Contents

143 Multi-Channel Reconciliation Sublayer

143.1 Overview

This clause describes the Multi-Channel Reconciliation Sublayer (MCRS) which enables multiple MACs to interface with multiple Physical Layers. Figure *143.1* shows the relationship between this MCRS and the ISO/IEC OSI reference model. Generally, single-channel RS specifications enabled a single MAC to interface to a single PHY in point-to-point links, or a multiple MACs to interface to a single PHY in P2MP links (e.g., EPON architectures). This concept is expanded in this clause to allow multiple MACs to interface with multiple PHYs (see Figure *143.1*).

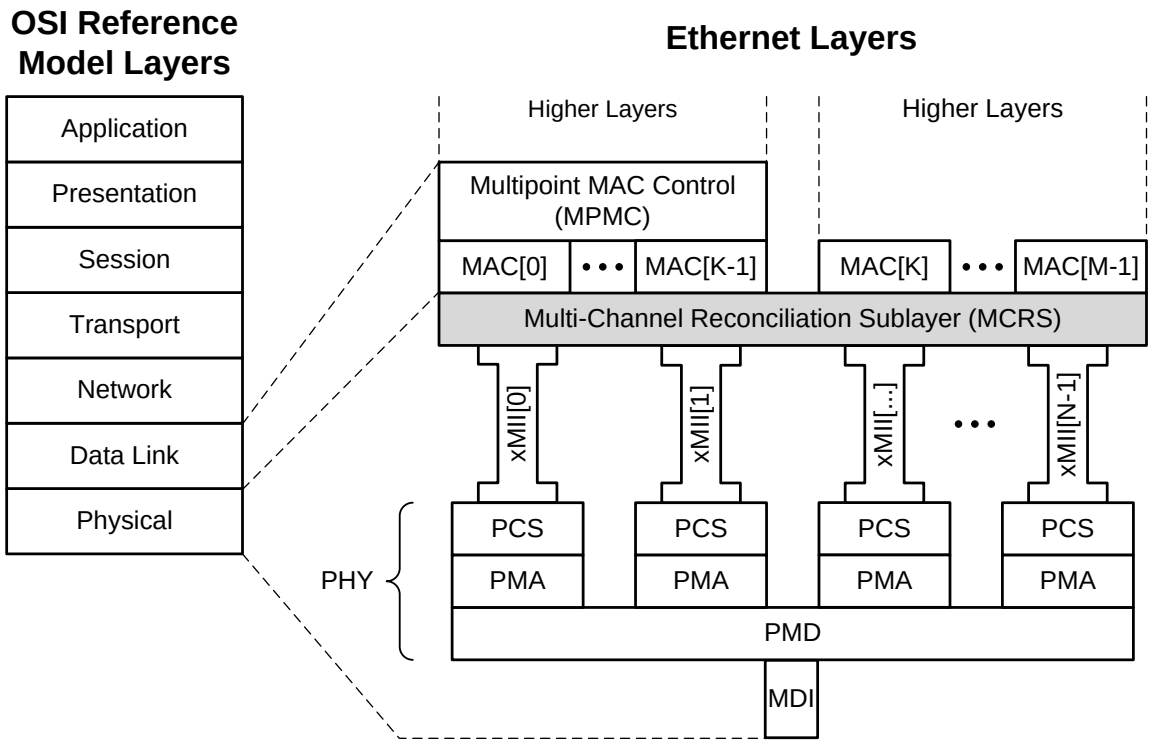

Figure 143.1 Relationship of MCRS to the OSI Reference Model

The MCRS adapts the bit-serial protocols of the MAC to the parallel format of the Physical Coding Sublayer (PCS) service interface. This clause defines an MCRS as an interface between the MAC sublayer and one or more xMIIs. In this clause, xMII is used as a generic term for the Media Independent Interfaces for implementations of 10 Gb/s and above. For example: for 10 Gb/s implementations, it is called XGMII; for 25 Gb/s implementations, it is called 25GMII. Though the xMII is an optional logical interface between the MAC sublayers and the Physical Layers, it is used extensively in this clause as a basis for specification.

Editor's Note: we should consider including MIIs operating at rates less than 10G and more than 25G if this is to be truly generic.

143.2 Summary of major concepts

The following are the major concepts of the MCRS:

a) The MCRS transmission is controlled by a higher layer (e.g., Multipoint MAC Control sublayer defined in Clause 144) via the use of MCRS_CTRL primitives.
b) The MCRS establishes a temporary binding of a single MAC instance to one or more xMII instances with all xMIIs operating at the same rate.
c) In the transmit direction, the MCRS converts the MAC serial data stream into the parallel data paths of multiple xMIIs servicing separate PHYs.
d) In the receive direction, the MCRS maps the signal sets provided by the xMIIs 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 MCRS generates continuous data or control characters in the transmit path and expects continuous data or control characters in the receive path.

143.2.1 Concept of a logical link and LLID

In point-to-multipoint architectures, such as EPON, the transmitting and receiving stations may include multiple MAC instances. Such architectures are best viewed as a collection of logical point-to-point and/or point-to-multipoint links. A point-to-point logical link connects a single MAC instance at the transmitting station to a single MAC instance at the receiving station. A point-to-multipoint logical link takes advantage of the P2MP topology and connects a single MAC instance at the transmitting station to multiple MAC instances at multiple receiving stations. The transmitting and receiving stations may be logically connected with each other via multiple logical links.

A logical link is created in the MCRS (below the MAC) by tagging each frame (or frame fragment) with a logical link identification (LLID) value. The MCRS Transmit function inserts a specific LLID value depending on which instance of MAC has sourced the frame. The MCRS Receive function directs the received frame (or frame fragment) to the specific MAC instance mapped to this LLID value, or to multiple MAC instances, in case of point-to-multipoint logical link. The concept of a logical link is further defined in {TBD 144}.

143.2.2 Concept of an MCRS channel

An MCRS channel is a single unidirectional transmission path through the MCRS. The number of channels contained within an MCRS generally corresponds to the number of xMII instances connected to the MCRS. Thus, an MCRS implementation that connects to N xMII instances contains N transmit MCRS channels and N receive MCRS channels. Some architectures (e.g., EPON) allow an xMII interface to only implement either receive or a transmit data path. In such architectures, the number of receive and transmit MCRS channels may be different.

143.2.3 Binding of multiple MACs to multiple xMII instances

The key function of the MCRS is the dynamic binding of a PLS_DATA[*m*] interface to one or more MCRS 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 MCRS channels.

The dynamic binding of MAC instances to MCRS transmit channels is controlled by the MCRS_CTRL.request() primitive described in 143.3.1.2.1. The dynamic binding of MAC instances to receive MCRS channels is determined by the LLID value of the incoming data.

143.2.4 Transmission and reception over multiple MCRS channels

143.2.4.1 Transmission unit

Within the MCRS, the basic unit of transmission is the Envelope Quantum (EQ). One EQ is represented by a 72-bit vector consisting of 64 data bits and eight control bits. For 36-bit wide xMIIs, such as 25GMII, an EQ is mapped to two successive xMII transfers as shown in Figure *143.2*.

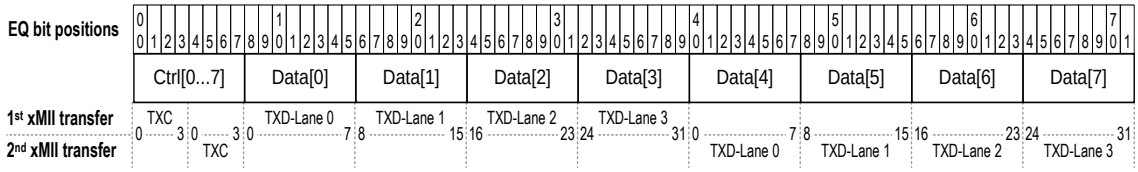

Figure 143.2 Envelope Quantum (EQ) format

143.2.4.2 Transmission Envelopes

The MCRS encapsulates data transmitted by a MAC instance in transmission envelopes. A transmission envelope represents a continuous transmission by a specific MAC instance (LLID) on a specific MCRS channel. A transmission envelope is always transmitted on a single MCRS channel. An envelope includes 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*).

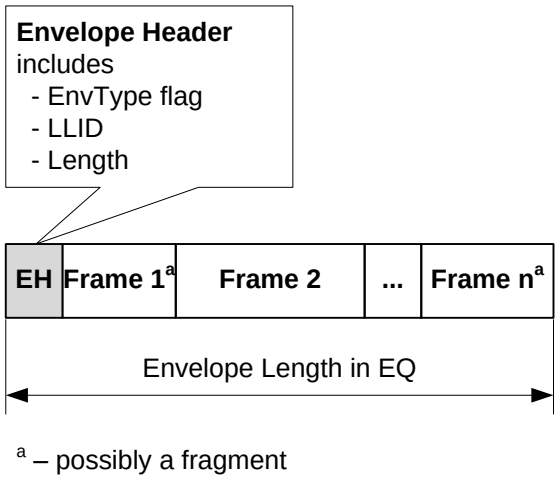

Figure 143.3 Transmission envelope structure

143.2.4.3 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, EnvType flag, and other fields defined in 143.3.2.

The LLID field identifies a specific logical link.

