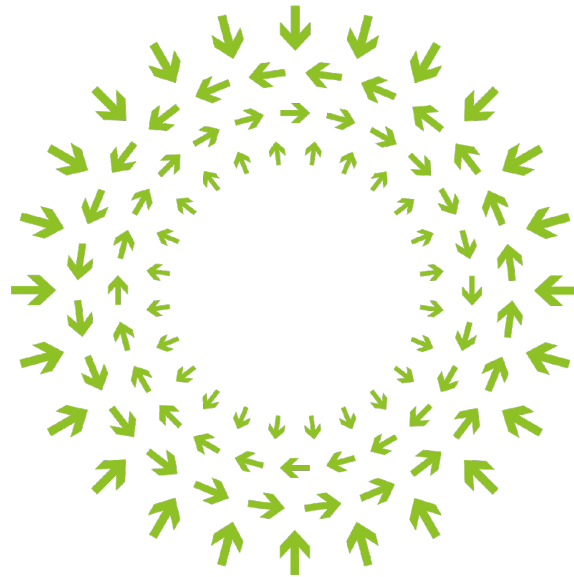




OPEN
Compute Project

Networking Group - Optics Reliability Subgroup

Transparent Remote Optical Transceiver Messaging with ePPS Design Specification

Revision 0.9

Authors:

Credo: Chris Collins

Version History

Date	Version #	Author	Description
8 OCT 2026	0.9	Chris Collins	Initial Release

Table of Contents

Version History	2
Open Web Foundation (OWF) CLA	6
Acknowledgements	7
Copyright / Trademark	7
1 Compliance with OCP Tenets	8
1.1 Openness	8
1.2 Efficiency	8
1.3 Impact	8
1.4 Scale	8
1.5 Sustainability	8
2 Overview	9
3 Scope	10
4 Conventions	10
5 Introduction	11
6 Overview	11
7 Implementation	13
7.1 Messaging Format and Communication Rate	13
7.2 Message Types	14
7.2.1 IDLE	14
7.2.2 ePPS TIME SYNC	14
7.2.3 ROUND TRIP (RTT) CALIBRATION	14
7.2.4 PREAMBLE	14
7.2.5 MESSAGE PAYLOAD	14
7.3 General Operation Description	14
7.4 High Level Transmit State Machine Operation	15
7.5 Transmit Side implementation (MSG_TX)	16
7.5.1 MSG_TX FSM	16
7.6 High level Receiver Lock State Machine Operation	18
7.6.1 MSG_RX Lock Detect State Machine Description	19
7.7 High level Receive Message State Machine Operation	21

- 7.7.1 MSG_RX FSM..... 21
- 8 Implementation for In-band ePPS Transport..... 23
 - 8.1 ePPS Clock Detection, Encoding and Transmission 23
 - 8.2 ePPS Receive and External Clock Regeneration 24
 - 8.3 Calibrating the Optical Link Round Trip Latency (RTT) 27
- 9 Other Considerations 28
 - 9.1 Protocol Considerations.....28
 - 9.2 Implementation Considerations.....28
- 10 Message Formats 29
 - 10.1 IDLE PACKET 29
 - 10.1.1 Message Receiver Status 29
 - 10.2 PREAMBLE PACKET 29
 - 10.3 MESSAGE PAYLOAD PACKET 30
 - 10.4 ePPS TIME SYNC PACKET 30
 - 10.5 ROUND TRIP LATENCY CALIBRATION PACKET.....30
- Appendix A – Implementation Checklist 31

Table of Figures

Figure 1: Conceptual Drawing of an In-band Message Channel _____	11
Figure 2: In-band Transparent Message Channel with ePPS High Precision Sync Clock _____	11
Figure 3: Conceptual Block Diagram of a Transparent In-band Messaging Link _____	12
Figure 4: Message Block Insertion into the Data Path _____	13
Figure 5: General Message Interface Operation _____	15
Figure 6: MSG_TX FSM _____	17
Figure 7: Lock Detect state Machine Inside the MSG_RX Block _____	20
Figure 8: MSG_RX FSM _____	22
Figure 9: Block Diagram of ePPS (Sync Clock) Detection & Transmission _____	23
Figure 10: ePPS Transmit FSM _____	24
Figure 11: Block Diagram of ePPS (Sync Clock) Extraction and Regeneration _____	25
Figure 12: Message Channel Timing showing ePPS Insertion (MSG_TX) and Extraction (MSG_RX) _____	26
Figure 13: IDLE Packet _____	29
Figure 14: 256-bit Preamble Packet _____	29
Figure 15: 256-bit Message Packet _____	30
Figure 16: 256-bit ePPS Packet Format _____	30
Figure 17: 256-bit Optical Link Latency Calibration (RTT) Packet Format _____	30

THE UPDATED DEFAULT CONTRIBUTOR LICENSE AGREEMENT (CLA) IS [OWFa 0.9](#). PLEASE VERIFY THE CORRECT CLA/FSA IS USED AND EXECUTED FOR THIS CONTRIBUTION.

Open Web Foundation (OWF) CLA

Contributions to this Specification are made under the terms and conditions set forth in **Modified Open Web Foundation Agreement 0.9 (OWFa 0.9)**. (As of October 16, 2024) (“Contribution License”) by:

Credo Semiconductor

Usage of this Specification is governed by the terms and conditions set forth in **Modified OWFa 0.9 Final Specification Agreement (FSA)** (As of October 16, 2024) (“**Specification License**”).

You can review the applicable Specification License(s) referenced above by the contributors to this Specification on the OCP website at <https://www.opencompute.org/contributions/templates-agreements>.

For actual executed copies of either agreement, please contact OCP directly.

Notes:

The above license does not apply to the Appendix or Appendices. The information in the Appendix or Appendices is for reference only and non-normative in nature.

NOTWITHSTANDING THE FOREGOING LICENSES, THIS SPECIFICATION IS PROVIDED BY OCP "AS IS" AND OCP EXPRESSLY DISCLAIMS ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, OR TITLE, RELATED TO THE SPECIFICATION. NOTICE IS HEREBY GIVEN, THAT OTHER RIGHTS NOT GRANTED AS SET FORTH ABOVE, INCLUDING WITHOUT LIMITATION, RIGHTS OF THIRD PARTIES WHO DID NOT EXECUTE THE ABOVE LICENSES, MAY BE IMPLICATED BY THE IMPLEMENTATION OF OR COMPLIANCE WITH THIS SPECIFICATION. OCP IS NOT RESPONSIBLE FOR IDENTIFYING RIGHTS FOR WHICH A LICENSE MAY BE REQUIRED IN ORDER TO IMPLEMENT THIS SPECIFICATION. THE ENTIRE RISK AS TO IMPLEMENTING OR OTHERWISE USING THE SPECIFICATION IS ASSUMED BY YOU. IN NO EVENT WILL OCP BE LIABLE TO YOU FOR ANY MONETARY DAMAGES WITH RESPECT TO ANY CLAIMS RELATED TO, OR ARISING OUT OF YOUR USE OF THIS SPECIFICATION, INCLUDING BUT NOT LIMITED TO ANY LIABILITY FOR LOST PROFITS

OR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR PUNITIVE DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND EVEN IF OCP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Acknowledgements

The Contributors of this Specification would like to acknowledge the following for their feedback:

Ashwin Upadhyia – Credo
Louis Lin – Credo
Yicong Wang - Credo
Evan Lin – Credo
JJ Wang – Credo

Copyright / Trademark

This specification references and utilizes information from **OIF-CMIS-05.3** and future looking **OIF-CMIS-05.4** and as such is subject to OIF terms.

1 Compliance with OCP Tenets

1.1 Openness

By contributing this specification to the OCP Networking Group – Optics Reliability Subgroup, it allows all members to enjoy the benefits of increased reliability and fewer link flaps in large scale AI networks.

1.2 Efficiency

The goal of contributing the Transparent Remote Optical Transceiver Messaging with ePPS Design Specification is to further enable the Enhanced Telemetry Optics Software Specification by driving commonality in various optical DSP implementations. This will enable more efficient qualifications and faster optical deployment by directly reducing the OPEX required by eliminating the need to rewrite host software for each different transceiver. Furthermore, by increasing the reliability and uptime of large-scale GPU clusters, operational efficiencies are enhanced with fewer system restarts, faster LLM training and less manual system debug.

1.3 Impact

Optical transceivers are a critical component in deploying AI fabrics. By helping to enhance their reliability, system operators benefit from fewer network failures and improved operational performance. When bare-metal server access is provided to end users, network operators lose visibility into optical endpoints leaving them blind in the event of a link problem. With remote management capability, compliant optics provide visibility right to the transceiver in the bare metal server. As GPU clusters grow to hundreds of thousands, or even millions of bare-metal compute instances, providing visibility into server link quality across an entire fabric becomes a fundamental necessity.

