10GBASE-KR FEC Tutorial

Andre Szczepanek (TI)
Ilango Ganga (Intel)
Cathy Liu (LSI logic)
Magesh Valliappan (Broadcom)

Acknowledgements

- Jim Hamstra (Flextronics)
- Winston Mok (PMC Sierra)
  - For the OIF CEI-P FEC, & work on DFE error propagation

- Andrey Belogolovy (Intel)
- Andrey Ovchinnikov (Intel)
  - For Code selection and simulation

- Luke Chang (Intel)
- Fulvio Spagna (Intel)
- Joe Caroselli (LSI Logic)
  - For 802.3ap TF contributions

Agenda

- Introduction
- 802.3ap FEC requirements & code selection
- DFE Error propagation
- Simulated Performance of the FEC
- Ease of Implementation
- Conclusion

Introduction

What is the 10GBASE-KR FEC ?

- An optional sub-layer of 802.3ap (Backplane Ethernet)
  - A generic sublayer to the 10GBASE-R PCS
  - Could be used by other clauses
    - But only 10GBASE-KR has the AN support to enable it
- Transports 10GBASE-R 64b/66b codewords in FEC protected blocks
  - Within the same data-rate
- A lightweight FEC, with limited coding gain, that is simple to implement
  - Targeted at single burst error correction

802.3ap FEC requirements & code selection

Ilango Ganga, Intel

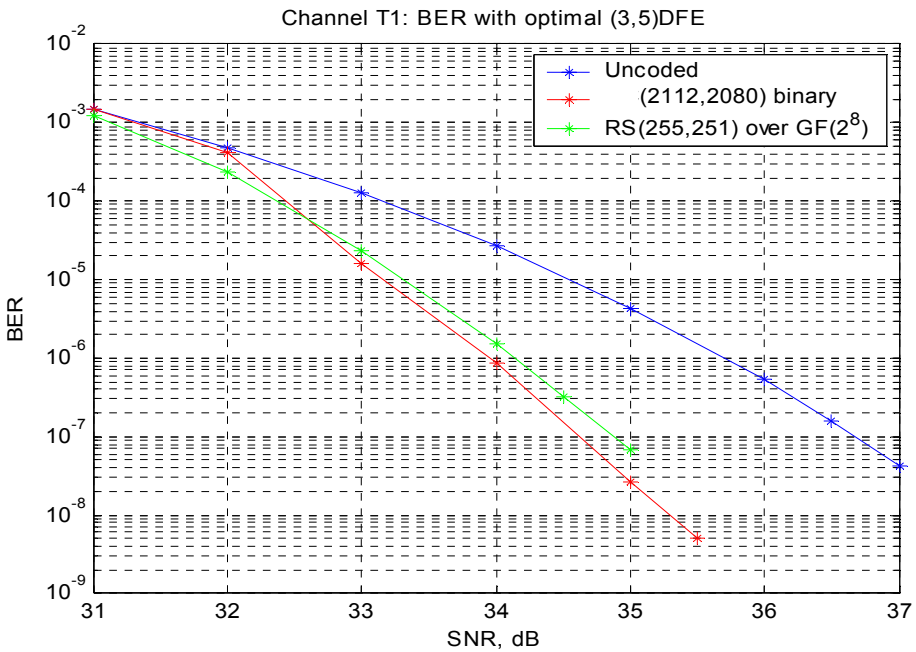
Objectives

- FEC to provide additional gain
  - BER objective of 10^{-12} or better on a broader set of backplane channels
  - Improve overall system reliability by significantly lowering BER
  - Improve Mean Time to False Packet Acceptance (MTTFPA) requirements for 10GbE
- Minimum changes to existing sublayers
  - Locate between PCS & PMA and be compatible with existing PCS (clause 49) & PMA (clause 51)
  - No increase in baud rate or decrease in payload rate
  - Low overhead (latency/area/power)
- Negotiate FEC capability through Auto-Negotiation

FEC overview

- Binary burst error correction code (2112, 2080)
  - Shortened cyclic code
  - Systematic: 2080 bits of payload and 32 bits of overhead
  - Can correct burst errors of up to 11 bits
- Modulation: NRZ
- Symbol rate: 10.3125G
- Compression and usage of 32 sync bits from 64B/66B blocks
- Compatibility with Clause 49 & Clause 51 (use of 16-bit data path as in XSBI)
- Synchronization at FEC block boundaries

Codes comparison



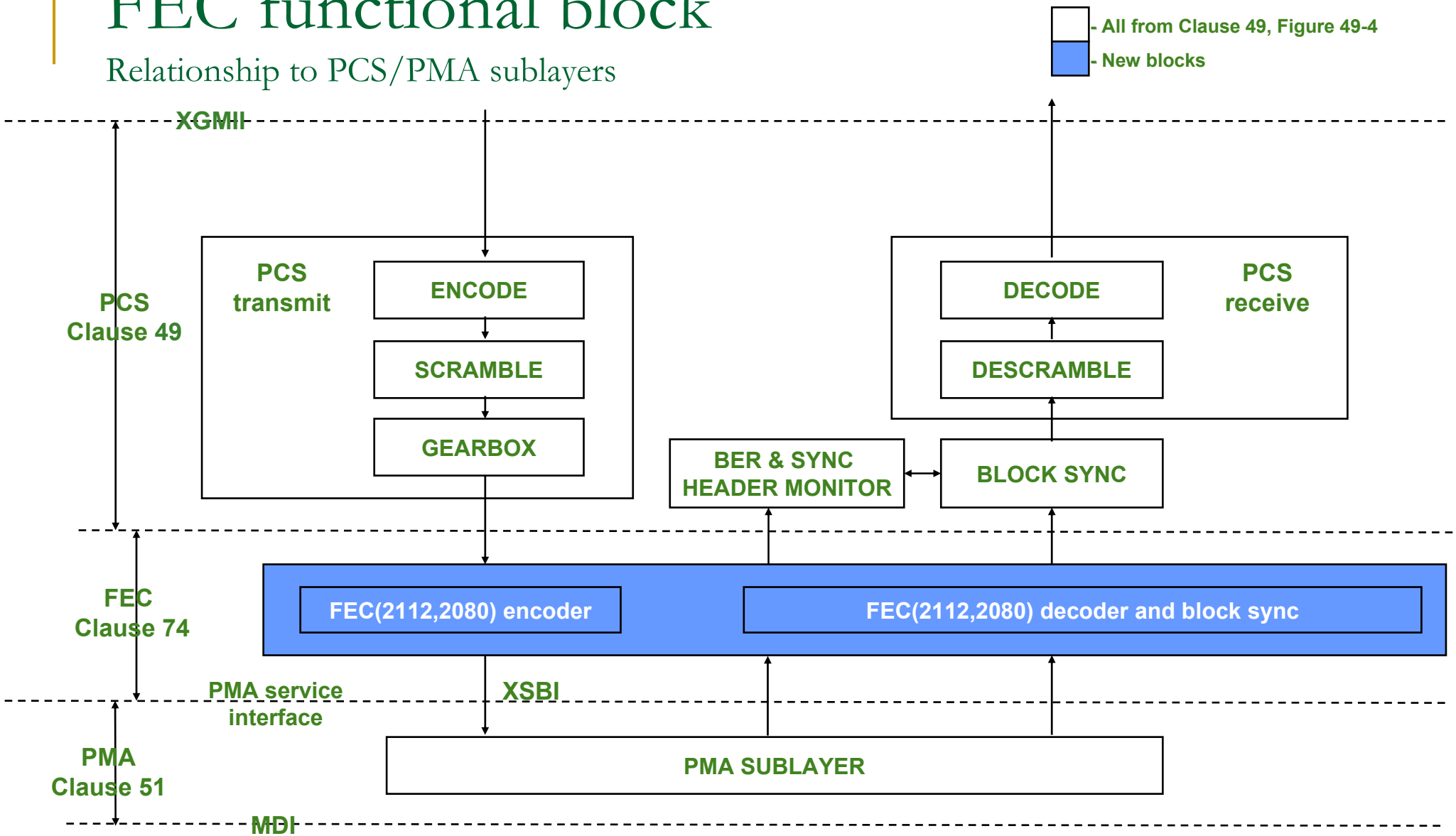
- Codes with 32 parity check bits were compared
  - Binary burst error correction code (2112,2080), with Meggitt decoder
  - RS(255,251) over GF(2^8) with Berlekamp decoder
- Coding gain of binary code is better
  - For the same coding gain (or better) RS codes should have 40 redundant bits
    - RS over GF(2^{10}) with 2 parity check symbols
- Can be implemented with a Meggitt decoder
  - Significantly simpler than Berlekamp decoders for RS codes

FEC code description

- **The (2112, 2080) burst error correction code is a shortened cyclic code with 32 redundant bits**
  - **Guaranteed errors burst length that can be corrected is t = 11 bits**
  - **It is a systematic code well suited for correction of the burst errors, typical in a backplane channel resulting from DFE error propagation**
  - **The (2112, 2080) code was constructed by shortening of cyclic code (42987, 42955)**
- **Generator polynomial**
  - **$g(x)=x^{32}+x^{23}+x^{21}+x^{11}+x^{2}+1$**
- **For (2112, 2080) code**
  - **encoder: systematic, represented by LFSR of length 32**
  - **decoder: Meggitt decoder for shortened cyclic codes**
  - **detector: syndrome calculation**
- **PN-2112 bit sequence**
  - **Generated by scrambler polynomial from Clause 49 $r(x)= x^{58}+x^{39}+1$ with initial state of $x^{57}=1$ and $x^{i-1}=x^{i}(XOR)1$ or binary 101010….**
  - **For every codeword PN-2112 sequence is returned to its initial state**
  - **Scrambling with PN-2112 sequence is necessary to maintain DC balance and to ensure FEC block sync (ensures any shift in code word is not equal to another)**

FEC functional block

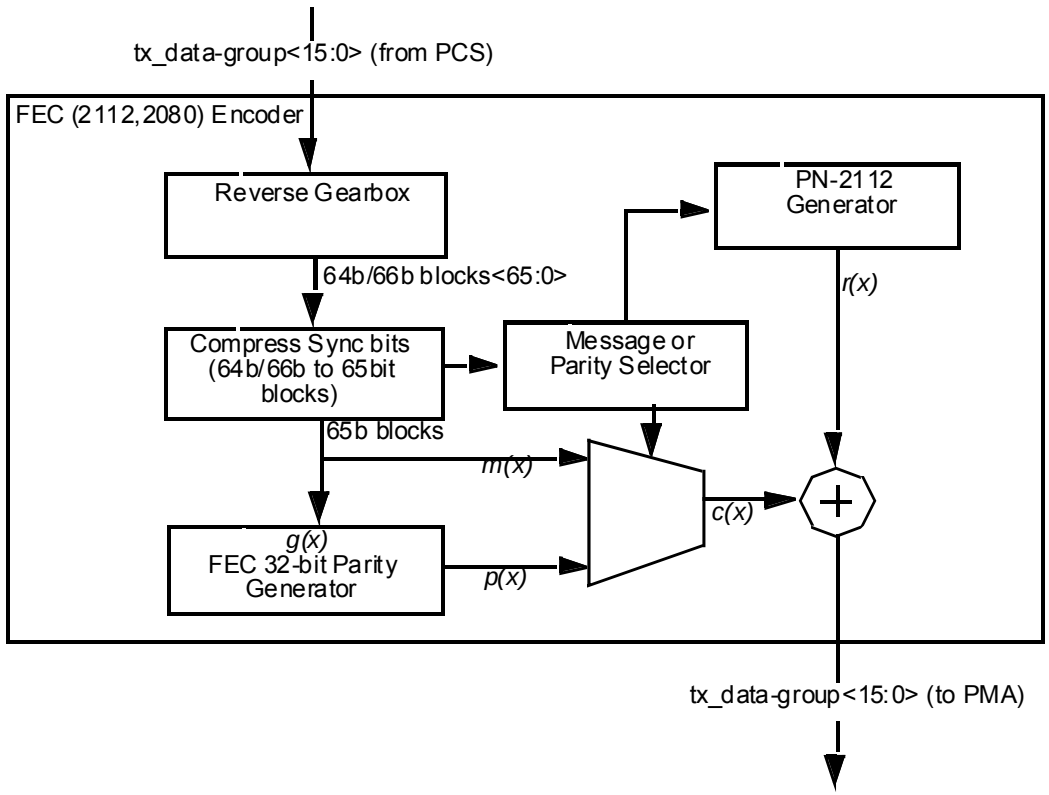
Relationship to PCS/PMA sublayers



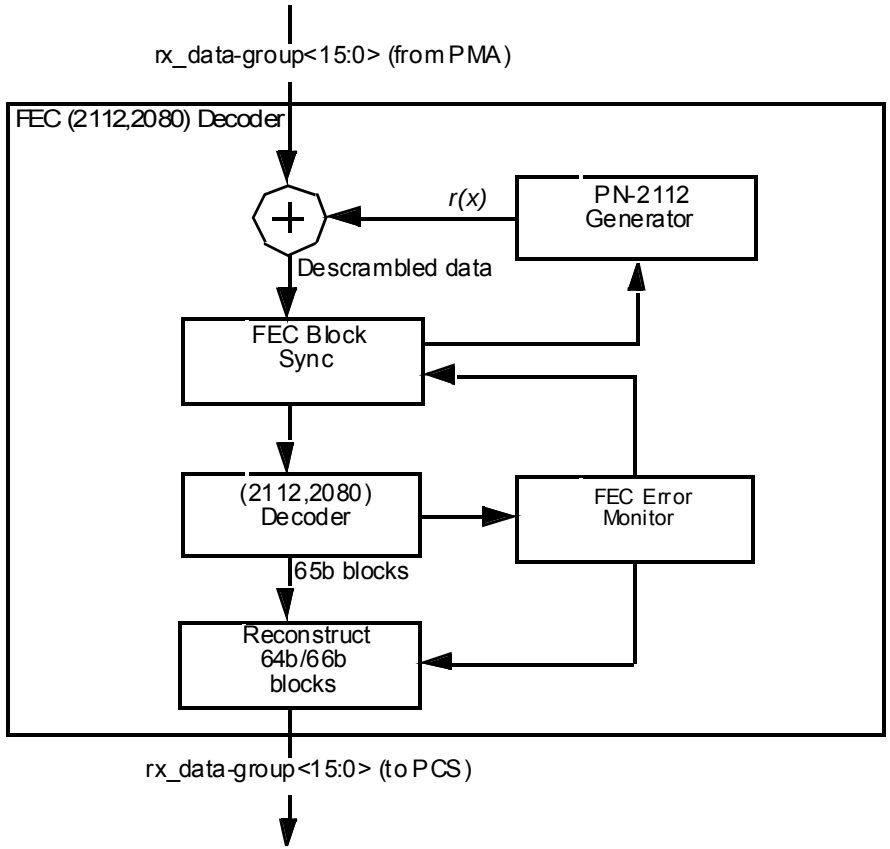
FEC sublayer

- Transparent to PCS and PMA (clauses 49 & 51)
  - 16-bit input and 16-bit output interface (optional)
  - Operates on 64b/66b block boundaries
- Uses shortened cyclic (2112, 2080) burst error correction code
  - For 32 parity check bits 1 of the 2 sync bits from 32 64B/66B blocks is used
  - Encoding latency is 32 bits
- Establishes synchronization at FEC block boundaries (32 64B/66B blocks)
  - Scrambling with PN-2112 sequence is necessary to maintain DC balance and to ensure FEC block sync (any shift in code word is not equal to another)
- Provides 2.0 - 2.5 dB energy gain

Block diagram of FEC sublayer



FEC (2112,2080) encoding



FEC (2112,2080) decoding

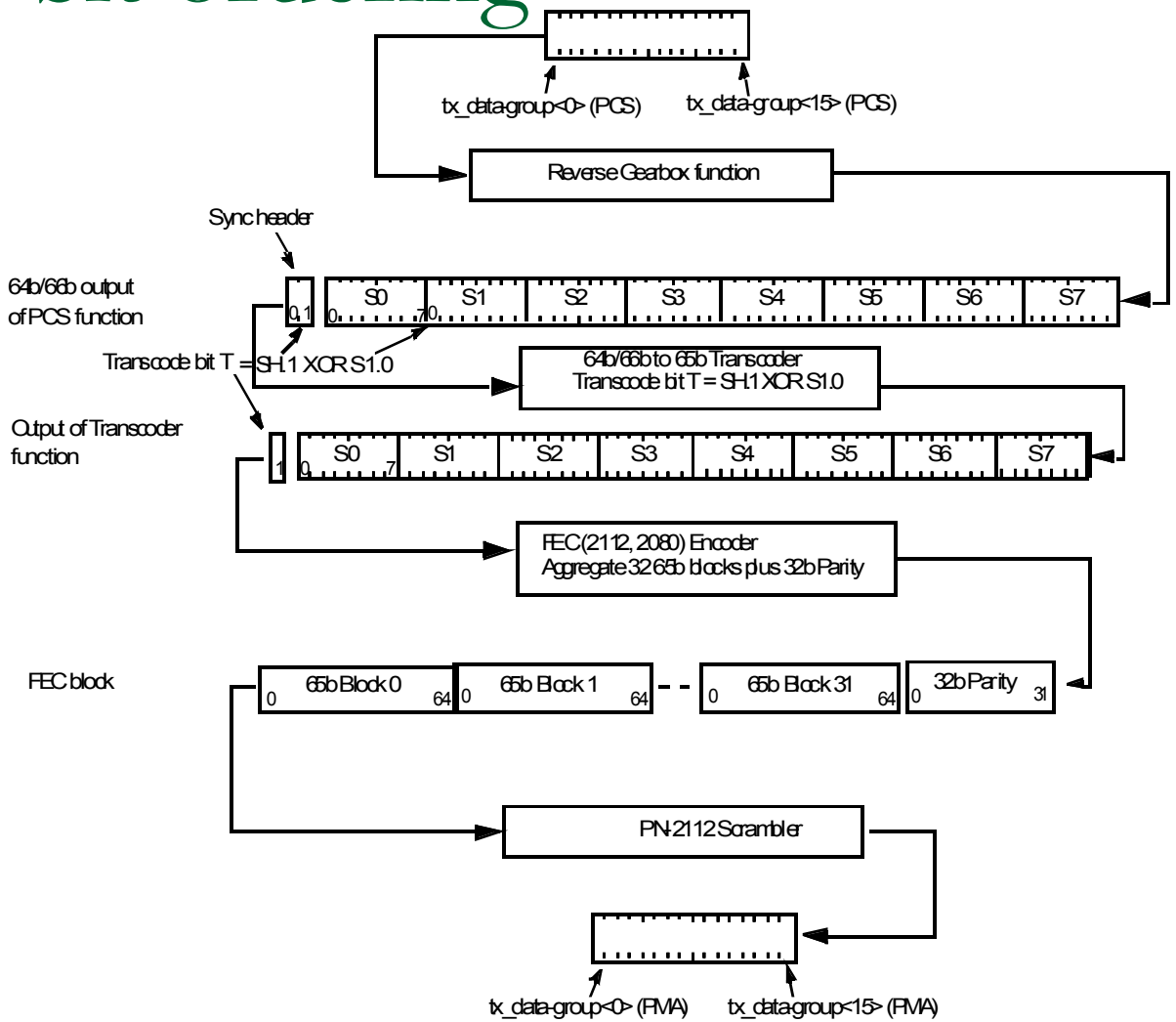
FEC block format

T_0	64 Bit Payload Word 0	T_1	64 Bit Payload Word 1	T_2	64 Bit Payload Word 2	T_3	64 Bit Payload Word 3
T_4	64 Bit Payload Word 4	T_5	64 Bit Payload Word 5	T_6	64 Bit Payload Word 6	T_7	64 Bit Payload Word 7
T_8	64 Bit Payload Word 8	T_9	64 Bit Payload Word 9	T_{10}	64 Bit Payload Word 10	T_{11}	64 Bit Payload Word 11
T_{12}	64 Bit Payload Word 12	T_{13}	64 Bit Payload Word 13	T_{14}	64 Bit Payload Word 14	T_{15}	64 Bit Payload Word 15
T_{16}	64 Bit Payload Word 16	T_{17}	64 Bit Payload Word 17	T_{18}	64 Bit Payload Word 18	T_{19}	64 Bit Payload Word 19
T_{20}	64 Bit Payload Word 20	T_{21}	64 Bit Payload Word 21	T_{22}	64 Bit Payload Word 22	T_{23}	64 Bit Payload Word 23
T_{24}	64 Bit Payload Word 24	T_{25}	64 Bit Payload Word 25	T_{26}	64 Bit Payload Word 26	T_{27}	64 Bit Payload Word 27
T_{28}	64 Bit Payload Word 28	T_{29}	64 Bit Payload Word 29	T_{30}	64 Bit Payload Word 30	T_{31}	64 Bit Payload Word 31
32 parity bits							

