MPRS Enhancement for Improved Error Handling

Glen Kramer, Broadcom Ltd.
Duane Remein, Huawei Technologies
Heaven Gaobo, Huawei Technologies
Jean-Christophe Marion, Tibit Communications

10G-EPON FEC Error Handling

The FEC decoding process shall provide a user option to indicate an uncorrectable FEC codeword (due to an excess of symbols containing errors) to higher layers. If this option is set to be true, the FEC decoding process checks for the value of decode_success. ***If the variable decode_success is set to false, then each sync header of the received payload blocks in the FEC codeword is set to a value of binary 00.*** However, the data blocks are nevertheless passed to the descrambler to maintain descrambling synchronization. When this option is set to FALSE and decode_success is FALSE then each received payload block is passed unchanged.

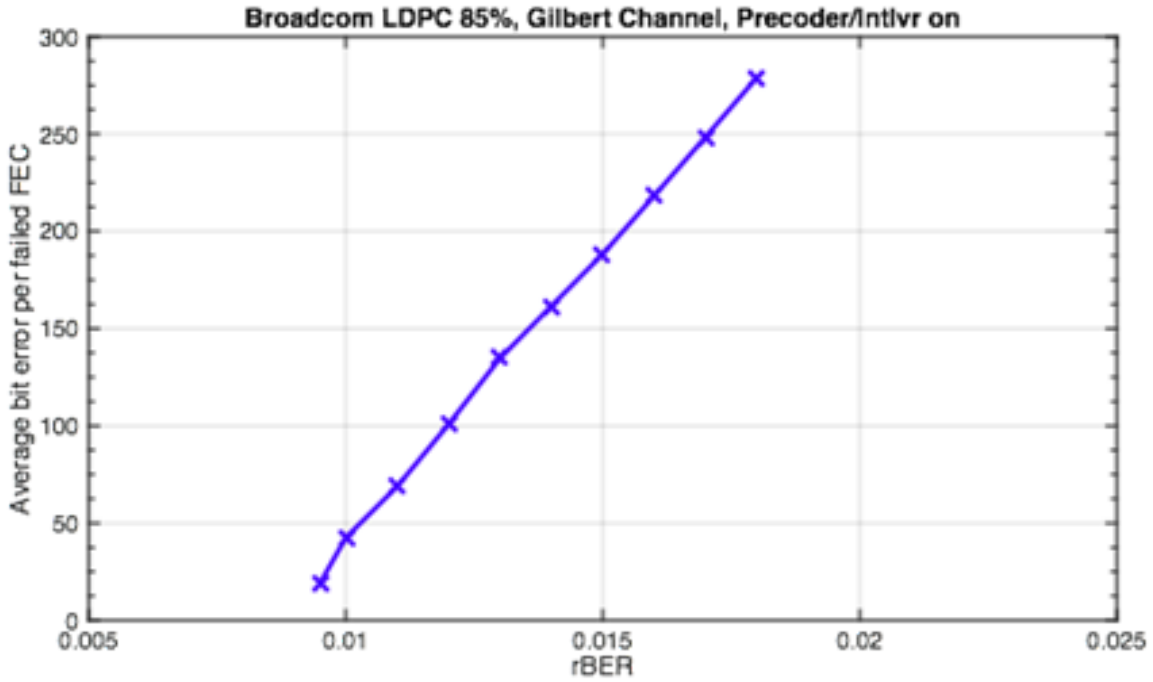
```
Read_outbuffer[i]
{
   int offset = 29+i*65
   for(j=0, j<65, j++)
      rx_coded_corrected<j+1> = outbuffer[j+offset]

   if (!decode_success AND mark_uncorrectable)
   {
      rx_coded_corrected<0> = 0
      rx_coded_corrected<1> = 0
   }
   else
      rx_coded_corrected<0> = !rx_coded_corrected<1>
   BlockToDescrambler()
}
```

What should we do in 802.3ca?

- Invalidating an entire uncorrectable FEC Codeword has even larger impact than in 10G-EPON:
  - ~2000 bytes vs. 216 bytes
  - If the codeword contains an envelope header, an entire envelope can be lost (up to 2^{22} EQs or 33 MB), even if the header did not have any bit errors.
  - If LLID's transmission is striped across 4 channels, invalidating a codeword with an envelope header on one channel will cause loss of all 4 envelopes.

Uncorrectable FEC Codewords

- FEC method will guarantee that input BER of ~10^{-2} gets reduced to output BER of 10^{-12}.
- Even if input errors are distributed uniformly (AWGN), output bit errors are clumped together within a FEC codeword
  - Most FEC codewords will come out from FEC decoder error-free
  - Uncorrectable codewords will have <u>many</u> residual errors



Corrupted Envelope Header

- In an uncorrectable FEC codeword, bit errors may fall on envelope headers.

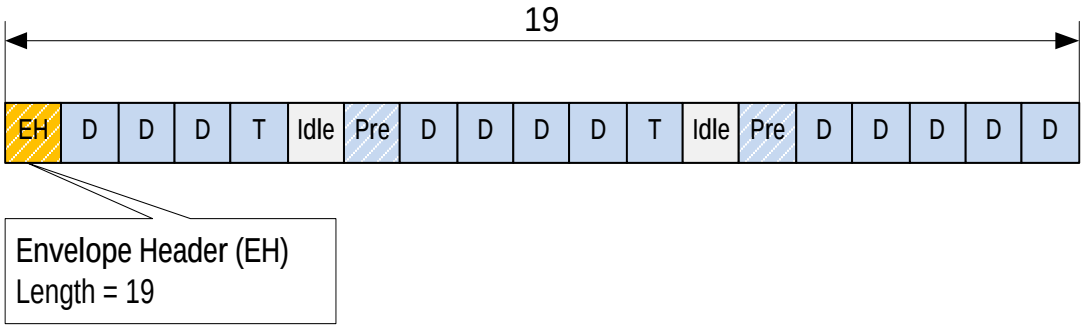
- False negatives are possible
  - Envelope headers may become corrupted in several ways:
    - Corrupted SH may make an envelope header appear as a data EQ (1 bit error)
    - Corrupted block type may make an envelope header appear like a termination block (4-bits errors)

