



10 Mb/s Single Twisted Pair Ethernet Proposed PCS Layer for Long Reach PHY

Dirk Ziegelmeier
Steffen Graber
Pepperl+Fuchs

Content

- PCS Overview
- Ternary Symbol Coding
- Running Disparity Handling and Checking
- Telegram Synchronization
- Side Stream Scrambling
- Signalizing during Idle Transmission
- Symbol Boundary Detection
- Polarity Detection
- Descrambler Synchronization
- Detection of Data and Idle Transmission
- Transmit State Machine
- Receive State Machine

PCS Overview

- Ternary (PAM-3) Symbol Coding (+1, 0, -1).
 - Symbol Rate of 7.5 MSPS/s using 4B3T Coding.
 - Running Disparity Handling and Checking.
 - Telegram Synchronization using robust Comma Sequences.
 - Side Stream Scrambling.
 - Automatic Descrambler Synchronization.
 - Detection of Data and Idle Transmission.
 - Signaling of „Remote Receiver Status“ and „Low Power Idle Request“ included in Idle Data.
 - MII Host Interface.
-
- Currently Energy Efficient Ethernet is not implemented (will need further modifications to state machines).

Ternary Symbol Coding

- PAM-3 Symbol Coding with 7.5 MSPS/s and controlled running disparity.
- 4 Bit (MII Nibble Data) are coded in a triple ternary symbol.
- Depending on the actual disparity (accumulated difference to mean value of the signal) the right coding word is being used.

Sd_n[3:0]	Disparity = 1	Disparity = 2	Disparity = 3	Disparity = 4
0000	+0+	0-0	0-0	0-0
0001	0-+	0-+	0-+	0-+
0010	+ -0	+ -0	+ -0	+ -0
0011	00+	00+	00+	--0
0100	-+0	-+0	-+0	-+0
0101	0++	-00	-00	-00
0110	-++	-++	-++	-++
0111	-0+	-0+	-0+	-0+
1000	+00	+00	+00	0--
1001	+++	+++	+++	---
1010	++-	++-	++-	++-
1011	+0-	+0-	+0-	+0-
1100	+++	-+-	-+-	-+-
1101	0+0	0+0	0+0	-0-
1110	0+-	0+-	0+-	0+-
1111	++0	00-	00-	00-

Running Disparity Handling and Checking

- To limit the maximum additional signal amplitude on the link segment, caused by BLW, a strict control of the running disparity is implemented by the used 4B3T coding.
- Depending on the current running disparity the transmitted code words are always chosen in a way, so that the running disparity is limited to a narrow band between 1 and 4.
- Doing a reset of the running disparity within the comma words and calculating the running disparity also within the receiver allows an easy additional error detection possibility at the receiver side.
- Even a single bit error leads to a difference in the calculated running disparity at the receiver side, and it is likely, that the subsequently received bit patterns do not fit the calculated running disparity at the receiver side anymore and therefore are already detected as a receive error before feeding the data to the MII interface.

Telegram Synchronization

- Telegram Synchronization is done using comma sequences.
- Each comma sequence consists of four triple ternary symbols (12 PAM-3 Symbols, 16 Bits)



- COMMA1 and COMMA2 use the triple ternary symbol (0, 0, 0), which is not being used within the normal data stream (during normal communication a maximum of 4 subsequent zeros are seen in the data stream).
- As the triple ternary symbol boundary is adjusted during the training process, theoretically just COMMA1 or COMMA2 would be enough for signaling, adding a second comma word increases the robustness of the delimiter for false detection.
- The next triple ternary symbol is DISPRESET3, which is chosen depending on the current running disparity.
- A total of 4 different DISPRESET3 symbols exist, as there are four possible running disparity values.
- The fourth triple ternary symbol provides information, if the delimiter is an SSD, an ESD or an ESD with Error.

