25GbE PCS Technical Feasibility

IEEE 25Gb/s Ethernet Study Group

TBD

Mark Gustlin – Xilinx
Gary Nicholl - Cisco
Dave Ofelt - Juniper

Introduction

- These slides explore the technical feasibility of a 25GbE PCS
- The 25 Gb/s rate is lower than currently shipping rates, so inherently this rate is feasible, but these slides explore the options for very high leverage from previous rates (10/40/100Gb/s)
- With high leverage from previous speeds, compact multi-rate implementations are possible

Assumptions

- **Possible PMDs of interest are:**
  - CR
  - KR
  - SR
- **Channel assumptions are similar if not identical to 100GBASE-CR4, KR4 and SR4**
  - Loss budgets are the same as a single xR4 channel
  - Assume crosstalk is similar (multiple 25GbEs run next to each other)
- **Assuming no KP channel needed?**
  - But architecture should support it if needed
- **Therefore a moderate strength FEC is required, assuming at this point that RS(528,514) is sufficient**
  - If the assumptions change then this might change also
- **Goal is to maximize re-use from previous projects**
  - Many devices will need to support 100GbE/40GbE/25GbE/10GbE on a given interface/port

Option 1

1. 64b/66b only, leveraging 40/100GBASE-R but run at 25.78125G
   - But without Alignment Markers
   - 64b alignment for encoding (leveraging clause 82)
2. Use the 256B/257B transcoding as defined in 802.3bj
3. RS-FEC encoded data always
   - Just sync up FEC correctable match, with 256b/257b transcoding
   - Bit slips until n FEC correctable blocks are found, loses lock after m FEC blocks are uncorrectable
   - Similar to clause 74 KR FEC

Option 2

1. 64b/66b only, leveraging 10GBASE-R but run at 25.78125G
   - No Alignment Markers
   - 32b alignment for encoding
2. Use the 256B/257B transcoding as defined in 802.3bj
3. RS-FEC encoded data always
   - Just sync up FEC correctable match, with 256b/257b transcoding
   - Bit slips until n FEC correctable blocks are found, loses lock after m FEC blocks are uncorrectable
   - Similar to clause 74 KR FEC

Option 3

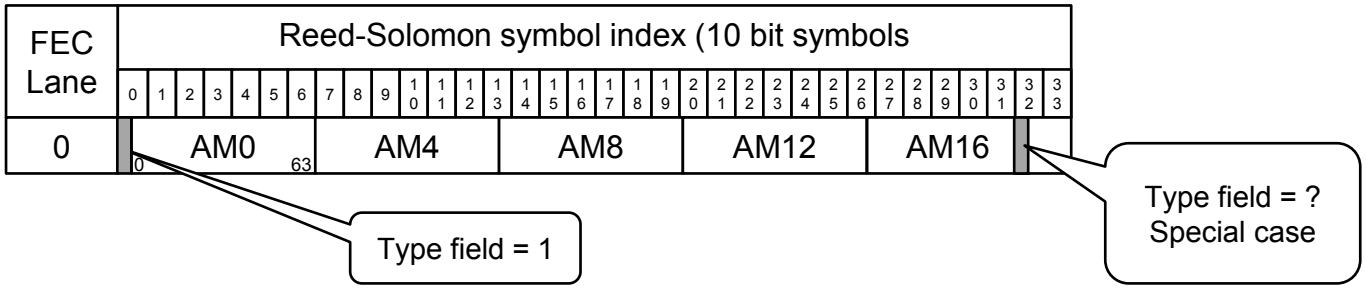
1. 64b/66b only, leveraging 40GBASE-R but run at 25.78125G
   - Single Alignment Marker (or single group of five)? Single PCS lane?
   - 64b alignment for encoding
2. Use the 256B/257B transcoding as defined in 802.3bj
   - No remapping of AMs needed though
3. RS-FEC encoded data always
   - With Alignment markers you can sync up the same as you do for 100G

Option 3 AMs

- If we add alignment markers, do we add one, or more??
- Reminder of what 100G looks like, AM0 and AM16 are used for block lock, especially for pre FEC lock with EEE for a rapid lane agnostic lock

FEC Lane	Reed-Solomon symbol index (10 bit symbols 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33					
0	AM0 (0 … 63)	AM4	AM8	AM12	AM16	5b pad
1	AM0	AM5	AM9	AM13	AM16	
2	AM0	AM6	AM10	AM14	AM16	
3	AM0	AM7	AM11	AM15	AM16	

- If you do a single 64b AM, then you would transcode it as if it is data, though then you can't share the lane lock with 100GbE
- Putting in 5x64b would allow more re-use with 100G? But since you don't have a block of 20, the pad stuff is strange, might just map the 5 blocks as if they are data



Questions

- Open questions
  - Should we have Alignment Markers even though we don't need them to be more compatible with other speeds that will co-exist most often?
    - If we don't have AMs, then we lose the BIP fields, don't think this is a big deal if we always require FEC?
  - Should we follow clause 81/82 rules for IPG and block types instead since co-existence with 100GbE is more important than 10GbE?

