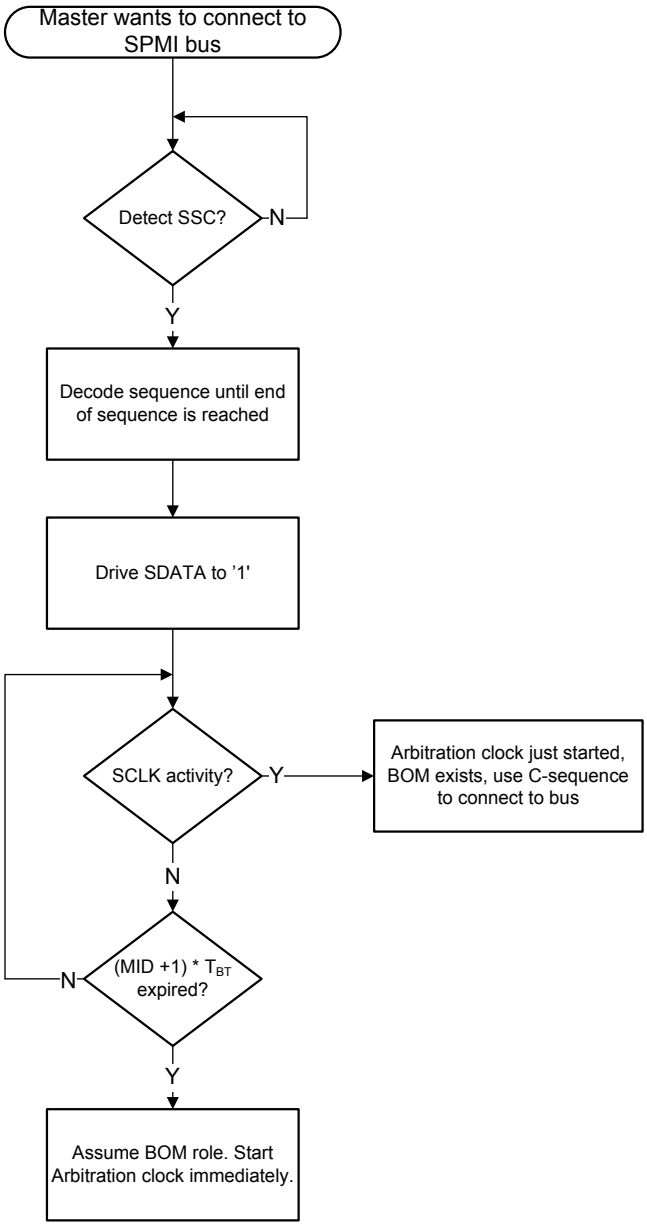
Step 2: After the Command Sequence ends, the Connecting Master shall request Bus Arbitration unless another device has already done so.

Step 3: Once the Bus Arbitration request is set (SDATA is logic level one), the Connecting Master shall wait for $(MID+1)*T_{BT}$ for the arbitration clock to start.

Step 4-A: If the arbitration clock starts (SCLK signal starts) there is a BOM on the bus. The Connecting Master shall set the C-bit during Bus Arbitration, then monitor the MID of the BOM in the C-Sequence (see Section 8.3.1) and set its MPL based on Table 15.

Step 4-B: If $(MID+1)*T_{BT}$ expires before SCLK starts the Connecting Master shall assume the role of BOM and shall start the arbitration clock immediately after $(MID+1)*T_{BT}$ has expired.

Step 1 through Step 3 and Step 4-B result in Bus Initialization. Figure 31 shows a flow diagram of the procedure. Bus Initialization is further explained in Section 9.3.



Note: This procedure runs simultaneously with the Detecting Bus Idle procedure

Figure 31 Detecting SSC to Connect to Bus Flow Diagram

9.2.2 Connecting by Detecting Bus Idle

If the Connecting Master has not yet detected SSC, it shall monitor the SPMI bus for the Bus Idle condition. The Connecting Master shall start an internal timer to wait for T_{BT} to expire while SCLK and SDATA are both logic level zero.

Whenever SCLK signal is detected to be logic level one the timer shall be reset to zero and started again.

If SDATA is detected to be logic level one, the Connecting Master shall try to use the procedure explained in Section 9.2.3 instead.

If the T_{BT} timer expires and SDATA and SCLK are both logic level zero, the SPMI Bus is idle. Thereafter, the Connecting Master shall connect to the bus using the following procedure. Note that these steps are the same as Step 2 through Step 4 as explained in Section 9.2.1.

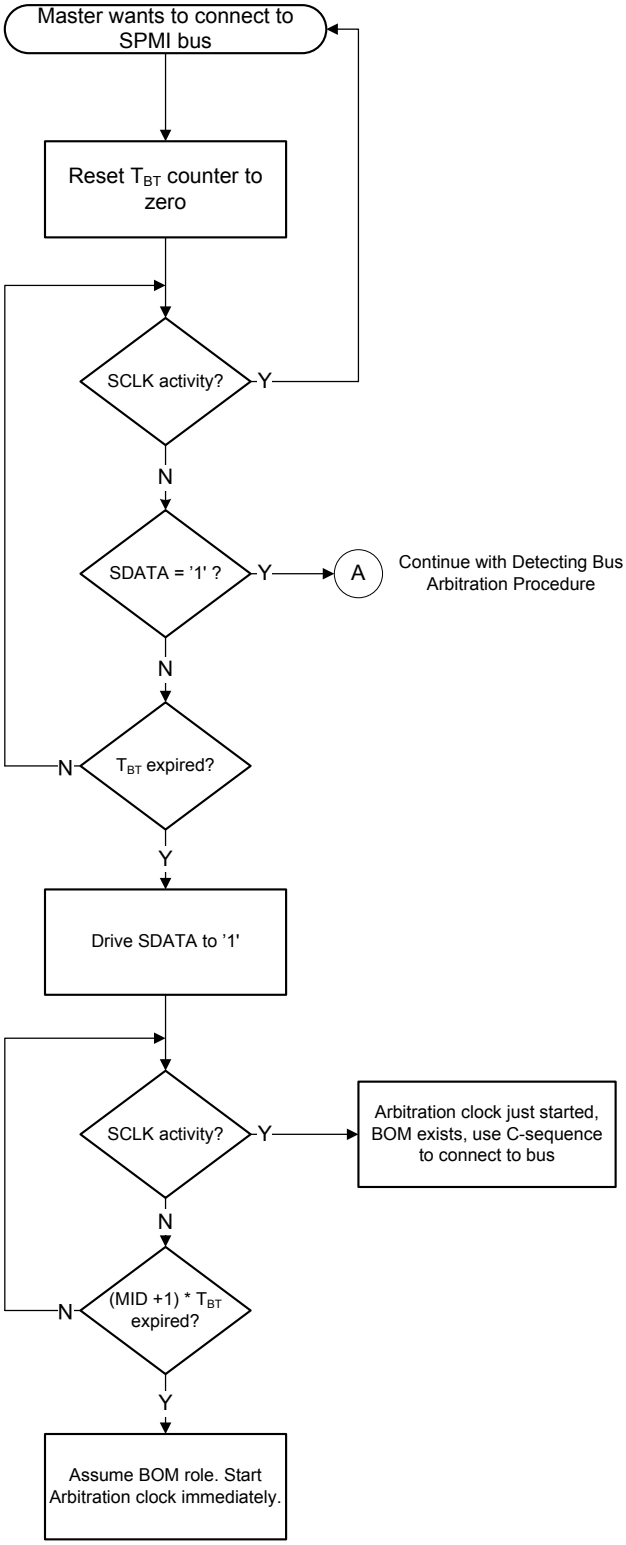
Step 2: The Connecting Master shall request Bus Arbitration unless another device has already done so.

Step 3: Once the Bus Arbitration request is set (SDATA is logic level one), the Connecting Master shall wait for (MID+1)*T_{BT} for arbitration clock to start.

Step 4-A: If arbitration clock starts (SCLK signal starts), there is a BOM on the bus, and the Connecting Master shall set the C-bit in the arbitration Sequence, then monitor the MID of the BOM in the C-Sequence (see Section 8.3.1) and set its MPL based on Table 15.

Step 4-B: If (MID+1)*T_{BT} expires before SCLK starts, the Connecting Master shall assume the role of BOM, and shall start arbitration clock itself immediately after (MID+1)*T_{BT} has expired.

Step 2, Step 3 and Step 4-B result in Bus Initialization. Figure 32 shows a flow diagram of the procedure. Bus Initialization is further explained in Section 9.3.



Note: This procedure runs simultaneously with the Detecting SSC procedure

Figure 32 Detecting Bus Idle to Connect to Bus Flow Diagram

9.2.3 Connecting by Detecting Bus Arbitration

Whenever SDATA is detected to be logic level one and the Connecting Master itself has not yet requested Bus Arbitration, either a Request Capable Slave or another Master has requested Bus Arbitration and there might be multiple other Masters in the middle of the process of connecting to the SPMI bus.

The Connecting Master shall use the following procedure.

Step 1: The Connecting Master shall start a timer counting for $5*T_{BT}$. If SCLK signal arrives during this time, the Master shall exit this procedure and try to use processes described in Section 9.2.1 and Section 9.2.2 instead. If SDATA signal goes to logic level zero during the $5*T_{BT}$ period, there has been noise on the bus, and the Connecting Master shall use processes described in Section 9.2.1 and Section 9.2.2 instead.

Step 2: If $5*T_{BT}$ expires and SDATA is still logic level one, the Connecting Master shall immediately assert SCLK to logic level one and start counting for $(MID+1)*T_{BT}$. The Connecting Master shall also drive SDATA to logic level one continuously.

Step 3. The Connecting Master shall release SCLK within 64 µs of setting SCLK to logic level one and set its SCLK drive output to a high-Z state.

Step 4-A: If while waiting for $(MID+1)*T_{BT}$ to expire the Connecting Master receives SCLK signal then a BOM exists on the bus and normal arbitration continues from the second SCLK pulse in arbitration Sequence. During Bus Arbitration the Connecting Master shall set C-bit to logic level one, observe the MID of the BOM during the next two SCLK cycles (see Section 8.3.1), and set its MPL based on the BOM MID and Table 15. After that the Master is connected. The Connecting Master shall set SDATA driver output to a high-Z state after the C-bit.

Step 4-B: If $(MID+1)*T_{BT}$ expires without SCLK arriving, the Connecting Master shall assume the BOM role. It will immediately drive SCLK signal to logic level zero for one internal SCLK period. Thereafter, it shall generate arbitration clock by driving the SCLK line. The Connecting Master shall keep SDATA at logic level one through the C-bit and then perform the C-Sequence as explained in Section 8.3.1. The Master is now connected and is the BOM.

Table 15 Connecting Master Priority Level

Present BOM MID	Connecting Master			
	MID = 0	MID = 1	MID = 2	MID = 3
0		MPL = 0	MPL = 1	MPL = 2
1	MPL = 2		MPL = 0	MPL = 1
2	MPL = 1	MPL = 2		MPL = 0
3	MPL = 0	MPL = 1	MPL = 2	

Step 1 through Step 3 and Step 4-B result in Bus Initialization. Figure 33 shows a flow diagram of the procedure. Bus Initialization is further explained in Section 9.3.

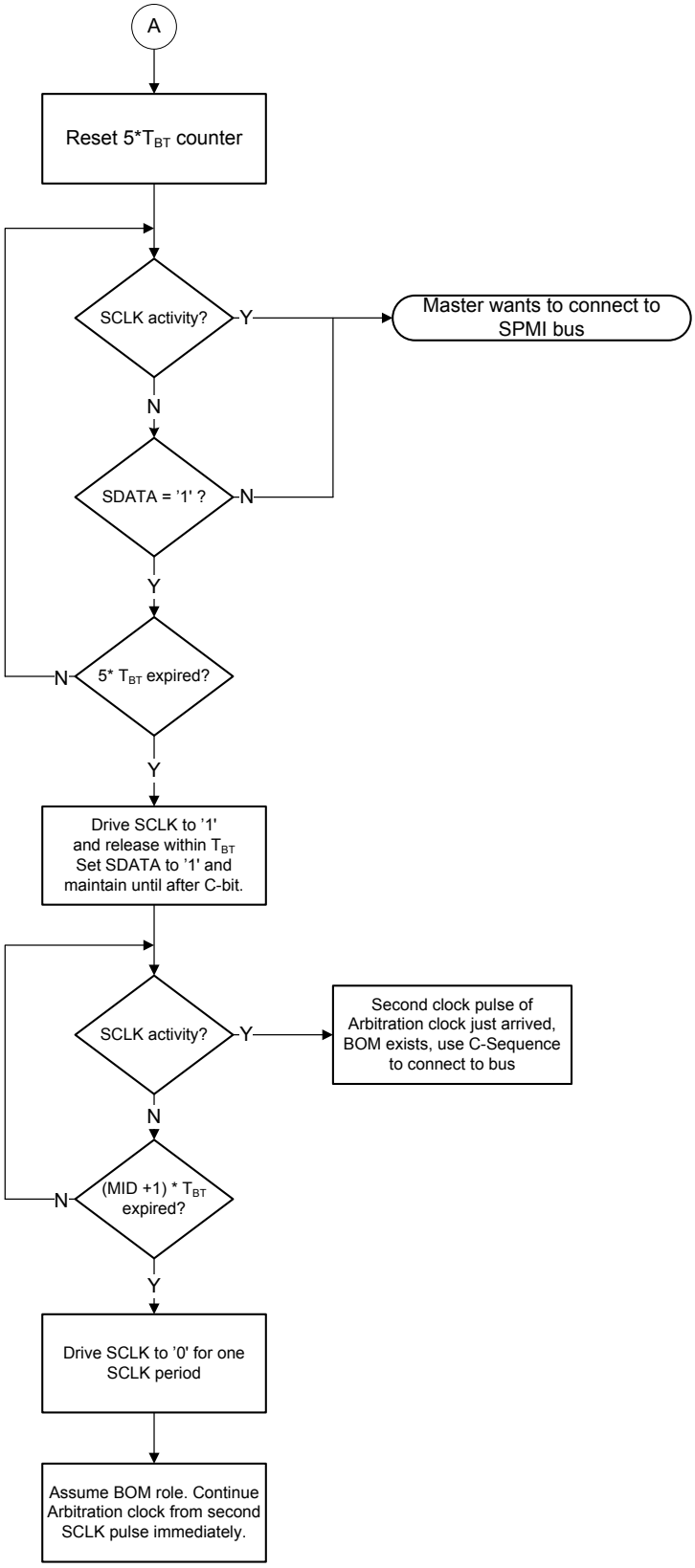


Figure 33 Detecting Bus Arbitration to Connect to Bus Flow Diagram

9.3 Bus Initialization

Bus initialization occurs automatically whenever there is no Bus Owner Master on the SPMI bus and a Master wakes up. The procedures explained in Section 9.2.1, Section 9.2.2 and Section 9.2.3 cause one of the waking Master devices to become the BOM.

