Contribution Notes (not to be included in the draft):
In this contribution text in red font is information to the TF participants and should be retained until Draft 1.0 at a minimum. Text in blue font is for information during review of this submission and need not be copied to the draft. Green text is used to highlight links. Figure in the draft or copied from motioned material shown as "postage stamps" (Frame file provided for figure drawn in native frame format).

1. Multi-Point Reconciliation Sublayer (MPRS) for 100G-EPON

1.4. Definitions (numbering per P802.3cj D2.1)

1.4.314a. MPRS channel: one of a number of defined paths along which data passes in a multi-point RS (see Clause 143).

143. Multi-Point Reconciliation Sublayer (MPRS) for 100G-EPON

143.1 Overview

This clause describes the Multi-Point Reconciliation Sublayer (MPRS) used with 100G-EPON point-to-multipoint (P2MP) networks. These are passive optical multipoint networks (PONs) that connect multiple data terminals to a single shared fiber. The architecture is asymmetric, based on a tree and branch topology utilizing passive optical splitters.

This clause extends Clause 106 to enable multiple MACs to interface with multiple Physical Layers. The number of MACs supported is limited only by the implementation. It is acceptable for only one MAC to be connected to this MPRS. **Error! Reference source not found.** shows the relationship between this MPRS and the ISO/IEC OSI reference model. Prior EPON specifications enabled multiple MACs to interface to a single PHY. This concept is expanded in this clause to allow multiple MACs to interface with up to four PHYs requiring up to four 25 Gigabit Media Independent Interfaces (25GMIIs).

The MPRS adapts the bit-serial protocols of the MAC to the parallel format of the PCS service interface. Though the 25GMII is an optional logical interface between the MAC sublayers and the Physical Layers, it is used extensively in this standard as a basis for specification. The 100G-EPON Physical Coding Sublayers (PCS) are specified to the 25GMII, so if not implemented, a conforming implementation behaves functionally as if the MPRS and 25GMIIs were implemented.

The 25GMII interfaces have the following characteristics:

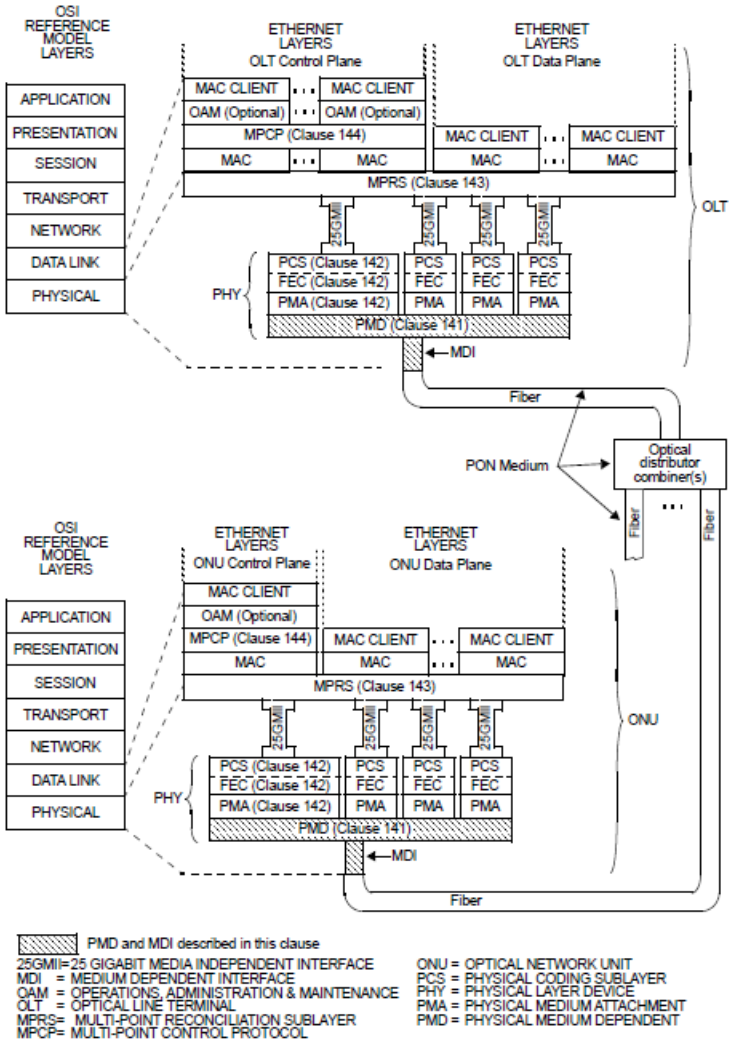
a) They are capable of supporting 25 Gb/s operation.

b) The data and delimiters are synchronous to clock reference.

c) They provide independent 32-bit-wide transmit and receive data paths.

d) They support full duplex operation only.

The MPRS may be used to provide asymmetric data rates, such as 25 Gb/s in the downstream direction and 10 Gbps Gb/s in the upstream direction. For information on use of the 10 Gigabit Media Independent Interface (XGMII) interface to provide this functionality, see Channels with asymmetric rates .



Per D0.5 and comment database.

Figure 143- 1- 100G-EPON Layering diagram.

143.2 Summary of major concepts

This subclause applies to the interface between the MAC and PHY in an optical line terminal (OLT) or an optical network unit (ONU). The physical implementation of the 25GMII is primarily intended to be chip-to-chip, but may also be used as a logical interface. This interface is used to provide media independence, so that an identical media access controller may be used with all 100G-EPON PHY types.

The following are the major concepts of this MPRS:

a) The MPRS requires use of a complementary Multipoint Control Protocol (MPCP) MAC control layer (see Clause 144).

b) The MPRS converts between the MAC serial data stream and the parallel data paths of up to four 25GMIIs servicing separate PHYs.

c) The P2MP nature of the PON system is enabled through the use of logical link identifiers (LLIDs) which associate the multiple MACs at the OLT with those at each ONU.

d) The MPRS maps the signal sets provided by the 25GMIIs to the PLS service primitives of individual MACs.

e) Each direction of data transfer is independent and serviced by data, control, and clock signals.

f) The MPRS generates continuous data or control characters on the transmit path and expects continuous data or control characters on the receive path.

143.2.1 Concept of a logical link and LLID

The 100G-EPON architecture is best viewed as a collection of logical point-to-point and point-to-multipoint links. A point-to-point logical link connects a single MAC instance at the OLT to a single MAC instance at the ONU. A point-to-multipoint logical link connects a single MAC instance at the OLT to multiple MAC instances at the ONUs. The OLT and ONUs may have multiple MAC instances and therefore may be connected with each other via multiple logical links.

A logical link is created in the MPRS (below MAC) by tagging each frame (or frame fragment) with a LLID value. The MPRS Transmit function inserts a specific LLID value depending on which instance of MAC has sourced the frame. The MPRS Receive function directs the received frame to the specific MAC instance mapped to this LLID value, or to multiple MAC instances, in case of point-to-multipoint logical link.

The 100G-EPON architecture uses several classes of LLIDs to provide enhanced segregation of PON transport layer control, management, and user traffic flows. These LLID classes are described in the following subclauses.

143.2.1.1 Physical Layer ID

The Physical Layer ID (PLID) is introduced to address specific needs of the PON physical layer. A successful ONU discovery and registration process, described in 144.x.x, results in the assignment of a single unique PLID value to the ONU. Traffic flows critical to 100G-EPON physical layer functions, such as multipoint control protocol data units (MPCPDUs) are transported using the PLID.

143.2.1.2 Management Link ID

Management Link ID (MLID) is used to carry management traffic flows, such as OAMPDUs (see 57.xxx). Each ONU is assigned a single MLID value as part of the ONU discovery and registration process, described in 144.x.x.

143.2.1.3 User Link ID

Subscriber traffic is carried using a separate class of LLID referred to a User Link ID (ULID). It is expected that a single subscriber may be assigned one or more ULIDs to allow for separation of traffic classes and types. ULID values are assigned (provisioned) to the ONUs using an appropriate management protocol outside the scope of this standard. ULID values need not have a one-

to-one binding of OLT MAC to an ONU MAC. A ULID value that binds a single OLT MAC to multiple ONU MACs represents a multicast ULID. Group Link ID

To assist in traffic management the 100G-EPON system supports consolidation of several LLIDs into arbitrary groups using the Group Link ID (GLID). For example, all LLIDs for a specific subscriber hosted on an ONU servicing numerous subscribers could be grouped together into a single GLID; in another example all LLIDs supporting a specific traffic class (e.g., best-effort traffic) on a multi-subscriber ONU could be grouped together. GLID values are used only for bandwidth granting and reporting purposes. The actual transmission is always identified by a PLID, an MLID, or a ULID value.

143.2.2 25Gb/s, 50Gb/s, and 100Gb/s operation over P2MP media

The 100G-EPON MPRS defined in this clause supports MAC data rates of 25Gb/s, 50Gb/s, and 100Gb/s.

143.2.2.1 MPRS channels

The 100G-EPON MPRS is defined as a set of four parallel MPRS channels in each direction. Each MPRS channel operates at 25Gb/s data rate. Each MPRS channel is bound to a separate PCS instance via a separate 25GMII instance. Thus, for any given system, there is a one-to-one correspondence between the MPRS channel count and the number of 25GMII instances supported.

Compliant implementations are not required to support all four MPRS channels. An implementation containing a single MPRS channel supports a MAC data rate of 25Gb/s. An implementation containing two MPRS channels supports MAC data rates of 25Gb/s and 50Gb/s. An implementation containing all four channels supports 25Gb/s, 50Gb/s, and 100Gb/s MAC data rates. Compliant implementations may include unequal numbers of receive and transmit channels, thus supporting asymmetric MAC data rates.

