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

ISBN 2-8318-5165-3

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 X

For price, see current catalogue

CONTENTS

Page

Clause Page

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 Varembé, 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
Shinkyo 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
Edward W. Aichinger
Ray S. Alderman
Richard P. Ames
Keith D. Anthony
Behrooz Bandall
Chris Bezirtzoglou
Michael L. Bradley
Scott M. Buck
Steven Cobb
Robert Crowde
Doug Degroot
Dante Del Corso
Samuel Duncan
Wilhelm P. Evertz
Wayne Fischer
Chee K. Fong
Joseph D. George
Larry E. Gerald
Robert R. Gianni
Steve Golson
Julio Gonzalez-Sanz
John Griffith
Hans H. Heilborn
Zoltan R. Hunor
Edgar Jacques
David V. James
Horace Jones
W. Frederick Ki
Ernst H. Kristiansen
Lak Ming Lam
Gerry Laws
Boon Lum Lim
Marlyn Miner
William E. Molyneaux
Richard Mueller
Michael Orlovsky
Mira Pauker
Philip K. Piele
Rochit Rajsuman
Brian Ramelson
Gary Sloane
Bob Snively
Michael Teener
Joseph P. Trainor
Robert Tripi
Rudolf Usselmann
Clarence M. Weaver
Michael Wenzel
Andrew Wilson
Robert J. Wood
David L. Wright
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
Clyde R. Camp
Donald C. Fleckenstein
Jay Forster*
David F. Franklin
Ramiro Garcia
Donald N. Heirman
Jim Isaak
Ben C. Johnson
Walter J. Karplus
Lorraine C. Kevra
E. G. Al Kiener
Ivor N. Knight
Joseph L. Koepfinger*
D. N. Jim Logothetis
T. Don Michael*
Marco Migliaro
L. John Rankine
Arthur K. Reilly
Ronald H. Reimer
Gary S. Robinson
Leonard L. Tripp
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.

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 Practices[2)]

---

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.

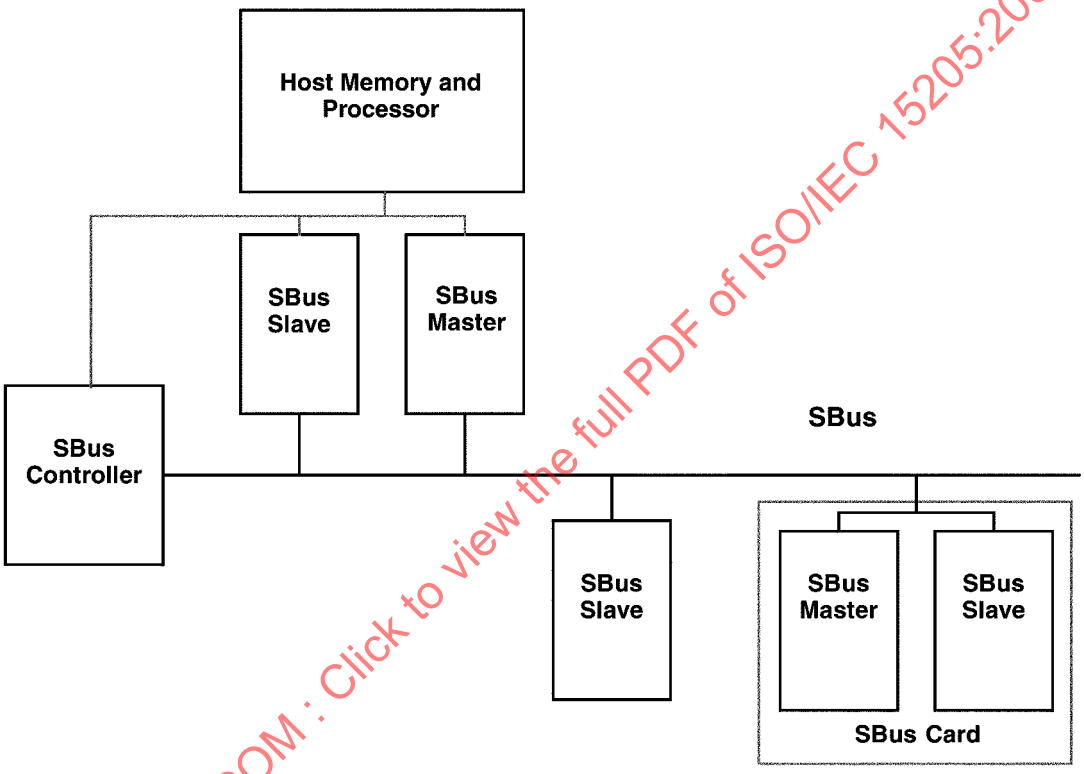


Figure 1 – Block diagram of representative SBus System

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.

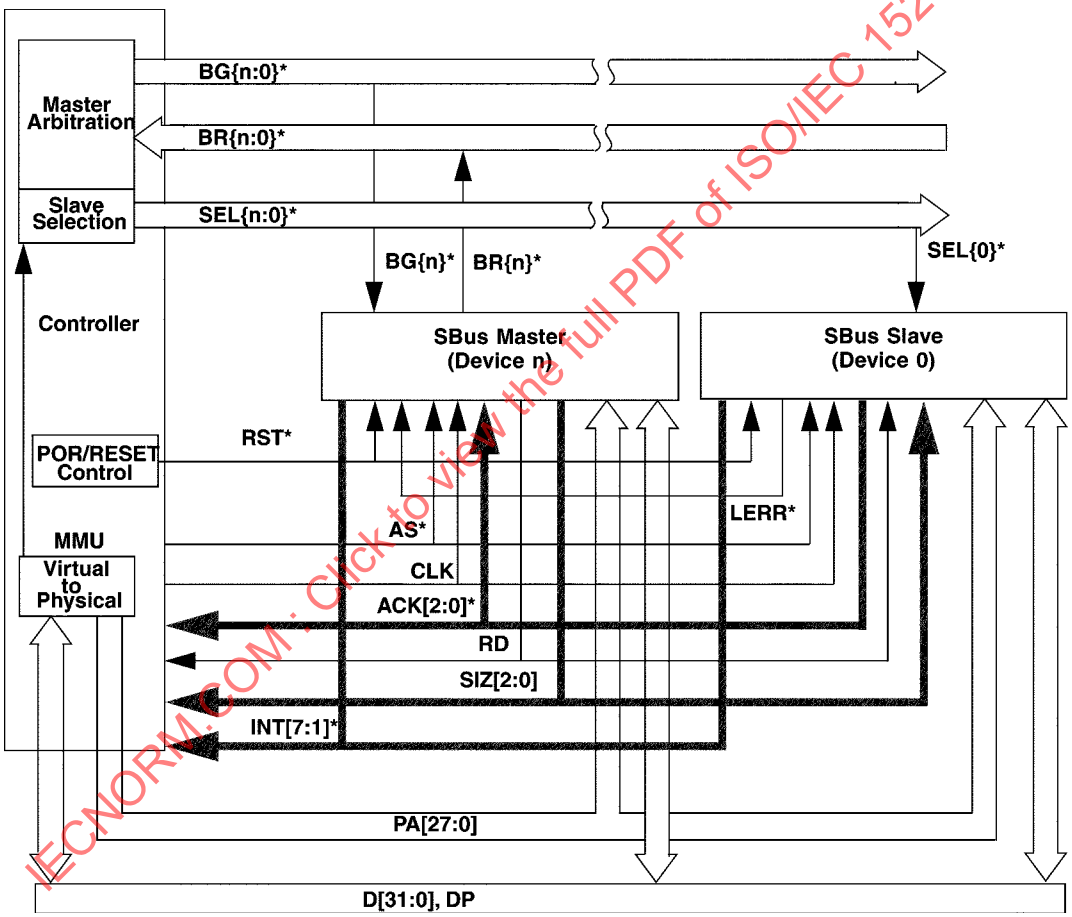


Figure 2 – SBus functional block diagram

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.

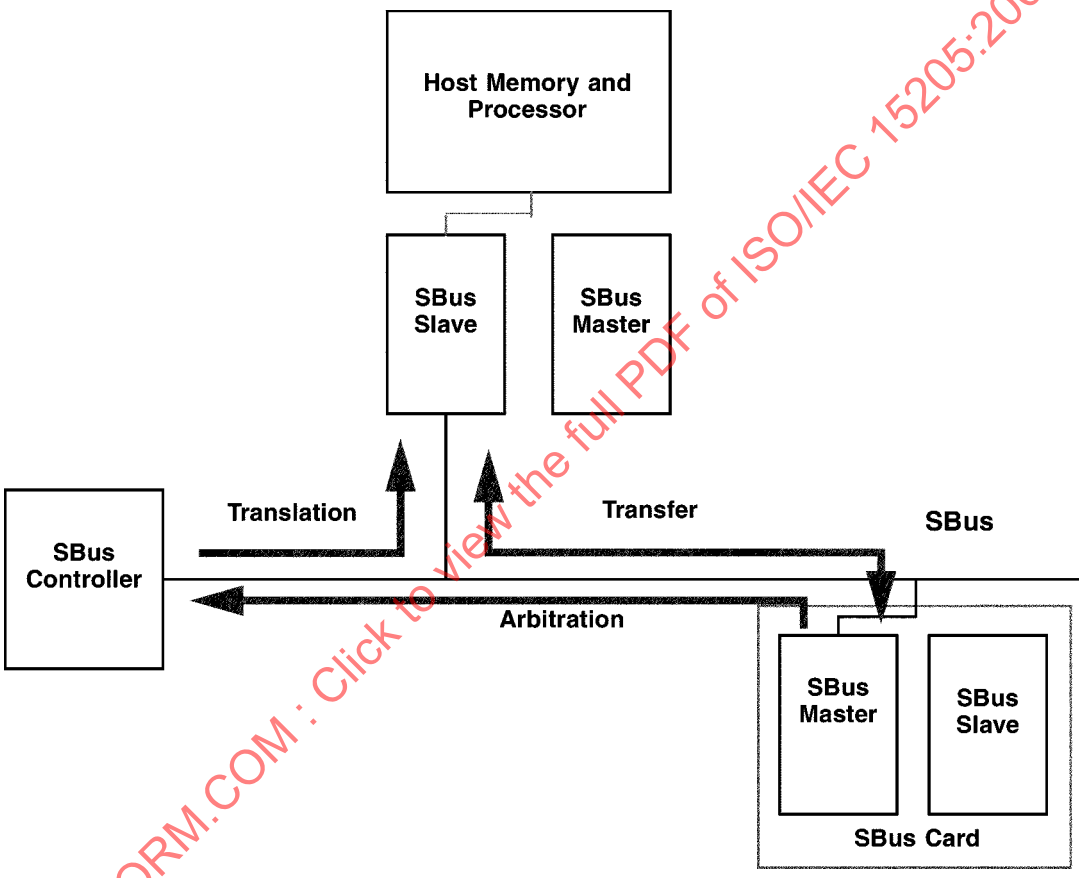


Figure 3 – Example of symmetric SBus cycle

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.

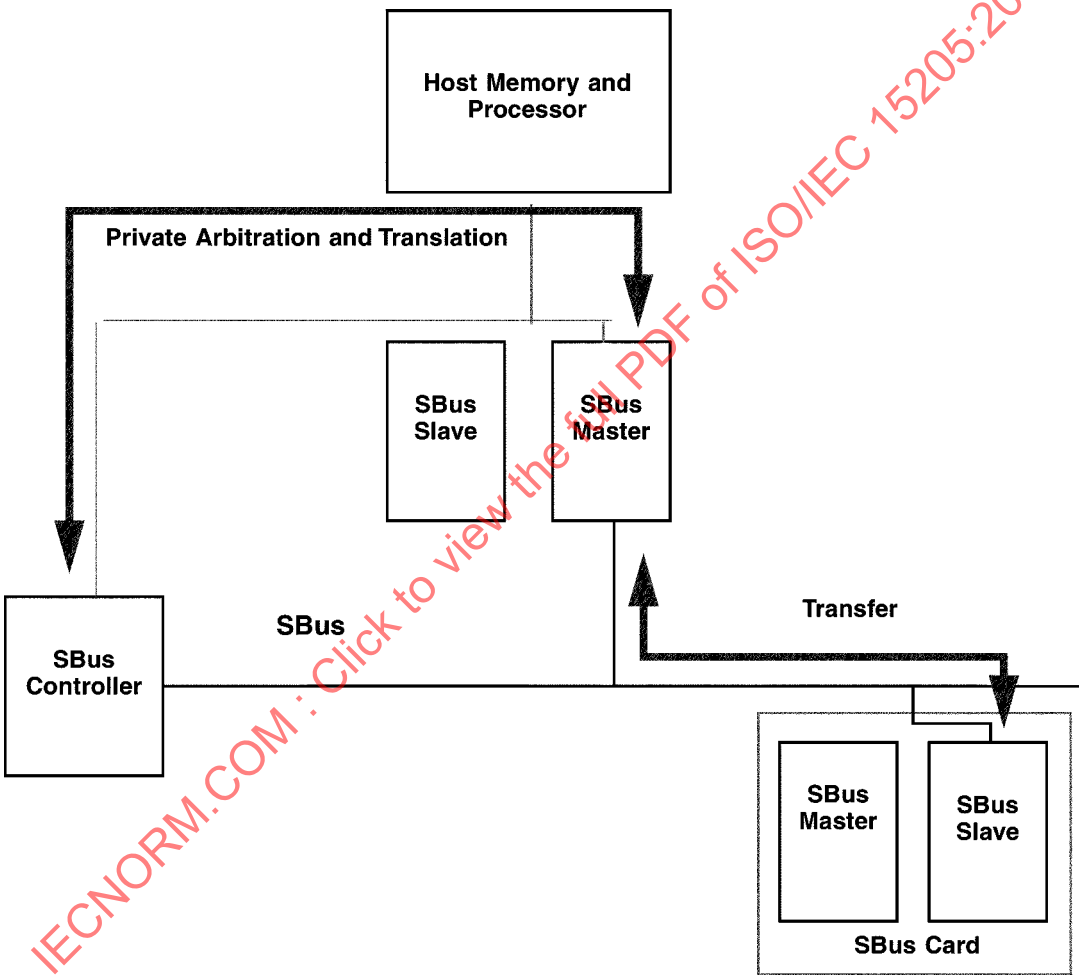


Figure 4 – Example of asymmetric SBus cycle

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.

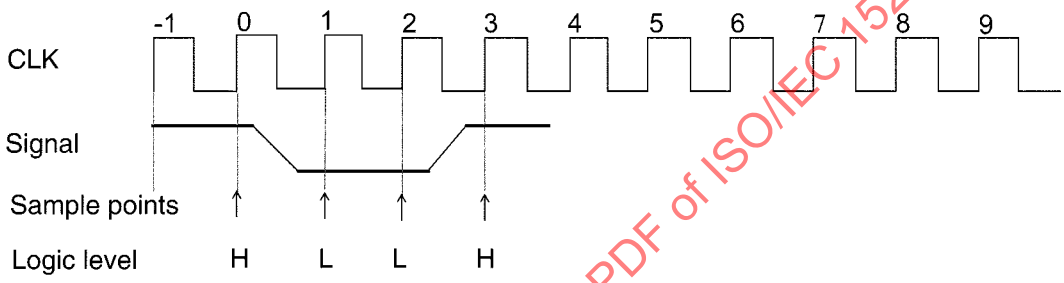


Figure 5 – Example of synchronous operation

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.

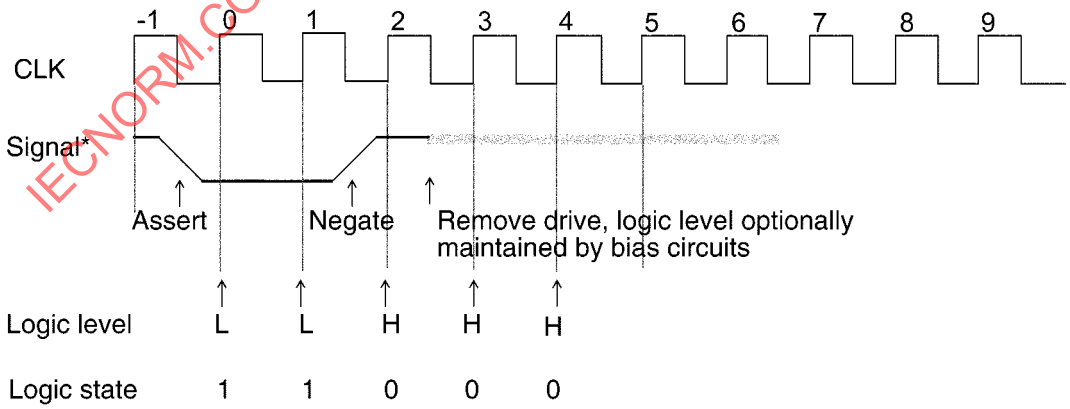


Figure 6 – Example of active negation on low-active control signal

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.

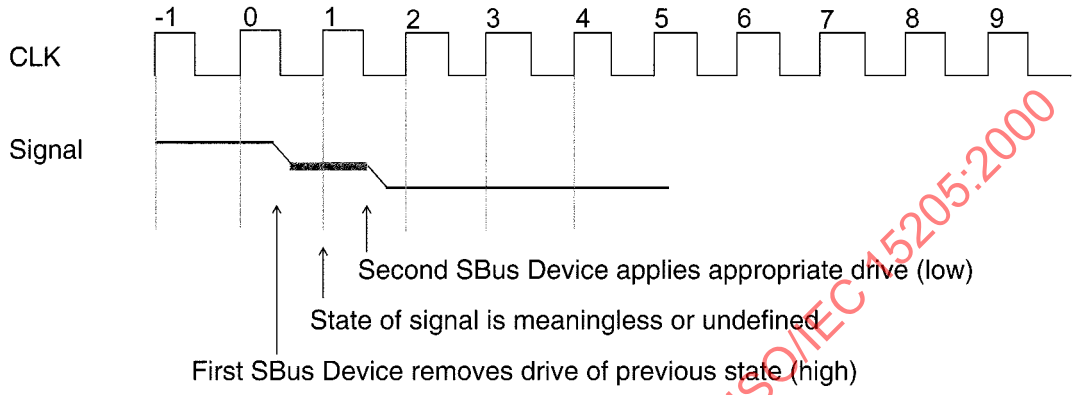


Figure 7 – Example of driver overlap protection

Buses 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.

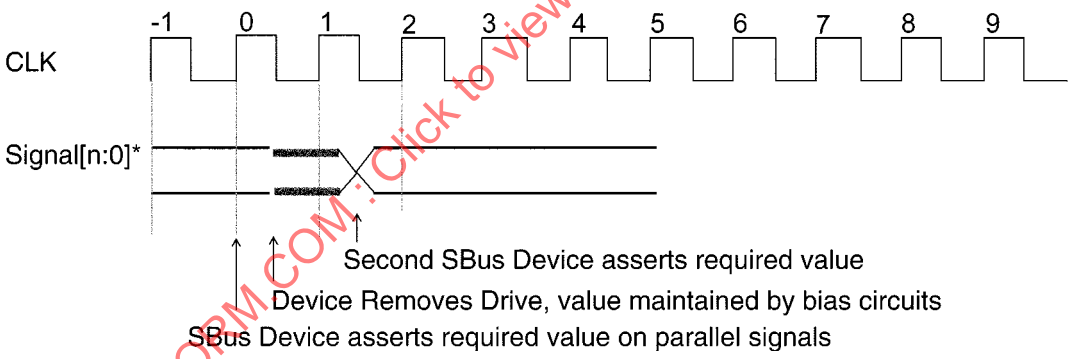


Figure 8 – Symbols for multiple signals operating as a unit

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.

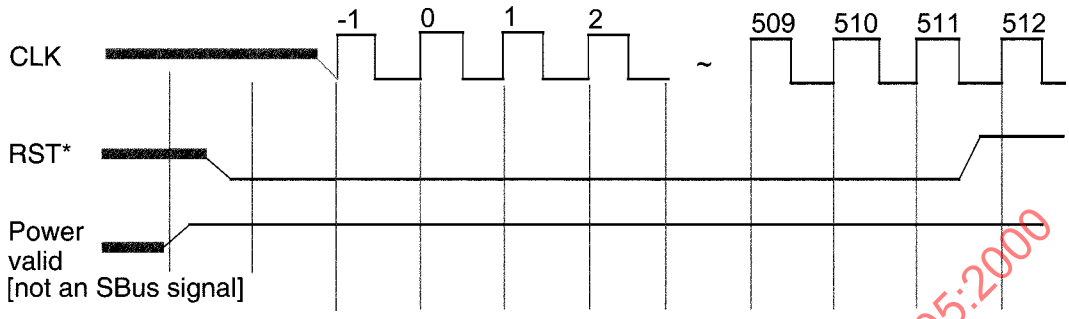


Figure 9 – Example of RST* timing

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.

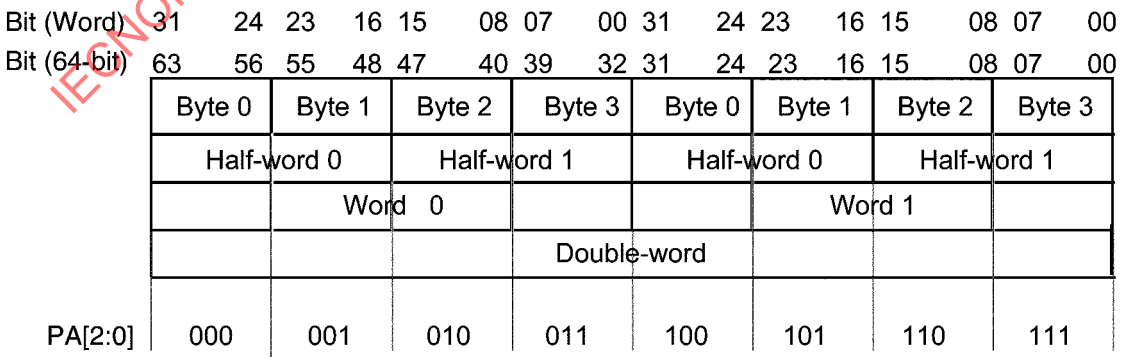


Figure 10 – Double-words, words, half-words, and bytes

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.

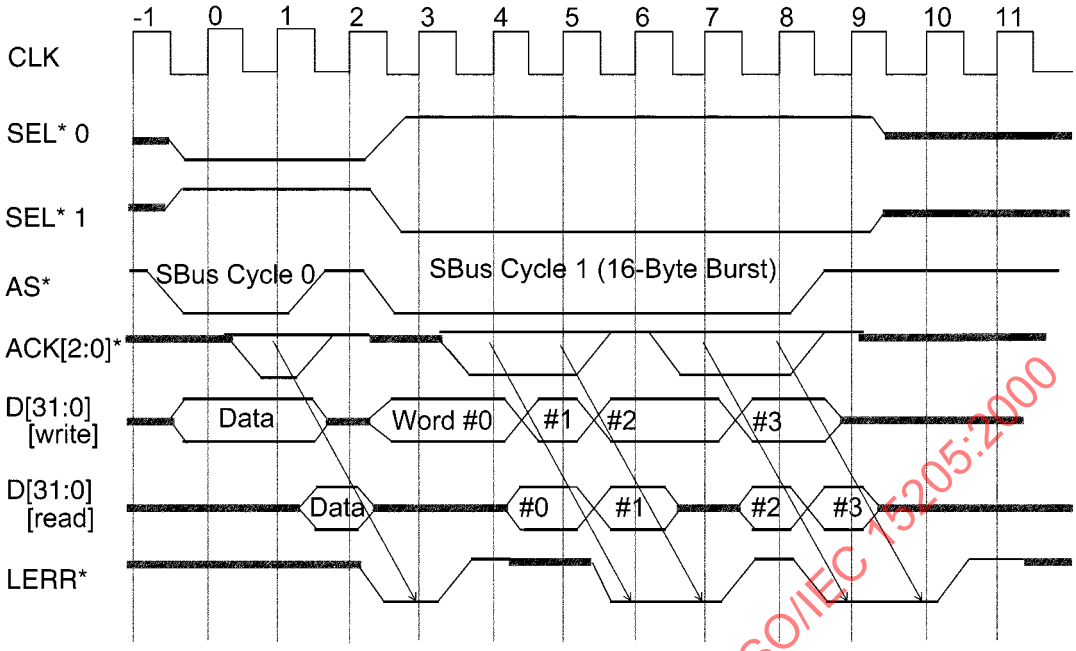


Figure 11 – Example of LERR* timing

4.13 INT[7:1]* signals

The seven low-asserted open-drain interrupt request signals, INT[7:1]*, are driven by SBus Devices to asynchronously signal the SBus System CPU through the SBus Controller. All interrupt request signals must be available to each SBus Card. SBus Devices on the motherboard are only required to have the relevant interrupt signals available. The INT[7:1]* signals may be bused to all SBus Devices or each SBus Device may drive its own private radial set of signals. The same interrupt signal may be asserted by more than one SBus Device at the same time. The mechanism for presenting the interrupts to the SBus System CPU and for identifying the actual SBus Device presenting an interrupt is not defined by this standard.