Total Block length = (32 x 65) + 32 = 2112 bits

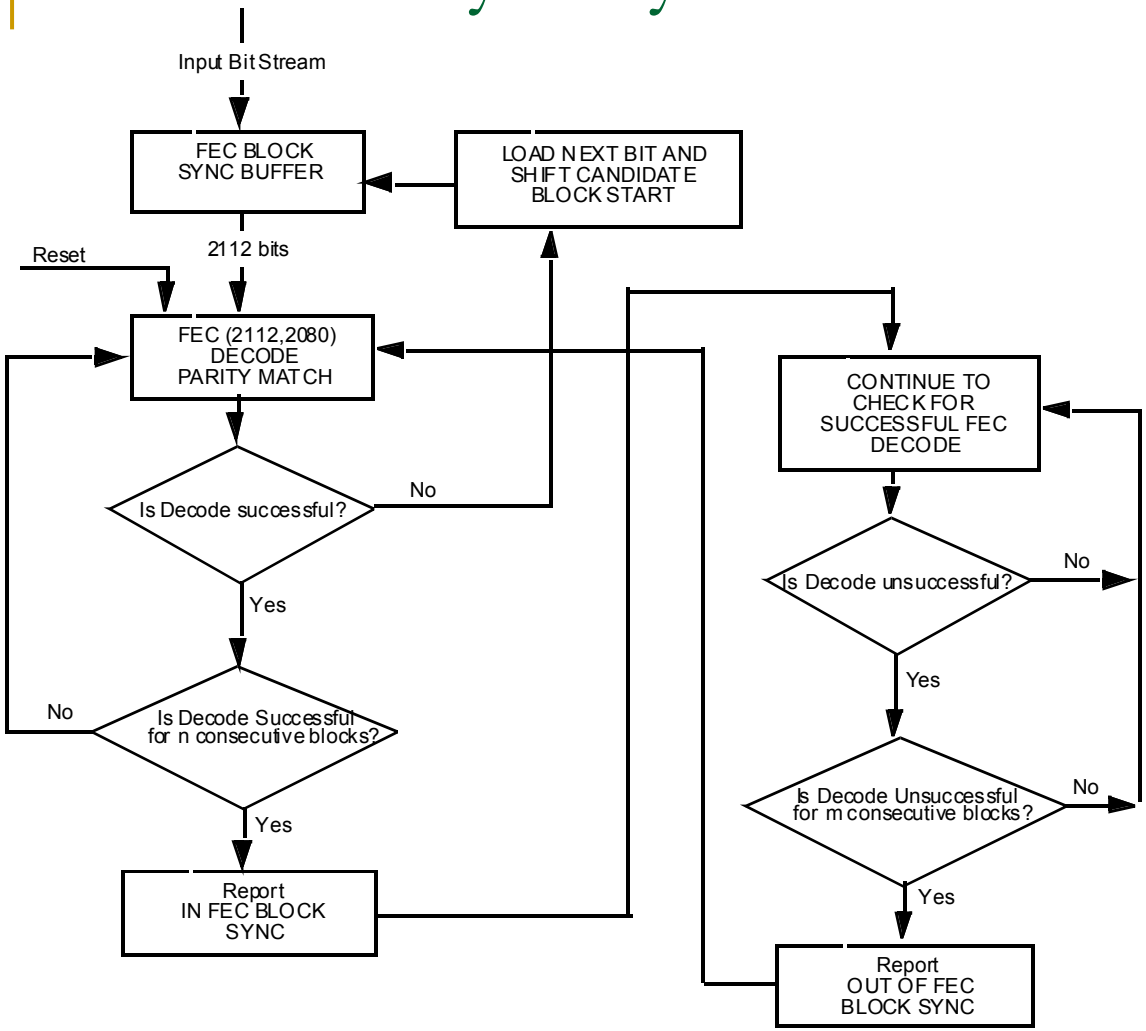
- Payload words carry the 10GBASE-R scrambled payload words
- Tn = Transcode bit carries the state of the 10GBASE-R sync bits for the associated payload word
  - Sync bits are compressed to a single bit then scrambled to ensure DC balance
  - 64b/66b sync bits are either 10 or 01 hence can be reconstructed from the T bit
  - Synchronization is achieved at FEC block level
- Block has the same overhead as 64B/66B encoding

Transmit bit ordering



FEC Transmit bit ordering

FEC sublayer synchronization



FEC (2112,2080) block sync and decoding

- Use conventional n/m serial locking techniques
- Similar to 64B/66B word sync State Machine
- Requires up to 2112 bit shifts to establish synchronization
- Uses error detection properties of (2112,2080) decoder and PN-2112 sequence for frame delineation
- Wrong synchronization probability is lower than 10^{-8}
- Loss of sync is reported if parity check failed for m >= 8 consecutive frames
- Sync is reported if parity check passed for n >= 4 consecutive frames

Auto-negotiation

- Auto-negotiation to advertise FEC capabilities for 10GBASE-KR PHY
- FEC capability for 10GBASE-KR PHY can be negotiated using Clause 73 AN
- Parameters advertised during Auto-Negotiation
  - FEC ability
  - FEC enable
- FEC is enabled on the link only if both link partners advertise they have FEC ability and one of them requests to enable FEC
- MDIO registers
  - 10GBASE-KR FEC ability and control registers to control FEC functionality through optional MDIO interface
  - Error correction statistics indicated in FEC corrected and uncorrected blocks counter registers

Summary

- The FEC code (2112, 2080) allows
  - to have ~2.0-2.5 dB energy gain
  - the BER of 10^{-12} or better on a broader set of channels
- 802.3ap test channels have error burst length of up to 11 bits
  - (2112, 2080) with minimum t = 11 bits is optimum code for 802.3ap channels
- Low latency
  - Encoder latency is 32 bits
  - Decoder latency is 2112+ bits (approx 200ns at 10G)
  - FEC function can be disabled to bypass decoder latency
- FEC block synchronization
  - 2112 bit block shifts will find lost sync, continuous sync monitoring during normal operation mode (uses conventional n/m serial locking techniques)
  - Required only at link start or in case of loss of connection

The Effect of DFE Error Propagation

Cathy Liu

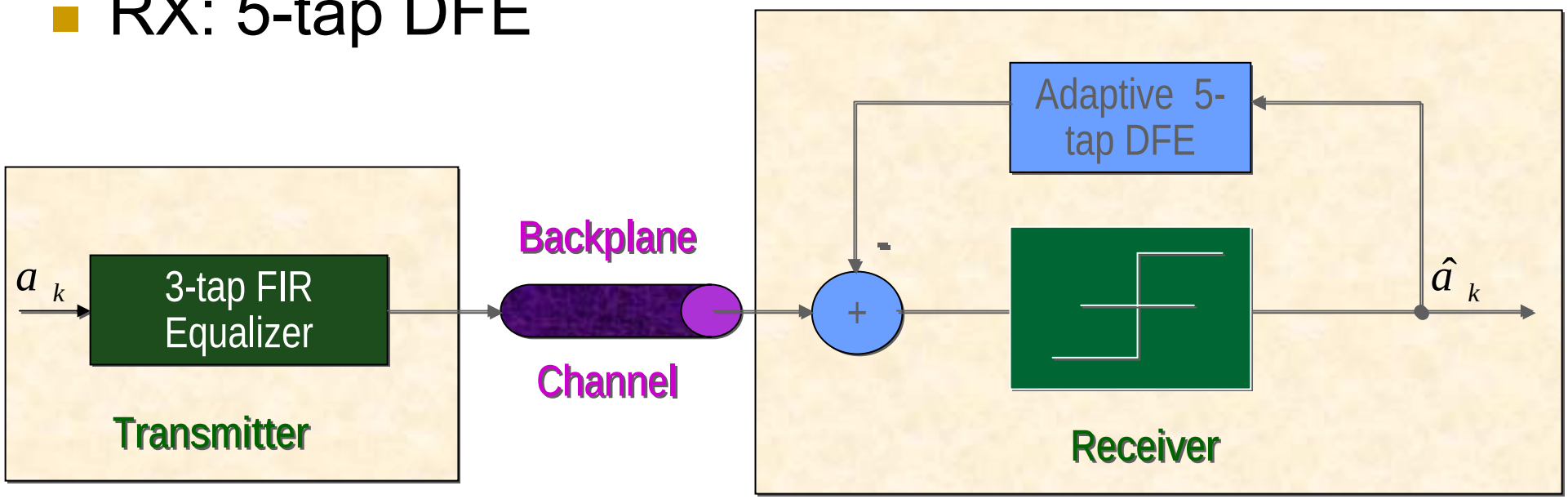
LSI Logic

Introduction

- A single bit error will produce a very long burst of errors with a high probability if a DFE with large number of taps (or large tap weights) is used.
- The effect of DFE error propagation on 10GBASE-KR channels is evaluated in this study.
  - Probability of long burst error run length
  - BER degradation
  - MTTFPA degradation
- Mitigation of DFE error propagation by using FEC is also evaluated in this study.

10GBASE-KR System Model

- 1/T=10.3125GHz
- TX: 3-tap transmit FIR
- RX: 5-tap DFE



Simulation Overview

- Analytic model is used to get slicer SNR at optimal sampling point.
  - Includes
    - Intersymbol Interference
      - Channel data: Tyco, Molex and Intel
      - A simple RC model with pole at 0.75*baud rate is used for the transmitter.
      - Mellitz capacitor-like package model included on both transmitter and receiver.
    - Tx Jitter
      - 0.013UI σ RJ added
    - Electronics White Noise
      - Signal-To-Electronics Noise Ratio 42dB
    - Crosstalk
      - One near-end crosstalk aggressor (the worst one) is considered
  - Does Not Include
    - Receiver Sensitivity
    - Duty Cycle Distortion
    - Other Sources of DJ
- 3-tap FIR and 5-tap DFE tap values are ideal.

DFE Error Propagation

- $\{y_1\, y_2\, y_3\, y_{4\ldots}\, y_n\}$ and $\{e_1\, e_2\, e_3\, e_{4\ldots}\, e_n\}$ are the received signal and the corresponding error pattern.
  - y_1 and e_1 are the oldest.
- $p(e_i|E)$: the probability that the detection of y_i is wrong under the assumption that an error pattern E happened in the pervious bits.
  - $p(e_2|\{e_1\}=1)$: the probability of y_2 in error assuming y_1 is in error.
  - $p(e_i|\{e_1,e_2\}=\{1,1\})$: the probability of y_i in error assuming y_1 and y_2 are both in error, $i > 2$.
- $p(e_i|E)$ can be calculated by our analytic simulator.

DFE Error Propagation $p(e_i|E)$:

BP	Tyco1	Tyco2	Tyco3	Tyco4	Tyco5	Tyco6	Tyco7
BER w/o E	3.02E-24	1.02E-24	1.41E-18	4.25E-32	3.10E-40	3.13E-20	2.17E-26
p(e2|{e1}=1)	2.17E-03	4.58E-03	1.94E-02	6.30E-12	8.80E-09	7.59E-02	1.15E-01
p(e3|{e1}=1)	9.79E-04	9.34E-04	1.07E-05	2.54E-04	3.43E-25	1.52E-17	3.96E-04
p(e4|{e1}=1)	3.07E-06	6.62E-06	5.02E-06	5.47E-08	1.80E-30	2.88E-15	1.13E-06
p(e5|{e1}=1)	3.06E-24	1.33E-22	1.63E-15	9.26E-20	1.31E-13	6.33E-07	1.46E-16
p(e6|{e1}=1)	3.16E-24	1.02E-24	4.77E-18	1.11E-31	1.86E-37	7.78E-18	1.44E-10

BP	Mi2	Mi3	Mi4	Mi5	Mo2	Mo3	Mo4	Mo5
BER w/o E	2.24E-19	7.02E-19	1.01E-18	1.48E-19	5.66E-21	1.60E-20	6.45E-21	2.91E-21
p(e2|{e1}=1)	6.00E-04	2.18E-03	1.93E-03	2.85E-03	7.28E-05	1.08E-05	6.64E-05	1.45E-04
p(e3|{e1}=1)	1.98E-04	5.15E-04	1.18E-04	6.20E-05	5.19E-04	1.78E-04	3.34E-04	1.19E-04
p(e4|{e1}=1)	2.65E-19	7.31E-19	3.87E-18	2.68E-19	1.67E-19	3.36E-19	6.08E-19	1.47E-17
p(e5|{e1}=1)	2.26E-19	7.86E-19	2.69E-17	3.04E-19	1.19E-17	3.08E-18	1.80E-19	4.12E-18
p(e6|{e1}=1)	4.96E-16	9.44E-15	8.78E-15	2.68E-18	6.52E-19	8.28E-18	1.02E-18	6.04E-21

BP	B1	B12	B20	M1	M20	T1	T12	T20
BER w/o E	5.62E-22	2.06E-20	1.52E-17	4.83E-16	6.46E-18	5.10E-11	5.52E-09	3.47E-07
p(e2|{e1}=1)	2.57E-01	1.07E-01	3.16E-02	9.39E-02	2.53E-03	3.07E-01	2.12E-01	1.44E-01
p(e3|{e1}=1)	1.77E-02	1.84E-04	7.58E-15	3.24E-06	2.31E-09	9.71E-02	1.83E-04	1.63E-04
p(e4|{e1}=1)	4.59E-16	7.57E-20	9.33E-16	3.78E-04	7.13E-10	5.67E-10	4.07E-07	1.94E-03
p(e5|{e1}=1)	6.80E-09	1.09E-08	1.00E-08	3.13E-09	2.93E-08	1.51E-04	4.32E-05	4.20E-05
p(e6|{e1}=1)	5.82E-22	5.14E-16	3.83E-15	3.64E-15	2.11E-14	1.41E-08	4.64E-06	8.84E-06

Run Length of Errors

- $p(rll=i)$: probability of a burst error with run length equal to i.
- rll_{max}: The maximum error run length to be considered.
  - It is related with the total number of DFE taps and the tap weights.
  - $rll_{max}=17$ in our simulation.

Run Length of Errors (cont.)

