1000BASE-X Physical Coding Sublayer (PCS) and Physical Medium Attachment (PMA) Tutorial

Jon Frain
6/15/98

Edited and Expanded by
Mike Henninger
4/13/2005

Abstract:

The purpose of this tutorial is to provide a detailed explanation of how the PCS and PMA operate within the confines of Clause 36 of the IEEE 802.3 standard. The reader is provided with a number of diagrams and tables to aid in understanding the purposes of Clause 36.

Table of Contents

I) Introduction:

The PCS and the PMA fit into the ISO/OSI stack model as shown in Figure 1 below:

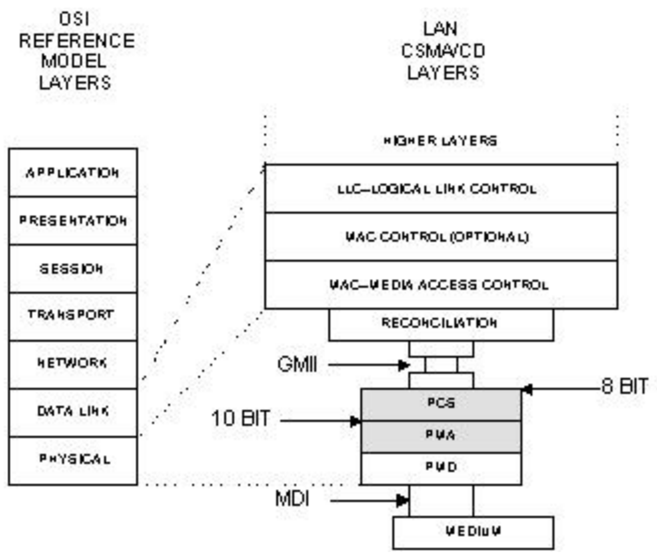

Figure 1: PCS and PMA relationship to the ISO/OSI model

The PCS and the PMA are both contained within the physical layer of the OSI reference model. The PCS and the Gigabit Media Independent Interface (GMII) communicate with one another via 8-bit parallel data lines and several control lines. The PCS is responsible for encoding each octet passed down from the GMII into ten bit code groups. The PCS is also responsible for decoding ten bit code groups passed up from the PMA into octets for use by the upper layers. The PCS also controls the auto-negotiation process which allows two separate gigabit devices to establish a link of which they are both capable of using.

The PMA is responsible for serializing each ten bit code group received from the PCS and sending the serialized data to the PMD. The PMA is responsible for deserializing every ten bit code group received from the PMD and passing it to the PCS. The PMA is also responsible for aligning the incoming serial data stream prior to passing ten bit code words up to the PCS.

II) PCS Fundamentals

A) 8B/10B Encoding/Decoding

Notation

The PCS examines each incoming octet passed down by the GMII and encodes it into a ten bit *code group*, this is referred to as 8B/10B encoding. Each octet is given a code group name according to the bit arrangement. Each octet is broken down into two groups; the first group contains the three most significant bits (y) and the second group contains the remaining five bits (x). Each data code group is named /Dx.y/, where the value of x represents the decimal value of the five least significant bits and y represents the value of the three most significant bits. For example:

- /D0.0/ = 000 00000
- /D6.2/ = 010 00110

- /D30.6/= 110 11101

here are also 12 special octets which may be encoded into ten bits. The PCS differentiates between special and data code words via a signal passed to it from the GMII. Special code words follow the same naming convention as data code words except they are named /Kx.y/ rather than /Dx.y/.

Motivation

One of the motivations behind the use of 8B/10B encoding lies with the ability to control the characteristics of the code words such as the number of ones and zeros and the consecutive number of ones or zeros. Another motivation behind the use of 8B/10B encoding is the ability to use special code words which would be impossible if no encoding was performed.

Features

- Every ten bit code group must fit into one of the following three possibilities (This helps limit the number of consecutive ones and zeros between any two code groups):
    1. Five ones and five zeros
    2. Four ones and six zeros
    3. Six ones and four zeros

- A special sequence of seven bits, called a **comma**, is used by the PMA in aligning the incoming serial stream. The comma is also used by the PCS in acquiring and maintaining synchronization. The following bit patterns represent the comma:
    - 0011111 (comma+)
    - 1100000 (comma-)

  The comma is contained within only the /K28.1/, /K28.5/ and /K28.7/ special code groups. The comma can not be transmitted across the boundaries of any two adjacent code groups unless an error has occurred.*(1)* This characteristic makes it very useful to the PMA in determining code group boundaries.

- DC balancing is achieved through the use of a running disparity calculation. Running disparity is designed to keep the number of ones transmitted by a station equal to the number of zeros transmitted by that station. This should keep the DC level balanced halfway between the 'one' voltage level and the 'zero' voltage level. Running disparity can take on one of two values: positive or negative. In the absence of errors, the running disparity value is positive if more ones have been transmitted than zeros and the running disparity value is negative if more zeros have been transmitted than ones since power-on or reset. Running disparity is explained in much more detail in section 2 of PCS Fundamentals.

- The 8B/10B encoding scheme was designed to provide a high transition density which makes synchronization of the incoming bit stream easier.

(1). The exception to this rule is if /K28.7/ is followed by any of the following special or data code groups: /K28.x/, /D3.x/, /D11.x/, /D12.x/, /D19.x/, or /D28.x/, where x is a value in the range 0 to 7, inclusive.

B) Running Disparity

Ten-bit code groups can be categorized into data (Dx.y), special (Kx.y) and invalid code groups. Each code group has two possible encoding values based upon the current running disparity. Table 36-1 contains all of the valid encodings of data bits 00-FF. Table 36-2 contains the 12 valid special code groups. Invalid code groups are ten-bit code groups which haven?t been defined within tables 36-1 or 36-2, and those code groups which are received or transmitted with the wrong running disparity.

Table 1 (below) gives examples of how four different eight-bit code groups are encoded based upon the current running disparity value. If the current running disparity is negative then the encoded value will come from the Current RD- column. If the current running disparity is positive then the encoded value will come from the Current RD+ column. It is possible for the ten bit code groups to be identical for both columns of a given code group. An example of this is D28.5 below.

The Current RD- column contains code groups which do not contain more zeros than ones. This is because in the absence of errors, the current negative running disparity value shows that more zeros have been transmitted than ones, and so a code group with more ones than zeros will have to be transmitted before another code group with more zeros than ones can be transmitted. The Current RD+ column contains no code groups which contain more ones than zeros for the opposite reasons.

Code Group Name	Data Octet	Current RD-	Current RD+	Ending RD
D1.0	000 00001	011101 0100	100010 1011	same
D4.1	001 00100	110101 1001	001010 1001	flip
D28.5	101 11100	001110 1010	001110 1010	same
K28.5	101 11100	001111 1010	110000 0101	flip

Table 1: 10B running disparity encoding examples

Subclause 36.2.4.4 defines how the running disparity of transmitted and received ten-bit code groups is calculated. Each ten-bit code group is broken down into two sub-blocks, the first of which is the most significant six-bits and the second is the remaining four bits. The running disparity at the end of the six-bit sub-block is the running disparity at the beginning of the four-bit sub-block. The running disparity at the end of the four-bit sub-block is the running disparity at the end of the code group.

