28-Sep-2012

MIPI® Alliance Specification for Serial Low-power Inter-chip Media Bus (SLIMbus®)

Version 1.1 – 28 September 2012

* NOTE TO IMPLEMENTERS *

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

The following table includes a list of technical updates made to this document since *MIPI Alliance Specification for Serial Low-power Inter-chip Media Bus (SLIMbus®)* v1.01.01 was released on 14 July 2008. Note, the table does not include editorial Corrections such as page formatting, typographical changes or paragraph and character style changes.

Section	Description
4.4.3	Clarified the requirement for Component support of Root Frequency, Clock Gear and Subframe Modes.
6.1	Minor wording updates, such as Cells within a Slot, Slots within a Subframe.
6.4.2	Added definition of Segment Offset with respect to Control Space Width.
7.1	Added Figure 33 "Example Information Slice using Elemental Access".
7.1.2	Clarified description of how to clear Information Elements.
7.1.2.4	Added requirement for clearing the RECONFIG_OBJECTION Information Element.
8.2.1.3	Added requirements after Table 30 "Arbitration Priority Codes" for using Arbitration Priority codes.
8.2.3	Added cross-reference to Section 11 "Core Message".
8.4.2	Added requirement for the Active Manager to keep the Enumeration Address of a Device to be assigned a Logical Address.
8.4.2.5	Clarified requirements and text regarding parsing of the Remaining Length field.

Section	Description
9.3.5.4.2	Deleted sentence regarding two successive bit errors.
9.3.5.5	Updated the comment for “Segment i+2” and replaced the T1 value to indicate the primary owner is requesting channel ownership.
9.5	Added paragraph describing the Port behavior when an unsupported Data Type request has been received.
9.5	Added definition of PDM Data Type. Also added Figure 54 “Packed PDM Segment DATA Field”.
10.1.2	Added requirement for Clock Receiver Components to support the synchronization for any combination of Clock Gear, Root Frequency and Subframe Mode.
10.1.2.1	Clarified the definition of the Undefined State.
10.1.2.4	Minor wording updates.
10.2.2	Updated first paragraph.
10.3.1	Added paragraph stating certain combinations of NEXT_MESSAGES are forbidden.
10.3.2	Rewording after Figure 67 “Restart of a Paused Bus Clock” for the Restart after Paused Clock. Paragraph added explaining Apparent Activation by Component.
10.4.2	Added a requirement prohibiting the active Manager from sending a RESET_DEVICE Message to the Interface Device of the Clock Sourcing Component.
10.4.3	Added a requirement prohibiting the active Manager from sending a RESET_DEVICE Message to the Framer of a Clock Sourcing Component.
10.5	Clarified the requirement for Clock Gear Changes when a function on a Device is not capable of performing at the new Clock Gear.
10.6.1	Clarified definitions and terms for Message, Guide, and Data Channel.
10.7.	Added details on Root Frequency change and on what happens when an active Framer does not support the requested next Root Frequency.
10.8	Added text before Figure 77 “Changing the Message Channel Characteristics” explaining the related bus clock for NCo and NCi.
10.9.1	Updated the text to be consistent with Figure 79 “Device Enumeration State Diagram”.
10.9.1.1	Removed the definition of a predetermined Logical Address.
10.9.1.5	Clarified the requirement for detaching from the bus.
10.11	Added a requirement for the active Manager to use the CHANGE_ARBITRATION_PRIORITY Message to assign a new Arbitration Priority code to a Device. Deleted or modified Device requirements for changing Arbitration Priorities of Messages at runtime.
11.1.2	Minor wording updates.
11.2.3	Minor wording updates.
11.3.2	Added cross-reference to Section 10.12.3 “Atomic Transactions” after “CM field”.
11.4.2	Added a requirement prohibiting the active Manager from sending NEXT_ACTIVE_FRAMER and NEXT_PAUSE_CLOCK Messages in the same Reconfiguration Sequence.

Section	Description
11.6.2	Same as Section 11.4.2.
11.5.2	Added cross-reference to Section 10.12.3 “Atomic Transactions” after “VU field”.
11.6	Added definition for EX_ERROR flag if implemented.
12.2.2	Added cross-reference to Section 10.4.3 “Device Reset” in table note 5 of Table 67 “Interface Device Core Message Support”.
12.2.4	Minor wording updates.
12.3.2	Added table note 3 to Table 70 “Manager Core Message Support”.
12.3.5	Removed last paragraph in section. The text explained the clearing of Information Elements, which is not applicable to read-only IEs.
12.4.2	Added cross-reference to Section 10.4.3 “Device Reset” in table note 2 of Table 73 “Framer Core Message Support”.
12.4.5	Minor wording updates.

MIPI® Alliance Specification for Serial Low-power Inter-chip Media Bus (SLIMbus®)

Version 1.1 – 28 September 2012

MIPI Board Approved 19-Mar-2013

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

NOTICE OF DISCLAIMER

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

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

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

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

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

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

Contents

Figures

Tables

Release History

Date	Release	Description
06-Mar-2007	v1.00.00	Initial Board approved release.
14-Jul-2008	v1.01.01	Board approved release.
19-Mar-2013	V1.1	Boad Approved release.

DRAFT MIPI Alliance Specification for SLIMbus

1 Introduction

Demand for multimedia functions within mobile terminals is increasing. A key driver for unit growth and product differentiation is digital audio.

Commonly used digital audio interfaces in mobile terminals such as I^2S and PCM are generally intended for point-to-point connection between an application processor and a single digital audio device. Typically, these interfaces only support one or two digital audio channels. As such, adding functions and digital audio channels to mobile terminals beyond those for voice communication and simple stereo music applications is very difficult without increasing the number of bus structures in the mobile terminal. Though scalable, just adding bus structures limits design flexibility and is costly in terms of pin count, package size, PCB layout area and power consumption.

In addition, numerous control bus structures such as I^2C, SPI, microWire™, UART and GPIOs, typically reside in parallel to audio buses not only for audio device control, but handling the countless low bandwidth control tasks required for lighting, haptic feedback, temperature monitoring and power sequencing.

SLIMbus provides the mobile terminal industry a standard, robust, scalable, low-power, high-speed, cost-effective, two wire multi-drop interface that supports a wide range of digital audio and control solutions for mobile terminals.

SLIMbus effectively replaces many other digital audio buses such as PCM and I^2S, as well as control buses such as I^2C, SPI, or UART, in a mobile terminal by providing flexible and dynamic assignment of bus bandwidth between digital audio and non-audio control and data functions.

SLIMbus is not backward compatible with any current digital audio bus or digital control bus.

1.1 Scope

SLIMbus interface protocols and commands are within the scope of this specification.

Electrical specifications at the physical terminals of SLIMbus Devices, as well as a description of signal timing relationships, are also within the scope of this specification.

Implementation details of the SLIMbus interface within an electronic device are not within the scope of this document.

1.2 Purpose

The purpose of this document is to specify a standard interface between baseband or application processors and peripheral components.

Implementing the SLIMbus Specification greatly increases the flexibility for mobile terminal system designers to realize multiple products within a product line quickly, each with widely varying digital audio and user interface features, without the addition of multiple bus structures.

SLIMbus reduces the time-to-market and design cost of mobile terminals by simplifying the interconnection of products from different manufacturers.

1.3 Summary of SLIMbus Features

A SLIMbus interface offers many benefits, including:

- Audio, data, bus and Device control on a single bus
- Reduced pin count for lower overall product cost
- Support for more than ten Components at typical bus lengths and speeds
- Support for multiple, high quality audio channels
- Multiple, concurrent sample rates on a single bus
- Efficient, host-less, peer-to-peer communication
- Standardized Message set for improved software reuse and increased interoperability
- Established framework for Device creation
- Use of common digital audio clocks as well as established system clocks
- Dynamic clock frequency changes for optimizing bus power consumption

2 Terminology

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

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

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

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

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

Throughout this document, the chronology for binary sequences and timing diagrams is from left (earlier in time) to right (later in time), unless otherwise specified. In multi-row sequence diagrams, the chronology is from upper rows (earlier in time) to lower rows (later in time), unless otherwise stated.

2.1 Definitions

Cardinal Frequency: A CLK frequency from the set 24.576 MHz, 12.288 MHz, 6.144 MHz etc. See Section 6.2.6 for more detail.

Cell: The smallest subdivision of the SLIMbus data stream is the Cell. A Cell is the region of the DATA line signal that lies between two consecutive positive edges of the CLK.

Class-specific Message: A Message with a Message Code that is interpreted by a destination Device according to the version of the Device Class of the source Device (Source-referred Messages) or to its own version of its Device Class (Destination-referred Messages).

Clock Gear: Ten Clock Gears (CG) are defined that divide the Root Frequency by powers of two. For a given Root Frequency, the Clock Gears provide a 512x range of frequencies for the SLIMbus CLK and can help optimize power consumption and bandwidth.

Clock Sourcing Component: A Clock Sourcing Component is the Component that contains the active Framer.

Clock Receiving Component: A Clock Receiving Component is a Component that does not contain the active Framer.

Collision: A Collision occurs when a Device detects a value on the DATA line that differs from the value it drove. Collisions cannot occur during normal operation of the bus. To detect error conditions, Devices check for Collisions as described in Section 5.1.1.

Component: A Component contains a set of Devices and the necessary logic needed to provide bus access to these Devices. Each Component contains an Interface Device and typically contains one or more other Devices.

Control Space: Control Space is a programmable portion of a Frame defined by the repetition of Subframes within the Frame and by the Control Space width in the Subframes. See Section 6.3.

Core Message: A Message with a Message Code that is interpreted in the same way by all Devices regardless of Device Class. See Section 11 for more information.

Data Channel: A portion of Data Space allocated for communication between a subset of the Devices on the SLIMbus. See Section 6.4.2.

Data Space: Data Space is a programmable portion of each Frame. If present, Data Space is interleaved with Control Space at the Subframe level. Refer to Section 6.1, Section 6.2 and Section 6.4 for more information.

Device: A Device is a separately addressable entity within a SLIMbus Component. Each Device belongs to a single Device Class and implements a single version of the corresponding Device Class definition. After enumeration each Device is addressed with a single Logical Address.

Device Class: A Device Class is a set of behaviors, Messages, Transport Protocols and Information Elements that define a specific type of Device. Device Classes are specified in Section 12.

Device Class Version: A Device Class Version (DCV) is an 8-bit code used to identify the specific implementation of Device Class for a particular Device.

Enumeration Address: An Enumeration Address is a 48-bit value that allows a Device to receive Messages. An Enumeration Address is comprised of four fields: Manufacturer ID, Product Code, Device Index and Instance Value. See Section 8.2.5.1 for details.

Frame: A Frame is a group of 192 contiguous Slots (768 Cells). The first Slot of each Frame carries the Frame Sync symbol. See Section 6.1.4.

Framer: A Framer is a Device that can be the source of the SLIMbus clock (CLK), Framing Channel and Guide Channel. There is one active Framer on an instance of SLIMbus at any one time. This document allows for alternate Framers and defines the process for handover from one Framer to another.

Frame Structure: The Frame Structure is the organization of defined areas within the TDM stream that allow for the transport of isochronous and packet data formats. See Section 6 for a complete description.

Frame Sync symbol: The Frame Sync symbol is a fixed value (0b1011) in the first Slot of every Frame, that a Component uses to achieve synchronization with the Frame Structure.

Framing Channel: A portion of Control Space allocated for the transmission of Frame Sync symbols and Framing Information. The Framing Channel is written to the bus by the active Framer.

Framing Information: The Framing Information is a portion of the Framing Channel that contains information on the Frame Structure organization and bus timing. Refer to Section 6.3.1.1 for more information.

Guide Channel: The Guide Channel provides the necessary information for Components to acquire and verify Message synchronization in the Message Channel. It uses two Slots of Control Space, called the Guide Byte, once every Superframe.

INT(): A mathematical function that returns the integer portion of a number. For example, **INT**(4.12) returns 4.

Interface Device: Each Component contains one Interface Device. The Interface Device is responsible for monitoring the Frame Layer and the Message Protocol, Component Reset, and status reporting.

Information Element. An Information Element is a specific piece of data that resides in a Device, and that is made available to other Devices via Messages. Some Information Elements are common for all Devices and are defined as Core IEs; others are specific to a Device Class or are user definable.

Logical Address: A Logical Address is an 8-bit number used to uniquely identify an enumerated Device on the bus. See Section 8.2.5.2.

Manager: A Manager is a Device that is responsible for bus configuration and Device management. Although multiple Components on a single SLIMbus may include Managers, only a single Manager may be active at any one time.

Message: A Message is an item that is sent from one Device to one or more other Devices through the Message Channel. Some Messages are common for all Devices and are defined as Core Messages; others are specific to a Device Class or are user definable.

Message Channel: The Message Channel is composed of Control Space Slots not allocated to the Framing Channel or Guide Channel. It is used for bus configuration and Device management as well as general purpose Device-to-Device Messaging.

Natural Frequency: A Natural Frequency describes a CLK line rate that is an exact multiple of a desired audio sample rate. A Natural Frequency allows such flows to be conveyed without flow control mechanisms. In addition, a Natural Frequency can also ease implementation by making the SLIMbus CLK directly usable within a Component. See Section 6.2.5 for a detailed explanation.

Phasing Signal: The Phasing Signal communicates an embedded 25 Hz timing reference. It is part of the Framing Information. Consuming two Cells per Superframe, it transports one 14-bit value called a Phase Snapshot every ten Superframes. See Section 6.3.1.8.

Port: A Port is the part of a Device through which data flows to or from a single Data Channel. The Port contains the necessary parameters, i.e. connection status, Channel Number, Transport Protocol used, and the relevant Data Channel Parameters for a Device to connect to the Data Channel. Port parameters may be fixed at design time or may be programmable. Ports are the only SLIMbus entities that are allowed to access the Data Space. A Device may have more than one Port.

Reconfiguration Boundary: The Reconfiguration Boundary is the first Superframe boundary that comes at least two Slots after the end of a RECONFIGURE_NOW Message.

Reconfiguration Message: A Reconfiguration Message is a Message that forms part of the Reconfiguration Sequence. See Section 11.4.

Reconfiguration Sequence: A sequence of Messages starting with BEGIN_RECONFIGURATION, followed by a variable number of NEXT_ Messages and finished with a RECONFIGURE_NOW Message is known as a Reconfiguration Sequence. This process provides the ability to change the bus organization and CLK frequency without disrupting the contents of any Data Channels.

Root Frequency: The Root Frequency is the frequency of the SLIMbus CLK line in Gear 10. In lower Clock Gears the CLK runs at the Root Frequency divided by $2^{(10-CG)}$, where CG is the Clock Gear. The Root Frequency establishes a basis for describing the SLIMbus CLK line frequency. In Gear 1 through Gear 9, typical systems can generate the SLIMbus CLK without ever generating a signal at the Root Frequency.

Root Superframe: Root Superframes are Clock Gear-independent divisions of the SLIMbus timeline. They have the same frequency and phasing as the bus's Superframes in Gear 10. In lower Clock Gears, each Superframe spans $2^{(10-CG)}$ Root Superframes, where CG is the Clock Gear. See Section 6.2.4.

Segment: A Segment is a distinct group of contiguous Data Space Slots that have been allocated to the same Data Channel.

Segment Distribution: The Segment Distribution (SD) defines the Segment Offset and the Segment Interval and is used in the allocation of Data Space Slots to Data Channels See Section 6.4.2.

Slot: A Slot is a group of four contiguous Cells. It is at the Slot level that the SLIMbus DATA signal is time-division-multiplexed between different channels.

Subframe: A Subframe is the division of the Frame Structure at which Control Space and, if present, Data Space are interleaved. The Subframe length can be 6, 8, 24, and 32 Slots (24, 32, 96, and 128 Cells).

Subframe Mode: The Subframe Mode (SM) is a binary code carried in the Framing Information that indicates what Subframe length and Control Space width are being used.

Superframe: A Superframe is composed of eight consecutive Frames and contains a complete set of Framing Information.

Transport Protocol: A Transport Protocol is an attribute of a Data Channel that describes how flow control is accomplished. Several Transport Protocols are defined. Each Transport Protocol describes a set of behaviors for a particular type of flow control.

User Message: A Message with a Message Code that is interpreted by a destination Device according to the product documentation of the source Device (Source-referred Messages) or of the destination Device (for Destination-referred Messages).

Whitening Signal: The Whitening Signal is part of the Framing Information and consumes one Cell per Superframe. The Whitening Signal helps to keep the spectrum of the SLIMbus DATA signal benign, improve analog performance and can ease system integration.

2.2 Abbreviations

e.g. For example (Latin: exempli gratia)

i.e. That is (Latin: id est)

2.3 Acronyms

AES	Audio Engineering Society
AT	Arbitration Type
CG	Clock Gear
DC	Device Class
DI	Device Index
DT	Destination Type
EA	Enumeration Address
FE	Framing Extension
IV	Instance Value
ISTO	Industry Standards and Technology Organization
LA	Logical Address
LPCM	Linear Pulse Code Modulation
LSB	Least Significant Bit
MC	Message Code
MI	Manufacturer ID
MT	Message Type
Mbps	Megabits per second
MEMS	Micro-Electro-Mechanical Structures
MIPI	Mobile Industry Processor Interface
MSB	Most Significant Bit
MT	Message Type
NRZI	Non-Return-to-Zero Inverted
PC	Product Code
PDM	Pulse Density Modulation
PI	Primary Integrity
PS	Phase Snapshot
RF	Root Frequency

740	RL	Remaining Length
741	SPDIF	Sony / Philips Digital Interface
742	SLIMbus	Serial Low-power Inter-chip Media Bus
743	SM	Subframe Mode
744	TDM	Time Division Multiplexed
745	TID	Transaction ID

3 References

[AES01] AES-12id-2006, *AES Information Document for digital audio measurements - Jitter performance specifications*, Audio Engineering Society, New York, NY, USA.

[IEC01] IEC 60958, *Digital audio interface*, International Electrotechnical Commission, Geneva, Switzerland.

[IEC02] IEC 61937, *Digital audio - Interface for non-LPCM-encoded audio bitstreams applying IEC 60958*, International Electrotechnical Commission, Geneva, Switzerland.

[MIPI01] MIPI Alliance, Inc., "MIPI Alliance Manufacturer ID Page", <http://mid.mipi.org/>, 28 September 2012.

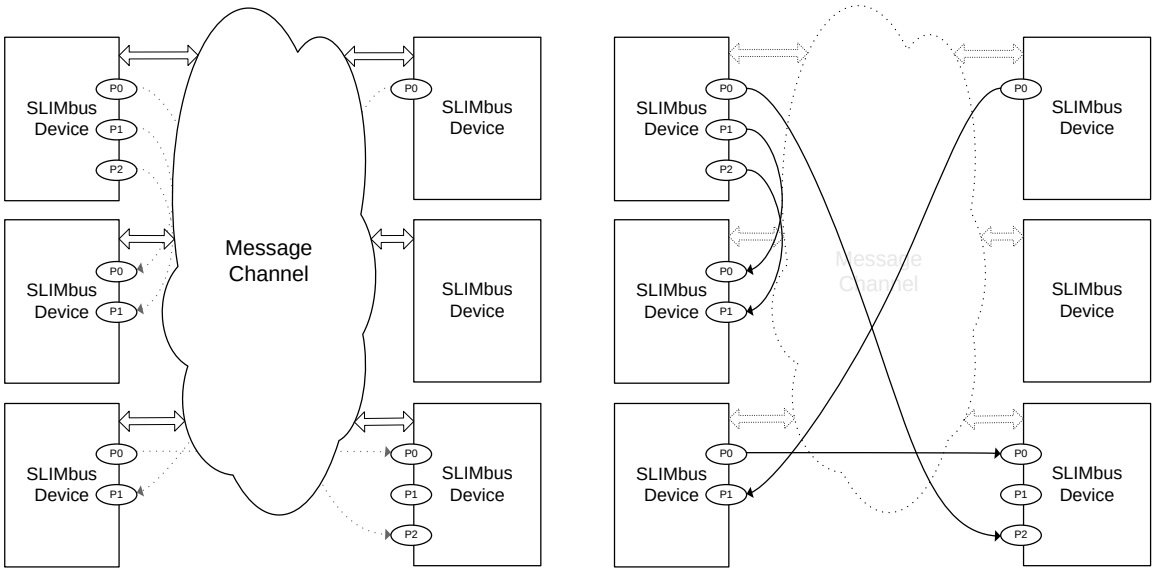
4 The SLIMbus Model

SLIMbus models a system as a set of Devices communicating over a shared bus. All transfers on the bus are handled by predefined Transport Protocols optimized for specific data flows. These protocols include several Transport Protocols designed to handle common flow types such as isochronous and asynchronous streams as well as protocols to handle bus control and configuration traffic. In order to provide system flexibility at a low cost, the shared bus is configured in a synchronous, two-wire, multidrop design and utilizes a TDM frame structure.

In order to reduce the requirements for Device implementation, this document is written in such a manner as to place restrictions on activity that would result in undesirable operation. Therefore, unless specifically stated otherwise, there is no requirement for an implementation to check for, or react to, SLIMbus activity that is prohibited by design. For example, a Device is not required to validate that another Device is permitted to send a particular type of command.

4.1 Device Communication

SLIMbus offers Devices two methods of communicating with each other, Messages and Data Channels. Messages can be used for various control functions such as bus management, configuration and status updates. Data Channels have been designed to provide the necessary services for data transport, particularly audio streaming. Figure 1 shows a conceptual view of a SLIMbus system.



(a) Message Communication (b) Channel-based Communication

Figure 1 SLIMbus System View

Note, in the figure a SLIMbus Device does not necessarily represent a packaged electronic component. See Section 4.3 for more information.

4.2 Bus Organization

SLIMbus uses a TDM frame structure that is optimized for the transport of constant bitrate media streams, but can accommodate various types of asynchronous data transport as well. Control information can also be interleaved with the data streams.

The SLIMbus Frame Structure is partitioned into two regions known as Control Space and Data Space. Control Space is used to supply configuration, synchronization and status information on the bus. Data Space carries application-specific data between Devices.

4.2.1 Control Space

The size of the Control Space is programmable and controlled by the Subframe Mode which determines the Subframe length and Control Space width. Control Space is composed of three channels of information: the Framing Channel, the Guide Channel and the Message Channel, each with a different responsibility.

The Framing Channel conveys two types of information, a Frame Sync symbol used to indicate the start of a Frame, and the Framing Information which includes information indicating the current state of the bus and the size of Control Space. The Framing Channel relative bandwidth is fixed.

The Guide Channel provides the necessary information for Devices to acquire and verify Message synchronization in the Message Channel. The Guide Channel relative bandwidth is fixed.

The Message Channel is used to transfer Messages between Devices. The bandwidth allocated to the Message Channel is programmable to allow an optimal balance between Message latency and throughput, and bus efficiency.

4.2.2 Data Space

Any bus bandwidth not allocated to Control Space is Data Space. Data Space is composed of zero or more Data Channels that are dynamically created by the active Manager depending on the application. The number of Data Channels depends on the size of Data Space and type of data streams carried by the channels.

4.3 Devices

A Device can be thought of as the logical implementation of a system feature such as audio input or output. As such, a Device needs several parts to handle control information as well as process any data required to realize the feature. The fundamental pieces of the Device are the Enumeration and Logical Addresses, Information and Value Elements, and Ports.

A Device identifies itself using two different addresses, an Enumeration Address and a Logical Address, to "connect" to control information. An Enumeration Address is a 48-bit number comprised of four fields: a Manufacturer ID, a Product Code, a Device Index and an Instance Value. An Enumeration Address is primarily used during Device discovery and bus configuration. For more information see Section 8.2.5.1.

A Logical Address is an 8-bit number assigned during the enumeration procedure. It provides a convenient, shorthand notation for directing control information to the appropriate Device. A Logical Address also provides some control of the bus arbitration procedure. See Section 8.2.5.2 for more information.

Bus and Device status information is maintained by each Device in standardized bit-fields called Information Elements. An Information Element can be Boolean, having only two distinct values, or enumerated, having many values. An Information Element can be read using a REQUEST-REPLY mechanism or a Device may autonomously send the Information Element status using a special REPORT Message. In addition, some Information Elements can be changed to their reset values using the CLEAR_INFORMATION Message.

Another, more general type of data store, the Value Element, can be used to provide a standardized method to read and update Device parameters. Similar to an Information Element, a Value Element can be Boolean or enumerated and also supports a REQUEST-REPLY mechanism. Unlike an Information Element, a Value Element can be set to a specific value using the CHANGE_VALUE Message.

To reduce bandwidth requirements, multiple Information or Value Elements can be accessed in a single transaction using Slices. See Section 7 for more information.

The final part of a Device is the Port. Just as a Logical Address provides a “connection” to control information, a Port provides the connection to data. A Device may have up to sixty-four Ports.

Port capabilities may differ based on the features supported by the Device and should be specified in the Component data sheet. Typical Port attributes include data directionality, i.e. input-only (sink), output-only (source), or both input and output; supported Transport Protocols; and data width. For example, the Port attributes for a MEMS microphone could be output-only, Isochronous Transport Protocol and 16-bit data width.

4.3.1 Device Classes

To facilitate Device design, this document provides definitions of four Device Classes. A Device Class definition specifies the minimum requirements for control information, Device behavior, Transport Protocol support and data storage necessary to implement a Device of that Device Class.

Devices based on three of the Device Class definitions: Interface Device, Manager and Framer, are required to create a functioning SLIMbus. The fourth Device Class, Generic, is available for use by any Device for which there is no specific Device Class defined.

4.3.1.1 Interface Device

An Interface Device is a member of the Interface Device Class and provides bus management services for the Component in which it resides. The Interface Device monitors the Frame Layer and Message Protocols implemented by the Component. The Interface Device also manages Component Reset so that a Component can properly sequence its Devices. In addition, the Interface Device reports information about the status of the Component. Each Component has one Interface Device. See Section 12.2 for more information.

4.3.1.2 Manager

Managers are Devices belonging to the Manager Device Class. A Manager has all the capabilities needed to administer the bus, such as bus enumeration, bus configuration and dynamic channel allocation. Multiple Managers may exist on a single bus, but only one of the Managers shall be active in the Manager role at any given time. The Manager responsible for administering the bus is known as the active Manager. While more than one Manager may be on the bus, the mechanism for changing Managers is outside the scope of this document. See Section 12.3 for more information.

4.3.1.3 Framer

Framers are Devices belonging to the Framer Device Class. A Framer drives the CLK line and places information on the DATA line required to establish the Frame Structure on the bus. Multiple Framers may exist on a single bus, but only one Framer drives the CLK line at any given time. The Framer that is driving the CLK line is called the active Framer. This document defines a mechanism for changing Framers while the bus is active. See Section 12.4 for more information.

4.3.1.4 Generic Device

Generic Devices are members of the Generic Device Class. The Generic Device Class provides basic SLIMbus functionality for a Device. See Section 12.5 for more information.

4.4 Components

In order to simplify the design and development of SLIMbus Components, each Component can be viewed as containing a set of Devices and the necessary circuitry needed to provide bus access to these Devices. A Component shall have one, and only one, Interface Device, but may have many Devices of other Device Classes. Basic requirements for Components are discussed in this section. However, Component design is beyond the scope of this document.

4.4.1 Simple Component Example

Figure 2 shows the building blocks of a simple Component that contains two Devices. Each Device communicates over the bus using the same Message Protocol. In addition, one of the Devices contains a single Port, P0, that uses a dedicated Transport Protocol to transfer data between the system feature it is implementing, possibly a stereo DAC, and other Devices on the bus.

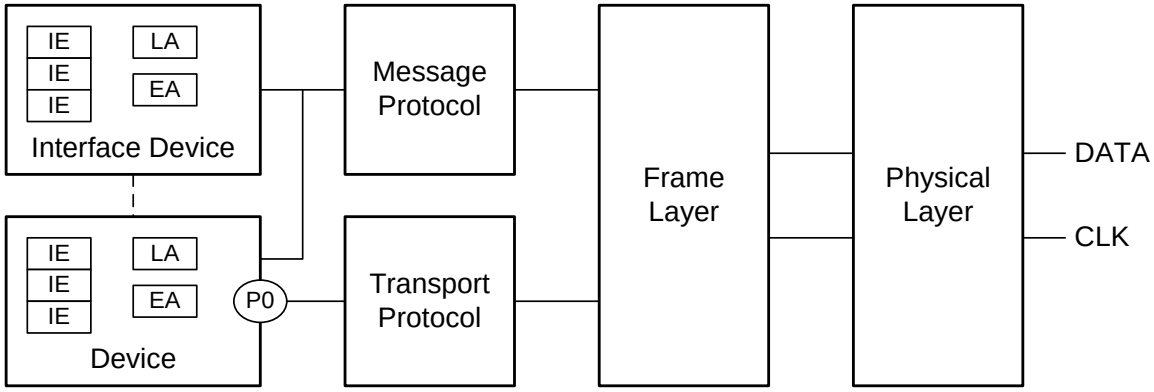


Figure 2 Simple SLIMbus Component Example

As can be seen in the figure, data and control information sent from a Device are first encoded using a specific protocol. For example, the Message Protocol for control information and the Isochronous Transport Protocol for data. The data and control streams are then interleaved by the Frame Layer and finally converted by the Physical Layer into electrical signals on the DATA and CLK lines. In the opposite direction, the signals on the CLK and DATA lines are translated by the Physical Layer into a bit stream. The bit stream is then split into data and control streams by the Frame Layer. The data and control streams are then decoded by the corresponding protocols and sent to the appropriate Devices within the Component.

4.4.2 Complex Component Example

A more complex Component example can be found in Figure 3, where three Devices are shown, one containing a significant number of Ports.

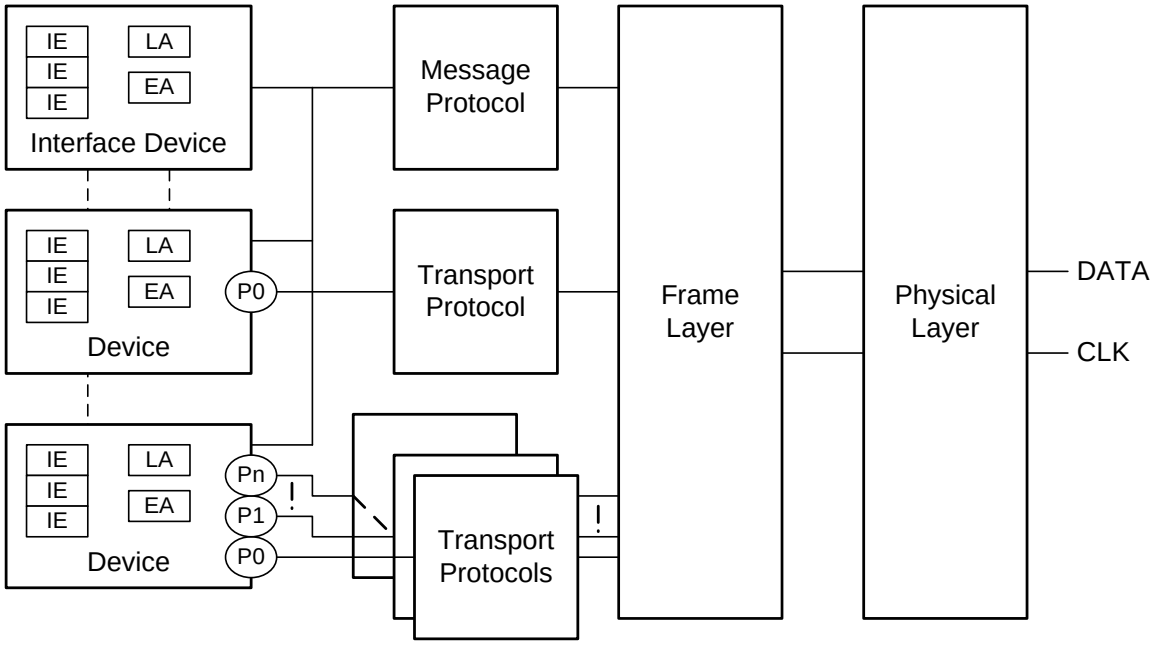


Figure 3 Complex SLIMbus Component Example

4.4.3 Component Requirements

A Component shall operate within the timing and electrical constraints given in Section 5. When the Component is a Clock Receiving Component, its Physical Layer shall be able to operate over the entire frequency range of 0 to 28.8 MHz. In addition, its Frame Layer and Message Protocol shall support all Subframe Modes, Clock Gears and Root Frequencies. A Clock Receiving Component shall also support access to IEs in all Subframe Modes, Clock Gears and Root Frequencies.

A Component shall be able to synchronize to the bus, and a Clock Receiving Component shall be able to determine the bus parameters from the Framing Information. In addition, a Component shall implement the Message Protocol.

4.5 Channels

Channels are used to convey information on the bus. Both control information and data can be transferred between a pair of Devices (inter-Device communication), or between one Device and many other Devices (Broadcast communication). Since SLIMbus provides both Data Channels and Control Space channels, protocols for each type are defined in this document.

4.5.1 Data Channels

Information carried by a Data Channel is application specific and many different data formats exist. In order to avoid encumbering a system with a large number of data protocols, this document, instead, identifies a handful of frequently used Transport Protocols that can be used with virtually any data format to dynamically create the required Data Channels.

Transport Protocols define a data flow type, such as isochronous or asynchronous, a flow control mechanism and a side-channel for any additional application-specific information. A common data structure, the Segment, is defined to hold the Transport Protocol information. See Section 6.4.1 for more information.

4.5.2 Control Space Channels

Control Space channels are unique on SLIMbus in that there is no common data structure defining the flow control for a channel. In addition, two of the channels, the Guide Channel and the Framing Channel, have no programmable attributes and utilize a fixed relative bandwidth. Consequently the definition for these two channels is somewhat simpler than other channels.

4.5.2.1 Framing Channel

The Framing Channel has no provision for flow control and has a fixed channel-relative bandwidth. The Framing Channel carries two types of information, the Frame Sync symbol and the Framing Information. Both pieces of information are sent at regular intervals though they are offset in time. See Section 6.3.1 for more information.

4.5.2.2 Guide Channel

The Guide Channel is the simplest of the three Control Space channels. Like the Framing Channel, no flow control is provided and the channel-relative bandwidth is fixed. Data is organized as a simple Guide Byte that is sent at regular intervals. See Section 6.3.2 for more information.

4.5.2.3 Message Channel

The Message Channel is the most complex of the Control Space channels. Flow control is provided in the form of an acknowledgement symbol and the channel-relative bandwidth is programmable. Various types of information can be carried in the Message Channel including bus configuration, Device control and Device status. Information is transferred between Devices in containers called Messages. See Section 8 for more information.

4.6 Frame Layer

The SLIMbus Frame Structure interleaves Control Space channels and Data Channels into a single, serialized bit stream. Each Component uses a functional block called the Frame Layer to combine control and data information into a single bit stream for transmission on the bus. Also, the Frame Layer is used to split the incoming bit stream into separate control and data streams.

In addition, a Framer also contains logic to generate the Guide and Framing Channels used to synchronize Devices on the bus.

Design of the Frame Layer is beyond the scope of this document.

4.7 Physical Layer

The Physical Layer provides for the transmission and reception of the SLIMbus bit stream between Components. SLIMbus uses a synchronous, two-wire, multi-drop bus architecture to transfer information between Components. The two lines, CLK and DATA, are the only wires required to implement the bus. The CLK line distributes a high-quality, unidirectional clock signal to all Components. The DATA line is bidirectional, carrying all information sent or received on the bus. Information is signaled using Non-Return-to-Zero Inverted, or NRZI encoding. See Section 5 for additional information.

4.8 System Example

Figure 4 shows an example system that can be designed with SLIMbus, where several Components are shown around the SLIMbus. The top-left Component is an Application Engine, the bottom-left is a baseband. On the right side two microphones and a dual channel DAC are shown. The data of the

microphones is transmitted to the Application Engine, where it is processed and transmitted to the baseband. Post processing of the incoming audio is performed in the stereo DAC and therefore the audio is transported directly from the baseband to the DAC.

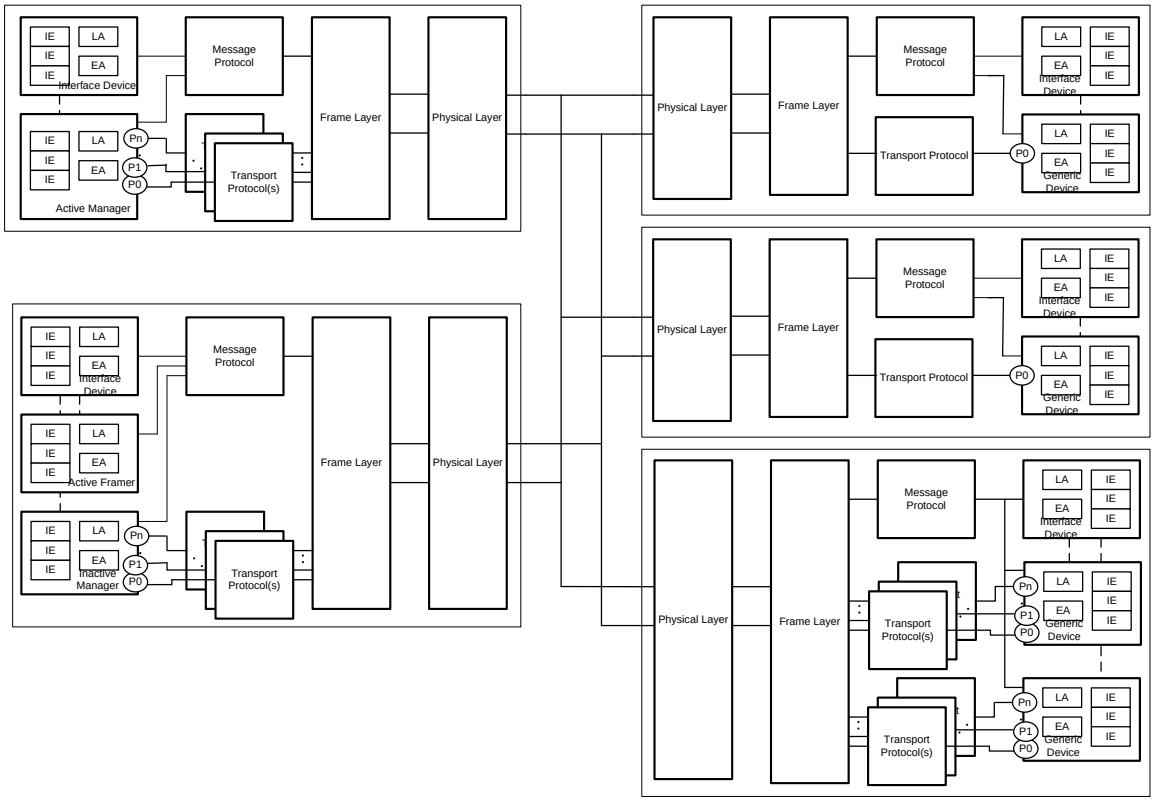


Figure 4 System Example

The Application Engine contains two Devices, one being an Interface Device and the other a Manager. In this configuration, the Manager in the Application Engine will be the active Manager. This Manager contains a number of Ports, used in this use case to transport audio data to and from the Application Engine. Note that in this example, the Ports support multiple Transport Protocols, so that the Application Engine can be used in multiple systems.

The baseband contains three Devices, one Interface Device, a Framer and another Manager. In this scenario, this Framer is the only one in the system and will be used to clock the bus. The Manager will not be used as the active Manager, and is inactive in this use case. However, as the Manager is the only Device that contains Ports, it is still configured to transmit and receive audio data.

The top two Components on the right are identical MEMS Microphones that both contain two Devices: An Interface Device and a Generic Device. These Generic Devices only contain a single Port, which in this example are only able to transmit audio data using the Isochronous Protocol, at a 16 kHz sample rate. Given that these are low cost Devices, they extensively use the bus clock signal internally and require a bus clock frequency of 3.072, 6.144, 12.288 or 24.576 MHz, in order to be able to generate the audio.

The bottom-right Component is a stereo DAC, and contains three Devices, one Interface Device, and two Generic Devices, one of which supports stereo audio playback. As the DAC has been designed to operate at a broader frequency range and in more complex use cases than the microphones, the Generic Devices have multiple Ports which support multiple Transport Protocols each.

5 Physical Layer

The SLIMbus Physical Layer can be thought of as divided into two parts, a Physical Medium Independent part and a Physical Medium Dependent part. Section 5.1 describes the Physical Medium Independent aspect and Section 5.2 describes the Physical Medium Dependent aspect in more detail.

A SLIMbus system uses a synchronous, two-wire, multi-drop bus to transfer information between Components. One wire is used to transfer a clock signal while the other wire transfer a data signal. Throughout this document the two wires are known as the CLK line and DATA line, respectively.

The CLK and DATA lines shall use single-ended, ground referenced, rail-to-rail, voltage mode signals. The DATA line shall be driven on the positive edge and read on the negative edge of the CLK signal. The CLK line shall have only two states, driven high and driven low. The DATA line can be driven high, driven low, or held at the high or low level.

DATA terminals shall be bidirectional. If a Component contains a Framer, the CLK terminal shall be bidirectional. If a Component does not contain a Framer, the CLK terminal shall be input-only. Figure 5 illustrates the conceptual IO configurations for a SLIMbus Component.

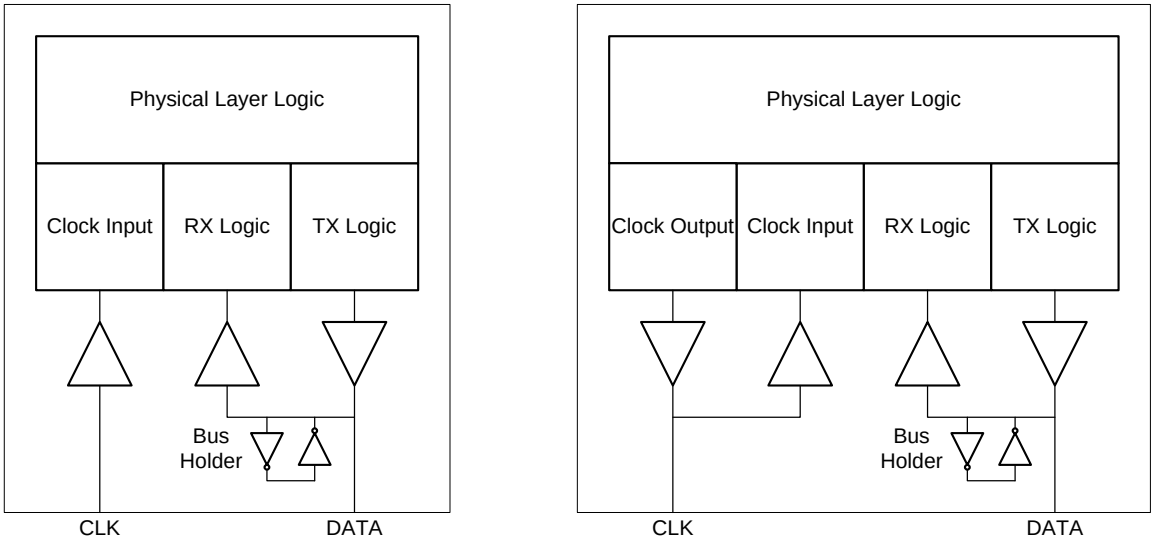
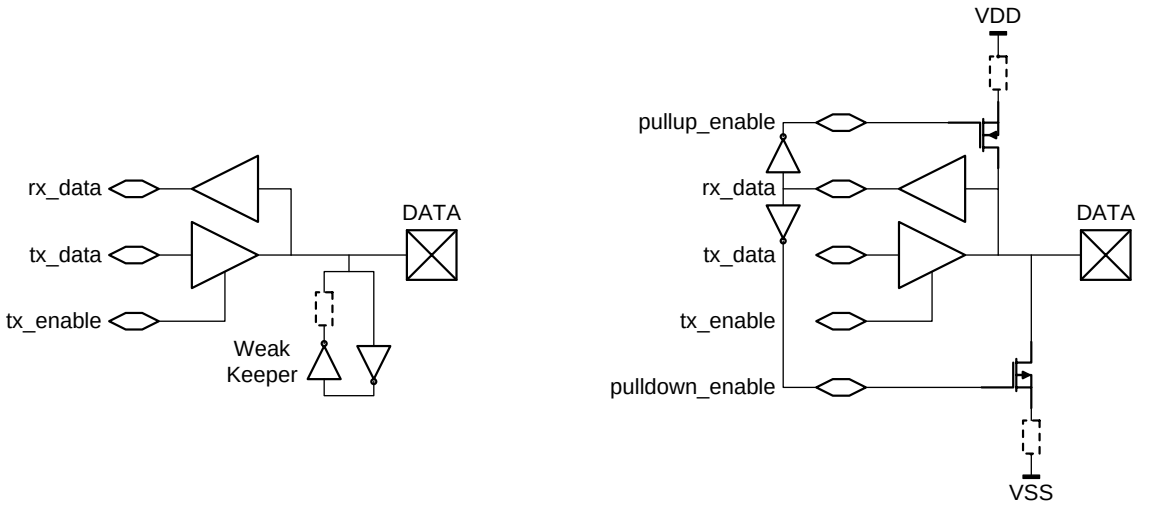


Figure 5 Conceptual IO Configurations for Components on SLIMbus

A DATA terminal shall implement a bus holder to maintain the DATA line at the last high or low level. Figure 6 shows two possible bus holder implementations.



(a) Conceptual bus holder (b) Generic CMOS IO containing weak pull-up/pull-down

Figure 6 DATA Terminal Bus Holder Examples

A Component stops driving the DATA line on the negative edge of the CLK signal, and allows the bus holder to maintain the signal level. This procedure avoids the risk of momentary contention when different Components drive the bus on consecutive clock cycles. See Figure 7 and Section 5.2.4.3 for additional information.

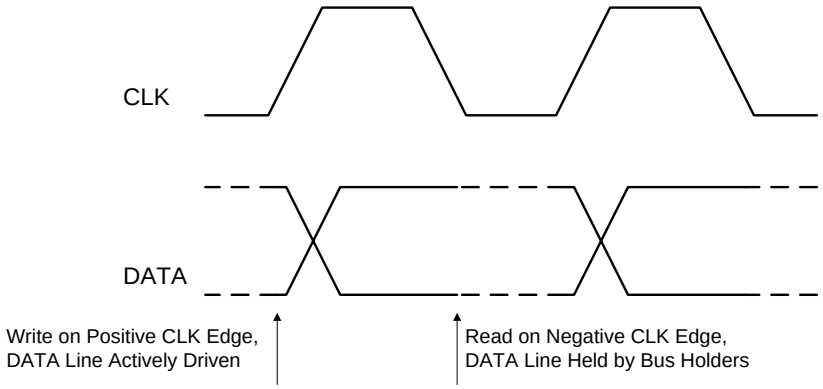


Figure 7 Output Driver is Disabled while CLK is Low

5.1 Physical Medium Independent Signaling and Logic Types

The CLK line shall be unencoded.

The DATA line shall be encoded using Non-Return-to-Zero Inverted (NRZI) coding, where a transition on the DATA line at the positive edge of the CLK line represents a logical one, while the absence of a transition represents a logical zero.

Except where otherwise specified, a Component shall transmit data by driving each bit onto the DATA line. This is known as conventional signaling.

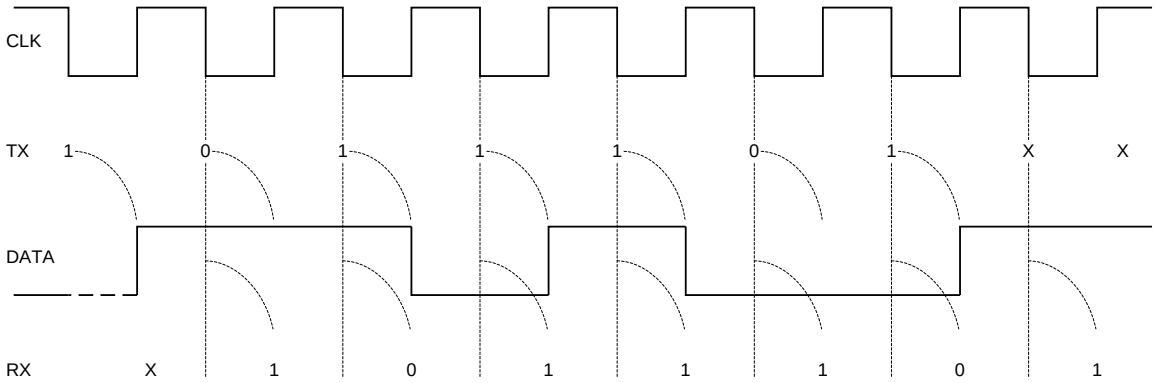


Figure 8 NRZI Signaling Example

In certain Cells, for example during arbitration and Message acknowledgement (see Section 8.4.1 and Section 8.4.2), more than one Component may simultaneously write to the DATA line. In these Cells, a Component drives the DATA line when it is writing a one but not when it is writing a zero. If no Component writes a one, the Cell is held by the bus holder at the last level driven. This method of writing to the DATA line is known as Logical-OR signaling.

When using Logical-OR signaling, a Component shall set its output driver to high-impedance, leaving the bus holder to maintain the previous level and signaling a logical zero, or drive the DATA line to the reverse of its state as read at the previous negative edge of CLK, signaling a logical one. Multiple Components may drive the same level onto the DATA line. Note that Logical-OR signaling avoids contention between Components.

Figure 9 shows an example with two Components that are both writing to the DATA line simultaneously using Logical-OR signaling. Driver 1 transmits 0b100 while Driver 2 transmits 0b101. With conventional signaling the two output drivers would contend the final bit with indeterminate results. With Logical-OR signaling, the possibility of contention is eliminated and the Component that drives a logical one always succeeds.

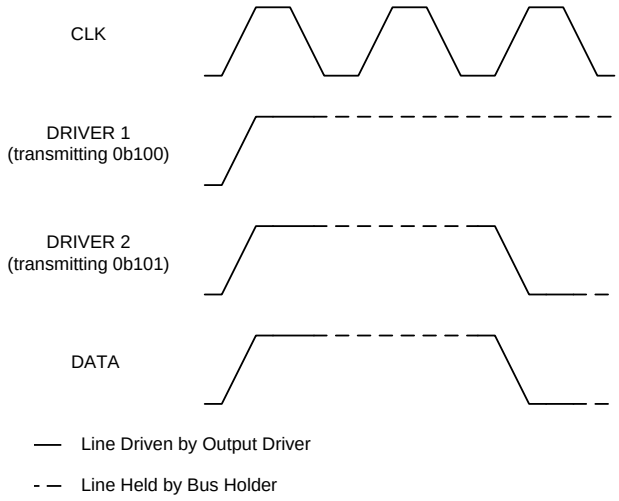


Figure 9 Logical-OR NRZI Signaling Example

5.1.1 Collision Detection

A Component shall implement Collision detection by monitoring the behavior of the DATA line while transmitting data on the SLIMbus. When a Component drives the DATA line, it shall read the line state,

LOW or HIGH, on the falling edge of the CLK signal. If the transmitted and received states differ, the Component shall record the occurrence of a Collision as described in Section 10.2.

A Component can cause a Collision only when it is driving the DATA line. Note that in Logical-OR signaling, transmitting a logical zero does not involve driving the DATA line (Section 5.1). Thus even arbitration and Message acknowledgement are normally Collision-free.

Collisions can have various physical causes, e.g. two Devices driving the DATA line simultaneously, clock latency from Component to Component, slow DATA line reflections, CLK line glitches, incorrect Component synchronization, incorrect programming of Segment locations and DATA line noise.

5.2 Physical Medium Dependent Specifications

A SLIMbus Component drives the CLK and DATA lines using CMOS-like signals, i.e. single-ended, ground referenced, rail-to-rail, voltage mode signals. Therefore, electrical specifications in this document are given relative to the supply voltage, VDD. The CLK and DATA terminals shall use the same signaling levels.

5.2.1 Signaling Voltages

The CLK terminal should be designed to operate even in the presence of repeated reflection-induced transient overvoltage and undervoltage conditions corresponding to 1.1*VDD and -0.1*VDD behind a 33 Ω resistance.

The DATA terminal should be designed to operate even in the presence of repeated reflection-induced transient overvoltage and undervoltage conditions corresponding to 1.5*VDD and -0.5*VDD behind a 33 Ω resistance.

This document uses 1.2 V and 1.8 V signaling to describe various parameters. However, no requirement is made on VDD, and therefore, the bus signaling voltages, used by a Component. Nonetheless, it is recommended that a SLIMbus Component support 1.2 V or 1.8 V signaling.

A Component using 1.8 V signaling shall meet the requirements given in Table 1.

Table 1 Parameters for Components using 1.8 V Signaling

Symbol	Parameter	Min	Max	Units
VDD	SLIMbus Supply Voltage	1.65	1.95	V
VIL	Input-LOW Voltage, settled	-0.1*VDD	0.35*VDD	V
VIH	Input-HIGH Voltage, settled	0.65*VDD	1.1*VDD	V

Additionally, the Component shall have the static output electrical characteristics shown in Table 2.

Table 2 Static Electrical Characteristics for 1.8 V Signaling

Symbol	Parameter	Condition	Min	Max	Units
VOL	Output-LOW Voltage	I_{OL} = 1 mA	0	0.1*VDD	V
VOH	Output-HIGH Voltage	I_{OH} = -1 mA	0.9*VDD	VDD	V
CIO	IO Capacitance			5	pF

The CIO should be designed to be below 3 pF for most cases. CIOmax allows for larger geometry, high voltage-capable devices

A Component using 1.2 V signaling shall meet the requirements given in Table 3.

Table 3 Parameters for Components using 1.2 V Signaling

Symbol	Parameter	Min	Max	Units
VDD	SLIMbus Supply Voltage	1.1	1.3	V
VIL	Input-LOW Voltage, settled	-0.1*VDD	0.35*VDD	V
VIH	Input-HIGH Voltage, settled	0.65*VDD	1.1*VDD	V

Additionally, the Component shall have the static output electrical characteristics shown in Table 4.

Table 4 Static Electrical Characteristics for 1.2 V Signaling

Symbol	Parameter	Condition	Min	Max	Units
VOL	Output-LOW Voltage	I_{OL} = 1 mA	0	0.1*VDD	V
VOH	Output-HIGH Voltage	I_{OH} = -1 mA	0.9*VDD	VDD	V
CIO	IO Capacitance			5	pF

The CIO should be designed to be below 3 pF for most cases. CIOmax allows for larger geometry, high voltage-capable devices

5.2.2 CLK Terminal Output Specifications

The clock signal is transmitted from the active Framer to all Components. The Component that contains the active Framer is called the Clock Sourcing Component. As the Clock Sourcing Component can be changed dynamically to any Framer on the bus, the physical location of the clock driver on the CLK line cannot be realistically constrained.

To reduce active power consumption the CLK line is typically not terminated. To avoid reflections and overvoltage problems on such a system, the CLK line shall be slew rate controlled. This constraint makes a conventional CMOS IO unsuitable for driving the CLK line at high speeds but it is possible to drive the CLK line at low speeds using a carefully matched CMOS driver. This document does not mandate a maximum output frequency for a Framer.

The clock driver is the circuit in every Framer that is responsible for driving the CLK signal. Components that do not implement a Framer do not have to implement any form of clock driver. When a Framer is not the active Framer, the clock driver shall be disabled. In such configurations the CLK terminal of the Device shall behave according to the CLK input specifications only. The clock driver output and input should share the same physical terminal on the Component as this reduces external routing overheads and corresponding loads.

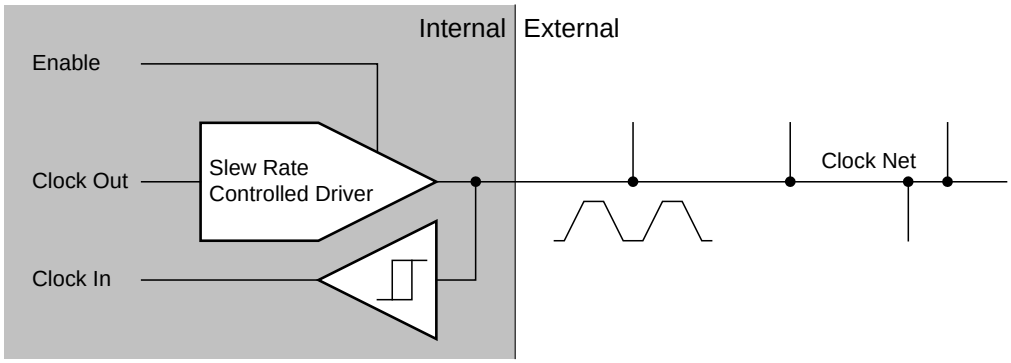


Figure 10 Recommended Clock Driver Implementation

The ability to select between different Framers on the bus according to dynamic system requirements means that Framers do not need to be capable of generating clock signals for all frequencies required by the system. The Component manufacturer shall specify the capabilities of Framers to allow the system designer to both select suitable Components and choose between multiple Framers within a system to optimize bandwidth and CLK line quality requirements.

The CLK line shall be treated as an unterminated transmission line. Fast edges on the clock signal will introduce reflections on the CLK line, degrading CLK line quality and could potentially generate glitches in a receiving Device. To avoid this problem, the slew rates of the CLK line must be carefully controlled.

The maximum allowable slew rate on the CLK line is directly related to the length of the longest end-to-end path on the CLK line. The longer this length, referred to as the bus diameter, the lower the slew rate needed to generate a reliable clock signal. Generally, the bus diameter is expected to be less than 20 cm, which results in a minimum transition time of 4 ns. To allow larger bus diameters, lower slew rates may be used. However, terminated topologies might be required for bus diameters over 50 cm.

When driving the test load specified in Section 5.2.2.1 the CLK line driver shall conform to the timing characteristics shown in Table 5.

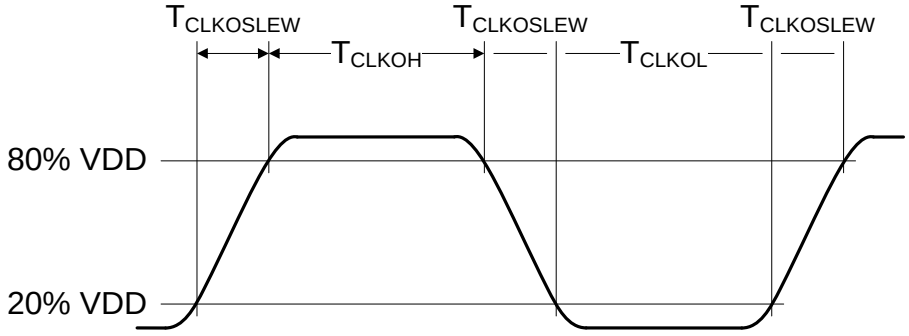


Figure 11 Clock Driver Output Waveform Constraints

Table 5 Clock Output Timing Characteristics

Symbol	Parameter	Condition	Min	Max	Units
T_{CLKOH}	Clock Output High Time		12	—	ns
T_{CLKOL}	Clock Output Low Time		12	—	ns
SR_{CLK}	Clock Output Slew Rate	20%<VO<80%	0.02*VDD	0.2*VDD	V/ns

The rise and fall times of the clock driver constrain both the maximum operating frequency and length of the CLK line. Manufacturers shall specify the Framer maximum and minimum slew rates into the specified test load.

The Framer may be configured outside of the SLIMbus via software, or any other method, with details of the load that will be applied to the CLK and DATA lines, thus allowing basic CMOS drivers with selectable output strengths to be used for low frequency operation of the SLIMbus.

5.2.2.1 Clock Driver Test Load

The clock driver will typically be presented with a reactive load. To avoid the generation of reflections, the voltage on the CLK line must be brought up and down slowly. This is tested by ensuring the maximum and minimum slew rates are adhered to into lumped models of the largest and smallest loads, represented by 15 pF and 75 pF. This constraint alone does not guarantee monotonic operation into all possible loads so the Framer should be designed to drive unterminated transmission lines of lengths of at least 20 cm while keeping the edges within the SR_{CLK} specification.

5.2.3 CLK Terminal Input Specifications

Each Component's set of timing requirements for correct operation shall meet the constraints given in Table 6. Its respective Min limits shall be no higher than the values in the table.

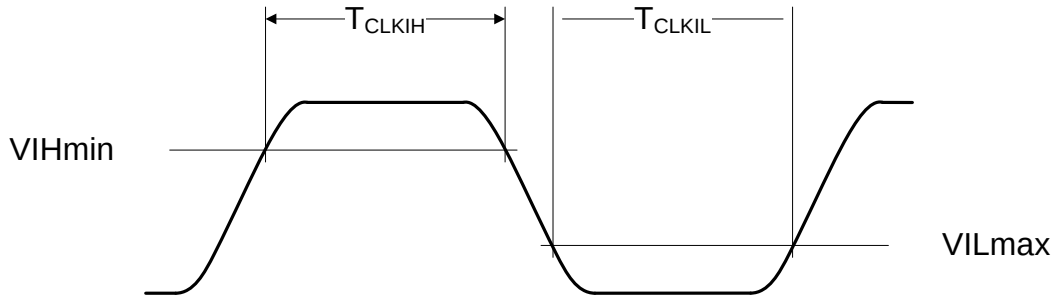


Figure 12 Received Clock Signal Constraints

Table 6 Clock Input Timing Requirements

Symbol	Parameter	Condition	Min	Max	Units
T_{CLKIH}	CLK Input High Time		12	—	ns
T_{CLKIL}	CLK Input Low Time		12	—	ns
SR_{CLKI}	Clock Input Slew Rate	20%<VI<80%	0.02*VDD	—	V/ns

Table 7 Clock Input Electrical Characteristics

Symbol	Parameter	Condition	Min	Max	Units
H_{CLKI}	CLK Input Hysteresis		50	—	mV

Use of a glitch rejection filter on the CLK input is optional. The glitch filter shall not in any way impact the system timing, i.e. the delay through any glitch rejection filter on the CLK input must be included in the T_{DV} constraint.

The CLK input shall be capable of receiving slowly changing edges without glitching.

5.2.4 DATA Terminal

As the DATA line is common to all Components it is possible during error conditions that two or more Components could drive the DATA line simultaneously with opposing values. To moderate such contention, the transmit driver of each DATA terminal shall be current limited.

The DATA terminal of a SLIMbus Device makes use of a modified CMOS IO structure that incorporates a bus holder and a method of ensuring the slew rate constraints are met into all loads. SLIMbus does not require the DATA signal to be used as a clock and consequently the DATA input does not require a glitch filter.

5.2.4.1 DATA IO Electrical Specifications

The DATA line IO shall meet the electrical specifications in Table 8.

Table 8 DATA Output Electrical Characteristics

Symbol	Parameter	Condition	Min	Max	Units
I_{SHORT}	Short Circuit Output Current	Tied to opposite rail	-100	100	mA

The output impedance of the DATA output driver is typically of the range 25 to 100 Ω but will vary with driver topology and may vary dynamically. The output impedance is constrained by both timing and a hard current limit to avoid serious damage potentially caused by momentary contention in error conditions.

The DATA terminal shall be over/undervoltage protected to allow for typical overshoots caused by possible fast rise times on long transmission lines. If the IO cannot receive signals in the range depicted by VIT_{DATA} it shall include “catch diodes” or similar device(s) to limit the local input voltage presented to the Component.

The maximum slew rate, SR_{DATA}max, is specified for a single Component driving the test load described in Section 5.2.4.2. The slew rate observed on the DATA line can be greater than SR_{DATA}max when multiple Components are driving the bus during Logical-OR signaling.

The output driver of a DATA terminal shall drive the test loads specified in Section 5.2.4.2 with the dynamic specifications shown in Table 9.

Table 9 DATA Output Timing Characteristics

Symbol	Parameter	Condition	Min	Max	Units
SR_{DATA}	Data Output Slew Rate	20%<VO<80%	—	0.5 * VDD	V/ns
T_{DV}	Time for Data Output Valid			12	ns

Care should be taken to ensure that any additional loading on the CLK or DATA lines, e.g. passive devices used to reduce EMI or reflections, is accounted for in the system timing budget.

