1000BASE-T1 PHY Encoder Proposal For Gigabit MAC Compatibility

IEEE 802.3bp - Plenary Meeting - March 2014

William Lo, Marvell

Supporters

- Dachin Zeng – Realtek
- Mehmet Tazebay – Broadcom
- Thomas Hogenmueller – Bosch
- Tom Brown - Vitesse
- Xiaofeng Wang – Qualcomm

Primary Objective

- **Insure 1000BASE-T1 PHY encoder can interoperate with existing gigabit Ethernet MACs**

Agenda

- Describe the incompatibility of using 64/65 encoding of Clause 55 when connected to gigabit Ethernet MACs
- Propose an encoder mapping to resolve incompatibility
- Extending to 8n/8n+1 coding as possible alternative options

THE INCOMPATIBILITY ISSUE

Problem with mapping 1G MAC data on Clause 55 64/65

- **Tentatively picked 64/65 encoding**
  - zimmerman_3bp_01_0913.pdf
  - Based on 10GBASE-T in Clause 55 (implied but not explicitly stated)
- **10 Gb/s MAC outputs data 4 bytes at a time on XGMII**
  - Start of packet always on byte 0 enforced by MAC
  - Two XGMII words placed onto 64/65 block
  - Hence start of packet can only be on bytes 0 or 4 of 64/65 block
- **64/65 block encoding of clause 55 can only map idle to start of packet transition on byte 0 and byte 4**
  - Not a problem when MAC outputs XGMII.
- **1000 Mb/s MAC outputs data 1 byte at a time on GMII**
  - Start of packet can start at any position and not just every 4th byte position
- **Compatibility problem when 1000 Mb/s MAC combined with 64/65 encoding of clause 55**
  - Cannot map start of packet in other positions

Clause 55 - 64/65 Bit Encoder Limitation

Input Data	data ctrl header	Block Payload							
Bit Position: / Data Block Format:	0	1							64
$D_0\ D_1\ D_2\ D_3/D_4\ D_5\ D_6\ D_7$	0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
Control Block Formats:		Block							
$C_0\ C_1\ C_2\ C_3/C_4\ C_5\ C_6\ C_7$	1	0x1E	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
$C_0\ C_1\ C_2\ C_3/O_4\ D_5\ D_6\ D_7$	1	0x2D	C_0	C_1	C_2	C_3	O_4	D_5	D_6	D_7
$C_0\ C_1\ C_2\ C_3/S_4\ D_5\ D_6\ D_7$	1	0x33	C_0	C_1	C_2	C_3		D_5	D_6	D_7
$O_0\ D_1\ D_2\ D_3/S_4\ D_5\ D_6\ D_7$	1	0x66	D_1	D_2	D_3	O_0		D_5	D_6	D_7
$O_0\ D_1\ D_2\ D_3/O_4\ D_5\ D_6\ D_7$	1	0x55	D_1	D_2	D_3	O_0	O_4	D_5	D_6	D_7
$S_0\ D_1\ D_2\ D_3/D_4\ D_5\ D_6\ D_7$	1	0x78	D_1	D_2	D_3	D_4		D_5	D_6	D_7
$O_0\ D_1\ D_2\ D_3/C_4\ C_5\ C_6\ C_7$	1	0x4B	D_1	D_2	D_3	O_0	C_4	C_5	C_6	C_7
$T_0\ C_1\ C_2\ C_3/C_4\ C_5\ C_6\ C_7$	1	0x87		C_1	C_2	C_3	C_4	C_5	C_6	C_7
$D_0\ T_1\ C_2\ C_3/C_4\ C_5\ C_6\ C_7$	1	0x99	D_0		C_2	C_3	C_4	C_5	C_6	C_7
$D_0\ D_1\ T_2\ C_3/C_4\ C_5\ C_6\ C_7$	1	0xAA	D_0	D_1		C_3	C_4	C_5	C_6	C_7
$D_0\ D_1\ D_2\ T_3/C_4\ C_5\ C_6\ C_7$	1	0xB4	D_0	D_1	D_2		C_4	C_5	C_6	C_7
$D_0\ D_1\ D_2\ D_3/T_4\ C_5\ C_6\ C_7$	1	0xCC	D_0	D_1	D_2	D_3		C_5	C_6	C_7
$D_0\ D_1\ D_2\ D_3/D_4\ T_5\ C_6\ C_7$	1	0xD2	D_0	D_1	D_2	D_3	D_4		C_6	C_7
$D_0\ D_1\ D_2\ D_3/D_4\ D_5\ T_6\ C_7$	1	0xE1	D_0	D_1	D_2	D_3	D_4	D_5		C_7
$D_0\ D_1\ D_2\ D_3/D_4\ D_5\ D_6\ T_7$	1	0xFF	D_0	D_1	D_2	D_3	D_4	D_5	D_6	