- False positives are not likely due to use of CRC8
  - Non-header looking like a header
  - An envelope header looking like a different envelope header (undetected errors)

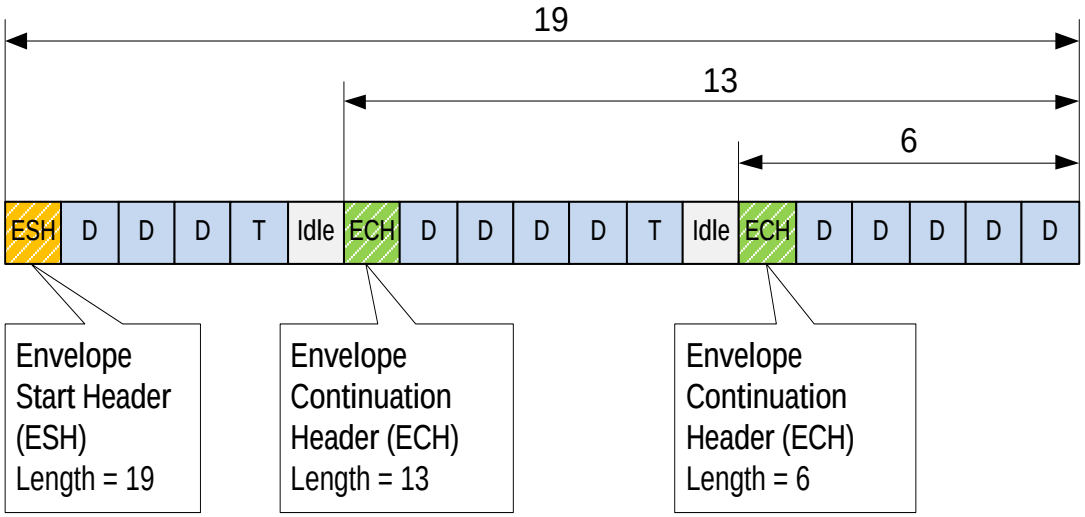
Proposal

- Define two types of Envelope Headers:
  - **Envelope Start Header (ESH)**
    - Inserted by MPRS at the beginning of every envelope
    - Shows length of entire envelope, including the ESH
    - Behaves just like EH in D0.4
  - **Envelope Continuation Header (ECH)**
    - Replaces the frame preamble
    - Shows the residual length of the envelope, including the ECH itself.

Existing Envelope Structure (D0.4)

