Backplane NRZ FEC Baseline Proposal

IEEE P802.3bj

March 2012 Hawaii

Stephen Bates – PMC-Sierra, Matt Brown – APM,
Roy Cideciyan – IBM, Mark Gustlin – Xilinx,
Adam Healey - LSI, Martin Langhammer - Altera,
Jeff Slavick – Avago, Zhongfeng Wang – Broadcom

Supporters and Contributors

Sudeep Bhoja - Broadcom

Matt Brown - APM

Frank Chang - Vitesse

Chris Cole – Finisar

Mike Dudek - QLogic

John F Ewen - IBM

Dimitrios Giannakopoulos – APM

Ziad Hatab – Vitesse

Elizabeth Kochuparambil - Cisco

Ryan Latchman - Mindspeed

Arthur Marris - Cadence

Mounir Meghelli – IBM

David Ofelt – Juniper Networks

Oren Sela – Mellanox

Farhad Shafai - Xilinx

Andre Szczepanek – Inphi

Introduction

- Over the past few meeting cycles many different FEC options have been presented, this paper selects a candidate FEC code for NRZ backplanes

- Considerations are:
  - Effective gain (includes raw FEC coding gain and burst error behavior)
  - Logic complexity and power
  - Achievable latency
  - Over-clocking requirements