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 processed by the MCRS and then 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 EnvType flag. In ESH, the EnvType flag carries the value of 1 and in ECH, the flag carries the value of 0. Figure *143.4* 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 143.3.2.

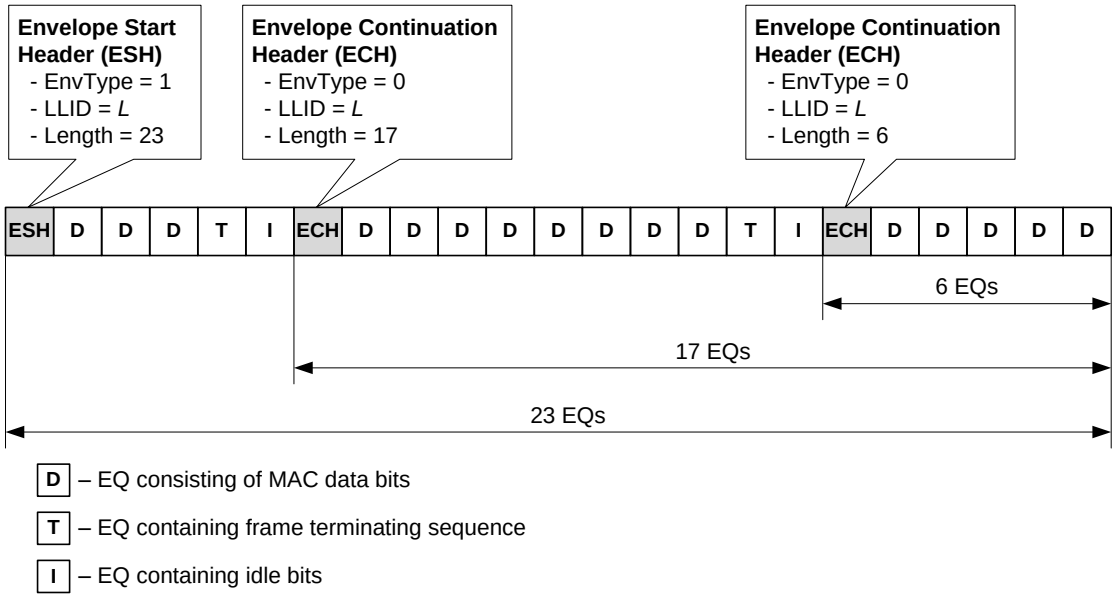

Figure 143.4 An illustration of transmission sequence consisting of three frames

143.2.4.4 Interpacket gap adjustment

Multi-lane xMIIs, such as 25GMII, require 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, because the MCRS replaces the frame preamble with an ECH, 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 not 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.

The minimum IPG of five octets is consistent with the requirements of 46.2.1 for XGMII (and hence applicable to 25GMII). Since the IPG either remains unchanged or is reduced, in order to prevent the MAC data rate from exceeding the specified maximum limit, MCRS provides a rate adjustment mechanism, whereby a MAC is paused for a predefined duration of time at a predefined repeating interval.

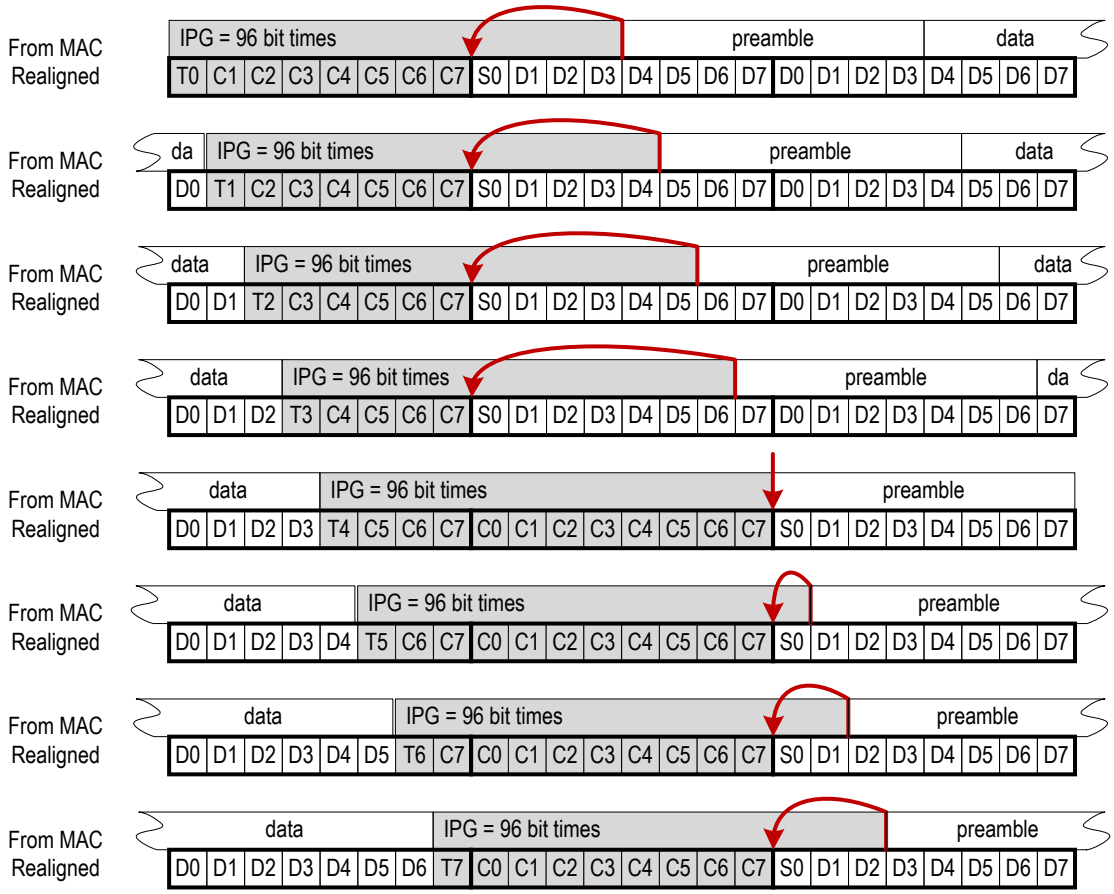

Figure 143.5 An illustration of a Start control character alignment to octet 0

143.2.5 Dynamic channel bonding

If the PLS_DATA[*m*] interface is bound to a single MCRS channel that is connected to an xMII instance, the corresponding MAC instance is able to transmit and receive at a data rate corresponding to that xMII data rate. For example, if the MCRS sublayer is connected to a 25GMII, that MAC instance is able to transmit and receive at 25 Gb/s.

However, in a system that supports multiple MCRS channels (i.e., MCRS is connected to multiple xMII instances), a single PLS_DATA[*m*] interface may be simultaneously bound to N_1 MCRS transmit channels and N_2 MCRS receive channels. In this case, again assuming the 25GMII, the corresponding MAC instance supports the transmit data rate of N_1×25 Gb/s and the receive data rate of N_2×25 Gb/s.

The channel bonding takes place when an LLID is assigned transmission envelopes on more than one channel. Such envelopes may happen to activate at the same time and to have the same duration, as illustrated in Figure 143-6 for LLID A. But most often the envelopes are not mutually aligned and just partially overlap as shown for LLID B.

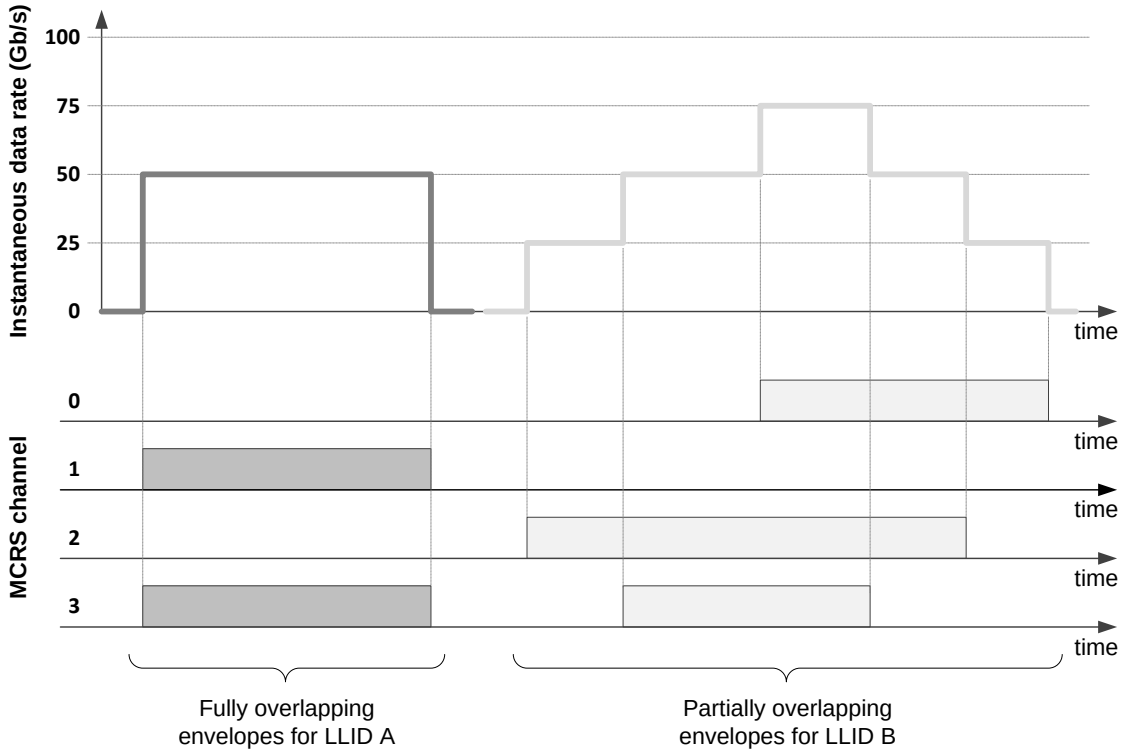

Figure 143.6 Full or partial envelope overlap and the resulting instantaneous data rate

An LLID (i.e., a MAC instance) that is given two or more overlapping envelopes on several MCRS channels is able to seamlessly increase its transmission data rate to the aggregated data rate of all the MCRS channels with the overlapping envelopes. This is referred to as dynamic channel bonding and it gives the system an ability to achieve a higher instantaneous transmission or reception rate than is available for any single MCRS channel. For example, a MAC instance connected to an MCRS with four 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.

