64b/66b coding update

Rick Walker, Birdy Amrutur, Tom Knotts

Agilent Laboratories, Palo Alto, CA

rick_walker@agilent.com

Richard Dugan

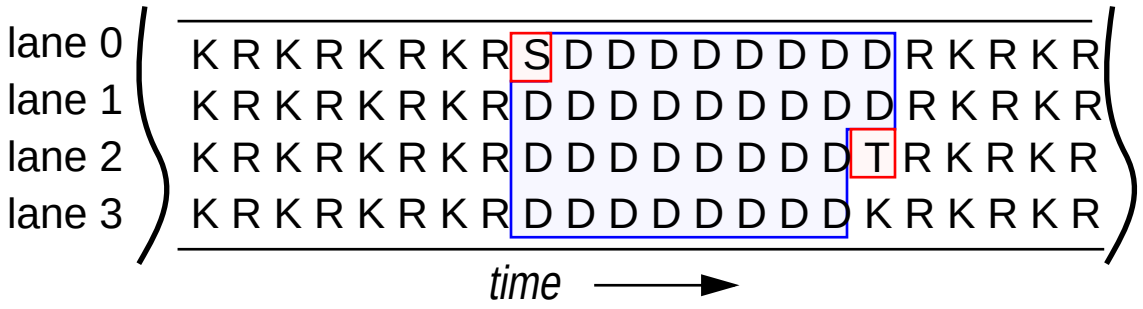
Agilent Technologies, Integrated Circuits Business Division, San Jose, CA

richard_dugan@agilent.com

Topics

- Code update
- Mean Time to False Packet Acceptance
- Coder Block Diagram and Gate Count
- Scrambler design
- Summary

Building frames with XAUI (HARI) mapping



S,T = SOP, EOP
K,R = control words (Z)
D = Data octets

pure data

D D
D D
D D
D D

D

pure control

Z Z
Z Z
Z Z
Z Z

Z

two possible packet startings

S D
D D
D D
D D

Z S
Z D
Z D
Z D

S

eight possible packet endings

T Z
Z Z
Z Z
Z Z

D Z
T Z
Z Z
Z Z

D Z
D Z
T Z
Z Z

D Z
D Z
D Z
T Z

D T
D Z
D Z
D Z

D D
D T
D Z
D Z

D D
D D
D T
D Z

D D
D D
D D
D T

T

Code Overview

Data Codewords have “01” sync preamble

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

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

Code Summary

Hari Pattern	Sync		Bit fields 0-63							
DDDD/DDDD	0	1	D0	D1	D2	D3	D4	D5	D6	D7
ZZZZ/ZZZZ	1	0	0x1e	Z0	Z1	Z2	Z3	Z4	Z5	Z6	Z6
ZZZZ/SDDD	1	0	0x33	Z0	Z1	Z2	Z3		D5	D6	D7
SDDD/DDDD	1	0	0x78	D1	D2	D3	D4	D5	D6	D7
TZZZ/ZZZZ	1	0	0x87		Z1	Z2	Z3	Z4	Z5	Z6	Z7
DTZZ/ZZZZ	1	0	0x99	D0		Z2	Z3	Z4	Z5	Z6	Z7
DDTZ/ZZZZ	1	0	0xaa	D0	D1		Z3	Z4	Z5	Z6	Z7
DDDT/ZZZZ	1	0	0xb4	D0	D1	D2		Z4	Z5	Z6	Z7
DDDD/TZZZ	1	0	0xcc	D0	D1	D2	D3		Z5	Z6	Z7
DDDD/DTZZ	1	0	0xd2	D0	D1	D2	D3	D4		Z6	Z7
DDDD/DDTZ	1	0	0xe1	D0	D1	D2	D3	D4	D5		Z7
DDDD/DDDT	1	0	0xff	D0	D1	D2	D3	D4	D5	D6

There are three choices per bit, so frames can be composed with 64, 4:1 multiplexors controlled according to 1 of 12 frame types

Control code mapping

8B/10B	name	shorthand	7-bit line code
K28.0	idle1	R	0x00
K28.1	busy idle0	Kb	0x1e
K28.2	reserved0	-	0x2d
K23.7	busy idle1	Rb	0x33
K27.7	start	S	encoded by TYPE byte
K29.7	terminate	T	encoded by TYPE byte
K28.4	reserved1	-	0x4b
K28.5	idle0	K	0x55
K30.7	error	E	0x66
K28.7	reserved2	-	0x78

- The 7-bit line codes representing 8B/10B control characters have 4-bit minimum hamming distance.

False Packet Acceptance Rate

- A key parameter of any code is the rate at which “damaged” packets are accepted as valid. In general, such a failure is capable of hard crashing a computer system.
- For 1Gb Ethernet the Mean Time to False Packet Acceptance (MTTFPA) was calculated to be approximately 60 billion years.
- Because 64b/66b has a uniform 4-bit Hamming protection, a conservative estimate can be made. Assume that packets with four or more errors will generate a false packet acceptance event. In practice, this overestimates the failure rate by about 2^{32} .

False Packet Acceptance Rate

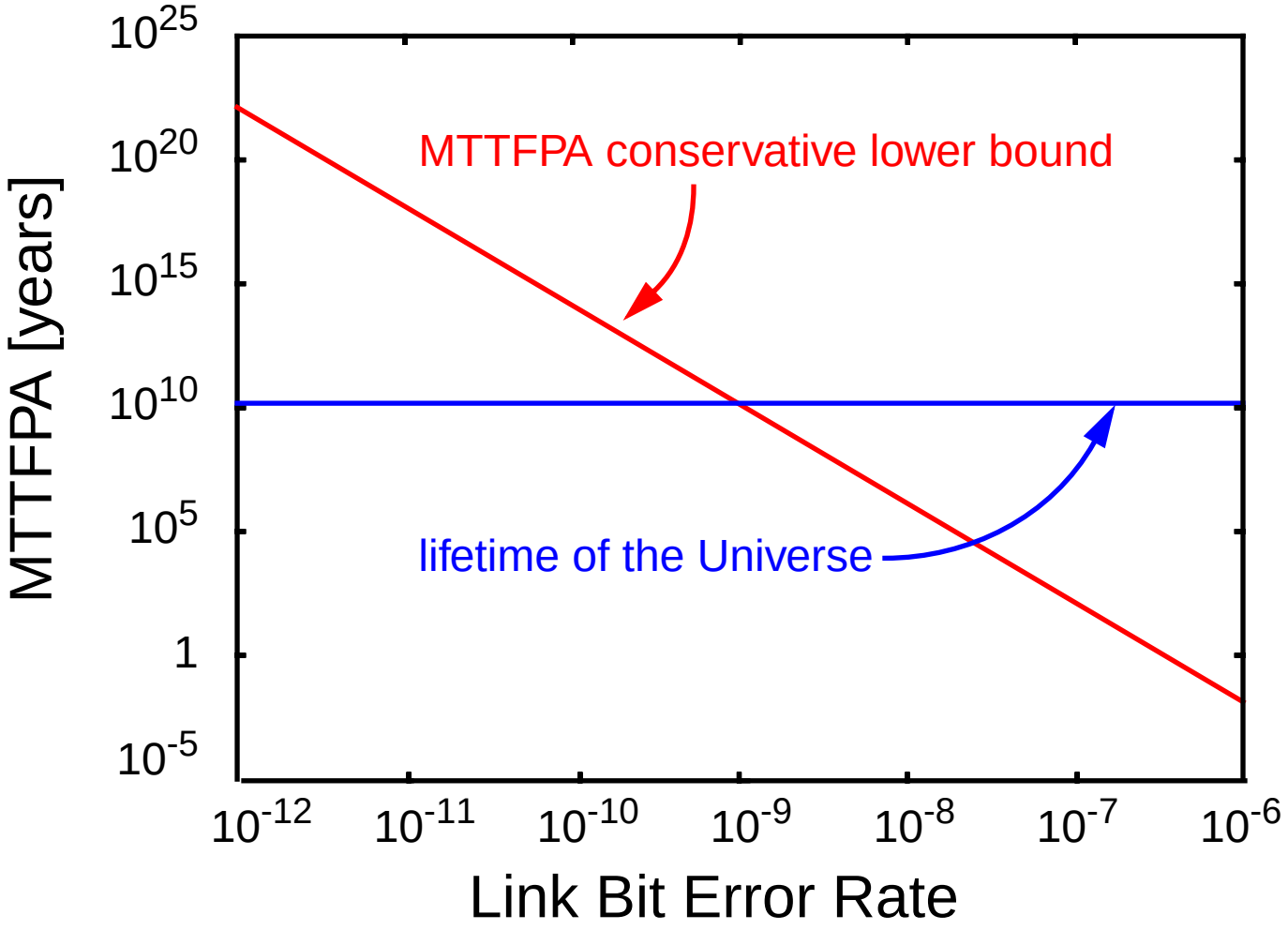
- P = coded packet size = 58+1526*8*66/64
- p_e = bit error rate, N = number of errors, t_b = bit time (1/10.3125G).
- probability of N errors in packet of size P:

$$p(N, P, p_e) = (1 - p_e)^{P-N} (p_e)^N \binom{P}{N}$$

- expected time for 4 or more errors:

$$MTTFPA > \frac{t_{bit} P}{1 - p(N, P, 0) - p(N, P, 1) - p(N, P, 2) - p(N, P, 3)}$$

