# Media Local Bus Specification

## Version 4.2

This document defines the Media Local Bus (MediaLB®) Physical Layer and Link Layer.

MediaLB is an on-PCB or inter-chip communication bus, specifically designed to standardize a common hardware interface and software API library. This standardization allows an application or multiple applications to access the MOST® (*Media Oriented Systems Transport*) Network data, or to communicate with other applications, with minimum effort. MediaLB is designed to support all MOST Networks; thereby, providing a simple migration path between different MOST Network architectures. MediaLB also supports all MOST Network data transport methods: synchronous stream data, asynchronous packet data, control message data, and isochronous data.

# Further Information

For more information on SMSC automotive products, including integrated circuits, software, and MOST development tools and modules, visit our web site: http://www.smsc-ais.com. Direct contact information is available at: http://www.smsc-ais.com/offices.

**SMSC Europe GmbH**
Bannwaldallee 48
D-76185 Karlsruhe
Germany

**SMSC**
80 Arkay Drive
Hauppauge, New York 11788
USA

# Technical Support

Contact information for technical support is available at: http://www.smsc-ais.com/contact.

# Legend

Ensure that all information within a document marked as 'Confidential' or 'Confidential Controlled Access' is handled solely in accordance with the agreement pursuant to which it is provided, and is not reproduced or disclosed to others without the prior written consent of SMSC. The confidential ranking of a document can be found in the footer of every page. This document supersedes and replaces all information previously supplied. The technical information in this document loses its validity with the next edition. Although the information is believed to be accurate, no responsibility is assumed for inaccuracies. Specifications and other documents mentioned in this document are subject to change without notice. SMSC reserves the right to make changes to this document and to the products at any time without notice. Neither the provision of this information nor the sale of the described products conveys any licenses under any patent rights or other intellectual property rights of SMSC or others. There are a number of patents and patents pending on the MOST technology and other technologies. No rights under these patents are conveyed without any specific agreement between the users and the patent owners. The products may contain design defects or errors known as anomalies, including but not necessarily limited to any which may be identified in this document, which may cause the product to deviate from published descriptions. Anomalies are described in errata sheets available upon request. SMSC products are not designed, intended, authorized or warranted for use in any life support or other application where product failure could cause or contribute to personal injury or severe property damage. Any and all such uses without prior written approval of an officer of SMSC will be fully at your own risk. TrueAuto is a trademark and MediaLB, SMSC and MOST are registered trademarks of Standard Microsystems Corporation (“SMSC”). Other names mentioned may be trademarks of their respective holders.

**SMSC DISCLAIMS AND EXCLUDES ANY AND ALL WARRANTIES, INCLUDING WITHOUT LIMITATION ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND AGAINST INFRINGEMENT AND THE LIKE, AND ANY AND ALL WARRANTIES ARISING FROM ANY COURSE OF DEALING OR USAGE OF TRADE. IN NO EVENT SHALL SMSC BE LIABLE FOR ANY DIRECT, INCIDENTAL, INDIRECT, SPECIAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES; OR FOR LOST DATA, PROFITS, SAVINGS OR REVENUES OF ANY KIND; REGARDLESS OF THE FORM OF ACTION, WHETHER BASED ON CONTRACT; TORT; NEGLIGENCE OF SMSC OR OTHERS; STRICT LIABILITY; BREACH OF WARRANTY; OR OTHERWISE; WHETHER OR NOT ANY REMEDY OF BUYER IS HELD TO HAVE FAILED OF ITS ESSENTIAL PURPOSE, AND WHETHER OR NOT SMSC HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.**

# Conventions

Within this manual, the following abbreviations and symbols are used to improve readability.

| Example | Description |
|---|---|
| **BIT** | Name of a single bit within a field |
| **FIELD.BIT** | Name of a single bit (BIT) in FIELD |
| x…y | Range from x to y, inclusive |
| **BITS[m:n]** | Groups of bits from m to n, inclusive |
| **PIN** | Pin Name |
| msb, lsb | Most significant bit, least significant bit |
| MSB, LSB | Most significant byte, least significant byte |
| zzzzb | Binary number (value zzzz) |
| 0xzzz | Hexadecimal number (value zzz) |
| zzh | Hexadecimal number (value zz) |
| rsvd | Reserved memory location. Must write 0, read value indeterminate |
| `code` | Instruction code, or API function or parameter |
| *Multi Word Name* | Used for multiple words that are considered a single unit, such as:<br>*Resource Allocate* message, or *Connection Label*, or *Decrement Stack Pointer* instruction. |
| *Section Name* | Section or Document name. |
| $\overline{\text{VAL}}$ | Over-bar indicates active low pin or register bit |
| x | Don't care |
| <Parameter> | <> indicate a Parameter is optional or is only used under some conditions |
| {,Parameter} | Braces indicate Parameter(s) that repeat one or more times. |
| [Parameter] | Brackets indicate a nested Parameter. This Parameter is not real and actually decodes into one or more real parameters. |

# MediaLB Definition of Terms

The following terms will be used when referring to specific implementations of MediaLB.

| Names | Description |
|---|---|
| ***Media Local Bus:*** | |
| *MLBC* | General reference to the Clock line of a Media Local Bus:<br>on a MediaLB 3-pin interface, connects to the **MLBCLK** pin<br>on a MediaLB 5-pin interface, connects to the **MLBCLK** pin<br>on a MediaLB 6-pin interface, connects to **MLBCP** and **MLBCN** pins |
| *MLBS* | General reference to the Signal line of a Media Local Bus:<br>on a MediaLB 3-pin interface, connects to the **MLBSIG** pin<br>on a MediaLB 5-pin interface, connects to the **MLBSO** and **MLBSI** pins<br>on a MediaLB 6-pin interface, connects to the **MLBSP** and **MLBSN** pins |
| *MLBD* | General reference to the Data line of a Media Local Bus:<br>on a MediaLB 3-pin interface, connects to the **MLBDAT** pin<br>on a MediaLB 5-pin interface, connects to the **MLBDO** and **MLBDI** pins<br>on a MediaLB 6-pin interface, connects to the **MLBDP** and **MLBDN** pins |
| ***MediaLB 3-pin Interface:*** | |
| **MLBCLK** | MediaLB Controller (*output*) pin connected to *MLBC*.<br>MediaLB Device (*input*) pin connected to *MLBC*. |
| **MLBSIG** | MediaLB Device (*I/O*) pin connected to *MLBS*. |
| **MLBDAT** | MediaLB Device (*I/O*) pin connected to *MLBD*. |
| ***MediaLB 5-pin Interface:*** | |
| **MLBCLK** | MediaLB Controller (*output*) pin connected to *MLBC*.<br>MediaLB Device (*input*) pin connected to *MLBC*. |
| **MLBSI** | MediaLB Device (*input*) pin connected to *MLBS* for *reception*. |
| **MLBSO** | MediaLB Device (*output*) pin connected to *MLBS* for *transmission*. |
| **MLBDI** | MediaLB Device (*input*) pin connected to *MLBD* for *reception*. |
| **MLBDO** | MediaLB Device (*output*) pin connected to *MLBD* for *transmission*. |
| ***MediaLB 6-pin Interface:*** | |
| **MLBCP** | MediaLB Controller (*output*) pin connected to the *positive* terminal of *MLBC*.<br>MediaLB Device (*input*) pin connected to the *positive* terminal of *MLBC*. |
| **MLBCN** | MediaLB Controller (*output*) pin connected to the *negative* terminal of *MLBC*.<br>MediaLB Device (*input*) pin connected to the *negative* terminal of *MLBC*. |
| **MLBSP** | MediaLB Device (*I/O*) pin connected to the *positive* terminal of *MLBS*. |
| **MLBSN** | MediaLB Device (*I/O*) pin connected to the *negative* terminal of *MLBS*. |
| **MLBDP** | MediaLB Device (*I/O*) pin connected to the *positive* terminal of *MLBD*. |
| **MLBDN** | MediaLB Device (*I/O*) pin connected to the *negative* terminal of *MLBD*. |

| Names | Description |
|---|---|
| ***Differential Voltage Definitions:*** | |
| $V_{O+}$ | *Output* voltage on **MLBCP**, **MLBSP**, or **MLBDP**. |
| $V_{O-}$ | *Output* voltage on **MLBCN**, **MLBSN**, or **MLBDN**. |
| $V_{IN+}$ | *Input* voltage on **MLBCP**, **MLBSP**, or **MLBDP**. |
| $V_{IN-}$ | *Input* voltage on **MLBCN**, **MLBSN**, or **MLBDN**. |
| $V_{OD}$ | Differential *output* voltage on *MLBC*, *MLBS*, or *MLBD*, where:<br>$V_{OD}$ on *MLBC* is equal to \| $V_{O+}$(**MLBCP**) - $V_{O-}$(**MLBCN**) \|<br>$V_{OD}$ on *MLBS* is equal to \| $V_{O+}$(**MLBSP**) - $V_{O-}$(**MLBSN**) \|<br>$V_{OD}$ on *MLBD* is equal to \| $V_{O+}$(**MLBDP**) - $V_{O-}$(**MLBDN**) \| |
| $V_{ID}$ | Differential *input* voltage on *MLBC*, *MLBS*, or *MLBD*, where:<br>$V_{ID}$ on *MLBC* is equal to ( $V_{IN+}$(**MLBCP**) - $V_{IN-}$(**MLBCN**) )<br>$V_{ID}$ on *MLBS* is equal to ( $V_{IN+}$(**MLBSP**) - $V_{IN-}$(**MLBSN**) )<br>$V_{ID}$ on *MLBD* is equal to ( $V_{IN+}$(**MLBDP**) - $V_{IN-}$(**MLBDN**) ) |
| $V_{OCM}$ | Common-mode *output* voltage on *MLBC*, *MLBS*, or *MLBD*, where:<br>$V_{OCM}$ on *MLBC* is equal to \| $V_{O+}$(**MLBCP**) + $V_{O-}$(**MLBCN**) \| / 2<br>$V_{OCM}$ on *MLBS* is equal to \| $V_{O+}$(**MLBSP**) + $V_{O-}$(**MLBSN**) \| / 2<br>$V_{OCM}$ on *MLBD* is equal to \| $V_{O+}$(**MLBDP**) + $V_{O-}$(**MLBDN**) \| / 2 |

# TABLE OF CONTENTS

# 1 Overview

The objective of Media Local Bus (MediaLB) is to map all the MOST Network data types (transport methods) into a single low-cost, scalable, and standardized hardware interface between a MediaLB Controller and at least one other MediaLB Device. The adoption of MediaLB simplifies the hardware interface, reduces the pin count, and facilitates the design of modular reusable hardware. From a software development perspective, the use of MediaLB relieves the system developer from the complexity of the MOST Network, which simplifies software development and enables the design of reusable software for different applications. This simplified, standardized interface shortens time-to-market and makes software maintenance effortless.

The link layer and three different physical layers are defined as part of this specification. The physical layer section describes pin configurations, operating speeds, and bus topology. The link layer section describes the compliance of the signaling and addressing protocol.

## 1.1 System Features

- Fully synchronous architecture
- A frame synchronization pattern (*FRAMESYNC*) enables easy Device synchronization to MOST Networks
- Multiple clock rates supported
- Scalable data rate for all MOST Network data transport methods
- Support of all MOST data transport methods: synchronous stream data, asynchronous packet data, control message data, and isochronous data
- Dedicated system-broadcast channel for administration
- Support of MediaLB Controller to MediaLB Device transfers and inter-MediaLB Devices transfers
- Broadcast support from one transmitter to multiple receivers for synchronous stream data
- Designed to ease hardware and software application development for MOST Networks

## 1.2 MediaLB Concept

The MediaLB topology supports communication among all MediaLB Devices, including the MediaLB Controller. The bus interface consists of a uni-directional line for clock (*MLBC*), a bi-directional line for signal information (*MLBS*), and a bi-directional line for data transfer (*MLBD*).

The MediaLB topology supports one Controller connected to one or more Devices, where the Controller is the interface between the MediaLB Devices and the MOST Network. The MediaLB Controller includes MediaLB Device functionality, and also generates the MediaLB clock (*MLBC*) that is synchronized to the MOST Network. This generated clock provides the timing for the entire MediaLB interface. The Controller will continue to generate *MLBC* even when the Controller loses lock with the MOST Network.

The *MLBS* line is a multiplexed signal which carries *ChannelAddresses* generated by the MediaLB Controller, as well as *Command* and *RxStatus* bytes from MediaLB Devices. Each *ChannelAddress* indicates which Device can transmit data and which Device (or Devices) can receive data on a particular logical channel.

The *MLBD* line is driven by the transmitting MediaLB Device and is received by all other MediaLB Devices, including the MediaLB Controller. The *MLBD* line carries the actual data (synchronous, asynchronous, control, or isochronous). For synchronous stream data transmission, multiple MediaLB Devices can receive the same data, in a broadcast fashion. The transmitting MediaLB Device indicates the particular type of data transmitted by sending the appropriate command on the *MLBS* line. The *Link Layer* section defines the different commands supported.

## 1.3 MediaLB Protocol

Once per MOST Network frame, the MediaLB Controller generates a unique *FRAMESYNC* pattern on the *MLBS* line. For all Devices on the bus, the end of the *FRAMESYNC* pattern defines the byte boundary and the channel boundary for the *MLBS* and *MLBD* lines.

Each four byte wide block (quadlet) in a MediaLB 3/5-pin frame is defined as a *physical channel*. For MediaLB 6-pin, the *physical channel* is larger than four bytes due to the addition of turnaround cycles. *Physical channels* can be grouped into multiple quadlets (which do not have to be consecutive) to form a *logical channel*. The MediaLB Controller handles channel arbitration, allocates channel bandwidth for MediaLB Devices, and manages the unique *ChannelAddresses* for referencing logical channels.

The MediaLB Controller initiates communication with MediaLB Devices by sending an assigned *ChannelAddress* on *MLBS* in each logical channel. This *ChannelAddress* indicates which MediaLB Device will transmit data and which MediaLB Devices will receive data in the following logical channel.

One *physical channel* after the *ChannelAddress* is sent on *MLBS*, the transmitting MediaLB Device associated with that *ChannelAddress* outputs a command byte (*Command*) on *MLBS* and respective data (*Data*) on *MLBD*, concurrently. The *Command* byte contains information about the data simultaneously being transmitted. The MediaLB Device receiving the data outputs a status byte (*RxStatus*) on *MLBS* after the transmitting Device sends the *Command* byte. This status response can indicate that the Device is ready to receive the data, or that the receiving Device is busy (e.g. cannot receive the data at present). Since synchronous stream data is sent in a broadcast fashion, Devices receiving synchronous data can never return a busy status response. In this situation, the *RxStatus* byte must not be actively driven onto the *MLBS* line by Devices receiving synchronous data.

The *ChannelAddresses* output by the Controller for each logical channel are used in normal data transport and can be statically or dynamically assigned. To support dynamic configuration of MediaLB Devices, a unique *DeviceAddress* must be assigned to all MediaLB Devices before startup. *DeviceAddresses* allow the External Host Controller (EHC) and MediaLB Controller to dynamically determine which Devices exist on the bus. At the request of a MediaLB Device (e.g. EHC), the Controller scans for *DeviceAddresses* in the *System Channel*. Once a Device is detected, a *ChannelAddress* for each logical channel can be assigned.

The *DeviceAddress*, *ChannelAddress*, *Command*, and *RxStatus* structures are described in the *Link Layer* section.

# 2 Physical Layer

This MediaLB specification defines three different physical layers. The pin configurations differ between each physical layer, as shown in Table 2-1.

| MediaLB Signal Names | MediaLB Physical Interface | | |
|---|---|---|---|
| | 3-pin (single-ended) | 5-pin (single-ended) | 6-pin (differential) |
| *MLBC* | **MLBCLK** | **MLBCLK** | **MLBCP** |
| | | | **MLBCN** |
| *MLBS* | **MLBSIG** | **MLBSI** | **MLBSP** |
| | | **MLBSO** | **MLBSN** |
| *MLBD* | **MLBDAT** | **MLBDI** | **MLBDP** |
| | | **MLBDO** | **MLBDN** |

**Table 2-1: MediaLB Physical Layer Pin Configurations**

MediaLB is based on an advanced scalable 3-pin interface consisting of the **MLBCLK**, **MLBSIG**, and **MLBDAT** pins. This interface is designed to support a maximum operating frequency of 1024×Fs (49.152 MHz at Fs = 48 kHz).

For legacy systems, a MediaLB 5-pin interface is supported. This interface splits out the *MLBS* and *MLBD* lines so that they are unidirectional. The *MLBS* line is comprised of the **MLBSI** pin for signal input and **MLBSO** for signal output; and the *MLBD* line becomes **MLBDI** for data input and **MLBDO** for data output. When more than two MediaLB Devices exist on the 5-pin bus, logic must exist to OR the output lines together. This interface is designed to support a maximum operating frequency of 512×Fs (24.576 MHz at Fs = 48 kHz).

By using differential data transmission, the MediaLB 6-pin interface supports higher speeds than MediaLB 3-pin. With a low voltage swing between each differential pair, MediaLB 6-pin is designed to support a maximum external clock operating frequency of 2048×Fs, with a MediaLB Device recovered clock speed up to 8192×Fs (4:1 recovered-to-external clock ratio). In recovered-to-external clock ratios such as 4:1, the higher speed recovered clock is used to generate and recover *MLBD* and *MLBS* information at a rate faster than *MLBC*. Clock recovery circuits at the MediaLB 6-pin interface will generate a higher speed recovered clock from the external bus clock on the *MLBC* differential line. In addition to higher speed, the use of differential signaling also eliminates the complexities associated with mixed voltage operation on application boards.

The available clock speeds for each MediaLB physical layer is given in Table 2-2.

| MediaLB Clock Line | MediaLB Physical Interface | | | |
|---|---|---|---|---|
| | 3-pin | 5-pin | 6-pin | |
| | Clock Speed | Clock Speed | Recovered-to-External Clock Ratio | Recovered Clock Speed |
| *MLBC* | 256×Fs<br>512×Fs<br>1024×Fs | 256×Fs<br>512×Fs | 1:1 | 2048×Fs |
| | | | 2:1 | 3072×Fs |
| | | | | 4096×Fs |
| | | | 4:1 | 6144×Fs |
| | | | | 8192×Fs |

**Table 2-2: MediaLB Physical Layer Operating Speeds**

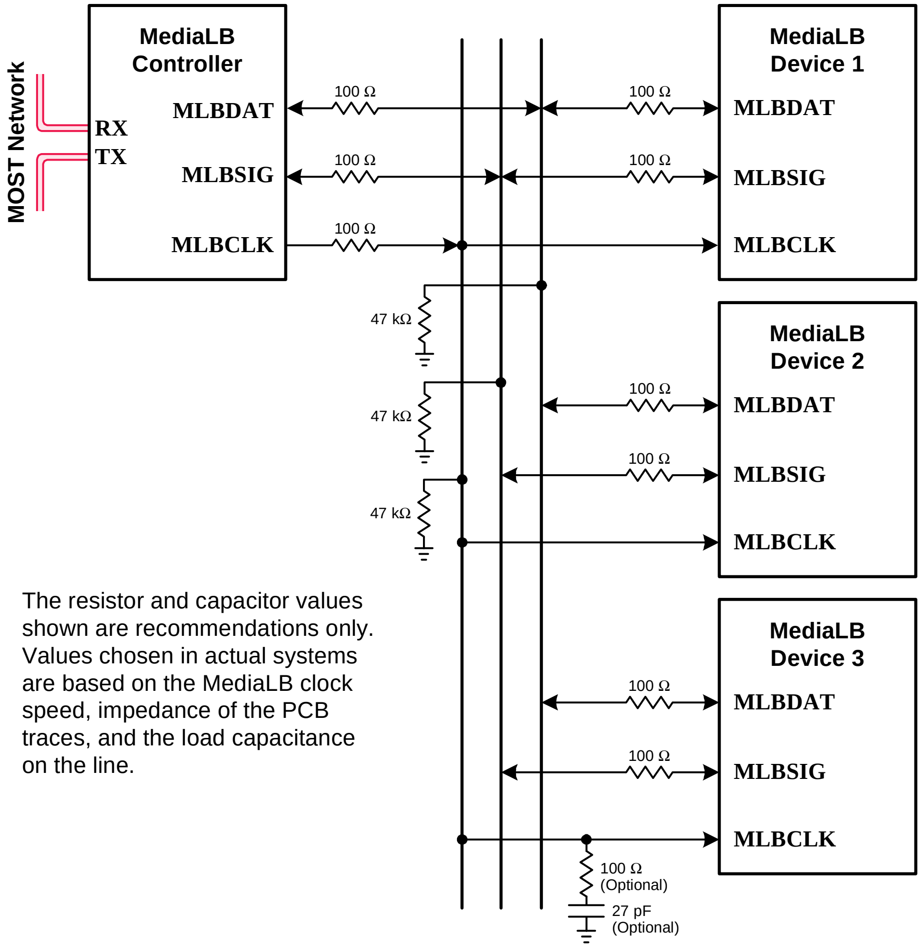
The MediaLB 5-pin interface is intended to support legacy designs only. New MediaLB designs should implement a 3-pin or 6-pin interface

## 2.1 MediaLB 3-pin Interface

### 2.1.1 Pin Description

The MediaLB system clock is generated by a single MediaLB Controller. The MediaLB Controller outputs the clock on the **MLBCLK** pin, which is connected to the clock input of all other MediaLB Devices in the system. All MediaLB Devices (including the MediaLB Controller), share the signals connected to the **MLBSIG** and **MLBDAT** pins.

Once per physical channel (quadlet) on the **MLBSIG** line, the Controller outputs the *ChannelAddress*, the transmitting Device outputs *Command*, and the receiving Device outputs *RxStatus*. Therefore, each Device must set **MLBSIG** high impedance when not driving in order to allow the other Devices to drive it. Once per physical channel, the transmitting Device must also drive data onto the **MLBDAT** line, and set the line to high impedance for physical channels not allocated to that particular Device. As illustrated in Figure 2-1, pull-down resistors are required on each signal to keep them in a known state when neither the Controller nor a Device is driving. Resistors are also recommended near the Controller and Device transmit lines for series termination and rise/fall time control. The clock line (**MLBCLK**) may optionally have AC-parallel termination near the farthest Device from the Controller to ensure a clean clock by minimizing reflections.



The resistor and capacitor values shown are recommendations only. Values chosen in actual systems are based on the MediaLB clock speed, impedance of the PCB traces, and the load capacitance on the line.

**Figure 2-1: MediaLB 3-pin Connection Diagram**

More information about MediaLB 3-pin terminations can be found in Appendix A.3.1.

### 2.1.2 Electrical Characteristics

Although the specifications below are listed for all MediaLB 3-pin clock speeds, individual chips or systems may choose to only implement specific speeds (e.g. 256×Fs, 512×Fs or 1024×Fs).

MediaLB Controllers configured as a timing-slave use the MOST Network as the PLL clocking source during normal operation; however, brief periods of unlock can occur. During these periods of network unlock, the PLL clocking source is switched to a local external crystal until the network relocks. The PLL is temporarily unlocked during these periods of switching between the network and the crystal. Specifications shown are applicable when the PLL is locked, unless otherwise specified. The electrical characteristics of MediaLB 3-pin given below are split into two categories; 256×Fs / 512×Fs and 1024×Fs.

### 2.1.2.1 MediaLB Clock Speed 256×Fs and 512×Fs

Ground = 0.0 V; Maximum load capacitance = 60 pF; Fs = 48 kHz; All timing parameters are specified from the valid voltage threshold as listed below; unless otherwise noted.

| Parameter | Symbol | Min | Typ | Max | Unit | Comment |
|---|---|---|---|---|---|---|
| Input voltage | | | | 3.6 | V | |
| Low-level input threshold | $V_{IL}$ | | | 0.7 | V | |
| High-level input threshold | $V_{IH}$ | 1.8 | | | V | Note 1 |
| Low-level output threshold | $V_{OL}$ | | | 0.4 | V | $I_{OL} = 6$ mA |
| High-level output threshold | $V_{OH}$ | 2.0 | | | V | $I_{OH} = -6$ mA |
| **MLBCLK** operating frequency (Note 2) | $f_{mck}$ | 11.264 | | <br>25.6 | MHz<br>MHz | 256×Fs at 44.0 kHz<br>512×Fs at 50.0 kHz |
| **MLBCLK** rise time | $t_{mckr}$ | | | 3 | ns | $V_{IL}$ to $V_{IH}$ |
| **MLBCLK** fall time | $t_{mckf}$ | | | 3 | ns | $V_{IH}$ to $V_{IL}$ |
| Input leakage current | $I_L$ | | | ±10 | µA | 0 < Vin < VDD |
| **MLBCLK** low time (Note 3) | $t_{mckl}$ | 30<br>14 | | | ns<br>ns | 256×Fs<br>512×Fs |
| **MLBCLK** high time (Notes 2, 3) | $t_{mckh}$ | 30<br>14 | | | ns<br>ns | 256×Fs<br>512×Fs |
| **MLBSIG/MLBDAT** receiver input valid to **MLBCLK** falling | $t_{dsmcf}$ | 1 | | | ns | |
| **MLBSIG/MLBDAT** receiver input hold from **MLBCLK** low | $t_{dhmcf}$ | $t_{mdzh}$ | | | ns | |
| **MLBSIG/MLBDAT** output high impedance from **MLBCLK** low | $t_{mcfdz}$ | 0 | | $t_{mckl}$ | ns | Note 4 |
| Bus hold from **MLBCLK** low | $t_{mdzh}$ | 4 | | | ns | Note 4 |

1. Higher $V_{IH}$ thresholds can be used; however, the risks associated with less noise margin in the system must be evaluated and assumed by the system implementor.
2. The Controller can shut off **MLBCLK** to place MediaLB in a low-power state. Depending on the time the clock is shut off, a runt pulse can occur on **MLBCLK**.
3. **MLBCLK** low and high times include pulse width variation.
4. The MediaLB driver can release the **MLBSIG/MLBDAT** line (e.g. high-impedance) as soon as **MLBCLK** is low; however, the logic state of the final driven bit on the line must remain on the bus for $t_{mdzh}$. Therefore, coupling must be minimized while meeting the maximum load capacitance listed.

### 2.1.2.2 MediaLB Clock Speed 1024×Fs

Ground = 0.0 V; Maximum load capacitance = 40 pF; Fs = 48 kHz; All timing parameters are specified from the valid voltage threshold as listed below; unless otherwise noted.

| Parameter | Symbol | Min | Typ | Max | Unit | Comment |
|---|---|---|---|---|---|---|
| Input voltage | | | | 3.6 | V | |
| Low-level input threshold | $V_{IL}$ | | | 0.7 | V | |
| High-level input threshold | $V_{IH}$ | 1.8 | | | V | Note 1 |
| Low-level output threshold | $V_{OL}$ | | | 0.4 | V | $I_{OL}$ = 6 mA |
| High-level output threshold | $V_{OH}$ | 2.0 | | | V | $I_{OH}$ = -6 mA |
| **MLBCLK** operating frequency (Note 2) | $f_{mck}$ | 45.056 | | 51.2 | MHz<br>MHz | 1024×Fs at 44.0 kHz<br>1024×Fs at 50.0 kHz |
| **MLBCLK** rise time | $t_{mckr}$ | | | 1 | ns | $V_{IL}$ to $V_{IH}$ |
| **MLBCLK** fall time | $t_{mckf}$ | | | 1 | ns | $V_{IH}$ to $V_{IL}$ |
| Input leakage current | $I_L$ | | | ±10 | µA | 0 < Vin < VDD |
| **MLBCLK** low time | $t_{mckl}$ | 6.1 | | | ns | Note 3 |
| **MLBCLK** high time | $t_{mckh}$ | 9.3 | | | ns | Notes 2, 3 |
| **MLBSIG/MLBDAT** receiver input valid to **MLBCLK** falling | $t_{dsmcf}$ | 1 | | | ns | |
| **MLBSIG/MLBDAT** receiver input hold from **MLBCLK** low | $t_{dhmcf}$ | $t_{mdzh}$ | | | ns | |
| **MLBSIG/MLBDAT** output high impedance from **MLBCLK** low | $t_{mcfdz}$ | 0 | | $t_{mckl}$ | ns | Note 4 |
| Bus hold from **MLBCLK** low | $t_{mdzh}$ | 2 | | | ns | Note 4 |

1. Higher $V_{IH}$ thresholds can be used; however, the risks associated with less noise margin in the system must be evaluated and assumed by the system implementor.
2. The Controller can shut off **MLBCLK** to place MediaLB in a low-power state. Depending on the time the clock is shut off, a runt pulse can occur on **MLBCLK**.
3. **MLBCLK** low and high times include pulse width variation.
4. The MediaLB driver can release the **MLBSIG/MLBDAT** line (e.g. high-impedance) as soon as **MLBCLK** is low; however, the logic state of the final driven bit on the line must remain on the bus for $t_{mdzh}$. Therefore, coupling must be minimized while meeting the maximum load capacitance listed.