The wakeup procedure allows one, a few or all four Master devices to wake up at the same time, or in any order concurrently. To illustrate this Figure 34 shows Master wakeup and subsequent bus initialization for four Master devices concurrently when they are waking up simultaneously, and there is no activity on the SPMI bus.

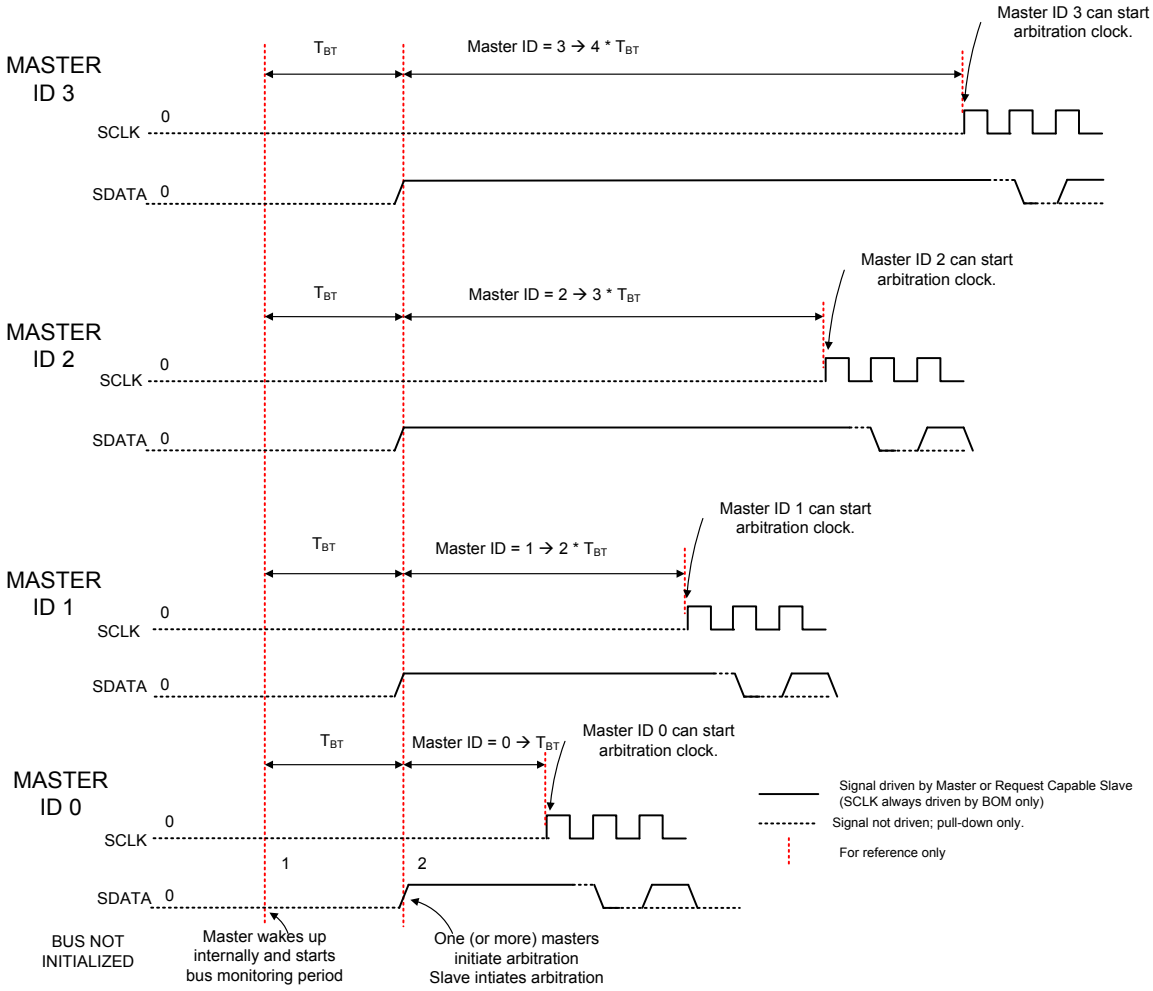


Figure 34 Bus Initialization

Figure 35 shows bus initialization when a Request Capable Slave device is requesting Bus Arbitration before a BOM is on the SPMI bus. In this case, the waking Master shall detect that the SDATA line is logic level one, and shall proceed to connect to the bus using the process explained in Section 9.2.3. Figure 35 shows the SCLK and SDATA activity for all Master devices initializing the bus at the same time to clarify the staggering mechanism that allows Master devices to independently wake up without disturbing each other.

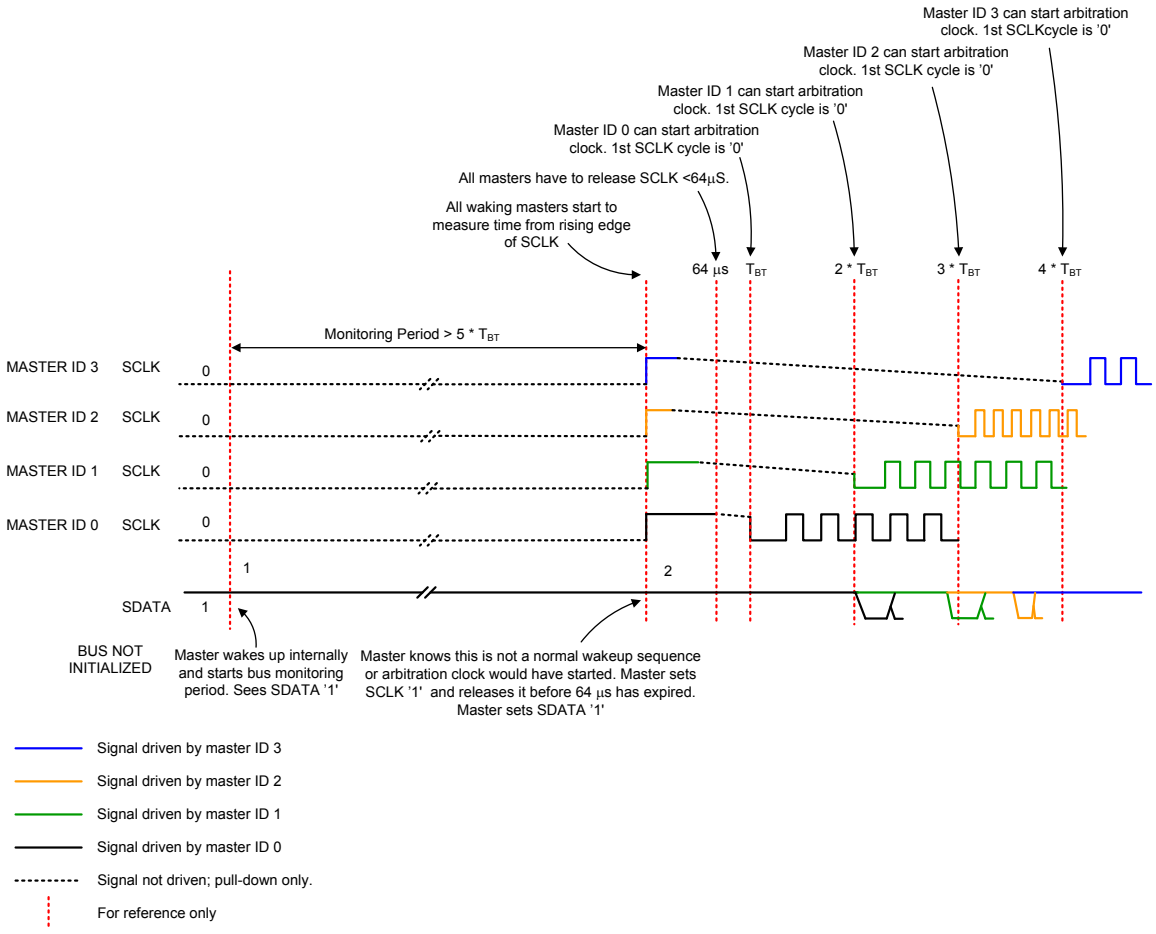


Figure 35 Bus Initialization with Pending Bus Arbitration Request by RCS

9.4 Disconnecting from the Bus

If a Master is not currently the Bus Owner Master, it may disconnect at any time it does not request the bus for sending a Sequence simply by ceasing to track the Master Arbitration Level and stopping any bus management related functions. A Master shall use the connecting procedure described in Section 9.2 if it wants to reconnect to the bus.

A Master may disconnect at any time after completing a Sequence. A Master shall not disconnect after it has requested the bus by asserting SDATA to logic level one, nor in the middle of a Sequence. If the Bus Owner Master wants to disconnect from the bus it shall send the Transfer Bus Ownership Sequence to announce its intention to leave the bus and to request another connected Master, if any, take bus ownership. Refer to Section 13.2.6 for details on the Transfer Bus Ownership Sequence. In no active Master devices remain on the bus at the end of the Disconnect Sequence, the bus becomes uninitialized.

9.4.1 Bus Monitoring by Disconnected Master

When a Master disconnects from the SPMI bus it may still monitor bus activity. Such a Master is called inactive (see Section 11.1.3). A system with Request Capable Slave devices shall have at least one inactive Master that shall monitor the SDATA line such that it will wake up whenever a Request Capable Slave requests Bus Arbitration on an uninitialized bus. Else Slave Communication Request (see Section 10) will not function.

10 Slave Communication Request

A Request Capable Slave has the capability to initiate and send Sequences to any other Master or Slave connected to the SPMI bus.

There is no minimum set of commands defined that a Request Capable Slave shall support for a Sequence initiated by the Request Capable Slave. In this case all commands are optional. However, both the Non-Request Capable Slave and the Request Capable Slave shall support all mandatory commands when addressed by another Master or Request Capable Slave. For a complete overview of the mandatory and optional commands please refer to Figure 55.

The Request Capable Slave devices can request communication on the bus by first requesting Bus Arbitration by setting SDATA to logic level one (see Section 8.4), and then by an alert request (using the A-bit) or a Slave request (using the SR-bit), as described in Section 10.1 and Section 10.2.

The BOM is responsible for generating the SCLK during the complete Slave communication Sequence.

The SDATA is handed over to the Slave that won arbitration after the BOM has generated the SSC. The SSC is described in more detail in Section 6.2.2.

Arbitration for the Request Capable Slave devices is described in Section 8.4.

10.1 Alert bit on an Initialized Bus

The A-bit is used to initiate Slave Arbitration for high priority Slave-to-Slave or Slave-to-Master Sequences.

When the Initialized bus is idle, one or more Request Capable Slave devices may request Bus Arbitration and subsequently set the A-bit to logic level one during the A-bit of the arbitration Sequence.

The Slave Arbitration using the A-bit is shown in Figure 21.

Figure 36 illustrates the flow diagram for Slave Arbitration using the A-bit.

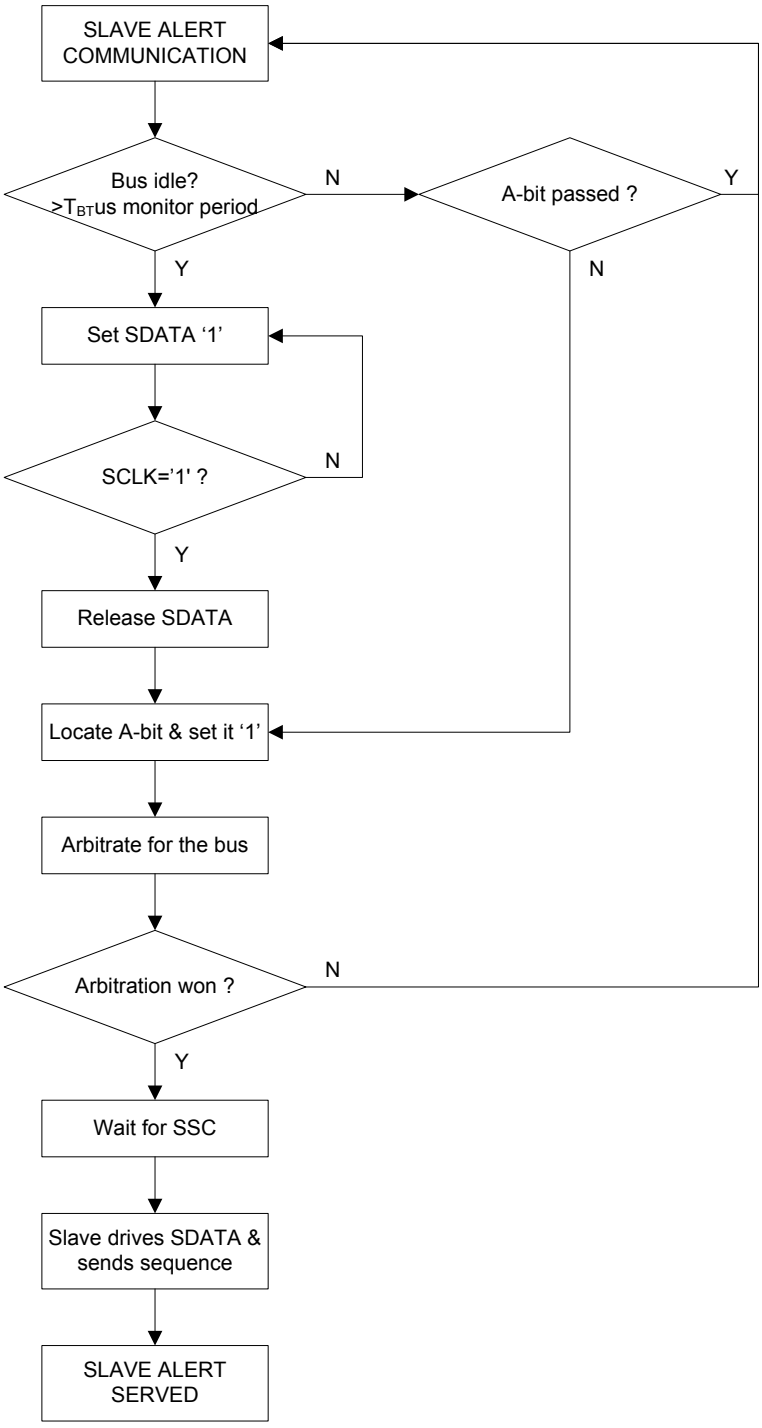


Figure 36 Flow Diagram for Slave Communication Request using the A-bit

10.2 Slave Request Bit on an Initialized Bus

The Slave Request bit (SR-bit) is used to request Slave Arbitration when a Request Capable Slave has a low priority Slave-to-Slave or Slave-to-Master Sequence to be sent.

