# 64b/66b PCS

**updated 6/30/2000**

**state machines modified 7/17/2000**

<table>
<thead>
<tr>
<th>Name</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rick Walker</td>
<td>Agilent</td>
</tr>
<tr>
<td>Richard Dugan</td>
<td>Agilent</td>
</tr>
<tr>
<td>Birdy Amrutur</td>
<td>Agilent</td>
</tr>
<tr>
<td>Rich Taborek</td>
<td>nSerial</td>
</tr>
<tr>
<td>Don Alderrou</td>
<td>nSerial</td>
</tr>
<tr>
<td>John Ewen</td>
<td>IBM</td>
</tr>
<tr>
<td>Mark Ritter</td>
<td>IBM</td>
</tr>
<tr>
<td>Al Bezoni</td>
<td>Lucent</td>
</tr>
<tr>
<td>Drew Plant</td>
<td>Agilent</td>
</tr>
<tr>
<td>Howard Frazier</td>
<td>Cisco</td>
</tr>
<tr>
<td>Paul Bottorff</td>
<td>Nortel</td>
</tr>
<tr>
<td>Shimon Mueller</td>
<td>Sun</td>
</tr>
<tr>
<td>Brad Booth</td>
<td>Intel</td>
</tr>
<tr>
<td>Kevin Daines</td>
<td>World Wide Packets</td>
</tr>
<tr>
<td>Osamu Ishida</td>
<td>NTT</td>
</tr>
<tr>
<td>Jason Yorks</td>
<td>Cielo</td>
</tr>
<tr>
<td>Henning Lysdal</td>
<td>Giga/Intel</td>
</tr>
<tr>
<td>Justin Chang</td>
<td>Quake</td>
</tr>
</tbody>
</table>
Topics

- Code review and update
- Test vectors
- Bit ordering sequence
- Frame sync algorithm and state machine
- TX,RX error detection state machines
- Optional code features
- Summary
### Building frames from 10GbE RS symbols

<table>
<thead>
<tr>
<th>octet 0</th>
<th>octet 1</th>
<th>octet 2</th>
<th>octet 3</th>
</tr>
</thead>
</table>

- **S, T** = SOP, EOP
- **I, E** = control words (Z)
- **D** = Data octets

Pure data:

<table>
<thead>
<tr>
<th>D D D D</th>
</tr>
</thead>
<tbody>
<tr>
<td>D D D D</td>
</tr>
<tr>
<td>D D D D</td>
</tr>
<tr>
<td>D D D D</td>
</tr>
</tbody>
</table>

Pure control:

<table>
<thead>
<tr>
<th>Z Z Z Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z Z Z Z</td>
</tr>
<tr>
<td>Z Z Z Z</td>
</tr>
<tr>
<td>Z Z Z Z</td>
</tr>
</tbody>
</table>

Two possible packet startings:

<table>
<thead>
<tr>
<th>S D</th>
<th>Z S</th>
</tr>
</thead>
<tbody>
<tr>
<td>D D</td>
<td>Z D</td>
</tr>
<tr>
<td>D D</td>
<td>Z D</td>
</tr>
<tr>
<td>D D</td>
<td>Z D</td>
</tr>
</tbody>
</table>

Eight possible packet endings:

<table>
<thead>
<tr>
<th>T Z</th>
<th>D Z</th>
<th>D Z</th>
<th>D Z</th>
<th>D T</th>
<th>D D</th>
<th>D D</th>
<th>D D</th>
</tr>
</thead>
</table>
Code Overview

Data Codewords have “01” sync preamble

0 1

64 bit data field (scrambled)

Mixed Data/Control frames are identified with a “10” sync preamble. Both the coded 56-bit payload and TYPE field are scrambled

1 0

8-bit TYPE

combined 56 bit data/control field (scrambled)

00,11 preambles are considered code errors and cause the packet to be invalidated by forcing an error (E) symbol on coder output
### Code Summary