1.4 Scale

Optical transceivers are being deployed at unprecedented rates in RDMA fabrics for AI. The shipment of 400G, 800G and 1.6T optical transceivers is expected to exceed 70 million units per year in 2027. This specification specifically targets these, and future higher rate, optical transceivers.

1.5 Sustainability

This contribution is specifically focused on making large-scale AI deployments more efficient by decreasing downtime due to optical networking issues. By making the Transparent Remote Transceiver Messaging Interface open source, the goal is to ensure that the entire ecosystem of transceiver vendors and network switch vendors can meet the industry demand for equipment.

2 Overview

This design specification defines a transparent, in-band remote messaging channel between two DSPs at opposite ends of an optical PMD link. The channel is created by slightly overclocking the line-side SerDes relative to the host side so that one extra 256-bit messaging slot can be inserted after every N 256-bit data blocks without disrupting normal data traffic. This extra slot provides the hardware transport used for remote DSP messaging, enhanced pulse per second (ePPS) time synchronization, and round-trip latency calibration.

The protocol defines five packet types that occupy the inserted 256-bit messaging slots: IDLE, PREAMBLE, MESSAGE PAYLOAD, ePPS TIME SYNC, and ROUND-TRIP CALIBRATION. Each packet type has a defined bit pattern, payload layout, and receiver processing behavior specified in this specification.

For interoperability, both link partners shall use the same programmable insertion interval N , the same INTER_MSG_IDLE_COUNT value, the same packet type definitions, and the same IDLE/PREAMBLE detection rules. The recommended defaults are $N = 30,000$ and INTER_MSG_IDLE_COUNT = 1. In steady state, IDLE packets maintain frame alignment and carry receiver lock status; a PREAMBLE packet indicates that five MESSAGE PAYLOAD packets will follow, separated by the configured number of IDLE packets.

The ePPS TIME SYNC packet transports timing information used to regenerate an ePPS clock at the far end of the link. ROUND TRIP CALIBRATION packets measure link propagation delay so the implementation can compensate for the latency of ePPS clock transmission.

Transmit and receive state machines shall establish both local and remote IDLE lock before any message exchange. The receiver searches for the IDLE pattern, confirms consecutive local IDLE matches, verifies remote lock status, and then applies hysteresis so isolated bit errors do not immediately cause lock loss. The detailed state definitions, thresholds, and loss-of-lock conditions are defined in this specification.

In steady state, IDLE packets are sent continuously every N cycles. Any message transfer is user initiated by filling the message buffer and setting the TX_MSG_READY bit.

The same in-band channel may distribute ePPS timing. The module acting as the clock source measures the sync-clock period and inserts a time-sync management packet containing the countdown value from the most recent sync edge to the next available management slot. The module acting as the target loads this value, counts down to regenerate the sync edge, and uses a configured or transmitted pulse width to recreate the output pulse. Optical link time-of-flight affects absolute link latency but not the recovered clock period.

For latency calibration, either DSP may initiate round trip time measurement using free-running counters and specific time stamping of when each packet is sent and received.

3 Scope

The purpose of this design specification is to define the hardware implementation necessary for an optical DSP to provide a transparent remote messaging interface between two ends of an optical link. The requirements herein are applicable to 400G, 800G, 1.6T and 3.2T compliant transceivers and are intended to be independent of the module PMD. The intent of this specification is that it will allow any DSP vendor to offer a compatible interface which is required by the Enhanced Telemetry Optics Software Specification.

The focus of this design specification is the hardware functionality that permits a remote messaging interface over a fiber link. The exact method to implement these functions is left to the DSP vendor. The implementation of the functionality is NOT within the scope of this specification.

4 Conventions

This section lists the standard conventions used in this specification. Normative language is mandatory.

- Shall = requirement
Requirements are mandatory, must be done.
Including the word “not” (as in shall not) specifies a mandatory prohibition, meaning the requirement explicitly forbids the action.
- Should = recommendation
Recommendations are optional yet indicate a preferred direction if implemented
Including the word “not” (as in should not) indicates a strong recommendation against performing an action
- May = allowable
Indicates flexibility of choice (option) with no implied preference.

5 Introduction

This specification describes a transparent communication scheme between the two DSPs at either end of a fiber optic link. A transparent, in-band messaging system is required to avoid extra fibers, lasers and receivers. The messages exchanged between the two DSPs shall be sent over the existing high-speed channel thereby eliminating the need for a separate fiber optic channel. This solution is compatible with both electrical and optical PMDs.

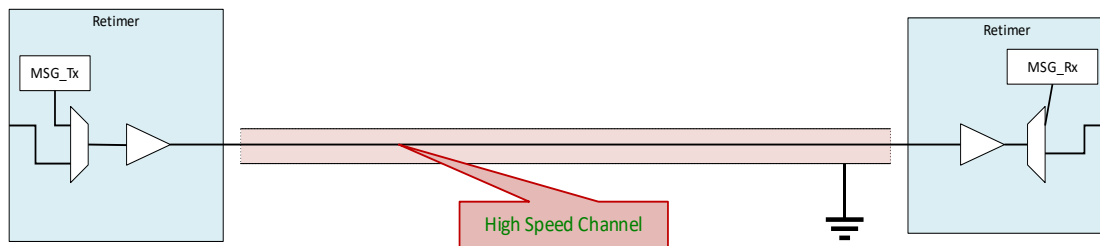


Figure 1: Conceptual Drawing of an In-band Message Channel

In addition, it is desirable for multiple network devices to be synchronized by distributing a precise clock (ex: ePPS clock) from a source device to one or more target devices. The preferred implementation is to send precision timing information for the start and end of each clock cycle over the existing high-speed channel using the transparent in-band management interface described above. The transparent management channel operates as a protocol agnostic communication link between the two ends of a fiber link while not disrupting the network traffic flowing through. The time-of-flight latency of the ePPS clock from the input of the source network device to the output of the target network device can be calibrated and stored as described in this specification.

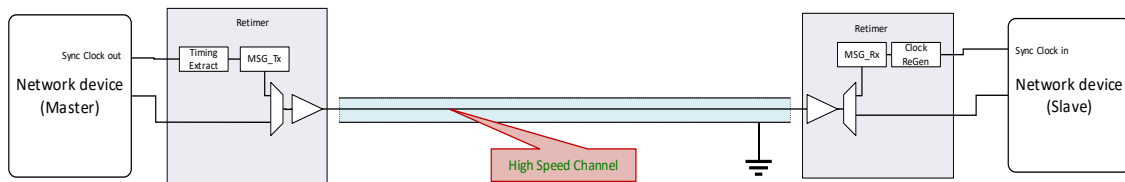


Figure 2: In-band Transparent Message Channel with ePPS High Precision Sync Clock

6 Overview

The extra bandwidth required to create a transparent in-band communication channel is achieved by slightly increasing the clock frequency on the line side of the DSP (Optical channel side).

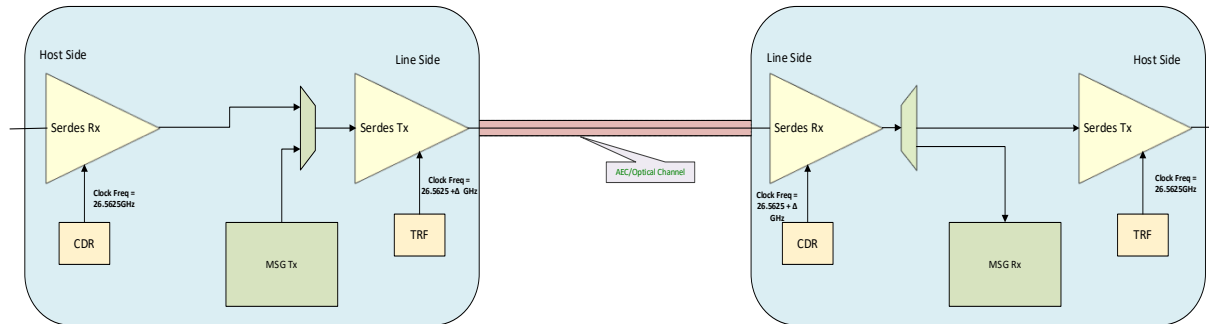


Figure 3: Conceptual Block Diagram of a Transparent In-band Messaging Link

The design shown in Figure 3 assumes the line side Tx Serdes can be configured to a clock frequency that is Δ GHz higher than its host side Rx Serdes. Similarly, on the remote end of the channel, the line side Rx Serdes is assumed to be capable of being clocked higher by a Δ GHz than its host side Tx Serdes. By using this overclocking, extra bits can be inserted into the data path which forms the basis of the messaging structure. With this design the host sides of the 2 DSP retimers continue to operate at the nominal Ethernet frequency to support error free traffic at the established line rate. Concurrently, the line (optical) sides operate at a slightly higher bitrate to allow for the insertion of a transparent messaging channel.

