

INTERNATIONAL STANDARD

ISO/IEC
15205
IEEE
Std 1496

First edition
2000-06

SBus – Chip and module interconnect bus



Reference number
ISO/IEC 15205:2000(E)
IEEE
Std 1496, 1993 Edition

Abstract: An input/output expansion bus with a 32- or 64-bit width is described in this standard. The SBus is designed for systems requiring a small number expansion ports. SBus Cards may be connected to a standard Sbus Connector mounted on the motherboard. SBus Devices may also be attached to the SBus directly on the system's motherboard. The dimensions of the SBus Card are 83,8 mm by 146,7 mm, making the cards appropriate for small computer systems that make extensive use of highly integrated circuits. The SBus Cards are designed to be installed in a plane parallel to the system's motherboard as mezzanine cards. They are designed to provide connections for devices external to the computer system through an exposed back panel. The form factor is useful in Futurebus+, VMEbus, desktop computers, and similar applications. The SBus has the capability of transferring data at rates up to 168 Mbytes/s, depending on the implementation options selected.

SBus Cards may either serve as Masters on the bus, providing all virtual address information as well as the data to be transferred, or they may serve as Slaves on the bus, providing data transfer according to the requirements of some other SBus Master. The SBus Master for a data transfer is selected by an arbitration process managed by the single SBus Controller on the SBus. The SBus Controller provides a virtual to physical address translation service.

Keywords: I/O bus, SBus, SBus Card, Standard for Boot Firmware.

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1993 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. First published in 1993.

ISBN 2-8318-5165-3

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

INTERNATIONAL STANDARD

ISO/IEC
15205
IEEE
Std 1496

First edition
2000-06

SBus – Chip and module interconnect bus

Sponsor

*Bus Architecture Standards Committee
of the IEEE Computer Society*



PRICE CODE **XA**

For price, see current catalogue

CONTENTS

	Page
FOREWORD	4
INTRODUCTION	5
Clause	
1 General.....	8
1.1 Scope and object	8
1.2 Normative references.....	8
2 Definitions, usage of special terms, acronyms, and editorial conventions	9
2.1 Definitions	9
2.2 Usage of special terms	13
2.3 Acronyms.....	13
2.4 Editorial conventions.....	13
3 Overview.....	14
3.1 System overview.....	14
3.2 Overview of configurations.....	16
3.3 General design information	19
3.4 Performance	21
4 Signal definitions	23
4.1 CLK signal	23
4.2 RST* signal	24
4.3 PA[27:0] signals.....	25
4.4 SEL* signal.....	25
4.5 AS* signal.....	26
4.6 BR* signal.....	26
4.7 BG* signal	26
4.8 D[31:0], D[63:0], and DP signals	27
4.9 SIZ[2:0] signals.....	28
4.10 RD signal.....	28
4.11 ACK[2:0]* signals.....	29
4.12 LERR* signal	31
4.13 INT[7:1]* signals	32
5 SBus cycle definitions	33
5.1 Arbitration Phase	33
5.2 Translation Phase	34
5.3 Extended Transfer Information Phase	36
5.4 Transfer Phase	40
5.5 Dual function SBus Devices.....	58
5.6 Exception conditions.....	59
5.7 Extended Transfer locking protocol.....	60

Clause	Page
6 SBus electrical requirements.....	62
6.1 Power	62
6.2 Electronic characteristics	63
6.3 Electronic timing requirements	66
6.4 Compliance requirements	69
7 Environmental requirements.....	69
7.1 Operating range.....	69
8 Mechanical requirements	70
8.1 SBus Slot Connector.....	70
8.2 SBus Card	74
8.3 Panel installation	88
9 SBus program interface	89
9.1 Introduction.....	89
9.2 Program format and interpretation	89
9.3 Required FCode attributes	90
9.4 FCode language	90
9.5 Special functions of Word 0	90
Annex A (informative) Compliance checklist	92
Annex B (informative) Known implementation variations.....	97
Bibliography	101
Index	102

SBus – CHIP AND MODULE INTERCONNECT BUS

FOREWORD

- 1) ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.
- 2) In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.
- 3) Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 15205 was prepared by subcommittee 26: Microprocessor systems, of ISO/IEC joint technical committee 1: Information technology.

International Standards are drafted in accordance with ISO/IEC Directives, Part 3.

Annexes A and B are for information only.

This standard must be used in conjunction with the latest edition of the following standard:
IEEE Std 1275.



International Electrotechnical Commission • 3, rue de Varembe, PO Box 131,
CH-1211-Geneva 20, Switzerland • Telephone: +41 22 919 0211 •
Telefax: +41 22 919 0300 • e-mail: inmail@iec.ch •
URL: <http://www.iec.ch>

INTRODUCTION

(This introduction is not a normative part of ISO/IEC 15205:2000, but is included for information only.)

This IEEE standard documents the implementation of the popular SBus interface. The SBus, originally developed and documented by Sun Microsystems as an I/O expansion bus, uses a standard form factor SBus Card that is a suitable size for the use of VLSI circuits in small computers. It has a high bandwidth and is capable of data transfer 8, 16, 32, or 64 bits in width. This standard includes the set of functionality originally documented by the *SBus Specification B.0* (Sun Microsystems Part #800.5922-10, Revision A, December 1990) and clarifies, corrects, and extends that functionality as required. The IEEE P1275 Working Group is developing a standard for boot firmware, which will define and document the initialization and boot interface for SBus Cards.

Special thanks are due to Bob Snively (P1496 Working Group draft technical editor) for the many hours spent in converting this document from the original *SBus Specification B.0* and editing it into its final form. Also deserving of thanks are Jim Lyle (P1496 Working Group vice Chair), Barbara Vance (P1496 Working Group former Secretary), Bob Gianni (P1496 Working Group Secretary), and Steve Hix (P1496 draft document editor) for their support in the Committee work and the generation of this document.

The following people were members of the P1496 Working Group that approved the draft for submission to IEEE for sponsor ballot:

Wayne Fischer, Chair

James Lyle, Co-Chair

Robert Gianni, Secretary

Robert Snively, Draft Technical Editor

Sanjay Adkar
Steven W. Aiken
Ray S. Alderman
Ravi Anantharaman
James Antonellis
Tom Armbruster
Jon K. Bennett
C.J. Beynon
Paul Borrill
Mike Chastain
Gary Croak
Scott Eichmann
Robert Elliott
Jurgen Fey
Larry Fiedler
William A. Fox
Paul Fulton
Brad Giffel

Steve Golson
Anthony A. Goodloe
James N. Hardage, Jr.
Hans Heilborn
Kai Holz
Timothy Hu
Mohammad Issa
Shinkyō Kaku
Kuljeet Kalkat
Thomas Kappler
Bill Keshlear
Gary Kidwell
Erik Kristenson
Ernst H. Kristiansen
Joel Libove
James Ludemann
Susan Mason
Shrenik Mehta
Donald J. Murphy

Elwood Parsons
Heinz Piorunneck
Jack Regula
Eayne Rickard
Michael Saari
Siamak Salimpour
Gary Sloane
Martin Sodos
Richard Spratt
Mike Strang
Lars Themsjö
David Therrien
Istvan Vadasz
Barbara Vance*
Naor Wallach
Eike Waltz
Leo Yuan
Janusz Zalewski

* Former Secretary

The following persons were on the balloting committee:

Amir Abouelnaga	Larry E. Gerald	Richard Mueller
Edward W. Aichinger	Robert R. Gianni	Michael Orlovsky
Ray S. Alderman	Steve Golson	Mira Pauker
Richard P. Ames	Julio Gonzalez-Sanz	Philip K. Piele
Keith D. Anthony	John Griffith	Rochit Rajsuman
Behrooz Bandall	Hans H. Heilborn	Brian Ramelson
Chris Bezirtzoglou	Zoltan R. Hunor	Gary Sloane
Michael L. Bradley	Edgar Jacques	Bob Snively
Scott M. Buck	David V. James	Michael Teener
Steven Cobb	Horace Jones	Joseph P. Trainor
Robert Crowde	W. Frederick Ki	Robert Tripi
Doug Degroot	Ernst H. Kristiansen	Rudolf Usselmann
Dante Del Corso	Lak Ming Lam	Clarence M. Weaver
Samuel Duncan	Gerry Laws	Michael Wenzel
Wilhelm P. Evertz	Boon Lum Lim	Andrew Wilson
Wayne Fischer	Marlyn Miner	Robert J. Wood
Chee K. Fong	William E. Molyneaux	David L. Wright
Joseph D. George		Oren Yuen

When the IEEE Standards Board approved this standard on June 17, 1993, it had the following membership:

Wallace S. Read, *Chair* **Donald C. Loughry, *Vice Chair***
Andrew G. Salem, *Secretary*

Gilles A. Baril	Ben C. Johnson	T. Don Michael*
Clyde R. Camp	Walter J. Karplus	Marco Migliaro
Donald C. Fleckenstein	Lorraine C. Kevra	L. John Rankine
Jay Forster*	E. G. Al Kiener	Arthur K. Reilly
David F. Franklin	Ivor N. Knight	Ronald H. Reimer
Ramiro Garcia	Joseph L. Koepfinger*	Gary S. Robinson
Donald N. Heirman	D. N. Jim Logothetis	Leonard L. Tripp
Jim Isaak		Donald W. Zipse

* Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
James Beall
Richard B. Engelman
David E. Soffrin
Stanley Warshaw

Paula M. Kelty
IEEE Standards Project Editor

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying all patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (508) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

SBus – CHIP AND MODULE INTERCONNECT BUS

1 General

1.1 Scope and object

SBus is a high performance computer I/O interface for connecting integrated circuits and SBus Cards to a computer system motherboard. This standard defines the mechanical, electrical, environmental, and protocol requirements for the design of SBus Cards and the computer system motherboard that supports those cards.