Table 143- 1 MPRS channel designation and capabilities

Designation	MPRS Channel	MPRS Channel Function
DC0	Downstream channel 0	All ONUs receive this MPRS channel, broadcast, ONU discovery
DC1	Downstream channel 1	Only ONUs capable of receiving at 50Gb/s and 100Gb/s support this MPRS channel.
DC2	Downstream channel 2	Only ONUs capable of receiving at 100Gb/s support these MPRS channels.
DC3	Downstream channel 3	
UC0	Upstream channel 0	All ONUs transmit on this MPRS channel, ONU discovery
UC1	Upstream channel 1	Only ONUs capable of transmitting at 50Gb/s and 100Gb/s support this MPRS channel.
UC2	Upstream channel 2	Only ONUs capable of transmitting at 100Gb/s support these MPRS channels.
UC3	Upstream channel 3	

143.2.2.2 Binding of multiple MACs to multiple PCS instances

The key function of the MPRS is to dynamically bind a PLS_DATA[*m*] interface to one or more MPRS channels (*m* represents the index of the MAC instance). The dynamic nature of the binding means that such a binding exists only for a predetermined interval of time during which a given MAC instance is expected to

transmit or receive data. After that time, the binding no longer exists, and a different MAC instance may bind to the same MPRS channel.

If the PLS_DATA[*m*] interface is bound to a single MPRS channel, the corresponding MAC instance is able to transmit and receive at a data rate of 25 Gb/s. However, a single PLS_DATA[m] interface may be bound to *N* MPRS channels (*N* = {1,2,3,4}). In this case, the MAC instance supports the data rate of *N*×25 Gb/s.

The dynamic binding of MAC instances to MPRS channels is controlled by the MPRS_CTRL.request() primitive described in 144.x.x.x.

143.2.3 Transmission and reception over multiple MPRS channels

143.2.3.1 Transmission unit

In 100G-EPON systems, the basic unit of transmission is the Envelope Quantum (EQ). Within the MPRS, one EQ is represented by a 72-bit vector consisting of 64 data bits and eight control bits. An EQ is mapped to two successive 25GMII transfers as shown in Figure 143- 2. Within the PCS, each EQ is converted into a single 66-bit block, according to the 64B/66B encoding rules (see 142.x.x.x).

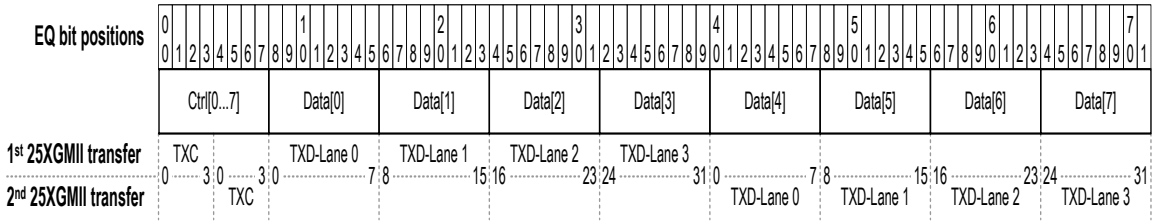
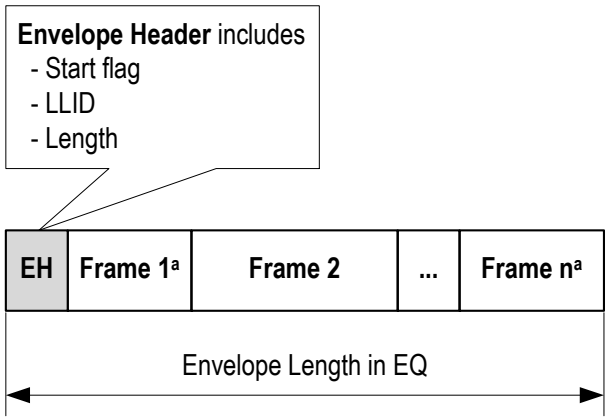


Figure 143- 2 - Envelope Quantum (EQ) Format

143.2.3.2 Transmission Envelopes

Transmitted data in 100G-EPON is contained in transmission envelopes. A transmission envelope represents a continuous transmission by a specific MAC instance (LLID) on a specific MPRS channel. A transmission envelope on a single MPRS channel encapsulates one or more data frames and can contain at most two partial frames (one at the beginning and one at the end of the envelope) and any number of whole frames (see Figure 143- 3).



a – possibly a fragment

Figure 143- 3 - Transmission envelope structure

143.2.3.3 Envelope Headers

Each transmission envelope begins with an envelope header (see Figure 143- 4). The envelope header consists of multiple fields, such as LLID, Length, Start flag, and other fields defined in 143.4.2.

The LLID field identifies a specific logical link. The LLID may represent a PLID, an MLID, or a ULID.

The size of the envelope header is exactly one EQ. The envelope header includes the Length field that shows the length of the entire envelope in units of EQ. The envelope header itself is counted as part of the envelope, therefore the minimum value of the Length field is one.

There are two distinct types of envelope headers; an envelope start header (ESH) and an envelope continuation header (ECH).

The ESH is inserted into the transmission stream at the beginning of every envelope, while no data is being taken from the corresponding MAC instance. At the receiving end, the ESH is discarded and no bits are passed to the corresponding MAC instance.

The ECH is inserted into the transmission stream in place of a data frame preamble. The length field of the ECH shows the residual length of the envelope. At the receiving end, the ECH is replaced with a normal frame preamble, which is passed to the corresponding MAC instance.

To distinguish the ESH and ECH, the envelope header includes a field called the Start flag. In ESH, the Start flag carries the value of 1 and in ECH, the flag carries the value of 0. Figure 143- 3 illustrates a transmission sequence for a single LLID N transmitting three frames (the first and the last frames are fragments). The format of the envelope header and field definitions are specified in 143.4.2.

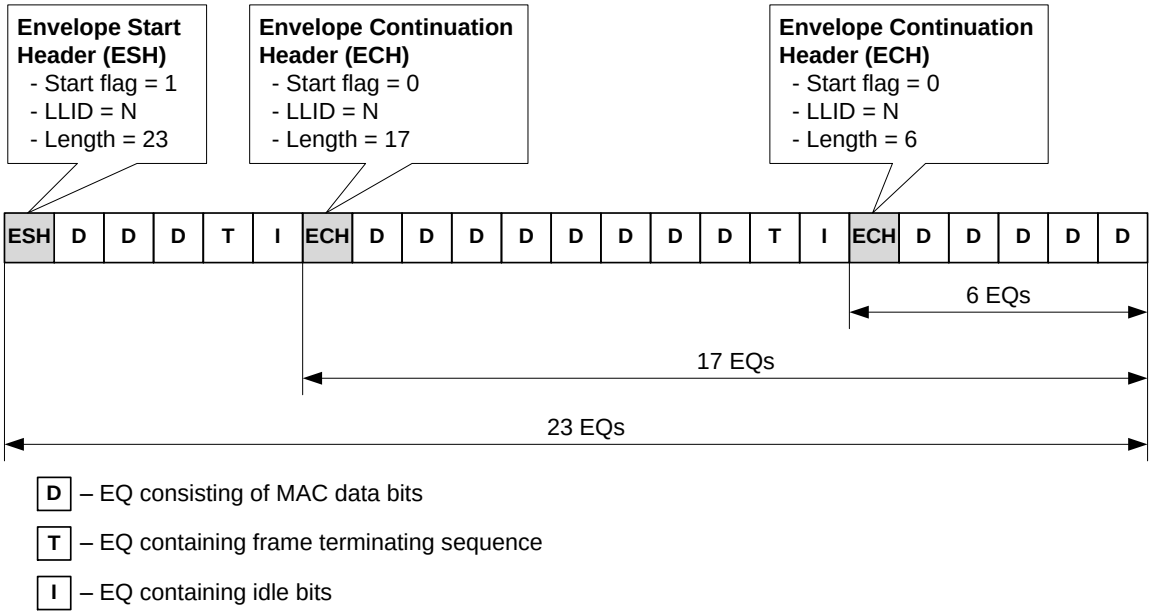


Figure 143- 4 - An illustration of a transmission sequence consisting of three frames

143.2.3.4 Interpacket gap adjustment

The 25GMII requires the alignment of the Start control character (first octet of preamble) to lane 0. Generally, a technique called Deficit Idle Count is used to accomplish this task (see 46.3.1.4). However, in the 100G-EPON systems, the MPRS replaces the frame preambles with an ECH. Therefore, in 100G-EPON systems, there is an additional requirement for the Start control character to be aligned to octet 0 of an EQ, such that the entire preamble occupies exactly one EQ and is not split across two consecutive EQs. To achieve such alignment, rather than maintaining a deficit idle count, the interpacket gap (IPG) is either unchanged or reduced, but is never expanded. The IPG may be reduced by up to seven octets from its default size of 96 bits. For the back-to-back data frames, the minimum guaranteed IPG is five octets.

The exact size of the IPG depends on the length of the previous data frame (for the case of back-to-back frames). Figure 143- 5 illustrates the IPG reduction for all possible positions of the end-of-frame character. The default preamble generated by the MAC and the reduced preamble are highlighted.

