Gigabit Ethernet 8B/10B PCS, Change Summary, Rev 5

- Summary of PCS functions
- Proposed modifications to 8B/10B PCS protocol by R. Taborek dated September 09, 1996
  - Protocol
    - F code elimination to simplify Link_Startup
    - C code changes to support all SERDES/10-bit comma detect
  - Automatic Configuration
    - Remote Fault expansion to 2 bits including Offline mode
    - Pause Capability bits (2) reserved
  - Rules
    - H code; Sequence recognition; Link error conditions

November 11-13 Interim Meeting, Hotel Vancouver, Vancouver, BC, Canada

Protocol Changes

- Comma Support
  - Enable usage of SERDES/10-bit parts which sync on comma+ only as well as those which sync on comma+ and comma-;
  - Pathological problem with C codes;
  - ***Proposal:*** Redefine C code as set C1 and C2 which alternate.

- 'F' Code Elimination
  - Original Link_Startup code (F) is ineffectual since C code was added to Link_Startup;
  - Eliminating F code from Link_Startup simplifies protocol and conveys Config_Reg to link partner immediately;
  - ***Proposal:*** Delete F code and associated state.

Automatic Configuration Changes

- Remote Fault Support
  - Expand RF field in Config_Reg base register to use two bits;
  - Add Test/Maintenance/Offline mode support to Link_Startup;
  - Extra bit loss mitigated by encoding Offline mode indication;
    - ***Proposed encoding:***

      00 - No error, link OK
      01 - Offline
      10 - Link Failure
      11 - Link Error

- Pause Capability Support
  - Support position reserved in PCS pending resolution of flow control support requirements;
  - ***Proposal:*** Two Config_Reg base register bits reserved (D7 & D8).

PCS Rules

- H Code Usage
  - ***Proposal:*** The first symbol in error and all subsequent symbols during carrier are replaced with H.

- Sequence Recognition
  - How many consecutive ordered sets should a receiver have to recognize before a sequence is?
    - (e.g., How many C's should be recognized before a receiver responds with C/Ack);
  - ***Proposal:*** 3.

PCS Link Error Rules

- Link Errors
  - *Proposal:* Three types of link errors are defined:
    - a) Link_Failure (severe link error);
      - 1) Loss_of_Synchronization failure;
      - 2) Loss_of_Signal;
    - b) Link_Active_Error: a link_error associated with the contents of a packet or carrier extension;
    - c) Link_Inactive_Error: a link_error not associated with the contents of a packet or carrier extension.
  - *Proposal:* Recovery for link errors:
    - a) Link_Startup for Link_Failure;
    - b) Link_Synchronization procedure for all other errors.

Defined Ordered Sets

Code	Function	Encoding	Beg. RD	End RD
~~F~~	~~Link_Not_Available~~	~~K28.5 D21.5~~	~~?~~	~~flip~~
C1	Link_Configuration 1	K28.5 D21.5 config_reg	?	flip*
C2	Link_Configuration 2	K28.5 D2.2 config_reg	?	same*
I1	Idle/Flip Disparity	K28.5 D5.6	+	-
I2	Idle/Disparity OK	K28.5 D16.2	-	-
S	SOP	K27.7	?	same
T	EOP1	K29.7	?	same
R	EOP2	K23.7	?	same
H	EOPinvalid	K30.7	?	same

* Ending RD not based on config_reg value

Ordered Set Coding Distance

Char	D21.5	D2.2	D16.2
D2.2	7	↓	↓
D16.2	7	4	↓
D5.6	4	7	4

- Data characters used in defined ordered sets are chosen for high transition density, proper disparity control, and sufficient coding distance.