A SLIMbus system shall meet the system level timing constraints in Table 10.

Table 10 DATA Input Timing Requirements

Symbol	Parameter	Condition	Min	Max	Units
T_H	Data Input Hold Time	measured with respect to VILmax	2	—	ns
T_{SETUP}	Data Input Setup Time	measured with respect to VIHmin	3.5	—	ns

T_{SETUP} timing shall be measured with respect to VIHmin of the Input Clock (see Figure 17). T_H timing shall be measured with respect to VILmax of the Input Clock (not shown). See Section 5.2.1 f or definitions of VIHmin and VILmax.

5.2.4.2 DATA Driver Test Load

The DATA signal does not need to be monotonic and as such capacitive test loads of 15 pF and 75 pF shall be used to verify the Slew rate, transient voltage, and transition time specifications.

5.2.4.3 Additional Timing Constraints

At some point between the SLIMbus CLK negative edge and the first possible positive edge, the transmitter shall disable its active driver and revert to its bus holder. The active driver shall meet the T_{DD} specification in Table 11.

Table 11 Driver Disable Timing Specification

Symbol	Parameter	Min	Max	Units
T_{DD}	Driver Disable Time	—	10	ns

The bus holder is able to hold the value on the bus as long as the bus remains in a powered state. As the clock transition times can be very large the data can be sampled by receivers when driven or when being held by the bus holders.

5.2.5 Bus Holder Specifications

The DATA IO shall contain a bus holder; this is specified in Table 12 as a minimum and maximum output impedance.

Table 12 Bus Holder Electrical Specification

Symbol	Parameter	Condition	Min	Max	Units
R_{DATAS}	Bus Holder Output Impedance	0.1*VDD < V < 0.9*VDD	10k	50k	Ω

This effectively creates a mask in the IV characteristic as shown in Figure 13.

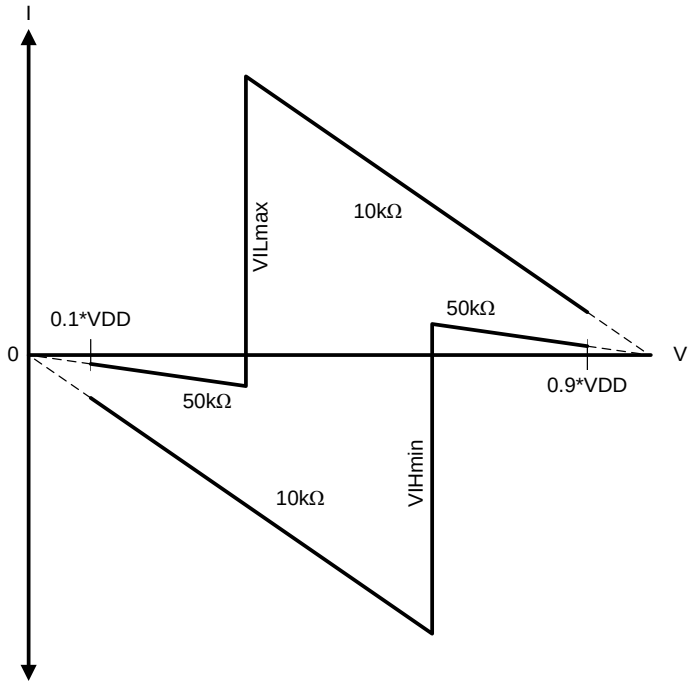


Figure 13 Bus Holder I/V Mask

All CLK and DATA lines implementing a bus holder shall comply to the bus holder I/V mask as shown in Figure 13. Consequently, this specification applies to all DATA terminals as well as any CLK terminals that implement a bus holder. A CLK line IO that does not implement a bus holder shall meet the leakage specification in Table 13.

Table 13 Input Leakage Constraints

Symbol	Parameter	Condition	Min	Max	Units
IL_{CLK}	Input Leakage Current	VI=0.1*VDD, VI=0.9*VDD	-5	5	µA

5.2.6 Component Power Down on an Active Bus

The Physical Layer of a Component may be designed so that it can be powered down while the bus remains active. While in a powered down state, or during the change to or from a powered down state, the CLK and DATA terminals of such a Component shall meet the requirements for IO capacitance and input leakage current given in Section 5.2.1 and Section 5.2.5, respectively.

5.2.7 System Timing Budget (informative)

The maximum frequency that the bus can operate is a physical constraint that depends on the length of the bus, the loads attached to the bus and the capabilities of the Framer. In sensibly designed systems SLIMbus can operate at all common system frequencies up to 20 MHz. Operation is possible beyond this but considerable care is required on the part of the system designer to guarantee that signal integrity and proper timing is maintained. The maximum frequency allowed by this specification is constrained by the clock driver quality and the internal timing of the data paths and the length of the bus.

As the DATA line is always driven on the positive edge, and sampled on the negative edge, of the CLK line, the timing is determined by the CLK high phase. The worst-case timing budget occurs when the Component that sees the latest CLK line positive edge transmits to the Device that sees the earliest CLK

line negative edge. Due to differences in trigger levels this does not necessarily correspond to the physical positions of the Devices on the CLK line.

5.2.7.1 Clock Uncertainty and Jitter

The maximum allowable CLK line uncertainty consists of transmission line delay, T_{SKEW}, longest rise or fall time at the receiver, $T_{CLKISLEW}$, and jitter effects, T_{JITTER}.

T_{SKEW} is directly related to the length of the longest path the CLK signal has to make. This is not the same as the bus diameter as the active Framer may be located in the center of the bus. The longest possible length is the bus diameter L and occurs when the Framer is located at the end of the CLK line. The transmission line delay specification applies at all voltages on the transition between VILmax and VIHmin at the CLK terminal of each Component on the bus.

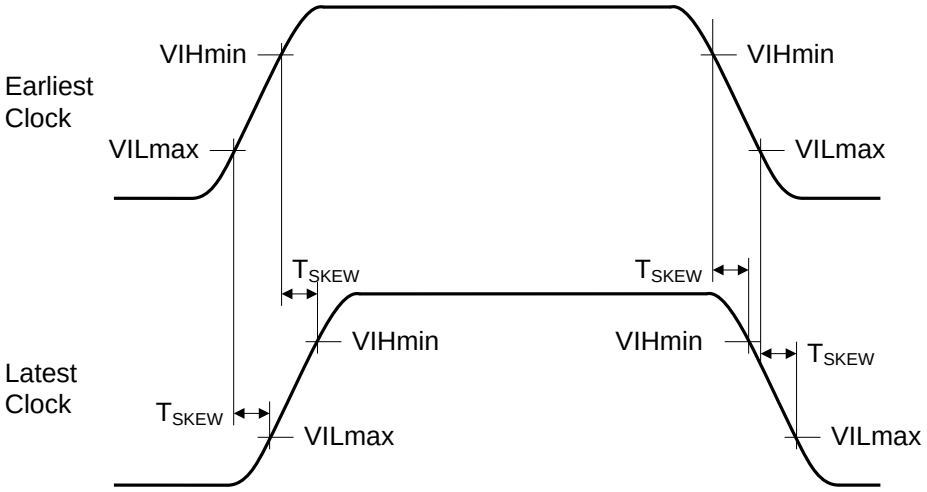


Figure 14 T_{SKEW} Specification

For a typical PCB, the transmission line delay, T_D, varies from about 60 to 75 ps per cm. Therefore, T_{SKEW} can be described as:

$$T_{SKEW} = L_{CLK} * T_{D/CM} \text{ (in ps)}$$

The clock uncertainty contribution from the input transition time, $T_{CLKISLEW}$, is the delay from VILmax to VIHmin at the lowest slew rate of the active Framer and is shown in Figure 15.

Figure 15 $T_{CLKISLEW}$ Specification

These restrictions define $T_{CLKISLEW}$ as:

$$T_{CLKISLEW} = 0.3 * T_{CLKOSLEW}\text{max}$$

The jitter of the CLK signal is not mandated in this document although the peak-peak period jitter, T_{JITTER}, must be taken into account when running at high frequencies where it impacts system timing. Note that T_{JITTER} includes all sources of jitter from both the time domain (phase noise related) and voltage domain (noise).

In systems where analog-to-digital or digital-to-analog conversion quality is important, additional clock jitter considerations might apply [AES01]. Such considerations are beyond the scope of this document. However, they are reflected in the clock QUALITY Information Element that is described in Section 12.4.5.1.

The combined clock uncertainty is referred to as $T_{CLKUNCERT}$ and is shown in Figure 16. $T_{CLKUNCERT}$ is taken to be the larger of the uncertainties calculated from the rising or negative edge of the CLK line.

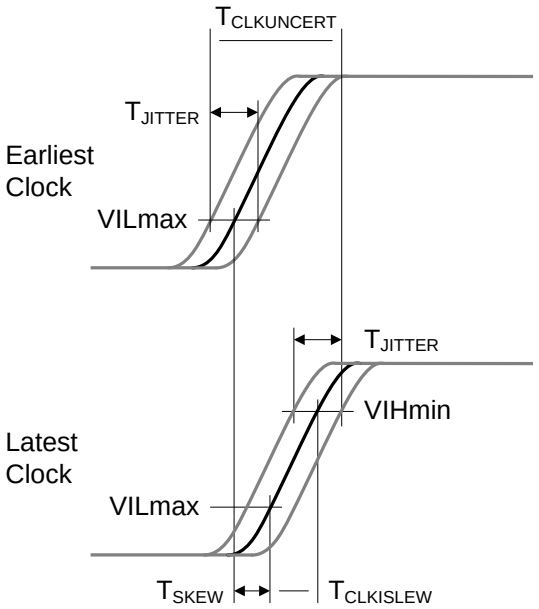


Figure 16 $T_{CLKUNCERT}$ Specification

$$T_{CLKUNCERT} = T_{SKEW} + T_{CLKISLEW} + T_{JITTER}$$

Extra margin should be added to the clock uncertainty to allow for noise on the CLK line and differences in the supplies and grounds from Component to Component.

5.2.7.2 Time to Drive Data Valid

The time to drive the data valid, T_{DV}, is the time for the data to become valid from the time the transmitter receives the positive edge of the CLK line. For a CMOS IO this is typically quoted into a fixed load. For SLIMbus, this specification must be met into the range of test loads. T_{DV} can be broken down into the Component-specific (internal) delays and the line-specific (external) delays:

- T_{DO} is the time from the positive edge of the CLK line on the transmitter to the point when the voltage on the DATA terminal of the transmitter is higher than VOL_{DATA} and lower than VOH_{DATA}.
- T_{SETTLE} is the time required for the voltage at all points on the DATA line to be higher than VOH_{DATA} or lower than VOL_{DATA}. This includes all transmission line delays.

T_{SETTLE} has an effect on the overshoot and EMI aspects of the SLIMbus system implementation. Overshoot and EMI increase as T_{SETTLE} becomes smaller.

5.2.7.3 Setup Time

T_{SETUP} is the minimum receiver setup time, measured at the receiver terminals. The signal on the DATA terminal is stable for the period of the setup time from the local negative edge to allow the receiver to accurately sample its state.

5.2.7.4 Timing Budget Calculation

The DATA line must be driven to a stable value in a time equivalent to the difference from the latest possible positive edge to the first possible negative edge, less the receiver setup time as shown in Figure 17.

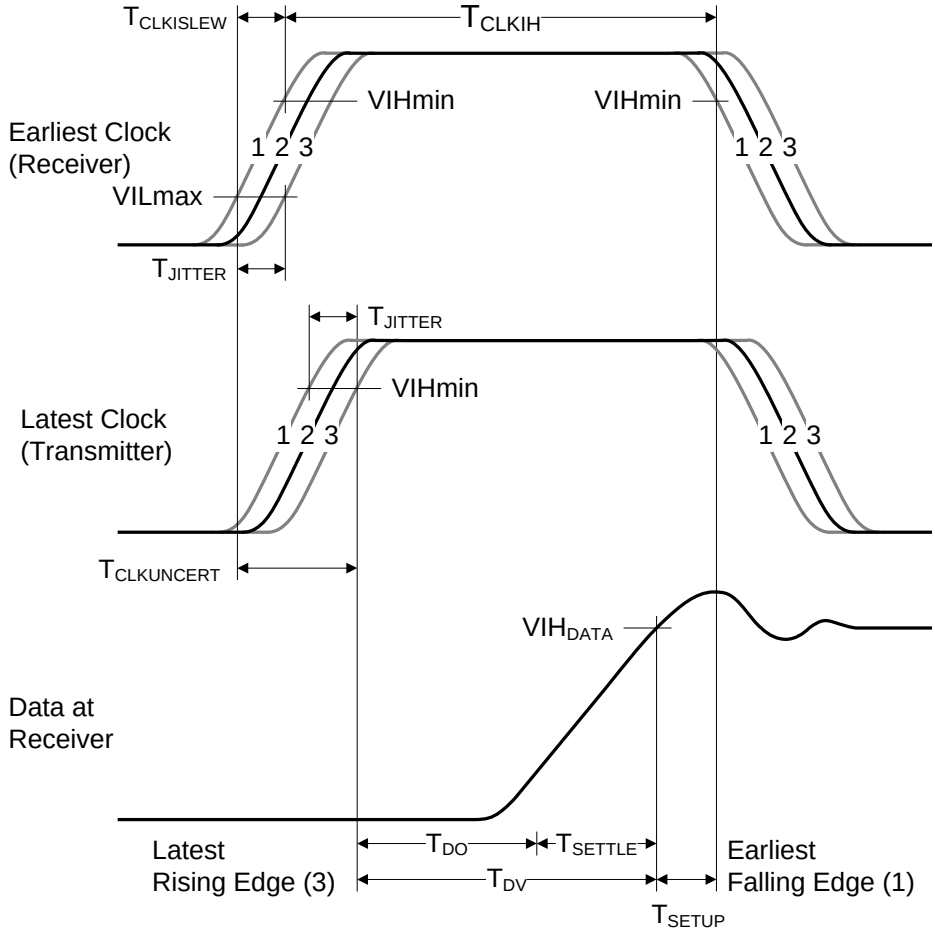


Figure 17 Data Transmit Timing Constraints

From this diagram it can be seen that the time available to transmit, $T_{CLKIH} + T_{CLKISLEW}$, must be long enough to accommodate clock uncertainty, time for the data to settle to a valid state on the DATA line and setup time:

$$T_{CLKIH} + T_{CLKISLEW} > T_{CLKUNCERT} + T_{DV} + T_{SETUP}$$

The clock uncertainty and waveform constraints will vary depending on the quality of the active Framer.

5.2.7.5 Timing Budget Examples

When operating the SLIMbus at frequencies above 20 MHz the timing budget becomes gradually more critical. This section describes how to calculate the timing budget for system designers who require operation of a SLIMbus at high frequencies.

The timing budget for SLIMbus consists of fixed delays that are directly limited in the specification and variable delays. An example of the timing budget at 20 MHz is provided in the following equations. It is assumed from these examples that the clock driver is 1.8 V and has a slew rate range of 0.12 to 0.45 V/ns (allowing a 20 cm bus and 9 ns maximum $T_{CLKOSLEW}$).

Example of Timing Budget at 19.2 MHz, 20 cm, 75 pF:

$T_{CLKUNCERT}$ = 7 ns ($T_{CLKISLEW}$ = 4.5 ns, 1.5 ns for T_{SKEW}, 0.5 ns for Jitter, 0.5 ns for some margin to cope with noise)

T_{DV} = 12 ns (7 ns T_{SETTLE} and 5 ns for T_{DO})

T_{SETUP} = 3.5 ns

Total transmit time = 22.5 ns

Total Available time = T_{CLKIH} + $T_{CLKISLEW}$ ≈ 26 ns (For identical linear slopes and 50/50 Duty cycle this should end up as 50% of a full clock cycle).

This leaves 3 ns for deviations in duty cycle, noise, extra jitter…

If operation at 26 MHz is required, T_{CLKIH} falls from 22.5 ns to 17.3 ns. To allow the bus to run at this speed the length or bus capacitance can be reduced at the system designer's discretion. This allows for a reduction in T_{SETTLE}.

Example of Timing Budget at 26 MHz, 10 cm, 35 pF:

$T_{CLKUNCERT}$ = 6.5 ns ($T_{CLKISLEW}$ = 4.5 ns, 1 ns for T_{SKEW}, 0.5 ns for Jitter, 0.5 ns for margin to cope with noise)

T_{DV} = 7 ns (3 ns T_{SETTLE} and 4 ns for T_{DO})

T_{SETUP} = 3.5 ns

Total transmit time = 17 ns

Total Available time = T_{CLKIH}+$T_{CLKISLEW}$ = 19.23 ns

For reliable transmission at 26 MHz less noise and jitter would be advisable.

With a lower bus capacitance it is likely that $T_{CLKISLEW}$ could also be reduced. T_{SKEW} could also be reduced, relaxing the timing budget further. Device manufacturers can provide SLIMbus compatible Devices that have tighter $T_{CLKISLEW}$, T_{DO}, and T_{SETUP} parameters to simplify the design of buses that operate to the maximum frequency permitted in Gear 10, 28.8 MHz.

If device designers struggle to meet the T_{SETUP} specification, for example on large devices with shared terminals, it should be noted that an internal delay on the inverted clock to the data sampling register directly increases setup time with no external effects.

At the other extreme a less well constrained clock driver can be used at limited speeds. Imagine a 1.8 V clock driver with a specified slew rate of 0.06 V/ns to 0.3 V/ns, giving minimum and maximum $T_{CLKOSLEW}$ times of 3.6 ns and 18 ns. This Framer can be used with a CLK line of up-to 30 cm at speeds up to 11.9 MHz.

Example of Timing Budget at 12.288 MHz, 30 cm, 75 pF:

$T_{CLKUNCERT}$ = 15 ns ($T_{CLKISLEW}$ = 10 ns, 2.5 ns for T_{SKEW}, 1 ns for Jitter, 1.5 ns for some margin to cope with noise)

T_{DV} = 12 ns (7 ns T_{SETTLE} and 5 ns for T_{DO}).

T_{SETUP} = 3.5 ns

Total transmit time = 30.5 ns

Total Available time = T_{CLKIH} + $T_{CLKISLEW}$ ≈ 40 ns (For identical linear slopes and 50/50 Duty cycle this should end up as 50% of a full clock cycle)

6 Frame Structure

The SLIMbus bit stream is organized in a Time Domain Multiplexed (TDM) configuration. This organization allows SLIMbus to carry control and data information as well as bus configuration information. Throughout this document the organization of information on the bus is called the Frame Structure.

At the highest level of abstraction, the Frame Structure is split into two regions known as Control Space and Data Space. Control Space carries bus configuration and synchronization information as well as other inter-Device communication. Data Space carries application-specific information such as isochronous, near-isochronous and asynchronous data streams. Control Space and Data Space can be further divided into channels, with each channel representing a particular information flow. The relative bandwidth used by the Control Space and Data Space is configurable so that the bus can be adapted to virtually any application. Refer to Section 6.3.1.4 and Section 10.6 for more details on the partitioning of the Control Space and Data Space.

At a more fundamental level, the Frame Structure is composed of five building blocks: Cells, Slots, Subframes, Frames and Superframes. Section 6.1 describes these building blocks in detail.

6.1 Cells, Slots, Subframes, Frames, and Superframes

SLIMbus uses a synchronous, two-wire bus to carry information between Devices. The following sections describe several conceptual parts to aid the reader in understanding how the Frame Structure is imposed on the bus signals.

6.1.1 Cell

The smallest subdivision of a SLIMbus data flow is the Cell. A Cell is defined as a region of the DATA signal that is bounded by two consecutive positive edges of the CLK line. Each Cell can hold a single bit of information.

6.1.2 Slot

A Slot is defined as four contiguous Cells and is the unit of bandwidth allocation on SLIMbus. Cells within a Slot are labeled C0 through C3.

Cells within a Slot shall be transmitted C3 first, followed by C2, C1 and C0, in that order.

6.1.3 Subframe

A Subframe is defined as the division of the Frame Structure at which Control Space and Data Space are interleaved. The Subframe length is programmable and shall be six, eight, twenty-four or thirty-two contiguous Slots (24, 32, 96 or 128 Cells). Slots within a Subframe are labeled S0 through S(n-1), where n is the Subframe length.

Slots within a Subframe shall be transmitted S0 first, followed by S1, S2,… S(n-1), in that order.

At least one Slot, and as many as all Slots, of a Subframe may be allocated to Control Space. Any Slots not allocated to Control Space are considered Data Space. See Section 6.3.1.4 for more information.

6.1.4 Frame

A Frame is defined as 192 contiguous Slots. Slots within a Frame are labeled S0 through S191.

Slots within a Frame shall be transmitted S0 first, followed by S1, S2,… S191, in that order.

The first Slot (Slot 0) of each Frame shall contain the Frame Sync symbol that a Component can use to synchronize to the bus. Slot 96 of each Frame shall contain four bits of Framing Information. See Section 6.3.1.1 for a description of Framing Information.

6.1.5 Superframe

A Superframe is defined as eight contiguous Frames (1536 Slots). Frames within a Superframe are labeled Frame 0 through Frame 7.

Frames within a Superframe shall be transmitted Frame 0 first, followed by Frame 1, Frame 2,… Frame 7, in that order.

Each Frame of a Superframe shall contain the Frame Sync symbol in Slot 0. In addition, the first Frame (Frame 0) of a Superframe shall contain the first four bits of Framing Information in Slot 96. Frames 1 through Frame 7 shall also contain four bits of Framing Information in Slot 96 with Frame 7 carrying the last four bits of Framing Information. Since a Superframe contains a complete set of Framing Information, a Component uses the Framing Information for Superframe synchronization. See Section 6.3.1.1 for a description of Framing Information.

6.1.6 Relationship of Frame Structure Constructs

Unlike the other basic Frame Structure constructs, Subframes do not have a single, fixed length. For any given bus configuration, the Subframe length is set to one of four possible values: 6, 8, 24, or 32 Slots. Consequently, the number of Subframes per Frame depends on the bus configuration. Table 14 summarizes the relationships between Cells, Slots, Subframes, Frames, and Superframes for the different Subframe lengths. The shaded table entries highlight the values that vary with the Subframe size.

Table 14 Relationship of the number of Cells, Slots, Subframes, Frames, Superframes

6-Slot Subframe	Cells	Slots	Subframes	Frames	Superframes
Cell	1	—	—	—	—
Slot	4	1	—	—	—
Subframe	24	6	1	—	—
Frame	768	192	32	1	—
Superframe	6144	1536	256	8	1
8-Slot Subframe	**Cells**	**Slots**	**Subframes**	**Frames**	**Superframes**
Cell	1	—	—	—	—
Slot	4	1	—	—	—
Subframe	32	8	1	—	—
Frame	768	192	24	1	
Superframe	6144	1536	192	8	1

24-Slot Subframe	Cells	Slots	Subframes	Frames	Superframes
Cell	1	—	—	—	—
Slot	4	1	—	—	—
Subframe	96	24	1	—	—
Frame	768	192	8	1	—
Superframe	6144	1536	64	8	1
32-Slot Subframe	**Cells**	**Slots**	**Subframes**	**Frames**	**Superframes**
Cell	1	—	—	—	—
Slot	4	1	—	—	—
Subframe	128	32	1	—	—
Frame	768	192	6	1	—
Superframe	6144	1536	48	8	1

6.1.7 Integrating the Concepts (informative)

Figure 18 illustrates the SLIMbus Frame Structure from the Cell level up to the Superframe level. The values in the Cells and Slots are examples, as is the division between the Control Space and the Data Space. Note that the figure represents only one possible bus configuration.

Control Space appears at the beginning of every Subframe and it only appears once per Subframe. This is the defining feature of the Subframe – it is the division with which the Control Space is interleaved within the Frame Structure. The Framing Channel, Guide Channel and Message Channel make up Control Space. Note that the Framing Channel has two components per Frame, the Frame Sync symbol and Framing Information.

The Frame Structure regains strict regularity at the Frame level. Each Frame contains 192 Slots and each Superframe contains eight Frames. Figure 18 does not attempt to distinguish between Control Space and Data Space in its representation of Frames or Superframes. However, the figure still shows the Framing Channel at the highest levels since it is a consistent and useful marker in the Frame Structure.

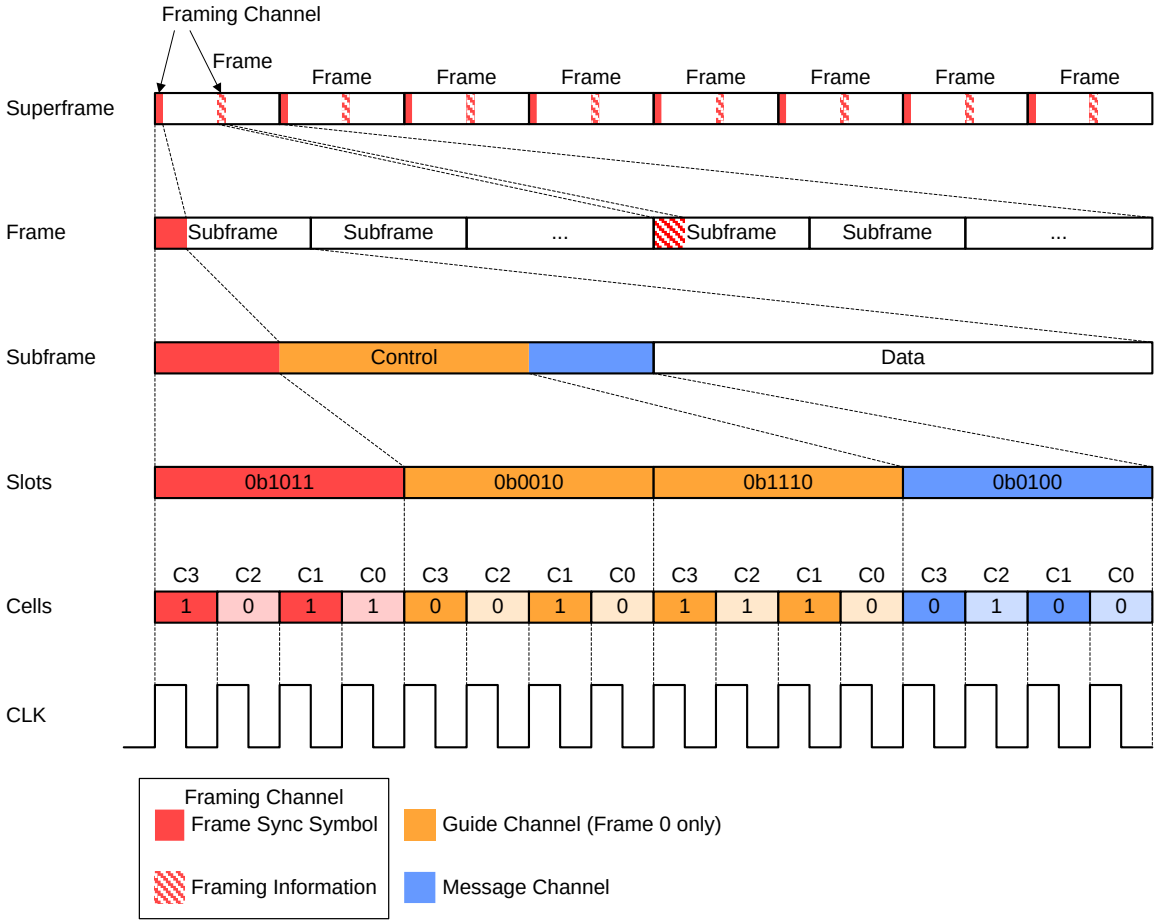


Figure 18 Overview of the SLIMbus TDM Scheme

6.2 Clock Gears, Frequencies, and Root Superframes

This document does not specify a particular CLK frequency. The CLK frequency can be set according to the application needs.

While a Superframe always consist of 1536 Slots (6144 Cells), it can vary in duration. In other words, the length of a Superframe is fixed in terms of Slots, and therefore Cells, but not in terms of time. Consequently, the Superframe rate can be changed on SLIMbus.

The Superframe rate can be changed while the bus is active. The Superframe rate of an active SLIMbus can be varied by changing the Clock Gear, the Root Frequency or both. Note that these changes can take place even while Data Channels are active and Messages are being transmitted in the Message Channel.

On SLIMbus, a temporal frame of reference that is independent of the Clock Gear is provided by the Root Frequency and Root Superframe size. In the top Clock Gear, Gear 10, the Root Frequency and Root Superframe size correspond to the actual CLK frequency and the actual Superframe size, respectively. Lowering the Clock Gear halves the CLK frequency and doubles the duration of a Superframe, but leaves the Root Frequency and Root Superframes unaltered. This is useful, for example, in minimizing the bus power consumption.

6.2.1 Clock Gears

Clock Gears provide a range of power-of-two frequency steps for operating SLIMbus and establish a framework for easily balancing bus bandwidth and power consumption. Increasing to the next Clock Gear without changing the Root Frequency shall double the SLIMbus CLK frequency. There are ten Clock Gears, Gear 1 to Gear 10, providing a frequency span of 512.

Note that each Clock Gear has a frequency span of 2.25x. For example, the Gear 5 frequency can range from 0.4 to 0.9 MHz. Therefore, SLIMbus CLK frequencies can vary from 0.025 (Gear 1, minimum frequency) to 28.8 MHz (Gear 10, maximum frequency) for a total span of 1152.

6.2.2 Clock Gear Example (informative)

Consider an example SLIMbus system that supports isochronous 16-bit flows at 8 kHz and 48 kHz sample rates. The Root Frequency is 24.576 MHz. The following description is a possible use case sequence in which the Clock Gear changes:

- Two 8 kHz sample rate flows are active so a CLK frequency of 384 kHz (Gear 4) is sufficient.
- Later on, two 48 kHz sample rate flows also become active. The CLK frequency is raised to 3.072 MHz by changing the Clock Gear to seven in order to accommodate the increased bandwidth requirements.
- Finally, the two 48 kHz sample rate flows become inactive leaving just the two 8 kHz sample rate flows. The Clock Gear is changed back to four, lowering the CLK frequency to 384 kHz and thus reducing power consumption.

Figure 19 shows how the Superframe rate changes as the Clock Gear is changed according to the needs of the use case.

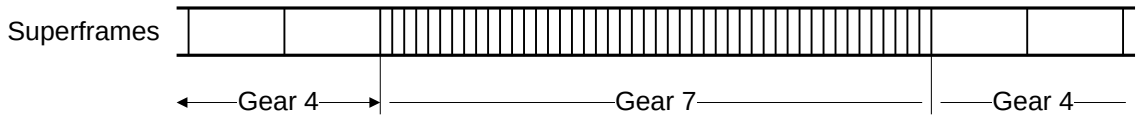


Figure 19 Clock Gear Example

This example demonstrates how SLIMbus Clock Gears can be used to minimize the bus power consumption. Instead of always running the bus at the highest required frequency for a platform, Clock Gears can be used to tune the CLK frequency for the current application.

6.2.3 Root Frequency

The Root Frequency shall be $2^{(10-CG)}$ times the frequency of the CLK line, where CG is the current Clock Gear. In Gear 10, the CLK frequency is the same as the Root Frequency.

Note that the Root Frequency indicates a Gear 10 frequency as described here, even if the active Framer does not support generation of such a frequency.

The Root Frequency may be changed while the bus is active.

6.2.4 Root Superframes

Many SLIMbus constructs and processes are defined with respect to Superframes. This includes such things as Data Channel definitions and the coordination of bus configuration changes. But the frequency of the Superframes depends on the Clock Gear, and their phasing depends on the timing of previous Clock

Gear changes. For some purposes it is more natural to use a reference framework that is independent of the Clock Gear. The concept of Root Superframes provides this framework.

Root Superframes are Clock Gear-independent divisions of the SLIMbus timeline. In Gear 10, Root Superframes have the same frequency and phasing as the bus's actual Superframes. With each step down the Clock Gears the duration of the actual Superframes progressively doubles, quadruples etc., but the Root Superframes remain unaltered. This concept is illustrated in Figure 20 through a Clock Gear change. Note that Superframe Boundaries are always also Root Superframe boundaries.

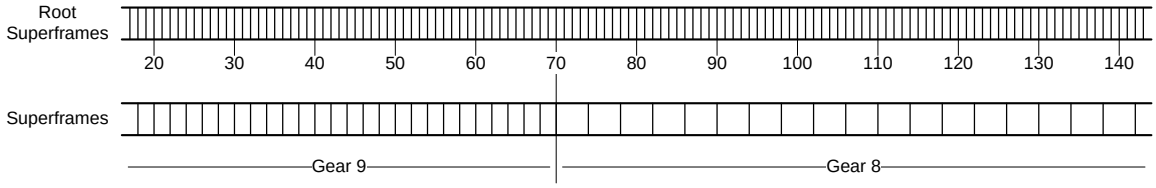


Figure 20 Root Superframe Example

Formally, Root Superframes are divisions of the SLIMbus Superframes into $2^{(10-CG)}$ equal parts, where CG is the Clock Gear. Each Root Superframe spans $6144/2^{(10-CG)}$ Cells. A consequence of these numbers is that every Root Superframe boundary falls on an actual Cell boundary, even in Gear 1.

Root Superframes are used by the Phasing Signal to communicate an embedded timing reference. See Section 6.3.1.8. Root Superframes are also used in the Locked Transport Protocol. See Section 9.3.4.

Note that the Root Superframe indicates a constant time reference based on Gear 10, even if the active Framer does not support an output CLK at Root Frequency.

6.2.5 Natural Frequencies

A Natural Frequency is defined as a CLK frequency that allows a family of data flows to be supported using the allowed channel rate multipliers. For example, Natural Frequencies for 11.025 kHz and 44.1 kHz sample rate flows include 5.6448 MHz, 11.2896 MHz, and 22.5792 MHz. Similarly, Natural Frequencies for 8 kHz and 48 kHz sample rate flows include 6.144 MHz, 12.288 MHz, and 24.576 MHz. An additional benefit is that an isochronous data flow using a permitted channel rate multiplier does not require flow control. See Table 20 for a list of the channel rate multipliers.

Data flows at rates other than a multiple of the Superframe rate can be supported using flow control at the expense of bus efficiency. Typically, a higher than necessary bandwidth is allocated to the data flow in order to use one of the prescribed multipliers, thus wasting some of the bandwidth. Also, additional Slots may be allocated for flow control signaling, thereby consuming more bandwidth. Refer to Section 9.3 for information on how flow control mechanisms work with Data Channels.

Figure 21 illustrates an example data flow on SLIMbus. In the figure, the Root Frequency has been selected so that the SLIMbus CLK frequency is a Natural Frequency for this data flow. The top timeline shows a data flow with a constant frequency. When the data flow is carried on SLIMbus as shown in the bottom timeline, the constant periodicity of the data flow is preserved. The data flow occupies each Segment of the associated Data Channel. Refer to Section 6.4.1 for details on Segments.

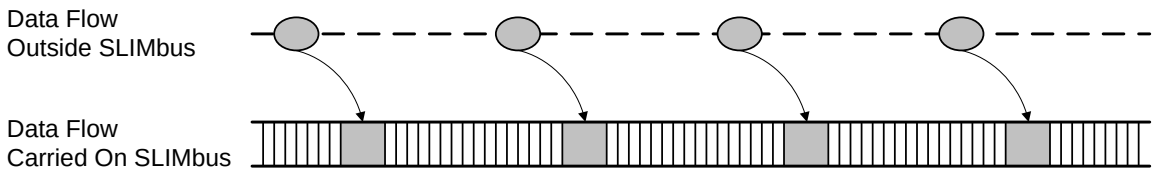


Figure 21 Natural Frequency Example

The use of Natural Frequencies is not mandated on SLIMbus. Refer to Annex C for details on Natural Frequencies across Clock Gears for each Presence Rate defined in Section 9.6.

6.2.6 Cardinal Frequencies

Natural Frequencies have properties that ease channel allocation on SLIMbus. For a given family of flows, they enable the use of a pure isochronous channel with no flow control. However, concurrent support for flows, or families of flows, with non-integer frequency relationships is sometimes required, e.g. 8 kHz and 44.1 kHz. In these cases, the CLK line cannot be set to a frequency that is a Natural Frequency for all flows on the bus.

In audio applications, one important family of sample rates is comprised entirely of multiples of 4 kHz, i.e. 8, 12, 16, 24, 32, 48, 96 kHz, etc. Another family is comprised entirely of multiples of 11.025 kHz, i.e. 11.025, 22.05, 44.1, 88.2 kHz, etc.

The CLK line frequencies 24.576 MHz, 12.288 MHz, 6.144 MHz, etc. have special significance because they can carry the 4 kHz family of flows isochronously, and the 11.025 kHz family of flows with relatively high efficiency. For this reason, these clock frequencies are referred to as Cardinal Frequencies. Note that there is only one Cardinal Frequency in each SLIMbus Clock Gear.

The use of Cardinal Frequencies is not mandated on SLIMbus.

6.3 Control Space

Control Space contains the information needed for a Device to discover the bus state (Framing Channel), to track Messages (Guide Channel), and the Messages themselves (Message Channel).

The Control Space size is programmable by setting the Control Space width (number of Control Space Slots per Subframe) and the Subframe length. The number of Control Space Slots per Frame is given by the following equation:

# of Slots per Frame = Control Space width in Slots * Frame length in Slots / Subframe length in Slots

For example, if the Control Space width is eight Slots and the Subframe length is twenty-four Slots, then the number of Control Space Slots per Frame is:

# of Slots per Frame = 8 Slots * 192 Slots per Frame / 24 Slots = 64 Slots/Frame

The number of Control Space Slots per Superframe is just eight times the number of Control Space Slots per Frame since there are always eight Frames per Superframe. In the example, there would be 512 Slots of Control Space per Superframe.

The Control Space width and Subframe length are not individually programmable, but are set as a single parameter known as the Subframe Mode. See Table 15 for the available Subframe Modes.

The Framing Channel occupies two Slots per Frame and the Guide Channel occupies two Slots per Superframe for all bus configurations. Control Space Slots not used for the Framing Channel or Guide Channel are allocated to the Message Channel.

Again, referring to the example, the number of Slots per Superframe allocated to the Message Channel is:

# of Slots per Superframe = (64 Slots/Frame – 2 Framing Channel Slots/Frame) * 8 Frames/Superframe - 2 Guide Channel Slots/Superframe = 494 Slots/Superframe

Figure 22 illustrates how Control and Data Space Slots make up the first Frame of a Superframe for one particular Subframe configuration. Guide Channel Slots are allocated only for the first Frame of a Superframe.

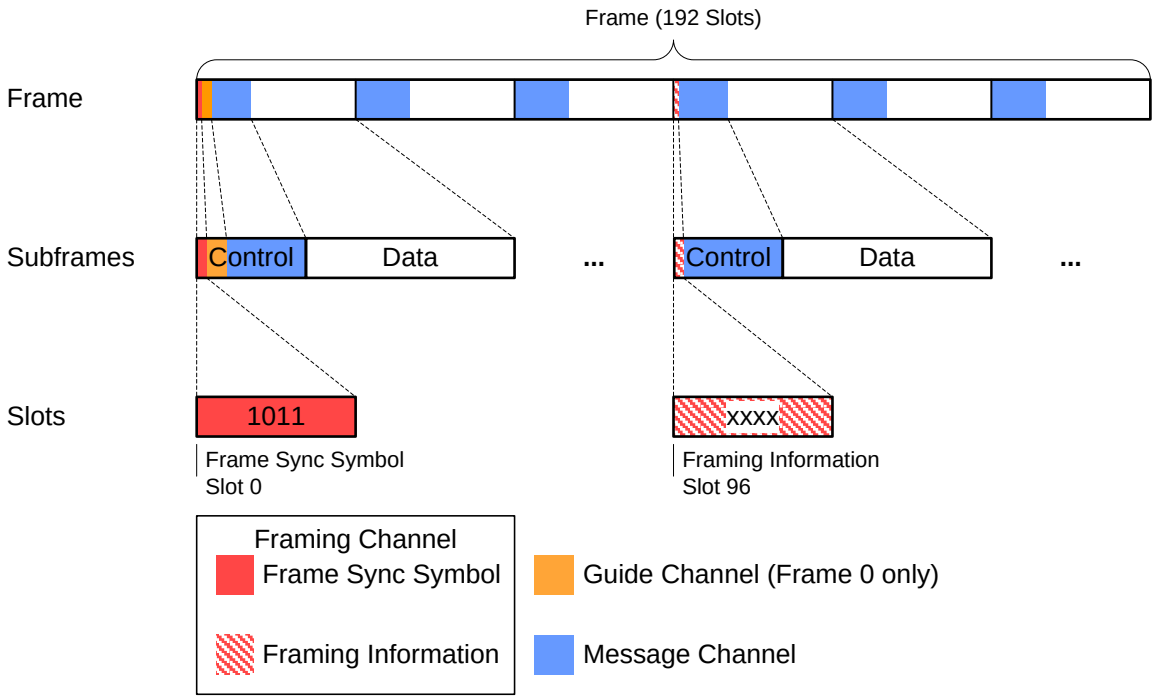


Figure 22 Example Organization of Control and Data Space Slots

6.3.1 Framing Channel

The Framing Channel provides information about the state of the bus and helps define the Frame Structure. The Frame Sync symbol and Framing Information are sent in the Framing Channel. The Framing Channel shall occupy Slot 0 and Slot 96 of each Frame.

The Frame Sync symbol is a fixed bit-sequence, 0b1011. The active Framer shall transmit the Frame Sync symbol once per Frame in Slot 0. Components shall use the Frame Sync symbol to lock onto the Frame Structure. See Section 10.1.2.3 for additional information.

The 32-bits of Framing Information contain all the data necessary for a Component to detect and verify the Subframe Mode, synchronize Device state machines and configure Components so they can properly communicate on the bus. See Section 6.3.1.1 for additional information.

The active Framer shall transmit four bits of Framing Information once per Frame in Slot 96. The complete Framing Information shall be sent in eight consecutive Frames.

Figure 23 shows the relationship between the various Subframe lengths and the Frame size. Note that Slot 0 and Slot 96 always occur at the start of a Subframe regardless of Subframe length.

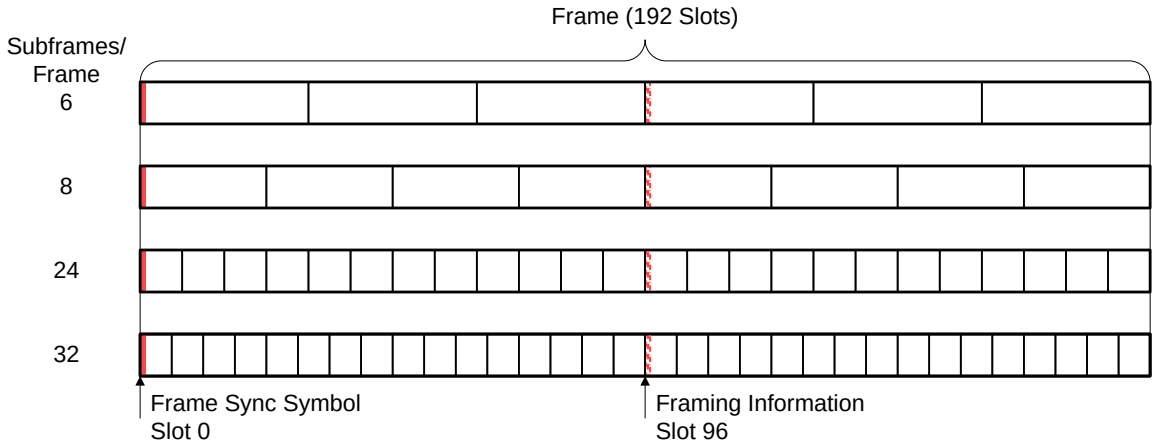


Figure 23 Overlap between Various Subframes

6.3.1.1 Framing Information

The 32-bit Framing Information is sent by the active Framer once per Superframe in eight consecutive Frames. Therefore, it takes 1536 Slots (6144 Cells) to acquire a set of Framing Information after a Component has locked onto the Frame Structure. Figure 24 shows the contents of the Framing Information Slots.

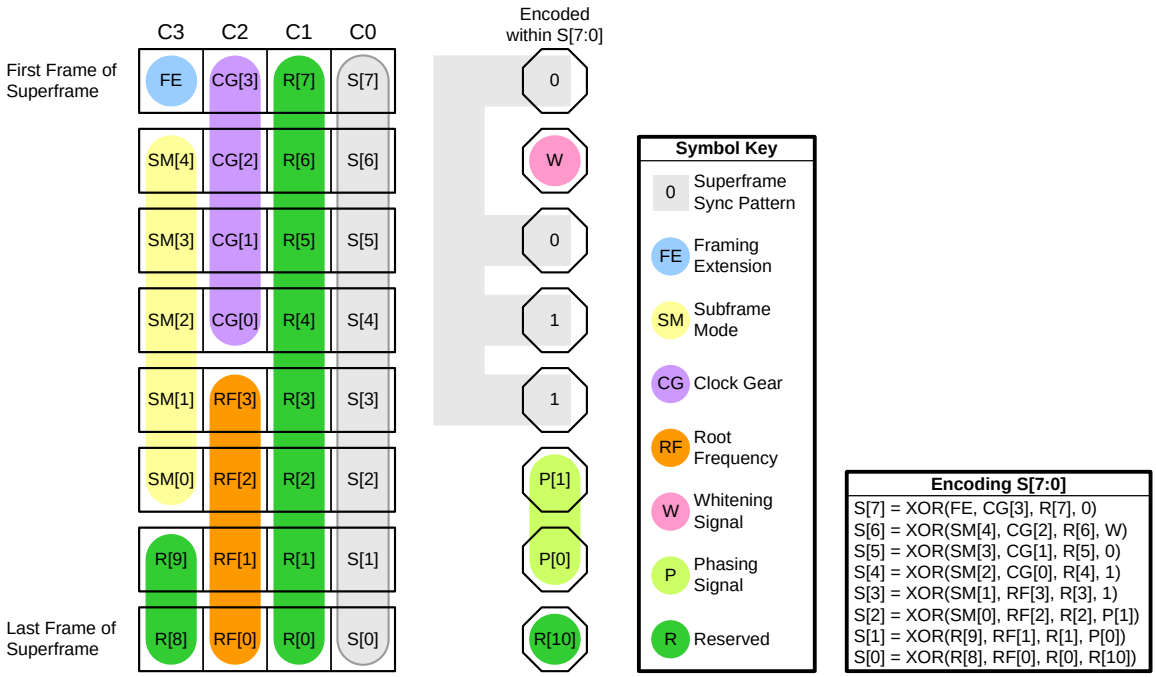


Figure 24 Distribution of Framing Information over Multiple Slots

Framing Information shall include a flag FE and fields SM, CG, RF and R as shown in Figure 24. It shall also include a stripe S into which the active Framer encodes a Superframe Sync pattern, a Whitening Signal and a Phasing Signal. The coding shall be performed via the parity of the Slot 96 nibbles. For example, in the sixth Frame of the Superframe, the active Framer shall send S[2] as the exclusive-OR of SM[0], RF[2],

R[2], and P[1]. A Receiver that uses the Phasing Signal shall decode P[1] as the exclusive-OR of SM[0], RF[2], R[2], and S[2]. Coding the stripe S moderates the worst case spectrum of the data waveform when the bus is carrying little or no traffic. This tends to reduce interference with analog circuitry. Coding of stripe S has no relation to detection and handling of errors in the Framing Information.

The active Framer shall set the reserved bits, R[10:0], to all zeros. Components reading the Framing Information shall not use the contents of the reserved bits except to decode the stripe S.

The information in fields SM, CG, and RF applies to the current Superframe only. A Component uses these fields to discover the bus configuration when joining the bus, and thereafter to verify the bus configuration on an ongoing basis. Changes shall be announced by the active Manager in the Message Channel using the process described in Section 10.3.1. A change in Framing Information without such an announcement shall indicate an error condition on the bus. See Section 10.2.4 for the proper response to such an error condition.

6.3.1.2 Framing Extension Field

The Framing Extension (FE) field signals the presence of specific Framing Information fields in the current Superframe. The active Framer shall set FE to 0 to indicate that the Subframe Mode (SM), Clock Gear (CG), Root Frequency (RF) and Reserved (R) fields are present in the current Superframe. If a Component receives a Superframe with FE=0, it shall obey all specified behaviors based on the SM, CG, and RF fields.

If a Component receives a Superframe with FE=1, it shall assume the RF, CG, SM and R fields are absent in the current Superframe. Note, the stripe S is still considered valid when FE=1.

The FE field shall have no effect on the mechanisms for acquisition and verification of the Frame synchronization. In addition, the FE field shall have no effect on the mechanism used for locking onto the Superframe Sync pattern. A Component that locks to the Superframe Sync pattern when FE=1 cannot determine the current Subframe Mode (SM) and therefore shall defer the process of acquiring Superframe synchronization and Message synchronization until the Subframe Mode has been indicated in the Framing Information.

For the purposes of verifying Superframe synchronization, Components can only check the SM, CG, and RF fields against their internal record when those fields are present, i.e. when FE=0. The FE field shall have no effect on the mechanisms for acquisition and verification of Message synchronization.

Note that the FE field differs from SM, CG, and RF fields in that it changes value without any prior announcement from the active Manager.

6.3.1.3 Superframe Sync Pattern

The Superframe Sync pattern is a punctured repeating pattern, 0X011XXX, that allows a Component to detect the beginning of a new Superframe. The active Framer shall encode the Superframe Sync pattern in stripe S of the Framing Information using the equations shown in Figure 24.

Clock Receiving Components lock on to the Superframe Sync pattern. The process of acquiring and verifying Superframe synchronization is described in Section 10.1.2.4.

6.3.1.4 Subframe Mode Field

The Subframe Mode encodes the Control Space width and Subframe length in a single field, SM[4:0], of the Framing Information.

The active Framer shall transmit the Subframe Mode using the coding shown in Table 15.

1574 **Table 15 Subframe Coding**

Coding (SM[4:0])	Control Space Width (Slots)	Subframe Length (Slots)	Comments
0b11111	24	32	
0b11110	Reserved	Reserved	Same as 0b00000
0b11101	16	32	
0b11100	16	24	
0b11011	12	32	
0b11010	12	24	
0b11001	8	32	
0b11000	8	24	
0b10111	6	32	
0b10110	6	24	
0b10101	6	8	
0b10100	Reserved	Reserved	Same as 0b00000
0b10011	4	32	
0b10010	4	24	
0b10001	4	8	
0b10000	4	6	
0b01111	3	32	
0b01110	3	24	
0b01101	3	8	
0b01100	3	6	
0b01011	2	32	
0b01010	2	24	
0b01001	2	8	
0b01000	2	6	
0b00111	1	32	
0b00110	1	24	
0b00101	1	8	
0b00100	1	6	
0b00011	Reserved	Reserved	Same as 0b00111
0b00010	Reserved	Reserved	Same as 0b00000
0b00001	Reserved	Reserved	Same as 0b00101
0b00000	8	8	100% Control Space, 0% Data Space

1575 The active Manager shall not configure the bus to use any “Reserved” Subframe Modes. A Component that
1576 observes a “Reserved” Subframe Mode coding in the Framing Information shall behave as specified in the
1577 “Comments” column in Table 15.

The active Framer shall only change the Subframe Mode at the direction of the active Manager as specified in Section 10.6.

As the Subframe Mode determines the relative allocation of the Control Space and Data Space, the concept of Subframe length has no practical effect when the Frame Structure is fully allocated to the Control Space. Even though the 100% Control Space mode (0b00000) is labeled “8/8” in Table 15, it can also be thought of as “6/6”, “24/24”, or “32/32”.

6.3.1.5 Clock Gear Field

The Clock Gear field, CG[3:0], in the Framing Information shall be encoded as shown in Table 16.

Table 16 SLIMbus Frequencies and Clock Gear Coding

Coding (CG[3:0])	Clock Gear	Minimum Frequency (MHz)	Maximum Frequency (MHz)
0b0000	Not Indicated	0	28.8
0b0001	1	0.025	0.05625
0b0010	2	0.05	0.1125
0b0011	3	0.1	0.225
0b0100	4	0.2	0.45
0b0101	5	0.4	0.9
0b0110	6	0.8	1.8
0b0111	7	1.6	3.6
0b1000	8	3.2	7.2
0b1001	9	6.4	14.4
0b1010	10	12.8	28.8
0b1011	Reserved	Reserved	Reserved
0b1100			
0b1101			
0b1110			
0b1111			

A Device that observes a “Not Indicated” or “Reserved” code in the Framing Information, and does not have independent knowledge of the CLK line frequency, should disable any functionality that depends on the CLK line being within a particular range of frequencies.

The active Manager shall not configure the bus to use any “Reserved” Clock Gear values. A Component that observes a “Reserved” Clock Gear coding in the Framing Information shall behave as if it received a “Not Indicated” Clock Gear code.

The active Framer shall only change the Clock Gear at the direction of the active Manager as specified in Section 10.5.

6.3.1.6 Root Frequency Field

Shared knowledge of the Root Frequency allows simple, low-cost programming of frequency-sensitive Components. Components can use the Root Frequency field, RF[3:0], to verify that they are at the

Cardinal, or any other preferred, frequency. The Root Frequency field also allows more complex Components to generate internal clocks without resorting to User Messages for configuring internal clock generation circuits.

The Root Frequency of SLIMbus indicates the frequency of the CLK line as if the bus were in Gear 10. The Root Frequency field, RF[3:0], in the Framing Information shall indicate the Root Frequency of the bus as coded in Table 17. Together with the Clock Gear field, the Root Frequency enables a Component to identify the frequency of the SLIMbus CLK line.

Table 17 Root Frequency

RF[3:0]	Root Frequency (MHz)	Phase Modulus (PM) (see Section 6.3.1.8)
0b1111	Reserved	Reserved
0b1110		
0b1101		
0b1100		
0b1011		
0b1010		
0b1001	27	5625
0b1000	26	8125
0b0111	25	15625
0b0110	24	625
0b0101	19.2	125
0b0100	16.8	875
0b0011	15.36	100
0b0010	22.5792	147
0b0001	24.576	160
0b0000	Not Indicated	defaults to 160

The “Not Indicated” code denotes that the operating frequency is either unknown or not listed in Table 17.

If the Clock Gear is indicated while the Root Frequency is not indicated, a Component does not know the SLIMbus CLK frequency. A Component may assume that the Cardinal Frequency in the current Clock Gear is in use. A Device that requires knowledge of the SLIMbus CLK frequency might not function properly if the frequency information is not indicated. Refer to Section 6.2.6 for more information on Cardinal Frequencies.

The active Manager shall not configure the bus to use any “Reserved” Root Frequency values. A Component that observes a reserved Root Frequency coding in the Framing Information shall behave as if it received a “Not Indicated” Root Frequency code.

The active Manager should ensure that the Root Frequency is set to “Not Indicated” (0b0000) whenever the Clock Gear is set to “Not Indicated” (0b0000).

The active Framer shall only change the Root Frequency at the direction of the active Manager as specified in Section 10.7.

6.3.1.7 Whitening Signal

The Whitening Signal is encoded in stripe S of the Framing Information as shown in Figure 24. It helps to keep the autocorrelation properties and electromagnetic spectrum of the data waveform reasonably benign at all times. This tends to ease mixed-signal Component design and system integration.

The active Framer shall write a sequence having appropriate properties to the Whitening Signal. Specifically, the absolute values of the sequence's autocorrelation coefficients shall be less than 0.1 for all lags from 1 up to at least 14. For example, any maximum length sequence of length 15 or more may be used.

A Clock Receiving Component is not required to make use of the Whitening Signal. Therefore, it may choose not to decode S6 of stripe S.

6.3.1.8 Phasing Signal

In some cases, all of the various SLIMbus-connected processes can be synchronized via the bus's Superframes or Root Superframes. For example, when the bus is operating at any of the Cardinal Frequencies its Root Superframes constitute a 4 kHz frequency-and-phase reference. In a basic telecoms system this might be the only timing reference that is required.

However, some applications need either a different timebase or a broader idea of time. To support such applications, a 25 Hz timing reference is embedded in the Framing Channel. 25 Hz has been chosen partly because it is the highest common factor of the more-common audio sample rates. Additionally, it provides a good foundation for timecode.

Using a simple per-Superframe flag to carry the timing reference would be impractical. The flag would toggle at the beat frequency of the Superframe rate and 25 Hz. When the bus clock runs at 13 MHz for example, this beat frequency is approximately 0.26 Hz. To obtain much higher update rates, the timing reference is instead communicated via numbers called Phase Snapshots. These are carried in the Phasing Signal.

The concept of Root Superframe numbering is fundamental to the Phasing Signal. The Root Superframes (Section 6.2.4) are conceptually numbered from 0 upwards, modulo a value called the Phase Modulus, PM. The numbering is such that the start of every Root Superframe 0 is also the start of a cycle of the 25 Hz timing reference. To achieve this, the Root Superframe rate divided by PM must be a unit fraction of 25, i.e. 25/N, where N is an integer. Therefore, the appropriate value for PM depends on the Root Frequency. For example, if the Root Frequency is 24 MHz, a value of 625 is appropriate. The Root Superframe rate divided by PM is then 24000000/(6144 * 625) = 6.25, which is 25/4. Table 17 specifies the PM values that shall be used at particular Root Frequencies. A Device may use a value different than the one in the table for “Not Indicated”. Mechanisms for changing the “Not Indicated” PM value are beyond the scope of this document.

The Phasing Signal consumes two Cells per Superframe. These Cells are designated P1 and P0 and are shown alongside the other Framing Information in Figure 24.

The active Framer shall generate the Phasing Signal and group the Superframes into consecutive blocks of ten. Within each Superframe block, the active Framer shall organize the Phasing Signal as shown in Figure 25.

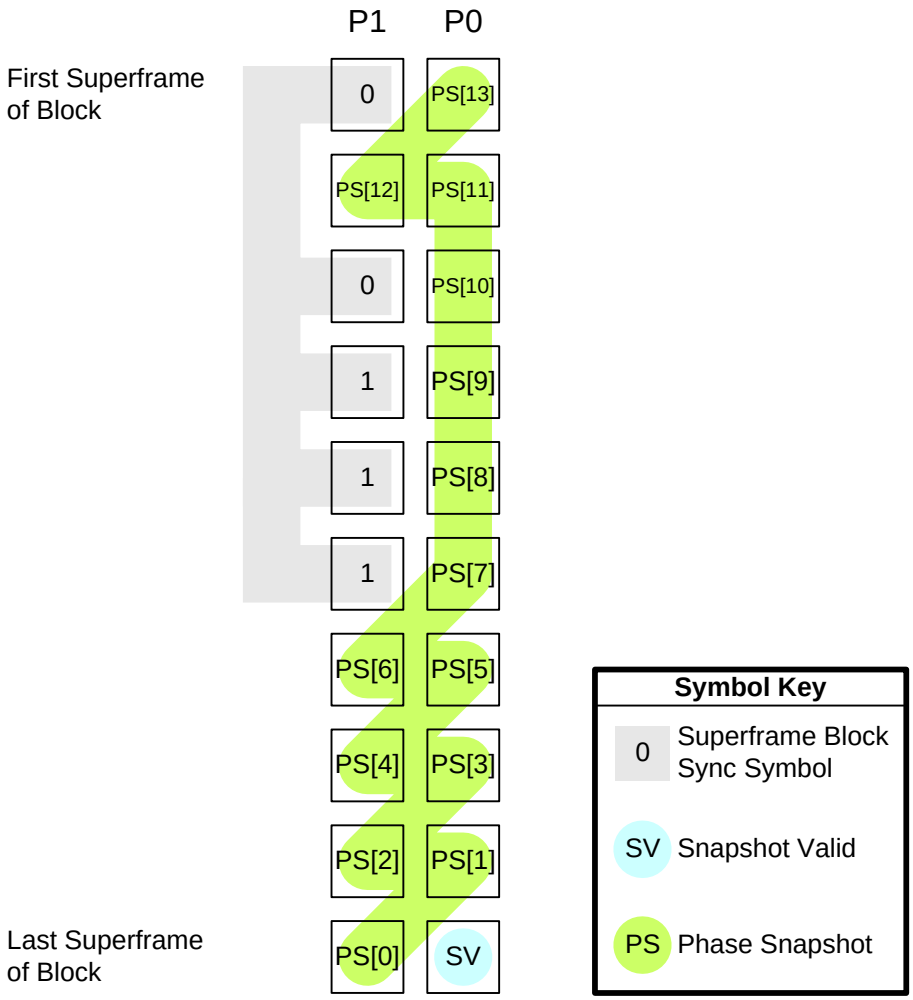


Figure 25 Phasing Signal Contents

The Superframe Block Sync symbol is a punctured repeating pattern, 0X0111XXXX, that allows a Component to determine the block boundaries. Once a Component determines the block boundaries, it can then access the Snapshot Valid, SV, and Phase Snapshot, PS[13:0], fields.

If the active Framer knows PS field contains valid information, it shall set the SV field to 1, otherwise it shall set the SV field to 0. A Component that observes SV=0 shall ignore the contents of the PS field.

A valid PS field shall contain the number of the first Root Superframe in the following block, calculated with the assumption that there will be neither a Root Frequency change nor a Clock Pause at the start of that block. Figure 26 shows an example of the Phasing Signal with valid and invalid PS fields.

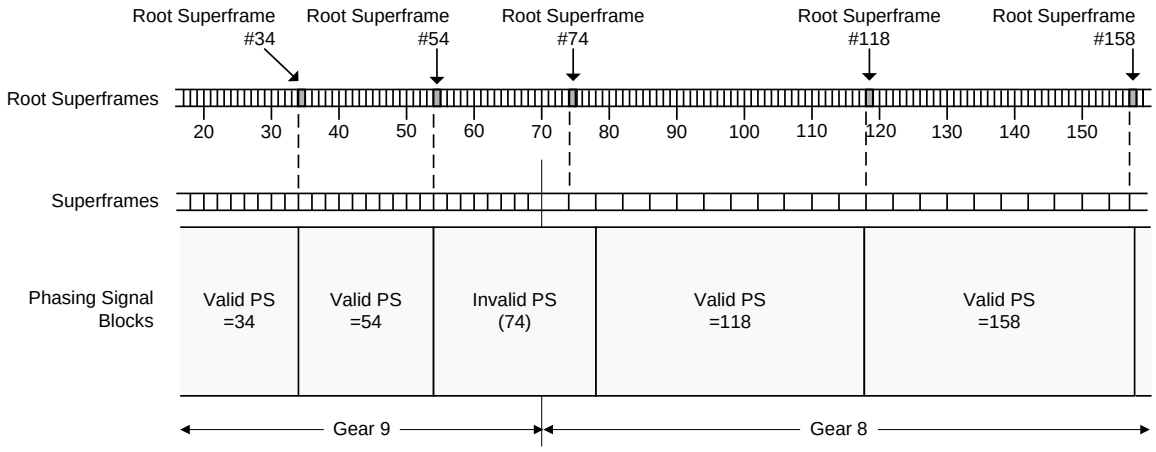


Figure 26 Phase Snapshot Example

In Gear 9, the Phase Snapshots increment by twenty because there are two Root Superframes per Superframe and ten Superframes in a Superframe block. In Gear 8 there are four Root Superframes per Superframe so the increment increases to forty, in Gear 7 the increment is eighty, and so on.

The active Framer may transmit SV=0 in blocks that span or immediately follow a Framer Handover, a Clock Gear change, a Root Frequency change or a Clock Pause. In all other blocks it shall transmit a valid Phase Snapshot with SV=1, except when the current Clock Gear is "Not Indicated" in which case it shall transmit PS[13:0]=0x0000 and SV=0. See Section 6.3.1.5.

An active Framer may adopt the following procedure for generating a legal Phasing Signal. In each Superframe block:

Transmit PS = (PHASEblock + 10 * $2^{(10-CG)}$) % PM.

where PHASEblock is the number of the first Root Superframe in the block, CG is the Clock Gear during that Root Superframe, and PM is the Phase Modulus during that Root Superframe.