### 2.1.2.3 Timing Diagrams

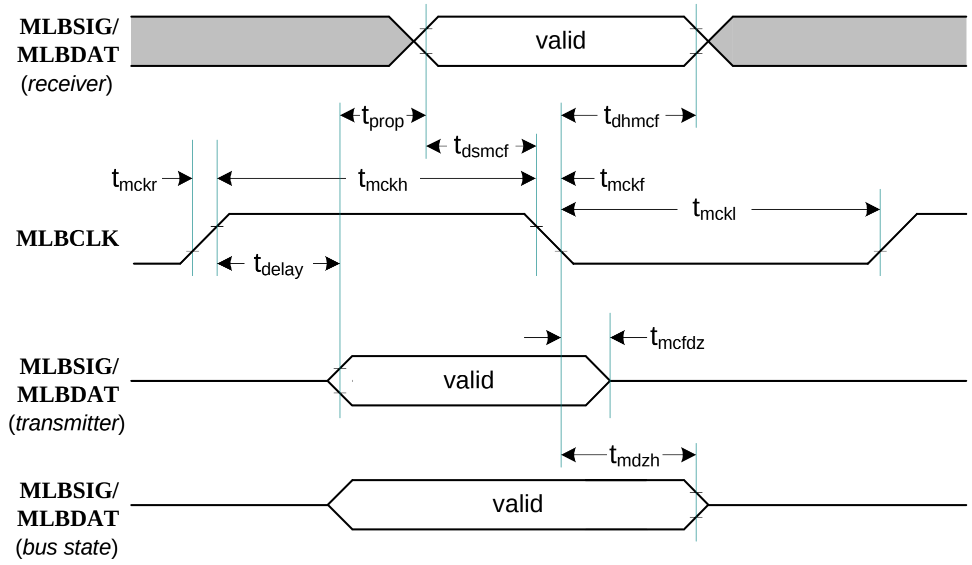


**Figure 2-2: MediaLB 3-pin Timing**

**MLBSIG/MLBDAT** is valid at a receiver input for at least $t_{dsmcf} + t_{dhmcf} + t_{mckf}$.
$t_{mcfdz}$ is only necessary when the next bit is transmitted by a different MediaLB Device. Recommendations and calculations for $t_{delay}$ and $t_{prop}$ are given in Appendix A.4.1.

## 2.2 MediaLB 5-pin Interface

### 2.2.1 Pin Description

The MediaLB 5-pin interface supports legacy Devices that do not have the ability to multiplex the input and output signals for data and signal. The 5-pin interface requires zero glue logic for two-node systems, and a simple logical OR'ing of the signal and data lines for greater than two-node systems. Even though the MediaLB 5-pin interface has separate data and signal input and output pins, the interface is used in a half-duplex fashion for consistency with the 3-pin interface.

If output signals can float (even while the Device is in reset), then pull-down resistors are required to keep un-driven signal and data lines from affecting the other nodes driving the signal and data lines. A two-node MediaLB 5-pin diagram is shown in Figure 2-3. The *MediaLB Device 1* is always driving **MLBDO** and **MLBSO**, which accounts for the lack of pull-down resistors. A four-node MediaLB 5-pin diagram is given in Figure 2-4. In this example, each Device must always drive **MLBDO** and **MLBSO**. The pull-down resistor values shown in Figures 2-3 and 2-4 are recommendations only.

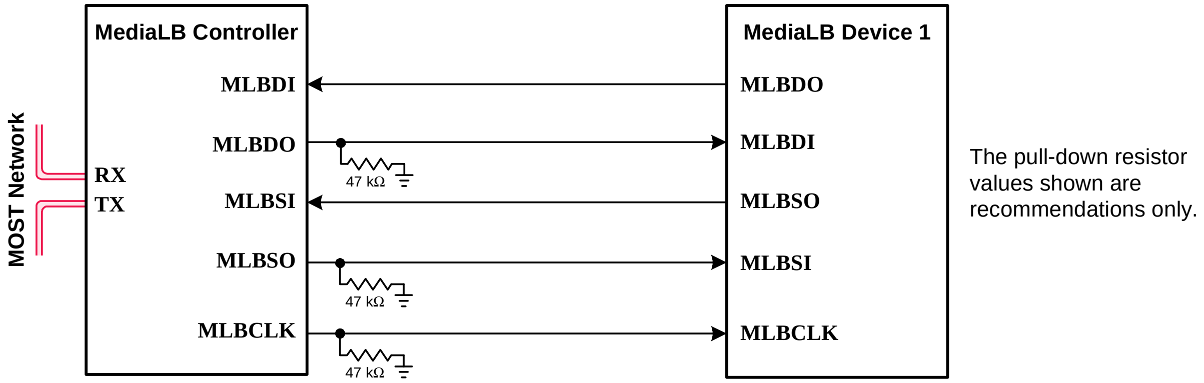


Figure 2-3: MediaLB 5-pin Two-Node Connection Diagram

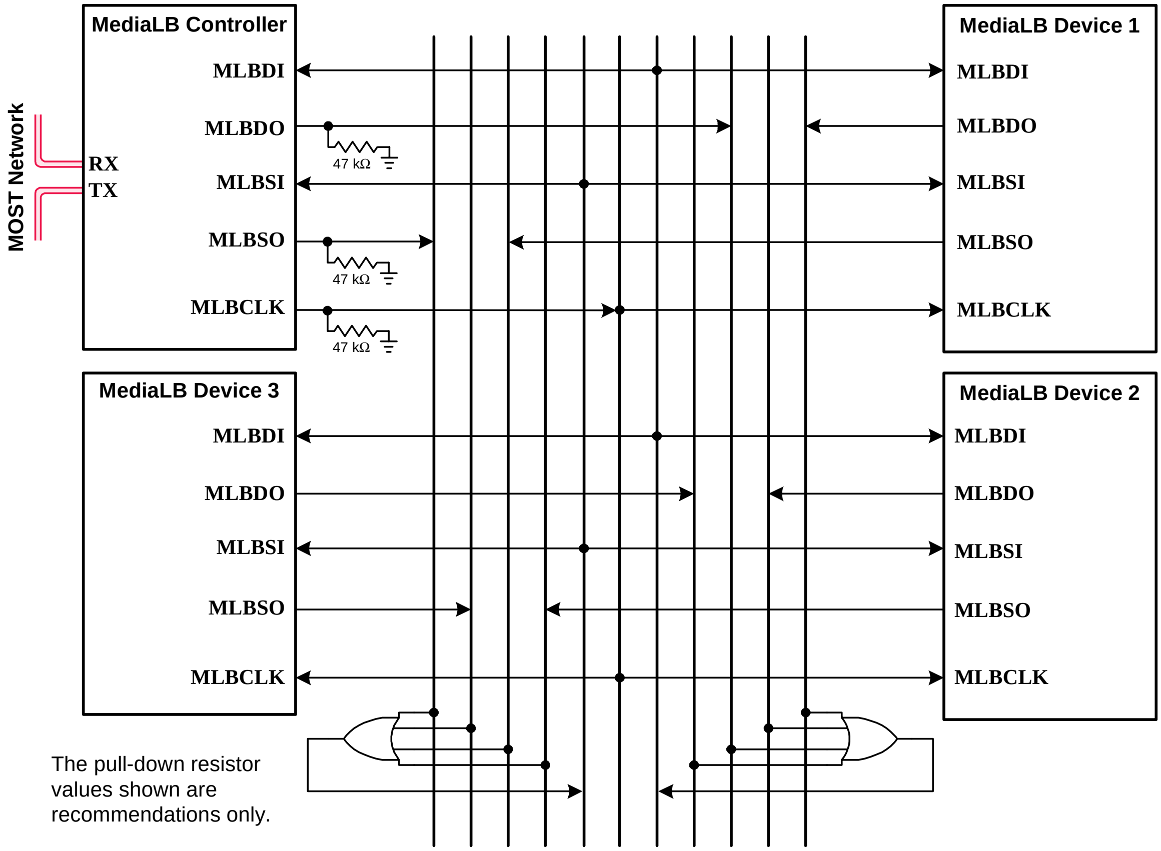


Figure 2-4: MediaLB 5-pin Four-Node Connection Diagram

## 2.2.2 Electrical Characteristics

In MediaLB 5-pin implementation the **MLBSO** and **MLBDO** lines are driven for a full **MLBCLK** period. Although the specifications below are listed for all MediaLB 5-pin clock speeds, individual chips or systems may choose to only implement specific speeds (e.g. 256×Fs or 512×Fs).

### 2.2.2.1 MediaLB Speed 256×Fs and 512×Fs

Ground = 0.0 V; Maximum load capacitance = 40 pF; Fs = 48 kHz; All timing parameters are specified from the valid voltage threshold as listed below; unless otherwise noted.

| Parameter | Symbol | Min | Typ | Max | Unit | Comment |
|---|---|---|---|---|---|---|
| Input voltage | | | | 3.6 | V | |
| Low-level input threshold | $V_{IL}$ | | | 0.7 | V | |
| High-level input threshold | $V_{IH}$ | 1.8 | | | V | Note 1 |
| Low-level output threshold | $V_{OL}$ | | | 0.4 | V | $I_{OL}$ = 1.6 mA |
| High-level output threshold | $V_{OH}$ | 2.0 | | | V | $I_{OH}$ = -1.6 mA |
| **MLBCLK** operating frequency (Note 2) | $f_{mck}$ | 11.264 | | 25.6 | MHz<br>MHz | 256×Fs at 44.0 kHz<br>512×Fs at 50.0 kHz |
| **MLBCLK** rise time | $t_{mckr}$ | | | 6 | ns | $V_{IL}$ to $V_{IH}$ |
| **MLBCLK** fall time | $t_{mckf}$ | | | 6 | ns | $V_{IH}$ to $V_{IL}$ |
| Input leakage current | $I_L$ | | | ±10 | µA | 0 < Vin < VDD |
| **MLBCLK** low time (Note 3) | $t_{mckl}$ | 30<br>15 | | | ns<br>ns | 256×Fs<br>512×Fs |
| **MLBCLK** high time (Note 3) | $t_{mckh}$ | 30<br>15 | | | ns<br>ns | 256×Fs<br>512×Fs |
| **MLBSI/MLBDI** valid to **MLBCLK** falling | $t_{dsmcf}$ | 3 | | | ns | Note 4 |
| **MLBSI/MLBDI** hold from **MLBCLK** low | $t_{dhmcf}$ | 5 | | | ns | |
| **MLBSO/MLBDO** low from **MLBCLK** high (Note 5) | $t_{mcrdl}$ | | | 20<br>10 | ns | 256×Fs<br>512×Fs |

1. Higher $V_{IH}$ thresholds can be used; however, the risks associated with less noise margin in the system must be evaluated and assumed by the customer.
2. The Controller can shut off **MLBCLK** to place MediaLB in a low-power state. Depending on the time the clock is shut off, a runt pulse can occur on **MLBCLK**.
3. **MLBCLK** low and high times include pulse width variation.
4. Gate delays due to OR'ing logic on the pins must be accounted for.
5. When a node is not driving valid data onto the bus, the **MLBSO** and **MLBDO** output lines must remain low. If the output lines can float at anytime, including while in reset, external pull-down resistors are required to keep the outputs from corrupting the MediaLB signal lines when not being driven.

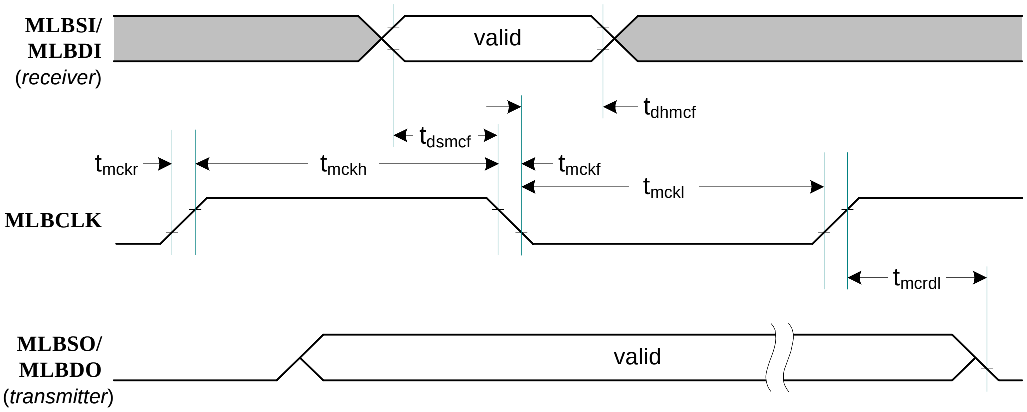


**Figure 2-5: MediaLB 5-pin Timing**

## 2.3 MediaLB 6-pin Interface

This section discusses differential signals, as well as the single-ended components of differential pairs. Refer to the *MediaLB Definition of Terms* on page 4 to distinguish signals names, as necessary.

### 2.3.1 Pin Description

The MediaLB Device on the 6-pin interface is attached to three differential signal pairs: **MLBCP**/**MLBCN**, **MLBSP**/**MLBSN** and **MLBDP**/**MLBDN**. The MediaLB Controller outputs the system clock and it is recovered by the clock recovery circuitry of each MediaLB Device. The clock recovery circuitry may generate a higher speed recovered clock from **MLBCP**/**MLBCN**.

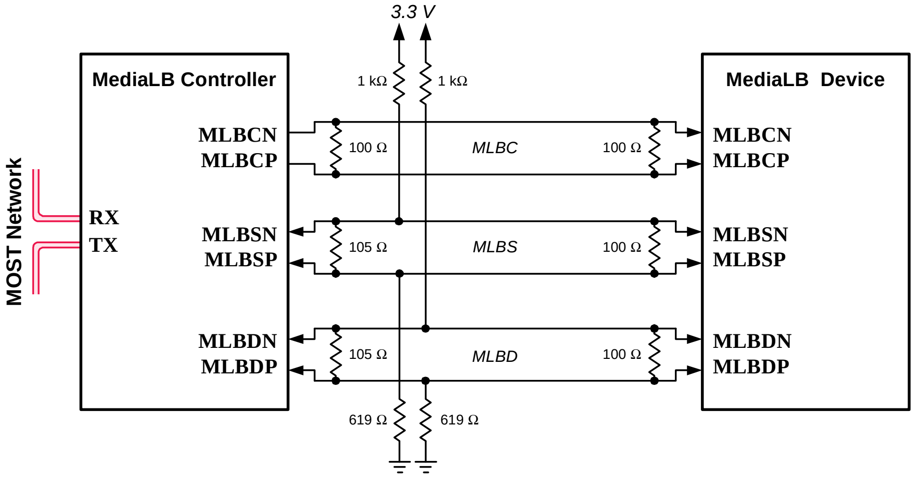
MediaLB is designed to support multipoint configurations, where multiple transceivers are connected to a single communication line to allow bi-directional communication. In order to effectively implement differential signaling in a multipoint application, special attention must be placed on transition rate, termination, and contention issues.

MediaLB is intended for implementation on a single PCB, utilizing short transmission lines without long stubs and small ground potential variation.

When the MediaLB 6-pin bus is implemented, discontinuities on the transmission lines should be minimized and stubs should be kept as short as possible. Each differential transmission line must be terminated with a parallel termination resistor (~100 Ω). This resistor terminates the differential line in its characteristic impedance and supplies the differential voltage for the current-mode driver. Because MediaLB 6-pin signals are bi-directional, the parallel termination resistor must be used at both ends of the transmission line.

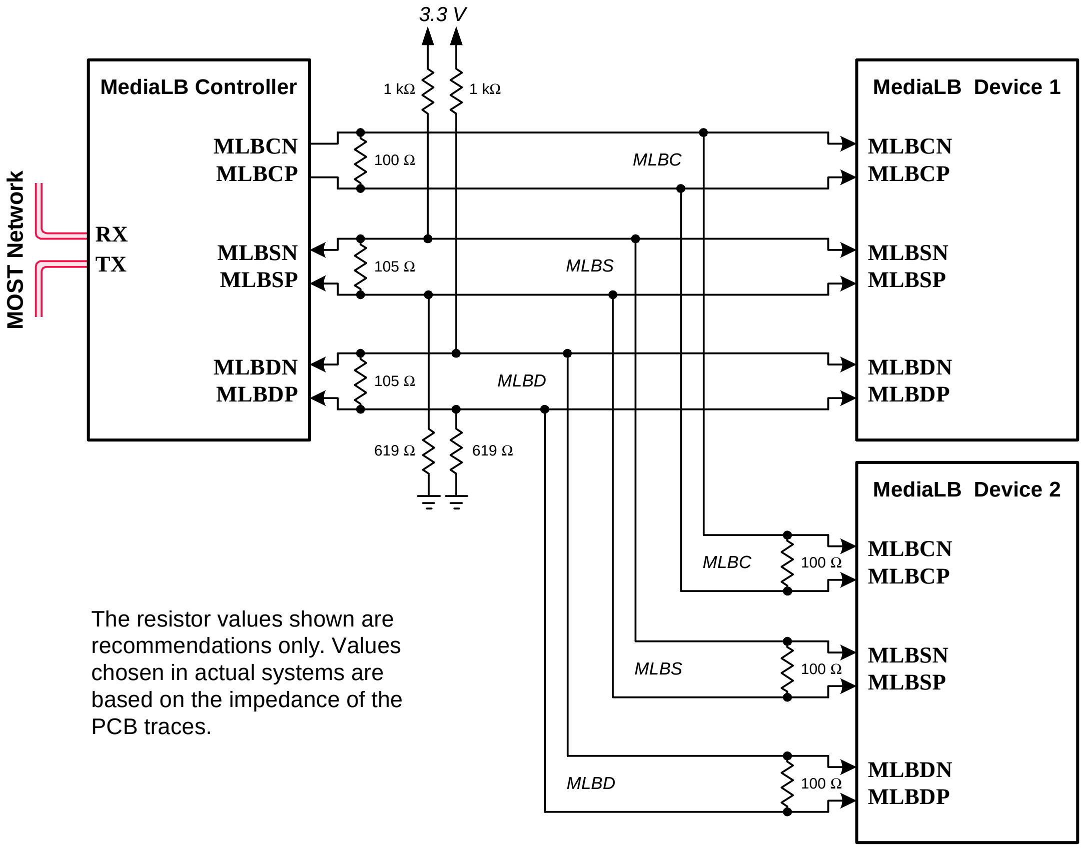
When the *MLBS* and *MLBD* lines are not driven, all receivers must recognize a logic low on these lines. To accomplish this, pull-up resistors are placed on **MLBSN**/**MLBDN** and pull-down resistors are placed on **MLBSP**/**MLBDP**. The pull-up/pull-down resistors on *MLBS* and *MLBD* introduce a differential voltage offset of -100 mV. Figures 2-6 and 2-7 show the connection diagrams for two-node and four-node MediaLB 6-pin systems. Since the pull-down and pull-up resistors change the termination impedance on the Controller side, the parallel termination resistor should be increased to 105 Ω.

An unknown state may exist on **MLBCP**/**MLBCN** during power-up or reset of the MediaLB Controller. Each MediaLB Device must be able to withstand this condition.



The resistor values shown are recommendations only. Values chosen in actual systems are based on the impedance of the PCB traces.

**Figure 2-6: MediaLB 6-pin Two-Node Connection Diagram**



The resistor values shown are recommendations only. Values chosen in actual systems are based on the impedance of the PCB traces.

**Figure 2-7: MediaLB 6-pin Three-Node Connection Diagram**

The MediaLB Device parallel termination resistors are placed near the pins of the furthest Device from the Controller. In order to support the edge rate of the MediaLB 6-pin interface, stub lengths must be minimized and pass-throughs must be used whenever possible. The MediaLB application example in Appendix A gives guidance when designing a multi-node MediaLB 6-pin system.

### 2.3.2 Electrical Characteristics

Although the specifications below are listed for all MediaLB 6-pin recovered clock speeds, individual chips or systems may choose to only implement specific speeds (e.g. 2048×Fs, 3072×Fs, 4096×Fs, 6144×Fs, or 8192×Fs).

Ground = 0.0 V; Maximum load capacitance = 5 pF; Fs = 48 kHz; All timing parameters are specified from the valid voltage threshold as listed below; unless otherwise noted.

| Parameter | Symbol | Min | Typ | Max | Unit | Comment |
|---|---|---|---|---|---|---|
| ***Driver Characteristics:*** | | | | | | |
| Differential output voltage (steady-state):<br>$\lvert V_{O+} - V_{O-} \rvert$ | $V_{OD}$ | 300 | | 500 | mV | Note 1 |
| Difference in differential output voltage between (high/low) steady-states:<br>$\lvert V_{OD,high} - V_{OD,low} \rvert$ | $\Delta V_{OD}$ | -50 | | 50 | mV | |
| Common-mode output voltage:<br>$( V_{O+} + V_{O-} ) / 2$ | $V_{OCM}$ | 1.0 | | 1.5 | V | |
| Difference in common-mode output between (high/low) steady-states:<br>$\lvert V_{OCM,high} - V_{OCM,low} \rvert$ | $\Delta V_{OCM}$ | -50 | | 50 | mV | |
| Variations in common-mode output during a logic state transitions | $V_{CMV}$ | | | 150 | mVpp | Note 2 |
| Short circuit current | $\lvert I_{OS} \rvert$ | | | 43 | mA | Note 3 |
| Differential output impedance | $Z_O$ | 1.6 | | | kΩ | |
| ***Receiver Characteristics:*** | | | | | | |
| Differential *clock* input: | | | | | | |
| logic low steady-state | $V_{ILC}$ | | | -50 | mV | Refer to Figure 2-10 |
| logic high steady-state | $V_{IHC}$ | 50 | | | mV | Note 4 |
| hysteresis | $V_{HSC}$ | -25 | | 25 | mV | |
| Differential *signal*/*data* input: | | | | | | |
| logic low steady-state | $V_{ILS}$ | | | -50 | mV | Refer to Figure 2-10 |
| logic high steady-state | $V_{IHS}$ | 50 | | | mV | Note 4 |
| Single-ended input voltage (steady-state): | | | | | | |
| **MLBSP**, **MLBDP** | $V_{IN+}$ | 0.5 | | 2.0 | V | |
| **MLBSN**, **MLBDN** | $V_{IN-}$ | 0.5 | | 2.0 | V | |

1. The single-ended output voltage of a driver is defined as $V_{O+}$ on **MLBCP**, **MLBSP**, and **MLBDP**. The single-ended output voltage of a driver is defined as $V_{O-}$ on **MLBCN**, **MLBSN**, and **MLBDN**.
2. Variations in the common-mode voltage can occur between logic states (e.g. during state transitions) as a result of differences in the transition rate of $V_{O+}$ and $V_{O-}$.
3. Short circuit current is applicable when $V_{O+}$ and $V_{O-}$ are shorted together and/or shorted to ground.
4. The logic state of the receiver is undefined when -50 mV < $V_{ID}$ < 50 mV as shown in Figure 2-10.

When switching between a logic high and a logic low state, $V_{O+}$ and $V_{O-}$ transitions should begin at the same time and change at the same rate. This keeps the common-mode voltage constant when during the state transitions. Additionally, the amplitude of the signal swing on $V_{O+}$ and $V_{O-}$ should remain the same from one steady-state to the next.

Figure 2-8 illustrates the output voltage characteristics of a MediaLB 6-pin driver. $V_{O+}$ and $V_{O-}$ represent the voltages at the two single-ended outputs of the differential signal. The difference between these voltages determines the logic state of the differential output ($V_{OD} = | V_{O+} - V_{O-} |$).

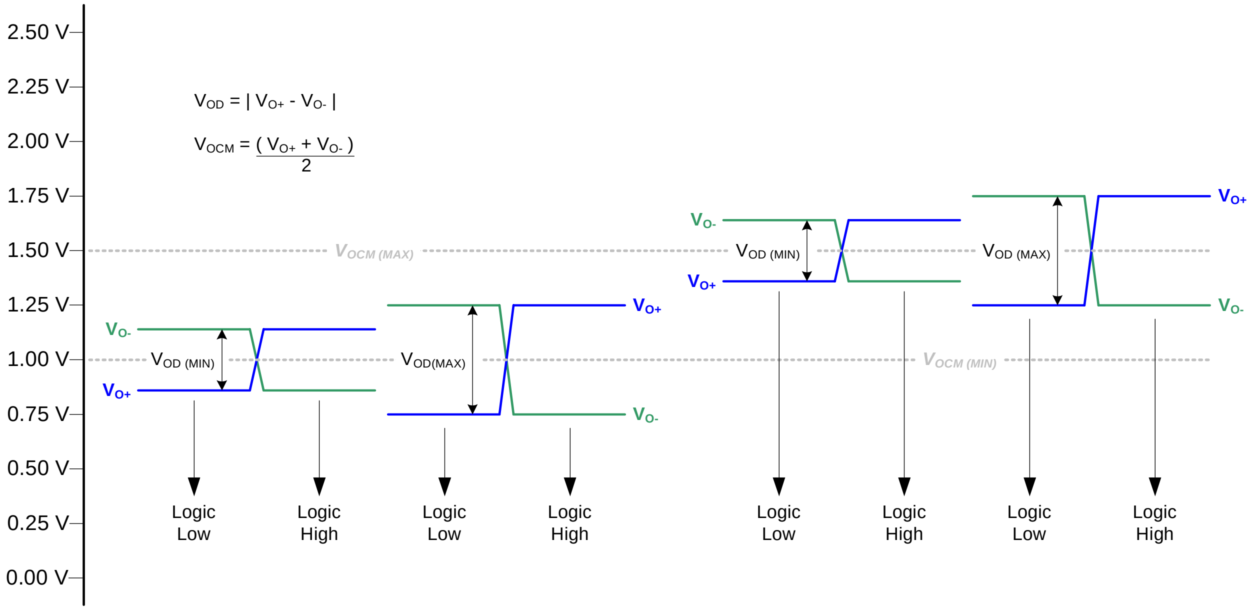


**Figure 2-8: MediaLB 6-pin Driver Levels**

Figure 2-9 illustrates the output voltage offset characteristics of a MediaLB 6-pin driver. $V_{OD,high}$ and $V_{OD,low}$ represent the two extremes of differential output voltage for high and low steady-states. $V_{OCM,high}$ and $V_{OCM,low}$ represent the two extremes of common-mode voltage for high and low steady-states. $V_{CMV}$ is the variance of the common-mode voltage during the logical transitions.

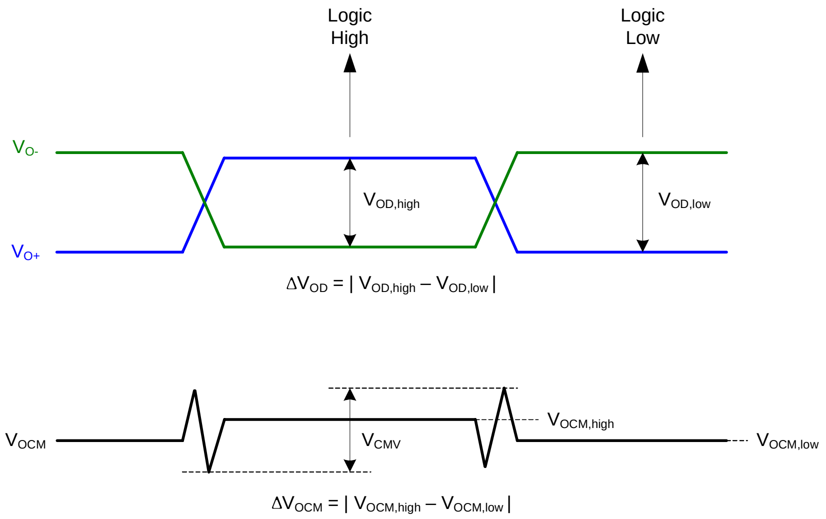


**Figure 2-9: MediaLB 6-pin Driver Offsets**

Figure 2-10 illustrates the input voltage characteristics of the MediaLB 6-pin clock, signal, and data receivers. $V_{IN+}$ and $V_{IN-}$ represent the voltages at the two single-ended inputs of a differential signal. The difference between these voltages determines the logic state of the differential input ($V_{ID} = V_{IN+} - V_{IN-}$).