Telegram Synchronization

- The following table shows the different possible DISPRESET3 symbols:

Symbol Triplet	Disparity = 1	Disparity = 2	Disparity = 3	Disparity = 4
DISPRESET3	(-1, 0, 1)	(-1, 0, 0)	(-1, 0, -1)	(-1, -1, -1)

- The symbols are chosen such that, depending on the current running disparity, a running disparity value of 1 is always reached (together with the SSD4/ESD4/ESD_ERR4 symbols this will lead to a running disparity of 2, as these symbols always have a disparity of 1).
- The following table shows the different possible SSD4/ESD4/ESD_ERR4 symbols:

Delimiter	(TA_n, TB_n, TC_n)
SSD4	(1, 1, -1)
ESD4	(1, -1, 1)
ESD_ERR4	(-1, 1, 1)

- The coding of the ternary delimiter symbols is chosen, so that they have the maximum possible distance within the coding, to detect a wrong delimiter, at least two single symbols have to be flipped.

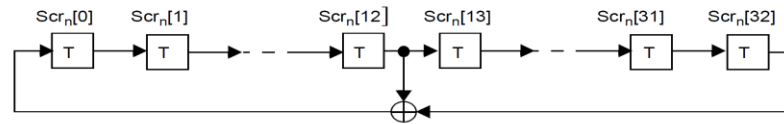
Side Stream Scrambling

- Parts of the 1000BASE-T (Clause 40) side stream scrambling are being used within the PCS.
- The following generator polynomials are being used within the master and the slave PHY transmitter:

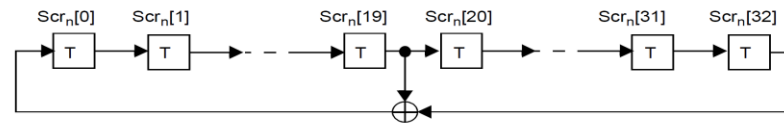
$$g_M(x) = 1 + x^{13} + x^{33}$$

$$g_S(x) = 1 + x^{20} + x^{33}$$

Side-stream scrambler employed by the MASTER PHY



Side-stream scrambler employed by the SLAVE PHY



- At the descrambler side master and slave polynomials are exchanged compared to the scrambler side:

$$g'_M(x) = 1 + x^{20} + x^{33}$$

$$g'_S(x) = 1 + x^{13} + x^{33}$$

Side Stream Scrambling

- Additionally to the scrambler polynomials an auxiliary polynomial is being used:

$$g(x) = x^3 \wedge x^8$$

- With the help of this auxiliary polynomial 4 bits $Sy_n[3:0]$ are being generated:

$$Sy_n[0] = Scr_n[0]$$

$$Sy_n[1] = g(Scr_n[0]) = Scr_n[3] \wedge Scr_n[8]$$

$$Sy_n[2] = g^2(Scr_n[0]) = Scr_n[6] \wedge Scr_n[16]$$

$$Sy_n[3] = g^3(Scr_n[0]) = Scr_n[9] \wedge Scr_n[14] \wedge Scr_n[19] \wedge Scr_n[24]$$

- By construction, the four bits $Sy_n[3:0]$ are derived from elements of the same maximum-length shift register sequence of length 2^{33-1} as $Scr_n[0]$, but shifted in time by varying delays. The associated delays are all large and different so that there is no apparent correlation among the bits.

Side Stream Scrambling

- From bits $Sy_n[3:0]$ another four bits $Sc_n[3:0]$ are generated (just to stay within the naming conventions):

$$Sc_n[3:0] = Sy_n[3:0]$$

- From scrambler bits $Sc_n[3:0]$, $TXD[3:0]$ and the control bits, bits $Sd_n[3:0]$ are generated:

$$Sd_n[3] = \begin{cases} Sc_n[3] \wedge TXD_n[3] & \text{if } (tx_enable_mii = TRUE) \\ Sc_n[3] \wedge 1 & \text{else if } (loc_rcvr_status = OK) \\ Sc_n[3] & \text{else} \end{cases}$$

$$Sd_n[2] = \begin{cases} Sc_n[2] \wedge TXD_n[2] & \text{if } (tx_enable_mii = TRUE) \\ Sc_n[1] \wedge 1 & \text{else if } (loc_lpi_req = TRUE) \\ Sc_n[1] & \text{else} \end{cases}$$

$$Sd_n[1:0] = \begin{cases} Sc_n[1:0] \wedge TXD_n[1:0] & \text{if } (tx_enable_mii = TRUE) \\ (Sc_n[2] Sc_n[0]) & \text{else} \end{cases}$$

Bits $Sd_n[3:0]$ are then 4B3T encoded and provided to the PMA.