If a Framer Handover, a Clock Gear change, Root Frequency change or Clock Pause takes place during the block, transmit SV=0, else transmit SV=1.

This procedure avoids the need for intelligent look-ahead. In Figure 26 for example, the value of 74 for the third Phase Snapshot would have been valid if the Clock Gear change had not taken place. Because the Clock Gear change did take place, the Phase Snapshot is invalid. The active Framer simply sends SV=0 in this Superframe block.

A Component that uses the Phasing Signal shall maintain its own internal count of Root Superframes. When such a Component acquires Superframe synchronization it shall watch for the next valid Phase Snapshot in the Phasing Signal. It shall initialize its internal Root Superframe count to that Phase Snapshot at the end of the corresponding block of Superframes. Thereafter, in normal operation the Component shall autonomously maintain its count of Root Superframes. It shall do this either by incrementing the count every $6144/2^{(10-CG)}$ Cells and wrapping it when it gets to PM, or in any way that results in the same externally observable behavior.

In normal operation the Component shall additionally check its internal count of Root Superframes against all the valid Phase Snapshots that it receives. It shall ignore isolated mismatches, but shall re-align its phase if it sees mismatches with two consecutive valid Phase Snapshots. It shall do this by initializing its internal Root Superframe count to the second of those Phase Snapshots at the end of the corresponding block of

Superframes. A Component that uses the Phasing Signal shall set its internal count of Root Superframes to zero whenever it observes a Root Frequency change, i.e. at a Reconfiguration Boundary where a NEXT_ROOT_FREQUENCY Message takes effect. The Root Superframe that immediately follows such a Reconfiguration Boundary shall be labeled number 0.

6.3.2 Guide Channel

The Guide Channel provides the necessary information for a Component to acquire and verify Message synchronization in the Message Channel. The Guide Channel occupies two Slots (one Slot-pair) per Superframe. These Slots are in Frame 0 for all bus configurations. The corresponding Slots in Frames 1 to 7 are allocated to the Message Channel.

The Guide Byte is the only information sent in the Guide Channel and shall be carried in the Superframe Slots as shown in Table 18.

Table 18 Guide Channel Slots

Control Space Width	GB[7:4]	GB[3:0]
1	Slot SL of the Superframe	Slot 2*SL of the Superframe
2	Slot 1 of the Superframe	Slot SL of the Superframe
3 or greater	Slot 1 of the Superframe	Slot 2 of the Superframe

Notes

GB is the Guide Byte

SL is the Subframe length in Slots

A Device in a Component shall not participate in the Message Channel until the Component has used the Guide Channel to achieve Message synchronization. Subsequently, the Component shall monitor the Guide Channel every Superframe to verify that it remains in, or if necessary reacquires, Message synchronization. Refer to Section 8.1 for detailed information on the construction and use of the Guide Channel.

6.3.3 Message Channel

The Message Channel consists of the Control Space Slots that are not used by the Framing Channel or the Guide Channel. Devices on the bus use the Message Channel to communicate with each other by means of the Message Protocol defined in Section 8. In order to transmit Messages, a Device uses a priority-based mechanism to gain access to the Message Channel. Refer to Section 10.11 for more information on policies for Message Channel priorities.

6.3.3.1 Message Channel Size

The size of the Message Channel varies according to the bus configuration.

The minimum number of Control Space Slots per Superframe arises in the case where the Subframe length is 32 Slots (6 Subframes/Frame) and just one Slot per Subframe is allocated to Control Space. This condition yields:

# of Slots per Superframe = 1 Slot/Subframe * 6 Subframes/Frame * 8 Frames/Superframe = 48 Slots/Superframe

For each Superframe, 16 Slots of the Control Space are used for the Framing Channel (two Slots per Frame) and two Slots are used for the Guide Channel. Therefore, the minimum number of Slots available for the Message Channel is:

# of Slots = (48 Slots - 16 Slots - 2 Slots)/Superframe = 30 Slots/Superframe

The maximum number of Control Space Slots per Superframe is given by the equation:

# of Slots = 192 Slots/Frame * 8 Frames/Superframe =1536 Slots/Superframe

Substituting 1536 Slots in the calculation for the number of Message Channel Slots yields:

# of Slots = (1536 Slots - 16 Slots - 2 Slots)/Superframe = 1518 Slots/Superframe

Thus, in a Superframe, the number of Slots available to the Message Channel ranges from 30 to 1518. A number of intermediate Message Channel sizes are available given the various Subframe Mode options shown in Table 15.

6.4 Data Space

Slots not explicitly allocated to Control Space are considered Data Space. The active Manager can organize these Slots into Data Channels to transport application-specific data streams between Devices on the bus. Data Space can contain up to 256 Data Channels.

A Data Channel is a stream of one or more contiguous Slots organized in a consistent data structure that repeats at a fixed interval. The largest possible group of contiguous Slots within a Data Channel is known as a Segment. The interval that the Segment repeats is known as the Segment Interval. The distance, in Slots, of the first Slot of the Segment from the first Slot (Slot 0) of the Superframe is known as the Segment Offset.

A Data Channel is uniquely determined by its Segment Length, Segment Interval, and Segment Offset. As such, a Data Channel can be viewed as a virtual bus with its own bandwidth guarantee and latency.

6.4.1 Segment Organization

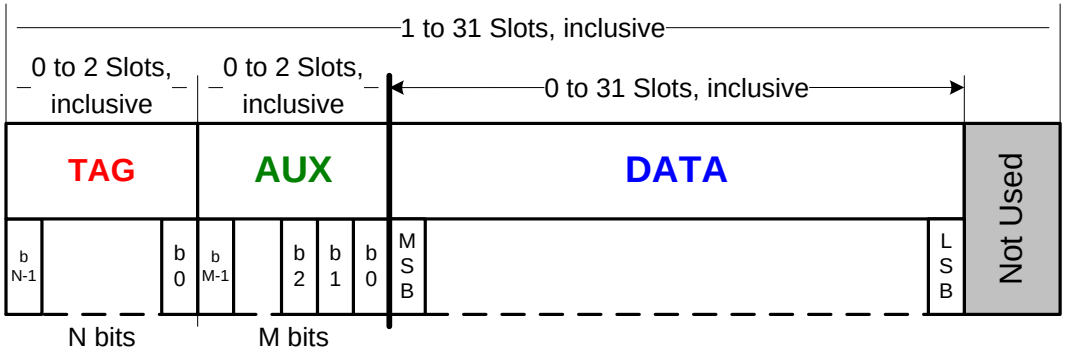


Figure 27 Segment Organization

A Segment is composed of three fields called the TAG, AUX, and DATA fields. Note the TAG and AUX fields are optional and may be omitted. These fields shall be organized within the Segment as shown in Figure 27. The exact composition of the Segment depends on the Transport Protocol. Refer to Section 9.3 for information on Transport Protocols.

The Segment structure shall be MSB aligned within the Segment. If the Segment Length is longer than the combined length of the TAG, AUX and DATA fields then the Slots after the DATA field are unused. The source Device shall not drive, and a sink Device shall ignore, the unused Slots.

6.4.2 Data Channel Properties

A Data Channel is defined by three parameters: the Segment Interval, the Segment Offset and the Segment Length.

The Segment Interval is defined as the number of Slots between the first Slot of a Segment and the first Slot of the next consecutive Segment in a Data Channel. The Segment Interval shall be constant throughout the Superframe. The Segment Interval is not inherently constrained by the Frame or Subframe length. A Data Channel may have multiple Segments in some or all of its Subframes.

The Segment Offset is defined as the number of Slots between the Superframe boundary at the start of a Superframe and the first Slot of the first Segment in a Data Channel within that Superframe. The Segment Offset shall be less than the Segment Interval. For example, a Data Channel with a Segment Offset of 200 has its first Slot per Superframe in Slot 8 of Frame 1 and has a Segment Interval greater than 200.

The Segment Length is defined as the number of Slots per Segment in a Data Channel and shall be encoded as shown in Table 19. A Device shall not send a reserved Segment Length code. A Device that receives a reserved Segment Length code shall behave as if there are no Slots allocated to that Data Channel.

Table 19 Segment Lengths

Segment Length (Slots)	Segment Length Code (SL[4:0])
1 to 31	0b00001 to 0b11111
Reserved	0b00000

A fourth parameter, the Channel Rate Multiplier is defined as the number of Segments per Superframe. Since the Superframe length is fixed (1536 Slots), the Segment Interval directly determines the Channel Rate Multiplier. See Table 20.

A Slot shall be allocated to only one Data Channel at a time. Data Channels shall not overlap.

The Segment Offset and the Segment Length are constrained by the current Subframe Mode.

A Component with an output Port shall not allow that Port to drive the DATA line during Slots allocated to Control Space, even if the Component receives a Data Channel definition that overlaps Control Space.

Together the Segment Interval and Segment Offset of a Data Channel are referred to as its Segment Distribution. The Segment Distribution shall be coded as shown in the “Segment Distribution Code” column of Table 20. In the table, 'S_N' refers to bit N of the Segment Offset value. This coding is used by the NEXT_DEFINE_CHANNEL Message (see Section 11.4.9) and relies on the fact that when the Segment Interval is a multiple of three, the two MSBs of the Segment Offset can only be 0b00, 0b01, or 0b10. This leaves 0b11 for identifying a Data Channel with a Segment Interval that is not a multiple of three. The Segment Offset value shall be greater than, or equal to, the Control Space Width defined in Table 15.

For example, the first row of the table covers the 1536 different distributions that have a Segment Interval of 1536 Slots. The eleventh row covers the 512 different distributions that have a Segment Interval of 512 Slots. Together these rows consume 2048 Segment Distribution codes, i.e. 0bXX1XXXXXXXXX.

A Device shall not send a reserved Segment Distribution code. A Device that receives a reserved Segment Distribution code shall behave as if SD=0b000000000000.

1798 **Table 20 Segment Distributions**

Segment Interval (Slots)	Channel Rate Multiplier (Segments per Superframe)	Segment Distribution Code SD[11:0]	Notes
1536	1	$S_{10}S_9$ 1 $S_8S_7S_6S_5S_4S_3S_2S_1S_0$	
768	2	S_9S_8 0 1 $S_7S_6S_5S_4S_3S_2S_1S_0$	
384	4	S_8S_7 0 0 1 $S_6S_5S_4S_3S_2S_1S_0$	
192	8	S_7S_6 0 0 0 1 $S_5S_4S_3S_2S_1S_0$	
96	16	S_6S_5 0 0 0 0 1 $S_4S_3S_2S_1S_0$	
48	32	S_5S_4 0 0 0 0 0 1 $S_3S_2S_1S_0$	
24	64	S_4S_3 0 0 0 0 0 0 1 $S_2S_1S_0$	
12	128	S_3S_2 0 0 0 0 0 0 0 1 S_1S_0	
6	256	S_2S_1 0 0 0 0 0 0 0 0 1 S_0	
3	512	S_1S_0 0 0 0 0 0 0 0 0 0 1	
512	3	1 1 1 $S_8S_7S_6S_5S_4S_3S_2S_1S_0$	
256	6	1 1 0 1 $S_7S_6S_5S_4S_3S_2S_1S_0$	
128	12	1 1 0 0 1 $S_6S_5S_4S_3S_2S_1S_0$	
64	24	1 1 0 0 0 1 $S_5S_4S_3S_2S_1S_0$	
32	48	1 1 0 0 0 0 1 $S_4S_3S_2S_1S_0$	
16	96	1 1 0 0 0 0 0 1 $S_3S_2S_1S_0$	
8	192	1 1 0 0 0 0 0 0 1 $S_2S_1S_0$	
4	384	1 1 0 0 0 0 0 0 0 1 S_1S_0	
2	768	1 1 0 0 0 0 0 0 0 0 1 S_0	S_0 cannot be zero.
Reserved	Reserved	1 1 0 0 0 0 0 0 0 0 0 1	
Reserved	Reserved	1 1 0 0 0 0 0 0 0 0 0 0	
Reserved	Reserved	1 0 0 0 0 0 0 0 0 0 0 0	
Reserved	Reserved	0 1 0 0 0 0 0 0 0 0 0 0	
∞	0	0 0 0 0 0 0 0 0 0 0 0 0	

1799 ### 6.4.3 Data Channel Definition Example (informative)

1800 Figure 28 shows an example SLIMbus bandwidth allocation for the first Frame of a Superframe. Each
1801 number represents a Slot where “0” is the first Slot of the Superframe.

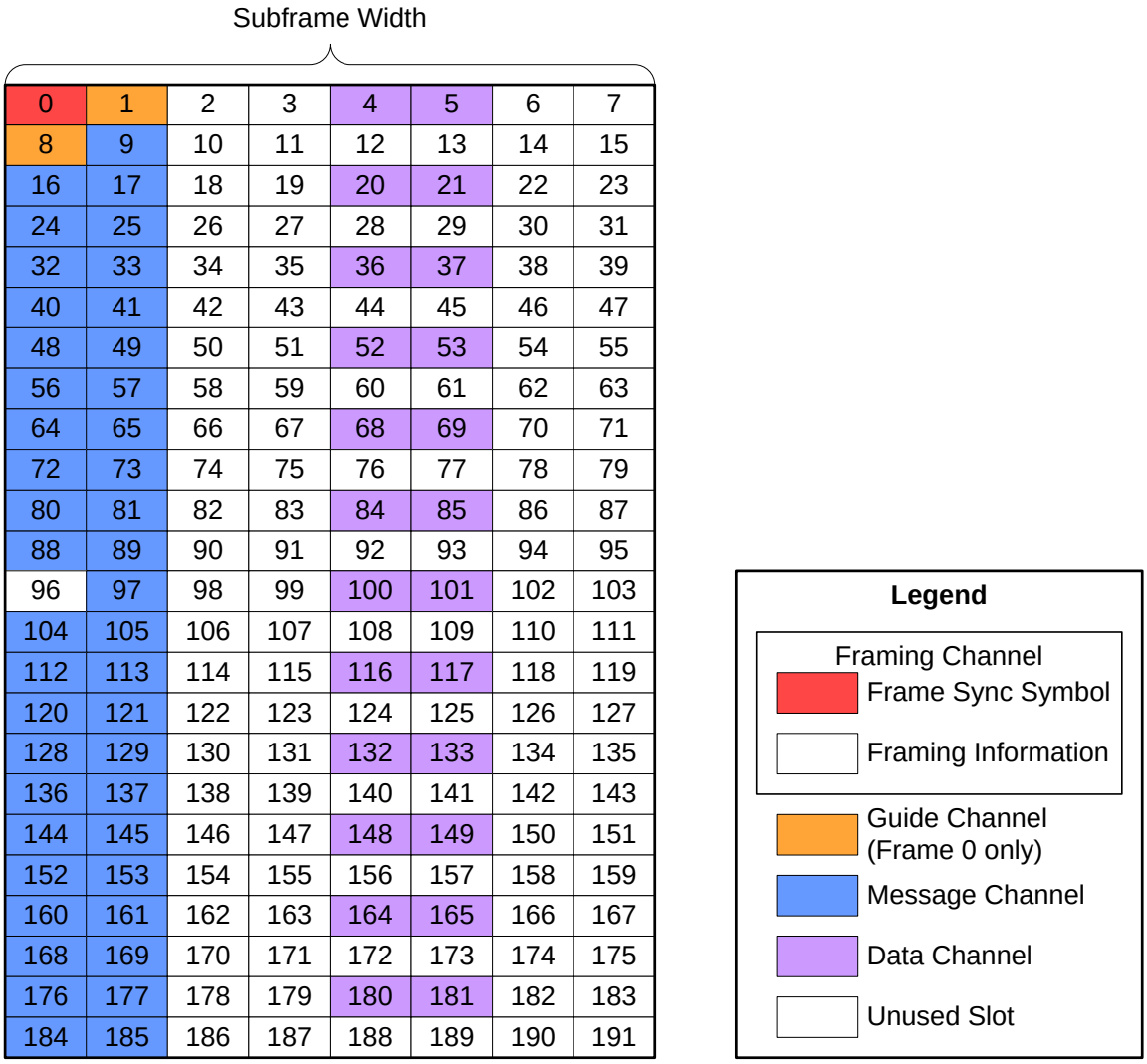


Figure 28 Data Channel Organization Example

The Control Space has a width of two Slots (8 Cells) and the Subframe length is eight Slots (32 Cells). This configuration corresponds to a Subframe Mode of 0b01001.

The Data Channel shown in the figure has the following attributes:

- Segment Length: 2
- Segment Offset: 4
- Segment Interval: 16

Consequently, the Data Channel is allocated using:

- Segment Length: 0b00010
- Segment Distribution: 0b110000010100

Note that a Data Channel has equidistant spacing throughout the Frame. This typically helps minimize buffering requirements as channel latency is consistent between Segments. Also, for a given Clock Gear, a Data Channel has a constant number of Segments per Superframe.

6.4.4 Segment Timing Model

Section 6.4.1 introduces the general structure of a Segment and Section 6.4.2 describes how Data Channels are allocated in the Frame Structure. The following three subsections describe an application model for how Segments are handled before and after they are carried on SLIMbus.

Components that support Data Channels shall behave like the model.

6.4.4.1 Segment Windows and Segment Window Boundaries

A Segment Window is defined as the range of Slots that a given Data Channel's Segment could possibly be located (ignoring overlap with the Control Space Slots). A Data Channel with a Segment Interval of M has 1536/M Segment Windows per Superframe where each Segment Window has a length of M Slots. The first Segment Window in a Superframe shall start at Slot 0 of the Superframe with each successive Segment Window beginning M Slots later.

A Segment Window Boundary is defined to be the time instant that one Segment Window ends and the next Segment Window begins. For each Data Channel with a unique Segment Interval, there are a set of Segment Windows and a corresponding set of Segment Window Boundaries. Note that the beginning of a Superframe shall be a Segment Window Boundary because it denotes the start of the first Segment Window in a Superframe. Moreover, the Superframe boundary shall be a Segment Window Boundary for all Data Channels regardless of their Segment Intervals. Even though different Segment Windows may have different lengths, there is always an integer number of Segment Windows per Superframe.

If data is present, a Port shall buffer a Segment's data during its Segment Window. The Segment Data shall be released to its destination at the next Segment Window Boundary.

6.4.4.2 Segment Window Boundary Example (informative)

Figure 29 shows an example of Segment Window Boundaries and Segment Windows. For a specific bus configuration and Data Channel allocation, the effect of the Segment Windows and Segment Window Boundaries is discussed.

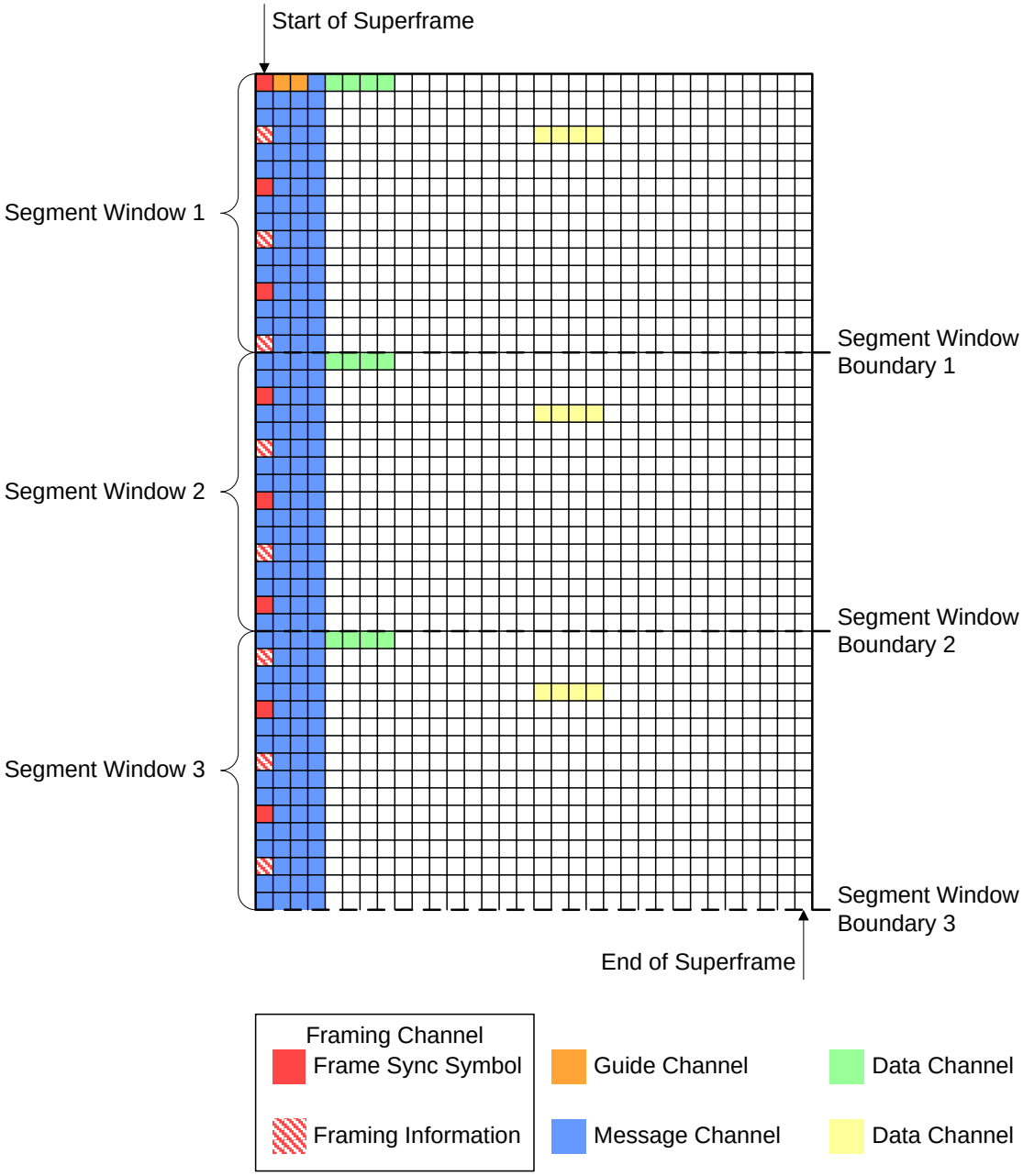


Figure 29 Segment Windows and Segment Window Boundaries Example

Figure 29 shows all 1536 Slots in a Superframe where each box represents one Slot. In this example, the Subframe length is 32 Slots, i.e. 48 Subframes per Superframe, and the Control Space width is four Slots.

Both Data Channels have a Segment Interval of 512 Slots and therefore three Segments per Superframe.

The data for the Segments that will be transmitted in Segment Window 2, must be available at Segment Window Boundary 1. That same data must be available to the sink at Segment Window Boundary 2.

As a consequence of the requirements on releasing data only on Segment Window Boundaries, Data Channels can be moved within a Segment Window without altering the latency of a single Segment. Thus, Data Channels can be moved seamlessly within the Frame Structure.

6.4.4.3 Repeatable Latency

To avoid phase issues between related Data Channels, consideration is given to the latency involved within the Device itself. That is, the latency that occurs before an output Port passes a Segment to the Frame Layer and the latency that occurs after an input Port passes up a Segment to a Device. These latencies are defined to be repeatable if the latency is consistent from occurrence to occurrence.

A Data Channel using the Isochronous Transport Protocol has repeatable latency if its source Device and sink Devices all have repeatable latency at every Segment Window Boundary.

A Data Channel using the Locked Transport Protocol has repeatable latency if its source Device and sink Devices all have repeatable latency at the start of every Root Superframe 0.

Achieving repeatable latency with the Pushed and Pulled Transport Protocols is more involved. For instance, the source Device and sink Devices must have repeatable latency at the instant that they start carrying the Data Channel.

7 Information Elements and Value Elements

Information and Value Elements are data stores used to hold status, configuration or other important information needed by a Device. An Element can hold a simple Boolean value or a more complex, multistate value.

7.1 Information Elements

Three categories of Information Element are defined: Core, Class-specific and User.

Core Information Elements contain the same type of information for all Devices of all Device Classes. Element Codes used to access the Information Elements address the same Device information regardless of Device Class.

Class-specific Information Elements contain the same type of information for all Devices of a particular Device Class, but may be different for Devices of a different Device Class. Element Codes used to access the Information Elements may address different Device information for each Device Class.

User Information Elements contain information that is specific to a specific product or product family. User Information Elements and their Element Codes are beyond the scope of this document.

The Information Elements are organized into an Information Map that has three one-kilobyte regions as shown in Figure 30. The figure also shows the associated Byte Address values. Byte Addresses in the range 0xC00 to 0xFFF are reserved. A Device shall not transmit a “Reserved” Byte Address. Section 10.12.2 mandates particular behavior for a Device that receives data requests containing Byte Addresses in the reserved range.

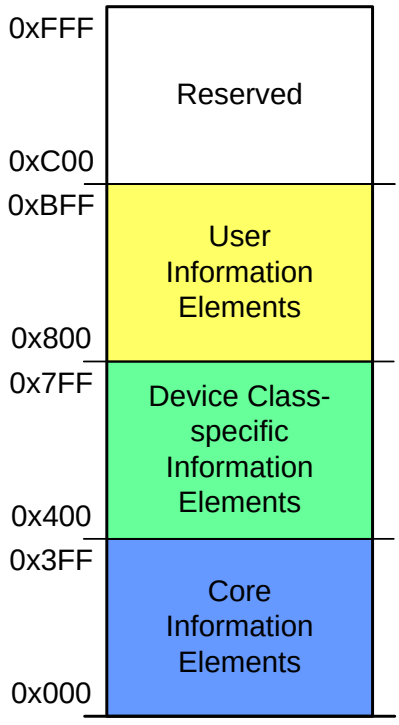


Figure 30 Information Map

Information Elements shall be located in the appropriate region of the Information Map. They shall not overlap other Information Elements in the map. Each Information Element shall occupy no more than sixteen bytes of the Information Map address space.

There are two methods of accessing Information Elements, byte-based access and elemental access. Note that a Device is typically required to support only byte-based accesses.

Byte-based accesses are analogous to generic memory reads and writes. The Element Code for such accesses includes the Byte Address of the first byte plus an indication of the number of bytes to be transferred (the Slice Size). For example, when the Byte Address is N and the Slice Size is two, the bytes at N and N+1 are transferred. When vacant parts of the Information Map are transferred using byte-based accesses, they shall appear as zeros.

Figure 31 and Table 21 show the structure and interpretation of the Element Codes that shall be used for byte-based accesses.

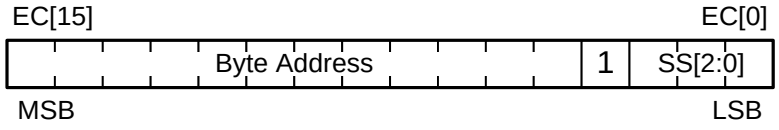


Figure 31 Element Code for Byte-based Access

Table 21 Slice Size

SS[2:0]	Slice Size (bytes)
0b000	1
0b001	2
0b010	3
0b011	4
0b100	6
0b101	8
0b110	12
0b111	16

In elemental accesses, the Element Code uniquely identifies a single Information Element.

Figure 32 shows the structure of the Element Codes that shall be used for elemental accesses. Here the Byte Address and the Bit Number together point to the Information Element's lowest bit, i.e. the one with the lowest address in the Information Map. The size of the Information Element is not expressed.

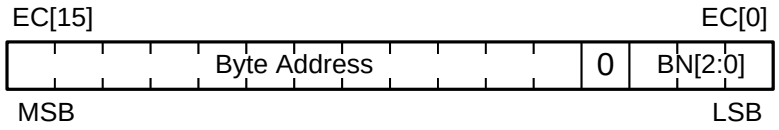


Figure 32 Element Code for Elemental Access

As indicated in Figure 31 and Figure 32, EC[3] is one for byte-based accesses and zero for elemental accesses.

Each Element Code defines a slice of the Information Map. This is referred to as an Information Slice. In a byte-based access, the Information Slice is a whole number of bytes long and can contain multiple Information Elements. In an elemental access, the Information Slice contains a single Information Element and has the same length as the Information Element as shown in Figure 33.

As specified in Section 10.12.2, Information requests may be handled with abbreviated replies in which the Message Payload contains only a Transaction ID.

A Device is not required to support both byte-based and elemental access methods except as provided in its Device Class definition. Typically, a Device supports only the byte-based access method.

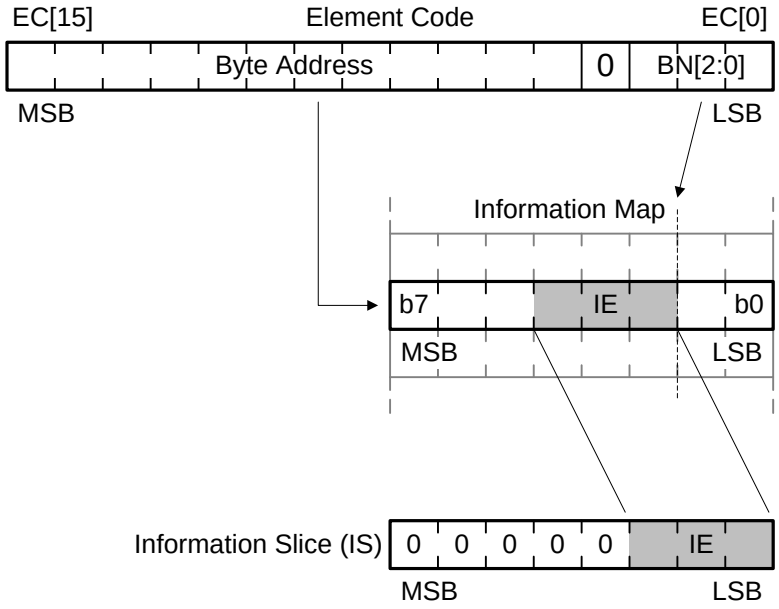


Figure 33 Example Information Slice using Elemental Access

7.1.1 Types of Information Elements

An Information Element that is only capable of being in one of two states is referred to as a Boolean Information Element. A Boolean Information Element shall obey the following rules:

- it shall be coded on one bit
- the logic value zero (0) shall indicate that it is in its FALSE condition
- the logic value one (1) shall indicate that it is in its TRUE condition
- the expression “cleared” shall mean that the value of the bit is made to the FALSE condition
- the expression “set” shall mean that the value of the bit is made to the TRUE condition
- if not implemented, it shall behave as if its value is FALSE

An Information Element that is capable of being in more than two states is referred to as Enumerated Information Element. An Enumerated Information Element shall obey the following rules:

- the expression “cleared” shall mean that the value of all bits is returned to the specified default value
- if not implemented, it shall behave as if it possesses the fixed value of all zeroes (0)

7.1.2 Core Information Elements

This section defines the Core Information Elements. It does not specify whether implementation of the Information Elements is mandatory or optional, or whether accesses of them are byte-based, elemental or both. Information Element support requirements are specified separately for each Device Class in the various Device Class definitions.

Table 22 Core Information Elements

Name	Description	Type	Size (bits)	Byte Address, Bit Number of lowest bit
DEVICE_CLASS	Indicates the Device's Device Class	read-only	8	0x009, 0
DEVICE_CLASS_VERSION	Version of the Device Class definition	read-only	8	0x008, 0
EX_ERROR	There has been a Message execution error	clearable	1	0x000, 3
RECONFIG_OBJECTION	The Device objects to the bus reconfiguration	read-only	1	0x000, 2
DATA_TX_COL	A Collision was detected in a Data Channel	clearable	1	0x000, 1
UNSPRTD_MSG	An unsupported Message has been received	clearable	1	0x000, 0

The Core Information Elements are listed in Table 22. Each of the Core Information Elements shall have the attributes identified for it in Table 22.

Each clearable Information Element in Table 22 shall be cleared by each of the following occurrences:

- the Component initially undergoing the Component Synchronization Process as described in Section 10.1.2 and Section 10.1.3.
- the Device being reset by a Device Reset, a Component Reset or a Bus Reset (Section 10.4), which changes all implemented Core Information Elements to their reset values.
- the Information Element being a target of an appropriately formulated REQUEST_CLEAR_INFORMATION or CLEAR_INFORMATION Message.

7.1.2.1 DEVICE_CLASS

The DEVICE_CLASS Information Element shall be equal to the Device Class Code of the Device.

7.1.2.2 DEVICE_CLASS_VERSION

The DEVICE_CLASS_VERSION Information Element shall be equal to the version of the Device Class implemented by the Device.

7.1.2.3 EX_ERROR

EX_ERROR is a clearable Boolean Information Element that indicates there has been an execution error related to Message sequencing or to a Message parameter value. Details can be found in Section 11.6.

The Information Element shall be set only when the Device containing the Information Element receives a Message that is supported, but that fails one or more sequence or parameter checks

7.1.2.4 RECONFIG_OBJECTION

RECONFIG_OBJECTION is a read-only Boolean Information Element that indicates when a Device objects to the announced bus reconfiguration. A Device that supports RECONFIG_OBJECTION shall set the Information Element to TRUE while it is unable to implement all of the bus reconfiguration announcements (Section 11.4) that it has received since the last BEGIN_RECONFIGURATION Message. The Device shall clear RECONFIG_OBJECTION when a BEGIN_RECONFIGURATION Message is received.

7.1.2.5 DATA_TX_COL

DATA_TX_COL is a clearable Boolean Information Element that indicates when a Collision has been detected in a Data Channel. The Device shall set DATA_TX_COL whenever the Component the Device resides within detects a DATA line Collision while one of the Device's Ports is writing to a Data Channel. See Section 10.2.2 for more information.

DATA_TX_COL shall always return FALSE if a Device does not implement any Ports.

7.1.2.6 UNSPRTD_MSG

UNSPRTD_MSG is a clearable Boolean Information Element that indicates when an unsupported Message has been received by a Device. The Device shall set UNSPRTD_MSG whenever it receives a non-broadcast Message that it does not implement as a Destination.

7.1.3 Reporting Core Information

This section provides guidelines for reporting the value of Core Information Elements to another Device. Some of the conditions listed in here should not occur during normal operation. However, during initial application development, these conditions can occur quite frequently. Reporting these conditions is intended primarily to aid debug efforts.

7.1.3.1 UNSPRTD_MSG

Reception of an unsupported Message is most likely the result of a procedural error and does not normally occur during bus operation. If this Information Element is set, a Device should send a REPORT_INFORMATION Message that includes byte 0x000 of the Information Map to the active Manager. The Device may send the same Message to the source Device that sent the unsupported Message.

7.1.3.2 EX_ERROR

Detection of an execution error is most likely the result of a procedural error and does not normally occur during bus operation. If this Information Element is set, a Device should send, at the earliest opportunity, a REPORT_INFORMATION Message that includes byte 0x000 of the Information Map to the active Manager.

7.1.3.3 DATA_TX_COL

Detection of a Collision in Data Space is most likely the result of a glitch on the DATA line, though it could also be caused by erroneously overlapping definitions of Segment Distributions. If this Information Element is set, a Device should send, at the earliest opportunity, a REPORT_INFORMATION Message that includes byte 0x000 of the Information Map to the active Manager.

7.1.3.4 RECONFIG_OBJECTION

If a Device chooses to object to an intended reconfiguration, the Device may send a REPORT_INFORMATION Message that includes byte 0x000 of the Information Map to the active Manager.

There can be deterministic timing advantages in designing a system in which the active Manager polls the RECONFIG_OBJECTION Information Elements of Devices that it knows have the capacity to analyze and object to reconfiguration.

7.1.3.5 DEVICE_CLASS

This Information Element is fixed and should not be reported by the REPORT_INFORMATION Message in normal bus operation.

7.1.3.6 DEVICE_CLASS_VERSION

This Information Element is fixed and should not be reported by the REPORT_INFORMATION Message in normal bus operation.

7.2 Value Elements

Value Elements are typically parameters that are used to configure Device behavior. The Value Elements are organized into a Value Map. Just like the Information Map (Section 7.1) the Value Map has three one-kilobyte regions as shown in Figure 34. Byte Addresses in the range 0xC00 to 0xFFF are reserved. A Device shall not transmit a “Reserved” Byte Address. Section 10.12.2 mandates particular behavior for a Device that receives data requests containing Byte Addresses in the reserved range.

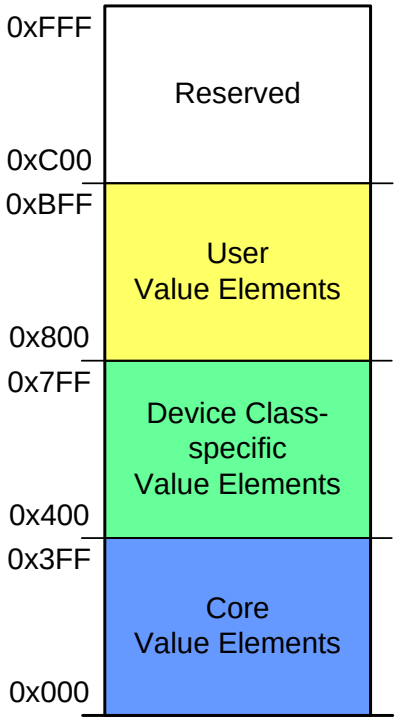


Figure 34 Value Map

Value Elements shall be located in the appropriate region of the Value Map. They shall not overlap other Value Elements in the map. Each Value Element shall occupy no more than sixteen bytes of the Value Map address space.

There is no requirement for a Device to use the entire Byte Address range. In addition, there is no required mapping between a Device's Byte Addresses and any internal memory map within a Component.

Value Messages use a 16-bit Element Code to identify a Value Slice. Byte-based and elemental accesses are possible. The structure and meaning of the Element Codes shall be the same as described in Section 7.1. When vacant parts of the Value Map are transferred using byte-based accesses, they shall appear as zeros.

Each Element Code defines a slice of the Value Map, referred to as a Value Slice. In a byte-based access, the Value Slice is a whole number of bytes long and can contain multiple Value Elements. In an elemental access, the Value Slice contains a single Value Element and has the same length as the Value Element.

As specified in Section 10.12.2, a Value REQUEST Message may be handled with an abbreviated REPLY Message in which the Message Payload contains only a Transaction ID.

8 Message Protocol and Tracking

This section defines the Message Protocol used by Devices to exchange Messages. The Message Protocol consists of the Message Channel, which carries Messages between Devices, the Guide Byte, which contains the necessary information for Components to synchronize with the Message Channel, the Guide Channel, which carries the Guide Byte, and a set of behaviors that describe the flow of Messages between Devices.

A Message is an integer number of bytes in length, ranging from a minimum of six to a maximum of thirty-nine bytes. Fields are provided for channel arbitration, source and destination addresses, Message contents (payload), Message integrity and acknowledgement. The Message syntax is described in Section 8.2.

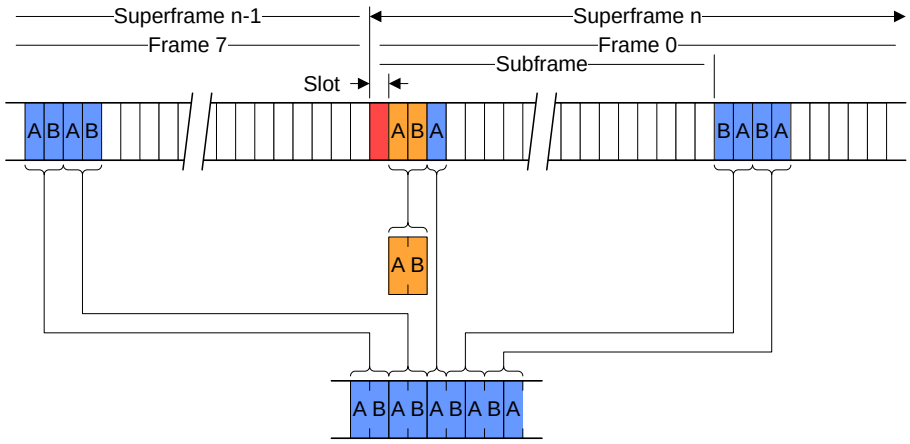
The Guide Byte contains fields so that a Device can determine the location of Messages within the Message Channel, and the integrity of the Guide Byte. The Guide Byte fields are described in Section 8.1.1 through Section 8.1.3.

Since two Slots are required to carry a byte, the Message Channel and the Guide Channel are organized as streams of Slot-pairs. These channels always occupy an even number of Slots in a Frame.

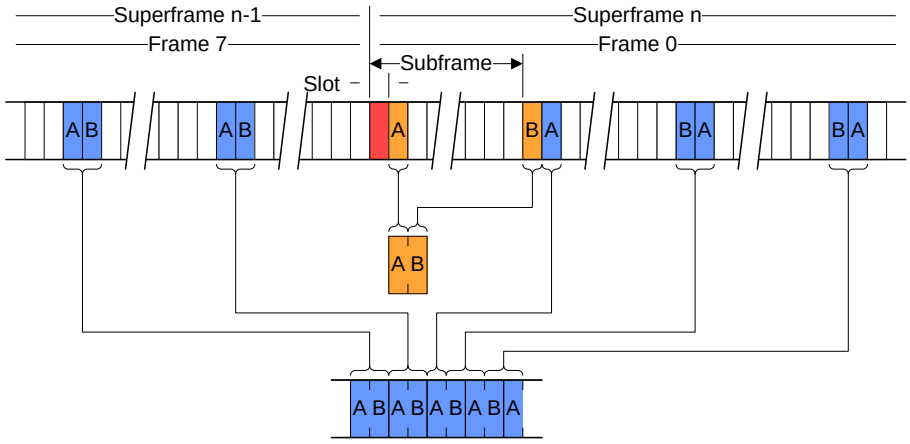
Both Slots of a Message Channel Slot-pair are always contained within a single Frame, though they can be in different Subframes. When a Message is put into the Message Channel, the Message is byte justified to a Slot-pair, i.e. the first Cell of the Slot-pair holds the first bit of the Message. The Message Channel is described in Section 6.3.3.

The Guide Channel occupies a predetermined set of Slots within a Superframe as described in Section 6.3.2. These Slots are located so that the Guide Channel Slot-pair never falls between the Slots of a Message Channel Slot-pair.

Figure 35 shows two example SLIMbus configurations using different Subframe Modes. In the first configuration, the Guide Channel Slot-pair is contiguous while in the second configuration the Guide Channel Slot-pair is split across two Subframes.



Four Control Space Slots per Subframe Example



Two Control Space Slots per Subframe Example

Figure 35 Message Channel and Guide Channel Overview

8.1 Guide Channel

The Guide Channel contains the information necessary for a Component to acquire and maintain Message synchronization.

The Guide Channel is composed of two Slots per Superframe, called the Guide Byte. The Guide Byte shall be written by the active Framer, and is illustrated in Table 23.

Table 23 Guide Byte

Byte	7	6	5	4	3	2	1	0
0	ENT	GV[4]	GV[3]	GV[2]	GV[1]	GV[0]	GI[1]	GI[0]

8.1.1 The Guide Integrity Field

The Guide Integrity field, GI[1:0], is a simple parity check of the Guide Byte contents. It provides some degree of error detection in the Guide Channel.

The active Framer shall generate the Guide Integrity field as follows:

- GI[0] = ~(GV[4] ^ GV[2] ^ GV[0])
- GI[1] = ~(ENT ^ GV[3] ^ GV[1])

A Component shall use the following formula to validate the Guide Byte:

Guide Byte Parity Check = (ENT ^ GV[3] ^ GV[1] ^ GI[1]) & (GV[4] ^ GV[2] ^ GV[0] ^ GI[0])

Note that the Guide Byte parity check detects all isolated Guide Channel errors that affect no more than two consecutive Cells. It also detects when the Guide Channel is not being driven.

8.1.2 The ENT Bit

The ENT bit indicates if the Guide Byte occurred between the Remaining Length (RL) field and the start of the next Message. If the Guide Byte occurs after the RL field, but before the start of the next Message, the ENT bit shall be set to zero. If the Guide Byte occurs elsewhere, the ENT bit shall be set to one.

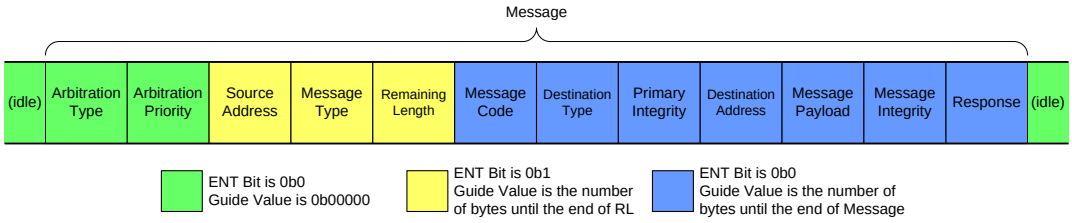


Figure 36 ENT Bit

Note that the Guide Value is 0b00000 while the Message Channel is idle, and that the Guide Byte cannot occur between the Arbitration Type and Arbitration Priority fields of a Message.

When the Guide Byte occurs between the Arbitration Priority and Source Address fields of a Message, the ENT bit is set to one. When the Guide Byte occurs between the Remaining Length and Message Code fields of a Message, the ENT bit is set to zero.

8.1.3 The Guide Value Field

The Guide Value, GV[4:0], indicates the number of bytes in the Message Channel until the end of the RL field, or until the end of the Message. Depending where the Guide Byte is inserted, the GV value has different meanings:

- When ENT=0, the Guide Value shall be the number of bytes in the Message Channel, 0 to 31, that remains after the Guide Byte until the end of the current Message.
- When ENT=1, the Guide Value shall be the number of bytes in the Message Channel, 1 to 7, that remains after the Guide Byte until the end of the RL field.

8.1.4 Guide Byte Corner Cases

If no Devices are sending a Message in the Message Channel, the ENT bit and the Guide Value shall be zero, i.e. ENT=0 and GV=0b00000.

If the Guide Byte occurs immediately after the end of a Message, the ENT bit and the Guide Value shall be zero.

8.2 Message Syntax

Devices exchange information over the Message Channel in the form of Messages. A Message consists of the four fields shown in Figure 37. The maximum Message length is thirty-nine bytes.

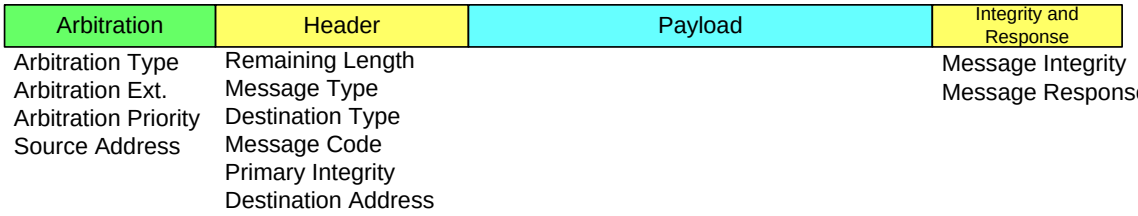


Figure 37 Message Fields

The Arbitration field is present in all Messages and contains four sub-fields: Arbitration Type, Arbitration Extension, Arbitration Priority and Source Address. See Section 8.2.1 for more information.

The Header field is present in all Messages and contains six sub-fields: Remaining Length, Message Type, Destination Type, Message Code, Primary Integrity, and Destination Address. See Section 8.2.1.4 for more information.

The Message Payload contains Message-specific parameters and data, and in some cases has zero length. See Section 8.2.3 for more information.

The Message Integrity and Response field is present in all Messages and contains two sub-fields, a 4-bit CRC and a 4-bit Message Response field. See Section 8.2.4 for more information. A Component shall send the Message Response sub-field using Logical-OR signaling as described in Section 5.1.

Table 24 describes the different fields that constitute a Message.

Table 24 Message Field Overview

Field	Bytes	Description
Arbitration	2 or 7	Message priority plus either a Logical or Enumeration Address-based arbitration from a source Device requesting access to the Message Channel.
Header	3, 4 or 9	Information sent by the source Device such as the length of the Message.
Message Payload	0 to Maximum Payload Length	Message parameters or data sent from the source Device to the destination Device.
Message Integrity and Response	1	4-bit CRC field sent by the source Device and 4-bit response field sent by the destination Devices.

A Device shall set all reserved Cells in a field to zero when sending a Message. Reserved Cells are indicated as “Rsvd” in the tables in this section.

A Device shall not send a Message with a field set to any of its reserved or illegal values.

8.2.1 Arbitration Field

The bits are allocated to the Cells within the bytes as shown in the tables in this section.

The Arbitration field contents are described in Table 25.

Table 25 Arbitration Field Overview

Field	Mnemonic	Comment
Arbitration Type	AT[3:0]	4-bit field indicating Short or Long Arbitration.
Arbitration Extension	AE	1-bit Arbitration Extension field.
Arbitration Priority	AP[2:0]	3-bit Arbitration Priority code.
Source Address	LA[7:0] or EA[47:0]	The 8-bit Logical Address or 48-bit Enumeration Address of the Device attempting to gain access to the bus.

There are two types of arbitration, called Short Arbitration and Long Arbitration. During Short Arbitration, a Device shall use its Logical Address when sending the arbitration sequence. During Long Arbitration, a Device shall use its Enumeration Address when sending the arbitration sequence.

Table 26 shows Short Arbitration used by a Device with a Logical Address (Section 8.2.5.2).

Table 26 Short Arbitration Field Composition

Byte	7	6	5	4	3	2	1	0
0	AT[3]=1	AT[2]=1	AT[1]=1	AT[0]=1	AE=0	AP[2]	AP[1]	AP[0]
1	LA[7]	LA[6]	LA[5]	LA[4]	LA[3]	LA[2]	LA[1]	LA[0]

Table 27 shows Long Arbitration used by a Device with an Enumeration Address (Section 8.2.5.1).

Table 27 Long Arbitration Field Composition

Byte	7	6	5	4	3	2	1	0
0	AT[3]=0	AT[2]=1	AT[1]=0	AT[0]=1	AE=0	AP[2]	AP[1]	AP[0]
1	EA[47]	EA[46]	EA[45]	EA[44]	EA[43]	EA[42]	EA[41]	EA[40]
2	EA[39]	EA[38]	EA[37]	EA[36]	EA[35]	EA[34]	EA[33]	EA[32]
3	EA[31]	EA[30]	EA[29]	EA[28]	EA[27]	EA[26]	EA[25]	EA[24]
4	EA[23]	EA[22]	EA[21]	EA[20]	EA[19]	EA[18]	EA[17]	EA[16]
5	EA[15]	EA[14]	EA[13]	EA[12]	EA[11]	EA[10]	EA[9]	EA[8]
6	EA[7]	EA[6]	EA[5]	EA[4]	EA[3]	EA[2]	EA[1]	EA[0]

8.2.1.1 Arbitration Type Field

The Arbitration Type field indicates whether the Arbitration field is Short (Logical Address from an enumerated Device) or Long (Enumeration Address from an unenumerated Device). In most cases, a Device uses Short arbitration for sending Messages. During Device discovery, or when a Device does not have a Logical Address, a Device uses Long arbitration.

When the Message Channel is idle (no Device is transmitting), all Cells are equal to zero.

All other values of the Arbitration Type field are illegal.

Table 28 Arbitration Types

Arbitration Type (AT[3:0])	Description
0b0000	No Arbitration
0b0001 to 0b0100	Illegal
0b0101	Long Arbitration
0b0110 to 0b1110	Illegal
0b1111	Short Arbitration

8.2.1.2 Arbitration Extension

The Arbitration Extension field, AE, is a single bit value that shall be treated by a Device as part of the normal arbitration sequence.

Table 29 Arbitration Extension

Arbitration Extension (AE)	Description
0b0	Normal
0b1	Reserved

8.2.1.3 Arbitration Priority Codes

The Arbitration Priority codes are defined in Table 30.

Table 30 Arbitration Priority Codes

Arbitration Priority (AP[2:0])	Description
0b000	Reserved
0b001	Low Priority Messages
0b010	Default Messages
0b011	High Priority Messages
0b100	Manager assigned only
0b101	Manager assigned only
0b110	Manager assigned only
0b111	Maximum Priority, for test and debug only

A Device shall use only “Low Priority”, “Default” and “High Priority” Arbitration Priority codes unless the Manager assigns one of the “Manager assigned only” codes. A Device shall not use an Arbitration Priority code with a “Reserved” value.

The Device documentation shall define the initial Arbitration Priority code for all Messages sent by a Device. The initial Arbitration Priority code for a Message should be the “Default” code, but may be different. Also, each Message may use a different code. For example, an alert Message could use the “High Priority” code while the REPORT_INFORMATION Message could use the “Default” code.

A Device shall not change the Arbitration Priority of a Message at runtime unless it receives, and supports, the CHANGE_ARBITRATION_PRIORITY Message (see Section 10.11).

When arbitrating with an Enumeration Address, a Device shall use only the Arbitration Priority code defined in the Device documentation.

For test and debug activities only, a Device may change its Arbitration Priority code to the "Maximum Priority" code.

8.2.1.4 Source Address

The Source Address is the Enumeration Address or Logical Address of the Device sending a Message. See Section 8.2.5.1 and Section 8.2.5.2 for descriptions of the Enumeration Address and Logical Address, respectively.

8.2.2 Header Field

There are three reserved Cells in the Header field. A Device shall set these Cells to zero, and the contents shall be used only for CRC generation.

An overview of the Header field can be found in Table 31.

Table 31 Header Fields

Field	Mnemonic	Description
Message Type	MT[2:0]	3-bit Message Type. Informs the destination Device how the Message Code is to be interpreted.
Remaining Length	RL[4:0]	5-bit Remaining Length. Specifies the remaining number of bytes in the Message. This is the length of the Header, Message Payload, and the Message Integrity and Response fields minus one.
Message Code	MC[6:0]	7-bit Message Code. Informs the destination Device about the nature of the Message. There are 128 Message Codes for each Message Type.
Destination Type	DT[1:0]	2-bit Destination Type. Informs the destination Device how the Destination Address is to be interpreted.
Primary Integrity	PI[3:0]	4-bit Primary Integrity. Provides error checking for the beginning of a Message, spanning the Arbitration Type to the Primary Integrity. The error checking is a 4-bit CRC with the polynomial given in Section 8.2.4.1.
Destination Address	LA[7:0] or EA[47:0] or not present	If the Destination Type is set to Logical Address, this field describes an 8-bit Logical Address of the Destination. If the Destination Type is set to Enumeration Address, this field describes a 48-bit Enumeration Address. If the Destination Type is set to Broadcast, this field is not present.

8.2.2.1 Message Type

The Message Type field informs the destination Device how it should interpret the Message Code. The defined Message Types are shown in Table 32.

Table 32 Message Types

Message Type (MT[2:0])	Description
0b000	Core Message
0b001	Destination-referred Class-specific Message
0b010	Destination-referred User Message
0b011	Illegal
0b100	Reserved
0b101	Source-referred Class-specific Message
0b110	Source-referred User Message
0b111	Illegal

8.2.2.2 Message Codes

Message Codes are specific to a Message Type. See Section 8.2.2.1 for a description of Message Types.

8.2.2.3 Destination Type

The Destination Type field informs the destination Device how the Destination Address is to be interpreted. There are three valid, and one reserved, code as shown in Table 33.

Table 33 Destination Types

Destination Type (DT[1:0])	Description
0b00	Destination is a Logical Address
0b01	Destination is an Enumeration Address
0b10	Reserved
0b11	All Devices are Destinations, no Destination Address included in Header

Depending on the Destination Type, the Header uses one of three different formats: Broadcast, Short or Long.

The Broadcast Header format is shown in Table 34. The Broadcast Header is used to send a Message to all Devices on the bus.

Table 34 Broadcast Header Format

Byte	7	6	5	4	3	2	1	0
0	MT[2]	MT[1]	MT[0]	RL[4]	RL[3]	RL[2]	RL[1]	RL[0]
1	Rsvd=0	MC[6]	MC[5]	MC[4]	MC[3]	MC[2]	MC[1]	MC[0]
2	Rsvd=0	Rsvd=0	DT[1]=1	DT[0]=1	PI[3]	PI[2]	PI[1]	PI[0]

The Short Header format is shown in Table 35. The Short Header is used to send a Message to a specific Device on the bus.

2178 **Table 35 Short Header Format with a Logical Address**

Byte	7	6	5	4	3	2	1	0
0	MT[2]	MT[1]	MT[0]	RL[4]	RL[3]	RL[2]	RL[1]	RL[0]
1	Rsvd=0	MC[6]	MC[5]	MC[4]	MC[3]	MC[2]	MC[1]	MC[0]
2	Rsvd=0	Rsvd=0	DT[1]=0	DT[0]=0	PI[3]	PI[2]	PI[1]	PI[0]
3	LA[7]	LA[6]	LA[5]	LA[4]	LA[3]	LA[2]	LA[1]	LA[0]

2179 The Long Header format is shown in Table 36. The Long Header is used to send a Message to a specific
2180 Device on the bus.

2181 **Table 36 Long Header Format with an Enumeration Address**

Byte	7	6	5	4	3	2	1	0
0	MT[2]	MT[1]	MT[0]	RL[4]	RL[3]	RL[2]	RL[1]	RL[0]
1	Rsvd=0	MC[6]	MC[5]	MC[4]	MC[3]	MC[2]	MC[1]	MC[0]
2	Rsvd=0	Rsvd=0	DT[1]=0	DT[0]=1	PI[3]	PI[2]	PI[1]	PI[0]
3	EA[47]	EA[46]	EA[45]	EA[44]	EA[43]	EA[42]	EA[41]	EA[40]
4	EA[39]	EA[38]	EA[37]	EA[36]	EA[35]	EA[34]	EA[33]	EA[32]
5	EA[31]	EA[30]	EA[29]	EA[28]	EA[27]	EA[26]	EA[25]	EA[24]
6	EA[23]	EA[22]	EA[21]	EA[20]	EA[19]	EA[18]	EA[17]	EA[16]
7	EA[15]	EA[14]	EA[13]	EA[12]	EA[11]	EA[10]	EA[9]	EA[8]
8	EA[7]	EA[6]	EA[5]	EA[4]	EA[3]	EA[2]	EA[1]	EA[0]

2182 ### 8.2.3 Message Payload Field

2183 The maximum length of the Message Payload field varies with Message Header type and can be
2184 twenty-two bytes (Long Header), twenty-seven bytes (Short Header) and twenty-eight bytes (Broadcast
2185 Header). The Message Payload field format is shown in Table 37. Note, unlike the Enumeration Address in
2186 the Arbitration and Long Header fields (see Table 27 and Table 36), the Message Payload field is organized
2187 with the Least Significant Byte first.

2188 **Table 37 Message Payload Format**

Byte	7	6	5	4	3	2	1	0
0	PD[7]	PD[6]	PD[5]	PD[4]	PD[3]	PD[2]	PD[1]	PD[0]
1	PD[15]	PD[14]	PD[13]	PD[12]	PD[11]	PD[10]	PD[9]	PD[8]
					.			
					.			
					.			
N	PD[8N+7]	PD[8N+6]	PD[8N+5]	PD[8N+4]	PD[8N+3]	PD[8N+2]	PD[8N+1]	PD[8N+0]

2189 The contents and the length of the Core Message Payload are defined in Section 11. Message Payload
2190 definitions for all other Message types are outside the scope of this specification.

2191 ### 8.2.4 Message Integrity and Response Field

2192 The Message Integrity and Response field is comprised of the sub-fields shown in Table 38.

Table 38 Message Integrity and Response Field

Field	Mnemonic	Description
Message Integrity	MI[3:0]	See Section 8.2.4.1
Message Response	MR[3:0]	All destination Devices indicate Positive or Negative Acknowledge according to Table 40

The Message Integrity and Response sub-fields are allocated to the Cells within the Integrity and Response field as shown in Table 39.

Table 39 Integrity and Response Field

	7	6	5	4	3	2	1	0
Byte 0	MI[3]	MI[2]	MI[1]	MI[0]	MR[3]	MR[2]	MR[1]	MR[0]

8.2.4.1 Integrity Fields

The two Integrity fields, Primary Integrity and Message Integrity, each use the same 4-bit Cyclic Redundancy Check (CRC) calculation to provide error detection. As shown in Figure 38, the Primary Integrity field covers the Arbitration field and the Message Type, Remaining Length, Message Code and Destination Type sub-fields of the Header field. The Message Integrity field additionally covers the Destination Address, if present, and the Message Payload, if any. Reserved Cells shall be included when performing Integrity CRC calculations.

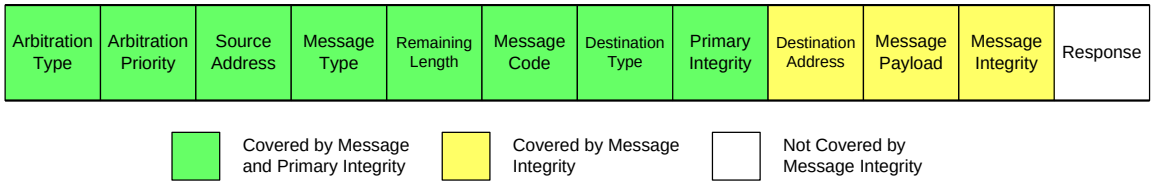


Figure 38 CRC Coverage

The CRC shall be calculated in the following manner. The CRC generator shall be initialized to 0b0000. The CRC calculation starts with the first Cell of the Message. During each following Cell the CRC shall be updated according to the formula CRC = $X^4 + X + 1$, as implemented in Figure 39 with the switch set to the XOR gate, or equivalent. Note that at the start of the relevant Integrity fields, the bit-values b_3 to b_0 relate to the CRC value as CRC[3:0] = [b_3, b_2, b_1, b_0].

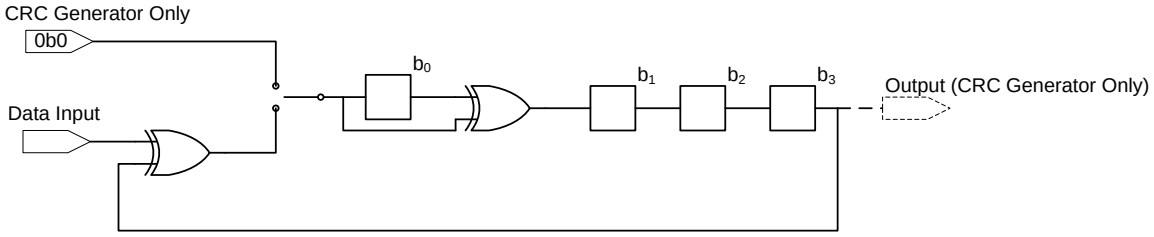


Figure 39 CRC Engine

The CRC passes if the state of the CRC engine at the end of the Integrity field is 0b0000, otherwise the CRC fails.

The Primary Integrity CRC calculation completes with the final Cell of the Destination Type field. The Message Integrity CRC calculation completes with the last Cell before the Message Integrity field.

For example, when the active Manager sends a RECONFIGURE_NOW Message with Arbitration Priority 0b110 and it receives only positive acknowledgements, the Message consists of the hexadecimal string 0xF6FF035F3B0A. The Primary Integrity CRC is 0xB and the Message Integrity CRC is 0x0, in this example.

8.2.4.2 Message Response Field

The Message Response field is encoded according to Table 40.

Table 40 Message Response Codes

Response Field MR[3:0]	Mnemonic	Description
0b1010	PACK	Positive Acknowledge
0b1111	NACK	Negative Acknowledge
0b0000	NORE	No Response
All others	UDEF	Undefined

8.2.5 Addressing

SLIMbus uses 48-bit Enumeration Addresses to uniquely identify enumerating Devices. During enumeration, a Device uses its Enumeration Address to announce its presence on the bus. The active Manager can respond to the Device using the Device's Enumeration Address. After enumeration, a Device is addressed by an 8-bit Logical Address.

8.2.5.1 Enumeration Address

An Enumeration Address is a 48-bit number comprised of four fields: Manufacturer ID, Product Code, Device Index and Instance Value, as shown in Table 41.

