

IEEE Standard for Fault Tolerant Extensions to the Futurebus+™ Architecture

Sponsor
**Bus Architecture Standards Committee
of the
IEEE Computer Society**

Approved December 13, 1994
IEEE Standards Board

Abstract: This standard is one in a family of Futurebus+™ standards. The Futurebus+ standards provide a set of tools with which to implement a bus architecture with performance and cost scalability over time for multiple generations of single- and multiple-bus multiprocessor systems. This standard provides fault tolerant extensions to Futurebus+ standards. As such, this standard provides the logical layer requirements for the transmission of data in a fault tolerant environment. When used in conjunction with other IEEE standards, the details to develop modular, open-architecture-based systems fulfilling user needs across a wide computing spectrum are available.

Keywords: bus, bus architecture, computers, data transmission, fault tolerance, Futurebus+, multiprocessor systems

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

Copyright © 1995 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1995. Printed in the United States of America

ISBN 1-55937-516-7

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

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

IEEE Standards documents may involve the use of patented technology. Their approval by the Institute of Electrical and Electronics Engineers does not mean that using such technology for the purpose of conforming to such standards is authorized by the patent owner. It is the obligation of the user of such technology to obtain all necessary permissions.

Introduction

(This introduction is not a part of IEEE Std 896.9-1994, IEEE Standard for Fault Tolerant Extensions to the Futurebus+™ Architecture.)

This document is a member of the family of IEEE P896 Futurebus+™ documents. It is sponsored by the Bus Architecture Standards Committee of the IEEE Computer Society. The initial work in the development of Futurebus+ was done under the auspices of the IEEE Computer Society Microprocessor Standards Committee. In 1988, both the United States Navy's Next Generation Computer Resources (NGCR) Task Force and the VMEbus International Trade Association (VITA) (a trade association of both VMEbus manufacturers and users) agreed to join the IEEE in revising IEEE Std 896.1-1987.

The primary goal of all these groups was to provide a new microprocessor bus standard that would be commercially viable and that would be acceptable to manufacturer groups and user communities.

This work resulted in the IEEE P896 family of standards, which defines Futurebus+. ISO/IEC 10857 : 1994 (IEEE Std 896.1-1991 and IEEE Std 896.1a-1993) defines the logical functionality of the set of signals that make up the bus. IEEE Std 896.2-1991 describes and specifies the physical layer (electrical characteristics, pin-outs, connector locations, board sizes, etc.) required and contains application environment profiles. IEEE Std 896.9-1994 describes the fault tolerant extensions to Futurebus+. These extensions are based on and are compatible with the Futurebus+ protocols defined in ISO/IEC 10857 : 1994. As such, this standard represents a fault tolerant layer that resides on top of ISO/IEC 10857 : 1994. IEEE Std 896.9-1994 describes a method for fault detection and a range of fault tolerant applications: fail-silent, fail-safe, and fail-operate.

Where deemed appropriate by the working group and the sponsor, further clauses may be added to this document at a later time by the mechanism of a supplement, without the need to revise the clauses previously standardized.

The software interface to the core registers, which are used to reset and configure nodes on a bus, is defined by the ISO/IEC 13213 : 1994 (IEEE Std 1212, 1994 Edition). This definition is shared by the IEEE Std 1596-1992 [Scalable Coherent Interface (SCI)], and IEEE P1394 (Serial Bus).

Acknowledgments

IEEE Std 896.9-1994 incorporates many years of effort in defining fault tolerance for Futurebus+ applications. This standard is the result of the combined efforts of many people from both industry and the government who feel that Futurebus+ can support very robust fault tolerant applications.

This standard was developed with support from the Navy's Next Generation Computer Resources (NGCR) program. Acknowledgment is expressed for the support provided by the United States Navy Officers Harrison Beasley, Robert Voigt, Vincent Squitieri, and Gary Pash. In particular, thanks are expressed to Harrison Beasley for his understanding of the issues and encouragement of this activity.

Thanks are extended to all working group members and contributors of this standard. Special thanks are due to those who have participated from the beginning of the working group effort in 1988 to the end in 1994. Because of the length of the development process and the large number of people involved at different times, undoubtedly some names may be inadvertently omitted from the list of working group members and contributors. Apologies are expressed to those people.

This standard has come about through the efforts of the many people who spent countless hours and days writing and reviewing the numerous iterations of this document. Special thanks are extended to several key contributors who provided expertise in semiconductor design engineering and applications for telecommunications, space, and military systems: Jay Cantrell, Ian Dobson, R. Stephen Hauck, Ralph Lachenmaier, Jeff Lear, Joseph Marshall, Lars Thernsjo, Dave Wickliff, and Robert Widlicka. Thanks are extended to R. Paul Dixon, the working group vice chair, who helped get this standard off the ground, and to Mike Dorsett, who has faithfully supported this effort through his work as the working group secretary. Much appreciation and thanks is expressed to Ed Jacques for the hundreds of hours spent editing and revising the many versions of this standard. Ed's dedication, persistence, and good judgment were key factors in the timely completion of this standard.

Working group members

The following members of the Futurebus+ Fault Tolerance Working Group were involved in the development and review of this standard:

Cory Ackerman
Daryl Adams
Katsuhiko Aka
Peter Alcivar
Sam Babb
Harrison A. Beasley
Mitch Beaumont
Steve Belvin
Chip Bettencourt
John Bolton
Mark Bunker
Douglas Caldwell
Jay Cantrell
Stephen Cecil
Mark Chestnutwood
Tom Clark
Tony Comito
Steven R. Corbesero
James Davidson
R. Paul Dixon
Ian Dobson
Mike Dorsett
Frank Duffy
Bruce Dunlop
Wilhelm Evertz
Wayne Fischer
Dan Flynn
John Franz
Richard Fryer
Gary Gardner
Robert George

Michael S. Foster, *Chair*

Darryl Gunn
Claes Goran Gustavsson
Emil Hahn
Mark Hassel
R. Stephen Hauck
Thomas G. Hillyer
Gary Hokenson
Fred Jackson
Ed Jacques
Jeff Jaska
Anatol Kaganovich
Loron Keith
David Kirk
Chris Koehle
Ralph Lachenmaier
Dale Langston
Ben LaPointe
C. A. Laurvick
Jeff Lear
Alexander Levin
Alexander Lewin
Rollins Linser
Ernest Mardaga
Edward Marian
Joseph R. Marshall
Brian Martin
Fred Martin
Karl McClure, Jr.
S. Fenton McDonald
Robert McKee

Thanos Mentzelopoulos
Tim Monaghan
Michael Munroe
Thomas Mylott
Ken Naumann
Clarence Nykl, Jr.
Katsuyuki Okada
Peter Onufryk
Jonathan Parlan
Gary Pash
Thomas Patric
Gary Postiglione
Thomas Potyraj
Edward Rodriguez
John Rynearson
William Schneider
Donald Senzig
Patricia Smith
Vincent Squitieri
Thomas Stretch
Haruhisa Suzuki
Lars Thernsjo
John Theus
Dean Van De Walker
Robert Voigt
Yoshika Wakimura
Larry Werlinger
Dave Wickliff
Robert Widlicka
David Wright
Janusz Zalewski

The following persons were on the balloting committee:

Barbara P. Aichinger
Keith D. Anthony
David Bartek
Harrison A. Beasley
John C. Brightwell
Charles Brill
Robert Crowder
Joe Devine
Ian Dobson
Sourav K. Dutta
Wayne Fischer
Michael S. Foster
Paul J. Fulton
Gary Gardner
Joseph D. George
David B. Gustavson

Mark Hassel
Thomas G. Hillyer
Frank Hom
Ed Jacques
Norman C. Joehlin
James R. Koser
Ralph Lachenmaier
Conrad Laurvick
Gerald E. Laws
Jeff Lear
Lim Boon Lum
Roland Marbot
Joseph R. Marshall
S. Fenton McDonald
John W. McKinstry
Thanos Mentzelopoulos
Klaus Dieter Mueller

Granville Ott
Elwood Parsons
Lalit Mohan Patnaik
Mira Pauker
Frederick E. Sauer
Norman Schneidewind
Donald Senzig
Richard Spratt
Robert Tripi
Thomas W. Vrankar
Yoshiaki Wakimura
Dave Wickliff
Robert Widlicka
J. Robert Wood
Oren Yuen
Janusz Zalewski

When the IEEE Standards Board approved this standard on December 13, 1994, it had the following membership:

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

Gilles A. Baril
Bruce B. Barrow
José A. Berrios de la Paz
Clyde R. Camp
James Costantino
Stephen L. Diamond
Donald C. Fleckenstein
Jay Forster^{*}
Ramiro Garcia

Donald N. Heirman
Richard J. Holleman
Jim Isaak
Ben C. Johnson
Sonny Kasturi
Lorraine C. Kevra
E. G. “Al” Kiener
Ivor N. Knight

Joseph L. Koepfinger^{*}
D. N. “Jim” Logothetis
L. Bruce McClung
Marco W. Migliaro
Mary Lou Padgett
Arthur K. Reilly
Ronald H. Reimer
Gary S. Robinson
Leonard L. Tripp

^{*}Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
James Beall

Richard B. Engelman
Robert E. Hebner

Mary Lynne Nielsen
IEEE Standards Project Editor

Futurebus+ is pending trademark registration by the Institute of Electrical and Electronics Engineers, Inc.