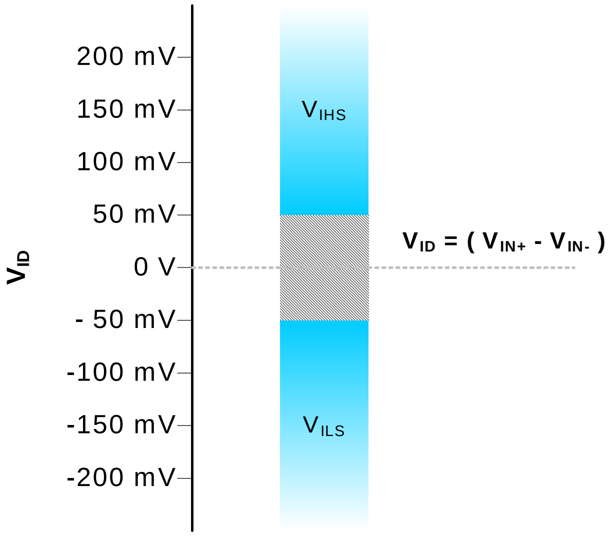


**Figure 2-10: MediaLB 6-pin Receivers Levels**

## 2.3.3 Timing Specifications

The recovered-to-external clock ratio may range from 1:1 to 4:1, with a maximum recovered clock frequency of 8192×Fs. When the clock ratio is greater than 1:1, a higher speed recovered clock is used to generate and recover *MLBD* and *MLBS* information at a rate faster than *MLBC*. Clock recovery circuits at the MediaLB 6-pin interface are required to generate the high-speed recovered clock from the differential *MLBC* bus clock.

All timing parameters are specified from the valid voltage ranges as listed below; unless otherwise noted.

| Parameter | Symbol | Min | Max | Unit | Comment |
|---|---|---|---|---|---|
| Differential transition time | $t_{mt}$ | | 1 | ns | 20% to 80% $V_{IN+/-}$<br>80% to 20% $V_{IN+/-}$ |
| **MLBCP/N** external clock operating frequency (Note 1) | $f_{mcke}$ | 67.584 | 102.4 | MHz<br>MHz | 1536×Fs at 44.0 kHz<br>2048×Fs at 50.0 kHz |
| Recovered clock operating frequency (Note 1) | $f_{mckr}$ | 90.112 | 409.6 | MHz<br>MHz | 2048×Fs at 44.0 kHz<br>8192×Fs at 50.0 kHz |
| Turnaround cycles:<br>**MLBDP/N** - following *Data*<br>**MLBSP/N** - following *Command*<br>**MLBSP/N** - following *RxStatus*<br>**MLBSP/N** - following *ChannelAddress* | | 3<br>1<br>1<br>1 | 3<br>1<br>1<br>1 | recovered clock cycles | 2048×Fs, 3072×Fs, and 4096×Fs |
| Turnaround cycles:<br>**MLBDP/N** - following *Data*<br>**MLBSP/N** - following *Command*<br>**MLBSP/N** - following *RxStatus*<br>**MLBSP/N** - following *ChannelAddress* | | 6<br>2<br>2<br>2 | 6<br>2<br>2<br>2 | recovered clock cycles | 6144×Fs and 8192×Fs |
| Cycle-to-cycle system jitter | $t_{jitter}$ | | 600 | ps | Note 2 |
| Transmitter **MLBSP/N** (**MLBDP/N**) output valid from transition of **MLBCP/N** (low-to-high) (Notes 3, 4) | $t_{delay}$ | 0.6 | 5.0 | ns | 2048×Fs |
| | | 0.6 | 2.5 | ns | 3072×Fs and 4096×Fs |
| | | 0.6<br>0.6 | 1.4<br>1.3 | ns<br>ns | 6144×Fs and 8192×Fs:<br>MediaLB Controller<br>MediaLB Device |
| Disable turnaround time from transition of **MLBCP/N** (low-to-high) (Note 3) | $t_{phz}$ | 0.6<br>0.6 | 7.0<br>3.5 | ns<br>ns | 2048×Fs<br>All other recovered clock speeds |
| Enable turnaround time from transition of **MLBCP/N** (low-to-high) (Note 3) | $t_{plz}$ | 0.6<br>0.6 | 11.2<br>5.6 | ns<br>ns | 2048×Fs<br>All other recovered clock speeds |
| **MLBSP/N** (**MLBDP/N**) valid to transition of **MLBCP/N** (low-to-high) (Note 3) | $t_{su}$ | 1<br>0.5<br>0.05 | | ns<br>ns<br>ns | 2048×Fs<br>3072×Fs and 4096×Fs<br>6144×Fs and 8192×Fs |
| **MLBSP/N** (**MLBDP/N**) hold from transition of **MLBCP/N** (low-to-high) (Notes 3, 5) | $t_{hd}$ | 0.8<br>0.6 | | ns<br>ns | MediaLB Controller<br>MediaLB Device |
| PCB propagation delay (Note 6) | $t_{prop}$ | 100 | 545 | ps<br>ps | All recovered clock speeds<br>8192×Fs at 50.0 kHz |

1. $f_{mcke(max)}$ and $f_{mckr(max)}$ include cycle-to-cycle system jitter ($t_{jitter}$).
2. Assumes a bit error rate of $10^{-9}$.
3. $t_{delay}$, $t_{phz}$, $t_{plz}$, $t_{su}$, and $t_{hd}$ may also be referenced from a low-to-high transition of the recovered clock for 2:1 and 4:1 recovered-to-external clock ratios.
4. Assistance with calculating $t_{delay}$ is given in Appendix A.4.2.
5. The transmitting Device must ensure valid data on **MLBSP/N** (**MLBDP/N**) for at least $t_{hd(min)}$ following the rising edge of **MLBCP/N**; receivers must latch **MLBSP/N** (**MLBDP/N**) data within $t_{hd(min)}$ of the rising edge of **MLBCP/N**.
6. Assumes 6.3 ps of propagation delay per mm of FR4.

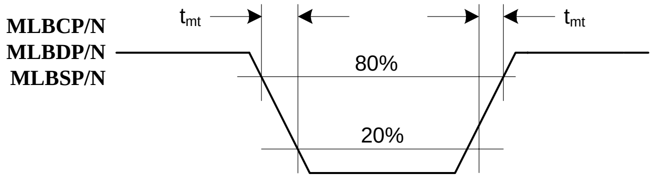


**Figure 2-11: MediaLB 6-pin Transition Time**

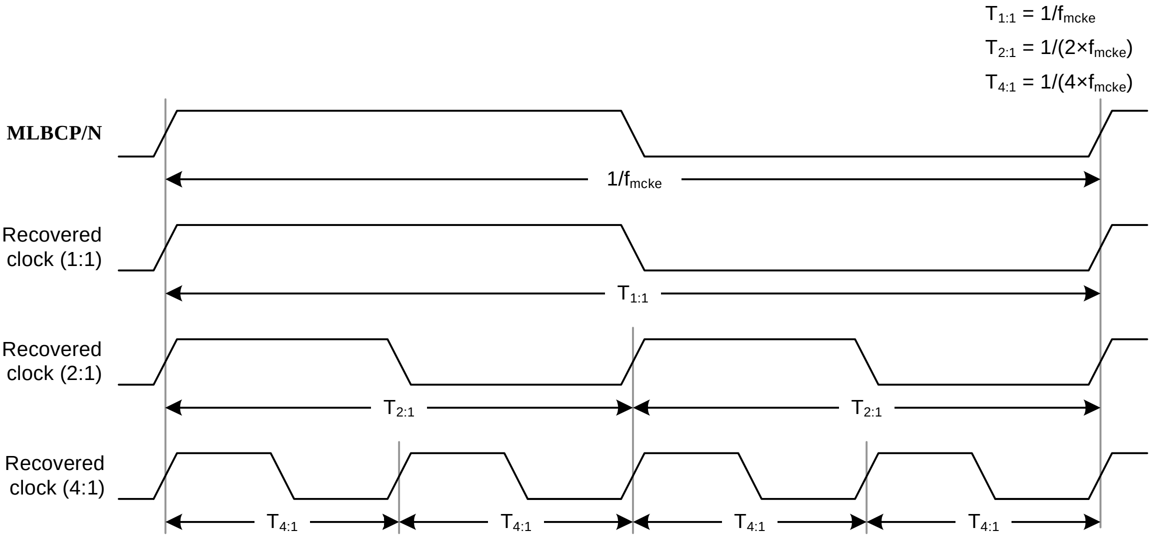


**Figure 2-12: MediaLB 6-pin Clock Definitions**

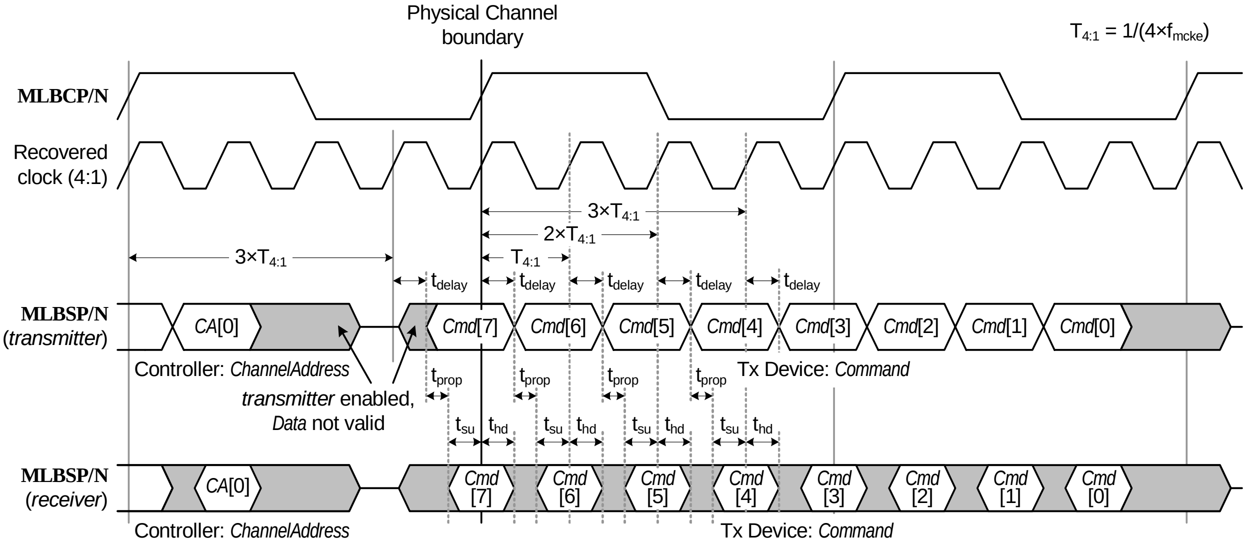


**Figure 2-13: MediaLB 6-pin Delay, Setup, and Hold Times**

$t_{delay}$, $t_{su}$, and $t_{hd}$ can also be referenced from $T_{4:1}$, $2×T_{4:1}$, and $3×T_{4:1}$ from the transition of **MLBCP/N** (low-to-high).

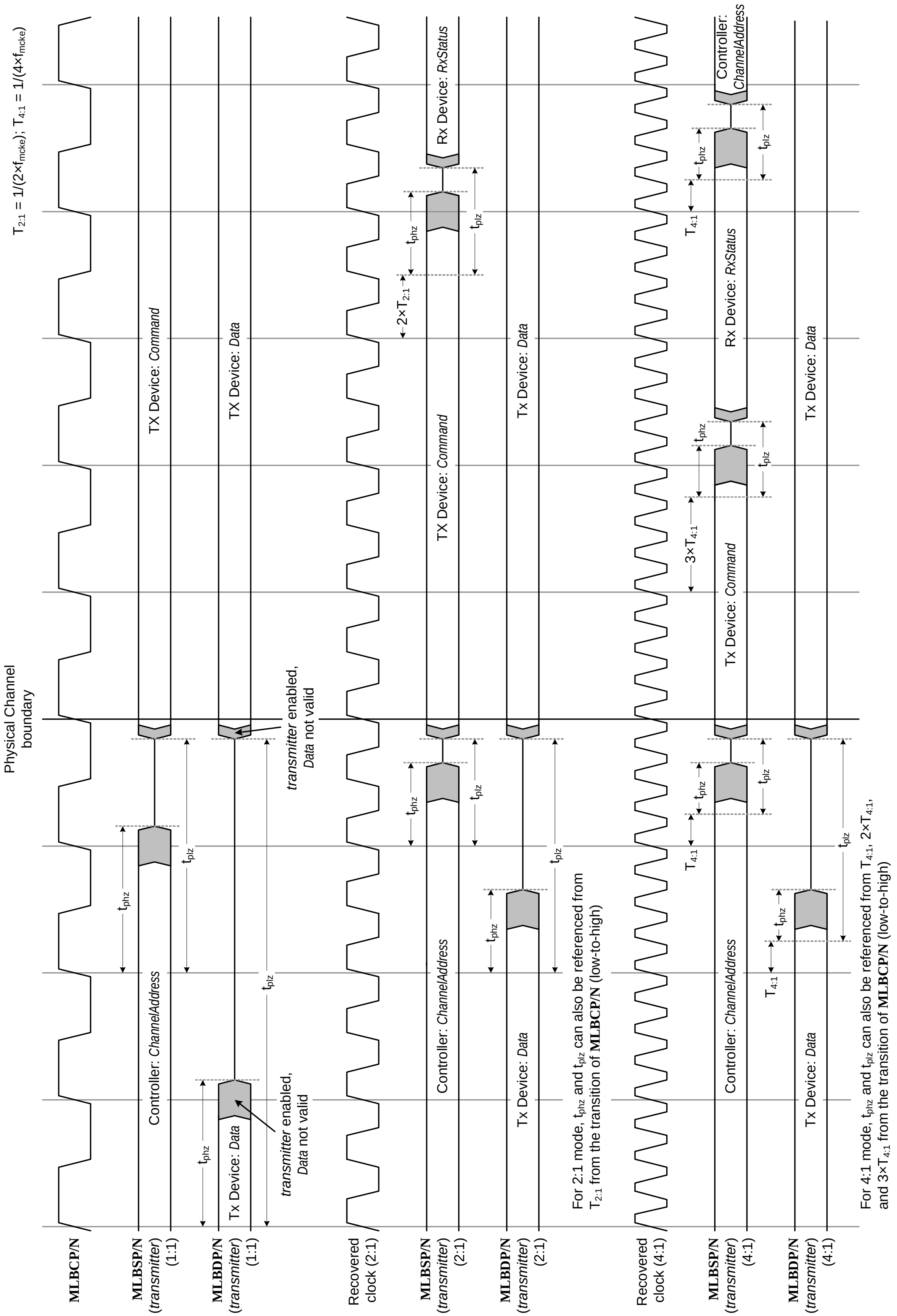

$T_{2:1} = 1/(2×f_{mcke})$; $T_{4:1} = 1/(4×f_{mcke})$
Physical Channel boundary
MLBCP/N
MLBSP/N (transmitter) (1:1)
MLBDP/N (transmitter) (1:1)
Recovered clock (2:1)
MLBSP/N (transmitter) (2:1)
MLBDP/N (transmitter) (2:1)
Recovered clock (4:1)
MLBSP/N (transmitter) (4:1)
MLBDP/N (transmitter) (4:1)
TX Device: *Command*
TX Device: *Data*
Controller: *ChannelAddress*
Tx Device: *Data*
*transmitter* enabled, *Data* not valid
*transmitter* enabled, *Data* not valid
$t_{phz}$
$t_{plz}$
Rx Device: *RxStatus*
TX Device: *Command*
Tx Device: *Data*
Controller: *ChannelAddress*
Tx Device: *Data*
$2×T_{2:1}$
For 2:1 mode, $t_{phz}$ and $t_{plz}$ can also be referenced from $T_{2:1}$ from the transition of MLBCP/N (low-to-high)
Controller: *ChannelAddress*
Rx Device: *RxStatus*
Tx Device: *Data*
Tx Device: *Command*
Controller: *ChannelAddress*
Tx Device: *Data*
$T_{4:1}$
$3×T_{4:1}$
For 4:1 mode, $t_{phz}$ and $t_{plz}$ can also be referenced from $T_{4:1}$, $2×T_{4:1}$, and $3×T_{4:1}$ from the transition of MLBCP/N (low-to-high)

**Figure 2-14: MediaLB 6-pin Disable and Enable Turnaround Times**

### 2.3.3.1 1:1 Clock Ratio

Figure 2-15 shows the structure of a MediaLB 6-pin physical channel in a 1:1 clock ratio, where the recovered clock rate is the same speed as the external clock (**MLBCP/N**). Gray areas indicate the bus turnaround cycles, the time for one transmitter to be turned off and next to be turned on. One turnaround cycle exists on **MLBSP/N** following *Command*, *RxStatus*, and *ChannelAddress*. A total of three turnaround cycles exist following *Data* on **MLBDP/N**.

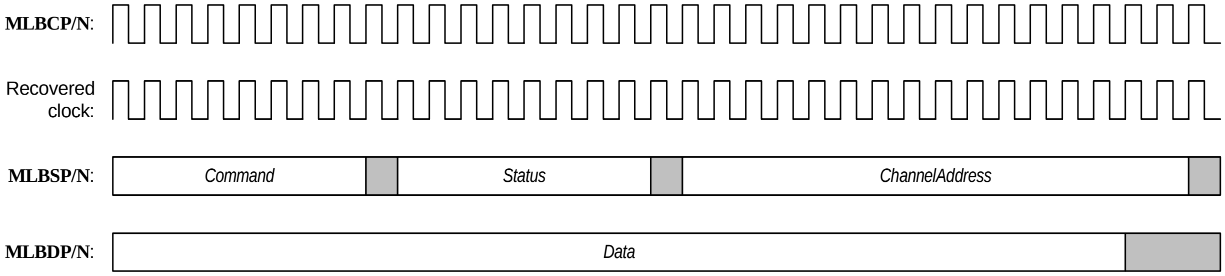


**Figure 2-15: MediaLB 6-pin 1:1 Clock Ratio**

### 2.3.3.2 2:1 Clock Ratio

Figure 2-16 shows the structure of a MediaLB 6-pin physical channel in a 2:1 clock ratio, where the recovered clock rate is twice the speed of the external clock (**MLBCP/N**). Gray areas indicate the bus turnaround cycles, the time for one transmitter to be turned off and next to be turned on. One turnaround cycle exists on **MLBSP/N** following *Command*, *RxStatus*, and *ChannelAddress*. A total of three turnaround cycles exist following *Data* on **MLBDP/N**.

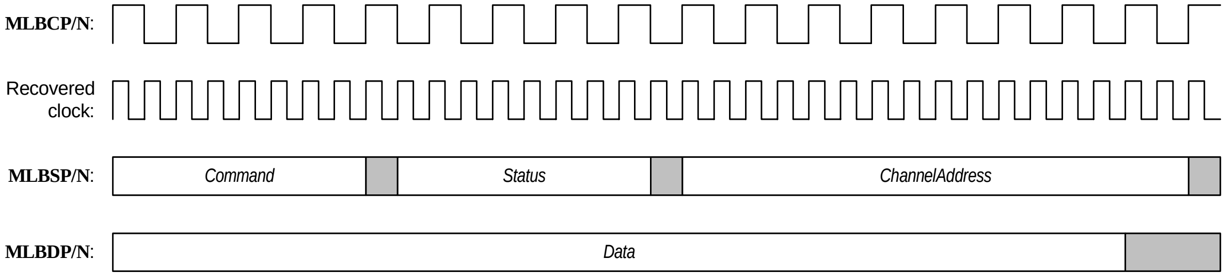


**Figure 2-16: MediaLB 6-pin 2:1 Clock Ratio**

### 2.3.3.3 4:1 Clock Ratio

Figure 2-17 shows the structure of a MediaLB 6-pin physical channel in a 4:1 clock ratio, where the recovered clock rate is four times the speed of the external clock (**MLBCP/N**). Gray areas indicate the bus turnaround cycles, the time for one transmitter to be turned off and next to be turned on. Two turnaround cycles exist on **MLBSP/N** following *Command*, *RxStatus*, and *ChannelAddress*. A total of six turnaround cycles exist following *Data* on **MLBDP/N**.

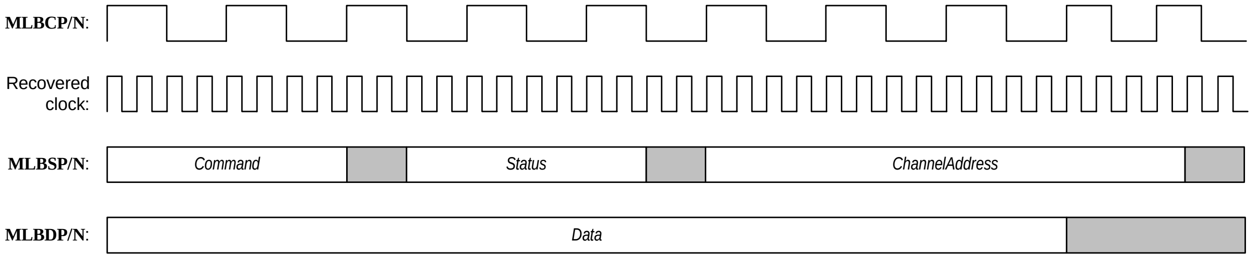


**Figure 2-17: MediaLB 6-pin 4:1 Clock Ratio**

# 3 Link Layer

The MediaLB link layer uses the concept of *ChannelAddress, Command*, *RxStatus*, and *Data* to transport all MOST Network data types and manage MediaLB. These terms are defined as follows:

- *ChannelAddress*:
  A 16-bit token, which is sent on the *MLBS* line by the MediaLB Controller at the end of a physical channel. A unique *ChannelAddress* defines a logical channel and grants a particular physical channel to a transmitting (Tx) and a receiving (Rx) MediaLB Device.
- *Command*:
  A byte-wide value sent by the transmitting (Tx) MediaLB Device on the *MLBS* line at the start of a physical channel. This command byte indicates the data type and additional control information to the Rx MediaLB Device. The Tx Device also outputs data on the *MLBD* signal during the same physical channel that *Command* is sent.
- *RxStatus*:
  A byte-wide value sent by the receiving (Rx) MediaLB Device on the *MLBS* line, after *Command* is sent. This status response provides a hardware handshaking mechanism and signals other control information, such as transmission errors, back to the sender.
- *Data*:
  The physical channel contains *Data* and is sent by the Tx MediaLB Device during the same physical channel in which *Command* is sent. This physical channel data must be transmitted left-justified, MSB first, most significant byte first. Note the Rx Device might return a status of busy, wherein the Tx Device must retransmit the same data in the next physical channel associated with the logical channel.

To dynamically configure *ChannelAddresses* for logical channels, a *DeviceAddress* can be pre-defined for MediaLB Devices. The *DeviceAddress* is a 16-bit address used in the *System Channel* with the *MLBScan* command to detect which MediaLB Devices exist.

## 3.1 ChannelAddresses

A MediaLB logical channel is defined as all physical channels associated with a single *ChannelAddress*. A logical channel on MediaLB is unidirectional; therefore, a single MediaLB Device sends data on a logical channel to one or more receiving Devices. If two Devices require bidirectional communication, then two MediaLB logical channels are required.

A *ChannelAddress* is 16-bits wide. Of the 16-bits, *ChannelAddress* (CA) bits 15 through 9 and the LSB are always zero. Only the eight bits CA[8:1] vary. A delay of one physical channel exists between the occurrence of the *ChannelAddress* and the actual physical channel granted. The 0x01FE *ChannelAddress* is defined as the *FRAMESYNC* pattern, where the end of the pattern determines the byte boundary, the physical channel boundary, and indicates that the MediaLB frame starts one physical channel later (PC0). The 0x0000 *ChannelAddress* is defined as the *BusIdle* state, which indicates that the corresponding physical channel is not assigned and not used by any Device. All odd *ChannelAddresses* are reserved; therefore, the LSB of a valid *ChannelAddress* is always zero. The *MLBS* line is in a consistent known state when not driven by any Device. For MediaLB 3/5-pin, this is achieved with the required weak pull-down. For MediaLB 6-pin, this is achieved as a result of a logic low differential input voltage offset at the MediaLB receivers (see Section 2.3.2).

| ChannelAddress* | Description |
|---|---|
| 0x0000 | *BusIdle* - Indicates that the physical channel is not being used, not assigned. |
| 0x0002..0x007E | 63 *ChannelAddresses* - defines the logical channels used in normal operation (MediaLB 3/5/6-pin) |
| 0x0080..0x01FC | Reserved |
| 0x01FE | *FRAMESYNC* - MediaLB frame alignment and *System Channel ChannelAddress* |
| 0x0200..0xFFFF | Reserved |

* All odd *ChannelAddresses* are reserved (LSB must be zero for valid *ChannelAddresses*).

**Table 3-1: MediaLB ChannelAddresses**

# 3.2 DeviceAddresses

*DeviceAddresses* are 16-bits wide, must be pre-assigned, and must be unique for each MediaLB Device. Of the 16-bits, *DeviceAddress* (DA) bits 15 through 9 and the LSB are always zero. Only the eight bits DA[8:1] vary. At the request of the EHC, *DeviceAddresses* can be scanned for by the MediaLB Controller to dynamically determine which Devices exist on MediaLB. Although Table 3-2 only lists 63 available *DeviceAddresses*, the *DeviceAddress* range can be expanded in the future up to the range 0x0002 through 0x01FC. *DeviceAddresses* are only used with the *MLBScan* command in the *System Channel* and are never assigned to physical channels. Once a Device is found, the *ChannelAddresses* used in normal operation can be assigned.

MediaLB Devices are encouraged to support dynamic configuration, where a preset *DeviceAddress* is used to assign the *ChannelAddresses* for each logical channel. Dynamic configuration avoids collisions of *ChannelAddresses* on different Devices.

To minimize collisions of *DeviceAddresses*, programmable Devices should assign the *DeviceAddress* via firmware. For non-programmable Devices, it is strongly recommended to have only the upper bits fixed, and have the lower bits configurable via pins on the Device. Table 3-2 is provided as a recommendation for fixing the upper *DeviceAddress* bits. Having the lower bits configurable via pins minimizes collisions with other manufacturer's Devices, as well as allows multiple instances of the same Device to coexist on the same MediaLB bus.

| DeviceAddresses | Range | Device Type |
|---|---|---|
| 0x0002..0x017E | | Reserved |
| 0x0180..0x0186 | 4 | External Host Controller Processors |
| 0x0188..0x018E | 4 | General Processors |
| 0x0190..0x0196 | 4 | TBD |
| 0x0198..0x019E | 4 | TBD |
| 0x01A0..0x01A6 | 4 | Digital Signal Processors |
| 0x01A8..0x01AE | 4 | TBD |
| 0x01B0..0x01B6 | 4 | Decoder Chips |
| 0x01B8..0x01BE | 4 | TBD |
| 0x01C0..0x01C6 | 4 | Encoder Chips |
| 0x01C8..0x01CE | 4 | TBD |
| 0x01D0..0x01DE | 8 | Digital-to-Analog Converters (DACs) |
| 0x01E0..0x01E6 | 4 | TBD |
| 0x01E8..0x01EE | 4 | TBD |
| 0x01F0..0x01FC | 7 | Analog-to-Digital Converters (ADCs) |

**Table 3-2: DeviceAddress Grouping**

## 3.3 Command Bytes

The MediaLB *Command* field is eight-bits wide and all odd values are reserved; therefore, the LSB of *Command* is always zero.

Transmitting MediaLB Devices (including the Controller) place *Command* on the *MLBS* line to indicate the type of data being transmitted on the *MLBD* line.