Table 41 Enumeration Address Fields

	7	6	5	4	3	2	1	0
EA[47:40]	MI[15]	MI[14]	MI[13]	MI[12]	MI[11]	MI[10]	MI[9]	MI[8]
EA[39:32]	MI[7]	MI[6]	MI[5]	MI[4]	MI[3]	MI[2]	MI[1]	MI[0]
EA[31:24]	PC[15]	PC[14]	PC[13]	PC[12]	PC[11]	PC[10]	PC[9]	PC[8]
EA[23:16]	PC[7]	PC[6]	PC[5]	PC[4]	PC[3]	PC[2]	PC[1]	PC[0]
EA[15:8]	DI[7]	DI[6]	DI[5]	DI[4]	DI[3]	DI[2]	DI[1]	DI[0]
EA[7:0]	IV[7]	IV[6]	IV[5]	IV[4]	IV[3]	IV[2]	IV[1]	IV[0]

The Manufacturer ID, MI[15:0], is assigned by the MIPI Alliance and uniquely identifies the company or organization responsible for the development of the Component. See [MIPI01] for details. Only Manufacturer IDs assigned by the MIPI Alliance shall be used in this field.

The Product Code field, PC[15:0], is assigned by the Component manufacturer. The Product Code should uniquely identify the Component model, e.g. family, variant, major version etc.

The Device Index field, DI[7:0], is assigned by the Component manufacturer. Each Device within a Component shall have a unique Device Index.

The Manufacturer ID, the Product Code and the set of Device Index values shall be identical for all instances of the same Component.

The Instance Value field, IV[7:0], may be configurable at system design time or during system manufacture using pin-strapping, non-volatile memory programming or other typical methods. Each instance of a Component in a system may have a unique Instance Value. Methods for determining the Instance Value are outside the scope of this document.

The Manufacturer ID, Product Code and the Instance Value shall be identical for all Devices in the same Component.

Enumeration Addresses should be unique across all Devices on the bus. Enumeration Addresses shall be unique across all Devices that can enumerate concurrently. If there are multiple instances of the same Component on the bus, and the instances contain Devices with identical Enumeration Addresses, these Devices shall enumerate in turn rather than concurrently. The definition of a mechanism for determining numeration order is outside the scope of this specification.

8.2.5.2 Logical Address

A Logical Address is used in Header and Destination Type fields to identify the source Device and destination Device of a Message. A Logical Address is an eight-bit number that uniquely identifies an enumerated Device on the bus and is assigned by the active Manager during the enumeration process. Some Logical Address values have special significance as indicated in Table 42.

Table 42 Logical Address Values

Logical Address	Description
0xFF	Active Manager. Shall not be assigned by active Manager
0xF0 to 0xFE	Reserved. Shall not be assigned by active Manager.
0x00 to 0xEF	Available for general use

8.3 Message Synchronization and Tracking

In order to transmit or receive Messages, a Component first acquires Message synchronization as described in Section 8.3.1. Once the Component achieves Message synchronization, it tracks all Messages in the Message Channel as described in Section 8.3.2. The Component also checks its internal state against every Guide Byte as described in Section 8.3.3. Behavior on the loss of Message synchronization is described in Section 8.3.4.

While a Component is not in Message synchronization, the Devices in that Component shall not transmit in the Message Channel.

8.3.1 Message Synchronization Acquisition

A Component shall achieve Frame and Superframe synchronization according to Section 10.1.2 or 10.1.3. A Component shall achieve Message synchronization in the following way:

- The Component shall read the upcoming Guide Byte, and check the parity.
  - If the parity check does not pass, a Device shall not use the Guide Byte contents and shall wait for the next Guide Byte.

- If ENT=1, the Guide Channel appeared during the arbitration sequence of a Message. In this case, the Guide Value indicates the number of bytes left until the end of the first byte of the Header, which contains the Remaining Length (RL) field. A Component shall track the Arbitration Sequence, read the RL field and track the Message until the end.
- If ENT=0, the Guide Channel did not appear during the arbitration sequence of a Message. In this case, the Guide Value indicates the number of bytes left until the end of the Message. A Component shall track the Message until the end. Note, if the Guide Value equals 0b00000, there is no Message ongoing in the Message Channel.
- In the absence of events that would cause loss of, or delay the acquisition of, synchronization, the Component shall achieve Message synchronization no later than the end of the following Guide Byte.

A state diagram showing a Component tracking the Guide Channel and Message Channel in order to achieve Message synchronization is shown in Figure 40.

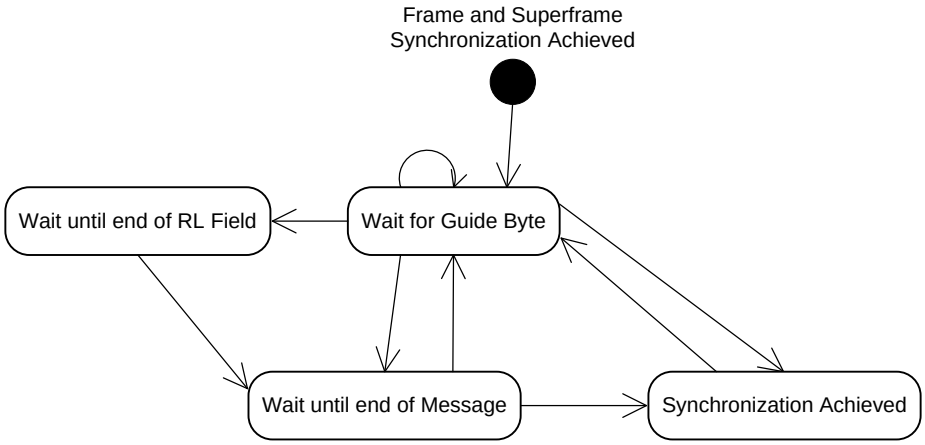


Figure 40 Message Synchronization State Diagram

As specified in Section 10.1.1.5, when the Clock Sourcing Component is not in Message synchronization, the active Framer writes the ENT bit and the Guide Value as zero.

8.3.2 Message Tracking

In order to remain synchronized to the Message Channel, a Component shall track the Messages appearing in the Message Channel. Message tracking involves reading different parts of the Message, e.g. the Remaining Length field in the Message Header, and continuously making judgments on whether to stay in Message synchronization or take other action. See Section 8.4.2.

8.3.3 Message Synchronization Maintenance

While tracking Messages, a Component shall continue monitoring every Guide Byte. If the Guide Byte Parity Check is equal to zero, the Component shall lose Message synchronization. Otherwise, the Component shall check the consistency of the Guide Byte with its internal Component state. The internal Component state is based on Message Tracking (see Section 8.3.2).

If a Component has a mismatch when comparing the Guide Byte information with its internal Message Channel tracking mechanism, the Component shall update its internal tracking mechanism with the Guide Byte information. The Component's Interface Device shall set the LOST_MS Information Element to TRUE and the current Message shall be considered aborted. A Component can be in a number of different conditions:

- A Device within the Component was transmitting a Message. The Device shall stop transmission of the current Message.
- A Device within the Component was receiving a Message. The Device shall not send a NACK or other response to the Message.
- No Device within the Component was sending or receiving a Message. No particular action is mandated for this condition.

The Component may resynchronize itself according to the Guide Byte information it just received.

8.3.4 Message Synchronization Loss

A Component that has lost track of the Message flow but that is still in synchronization with the Superframe shall wait for the next Superframe boundary and again achieve synchronization with the Guide Channel. The Component shall update its Core Information Elements and set its Interface Device's LOST_MS Information Element to TRUE.

This error shall have no effect on the way the Component is using the Data Space.

8.4 Message Protocol

The Message Protocol is a set of processes that provide a mechanism for assembling and checking parts of Messages passing between Devices and the Frame Layer.

8.4.1 Message Transmission

When preparing a Message for transmission, a Device shall obey the Message syntax rules in Section 8.2.

Because the Message Channel is a shared resource, an arbitration mechanism is defined to let a Device determine whether it is allowed to transmit a Message.

There are two different types of arbitration, called Short and Long arbitration. In almost all cases, a Device uses Short arbitration, where the arbitration sequence contains the Logical Address of the Device. During enumeration, a Device uses Long arbitration, where the arbitration sequence contains the Enumeration Address of the Device.

A Device may initiate arbitration in the first Cell of any Message Channel Slot-pair when the channel is available. A Device shall not start arbitration in any other Cell, or while its Component has determined that the channel is unavailable. There are two situations in which the Message Channel is available:

- Immediately following the end of a Message in the Message Channel, as determined by tracking Messages or from a valid Guide Value.
- When the Message Channel is unused, i.e. when the previous Message Channel Slot-pair was unused.

A Device shall use Logical-OR signaling (Section 5.1) while arbitrating. The Device shall verify the state of the bus after every Cell. If the value on the bus is different from the value transmitted by the Device, the Device loses arbitration. If a Device loses arbitration, it shall stop transmitting the Arbitration field within half a Cell, i.e. it shall not transmit another bit.

A Device obtains access to the Message Channel if it reaches the end of the Arbitration field without losing arbitration. The Device shall proceed to send the Header, Payload and Message Integrity fields of its Message. A Device that is not transmitting shall parse the Message to determine if it is a destination of the Message.

The source Device shall read the Response field and shall act as indicated in Table 43.

Table 43 Message Response Interpretation

Response	Action
PACK	The Device shall assume that all destination Devices correctly received the Message.
NACK	The Device shall assume that none of the destination Devices correctly received the Message.
NORE	The Device shall assume that none of the destination Devices received the Message.
UDEF	The Device shall assume that none of the destination Devices received the Message.

The Message Response field might not contain a PACK for a variety of reasons. Therefore, the appropriate reaction for any given response varies depending on the application. At a minimum, the source Device should retransmit the Message at least once, but should not retransmit the Message indefinitely.

Any Device may restart arbitration as soon as the Message has ended.

8.4.2 Message Reception

A Device intended to receive a Message is known as a destination Device. A destination Device shall respond to the source Device by sending an appropriate acknowledgement in the Response field. While sending its acknowledgement, a Device shall use Logical-OR signaling as described in Section 5.1. A Device that is not a destination Device shall not acknowledge the Message. The destination Device shall behave as though the Messages were executed in the order in which they were received.

A Component shall always parse the following Message fields (Section 8.2):

- The Arbitration Type, so that it knows the length of the Arbitration field
- The Message Type, some MT codes causes the Component to lose Message synchronization
- The Remaining Length, in order to maintain Message synchronization
- The Primary Integrity, for CRC calculation
- The Message Response, to see the result of the CRC calculation

Message parsing and the relation between CRC failure and Response field behavior can be visualized as in the state diagram in Figure 41.

A Device temporarily records the Logical Address of the source Device. A Device should discard its record of the source Device's Logical Address if it is not a destination of the Message. A destination Device should discard its record of the source Device's Logical Address when it has completed any necessary transactions based on the Message. For example, a destination Device of a REQUEST_INFORMATION Message needs the source Device's Logical Address until it has sent or abandoned the corresponding REPLY_INFORMATION Message (see Section 10.12).

In the case of the active Manager receiving a REPORT_PRESENT Message, if the Manager intends to assign a Logical Address to the source Device then it maintains the source Device's Enumeration Address until it has completed sending the ASSIGN_LOGICAL_ADDRESS Message.

A destination Device is not required to check that a Message Code is appropriate from a particular source Device.

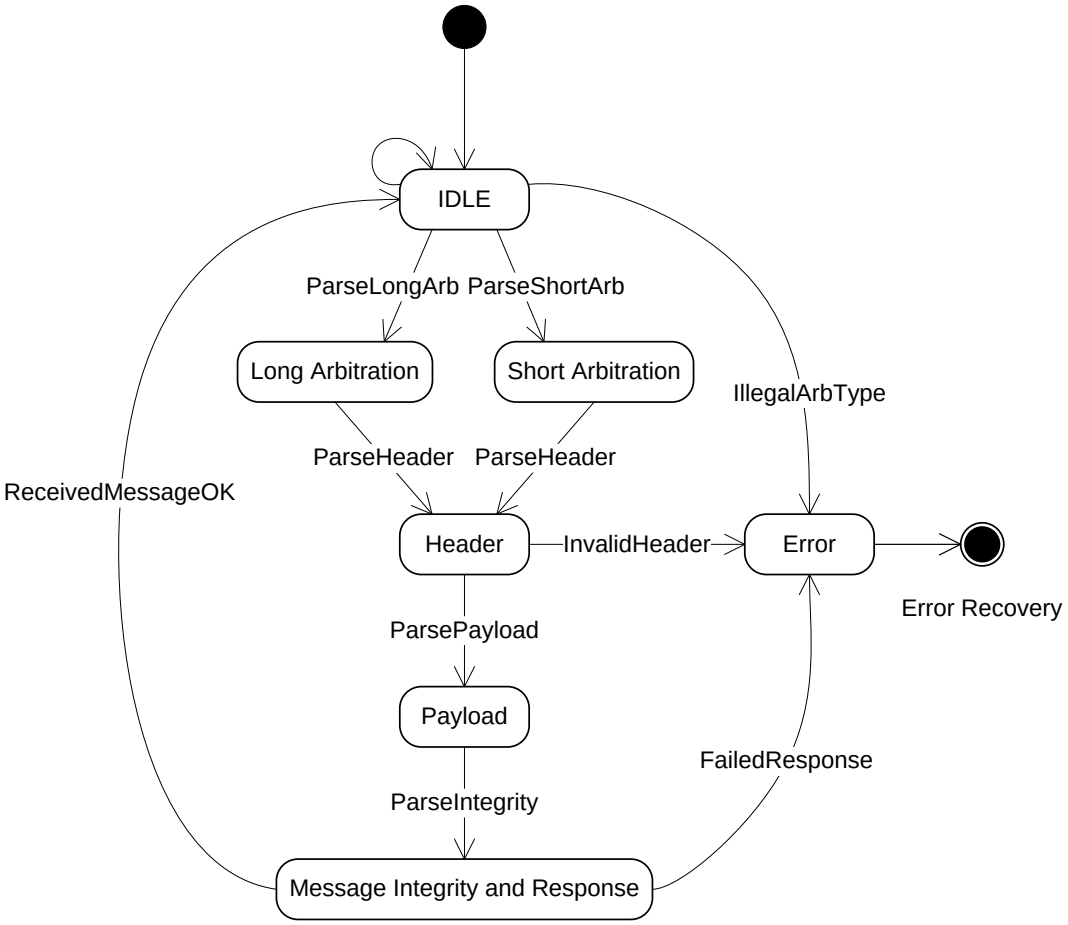


Figure 41 Basic Message Parsing Diagram

8.4.2.1 Arbitration Sequence Parsing

While in the *Idle* state, a Component shall read the first Slot of every Message Channel Slot-pair and interpret it according to Table 28. The Component shall ignore the contents of the second Slot of a Message Channel Slot-pair.

Depending upon the value of the Arbitration Type, the Component shall either stay in the *Idle* State, move to one of the Arbitration states, or when a Component observes an illegal value in the Arbitration field, the Component shall change to the *Error* state in Figure 41.

In the latter case, the Component's Interface Device shall change its LOST_MS Information Element to TRUE and lose Message synchronization as described in Section 8.3.4.

When the Arbitration Type indicates Short Arbitration, the Component shall move to the *Short Arbitration* state in Figure 41. When the Arbitration type indicates Long Arbitration, the Component shall move to the Long Arbitration state in Figure 41.

After successfully parsing one of the Arbitration sequences, the Component shall continue parsing the Header field.

8.4.2.2 Message Type Field Parsing

A Component that observes a Message with an Illegal Message Type shall discard the Message and lose Message synchronization (*InvalidHeader* transition in Figure 41).

A Component that observes a Message with Reserved Message Type shall discard the Message but stay in Message synchronization.

For all other values of the Message Type, a Component shall continue to receive the Message and parse the Message Code Field.

8.4.2.3 Message Code Field Parsing

A Core Message shall be interpreted by a Device using Table 65.

A Destination-referred Class-specific Message shall be decoded by a Device using its own Device Class Message Code table. See Section 12.

A Destination-referred User Message shall be decoded by a Device using its own User Message Code table.

A Source-referred Class-specific Message shall be decoded by a Device using the Message Code table for the Device Class of the source Device of the Message. In order to interpret the Message Code, a destination Device of a Source-referred Class-specific Message needs to know the Device Class of the Device that sent the Message. A Source-referred Class-specific Message received from a Device whose Device Class is unknown shall be treated as unsupported by the destination Device. For more information on unsupported Messages see Section 10.2.5.

A Source-referred User Message shall be decoded by a Device using the User Message Code table for the source Device of the Message. In order to interpret the Message Code, a Destination of a Source-referred User Message needs to know the User Messages of the Device that sent the Message. A Source-referred User Message received from a Device whose User Messages are unknown shall be treated as unsupported by the destination Device.

Many Message Codes do not have application to all Destination Types. A Device is not required to check that a Message Code is appropriate for a particular Destination Type.

8.4.2.4 Destination Type Field Parsing

If a Device observes a Message with a reserved value for the Destination Type, the Device shall discard the Message and shall not act on the contents.

8.4.2.5 Remaining Length Field Parsing

A Component shall use the Remaining Length field to locate the Message Integrity field. An error in the Remaining Length field generally causes the Primary or Message Integrity Field CRC check to fail and the Component to discard the Message.

When interpreting a Core Message, a Component may check whether the RL field is shorter than allowed for that Message, and discard Messages for which this is the case. However, such checking is not mandatory as such errors are generally caught by the PI and MI integrity checks.

If a Device receives a Core Message that is longer than necessary to carry the expected content, the Device shall assume that the expected content is in the first part of the Message Payload, with a layout as specified for that Message in Section 11. The Device shall use the second part of the Message Payload only for CRC calculation.

8.4.2.6 Primary Integrity Field Parsing

A Component shall read the Primary Integrity and verify the value with its internal CRC as described in Section 8.2.4.1. If the CRC fails, the Component shall change to the *Error* state and lose Message synchronization (*InvalidHeader* transition in Figure 41).

8.4.2.7 Message Response Field Parsing

A Device that is a destination of the incoming Message shall write to the Response Field as mandated in Table 44. A Device that is not a destination shall not write to the Response Field.

Table 44 Message Response Writing Rules

Response	Description
PACK	A destination Device shall issue PACK when the Message Integrity CRC passes, except when the Device requires retransmission of the Message.
NACK	A destination Device shall issue NACK if the Message Integrity CRC fails. A destination Device should issue NACK if it needs retransmission of the Message.
NORE	A destination Device shall not issue NORE.
UDEF	A destination Device shall not issue UDEF.

A Device that is a destination of the incoming Message shall read and react to the Response field as mandated in Table 45.

Table 45 Message Response Reading Rules for Destination Devices

Issued Response	Read Response	Action
PACK	PACK	The Device shall start processing the received Message.
PACK	NACK	The Device shall discard the received Message.
NACK	NACK	
NACK	PACK	The Device shall discard the received Message. It shall also cause its Component to lose Message synchronization, return to the *SeekingMessageSync* state and assert LOST_MS.
PACK, NACK	NORE, UDEF	

A Device that is not a destination of the incoming Message shall react to the Response field as mandated in Table 46.

Table 46 Message Response Reading Rules for Non-destination Devices

Issued Response	Read Response	Action
NORE	UDEF	The Device shall cause its Component to lose Message synchronization, return to the *SeekingMessageSync* state and assert LOST_MS.

9 Transport Protocols

9.1 Data Channels and Flow Control

To completely define a Data Channel, the following parameter sets must be communicated to the Devices involved: a Channel Definition, a Content Definition, and whether the Data Channel is active or inactive.

A Channel Definition indicates the location and size of the Segments in the Frame Structure. It also determines the Transport Protocol that shall be used in the Data Channel. The Transport Protocol defines the flow control or handshaking method used by the Ports to access the channel. A number of Transport Protocols are defined in this section, ranging from isochronous, where every Segment contains data, to asynchronous, where handshaking is used to emulate UART traffic. See also Section 10.13.

The Content Definition contains a number of parameters that provide additional information on the data being transported through the Data Channel. It indicates the Presence Rate (Rate of Segments that contain data), the Data Type, the DATA field length (in Slots), the Auxiliary Format (see Section 9.4) and whether the flow is Frequency Locked.

9.2 Channel Taxonomy

The Channel Definition and Content Definition shall be broadcast through the Message Channel. This section gives guidance on how to determine the Message parameters for the relevant Messages (NEXT_DEFINE_CONTENT, CHANGE_CONTENT, and NEXT_DEFINE_CHANNEL).

A first step is to determine whether flow control is needed, and if it is, what type is appropriate. This typically depends on the Devices and the type of Data involved. If needed, one of the two supported styles of flow control needs to be selected: Single-ended, or Double-ended flow control.

9.2.1 Absence of Flow Control

In cases where the frequency of the CLK line is an integer multiple of the flow rate, no flow control is needed. See for instance Section 6.2.5 on Natural Frequencies. In these cases, the Isochronous Transport Protocol (see Section 9.3.1) can be used. If the Isochronous Transport Protocol is used, the Frequency Locked field must be set to indicate that the frequency of the flow and the bus frequency are locked.

9.2.2 Single-ended Flow Control

Single-ended flows are flows where presence or absence of data is regulated by a shared algorithm (for the Locked Protocol), or by the use of a ‘Presence’ bit (for the Pulled and Pushed Protocol respectively). These protocols are designed to optimally carry constant bitrate media streams (such as LPCM audio). The exact flow control method suited for a flow depends on the Root Frequency, as well as the characteristics of the flow.

Independent of the type of data that is transported, the Frequency Locked field indicates whether the flow rate is locked to the CLK line. If the Frequency Locked field is set, the ratio between the flow rate and the CLK frequency is exactly PR / ($RF*2^{(10-CG)}$). If the Frequency Locked field is not set, this ratio might be approximate.

9.2.3 Double-ended Flow Control

With double-ended handshaking, either Device involved in the transport can pause the transfer of data. This is achieved via two or more flow control bits in every Segment’s TAG field. The four Asynchronous

Transport Protocols all use this type of flow control. These Transport Protocols have been designed to optimally support asynchronous data flow, such as data traffic to and from a Bluetooth® modem.

Independent of the type of data that is transported, the Frequency Locked field should not be set, and the Presence Rate field should be set to 'not indicated.' These parameters should be ignored by Devices when initializing an Asynchronous transport.

9.3 Transport Protocols

A Transport Protocol determines how flow control is performed for that specific channel and how the flow control information shall be mapped to the bits in the TAG field.

A Data Channel has exactly one data source at a time and may have one or more data sinks. The source produces data and a sink consumes data. The TAG bits are used to carry the flow control/information.

Each TAG bit has an associated Read/Write state for every Device that indicates if the Device shall read or write the TAG bit. The R/W default state is defined by the protocol in use. Only one Device at a time is allowed to write in a TAG bit. Multiple Devices can read the same TAG bit simultaneously. TAG bits that are identified as "Reserved" shall be written as zeros, and shall be ignored by receiving Devices.

Table 47 lists the Transport Protocols that can be used when defining a Data Channel. A Device may support all, or only a subset, of these Transport Protocols. "Reserved" values shall not be used when defining a channel. A Device shall treat "Reserved" Transport Protocol codes as unsupported Transport Protocols. See Section 11.4.9 for more information.

Table 47 Transport Protocol Overview

TP	Protocol Name	Type	# of TAG field Slots
0	Isochronous Protocol	Multicast	0
1	Pushed Protocol	Multicast	1
2	Pulled Protocol	Unicast	1
3	Locked Protocol	Multicast	0
4	Asynchronous Protocol – Simplex	Unicast	1
5	Asynchronous Protocol – Half-duplex	Unicast	1
6	Extended Asynchronous Protocol – Simplex	Unicast	2
7	Extended Asynchronous Protocol – Half-duplex	Unicast	2
8 to 13	Reserved	—	—
14	User Defined Protocol 1	—	1
15	User Defined Protocol 2	—	2

9.3.1 Isochronous Protocol

This protocol does not provide any flow information nor flow control. It should be used to carry data whose rate matches exactly the channel rate, or where flow control is embedded in the data. There are no TAG bits used, therefore the length of the TAG field shall be zero Slots.

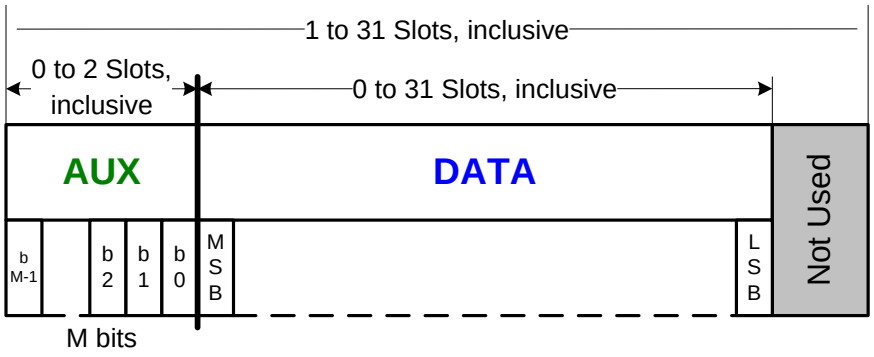


Figure 42 Isochronous Protocol Segment Organization

A typical case is the transport of a 48 kHz sample rate PCM audio (samples available 48000 times per second) using a 48 kHz channel rate (channel Segment available 48000 times per second).

As an example, when 16-bits linear PCM audio is transported over the bus, the DATA field is four Slots wide and there are no AUX bits. The Segment size is then four Slots (16-bits).

The Isochronous Protocol allows multiple sinks to be connected to a Data Channel.

9.3.2 Pushed Protocol

The Pushed protocol includes flow information. It is used to carry data whose rate is equal to, or lower than, the channel rate. The source Device drives the data flow and the TAG bits indicate availability of data in the DATA field.

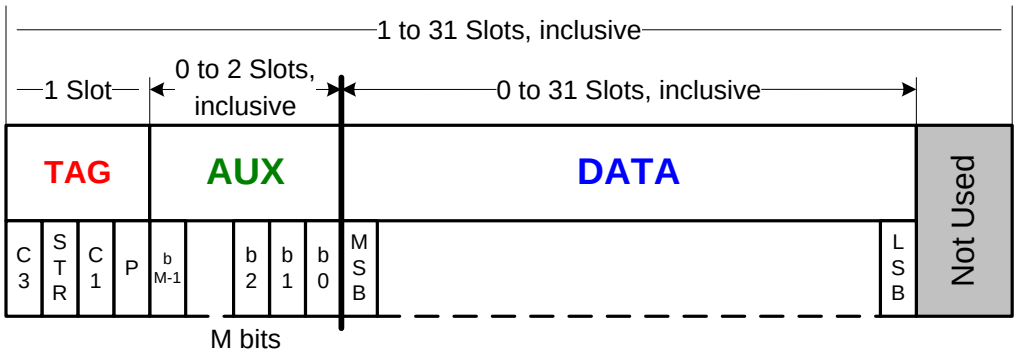


Figure 43 Pushed Protocol Segment Organization

When the source does not use the Segment, it shall reset the STROBE bit and not drive the remaining Cells of the Segment following C1:

- TAG Cell C0, PRESENCE bit
- The AUX field Slots, if any
- The DATA field Slots, if any

When the source does use the Segment, it shall set the STROBE bit and shall drive C0, the AUX Slots and the DATA Slots.

When the source decides to send data in the DATA field, it shall set the P bit. If the STROBE bit is one, but the P bit is zero, the source shall drive the AUX field Slots and the DATA field Slots with zeros.

When the STROBE bit is one, a sink shall read the P bit to detect if there is data in the DATA field. If the P bit is set, the sink shall use the contents of the DATA field. If the P bit is not set, the AUX and DATA fields may be ignored.

The STROBE bit indicates when data should be present in the Segment.

When the Frequency Locked bit is set, the STROBE bit shall provide the sink(s) with a pattern that is derived from the ratio between the data rate and the channel rate. The P bit can be used to verify if the data stream is active. For example, when the STROBE bit is one and the P bit is zero, the source has no data to transport.

When the STROBE bit is equal to zero, the PRESENCE bit shall not be driven by the source and should not be read by a sink.

Table 48 TAG Semantics for the Pushed Protocol

Cell	Name	Source	Sink(s)	Description
C3	—	Write 0	Ignore	Reserved
C2	STR	Write	Read	STROBE bit. STR=1 indicates that data is expected to be present in the Segment
C1	—	Write 0	Ignore	Reserved
C0	P	Write (conditional)	Read (conditional)	PRESENCE bit. P=1 indicates that data is present in the rest of the Segment

A use case for which the Pushed Protocol is well suited is the transport of 44.1 kHz sample rate PCM audio (samples available approximately 44100 times per second) using a 48 kHz channel rate (channel Segment available 48000 times per second). There will be about 147 samples available every 160 channel Segments. Some of the Segments will have to be ignored by the sink. In this case, the sink knows which Segments to use by reading the P bit in the TAG field Slot.

When the Pushed protocol is used to carry constant bitrate streams such as LPCM audio, the source shall constrain the pattern of occupied Segments, i.e. Segments with P=1, to avoid buffer overflow/underflow at the sink(s). In steady-state operation, the source shall constrain the peak-to-peak wideband jitter of the occupied output Segments to less than two times the reciprocal of the stream's Presence Rate (Section 9.6). An effect of this is that sinks typically do not need to buffer more than four Segments.

The Pushed Protocol allows multiple sinks to be connected to the same Data Channel (multicast), as there is no feedback from the sinks.

9.3.2.1 Example Tag Slot Values (informative)

Table 49 gives an example of a sequence of events, making more explicit the mechanisms described in Section 9.3.2.

Table 49 Example TAG Slot Values for a Pushed Channel

Sequence	C3	STR	C1	P	Comment
Segment i	0	1	0	1	The source has a sample available. It sets the STR bit and the P bit
Segment i+1	0	1	0	1	The source has a sample available. It sets the STR bit and the P bit
Segment i+2	0	0	0	—	The source does not have a sample available. It resets the STR bit and does not drive the Segment after C1

Sequence	C3	STR	C1	P	Comment
Segment i+3	0	1	0	1	The source has a sample available. It sets the STR bit and the P bit
Segment i+4	0	1	0	1	The source has a sample available. It sets the STR bit and the P bit
Segment i+5	0	0	0	—	The source does not have a sample available. It resets the STR bit and does not drive the Segment after C1

9.3.3 Pulled Protocol

The Pulled Transport Protocol provides a flow control mechanism where the sink Device requests, or pulls, data from the source Device when needed. The TAG bits indicate availability of data in the DATA field.

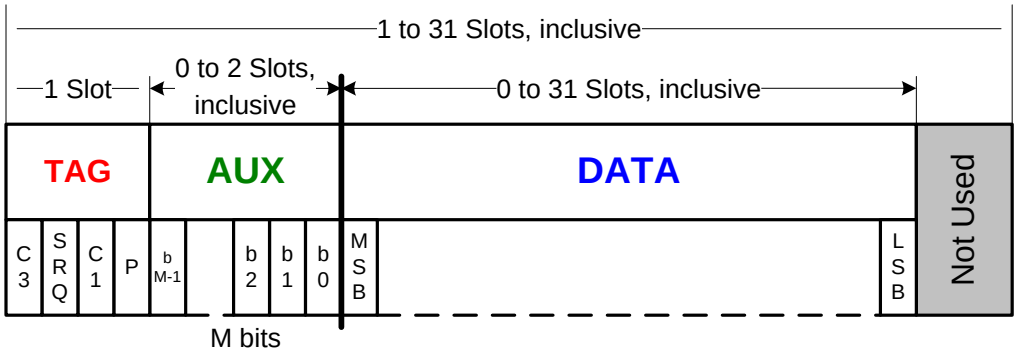


Figure 44 Pulled Protocol Segment Organization

Table 50 TAG Semantics for Pulled Protocol

Cell	Name	Source	Sink	Description
C3	—	Write 0	Ignore	Reserved
C2	SRQ	Read	Write	Sample Request. SRQ=1 requests a sample in the current Segment
C1	—	Write 0	Ignore	Reserved
C0	P	Write (conditional)	Read (conditional)	PRESENCE bit. P=1 indicates that data is present in the DATA field

The sink shall drive the SRQ (Sample Request) bit. The Source shall read the SRQ bit.

In steady-state operation, when SRQ=1 the source shall provide a valid sample in the current Segment. Whenever it provides a valid sample it shall set the P bit to indicate that the sample is present.

When SRQ=0, the source shall not drive the remaining Cells of the Segment following C1. If, for any reason, the source is not capable of servicing the sample request on time, it shall reset P to zero and write zeros to that Segment's AUX and DATA fields. The Sink and Source shall be designed such that, in steady-state operation, every Segment with a sample request is serviced within the Segment.

The use cases for the Pulled Protocol are similar to the ones of the Pushed Protocol. For example, an audio DAC can be designed to use its own clock when independence from the SLIMbus CLK line is desired. The Pulled Profile manages the frequency mismatch between the CLK line and the DAC's clock.

When the Pulled Protocol is used to carry constant bitrate streams such as LPCM audio, the sink shall constrain the pattern of Segment requests, i.e. Segments with SRQ=1, to avoid buffer overflow/underflow at the source. In steady-state operation, the sink shall constrain the peak-to-peak wideband jitter of the Segment requests to less than two times the reciprocal of the stream's Presence Rate. An effect of this is that sources typically do not need to buffer more than four Segments.

The Pulled Protocol does not allow multiple sinks to be connected to the same Data Channel. It is a point-to-point link.

9.3.3.1 Example for the Pulled Protocol (informative)

Table 51 gives an example of a sequence of events, making more explicit the mechanisms described in Section 9.3.3.

Table 51 Example TAG Sequence for the Pulled Protocol

Sequence	C3	SRQ	C1	P	Comment
Segment i	0	0	0	—	The sink does not request a sample. The source does not drive the Segment after C1.
Segment i+1	0	1	0	1	The source gets a sample request (SRQ=1) and writes data in the Segment (P=1). The sink reads the Segment data.
Segment i+2	0	1	0	1	The source gets a sample request (SRQ=1) and writes data in the Segment (P=1). The sink reads the Segment data.
Segment i+3	0	0	0	—	The sink does not request a sample. The source does not drive the Segment after C1.
Segment i+4	0	1	0	0	The sink requests a sample but the source does not have a sample available on time. It resets P and writes 0 in the remaining Cells of the Segment.

9.3.4 Locked Protocol

The Locked Transport Protocol is applicable to constant bitrate streams that are frequency locked to the CLK line. There are no TAG bits used, therefore the length of the TAG field shall be zero Slots.

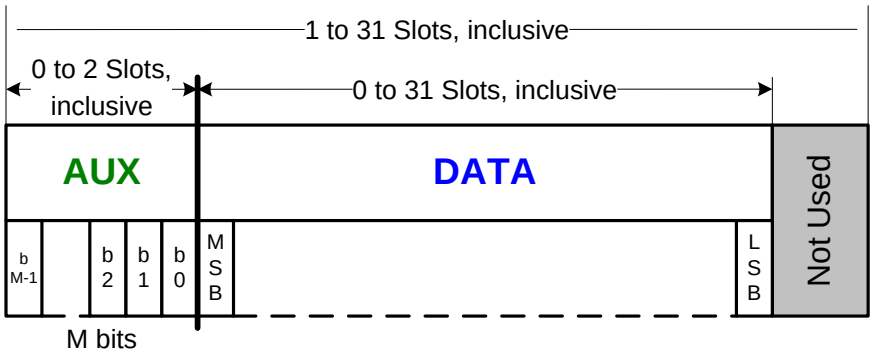


Figure 45 Locked Protocol Segment Organization

In place of a PRESENCE bit, P, that requires a TAG field Slot, the Locked Protocol uses virtual PRESENCE bits, VP. Copies of the VP bits shall be separately synthesized at each of the source and sink Devices. The mechanisms for this are synchronized by the Phasing Signal.

The source Device shall write data to those of its Segments whose VP bit evaluates to 1, and the sink or sinks shall read that data. When the VP bit evaluates to 0 the source Device shall not drive the Segment, and the sink or sinks shall ignore the contents of the Segment.

Section 6.2.4 introduces Root Superframes and Section 6.3.1.8 specifies Root Superframe numbering. For any point in the SLIMbus waveform, the distance of that point from the start of Root Superframe 0 can be expressed in Root Superframes modulo PM. Figure 46 illustrates this concept for operation at a Cardinal Frequency with Data Channels having 48,000 Segments per second. The figure shows Segment Windows (Section 6.4.4.1) rather than individual Segments.

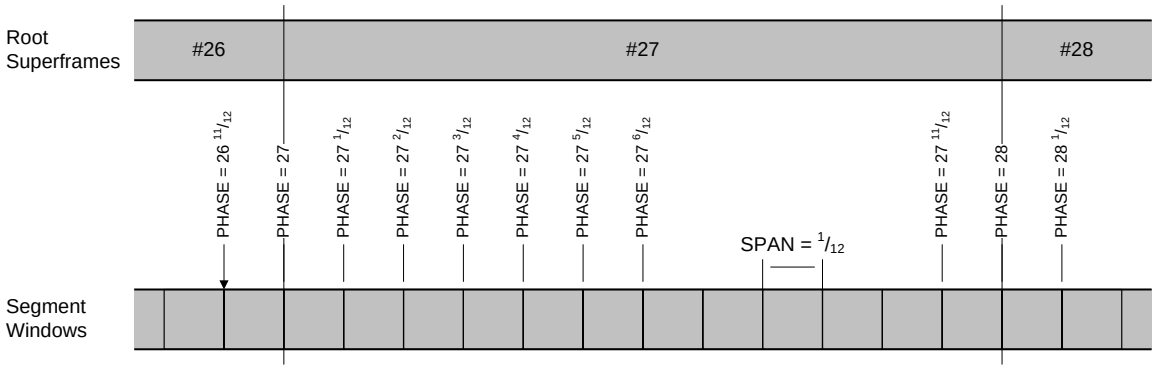


Figure 46 Segment Window PHASES and SPAN

In the Locked Protocol, the value of each Segment's VP bit shall be as follows:

If **INT**($PHASE_{SW}$ * RATIO) = **INT**(($PHASE_{SW}$ + SPAN) * RATIO) then VP=0, else VP=1.

where

INT() indicates the integer operator.

$PHASE_{SW}$ is the distance of the start of the associated Segment Window from the start of Root Superframe 0, expressed in Root Superframes modulo PM.

SPAN is the extent of the Segment Windows, expressed in Root Superframes, i.e. SPAN = $2^{(10 - CG)}$ * Segment Interval / 1536.

RATIO is the ratio of the stream's Presence Rate to the bus's Root Superframe rate, i.e. RATIO = 6144 * Presence Rate / Root Frequency.

Figure 47 illustrates VP evaluation for 44.1k flows in 48k channels with a 12.288 MHz clock.

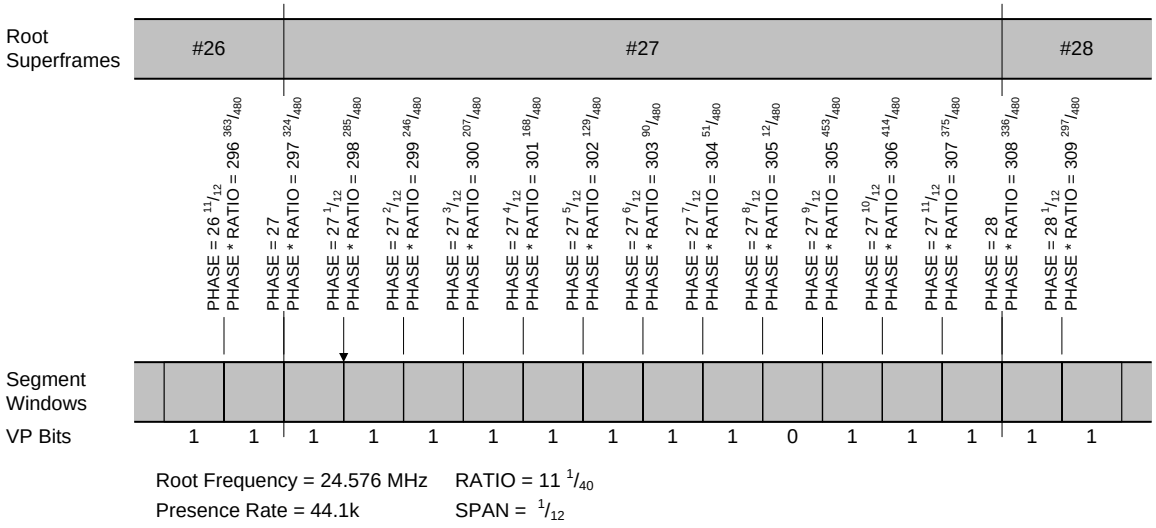


Figure 47 VP Evaluation Example, 44.1k Flows

Figure 48 provides a further illustration covering 16k flows with a 13 MHz or 26 MHz clock.

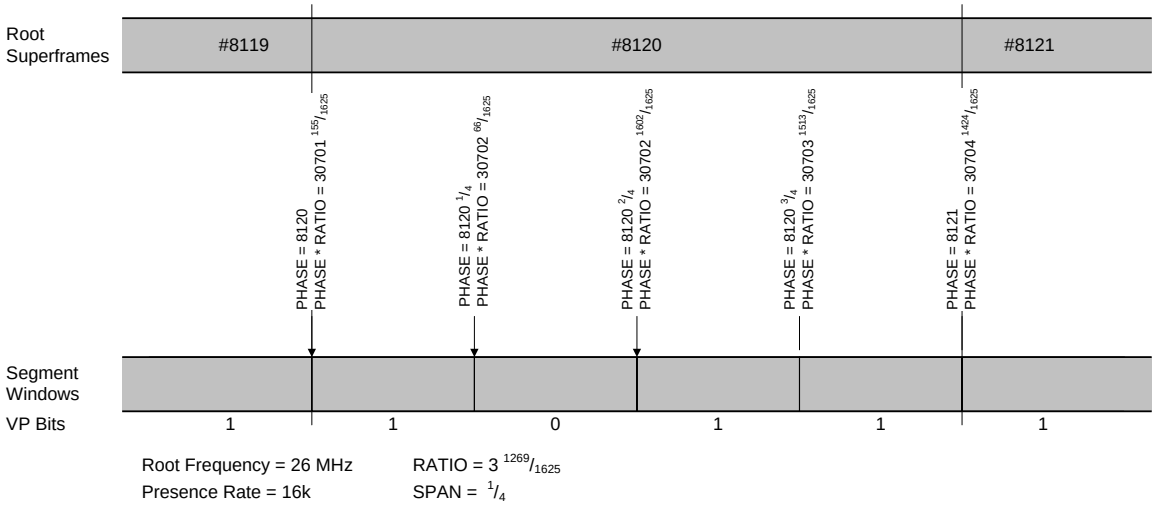


Figure 48 VP Evaluation Example, 16k Flows

Components that use the Locked Protocol shall maintain an internal count of Root Superframes modulo PM. This shall be synchronized to the Phasing Signal and to Root Frequency changes as detailed in Section 6.3.1.8.

The Locked Protocol shall not be used by Components that do not know the bus's Clock Gear or the stream's Presence Rate. This situation typically arises when the Clock Gear or the Presence Rate is 'Not Indicated', i.e. CG=0b0000 or PR=0b0000000.

9.3.5 Asynchronous Protocol

The Asynchronous protocol provides the necessary means to achieve an asynchronous data transmission, including flow control signals, over a serial synchronous link. The Asynchronous Protocol shall be used only in a point-to-point configuration between a single source Device and a single sink Device.

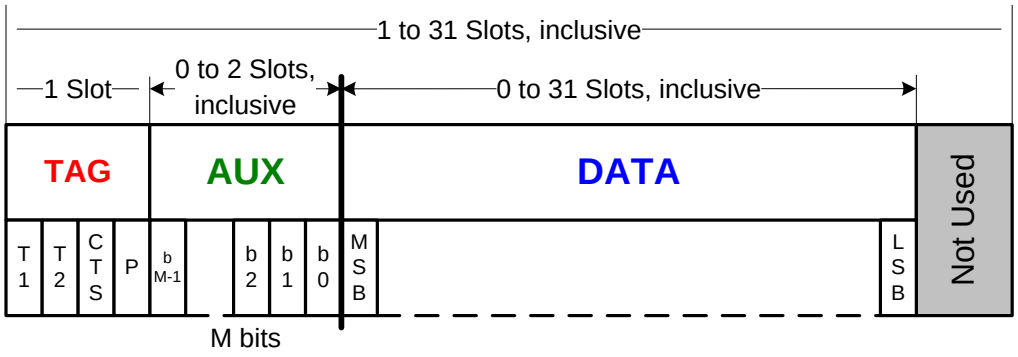


Figure 49 Asynchronous Protocol Segment Organization

The Asynchronous Protocol TAG bit definition is given in Table 52.

Table 52 TAG Semantics for the Asynchronous Protocol

Cell	Name	Primary Owner	Secondary Owner	Description
C3	T1	Write	Read	Primary Owner Token. Used by the primary owner to indicate that it owns the AUX+DATA fields
C2	T2	Read	Write	Secondary Owner Token. Used by the secondary owner to indicate that it owns the AUX+DATA fields
C1	CTS	R/W	W/R	Clear To Send. CTS=0 indicates that the source Device shall stop sending data
C0	P	W/R	R/W	PRESENCE bit. P=1 indicates that data is present in the current Segment

The TAG bits are used to indicate the following aspects of the channel:

- Ownership of the AUX and DATA field Slots
- Ability of the sink Device to receive data
- Availability of data in the Segment

9.3.5.1 Channel Ownership

When a channel with the Asynchronous Half-duplex protocol is activated, it shall be idle and neither end of the link has ownership. When a Device needs to gain ownership of the channel, it shall use its associated token bit. The primary owner shall use the T1 bit, and the secondary owner shall use the T2 bit. The primary owner has a higher priority than the secondary owner. In order to protect against single bit errors, token bits shall be repeated in two successive Segments to change ownership state. The state diagram in Figure 50 defines how ownership of the channel is established, maintained, and relinquished in the Half-duplex protocol.

The state diagram in Figure 50 is interoperable with, but not functionally equivalent to, the Asynchronous Protocol State Diagram defined in SLIMbus specifications prior to version 1.01.01.

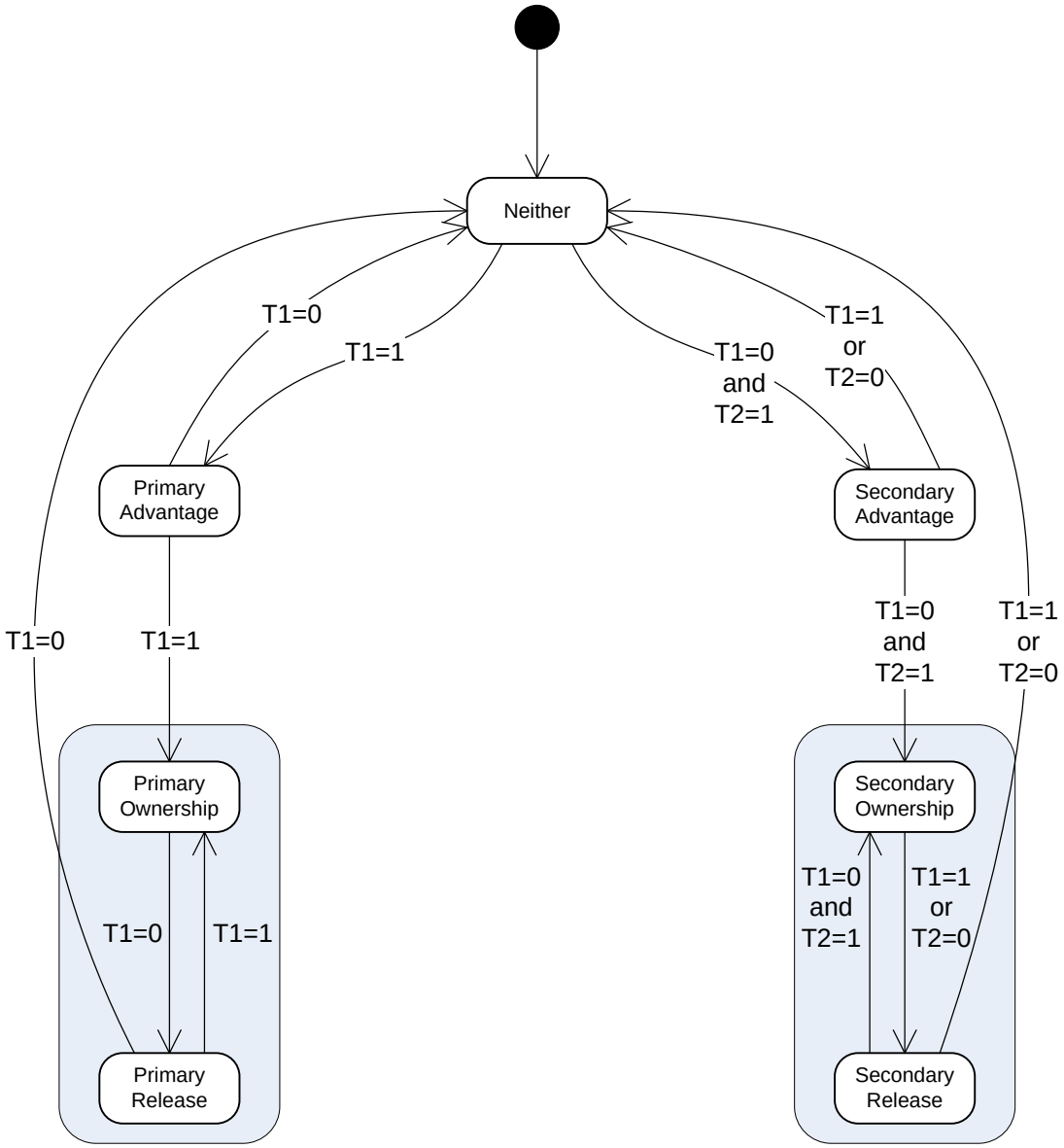


Figure 50 Asynchronous Half-duplex Protocol State Diagram

In the *Idle* state, a change in T1 or T2 in a Segment causes a change in state. If only T2=1, the secondary owner has the advantage. If T1=1, the primary owner has the advantage. A second Segment in which the same token is one gives ownership to that end of the link. In the *Primary Ownership* state, a change to T1=0 causes a transition to the *Primary Release* state. In this case, a second change to T1=0 causes a return to the *Idle* state, but a change to T1=1 causes a return to the *Primary Ownership* state. In the *Secondary Ownership* state, changes in the T2 bit mirror those for T1 in the *Primary Ownership* state. However, it also must react to T1=1 which indicates that the primary end of the link wants to take ownership, and causes a change to the *Secondary Release* state. In this case, a second Segment with T1=1 causes a change to the *Neither* state, but a change to T1=0 causes a return to the *Secondary Ownership* state. States inside the shaded boxes in Figure 50 are those in which ownership is established. A change in state observed in the current Segment shall take effect in the next Segment.

The Asynchronous Protocol can provide either a simplex or half-duplex transport within a Data Channel. Simplex versus half-duplex operation is noted in Table 47.

When a Data Channel is activated using the Asynchronous Simplex protocol, the Secondary Owner shall not attempt to take ownership of the channel, and the primary owner shall disregard the T2 bit. When full duplex communication is required, two Data Channels may be opened using the Asynchronous Simplex protocol. In this case, each Device would be the Primary Owner of one simplex Asynchronous Data Channel.

At all times when the channel does not have an owner, the remaining bits in the tag shall not be driven by either end of the link, and the AUX and DATA field Slots shall not be driven by either end of the link. When the channel does have an owner, the CTS bit shall not be driven by the owner, the P bit shall not be driven by the non-owner, and the AUX and DATA field Slots shall not be driven by the non-owner.

Once channel ownership is established, the non-owner shall begin driving the CTS bit, the owner shall begin driving the P bit, and the owner shall begin driving the AUX and DATA field Slots.

9.3.5.2 Flow Control

In order to protect against single bit errors, flow control shall be determined from the value of the CTS bit in two successive Segments. If the value in both the previous Segment and the current Segment are zero, flow is off. If the value in both the previous Segment and the current Segment are one, flow is on. If the value in the previous Segment differs from the value in the current Segment, CTS is invalid, and flow shall be off.

If the channel owner sees a change in flow from off to on, it shall not place AUX or DATA in the channel until the next Segment. If the channel owner sees a change in flow from on to off, it shall not place AUX or DATA in the Channel after the current Segment.

If the channel does not have ownership established, flow control shall be off regardless of the values seen on CTS.

Table 53 Flow Control Conditions

State	CTS	Flow
Primary Ownership	Valid 1	ON
Secondary Ownership	Valid 1	ON
Primary Release	Valid 1	ON
Secondary Release	Valid 1	ON
Primary Advantage	—	OFF
Secondary Advantage	—	OFF
Neither	—	OFF
—	Valid 0	OFF
—	Invalid	OFF

9.3.5.3 Data Transmission

If flow is on, the channel owner shall indicate whether data is present in the AUX and DATA field Slots by setting the P bit in the TAG field Slot of the associated Segment. The P bit is read-only from the existing Segment.

9.3.5.4 Behavior on Bit Errors

The following sections describe the impact of bit errors on the TAG field Slots and the potential impact on the behavior of the Asynchronous Transport Protocol.

9.3.5.4.1 Bit Errors in T1 or T2

The state machine for channel ownership has been designed such that single bit errors that affect both the primary and the secondary owner will not affect ownership of the channel. A transition will occur to an intermediate state. In this state, an error-free token bit in the following Segment will return ownership to the correct state.

A second, successive bit error could cause an undesired change in the channel ownership, but no single bit error will have any such effect.

9.3.5.4.2 Bit Errors in CTS

Any change in the value of CTS causes the flow control state to become invalid. If the change in value is caused by a bit error in a single Segment, the flow control will be invalid for two consecutive Segments. This means that the channel owner will be flowed off for a minimum of one and a maximum of two Segments.

Bit errors in CTS have no effect if the channel ownership is not established.

9.3.5.4.3 Bit Errors in P

A spurious bit error which changes the value of P during a transmission can be detected only at higher layers.

If P is supposed to be 1 but is read as a 0, the data conveyed in the Segment will be ignored, resulting in data loss.

If P is supposed to be 0 but is read as a 1, the receiver will treat the Segment content as valid, resulting in spurious data in the transmission.

Bit errors in P have no effect if the channel ownership is not established or if flow is off.

9.3.5.5 Examples for the Asynchronous Protocol (informative)

Table 54 gives an example of a sequence of Segments illustrating the processes involved in channel ownership described in Section 9.3.5.1.

Table 54 Example TAG Slot Sequence for Channel Ownership

Segment	T1	T2	CTS	P	Comment
Segment i	0	0	Z	Z	The channel is idle. Neither the primary nor the secondary owner is requesting ownership of the channel, and the ownership state is *Neither*.
Segment i+1	0	1	Z	Z	The secondary owner requests channel ownership (T2=1), and the ownership state changes to *Secondary Advantage*".
Segment i+2	0	1	Z	Z	The secondary owner requests channel ownership (T2=1), and the ownership state changes to *Secondary Ownership*".

Segment	T1	T2	CTS	P	Comment
Segment i+3	1	1	Z	Z	The primary owner requests channel ownership (T1=1), overriding the T2 bit. The ownership state changes to *Secondary Release*.
Segment i+4	1	0	Z	Z	The primary owner continues to request channel ownership (T1=1), and the ownership state changes to *Neither*.
Segment i+5	1	0	Z	Z	The primary owner continues to request channel ownership (T1=1), and the ownership state changes to *Primary Advantage*.
Segment i+6	1	0	Z	Z	The primary owner continues to request channel ownership (T1=1), and the ownership state changes to *Primary Ownership*.
Segment i+7	1	0	0	0	The secondary owner begins driving the CTS bit, and the primary owner begins driving the P bit, as well as the AUX and DATA field Slots.
Segment i+8	0	0	0	0	An error on the T1 bit causes it to be read as zero. The ownership state changes to *Primary Release* but the primary owner continues to own the channel.
Segment i+9	1	0	0	0	The T1 bit is read correctly again as a 1, and the ownership state returns to *Primary Ownership*.
Segment i+10	0	0	0	0	The primary owner ceases to request channel ownership (T1=0) and the ownership state changes to *Primary Release*
Segment i+11	0	0	0	0	The primary owner continues with T1=0, and the ownership state returns to *Neither*.
Segment i+12	0	0	Z	Z	The channel has no owner.

Notes

"Z" represents a not driven condition.

Table 55 gives an example of a sequence of Segments, illustrating the processes involved in flow control and data transmission described in Section 9.3.4. Note that in the following example, the primary owner gains channel ownership and sends data, however a similar sequence applies should the secondary owner gain channel ownership.

Table 55 Example TAG Slot Sequence for Flow Control and Data Transfer

Segment	T1	T2	CTS	P	Comment
Segment i	0	0	Z	Z	The channel is idle. Neither the primary nor the secondary owner is requesting ownership of the channel, and the ownership state is *Neither*.
Segment i+1	1	0	Z	Z	The primary owner requests channel ownership (T1=1), and the ownership state changes to *Primary Advantage*.
Segment i+2	1	0	Z	Z	The primary owner continues to request channel ownership (T1=1), and the ownership state changes to *Primary Ownership*. The primary owner becomes the DATA source and the secondary owner becomes the DATA sink.

Segment	T1	T2	CTS	P	Comment
Segment i+3	1	0	1	0	The channel ownership is now established and the sink may set the CTS bit to indicate to the source that it is ready to accept data. Note that there is no requirement for CTS to change immediately after channel ownership is established, but in this example it does.
Segment i+4	1	0	1	0	With the second occurrence of CTS=1, flow control is on. The DATA source has not set P in this Segment.
Segment i+5	1	0	1	1	One Segment after flow on, the source may start writing data in the current Segment, and in this example does so by setting P=1.
Segment i+6	1	0	1	1	A second Segment in which the DATA source sends data to the DATA sink.
Segment i+7	1	0	1	0	The DATA sink allows data to be present in the current Segment (CTS=1) but the source does not have any DATA to write in the Segment (P=0).
Segment i+8	1	0	1	1	Another Segment in which the DATA source sends data to the DATA sink.
Segment i+9	1	0	0	1	The DATA sink stops allowing data to be sent (CTS=0). Flow control is marked invalid because the last 2 CTS values are not equal. The DATA source may send data in the current Segment, and in this example does so.
Segment i+10	1	0	0	0	The DATA sink continues to prevent data from being sent (CTS=0). Flow control is off. The DATA source does not send data in the current Segment.
Segment i+11	0	0	0	0	The source has finished its transmission and releases the token T1=0. CTS and P are reset to 0 by their respective owner.
Segment i+12	0	0	0	0	The primary owner continues with T1=0, and the ownership state returns to *Neither*.
Segment i+13	0	0	Z	Z	The channel has no owner.

Notes

"Z" represents a not driven condition.

9.3.6 Extended Asynchronous Protocol

When the channel rate is low and the required data rate is high, it is possible to send multiple bytes in the Segment DATA field. The Extended Asynchronous Protocol has a DATA field size that can range from 2 to 28 Slots inclusive by increments of two Slots. The processes of channel ownership management and flow control in the Extended Asynchronous Protocol are identical to the ones in the Asynchronous Protocol.

The difference lies in the addition of a second TAG field Slot. This is written by the source of the Data and shall indicate how many bytes are valid (and associated to the P bit) in the DATA field because the data transfer length might not be a multiple of the DATA field length. This value is referred to as the number of valid bytes (NVB).

The Frame Structure does not allow more than 31 Slots in the Segment DATA field. Since the Extended Asynchronous Transport Protocol requires a DATA field that is an even number of bytes, NVB can take

values in the range 0 to 14. In any case, the largest value of NVB is bounded by the number of consecutive bytes available in the Data Space, which depends on the Subframe Mode and the Extended Asynchronous Data Channel's Segment Interval. See Section 6.3.1.4 and Section 6.4.2 for more information on Subframe coding and the Segment Interval, respectively.

If the channel is not owned by either the source Device or the sink Device, the second TAG field Slot shall not be driven.

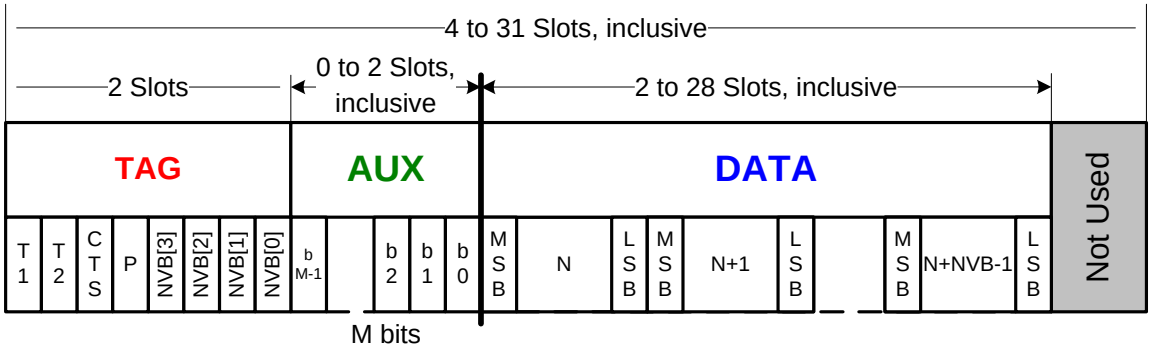


Figure 51 Extended Asynchronous Protocol Segment Organization

In addition to the information already conveyed by the TAG bits, the source Device shall indicate how many bytes are valid (and associated to the P bit) in the DATA field because the data transfer length might not be a multiple of the DATA field length. A second TAG field Slot (4-bits) is allocated to the coding of the number of valid bytes (NVB).

Table 56 TAG field Slot Layout for the Extended Asynchronous Protocol

Cell	Name	Source	Sink	Description
C7	T1	W	R	Primary Owner Token. Used by the primary owner to indicate that it owns the AUX+DATA fields
C6	T2	R	W	Secondary Owner Token. Used by the secondary owner to indicate that it owns the AUX+DATA fields
C5	CTS	R/W	W/R	Clear To Send. CTS=0 indicates that the source shall stop transmitting data
C4	P	W/R	R/W	PRESENCE bit. P=1 indicates that data is present in the current Segment
C3	NVB[3]	W/R	R/W	MSB of the NVB value
C2	NVB[2]	W/R	R/W	…
C1	NVB[1]	W/R	R/W	…
C0	NVB[0]	W/R	R/W	LSB of the NVB value

9.3.6.1 Example for the Extended Asynchronous Protocol (informative)

Table 57 gives an example of a sequence of events, making more explicit the mechanisms described in Section 9.3.6. The example assumes that the DATA field is sixteen Slots (eight bytes) wide and that a data transfer of twenty-one bytes (two packets of eight bytes and one packet of five bytes) shall be transported by the Data Channel.

2753 **Table 57 Example TAG Slot Sequence for the Extended Asynchronous Protocol**

Segment	T1	T2	CTS	P	NVB[3:0]	Comment
Segment i	1	0	1	0	0b0000	The sink sets the CTS bit to indicate to the source that it is ready to accept data.
Segment i+1	1	0	1	0	0b0000	Second occurrence of CTS=1. Flow is now on, but the source is not allowed to send data in this Segment.
Segment i+2	1	0	1	1	0b1000	Flow continues to be on, and the source writes DATA (first eight bytes) in the Segment (P=1).
Segment i+3	1	0	0	1	0b1000	Flow becomes invalid because CTS is zero, but the source writes DATA (second eight bytes) in the Segment (P=1).
Segment i+4	1	0	0	0	0b0000	Flow is now off, and the source is not allowed to write any further data until it is flowed on again.
Segment i+5	1	0	1	0	0b0000	The sink sets the CTS bit to indicate that it is ready to accept data again, but as flow is still invalid, the source cannot write DATA in this Segment (P=0).
Segment i+6	1	0	1	0	0b0000	Second occurrence of CTS=1. Flow is now on, but the source is not allowed to send DATA in this Segment.
Segment i+7	1	0	1	1	0b0101	Flow continues to be on, and the source writes DATA (final five bytes) in the Segment (P=1).
Segment i+8	1	0	1	0	0b0000	The source has finished its transmission and although still flowed on does not place DATA in the Segment (P=0).
Segment i+9	0	0	1	0	0b0000	The source ceases requesting ownership of the channel (T1=0).
Segment i+10	0	0	0	0	0b0000	The primary owner continues with T1=0, and the ownership state returns to *Neither*.
Segment i+11	0	0	Z	Z	0bZZZZ	The channel ownership state is now *Neither*.