When the Initialized bus is idle, Request Capable Slave devices may request Bus Arbitration by setting SDATA to logic level one. Then one or more Slaves may initiate a Slave Arbitration by driving SDATA to logic level one during the SR-bit of the arbitration Sequence.

The Slave Arbitration using the SR-bit is shown in Figure 22.

Figure 37 illustrates the flow diagram for a Slave Arbitration using the SR-bit.

Figure 37 Flow Diagram for Slave Communication Request using the SR-bit

10.3 Slave Communication Request on Uninitialized Bus

When the bus is Uninitialized, Request Capable Slave devices can still request the bus for sending their commands by raising SDATA after detecting the bus is idle. The request shall remain active until one Master initializes the bus, after which the Request Capable Slave shall follow the protocol described in Section 10.1 and Section 10.2.

11 SPMI Master Requirements

11.1 Master Operating States

Figure 38 shows the three operating states, BUS OWNER MASTER, CONNECTED, and DISCONNECTED, that describe the operation of a SPMI Master.

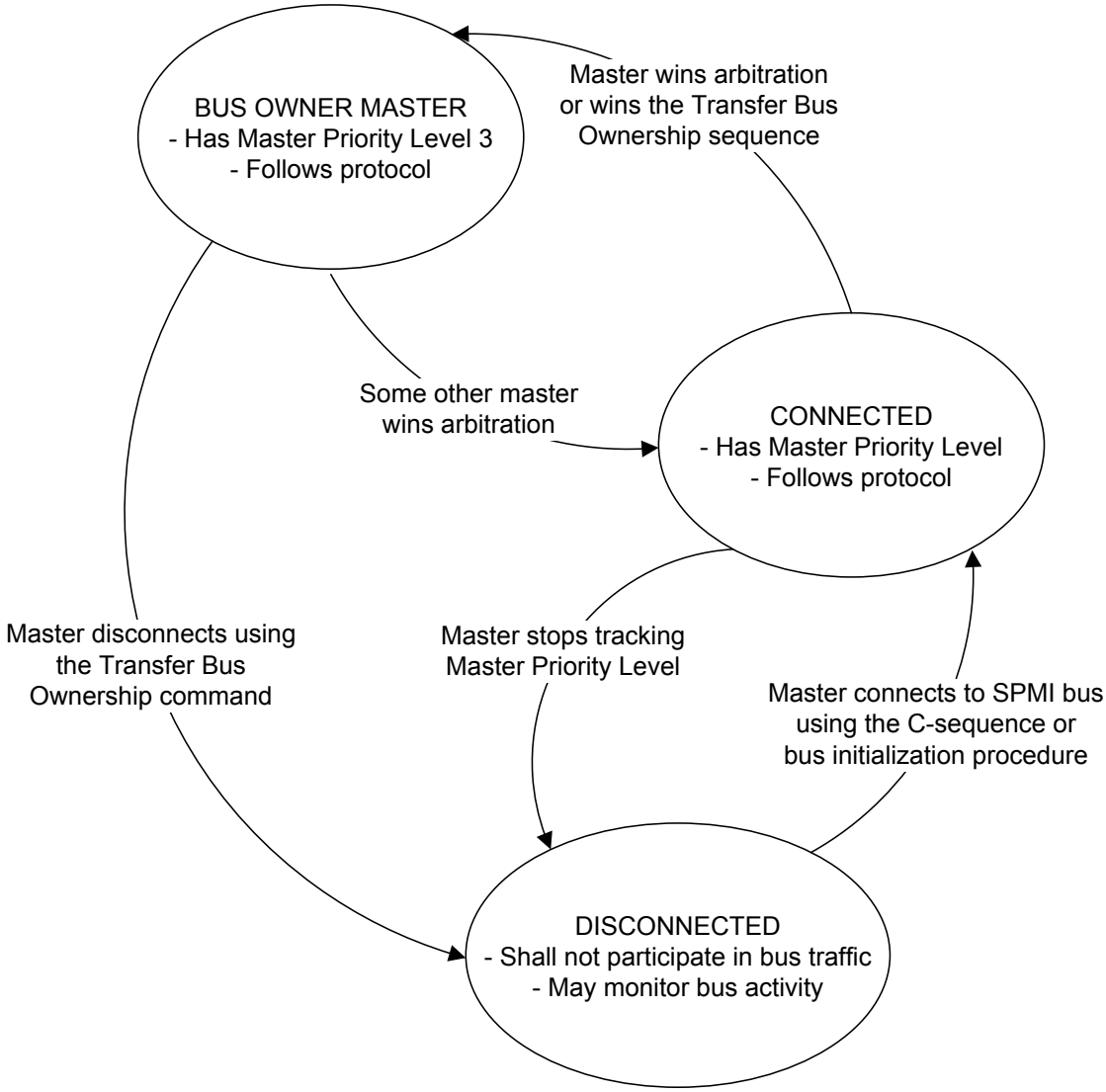


Figure 38 SPMI Master Operating States

11.1.1 Bus Owner Master

The Bus Owner Master shall provide the arbitration clock for the arbitration Sequence. There shall be only one Bus Owner Master at any one time. The Bus Owner Master shall have MPL=3, i.e. the lowest priority.

The Bus Owner Master shall also provide the SSC and SCLK signals for any Sequence transmitted by a Request Capable Slave.

If a Bus Owner Master wishes to disconnect from the SPMI Bus, it shall request arbitration, and after winning Master arbitration, it shall issue the Transfer Bus Ownership Sequence. After issuing this Sequence, the Bus Owner Master shall disconnect from the bus. If the BOM loses Master arbitration before it is able to send the TBO Sequence, it shall no longer be the BOM, and can disconnect simply by ceasing to follow Bus Arbitration and Master Priority Level as explained in Section 11.1.2.

During the TBO Sequence, the Connected Master with the next lowest MPL after the BOM shall become the new Bus Owner Master and set its MPL to 3. All other connected Master devices shall recalculate their MPLs. If the disconnecting Bus Owner Master is the last Connected Master on the bus, the bus state is referred to as Bus Uninitialized. See Section 9.1 for further details.

11.1.2 Connected Master

A Master is connected to the SPMI bus if it has a MPL and it monitors the Master Arbitration Sequence. If a Connected Master stops following the Bus Arbitration Sequence for any reason, that Master shall be deemed to have disconnected from the Bus. A Disconnected Master shall use the Connect Sequence to reconnect to the bus if the bus is Initialized (see Section 8.3.1) or the Bus Initialization Sequence (see Section 9.3) if the bus is Uninitialized.

11.1.3 Disconnected Master

A Master is disconnected from the bus if it does not maintain a MPL and does not respond to bus events. A Disconnected Master shall connect to the bus as described in Section 9.2. A Disconnected Master that monitors the bus activity is called an Inactive Master. The bus monitoring capability of an Inactive Master is out of scope for this document.

11.2 Optional Command Support

A Master shall decode all Command Frames defined in Section 13, including Command Frames defined as optional, to determine the end of a Sequence, even if the Master does not use the optional Command Frames.

11.3 Master External Control Signals

A Master may implement the same, or similar, control inputs and outputs as those provided for a Slave. See Section 12.2.

A Master may also have control inputs that correspond to Slave control outputs. For example, a Master may have a PWROK input corresponding to the Slave PWROK output for the purpose of generating a power-on-reset.

12 SPMI Slave Requirements

There are two types of Slaves, Request Capable Slave devices and Non-Request Capable Slave devices. Request Capable Slave devices can initiate a request for communication on the bus, either by an alert request (using the A-bit) or a Slave request communication (using the SR-bit). During communication by a Request Capable Slave, the BOM supplies the clock on the bus. In order to distinguish an initialized bus from an un-initialized bus, a Request Capable Slave shall have an internal time base so it can determine if the bus is idle.

12.1 Register Map

The SPMI Slave register slave is a single memory map of up to 65536 8-bit registers. Different commands may access different portions of the memory map as shown in the following list:

- Register 0 Write command only accesses the register at address 0
- Register Read/Write commands may access registers from address 0 to 31
- Extended Register Read/Write commands may access registers from address 0 to 255
- Extended Register Read/Write Long commands may access registers from address 0 to 65535.

Note

Bus traffic may be reduced by using the appropriate Read/Write command.

12.2 Slave External Control Signals

A Slave may have three distinct I/O signals, to control or indicate the Slave state, that are independent of the SPMI bus. Other signals unrelated to the SPMI Specification may be present on the Slave.

- Inputs ENABLE and RESETN
- Output PWROK.

12.2.1 RESETN

RESETN is an asynchronous, active-low input signal. Pulling RESETN low shall cause the PMIC to go to the STARTUP state. Voltage regulation is lost once RESETN is asserted. RESETN is optional.

12.2.2 ENABLE

ENABLE is an asynchronous, active-high input signal. The PMIC shall start its power-up Sequence once the ENABLE signal goes high while in the STARTUP state. If the ENABLE signal goes low in any state the device shall go to the STARTUP state. ENABLE is optional.

12.2.3 PWROK

PWROK in an asynchronous, active-high output signal. When PWROK is high, the SPMI controlled regulators on the PMIC have valid output voltages. PWROK can be used to generate reset signals, etc. for the SoC.

12.2.4 Exceptional Control Inputs

A PMIC can use information about its operating conditions and environment to adjust its operating state within those states. These inputs are defined by the manufacturer of the PMIC and should be constrained to

sensing environmental and operational parameters required to determine the operating conditions of the device. For example a device may sense input and output voltages, currents, temperature, pressure, ambient lighting, radiation and so on.

12.2.5 Other Control Inputs and Outputs

If a Slave is part of a larger IC with external control inputs, these external control inputs can be used to manipulate the RESETN and ENABLE input signals, which are internal to the larger PMIC. If RESETN and ENABLE operation is different than that specified in Section 12.2.1 and Section 12.2.2 then they shall be named differently to avoid confusion. The same principle applies to the PWROK output signal.

Figure 39 shows two example implementations. The first implementation is a stand-alone SPMI Slave with external control inputs and outputs. The second implementation is an integrated SPMI Slave with controls internal to the host device.

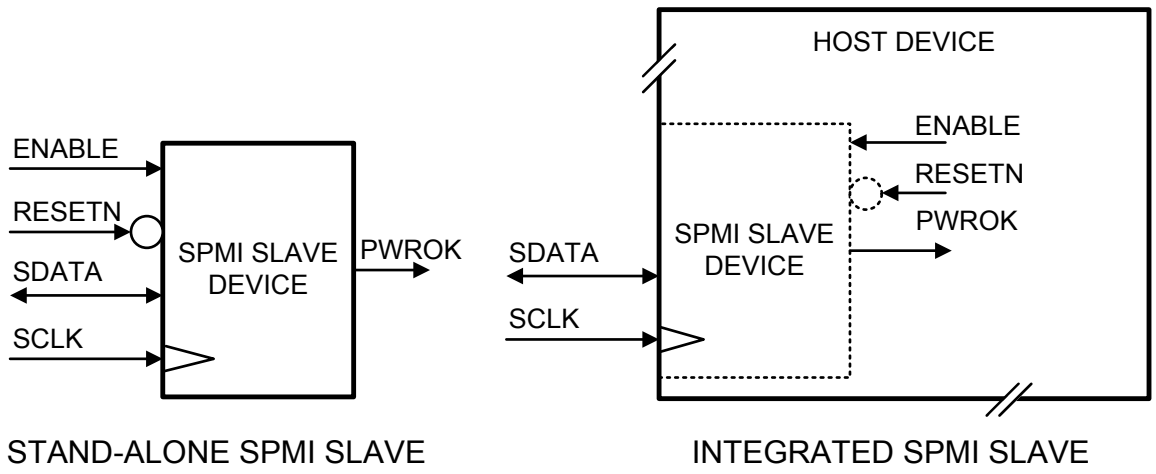


Figure 39 Alternate External Control Signal Configurations

12.2.6 Multiple Logical Slaves on a Single Physical Device

A single physical device may have more than one logical Slave device. In this case the logical Slaves can share the external control inputs (or corresponding host device internal signals). SPMI protocol commands operate on individual logical Slaves. A logical Slave's operation may be Sequenced or coupled to another Slave's operation.

12.3 Slave Operating States

SPMI Slaves shall have the four operating states, ACTIVE, SLEEP, SHUTDOWN and STARTUP, as shown in Figure 40. Separate SHUTDOWN and STARTUP states are provided to allow an independent SPMI command initiated shutdown process without the requirement to simultaneously configure the external control signals to prevent an immediate re-boot from the STARTUP state. The external control signals (see Section 12.2) have priority over the SPMI commands.

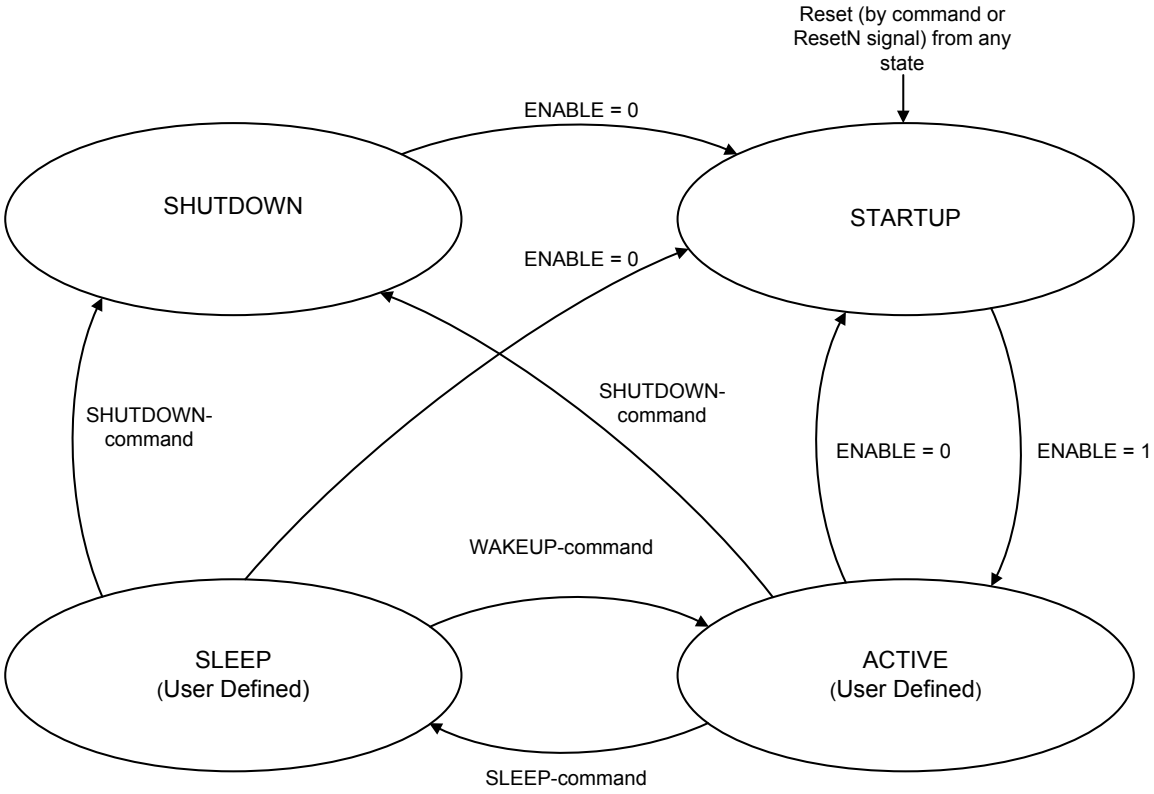


Figure 40 SPMI Slave State Diagram

12.3.1 STARTUP

The STARTUP state is the default state after a reset. The STARTUP state is entered from any other state if the ENABLE signal is asserted to logic level zero. All regulators shall be off in this state and the PWROK output shall be logic level zero. The power-up (or power-down) Sequence of the regulators is device specific. The Sequences in which voltage regulators activate, and the time required to activate them, depend on the needs of the target application and SoC. When the ENABLE signal is asserted high, a Slave device shall transition into the ACTIVE state.