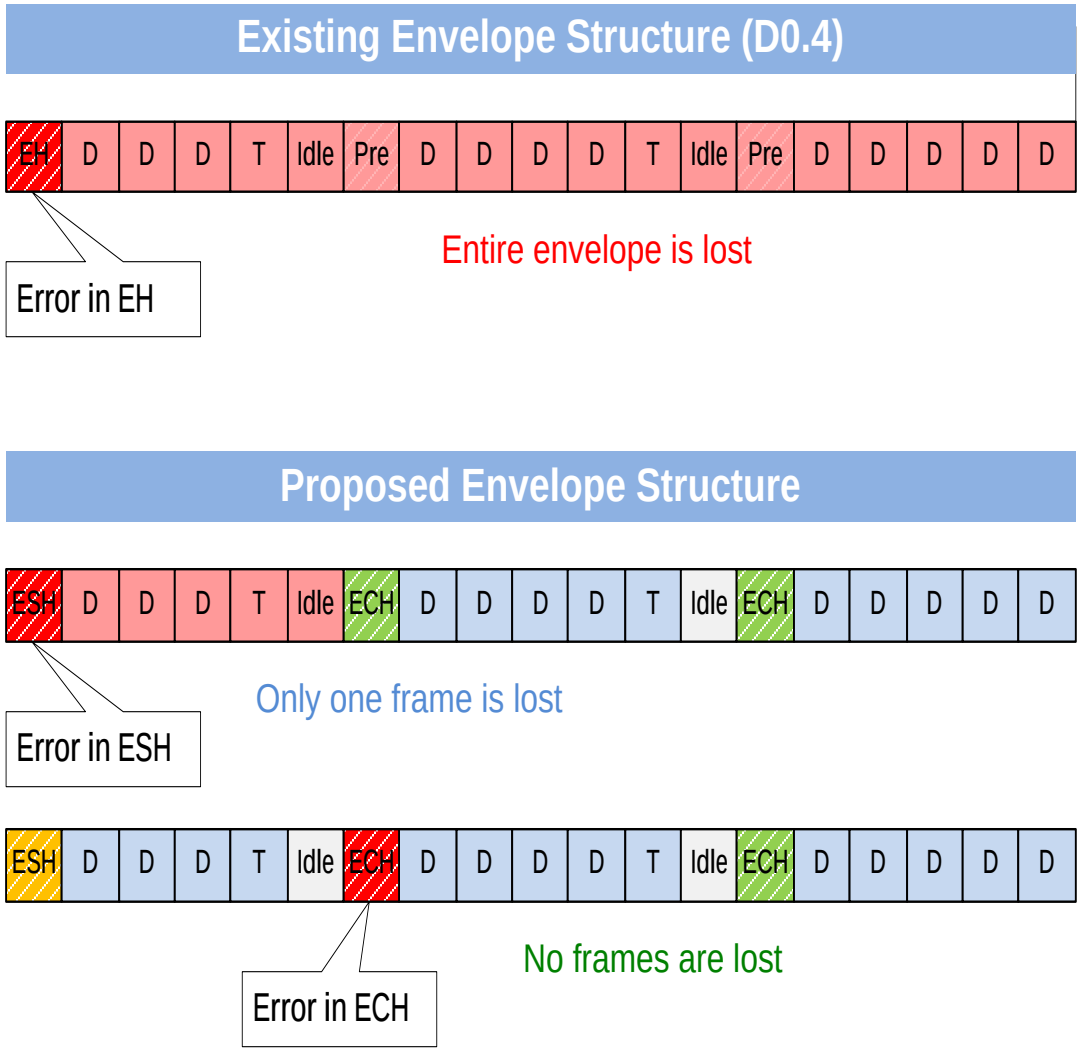


Proposed Envelope Structure



Error Handling Improvements

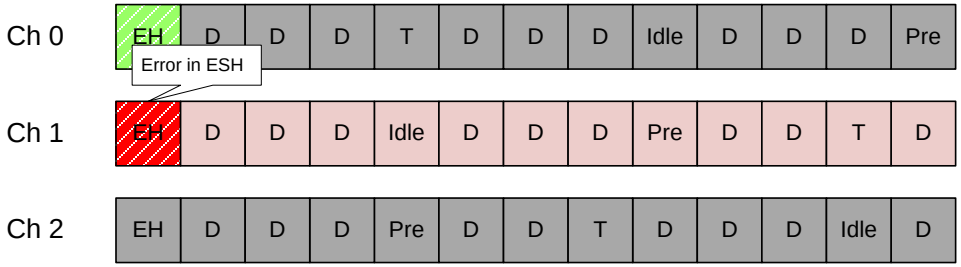
- In current version, an error in envelope header results in loss of entire envelope
- In new proposal, the ECH is an additional opportunity for error recovery
  - Loss of ESH results in loss of at most one frame
  - Loss of ECH results in no data loss (previous envelope header info remains valid)



Error Handling in Multi-Channel Case

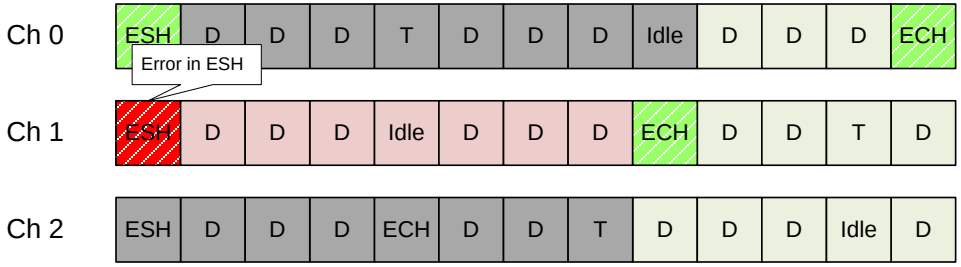
- In current version, an error in envelope header results in loss of multiple envelopes
- In new proposal, multiple frames may be lost, if the error happen in the ESH on one of the channels. But the data will recover when the next ECH is encountered on the affected channel.

Existing Envelope Structure (D0.4)

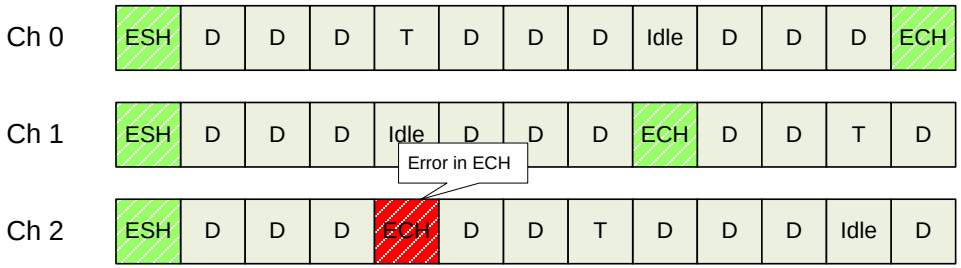


Entire Envelope is Lost

Proposed Envelope Structure



Only two frames are lost



No frames are lost

Definitions and Modifications to State Diagrams

Definitions (1/2)