An SBus Device may assert one or more of the INT[7:1]* signals at any time. An SBus Device shall use open-drain output drivers to drive the INT[7:1]* signals. After an interrupt has been serviced, the SBus Device shall stop driving the interrupt line, which will allow the interrupt signal to be returned to the negated state by the specified pull-up resistors.

An SBus Device may ignore the setup and hold times with respect to CLK and may ignore the state of SEL* when asserting and releasing the interrupt signals.

Upon asserting an interrupt, the SBus Device shall have appropriate stored values, readable by the CPU through the SBus, to indicate that the Device is generating the interrupt. To assure a low-interrupt processing latency, Retry Acknowledgments should not be generated during SBus access to the interrupt indication register. The Slave port of the SBus Device shall include a mechanism controlled by the SBus System CPU to clear the interrupt. SBus Devices shall not remove an asserted interrupt until requested to do so by the SBus System CPU.

The highest priority interrupt to the SBus System CPU is generated by INT[7]*. The interrupt priority decreases for each signal down to INT[1]*, which generates the lowest priority interrupt.

The interrupt synchronization mechanism on the SBus Controller should be designed to prevent metastability from being a source of failures. When interrupt generation on the SBus Device is naturally synchronous with CLK, the Device should meet the standard SBus setup and hold times to further reduce the possibility of metastable behavior.

The SBus Slave port of an SBus Device should include a mechanism to enable and disable the presentation of interrupts to the SBus. Interrupts shall be disabled by the assertion of RST* and remain disabled until enabled by system CPU software.

5 SBus cycle definitions

Every SBus cycle begins with an Arbitration Phase managed by the SBus Controller to identify which SBus Master will be granted control of the SBus. Following the Arbitration Phase, the SBus Master and the SBus Controller cooperate in a Translation Phase to convert the virtual addresses used by the SBus Master to the physical addresses and selection signals understood by the SBus Slaves. After the Translation Phase, those SBus Masters and SBus Slaves performing Extended Transfers will then perform an Extended Transfer Information Phase. Finally, the SBus Master and the SBus Slave will cooperate in the Transfer Phase to perform and acknowledge the required data transfer. There are small differences in timing between 32-bit SBus Transfer Phases and 64-bit Extended Transfer Phases. The following sections provide detailed descriptions of each of these phases.

Asymmetric SBus cycles, described in 3.2.2, do not place the Arbitration and Translation Phases on the SBus, since the required information exchanges are performed privately among the combined SBus Master and Controller. On the SBus, asymmetric SBus cycles appear to be Transfer Phases only. Asymmetric Extended Transfers appear to be only an Extended Transfer Information Phase followed by a Transfer Phase.

SBus Systems conforming to the standard and meeting the interoperability requirements explained in 5.4.3 will operate correctly with any combination of 32-bit and 64-bit Extended Transfer SBus Masters, SBus Slaves, and SBus Controllers. SBus Masters and Slaves using only the 32-bit protocols will work correctly in systems that implement the 64-bit Extended Transfers and SBus Masters and Slaves using the 64-bit Extended Transfers will operate correctly in systems that implement only 32-bit Transfer Phases. SBus Masters and Slaves using 64-bit Extended Transfers can interchange information with 32-bit SBus Masters and Slaves.

5.1 Arbitration Phase

An SBus cycle begins with an Arbitration Phase that determines which SBus Master will have use of the SBus for the duration of the cycle. An Arbitration Phase begins when one or more SBus Masters indicate to the SBus Controller by asserting their respective BR* signals that they each require an SBus cycle. The SBus Controller selects one of the SBus Masters to use the SBus and asserts BG* to that SBus Master, giving it authority to perform an SBus cycle. The SBus Master determines that it has been selected when it finds that its BG* signal has been asserted and qualified by the rising edge of the CLK signal. The Bus Master that has been granted mastery of the SBus shall begin the Translation Phase at the next rising edge of the CLK signal by placing the proper virtual address and descriptive signals on the SBus.

The Arbitration Phase is identical for 32-bit transfer and for Extended Transfer cycles.

Arbitration among SBus Masters shall be fair, as defined by the following two rules:

– an SBus Master granted use of the bus during clock cycles T_m through T_n shall not be allowed to use the bus again until all other Masters which asserted their respective BR* signals during any clock $T \leq T_n$ have been granted use of the bus;

– within the above constraint, requests do not need to be processed in chronological order.

Once it has asserted BR*, an SBus Master shall leave the signal asserted until it receives BG*. During the clock cycle immediately following the assertion of BG*, the Master shall negate BR* for at least one clock cycle. The SBus Controller shall keep BG* asserted until the end of the bus cycle. If the SBus Master fails in such a manner that BR* is negated before BG* is asserted, the SBus Controller shall not assert BG* if it detects the condition before the cycle in which it would assert BG*.

Asymmetric SBus cycles perform arbitration between the SBus Controller and the integrated SBus Masters without exposing any signals on the SBus.

5.2 Translation Phase

A Translation Phase begins after the SBus Controller has granted access to an SBus Master by asserting BG* to the SBus Master. The selected SBus Master places the virtual address of the data to be transferred on the D[31:0] signals a setup time before the rising edge of the CLK signal immediately after the clock cycle that validated BG*.

In addition, the SBus Master places the proper values on the SIZ[2:0] and RD signals to instruct the SBus Controller what operations are to be performed. For Extended Transfers, the SBus Master provides this information to the SBus Controller for one cycle, during which, the SBus Controller obtains the values for these signals and the virtual address on that rising edge of the CLK signal. For 32-bit operations, the selected SBus Master provides the SIZ[2:0] and RD signals for the duration of the Translation Phase and the Transfer Phase.

The SBus Controller may then take an arbitrary number of clock cycles to translate the address and determine the proper SBus Slave to be selected and the proper physical address to present to the Slave. After translating the virtual address into a physical address, the SBus Controller begins the Transfer Phase of a 32-bit operation by asserting AS* and providing the physical address information required by the SBus Slave receiving the SEL* signal. For 64-bit Extended Transfers, the SBus Controller marks the final cycle of the Extended Transfer Information Phase and the subsequent beginning of the Extended Transfer Phase by asserting AS* and providing for one cycle the physical address information.

The SBus Master shall be ready for the Transfer Phase on the cycle after the virtual address is provided, even though several cycles may be used by the SBus Controller in translating the address and starting the next phase. If a 32-bit read operation is being performed, the SBus Master shall prepare for the Transfer Phase by releasing the D[31:0] signals so that data can be received. If a 32-bit write operation is being performed, the SBus Master shall prepare for the Transfer Phase by asserting the first write information on the D[31:0] signals. If an Extended Transfer is being performed, the SBus Master shall immediately assert the Extended Transfer Information on the D[31:0] signals and negate the RD signal in preparation to begin the Extended Transfer Information Phase.

If, as a result of a translation fault, the use of an invalid SIZ[2:0] value, or access violation, the SBus Controller needs to abort the bus cycle before entering the Transfer Phase or Extended Transfer Information Phase, it shall not assert SEL* to any SBus Slave and shall signal the presence of an error to the SBus Master with an Error Acknowledgment code placed on the ACK[2:0]* signals. The SBus Controller shall not assert AS* when aborting the cycle in this manner. The Error Acknowledgment may be presented to the SBus Master as early as the cycle after the beginning of the Translation Phase. Such errors are often the result of an incorrectly initialized SBus Master requesting actions or specifying addresses not supported by the SBus Controller.

Figure 12 shows an example of the presentation of an Error Acknowledgment by the SBus Controller. In this example, a new SBus cycle begins immediately after the Error Acknowledgment is presented.

If the SBus Controller requires a retry of a translation because translation resources are presently not available or need to be updated, the SBus Controller may request that the SBus Master retry the SBus cycle by presenting a Retry Acknowledgment on the ACK[2:0]* signals. The Retry Acknowledgment timing and cycle termination are exactly the same as the procedure for Error Acknowledgment described above. After the SBus cycle is terminated, the SBus Master follows the normal retry procedure.

The method for translating a virtual address to a physical address and associating the address with a particular SBus Slave is not defined by this standard. An SBus Controller shall provide for at least one size of translation page 65 536 bytes or smaller. This requirement allows designers of SBus Masters and Slaves to group addressable registers within a conveniently small page that may be protected through the virtual address to physical address mappings. Support for additional sizes of pages larger than this limit is allowed.

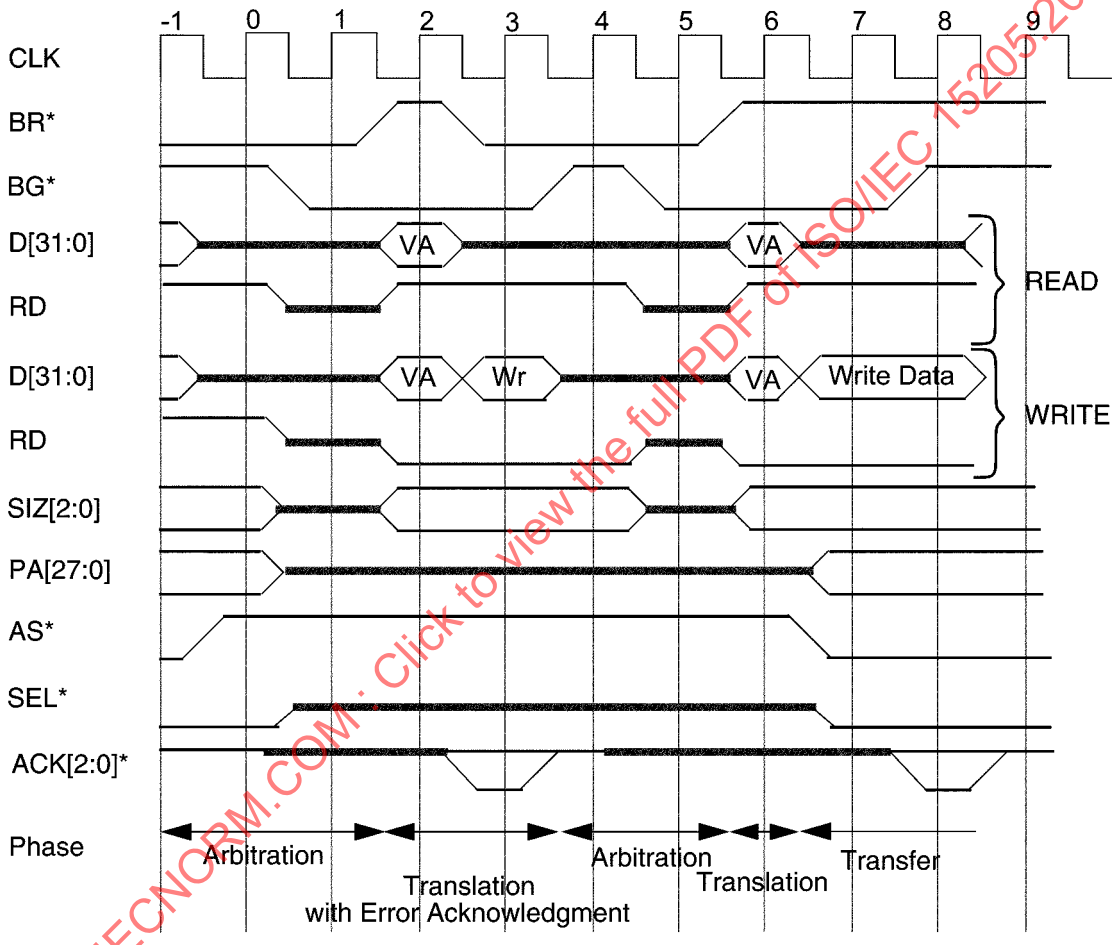


Figure 12 – Example of presentation of Error Acknowledgment by SBus Controller during Translation Phase

The end of the Translation Phase and the beginning of the Extended Transfer Information Phase or the Transfer Phase is indicated by the assertion of AS* and other information to the SBus Slave receiving SEL*.

SBus Systems having the SBus Controller integrated with an SBus Slave and performing a 32-bit transfer with that Slave need not assert AS* to indicate the end of the Translation Phase to the SBus.

Asymmetric SBus cycles perform the Translation Phase between the SBus Controller and the integrated SBus Masters without exposing any signals on the SBus itself.

Figure 13 shows the signal timing for a typical 32-bit symmetric SBus cycle. In this particular example, the assertion of PA[27:0], SEL*, and AS* by the SBus Controller is delayed by one wait cycle.

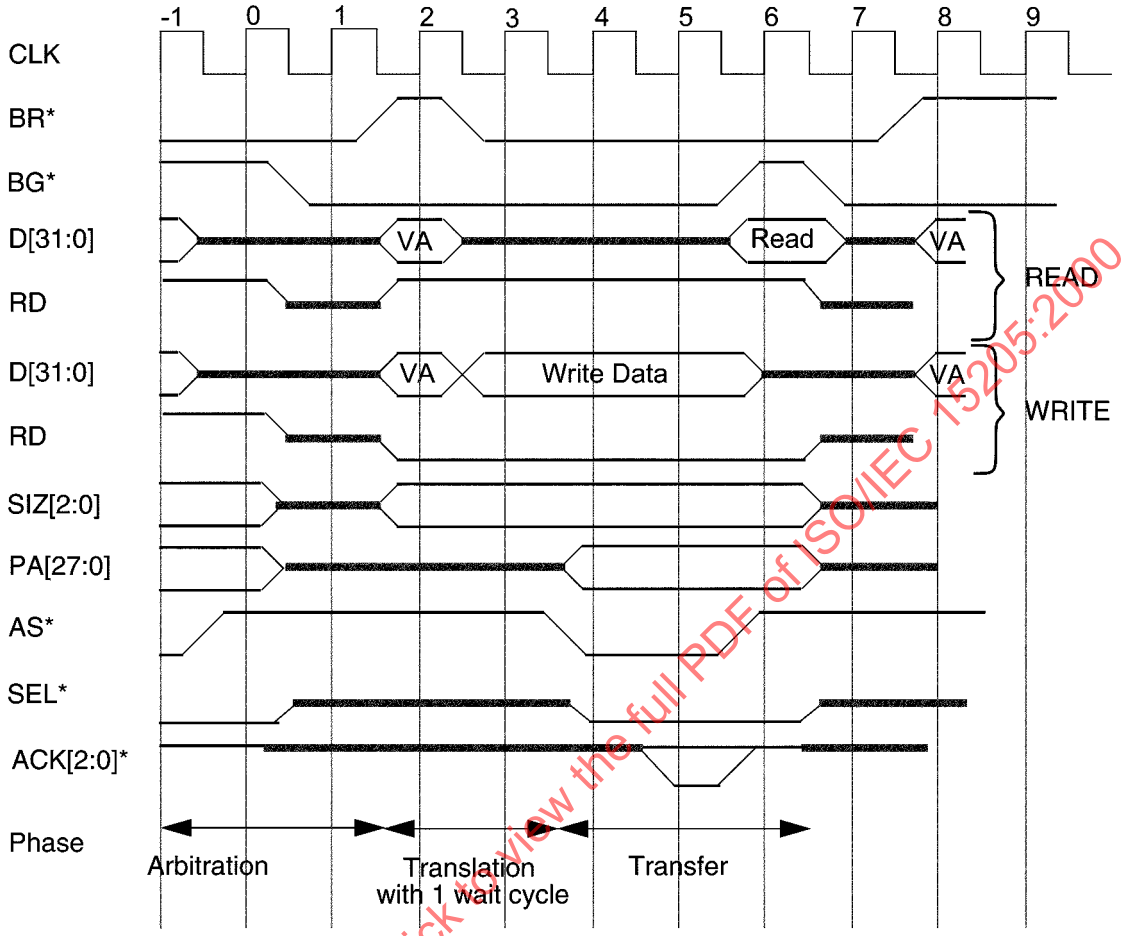


Figure 13 – Example of 32-bit symmetric SBus cycle

5.3 Extended Transfer Information Phase

The Extended Transfer Information Phase is used only for those SBus cycles requested by the current SBus Master that have a SIZ[2:0] value of Extended Transfer. The phase is not performed under any other conditions.

The Extended Transfer Information Phase provides the more complete information needed for the special data transfers of Extended Transfers, but not required for 32-bit transfers. The SBus Master shall present four bytes of Extended Transfer Information on the D[31:0] signals on the cycle after the SBus Master has provided the desired virtual address information on the same signals. The Extended Transfer Information shall remain presented until the SBus Controller has completed the address translation and has started the Transfer Phase by selecting the proper SBus Slave, providing the proper physical address on the PA[27:0] signals, and indicating that the information for the Transfer Phase is valid by asserting AS*. The size value of Extended Transfer and the Extended Information bytes remain asserted by the SBus Master and the physical address information remains asserted by the SBus Controller for the first SBus clock cycle that AS* is presented. The SIZ[2:0], RD, D[31:0] and PA[27:0] signals shall then be released so that data transfer may begin as soon as the next cycle.

The RD signal shall be negated by the SBus Master during the first cycle of the Extended Transfer Information Phase. The RD signal drivers may be disabled at any time between the negation and before the second cycle after AS* is presented, since the SBus bias circuits maintain the RD signal in the negated state until the signal is used as part of the subsequent data transfer.

Asymmetric SBus cycles begin the Extended Transfer Information Phase without exposing previous internal arbitration and translation processes on the SBus. The beginning of the Extended Transfer Information Phase is indicated by the assertion of AS*.

Figure 14 shows typical timings for a 64-bit symmetric SBus cycle.

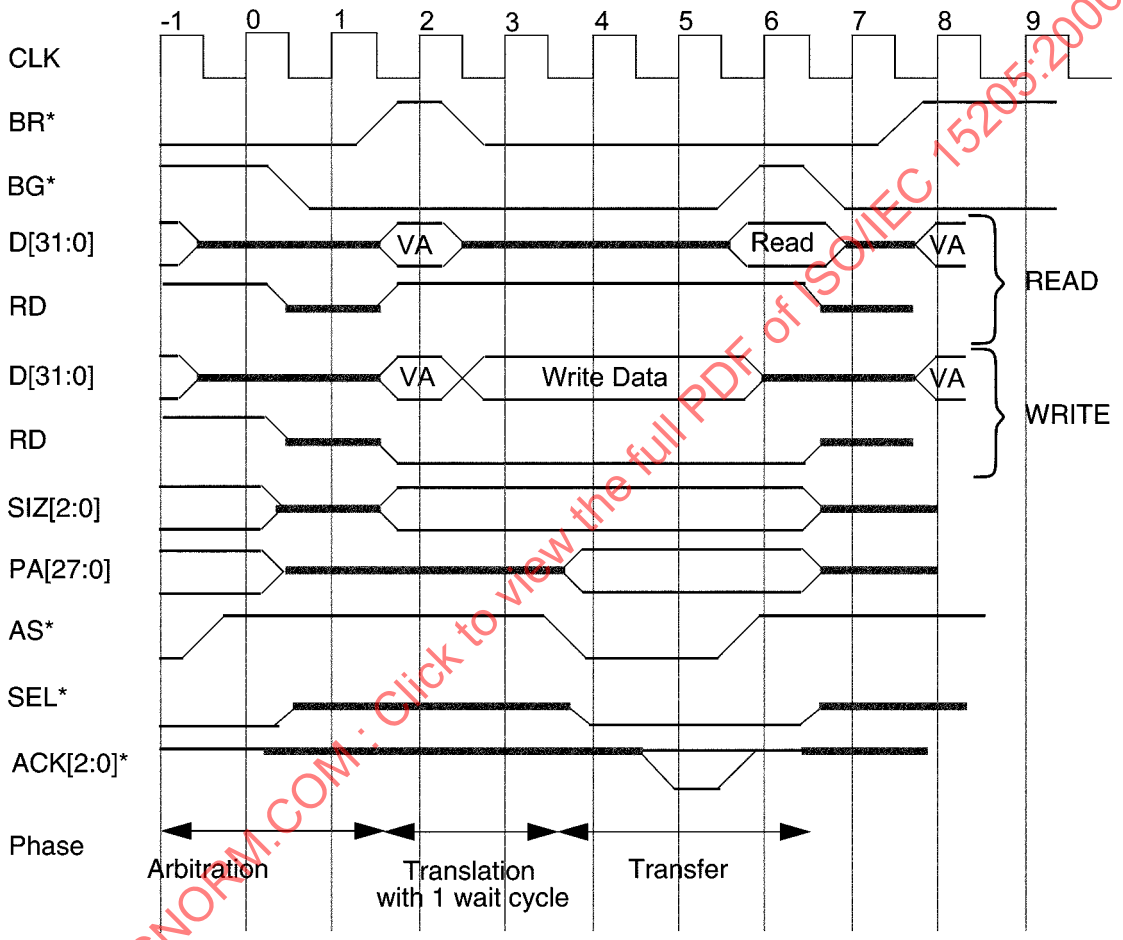


Figure 14 – Example of 64-bit Extended Transfer symmetric SBus cycle

SBus Slaves shall latch the Extended Transfer Information and associated information describing the Transfer Phase to be performed on the rising edge of the CLK signal when AS* is first asserted. The Extended Transfer Information format is summarized in the following table and described in detail in the following subclauses. The reserved bits are reserved for use by future revisions of this standard and shall be set to 0 by the SBus Master. If an SBus Slave determines that unsupported or reserved fields or field values are set, the SBus Slave shall respond with an Error Acknowledgment.

The SBus Controller may also examine the Extended Transfer Information for validity in the following manner.

The SBus Controller may extend the translation cycle for one or more cycles, allowing the Controller to analyze the Extended Transfer Information provided by the SBus Master. If it detects any errors in the information, it may immediately present an Error Acknowledgment, terminating the SBus cycle without entering the Transfer Phase as described in 5.2. This mechanism contributes an extra clock cycle of overhead to every Extended Transfer SBus cycle, but provides early notification of errors to the SBus Master and prevents the use of invalid Extended Transfer Information by an SBus Slave.

The Extended Transfer Information is summarized in Table 5.

Table 5 – Extended Transfer Information

Signal line(s)	Transfer Information field
D[31]	Extended Type
D[30:28]	Extended Transfer Count[2:0]
D[27]	Extended Read
D[26:25]	Extended Lock[1:0]
D[24:0]	For Extended Type = 0 and Extended Transfer Count value valid, D[24:0] are undefined and may be any value
	For Extended Type = 0 and Extended Transfer Count value reserved, D[24:0] are reserved
	For Extended Type = 1, D[24:0] are reserved

5.3.1 Extended Type field

During an Extended Transfer Information Phase, the Extended Type field shall be set to specify the type of Extended Transfer that is to be performed. The Extended Type field is a 1-bit field transferred on the D[31] signal. This standard defines only the 64-bit Transfer Type. Other transfer types may be defined by extensions of this standard using the reserved Extended Type value. SBus Masters performing Extended Transfers shall not use the reserved value of the Extended Type field. Slaves that support Extended Transfers shall decode the Extended Type value. If the reserved Extended Type value is received by an SBus Slave, the Slave shall generate an Error Acknowledgment.

Table 6 shows defined Extended Type values.

Table 6 – Extended Type values

Extended Type, D[31]	Function
0	64-bit Transfer
1	Reserved

5.3.2 Extended Transfer Count field

During an Extended Transfer, the value of the 3-bit Extended Transfer Count field, not the encoding of the SIZ[2:0] signals, shall determine how many units of data are transferred during the Transfer Phase of the SBus cycle. Extended Transfer Count field bits 2, 1, and 0 are transferred on D[30], D[29], and D[28], respectively. Table 7 indicates the values that shall be set to establish the number of units of data to be transferred.

The requirements for SBus Masters, SBus Slaves, and SBus Controllers with respect to the selection of Extended Transfer Count values to be implemented is explained in 5.4.3.

One, two, and four byte transfers are not supported by 64-bit Extended Transfers, but are only supported using the 32-bit SBus transfer protocols. SBus Masters shall not request an Extended Transfer Count value that is not supported by the SBus Master. SBus Masters should not request an Extended Transfer Count value that is reserved or not supported by the SBus Slave to be selected. SBus Slaves shall issue an Error Acknowledgment if they receive an Extended Transfer Count value requesting a data transfer size that the SBus Slave does not support or that is reserved.

Table 7 shows Extended Transfer Count values. The SBus Controller may present an error condition as described in 5.3 if the Controller verifies that the Extended Transfer Count value is reserved.

Table 7 – Extended Transfer Count values

Extended Transfer Count			Number of transfers	Byte transferred, 64-bit Extended Transfer Type
ETC[1] D[30]	ETC[1] D[29]	ETC[0] D[28]		
0	0	0	Reserved	
0	0	1	Reserved	
0	1	0	Reserved	
0	1	1	1	8 bytes
1	0	0	2	16 bytes
1	0	1	4	32 bytes
1	1	0	8	64 bytes
1	1	1	16	128 bytes

5.3.3 Extended Read field

During an Extended Transfer, the value of the 1-bit Extended Read field shall determine the direction of data transfer between the SBus Master and the SBus Slave. The Extended Read field is transmitted on the D[27] signal. The 1 value indicates a read from the SBus Slave to the SBus Master is to be performed. The 0 value indicates a write from the SBus Master to the SBus Slave is to be performed.