CLAUSE	PAGE
1. Overview	1
1.1 Scope	1
1.2 Purpose	1
1.3 Fault tolerant system models	2
1.4 Use of this standard	3
2. References	4
3. Definitions	5
3.1 Special word usage	5
3.2 Definitions for Futurebus+ systems	5
3.3 Definitions for fault tolerant systems	5
3.4 Syntax	7
4. Error detection	7
4.1 Introduction	7
4.2 Fault detection mechanisms	9
5. Error handling	13
5.1 Introduction	13
5.2 Error descriptions	14
6. Error reports	43
6.1 Introduction	43
6.2 Snap groups	44
6.3 System notification	47
7. Error recovery	48
7.1 Introduction	48
7.2 Recovery strategies	50
7.3 Retry strategies	51
7.4 Containment strategies	51
7.5 Local error signal	52
7.6 Alternate access	52
8. CSR definitions	52
8.1 Introduction	52
8.2 CSR summary	53
8.3 ROM CSR extensions	53
8.4 Futurebus+-dependent CSR extensions	54
8.5 Core CSR extensions	70
8.6 Alternate access bus-specific CSRs	77
Annex A (informative) Failure modes effects analysis	78

IEEE Standard for Fault Tolerant Extensions to the Futurebus+™ Architecture

1. Overview

1.1 Scope

This standard specifies the functions needed to implement a fault tolerant Futurebus+™ backplane. It does *not* specify how to implement a fault tolerant system or how to implement a fault tolerant backplane. The functions specified are required for any system using Futurebus+ with a fault tolerant requirement. It is expected that this standard will be used in its entirety and will be cited by Futurebus+ application environment profiles requiring fault tolerance.

This standard is composed of eight clauses and an annex. References required by this standard are defined in clause 2. Key words used in this standard are defined in clause 3.. Detection mechanisms (not covered by ISO/IEC 10857 :1994 and IEEE Std 896.2-1991)¹ that support single-bit and single-point fault coverage are discussed in clause 4. A detailed description of error handling requirements is described in clause 5. A description of the error reports generated from the detection of an error are characterized in clause 6. Basic recovery functions (i.e., retry) are covered in clause 7. CSR definitions not covered by clause 3 of IEEE Std 896.2-1991 are specified in clause 8. The annex contains a Failure Modes and Effects Analysis (FMEA) of the signals protected by this standard.

This standard is an extension of the concepts developed in ISO/IEC 10857: 1994 and IEEE Std 896.2-1991. Knowledge of those standards is necessary for a complete understanding of this standard. Because the cache coherency and message passing protocols described in ISO/IEC 10857 :1994 are built on top of the basic signaling conventions, they are beyond the scope of this standard and are not addressed by this document. However, they may be addressed by a future extension to this standard.

1.2 Purpose

The high-level goal of this standard is to ensure that Futurebus+ systems requiring fault tolerance can be implemented with minimal impact on performance and costs. Specific goals supported by this standard include the following:

- a) No bad data is transferred across the backplane undetected.
- b) At a minimum, single-bit errors are detected.
- c) Errors are logged and reported.
- d) Backplane supports system halt capability after a fault is detected.

¹Information about references can be found in clause 2.

It is unrealistic to consider all possible errors for fault detection. It is, however, reasonable to consider a set of errors that covers a significant portion of the possible errors. The set containing single-bit and single-point failures is considered such a set. All single-bit and single-point failures on all signal lines should be detected and reported to all modules on the bus. Without 100% detection and reporting, it is possible for an error to corrupt state information in a module on the bus without the knowledge of the module or system. While this is not ideal, it is equally unrealistic to require the detection of 100% of all single-bit and single-point failures, just as it is unrealistic to consider detection of all possible errors. It is always possible to conceive and contrive a set of circumstances that create a window of opportunity to defeat detection mechanisms. The success of detection mechanisms relies on the fact that the probability that an error may not be apparent is acceptably small. For the purposes of this standard, this probability is acceptably small if it exceeds the probability of a double-bit error. For systems requiring better fault coverage, measures beyond the scope of this standard have to be taken.

The aim of the extensions specified in this standard is to establish the low-level framework that brings the fault detection, recovery, and error reporting capabilities of Futurebus+ up to the level demanded by any system that claims fault tolerance. These extensions do *not* establish a complete definition of a fault tolerant environment. Higher level issues, like systems with dual-redundant buses, systems using networked back-planes to achieve fault isolation, and many other configurations, are left to the system designer or to profiles.

NOTE — These goals have been formulated cooperatively and are consistent with those required by the following Futurebus+ working groups: Military, Telecommunications, and Spaceborne.

1.3 Fault tolerant system models

The fault tolerant extensions defined in this standard are designed to allow a wide variety of fault tolerant systems to be built using Futurebus+. Defining the variety of possible fault tolerant systems as a spectrum, two endpoints and a significant midpoint were chosen as system models and guides. This provides consistent points of reference for the various types of systems developed with this standard. Three models have been chosen and are defined in the remainder of this clause. Each model includes all of the properties of the preceding model. These models are nonbinding guidelines in this standard. Their purpose is to provide points of reference when doing system analysis using Futurebus+ and the extensions for fault tolerance.

1.3.1 Fail-Stop

The fail-stop model represents fault tolerant systems at one end of the spectrum, where system requirements are such that system state information must be protected from corruption due to an error. While this requirement is for the system, the scope of this standard covers only the backplane bus, including protocol controllers and signal transceivers. A fail-stop system requires fault detection, error logging, and a method of halting the system in a defined state.

1.3.2 Fail-Safe

For the middle of the spectrum, this standard defines a fail-safe system as one whose failure can only be, or is to an acceptable extent, a benign failure. A benign failure is one where the consequences of the failure are of the same order of magnitude (in terms of cost) as the benefit provided by the service delivery in the absence of failure. The term “safe” refers to the fact that the system fails in such a way as to avoid catastrophic consequences on the system or the environment. Two types of fail-safe systems are further defined: fail-safe-stop and fail-safe-continue. A fail-safe-stop system detects a failure, takes action to assure that the system is in a state that does not cause catastrophic consequences, and then stops. A fail-safe-continue system detects a failure, takes action to assure that the system is in a state that does not cause catastrophic consequences, reconfigures, and then continues.

A system designed to this level of fault tolerance meets the requirements of the fail-stop model since state information has to be protected in order to ensure the system ends up in a “safe” state. This model has the same detection, recovery, and error reporting requirements as the fail-stop model, but it requires the presence of an alternate access bus and related access to the Futurebus+ control and state registers (CSRs). These items are application specific and therefore beyond the scope of this standard. It is expected they would be specified by a profile covering an application area requiring a system that approximates that described by the fail-safe model.

1.3.3 Fail-Operate

For the other end of the fault tolerant spectrum, this standard defines a fail-operate system as one that has to operate in the presence of errors. This may be achieved through hardware or software masking, with the mechanism selected based on system performance requirements. The reliability requirements of the system will determine how many errors can be tolerated and how much redundancy is needed for each element. This reliability is especially dependent on environment and length of mission.

A system designed to operate as described by this model would meet the requirements of both the fail-stop and fail-safe models, since faults are masked and the system continues operating as if the fault did not occur (with the exception of background fault isolation and recovery tasks). This model has the same detection, recovery, and error reporting requirements as the other models, and it requires an alternate access bus and related CSRs as for the fail-safe model. In addition, this model requires that redundant resources and resource management be available to support fault masking. The type and amount of redundancy and management required is application specific and therefore beyond the scope of this standard. It is expected that redundancy features would be specified by a profile covering an application environment requiring a system that approximates that described by the fail-operate model.

1.4 Use of this standard

This standard was developed as a layered product (see figure 1-1) that sits between a profile and the base Futurebus+ standards (ISO/IEC 10857: 1994 and IEEE Std 896.2-1991). The expectation is that the users of this standard will not start with this document but instead will be directed to it when it is referenced by profiles that are characterized as fault tolerant.