Running Disparity of Transmitted Code Groups

Figure 2 (below) demonstrates how running disparity is determined for each transmitted 10-bit code group. Several variables in the state diagram require explanation. Zeros represents the number of zeros contained within the sub-block and ones represents the number of ones contained within the sub-block. 6bit-block contains the most significant six bits of the received code group and 4bit-block contains the remaining four bits of the received code group. RD is the current value of the transmitter?s running disparity. The INITIALIZE_RD state is where the transmitter?s running disparity is set upon start-up. Subclause 36.2.4.4 defines how the transmitter is to set the running disparity upon start-up as follows:

"After powering on or exiting a test mode, the transmitter shall assume the negative value for its initial running disparity."

The running disparity at the beginning of a sub-block is only used in calculating the running disparity at the end of the sub-block when an equal number of ones and zeros are contained within the sub-block. Otherwise the previous running disparity is irrelevant when calculating the running disparity of a code group.

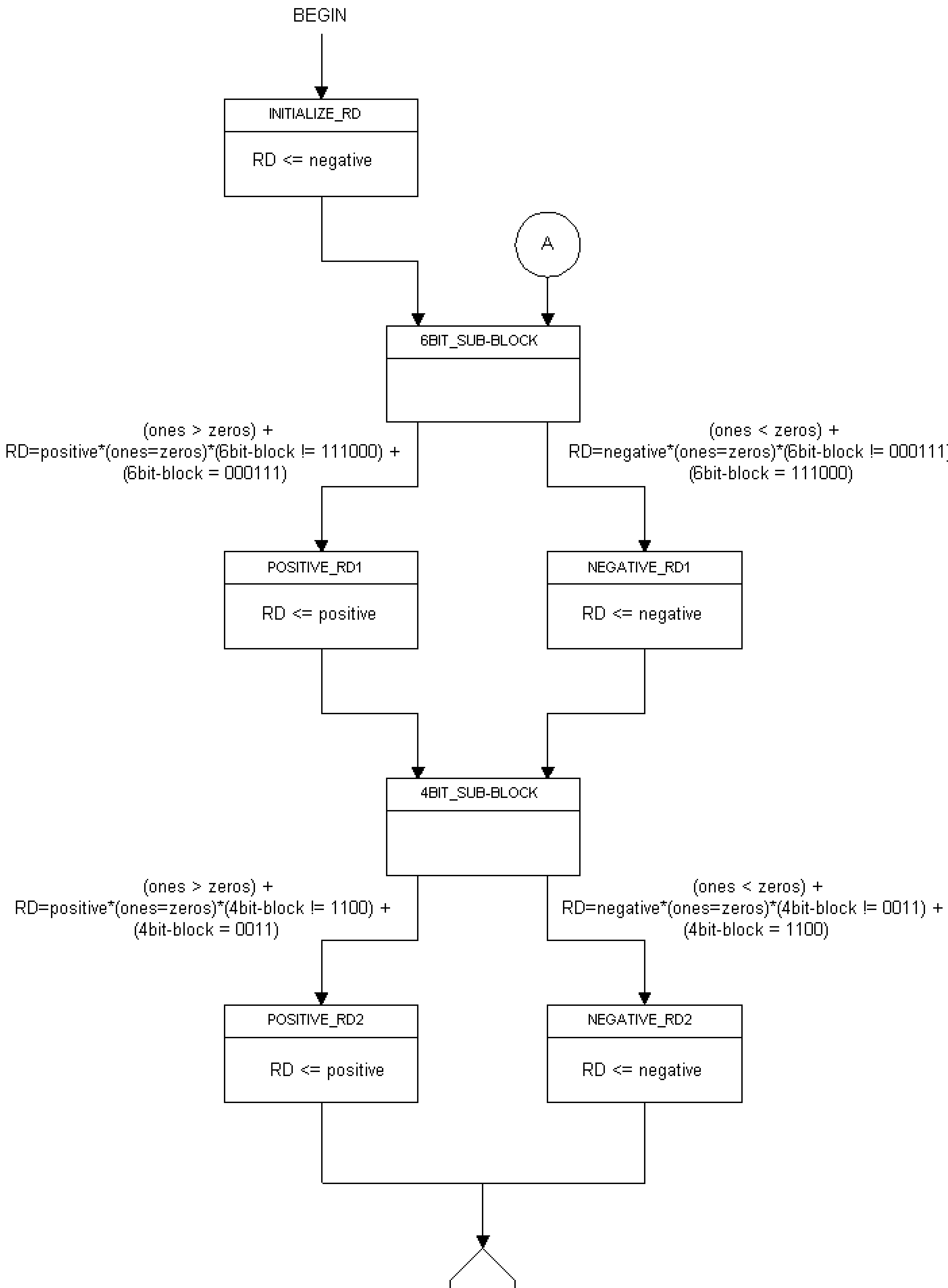
BEGIN
INITIALIZE_RD
RD <= negative
A
6BIT_SUB-BLOCK
(ones > zeros) +
RD=positive*(ones=zeros)*(6bit-block != 111000) +
(6bit-block = 000111)
(ones < zeros) +
RD=negative*(ones=zeros)*(6bit-block != 000111)
(6bit-block = 111000)
POSITIVE_RD1
RD <= positive
NEGATIVE_RD1
RD <= negative
4BIT_SUB-BLOCK
(ones > zeros) +
RD=positive*(ones=zeros)*(4bit-block != 1100) +
(4bit-block = 0011)
(ones < zeros) +
RD=negative*(ones=zeros)*(4bit-block != 0011) +
(4bit-block = 1100)
POSITIVE_RD2
RD <= positive
NEGATIVE_RD2
RD <= negative

Figure 2: PCS transmitter's running disparity state diagram

Running Disparity of Received Code groups

The PCS Receiver's state diagram is identical to the PCS Transmitter?s state diagram with one exception in the INITIALIZE_RD state. Subclause 36.2.4.4 defines how the receiver is to set the running disparity upon start-up as follows:

"After powering on or exiting a test mode, the receiver should assume either the positive or negative value for its initial running disparity."

This allows the vendor to arbitrarily choose which receiver running disparity value to start with. Because of this, the INITIALIZE_RD state doesn?t specify whether the initial running disparity is set to positive or negative. And so, it is possible at start-up to receive several code groups and assign the wrong running disparity to them. This is only possible if the six-bit and four-bit sub-blocks of these code groups each contain an equal number of ones and zeros and these sub-blocks don?t contain 111000 or 000111 or 1100 or 0011. Code groups that contain the same number of ones as zeros maintain the current running disparity, whether it be positive or negative, and therefore will not be recognized as invalid code groups. This shouldn?t be a problem because after start-up the device will begin auto-negotiation with the link partner. Which means that /K28.5/ will be received. The six-bit sub-blocks contained within /K28.5/ consist of either 110000 or 001111, which will correct any preceding running disparity errors. If the device is a manually configured device, it should receive IDLEs which also contain /K28.5/. In addition to receiving /K28.5/, any received code group that does not maintain the current running disparity will be flagged as an error. Even though this code group is marked as an error, it establishes the correct running disparity on the receiving end so that transmission may continue error free. An example of this is shown below.