The value of Extended Read field shall be identical to the value the SBus Master drove onto the SBus signal RD during the initiation of the Translation Phase of the SBus cycle. The SBus Controller may present an error condition as described in 5.3 if the SBus Controller verifies that the Extended Read field is invalid.

5.3.4 Extended Lock field

The description of the function and required protocol to perform resource locking using the Extended Lock field is contained in 5.7.

During an Extended Transfer, the 2-bit Extended Lock field shall be used to control an SBus Slave's Lock flag associated with the resource identified by PA[27:0]. When the Lock flag is set, the SBus Slave shall permit access only in accordance with the locking protocol and only through the SBus port for which the Lock flag was set.

Table 8 shows defined Lock field values.

Table 8 – Extended Lock values

Extended Lock		Lock field values
EL[1] D[26]	EL[0] D[25]	
0	0	Normal Transfer
0	1	Initiate Lock
1	0	Maintain Lock
1	1	End Lock

Any SBus Master performing a sequence of locked transfers shall perform the first transfer of the sequence with the Extended Lock value set to Initiate Lock. This sets the SBus Slave's Lock flag for that port and address space. All except the last of subsequent transfers to the locked SBus Slave shall use the Maintain Lock value. This maintains the Lock flag in the set state and additionally permits the requested transfer to be performed. The last transfer of the locked sequence shall have the Extended Lock field value set to End Lock. After that SBus cycle is completed, the Lock flag shall be reset and normal operation shall continue.

5.4 Transfer Phase

The Transfer Phase begins when the SBus Controller has completed the translation of the virtual address to select the proper SBus Slave and has generated the proper physical address. The other information required for completion of the cycle is provided by the signal lines RD and SIZ[2:0] for 32-bit transfers and by the Extended Transfer Information on D[31:0] for Extended Transfers. The SBus Controller then asserts the SEL* signal to the proper SBus Slave, provides the required physical address information on PA[27:0], and asserts AS* to validate the state of those signals and begin the Transfer Phase.

The Transfer Phase uses the D[31:0] signals or the D[63:0] signals to transfer the requested information between the SBus Master and the SBus Slave. Each data transfer is associated with the presentation of a single Data Acknowledgment for that particular transfer. The precise timing of the presentation of data with respect to the Data Acknowledgment varies depending on whether the operation is a 32-bit transfer or a 64-bit Extended Transfer and on whether the operation is a read or a write. A detailed description of the Transfer Phase for each case is provided in the following sections.

The Transfer Phase ends normally when the last Data Acknowledgment required for the Transfer Phase has been transmitted. The SBus Controller, which monitors the Acknowledgments to determine when the Transfer Phase is completed, then negates the AS* signal to remove the SBus Slave and negates the BG* signal to remove the SBus Master from the SBus. The AS* signal shall be negated at the same time as or before the BG* signal is negated and one or more clock cycles before the SEL* signal is negated. The SBus Controller shall end the Transfer Phase only when all requested data has been transferred, or when a Retry Acknowledgment, an Error Acknowledgment, or a timeout has occurred.

Asymmetric 32-bit SBus cycles begin the Transfer Phase immediately without exposing previous internal arbitration and translation processes on the SBus. The beginning of the Transfer Phase is indicated by the assertion of AS* to the SBus Slave receiving the SEL* signal. The proper signals shall be established by the SBus Controller and the SBus Master to provide the SBus Slave the information required to execute the Transfer Phase. If an Extended Transfer is being performed, an Extended Transfer Information Phase shall be executed before the Transfer Phase is performed.

SBus Systems having an SBus Controller integrated with an SBus Slave port may choose during 32-bit transfers to not present PA[27:0], SEL* or AS* on the SBus, since those signals can be transmitted among the integrated Devices directly. The physical address signals should be presented on the SBus so that SBus analysis tools can capture the relevant transfer addresses.

5.4.1 32-bit Transfer Phase

The 32-bit Transfer Phase begins when the SBus Controller places the proper physical address on the PA[27:0] signals, generates the SEL* signal to the SBus Slave to be addressed, and asserts the AS* signal. SEL* and PA[27:0] shall be asserted no later than the clock cycle on which AS* is asserted. The SBus Master shall assert or continue to assert the SIZ[2:0] signal indicating any data transfer size except Extended Transfer, and the RD signal to indicate whether the data transfer is to be a read from the SBus Slave or a write to the SBus Slave. The SIZ[2:0] and RD signals shall be asserted to the correct values during the entire Transfer Phase.

The selected SBus Slave then has up to the timeout number of clock cycles to complete the requested transfer and issue all required Acknowledgments. There are the timeout number of clock cycles available to transfer data during a complete Transfer Phase, not the timeout number of clock cycles per data transfer within a burst. In the case of a Burst Transfer, the Slave generates one Word Acknowledgment for each word transferred. For single word transfers or for Burst Transfers where the information cannot be transferred at the full clock rate, the SBus Slave then drives ACK[2:0]* back to the Idle state after each Data Acknowledgment. On the last Data Acknowledgment, the SBus Slave drives the ACK[2:0]* signals to the Idle state and, in the following clock cycle, releases the drivers. The Slave may assert LERR* two clock cycles after the assertion of the corresponding Data Acknowledgment or Error Acknowledgment.

The SBus Controller shall keep AS* asserted until after the SBus Slave gives its final Data Acknowledgment or until the SBus Slave or SBus Controller asserts a Retry or Error Acknowledgment. If the SBus Controller is integrated with the SBus Slave, the SBus Controller may or may not have asserted AS*. The SBus Controller shall monitor SIZ[2:0] and ACK[2:0]* to be able to identify which Data Acknowledgment is the last one or to detect a Retry or Error Acknowledgment. The SBus Controller shall maintain PA[27:0] and SEL* in their correct state at least one hold time after the rising edge of CLK that will first identify that the AS* signal has been negated.

The SBus Master shall maintain SIZ[2:0] and RD in their correct state until the clock cycle after BG* is negated. During the clock cycle following the rising edge of CLK detecting the negation of BG*, the current SBus Master shall release SIZ[2:0], and RD. The SBus Controller must keep BG* asserted at least as long as AS* so that the SIZ[2:0] and RD signals will be valid during the entire period that AS* is asserted.

SBus Masters executing symmetric 32-bit SBus cycles shall use BG* to perform all phase sequencing. The SBus Master first uses the assertion of BG* to know when to place the virtual address on the D[31:0] signals and the correct values on the SIZ[2:0] and RD signals during the Translation Phase. The SBus Master shall use the negation of BG* to indicate that it shall release RD, SIZ[2:0], and D[31:0] at the end of the Transfer Phase. The Master shall not use or make assumptions about AS* during a Transfer Phase. The SBus Master uses ACK[2:0]* signals from the SBus Slave or the SBus Controller to manage its data transfers.

The relative timing of AS*, SEL*, PA[27:0], SIZ[2:0], and RD is intended to allow an SBus Slave to use the AS* as a validating signal for the other signals such that the signals do not need to be latched. AS* shall only be considered asserted or negated at the rising edge of CLK. This requires that SEL*, PA[27:0], SIZ[2:0], and RD be valid at least a setup time before the rising edge of CLK detects the assertion of AS* and remain valid at least a hold time after the rising edge of CLK that first detects the negation of AS*, as shown in Figure 13.

When the SBus Master reads data from an SBus Slave, the SBus Slave generates a Data Acknowledgment to indicate to the Master that the data values will be valid on D[31:0] on the rising edge of the CLK signal immediately following the rising edge that validates the assertion of the Data Acknowledgment. The D[31:0] signals may be driven to a valid state at any time before the clock cycle after the corresponding Data Acknowledgment, but shall have the data valid a setup time before the rising edge of the CLK signal after the corresponding Data Acknowledgment. Only one clock cycle of Data Acknowledgment shall be driven for each data unit to be read. Data Acknowledgment may be asserted any time after the CLK edge that validates the assertion of AS* and before the end of the timeout period.

When the SBus Master writes data to an SBus Slave, the SBus Master shall drive the first word of data on the D[31:0] signals on the cycle following the presentation of the virtual address during the Translation Phase. If the SBus Master and SBus Controller are performing an asymmetric SBus cycle, the first word of data shall be driven onto the D[31:0] signals no later than the same clock cycle in which AS* is asserted. The SBus Master shall keep the write data stable until the SBus Slave has provided the correct Data Acknowledgment. If a Burst Transfer is being performed, the SBus Master shall provide the next word of data on the next cycle after the Data Acknowledgment for the previous data has been asserted and again maintain the data valid until the Data Acknowledgment is again provided for that next word. Whether the operation is a single transfer or a Burst Transfer, the SBus Slave shall not expect the data to be valid after the Slave has provided the Data Acknowledgment for that data. On the last Data Acknowledgment of a Transfer Phase, the D[31:0] signals shall be released by the SBus Master no later than the cycle after BG* has been negated.

For Burst Transfers, an SBus Slave capable of transferring a word per clock cycle keeps ACK[2:0]* asserted for each rising edge of CLK as a word is transferred. During write operations, the SBus Slave is acknowledging data on the data lines during the rising edge of CLK it is asserting the Data Acknowledgment. Thus, the Slave samples the data at the same time the Master samples the Acknowledgment for that data. During read operations, the Acknowledgment is pipelined. Thus, the SBus Slave first generates the Data Acknowledgment and, for the following rising edge of CLK, drives the data lines.

SBus Slaves not able to perform data transfers at the clock rate shall drive ACK[2:0]* back to the idle state during the times that data is not valid or cannot be sensed on the D[31:0] signals. In all cases, after the final Data Acknowledgment, the Slave shall drive the Idle Acknowledgment on ACK[2:0]* for one clock cycle, and then release the ACK[2:0]* signals.

Figure 15 shows typical Transfer Phase timings. In this particular example, the SBus Controller negates AS* three cycles later than the earliest possible timing, as allowed by 4.5 and 4.7.

A minimum length asymmetric SBus cycle requires three clock cycles:

a) the first for asserting AS*;

b) the second for asserting ACK[2:0]*;

c) the third for negating AS*.

This timing allows *n* words to be transferred in *n* + 2 clock cycles, assuming that a read is not followed immediately by an asymmetric write cycle. Figure 16 shows this normal case.

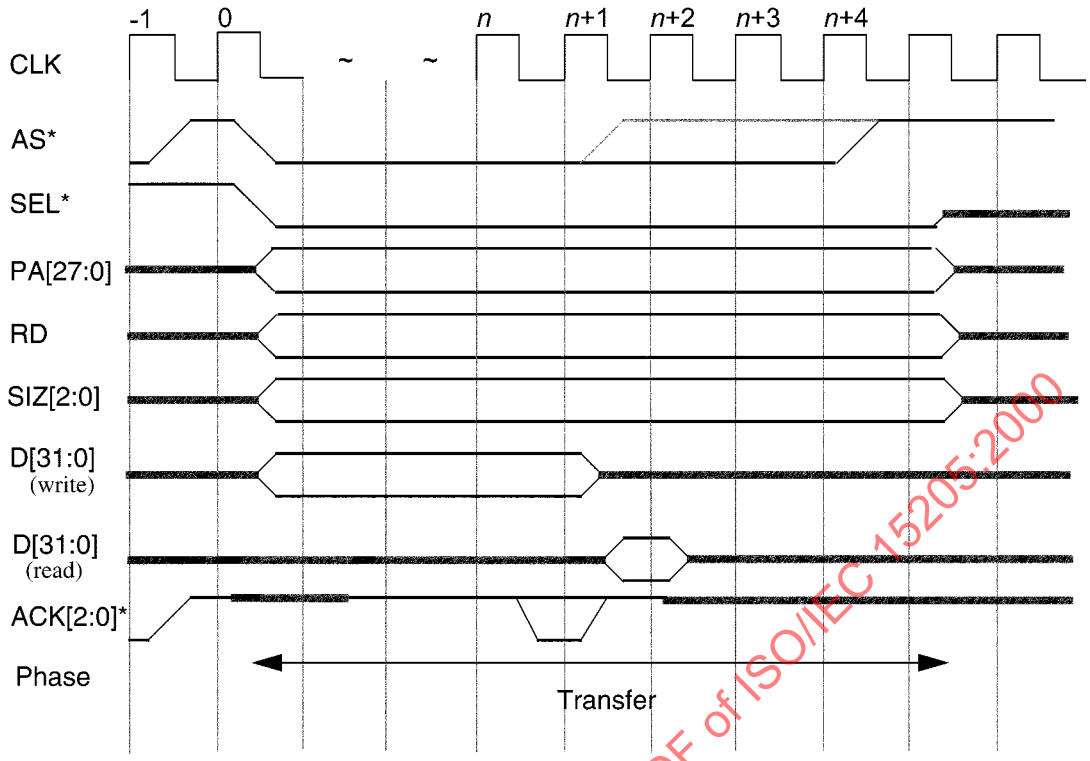


Figure 15 – 32-bit Transfer Phase, asymmetric SBus cycle

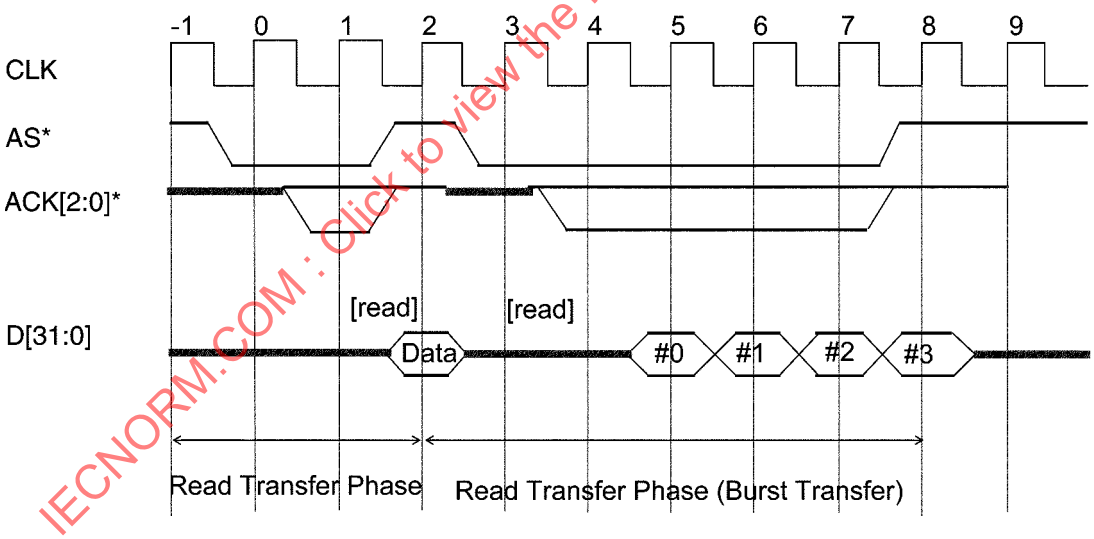


Figure 16 – Read following read, fastest case 32-bit asymmetric SBus cycle

For the special case of an asymmetric write SBus cycle following a read, one additional unused clock cycle is required to prevent two different Devices from simultaneously driving D[31:0]. Figure17 shows such a case.

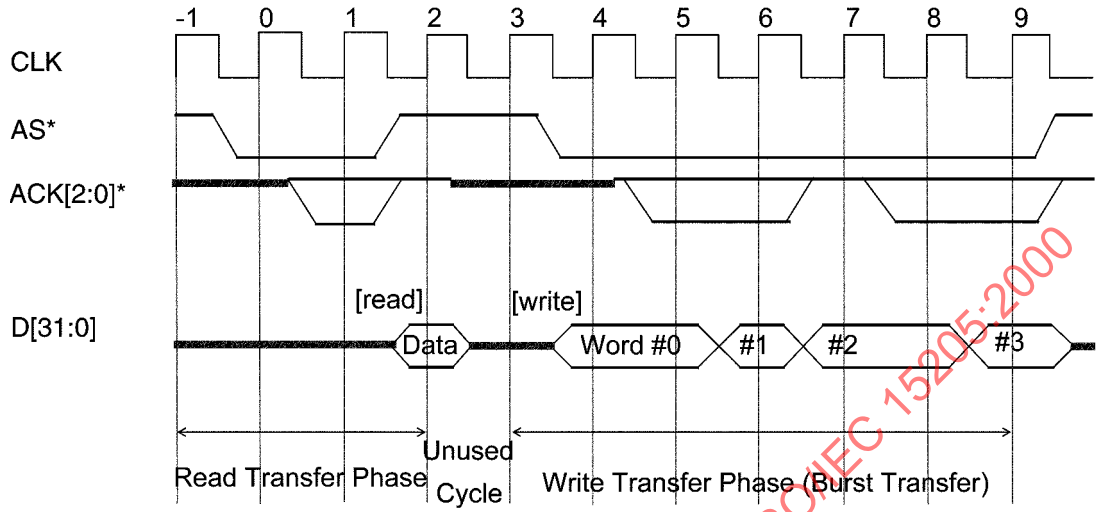


Figure 17 – Write following read, asymmetric SBus cycle

5.4.2 64-bit Extended Transfer Phase

The Extended Transfer Phase begins when the SBus Controller places the proper physical address on the PA[27:0] signals, generates the SEL* signal to the SBus Slave to be addressed, and asserts the AS* signal. The SEL* and PA[27:0] signals shall be asserted no later than the cycle on which AS* is asserted. The SBus Master has already provided the Extended Transfer Information on D[31:0], which contains the necessary information to perform the Transfer Phase.

The SBus Slave shall latch three pieces of information on the rising edge of the CLK signal that validates the assertion of the AS* signal:

a) the physical address (present on PA[27:0]);

b) the fact that an Extended Transfer is being performed (indicated by SIZ[2:0]);

c) the Extended Transfer Information (present on D[31:25]).

Unlike 32-bit transfers, where this information remains valid throughout the bus cycle, in an Extended Transfer this information is valid only at the clock edge validating the assertion of AS*. During the clock cycle after the assertion of AS*, the SBus Controller shall release the PA[27:0] signals. The SBus Master shall release the SIZ[2:0], RD, and D[31:0] signals. Those signals are then used to carry D[63:32] as shown in Table 2. The SBus Controller shall have pull-down resistors on RD and SIZ[2], and pull-up resistors on SIZ[1:0] to maintain them in the proper state until they are driven as part of the Extended Transfer data bus. Holding amplifiers may be used as described in 6.2.3. The only type of data transfer protocol presently defined for the Extended Data Transfer is the 64-bit data transfer.

The selected SBus Slave then has up to the timeout number of clock cycles to complete the requested transfer and issue the proper Data Acknowledgment on ACK[2:0]*. For 64-bit type data transfers, the Data Acknowledgment shall be a Double-word Acknowledgment. In the case of a Burst Transfer, the Slave generates one Double-word Acknowledgment for each double-word transferred. For single double-word transfers or for Burst Transfers where the information cannot be transferred at the full clock rate, the SBus Slave then drives ACK[2:0]* back to the Idle state after the Double-word Acknowledgment until the next Double-word Acknowledgment is valid. On the last Double-word Acknowledgment, the SBus Slave drives the ACK[2:0]* signals to the Idle state and, in the following clock cycle, releases the drivers.

The Slave may assert LERR* for one rising edge of the CLK signal, two clock cycles after the assertion of the corresponding Double-word Acknowledgment or Error Acknowledgment.

SBus Slaves that do not support Extended Transfers shall issue an Error Acknowledgment to an SBus Master that requests a SIZ[2:0] code of Extended Transfer.

An SBus Master performing Extended Transfers shall use the assertion of BG* to know when to place the virtual address on the D[31:0] signals, the Extended Transfer value on the SIZ[2:0] signals and the proper state on the RD signal during the Translation Phase. Unlike 32-bit transfers, the SBus Master performing Extended Transfers shall also monitor the assertion of AS* to determine when to stop driving SIZ[2:0] and the Extended Transfer Information on D[31:0]. The SBus Master shall use the negation of BG* to determine that the Extended Transfer Phase is completed.

Figure 18 shows an example of a 64-bit asymmetric Extended Transfer SBus cycle.

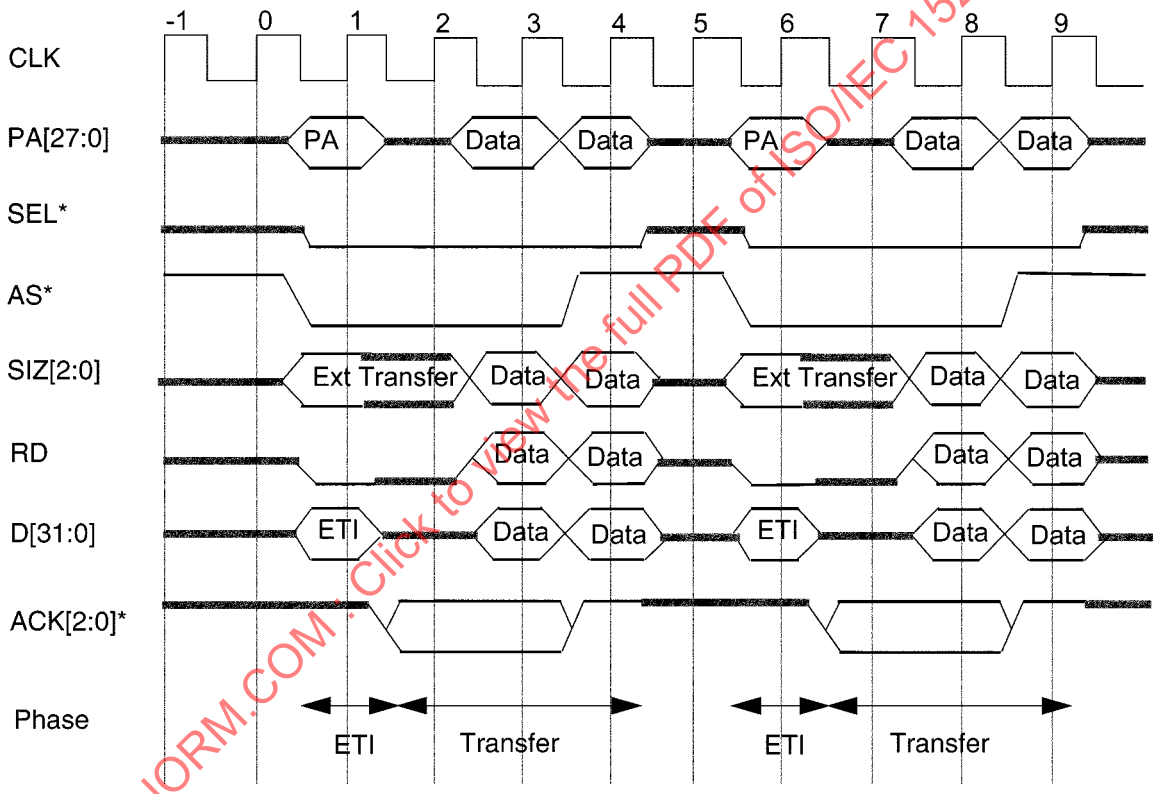


Figure 18 – Example of asymmetric 64-bit Extended Transfer SBus cycle

During a 64-bit read operation, the SBus Slave shall drive D[63:0] with 8 bytes of data meeting the proper setup and hold times for the CLK edge following the Double-word Acknowledgment. Thus, Extended Transfers during a read operation follow the same timing as 32-bit read transfers.

The 64-bit Extended Transfer uses the SBus signals as shown in Table 2 to implement a bus width of 64 bits. D[63] is the most significant bit of the double-word.

During a write operation, the SBus Master shall drive D[63:0] with 8 bytes of data meeting the proper setup and hold times for the CLK edge following the SBus Slave's Double-word Acknowledgment. Extended Transfers during a write operation do not follow the same timing as 32-bit write transfers, but instead use the same timing as 64-bit read transfers. The later receipt of data may require that an SBus Slave present errors during write operations with LERR* instead of Error Acknowledgment.

For Burst Transfers, an SBus Slave capable of transferring a double-word for each rising CLK edge keeps ACK[2:0]* asserted for each clock cycle as a word is transferred. During both read and write operations, the Acknowledgment is pipelined. Thus, the SBus Slave first generates the Double-word Acknowledgment and, using the following rising CLK edge for timing, transmits or receives the D[63:0] signals.

Extended Transfer 64-bit operations are required to transfer binary multiples of 64 bits. If the SBus Master or the SBus Slave require access to a smaller data field than 64 bits, 32-bit Transfer Phases shall be used. Only the Double-word Acknowledgment is valid during 64-bit Extended Transfers. Bus Sizing shall not be implemented for Extended Transfers.

For each Slave, the physical address space for Extended Transfers shall be the same as for 32-bit transfers, making it possible to access data using the same physical address with either 32-bit transfers or Extended Transfers.

