Stateless 64B/66B Encode/Decode for 800GbE and 1.6TbE

Eugene Opsasnick - Broadcom

IEEE P802.3df
October 2022

Introduction

- The focus of this proposal:
  - The state diagrams that track correct sequences of blocks
  - The state diagrams are shown in:
    - Fig. 119-14 and 119-15 (200/400GBASE-R)
    - As well as Fig. 82-16 and 82-17 (40GBASE-R and 100GBASE-R)
    - Essentially the same as Fig. 49-16 and 49-17 (10GBASE-R)

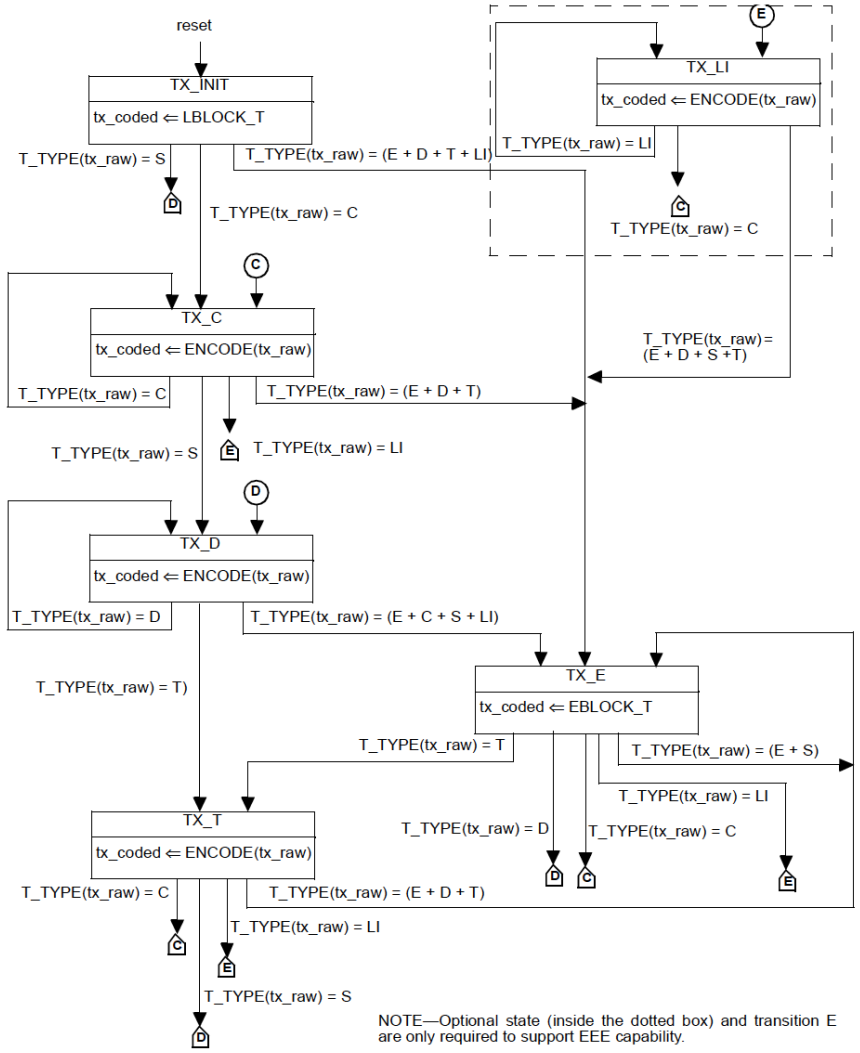
- Not the focus of the proposal:
  - The 64B/66B Block Codes defined in CL 82 (also used in CL 119)
  - Figure 82-5 defines the block codes
  - This proposal does not propose changes to these codes

Figure 82-5: 64B/66B Block Formats

- No Changes
- (D)ata Block
  - Sync=01
- (C)ontrol Block
  - Sync=10
  - BT=0x1E or 0X4B
- (S)tart Block
  - Sync=10
  - BT=0x78
- (T)erminate Block
  - Sync=10
  - BT=0x87, 0x99,0xAA,0xB4,0xCC, 0xD2,0xE1, or 0xFF
- (E)rror Block
  - Anything else