Device Receives:

- 101001 0110 +/-D5.6
- 101001 0110 +/-D5.6
- 001111 1010 -K28.5
- 100100 0101 +D16.2

Beginning RD	6-bit block	RD after 6-bit block	4-bit block	Ending RD	Code-group
-	101001	-	0110	-	+/-D5.6 (maintain)
-	101001	-	0110	-	+/-D5.6 (maintain)
-	001111	+	1010	+	-K28.5 (flip)
+	100100	-	0101	-	+D16.2 (flip)

If the receiver initially chooses RD+

Beginning RD	6-bit block	RD after 6-bit block	4-bit block	Ending RD	Code-group
+	101001	+	0110	+	+/-D5.6 (Invalid for RD+ column)
+	101001	+	0110	+	+/-D5.6 (Invalid for RD+ column)
+	001111	+	1010	+	ERROR invalid code (Invalid for RD+ column)
+	100100	-	0101	-	+D16.2

The running disparity at the beginning of a sub-block is only used in calculating the running disparity at the end of the sub-block when an equal number of ones and zeros are contained within the sub-block. Otherwise the previous running disparity is irrelevant when calculating the running disparity of a code group.

C) Ordered Sets:

Notation

The notation used for ordered sets is similar to that used for code groups. Code groups are written as either /Dx.y/ or /Kx.y/. Ordered sets are written in the form of /XY/ where X is a letter and Y is sometimes used and contains a number. The defined ordered sets are: /C/, /C1/, /C2/, /I/, /I1/, /I2/, /R/, /S/, /T/ and /V/.

Definition

- Consist of either one, two or four code groups
- The first code group must be a special code group
- The second code group must be a data code group.

Defined ordered sets

/C/ = Configuration (/C1/ or /C2/)

- /C1/ = /K28.5/D21.5/config_reg[7:0]/config_reg[15:8]/
- /C2/ = /K28.5/D2.2/config_reg[7:0]/config_reg[15:8]/
- /K28.5/ is used as the first code group because it contains a comma which is a unique data pattern that was defined previously. The reception of this code group will not happen during a data packet unless there is a data error. This makes it very useful for use with very specific ordered sets such as Idle and Configuration.
- Continuous repetition of ordered sets /C1/ alternating with /C2/. It is used to convey the 16-bit Configuration Register to the link partner.

- /C1/ will flip the current running disparity after the transmission of /D21.5/. This is because /K28.5/ will flip the running disparity and /D21.5/ will maintain the current running disparity.
- /C2/ will sustain the current running disparity after the transmission of /D2.2/. This is because both /K28.5/ and /D2.2/ flip the current running disparity.
- /D21.5/ and /D2.2/ were chosen for their high bit transition density.

/I/ = IDLE (/I1/ or /I2/)

- /I1/ = /K28.5/D5.6/ * /I2/ = /K28.5/D16.2/
- Transmitted continuously and repetitively whenever the GMII interface is idle (TX_EN and TX_ER are both inactive). It provides a continuous fill pattern to establish and maintain clock synchronization.
- /I1/ is transmitted only if the current running disparity value is positive. In which case it is used to correct to the RD- state.
- /I2/ is transmitted when the current running disparity value is negative. It is used to maintain the RD- state.
- /D5.6/ and /D16.2/ were chosen for their high bit transition density.

/S/ = Start_of_Packet delimiter

- /S/ = /K27.7/
- Used to delineate the starting boundary of a data sequence.

/T/ = End_of_Packet delimiter

- /T/ = /K29.7/
- Used to delineate the ending boundary of a packet. The EPD is transmitted by the PCS following each de-assertion of TX_EN on the GMII, which follows the last data octet composing the FCS of the MAC packet.
- The EPD consists of either /T/R/I/ or /T/R/R/.

/R/ = Carrier_Extend

- /R/ = /K23.7/
- Used to extend the duration of a carrier event.
- Used to separate packets within a burst of packets.
- Used to pad the last or only packet of a burst of packets so that the subsequent /I/ is aligned on an even-numbered code group boundary.
- Used in the EPD.
- /R/ is required within the EPD to meet the Hamming distance requirement for ending delimiters.

/V/ = Error_Propagation

- /V/ = /K30.7/

- The presence of Error_Propagation (or an invalid code group) on the medium denotes a collision artifact or an error condition. Transmitted upon the assertion of TX_EN and TX_ER from the GMII, or the assertion of TX_ER with the de-assertion of TX_EN while TXD<7:0> is not equal to 0F.

III) PCS Functions

The PCS can be broken down into four major functions:

- Synchronization Process
- Transmit Process
- Receive Process
- Auto-negotiation Process

Figure 3 below contains a block diagram of the PCS.

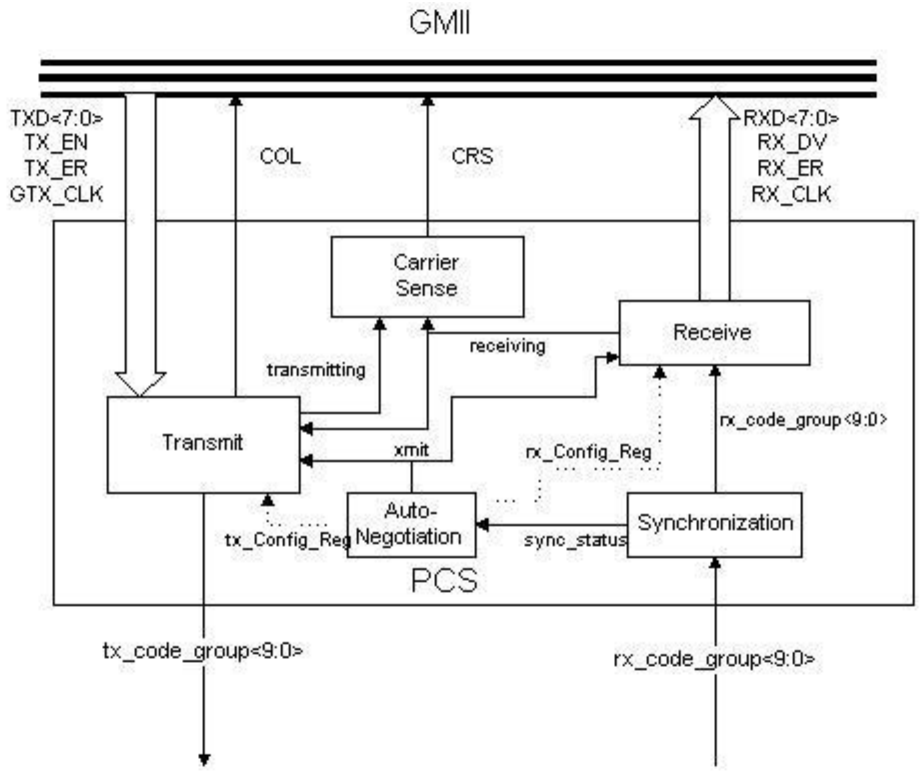

Figure 3: PCS Block Diagram

Services provided to the GMII:

- Encoding/Decoding of GMII data octets to/from ten-bit code groups (8B/10B) for communication within the underlying PMA.