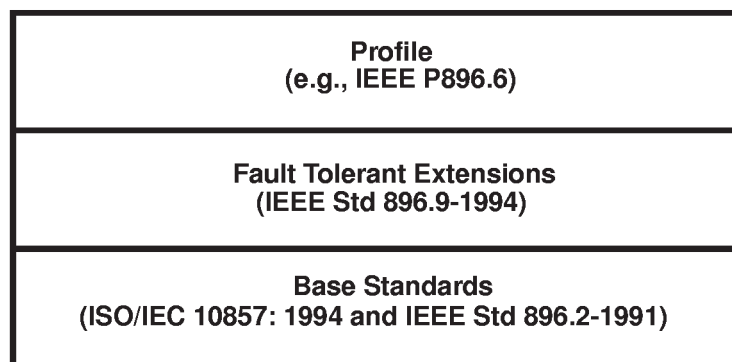


Figure 1-1—Document layers

To the silicon implementor, this standard describes the additional functions that need to be included in a fault tolerant design. The implementor has to use the base standards and then add the detection mechanisms described in clause 4 the error reports of clause 6, the error recoveries of clause 7, and the CSRs of clause 8. These are woven together in clause 5 and sorted by error classes. By using the error classes of clause 5, additional restrictions are placed on the implementations specified in the other clauses. The error recovery clause carefully specifies what has to be done versus what could be done. Thus, the silicon implementor has some latitude in deciding what has to be implemented in silicon and what can be deferred to the software in a system. It is crucial that such tradeoffs be documented so that system builders can include all required features.

To the module or system builder, this standard provides a wealth of information. A functional fault analysis (i.e., an analysis of error classes and how they are handled at each error containment region) is a recommended front-end system engineering activity. A module is a natural error containment region. The error handling

scenarios provided in clause 5 provide most of the potential interface error classes. These may then be extended to include the actual functions of the module and any user interfaces on a module-by-module basis to provide a complete view of the error classes at the Futurebus+ level of the system. These may be further classed together at the next level of system interface (perhaps between Futurebus+-based systems) until a complete analysis is completed. If a look at error containment at the component level or within a module is necessary, the FMEA provided in Annex AA provides a useful starting point to be once again expanded by the particulars of the module.

To the device driver writer, the detailed CSR definitions of clause 8 are important. These have to be supplemented by the information in ISO/IEC 10857: 1994, IEEE Std 896.2-1991, and ISO/IEC 13213: 1994. The error scenarios in clause 5 indicate what shall be logged in each error case and provide a structure for handling the many types of errors that may be detected. Recovery descriptions in clause 7 can be the starting point for error handling routines. Fault injection and diagnostic services can be built by using the additional CSRs provided. Because the error handling and diagnostic services are contained within the device driver, most of this standard is transparent at the application program interface.

2. References

IEEE Std 896.2-1991, IEEE Standard for Futurebus+™—Physical Layer and Profile Specification.²

IEEE Std 896.2a-1994, IEEE Standard for Futurebus+™—Physical Layer and Profile Specification: Errata, Corrections, and Clarifications.

IEEE Std 896.3-1993, IEEE Recommended Practices for Futurebus+™.

IEEE Std 896.5-1993, IEEE Standard for Futurebus+™, Profile M (Military).

IEEE P896.6 (D3.0), Draft Standard for Futurebus+™, Telecommunications Systems, Profile T.³

IEEE P896.10 (D3.4), Draft Standard for Futurebus+™, Spaceborne Systems, Profile S.⁴

IEEE Std 1194.1-1991, IEEE Standard for Electrical Characteristics of Backplane Transceiver Logic (BTL) Interface Circuits.

ISO/IEC 10857:1994 (Incorporates IEEE Std 896.1-1991 and IEEE Std 896.1a-1993), Information technology—Microprocessor Systems—Futurebus+™—Logical Protocol Specification.⁵

ISO/IEC 13213: 1994 (IEEE Std 1212 1994 Edition), Information technology—Microprocessor systems—Control and Status Registers (CSR) Architecture for microcomputer buses.

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

³This authorized standards project was not approved by the IEEE Standards Board at the time this went to press. It is available from the IEEE Service Center.

⁴This authorized standards project was not approved by the IEEE Standards Board at the time this went to press. It is available from the IEEE Service Center.

⁵ISO publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse. ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA. IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

3. Definitions

3.1 Special word usage

3.1.1 may: A key word that indicates flexibility of choice with no implied preference.

3.1.2 shall: A key word that indicates a mandatory requirement. Designers must implement all such mandatory requirements to ensure interoperability with Futurebus+ conforming products and to claim conformance to this standard.

3.1.3 should: A key word that indicates flexibility of choice with a strongly preferred implementation. The phrase *it is recommended* is used interchangeably with the key word *should*.

3.2 Definitions for Futurebus+ systems

3.2.1 board: A device connected to a bus. Usually constructed from a printed circuit board. Also referred to as a module.

3.2.2 control and status register (CSR): A memory-mapped register that is accessed through read and write transactions and is used to observe the state of a node or to control its operation.

3.2.3 CSR: Acronym for **control and status register** (*See: 3.2.2*).

3.2.4 edge-sensitive signal: Signals whose leading and/or trailing edges are used to strobe information contained on level sensitive signals.

3.2.5 level-sensitive signal: Signals whose high or low state is sampled based on the leading and/or trailing edges of edge-sensitive strobe signals.

3.2.6 node_id: A 16 b number that defines the initial node address space. During system initialization, system-wide unique node_id values may be assigned to nodes within a tightly coupled system.

3.2.7 primary bus: The collection of signals that provides the system with the basic mechanism for exchanging data between boards.

3.2.8 secondary bus: A collection of signals that provides the system with an alternate mechanism for exchanging data between boards as a means to recover from faults in the primary bus.

3.3 Definitions for fault tolerant systems

3.3.1 availability: The fraction of time that the system is actually capable of performing its mission.

3.3.2 benign failure: Failure whose penalties are of the same order of magnitude as the benefit provided by correct service delivery.

3.3.3 compensation, error: Form of error processing when the erroneous state contains enough redundancy to enable correct service delivery.

3.3.4 coverage: Measure of the representative nature of situations to which a system is submitted during its validation compared to the actual situations it will be confronted with during its operational life.

3.3.5 detection, error: The action of identifying that a system state is erroneous.

3.3.6 detected error: Error recognized as such by a detection algorithm or mechanism.

3.3.7 diagnosis, fault: The action of determining the cause of an error in location and nature.

3.3.8 error: Manifestation of a failure in a system.

3.3.9 error compensation: *See: compensation, error* (3.3.3).

3.3.10 error detection: *See: detection, error* (3.3.5).

3.3.11 error recovery: *See: recovery, error* (3.3.26).

3.3.12 error report: A summary, either in full or in part, of the error log.

3.3.13 fault diagnosis: *See: diagnosis, fault* (3.3.7).

3.3.14 fault removal: *See: removal, fault* (3.3.28).

3.3.15 fault tolerance: The ability of a system or a component to continue normal operation despite the presence of hardware or software faults.

3.3.16 Fail-Operate system: A system that can operate in the presence of faults.

3.3.17 Fail-Safe system: A system whose failures can only be, or are to an acceptable extent, benign failures.

3.3.18 Fail-Stop system: A system whose failures can only be, or are to an acceptable extent, stopping failures.

3.3.19 failure: An event in which a system or system component does not perform a required function within specified limits. A failure may be produced when a fault is encountered.

3.3.20 fault: A defect in a hardware device or component; for example, a short circuit or broken wire.

3.3.21 fault masking: *See: masking, fault* (3.3.25).

3.3.22 fault tolerance: *See: tolerance, fault* (3.3.38).

3.3.23 integrity: The condition of being unimpaired.

3.3.24 intermittent fault: A temporary or unpredictable fault in a component.

3.3.25 masking, fault: The result of applying error compensation systematically, even in the absence of error.

3.3.26 recovery, error: Form of error processing where an error-free state is substituted for an erroneous state.

3.3.27 reliability: The ability of a system or component to perform its required functions under stated conditions for a specified period of time.

3.3.28 removal, fault: Methods and techniques aimed at reducing the presence (number, seriousness) of faults.

3.3.29 service: System behavior as perceived by the system user.

3.3.30 single point of failure: With respect to a system, a failure that would result in the inability of that system to perform its intended function.

3.3.31 specification, system: Agreed description of the requirements of the system.

3.3.32 state, system: The values assumed at a given instant by the variables that define the characteristics of a system, component, or simulation.

3.3.33 system: A set of interconnected boards that achieve a specified objective by performing designated functions.

