# Joint Tactical Radio System Standard
# Modem Hardware Abstraction Layer
# Application Program Interface

**Version: 2.13.2**
**26 June 2013**

Statement A- Approved for public release; distribution is unlimited (17 July 2013).

# Revision History

| Version | Authorization | Description | Last Modified Date |
|---|---|---|---|
| 2.11.1 | | Preparation for public release<br>**ICWG Approved** | 02-May-2007 |
| 2.11.2<br><Draft> | | **Misc. Redlines**<br>**Change Proposal:**<br>Re-scoped MHALPhysicalDestination by<br>- Deprecating: MHAL::PF_MHALPacketConsumer::MHALPhysicalDestination<br>- Adding MHAL::MHALPhysicalDestination<br>- Changing addTxRoute Operation signature to utilize re-scoped MHALPhysicalDestination<br>- Updated *MhalDevice.idl* | 20-Jul-2009 |
| 2.12<br><Final Draft> | | **JTRS Community Review:**<br>- Added clarifying text to Section E.2.1, MHAL RF Chain Coordinator<br>- Misc. Redlines | 19-Aug-2009 |
| 2.12 | | **ICWG Approved** | 27-Aug-2009 |
| 2.13 | | Preparation for public release<br>**ICWG Approved** | 29-Jun-2010 |
| 2.13.1<br><Draft> | | Technical Correction:<br>- Corrected LD for TXBLOCKED command response<br>- Added missing response command for TXSTAT command request | 23-Jul-2010 |
| 2.13.1<br><Final Draft> | | No further changes. | 02-Feb-2011 |
| 2.13.1 | | No further changes.<br>**ICWG Approved** | 29-Mar-2011 |
| 2.13.2 | | Preparation for public release<br>**ICWG Approved** | 26-Jun-2013 |

# Table of Contents

**A. MHAL ..... 10**

**B. MHAL GPP API EXTENSION ..... 28**

**C. MHAL DSP API EXTENSION ..... 42**

**D. MHAL FPGA API EXTENSION ..... 52**

**E. MHAL RF CHAIN COORDINATOR API EXTENSION ..... 71**

# Table of Contents

**A. MHAL ..... 10**
**A.1 Introduction ..... 10**
A.1.1 Overview ..... 11
A.1.2 Service Layer Description ..... 12
A.1.3 Referenced Documents ..... 12
A.1.3.1 Government Documents ..... 12
A.1.3.2 Commercial Standards ..... 12
**A.2 Services ..... 13**
A.2.1 MHAL Communication Service ..... 13
A.2.1.1 Data Sink functions ..... 13
A.2.1.2 Data Source Functions ..... 14
A.2.1.3 MHAL Message ..... 14
A.2.2 MHAL Messaging Between Logical Destinations ..... 16
A.2.2.1 Push Only Communication Service ..... 16
A.2.2.2 MHAL Communications Flow Control ..... 16
**A.3 Service Primitives and Attributes ..... 17**
**A.4 Interface Definitions ..... 17**
**A.5 Data Types and Exceptions ..... 17**
A.5.1 MHAL Communication Routing Types ..... 17
A.5.1.1 MhalByte Type ..... 17
A.5.2 Common MHAL Message Construction Macros ..... 17
A.5.2.1 buildMhalMsg Macro ..... 17
A.5.2.2 Mhal_put8, Mhal_put16 & Mhal_put32 Macros ..... 18
A.5.2.3 Mhal_putMem Macro ..... 19
A.5.2.4 endBuildMhalMsg Macro ..... 20
A.5.3 Common MHAL Message Extraction Macros ..... 21
A.5.3.1 MhalParseStruct Structure ..... 21
A.5.3.2 accessMhalMsg Macro ..... 22
A.5.3.3 Mhal_get8, Mhal_get16 & Mhal_get32 Macros ..... 22
A.5.3.4 Mhal_Length Macro ..... 22
A.5.3.5 MhalMsgIndex Macro ..... 23
A.5.3.6 Mhal_EOM Macro ..... 23
A.5.3.7 MhalMsgField Macro ..... 23
A.5.3.8 endAccessMhalMsg Macro ..... 24
A.5.3.9 accessMhalField Macro ..... 24
A.5.3.10 endAccessMhalField Macro ..... 25
**Appendix A.A – Abbreviations and Acronyms ..... 26**
**Appendix A.B – Performance Specification ..... 27**

**B. MHAL GPP API EXTENSION ..... 28**
**B.1 Introduction ..... 28**
B.1.1 Overview ..... 28
B.1.2 Service Layer Description ..... 29

B.1.2.1 MHAL Port Connections ..... 29
B.1.2.2 Modes of Service ..... 29
B.1.2.3 Service States ..... 29
B.1.2.4 MHAL State Diagram ..... 29
**B.2 Services ..... 31**
B.2.1 Provide Services ..... 31
B.2.2 Use Services ..... 32
B.2.3 Interface Modules ..... 33
B.2.3.1 MHAL ..... 33
**B.3 Service Primitives and Attributes ..... 34**
B.3.1 MHAL::MHALPacketConsumer ..... 34
B.3.1.1 *pushPacket* Operation ..... 34
B.3.2 MHAL::PF_MHALPacketConsumer ..... 35
B.3.2.1 *addTxRoute* Operation ..... 35
B.3.3 MHAL::WF_MHALPacketConsumer ..... 36
B.3.3.1 getRxRoutes Operation ..... 36
**B.4 IDL ..... 37**
B.4.1 MhalDevice IDL ..... 37
**B.5 Data Types and Exceptions ..... 39**
B.5.1 MHAL Types ..... 40
B.5.1.1 MHAL::MHALPhysicalDestination Type ..... 40
B.5.2 MHAL::PF_MHALPacketConsumer Types ..... 40
B.5.2.1 MHAL::PF_MHALPacketConsumer::MHALPhysicalDestination Type <Deprecated> ..... 40
B.5.3 MHAL::WF_MHALPacketConsumer Types ..... 40
B.5.3.1 MHAL::WF_MHALPacketConsumer::RxRouteSequence Type ..... 40
**Appendix B.A – Abbreviations and Acronyms ..... 41**
**Appendix B.B – Performance Specification ..... 41**

**C. MHAL DSP API EXTENSION ..... 42**
**C.1 Introduction ..... 42**
C.1.1 Overview ..... 42
**C.2 Services ..... 43**
C.2.1 MHAL Communication Routing Module ..... 43
C.2.2 MHAL Message In-Use Module (*optional*) ..... 43
**C.3 Service Primitives and Attributes ..... 44**
C.3.1 MHAL Communication Routing ..... 44
C.3.1.1 *Mhal_Comm* Function ..... 44
C.3.1.2 reroute_LD_sink Function ..... 45
C.3.1.3 *LD_of* Function ..... 46
C.3.2 MHAL Message In-Use (*optional*) ..... 47
C.3.2.1 setMsgInUse Function ..... 47
C.3.2.2 clrMsgInUse Function ..... 48
C.3.2.3 isMsgInUse Function ..... 49
**C.4 Interface Definitions ..... 50**

**C.5 Data Types and Exceptions** ..... **50**
C.5.1 MHAL Communication Routing Types ..... 50
C.5.1.1 funPtr Type ..... 50
**Appendix C.A – Abbreviations and Acronyms** ..... **51**
**Appendix C.B – Performance Specification** ..... **51**

**D. MHAL FPGA API EXTENSION** ..... **52**
**D.1 Introduction** ..... **52**
D.1.1 Overview ..... 52
D.1.2 Service Layer Description ..... 52
D.1.2.1 MHAL FPGA Signals ..... 52
D.1.2.2 MHAL FPGA Timing ..... 57
**D.2 Services** ..... **64**
**D.3 Service Primitives and Attributes** ..... **64**
**D.4 Entity Definitions** ..... **64**
D.4.1 MHAL FPGA Transmit Node – Multi-Depth FIFO Entity Description ..... 64
D.4.2 MHAL FPGA Receive Node – Multi-Depth FIFO Entity Description ..... 65
D.4.3 MHAL FPGA Receive Node – Single-Depth FIFO Entity Description ..... 66
D.4.4 MHAL FPGA Receive Node – RAM Entity Description ..... 67
D.4.5 MHAL FPGA Receive Node – N-Word Register Entity Description ..... 68
D.4.6 MHAL FPGA Receive Node – Strobe Entity Description ..... 69
**D.5 Data Types and Exceptions** ..... **69**
**Appendix D.A – Abbreviations and Acronyms** ..... **70**
**Appendix D.B – Performance Specification** ..... **70**

**E. MHAL RF CHAIN COORDINATOR API EXTENSION** ..... **71**
**E.1 Introduction** ..... **71**
E.1.1 Overview ..... 71
**E.2 Services** ..... **72**
E.2.1 MHAL RF Chain Coordinator ..... 72
E.2.2 Sink Functions ..... 73
E.2.3 Source Functions ..... 74
**E.3 Service Primitives and Attributes** ..... **75**
E.3.1 *General* Sink Functions ..... 75
E.3.1.1 RFC_DefModulationMode Command ..... 75
E.3.1.2 RFC_ModulationMode Command ..... 77
E.3.1.3 RFC_RxAGCAttackTime Command *(optional)* ..... 78
E.3.1.4 RFC_RxAGCDecayTime Command *(optional)* ..... 79
E.3.1.5 RFC_TxALCAttackTime Command ..... 80
E.3.1.6 RFC_TxALCDecayTime Command ..... 81
E.3.1.7 RFC_TxEnvDecayTime Command ..... 82
E.3.1.8 RFC_TxEnvRiseTime Command ..... 83
E.3.1.9 RFC_ChannelFrequency Command ..... 84
E.3.1.10 RFC_ChannelRxModeSet Command ..... 85
E.3.1.11 RFC_ChannelTxModeSet Command ..... 86

E.3.1.12 RFC_TxPower Command ..... 87
E.3.1.13 RFC_ChannelTxKeyDisable Command ..... 88
E.3.1.14 RFC_ChannelTxKeyEnable Command ..... 89
E.3.1.15 RFC_ConnectTxBlock Command ..... 90
E.3.1.16 RFC_MasterExecuteNow Command ..... 91
E.3.1.17 RFC_ReceiverGainControl Command ..... 92
E.3.2 Supervisory Sink Functions ..... 93
E.3.2.1 RFC_ChannelStandbyModeSet Command ..... 93
E.3.2.2 RFC_TxBusyStatusRequest Command ..... 94
E.3.3 General Source Functions ..... 95
E.3.3.1 RFC_TxBlocked Command ..... 95
E.3.3.2 RFC_TxBusyStatusResponse Command ..... 96
**E.4 Interface Definitions ..... 97**
**E.5 Data Types and Exceptions ..... 97**
**Appendix E.A – Abbreviations and Acronyms ..... 98**
**Appendix E.B – Performance Specification ..... 98**
**Appendix E.C – Multi-Command Message Example ..... 98**

# Lists of Figures

**FIGURE 1 – MHAL API REFERENCE DEPLOYMENT DIAGRAM** ..... 11
**FIGURE 2 – STANDARD MESSAGE STRUCTURE FOR MHAL COMMUNICATION** ..... 14
**FIGURE 3 – MHAL PORT DIAGRAM** ..... 29
**FIGURE 4 – MHAL STATE DIAGRAM** ..... 30
**FIGURE 5 – MHAL INTERFACE CLASS DIAGRAM** ..... 33
**FIGURE 6 – MHAL COMPONENT DIAGRAM** ..... 39
**FIGURE 7 – MULTI-DEPTH FIFO TRANSMIT COMMUNICATION DEVICE** ..... 53
**FIGURE 8 – MULTI-DEPTH FIFO RECEIVE COMMUNICATION DEVICE** ..... 54
**FIGURE 9 – SINGLE-DEPTH FIFO RECEIVE COMMUNICATION DEVICE** ..... 55
**FIGURE 10 – RAM RECEIVE COMMUNICATION DEVICE** ..... 55
**FIGURE 11 – N-WORD REGISTER RECEIVE COMMUNICATION DEVICE** ..... 56
**FIGURE 12 – STROBE RECEIVE COMMUNICATION DEVICE** ..... 57
**FIGURE 13 – REFERENCE TIMING DIAGRAM FOR MULTI-DEPTH FIFO TRANSMIT NODE** ..... 58
**FIGURE 14 – REFERENCE TIMING DIAGRAM FOR MULTI-DEPTH FIFO RECEIVE NODE** ..... 59
**FIGURE 15 – REFERENCE TIMING DIAGRAM FOR SINGLE-DEPTH FIFO RECEIVE NODE** ..... 60
**FIGURE 16 – REFERENCE TIMING DIAGRAM FOR RAM RECEIVE NODE** ..... 61
**FIGURE 17 – REFERENCE TIMING DIAGRAM FOR N-WORD RECEIVE REGISTER** ..... 62
**FIGURE 18 – REFERENCE TIMING DIAGRAM FOR A STROBE** ..... 63

# List of Tables

**TABLE 1 – MHAL MESSAGE FIELD SUMMARY** ..... 14
**TABLE 2 – MHAL DEVICE PROVIDE SERVICE INTERFACE** ..... 31
**TABLE 3 – MHAL USE SERVICE INTERFACE** ..... 32
**TABLE 4 – MHAL PERFORMANCE SPECIFICATION** ..... 41
**TABLE 5 – MHAL RF CHAIN COORDINATOR GENERAL SINK FUNCTION INTERFACE** ..... 73
**TABLE 6 – MHAL RF CHAIN COORDINATOR SUPERVISORY SINK FUNCTION INTERFACE** ..... 73
**TABLE 7 – MHAL RF CHAIN COORDINATOR GENERAL SOURCE FUNCTION INTERFACE** ..... 74
**TABLE 8 – EXAMPLE MULTI-MESSAGE COMMAND STRUCTURE** ..... 98

# A. MHAL

## A.1 INTRODUCTION

This API provides information to the software developer to utilize the *Modem Hardware Abstraction Layer* (*MHAL*) interfaces in the waveform target configurations.

The MHAL API abstracts the JTR channel modem interfaces from the application software. The MHAL API supports communications between application components hosted on General Purpose Processors (GPPs), Modem Digital Signal Processors (DSPs) and/or Modem Field Programmable Gate Arrays (FPGAs).

For the purposes of this API the following applies to processor naming conventions:

- A GPP represents a CORBA capable processor (This could be a DSP that supports CORBA.)
- A DSP represents a C capable processor, but does not provide CORBA capability.
- An FPGA represents a HDL capable processor, again without CORBA capability.

The concept of the MHAL API is to provide a consistent host environment for waveform applications and waveforms across all JTRS platforms. Because the waveform-side interfaces are presented by the MHAL API, abstraction elements remains the same from platform to platform, thus waveform and application components can be ported readily between JTR sets. The platform-side interfaces of the MHAL are defined by the JTR set for its particular architecture and mission. This document defines a common set of MHAL services and interfaces required by most JTR sets.

From one MHAL Computational Element (CE) (i.e. GPP, FPGA, or DSP), it is possible to access any of the other CEs (i.e. FPGA, DSP, and RF units) using the routing service defined in A.2.1 MHAL Communication Service. The MHAL message format is the same for all MHAL Computational Elements (CE) and is defined in A.2.1.3.1 MHAL Message Structure. The MHAL GPP is the Software Communications Architecture (SCA) *CF::Device* interface (see reference [3]) and is defined in section B MHAL GPP API Extension. Because the DSP environment does not readily support dynamic linkable objects, the *MHAL DSP* is a library of standardized components that are linked into the waveform code at build time. The external interfaces and transport are JTR set defined, but the exposed interfaces to DSP waveform components are consistent across all JTRS products and are defined in section C MHAL DSP API Extension. Likewise the MHAL FPGA consists of an FPGA entity library that is linked into a waveform build. The JTR set interfaces are unique, but the interfaces exposed to waveform components are consistent across the JTR products and are defined in section D MHAL FPGA API Extension.

**Figure 1** shows a reference deployment of the MHAL API. The MHAL API does not specify the number of Computational Elements a JTR platform shall provide. The MHAL API does not specify the platform specific transport, implementation or hardware architecture.