- A proposal is shown for an NRZ backplane FEC

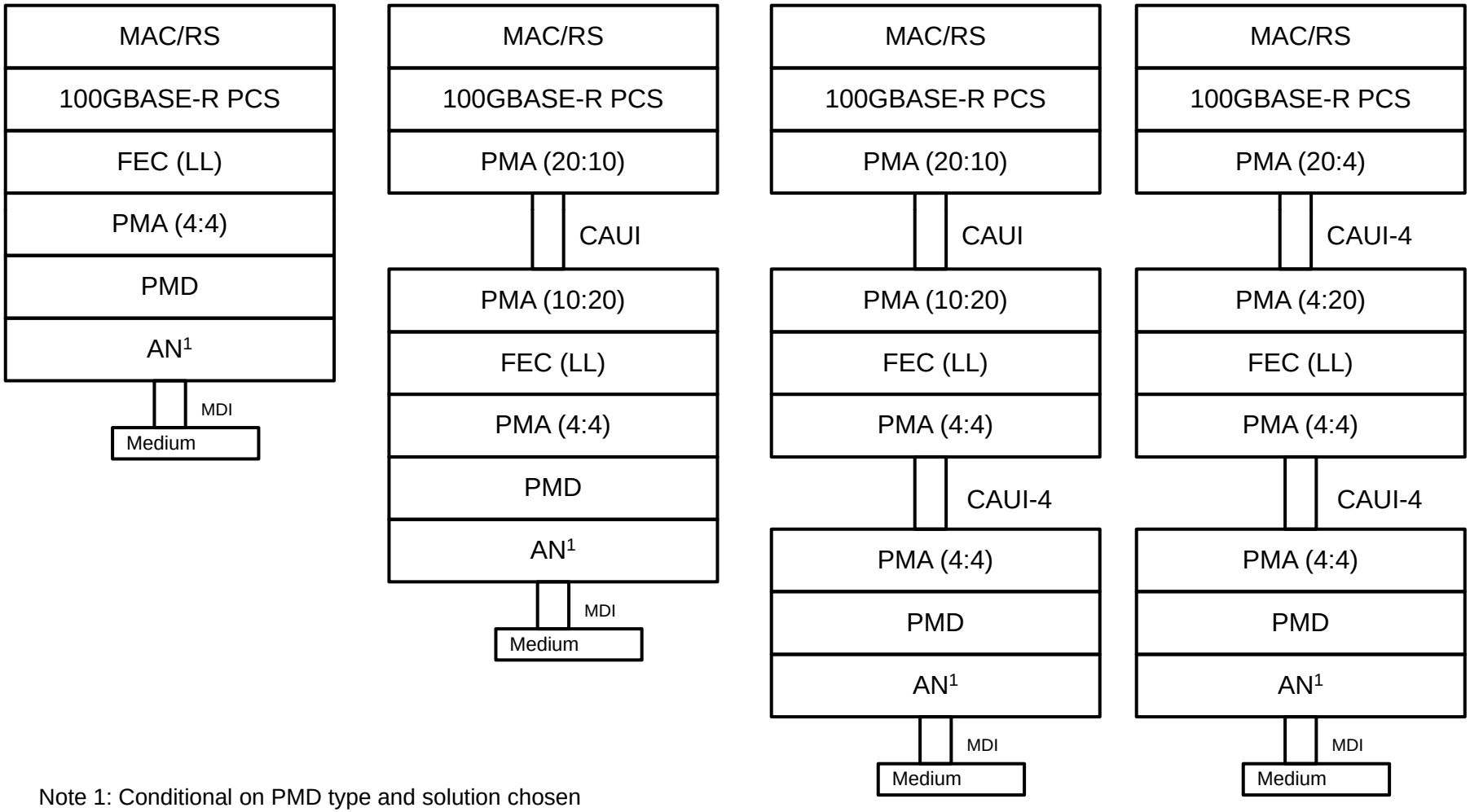
- The FEC processing flow is updated

Proposed FEC Operation

- Backplane NRZ:
  - Same lane rate (25.78G) with or without FEC
  - FEC is optional (to implement and to use) for a channel loss of 30dB or less, and FEC is required for a channel > 30dB up to 35dB
  - Without FEC enabled there is no transcoding (64b/66b encoding)
  - With FEC enabled we use 256B/257B transcoding
  - FEC use is auto-negotiated
  - Proposed FEC code is option 1 from the 0% overhead FEC options (~4.9dB of gain at 1e-15), see slide 6
- Backplane PAM4:
  - Covered separately in brown_01_0312
  - The goal is to share the same transcoding and similar striping, though the FEC code must have more gain and therefore it is different for PAM4
- Copper cable:
  - Use same FEC as NRZ backplane for achieving 5m objective
    - Need to decide how to auto-negotiate it based on cable loss
    - Or always encode but don’t decode unless necessary (but what about MTTFPA)

Low Latency FEC Architecture

- The figures below show possible striped (and therefore low latency) FEC architectures



Note 1: Conditional on PMD type and solution chosen

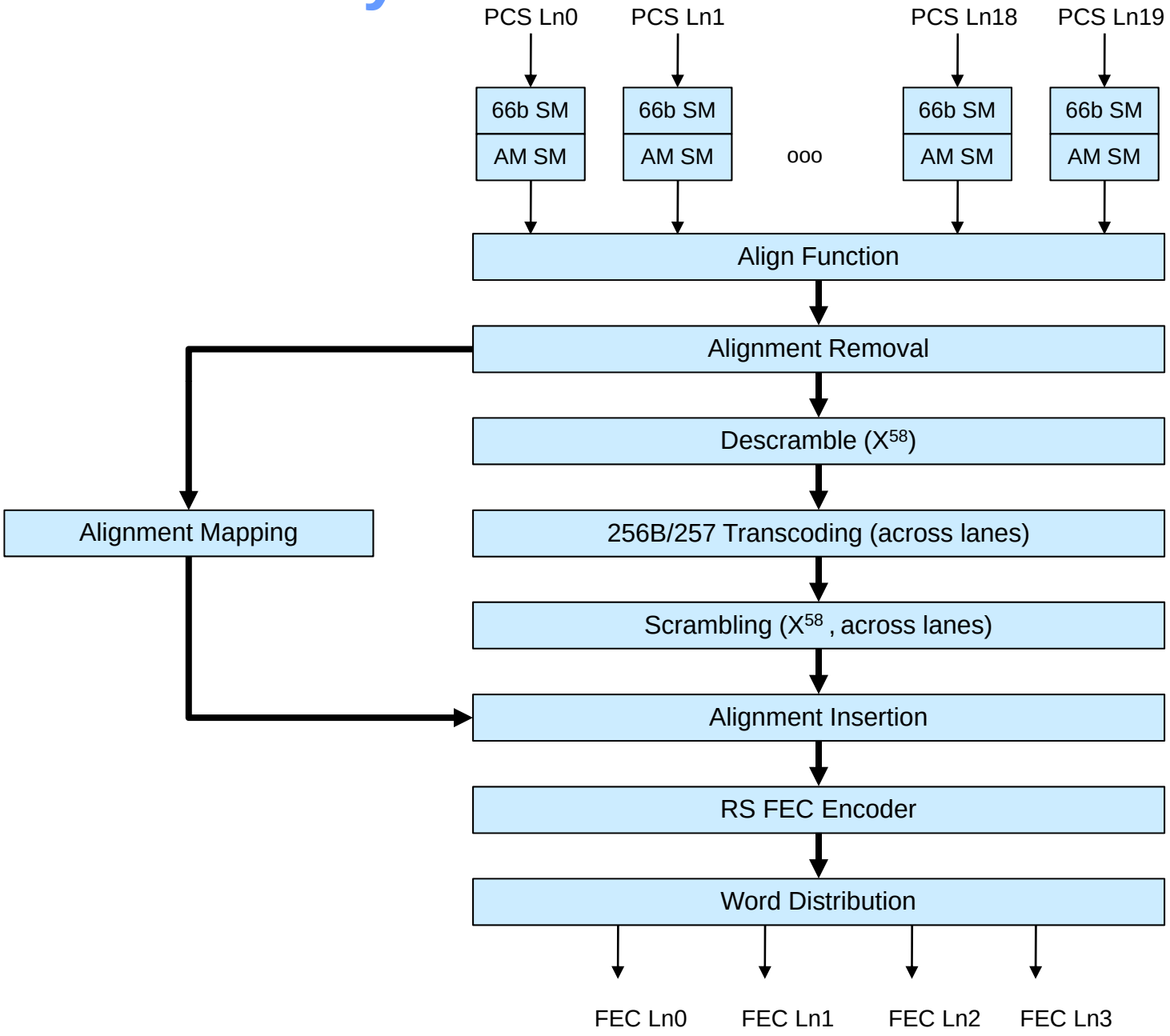
Note: LL = Low Latency CAUI-4 – assumed new 25G+ interface