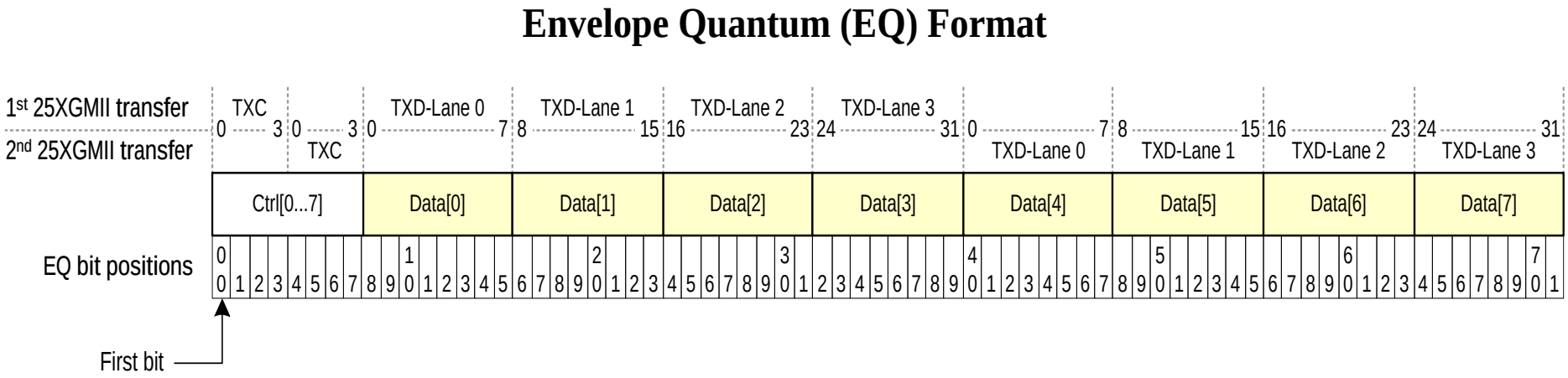
- **Grant** – a transmission opportunity allocated to a specific LLID. A grant is issued by the OLT and is delivered to an ONU in a GATE MPCPDU.
- **Logical Link ID (LLID)** – a collective term that refers to *Physical Layer ID (PLID)*, *Management Link ID (MLID)*, *User Link ID (ULID),* or a *Group Link ID (GLID)*.
- **Envelope** – a continuous transmission by a specific LLID (excluding GLID) on a specific channel. An envelope starts with an Envelope Start Header (ESH) and continues uninterrupted for the number of EQs represented by the EnvLength parameter.
- **Envelope Descriptor** – a tuple that includes LLID, Start Time, and Length parameters that describe a specific envelope. Envelope Descriptors are generated by the local MPCP sublayer and are passed to MPRS at appropriate time to start the envelope transmission.

Definitions (2/2)

- **Envelope Header** – a special marker that is inserted at the beginning of every envelope (Envelope Start Header) and in place of every frame preamble (Envelope Continuation Header)

Envelope Quantum (EQ) Format

- Envelope Quantum is a 72-bit vector representing two 25GMII transfers

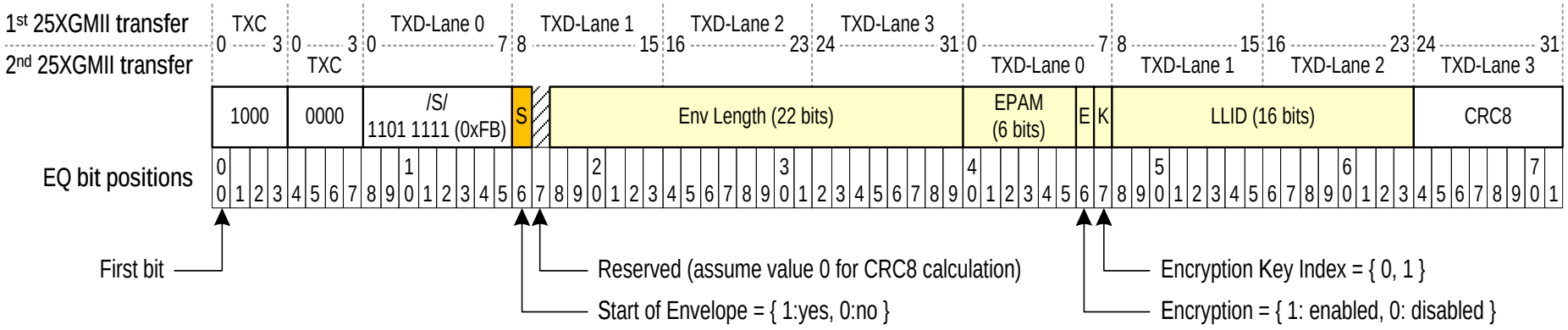


- Logically, EQ consists of an array of 8 control bits (Ctrl[0…7]) and an array of 8 data octets (Data[0…7])

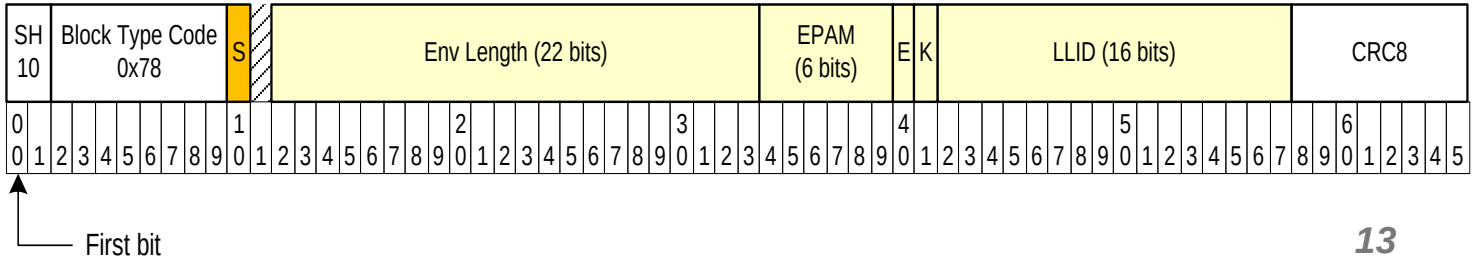
Envelope Header Format

- Both ESH and ECH have identical formats
- The only difference is the value of the ***Start*** flag (replaces Preamble flag)
  - In ESH, Start = 1 (true)
  - In ECH, Start = 0 (false)

Envelope Header (EQ format)



66-bit encoding of Envelope Header



Summary of Proposed Changes

1. Envelope length includes the envelope header. Therefore, an MPRS_CTRL.request() from MPCP with env_length = 0 is considered a malformed request and is ignored by MPRS Input Process.