The MHAL API does specify the MHAL protocol interfaces (i.e. Data Sink and Data Source functions highlighted in **Figure 1**) of different Computational Elements for communication between the waveform and hardware.

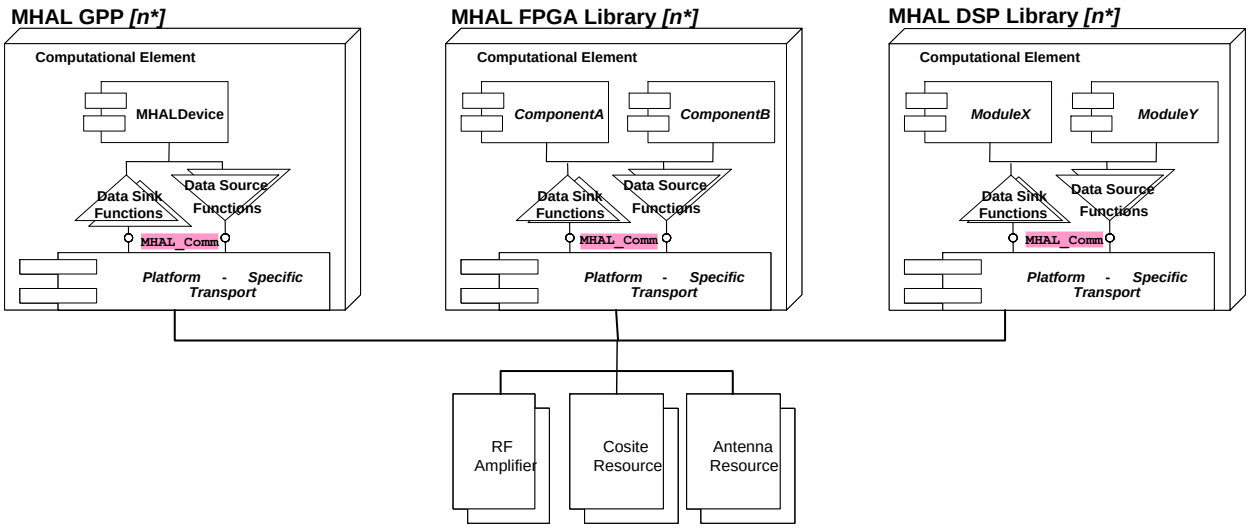


**Figure 1 – MHAL API Reference Deployment Diagram**

# A.1.1 Overview

This document contains as follows:

a. Section A.1, *Introduction*, of this document contains the introductory material regarding the Overview, Service Layer description, and Referenced Documents of this document.

b. Section A.2, *Services*, provides summary of service uses.

c. Section A.3, *Service Primitives and Attributes*

d. Section A.4, *Interface Definitions*

e. Section A.5, *Data Types and Exceptions*

f. Appendix A.A – *Abbreviations and Acronyms*

g. Appendix A.B – *Performance Specification*

## A.1.2 Service Layer Description

Not applicable

## A.1.3 Referenced Documents

The following documents of the exact issue shown form a part of this specification to the extent specified herein.

### A.1.3.1 Government Documents

The following documents are part of this specification as specified herein.

#### A.1.3.1.1 Specifications

##### A.1.3.1.1.1 Federal Specifications

None

##### A.1.3.1.1.2 Military Specifications

None

#### A.1.3.1.2 Other Government Agency Documents

[1] JTRS Standard, “JTRS Standard CORBA Types,” JPEO, Version 1.0.2
[2] JTRS Standard, “Packet API,” JPEO, Version 2.0.2
[3] JTRS Standard, “Software Communications Architecture (SCA),” JPEO, Version 2.2.2

### A.1.3.2 Commercial Standards

None

# A.2 SERVICES

## A.2.1 MHAL Communication Service

MHAL Communication Service is provided by a MHAL Communications Function working in conjunction with the MHAL Interface Components. MHAL Interface Components provide the message transport and an MHAL Communications Function provides an abstracted message routing function. Waveforms shall use the MHAL Communications Service for all data and control flowing between software components residing in different CEs where at least one CE does not support CORBA.

MHAL Interface Components may consist of Software (SW) Drivers and FPGA Interfaces. Both provide the same system function. An FPGA Interface includes a mechanization of the desired physical interface whereas a SW Driver manipulates interface HW provided by the SW's host processor (i.e. GPP or DSP).

A Data Source launches messages by calling an MHAL Communications Function and a Data Sink receives messages by being called by an MHAL Communications Function. The MHAL Communication Service provides an asynchronous variable length messaging service between Data Sources and Data Sinks. Communicating Data Sources and Sinks may be located in separate CEs or within the same CE.

If communicating within the same CE, it is not required to use the MHAL Communications Service.

Data Sink & Source Functions for the RF Chain Coordinator are defined in section E MHAL RF Chain Coordinator API Extension.

### A.2.1.1 Data Sink functions

A "sink function" is defined as the translation of the MHAL message into the expected behavior associated with message parameters. An MHAL supplied sink function can process the message or route the message to another CE. A Waveform supplied sink function is expected to process the message as soon as possible. Sink functions are always associated with a logical destination with the association being dependent on the CE where the message is sourced.

As each CE has a different approach to causing the sink process to be executed, it is necessary to consider the call in a generic form. All callable Data Sink Functions have the following generic form (in C) where developers replace *DataSinkFunctionX* with a component specific function:

```
void DataSinkFunctionX (byte* MessagePointer);
```

Note: `MessagePointer` points to the least significant byte of the LD element of the message to be processed.

The SCA-compliant CORBA MHAL API for GPP uses a pushPacket(…) call whereas function calls on the DSP are serviced through function Mhal_Comm(…). In the FPGA interfaces, a receive node contains a signal interface that is asserted when a complete message is received. Callable Data Sinks Functions include:

- GPP to GPP; DSP; FPGA; external RF Data Sink Functions via a pushPacket() call
- DSP to GPP, DSP, FPGA, external RF Data Sink Functions via an Mhal_Comm() call
- Waveform Application software Data Sink Functions using the same interface as Data Sink functions

- In general; any Data Sink function callable by or through the MHAL

### A.2.1.2 Data Source Functions

A Data Source Function is defined as the execution thread that issues an MHAL message. The MHAL message may be the result of performing behavior associated with waveform's API or it may be the direct result of processing a received MHAL message. In this latter case, the Data Source Function is also a sink function. The distinction of a sink function versus a source function is important on how the MHAL message is processed by the communication service. MHAL source functions can exist on any CE.

### A.2.1.3 MHAL Message

#### A.2.1.3.1 MHAL Message Structure

All messages transported by the MHAL have the structure depicted in Figure 2.

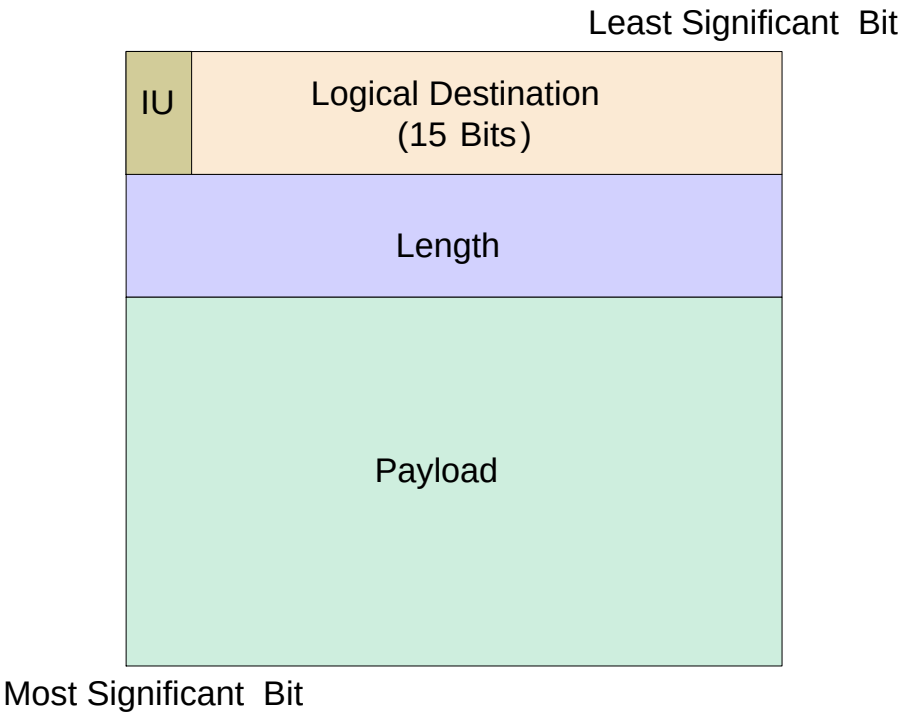


**Figure 2 – Standard Message Structure for MHAL Communication**

MHAL message structure includes the information required to maintain an orderly processing of message buffers. The value for In-Use (IU) bit is used by the MHAL for internal message flow control. The resulting Logical Destination consists of 15 bits. The message payload may be `zero bytes` to `MaxMessageLength - 4 bytes` in length. An MHAL message field summary is detailed in **Table 1**. All data is least significant bit first and byte first.

**Table 1 – MHAL Message Field Summary**

| Message Field Name | Bit Location | Length | Valid Range |
|---|---|---|---|
| Logical Destination (LD) | `0 to 14` | `15 bits` | `0 to 2^15-1` |

| Message Field Name | Bit Location | Length | Valid Range |
|---|---|---|---|
| In-Use (IU) | `15` | `1 bit` | `0 or 1` |
| Length | `16 to 31` | `16 bits` | `0 to 2^16-5` |
| Payload | `32 to (2^16-31)` | `2^16-8 bytes` | `N/A` |

Note: Waveforms must consider the latency impact on the design when large messages are used.

#### A.2.1.3.1.1 MHAL Message IU Bit

The MHAL Message IU Bit is for Data Sinks and Data Sources that share memory and is not for use across CEs that do not. The IU bit is used as an In-Use bit and may be ignored by the user. The IU bit should be utilized when the message buffer memory is shared between a Data Source and targeted Data Sink. The internal transport layer uses the IU bit to signal when the Data Sink has completed processing a MHAL message.

Data Sinks must be required to inform MHAL that the message pointer is In-Use before returning from the function.

For the *MHAL DSP* access functions are provided to allow waveforms to employ the IU bit (see C.2.2 MHAL Message In-Use Module (*optional*)).

#### A.2.1.3.1.2 MHAL Message Logical Destination

The Logical Destination (LD) is used to route messages to destination Data Sink functions. Every target Data Sink function is associated with a LD. Each LD represents an interface and does not necessarily define a one-to-one mapping to a function. A LD can represent a single DSP function or FPGA node, or it can represent a collection of functions or nodes that can be signaled concurrently.

Each LD is an integer constant whose value may change from build to build but whose symbolic reference is fixed. LDs are globally defined across all CE domains.

The LD shall be referenced symbolically in user source code (C or VHDL) so that its value may change without impacting user source code.

#### A.2.1.3.1.3 MHAL Message MaxMessageLength

The Maximum Message Length is a modifiable variable with an upper range of `2^16 bytes-4`. The minimum payload length is zero bytes. Often this 0 byte payload is referred to as a soft event.

The message length limitation is a suggested convention that should be followed to assure adequate system communications agility. Since several source-sink pairs may be configured to communicate over the same physical interface, a relatively modest maximum message length assures that each source gets a chance to deliver its message in a timely fashion.

### A.2.1.3.2 MHAL Message Byte Orientation

MHAL Messages are encoded using Little-Endian byte orientation. A multiple byte data element is ordered least significant byte (LSB) to most significant byte (MSB) in order of increasing address. A

pointer referencing this data element points to the least significant byte or lowest address occupied by the data element.

This document recommends that the host FPGA and DSP are configured to be Little-Endian so that data marshalling will not be required in these CEs.

If it is not possible to constrain the CE Endian orientation the SW written for the CE must build MHAL Messages in an Endian-Independent manner. In order for GPP SW to be portable, it must always marshal the data in a way that does not assume a particular byte orientation. It must build the MHAL message element by element with each byte in the proper order.

#### A.2.1.3.3 MHAL Message Processing

Data Source Functions provide all message pre-processing. Data Sink Functions access the data in these buffers by using the `MessagePointer` parameter passed to them.

#### A.2.1.3.4 MHAL Message Persistence

Messages held by a Data Source have finite guaranteed persistence.

## A.2.2 MHAL Messaging Between Logical Destinations

### A.2.2.1 Push Only Communication Service

The MHAL Communication Service shall provide a Push-Only communications service. Data can be "written" to a destination but cannot be directly "read" in one operation.

### A.2.2.2 MHAL Communications Flow Control

MHAL flow control is accomplished in the Data Sink Functions.

# A.3 SERVICE PRIMITIVES AND ATTRIBUTES

None

# A.4 INTERFACE DEFINITIONS

None

# A.5 DATA TYPES AND EXCEPTIONS

## A.5.1 MHAL Communication Routing Types

### A.5.1.1 MhalByte Type

Defines a type used to declare a byte of data in an MHAL message

#### A.5.1.1.1 Synopsis

#define MhalByte unsigned char

## A.5.2 Common MHAL Message Construction Macros

### A.5.2.1 buildMhalMsg Macro

This macro initializes the header of an MHAL message. This macro can set up a processing block within the function using it. It must be terminated with the `endBuildMhalMsg()` macro. This is a non-reentrant macro.

#### A.5.2.1.1 Synopsis

*#define buildMhalMsg (messagePointer, LogicalDestination)*

#### A.5.2.1.2 Arguments

| Parameter Name | Description |
|---|---|
| *messagePointer* | Address to a region of memory where the MHAL message is to reside |
| *LogicalDestination* | Logical destination identifier for the targeted Data Sink Function. |

#### A.5.2.1.3 Return Value

Pointer to first byte in payload to be written.

### A.5.2.2 Mhal_put8, Mhal_put16 & Mhal_put32 Macros

This macro stores the LSB n-bits of value into the next available byte(s) available in the message declared with the `buildMhalMsg` macro. After any `Mhal_put`*n* macro call the next available byte position in the message is set to the first byte after the data just written.

Mhal_put8 - least significant 8 bits moved into 1 byte
Mhal_put16 - least significant 16 bits moved into 2 bytes
Mhal_put32 - 32 bits moved into 4 bytes

#### A.5.2.2.1 Synopsis

*#define Mhal_put8 (value)*

*#define Mhal_put16 (value)*

*#define Mhal_put32 (value)*

#### A.5.2.2.2 Arguments

| Parameter Name | Description |
|---|---|
| value | Data to be passed in the message. |

#### A.5.2.2.3 Return Value

None

### A.5.2.3 Mhal_putMem Macro

This macro can be used to copy a block of memory to the MHAL message currently being constructed. Similar to the standard memcpy function, this function will copy a block of data from `sourceData` of length `size` to the MHAL message. This function is useful for text strings. If the processor is Little Endian format this macro can be used to copy a structure.

#### A.5.2.3.1 Synopsis

*#define Mhal_putMem(sourceData, size)*

#### A.5.2.3.2 Arguments

| Parameter Name | Description |
|---|---|
| sourceData | Address to a region of memory that contains the data being passed in the message |
| size | Size in bytes of the data block being copied |

#### A.5.2.3.3 Return Value

None

### A.5.2.4 endBuildMhalMsg Macro

This macro calculates the length of the MHAL message and stores it in the MHAL header. This ends the construction of a particular MHAL message.

#### A.5.2.4.1 Synopsis

*#define endBuildMhalMsg()*

#### A.5.2.4.2 Arguments

None

#### A.5.2.4.3 Return Value

None

## A.5.3 Common MHAL Message Extraction Macros

All MHAL extraction macros except `accessMhalMsg` and `endAccessMhalMsg` macro can be used within the block structure of `accessMhalField – endAccessMhalField`.

### A.5.3.1 MhalParseStruct Structure

This structure defines a structure for extracting data from an MHAL message. This structure is used to manage the extraction of data from an MHAL message. It is created within the call to the `accessMhalMsg()` operation. The names of the fields are such that a developer using standard coding practices would not select the name, thus avoiding name conflicts with developer code.