143.2.5.1 LLID transmission over multiple MCRS channels

The dynamic channel bonding is achieved by interleaving data belonging to a single LLID (i.e., data from a single MAC instance) over multiple envelopes on multiple MCRS channels, as illustrated in Figure *143.7*.

The unit of interleaving is one EQ. The overlapping envelopes are filled with EQs in the increasing order of MCRS channel index.

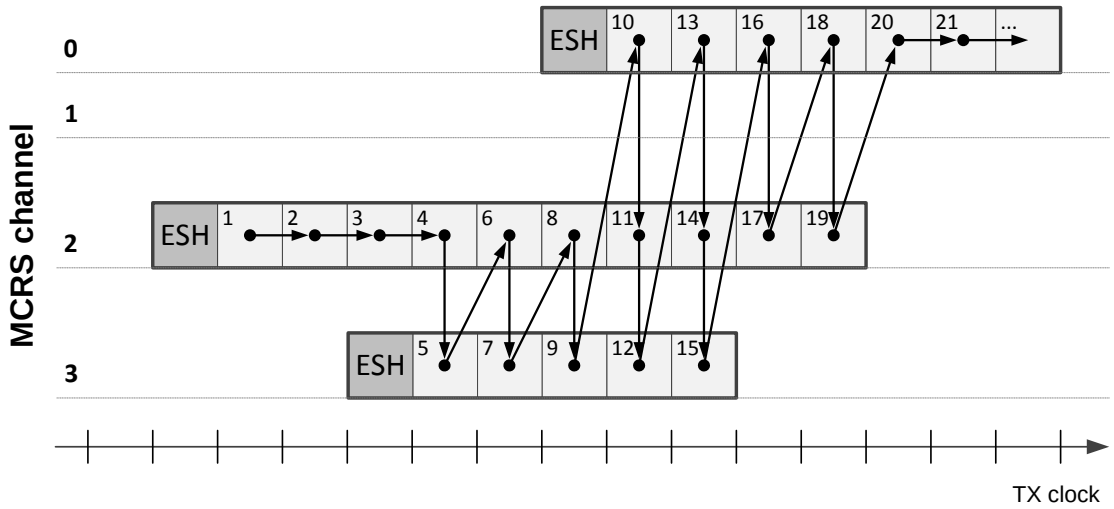

Figure 143.7 Fill order of overlapping envelopes

143.2.5.2 MCRS channel skew remediation mechanism

In a multi-channel system that uses multiple wavelengths to carry different MCRS channels, the channels have unequal propagation delays. This variable propagation delay results in a timing skew between signals received on separate MCRS channels. Other timing variability can accumulate in the sublayers below the MCRS, exacerbating this timing skew.

To properly restore the order of data transmitted over multiple bonded MCRS 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 MCRS and an envelope reception buffer (ENV_RX) in the receiving MCRS. As envelopes traverse the ENV_TX buffer (before the skew has impacted any of the MCRS 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 variability that may accumulate in the sublayers below MCRS.

143.2.5.3 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. This provides sufficient buffering to mitigate approximately 80 ns of skew between any two channels (assuming a 25GMII). If an application requires additional skew mitigation the number of buffer rows can be increased. Each element of the buffer holds a 72-bit vector containing one EQ (see 143.2.4.1).

The EQs are written into the ENV_TX and read from ENV_RX buffers first by row, then by column, as shown in Figure 143.8.

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 143.8).

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 passed before an EQ from a column with the higher index.

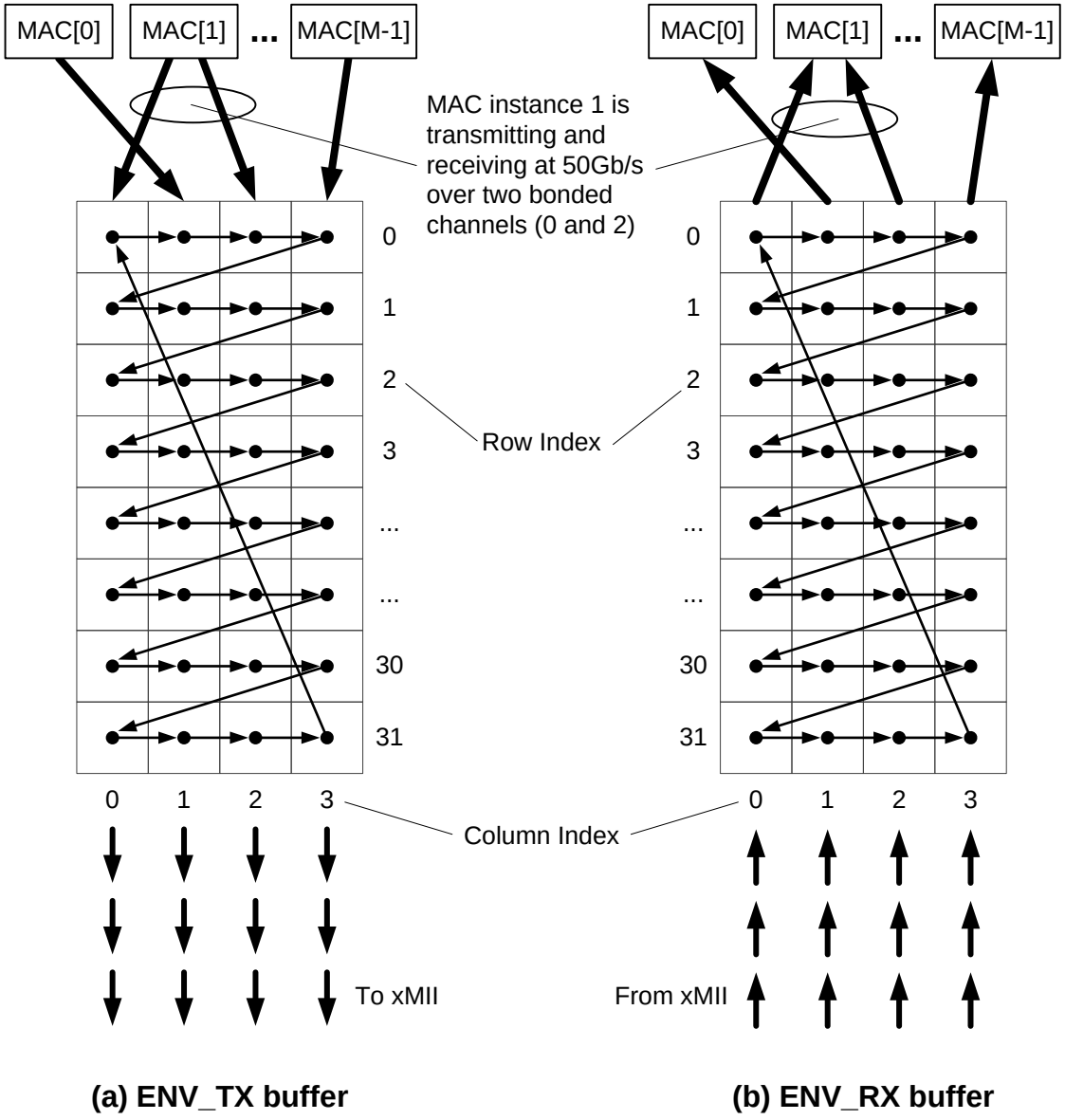

Figure 143.8 Internal structure of ENV_TX and ENV_RX buffers

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 xMII transmit clock (TX_CLK). In the ENV_RX, the read and write pointers advance synchronously with the xMII receive clock (RX_CLK). However, the value of the receive channel write pointer is updated whenever an envelope header is received.

143.2.5.4 Envelope Position Alignment Marker

The relative envelope position recorded in an envelope header by the MCRS transmit function is simply the logical equivalent of 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 MCRS receive function, the EPAM field is extracted and its

value is used to update 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 143.9 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 in the same order as shown in Figure 143.9 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 MCRS 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.