7 Implementation

7.1 Messaging Format and Communication Rate

The basic message frame size is 1200-bits. Each 1200-bit message is broken up into five (5), 240-bit blocks and are sent periodically to the link partner. The link partner is responsible for extracting the five 240-bit blocks from the data stream and rebuilding the complete 1200-bit message. The hardware implementation does not define the communication protocol. It is designed to provide a reliable method for the firmware to send and receive 1200-bit messages across a live optical link with a bandwidth of roughly 3 Mbit/s even in the presence of a background bit error rate on the data path.

To achieve this messaging bandwidth, the line bitrate shall be increased by ΔPPM . To calculate the ΔPPM value, we define a value N . N defines the duty cycle for inserting extra messaging bits into the data stream. More specifically, N is the number of 256-bit data blocks that are transmitted before a single extra 256-bit block is inserted into the data path that contains a single 240-bit message block. The remaining 16-bits of the 256-bit block are vendor defined. See Figure 4.

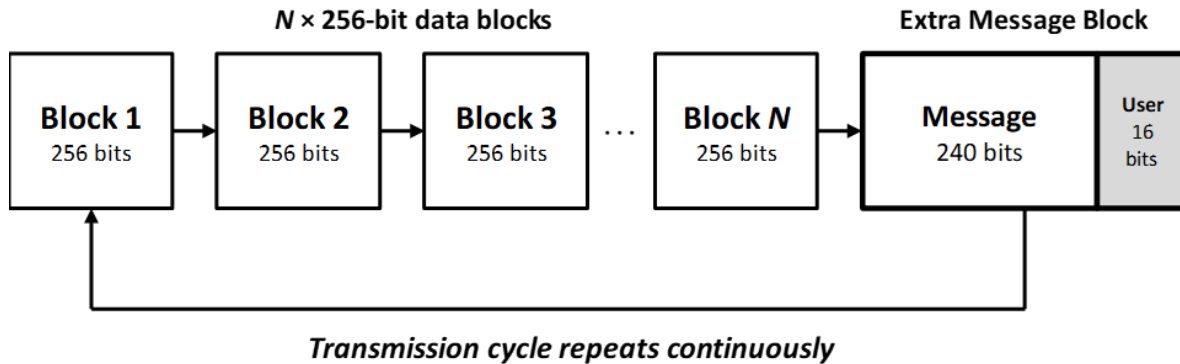


Figure 4: Message Block Insertion into the Data Path

In this implementation N shall be a register programmable value with a default of 30,000. Using the value of N , the ΔPPM value can be calculated as shown below.

$$\Delta\text{PPM} = \left(\frac{1}{N}\right) * 1e^6, \quad \text{where } N = 30,000$$

$$\Delta\text{PPM} = 33.33$$

$$\Delta\text{GHz} = 7.083 \text{ Mbit/s}$$

The message block sent every N cycles can be one of the five (5) message types listed in section 7.2.

7.2 Message Types

The implementation defines five different packet types that are used to implement the hardware state machine. Each of the messages uses the same 256-bit time slot but have different formats for receiver identification and processing. The five packet types are:

1. IDLE
2. ePPS TIME SYNC
3. ROUND TRIP CALIBRATION
4. PREAMBLE
5. MESSAGE PAYLOAD

7.2.1 IDLE

The IDLE packet format is shown in 10.1.

7.2.2 ePPS TIME SYNC

The ePPS TIME SYNC packet format is shown in 10.4.

7.2.3 ROUND TRIP (RTT) CALIBRATION

The ROUND-TRIP CALIBRATION packet format is shown in 10.5.

7.2.4 PREAMBLE

The PREAMBLE packet format is shown in 10.2.

7.2.5 MESSAGE PAYLOAD

The MESSAGE PAYLOAD packet is used to transmit the five (5) 240-bit blocks that compromise the full 1200-bit message as described in section 7.1. The 16 unused uppermost bits from the complete 256-bit payload are vendor defined. The packet format is shown in 10.3.

7.3 General Operation Description

The general operation of the transparent messaging interface is described in Figure 5. In the steady state, when no message is pending, the *IDLE* block is sent continuously after every *N* blocks of data stream traffic. This allows the receive side of the link to maintain the correct bit position of the messaging blocks. Once a message is ready to be sent, one *PREAMBLE* block is sent followed by the five *MESSAGE PAYLOAD* blocks required to transmit the entire 1200-bit message payload. A variable called *INTER_MSG_IDLE_COUNT* is used to define the number of *IDLE* frames inserted between each consecutive *MESSAGE PAYLOAD* frame. The default value of *INTER_MSG_IDLE_COUNT* = 1 but this value shall be register programmable.

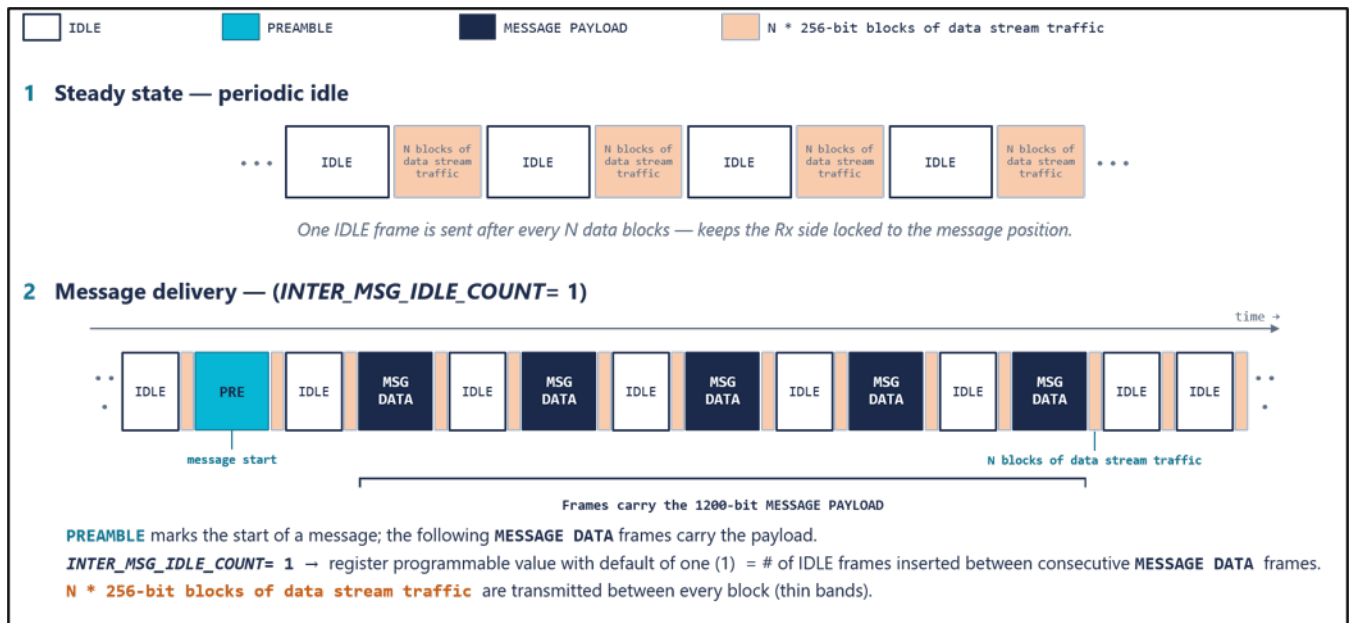


Figure 5: General Message Interface Operation

7.4 High Level Transmit State Machine Operation

By default, the Tx messaging state machine stays in the **INIT** state until it is enabled. When enabled, it transitions to the **TX_LOCK** state. In the **TX_LOCK** state, IDLE messages with the status **LOCAL_IDLE_UNLOCK** are transmitted continuously while simultaneously monitoring the received data for incoming IDLE messages from the remote transmitter. Once IDLE messages are detected and the receiver locks onto a continuous stream of incoming IDLE messages it declares **LOCAL_IDLE_LOCK** and transitions to the **REM_WAIT** state.

While in the **REM_WAIT** state, IDLE messages continue to be sent but status is changed to **LOCAL_IDLE_LOCK**. This updated IDLE message indicates to the remote receiver that its link partner has locked onto the messaging frames. At the same time during the **REM_WAIT** state, the local receiver monitors the incoming messages searching for **REMOTE_IDLE_LOCK** flag inside the IDLE messages. Once **REMOTE_IDLE_LOCK** is detected, this triggers the transition to the **IDLE** state.