- Carrier Sense (CRS) and Collision Detect (COL) indications.
- Managing the auto-negotiation process by informing it when it has lost synchronization of the received code_groups. Auto-negotiation can be instructed to restart if /C/ ordered_sets are received from the other station after the link has been established

A) PCS Synchronization Process

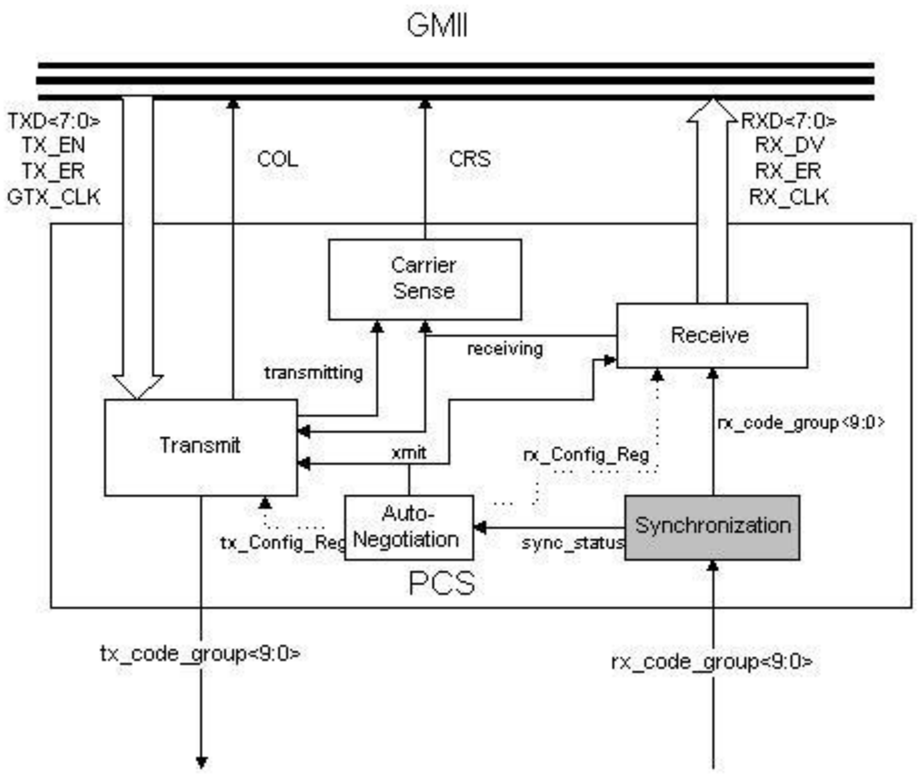

Figure 4: PCS Block Diagram with Synchronization Process highlighted

Overview

The purpose of the PCS synchronization process is to verify that the PMA is correctly aligning code groups from the serial stream it is receiving. Figure 36-9 defines how the synchronization process must operate. It is highly recommended that the reader have a hard copy of figure 36-9 handy when going through this portion of the tutorial.

Synchronization is acquired upon the reception of three ordered_sets each starting with a code_group containing a comma. Each comma must be followed by an odd number of valid data code_groups. No invalid code_groups can be received during the reception of these three ordered_sets.

Once synchronization is acquired, the synchronization process begins counting the number of invalid code_groups received. That count is incremented for every code_group received that is invalid or contains a comma in an odd code_group position. That count is decremented for every four consecutive valid code_groups received (a comma received in an even code group position is considered valid). The count never goes below zero and if it reaches four, sync_status is set to FAIL.

The following section discusses exactly how the PCS acquires synchronization based upon the code groups being received by the underlying PMA.

Acquiring Synchronization

After powering on or resetting, the PCS synchronization process does not have synchronization and it is in the LOSS_OF_SYNC state. The synchronization process looks for the reception of a /COMMA/ (a code group containing a comma). It then assigns that code group to be an evenly aligned code group. The next code group received is assigned to be an odd code group. Code groups received thereafter are alternately assigned to even and odd code group alignments. Table 2 demonstrates this process. The column in bold face type represents the first comma received by the synchronization process while it's in the LOSS_OF_SYNC state.

Table 2: Synchronization process: Reception of first comma

rx_code_group	**/K28.5/**	/D21.5/	/Dx.y/	/Dx.y/	/K28.5/	/D2.2/	/Dx.y/	/Dx.y/	/K28.5/	/D16.2/
code group alignment	**EVEN**	ODD	EVEN	ODD	EVEN	ODD	EVEN	ODD	EVEN	ODD

For convenience, we use the notation LOS to denote the LOSS_OF_SYNC state. Furthermore, we will use CDx to represent the state COMMA_DETECT_x, ASy to represent ACQUIRE_SYNC_y, and SAz to represent SYNC_ACQUIRED_z.

The process begins in the LOS state where sync_status is set to FAIL. When a code_group containing a comma is detected, the process transitions to the CD1 state, the variable rx_even is set to TRUE, and the next code_group is examined. If the code_group is a valid data code_group, the process transitions to the AS1 state and sets rx_even to FALSE. If the code_group is not a valid data code_group, the process returns to the LOS state.

While in the AS1 state, the process examines each new code_group. If the code_group is a valid data code_group, the process toggles the rx_even variable. If the code_group contains the comma character and rx_even is FALSE, the process transitions to the CD2 state and toggles rx_even. If the code_group does not satisfy either of these conditions, then the variable cgbad is asserted by the PCS and the process returns to the LOS state.

The same mechanism transports the process to the CD3 state or returns it to the LOS state. If the process enters the CD3 state and the next code group is a valid data code_group, the process will transition to the SA1 state where sync_status is set to OK. Otherwise, the process will return to the LOS state.

Thus, synchronization is achieved upon the reception of three ordered_sets each starting with a code_group containing a comma. Each comma must be followed by an odd number of valid data code_groups. No invalid code_groups can be received during the reception of these three ordered_sets. The following tables give examples of the state transitions that are made in the synchronization state machine.

Table 3: Acquiring synchronization with /I/ ordered_sets

code-group	/D/	/K28.5/	/D16.2/	/K28.5/	/D16.2/	/K28.5/	/D16.2/
state	LOS	CD1	AS1	CD2	AS2	CD3	SA1
rx_even		TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
sync_status	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	OK

Table 4: Acquiring synchronization with /C/ ordered_sets

code_group	/D/	/K28.5/	/D21.5/	/D0.0/	/D0.0/	/K28.5/	/D2.2/
state	LOS	CD1	AS1	AS1	AS1	CD2	AS2
rx_even		TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
sync_status	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL

code_group	/D0.0/	/D0.0/	/K28.5/	/D21.5/	/D0.0/	/D0.0/	/K28.5/
state	AS2	AS2	CD3	SA1	SA1	SA1	SA1
rx_even	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
sync_status	FAIL	FAIL	FAIL	OK	OK	OK	OK

Synchronization is acquired if three consecutive ordered sets which each begin with a /COMMA/ are received. The /COMMA/ must be followed by an odd number of valid /D/ code groups. The number of valid /D/?s following the /COMMA/ does not have an upper limit. The synchronization process moves to the SYNC_ACQUIRED_1 state and sets the flag sync_status=OK when synchronization is acquired. If at any time prior to acquiring synchronization the PCS receives a /COMMA/ in an odd code group or if it receives an /INVALID/, a code group that isn?t found in the correct running disparity column of tables 36-1 or 36-2, the PCS synchronization process returns to the LOSS_OF_SYNC state.