False Packet Acceptance Rate

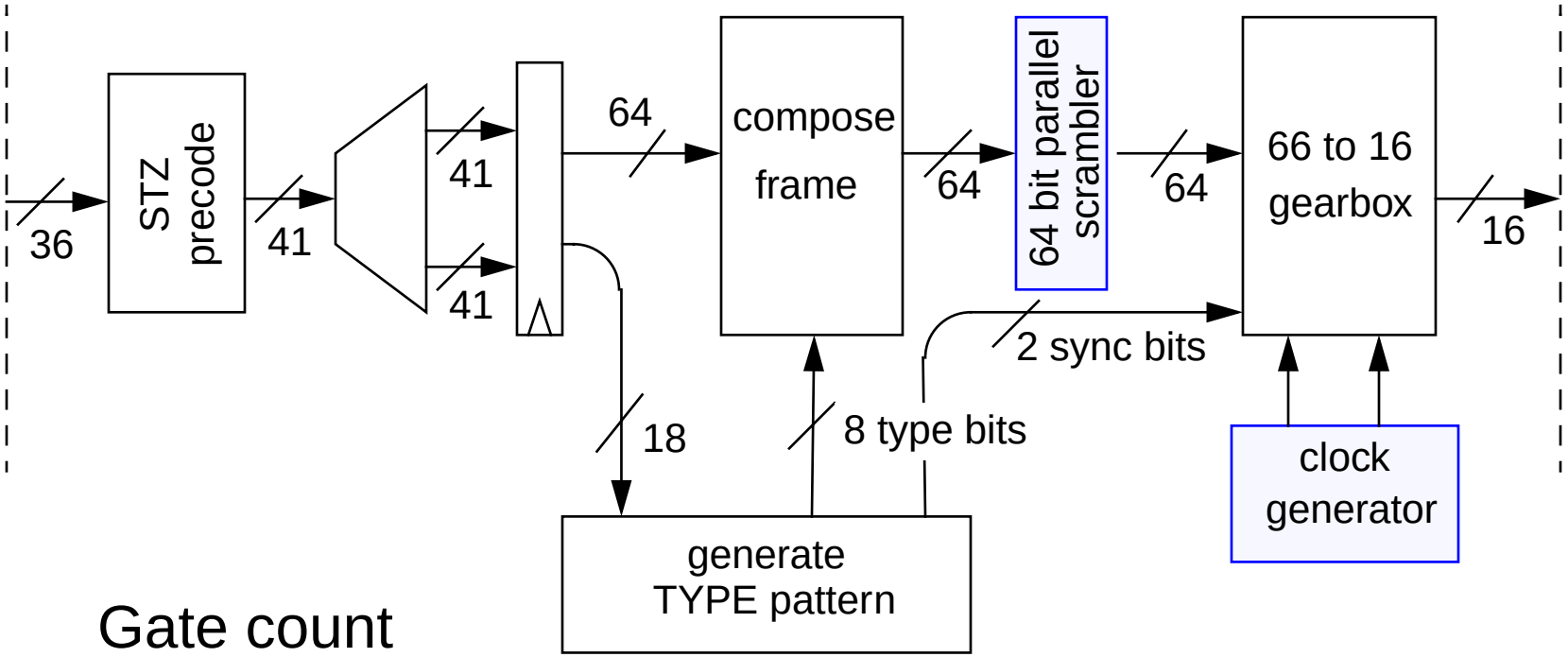


False Packet Acceptance Rate

Summary

- At a 10e-9 BER and 10.3Gb/s, the MTTFPA of 64b/66b is approximately equal to the 1Gb Ethernet 8b/10b performance at 10e-11 BER
- If 10G Ethernet maintains the same 10e-11 specification for PMD raw error rate, then 64/66b gives 7 orders of magnitude improvement in MTTFPA compared to coding used for 1G Ethernet