Config_Register Base Register

D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
					FD	**HD**	**PS1**	**PS2**				**RF1**	**RF2**	**ACK**	**NP**

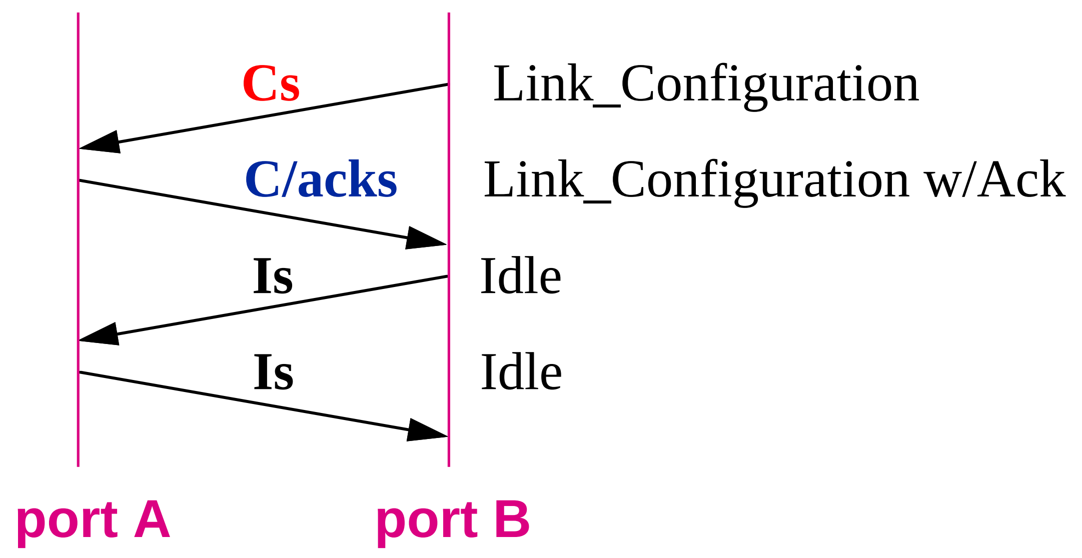
Config_Register Base Register bit usage:

D5/FD:	Full duplex capable
D6/HD:	Half duplex capable
D7,8/PS1,2:	Pause function
D12,13/RF1,2:	Remote Fault function $\longrightarrow$ Remote Fault function encoding
D14/ACK:	Acknowledge
D15/NP:	Next Page (Escape)