Two types of MediaLB commands are defined: *Normal* and *System*. *Normal* commands are those sent by the transmitting MediaLB Device (or Controller) in non-*System Channels*. *System* commands are those sent by the MediaLB Controller in the *System Channel*.

| Value* | Command | Description |
|---|---|---|
| ***Normal Commands (Tx Device sends in non-System Channels):*** | | |
| 00h | *NoData* | No data to send out in this physical channel. |
| 02h..0Eh | rsvd | Reserved |
| 10h | *SyncData* | Tx Device sends out *SyncData* command to indicate synchronous stream data. |
| 12h..1Eh | rsvd | Reserved |
| 20h | *AsyncStart* | Asynchronous logical channel. Start of a packet. |
| 22h | *AsyncContinue* | Asynchronous logical channel. Middle of a packet. |
| 24h | *AsyncEnd* | Asynchronous logical channel. End of a packet. |
| 26h | *AsyncBreak* | Asynchronous logical channel. Indicates a packet stop. No valid data present on the *MLBD* line. |
| 28h..2Eh | rsvd | Reserved |
| 30h | *ControlStart* | Control logical channel. Start of a message. |
| 32h | *ControlContinue* | Control logical channel. Middle of a message. |
| 34h | *ControlEnd* | Control logical channel. End of a message. |
| 36h | *ControlBreak* | Control logical channel. Indicates a message stop. No valid data present on the *MLBD* line. |
| 38h..3Eh | rsvd | Reserved |
| 40h | *IsoNoData* | Isochronous logical channel, no data valid. |
| 42h | *Iso1Byte* | Isochronous logical channel, one data byte valid. First byte (MSB) transmitted/received is valid. Last three bytes in physical channel are empty. |
| 44h | *Iso2Bytes* | Isochronous logical channel, first two data bytes valid. First byte transmitted/received is the MSB. Last two bytes in physical channel are empty. |
| 46h | *Iso3Bytes* | Isochronous logical channel, first three data bytes valid. First byte transmitted/received is the MSB. Last byte in physical channel is empty. |
| 48h | *Iso4Bytes* | Isochronous logical channel, all four data bytes valid. First byte transmitted/received is the MSB. |
| 4Ah..4Eh | rsvd | Reserved |
| 50h | *IsoSync1Byte* | Isochronous logical channel, one data byte valid and start of a block.<br>First byte transmitted/received is valid. Last three bytes in physical channel are empty. |
| 52h | *IsoSync2Bytes* | Isochronous logical channel, two data bytes valid and start of a block.<br>First byte transmitted/received is the MSB. Last two bytes in the physical channel are empty. |
| 54h | *IsoSync3Bytes* | Isochronous logical channel, three data bytes valid and start of a block.<br>First byte transmitted/received is the MSB. Last byte in physical channel is empty. |
| 56h | *IsoSync4Bytes* | Isochronous logical channel, all four data bytes valid and start of a block.<br>First byte transmitted/received is the MSB. |
| 58h..DEh | rsvd | Reserved |

* All odd values (LSB set) are reserved.

**Table 3-3: MediaLB Commands**

| Value* | Command | Description |
|---|---|---|
| ***System Commands (Controller sends in System Channel):*** | | |
| 00h | *NoData* | The Controller has no *System* command to send out. |
| E0h | *MOSTLock* | The Controller issues a MOST Network lock command in the *System Channel* to notify Devices that the MOST Network is in lock. |
| E2h | *MOSTUnlock* | The Controller issues a MOST Network unlock command in the *System Channel* to notify Devices that the MOST Network is unlocked. |
| E4h | *MLBScan* | The Controller issues an MediaLB scan command in the *System Channel* and uses the *MLBD* line to indicate the *DeviceAddress* which is currently being scanned. All Devices supporting *MLBScan* must compare the received *DeviceAddress* against their internal *DeviceAddress*, and if a match occurs, a Device responds in the following *System Channel* with one of the *System* responses as specified in Table 3-4. |
| E6h | *MLBSubCmd* | The Controller outputs a sub-command in the *System Channel*. The sub-command is part of the data on the *MLBD* line. |
| E8h..FCh | rsvd | Reserved |
| FEh | *MLBReset* | The Controller outputs a MediaLB reset on the *System Channel MLBS* line. If the first two-bytes are zero on the *MLBD* line, then the system reset is a broadcast system reset and every Device should reset its MediaLB interface. Otherwise, the *MLBD* line contains the *DeviceAddress* of the Device being asked to reset its own MediaLB interface. |

* All odd values (LSB set) are reserved.

**Table 3-3: MediaLB Commands (Continued)**

For synchronous logical channels, the *NoData* command indicates that the Tx Device assigned to that *ChannelAddress* has not setup the channel yet. For asynchronous and control logical channels, *NoData* is used during packet data transfer when there is no data available to transmit.

## 3.4 RxStatus Bytes

The MediaLB *RxStatus* field is eight-bits wide and all odd values are reserved; therefore, the LSB of *RxStatus* is always zero. Receiving Devices must place *RxStatus* on the *MLBS* line after the Tx Device command byte (*Command*). The *RxStatus* status responses are divided into two categories: current state and feedback. The current state *RxStatus* indicates the status of the Rx Device in the current physical channel, whereas the feedback *RxStatus* is a response to a *Command* in the previous logical channel. For *Normal* responses, only the *ReceiverProtocolError* is a feedback *RxStatus* byte. All *System* responses are also feedback *RxStatus* bytes.

Two types of MediaLB status responses are defined: *Normal* and *System*. *Normal* status responses are sent by the receiving MediaLB Device (or Controller) in the non-*System Channels*. *System* status responses are sent by the receiving MediaLB Device in the *System Channel*.

For synchronous data reception, the Rx Device does not drive a response. For MediaLB 3/5-pin, the pull-down resistor on the *MLBS* line will implement the *ReceiverReady* response automatically (cannot be delayed or stopped). For MediaLB 6-pin, the *ReceiverReady* command is also the default *MLBS* state. This is a result of a logic low differential input voltage offset at the MediaLB receivers (see Section 2.3.2).

For control or asynchronous packet reception, the Rx Device responds to a control or asynchronous command with *ReceiverReady* if it can accept the quadlet on the *MLBD* line. If the Rx Device cannot accept the quadlet, then it will respond with a status of *ReceiverBusy*. If the Rx Device needs to stop or cancel the packet transmission, it can respond with a status of *ReceiverBreak*, in which case the Tx Device must stop transmitting the current packet.

When the Rx Device recognizes an error, the *ReceiverProtocolError* status response is sent in the next physical channel that is part of the logical channel. The status response of *ReceiverProtocolError* is issued by the Rx Device under certain conditions, see Section 3.9 for details.

| Value* | RxStatus | Description |
|---|---|---|
| ***Normal Responses (Rx Device response in non-System Channels):*** | | |
| 00h | *ReceiverReady* | Current state indicating the receiving Device is ready to receive the data. This is the default for the bus. The Rx Devices should not drive this response for broadcast channels. |
| 02h..0Eh | rsvd | Reserved |
| 10h | *ReceiverBusy* | Current state indicating the Rx Device is not ready to receive the data. The data must be retransmitted in the next physical channel associated with this logical channel. This response is not allowed on synchronous channels. |
| 12h..6Eh | rsvd | Reserved |
| 70h | *ReceiverBreak* | Current state indicating the Rx Device will not receive additional data quadlets and requests termination of the data transmission. Only allowed on control and asynchronous channels. |
| 72h | *ReceiverProtocolError* | Feedback indicating the command received in the prior physical channel (of this logical channel) did not match the pre-defined channel format or was out of sequence. Only allowed on control and asynchronous channels. |
| 74h..7Eh | rsvd | Reserved |
| ***System Responses (Rx Device response in System Channel):*** | | |
| 00h | *DeviceNotPresent* | No response to *DeviceAddress* scan (*MLBScan*). No Device associated with that *DeviceAddress* exists on MediaLB. |
| 80h | *DevicePresent* | Device response to *DeviceAddress* scan (*MLBScan*), where the scanned Device has all *ChannelAddresses* assigned. |
| 82h | *DeviceServiceRequest* | Device response to *DeviceAddress* scan (*MLBScan*), where the scanned Device needs some or all its *ChannelAddresses* configured. |
| 84h..FEh | rsvd | Reserved |

* All odd values (LSB set) are reserved.

**Table 3-4: MediaLB RxStatus Responses**

## 3.5 System Commands

The Controller sends out *System* commands in the physical channel associated with the *FRAMESYNC* MediaLB frame alignment *ChannelAddress* (PC0). The *NoData* command indicates no command exists on the *System Channel* for this frame. All *System* commands are optional and may or may not be implemented on the MediaLB Controller. Additionally, *System* responses (including dynamic configuration) are optional and may or may not be implemented on a specific MediaLB Device.

The *MOSTLock* and *MOSTUnlock* commands indicate the status of the Controller relative to the MOST Network. When the Controller is not locked to the MOST Network (*MOSTUnlock*), all MediaLB data being transferred to or from the MOST Network must also stop. Buffers in the Controller could delay the stopping point to beyond when *MOSTUnlock* shows up on MediaLB.

The *MLBReset* command is designed to place the MediaLB interface in one or all Devices in a known state. When a MediaLB Device receives the *MLBReset* command, it will look at the corresponding first two received (most significant) data bytes on the *MLBD* line:

- If the first two bytes are zero, then all MediaLB Devices must reset their MediaLB interface to an initialized known state (broadcast reset to all Devices).
- If the first two bytes match the local *DeviceAddress*, then only the Device with the matching *DeviceAddress* will reset its MediaLB interface to an initialized known state (reset targeted to only one Device).

The *MLBSubCmd* command is used for configuration and status information from the Controller to Devices. A sub-command is contained in the first byte of the *MLBD* quadlet. When MediaLB Device interfaces receive the *MLBSubCmd* command, they will store the command and corresponding data quadlet (sub-command). Currently, only one sub-command is defined (*scSetCA*) and is used in dynamic configuration.

MediaLB Devices and *ChannelAddresses* can be configured using two methods: static or dynamic. When the EHC MediaLB Device uses the dynamic method, it instructs the Controller to scan for other MediaLB Devices. As Devices are found, the EHC then instructs the Controller to configure the found Device via the *MLBSubCmd* command.

The EHC determines which *DeviceAddresses* to scan for and, once a Device is found, which *ChannelAddresses* to assign. The EHC uses the pre-defined logical channels opened when MediaLB was started to transfer messages to the Controller. The EHC sends a message to the Controller to start scanning for a particular *DeviceAddress*. The Controller then sends the *MLBScan* command into the *System Channel*, and places the *DeviceAddress* into the first two bytes (most significant or first two transmitted) of the *System Channel* on *MLBD*.

An Rx Device with a matching *DeviceAddress* must send a status response of *DevicePresent* in the next *System Channel* if the *ChannelAddresses* are already assigned or fixed. If the *ChannelAddresses* have not been assigned, then the Rx Device must respond with *DeviceServiceRequest*.

If a Device is found, the Controller sends a message to the EHC indicating the Device's presence and whether the Device needs to be configured or not. For Devices that need to be configured (requesting service), the EHC must then send a message to the Controller defining which *ChannelAddresses* to send to the Device. The Controller then sends this information to all Devices using the *MLBSubCmd* command in the *System Channel*.

The *MLBSubCmd* command data field contains four bytes that are defined as follows:

| 31 24 | 23 16 | 15 8 | 7 0 |
|---|---|---|---|
| sub-command = scSetCA | DA[8:1] | CA[8:1] | Index |

**Figure 3-1: Sub-Command scSetCA Quadlet**

The *scSetCA* (01h) sub-command (under the MediaLB *MLBSubCmd* command) supports dynamic configuration of MediaLB *ChannelAddresses*. The bytes are defined as follows:

- scSetCA (01h) - Sub-command to Set *ChannelAddress*. Indicates that the rest of the bytes are logical channel configuration information.
- DA[8:1] - *DeviceAddress* bits 8 through 1, where all other bits are zero. Matches the *DeviceAddress* found during the *MLBScan* command.
- CA[8:1] - *ChannelAddress* bits 8 through 1, where all other bits are zero. Assigned *ChannelAddress* associated with a specific Index (Device's logical channel) below.
- Index - Indicates which logical channel within a Device to associate the *ChannelAddress* with. This index enables a Device to support multiple logical channels. Index 0 and 1 are reserved for control channels. Devices that do not support control channels will start at Index 2 (with Indices 0 and 1 unused).

MediaLB Devices receiving this sub-command should check the DA[8:1] byte to determine whether this *DeviceAddress* matches its own. If the *DeviceAddress* matches, then the Device uses the *ChannelAddress* (CA[8:1] bits) for the logical channel associated with that Index. If a Device is reset or drops off MediaLB, it must re-initialize to its power-up state and discard any previously assigned *ChannelAddresses*.

MediaLB Device documentation must contain a table defining the relationship between the Index value, the particular logical channel associated with it, and the type and maximum bandwidth supported. In addition, the Device must indicate how many frames are needed to set the *ChannelAddress* once the *scSetCA* sub-command has been received. The EHC must use this data to determine the wait between setting Indices/Logical channels.

## 3.6 Data Structure for MediaLB 3/5-pin

The MediaLB data structure consists of a *ChannelAddress*, a Tx command (*Command*), an Rx response (*RxStatus*), and four data bytes (*Data*).

A MediaLB data structure flow is:

- The MediaLB Controller places a *ChannelAddress* on the *MLBS* line. This addresses two or more MediaLB Devices. One acts as a Tx MediaLB Device and the other or others act as Rx MediaLB Devices.
- After a fixed delay of 4 bytes (one quadlet or physical channel), the addressed Tx MediaLB Device responds by shifting out a command byte (*Command*) onto the *MLBS* line, coincident with the start of 4 bytes of data onto the *MLBD* line.
- The Rx MediaLB Device responds in the same physical channel by shifting out its status response (*RxStatus*) onto the *MLBS* line after the Tx Device's *Command*. The *RxStatus* reports the status of the receiving Device to the sender. For asynchronous, control and isochronous (non-broadcast) transmissions, the data sent is accepted if the receiver presents a status response of *ReceiverReady* or rejected if the receiver presents a status response of *ReceiverBusy*. For synchronous and isochronous (broadcast) transmissions, the receiving Device must not drive any *RxStatus*, thereby defaulting to *ReceiverReady*. Synchronous (and some isochronous) data is sent in a broadcast fashion and supports multiple receiving Devices.

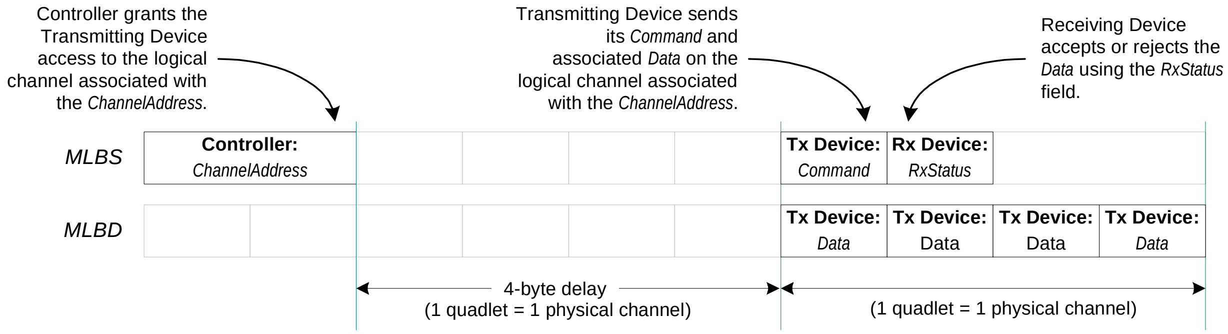

Controller grants the Transmitting Device access to the logical channel associated with the ChannelAddress.
Transmitting Device sends its Command and associated Data on the logical channel associated with the ChannelAddress.
Receiving Device accepts or rejects the Data using the RxStatus field.
MLBS
Controller: ChannelAddress
Tx Device: Command
Rx Device: RxStatus
MLBD
Tx Device: Data
Tx Device: Data
Tx Device: Data
Tx Device: Data
4-byte delay
(1 quadlet = 1 physical channel)
(1 quadlet = 1 physical channel)

**Figure 3-2: MediaLB 3/5-pin Data Structure**

During normal operation, the MediaLB Controller initiates a transfer by sending out the *ChannelAddress* on the *MLBS* line, and then stops driving (high-impedance) the *MLBS* line. When a MediaLB Device recognizes the *ChannelAddress* as related to one of its channels, the Tx Device will generate the *Command* on the *MLBS* line and place the data on the *MLBD* line. The Rx Device will generate the *RxStatus* on the *MLBS* line, after the *Command*. Both *Command* and *RxStatus* are output in the second quadlet after the matching *ChannelAddress* occurred. If the Rx Device reports a status response of *ReceiverBusy*, then the Tx Device must retransmit the *Command* and *Data* in the next physical channel assigned to that same *ChannelAddress* (next quadlet in the logical channel). If the Tx Device transmits the *NoData* command, the Rx Device ignores the data on the *MLBD* line.

This results in the following scheme:

Controller: *ChannelAddress* → Tx Device: *Command* → Rx Device: *RxStatus*

Since for synchronous data transmission (*SyncData*) the status response must always be *ReceiverReady* (bus default when signal not driven), synchronous data supports broadcast transmission to multiple Rx Devices.

After the Tx Device outputs *Command*, it must stop driving the *MLBS* line to allow the Rx Device to output *RxStatus*. At the end of the physical channel, the Tx Device must also stop driving the *MLBD* line unless the *ChannelAddress* for the next physical channel is also assigned to it. Likewise, after the Rx Device outputs *RxStatus*, it must stop driving the *MLBS* line to allow the Controller to output another *ChannelAddress*.

Figure 3-3 illustrates which Device is driving the MediaLB signal and data lines, using the 256Fs speed as an example. Depending on the number of physical channels that are grouped into logical channels, fewer unique *ChannelAddresses* may be seen in the frame. In Figure 3-3, each logical channel is one quadlet (one physical channel), mapping to seven *ChannelAddresses* (B through H). If one logical channel consisted of two quadlets and another consisted of three quadlets, then only four unique *ChannelAddresses* would be seen on MediaLB (B through E).

For MediaLB synchronization purposes, *ChannelAddress* 0x01FE is defined as the *FRAMESYNC* pattern. The MediaLB Controller generates this pattern once per MOST Network frame on the *MLBS* line. The MediaLB link layer is designed to ensure that this bit pattern is unique on the bus.

All MediaLB Devices must synchronize their byte boundary and their physical channel boundary upon receiving the *FRAMESYNC* pattern. The end of the *FRAMESYNC* pattern also indicates that four bytes later is the start of the MediaLB frame (PC0), and the *System Channel*. The actual number of physical channels supported is determined by the MediaLB clock speed. Table 3-5 illustrates the number of available quadlets and physical channels per frame for MediaLB 3/5-pin speed modes.

| MediaLB Speed | Physical Channels per Frame | Available Physical Channels per Frame* |
|---|---|---|
| 256×Fs | 8 | 7 (PC1 - PC7) |
| 512×Fs | 16 | 15 (PC1 - PC15) |
| 1024×Fs | 32 | 31 (PC1 - PC31) |

* PC0 (first physical channel of the MediaLB frame) is always used as the *System Channel*.

**Table 3-5: MediaLB 3/5-pin Valid Physical Channels**

The *MLBS* and *MLBD* physical channel associated with the *FRAMESYNC ChannelAddress* (PC0), is defined as the *System Channel* and can be used by the Controller to broadcast system control and status information to all Devices. Examples of *System* commands are *MLBReset* and *MLBScan*. Status examples include *MOSTLock* and *MOSTUnlock* which indicate the status of the MOST Network to MediaLB Devices.

MediaLB supports both static physical channel assignments or dynamic implementations. As an example of a static implementation, the Controller can automatically open a pair of logical channels at power-up. Through these channels, the rest of the MediaLB bandwidth can be configured by a MediaLB Device (generally the EHC). For a dynamic implementation, the EHC can request the Controller to scan for specific *DeviceAddresses* and then configure the Devices found (see the *MLBScan System* command).

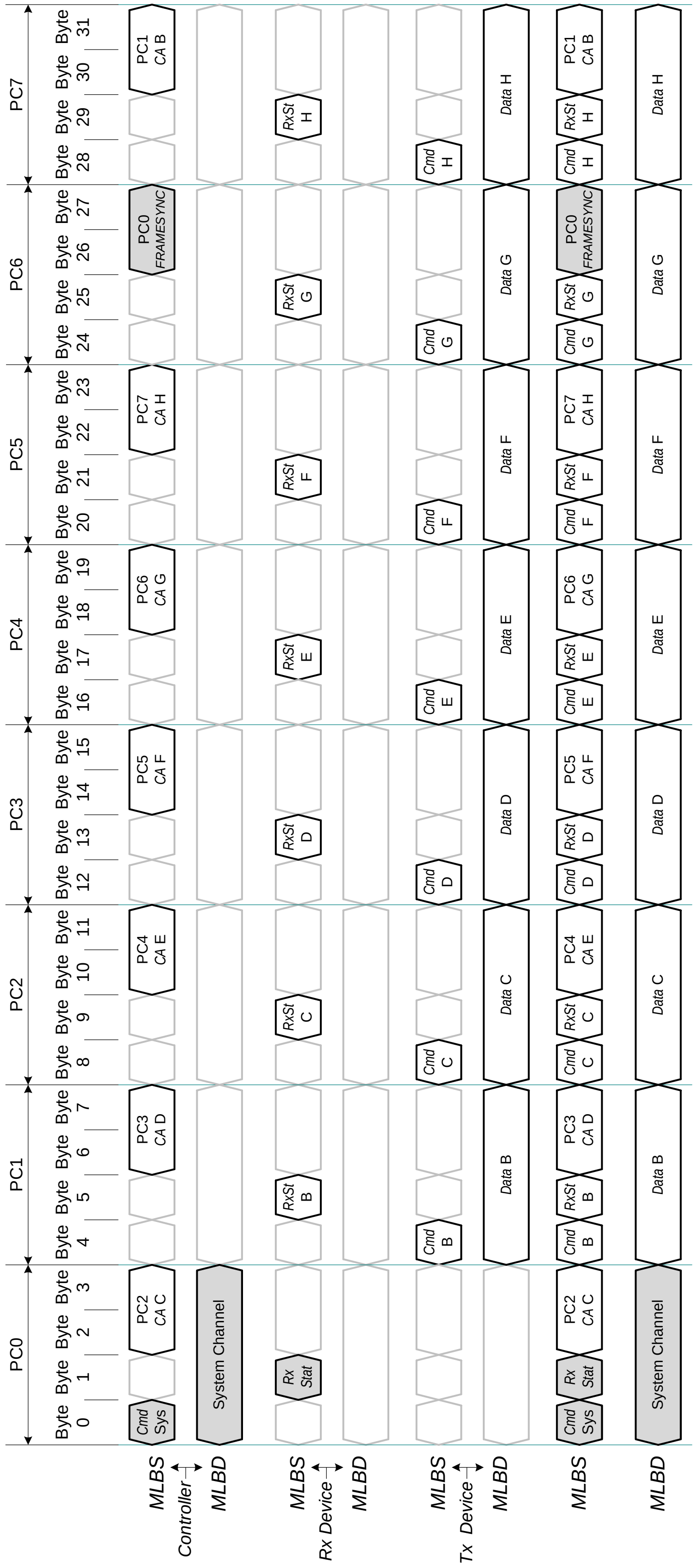


**Figure 3-3: MediaLB 3/5-pin 256Fs Interface Example**

## 3.7 Data Structure for MediaLB 6-pin

The data structure of MediaLB 6-pin resembles that of MediaLB 3/5-pin (with *ChannelAddress*, *Command*, *RxStatus*, and *Data*); however, turnaround cycles are embedded within the data structure which allow time for one transmitter to be turned off and the next to be turned on. Figure 3-4 illustrates the structure of a single physical channel (PC) on MediaLB 6-pin.

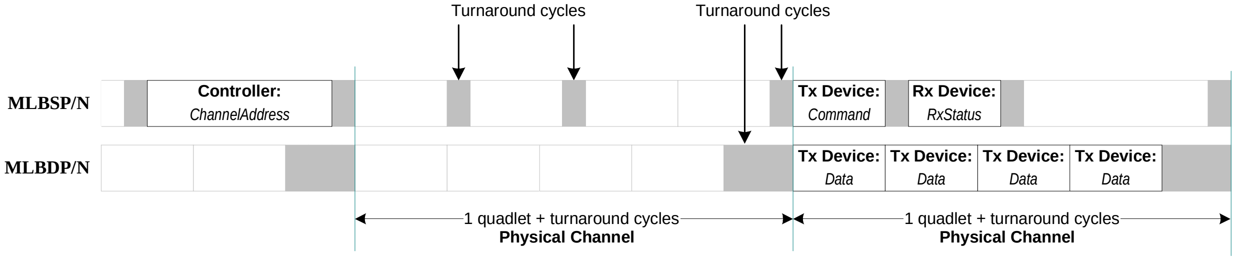


**Figure 3-4: MediaLB 6-pin Data Structure**

The length of the physical channel in MediaLB 6-pin is longer than 1 quadlet as a result of the embedded turnaround cycles. The total number of physical channels within a MediaLB frame must be an integer, so dead cycles will exist to pad the end of the MediaLB frame. The number of turnaround cycles in a single physical channel, the number of physical channels supported, and the number of dead cycles per frame is determined by the clock ratio selected, see Table 3-6.

| Recovered-to-External Clock Ratio | Recovered Clock Speed | Physical Channel Structure | Physical Channels per Frame | Available Physical Channels per Frame[1] | Dead Cycles per Frame[2] |
|---|---|---|---|---|---|
| 1:1 | 2048×Fs | 4 bytes plus 3 turnaround cycles (35 total cycles) | 58 | 57 (PC1 - PC57) | 18 |
| 2:1 | 3072×Fs<br>4096×Fs | | 87<br>117 | 86 (PC1 - PC86)<br>116 (PC1 - PC116) | 27<br>1 |
| 4:1 | 6144×Fs<br>8192×Fs | 4 bytes plus 6 turnaround cycles (38 total cycles) | 161<br>215 | 160 (PC1 - PC160)<br>214 (PC1 - PC214) | 26<br>22 |

1. PC0 (first physical channel of the MediaLB frame) is always used as the *System Channel*.
2. Dead cycles occur at the end of the MediaLB frame. During this time, no MediaLB Device drives the *MLBD* or *MLBS* line.

**Table 3-6: MediaLB 6-pin Valid Physical Channels**

The use of the *Command*, *RxStatus*, and *ChannelAddress* within a physical channel remains applicable to MediaLB 6-pin, including the use of the *FRAMESYNC ChannelAddress* (0x01FE) for synchronization purposes. All MediaLB Devices must synchronize their byte boundary and their physical channel boundary upon receiving the *FRAMESYNC* pattern. The end of the *FRAMESYNC* pattern indicates that the start of the MediaLB frame (PC0), will begin following 1 physical channel and the end of the dead cycles.