0% Over-clocking Options

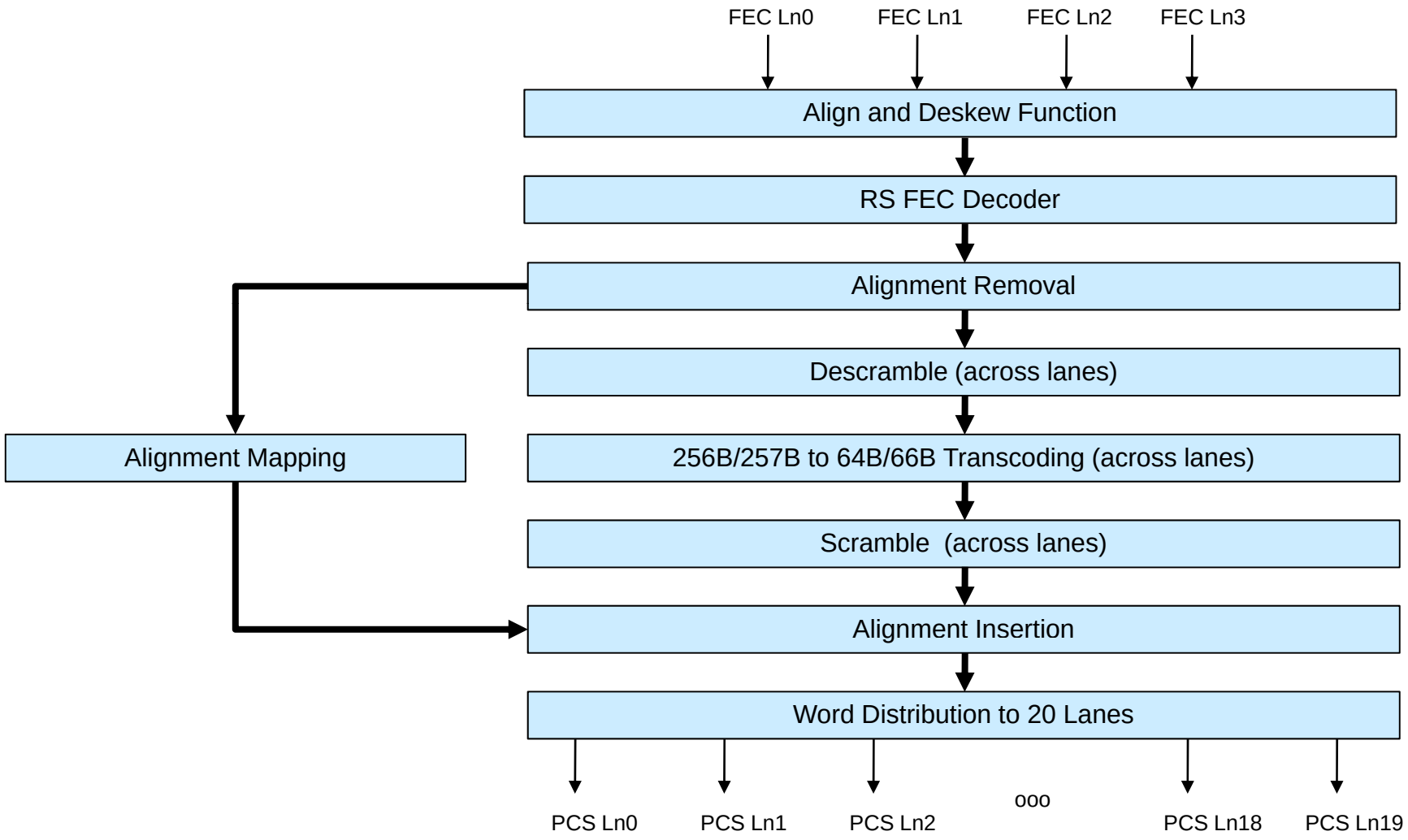
- Option 1 is our preferred choice, it has the good gain, latency < 100ns, has an acceptable complexity and the 256Bb/257B transcoding prevents the mixing of AM and Control/Data within a transcoded block
- All options assume some trans-coding across lanes
- None of these options can handle cross lane correlated error bursts well
- All have an integer reference clock multiplier (RCM = 165)

Option	FEC Code RS(n, k, t, m)	Trans-coding	Effective Gain BER= 10^{-15}	Overall Latency	Total Area (40nm gates)	Total Power	Input BER for 10^{-15} BER	Input BER for 10^{-12} BER
1	RS(528, 514, 7, 10)	256b/257b	4.87 dB	94.3 ns	244k	90 mW	$4.68x10^{-6}$	$2.34x10^{-5}$
2	RS(528, 513, 7, 10)	512b/513b	4.87 dB	99.4 ns	285k	105 mW	$4.68x10^{-6}$	$2.34x10^{-5}$
3	RS(528, 516, 6, 10)	512b/516b	4.52 dB	96.8 ns	243k	88 mW	$1.86x10^{-6}$	$1.12x10^{-5}$
4a	RS(468, 456, 6, 9)	512b/513b	4.51 dB	96.3 ns	197k	72 mW	$1.82x10^{-6}$	$1.23x10^{-5}$
4b	RS(234, 228, 3, 9)	512b/513b	2.06 dB	52.9 ns	108k	40 mW	$2.39x10^{-10}$	$9.77x10^{-8}$
5a	RS(528,516,6,10)	256b/258b	4.52 dB	90 ns	212k	77mW	$1.86x10^{-6}$	$1.12x10^{-5}$
5b	RS(264,258,3,10)	256b/258b	2.35dB	49 ns	113k	41mW	$9.12x10^{-10}$	$1.12x10^{-7}$