#### A.5.3.1.1 Synopsis

*typedef struct {*

*MhalByte* MsgptR;*

*MhalByte* MsgnbP;*

*} MhalParseStruct;*

#### A.5.3.1.2 Attributes

| Struct | Attributes | Type | Valid Range | Description |
|---|---|---|---|---|
| MhalParseStruct | MsgptR | MhalByte* | N/A | Contains a pointer to the first byte of the MHAL message. This is the low byte of the LD in the header. I.e. `Mhal_Length` macro and `MhalMsgIndex` macro would use this field to extract the message length or determine the byte index to be parsed |
| | MsgnbP | MhalByte* | N/A | The message next byte pointer of data in the payload to extract. This field is advanced whenever the `Mhal_getn` macros are invoked. This is used is testing for message completion. |

#### A.5.3.1.3 Return Value

Not applicable

### A.5.3.2 accessMhalMsg Macro

This macro reads the header of an MHAL and initializes the extraction process for an MHAL message. This macro can set up a processing block within the function using it. It must be terminated with the `endAccessMhalMsg` macro. The `messagePointer` argument will point to the MHAL message from which the user wishes to extract the data

#### A.5.3.2.1 Synopsis

*#define accessMhalMsg(messagePointer)*

#### A.5.3.2.2 Arguments

| Parameter Name | Description |
|---|---|
| messagePointer | Address to a region of memory where the MHAL message resides |

#### A.5.3.2.3 Return Value

None

### A.5.3.3 Mhal_get8, Mhal_get16 & Mhal_get32 Macros

This macro reads the next n-bytes from the message and stores the value into the LSB n-bits defined by the destination. After any `Mhal_get`*n* macro call the next available byte position in the message is set to the first byte after the data just read.

Mhal_get8 - 1 byte read, least significant 8 bits written
Mhal_get16 - 2 bytes read, least significant 16 bits written
Mhal_get32 - 4 bytes read, 32 bits written

#### A.5.3.3.1 Synopsis

*#define Mhal_get8(destination)*

*#define Mhal_get16(destination)*

*#define Mhal_get32(destination)*

#### A.5.3.3.2 Arguments

| Parameter Name | Description |
|---|---|
| Destination | Name of a location in memory where the parsed data is to be stored. Destination is only referenced once so that auto-increment of a pointer reference can be used. |

#### A.5.3.3.3 Return Value

None

### A.5.3.4 Mhal_Length Macro

This macro provides the length in bytes of the current MHAL message being processed.

#### A.5.3.4.1 Synopsis

*#define Mhal_Length*

#### A.5.3.4.2 Arguments

None

#### A.5.3.4.3 Return Value

Length in bytes of the message. This includes the MHAL header size.

### A.5.3.5 MhalMsgIndex Macro

This macro provides the byte index of the next byte to be read in the current MHAL message being processed.

#### A.5.3.5.1 Synopsis

*#define MhalMsgIndex*

#### A.5.3.5.2 Arguments

None

#### A.5.3.5.3 Return Value

Byte index within the message.

### A.5.3.6 Mhal_EOM Macro

This macro checks to determine if the next read position is at or past the end of the MHAL message being processed.

#### A.5.3.6.1 Synopsis

*#define Mhal_EOM*

#### A.5.3.6.2 Arguments

None

#### A.5.3.6.3 Return Value

Boolean TRUE if at the "End Of Message".

### A.5.3.7 MhalMsgField Macro

This macro provides access the structure representing the MHAL message being processed when the `accessMhalMsg()` macro was encountered. This may be used to acquire the input argument to pass the MHAL message to a field parsing function inline or to another function within the source file processing the MHAL message.

#### A.5.3.7.1 Synopsis

*#define MhalMsgField &MhalpS*

#### A.5.3.7.2 Arguments

None

#### A.5.3.7.3 Return Value

Structure representing the MHAL message being processed when the accessMhalMsg() macro was encountered

### A.5.3.8 endAccessMhalMsg Macro

This macro terminates the processing block of an MHAL message which was initiated with a call to AccessMhalMsg().

#### A.5.3.8.1 Synopsis

*#define endAccessMhalMsg*

#### A.5.3.8.2 Arguments

None

#### A.5.3.8.3 Return Value

None

### A.5.3.9 accessMhalField Macro

This macro is paired with the `endAccessMhalField` macro as a block structure to allow parsing the data within the two macro calls. This macro allows the developer to use the `Mhal_get`*`n`* macros in nested parsing functions that are only knowledgeable about the field or data structure they parse.

#### A.5.3.9.1 Synopsis

*#define accessMhalField(parseMsgStruct)*

#### A.5.3.9.2 Arguments

| Parameter Name | Description |
|---|---|
| parseMsgStruct | Pointer to the `MhalParseStruct` macro that was created by the call to `accessMhalMsg` macro. |

#### A.5.3.9.3 Return Value

None

### A.5.3.10 endAccessMhalField Macro

This macro is paired with the `accessMhalField` macro. This macro terminates the block structure. As a result of encountering this macro, the current read position in the `MhalParseStruct` passed into the `accessMhalField` macro is updated. Thus, any additional parsing of the message by other functions will commence on the next byte after accessed MHAL field.

#### A.5.3.10.1 Synopsis

*#define endAccessMhalField*

#### A.5.3.10.2 Arguments

None

#### A.5.3.10.3 Return Value

None

# APPENDIX A.A – ABBREVIATIONS AND ACRONYMS

| Acronym | Definition |
|---|---|
| ADDR | Address |
| AGC | Automatic Gain Control |
| ALC | Automatic Level Control |
| API | Application Program Interface |
| CE | Computational Element |
| CF | Core Framework |
| CLK | Clock |
| CORBA | Common Object Request Broker Architecture |
| DOD | Department of Defense |
| DSP | Digital Signal Processor |
| EN | Enable |
| EOM | End Of Message |
| FIFO | First In First Out |
| FPGA | Field Programmable Gate Array |
| FREQ | Frequency |
| GPP | General Purpose Processor |
| HDL | Hardware Description Language |
| HW | Hardware |
| ID | Identifier |
| IDL | Interface Definition Language |
| IU | In Use |
| JPEO | Joint Program Executive Office |
| JTNC | Joint Tactical Networking Center |
| JTR | Joint Tactical Radio |
| JTRS | Joint Tactical Radio System |
| LD | Logical Destination |
| LSB | Least Significant Byte |
| MHAL | Modem Hardware Abstraction Layer |
| MSB | Most Significant Byte |
| PF | Platform |
| RAM | Random Access Memory |
| REG | Regular |
| RF | Radio Frequency |
| RFC | RF Chain |
| rsp | Response |
| RX | Receive |
| SCA | Software Communications Architecture |
| STRB | Strobe |
| SW | Software |
| TX | Transmit |
| UML | Unified Modeling Language |
| VHDL | VHSIC Hardware Description Language |
| WF | Waveform |

# APPENDIX A.B – PERFORMANCE SPECIFICATION

Not applicable

# B. MHAL GPP API EXTENSION

## B.1 INTRODUCTION

The *MHAL Device* supports methods and attributes that are specific to the General Purpose Processor (GPP) Modem Hardware (HW) device it represents. For the purposes of this API the following applies to processor naming conventions:

- A GPP represents a CORBA capable processor (This could be a DSP that supports CORBA.)

SCA devices do not have a single interface, but should instead be considered a composition of objects and interfaces. As an example, each "provides" port on a device represents a CORBA servant. As a consequence, a device cannot be generally defined with a single IDL file. This API documents a GPP Modem HW component rather than a single interface.

This API provides information to the software developer to utilize the *MHAL Device* interfaces in the Waveform target configurations.

The General Purpose Processor (GPP) portion of the MHAL defines a SCA provides port, *MHALPacketConsumer* interface, as the mechanism by which data is moved between waveform applications and the *MHAL Device*. *MHAL Device* defines a SCA provides port that provides the *MHALPacketConsumer* interface and *MHAL Device* defines a SCA uses port that uses the same interface. It is assumed that each waveform component that interfaces with *MHAL Device* with the intention of sending data through *MHAL Device* will implement a uses port and that each waveform component that interfaces with *MHAL Device* with the intention of receiving data from *MHAL Device* will implement a provides port.

### B.1.1 Overview

This document contains as follows:

a. Section B.1, *Introduction*, of this document contains the introductory material regarding the overview, and Service Layer description.

b. Section B.2, *Services*, provides summary of service interface uses, interface for each device component, port connections, and sequence diagrams.

c. Section B.3, *Service Primitives and Attributes*, specifies the operations that are provided by the *MHAL Device*.

d. Section B.4, *IDL*

e. Section B.5, *Data Types and Exceptions*.

f. Appendix B.A – *Abbreviations and Acronyms*

g. Appendix B.B – *Performance Specification*

# B.1.2 Service Layer Description

## B.1.2.1 MHAL Port Connections

**Figure 3** shows the port connections for the *MHAL*.

Note: All port names are for reference only.

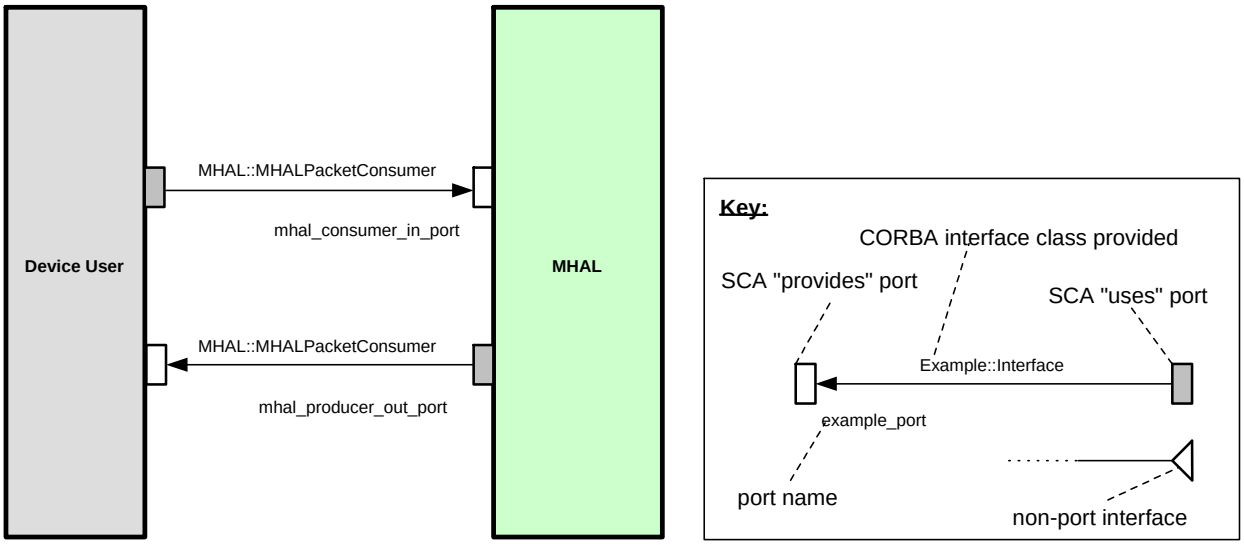


**Figure 3 – MHAL Port Diagram**

***MHAL* Provides Ports Definitions**

**mhal_consumer_in_port** is provided by the *MHAL* to consume packet data through operations such as pushPacket, get packet size information and addTxRoute.

***MHAL* Uses Ports Definitions**

**mhal_producer_out_port** is used by the *MHAL* to push packet data through operations such as pushPacket, get packet size information and getRxRoutes.

## B.1.2.2 Modes of Service

Not applicable

## B.1.2.3 Service States

## B.1.2.4 MHAL State Diagram

The *MHAL* states are illustrated in **Figure 4**. The *MHAL* states ensure that received operations are only executed when the *MHAL* is in the proper state. The five states of the *MHAL* are as follow:

- CONSTRUCTED - The state transitioned to upon successful creation.
- INITIALIZED - The state transitioned to upon successful initialization.
- ENABLED - The state transitioned to upon successful start.
- DISABLED - The state transitioned to upon successful stop.
- RELEASED - The state transitioned to upon successful release.

The *MHAL* transitions between states in response to the initialize, start, stop and releaseObject operations.

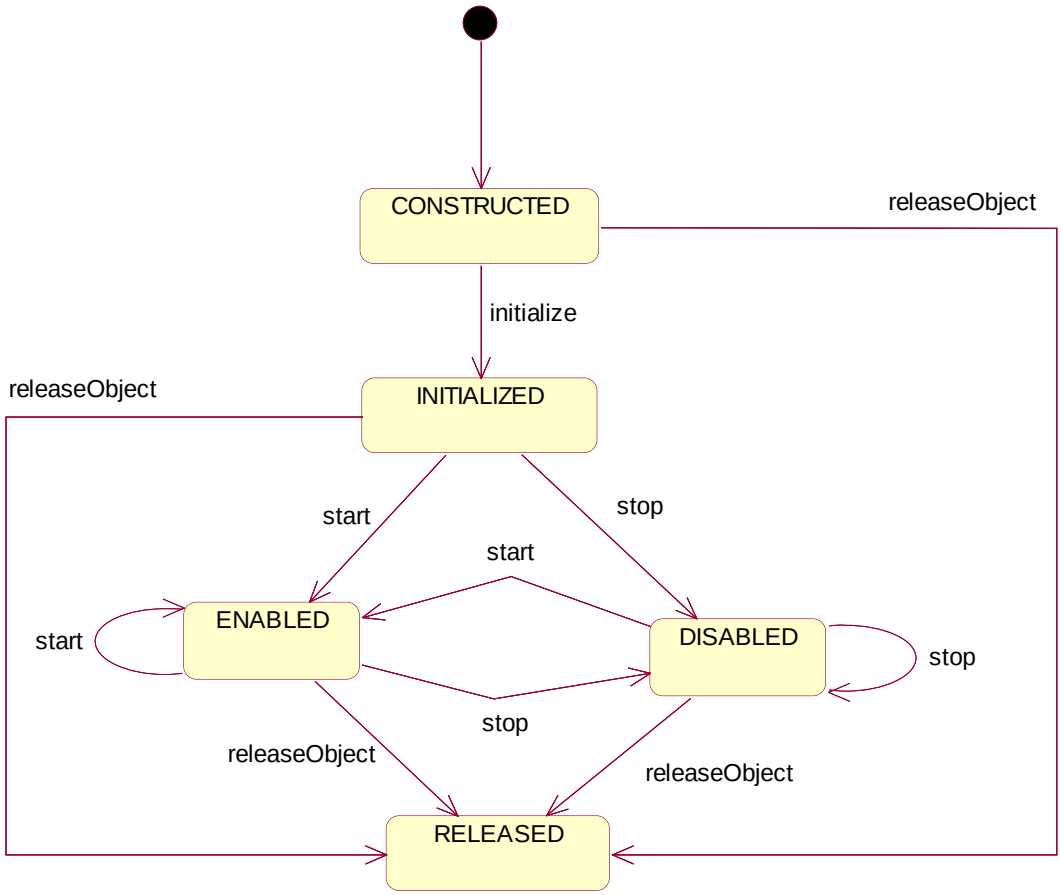


**Figure 4 – MHAL State Diagram**

# B.2 SERVICES

The MHAL CORBA-compliant API separates platform interfaces from waveform interfaces. MHAL::PF_MHALPacketConsumer specifies the interfaces accessed by the JTR set, and the MHAL::WF_MHALPacketConsumer specifies the interface accessed by a waveform or application.

## B.2.1 Provide Services

The *MHAL* Provide Service consists of the following service ports in **Table 2**, interfaces, and primitives, which can be called by other client components. Detail definition of the interfaces and services shaded in *grey* is provided by the Packet API (see reference [2]).