<table>
<thead>
<tr>
<th>Input Data (first RS transfer / second RS transfer)</th>
<th>Sync [0] [1]</th>
<th>Bit fields</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[2] [65]</td>
<td></td>
</tr>
<tr>
<td>(D_0D_1D_2D_3/D_4D_5D_6D_7)</td>
<td>0 1</td>
<td>(D_0) [0] [7]</td>
</tr>
<tr>
<td>(Z_0Z_1Z_2Z_3/Z_4Z_5Z_6Z_7)</td>
<td>1 0</td>
<td>0x1e C_0 [0] [6]</td>
</tr>
<tr>
<td>(Z_0Z_1Z_2Z_3/S_4D_5D_6D_7)</td>
<td>1 0</td>
<td>0x33 C_0 [0] [6]</td>
</tr>
<tr>
<td>(S_0D_1D_2D_3/D_4D_5D_6D_7)</td>
<td>1 0</td>
<td>0x78 D_1 [0] [7]</td>
</tr>
<tr>
<td>(T_0Z_1Z_2Z_3/Z_4Z_5Z_6Z_7)</td>
<td>1 0</td>
<td>0x87 C_1 [0] [6]</td>
</tr>
<tr>
<td>(D_0T_1Z_2Z_3/Z_4Z_5Z_6Z_7)</td>
<td>1 0</td>
<td>0x99 D_0 [0] [7]</td>
</tr>
<tr>
<td>(D_0D_1T_2Z_3/Z_4Z_5Z_6Z_7)</td>
<td>1 0</td>
<td>0xaa D_0 [0] [7]</td>
</tr>
<tr>
<td>(D_0D_1D_2T_3/Z_4Z_5Z_6Z_7)</td>
<td>1 0</td>
<td>0xb4 D_0 [0] [7]</td>
</tr>
<tr>
<td>(D_0D_1D_2D_3/T_4Z_5Z_6Z_7)</td>
<td>1 0</td>
<td>0xcc D_0 [0] [7]</td>
</tr>
<tr>
<td>(D_0D_1D_2D_3/D_4T_5Z_6Z_7)</td>
<td>1 0</td>
<td>0xd2 D_0 [0] [7]</td>
</tr>
<tr>
<td>(D_0D_1D_2D_3/D_4D_5T_6Z_7)</td>
<td>1 0</td>
<td>0xe1 D_0 [0] [7]</td>
</tr>
<tr>
<td>(D_0D_1D_2D_3/D_4D_5D_6T_7)</td>
<td>1 0</td>
<td>0xff D_0 [0] [7]</td>
</tr>
</tbody>
</table>

- all undefined bit fields (in yellow) are set to zero for 10GbE
RS “Z” code to 7 bit “C” field mapping

<table>
<thead>
<tr>
<th>RS Z value</th>
<th>name</th>
<th>shorthand</th>
<th>7-bit C field line code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x07,1</td>
<td>idle</td>
<td>[I]</td>
<td>0x00</td>
</tr>
<tr>
<td>0xfb,1</td>
<td>start</td>
<td>[S]</td>
<td>encoded by TYPE byte</td>
</tr>
<tr>
<td>0xfd,1</td>
<td>terminate</td>
<td>[T]</td>
<td>encoded by TYPE byte</td>
</tr>
<tr>
<td>0xfe,1</td>
<td>error</td>
<td>[E]</td>
<td>0x1e</td>
</tr>
<tr>
<td>0x1c,1</td>
<td>reserved0</td>
<td>-</td>
<td>0x2d</td>
</tr>
<tr>
<td>0x3c,1</td>
<td>reserved1</td>
<td>-</td>
<td>0x33</td>
</tr>
<tr>
<td>0x7c,1</td>
<td>reserved2</td>
<td>-</td>
<td>0x4b</td>
</tr>
<tr>
<td>0xbc,1</td>
<td>reserved3</td>
<td>-</td>
<td>0x55</td>
</tr>
<tr>
<td>0xdc,1</td>
<td>reserved4</td>
<td>-</td>
<td>0x66</td>
</tr>
<tr>
<td>0xf7,1</td>
<td>reserved5</td>
<td>-</td>
<td>0x78</td>
</tr>
</tbody>
</table>
Bit ordering sequence

---

IEEE 802.3ae
Task Force

64b/66b Coding Update
Scrambler definition

Serial form of the Scrambler:

The serial form of the scrambler is shown here for bit ordering purposes. Parallel implementations could also be used. For details see:

http://grouper.ieee.org/groups/802/3/ae/public/mar00/walker_1_0300.pdf
Sample 64b/66b Test Vector

- Start with a minimum length (64 byte) Ethernet packet with preamble and CRC

55 55 55 55 55 55 d5 08 00 20 77 05 38 0e 8b 00 00 00 00 08 00 45 00 00 28 1c 66 00 00 1b 06 9e
d7 00 00 59 4d 00 00 68 d1 39 28 4a eb 00 00 30 77 00 00 7a 0c 50 12 1e d2 62 84 00 00 00 00 00
00 00 00 93 eb f7 79

