ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

100BASE-T1L Block Structure

Philip Curran
Brian Murray
Jacobo Riesco

Introduction

- This presentation presents an update to the text for clause 199.3.3.4 Block structure in draft 0.2
  - As we discussed in our presentation at the February Electronic Interim there is ambiguity and missing cases in the current tables and text used in draft 0.2 and this needs be addressed for draft 1.0
  - See Comments on 802.3dg Draft 0.2
- This update does NOT include support for sequence ordered sets
  - We believe that writing this clause unambiguously while also supporting sequence ordered sets would require the use of a state diagram and this is significantly more complicated
- This version does include support for Assert Remote Fault
  - The use of Assert Remote Fault is discussed in Murray_01_03122025

Mapping MII Transfers to 8N/(8N + 1) Blocks

- Figure 199-5 is a diagram showing the mapping of the MII transfers to the 8N/(8N+1) blocks and 6T symbols

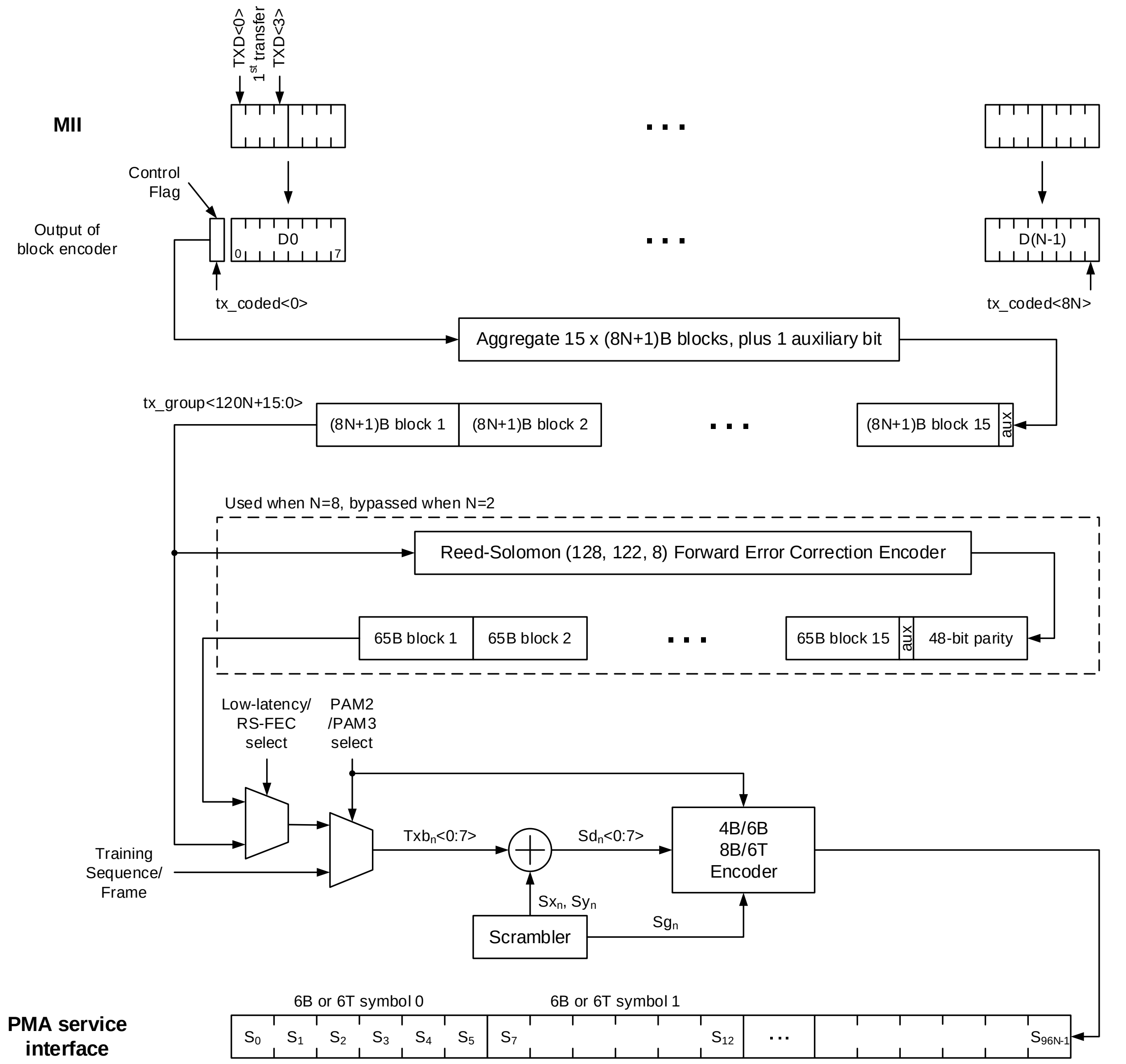

Figure 199–5— PCS Transmit bit ordering

Encoding of data on the MII into an 8N/(8N+1) Block

ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

- Th encoding of data on the MII into an 8N/(8N+1) block is done as follows
- Block structure
  - Blocks consist of 8N + 1 bits
  - The first bit of a block is the control flag
  - Blocks are either data blocks or control blocks
    - The control flag is 0 for data blocks and 1 for control blocks
  - The remainder of the block contains the payload
- The encoding is a two-step process
  - The first step converts two MII transfers at a time into a control symbol indication TS, and an octet, TOCT
  - The second step packs a set of N values of TS and of TOCT into an 8N+1 bit block
- Transfers over the MII are delineated as alternating even and odd transfers
  - The conversion of the even and odd transfers to the values TS and TOCT follows Table 199–2 using the categories defined in Table 199–1

MII Transfer Categories

- The MII transfer categories are defined in Table 199-1
  - An MII transfer may belong to more than one category