12.3.2 ACTIVE

The ACTIVE state is the normal operating state of a Slave after the power-on Sequence. The state of any regulator or sub-module on the Slave is either user-defined or specified by the device manufacturer. In the ACTIVE state, a SPMI Master may control any of the voltages on the PMIC by programming the corresponding SPMI control registers. Changing the state of a voltage regulator does not necessarily cause a transition away from the ACTIVE state.

12.3.3 SLEEP

The SLEEP state is another user-defined state and is similar to the ACTIVE state in that the state of any regulator or sub-module on the Slave is either user-defined or specified by the device manufacturer. The SLEEP state is meant to allow for a lower power state that may be entered automatically under hardware control.

In the SLEEP state the SPMI Master may control any of the voltages on the PMIC by programming the corresponding SPMI control registers if the internal register clock is available within the Slave. Changing the state of a voltage regulator does not necessarily cause a transition away from the SLEEP state.

12.3.4 SHUTDOWN

In the SHUTDOWN state, a Slave shall turn off all regulators, thus causing all output voltages to go to zero. Exit from the SHUTDOWN state is achieved with a RESETN command, external signal or by setting the external ENABLE to a logic level zero and then back to a logic level one.

12.3.5 Exceptional State Transitions

This document does not specify the external or environmental conditions required for PMIC operation, only the normal operation that occurs while external and environmental factors are within operating limits. The PMIC can have operating limit monitoring, and the limit monitoring can affect PMIC state transitions, triggering state transitions like going to the SHUTDOWN state. New, top-level states may not be added into the Slave state behavior although sub-states may be added under the top level states. Examples of exceptional state transitions are PMIC shutdown triggered by excessive die temperature or exceeding the output current of a regulator.

12.4 Optional Command Support

A Request Capable Slave shall decode all Command Frames defined in Section 13 to determine the end of a Sequence, even if the Slave does not use the Command Frames.

13 Command Sequences

The Command Sequences of the SPMI protocol only apply to an initialized SPMI bus. Each Sequence accomplishes one complete transaction over the interface. Command Sequences shall be sent only by a Master device or a Request Capable Slave device. A Non-Request Capable Slave device shall not send Command Sequences.

A SPMI Master device shall support all Command Sequences. A Request Capable Slave device may support any, all or none of the Command Sequences. A Non-Request Capable Slave device acting as a receiver may support any, all or none of the Command Sequences.

13.1 Command Sequence Summary

This section describes all Command Sequences defined in the SPMI protocol.

See Section 6 for more information about Sequences and other SPMI constructs. The coding of the Command Frame is shown in Table 16.

Table 16 Summary of SPMI Commands

Command Frame Payload	Description
0x00 to 0x0F	Extended Register Write
0x10	Reset
0x11	Sleep
0x12	Shutdown
0x13	Wakeup
0x14	Authenticate
0x15	Master Read
0x16	Master Write
0x17 to 0x19	Reserved
0x1A	Transfer Bus Ownership
0x1B	Device Descriptor Block Master Read
0x1C	Device Descriptor Block Slave Read
0x1D to 0x1F	Reserved
0x20 to 0x2F	Extended Register Read
0x30 to 0x37	Extended Register Write Long
0x38 to 0x3F	Extended Register Read Long
0x40 to 0x5F	Register Write
0x60 to 0x7F	Register Read
0x80 to 0xFF	Register 0 Write

13.2 Command Sequence Descriptions

Each Command Sequence is described in detail in the following sections.

13.2.1 Extended Register Write Command Sequence

The Extended Register Write allows write access to the extended register space on a Slave device. One to sixteen bytes of data shall be written in a single Command Sequence. The extended register address shall be supplied in a separate Data Frame in the Command Sequence.

Figure 41 shows the Extended Register Write Command Sequence. The Command Sequence starts with the SSC followed by the Extended Register Write Command Frame, an Address Frame and one or more Data Frames with the data to be written. Note that the Data Frames immediately follow the Command Frame in a continuous Sequence. ACK/NACK follows the Data Frame. The Command Sequence ends with a Bus Park Cycle.

The four LSBs of the Extended Register Write Command Frame, BC[3:0] in Figure 41, indicate the number of bytes to be written in the Command Sequence. BC3 is the byte count MSB. 0b0000 indicates one byte shall be written and 0b1111 indicates sixteen bytes shall be written.

The register address in the Command Sequence contains the address of the first extended register to be written. If more than one byte is written in a single Command Sequence then the Slave's local extended register address shall be automatically incremented by one for each byte written up to address 0xFF, starting from the address indicated in the Address Frame. If the extended register address reaches 0xFF before the last Data Frame in the Sequence then the content of the register at address 0xFF is overwritten with the overflow data.

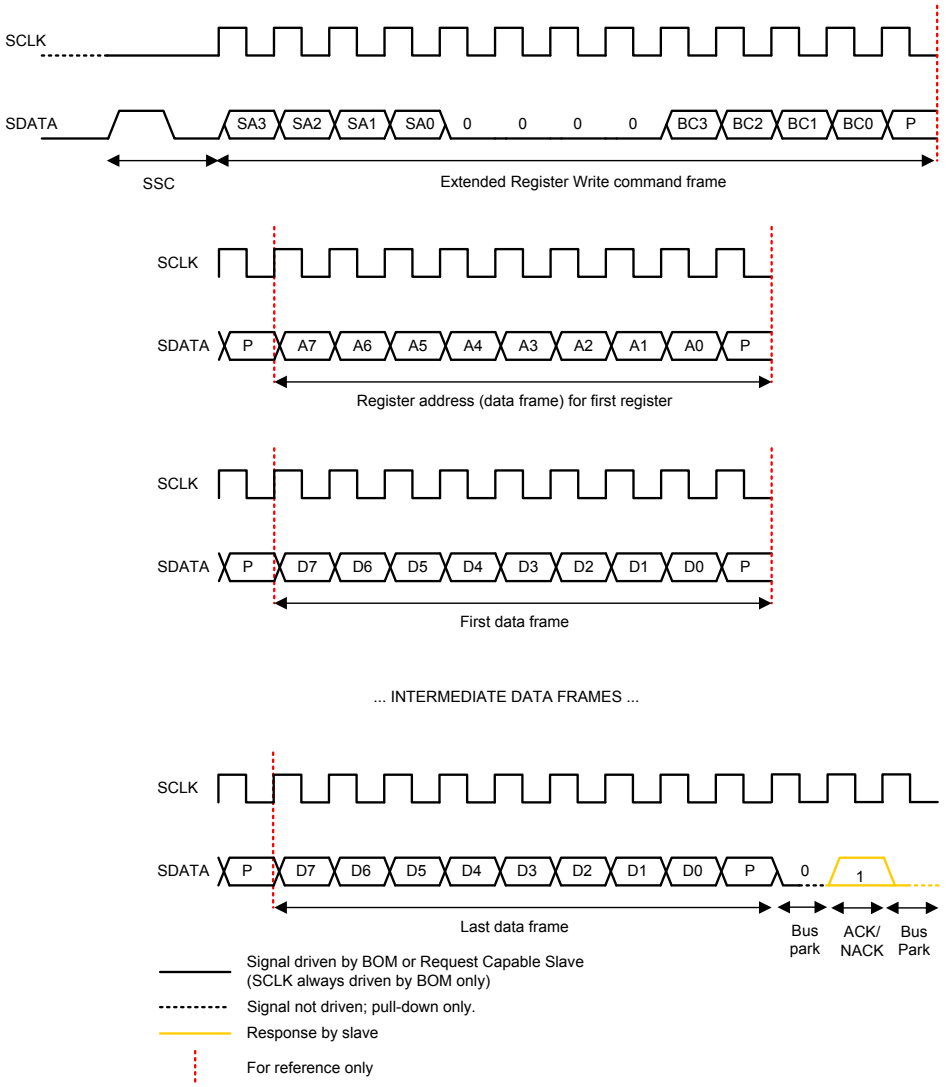


Figure 41 Extended Register Write Command Sequence

13.2.2 Extended Register Read Command Sequence

The Extended Register Read Command Sequence allows read access to the extended register space on a Slave device. One to sixteen bytes of data shall be read in a single Command Sequence. The extended register address shall be supplied in a separate Data Frame in the Command Sequence.

Figure 42 shows the Extended Register Read Command Sequence. The Command Sequence starts with the SSC followed by the Extended Register Read Command Frame, an Address Frame and one or more Data Frames with the data read from the Slave. A Bus Park Cycle occurs between the Address Frame and the Data Frames. The Command Sequence ends with a Bus Park Cycle.

The four LSBs of the Extended Register Write Command Frame, BC[3:0] in Figure 42, indicate the number of bytes to be read in the Command Sequence. BC3 is the byte count MSB. 0b0000 indicates one byte shall be read and 0b1111 indicates sixteen bytes shall be read.

The register address in the Command Sequence contains the address of the first extended register to be read. If more than one byte is read in a single Command Sequence then the Slave's local extended register

address shall be automatically incremented by one for each byte read up to address 0xFF, starting from the address indicated in the Address Frame. If the extended register address reaches 0xFF before the last Data Frame in the Command Sequence then the content of the register at address 0xFF is read multiple times. An Extended Register Read command by the Master to an unsupported Slave extended register address results in a No Response Frame from the Slave. If the address that results from auto-incrementing the Slave's local extended register address is for an unsupported register then the Slave sends a No Response Frame to the Master in place of the Data Frame. The Command Sequence continues from the next extended register address.

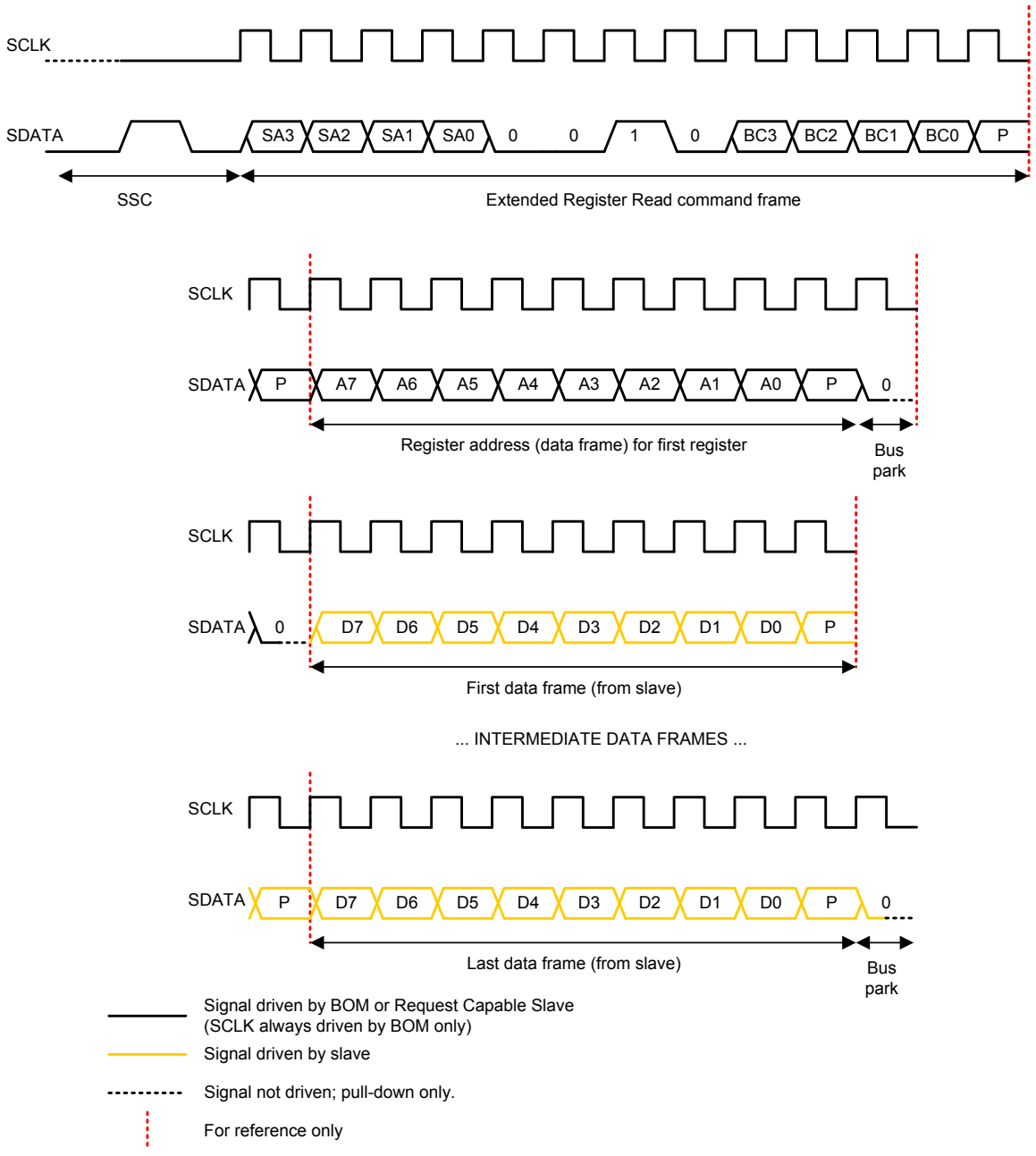


Figure 42 Extended Register Read Command Sequence

13.2.3 Reset, Sleep, Shutdown, Wakeup Command Sequences

The Reset, Sleep, Shutdown, and Wakeup Command Sequences are similar in format and thus are described together in this section. Each Command Sequence starts with the SSC. The SSC is followed by a Command Frame and the Sequence ends with a Bus Park Cycle. The Command Frame is unique for each command. Figure 43 shows the Command Sequences for these commands and the values for each Command Frame.