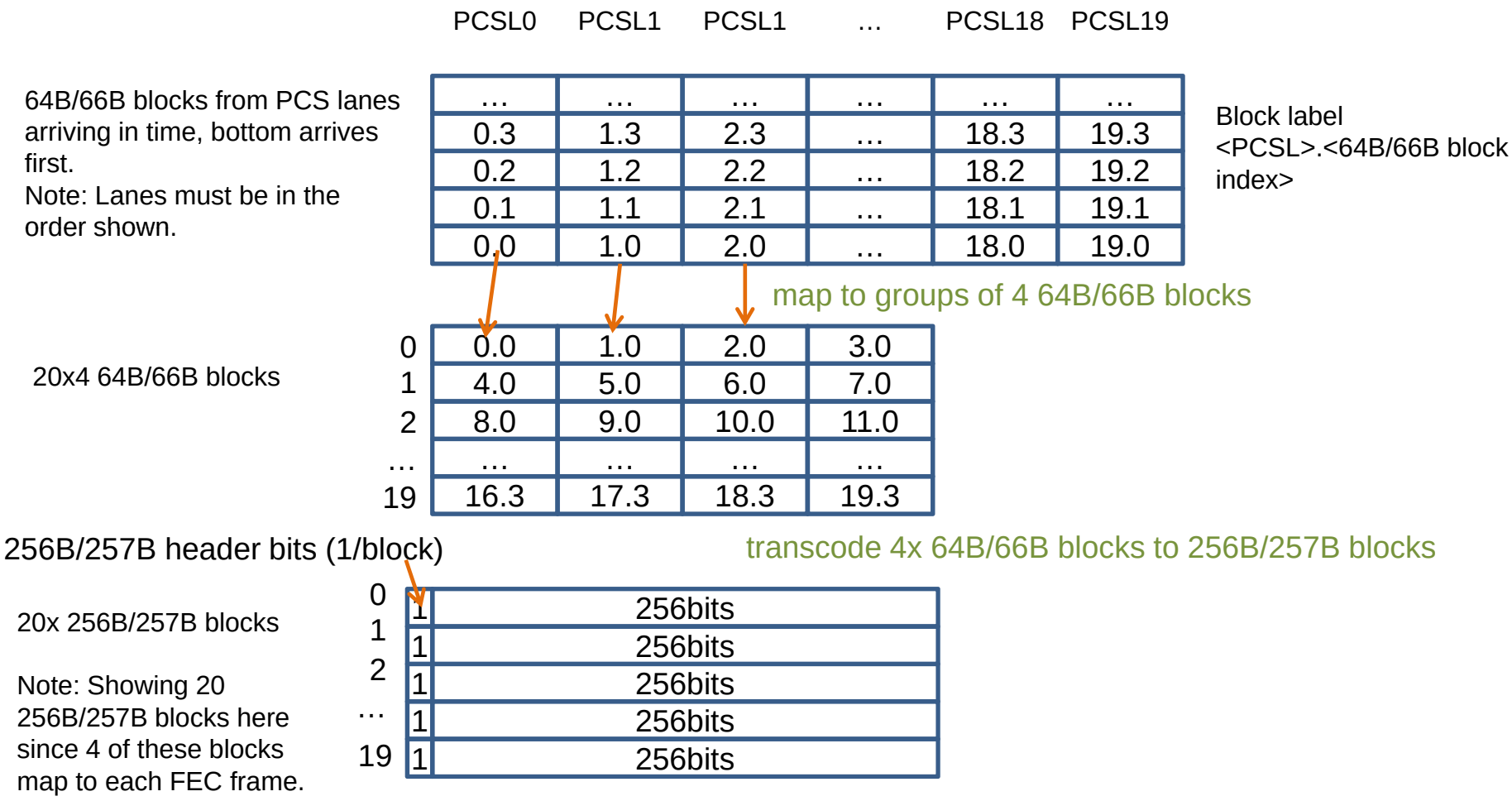
Low Latency TX FEC Architecture



Low Latency RX FEC Architecture

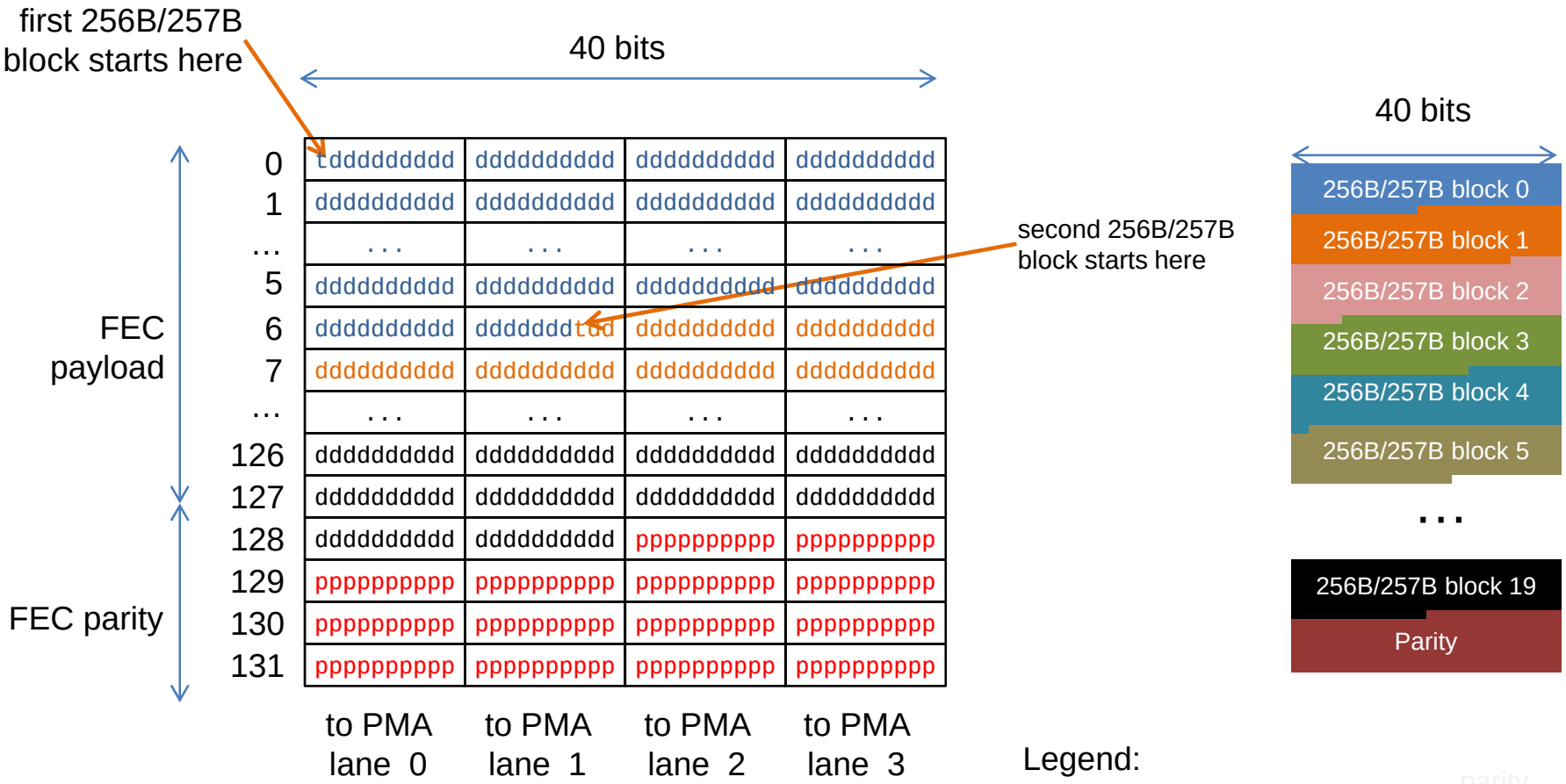


Mapping 64B/66B blocks to 256B/257B



Format of the next few slides are borrowed from brown_01_0112

FEC frame structure



Legend:
“t” = 256B/257B header bit
“d” = 256B/257B data bit
“p” = FEC parity bit

FEC frame structure, more detail

Row index	RS Symbol index															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	B0															
1	B0									7b	3b	B1				
2	B1															