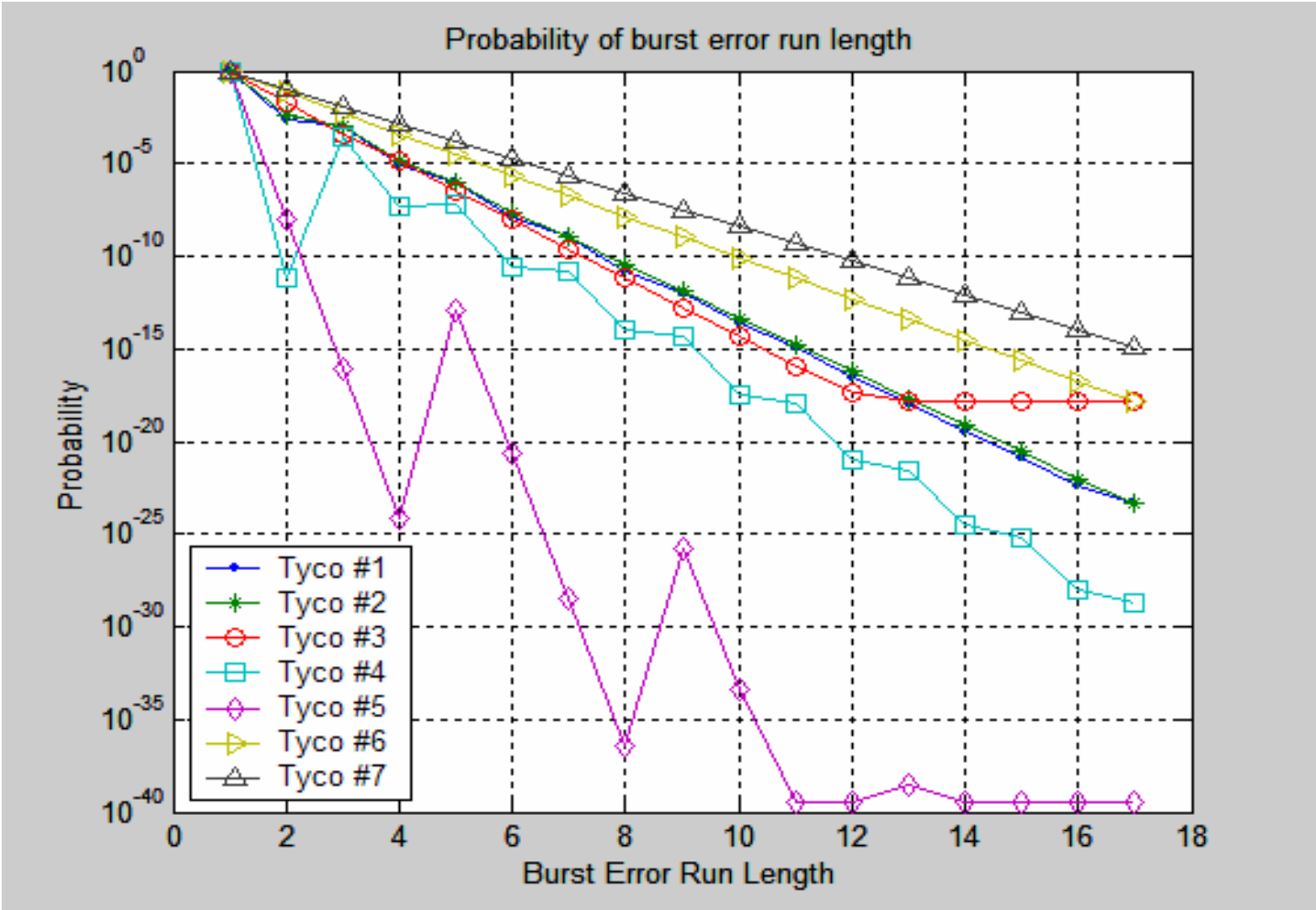
$$p(rll=1) = \prod_{i=2}^{rll_{\max}+1} (1 - p(e_i \mid \{e_1\} = 1))$$

$$p(rll=2) = p(e_2 \mid \{e_1\} = 1) \cdot \prod_{i=3}^{rll_{\max}+2} (1 - p(e_i \mid \{e_1, e_2\} = \{1,1\}))$$

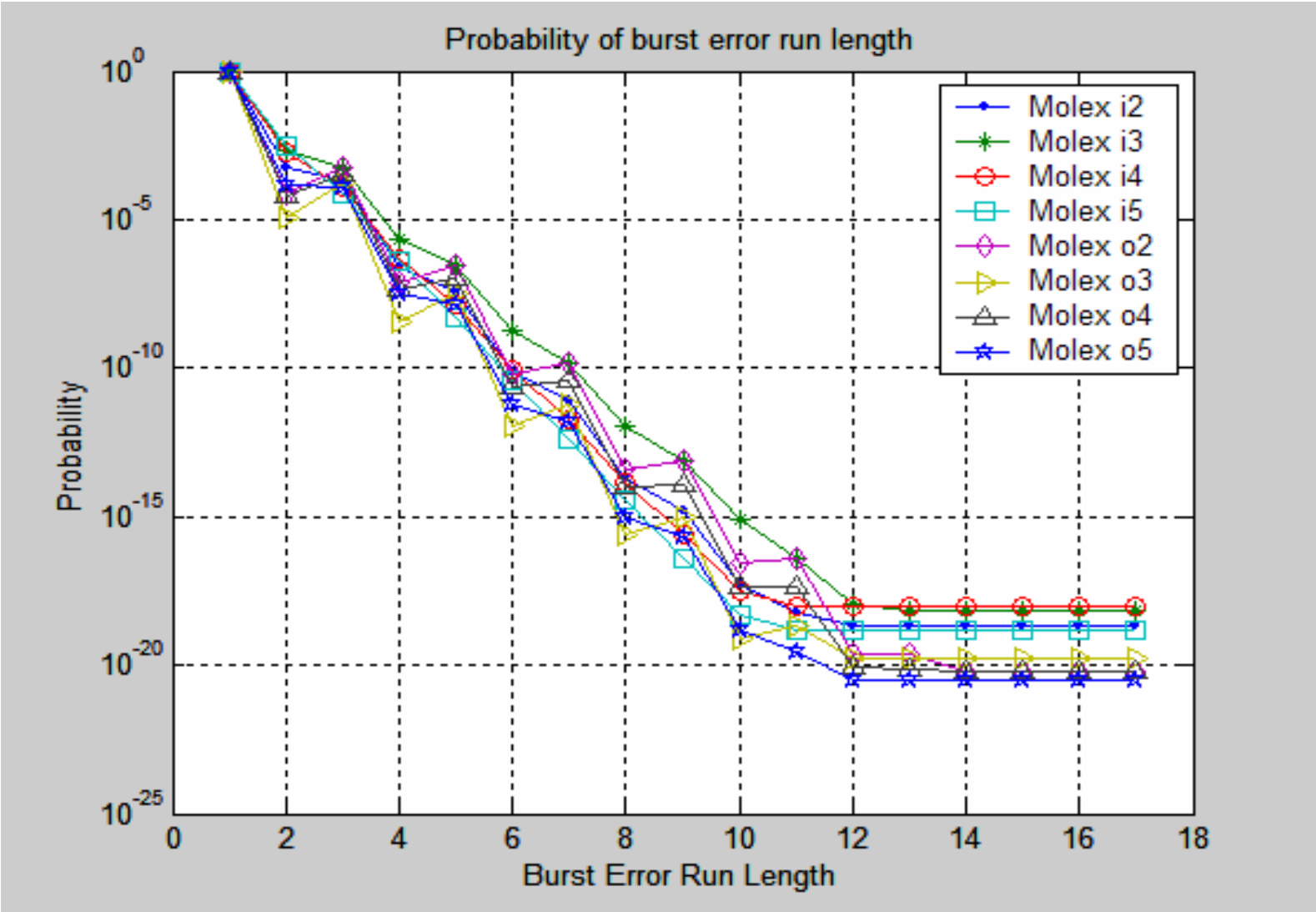
$$p(rll=3) = \underbrace{p(e_2 \mid \{e_1\} = 1) \cdot p(e_3 \mid \{e_1, e_2\} = \{1,1\}) \cdot \prod_{i=4}^{rll_{\max}+3} (1 - p(e_i \mid \{e_1, e_2, e_3\} = \{1,1,1\}))}_{p(E_{3,1}=\{111\})}$$

$$+ \underbrace{(1 - p(e_2 \mid \{e_1\} = 1)) \cdot p(e_3 \mid \{e_1, e_2\} = \{1,0\}) \cdot \prod_{i=4}^{rll_{\max}+3} (1 - p(e_i \mid \{e_1, e_2, e_3\} = \{1,0,1\}))}_{p(E_{3,2}=\{101\})}$$

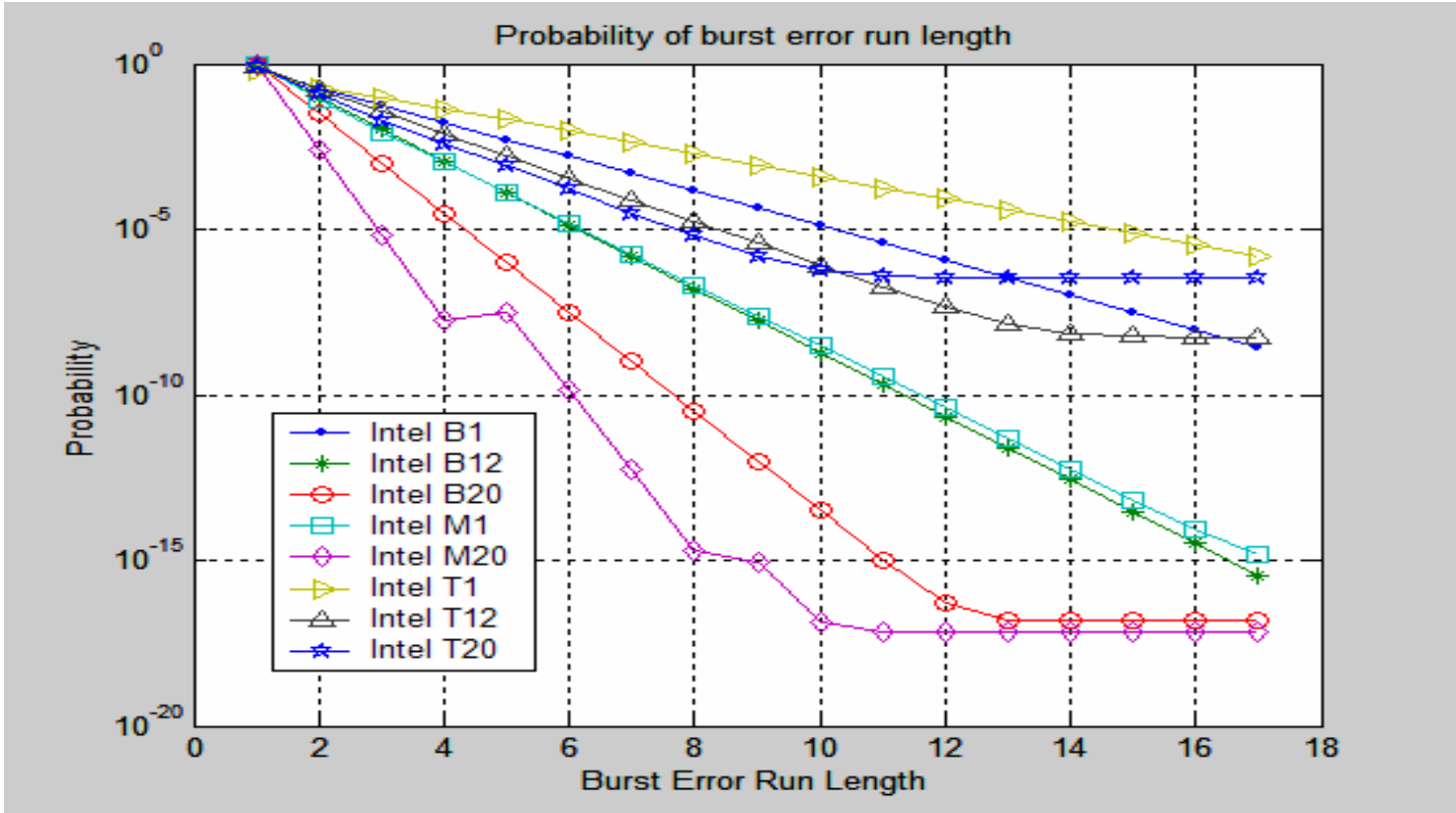
$p(rll{=}i)$: Tyco


Probability of burst error run length
Probability
Burst Error Run Length
Tyco #1
Tyco #2
Tyco #3
Tyco #4
Tyco #5
Tyco #6
Tyco #7

$p(rll{=}i)$: Molex


Probability of burst error run length
Molex i2
Molex i3
Molex i4
Molex i5
Molex o2
Molex o3
Molex o4
Molex o5
Probability
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{-20}
10^{-25}
0
2
4
6
8
10
12
14
16
18
Burst Error Run Length

$p(rll{=}i)$: Intel



- Normally, those channels that require larger DFE tap weights have larger $p(rll{=}i)$ for long error run length.

BER with DFE Error Propagation

- n: considered block length.
- p_1 is the probability that a bit in error when all previous bits are error free, i.e., random bit error rate.
- $p(W(E)=i)$: probability that i bits in error among n block bits.
- $W(E)$: weight of error pattern E .
  - $W(101)=2$.

BER with DFE Error Propagation (cont.)

- Block error rate

$$p_{block} = \sum_{i=1}^{\infty} P(W(E) = i)$$

$$\approx \sum_{i=1}^{\infty} p(burst\ error\ of\ \ i\ bits)$$

$$\approx \sum_{i=1}^{rll_{\max}} n \cdot p_1 \cdot p(rll = i) \cdot (1 - p_1)^{n - rll_{\max} - i}$$

- Bit error rate (BER)

$$BER = p_{bit} = \sum_{i=1}^{rll_{\max}} \sum_{\text{all } E} p(rll = i, E) \cdot W(E) \cdot p_1 \cdot (1 - p_1)^{n - rll_{\max} - i}$$

BER with DFE Error Propagation (cont.)

BP	Tyco1	Tyco2	Tyco3	Tyco4	Tyco5	Tyco6	Tyco7
P_1	**3.02E-24**	**1.02E-24**	**1.41E-18**	**4.25E-32**	**3.10E-40**	**3.13E-20**	**2.17E-26**
BER	**3.03E-24**	**1.03E-24**	**1.44E-18**	**4.25E-32**	**3.10E-40**	**3.39E-20**	**2.45E-26**

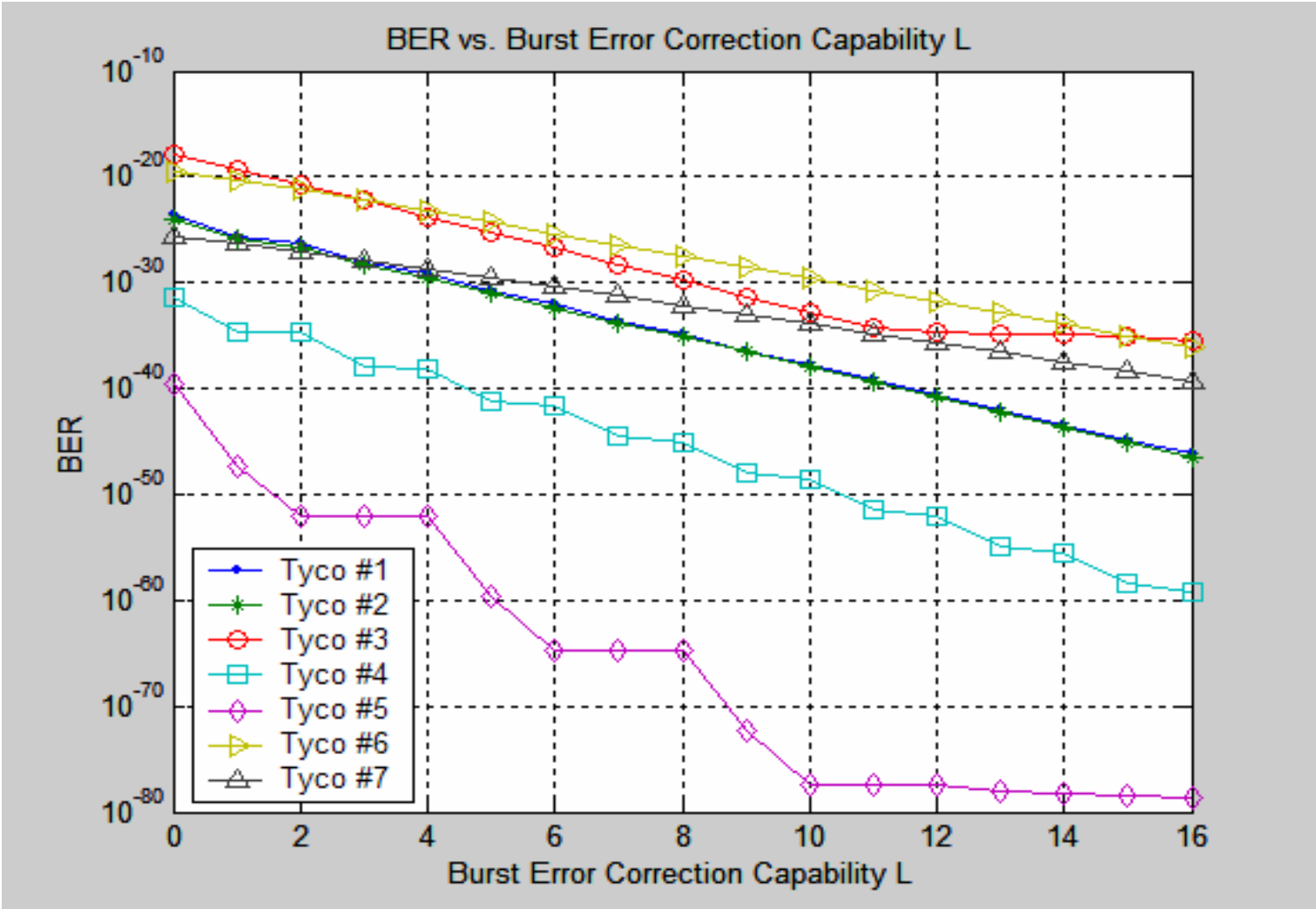
- The DFE error propagation may not degrade BER much, but the high probability of having long burst errors may:
  - result in catastrophic increases in post-FEC bit error rate, hence burst-error-correcting codes are more desirable.
  - significantly increase error detection failure rate for some applications like Ethernet.