13.2.3.1 Reset

Writing the Reset command initializes the Slave and forces all registers to their reset values. The Slave shall enter the STARTUP state after receiving a Reset command, and acknowledging it with ACK/NACK. See Section 12.3.1.

13.2.3.2 Sleep

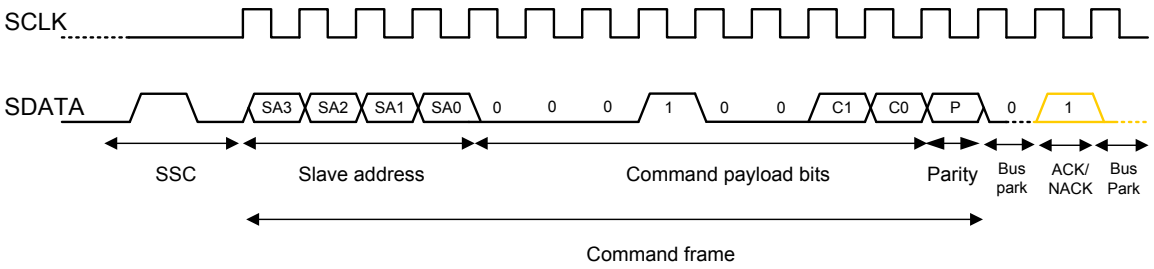
The Sleep command causes the Slave to enter the user defined SLEEP state after acknowledging it with ACK/NACK. See Section 12.3.3.

13.2.3.3 Shutdown

The Shutdown command causes the Slave to enter the SHUTDOWN state after using ACK/NACK to acknowledge the command. See Section 12.3.4. The Slave enters the SHUTDOWN state after a Shutdown command is received while in either the ACTIVE or SLEEP states. In the STARTUP state, the Shutdown command is ignored because the reset and enable signals take priority. See Section 12.3.

13.2.3.4 Wakeup

The Wakeup command causes the Slave to move from the SLEEP state to the ACTIVE state. The Slave acknowledges receiving the Wakeup command using ACK/NACK. See Section 12.3.2 and Section 12.3.3.



Command Bits	C1	C0
RESET	0	0
SLEEP	0	1
SHUTDOWN	1	0
WAKEUP	1	1

Signal driven by BOM or Request Capable Slave (SCLK always driven by BOM only)
Signal not driven; pull-down only.
Response by slave

Figure 43 Reset, Sleep, Shutdown and Wakeup Command Sequences

13.2.4 Authentication Command Sequence

The Authenticate command starts a 9-Frame Command Sequence consisting of a Command Frame, four Challenge Frames and four Response Frames, to interrogate a Slave device for identification. Challenge and Response Frames are Data Frames. Challenge and response data patterns are manufacturer defined, and

may be encrypted. This Command Sequence is intended to support intellectual property management and device identification. A Slave that does not support the Authentication Command Sequence shall respond with No Response Frames (see Section 6.2.3.3). Authentication Command Sequence shall not use ACK/NACK as the Response Frames fulfill this function.

The Authentication Command Sequence starts with the SSC and the Authenticate Command Frame followed by alternating Challenge and Response Frames. Challenge Frames shall be transmitted using the Data Frame format. A Slave shall transmit Response Frames using the Data Frame format. Challenge Frames from the Master are interleaved with Response Frames from the Slave as shown in Figure 44.

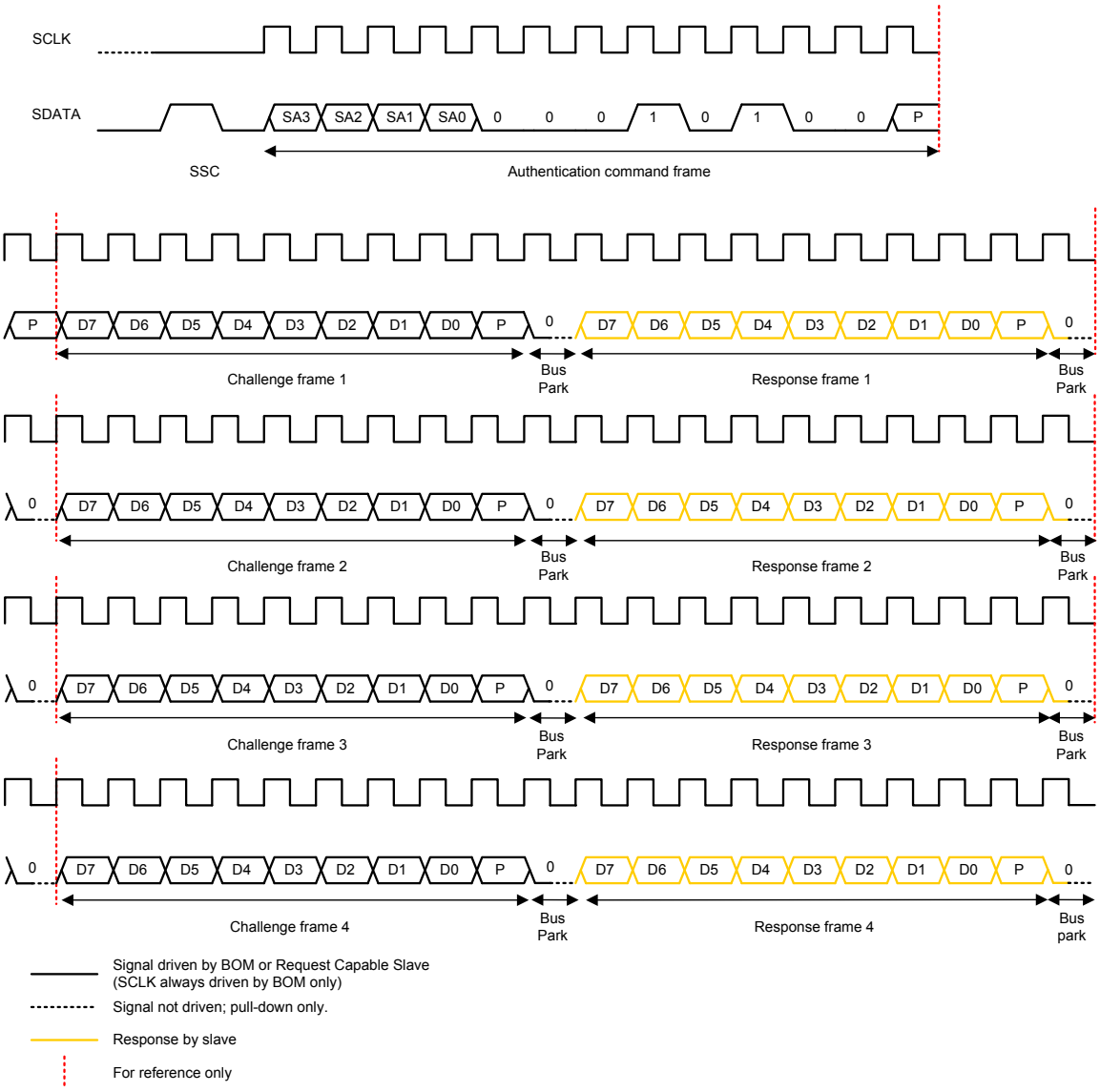


Figure 44 Authenticate Command Sequence

13.2.5 Master Write Command Sequence

The Master Write Command Sequence enables a Master or Request Capable Slave in a multi-Master system to write a register of another Master on the bus. The Command Sequence consists of three Frames: a Master write Command Frame, a Data Frame with register address and a Data Frame with write data. The

Command Sequence also includes the SSC, ACK/NACK and a Bus Park Cycle as shown in Figure 45. A Slave shall ignore this command.

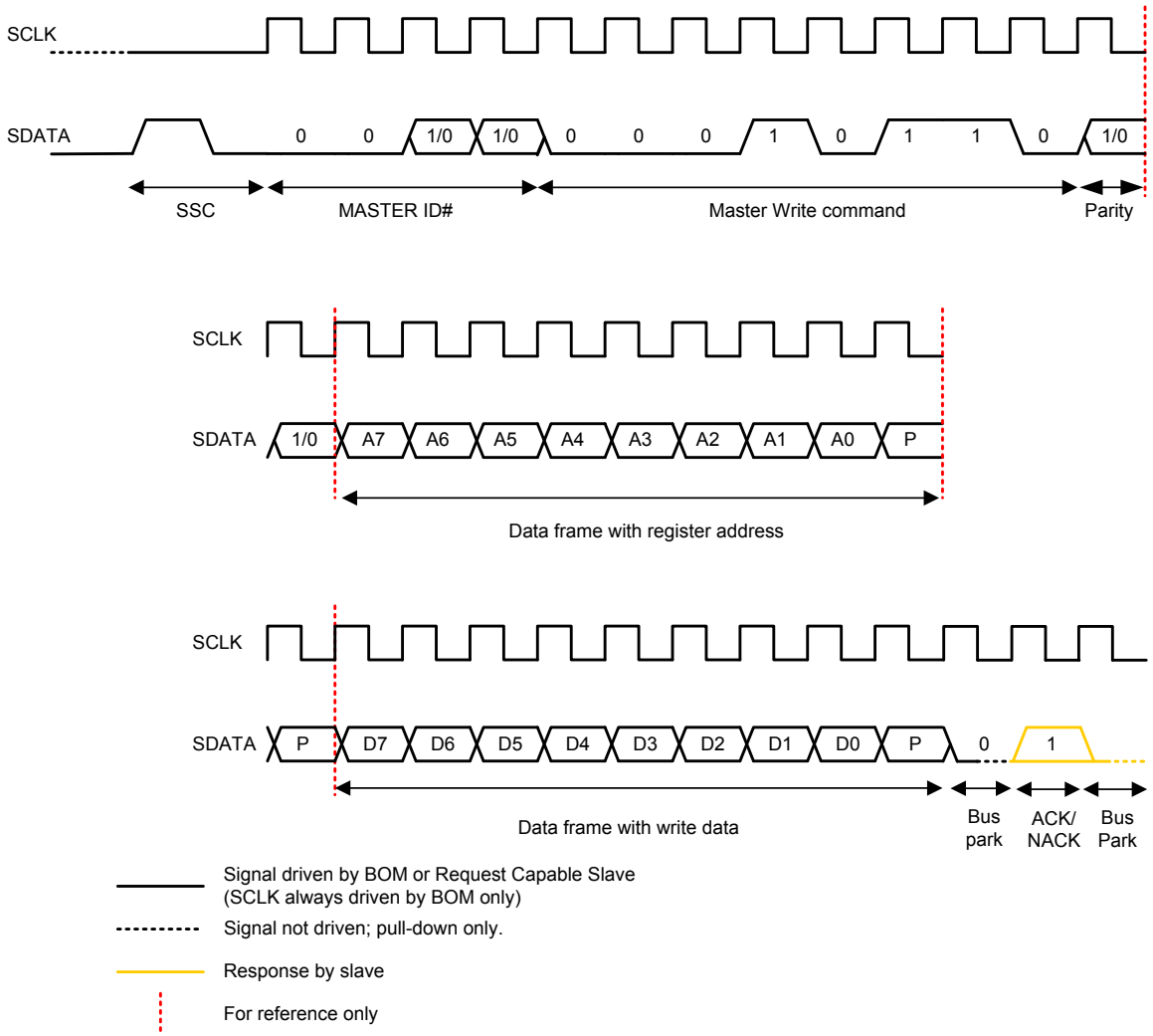


Figure 45 Master Write Command Sequence

13.2.6 Master Read Command Sequence

The Master Read Command Sequence enables a Master or Request Capable Slave to access the register space of another Master on the bus. The Command Sequence consists of three Frames: a Master read Command Frame, a Data Frame with a register address and a Data Frame from the other Master with the contents of the register. The Command Sequence also includes the SSC and Bus Park Cycles as shown in Figure 46. A Slave shall ignore this command.

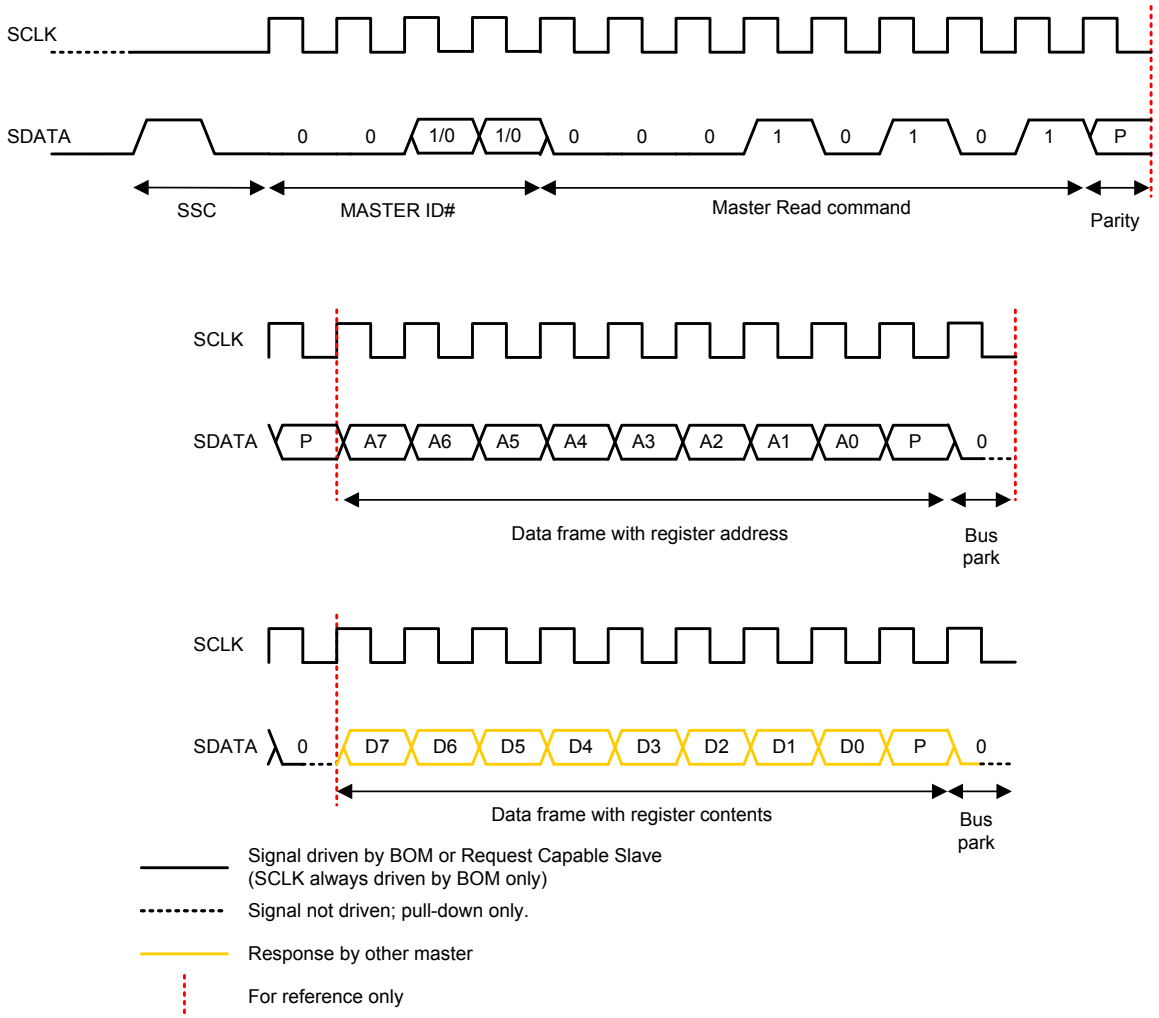


Figure 46 Master Read Command Sequence

13.2.7 Transfer Bus Ownership Command Sequence

The Transfer Bus Ownership (TBO) Command Sequence enables the transfer of bus ownership to another connected Master in the event that the BOM wants to disconnect from the bus. This Command Sequence is required for the Bus Owner Master to disconnect from the bus.