Figure 3-3 illustrates the 1:1 clock ratio, with a recovered clock speed of 2048×Fs. At this speed, the cycles during each MediaLB frame are used as follows:

(58 physical channels) * ((32 data cycles/PC) + (3 turnaround cycles/PC)) + (18 dead cycles) = 2048 cycles

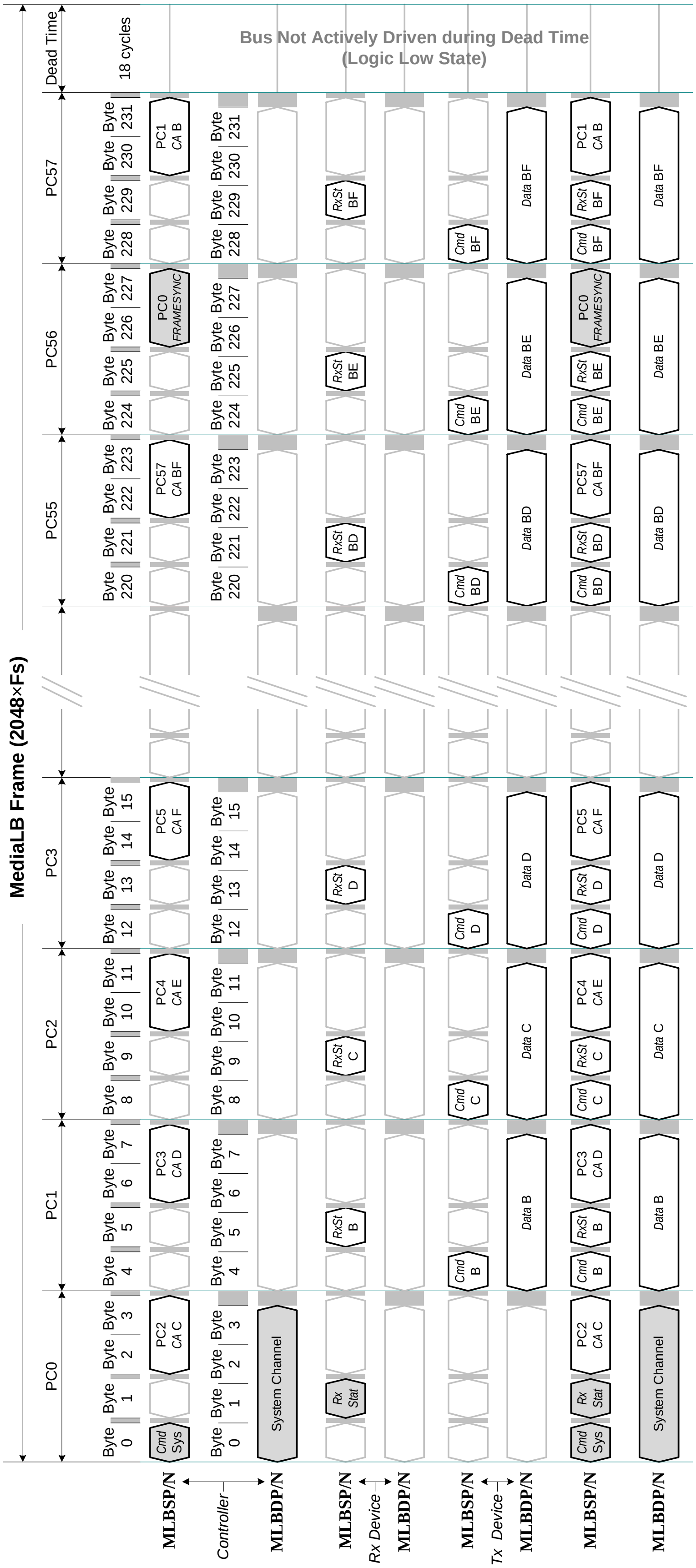


**Figure 3-5: MediaLB 6-pin 2048×Fs (1:1) Interface Example**

## 3.8 Initialization

At power up, the MediaLB Controller might output a *MLBReset* command in the *System Channel* (all *System* commands are optional). Upon reception of the *MLBReset* command, all MediaLB Devices will cancel any current transmissions or receptions and clear their buffers.

Two scenarios are supported to configure MediaLB Devices and *ChannelAddresses*:

- Static pre-configured before startup. The system implementor decides which *ChannelAddresses* are to be used for every communication path on MediaLB. This static MediaLB configuration can be communicated by the EHC to the Controller through pre-defined power-up logical channels or through a secondary port.
- Dynamically at run-time. Dynamic configuration allows the board designer to support multiple build options where the EHC can query to find out if a particular Device is present or not on a particular board. The EHC instructs the Controller to scan for a particular *DeviceAddress* in the *System Channel*. The Controller uses the *MLBScan* command to look for a Device. The Controller then notifies the EHC whether the Device is present or not. If the Device is present, then the EHC can instruct the Controller to set the *ChannelAddresses* for the Device found. The EHC sends messages to the Controller to set each Indices/Logical channel, and waits the appropriate amount of time between each message as specified in the Device's documentation. When that particular Device is configured, the EHC can instruct the Controller to scan for the next Device.

Since the MediaLB Controller is the interface between the MediaLB Devices and the MOST Network, the Controller provides the *MLBC* signal and will also continue to operate even when the MOST Network is unlocked. When no activity exists on MediaLB, the Controller can shut off the *MLBC* placing MediaLB in a low-power state. The *ChannelAddress* assignments are not affected in low-power state; therefore, the same communication paths exists once *MLBC* is restarted.

MediaLB Devices are synchronously slaved to the MediaLB Controller through the *MLBC* signal. Since the Controller is synchronized to the MOST Network, the *MLBC* signal provides Network synchronization to all MediaLB Devices. Once the Controller starts up *MLBC*, all MediaLB Devices must synchronize to the MediaLB frame before communication can commence. When not frame-locked, Devices must search for the *FRAMESYNC* pattern, which defines a byte and physical channel boundary. Additionally, the start of the MediaLB frame (PC0) occurs one quadlet after *FRAMESYNC* is present on the bus. Even when a Device is frame-locked, it should check every frame continuing to validate that it remains frame-locked. While frame-locked, the Device can access MediaLB according the rules of the MediaLB protocol.

A MediaLB Device must perform the following operations:

- Rules for synchronization to MediaLB:
  - When locked, as long as *FRAMESYNC* is detected at the expected time, the Device must not synchronize to unexpected *FRAMESYNC* patterns.
  - When locked and *FRAMESYNC* is not detected at the expected time for two consecutive frames, declare unlock, and the Device stops driving *MLBS* and *MLBD*.
  - When unlocked and *FRAMESYNC* is detected at the same time for three consecutive frames, declare lock, and the Device can resume driving *MLBS* and *MLBD* when appropriate.
- When the Tx Device for a physical channel, it drives *Command* onto *MLBS* at the beginning of the physical channel and then sets *MLBS* to a high impedance state. In addition, the Tx Device drives the data quadlet onto *MLBD* line for the duration of the physical channel, and then sets the *MLBD* line to a high impedance state. The *NoData* command is the default for the *MLBS* line and does not need to be driven by the Tx Device.
- When the Rx Device for a physical channel, it drives *RxStatus* onto *MLBS* in the second byte of the physical channel and then sets *MLBS* to a high impedance state for asynchronous, control and isochronous (non-broadcast) transmissions. When no *RxStatus* is driven, the *MLBS* line defaults to *ReceiverReady*; however, it is recommended that the Rx Device drive the *ReceiverReady* response for non-broadcast transmissions.
- When the Rx Device for a physical channel, it must not drive any *RxStatus* (defaulting to *ReceiverReady*) for synchronous and isochronous (broadcast) transmissions.

## 3.9 Data Transport Methods

MediaLB supports four data transport methods: synchronous stream data, asynchronous packet data, control message data and isochronous data. Synchronous stream data is transmitted in a broadcast fashion, where the only response allowed by an Rx Device is *ReceiverReady* (*MLBS* default). Control and asynchronous transport methods are packet based and support only one Rx Device at a time. Control and asynchronous transmissions require start and end commands to delineate the packets. Isochronous data can be broadcast if all Rx Devices do not use the status response of *ReceiverBusy*. Otherwise, isochronous transmissions must be to a single Rx Device.

### 3.9.1 Control and Asynchronous

Both the control and asynchronous commands define the boundaries of a packet message and work similarly. The following discussion on control packets also applies to asynchronous packets with the commands changed to the asynchronous versions.

For control packets, the *ControlStart* command is sent by the Tx Device at the start of a message. After the first quadlet of the message, middle quadlets will use the *ControlContinue* command. For the last quadlet of the packet, the Tx Device uses the *ControlEnd* command. If the command sequence is received out of order, the Rx Device sends the status response of *ReceiverProtocolError* in the next quadlet of the logical channel.

If the Tx Device must abort the packet while it's being transmitted, the *ControlBreak* command is sent. Assuming a message is to be retransmitted after the *ControlBreak* command is sent, the message must be restarted from the beginning (cannot resume with the *ControlContinue* command).

The protocol flow for a Tx Device is illustrated in Figure 3-6 through Figure 3-8. Although these diagrams illustrate control packet transmission, they also apply to asynchronous packets where the commands that start with *Control* are replaced by *Async*. The data transfer blocks (slanted rectangle shapes) occur only during a physical channel (PCn) associated with the logical channel defined by a single *ChannelAddress*.

The flow diagram contains four states: Idle, Start, Continue, and End. Each state uses a different command when sending the data. The Idle state is the starting point, waiting for the application to initiate a packet transfer. When a quadlet is ready to be transferred, the flow diagram moves to the Start state.

If a *ControlEnd* command is sent in the physical channel preceding a *ReceiverProtocolError* RxStatus (in either the Idle or Start state), the *ReceiverProtocolError* status response must be assigned to the previous packet transmitted. Alternatively, a status response of *ReceiverProtocolError* (in either the Idle or Start state) must not be assigned to the previous packet transmitted unless *ControlEnd* was sent in the preceding physical channel.

Once a quadlet has been sent successfully, the flow diagram moves to the Continue state, depicted in Figure 3-7, and stays there until all but the last quadlet has been transmitted. The last quadlet is transmitted in the End state, which is depicted in Figure 3-8.

The protocol flow for an Rx Device is illustrated in Figure 3-9. This flow diagram consists of only two states: Idle and Continue. The Idle state is the starting point where the Rx Device is waiting for a packet start command. Once a start command has been received (*ControlStart* or *AsyncStart*), the flow diagram moves to the Continue state. The reception of a *ControlEnd* command completes the transfer and moves the flow diagram back to the Idle state, where it waits for the next packet.

The protocol flow for an Rx Device, as described in Figure 3-9, should be used as a reference for standard MediaLB Devices. According to this flow, a *ReceiverProtocolError* status response may be sent by an Rx Device only in the Continue state; however, more enhanced MediaLB Devices can also conduct protocol checks in the Idle state. In this case, a *ReceiverProtocolError* status response could be sent for example, if a logical channel is setup for control data and an isochronous or synchronous command is received. Protocol checks in the Continue state may be expanded beyond the flow shown in Figure 3-9 when required by specific implementations.

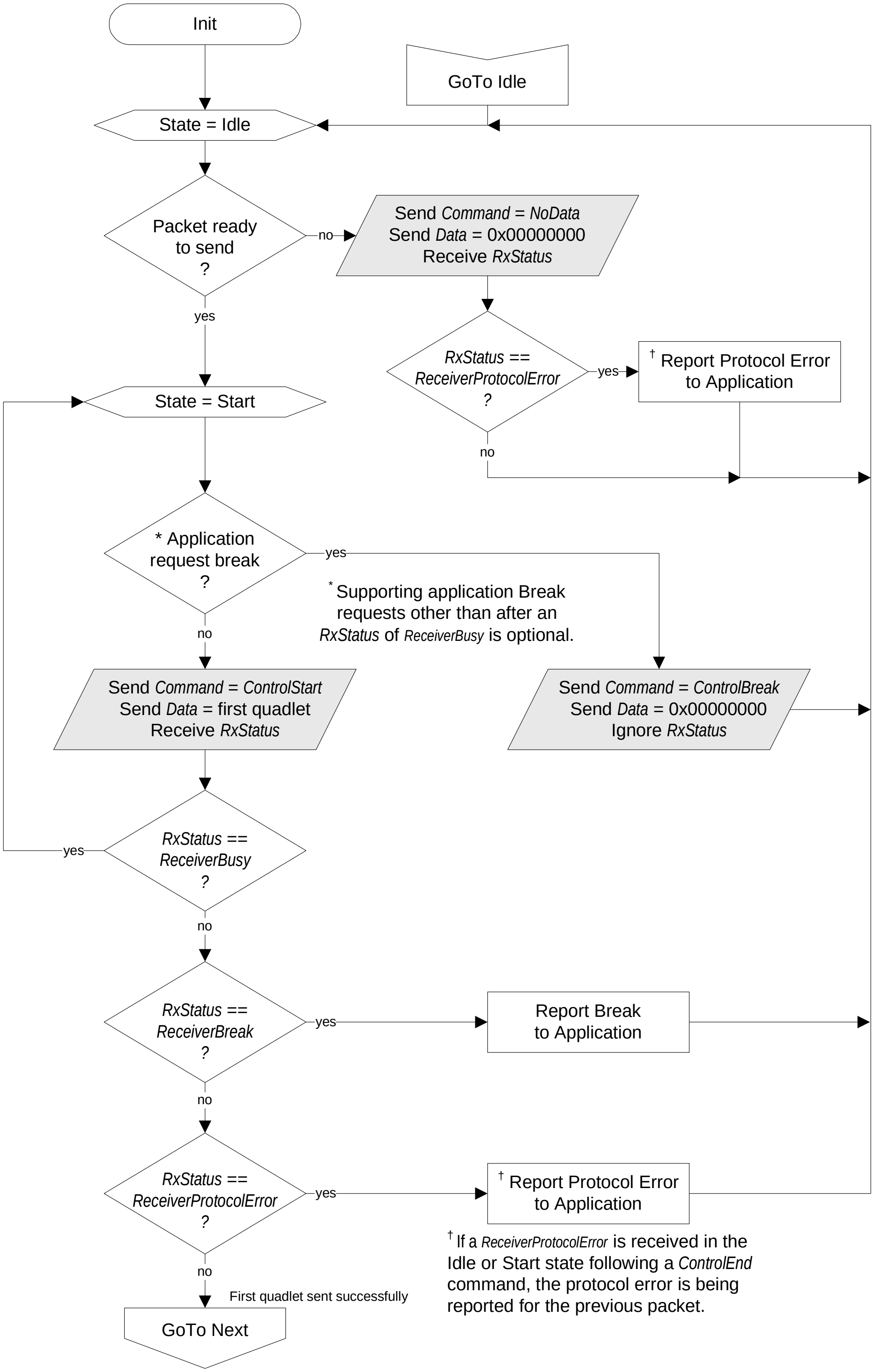

Init
GoTo Idle
State = Idle
Packet ready to send ?
no
Send Command = NoData
Send Data = 0x00000000
Receive RxStatus
yes
RxStatus == ReceiverProtocolError ?
yes
† Report Protocol Error to Application
no
State = Start
* Application request break ?
yes
* Supporting application Break requests other than after an RxStatus of ReceiverBusy is optional.
no
Send Command = ControlStart
Send Data = first quadlet
Receive RxStatus
Send Command = ControlBreak
Send Data = 0x00000000
Ignore RxStatus
RxStatus == ReceiverBusy ?
yes
no
RxStatus == ReceiverBreak ?
yes
Report Break to Application
no
RxStatus == ReceiverProtocolError ?
yes
† Report Protocol Error to Application
† If a ReceiverProtocolError is received in the Idle or Start state following a ControlEnd command, the protocol error is being reported for the previous packet.
no
First quadlet sent successfully
GoTo Next

**Figure 3-6: Control Packet Tx Device Protocol: Start**

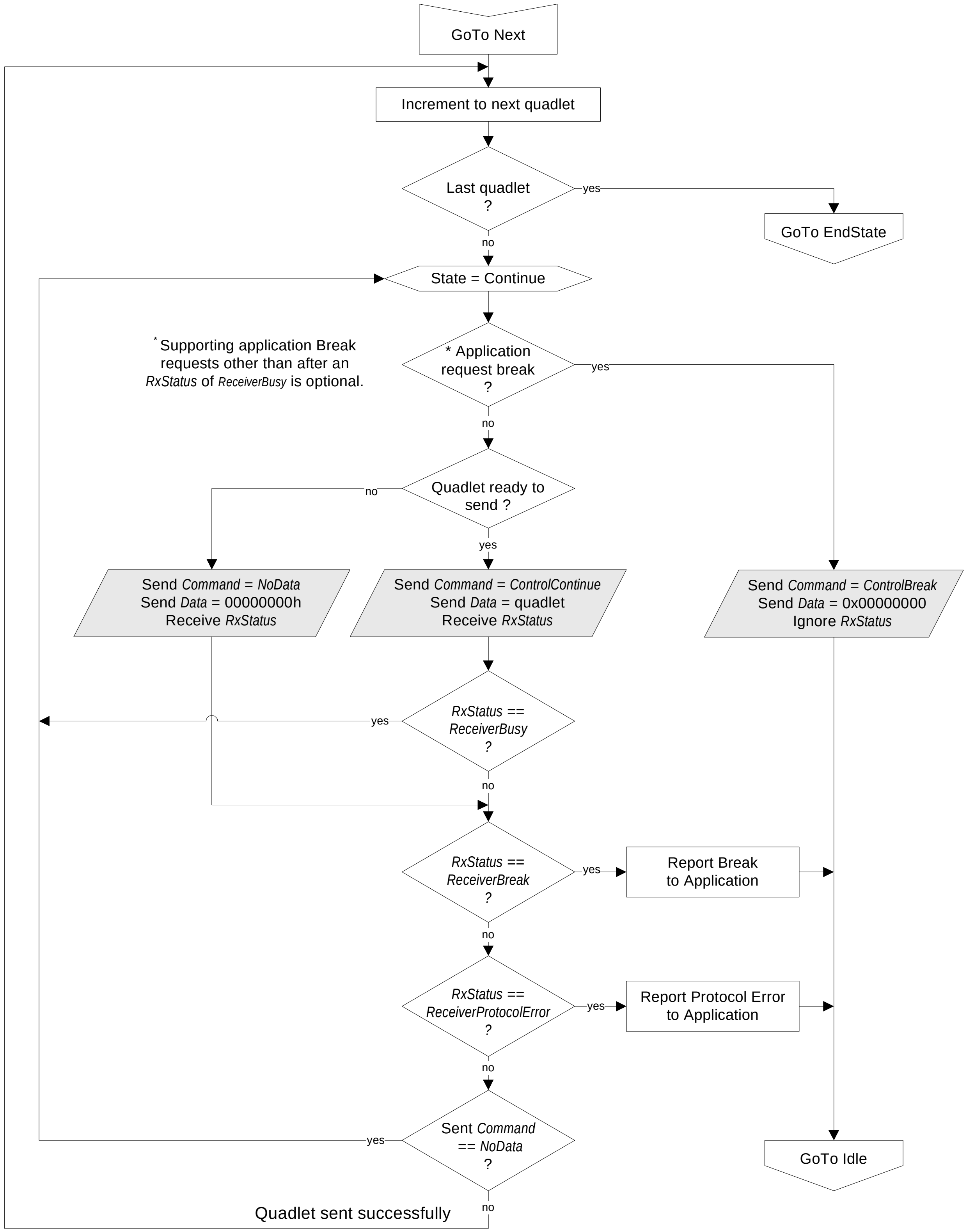

GoTo Next
Increment to next quadlet
Last quadlet ?
yes
GoTo EndState
no
State = Continue
* Supporting application Break requests other than after an RxStatus of ReceiverBusy is optional.
* Application request break ?
yes
no
Quadlet ready to send ?
no
yes
Send Command = NoData
Send Data = 00000000h
Receive RxStatus
Send Command = ControlContinue
Send Data = quadlet
Receive RxStatus
Send Command = ControlBreak
Send Data = 0x00000000
Ignore RxStatus
RxStatus == ReceiverBusy ?
yes
no
RxStatus == ReceiverBreak ?
yes
Report Break to Application
no
RxStatus == ReceiverProtocolError ?
yes
Report Protocol Error to Application
no
Sent Command == NoData ?
yes
no
GoTo Idle
Quadlet sent successfully

**Figure 3-7: Control Packet Tx Device Protocol: Middle**

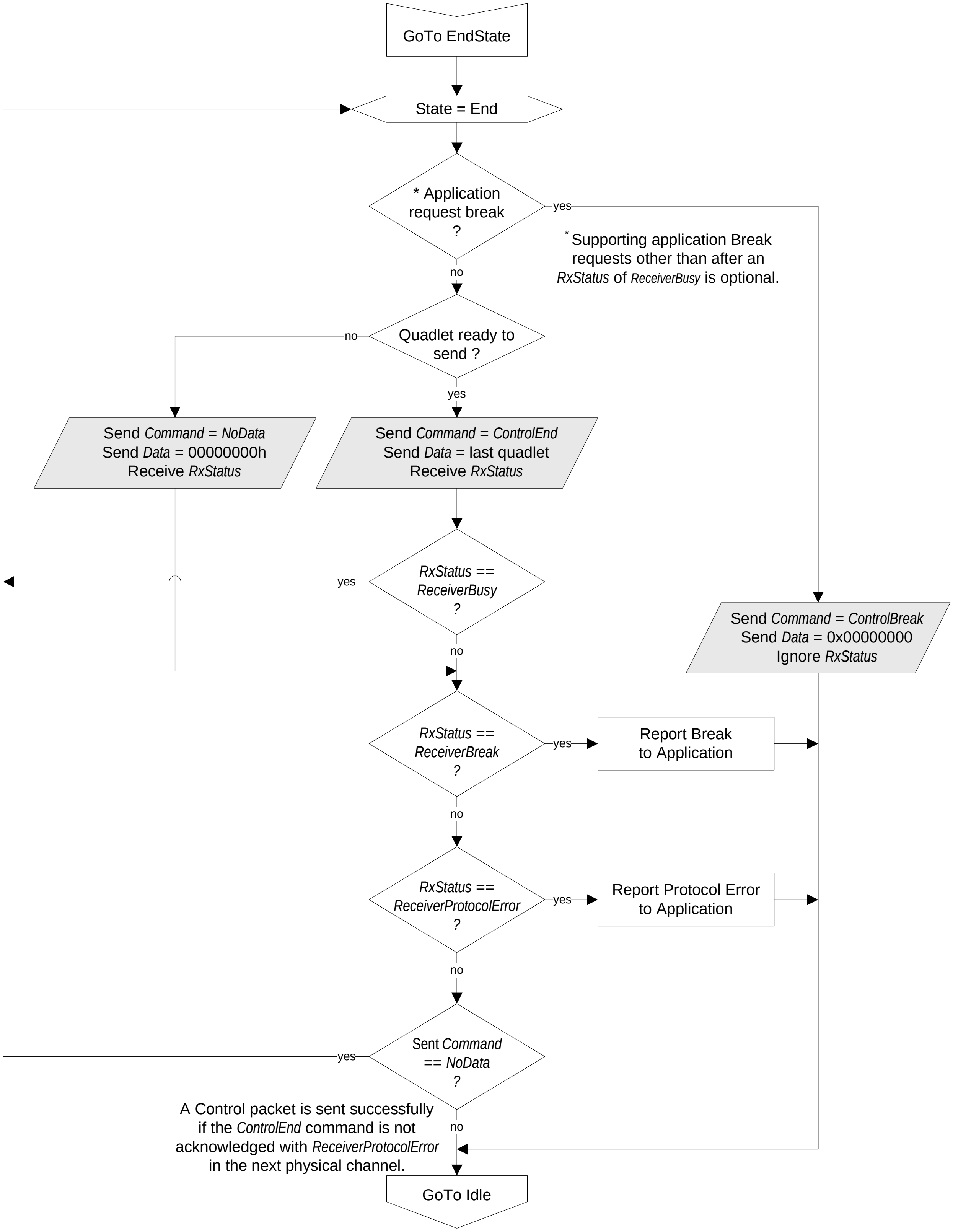

GoTo EndState
State = End
* Application request break ?
yes
no
* Supporting application Break requests other than after an RxStatus of ReceiverBusy is optional.
Quadlet ready to send ?
no
yes
Send Command = NoData
Send Data = 00000000h
Receive RxStatus
Send Command = ControlEnd
Send Data = last quadlet
Receive RxStatus
RxStatus == ReceiverBusy ?
yes
no
Send Command = ControlBreak
Send Data = 0x00000000
Ignore RxStatus
RxStatus == ReceiverBreak ?
yes
no
Report Break to Application
RxStatus == ReceiverProtocolError ?
yes
no
Report Protocol Error to Application
Sent Command == NoData ?
yes
no
A Control packet is sent successfully if the ControlEnd command is not acknowledged with ReceiverProtocolError in the next physical channel.
GoTo Idle

**Figure 3-8: Control Packet Tx Device Protocol: End**

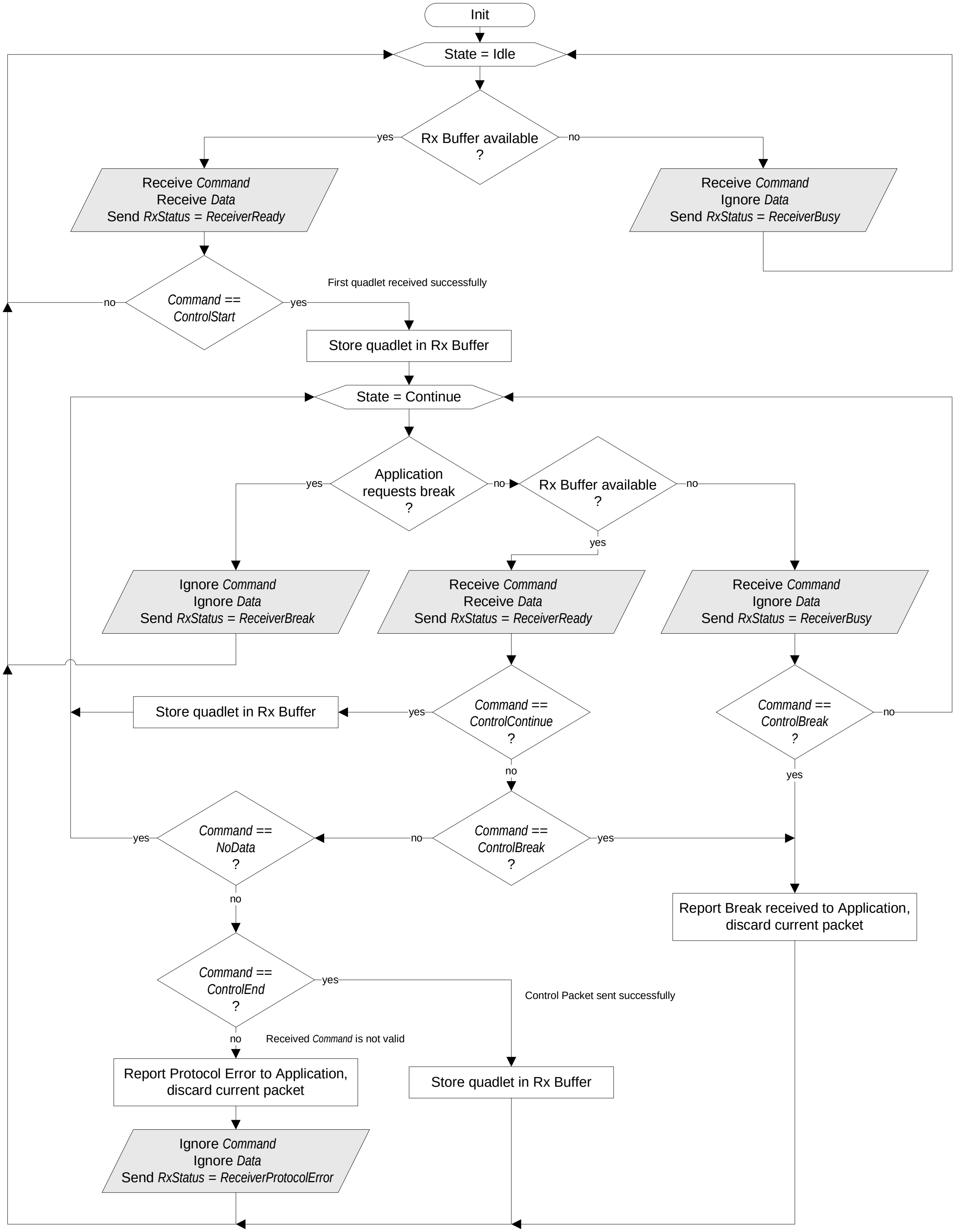

Init
State = Idle
Rx Buffer available ?
yes
no
Receive Command
Receive Data
Send RxStatus = ReceiverReady
Receive Command
Ignore Data
Send RxStatus = ReceiverBusy
Command == ControlStart
no
yes
First quadlet received successfully
Store quadlet in Rx Buffer
State = Continue
Application requests break ?
yes
no
Rx Buffer available ?
no
yes
Ignore Command
Ignore Data
Send RxStatus = ReceiverBreak
Receive Command
Receive Data
Send RxStatus = ReceiverReady
Receive Command
Ignore Data
Send RxStatus = ReceiverBusy
Store quadlet in Rx Buffer
yes
Command == ControlContinue ?
no
Command == ControlBreak ?
no
yes
yes
Command == NoData ?
no
Command == ControlBreak ?
yes
no
Report Break received to Application, discard current packet
Command == ControlEnd ?
yes
Control Packet sent successfully
no
Received Command is not valid
Report Protocol Error to Application, discard current packet
Store quadlet in Rx Buffer
Ignore Command
Ignore Data
Send RxStatus = ReceiverProtocolError

**Figure 3-9: Control Packet Rx Device Protocol**

## 3.9.2 Synchronous

Synchronous stream data is sent in a continuous and broadcast fashion, without block information. Therefore, receiving Devices must not respond to the synchronous command; thereby leaving *RxStatus* in the *ReceiverReady* state (logic low). For MediaLB 3/5-pin, the required pull-down on *MLBS* leaves this signal in the *ReceiverReady* command when no synchronous data is transmitted on the *MLBD* line. For MediaLB 6-pin, the *ReceiverReady* command is also the default *MLBS* state. This is a result of the logic low differential input voltage offset at the MediaLB receivers (see Section 2.3.2) when no Device is transmitting. Data is sent MSB first, left aligned in the quadlet, and left channel first for stereo data.