5.4.3 Data size and Acknowledgments

Each type of SBus Device has requirements on its capability to generate, accept, and manage the various values of SIZ[2:0], ACK[2:0]*, and Extended Transfer Count. Depending upon the particular implementation selected, the SBus Device may either interoperate freely with all SBus Devices or may operate only with those having compatible characteristics.

SBus Masters use the SIZ[2:0] signals to select a preferred data width for a single unit of data transfer or to select a required number of word transfers to be performed in a Burst Transfer. When performing a single data unit transfer, the SBus Master may be advised by the ACK[2:0]* value received from the SBus Slave that its preferred data width cannot be provided, but that Bus Sizing is being used to provide an alternative data width. For Burst Transfers, SBus Slaves shall provide the number of Word Acknowledgments required to complete the Burst Transfer or provide an Error Acknowledgment indicating that the requirement cannot be met. If the SBus Master selects a 64-bit Extended Transfer, the required number of double-word transfers to be performed during the Extended Transfer is specified by the Extended Transfer Count. For 64-bit Extended Transfers, SBus Slaves shall provide the number of Double-word Acknowledgments required to complete the Burst Transfer or provide an Error Acknowledgment indicating that the requirement cannot be met.

The SBus Master uses the ACK[2:0]* signals from the Slave to indicate the transfer of the specified data. Any ACK[2:0]* value not accepted by the SBus Master shall be treated by the SBus Master as if the ACK[2:0]* value were an Error Acknowledgment. An SBus Master shall not request a Burst Transfer or Extended Transfer size that it does not implement.

SBus Slaves use the SIZ[2:0] signals to identify the transfer width that the SBus Master prefers or to identify the required number of word transfers to be performed in a Burst Transfer. For single data unit transfers, the SBus Slave may advise the SBus Master that it cannot support the requested data width by presenting an Error Acknowledgment. For single unit transfers, the SBus Slave may return an ACK[2:0]* value indicating that the data transfer is occurring with a width different from that specified by the SIZ[2:0] signals. This may result in Bus Sizing. For Burst or Extended Transfers, the SBus Slaves shall either provide the number of Word or Double-word Acknowledgments required to perform the transfer or shall provide an Error Acknowledgment indicating that the required transfer cannot be performed. If the SBus Slave has received a request for a 64-bit Extended Transfer, the number of double-word transfers required to complete the Extended Transfer is specified by the Extended Transfer Count.

SBus Controllers use the SIZ[2:0] signals to determine the expected number and type of Data Acknowledgments that will be performed during the Transfer Phase, then use the ACK[2:0]* signals to determine when the Transfer Phase is completed so that the Controller can negate BG* and AS* properly.

Any value of ACK[2:0]* determined by the SBus Controller to be reserved, not implemented, or inappropriate for the requested operation shall be treated by the SBus Controller as if the value were an Error Acknowledgment. In that case, BG* and AS* may be negated before the SBus Master and the SBus Slave expect the SBus cycle to end. Masters and Slaves shall be so designed as to be sensitive to BG* and AS*, respectively, and to end any SBus cycle in progress when these signals are negated, even if such negation occurs earlier than predicted by the normal SBus signalling protocol.

The minimum requirements and additional recommendations for each SBus Device are contained in the following outline.

5.4.3.1 SBus Master

a) All SBus Masters shall implement single data unit transfers, including byte, half-word, and single word transfers according to the following rules:

   1) SBus Masters shall generate at least one of request SIZ[2:0] values: byte, half-word, or word;

   2) SBus Masters shall accept a single data unit transferred with the value of ACK[2:0]* that matches the requested SIZ[2:0] value;

   3) SBus Masters should implement full interoperability. To implement full interoperability, the SBus Master shall meet the following requirements;

      i) any ACK[2:0]* with a smaller than requested size shall be accepted according to the rules for Bus Sizing and appropriate follow-on cycles generated to obtain the required data,

      ii) any ACK[2:0]* with a larger than requested size shall be accepted and the data obtained according to the rules for data organization. The data organization rules require that the SBus Master implement the full 32-bit width of the SBus and provide appropriate byte and half-word routing.

b) SBus Masters should implement Burst Transfers according to the following guidelines to achieve maximum performance and interoperability:

   1) SBus Masters that implement any Burst Transfers shall perform at least one request of SIZ[2:0] value of byte, half-word, or word and shall accept a single data unit with the value of ACK[2:0]* that matches the requested SIZ[2:0] value (see items a) 1) and a) 2) above);

2) SBus Masters that implement any Burst Transfers should meet the requirement for full interoperability for single data unit transfers (see item a) 3) above);

3) SBus Masters should implement one of the following sets of 32-bit wide Burst Transfers;

   – no bursts allowed
   – 4-word burst only (recommended minimum)
   – 2- and 4-word bursts only
   – 2-, 4-, and 8-word bursts only
   – 2-, 4-, 8-, and 16-word bursts;

4) for maximum performance, SBus Masters should execute the largest Burst Transfer size accepted by the SBus Slave that is to execute the data transfer.

c) SBus Masters that implement the optional 64-bit Extended Transfer function should do so according to the following guidelines to achieve maximum performance:

1) SBus Masters that implement any 64-bit Extended Transfer shall perform at least one request of SIZ[2:0] value of byte, half-word, or word and shall accept a single data unit with the value of ACK[2:0]* that matches the requested SIZ[2:0] value (see items a) 1) and a) 2) above);

2) SBus Masters that implement any 64-bit Extended Transfer should meet the requirement for full interoperability for single data unit transfers (see item a) 3) above);

3) SBus Masters should implement one of the following sets of 32-bit wide Burst Transfers and 64-bit Extended Transfers;

   – 1 double-word burst and a 2-word burst
   – 1 and 2 double-word bursts and a 2- and 4-word burst
   – 1-, 2-, and 4-double-word bursts and a 2-, 4-, and 8-word burst
   – 1-, 2-, 4-, and 8-double-word bursts and a 2-, 4-, 8-, and 16-word burst
   – 1-, 2-, 4-, 8-, and 16-double-word bursts and a 2-, 4-, 8-, and 16-word burst;

4) for maximum performance, SBus Masters should execute the largest Burst or Extended Transfer size accepted by the SBus Slave that is to execute the data transfer.

5.4.3.2 SBus Slave

a) All SBus Slaves shall implement single data unit transfers, including byte, half-word, and single word transfers according to the following rules:

1) SBus Slaves shall properly accept at least one of SIZ[2:0] values: byte, half-word, or word;

2) SBus Slaves should implement full interoperability. To implement full interoperability, an SBus Slave shall respond to any single data unit transfer SIZ[2:0] value with an ACK[2:0]* value consistent with the SBus Slave's actual data width. The required data organization and address alignment is specified in 5.4.5. Any SBus Slave meeting this requirement is fully interoperable with any SBus Master that complies with the full interoperability requirements. Note that no more than one of the three single data unit transfers is actually required to be performed with matching SIZ[2:0] and ACK[2:0]* values, since a fully interoperable SBus Master is capable of Bus Sizing or interpreting the data organization correctly for all other cases.

b) SBus Slaves should implement Burst Transfers according to the following guidelines to achieve maximum performance:

1) SBus Slaves that implement any Burst Transfers should meet the requirement for full interoperability for single data unit transfers (see item a) 2) above);

2) for maximum performance and full interoperability with any SBus Master, SBus Slaves should implement one of the following sets of 32-bit wide Burst Transfers: No burst implemented;
   - 4-word burst only (recommended minimum)
   - 2- and 4-word bursts only
   - 2-, 4-, and 8-word bursts only
   - 2-, 4-, 8-, and 16-word bursts.

c) SBus Slaves that implement the optional 64-bit Extended Transfer function should do so according to the following guidelines to achieve maximum performance:
   1) SBus Slaves that implement any 64-bit Extended Transfer should meet the requirement for full interoperability for single data unit transfers (see item a) 2) above);
   2) for maximum performance and full interoperability with any SBus Master, SBus Slaves should implement one of the following sets of 32-bit wide Burst Transfers and 64-bit Extended Transfers
      - 1 double-word burst and a 2-word burst
      - 1- and 2-double-word bursts and a 2- and 4-word burst
      - 1-, 2-, and 4-double-word bursts and a 2-, 4-, and 8-word burst
      - 1-, 2-, 4-, and 8-double-word bursts and a 2-, 4-, 8-, and 16-word burst
      - 1-, 2-, 4-, 8-, and 16-double-word bursts and a 2-, 4-, 8-, and 16-word burst.

5.4.3.3 SBus Controller

a) For single data unit transfers, all SIZ[2:0] values and all ACK[2:0]* values shall be implemented.

b) All Burst Transfer SIZ[2:0] values and the required counting of Word Acknowledgments shall be implemented.

c) If any of the optional 64-bit Extended Transfer functions are supported, all ETI sizes and Double-word Acknowledgment counts shall be implemented as well as all 32-bit Burst Transfers and single data unit transfers.

5.4.4 Burst Transfers

Burst Transfers allow the transfer of multiple words or double-words in a single Transfer Phase. Burst Transfers should be used whenever possible for communication between SBus Devices, since their use increases SBus throughput and decreases SBus protocol overhead.

The virtual address and the corresponding physical address of the lowest byte of a block of data being transferred shall always be block aligned. The number of bytes in the transfer are specified by the value provided by the SIZ[2:0] bits or the Extended Transfer Information Transfer Count field. For example,

– a 2-/4-/8-/16-word burst may be read and written only from a block whose lowest address's 3/4/5/6 least significant bits are 0;

– a 2-/4-/8-/16-double-word burst may be read and written only from a block whose lowest address's 4/5/6/7 least significant bits are 0.

Although the lowest byte address of the block of data being transferred shall be properly aligned, the Burst Transfer may begin with any word or double-word within the block. The virtual address generated by the SBus Master and the resulting physical address from the SBus Controller need have only its two least significant bits be 0 for word transfers or three least significant bits be 0 for double-word transfers. The transfer begins with the word or

double-word specified by the physical address. For each subsequent word or double-word transferred, the address is internally incremented by the SBus Slave and the SBus Master by 4 or 8 up to the block boundary specified by the transfer size information, then wrapped back to the address of the first data unit of the block. This address wrapping shall be implemented by all SBus Slaves performing Burst Transfers. The SBus Master privately manages the internal addressing of the bytes being transferred. The SBus Controller manages the Transfer Phase by counting Acknowledgments as described in 5.4. Figure 19 shows the timing of two asymmetric SBus cycles, a 32-bit write operation followed by a 32-bit Burst Transfer Phase using address wrapping.

Address wrapping during Burst Transfers allows an SBus Master that implements a cache to transfer first the word that caused a cache miss, and then the rest of the words to fill up the cache line. Thus, the processor using information provided from the SBus Master can begin execution immediately without having to wait for the first part of the block to be cached.

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 Slaves shall implement burst transfers in accordance with the requirements of 5.4.3. If any burst capability is supported, the SBus Slave should provide the appropriate burst-size attribute in its FCode program.

5.4.5 Data organization

The organization and addressing structure of the bytes to be transferred on D[31:0] and, for 64-bit Extended Transfers, on D[63:0] is specified in 4.8.

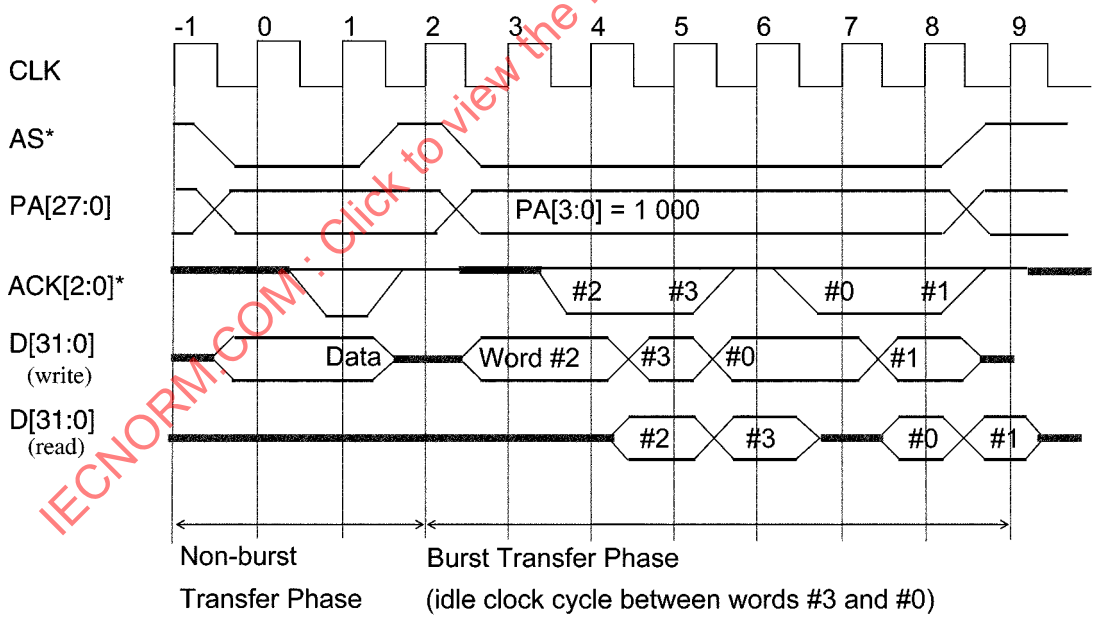


Figure 19 – Burst Transfer Phase, asymmetric cycles, 32-bit transfers, address wrapping

An SBus Master shall implement the full 32 bits of the D[31:0] bus. The SBus Master shall steer the bytes, half-words, or words of data to be transferred on the SBus such that the SBus Slave will always be able to properly transfer the bytes of data in the proper location on the D[31:0] signals. SBus Slaves shall transfer data to and from the SBus with a width of either one byte (D[31:24]), a half-word (D[31:16]) or a word of the data bus. The SBus Slave provides a Data Acknowledgment to indicate from which set of signals it received the data or to which set of signals it presented the data.

Burst Transfers shall use the full 32-bit width of the D[31:0] bus. 64-bit Extended Transfers shall use the full 64-bit width of the D[63:0] bus, utilizing the multiplexed lines to transmit the high-order 32 bits.

When an SBus Master performs a byte write to a specified address, it shall always duplicate the byte of data at all the locations where it may be valid for an SBus Slave to look for the data. The SBus Slave shall always take the data from the location corresponding to the physical address bits and to the type of Data Acknowledgment it elects to return to the SBus Master.

When an SBus Master performs a half-word write to a specified address, it shall duplicate the half-word of data on all half-word locations where it may be valid for an SBus Slave to look for the data. The SBus Slave shall always take the data from the location corresponding to the physical address bits and to the type of Data Acknowledgment it elects to return to the SBus Master.

During Read operations, the SBus Slave shall present data at the location indicated by the low-order physical address bits and the type of Data Acknowledgment the Slave presents. The SBus Master shall interpret the location of the data from the low-order address bits and the type of Data Acknowledgment the SBus Slave presents. The SBus Master shall perform the appropriate mapping to the correct address boundaries.

SBus Masters shall not generate unaligned data transfers, where the PA[27:0] low order bits are not consistent with the specified size of the transfer, unless the data transfer is consistent with the rules of Bus Sizing follow-on transfers as shown in Table 9.

SBus Slaves that expect to operate correctly with Bus Sizing shall always use a single Data Acknowledgment size within a referenced word. The SBus Slave shall use the required data placement for the particular value of SIZ[2:0], Data Acknowledgment and address as shown in Table 13. The SBus Slave does not distinguish between accesses that are follow-on cycles for Bus Sizing and accesses that meet the rules for Bus Sizing, but are generated individually.

An SBus Slave shall generate an Error Acknowledgment if an unaligned data transfer is requested that violates the rules for Bus Sizing follow-on cycles. An example of such a violation is a 16-bit transfer request aligned on odd byte boundaries to an SBus Slave that generates Half-word Acknowledgments in that address space.

Properly aligned data transfers are also allowed during Bus Sizing operations, as described in 5.4.6. Properly aligned data transfers are required for operation within word boundaries that do not perform Bus Sizing.

Figure 10 provides information about the naming and addressing of words, half-words, and bytes.

Table 9 shows data placement requirements as a function of the requested size and the responding Data Acknowledgment.

5.4.6 Bus Sizing

If an SBus Master requests a half-word from a byte Slave or requests a word from a half-word or byte Slave, the SBus Slave may choose to return an Error Acknowledgment. Alternatively, the SBus Slave can return a Data Acknowledgment acknowledging that portion of the transfer that it can perform and implicitly requesting the SBus Master to perform additional follow-on cycles to transfer the remaining bytes. This mechanism of dividing a single SBus cycle into multiple separate SBus cycles performing smaller data transfers is called Bus Sizing. The Bus Sizing mechanism allows an SBus Master to treat the Slave as though it were a word or half-word SBus Device, even though the SBus Slave may implement only half-word or byte transfers.

Bus Sizing shall only be requested by an SBus Slave when the SBus Master requests word or half-word transfers. Bus Sizing shall not be performed during any Burst or Extended Transfers. An SBus Slave unable to support a Burst or Extended Transfer shall issue an Error Acknowledgment if an SBus Master attempts such a transfer. SBus Slaves shall never issue a byte or Half-word Acknowledgment in response to a request for a Burst or Extended Transfer.

Table 9 – Data placement requirements

Requested size SIZ[2:0]	Physical address low bits	SBus Master write data placement	Data Acknowledgment	Data location read by Master read/write by Slave
Extended 64-bit SIZ[2:0] = 011	000	Double-word, D[63:0]	Double-word	64-bit assignments for double-word, D[63:0]
Word SIZ[2:0] = 000	00	Word, D[31:0]	Word	Word, D[31:0]
			Half-word	Half-word 0, D[31:16]
			Byte	Byte 0, D[31:24]
	01	Byte 0, D[31:24]	Byte (follow-on only)	Byte 0, D[31:24]
	10	Half-word 0, D[31:16]	Half-word (follow-on only)	Half-word 0, D[31:16]
		Byte 0, D[31:24]	Byte (follow-on only)	Byte 0, D[31:24]
	11	Byte 0, D[31:24]	Byte (follow-on only)	Byte 0, D[31:24]
Half-word SIZ[2:0] = 010	00	Half-word 0, D[31:16]	Word	Half-word 0, D[31:16]
			Half-word	Half-word 0, D[31:16]
			Byte	Byte 0, D[31:24]
	01	Byte 0, D[31:24]	Byte (follow-on only)	Byte 0, D[31:24]
	10	Half-word 0, 1, D[31:16], and D[15:0]	Word	Half-word 1, D[15:0]
			Half-word	Half-word 0, D[31:16]
			Byte	Byte 0, D[31:24]
	11	Byte 0, D[31:24]	Byte (follow-on only)	Byte 0, D[31:24]
Byte SIZ[2:0] = 001	00	Byte 0, D[31:24]	Word	Byte 0, D[31:24]
			Half-word	Byte 0, D[31:24]
			Byte	Byte 0, D[31:24]
	01	Byte 0, 1, D[31:24] and D[23:16]	Word	Byte 1, D[23:16]
			Half-word	Byte 1, D[23:16]
			Byte	Byte 0, D[31:24]
	10	Byte 0, 2, D[31:24] and D[15:08]	Word	Byte 2, D[15:08]
			Half-word	Byte 0, D[31:24]
			Byte	Byte 0, D[31:24]
	11	Byte 0, 1, 3, D[31:24] and D[23:16] and D[07:00]	Word	Byte 3, D[07:00]
			Half-word	Byte 1, D[23:16]
			Byte	Byte 0, D[31:24]

Support for Bus Sizing principally affects the design of the SBus Master. SBus Masters are not required to support Bus Sizing. If an SBus Master does not support Bus Sizing, it should not initiate a transfer that might require an SBus Slave to request Bus Sizing. If the properties of the addressed SBus Slave are not known, an SBus Master that does not support Bus Sizing should use the unexpected Data Acknowledgment size returned by the Slave as an indication that it may only be able to communicate with that Slave using the value of SIZ[2:0] corresponding to the unexpected Data Acknowledgment. See 5.4.3.

During Bus Sizing, each byte or half-word must be transferred using an independent SBus cycle. The first unit of data (the one that invoked Bus Sizing) shall always be transferred as part of the original bus cycle. Follow-on cycles use an independent SBus cycle to transfer each additional unit of data. An SBus Slave must treat every cycle individually, using no retained state information about whether previous Bus Sizing cycles have occurred. The SBus Master must generate the correct address for the unit of data being transferred during each cycle. SBus Masters shall change only the two least significant address bits in follow-on bus cycles.

An SBus Slave shall respond with a single type of Data Acknowledgment for each SBus cycle within a referenced word as explained in 5.4.5. As an example, an SBus Slave that responds with a Byte Acknowledgment for the first byte of the transfer must respond with a Byte Acknowledgment for each of the remaining transfers accessing data in the same word. An SBus Slave requires no special hardware to take advantage of Bus Sizing.

The data path location is determined by the SBus Slave's Data Acknowledgment. The type of Data Acknowledgment returned by an SBus Slave shall not depend on the transfer width requested by the SBus Master. The Data Acknowledgment shall correctly define the SBus Slave's own data path width. The SBus Master correctly places or fetches the first element of data to be transferred according to the rules of data organization. The SBus Master shall place or sense the elements of data in the proper data locations for each subsequent follow-on cycle according to the same rules.

An SBus Master may stop performing Bus Sizing follow-on cycles after any cycle and continue with normal operation, including the execution of another transfer which requires Bus Sizing. If the SBus Slave issues a Retry Acknowledgment, the SBus Master shall retry the current SBus cycle using the normal retry procedures, regardless of whether or not the current cycle is an original or a follow-on cycle. An Error Acknowledgment may be issued if required during any SBus cycle of a Bus Sizing operation.

During the follow-on bus cycles, the Master may keep the SIZ[2:0] signal set at the original size. This results in follow-on cycles that appear to be unaligned transfers. Since the Data Acknowledgment returned by the Slave must be the same for each data element within the word, these cycles are completed with the same Acknowledgment used by the Slave for the original transfer.

SBus bridges operating as SBus Slaves shall never initiate Bus Sizing. For complete transparency, SBus bridges should support all transfer sizes and acknowledgments. SBus Masters shall not unconditionally expect Bus Sizing to be performed, even from SBus Slaves known to require Bus Sizing, since an intervening bus bridge may hide the Bus Sizing performed by an SBus Slave from the SBus Master.

SBus Masters serving the CPU of SBus Systems should support Bus Sizing for both byte and half-word SBus Slaves to maintain full compatibility with common operating systems. Any SBus Master is allowed to support Bus Sizing. Each SBus Master's implementation of Bus Sizing is independent of all other SBus Masters support.

SBus Slaves can use Bus Sizing to reduce software complexity in some cases. For example, an 8-bit frame-buffer that is otherwise functionally identical to a 32-bit frame-buffer can use the software intended for the 32-bit frame-buffer without modification.

5.4.7 Parity checking

An SBus System has the option of implementing parity checking on data and virtual address transfers if parity generation and checking is implemented on the SBus Controller and the installed SBus Masters and SBus Slaves. The SBus System determines that the necessary SBus Devices support parity checking by examining their FCode program for the appropriate parity attribute to be present. Parity checking provides increased confidence in the integrity of those transfers that have been checked.

When the SBus is reset, parity checking shall be disabled. Parity checking is enabled under FCode or software control. Parity checking may be enabled between any SBus Slave and SBus Master pair that supports parity checking. If an SBus Slave is also accessed by an SBus Master that does not generate parity, parity checking should only be enabled if there is an additional mechanism available to selectively activate parity checking for each SBus cycle. SBus Slaves that use different address spaces for different SBus Masters and SBus Slaves integrated with SBus Controllers have the capability of supporting such combinations of SBus Masters. If parity checking is enabled, the SBus Controller may optionally be enabled to check the parity of the virtual address transmitted by the SBus Master. SBus Systems may choose to enable parity checking only if all installed SBus Devices support parity checking. Odd parity over the 32 or 64 relevant data bits and the DP bit should always be presented by SBus Devices that support parity checking, even if parity checking is disabled.

When parity checking is enabled in the SBus Controller, the Controller shall verify that it receives correct odd parity on the virtual address, bits D[31:0] and DP, during the Translation Phase. It shall verify that it receives correct odd parity on the Extended Transfer Information, bits D[31:0] and DP, during the Extended Transfer Information Phase, if any. The SBus Controller shall issue an Error Acknowledgment to the Master and end the SBus cycle if incorrect parity is detected. See 5.3 for information on the error presentation timing during the Extended Transfer Information Phase.

When parity checking is enabled by an SBus Device that is receiving data during a Transfer Phase, the Device receiving the information shall verify that correct odd parity on D[31:0] and DP is received each time the valid data is clocked into the SBus Device. For a read, the receiving SBus Device is the Master, while for a write, the receiving Device is the Slave.

When parity checking is enabled by an SBus Device that is receiving data during a 64-bit Extended Transfer Phase, the Device receiving the information shall verify that correct odd parity on D[63:0] and DP is received each time the valid data is clocked into the SBus Device.