3.3.34 system failure: Malfunctions in the hardware and software that could compromise the security of the system (for example, non-security-related failures and design flaws are not considered). The malfunctions include both intentional and inadvertent design or implementation flaws (including malicious hardware and software) and component failures. For intentional attacks, this threat area assumes that an intruder has access to the design or implementation processes of the system or to the operational system in such a way as to be able to cause a failure in a component. For inadvertent attacks, there may not be a specific intruder.

3.3.35 system specification: *See: specification, system* (3.3.31).

3.3.36 system state: *See: state, system* (3.3.32).

3.3.37 testing: Dynamic verification performed with valued inputs.

3.3.38 tolerance, fault: Methods and techniques aimed at providing a service complying with the specification in spite of faults.

3.3.39 transient: A fault or error resulting from temporary environmental conditions.

3.3.40 validation: Evaluating a system or component during or at the end of the development process to determine whether it satisfies specific requirements.

3.3.41 verification: Evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Contrast with: validation.

3.4 Syntax

3.4.1 CSR naming convention

Fields within the CSRs are denoted in uppercase, bold notation using the following convention:

OWNING_MODULE.CSR_NAME.CSR_FIELD

The optional **OWNING_MODULE** attributes are master (if the master attribute is true), slave (if the master attribute is false), or module (if the master attribute is irrelevant in the context). **CSR_NAME** is the name of the referenced CSR register. **CSR_FIELD** is a contiguous series of bits defined within the CSR register. Symbols used in equations obey the syntax and precedence noted in subclause 2.4 of ISO/IEC 10857: 1994. In addition, the change in the state of fields within CSRs is noted with the following convention:

CSR_NAME.CSR_FIELD <- new-state

Where the left-hand side of the expression denotes the field that shall be updated with the right-hand side of the expression as a result of execution of the expression.

3.4.2 Numbering conventions

- Hexadecimal numbers in this standard are represented by the notation 0x####, where each “#” is integer from 0–9 or a letter from A–F.
- Binary numbers in this standard are represented by a sequence of 1s and 0s with subscript 2 (e.g., $10 = 0 \times A = 1010_2$).
- All other numbers in the standard are decimal numbers.
- Superscript (e.g., 2^{32}) indicates “to the power of.”
- All register names and bit names are uppercased.

3.4.3 CSR access types

- RO (Read-Only): A CSR location from which data can only be read.
- R/W (Read or Write): A CSR location from which data can either be read or written.
- WO (Write-Only): A CSR location to which data can only be written.
- R/W 1C (Read/Write Ones to Clear): A CSR location from which data can be written to clear specific bits in a register. For example, if a write one to clear register contained 0xFFFFFFFF and the value 0x00800000 was written into it, the contents of the register would become 0xFF7FFFFF.

4. Error detection

4.1 Introduction

A fault tolerant system is a system that, through a combination of hardware, software, and system design, can take appropriate actions in the event of an error. Following an error, a fault tolerant system would continue to provide an acceptable level of service until the fault that caused the error disappears, the error is corrected, or the service provided is no longer needed. For the purposes of this standard, an acceptable level of service implies one of the following system definitions: systems that protect the system state information from corruption, systems that prevent catastrophic consequences due to an error, and systems that have to continue to operate in the presence of errors. (Note that the first system definition is a subset of the second and the second is a subset of the third. If one were to implement a system that operates continuously in the presence of errors, one would also have a system that prevents catastrophic consequences and that protects system state information. A system designer does not, however, need to implement this

type of system if all that is required is to keep all system state information from being corrupted.) This standard uses these system definitions as a guide for specifying a minimum set of capabilities required to implement a fault tolerant system using Futurebus+ backplane(s).

The base Futurebus+ standards (see ISO/IEC 10857: 1994, IEEE Std 896.2-1991, and IEEE Std 896.2a 1993) include many fault tolerant facilities: byte parity, a well-defined protocol that can be monitored for divergence from the standard, error symptom logging, a diagnostic facility, retry, and live insertion. (Live insertion does not contribute to fault tolerance. It is included here because it contributes to increased availability and continuous operation.)

While the fault tolerant facilities of Futurebus+ are superior to other commercial buses, these standards do not provide all that is needed to implement a fault tolerant Futurebus+ backplane. Fault tolerant systems require the backplane to meet the following criteria:

- a) Information has to be communicated between modules in a timely manner.
- b) All single errors that may occur on the backplane (including the bus transceivers and controllers) have to be detectable and reportable.
- c) An alternate communications path has to be provided (in many cases).
- d) Redundancy in some form has to be present to recover.

Not all Futurebus+ signals defined in ISO/IEC 10857: 1994 are protected. Because multiple modules can simultaneously assert bits in the status and capability fields, these signals cannot be parity protected. In addition, the synchronization signals are difficult to protect. This clause describes facilities added to verify the integrity of the status, capability, geographical address, reset, and synchronization signals in Futurebus+ as well as reviewing how the facilities in the base standards can be used to provide full fault tolerance. With these extensions, Futurebus+ permits the design of fail-stop modules and repair or masking via retry or spare modules. If repair is unsuccessful, information is made available to permit operating system or operator repair or masking. Clause 5 specifies the error handling procedures. Clause 6 specifies the error reporting mechanism while clause 7 specifies error recovery procedures.

Futurebus+ is composed of six groups of signals: information, synchronization, arbitrated message, reset, central arbitration, and geographical address. The signals in the various groups are given in table 4-1. For many of the tables in this standard, a description of all signals except those marked with an asterisk (*) are given in ISO/IEC 10857: 1994. The signals marked with a cross (†) are specified in this standard. Note that five additional signals are required for fault tolerant operation independent of the bus width.

Table 4-1—Signals for fault tolerant extensions

Data path width		32 b	64 b	128 b	256 b
Information Lines					
AD[63..0]*	Address/Data	32	64	64	64
D[255..64]*	Data	-	-	64	192
BP[31..0]*	Bus Parity	4	8	16	32
TG[7..0]*	TaG	8	8	8	8
TP*	Tag Parity	1	1	1	1
CM[7..0]*	CoMmand	8	8	8	8
CP*	Command Parity	1	1	1	1
ST[7..0]*	STatus	8	8	8	8
CA[2..0]*	CApability	3	3	3	3
RF*†	ReFlected	1	1	1	1
TED*†	Transaction Error Duplicate	1	1	1	1
SLD*†	SeLect Duplicate	1	1	1	1
Synchronization Lines					
AS*,AK*,AI*	Address handshake	3	3	3	3
DS*,DK*,DI*	Data handshake	3	3	3	3
ET*	End of Tenure	1	1	1	1
Arbitrated Message Lines					
AB[7..0]*	Arbitrated Message Bus	8	8	8	8
ABP*	Arbitrated Message Bus Parity	1	1	1	1
AP*,AQ*,AR*	Arbitrated Message Sync	3	3	3	3
AC[1..0]*	Arbitrated Message Condition	2	2	2	2
Reset/Bus Initialize Line					
RE*	REset	1	1	1	1
RED*†	REset Duplicate	1	1	1	1
Central Arbitration					
RQ[1..0]*	ReQuest	2	2	2	2
GR*	GRant	1	1	1	1
PE*	PreEmption	1	1	1	1
Geographical Address					
GA[4..0]*	Geographical Address	5	5	5	5
GAP*†	Geographical Address Parity	1	1	1	1
Total active signals		101	137	209	353

NOTE — A description of all signals except those marked with an asterisk (*) are given in ISO/IEC 10857: 1994. The signals marked with a cross (†) are specified in this standard.

4.2 Fault detection mechanisms

From another point of view, the Futurebus+ standard defines two types of signals: level sensitive and edge sensitive. The information lines, the arbitrated message bus, and the geographical address lines are the examples of level-sensitive signals in ISO/IEC 10857: 1994 and IEEE Std 896.2-1991. Redundant information has to be provided to protect the level-sensitive signals since there is usually no relationship between the signals. In Futurebus+, level-sensitive signals are typically protected by parity. The status and capability signals are level sensitive but cannot be protected by parity. Duplication and reflection are introduced in this standard to protect these lines.

Examples of edge-sensitive signals are the synchronization and arbitrated message synchronization lines. Edge-sensitive signals need not be protected by redundancy techniques such as parity since they are tightly coupled in the synchronization handshakes and can be either directly or indirectly monitored for their agreement with the protocol. Since Futurebus+ is an asynchronous bus, timeouts are used to detect situations where a signal never arrives.

4.2.1 Parity

For compelled transfers, level-sensitive signals are sampled only when a synchronization signal indicates they are valid. The first four of the following groups of signals are specified in ISO/IEC 10857: 1994 as protected by parity. The fifth group shall be protected by the parity bit specified in this standard.