Every SBus Card shall implement appropriate self-descriptive and initialization firmware using FCode, which is similar to the Forth programming language. The details of this firmware standard are beyond the scope of this standard.¹⁾ In addition, other software interfaces may be used for communication with SBus Cards.

SBus is intended to provide a high performance I/O bus interface with a small mechanical form factor. The small size, high levels of integration, and low power usage of SBus Cards enable them to be used in laptop computers, compact desktop computers, and other applications requiring similar characteristics. SBus Cards are mounted in a plane parallel to the motherboard of the computer system, allowing the computer system to have a low profile. SBus is not designed as a general purpose backplane bus.

SBus allows transfers to be in units of 8, 16, 32, or 64 bits. Burst transfers are allowed to further improve performance. SBus allows a number of SBus Master devices to arbitrate for access to the bus. The chosen SBus Master provides a 32-bit virtual address which the SBus Controller maps to the selection of the proper SBus Slave and the development of the 28-bit physical address for that Slave. The selected SBus Slave then performs the data transfers requested by the SBus Master. Simple SBus Cards may be designed to operate solely as Slaves on the SBus.

1.2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this International Standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of IEC and ISO maintain registers of currently valid International Standards.

IEEE Std 1275:1994, IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices²⁾

¹⁾ A firmware interface standard is under consideration.

²⁾ IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (standards.ieee.org/).

2 Definitions, usage of special terms, acronyms, and editorial conventions

2.1 Definitions

For the purposes of this standard the following definitions apply.

2.1.1

assert

- a) for a single signal: to drive a signal to the one (1), or asserted, logic state.
- b) for a set of parallel signals of the same function: to place the desired logic state pattern on the bus, which may include both one and zero values

2.1.2

byte

set of eight bit-parallel signals corresponding to binary digits operated on as a unit

The most significant bit carries index value 7 and the least significant bit carries index value 0.

2.1.3

byte Slave

SBus Slave having a data path only through bits D[31:24] of the data bus

2.1.4

Bus Sizing

the dynamic modification of the data transfer width to meet the SBus Slave's bus width requirements [see 5.4.6]

2.1.5

CLK

a fixed-frequency clock signal; the main SBus timing signal

2.1.6

clock cycle

one period of the CLK signal, beginning with the rising edge of the signal and ending on the following rising edge of the signal

2.1.7

Controller

see **SBus Controller**

2.1.8

central processing unit (CPU)

describes that part of a computer that does the primary computational functions; loosely describes the computer system other than connected input and output devices

2.1.9

double-word

eight bytes or 64 bits operated on as a unit.

The most significant byte carries index value 0 and the least significant byte carries index value 7.

2.1.10

half-word

two bytes or 16 bits operated on as a unit

The most significant byte carries index value 0 and the least significant byte carries index value 1.

2.1.11

half-word Slave

an SBus Slave having a data path only through bits D[31:16] of the data bus

2.1.12

high (H) level

a signal voltage within the more positive (less negative) of the two ranges of logic levels chosen to represent the logic states

2.1.13

holding amplifier

receiver circuit incorporating feedback that maintains the present input logic level in the absence of any other drive signals on the signal line

2.1.14

logic state

one of two possible abstract states that may be taken on by a binary logic variable

See **one**, **zero**, **assert**, **negate**, **signal state**.

2.1.15

logic level

any level within one of two non-overlapping ranges of values of voltage used to represent the logic states

See **high**, **low**.

2.1.16

low (L) level

a signal voltage within the more negative (less positive) of the two ranges of logic levels chosen to represent the logic states

2.1.17

mandatory

The referenced item is required to claim compliance with this standard.

2.1.18

Master

See **SBus Master**.

2.1.19

motherboard

the printed circuit board on which an SBus Card is mounted through the connectors specified by this standard

2.1.20

negate

to drive a signal or a parallel set of signals to the zero logic state

2.1.21

odd parity

within a field or set of fields, an odd number of bits having the logical state of one; the exclusive-OR of all the bits being checked has the value of 1

2.1.22

one ("1")

a true logic state or a true condition of a variable

2.1.23

optional

The referenced item is not required to claim compliance with this standard. Implementation of an optional item should be as defined in this standard.

2.1.24

reserved

the term used for signals, bits, fields, and code values that are set aside for future standardization

2.1.25

SBus

- a) the correct spelling of the noun describing the bus defined by this standard
- b) the name for the Chip and Module Interconnect Bus described by this standard

2.1.26

SBus Card

a physical printed circuit assembly that conforms to the single-width or double-width mechanical specifications; meets the connector, power, and signal assignment requirements of this standard and contains one or more SBus Devices

2.1.27

SBus Controller

the SBus Device that performs all the centralized services for the SBus, including bias circuitry, arbitration, and address translation for SBus Masters, and selection of and time-outs for SBus Slaves

2.1.28

SBus cycle

one complete operation on the SBus, consisting of a set of phases beginning with an optional Arbitration Phase and progressing through the optional Translation Phase, the optional Extended Transfer Information Phase, and the Transfer Phase

2.1.29

SBus Device

a set of circuitry complying with the electrical and protocol requirements of the SBus and properly implementing all the signals of the SBus

An SBus Device may reside on the computer motherboard or it may be on an SBus Card. See **SBus Controller**, **SBus Master**, **SBus Slave**.

2.1.30

SBus Master

the SBus Device that requests data transfers to be performed by an SBus Slave

2.1.31

SBus Master port

in an SBus Device that combines both an SBus Master and an SBus Slave, the circuitry that is associated with the SBus Master

2.1.32

SBus Slave

the SBus Device providing the function of performing the data transfers requested by an SBus Master; the address space for the data transfers may contain data, control registers, or sense registers

2.1.33

SBus Slave port

in an SBus Device that combines both an SBus Master and an SBus Slave, the circuitry that is associated with the SBus Slave

2.1.34

SBus Slot

the location on a computer motherboard in which an SBus Card may be installed

The SBus Slot has the connector, the electrical characteristics, and the physical volumes and dimensions that are required by this standard.

2.1.35

SBus Specification

SBus Specification B.0 [B1]¹⁾, an earlier specification of SBus, now superseded by IEEE Std 1496

2.1.36

SBus standard

IEEE Std 1496, IEEE Standard for a Chip and Module Interconnect Bus: SBus

2.1.37

SBus System

a computer system containing a motherboard with at least an SBus Controller and some combination of zero or more SBus Slots which may be populated with SBus Cards

The SBus System may additionally have SBus Devices integrated on the motherboard. The SBus System includes the electronic, powering, cooling, and mechanical support functions required by the installed SBus Devices and SBus Slots.

2.1.38

signal assertion

- a) the act of driving a signal to the true state
- b) the act of driving a bus of signals to the correct pattern of ones and zeros

2.1.39

signal negation

the act of driving a signal to the false state

2.1.40

signal release

the act of removing electronic drive to a signal thereby placing the driver in a high-impedance condition

Release of an SBus signal will leave the SBus signal in its last state unless the signal is specified to have bias circuits attached that return the signal to a specified state.

2.1.41

signal state

logic state

2.1.42

Slave

See **SBus Slave**.

¹⁾ The numbers in brackets preceded by the letter B correspond to those of the bibliography in annex A.

2.1.43

word

four bytes or 32 bits operated on as a unit

The most significant byte carries index value 0 and the least significant byte carries the index value 3.

2.1.44

zero

a false logic state or a false condition of a variable

2.2 Usage of special terms

The use of special terms in this standard is explained here to prevent possible confusion:

2.2.1

shall

used when stating mandatory requirements of the standard

2.2.2

should

used when stating recommendations that are understood to be advisory

2.2.3

may

used to indicate optional directives or features

2.3 Acronyms

CPU central processing unit

LSB least significant bit

MMU memory management unit

MSB most significant bit

2.4 Editorial conventions

The following editorial conventions are used in this standard.

2.4.1 Signal names

A signal name followed by a number range in square brackets represents a set of logically related signals. The first number in the range indicates the most significant bit. As an example, D[31:0] describes the 32 bits of the data bus signals, with bit 31 being the most significant bit and bit 0 being the least significant bit.

An asterisk is appended to a signal name to indicate that its asserted or 1 state is present when the signal line's voltage is at its more negative logic level. The absence of an asterisk indicates that the signal is asserted or "1" when the signal line's voltage is at its more positive logic level. As an example, SEL* is asserted or "1" when the signal is at its more negative logic level.

2.4.2 Numbers

Values of signal names, including the value of a range of logically related signals grouped by a bracket, will be indicated as binary values with a binary character for each signal in the range. As an example, the notation PA[2:0] = 110 indicates that PA[2] has a 1 binary value, PA[1] has a 1 binary value, and that PA[0] has a 0 binary value.

All other numeric values are decimal values except where another number base is explicitly identified by the text.