In the **IDLE** state, both local and remote messaging interfaces remain locked to the message frames and continue to send IDLE messages until the **TX_MSG_READY = 1** is asserted by firmware indicating that a new 1200-bit message frame has been placed in the transmit message buffer. Once **TX_MSG_READY = 1** is detected, the **PREAMBLE** message is sent and the state transitions to **TXMSG_IDLE**. The **PREAMBLE** message indicates to the remote receiver that a message packet will follow.

After entering the **TXMSG_IDLE** state, IDLE frames are sent, and the IDLE frames number is based on a special counter indicating the **TX_IDLE_CNTR** value which will increment to

INTER_MSG_IDLE_COUNT during the **TXMSG_IDLE** state. Following these *IDLE* frames, the state transitions to **TX_MSG** and the first 240-bit payload from the message buffer is sent. Following this message block, the state transitions back to **TXMSG_IDLE** and some *IDLE* frames are sent with the TX_IDLE_CNTR value incremented by one (1). Then the state transitions back to **TX_MSG** and the next 240-bit block of the message payload is sent over the interface. This sequence of *IDLE* messages followed by 256-bit message packet (of which 240-bit are message payload) continues until all five (5) of the 240-bit message blocks have been successfully sent. At this point, the state transitions back to **IDLE** and waits for the next TX_MSG_READY=1 trigger.

7.5 Transmit Side implementation (MSG_TX)

The design implements message word insertion logic. To allow the insertion of an extra 256-bit word, the line side clock is clocked at a higher frequency from the host side receiver as noted in section 7.1.

7.5.1 MSG_TX FSM

The MSG_TX FSM state machine provides controls for the different states of the MSG_TX as described in Section 7.3 and shown in Figure 6. All state transitions other than to and from INIT are made at the **N** count.

In the **TX_LOCK** state, the MSG_TX block sends IDLE_Packet+UNLOCK_STATUS every **N** cycles. The MSG_TX waits until the local Rx has locked on to the IDLE_Packet and inserts IDLE_Packet+LOCK_STATUS on its transmitter while transitioning to **REM_WAIT** state.

In the **REM_WAIT** state, the State machine waits for the local Rx to set the REMOTE_IDLE_LOCK before transitioning to the **IDLE** state. During this state it continues to send IDLE_Packet+LOCK_STATUS if the LOCAL_IDLE_LOCK is set.

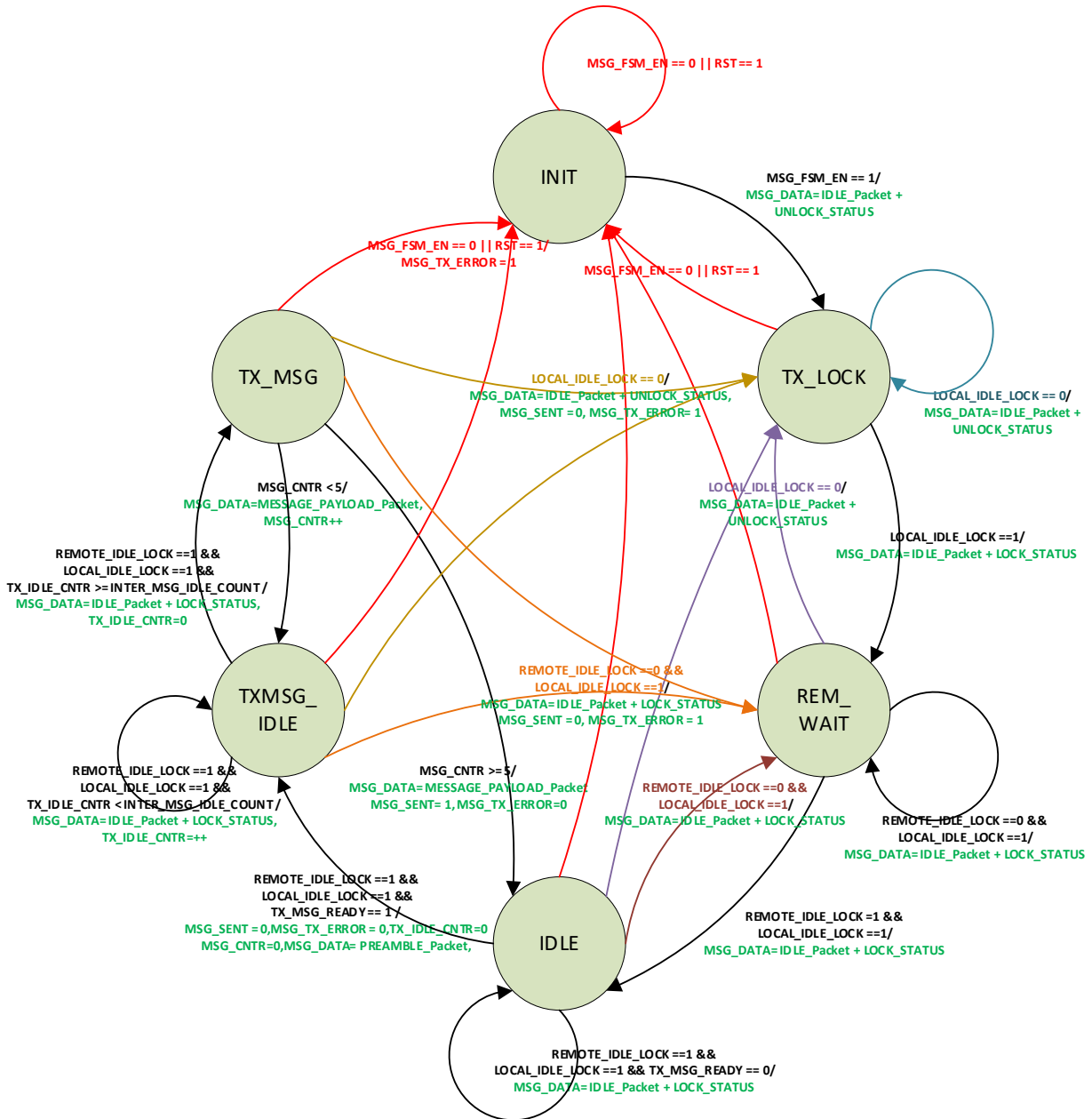


Figure 6: MSG_TX FSM

In the **IDLE** state, the MSG_TX sends the IDLE_Packet+LOCK_STATUS. The FW fills the 1200-bit TxMSG Buffer with the message to be transmitted and then sets TX_MSG_READY. On detecting the TX_MSG_READY==1, the MSG_TX sends MSG_PREAMBLE as the MSG_DATA at the next N^{th} cycle and transitions to the **TXMSG_IDLE** state.

In the **TXMSG_IDLE** state the TX_MSG FSM inserts IDLE_Packet+LOCK_STATUS so that the remote retimer can continue to stay locked. The number of IDLE_Packet+LOCK_STATUS MSG_DATA sent in this state is set by the INTER_MSG_IDLE_COUNT value. When the IDLES are sent the state transitions to **TX_MSG**.

In the **TX_MSG** state, the logic inserts one 240-bit message payload from the TxMSG Buffer and transitions back to the **TXMSG_IDLE** state. The transition from **TXMSG_IDLE** to **TX_MSG** and back continues until all the message payload from the TxMSG buffer is sent out. At this point, the logic sets the MSG_SENT bit and transitions back to the **IDLE** state. To transmit the next 1200-bit message from TXMSG buffer, Firmware shall set TX_MSG_READY = 1 after back to the **IDLE** state.

If the logic detects that either the LOCAL_IDLE_LOCK or REMOTE_IDLE_LOCK signals are de-asserted in the **IDLE** and **TXMSG_IDLE** states, the MSG_TX FSM transitions to the **TX_LOCK** state waits until locks are restored. If the TX_MSG FSM was in the middle of transferring a message packet, it sets the MSG_TX_ERROR bit.

Figure 5 visualizes the data stream at the line side transmit before and after message insertion. As shown in the figure, the MSG_DATA appears on every $(1 + \text{INTER_MSG_IDLE_COUNT}) * N$ cycles.

7.6 High level Receiver Lock State Machine Operation

The MSG_RX module implements 256-bit message extraction logic. To allow the extraction of the 256-bit message extraction from the Rx data stream, the extracted receive clock is at a higher frequency of Δ GHz. Each 240-bit message block word is captured from the Rx data stream and saved in the 1200-bit Rx MSG buffer.

The receiver has separate Rx Lock and Rx Message state machines. This ensures that the messages are only received with the receiver locked properly to the incoming message stream.

By default, the Rx Lock messaging state machine stays in the **INIT** state until it is enabled by a register bit and then transitions to the **SEEK** state. In the **SEEK** state, the message receive block searches through the incoming bit streams in sliding blocks of 256-bits searching for the **IDLE** message pattern. Once detected, the state transitions to **LOCK**.