The following section defines how the PCS can lose synchronization once it has been acquired.

Maintaining and Losing Synchronization

For convenience, we use the notation LOS to denote the LOSS_OF_SYNC state, SAz to represent the SYNC_ACQUIRED_z state and SAzA to represent the SYNC_ACQUIRED_zA state. We assume that the synchronization process has reached the SA1 state (refer to discussion from test 36.1.1) and that sync_status has been set to OK.

While in the SA1 state, the PCS synchronization process examines each new code_group. If the code_group is a valid data code_group or contains a comma when rx_even is FALSE, the PCS asserts the variable cggood and the synchronization process toggles the rx_even variable. Otherwise, the PCS asserts the variable cgbad and the process moves to the SA2 state, toggles the rx_even variable, and sets the variable good_cgs to 0.

If the next code_group is a valid code_group which causes the PCS to assert the variable cggood, the process transitions to the SA2A state, toggles the rx_even variable, and increments good_cgs. Otherwise it continues on to the SA3 state.

While in the SA2A state, the process examines each new code_group. For each code_group which causes the PCS to assert cggood, the variable good_cgs is incremented. If good_cgs reaches three and if the next code_group received asserts cggood, the process returns to the SA1 state. Otherwise, the process transitions to the SA3 state.

Once in the SA3 state, the process may return to the SA2 state via the SA3A state using the same mechanisms that take the process from the SA2 state to the SA1 state. However, another invalid code_group or comma received when rx_even is TRUE will take the process to the SA4 state.

If the process fails to return to the SA3 state via the SA4A state, it will transition to LOS where sync_status is set to FAIL.

Thus, once sync_status is set to OK, the synchronization process begins counting the number of invalid code_groups received. That count is incremented for every code_group received that is invalid or contains a comma when rx_even is TRUE. That count is decremented for every four consecutive valid code_groups received (a comma received when rx_even is FALSE is considered valid). The count never goes below zero and if it reaches four, sync_status is set to FAIL.

Table 5: Loss of Synchronization

code_group		/K28.5/	/K28.5/	/D/	/D/	/D/
state	SA1	SA1	SA2	SA2A	SA2A	SA2A
rx_even	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
good_cgs			0	1	2	3
sync_status	OK	OK	OK	OK	OK	OK

code_group	/D/	/INVALID/	/INVALID/	/D/	/K28.5/	/INVALID/
state	SA1	SA2	SA3	SA3A	SA4	LOS
rx_even	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
good_cgs	3	0	0	1	0	0
sync_status	OK	OK	OK	OK	OK	FAIL

B) PCS Transmit Process

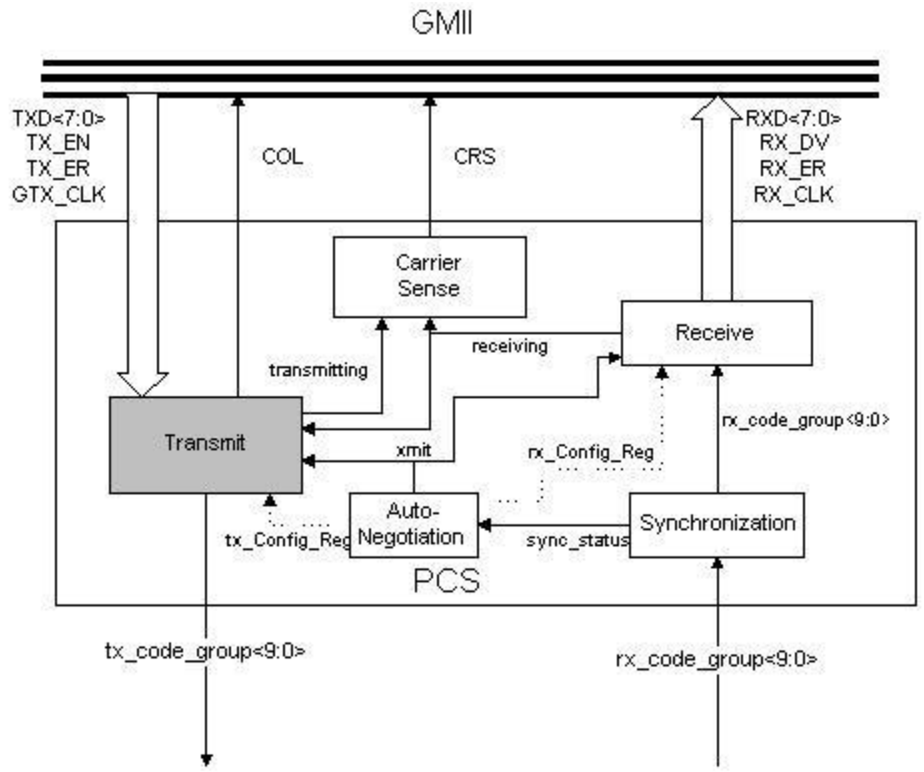

Figure 5: PCS Block Diagram with Transmit Process highlighted

The PCS transmit process is responsible for the 8B/10B encoding of the data octets from the GMII.

The PCS transmit process sends the signal 'transmitting' to the Carrier Sense function of the PCS whenever the PCS transmit process is sending out data packets. The signal 'receiving' is sent out by the PCS receive process whenever it is receiving packets. The PCS transmit process is responsible for checking to see if the PCS is both sending and receiving data. If (receiving = 1 AND transmitting = 1) then the collision signal COL is sent to the GMII.

The Carrier Sense mechanism of the PCS reports Carrier events to the GMII via the CRS signal. CRS is asserted when (receiving OR transmitting =1) and is de-asserted when (transmitting AND receiving =0). (Note: CRS is asserted for repeaters when receiving=TRUE and de-asserted when receiving=FALSE.)

PCS Transmit Data Format

Data packets are transmitted according to the following requirements:

- /I/ must be transmitted in an even code group position.
- /I/ must precede /S/ for the first packet of a burst of packets or the only packet of non-burst packets.

    - This means that /S/ will be transmitted in an even code group position if it follows /I/.
- /R/ must precede /S/ for the second and subsequent packets within a burst of packets.
- /S/ may be transmitted in an even or an odd code position when following /R/.

Table 6 demonstrates that /S/ must be transmitted in an even code group position when following /I/.

Table 6: Transmitting SPD

Ordered_set	xx	/I/		/S/	xx
Code group	xx	/K28.5/	/D16.2/	/K27.7/	xx
Alignment	xx	EVEN	ODD	EVEN	xx

Table 7 shows the EPD sequence when /T/ falls in an even code group position. In this case, /I/ will also fall in an even code group position.