    - For example, normal inter- frame signaling belongs to the categories NIF and IDL
- Complementary category
  - For each category there is a complementary category which includes all MII transfers that do not belong to that category
  - The complementary category is denoted by placing the unary logical negation operator (!) before the category name
- ALPI
  - The category ALPI is only used when EEE is enabled for the link
  - Otherwise Assert LPI signaling is treated as normal inter-frame signaling
- ARF
  - The category ARF is Assert remote fault

Table 199–1— MII transfer categories

Category	tx_enable	tx_error	TXD<3:0>	Description
	loc_phy_ready = FALSE			
NOT_RDY	–	–	–	PHY not ready for MII transfer
	loc_phy_ready = TRUE			
DAT	1	0	–	Normal data transmission
ERR	1	1	–	Transmit error propagation
NIF	0	0	–	Normal inter-frame
ALPI	0	1	0001	Assert LPI
ARF	0	1	0100	Assert remote fault
IDL	0	–	–	

MII Transfer to TS and TOCT Mapping

ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

- Evaluation from top to bottom
  - Evaluation of the even and odd transfers against the conditions listed in the table proceeds from top to bottom
  - The first condition that is satisfied has priority, and TS and TOCT are set in accordance with the corresponding row in the table
- TOCT values
  - The table shows the TOCT values for control symbols using symbolic representations for clarity
  - The associated numerical values is shown in Table 199–3
- Delayed encoding
  - **dly_enc** records a requirement to implement a delayed encoding
  - This condition arises when a pair of transfers require two control symbols to be encoded
  - The encoding of one of these control symbols is delayed
  - For example, the encoding of transmit error propagation is delayed if it is signaled during the first MII transfer of a frame
  - When transmit error propagation is signaled during the last transfer of the frame and this transfer is even, the frame is extended by one byte
    - This allows the decoder to observe both the transmit error propagation event and the mismatch in byte alignment
    - The Transmit Error Propagation symbol (/E/) is followed by the Terminate symbol /Tu0/