Signalizing during Idle Transmission

- During transmission of idle data the two upper bits of each 4 bit nibble are used to transmit the local receiver status and an optional local low power idle request signal, needed for a later EEE implementation.
- To provide a reliable data transmission, it is checked that the same information is received at least 8 times in a row during idle transmission with the same value, before the status bit is being changed at the receiver side.
- During idle transmission $Sc_n[1]$ and $Sc_n[2]$ are exchanged compared to a normal data transmission.
- This is necessary to be able to reliably distinguish a „zero“ data transmission from an idle transmission.

Symbol Boundary Detection

- During initial training phase the correct boundary of the triple ternary symbols needs to be detected.
- This can be done independently from the polarity detection and descrambler synchronization.
- Initially the PHYs are only sending idle data without comma sequences, therefore the symbol (0, 0, 0) will not be present within the received data stream.
- If symbol (0, 0, 0) is being received, the receiver knows, that it is not being synchronized to the correct symbol boundary and therefore the input to the symbol decoder needs to be shifted by one position.
- If then still a (0, 0, 0) symbol is being received, another shift is necessary.
- If there is a receive error and the shift has been performed accidentally, another two shift operations will be necessary (the same shifting algorithm can be used).
- Shifting can be done independent from other operations until a descrambler lock is being reached.

Polarity Detection

- Additionally to the symbol boundary detection a signal polarity detection is necessary on the master and the slave PHY side.
- Initially normal polarity can be assumed and when a descrambler lock is not reached within a certain amount of time (e.g. after receiving 128 triple ternary symbols), the polarity at the receiver side is reversed.
- The toggling of the receive polarity is done until a descrambler lock is being reached.
- Polarity detection can be done in parallel to the symbol boundary detection.
- If there is a receive bit error during polarity detection, the scrambler lock will not be reached and another polarity toggle cycle is necessary.
- The polarity of the transmitted signal is not being changed.
- Therefore the slave as well as the master PHY have to implement the same polarity detection algorithm.
- This allows for an independent polarity detection on the master as well as on the slave side.