2754 ***Notes***

2755 *"Z" represents a not driven condition.*

2756 ### 9.3.7 User Defined Protocol

2757 User protocols provide the ability to extend the SLIMbus data transmission mechanisms. When a User
2758 protocol is used in a Data Channel, it is assumed that a Device connected to the Data Channel knows the
2759 definition of the TAG bits and how they are used. A user protocol description should also define the default
2760 Read/Write states of these TAG bits.

2761 There are two IDs available for User defined protocols. The first protocol ID, User 1, indicates a protocol
2762 with one TAG field Slot. The second protocol ID, User 2, indicates a protocol with two TAG field Slots.

9.4 Auxiliary Bits Format

The auxiliary (AUX) bits carry side information linked to the content of the DATA field. They usually depend on the type of Device producing the data. The AUX bits shall be driven by the DATA source.

9.4.1 SPDIF Tunneling

The IEC 60958 [IEC01] standard defines 4-bits that are transmitted concurrently with audio data and framing information within an IEC 60958 sub-frame. It may be necessary to carry these bits along with any data on SLIMbus.

An IEC 60958 frame is uniquely composed of two sub-frames, each one made of 4-bit fields.

Figure 52 SPDIF Sub-frame Format

The first IEC 60958 sub-frame (left or "A" channel in stereophonic operation) normally starts with preamble "M". However, the preamble changes to preamble "B" once every 192 frames to identify the start of the block structure used to organize the channel status information. The second IEC 60958 sub-frame (right or "B" channel in stereophonic operation) always starts with preamble "W".

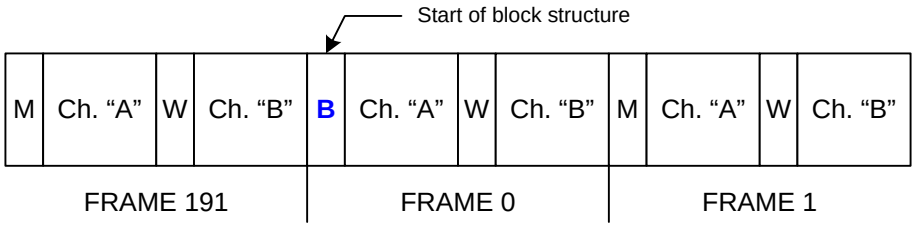


Figure 53 SPDIF Block Structure

Bits 28 to 31 are of interest for the definition of the AUX bit field. Bit 4 to 27 shall be carried in the DATA field.

- V is a validity bit, which is set when the channel does not carry linear PCM audio.
- U is the User Data bit, which is part of a block structure of 192-bits (see IEC60958).
- C is the channel status, which is part of a block structure of 192-bits (see IEC60958).
- P is the parity bit, computed in such a way that bits 4 to 31 inclusive carry an even number of ones and an even number of zeros (even parity).

The IEC60958 preamble is not transported with the data over SLIMbus. Therefore, the start of block structure must be announced in a different manner. The P bit can easily be reconstructed by an SPDIF transmitter and does not carry crucial information. A “Z” bit indicating the start of a block structure can replace it. The Z bit shall be changed to one every IEC60958 frame 0 and changed to zero for all other frames.

Table 58 AUX Bit Definition for IEC60958

Cell	Name	Description
C3	Z	Frame 0 of IEC60958 block
C2	C	Channel status bit, organized in blocks of 192-bits
C1	U	User Data bit, organized in blocks of 192-bits
C0	V	Validity. Set when the channel does not carry linear PCM audio

9.4.2 User-Defined AUX Field

A user-defined AUX field format provides the ability to define custom sideband information. When a user-defined AUX field format is used with a Data Channel, it is assumed that a Device connected to the Data Channel knows the definition of the AUX bits.

Two formats are provided for the AUX field, the first with four AUX bits and the second with eight AUX bits.

9.4.3 AUX Field Format ID

Table 59 lists the AUX Field Format IDs that shall be used when defining a channel. A Device shall not send a reserved AUX Field Format code when defining or updating the content of a Data Channel. A Device that receives a “Reserved” AUX format codes shall behave as if it received an unsupported AUX format code.

If a Data Source does not support the AUX format as specified for a Data Channel, it shall fill the AUX field with zeroes. A Data Source that receives an unsupported AUX format should set its EX_ERROR Information Element if it supports such range checking. Refer to Section 11.6 for more details on parameter range checking.

If a Data Sink does not support the AUX format as specified for a Data Channel, the Data Sink shall ignore the contents of the AUX field. A Data Sink that receives an unsupported AUX format may set its EX_ERROR Information Element if it supports such range-checking.

Table 59 AUX Field Format ID

AF[3:0]	Number of AUX field Slots	Usage of AUX field Slots
0b0000	0	Not applicable
0b0001	1	ZCUV for tunneling IEC60958
0b0010	1	Reserved
0b0011	1	
0b0100	1	
0b0101	1	
0b0110	1	
0b0111	1	
0b1000	1	
0b1001	1	
0b1010	1	
0b1011	1	User defined

AF[3:0]	Number of AUX field Slots	Usage of AUX field Slots
0b1100	2	Reserved
0b1101	2	
0b1110	2	
0b1111	2	User defined

9.5 Data Type Field

The Data Type field identifies the DATA field content of a Segment and shall be coded as shown in Table 60. The Data Type field is used in the NEXT_DEFINE_CONTENT and CHANGE_CONTENT Messages.

Table 60 Data Types

DT[3:0]	Data Type
0b0000	Not Indicated
0b0001	LPCM audio (offset sign-and-magnitude, MSB justified, MSB first)
0b0010	IEC61937 Compressed audio
0b0011	Packed PDM audio
0b0100 to 0b1101	Reserved
0b1110	User Defined 1
0b1111	User Defined 2

A source Device may use the “Not indicated” Data Type code to inform the destination Device the contents of a Data Channel are not one of the defined Data Types or are unidentified.

A Device shall not define nor update the content of a Data Channel using a “Reserved” Data Type. A Device that receives a NEXT_DEFINE_CONTENT or CHANGE_CONTENT Message with a “Reserved” value in the Data Type field shall behave as if it received a “Not Indicated” Data Type.

A Port shall behave as if it received a “Not Indicated” Data Type when a NEXT_DEFINE_CONTENT or CHANGE_CONTENT Message specifies a Data Type that the Port does not support.

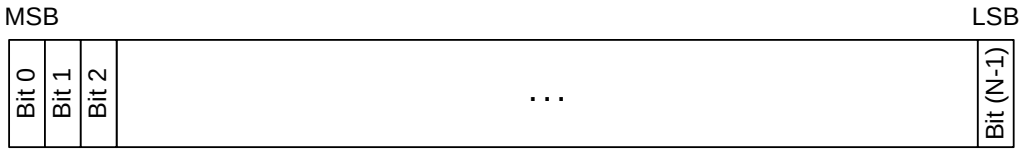
There shall be exactly one audio channel per Data Channel for the LPCM audio Data Type. The audio samples on the DATA line shall use the offset sign-and-magnitude number format. Applications typically use a two’s-complement number format for LPCM audio samples. Consequently, Sources and Sinks of LPCM audio will likely need to translate between these two formats. The translation between formats is inexpensive, especially when done bit-serially in hardware. Table 61 shows common values in both offset sign-and-magnitude and two’s-complement formats.

Table 61 LPCM Audio Number Format Comparison

Signed Fraction	Offset Sign-and-Magnitude	Two’s-complement
+ 32767 / 32768	b_0_111111111111111	b_0_111111111111111
+ 127 / 32768	b_0_000000001111111	b_0_000000001111111
+ 7 / 32768	b_0_000000000000111	b_0_000000000000111
+ 3 / 32768	b_0_000000000000011	b_0_000000000000011
+ 1 / 32768	b_0_000000000000001	b_0_000000000000001

Signed Fraction	Offset Sign-and-Magnitude	Two's-complement
0	b_0_000000000000000	b_0_000000000000000
- 1 / 32768	b_1_000000000000000	b_1_111111111111111
- 3 / 32768	b_1_000000000000010	b_1_111111111111101
- 7 / 32768	b_1_000000000000110	b_1_111111111111001
- 127 / 32768	b_1_000000011111110	b_1_111111100000001
- 32767 / 32768	b_1_111111111111110	b_1_000000000000001

With quiet audio signals, the offset sign-and-magnitude format has the majority of its bits equal to zero. Since the DATA line uses NRZI coding, power consumption in the physical layer is reduced for quiet signals. In addition, conducted and radiated interference tends to be less with the offset sign-and-magnitude format than with the two's-complement format.



N: Number of PDM bits in Segment **DATA** Field

Figure 54 Packed PDM Segment DATA Field

The Packed PDM audio Data Type concatenates a number of consecutive PDM stream bits into the DATA field of a Segment as shown in Figure 54. A one-bit PDM stream is assumed for this Data Type. The PDM transfer shall be lossless, i.e., bits shall not be omitted. The Data Channel bandwidth shall be greater than, or equal to, the PDM bit rate. The MSB shall contain the oldest sample (bit), and the LSB shall contain the most recent sample.

9.6 Presence Rate

The Presence Rate of a constant bitrate stream is the mean flow rate of that stream expressed in occupied Segments of that Data Channel per second. For variable bitrate streams it is an upper bound on the flow rate, also expressed in occupied Segments per second.

Table 62 Presence Rate

	PR[6:3]			
PR[2:0]	0b0000	0b0001	0b0010	0b0011 to 0b1111
0b000	Not Indicated	Reserved	4k	Reserved
0b001	12k	11.025k	8k	
0b010	24k	22.05k	16k	
0b011	48k	44.1k	32k	
0b100	96k	88.2k	64k	
0b101	192k	176.4k	128k	
0b110	384k	352.8k	256k	
0b111	768k	705.6k	512k	

For each Presence Rate defined in Table 62, Annex C identifies the Natural Frequencies for all Clock Gears.

A Device shall neither define nor update the content of a Data Channel using a “Reserved” Presence Rate. A Device that observes such a code in a Data Channel's content definition or update Message shall behave as if it received a “Not Indicated” Presence Rate.

9.7 DATA Field Length

Information on the length of the DATA field is carried via DL[4:0] in the NEXT_DEFINE_CONTENT and CHANGE_CONTENT Messages.

Table 63 DATA Field Length

DL[4:0]	Description
0b00000	Reserved
0b00001 to 0b11111	1 to 31 Slots, respectively

A Device shall not define nor update the content of a Data Channel using a “Reserved” Data Length. The Device behavior is not specified when the Device receives a CHANGE_CONTENT or NEXT_DEFINE_CONTENT Message with a “Reserved” value in DL.

10 Bus Processes

This section describes and specifies several key SLIMbus processes, bringing together aspects from all sections. Message sequence charts are used extensively to describe the Message interactions in the Message Channel. All charts and text assume that Messaging is error free, i.e. no Messages are corrupted or are lost, unless explicitly mentioned.

Figure 55 shows an example of a simple Message sequence.

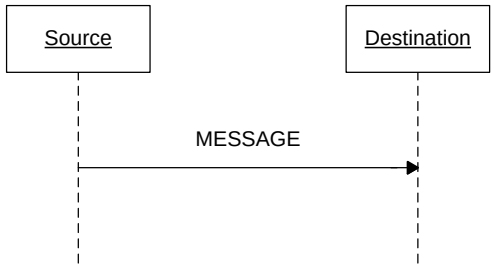


Figure 55 Message Sequence Example

Some of the transactions are split. The request and reply are contained in separate Messages. Figure 56 contain an example of the split transaction.

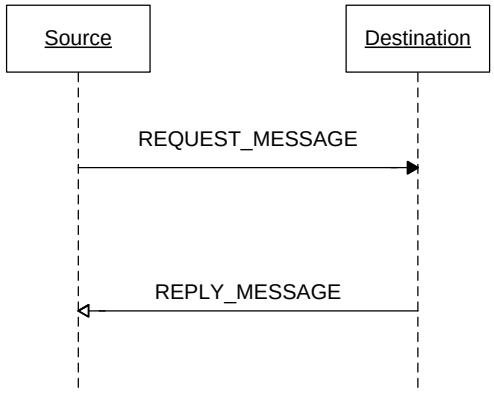


Figure 56 Split Transaction Example

10.1 Boot and Synchronization Processes

The SLIMbus boot and synchronization processes have been divided into different phases. For each phase, the allowed behavior of a Component on the bus and the condition for switching to the next phase is defined in this section.

A Component shall be capable of joining the bus by following the appropriate boot and synchronization processes. The active Framer shall boot according to Section 10.1.1. Clock Receiving and Clock Sourcing Components shall synchronize according to Section 10.1.2 and Section 10.1.3, respectively.

A Component cycles through various states, sometimes at a different rate than other Components. At any given point in time, there is not necessarily a single state that represents all Components on the bus. Therefore, the state diagrams in this section are described from the viewpoint of a single Component.

In order to have more control over system power consumption, the SLIMbus protocol allows a Component to stop participating on the bus while the SLIMbus is active.

This section also describes the behavior of a Component after it has detected loss of Frame, Superframe or Message synchronization. A Component shall implement the necessary recovery mechanisms to deal with the loss of Superframe synchronization, Frame synchronization, and Message synchronization.

10.1.1 Active Framer Boot Sequence

The Clock Sourcing Component is responsible for booting the SLIMbus. As such, it is responsible for starting the CLK, for writing the information in the Framing Channel and for transmitting the Guide Channel. During the boot process, the active Framer shall progress through five different states: *Undefined*, *CheckingDataLine*, *StartingClock*, *StartingData*, and *Booted*. A state diagram that describes the mandated behavior can be found in Figure 57. Figure 58 shows the detectable behavior of the Clock Sourcing Component on the bus.

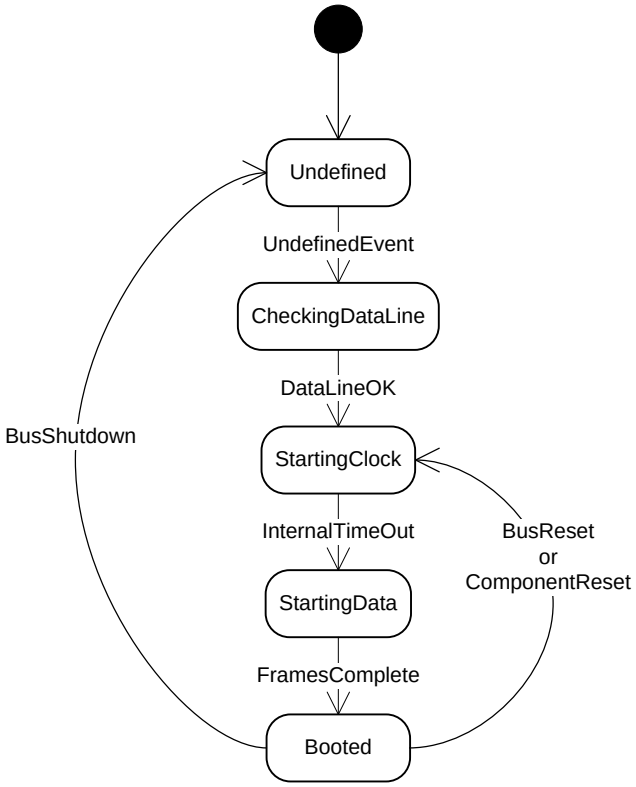


Figure 57 Boot Process for the Active Framer

10.1.1.1 Undefined State

In the *Undefined* state, no requirement on the behavior of the CLK and DATA terminals is made in this document. The event that initiates the change to the *CheckingDataLine* state is also outside of the scope of this document.

10.1.1.2 CheckingDataLine State

The *CheckingDataLine* state allows the active Framer to verify that the DATA line is available for use, that the bus holders are operational and that Devices connected to the bus are in a state suitable to start driving the CLK line.

In the *CheckingDataLine* state, the active Framer shall transmit the Framing Channel and the Guide Channel on the DATA line. The active Framer may continuously drive the CLK line below VOL or toggle the CLK line. The active Framer shall read the state of the DATA line on every negative clock edge (internal or CLK), and perform the following checks. In the driven Cells, i.e. in the Framing Channel Slots and the Guide Channel Slots, the active Framer shall verify that the read state of the DATA line is the same as the driven state. For undriven Cells, the active Framer shall verify that the read state of the DATA line is the same as the read state for the previous Cell.

The active Framer shall not move to the *StartingClock* state until there have been eight consecutive Frames with no DATA line-check failures. The active Framer may implement a time out function and return to the *Undefined* state if it is not able to continue to the *StartingClock* state within a reasonable period of time. This period of time is beyond the scope of this specification.

The active Framer shall have all Framing Information, and any additional information, necessary to start the bus before this state is reached. Any requirements for obtaining this information shall be described in the Framer documentation.

10.1.1.3 StartingClock State

In the *StartingClock* state, the active Framer shall start clocking (toggling) the CLK line. The Clock Sourcing Component shall refrain from sending any data (including the Framing Channel) on the DATA line, for four Frames. After four Frames, the active Framer shall change to the *StartingData* state.

The initial frequency for the CLK line is not mandated in this document. This state can be used to stabilize the CLK frequency, slew rate or to test the CLK line to otherwise optimize the clock driver circuit, so long as this does not violate the constraints in Section 5.2.2.

10.1.1.4 StartingData State

In the *StartingData* state, the active Framer shall continue clocking the bus. The Framing Channel and Guide Channel shall be written by the active Framer while in the *StartingData* state, enabling rapid synchronization of the Devices on the bus.

The active Framer shall not transmit any Messages while in this state. Starting at the Superframe boundary, the active Framer shall transmit sixteen Frames and then change to the *Booted* state.

10.1.1.5 Booted State

In the *Booted* state, the active Framer continues to write the Framing Channel and Guide Channel to the bus. While the Clock Sourcing Component is not in Message synchronization (Section 8), the active Framer shall write the ENT bit and the Guide Value as zero. An active Framer in the *Booted* state can be moved to the *StartingClock* state as described in Section 10.4.1 and Section 10.4.2.

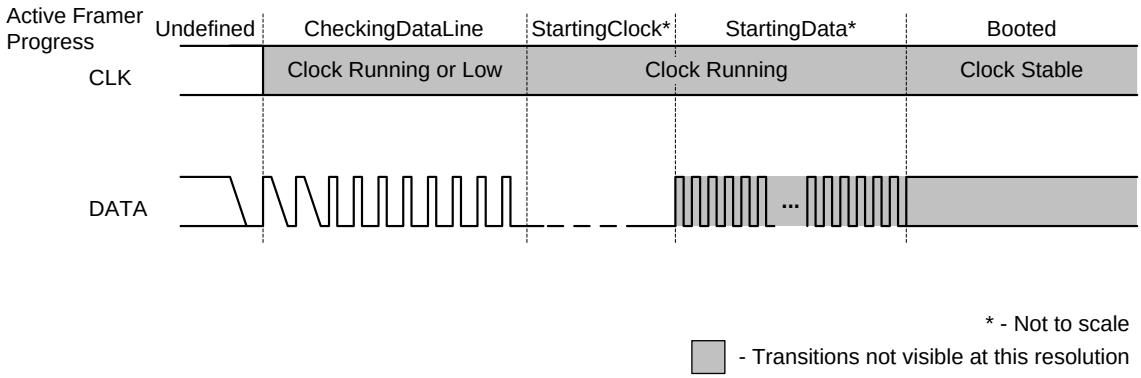


Figure 58 Example CLK and DATA Line Behavior during Boot

10.1.2 Component Synchronization – Clock Receiver

A Clock Receiving Component shall support the synchronization process described in this section for any combination of Clock Gear, Root Frequency and Subframe Mode.

During the synchronization process, a Clock Receiving Component extracts any necessary information about the current state of the bus from the two bus lines. Therefore, changes between states are initiated by observing the CLK and the DATA line.

The synchronization process takes a Component through six states: *Undefined*, *Reset*, *SeekingFrameSync*, *SeekingSuperframeSync*, *SeekingMessageSync* and *Operational*. See Figure 59. A Component that does not contain the active Framer shall only transmit data on the SLIMbus while in the *Operational* state or the *SeekingMessageSync* state.

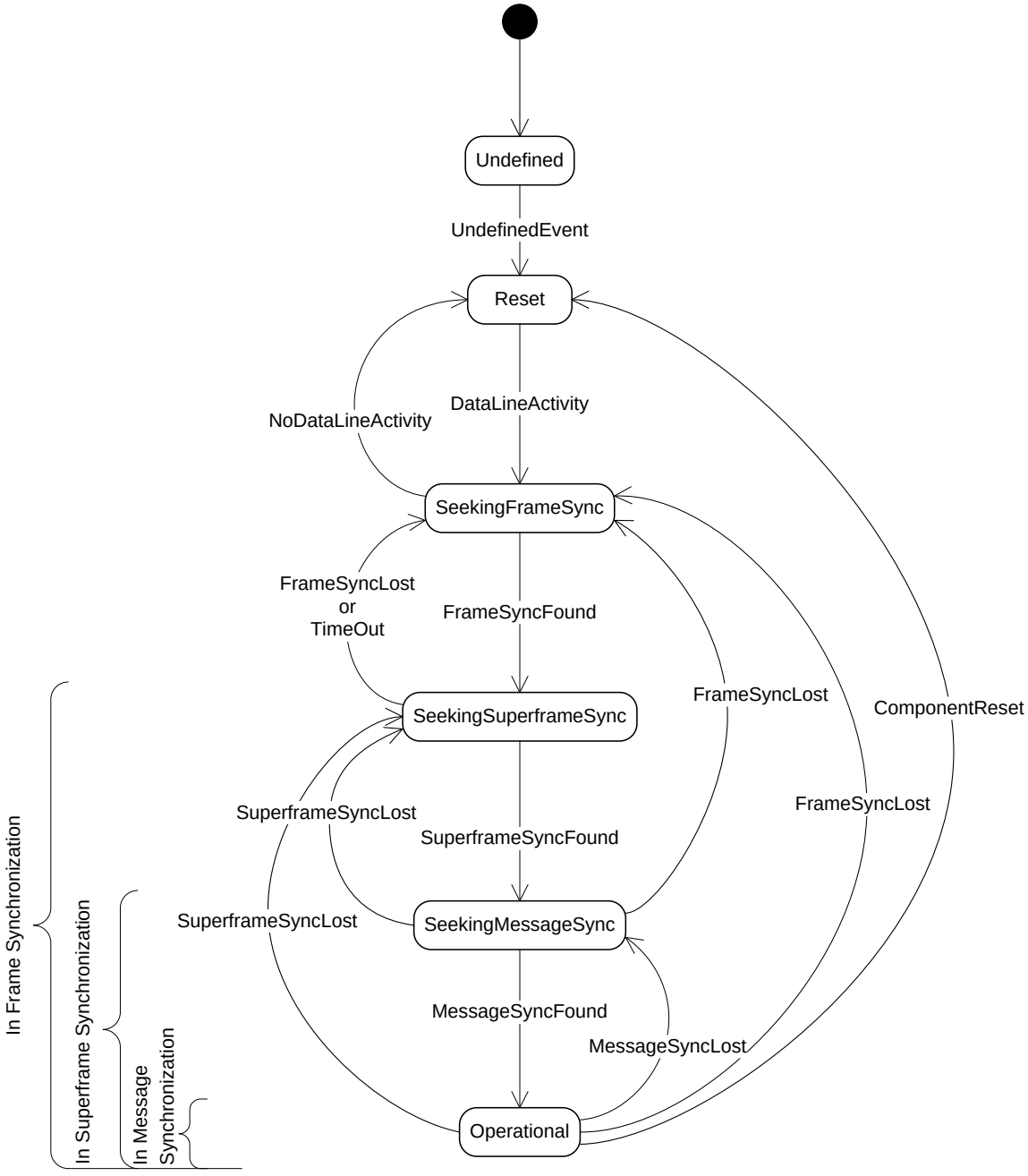


Figure 59 Component Synchronization Process

A Component resides in the *Undefined* state if it is not tracking events on the SLIMbus. An external event (outside the scope of this specification) initiates the change to the *Reset* state (Figure 59, *UndefinedEvent* transition). A transition on the DATA line causes the Component to move to the *SeekingFrameSync* state. The Component remains in this state until three Frame Sync symbols have been detected. Then a change to the *SeekingSuperframeSync* state occurs (Figure 59, *FrameSyncFound* transition), where the Framing Channel is parsed until the Superframe boundary has been found, and the complete Framing Information has been received and processed to find the location of the Message Channel. After a valid Guide Byte is received, the Component moves to the *Operational* state (Figure 59, *MessageSyncFound* transition).

Often, a Component begins synchronizing to the bus as the active Framer starts booting. In such cases, a Component's progress through its synchronization states tends to be smooth and deterministic. In contrast,

when a Component synchronizes to a bus that is already active, the time it takes to reach the *Operational* state is non-deterministic. This is due to the possible occurrence of the Frame Sync symbol in other bus traffic.

10.1.2.1 Undefined State

In the *Undefined* state, the behavior of the Component is not characterized. A chip-level reset signal or software event could be used to bring the Component out of the *Undefined* state though the specific event that initiates the change of state is outside the scope of this document. A Component in the *Undefined* state shall not impede the bus at any time.

10.1.2.2 Reset State

In the *Reset* state, a Component shall not drive the bus and shall not impede other Components from becoming operational. If the Component is ready to join the bus, the Component shall sample the DATA line at every negative edge of the CLK line. As soon as a change on the DATA line is detected, the Component should enter the *SeekingFrameSync* state (Figure 59, *DataLineActivity* transition).

If a Component moves to the *Reset* state from the *SeekingFrameSync* or *Operational* states, the Component shall cause each Device in the Component to undergo a Device Reset (Section 10.4.3) and release its Logical Address.

10.1.2.3 SeekingFrameSync State

In the *SeekingFrameSync* state, Components shall search for repetitions of the Frame Sync symbol. A Component detecting three Frame Sync symbols repeating at an interval of 192 Slots (768 Cells) shall change to the *SeekingSuperframeSync* state (Figure 59, *FrameSyncFound* transition). During boot, this transition occurs 385 Slots (1540 Cells) after the first bit of the first Frame Sync symbol has been detected. The beginning of the first Cell of each of the detected Frame Sync symbols is a Frame boundary. As indicated in Section 6, Frame boundaries are also Subframe boundaries and Slot boundaries. A Component that has achieved Frame Sync may begin treating the incoming data in Slot form.

In order to provide a reset mechanism for a Component that has not achieved Frame synchronization, the Component shall monitor DATA line activity. The Component shall move to the *Reset* state if it detects DATA line inactivity (no transitions on the DATA line) for 1536 consecutive Cells (Figure 59, *NoDataLineActivity* transition). It may move to the *Reset* state after fewer than 1536 Cells. However, the Component shall not move to the *Reset* state before it detects DATA line inactivity of at least 768 consecutive Cells.

A Component that has achieved Frame synchronization shall continuously verify Frame Sync symbols. If verification fails in two consecutive Frames, a Component shall move to the *SeekingFrameSync* state (Figure 59, *FrameSyncLost* transitions). The Component shall make this move no more than 764 CLK periods after the second Frame Sync symbol verification failure and shall cause all of its Devices to reset their Ports, as defined in Section 10.4.4. Isolated errors in the verification of Frame Sync symbols shall be ignored.

10.1.2.4 SeekingSuperframeSync State

In the *SeekingSuperframeSync* state, a Component verifies the contents of the Framing Information. See Section 6.3.1.1.

A Component shall determine from the Superframe Sync pattern which of the eight Frame boundaries is the Superframe boundary.

A Component shall not change to the *SeekingMessageSync* state before it has verified the Superframe Sync pattern over sixteen consecutive Frames. If the Component has not verified the Superframe Sync pattern after thirty-two Frames, it shall change to the *SeekingFrameSync* state (Figure 59, *TimeOut* transition) and seek Frame synchronization at a different phase.

For Superframes with FE=0, a Component shall read the SM field of the Framing Information and shall not change to the *SeekingMessageSync* state until it has observed the contents of this field as being identical across two Superframes.

In addition, if the Component uses the CG or RF field it shall read the used field and shall not change to the *SeekingMessageSync* state until it has observed the contents as being identical across two consecutive occurrences of the field.

A Component that has achieved Superframe synchronization shall continuously verify the Superframe Sync pattern. If verification fails in two consecutive Superframes, the Component shall move to the *SeekingSuperframeSync* state by the end of the Superframe (Figure 59, *SuperframeSyncLost* transitions). Isolated verification errors shall be ignored. A Component shall also continuously verify the SM, and those of the CG and RF Framing Information fields that they use. For each of those fields, if the Component sees a verification failure in the last two Superframes with FE=0, it shall move to the *SeekingSuperframeSync* state by the end of the Superframe (Figure 59, *SuperframeSyncLost* transitions). Other conditions that cause the Component to move to the *SeekingSuperframeSync* state are detailed in Section 10.2.4 and Section 10.3.1.

A Component shall cause all Devices within the Component to reset their Ports as defined in Section 10.4.4 when moving from the *SeekingMessageSync* or *Operational* state to the *SeekingSuperframeSync* state.

10.1.2.5 SeekingMessageSync State

A Component shall observe the Guide Channel and try to achieve Message synchronization as mandated by Section 8.3.1. Once the Component has achieved Message synchronization, it shall enter the *Operational* state.

A Component that has achieved Message synchronization shall continuously verify the Guide Channel and the Message Channel as described in Section 8.3.2. Various events can cause a Component to assume it has lost Message synchronization. Refer to Section 8 and Section 10.2.3 for details. If this happens, the Component shall move to the *SeekingMessageSync* state.

10.1.2.6 Operational State

In the *Operational* state, a Component has coherent information on the Subframe Mode and the other information in the Framing Channel and has access to the Message Channel.

A Component Reset (see Section 10.4.2) moves a Component in the *Operational* state to the *Reset* state.

10.1.3 Component Synchronization – Clock Source

The Message synchronization requirements in Section 8.3, Section 10.2.3 and Section 12.2.5.2 apply to the Clock Sourcing Component just as much as they apply to Clock Receiving Components. The same is true of the Bus Reset and Component Reset requirements in Section 10.4.1 and Section 10.4.2. Consequently, a Component has to support the *Reset*, *SeekingMessageSync* and *Operational* states of Figure 59, even while it is the Clock Sourcing Component.

Referring to Figure 59, a Component should support all of the states in the figure while it is the Clock Sourcing Component. While a Component is the Clock Sourcing Component, it may use information

passed internally from the active Framer to progress more smoothly through the *SeekingFrameSync* and *SeekingSuperframeSync* states.

10.1.4 Example of the Boot and Synchronization Processes (informative)

Figure 60 shows the synchronization process of a Component that synchronizes at the same time as the active Framer boots, and its reaction to the signals it receives. The Component autonomously reaches the *SeekingFrameSync* state, acquires Frame synchronization and starts on the Superframe synchronization. Note, at this resolution the Message synchronization state is not visible.

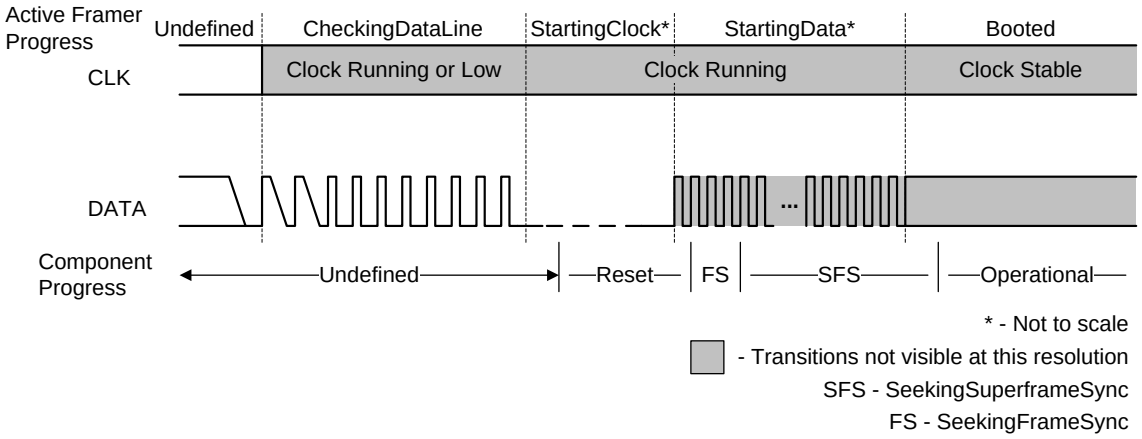


Figure 60 Typical SLIMbus Boot Behavior

10.2 Error Handling

A Device shall maintain a number of Information Elements that can be communicated over the SLIMbus. These Information Elements are defined in Section 7.1.2. Additionally, Class-specific Information Elements have been defined for Interface Devices, Managers and Framers described in Section 12.2.4, Section 12.3.4 and Section 12.4.4, respectively. This section describes a number of detectable errors that influence these Information Elements.

10.2.1 Transmission Errors

Collisions can occur in the Control Space and Data Space.

For instance, the following non-exhaustive list of possibilities can contribute to Collision in the Data Space:

- Incorrect programming of the channel parameters, e.g. Segment Distribution and Segment Length.
- Activating two or more sources with the same Channel Number

For the Control Space, the following non-exhaustive list of possibilities can contribute to Collisions:

- Two or more Devices using the same Logical Address
- Incorrectly interpreting the Frame and Superframe boundaries
- Incorrectly interpreting the current Subframe Mode
- Incorrectly interpreting Message boundaries

The DATA line is monitored for Collisions using the mechanism described in Section 5.1.1.

Depending on whether the Collision is detected in the Data Space or Control Space, the Component shall set the appropriate Information Element of the relevant Device.

10.2.2 Behavior on Collisions within Segments

If a Device detects a Collision during transmission of a Segment, it shall cease the transmission before the start of the next Cell. The Device shall resume transmitting at the beginning of the next Segment, where the next DATA Segment is transmitted. The Segment that experienced the collision is not retransmitted. The Device shall also change the DATA_TX_COL Core Information Element to TRUE at the start of transmit back-off. See Section 7.1.2.5.

Note that a Collision in a Data Channel does not cause a Component to stop sending in the Framing Channel, Message Channel, or any other channel.

10.2.3 Behavior on Collisions within the Message Channel

A Component shall initiate a transmit back-off when a Collision is detected in the Message Channel and set the MC_TX_COL Information Element in its Interface Device. See Section 12.2.5.5. During the transmit back-off, the SLIMbus Component shall not attempt any further transmission in the Message Channel. A transmit back-off shall commence before the beginning of the next Cell. A Component shall lose Message synchronization and return to the *SeekingMessageSync* state. The Component shall need to regain Message synchronization by waiting for the next valid Guide Byte.

If the Component was transmitting a Message, the SLIMbus Component should retransmit the complete Message, using the normal arbitration mechanism, after it has regained Message synchronization.

Immediately after the transmit back-off is initiated, the Component should start the Message synchronization process. See Section 8.3. The transmit back-off shall be terminated as soon as Message synchronization is regained.

Note that a Collision in the Message Channel does not cause a Component to suspend a transmission in the Framing Channel or a Data Channel.

10.2.4 Framing Errors

The mechanisms described in Figure 59 also play a role in recovering from bit errors, spurious CLK line transitions and some of the higher protocol errors on the SLIMbus. Spurious CLK line transitions result in a Component moving from the *Operational* state to the *SeekingFrameSync* state because verification of the Frame Sync symbol fails. In addition, a Component continuously verifies one or more of the Framing Information fields. If, for some reason, the information changes without detecting the appropriate Message in the Message Channel, a Component shall assume that it has missed a vital bus Message and change to the *SeekingSuperframeSync* state (see Figure 59).

10.2.5 Messaging Errors

A Device shall set the UNSPRTD_MSG Information Element if it receives a unicast Message that it does not implement as a destination Device. Note that this applies for all Message Types as found in Table 32. The Device shall not act upon an unsupported Message with the exception of setting the UNSPRTD_MSG Information Element.

A destination Device shall not set the UNSPRTD_MSG Information Element if it receives a broadcast Message that is optional for the Device Class to which the destination Device belongs.

A destination Device shall discard a Message it determines to be unsupported.

10.2.6 Reporting Core Information

A Device may report the value of Core Information Elements to other Devices using the REPORT_INFORMATION Message (Section 11.3.5). For more information on reporting Information Elements see Section 7.1.3.

10.3 Bus Management

A number of bus processes require that all Components on the bus perform a change of state at exactly the right moment (at a Superframe boundary). If a Device objects to a particular change of state, the active Manager shall configure the Device so that it does not interfere with bus operation while in the new bus state.

This section captures these processes and specifies necessary timing and behavior of the different Components and Devices.

10.3.1 General Behavior

The bus processes described in this section all follow the same sequence: A BEGIN_RECONFIGURATION Message, followed by a number of Reconfiguration Messages that can be identified by the NEXT_ prefix, are terminated by the RECONFIGURE_NOW Message, which causes all Devices to change their mode of operation at a well defined Superframe boundary. Just like all the other Messages, Reconfiguration Messages shall behave as though they are executed in order. Figure 61 shows an example of three Reconfiguration Messages followed by a RECONFIGURE_NOW Message.

The active Manager shall indicate the start of a Reconfiguration Sequence by broadcasting the BEGIN_RECONFIGURATION Message. No Reconfiguration Messages shall be transmitted without first sending the BEGIN_RECONFIGURATION Message. The active Manager shall end a Reconfiguration Sequence with the RECONFIGURE_NOW Message.

Certain combinations of NEXT_ Messages are prohibited because they can lead to undefined behavior. For instance, NEXT_PAUSE_CLOCK and NEXT_ACTIVE_FRAMER Messages in the same Reconfiguration sequence could cause troubles for the Clock Sourcing Components.

A Component shall assume that it has received all Reconfiguration Messages if it has not lost Message synchronization between receiving a BEGIN_RECONFIGURATION Message and a RECONFIGURE_NOW Message. If a Component receives a RECONFIGURE_NOW Message without first receiving a corresponding BEGIN_RECONFIGURATION Message, or if a Component loses Message synchronization between the BEGIN_RECONFIGURATION and RECONFIGURE_NOW Messages, the Component shall set the EX_ERROR Information Element as defined in Section 7.1.2.3 if the EX_ERROR Information Element is implemented. In addition, a Clock Receiving Component shall lose Superframe synchronization at the Reconfiguration Boundary (Figure 59, *SuperframeSyncLost* transition).

The effect of the RECONFIGURE_NOW Message is the cumulative effect of all Reconfiguration Messages, as though they are executed in order within a single Cell period. This implies that multiple Reconfiguration Messages that change the same bus property, such as Clock Gear, behave as though only the last change was received.

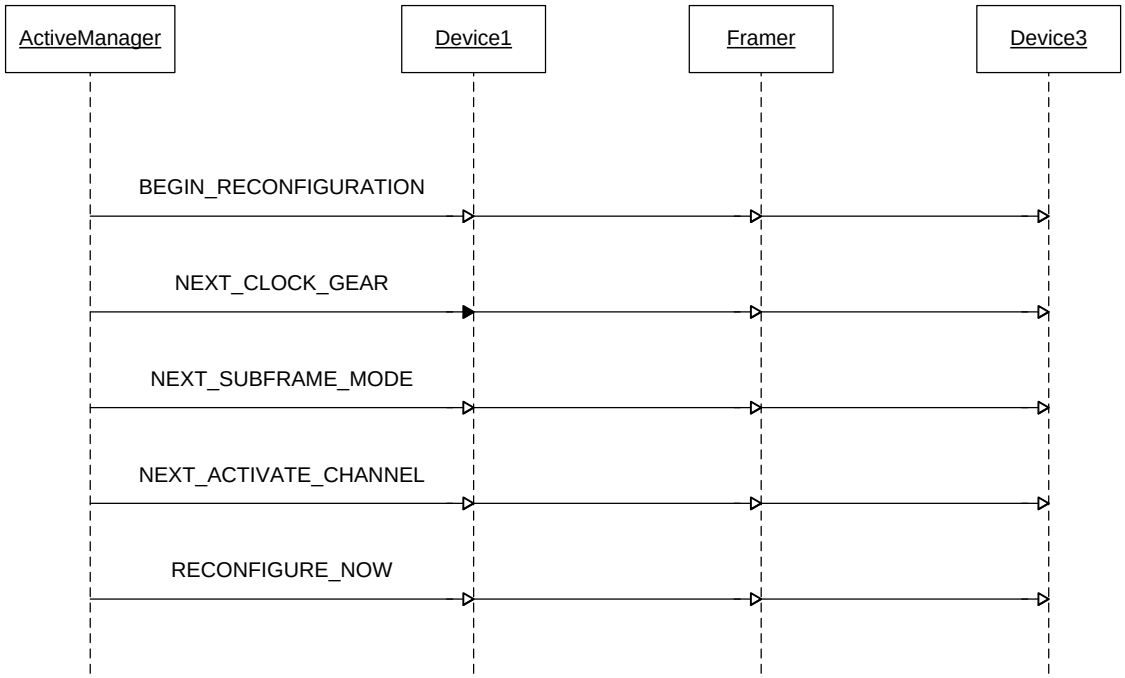


Figure 61 Example Bus Management Sequence

10.3.1.1 RECONFIGURE_NOW Timing

Pending changes shall take effect at the first Superframe boundary that comes at least two Slots after the end of the RECONFIGURE_NOW Message. This Superframe boundary is called the Reconfiguration Boundary. See Figure 62 and Figure 63. Note, a Component that does not observe a PACK in response to a RECONFIGURE_NOW Message shall not execute the Reconfiguration Messages. A Device within the Component shall not discard the announced changes. As with any Message that does not receive a PACK, the active Manager should retransmit the RECONFIGURE_NOW Message if a change to the bus configuration is still required.

A Component shall behave as though all Reconfiguration Messages are executed at the associated Reconfiguration Boundary. Note that although the number of Messages in a Reconfiguration Sequence is not limited, the number of parameters that the Messages effect is limited.

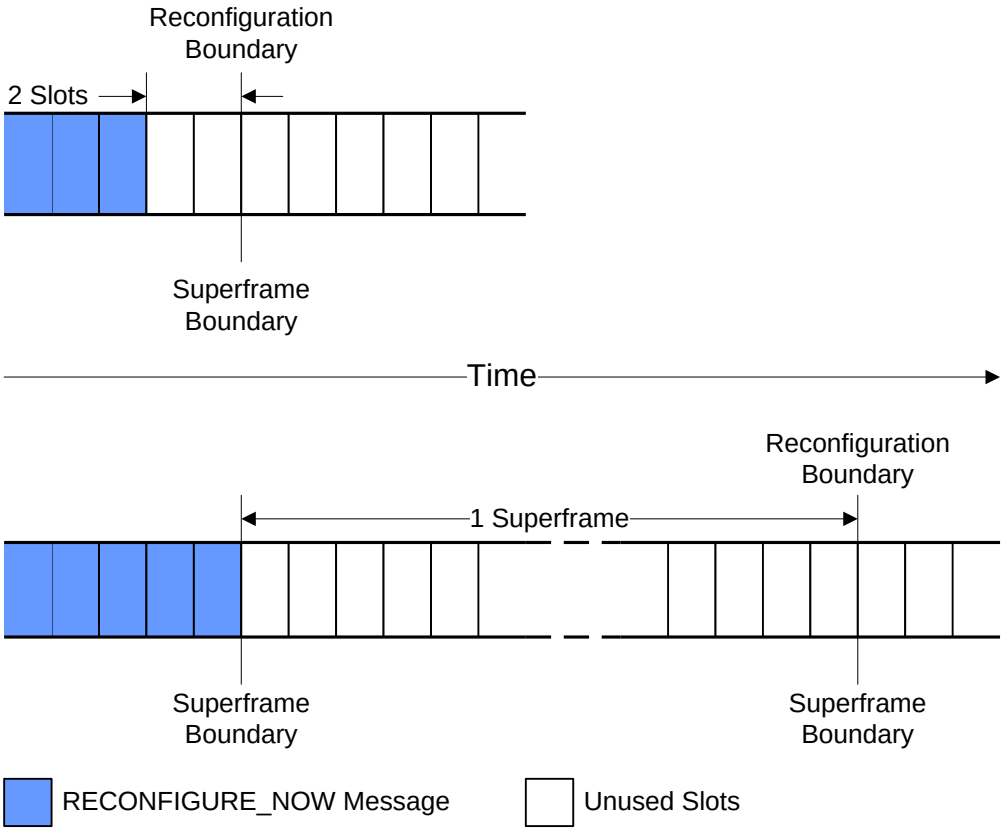


Figure 62 RECONFIGURE_NOW Message Timing

10.3.1.2 Behavior after a RECONFIGURE_NOW

After a RECONFIGURE_NOW Message, a Component and its Devices continue operation until the Reconfiguration Boundary. At that moment, the Component shall change its behavior to reflect all of the pending Messages.

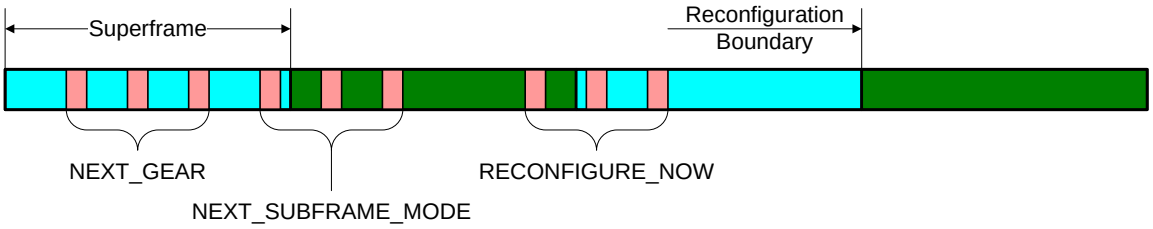


Figure 63 Using RECONFIGURE_NOW to Switch between Operating Mode

A Device might not be able to execute the pending changes of a RECONFIGURE_NOW Message at the Reconfiguration Boundary, because it might not have received all Reconfiguration Messages. This occurs for instance when a Component becomes synchronized to the SLIMbus after the first NEXT Message has been sent, but before the RECONFIGURE_NOW Message.

Figure 64 shows what happens when a Component that does not have the necessary information tries to execute the RECONFIGURE_NOW Message. The Component just gains Superframe synchronization after the first Superframe, only for its Devices to receive a RECONFIGURE_NOW Message.

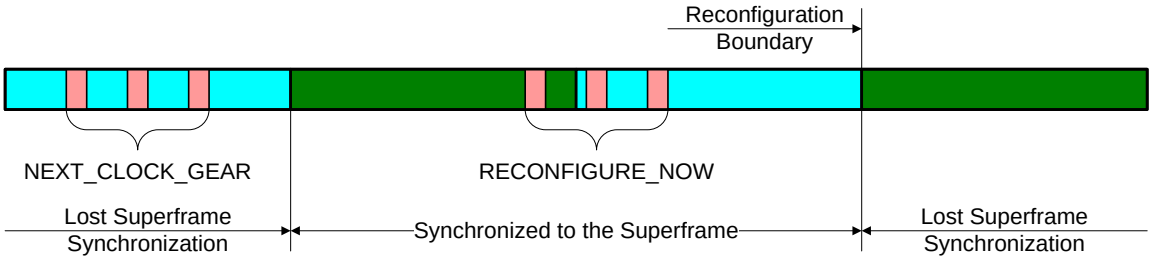


Figure 64 Loss of Superframe Synchronization after RECONFIGURE_NOW Message

10.3.1.3 Behavior of the BEGIN_RECONFIGURATION Message

The BEGIN_RECONFIGURATION Message indicates to a Device the start of a Reconfiguration Sequence. If the active Manager needs to cancel all announced changes, it shall send a second BEGIN_RECONFIGURATION Message. The reception of a BEGIN_RECONFIGURATION Message shall clear all pending bus state changes that have been communicated by Reconfiguration Messages.

10.3.1.4 Combinations of Reconfiguration Messages

The order of the Reconfiguration Messages influences the result of the RECONFIGURE_NOW Message. Multiple Reconfiguration Messages can influence the same parameter in a Component, e.g. the Clock Gear. In this case, the most recently received value shall be used to update the parameter.

Reconfiguration Messages that receive a negative acknowledge shall be discarded by a Component.

After the RECONFIGURE_NOW Message is sent, the active Manager shall not broadcast any Reconfiguration Messages until after the Reconfiguration Boundary. The active Manager may prevent error conditions from occurring by simply delaying all of its outgoing Messages until after the Reconfiguration Boundary.

10.3.1.5 Verification of Announced Changes

At any point during the Reconfiguration Messages, the active Manager may interact with any or all of the SLIMbus Devices to verify whether the Device can still operate under the announced conditions. A Device may implement the RECONFIG_OBJECTION Information Element as specified in Section 7.1.2.4. The active Manager shall use the REQUEST_INFORMATION and the REPLY_INFORMATION Message pair for this interaction. The active Manager may use this mechanism before sending the RECONFIGURE_NOW Message. See Figure 65 for an example.

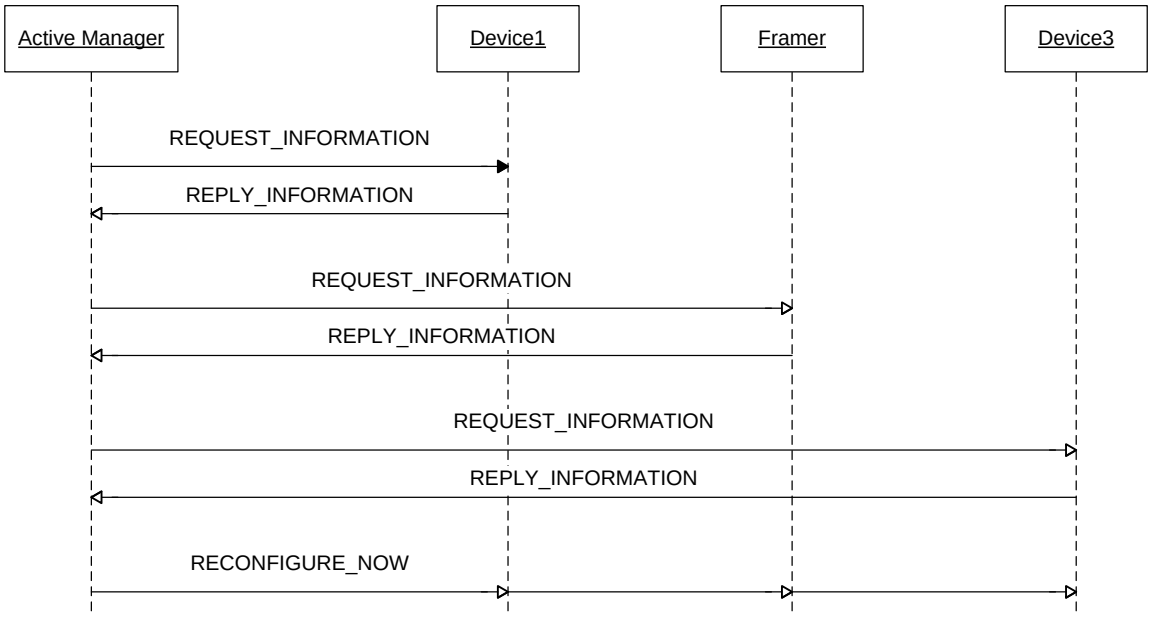


Figure 65 Verification of Reconfiguration Message Example

10.3.2 Pausing and Restarting the Bus Clock

The active Manager may request the active Framer pause the bus clock at the associated Reconfiguration Boundary by sending a NEXT_PAUSE_CLOCK Message during a Reconfiguration sequence. Just like most Reconfiguration Messages, the NEXT_PAUSE_CLOCK Message can be stacked with any other Reconfiguration Message.

If any other Reconfiguration Messages have been sent, their changes take effect when the bus is restarted after a Clock Pause. For example, if a NEXT_CLOCK_GEAR Message has been sent, the active Framer restarts the bus in the announced Clock Gear.

The payload of the NEXT_PAUSE_CLOCK Message defines the clocking recovery behavior of the active Framer. A Framer shall support all Clocking Recovery strategies listed as “Mandatory”, and may also support any Clocking Recovery strategy listed as “Optional”, in Table 66. An active Framer that receives an unsupported value shall behave as if it received an Unspecified Delay value.

The behavior during a Clock Pause is shown at the bus level in Figure 66 and Figure 67.

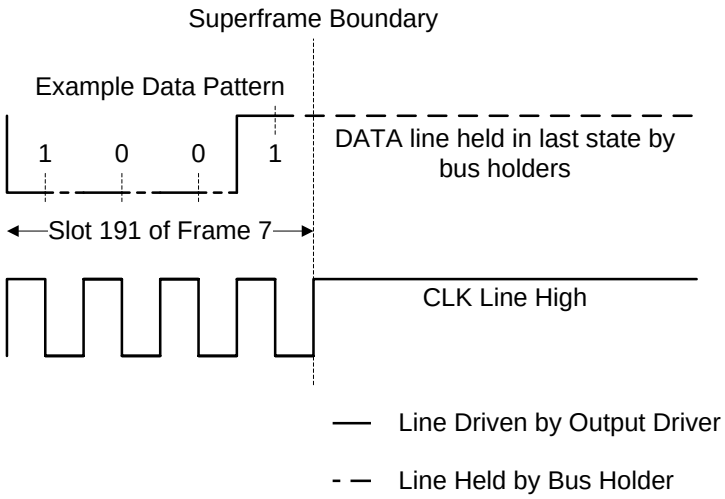


Figure 66 Physical Layer Behavior during a Clock Pause

The active Framer shall pause the CLK after the positive edge of the first clock cycle of the new Superframe, but shall not send the first bit of the Frame Sync symbol, i.e. the active Framer does not toggle the DATA line at the Superframe boundary.

The Framer shall stop the CLK line at the Superframe boundary, after the first low-to-high transition of the first clock cycle of the new Superframe, but before the first high-to-low transition. Note that because of the NRZI coding, the DATA signal can be either high (VIH) or low (VIL) when the CLK is stopped.

The active Manager shall transmit the necessary Messages to suspend any activity on the bus and in a Device prior to activating a Clock Pause. A channel that is not deactivated by the active Manager shall remain active over a Clock Pause.

If a Message is interrupted by a Clock Pause, i.e. only part of the Message has been sent at the Superframe boundary, the source Device shall resume sending the rest of the Message in the first Message Slot after the Framer has resumed toggling the CLK line; a Destination Device shall act upon the Message normally.

A Component, its Devices and their Ports shall resume normal operation after wake-up as though the Clock Pause never happened.

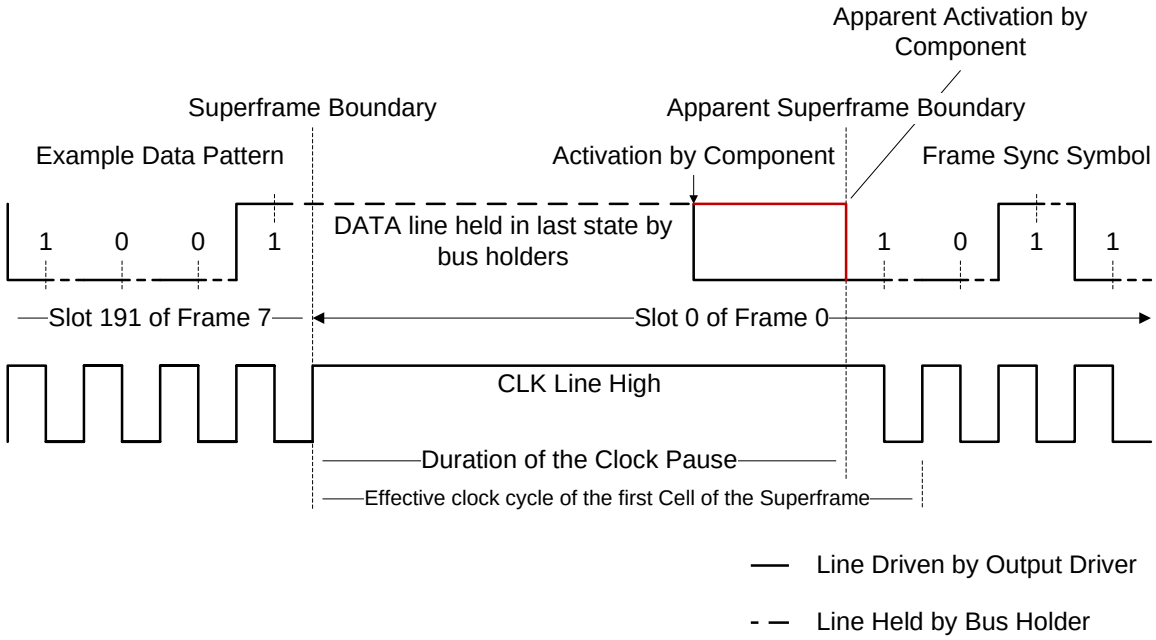


Figure 67 Restart of a Paused Bus Clock

A Device may prompt the Framer to restart the CLK by transmitting a logical one on the DATA line during a Clock Pause. In this case, the Device shall keep driving the DATA line until the negative edge of the CLK signal. After observing a change of state of the DATA line, the Framer shall restart the CLK signal. The timing requirements for the CLK restart are mandated by the active Manager in the payload of the NEXT_PAUSE_CLOCK Message. See Table 66.

At the Clock restart, the CLK line and Frame Structure shall be adapted to reflect any additional announced changes. On the first clock cycle, the active Framer may send the first bit of the Frame Sync symbol even though the DATA line is still driven by the Device that prompted the Framer to restart the CLK.

When the Framer restarts the Clock, the beginning of the first clock cycle is the apparent activation by the Component from the perspective of a device on the bus. This first Cell represents the start of the Apparent Superframe Boundary and can be any phase relation to the Superframe Boundary at the beginning of the Clock Pause.

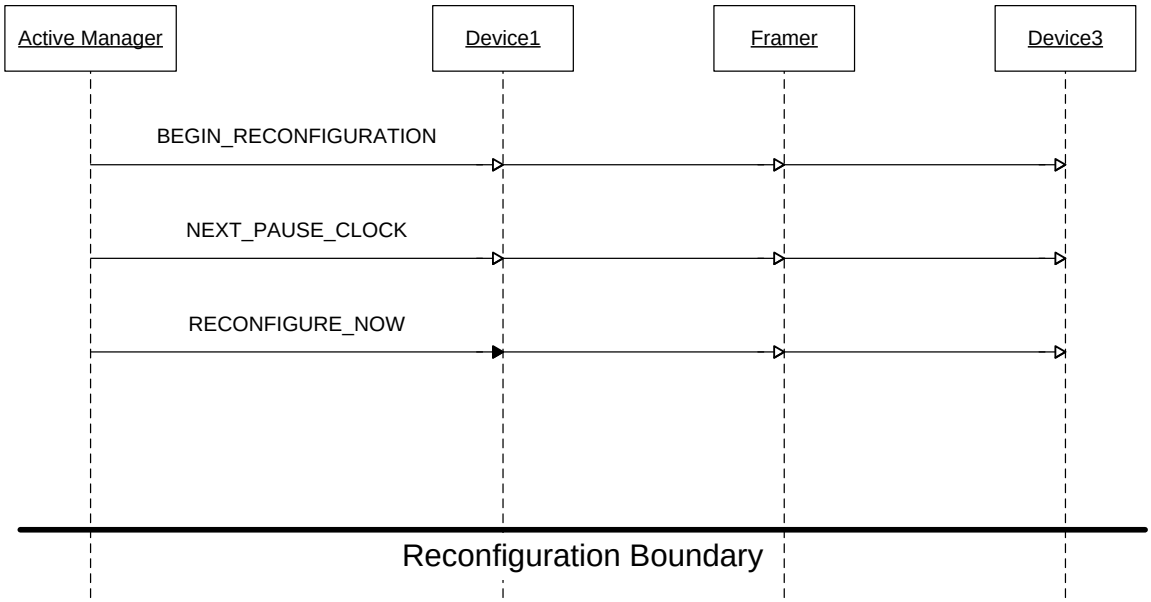


Figure 68 Clock Pause Announcement

10.3.3 Bus Shutdown

The active Manager shall announce shutdown of the bus by broadcasting the NEXT_SHUTDOWN_BUS Message. The active Framer shall respond to this command by moving to the *Undefined* state (Figure 57) at the associated Reconfiguration Boundary. At this point, a Component whose Interface Device supports NEXT_SHUTDOWN_BUS shall move to the *Undefined* state (Figure 59). If the bus reboots after having been in the *Undefined* state, these Components shall go through the complete boot process as described in Section 10.1. A Component can be powered down using mechanisms outside of the scope of this specification. The bus can be restarted only by going through the boot sequence.

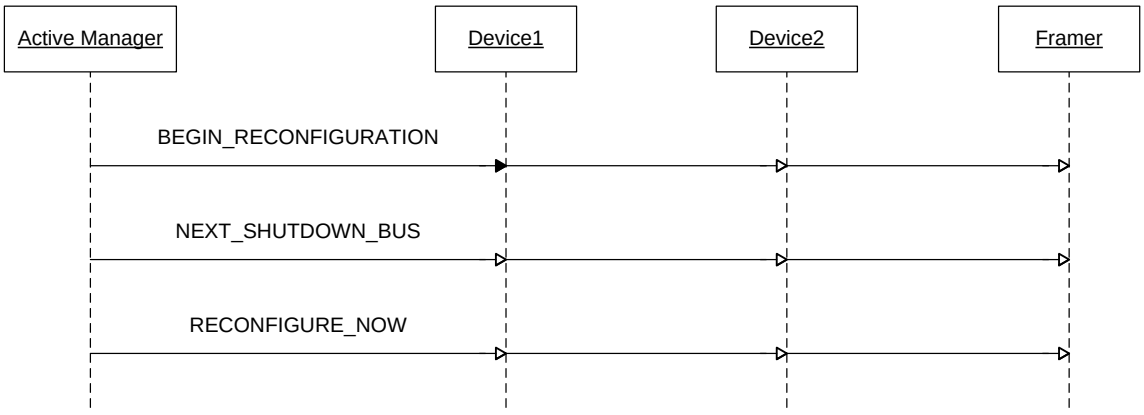


Figure 69 Shutdown Announcement

10.4 Reset Hierarchy

This section defines how the reset of SLIMbus elements, including the entire bus, is accomplished. For clarity, the processes are described in a top-down, hierarchical fashion, beginning with a reset of the entire bus, followed by a Component Reset, a Device Reset and a Port Reset.

10.4.1 Bus Reset

A Bus Reset causes all Components on the bus to undergo a Component Reset (Section 10.4.2). To initiate the Bus Reset, the active Manager sends a Reconfiguration Sequence that contains a NEXT_RESET_BUS Message. If a Component is in its *Operational* state and its Interface Device supports the NEXT_RESET_BUS Message, the Bus Reset shall directly cause the Component to undergo a Component Reset. Otherwise, the Component resets indirectly as described in the following paragraphs.

At the Reconfiguration Boundary that is associated with the NEXT_RESET_BUS announcement, the active Framer shall leave the *Booted* state as shown in Figure 70 (*BusReset* transition) and repeat its boot sequence starting at the *StartingClock* state (see Section 10.1.1.3 and beyond). The active Framer may autonomously change the Clock Gear, Subframe Mode and Root Frequency. If the active Framer does change any of the bus parameters, it shall use legal values as described in Section 6.3.1.