**Table 2 – MHAL Device Provide Service Interface**

| Service Group (Port Name) | Service (Interface Provided) | | Primitives (Provided) | Parameter Name or Return Value | Valid Range |
|---|---|---|---|---|---|
| mhal_consumer_in_port | MHAL::MHAL PacketConsumer | | pushPacket() | See section B.3 Service Primitives and Attributes | See section B.3 Service Primitives and Attributes |
| | | MHAL::PF_MHAL PacketConsumer | addTxRoute() | See section B.3 Service Primitives and Attributes | See section B.3 Service Primitives and Attributes |
| | | Packet:: PayloadStatus *(see reference [2])* | getMaxPayloadSize() | *Return Value* | N/A |
| | | | getMinPayloadSize() | *Return Value* | N/A |
| | | | getDesiredPayloadSize()* | *Return Value* | 0 |
| | | | getMinOverrideTimeout()* | *Return Value* | 0 |

* Operation usage not required.

## B.2.2 Use Services

The *MHAL* Use Service set consists of the following service ports in **Table 3**, interfaces, and primitives. Since the *MHAL* acts as a client with respect to these services from other components, it is required to connect these ports with corresponding service ports applied by the server component. The *MHAL* uses the Port Name as connectionId for the connection.

**Table 3 – MHAL Use Service Interface**

| Service Group (Port Name) | Service (Interface Provided) | | Primitives (Provided) | Parameter Name or Return Value | Valid Range |
|---|---|---|---|---|---|
| mhal_producer_out_port | MHAL::MHALPacketConsumer | | pushPacket() | See section B.3 Service Primitives and Attributes | See section B.3 Service Primitives and Attributes |
| | | MHAL::WF_MHAL PacketConsumer | getRxRoutes() | See section B.3 Service Primitives and Attributes | See section B.3 Service Primitives and Attributes |
| | | Packet:: PayloadStatus (see reference [2]) | getMaxPayloadSize() | *Return Value* | N/A |
| | | | getMinPayloadSize() | *Return Value* | N/A |
| | | | getDesiredPayloadSize()* | *Return Value* | 0 |
| | | | getMinOverrideTimeout()* | *Return Value* | 0 |

* Operation usage not required.

## B.2.3 Interface Modules

### B.2.3.1 MHAL

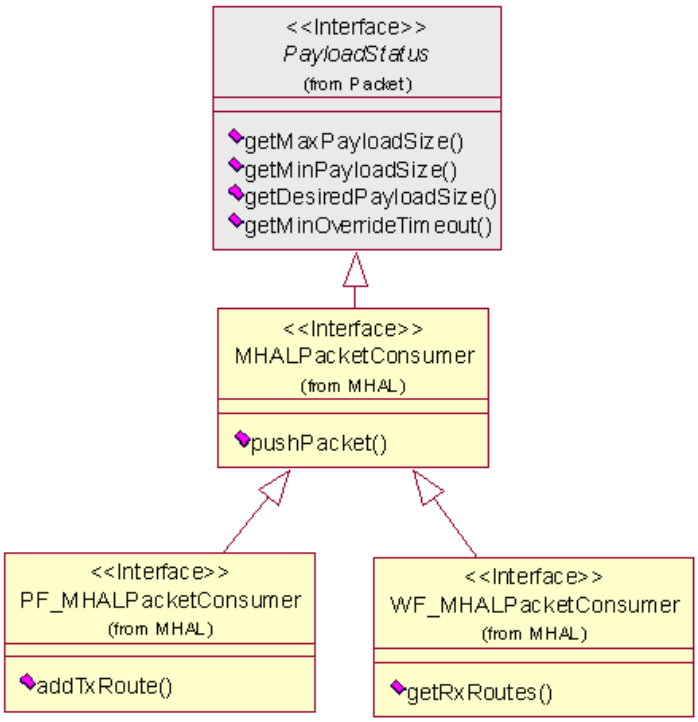


**Figure 5 – MHAL Interface Class Diagram**

#### B.2.3.1.1 MHALPacketConsumer Interface Description

The interface design of the *MHALPacketConsumer* is shown in **Figure 5**. It extends the *Packet::PayloadStatus* interface to provide the ability to receive a data packet to and from a service user/provider (*See Packet API* [2]).

#### B.2.3.1.2 PF_MHALPacketConsumer Interface Description

The interface design of the *PF_MHALPacketConsumer* is shown in **Figure 5**. It extends the *MHALPacketConsumer* interface to provide the ability for the waveform to add a mapping between a logical destination and a physical destination for the service user.

#### B.2.3.1.3 WF_MHALPacketConsumer Interface Description

The interface design of the *WF_MHALPacketConsumer* is shown in **Figure 5**. It extends the *MHALPacketConsumer* interface to provide the ability for the MHAL to retrieve the logical destinations contained within a waveform component.

# B.3 SERVICE PRIMITIVES AND ATTRIBUTES

To enhance the readability of this API document and to avoid duplication of data, the type definitions of all structured types (i.e., structures, typedefs, exceptions, enumerations and unions) used by the Service Primitives and Attributes have been co-located in section B.5 Data Types and Exceptions. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

## B.3.1 MHAL::MHALPacketConsumer

### B.3.1.1 *pushPacket* Operation

The *pushPacket* operation provides the ability to push data packets to the packet consumer.

#### B.3.1.1.1 Synopsis

*void pushPacket (in unsigned short logicalDest, in JTRS::OctetSequence payload);*

#### B.3.1.1.2 Parameters

| Parameter Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| logicalDest | The logical destination for the message | unsigned short | Logical Destination ID | 0 – 32767 |
| payload | The real-time data (includes logical destination & message length) | JTRS::OctetSequence (see reference [1]) | N/A | N/A |

#### B.3.1.1.3 State

ENABLED CF::Device::operationalState.

#### B.3.1.1.4 New State

This operation does not cause a state change.

#### B.3.1.1.5 Return Value

None

#### B.3.1.1.6 Originator

Service Provider

#### B.3.1.1.7 Exceptions

None

## B.3.2 MHAL::PF_MHALPacketConsumer

### B.3.2.1 *addTxRoute* Operation

The *addTxRoute* operation saves the passed in parameters as a mapping between a logical destination and a physical destination. This information is used in future *pushPacket* calls for routing the received message. For the MHAL GPP, the mapping between logical and physical destinations is done at run-time. A WF must call this operation for each logical destination.

Note: In the *MHAL DSP*, the mapping between logical and physical destinations may be achieved with a header file at build time.

#### B.3.2.1.1 Synopsis

*void addTxRoute ( in unsigned short logicalDest,*
*in MHAL::MHALPhysicalDestination physicalDest);*

#### B.3.2.1.2 Parameters

| Parameter Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| logicalDest | The logical destination for the message | unsigned short | N/A | 0 – 32767 |
| physicalDest | Physical destination | MHAL::MHALPhysicalDestination | unsigned short | See B.5.1.1 |

#### B.3.2.1.3 State

ENABLED CF::Device::operationalState.

#### B.3.2.1.4 New State

This operation does not cause a state change.

#### B.3.2.1.5 Return Value

None

#### B.3.2.1.6 Originator

Service Provider.

#### B.3.2.1.7 Exceptions

None

# B.3.3 MHAL::WF_MHALPacketConsumer

## B.3.3.1 getRxRoutes Operation

The *getRxRoutes* operation is intended for waveform developers who implement the MHAL Interface. This method will be called from the MHAL in order to retrieve the logical destinations contained within a waveform component. This routing information is used by MHAL when it receives a message and must route it to a component within the GPP.

### B.3.3.1.1 Synopsis

*MHAL::WF_MHALPacketConsumer::RxRouteSequence getRxRoutes ();*

### B.3.3.1.2 Parameters

None

### B.3.3.1.3 State

ENABLED CF::Device::operationalState.

### B.3.3.1.4 New State

This operation does not cause a state change.

### B.3.3.1.5 Return Value

| Description | Type | Units | Valid Range |
|---|---|---|---|
| Returns a valid sequence containing each logical destination ID for which the waveform requests messages.<br><br>A sequence of zero (0) length indicates that the waveform application has not completed configuring these logical destinations and not all logical destinations are known.<br><br>A sequence of one (1) length and a single value of 0 (special NULL logical destination ID) indicates that the waveform application does not wish to receive messages from MHAL. | MHAL:: WF_MHALPacketConsumer:: RxRouteSequence | Logical Destination ID | N/A |

### B.3.3.1.6 Originator

Service Provider.

### B.3.3.1.7 Exceptions

None

# B.4 IDL

## B.4.1 MhalDevice IDL

```
/*
** MhalDevice.idl
*/

#ifndef __MHALDEVICE_DEFINED
#define __MHALDEVICE_DEFINED

#ifndef __JTRSCORBATYPES_DEFINED
#include "JtrsCorbaTypes.idl"
#endif
#ifndef __PACKET_DEFINED
#include "Packet.idl"
#endif

module MHAL {

        interface MHALPacketConsumer : Packet::PayloadStatus {

                void pushPacket (
                        in unsigned short logicalDest,
                        in JTRS::OctetSequence payload
                );

        };

        // Known MHAL Physical Destination Types
        typedef JTRS::ExtEnum MHALPhysicalDestination;
        const MHALPhysicalDestination   MHALPhysicalDestination_NONE = 0;
        const MHALPhysicalDestination   PHYSICAL_DESTINATION_DSP = MHALPhysicalDestination_NONE+1;
        const MHALPhysicalDestination   PHYSICAL_DESTINATION_FPGA = PHYSICAL_DESTINATION_DSP +1;
        const MHALPhysicalDestination   PHYSICAL_DESTINATION_GPP = PHYSICAL_DESTINATION_FPGA +1;

        interface PF_MHALPacketConsumer : MHALPacketConsumer {
                // Following type and consts are deprecated - Use versions scoped to MHAL module!
                typedef MHAL::MHALPhysicalDestination MHALPhysicalDestination;
                const MHALPhysicalDestination   MHALPhysicalDestination_NONE = MHAL::MHALPhysicalDestination_NONE;
                const MHALPhysicalDestination   PHYSICAL_DESTINATION_DSP = MHAL::PHYSICAL_DESTINATION_DSP;
                const MHALPhysicalDestination   PHYSICAL_DESTINATION_FPGA = MHAL::PHYSICAL_DESTINATION_FPGA;
                const MHALPhysicalDestination   PHYSICAL_DESTINATION_GPP = MHAL::PHYSICAL_DESTINATION_GPP;

                void addTxRoute (
```

```
                        in unsigned short logicalDest,
                        in MHAL::MHALPhysicalDestination physicalDest
                );

        };

        interface WF_MHALPacketConsumer : MHALPacketConsumer {

                typedef sequence<unsigned short> RxRouteSequence;

                MHAL::WF_MHALPacketConsumer::RxRouteSequence getRxRoutes ();

        };

};

#endif
```

# B.5 DATA TYPES AND EXCEPTIONS

This section contains the Device component UML diagram and the definitions of all data types referenced (directly or indirectly) by section B.3 Service Primitives and Attributes.

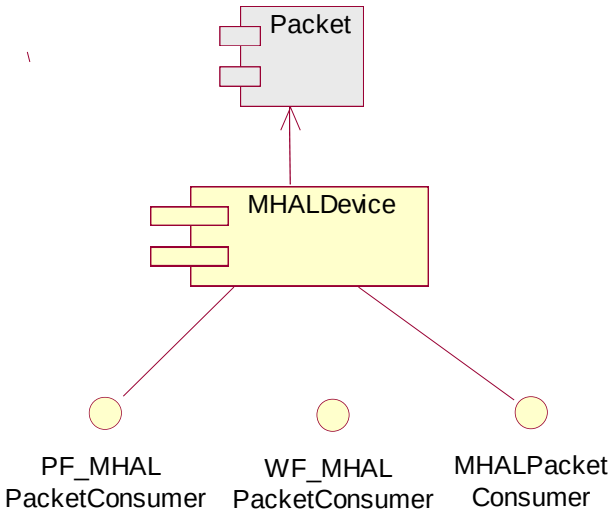


**Figure 6 – MHAL Component Diagram**

## B.5.1 MHAL Types

### B.5.1.1 MHAL::MHALPhysicalDestination Type

The MHALPhysicalDestination type is used to set the physical destination.

| Typedef | Implementation Type |
|---|---|
| MHALPhysicalDestination | unsigned short |

The following table defines the valid set of values for the above type:

| Type | Valid Range | Description |
|---|---|---|
| MHALPhysicalDestination | PHYSICAL_DESTINATION_DSP | DSP physical destination |
| | PHYSICAL_DESTINATION_FPGA | FPGA physical destination |
| | PHYSICAL_DESTINATION_GPP | GPP physical destination |

## B.5.2 MHAL::PF_MHALPacketConsumer Types

### B.5.2.1 MHAL::PF_MHALPacketConsumer::MHALPhysicalDestination Type <Deprecated>

The MHALPhysicalDestination type is used to set the physical destination.

| Typedef | Implementation Type |
|---|---|
| MHALPhysicalDestination | unsigned short |

The following table defines the valid set of values for the above type:

| Type | Valid Range | Description |
|---|---|---|
| MHALPhysicalDestination | PHYSICAL_DESTINATION_DSP | DSP physical destination |
| | PHYSICAL_DESTINATION_FPGA | FPGA physical destination |
| | PHYSICAL_DESTINATION_GPP | GPP physical destination |

## B.5.3 MHAL::WF_MHALPacketConsumer Types

### B.5.3.1 MHAL::WF_MHALPacketConsumer::RxRouteSequence Type

The RxRouteSequence defines an unsigned short CORBA sequence of logical destination IDs.

| Typedef | Implementation Type |
|---|---|
| RxRouteSequence | sequence<unsigned short> |

# APPENDIX B.A – ABBREVIATIONS AND ACRONYMS

See section Appendix A.A.

# APPENDIX B.B – PERFORMANCE SPECIFICATION

Table 4 provides a template for the generic performance specification for the *MHAL GPP* API Extension that will be documented in the waveform or user using the interface. This performance specification corresponds to the port diagram in Figure 3.

**Table 4 – MHAL Performance Specification**

| Specification | Description | Units | Value |
|---|---|---|---|
| Worst Case Command Execution Time for mhal_consumer_in_port | * | * | * |
| Worst Case Command Execution Time for mhal_producer_out_port | * | * | * |

Note: (*) These values should be filled in by individual developers.

# C. MHAL DSP API EXTENSION

## C.1 INTRODUCTION

The *MHAL DSP* API consists of a collection of C function specifications that provide services to route MHAL communications, and MHAL message status. For the purposes of this API the following applies to processor naming conventions:

- A DSP represents a C capable processor, but does not provide CORBA capability.

The service user includes the C function specifications in the service users DSP code. Calls to the *MHAL DSP* functions are made from the service users DSP source code. It is important to structure aspects of the *MHAL DSP* and service user modularity and header file structure in order to facilitate an orderly build process and to minimize the effort involved in integrating a service user with a host-specific *MHAL DSP*.

### C.1.1 Overview

This document contains as follows:

a. Section C.1, *Introduction*, of this document contains the introductory material regarding the Overview.

b. Section C.2, *Services*, provides summary of service uses.

c. Section C.3, *Service Primitives and Attributes* specifies the functions that are provided by the *MHAL DSP*.

d. Section C.4, *Interface Definitions*

e. Section C.5, *Data Types and Exceptions* specifies the data types that are provided by the *MHAL DSP*.

*f. Appendix C.A – Abbreviations and Acronyms*

g. *Appendix C.B – Performance Specification*

# C.2 SERVICES

## C.2.1 MHAL Communication Routing Module

This feature provides additional capabilities to manage the *MHAL DSP* Data Sink. Support of this feature is through the functions defined in the following sections:

- C.3.1.1 *Mhal_Comm* Function
- C.3.1.2 reroute_LD_sink Function
- C.3.1.3 *LD_of* Function

## C.2.2 MHAL Message In-Use Module (*optional*)

This feature allows the waveform to be able to detect a message buffer that is still In-Use by a Data Sink. Support of this feature is through the functions defined in the following sections:

- C.3.2.1 setMsgInUse Function
- C.3.2.2 clrMsgInUse Function
- C.3.2.3 isMsgInUse Function

These functions utilize the IU bit of the MHAL Message (A.2.1.3.1.1 MHAL Message IU Bit). Communication devices use these message in-use functions to indicate that the message is in use until successfully transmitted over the communication device.