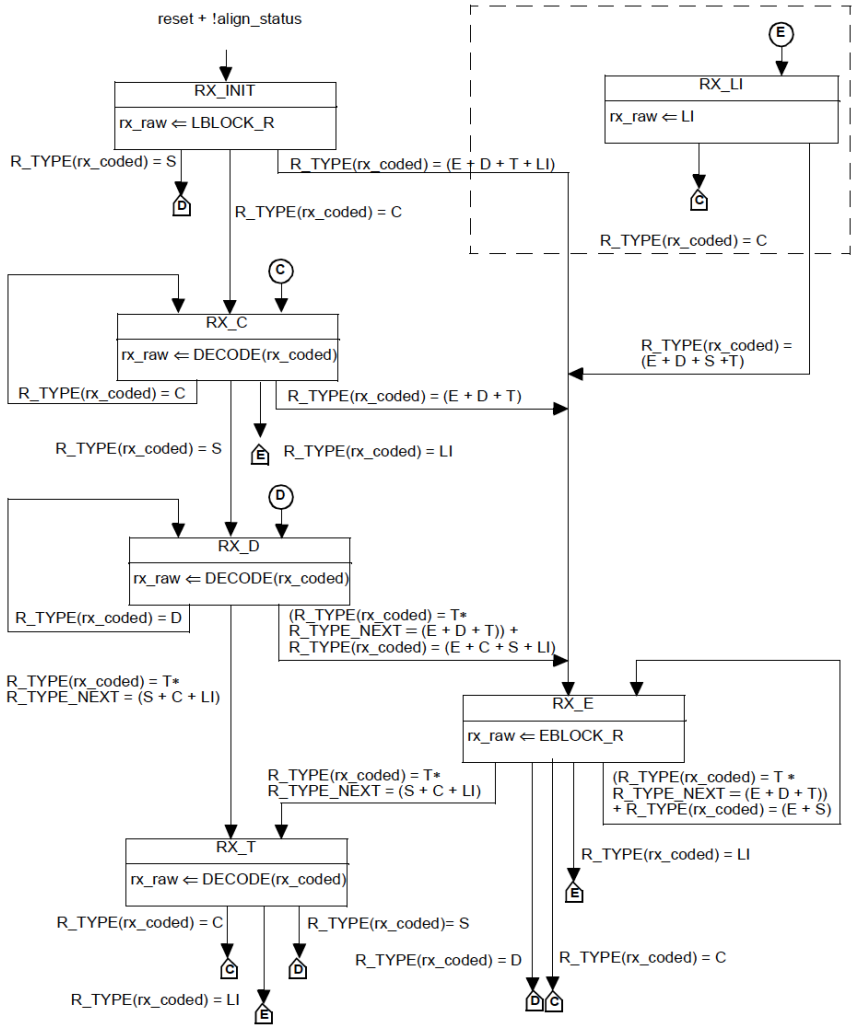
Input Data	Sync	Block Payload							
Bit Position: / Data Block Format:	0 1	2							65
$D_0\ D_1\ D_2\ D_3\ D_4\ D_5\ D_6\ D_7$	01	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
Control Block Formats:		**Block Type Field**							
$C_0\ C_1\ C_2\ C_3\ C_4\ C_5\ C_6\ C_7$	10	0x1E	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
$S_0\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

Figure 119-14: Transmit State Diagram



- After Reset, the “normal flow” is:
  1. State TX_C
     - One or more IDLE Control blocks
  2. State TX_D
     - Transmit a (S)tart block
  3. State TX_D * n
     - Several blocks of (D)ata
  4. State TX_T
     - Transmit a (T)erminate block
  - Repeat sequence 1-4
    - Can skip #1 and directly send (S) block in step #2.
- For any (E)rror block or illegal sequence:
  - Go to TX_E Error State, send EBLOCK_T
  - Following TX_E state, all block types are ok to send except (S)tart blocks.
    - **<u>This restriction on (S)tart blocks is not necessary</u>**

Figure 119-15: Receive State Diagram



NOTE—Optional state (inside the dotted box) and transition E are only required to support EEE capability.