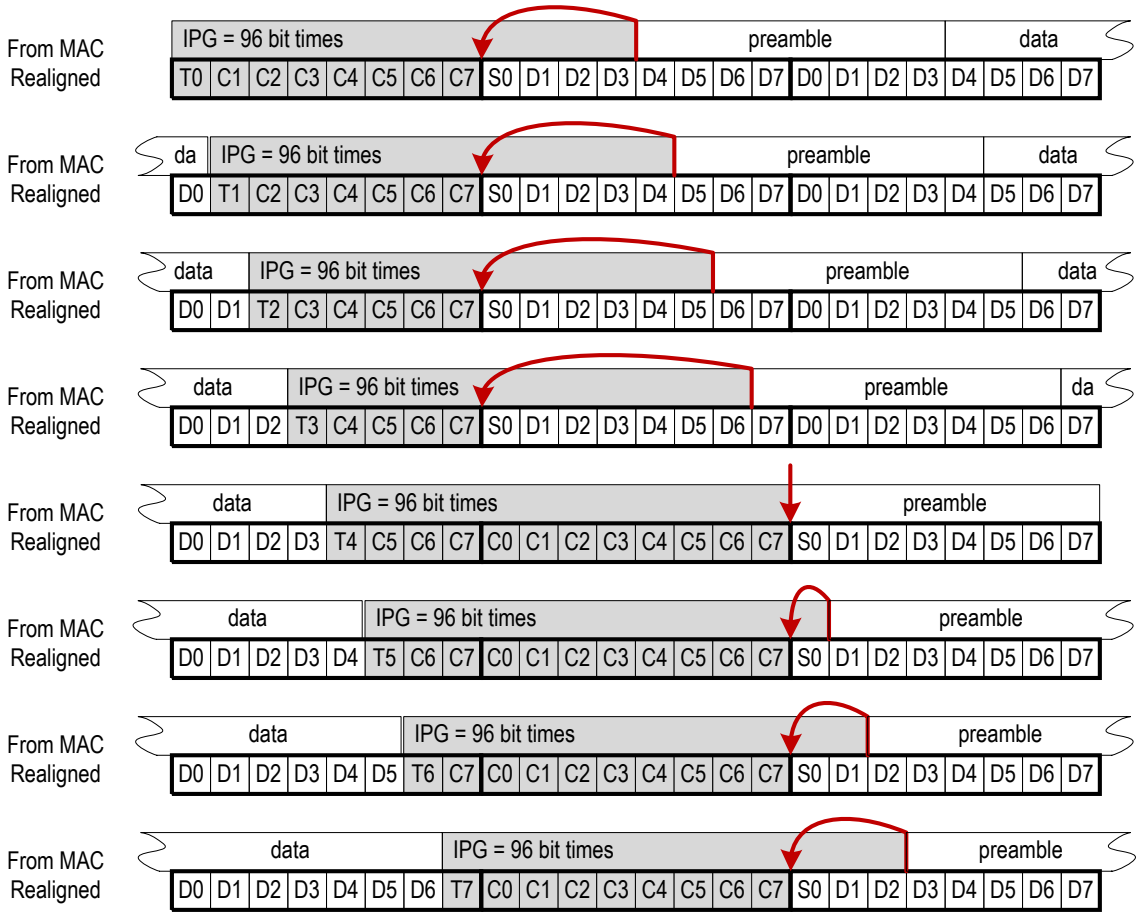


Figure 143- 5 - An illustration of a Start control character alignment to octet 0

The minimum IPG of five octets is consistent with the requirements of section 46.2.1 for XGMII (and hence applicable to 25GMII). The specified method of Start control character alignment in 100G-EPON systems does not cause the maximum MAC rate to exceed the specified limit because FEC parity overhead always exceeds the data rate gain due to IPG reduction (see 142.x.x on FEC details).

143.2.4 Dynamic channel bonding

A 100G-EPON system may serve ONUs that support different numbers of MPRS channels (see 143.2.2.1). Therefore, some ONUs are only able to receive and transmit data on MPRS channels DC0 and UC0, some are able to transmit on DC0 and DC1 and transmit on UC0 and UC1 or just on UC0. Some ONUs may support all four MPRS channels in each direction. Such flexibility of media access may result in a single LLID being assigned transmission envelopes on more than one channel. Such envelopes may happen to activate at the same time and to have the same duration, but most often the envelopes are not mutually aligned and just partially overlap as illustrated in Figure 143- 6.

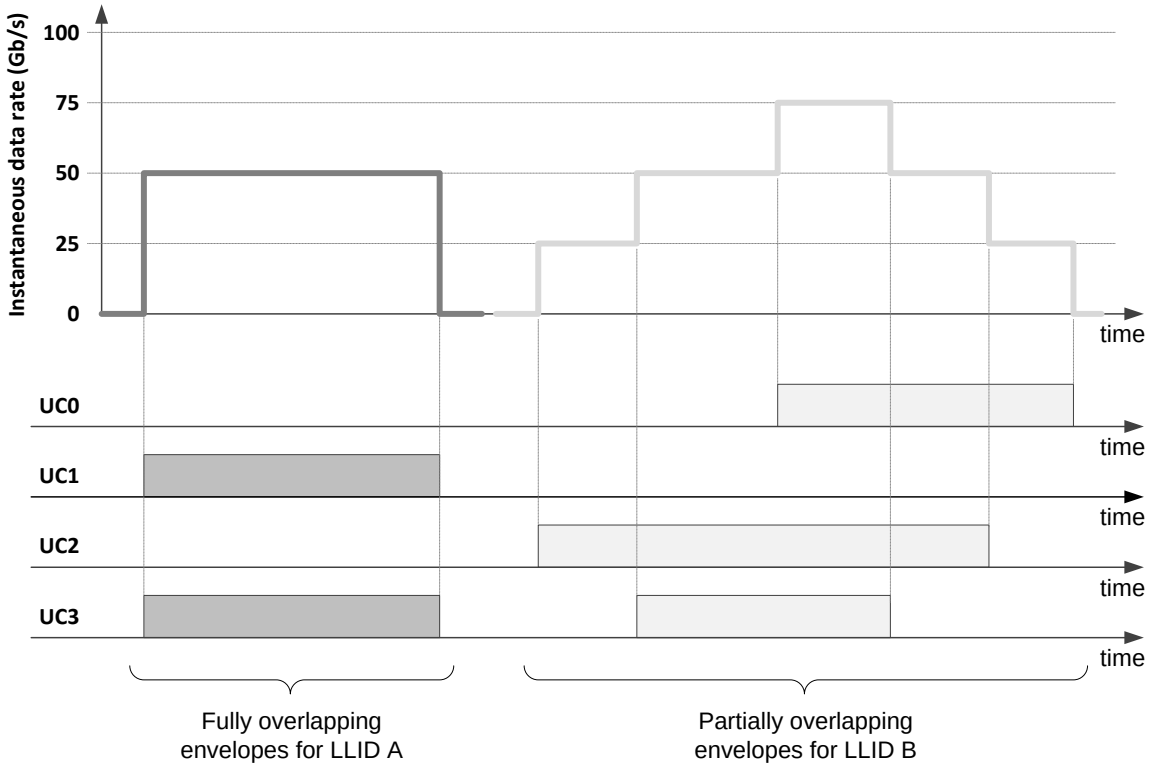


Figure 143- 6 - Full or partial envelope overlap and the resulting instantaneous data rate

An LLID that is given two or more overlapping envelopes on several MPRS channels is able to seamlessly increase its transmission data rate to the aggregated data rate of all the MPRS channels with the overlapping envelopes. This is referred to as dynamic channel bonding and it gives the 100G-EPON system the ability to achieve a higher instantaneous transmission rate than is available for any single MPRS channel. Thus, a 100G-EPON system with four MPRS channels of 25 Gb/s each can achieve an instantaneous transmission rate of 25, 50, 75, or 100 Gb/s by varying, in real time, the number of channels that are bonded to send data from a single LLID.

143.2.4.1 LLID transmission over multiple MPRS channels

The dynamic channel binding is achieved by interleaving data belonging to a single LLID (i.e., data from a single MAC instance) over multiple envelopes on multiple MPRS channels, as illustrated in 7. The unit of interleaving is one EQ. The overlapping envelopes are filled with EQs in the increasing order of MPRS channel index.

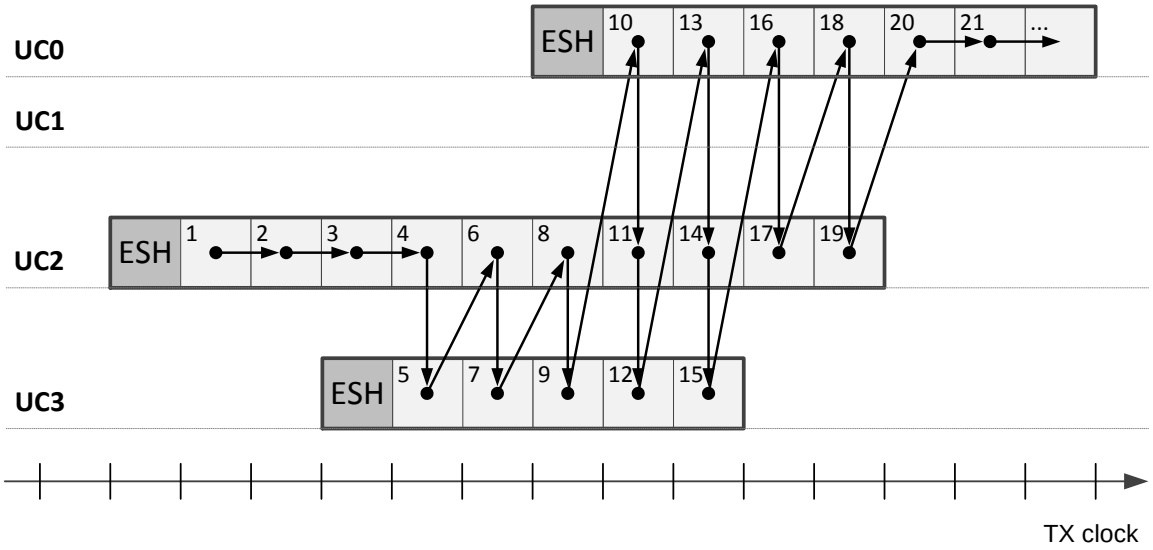


Figure 143- 7 - Fill order of overlapping envelopes

143.2.4.2 MPRS channel skew remediation mechanism

In a 100G-EPON system that uses multiple wavelengths to carry different MPRS channels, the channels have unequal propagation delays due to the dependence of speed of light in the fiber on the wavelength. This variable propagation delay results in a timing skew between signals received on separate MPRS channels. Other timing variabilities can accumulate in the sublayers below the MPRS, exacerbating this timing skew.

To properly restore the order of data transmitted over multiple bonded MPRS channels, the skew between the channels has to be eliminated at the receiver. The skew remediation mechanism is based on two buffers: an envelope transmission buffer (ENV_TX) in the transmitting MPRS and an envelope reception buffer (ENV_RX) in the receiving MPRS. As envelopes traverse the ENV_TX buffer (before the skew has impacted any of the MPRS channels), their relative position in the ENV_TX buffer is recorded and transmitted to the ENV_RX. At the receiving station, the envelopes received on multiple channels are aligned in the ENV_RX buffer using the position information received from the transmitting device. The relative alignment of envelopes in the ENV_RX becomes identical to their relative alignment that existed in ENV_TX. This envelope alignment method results in the complete elimination of any skew between the channels, as well as any timing variabilities that may accumulate in the sublayers below MPRS.