Table 7: Transmitting SPD with /T/R/K28.5/

Ordered set	xx	/T/	/R/	/I/	
Code group	xx	/K29.7/	/K23.7/	/K28.5/	/D5.6 or D16.2/
Alignment	xx	EVEN	ODD	EVEN	ODD

Table 8 shows what happens when /T/ falls in an odd code group position. In this case, the normal /T/R/K28.5/ EPD will not suffice because it would cause /I/ to fall in an odd code group position.

Table 8: Transmitting EPD with /T/R/R/

Ordered Set	xx	/T/	/R/	/R/	/I/	
Code group	xx	/K29.7/	/K23.7/	/K23.7/	/K28.5/	/D5.6 OR D16.2/
Alignment	xx	ODD	EVEN	ODD	EVEN	ODD

Encodings of TXD<7:0>,TX_EN and TX_ER

When xmit=DATA the PCS transmit process is controlled by the GMII with the TX_EN, TX_ER and TXD<7:0> signals. Table 9 shows how the PCS transmit process interprets the values of these signals and the ordered sets transmitted upon their assertion.

TX_EN is the transmit enable signal which is asserted whenever a packet is being sent. If the GMII recognizes an error during data transmission, it will assert TX_ER as well to alert the PCS to send a /V/ code group. TX_ER is used to indicate a transmission error, except when transmitting extension.

TX_EN is de-asserted when the CRC field has been transmitted. Carrier extension is transmitted after the packet when TX_EN and TX_ER are both flipped so that TX_EN=0 and TX_ER=1 and TXD<7:0> = 0F. If TXD<7:0> is not equal to 0F, then the PCS considers it an error and transmits a /V/ ordered set.

If TX_EN=0 and TX_ER=0, then the PCS transmits IDLE.

Table 9: Use of TX_EN, TX_ER and TXD<7:0>

TX_EN	**TX_ER**	**TXD<7:0>**	**Indication**	**Ordered_sets**
0	0	00 through FF	Normal Interframe	/I/
0	1	0F	Carrier Extend	/R/
0	1	1F	Carrier Extend Error	/V/
0	1	00 through FF(except 0F and 1F)	Reserved (Interpreted as Carrier Extend Error)	/V/
1	0	XX	Normal Data Transmission	/S/D/T/R/
1	1	XX	Transmit Error Propagation	/V/

C) PCS Receive Process

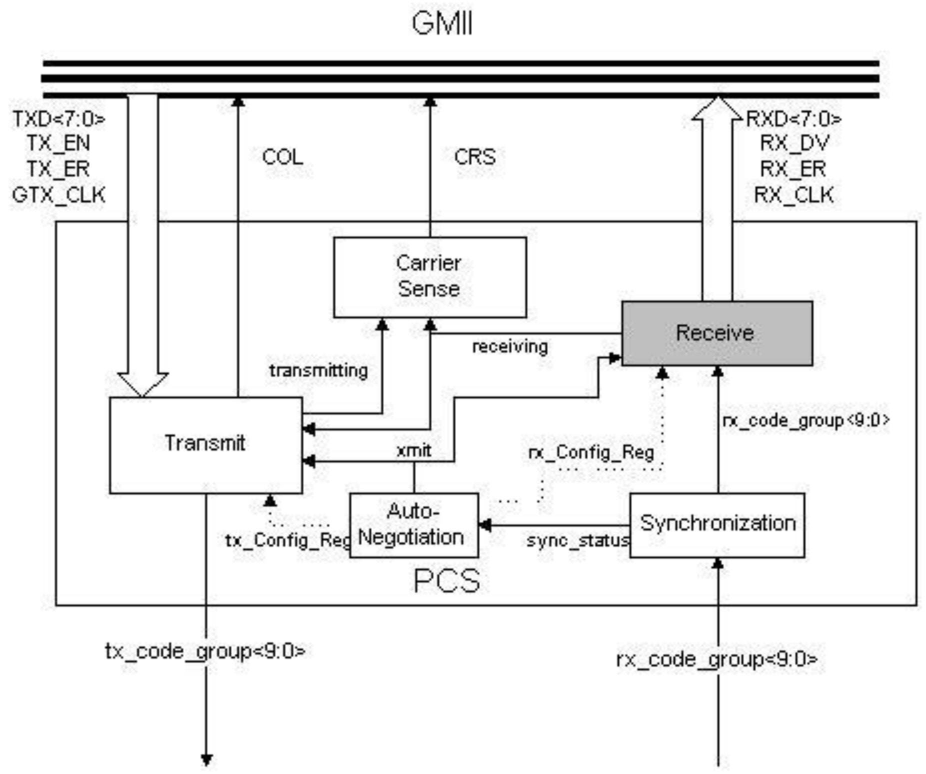

Figure 6: PCS Block Diagram with Receive Process highlighted

The PCS receive process is responsible for decoding the incoming 10-bit code groups into their corresponding octets for transmission by the GMII. The receive process sends out the signal 'receiving' to the PCS transmit process and the Carrier Sense process for use in notifying the upper layers that there is carrier on the line or that a collision has occurred.

- Carrier Detection is used by the MAC for deferral purposes and by the PCS transmit process for collision detection. A carrier event is signaled by the assertion of 'receiving'. The signal 'receiving' is asserted upon the reception of a code group with at least a two bit difference from the /K28.5/ code group for code groups received in an even code group position.
    - /K28.5/ = 101 11100
    - /K27.7/ = 111 11011
- False Carrier is defined as a carrier event packet not beginning with /S/. The receive process replaces the beginning code group with 0E and asserts RX_ER when a false carrier event is received.

The incoming code groups are checked for the existence of /INVALID/ data or mis-aligned /COMMA/s. The receive process also checks to see if an /I/ or /C/ ordered set is received prior to receiving the EPD. If this occurs then the packet has ended early because once the /S/ has been received the receive process expects to see a series of /D/ code groups followed by the /T/R/R/ or /T/R/I/. If /I/ or /C/ arrived prior to /T/ then an error has occurred.

The /K28.5/ code group is used at the beginning of the /C/ and /I/ ordered sets. While the PCS receive process is receiving a packet, it expects to see /T/ prior to seeing the next /K28.5/. A one bit error in some of the data code groups can cause them to become /K28.5/. If this one bit error happens to be followed by /D21.5/ or /D2.2/ then the beginning of a /C/ ordered_set will have been created. To account for the possibility that a one bit error has occurred, the PCS receive process checks the received code groups three code groups at a time. Doing this allows the receive process to treat /K28.5/ received in an odd code group position as an /INVALID/ code group rather than assuming a /C/ or /I/ ordered_set had been received. If /K28.5/ is received in an even code_group position and is followed by /D21.5/ or /D2.2/ the receive process assumes a /C/ ordered _set is being received. If these two code_groups are followed by two more /D/ code_groups, the receive process indicates to the auto-negotiation process that a /C/ ordered_set has indeed been received via the RX_UNITDATA.indicate(/C/) signal.