Descrambler Synchronization

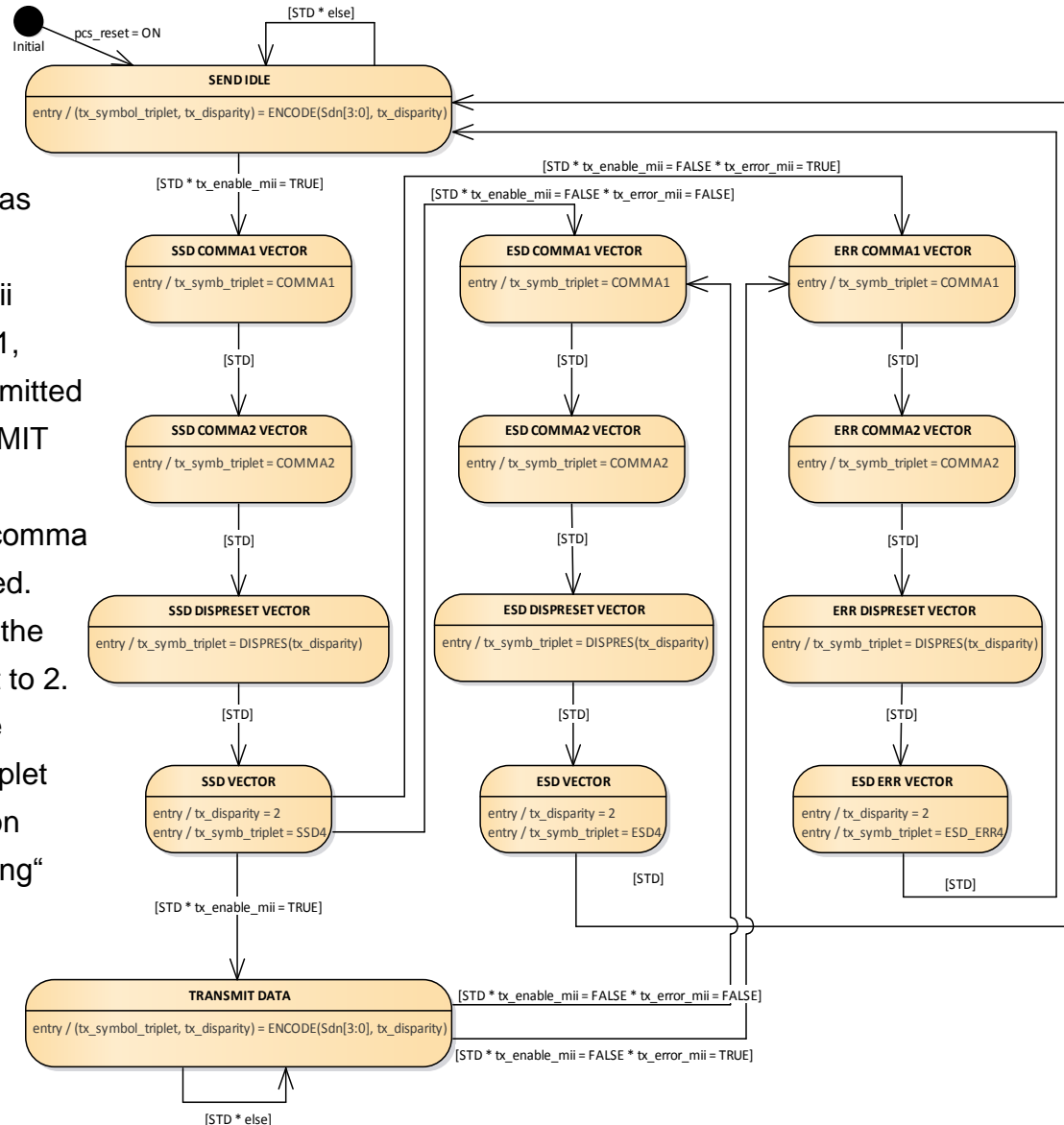
- During idle transmission bits $Sd_n[1:0]$ contain only the raw scrambler data $Sc_n[2]$ and $Sc_n[0]$ (only $Sd_n[3:2]$ contains control information XORed with the scrambler data).
- Therefore the transmitted $Sd_n[0]$ is identical to $Sc_n[0]$ of the side stream scrambler at the transmitter side.
- During descrambler synchronization, this value can be used to load the side stream scrambler at the receiver side with the correct data.
- As the side stream scrambler has a length of 33 bit, after 33 triple ternary symbols have been received (and no bit error is assumed) the side stream scrambler at the receiver side is in sync with the side stream scrambler at the transmitter side.
- To check, that there has been no error during synchronization, bits $Sd_n[1:0]$ can be checked against the descrambler output for some time (e.g. at least for 33 triple ternary symbols, as the descrambler has a length of 33 bit), to make sure that the descrambler is locked correctly.

Detection of Data and Idle Transmission

- During normal communication a data transmission starts after receiving a SSD (start of stream delimiter), idle data transmission starts after receiving an ESD (end of stream delimiter).
- All comma sequences contain a high number of bits and are checked to be valid over the full length of the sequence for bit errors, therefore it is very unlikely to accidentally detect a delimiter within a normal data or idle stream.
- Nevertheless it could happen that during transmission of a delimiter a bit error occurs.
- If such a case is detected, the receive state machine waits until idle data are detected again.
- As within the idle data stream the lower two bits of a 4 bit nibble always contain the content of scrambler bits 2 and 0, after descrambling these bits always must be zero, if idle data are being transmitted.
- Analyzing, that e.g. at least the last 8 received and decoded triple ternary symbols in a sequence fulfil this requirement, idle data transmission is assumed and the receiver is set to IDLE state again.