The Transfer Bus Ownership Command Sequence starts with an SSC followed by the TBO Command Frame and a Data Frame containing responses, if any, from other Master devices on the bus, and ends with a Bus Handover cycle as shown in Figure 47. When the BOM sends this command, and disconnects from the bus, it shall use the bus connecting procedure (see Section 9) prior to sending a new Command Sequence on the bus.

In response to the Transfer Bus Ownership Command Sequence, if a Master is connected, it shall raise SDATA during the correct SCLK cycle according to its MPL, and subsequently sends its MID on the following two SCLK cycles followed by zeros to complete the eight data slots, a parity bit, and a Bus Park Cycle. The first Master that drives SDATA to logic level one shall become the new Bus Owner Master, and sets its MPL to 3 (lowest priority). Masters that have not become the BOM shall not drive SDATA to logic level one during its SCLK cycle. Other Master MPLs are defined using the Round-Robin arbitration

algorithm. If there are no other Connected Masters on the bus, the Data Frame shall be a No Response Frame after which the BOM shall disconnect from the bus.

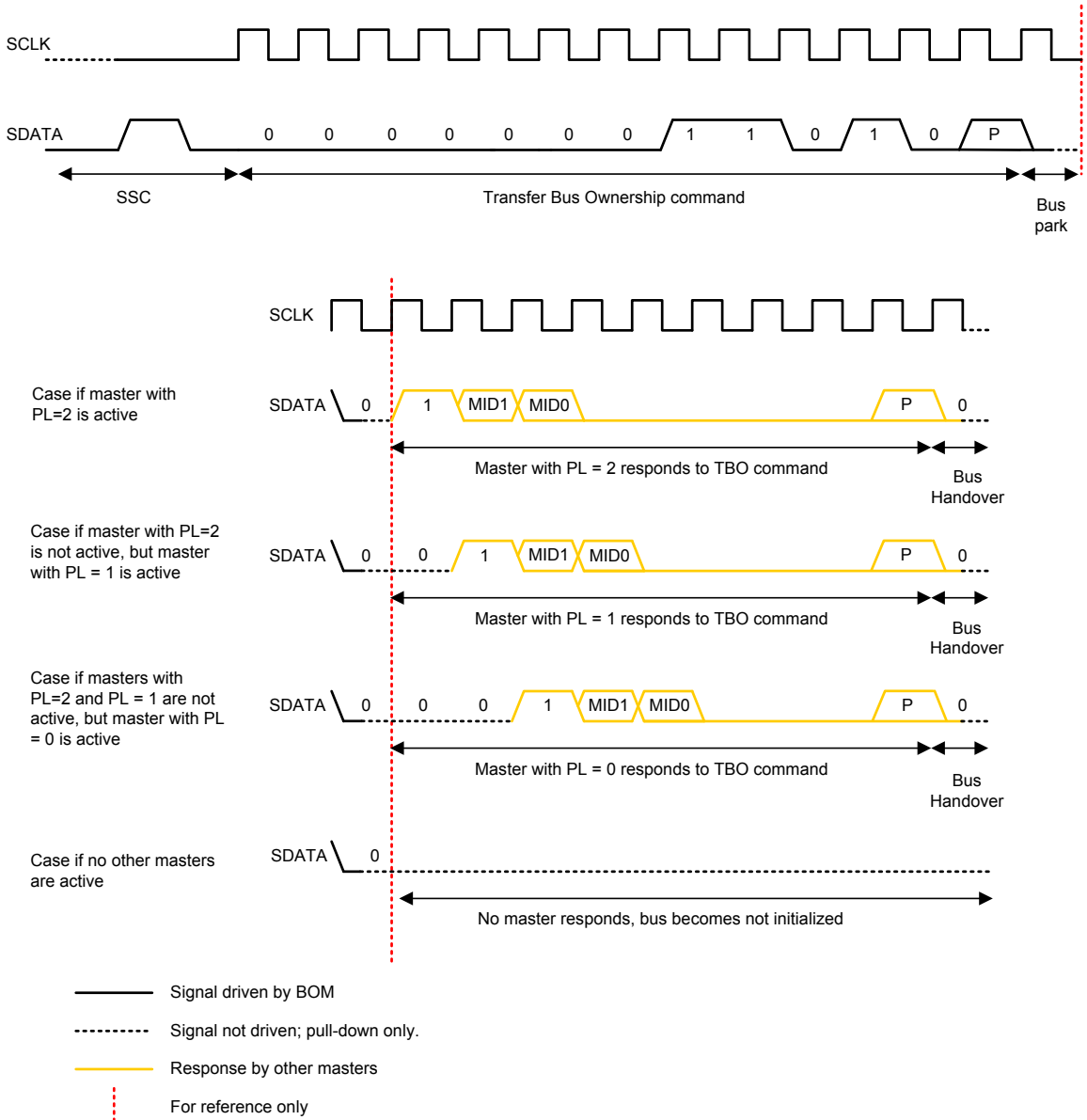


Figure 47 Transfer Bus Ownership Command Sequence

13.2.8 Device Descriptor Block Slave Read Command Sequence

The Device Descriptor Block Slave Read Command Sequence starts with an SSC followed by the Device Descriptor Block Slave Read Command Frame. The response from the Slave is ten bytes long. The bit order of the Device Descriptor Block is MSB to LSB as shown in Figure 48. *MIPI Alliance Specification for Device Descriptor Block (DDB)* [MIPI01] defines the contents of the Device Descriptor Block.

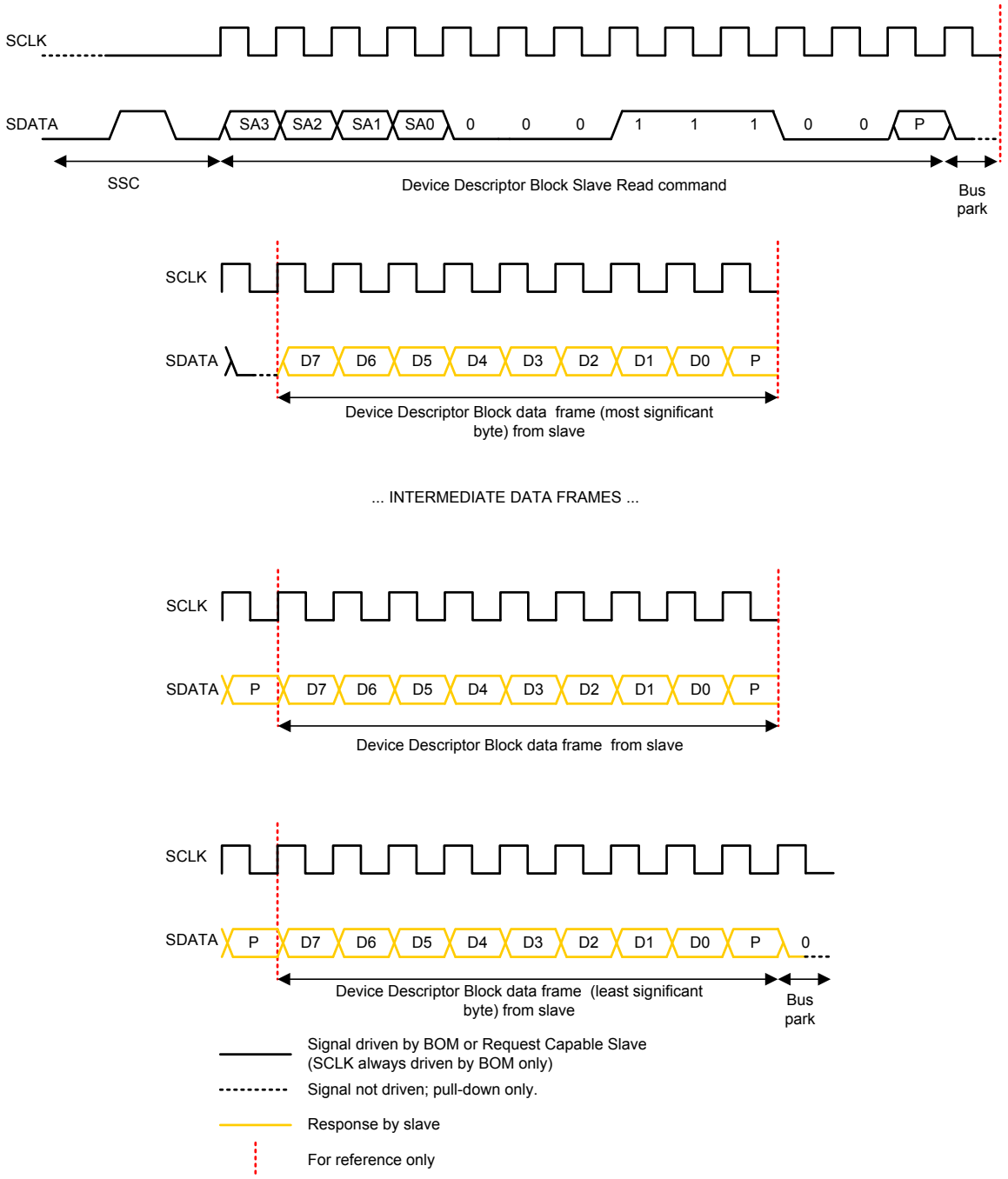


Figure 48 Device Descriptor Block Slave Read Command Sequence

13.2.9 Device Descriptor Block Master Read Command Sequence

The Device Descriptor Block Master Read Command Sequence starts with an SSC followed by the Device Descriptor Block Master Read Command Frame. The response from the Master is ten bytes long. The bit order of the Device Descriptor Block is MSB to LSB as shown in Figure 49. *MIPI Alliance Specification for Device Descriptor Block (DDB)* [MIPI01] defines the contents of the Device Descriptor Block.

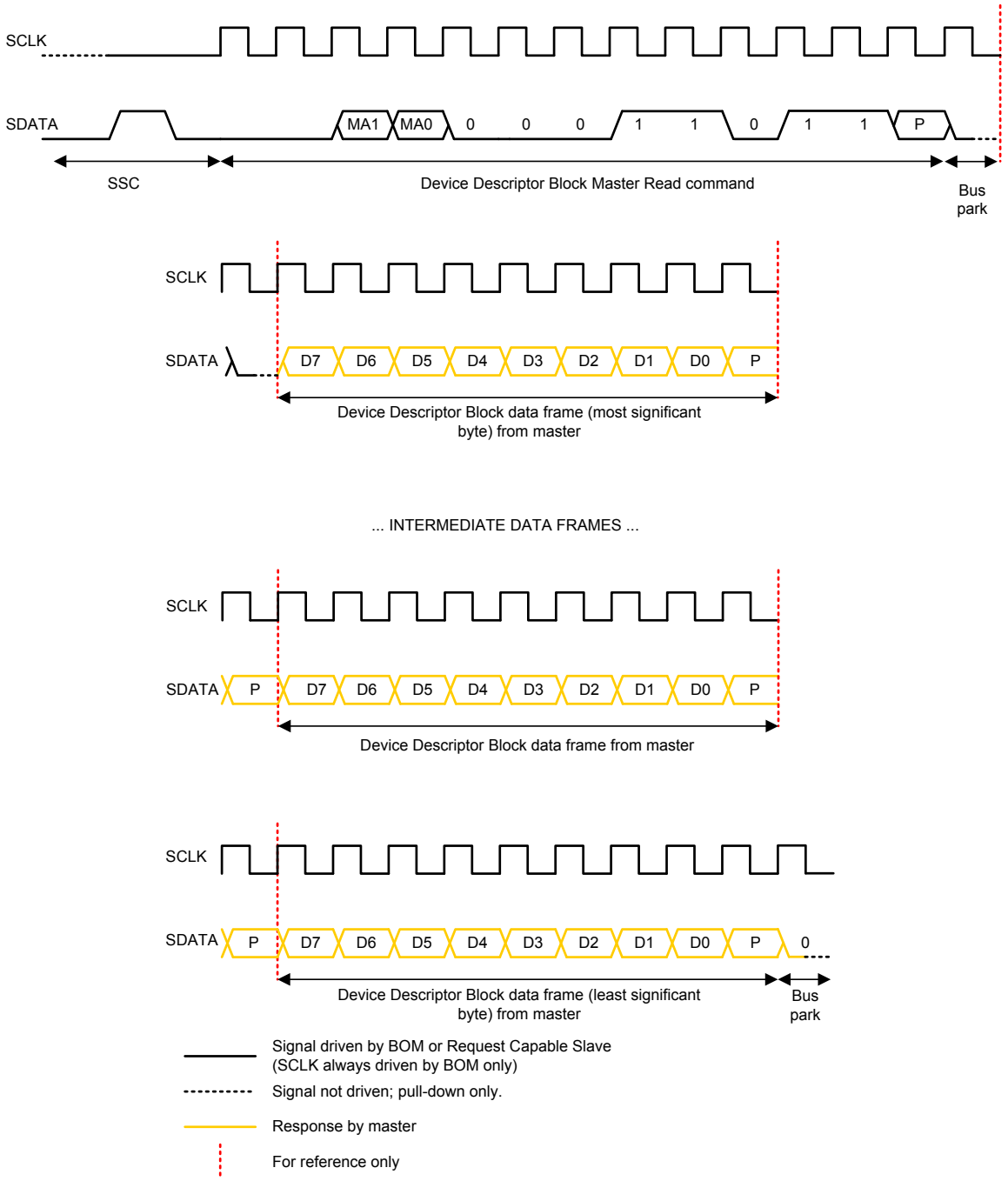


Figure 49 Device Descriptor Block Master Read Command Sequence

13.2.10 Extended Register Write Long Command Sequence

The Extended Register Write Long Command Sequence allows write access to the extended register space on a Slave device using a 16-bit address. One to eight bytes of data may be written in a single Command Sequence. The extended register address is supplied in two Address Frames within the Command Sequence. ACK/NACK shall be used to verify the completion of the entire Extended Register Write Long Command Sequence.