In the **LOCK** state, the receiver continues to search in the same 256-bit location in the incoming data stream for additional **IDLE** message. If at least MIN_LOCAL_IDLE==16 consecutive **IDLE** messages are detected in the same data stream time slot, then LOCAL_IDLE_LOCK is asserted and the state transitions to **REM_LOCK**. If MIN_LOCAL_IDLE_LOCK consecutive **IDLE** messages are not detected, the state transitions back to **SEEK**.

Once in the **REM_LOCK** state, the receiver monitoring and counting the incoming **IDLE** messages waiting to detect the remote lock field asserted in the **IDLE** message. Once the field is detected, the state transitions to **REM_LOCK2**.

In **REM_LOCK2**, the receiver counts the consecutive number of received **IDLE** messages with the remote lock field asserted. Once the count equals the MIN_REM_LOCKS==16, REMOTE_IDLE_LOCK=1 is asserted and the state transitions to **LK_IDLE**. If the consecutive **IDLE** messages with remote lock field asserted do not reach MIN_REM_LOCKS, the state transitions back to **REM_LOCK**.

In the **LK_IDLE** state both the receiver and transmitter are locked to the *IDLE* messages. The receiver continuously searches for the *PREAMBLE* message which indicates that message will be following. Once the *PREAMBLE* message is detected, the state transitions to **LK_MSG_IDLE**.

In the **LK_MSG_IDLE** state, the receiver expects to receive `INTER_MSG_IDLE_COUNT` of *IDLE* messages. After receiving these *IDLE* messages, the state transitions to **LK_MSG**.

In the **LK_MSG** state, the receiver receives the first 256-bit *MESSAGE packet* (which contains 240-Payload of the 1200-bit message frame) and increments the `MSG_CNT` and transitions back to the **LK_MSG_IDLE** state. The receiver detects another `INTER_MSG_IDLE_COUNT==1` of *IDLE* messages, transitions to the **LK_MSG** state again and receives the next 256-bit message packet. This alternating *IDLE* message followed by 256-bit *MESSAGE Packet* continues until all five (5) 256-bit *MESSAGE Packets* are received. At this point 1200-bit received message buffer has the complete message frame. The state machine transitions back to the **LK_IDLE** state and waits for the next *PREAMBLE* message to be received.

7.6.1 MSG_RX Lock Detect State Machine Description

Figure 7 shows the Lock Detect state Machine inside the `MSG_RX` block. All transitions except the ones to `INIT` and `SEEK` are at the **N** count. During `SEEK`, the lock logic tries to match the *IDLE_Packet* in any **256**-bit sequence.

The FSM logic first searches for the location of the *IDLE_Packet* over a possible 256 locations in the `SEEK` state. Each time it fails to find the *IDLE_Packet* over **N** consecutive cycles, it slips to the next bit location. Once the *IDLE_Packet* is located, the **N** counter is reset to mark the location of the message block in the data stream.

To establish `LOCAL_IDLE_LOCK`, the logic looks for *IDLE_Packet* matches for the next `MIN_LOCAL_LOCKS` consecutive **N** cycle. Once `LOCAL_IDLE_LOCK` is established it looks at bits[63:60] of the *IDLE_Packet* for the `LOCK_STATUS` from the remote `MSG` block. In the `REM_LOCK2` state, the lock detect logic matches the `LOCK_STATUS` from the remote `MSG` for the next `MIN_REM_LOCKS` **N** cycles before declaring `REMOTE_IDLE_LOCK`.

Figure 7: Lock Detect state Machine Inside the MSG RX Block

Once both LOCAL_IDLE_LOCK and REMOTE_IDLE_LOCK are achieved, the logic applies hysteresis to the criteria to declare loss of IDLE_LOCK. First, it allows a 1-bit error in the IDLE_Packet and LOCK_STATUS matches. Then, over the most recent 32 collected IDLE slots, it declares loss of local IDLE_LOCK or remote IDLE_LOCK when fewer than 12 IDLE_Packet matches or fewer than 12 LOCK_STATUS matches are observed, respectively.

When the MSG_RX FSM detects a MSG_PREAMBLE, the Lock Detect FSM moves to LK_MSG_IDLE state. In this state it looks for INTER_MSG_IDLE_COUNT of IDLE_Packets before jumping to the LK_MSG state. When in LK_MSG_IDLE and LK_MSG states, the lock detect ignores the LOCK_STATUS from the remote MSG. This is done so that the entire message

packet sent by the remote is captured safely. Also, only the `IDLE_Packet`¹ matches between each message packet is counted towards the `LOCAL_IDLE_LOCK` history. The message packets and preamble are not counted as `IDLE_Packet` mismatches.

7.7 High level Receive Message State Machine Operation

By default, the Rx Lock messaging state machine stays in the **INIT** state until it is enabled by a register bit and then transitions to the **LK_WAIT** state. In the **LK_WAIT** state, the message receiver waits for the `IDLE_LOCK=1` to be asserted which (from the Rx Lock state machine) indicates that both `LOCAL_IDLE_LOCK=1` and `REMOTE_IDLE_LOCK=1`. Once detected, the state transitions to **IDLE**.

In the **IDLE** state, the receiver searches for the *PREAMBLE* message. Once a *PREAMBLE* message is detected, the receiver advances to the **MSG_IDLE** state.

In the **MSG_IDLE** state, the state machine ignores all the N^{th} cycles until the `IDLE_CNTR` (controlled by Lock Detect FSM) reaches the `INTER_MSG_IDLE_COUNT` value. At this point, the state transitions to **MSGEXTR**.

The next message packet shall contain the first 256-bits of MESSAGE Packet, of which 240-bits valid payload will be captured and stored in the RxMSG Buffer. It transitions back to the **MSG_IDLE** state. This loop, from **MSG_IDLE** to **MSGEXTR** and back, continues until all five (5) message payloads of the *MESSAGE Frame* are captured. Following this, the state transitions back to **IDLE** state while asserting the `RX_MSG_READY` signal. On reading a logic high on `RX_MSG_READY`, the firmware should move the contents of the RxMSG Buffer. After the buffer has been copied over, the firmware de-asserts the `RX_MSG_READY` signal. This indicates to the state machine to capture the next message packet when it detects a *PREAMBLE*.

7.7.1 MSG_RX FSM

Figure 8 shows the state machine in MSG_RX. All state transitions except the ones to and from INIT are at the N count.

As in the Lock Detect state machine, the MSG_RX FSM also is enabled by the `MSG_FSM_EN` setting by the firmware. At the **LK_WAIT** state, the FSM waits until both the local and remote idle locks are asserted by the Lock Detect logic.

¹ ePPS Packets and RTT Cal Packets contain the `LOCK_STATUS`. For these packets the `IDLE_Packet` match is forced, but the lock status is recorded in `REMOTE_IDLE_LOCK` history.