Table 199–2— MII transfer to TS and TOCT mapping

Even transfer	Odd transfer	Current dly_enc	TS	TOCT	Next dly_enc
NOT_RDY	–	–	1	/Ix/	FALSE
–	NOT_RDY	–	1	/Ix/	FALSE
DAT after IDL	!ERR	–	1	/Sp/	FALSE
DAT after IDL	ERR	–	1	/Sp/	TRUE
ERR after IDL	–	–	1	/Sp/	TRUE
IDL	DAT	–	1	/Su/	FALSE
IDL	ERR	–	1	/Su/	TRUE
DAT after !IDL	DAT	TRUE	1	/E/	FALSE
IDL after !IDL	–	–	1	/Tp/	FALSE
DAT and TXD = 0x0	IDL	–	1	/Tu0/	FALSE
DAT and TXD = 0x1	IDL	–	1	/Tu1/	FALSE
DAT and TXD = 0x2	IDL	–	1	/Tu2/	FALSE
DAT and TXD = 0x3	IDL	–	1	/Tu3/	FALSE
DAT and TXD = 0x4	IDL	–	1	/Tu4/	FALSE
DAT and TXD = 0x5	IDL	–	1	/Tu5/	FALSE
DAT and TXD = 0x6	IDL	–	1	/Tu6/	FALSE
DAT and TXD = 0x7	IDL	–	1	/Tu7/	FALSE
DAT and TXD = 0x8	IDL	–	1	/Tu8/	FALSE
DAT and TXD = 0x9	IDL	–	1	/Tu9/	FALSE
DAT and TXD = 0xA	IDL	–	1	/TuA/	FALSE
DAT and TXD = 0xB	IDL	–	1	/TuB/	FALSE
DAT and TXD = 0xC	IDL	–	1	/TuC/	FALSE
DAT and TXD = 0xD	IDL	–	1	/TuD/	FALSE
DAT and TXD = 0xE	IDL	–	1	/TuE/	FALSE
DAT and TXD = 0xF	IDL	–	1	/TuF/	FALSE
ERR after !IDL	IDL	–	1	/E/	TRUE
IDL after IDL	IDL	TRUE	1	/Tu0/	FALSE
ERR after !IDL	!IDL	–	1	/E/	FALSE
DAT after !IDL	ERR	–	1	/E/	FALSE
ALPI after IDL	ALPI	FALSE	1	/L/	FALSE
ARF after IDL	ARF	FALSE	1	/Q/	FALSE
NIF after IDL	IDL	FALSE	1	/I/	FALSE
IDL after IDL	NIF	FALSE	1	/I/	FALSE
DAT after !IDL TOCT<3:0> = TXD	DAT TOCT<7:4> = TXD	FALSE	0	Data Value	FALSE
Otherwise		–	1	/I/	FALSE

TOCT Symbol to TOCT Value Mapping

- Table 199-3 is the mapping of the symbolic representation to the numerical value of TOCT

Table 199–3— TOCT symbol to TOCT value mapping

TOCT Symbol	Definition	TOCT Value
/Q/	Assert remote fault	0x00
/E/	Transmit Error Propagation	0x10
/I/	Normal Inter-Frame, loc_phy_ready = OK	0x08
/Su/	Start of packet on odd nibble	0x18
/Tp/	End of packet after odd nibble	0x04
/L/	Assert LPI	0x14
/Ix/	Normal Inter-Frame, loc_phy_ready = NOT_OK	0x0C
/Sp/	Start of packet on even nibble	0x1C
/Tu0/	End of packet after even nibble, last data nibble = 0x0	0x01
/Tu8/	End of packet after even nibble, last data nibble = 0x8	0x11
/Tu4/	End of packet after even nibble, last data nibble = 0x4	0x09
/TuC/	End of packet after even nibble, last data nibble = 0xC	0x19
/Tu2/	End of packet after even nibble, last data nibble = 0x2	0x05
/TuA/	End of packet after even nibble, last data nibble = 0xA	0x15
/Tu6/	End of packet after even nibble, last data nibble = 0x6	0x0D
/TuE/	End of packet after even nibble, last data nibble = 0xE	0x1D
/Tu1/	End of packet after even nibble, last data nibble = 0x1	0x03
/Tu9/	End of packet after even nibble, last data nibble = 0x9	0x13
/Tu5/	End of packet after even nibble, last data nibble = 0x5	0x0B
/TuD/	End of packet after even nibble, last data nibble = 0xD	0x1B
/Tu3/	End of packet after even nibble, last data nibble = 0x3	0x07
/TuB/	End of packet after even nibble, last data nibble = 0xB	0x17
/Tu7/	End of packet after even nibble, last data nibble = 0x7	0x0F
/TuF/	End of packet after even nibble, last data nibble = 0xF	0x1F