- After Reset, the ”normal Flow” is
  - Same as TX flow
  - RX_C (Idles) -> RX_D (Start) -> RX_D (Data) -> RX_T (Terminate)
  - Repeat
- For any (E)rror block or illegal sequence:
  - Go to RX_E Error State, send EBLOCK_R
  - Following RX_E state, all block types are ok to send except (S)tart blocks.
    - **<u>This restriction on (S)tart blocks is not always necessary</u>**
- Additional Restriction on (T)erminate blocks
  - The next block after the (T) must be a correct block type: (S) or (C) or (LI)
    - **<u>This restriction on (T) blocks is not necessary</u>**

State Diagram Observations

- TX and RX State Diagrams enforce correct block sequences
  - Each block must be evaluated to generate the next state before encoding the next block.
  - Bad block:
    - Replace block with EBLOCK (a control block with 8 Error chars)
  - Bad sequence:
    - Also replace block with EBLOCK
  - Both “bad block” and “bad sequence” result in going to the Error State
    - In Error state, a good Start block is replaced with an EBLOCK
      - Necessary in RX following a “Bad block” - Scrambler can cause next block to be bad even if good FEC.
  - RX State Diagram also looks for a correct “next” block to validate a Terminate block
    - Why is this needed? Probably just an “extra check” that could be done before FEC was introduced?
- Since FEC was introduced
  - Hamming distance between block type codes is not guaranteed due to transcoding to 257b
  - Most RX errors are either corrected by FEC, else block is marked as “bad” if uncorrectable.

Implications of Faster Port Speeds

- As port speeds increase, implementations can either increase clock frequency or make data busses wider
  - It is getting harder to increase clock frequency to keep up with port speed increases
  - Data busses are most likely getting wider

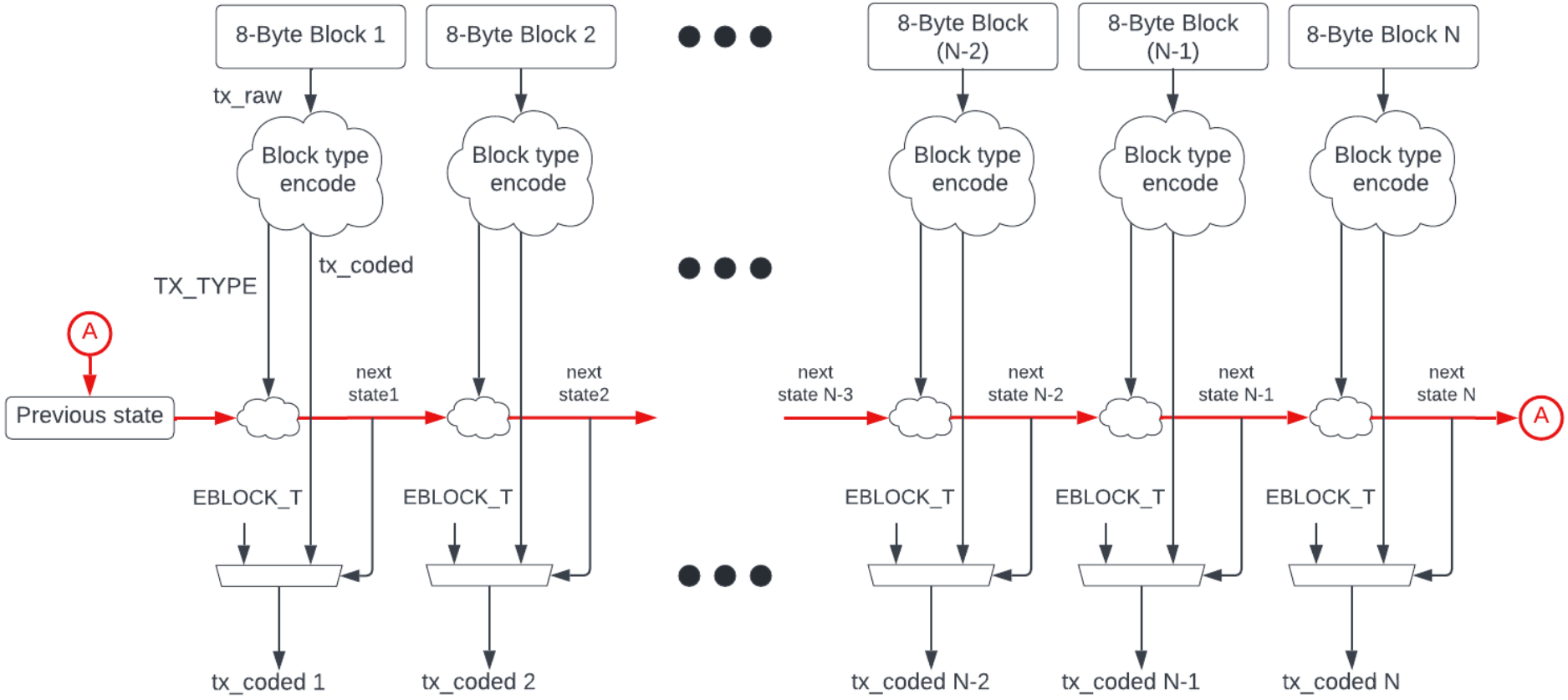
- As the internal PCS data bus gets wider, the 64B/66B encode/decode state machines are creating a longer logic path within a single cycle
  - Each 8-byte block must be encoded/decoded one at a time and influences the encoding of the following 8-byte block due to the CL 119 state diagrams.
  - With multiple 8-byte blocks within the PCS data bus, logic must propagate from the first 8-byte block all the way through to the last 8-byte block in a single cycle.