2.4.3 Physical dimensions

Physical dimensions are indicated first in millimeters, then followed by the same dimension in square brackets specified in inches. In cases where only one dimension is supplied, it is assumed to be in millimeters. The normative dimension is the metric dimension. Dimensions that are provided for reference only are enclosed in parentheses. Reference dimensions represent a usual or typical dimension, but may be varied to meet special requirements.

3 Overview

3.1 System overview

An SBus System is a computer system containing a motherboard with at least an SBus Controller and a set of SBus signals connecting the SBus Controller to some number of SBus Masters and SBus Slaves mounted on the motherboard and/or to some number of standard SBus Slots. The SBus Devices mounted in the SBus Slots are on SBus Cards having standard mechanical dimensions. The SBus Controller provides a set of services used by all SBus Devices, both Masters and Slaves. SBus Masters use the arbitration, translation, and monitoring services of the SBus Controller in transferring information to and from SBus Slaves. A complete information transfer operation between a Master and a Slave is an SBus cycle.

SBus Masters initiate each SBus cycle by performing an Arbitration Phase with the SBus Controller. When the SBus Controller grants a particular SBus Master access to the SBus, the Master provides a virtual address for the data transfer as well as information about the data transfer size and direction to the SBus Controller. The Controller translates the virtual address to identify and select the correct SBus Slave and to generate the proper physical address for that Slave during the Translation Phase. The SBus Slave then transfers the requested data to or from the Master during the Transfer Phase.

The Transfer Phase consists of a single transfer or consists of a Burst Transfer sending or receiving several transfers in the same phase. An SBus Master can use the Extended Transfer Information Phase to request that the transfers be 64 bits wide. The Extended Transfer Information Phase also can lock the Slave to the Master for more than one SBus cycle. If the Transfer Phase is a single transfer, the width of the transfer can be regulated by the SBus Slave using the Bus Sizing mechanism. If the selected SBus Slave cannot service a request immediately, the Slave can request that the Master try the request at a later time using a Retry Acknowledgment.

Figure 1 shows a block diagram of an SBus System with one attached SBus Card.

3.1.1 SBus Master

SBus Masters request the necessary read and write transfers to execute the desired functions. The SBus System CPU usually controls at least one SBus Master directly. Other SBus Masters are managed by internal programming or by associated SBus Slaves that receive instructions from the host CPU through the CPU's SBus Master. Each SBus Master gains control of the SBus using the Arbitration Phase, then orders the SBus to perform the required functions.

The SBus Master may respond to SBus Slave acknowledgments that indicate that the SBus Slave requires different sizes of transfer by subsequently performing the special sizes of transfer required by the SBus Slave.

The SBus Master monitors the successful completion of a transfer and aborts or retries the transfer as required by the standard.

SBus Masters may be integrated with the SBus Controller function. In this case, the SBus Master and SBus Controller often implement their own arbitration and translation functions using internal interfaces that are not visible to other SBus Devices on the SBus.

Any SBus Master is able to communicate with any SBus Slave on the same SBus. No limitations restrict SBus Masters to operations into and out of system memory alone. A Master may perform SBus cycles between itself and an SBus Slave in another SBus Slot or in the same slot.

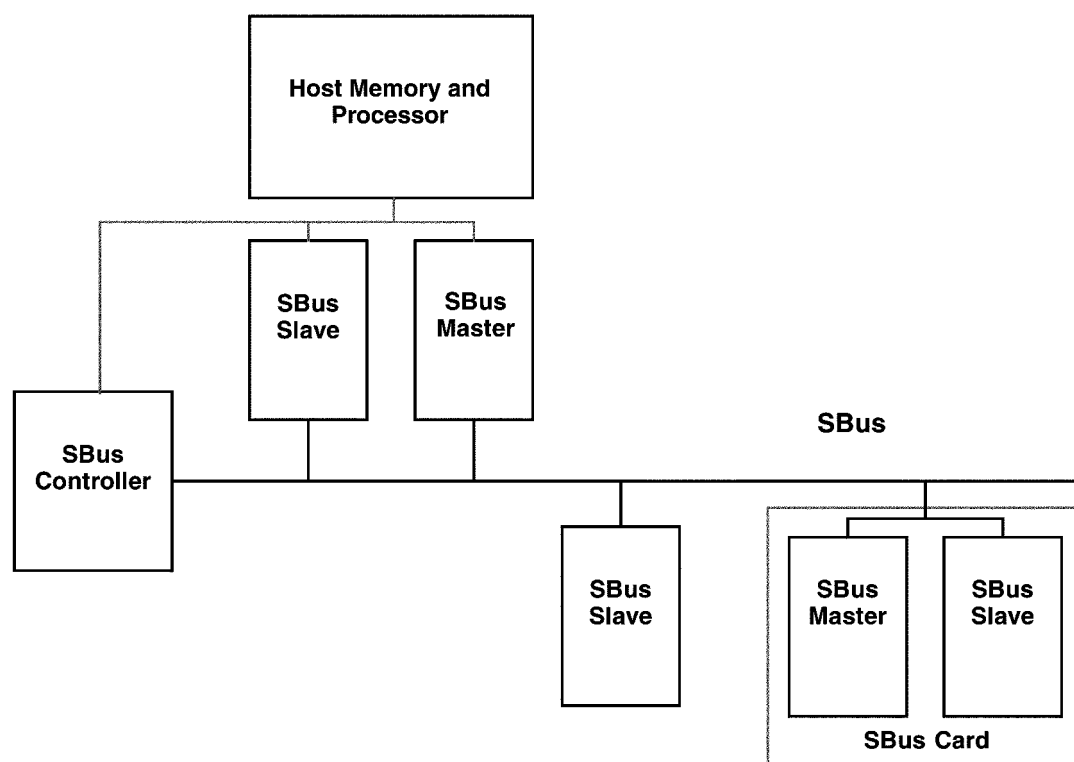


Figure 1 – Block diagram of representative SBus System

If multiple, independent SBuses are combined in one SBus System, communication between an SBus Master on one SBus and an SBus Slave on another SBus is not defined by this standard.

The number of SBus Slaves and Masters is limited by total capacitive loading, by the functional requirements of the SBus System, and by mechanical considerations.

3.1.2 SBus Controller

There is one SBus Controller on each SBus of an SBus System. The SBus Controller performs all the centralized management functions required by the SBus protocol. The Controller provides all the bus synchronization clocking, manages arbitration among SBus Masters for use of the SBus, and provides the virtual address to physical address translation service for SBus Masters. The translation service includes the determination of which Slave it is to be selected.

The SBus Controller counts the number of data transfers performed during an SBus cycle to determine when to end the Transfer Phase. If a failure causes no data transfers to occur, the Controller provides a time-out service for the SBus Master.

The SBus Controller and its associated circuitry provide the pull-up and pull-down circuits required for proper operation of the SBus.

The virtual address to physical address translation makes use of a memory management unit (MMU). The SBus Controller may have a private MMU or share its MMU with other system components, including the CPU or the SBus Controller for another SBus. This standard does not specify the MMU or system memory caching implementations. This standard does not specify the control and sense registers used by the host CPU to manage the SBus Controller and the MMU.

3.1.3 SBus Slave

SBus Slaves monitor the state of the SBus to determine if requests are being made from some SBus Master. The SBus Slave selected by the SBus Controller performs the required data access operation by passing the requested data to or from the Master during the Transfer Phase. The SBus Controller provides the necessary selection and control signals to manage the Slave.

SBus Slaves provide appropriate acknowledgments to the SBus Master to indicate the successful or unsuccessful termination of the Transfer Phase, to indicate when the bytes of data are transferred by the SBus Slave, and to indicate the width of the data transfer being performed. Slaves perform a single transfer or a burst transfer according to the request from the SBus Master and consistent with the Slave's capabilities. An SBus Slave requests Bus Sizing during a single transfer if it cannot provide the width of data requested by the Master. The Master then performs additional SBus cycles to obtain the data not transferred in the first SBus cycle. SBus Masters execute retries of a data transfer if the SBus Slave indicates that it is temporarily unable to transfer the required data.

SBus Slaves use the physical address generated by the SBus Controller during the Translation Phase to determine which data is to be transferred. The SBus Controller selects the correct SBus Slave based on the virtual address provided by the SBus Master using algorithms not specified by this standard.

An SBus Slave may be integrated with an SBus Controller. In that case, the SBus Controller and SBus Slave may generate the selection and physical address signals internally, without presenting their information on the SBus itself. Presentation of physical address information on the SBus, while not required, provides useful information to SBus analysis tools.

Figure 2 outlines SBus functional relationships.

3.2 Overview of configurations

SBus Devices (SBus Masters, SBus Slaves, and the SBus Controller) may be installed in an SBus System in a wide variety of configurations. The following configurations are typical of those that have already been implemented. In general, the SBus Controller is implemented as part of the circuitry of the motherboard of an SBus System. One or more SBus Devices may also be implemented as part of the motherboard. SBus Slots may also be included on the motherboard, in which case, SBus Masters and SBus Slaves implemented on SBus Cards may be installed in those slots in any combination.

3.2.1 Symmetric SBus cycles

A symmetric SBus cycle is performed between an independent SBus Master and SBus Slave using the services of an independent SBus Controller. A symmetric SBus cycle requires the execution of an Arbitration Phase between the SBus Master and Controller to select the Master that will have control of the SBus. The Controller then performs a Translation Phase to select the proper SBus Slave and to provide the physical address information to the Slave. Data is then transferred between the SBus Master and Slave using a Transfer Phase. Transfers that are 64-bits wide require an additional Extended Transfer Information Phase between the SBus Master and Slave.