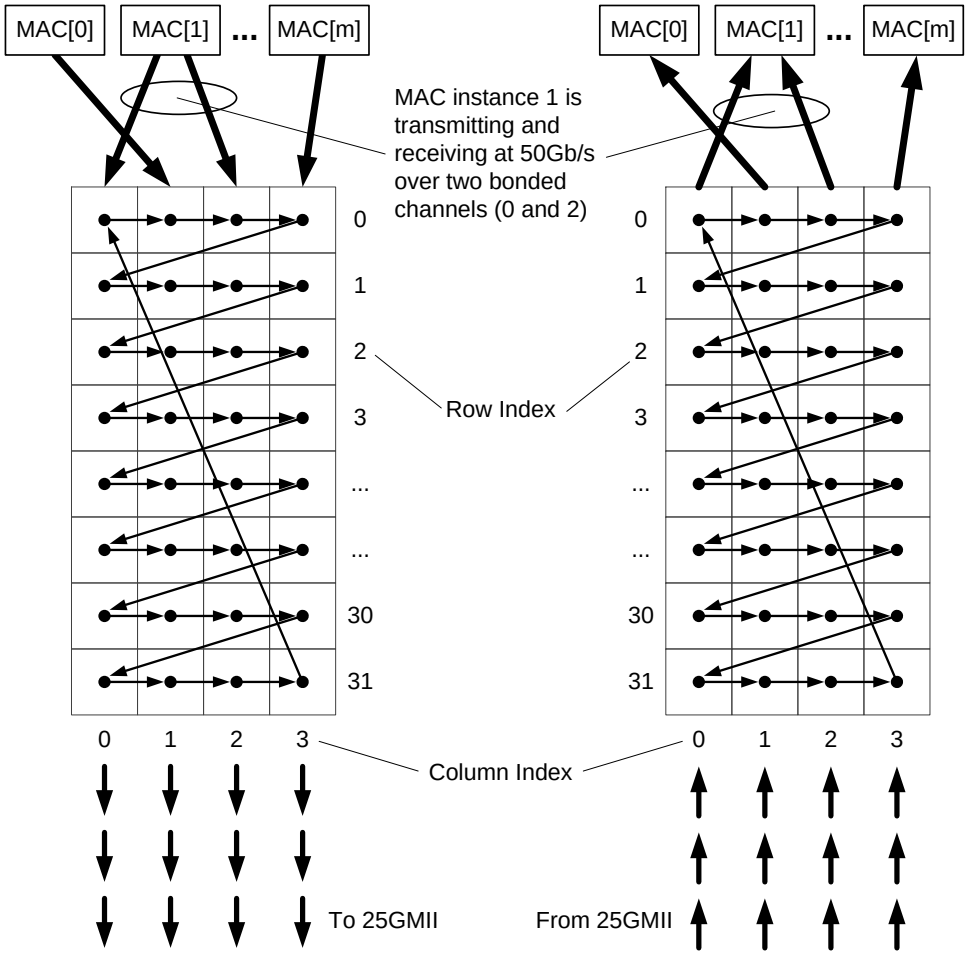
143.2.4.3 ENV_TX and ENV_RX buffers

The ENV_TX and ENV_RX buffers are two-dimensional buffers organized into rows and columns. The number of columns is equal to the number of channels supported by the device. The number of rows is set to 32, which provides buffering delay that is at least twice as large as the maximum possible skew/delay variability among any pair of channels. Each element of the buffer holds a 72-bit vector containing one EQ (see 143.2.3.1).

The EQs are written into the ENV_TX and read from ENV_RX buffers first by row, then by column, as shown in Figure 143.x.

In the ENV_TX buffer all columns of a row are written before the write pointer shifts to the next row. The EQs written into each column may be sourced by different MAC instances, if the envelopes on different channels belonged to different LLIDs, or from the same MAC instance, in case of multiple channels bonded to serve the same LLID (see Figure 143. **{Fill order of overlapping envelopes}**).

Similarly, in the receiving device, the EQs read from different columns may be passed to different MAC instances, if the envelope headers on different channels carried different LLID values, or the EQs may be passed to the same MAC instance, if the envelope headers carried the same LLID value. In case of EQs from different columns being passed to the same MAC instance, the EQ from a column with the lower index is always passed before an EQ from a column with the higher index.



(a) ENV_TX buffer **(b) ENV_RX buffer**

Figure 143- 8 Internal structure of ENV_TX and ENV_RX buffers

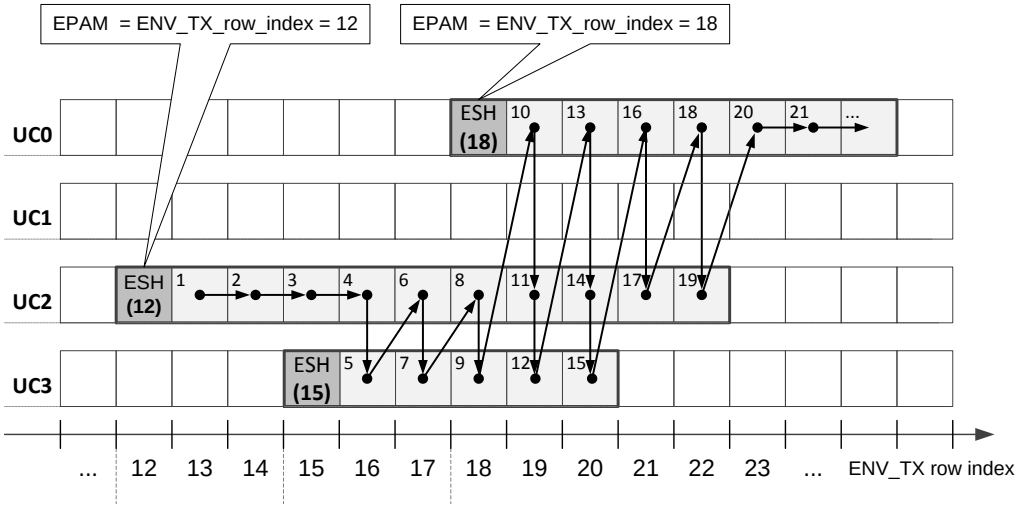
The ENV_TX and ENV_RX are circular buffers – after reading the last row, the read pointer shifts back to row 0. In ENV_TX, the read and write pointers advance synchronously with the 25GMII transmit clock (TX_CLK). In the ENV_RX, the read and write pointers advance synchronously with the 25GMII receive clock (RX_CLK). However, the value of the write pointer is updated whenever an envelope header is received.

{additional text needed to describe ENV_RX writing process}

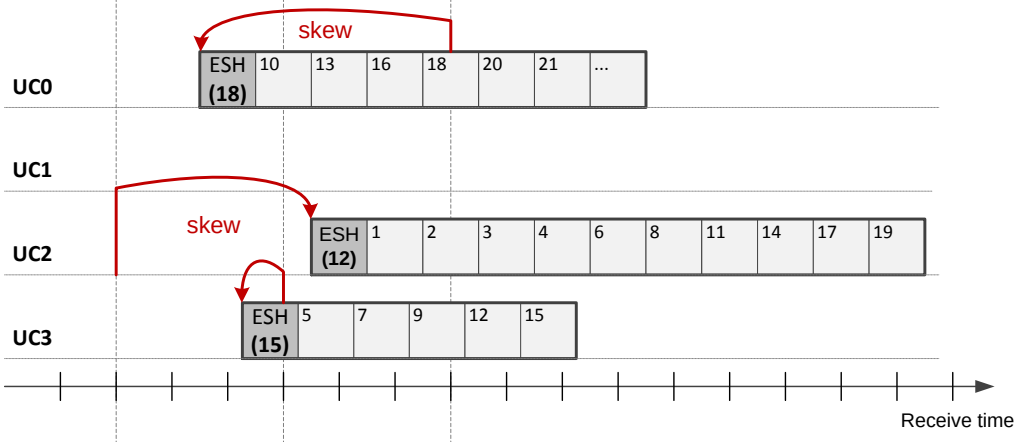
143.2.4.4 Envelope Position Alignment Marker

The relative envelope position recorded in an envelope header by the MPRS transmit function is simply the ENV_TX buffer row index into which the given envelope header was written. This information is placed in an envelope header field called the Envelope Position Alignment Marker (EPAM). When an envelope header is received by the MPRS receive function, the EPAM field is extracted and its value is used as the write pointer (row index) into which this envelope header is to be written. The remainder of the envelope is then written sequentially into the same column following the envelope header.

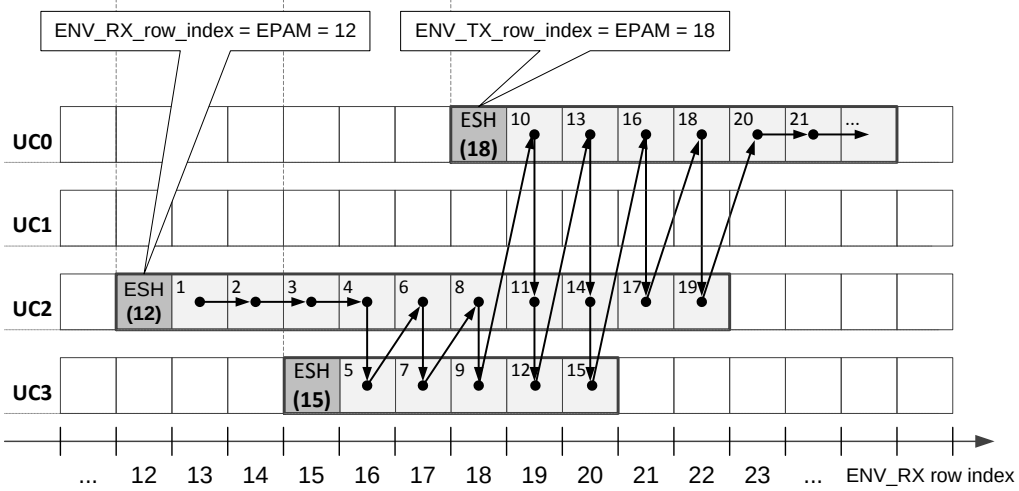
Figure 143.x illustrates (a) the initial envelope positions in the ENV_TX buffer, (b) the accumulated channel-dependent skew of the received channels at the ENV_RX buffer, and (c) the restored alignment based on EPAM value carried in each envelope header. As the true relative positions of the envelopes are restored, reading the data is the same order as shown in Figure 143.x properly serializes the data received over the multiple bonded channels. At the receiving station, regardless of the amount of accumulated skew, EQs transmitted at the same time from the same MPRS are placed in the same row of the ENV_RX. As the ENV_RX is read out in a row-by-row order over all channels the receiver effectively realigns the EQs to the same order they were transmitted in.