The occurrence of a parity error during a Transfer Phase or a 64-bit Extended Transfer Phase is indicated in the following manner. If an SBus Slave detects the parity error, it shall notify the SBus Master and SBus Controller by presenting the LERR* signal with the proper timing. If Acknowledgments are delayed long enough to allow checking of received data, the SBus Slave may present an Error Acknowledgment instead of the Data Acknowledgment/LERR* sequence. If it is an SBus Master that detects the parity error, it should generate an interrupt or other error indication to the SBus System's CPU. The recovery and reporting mechanisms are not defined by this standard.

During any clock cycle that the signals D[31:0] remain released and in their high impedance state, the DP signal shall also remain released. DP is driven with the same timings and by the same SBus Device that generates the D[31:0] or D[63:0] signals. If this standard does not specify what the value of the bits are for a particular data location, then the unspecified data signals shall still be driven to a valid logic level and parity shall be generated and checked over whatever value is presented to the D[31:0] signals. SBus Slaves that connect to only a subset of D[31:0] (i.e. byte or half-word Slaves) shall not generate or check parity.

5.4.8 Retry Acknowledgment

5.4.8.1 Allowable uses for Retry Acknowledgment

A Retry Acknowledgment may be transmitted by an SBus Slave during the Transfer Phase when resources are temporarily unavailable to perform the requested operation. If the resources are permanently unavailable, do not exist, or are not operating correctly, an Error Acknowledgment shall be asserted by an SBus Slave instead of a Retry Acknowledgment.

The standard explicitly describes the following conditions under which Retry Acknowledgment is appropriate. While other cases may also demand Retry Acknowledgment, the unpredictable effects on performance and the possibility of causing system hangs or deadlock conditions requires great care in the design of SBus Slaves that make use of Retry Acknowledgment for those cases not explicitly described.

a) Unavailability of the required resources may force an SBus Slave to transmit Retry Acknowledgment. In this case, the SBus Slave does not retain any state information. The SBus Slave will continue to transmit Retry Acknowledgments to all requests for the unavailable resources until the resources become available and the appropriate Data Acknowledgment indicates that the requested transfer is performed. If the SBus Slave determines that the resource is not ever going to become available or that the resource is not operational, the SBus Slave shall return Error Acknowledgment.

b) Long latency in accessing a resource may make it more efficient for an SBus Slave to perform a Retry Acknowledgment as a mechanism for performing a disconnected SBus cycle. In this case, the SBus Slave retains state information and initiates the access requested by the retried SBus cycle so that when the request is performed again by the SBus Master, the data will be available to complete the request. The SBus Slave shall provide the appropriate Data Acknowledgment to the SBus Master when the SBus Master repeats the same request after the data has been accessed. The SBus Slave shall fulfill the request with the proper data after verifying that the proper physical address, direction of transfer, and size are specified in the repeated request. Note that the data will be valid regardless of whether the access that actually obtains the data is from the SBus Master that first received the Retry Acknowledgment or another SBus Master that just happens to perform the same operation at the time the data has been made available. SBus Slaves should be designed to minimize the frequency of occurrence of long latency accesses. Some SBus Slaves may be able to make predictions about the access pattern to long latency data such that the data can be prefetched after the first access of a string of sequential accesses has been identified, eliminating the requirement for subsequent disconnected SBus cycles. Particular care should be taken in the design of hardware and software so that critical control data is not inadvertently made invalid by such prefetch operations.

c) Bus Locking operations may force an SBus Slave to transmit Retry Acknowledgment. If the requested resource is temporarily unavailable because the resource is shared and has been locked, the SBus Slave will transmit Retry Acknowledgments to all requests that would violate the lock requirements. No state information is retained by the SBus Slave. The Bus Locking protocol is described in 5.3.4 and 5.7.

d) An SBus Controller may issue a Retry Acknowledgment during the Translation Phase as described in 5.2.

5.4.8.2 Bus Slave presentation of Retry Acknowledgment

An SBus Slave may issue a Retry Acknowledgment instead of an Error or Data Acknowledgment as the first and only Acknowledgment in a Transfer Phase. Once a Data Acknowledgment has been presented in a Burst Transfer, the SBus Slave shall not present a Retry Acknowledgment. A Retry Acknowledgment shall be presented within the timeout number of clock cycles following the assertion by the SBus Controller of AS*. The Retry Acknowledgment should be offered as soon as possible by the SBus Slave to avoid undesirable bus overhead.

An SBus Slave may issue a Retry Acknowledgment on any SBus cycle, including a Bus Sizing follow-on cycle.

An SBus Slave is permitted to continue issuing Retry Acknowledgments until it is able to complete the transfer. The SBus Controller and the SBus Masters shall not implement a mechanism for limiting the number of Retry Acknowledgments a Slave may issue. SBus Slaves shall implement a mechanism for terminating indefinitely repeated Retry Acknowledgments by generating an Error Acknowledgment.

An SBus Slave shall not use stateless retry to control the order of accesses of various Masters. The SBus Controller has complete control over which SBus Master is next granted access to the Slave.

If the SBus Slave presents a Retry Acknowledgment to perform a disconnected SBus cycle, it shall check the physical address, direction, and size of the subsequent requests to verify that the proper data is being provided to the SBus Master. The SBus Slave is allowed to accept as many disconnected SBus cycle operations as it has resources to support such operations. If more disconnected operations are attempted than the SBus Slave supports, subsequent attempts to perform an operation will also be acknowledged with Retry Acknowledgments, but without capturing state information and without performing the requested access.

To prevent starvation in cases where multiple SBus Masters are competing for an SBus Slave's resources, when an SBus Slave completes the data transfer associated with the disconnected cycle and presents the appropriate Data Acknowledgment, the SBus Slave shall be able to establish a new disconnected cycle operation in the next SBus cycle. An SBus Slave shall not have a temporary busy period that causes the presentation of stateless Retry Acknowledgment after the successful completion of a data transfer.

The SBus Slave shall manage any coherency problems associated with read and write accesses to the same physical address. Such problems may require the support of software in the drivers managing the SBus Slave.

The control registers of the SBus Slave port of an SBus Master shall be accessible to another SBus Master at all times. Retry Acknowledgments shall not be performed because the SBus Slave port is blocked by SBus Master port activities, but may be performed if the SBus Slave port is only temporarily unavailable for other reasons.

5.4.8.3 SBus Controller presentation and Retry Acknowledgment management

The SBus Controller shall follow its normal arbitration algorithm after receiving a Retry Acknowledgment.

After receiving or generating a Retry Acknowledgment, an SBus Controller shall terminate the SBus cycle. The SBus Master is forced to arbitrate in the normal manner to attempt another execution of the SBus cycle that received a Retry Acknowledgment.

SBus Master processing of Retry Acknowledgment. An SBus Master processes a Retry Acknowledgment from an SBus Slave by repeating the same request to the SBus Slave until the SBus Slave responds with Error Acknowledgment or the appropriate Data Acknowledgment. The repeated request shall be identical in address, size, and direction to the original request that received the Retry Acknowledgment. The SBus Master enters the normal arbitration cycle to gain control of the bus for each attempt to repeat the original request. The SBus Master shall repeat the attempt to perform the request until one of the following events terminates its efforts.

a) The SBus Slave presents an Error Acknowledgment.

b) The SBus Slave presents the appropriate Data Acknowledgment.

c) The SBus Master is reset by assertion of the RST* signal.

d) The SBus Master is ordered through its SBus Slave port to terminate the attempts to perform the request. This is performed by setting an SBus Master reset control bit in a software accessible control register of the SBus Slave port. Upon receiving such a reset, the SBus Master shall not negate BR* until BG* is asserted and the requested SBus cycle is completed or terminated by a Retry Acknowledgment.

   The use of this software-managed termination of indefinitely repeating retries may leave the SBus Slave in a state such that Retry Acknowledgments will be given to all SBus Masters. The SBus Slave may also require software-generated reset operations to resume normal operation.

An SBus Master shall repeat only the single request that has received Retry Acknowledgment until the repeated operation is completed, except as described below.

Some SBus Masters may be capable of generating and managing more than one outstanding request. As an example, if an SBus cycle to a particular SBus Slave is terminated with a Retry Acknowledgment, the SBus Master may still be able to execute other requests to the same or other SBus Slaves. The SBus Master shall periodically repeat each request that was terminated with a Retry Acknowledgment to guarantee that it completes each request in a timely manner. This capability of managing multiple outstanding requests is an optional function that requires proper support from the SBus Master, SBus Slave, and system software to prevent abnormal hang conditions and deadlock conditions. Each SBus Master capable of managing multiple outstanding requests shall not enable that capability until specifically requested to do so through its SBus Slave port. After power on and after receiving a RST* signal, the SBus Master shall return to the default state, which requires that the SBus Master repeat only the single request that has received Retry Acknowledgment.

If the SBus Master has been forced to repeat an operation for a period longer than expected, it may conclude that the operation will not be possible to complete. The SBus Master may generate an interrupt to the SBus Controller. SBus System software can then interrogate the SBus Master through its SBus Slave port to determine the reason for the interrupt. The SBus Master shall continue to repeat the operation receiving the Retry Acknowledgment until specifically instructed by one of the mechanisms described above. Such recovery may disrupt SBus System operation temporarily. Only a hardware or software failure should cause such a recovery to be required.

The SBus does not support an SBus Master identifier. If the functions performed by an SBus Slave require it to know from which SBus Master a request is coming, software conventions can assign multiple address spaces within the SBus Slave with a separate address space assigned for the use of each SBus Master.

5.4.9 Error Acknowledgment during Transfer Phase

SBus Slaves may issue an Error Acknowledgment during any Transfer Phase. The Error Acknowledgment shall be issued within the timeout period. An Error Acknowledgment terminates the transfer. The SBus Controller shall terminate the SBus cycle by negating AS* and BG* with the correct timing after receiving an Error Acknowledgment from an SBus Slave. The SBus Master receives and manages the Error Acknowledgment. The Master's recovery procedure is not specified by this standard.

An Error Acknowledgment may be issued instead of Data Acknowledgment at any point in a Burst Transfer. The Error Acknowledgment may be issued before any data has been transferred or after one or more words of data have been transferred. The Error Acknowledgment requests that the SBus Controller terminate the SBus cycle immediately. The SBus Master and SBus Slave shall not expect or send any additional data or Data Acknowledgments after an Error Acknowledgment has terminated an SBus cycle.

Error Acknowledgment should not be used for flow control. In many high-performance systems, caching and buffering make it impossible to signal such errors in a synchronous fashion. Alternative techniques, such as the use of high and low watermarks on FIFOs, interrupts, and the use of SBus Master capabilities should be considered.

The error mechanism on the SBus is designed on the assumption that the errors for which Error Acknowledgment will be used occur rarely. Properly initialized SBus Masters should not invoke operations that will always cause an SBus Slave or Controller to present an Error Acknowledgment. Individual SBus Masters and Slaves are responsible for collecting whatever state information is required to define the error condition and perform the proper recovery actions. Information that the SBus Slave has collected about the condition causing the Error Acknowledgment should be available to the SBus Master from status registers in the Slave.

In some systems, the SBus Controller generates an interrupt to the CPU if the SBus Controller detects an Error Acknowledgment. The SBus Controller may or may not choose to keep a copy of the original virtual address as an aid to error tracking.

5.5 Dual function SBus Devices

If an SBus Device has both SBus Master and SBus Slave capabilities, the SBus Device shall allow the circuitry associated with the SBus Slave function (the Slave port) to be accessed regardless of whether the SBus Master is enabled and operating. This allows the SBus System the option of enabling and disabling the SBus Master with software if the Device allows the SBus Master to be disabled. This additionally prevents two SBus Masters that are accessing each other's Slave ports from being forced into an indefinite retry period because both Slave ports are unavailable while each respective Master is attempting to retry a transfer.

A dual function SBus Device's SBus Master can address the same Device's SBus Slave function. In this case, normal operation shall occur. All the normal SBus protocols shall appear on the SBus signal lines, since the SBus Controller must have access to them to perform its operations correctly.

5.6 Exception conditions

5.6.1 Late Error

Errors that cannot be presented exactly at the time that data is transferred may be presented through the Late Error mechanism. The LERR* signal indicates that such an error has occurred. The LERR* signal shall be generated and interpreted as described in 4.12.

Although LERR* may follow a Retry or Error Acknowledgment, this usage of LERR* is discouraged. In such cases, the standard responses to the Retry or Error Acknowledgment shall be executed by the SBus Master and Controller and the LERR* signal shall be ignored.

5.6.2 Timeouts

The timeout number of clock cycles shall be 255.

An SBus Master in a properly functioning SBus System is guaranteed to receive an Acknowledgment other than Idle Acknowledgment from the selected SBus Slave within 255 clock cycles of beginning the Transfer Phase. When the AS* signal is asserted, the SBus Controller begins a count of at least 255 clock cycles. If the selected SBus Slave has not presented an appropriate Acknowledgment by the end of the timeout count because the Slave is not installed or not operational, the SBus Controller shall assert an Error Acknowledgment to indicate the timeout condition. The SBus Controller asserts the Error Acknowledgment code for one cycle, then asserts the Idle Acknowledgment for one cycle, then releases ACK[2:0]*. The SBus Controller negates AS* and BG* as early as the cycle after presenting the Error Acknowledgment, as it would for an Error Acknowledgment presented by the SBus Slave.

Figure 20 shows bus timeouts.

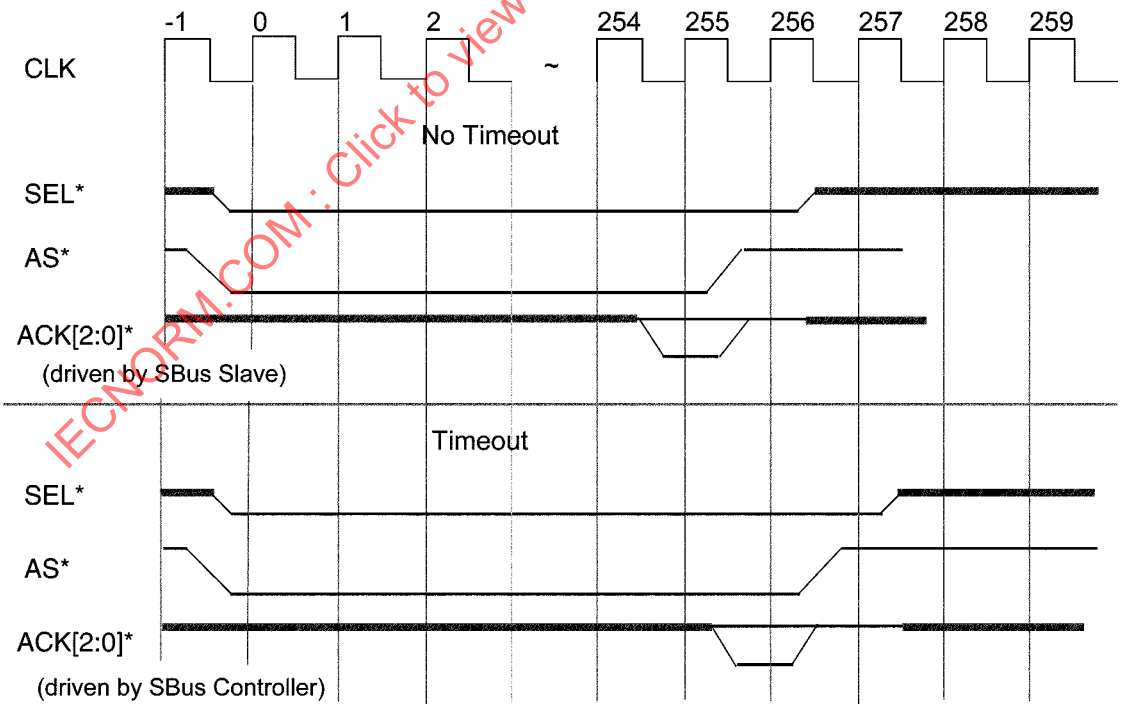


Figure 20 – Bus timeouts

An SBus Slave shall issue an Error Acknowledgment if an SBus Master attempts to perform an operation to the SBus Slave that the SBus Slave cannot execute.

The SBus Slave shall not depend on the timeout function of the SBus Controller to present such Error Acknowledgments because

a) the error isolation information provided by the SBus Slave is important to the error recovery process,

b) proper behavior of each SBus Slave is required for maximum system performance.

An SBus Slave temporarily unable to respond within the timeout period, but which does not wish to abort the transfer, shall use a Retry Acknowledgment.

Once an SBus Slave has not responded with a valid Acknowledgment within the timeout period, it shall not respond to the current bus cycle, since such a response could conflict with the timeout Error Acknowledgment presented by the SBus Controller.

The standard does not specify the way in which the occurrence of a timeout will be presented to the SBus System's CPU by the SBus Master and SBus Controller.

The standard specifies only the SBus Controller managed timeout procedure. Note that hardware failures may occur that make it impossible for the SBus Controller to perform its timeout. Such failures are detected by the SBus System and reported or recovered using software and hardware mechanisms not specified by this standard.

5.6.3 Interrupts

Interrupts are presented to the SBus System from SBus Masters and SBus Slaves through the INT[7:1]* signals as specified in 4.13. The assignment, masking, processing, and presentation of interrupts to the SBus System's CPU is not specified by this standard.

5.7 Extended Transfer locking protocol

5.7.1 Description of Bus Locking

Some SBus Slaves have resources that are dual or multi-ported. Such resources may include memory, registers, and control bits. Both the local SBus and some remote processor or bus may be able to perform operations to those resources. For some purposes, those resources may have to be dedicated exclusively to a particular Master on the local SBus or to the remote bus for a short time to update semaphores or shared information. The SBus Locking function provides the mechanism for reserving and dedicating those resources. The SBus Locking function shall only be performed and managed by Extended Transfer SBus cycles, as described in 5.3.4.

The Bus Locking function requires one Lock flag for each bus for each independently lockable resource. The Lock flag shall be implemented such that only one Lock flag can be set at any time for each independently lockable resource. This requires the implementation of a tie-breaking arbitration mechanism to prevent the setting of more than one Lock flag when simultaneous requests from different bus ports are made for the same lockable resource. The same SBus Slave may also have resources that are not controlled by locking.

When the Lock flag is set for a bus, that bus has exclusive access to the locked resource and all other buses will be prohibited from access to the resource by whatever mechanism is appropriate to the bus architecture. In the SBus architecture, an SBus Slave shall prohibit access to the resource by performing a Retry Acknowledgment to requests against the resource that are not consistent with the locking protocol. The SBus architecture not only provides for locking the resource to the particular SBus, but also locks the resource to a particular SBus Master.

The SBus manages the Lock flag using the Extended Transfer Information Lock field.

5.7.2 SBus Bus Locking protocol

Bus Locking shall only be performed using SBus Extended Transfers. The Extended Lock field is used to set and reset the Lock flag and to manage the locking protocol. The following four Extended Lock values are defined in Table 11:

– Normal
– Initiate Lock
– Maintain Lock
– End Lock

An SBus Master accessing unlocked or unlockable resources shall use either 32-bit transfers or the Normal value of the Extended Lock field. When the SBus Master chooses to lock a resource for its exclusive use, the Master sets the Lock flag by setting the Extended Lock field to the Initiate Lock value for the first transfer to be performed within the set of locked transfers. If the setting of the Lock flag is successful, that transfer will be performed and the SBus Slave shall prohibit all access by other ports. The SBus Slave shall additionally reject any SBus cycles other than Extended Transfers having either a Maintain Lock or an End Lock value in their Extended Lock fields by responding with a Retry Acknowledgment. This convention prohibits any 32-bit access, or any Extended Transfer with the Extended Lock values of Normal access, or any new Initiate Lock access from being performed, limiting access to only the SBus Master that performed the successful Initiate Lock SBus cycle.

If the setting of the Lock flag is unsuccessful, the SBus Slave shall perform a Retry Acknowledgment. The SBus Master shall respond to the Retry Acknowledgment by continuing to attempt the uncompleted transfer in the normal way. SBus Masters that have not successfully completed an Extended Transfer with the Initiate Lock value set in the Extended Lock field shall not perform any operations with the Extended Lock values of Maintain Lock or End Lock to the addressed resource. Such accesses violate the protocol and generate operations that the locked SBus Slave does not successfully recognize and reject. All other rules with respect to Retry are as specified in 5.4.8.

The SBus Master that has successfully set the Lock flag with an Initiate Lock value in its Extended Lock field may continue operating against the locked port using Extended Transfers with the Extended Lock value of Maintain Lock. The SBus Master may perform zero or more operations with the value of Maintain Lock.

When the SBus Master transmits the last SBus cycle of a series of locked transfers, it indicates to the SBus Slave that the Lock flag shall be reset by transmitting the last Extended Transfer with the Extended Lock field set to the value of End Lock. When the requested SBus operation has been successfully completed by the SBus Slave, the Slave shall set the Lock flag back to the reset value, allowing other ports to contend for the resource again.

If an SBus Master attempts to execute an Extended Transfer with an Extended Lock value other than Normal to an SBus Slave resource that does not support locking, the SBus Slave shall respond with an Error Acknowledgment. If an SBus Master attempts to execute an Extended Transfer with an Extended Lock value of Maintain Lock or End Lock to an SBus Slave resource that has not been locked to the SBus port, the SBus Slave shall respond with an Error Acknowledgment.

SBus Slaves should provide those registers that are needed to perform error recovery operations, including sensing and resetting the locking bits, outside the address space used by shared resources so that special recovery operations will not be prohibited by the locking protocol.

6 SBus electrical requirements

6.1 Power

SBus Cards and SBus Systems shall comply with the following power supply requirements per SBus Slot. Double-width SBus Cards may consume the total amount of power specified for both SBus Slots. Current should be drawn approximately equally through all pins.

The average currents (I_{AVG}) specified for the +5 V, +12 V, and 12 V supplies include any transient or peak currents (I_{PEAK}). SBus Cards drawing transient currents greater than the average shall draw a quiescent current low enough to make the average current over any 500 ms period comply with the value given for I_{AVG} in Table 10. The duty cycle for peak currents is specified by the requirements on the average current.

For each SBus Slot Connector, there are 5 pins for +5 V power, 1 pin for +12 V power, 1 pin for 12 V power, and 7 pins for signal and power ground. An SBus Card shall not draw more than the specified current from each power supply for each SBus Slot. The SBus System shall guarantee that the power supplies, inclusive of noise and ripple, remain within the specified voltage tolerance. Adjacent SBus Slots that allow the connection of double-width SBus Cards shall share the same power supplies and ground return paths.

SBus Cards should provide sufficient power supply decoupling to prevent the propagation of noise to the SBus System.

The SBus Card's power usage is standardized to allow operation in all SBus Systems that are compliant with this standard.

6.1.1 SBus Card power

SBus Cards shall dissipate no more than 2 W on the bottom or solder side of the card into the airspace of the motherboard. Double-width SBus Cards shall comply with this requirement at each SBus Slot location to prevent local overheating on the motherboard. The power dissipation should be distributed along the length of the card across the airflow to avoid local overheating. The SBus Card is allowed to dissipate all the power that can be supplied to it within the specified current limits, approximately 11.3 W for each SBus Slot.

Table 10 shows SBus power limits.

Table 10 – SBus Power Parameters

Parameter	Condition	Symbol	Min.	Max.	Unit
+5 V supply	I = 2 A	+5 V	4.75	5.25 V	V
+12 V supply	I = 30 mA	+12 V	11.25	12.75	V
–12 V supply	I = –30 mA	–12 V	–12.75	–11.25	V
Average current 5 V	Over entire specified range of each supply voltage	I_{AVG5}		2.0*	A
Average current +12 V		I_{AVG+12}		0.03*	A
Average current –12 V		$I_{AVG–12}$		–0.03*	A
Peak current 5 V	Duty cycle of peak current such that I_{AVG} is not being exceeded for each supply	I_{PEAK5}		3.0	A
Peak current +12 V		$I_{PEAK+12}$		0.05	A
Peak current –12 V		$I_{PEAK–12}$		–0.05	A
* I_{AVG} averaged over any 500 ms interval.					

6.2 Electronic characteristics

The electronic level, the setup time, the hold time, and the delay requirements of the SBus are selected to be compatible with many CMOS logic families and CMOS gate arrays. It is the responsibility of the SBus System to properly implement and drive the SBus with the characteristics required by the SBus Cards. Circuit modelling and lab measurements may be required to determine the characteristics of a particular CMOS family and card layout.

An individual connector of a double-width SBus Card has the same electronic and loading requirements as a connector of a single-width SBus Card. A double-width SBus Card may choose to make no signal connection to one of its two SBus Connectors. If a particular SBus Connector of a double-width SBus Card drives and receives any SBus signals, it shall drive and receive the complete set of associated SBus signals independent of the other SBus Connector. There is no guarantee that the two SBus Slots in which the double-width SBus Card is installed have the same SBus Controller or the same clock frequency.

The various SBus signals have slightly different drive, capacitive, and bias circuit requirements. When the requirements for a particular signal are special, they will be specifically pointed out in the following sections.