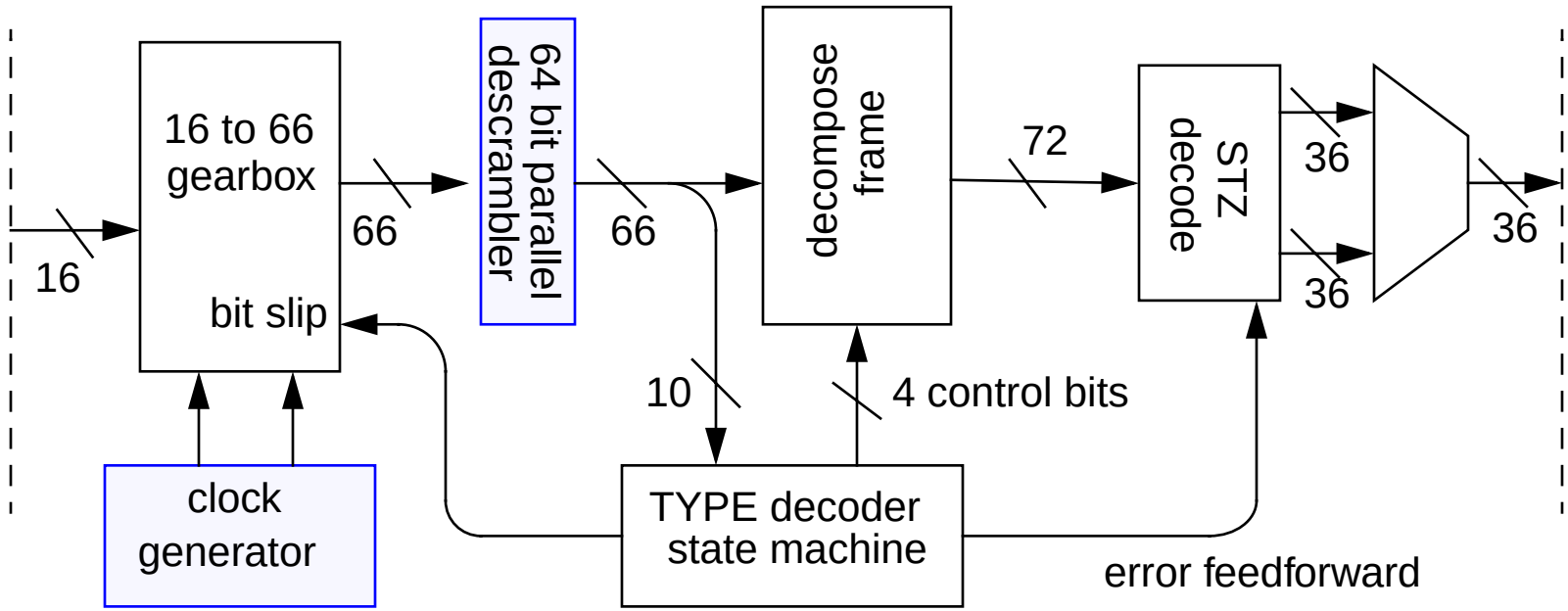
Coder Block Diagram



Gate count

- Logic: 731 logic cells + 234 flops
- Gearbox + clock generator: ~1400 flops

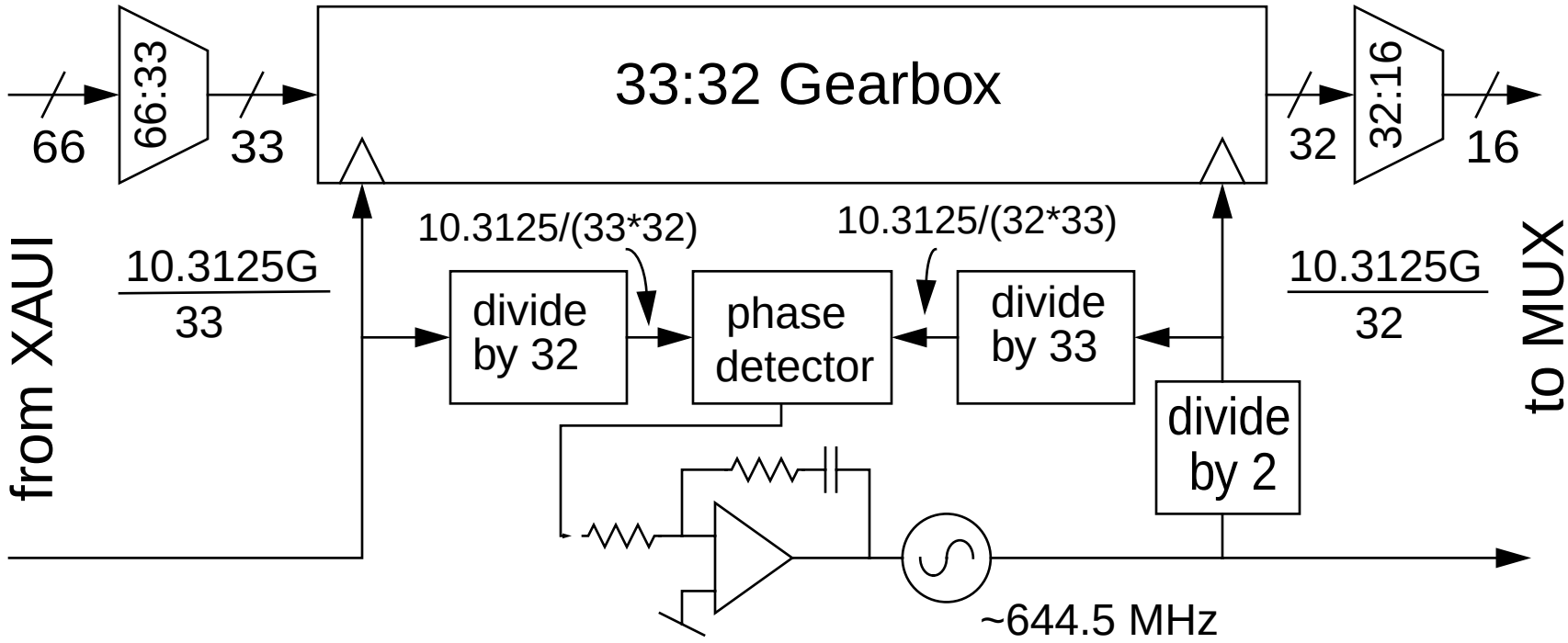
Decoder Block Diagram



Gate count

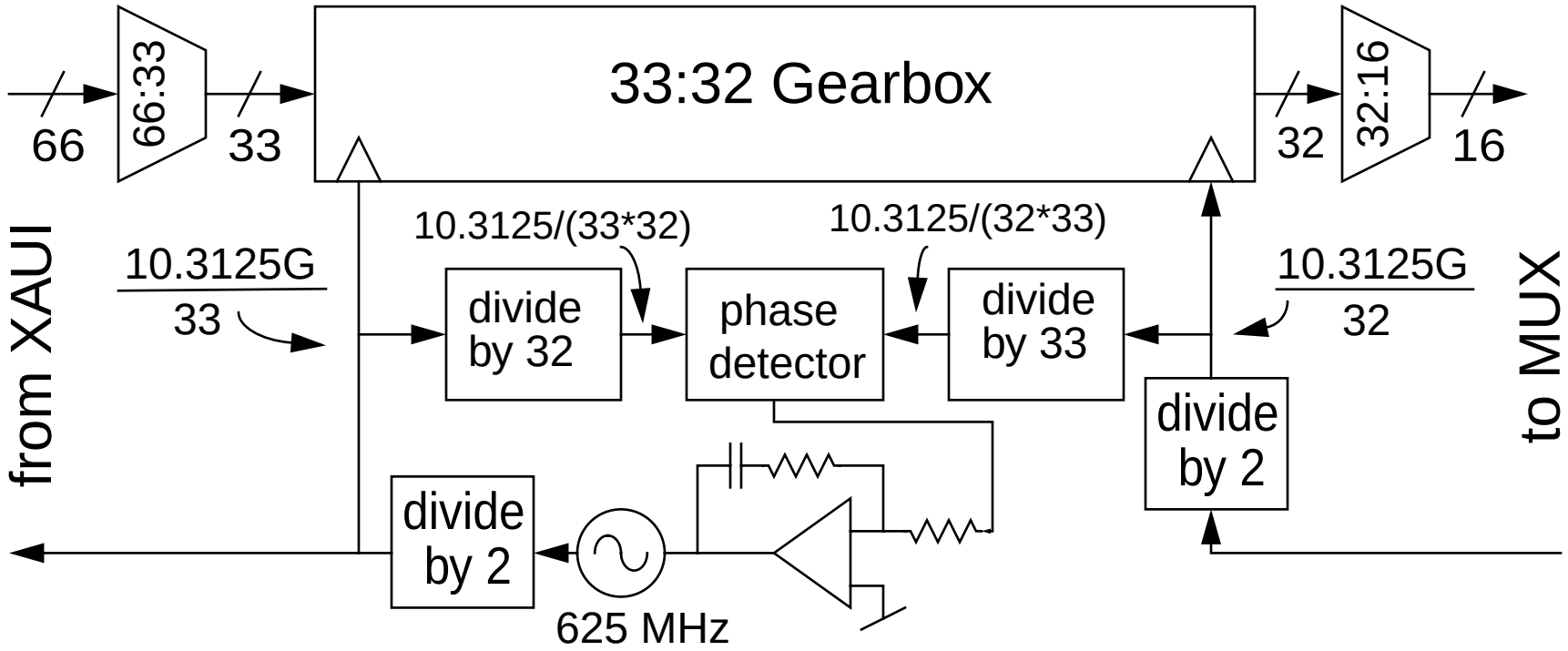
- Logic: 706 logic cells + 144 flops
- Gearbox + clock generator: ~1400 flops

TX Clock synthesis I



- PLL locks to received XAUI clock
- PLL provides clock to 16:1 MUX

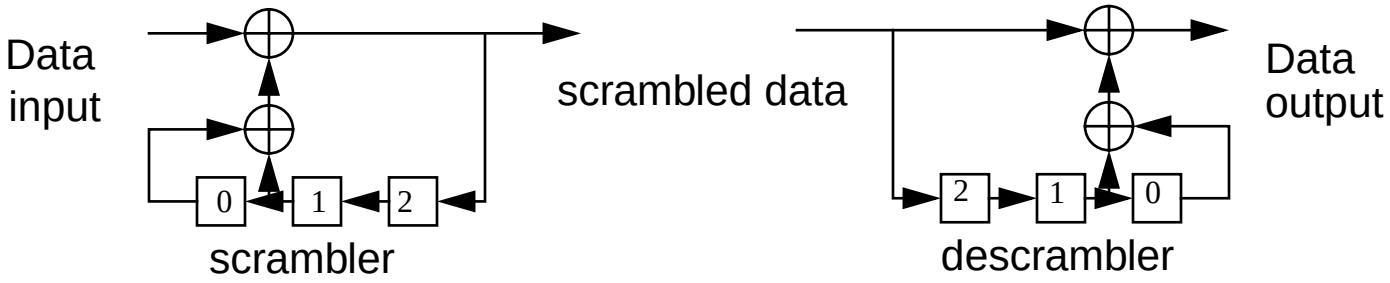
TX Clock synthesis II



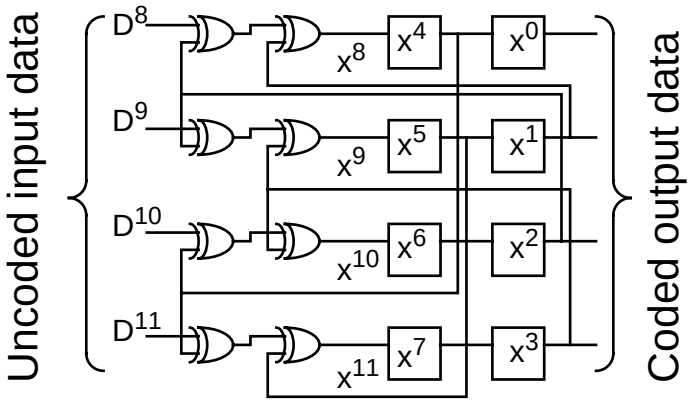
- PLL locks to MUX clock output
- PLL provides clock to XAUI output FIFO

Scrambling principle

An example 3-bit scrambler/descrambler in serial form:



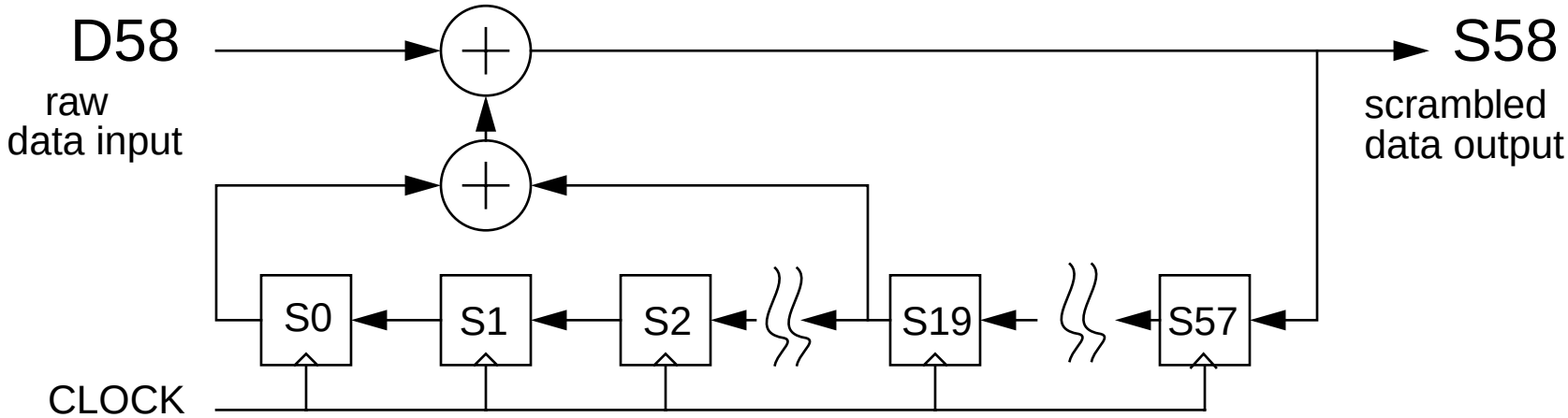
parallel form:



- Self synchronizing scrambler
- Can be parallelized for efficient implementation
- Using along pattern length reduces possibility of jamming (eg: $x^{58}+x^{19}+1=0$)
- Long pattern length self-synchronizing scramblers exist that do not compromise Ethernet CRC coverage

Derivation of Parallel Scrambler

Start with the Serial form of the Scrambler:



Write the recursion equation:

$S58 = D58 + S19 + S0$ or $D58 = S58 + S19 + S0$

notice that if we set the data input == 0, then we get
0 = S58 + S19 + S0, which is often written as: $X^{58} + X^{19} + 1 = 0$

Derivation of Parallel Scrambler

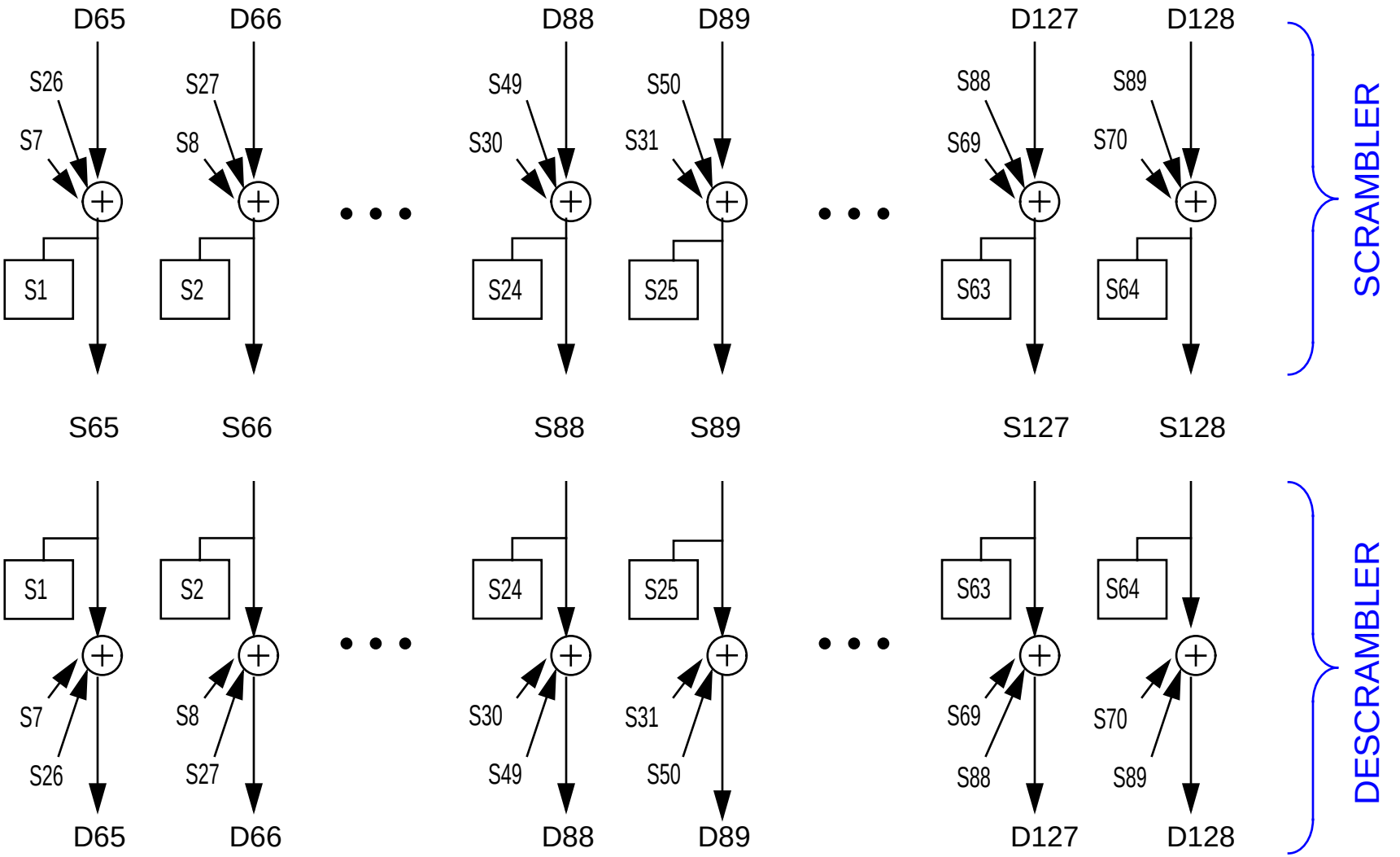
Use recursion equation to write terms for all parallel bits:

S58 = D58 + S19 +S0, so

S128 = D128 + S89 + S70
S127 = D127 + S88 + S69
S126 = D126 + S87 + S68
...
...
S66 = D66 + S27 + S8
S65 = D65 + S26 + S7

These equations are easily implemented with a parallel register and a handful of XOR gates.