2. Any time a preamble is read from the MAC, it is replaced in MPRS Input Process by a Envelope Continuation Header (ECH)

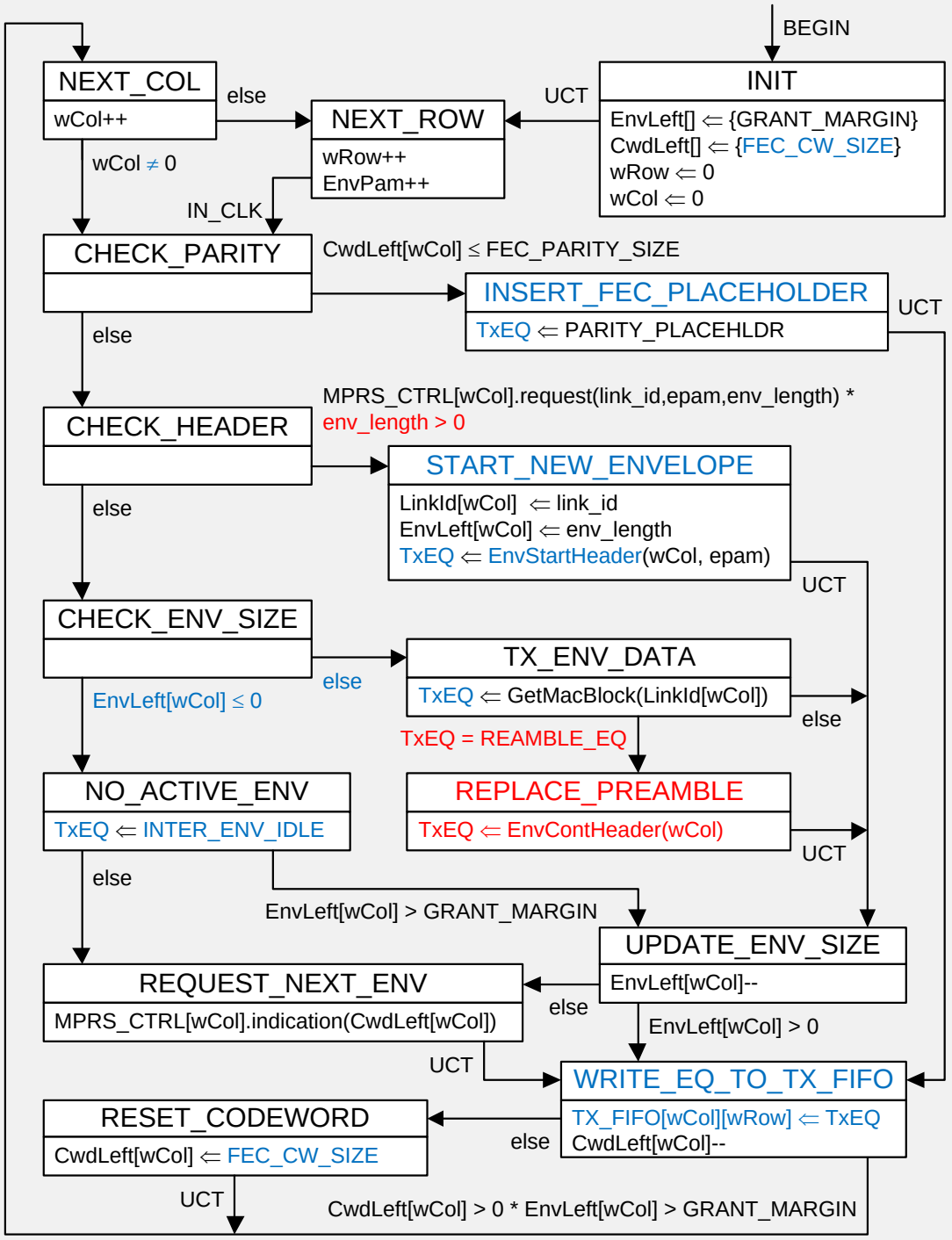
3. Any time a ECH is encountered by the MPRS Output Process, it is replaced by a normal frame preamble before the EQ is passed to the MAC.

MPRS Input Process

- Function **GetEnvHeader(…)** is replaced by two functions:
  - **EnvStartHeader(…)** returns the value of ESH
  - **EnvContHeader(…)** returns the value of ECH

- A new state **REPLACE_PREAMBLE** is added

- **Red text** – new behavior, technical change
- **Blue text** – no change in behavior vs. D0.4, editorial change



EnvStartHeader(…) function

EnvStartHeader(int2 col, int6 epam) function creates a new envelope header with *Start* flag equal 1, indicating that it is a start of a new envelope. If this envelope starts a new burst, update EnvPam to the value provided by MPCP ('epam').

```
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;
 }
```

EnvContHeader(…) function

EnvContHeader(int2 col) function creates a new envelope header with *Start* flag equal 0, indicating that it is a continuation of the current envelope. This function never updates EnvPam.

```
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<0:63>);     //Calculate CRC8

    return hdr;
 }
```

GetMacBlock(...) function

GetMacBlock(int16 link_id) function retrieves 64 bits from a MAC instance identified by link_id parameter and returns them as an envelope quantum (see Figure x-xx). If the retrieved bits contain partial frame preamble, the preamble is shifted forward such that the entire preamble is returned in one EQ. The function behavior is represented by the following code:

```
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])

    for( octet_index = 0; octet_index < 8, octet_index++ )
    {
        tx_data = GetMacOctet( link_id );              // Get 8 bits from MAC
```