3	B1			4b	6b	B2										
4	B2													1b	9b	B3
5	B3															
6	B3							8b	2b	B4						
7	B4															
8	5b	5b	B5													
9	B5										2b	8b	B6			
10	B6															
11	B6			9b	1b	B7										
12	B7													6b	4b	B8
13	B8															
14	B8								3b	7b	B9					
15	B9															
16	B9	B10														
...	...															
31	B19															
32	B19	Checksum														

256B/257B Transcoding

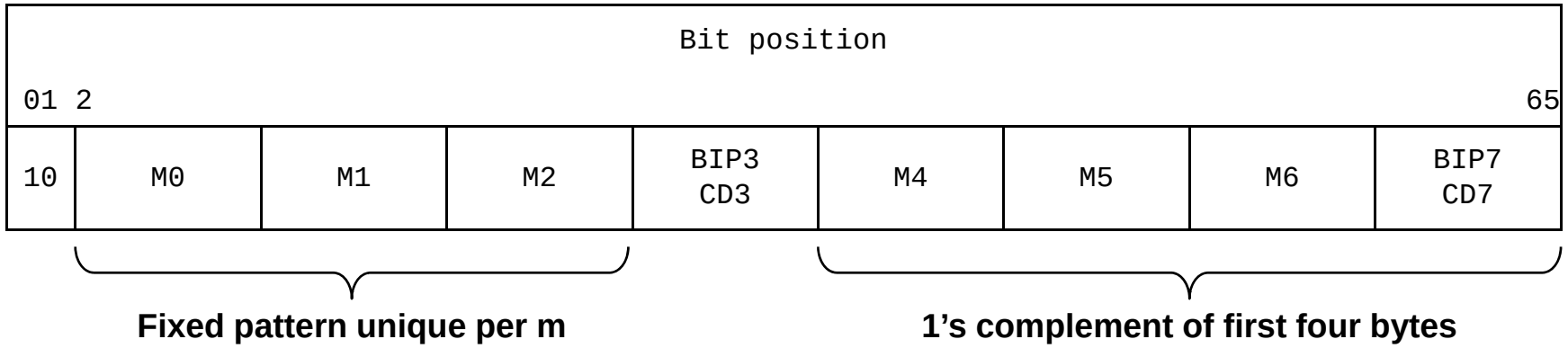
- The details of the proposed transcoding are in cideciyan_01_0312