6.2.1 Capacitive loading requirements

Each SBus Card shall contribute no more than 20 pF capacitive loading measured at the SBus Card Slot on each signal except SEL*, BR*, and BG*. This requirement includes the capacitive effects of any connectors and printed circuit board traces associated with the SBus Card.

SEL*, BR*, and BG* shall contribute no more than 60 pF per signal on the SBus Slot. This requirement includes the capacitive effects of any connectors and printed circuit board traces associated with the SBus Card.

The CLK signal on an SBus Card shall contribute between 12 pF and 20 pF of capacitive loading. The narrow range of loading is required to assure proper relative timing of CLK on all SBus Cards and SBus Devices. The total loading of CLK is not specified. CLK shall be driven by the SBus System such that the timing requirements are met over the entire range of capacitive loading specified for an SBus Card, even when some SBus Cards are not installed. To meet the very precise skew requirements on the CLK, the CLK should be driven individually to each SBus Slot with low skew clock driver circuits. The CLK signal line delays and loadings should be matched.

The total capacitance on any signal of the SBus shall not exceed 160 pF if the SBus System is operating at 20 MHz or less, or 100 pF if the SBus System is operating between 20 MHz and 25 MHz. The total capacitance figure shall include all SBus Devices on the motherboard, all wiring, via, connector and pin-related capacitance on the motherboard, and a 20 pF capacitance budget for each SBus Card that can be installed. The total capacitance on individually driven signals like SEL* and CLK is not specified. The SBus System shall provide the correct timings using the SBus Card signal capacitance budgets specified for those signals.

To reach these low-capacitive loading requirements, SBus Devices or SBus Cards should not connect more than a single integrated circuit I/O pin to any SBus signal. SBus Devices requiring more than one I/O pin should buffer the signal.

Table 11 shows SBus capacitive loading limits.

Table 11 – Capacitive loading

Parameter	Condition	Min.	Max.
Loading per SBus Card for signals other than SEL*, BR*, and BG*		0 pF	20 pF
Loading per SBus Card for SEL*, BR*, and BG*		0 pF	60 pF
Loading per SBus Card for CLK		12 pF	20 pF
Total loading per signal for signals other than SEL*, BR*, CLK, and BG*	$F_{CLK} \leq 20$ MHz		160 pF
Total loading per signal for signals other than SEL*, BR*, CLK, and BG*	20 MHz $< F_{CLK} \leq 25$ MHz		100 pF

6.2.2 Printed wire length requirements

Traces branching off the main line routing of an SBus signal and SBus signal traces on an SBus Card are stubs off the main line. Stubs on all SBus signals except INT[7:1]* shall not exceed 65 mm and should be as short as possible. Stubs on INT[7:1]* shall not exceed 75 mm and should be as short as possible.

The maximum length of printed circuit trace on the SBus System should be such that the unloaded signal propagation delay between any two SBus Devices be less than 2.5 ns.

6.2.3 Signal bias circuits

The SBus is intended for operation on short buses contained on a single SBus motherboard. As a result, rise times are usually long compared with propagation delays. In such environments, transmission line characteristics are typically of secondary importance and are not considered by the standard except as specifically noted.

As long as SBus current drive requirements are not exceeded, SBus motherboards may have pull-ups, pulldowns, or other bias circuits as required. The primary purpose of the bias circuits is to hold the inputs to receivers of the signals in a valid logic level to minimize switching currents that may be drawn at states outside the range of valid logic levels. Feedback bias circuits, commonly known as holding amplifiers, may be used for this purpose for many of the signals as long as the circuit does not present more load to the SBus signals than the corresponding resistive bias circuits would present. The bias circuits recommended in the following chart are selected to avoid excess power dissipation. The bias circuit function is provided as a part of the SBus Controller services to the SBus.

Receiver circuit designs shall not fail if the SBus signals are left at a value outside the range of valid logic levels but within the operational voltage range. Possible failure modes that should be avoided in the designs include increases in input leakage, increases in normal current consumption, damaging internal current flows, invalid internal logical states, and circuit oscillation.

Table 12 shows SBus signal bias circuit and drive requirements.

Table 12 – Bias circuit and drive requirements

Signal name	Drive negated before released	Resistor/holding amp	Bias circuit
PA[27:0]	NA	NA	32-bit: none
	NA	HA ok	64-bit 10 kΩ pull-up (2 kΩ pull-down, unused high-order bits)
SEL*	NA	NA	None
D[31:0]	NA	HA ok	10 kΩ pull-up
DP	NA	HA ok	10 kΩ pull-up
SIZ[1:0]	NA	Resistor best	10 kΩ pull-up
SIZ[2]	NA	Resistor best	32-bit 10 kΩ pull-up
	NA	Resistor best	64-bit 1 kΩ to 2 kΩ pull-down
RD	NA	Resistor best	32-bit 10 kΩ pull-up
	Negated	Resistor best	64-bit 1 kΩ to 2 kΩ pull-down
CLK	NA	NA	None
AS*	NA	NA	None
ACK[2:0]*	Negated	No HA	10 kΩ pull-up
ERR*	Negated	No HA	10 kΩ pull-up
BR*	NA	No HA	10 kΩ pull-up on each SBus Slot
BG*	NA	NA	None
RST*	NA	NA	None
INT[7:1]*	Overlap drive ok	No HA	2 kΩ pull-up

The second column of the chart above indicates which SBus signals shall be driven to their negated state before the driving SBus Device disables its driver. The SBus protocol protects the SBus Devices from overlapping the driving of signals with other SBus Devices except for the INT[7:1]* signals, as indicated above. In any case, when an SBus signal is released, it should first have been driven to a valid logic level.

The third column recommends the proper bias circuit technology. The use of resistor bias circuits (pull-ups or pull-downs) is always permissible except for those signals indicated as NA, implying that the signals are always driven by the SBus Controller or SBus Master and therefore do not require any bias circuit. Holding amplifiers may be used on those signals where HA ok is indicated. Holding amplifiers should be avoided where Resistor best is indicated. Resistors and not holding amplifiers shall be used where No HA is indicated.

The fourth column indicates the proper bias circuit network and polarity. Note that the bias circuit type of some signals varies depending on whether the SBus Controller supports Extended Transfers. Pull-down resistors shall be in the range of 1 kΩ – 2 kΩ. The value shall be low enough to maintain the specified low level in the presence of the worst-case specified leakage.

6.2.4 DC parameters

Each SBus Device on the motherboard shall meet the following loading and driving requirements. The SBus Controller drivers and receivers shall meet the following loading and driving requirements as well, but additional d.c. loads are provided by the bias circuits or holding amplifiers functionally associated with the SBus Controller. Each SBus Card shall meet the following loading and driving requirements with respect to its SBus connector. The a.c. requirements specified for SBus Devices may require the use of drivers with d.c. current capability greater than the specified minimum.

SBus signals are similar in voltage level to TTL signals, but require the low-leakage characteristics of CMOS circuitry. The parameters are compatible with a number of CMOS gate array families as well as other standard families of components.

The SBus receivers and drivers should have appropriate clamping diodes to protect the circuits from electrostatic discharge and from ringing.

Standard TTL circuit families and standard CMOS circuit families do not meet the requirements of the SBus. The detailed requirements are specified in the following table. The driver and receiver requirements for all the lines are identical except the drive requirements for the INT[7:1]* signals, which have a special driver requirement.

Table 13 outlines SBus d.c. electrical limits.

Table 13 – DC parameters

Parameter	Condition	Symbol	Min.	Max.	Unit
Input low voltage		V_{IL}	–0.5	0.8	V
Input high voltage		V_{IH}	2.0	V_{CC} + 0.5	V
Input leakage voltage	$-0.5 \le V_{IN} \le V_{CC} + 0.5$ V	I_L	–30	30	µA
For tri-state circuits: (all signals except INT[7:1]*)					
Output low voltage	I_{OL} = 4.0 mA	V_{OL}	–0.4	+0.4	V
Output high voltage	I_{OH} = –2.5 mA	V_{OH}	2.4	V_{CC} + 0.5	V
Output leakage current (Driver disabled)	$-0.5 \le V_{IO} \le V_{CC} + 0.5$ V	I_Z	–30	30	µA
For open drain circuits: (INT[7:1]* signals)					
Output low voltage	I_{OL} = 4.0 mA	V_{OL}	–0.4	+0.4	V
Driver leakage current (Driver negated)	$-0.5 \le V_{IO} \le V_{CC} + 0.5$ V	I_Z	–30	30	µA

6.3 Electronic timing requirements

6.3.1 AC parameters and measurements

SBus signal drivers shall be capable of meeting the timing specifications shown in the following table when driving the specified capacitive load.

SBus Cards shall be capable of operating across the entire allowable clock frequency range.

Rise and fall times are measured between 0.6 V and 2.2 V.

Setup, hold, and delay times are measured from the point where the signal being measured crosses the 1.5 V threshold.

6.3.2 Specification of clock skew

The skew between the rising edge of the CLK signal (T_{CS}) at all SBus Slot Connectors shall not exceed the value specified in Table 14. The CLK skew specification shall be met over the entire specified range of capacitive loading of the SBus Card. The skew measurements shall be made with a lumped capacitive load CL simulating the installed SBus Card.

6.3.3 Specification of input timing requirements

The timings and specification methods have been selected to guarantee that an SBus Card will be provided with signals that meet the specified set up and hold times with respect to the rising edge of the CLK signal. The same timings are available at SBus Devices installed on the SBus motherboard when the SBus Card outputs meet the specified output timings.

6.3.4 Specification of output timing requirements

For those signals driven by the SBus System, the output timings shall be guaranteed by the SBus System at the driving SBus Device with all SBus Cards installed. This will guarantee that the input timing requirement will be met at the SBus Card Connector.

For those signals driven by the SBus Card, the output timings shall be guaranteed at the SBus Card Connector with the specified total capacitive loading, simulated by a lumped capacitance C_L.

SBus Cards shall be designed to operate using the timings specified for 25 MHz operation to assure maximum interoperability.

The timing requirements for SBus Devices built into SBus Systems can be relaxed for clock frequencies lower than 25 MHz. The difference between the actual clock cycle and the 40 ns shortest allowable clock cycle can be added to the time allowed from the rising edge of CLK to the output being valid (T_{OD}). As an example, the T_{OD} for a system with a 20 MHz clock is 30 ns with a load of 160 pF and 26 ns with a load of 50 pF.

The output timing requirements are specified at two loadings. The higher capacitive loading is typical of heavily loaded SBus environments that have rise times limited by the capacitive loading. The dual timing requirement is intended to discourage the use of excessively powerful drivers to meet the timing requirements under full loads. Such drivers may cause ringing in lightly loaded SBus Systems. The faster timing specified for lightly loaded environments guarantees that the SBus signals will settle to the correct state in time to meet the input timing requirements.

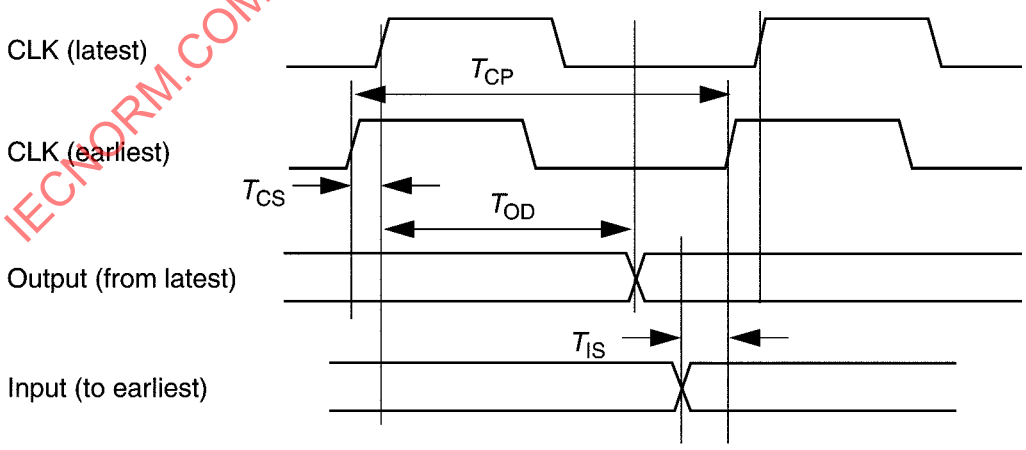
The specified input and output timings provide a skew budget of 3.5 ns, which takes into consideration propagation delay and similar tolerances.

Table 14 shows SBus a.c. signal limits.

Table 14 – AC parameters

Parameter	Condition	Symbol	Min.	Max.	Unit
CLK frequency	±200 ppm tolerance	F_{CLK}	16	25	MHz
CLK period	±200 ppm tolerance	T_{CP}	40	62.5	ns
CLK high time	F_{CLK} = 16 MHz – 25 MHz	T_{CH}	17		ns
CLK low time	F_{CLK} = 16 MHz – 25 MHz	T_{CL}	17		ns
Worst Case Skew between CLK signals for all SBus Slots	12 pF ≤ C_L ≤ 20 pF	T_{CS}	0	1.5	ns
CLK rise and fall time	C_L = 12 pF	T_{CR}, T_{CF}	1		ns
	C_L = 20 pF	T_{CR}, T_{CF}		3	ns
INT[7:1]* fall time	C_L = 160 pF, R_{PU} = 2 kΩ	T_{IF}	5	20	ns
INT[7:1]* rise time	C_L = 160 pF, R_{PU} = 2 kΩ	T_{IR}	5	350	ns
Rising edge of CLK to next cycle output valid @ 25 MHz	20 MHz ≤ F_{CLK} ≤ 25 MHz C_L = 100 pF	T_{OD25}	2.5	20	ns
Rising edge of CLK to next cycle output valid @ 25 MHz	20 MHz ≤ F_{CLK} ≤ 25 MHz C_L = 50 pF	T_{OD25}	2.5	16	ns
Output hold time after rising edge of CLK	C_L = 50 pF	T_{OH}	2.5		ns
Rising edge of CLK to output Hi Z		T_Z		T_{CP}–5	ns
Input setup time required before rising edge of CLK		T_{IS}		15	ns
Input hold time required after rising edge of CLK		T_{IH}		1	ns

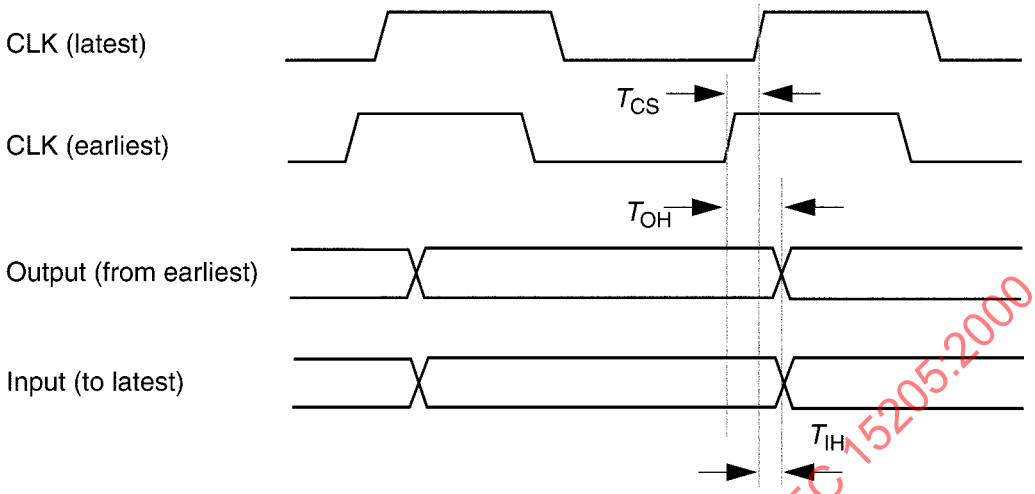
Figure 21 clarifies the relationship between the skew budget, propagation delay, input setup time, and clock to output delay.



$T_{IS} = T_{CP} - T_{OD} - T_{CS}$ – Prop Budget (max) – Skew Budget (max)
15 ns = 40 ns – 20 ns – 1.5 ns – 2.5 ns – 1 ns

Figure 21 – Clock to output and setup time relationships, including clock skew

Figure 22 clarifies the relationship between the skew budget, propagation delay, input hold time, and output hold time requirements.



$T_{IH} = T_{OH} - T_{CS}$ – Skew Budget (minimum)
1 ns = 2.5 ns – 1.5 ns – 0 ns

Figure 22 – Output and input hold time relationships, including clock skew

6.4 Compliance requirements

This standard does not specify requirements for compliance with safety or emission standards except that SBus Cards delivering optical or electrical power outside the SBus System shall have labels indicating their compliance with applicable power delivery safety standards.

SBus Cards should be designed to meet applicable requirements for emissions and safety, including labelling. Such design practices include proper shielding of external connectors, proper grounding practices, and proper electronic isolation as required by the function of the card.

7 Environmental requirements

7.1 Operating range

SBus Cards shall operate over a temperature range from 0 °C to +70 °C.

SBus Systems shall provide adequate cooling to maintain this operating environment for their full complement of SBus Cards as long as the installed SBus Cards meet the power and mechanical requirements of this standard.

8 Mechanical requirements

8.1 SBus Slot Connector

The SBus Slot shall use a high-density 96-pin SBus Connector. SBus Cards shall use a male SBus Card Connector mounted on the solder side of the board. SBus motherboards implementing SBus Slots shall use a female SBus Slot Connector mounted to allow proper mechanical support and electrical shielding and proper placement with respect to the SBus System's backplate.

Table 15 outlines the SBus Connector pinout.

Table 15 – SBus Connector pinout

01. GND	25. D[31]	49. CLK	73. D[30]
02. BR*	26. SIZ[0]	50. BG*	74. SIZ[1]
03. SEL*	27. SIZ[2]	51. AS*	75. PPRD
04. INT[1]*	28. INT[7]*	52. GND	76. GND
05. D[00]	29. PA[00]	53. D[01]	77. PA[01]
06. D[02]	30. PA[02]	54. D[03]	78. PA[03]
07. D[04]	31. PA[04]	55. D[05]	79. PA[05]
08. INT[2]*	32. EER*	56. +5 V	80. +5 V
09. D[06]	33. PA[06]	57. D[07]	81. PA[07]
10. D[08]	34. PA[08]	58. D[09]	82. PA[09]
11. D[10]	35. PA[10]	59. D[11]	83. PA[11]
12. INT[3]*	36. ACK[0]*	60. GND	84. GND
13. D[12]	37. PA[12]	61. D[13]	85. PA[13]
14. D[14]	38. PA[14]	62. D[15]	86. PA[15]
15. D[16]	39. PA[16]	63. D[17]	87. PA[17]
16. INT[4]*	40. ACK[1]*	64. +5 V	88. +5 V
17. D[19]	41. PA[18]	65. D[18]	89. PA[19]
18. D[21]	42. PA[20]	66. D[20]	90. PA[21]
19. D[23]	43. PA[22]	67. D[22]	91. PA[23]
20. INT[5]*	44. ACK[2]*	68. GND	92. GND
21. D[25]	45. PA[24]	69. D[24]	93. PA[25]
22. D[27]	46. PA[26]	70. D[26]	94. PA[27]
23. D[29]	47. DP	71. D[28]	95. RST*
24. INT[6]*	48. –12 V	72. +5 V	96. +12 V

Figure 23 shows the SBus Connector layout.

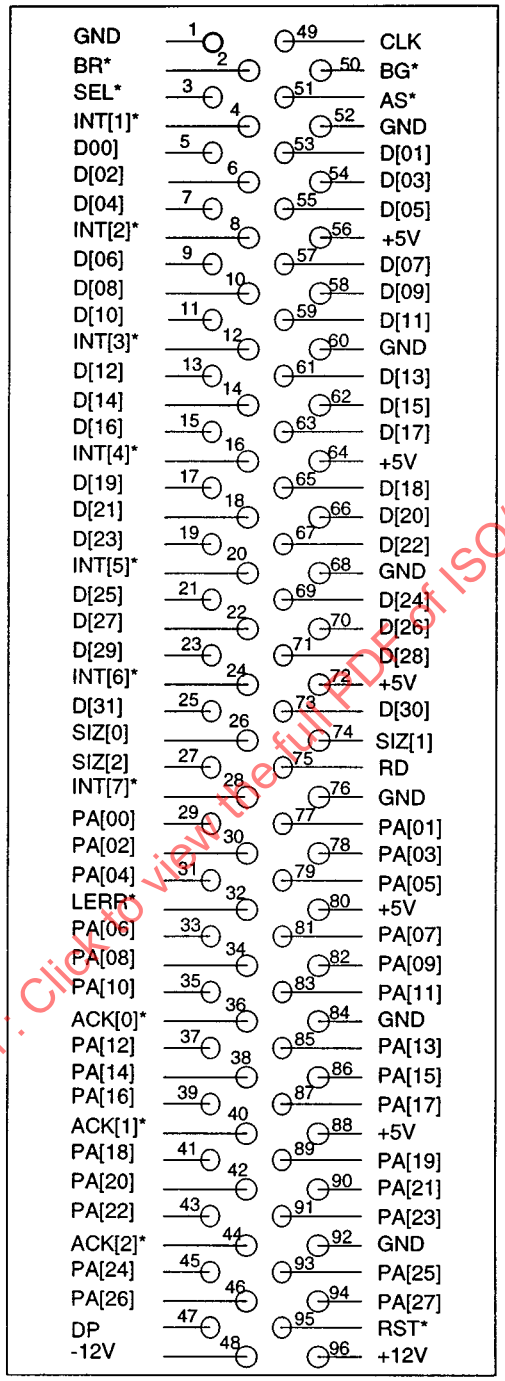


Figure 23 – Bus Slot signal location

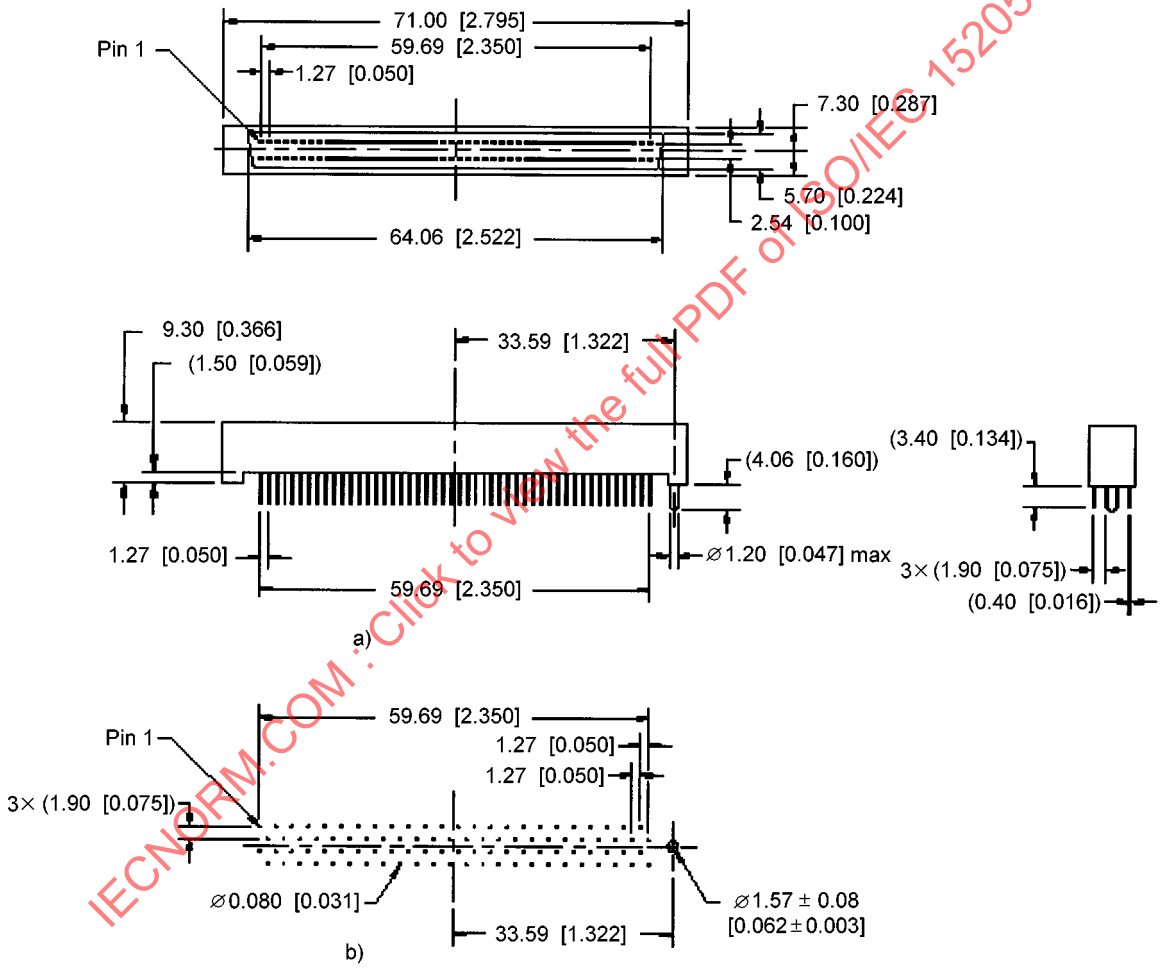
Double-width SBus Cards shall have two connectors as shown in Figure 32.