(a) Envelope Header records the row index of this header in the ENV_TX buffer



(b) Data is received on multiple channels with different propagation delays (skew)



(c) Envelopes are positioned in the ENV_RX buffer in a row index as recorded in the ESH

Figure 143- 9 Illustration of skew elimination by envelope position alignment in ENV_RX buffer

143.3 100G-EPON MPRS Requirements

{text}

143.3.1 MPRS and MPCP clock synchronization

{text - general description of clocks used in MPRS and their relationships, It might be a good idea to split this into three sub-section; common, OLT, and ONU}

143.3.2 Delay variability constraints

The Multi-Point Control Protocol (MPCP) relies on strict timing based on the distribution of timestamps. The actual delay is implementation dependent but an implementation shall maintain a combined delay variation through MPRS of no more than {TBD} EQ (see 144.x.x.x) so as not to interfere with the MPCP timing.

EDITORS NOTE: in the above paragraph derived from Cl 76.1.2, “1 TQ” was changed to “1 EQ”. In Cl 76.1.2 this applied to the combined MPRS, PCS, & PMA. A revised value may be needed.

143.4 MPRS Functional Specifications

143.4.1 MPRS Interfaces

Interfaces to the MPRS are illustrated in Figure 143- 10. In addition to the multiple PLS service interfaces (one per MAC) and up to four 25GMIIs there is a MPRS_CTRL interface that connects to the MPCP.

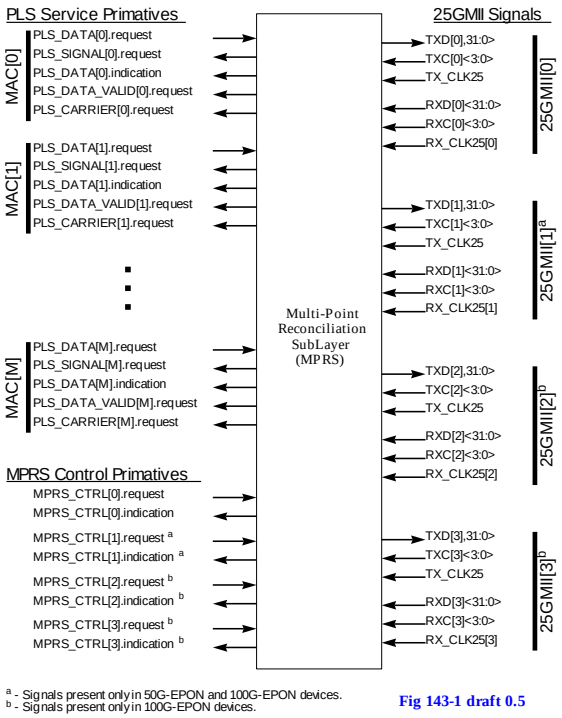


Figure 143–4—100G-EPON MPRS inputs and outputs

Figure 143- 10 100G-EPON MPRS input and outputs

143.4.1.1 PLS service primitives

In prior EPON generations, only one PLS service interface was active at any given moment; this is still true for 25/25G-EPON systems. However, for other 100G-EPON systems there may be one, two or four PLS service interfaces active at any given time.

The mapping of the PLS service primitives to 25GMII signals is shown in Table 143- 2 for PLS_DATA[].request primitives and in Table 143- 3 for PLS_DATA[].indications primitives. These are similar to the mappings described in 106.1.7. However, in multi-channel 100G-EPON systems there are multiple 25GMIIs and therefor an index is added to the 25GMII signals to indicate which of the 25GMIIs to use.

Table 143- 2 Mapping of PLS_DATA.request primitives

MAC operating speed	Transmit interface	Signals
25G-EPON	25GMII[*0*]	TXD[*0*]<31:0>, TXC[*0*]<3:0> and TX_CLK[1]

[1] All transmit 25GMII interfaces share a common clock.

50G-EPON	25GMII[*0*] 25GMII[*1*]	TXD[*0*]<31:0>, TXC[*0*]<3:0> and TX_CLK[a] TXD[*1*]<31:0>, TXC[*1*]<3:0>
100G-EPON	25GMII[*0*] 25GMII[*1*] 25GMII[*2*] 25GMII[*3*]	TXD[*0*]<31:0>, TXC[*0*]<3:0> and TX_CLK[a] TXD[*1*]<31:0>, TXC[*1*]<3:0> TXD[*2*]<31:0>, TXC[*2*]<3:0> TXD[*3*]<31:0>, TXC[*3*]<3:0>

Table 143- 3 Mapping of PLS_DATA.indication primitives

MAC operating speed	Receive interface	Signals
25G-EPON	25GMII[*0*]	RXD[*0*]<31:0>, RXC[*0*]<3:0> and RX_CLK[*0*]
50G-EPON	25GMII[*0*] 25GMII[*1*]	RXD[*0*]<31:0>, RXC[*0*]<3:0> and RX_CLK[*0*] RXD[*1*]<31:0>, RXC[*1*]<3:0> and RX_CLK[*1*]
100G-EPON	25GMII[*0*] 25GMII[*1*] 25GMII[*2*] 25GMII[*3*]	RXD[*0*]<31:0>, RXC[*0*]<3:0> and RX_CLK[*0*] RXD[*1*]<31:0>, RXC[*1*]<3:0> and RX_CLK[*1*] RXD[*2*]<31:0>, RXC[*2*]<3:0> and RX_CLK[*2*] RXD[*3*]<31:0>, RXC[*3*]<3:0> and RX_CLK[*3*]

143.4.1.1.1 Mapping of PLS_DATA[*ch*].request primitive

The MPRS maps the primitive PLS_DATA.request to the 25GMII signals TXD[*x*]<31:0>, TXC[*x*]<3:0>, and TX_CLK in the same way as for the XGMII as specified in 46.1.7.1.

143.4.1.1.2 Mapping of PLS_SIGNAL[*ch*].indication primitive

100G-EPON operation supports full duplex operation only. The MPRS does not generate the PLS_SIGNAL.indication primitive.

143.4.1.1.3 Mapping of PLS_DATA[*ch*].indication primitive

The MPRS maps the primitive PLS_DATA.indication to the 25GMII signals RXD[*x*]<31:0>, RXC[*x*]<3:0> and RX_CLK[*x*] in the same way as for the XGMII as specified in 46.1.7.2.

143.4.1.1.4 Mapping of PLS_DATA_VALID[*ch*].indication primitive

The MPRS maps the primitive PLS_DATA_VALID.indication to the 25GMII signals RXC[*x*]<3:0> and RXD[*x*]<31:0> in the same way as for the XGMII as specified in 46.1.7.5.

143.4.1.1.5 Mapping of PLS_CARRIER[*ch*].indication primitive

100G-EPON operation supports full duplex operation only. The MPRS never generates the PL-S_CARRIER.indication primitive.

143.4.1.2 MPRS Control Primitives

The 100G-EPON MPRS inputs the MPRS_CTRL[*ch*].request primitives from the MPCP and outputs to the MPCP the MPRS_CTRL[*ch*].indication primitives (see **Error! Reference source not found.**).

143.4.1.2.1 MPRS_CTRL[*ch*].request(*link_id*, *epam*, *env_length*) primitive

The MPCP requests the MPRS to transmit the next envelope using the MPRS_CTRL[*ch*].request(*link_id*, *epam*, *env_length*) primitive. This opens an envelope on channel *ch* for the LLID specified by *link_id* with a length (in EQs) of *env_length*. If all channels are idle the *EnvPam* variable (see Variables) is set to the value of *epam* (see EnvStartHeader() function definition in section Functions).

143.4.1.2.2 MPRS_CTRL[*ch*].indication(*cw_left*) primitive

The Input Process (see Figure 143- 13) requests the next envelope from the MPCP after the completion of the previous envelope using the MPRS_CTRL[*ch*].indication(*cw_left*) primitive. This primitive indicates to the MPCP that the MPRS is available for the next envelope and shows the available space in the current FEC codeword. In the absence of an active envelope, the MPRS_CTRL[*ch*].indication(*cw_left*) primitive is generated continuously on every *IN_CLK* transition (see Variables). The MPCP can decide whether to issue a new envelope immediately adjacent to the previous envelope (for envelopes that are expected to be packed in the same transmission burst), or wait for the start of next FEC codeword (for envelopes that are expected to be in a separate transmission burst). After the gap from the last envelope exceeds *GRANT_MARGIN* (the time required to turn on the laser), every MPRS_C-TRL[*ch*].indication(*cw_left*) indicates that a full FEC codeword is available (*cw_left* = FEC_CW_SIZE), implying that the next envelope starts with a new FEC codeword.

EDITORS NOTE: the paras above were largely taken from kramer_3ca_2c_0916 slide 27. We might want to move this to the MPCP clause and just ref MPCP from here.

143.4.1.3 25GMII interfaces

The 25GMII is specified to support 25 Gb/s operation. The structure of each of the 25GMII interfaces in a 100G-EPON system is identical to the XGMII structure specified in 46.1.6. The 25GMII data stream has the same characteristics as the XGMII data stream described in 46.2 with the exception of the clock rate which is 390.625 MHz for 25GMII (see 106.3).

For mapping between the 25GMII signals and the PLS Service interface, see PLS service primitives .

For multi-channel 100G-EPON systems the transmit 25GMIIs are synchronous and only one TX_CLK is required.