Symmetric SBus cycles are performed by Masters that are not combined with Controllers. Masters located on SBus Cards perform symmetric SBus cycles to any Slave. The Master under control of the host CPU performs symmetric SBus cycles if it is not combined with the SBus Controller.

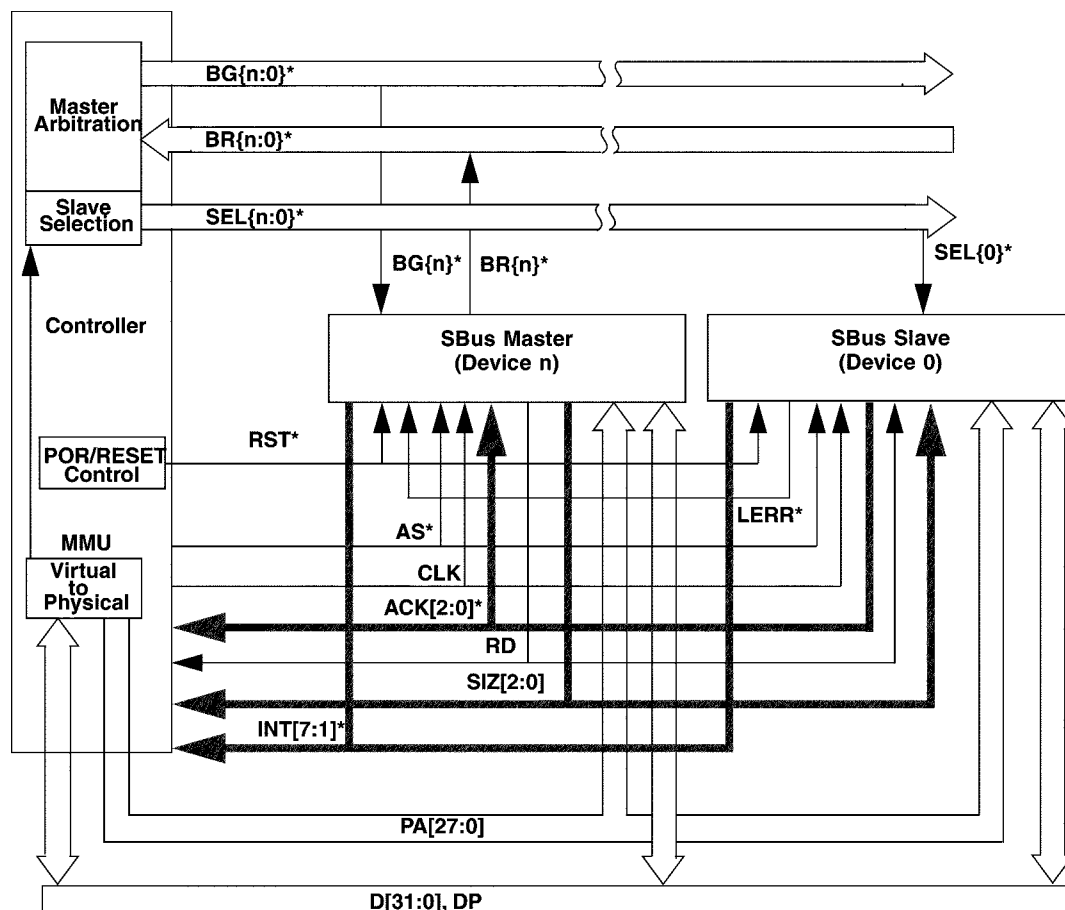


Figure 2 – SBus functional block diagram

SBus Systems shall support symmetric SBus cycles between a Master installed in any SBus Slot and any SBus Slave.

The SBus standard does not specify the method of connecting the host CPU to the host's SBus Master or to the SBus Controller.

Figure 3 shows an example of an SBus System performing a symmetric SBus cycle.

3.2.2 Asymmetric SBus cycles

An SBus cycle may be performed between an SBus Master and an SBus Slave without performing an Arbitration Phase and a Translation Phase on the SBus if the SBus Master and the SBus Controller share the necessary functions to perform arbitration and translation privately. The SBus Master communicates privately with the SBus Controller to gain control of the SBus. The SBus Controller then uses the information provided to it privately by the SBus Master to select the proper SBus Slave, provide the physical address information to the Slave, and immediately begin the Transfer Phase. Transfers that are 64-bits wide require an additional Extended Transfer Information Phase between the SBus Master and Slave.

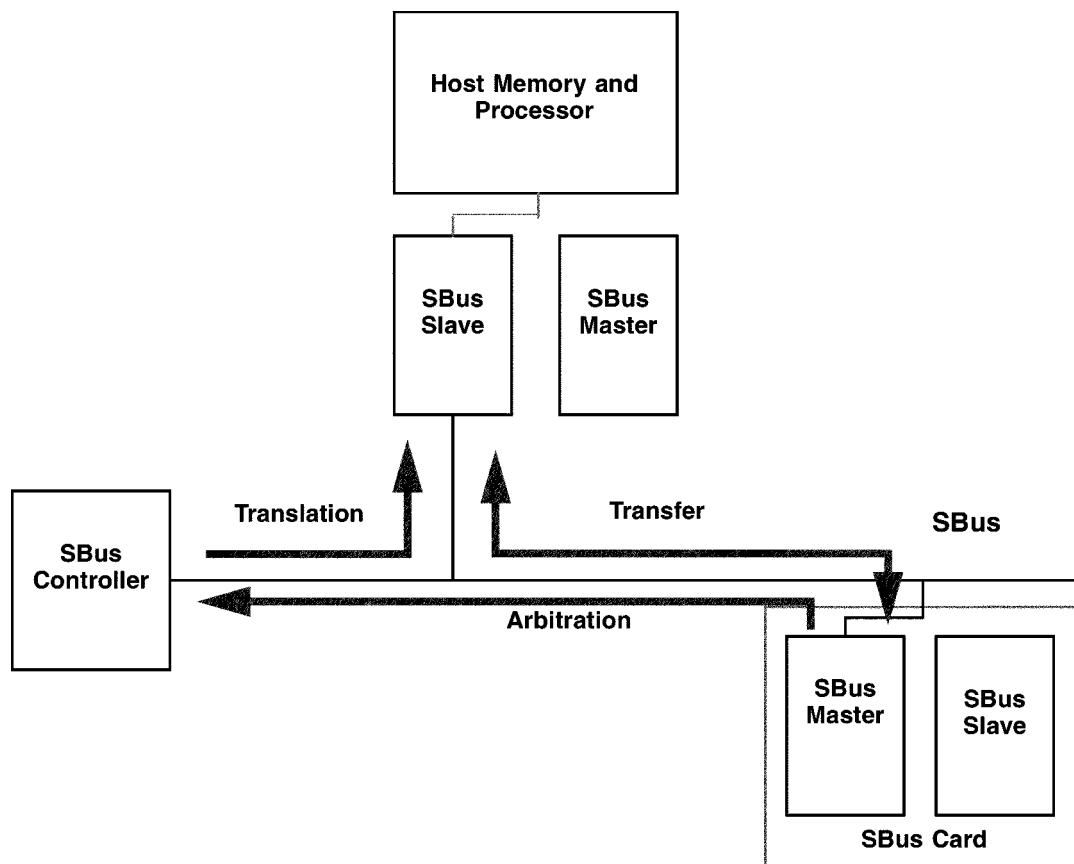


Figure 3 – Example of symmetric SBus cycle

Asymmetric SBus cycles shall only be performed by SBus Masters that are combined with the SBus Controller. While an SBus Master combined with the Controller is allowed to perform symmetrical SBus cycles, the lower overhead associated with asymmetric SBus cycles provides a performance advantage. While any SBus Master could be combined with the SBus Controller, the Masters associated with the host CPU are typically the only Masters that are capable of performing asymmetric SBus cycles.

The SBus standard does not specify the method of combining the SBus Controller and the SBus Masters that perform asymmetric SBus cycles.

Figure 4 shows an example of an SBus System performing an asymmetric SBus cycle.

3.2.3 Multiple SBus Systems

SBus Systems may have more than one SBus. The logical signals of each SBus shall be independent and each separate SBus shall have its own SBus Controller circuitry. This architecture allows systems to be built in which the aggregate system I/O bandwidth is higher than that provided by a single SBus.

Such configurations may place limitations on communication between SBus Devices on different SBuses. The standard only defines the behavior of those SBus Devices sharing the same SBus.