Latency is equal to 2-cascaded, 3-input XOR delays, and 64 scrambled bits are computed all at once.


D65
D66
D88
D89
D127
D128
S26
S27
S49
S50
S88
S89
S7
S8
S30
S31
S69
S70
S1
S2
S24
S25
S63
S64
SCRAMBLER
S65
S66
S88
S89
S127
S128
S1
S2
S24
S25
S63
S64
S7
S8
S30
S31
S69
S70
S26
S27
S49
S50
S88
S89
D65
D66
D88
D89
D127
D128
DESCRAMBLER

Summary

- Since the last meeting, we've finished a gate level implementation and folded the results back into simplifying the code definition
- An analysis of Mean Time to False Packet Acceptance (MTTFPA) shows acceptable performance with 10e-9 BERs.
- Gate counts for 64/66 are reasonable and on par with the complexity of the four 8b/10b decoders need for XAUI
- Clock Generation is straightforward and easy in any process capable of XAUI PLL performance

Thread Links			Date Links		
Thread Prev	Thread Next	Thread Index	Date Prev	Date Next	Date Index

Re: 850 nm solutions

- *To*: "THALER,PAT (A-Roseville,ex1)" <pat_thaler@agilent.com>, "Rick Walker" <walker@cutter.hpl.hp.com>, <stds-802-3-hssg@ieee.org>
- *Subject*: Re: 850 nm solutions
- *From*: "Roy Bynum" <rabynum@mindspring.com>
- *Date*: Sat, 29 Apr 2000 18:05:54 -0500
- *References*: <1BEBA5E8600DD4119A50009027AF54A0C5F0EB@axcs04.cs.itc.hp.com>
- *Reply-To*: "Roy Bynum" <rabynum@mindspring.com>
- *Sender*: owner-stds-802-3-hssg@ieee.org

Pat,

What I am curious about what you say is that it is "different" "groups" that came up with "Hari" and "XAUI", but those "groups" seem to contain the same "people", and are representing the same "vendors" in the "different" "groups". If it is the same "people" then it is effectively the same "group" in the different organizations. Technical details aside, it is the massive similarities that demonstrate the "commonality" and the repeated efforts to insert it into an "unrelated" standard. When an organization is then formed that has limited closed membership with the effective purpose of creating voting blocks within the open organizations then the process of creating "open" standards becomes skewed. As a potential customer of the results of the IEEE P802.3ae TF, I find this disturbing.

Thank you,
Roy Bynum

----- Original Message -----
From: THALER,PAT (A-Roseville,ex1) <pat_thaler@agilent.com>
To: Roy Bynum <rabynum@mindspring.com>; THALER,PAT (A-Roseville,ex1) <pat_thaler@agilent.com>; Rick Walker <walker@cutter.hpl.hp.com>; <stds-802-3-hssg@ieee.org>
Sent: Friday, April 28, 2000 6:40 PM
Subject: RE: 850 nm solutions

> Roy,
>
> Your note seems to imply that Hari was developed within Infiniband and then
> introduced from
> there into 802.3. This is not my understanding of its history. The
> Infiniband group developed/
> is developing a 2.5 Gbaud/s serial link for use in 1-wide, 4-wide, and
> 12-wide configurations
> using the 8B/10B code. Somewhat in parallel with this, people from the Fibre
> Channel and
> Ethernet communitties got together to look at what might be good interfaces
> to use between
> physical layer chips for 10 Gbit/s implementations and came up with Hari and
> Sali which
> are roughly equivalent to the current proposals for XAUI and XGMII. These
> people also
> chose the 8B/10B code for Hari. Since one 4x2.5 Gbit'isn 8B/10B interface is
> pretty much
> like another, there is similarity between Hari and the Infiniband x4
> interface though
> there is a 25% speed difference.
>
> The interfaces were each developed by communities focused on their market's
> needs. In my
> opinion, the decision to use different speeds was driven by differences in
> the respective
> market needs.
>
> An interface at these speeds is analog. This is particularly true if it is
> to serve the
> length of traces likely to be found between transceivers and switch chips.
> Taking analog
> considerations into account when we develop the standard will enable
> cost-effective,
> robust designs. XAUI is very suitable to the use for which it has been
> proposed.
>
> The point of my note was: if we were going to standardize a short run copper
> link, it
> would make sense to look at what could be done on a 4-wide connection vs. a

> 10 Gbit
> serial connection. Our existing decision has been to not do a short copper
> link -
> probably driven in part by the low usage of 1000BASE-CX.
>
> Regards,
> Pat Thaler
>
> -----Original Message-----
> From: Roy Bynum [mailto:rabynum@mindspring.com]
> Sent: Wednesday, April 26, 2000 6:43 PM
> To: THALER,PAT (A-Roseville,ex1); Rick Walker; stds-802-3-hssg@ieee.org
> Subject: Re: 850 nm solutions
>
>
> Pat,
>
> For Infiniband, I think that HARI is a very good solution. I question the
> way that it was introduced and developed as part of the
> effort in something that is not Infiniband. If people want to make products
> for Infiniband, I have no problem with that. As a
> customer, I question the motivations of my vendors to have me pay for the
> development of technology that was actually intended for
> another use. I wonder how much that has already increased the price of the
> product that I will be receiving. I wonder even more
> how much the vendor was actually trying to develop something for my use
> instead of somebody else, and gave me, the customer, the
> "left overs". I wonder how much better the product, that I may buy, would
> have been better if the vendor had not been developing
> technology for another use.
>
> As a customer, I was hoping to receive an 802.3 Ethernet product that
> treated the interface to the optical domain as a digital
> optical system, not an analog copper system, which you refer to for the use
> of HARI. As a customer I was hoping that the vendors
> would listen to me and my requirements and look at it as an opportunity to
> enter a market that is as large as the global Internet,
> instead of staying in the collective enterprise space. Vendors that are not
> looking at the market correctly have already lost their
> market share in the Internet backbone, and they are about to start loosing
> it at the access edge as well. History has shown that
> customers will get what they want one way or another.
>
> The response of a BIG customer,
> Thank you,
> Roy Bynum
>
>
>
>
> ----- Original Message -----
> From: THALER,PAT (A-Roseville,ex1) <pat_thaler@agilent.com>
> To: Rick Walker <walker@cutter.hpl.hp.com>; <stds-802-3-hssg@ieee.org>
> Sent: Wednesday, April 26, 2000 1:08 PM
> Subject: RE: 850 nm solutions
>
>
> >
> >
> > Infiniband will be using something very similar to the HARI interface over
> > short copper links though the distance goal is, I think, 6 m. To travel
> over
> > short copper cables, it may make sense to use a 4 wide signal from HARI
> > rather than 10 Gbit/s serial.
> >
> > -----Original Message-----
> > From: Rick Walker [mailto:walker@cutter.hpl.hp.com]
> > Sent: Wednesday, April 19, 2000 4:58 PM
> > To: stds-802-3-hssg@ieee.org
> > Subject: Re: 850 nm solutions
> >
> >
> >
> >
> > > Jim Tatum writes:
> > > > But why does it matter? Why limit the users? Why not put in the table.
> > It
> > > > costs nothing. Just put in what the model and data tell us to. It is
> > > > my opinion that a large percentage of 10GB style links are going to be
> > > > very short, less than 10m. If you look at the way many fiber ports
> > > > are being used today, many are in the 10m range. Also, since copper
> > > > cables are going to be EXTREMELY challanged to go that distance at
> > > > 10GB, why not let the market choose the lowest cost solution using
> > > > 850nm VCSELs and 62.5um fiber?
> > >
> > > FWIW, I agree that 10G across CAT-6 or other twisted pair would be very
> > > difficult. However 10G across coaxial cable is fairly easy. It can be

> > done with 0.1" diameter coaxial cable using simple NRZ data encoding. A
> > simple FIR pre-equalizer can double this distance. Without a doubt
> > copper would be the cheapest solution for links under 10M. I would
> > estimate a mature chipset price of about $50 per end and $15 for the
> > cable.
> >
> > This performance was demonstrated in 1998 using a 25GHz bipolar chipset.
> > See: Walker, R. C., K. Hsieh, T. A. Knotts and C. Yen, "A 10Gb/s
> > Si-Bipolar TX/RX Chipset for Computer Data Transmission" , ISSCC Digest
> > of Technical Papers 41(February 1998), 302,303,450.
> >
> > A Copper PHY was voted down by the committee because it was thought that
> > there was no market for this type of low-cost short distance link.
> >
> > kind regards,
> > --
> > Rick Walker

- **References**:
  - **RE: 850 nm solutions**
    - *From:* THALER,PAT (A-Roseville,ex1)

- Prev by Date: **RE: XAUI IO specs**
- Next by Date: **RE: XAUI IO specs**
- Prev by thread: **RE: 850 nm solutions**
- Next by thread: **Re: 850 nm solutions**
- Index(es):
  - **Date**
  - **Thread**