143.4.2 Envelope Header format

The transmission envelope header, as illustrated in **Error! Reference source not found.**, includes two successive 36-bit transfers over the 25GMII interface. Each 36-bit transfer includes four control bits followed by 32 information bits. These information bits contain the envelope header which includes a Start Control Code, a Start Flag bit, a 22-bit Envelope Length field, an Envelope Position Alignment Marker (EPAM) field, two bits (E and S) reserved for encryption purposes, an LLID field, and an 8-bit cyclic redundancy check (CRC8). The envelope length represents the number of EQ in the envelope. An EQ contains eight octets of information so the length of the envelope header is one EQ. The EPAM is used by the receiving MPRS to remove any timing skew that may have occurred during the transmission of the envelope from the transmitting MPRS to the receiving MPRS. The LLID field is set to the value of the LLID of the MAC associated with the data in the envelope. The CRC8 field is used for error detection within the header and covers all bits in the two 25GMII transfers of the header, including the two 4-bit TXC fields. There is one reserved bit (EQ bit 17) and it is set to zero at the transmitter and its value is ignored at the received except for the purposes of calculating the CRC8. The envelope header shall use the format show in Table 143- 4.

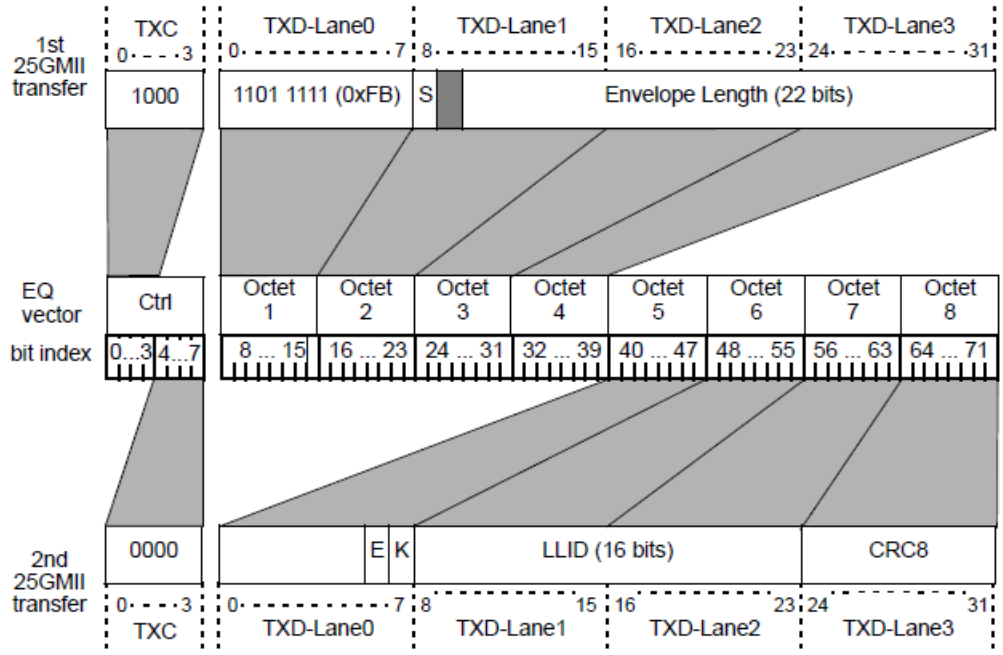


Figure 143- 11-Transmission envelope header format

Table 143- 4 Envelope Header EQ

EQ Bits	Value	Description
0-7	0x80	Control bits corresponding to TXC<3:0> in two successive MII transfers
8-15	0xFB	Start Control Code
16	0 for ECH 1 for ESH	Start Flag
17	0	reserved
18-39	varies	Length of envelope (in EQ)
40-45	varies	Envelope Position Alignment Marker (Number of bits matches the size of wRow)
46-47	0x0	reserved
48-63	varies	LLID
64-71	varies	CRC8

The envelope start header has the Start Flag set to one whereas the envelope continuation header has the Start Flag set to zero. The envelope start header is used to indicate the beginning of a transmission from a specific LLID. The Envelope Length field in the envelope start header does not include the envelope start header. The envelope continuation header replaces any preambles encountered in the transmission and, in this case, the Envelope Length field includes the envelope continuation header.

143.4.3 Transmit functional specifications

A functional block diagram of the 100G-EPON transmit path is illustrated in Figure 143- 12 The MPRS interfaces are described in MPRS Interfaces . The 100G-EPON transmit path is composed of two processes and one buffer. The Input Process, described in Input Process , accepts MAC data, formats it into EQs and stores these EQs in the ENV_TX buffer. The Transmit Process, described in Transmit Process , pulls EQs from the ENV_TX buffer and feeds them to two successive transfers on the appropriate 25GMII.

The MPRS in the OLT shall operate in unidirectional mode as defined in 66.4.

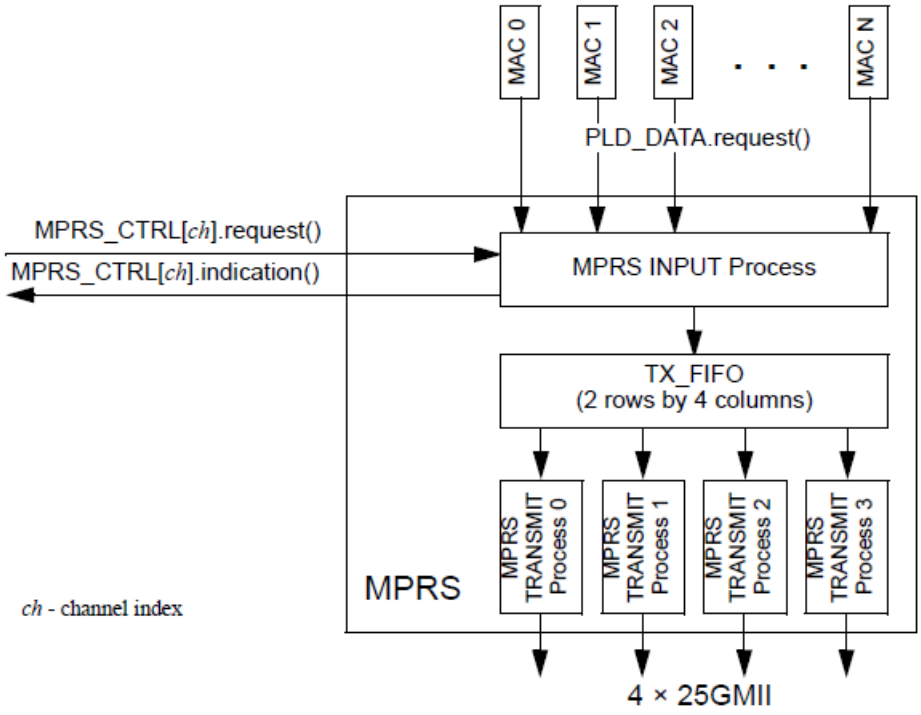


Figure 143- 12 100G-EPON MPRS transmit functional block diagram

143.4.3.1 Conventions

The notation used in the state diagrams in this subclause follows the conventions in 21.5. Should there be a discrepancy between a state diagram and descriptive text, the state diagram prevails. The notation + + after a counter indicates it is to be incremented by 1. The notation – – after a counter indicates it is to be decremented by 1. The notation – = after a counter indicates that the counter value is to be decremented by the following value. The notation + = after a counter indicates that the counter value is to be incremented by the following value. Code examples given in this clause adhere to the style of the "C" programming language. Variables are unsigned unless stated otherwise.

143.4.3.2 Constants

FEC_CW_SIZE
TYPE: integer
Value: {TBD}
The size of the FEC codeword in EQs.

FEC_PARITY_SIZE
TYPE: integer
Value: {TBD}
The size of the FEC parity word in EQs.

INTER_ENV_IDLE
TYPE: {TBD}

Value: {TBD}
A control code written to the ENV_TX when no active envelope exists (i.e., the channel is not assigned to any LLID).

PARITY_PLACEHLDR
TYPE {TBD}
Value: {TBD}
A control code written to the ENV_TX when data from MAC is paused to allow for FEC parity insertion. In the PCS sublayer, PARITY_PLACEHLDR code is overwritten by the calculated parity values.

PREAMBLE_EQ
TYPE: 72-bit vector
Value: 0x80 FB 55 55 55 55 55 55 5D
The value of an EQ returned by the GetMacBlock function which represents a preamble.

143.4.3.3 Variables

ch
TYPE: 2-bit integer
The *ch* variable represents the index of a specific 25GMII channel or the corresponding ENV_TX buffer, or ENV_RX buffer column.

CwdLeft[c]
TYPE: {TBD}-bit integer
The *CwdLeft* variables track the remaining space in the FEC codeword for channel c. Upon filling the payload portion of the codeword (*CwdLeft* = FEC_PARITY_SIZE), the Input Process will defer taking more data from the MAC to allow FEC parity to be inserted.

EnvLeft[*c*]
TYPE: 23-bit signed integer
If positive *EnvLeft* represents the length remaining in the current envelope for channel *c*, if negative this variable represents the number of EQ periods since the end of the last envelope on the channel.

EnvPam
TYPE: 6-bit integer
The *EnvPam* variable indicates the row index in the ENV_RX into which the received data is to be written, its primary function is to remove skew accumulated during transport between two or more channels from a single transmitter. This variable is set when all channels are idle and is loaded into the envelope header using the EnvStartHeader and EnvContHeader functions (see Functions). The variable is incremented after all ENV_TX columns have been read (i.e., once each *IN_CLK*).

GRANT_MARGIN
TYPE: {TBD}-bit integer
The *GRANT_MARGIN* variable represents the amount of time, in EQs, necessary to turn on the laser.

IN_CLK
TYPE: Boolean
The *IN_CLK* clear on read variable is set to True on each positive edge of the TX_CLK signal.

InEQ

TYPE: 72-bit binary array

A temporary holding variable for one EQ used in the Input Process.

LinkID[*c*]

TYPE: 16-bit integer

The *LinkID*[*c*] variables represent the MAC (LLID) being transferred by the Input Process or Output Process for channel *c*.

rRow

TYPE: 1-bit integer

The variable *rRow* represents the row in the ENV_TX buffer currently being read by the Transmit Process. The value of this variable is synchronized to *wRow* and is equal *wRow* - 1.