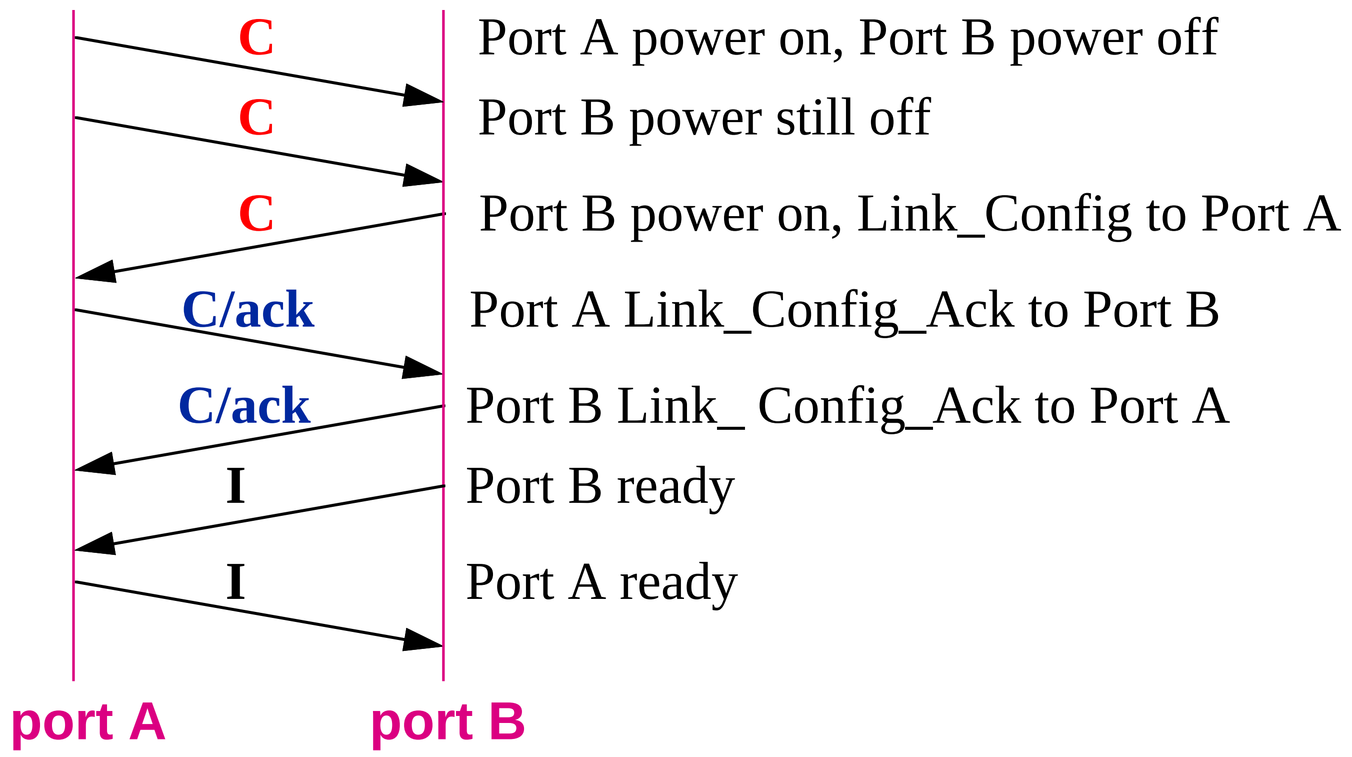
Remote Fault function encoding

00 - No error, link OK
01 - Offline
10 - Link Failure
11 - Link Error

Link_Startup Procedure



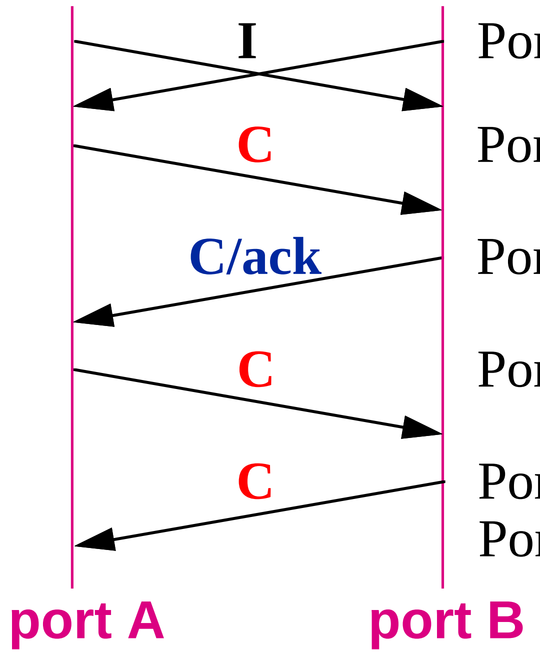
Power-On Procedure
One port powers on first



C
Port A power on, Port B power off
C
Port B power still off
C
Port B power on, Link_Config to Port A
C/ack
Port A Link_Config_Ack to Port B
C/ack
Port B Link_ Config_Ack to Port A
I
Port B ready
I
Port A ready
port A
port B