Python Code for 8N/(8N+1) Block Encoding

- Python code
  - The 8N/(8N+1) encoder should produce the same result as the following code

```
# Definition of variables:
#
# Inputs:
# N         Integer set to 2 when RS-FEC is disabled, or 8, when
#           RS-FEC is enabled.
# TS        List-like object of length N. TS[i] is 1 if TOCT[i] is
#           a control symbol and is 0 otherwise.
# TOCT      List-like object containing N integers, each in the
#           range 0 - 255.
#
# Output:
# tx_coded List containing 8N+1 integers, each having value 0 or 1.

for i in range(N):
  if i > 0:
    TS_prev = TS[i-1]
  else:
    TS_prev = 1

  TOR = int(1 in TS[i:])

  if i < N-1:
    TOR_next = int(1 in TS[i+1:])
  else:
    TOR_next = 0

  if TS_prev and TOR:
    TPTR = i + TS[i:].index(1)

  if TS[i]:
    if (TOCT[i] & 0x01):
      TCTL = (TOCT[i] & 0x1F)
    else:
      TCTL = (TOCT[i] & 0x1C) | (TOR_next << 1)

  if TOR:
    tx_octet = 0

    if TS_prev:
      tx_octet |= TPTR
    else:
      tx_octet |= ((TOCT[i-1] >> 5) & 0x07)

    if TS[i]:
      tx_octet |= (TCTL << 3)
    else:
      tx_octet |= ((TOCT[i] & 0x1F) << 3)
  else:
    tx_octet = TOCT[i]

  if i == 0:
    tx_coded = [TOR]

  for bit_indx in range(8):
    tx_coded.append((tx_octet >> bit_indx) & 0b1)
```

Changes to Fault Signaling / Sequence Ordered Sets

- The task force could decide not to support fault signaling at all **or** could decide to fully support adding Sequence ordered sets as in clause 46
- Removing support for fault signaling
  - If it is decided not to support Assert Remote Fault or fault signaling then the changes to the tables are very small
    - Remove row 'ARF' from Table 199-1
    - Remove row 'ARF after IDL' from Table 199-2
    - Remove row 'Q' from Table 199-3
  - The text remains the same
- Adding support for Sequence ordered sets
  - If support for Sequence ordered sets is required then much bigger changes are required
  - We believe this would require the use of a state diagram and would be significantly more complicated

Proposed Text for Draft 1.0 - Block Structure

199.3.3.4 Block structure

Blocks consist of 8N + 1 bits. The first bit of a block is the control flag. Blocks are either data blocks or control blocks. The control flag is 0 for data blocks and 1 for control blocks. The remainder of the block contains the payload.

The encoding of data on the MII into an (8N)B/(8N+1)B block is a two-step process. The first step converts two MII transfers at a time into a control symbol indication, TS, and an octet, TOCT. The second step packs a set of N values of TS and of TOCT into an 8N+1 bit block.

Transfers over the MII are delineated as alternating even and odd transfers. The conversion of the even and odd transfers to the values TS and TOCT shall follow Table 199–2 using the categories defined in Table 199–1. The state variable dly_enc records a requirement to implement a delayed encoding. This condition arises when a pair of transfers require two control symbols to be encoded. The encoding of one of these control symbols is delayed. For example, the encoding of transmit error propagation is delayed if it is signaled during the first MII transfer of a frame. The conditions to set the dly_enc variable to TRUE or FALSE shall follow Table 199–2.

The category ALPI is only used when EEE is enabled for the link. Otherwise Assert LPI signaling is treated as normal inter-frame signaling.

An MII transfer may belong to more than one of the categories in Table 199–1. For example, normal inter-frame signaling belongs to the categories NIF and IDL.

For each category in Table 199–1 there is a complementary category which includes all MII transfers that do not belong to that category. The complementary category is denoted by placing the unary logical negation operator (!) before the category name.

Table 199–1— MII transfer categories

Category	tx_enable	tx_error	TXD<3:0>	Description
	loc_phy_ready = FALSE			
NOT_RDY	–	–	–	PHY not ready for MII transfer
	loc_phy_ready = TRUE			
DAT	1	0	–	Normal data transmission
ERR	1	1	–	Transmit error propagation
NIF	0	0	–	Normal inter-frame
ALPI	0	1	0001	Assert LPI
ARF	0	1	0100	Assert remote fault
IDL	0	–	–	

Proposed Text for Draft 1.0 - Block Structure

Evaluation of the even and odd transfers against the conditions listed in Table 199–2 proceeds from top to bottom. The first condition that is satisfied has priority, and TS and TOCT are set in accordance with the corresponding row in the table.

When transmit error propagation is signaled during the last transfer of the frame and this transfer is even, the frame is extended by one byte. This allows the decoder to observe both the transmit error propagation event and the mismatch in byte alignment. In this case the Transmit Error Propagation symbol (/E/) is followed by the Terminate symbol /Tu0/.

Table 199–2 shows the TOCT values for control symbols using symbolic representations for clarity. The mapping from these symbolic representations to the associated numerical values is shown in Table 199–3.