BER with Burst error correcting FEC Coding

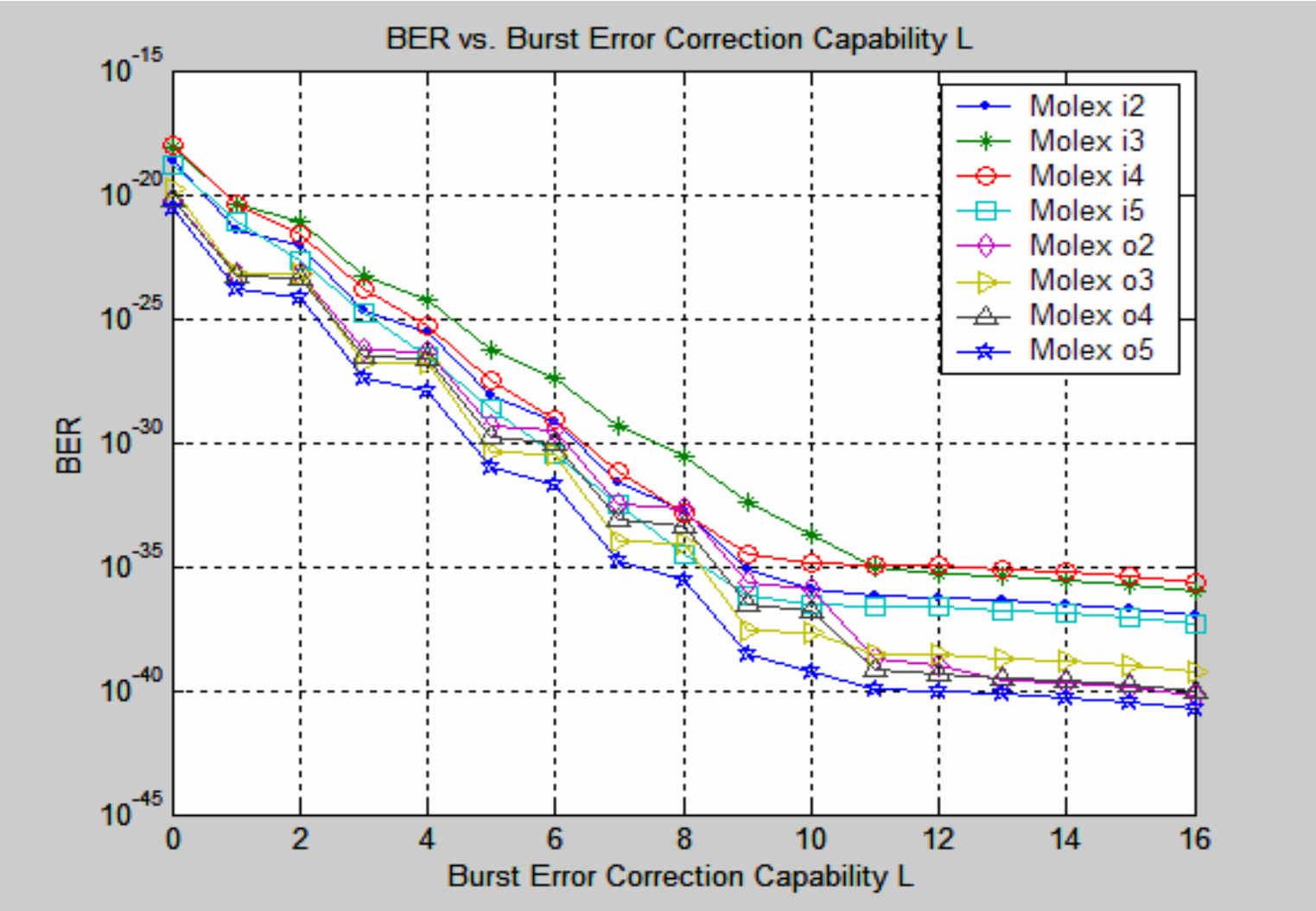
- Burst error correcting capability =L.
- Bit error rate

$$BER_{post-FEC} = \sum_{i=L+1}^{rll_{\max}} \sum_{\text{all } E} p(rll = i, E) \cdot W(E) \cdot p_1 \cdot (1 - p_1)^{n - rll_{\max} - i}$$

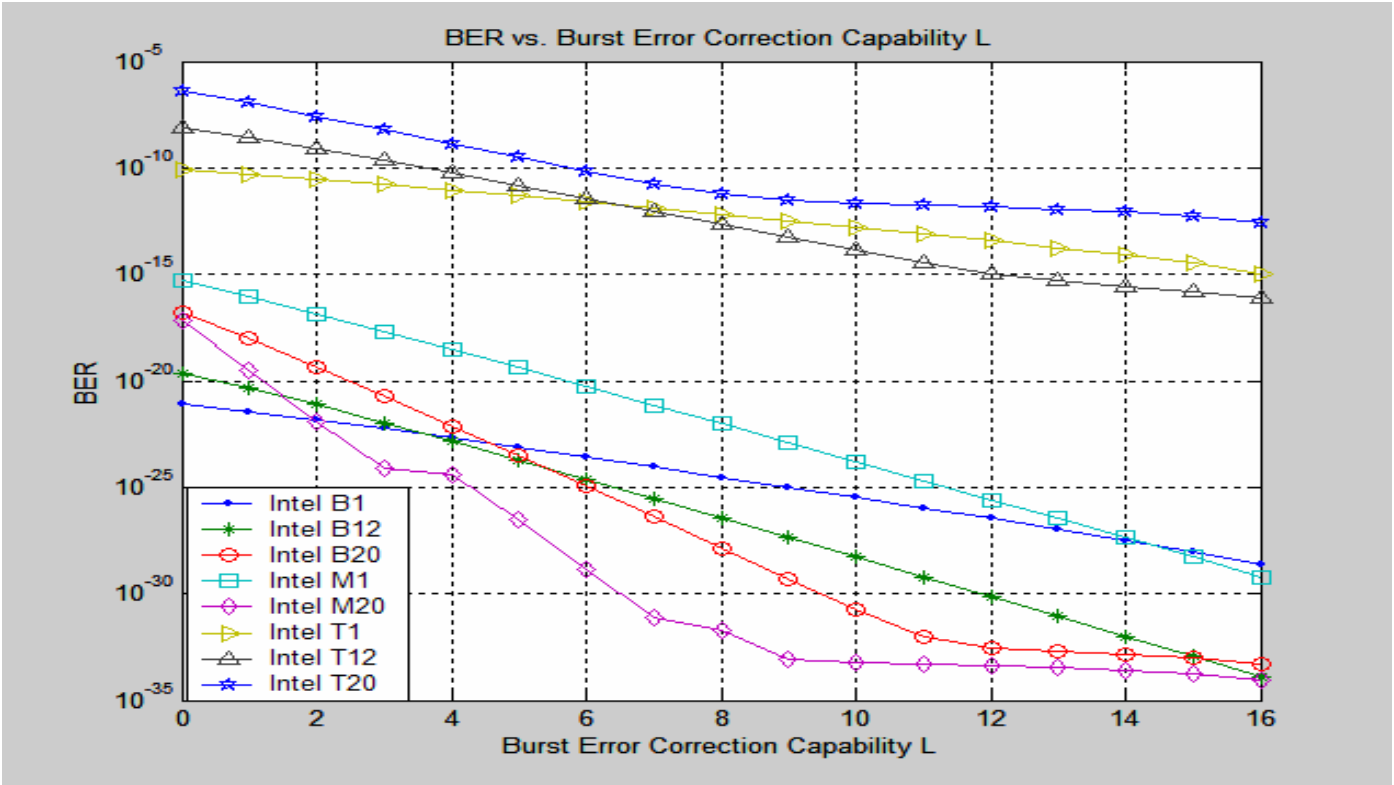
BER improvement: Tyco



BER improvement: Molex



BER improvement: Intel



- For some channels, L=11 can improve BER from 10^{-10} to 10^{-12}.

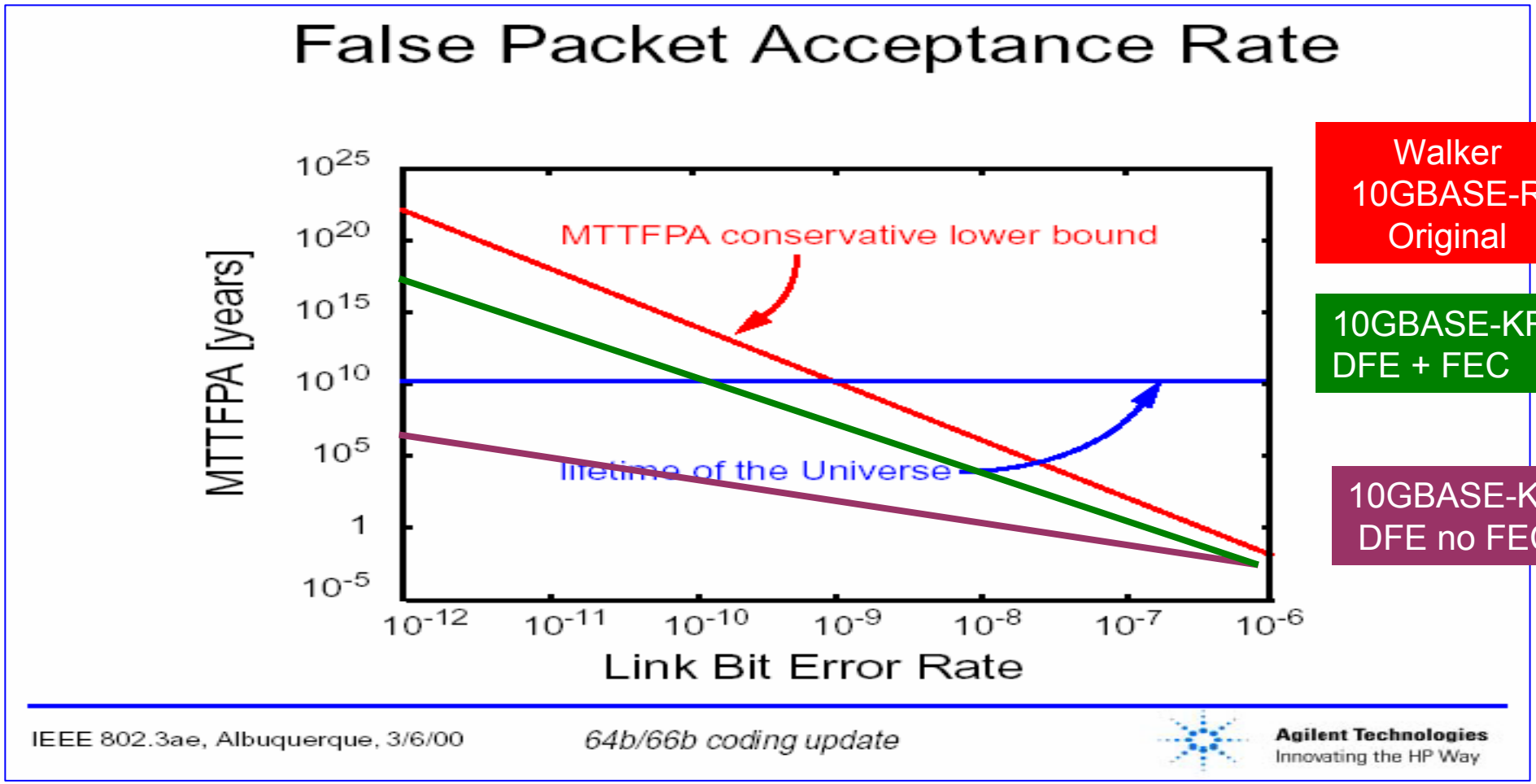
DFE Error propagation & MTTFPA

- The Ethernet CRC32 has considerable error detection capability
  - Hamming distance of 4 to detect any 3 bits in error in packet
  - Also detect any 32-bit burst or any two 8-bit burst in packet
  - This provides an Mean Time to False Packet Acceptance (MTTFPA) in the billions of years.
- The self-synchronous scrambler in 64B66B compromises the burst error detection capabilities of the Ethernet CRC32.
- Because 64B66B still has a uniform 4-bit Hamming protection, an estimate can be made MTTFPA $>2^{32}$ × expected time for 4 or more errors.
- For non-bursty channel, the probability of getting 4 or more errors is quite low. Hence, MTTFPA is still acceptable. (walker_1_0300.pdf)

DFE Error propagation & MTTFPA (Cont.)

- However, due to the DFE error propagation the probability of getting 4 or more errors is quite high (can be as high as 10^{-3} for some channels), which degrades MTTFPA significantly.
  - MTTPFA > 13.1M years ($p_1=10^{-12}$, 10.3Gbps, $p(rll=4)=10^{-3}$)
  - Compared with the life time of universe, billions of years, DFE error propagation reduces MTTFPA.
- For those compliance channels, with FEC of L=11 can reduce $p(rll=4)<10^{-10}$
  - 10GBASE-KR FEC (2112,2080) will improve MTTFPA of the worst channel to 2.9E+14 years.

1E-3 FEC results overlaid on Walker graph



FEC MTTFPA based on probability of >3 bit burst = 1E-3, & probability of >11 bit burst = 1E-10

Summary of DFE Error Propagation

- Due to the DFE error propagation, BER degradation may not severe, but the high probability of long burst error may cause large error detection failure rate.
- Better MTTFPA (lifetime of the universe 10^{10} years) can be obtained if FEC (L=11) is used.
  - MTTPFA of 13.1M years -> 2.9E+14 years
- Furthermore, burst error FEC can improve performance margins.
  - Substantially Improves the BER of barely compliant links

Simulation Results

Magesh Valliappan
Broadcom

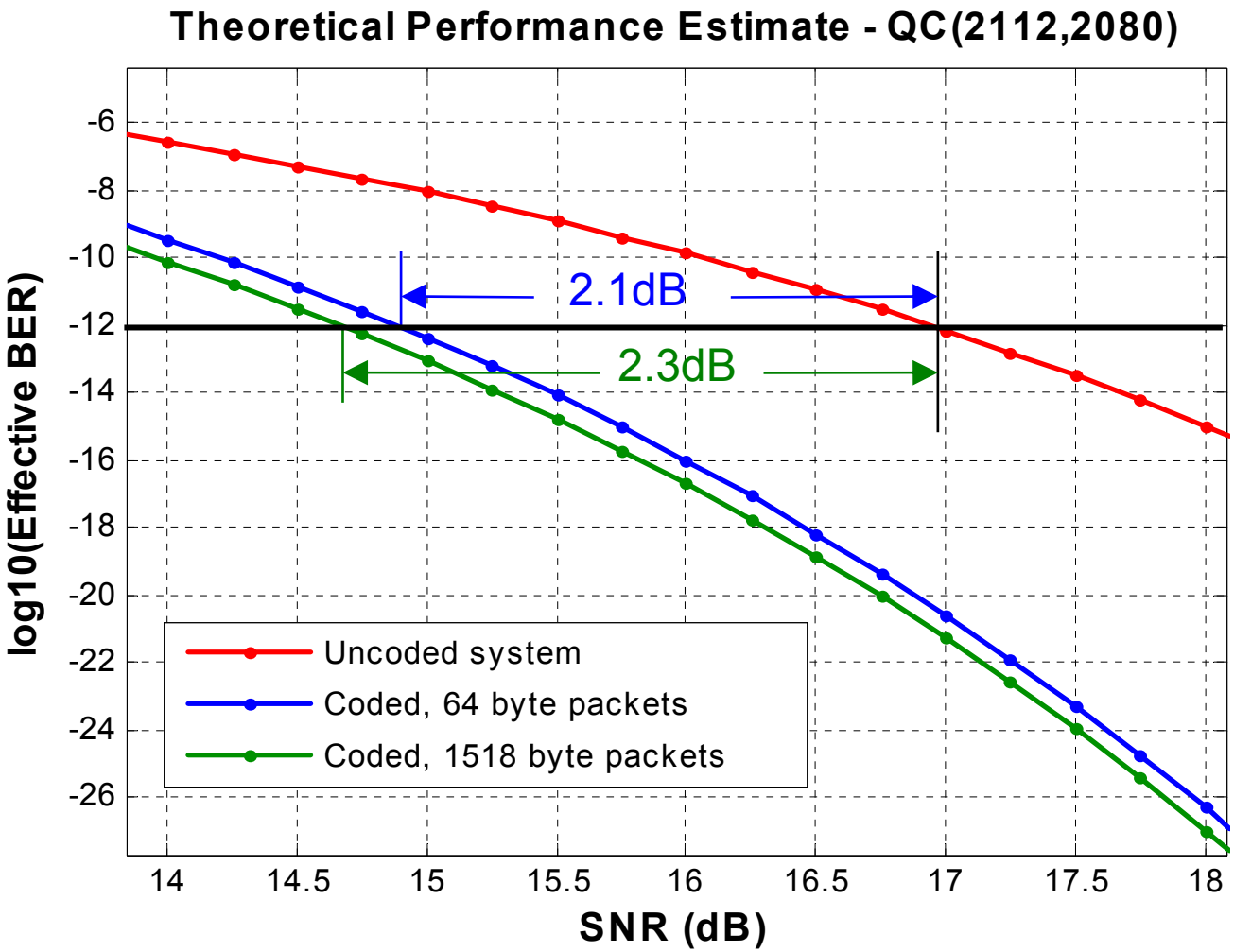
Simulation Results

- Theoretical Estimate of Performance
- Time Domain Simulation Based Performance Estimates
  - Results from 2 independent simulation environments presented in
    - valliappan_01_1105.pdf
    - ganga_02_1105.pdf
- Synchronization and Unlock Time Estimates

Theoretical Coding Gain Estimate

- Assumptions:
  - Assume memory-less Binary Symmetric Channel (BSC), but with finite error propagation (to model DFE effects)
  - Probability of first error in a burst based on Gaussian noise model
  - DFE does not propagate errors beyond decoder's capability
  - Uncorrected errors are detected and entire code block is rejected
- Method:
  - Estimate Ethernet Frame Error Rate for each SNR using probabilities
  - Translate Frame Error Rate to "Effective BER" of ideal memory-less BSC with Gaussian noise
- Results
- Coding Gain Estimate (from Analysis):
  - 64 byte packets: 2.1 dB
  - 1518 byte packets: 2.3 dB