# C.3 SERVICE PRIMITIVES AND ATTRIBUTES

To enhance the readability of this API document and to avoid duplication of data, the type definitions of all structured types (i.e., structures and typedefs) used by the module messages have been co-located in section C.4.

## C.3.1 MHAL Communication Routing

### C.3.1.1 *Mhal_Comm* Function

This function acts as a router. In the DSP, messages are launched by calling `Mhal_Comm` function with the Message Pointer parameter (`bufferPtr`) pointing to the Logical Destination of the message to be sent. This function is equivalent to a one way `pushPacket()` call in CORBA on the MHAL GPP.

#### C.3.1.1.1 Synopsis

*void Mhal_Comm (MhalByte* bufferPtr);*

#### C.3.1.1.2 Arguments

| Argument Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| bufferPtr | A byte pointer to a message that needs to be sent. | MhalByte* | N/A | N/A |

#### C.3.1.1.3 Return Value

None

### C.3.1.2 reroute_LD_sink Function

This function allows a waveform to reroute the Data Sink Function for a given LD to a different LD during waveform operation.

#### C.3.1.2.1 Synopsis

*funPtr reroute_LD_sink (short LD, funPtr newSinkFx);*

#### C.3.1.2.2 Arguments

| Argument Name | Description | Type |
|---|---|---|
| LD | Logical destination that is being rerouted to new Data Sink function | short |
| newSinkFx | The new Data Sink Function (fx) that is called when a message is sent to the LD | funPtr |

#### C.3.1.2.3 Return Value

| Description | Type |
|---|---|
| Previous Data Sink Function used for the LD. This can be used to restore the sink process normally associated with the sink. | funPtr |

### C.3.1.3 *LD_of* Function

Function to return the extracted LD from the message. The returned LD is only the 15 bit LD value.

#### C.3.1.3.1 Synopsis

*short LD_of (MhalByte* msgPtr);*

#### C.3.1.3.2 Arguments

| Argument Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| msgPtr | A pointer to an MHAL Message. | MhalByte* | N/A | N/A |

#### C.3.1.3.3 Return Value

A 16-bit value containing the LD from the MHAL message.

## C.3.2 MHAL Message In-Use (*optional*)

### C.3.2.1 setMsgInUse Function

This function marks the message as “in use” because the targeted Data Sink is busy.

A *setMsgInUse* call must be paired with a *clrMsgInUse* call before a sequential *setMsgInUse* call is made.

#### C.3.2.1.1 Synopsis

*void setMsgInUse (MhalByte* msgPtr);*

#### C.3.2.1.2 Arguments

| Argument Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| msgPtr | A pointer to an MHAL message. | MhalByte* | N/A | N/A |

#### C.3.2.1.3 Return Value

None

### C.3.2.2 clrMsgInUse Function

This function indicates that a message has been processed sufficiently and that the storage space related to the message can be reused. This function marks the message as not "In-Use".

#### C.3.2.2.1 Synopsis

*void clrMsgInUse (MhalByte* msgPtr);*

#### C.3.2.2.2 Arguments

| Argument Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| msgPtr | A pointer to an MHAL message. | MhalByte* | N/A | N/A |

#### C.3.2.2.3 Return Value

None

### C.3.2.3 isMsgInUse Function

This function examines the message and determines if the `setMsgInUse()` function has been called against the message and no matching `clrInUseMsg()` call has been made.

#### C.3.2.3.1 Synopsis

*bool isMsgInUse(MhalByte* msgPtr);*

#### C.3.2.3.2 Arguments

| Argument Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| msgPtr | A pointer to an MHAL message. | MhalByte* | N/A | N/A |

#### C.3.2.3.3 Return Value

A boolean signifying whether the IU bit (i.e. busy bit) is set or cleared. `TRUE` when a Data Sink Function has claimed the message as still needing processing (i.e. the `setMsgInUse()` function has been called and no corresponding `clrMsgInUse()` has been called). `FALSE` when the message is no longer being claimed by a Data Sink Function.

# C.4 INTERFACE DEFINITIONS

None

# C.5 DATA TYPES AND EXCEPTIONS

## C.5.1 MHAL Communication Routing Types

### C.5.1.1 funPtr Type

This type defines a function pointer prototype for a Data Sink Function that executes on the DSP.

#### C.5.1.1.1 Synopsis

**typedef void(*funPtr)(MhalByte* msgPtr);**

# APPENDIX C.A – ABBREVIATIONS AND ACRONYMS

See section Appendix A.A.

# APPENDIX C.B – PERFORMANCE SPECIFICATION

Not applicable

# D. MHAL FPGA API EXTENSION

## D.1 INTRODUCTION

The *MHAL FPGA* API consists of a collection of transmit and receive node user signal and timing descriptions that provide services to route MHAL communications. For the purposes of this API the following applies to processor naming conventions:

- An FPGA represents a HDL capable processor, again without CORBA capability.

At build time, the MHAL FPGA interface VHDL is compiled together with the waveform HDL to form a single loadable FPGA image for the target platform. A VHDL entity description defines each MHAL interface component available to the waveform FPGA developer.

### D.1.1 Overview

This document contains as follows:

a. Section D.1, *Introduction*, of this document contains the introductory material regarding the Overview, and Service Layer description.

b. Section D.2, *Services*

c. Section D.3, *Service Primitives and Attributes*

d. Section D.4, *Entity Definitions*

e. Section D.5, *Data Types and Exceptions*

f. Appendix D.A – *Abbreviations and Acronyms*

g. Appendix D.A – *Abbreviations and Acronyms*

### D.1.2 Service Layer Description

#### D.1.2.1 MHAL FPGA Signals

##### D.1.2.1.1 Transmit Node

Transmit nodes are used to send data from an I/O device to any other device in the system. Two different transmit nodes are available to the user.

###### D.1.2.1.1.1 Multi-Depth FIFO Tx Node

The MHAL Multi-Depth FIFO Transmit Communication Device has the following characteristics:

- Allows for multiple messages to be queued up to transmit.
- User must create the entire message (No Auto-Header Implemented).
- Any User Clock can be used.
- Allows for byte granularity of the message.

Figure 7 describes a Multi-Depth FIFO Transmit Communication Device that can be instantiated to allow communication from the waveform section to the outside software and hardware components.

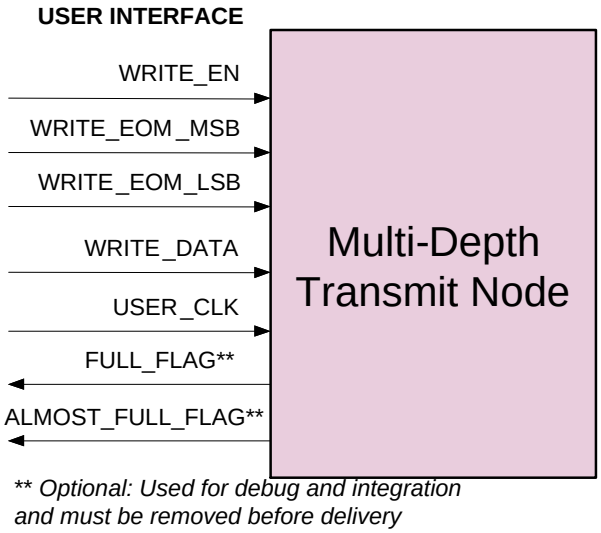


**Figure 7 – Multi-Depth FIFO Transmit Communication Device**

In Figure 7, the User Interface signals behave as:

| | |
|---|---|
| WRITE_EN | Write enable for the data contained in WRITE_DATA |
| WRITE_EOM_MSB | "1" indicates "end of message" in the high byte of the data |
| WRITE_EOM_LSB | "1" indicated "end of message" in the low byte of the data |
| WRITE_DATA | 16-bit wide data bus |
| USER_CLK | Data is strobed into the TX Node on rising edge of the USER_CLK when WRITE_EN is a "1" |
| FULL_FLAG | Indicates Full FIFO (*optional*) |
| ALMOST_FULL_FLAG | Indicates Almost Full FIFO, i.e. room for at least 1 more word (*optional*) |

The Multi-Depth Transmit FIFO supports buffering of one or more MHAL messages. The Data Sink Function must be accessible to the transmit chain where this interface is deployed. That is the Data Sink Function can be within the current FPGA or external to the FPGA via a transport device. The interface requires that the entire MHAL message, MHAL header and payload, be provided. Thus, when the Multi-Depth FIFO Transmit node is used, messages can be queued up to different Data Sink Functions and delayed by the number of messages ahead of a message in the queue. The device interface associated with the transport between two CEs can result in a clock rate that is different than the user provided clock. The implementation of the Multi-Depth Transmit queue must size the FIFO large enough to hold the expected traffic and includes delays caused by all other transmit nodes connected to the same transport chain.

## D.1.2.1.2 Receive Nodes

The figures 8-12, describe an MHAL Receive Communication Device that can be instantiated to allow communication from the waveform section to the outside software and hardware components.

#### D.1.2.1.2.1 Multi-Depth FIFO Rx Node

The MHAL Multi-Depth FIFO Receive Communication Device has the following characteristics:

- Allows multiple messages to be queued up for use by the user.
- Any User Clock can be used.
- Only receive node that allows for byte granularity of the message.
- LD associated with node defined as parameter to instantiation of the node in VHDL.

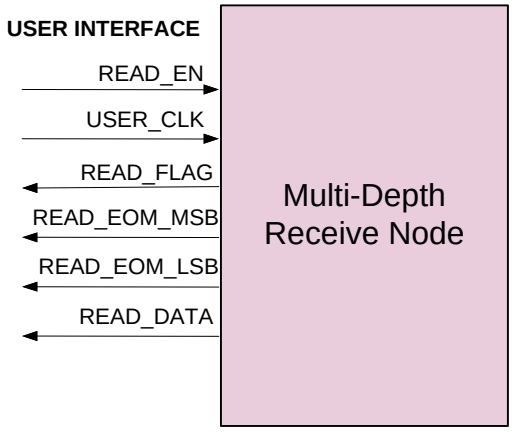


**Figure 8 – Multi-Depth FIFO Receive Communication Device**

In Figure 8, the User Interface signals behave as:

| | |
|---|---|
| READ_EN | Read enable for the data contained in READ_DATA |
| USER_CLK | Clock supplied by user that sets output rate. Rising edge of the USER_CLK when WRITE_EN is where data changes |
| READ_FLAG | "1" indicates packet received and ready to process by user |
| READ_EOM_MSB | "1" indicates "end of message" in the high byte of the data |
| READ_EOM_LSB | "1" indicated "end of message" in the low byte of the data |
| READ_DATA | 16 bit wide data bus |

#### D.1.2.1.2.2 Single-Depth FIFO Rx Node

The MHAL Single-Depth FIFO Receive Communication Device has the following characteristics:

- User Data received from node and placed into a FIFO of user-defined width and depth (Note: The width must be a multiple of 16 bits).
- User must read all data before USER_DATA_NEW is signaled.

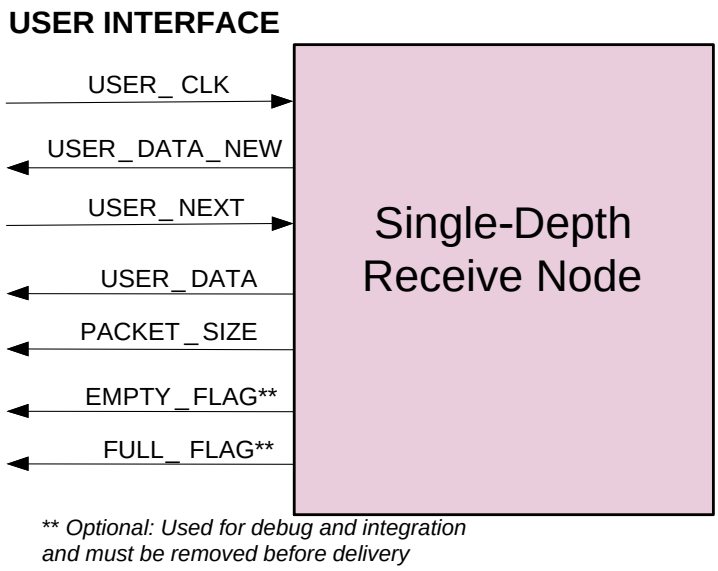


**Figure 9 – Single-Depth FIFO Receive Communication Device**

In Figure 9, the User Interface signals behave as:

| | |
|---|---|
| USER_CLK | Clock supplied by the User that sets the output rate. |
| USER_DATA_NEW | New User Data Available |
| USER_NEXT | Acknowledge USER_DATA has been read, advance to next data word |
| USER_DATA | User-specified-width data bus |
| PACKET_SIZE | Indicates Message Size |
| EMPTY_FLAG | Indicates Empty FIFO *(optional)* |
| FULL_FLAG | Indicates Full FIFO *(optional)* |

### D.1.2.1.2.3 RAM Rx Node

The MHAL RAM Receive Communication Device has the following characteristics:

- User Data received from node and placed in a user-defined width by user-defined length RAM (Note: The width must be a multiple of 16 bits.)
- Useful for interleavers.
- LD associated with node defined as parameter to instantiation of the node in VHDL.

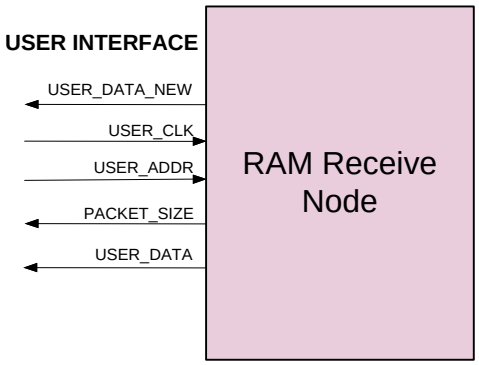


**Figure 10 – RAM Receive Communication Device**

In Figure 10, the User Interface signals behave as:

| | |
|---|---|
| USER_DATA_NEW | New User Data Available |
| USER_CLK | Clock supplied by the User that sets the output rate. |
| USER_ADDR | RAM Address Signal |
| PACKET_SIZE | Indicates Message Size |
| USER_DATA | User-specified-width data bus |

### D.1.2.1.2.4 N-Word Rx Register

The MHAL N-Word Register Receive Communication Device has the following characteristics:

- User Data received from node and placed in a register of user-defined width (Note: The width must be a multiple of 16 bits.
- Useful configuration and status.
- LD associated with node defined as parameter to instantiation of the node in VHDL.

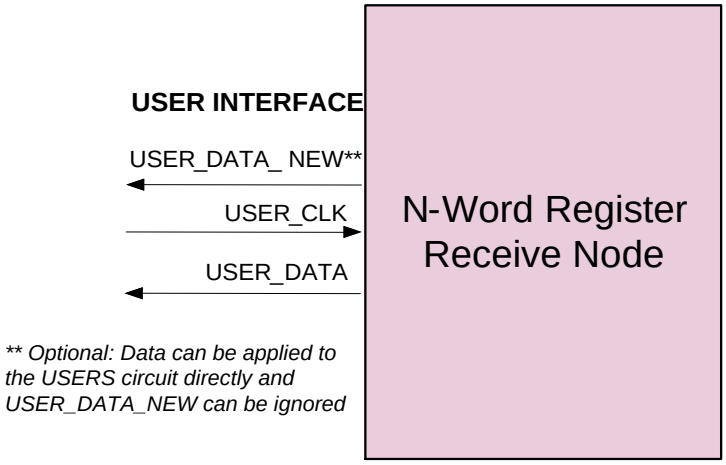


**Figure 11 – N-Word Register Receive Communication Device**

In Figure 11, the User Interface signals behave as:

| | |
|---|---|
| USER_DATA_NEW | New user data available, can be used to allow the USER to decide when the data should be applied *(optional)* |
| USER_CLK | Clock supplied by the User that sets the output rate |
| USER_DATA | User-specified-width data bus |

### D.1.2.1.2.5 Strobe Rx Node

The MHAL Strobe Receive Communication Device has the following characteristics:

- `Mhal_Comm()` message with no payload, typically used as an event.
- LD associated with node defined as parameter to instantiation of the node in VHDL.