- a) **AD[]*** and **D[]*** signals are protected by **BP[]***.
- b) **TG[]*** signals are protected by **TP***.
- c) **CM[]*** signals are protected by **CP***.
- d) **AB[]*** signals are protected by **ABP***.
- e) **GA[]*** signals are protected by **GAP*[†]**.

For packet mode transfers, the **AD[]***, **D[]***, and **TG[]*** signals are sampled by synchronization signals derived from each signal.

4.2.2 Reflected protection

There are two main groups of level-sensitive signals that are not protected by parity: status (**ST[]***) and capability (**CA[]***). During a transaction, the values that the master samples on the status and capability signals shall be “reflected” back to all the slaves on later phases of the transaction. The status and capability signal values sampled during the connection phase and data phases shall be reflected back to the slaves on the parity-protected command signals. The **RF*** signal shall be protected by the signal it reflects in the subsequent phase. By doing so, the slaves can verify those status and capability signals that they asserted have been correctly observed by the master. Reflection shall be used to protect the status and capability signals indicated in table 4-2.

Should any of the status or capability signals be falsely asserted by the master, the false assertion will be detected as indicated in table 4-3.

4.2.3 Duplicated protection

Reflection cannot cover all fault cases for the transaction error **TE*** (**ST0***) and the selected **SL*** (**ST2***) status signals. Since **TE*** is asserted during the disconnection phase of a transaction, it cannot be reflected in the same transaction. Because **SL*** is asserted during every transaction with a valid address, reflection will not detect the fault where **SL*** is always asserted. As a result, these signals plus the reset (**RE***) signal shall be duplicated (**TED*[†]**, **SLD*[†]**, and **RED*[†]** respectively). Both the duplicated signal and the signal it is protecting shall be asserted and released in unison for these three pairs. Since **TE*** is asserted by the disconnection phase when **BE*** is asserted, **BE*** is effectively duplicated by **TE***. When the protected signal and its duplicate do not agree, a fault shall be indicated.

Table 4-2—Reflected signals

Status/capability signal	Command phase in which signal is reflected		
	Connection	Data	Disconnection
ST1* (BS/ED*)	—	—	CM2*
ST3* (BC*)	—	CM3*	—
ST4* (TF*)	—	LT*¹	CM0*
ST5* (IV*)	—	CM4* (SW*)	CM4* (SW*)
ST7* (WT*)	—	—	CM3*
CA0* (TS*)	—	CM2*	—
CA1* (CO*)	—	RF*[†]	—
CA2* (SR*)	—	—	RF*[†]

¹Reflected during multipacket mode transactions only.

Table 4-3—Status/Capability false assertion

False assertion	Detection mechanism
BS*/ED*	Benign
BC*	Benign
TF*	Benign
IV*	Timeout
WT*	Timeout
TS*	Shall be detected by slaves since signal asserted during entire transaction
CO*	Shall be detected by slaves since signal asserted during the entire transaction
SR*	Shall be detected by slaves since signal asserted during entire transaction

4.2.4 Protocol protection

Modules shall detect protocol violations by checking the state of certain synchronization signals at the transition time of other synchronization signals. The state of the monitored signals can be verified by comparing them at the appropriate times to the state of the particular signals that the module controls. This technique is effective with both the parallel and the arbitration protocols.

4.2.4.1 Parallel protocol handshake protection

Modules shall observe the state of parallel protocol signals at specified times. Table 4-4 specifies the parallel protocol violations that shall be detected when the indicated signal edge occurs.

Table 4-4—Parallel protocol handshake protection

Module protocol violation	Detection condition
$\sim\text{as}^* \mid \sim\text{AS}^* \mid \sim\text{AK}^* \mid \sim\text{AIf} \mid \text{DS}^{*1}$	ai* released
$\sim\text{as}^* \mid \sim\text{AS}^*$	ak* assertion
$\text{as}^* \mid \text{AS}^*$	ai* assertion
$\text{as}^* \mid \text{AS}^* \mid \sim\text{AI}^* \mid \text{AKf}^{*2}$	ak* released
$\sim\text{ai}^* \mid \sim\text{AI}^* \mid \text{ak}^* \mid \text{AK}^* \mid \text{AS}^{*3}$	as* assertion
$\sim\text{ak}^* \mid \sim\text{AK}^* \mid \text{ai}^* \mid \text{AI}^* \mid \sim\text{AS}^{*4}$	as* released
$\text{ds}^* \mid \text{DS}^* \mid \text{dk}^* \mid \text{DK}^* \mid \text{di}^* \mid \text{DI}^*$	as* assertion
$((\sim\text{ds}^* \mid \sim\text{DS}^*) \& \text{AS}^*) \mid \sim\text{DI}^{*5}$	di* released
$\sim\text{ds}^* \mid \sim\text{DS}^*$	dk* assertion
$\text{ds}^* \mid \text{DS}^*$	di* assertion
$((\text{ds}^* \mid \text{DS}^*) \& \text{AS}^*) \mid \sim\text{DI}^* \mid \sim\text{DKf}^{*6}$	dk* released
$\sim\text{di}^* \mid \sim\text{DI}^* \mid \text{dk}^* \mid \text{DK}^* \mid \text{DS}^* \mid \sim\text{AS}^* \mid \sim\text{AK}^* \mid \text{AI}^{*7}$	ds* assertion
$\sim\text{dk}^* \mid \sim\text{DK}^* \mid \text{di}^* \mid \text{DI}^* \mid \sim\text{DS}^{*8}$	ds* released

1At **ai*** release, **AIf** is required to be asserted within the implementation. Including $\sim\text{AK}^*$ allows slaves that are the first in their release of **ai*** to verify their own assertion of **AK***. Including **DS*** ensures that the early release of **AI*** by a slave will be detected if either $\sim\text{AS}^*$ (indicating an address-only transaction and release of **AS*** by a master) or **DS*** (indicating a data phase) is asserted. The implementation has to account for setup/hold time for the sampled signals. This enables self-checking of the sampled signals by the implementation.

2At **ak*** release, **AKf*** is required to be asserted within the implementation. Including $\sim\text{AI}^*$ allows slaves that are the first in their release of **ai*** to verify their own assertion of **AI***. The implementation has to account for setup/hold time for the sampled signals. This enables self-checking of the sampled signals by the implementation.

3At **as*** assertion, **AS*** is required to be released within the implementation. The implementation has to account for setup/hold time for the sampled signals. This enables self-checking of the sampled signals by the implementation.

4At **as*** release, **AS*** is required to be asserted within the implementation. The implementation has to account for setup/hold time for the sampled signals. This enables self-checking of the sampled signals by the implementation.

5At **di*** release, **DI*** is required to be asserted within the implementation. The implementation has to account for setup/hold time for the sampled signals. This enables self-checking of the sampled signals by the implementation.

6At **dk*** release, **DKf*** is required to be asserted within the implementation. The implementation has to account for setup/hold time for the sampled signals. This enables self-checking of the sampled signals by the implementation.

7At **ds*** assertion, **DS*** is required to be released within the implementation. The implementation has to account for setup/hold time for the sampled signals. This enables self-checking of the sampled signals by the implementation.

8At **ds*** release, **DS*** is required to be asserted within the implementation. The implementation has to account for setup/hold time for the sampled signals. This enables self-checking of the sampled signals by the implementation.

4.2.4.2 Bus tenure protocol protection

Modules shall observe the state of the bus tenure signals at specified times. Table 4-5 specifies the bus tenure violations that shall be detected when the indicated signal edge occurs.

Table 4-5—Bus tenure protocol protection

Module protocol violation	When
REQUEST et ¹	GRANT released
~REQUEST ²	GRANT assertion
~GRANT	REQUEST released
(~ET * and ~GRANT) (~ET * and ~REQUEST and GRANT)	as * assertion
~ET *	as * released
PE *	grant released

¹Where **REQUEST** indicates that a local module is requesting bus tenure, **REQUEST** would correspond to the central arbitration signals (**rq0*** | **rq1***) or may represent request indicators to a local distributed arbitration controller (as long as the local distributed arbiter follows a request/grant protocol similar to the central arbitration signals). **GRANT** is an indication that the local module has been granted bus tenure (pending **ET*** release). **GRANT** would correspond to the active high version of the **GR*** signal or the grant indicator from a local distributed arbiter.

²The module would benefit from checking that its **REQUEST** was asserted.

The central arbiter shall observe the state of the central arbitration signals at specified times. Table 4-6 specifies violations that shall be detected by the central arbiter when the indicated edge occurs.