Block Type Deltas

- **10GbE aligns data on 32b boundaries, 40/100GbE aligns always on 64b boundaries**
  - This leads to a few differences between the block encodings and also the IPG rules (4 less block types required for 40/100GbE)

Input Data	Sync	Block Payload							
Bit Position: / Data Block Format:	0 1	2							65
$D_0\,D_1\,D_2\,D_3\,D_4\,D_5\,D_6\,D_7$	01	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
Control Block Formats:		Block Type Field							
$C_0\,C_1\,C_2\,C_3/C_4\,C_5\,C_6\,C_7$	10	0x1e	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
$C_0\,C_1\,C_2\,C_3/O_4\,D_5\,D_6\,D_7$	10	0x2d	C_0	C_1	C_2	C_3	O_4	D_5	D_6	D_7
$C_0\,C_1\,C_2\,C_3/S_4\,D_5\,D_6\,D_7$	10	0x33	C_0	C_1	C_2	C_3		D_5	D_6	D_7
$O_0\,D_1\,D_2\,D_3/S_4\,D_5\,D_6\,D_7$	10	0x66	D_1	D_2	D_3	O_0		D_5	D_6	D_7
$O_0\,D_1\,D_2\,D_3/O_4\,D_5\,D_6\,D_7$	10	0x55	D_1	D_2	D_3	O_0	O_4	D_5	D_6	D_7
$S_0\,D_1\,D_2\,D_3/D_4\,D_5\,D_6\,D_7$	10	0x78	D_1	D_2	D_3	D_4		D_5	D_6	D_7
$O_0\,D_1\,D_2\,D_3/C_4\,C_5\,C_6\,C_7$	10	0x4b	D_1	D_2	D_3	O_0	C_4	C_5	C_6	C_7
$T_0\,C_1\,C_2\,C_3/C_4\,C_5\,C_6\,C_7$	10	0x87		C_1	C_2	C_3	C_4	C_5	C_6	C_7
$D_0\,T_1\,C_2\,C_3/C_4\,C_5\,C_6\,C_7$	10	0x99	D_0		C_2	C_3	C_4	C_5	C_6	C_7
$D_0\,D_1\,T_2\,C_3/C_4\,C_5\,C_6\,C_7$	10	0xaa	D_0	D_1		C_3	C_4	C_5	C_6	C_7
$D_0\,D_1\,D_2\,T_3/C_4\,C_5\,C_6\,C_7$	10	0xb4	D_0	D_1	D_2		C_4	C_5	C_6	C_7
$D_0\,D_1\,D_2\,D_3/T_4\,C_5\,C_6\,C_7$	10	0xcc	D_0	D_1	D_2	D_3		C_5	C_6	C_7
$D_0\,D_1\,D_2\,D_3/D_4\,T_5\,C_6\,C_7$	10	0xd2	D_0	D_1	D_2	D_3	D_4		C_6	C_7
$D_0\,D_1\,D_2\,D_3/D_4\,D_5\,T_6\,C_7$	10	0xe1	D_0	D_1	D_2	D_3	D_4	D_5		C_7
$D_0\,D_1\,D_2\,D_3/D_4\,D_5\,D_6\,T_7$	10	0xff	D_0	D_1	D_2	D_3	D_4	D_5	D_6	