Transmit State Machine

- There are two main states „SEND IDLE“ and “TRANSMIT DATA”.
- „STD“ is an alias for symbol_triplet_timer_done and is triggered each time a tx_symbol_triplet has been transmitted.
- Being in „SEND IDLE“ as soon as tx_enable_mii becomes TRUE, a comma sequence (COMMA1, COMMA2, DISPRESET3, SSD4) is being transmitted and afterwards the state machine is in „TRANSMIT DATA“ state.
- If tx_enable_mii becomes „FALSE“ an second comma sequence (with ESD4/ESD_ERR4) is transmitted.
- While transmitting SSD4, ESD4 or ESD_ERR4 the running disparity on the transmitter side is reset to 2.
- During transmission of idle or data symbols, the ENCODE function, calculates the tx_symbol_triplet from the binary 4 bit values of $Sd_n[3:0]$, based on the table shown in chapter „ternary symbol coding“ within this presentation.
- Additionally this function also calculates the tx_disparity, based on the disparity (DC offset) of each individual tx_symbol_triplet.



Transmit State Machine

- Within the transmit state machine the ENCODE function is being used:

$$(tx_symbol_triplet, tx_disparity) = ENCODE(Sd_n[3:0], tx_disparity)$$

- This function encodes the nibble data $Sd_n[3:0]$ provided by the scrambler, based on the current $tx_disparity$ (transmit running disparity).
- The $tx_disparity$ can be between 1 and 4 and the respective $tx_symbol_triplet$ is taken from the 4B3T encoding table shown within this presentation in chapter „Ternary Symbol Coding“ based on the $Sd_n[3:0]$ value and the $tx_disparity$:

$$tx_symbol_triplet = table_{4B3T}(Sd_n[3:0], tx_disparity)$$

- The second output value of this function is an updated $tx_disparity$ value.
- This value is calculated in the following way:

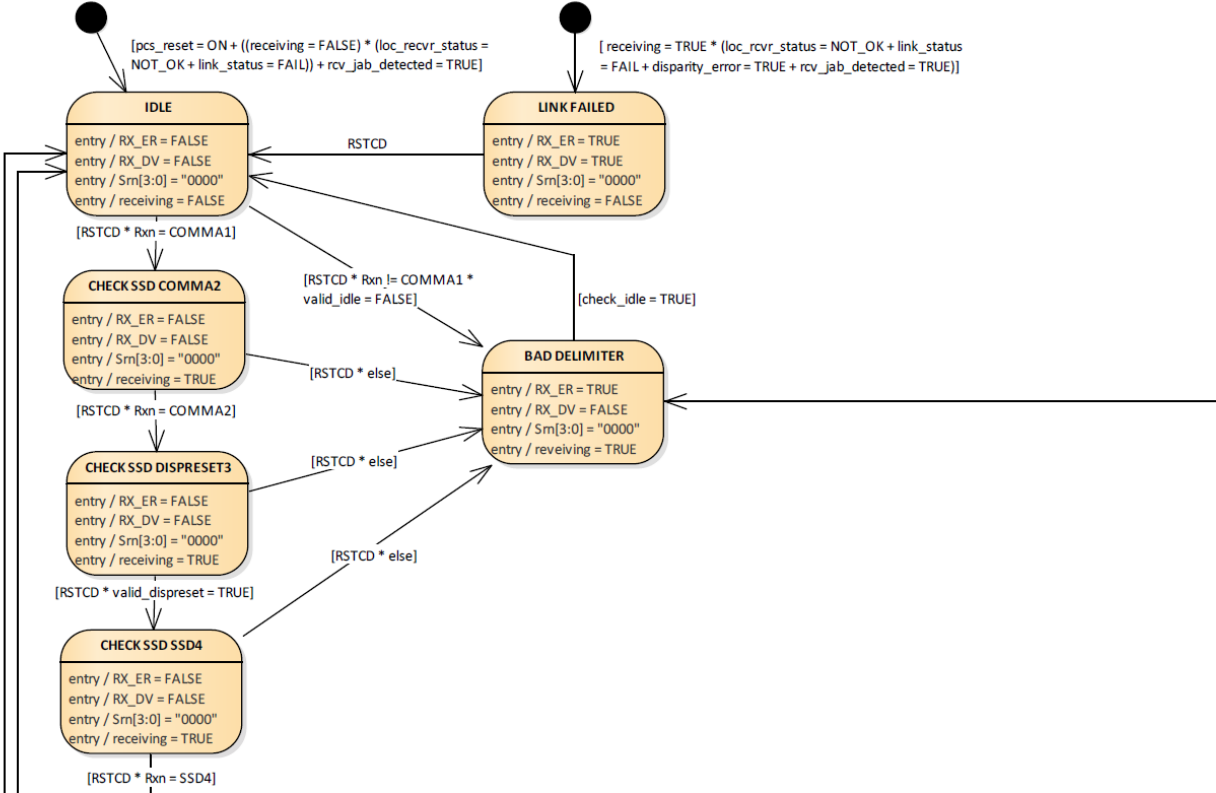
$$tx_disparity = tx_disparity + disparity\ of\ currently\ encoded\ tx_symbol_triplet$$