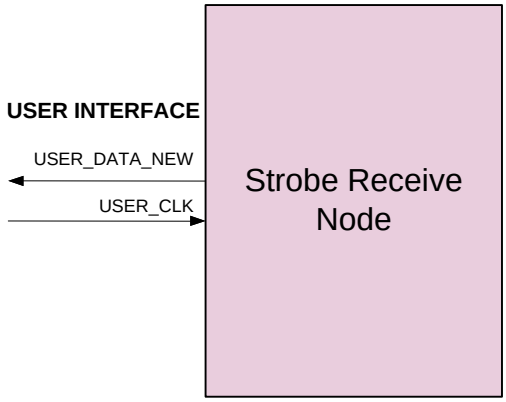


**Figure 12 – Strobe Receive Communication Device**

In Figure 12, the User Interface signals behave as:

| | |
|---|---|
| USER_DATA_NEW | Receive Node was hit with a message |
| USER_CLK | Clock supplied by the User that sets the output rate |

## D.1.2.2 MHAL FPGA Timing

## D.1.2.2.1 Transmit Node

### D.1.2.2.1.1 Multi-Depth FIFO Tx Node

**Figure 13** shows a generalized timing diagram for the Multi-Depth FIFO Transmit Node.

The original message to the DSP is formatted as follows:

```
LD     SIZE   PAYLOAD
0222   000C    26,C1,   E8,14,   D7,B9,   CF,38
```

For this node the depth is set to 1024 and the data is 16 bit. This node is limited to a message queue of 32 messages. The total size of any one message or multiple messages is not allowed to exceed 1024 words. This node requires that the LD and SIZE of each message be written as the first 4 bytes of the PAYLOAD.

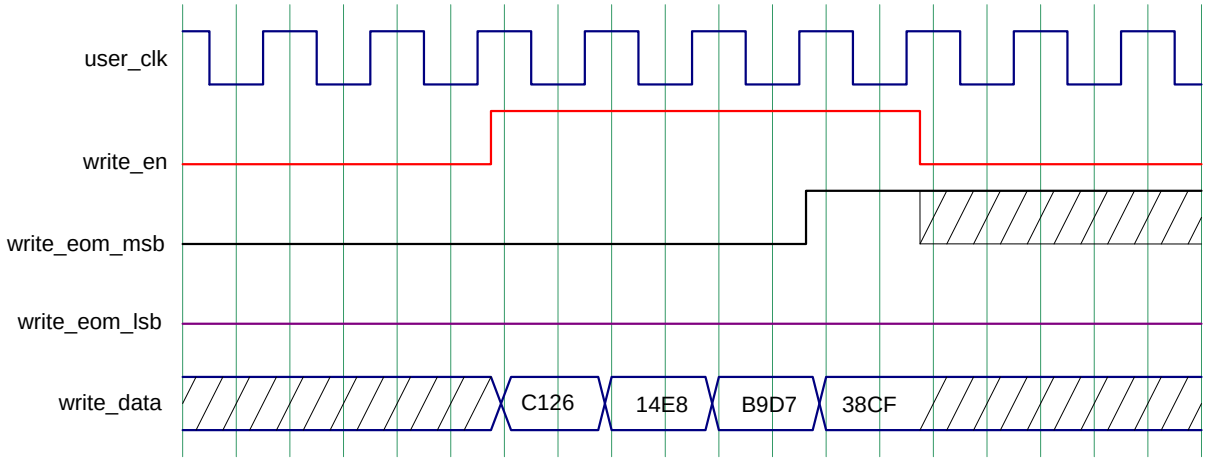


**Figure 13 – Reference Timing Diagram for Multi-Depth FIFO Transmit Node**

For **Figure 13**, the timing description is as follows:

1. The WRITE_EN signal shall enable the FIFO to write the contents of the WRITE_DATA bus to the next FIFO address. If WRITE_EN is high on the rising edge of the WRITE_CLK, the data bus will be copied to the FIFO. The WRITE_EN signal goes high as the first data word of the message is clocked onto the bus. WRITE_EN stays high for the duration of the message, returning low when the first null word (word not belonging to this or any other message) is clocked on the data bus.
2. WRITE_EOM_MSB and WRITE_EOM_LSB are used to indicate the position of the last message byte. When the message contains an odd number of bytes, the WRITE_EOM_LSB is set. When the message contains an even number of bytes, the WRITE_EOM_MSB is set.

## D.1.2.2.2 Receive Nodes

### D.1.2.2.2.1 Multi-Depth FIFO Rx Node

**Figure 14** shows a generalized timing diagram for the Multi-Depth FIFO Receive Node.

The original message from the DSP is formatted as follows:

```
LD     SIZE   PAYLOAD
0214   000C   3D,A0,   26,C1,   E8,14,   D7,B9
```

The total PAYLOAD of all messages written to this node cannot exceed the total depth of the NODE. For example: If the node is instantiated with a depth of 1024 (i.e. number of words), then can write 1 1024 WORD message or 2 512 WORD messages or some other unequal combination of WORD message sizes as long as the total is not more than 1024.

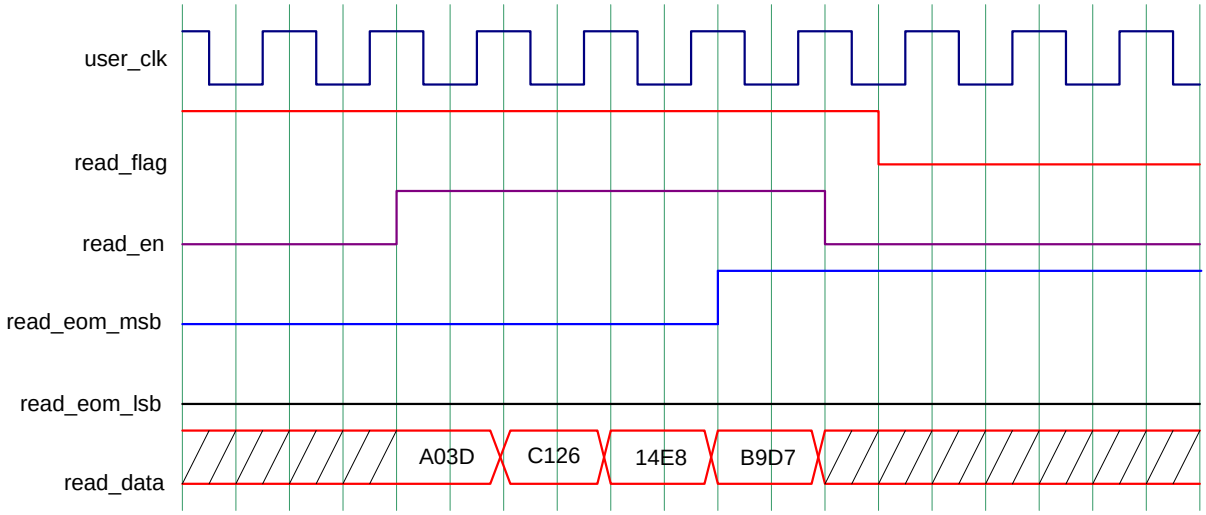


**Figure 14 – Reference Timing Diagram for Multi-Depth FIFO Receive Node**

For **Figure 14**, the timing description is as follows:

1. READ_FLAG indicates a complete message has been written to the FIFO and is ready to be read.
2. READ_EN will read the contents of the next FIFO address and place it on the READ_DATA bus. A new message word will be available on the FIFO data bus upon each rising USER_CLK edge while READ_EN is high. The first word is available before READ_EN is asserted and will be so until the first rising USER_CLK edge while READ_EN is high.
3. READ_EOM_MSB and READ_EOM_LSB are used to indicate the position of the last message byte. When the message contains an odd number of bytes, the READ_EOM_LSB is set. When the message contains an even number of bytes, the READ_EOM_MSB is set.

### D.1.2.2.2.2 Single-Depth FIFO Rx Node

**Figure 15** shows a generalized timing diagram for the 512x16 FIFO implemented with the MHAL Single-Depth FIFO Receive Node.

The original message from the DSP is formatted as follows:

```
LD     SIZE   PAYLOAD
0211   000A   3D,A0,   26,C1,   D7,B9
```

The PAYLOAD is shown in the BYTE order that the data is written. For this example, up to 512 16 bit words can be written to the FIFO at one time. The output data of this FIFO is not registered. This FIFO has been implemented in the Show-ahead synchronous FIFO mode. The data becomes available before the ‘rdreq’ (USER_NEXT) is asserted.

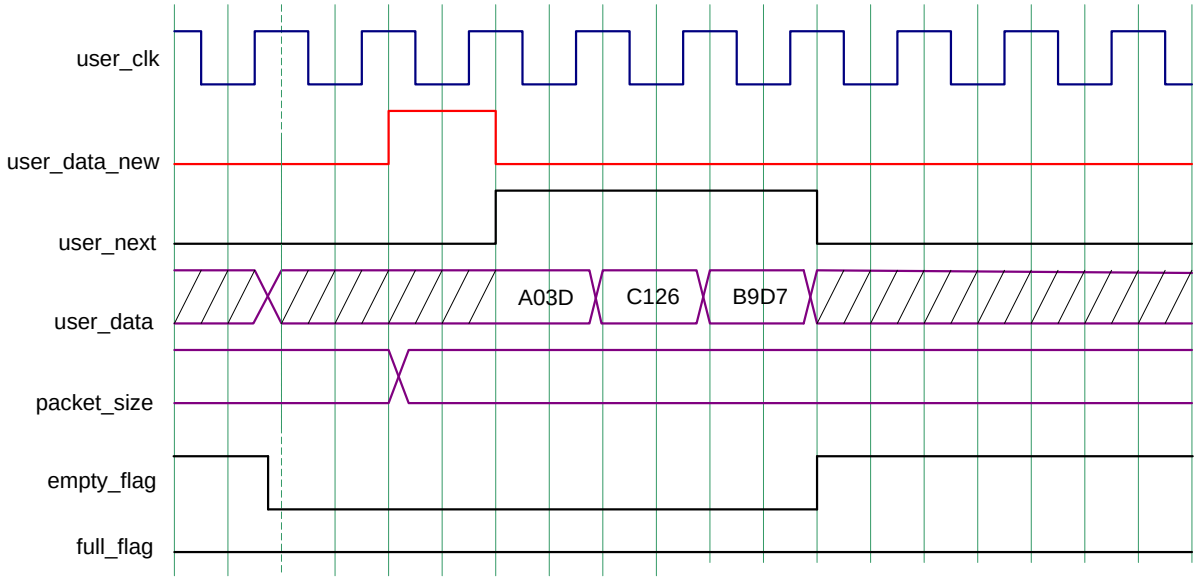


**Figure 15 – Reference Timing Diagram for Single-Depth FIFO Receive Node**

For **Figure 15**, the timing description is as follows:

1. USER_DATA_NEW signals the USER that USER_DATA is available. USER_DATA_NEW is one single pulse, the width of one USER_CLK.
2. USER_NEXT acts as a read acknowledge for each word of USER_DATA for every USER_CLK that it is high. All words must be acknowledged until the EMPTY_FLAG is set before another MHAL message can be sent.
3. The first word of USER_DATA is available immediately. Each word of USER_DATA must be acknowledged with USER_NEXT.
4. PACKET_SIZE is available at the start of the USER_DATA_NEW pulse, and will continue to be available until another MHAL message overwrites the node contents.
5. EMPTY_FLAG, FULL_FLAG. Accuracy of these two flags is 1 to 3 USER_CLKs after the data. This depends on the ratio of the MHAL_CLK and the USER_CLK. The larger the ratio the larger the accuracy error.

### D.1.2.2.2.3 RAM Rx Node

**Figure 16** shows a generalized timing diagram for the 256x16 RAM Receive Node.

This is an example of a 256x16 RAM implemented with the MHAL_RX_RAM node.
The original message from the DSP is formatted as follows:

```
LD      SIZE    PAYLOAD
0213    0012    00,02,  00,01,  00,08,  22,02,  E8,14,  26,C1,  CF, 38
```

The PAYLOAD is shown in the BYTE order that the data is written. This is a simple RAM. For this example, up to 256 16 bit words can be written to the RAM at one time. The first word in the PAYLOAD is written to address 00 in the RAM. The RAM address is incremented by one as the PAYLOAD data is written in. For this example, the RAM contents will be as shown below:

```
RAM ADDR      RAM DATA
```

```
01          0200
02          0100
03          0800
04          0222
05          14E8
06          C126
07          38CF
```

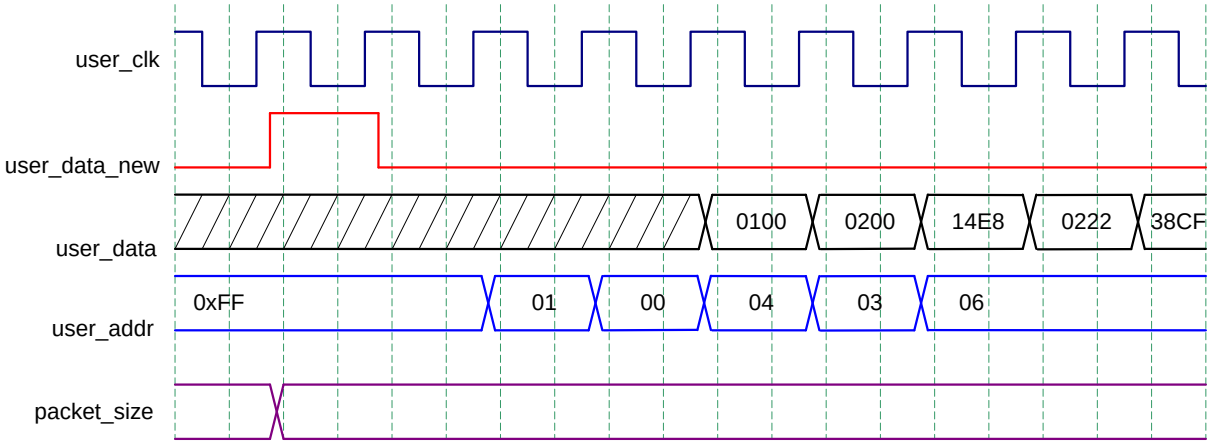


**Figure 16 – Reference Timing Diagram for RAM Receive Node**

For **Figure 16**, the timing description is as follows:

1. USER_DATA_NEW signals the USER that USER_DATA is available. USER_DATA_NEW is one single pulse, the width of one USER_CLK.
2. USER_DATA is available one USER_CLK after a valid USER_ADDR and will continue to be available until the USER_ADDR changes or another MHAL message overwrites the node contents. The output data in the RAM is registered on the USER_CLK.
3. PACKET_SIZE is available at the start of the USER_DATA_NEW pulse, and will continue to be available until another MHAL message overwrites the node contents.

### D.1.2.2.2.4 N-Word Rx Register

**Figure 17** shows a generalized timing diagram for the N-word Receive Register (32 bit with LSB on right hand side).

The original message from the DSP is formatted as follows:

```
LD      SIZE    PAYLOAD
0214    000A    22,02,01,00,02,00
```

The PAYLOAD is shown in the BYTE order that the data is written. Extra data is written to show that more data than the width of the register can be written, but only the last bytes equal to the width will be kept in the register.

Therefore, in the example above the 0222 is shifted out completely which leaves 0001,0002. Data is written in LSB BYTE to MSB BYTE order. So the first 2 bytes written in the PAYLOAD is the LSB WORD in the register. The last 2 bytes written is the MSB WORD in the register. If more data is written than the register can hold, then the LSB WORD is shifted out and all the other data words will shift MSB WORD to LSB WORD to allow the additional MSB WORD to shift in.