Figure 3-10 illustrates the synchronous data formats for MediaLB 3/5-pin. For stereo 24-bit data, two physical channels (PCn) are needed per frame where the data is packed and left-justified in the two quadlets. In the 32-bit sequential format, data fills the entire quadlet with the internal data format determined by the system implementor. For MediaLB 6-pin, the synchronous data structure shown in Figure 3-10 is also applicable, with the exception of the additional turnaround cycles on *MLBS* and *MLBD*.

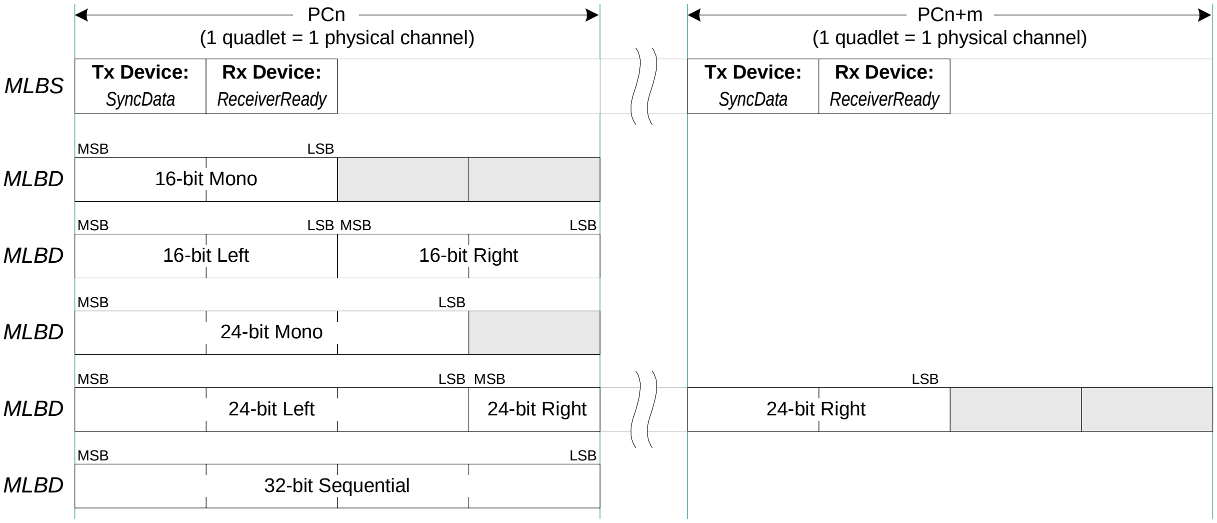


**Figure 3-10: MediaLB 3/5-pin Synchronous Data Structure**

The synchronous flow for a Tx Device is illustrated in Figure 3-11. The data transfer blocks (slanted rectangle shapes) occur only during a physical channel (PCn) associated with the logical channel defined by a single *ChannelAddress*. The flow diagram contains only one state: Transmit. Once a channel has been initialized, the Transmit state is entered. If a Tx Device has no data to transmit, it must still send the *SyncData* command and set the actual data to a safe value, such as all zeros. To stop sending synchronous data, the logical channel must be eliminated (*ChannelAddress* removed from MediaLB).

The synchronous flow for an Rx Device is illustrated in Figure 3-12. The flow diagram also contains only one state, Continue, where the Rx Device waits for data from the Tx Device. No command other than *SyncData* is expected or allowed. When the *SyncData* command is detected, the corresponding data quadlet sent with the command is received and stored in the Rx buffer. Any command received, other than *SyncData*, is a *ProtocolError* and should be reported to the application. Furthermore, the data quadlet received with the invalid command is discarded and replaced with a safe value.

Since the default bus state is *ReceiverReady*, the Rx Device must not drive the *MLBS* line with *RxStatus* since *ReceiverReady* is the only allowable response for synchronous data. The system stops the transfer of synchronous data by eliminating the logical channel (*ChannelAddress*) from the bus. If an Rx Device does not receive its *ChannelAddress* in the frame, it should assume that the channel is not setup yet, or that the logical channel has been eliminated and should respond accordingly (for example, mute outputs).

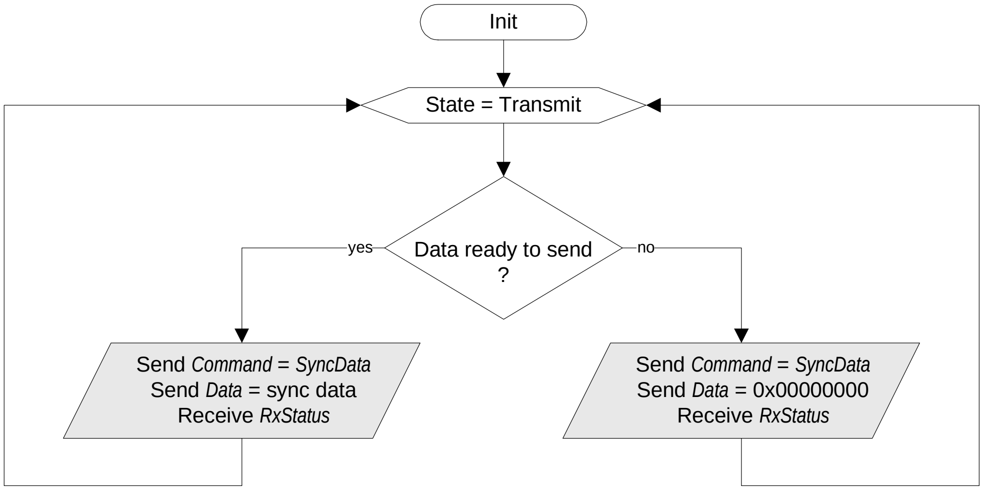


**Figure 3-11: Synchronous Data Tx Device Protocol**

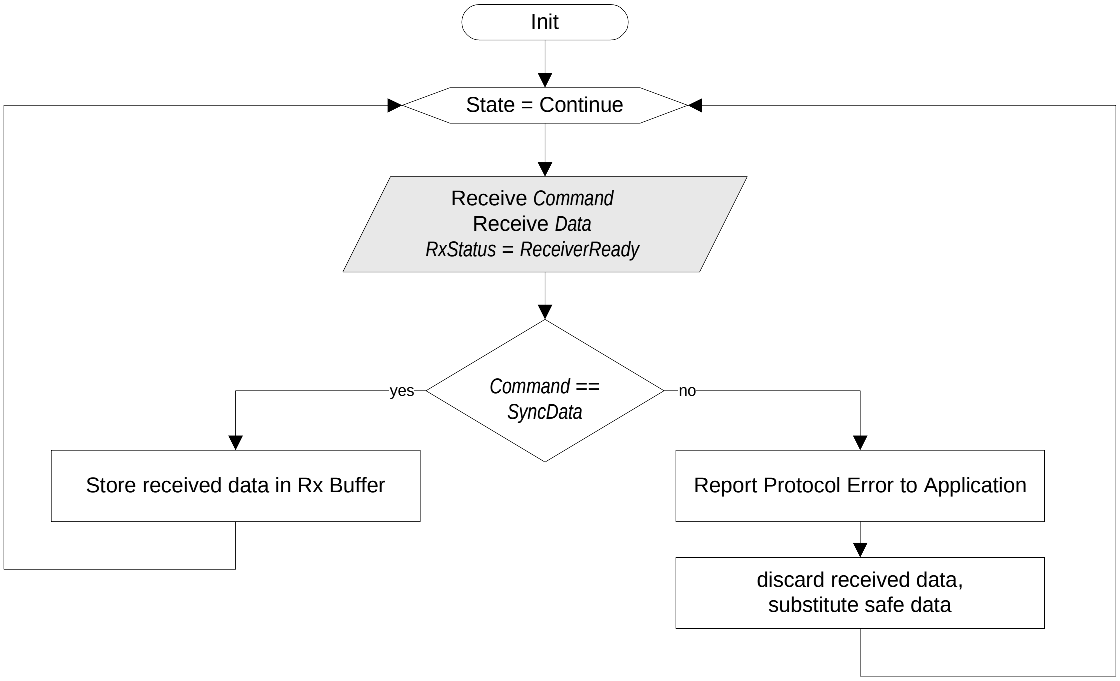


**Figure 3-12: Synchronous Data Rx Device Protocol**

## 3.9.3 Isochronous

Isochronous data is sent in a streaming fashion, similar to synchronous data. However, the isochronous commands indicate the start of a block and how many bytes are valid in the concurrent transmitted quadlet. Valid bytes are left-justified in the quadlet, as illustrated in Figure 3-13. When isochronous data is being transported (channel active), but no data is available for the current quadlet, the *IsoNoData* command is sent by the Tx Device. For MediaLB 6-pin, the isochronous data structure shown in Figure 3-13 is also applicable, with the exception of the additional turnaround cycles on *MLBS* and *MLBD*.

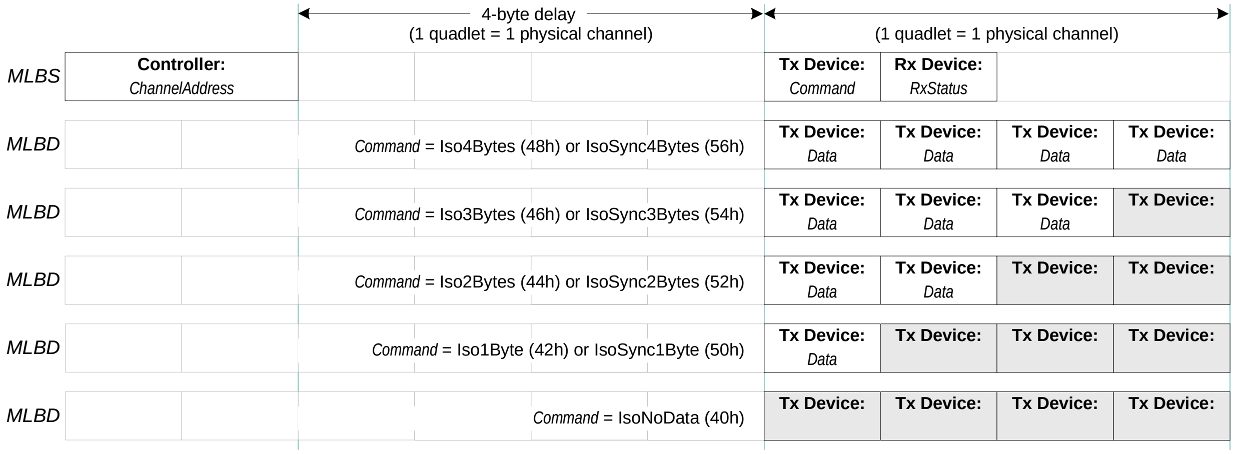


**Figure 3-13: MediaLB 3/5-pin Isochronous Data Structure**

The isochronous flow for a Tx Device is illustrated in Figure 3-14. The data transfer blocks (slanted rectangle shapes) occur only during a physical channel (PCn) associated with the logical channel defined by a single *ChannelAddress*. Similar to the synchronous flow, isochronous data immediately starts transmitting. When data exists from the application, the *IsoSync?Bytes* commands are used to indicate the start of a block, which provides alignment information to the Rx Device. The *Iso?Bytes* commands indicate the middle of a block of data. The definition of *block* for isochronous data is outside the scope of this document. For physical channels that transfer less than four bytes, the Rx Device must only use/store the number of valid bytes, and ignore the unused portion.

The isochronous flow for an Rx Device is illustrated in Figure 3-15. The *NoData* command indicates that the channel is not setup yet. Once an isochronous channel is setup, the Rx Device continually receives the channel data, similar to synchronous data. The only two valid responses for an isochronous channel are *ReceiverBusy*, and the default bus state of *ReceiverReady*. Although Rx Devices can respond with *ReceiverBusy*, its use should be minimized, since Tx Devices may not be able to store much isochronous data that gets backed up due to the *ReceiverBusy* responses. If any Rx Device uses *ReceiverBusy*, then only one Rx Device is allowed. If all targeted Rx Devices do not drive *RxStatus* (default *ReceiverReady* response), then the isochronous stream can support multiple Rx Devices (broadcast).

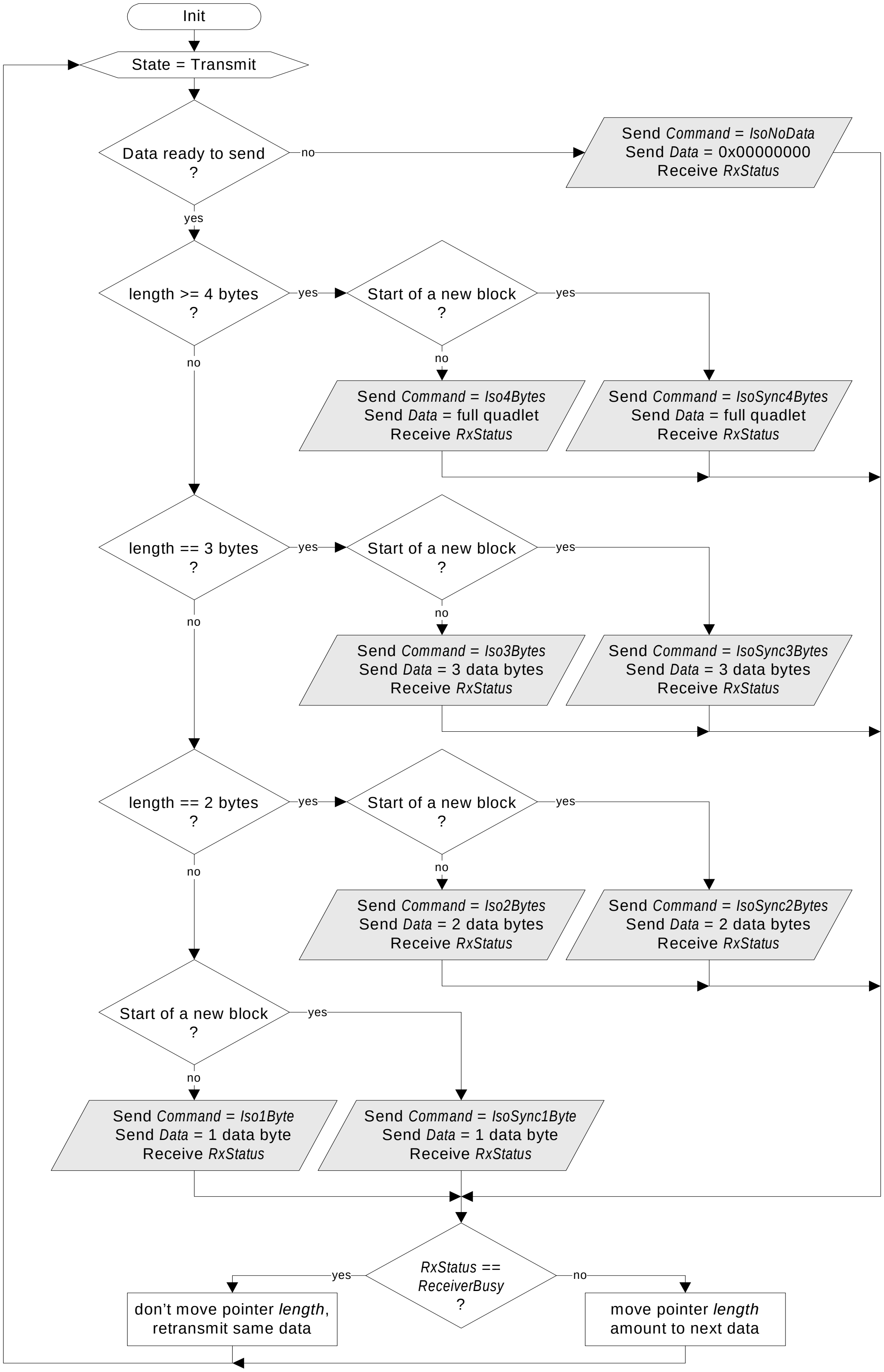


**Figure 3-14: Isochronous Data Tx Device Protocol**

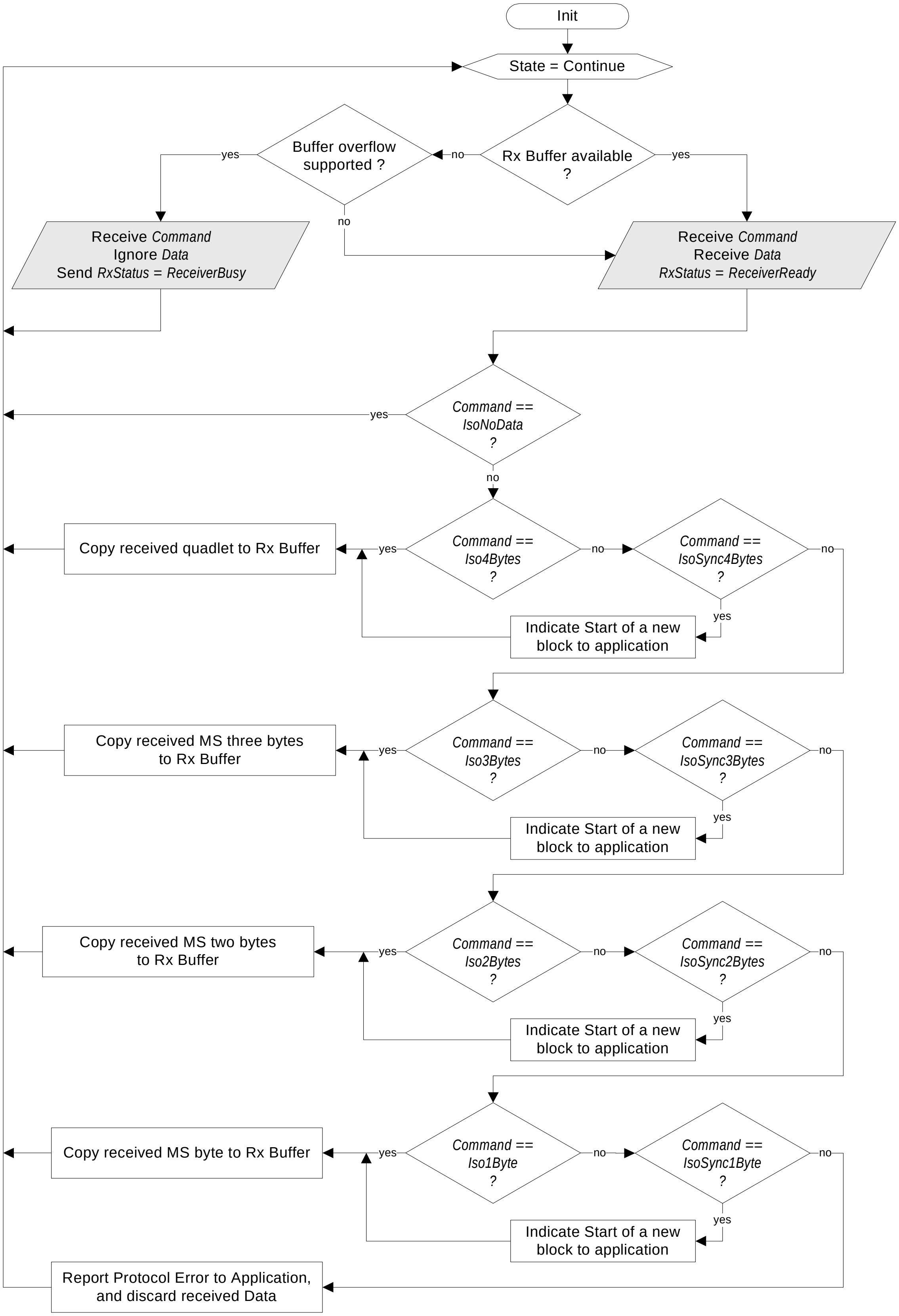

Init
State = Continue
Rx Buffer available ?
Buffer overflow supported ?
yes
no
yes
no
Receive *Command*
Ignore *Data*
Send *RxStatus* = *ReceiverBusy*
Receive *Command*
Receive *Data*
*RxStatus* = *ReceiverReady*
*Command* == *IsoNoData* ?
yes
no
*Command* == *Iso4Bytes* ?
*Command* == *IsoSync4Bytes* ?
Copy received quadlet to Rx Buffer
Indicate Start of a new block to application
*Command* == *Iso3Bytes* ?
*Command* == *IsoSync3Bytes* ?
Copy received MS three bytes to Rx Buffer
Indicate Start of a new block to application
*Command* == *Iso2Bytes* ?
*Command* == *IsoSync2Bytes* ?
Copy received MS two bytes to Rx Buffer
Indicate Start of a new block to application
*Command* == *Iso1Byte* ?
*Command* == *IsoSync1Byte* ?
Copy received MS byte to Rx Buffer
Indicate Start of a new block to application
Report Protocol Error to Application, and discard received Data

**Figure 3-15: Isochronous Data Rx Device Protocol**

# 4 Compliance

The MediaLB specification is targeted towards many levels of chip complexity and native intelligence. Therefore, different levels of implementation are allowed to support MediaLB and still remain compliant to this specification. The Physical Layer portion of this specification must be met by all Devices for whichever speeds a particular Device supports. All MediaLB Devices must support the rules for synchronization to MediaLB.

For MediaLB Controllers, all *System* commands are optional, including support for dynamic system configuration and *DeviceAddresses*.

For MediaLB Devices, support for all transport methods is optional. If a MediaLB Device supports a particular transport method, it must fully support it including all *Command* bytes and *RxStatus* responses associated with that transport method. For asynchronous and control methods, the Protocol error responses can be expanded for additional error checking, based on specific implementations. Any extra error checking that causes a Protocol error to be transmitted must be listed in the Device documentation.

For MediaLB Devices, support for *System* responses and dynamic configuration are optional. If dynamic configuration is supported, it must comply with the specifications listed in this document.

All MediaLB Devices must specify clearly in documentation what MediaLB speeds, *System* commands, and transport methods they support. In addition, MediaLB Devices must clearly state the *DeviceAddress* as well as the Index and associated transport method used in configuring the *ChannelAddress*.

# Appendix A: Applications Information

The information in this chapter is provided as guidelines and should not be construed as part of the actual MediaLB specification. This information should be tested and adapted as needed to a particular design.

## A.1 Application-Specific Requirements

In addition to the MediaLB Link Layer requirements, MediaLB Controllers may have other criteria that should be considered when implementing a MediaLB interface.

As an example, the MOST *Intelligent Network Interface Controller* (INIC) Transceiver chips, which also function as MediaLB Controllers, transfer control messages and asynchronous packets across MediaLB using a Port Message Protocol. To support the INIC MediaLB Controllers and the Port Message Protocol efficiently, MediaLB Devices should integrate the optimal amount of buffers needed to handle control messages and asynchronous packets.

The MOST25 and MOST50 INICs (OS81050/60 [1,2] and OS81082/92 [3,4]) transfer control messages as large as 64-bytes and asynchronous MOST Data Packets (MDPs) as large as 1040-bytes over the MediaLB interface. For efficient data transfer, MediaLB Devices communicating with the OS81050 and OS81082 should be prepared to buffer these control messages and asynchronous MDPs with the aforementioned sizes.

The MediaLB Controller for the MOST150 INIC (OS81110 [5]) is capable of transferring Control Messages with a maximum size of 64-bytes and asynchronous MDPs as large as 1040-bytes over the MediaLB interface. Along with these, the MOST150 INIC also supports asynchronous MOST Ethernet Packets (MEPs) as large as 1536-bytes. MediaLB Devices utilizing an OS81110 should incorporate buffers to handle the aforementioned message and packet sizes.

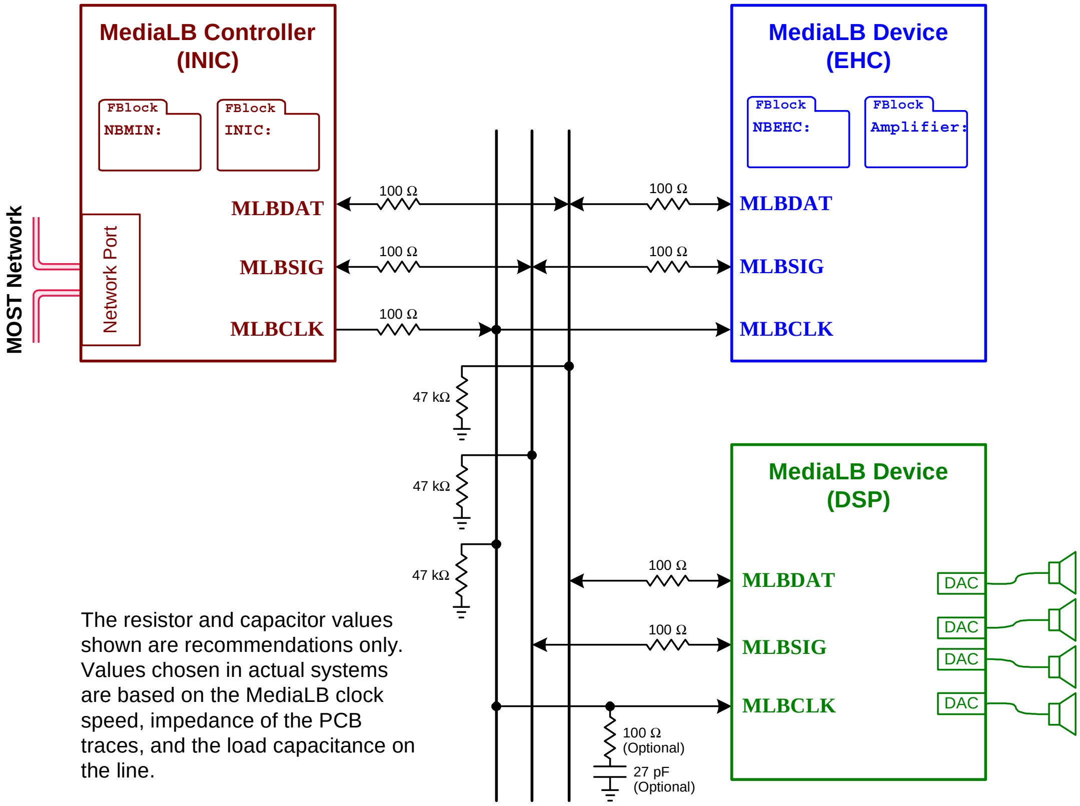
For further details about the size of control messages and asynchronous packets transferred by the MOST INIC Transceiver chips refer to the INIC datasheets.

## A.2 MediaLB Configuration Example

The following example provides information about connecting and configuring a MediaLB 3-pin and MediaLB 6-pin system, which incorporates a MOST INIC Transceiver chip as a MediaLB Controller. For *ChannelAddress* assignment a static configuration is used. The data and commands listed to configure the MediaLB Controller are intended to give the system architect an overview and are not meant to show a complete design. Additional commands may be needed in a real implementation.