Start of packet on byte 4

Start of packet on byte 0

Cannot start on bytes 1, 2, 3, 5, 6, 7

Limited Block Type Field

Figure 55–9—64B/65B block formats

XGMII vs GMII

XGMII – 4 bytes wide

- Start of packet always on lane 0
- 10 Gb/s MAC uses deficit idle counting rule to force start of packet on lane 0

Lane 0	Lane 1	Lane 2	Lane 3
Byte 4n	Byte 4n + 1	Byte 4n + 2	Byte 4n + 3
Byte 4(n+1)	Byte 4(n+1) + 1	Byte 4(n+1) + 2	Byte 4(n+1) + 3
Byte 4(n+2)	Byte 4(n+2) + 1	Byte 4(n+2) + 2	Byte 4(n+2) + 3

GMII – 1 byte wide

- Start of packet can be any byte
- 1000 Mb/s MAC has no concept of deficit idle counting

Byte n
Byte n + 1
Byte n + 2
Byte n + 3
Byte n + 4
Byte n + 5
Byte n + 6
Byte n + 7
Byte n + 8
Byte n + 9
Byte n + 10
Byte n + 11

Possible workarounds – and why we shouldn't use them

- **Implement deficit idle counting in 1000 Mb/s MAC**
  - Really do not want to touch the MAC when defining the PHY
  - Need a way sync up which byte on the GMII is the 4th byte
- **Implement some form of deficit idle counting in the PHY**
  - Packets will have non-constant latency
  - Will be problematic for MACs that shrink IPG on purpose
- **Add more block types to handle start of packet in any position**
  - Too many permutations
  - Not flexible for future expansion

PROPOSED SOLUTION

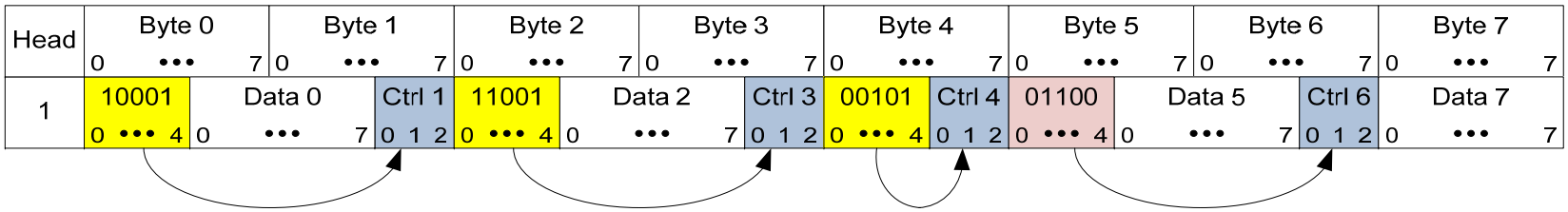
A more flexible way of 64/65 encoding

- Instead of using fixed block types, use pointers instead
- No change if all bytes are data bytes – same as before

0	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7
	0 ••• 7	0 ••• 7	0 ••• 7	0 ••• 7	0 ••• 7	0 ••• 7	0 ••• 7	0 ••• 7