Figure 49–7—64B/66B block formats

Input Data	Sync	Block Payload							
Bit Position: / Data Block Format:	0 1	2							65
$D_0\,D_1\,D_2\,D_3\,D_4\,D_5\,D_6\,D_7$	01	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
Control Block Formats:		Block Type Field							
$C_0\,C_1\,C_2\,C_3\,C_4\,C_5\,C_6\,C_7$	10	0x1E	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
$S_0\,D_1\,D_2\,D_3\,D_4\,D_5\,D_6\,D_7$	10	0x78	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$O_0\,D_1\,D_2\,D_3\,Z_4\,Z_5\,Z_6\,Z_7$	10	0x4B	D_1	D_2	D_3	O_0	0x000_0000			
$T_0\,C_1\,C_2\,C_3\,C_4\,C_5\,C_6\,C_7$	10	0x87		C_1	C_2	C_3	C_4	C_5	C_6	C_7
$D_0\,T_1\,C_2\,C_3\,C_4\,C_5\,C_6\,C_7$	10	0x99	D_0		C_2	C_3	C_4	C_5	C_6	C_7
$D_0\,D_1\,T_2\,C_3\,C_4\,C_5\,C_6\,C_7$	10	0xAA	D_0	D_1		C_3	C_4	C_5	C_6	C_7
$D_0\,D_1\,D_2\,T_3\,C_4\,C_5\,C_6\,C_7$	10	0xB4	D_0	D_1	D_2		C_4	C_5	C_6	C_7
$D_0\,D_1\,D_2\,D_3\,T_4\,C_5\,C_6\,C_7$	10	0xCC	D_0	D_1	D_2	D_3		C_5	C_6	C_7
$D_0\,D_1\,D_2\,D_3\,D_4\,T_5\,C_6\,C_7$	10	0xD2	D_0	D_1	D_2	D_3	D_4		C_6	C_7
$D_0\,D_1\,D_2\,D_3\,D_4\,D_5\,T_6\,C_7$	10	0xE1	D_0	D_1	D_2	D_3	D_4	D_5		C_7
$D_0\,D_1\,D_2\,D_3\,D_4\,D_5\,D_6\,T_7$	10	0xFF	D_0	D_1	D_2	D_3	D_4	D_5	D_6	

Figure 82–5—64B/66B block formats

IPG Rules

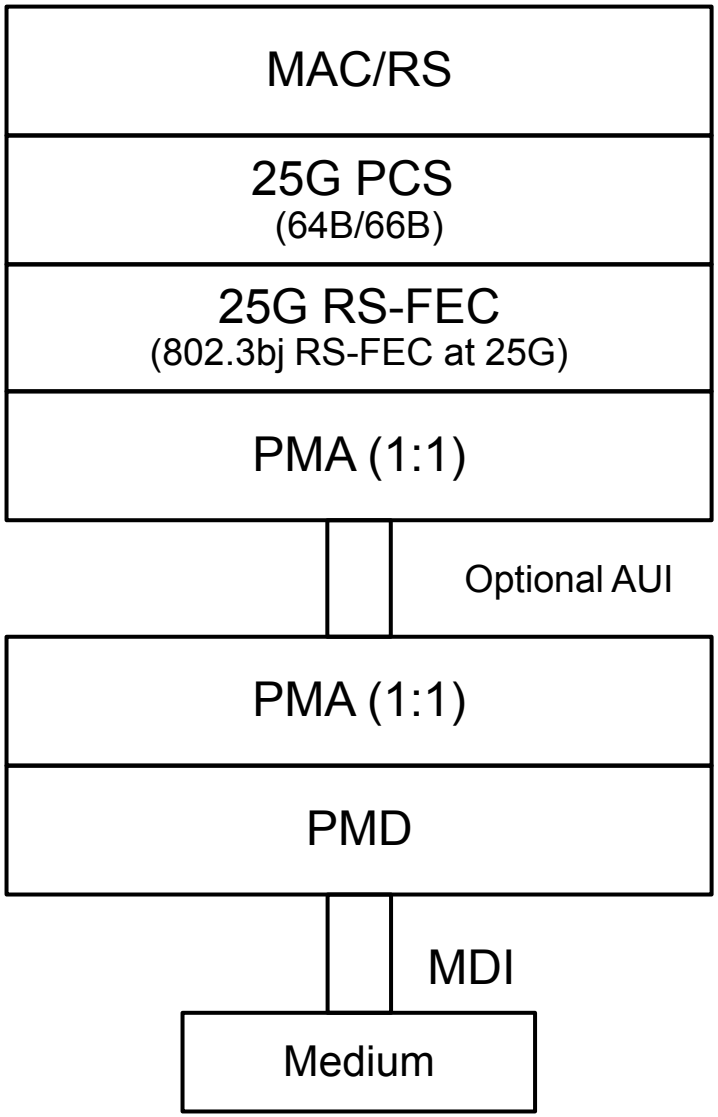
- At 10GbE the Deficit Idle Counter is bound between zero and three, minimum IPG is 5 Bytes
- At 40/100GbE Deficit Idle Counter is bound between zero and seven, minimum IPG is 1Byte

NOTE 7—For 40 Gb/s and 100 Gb/s operation, the received interpacket gap (the spacing between two packets, from the last bit of the FCS field of the first packet to the first bit of the Preamble of the second packet) can have a minimum value of 8 BT (bit times), as measured at the XLGMII or CGMII receive signals at the DTE due to clock tolerance and lane alignment requirements.