- The function DISPRES returns one of the four possible DISPRESET3 triple ternary symbols (see chapter „Ternary Symbol Coding“ within this presentation), depending on the actual $tx_disparity$:

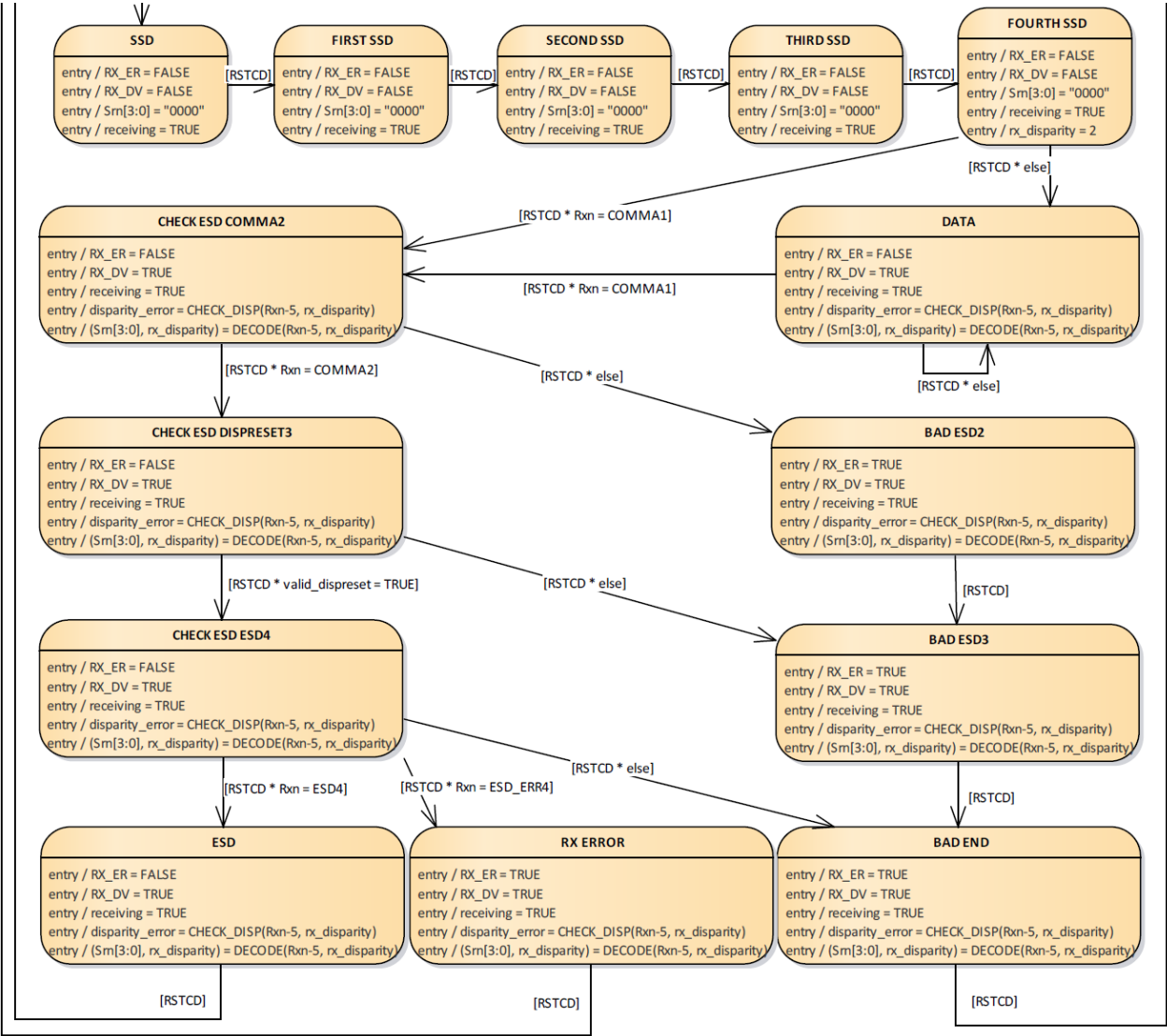
$$tx_symbol_triplet = table_{DISPRESET3}(tx_disparity)$$