64b/66b PCS

updated 6/30/2000

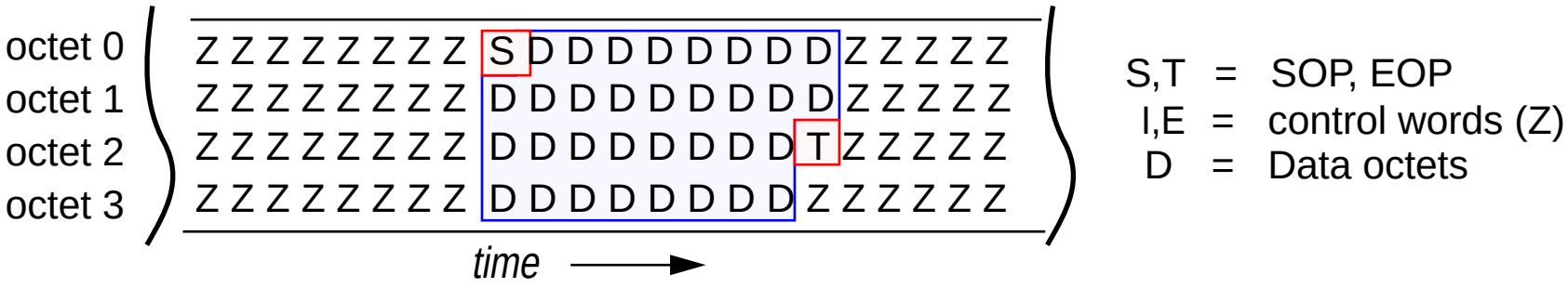
state machines modified 7/17/2000

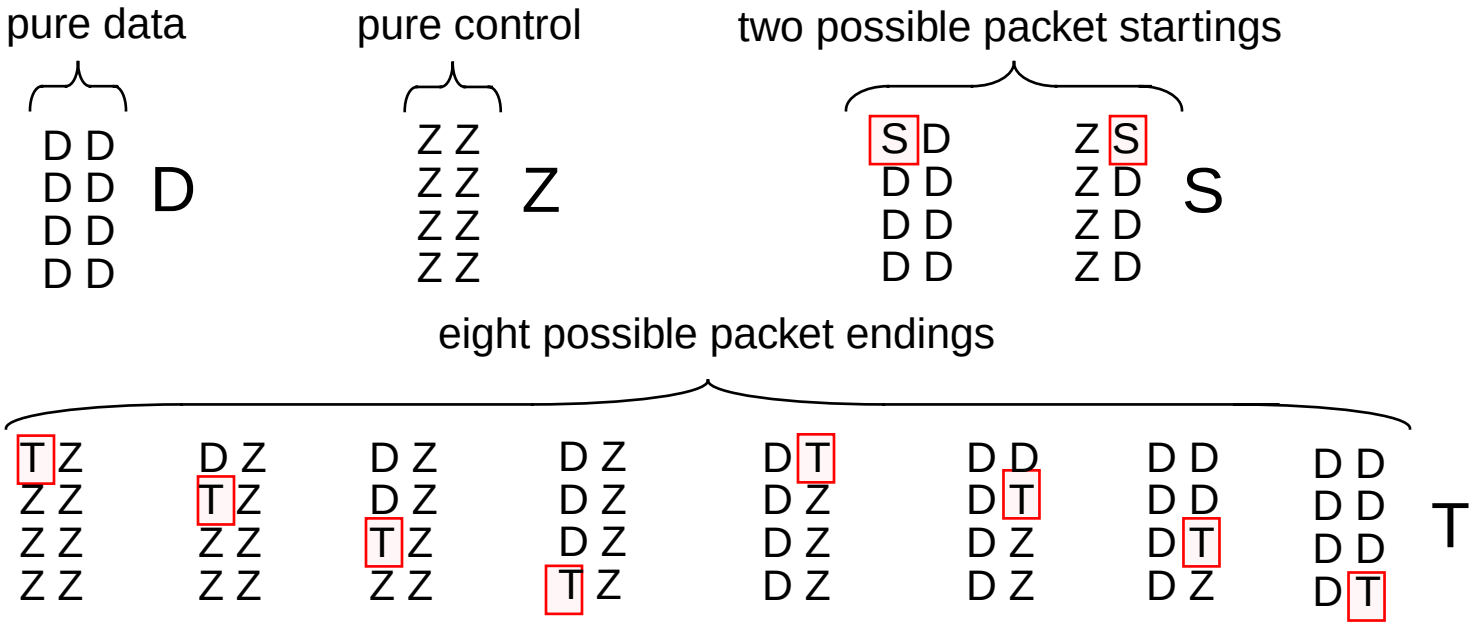
Rick Walker	Agilent	Howard Frazier	Cisco
Richard Dugan	Agilent	Paul Bottorff	Nortel
Birdy Amrutur	Agilent	Shimon Mueller	Sun
Rich Taborek	nSerial	Brad Booth	Intel
Don Alderrou	nSerial	Kevin Daines	World Wide Packets
John Ewen	IBM	Osamu Ishida	NTT
Mark Ritter	IBM	Jason Yorks	Cielo
Al Bezoni	Lucent	Henning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

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





Code Overview

Data Codewords have “01” sync preamble

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

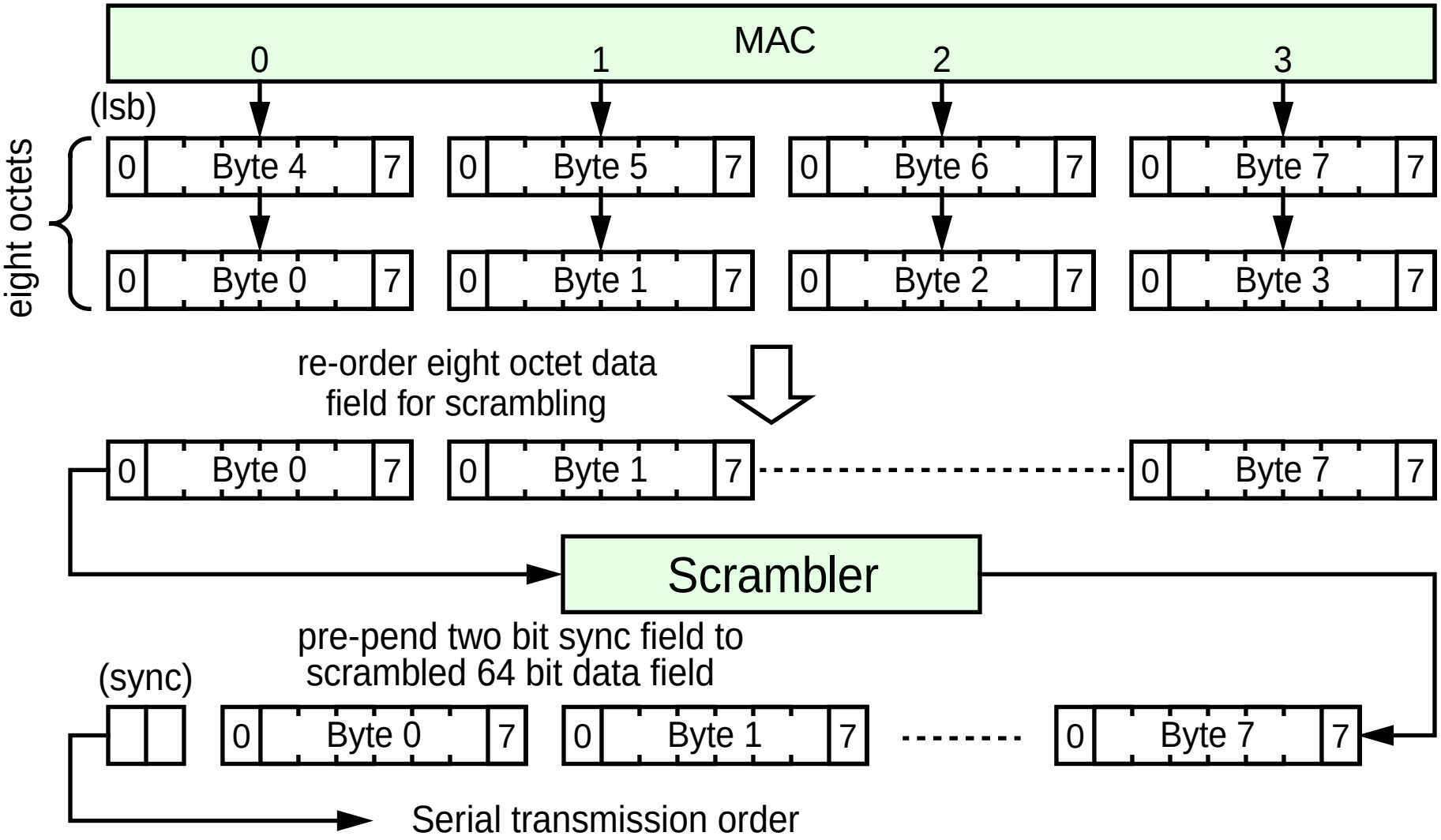
Input Data (first RS transfer / second RS transfer)	Sync [0]	Sync [1]	Bit fields [2]							[65]
$D_0D_1D_2D_3/D_4D_5D_6D_7$	0	1	D_0 [0] [7]	D_1 [0] [7]	D_2 [0] [7]	D_3 [0] [7]	D_4 [0] [7]	D_5 [0] [7]	D_6 [0] [7]	D_7 [0] [7]
$Z_0Z_1Z_2Z_3/Z_4Z_5Z_6Z_7$	1	0	0x1e "01111000"	C_0 [0] [6]	C_1 [0] [6]	C_2 [0] [6]	C_3 [0] [6]	C_4 [0] [6]	C_5 [0] [6]	C_6 [0] [6]	C_7 [0] [6]
$Z_0Z_1Z_2Z_3/S_4D_5D_6D_7$	1	0	0x33	C_0	C_1	C_2	C_3		D_5	D_6	D_7
$S_0D_1D_2D_3/D_4D_5D_6D_7$	1	0	0x78	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
$T_0Z_1Z_2Z_3/Z_4Z_5Z_6Z_7$	1	0	0x87		C_1	C_2	C_3	C_4	C_5	C_6	C_7
$D_0T_1Z_2Z_3/Z_4Z_5Z_6Z_7$	1	0	0x99	D_0		C_2	C_3	C_4	C_5	C_6	C_7
$D_0D_1T_2Z_3/Z_4Z_5Z_6Z_7$	1	0	0xaa	D_0	D_1		C_3	C_4	C_5	C_6	C_7
$D_0D_1D_2T_3/Z_4Z_5Z_6Z_7$	1	0	0xb4	D_0	D_1	D_2		C_4	C_5	C_6	C_7
$D_0D_1D_2D_3/T_4Z_5Z_6Z_7$	1	0	0xcc	D_0	D_1	D_2	D_3		C_5	C_6	C_7
$D_0D_1D_2D_3/D_4T_5Z_6Z_7$	1	0	0xd2	D_0	D_1	D_2	D_3	D_4		C_6	C_7
$D_0D_1D_2D_3/D_4D_5T_6Z_7$	1	0	0xe1	D_0	D_1	D_2	D_3	D_4	D_5		C_7
$D_0D_1D_2D_3/D_4D_5D_6T_7$	1	0	0xff	D_0	D_1	D_2	D_3	D_4	D_5	D_6	