NOTE 4—For 10 Gb/s operation, the spacing between two packets, from the last bit of the FCS field of the first packet to the first bit of the Preamble of the second packet, can have a minimum value of 40 BT (bit times), as measured at the XGMII receive signals at the DTE. This interpacket gap shrinkage may be caused by variable network delays and clock tolerances.

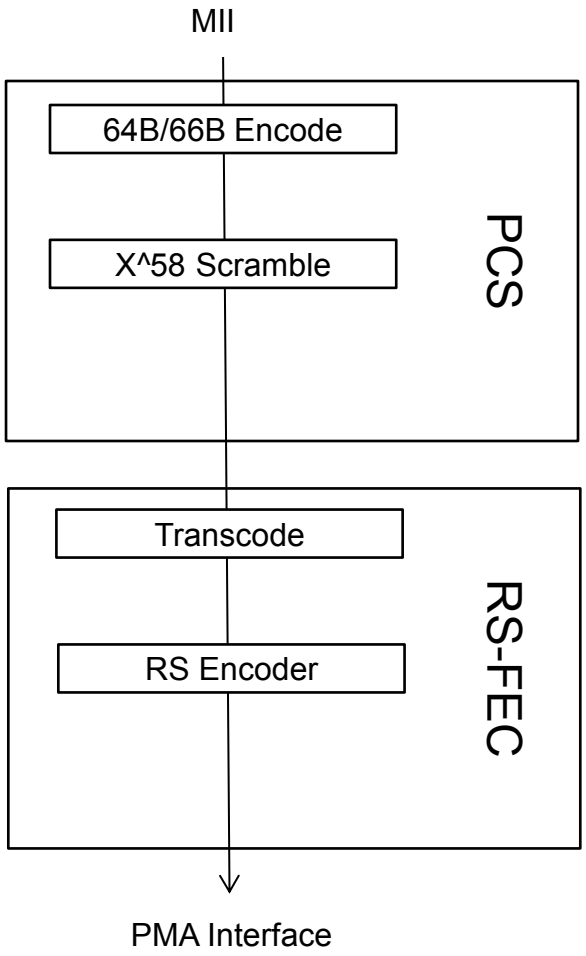
25GbE Architecture With RS-FEC

- PCS is 64B/66B based, no AMs
- Required RS-FEC sublayer
- 1 lane below the RS-FEC sublayer



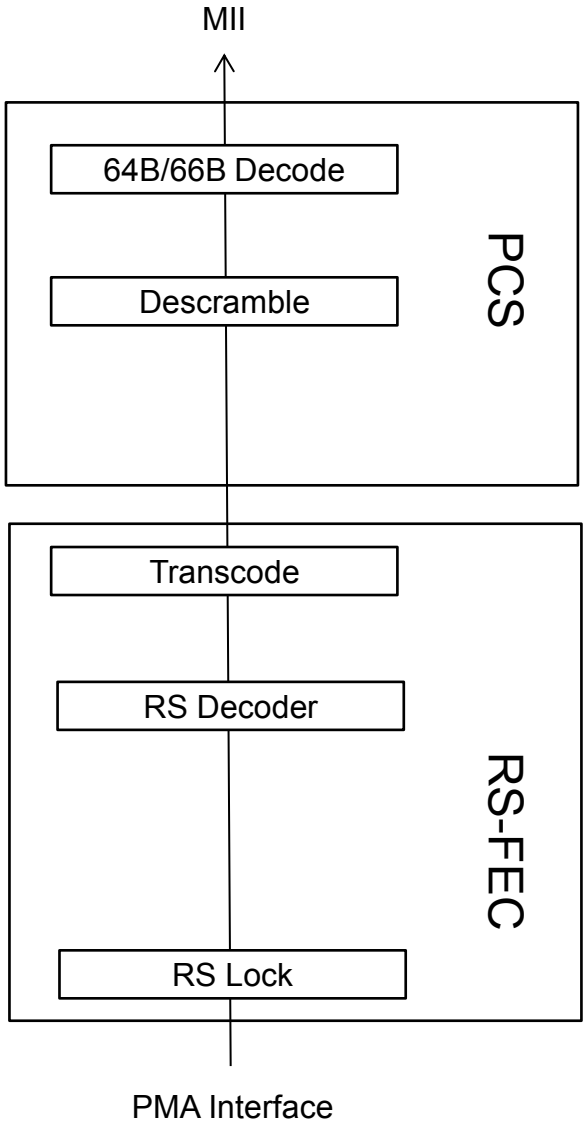
Data Flow – TX Option 1

- RS-FEC sublayer re-uses the transcoding function and the RS encoder from 802.3bj