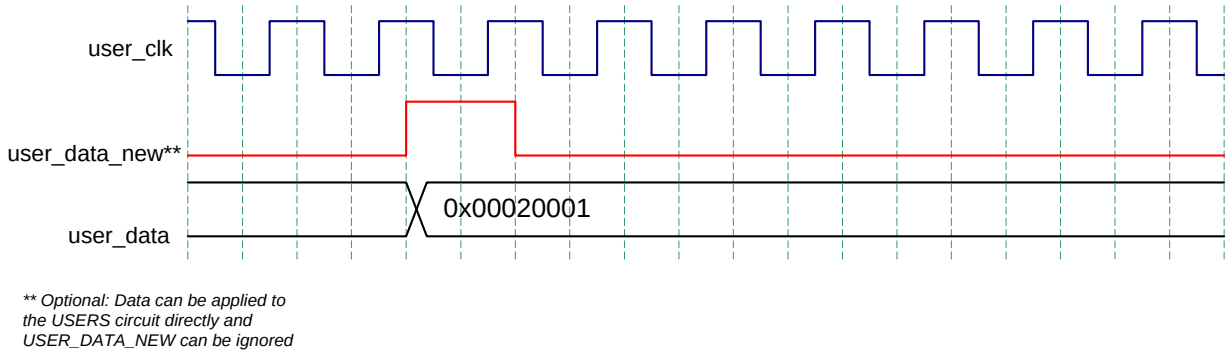


**Figure 17 – Reference Timing Diagram for N-word Receive Register**

For
**Figure 17**, the timing description is as follows:

1. USER_DATA_NEW signals the USER that USER_DATA is available. USER_DATA_NEW is one single pulse, the width of one USER_CLK.
    a. For example: If the timing between when the DSP updates the register and when the USER needs the data is not that well defined or cannot be closely defined, the USER can use this signal as an alert that the DSP has updated the register, then when the user is ready, the data can be grabbed at a later time. The USER should at least have some window within which the data must be used, because if the DSP sends another message to the register, before the USER reads it, the data will be overwritten with new data. It is the responsibility of the USER to understand their system timing and make adjustments for message latency to make sure data is used in a timely manner.
2. USER_DATA is available at the start of the USER_DATA_NEW pulse, and will continue to be available until another MHAL message overwrites the node contents.

### D.1.2.2.2.5 Strobe Rx Node

**Figure 18** shows a generalized timing diagram for a Strobe.

The original message from the DSP is formatted as follows:

```
LD     SIZE   PAYLOAD
0215   0004   None
```

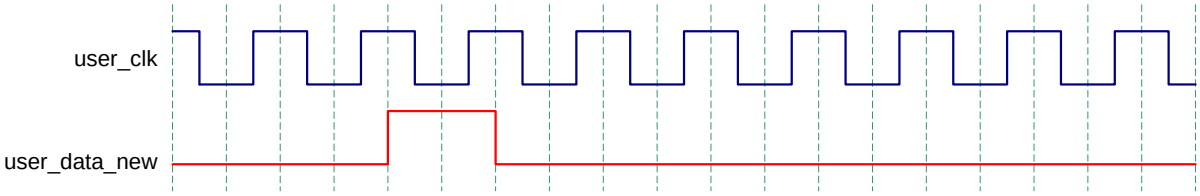


**Figure 18 – Reference Timing Diagram for a Strobe**

For **Figure 18**, the timing description is as follows:

1. USER_DATA_NEW is the STROBE. USER_DATA_NEW is one single pulse, the width of one USER_CLK.

## D.2 SERVICES

Not applicable

## D.3 SERVICE PRIMITIVES AND ATTRIBUTES

Not applicable

## D.4 ENTITY DEFINITIONS

### D.4.1 MHAL FPGA Transmit Node – Multi-Depth FIFO Entity Description

```
-------------------------------------------------------------------------------
-- MHAL FPGA Transmit Node - Multi-Depth
-------------------------------------------------------------------------------

entity MHAL_TX_MULTI is

  port(

    ---------------------------------------------------------------------------
    -- JTR Set INTERFACE
    ---------------------------------------------------------------------------

    -- <***To be filled in by platform developers***>

    ---------------------------------------------------------------------------
    -- USER INTERFACE
    ---------------------------------------------------------------------------

    -- Write Interface of FIFO

    -- FIFO Write Enable
    WRITE_EN          : in std_logic;

    -- Flag indicating last byte of message
    WRITE_EOM_MSB     : in std_logic;

    -- Flag indicating last byte of message
    WRITE_EOM_LSB     : in std_logic;

    -- Data to be written to FIFO
    WRITE_DATA        : in std_logic_vector(15 downto 0)

    -- Clock for Write Interface of FIFO from User
    USER_CLK          : in std_logic;

    -- Indicates full fifo *** optional ***
    FULL_FLAG         :out std_logic;

    -- Indicates almost full fifo *** optional ***
    ALMOST_FULL_FLAG         :out std_logic

  );

end entity TX_NODE;
```

## D.4.2 MHAL FPGA Receive Node – Multi-Depth FIFO Entity Description

```
-------------------------------------------------------------------------------
-- MHAL FPGA Receive Node - Multi-Depth FIFO
-------------------------------------------------------------------------------

entity MHAL_RX_MULTI is

  generic(

    -- MHAL node logical destination
    G_LOGICAL_DESTINATION    : natural := 0;

  );

  port(

    ---------------------------------------------------------------------------
    -- JTR Set INTERFACE
    ---------------------------------------------------------------------------

    -- <***To be filled in by platform developers***>

    ---------------------------------------------------------------------------
    -- USER INTERFACE
    ---------------------------------------------------------------------------

    -- Read Interface of FIFO

    -- FIFO Read Enable (Acknowledge)
    READ_EN          :in std_logic;

    -- Clock for Read Interface of FIFO from User
    USER_CLK         :in std_logic;

    -- Flag indicating message is ready to be read from FIFO
    READ_FLAG        :out std_logic;

    -- Flag indicating last byte of message
    READ_EOM_MSB     :out std_logic;

    -- Flag indicating last byte of message
    READ_EOM_LSB     :out std_logic;

    -- most recent word read from FIFO
    READ_DATA        :out std_logic_vector(15 downto 0)

  );

end entity RX_NODE;
```

## D.4.3 MHAL FPGA Receive Node – Single-Depth FIFO Entity Description

```
-------------------------------------------------------------------------------
-- MHAL FPGA Receive Node - Single-Depth FIFO
-------------------------------------------------------------------------------

entity MHAL_RX_SINGLE is

  generic(

    -- Width of MHAL data
    G_USER_WIDTH     :integer := 16;

    -- MHAL node logical destination
    G_LOGICAL_DESTINATION   : natural := 0

  );

  port(

    ---------------------------------------------------------------------------
    -- JTR Set INTERFACE
    ---------------------------------------------------------------------------

    -- <***To be filled in by platform developers***>

    ---------------------------------------------------------------------------
    -- USER INTERFACE
    ---------------------------------------------------------------------------

    -- Read Interface of FIFO

    -- Clock for Read Interface of FIFO from User
    USER_CLK         :in std_logic;

    USER_DATA_NEW    :out std_logic;

    USER_NEXT        :in std_logic;

    USER_DATA        :out std_logic_vector (G_USER_WIDTH - 1 downto 0)

    PACKET_SIZE      :out std_logic_vector(15 downto 0)

    -- *** optional ***
    EMPTY_FLAG       :out std_logic;

    -- *** optional ***
    FULL_FLAG        :out std_logic

  );

end entity RX_NODE;
```

## D.4.4 MHAL FPGA Receive Node – RAM Entity Description

```
-------------------------------------------------------------------------------
-- MHAL FPGA Receive Node - RAM
-------------------------------------------------------------------------------

entity MHAL_RX_RAM is

  generic(

    -- Width of MHAL data
    G_USER_WIDTH            :natural := 16;

    -- MHAL node logical destination
    G_LOGICAL_DESTINATION   :natural := 0

  );

  port(

    ---------------------------------------------------------------------------
    -- JTR Set INTERFACE
    ---------------------------------------------------------------------------

    -- <***To be filled in by platform developers***>

    ---------------------------------------------------------------------------
    -- USER INTERFACE
    ---------------------------------------------------------------------------

    USER_DATA_NEW    :out std_logic;

    USER_CLK         :in std_logic;

    USER_ADDR        :in std_logic_vector(G_ADDR_WIDTH - 1 downto 0)

    PACKET_SIZE      :out std_logic_vector(15 downto 0)

    USER_DATA        :out std_logic_vector(G_USER_WIDTH - 1 downto 0)

  );

end entity MHAL_RX_RAM;
```

## D.4.5 MHAL FPGA Receive Node – N-Word Register Entity Description

```
-------------------------------------------------------------------------------
-- MHAL FPGA Receive Node - N-Word Register
-------------------------------------------------------------------------------

entity MHAL_RX_REG is

  generic(

    -- Width of MHAL data
    G_USER_WIDTH            :natural := 16;

    -- MHAL node logical destination
    G_LOGICAL_DESTINATION   :natural := 0

  );

  port(

    ---------------------------------------------------------------------------
    -- JTR Set INTERFACE
    ---------------------------------------------------------------------------

    -- <***To be filled in by platform developers***>

    ---------------------------------------------------------------------------
    -- USER INTERFACE
    ---------------------------------------------------------------------------

    -- *** optional ***
    USER_DATA_NEW    :out std_logic;

    USER_CLK         :in  std_logic;

    USER_DATA        :out std_logic_vector (G_USER_WIDTH - 1 downto 0)

  );

end entity MHAL_RX_REG;
```

## D.4.6 MHAL FPGA Receive Node – Strobe Entity Description

```
-------------------------------------------------------------------------------
-- MHAL FPGA Receive Node - Strobe
-------------------------------------------------------------------------------

entity MHAL_RX_STRB is

  generic(

    -- MHAL node logical destination
    G_LOGICAL_DESTINATION   : natural := 0

  );

  port(

    ---------------------------------------------------------------------------
    -- JTR Set INTERFACE
    ---------------------------------------------------------------------------

    -- <***To be filled in by platform developers***>

    ---------------------------------------------------------------------------
    -- USER INTERFACE
    ---------------------------------------------------------------------------

    USER_DATA_NEW    :out std_logic;

    USER_CLK         :in std_logic

  );

end entity MHAL_RX_STRB;
```

## D.5 DATA TYPES AND EXCEPTIONS

None

# APPENDIX D.A – ABBREVIATIONS AND ACRONYMS

See section Appendix A.A.

# APPENDIX D.B – PERFORMANCE SPECIFICATION

Not applicable

# E. MHAL RF CHAIN COORDINATOR API EXTENSION

## E.1 INTRODUCTION

The *MHAL RF Chain Coordinator* API consists of a set of sink and source functions that provide for coordinated control of a JTRS Communications Channel's RF resources.

The *MHAL RF Chain Coordinator* API implementation for the JTR set may be deployed on any of the CEs.

### E.1.1 Overview

This document contains as follows:

a. Section E.1, *Introduction*, of this document contains the introductory material regarding the Overview.

b. Section E.2, *Services*

c. Section E.3, *Service Primitives and Attributes*, specifies the functions that are provided by the *MHAL RF Chain Coordinator*.

d. Section E.4, *Interface Definitions*

e. Section E.5, *Data Types and Exceptions*

f. Appendix E.A – *Abbreviations and Acronyms*

g. Appendix E.B – *Performance Specification*

h. Appendix E.C – *Multi-Command Message Example*

# E.2 SERVICES

The API for the *MHAL RF Chain Coordinator* takes the form of commands. As specified in A.2.1.3.1 MHAL Message Structure the MHAL Message payload contains a sequence of commands that consist of:

- Command Symbol followed by
- Command Parameter(s) (if applicable)

The Command Symbol values are given by symbolic reference similar to the LD and are predefined integer constants.

Parameters vary in size but are designed to be an integer multiple of 16 bits.

A message to the *MHAL RF Chain Coordinator* may contain one or more commands. No delimiters are required or allowed between commands. Each command in a multi-command message is processed in the order given.

## E.2.1 MHAL RF Chain Coordinator

This feature provides additional capabilities to manage the *MHAL RF Chain Coordinator* Data Sink. Support of this feature is through the functions defined in the following sections:

- E.3.1 *General* Sink Functions
- E.3.2 Supervisory Sink Functions
- E.3.3 General Source Functions

The *MHAL RF Chain Coordinator* is a Data Sink and is accessible from any CE via the Mhal_Comm.

The *MHAL RF Chain Coordinator* is associated with one or more Logical Destination (LD) symbolic references. Each LD can be independently set via the Command symbols.

The symbolic names for the integer constants used as the LDs by the MHAL Communications Function to route messages to the *MHAL RF Chain Coordinator* should begin with `RFCHAIN`.

## E.2.2 Sink Functions

The *RF Chain General* sink functions consist of the following commands in **Table 5**, which can be called by other components.

**Table 5 – MHAL RF Chain Coordinator General Sink Function Interface**

| Type* | Command Name (Sink) | Command Symbol | Parameter Format / Unit | Parameter Valid Range |
|---|---|---|---|---|
| config | RFC_DefModulationMode | DEFMODMODE | Multi-Value (see E.3.1.1) | |
| | RFC_ModulationMode | MODMODE | Int 16 / N/A | 0 – 7 |
| | RFC_RxAGCAttackTime** | TXAGCATTACKTIME | Int 16 / us | Not specified |
| | RFC_RxAGCDecayTime** | TXAGCDECAYTIME | Int 16 / us | Not specified |
| | RFC_TxALCAttackTime | TXALCATTACKTIME | Int 16 / us | Not specified |
| | RFC_TxALCDecayTime | TXALCDECAYTIME | Int 16 / us | Not specified |
| | RFC_TxEnvDecayTime | TXENVDECAYTIME | Int 16 / us | Not specified |
| | RFC_TxEnvRiseTime | TXENVRISETIME | Int 16 / us | Not specified |
| dynamic-D | RFC_ChannelFrequency | FREQ | Unsigned Int 32 / Hz | 1 - 4294967295 |
| | RFC_ChannelRxModeSet | RXMODE | None | N/A |
| | RFC_ChannelTxModeSet | TXMODE | None | N/A |
| | RFC_TxPower | TXPOWER | Int 16 / dBW | -60 – 60 |
| dynamic-I | RFC_ChannelTxKeyDisable | TXKEYOFF | None | N/A |
| | RFC_ChannelTxKeyEnable | TXKEYON | None | N/A |
| | RFC_ConnectTxBlock | CONNECTTXBLOCK | Int 16 / LD | N/A |
| | RFC_MasterExecuteNow | MEXECUTE | None | None |
| | RFC_ReceiverGainControl | RXGAIN | Int 16 / dBFS | Not specified |

* "Type" Legend:
config - Channel Configuration Item-command message based
dynamic-D - Delayed Execution-command message based- Execute Now Trigger
dynamic-I - Immediate Execution-command message based
** Optional

The *RF Chain Supervisory* sink functions consist of the following commands in **Table 6**, which can be called by other components.

**Table 6 – MHAL RF Chain Coordinator Supervisory Sink Function Interface**

| Type* | Command Name (Sink) | Command Symbol | Parameter Format / Unit | Parameter Valid Range |
|---|---|---|---|---|
| dynamic-I | RFC_ChannelStandbyModeSet | STANDBYMODE | None | N/A |
| | RFC_TxBusyStatusRequest | TXSTAT | Int 16 / LD | N/A |

| Type* | Command Name (Sink) | Command Symbol | Parameter Format / Unit | Parameter Valid Range |
|---|---|---|---|---|

* "Type" Legend:
dynamic-I - Immediate Execution-command message based

## E.2.3 Source Functions

The *RF Chain General* source functions consist of the following commands in **Table 7**, which are used by the RF Chain.

**Table 7 – MHAL RF Chain Coordinator General Source Function Interface**

| Type* | Command (Source) | Command Symbol | Parameter Format / Unit | Parameter Valid Range |
|---|---|---|---|---|
| rsp-LD_1 | RFC_TxBlocked | TXBLOCKED | None | N/A |
| rsp-LD_2 | RFC_TxBusyStatusResponse | TXSTATRSP | Int 16 / Status | N/A |

* "Type" Legend:
rsp-LD_# - Response to the command that containing LD#. The LD# is a sink for this response.