How to Handle Alignment Markers?

- AMs are special patterns that we will use to find alignment between the 4 physical lanes on the receiver before decoding the FEC block
  - These are used to find alignment on 20 PCS lanes for 802.3ba
- But Alignment Markers are constructed to be sent out sequentially. With our 10 bit striping across physical lanes (due to the 10 bit RS symbols), we want to remap the AMs so that the AM properties are preserved on a per physical lane basis
  - We want to preserve their DC balance and random like properties, note that they are not scrambled
- Therefore we remove the AMs and then remap them back into the data stream in a new form by taking into account the 10 bit striping
- AMs also are not transcoded, instead the two header bits are stripped, then 5 dummy bits (set to a fixed pattern) are added to the end of the 20 AMs to pad out the size to 1285 bits (64*20 = 1280 and 257b * 5 = 1285)
  - Dummy bits are set to b00101 and b11010 in an alternating pattern
- If there are any errors in the received Alignment Markers (in the M0/1/2 patterns) when we do the mapping, it is not required that the errors be corrected
- Will we allow lane flexibility, where any logical lane can be transmitted on any physical lane. This Requires 4x4:1 muxes on the RX side, but is more consistent with 802.3ba.
  - Note that auto negotiation has to be on a designated lane

Alignment Marker Format

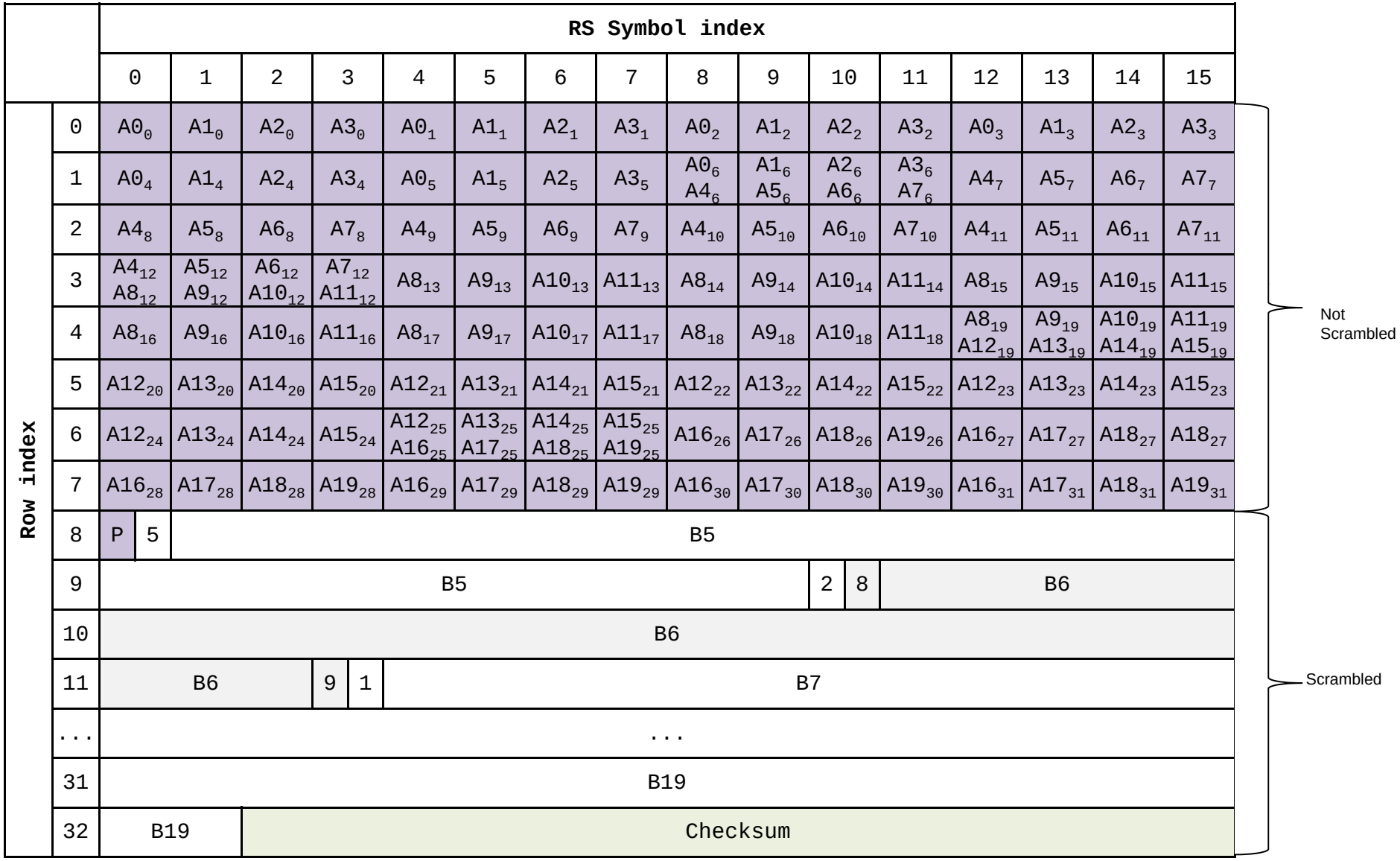
66-bit alignment marker m, 64-bit payload denoted as AM