Encodings of RXD<7:0>,RX_DV and RX_ER

Table 10: Use of RX_DV, RX_ER and RXD<7:0>

RX_DV	RX_ER	RXD<7:0>	Indication	Ordered Sets
0	0	00 through FF	Normal Inter-frame	/I/
0	1	00	Normal Inter-frame	/I/
0	1	01-1D	Reserved	-
0	1	0E	False Carrier	-
0	1	0F	Carrier Extend	/R/
0	1	10 through 1E	Reserved	-
0	1	1F	Carrier Extend Error	-
0	1	20 through FF	Reserved	-
1	0	00 through FF	Normal Data Reception	/S/D/T/R/
1	1	00 through FF	Data REception Error	/V/

D) Auto-Negotiation Process

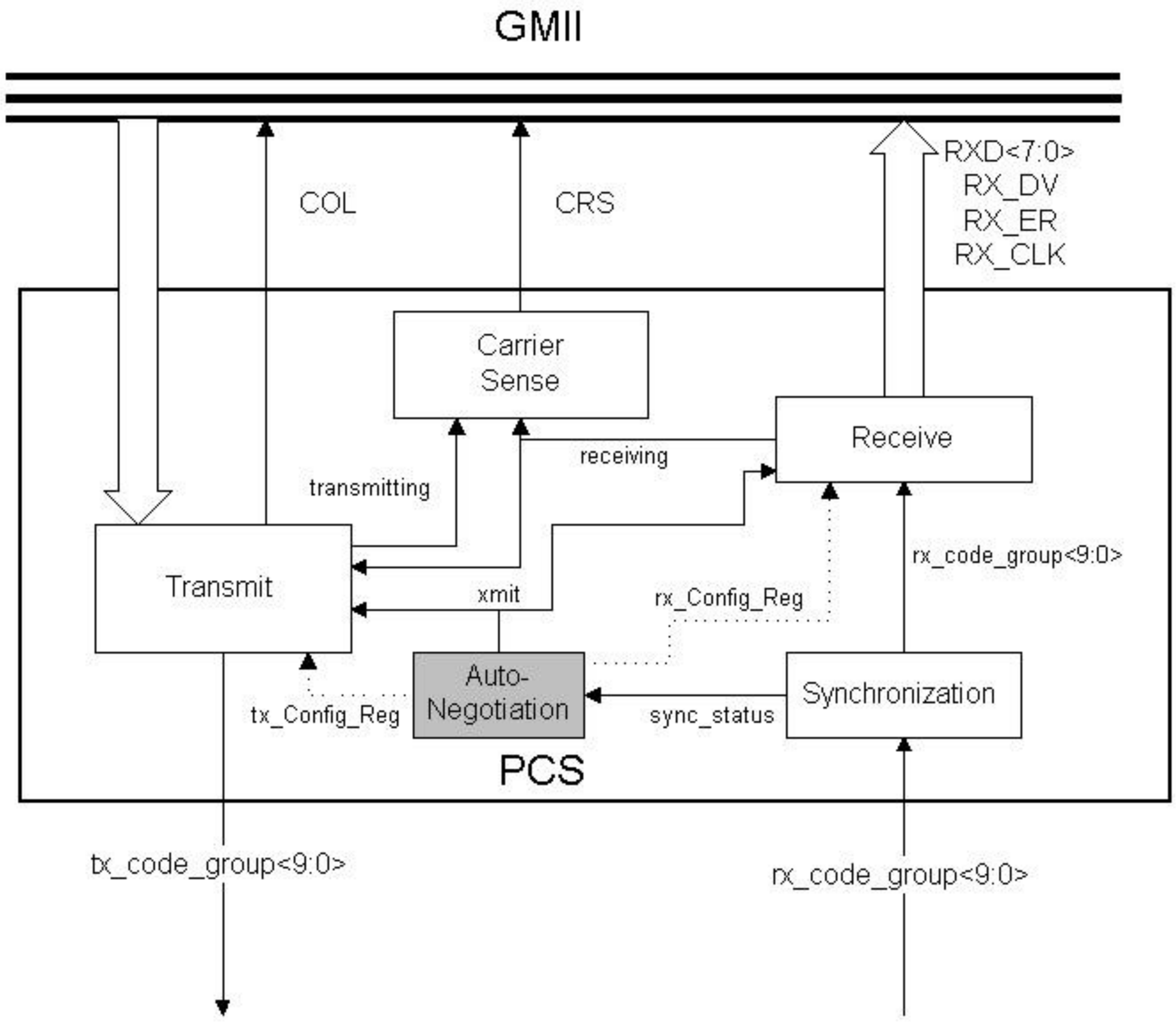

Figure 7: PCS Block Diagram with Auto-Negotiation Process highlighted

The Auto-Negotiation Process is responsible for determining a common set of abilities supported by both the local device and the link partner. The auto-negotiation process is defined in Clause 37 of the IEEE 802.3 Standard and so is out of the scope of this tutorial. This process can only be called if the synchronization process has been completed and, once called, it controls what is transmitted by the Transmit Process (xmit is set to 'configuration'). Once the auto-negotiation process has been completed the PCS Transmit process can transmit frame passed down from the MAC via the GMII (xmit is set to data).

IV) Physical Medium Attachment (PMA) sublayer Fundamentals

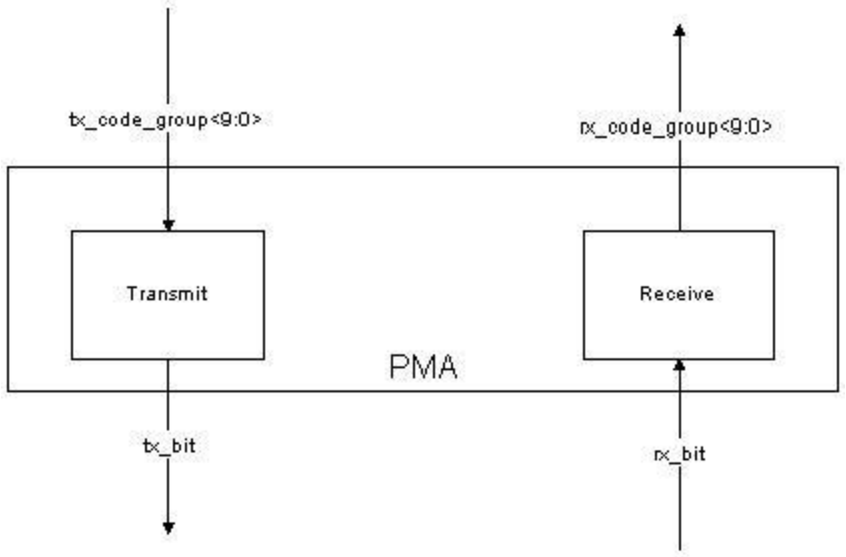

Figure 8: PMA Block Diagram

A) PMA Functions

The PMA sublayer is responsible for aligning the incoming serial stream of data. The PMA receive process is allowed to lose up to four 10-bit code groups in the alignment process. This is referred to as code slippage. The PMA receive process aligns the incoming code groups by finding /COMMA/s in the data stream and aligning the subsequent code groups according to the /COMMA/. Alignment by the PMA is essential if the PCS receive process is to operate correctly. If misaligned /COMMA/s and/or data are repeatedly sent by the PMA, synchronization cannot be obtained by the PCS.