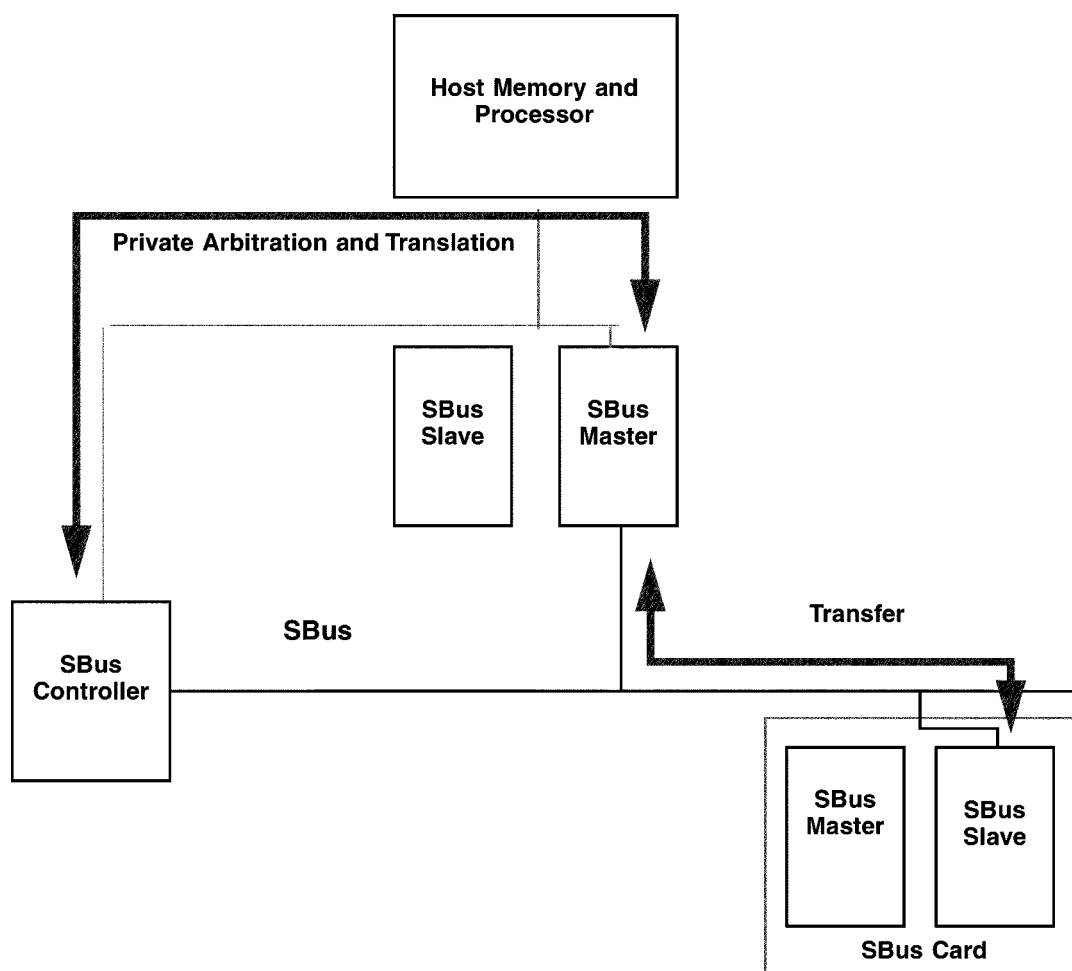


Figure 4 – Example of asymmetric SBus cycle

3.3 General design information

3.3.1 Protocol control

The SBus Controller generates a fixed-frequency clock signal (CLK) in the range of 16 MHz to 25 MHz as specified in 6.3. The logical state of SBus protocol control signals is determined at the time of the rising edge of CLK. Signals shall be driven so that they meet the SBus setup time and hold time requirements with respect to the rising edge of CLK at the SBus Device. SBus interrupts are not required to be synchronous with respect to CLK.

The SBus protocol indicates that the state of one or more of the information signals, SEL*, RD, SIZ[2:0], PA[27:0], D[31:0], D[63:32], and DP is required to be valid on a specific clock edge by using one of the protocol control signals, AS* or ACK[2:0]*. The information signals shall meet the specified setup time and hold time with respect to the rising edge of the CLK signal when the signals are required to be valid by the appropriate protocol control signal. When the protocol control signals reflect such a state that the information signals are not of interest, there are no requirements on their state or setup and hold times.

A transition that occurs in preparation for an edge of a clock is said to occur in the clock cycle before the edge. As an example, the sample signal below transitions from low to high in the clock cycle after clock edge 2 samples the low value and in the clock cycle before clock edge 3 samples the high value. This convention will be followed through the remainder of the document and applies to signal transitions and phase transitions.

Figure 5 shows an example of synchronous operation timing.

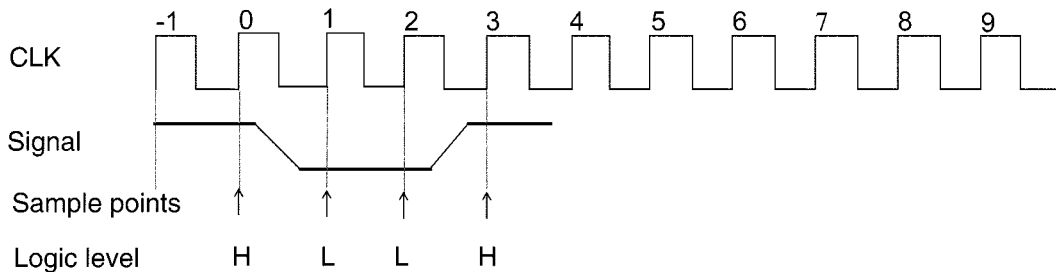


Figure 5 – Example of synchronous operation

3.3.2 Bus driving

Certain SBus control signals, when they have been asserted, shall be actively driven to the negated state before the drive is removed. The use of active negation before the drive is removed allows the operation of the bus at speeds up to the maximum specified clock rate without the need for low resistance pull-up resistors and high-current output drivers.

Figure 6 shows an example of active negation.

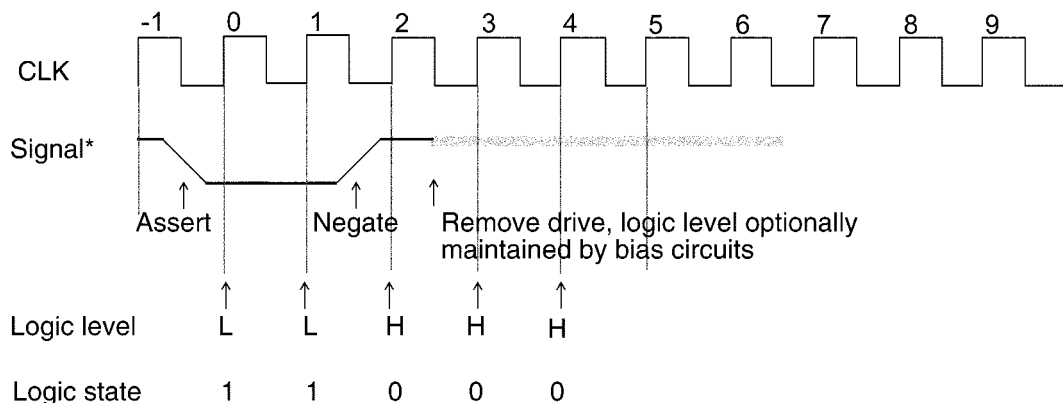


Figure 6 – Example of active negation on low-active control signal

No signal, except open-drain interrupts, shall be driven by two outputs during the same clock cycle. This guarantees that SBus output drivers never drive conflicting levels, which can result in inconsistent operation, excessive power dissipation, and early circuit failure.

Figure 7 shows an SBus Device driving a high logic level for cycle 0, then removing its drive. In this case, the grey line indicates that the state of the signal at cycle 1 is unknown or meaningless. At cycle 2, a second SBus Device has driven a low logic level.

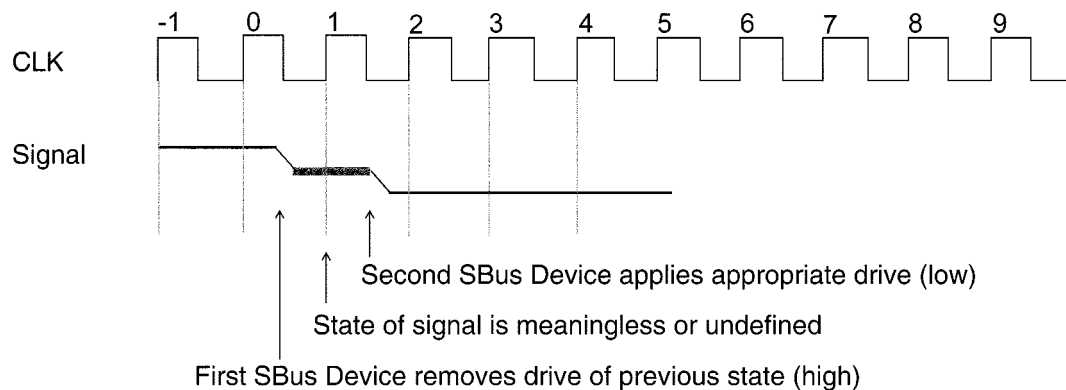


Figure 7 – Example of driver overlap protection

Bus or other signals operating as a unit and having a state that is encoded in the high and low states of several lines will be symbolized in the timing diagrams as in Figure 8. The values of such lines will be specified by appropriate notations where they are not obvious from normal operation of the SBus.