Both SBus Slot Connectors may be keyed or unkeyed. A keyed connector is identical to the unkeyed connector, except for the addition of a small plastic tab to assist in the proper installation of the connector at manufacturing time.

SBus Cards and motherboards should be laid out using the keyed connector PCB mounting hole pattern when through-hole connectors are used, allowing the use of both keyed and unkeyed connectors. The mounting information for the connector is provided for reference only. Surface mount and other mounting methods may be used. The female SBus Slot Connector may implement any appropriate mounting height from the motherboard. The mounting height of the SBus Card is required to provide the minimum component clearance between the SBus Cards's solder side components and the motherboard components that is specified by Figure 27.

The connector must be able to withstand short-term elevated temperature exposures in accordance with the time/temperature profiles of the manufacturing process used in the assembly of the SBus Card or SBus System. The connector and connector material must tolerate continuous contact pin operation at 105 °C maximum.

Figure 24 shows the mechanical details of the male SBus Card Connector.



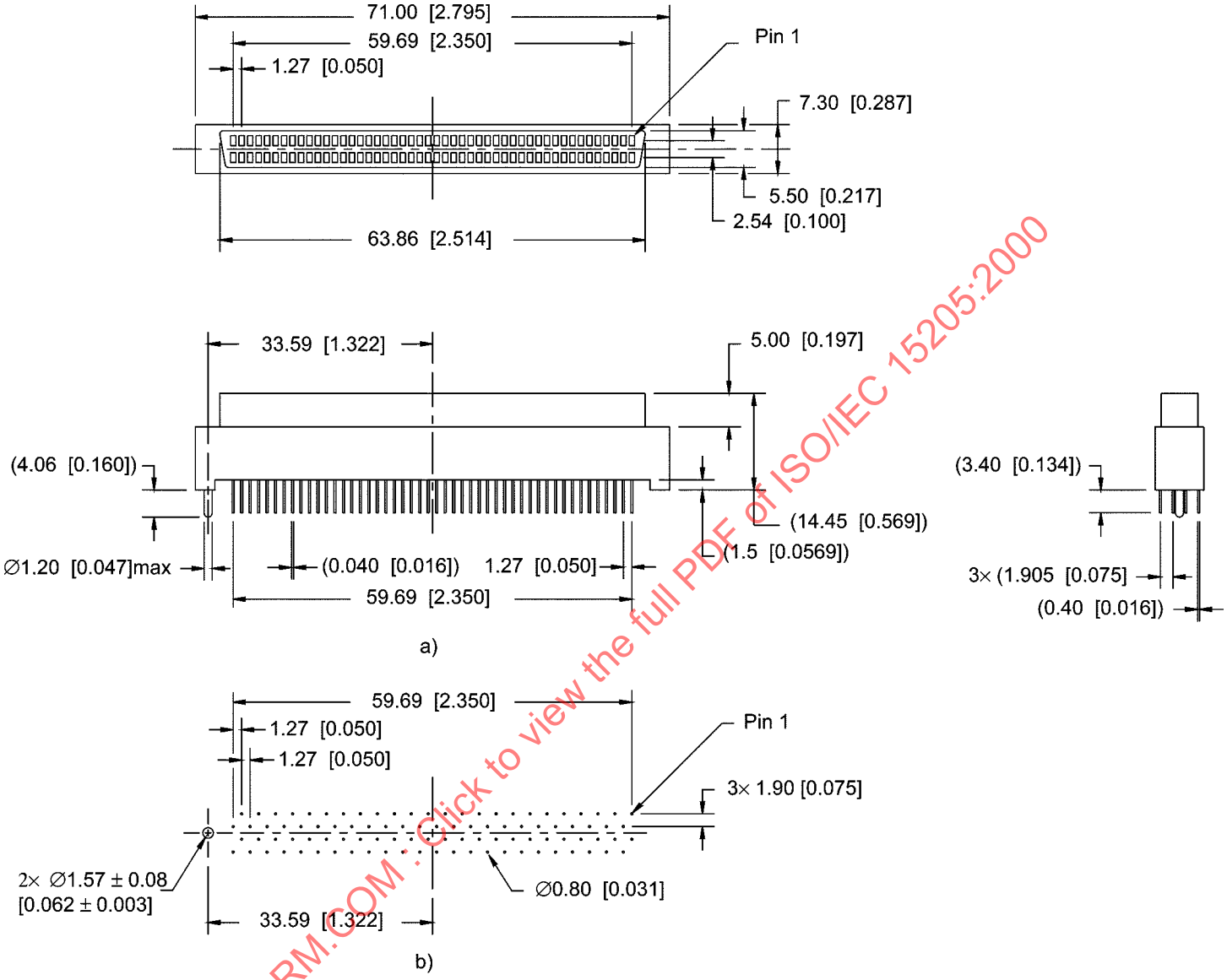
Dimensions in millimetres
[Dimensions in inches]

Key

a) Connector dimensions
b) Typical PCB mounting hole pattern (reference only).
c) Tolerances:
.X = ±0.3
.XX = ±0.12
Angle = ±0.5°

Figure 24 – Male SBus Card Connector

Figure 25 shows the mechanical details of the female SBus Slot Connector. (The unkeyed connector is compatible with the keyed connector.)



Dimensions in millimetres
[Dimensions in inches]

Key

a) Connector dimensions

b) Typical PCB mounting hole pattern (reference only).

c) Tolerances:

.X = ±0.3
.XX = ±0.12
Angle = ±0.5°

Figure 25 – Female SBus Slot Connector

8.2 SBus Card

Two types of SBus Cards are specified

– single width,
– double width.

SBus Systems having SBus Slots shall support single-width SBus Cards. The support of double-width SBus Cards is optional.

8.2.1 Board materials

The board shall meet the dimensional requirements indicated in the following drawings. The combination of board warpage, component lead length, and component height shall not exceed the specified maximum allowable component or lead height limits as shown in Figure 27. The board material shall meet applicable fire and smoke regulations. The board material shall meet the electrical and environmental requirements specified by this standard.

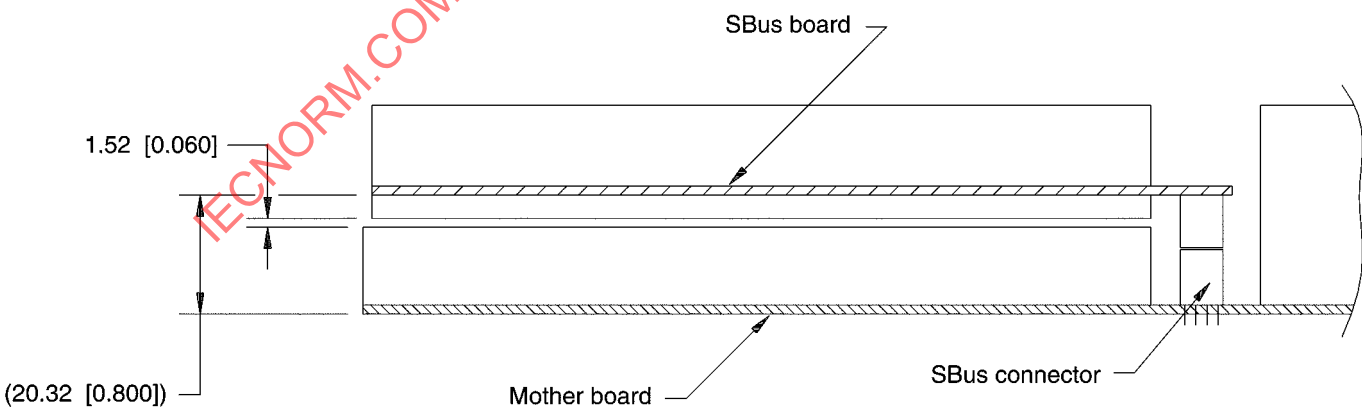
The board thickness shall be 1.6 mm 0.2 mm, in the trace free area specified by note 2 of Figure 28 and Figure 32. This allows for standard retention hardware and retainer designs. Other portions of the SBus Card may have any required circuit board thickness consistent with the maximum allowable component and lead heights.

8.2.2 Component clearance

Figure 26 specifies the minimum component clearance that shall exist between components on the motherboard and components on the solder side of the SBus Card. The clearance is intended to prevent contact between the SBus Card and the motherboard components.

The maximum component height, including board thickness, shall be as specified by Figure 27. The maximum component or lead height below the board shall be as specified by Figure 27. The specified component and lead height shall be absolute maximums and shall not be exceeded under conditions of board warpage.

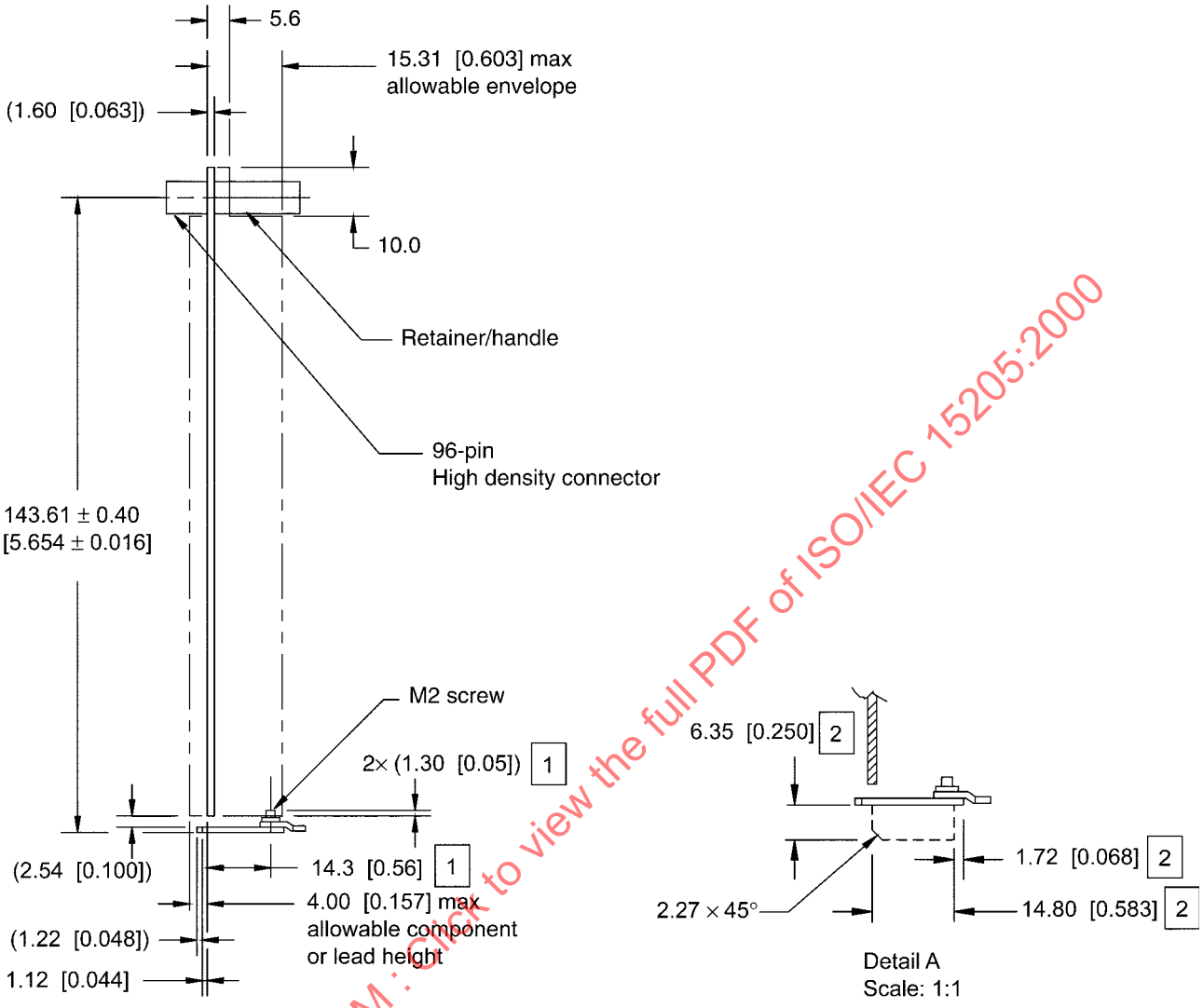
Figure 26 shows the minimum component gap for SBus Cards.



Dimensions in millimetres
[Dimensions in inches]

Figure 26 – Minimum component gap

Figure 27 shows allowable SBus component clearance.



Dimensions in millimetres
[Dimensions in inches]

Tolerances:
.X = ±0.7
.XX = ±0.4
Angle = ±0.5°

Key

1 Area occupied by M2 screws (see top view figures).

2 Recommended protrusion zone for connector and hardware.
Dimensions to be observed if back plate adaptor is used (see detail A).

Figure 27 – Component clearance

8.2.3 Backplate

Every SBus Card shall have a metal backplate. Figures 28 through 31 document the dimensions for a single-wide SBus Card, while Figures 32 through 35 document the dimensions for a double-wide SBus Card. The basic backplate is designed for installation in systems having limited maximum heights, such as VME-based applications. The addition of the backplate adapter allows the SBus Card to be installed in those desktop environments that use the adapter as part of the retention mechanism.

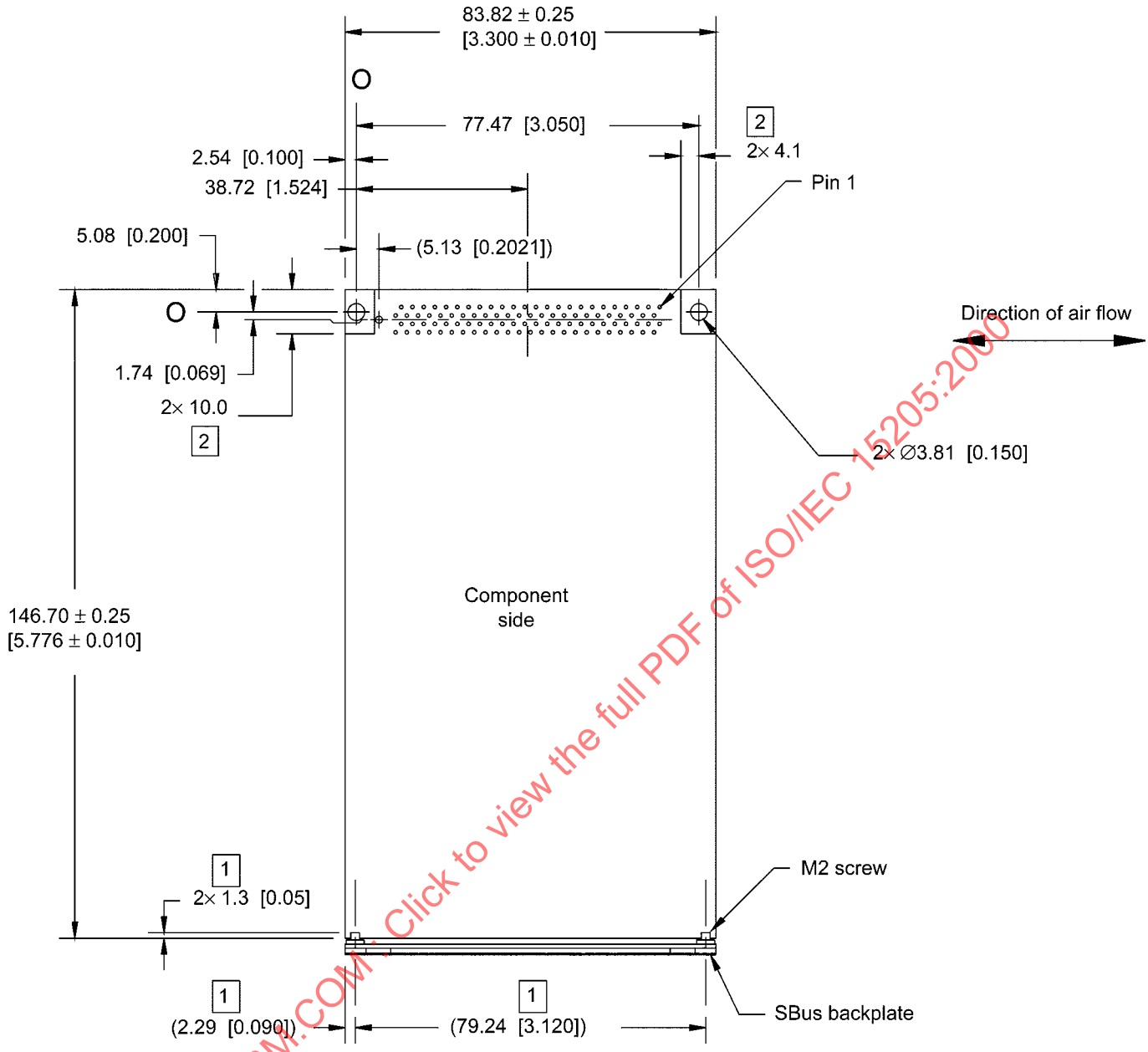
It is permissible to replace double-width backplates with two single-width backplates appropriately spaced across the back of the SBus card.

The area available for connector protrusions beyond the backplate shall be treated as a rectangular solid that extends perpendicular to the backplate, as shown in Figure 27, detail A. The width of the connector shall not exceed that specified by Figures 29 and 33. To allow the installation of the SBus Card in any SBus System, any connector used shall fit entirely within this space. SBus Systems shall be designed to allow the installation of an SBus Card with the maximum protrusion volume occupied by a connector. The connector dimensions include the connector shell and retention hardware. The SBus System's case shall allow for the connector clearance tunnel defined by item 1 of Figures 29 and 33.

After it is installed in the SBus System, the backplate shall be electrically connected to the chassis ground of the System. It shall not be connected to the logic ground on the SBus Card directly or through circuit components. Stray capacitance or inductance coupling the backplate to the logic board should be minimized. It may be necessary to electrically isolate the backplate connectors of an SBus Card from the backplate to meet this requirement.

8.2.4 Single-wide board and backplate

Figure 28 shows a single-width SBus Card. Figures 29 through 31 show a single-width backplate.



Dimensions in millimetres
[Dimensions in inches]

Tolerances:
.X = ±0.7
.XX = ±0.4
Angle = ±0.5°

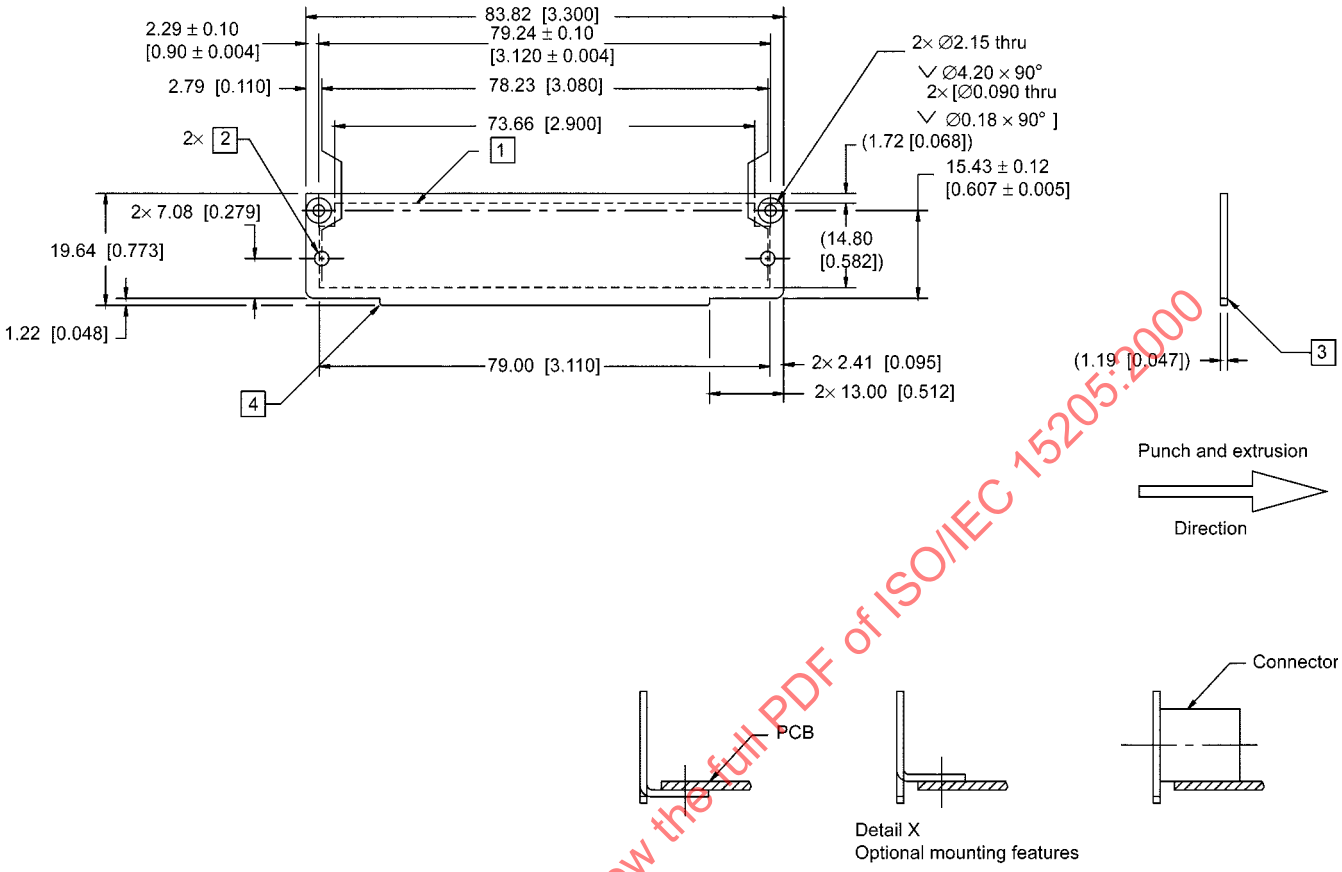
Key

1 Area occupied by M2 screws. See side view for complete detail.

2 Component and trace free area, all layers.

Figure 28 – Single-width SBus Card

Figure 29 shows a single-width backplane.



Dimensions in millimetres
[Dimensions in inches]

Tolerances:
.X = ±0.4
.XX = ±0.20
Angle = ±0.5°

Key

Available connector opening.

1 The available connector opening should be interpreted as a tunnel that extends outside the SBus backplate. The external connector body and connector mating parts shall stay within this tunnel.

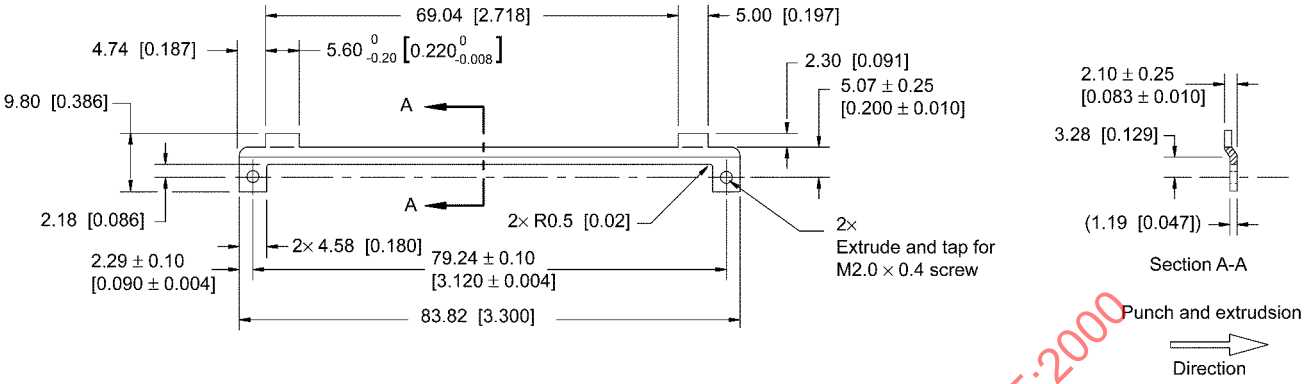
2 Extrude and tap for M2.5 × 0.45 screw.

3 See detail X.

4 Break exposed sharp edges for safety.

Figure 29 – Single-width backplate

Figure 30 shows a single-width backplate adaptor.



Dimensions in millimetres
[Dimensions in inches]

Tolerances:
.X = ±0.4
.XX = ±0.20
Angle = ±0.5°

Figure 30 – Detail of single-width backplate adaptor

Figure 31 shows a single-width SBus backplate assembly.