- Add SOP, EOP, Idles and convert to RS indications

07,1 07,1 07,1 07,1 07,1 07,1 07,1 07,1 fb,1 55,0 55,0 55,0 55,0 55,0 55,0 d5,0
08,0 00,0 20,0 77,0 05,0 38,0 0e,0 8b,0 00,0 00,0 00,0 00,0 00,0 08,0 00,0 45,0 00,0
00,0 28,0 1c,0 66,0 00,0 00,0 1b,0 06,0 9e,0 d7,0 00,0 00,0 59,0 4d,0 00,0 00,0
68,0 d1,0 39,0 28,0 4a,0 eb,0 00,0 00,0 00,0 00,0 00,0 00,0 00,0 00,0 00,0 00,0 00,0 93,0 eb,0 f7,0 79,0
fd,1 07,1 07,1 07,1 07,1 07,1 07,1 07,1

- Arrange bytes into frames with type indicators and sync bits

"10" 1e 00 00 00 00 00 00 00 00 00 00 00 00 00 "10" 78 55 55 55 55 55 55 55 d5 "01" 08 00 20 77 05 38 0e 8b
"01" 00 00 00 00 08 00 45 00 "01" 00 28 1c 66 00 00 1b 06 "01" 9e d7 00 00 59 4d 00 00
"01" 68 d1 39 28 4a eb 00 00 "01" 30 77 00 00 7a 0c 50 12 "01" 1e d2 62 84 00 00 00 00
"01" 00 00 00 93 eb f7 79 "10" 87 00 00 00 00 00 00 00 00 00 00 00 00 00

- Scramble and transmit left-to-right, lsb first, (scrambler initial state is set to all ones)

"10" 1e 00 00 00 80 f0 ff 7b "10" 78 15 ad aa aa 16 30 62
"01" 08 e1 81 c5 6e 7c 76 6a "01" e6 30 28 80 cc aa f4 8d
"01" 83 ee 49 ae 6d 93 db 2c "01" f3 46 70 db 82 5a 90 74
"01" 1e 51 79 6b 1a 25 7a c5 "01" 41 1f bf d4 0c 44 ca 4a
"01" 09 28 12 d2 b5 2d 3f 2c "01" 49 92 de c8 b3 33 0e 32
"10" 2a a3 3a c8 d7 ad 99 b5
Frame alignment algorithm

Look for presence of “01” or “10” sync patterns every 66 bits.

This can be done either in parallel, by looking at all possible locations, or in serial by looking at only one potential location.

In either case, a frame sync detector is used to statistically qualify a valid sync alignment.

In the parallel case, a barrel shifter can immediately make the phase shift adjustment. In the serial case, a sync error is used to cycle-slip the demultiplexer to hunt for a valid sync phase.

So what algorithm should be used for reliable and rapid frame sync detection?
Frame sync criteria

If misaligned, then sync error rate will be 50%. We must quickly assert loss of sync and “slip” our alignment to another candidate location.

If already aligned with good BER (<10e-9), then we want to stay in sync with very high reliability.

If BER is worse than 10e-4 we should suppress sync, to avoid likelihood of False Packet Acceptance due to CRC failures.

<table>
<thead>
<tr>
<th>BER</th>
<th>current sync state</th>
<th>next sync state</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>in</td>
<td>out</td>
<td>should be fast</td>
</tr>
<tr>
<td>&gt;10e-4</td>
<td>in</td>
<td>out</td>
<td>prevents MTTFPA events, can be relatively slow to trigger</td>
</tr>
<tr>
<td>&lt;10e-9</td>
<td>out</td>
<td>in</td>
<td>should be fast</td>
</tr>
</tbody>
</table>
Frame sync algorithm

- Frame sync is acquired after 64 contiguous frames have been received with valid “01” or “10” sync headers.
- Frame sync is declared lost after 32 “11” or “00” sync patterns have been declared in any block of 64 frames.
- In addition, if there are 16 or more errors within any 125us time interval (~10^-4 BER), then frame sync is inhibited.

\[
\text{start} \quad \begin{cases} 
\text{OUT} & (\text{BER} < 10^{-4}) \ & \ & \ & \text{64 contiguous error-free frames} \\
\text{IN} & (\text{BER} > 10^{-4}) \ || \ & \ & \ & 32 \text{ or more errors in 64 frames}
\end{cases}
\]
64/66 frame sync performance