Strip sync. header and map alignment marker payloads to appear on FEC lanes as shown

	RS symbol index																													
	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
FECL<0>	A0								A4							A8								A12						A16		
FECL<1>	A1								A5							A9								A13						A17		
FECL<2>	A2								A6							A10								A14						A18		
FECL<3>	A3								A7							A11								A15						A19		

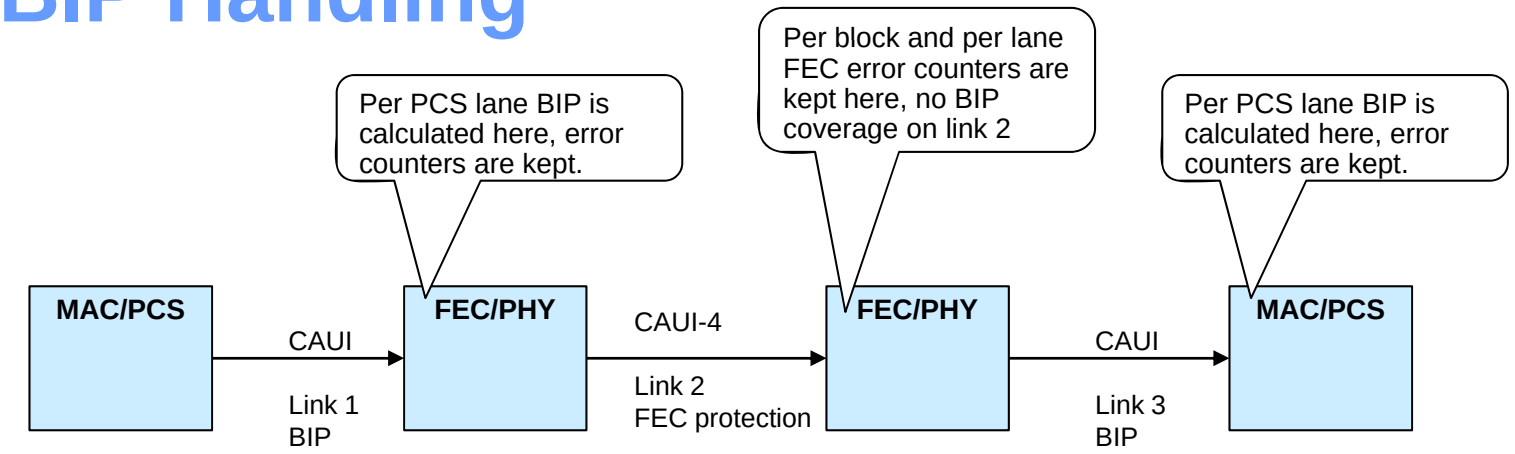
FEC frame structure with AMs Split up



Scrambling

- All data (except AMs) sent across the parallel FEC encoded links must be re-scrambled after transcoding since we de-scrambled the data before transcoding
- Proposal is to use the X^58 self synchronous scrambler that is normally used, it runs on the entire 257bit block, except for the Alignment Markers blocks.

BIP Handling



- BIP values are calculated after scrambling and then inserted into the Alignment Markers as part of the 802.3ba PCS
- When performing transcoding then, we must check the BIP values before we descramble
- What do we do on FEC Transmit though? Do we recalculate them? Or leave the old values just for random fill?
  - Proposal is to leave the BIP values as is for filler when being carried across the FEC link, this also works nicely for the EEE case where the countdown fields also must be carried transparently
  - BIP is regenerated on the far end when the 100GBASE-R PCS is recreated
  - This allows all errors to be isolated, except for the rare case of uncorrectable FEC errors