Table 4-6—Central arbiter protocol protection

Central arbiter violation	When
~REQUEST ¹	GRANT assertion
REQUEST	GRANT released
More than one GRANT asserted ²	Any Grant asserted

¹In this situation, **REQUEST** refers to the request lines (e.g., **RQ0**–**1*** and **RQ1**–**1***) from the particular module for which the **GRANT** (e.g., **GR**–**1***) line applies.

²The central arbiter has to have only one **GRANT** signal (i.e., **GR**–**1*** XOR **GR**–**2*** XOR...**GR**–**n***) asserted when a grant is issued (i.e., **GR**–**1*** | **GR**–**2*** |...| **GR**–**n***).

Modules shall observe the state of the arbitrated message signals at specified times. Table 4-7 specifies violations that shall be detected by each module when the indicated edge occurs.

Table 4-7—Arbitrated message protocol protection

Module protocol violation	When
AQ *	ap * assertion
~AQ *	ap * released
AR *	aq * assertion
~AR *	aq * released
AP *	ar * assertion
~AP *	ar * released

4.2.5 Timeout protection

While providing for coverage of a unique set of fault types, timer expiration provides a level of protection that supplements coverage at lower levels. Table 4-8 describes functions that shall be protected, the specific CSR timeout registers, and where the CSR registers are defined.

Table 4-8—Timeout protection

Protected function	Error	Definition
Arbitration	ARB_TIMEOUT_ERROR	ERROR_LO (ISO/IEC 10857 : 1994)
Transaction	TRANSACTION_TIMEOUT	ERROR_HI (ISO/IEC 10857 : 1994)
Reset	RESET_TIMEOUT	ERROR_LO (see 8.5.1)
Split Transaction	SPLIT_TIMEOUT	ERROR_HI (ISO/IEC 10857 : 1994)
Request Tenure	REQ_TENURE_TIMEOUT	ERROR_LO (see 8.5.1)
Arbitration Message	ARB_MSG_TIMEOUT	ERROR_LO (see 8.5.1)

5. Error handling

5.1 Introduction

The following subclauses provide a specification of the errors detected and the resulting error reporting, recovery actions, retry strategies, and containment strategies.

5.1.1 Error subclass

Error subclasses are unique combinations of fault detection mechanisms, detection phases, protected signals, error logging, and recovery strategies.

5.1.2 Detection mechanism

The detection mechanism refers to those major forms of fault detection covered in clause 4.

5.1.3 Detection phase

The detection phase is a reference to the time period in which the fault is detected (but not necessarily logged and recovered). This can be connection, data and disconnection for parallel bus tenure, arbitration phase and pass for arbitration competition, and the all-encompassing “any” category for those events that are not bound to any particular sequence (e.g., reset).

5.1.4 Protected signals

Protected signals are those bus lines that are covered by this error subclass.

5.1.5 Error log snapshot

This refers to those error register signals that are set, cleared, or frozen copies of the states of other register fields that do not themselves freeze as a result of this error subclass. Groups of signals may be referred to by a group name. See 6.2 for a detailed specification of these snapshot groups.

5.1.6 Bus error signal

This describes the signal actively asserted by the detecting module on the bus to indicate that an error state has been entered. Groups of signals may be referred to by a group name. See 6.3 for a detailed specification of these error signal groups.

5.1.7 Recovery strategy

This describes how the detecting module should egress from any transaction in progress. A recovery action may be referred to by a name. See 7.1 for a detailed specification of these named recovery actions.

5.1.8 Retry strategy

For certain error subclasses that may be transient in nature, this describes how the transaction master may be allowed to retry the transaction in the expectation of successful termination. A strategy may be referred to by a name. See 7.3 for a detailed specification of these named strategies.

5.1.9 Containment strategy

For some fault tolerant system models, this subclause describes the recommended procedure by which the module may limit further participation. This is in the expectation that further participation could aggravate the fault state while module isolation can minimize fault propagation potential. A strategy may be referred to by a name. See 7.4 for a detailed specification of these named strategies.

5.2 Error descriptions

The following subclauses contain a detailed description of the error classes that this standard shall detect (parity, reflection, duplication, protocol, and timeout). The local error signal (Local_Report) shall be actively asserted by the detecting module (see 7.5) on the local node level to indicate that an error state has been entered (except when noted otherwise).

If the **FTE_ENABLE** bit of the **MAINT_CONTROL** CSR is set, then a module shall perform the following specified error detection checks. If the **FTE_ENABLE** bit is cleared, then a module shall not perform any error checks beyond those called for in the base standards (see 1.4). Independent of the state of **FTE_ENABLE**, all fault tolerant extension (FTE)-capable modules (i.e., **FTE_CAP** bit is set in the **NODE_CAPABILITIES_EXT** CSR) shall generate the reflected status/capability information and the duplicated signals. In addition, independent of the state of **FTE_ENABLE**, all FTE-capable modules shall perform the geographical address parity error detection during module power-up/reset. Note that, this implies that FTE-capable modules can only be used in a FTE-capable system backplane. However, by leaving **FTE_ENABLE** cleared within all modules, FTE-capable modules and modules that are not FTE capable may interoperate within a FTE-capable system backplane.

5.2.1 Parity detection class

The general rule is that modules only check the parity of lines with which they are concerned. During the connection phase, all aligned modules shall check parity on the **AD[]***, **CM[]***, and (optionally) **TG[]*** lines. During the data and disconnection phase, only participating modules shall check parity on these lines and **D[]***.

The parity detection class contains the subclasses discussed in the following subclauses.

5.2.1.1 Address parity error

<i>Description:</i>	This error shall be detected by any module decoding the address signals and the AW* command signal after AS* (address synchronization) has been asserted. Parity is checked on the active byte lanes (as qualified by AW*). Modules detect this error if incorrect parity is detected during the connection phase on the AD[]* lines.
<i>Detection mechanism:</i>	Vertical parity verification
<i>Detection phase:</i>	Connection
<i>Protected signal(s):</i>	AD[]* and BP[]* (as qualified by AW*) Specific lane protection is as specified in ISO/IEC 10857 :1994
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_HI.DATA/ADDR_PAR_ERROR
<i>Bus error signal:</i>	Slave_Assertion_Connection_Phase
<i>Recovery strategy:</i>	Slave_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.1.2 Data parity error

5.2.1.2.1 Read data parity error

<i>Description:</i>	The master shall detect a data parity error if incorrect parity is decoded during the data phase on the AD[]* or D[]* lines (as qualified by DW*).
<i>Detection mechanism:</i>	Vertical parity verification
<i>Detection phase:</i>	Data
<i>Protected signal(s):</i>	AD[]* , BP[]* and D[]* (as qualified by DW*)
<i>Error log snapshot:</i>	Master_Snap_Group MASTER.ERROR_HI.DATA/ADDR_PAR_ERROR
<i>Bus error signal:</i>	Master_Assertion_Group
<i>Recovery strategy:</i>	Master_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	Master_Retry
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.1.2.2 Write data parity error

<i>Description:</i>	A participating slave shall detect a data parity error if incorrect parity is decoded during the data phase on the AD[]* or D[]* lines (as qualified by DW*).
<i>Detection mechanism:</i>	Vertical parity verification
<i>Detection phase:</i>	Data
<i>Protected signal(s):</i>	AD[]* , BP[]* , and D[]* (as qualified by DW*) Specific lane protection is as specified in ISO/IEC 10857 :1994
<i>Error log snapshot:</i>	Slave_Snap_Group SLAVE.ERROR_HI.DATA/ADDR_PAR_ERROR

<i>Bus error signal:</i>	Slave_Assertion_Data_Phase
<i>Recovery strategy:</i>	Slave_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.1.2.3 Disconnect data parity error

<i>Description:</i>	Participating slaves shall detect a data parity error if incorrect parity is decoded during the disconnection phase on the AD[]* lines when those lines are activated for specified information. No parity checking on disconnection data should be done on connected read transactions because the state on the data lines of the bus is undefined in the disconnection phase following a connected read.
<i>Detection mechanism:</i>	Vertical parity verification
<i>Detection phase:</i>	Disconnection
<i>Protected signal(s):</i>	AD[]* and BP[]* Specific lane protection is as specified in ISO/IEC 10857: 1994
<i>Error log snapshot:</i>	Slave_Snap_Group SLAVE.ERROR_HI.DATA/ADDR_PAR_ERROR
<i>Bus error signal:</i>	Slave_Assertion_Disconnection_Phase
<i>Recovery strategy:</i>	Slave_Recovery_Disconnect_Phase
<i>Retry strategy:</i>	None
<i>Containment strategy:</i>	None