Table 199–2— MII transfer to TS and TOCT mapping

Even transfer	Odd transfer	Current dly_enc	TS	TOCT	Next dly_enc
NOT_RDY	–	–	1	/Ix/	FALSE
–	NOT_RDY	–	1	/Ix/	FALSE
DAT after IDL	!ERR	–	1	/Sp/	FALSE
DAT after IDL	ERR	–	1	/Sp/	TRUE
ERR after IDL	–	–	1	/Sp/	TRUE
IDL	DAT	–	1	/Su/	FALSE
IDL	ERR	–	1	/Su/	TRUE
DAT after !IDL	DAT	TRUE	1	/E/	FALSE
IDL after !IDL	–	–	1	/Tp/	FALSE
DAT and TXD = 0x0	IDL	–	1	/Tu0/	FALSE
DAT and TXD = 0x1	IDL	–	1	/Tu1/	FALSE
DAT and TXD = 0x2	IDL	–	1	/Tu2/	FALSE
DAT and TXD = 0x3	IDL	–	1	/Tu3/	FALSE
DAT and TXD = 0x4	IDL	–	1	/Tu4/	FALSE
DAT and TXD = 0x5	IDL	–	1	/Tu5/	FALSE
DAT and TXD = 0x6	IDL	–	1	/Tu6/	FALSE
DAT and TXD = 0x7	IDL	–	1	/Tu7/	FALSE
DAT and TXD = 0x8	IDL	–	1	/Tu8/	FALSE
DAT and TXD = 0x9	IDL	–	1	/Tu9/	FALSE
DAT and TXD = 0xA	IDL	–	1	/TuA/	FALSE
DAT and TXD = 0xB	IDL	–	1	/TuB/	FALSE
DAT and TXD = 0xC	IDL	–	1	/TuC/	FALSE
DAT and TXD = 0xD	IDL	–	1	/TuD/	FALSE
DAT and TXD = 0xE	IDL	–	1	/TuE/	FALSE
DAT and TXD = 0xF	IDL	–	1	/TuF/	FALSE
ERR after !IDL	IDL	–	1	/E/	TRUE
IDL after IDL	IDL	TRUE	1	/Tu0/	FALSE
ERR after !IDL	!IDL	–	1	/E/	FALSE
DAT after !IDL	ERR	–	1	/E/	FALSE
ALPI after IDL	ALPI	FALSE	1	/L/	FALSE
ARF after IDL	ARF	FALSE	1	/Q/	FALSE
NIF after IDL	IDL	FALSE	1	/I/	FALSE
IDL after IDL	NIF	FALSE	1	/I/	FALSE
DAT after !IDL TOCT<3:0> = TXD	DAT TOCT<7:4> = TXD	FALSE	0	Data Value	FALSE
Otherwise		–	1	/I/	FALSE

Proposed Text for Draft 1.0 - Block Structure

N values of TS and of TOCT are grouped together and presented to the (8N)B/(8N+1)B encoding process. Blocks consist of 17 bits (N = 2) when RS-FEC is disabled and 65 bits (N = 8) when RS-FEC is enabled. The first bit of a block is the control flag. Blocks are either data blocks, if all the octets in the block are data octets, or control blocks, if there is at least one control octet in the block. The control flag is 0 for data blocks and 1 for control blocks. The remainder of the block contains the payload.

The payload of data blocks contains N data octets. The payload of control blocks begins with a 3-bit pointer field that indicates the number of the octet containing the first control octet within the block. Octets are numbered 0 to N-1.

If the first octet in the block is a control octet, the pointer field is followed by a 5-bit control code. Otherwise, the pointer field is followed by one or more data octets until the position of the next control octet. The control code indicates the type of the control symbol and whether more control octets follow in the block.

If there are additional control octets in the block, the control code is followed by a 3-bit pointer field to the next control octet. The pointer field may be followed by a data octet or by a control code depending on the value of the pointer field. In this way any combination of N data octets and control symbols may be encapsulated within an (8N)B/(8N+1)B block.