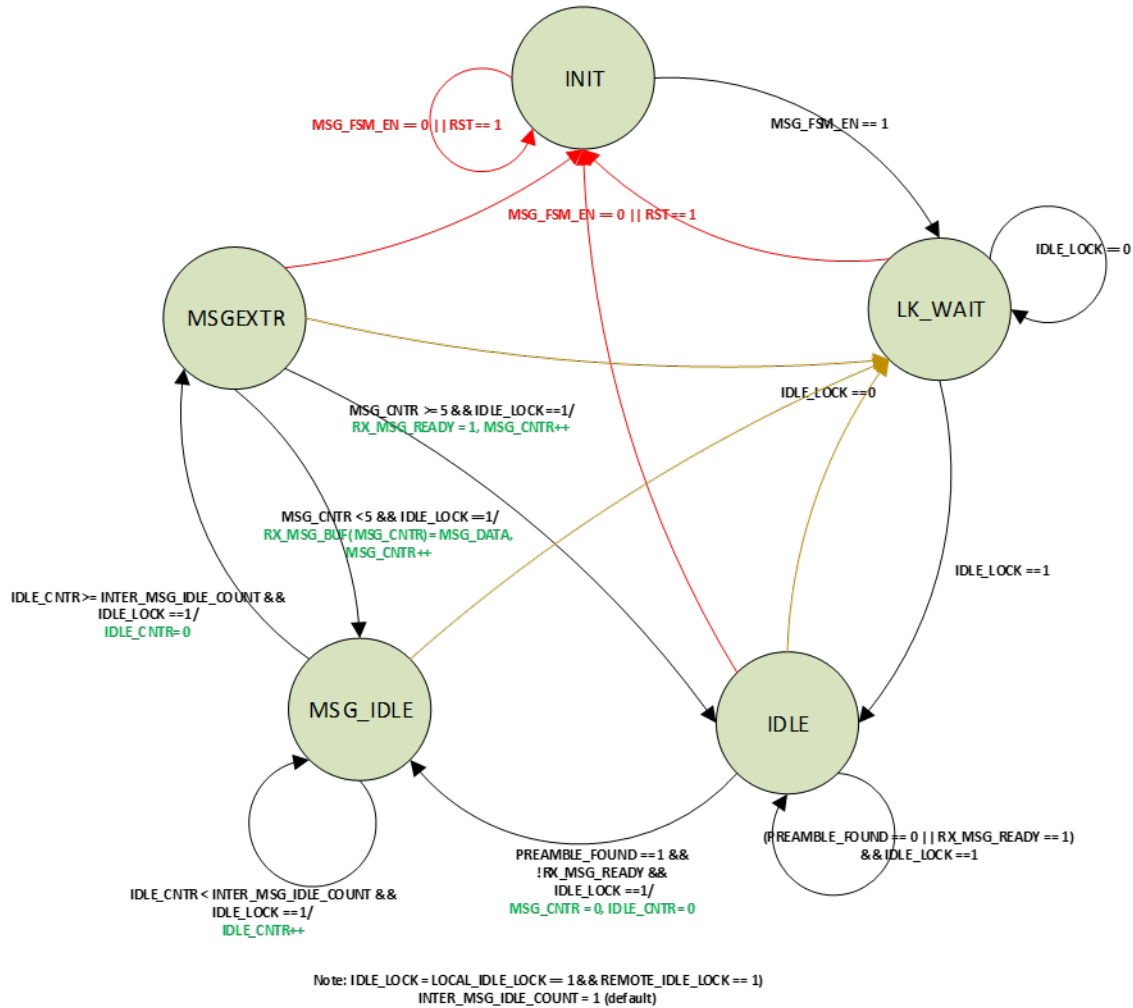


Figure 8: MSG_RX FSM

In the IDLE state, the state machine waits until an MSG_PREAMBLE message match is found. The matching logic allows for 1-bit error tolerance. If the RX_MSG_READY message is asserted when the MSG_PREAMBLE is detected, the state machine transitions to the MSG_IDLE state.

In the MSG_IDLE state, the state machine ignores all the **N** cycles until, the IDLE_CNTR (controlled by Lock Detect FSM) reaches the INTER_MSG_IDLE_COUNT value. At this, the FSM transitions to MSGEXTR state. At the next **N** cycle the message block shall contain the message payload, which is captured and stored in the RxMSG Buffer. It transitions back to the MSG_IDLE state. This loop, from MSG_IDLE to MSGEXTR and back, continues until all 5 message payloads of the message packet are captured. Following this, the FSM transitions back to the IDLE state while asserting the RX_MSG_READY signal. On reading logic high on RX_MSG_READY, the implementor shall latch the contents of the RxMSG Buffer and move the entire message to an accessible register location (for further decode and action). After the buffer has been copied over, the RX_MSG_READY signal shall be deasserted. This allows the FSM to capture the next message packet when it detects a MSG_PREAMBLE.

8 Implementation for In-band ePPS Transport

8.1 ePPS Clock Detection, Encoding and Transmission

The ePPS implementation (also referred to as 1PPS in places) involves a source (clock source) and a target (clock destination) side of the link. Figure 9 shows a block diagram of the Sync clock² edge detection and transmission logic inside the DSP located on the source side of the optical link. On the Rising edge of the clock, a Counter1 (counts Sync clock period using a high-speed clock) starts counting from 0. At the next rising edge of the Sync clock, the value of Counter1 is loaded onto Counter2 (Counts down the time elapsed from Sync Clock edge).

The messaging channel is used to insert an ePPS TIME SYNC at regular intervals. These ePPS TIME SYNC packets are asynchronous to the Sync clock and so the time between the Sync clock edge and the next management packet can vary. Counter2 is basically counting down the elapsed time from the Sync clock edge to the next available ePPS TIME SYNC management packet. At the next available packet, the Counter2 value is loaded onto the packet payload as shown in section 10.4. See the ePPS transmit state machine in Figure 10. The management packet with the Counter2 value in it is appended with a header to distinguish it from other *IDLE* packets.

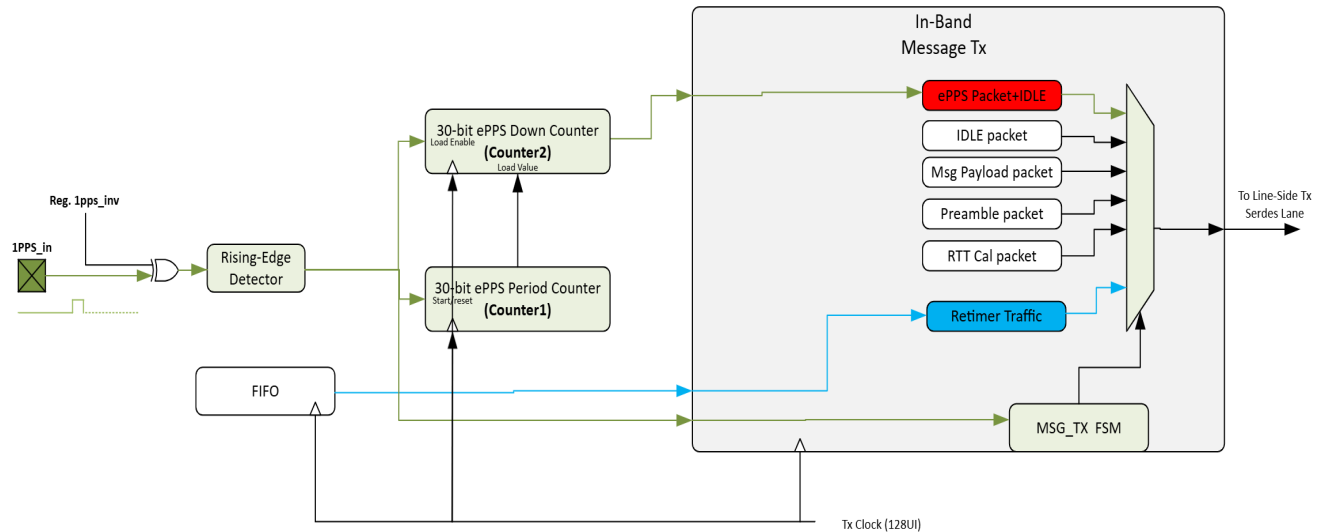


Figure 9: Block Diagram of ePPS (Sync Clock) Detection & Transmission

² In the implementation description, the Sync clock from the source network device is sometimes referred to as 1PPS signal or 1PPS_in

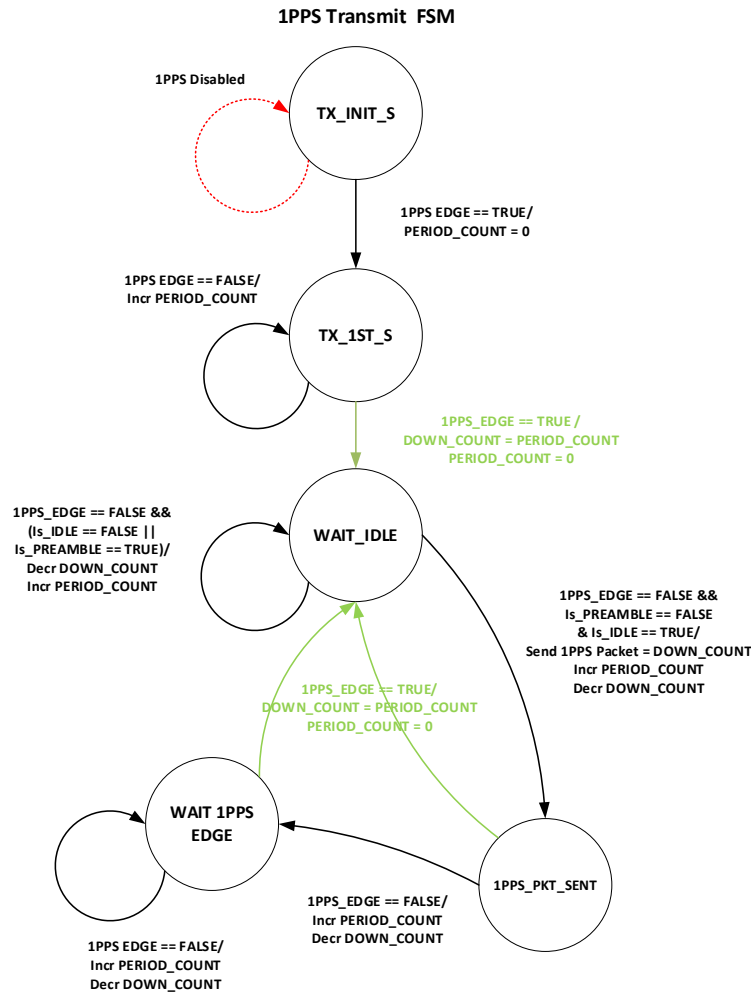


Figure 10: ePPS Transmit FSM

8.2 ePPS Receive and External Clock Regeneration

At the other end of the optical link, (target side) the receiver inside the DSP extracts the ePPS TIME SYNC packets as shown in Figure 11. The Counter2 Value from the ePPS TIME SYNC packet is loaded into Counter3 (counts the remainder of the Sync clock period) and then a countdown to 0 is initiated. At 0, the pulse generator rising edge is triggered which sends a rising edge on the physical ePPS clock output pin of the DSP. At 0, the pulse-width counter also starts. When the pulse-width counter reaches the set pulse-width value, it triggers a falling edge on the ePPS clock output pin of the DSP. The pulse-width can either be preset by the system or sent

over the management channel to reconstruct the Sync clock³. The output of the pulse generator (DSP output pin) reproduces the sync clock with the same period as the original ePPS clock input from the source side. The pulse generator may use the “Period Range” bits[33:32] of the incoming ePPS TIME SYNC packet to determine the duration of the pulse. The Table 1 shows the conversion from period range value to pulse-width.