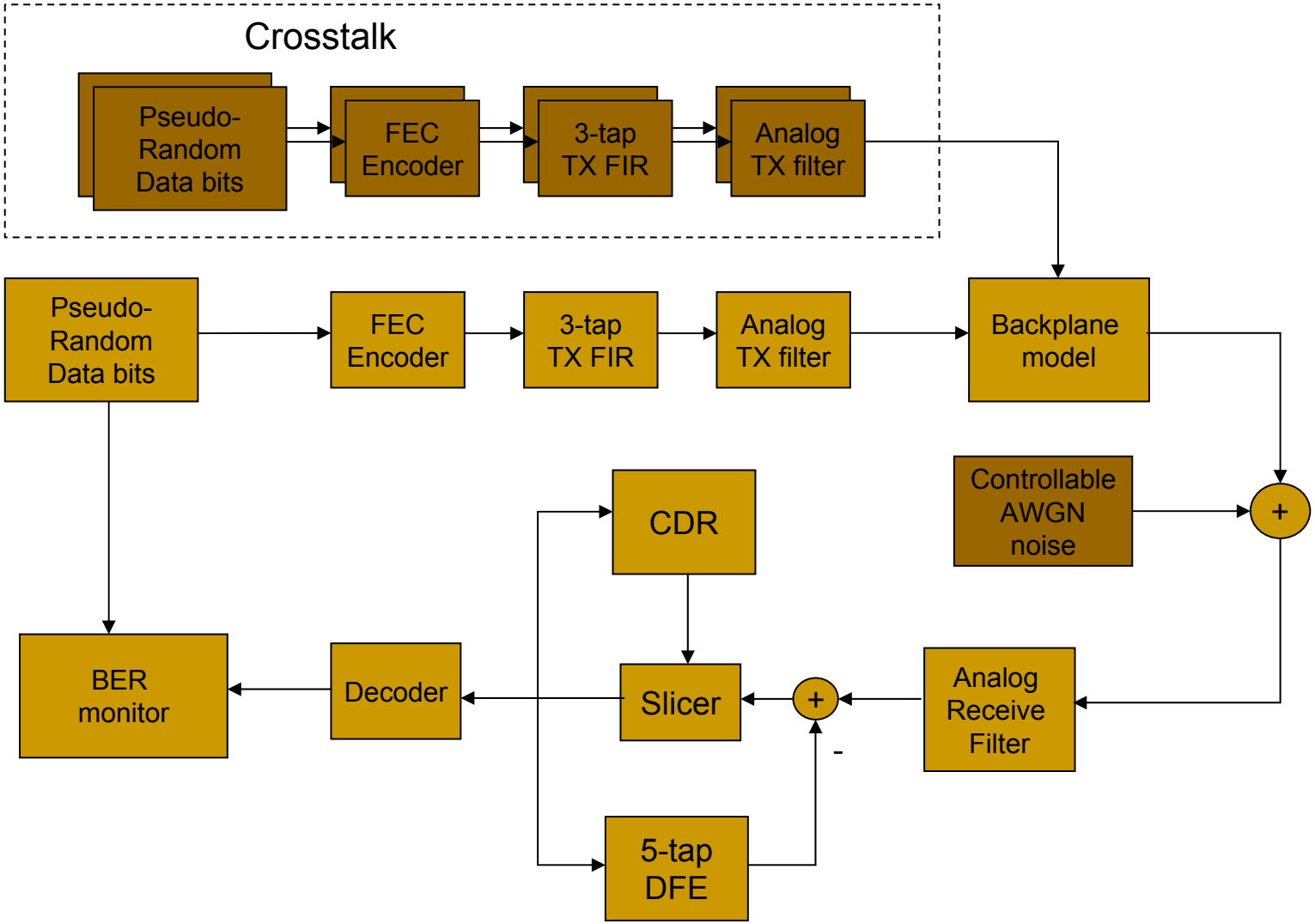
Theoretical Coding Gain Estimates



Time Domain Simulation Results – I

Courtesy of Broadcom

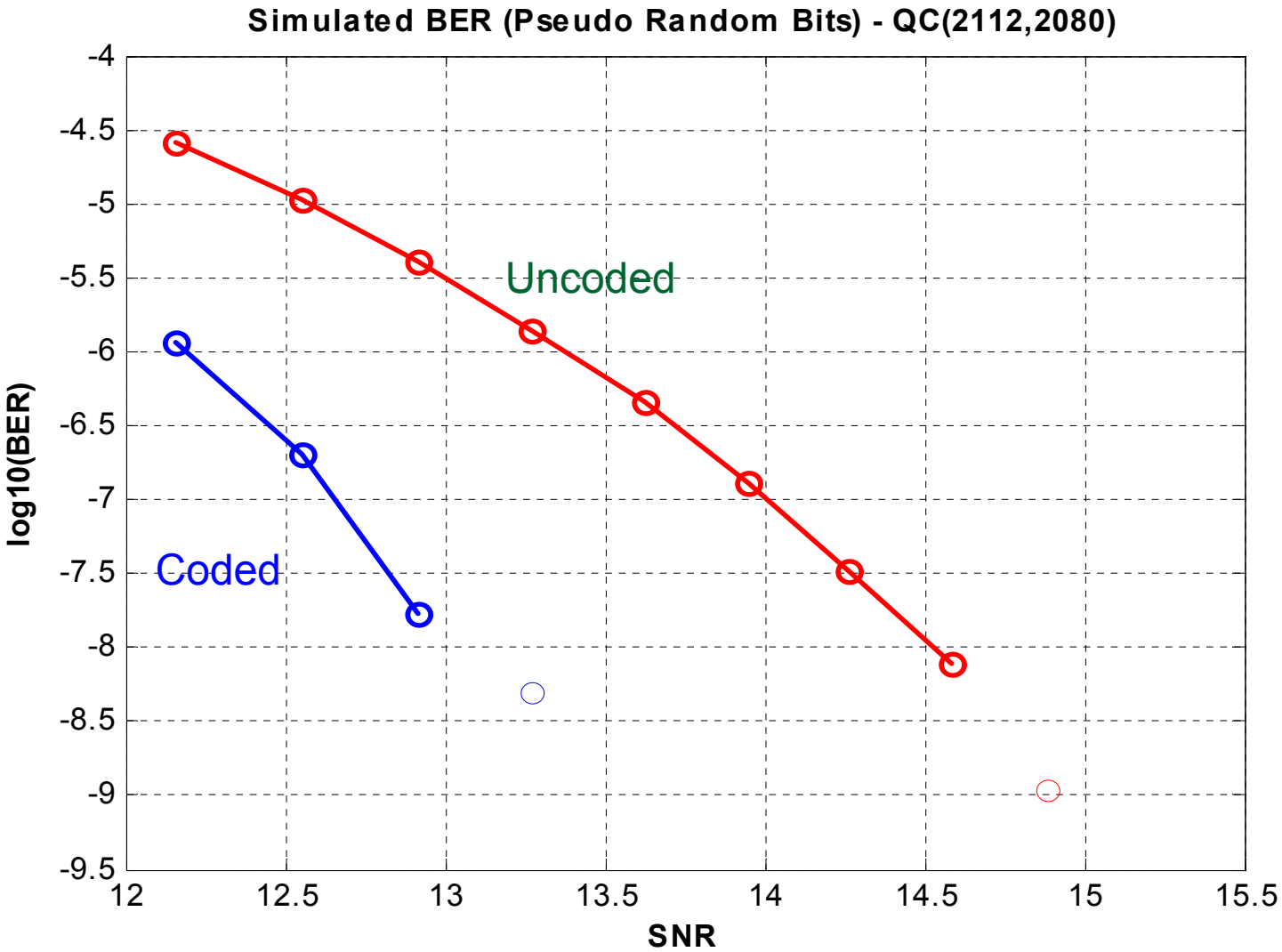
Time Domain Simulation Setup



Time Domain Simulation Setup

- Tyco Case 7 Backplane model from 802.3ap website
- TX
  - 0.01 UI RJ
  - 3-tap FIR
  - Simple RC model for transmitter
- Package model included
- Crosstalk – 2 aggressors, with similar TX configuration
- RX
  - Adaptive CDR circuit
  - 5-tap DFE
  - Ideal slicer
- AWGN noise source controlled to change BER
- Simulated over 1800 million bits
- Pseudo random bits, PCS not modeled

Simulation Results



Simulation Results

- Consistent with 2 - 2.5 dB coding gain at 1e-12
  - Projected to 1e-12, effective SNR improvement = **2.35dB**
    - Assuming no error floor
    - 1.20dB at 1e-6, 1.35dB at 1e-7, 1.55dB at 1e-8 (0.2dB per decade of BER)
  - At 3e-5, BER improved by 2.5 orders of magnitude
    - expected to improve with SNR
- At SNR=13.25dB, the error propagation statistics are

Burst length	Percent	Burst length	Percent
1	74.6	7	0
2	16.4	8	0
3	7.6	9	0
4	1.3	10	0
5	0.1	11	0
6	0	>11	0.1

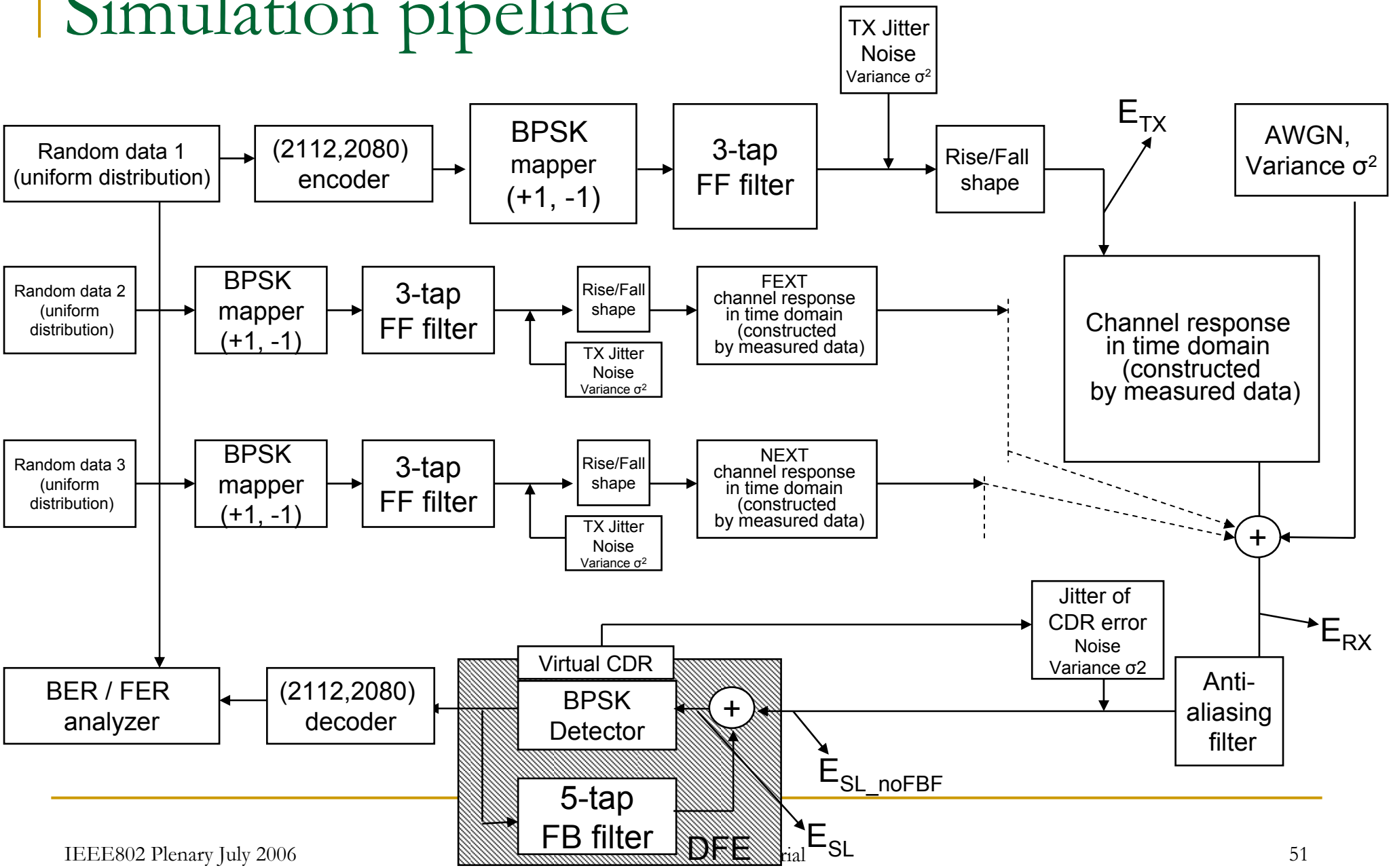
Time Domain Simulation Results – II

Courtesy of Intel

FEC Simulation Model Overview

- **Simulation Environment includes**
  - **TX FIR filter**
  - **TX Jitter and rise/fall time shaper**
  - **Cross talk (NEXT, FEXT)**
  - **Effect of CDR**
- **Plot BER curve with SNR at slicer**
- **Observed Error distribution**
  - **P(m,n)-characteristics, burst error length**

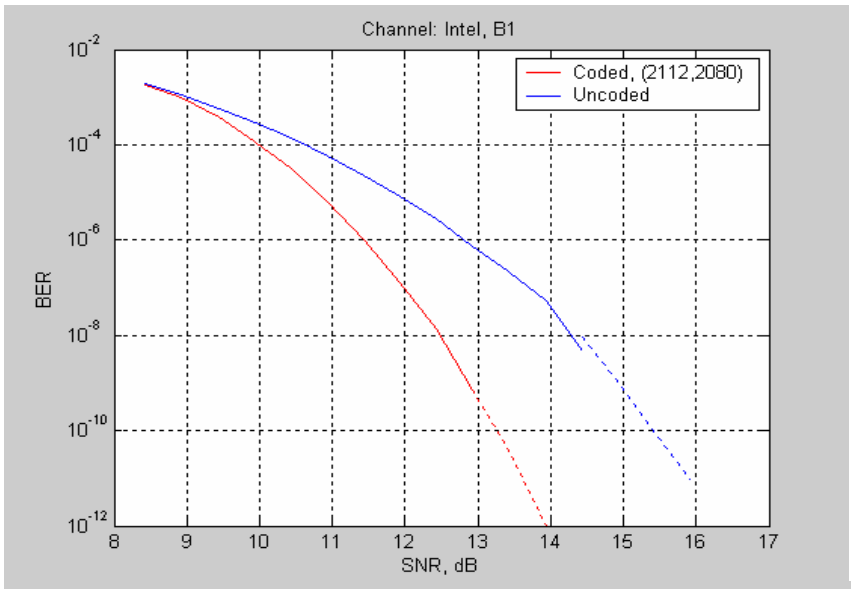
Simulation pipeline

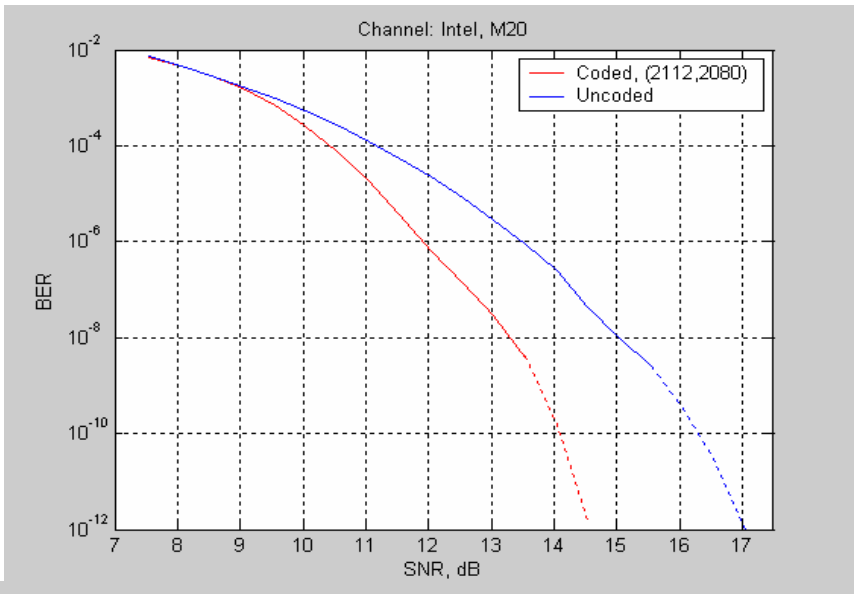


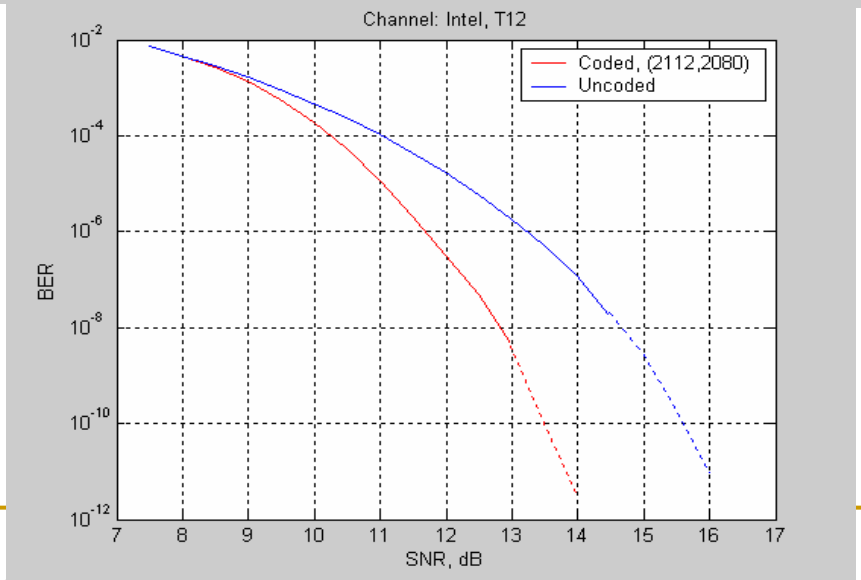
Simulation conditions

- **Time Domain simulation**
- **Simulation model includes**
  - **3 tap FIR filter: Optimal for the given channel**
  - **Rise/Fall time shaper: 24ps**
  - **Tx Jitter: 0.05UI(random, variance), 0.05UI(sine, amplitude)**
  - **Channels: Time domain response constructed from frequency domain parameters from 802.3ap channel model library**
  - **Cross talk: 1 NEXT and 1 FEXT aggressor (Intel, Molex, Tyco)**
  - **DFE: Optimal for the given channel**
  - **CDR effect: By equivalent noise with variance 0.01-0.04**
  - **Anti-Aliasing filter: 2-pole filter p1 = 0.7/T, p2 = 1.0/T**

Simulation results - Intel channels





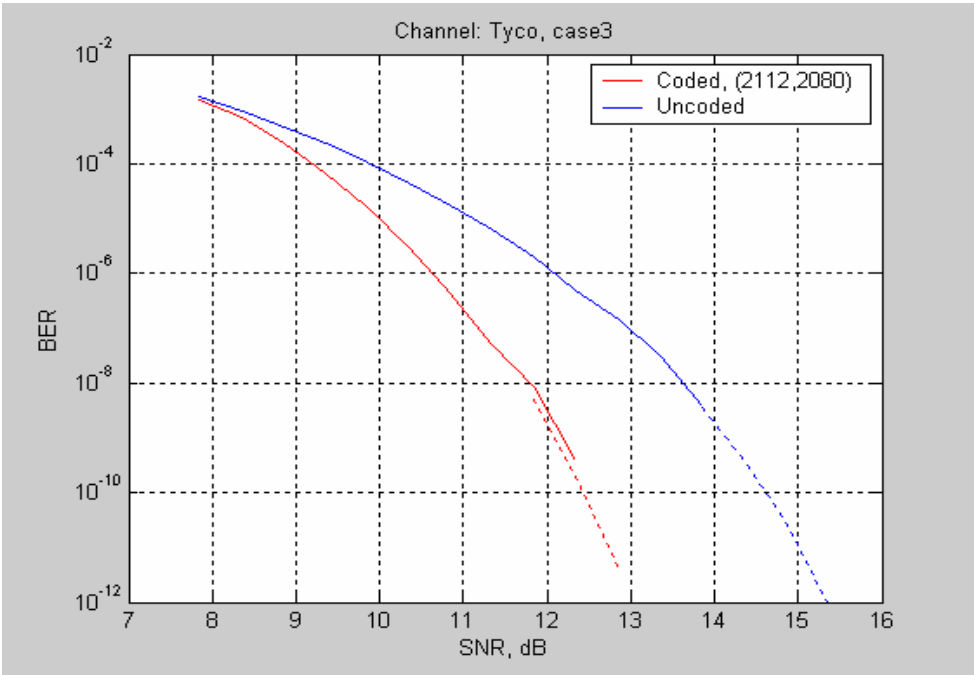


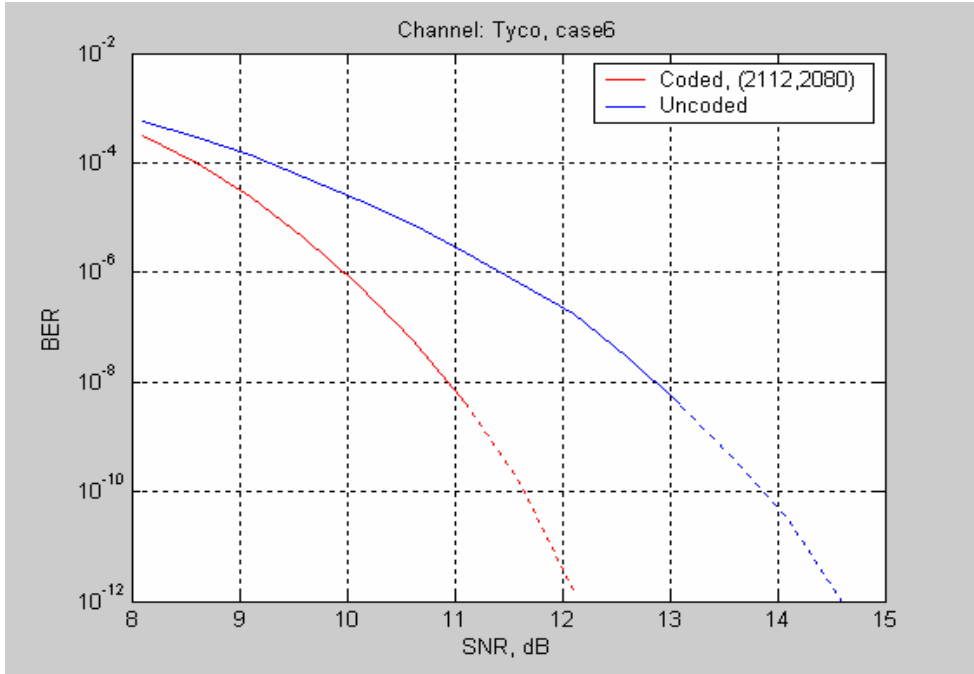
Intel Test channels (Peters_01_0605)
T12, M20, B1