- all undefined bit fields (in yellow) are set to zero for 10GbE

RS “Z” code to 7 bit “C” field mapping

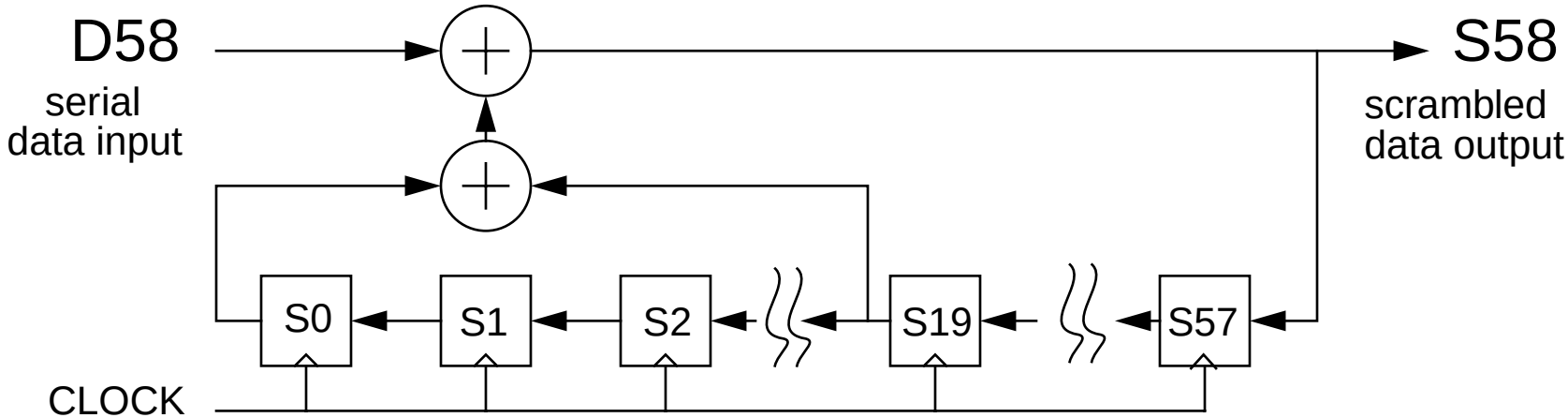
RS Z value	name	shorthand	7-bit C field line code
0x07,1	idle	[I]	0x00
0xfb,1	start	[S]	encoded by TYPE byte
0xfd,1	terminate	[T]	encoded by TYPE byte
0xfe,1	error	[E]	0x1e
0x1c,1	reserved0	-	0x2d
0x3c,1	reserved1	-	0x33
0x7c,1	reserved2	-	0x4b
0xbc,1	reserved3	-	0x55
0xdc,1	reserved4	-	0x66
0xf7,1	reserved5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

Bit ordering sequence



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 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  30,0 77,0 00,0 00,0 7a,0 0c,0 50,0 12,0
1e,0 d2,0 62,0 84,0 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 "10" 78 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 00 93 eb f7 79 "10" 87 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 demultiplexor 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

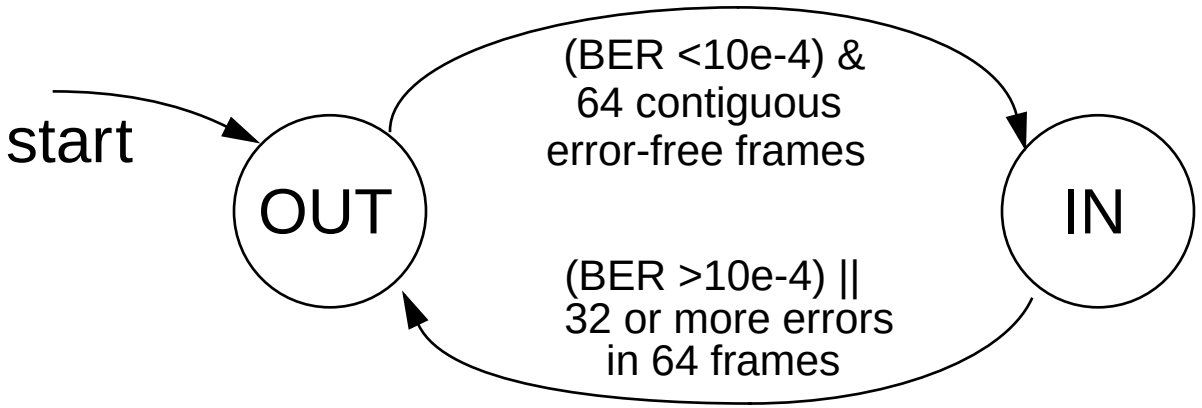
If already aligned with good BER (<10e-9), then we want to stay in sync with very high reliability

If BER is worse than10e-4 we should suppress sync, to avoid likelyhood of False Packet Acceptance due to CRC failures

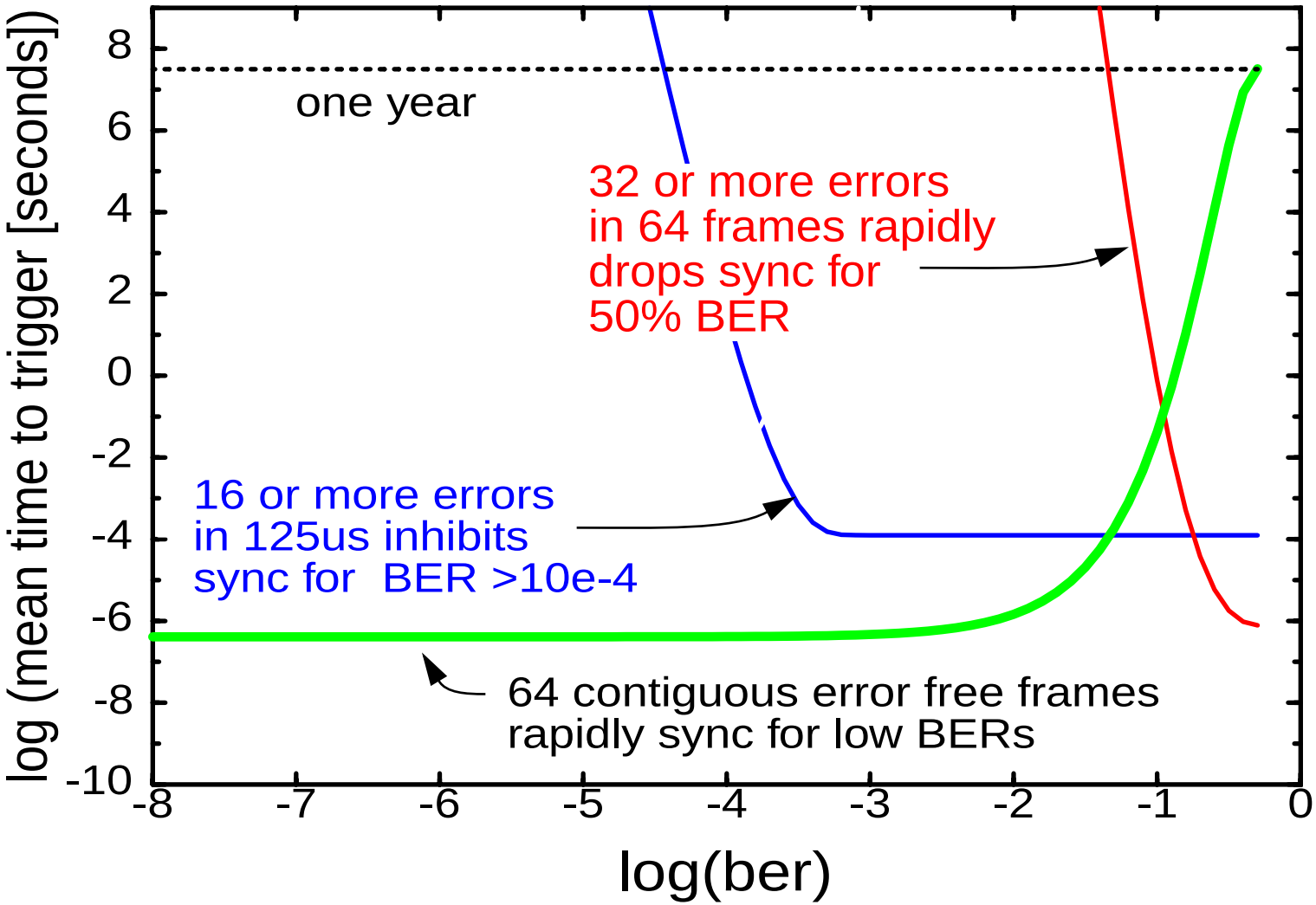
BER	current sync state	next sync state	notes
~50%	in	out	should be fast
>10e-4	in	out	prevents MTTFPA events, can be relatively slow to trigger
<10e-9	out	in	should be fast