5.2.1.3 Command parity error

5.2.1.3.1 Connection command parity error

<i>Description:</i>	This error shall be detected by any module after AS* (address synchronization) has been asserted in the connection phase.
<i>Detection mechanism:</i>	Vertical parity verification
<i>Detection phase:</i>	Connection
<i>Protected signal(s):</i>	CM[]* and CP[]*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_HI.COMMAND_PAR_ERROR SLAVE.FTE_HOLD <- CP*
<i>Bus error signal:</i>	Slave_Assertion_Connection_Phase
<i>Recovery strategy:</i>	Slave_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.1.3.2 Data command parity error

<i>Description:</i>	Participating slaves shall detect a command parity error if incorrect parity is decoded during the data phase on the CM[]* lines. Even though the command is only used during the first data beat of a transaction, the parity on this field has to be correct for every data beat.
<i>Detection mechanism:</i>	Vertical parity verification
<i>Detection phase:</i>	Data
<i>Protected signal(s):</i>	CM[]* and CP[]*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_HI.COMMAND_PAR_ERROR SLAVE.FTE_HOLD <- CP*
<i>Bus error signal:</i>	Slave_Assertion_Data_Phase
<i>Recovery strategy:</i>	Slave_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.1.3.3 Disconnect command parity error

<i>Description:</i>	Participating slaves shall detect a command parity error if incorrect parity is decoded during the disconnect phase on the CM[]* lines.
<i>Detection mechanism:</i>	Vertical parity verification
<i>Detection phase:</i>	Disconnection
<i>Protected signal(s):</i>	CM[]* and CP[]*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_HI.COMMAND_PAR_ERROR SLAVE.FTE_HOLD <- CP*
<i>Bus error signal:</i>	Slave_Assertion_Disconnection_Phase
<i>Recovery strategy:</i>	Slave_Recovery_Disconnect_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.1.4 Tag parity error

5.2.1.4.1 Connection tag parity error

<i>Description:</i>	This error shall be detected by any module decoding the tag signals after AS* (address synchronization) has been asserted. Modules detect this error if incorrect parity is detected during the connection phase on the TG[]* lines and these lines are activated.
<i>Detection mechanism:</i>	Vertical parity verification
<i>Detection phase:</i>	Connection
<i>Protected signal(s):</i>	TG[]* and TP*

Error log snapshot: Slave_Snap_Group(Master_Not_Principal)
SLAVE.ERROR_LO.TAG_PARITY_ERROR
SLAVE.TAG_HOLD.TAG <- TG[]* (connection phase)
SLAVE.TAG_HOLD.TAG_PARITY <- TP* (connection phase)

Bus error signal: Slave_Assertion_Connection_Phase

Recovery strategy: Slave_Recovery_Connect/Data_Phase

Retry strategy: None, discretion of master

Containment strategy: Module_Containment_Transaction_Error

5.2.1.4.2 Data read tag parity error

Description: The master shall detect this error if incorrect parity is detected during the data phase on the **TG[]*** lines and these lines are activated.

Detection mechanism: Vertical parity verification

Detection phase: Data

Protected signal(s): **TG[]*** and **TP***

Error log snapshot: Master_Snap_Group(Not_Principal)
MASTER.ERROR_LO.TAG_PARITY_ERROR
MASTER.TAG_HOLD.TAG <- TG[]* (data phase)
MASTER.TAG_HOLD.TAG_PARITY <- TP* (data phase)

Bus error signal: None

Recovery strategy: Master_Recovery_Connect/Data_Phase

Retry strategy: Master_Retry

Containment strategy: Module_Containment_Transaction_Error

5.2.1.4.3 Data write tag parity error

Description: Participating slaves shall detect this error if incorrect parity is detected during the data phase on the **TG[]*** lines and these lines are activated.

Detection mechanism: Vertical parity verification

Detection phase: data

Protected signal(s): **TG[]*** and **TP[]***

Error log snapshot: Slave_Snap_Group(Master_Not_Principal)
SLAVE.ERROR_LO.TAG_PARITY_ERROR
SLAVE.TAG_HOLD.TAG <- TG[]* (data phase)
SLAVE.TAG_HOLD.TAG_PARITY <- TP* (data phase)

Bus error signal: Slave_Assertion_Data_Phase

Recovery strategy: Slave_Recovery_Connect/Data_Phase

Retry strategy: None, discretion of master

Containment strategy: Module_Containment_Transaction_Error

5.2.1.4.4 Disconnect tag parity error

<i>Description:</i>	Participating slaves shall detect this error if incorrect parity is detected during the disconnect phase on the TG[]* lines and these lines are activated.
<i>Detection mechanism:</i>	Vertical parity verification
<i>Detection phase:</i>	Disconnection
<i>Protected signal(s):</i>	TG[]* and TP[]*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_LO.TAG_PARITY_ERROR SLAVE.TAG_HOLD.TAG <- TG[]* (disconnect phase) SLAVE.TAG_HOLD.TAG_PARITY <- TP* (disconnect phase)
<i>Bus error signal:</i>	Slave_Assertion_Disconnection_Phase
<i>Recovery strategy:</i>	Slave_Recovery_Disconnect_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.1.5 Geographical address parity

5.2.1.5.1 Reset geographical address parity error

<i>Description:</i>	The geographical address lines, GA[]* , are not protected. Since the GA[]* pins are exposed, a careless removal or insertion could damage the pins such that two modules would appear to have the same geographical address. An even parity bit, GAP* , shall be defined that protects GA[]* .
<i>Detection mechanism:</i>	Vertical parity verification of even parity
<i>Detection Phase:</i>	Any reset operation that transfers the backplane signals asserted by GA[]* to the NODE_ID CSR. (Defined as POWER_UP or SYS_RESET in subclause 7.2.1 of ISO/IEC 10857: 1994).
<i>Protected signal(s):</i>	GA[]* and GAP*
<i>Error log snapshot</i>	Slave_Snap_Protocol SLAVE.ERROR_LO.GA_PAR_ERROR SLAVE.NODE_ID.GEOGRAPHICAL_ADDRESS <-GA[]* SLAVE.FTE_HOLD.GA_PAR <- GAP* NOTE — The last two operations are performed for any reset operation, not only those that incur reset geographical address parity error.
<i>Bus error signal:</i>	None
<i>Recovery strategy:</i>	None
<i>Retry strategy:</i>	None
<i>Containment strategy:</i>	Module_Containment_Geographic_Error

5.2.1.6 Arbitration bus parity

5.2.1.6.1 Arbitrated message parity

<i>Description:</i>	The arbitrated message bus signals, AB[]* , shall be protected by a parity bit, ABP* . During Phase 3 arbitration, all modules latch the value of the winning competition number from the bus and check its parity. If a module detects a parity error, it asserts AC0* and AC1* .
<i>Detection mechanism:</i>	Vertical parity verification of even parity
<i>Detection phase:</i>	Arbitration phases 3–5.
<i>Protected signal(s):</i>	AB[]* and ABP*
<i>Error log snapshot:</i>	Snapshot shall occur during Phase 5 (before AQ* release). Slave_Snap_Protocol SLAVE.ERROR_LO.ARB_COMPARISON_OR_PARITY_ERROR SLAVE.FTE_HOLD.ARB_PASS <- 0 (if Pass 2 competition) SLAVE.FTE_HOLD.ARB_PASS <- 1 (if Pass 1 competition) SLAVE.FTE_HOLD.ARB_PARITY <- ABP* (from Pass 2) SLAVE.FTE_HOLD.ARB_NUMBER <- AB[]* (from Pass 2)
<i>Bus error signal:</i>	AC0* and AC1* asserted during Phase 3 and sustained until next Phase 0 No error signal shall be asserted on the ST[]* lines
<i>Recovery strategy:</i>	AC0* and AC1* are sustained through remaining competition phases. The transaction in progress on the AD[]* bus shall proceed to completion
<i>Retry strategy:</i>	None (undefined)
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.2 Reflection detection class

As noted elsewhere, status (**ST[]***) and capability (**CA[]***) lines are level-sensitive signals that are not protected by parity. Faults on these lines may or may not be detectable in the wide range of protocol errors covered elsewhere. Instead, the state of the signal is reflected back (effectively a form of temporal redundancy) in a later phase for validation by the originator.

Note that the error cases detected with duplicate lines (e.g. **SLD***, **TED***) and reflected fields (e.g. **CO***, **SR***) are required to capture valid states of **SLD***, **TED***, and **RF*** to allow valid observation of these fields and subsequent determination of the fault type while determining the cause of the **STATUS/CAPABILITY_ERROR**. The reflection detection class contains the subclasses discussed in the following subclasses.