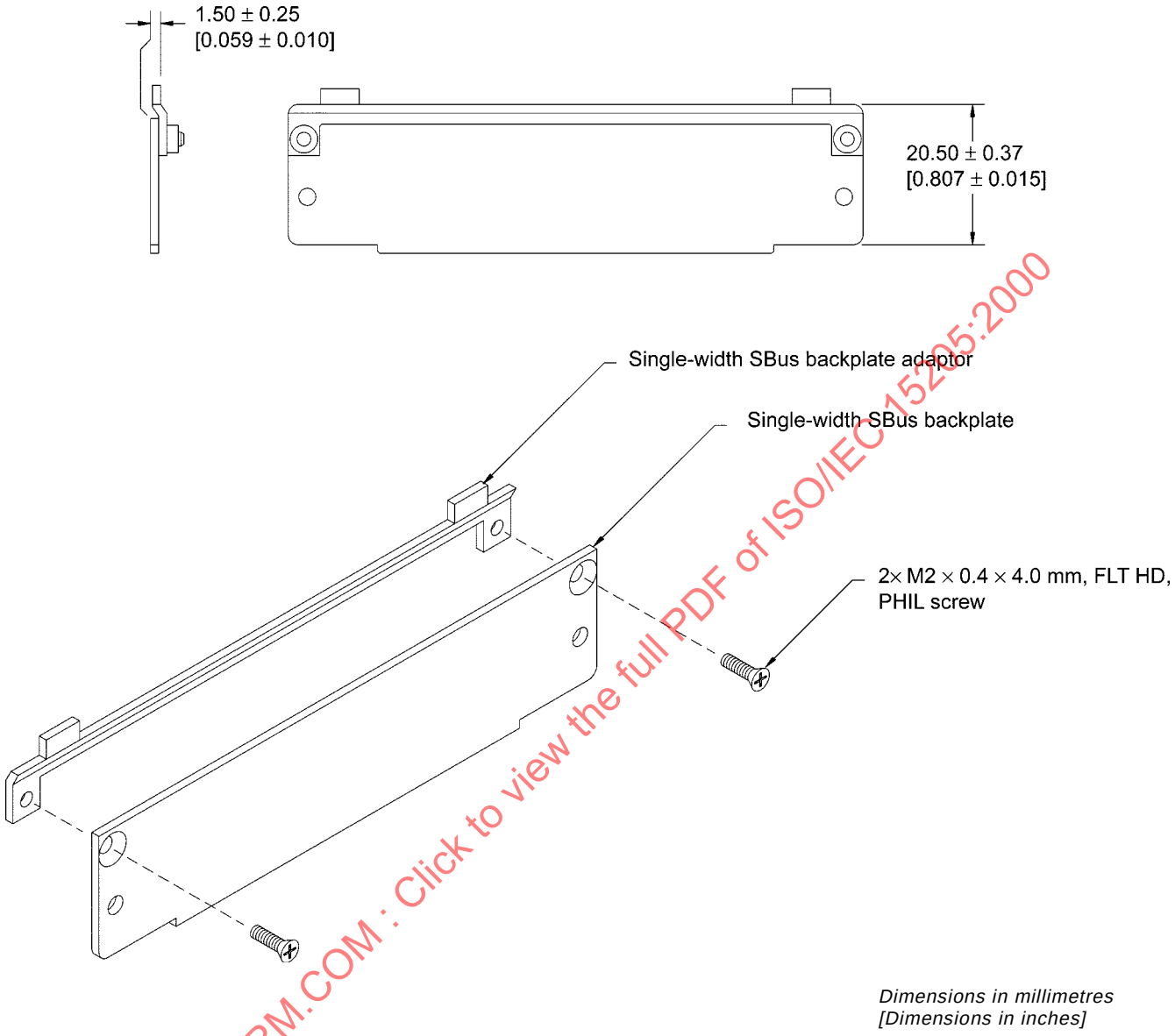
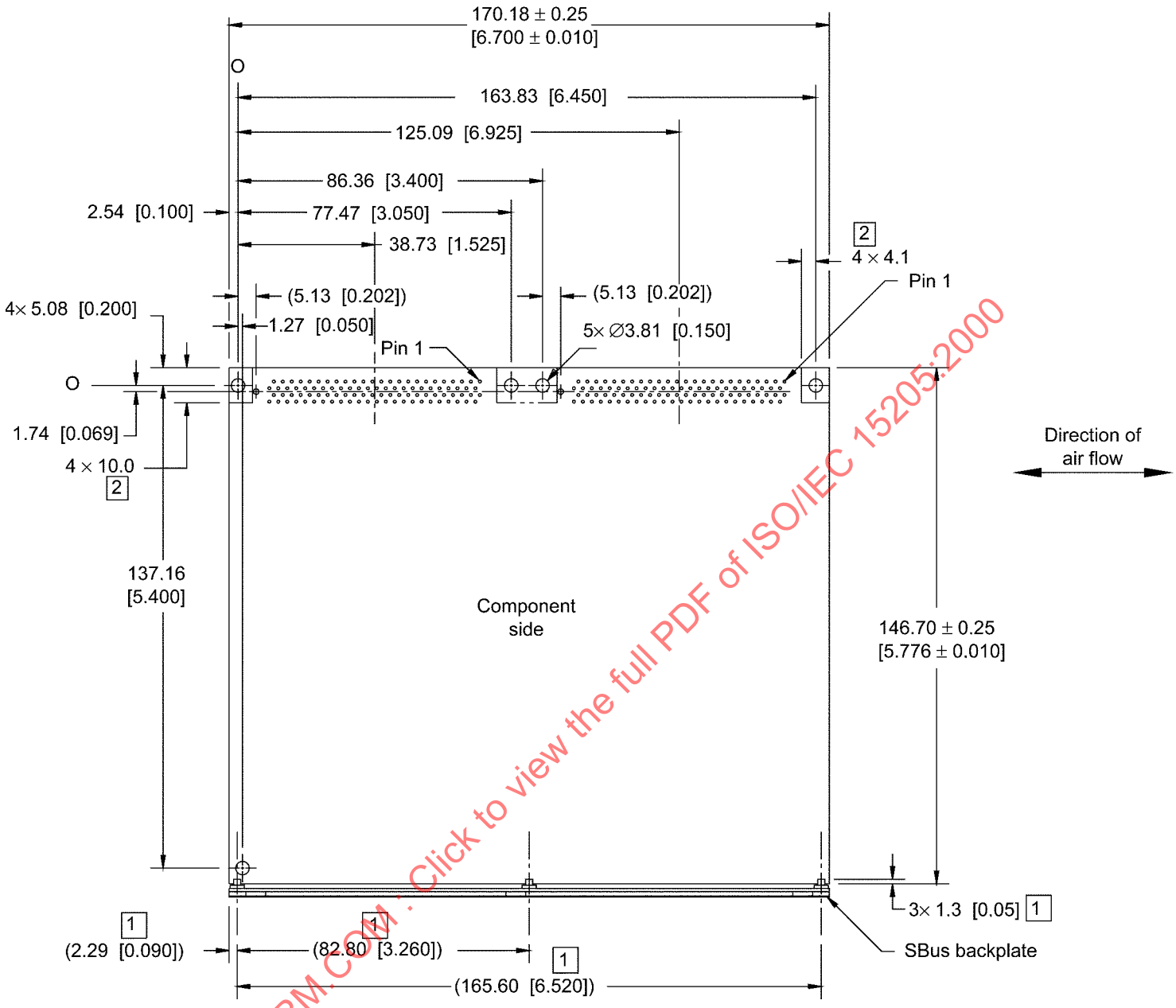


Figure 31 – Single-width backplate assembly

8.2.5 Double-wide board

Figure 32 shows a double-width SBus Card. Figure 33 shows a double-width SBus backplate. Figure 34 shows a double-width backplate adaptor. Figure 35 shows a complete double-width backplate assembly.



Dimensions in millimetres
[Dimensions in inches]

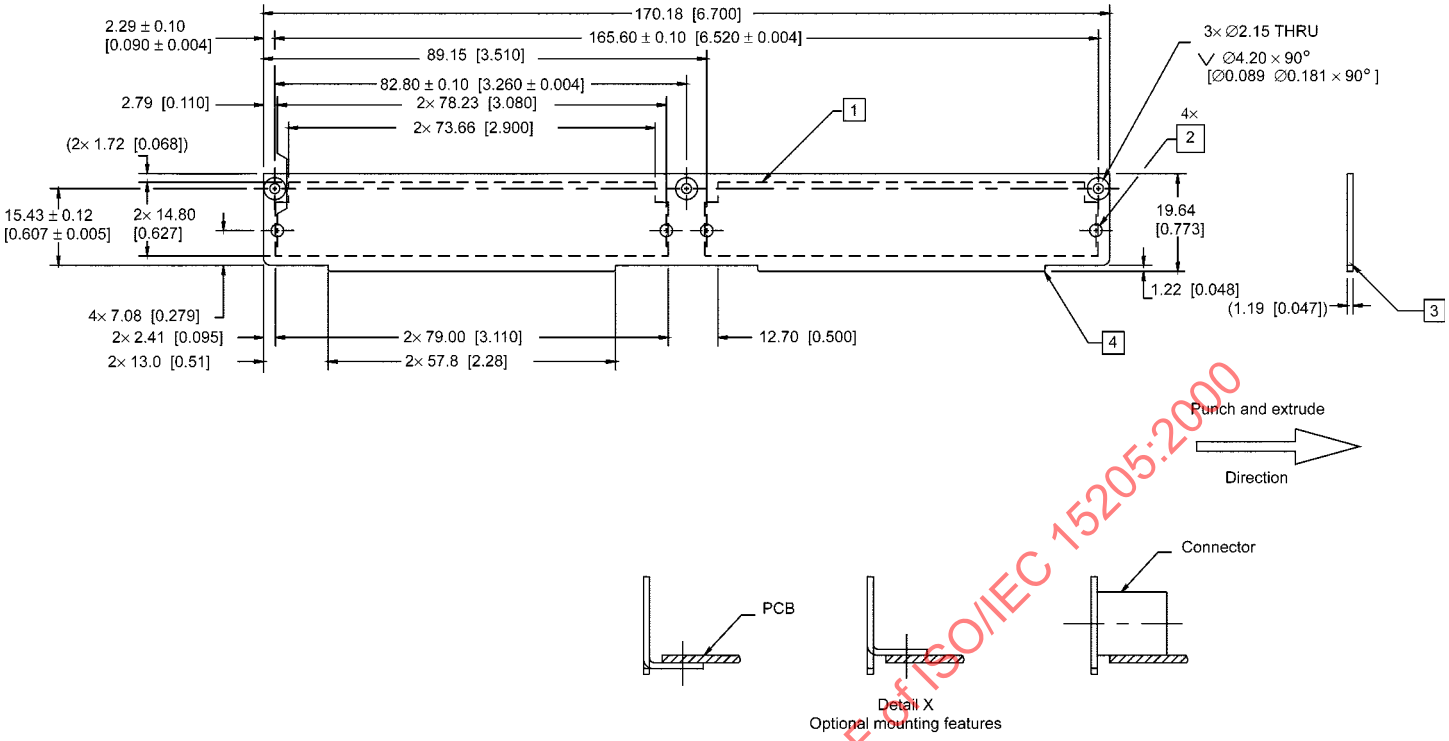
Tolerances:
.X = ±0.7
.XX = ±0.4
Angle = ±0.5°

Key

1 Area occupied by M2 screws. See side view for complete detail.

2 Component and trace free area, all layers.

Figure 32 – Double-width SBus Card



Dimensions in millimetres
[Dimensions in inches]

Tolerances:
.X = ±0.4
.XX = ±0.20
Angle = ±0.5°

Key

Available connector opening.

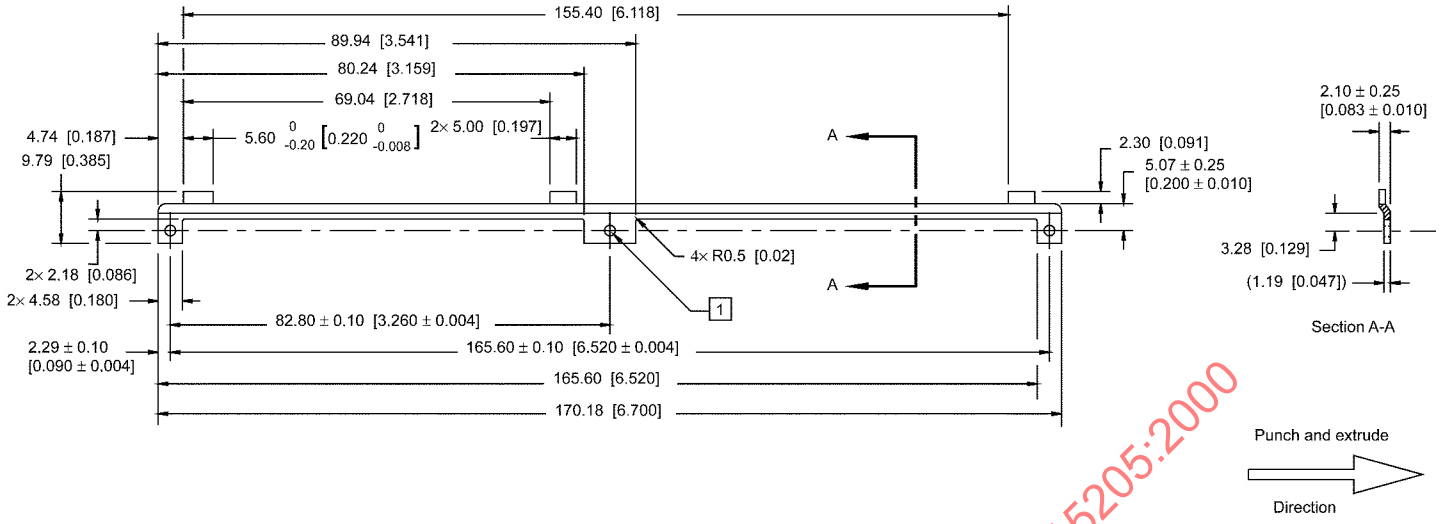
1 The available connector opening should be interpreted as a tunnel that extends outside the SBus backplate. The external connector body and connector mating parts shall stay within this tunnel.

2 Extrude and tap for M2.5 × 0.45 screw.

3 See detail X.

4 Break exposed sharp edges for safety.

Figure 33 – Double-width backplate



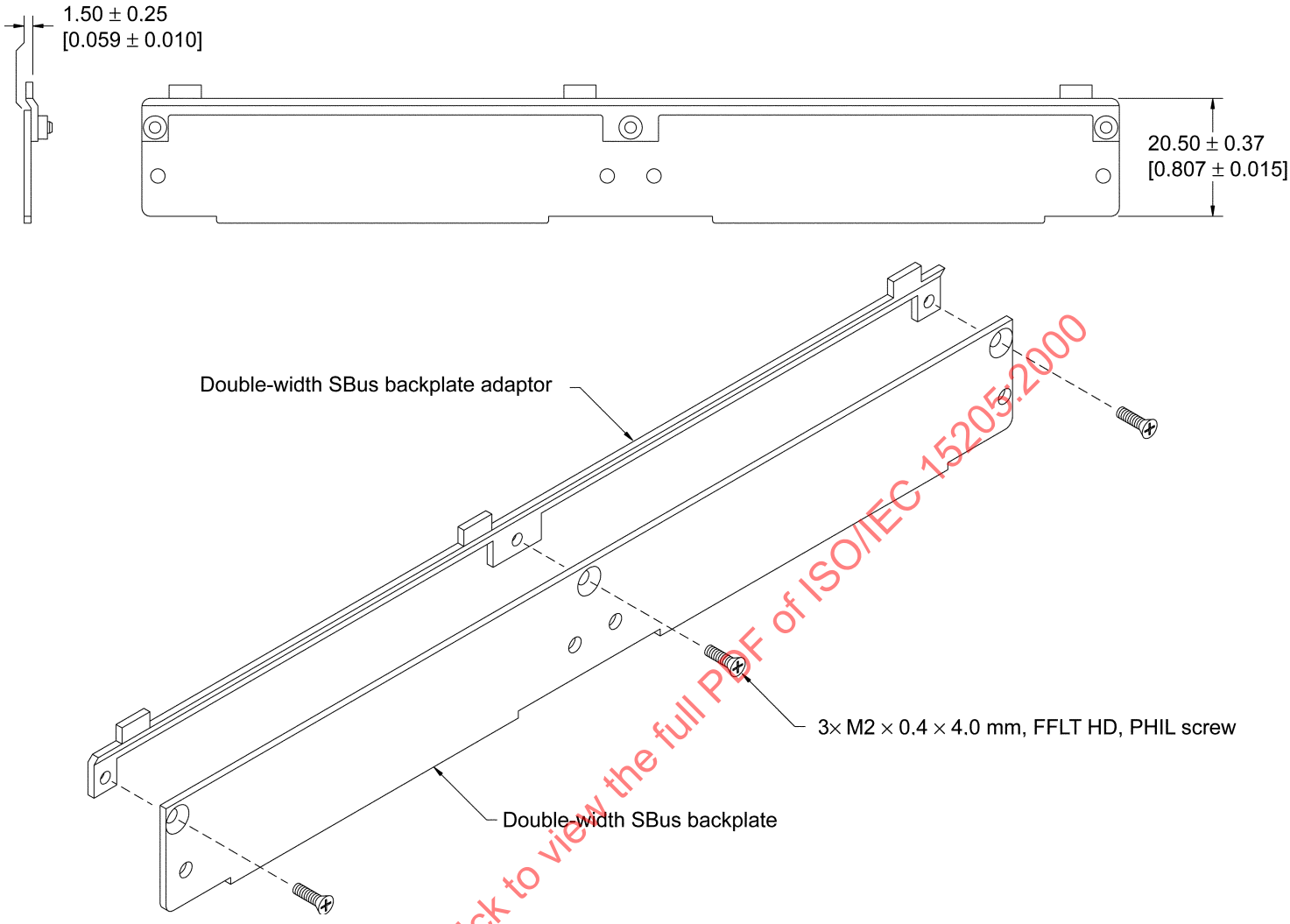
Dimensions in millimetres
[Dimensions in inches]

Tolerances:
.X = ±0.4
.XX = ±0.20
Angle = ±0.5°

Key

1 Extrude and tap for M2.0 × 0.4 screw.

Figure 34 – Detail of double-width backplate adaptor



Dimensions in millimetres
[Dimensions in inches]

Figure 35 – Double-width backplate assembly

8.2.6 Retention and standoff

The following figure documents the SBus retainer envelope. The retainer assists in the insertion or removal of the SBus Card. In some desktop SBus Systems, the retainer provides mechanical restraint for the SBus Card against shock and vibration.

A retainer fitting this design envelope shall be supplied with each SBus Card. The retainer shall have the strength required for its use in removing and inserting SBus Cards. The retainer shall be of an electrically nonconductive material. The retainer shall be removable from the SBus Card so that alternative SBus Card retention systems can be installed. If the SBus System requires a card retention structure other than the SBus retainer, it shall be provided by the SBus System.

Figure 36 shows an SBus retainer.

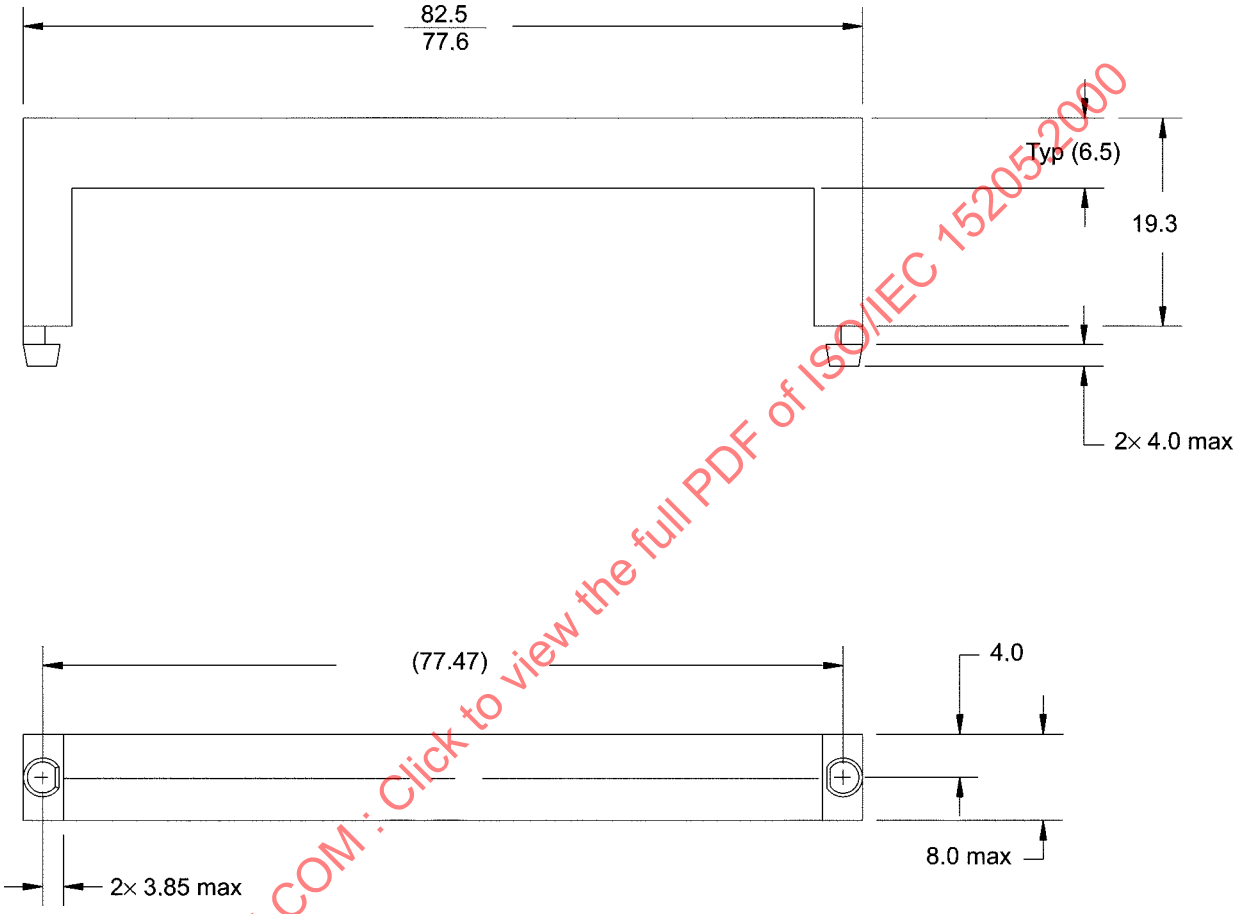
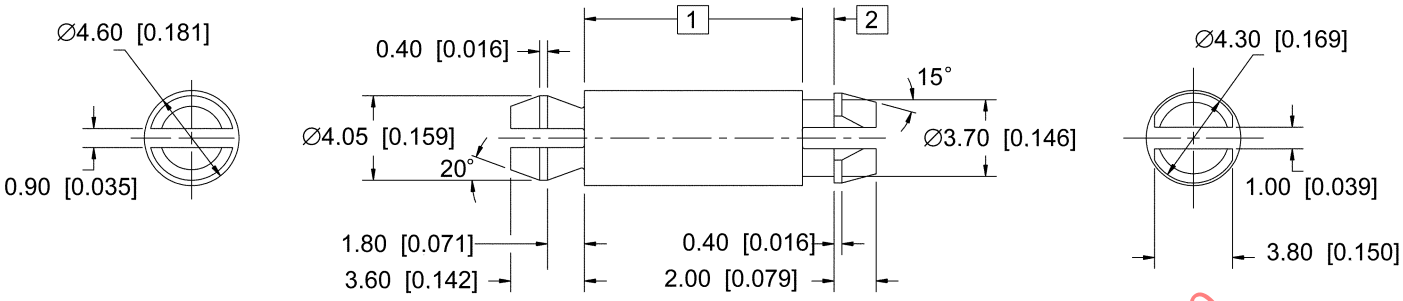


Figure 36 – SBus retainer

In non-desktop applications, such as laptops and VME-based applications, the retainer may not be necessary and can be easily removed. The tooling holes can be used for alternative card retention mechanisms, including various standoffs and clips.

Figure 37 shows the dimensions of an allowable SBus standoff.



Dimensions in millimetres
[Dimensions in inches]

Key

1 Equal to engaged height of SBus connectors.

2 Equal to thickness of mother board.

3 Material: Nylon 6/6.

Figure 37 – Allowable SBus standoff

Figure 38 shows an example of SBus installation.

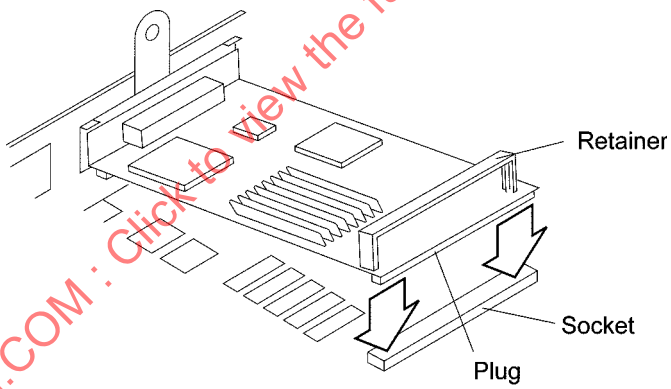


Figure 38 – SBus installation example