Period Range [33:32]	ePPS Frequency Range	Pulse Width
0	1 kHz – 99.01 Hz	~300μs
1	99.01 Hz – 12.376 Hz	~2.5ms
2	12.376 Hz – 1.5477 Hz	~20.2ms
3	1.5477 Hz – 1 Hz	~161.5ms

Table 1: Period-Range to Pulse-width conversion

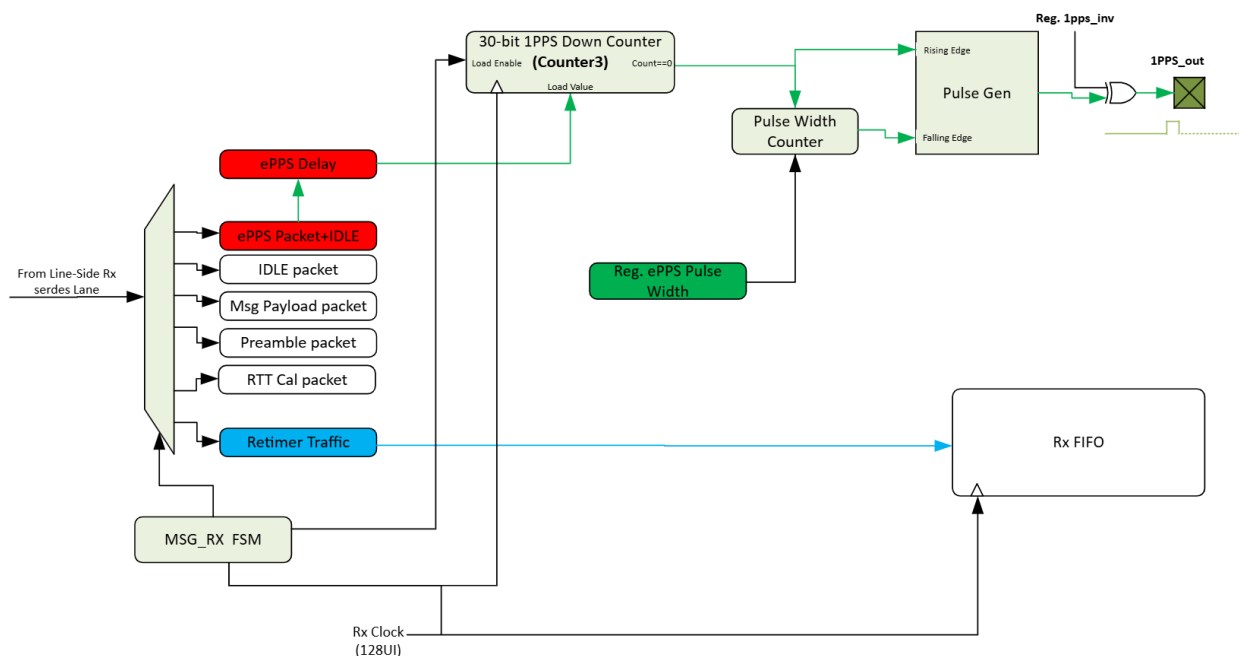


Figure 11: Block Diagram of ePPS (Sync Clock) Extraction and Regeneration

In Figure 12, the first rising edge of the ePPS input clock (1PPS_in) triggers Counter1 to count to D_p (the sync clock period) until the rising edge P_2 . Counter2 then counts down from D_p to DC_2 at

³ In the implementation description, the reconstructed Sync clock going to the target network device is sometimes referred to as 1PPS out

which point the next ePPS TIME SYNC packet captures and sends this value over the messaging channel.

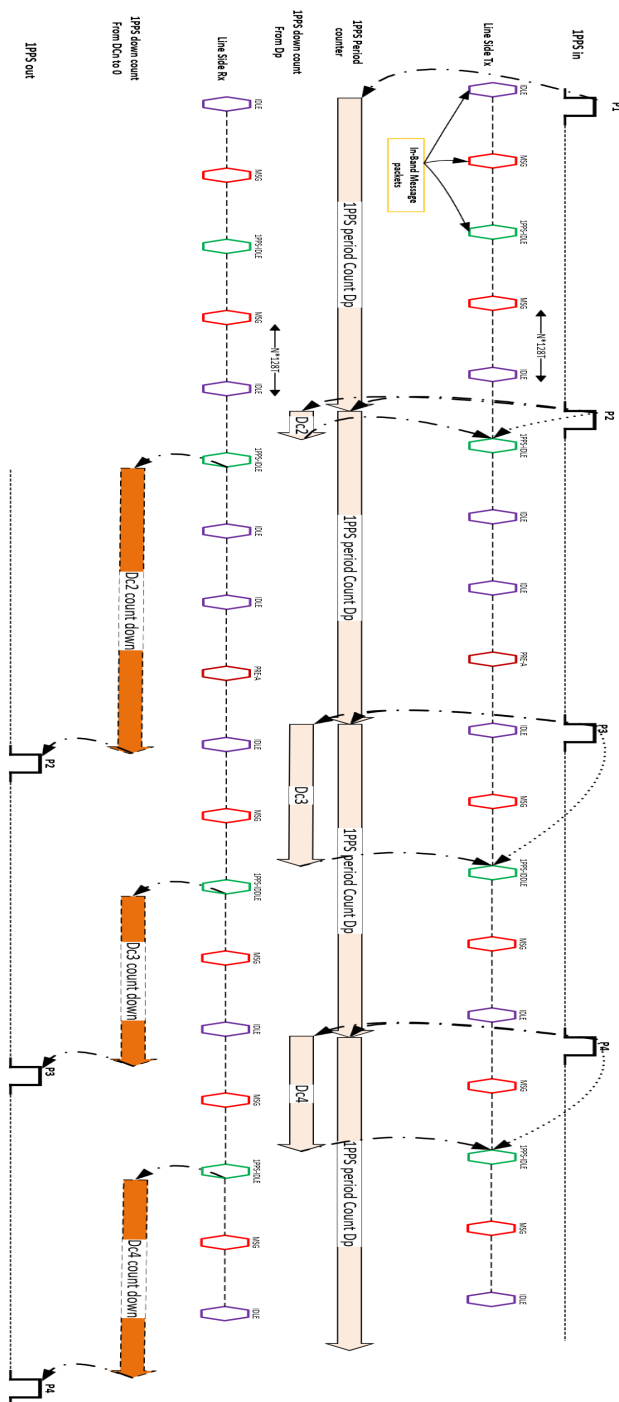


Figure 12: Message Channel Timing showing ePPS Insertion (MSG_TX) and Extraction (MSG_RX)

On the Receive (target) side DSP, the ePPS TIME SYNC packet with count DC2 arrives after time T_f (flight time over the optical link). Counter3 counts down from DC2 to 0 and triggers a rising edge on the Sync clock out (1PPS_out). Since the DC2 value is relative to the timing of the management packet, the flight time T_f does not affect the period of the ePPS clock. The flight time contributes to the latency of the clock edge.

8.3 Calibrating the Optical Link Round Trip Latency (RTT)

Latency of the ePPS clock inside the DSP can be calculated through simulations. However, the latency across the fiber medium can vary based on its length and shall be calibrated on startup.

To calculate this, the DSPs on both sides of the link maintain a free running counter. The calibration can be initiated by either DSP. The initiating DSP will be the source for the purposes of latency calibration. The source DSP records the time T_0 when it sends out the ROUND TRIP LATENCY CALIBRATION packet over its management channel in place of the next IDLE message (like the ePPS TIME SYNC operation). The DSP on the remote side (target) records the time T_1 on its free running counter when it receives the ROUND TRIP LATENCY CALIBRATION packet. The target DSP then sends an acknowledgement ROUND TRIP LATENCY CALIBRATION packet back to the source and records the time T_2 . The source records the time, T_3 on its free running counter again when it receives the acknowledgement packet. A calibration ID is defined and maintained for each complete round trip of a ROUND TRIP LATENCY CALIBRATION packet from source to target and back to source. See Section 10.5 for the ID field in the ROUND-TRIP LATENCY CALIBRATION packet. If multiple iterations of the calibration are performed, the ID shall change for each iteration.