Figure 50 shows the Extended Register Write Long Command Sequence. The Extended Register Write Long Command Sequence starts with an SSC followed by the Extended Register Write Long Command Frame, two Address Frames and one or more Data Frames with the data to be written followed by ACK/NACK. The Command Sequence ends with a Bus Park Cycle.

The three LSBs of the Extended Register Write Command Frame, BC[2:0] in Figure 50, indicate the number of bytes to be written in the Command Sequence. BC2 is the byte count MSB. 0b000 indicates one byte shall be written and 0b111 indicates eight bytes shall be written.

The register address in the Command Sequence contains the address of the first extended register to be written. If more than one byte is written in a single Sequence then the Slave's local extended register address shall be automatically incremented by one for each byte written up to address 0xFFFF, starting from the address indicated in the Address Frame. If the extended register address reaches 0xFFFF before the last Data Frame in the Command Sequence then the content of the register at address 0xFFFF is overwritten with the overflow data.

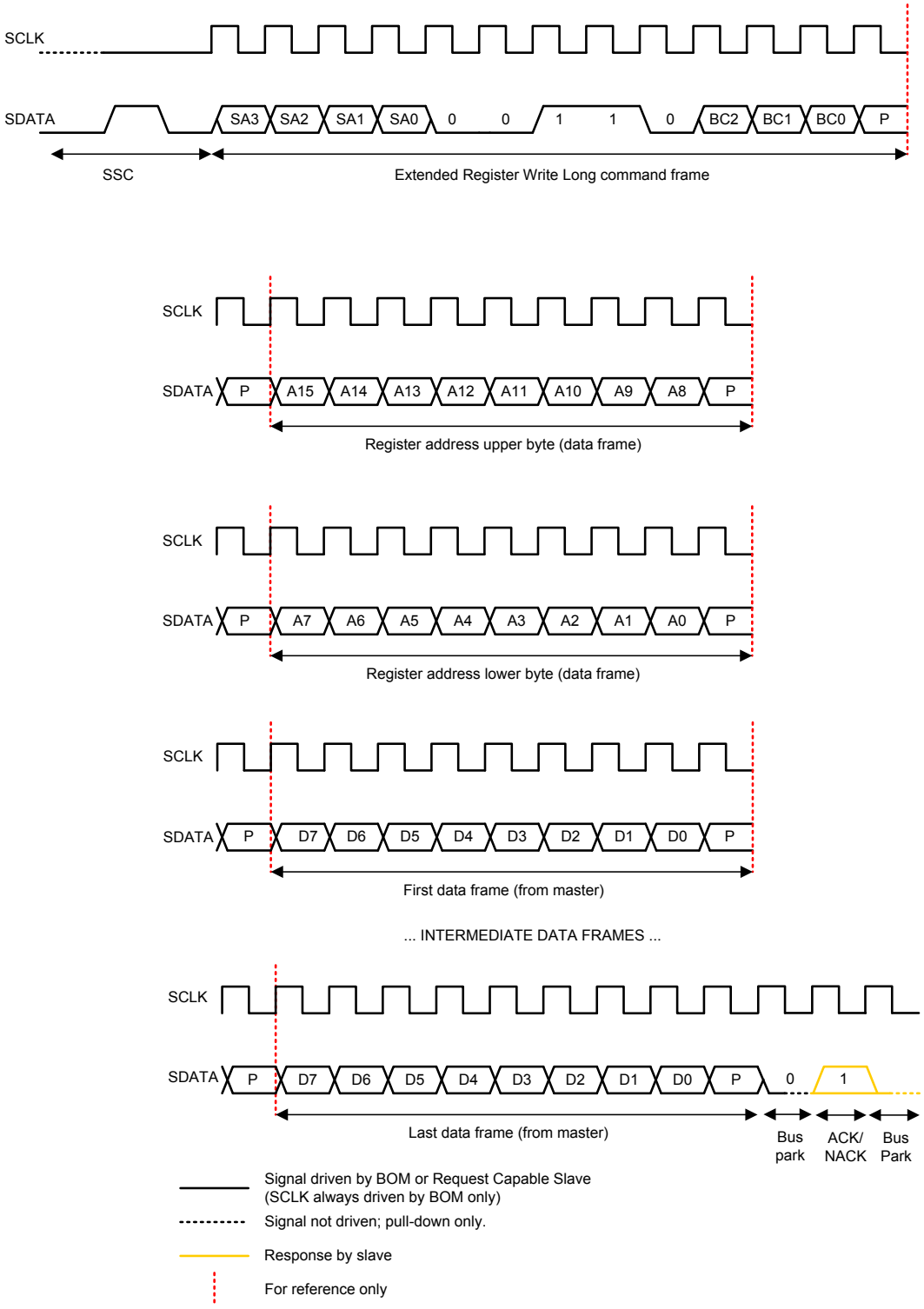


Figure 50 Extended Register Write Long Command Sequence

13.2.11 Extended Register Read Long Command Sequence

The Extended Register Read Long Command Sequence allows read access to the extended register space on a Slave device using a 16-bit address. One to eight bytes of data may be read in a single Command

Sequence. The extended register address shall be supplied in two Address Frames within the Command Sequence.

Figure 51 shows the Extended Register Read Long Command Sequence. The Command Sequence starts with the SSC followed by the Extended Register Read Long Command Frame, two Address Frames and one or more Data Frames with the data read from the Slave. A Bus Park Cycle occurs between the Address Frame and the Data Frames. The Command Sequence ends with a Bus Park Cycle.

The three LSBs of the Extended Register Read Long Command Frame, BC[2:0] in Figure 51, indicate the number of bytes to be read in the Sequence. BC2 is the byte count MSB. 0b000 indicates one byte shall be read and 0b111 indicates eight bytes shall be read.

The register address in the Command Sequence contains the address of the first extended register to be read. If more than one byte is read in a single Command Sequence then the Slave's local extended register address shall be automatically incremented by one for each byte read up to address 0xFFFF, starting from the address indicated in the Address Frame. If the extended register address reaches 0xFFFF before the last Data Frame in the Command Sequence then the content of the register at address 0xFFFF is read multiple times. An Extended Register Read command by the Master to an unsupported Slave extended register address results in a No Response Frame from the Slave. If the address that results from auto-incrementing the Slave's local extended register address is for an unsupported register then the Slave sends a No Response Frame to the Master in place of the Data Frame. The Command Sequence continues from the next extended register address.

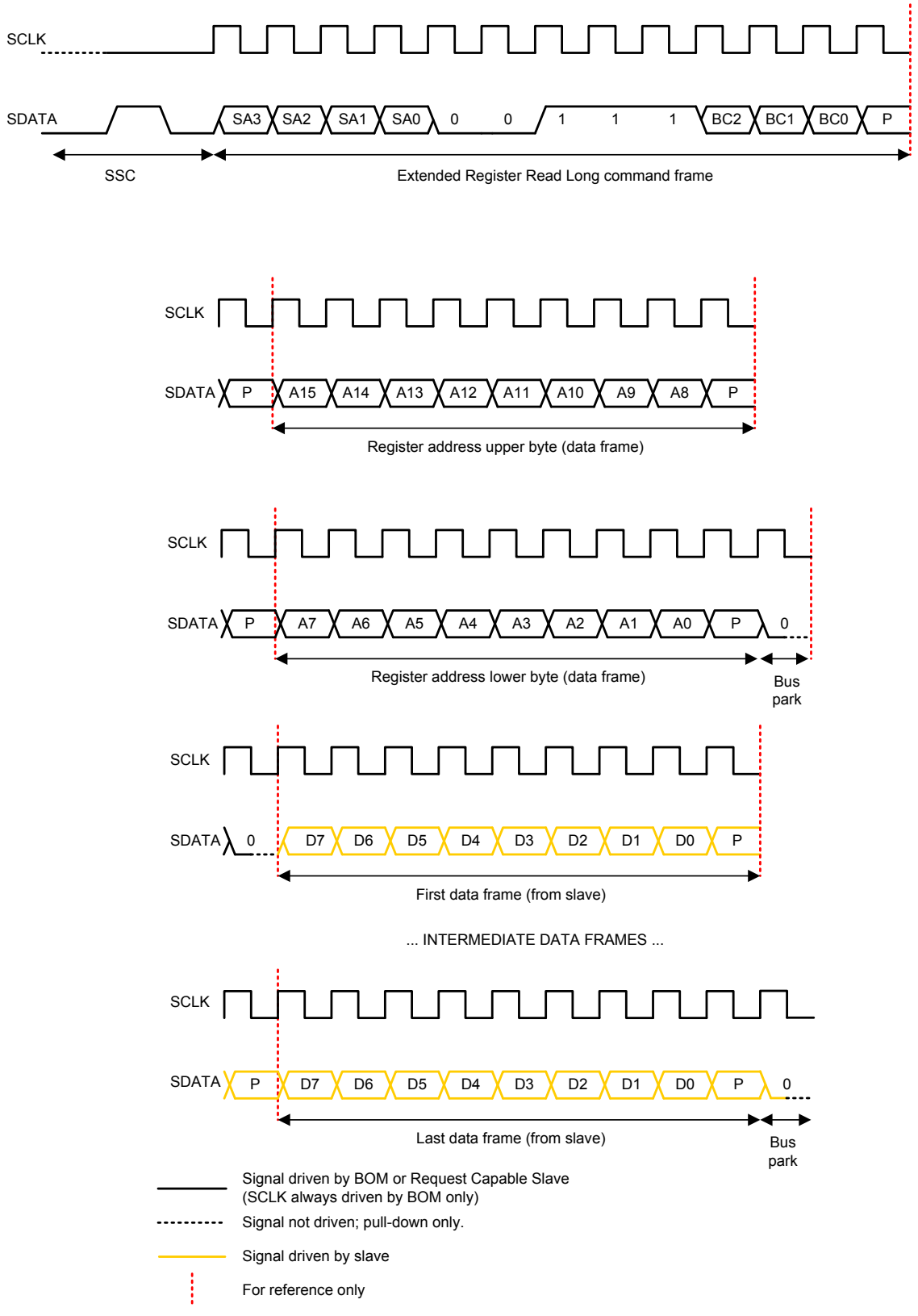


Figure 51 Extended Register Read Long Command Sequence

13.2.12 Register Write Command Sequence

Figure 52 shows the Register Write Command Sequence. The Command Sequence starts with an SSC, followed by the write Command Frame containing the Slave address and the target register address and a Data Frame containing the data to be written followed by ACK/NACK. The Command Sequence ends with a Bus Park Cycle.

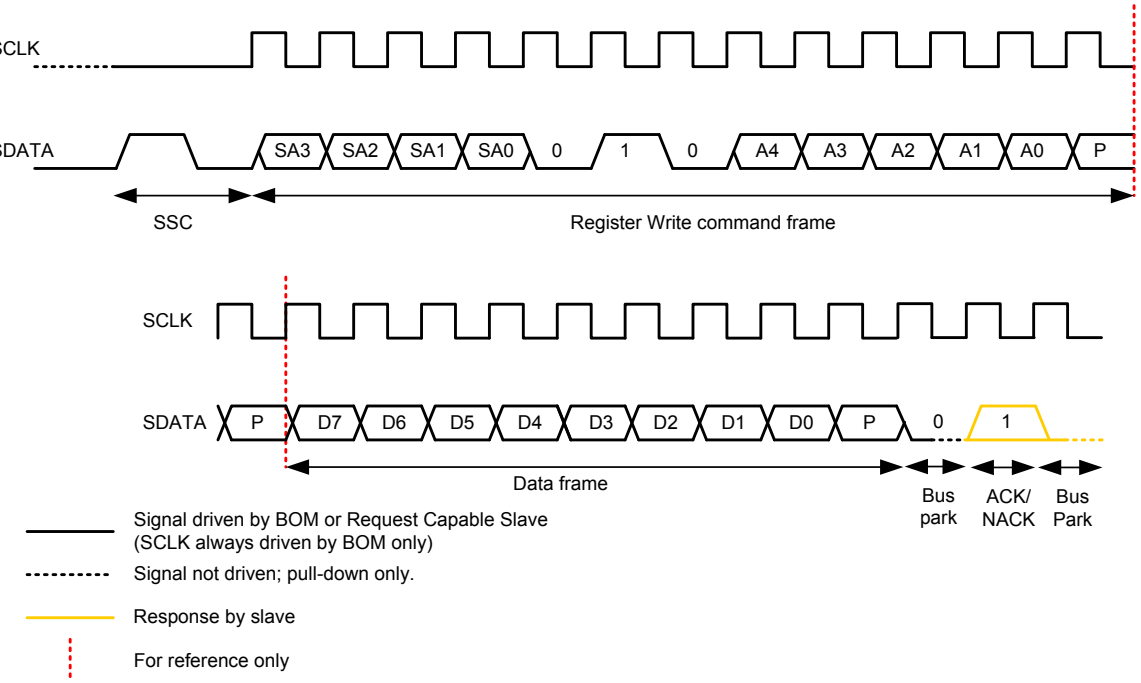


Figure 52 Register Write Command Sequence

13.2.13 Register Read Command Sequence

Figure 53 shows the Register Read Command Sequence. The Sequence starts with an SSC, followed by the read Command Frame, a one-clock cycle Bus Park Cycle and a Data Frame sent by the Slave device. The Command Sequence ends with another Bus Park Cycle.