TX_CLK[*c*]

TYPE: Boolean

Each *TX_CLK*[*c*] clear on read variable is set to True on each edge, positive and negative, of the T-X_CLK signal for channel *c*.

ENV_TX[*c*][*r*]

TYPE: 72-bit binary array

The ENV_TX buffer is used to transfer information between the Input Process and the Transmit Process. Each cell, represented by the variables *ENV_TX*[*c*][*r*], in this buffer stores one EQ (a 72-bit vector) of information. The buffer has N columns (c) and two rows (r). The number of columns is dependent on the number of channels supported. For 100Gb/s devices N = 4, for 5050Gb/s devices N = 2 and for 25Gb/s devices N = 1. The buffer is filled in a cyclic pattern row-by-row. The source LLID for each cell is determined by the MPRS_CTRL[].request() primitive.

wCol

TYPE: 2-bit integer

The *wCol* variable represents the ENV_TX buffer column currently being written by the Input Process. Each column corresponds to a separate transmission channel, i.e., a separate 25GMII interface.

wRow

TYPE: 1-bit integer

The variable *wRow* represents the ENV_TX buffer row index currently being written by the Input Process. The value of *rRow* is synchronized to this variable and is equal to *wRow* - 1.

143.4.3.4 Functions

EnvContHeader(*wCol*)

The EnvContHeader() function returns a new envelope header with the Start Flag equal to 0, indicating that it is a continuation of the current envelope.

```
EnvContHeader(int2 col)
{
   EQ hdr;
   hdr<0:7> = 0x80;                                    //Control bits
   (1000-0000b)
   hdr<8:15> = 0xFB;                                   //S-character
```

```
   hdr<16> = 0;                                    //Envelope Continuation
   Header
   hdr<18:39> = EnvLeft[col];                                  //En-
   vLength
   hdr<40:45> = EnvPam;                                 //EPAM
   hdr<48:63> = LinkId[col];                                   //LLID
   hdr<64:71> = CRC8(hdr<0:63>);                                 //Cal-
   culate CRC8
   return hdr;
}
```

EnvStartHeader(*wCol*, *epam*)

The EnvStartHeader() function returns a new envelope header with the Start Flag equal to 1, indicating that it is a start of a new envelope. If this envelope starts a new burst (i.e., all channels are idle) it updates *EnvPam* to the value of the *epam* variable provided in the M-RPR_CTRL[].request primitive.

```
EnvStartHeader(int2 col, int5 epam)
{
   EQ hdr;
   // Use provided 'epam' value if this envelope starts a new burst
   if( EnvLeft[col+1] == GRANT_MARGIN &&
      EnvLeft[col+2] == GRANT_MARGIN &&
      EnvLeft[col+3] == GRANT_MARGIN ) EnvPam = epam;

   hdr<0:7> = 0x80;                                 //Control bits (1000-0000b)
   hdr<8:15> = 0xFB;                                  //S-character
   hdr<16> = 1;                                  //Envelope Start Header
   hdr<18:39> = EnvLeft[col];                                  //EnvLength
   hdr<40:45> = EnvPam;                                 //EPAM
   hdr<48:63> = LinkId[col];                                   //LLID
   hdr<64:71> = CRC8(hdr<0:63>);                                  //Calculate
   CRC8
   return hdr;
}
```

GetMacBlock(*link_id*)

The GetMacBlock function retrieves eight octets (64 bits) of data from a MAC identified by the *link_id* parameter and returns an EQ (72-bits) that contains both the data and the corresponding eight control bits. If the retrieved bits contain a partial frame preamble, the preamble is shifted forward such that the entire preamble is returned in one EQ in which case the function invokes the PLD_DATA.request() primitive up to 127 times. If no data is available from the MAC for a particular byte the function returns IDLE control code for that octet. This is a blocking function that returns control to the calling routine after 64 or more successive invocations of PLS_DATA.request() primitive.

```
IdleFlag[.] = {true};                    // Previous octet from a given MAC
   (link_id) was an
                      // Idle. Global array of booleans that retain their
                      // valuesbetween successive calls to GetMacBlock()

EQ GetMacBlock(int16 link_id)
{
   EQ eq;                        // Consists of 8 bits of control (Ctrl[0…7])
                         // and 8 octets of data (Data[0…7])
   for( octet_index = 0; octet_index < 8, octet_index++ )
   {
```

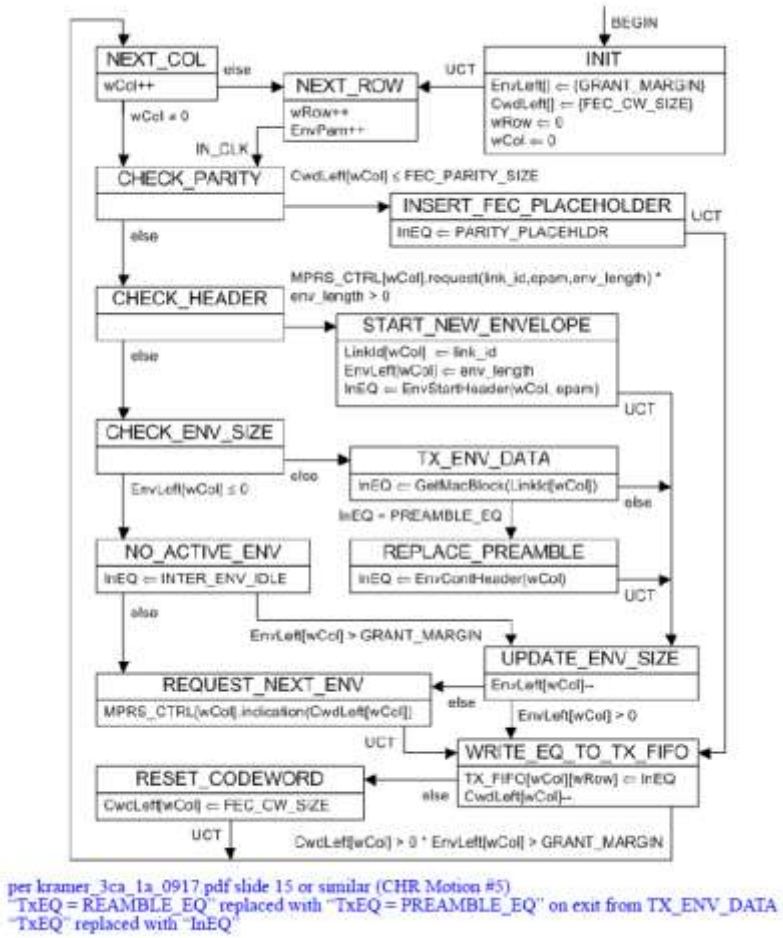
```
        tx_data = GetMacOctet( link_id );
        // Get 8 bits from MAC
        if( IsIdle(tx_data) AND !IdleFlag[link_id] )
                    // 1st Idle after Data
        {
           IdleFlag[link_id] = true;
           eq.Ctrl[octet_index] = 1;                                    //
     Store /T/-character
           eq.Data[octet_index] = 0xFD;
        }
        else if( IsIdle(tx_data) )                                      //
     Idle after Idle
        {
           eq.Ctrl[octet_index] = 1;                                    //
     Store /I/-character
           eq.Data[octet_index] = 0x07;
        }
        else if( IdleFlag[link_id] )
        // 1st Data after Idle
        {
           IdleFlag[link_id] = false;
           octet_index = 0;                                  // Shift to
     octet 0
           eq.Ctrl[octet_index] = 1;                                    //
     Store /S/-character
        eq.Data[octet_index] = 0xFB;
        }
        else                                            // Data after Data
        {
           eq.Ctrl[octet_index] = 0;                                    //
     Store Data octet
           eq.Data[octet_index] = tx_data;
        }
     }
     return eq;
  }
```

143.4.3.5 State Diagrams

143.4.3.5.1 Input Process

The 100G-EPON MPRS Input Process shall implement the state diagram as depicted in Figure 143- 13.

The Input Process accepts data from a MAC interface and transfers that data to the ENV_TX one EQ at a time. The process prepends a start envelope header to each envelope and overwrites each preamble with an envelope continuation header. Only one instance of the process is needed. The Input Process fills one full row (four columns) of the ENV_TX buffer on each cycle of *IN_CLK* (*IN_CLK* is half the effective rate of *TX_CLK*). In case of overlapping envelopes, blocks in multiple columns will be retrieved from the same MAC. The process keeps track of the envelope sizes for each LLID and does not exceed the allowed number of EQs for a given envelope. The process adjusts the MAC rate to account for FEC parity insertion in the PCS.



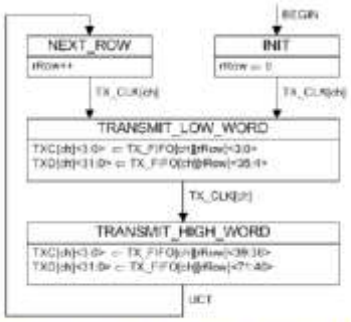
per kramer_3ca_1a_0917.pdf slide 15 or similar (CHR Motion #5)
"TxEQ = REAMBLE_EQ" replaced with "TxEQ = PREAMBLE_EQ" on exit from TX_ENV_DATA
"TxEQ" replaced with "InEQ"

Figure 143- 13 MPRS Input Process state diagram

143.4.3.5.2 Transmit Process

The 100G-EPON Transmit Process shall implement the state diagram as depicted in Figure 143- 14. One instance of the state diagram is instantiated for each 25GMII.