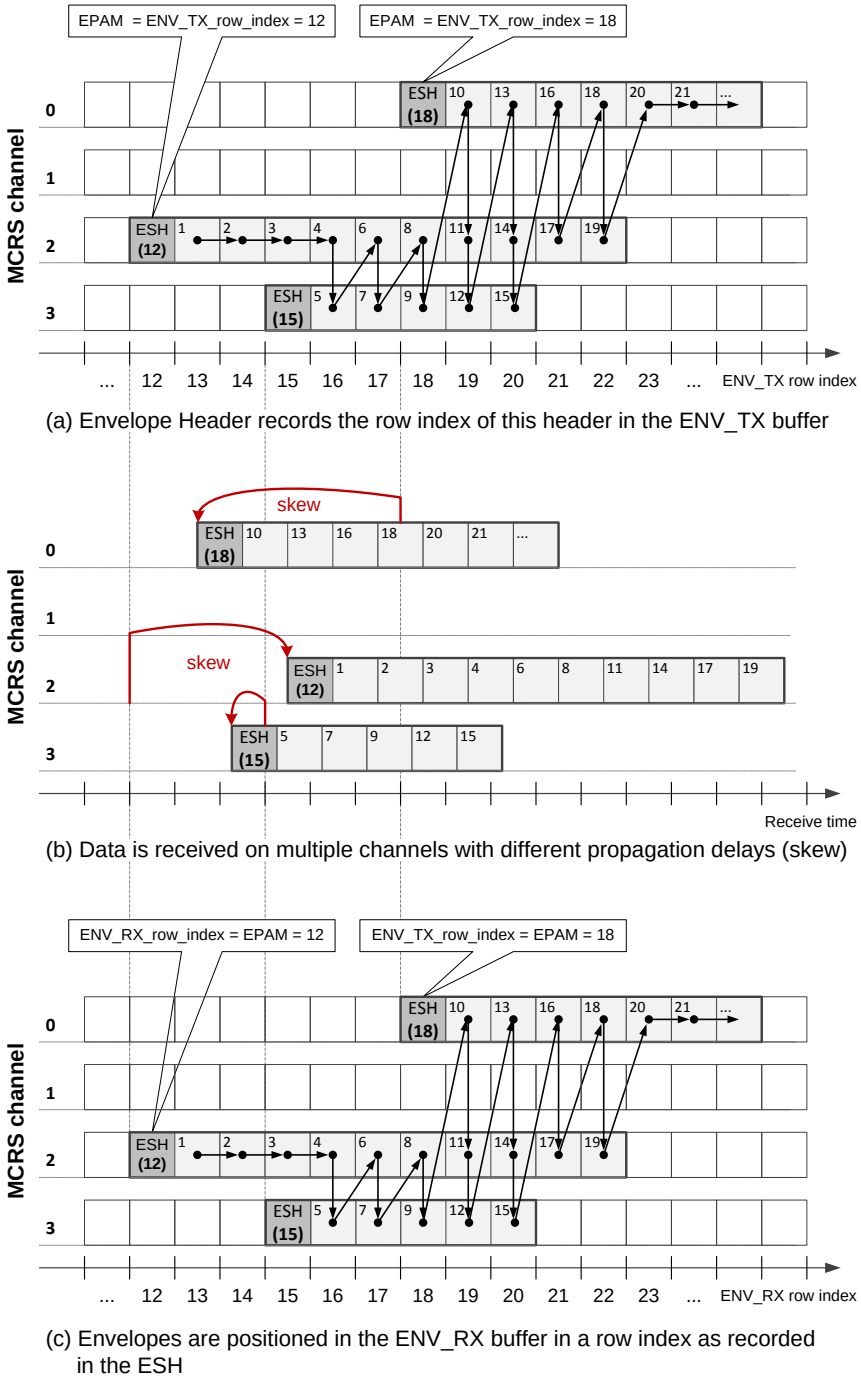

Figure 143.9 Illustration of skew elimination by envelope position alignment in ENV_RX buffer

143.3 MCRS Functional Specifications

143.3.1 MCRS Interfaces

Interfaces to the MCRS are illustrated in Figure *143.10*. In addition to the *M* PLS service interfaces (one per MAC) and *N* xMII instances, there is an MCRS_CTRL interface that connects to the higher layers (see Figure *143.1* and Figure *143.18*).

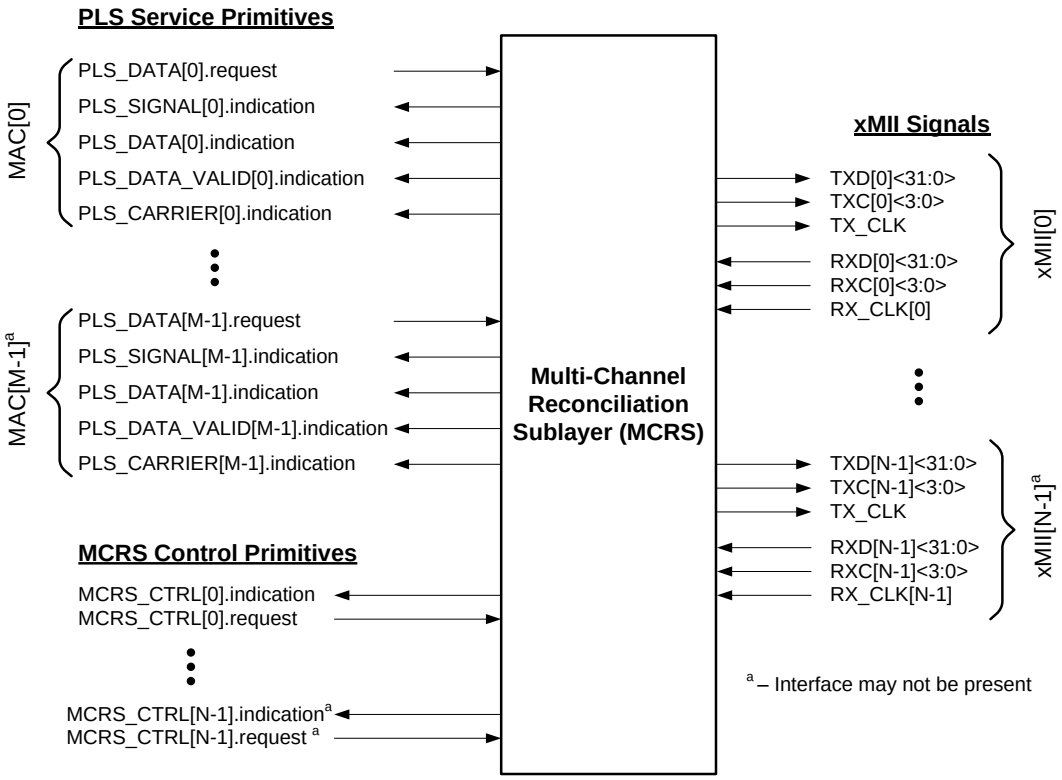

Figure 143.10 Multi-Channel Reconciliation Sublayer (MCRS) inputs and outputs

143.3.1.1 PLS service primitives

In all single channel RSs only one PLS service interface is active at any given moment; this is still true for systems with only one active MCRS channel. However, for systems with more than one MCRS channel there may be multiple PLS service interfaces active at any given time.

The mapping of the PLS service primitives to xMII signals are shown in 143.3.1.1.1 for PLS_DATA[].request primitives and in 143.3.1.1.3 for PLS_DATA[].indications primitives. These are similar to the mappings described in 46.1.7. However, in systems with multiple MCRS channels there are multiple xMIIs and therefore an index is added to the xMII signals to indicate which of the xMIIs to use.

Table 143– **Table 143–2—Mapping of PLS_DATA.request primitives**

MAC operating speed	MCRS channels	Transmit interface	Signals
10 Gb/s	1	XGMII[*0*]	TXD[*0*]<31:0>, TXC[*0*]<3:0> and TX_CLK[1]
25 Gb/s	1	25GMII[*0*]	TXD[*0*]<31:0>, TXC[*0*]<3:0> and TX_CLK[1]
50 Gb/s	2	25GMII[*0*] 25GMII[*1*]	TXD[*0*]<31:0>, TXC[*0*]<3:0> and TX_CLK[1] TXD[*1*]<31:0>, TXC[*1*]<3:0>
Nx25 Gb/s	*N*	25GMII[*0*] 25GMII[*1*] 25GMII[*2*] … 25GMII[*N-1*]	TXD[*0*]<31:0>, TXC[*0*]<3:0> and TX_CLK[1] TXD[*1*]<31:0>, TXC[*1*]<3:0> TXD[*2*]<31:0>, TXC[*2*]<3:0> … TXD[*N-1*]<31:0>, TXC[*N-1*]<3:0>

Table 143– **Table 143–3—Mapping of PLS_DATA.indication primitives**

MAC operating speed	MCRS channels	Receive interface	Signals
10 Gb/s	1	XGMII[*0*]	RXD[*0*]<31:0>, RXC[*0*]<3:0> and RX_CLK[*0*]
25 Gb/s	1	25GMII[*0*]	RXD[*0*]<31:0>, RXC[*0*]<3:0> and RX_CLK[*0*]
50 Gb/s	3	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*]
Nx25 Gb/s	*N*	25GMII[*0*] 25GMII[*1*] 25GMII[*2*] … 25GMII[*N-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*] RXD[*2*]<31:0>, RXC[*2*]<3:0> and RX_CLK[*2*] … RXD[*N-1*]<31:0>, RXC[*N-1*]<3:0> and RX_CLK[*N-1*]

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

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

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

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

The MCRS support full duplex operation only and does not generate the PLS_SIGNAL.indication primitive.