Frame sync algorithm

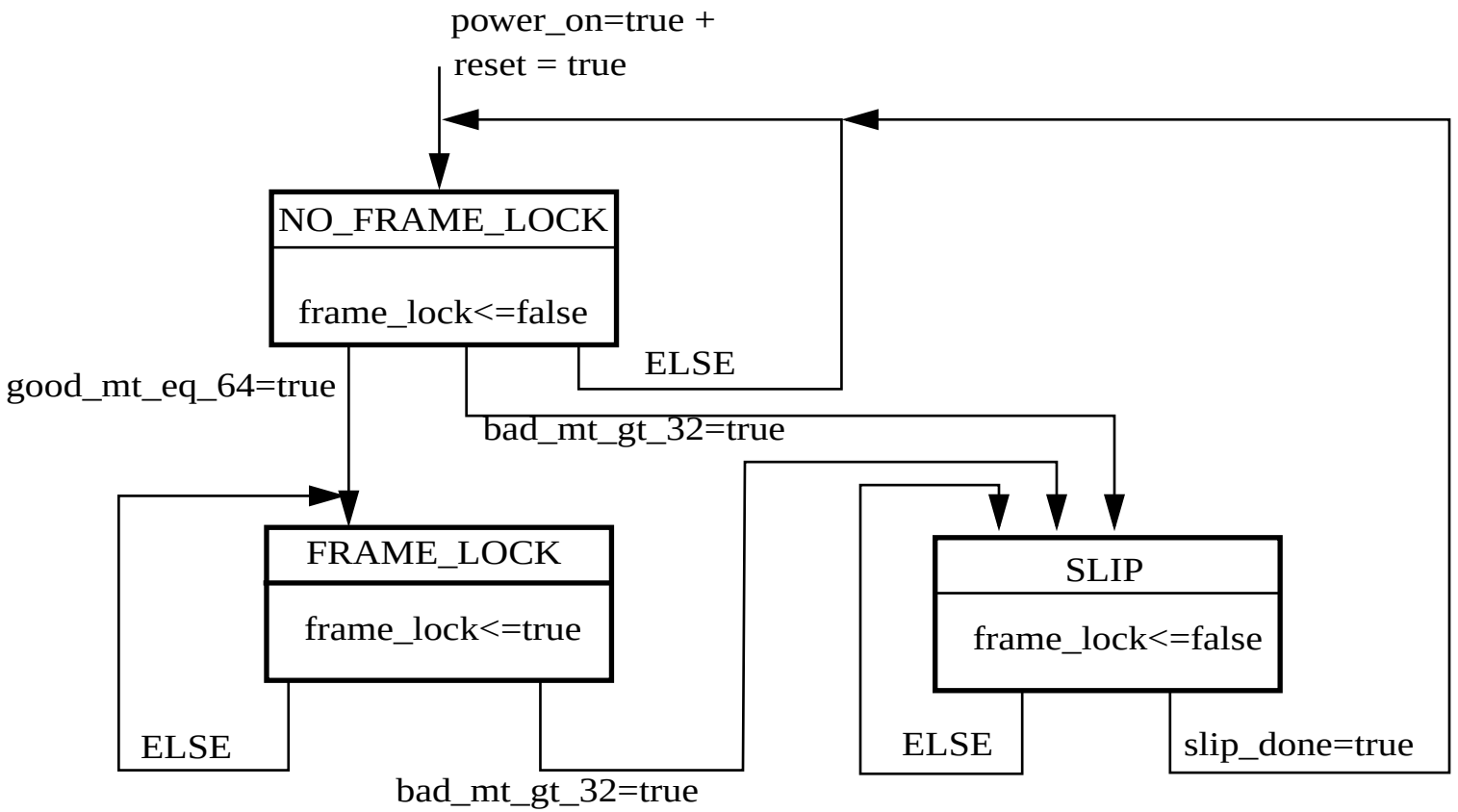
- frame sync is acquired after 64 contiguous frames have been received with valid “01” or “10” sync headers
- frame sync is declared lost after 32 “11” or “00” sync patterns have been declared in any block of 64 frames
- In addition, if there are 16 or more errors within any 125us time interval (~10e-4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



64/66 frame sync performance

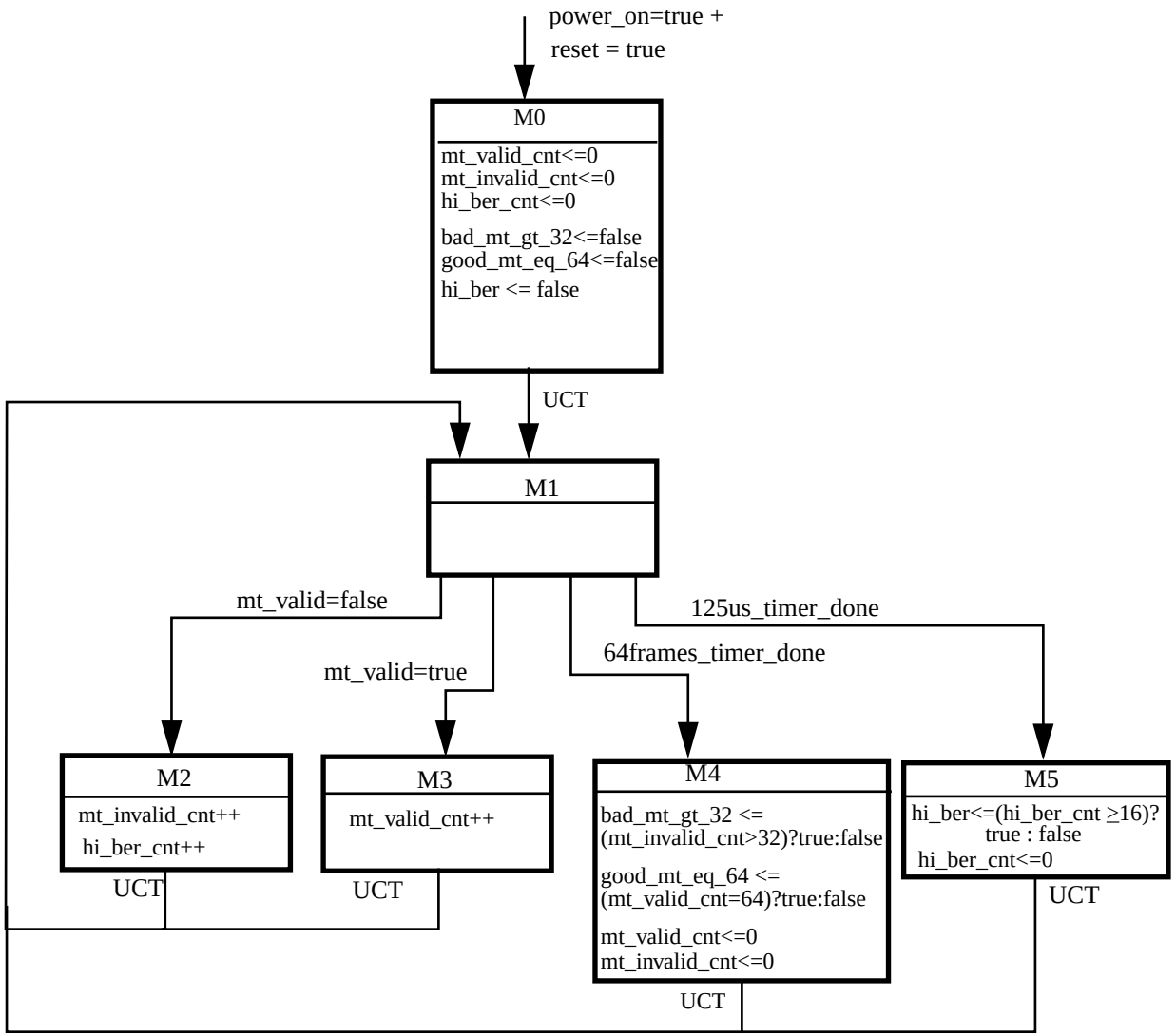


Frame lock process



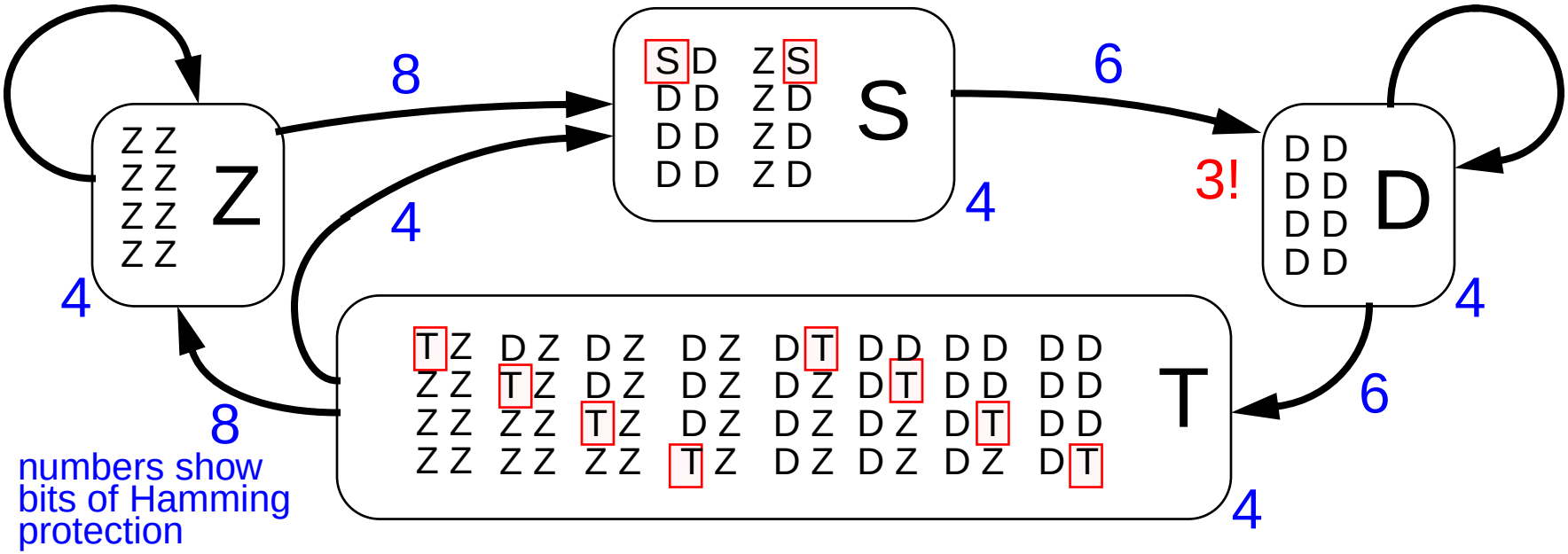
Receiver Synchronization condition
sync_done <= frame_lock=true * hi_ber=false