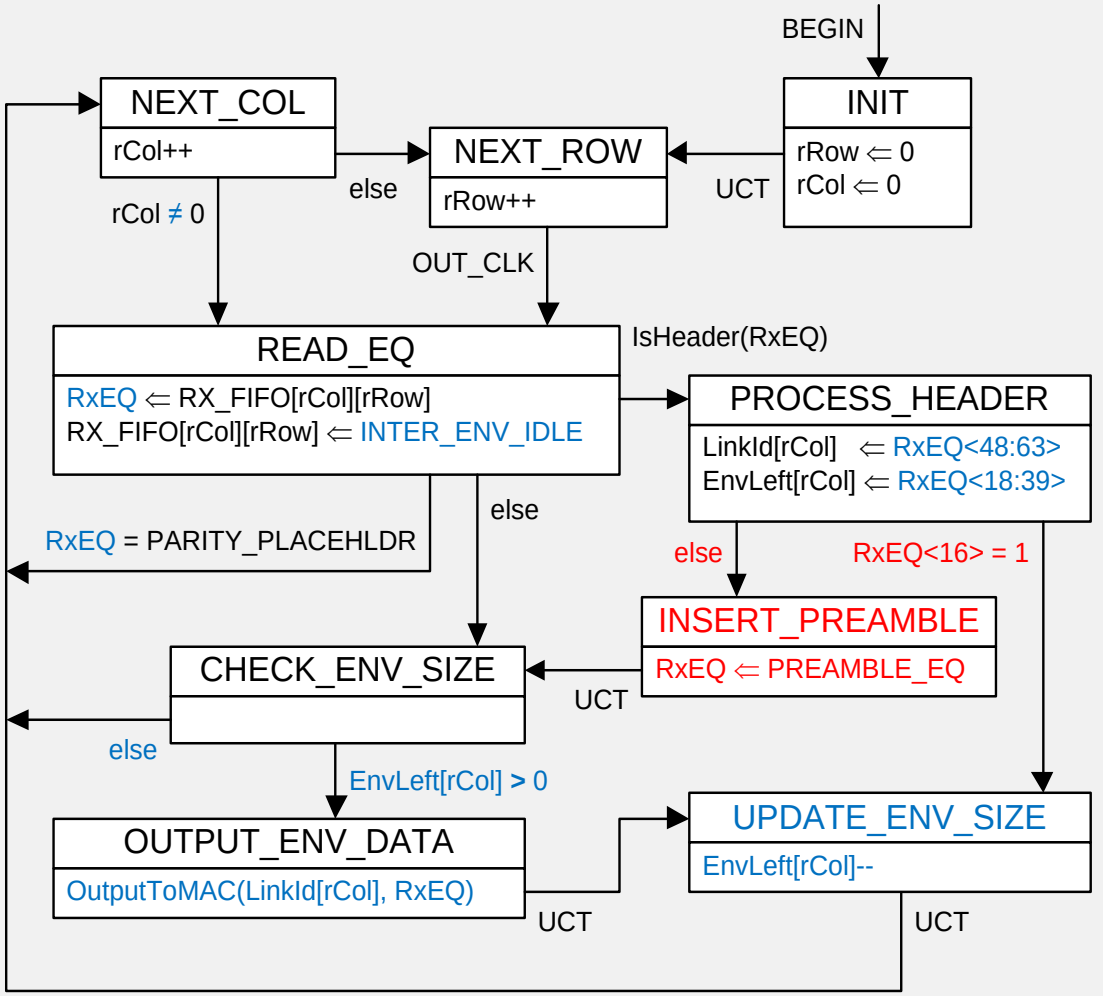
Continued on the next slide…

GetMacBlock(...) function

```
…
        if( IsIdle(tx_data) AND !IdleFlag[link_id] )    // The first 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] )                    // The first 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 regular Data octet
            eq.Data[octet_index] = tx_data;
        }
    }
    return eq;
}
```

MPRS Output Process

- Added state INSERT_PREAMBLE
  - Replaces ECH with a regular preamble before passing RxEQ to MAC



- **Red text** – new behavior, technical change
- **Blue text** – no change in behavior vs. D0.4, editorial change

OutputToMac(...) function

OutputToMac(int16 link_id, EQ eq) function transfers 64 bits of data given in the parameter ‘eq’ to the MAC instance identified by the parameter ‘link_id’.

```
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 )    // Receive /S/ control character
        {
            data_valid = true;
            rx_data = 0x55;                         // Replace /S/ with preamble byte
        }
        else                                        // Receive any 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
    }
}
```