After the Reconfiguration Boundary, the Clock Sourcing Component sends four Frames without any information in the Framing Channel (0x0 in all Framing Slots). This sequence causes a Component not directly reset by the NEXT_BUS_RESET Message to lose Frame synchronization in two Frames, and then causes the Component to perform a Component Reset in four Frames, as described in Section 10.1.2.2 and Section 10.1.2.3.

Note, that if the NEXT_RESET_BUS announcement is combined with other Reconfiguration Messages, the changes to the bus configuration induced by these other announcements are likely to be negated by the subsequent reset sequence. As an example, the effect of a NEXT_ACTIVATE_CHANNEL is cancelled several Frames later by the absence of the Frame Sync symbols thus causing any Devices involved in the particular transport to lose Frame synchronization.

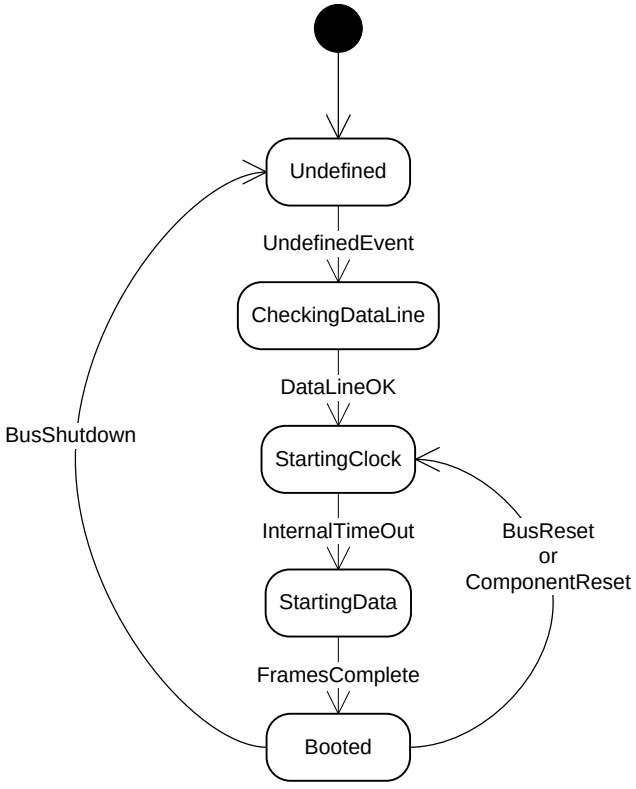


Figure 70 Active Framer Reset State Diagram

10.4.2 Component Reset

A Component Reset causes a Component to repeat part of its synchronization process. Also, all Devices within the Component are reset and release their Logical Addresses. There are two ways to reset a Component:

- Reset of the complete bus as described in Section 10.4.1.
- Reset of a single Component. To perform a Component Reset, the active Manager shall send a RESET_DEVICE Message to the Interface Device of the Component that it wants to reset. A Component in the *Operational* state that receives this Message shall move to the *Reset* state (Figure 59). The active Manager shall not send a RESET_DEVICE Message to the Interface Device of the Clock Sourcing Component because this would cause the active Framer to repeat its boot sequence starting at the StartingClock state. Instead, the active Manager should use a NEXT_RESET_BUS Message to initiate a Bus Reset as described in Section 10.4.1.

A Clock Receiving Component shall perform the Reset without disturbing the bus operation, i.e. without generating glitches on the bus. Following a Component Reset, the Component shall follow the process defined in Section 10.1.2 or Section 10.1.3 to resynchronize to the bus.

10.4.3 Device Reset

A Device Reset shall result in the Device performing a Port Reset for all of its Ports (Section 10.4.4), if any, and performing a reset of all implemented Core Information Elements (Section 7.1.2).

A Device Reset can be initiated in multiple ways:

- as part of a complete Bus Reset,
- as part of a Component Reset that causes a transition from *Operational* state to Reset state as defined in Section 10.1.2.2,
- by receiving the RESET_DEVICE Message,
- by NoDataLineActivity, causing a Component to move to the Reset state.

As the result of a reset, a Device shall change all implemented Core Information Elements to their reset values. If the Device Reset is not part of a Component Reset, the Component shall remain in the *Operational* state and maintain its Logical Address. Any announced changes shall be lost by the Device and it shall act as though it has not received a BEGIN_RECONFIGURATION Message. If the Device contains any Ports, the Device shall reset these Ports, and as a result stop participating in any ongoing transports.

There are no timing requirements for the execution of the RESET_DEVICE Message, although a Device should be designed in such a way that the time it takes for resetting it is minimal. A RESET_DEVICE Message shall not be sent to the Interface Device or the Framer of the Clock Sourcing Component.

A Device Class definition should contain additional Class-specific requirements for the RESET_DEVICE Message and that the Component documentation contains a description of product specific behavior. For instance, guidance should be given on the reset of Class-specific Information Elements.

For the Device Classes defined in this document, special reset behavior is defined in Section 12.

10.4.4 Resetting a Port

Resetting a Port shall result in that Port stopping participation in any ongoing data transfer. The Port shall move to the *Disconnected* state in Figure 82.

A Port Reset can be initiated in the following ways:

- as part of a complete Bus Reset
- as part of a Component or Device Reset
- as a result of DISCONNECT_PORT or NEXT_REMOVE_CHANNEL Message (Section 10.13)
- if the Component loses Superframe or Frame synchronization

10.5 Clock Gear Changes

The active Manager shall announce a Clock Gear change by sending the NEXT_CLOCK_GEAR Message using the broadcast mechanism. The NEXT_CLOCK_GEAR Message is intended to allow a Device to prepare for the Clock Gear change by, for instance, calculating the divider settings for a PLL that will be needed after the Clock Gear change. If a function on a Device is not capable of performing at the new Clock Gear, the active Manager shall disable or reconfigure the function before the new Clock Gear is used on the bus. The Device shall be able to keep synchronization and handle the Message Protocol after the Clock Gear change.

A Component shall configure its Frame Layer according to the pending Clock Gear at the associated Reconfiguration Boundary.

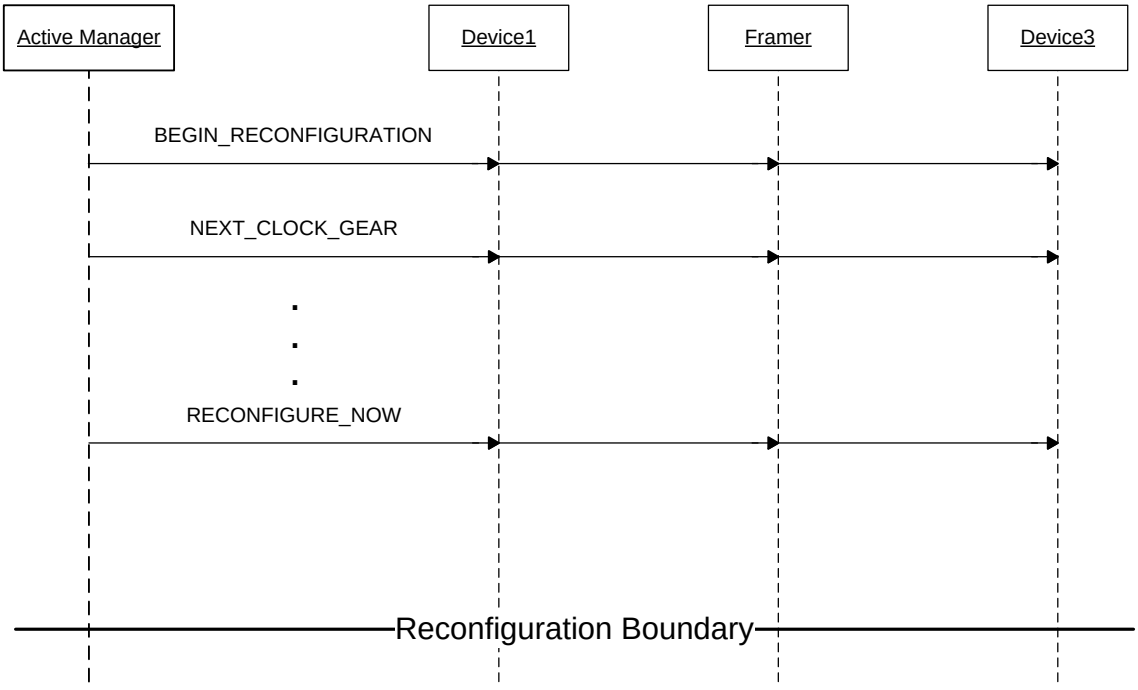


Figure 71 Clock Gear Change Announcement

The active Manager shall broadcast the RECONFIGURE_NOW Message to initiate a Clock Gear change.

10.5.1 Behavior of Channels during a Clock Gear Change

10.5.1.1 Framing Channel

Channel location in the Frame and Superframe for the Framing Channel shall remain constant over a Clock Gear change. As a result, repetition rates of Frames and Superframes changes, as discussed in Section

10.5.1.4. After the Reconfiguration Boundary, the active Framer shall transmit the new Clock Gear value in the Framing Channel as instructed by the active Manager.

10.5.1.2 Message Channel

A NEXT_CLOCK_GEAR Message shall not influence the location and parameters of the Message Channel.

10.5.1.3 Data Channels

In absence of explicit Messages from the active Manager changing the definition of a Data Channel, the Data Channel parameters shall remain constant over a Clock Gear change. While the active Manager may use the default behavior, it should explicitly reprogram all channel definitions before changing the Clock Gear.

Typical Messages sent by the active Manager during a Clock Gear change include NEXT_DEFINE_CHANNEL (Section 11.4.9), NEXT_DEACTIVATE_CHANNEL (Section 11.4.12) and NEXT_REMOVE_CHANNEL (Section 11.4.13).

10.5.1.4 Clock Gear Change Example (informative)

As an example, Figure 72 through Figure 74 show a schematic representation of the SLIMbus using three different Clock Gears. In the example, the Segments repeat at 48 kHz. The Root Frequency of the bus is assumed to be 24.576 MHz, and uses Subframe Mode 0b01001. In Figure 72 the bus is in Gear 6, and there is enough bandwidth available for a single 16-bit wide, 48 kHz sample rate flow. The distance between two Segments is exactly eight Slots. Note, in the figure each numbered box represents one Slot.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95

Figure 72 First 96 Slots of a Superframe in Gear 6

In Figure 73, the active Manager has changed the Clock Gear, such that there is bandwidth available for a second channel. Typically, the active Manager should send a NEXT_DEFINE_CHANNEL Message that increases the repeat distance between two Segments to sixteen to keep the repetition rate (in seconds) constant. As a result, the intermediate Slots (10-13) become available for the second channel. Note also that the Control Space has doubled in bandwidth, as the Subframe Mode is constant. If the active Manager does not send the appropriate NEXT_DEFINE_CHANNEL Message, the channel parameters stay the same over the Reconfiguration Boundary and the channel doubles in bandwidth, i.e. the repetition rate doubles.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191

Figure 73 Full Frame in Gear 7

In Figure 74, the active Manager has increased the Clock Gear again to Gear 8. This has again doubled the number of Slots available for the Control Space. In addition, it has become possible to transport four 16-bit, 48 kHz sample rate flows, assuming that the active Manager adapts the channel parameters again.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287
288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319
320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351
352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383

Figure 74 Two Frames in Gear 8

10.6 Subframe Mode Change

The active Manager shall announce a change of Subframe length by broadcasting the NEXT_SUBFRAME_MODE Message. This Message allows a Device to prepare for the changes in the definition of the Message Channel. The new Subframe Mode is indicated in the payload of the NEXT_SUBFRAME_MODE Message according to Table 15.

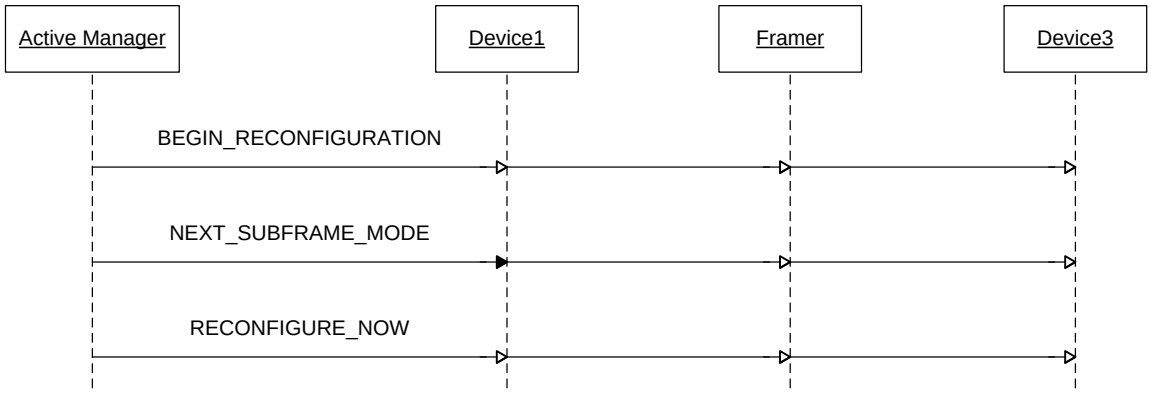


Figure 75 Changing the Message Channel Characteristics

The active Manager shall broadcast a RECONFIGURE_NOW Message to initiate the announced Subframe Mode change.

A Component shall configure its Frame Layer according to the pending Subframe Mode at the associated Reconfiguration Boundary.

10.6.1 Behavior of Channels during a Change of Subframe Length

10.6.1.1 Framing Channel

The change of Subframe Mode has no influence on the location and width of the Framing Channel, but the active Framer shall adapt the content of the Framing Channel to match the new bus parameters.

10.6.1.2 Message Channel

The Subframe Mode determines Message Channel and Guide Channel parameters, as defined in Section 6.3. If the new definition of the Message and Guide Channels overlaps with existing Data Channels, the active Manager shall move or cancel the Data Channels before it broadcasts the RECONFIGURE_NOW Message.

10.6.1.3 Data Channels

The NEXT_SUBFRAME_MODE has no influence on Data Channel definition. However, because of programming errors, or in some other extreme cases, overlap between the Message Channel or Guide Channel and a Data Channel could occur. In this case the Component in which this conflict occurs shall give the Message and Guide Channels priority over the Data Channel. This implies that a Component shall never overwrite the Message Channel or Guide Channel with a Segment, even though the overlap is the result of explicit programming by the active Manager.

10.7 Root Frequency Change

The active Manager shall announce a Root Frequency change by sending the NEXT_ROOT_FREQUENCY Message using the broadcast mechanism. This Message is intended to allow a Device to prepare for the frequency change. If a function on a Device is not capable of performing at the new Root Frequency, the active Manager shall disable or reconfigure the function, before the new Root Frequency is used on the bus. The Device shall be able to keep synchronization and handle the Message Protocol after the Root Frequency change.

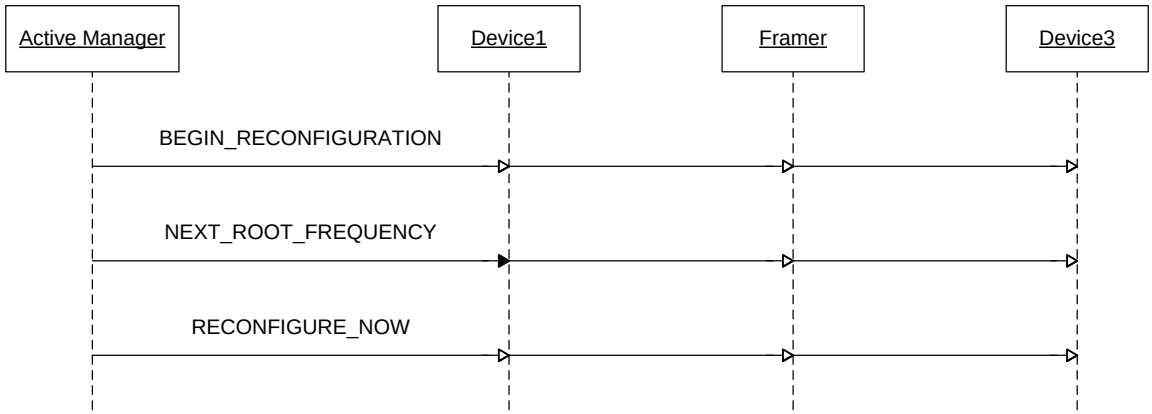


Figure 76 Root Frequency Change Announcement

The active Manager shall broadcast the RECONFIGURE_NOW Message to initiate the Root Frequency change. At the Reconfiguration Boundary, the active Framer shall change the CLK line to match the announced Root Frequency and Clock Gear provided the active Framer supports the requested frequency. Frequencies supported by the active Framer should be specified in the Component's data sheet. An active Framer should set RECONFIG_OBJECTION when an unsupported Root Frequency and Clock Gear combination is requested. The active Manager should verify whether or not the active Framer is capable of producing the announced Root Frequency and Clock Gear combination by requesting the RECONFIG_OBJECTION Information Element specified in Section 7.1.2.4.

A Component shall configure its Frame Layer according to the pending Root Frequency at the associated Reconfiguration Boundary.

10.7.1 Behavior of Channels during a Root Frequency Change

10.7.1.1 Framing Channel

At the associated Reconfiguration Boundary, the active Framer shall change the contents of the Root Frequency (RF) field in the Framing Channel to reflect the new Root Frequency.

10.7.1.2 Message Channel

Unless additional changes are announced, the Control Space width and Subframe length remain constant. Therefore, the Message Channel bandwidth scales with the bus frequency.

10.7.1.3 Data Channels

In absence of explicit Messages from the active Manager changing the definition of the Data Channel, the Segment Distribution and Segment Length remain constant over a Root Frequency change. While the active Manager may use the default behavior, it should explicitly reprogram all channel definitions before changing the Root Frequency.

Typical Messages sent by the active Manager during a Clock Gear change include NEXT_DEFINE_CHANNEL (Section 11.4.9), NEXT_DEACTIVATE_CHANNEL (Section 11.4.12) and NEXT_REMOVE_CHANNEL (Section 11.4.13).

Note that Root Frequency changes can influence the Transport Protocol that is needed to transport data through a channel.

10.8 Framer Handover

The active Manager shall announce Framer handover by broadcasting the NEXT_ACTIVE_FRAMER Message. Unless autonomous behavior is mandated by the Device Class definition, the active Manager shall transmit the necessary Messages to prepare all Devices for the upcoming change.

In this section, the Framer that is leaving the *Active* state is denoted as the outgoing Framer, while the Framer that is intended to become the active Framer is denoted as the incoming Framer.

The active Manager shall determine the values for the Message parameters ***NCo*** and ***NCi*** such that there is no contention on the CLK line during the handover.

NCo is based on the current configured bus clock period, and ***NCi*** is based on the incoming Framer clock period, including any pending Root Frequency or Clock Gear changes.

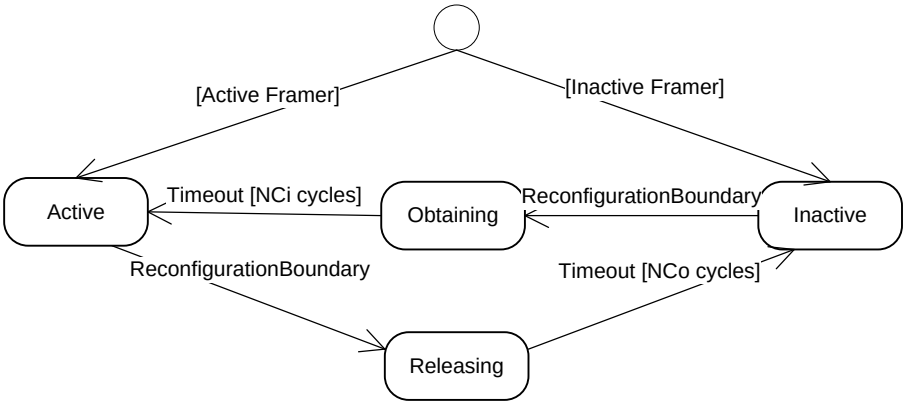


Figure 77 Framer Handover State Diagram

As shown in Figure 77, Framer handover happens in two distinct steps. The Incoming Framer moves from the *Inactive* state to the *Obtaining* state at the Reconfiguration Boundary. The change from the *Obtaining* state to the *Active* state happens after a timeout of ***NCi*** cycles.

Framer Handover also typically implies a change from Clock Sourcing Component to Clock Receiving Component for the Component that holds the outgoing Framer and a change from Clock Receiving Component to Clock Sourcing Component for the incoming Framer. Therefore, these Components change their behavior, such as reset recovery, as specified in Section 10.4.

At boot time, the Framer of the Clock Sourcing Component moves to the *Active* state in Figure 77 and then boots as described in Figure 57. Any other Framers on the bus enter the *Inactive* state in Figure 77. The active Framer needs to be in the *Booted* state (Figure 57) and both Components in the *Operational* state (Figure 59) in order to perform a Framer handover.

In a general case, ***NCo*** and ***NCi*** can be calculated from ***NCo*** = **INT**(4*Fo/Fi)+1 and ***NCi*** = **INT**(4*Fi/Fo). See Annex D for a detailed explanation on how to calculate ***NCo*** and ***NCi***.

10.8.1 Releasing the Active Framer Role

Immediately after the Reconfiguration Boundary, the active Framer shall drive the CLK line HIGH for exactly ***NCo*** cycles. When ***NCo*** cycles have expired, the outgoing Framer shall stop driving the CLK line. When releasing its role, the active Framer shall not drive the DATA line.

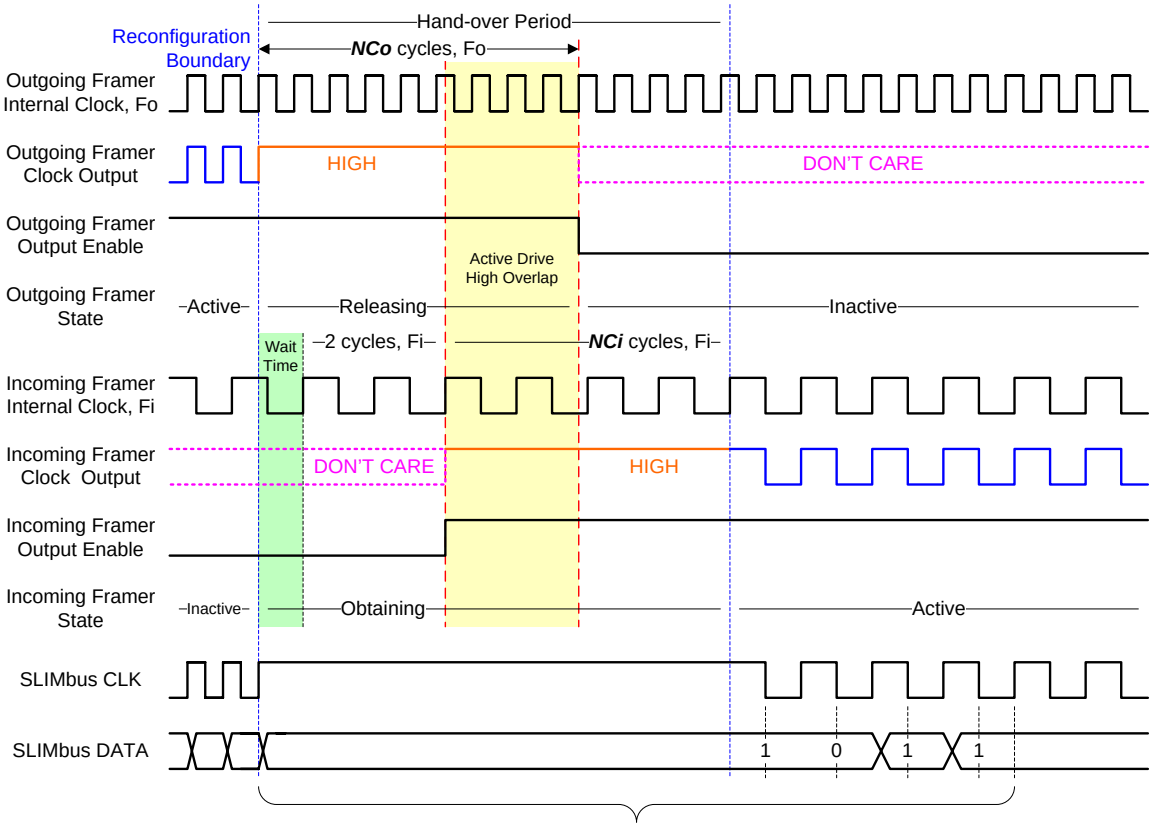


Figure 78 Bus Behavior for Framer Handover

10.8.2 Obtaining the Active Framer Role

Starting at the third positive edge of its internal clock following the Reconfiguration Boundary, the incoming Framer shall drive the CLK line HIGH for ***NCi*** cycles. When ***NCi*** cycles have expired, the incoming Framer shall start clocking the bus (toggling the CLK line). The incoming Framer shall drive the DATA line with the Frame Sync symbol that starts the next Superframe starting at any point from the Reconfiguration Boundary to the end of the Hand-Over period, i.e. when ***NCi*** cycles have expired.

10.8.3 Error Cases

Two error cases are possible:

- The outgoing Framer has lost Message synchronization after the NEXT_ACTIVE_FRAMER Message has been received, or has not received the NEXT_ACTIVE_FRAMER Message. This causes the outgoing Framer to discard all stacked changes, and therefore it will not participate in the handover.
- The incoming Framer has lost Message synchronization after the NEXT_ACTIVE_FRAMER Message has been received, or has not received the NEXT_ACTIVE_FRAMER Message. This causes the outgoing Framer to act normally, but the incoming Framer will not take over. As a result, the bus will not be clocked anymore.

10.8.4 Managing Inactive Framer Behavior (informative)

An inactive Framer implicitly obtains its Clock Gear, Subframe Mode and Root Frequency settings from the Framing Information fields transmitted by the active Framer. If supported, an inactive Framer also obeys NEXT_CLOCK_GEAR, NEXT_SUBFRAME_MODE and NEXT_ROOT_FREQUENCY Reconfiguration Messages transmitted by the active Manager. The active Manager should reconfigure the incoming Framer during Framer handover if the incoming Framer does not support the outgoing Framer's configuration. There might be a difference in phase, or a slight difference in frequency between the bus clock before and after the Framer handover as the clocks might be derived from different oscillators.

10.9 Device Discovery

10.9.1 Behavior of Devices other than the Active Manager

The behavior of a Device during Device discovery is shown in Figure 79. Depending on the state the Device is in, the behavior is different. If a Device is unenumerated, has not yet received a Logical Address and gains Message synchronization, it shall enter the *TryingToAnnounce* state and send a REPORT_PRESENT Message to the active Manager. The payload of the REPORT_PRESENT Message shall contain the Device Class code and the Device Class Version (DCV) of the Device. As soon as the REPORT_PRESENT has been acknowledged, the Device enters the *WaitingForLA* state. In this state, the Device shall wait until it receives an ASSIGN_LOGICAL_ADDRESS or a REQUEST_SELF_ANNOUNCEMENT Message. The ASSIGN_LOGICAL_ADDRESS Message shall cause the Device to change to the *Enumerated* state. The REQUEST_SELF_ANNOUNCEMENT Message shall cause the Device to return to the *TryingToAnnounce* state.

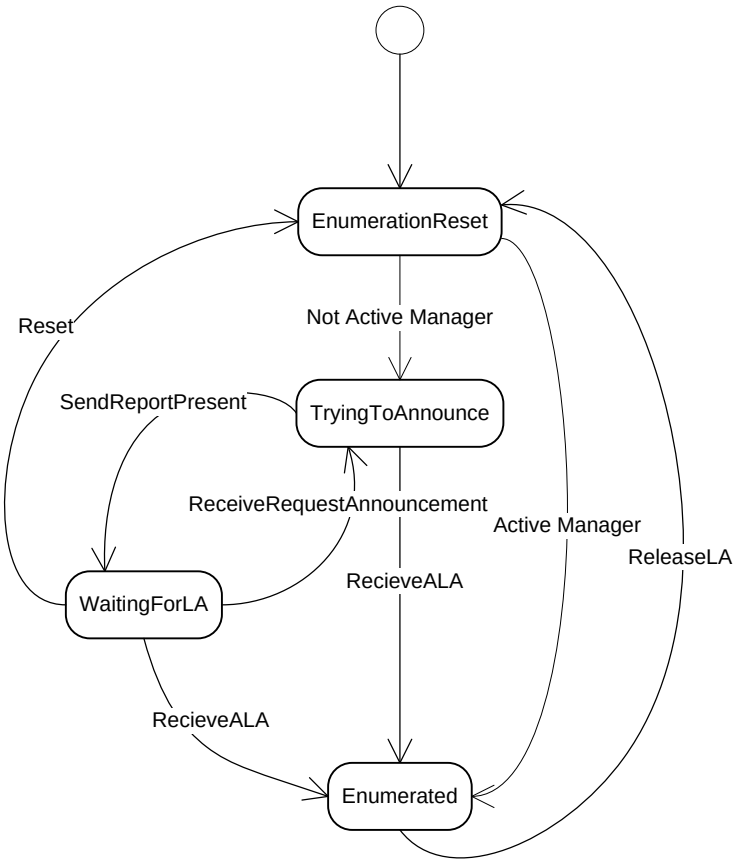


Figure 79 Device Enumeration State Diagram

10.9.1.1 EnumerationReset State

A Device enters the *EnumerationReset* state upon initial boot or if it releases its Logical Address. A Device releases its Logical Address if it detaches from the bus (see Section 10.9.1.5) or if its Component reaches the *Reset* state of Figure 59 (also see Section 10.1.2.2, Section 10.1.2.3 and Section 10.4.2).

A Device leaves the *EnumerationReset* state upon achieving Message synchronization. In the case were the Device does not have a Logical Address, it proceeds to the *TryingToAnnounce* state.

10.9.1.2 TryingToAnnounce State

While in the *TryingToAnnounce* state, a Device shall attempt to send the REPORT_PRESENT Message. The Device shall repeat the REPORT_PRESENT Message until it is positively acknowledged by the active Manager. After receiving the positive acknowledgement from the active Manager, the Device shall move to the *WaitingForLA* state. In addition, the Device shall act upon an ASSIGN_LOGICAL_ADDRESS Message sent to the Enumeration Address of the Device by changing to the *Enumerated* state.

Although a Device in the *TryingToAnnounce* state cannot be the destination of a unicast Message, it shall still observe broadcast Messages and be prepared for a change in the bus configuration, so as not to disrupt normal bus operation while sending the REPORT_PRESENT Message.

10.9.1.3 WaitingForLA State

While in the *WaitingForLA* state, a Device shall not send any Messages, but act only upon receiving an ASSIGN_LOGICAL_ADDRESS Message or a REQUEST_SELF_ANNOUNCEMENT Message.

Although a Device in the *WaitingForLA* state cannot be the destination of a unicast Message, it shall still observe broadcast Messages and be prepared for changes in bus configuration.

10.9.1.4 Enumerated State

A Device in the *Enumerated* state shall ignore all REQUEST_SELF_ANNOUNCEMENT Messages. The Device is now able to be a full participant in the Message Channel and can be assigned to Data Channels in the Data Space.

10.9.1.5 Device Behavior when Detaching from the Bus

Once Enumerated, a Device is expected to remain active until the bus is shutdown, or the Device is powered off or reset. A Device that is capable of detaching from the bus shall transmit the REPORT_ABSENT Message to the active Manager when it will no longer participate in SLIMbus activity. The Device shall continue to transmit the Message until it receives a positive acknowledge. Once the Device receives a positive acknowledge, it shall surrender its Logical Address and shall leave the *Enumerated* state.

If a Device that has left the bus subsequently wishes to re-join, it shall enter the *EnumerationReset* state and shall follow the process described in Section 10.9.1.2. The Device shall not assume that it will regain the same Logical Address as during its previous participation on the bus.

When all Devices within a Component detach from the bus, the Interface Device of that Component should send a REPORT_ABSENT Message. This Message shall be interpreted by the active Manager as though all active Devices in that Component have sent the REPORT_ABSENT Message separately.

If the Device leaves and rejoins the bus without having received an acknowledgement of its REPORT_ABSENT Message, the active Manager will see a second REPORT_PRESENT from the Device without a REPORT_ABSENT in-between. The active Manager may consider this an error if it had no

knowledge of the Device detaching from the bus. This can be handled in the same manner as other error cases that produce the same situation, such as a Device that has suffered temporary power loss.

10.9.2 Active Manager Behavior

Following the completion of the boot process for its Component, the active Manager shall bypass the *TryingToAnnounce* and *WaitingForLA* states shown in Figure 79, shall enter the *Enumerated* state and shall use the Logical Address 0xFF.

10.9.2.1 Static Configuration (informative)

If the bus configuration is determined at design time, ASSIGN_LOGICAL_ADDRESS Messages can be transmitted by the active Manager as soon as it has achieved Message synchronization. Transmission of the ASSIGN_LOGICAL_ADDRESS Message will fail if the destination Device has not yet achieved Message synchronization. The active Manager should wait and retry if this happens. Alternatively, the active Manager can wait until a Device has achieved Message synchronization before trying to assign a Logical Address. In this case, the active Manager waits until it receives a REPORT_PRESENT Message from the Device before sending ASSIGN_LOGICAL_ADDRESS.

10.9.2.2 Other Situations (informative)

If the active Manager does not have complete knowledge of the Devices on the bus, it should wait for the relevant REPORT_PRESENT Messages, before sending the corresponding ASSIGN_LOGICAL_ADDRESS Messages. This can be the case for instance when multiple vendors are supplying similar parts for the same Device.

10.9.2.3 Dealing with Corner Cases

If necessary, the active Manager can initiate a retransmission of the REPORT_PRESENT Message by broadcasting the REQUEST_SELF_ANNOUNCEMENT Message. This causes a Device that has not received a Logical Address to retransmit a REPORT_PRESENT Message, as mandated in Section 10.9.1. If the active Manager needs to cause all Devices to repeat the REPORT_PRESENT Message, it shall first force all Devices to return to the *TryingToAnnounce* state. This can be accomplished, for instance, by performing a Bus Reset as described in Section 10.4.

In some extreme situations, for instance after a power glitch, a Device might resend the REPORT_PRESENT. In this case, the active Manager should detect that the Device has announced itself multiple times and take appropriate action.

10.10 Assigning and Changing a Device Address

As already stated in Section 10.9, the active Manager shall be responsible for the assignment of Logical Addresses. There are two Core Messages that deal with Address configuration, called ASSIGN_LOGICAL_ADDRESS and CHANGE_LOGICAL_ADDRESS.

By design, every Device has an Enumeration Address (see Section 8.2.5.1). Initially, except for the active Manager, a Device does not have a Logical Address, which is used as the Source and Destination of most Messages. The ASSIGN_LOGICAL_ADDRESS Message shall be used by the active Manager to assign a Logical Address to a Device. That Logical Address shall be used by the destination Device during the arbitration sequence of all outgoing Messages. In addition, the ASSIGN_LOGICAL_ADDRESS Message also informs a Device that it shall be the destination Device of all Messages that contain that specific Logical Address in their Header. As described in Section 11.1.2, the ASSIGN_LOGICAL_ADDRESS Message shall contain the Enumeration Address (See Section 8.2.5.1) of the destination Device.

There are no requirements on the timing of the ASSIGN_LOGICAL_ADDRESS Message. Note, that if Message execution is delayed for some reason, the Device remains severely limited in the Messages that it may send (see Section 10.9), and that Messages that have this address as Destination will not be acknowledged.

The active Manager shall ensure that Logical Addresses are unique at bus level so that no two Devices try to use the same Logical Address at the same time.

A Device shall use the assigned Logical Address, until a Component Reset (see Section 10.4) forces the Device to discard the address, the Device detaches from the bus (see Section 10.9.1.5), or until a CHANGE_LOGICAL_ADDRESS Message is received.

A Device shall ignore a second ASSIGN_LOGICAL_ADDRESS Message and shall not use the Logical Address in the payload of that second Message. If implemented, the Device should set the EX_ERROR Information Element, if an ASSIGN_LOGICAL_ADDRESS Message is received while the Device already has a Logical Address.

The CHANGE_LOGICAL_ADDRESS Message shall be used by the active Manager to inform a Device of a change in the Logical Address that it shall use during the Short arbitration sequence of all outgoing Messages. In addition, this Message also informs a Device that it is the destination Device of a Message that contains that new Logical Address in its Header.

There are no requirements on the timing of the CHANGE_LOGICAL_ADDRESS Message, although execution should be timely. Note, that until the execution has finished, Messages that have this address as Destination will not be acknowledged. In addition, Messages that are sent by the destination of the CHANGE_LOGICAL_ADDRESS Message will still use the old Address.

10.11 Changing the Device Arbitration Priority

The active Manager shall use the CHANGE_ARBITRATION_PRIORITY Message to assign a new Arbitration Priority code to a Device. A Device that supports the CHANGE_ARBITRATION_PRIORITY Message may use the assigned Arbitration Priority code for some, or all, of its outgoing Messages. For any Message that does not use the newly assigned Arbitration Priority Code, the Device shall use the initial value defined for the Message in the Device documentation

Note that the active Manager may also reassign Logical Addresses based on the desired Priority to gain additional control over the arbitration behavior on the bus.

10.12 Information Requests and Value Requests

A Device that supports the REQUEST_INFORMATION, REQUEST_CLEAR_INFORMATION, REQUEST_VALUE or REQUEST_CHANGE_VALUE Messages shall adhere to the behavior specified in this section. These four Messages are called REQUEST Messages, while the REPLY_VALUE and REPLY_INFORMATION Messages are called REPLY Messages.

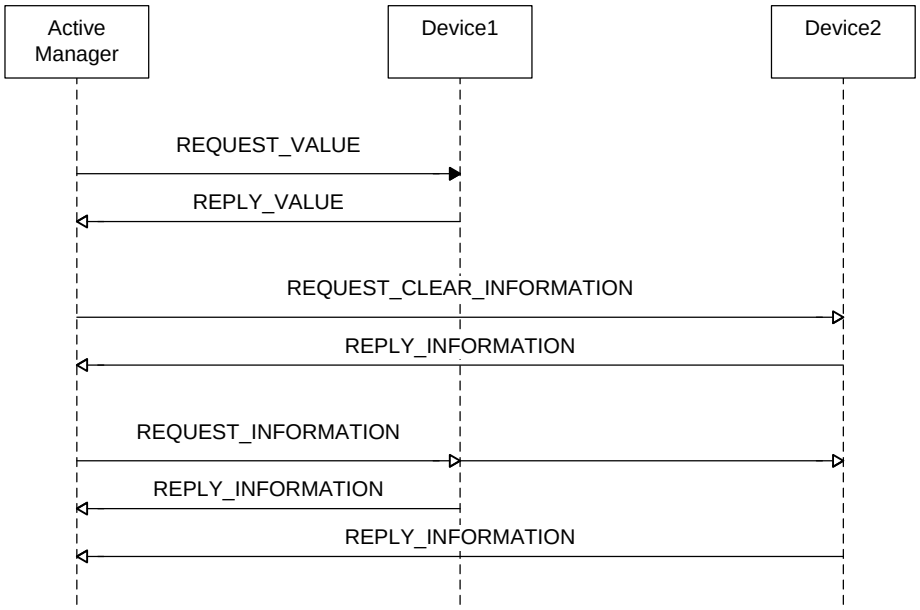


Figure 80 REQUEST-REPLY Example

An example sequence diagram between an active Manager and multiple Devices with multiple REQUEST-REPLY interactions is shown in Figure 80.

10.12.1 Behavior of the Transaction Initiator

The source Device of each REQUEST Message shall generate a Transaction ID (TID) for the Message and then wait for the corresponding REPLY Message with the same Transaction ID. The algorithm to determine the Transaction ID is out of scope for this document. Devices may elect to generate only a single REQUEST Message at a time, and wait for the corresponding REPLY Message before generating a second REQUEST Message. In this case, the TID can always have the same value. Another approach is to increment the TID between consecutive REQUESTs.

Normally, the initiator receives a corresponding REPLY Message. However, in some cases a REPLY Message might not arrive. Therefore, the initiator may expire pending transactions if the request remains pending. This implies that the Transaction ID can be used again and a REPLY Message is not expected anymore by the initiator. For these Messages, the source Device may choose to resend the REQUEST.

If a REPLY Message is received without a corresponding pending Transaction ID, the Message should be discarded by the receiving Device.

10.12.2 Behavior of the Transaction Target

A destination Device of a REQUEST Message shall send a corresponding REPLY Message. The REPLY Message shall contain the same Transaction ID as the REQUEST Message, and shall have the Logical Address of the source of the REQUEST Message as its Destination Address.

If the Element Code of a REQUEST Message is in the range 0xC000 to 0xFFFF or is an elemental access that the target does not support, the target shall abbreviate the REPLY Message such that its payload is just the Transaction ID. Examples of elemental accesses that may be unsupported are requests to an unused region of the Information Map or requests to bit numbers other than the lowest bit number of an Information Element. The target may also do the same as a way of denying data requests, e.g. for security reasons.

A Device shall support at least one open transaction, and may support more than one.

If the REPLY does not receive a positive Acknowledge, the REPLY Message should be retransmitted. REPLY Messages shall not be retransmitted indefinitely.

10.12.3 Atomic Transactions

The REQUEST_CLEAR_INFORMATION and the REQUEST_CHANGE_VALUE Messages request the Target to behave as though it performs three distinct actions: saves the current contents of the indicated Information or Value Slice, clears or updates the indicated Information or Value Slice and replies to the REQUEST Message with the saved contents.

In addition, the Target shall behave as though the first two actions are a single, atomic operation. In other words, the contents of the indicated Information Slice or Value Slice shall not change between saving the contents and clearing or updating the contents.

10.13 Channel Management

The active Manager shall be responsible for initializing a Data Channel and communicating the relevant content parameters to all Devices using that Data Channel.

10.13.1 Initializing a Data Transport

An example of the active Manager constructing a Data Channel is found in Figure 81. Initializing a Data Channel in this way often occurs in distinct phases, although this specification allows for exchanging the order of some of the Messages.

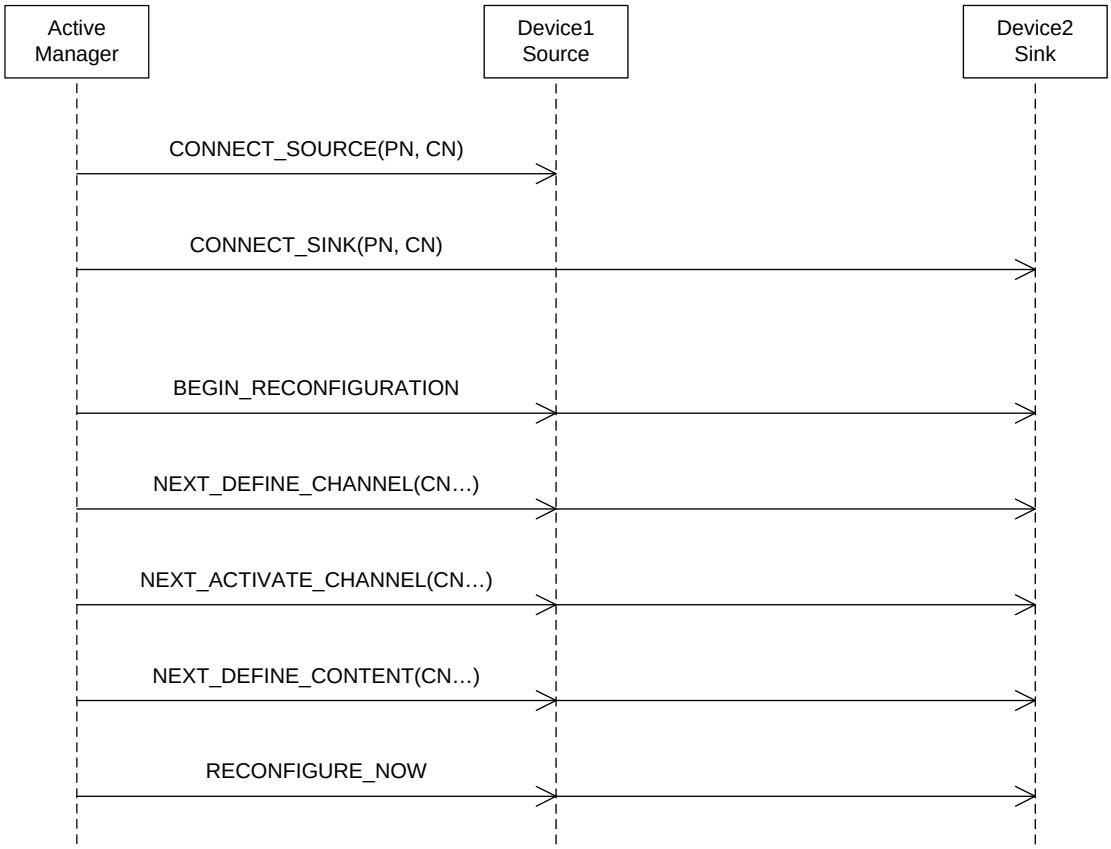


Figure 81 Example Transport Setup

In the example of Figure 81, four distinct steps can be distinguished: Port association, Data Channel definition, content definition, and channel activation.

Ports are associated with channels through the CONNECT_SOURCE or CONNECT_SINK Messages, which indicate to a Device a Port and Channel Number. The channel and content parameters are communicated through the various DEFINE_CONTENT and DEFINE_CHANNEL Messages and the activation is announced by the NEXT_ACTIVATE_CHANNEL Message.

Figure 82 contains a state diagram showing the normative behavior for Ports. Initially, a Port is in the *Disconnected* state, where it shall not transmit or receive any data. This is also the state where the Port resides after it has been reset. As soon as the Port is connected to a Data Channel, see Section 10.13.1.1, it moves to the *Unconfigured* state. As long as a Port is in this state, it shall not transmit or receive any data. In the *Unconfigured* state, the Port shall be able to receive channel configuration Messages for the Channel Number specified by the CONNECT_SOURCE or CONNECT_SINK Message and shall act upon them as specified for the Message. Channel configuration messages are listed in Table 64.

To move to the *Configured* state, a Port needs to receive all necessary configuration parameters. The timing for this transition depends on the Messages that are used. For example, in Figure 81 all the necessary parameters are communicated in the same Reconfiguration Sequence, and therefore the Port shall move to the *Configured* state at the associated Reconfiguration Boundary. If the necessary parameters are communicated as part of multiple Reconfiguration Sequences, the Port shall move to the *Configured* state at the Reconfiguration Boundary that belongs to the sequence that contains the last set of parameters. If a CHANGE_CONTENT Message is used to broadcast some of the necessary parameters, and the remaining parameters are sent through a Reconfiguration Sequence, the Port shall move to the *Configured* state as soon as all parameters have taken effect, either at the associated Reconfiguration Boundary, or at the first

Superframe boundary that occurs at least two Slots after the end of the CHANGE_CONTENT Message, whichever comes last.

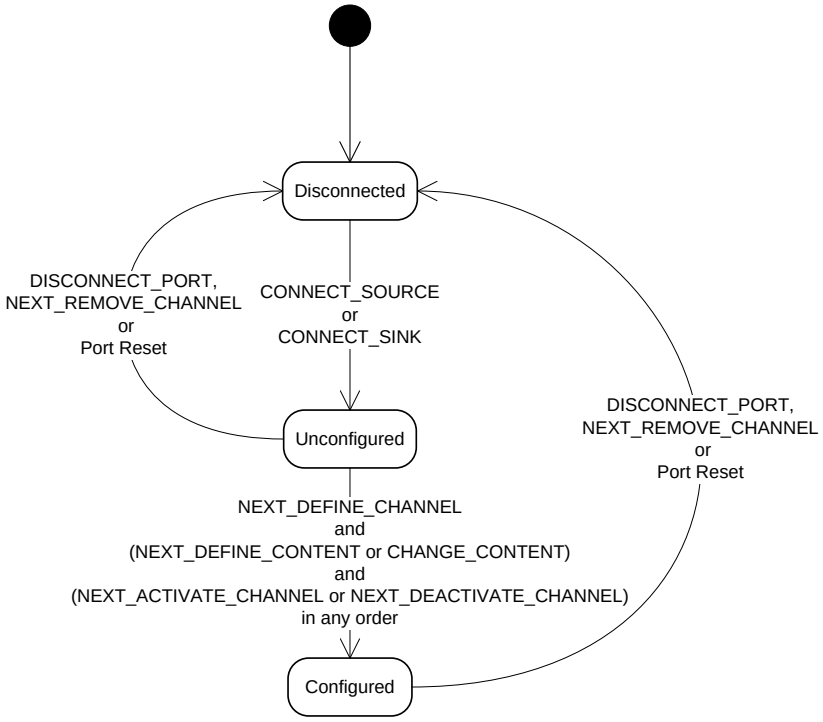


Figure 82 Port State Diagram

10.13.1.1 Connecting Ports

For the Isochronous, Pushed, Pulled and Locked Transport Protocols, the Data Channel shall have exactly one Port associated to it by the CONNECT_SOURCE Message. For the Asynchronous and Extended Asynchronous Transport Protocols, the transport shall have exactly one primary owner associated to it by the CONNECT_SOURCE Message, and one secondary owner associated to it by the CONNECT_SINK Message. Together, the CONNECT_SOURCE and the CONNECT_SINK Message are called Connection Messages.

Note, that these Connection Messages are not Reconfiguration Messages and may be transmitted by their source Device (typically, but not restricted to, the active Manager) outside a Reconfiguration Sequence. The Connection Messages shall cause the destination Device to start capturing all channel configuration Messages (Table 64) that contain the same Channel Number as the Connection Message.

Connection Messages may be transmitted, and retransmitted, at any time. When an already connected Port receives a repeated Connection Message (a Connection Message that does not change the Port connection), the Port shall continue to function as if the repeated Connection Message had not been received.

Note that this specification does not require a Port to check whether Connection Messages change its connection. Consequently, when a connection is to be changed, the Device sending the Connection Message is responsible for managing the state of the Port. This typically involves sending a DISCONNECT_PORT Message before the new Connection Message.

A Port may check whether Connection Messages change its connection, and may take appropriate action if they do make a change, including moving from the *Configured* state to the *Unconfigured* state.

A Port shall capture and act upon the Messages that contain the same Channel Number as the last received Connection Message, regardless of its current state.

Table 64 Channel Configuration Messages

Message Code (MC[6:0])	Description
0x50	NEXT_DEFINE_CHANNEL (CN, TP, SD, SL)
0x51	NEXT_DEFINE_CONTENT (CN, FL, PR, AF, DT, CL, DL)
0x18	CHANGE_CONTENT (CN, FL, PR, AF, DT, CL, DL)
0x54	NEXT_ACTIVATE_CHANNEL (CN)
0x55	NEXT_DEACTIVATE_CHANNEL (CN)
0x58	NEXT_REMOVE_CHANNEL (CN)

10.13.1.2 Defining the Channel and the Content

After the active Manager has notified all the Devices about their involvement in the transport, the Transport Protocol and other parameters of the transport can be sent to the Devices. A Port can be notified of the Channel and Content Parameters as soon as it knows the Channel Number.

10.13.1.2.1 The NEXT_DEFINE_CHANNEL Message

To communicate the channel definition, the active Manager shall broadcast the NEXT_DEFINE_CHANNEL Message as part of a Reconfiguration Sequence. After the associated Reconfiguration Boundary, a Device shall ensure that the Segment Distribution, Transport Protocol, and Segment Length from the payload of this Message are used for the Data Channel. As described in Section 10.3.1.1, the channel definition changes from the NEXT_DEFINE_CHANNEL Message are synchronized with the Subframe Mode changes, to allow for seamless Root Frequency, Clock Gear and Subframe Mode transitions.

The active Manager is responsible for allocating the available SLIMbus bandwidth in such a way that none of the Data Channels overlap with each other or with the Control Space.

The NEXT_DEFINE_CHANNEL Message may be transmitted by the active Manager at any time, implying that it influences the state of a Port in the *Unconfigured* and the *Configured* states. See Figure 82.

A Port based on versions of the SLIMbus specification prior to version 1.01.01 may ignore NEXT_DEFINE_CHANNEL Messages that specify an unsupported Transport Protocol. However, such behavior can lead to problems with Ports in the *Configured* state, if the Transport Protocol changes. A Port based on SLIMbus Specifications version 1.01.01 and later should not transmit or receive in a Data Channel if a NEXT_DEFINE_CHANNEL Message specifies a Transport Protocol that the Port does not support.

A Device should set its EX_ERROR Information Element if it supports such range checking. Refer to Section 11.6 for more details on parameter range checking.

10.13.1.2.2 Defining the Channel Content

The active Manager should notify the Devices involved in a transport of the Segment format that shall be used by the data source or sinks. To do so, the active Manager should broadcast the NEXT_DEFINE_CONTENT Message, or the CHANGE_CONTENT Message. In addition, the CHANGE_CONTENT Message may be used by a Device to set or change the Content Definition of a Channel.

The NEXT_DEFINE_CONTENT Message changes the Content Parameters at the associated Reconfiguration Boundary, just like any of the other Reconfiguration Messages. The CHANGE_CONTENT Message changes the Content Parameters at the first Superframe boundary after the Message. If the Port is active, or becomes active at that moment, the new Content Definition shall be used. If the Port remains inactive, the Content Definition shall be stored for future use.

The NEXT_DEFINE_CONTENT and CHANGE_CONTENT Messages may be sent by the active Manager at any time, implying that it can influence the state of a Port in the *Unconfigured* and the *Configured* state (see Figure 82). If the Port is in the *Configured* state and is active, the Content Definition shall be used by the Port from the appropriate time. Depending on which Content Definition Message has been used the appropriate time is either the corresponding Reconfiguration Boundary or the upcoming Superframe boundary.

10.13.1.2.3 Activating the Channel

The active Manager shall broadcast the NEXT_ACTIVATE_CHANNEL Message as part of a Reconfiguration Sequence to notify the Devices involved in a transport that the Data Channel shall be activated at the associated Reconfiguration Boundary.

The active Manager shall ensure that Device Ports have the necessary information to operate properly. If, for any reason, one or more of the channel definition, content definition and channel activation Messages have not been received, the Port shall remain inoperable, and shall not transmit or receive any information in the Data Channel.

Note that this does not imply that the NEXT_DEFINE_CHANNEL, NEXT_DEFINE_CONTENT or NEXT_ACTIVATE CHANNEL Messages need to be sent during the same Reconfiguration sequence. Typically, the active Manager sends these Messages during the same sequence, when it is constructing the Data Channel. Subsequent changes however, can be made by sending only one of the Messages.

10.13.1.2.4 Full Configuration

As soon as a Port has received the Channel Definition, the Content Definition, and either the NEXT_ACTIVATE_CHANNEL or the NEXT_DEACTIVATE_CHANNEL Message, it shall move to the *Configured* state at the appropriate time, which shall be the moment at which the last change takes effect.

10.13.2 Disconnecting Ports

The DISCONNECT_PORT Message instructs a Port that it shall no longer be involved with a particular Data Channel. Regardless of the Port state, it shall reset and return to the *Disconnected* state. This implies that the Port shall cease to transmit or receive data in the Data Channel. In addition, the Port shall discard any knowledge of the Data Channel Definition and Content and shall discard the Channel Number. These changes shall take effect at the first Superframe boundary that comes at least two Slots after the end of the DISCONNECT_PORT Message.

In addition, a Port Reset causes the Port to move to the *Disconnected* state as described in Section 10.4.4.

10.13.3 Pausing a Data Channel

To temporarily suspend Data Channel operation, the NEXT_DEACTIVATE_CHANNEL shall be broadcast by the active Manager. This Message may be used in both the *Unconfigured* and the *Configured* state. A Port shall not transmit or receive data in a deactivated Data Channel. A Port shall maintain its knowledge of the Data Channel and Content definition, so that a NEXT_ACTIVATE_CHANNEL will resume Data Channel operation.

10.13.4 Cancelling a Data Channel

To release the bandwidth allocated to a Data Channel, the active Manager shall broadcast a NEXT_REMOVE_CHANNEL with the relevant Channel Number during one of the reconfiguration sequences. At the associated Reconfiguration Boundary, all Ports involved in the transport shall be reset and return to the *Disconnected* state (Figure 82).

10.13.5 Changing Content Definitions

In some use cases, the source Device of the transport needs to inform the sink(s) of a change in the channel format. In this case, the source Device should send the CHANGE_CONTENT Message with the new content description in the payload. The announced content parameters shall be used by the source Device from the first Superframe boundary at least two Slots after the transmission of the CHANGE_CONTENT Message.

10.13.6 Moving a Data Channel

When a Data Channel has to be moved in the Data Space to free some bandwidth or to fit in the new available space left after a mode change, the active Manager shall broadcast the NEXT_DEFINE_CHANNEL Message to notify the Devices involved in the transport of the new channel parameters. The Segment length typically remains unchanged. The Segment Distribution field shall be computed accordingly to the expected move.

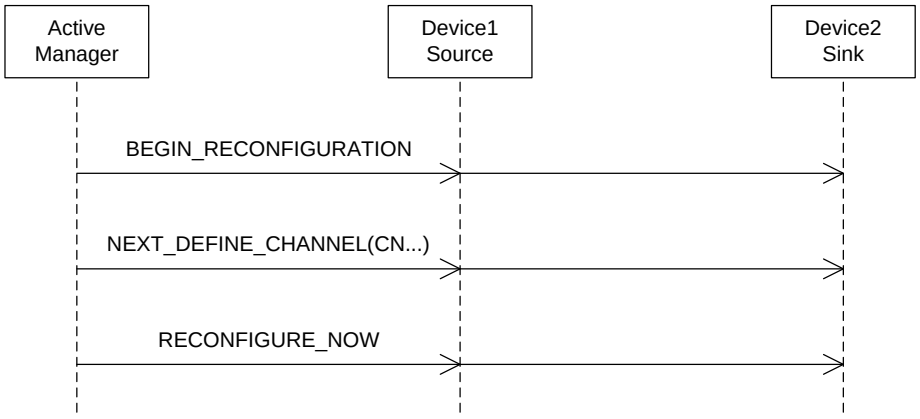


Figure 83 Move Channel Example

11 Core Messages

Messages of the Core Message Type shall be interpreted in the same way by all Devices. They are intended for managing and operating the SLIMbus itself and are interpreted using Table 65.

A Device does not need to support all Core Messages. In fact, a minimal subset is defined in each version of the definition of each Device Class. Refer to the relevant Device Class definition for details.

Table 65 Core Message Codes

Message Code (MC[6:0])	Description
	Device Management Messages
0x01	REPORT_PRESENT (DC, DCV)
0x02	ASSIGN_LOGICAL_ADDRESS (LA)
0x04	RESET_DEVICE ()
0x08	CHANGE_LOGICAL_ADDRESS (LA)
0x09	CHANGE_ARBITRATION_PRIORITY (AP)
0x0C	REQUEST_SELF_ANNOUNCEMENT ()
0x0F	REPORT_ABSENT ()
	Data Channel Management Messages
0x10	CONNECT_SOURCE (PN, CN)
0x11	CONNECT_SINK (PN, CN)
0x14	DISCONNECT_PORT (PN)
0x18	CHANGE_CONTENT (CN, FL, PR, AF, DT, CL, DL)
	Information Management Messages
0x20	REQUEST_INFORMATION (TID, EC)
0x21	REQUEST_CLEAR_INFORMATION (TID, EC, CM)
0x24	REPLY_INFORMATION (TID, IS)
0x28	CLEAR_INFORMATION (EC, CM)
0x29	REPORT_INFORMATION (EC, IS)
	Reconfiguration Messages
0x40	BEGIN_RECONFIGURATION ()
0x44	NEXT_ACTIVE_FRAMER (LAIF, NCo, NCi)
0x45	NEXT_SUBFRAME_MODE (SM)
0x46	NEXT_CLOCK_GEAR (CG)
0x47	NEXT_ROOT_FREQUENCY (RF)
0x4A	NEXT_PAUSE_CLOCK (RT)
0x4B	NEXT_RESET_BUS ()
0x4C	NEXT_SHUTDOWN_BUS ()
0x50	NEXT_DEFINE_CHANNEL (CN, TP, SD, SL)

Message Code (MC[6:0])	Description
0x51	NEXT_DEFINE_CONTENT (CN, FL, PR, AF, DT, CL, DL)
0x54	NEXT_ACTIVATE_CHANNEL (CN)
0x55	NEXT_DEACTIVATE_CHANNEL (CN)
0x58	NEXT_REMOVE_CHANNEL (CN)
0x5F	RECONFIGURE_NOW ()
	Value Management Messages
0x60	REQUEST_VALUE (TID, EC)
0x61	REQUEST_CHANGE_VALUE (TID, EC, VU)
0x64	REPLY_VALUE (TID, VS)
0x68	CHANGE_VALUE (EC, VU)

All unassigned Core Message codes are reserved. A source Device shall not send a Message with a reserved Core Message Code. A destination Device that receives a Message with a reserved Core Message Code shall treat the Message as unsupported. Refer to Section 10.2.5 for details on unsupported Messages.

The following sections define the semantics of each of the Messages in Table 65, including the meaning of the payload and response fields where relevant. A dash (-) in a Cell indicates that the Cell is reserved. Sources shall send zeros in all reserved Cells. Destinations shall not use the contents of reserved Cells, except for Message integrity verification.

11.1 Device Management Messages

Device Management Messages are used for discovery and configuration of Devices on the bus.

11.1.1 REPORT_PRESENT (DC, DCV)

The REPORT_PRESENT Message is sent by an unenumerated Device to announce its presence on the bus.

Message Format

This Message uses a Long Arbitration field and a Short Header field.

Payload

Two bytes holding the Device Class code, DC[7:0], and the Device Class Version code, DCV[7:0], of the Device.

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	DC[7]	DC[6]	DC[5]	DC[4]	DC[3]	DC[2]	DC[1]	DC[0]
Byte 1	DCV[7]	DCV[6]	DCV[5]	DCV[4]	DCV[3]	DCV[2]	DCV[1]	DCV[0]

Description

This Message shall be sent only to the active Manager.

This Message shall be sent by an unenumerated Device after it has gained Message synchronization as described in Section 10.9.1.2.

11.1.2 ASSIGN_LOGICAL_ADDRESS (LA)

The ASSIGN_LOGICAL_ADDRESS Message assigns a Logical Address to a Device.

Message Format

This Message uses a Short Arbitration field and a Long Header field.

Payload

A single byte holding a Logical Address, LA[7:0].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	LA[7]	LA[6]	LA[5]	LA[4]	LA[3]	LA[2]	LA[1]	LA[0]

Description

This Message shall be sent only by the active Manager.

This Message may be sent as part of the Device Discovery and Enumeration as specified in Section 10.9 and Section 10.10.

11.1.3 RESET_DEVICE ()

The RESET_DEVICE Message informs a Device to perform its reset procedure.

Message Format

This Message uses a Short Arbitration field and a Short Header field.

Payload

None

Description

This Message shall be sent only by the active Manager.

This Message informs a Device to perform its reset procedure. General rules for performing a reset are given in Section 10.4. Specific rules for each Device Class are given in the relevant Device Class definition in Section 12.

11.1.4 CHANGE_LOGICAL_ADDRESS (LA)

The CHANGE_LOGICAL_ADDRESS Message changes the value of the Logical Address of the destination Device.

Message Format

This Message uses a Short Arbitration field and a Short Header field.

Payload

A single byte holding the new Logical Address, LA[7:0].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	LA[7]	LA[6]	LA[5]	LA[4]	LA[3]	LA[2]	LA[1]	LA[0]

3864

3865 **Description**

3866 This Message shall be sent only by the active Manager.

3867 This Message may be used by the active Manager to change the Logical Address of a Device as specified in
3868 Section 10.10.

3869 11.1.5 CHANGE_ARBITRATION_PRIORITY (AP)

3870 The CHANGE_ARBITRATION_PRIORITY Message is sent to one or more Devices to change the value
3871 of their Arbitration Priority.