BER monitor process

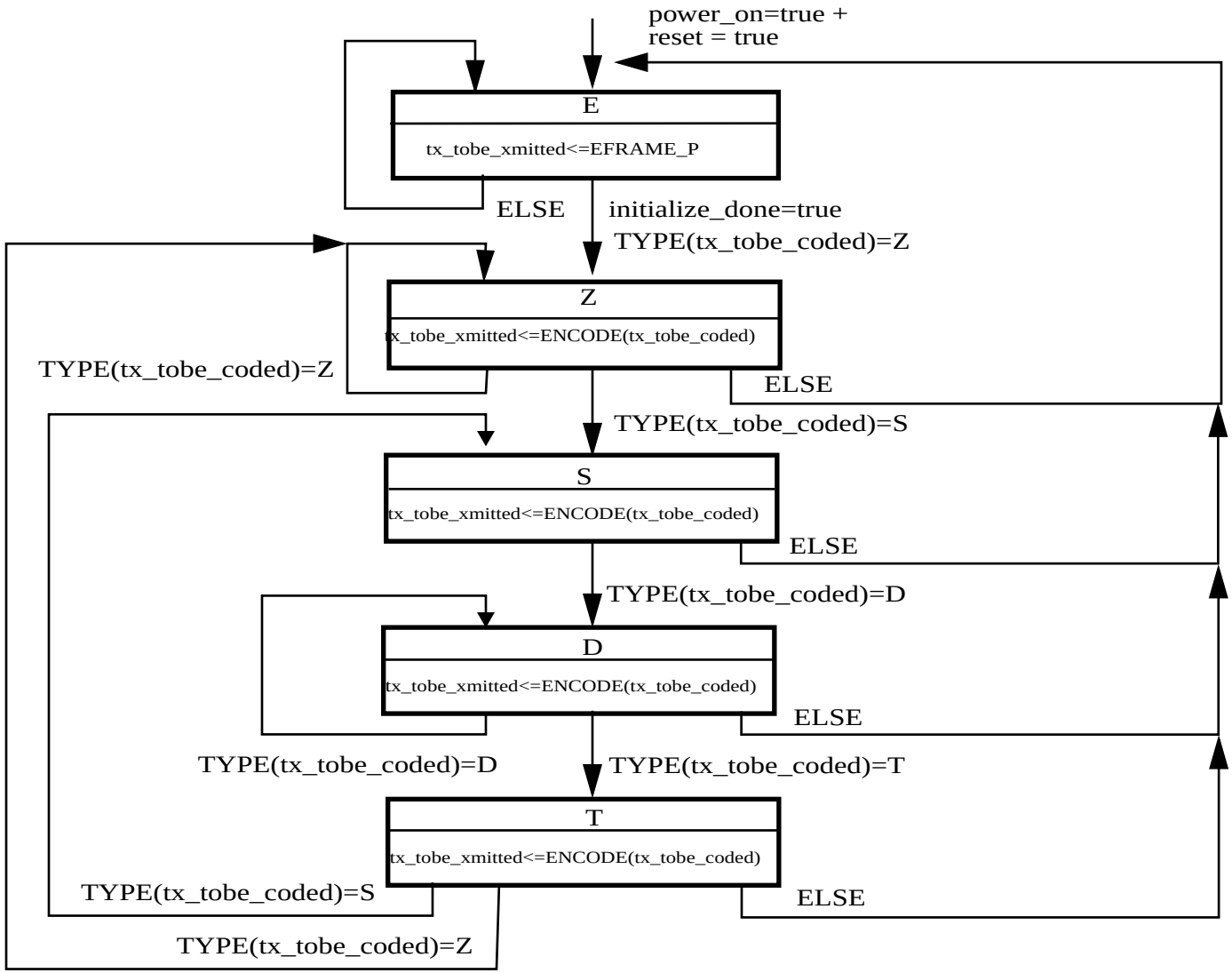


Packet boundary protection

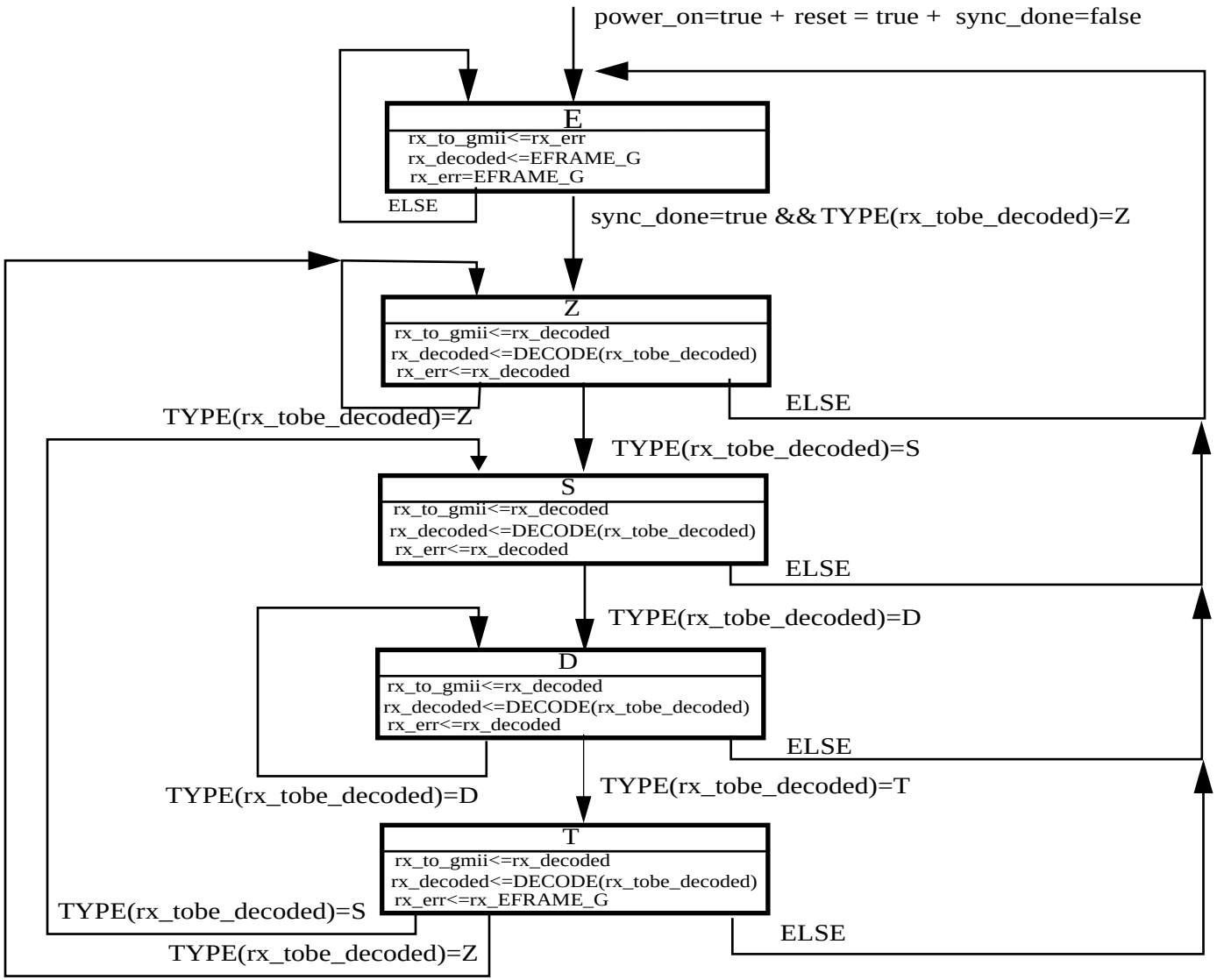
- A 2 bit error in the sync preamble can convert a packet boundary (S,T) into a Data frame (D) and vice-versa. However, all such errors violate frame sequencing rules unless another 4 errors recreate a false S,T packet (a total of six errors). Frame sequence errors invalidate the packet by forcing an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



TX process


power_on=true +
reset = true
E
tx_tobe_xmitted<=EFRAME_P
ELSE
initialize_done=true
TYPE(tx_tobe_coded)=Z
Z
tx_tobe_xmitted<=ENCODE(tx_tobe_coded)
TYPE(tx_tobe_coded)=Z
ELSE
TYPE(tx_tobe_coded)=S
S
tx_tobe_xmitted<=ENCODE(tx_tobe_coded)
ELSE
TYPE(tx_tobe_coded)=D
D
tx_tobe_xmitted<=ENCODE(tx_tobe_coded)
ELSE
TYPE(tx_tobe_coded)=D
TYPE(tx_tobe_coded)=T
T
tx_tobe_xmitted<=ENCODE(tx_tobe_coded)
TYPE(tx_tobe_coded)=S
ELSE
TYPE(tx_tobe_coded)=Z

RX process


power_on=true + reset = true + sync_done=false
E
rx_to_gmii<=rx_err
rx_decoded<=EFRAME_G
rx_err=EFRAME_G
ELSE
sync_done=true && TYPE(rx_tobe_decoded)=Z
Z
rx_to_gmii<=rx_decoded
rx_decoded<=DECODE(rx_tobe_decoded)
rx_err<=rx_decoded
TYPE(rx_tobe_decoded)=Z
ELSE
TYPE(rx_tobe_decoded)=S
S
rx_to_gmii<=rx_decoded
rx_decoded<=DECODE(rx_tobe_decoded)
rx_err<=rx_decoded
ELSE
TYPE(rx_tobe_decoded)=D
D
rx_to_gmii<=rx_decoded
rx_decoded<=DECODE(rx_tobe_decoded)
rx_err<=rx_decoded
ELSE
TYPE(rx_tobe_decoded)=D
TYPE(rx_tobe_decoded)=T
T
rx_to_gmii<=rx_decoded
rx_decoded<=DECODE(rx_tobe_decoded)
rx_err<=rx_EFRAME_G
TYPE(rx_tobe_decoded)=S
ELSE
TYPE(rx_tobe_decoded)=Z

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

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

rs value	name	shorthand	7-bit line code
0x5c,1	FC ordered-set	[Of]	encoded by TYPE byte
0x9c,1	10 GbE Link Signalling	[LS]	encoded by TYPE byte

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

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.

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

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

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

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

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

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

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