Power-Off Procedure

One port powers off



Ports active

Port A Link_Configuration w/Offline

Port B Link_Config_Ack of Port A Offline

Port A signals Offline for a minimum time

Port A powered off;
Port B awaits Link_Startup from Port A

Link Information Example

Part 1/2

802.3 info	8B10B codes	802.3 info	8B10B codes
Link_Configuration	C1	FCS3	Data = FCS3 octet...
Link_Configuration	C2	FCS4	Data = FCS4 octet...
Link_Configuration	C1	EOP1	T
Link_Configuration	C2	EOP2	R
~	~	EOP2	R
Link_Configuration	C1	~	~
Link_Configuration	C2	EOP2	R (odd-num character)
Idle/Disparity OK	I2	Idle/Flip Disparity	I1
Idle/Disparity OK	I2	Idle/Disparity OK	I2
~	~	~	~
Idle/Disparity OK	I2	Idle/Disparity OK	I2
0 PREAMBLE	S	Link_Configuration	C1 (error or reconfig)
1 PREAMBLE	Data = 10101010	Link_Configuration	C2
2 PREAMBLE	Data = 10101010	Link_Configuration	C1
3 PREAMBLE	Data = 10101010	Link_Configuration	C2
4 PREAMBLE	Data = 10101010	~	~
5 PREAMBLE	Data = 10101010	Link_Configuration	C1
6 PREAMBLE	Data = 10101010	Link_Configuration	C2
7 SFD	Data = 10101011	Idle/Flip Disparity	I1
8 DA	Data = DA...	Idle/Disparity OK	I2
~	~	Idle/Disparity OK	I2
LLC DATA	Data = LLC DATA...	Idle/Disparity OK	I2
~	~	Idle/Disparity OK	I2
FCS1	Data = FCS1 octet...	Idle/Disparity OK	I2
FCS2	Data = FCS2 octet...	Idle/Disparity OK	I2

Link Information Example

Part 2/2

802.3 info	8B10B codes
Idle/Disparity OK	I2
~	~
Idle/Disparity OK	I2
0 PREAMBLE	S
1 PREAMBLE	Data = 10101010
2 PREAMBLE	Data = 10101010
3 PREAMBLE	Data = 10101010
4 PREAMBLE	Data = 10101010
5 PREAMBLE	Data = 10101010
6 PREAMBLE	Data = 10101010
7 SFD	Data = 10101011
8 DA	Data = DA...
~	~
LLC DATA	Data = LLC DATA...
~	~
FCS1	Data = FCS1 octet...
FCS2	Data = FCS2 octet...
FCS3	Data = FCS3 octet...
FCS4	Data = FCS4 octet...
EOP1	T (even char/no extension)
EOP2	R
Idle/Flip Disparity	I1
Idle/Disparity OK	I2
~	~

Summary

- 8B/10B codes and the proposed coding structure is **efficient, robust, and flexible** enough to meet **Gigabit Ethernet PCS requirements**.
- No other 8B/10B changes are planned.
- 8B/10B PCS draft generation is underway.

Idle addition/removal

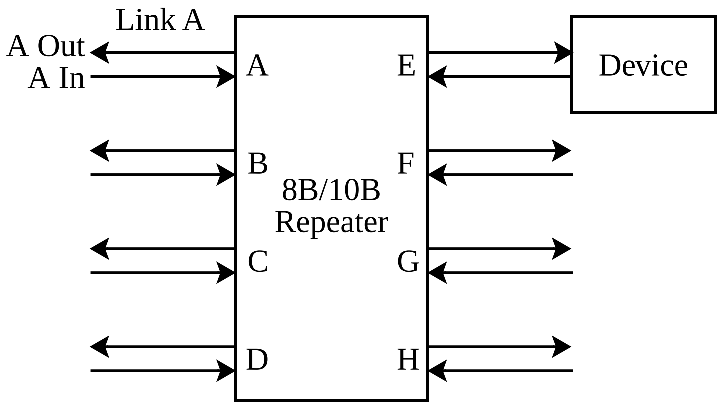
Consensus from 802.3z and Gigabit Ethernet Alliance committee and reflector discussion indicates that there is a requirement to allow implementation of repeaters which both include and do not include 8B/10B encoders/decoders (ENDECs). This being the case, there is a requirement to be able to add/remove idles in ENDEC-less repeaters.

The primary issue for raising this issue is 8B/10B protocol simplification. I had hoped to reduce the set of the idle codes which stabilizes running disparity and eliminate the requirement for the PCS to determine whether to send I1 or I2 after packets or other non-idle events.

After examining 8B/10B and ENDEC-less PCS issues in detail, I conclude that the simplest solution is to retain both flavors of idles. Read on if you’re interested in the analysis.