143.3.1.1.3 Mapping of PLS_DATA[ch].indication primitive

The MCRS maps the primitive PLS_DATA.indication to the xMII 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.

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

The MCRS maps the primitive PLS_DATA_VALID.indication to the xMII 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.

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

The MCRS supports full duplex operation only and does not generate the PLS_CARRIER.indication primitive.

143.3.1.2 MCRS Control Primitives

The MCRS inputs the MCRS_CTRL[*ch*].request primitives from the MPCP and outputs to the MPCP the MCRS_CTRL[*ch*].indication primitives.

143.3.1.2.1 MCRS_CTRL[*ch*].request(*link_id, epam, env_length*) primitive

The MPCP requests the MCRS to transmit the next envelope using the MCRS_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 143.3.3.4) is set to the value of *epam* (see *EnvStartHeader()* function definition in 143.3.3.5).

143.3.1.2.2 MCRS_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 MCRS_CTRL[*ch*].indication() primitive. This primitive indicates to the MPCP that the MCRS is available for the next envelope in a given channel. In the absence of an active envelope, the MCRS_CTRL[*ch*].indication() primitive is generated continuously on every IN_CLK transition (see 143.3.3.4). 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. If the MPCP has determined that a transmission opportunity has ended it signals that condition by issuing an envelope with link_id set to 0x00 00.

143.3.1.3 XGMII interfaces

The XMII is specified to support 10 Gb/s operation. The structure of each of the XGMII interfaces in an MCRS system is as specified in 46.1.6.

For mapping between the XGMII signals and the PLS Service interface, see 143.3.1.1.1 and 143.3.1.1.3.

For multi-channel MCRS systems the transmit XGMIIs are synchronous and only one TX_CLK is required.

143.3.1.4 25GMII interfaces

The 25GMII is specified to support 25 Gb/s operation. The structure of each of the 25GMII interfaces in an MCRS 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.

For mapping between the 25GMII signals and the PLS Service interface, see 143.3.1.1.1 and 143.3.1.1.3.

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

143.3.2 Envelope Header format

Each envelope initiated by the MCRS begins with a 72-bit envelope header. The envelope header includes a Start Control Code, an EnvType 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.

When the xMII is 36-bits wide the transmission envelope header, as illustrated in Figure *143.11*, includes two successive transfers over the xMII. Each 36-bit transfer includes four control bits followed by 32 information bits.

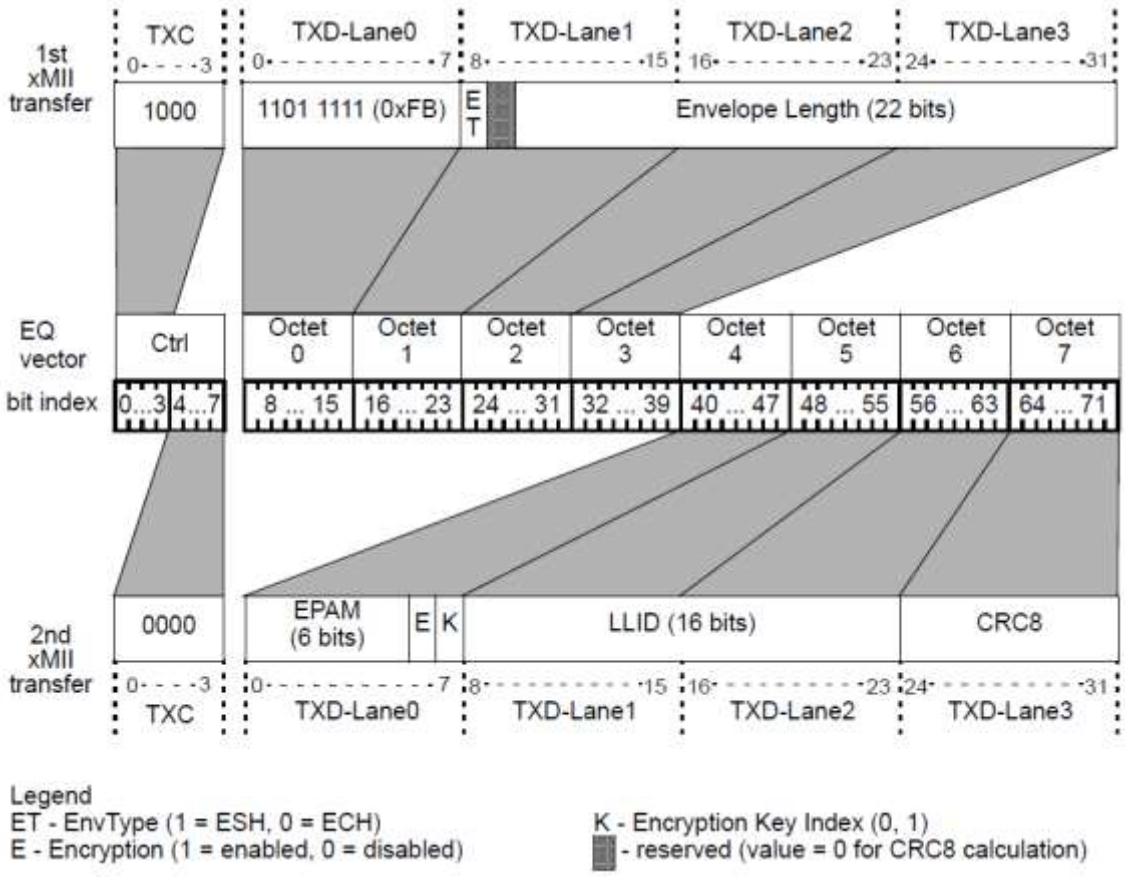

Figure 143.11 Transmission envelope header format

An EQ contains eight octets of information so the length of the envelope header is one EQ. The EPAM is used by the receiving MCRS to remove any timing skew that may have occurred during the transmission of the envelope from the source MCRS to the destination MCRS. 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. 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 1.

Table 1 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	EnvType 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 covering bits 8-63

The envelope start header has the EnvType flag set to one whereas the envelope continuation header has the EnvType 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 header includes the envelope header itself (1 EQ). The envelope continuation header replaces any preambles encountered in the transmission and, in this case, the Envelope Length field includes the envelope continuation header.

143.3.3 Transmit functional specifications

A functional block diagram of the MCRS transmit path is illustrated in Figure *143.12*. The MCRS interfaces are described in 143.3.1. The MCRS transmit path is composed of two processes and one buffer.

The Input Process, described in 143.3.3.6.1, accepts MAC data, formats it into EQs and stores these EQs in the ENV_TX buffer. The Transmit Process, described in 143.3.3.6.2, pulls EQs from the ENV_TX buffer and feeds them to two successive transfers on the appropriate xMII.

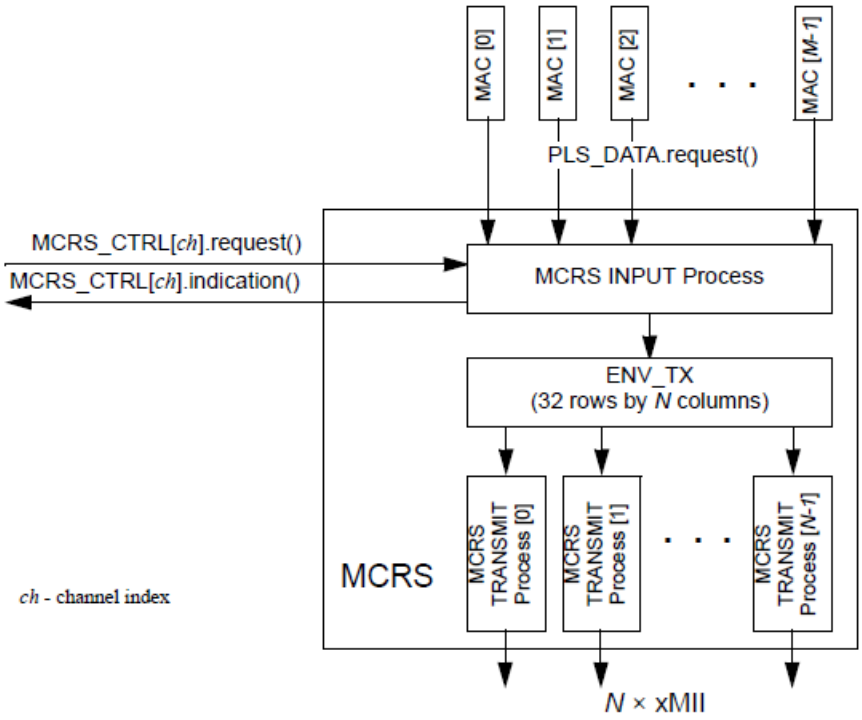

Figure 143.12 MCRS transmit functional block diagram

143.3.3.1 Conventions

143.3.3.2 State diagram conventions

The body of this standard comprises state diagrams, including the associated definitions of variables, constants, and functions. In case of any discrepancies between a state diagram and descriptive text, the state diagram prevails.