Receive State Machine

- The receive state machine has two main states „IDLE“ and „DATA“.
- RTSCD is an abbreviation for „Receive Symbol Triplet Conversion Done“ and is synchronized with the PCS receive clock.
- When being in „IDLE“ state, as soon as the first COMMA1 symbol (0, 0, 0) is received, the „IDLE“ state is left and the decoding of the comma sequence (SSD) takes place.
- If the comma sequence can be decoded correctly, after filling the receive FIFO buffer the state machine goes into „DATA“ state.



Receive State Machine



Receive State Machine

- If there is a decoding error, the state machine goes into the „BAD DELIMITER“ state and waits until function „check_idle“ returns TRUE (e.g. after continuously receiving 8 IDLE triple ternary symbols).
- When being in „DATA“ state and receiving the first COMMA1 (0, 0, 0) symbol, the end of stream delimiter is being decoded, while the receive data are delivered from the receive FIFO buffer (this is the reason for the complexity of the state machine).
- If the ESD or ESD_ERR delimiters can be decoded correctly, the state machine goes back into „IDLE“ state, otherwise the state machine goes into state „BAD DELIMITER“.
- If an ESD_ERR or a wrong ESD sequence is being received the RX_ER output of the MII is being set.
- The DECODE function within the receive state machine decodes the received triple ternary symbols

$$Sr_n[3:0] = \text{inverse_table}_{4B3T}(Rx_n)$$

- The second output value of this function is an updated rx_disparity value.
- This value is calculated in the following way:

$$rx_disparity = rx_disparity + \text{disparity of currently received } Rx_n$$

- The CHECK_DISP function checks, if the currently received triple ternary symbol is allowed for the current rx_disparity.

$$\text{disparity_error} = (Rx_n \neq \text{table}_{4B3T}(\text{inverse_table}_{4B3T}(Rx_n), rx_disparity))$$

- The valid_dispreset function returns TRUE, if one of the 4 allowed DISPRESET3 values is being received.
- The check_idle function returns TRUE, if e.g. 8 subsequent IDLE triple ternary symbols have been received.

Thank You