3872 **Message Format**

3873 This Message uses a Short Arbitration field and a Short or Broadcast Header field.

3874 **Payload**

3875 A single byte holding the new Arbitration Priority, AP[2:0].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	—	—	—	—	—	AP[2]	AP[1]	AP[0]

3876 **Description**

3877 This Message shall be sent only by the active Manager.

3878 This Message may be sent by the active Manager to change the Arbitration Priority of a Device as specified
3879 in Section 10.11.

3880 11.1.6 REQUEST_SELF_ANNOUNCEMENT ()

3881 The REQUEST_SELF_ANNOUNCEMENT Message is sent by the active Manager to request an
3882 unenumerated Device retransmit a REPORT_PRESENT Message.

3883 **Message Format**

3884 This Message uses a Short Arbitration field and a Broadcast Header field.

3885 **Payload**

3886 None

3887 **Description**

3888 This Message shall be sent only by the active Manager.

3889 This Message may be sent by the active Manager as part of the Device Discovery as specified in Section
3890 10.9.

11.1.7 REPORT_ABSENT ()

The REPORT_ABSENT Message shall be sent from a Device to announce that it is about to leave the bus.

Message Format

This Message uses a Short Arbitration field and a Short Header field.

Payload

None

Description

This Message shall be sent only to the active Manager.

A Device shall send this Message when it intends to leave the bus as described in Section 10.9.1.5.

11.2 Data Channel Management Messages

Data Channel Management Messages are used to communicate information to Devices about Channels in the Data Space.

11.2.1 CONNECT_SOURCE (PN, CN)

The CONNECT_SOURCE Message informs the Device to connect the specified Port, PN, to the specified Data Channel, CN. The Port shall act as the data source for the Data Channel.

Message Format

This Message uses a Short Arbitration field and a Short Header field.

Payload

Two bytes holding the Port Number, PN[5:0], and the Data Channel Number, CN[7:0].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	—	—	PN[5]	PN[4]	PN[3]	PN[2]	PN[1]	PN[0]
Byte 1	CN[7]	CN[6]	CN[5]	CN[4]	CN[3]	CN[2]	CN[1]	CN[0]

Description

This Message informs a Device to connect its Port PN to channel CN. Refer to Section 10.13.1.1 for further information.

11.2.2 CONNECT_SINK (PN, CN)

The CONNECT_SINK Message informs the Device to connect the specified Port, PN, to the specified Data Channel, CN. The Port shall act as a data sink for the Data Channel.

Message Format

This Message uses a Short Arbitration field and a Short Header field.

Payload

Two bytes holding the Port Number, PN[5:0], and the Data Channel Number, CN[7:0].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	—	—	PN[5]	PN[4]	PN[3]	PN[2]	PN[1]	PN[0]
Byte 1	CN[7]	CN[6]	CN[5]	CN[4]	CN[3]	CN[2]	CN[1]	CN[0]

Description

This Message informs a Device to connect its Port PN to channel CN. Refer to Section 10.13.1.1 for further information.

11.2.3 DISCONNECT_PORT (PN)

The DISCONNECT_PORT Message informs the Device to disconnect the Port specified by PN.

Message Format

This Message uses a Short Arbitration field and a Short Header field.

Payload

A single byte holding the Port Number, PN[5:0].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	—	—	PN[5]	PN[4]	PN[3]	PN[2]	PN[1]	PN[0]

Description

This Message informs a Device to disconnect its Port PN from the previously connected channel. See Section 10.13.2 for more information.

11.2.4 CHANGE_CONTENT (CN, FL, PR, AF, DT, CL, DL)

The CHANGE_CONTENT Message broadcasts detailed information about the structure of the Data Channel contents.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

Four bytes holding a Data Channel Number, CN[7:0], for use as an identifier, followed by a list of parameters that define the Data Channel content:

- Frequency Locked bit, FL
- Presence Rate, PR[6:0]
- Auxiliary Bit Format AF[3:0]
- Data Type, DT[3:0]
- Channel Link bit, CL

- DATA Length, DL[4:0]

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	CN[7]	CN[6]	CN[5]	CN[4]	CN[3]	CN[2]	CN[1]	CN[0]
Byte 1	FL	PR[6]	PR[5]	PR[4]	PR[3]	PR[2]	PR[1]	PR[0]
Byte 2	AF[3]	AF[2]	AF[1]	AF[0]	DT[3]	DT[2]	DT[1]	DT[0]
Byte 3	—	—	CL	DL[4]	DL[3]	DL[2]	DL[1]	DL[0]

Description

This Message informs a Device of the structure of the Data Channel content. Refer to Section 10.13.5 for more information.

The FL bit indicates whether the content flow is known to be frequency locked to the SLIMbus CLK signal. See Section 9.2.

The PR field describes the Presence Rate. See Section 9.6.

Auxiliary bit formats and data types are defined in Section 9.4.

The CL bit, when 1, indicates that the content of channel CN is related to that of channel CN-1. When the CL bit is 0, the contents of channel CN is not necessarily related to the contents of channel CN-1.

The DL field indicates the length of the DATA field, in Slots, as specified in Section 9.7.

11.3 Information Management Messages

Information Management Messages are used to communicate and manage Information Elements. A Device may send information to another Device in reply to a REQUEST Message or autonomously in the form of a REPORT Message. In addition, a Device may be requested to selectively change Information Elements to their reset values.

In all cases, Information Elements are addressed through an Information Slice, which may contain one or more Information Elements. See Section 7.1 for more on Information Slices.

11.3.1 REQUEST_INFORMATION (TID, EC)

The REQUEST_INFORMATION Message instructs a Device to send the indicated Information Slice.

Message Format

This Message uses a Short Arbitration field and a Short or Broadcast Header field.

Payload

Three bytes holding an 8-bit Transaction ID, TID[7:0], followed by a 16-bit Element Code, EC[15:0], indicating the Information Slice.

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	TID[7]	TID[6]	TID[5]	TID[4]	TID[3]	TID[2]	TID[1]	TID[0]
Byte 1	EC[7]	EC[6]	EC[5]	EC[4]	EC[3]	EC[2]	EC[1]	EC[0]
Byte 2	EC[15]	EC[14]	EC[13]	EC[12]	EC[11]	EC[10]	EC[9]	EC[8]

Description

This Message instructs the destination Device to return the specified Information Slice. Refer to Section 10.12.1 for more information on REQUEST-REPLY transactions and to Section 7.1 for the definition of Element Codes.

11.3.2 REQUEST_CLEAR_INFORMATION (TID, EC, CM)

The REQUEST_CLEAR_INFORMATION Message instructs a Device to send the indicated Information Slice and to clear all, or parts, of that Information Slice.

Message Format

This Message uses a Short Arbitration field and a Short or Broadcast Header field.

Payload

Three to nineteen bytes holding an 8-bit Transaction ID, TID[7:0]; a 16-bit Element Code, EC[15:0], indicating the Information Slice; followed by zero to sixteen bytes of Clear Mask, CM[].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	TID[7]	TID[6]	TID[5]	TID[4]	TID[3]	TID[2]	TID[1]	TID[0]
Byte 1	EC[7]	EC[6]	EC[5]	EC[4]	EC[3]	EC[2]	EC[1]	EC[0]
Byte 2	EC[15]	EC[14]	EC[13]	EC[12]	EC[11]	EC[10]	EC[9]	EC[8]
Byte 3	CM[7]	CM[6]	CM[5]	CM[4]	CM[3]	CM[2]	CM[1]	CM[0]
. . .								
Byte N+2	CM[8N-1]	CM[8N-2]	CM[8N-3]	CM[8N-4]	CM[8N-5]	CM[8N-6]	CM[8N-7]	CM[8N-8]

Description

This Message asks the destination Device to return the identified Information Slice, and to atomically change the value of the Information Slice according to the CM field (see Section 10.12.3). The CM field shall be placed in the LSBs of the relevant payload byte(s).

A Device shall change to their reset values, those bits of the identified Information Slice for which the corresponding Clear Mask bit is 1.

If the size of Clear Mask is smaller than the size of the Information Slice, the Device shall pad the MSBs of Clear Mask with ones. For example, if the size of Clear Mask is zero bytes, the Device shall change all bits of the Information Slice to their reset values. If the size of Clear Mask is larger than the size of the Information Slice, the Device shall change only the portion of the Information Map corresponding to the identified Information Slice.

Refer to Section 10.12 for more information on REQUEST-REPLY transactions and to Section 7.1 for the definition of Element Codes.

11.3.3 REPLY_INFORMATION (TID, IS)

The REPLY_INFORMATION Message is sent by a Device in response to a REQUEST_INFORMATION or REQUEST_CLEAR_INFORMATION Message.

Message Format

This Message uses a Short Arbitration field and a Short Header field.

Payload

One to seventeen bytes holding an 8-bit Transaction ID, TID[7:0], optionally followed by the requested Information Slice, IS[] in one to sixteen bytes.

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	TID[7]	TID[6]	TID[5]	TID[4]	TID[3]	TID[2]	TID[1]	TID[0]
Byte 1	IS[7]	IS[6]	IS[5]	IS[4]	IS[3]	IS[2]	IS[1]	IS[0]
					.			
					.			
					.			
Byte N	IS[8N-1]	IS[8N-2]	IS[8N-3]	IS[8N-4]	IS[8N-5]	IS[8N-6]	IS[8N-7]	IS[8N-8]

Description

The Destination Address field shall be the Logical Address of the Device that sent the REQUEST_INFORMATION or REQUEST_CLEAR_INFORMATION Message.

The Transaction ID TID[7:0] shall be the Transaction ID of the corresponding REQUEST_INFORMATION or REQUEST_CLEAR_INFORMATION Message

As specified in Section 10.12.2, the payload may consist of only the Transaction ID. In all other cases the Payload shall contain the Information Slice that was identified in the corresponding REQUEST Message. The Information Slice shall be placed in the LSBs of the relevant payload byte(s). Any unused bits shall be filled with zeros. Refer to Section 10.12 for more information on REQUEST-REPLY transactions and Section 7.1for the definition of Element Codes.

11.3.4 CLEAR_INFORMATION (EC, CM)

The CLEAR_INFORMATION Message instructs a Device to selectively clear all, or parts, of the Information Slice.

Message Format

This Message uses a Short Arbitration field and a Short or Broadcast Header field.

Payload

Two to eighteen bytes holding a 16-bit Element Code, EC[15:0], indicating the Information Slice, followed by zero to sixteen bytes of Clear Mask, CM[].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	EC[7]	EC[6]	EC[5]	EC[4]	EC[3]	EC[2]	EC[1]	EC[0]
Byte 1	EC[15]	EC[14]	EC[13]	EC[12]	EC[11]	EC[10]	EC[9]	EC[8]
Byte 2	CM[7]	CM[6]	CM[5]	CM[4]	CM[3]	CM[2]	CM[1]	CM[0]

	D7	D6	D5	D4	D3	D2	D1	D0
					.			
					.			
					.			
Byte N+1	CM[8N-1]	CM[8N-2]	CM[8N-3]	CM[8N-4]	CM[8N-5]	CM[8N-6]	CM[8N-7]	CM[8N-8]

Description

The CM field shall be placed in the LSBs of the relevant payload byte(s).

A Device shall change to their reset values, those bits of the identified Information Slice for which the corresponding Clear Mask bit is 1.

If the size of Clear Mask is smaller than the size of the Information Slice, the Device shall pad the MSBs of Clear Mask with ones. For example, if the size of Clear Mask is zero bytes, the Device shall change all bits of the Information Slice to their reset values. If the size of Clear Mask is larger than the size of the Information Slice, the Device shall change only the portion of the Information Map corresponding to the identified Information Slice.

Refer to Section 7.1 for the definition of Element Codes.

11.3.5 REPORT_INFORMATION (EC, IS)

The REPORT_INFORMATION Message is used by a Device to inform another Device about a change in an Information Slice.

Message Format

This Message uses a Short Arbitration field and a Short or Broadcast Header field.

Payload

Three to eighteen bytes holding a 16-bit Element Code, EC[15:0], followed by one to sixteen bytes of the corresponding Information Slice, IS[].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	EC[7]	EC[6]	EC[5]	EC[4]	EC[3]	EC[2]	EC[1]	EC[0]
Byte 1	EC[15]	EC[14]	EC[13]	EC[12]	EC[11]	EC[10]	EC[9]	EC[8]
Byte 2	IS[7]	IS[6]	IS[5]	IS[4]	IS[3]	IS[2]	IS[1]	IS[0]
					.			
					.			
					.			
Byte N+1	IS[8N-1]	IS[8N-2]	IS[8N-3]	IS[8N-4]	IS[8N-5]	IS[8N-6]	IS[8N-7]	IS[8N-8]

Description

This Message is normally, though not exclusively, used to indicate a change in an internal Information Slice. The Information Slice shall be placed in the LSBs of the relevant payload byte(s). Any unused bits shall be filled with zeros. Refer to Section 7.1.3, Section 12.2.6, Section 12.3.6 and Section 12.4.6 for more information on reporting Information Elements.

11.4 Reconfiguration Messages

Reconfiguration Messages are used for tasks whose effect must be synchronized between all Devices.

11.4.1 BEGIN_RECONFIGURATION ()

The BEGIN_RECONFIGURATION Message informs all Devices of the start of a Reconfiguration Sequence.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

None

Description

This Message shall be sent only by the active Manager.

This Message is sent by the active Manager to indicate the start, or restart, of a Reconfiguration Sequence. See Section 10.3.1.

11.4.2 NEXT_ACTIVE_FRAMER (LAIF, NCo, NCi)

The NEXT_ACTIVE_FRAMER Message informs a Device that the active Framer is going to hand over the role of Framer to a specified inactive Framer with the Logical Address LAIF. See Section 10.8.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

Four bytes holding an 8-bit Logical Address of the Incoming Framer, LAIF[7:0], followed by two packed bytes holding the Number of Outgoing Framer Clock cycles, NCo[11:0], and the Number of Incoming Framer Clock cycles, NCi[11:0].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	LAIF[7]	LAIF[6]	LAIF[5]	LAIF[4]	LAIF[3]	LAIF[2]	LAIF[1]	LAIF[0]
Byte 1	NCo[7]	NCo[6]	NCo[5]	NCo[4]	NCo[3]	NCo[2]	NCo[1]	NCo[0]
Byte 2	NCi[3]	NCi[2]	NCi[1]	NCi[0]	NCo[11]	NCo[10]	NCo[9]	NCo[8]
Byte 3	NCi[11]	NCi[10]	NCi[9]	NCi[8]	NCi[7]	NCi[6]	NCi[5]	NCi[4]

Description

This Message shall only be sent by the active Manager. The Manager shall not send a NEXT_ACTIVE_FRAMER and a NEXT_PAUSE_CLOCK Message in the same Reconfiguration sequence.

This Message informs a destination Device that the specified inactive Framer shall become the active Framer and the source Device shall become an inactive Framer at the associated Reconfiguration Boundary. See Section 10.8 for more information.

The Message parameters ***NCo*** and ***NCi*** are the number of internal clock cycles that the active and inactive Framers, respectively, shall hold the CLK line HIGH.

11.4.3 NEXT_SUBFRAME_MODE (SM)

The NEXT_SUBFRAME_MODE Message informs a Device that the active Manager intends to change the Subframe Mode.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

A single byte holding the new Subframe Mode, SM[4:0].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	—	—	—	SM[4]	SM[3]	SM[2]	SM[1]	SM[0]

Description

This Message shall be sent only by the active Manager.

This Message informs a Device that the active Manager intends to change the Subframe Mode to the value in the Payload field at the associated Reconfiguration Boundary. The Subframe Mode shall be encoded using the values in Table 15. Reserved values shall not be used.

11.4.4 NEXT_CLOCK_GEAR (CG)

Message from the active Manager to inform all destinations that it intends to change the Clock Gear.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

A single byte holding the new Clock Gear, CG[3:0].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	—	—	—	—	CG[3]	CG[2]	CG[1]	CG[0]

Description

This Message shall be sent only by the active Manager.

This Message informs a Device that the active Manager intends to change the Clock Gear to the value in the Payload field at the associated Reconfiguration Boundary. The Clock Gear shall be encoded using the values in Table 16. Reserved values shall not be used. See Section 10.6 for more information.

11.4.5 NEXT_ROOT_FREQUENCY (RF)

The NEXT_ROOT_FREQUENCY Message informs a Device that the active Manager intends to change the Root Frequency.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

A single byte holding the new Root Frequency, RF[3:0]

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	—	—	—	—	RF[3]	RF[2]	RF[1]	RF[0]

Description

This Message shall be sent only by the active Manager.

This Message informs a Device that the active Manager intends to change the Root Frequency to the value in the Payload field at the associated Reconfiguration Boundary. The Root Frequency shall be encoded using the values in Table 17. Reserved values shall not be used. See Section 10.7 for more information.

This Message also affects the behavior of Components that use the Phasing Signal. See Section 6.3.1.8.

11.4.6 NEXT_PAUSE_CLOCK (RT)

The NEXT_PAUSE_CLOCK Message informs a Device that the active Manager intends to have the active Framer pause the bus.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

A single byte holding the new Restart Time, RT[7:0].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	RT[7]	RT[6]	RT[5]	RT[4]	RT[3]	RT[2]	RT[1]	RT[0]

Description

This Message shall be sent only by the active Manager. The Manager shall not send a NEXT_ACTIVE_FRAMER and a NEXT_PAUSE_CLOCK Message in the same Reconfiguration sequence.

This Message informs a Device that the active Manager intends to have the active Framer pause the bus at the associated Superframe boundary. The Restart Time held in the Payload field shall be encoded using the values in Table 66. Reserved values shall not be used.

See Section 10.3.2 for details of SLIMbus Clock Pause.

Table 66 NEXT_PAUSE_CLOCK Payload Value

RT[7:0]	Description	Requirement
0x00	Fast Recovery. After a restart request, the active Framer shall resume toggling the CLK line within four cycles of the CLK line frequency (as indicated by the Clock Gear and Root Frequency) used for the upcoming Frame.	Optional
0x01	Constant Phase Recovery. After the restart request, the active Framer shall resume toggling the CLK line so the duration of the Pause is an integer number of Superframes in the upcoming Clock Gear.	Optional
0x02	Unspecified Delay. After a restart request, the active Framer shall resume toggling the CLK line after an unspecified delay. No requirement exists for the time the active Framer takes to restart the clock signal. This allows for system designers to stop all clock generation circuitry to save power. The Framer documentation shall indicate the restart delay period.	Mandatory
0x03 to 0xFF	Reserved.	N/A

11.4.7 NEXT_RESET_BUS ()

The NEXT_RESET_BUS Message informs a Device that the active Manager intends to have the active Framer reset the bus.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

None

Description

This Message shall be sent only by the active Manager.

This Message informs a Device that the active Manager intends to have the active Framer reset the bus at the associated Reconfiguration Boundary as described in Section 10.4.1.

11.4.8 NEXT_SHUTDOWN_BUS ()

The NEXT_SHUTDOWN_BUS Message informs a Device that the active Manager intends to shutdown the bus.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

None

Description

This Message shall be sent only by the active Manager.

This Message informs a Device that the active Manager intends to shutdown the bus at the associated Reconfiguration Boundary. See Section 10.3.3.

11.4.9 NEXT_DEFINE_CHANNEL (CN, TP, SD, SL)

The NEXT_DEFINE_CHANNEL Message informs a Device that the active Manager intends to define the parameters of a Data Channel.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

Four bytes holding a Data Channel Number, CN[7:0], for use as an identifier, followed by a list of parameters that define the properties of the Data Channel:

- Transport Protocol, TP[3:0]
- Segment Distribution, SD[11:0]
- Segment Length, in Slots, SL[4:0]

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	CN[7]	CN[6]	CN[5]	CN[4]	CN[3]	CN[2]	CN[1]	CN[0]
Byte 1	SD[7]	SD[6]	SD[5]	SD[4]	SD[3]	SD[2]	SD[1]	SD[0]
Byte 2	TP[3]	TP[2]	TP[1]	TP[0]	SD[11]	SD[10]	SD[9]	SD[8]
Byte 3	0	0	0	SL[4]	SL[3]	SL[2]	SL[1]	SL[0]

Description

This Message shall be sent only by the active Manager.

This Message informs a Device of the structure of a Data Channel. If the Data Channel, CN, is already defined, its attributes shall be modified according to the parameters in the Payload field. If the Data Channel is not defined, it shall be created.

Note that this Message does not affect whether a Data Channel is activated or not. Refer to Section 10.13 for details on Data Channel management.

This Message takes effect at the associated Reconfiguration Boundary. See Section 10.3.1 for more information.

11.4.10 NEXT_DEFINE_CONTENT (CN, FL, PR, AF, DT, CL, DL)

The NEXT_DEFINE_CONTENT Message informs a Device how the Data Channel CN shall be used.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

Four bytes holding a Data Channel Number, CN[7:0], for use as an identifier, followed by a list of parameters that define the Data Channel content:

- Frequency Locked bit, FL
- Presence Rate, PR[6:0]
- Auxiliary Bit Format, AF[3:0]
- Data Type, DT[3:0]
- Channel Link bit, CL
- DATA Length, DL[4:0]

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	CN[7]	CN[6]	CN[5]	CN[4]	CN[3]	CN[2]	CN[1]	CN[0]
Byte 1	FL	PR[6]	PR[5]	PR[4]	PR[3]	PR[2]	PR[1]	PR[0]
Byte 2	AF[3]	AF[2]	AF[1]	AF[0]	DT[3]	DT[2]	DT[1]	DT[0]
Byte 3	0	0	CL	DL[4]	DL[3]	DL[2]	DL[1]	DL[0]

Description

This Message shall be sent only by the active Manager.

This Message informs a Device of the structure of the Data Channel content. The new Data Channel content shall take effect at the associated Reconfiguration Boundary following the Message reception. Refer to 10.13.5 for more information.

The FL bit indicates whether the content flow is known to be frequency locked to the SLIMbus CLK signal. See Section 9.2.

The PR field describes the Presence Rate. See Section 9.6.

Auxiliary bit formats and data types are defined in Section 9.4.

The CL bit, when 1, indicates that the content of channel CN is related to that of channel CN-1. When the CL bit is 0, the contents of channel CN is not necessarily related to the contents of channel CN-1.

The DL field indicates the length of the DATA field in Slots as specified in Section 9.7.

11.4.11 NEXT_ACTIVATE_CHANNEL (CN)

The NEXT_ACTIVATE_CHANNEL Message is used to switch on the specified Data Channel.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

A single byte holding the Channel Number, CN[7:0], of the Data Channel that is being activated.

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	CN[7]	CN[6]	CN[5]	CN[4]	CN[3]	CN[2]	CN[1]	CN[0]

Description

This Message shall be sent only by the active Manager.

This Message is sent by the active Manager as part of the process of initializing a data transport as specified in Section 10.13.1.

11.4.12 NEXT_DEACTIVATE_CHANNEL (CN)

The NEXT_DEACTIVATE_CHANNEL Message is used to switch off a specified Data Channel.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

A single byte holding the Channel Number, CN[7:0], of the Data Channel that is being deactivated.

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	CN[7]	CN[6]	CN[5]	CN[4]	CN[3]	CN[2]	CN[1]	CN[0]

Description

This Message shall be sent only by the active Manager.

This Message is sent by the active Manager as part of the process of suspending a data transport as specified in Section 10.13.3.

11.4.13 NEXT_REMOVE_CHANNEL (CN)

The NEXT_REMOVE_CHANNEL Message is used to switch off and disconnect the specified Data Channel.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

A single byte holding the Channel Number, CN[7:0], of the Data Channel that is being removed from the bus.

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	CN[7]	CN[6]	CN[5]	CN[4]	CN[3]	CN[2]	CN[1]	CN[0]

Description

This Message shall be sent only by the active Manager.

This Message informs a Device the active Manager intends to remove an existing Data Channel from the bus. A Device shall disconnect and reset any Ports connected to the specified Data Channel at the associated Reconfiguration Boundary.

This Message is sent by the active Manager as part of cancelling a Data Channel as specified in Section 10.13.6.

11.4.14 RECONFIGURE_NOW ()

The RECONFIGURE_NOW Message is used to change the bus configuration.

Message Format

This Message uses a Short Arbitration field and a Broadcast Header field.

Payload

None

Description

This Message shall be sent only by the active Manager.

This Message informs a Device the active Manager intends to change the bus configuration at the associated Reconfiguration Boundary. See Section 10.3.1.

11.5 Value Management Messages

Value Management Messages are used to communicate and update Value Elements. A Device may send information to another Device in reply to a REQUEST Message. In addition, a Device may be requested to update Value Elements.

In all cases, Value Elements are addressed through a Value Slice, which may contain one or more Value Elements. See Section 7.2 for more information on Value Slices.

11.5.1 REQUEST_VALUE (TID, EC)

The REQUEST_VALUE Message instructs a Device to send the indicated Value Slice.

Message Format

This Message uses a Short Arbitration field and a Short or Broadcast Header field.

Payload

Three bytes holding an 8-bit Transaction ID, TID[7:0], followed by a 16-bit Element Code, EC[15:0], indicating the Value Slice.

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	TID[7]	TID[6]	TID[5]	TID[4]	TID[3]	TID[2]	TID[1]	TID[0]
Byte 1	EC[7]	EC[6]	EC[5]	EC[4]	EC[3]	EC[2]	EC[1]	EC[0]
Byte 2	EC[15]	EC[14]	EC[13]	EC[12]	EC[11]	EC[10]	EC[9]	EC[8]

Description

This Message instructs the destination Device to return the specified Value Slice. Refer to Section 10.12.1 for more information and to Section 7.1 for the definition of the Element Codes.

11.5.2 REQUEST_CHANGE_VALUE (TID, EC, VU)

The REQUEST_CHANGE_VALUE Message instructs a Device to send the specified Value Slice and to update that Value Slice.

Message Format

This Message uses a Short Arbitration field and a Short or Broadcast Header field.

Payload

Three to nineteen bytes holding an 8-bit Transaction ID TID[7:0]; a 16-bit Element Code, EC[15:0], indicating the Value Slice; followed by zero to sixteen bytes of Value Update, VU[].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	TID[7]	TID[6]	TID[5]	TID[4]	TID[3]	TID[2]	TID[1]	TID[0]
Byte 1	EC[7]	EC[6]	EC[5]	EC[4]	EC[3]	EC[2]	EC[1]	EC[0]
Byte 2	EC[15]	EC[14]	EC[13]	EC[12]	EC[11]	EC[10]	EC[9]	EC[8]
Byte 3	VU[7]	VU[6]	VU[5]	VU[4]	VU[3]	VU[2]	VU[1]	VU[0]
					.			
					.			
					.			
Byte N+2	VU[8N-1]	VU[8N-2]	VU[8N-3]	VU[8N-4]	VU[8N-5]	VU[8N-6]	VU[8N-7]	VU[8N-8]

Description

This Message asks the destination Device to return the identified Value Slice, and to atomically change the value of that Value Slice according to the VU field (see Section 10.12.3). The VU field shall be placed in the LSBs of the relevant payload byte(s).

A Device shall overwrite the identified Value Slice with Value Update. If the size of Value Update is smaller than the size of the Value Slice, the Device shall pad the MSBs of Value Update with zeroes. For example, if the size of Value Update is zero bytes, the Device shall change all bits of the Value Slice to zeroes. If the size of Value Update is larger than the size of the Value Slice, the Device shall change only the portion of the Value Map corresponding to the identified Value Slice.

Refer to Section 10.12 for more information on REQUEST-REPLY transactions and to Section 7.1 for the definition of Element Codes.

11.5.3 REPLY_VALUE (TID, VS)

The REPLY_VALUE Message is sent by a Device in response to a REQUEST_VALUE or REQUEST_CHANGE_VALUE Message.

Message Format

This Message uses a Short Arbitration field and a Short Header field.

4279 **Payload**

4280 One to seventeen bytes holding an 8-bit Transaction ID, optionally followed by the requested Value Slice,
4281 VS[] in one to sixteen bytes.

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	TID[7]	TID[6]	TID[5]	TID[4]	TID[3]	TID[2]	TID[1]	TID[0]
Byte 1	VS[7]	VS[6]	VS[5]	VS[4]	VS[3]	VS[2]	VS[1]	VS[0]
.								
.								
.								
Byte N	VS[8N-1]	VS[8N-2]	VS[8N-3]	VS[8N-4]	VS[8N-5]	VS[8N-6]	VS[8N-7]	VS[8N-8]

4282 **Description**

4283 The Destination Address field shall be the Logical Address of the Device that sent the REQUEST_VALUE
4284 or REQUEST_CHANGE_VALUE Message.

4285 The Transaction ID TID[7:0] shall be the Transaction ID of the corresponding REQUEST_VALUE or
4286 REQUEST_CHANGE_VALUE Message

4287 As specified in Section 10.12.2, the payload may consist of only the Transaction ID. In all other cases the
4288 payload shall contain the Value Slice that was identified in the corresponding REQUEST Message. The
4289 Value Slice shall be placed in the LSBs of the relevant payload byte(s). Any unused bits shall be filled with
4290 zeros. Refer to Section 10.12 and Section 7.2 for more information.

4291 ### 11.5.4 CHANGE_VALUE (EC, VU)

4292 The CHANGE_VALUE Message instructs a Device to update the indicated Value Slice.

4293 **Message Format**

4294 This Message uses a Short Arbitration field and a Short or Broadcast Header field.

4295 **Payload**

4296 Two to eighteen bytes holding a 16-bit Element Code, EC[15:0], followed by zero to sixteen bytes of Value
4297 Update, VU[].

	D7	D6	D5	D4	D3	D2	D1	D0
Byte 0	EC[7]	EC[6]	EC[5]	EC[4]	EC[3]	EC[2]	EC[1]	EC[0]
Byte 1	EC[15]	EC[14]	EC[13]	EC[12]	EC[11]	EC[10]	EC[9]	EC[8]
Byte 2	VU[7]	VU[6]	VU[5]	VU[4]	VU[3]	VU[2]	VU[1]	VU[0]
.								
.								
.								
Byte N+1	VU[8N-1]	VU[8N-2]	VU[8N-3]	VU[8N-4]	VU[8N-5]	VU[8N-6]	VU[8N-7]	VU[8N-8]

Description

The VU field shall be placed in the LSBs of the relevant payload byte(s).

A Device shall overwrite the identified Value Slice with Value Update. If the size of Value Update is smaller than the size of the Value Slice, the Device shall pad the MSBs of Value Update with zeroes. For example, if the size of Value Update is zero bytes, the Device shall change all bits of the Value Slice to zeroes. If the size of Value Update is larger than the size of the Value Slice, the Device shall change only the portion of the Value Map corresponding to the identified Value Slice.

Refer to Section 7.1 for the definition of Element Codes.

11.6 Core Message Range Checking

This section provides guidelines on range checking for a received Core Message.

If the EX_ERROR Core Information Element is implemented, it shall be set when:

- the Device is the destination of the Message
- the received Core Message is supported by the Device, and
- the indicated check procedure is implemented, and
- the received Core Message fails the check.

11.6.1 Core Message Sequence Checks

Received an ASSIGN_LOGICAL_ADDRESS Message when a Logical Address has already been assigned

Received any NEXT Message before receiving a BEGIN_RECONFIGURATION Message

Received a RECONFIGURE_NOW Message before receiving a BEGIN_RECONFIGURATION Message

A RECONFIGURE_NOW Message is transmitted between the RECONFIGURE_NOW and the Reconfiguration Boundary

A BEGIN_RECONFIGURATION Message is transmitted between the RECONFIGURE_NOW and the Reconfiguration Boundary

11.6.2 Core Message Parameter Checks

Received an ASSIGN_LOGICAL_ADDRESS Message with a Logical Address prohibited in Table 42

Received a CHANGE_LOGICAL_ADDRESS Message with a Logical Address prohibited in Table 42

Received a CHANGE_ARBITRATION_PRIORITY Message with an Arbitration Priority prohibited in Table 30

Received a CONNECT_SOURCE Message with a Port number that does not exist on the Device

Received a CONNECT_SOURCE Message with the number of a local Port that is configured as input-only

Received a CONNECT_SINK Message with a Port number that does not exist on the Device

Received a CONNECT_SINK Message with the number of a local Port that is configured as output-only

Received a DISCONNECT_PORT Message with a Port number that does not exist on the Device

Received a CHANGE_CONTENT Message with a parameter value that a local Port connected to the channel does not support

Received a CHANGE_CONTENT Message with the Channel Number equal to 0 if the Channel Link bit is set

Received a REQUEST_CLEAR_INFORMATION Message with the elemental access code of an Information Element that is read-only

Received a CLEAR_INFORMATION Message with the elemental access code of an Information Element that is read-only

Received a NEXT_ACTIVE_FRAMER Message as the specified next active Framer if the Device is not a member of the Framer Device Class

Received a NEXT_DEFINE_CHANNEL Message with a parameter value that a local Port connected to the channel does not support

Received a NEXT_DEFINE_CONTENT Message with a parameter value that a local Port connected to the channel does not support

Received a NEXT_DEFINE_CONTENT Message with the Channel Number equal to 0 if the Channel Link bit is set

Received a REQUEST_CHANGE_VALUE Message with the elemental access code of a Value Element that is read-only

Received a CHANGE_VALUE Message with the elemental access code of a Value Element that is read-only

Received a Core Message with the Remaining Length Header field indicating a Message Payload shorter than specified in Section 11 for the Message

12 Device Classes

A Device Class is a category of Devices that share certain characteristics and functions. A Device Class definition is a collection of requirements on the behavior and supported features of a member of that Device Class. By fulfilling these requirements, a high level of interoperability between Devices can be achieved.

A Device Class definition shall specify the following items:

- The Device Class code.
- The version of the Device Class definition.
- Transport support requirements. These requirements include the minimum number of Ports and required attributes, e.g. directionality, Transport Protocols, etc.
- Message support requirements. These requirements include the Core Messages (also see Section 12.1), the definition of Class-specific Messages and requirements for supporting the Class-specific Messages.
- Information Element support requirements. These requirements include Core Information Elements (also see Section 12.1), the definition of the Class-specific Information Elements and the requirements for supporting the Class-specific Information Elements.
- Value Element support requirements. These requirements include Core Value Elements, the definition of Class-specific Value Elements and requirements for supporting the Class-specific Value Elements.
- Operation requirements. These requirements include all behavior that is important to the operation of a Device that belongs to that Device Class.

12.1 Device Class-independent Requirements

This section describes a set of requirements that apply to a Device regardless of its particular Device Class.

A Device shall belong to only one Device Class and shall implement only one version of the corresponding Device Class.

The Device Class shall be represented by an 8-bit code. Only a Device Class code assigned by the MIPI Alliance shall be used by a Device.

The Device Class Version (DCV) shall be represented by an 8-bit code. The first version of a Device Class definition shall specify DCV equal to 0x01. Later versions of the Device Class shall have a number one greater than the previously defined version, up to a maximum value of 255 (0xFF).

A Device shall be capable of sending the following Core Messages:

- REPLY_INFORMATION
- REPORT_PRESENT

A Device shall be capable of receiving the following Core Messages and shall perform the action specified for the Message:

- ASSIGN_LOGICAL_ADDRESS
- CHANGE_LOGICAL_ADDRESS

- CLEAR_INFORMATION
- REQUEST_CLEAR_INFORMATION
- REQUEST_INFORMATION
- REQUEST_SELF_ANNOUNCEMENT
- RESET_DEVICE

This does not constitute a requirement for a Device to support all forms of the CLEAR_INFORMATION, REQUEST_CLEAR_INFORMATION and REQUEST_INFORMATION Messages.

A Device that supports at least one Transport Protocol shall also be capable of sending the following Core Messages:

- <none>

A Device that supports at least one Transport Protocol shall also be capable of receiving the following Core Messages and shall perform the action specified for the Message:

- BEGIN_RECONFIGURATION
- CHANGE_CONTENT
- CONNECT_SINK (Sink Devices Only)
- CONNECT_SOURCE (Source Devices Only)
- DISCONNECT_PORT
- NEXT_ACTIVATE_CHANNEL
- NEXT_DEACTIVATE_CHANNEL
- NEXT_DEFINE_CHANNEL
- NEXT_DEFINE_CONTENT
- NEXT_REMOVE_CHANNEL
- RECONFIGURE_NOW

In the following rules, the statement “shall be able to generate” means that the Device shall be able to create and transmit the Message when it is required for one of the sequences as described in the Message and the sequence description.

In the following rules, the statement “shall be able to receive and interpret” means that the Device shall comply with the description of the Message and the description of sequences associated with this Message as described in the relevant sections, and shall perform any action mandated by the reception of the Message. If no action is required from the Device, it may ignore the Message but shall behave as if the Message has been successfully supported and processed.

A Device shall obey the following rules for Core Messages:

- All mandatory Core Messages shall be implemented
- Any optional Core Message may be implemented
- Core Messages that are neither mandatory nor optional shall not be implemented
- Core Messages that are neither mandatory nor optional shall be not be acted upon when received

In addition, a Device shall obey the following rules for Class-specific Messages listed for its own Device Class:

- The Device shall be able to transmit every Source-referred Class-specific Message that is mandatory as Source for its Device Class Version
- The Device may transmit any Source-referred Class-specific Message that is optional as Source for its Device Class Version
- The Device shall not transmit any other Source-referred Class-specific Message
- The Device shall be able to receive and interpret every Destination-referred Class-specific Message that is mandatory as Destination for its Device Class Version
- The Device may receive and interpret any Destination-referred Class-specific Message that is optional as Destination for its Device Class Version
- The Device shall treat all other received unicast Destination-referred Class-specific Messages as unsupported

In addition, a Device shall obey the following rules for Class-specific Messages listed for all Device Classes other than its own:

- The Device may transmit any Destination-referred Class-specific Message
- The Device may receive and interpret any Source-referred Class-specific Message

In addition, a Device shall obey the following rules for User Messages:

- The Device may transmit any Source-referred User Message
- The Device may receive and interpret any Destination-referred User Message
- The Device may transmit any Destination-referred User Message
- The Device may receive and interpret any Source-referred User Message

A Device shall support the following Core Information Elements:

- DATA_TX_COL
- DEVICE_CLASS
- DEVICE_CLASS_VERSION
- UNSPRTD_MSG

12.2 Interface Device Class

This section is the Device Class definition for SLIMbus Interface Devices.

A SLIMbus Interface Device shall have Device Class code 0xFD. A SLIMbus Interface Device that implements this version of the Device Class definition shall have Device Class Version code 0x01.

12.2.1 Port and Transport Protocol Support

There are no Port or Transport Protocol requirements defined for the Interface Device Class.

12.2.2 Interface Device Core Message Support

A Device of the Interface Device Class (Interface Device) shall be able to send all Core Messages in the "Mandatory" column, and may also send any Core Message in the "Optional" column, of the "As Source" section of Table 67.

An Interface Device shall be able to receive all Core Messages in the "Mandatory" column, and may also receive any Core Message in the "Optional" column, of the "As Destination" section of Table 67.

Table 67 Interface Device Core Message Support

As Source	
Mandatory	**Optional**
REPLY_INFORMATION	CHANGE_CONTENT
REPORT_INFORMATION	CHANGE_VALUE
REPORT_PRESENT	CLEAR_INFORMATION
	REPLY_VALUE
	REPORT_ABSENT (2)
	REQUEST_CHANGE_VALUE
	REQUEST_CLEAR_INFORMATION
	REQUEST_INFORMATION
	REQUEST_VALUE
As Destination	
Mandatory	**Optional**
ASSIGN_LOGICAL_ADDRESS	CHANGE_ARBITRATION_PRIORITY
BEGIN_RECONFIGURATION	CHANGE_CONTENT (1)
CHANGE_LOGICAL_ADDRESS	CHANGE_VALUE
CLEAR_INFORMATION	CONNECT_SINK (1)
NEXT_SUBFRAME_MODE	CONNECT_SOURCE (1)
RECONFIGURE_NOW	DISCONNECT_PORT (1)
REQUEST_CLEAR_INFORMATION	NEXT_ACTIVATE_CHANNEL (1)
REQUEST_INFORMATION	NEXT_ACTIVE_FRAMER
REQUEST_SELF_ANNOUNCEMENT	NEXT_CLOCK_GEAR (3)
RESET_DEVICE (5)	NEXT_DEACTIVATE_CHANNEL (1)
	NEXT_DEFINE_CHANNEL (1)
	NEXT_DEFINE_CONTENT (1)
	NEXT_PAUSE_CLOCK
	NEXT_REMOVE_CHANNEL (1)
	NEXT_RESET_BUS
	NEXT_ROOT_FREQUENCY (4)
	NEXT_SHUTDOWN_BUS
	REPLY_INFORMATION
	REPLY_VALUE

	REPORT_INFORMATION
	REQUEST_CHANGE_VALUE
	REQUEST_VALUE

Notes

1. *Support for these Messages is required if the Device supports a Transport Protocol (see Section 12.1).*
2. *The REPORT_ABSENT Message has special meaning when sent by an Interface Device. As described in Section 10.9.1.5, it means that the entire Component is going to cease participation in SLIMbus activity.*
3. *Support for NEXT_CLOCK_GEAR is required if any part of the Component uses the Clock Gear information (see Section 10.2.4 and Section 10.1.2.4).*
4. *Support for NEXT_ROOT_FREQUENCY is required if any part of the Component uses the Root Frequency information (see Section 10.2.4 and Section 10.1.2.4).*
5. *See Section 10.4.3.*

Messages shown in italics are mandatory for all Devices.

12.2.3 Interface Device Core Information Element Support

An Interface Device shall support all Core Information Elements in the “Mandatory” column, and may also support any Core Information Element in the “Optional” column, of Table 68.

Table 68 Interface Device Core Information Element Support

Mandatory	Optional
DATA_TX_COL	EX_ERROR
DEVICE_CLASS	RECONFIG_OBJECTION
DEVICE_CLASS_VERSION	
UNSPRTD_MSG	

Notes

Information Elements shown in italics are mandatory for all Devices.

An Interface Device shall support byte-based requests and clears of its Core Information Elements using 1, 2 and 4-byte Slice Sizes. The Device may also support other byte-based and elemental accesses. See Section 7.1 for descriptions of byte-based and elemental access methods.

12.2.4 Interface Device Class-specific Messages

No Class-specific Messages are defined for the Interface Device Class.

12.2.5 Interface Device Class-specific Information Elements

This section defines the Interface Device Class-specific Information Elements, and specifies the associated support requirements. Each Information Element shall have the attributes identified for it in Table 69.

An Interface Device shall support all Information Elements that are shown as "Mandatory", and may also support any Information Element that is shown as "Optional", in Table 69.

Table 69 Interface Device Class-specific Information Elements

Name	Description	Type	Size (bits)	Byte Address, Bit Number of Lowest Bit	Support
DATA_SLOT_OVERLAP	Internal Port contention	Clearable	1	0x400, 4	Mandatory
LOST_MS	Lost Message synchronization	Clearable	1	0x400, 3	Mandatory
LOST_SFS	Lost Superframe synchronization	Clearable	1	0x400, 2	Mandatory
LOST_FS	Lost Frame synchronization	Clearable	1	0x400, 1	Mandatory
MC_TX_COL	Message Channel Collision detected	Clearable	1	0x400, 0	Mandatory

An Interface Device shall support byte-based requests and clears of its Class-specific Information Elements using 1, 2 and 4-byte Slice Sizes. The Device may also support other byte-based and elemental accesses. See Section 7.1 for descriptions of byte-based and elemental access methods.

Each clearable Information Element in Table 69 shall be cleared by any of the following occurrences:

- the Component initially undergoing the Component Synchronization Process as described in Section 10.1.2 and Section 10.1.3
- the Interface Device being reset by a Device Reset, a Component Reset or a Bus Reset (Section 10.4)
- the Information Element being a target of an appropriately formulated REQUEST_CLEAR_INFORMATION or CLEAR_INFORMATION Message

12.2.5.1 DATA_SLOT_OVERLAP

DATA_SLOT_OVERLAP is a clearable Boolean Information Element. It indicates that there has been internal contention between Ports. An Interface Device of a Component shall set DATA_SLOT_OVERLAP whenever it suppresses the output of a Port within the Component due to Segment overlap with the Control Space or with the output of another Port.

When the Control Space is configured to overlap with a Segment, the Component in which this conflict occurs gives the Control Space priority over the Data Channel as stated in Section 6.4.2. Consequently, a Component never overwrites the Control Space with a Segment, even though the overlap is the result of explicit programming.

12.2.5.2 LOST_MS

LOST_MS is a clearable Boolean Information Element. It indicates that the Component has lost Message synchronization. An Interface Device shall set LOST_MS whenever it observes any of the following:

- Failed Guide Byte Parity Check (Section 8.3.3)
- Mismatch of the Guide Byte (Section 8.3.2)
- “Illegal” Arbitration Type (Section 8.2.1.1)
- “Illegal” Message Type (Section 8.2.2.1)

- Failure of the Primary Integrity CRC (Section 8.2.4.1)
- "Undefined" Message Response code (Section 8.2.4)
- Collision in the Message Channel (Section 10.2.3)

12.2.5.3 LOST_SFS

LOST_SFS is a clearable Boolean Information Element. It indicates that the Component has lost Superframe synchronization. An Interface Device shall set LOST_SFS whenever the Component moves to the *SeekingSuperframeSync* state from a state other than the *SeekingFrameSync* state (Figure 59).

12.2.5.4 LOST_FS

LOST_FS is a clearable Boolean Information Element. It indicates that the Component has lost Frame synchronization. The Interface Device of a Component shall set LOST_FS whenever the Component moves to the *SeekingFrameSync* state from a state other than the *Reset* state (Figure 59).

12.2.5.5 MC_TX_COL

MC_TX_COL is a clearable Boolean Information Element. It indicates that a Collision has been detected in the Message Channel. The Interface Device of a Component shall set MC_TX_COL whenever it detects a DATA line Collision (Section 5.1.1) while a Device within the Component is writing to the Message Channel.

An Interface Device based on SLIMbus specifications prior to version 1.01.01 may suppress the setting of MC_TX_COL while the Component is using Logical-OR signaling. However, an Interface Device based on SLIMbus Specification version 1.01.01 and later should not suppress the setting of MC_TX_COL while the Component is using Logical-OR signaling.

12.2.6 Reporting Class-specific Information

This section provides guidelines for reporting the value of Interface Device Class-specific Information Elements to another Device.

MC_TX_COL

Detection of a Collision in the Message Channel is most likely to be as a result of a glitch on the DATA line. However, a Component that has not yet detected it has lost Message synchronization can cause such a Collision by sending data from at least one of its Devices. The Interface Device shall send a REPORT_INFORMATION Message, including byte 0x400 of the Information Map, to the active Manager once it has completed the back-off process described in Section 10.2.3.

LOST_FS

Loss of Frame synchronization can be caused by a number of conditions during normal bus operation. If an Interface Device detects that its Component has lost Frame synchronization, it shall send a REPORT_INFORMATION Message, including byte 0x400 of the Information Map, to the active Manager once it has regained Message synchronization.

LOST_SFS

Loss of Superframe synchronization can be caused by a number of conditions during normal bus operation. If an Interface Device detects that its Component has lost Superframe synchronization, it shall send a REPORT_INFORMATION Message, including byte 0x400 of the Information Map, to the active Manager once it has regained Message synchronization.

LOST_MS

Loss of Message synchronization can be caused by a number of conditions during normal bus operation. If an Interface Device detects that its Component has lost Message synchronization, it shall send a REPORT_INFORMATION Message, including byte 0x400 of the Information Map, to the active Manager once it has regained Message synchronization.

DATA_SLOT_OVERLAP

A Data Space Slot Overlap is most likely the result of a procedural error when assigning Segment Distributions, and does not occur during normal bus operation. If an Interface Device detects a Data Space Slot overlap, it shall send a REPORT_INFORMATION Message, including byte 0x400 of the Information Map, to the active Manager at the earliest opportunity.

12.2.7 Interface Device Class-specific Value Elements

No Class-specific Value Elements are defined for the Interface Device Class.

12.2.8 Boot Requirements

There are no requirements for an Interface Device during the boot process.

12.2.9 Bus Reset Requirements

No action is required by an Interface Device when it receives a NEXT_RESET_BUS Message. Note that if, and when, a SLIMbus Reset takes effect, the Device shall reset as a result of the process described in Section 10.4.

12.2.10 Device Reset Requirements

When an Interface Device receives a RESET_DEVICE Message, it shall perform the following actions in addition to those specified for all Devices in Section 10.4.3:

- The Interface Device shall cause all Devices within the same Component to perform a reset as described in Section 10.4.2.

12.2.11 Component Reset Requirements

When an Interface Device receives a Component Reset by the mechanisms described in Section 10.4.2, it shall perform the actions specified in that section. No additional Component Reset requirements are mandated for an Interface Device.

12.2.12 Additional Requirements

An Interface Device monitors the Frame Layer within its Component. It also monitors the Message Protocol within its Component. Consequently, an Interface Device is responsible for tracking and reporting the following status information:

- Collisions in the Message Channel
- Loss and recovery of Superframe synchronization
- Loss and recovery of Frame synchronization
- Loss and recovery of Message synchronization
- Internal arbitration errors

Note that, in contrast to the listed Collisions that are tracked only by an Interface Device, all Devices are responsible for tracking Collisions in any Data Channel for which they are the data source.

12.3 Manager Device Class

This section is the Device Class definition for SLIMbus Managers.

A SLIMbus Manager shall have Device Class code 0xFF. A SLIMbus Manager that implements this version of the Device Class definition shall have Device Class Version code 0x01.

A Manager may be in one of two roles: active or inactive. As described in Section 4.3.1.1, there is only one active Manager on the bus, and this Device has unique requirements placed upon it. The support requirements for Messages and Information Elements can vary depending on the role, as can some aspects of operation.

12.3.1 Port and Transport Protocol Support

There are no Port or Transport Protocol requirements defined for the Manager Device Class.

12.3.2 Manager Core Message Support

A Manager shall be able to send all Core Messages in the “Mandatory” column, and may also send any Core Message in the “Optional” column, of the “As Source” section of Table 70.

A Manager shall be able to receive all Core Messages in the “Mandatory” column, and may also receive any Core Message in the “Optional” column, of the “As Destination” section of Table 70.

Table 70 Manager Core Message Support

As Source	
Mandatory	**Optional**
ASSIGN_LOGICAL_ADDRESS (2)	CHANGE_ARBITRATION_PRIORITY (2)
BEGIN_RECONFIGURATION (2)	CHANGE_CONTENT
CHANGE_LOGICAL_ADDRESS (2)	CHANGE_VALUE
CLEAR_INFORMATION	NEXT_PAUSE_CLOCK (2)
CONNECT_SINK (2)	NEXT_SHUTDOWN_BUS (2)
CONNECT_SOURCE (2)	REPLY_VALUE
DISCONNECT_PORT (2)	REPORT_ABSENT
NEXT_ACTIVATE_CHANNEL (2)	REPORT_INFORMATION
NEXT_ACTIVE_FRAMER (2)	REQUEST_CHANGE_VALUE
NEXT_CLOCK_GEAR (2)	REQUEST_SELF_ANNOUNCEMENT (2)
NEXT_DEACTIVATE_CHANNEL (2)	REQUEST_VALUE
NEXT_DEFINE_CHANNEL (2)	
NEXT_DEFINE_CONTENT (2)	
NEXT_REMOVE_CHANNEL (2)	
NEXT_RESET_BUS (2)	
NEXT_ROOT_FREQUENCY (2)	
NEXT_SUBFRAME_MODE (2)	

RECONFIGURE_NOW (2) *REPLY_INFORMATION* *REPORT_PRESENT (3)* REQUEST_CLEAR_INFORMATION REQUEST_INFORMATION RESET_DEVICE (2)	
As Destination	
Mandatory	**Optional**
ASSIGN_LOGICAL_ADDRESS (3) *CHANGE_LOGICAL_ADDRESS (3)* *CLEAR_INFORMATION* REPLY_INFORMATION REPORT_INFORMATION REPORT_PRESENT (2) *REQUEST_CLEAR_INFORMATION* REQUEST_INFORMATION REQUEST_SELF_ANNOUNCEMENT RESET_DEVICE (3)	BEGIN_RECONFIGURATION (1) CHANGE_ARBITRATION_PRIORITY CHANGE_CONTENT (1) CHANGE_VALUE CONNECT_SINK (1) CONNECT_SOURCE (1) DISCONNECT_PORT (1) NEXT_ACTIVATE_CHANNEL (1) NEXT_ACTIVE_FRAMER NEXT_CLOCK_GEAR NEXT_DEACTIVATE_CHANNEL (1) NEXT_DEFINE_CHANNEL (1) NEXT_DEFINE_CONTENT (1) NEXT_PAUSE_CLOCK NEXT_REMOVE_CHANNEL (1) NEXT_RESET_BUS NEXT_ROOT_FREQUENCY NEXT_SHUTDOWN_BUS NEXT_SUBFRAME_MODE RECONFIGURE_NOW (1) REPLY_VALUE REPORT_ABSENT (2) REQUEST_CHANGE_VALUE REQUEST_VALUE

Notes

1. *Support for these Messages is required if the Device supports a Transport Protocol (see Section 12.1).*
2. *These Messages shall be supported only when the Device is the active Manager.*
3. *These Messages shall be supported only when the Device is not the active Manager.*

Messages shown in italics are mandatory for all Devices.

12.3.3 Manager Core Information Element Support

A Manager shall support all Core Information Elements in the “Mandatory” column, and may also support any Core Information Element in the “Optional” column, of Table 71.

Table 71 Manager Core Information Element Support

Mandatory	Optional
DATA_TX_COL	EX_ERROR
DEVICE_CLASS	RECONFIG_OBJECTION
DEVICE_CLASS_VERSION	
UNSPRTD_MSG	

Notes

Information Elements shown in italics are mandatory for all Devices.

A Manager shall support byte-based requests and clears of its Core Information Elements using 1, 2 and 4-byte Slice Sizes. The Device may also support other byte-based and elemental accesses. See Section 7.1 for descriptions of byte-based and elemental access methods.

12.3.4 Manager Class-specific Messages

No Class-specific Messages are defined for the Manager Device Class.

12.3.5 Manager Class-specific Information Elements

This section defines the Manager Class-specific Information Elements, and specifies the associated support requirements. Each Information Element shall have the attributes identified for it in Table 72.

A Manager shall support all Information Elements that are shown as "Mandatory", and may also support any Information Element that is shown as "Optional", in Table 72.

Table 72 Manager Class-specific Information Elements

Name	Description	Type	Size (bits)	Byte Address, Bit Number of Lowest Bit	Support
ACTIVE_MANAGER	The Device is the active Manager	Read only	1	0x400, 0	Mandatory

A Manager shall support byte-based requests and clears of its Class-specific Information Elements using 1, 2 and 4-byte Slice Sizes. The Device may also support other byte-based and elemental accesses. See Section 7.1 for descriptions of byte-based and elemental access methods.

12.3.5.1 ACTIVE_MANAGER

ACTIVE_MANAGER is a read-only Boolean Information Element. It shall have the value TRUE if the Device is the active Manager and FALSE if the Device is not the active Manager (Section 4.3.1.2).

12.3.6 Reporting Class-specific Information

This section provides guidelines for reporting the value of Manager Class-specific Information Elements to another Device.

ACTIVE_MANAGER

This Information Element is fixed and should not be reported by the REPORT_INFORMATION Message during normal bus operation except when sent as part of byte 0x400 of the Information Map.

12.3.7 Manager Class-specific Value Elements

No Class-specific Value Elements are defined for the Manager Device Class.

12.3.8 Boot Requirements

There are no requirements for a Manager during the boot process.

12.3.9 Bus Reset Requirements

No action is required by a Manager when it receives a NEXT_RESET_BUS Message.

12.3.10 Device Reset Requirements

When a Manager receives a RESET_DEVICE Message, the only actions required are those specified for all Devices in Section 10.4.3.

12.3.11 Component Reset Requirements

When a Manager receives a Component Reset by the mechanisms described in Section 10.4.2, it shall perform the actions specified in that section. No additional Component Reset requirements are mandated for a Manager.

12.3.12 Additional Requirements

The active Manager has many requirements that are unique in that it is responsible for the establishment, maintenance, and shutdown of the entire SLIMbus system.

12.4 Framer Device Class

This section is the Device Class definition for SLIMbus Framers.

A SLIMbus Framer shall have Device Class code 0xFE. A SLIMbus Framer that implements this version of the Device Class definition shall have Device Class Version code 0x01.

A Framer may be in one of two roles: active or inactive. As described in Section 4.3.1.3, there is only one active Framer on the bus at a given time, and this Device has unique requirements placed upon it. The support requirements for Messages and Information Elements can vary depending on the role, as can some aspects of operation.

12.4.1 Port and Transport Protocol Support

There are no Port or Transport Protocol requirements defined for the Framer Device Class.

12.4.2 Framer Core Message Support

A Framer shall be able to send all Core Messages in the “Mandatory” column, and may also send any Core Message in the “Optional” column, of the “As Source” section of Table 73.

A Framer shall be able to receive all Core Messages in the “Mandatory” column, and may also receive any Core Message in the “Optional” column, of the “As Destination” section of Table 73.

Note that the requirements for Core Message support are the same whether a Device is the active Framer or an inactive Framer. However, the actions taken by the Device when certain Messages are received may be different depending on whether the Device is the active Framer or an inactive Framer.

Table 73 Framer Core Message Support

As Source	
Mandatory	**Optional**
REPLY_INFORMATION	CHANGE_CONTENT
REPORT_PRESENT	CHANGE_VALUE
	CLEAR_INFORMATION
	REPLY_VALUE
	REPORT_ABSENT
	REPORT_INFORMATION
	REQUEST_CHANGE_VALUE
	REQUEST_CLEAR_INFORMATION
	REQUEST_INFORMATION
	REQUEST_VALUE
As Destination	
Mandatory	**Optional**
ASSIGN_LOGICAL_ADDRESS	CHANGE_ARBITRATION_PRIORITY
BEGIN_RECONFIGURATION	CHANGE_CONTENT (1)
CHANGE_LOGICAL_ADDRESS	CHANGE_VALUE
CLEAR_INFORMATION	CONNECT_SINK (1)
NEXT_ACTIVE_FRAMER	CONNECT_SOURCE (1)
NEXT_CLOCK_GEAR	DISCONNECT_PORT (1)
NEXT_PAUSE_CLOCK	NEXT_ACTIVATE_CHANNEL (1)
NEXT_RESET_BUS	NEXT_DEACTIVATE_CHANNEL (1)
NEXT_ROOT_FREQUENCY	NEXT_DEFINE_CHANNEL (1)
NEXT_SHUTDOWN_BUS	NEXT_DEFINE_CONTENT (1)
NEXT_SUBFRAME_MODE	NEXT_REMOVE_CHANNEL (1)
RECONFIGURE_NOW	REPLY_INFORMATION
REQUEST_CLEAR_INFORMATION	REPLY_VALUE
REQUEST_INFORMATION	REPORT_INFORMATION
REQUEST_SELF_ANNOUNCEMENT	REQUEST_CHANGE_VALUE
RESET_DEVICE (2)	REQUEST_VALUE

Notes

1. *Support for these Messages is required if the Device supports a Transport Protocol (see Section 12.1).*
2. *See Section 10.4.3.*

Messages in italics are mandatory for all Devices.

12.4.3 Framer Core Information Element Support

A Framer shall support all Core Information Elements in the "Mandatory" column, and may also support any Core Information Element in the "Optional" column, of Table 74.

Table 74 Framer Core Information Element Support

Mandatory	Optional
DATA_TX_COL	EX_ERROR
DEVICE_CLASS	RECONFIG_OBJECTION
DEVICE_CLASS_VERSION	
UNSPRTD_MSG	

Notes

Information Elements shown in italics are mandatory for all Devices.

A Framer shall support byte-based requests and clears of its Core Information Elements using 1, 2 and 4-byte Slice Sizes. The Device may also support other byte-based and elemental accesses. See Section 7.1 for descriptions of byte-based and elemental access methods.

12.4.4 Framer Class-specific Messages

No Class-specific Messages are defined for the Framer Device Class.

12.4.5 Framer Class-specific Information Elements

This section defines the Framer Class-specific Information Elements, and specifies the associated support requirements. Each Information Element shall have the attributes identified for it in Table 75.

A Framer shall support all Information Elements that are shown as "Mandatory", and may also support any Information Element that is shown as "Optional", in Table 75.

Table 75 Framer Class-specific Information Elements

Name	Description	Type	Size (bits)	Byte Address, Bit Number of Lowest Bit	Support
QUALITY	Quality of the generated clock	Read only	2	0x400, 6	Optional
GC_TX_COL	Guide Channel Collision detected	Clearable	1	0x400, 3	Mandatory
FI_TX_COL	Framing Information Collision detected	Clearable	1	0x400, 2	Mandatory

Name	Description	Type	Size (bits)	Byte Address, Bit Number of Lowest Bit	Support
FS_TX_COL	Frame Sync Symbol Collision detected	Clearable	1	0x400, 1	Mandatory
ACTIVE_FRAMER	The Device is the active Framer	Read only	1	0x400, 0	Mandatory

A Framer shall support byte-based requests and clears of its Class-specific Information Elements using 1, 2 and 4-byte Slice Sizes. The Device may also support other byte-based and elemental accesses. See Section 7.1 for descriptions of byte-based and elemental access methods.

Each clearable Information Element in Table 75 shall be cleared by any of the following occurrences:

- the Component initially undergoing the Component Synchronization Process as described in Section 10.1.2 and Section 10.1.3
- the Framer being reset by a Device Reset, a Component Reset or a Bus Reset (Section 10.4)
- the Information Element being a target of an appropriately formulated REQUEST_CLEAR_INFORMATION or CLEAR_INFORMATION Message

12.4.5.1 QUALITY

QUALITY is a read-only, enumerated Information Element and is coded using 2-bits. It is used to report the quality of the CLK signal that is generated by the Framer. The Framer shall code this Information Element with one of the values listed in Table 76.

Table 76 QUALITY Information Element

Field	Description
0b00	Potentially irregular punctured clock unsuitable for use as a timing reference for generic phase-locked loops.
0b01	Irregular clock that has a lower-frequency regular clock embedded. For example, such clocks can be created by cyclic pulse swallowing.
0b10	Regular clock that potentially has more than 1 ns RMS of wideband jitter (100 Hz measurement corner, see [AES01]).
0b11	Low-jitter clock that is known to have less than 1 ns RMS of wideband jitter (100 Hz measurement corner, see [AES01]).

The QUALITY Information Element may be changed by mechanisms outside the scope of this document.

12.4.5.2 GC_TX_COL

GC_TX_COL is a clearable Boolean Information Element. It indicates a Collision has been detected in the Guide Channel. The Framer shall set GC_TX_COL whenever it detects a DATA line Collision (Section 5.1.1) while it is writing the Guide Byte.

12.4.5.3 FI_TX_COL

FI_TX_COL is a clearable Boolean Information Element. It indicates a Collision has been detected in a Framing Information Slot. The Framer shall set FI_TX_COL whenever it detects a DATA line Collision (Section 5.1.1) while it is writing the Framing Information.

12.4.5.4 FS_TX_COL

FS_TX_COL is a clearable Boolean Information Element. It indicates a Collision has been detected in a Frame Sync Slot. The Framer shall set FS_TX_COL whenever it detects a DATA line Collision (Section 5.1.1) while it is writing the Frame Sync symbol.

12.4.5.5 ACTIVE_FRAMER

ACTIVE_FRAMER is a read-only Boolean Information Element. It shall have the value TRUE if the Device is the active Framer and FALSE if the Device is not the active Framer (Section 4.3.1.3 and Section 10.8).

12.4.6 Reporting Class-specific Information

This section provides guidelines for reporting the value of Framer Class-specific Information Elements to another Device.

QUALITY

A change in the quality of the CLK signal may occur at any time as part of the normal operation of the bus. The active Framer should send a REPORT_INFORMATION Message that includes byte 0x400 of the Information Map to the active Manager, if this Information Element changes to a lower quality value. The active Framer may also send a REPORT_INFORMATION Message that includes byte 0x400 of the Information Map to the active Manager, if this Information Element changes to a higher quality value.