The notation ++ after a variable indicates it is to be incremented by 1. The notation – – after a variable indicates it is to be decremented by 1. The notation –= after a variable indicates that the counter value is to be decremented by the following value. The notation += after a variable 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.

The vector notations used in the state diagrams for bit vector use 0 to mark the first received bit and so on (for example data<15:0>), following the conventions of 3.1 for bit ordering.

143.3.3.3 Constants

ADJ_BLOCK_SIZE
TYPE: {TBD}
Value: {TBD}
{ description }

INTER_ENV_IDLE
TYPE: 72-bit vector
Value: 0xFF 08 08 08 08 08 08 08 08
The value of an EQ which represents idle space between transmissions.

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.

RATE_ADJ_EQ
TYPE: 72-bit vector
Value: {TBD}
{ description }

RATE_ADJ_SIZE
TYPE: {TBD}
Value: {TBD}
{ description }

143.3.3.4 Variables

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

BlkLeft[c]
TYPE: {TBD}
{ description }

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 100 Gb/s devices $N = 4$, for 50 Gb/s devices $N = 2$, and for 25 Gb/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 MCRS_CTRL[].request() primitive.

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 143.3.3.5). The variable is incremented after all ENV_TX columns have been read (i.e., once each IN_CLK).

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: 6-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 TX_CLK signal for channel *c* (see Table 143-2).

TxActive[c]

TYPE: Boolean

{ description }

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 xMII interface.

wRow

TYPE: 6-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.

143.3.3.5 Functions

EnvContHeader(wCol)

The *EnvContHeader()* function returns a new envelope header with the EnvType 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];                //EnvLength
     hdr<40:45> = EnvPam;                      //EPAM
     hdr<48:63> = LinkId[col];                 //LLID
     hdr<64:71> = CRC8(hdr<8:63>);             //Calculate CRC8
     return hdr;
}
```

EnvStartHeader(wCol, epam)

The *EnvStartHeader()* function returns a new envelope header with the EnvType 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 MRPR_CTRL[].request primitive.

```
EnvStartHeader(int2 col, int5 epam)
```

```
{
     EQ hdr;
     // Use provided 'epam' value if this envelope starts a new burst
     if( !TxActive[col+1] &&
         !TxActive[col+2] &&
            …
         !TxActive[col+N-1]) 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<8:63>); //Calculate CRC8
     return hdr;
}
```

GetFillerEQ (wCol)

```
EQ GetFillerEQ( wCol )
{
     if( TxActive[wCol] )
        return IEI_EQ;      //Inter-Envelope Idle
     else
        return IBI_EQ;      //Inter-Burst Idle
}
```

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 PLS_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 values between 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])
    if( link_id == 0x00-00 )
        return IBI_EQ;      // Inter-burst Idle

    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;
    }
```

143.3.3.6 State Diagrams

143.3.3.6.1 Input Process

The MCRS 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 an envelope start 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 (all 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 are 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.

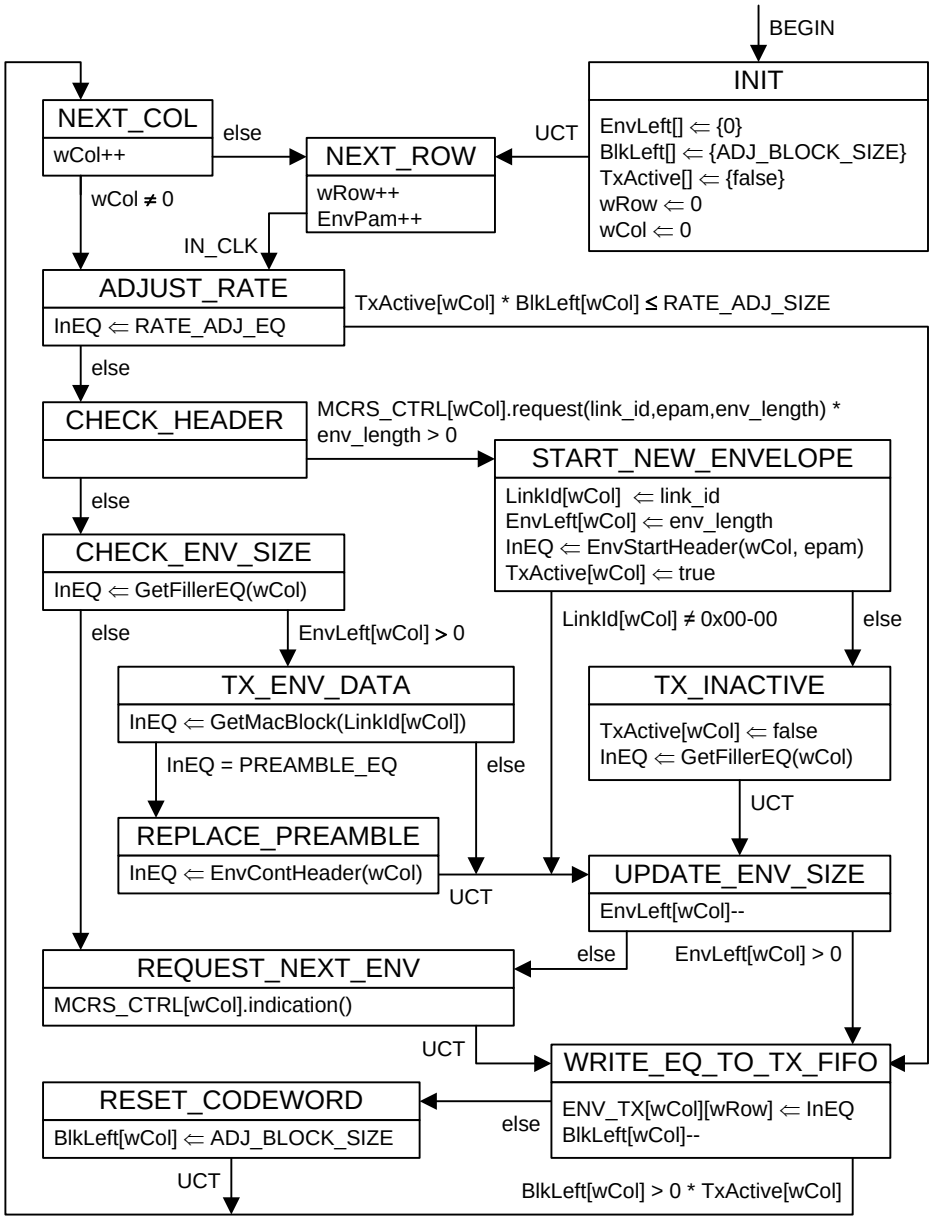

Figure 143.13 MCRS Transmit Function, Input Process state diagram

143.3.3.6.2 Transmit Process

The MCRS Transmit Process shall implement the state diagram as depicted in Figure *143.14*. One instance of the state diagram is instantiated for each xMII.

The Transmit Process outputs one 36-bit vector (TXD[*ch*]<31:0> + TXC[*ch*]<3:0>) to its associated xMII interface on each edge of the TX_CLK 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_CLK.

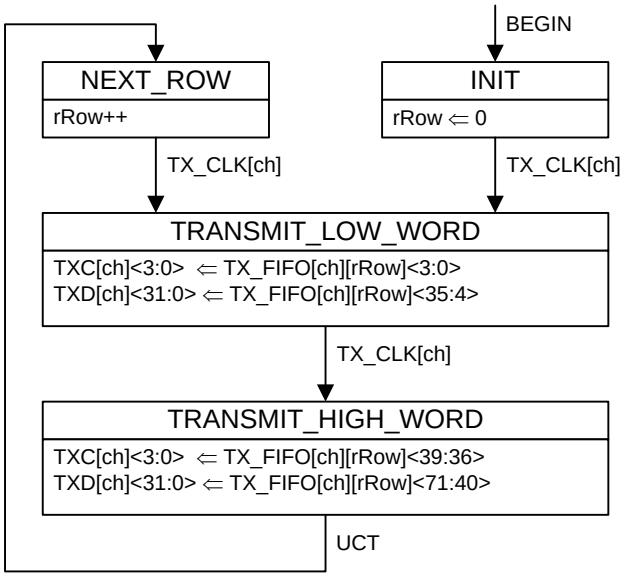

Figure 143.14 MCRS Transmit Function, Transmit Process state diagram

143.3.4 Receive functional specifications

A functional block diagram of the MCRS receive path is illustrated in Figure *143.15*. The MCRS interfaces are described in 143.3.1. The MCRS receive path is composed of two processes and one buffer. The Receive Process, described in greater detail in 143.3.4.5.1, accepts two successive transfers from the associated xMII 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 143.3.4.5.2, pulls EQs from the ENV_RX buffer and feeds them to the appropriate MAC as specified by the current LLID for that receive channel.