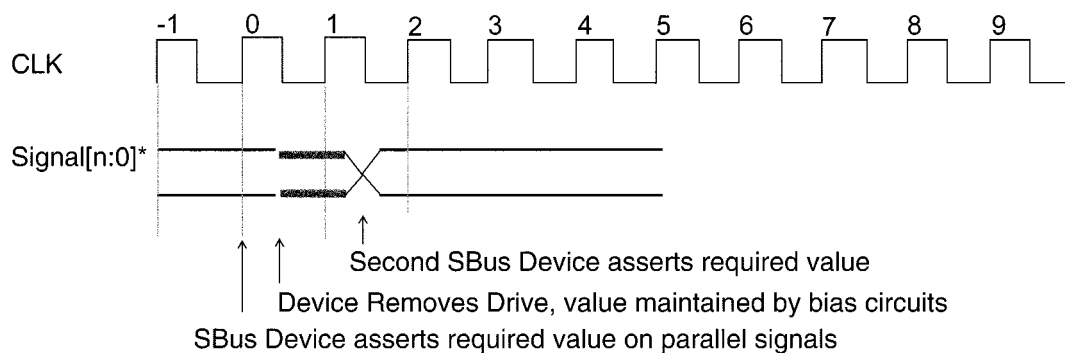


Figure 8 – Symbols for multiple signals operating as a unit

3.4 Performance

The SBus has a high performance capability, but that capability cannot be realized if SBus Devices with excessive overhead are included in the SBus System. Both the data access latency and the protocol execution time of SBus Devices contribute to overhead. This standard does not specify the performance characteristics required of an SBus Device, nor does it predict the resulting throughput or latency of the resulting SBus System.

3.4.1 Clock rates

SBus performance is maximized by operating the SBus clock at the maximum specified clock frequency. SBus Cards shall be capable of operating at the maximum SBus clock frequency to assure that they can be installed in all SBus Systems and will provide maximum performance.

3.4.2 Transfer rates

SBus transfer rate is maximized when 32-bit Burst Transfers or 64-bit Extended Transfer bursts are used to transfer data.

For asymmetric SBus cycles, as few as two clock cycles of overhead are possible, providing a Burst Transfer rate of 64 bytes every 18 clock cycles or 88 Mbytes/s. For 64-bit Extended Transfers, with three clock cycles of overhead, the corresponding Burst Transfer rate is 168 Mbytes/s.

Symmetric SBus cycles have at least two additional clock cycles of overhead for the translation cycle, resulting in a minimum of 20 clock cycles to transfer 64 bytes, or 80 Mbytes/s at a clock rate of 25 MHz. For 64-bit Extended Transfers, the corresponding Burst Transfer rate is 160 Mbytes/s.

SBus Systems will not meet these maximum rates if additional cycles are used in address translation, in waiting for data access to SBus Slaves, or in sequencing the SBus protocol. For best system performance, SBus Devices should operate with as few such wasted cycles as possible.

3.4.3 Latency

The SBus architecture does not guarantee a maximum worst-case latency if Retry Acknowledgments are used. Most modern systems with bus bridge architectures do perform Retry Acknowledgments under some conditions. SBus Masters and SBus Slaves should be designed to operate correctly even when responses are slow. Slow responses should not cause data overrun or underrun conditions. Buffering and appropriate flow control algorithms should be implemented as necessary to prevent failures caused by large response latencies. SBus Masters and Slaves that depend on constrained latency may not operate in all system environments.

Latency is influenced by the following factors:

- the number of SBus Masters;
- the request stream from each SBus Master;
- the resources available to each SBus Slave to perform the desired requests;
- the presence of bus bridges;
- the overhead associated with the execution of an SBus cycle;
- the occurrence of retries.

Each individual SBus System must be analyzed with respect to these parameters to determine the worst-case, typical, and average latencies.

4 Signal definitions

Table 1 shows all defined SBus signals.

Table 1 – SBus signals

Signal name	Description	32-bits operation driven/received by			64-bits Extended Transfer driven/received by		
		Controller	Master	Slave	Controller	Master	Slave
PA[27:0]	Physical Address	D		R	D		R
D[59:32]	Extended Transfer Data					D/R	R/D
SEL*	Select (1/Slave)	D		R	D		R
D[31:0]	Data	R	D/R	R/D	R	D	R
D[31:0]	Extended Transfer Data					D/R	R/D
SIZ[2:0]	Transfer Size	R	D	R	R	D	R
D[62:60]	Extended Transfer Data					D/R	R/D
RD	Transfer Direction	R	D	R	R	D	R
D[63]	Extended Transfer Data					D/R	R/D
CLK	Clock	D	R	R	D	R	R
AS*	Address Strobe	D		R	D	R	R
ACK[2:0]*	Acknowledgment	D/R	R	D	D/R	R	D
LEER*	Late Error		R	D		R	D
BR*	Bus Request (1/Master)	R	D		R	D	
BG*	Bus Grant (1/Master)	D	R		D	R	
RST*	Reset	D	R	R	D	R	R
INT[7:1]*	Interrupt Request	R		D	R		D
DP	Data Parity	R	D/R	R/D	R	D/R	R/D

The relationship among the signals is diagrammed in Figure 2, SBus functional block diagram.

4.1 CLK signal

The SBus Controller generates a fixed-frequency square-wave clock signal, CLK, which is used by all SBus Devices as a reference signal that validates most other SBus signal states. CLK shall be a fixed frequency in the range specified by 6.3. Every SBus Card shall operate over this entire range of clock frequencies.

The physical length, configuration, capacitance, and drive characteristics of the SBus shall be such that the rising transition of CLK at each SBus Device occurs within the specified clock skew time of the time the signal appears at each other SBus Device. The rise and fall times of CLK shall not exceed the specified value at any SBus Slot. Since the specifications for rise time and clock skew are very demanding, CLK to each SBus Device should be driven by a separate clock driver. The CLK distribution lines should be balanced in length and capacitive load. For more detailed timing information, see 6.3.2.

SBus Devices shall sample SBus signals on the rising edge of CLK, with three exceptions. Those transitions which are not synchronized with the rising edge of CLK are the assertion of RST*, the assertion of INT[7:1]*, and the negation of INT[7:1]*.

CLK is not guaranteed to be a symmetrical square wave.

CLK may vary in frequency from system to system. SBus Cards should not use CLK as a known frequency source. If a known frequency source is required for operation of an SBus Card, it should be generated from some other clock source.

Some SBus Devices are able to change the number of clock cycles required to perform internal functions, such as memory reads and writes, as a function of the SBus CLK frequency. The approximate frequency of CLK is determined by examining a special FCode attribute using the SBus Device's FCode program. The program can then initialize the SBus Device for optimum use of CLK. The FCode attributes are described by IEEE Std 1275. Chapter 9 provides a brief summary of the characteristics of FCode.

4.2 RST* signal

The reset signal, RST*, is a low-asserted signal that is driven by the SBus Controller to properly initialize all SBus Devices as required by the SBus System.

The SBus Controller shall assert RST* after power is applied to the SBus Devices. SBus Controllers should assert RST* as soon as possible after power is applied to the SBus System to prevent the execution of SBus transfers involving SBus Devices that have not executed their self-initialization process. RST* may be asserted at other times by software or hardware controls as required by the SBus System implementation. SBus Controllers should be implemented with a software managed reset capability.

RST* shall be asserted for a minimum count of 512 cycles of CLK before being negated. In the case of SBus System power-up, power to the SBus Devices shall be within the specified range of values before the SBus Controller begins counting the 512 clock cycles. The assertion of RST* is not required to be synchronized to CLK. The negation of RST* shall meet the required setup and hold times with respect to CLK.

All SBus signals except RST* and CLK driven by the SBus Controller shall immediately enter their reset state when RST* is asserted, even though CLK might not be present or valid when RST* is first asserted. The reset state for signals driven by a single source (BR*, BG*, AS*, SEL*, and PA[27:0] for 32-bit transfer SBus Devices) shall be negated. The reset state for signals potentially driven by more than one source (D[31:0], DP, RD, ACK[2:0]*, SIZ[2:0], LERR*, INT[7:1]*, and PA[27:0] for 64-bit Extended Transfer SBus Devices) shall be released.

RST* may be driven as a single signal to all SBus Devices or it may be driven separately to smaller groups of the attached SBus Devices. RST* shall not be used to individually reset SBus Devices. The assertion of RST* shall appear at all SBus Devices within two clock cycles of the time the assertion first appears at any SBus Device. RST* shall be negated at all SBus Devices on the same clock cycle.

After RST* has been asserted to an SBus Device, it shall perform whatever operations are required to complete its internal initialization. The internal reset processes may begin at any time during the assertion of RST* or after the negation of RST*.

An SBus Card shall complete its internal initialization process within 100 ms of the negation of RST* and be prepared for normal SBus operations after that time. SBus Systems should be programmed to allow a minimum of 100 ms from the time that RST* is negated until the first access of any kind to any SBus Card. This allows each SBus Card to complete any internal initialization that may be required. SBus Systems that do not follow this recommendation may not operate with those SBus Cards requiring significant time for internal initialization. Systems that have SBus Devices attached directly to the motherboard need not delay accesses to those Devices, since the SBus System has knowledge about the precise initialization requirements of those Devices.

Because RST* is the signal that invokes the initialization process and because it may be generated as a result of a software or hardware operation other than power on, an SBus Device shall not rely only on the detection of a power-on process to perform initialization. Figure 9 shows typical RST* timing.