The flight time or latency through the optical link is calculated by firmware as:

Time of flight latency : $T_f = ((T_3 - T_0) - (T_2 - T_1)) / 2$

9 Other Considerations

9.1 Protocol Considerations

- Each MESSAGE PAYLOAD packet shall use only the lower 240-bits of the 256-bit time slot. The most significant 16-bits of the packet is a vendor specific usage field.
- For the MESSAGE PAYLOAD to be sent and captured in a reliable way, a programmable CRC field and packet numbering scheme is defined in the firmware layer.
- Firmware will implement an ACK/NACK protocol between the 2 DSPs so that corrupted messages will be retransmitted.
- If the LOCAL_IDLE_LOCK == 0 when RX_MSG_READY == 1, then firmware shall discard the last message and send a NACK.

9.2 Implementation Considerations

- The IDLE packet sent after the local Rx loses lock, shall indicate the status as LOCAL UNLOCKED.

10 Message Formats

10.1 IDLE PACKET

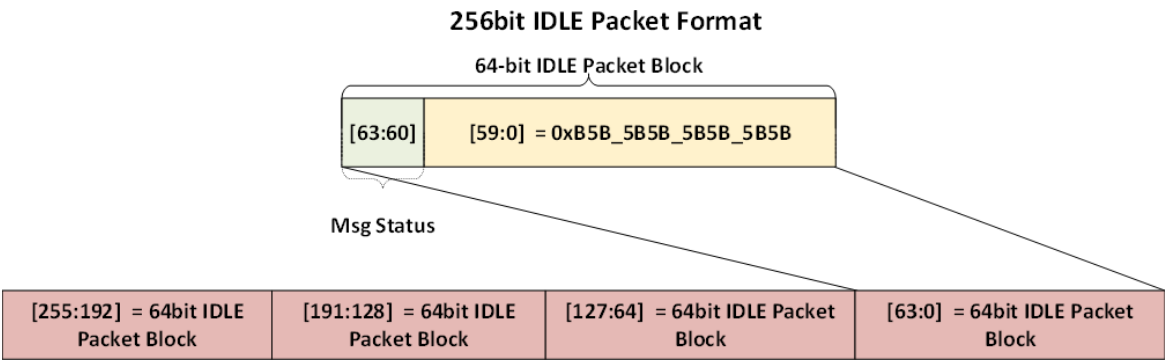


Figure 13: IDLE Packet

10.1.1 Message Receiver Status

A 4-bit field shown as Msg Status in Figure 13 carries the status of the local receiver. If the local receiver is locked to the incoming message stream, the Msg Status in each transmitted IDLE packet shall be set to 0xC. If the local receiver is NOT locked to the incoming message stream, the Msg Status in each transmitted IDLE packet shall be set to 0x3. The 4-bit Msg Status shall indicate either LOCKED or UNLOCKED.

10.2 PREAMBLE PACKET

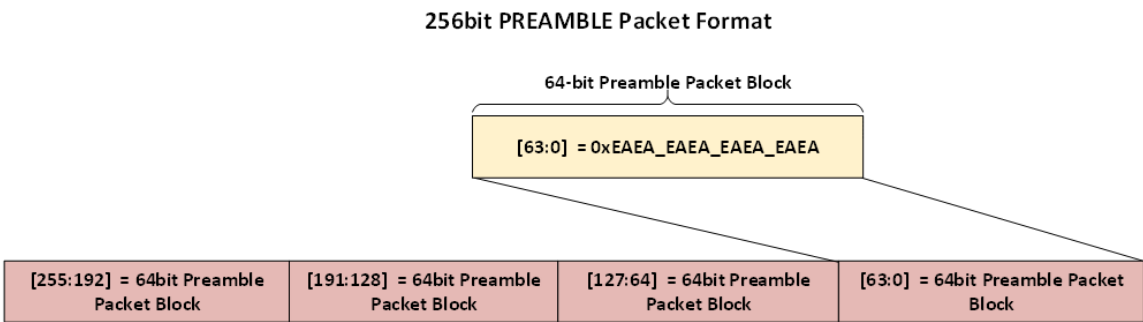


Figure 14: 256-bit Preamble Packet

10.3 MESSAGE PAYLOAD PACKET

256bit Message Packet Format

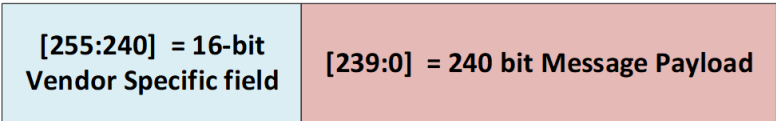


Figure 15: 256-bit Message Packet

10.4 ePPS TIME SYNC PACKET

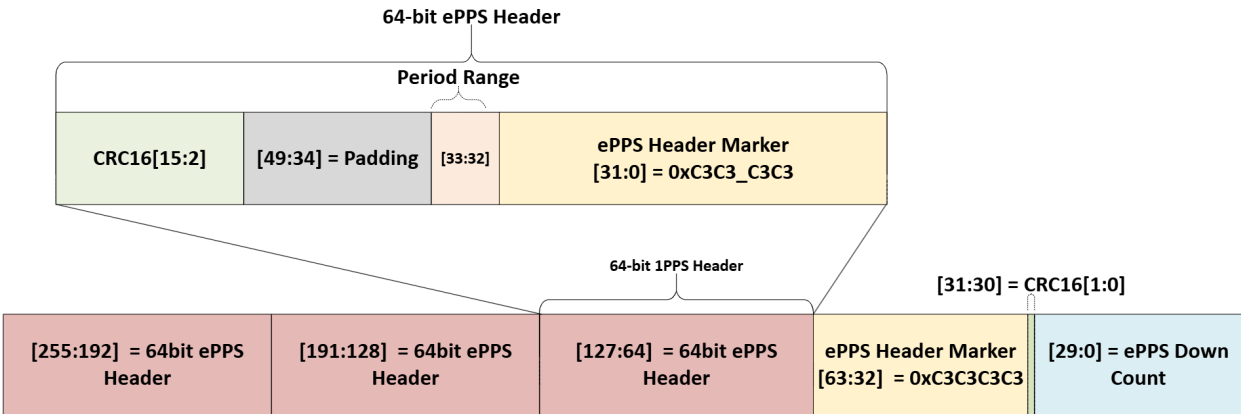


Figure 16: 256-bit ePPS Packet Format

10.5 ROUND TRIP LATENCY CALIBRATION PACKET

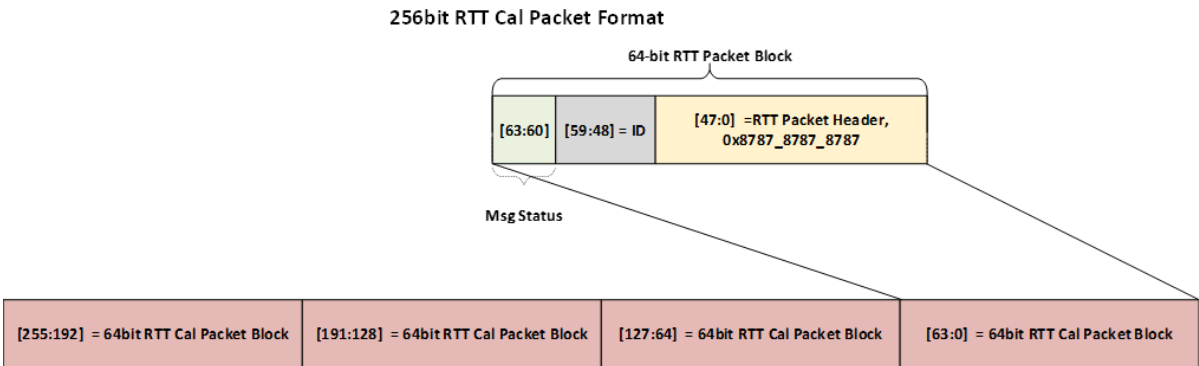


Figure 17: 256-bit Optical Link Latency Calibration (RTT) Packet Format

Appendix A – Implementation Checklist

Implementation checklist: match N , Δ PPM calculation, packet formats, IDLE/PREAMBLE patterns, LOCK/UNLOCK status values, INTER_MSG_IDLE_COUNT, lock thresholds, one-bit pattern tolerance, Tx/Rx buffer handshakes, ACK/NACK behavior, ePPS packet interpretation, RTT timestamp handling, and behavior after local or remote lock loss.