The Transmit Process outputs one 36-bit vector (TXD[x]<31:0> + TXC[x]<3:0>) to its associated 25GMII interface on each edge of the TX_CLK25 signal. There is one instantiation of the Transmit Process for each channel implemented in the device. The main function of the process is to transmit a column of one row from ENV_TX buffer on an existing channel. The Transmit Process is synchronized to TX_CLK25.



kramer_3ca_1a_1116.pdf slide 13 or similar (SA Motion #4)

Figure 143- 14 MPRS Transmit Process state diagram

143.4.4 Receive functional specifications

A functional block diagram of the 100G-EPON receive path is illustrated in Figure 143- 15. The MPRS interfaces are described in 143.4.1. The 100G-EPON receive path is composed of two processes and one buffer. The Receive Process, described in greater detail in Changes to Receive Process , accepts two successive transfers from the associated 25GMII and consolidates them into an EQ which is stored in the appropriate row of the ENV_RX. The Output Process, described in greater detail in Changes to Output Process , pulls EQs from the ENV_RX buffer and feeds them to the appropriate MAC as specified by the current LLID for that receive channel.

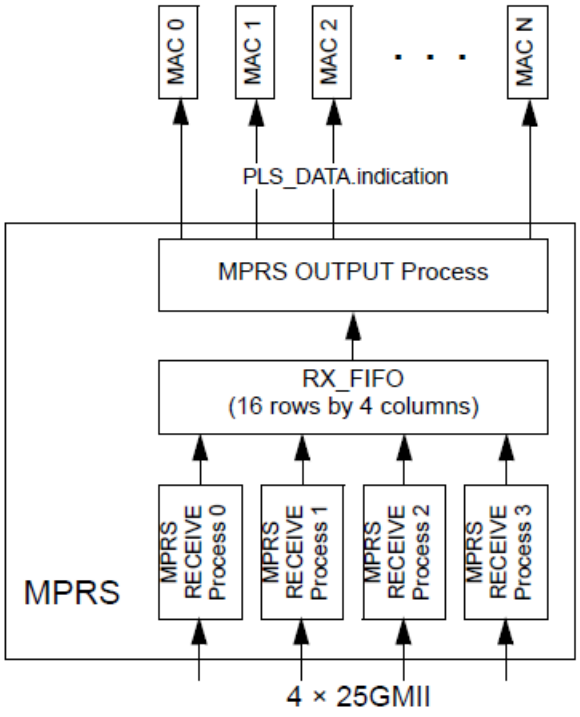


Figure 143- 15 100G-EPON MPRS receive functional block diagram

143.4.4.1 Conventions

See Conventions .

143.4.4.2 Constants

INTER_ENV_IDLE
See Constants .

PARITY_PLACEHLDR
See Constants .

143.4.4.3 Variables

ch
See Variables .

EnvLeft[c]
See Variables .

LinkID[c]
See Variables .

OUT_CLK
TYPE: Boolean
The *OUT_CLK* clear on read variable is set to True on each positive edge of *T-X_CLK* and runs at half the frequency of *TX_CLK*.

OutEQ
TYPE: 72-bit binary array
A temporary holding variable for one EQ used in the Output Process.

rCol
TYPE: 2-bit integer
The *rCol* variable represents the ENV_RX buffer column currently being read by the Output Process. Each column corresponds to a separate reception channel, i.e., a separate 25GMII interface.

rRow
TYPE: 6-bit integer
The *rRow* variable represents the ENV_RX buffer row index currently being read by the Output Process.

RX_CLK[*c*]
TYPE: Boolean
The *RX_CLK*[*c*] clear on read variables are set to True on each edge of the RX_CLK[*c*] signals and represent the continuous clock that provides the timing reference for the transfer of the RXC[*c*]<3:0> and RXD[*c*]<31:0> signals received on the 25GMII channel c.

ENV_RX[*c*][*r*]
TYPE: 72-bit binary array

The RX-FIFO buffer is used to transfer information between the Receive Process and the Output Process. Each cell, represented by the variables *ENV_RX*[*c*][*r*], in this buffer stores one EQ (a 72-bit vector) of information. The buffer has N columns (*c*) and M rows (*r*). The number of columns is dependent on the number of channels supported. For 100Gb/s devices N = 4, for 50Gb/s devices N = 2 and for 25Gb/s devices N = 1. The size of M is application specific but must be greater than or equal to the maximum value of *EnvPam*. The buffer is filled in a cyclic pattern row-by-row by the Receive Process and emptied by the Output Process.

RxEQ

TYPE: 72-bit binary

The *RxEQ* variable represents the most recent EQ received from a 25GMII interface.

143.4.4.4 Functions

IsHeader(*eq*)

The IsHeader(*eq*) function returns true if the parameter *eq* represents an envelope header. An envelope header always begins with a /S/ Start Control Character.

```
bool IsHeader(EQ eq)
 {
    return( eq<7:0> == 0x80 AND                                    //
    Control bits
       eq<15:8> == 0xFB AND                                  // Start
    Control Code /S/
       eq,64:71> == CRC8(eq<0:63>));                              //
    Matching CRC8
 }
```

AUTHORS NOTE: the IsHeader() function was originally defined to detect an Ordered Set. This has been changed to detect a Start Control Code.

IsMisaligned(*eq*)

The IsMisaligned(*eq*) function returns true if the parameter *eq* is misaligned, i.e., shifted by half-EQ.

```
 bool IsMisaligned(EQ eq )
 {
    return(( eq<39:36> == 0xF AND                                  // Mis-
    aligned NO_ENV_CODE
       eq<71:40> == NO_ENV_CODE_LO )                               // …
    s.b. NO_ENV_CODE_HI
       OR
       ( eq<39:36> == 0x8 AND                              // Misaligned
    Env. Header
       eq<47:40> == OCODE1 ));                                  // … s.b.
    OCODE2
 }
```

OutputToMac(*LinkID*[*rCol*], *OutEQ*)

The OutputToMac(*LinkID*[*rCol*], *OutEQ*) function transfers the eight information bytes in the *OutEQ* parameter to the MAC associated with the LLID value of *LinkID*[*rCol*] per the eight control bits in the *OutEQ* parameter.

```
 OutputToMac(int16 link_id, EQ eq)
 {
```

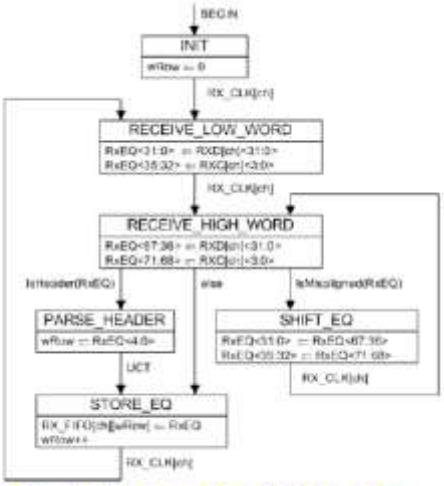
```
      for( octet_index = 0; octet_index < 8, octet_index++ )
      {
         if ( eq.Ctrl[octet_index] == 0 )
         // Receive data octet
         {
            data_valid = true;
            rx_data = eq.Data[octet_index];
         }
         else if ( eq.Data[octet_index] == 0xFB )
               // Rx /S/ control character
         {
            data_valid = true;
            rx_data = 0x55;                                        // Replace /S/
      with preamble
         }
         else                                          // Rx other ctrl. character
         {                                          // including /T/ (value 0xFD)
            data_valid = false;
            rx_data = 0x07;                                        // Replace with
      /I/
         }
         SetMacOctet( link_id, rx_data, data_valid );
                     // Set 8 bits to MAC
      }
   }
```

143.4.4.5 State Diagrams

143.4.4.5.1 Receive Process

The 100G-EPON Receive Process shall implement the state diagram as depicted in Figure 143- 16.

This process forms an EQ from two successive 25GMII transfers. The process first verifies proper alignment of the EQ and, if misaligned, shifts the input by half of an EQ (four bytes). No other error checking is performed by this process. When an envelope header is received, the EPAM field is extracted and used as a write position into the ENV_RX buffer. Because the phase of the receive clock (RX_CLK[*ch*]) in every channel is different, due to different ONUs and transport skew, a separate instance of the Receive Process is required for each channel implemented.



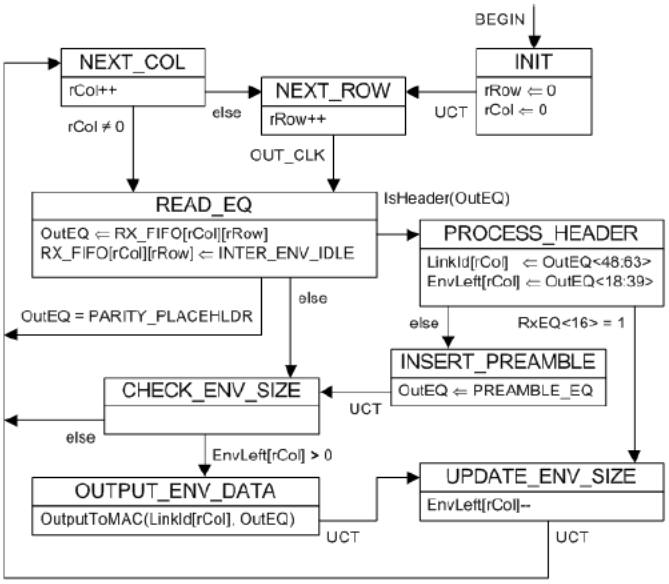
kramer_3ca_3b_1116.pdf slide 3 or similar (SA Motion #6)

Figure 143- 16 100G-EPON MPRS receive process state diagram

143.4.4.5.2 Output Process

The 100G-EPON MPRS Output Process shall implement the state diagram as depicted in Figure 143- 17.

The Output Process outputs EQs to the proper MAC. In the case of overlapping envelopes from the same LLID, data from multiple channels is properly serialized. A corrupted header may lead to loss of a frame, but no subsequent frames will be lost due to the error.



kramer_1a_0917.pdf slide 20 or similar (CHR Motion #5)
Replaced "RxEQ" with "OutEQ"

Figure 143- 17 MPRS Output Process state diagram

143.5 Channels with asymmetric rates

143.5.1 Mapping of 25GMII and XGMII primitives at the OLT

143.5.2 Mapping of 25GMII and XGMII primitives at the ONU

143.5.3 MPRS channel operation at 10 Gb/s

143.5.3.1 Changes to Input Process

143.5.3.2 Changes to Transmit Process

143.5.3.3 Changes to Receive Process

143.5.3.4 Changes to Output Process