Table 199–3— TOCT symbol to TOCT value mapping

TOCT Symbol	Definition	TOCT Value
/Q/	Assert remote fault	0x00
/E/	Transmit Error Propagation	0x10
/I/	Normal Inter-Frame, loc_phy_ready = OK	0x08
/Su/	Start of packet on odd nibble	0x18
/Tp/	End of packet after odd nibble	0x04
/L/	Assert LPI	0x14
/Ix/	Normal Inter-Frame, loc_phy_ready = NOT_OK	0x0C
/Sp/	Start of packet on even nibble	0x1C
/Tu0/	End of packet after even nibble, last data nibble = 0x0	0x01
/Tu8/	End of packet after even nibble, last data nibble = 0x8	0x11
/Tu4/	End of packet after even nibble, last data nibble = 0x4	0x09
/TuC/	End of packet after even nibble, last data nibble = 0xC	0x19
/Tu2/	End of packet after even nibble, last data nibble = 0x2	0x05
/TuA/	End of packet after even nibble, last data nibble = 0xA	0x15
/Tu6/	End of packet after even nibble, last data nibble = 0x6	0x0D
/TuE/	End of packet after even nibble, last data nibble = 0xE	0x1D
/Tu1/	End of packet after even nibble, last data nibble = 0x1	0x03
/Tu9/	End of packet after even nibble, last data nibble = 0x9	0x13
/Tu5/	End of packet after even nibble, last data nibble = 0x5	0x0B
/TuD/	End of packet after even nibble, last data nibble = 0xD	0x1B
/Tu3/	End of packet after even nibble, last data nibble = 0x3	0x07
/TuB/	End of packet after even nibble, last data nibble = 0xB	0x17
/Tu7/	End of packet after even nibble, last data nibble = 0x7	0x0F
/TuF/	End of packet after even nibble, last data nibble = 0xF	0x1F

Proposed Text for Draft 1.0 - Block Structure

In the code that follows, the variable tx_octet represents the current block octet. When this octet is a control octet, bits tx_octet[7:3] contain the control code. The control code is generated from bits TOCT[4:0] for the control symbol as specified in Table 199–3. Bits tx_octet[7:5] are set equal to bits TOCT[4:2] and bit tx_octet[3] is set equal to bit TOCT[0]. When bit TOCT[0] is 0, tx_octet[3] is set to 0 and bit tx_octet[4] is set to indicate whether more control octets follow in the block. When bit TOCT[0] is 1, tx_octet[3] is set to 1 and it may be inferred that the next octet in the block, if any, is a control octet.

The (8N)B/(8N+1)B encoder shall produce the same result as the following code[1, 2]:

```
# Definition of variables:
#
# Inputs:
# N        Integer set to 2 when RS-FEC is disabled, or 8, when
#          RS-FEC is enabled.
# TS       List-like object of length N. TS[i] is 1 if TOCT[i] is
#          a control symbol and is 0 otherwise.
# TOCT     List-like object containing N integers, each in the
#          range 0 - 255.
#
# Output:
# tx_coded List containing 8N+1 integers, each having value 0 or 1.

for i in range(N):
  if i > 0:
    TS_prev = TS[i-1]
  else:
    TS_prev = 1

  TOR = int(1 in TS[i:])

  if i < N-1:
    TOR_next = int(1 in TS[i+1:])
  else:
    TOR_next = 0

  if TS_prev and TOR:
    TPTR = i + TS[i:].index(1)

  if TS[i]:
    if (TOCT[i] & 0x01):
      TCTL = (TOCT[i] & 0x1F)
    else:
      TCTL = (TOCT[i] & 0x1C) | (TOR_next << 1)

  if TOR:
    tx_octet = 0

    if TS_prev:
      tx_octet |= TPTR
    else:
      tx_octet |= ((TOCT[i-1] >> 5) & 0x07)

    if TS[i]:
      tx_octet |= (TCTL << 3)
    else:
      tx_octet |= ((TOCT[i] & 0x1F) << 3)
  else:
    tx_octet = TOCT[i]

  if i == 0:
    tx_coded = [TOR]

  for bit_indx in range(8):
    tx_coded.append((tx_octet >> bit_indx) & 0b1)
```

[1] This code is written in the Python programming language; however, use of this language does not indicate an endorsement of Python by IEEE and, as such, any tool can be used to perform this calculation.

[2] Copyright release for Python code: Users of this standard may freely copy or reproduce the Python code in this subclause so it can be used for its intended purpose.

Conclusions

- A update to the tables to encode the MII transfers into 8N/(8N+1) blocks has been presented
  - This update addresses and fixes issues discussed at the February Electronic Interim
  - These tables do NOT support Sequence ordered sets
  - These table do support Assert Remote Fault and fault signaling
- This presentation presents an update to the text and tables for clause 199.3.3.4 Block structure for draft 1.0

Questions ?