All 8B/10B-based repeaters, whether ENDEC or ENDEC-less have the following characteristics:

- Each link (e.g., Link A) consists of an inbound and outbound fibre. The fibre may be fiber-optic or copper media;
- Each incoming fibre (e.g., A_In) is driven by and in phase with the transmitter of the device at the other end of the link;
- The running disparity of all 10-bit codes on each incoming fibre is specified by the PCS layer of the device at the other end of the link;
- Each outgoing fibre (e.g., A_Out) is driven by and in phase with the transmitter of the repeater (note that all transmitters of the repeater may use master or individual clocks);
- The running disparity of 10-bit codes on each outgoing fibre is specified by the individual repeater port;



For 8B/10B purposes, the repeater sends idles on all outgoing fibres after Link_Startup and prior to the transmission of any packets. When a packet is received on any incoming fibre, that packet is repeated on all outgoing fibres except for the one associated with packet’s incoming link. All other events are handled in a similar manner. Link-specific events such as Link_Startup are not propagated from incoming link to any other link.

A speed-matching or elasticity buffer is used to compensate for clock rate differences between the incoming link and repeater outgoing link(s) by adding or removing 8-bit codes or 10-bit characters from the buffer only when traffic is flowing between a repeater incoming link and one or more outgoing links.

<u>8B/10B ENDEC REPEATER</u>

An 8B/10B ENDEC repeater is one which decodes the incoming 10-bit stream to 8-bit data and special codes, goes through some above-PCS-level logic, a speed-matching buffer, some more logic, encodes the outgoing 8-bit data and special codes according to the current running disparity of the outgoing bit stream and sends out a 10-bit stream. The encoder associated with each port insures that running disparity is continuously maintained, regardless of the input source (internal to the repeater).

Since an 8B/10B ENDEC repeater actually regenerates all 10-bit codes to be in sync with the current running disparity of its outgoing links, idle addition/removal required for speed-matching the system transmitting source and repeater outgoing link is a simple matter of adding/deleting 8-bit idle CODEs, rather than idle CHARACTERs. As a result, only a single idle is required since outgoing link running disparity is always properly generated by the ENDEC associated with the outgoing link.

<u>8B/10B ENDEC-LESS REPEATER</u>

An 8B/10B ENDEC-less repeater is one which the incoming bit stream goes through some simple logic, a speed-matching buffer, some more logic and is sent out as a 10-bit stream which is in sync with the current running disparity of the outgoing bit stream. No encoder associated with each port is present to insure that running disparity is continuously maintained, regardless of the input source (internal to the repeater).

I'm not even going to mention the possibility that some clever implementers can come up with a way to continuously maintain outbound link running disparity, regardless of the input source or running disparity thereof, or that it’s not important to do so, and OK to just knowingly transmit a stream containing running disparity errors. Ooops... to late... I said it... but please ignore.

I’ll take a moment here to point out that an 8B/10B ENDEC-less repeater is a reasonable implementation (I’m not making a marketability assessment here). Support functions such as F and C code protocol and auto-configuration can be handled by directly using the 10-bit code versions of the 8-bit Link_Startup codes.

In order to simplify 8B/10B ENDEC-less repeater implementation, it is advantageous to insure that the running disparity of the outgoing bit stream is in sync with that of any source. This is currently accomplished by stabilizing the running disparity soon after all events, such as Link_Startup and packets, by utilizing the two currently defined idle codes, I1 and I2. I1 changes the running disparity from + to - and I1 maintains the running disparity at -. Simplification of 8B/10B protocol by eliminating the correcting idle code (I1) would serve to complicate 8B/10B ENDEC-less repeater implementation, or, at a minimum, require a different running disparity stabilization scheme.

RUNNING DISPARITY STABILIZATION

As stated above the current scheme to stabilize bit stream running disparity is accomplished by utilizing the two currently defined idle codes, I1 and I2. I1 changes the running disparity from + to - and I1 maintains the running disparity at -.