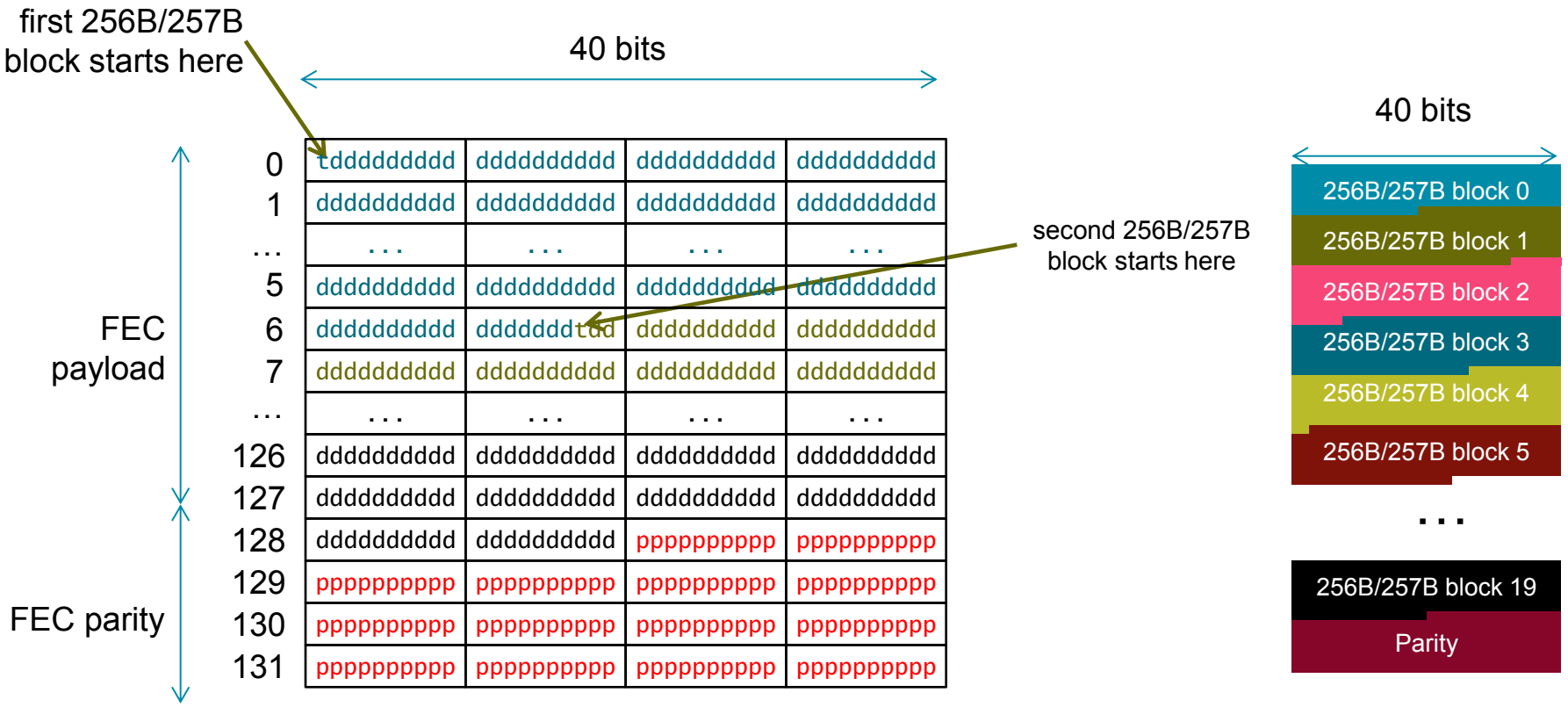


Data Flow – RX Option 1

- RS-FEC sublayer re-uses the transcoding function and the RS decoder from 802.3bj



FEC frame structure



Thoughts on Latency

- **Goal of 802.3bj was achievable 100ns latency adder, that can be achieved depending on your implementation complexity**
  - Major contributors:
    - Block arrival wait time: ~50ns for 100G
    - Decoder processing time: ~50ns
    - Total ~100ns
- **With 25GbE running ¼ the rate:**
  - Major contributors:
    - Block arrival wait time: ~200ns for 25G
    - Decoder processing time: ~50ns
    - Total ~250ns

Scrambling

- Likely a desire to re-use the $X^{58} + X^{39} + 1$ scrambler from previous speeds, and in a self synchronous mode
- Keep the xor function for the 256B/257B transcoded data, which randomizes the header overhead

Thanks!