D0.4 Input Process SD Comparison

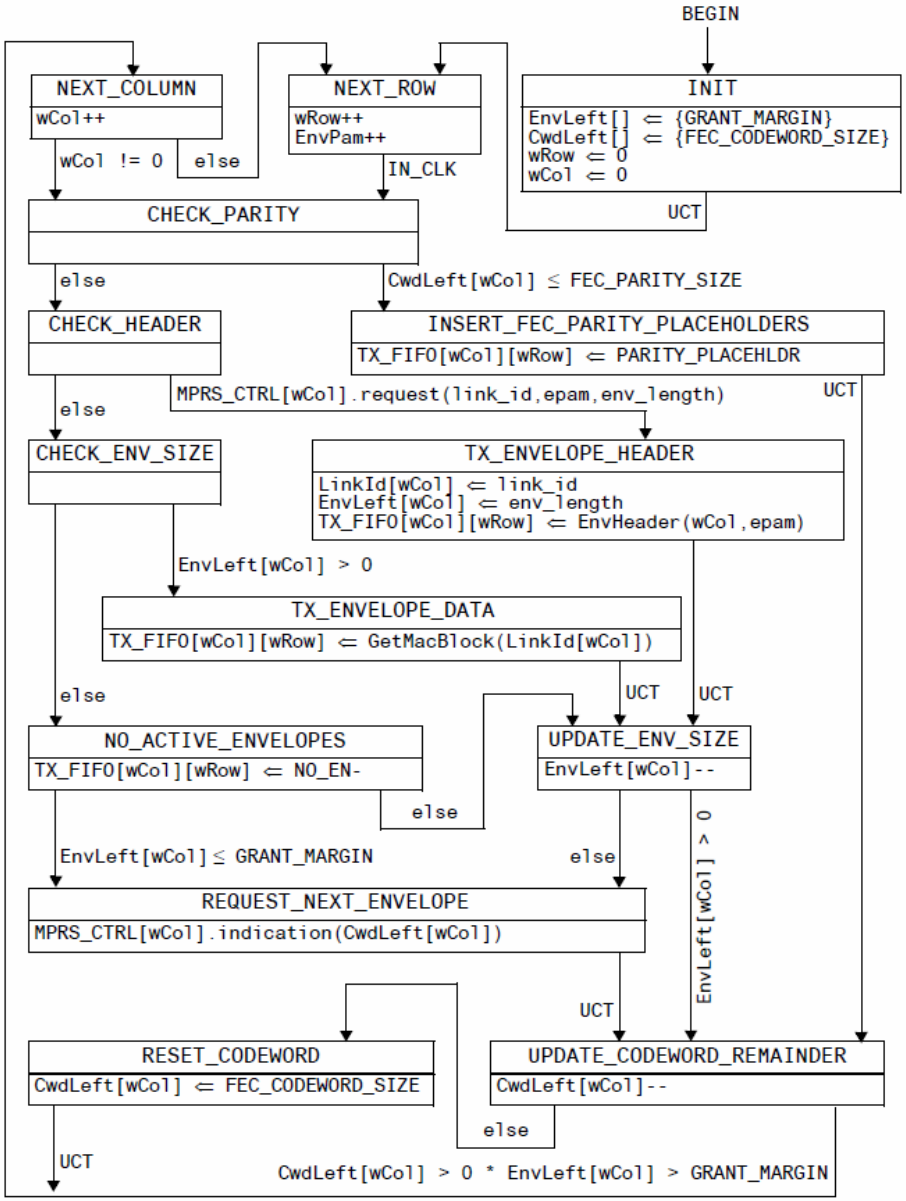
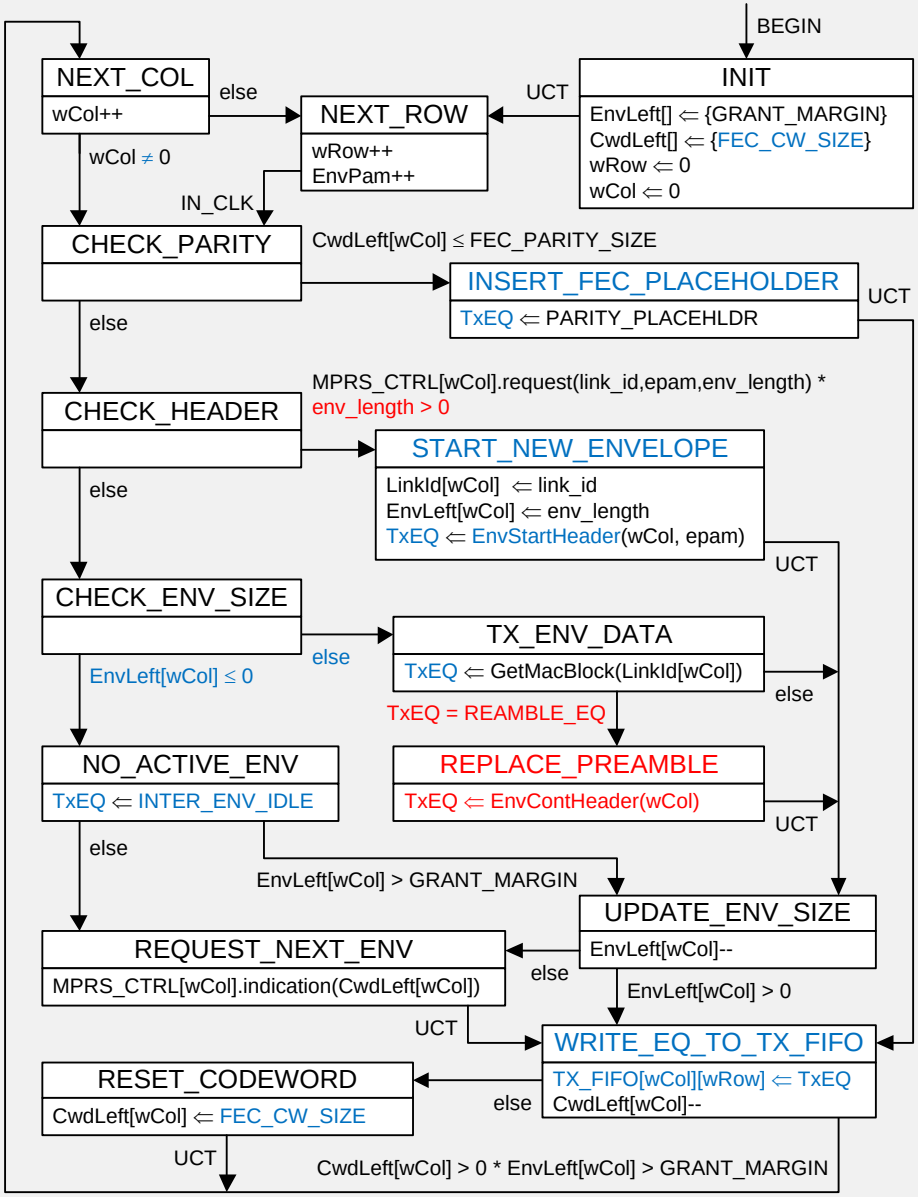