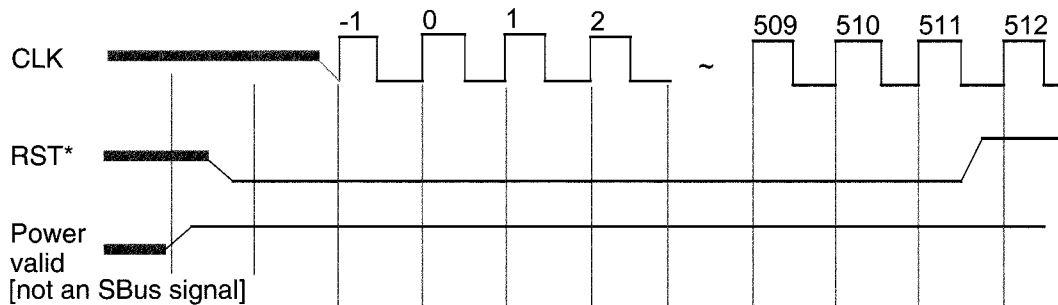


Figure 9 – Example of RST* timing

4.3 PA[27:0] signals

The SBus Controller shall drive 28 high active signals, PA[27:0], to provide the physical address information to the selected SBus Slave. The most significant bit of the physical address information is PA[27].

During the Transfer Phase of 64-bit Extended Transfers, the SBus Controller shall release PA[27:0] so that the SBus Master and the SBus Slave can use the signal paths for D[59:32].

The SBus Controller generates the PA[27:0] bits and selects an SBus Slave using the proper mapping from the virtual address presented during the Translation Phase. If the SBus Controller is integrated with an SBus Master, then the PA[27:0] bits may be developed using information internal to the combined SBus Devices. The algorithm for generating the PA[27:0] bits depends on the particular SBus System.

The PA[27:0] signals shall remain at the proper valid logic level during the entire required period of assertion. For Extended Transfers, the protocol requires that the PA[27:0] state information be retained by the SBus Slave during the Transfer Phase, but SBus Slaves operating in 32-bit mode may choose not to latch the value into registers. SBus Slaves performing 32-bit Burst Transfers shall internally generate the correct low-order address bits required by the transfer.

4.4 SEL* signal

The SBus Controller shall drive a separate low-asserted selection line, SEL*, to each SBus Device that has an SBus Slave capability and to each SBus Slot.

The SBus Controller selects the SBus Slave that will receive SEL* using the proper mapping from the virtual address presented during the Translation Phase. If the SBus Controller is integrated with an SBus Master, then the proper SEL* may be developed using information internal to the combined SBus Devices. The algorithm for generating SEL* depends on the particular SBus System.

SEL* shall remain at the proper valid logic level during the entire period required by the assertion of AS*. SEL* shall be at a valid one or zero state at least a setup time before the rising edge of CLK that will identify AS*. SEL* shall remain stable until the clock cycle after AS* is negated. SEL* is not defined outside the period established by AS*.

4.5 AS* signal

The SBus Controller shall drive a low-asserted address strobe signal, AS*, to the attached SBus Slave Devices and to the SBus Slots.

AS* is asserted to indicate that SEL* and the PA[27:0] signals are valid and that a Transfer Phase is in progress. PA[27:0] and SEL* shall remain stable until the clock cycle after AS* is negated. The logic state of SEL* and PA[27:0] is not guaranteed except where qualified by AS*.

AS* shall not be driven for SBus cycles that have been terminated during the Translation Phase by an Error or Retry Acknowledgment from the SBus Controller. AS* may or may not be driven by SBus Controllers to an SBus Slave that is integrated with the Controller if the transfer is a 32-bit transfer.

After an SBus Slave has been selected by the assertion of AS* and SEL*, the Slave must generate the appropriate acknowledgments to terminate the bus cycle within the specified timeout period of clock cycles (see 5.6.2). The SBus Controller shall assert AS* at least until the clock cycle after the clock edge that sampled the final acknowledgment for the transfer. The acknowledgment may be a Data Acknowledgment, a Retry Acknowledgment, an Error Acknowledgment, or a Controller-generated timeout Error Acknowledgment. After the acknowledgment, the SBus Controller shall negate AS* for at least one clock cycle. An SBus Slave shall check for AS* being negated to delineate successive bus cycles. An SBus Controller should negate AS* during the clock cycle following the final acknowledgment to achieve maximum performance.

4.6 BR* signal

The bus request signal, BR*, is a low-asserted signal driven by an SBus Master. Each SBus Master provides a separate BR* to the SBus Controller. Since an SBus Master can be plugged into any SBus Slot, each SBus Slot shall have a separate BR*. The SBus Controller arbitrates among those SBus Masters requesting control of the SBus and grants control to only one of those SBus Masters by asserting BG* to that SBus Master.

After asserting its BR*, the SBus Master shall leave the signal asserted until it has been granted bus mastery by the SBus Controller. Bus mastery is granted to the individual SBus Master when the SBus Controller asserts BG* to the SBus Master. During the clock cycle immediately following the assertion of BG*, the Master must negate BR* for at least one clock cycle.

4.7 BG* signal

The bus grant signal, BG*, is a low-asserted signal. The SBus Controller drives a separate BG* to each SBus Master. Since an SBus Master can be plugged into any SBus Slot, each SBus Slot shall have a separate BG*. During the Arbitration Phase, the SBus Controller chooses the SBus Master that will be granted control of the SBus from among those that have asserted a BR* to the SBus Controller. The SBus Controller notifies the SBus Master that it has been granted control by asserting a BG* to that SBus Master. Only one BG* shall be active at any time. The BG* shall be held asserted by the SBus Controller at least until the clock cycle following the rising edge of CLK that sampled the last Acknowledgment expected from the SBus Slave or the Acknowledgment generated by the SBus Controller.

4.8 D[31:0], D[63:0], and DP signals

The 32 high active data bus signals, D[31:0], and the optional data parity signal, DP, are used by the SBus Master to transmit virtual address information to the SBus Controller and are used by the SBus Master and SBus Slave to transfer data from one to the other. D[31] is the most significant bit and D[0] is the least significant bit on the data bus. DP optionally maintains odd parity over bits D[31:0] or D[63:0] as described in 5.4.7.

Four data transfer widths are defined:

- byte: 8 bits of data;
- half-word: 16 bits of data;
- word: 32 bits of data;
- double-word: 64 bits of data (optional data size).

SBus also supports multiple transfers of word or double-word data in a single SBus Transfer Phase. These transfers are called Burst Transfers.

The data signals shall be valid at the receiving SBus Device for a setup time before and for a hold time after the rising edge of the CLK that is indicated by the ACK[2:0]* signals as the edge that is to be used for sampling the data signals. The data signals are not required to be valid outside that time period.

The optional 64-bit transfer uses D[31:0] for the low-order 32 bits of data to be transferred. The high-order bits are transferred using SBus signals that are otherwise unused during the Extended Transfer Phase. Table 2 shows 64-bit Extended Transfer data-bit assignments.

Table 2 – Data-bit assignments for 64-bit Extended Transfers

Signal name	Data-bit assignment
RD	D[63]
SIZ[2:0]	D[62:60]
PA[27:0]	D[59:32]
D[31:0]	D[31:0]

Figure 10 defines the bit, byte, half-word, word, and double-word names. The address presented on the PA[27:0] signals specifies the most significant byte of the addressed data entity.

Bit (Word)	31	24	23	16	15	08	07	00	31	24	23	16	15	08	07	00		
Bit (64-bit)	63	56	55	48	47	40	39	32	31	24	23	16	15	08	07	00		
	Byte 0		Byte 1		Byte 2		Byte 3		Byte 0		Byte 1		Byte 2		Byte 3			
	Half-word 0				Half-word 1				Half-word 0				Half-word 1					
	Word 0										Word 1							
							Double-word											
PA[2:0]	000		001		010		011		100		101		110		111			

Figure 10 – Double-words, words, half-words, and bytes

4.9 SIZ[2:0] signals

The size signals, SIZ[2:0], are high active signals that shall be driven by the SBus Master that has been granted bus mastery by the SBus Controller. SBus Masters transmit a code describing the amount of data to be transferred during the Transfer Phase of the SBus cycle.

During the Transfer Phase of 64-bit Extended Transfers, the SBus Master shall release SIZ[2:0] so that the SBus Master and the SBus Slave can use the signal paths for D[62:60].

For each SBus cycle, the SBus Master having bus mastery determines how much data it wishes to transfer. The relationship between the SIZ[2:0] signals and the allowable data transfers is explained in 5.4.3. The SBus Master shall drive SIZ[2:0] to its proper state during the clock cycle following the assertion of BG*. For 32-bit transfers, SIZ[2:0] remains asserted during the entire Translation and Transfer Phases. For 64-bit Extended Transfers, the SBus Master releases SIZ[2:0] after the Extended Transfer Information Phase.

The requirements for SBus Masters, SBus Slaves, and SBus Controllers with respect to the selection of SIZ[2:0] values to be implemented and the corresponding ACK[2:0]* values to be generated is explained in 5.4.3.

The SIZ[2:0] signals shall remain at the proper valid logic level during the entire required period of assertion. For Extended Transfers, the protocol requires that the SIZ[2:0] state information be retained by the SBus Slave. SBus Slaves that are performing 32-bit transfers may use the value on the SIZ[2:0] signals instead of latching the SIZ[2:0] value into registers.

Table 3 indicates the transfer size associated with each value of SIZ[2:0].

4.10 RD signal

The read signal, RD, is a high active signal that shall be driven by the SBus Master that has been granted bus mastery by the SBus Controller. The SBus Master shall use RD to indicate whether it desires to read data from the SBus Slave (RD set to one) or write data to the SBus Slave (RD set to zero).