# E.3 SERVICE PRIMITIVES AND ATTRIBUTES

To enhance the readability of this API document and to avoid duplication of data, the type definitions of all structured types (i.e., structures, typedefs, exceptions, enumerations and unions) used by the Service Primitives and Attributes have been co-located in section E.5 Data Types and Exceptions. This cross-reference of types also includes any nested structures in the event of a structure of structures or an array of structures.

## E.3.1 *General* Sink Functions

### E.3.1.1 RFC_DefModulationMode Command

This command defines the initial setup and preset performance configuration for a particular modulation mode.

#### E.3.1.1.1 Command Structure

| <MHAL Header | | | > | <Payload | |
|---|---|---|---|---|---|
| IU Bit | LD | Length | | Command Symbol | Parameters |
| *x* 1 | **RFCHAIN** 15 | **17** 16 | | **DEFMODMODE** 16 | *ModMode Number* 16 |
| | | | | | *Envelope Rise Time* 16 |
| | | | | | *Envelope Fall Time* 16 |
| | | | | | *ALC Attack Gain* 16 |
| | | | | | *ALC Decay Gain* 16 |
| | | | | | *ALC Peak Limit* 8 > LSB |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.1.2 Parameters

| Parameter Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| Mod Mode Number | Modulation number. This value is used as the parameter to the MODMODE command. All subsequent fields are sent to the RF devices when this MODMODE number is selected. | Int 16 | N/A | 0 – 7 |
| Envelope Rise Time | The power amplifier transmit signal envelope rise time of the RF Chain Channel. | Int 16 | us | Not specified |
| Envelope Fall Time | The power amplifier transmit signal envelope decay time of the RF Chain Channel. | Int 16 | us | Not specified |
| ALC Attack Gain | The power amplifier ALC attack time of the RF Chain Channel. | Int 16 | us | Not specified |
| ALC Decay Gain | The power amplifier ALC decay time of the RF Chain Channel. | Int 16 | us | Not specified |
| ALC Peak Limit | The power amplifier ALC peak limit of the | Int 8 | dBm | Not |

| | | | | |
|---|---|---|---|---|
| | RF Chain Channel. | | | specified |

### E.3.1.1.3 Originator

Service User

## E.3.1.2 RFC_ModulationMode Command

This command sets the modulation mode of the RF Chain Channel to the value supplied.

The values that are sent to the RF devices are controlled by the waveform's definition of the MODMODE through the DEFMODMODE command.

This command allows for individual configuration of the modulation mode.

### E.3.1.2.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* $_{1}$ | **RFCHAIN** $_{15}$ | **8** $_{16}$ | **MODMODE** $_{16}$ | *Modulation Mode* $_{16}$ |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

### E.3.1.2.2 Parameters

| **Parameter Name** | **Description** | **Type** | **Units** | **Valid Range** |
|---|---|---|---|---|
| Modulation Mode | The same value as the parameter ModMode Number defined in the DEFMODMODE command. This causes this configuration of the RF devices to occur. | Int 16 | N/A | 0 – 7 |

### E.3.1.2.3 Originator

Service User

### E.3.1.3 RFC_RxAGCAttackTime Command *(optional)*

This command sets the power amplifier AGC attack time of the RF Chain Channel to the value supplied.

This command allows for individual configuration of the AGC attack time.

#### E.3.1.3.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* 1 | **RFCHAIN** 15 | **8** 16 | **RXAGCATTACKTIME** 16 | *AGC Attack Time* 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.3.2 Parameters

| Parameter Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| AGC Attack Time | A 16-bit integer representing the AGC attack time. | Int 16 | us | Not specified |

#### E.3.1.3.3 Originator

Service User

### E.3.1.4 RFC_RxAGCDecayTime Command *(optional)*

This command sets the power amplifier AGC decay time of the RF Chain Channel to the value supplied.

This command allows for individual configuration of the AGC decay time.

#### E.3.1.4.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* 1 | **RFCHAIN** 15 | **8** 16 | **RXAGCDECAYTIME** 16 | *AGC Decay Time* 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.4.2 Parameters

| **Parameter Name** | **Description** | **Type** | **Units** | **Valid Range** |
|---|---|---|---|---|
| AGC Decay Time | A 16-bit integer representing the AGC decay time. | Int16 | us | Not specified |

#### E.3.1.4.3 Originator

Service User

### E.3.1.5 RFC_TxALCAttackTime Command

This command sets the power amplifier ALC attack time of the RF Chain Channel to the value supplied.

This command allows for individual configuration of the ALC attack time.

#### E.3.1.5.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* 1 | **RFCHAIN** 15 | **8** 16 | **TXALCATTACKTIME** 16 | *ALC Attack Time* 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.5.2 Parameters

| Parameter Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| ALC Attack Time | A 16-bit integer representing the ALC attack time. | Int 16 | us | Not specified |

#### E.3.1.5.3 Originator

Service User

### E.3.1.6 RFC_TxALCDecayTime Command

This command sets the power amplifier ALC decay time of the RF Chain Channel to the value supplied.

This command allows for individual configuration of the ALC decay time.

#### E.3.1.6.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* 1 | **RFCHAIN** 15 | **8** 16 | **TXALCDECAYTIME** 16 | *ALC Decay Time* 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.6.2 Parameters

| Parameter Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| ALC Decay Time | A 16-bit integer representing the ALC decay time. | Int16 | us | Not specified |

#### E.3.1.6.3 Originator

Service User

### E.3.1.7 RFC_TxEnvDecayTime Command

This command sets the power amplifier transmit signal envelope decay time of the RF Chain Channel to the value supplied.

This command allows for individual configuration of the envelope decay time.

#### E.3.1.7.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* 1 | **RFCHAIN** 15 | **8** 16 | **TXENVDECAYTIME** 16 | *Envelope Decay Time* 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.7.2 Parameters

| Parameter Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| Envelope Decay Time | A 16-bit integer representing the envelope decay time. | Int 16 | us | Not specified |

#### E.3.1.7.3 Originator

Service User

### E.3.1.8 RFC_TxEnvRiseTime Command

This command sets the power amplifier transmit signal envelope rise time of the RF Chain Channel to the value supplied.

This command allows for individual configuration of the envelope rise time.

#### E.3.1.8.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* 1 | **RFCHAIN** 15 | **8** 16 | **TXENVRISETIME** 16 | *Envelope Rise Time* 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.8.2 Parameters

| Parameter Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| Envelope Rise Time | A 16-bit integer representing the Envelope rise time. | Int 16 | us | Not specified |

#### E.3.1.8.3 Originator

Service User

### E.3.1.9 RFC_ChannelFrequency Command

This command sets the Channel RF frequency to the value supplied.

#### E.3.1.9.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* 1 | **RFCHAIN** 15 | **10** 16 | **FREQ** 16 | *frequency* 32 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.9.2 Parameters

| **Parameter Logical Name** | **Description** | **Type** | **Units** | **Valid Range** |
|---|---|---|---|---|
| frequency | Value representing frequency. | Unsigned Int 32 | Hz | 0 - 4294967295 |

#### E.3.1.9.3 Originator

Service User

### E.3.1.10 RFC_ChannelRxModeSet Command

This command sets the RF mode to *RX*.

If the current mode is *TX or Standby*, this command toggles the setting to *RX*.
If the current mode is RX, this command has no effect.

#### E.3.1.10.1 Command Structure

| <MHAL Header > | | | <Payload > |
|---|---|---|---|
| IU Bit | LD | Length | Command Symbol |
| *x* 1 | **RFCHAIN** 15 | **6** 16 | **RXMODE** 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.10.2 Parameters

None

#### E.3.1.10.3 Originator

Service User

### E.3.1.11 RFC_ChannelTxModeSet Command

This command sets the RF mode to *TX*.

If the current mode is *RX or Standby*, this command toggles the setting to *TX*.

If the current mode is *TX*, this command has no effect.

#### E.3.1.11.1 Command Structure

| <MHAL Header > | | | <Payload > |
|---|---|---|---|
| IU Bit | LD | Length | Command Symbol |
| *x* 1 | **RFCHAIN** 15 | **6** 16 | **TXMODE** 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.11.2 Parameters

None

#### E.3.1.11.3 Originator

Service User

### E.3.1.12 RFC_TxPower Command

This command sets the transmitter output power to the value supplied.

#### E.3.1.12.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* 1 | **RFCHAIN** 15 | **8** 16 | **TXPOWER** 16 | *Tx Output Power* 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.12.2 Parameters

| **Parameter Name** | **Description** | **Type** | **Units** | **Valid Range** |
|---|---|---|---|---|
| Tx Output Power | Value that will be set as the transmitter output Power. | Int 16 | dBW<br>(i.e. dB relative to 1 Watt) | N/A |

#### E.3.1.12.3 Originator

Service User

### E.3.1.13 RFC_ChannelTxKeyDisable Command

This command disables the transmit key and keeps RF from being put out over the air. The same source that initiated the key should be used to terminate the transmit key.

#### E.3.1.13.1 Command Structure

| <MHAL Header > | | | <Payload > |
|---|---|---|---|
| IU Bit | LD | Length | Command Symbol |
| *x* 1 | **RFCHAIN** 15 | **6** 16 | **TXKEYOFF** 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.13.2 Parameters

None

#### E.3.1.13.3 Originator

Service User

### E.3.1.14 RFC_ChannelTxKeyEnable Command

This command enables the transmit key and allows RF to be put out over the air. The same source that initiated the key should be used to terminate the transmit key.

#### E.3.1.14.1 Command Structure

| <MHAL Header > | | | <Payload > |
|---|---|---|---|
| IU Bit | LD | Length | Command Symbol |
| *x* $_{1}$ | **RFCHAIN** $_{15}$ | **6** $_{16}$ | **TXKEYON** $_{16}$ |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.14.2 Parameters

None

#### E.3.1.14.3 Originator

Service User

### E.3.1.15 RFC_ConnectTxBlock Command

This command identifies a LD that is to receive a TX Mode Blocked (TXBLOCKED) response. This is used by all waveforms to be able to react to a blocked transmit asset.

#### E.3.1.15.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* $_{1}$ | **RFCHAIN** $_{15}$ | **8** $_{16}$ | **CONNECTTXBLOCK** $_{16}$ | *LD* $_{16}$ |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.15.2 Parameters

| Parameter Name | Description | Type | Units | Valid Range |
|---|---|---|---|---|
| LD | Requestor's logical destination that receives the TX Mode Blocked response. | Int 16 | LD | N/A |

#### E.3.1.15.3 Originator

Service User

### E.3.1.16 RFC_MasterExecuteNow Command

This command causes all of the previously issued deferred commands (i.e. type dynamic-D) to execute in the RF Chain. The RF Chain Coordinator shall synchronize all MEXECUTE messages.

#### E.3.1.16.1 Command Structure

| <MHAL Header > | | | <Payload > |
|---|---|---|---|
| IU Bit | LD | Length | Command Symbol |
| *x* 1 | **RFCHAIN** 15 | **6** 16 | **MEXECUTE** 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.16.2 Parameters

None

#### E.3.1.16.3 Originator

Service User

### E.3.1.17 RFC_ReceiverGainControl Command

This command sets the receiver gain to the value supplied. This command is to supplement not to override HW AGC.

#### E.3.1.17.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* $_{1}$ | **RFCHAIN** $_{15}$ | **8** $_{16}$ | **RXGAIN** $_{16}$ | *Receive Gain* $_{16}$ |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.1.17.2 Parameters

| **Parameter Logical Name** | **Description** | **Type** | **Units** | **Valid Range** |
|---|---|---|---|---|
| Receive Gain | A 16-bit integer value that will be set as receive gain. | Int 16 | dBFS<br>(i.e. dB relative to full scale output) | Not specified |

#### E.3.1.17.3 Originator

Service User

## E.3.2 Supervisory Sink Functions

### E.3.2.1 RFC_ChannelStandbyModeSet Command

This command sets the RF mode to *Standby*.

If the current mode is *RX or TX*, this command toggles the setting to *Standby*.

If the current mode is *Standby*, this command has no effect.

#### E.3.2.1.1 Command Structure

| <MHAL Header > | | | <Payload > |
|---|---|---|---|
| IU Bit | LD | Length | Command Symbol |
| *x* 1 | **RFCHAIN** 15 | **6** 16 | **STANDBYMODE** 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.2.1.2 Parameters

None

#### E.3.2.1.3 Originator

Service User

### E.3.2.2 RFC_TxBusyStatusRequest Command

This command determines if the transmit resources are available for the use by the waveform.

#### E.3.2.2.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* 1 | **RFCHAIN** 15 | **8** 16 | **TXSTAT** 16 | *LD* 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits

#### E.3.2.2.2 Parameters

| **Parameter Name** | **Description** | **Type** | **Units** | **Valid Range** |
|---|---|---|---|---|
| LD | Logical destination where the transmit status response is reported. | Int 16 | LD | N/A |

#### E.3.2.2.3 Originator

Service User

# E.3.3 General Source Functions

## E.3.3.1 RFC_TxBlocked Command

This command is used to report that Transmit Mode (TXMODE) cannot be entered or is lost.

The *MHAL RF Chain Coordinator* provides a TX Mode Blocked (TXBLOCKED) response subsequent to the MEXECUTE command related to TXMODE to indicate that some resource in the RF Chain is blocking the transmit. When this is sent, the RF Chain is in RXMODE. The waveform reestablishes full operation of the RXMODE characteristics. This occurs while in TXMODE if the transmit asset is lost.

### E.3.3.1.1 Command Structure

| <MHAL Header > | | | <Payload > |
|---|---|---|---|
| IU Bit | LD | Length | Command Symbol |
| *x* $_{1}$ | *resp_LD* $_{15}$ | **6** $_{16}$ | **TXBLOCKED** $_{16}$ |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits
Note3: Where resp_LD is the LD that was provided by the Connect Transmit Mode Blocked (CONNECTTXBLOCK) command.

### E.3.3.1.2 Parameters

None

### E.3.3.1.3 Originator

Service Provider

### E.3.3.2 RFC_TxBusyStatusResponse Command

This command is used to report whether resources are available for use by the waveform.

#### E.3.3.2.1 Command Structure

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* 1 | *resp_LD* 15 | **8** 16 | **TXSTATRSP** 16 | *Status* 16 |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits
Note3: Where resp_LD is the LD that was provided by the TXSTAT command.

#### E.3.3.2.2 Parameters

| **Parameter Name** | **Description** | **Type** | **Units** | **Valid Range** |
|---|---|---|---|---|
| Status | Availability of resources for use by waveform | Int 16 | 0 = Available<br>1 = Busy | 0, 1 |

#### E.3.3.2.3 Originator

Service Provider

## E.4 INTERFACE DEFINITIONS

None

## E.5 DATA TYPES AND EXCEPTIONS

Not applicable

# APPENDIX E.A – ABBREVIATIONS AND ACRONYMS

See section Appendix A.A.

# APPENDIX E.B – PERFORMANCE SPECIFICATION

Not applicable

# APPENDIX E.C – MULTI-COMMAND MESSAGE EXAMPLE

This example shows the MHAL message layout of three commands sent via one Mhal_Comm call to the RF Chain Coordinator

**Table 8 – Example Multi-Message Command Structure**

| <MHAL Header > | | | <Payload > | |
|---|---|---|---|---|
| IU Bit | LD | Length | Command Symbol | Parameter |
| *x* $_{1}$ | **RFCHAIN** $_{15}$ | **16** $_{16}$ | **RXMODE** $_{16}$ | *N/A*** $_{0}$ |
| | | | **MODMODE** $_{16}$ | *Modulation Mode* $_{16}$ |
| | | | **FREQ** $_{16}$ | *frequency* $_{32}$ |

Note1: Where x = 0 or 1 (See A.2.1.3.1.1 MHAL Message IU Bit)
Note2: Subscript indicates field width in bits
** This field is noted for table format consistency; the RFC_ChannelRxModeSet Command has no parameters.

In **Table 8** the multi-command message includes the RFC_ChannelRxModeSet Command,

RFC_ModulationMode Command, and RFC_ChannelFrequency Command.