SNR = SNR at slicer
Simulations to BER of 10^{-8}/10^{-9} and extrapolated to 10^{-12}

Sims show ~2dB coding gain at BER 10^{-9}

Simulation results - Tyco channels



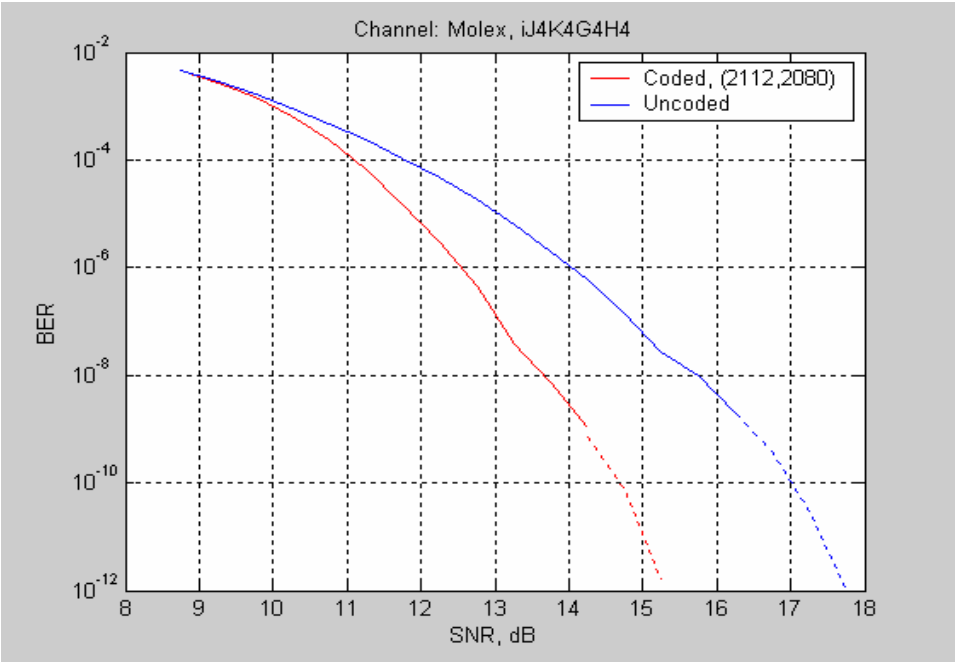


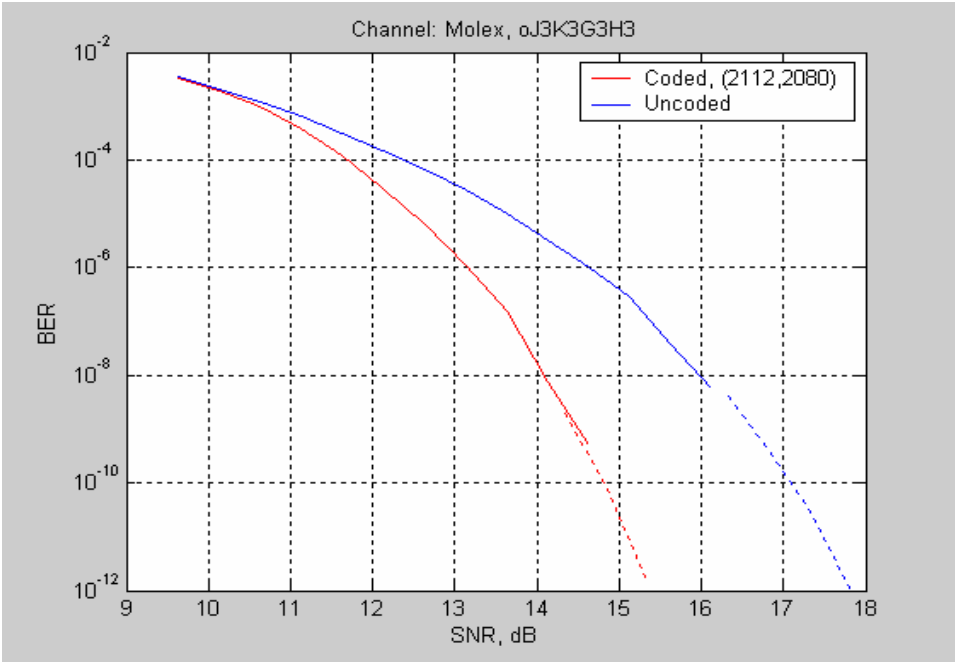
Sims show ~2dB coding gain at BER 10^{-9}

Tyco Test channels
Case 3, Case 6

SNR = SNR at slicer
Simulations to BER of 10^{-8}/10^{-9} and extrapolated to 10^{-12}

Simulation results - Molex channels





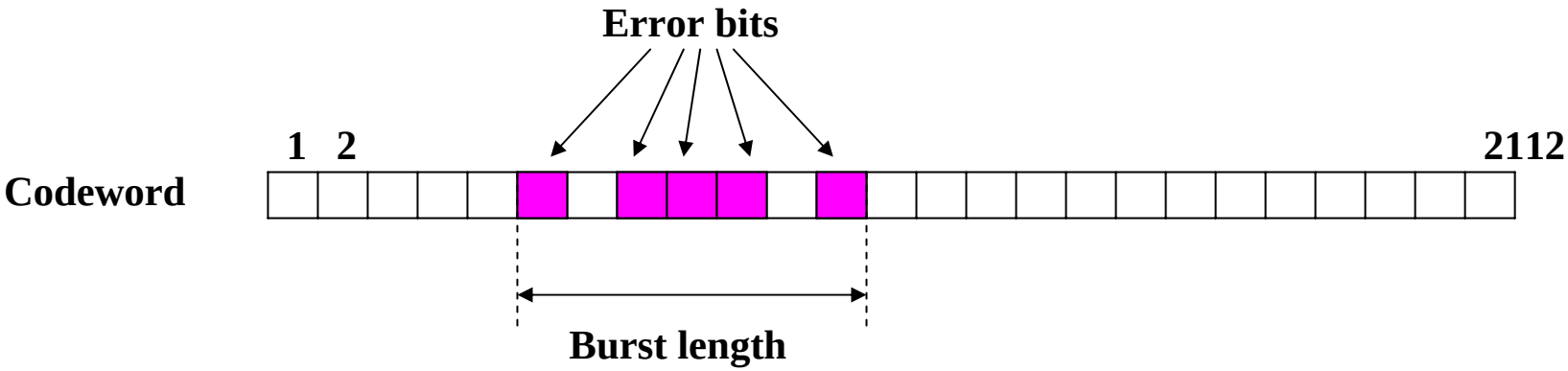
Molex test channels
Inbound J4K4G4H4
Outbound J3K3G3H3

Sims show ~2dB coding gain at BER 10^{-9}

SNR = SNR at slicer
Simulations to BER of 10^{-8}/10^{-9} and extrapolated to 10^{-12}

Error distribution observed in Simulations

- **P(m,n) characteristics (for 7 channels)**
  - **Normalized number of frames that have m errors**
- **Burst lengths (for 7 channels)**
  - **Error burst length is the distance between first and last error inside 1 codeword**



Error distribution, Intel channels

B1

m	Pr(m,2112)
0	0,983481
1	0,015378
2	0,001123
3	1,7E-05
4	1E-06
5	0
6	0
7	0
8	0
9	0
10	0
11	0

m	Pr(burst of length m)
1	0,930928022
2	0,061020643
3	0,000605364
4	0
5	0
6	0
7	0
8	6,05364E-05
9	6,05364E-05
10	0
11	0
>11	0,007324899

- **Data in all tables:**
- **P(m,n)-characteristics for frames of length 2112**
- **Burst length distribution**
  - **Normalized probability of error burst event of given length for error frames**
- **$2*10^9$ bits simulated for each channel at SNR that gives coded BER about 10^{-8}**

Error distribution, Intel channels (2)

T12

m	Pr(m,2112)
0	0,993339
1	0,001518
2	0,004781
3	0,00028
4	7,5E-05
5	6E-06
6	1E-06
7	0
8	0
9	0
10	0
11	0

m	Pr(burst of len m)
1	0,22789371
2	0,710253716
3	0,039333433
4	0,013511485
5	0,004954211
6	0,00060051
7	0,000150128
8	0
9	0
10	0,000150128
11	0
>11	0,00315268

M20

m	Pr(m,2112)
0	0,996229
1	0,001594
2	0,002039
3	0,000108
4	2,6E-05
5	4E-06
6	0
7	0
8	0
9	0
10	0
11	0

m	Pr(burst of len m)
1	0,422699549
2	0,538053567
3	0,029965526
4	0,007159905
5	0,001325908
6	0
7	0
8	0
9	0
10	0
11	0
>11	0,000795545

Error distribution, Tyco channels

Case3

m	Pr(m,2112)
0	0,99088
1	0,005896
2	0,002988
3	0,000203
4	3,1E-05
5	2E-06
6	0
7	0
8	0
9	0
10	0
11	0

m	Pr(burst of len m)
1	0,646491228
2	0,325
3	0,021162281
4	0,002850877
5	0,000219298
6	0
7	0
8	0
9	0
10	0,000109649
11	0
>11	0,004166667

Case6

m	Pr(m,2112)
0	0,990547
1	0,004483
2	0,004148
3	0,000604
4	0,000163
5	4E-05
6	1,4E-05
7	1E-06
8	0
9	0
10	0
11	0

m	Pr(burst of len m)
1	0,474240982
2	0,436898339
3	0,062308262
4	0,015867978
5	0,004125674
6	0,001586798
7	0,000105787
8	0
9	0
10	0,000105787
11	0
>11	0,004760394

Error distribution, Molex channels

Inbound J4K4G4H4

m	Pr(m,2112)	m	Pr(burst of len m)
0	0,991277	1	0,532385647
1	0,004644	2	0,428866216
2	0,003764	3	0,026940273
3	0,000242	4	0,006534449
4	6,7E-05	5	0,000573197
5	6E-06	6	0,000114639
6	0	7	0
7	0	8	0
8	0	9	0
9	0	10	0,000114639
10	0	11	0
11	0	>11	0,004470939

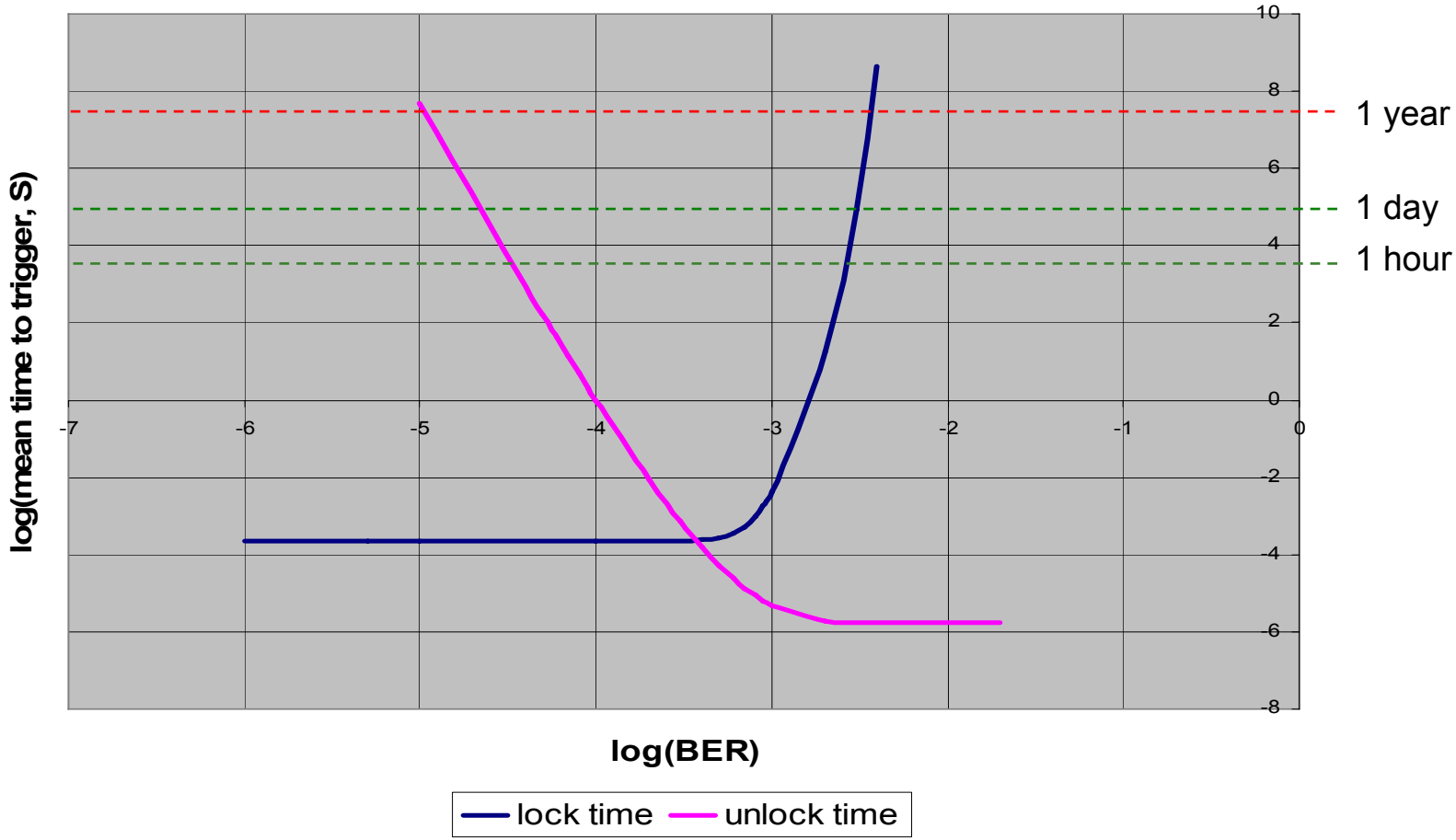
Outbound J3K3G3H3

m	Pr(m,2112)	m	Pr(burst of len m)
0	0,985625	1	0,601321739
1	0,008644	2	0,367095652
2	0,005329	3	0,02066087
3	0,000328	4	0,003756522
4	6,7E-05	5	0,000347826
5	6E-06	6	0,00013913
6	1E-06	7	0
7	0	8	0
8	0	9	0
9	0	10	0,00013913
10	0	11	0
11	0	>11	0,00653913

Synchronization and Unlock Time

- Synchronization time
  - Assume random start location
  - Average of 2112/2 FEC blocks before correct bit synchronization found
  - Need 4 consecutive blocks with “valid parity” to set fec_block_lock
  - At low BER, **synchronization time ~= 0.22 milliseconds**
    - 2112*(2112/2 + 4) bit times
- Unlock Time
  - Assume random error events
  - Need 8 consecutive “invalid parity” events
  - At very high BER, **unlock time ~= 1.6 microseconds**
    - 2112*8 bit times
- Both times are plotted versus BER in next slide

Synchronization and Unlock Time



Ease of Implementation

Andre Szczepanek

Texas Instruments

Agenda

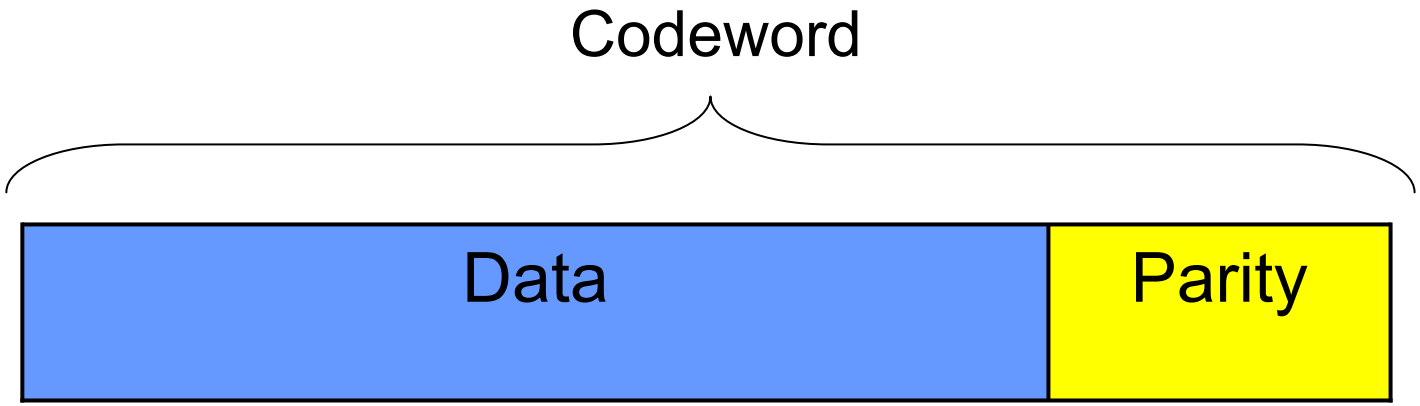
- Show that implementing 10GBASE-KR FEC is based on the same principles as generating/checking 802 CRCs, and is easily achieved using an extension of the existing techniques.