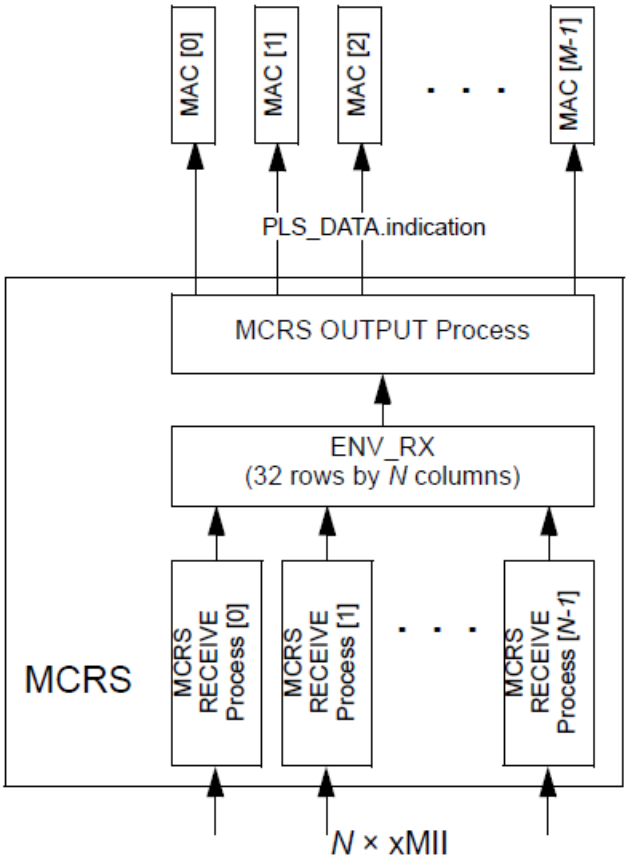

Figure 143.15 MCRS receive functional block diagram

143.3.4.1 Conventions

See 143.3.3.1.

143.3.4.2 Constants

ES_HEADER
: TYPE: integer
Value: 1
The value of the envelope EnvType flag indicating the header is an envelope start header.

INTER_ENV_IDLE
: See 143.3.3.3.

RATE_ADJ_EQ
: See 143.3.3.3.

PREAMBLE_EQ
: See 143.3.3.3.

143.3.4.3 Variables

ch
: See 143.3.3.4.

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 100 Gb/s devices *N* = 4, for 50 Gb/s devices *N* = 2, and for 25 Gb/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.

EnvLeft[c]

See 143.3.3.4.

LinkId[c]

See 143.3.3.4.

OUT_CLK

TYPE: Boolean

The *OUT_CLK* clear on read variable is set to True on each positive edge of TX_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 xMII 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 xMII channel *c*.

RxEQ

TYPE: 72-bit binary

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

143.3.4.4 Functions

IsHeader(eq)

The *IsHeader(eq)* function returns true if the parameter eq represents an envelope header. An envelope header 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
```

```
}
```

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 INTER_ENV_IDLE
    eq<71:40> == 0x08080808 ) // … s.b. INTER_ENV_IDLE
    OR
    ( eq<39:36> == 0x8 AND // Misaligned Env. Header
    eq<47:40> == 0xFB )); // … s.b. Start Control Code
}
```