5.2.2.1 Busy error

<i>Description:</i>	The busy signal (BS*) is received by the current master during the connection phase on ST1* and is reflected on CM2* during the disconnection phase. A slave that asserts BS* shall verify its assertion by monitoring CM2* .
<i>Detection mechanism:</i>	Busy verification using reflection
<i>Detection phase:</i>	Disconnection
<i>Protected signal(s):</i>	ST1* and CM2*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_LO.STATUS/CAPABILITY_ERROR SLAVE.FTE_HOLD.TED <- TED* SLAVE.FTE_HOLD.SLD <- SLD* SLAVE.FTE_HOLD.RF <- RF*

<i>Bus error signal:</i>	Slave_Assertion_Reflection_Error
<i>Recovery strategy:</i>	Slave_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.2.2 End of data error

<i>Description:</i>	The end of data signal (ED*) is received by the current master during the data phase on ST1* and is reflected on CM2* during the disconnection phase. A slave that asserts ED* shall verify its assertion by monitoring CM2* . ISO/IEC 10857: 1994 permits data beats to go past an end of data status (ED*) with the expectation that a beat error status (BE*) will accompany all data beats beyond the end of data status (ED*). The fault tolerant profile is restricted to disallow this overrun.
<i>Detection mechanism:</i>	End of data verification using reflection
<i>Detection phase:</i>	Disconnection
<i>Protected signal(s):</i>	ST1* and CM2*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_LO.STATUS/CAPABILITY_ERROR SLAVE.FTE_HOLD.TED <- TED* SLAVE.FTE_HOLD.SLD <- SLD* SLAVE.FTE_HOLD.RF <- RF*
<i>Bus error signal:</i>	Slave_Assertion_Reflection_Error
<i>Recovery strategy:</i>	Slave_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.2.3 Broadcast error

<i>Description:</i>	The broadcast status signal (BC*) is received by the current master on ST3* during the connection phase and reflected on CM3* during the data phase. A slave that asserts BC* shall verify its assertion by monitoring CM3* .
<i>Detection mechanism:</i>	Broadcast verification using reflection
<i>Detection phase:</i>	Data
<i>Protected signal(s):</i>	ST3* and CM3*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_LO.STATUS/CAPABILITY_ERROR SLAVE.FTE_HOLD.TED <- TED* SLAVE.FTE_HOLD.SLD <- SLD* SLAVE.FTE_HOLD.RF <- RF*
<i>Bus error signal:</i>	Slave_Assertion_Reflection_Error
<i>Recovery strategy:</i>	Slave_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.2.4 Transaction flag error

5.2.2.4.1 Connection transaction flag error

<i>Description:</i>	The transaction flag signal (TF*) as received by the current master on ST4* during the connection phase is reflected on LT* during the data phase. A slave that asserts TF* during the connection phase shall verify its assertion by monitoring LT* .
<i>Detection mechanism:</i>	Transaction flag verification using reflection
<i>Detection phase:</i>	Data
<i>Protected signal(s):</i>	ST4* and LT*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_LO.STAT US/CAPABILITY_ERROR SLAVE.FTE_HOLD.TED <- TED* SLAVE.FTE_HOLD.SLD <- SLD* SLAVE.FTE_HOLD.RF <- RF*
<i>Bus error signal:</i>	Slave_Assertion_Reflection_Error
<i>Recovery strategy:</i>	Slave_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.2.4.2 Data transaction flag error

<i>Description:</i>	The transaction flag signal (TF*) as received by the current master on ST4* during the data phase is reflected on CM0* during the disconnection phase. A slave that asserts TF* during the data phase shall verify its assertion by monitoring CM0* . The CM0* bit of the command field is only used in the disconnection phase of lock transactions (LK*). TF* can only be used by the cache and message passing protocols, neither of which allows locks in any form. Therefore, TF* and LK* cannot occur during the same transaction.
<i>Detection mechanism:</i>	Transaction flag verification using reflection
<i>Detection phase:</i>	Disconnection
<i>Protected signal(s):</i>	ST4* and CM0*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_LO.STATUS/CAPABILITY_ERROR SLAVE.FTE_HOLD.TED <- TED* SLAVE.FTE_HOLD.SLD <- SLD* SLAVE.FTE_HOLD.RF <- RF*
<i>Bus error signal:</i>	Slave_Assertion_Reflection_Error
<i>Recovery strategy:</i>	Slave_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.2.5 Intervention error

5.2.2.5.1 Connection intervention error

<i>Description:</i>	The assertion of the intervention signal (IV*) as received by the current master on ST5* during the connection phase is reflected by the master in the SW* during the data phase. A slave that asserts IV* during the connection phase shall verify its assertion by monitoring SW* .
<i>Detection mechanism:</i>	Intervention verification using reflection
<i>Detection phase:</i>	Data
<i>Protected signal(s):</i>	IV* and SW*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_LO.STATUS/CAPABILITY_ERROR SLAVE.FTE_HOLD.TED <- TED* SLAVE.FTE_HOLD.SLD <- SLD* SLAVE.FTE_HOLD.RF <- RF*
<i>Bus error signal:</i>	Slave_Assertion_Reflection_Error
<i>Recovery strategy:</i>	Slave_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.2.5.2 Data intervention error

<i>Description:</i>	The assertion of the intervention signal (IV*) as received by the current master on ST5* during the data phase is reflected by the master in the SW* during the disconnection phase. A slave that asserts IV* during the data phase shall verify its assertion by monitoring SW* .
<i>Detection mechanism:</i>	Intervention verification using reflection
<i>Detection phase:</i>	Disconnection
<i>Protected signal(s):</i>	IV* and SW*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_LO.STATUS/CAPABILITY_ERROR SLAVE.FTE_HOLD.TED <- TED* SLAVE.FTE_HOLD.SLD <- SLD* SLAVE.FTE_HOLD.RF <- RF*
<i>Bus error signal:</i>	Slave_Assertion_Reflection_Error
<i>Recovery strategy:</i>	Slave_Recovery_Connect/DataPhase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.2.6 Wait error

<i>Description:</i>	The assertion of the wait signal (WT*) as received by the current master on ST7* during the connection or data phase is reflected by the master on CM3* during the disconnection phase. A slave that asserts WT* shall verify its assertion by monitoring CM3* .
<i>Detection mechanism:</i>	Wait verification using reflection

<i>Detection phase:</i>	Disconnection
<i>Protected signal(s):</i>	WT* and CM3*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_LO.STATUS/CAPABILITY_ERROR SLAVE.FTE_HOLD.TED <- TED* SLAVE.FTE_HOLD.SLD <- SLD* SLAVE.FTE_HOLD.RF <- RF*
<i>Bus error signal:</i>	Slave_Assertion_Refection_Error
<i>Recovery strategy:</i>	Slave_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.2.7 Packet speed error

<i>Description:</i>	The assertion of the packet speed signal (TS*) as received by the current master on CA0* during the connection phase is reflected on CM2* during the data phase. All participating slaves shall verify that the state of reflected TS* (i.e., CM[2]* during the data phase) matches their earlier observation of TS* during the connection phase. Therefore, CM2* can be used to reflect transfer speed (TS*) during the data phase.
<i>Detection mechanism:</i>	Transmit speed verification using reflection
<i>Detection phase:</i>	Data
<i>Protected signal(s):</i>	TS* and CM2*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_LO.STATUS/CAPABILITY_ERROR SLAVE.FTE_HOLD.TED <- TED* SLAVE.FTE_HOLD.SLD <- SLD* SLAVE.FTE_HOLD.RF <- RF*
<i>Bus error signal:</i>	Bus Error Signal: Slave_Assertion_Refection_Error
<i>Recovery strategy:</i>	Slave_Recovery_Connect/Data_Phase
<i>Retry strategy:</i>	None, discretion of master
<i>Containment strategy:</i>	Module_Containment_Transaction_Error

5.2.2.8 Compelled error

<i>Description:</i>	The compelled signal (CO*) as received by the current master on CA1* during the connection phase is reflected by the RF* signal during the data phase. All participating0 slaves shall verify that the state of reflected CO* (i.e., RF* during the data phase) matches their earlier observation of CO* during the connection phase.
<i>Detection mechanism:</i>	Compelled mode verification using reflection
<i>Detection phase:</i>	Data
<i>Protected signal(s):</i>	CO* and RF*
<i>Error log snapshot:</i>	Slave_Snap_Group(Master_Not_Principal) SLAVE.ERROR_LO.STATUS/CAPABILITY_ERROR SLAVE.FTE_HOLD.TED <- TED* SLAVE.FTE_HOLD.SLD <- SLD* SLAVE.FTE_HOLD.RF <- RF*