Example data bus width and frequency required for 800GbE/1.6TbE PCS

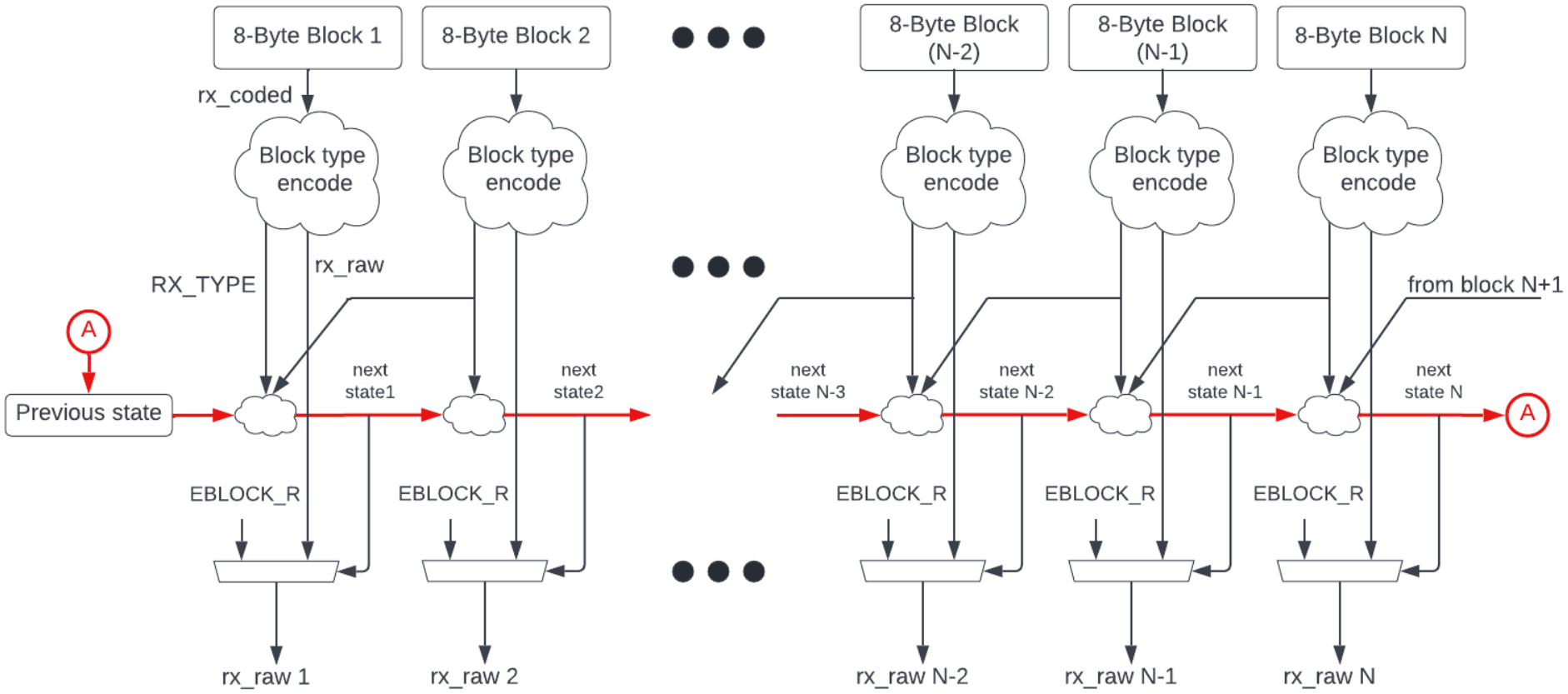
PCS bus width (bytes)	Number of 8-byte blocks in data bus	Frequency required for 800GbE	Frequency required for 1.6TbE
256 bytes	32 blocks	390.625 MHz	781.25 MHz
200	25	500 MHz	1 GHz
128	16	781.25 MHz	1.5625 GHz
100	12	1.042 GHz	2.083 GHz
64	8	1.5625 GHz	3.125 GHz

- To keep the clock frequency under 1GHz, the current state machines would require PCS state to propagate through 25-32 sets of 8-byte blocks.
- This can easily require 64 levels of logic.
- As port speeds increase, and data busses get wider, this path gets even longer.