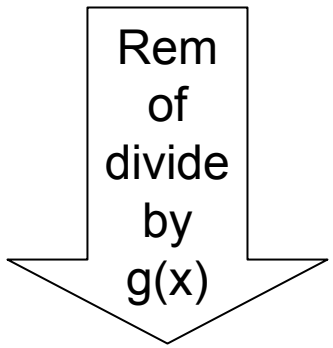
Cyclic block codes

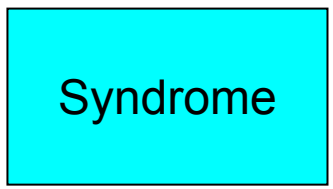
- This is a class of codes that includes the 802 CRCs
  - A Cyclic block code is defined by a generator polynomial g(x)
- Encoding consists of adding a set of parity bits onto the data to create a "codeword"
    - More commonly called a frame or a block
  - The parity is the remainder from the polynomial division of the data bits by g(x)
    - Easily implemented using an LFSR (same process as CRC generation)

- Error detection/correction consists of calculating the "syndrome" of the received codeword
    - The syndrome is the difference between locally generated and received parity.
    - For cyclic block codes the syndrome is also the remainder from the polynomial division of the received codeword by g(x)
    - Can also be implemented with an LSFR
  - If the syndrome is zero the codeword is correct
  - If the syndrome is non-zero it can be used to determine the most likely error

Codewords, Parity, & Syndromes





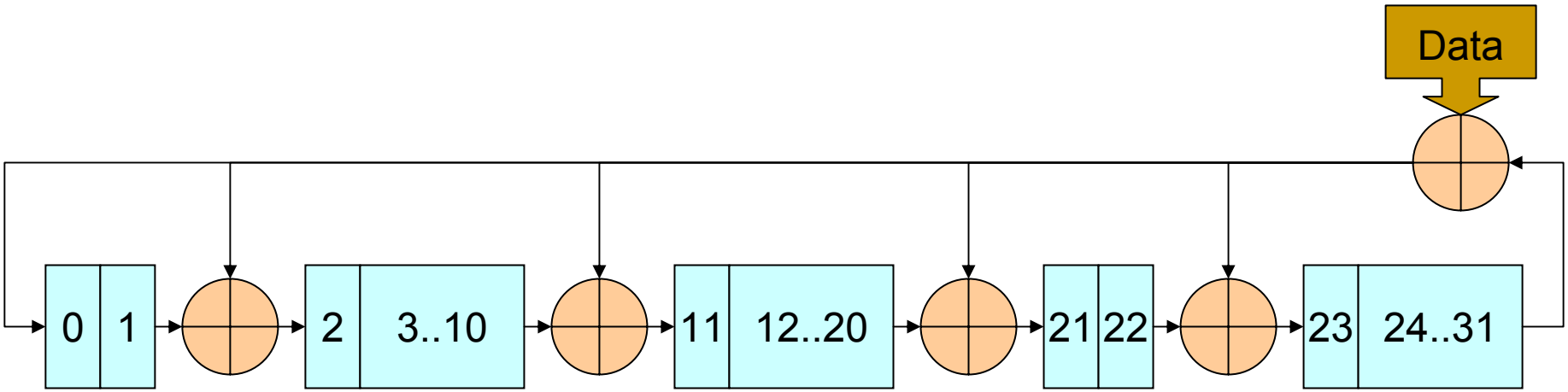


The Syndrome is also equal to

Local Parity XOR received Parity

Syndrome = 0 if codeword is good

FEC Parity/Syndrome generation



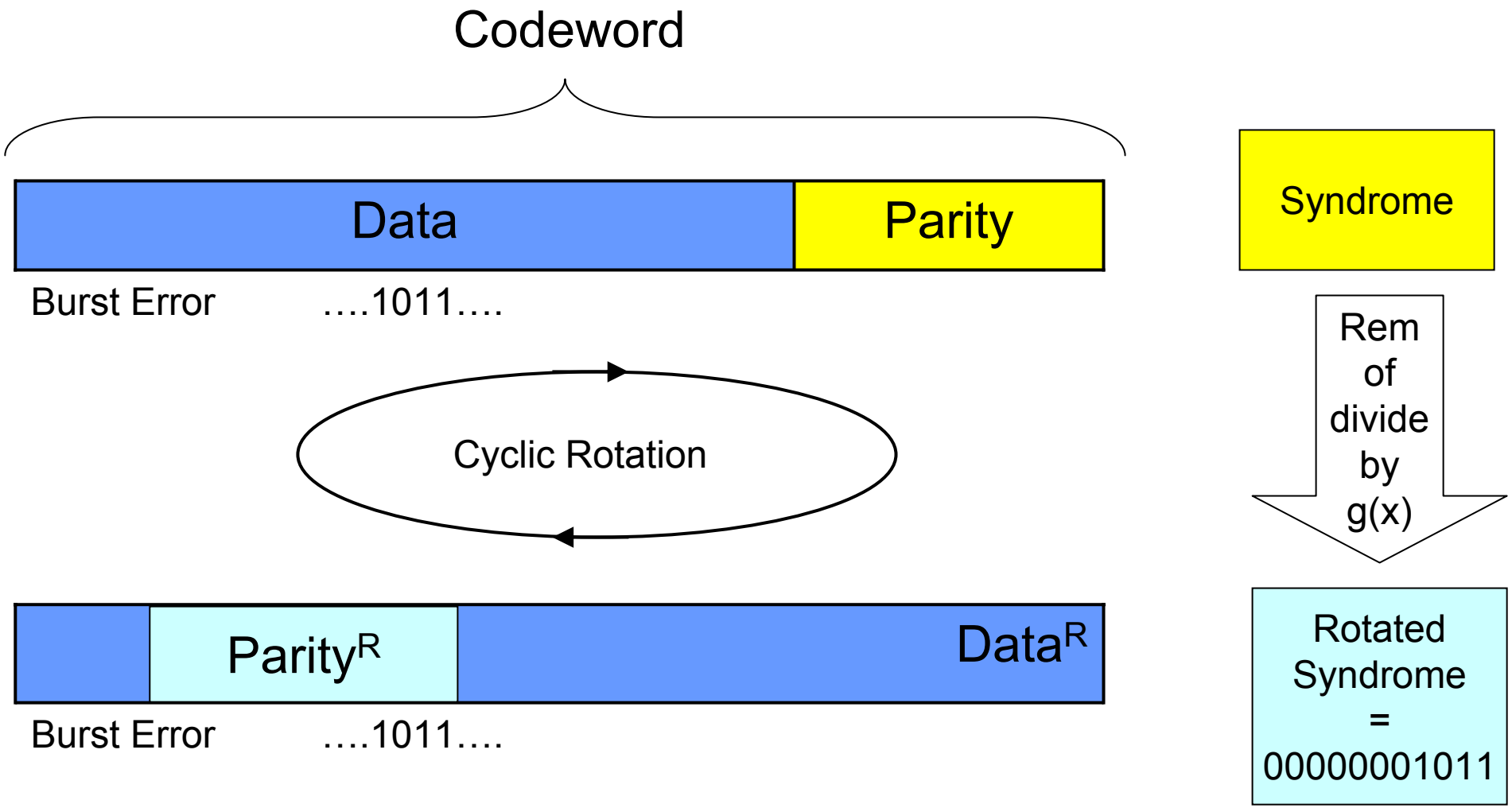
$1 + X^2 + X^{11} + X^{21} + X^{23} + X^{32}$

$= g(x)$

Cyclic block codes have cyclic symmetry

- A cyclic shift of a codeword (data+parity) is also a codeword
  - A good frame will have a zero syndrome
- Division of the Syndrome by g(x) produces the syndrome of the codeword cyclically shifted right by one place
  - It is straightforward to generate combinatorial logic to “rotate” a syndrome by any amount
    - Same task as creating a parallel CRC generator
- Syndrome rotation can be used to search for likely error locations
  - The “Meggitt” decoder serially searches a codeword for the syndromes of known error patterns

Burst Error trapping



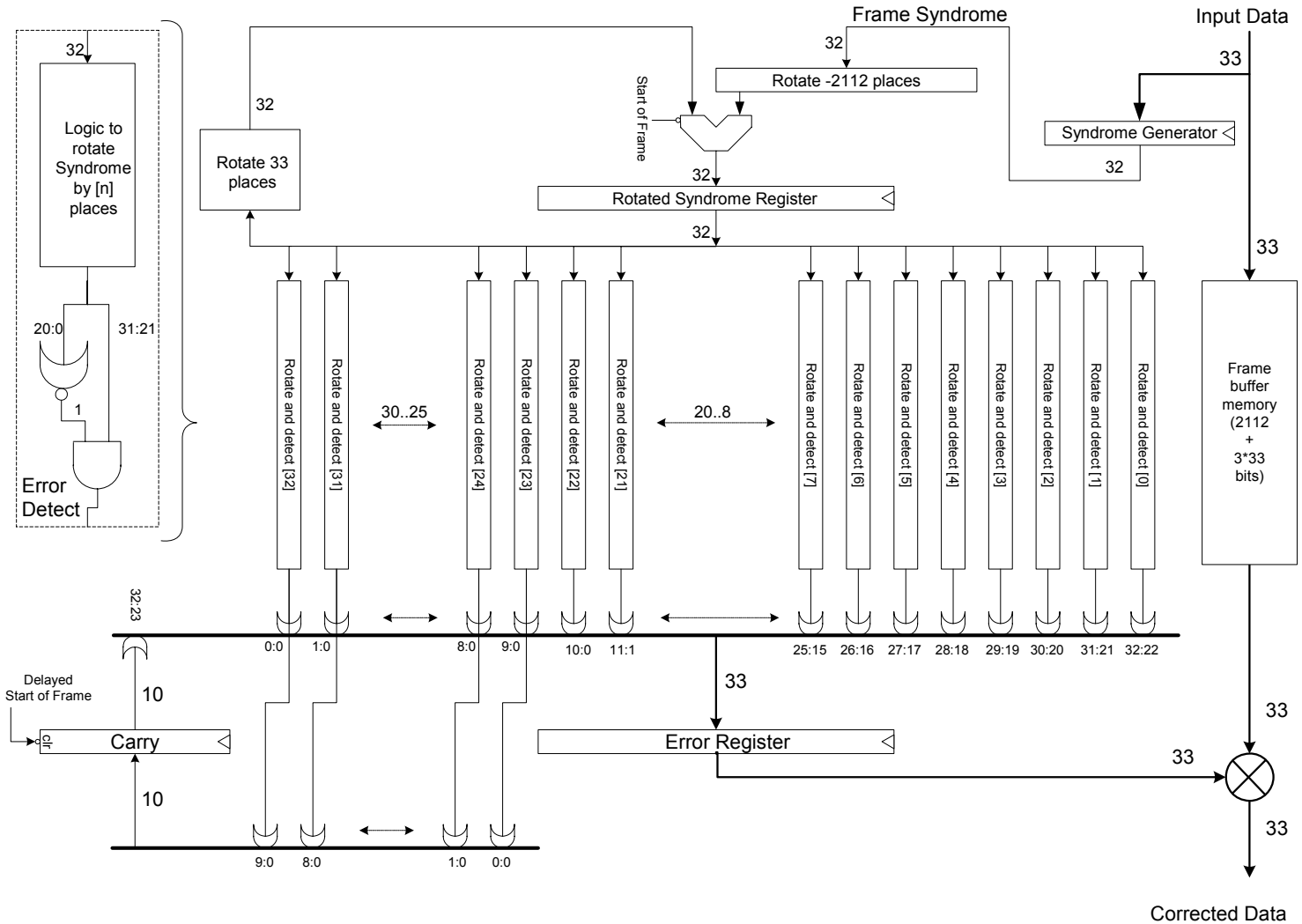
Single burst errors have easily recognisable syndromes

- Consider a correctable burst error in the last parity bits
  - Within the burst correcting capability “l” of the code
    - l = 11 for the 10BASE-KR FEC
  - The syndrome of this error will contain the error pattern.
    - The syndrome is the XOR of the sent and received parity
  - So, of the “s” syndrome bits, the first s-l bits will be zero, the last l bits will be the burst error pattern
- Remember that due to cyclic symmetry every data bit is a parity bit of some rotated codeword
  - We can find the position of a burst error of l or less bits by searching for a syndrome rotation with the first s-l bits equal to zero
    - The syndrome can then be XOR’ed with the data to correct it
- Error correction with the 10GBASE-KR FEC is simply a process of searching for a 32bit syndrome with the LS 21 bits equal to zero.

Implementing the 10GBASE-KR FEC

- Parity generation uses an LFSR - like CRC
  - Must be parallelized to meet the data-rate
    - 32 bit parallel at least (~320Mhz)
- Error detection/correct is a multi-stage process
  - Syndrome generation (similar to parity generation)
    - Adds 1 frame time of latency (must receive whole frame)
  - Syndrome alignment
    - Rotate syndrome to account for shortened code
  - Parallel error trapping and correction
    - Syndromes can be generated and checked in parallel for a whole data word (e.g. 33 bits)
    - Trapped burst patterns can be XOR'ed with data to correct them
  - Error reporting
    - 10GBASE-KR provides the option of marking/corrupting 64b66 codewords to indicate uncorrectable frames
    - Adds 1 frame time of latency (must attempt to correct whole frame)

Parallel error trapping and correction



10Gbase-KR FEC complexity

- Area for a 33bit wide datapath @312.5Mhz
  - Transmit framer/coder : 3K gates
  - Rx Sync & Syndrome generation : 4K gates
  - Rx Parallel Error trapping and correction : 7K gates
  - Dual port RAM
    - 33 x (64 +3) without error marking
    - 33 x (64+ 3 + 64) with error marking
  - Rx deframer : 1K gates
- Total Rx/Tx = ~15K gates + RAM
- Latency
  - Without error marking (minimum latency)
    - 33 x (64 + 3) bits = 2211
  - With error marking (1 extra frame time)
    - 33 x (64 + 3 + 64) bits = 4323

Summary

- The cyclic symmetry of the 10GBASE-KR FEC allows ≤11 bit burst errors to be easily detected and corrected
- The syndrome generation and rotation uses polynomial arithmetic implemented in LFSRs
  - Just like the Ethernet CRC
- These operations can easily be parallelized using the same principles used to create parallel CRC generator/checkers

Wrap Up

Andre Szczepanek

Texas Instruments

Agenda

- Conclusion
  - FEC selection
  - Benefits of using the 10GBASE-KR FEC

FEC selection

- The DFE equalization used for 10GBASE-KR causes significant error propagation, as we have demonstrated
  - Significant levels of burst errors, but still <1%
    - Insignificant effect on overall BER
    - Significant effect on MTTFPA
- This drove selection of a Single Burst Error Correcting code
  - They Outperform RS codes in this context
    - Have higher coding gain at the BERs of interest
      - Marginal/Adequate/Good BER
    - Is simpler to implement

Benefits of using 10GBASE-KR FEC

- Applicable to any 10Gbase-R based PHY with similar error characteristics
  - Significant levels of burst errors
- Substantially Improves the BER of just compliant links
  - $1e^{-12}$ BER → $1e^{-18}$ BER
  - 1 error every 100 seconds → 1 error every 3 years
- Improves MTTFPA by many orders of magnitude
- Improves overall system reliability by significantly lowering BER
- Simulations show 2 to 2.5db of coding gain for 802.3ap channels
  - Can make marginal channels operate at better than $1e^{-12}$ BER
- Low complexity FEC implementation
  - Minimal extra gates
  - Minimal additional Latency

References

[1] Richard E. Blahut, *Algebraic Codes for Data Transmission*, Cambridge University Press, 2003

[2] S. Lin and D. Costello, *Error Control Coding : Fundamentals and Applications,* Prentice Hall, Englewood Cliffs, New Jersey 1983

802.3ap TF contributions

- **ganga_02_0905**
- **ganga_02_1105**
- **liu_01_1105**
- **szczepanek_01_0305**
- **valliappan_01_1105**

Backup

Shortening cyclic codes

- Cyclic block codes are usually shortened to make practical block sizes
  - This is equivalent to stuffing the missing bits with virtual zero bits
    - Has no effect on parity/syndrome generation
    - Does affect the relative cyclic rotation of the syndrome wrt the codeword
      - Need to rotate the syndrome to match the codeword start, before beginning error correction
        - 10GBASE-KR is a (2112,2080) shortened form of a (42987,42955) code
      - Need to rotate by +40875 or -2112
- Because of cyclic symmetry un-shortened codes need to correct "wrap-around" errors
  - A burst can exist over the last parity and first data bits
  - Codes shortened by more than l, can't have "wrap-arounds"
- The 10GBASE-KR code is a shortened code
  - No wrap-arounds to worry about !
  - Will need to rotate syndrome before beginning error correction