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
     }
}
```

SetMacOctet(link_id, rx_data, data_valid)

This function shifts eight bits in *rx_data* to the MAC using the PLS_DATA.indication, along with the *data_valid* Boolean using the PLS_DATA_VALID.indication to the MAC associated with *link_id*.

143.3.4.5 State Diagrams

143.3.4.5.1 Receive Process

The MCRS Receive Process shall implement the state diagram as depicted in Figure *143.16*.

This process forms an EQ from two successive xMII 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 delay and transport skew, a separate instance of the Receive Process is required for each channel implemented.

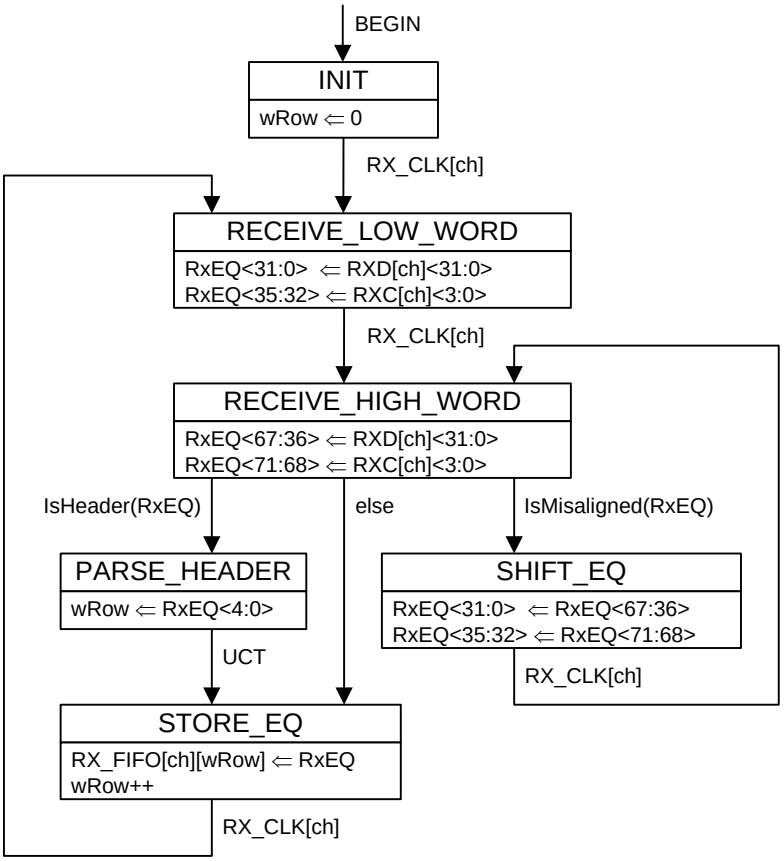

Figure 143.16 MCRS Receive Function, Receive Process state diagram

143.3.4.5.2 Output Process

The MCRS 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 are lost due to the error since the next envelope continuation header resynchronizes the process for the following frame.

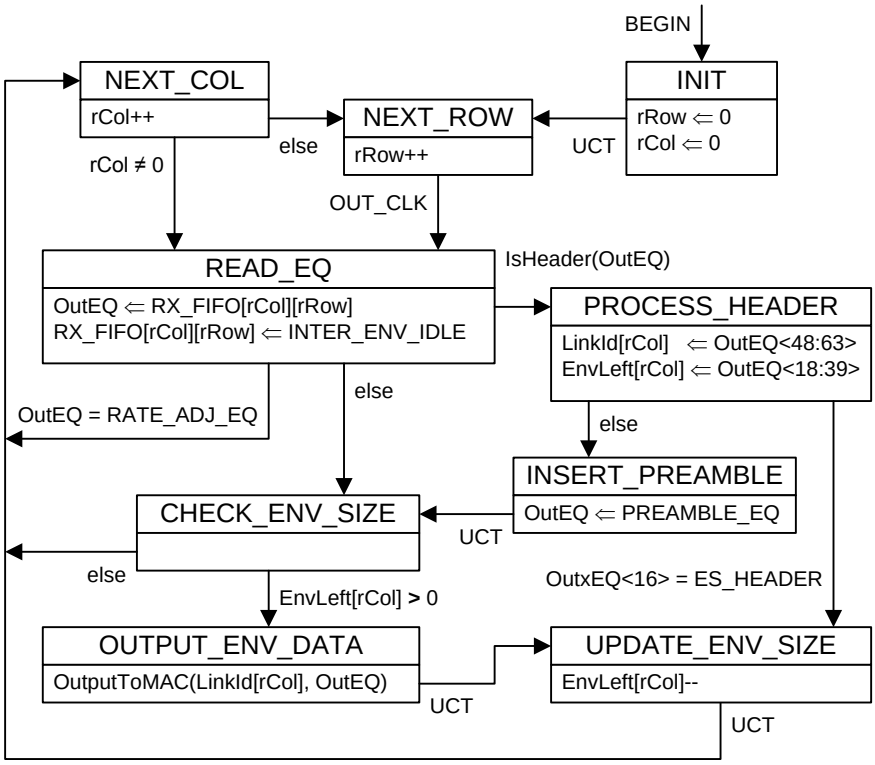

Figure 143.17 MCRS Receive Function, Output Process state diagram

143.4 Nx25G-EPON MCRS Requirements

143.4.1 Nx25G-EPON architecture

This subclause describes the MCRS requirements for Nx25G-EPON point-to-multipoint (P2MP) networks. These are passive optical multipoint networks (PONs) that connect multiple optical network units (ONUs) to a single optical line terminal (OLT). The architecture is asymmetric, based on a tree and branch topology utilizing passive optical splitters.

A transmission direction from the OLT towards the ONUs is referred as the downstream direction and transmission direction from an ONU toward the OLT is referred as the upstream direction.

The MCRS is used with Nx25G-EPON point-to-multipoint (P2MP) networks in order to interface multiple MAC instances with one or two 25GMII channels in each direction. Figure 143-x illustrates the relationship of the MCRS and the OSI protocol stack for Nx25G–EPON.

Nx25G-EPON OLT and ONU PMDs are defined in Clause 141, with the respective Nx25G-EPON PCS defined in Clause 142

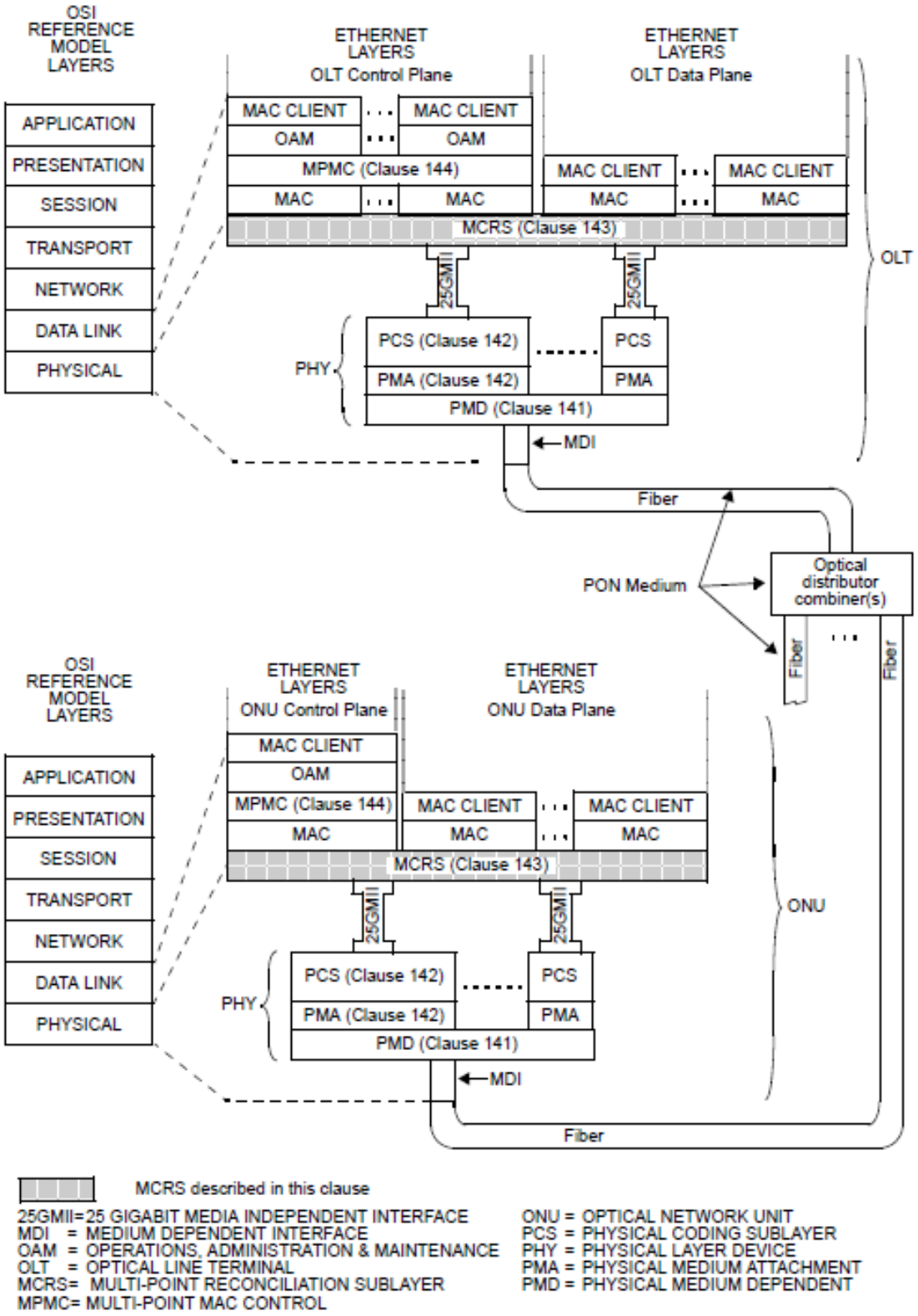

Figure 143.18 Relationship of EPON P2MP PMD to the ISO/IEC OSI reference model

The MCRS in Nx25G-EPON architecture serves as an interfaces sublayer between the MAC sublayer and 25GMII. 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.

143.4.1.1 MCRS channels

An MCRS channel that carries information from the OLT to the ONU is referred to as the downstream channel (DC), and the channel that carries information from an ONU to the OLT is referred to as the upstream channel (UC).

The 25/10G-EPON and 25G/25G-EPON architectures shall implement a single MCRS channel in each direction. The 50/10G-EPON and 50/25G-EPON architectures shall implement two MCRS channels in the downstream direction and a single channel in the upstream direction. The 50/50G-EPON architecture shall implement two channels in each direction. When two channels are implemented in the same direction, channel bonding of these two channels shall be supported. Table 2 summarizes MCRS channels for Nx25G-EPON.

Each MCRS channel is bound to a separate PCS instance via a separate xMII instance. Channels operating at 25 Gb/s are bound to 25GMII, whereas the channel operating at 10 Gb/s is bound to XGMII instance. Thus, for any given system, there is a one-to-one correspondence between the MCRS channel count and the number of xMII instances supported.

Table 2 MCRS channel designation and capabilities

Designation	MCRS Channel	MCRS Channel Function
DC0	Downstream channel 0	All ONUs receive this MCRS channel, broadcast, ONU discovery
DC1	Downstream channel 1	Only ONUs capable of receiving at 50 Gb/s support this MCRS channel.
UC0	Upstream channel 0	All ONUs transmit on this MCRS channel, ONU discovery
UC1	Upstream channel 1	Only ONUs capable of transmitting at 50 Gb/s support this MCRS channel.

143.4.1.2 Symmetric and Asymmetric Data Rates

The Nx25G-EPON architecture supports symmetric and asymmetric data rates. The symmetric data rate systems include 25/25G-EPON or 50/50G-EPON. The asymmetric rate systems include 25/10G-EPON or 50/10G-EPON, and 50/25G-EPON.

A distinction is made regarding the underlying mechanisms of achieving the asymmetric data rates. In 25/10G-EPON systems, the asymmetric data rate is achieved via the MCRS channel *rate* asymmetry, were a single downstream MCRS channel DC0 operates at 25 Gb/s and a single upstream MCRS channel UC0 operates at 10 Gb/s. Additional details for MCRS implementations supporting the channel rate asymmetry are provided in 143.4.4.

In 50/25G-EPON systems, the asymmetric data rate is achieved via the MCRS channel number asymmetry, were two MCRS channels are active in the downstream direction (DC0 and DC1), but only a single MCRS channel UC0 is active in the upstream direction. Note that every upstream and downstream MCRS channels operate at the data line rate of 25 Gb/s.

Both the channel rate asymmetry and the channel number asymmetry mechanisms can be combined, as is the case in 50/10G-EPON systems, where there are two downstream MCRS channels operating at 25 Gb/s and a single upstream MCRS channel operating at 10 Gb/s.

An Nx25G-EPON system may serve ONUs that support different numbers of MCRS channels (see 143.4.1). Therefore, some ONUs are only able to receive and transmit data on MCRS channels DC0 and UC0, some are able to transmit on DC0 and DC1 and transmit on UC0 and UC1 or just on UC0.

143.4.2 MCRS and MPCP clock synchronization

EDITORS NOTE: general description of clocks used in MCRS and their relationships, It might be a good idea to split this into three sub-section; common, OLT, and ONU

143.4.3 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 MCRS 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 "TBD EQ". In Cl 76.1.2 this applied to the combined MCRS, PCS, & PMA. A revised value is needed.

143.4.4 Channels with asymmetric rates

143.4.4.1 Mapping of 25GMII and XGMII primitives at the OLT

143.4.4.2 Mapping of 25GMII and XGMII primitives at the ONU

143.4.4.3 MCRS channel operation at 10 Gb/s

143.4.4.3.1 Changes to Input Process

143.4.4.3.2 Changes to Transmit Process

143.4.4.3.3 Changes to Receive Process

143.4.4.3.4 Changes to Output Process

143.5 Protocol implementation conformance statement (PICS) proforma for Clause 143, Multi-Channel Reconciliation Sublayer for Nx25G-EPON[2]

143.5.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Clause 143, Multi-Channel Reconciliation Sublayer for Nx25G-EPON, shall complete the following protocol implementation conformance statement (PICS) proforma.

A detailed description of the symbols used in the PICS proforma, along with instructions for completing the PICS proforma, can be found in Clause 21.

143.5.2 Identification

143.5.2.1 Implementation identification

Supplier	
Contact point for inquiries about the PICS	
Implementation Name(s) and Version(s)	
Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s)	
NOTE 1—Only the first three items are required for all implementations; other information may be completed as appropriate in meeting the requirements for the identification. NOTE 2—The terms Name and Version should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).	

143.5.2.2

143.5.2.3 Protocol summary

Identification of protocol standard	IEEE Std 802.3-201x, Clause 143, Multi-Channel Reconciliation Sublayer for Nx25G-EPON
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	
Have any Exception items been required? No [ ] Yes [ ] (See Clause 21; the answer Yes means that the implementation does not conform to IEEE Std 802.3-2015.)	
Date of Statement	

[2]*Copyright release for PICS proformas:* Users of this standard may freely reproduce the PICS proforma in this subclause so that it can be used for its intended purpose and may further publish the completed PICS.

143.5.2.4

143.5.2.5 Major capabilities/options

Item	Feature	Subclause	Value/Comment	Status	Support

143.5.3