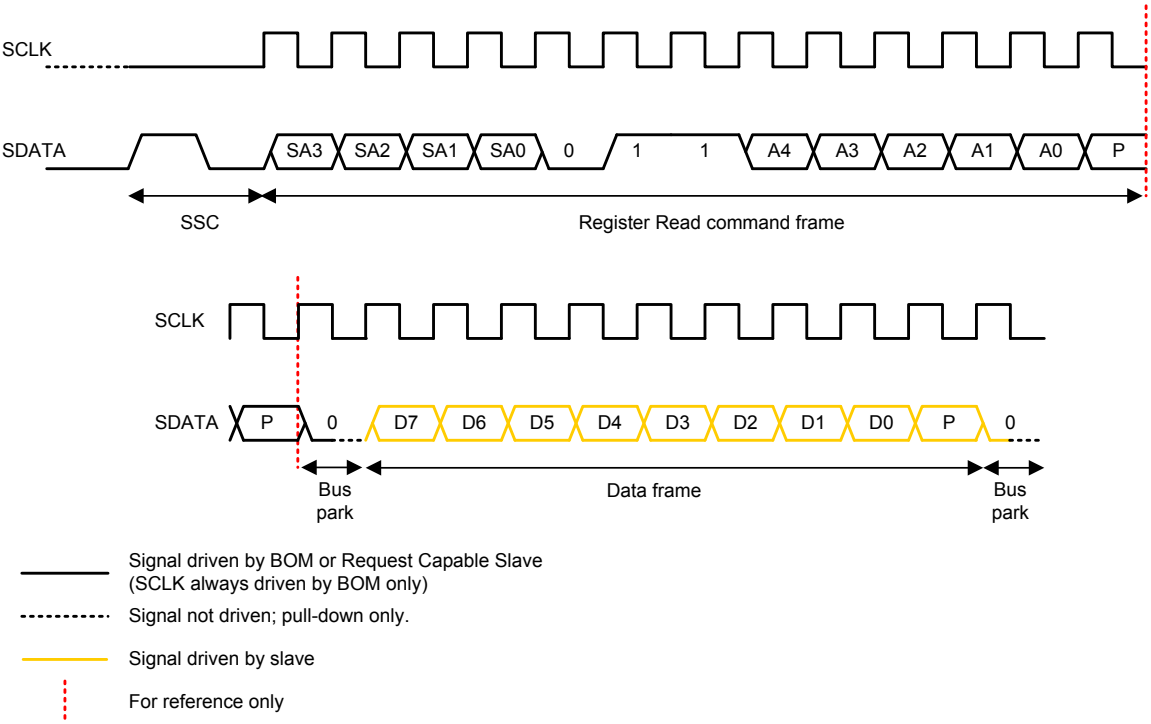


Figure 53 Register Read Command Sequence

13.2.14 Register 0 Write Command Sequence

Figure 54 shows the Register 0 Write Command Sequence. The Command Sequence starts with an SSC, followed by the Register 0 write Command Frame containing the Slave address, a logic one, and a 7-bit word to be written to Register 0. The Slave verifies receiving the correct data with ACK/NACK. The Command Sequence ends with a Bus Park Cycle.

This Command Sequence is used to support advanced power management functions that require repetitive Command Sequences to a fixed Slave register with minimal bandwidth consumption.

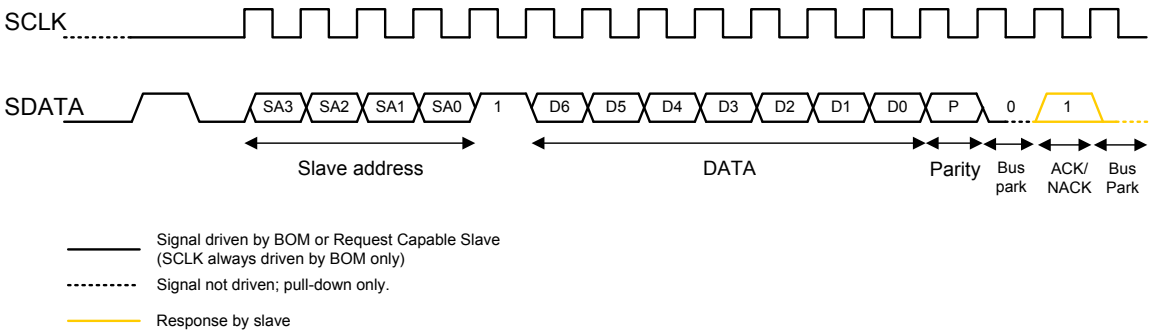


Figure 54 Register 0 Write Command Sequence

13.3 Error Handling

The SPMI protocol provides two means of error detection, parity check on each Frame and a check for invalid fields such as commands and addresses. The response of a device receiving either of these types of

error depends on whether the device is a Master or a Slave, and the Frame in which the error occurs. The following sections describe the error condition and the response of each device type to the error.

13.3.1 Parity Error in the Command Frame

If a Master receives a Sequence with an incorrect parity value in the Command Frame, it shall ignore the Command Sequence, disconnect from the bus and try to reconnect to the bus.

If a Slave receives a Command Sequence with an incorrect parity value in the Command Frame, it shall ignore the Command Sequence and reset its command decode logic on the next SSC.

There is no means to detect multiple bit errors that alias onto known commands with the correct parity, except that the number of SCLK cycles might not match with the expected length. In this situation, a Master shall consider itself disconnected from the bus and shall connect to the bus to send a Sequence as defined in Section 9.2. If a Slave receives a different number of SCLK cycles than it expects for a given Command Frame, it shall reset its command decode logic on the next SSC.

If a Parity Error occurs in the Command Frame sent by an RCS Device, the BOM shall stop the SCLK for $> T_{BT}$. The RCS Device shall detect that the SCLK has stopped for T_{BT}, shall reset itself and place its SDATA line in a High-Z state.

13.3.2 Unsupported Commands

If a Master receives a Command Sequence with an unsupported Command Frame, it shall ignore the command, disconnect from the bus and try to reconnect to the bus.

If a Slave receives a Command Sequence with an unsupported Command Frame, it shall ignore the Command Sequence and reset its command decode logic on the next SSC.

13.3.3 Parity Error in the Address Frame

If a Master receives a Command Sequence with an incorrect parity value in an Address Frame, it shall ignore the Command Sequence. Since the length of the Command Sequence is known, the Master shall wait until the Command Sequence completes the requisite number of SCLK cycles before initiating a new arbitration request. The Master is not required to disconnect from the bus. If the Address Frame parity error occurs in a Write Command Sequence, the Master shall inform the writing Device of the error using ACK/NACK.

If a Slave receives a Command Sequence with an incorrect parity value in the Address Frame, it shall ignore the Command Sequence. If the Address Frame parity error occurs in a Write Command Sequence, the Slave shall inform the writing Device of the error using ACK/NACK.

13.3.4 Parity Error in the Data Frame

If a Master receives a Command Sequence with an incorrect parity value in a Data Frame, it shall ignore the Command Sequence and use ACK/NACK to inform the writing Device of the error. Since the length of the Command Sequence is known, the Master shall wait until the Command Sequence completes the requisite number of SCLK cycles before initiating a new arbitration request. The Master is not required to disconnect from the bus.

If a Slave receives a Command Sequence with an incorrect parity value in a Data Frame, the Slave shall ignore all Data Frames after encountering the first Parity Error. The Slave may ignore the entire Command Sequence if a single Data Frame contains a Parity Error.

Address incrementing shall not be affected due to a Data Frame Parity Error for multi-byte write Command Sequences.

13.3.5 Unsupported Address

If a Master or Slave receives a read command with an unsupported address, the device shall send No Response Frames instead of Data Frames.

If a Master or Slave receives a write command with an unsupported address, the device shall ignore the data in the unsupported addresses and inform the writing Device of the error using ACK/NACK.

Annex A SPMI Reference (informative)

A.1 DC Operating Conditions

Table 17 SPMI DC Electrical Specifications

Description	Symbol	Min	Typ	Max	Unit
Number of Masters	N_{master}	1		4	
Number of Slaves	N_{slaves}	0		16	
Characterized Capacitive Bus Loading	C_{LOAD}	15		50	pF
Slave SDATA Pull Down Resistance	R_{DS}	0.5		2.0	MΩ
Slave SCLK Pull Down Resistance	R_{CS}	0.5		2.0	MΩ
Total System SDATA Pull Down Resistance	R_{DT}	0.125		2.0	MΩ
Total System SCLK Pull Down Resistance	R_{CT}	0.125		2.0	MΩ
Output Low Voltage	V_{OL}	0		0.2*VDD	V
Output High Voltage	V_{OH}	0.8*VDD		VDD	V
Positive Going Threshold Voltage	V_{TP}	0.4*VDD		0.7*VDD	V
Negative Going Threshold Voltage	V_{TN}	0.3*VDD		0.6*VDD	V
Hysteresis Voltage (V_{TP} - V_{TN})	V_H	0.1*VDD		0.4*VDD	V
SPMI Supply Voltage	VDD				
1.8 V signaling		1.65		1.95	V
1.2 V signaling		1.1		1.3	V

A.2 AC Operating Conditions

A.2.1 High Speed (HS) Device Class

Table 18 SPMI HS Device AC Specifications

Description	Symbol	Min	Typ	Max	Unit
SCLK Frequency (typical)	F_{SCLK}	0.032000		26	MHz
SCLK Period (1/FSCLK)	T_{SCLK}	38		32000	ns
SCLK Output High Time	T_{SCLKOH}	12			ns
SCLK Output Low Time	T_{SCLKOL}	12			ns
SCLK Output Transition Time (Rise / Fall)	$T_{SCLKOTR}$	2.1		5.3	ns
SDATA Output Transition Time (Rise / Fall)	$T_{SDATAOTR}$	2.1		5.3	ns
SDATA Output Valid Time	T_D	0		11	ns
SDATA Setup Time	T_S	1			ns
SDATA Hold Time	T_H	5			ns
SDATA Drive Release Time	T_{SDATAZ}			10	ns

Timing characterized as explained in Section 5.6.

A.2.2 Low Speed (LS) Device Class

Table 19 SPMI LS Device AC Specifications

Description	Symbol	Min	Typ	Max	Unit
SCLK Frequency	F_{SCLK}	0. 032000		15	MHz
SCLK Period ($1/F_{SCLK}$)	T_{SCLK}	66.67		31250	ns
SCLK Output High Time	T_{SCLKOH}	22			ns
SCLK Output Low Time	T_{SCLKOL}	22			ns
SCLK Output Transition Time (Rise / Fall)	$T_{SCLKOTR}$	2.1		8	ns
SDATA Output Transition Time (Rise / Fall)	$T_{SDATAOTR}$	2.1		8	ns
SDATA Output Valid Time	T_D	0		20	ns
SDATA Setup Time	T_S	2			ns
SDATA Hold Time	T_H	5			ns
SDATA Drive Release Time	T_{SDATAZ}			18	ns

Timing characterized as explained in Section 5.6.

A.2.3 Other Signaling Electrical Specifications

Table 20 defines SPMI Bus Timeout specification.

Table 20 SPMI Bus Timeout Specification

Description	Symbol	Min	Typ	Max	Unit
Bus Timeout Period	T_{BT}		96		µs

A.3 Command Sequence Reference (informative)

Mandatory Commands			Information			
M	RCS	NRCS	Hex	Description	SSC	Command Frame	Data Frame
Y	O	O	00 - 0F	Extended Register Write		SA[3:0] 0 0 0 0 BC[3:0] (0 0 0 0 – 1 1 1 1) P	Address[7:0] P Up to 16 Bytes of Data with Parity BP A/N BP
Y	O	O	10	Reset		SA[3:0] 0 0 0 1 0 0 0 0 P	BP A/N BP
Y	O	O	11	Sleep		SA[3:0] 0 0 0 1 0 0 0 1 P	BP A/N BP
Y	O	O	12	Shutdown		SA[3:0] 0 0 0 1 0 0 1 0 P	BP A/N BP
Y	O	O	13	Wakeup		SA[3:0] 0 0 0 1 0 0 1 1 P	BP A/N BP
Y	O	O	14	Authenticate		SA[3:0] 0 0 0 1 0 1 0 0 P	Challenge Data 1 [7:0] P BP Response Data 1 [7:0] P BP Challenge/Response 2,3,&4 BP
Y	O	N	15	Master Read		0 0 MA[1:0] 0 0 0 1 0 1 0 1 P	Address[7:0] P BP Data[7:0] P BP
Y	O	N	16	Master Write		0 0 MA[1:0] 0 0 0 1 0 1 1 0 P	Address[7:0] P Data[7:0] P BP A/N BP
			17	(Reserved)		0 0 0 1 0 1 1 1 P	Undefined
			18	(Reserved)		0 0 0 1 1 0 0 0 P	Undefined
			19	(Reserved)		0 0 0 1 1 0 0 1 P	Undefined
Y	N	N	1A	Transfer Bus Ownership		0 0 0 0 0 0 0 1 1 0 1 0 P	BP MP2 A* B* C* D* BP BP BP P BP
Y	O	N	1B	Device Descriptor Block Master Read		0 0 MA[1:0] 0 0 0 1 1 0 1 1 P	BP Data 9 [7:0] P 8 Bytes of Data + Parity Data 0 [7:0] P BP
Y	O	O	1C	Device Descriptor Block Slave Read		SA[3:0] 0 0 0 1 1 1 0 0 P	BP Data 9 [7:0] P 8 Bytes of Data + Parity Data 1 [7:0] P BP
			1D	(Reserved)		0 0 0 1 1 1 0 1 P	Undefined
			1E	(Reserved)	1 0	0 0 0 1 1 1 1 0 P	Undefined
			1F	(Reserved)		0 0 0 1 1 1 1 1 P	Undefined
Y	O	O	20 - 2F	Extended Register Read		SA[3:0] 0 0 1 0 BC[3:0] (0 0 0 0 – 1 1 1 1) P	Address[7:0] P BP Up to 16 Bytes of Data with Parity BP
Y	O	O	30 - 37	Extended Register Write Long		SA[3:0] 0 0 1 1 0 BC[2:0] (0 0 0 – 1 1 1) P	Address[15:8] P Address[7:0] P Up to 8 Bytes of Data with Parity BP A/N BP
Y	O	O	38 - 3F	Extended Register Read Long		SA[3:0] 0 0 1 1 1 BC[2:0] (0 0 0 – 1 1 1) P	Address[15:8] P Address[7:0] P BP Up to 8 Bytes of Data with Parity BP
Y	O	O	40 - 5F	Register Write		SA[3:0] 0 1 0 Address[4:0] (0 0 0 0 0 – 1 1 1 1 1) P	Data[7:0] P BP A/N BP
Y	O	O	60 - 7F	Register Read		SA[3:0] 0 1 1 Address[4:0] (0 0 0 0 0 – 1 1 1 1 1) P	BP Data[7:0] P BP
Y	O	O	80 - FF	Register 0 Write		SA[3:0] 1 Data[6:0] (0 0 0 0 0 0 0 – 1 1 1 1 1 1 1) P	BP A/N BP

Figure 55 Command Sequence Formats

Legend:

BC = Byte Count
MA = Master Address
NRCS = Non-Request Capable Slave
RCS = Request Capable Slave
BP = Bus Park Cycle
MPx = Master with Priority x
O = Optional
SA = Slave Address
M = Master
N = Not supported
P = Parity
Y = Mandatory
A/N = ACK/NACK

Participants

The following list includes those persons who participated in the Working Group that developed this Specification and who consented to appear on this list.

Atukula, Radha Krishna – Research in Motion Limited

Chun, Christopher – Qualcomm Incorporated

Hambro, Glen – IEEE-ISTO (staff)

Johnson, Robert C – IEEE-ISTO (staff)

Le Meur, Alice – Texas Instruments Incorporated

Pennanen, Juha – Texas Instruments Incorporated