33bit Parallel parity equations (1)

```
next_c33[ 0] = this_c[  8] ^ this_c[ 10] ^ this_c[ 17] ^ this_c[ 20] ^ this_c[ 21] ^ this_c[ 26] ^
               this_c[ 28] ^ this_c[ 29] ^ this_c[ 30] ^ this_c[ 31] ^ this_d[ 32] ^ this_d[ 23] ^
               this_d[ 21] ^ this_d[ 14] ^ this_d[ 11] ^ this_d[ 10] ^ this_d[  5] ^ this_d[  3] ^
               this_d[  2] ^ this_d[  1] ^ this_d[  0];
next_c33[ 1] = this_c[  0] ^ this_c[  9] ^ this_c[ 11] ^ this_c[ 18] ^ this_c[ 21] ^ this_c[ 22] ^
               this_c[ 27] ^ this_c[ 29] ^ this_c[ 30] ^ this_c[ 31] ^ this_d[ 31] ^ this_d[ 22] ^
               this_d[ 20] ^ this_d[ 13] ^ this_d[ 10] ^ this_d[  9] ^ this_d[  4] ^ this_d[  2] ^
               this_d[  1] ^ this_d[  0];
next_c33[ 2] = this_c[  1] ^ this_c[  8] ^ this_c[ 12] ^ this_c[ 17] ^ this_c[ 19] ^ this_c[ 20] ^
               this_c[ 21] ^ this_c[ 22] ^ this_c[ 23] ^ this_c[ 26] ^ this_c[ 29] ^ this_d[ 32] ^
               this_d[ 30] ^ this_d[ 23] ^ this_d[ 19] ^ this_d[ 14] ^ this_d[ 12] ^ this_d[ 11] ^
               this_d[ 10] ^ this_d[  9] ^ this_d[  8] ^ this_d[  5] ^ this_d[  2];
next_c33[ 3] = this_c[  0] ^ this_c[  2] ^ this_c[  9] ^ this_c[ 13] ^ this_c[ 18] ^ this_c[ 20] ^
               this_c[ 21] ^ this_c[ 22] ^ this_c[ 23] ^ this_c[ 24] ^ this_c[ 27] ^ this_c[ 30] ^
               this_d[ 31] ^ this_d[ 29] ^ this_d[ 22] ^ this_d[ 18] ^ this_d[ 13] ^ this_d[ 11] ^
               this_d[ 10] ^ this_d[  9] ^ this_d[  8] ^ this_d[  7] ^ this_d[  4] ^ this_d[  1];
next_c33[ 4] = this_c[  1] ^ this_c[  3] ^ this_c[ 10] ^ this_c[ 14] ^ this_c[ 19] ^ this_c[ 21] ^
               this_c[ 22] ^ this_c[ 23] ^ this_c[ 24] ^ this_c[ 25] ^ this_c[ 28] ^ this_c[ 31] ^
               this_d[ 30] ^ this_d[ 28] ^ this_d[ 21] ^ this_d[ 17] ^ this_d[ 12] ^ this_d[ 10] ^
               this_d[  9] ^ this_d[  8] ^ this_d[  7] ^ this_d[  6] ^ this_d[  3] ^ this_d[  0];
next_c33[ 5] = this_c[  2] ^ this_c[  4] ^ this_c[ 11] ^ this_c[ 15] ^ this_c[ 20] ^ this_c[ 22] ^
               this_c[ 23] ^ this_c[ 24] ^ this_c[ 25] ^ this_c[ 26] ^ this_c[ 29] ^ this_d[ 29] ^
               this_d[ 27] ^ this_d[ 20] ^ this_d[ 16] ^ this_d[ 11] ^ this_d[  9] ^ this_d[  8] ^
               this_d[  7] ^ this_d[  6] ^ this_d[  5] ^ this_d[  2];
next_c33[ 6] = this_c[  3] ^ this_c[  5] ^ this_c[ 12] ^ this_c[ 16] ^ this_c[ 21] ^ this_c[ 23] ^
               this_c[ 24] ^ this_c[ 25] ^ this_c[ 26] ^ this_c[ 27] ^ this_c[ 30] ^ this_d[ 28] ^
               this_d[ 26] ^ this_d[ 19] ^ this_d[ 15] ^ this_d[ 10] ^ this_d[  8] ^ this_d[  7] ^
               this_d[  6] ^ this_d[  5] ^ this_d[  4] ^ this_d[  1];
next_c33[ 7] = this_c[  4] ^ this_c[  6] ^ this_c[ 13] ^ this_c[ 17] ^ this_c[ 22] ^ this_c[ 24] ^
               this_c[ 25] ^ this_c[ 26] ^ this_c[ 27] ^ this_c[ 28] ^ this_c[ 31] ^ this_d[ 27] ^
               this_d[ 25] ^ this_d[ 18] ^ this_d[ 14] ^ this_d[  9] ^ this_d[  7] ^ this_d[  6] ^
               this_d[  5] ^ this_d[  4] ^ this_d[  3] ^ this_d[  0];
next_c33[ 8] = this_c[  5] ^ this_c[  7] ^ this_c[ 14] ^ this_c[ 18] ^ this_c[ 23] ^ this_c[ 25] ^
               this_c[ 26] ^ this_c[ 27] ^ this_c[ 28] ^ this_c[ 29] ^ this_d[ 26] ^ this_d[ 24] ^
               this_d[ 17] ^ this_d[ 13] ^ this_d[  8] ^ this_d[  6] ^ this_d[  5] ^ this_d[  4] ^
               this_d[  3] ^ this_d[  2];
next_c33[ 9] = this_c[  6] ^ this_c[  8] ^ this_c[ 15] ^ this_c[ 19] ^ this_c[ 24] ^ this_c[ 26] ^
               this_c[ 27] ^ this_c[ 28] ^ this_c[ 29] ^ this_c[ 30] ^ this_d[ 25] ^ this_d[ 23] ^
               this_d[ 16] ^ this_d[ 12] ^ this_d[  7] ^ this_d[  5] ^ this_d[  4] ^ this_d[  3] ^
               this_d[  2] ^ this_d[  1];
```

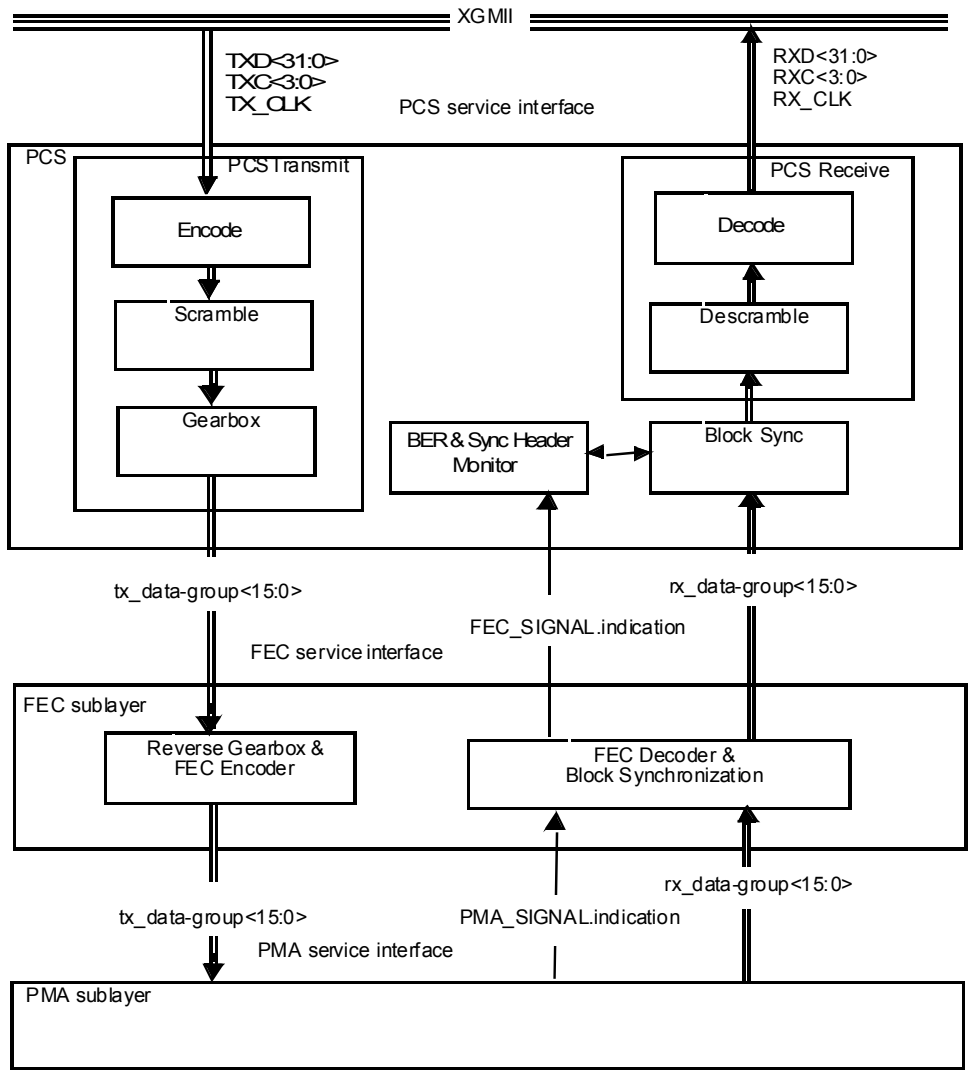
33bit Parallel parity equations (2)

```
next_c33[10] = this_c[  7] ^ this_c[  9] ^ this_c[ 16] ^ this_c[ 20] ^ this_c[ 25] ^ this_c[ 27] ^
               this_c[ 28] ^ this_c[ 29] ^ this_c[ 30] ^ this_c[ 31] ^ this_d[ 24] ^ this_d[ 22] ^
               this_d[ 15] ^ this_d[ 11] ^ this_d[  6] ^ this_d[  4] ^ this_d[  3] ^ this_d[  2] ^
               this_d[  1] ^ this_d[  0];
next_c33[11] = this_c[ 20] ^ this_d[ 32] ^ this_d[ 11];
next_c33[12] = this_c[  0] ^ this_c[ 21] ^ this_d[ 31] ^ this_d[ 10];
next_c33[13] = this_c[  1] ^ this_c[ 22] ^ this_d[ 30] ^ this_d[  9];
next_c33[14] = this_c[  2] ^ this_c[ 23] ^ this_d[ 29] ^ this_d[  8];
next_c33[15] = this_c[  3] ^ this_c[ 24] ^ this_d[ 28] ^ this_d[  7];
next_c33[16] = this_c[  4] ^ this_c[ 25] ^ this_d[ 27] ^ this_d[  6];
next_c33[17] = this_c[  5] ^ this_c[ 26] ^ this_d[ 26] ^ this_d[  5];
next_c33[18] = this_c[  6] ^ this_c[ 27] ^ this_d[ 25] ^ this_d[  4];
next_c33[19] = this_c[  7] ^ this_c[ 28] ^ this_d[ 24] ^ this_d[  3];
next_c33[20] = this_c[  8] ^ this_c[ 29] ^ this_d[ 23] ^ this_d[  2];
next_c33[21] = this_c[  8] ^ this_c[  9] ^ this_c[ 10] ^ this_c[ 17] ^ this_c[ 20] ^ this_c[ 21] ^
               this_c[ 26] ^ this_c[ 28] ^ this_c[ 29] ^ this_c[ 31] ^ this_d[ 32] ^ this_d[ 23] ^
               this_d[ 22] ^ this_d[ 21] ^ this_d[ 14] ^ this_d[ 11] ^ this_d[ 10] ^ this_d[  5] ^
               this_d[  3] ^ this_d[  2] ^ this_d[  0];
next_c33[22] = this_c[  0] ^ this_c[  9] ^ this_c[ 10] ^ this_c[ 11] ^ this_c[ 18] ^ this_c[ 21] ^
               this_c[ 22] ^ this_c[ 27] ^ this_c[ 29] ^ this_c[ 30] ^ this_d[ 31] ^ this_d[ 22] ^
               this_d[ 21] ^ this_d[ 20] ^ this_d[ 13] ^ this_d[ 10] ^ this_d[  9] ^ this_d[  4] ^
               this_d[  2] ^ this_d[  1];
next_c33[23] = this_c[  1] ^ this_c[  8] ^ this_c[ 11] ^ this_c[ 12] ^ this_c[ 17] ^ this_c[ 19] ^
               this_c[ 20] ^ this_c[ 21] ^ this_c[ 22] ^ this_c[ 23] ^ this_c[ 26] ^ this_c[ 29] ^
               this_d[ 32] ^ this_d[ 30] ^ this_d[ 23] ^ this_d[ 20] ^ this_d[ 19] ^ this_d[ 14] ^
               this_d[ 12] ^ this_d[ 11] ^ this_d[ 10] ^ this_d[  9] ^ this_d[  8] ^ this_d[  5] ^
               this_d[  2];
next_c33[24] = this_c[  0] ^ this_c[  2] ^ this_c[  9] ^ this_c[ 12] ^ this_c[ 13] ^ this_c[ 18] ^
               this_c[ 20] ^ this_c[ 21] ^ this_c[ 22] ^ this_c[ 23] ^ this_c[ 24] ^ this_c[ 27] ^
               this_c[ 30] ^ this_d[ 31] ^ this_d[ 29] ^ this_d[ 22] ^ this_d[ 19] ^ this_d[ 18] ^
               this_d[ 13] ^ this_d[ 11] ^ this_d[ 10] ^ this_d[  9] ^ this_d[  8] ^ this_d[  7] ^
               this_d[  4] ^ this_d[  1];
next_c33[25] = this_c[  1] ^ this_c[  3] ^ this_c[ 10] ^ this_c[ 13] ^ this_c[ 14] ^ this_c[ 19] ^
               this_c[ 21] ^ this_c[ 22] ^ this_c[ 23] ^ this_c[ 24] ^ this_c[ 25] ^ this_c[ 28] ^
               this_c[ 31] ^ this_d[ 30] ^ this_d[ 28] ^ this_d[ 21] ^ this_d[ 18] ^ this_d[ 17] ^
               this_d[ 12] ^ this_d[ 10] ^ this_d[  9] ^ this_d[  8] ^ this_d[  7] ^ this_d[  6] ^
               this_d[  3] ^ this_d[  0];
next_c33[26] = this_c[  2] ^ this_c[  4] ^ this_c[ 11] ^ this_c[ 14] ^ this_c[ 15] ^ this_c[ 20] ^
               this_c[ 22] ^ this_c[ 23] ^ this_c[ 24] ^ this_c[ 25] ^ this_c[ 26] ^ this_c[ 29] ^
               this_d[ 29] ^ this_d[ 27] ^ this_d[ 20] ^ this_d[ 17] ^ this_d[ 16] ^ this_d[ 11] ^
               this_d[  9] ^ this_d[  8] ^ this_d[  7] ^ this_d[  6] ^ this_d[  5] ^ this_d[  2];
```

33bit Parallel parity equations (3)

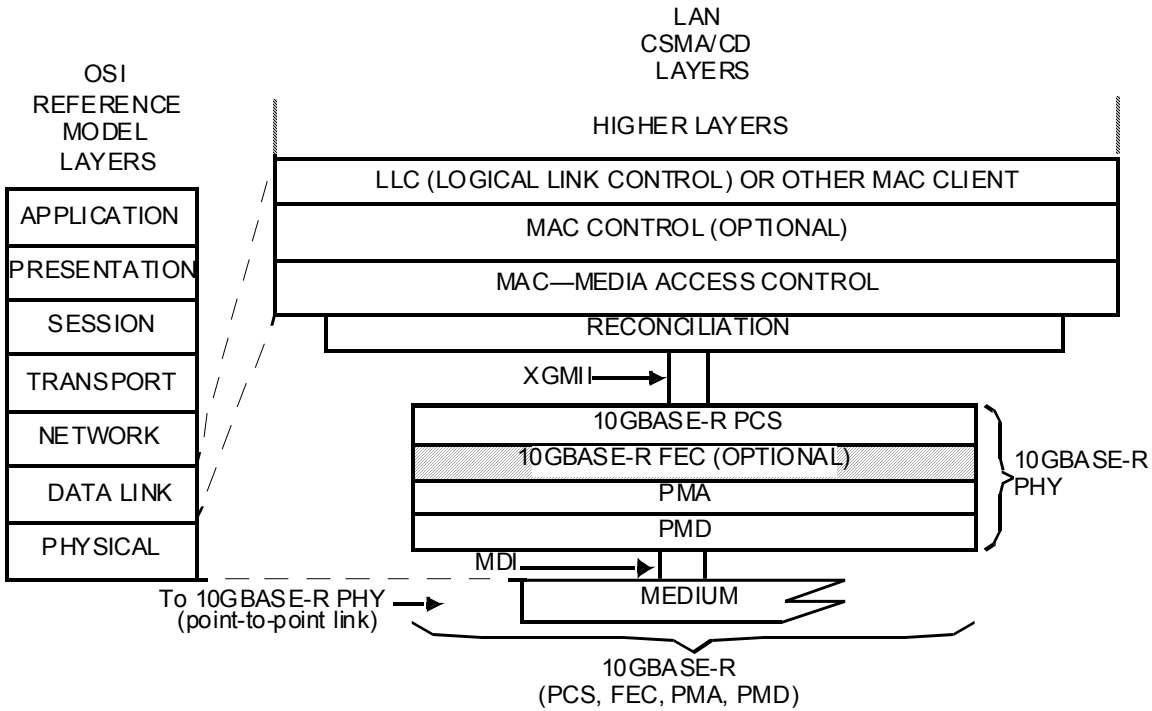
```
next_c33[27] = this_c[  3] ^ this_c[  5] ^ this_c[ 12] ^ this_c[ 15] ^ this_c[ 16] ^ this_c[ 21] ^
               this_c[ 23] ^ this_c[ 24] ^ this_c[ 25] ^ this_c[ 26] ^ this_c[ 27] ^ this_c[ 30] ^
               this_d[ 28] ^ this_d[ 26] ^ this_d[ 19] ^ this_d[ 16] ^ this_d[ 15] ^ this_d[ 10] ^
               this_d[  8] ^ this_d[  7] ^ this_d[  6] ^ this_d[  5] ^ this_d[  4] ^ this_d[  1];
next_c33[28] = this_c[  4] ^ this_c[  6] ^ this_c[ 13] ^ this_c[ 16] ^ this_c[ 17] ^ this_c[ 22] ^
               this_c[ 24] ^ this_c[ 25] ^ this_c[ 26] ^ this_c[ 27] ^ this_c[ 28] ^ this_c[ 31] ^
               this_d[ 27] ^ this_d[ 25] ^ this_d[ 18] ^ this_d[ 15] ^ this_d[ 14] ^ this_d[  9] ^
               this_d[  7] ^ this_d[  6] ^ this_d[  5] ^ this_d[  4] ^ this_d[  3] ^ this_d[  0];
next_c33[29] = this_c[  5] ^ this_c[  7] ^ this_c[ 14] ^ this_c[ 17] ^ this_c[ 18] ^ this_c[ 23] ^
               this_c[ 25] ^ this_c[ 26] ^ this_c[ 27] ^ this_c[ 28] ^ this_c[ 29] ^ this_d[ 26] ^
               this_d[ 24] ^ this_d[ 17] ^ this_d[ 14] ^ this_d[ 13] ^ this_d[  8] ^ this_d[  6] ^
               this_d[  5] ^ this_d[  4] ^ this_d[  3] ^ this_d[  2];
next_c33[30] = this_c[  6] ^ this_c[  8] ^ this_c[ 15] ^ this_c[ 18] ^ this_c[ 19] ^ this_c[ 24] ^
               this_c[ 26] ^ this_c[ 27] ^ this_c[ 28] ^ this_c[ 29] ^ this_c[ 30] ^ this_d[ 25] ^
               this_d[ 23] ^ this_d[ 16] ^ this_d[ 13] ^ this_d[ 12] ^ this_d[  7] ^ this_d[  5] ^
               this_d[  4] ^ this_d[  3] ^ this_d[  2] ^ this_d[  1];
next_c33[31] = this_c[  7] ^ this_c[  9] ^ this_c[ 16] ^ this_c[ 19] ^ this_c[ 20] ^ this_c[ 25] ^
               this_c[ 27] ^ this_c[ 28] ^ this_c[ 29] ^ this_c[ 30] ^ this_c[ 31] ^ this_d[ 24] ^
               this_d[ 22] ^ this_d[ 15] ^ this_d[ 12] ^ this_d[ 11] ^ this_d[  6] ^ this_d[  4] ^
               this_d[  3] ^ this_d[  2] ^ this_d[  1] ^ this_d[  0];
```

FEC layering



FEC functional block diagram

FEC layering



MDI=MEDIUM DEPENDENT INTERFACE
PCS=PHYSICAL CODING SUBLAYER
PHY=PHYSICAL LAYER DEVICE
PMA=PHYSICAL MEDIUM ATTACHMENT
PMD=PHYSICAL MEDIUM DEPENDENT
FEC=FORWARD ERROR CORRECTION
XGMII=10 GIGABIT MEDIA INDEPENDENT INTERFACE

10GBASE-R FEC relationship to ISO/IEC Open Systems Interconnection (OSI) reference model and the IEEE 802.3 CSMA/CD LAN model

Auto-Negotiation

- Auto-negotiation advertises FEC capabilities in PHY
  - Clause 73 AN is used by 10GBASE-KR
- FEC is disabled by default
- FEC is enabled if if both link partners advertise they have the ability, and one requests it enabled