Error Marking

- If the FEC decoder can't correct errors due to there being too many, how should the FEC processing mark the blocks so that the downstream logic knows that there are uncorrectable errors and drops all associated packets?
- In Clause 74 the sync headers are marked with invalid values (11) to indicate to the PCS that the blocks are in error. Should we do the same?
    - In clause 74 all 32 sync headers are marked as invalid for a FEC block that cannot be corrected (due to multiple FEC blocks being interleaved) for 100GE
    - The 802.3ba state machines do not have an issue with this, it takes 65 invalid sync headers within a 1k window to go out of block lock
- In our case, we have 80x66b blocks within a single FEC block, so we would need to invalidate the 1st, 9th, 17th, 25th, 33rd, 41st, 49th, 57th, 65th, 73rd, 80th blocks (11 block headers total to ensure all 64B packets that might be sent are dropped). We can follow the clause 74 method, setting to 11 the sync header bits when we transcode back to 64B/66B.
    - Is this good enough? With our strong FEC, if there are errors that can't be corrected, that means major issues, do we have to protect against things like spoofed faults etc?
- Another possibility is to mark the 66B blocks with error codes (valid control blocks with all control codes set to 0x1E). This is relatively easy to do since the data is descrambled due to the transcoding. The advantage is that you are not mucking with the sync headers and causing sync header errors to increase, possibly unnecessarily. Also you would set to /E/s all blocks that were part of the FEC block, this seems a little bit more robust than just setting the sync header errors.
- Plan is to mark all 66b blocks that are contained within an uncorrectable FEC block with /E/ (a valid control block with all control codes set to 0x1E). This is true even for blocks that we are sure are AMs (by their repetitive position).

Alignment Marker Lock

- The details of the AM lock on a per FEC lane basis is still under investigation
- The FEC AM lock SM will operate on the pre FEC data running at a poor BER and the design will take that into account
- We need to ensure that the FEC lock SM plays nicely with the downstream AM/block lock SMs
- Need to decide what to send to the higher layer if we loose FEC lock, assume that this will just be local fault.

NRZ RS Codeword (528,514,7,10)

- Field Polynomial: $g(x) = x^{10} + x^3 + 1$
- Generator Polynomial: $G(x) = (x - \alpha^0)(x - \alpha^1)(x - \alpha^2)\ldots(x - \alpha^{13})$
- Generator Polynomial Example

  Coefficients in reverse order (decimal and hex)

  432 290 945 265 592 391 614 900 925 656 32 701 6 904 1

  0x1b0 0x122 0x3b1 0x109 0x250 0x187 0x266 0x384 0x39d 0x290 0x20 0x2bd 0x6 0x388 0x1
- Codeword Example

  k = 514 symbol values from 1023 decremented to 510

  Followed by (n-k) = 14 check symbols

  451 952 674 140 539 287 460 438 559 883 542 885 930 191 decimal

  0x1c3 0x3b8 0x2a2 0x8c 0x21b 0x11f 0x1cc 0x1b6 0x22f 0x373 0x21e 0x375 0x3a2 0xbf

NRZ RS Codeword (528,514,7,10)

- Codeword Example

1023 1022 1021 1020 1019 1018 1017 1016 1015 1014 1013 1012 1011 1010 1009 1008 1007 1006
1005 1004 1003 1002 1001 1000 999 998 997 996 995 994 993 992 991 990 989 988 987 986 985
984 983 982 981 980 979 978 977 976 975 974 973 972 971 970 969 968 967 966 965 964 963 962
961 960 959 958 957 956 955 954 953 952 951 950 949 948 947 946 945 944 943 942 941 940 939
938 937 936 935 934 933 932 931 930 929 928 927 926 925 924 923 922 921 920 919 918 917 916
915 914 913 912 911 910 909 908 907 906 905 904 903 902 901 900 899 898 897 896 895 894 893
892 891 890 889 888 887 886 885 884 883 882 881 880 879 878 877 876 875 874 873 872 871 870
869 868 867 866 865 864 863 862 861 860 859 858 857 856 855 854 853 852 851 850 849 848 847
846 845 844 843 842 841 840 839 838 837 836 835 834 833 832 831 830 829 828 827 826 825 824
823 822 821 820 819 818 817 816 815 814 813 812 811 810 809 808 807 806 805 804 803 802 801
800 799 798 797 796 795 794 793 792 791 790 789 788 787 786 785 784 783 782 781 780 779 778
777 776 775 774 773 772 771 770 769 768 767 766 765 764 763 762 761 760 759 758 757 756 755
754 753 752 751 750 749 748 747 746 745 744 743 742 741 740 739 738 737 736 735 734 733 732
731 730 729 728 727 726 725 724 723 722 721 720 719 718 717 716 715 714 713 712 711 710 709
708 707 706 705 704 703 702 701 700 699 698 697 696 695 694 693 692 691 690 689 688 687 686
685 684 683 682 681 680 679 678 677 676 675 674 673 672 671 670 669 668 667 666 665 664 663
662 661 660 659 658 657 656 655 654 653 652 651 650 649 648 647 646 645 644 643 642 641 640
639 638 637 636 635 634 633 632 631 630 629 628 627 626 625 624 623 622 621 620 619 618 617
616 615 614 613 612 611 610 609 608 607 606 605 604 603 602 601 600 599 598 597 596 595 594
593 592 591 590 589 588 587 586 585 584 583 582 581 580 579 578 577 576 575 574 573 572 571
570 569 568 567 566 565 564 563 562 561 560 559 558 557 556 555 554 553 552 551 550 549 548
547 546 545 544 543 542 541 540 539 538 537 536 535 534 533 532 531 530 529 528 527 526 525
524 523 522 521 520 519 518 517 516 515 514 513 512 511 510 451 952 674 140 539 287 460 438
559 883 542 885 930 191

Thanks!