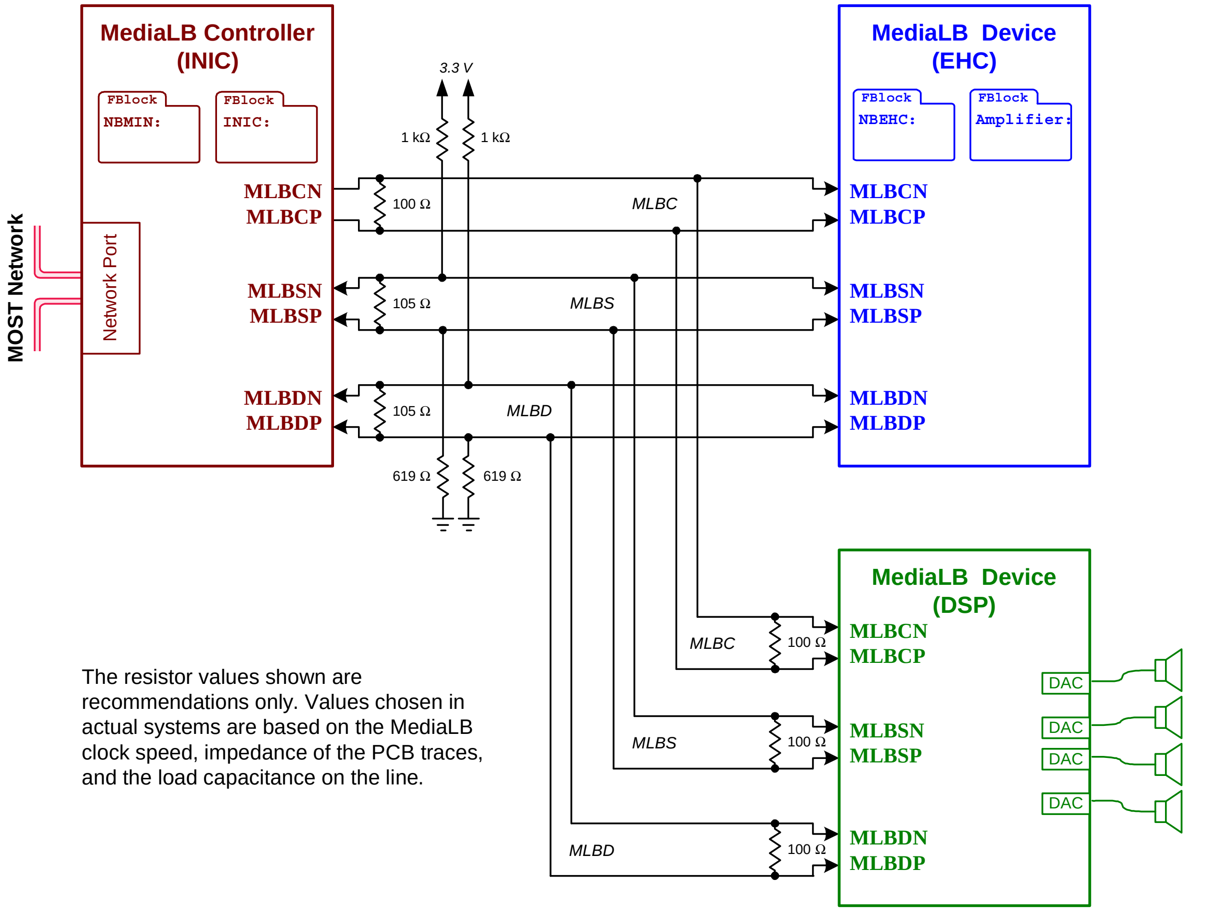
This example consists of three MediaLB Devices: The MediaLB Controller, an External Host Controller (EHC), and a Digital Signal Processor (DSP). Figure A-1 illustrates a MediaLB 3-pin application setup and Figure A-2 a MediaLB 6-pin application setup.

- The MediaLB Controller used in this example is an OS81050 [1] for the MediaLB 3-pin diagram and an OS81110 [5] for the MediaLB 6-pin diagram.
- The EHC configures the MediaLB Controller, allocates MediaLB bandwidth, and configures the DSP. The EHC also manages the MOST control messages and contains two function blocks: FBlock `NetBlock` and FBlock `Amplifier` (which manages the DSP channels).
- The DSP communicates with the EHC and supports a 4-channel (two stereo pairs) sink for 16-bit streaming data from the MOST Network. The EHC manages the DSP directly; therefore, the DSP does not contain any function blocks.



The resistor and capacitor values shown are recommendations only. Values chosen in actual systems are based on the MediaLB clock speed, impedance of the PCB traces, and the load capacitance on the line.

**Figure A-1: MediaLB 3-pin Example Connection Diagram**



The resistor values shown are recommendations only. Values chosen in actual systems are based on the MediaLB clock speed, impedance of the PCB traces, and the load capacitance on the line.

**Figure A-2: MediaLB 6-pin Example Connection Diagram**

In this example, MediaLB is configured statically. The following fixed *ChannelAddress* (CA) assignment is defined:

- CA 0x0002, 0x0004 - This logical channel pair is used for opening other logical channels on MediaLB and communicating control messages between the EHC and the Controller. These *ChannelAddresses* are opened automatically by the Controller when static MediaLB configuration is selected. The MediaLB Controller listens to *ChannelAddress* 0x0004 for configuration information from the EHC, and responds to the EHC on *ChannelAddress* 0x0002.
- CA 0x0030 - This logical channel is configured by the EHC for transmitting synchronous data (one stereo pair) to the DSP from the MOST Network (via the Controller).
- CA 0x0032 - This logical channel is configured by the EHC for transmitting synchronous data (a second stereo pair) to the DSP from the MOST Network (via the Controller).
- CA 0x0034, 0x0036 - These logical channels are opened by the EHC for communication with the DSP. The DSP listens to *ChannelAddress* 0x0034 for information from the EHC, and responds to the EHC on *ChannelAddress* 0x0036.

DSP firmware is internally configured to use the following *ChannelAddresses*: 0x0030, 0x0032, 0x0034, and 0x0036. The EHC manages the MOST Network Control messages and connects the MediaLB logical channels to the MOST stream data when needed.

For the MediaLB 3-pin system example it is assumed that the OS81050 Controller power-up defaults are MediaLB 3-pin mode with 256×Fs clock speed. In the case of the MediaLB 6-pin system, the OS81110 opens the MediaLB 6-pin port with 2048×Fs clock speed by default.

According to the *OS81050 Data Sheet* [1] and the *OS81110 Data Sheet*[5], the Controller opens 0x0002 and 0x0004 logical channels to communicate with the EHC. By sending messages to the Controller, the EHC opens and configures the rest of the MediaLB logical channels (e.g. 0x0030, 0x0032, 0x0034, 0x0036).

> The control messages sent by the EHC in this example are intended to show the basic configuration and connection mechanisms, and should not be construed as a complete implementation. For more information on control messages, refer to the *INIC API User's Manuals* [6,7,8,9].

The Control message format across MediaLB matches that of the *MOST Specification* [10] and is defined as:

```
DeviceID.FBlockID.InstID.FktID.OpType(Parameters)
```

After power-up or reset, the MediaLB Controller is in *Protected Mode*. While the Controller is in this mode, MediaLB Devices are not visible to the MOST Network and the EHC can not configure the MediaLB port of the Controller for further communication. In order to exit Protected Mode, the EHC sends the following message to the INIC, using logical channel 0x0004 (the `DeviceID` is removed for brevity):

```
INIC.00.EHCIState.Set(EHCI_SemiProtected)
```

This control message places the MediaLB Controller in *SemiProtected Mode*, which allows the EHC application to configure the INIC and the MediaLB port. Once in *SemiProtected Mode*, the MediaLB Controller is placed in *Attached Mode*, which makes the MediaLB Devices visible to the MOST Network. In order to enter *Attached Mode*, the EHC sends the following message to the Controller:

```
INIC.00.EHCIState.Set(EHCI_Attached)
```

After the INIC is in *Attached Mode* the following message is sent to the Controller through logical channel 0x0004 to setup a MediaLB streaming data socket:

```
INIC.00.CreateSocket.StartResult(MediaLBPort,Out,StreamData,0x0004,0x0030)
```

where:

- `MediaLBPort` is the Port ID for MediaLB [6,9],
- `Out` is the direction of data flow relative to the Controller (Controller to DSP),
- `StreamData` specifies the type of data to be transported on the channel,
- `0x0004` is the block width (or logical channel width) in bytes, and
- `0x0030` is the *ChannelAddress* associated with the logical channel being created.

This message will return a socket handle (`DSP1SocketHandle`), used to connect to incoming synchronous data from the MOST Network.

In order to setup the second logical channel for the DSP, a similar message is sent:

```
INIC.00.CreateSocket.StartResult(MediaLBPort,Out,StreamData,0x0004,0x0032)
```

This message will return a second socket handle (`DSP2SocketHandle`), used to connect to incoming synchronous data from the MOST Network. For this example, the DSP logical channels for stereo data are 16-bits each, which fits into one MediaLB physical channel (four bytes).

An additional pair of logical channels for communication between the EHC and the DSP is also needed. No socket is necessary for these logical channels, as the Controller is not involved in the actual transportation of the data between the EHC and DSP. The Controller is only used to reserve MediaLB bandwidth for this inter-MediaLB Device communication, using the following messages:

```
INIC.00.MediaLBAllocateOnly.StartResult(0x0034,0x0004)
INIC.00.MediaLBAllocateOnly.StartResult(0x0036,0x0004)
```

One logical channel is created for the EHC to DSP communications (*ChannelAddress* 0x0034), and one is created for the DSP to EHC communications (*ChannelAddress* 0x0036). The first parameter indicates the *ChannelAddress* used for the logical channel. The last parameter determines the number of MediaLB bytes to attach to the logical channel.

This example does not show the creation of two stereo streaming data sockets on the MOST Network, but assumes they are already created with the handles `NET1SocketHandle` and `NET2SocketHandle`. To attach the MOST Network and the MediaLB DSP streaming data sockets, the following messages are used:

```
INIC.00.ConnectSockets.StartResult(NET1SocketHandle, DSP1SocketHandle)
INIC.00.ConnectSockets.StartResult(NET2SocketHandle, DSP2SocketHandle)
```

These messages route the MOST Network synchronous data to the outgoing DSP logical channels (0x0032 and 0x0034). Once the sockets are connected, the streaming data from the MOST Network flows through the MediaLB Controller to the appropriate DSP logical channels.

An example MediaLB interface timing diagram is shown in Figure A-3, on page 57 for MediaLB 3-pin and Figure A-4, on page 58 for MediaLB 6-pin. These diagrams depict a system that is fully configured by the EHC with all available logical channels operational. The color associated with the data (on *MLBS* and *MLBD*) determines which Device is transmitting. Black indicates general MediaLB Controller data (i.e. *System* commands).

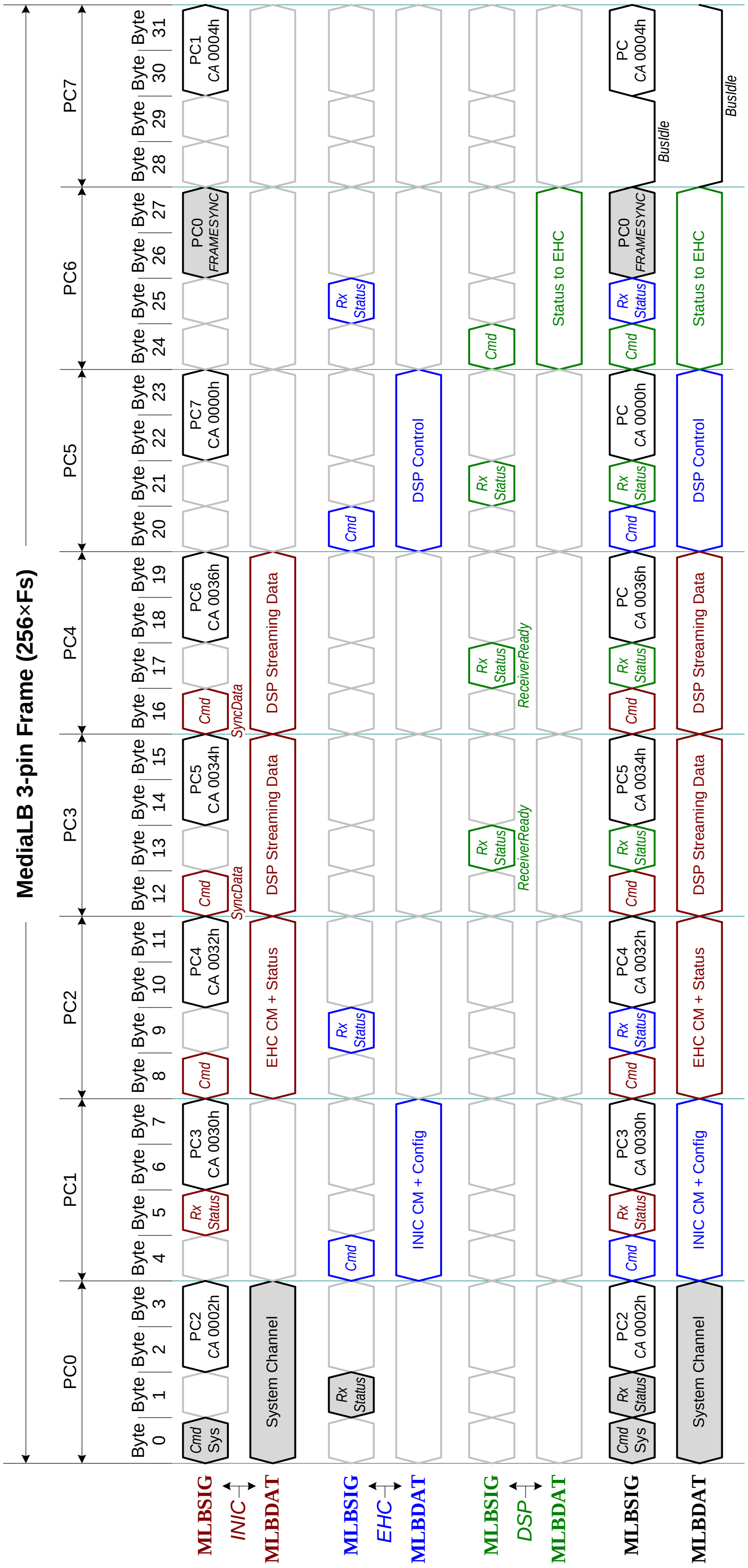


Figure A-3: MediaLB 3-pin Application Timing Example

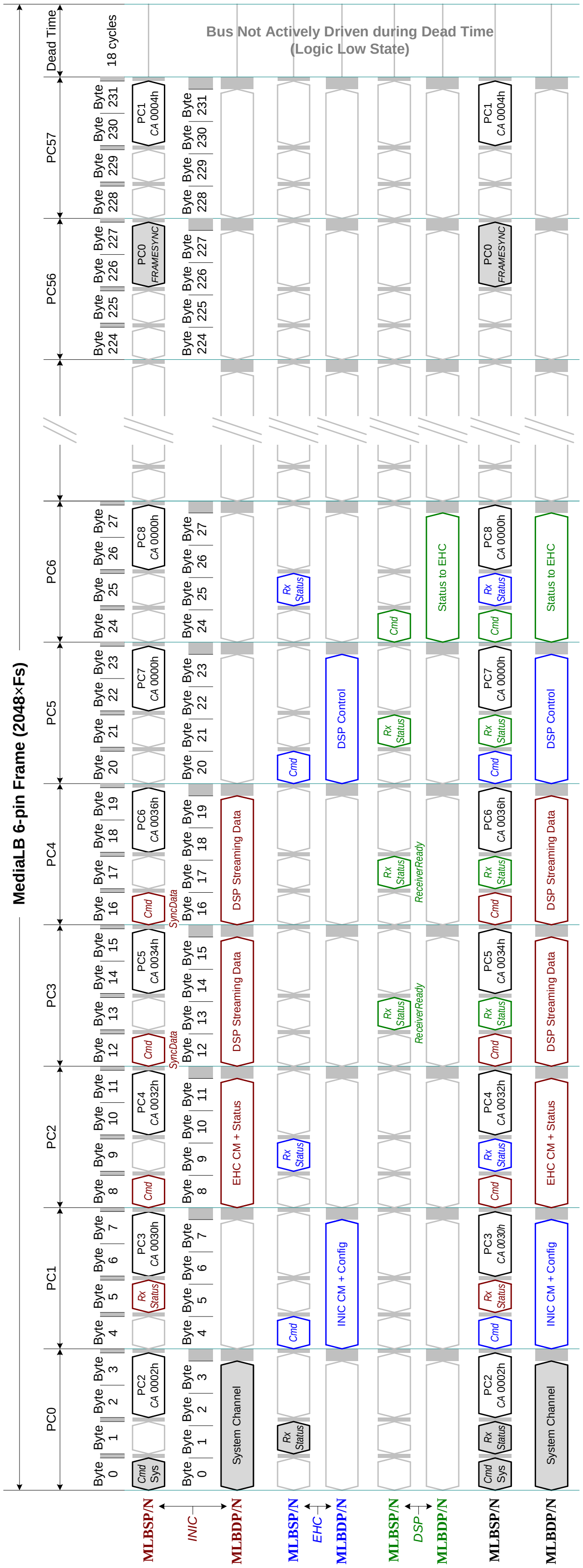


Figure A-4: MediaLB 6-pin Application Timing Example

# A.3 MediaLB Termination

## A.3.1 MediaLB 3-pin

The MediaLB 3-pin lines require pull-down resistors to keep the signals in a known state when no Device is driving the lines, as shown in Figure A-1 on page 53. The **MLBSIG** and **MLBDAT** lines should have pull-down resistors weak enough for the line to hold a logic high for the bus hold time, $t_{mdzh}$. This hold time is needed on **MLBSIG** and **MLBDAT** to allow Rx Devices time to latch the data before the logic high level is drained off. Since leakage should be kept to a minimum to maintain the bus hold time, conformal coating is recommended for systems where condensation can occur.

Series resistors can also be used at each Device for series termination and to limit rise and fall times, thereby minimizing the EMI profile. The value chosen should be based on the intended MediaLB clock speed, the impedance of the PCB traces, and capacitive loading on the actual lines. For the highest MediaLB speed where the timing requirements are more stringent, special care should be taken regarding the size or use of series resistors to guarantee that the timing specifications are not violated.

In addition to a series resistor at the Controller, the **MLBCLK** line should have a weak pull-down resistor to keep the signal in a known state at startup as well as when the **MLBCLK** is shut down. Due to the speed of **MLBCLK**, it is recommended that a parallel AC termination option be placed at the farthest Device from the Controller to minimize reflections as needed.

## A.3.2 MediaLB 6-pin

The MediaLB 6-pin implementation requires source and sink terminations on each differential pair to reduce reflections. The value of these terminations should match the characteristic line impedance. The MediaLB 6-pin Controller also requires pull-up and pull-down resistors to retain the common-mode voltage at a steady value when the bus is not being driven (only for *MLBS* and *MLBD*). The values of these resistors are chosen so that a logic low is present during an idle state. Figure A-5 illustrates a typical MediaLB 6-pin differential pair termination scheme.

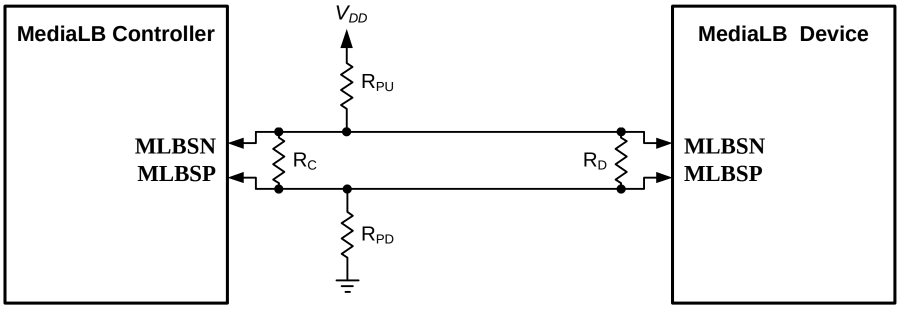


**Figure A-5: MediaLB 6-pin Termination Scheme**

$R_D$ is the termination resistance at the MediaLB Device and its value matches the characteristic impedance of the PCB traces. The termination at the Controller also matches that of the Device. Using parallel resistance and Kirchhoff's current law yields the following three equations:

$$1:\ R_C = \frac{R_D \times (R_{PU} + R_{PD})}{R_{PU} + R_{PD} - R_D}$$

$$2:\ R_{PU} = -\frac{\frac{R_C \times R_D}{R_C + R_D} \times \left(V_{DD} - V_{OCM} + \frac{V_{ID}}{2}\right)}{V_{ID}}$$

$$3:\ R_{PD} = -\frac{\frac{R_C \times R_D}{R_C + R_D} \times \left(V_{OCM} + \frac{V_{ID}}{2}\right)}{V_{ID}}$$

As an example, using these known values:
$R_D$ = 100 Ω, $V_{DD}$ = 3.3 V, $V_{OCM}$ = 1.2 V, and $V_{ID}$ = -100 mV (logic-low idle state),

Yields the following results:
$R_C$ ≈ 105 Ω, $R_{PU}$ ≈ 1 kΩ, and $R_{PD}$ ≈ 619 Ω.

Figure A-2, on page 54 shows a typical MediaLB 6-pin multi-node setup. Note that the termination for the Devices is located at the furthest Device from the Controller.

# A.4 MediaLB Design

Due to the high-speed signals of the MediaLB interface, designers must be particularly aware of the delays associated with designing logic to accommodate the MediaLB data rate, especially at higher speeds. On-chip logic must also be designed to match the delays between *MLBC* and the *MLBS*/MLBD signals.

## A.4.1 MediaLB 3-pin

Figure A-6 illustrates a typical MediaLB 3-pin interface between a Tx Device and a Rx Device (with termination removed for clarity). For this discussion, the **MLBSIG** and **MLBDAT** lines are both considered data lines being clocked by **MLBCLK**. All delays related to getting these signals on and off chip, as well as to the intended latches, must be considered. In Figure A-6, the delays are as follows:

- $t_{inpad}$ - time required to get a signal through the input pad.
- $t_{logA}$ - delay to get through the clock-tree synthesis logic on **MLBCLK.**
- $t_{ckQ}$ - clock to Q output delay of the flip-flop.
- $t_{logB}$ - delay due to output pin multiplexing and routing on **MLBSIG/MLBDAT.**
- $t_{logC}$ - delay due to input pin buffering and routing on **MLBSIG/MLBDAT.**
- $t_{outpad}$ - time required to get a signal through the output pad on **MLBSIG/MLBDAT.**
- $t_{Dck}$ - D flip-flop setup to clock going low delay.
- $t_{mdatd}$ and $t_{mckd}$ - on-board routing delays, which are important to keep matched (minimize skew), and can be a significant factor when using MediaLB 3-pin at the 1024×Fs speed.

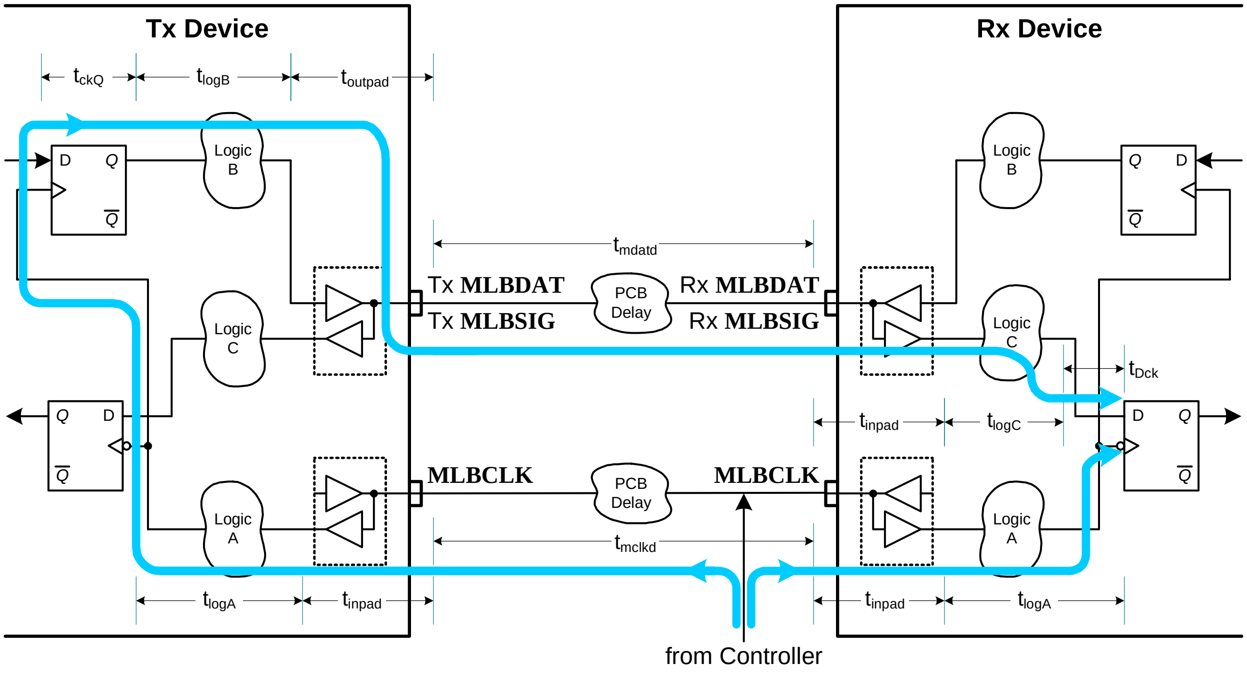


**Figure A-6: MediaLB 3-pin Gate-Level Signal Delays**

The MediaLB Tx Device utilizes a clock transmitted from the controller (over **MLBCLK**) to latch the data/signal onto **MLBSIG** and **MLBDAT**. The MediaLB Rx Device latches the data/signal from **MLBSIG** and **MLBDAT** utilizing the same clock from the controller. Since data/signal is driven at the rising edge of clock and captured on the falling edge of clock, the delay and setup times must add up to less than the clock high time ($t_{mckh}$).

$$t_{mckh} \geq t_{mckd} + t_{inpad} + t_{logA} + t_{ckQ} + t_{logB} + t_{outpad} + t_{mdatd} + t_{inpad} + t_{logC} + t_{Dck} - t_{inpad} - t_{logA}$$

Assuming the on-board routing delays are the same for clock and data/signal:

$$t_{prop} = t_{mckd} = t_{mdatd}$$

where $t_{prop}$ is the on-board trace delay (approximately 6.3 ps per mm of FR4).

The Tx Device delay from receiving a rising edge on **MLBCLK** to transmitting a valid data/signal on **MLBSIG** and **MLBDAT** is defined as $t_{delay}$.

$$t_{delay} = t_{inpad} + t_{logA} + t_{ckQ} + t_{logB} + t_{outpad}$$

The Rx Device must be able to latch the incoming data/signal bits $t_{dsmcf}$ before **MLBCLK** goes low.

$$t_{dsmcf} \geq t_{inpad} + t_{logC} + t_{Dck} - t_{inpad} - t_{logA} \quad \text{or}$$

$$t_{dsmcf} \geq t_{logC} + t_{Dck} - t_{logA}$$

Substituting into the first equation results in:

$$t_{mckh} \geq t_{delay} + t_{dsmcf} + 2t_{prop}$$

The maximum $t_{delay}$ is given as:

$$t_{delay(max)} = t_{mckh} - t_{dsmcf(min)} - 2t_{prop}$$

Using recommended maximum $t_{delay}$ results in the following $t_{prop}$ and trace length values given in Table A-1.

| MediaLB 3-pin Clock Speed | $t_{delay(max)}$ (ns) | $t_{prop}$ (ns) | Trace Length (mm) |
|---|---|---|---|
| 256×Fs | 20 | 4.5 | 714.3 |
| 512×Fs | 10 | 1.5 | 238.1 |
| 1024×Fs | 7 | 0.65 | 103.2 |

**Table A-1: Recommended MediaLB 3-pin $t_{delay}$, $t_{prop}$, and Trace Length Values**

## A.4.2 MediaLB 6-pin

Figure A-7 illustrates a typical MediaLB 6-pin interface between a Controller and a Device. In this example **MLBDP/N** and **MLBSP/N** are combined because they both behave similarly with respect to the clock, **MLBCP/N**.

The following parameters are used for the timing calculations for Figure A-7.

- $t_{cyc}$ - recovered clock period ($1/f_{mckr}$).
- $t_{ckQ}$ - recovered clock to Q output delay of the flip-flop.
- $t_{delrc}$ - Controller clock receiver delay.
- $t_{delrd}$ - Device clock receiver delay.
- $t_{deltd}$ - Device data/signal transmitter delay.
- $t_{propck}$ - on-board routing delay for the clock differential pair.
- $t_{propsd}$ - on-board routing delay for the data/signal differential pair.
- $t_{surc}$ - setup time for the Controller data/signal receiver.