Figure 143–1—ONU MPRS Input Process state diagram



D0.4 Output Process SD Comparison

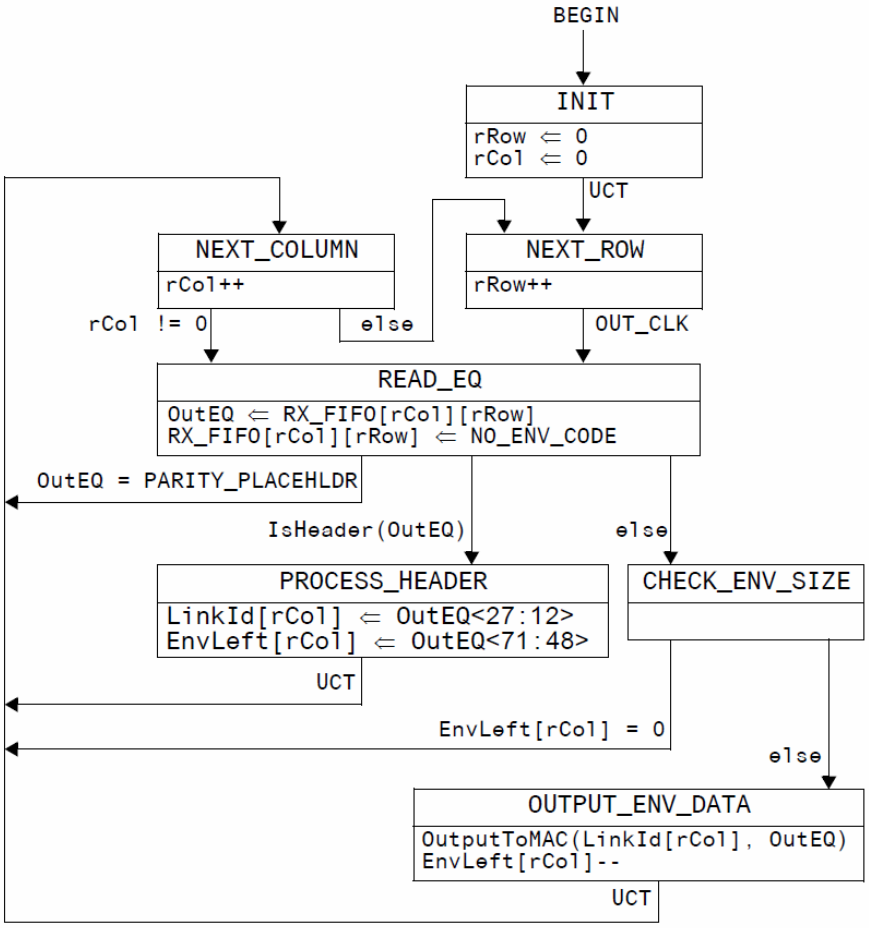
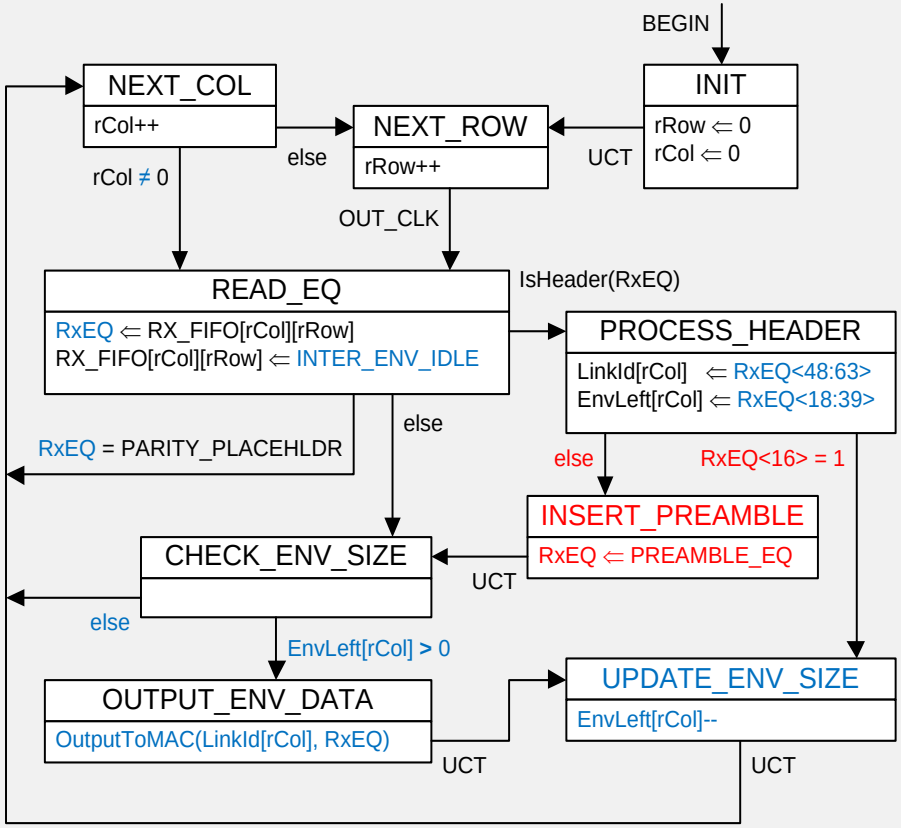


Figure 143–4—OLT MPRS Output Process state diagram



Motion #5

Adopt the State Diagrams and Definitions for transmission envelopes shown in kramer_3ca_1_0917 slides 10-21.

Moved: _Glen Kramer

Seconded: _Frank Effenberger

For: _22____

Against: _0____

Abstain: _2____

Technical (≥ 75%) Motion Passed

Thank You