The PMA is responsible for the serialization of the incoming 10-bit code groups from the PCS. This is accomplished through the use of a 10x clock multiplier. The PCS transmits the code groups in parallel at 125 MHz and the PMA sends them out bitwise at a rate of 1.25 GHz.

The PMA is also responsible for the deserialization of the data coming in from the PMD. The data arrives serially at a rate of 1.25 GHz and is transmitted in parallel to the PCS at a rate of 125 MHz.

The PMA is also responsible for recovering the clock from the incoming data stream. A Phased Lock Loop circuit is included in both the PMA receive and transmit circuitry.

B) Start-up Protocol

The PCS start-up protocol for auto-negotiating devices can be divided into two components:

1. Synchronization Process
2. Auto-Negotiation Process

Auto-negotiating, and manually configured, devices are unable to interpret any received code_group until synchronization has been acquired. Once synchronization has been acquired the PCS is then able to receive and interpret the incoming code_groups.

Auto-negotiating devices begin with the variable xmit set to CONFIGURATION. Before data transmission can begin, auto-negotiating devices first need to receive three consecutive, consistent /C/ ordered_sets. Consistent /C/ ordered_sets must contain the same code_groups within the last two code_groups of each /C/ ordered_set, (ignoring the ACK(nowledge bit). Once three consecutive, consistent /C/ ordered_sets have been received the auto-negotiation process looks for three consecutive, consistent /C/ ordered_sets which have the ACK bit set to 1. After a period of time the variable xmit is set to IDLE at which point the device begins transmitting /I/ ordered_sets After a specific amount of time xmit is set to DATA and the auto-negotiating device is then able to transmit and receive data, assuming the partner device also received three consecutive, consistent /C/ ordered_sets followed by three consecutive, consistent /C/ ordered_sets with the ACK bit set to 1.

Figure 9 contains a time-space diagram outlining the process. (Note: manually configured devices skip the process of transmitting /C/ ordered_sets and begin with xmit=DATA.)

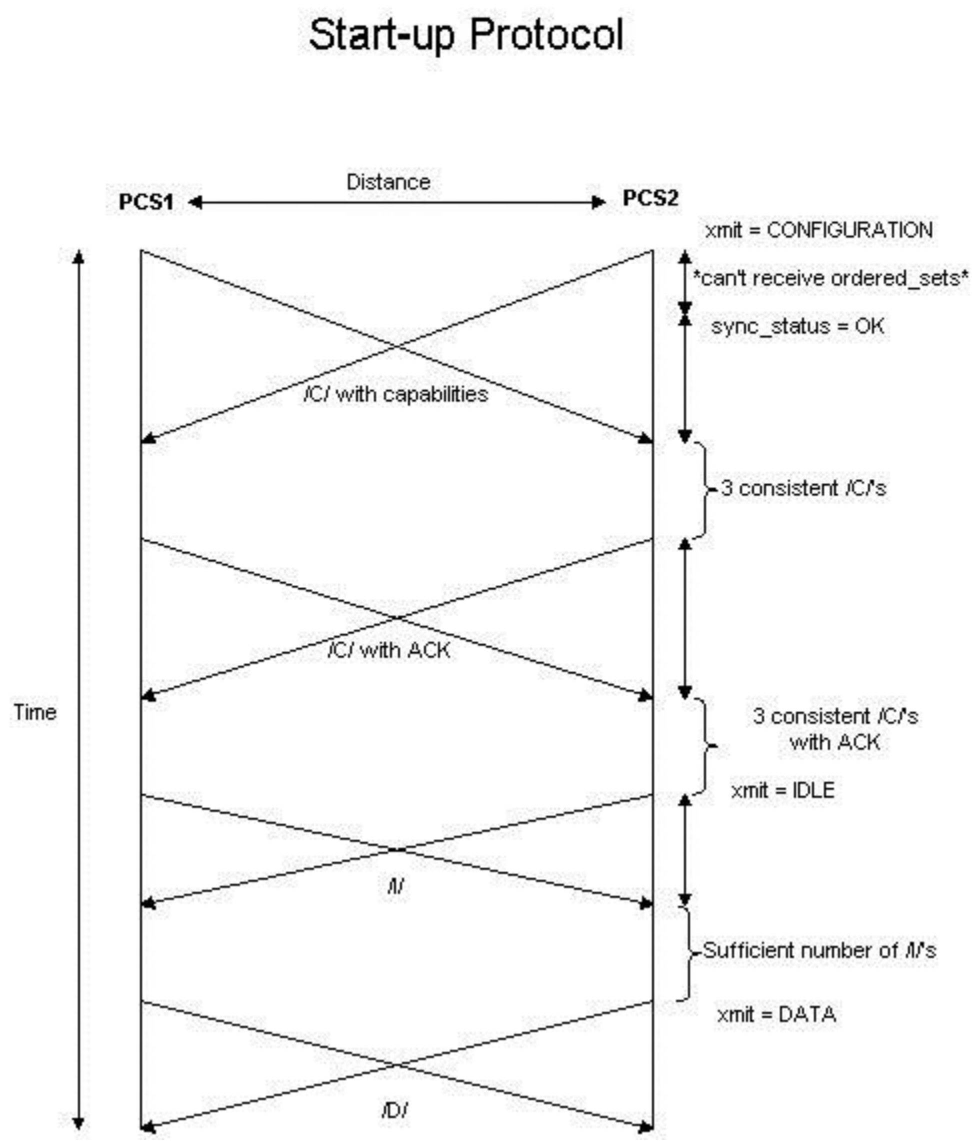

Figure 9: Start-up Protocol

V) Conclusion

Whether a device is an auto-negotiating device or a manually configured device, synchronization is the heart of the entire PCS transmit and receive process. The PMA is responsible for monitoring the incoming serial bit stream and aligning it into code_groups. If the PMA fails to recognize and properly align commas, the PCS will be unable to receive any code_groups from its link partner.

Manually configured devices require properly aligned commas in order to acquire and maintain synchronization from the incoming /I/ ordered_sets. Auto-negotiating devices have the added burden of needing to recognize /C/ ordered_sets and differentiating them from /I/ ordered_sets.

Prior to transmitting data, auto-negotiating devices are required to receive a certain number of consecutive, consistent /C/ ordered_sets. This is essential in order to ensure that both auto-negotiating devices will be communicating on the same terms.

Once synchronization has been acquired and the communication protocol has been determined between the link partners, the devices are allowed to transmit data to one another. To help ensure that the devices will be able to correctly interpret the received code_groups, an 8B/10B encoding scheme is utilized. Such a coding scheme along with the use of running disparity allows for a predictable and controlled set of code_groups that will be transmitted across the channel. Because of the number of extra code_groups created by the encoding scheme, special control words can be implemented for use in the interpacket gap, start-of and end-of packet delimiters, and configuration ordered_sets among others.