The MediaLB Device utilizes a clock transmitted from the Controller (over **MLBCP/N**) to latch the data/signal onto **MLBSP/N** and **MLBDP/N**. The MediaLB Controller latches the data/signal from **MLBSP/N** and **MLBDP/N** utilizing the same clock recovered internally from *MediaLB Clock Out*. In order to meet the timing requirements the delay and setup times must add up to less than the recovered clock period.

$$t_{cyc} \geq t_{propck} + t_{delrd} + t_{ckQ} + t_{deltd} + t_{propsd} + t_{surc} - t_{delrc}$$

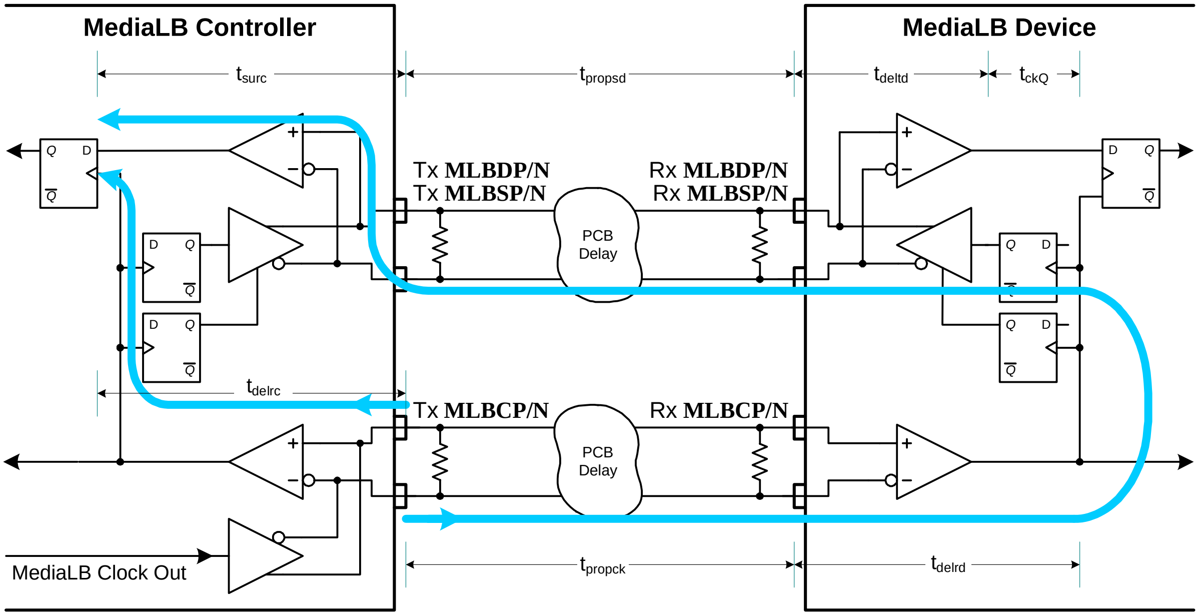


**Figure A-7: MediaLB 6-pin Gate-Level Signal Timing**

Assuming the on-board propagation delays are the same for clock and data/signal:

$$t_{prop} = t_{propck} = t_{propsd}$$

where $t_{prop}$ is the on-board trace delay (approximately 6.3 ps per mm of FR4).

The Device delay from receiving a clock change (low-to-high) on **MLBCP/N** to transmitting a valid data/signal on **MLBDP/N** and **MLBSP/N** is defined as $t_{delay}$.

$$t_{delay} = t_{delrd} + t_{ckQ} + t_{deltd}$$

The Controller setup from receiving valid data on **MLBDP/N** or **MLBSP/N** from a clock change (low-to-high) on the recovered clock is defined as $t_{su}$.

$$t_{su} = t_{surc} - t_{delrc}$$

Substituting into the first equation results in:

$$t_{cyc} \geq 2t_{prop} + t_{delay} + t_{su}$$

The maximum $t_{delay}$ is given by:

$$t_{delay(max)} = t_{cyc} - 2t_{prop} - t_{su(min)}$$

## A.5 MediaLB Board Layout

### A.5.1 MediaLB 3-pin

In order to support MediaLB 3-pin data rates, the following guidelines are recommended:

- Route all traces over a continuous plane.
- Match **MLBCLK**, **MLBSIG**, and **MLBDAT** trace lengths to minimize clock to data skew.
- Increase the spacing of **MLBCLK** trace to **MLBSIG** and **MLBDAT** traces.
- Use two 45 degree bends instead of a single 90 degree bend.
- Avoid routing other signal traces along the MediaLB bus for long parallel runs.

### A.5.2 MediaLB 6-pin

In order to support the high-speed MediaLB 6-pin bus, the following guidelines are recommended:

- Minimize trace lengths between nodes, thus reducing propagation time.
- Utilize a well controlled matched trace impedance for the balanced differential pairs.
- Maintain constant spacing between the traces of a differential pair.
- Route all traces over a continuous plane, preferably on the top or bottom layers with a continuous ground plane beneath.
- Match **MLBCP/N**, **MLBSP/N**, and **MLBDP/N** differential pair lengths to one another to minimize clock to data skew.
- Separate the **MLBSP/N** and **MLBDP/N** differential pairs from the **MLBCP/N** differential pair and avoid parallel runs over long distances.
- Use two 45 degree bends instead of a single 90 degree bend. Keep in mind that every corner may cause reflections.
- Minimize the use of vias.
- Eliminate test points and use component pads as access points.
- Use pass-throughs whenever possible, especially in multi-node high-speed systems.
- Minimize the length of stubs.
- Place termination resistors as close to the IC pins as possible.

A sample MediaLB 6-pin layout is given in Appendix B.2.2 on page 68.

# Appendix B: MediaLB Debug Headers

## B.1 MediaLB 3/5-pin Low-Speed Debug Header

A standard MediaLB 3/5-pin low-speed debug header is defined to allow various MediaLB analysis and debug tools to connect directly into the application. This header is used for monitoring MediaLB 3/5-pin interfaces, and should be placed as close as possible to the MediaLB interface to prevent unnecessary extension of MediaLB signal traces.

The pinout of the MediaLB Debug Header, as shown in Figure B-1, is compatible with *MediaLB Active Pods* for MediaLB 3/5-pin interfaces. The active pods are utilized by Version 2.0 and higher of the *MediaLB Analyzer* [11]. This header should be a 10-pin, 2 mm male connector, such as Molex 87332-1020 (shrouded) or Samtec TMM-105-06-T-D-SM (not shrouded).

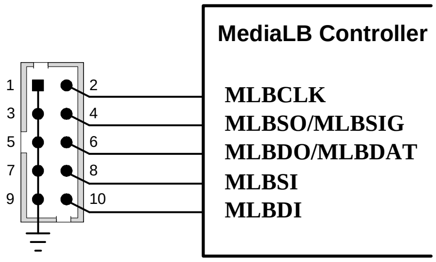


**Figure B-1: MediaLB 3/5-pin Low-Speed Debug Header**

## B.2 MediaLB 3/6-pin High-Speed Debug Header

A standard MediaLB 3/6-pin high-speed debug header is defined to allow MediaLB analysis and debug tools to connect directly into the application. This header is used for monitoring single-ended MediaLB 3-pin interfaces as well as differential MediaLB 6-pin interfaces. Careful consideration should be taken with respect to layout to ensure that the debug connector is placed as close to the MediaLB traces as possible, minimizing the length of any stubs.

The pinout of the MediaLB Debug Header, as shown in Figure B-2, is compatible with *MediaLB Active Pods* for MediaLB 3/6-pin interfaces. The active pods are utilized by Version 2.0 and higher of the *MediaLB Analyzer* [11]. It is essential that this 40-pin header supports high-speed data rates. An example header is Samtec's QSH-020-01-L-D-DP-A.

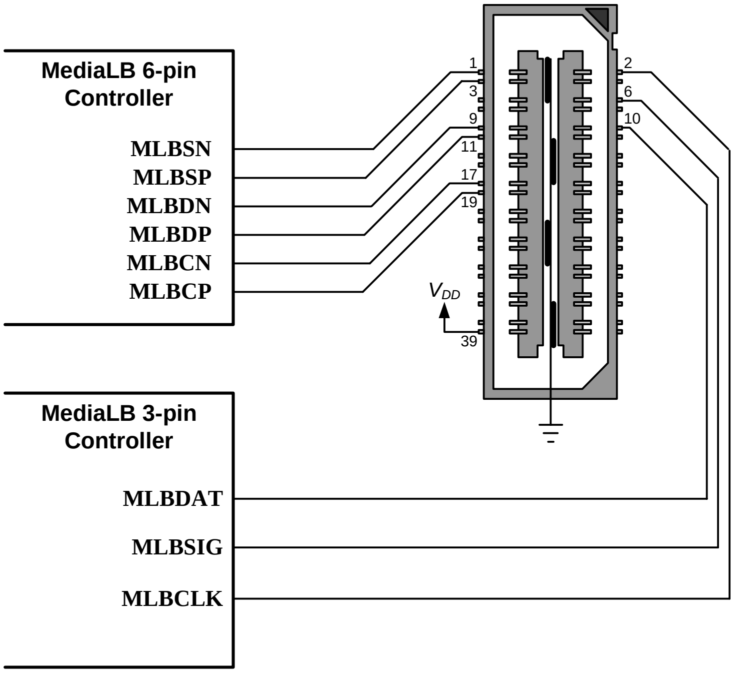


Figure B-2: MediaLB 3/6-pin High-Speed Debug Header

## B.2.1 Schematic Example

A sample MediaLB 6-pin debug header schematic is shown in Figure B-3. Series resistors R4 - R9 provide isolation for the debug connector. The value of these series resistors is dependent on the MediaLB speed and the line capacitance. In order to reduce the reflections occurring on the differential pairs, the debug header should be placed as close as possible to the MediaLB bus lines.

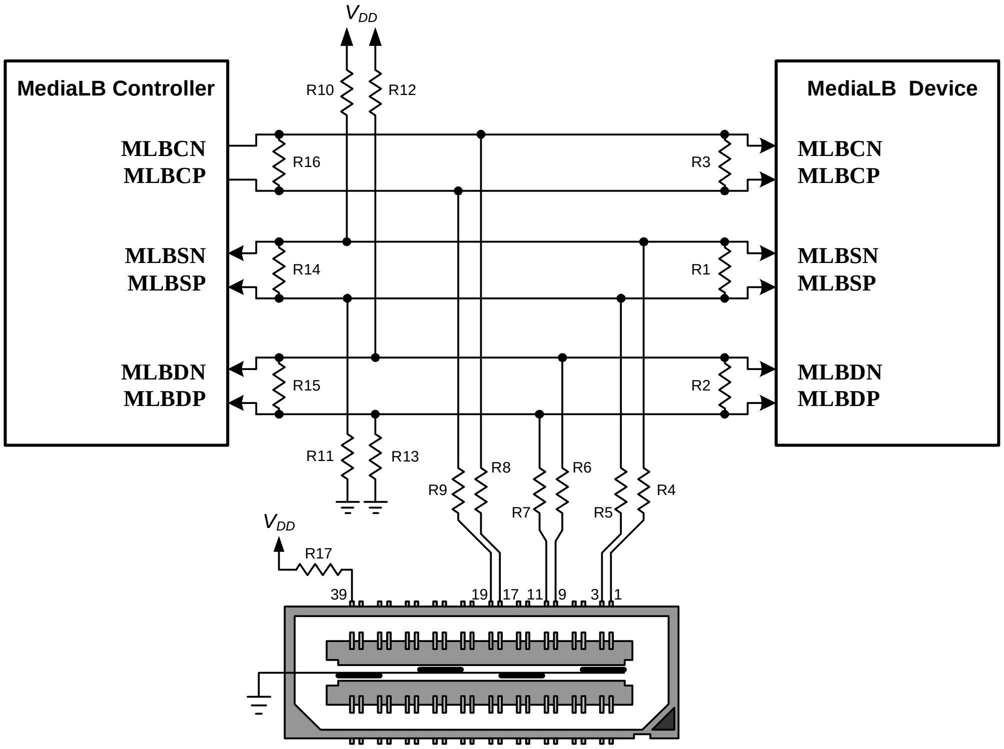


**Figure B-3: MediaLB 3/6-pin High-Speed Debug Header Schematic**

## B.2.2 Layout Example

A sample MediaLB 6-pin debug header layout is shown in Figure B-4. The part references designators are matched to the schematic shown in Figure B-3. The differential pair trace lengths are matched between *MLBS*, *MLBD*, and *MLBC*. In order to minimize stub lengths, the debug header is placed directly on top of the differential pair traces.

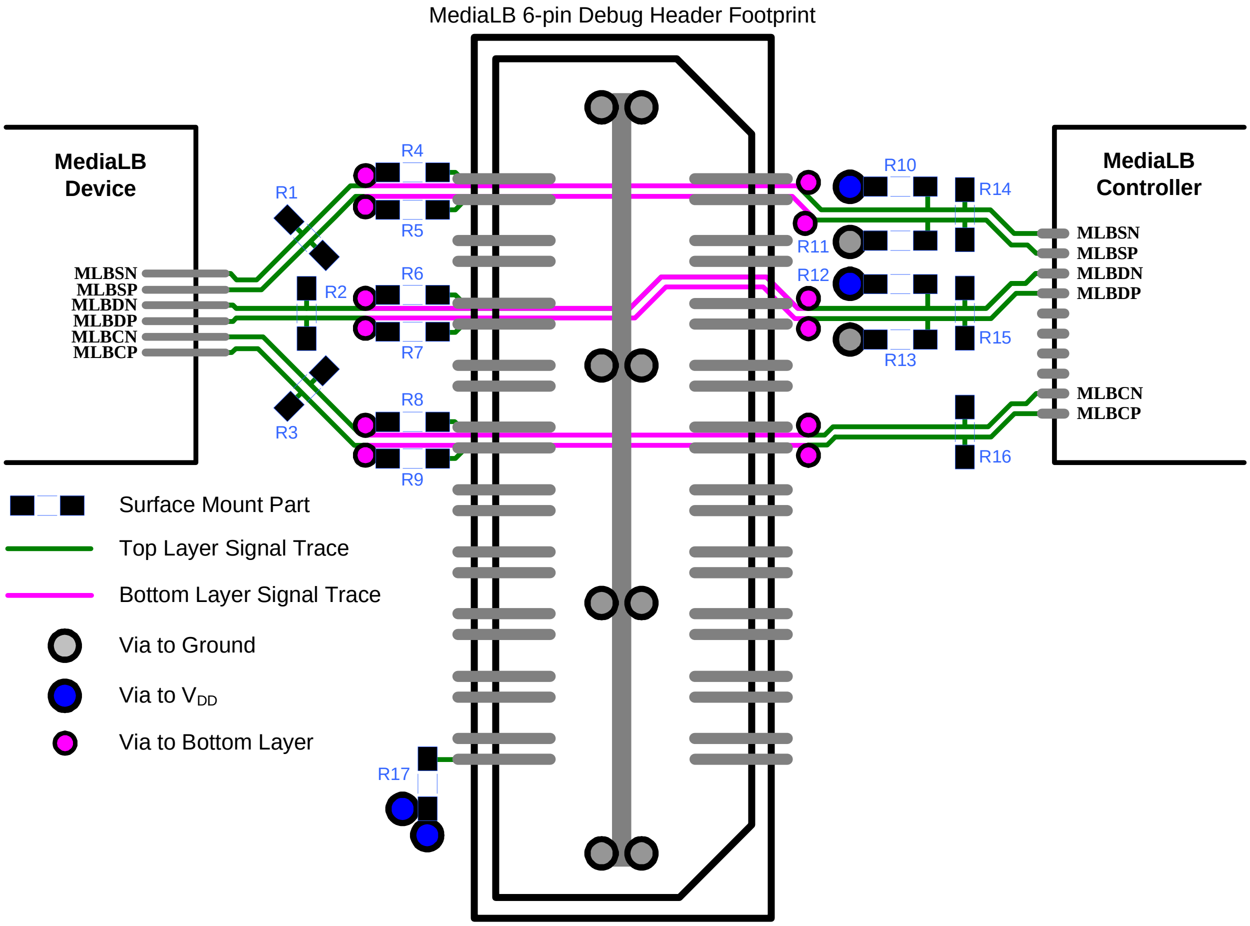


**Figure B-4: MediaLB 3/6-pin High-Speed Debug Header Layout**

# B.3 MediaLB 6-pin High-Speed Debug Header

In addition to the 40-pin MediaLB 3/6-pin high-speed debug header described in Appendix B.2 a 19-pin MediaLB 6-pin high-speed header is also defined. This 19-pin header can be used for all MediaLB 6-pin clock speeds, but is especially useful for the highest MediaLB 6-pin data-rates (6144×Fs and 8192×Fs). A sample schematic is shown in Figure B-5. Bus repeaters are used to spy the MediaLB 6-pin network traffic. This allows the 19-pin connector to be conveniently routed away from the MediaLB bus lines, see Figure B-6. An example of a high-speed LVDS bus repeater is Maxim's MAX9180EXT+. This debug method is compatible with Version 2.0 and higher of the *MediaLB Analyzer* [11]. A sample 19-pin MediaLB 6-pin high-speed debug header is Samtec's HDMI-19-02-1-SM or HDMI-19-02-S-SM.

## B.3.1 Schematic Example

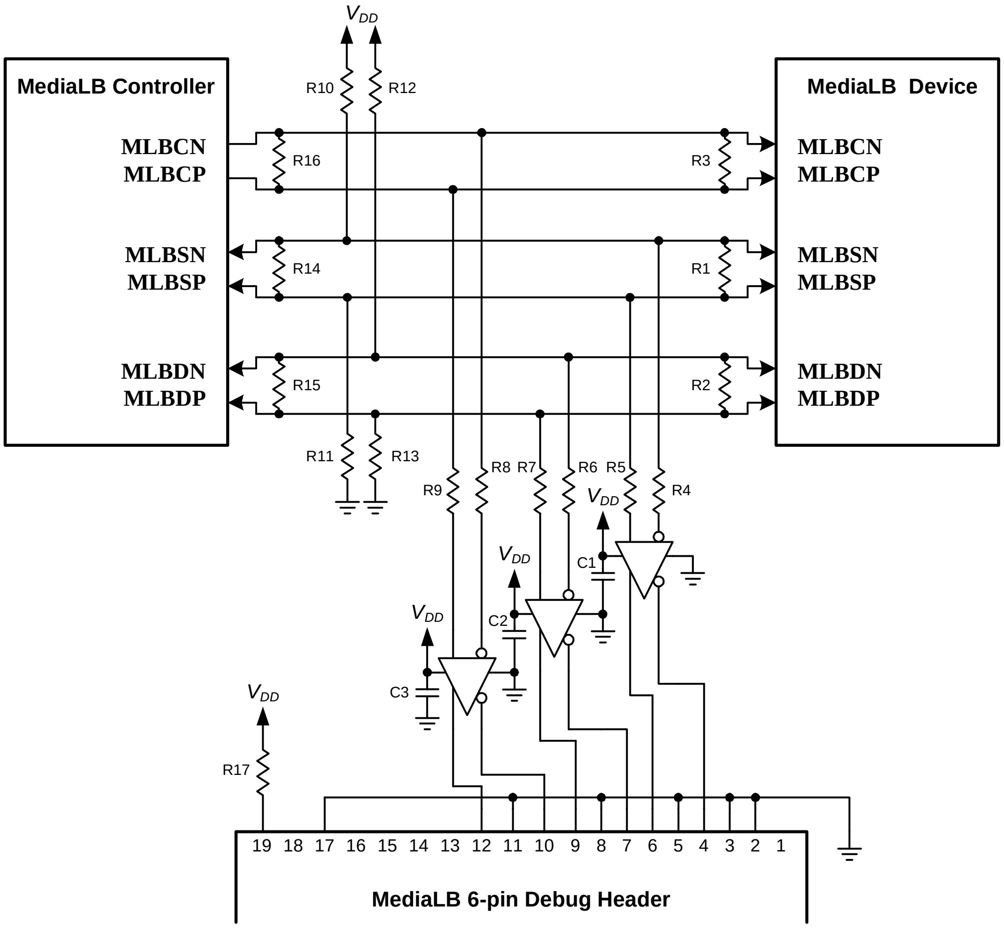


**Figure B-5: MediaLB 6-pin High-Speed Debug Header Schematic (4:1 Clock Speed)**

## B.3.2 Layout Example

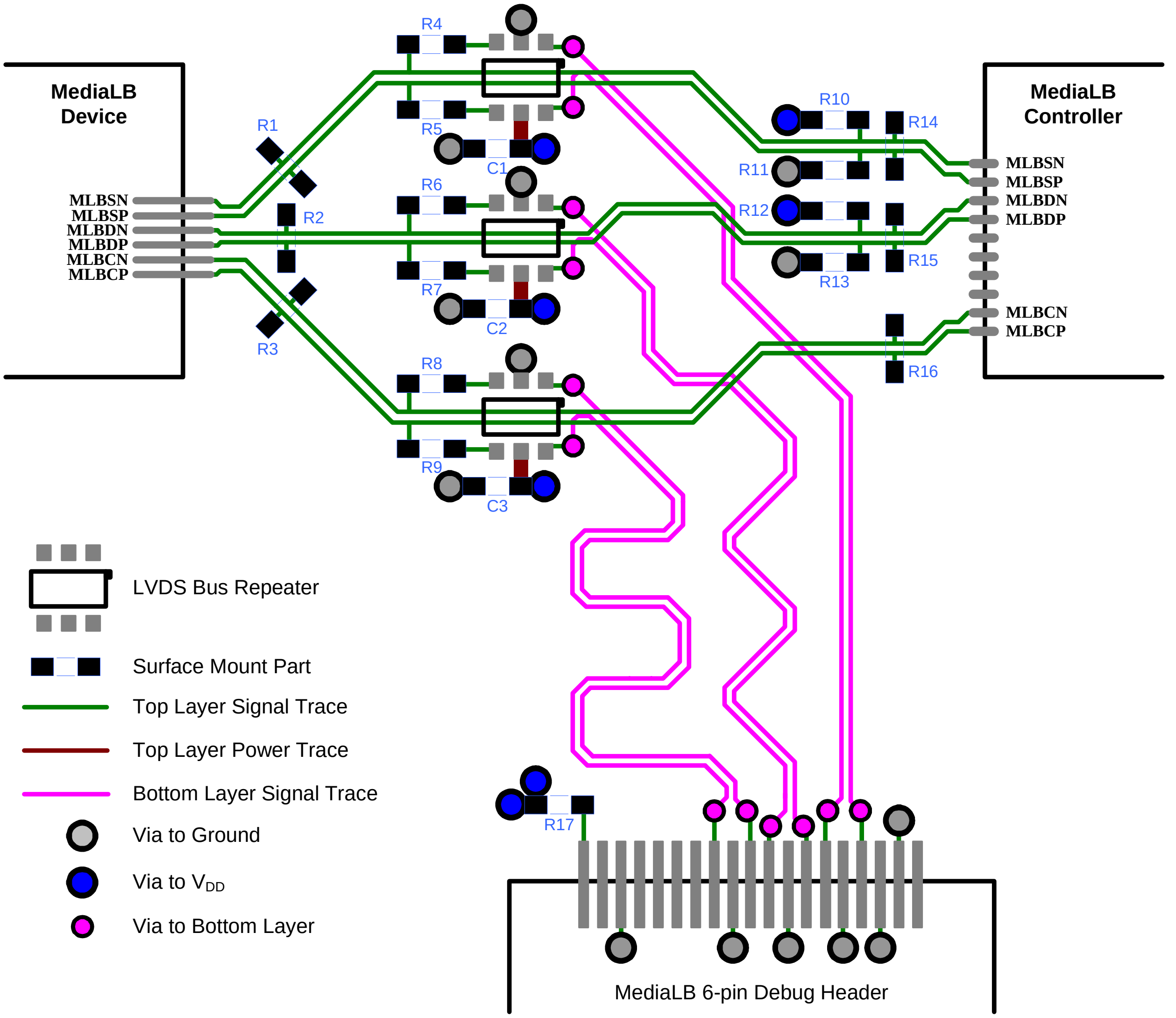


**Figure B-6: MediaLB 6-pin High-Speed Debug Header Layout (4:1 Clock Speed)**

# Appendix C: References

[1] OS81050 Data Sheet
SMSC.
Contact support-ais-de@smsc.com

[2] OS81060 Data Sheet
SMSC.
Contact support-ais-de@smsc.com

[3] OS81082 Data Sheet
SMSC.
Contact support-ais-de@smsc.com

[4] OS81092 Data Sheet
SMSC.
Contact support-ais-de@smsc.com

[5] OS81110 Data Sheet
SMSC.
Contact support-ais-de@smsc.com

[6] OS8105x/6x INIC API User's Manual
SMSC.
Contact support-ais-de@smsc.com

[7] OS81082 INIC API User's Manual
SMSC.
Contact support-ais-de@smsc.com

[8] OS81092 INIC API User's Manual
SMSC.
Contact support-ais-de@smsc.com

[9] OS81110 INIC API User's Manual
SMSC.
Contact support-ais-de@smsc.com

[10] MOST Specification
MOST Cooperation.
http://www.mostcooperation.com/publications/Specifications_Organizational_Procedures/index.html?dir=291

[11] MediaLB Analyzer Hardware Manual
SMSC. V2.0.X-2
Contact support-ais-de@smsc.com

# Appendix D: Revision History

| | Description of Changes |
|---|---|
| ***TB0400AN4V2:*** | |
| | 1. Corrected Figure 2-13 to big-endian from little-ended<br>2. Updated $f_{mck}$, $t_{mckl}$, and $t_{mckh}$ for MediaLB 3-pin and 5-pin (Sections 2.1.2.1 and 2.1.2.2, and 2.2.2.1)<br>3. Removed $t_{mpwv}$ for MediaLB 3-pin and 5-pin (Sections 2.1.2.1, 2.1.2.2, and 2.2.2.1); **MLBCLK** low and high times ($t_{mckl}$ and $t_{mckh}$) include the pulse width distortion<br>4. Updated $f_{mcke}$, $f_{mckr}$, $t_{delay}$, $t_{phz}$, $t_{plz}$, $t_{su}$, and $t_{hd}$ for MediaLB 6-pin in Section 2.3.3<br>5. Added $t_{prop}$ for MediaLB 6-pin in Section 2.3.3 and Figure 2-13<br>6. Updated Control Packet Tx Device Protocol flowchart (Figure 3-8)<br>7. Removed $t_{jitter}$ from Appendix A.4.2 since $t_{cyc}$ ($1/f_{mckr}$) includes jitter<br>8. Added $t_{delay}$ and $t_{prop}$ to Figure 2-2<br>9. Added recommended MediaLB 3-pin $t_{delay}$, $t_{prop}$, and trace length values in Table A-1<br>10. Removed $t_{mckc}$ for MediaLB 3-pin and 5-pin (Sections 2.1.2.1, 2.1.2.2, and 2.2.2.1) |
| ***TB0400AN4V1:*** | |
| | 1. Updated security marker from *Confidential* to *Normal*. |
| ***TB0400AN4V0:*** | |
| | 1. Various format and content changes have been made throughout the document to reflect the migration from OASIS SiliconSystems to SMSC<br>2. Incorporated information about MediaLB 6-pin physical interface<br>3. Updated MediaLB 3-pin connection diagram (Figure 2-1)<br>4. Updated MediaLB 3-pin electrical specifications (Sections 2.1.2.1 and 2.1.2.2)<br>5. Changed command names in Table 3-3 (*MOST_Lock*, *MOST_Unlock*, *MlbScan*, *MlbSubCmd*, and *MlbReset*)<br>6. Split the Compliance section from the Link Layer chapter (Chapter 4)<br>7. Combined and updated application information for MediaLB 3/6-pin (Appendix A)<br>8. Added MediaLB debug header sections (Appendices B.2 and B.3)<br>9. Updated reference list (Appendix C) |
| ***TB0400AN3V0:*** | |
| | 1. Incorporated changes outlined in *MediaLB Specification Update to Version 2.0* (TB0402AN2)<br>2. Updated MediaLB frame byte order (Figure 3-3 and Figure 4-3)<br>3. Updated Packet TX flowcharts (Figure 3-4, Figure 3-5, and Figure 3-6)<br>4. Clarification to *RxStatus* responses (Section 3.7)<br>5. Modifications to MediaLB example (Section 4.4)<br>6. Updated MediaLB electrical specifications (Section 2.1.2 and Appendix A)<br>7. Added Note 1 to MediaLB specification tables (Section 2.1.2 and Appendix A)<br>8. Added MediaLB Debug Header information (Appendix B) |
| ***TB0400AN2V0:*** | |
| | 1. Updated Physical Layer timing<br>2. Added 1024Fs Physical Layer timing<br>3. Added Isochronous Transport Method support<br>4. Reworded and reorganized document<br>5. Added *Applications Information for MediaLB 3-pin* chapter<br>6. Added *Compliance* section |
| ***Document 1V0-00:*** | |
| | 1. First released specification |

**Table D-1: Revision History**