- 64 contiguous error free frames rapidly sync for low BERs.
- 16 or more errors in 125us inhibits sync for BER >10e-4.
- 32 or more errors in 64 frames rapidly drops sync for 50% BER.
- One year.
Frame lock process

receiver synchronization condition
sync_done <= frame_lock=true * hi_ber=false

IEEE 802.3ae Task Force

La Jolla, CA July 10-14, 2000 64b/66b Coding Update
BER monitor process

```
power_on=true +
reset = true

M0
mt_valid_cnt<=0
mt_invalid_cnt<=0
hi_ber_cnt<=0
bad_mt_gt_32<=false
good_mt_eq_64<=false
hi_ber <= false

M1
mt_valid=false

M2
mt_invalid_cnt++
hi_ber_cnt++

M3
mt_valid_cnt++

M4
bad_mt gt 32 <= (mt_invalid_cnt>32)?true:false
good_mt eq 64 <= (mt_valid_cnt=64)?true:false
mt_valid_cnt<=0
mt_invalid_cnt<=0

M5
hi_ber<=(hi_ber_cnt>=16)?true:false
hi_ber_cnt<=0

```

IEEE 802.3ae
Task Force

La Jolla, CA  July 10-14, 2000

64b/66b Coding Update
Packet boundary protection

- A 2 bit error in the sync preamble can convert a packet boundary (S,T) into a Data frame (D) and vice-versa. However, all such errors violate frame sequencing rules unless another 4 errors recreate a false S,T packet (a total of six errors). Frame sequence errors invalidate the packet by forcing an (E) on the coder output.
**TX process**

```
power_on=true + reset=true

ELSE initialize_done=true
TYPE(tx_tobe_coded)=Z

ELSE
TYPE(tx_tobe_coded)=Z

ELSE
TYPE(tx_tobe_coded)=S

ELSE
TYPE(tx_tobe_coded)=D

ELSE
TYPE(tx_tobe_coded)=T

ELSE
TYPE(tx_tobe_coded)=S
```
RX process

- \(\text{power\_on=true + reset = true + sync\_done=false}\)
- \(\text{TYPE(rx\_tobe\_decoded)=Z}\)
- \(\text{sync\_done=true && TYPE(rx\_tobe\_decoded)=Z}\)
- \(\text{rx\_to\_gmii=rx\_err}\)
- \(\text{rx\_decoded=EFRAME\_G}\)
- \(\text{rx\_err=EFRAME\_G}\)
- \(\text{ELSE}\)
- \(\text{rx\_to\_gmii=rx\_decoded}\)
- \(\text{rx\_decoded=DECODE(rx\_tobe\_decoded)}\)
- \(\text{rx\_err=rx\_decoded}\)
- \(\text{ELSE}\)
- \(\text{rx\_to\_gmii=rx\_decoded}\)
- \(\text{rx\_decoded=DECODE(rx\_tobe\_decoded)}\)
- \(\text{rx\_err=rx\_decoded}\)
- \(\text{ELSE}\)
- \(\text{rx\_to\_gmii=rx\_decoded}\)
- \(\text{rx\_decoded=DECODE(rx\_tobe\_decoded)}\)
- \(\text{rx\_err=rx\_decoded}\)
- \(\text{ELSE}\)
Optional Code Features

- Special frames are reserved to support ordered sets for both Fiber Channel and 10GbE Link Signalling Sublayer (LSS)
- x,y ordered-set IDs are “1111” for FC and “0000” for 10GbE LSS

<table>
<thead>
<tr>
<th>XGMII Pattern</th>
<th>Sync</th>
<th>Bit fields 0-63</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZZZZ/ODDD</td>
<td>1</td>
<td>0x2d Z0 Z1 Z2 Z3 y D5 D6 D7</td>
</tr>
<tr>
<td>ODDD/ZZZZ</td>
<td>1</td>
<td>0x4b D1 D2 D3 x Z4 Z5 Z6 Z7</td>
</tr>
<tr>
<td>ODDD/ODDD</td>
<td>1</td>
<td>0x55 D1 D2 D3 x y D5 D6 D7</td>
</tr>
<tr>
<td>ODDD/DDDD</td>
<td>1</td>
<td>0x66 D1 D2 D3 x y D5 D6 D7</td>
</tr>
<tr>
<td>SDDD/DDDD</td>
<td>1</td>
<td>0x78 D1 D2 D3 D4 D5 D6 D7</td>
</tr>
<tr>
<td>undefined</td>
<td>1</td>
<td>0x00 reserved for future expansion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rs value</th>
<th>name</th>
<th>shorthand</th>
<th>7-bit line code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x5c,1</td>
<td>FC ordered-set</td>
<td>[Of]</td>
<td>encoded by TYPE byte</td>
</tr>
<tr>
<td>0x9c,1</td>
<td>10 GbE Link Signalling</td>
<td>[LS]</td>
<td>encoded by TYPE byte</td>
</tr>
</tbody>
</table>
Summary