During the Transfer Phase of 64-bit Extended Transfers, the SBus Master shall release RD so that the SBus Master and the SBus Slave can use the signal path for D[63].

Table 3 – SIZ[2:0]

Function	Bytes transferred	SIZ[2]	SIZ[1]	SIZ[0]
Word transfer	4	0(L)	0(L)	0(L)
Byte transfer	1	0(L)	0(L)	1(H)
Half-word transfer	2	0(L)	1(H)	0(L)
Extended Transfer (see 5.3.2)		0(L)	1(H)	1(H)
4-word burst	16	1(H)	0(L)	0(L)
8-word burst	32	1(H)	0(L)	1(H)
16-word burst	64	1(H)	1(H)	0(L)
2-word burst	8	1(H)	1(H)	1(H)

RD shall be driven to the proper state by the SBus Master that has been granted bus mastery beginning with the clock cycle following the assertion of BG* by the Controller.

For SBus Controllers having the SBus Master integrated in such a manner that no arbitration cycle takes place, RD shall be stable from the clock cycle in which AS* is asserted by the SBus Controller. It shall remain stable until the clock cycle following the negation of AS*.

RD shall remain at the proper valid logic level during the entire required period of assertion. For Extended Transfers, the protocol requires that the RD state information be retained by the SBus Slave. For 32-bit transfers, RD shall remain stable until the clock cycle following BG* being negated. SBus Slaves that only perform 32-bit transfers may choose not to latch the logic state of RD into registers.

SBus Controllers are allowed to use RD to assist in protecting segments of the virtual address space from invalid accesses.

4.11 ACK[2:0]* signals

The acknowledgment signals, ACK[2:0]*, are low-asserted signals that shall be driven by the selected SBus Slave or by the SBus Controller. When driven by the SBus Controller, ACK[2:0]* is used to indicate to the current SBus Master that an operation has failed or must be retried. When driven by the SBus Slave, ACK[2:0]* either qualifies the transfer of data to or from an SBus Master or indicates to the SBus Master that an operation has failed or must be retried.

The following protocol attempts to prevent the SBus Slave and the SBus Controller from presenting conflicting states on ACK[2:0]* in the presence of most error conditions. The SBus Controller may drive ACK[2:0]* before AS* is asserted to indicate that the Controller has detected an error or is requesting a retry during the Translation Phase.

If the SBus Controller does drive ACK[2:0]* during the Translation Phase, it shall not assert SEL* or AS* and shall negate BG*, ending the SBus cycle. If the SBus Controller did not end the SBus cycle by driving ACK[2:0]* during the Translation Phase, the SBus Slave is then allowed to drive ACK[2:0]* until the specified timeout number of clock cycles of the Transfer Phase to provide the proper data transfer, error, or retry indications. If the SBus Slave did not drive all required acknowledgments on ACK[2:0]* within the timeout period of the Transfer Phase, then the SBus Controller shall drive an Error Acknowledgment onto the ACK[2:0]* signals to indicate a timeout error. At times other than those allowed by the protocol, the SBus Slave and Controller shall release ACK[2:0]*.

SBus Transfer Phases are sequenced or terminated by the presentation of one or more acknowledgment code values on the ACK[2:0]* signals. Four major classes of acknowledgments are defined.

Data Acknowledgments indicate the successful transfer of the specified unit of data between an SBus Master and an SBus Slave. The particular Data Acknowledgment selected by the SBus Slave indicates the width of the data elements being transferred. The data width implemented by an SBus Slave may vary from one portion of its address space to another. The actual data width of the Slave may be masked from the SBus Master by SBus bridges.

An Error Acknowledgment indicates that the requested transfer was unsuccessful and ends the Transfer Phase. See 5.4.9.

A Retry Acknowledgment indicates that the selected SBus Slave was temporarily unable to perform the requested transfer. The Retry Acknowledgment ends the Transfer Phase. The SBus Master shall retry the SBus cycle. See 5.4.8.

An Idle Acknowledgment is a special code on ACK[2:0]* that indicates that no acknowledgment information is being presented. The reserved Code is defined for future standardization.

Table 4 shows the encoded values for each type of acknowledgment.

Table 4 – ACK[2:0]* encoded values

Function	ACK[2]*	ACK[1]*	ACK[0]*
Idle	0(H)	0(H)	0(H)
Error Acknowledgment	0(H)	0(H)	1(L)
Byte Acknowledgment	0(H)	1(L)	0(H)
Retry Acknowledgment	0(H)	1(L)	1(L)
Word Acknowledgment	1(L)	0(H)	0(H)
Double-word Acknowledgment	1(L)	0(H)	1(L)
Half-word Acknowledgment	1(L)	1(L)	0(H)
Reserved	1(L)	1(L)	1(L)
NOTE ACK[2:0]* are active low signals. See 2.1.			

SBus Slaves present an acknowledgment code by driving ACK[2:0]* to a value other than the Idle value for one setup time before the rising edge of CLK and holding the value at least one hold time after the same rising edge. The acknowledgment code is interpreted by the SBus Master and Controller at the time of the rising edge of CLK to determine when the requested data transfer is being executed. Consecutive or separated presentations of the Word Acknowledgment or the Double-word Acknowledgment may be presented, one for each of the several data transfers associated with Burst Transfers. If the data transfers are not performed in successive cycles, ACK[2:0]* shall be driven to the Idle value for the proper setup and hold times for at least the first rising edge of CLK not associated with a data transfer. For subsequent rising edges of CLK not associated with a data transfer, the SBus Slave may drive the Idle value or depend on the bias circuits to maintain the Idle value.

After the last acknowledgment of the SBus cycle has been presented, the SBus Slave shall drive ACK[2:0]* to the Idle value for one clock cycle, after which the Slave shall release the ACK[2:0]* signals. Before driving ACK[2:0]*, the SBus Slave may depend on the bias circuits of the ACK[2:0]* signals to maintain the released ACK[2:0]* signals at the Idle value. After an Idle Acknowledgment state has been established by driving the ACK[2:0]* to the Idle value for one clock cycle, the SBus Slave may again depend on the bias circuits to continue maintaining the ACK[2:0]* signals at the Idle value.

Instead of continuously driving ACK[2:0]* to the Idle value, the SBus Slave should rely on the terminations to maintain that value. The SBus Controller can then assert the Error Acknowledgment value required to indicate a timeout without generating a conflict with the SBus Slave drivers and without requiring a special timer to release the Slave drivers.

The requirements for SBus Masters, SBus Slaves, and SBus Controllers with respect to the selection of SIZ[2:0] values to be implemented and the corresponding ACK[2:0]* values to be generated is explained in 5.4.3.

SBus Masters shall complete their SBus sequencing properly upon receiving any encoding of ACK[2:0]*. SBus Masters shall support the Error Acknowledgment code value. No specific interpretation of Error Acknowledgment is defined by this standard other than the requirement to terminate the Transfer Phase. An SBus Master's additional responses to the assertion of Error Acknowledgment are defined by the implementation of the Master. SBus Masters shall support Retry Acknowledgment by retrying the SBus cycle.

The Bus Controller shall monitor SIZ[2:0], and ACK[2:0]* so that it is able to negate AS* and BG* after the SBus Slave has issued the last acknowledgment for that SBus cycle. The SBus Controller shall count Data Acknowledgments to be sure that the requested transfer has been completed. The SBus Controller shall terminate the SBus cycle if the SBus Slave presents an Error Acknowledgment or a Retry Acknowledgment. The SBus Controller shall present Error Acknowledgment if the SBus Slave does not respond with all required acknowledgments within the timeout period. See 5.6.2.

4.12 LERR* signal

The late error signal, LERR*, is a low-asserted signal that is driven by an SBus Slave to indicate to the SBus Master and the SBus Controller that an error was detected. LERR* does not end the Transfer Phase. The SBus Master shall complete the entire transfer. LERR* is typically used either to indicate that an error was detected so late that it could not be presented using Error Acknowledgment or that the error was not related to the proper sequencing of the transfer. Examples include parity errors and data-storing errors.

If an SBus Slave detects an error that requires it to assert LERR*, the signal shall be asserted with the required setup and hold times at the rising edge of CLK two clock cycles after the corresponding Data Acknowledgment was presented. LERR* shall not be asserted instead of an Error or Retry Acknowledgment required by the standard. LERR* may be asserted after an Error Acknowledgment related to a function not reported by LERR*. LERR* shall be driven to its negated state on the cycle after its presentation unless a second error related to a second Data Acknowledgment has been detected. The SBus Slave may depend on the termination on LERR* to maintain the signal in its negated state at other times. The SBus Slave shall not actively negate LERR* for the first two cycles after AS* is first asserted, since this may conflict with the assertion of LERR* by a previously selected Slave.

LERR* may be asserted at a time when the SBus Master may have started a new bus cycle or is no longer owner of the bus. LERR* may be asserted by an SBus Slave that is no longer receiving an SEL*. SBus Devices shall associate LERR* with the acknowledgment two cycles before LERR*, even if such an acknowledgment occurred between an SBus Master and SBus Slave that are no longer receiving a valid BG* and SEL* from the SBus Controller.

No specific interpretation of LERR* is defined by this standard. The SBus Master's and the SBus Controller's responses to the assertion of LERR* by the selected SBus Slave are defined by the implementation of the particular SBus Devices.

Figure 11 shows an example of LERR* timing.