Stateless 64B/66B Encode/Decode

IEEE P802.3df Logic Ad Hoc

April 2022

Eugene Opsasnick - Broadcom

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. 48-16 and 49-17 (10GBASE-R)

- Not the 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

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 will likely be getting wider

- As the MII bus gets wider, the 64B/66B encode/decode state machines are creating longer logic paths within a single cycle
  - Each 8-byte block must be encoded one at a time and influences the encoding of the following 8-byte block due to the CL 119 state machines.
  - With multiple 8-byte blocks making up the MII bus width, 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 width and frequency required for 800GbE/1.6TbE on MII

MII bus width (bytes)	Number of 8-byte blocks in data bus width	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	12.5	1 GHz	2 GHz
64	8	1.5625 GHz	3.125 GHz
32	4	3.125 GHz	6.25 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 level of logic.
- As port speeds increase, keeping the 64B/66B state machine is not sustainable.

Stateless 64B/66B Encode

- Stateless encode can be done by looking at two contiguous 8-byte blocks, and their “input” values. (Refer to Fig 119-14)

Reset	Current block T_TYPE (tx_raw)	Previous block T_TYPE (tx_raw)	Current Block result (tx_coded)	Current block output type
1	X	X	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 + E + LI		C
0	LI	C + T + E + LI		LI
0	E	X	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. (Refer to Fig. 119-15)

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

Stateless Encode/Decode Highlights

- Good packets are still identified by a valid sequence of data:
  - (T or C), S, D, …, D, T
  - All invalid sequences are recognized and replaced with an EBLOCK
- Stateless Encode/Decode is compatible with the current state machines. They are almost the same.
  - Valid sequences are encoded/decoded exactly the same
  - Any invalid sequence within a packet invalidates the packet with an EBLOCK.
- Some error cases are not encoded the same:
  - For example, TX raw: ..., C, T, S, … is currently encoded as …, C, E, E, ...
  - But is now encoded as …, C, E, S … by the stateless encoder table.

Summary

- The current 64B/66B encode/decode is a serialized process
  - The encoding of each 8-byte block depends on the final encoded/decoded value of the previous block(s) using the state diagrams.
- The stateless approach allows for a parallel encode/decode
  - Each block only needs to reference the “un-encoded” value of the current and previous block.
  - Allows for wider bus implementations
- As port speeds increase the “stateful” approach is not a sustainable solution.

Questions

- Why does Receive State Machine require a “valid” block after a ‘T’ block to decode the ‘T’ block and not replace with EBLOCK?
  - If a valid sequence of “S, D, …, D, T” is received, why does it matter what comes after the T block? Why can’t this packet be deemed valid?
  - Stateless receive decoding can also require the ”following rx_coded block” to be a ”valid” block (S + C + LI) after the T block, but why require this?

- If adopted for 800GbE and 1.6TbE and since the stateless 64B/66B encode/decode is compatible with the CL 119 state diagrams, can the stateless encode/decode also be used as an option for 200GbE and 400GbE port speeds?

Thank You