Typical 64B/66B TX Encoding datapath
--> N Blocks per cycle, Long path through N blocks



Typical 64B/66B RX Decoding datapath
--> N Blocks per cycle, Long path through N blocks



Stateless 64B/66B Encode

- Stateless encode can be done by looking at two contiguous 8-byte blocks, and their “input” values.

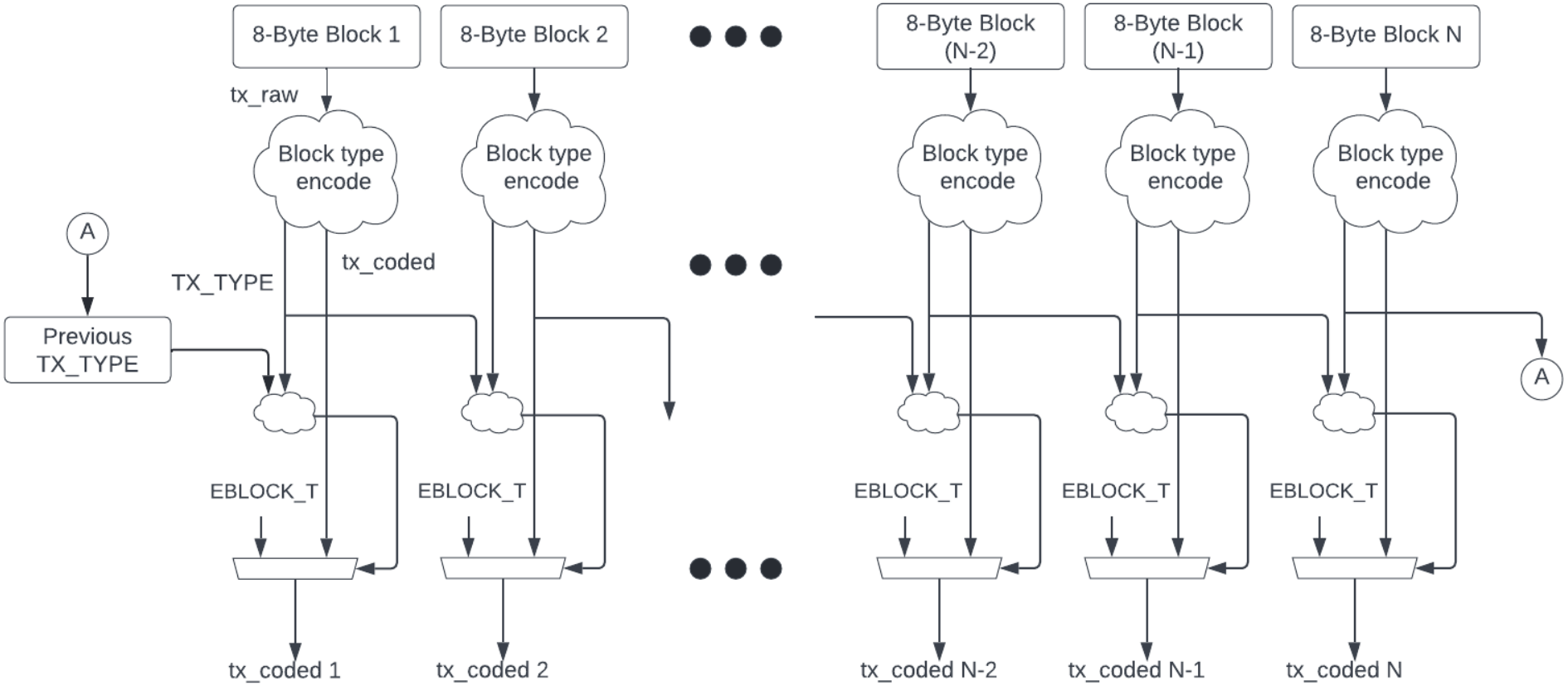
Reset	Current block T_TYPE (tx_raw)	Previous block T_TYPE (tx_raw)	Current Block result (tx_coded)	Current block output type
1	*	*	LBLOCK_T	Local fault
0	S	C + T	ENCODE(tx_raw)	S
0	D	S + D		D
0	T	S + D		T
0	C	C + T + LI		C
0	LI	C + T + LI		LI
0	E	*	EBLOCK_T	Error block
0	S + D + T + C + LI	Anything other than above	EBLOCK_T	Error block

Stateless 64B/66B Decode

- Stateless decode can also be done by looking at two contiguous 8-byte blocks, and their “input” values.