Alternatively, it has been suggested that running disparity may be stabilized by forcing event ending running disparity to a known value. This means that packets, C’s, and any other events which either end or abort must do so with known ending running disparity. I maintain that event abort precludes a simpler protocol than running disparity correction, which I1/I2 currently provides. For example, if a packet is aborted early due to collision, running disparity control is most easily accomplished by running disparity correction via I1/I2, no other symbols are appropriate at this time.

Multiple T or R codes may be specified which cause a good packet to end with negative running disparity. In addition, C codes may be re-specified/restricted to force negative ending running disparity. However, a scheme such as I1/I2 would still be required to handle event abort situations. Therefore, it is simpler to utilize the I1/I2 scheme to stabilize running disparity in all instances and no change to the currently proposed Idle codes are suggested.

C-code Running Disparity Control/Comma Support

Fibre Channel uses only the negative beginning running disparity (RD) form of the K28.5 character. The FC 10-bit spec requires that alignment on only the seven-bit comma+ sequence b’001111’ which is contained in the negative RD version of the K28.5 character (-K28.5). Alignment on comma- (b’1100000’) or either the positive or negative RD version of the K28.5 character is optional.

Gigabit Ethernet uses both forms of K28.5, otherwise codes used must be expanded or restricted to better control RD. A Pathological case exist during Link_Startup where a transmitter may not send -K28.5 for an extended period of time, preventing a receiver from acquiring synchronization in certain error situations.

Several FC 10-bit compliant component vendors have been identified to be exposed to the above pathological case. Unless a change is made to Gigabit Ethernet C-code definition, those vendors parts will be exposed to known error conditions and would be considered inferior to other vendors parts which support both comma+ and comma- alignment. It was never my intention to exclude these components for use in reliable Gigabit Ethernet implementations.

The proposal below suggests a change to C-code definitions to eliminate this exposure and enable the usage of 10-bit parts which align on comma+ only as well as those which align on comma+ and any of the other optional alignment codes (i.e., comma-, -K28.5, or +K28.5).

Proposal: Redefine C as Ca+Cb where: Ca has two flavors Ca1 and Ca2

- Ca1 = K28.5 Dx.y(flip RD)
- Ca2 = K28.5 Dx.y(same RD)
- Cb = config_reg

The C sequence alternates as Ca1+Cb and Ca2+Cb forcing the transmission of comma+ on half the C codes. The proof is as follows:

1. All 8B/10B special and data characters either flip or maintain the running disparity at +1 (+ or positive) or -1 (- or negative) at the end of a transmission character boundary;
2. Some 10-bit parts only sync on comma+;
3. For K28.5, if beginning RD is +, comma- is sent; if beginning RD is -, comma+ is sent;
4. Cb = config_reg = Cb1+Cb2 = config_reg bits D0:D7 + config_reg bits D8:D15;
5. For all combinations of Cb1 and Cb2:
   if beginning RD is positive, then:
   ```
   +Cb1+Cb2+ = +Cb+; or
   +Cb1+Cb2- = +Cb-; or
   +Cb1-Cb2- = +Cb-; or
   +Cb1-Cb2+ = +Cb+;
   ```
   if beginning RD is negative, then:
   ```
   -Cb1-Cb2- = -Cb-; or
   -Cb1-Cb2+ = -Cb+; or
   -Cb1+Cb2+ = -Cb+; or
   -Cb1+Cb2- = -Cb-;
   ```

6. For all combinations of Ca,Cb running disparity is:
```
...+Ca1-Cb+Ca2+Cb-Ca1+Cb-Ca2-Cb+Ca1-...;
...+Ca1-Cb-Ca2-Cb-Ca1+Cb+Ca2+Cb+Ca1-...;
...-Ca1+Cb-Ca2-Cb+Ca1-Cb+Ca2+Cb-Ca1+...;
...-Ca1+Cb+Ca2+Cb+Ca1-Cb-Ca2-Cb-Ca1+...;
```
7. Comma+ is transmitted for two consecutive C’s followed by comma- for two consecutive C’s for the duration of the C sequence.