GC_TX_COL

If the active Framer detects a Collision in the Guide Channel, it is either the result of a single glitch on the DATA line or of at least one Device writing in the wrong Slot. The active Framer may attempt to send a REPORT_INFORMATION Message that includes byte 0x400 of the Information Map to the active Manager once the Guide Channel is free of Collisions.

FI_TX_COL

If the active Framer detects a Collision in the Framing Information, it is either the result of a single glitch on the DATA line or of at least one Device writing in the wrong Slot. The active Framer may attempt to send a REPORT_INFORMATION Message that includes byte 0x400 of the Information Map to the active Manager once the Framing Channel is free of Collisions.

FS_TX_COL

If the active Framer detects a Collision in the Frame Sync symbol, it is either the result of a single glitch on the DATA line or of at least one Device writing in the wrong Slot. The active Framer should attempt to send a REPORT_INFORMATION Message that includes byte 0x400 of the Information Map to the active Manager once the Framing Channel is free of Collisions.

ACTIVE_FRAMER

The ACTIVE_FRAMER Information Element is changed only by the process of Framer handover. Since Framer handover is initiated by the active Manager, this Information Element should not be reported by the REPORT_INFORMATION Message in normal bus operation except when sent as part of byte 0x400 of the Information Map.

12.4.7 Framer Class-specific Value Elements

No Class-specific Value Elements are defined for the Framer Device Class.

12.4.8 Boot Requirements

If a Framer is part of the Clock Sourcing Component used to boot the bus, it shall provide an initial value for the Subframe Mode, Clock Gear, and Root Frequency for the Component. These values shall be used by the Component until changed by the active Manager as described in Section 10.5, Section 10.6, and Section 10.7.

12.4.9 Bus Reset Requirements

When the active Framer receives a NEXT_RESET_BUS Message, it shall prepare to perform a Bus Reset as described in Section 10.4.

No action is required by an inactive Framer when it receives a NEXT_RESET_BUS Message.

12.4.10 Device Reset Requirements

When the active Framer receives a RESET_DEVICE Message, it shall perform the following actions in addition to those specified for all Devices in Section 10.4.3:

- The active Framer shall continue clocking the bus and continue writing the Framing Channel and Guide Channel.

When an inactive Framer receives a RESET_DEVICE Message, the only actions required are those specified for all Devices in Section 10.4.3.

12.4.11 Component Reset Requirements

When a Framer receives a Component Reset by the mechanisms described in Section 10.4.2, it shall perform the actions specified in that section. No additional Component Reset requirements are mandated for Devices of the Framer Device Class.

12.4.12 Additional Requirements

The active Framer is responsible for tracking and reporting certain items to do with the Component's physical interface:

- Collisions with the Frame Sync symbol
- Collisions in the Framing Information
- Collisions in the Guide Channel field

Note that, in contrast to the listed Collisions that are tracked only by the active Framer, all Devices are responsible for tracking Collisions in any Data Channels for which they are the data source.

The active Framer also has other requirements that are unique in that it is responsible for toggling the CLK line, sending the Frame Sync symbol, sending the Framing Information and sending the Guide Byte.

A Framer shall be capable of assuming and relinquishing the role of active Framer using the process described in Section 10.8.

12.5 Generic Device Class

This section presents requirements that apply to those members of the Generic Device Class that implement SLIMbus.

A Member of the Generic Device Class shall have Device Class code 0x00. In the absence of a separate Generic Device Class definition, a member of the Generic Device Class shall have Device Class Version code 0x01.

12.5.1 Port and Transport Protocol Support

There are no Port or Transport Protocol requirements defined for the Generic Device Class.

12.5.2 Generic Device Core Message Support

A Device of the Generic Device Class (Generic Device) shall be able to send all Core Messages in the "Mandatory" column, and may also send any Core Message in the "Optional" column, of the "As Source" section of Table 77.

A Generic Device shall be able to receive all Core Messages in the "Mandatory" column, and may also receive any Core Message in the "Optional" column, of the "As Destination" section of Table 77.

Table 77 Generic Device Core Message Support

As Source	
Mandatory	**Optional**
REPLY_INFORMATION	CHANGE_CONTENT
REPORT_PRESENT	CHANGE_VALUE
	CLEAR_INFORMATION
	REPLY_VALUE
	REPORT_ABSENT
	REPORT_INFORMATION
	REQUEST_CHANGE_VALUE
	REQUEST_CLEAR_INFORMATION
	REQUEST_INFORMATION
	REQUEST_VALUE
As Destination	
Mandatory	**Optional**
ASSIGN_LOGICAL_ADDRESS	BEGIN_RECONFIGURATION (1)
CHANGE_LOGICAL_ADDRESS	CHANGE_ARBITRATION_PRIORITY
CLEAR_INFORMATION	CHANGE_CONTENT (1)
REQUEST_CLEAR_INFORMATION	CHANGE_VALUE

REQUEST_INFORMATION	CONNECT_SINK (1)
REQUEST_SELF_ANNOUNCEMENT	CONNECT_SOURCE (1)
RESET_DEVICE	DISCONNECT_PORT (1)
	NEXT_ACTIVATE_CHANNEL (1)
	NEXT_ACTIVE_FRAMER
	NEXT_CLOCK_GEAR
	NEXT_DEACTIVATE_CHANNEL (1)
	NEXT_DEFINE_CHANNEL (1)
	NEXT_DEFINE_CONTENT (1)
	NEXT_PAUSE_CLOCK
	NEXT_REMOVE_CHANNEL (1)
	NEXT_ROOT_FREQUENCY
	NEXT_RESET_BUS
	NEXT_SHUTDOWN_BUS
	NEXT_SUBFRAME_MODE
	RECONFIGURE_NOW (1)
	REPLY_INFORMATION
	REPLY_VALUE
	REPORT_INFORMATION
	REQUEST_VALUE
	REQUEST_CHANGE_VALUE

Notes

1. *Support for these Messages is required if the Device supports a Transport Protocol (see Section 12.1).*

Messages shown in italics are mandatory for all Devices.

12.5.3 Generic Device Core Information Element Support

A Generic Device shall support all Core Information Elements in the “Mandatory” column, and may also support any Core Information Element in the “Optional” column, of Table 78.

Table 78 Generic Device Core Information Element Support

Mandatory	Optional
DATA_TX_COL	EX_ERROR
DEVICE_CLASS	RECONFIG_OBJECTION
DEVICE_CLASS_VERSION	
UNSPRTD_MSG	

Notes

Information Elements shown in italics are mandatory for all Devices.

A Generic Device shall support byte-based requests and clears of its Core Information Elements using 1, 2 and 4-byte Slice Sizes. The Device may also support other byte-based and elemental accesses. See Section 7.1 for descriptions of byte-based and elemental access methods.

12.5.4 Generic Device Class-specific Messages

No Class-specific Messages are defined for the Generic Device Class.

12.5.5 Generic Device Class-specific Information Elements

No Class-specific Information Elements are defined for the Generic Device Class.

12.5.6 Reporting Class-specific Information

No guidelines for reporting the value of Generic Device Class-specific Information Elements are provided.

12.5.7 Generic Device Class-specific Value Elements

No Class-specific Value Elements are defined for the Generic Device Class.

12.5.8 Boot Requirements

There are no requirements for a Generic Device during the boot process.

12.5.9 Bus Reset Requirements

No action is required by a Generic Device when it receives a NEXT_RESET_BUS Message.

12.5.10 Device Reset Requirements

When a Generic Device receives a RESET_DEVICE Message, the only actions required are those specified for all Devices in Section 10.4.3.

12.5.11 Component Reset Requirements

When a Generic Device receives a Component Reset by the mechanisms described in Section 10.4.2, it shall perform the actions specified in that section. No additional Component Reset requirements are mandated for a Generic Device.

12.5.12 Additional Requirements

No additional requirements are defined for the Generic Device Class.

Annex A Usage Examples (informative)

SLIMbus enables many different ways to partition audio functionality in a system. Therefore, the use cases in this section can be seen as inspiration for basic SLIMbus system configurations. Assumptions are made to present as realistic as possible scenarios without getting into implementation-specific details.

A.1 Assumed Pre-conditions

- Setup and configuration of SLIMbus is not included in the use case examples.
- Combined Application and baseband processor (for simplicity only)
- Manager and Framer included in the combined application/baseband processor
- Microphone can support only 8 kHz sample rate source
- Ports in Devices assumed to be statically assigned and unidirectional
- Music has a 44.1 kHz sample rate and voice call audio has an 8 kHz sample rate

A.2 Use Cases

The use cases in this section are all based on a low complexity audio system with five physical Components on SLIMbus. The Components are: application processor, microphone, Bluetooth modem, stereo amplifier with loudspeakers, and voice call speaker.

A.2.1 Use Case 1 – Voice Call, Handset

Use Case 1 is the ordinary voice call use case. Downlink audio is routed from the baseband Component to the voice call speaker. Also, uplink audio is routed from the microphone to the application processor.

The voice call audio is transferred using the Isochronous Transport Protocol.

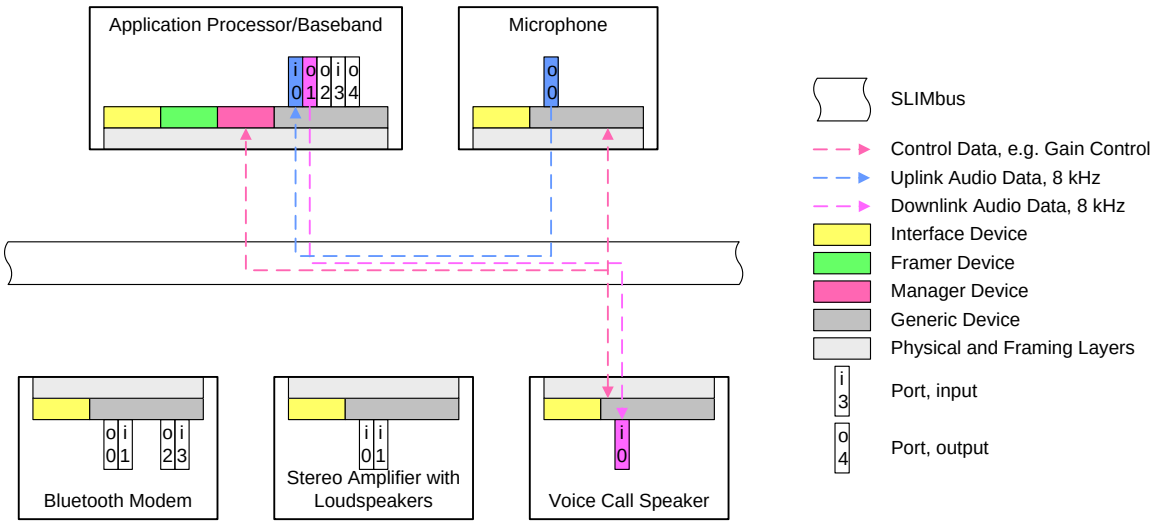


Figure 84 Use Case 1

The bus could be configured according to Table 79 and Table 80.

4871 **Table 79 Use Case 1 and 2 Configuration**

Description	Value	Units
Root Frequency	24.576	MHz
Clock Gear	4	
Actual Bus Clock	0.384	MHz
Subframe Rate	4	kHz
Frame Rate	0.5	kHz
Superframe Rate	62.5	Hz
Subframe Length	24	Slots
Control Space Width	4	Slots
Control Space Bandwidth	64	kbps
Data Channel Rate – Voice	8	kHz
Segment Length – Voice	4	Slots
Data Channel Bandwidth – Voice	128	kbps

4872 **Table 80 Use Case 1 and 2 – Slot Layout**

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80	81	82	83
84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107
108	109	110	111	112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127	128	129	130	131
132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155
156	157	158	159	160	161	162	163	164	165	166	167
168	169	170	171	172	173	174	175	176	177	178	179
180	181	182	183	184	185	186	187	188	189	190	191

4873 The font color in Table 80 is intended to be connected to the colors of the lines in the routing picture. The
4874 two orange Slots in the Control Space hold the Guide Channel (only in the first Frame of every
4875 Superframe).

4876 ### A.2.2 Use Case 2 – Voice Call, Handheld Hands Free

4877 Use Case 2 utilizes the same Slot layout as the example in Section A.2.1. In this example, the stereo
4878 amplifier is used as the sink for the downlink voice call audio.

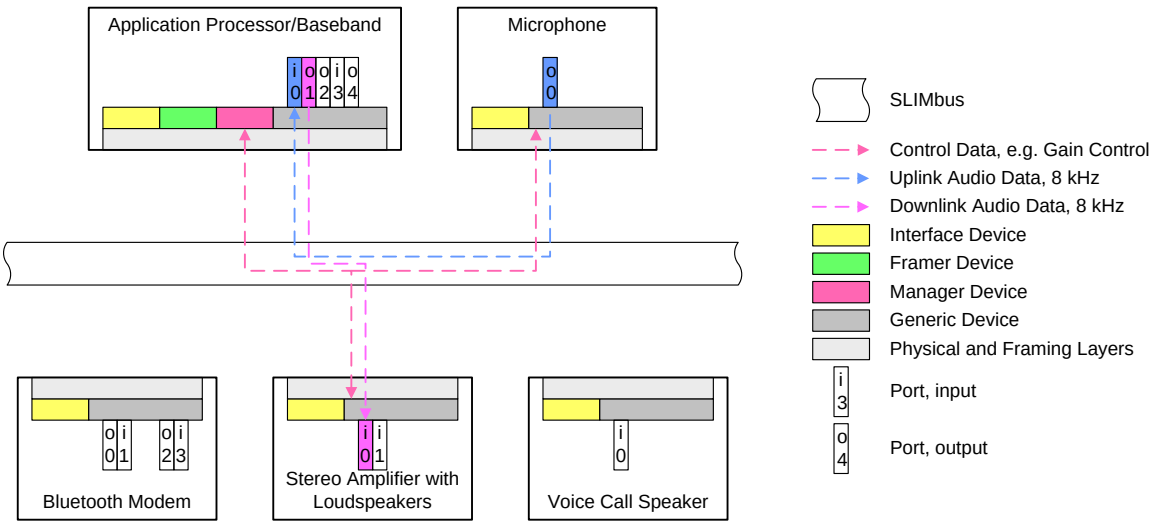


Figure 85 Use Case 2, Hands Free

A.2.3 Use Case 3 – Voice Call, Bluetooth® Headset

In Use Case 3, the voice call audio is routed to and from the Bluetooth modem and the application processor. To handle control and data communication between the application processor and the Bluetooth modem, a full-duplex Asynchronous link is set up between those Devices.

Voice call data is transferred using the Isochronous Transport Protocol while the Bluetooth data is transferred using the Simplex Asynchronous Transport Protocol.

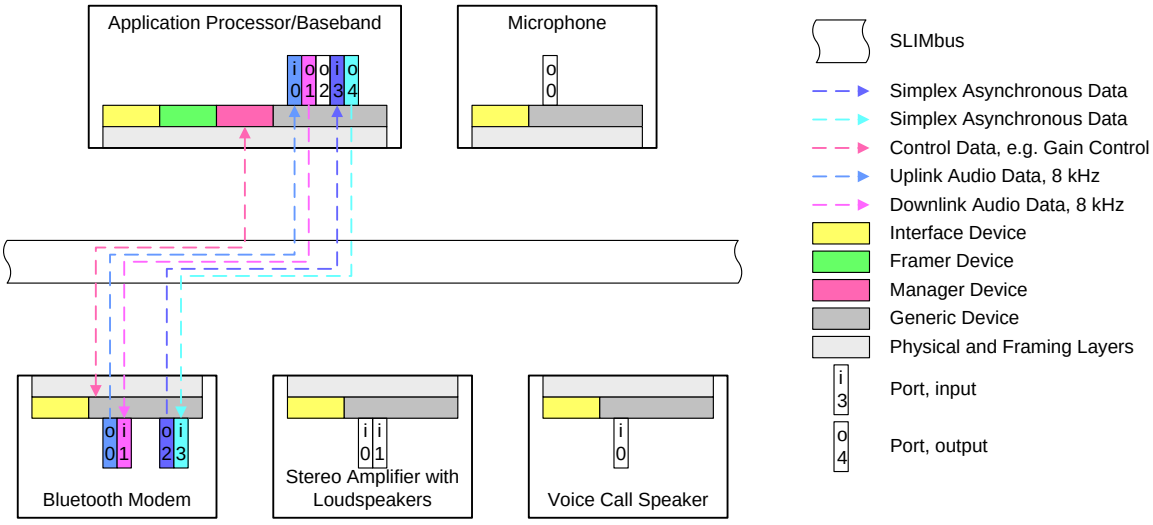


Figure 86 Use Case 3, Bluetooth

The bus could be configured according to Table 81 and Table 82.

Table 81 Use Case 3 Configuration

Description	Value	Units
Root Frequency	24.576	MHz
Clock Gear	4	

Description	Value	Units
Actual Bus Clock	0.384	MHz
Subframe Rate	4	kHz
Frame Rate	0.5	kHz
Superframe Rate	62.5	Hz
Subframe Length	24	Slots
Control Space Width	4	Slots
Control Space Bandwidth	64	kbps
Segment Length – Voice	4	Slots
Data Channel Rate – Voice	8	kHz
Data Channel Bandwidth – Voice	128	kbps
Segment Length – Bluetooth data	4	Slots
Data Channel Rate – Bluetooth data	2	kHz
Data Channel Bandwidth – Bluetooth data	32	kbps

Table 82 Use Case 3 – Slot Layout

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80	81	82	83
84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107
108	109	110	111	112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127	128	129	130	131
132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155
156	157	158	159	160	161	162	163	164	165	166	167
168	169	170	171	172	173	174	175	176	177	178	179
180	181	182	183	184	185	186	187	188	189	190	191

The font color in Table 82 is intended to be connected to the colors of the lines in the routing picture. The two orange Slots in the Control Space hold the Guide Channel (only in the first Frame of every Superframe).

A.2.4 Use Case 4 – Music Playback, Built-in Stereo Speakers

In Use Case 4, music data is transferred from the application processor to the stereo amplifier.

Music is transferred using the Pushed Transport Protocol with four Slots of audio data and one TAG field Slot. This configuration enables a 44.1 kHz sample rate flow to be distributed in a 48 kHz sample rate channel.

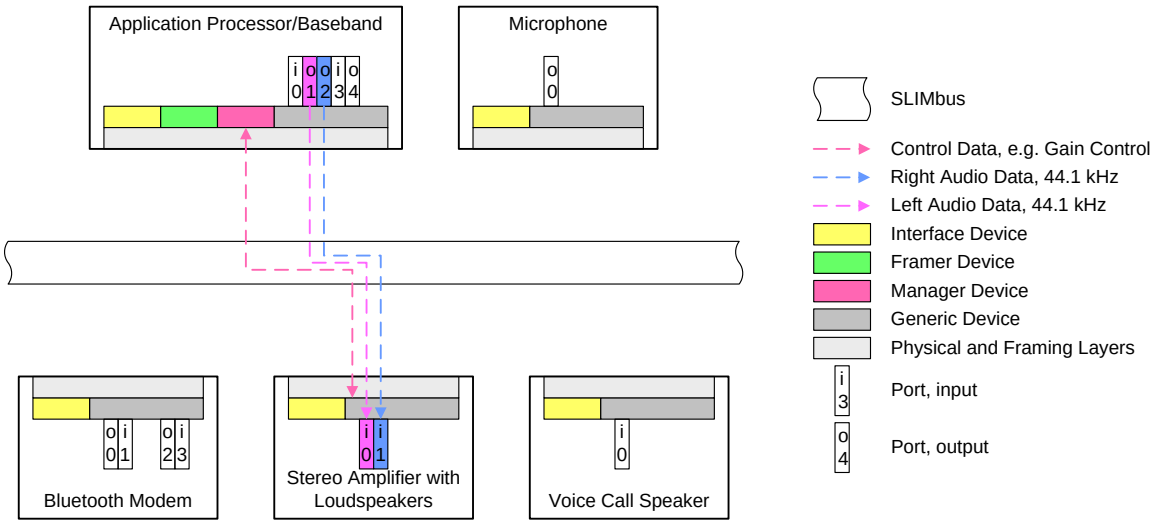


Figure 87 Use Case 4, Built-in Speakers

The bus could be set up according to Table 83 and Table 84.

Table 83 Use Case 4 Configuration

Description	Value	Units
Root Frequency	24.576	MHz
Clock Gear	7	
Actual Bus Clock	3.072	MHz
Subframe Rate	24	kHz
Frame Rate	4	kHz
Superframe Rate	0.5	kHz
Subframe Length	32	Slots
Control Space Width	4	Slots
Control Space Bandwidth	384	kbps
Segment Length – Music	5	Slots
Data Channel Rate – Music	48	kHz
Data Channel Bandwidth – Music	960	kbps

Table 84 Use Case 4 – Slot Layout

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79

80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191

The font color in Table 84 is intended to be connected to the colors of the lines in the routing picture. The two orange Slots in the Control Space hold the Guide Channel (only in the first Frame of every Superframe).

A.2.5 Use Case 5 – Ring Signal and Voice Answer

In Use Case 5, a ring signal is generated by the application processor and is routed to the stereo amplifier. In addition, the phone is configured to respond to voice control commands.

Music is transferred using the Pushed Transport Protocol with four Slots of audio data and one TAG field Slot. This configuration enables a 44.1 kHz sample rate flow to be distributed in a 48 kHz sample rate channel.

The voice call audio is transferred using the Isochronous Transport Protocol.

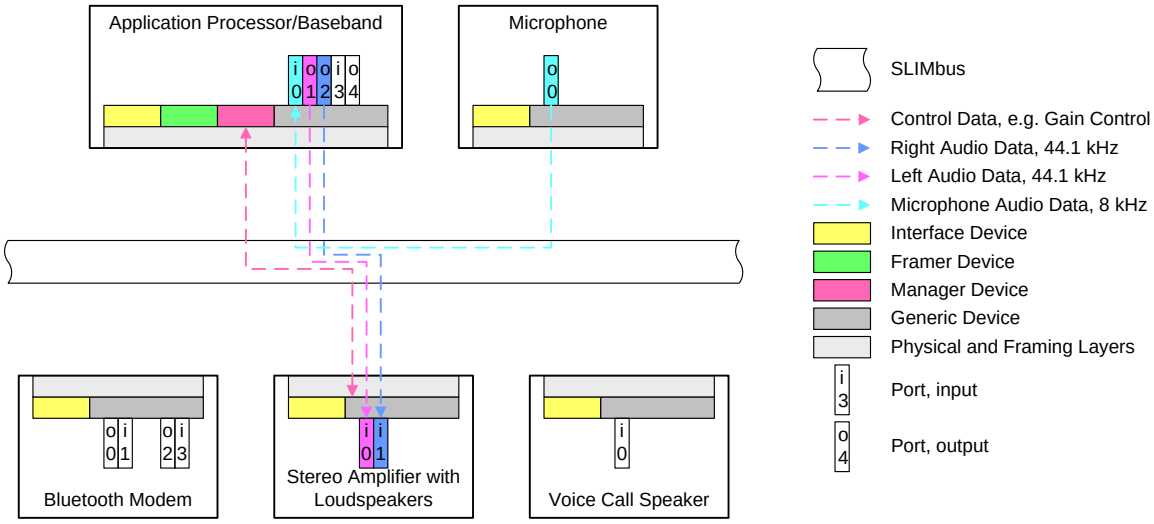


Figure 88 Use Case 5, Ring Signal

The bus could be set up according to Table 85 and Table 86.

Table 85 Use Case 5 Configuration

Description	Value	Unit
Root Frequency	24.576	MHz
Clock Gear	7	
Actual bus clock	3.072	MHz

Description	Value	Unit
Subframe Rate	24	kHz
Frame Rate	4	kHz
Superframe Rate	0.5	kHz
Subframe Length	32	Slots
Control Space Width	4	Slots
Control Space Bandwidth	384	kbps
Segment Length – Voice	4	Slots
Data Channel Rate – Voice	8	kHz
Data Channel Bandwidth – Voice	128	kbps
Segment Length – Music	5	Slots
Data Channel Rate – Music	48	kHz
Data Channel Bandwidth – Music	960	kbps

Table 86 Use Case 5 -- Slot Layout

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191

The font color in Table 86 is intended to be connected to the colors of the lines in the routing picture. The two orange Slots in the Control Space hold the Guide Channel (only in the first Frame of every Superframe).

Annex B Physical Layer Implementation Notes (informative)

B.1 Why Does SLIMbus Require a Dedicated Physical Layer?

SLIMbus is a multi-drop bus that allows multiple slow speed devices within a mobile handset to communicate in a way that minimizes system cost. It does this primarily by eliminating a number of legacy point-to-point serial interfaces and associated electrical traces. A conceptual SLIMbus system topology is shown in Figure 89

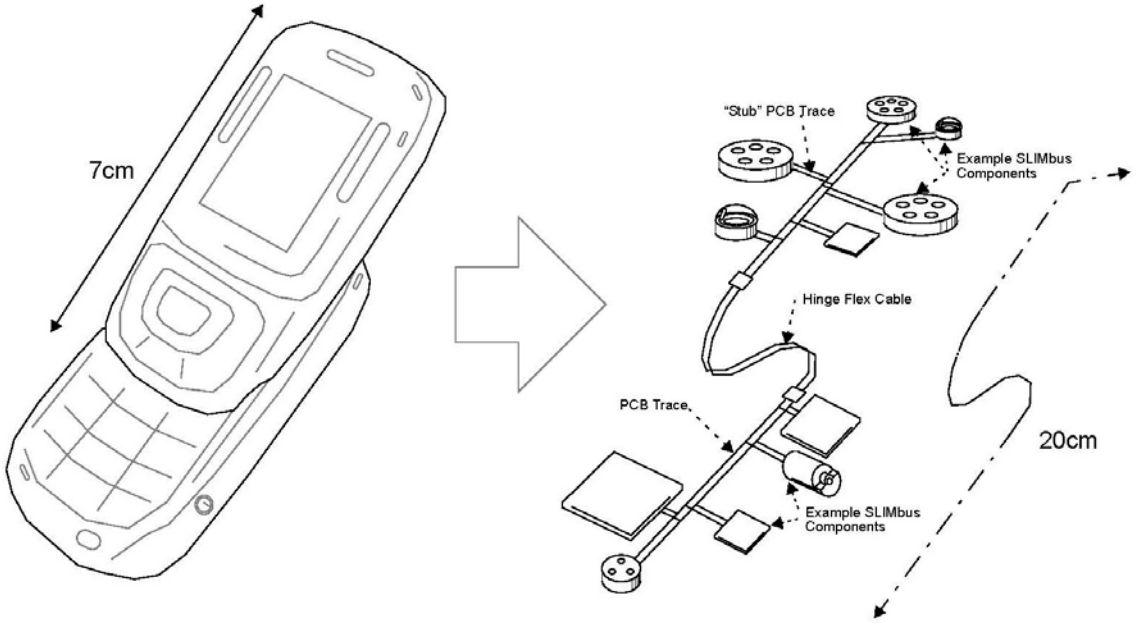


Figure 89 System Topology of an Example Handset Implementing SLIMbus

Here we clearly see how a reduction in the number of low speed point-to-point interfaces simplifies the design of the handset. This diagram also serves to highlight the complexities introduced at the physical layer by a move from multiple point-to-point buses to one larger multi-drop bus, for example line length, increased loading and possibly flexible wiring with varying line impendence to cope with hinges and sliders.

SLIMbus is designed to be capable of existing cost-effectively on multiple types of Device, from loudspeakers, microphones and HID systems to low voltage baseband controllers and wireless modems. The aforementioned systems exert conflicting constraints on the physical layer of the interface and the resulting specification has been optimized for saving cost and power at the system level, sometimes at the expense of additional complexity, and perhaps cost, at the Device level. Where possible this complexity has been minimized and it is envisioned that the cost of any such complexity will reduce over time as adoption increases.

B.2 Typical System Power Budget

The power required to transmit a digital signal is usually modeled using the following approximation:

$$P = fCV^2 + Ps + Pl$$

where the active power, fCV^2, typically dominates. The short circuit and leakage (quiescent) power will depend on the chosen driver topology. An approximation of active bus power consumption with random data would be 1.25 * fCV^2. This gives a range of:

12.288 MHz into 75 pF at 1.8 V requires 4 mW

384 kHz into 30 pF at 1.2 V requires 21 μW

B.3 Example Slew Rate Control Circuits

There are numerous methods for controlling the slew rate of the CLK line. Three example methods are outlined here as examples, though no particular topology is recommended:

- Nested Miller Driver
- Gradual Turn On Driver
- Transition Time Locked Driver

B.3.1 Nested Miller AC Feedback Driver

The concept of using AC feedback on an output driver is quite simple. When the output voltage is moving rapidly this is fed back to the pre-driver to reduce the overdrive on the output pair, regulating the slew rate into a wide range of loads by directly limiting the drivers bandwidth.

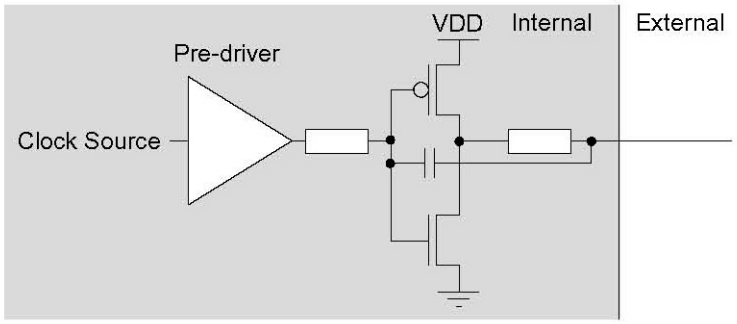


Figure 90 Using AC Feedback in a CMOS Output Driver

This technique is similar to the stabilization capacitors used to create a dominant pole in driver topologies aimed for transmitting analog outputs. In such a situation the two output FETs are driven with pre-driver signals that are biased to allow an analog output, such biasing is not required for SLIMbus where the input and output signals are firmly digital.

B.3.2 Gradual Turn-On Driver

Gradual turn on drivers use sets of output drivers that can be turned on in parallel at different times.

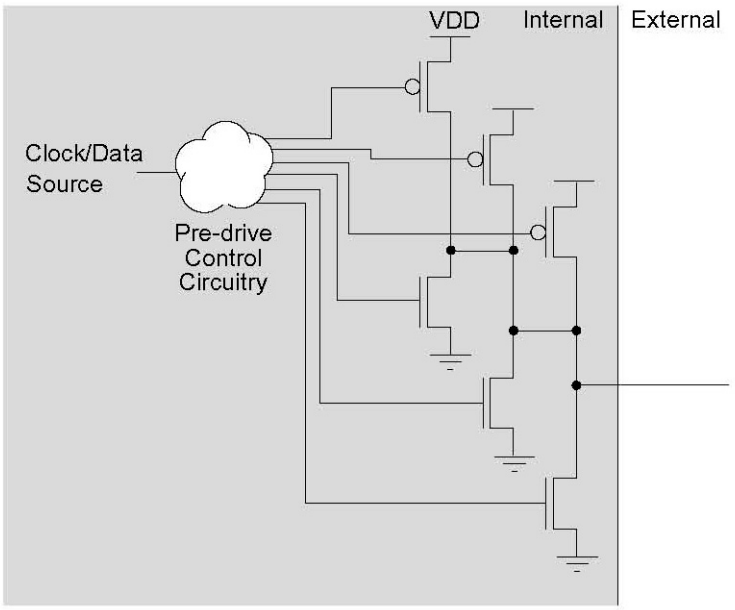


Figure 91 GTO Driver Topology

The function of such a driver is best explained by example. A GTO driver with three parallel drivers that turn on 3 ns after each other is demonstrated into 15 pF, 35 pF and 75 pF loads in Figure 92.

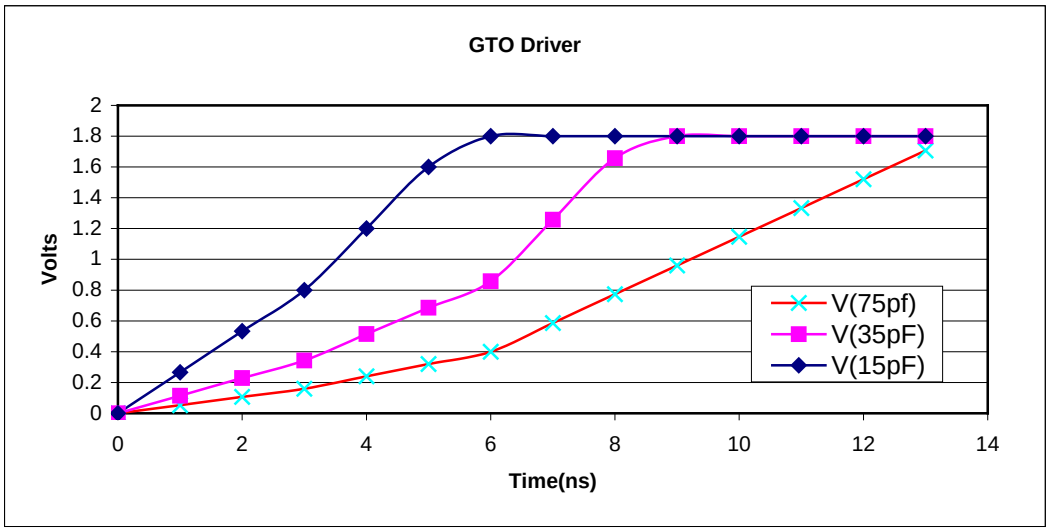


Figure 92 Response of GTO Driver into a Lumped Load

Here the three drivers are 4 mA, 2 mA and 8 mA. The subsequent output currents will be 4, 6 then 14 mA at 0 to 2 ns,3 to 5 ns and 6 ns until the output reaches opposite rail and the VDS across the output drivers reduces to zero. This creates three different regions; small loads will have completed the rise or fall before the last driver is enabled while large loads may require all three stages to become active. Ensuring that the maximum slew rate for the CLK line is not exceeded using GTO techniques alone is difficult, however his technique can be improved with a small amount of feedback, either analog (nested miller cap) or some digital feedback to the pre-drive control circuit. The order in which the output FETs are enabled can also be varied to allow the slope to be controlled throughout the transition.

B.3.3 Transition Time Locked Driver

In some situations where the CLK line must be very tightly controlled, a time-based reference can be introduced. For the CLK line driver this is convenient, the transition time can be measured and compared to the clock period.

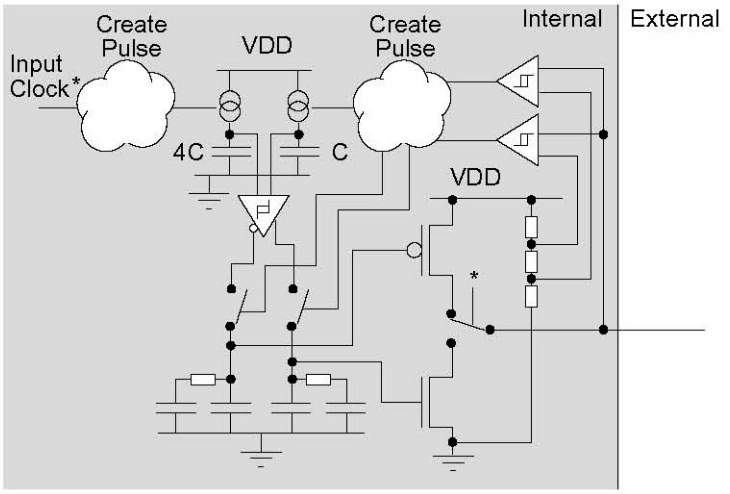


Figure 93 Transition Time Locked Driver Example

In the example circuit a pair of comparators is set to trigger at 20% and 80% of VDD. The output of the comparators feeds some decode logic that produces three outputs, one that is high during the rising transition time, one that indicates the falling transition time and the OR of both signals.

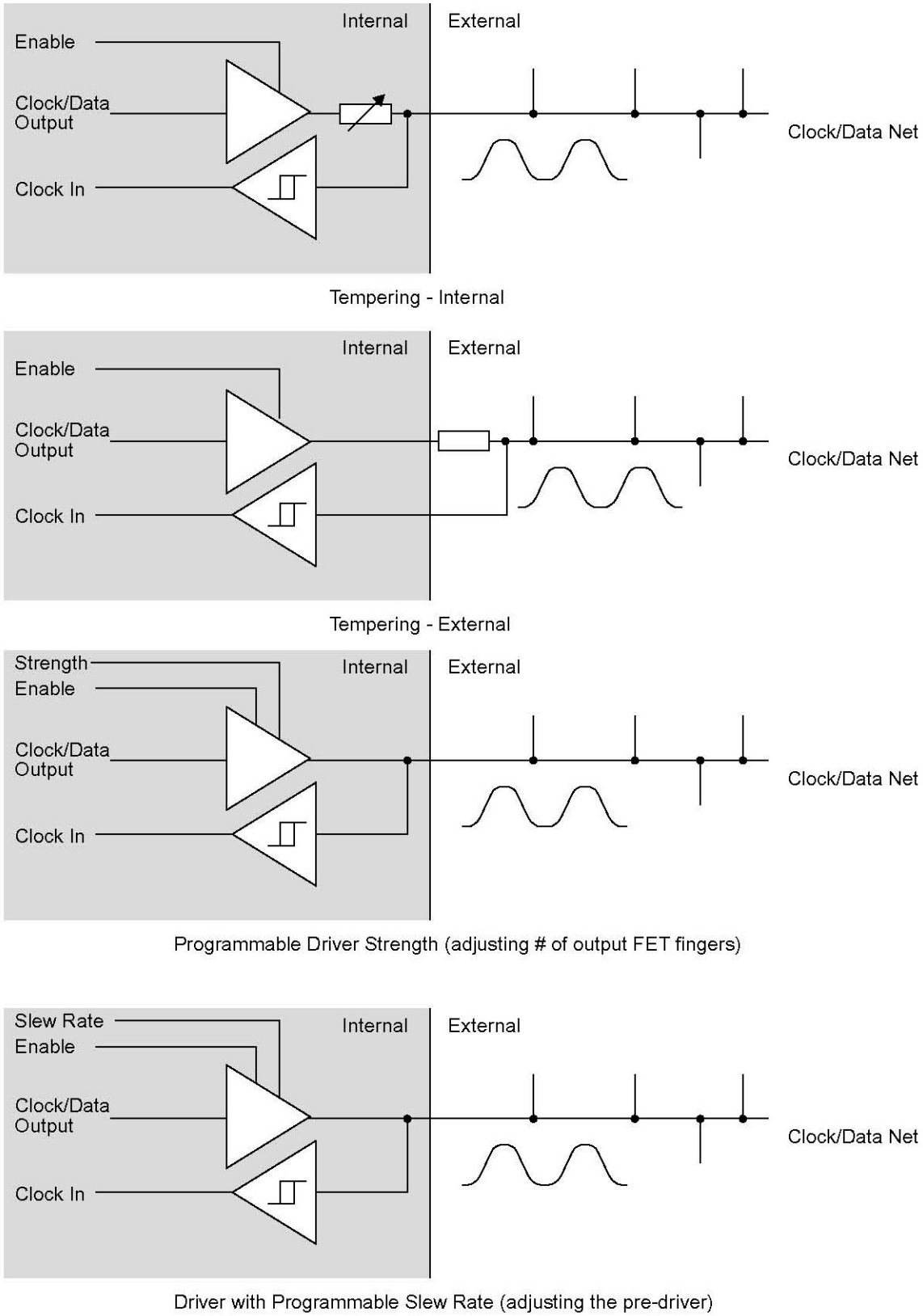
The combined signal enables a current source, which charges a cap. The reference input, the source clock is gated and the pulses are used to charge a second capacitor with a matched current source. The ratios of the capacitors are adjusted to select the desired ratio of clock period to transition time.

Every transition time a comparison is made between the voltages on the two capacitors, if the transition time was found to be too small the filtered bias voltage is reduced to compensate.

Such a feedback loop can be combined with the previous two mentioned techniques. With slightly more circuitry it also allows any induced or intrinsic jitter to be noise shaped away from both converter over-sampling frequencies and the audio band.

B.4 Notes on Implementing SLIMbus using CMOS IOs

In situations where SLIMbus must be implemented using conventional (non-slew rate controlled) CMOS IO structures certain compromises must be made.



5003

5004 **Figure 94 SLIMbus IO Examples using Traditional CMOS IO Driver Topologies**

The IO drivers should have the correct output impedance to drive the line while always meeting the requirements and constraints in Section 5. These IO drivers may be controlled outside of SLIMbus using knowledge of the load in order to meet the specification. This side-band knowledge eliminates the largest variable. Several methods of obtaining such control are as follows:

- The DATA input should be overvoltage protected to ensure transmission line reflections do not damage the device.
- If the Component includes a Framer then the maximum frequency of the CLK line driver will be limited. The maximum output slew rate must stay within the limits of the specification across all variations of PVT and loads. This may be achieved through the output impedance and/or through a slew rate limitation. Because of the high dependency on PVT and load of the CMOS IO, the transition times will be relatively long in some cases. The maximum operating frequency achievable into test loads must be quoted by the device manufacturer.
- As stated in Section 5, a Component can be SLIMbus-compliant even if it meets the slew rate specifications into the maximum load at only reduced CLK line frequencies.
- A Component may implement a single terminal using multiple physical pins. This enables the use of external Components with tighter tolerances than is possible using standard IOs. If SLIMbus is to be implemented as a muxed option behind legacy buses such as I^2S then such pins may become available.
- Hysteresis on the CLK line is required. Given the possibly reduced CLK line driver strength much care should be taken to ensure the CLK line seen by the active Manager does not glitch. CMOS IOs for clock reception that include hysteresis are not unusual.

Annex C Natural Frequencies (informative)

SLIMbus defines three families of Presence Rates in Table 62. The specified Presence Rates are power-of-two multiples of 4k, 11.025k, or 12k. Isochronous Data Channels based on these Presence Rates are supported by configuring the SLIMbus clock to use an appropriate Natural Frequency. Refer to Section 6.2.5 for more details on Natural Frequencies.

Table 87 shows the Natural Frequencies across all Clock Gears for the 4k family of Presence Rates. Note that some combinations of Presence Rate and Clock Gear which can be supported with either of two Natural Frequencies. One of these Natural Frequencies is derived from the 24.576 MHz Root Frequency and is the Cardinal Frequency of the associated Clock Gear. The other Natural Frequency is derived from 16.384 MHz which has no entry in Table 17 and thus must be supported with the "Not Indicated" code. Dashed lines indicate Presence Rates which cannot be supported in the indicated Clock Gear.

Table 87 Natural Frequencies for 4k Family of Flows

Clock Gear	Natural Frequency (MHz) as a Function of Presence Rate							
	4k	8k	16k	32k	64k	128k	256k	512k
10	24.576	24.576	24.576	24.576	24.576	24.576	24.576	24.576
		16.384	16.384	16.384	16.384	16.384	16.384	16.384
9	12.288	12.288	12.288	12.288	12.288	12.288	12.288	12.288
	8.192	8.192	8.192	8.192	8.192	8.192	8.192	8.192
8	6.144	6.144	6.144	6.144	6.144	6.144	6.144	6.144
	4.096	4.096	4.096	4.096	4.096	4.096	4.096	4.096
7	3.072	3.072	3.072	3.072	3.072	3.072	3.072	—
	2.048	2.048	2.048	2.048	2.048	2.048	2.048	
6	1.536	1.536	1.536	1.536	1.536	1.536	—	—
	1.024	1.024	1.024	1.024	1.024	1.024		
5	0.768	0.768	0.768	0.768	0.768	—	—	—
	0.512	0.512	0.512	0.512	0.512			
4	0.384	0.384	0.384	0.384	—	—	—	—
	0.256	0.256	0.256	0.256				
3	0.192	0.192	0.192	—	—	—	—	—
	0.128	0.128	0.128					
2	0.096	0.096	—	—	—	—	—	—
	0.064	0.064						
1	0.048	—	—	—	—	—	—	—
	0.032							

Table 88 shows the Natural Frequencies across all Clock Gears for the 11.025k family of Presence Rates. Note that some combinations of Presence Rate and Clock Gear which can be supported with either of two Natural Frequencies. One of these Natural Frequencies is derived from 22.5792 MHz which is a defined Root Frequency on SLIMbus. The other Natural Frequency is derived from 16.9344 MHz which has no entry in Table 17 and thus must be supported with the "Not Indicated" code. Dashed lines indicate Presence Rates which cannot be supported in the indicated Clock Gear.

5044 **Table 88 Natural Frequencies for 11.025k Family of Flows**

Clock Gear	Natural Frequency (MHz) as a Function of Presence Rate						
	11.025k	22.05k	44.1k	88.2k	176.4k	352.8k	705.6k
10	22.5792	22.5792	22.5792	22.5792	22.5792	22.5792	22.5792
	16.9344	16.9344	16.9344	16.9344	16.9344	16.9344	16.9344
9	11.2896	11.2896	11.2896	11.2896	11.2896	11.2896	11.2896
	8.4672	8.4672	8.4672	8.4672	8.4672	8.4672	8.4672
8	5.6448	5.6448	5.6448	5.6448	5.6448	5.6448	5.6448
	4.2336	4.2336	4.2336	4.2336	4.2336	4.2336	
7	2.8224	2.8224	2.8224	2.8224	2.8224	2.8224	—
	2.1168	2.1168	2.1168	2.1168	2.1168		
6	1.4112	1.4112	1.4112	1.4112	1.4112	—	—
	1.0584	1.0584	1.0584	1.0584			
5	0.7056	0.7056	0.7056	0.7056	—	—	—
	0.5292	0.5292	0.5292				
4	0.3528	0.3528	0.3528	—	—	—	—
	0.2646	0.2646					
3	0.1764	0.1764	—	—	—	—	—
	0.1323						
2	0.0882	—	—	—	—	—	—
1	—	—	—	—	—	—	—

5045 Table 89 shows the Natural Frequencies across all Clock Gears for the 12k family of Presence Rates. Note
5046 that some combinations of Presence Rate and Clock Gear which can be supported with either of two
5047 Natural Frequencies. One of these Natural Frequencies is derived from the 24.576 MHz Root Frequency
5048 and is the Cardinal Frequency of the associated Clock Gear. The other Natural Frequency is derived from
5049 18.432 MHz which has no entry in Table 17 and thus must be supported with the “Not Indicated” code.
5050 Dashed lines indicate Presence Rates which cannot be supported in the indicated Clock Gear.

5051 **Table 89 Natural Frequencies for 12k Family of Flows**

Clock Gear	Natural Frequency (MHz) as a Function of Presence Rate						
	12k	24k	48k	96k	192k	384k	768k
10	24.576	24.576	24.576	24.576	24.576	24.576	24.576
	18.432	18.432	18.432	18.432	18.432	18.432	18.432
9	12.288	12.288	12.288	12.288	12.288	12.288	12.288
	9.216	9.216	9.216	9.216	9.216	9.216	9.216
8	6.144	6.144	6.144	6.144	6.144	6.144	6.144
	4.608	4.608	4.608	4.608	4.608	4.608	
7	3.072	3.072	3.072	3.072	3.072	3.072	
	2.304	2.304	2.304	2.304	2.304		

Clock Gear	Natural Frequency (MHz) as a Function of Presence Rate						
	12k	24k	48k	96k	192k	384k	768k
6	1.536 1.152	1.536 1.152	1.536 1.152	1.536 1.152	1.536	—	—
5	0.768 0.576	0.768 0.576	0.768 0.576	0.768	—	—	—
4	0.384 0.288	0.384 0.288	0.384	—	—	—	—
3	0.192 0.144	0.192	—	—	—	—	—
2	0.096	—	—	—	—	—	—
1	—	—	—	—	—	—	—

Table 87 and Table 89 allow for some interesting observations. First, the Cardinal Frequency appears a Natural Frequency for both the 4k and the 12k Presence Rate families. Second, the 4k and 12k Presence Rates also have unique sets of Natural Frequencies, e.g. 9.216 MHz versus 8.192 MHz in Gear 9.

Annex D Determining *NCo* and *NCi* (informative)

NCo and ***NCi*** are computed to allow for at least one clock cycle during each phase of the handover mechanism. This additional time covers any uncertainties on the rise and fall times and on the clock frequency knowledge.

The first example assumes that the incoming and outgoing Framers are clocking the bus with the same frequency. The results are then generalized to different frequencies.

If the bus frequency does not change, the incoming Framer must wait at least two full clock cycles after the Reconfiguration Boundary before driving the CLK line HIGH. As the incoming Framer clock and the outgoing Framer clock are not necessarily phased locked, it takes two to three clock cycles for the incoming Framer to start driving the CLK line HIGH. The period starting at the Reconfiguration Boundary and ending at the positive edge of the Incoming Framer's internal clock is called "wait time"

The overlap region, the period when both Framers are driving the CLK line HIGH, must last at least one clock cycle. Therefore, the outgoing Framer drives the CLK line to HIGH during at least four clock cycles (up to one clock cycle for the wait time, two clock cycles while the outgoing Framer is the only one driving the CLK line HIGH and one clock cycle while both Framers are driving the CLK line HIGH).

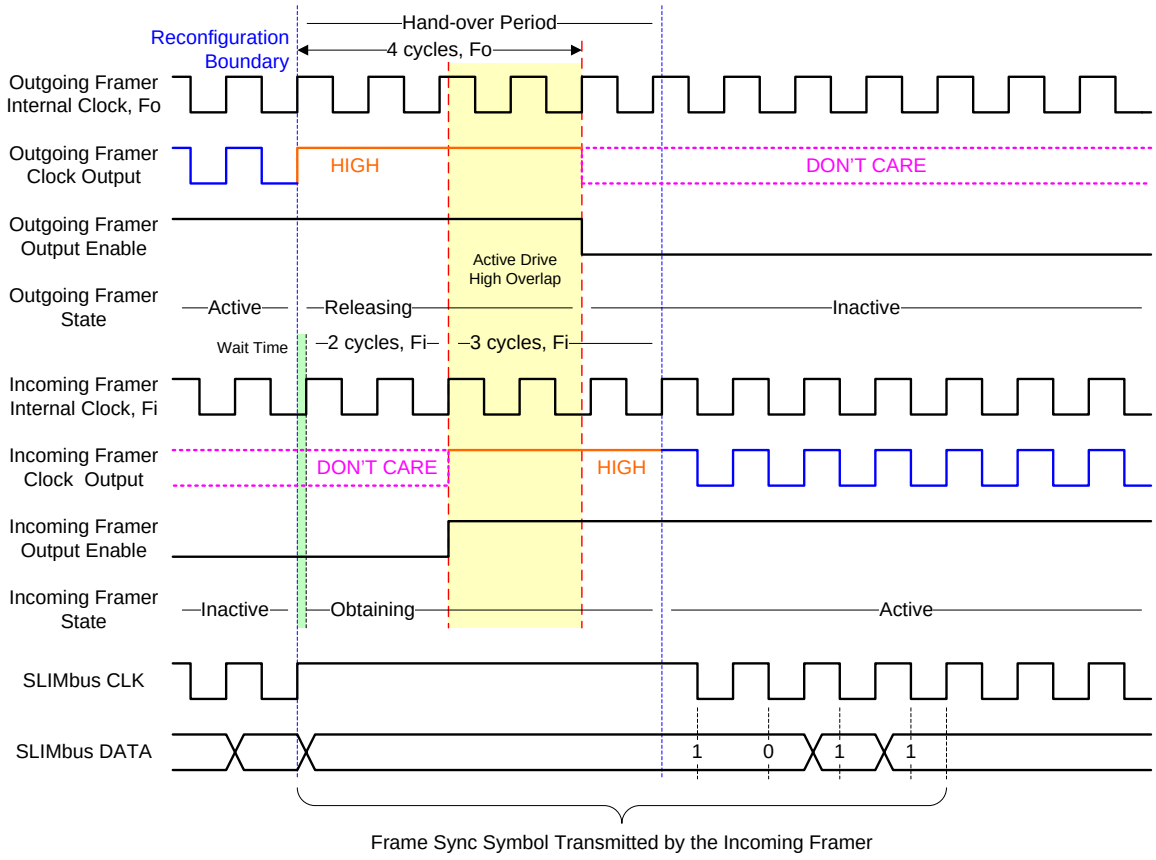


Figure 95 Minimal Wait Time and Maximal Overlap Period

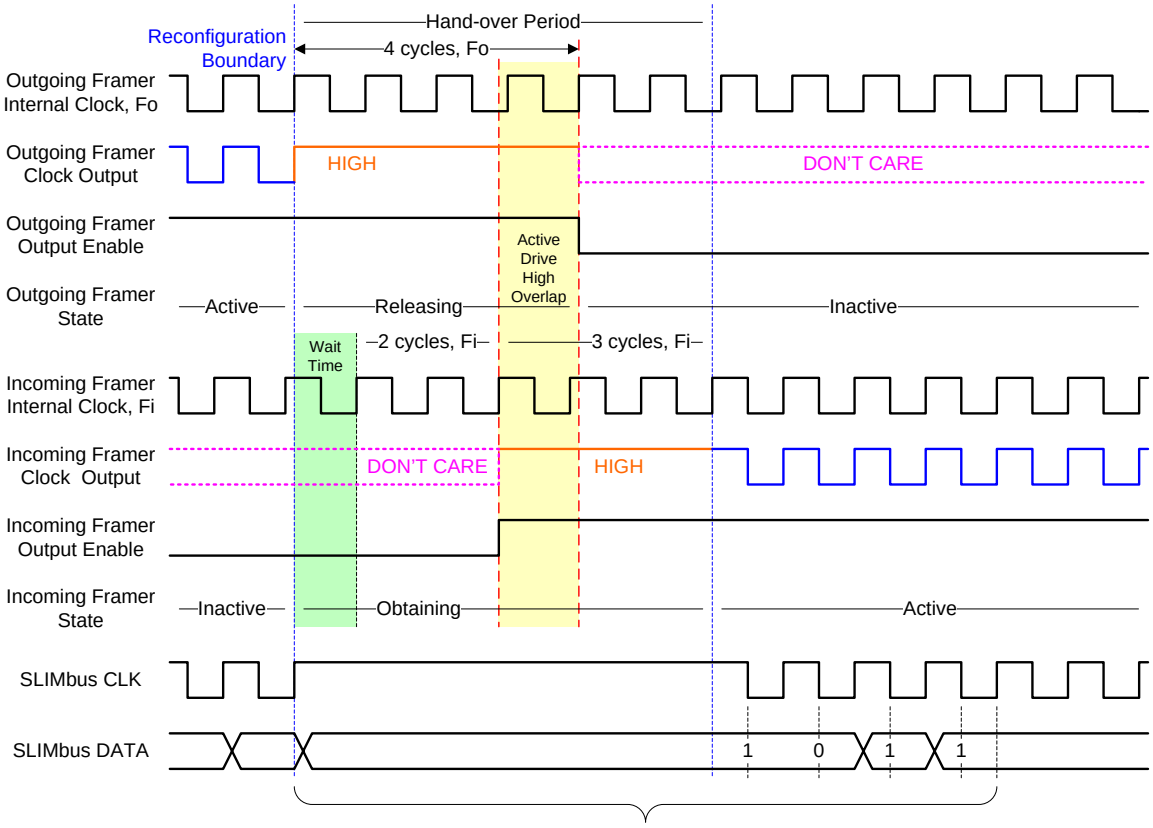


Figure 96 Maximal Wait Time and Minimal Overlap Period

Because of the wait time, the overlap period can last from one to two clock cycles. As the time the incoming Framer spends driving the CLK line alone must be at least one clock cycle, the incoming Framer must spend at least three clock cycles (the maximum overlap period plus one clock cycle) driving the CLK line HIGH.

When the Framer frequencies are different, the three and four clock cycle periods are replaced by ***NCo*** and ***NCi***, respectively. Both ***NCo*** and ***NCi*** depend on the ratio between the outgoing and incoming Framer clock frequencies. There are two cases to consider, Fi > Fo and Fi < Fo.

D.1 Fi > Fo

The incoming Framer clock frequency, Fi, is higher than the outgoing Framer clock frequency, Fo, (Fi>Fo). The constraints of having at least one clock cycle of the incoming Framer after the overlap period defines the relation between ***NCi*** and the clock period of the incoming Framer, Ti, and the clock period of the outgoing Framer, To. ***NCi****Ti + 2*Ti +Hold Time ≥ 4*To + Ti. As the hold time can be as small as 0, ***NCi*** is defined by ***NCi*** ≥ 4*To/Ti - 1. Therefore, the smallest possible value is ***NCi*** = **INT**(4*To/Ti).

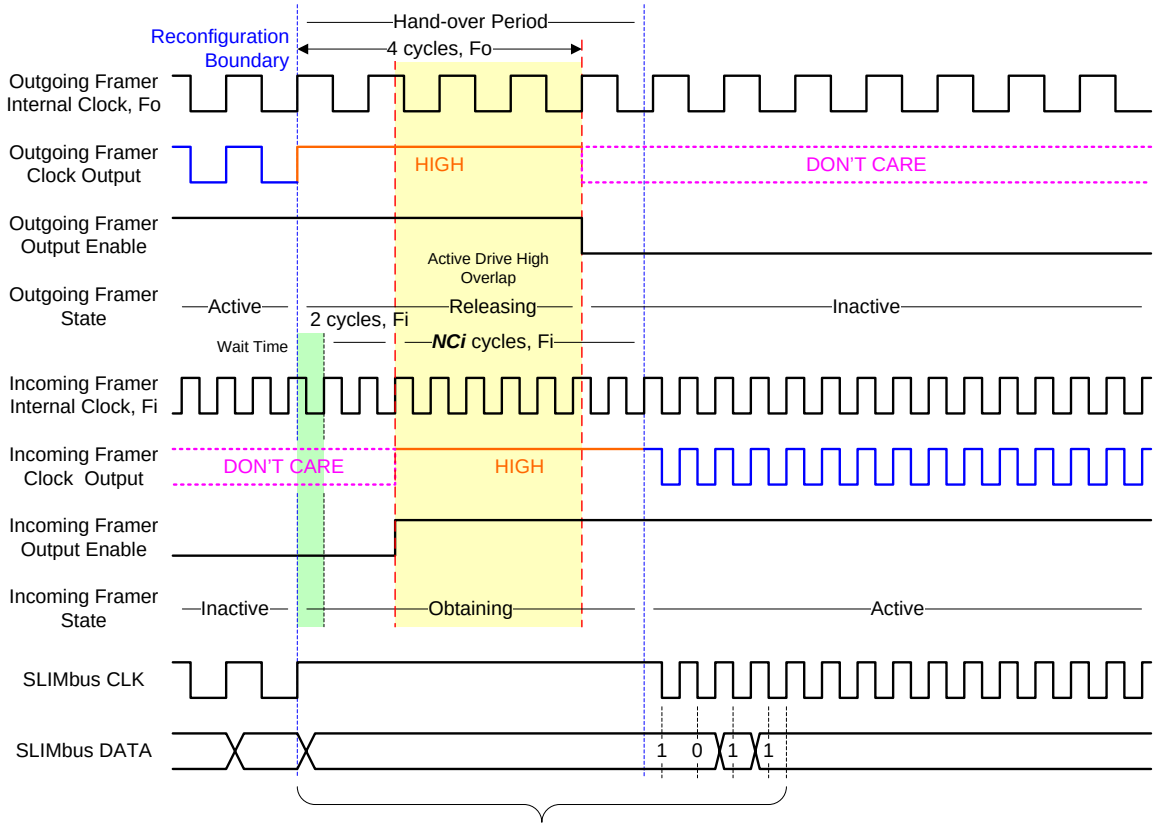


Figure 97 Determining *NCi*

D.2 Fi < Fo

The incoming Framer clock frequency is lower than the outgoing Framer clock frequency (Fi<Fo). The constraints of having at least one clock cycle of every clock in the overlap period defines the relation between ***NCo*** and (Ti, To). ***NCo****To ≥ Hold Time + 2*Ti + Ti. As the hold time can be as big as Ti, ***NCo*** is defined by ***NCo****To ≥ 4*Ti. Therefore, the smallest possible value is ***NCo*** = **INT**(4*Ti/To)+1.

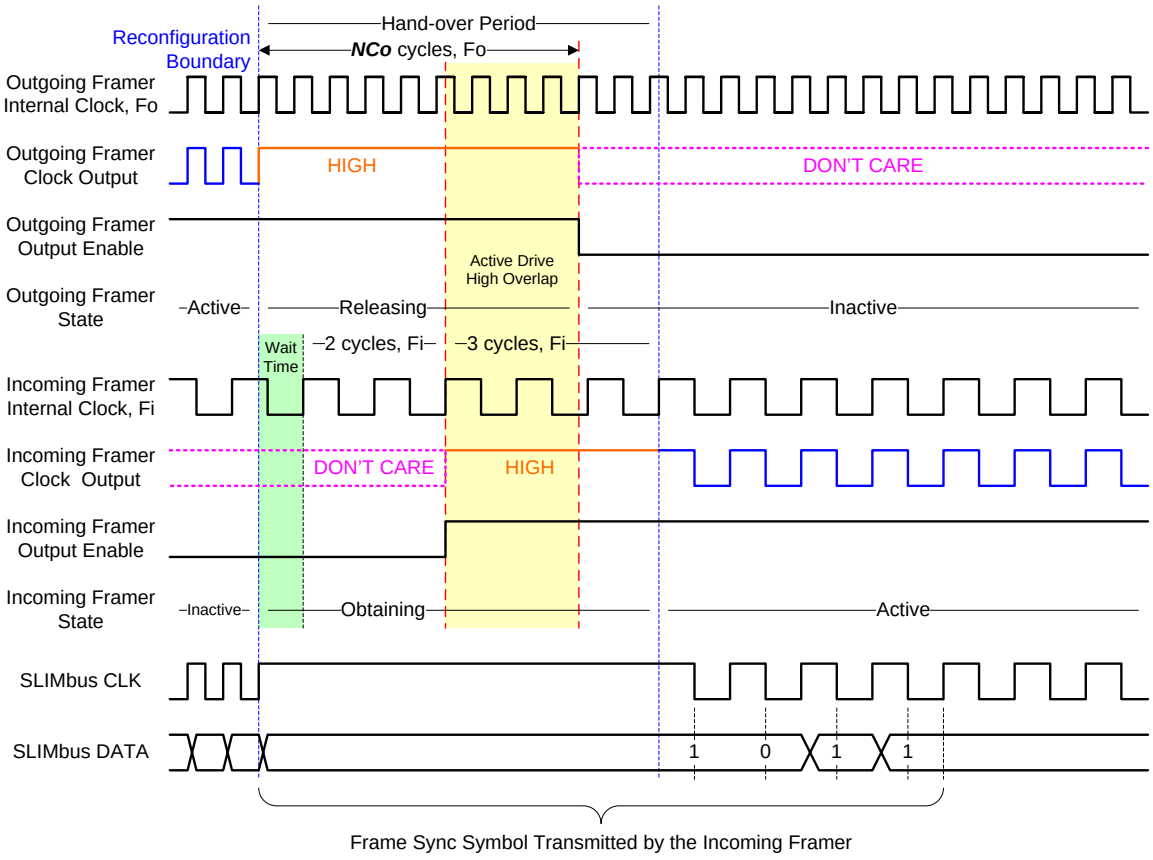


Figure 98 Determining *NCo*

Contributors

The following companies have contributed to the development of this document.

Analog Devices, Inc.

austriamicrosystems AG

Ericsson AB

Freescale Semiconductor

Infineon Technologies AG

Intel Corporation

LnK

Marvell Semiconductor

Motorola

National Semiconductor

Nokia Corporation

NXP Semiconductors

RF Micro Devices

Sonion

Sony Ericsson Mobile Communications

STMicroelectronics N.V.

Synopsys, Inc

Texas Instruments Incorporated

Toshiba Corporation

Wolfson Microelectronics