Reset	Current block R_TYPE (rx_coded)	Previous block R_TYPE (rx_coded)	Current Block result (rx_raw)	Current block output type
1	*	*	LBLOCK_R	Local fault
0	S	C + T	DECODE(rx_coded)	S
0	D	S + D		D
0	T	S + D		T
0	C	C + T + LI		C
0	LI	C + T + LI		LI
0	E	*	EBLOCK_R	Error block
0	S + D + T + C + LI	Anything other than above	EBLOCK_R	Error block

Stateless 64B/66B Encode Datapath ---> with “two block sequence check”



Examples of Differences (only w/ Errors)

Input Sequence	TX State Machine Output	RX State Machine Output	Stateless Output
S,D,D,E,C,S	S,D,D,E,C,S	S,D,D,E,C,S	S,D,D,E,E,S
S,D,D,T,E,S,D	S,D,D,T,E,E,D	S,D,D,E,E,E,D	S,D,D,T,E,E,D

Stateless Encode/Decode Summary

- Good packets are still identified by a valid sequence of data:
  - S, D, …, D, T
  - All invalid sequences are recognized and replaced with an EBLOCK
    - Invalid sequence -> replace block with EBLOCK
    - Bad block -> replace block and next block with EBLOCK (covers scrambler propagation)
- Stateless Encode/Decode is compatible with the current state machines.
  - Valid sequences are encoded/decoded exactly the same
  - Any invalid sequence within a packet invalidates the packet with an EBLOCK.
  - Some error sequences are not decoded exactly the same (but this is ok)
- As port speeds increase the “stateful” approach is not a sustainable solution as the PCS data bus gets wider.
  - We must allow for a wide variety of implementations: ASIC, FPGA, etc.

Options for 800GbE and 1.6TbE Port Speeds

- Option A:
  - Replace TX and RX PCS State Diagrams with Stateless truth table description
- Option B:
  - Keep TX and RX PCS State Diagrams, and also allow the Stateless block sequences (fully interoperable)
- Option C:
  - Keep the TX and RX State Diagrams as the only standards compliant implementation

Straw Polls

- For 800GbE 64B/66B Coding, I would favor:
  - (a) Replace PCS State Diagrams with Stateless truth table
  - (b) Keep PCS State Diagrams, and add Stateless truth table alternative
  - (c) Keep PCS State Diagrams only
  - (d) Need more time to study or more information
  - (e) Abstain

- For 1.6TbE 64B/66B Coding, I would favor:
  - (a) Replace PCS State Diagrams with Stateless truth table
  - (b) Keep PCS State Diagrams, and add Stateless truth table alternative
  - (c) Keep PCS State Diagrams only
  - (d) Need more time to study or more information
  - (e) Abstain

Thank You

More Examples of Differences (only w/ Errors)

Input Sequence	TX State Machine Output	RX State Machine Output	Stateless Output
S,D,D,E,C,S	S,D,D,E,C,S	S,D,D,E,C,S	S,D,D,E,E,S
S,D,D,T,E,S,D	S,D,D,T,E,E,D	S,D,D,E,E,E,D	S,D,D,T,E,E,D
S,D,E,T,C,S,D	S,D,E,T,C,S,D	S,D,E,T,C,S,D	S,D,E,E,C,S,D
S,E,D,T,C,S,D	S,E,D,T,C,S,D	S,E,D,T,C,S,D	S,E,E,T,C,S,D
S,D,D,T,T,C,C	S,D,D,T,E,C,C	S,D,D,E,T,C,C	S,D,D,T,E,C,C
S,D,D,T,T,S,D	S,D,D,T,E,E,D	S,D,D,E,T,S,D	S,D,D,T,E,S,D
S,D,D,C,C,D	S,D,D,E,C,E	S,D,D,E,C,E	S,D,D,E,C,E
S,D,D,C,S,D	S,D,D,E,E,D	S,D,D,E,E,D	S,D,D,E,S,D
S,D,D,E,S,D	S,D,D,E,E,D	S,D,D,E,E,D	S,D,D,E,E,D