• We’ve shown a simple and reliable algorithm for 64b/66b frame sync detection
• Bit ordering has been clarified to be compatible with Ethernet CRC definition
• The TX and RX error control state machines have been presented
• A simple test vector has been produced to help to verify new implementations
• Optional 64b/66b extensions exist to support FC ordered sets and LS signalling
Supplementary slides
State machine notation conventions

Variables

TXD<35:0> ..................................TXD signal of GMII
RXD<35:0> ..................................RXD signal of GMII

tx_tobe_coded<71:0> ..........................72 bit vector which is to be encoded by the PCS before transmission to the PMA. It is formed by concatenation of two consecutive TXD vectors. With the most recently received TXD word in the 35:0 bit locations.

tx_tobe_xmitted<65:0> .......................A 66 bit vector which is the result of a PCS ENCODE operation and is to be transmitted to the PMA.
rx_tobe_decoded<65:0> ......................A 66 bit vector containing the most recently received code word from the PMA.
rx_decoded<71:0> ..................................72 bit vector which is the result of the PCS DECODE operation on the received bit vector, rx_tobe_decoded

rx_to_gmii<71:0> .............................72 bit vector which is a pipelined delayed copy of rx_decoded. This is sent to GMII in two steps of 36 bits each. Bits 71:36 are sent first to RXD, followed by bits 35:0.

rx_err<71:0> ..................................This holds either a pipeline delayed copy of rx_decoded or the error frame EFRAME_G state .........................Holds the current state of the transmit or the receive process.
sync_done.................................Boolean variable is set true when receiver is synchronized and set to false when receiver looses frame lock.
frame_lock..............................................boolean variable is set true when receiver acquires frame delineation

mt_valid...............................................boolean variable is set true if received frame rx_tobe_decoded has valid frame prefix bits. i.e, mt_valid = rx_tobe_decoded[65] ^ rx_tobe_decoded[64]

mt_valid_cnt ..................................Holds the number of frames within a window of 64 frames, with valid prefix bits
mt_invalid_cnt ..................................Holds the number of frames within a window of 64 frames with invalid prefix bits
good_mt_eq_64.................................Boolean variable is set true when there are 64 contiguous valid prefix bits
bad_mt_gt_32.................................Boolean variable is set true when there are at least 32 invalid prefix bits within a block of 64
hi_ber_cnt..............................................Holds the number of with invalid prefix bits, within a 125us period

hi_ber..............................................Boolean is asserted true when the hi_ber_cnt exceeds 16 indicating a bit error rate >=10^-4

slip_done...........................................Boolean variable is set true when the hi_ber_cnt exceeds 16 indicating a bit error rate >=10^-4
State machine notation conventions

Constants

```c
const enum FRAME_TYPE = { Z, S, T, D} ......................... Each 72 bit vector, tx_tobe_coded and the 66 bit vector, rx_tobe_decoded, can be classified to belong to one of the four types depending on its contents. The frame types Z, S, T, D are defined in TBD.
```

```c
EFRAME_G<71:0> .......................... 72 bit vector to be sent to the GMII interface and represents a error octet in all the eight octet locations
```

```c
EFRAME_P<65:0> .......................... 66 bit vector to be sent to the PMA and represents a error octet in all the eight octet locations,
```

Functions

```c
ENCODE( tx_tobe_coded<71:0>) .......................... Encodes the 72 bit vector into a 66 bit vector to be transmitted to the PMA
```

```c
DECODE( rx_tobe_decoded<65:0> ) .......................... Decodes the 66 bit vector into a 72 bit vector to be sent to the GMII
```

```c
TYPE( tx_tobe_coded<71:0> )TYPE( rx_tobe_decoded<65:0>) .......................... Decodes the FRAME_TYPE of the tx_tobe_coded<71:0> bit vector or the rx_tobe_decoded<65:0>
```

Timers

```c
64frames_timer_done .......................... Timer which is triggered once every 64 of the 66-bit frames in the receive process
```

```c
125us_timer_done .......................... Timer which is triggered once every 125us (is approximately 2\(^{14}\) 66-bit frames in the receive process).
```