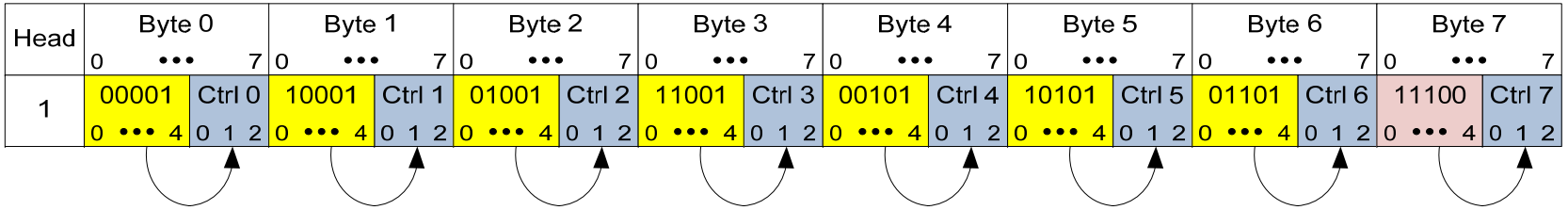
Control Block Encoding

- If byte is control byte use 5 bit pointer + 3 bit control code
- If byte is data use 8 bit data
- Bit 0 to 3 of pointer points to next byte that is a control symbol
- Bit 4 of pointer indicates whether the next control symbol is the final control symbol of the block
  - 0 = final one, 1 = more control symbols
- Example: D/C/D/C/C/D/C/D

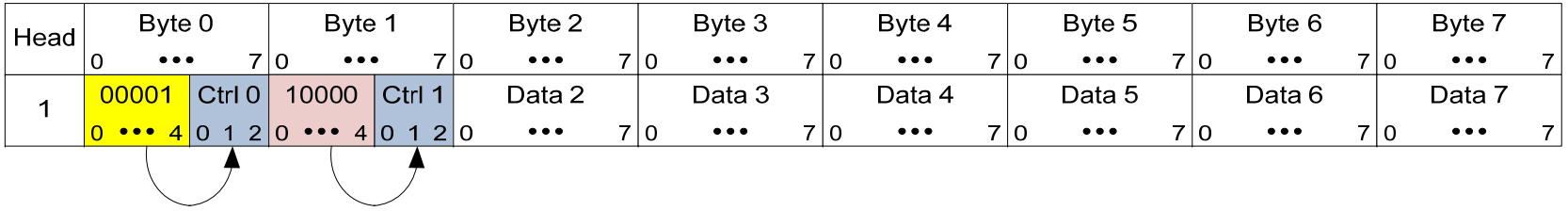


More Examples

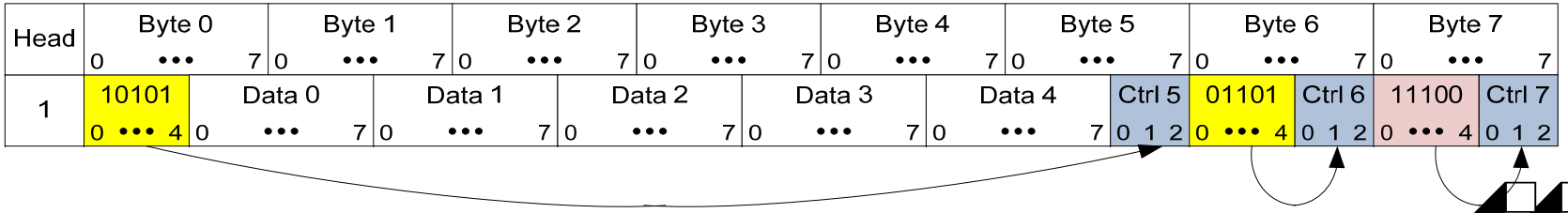
All control codes



Start of packet on byte 2



End of packet on byte 4



MARVELL

GMII Control Code Mapping

- **3 bit control code**

Control Code[0:2]	GMII Transmit	GMII Receive
001	Transmit Error Propagation	Data Reception Error
010	Normal Inter-Frame	Normal Inter-Frame
101	Assert Low Power Idle	Assert Low Power Idle
else	Reserved	Reserved

- **Note that 10G start and terminate control symbols are not needed**
  - **1G MAC does not use start or terminate symbols**

No Change To Clause 55 Scrambling

- **Master**
  - $G(x) = 1 + x^{39} + x^{58}$
- **Slave**
  - $G(x) = 1 + x^{19} + x^{58}$

POSSIBLE EXTENSIONS

Pointers Frees Implementation From 64/65

- Data symbol and control symbol both occupy 8 bits
  - Possible since Block Type field is not used
- Block no longer bounded to be 8 bytes long + header bit
- Can implement 8N / 8N + 1 block where N = 1 to 16
  - i.e. 32/33, 64/65, 120/121, 128/129, etc.
- 128/129 even lower overhead than 64/65
- Can pick N to better fit the 8N / 8N + 1 block into FEC block
- Can extend concept to N = 1 to 32 if we use 6 bit pointers and 2 bit control code for a max of 256/257

Formal Encoder Definition (5 bit pointer)

- **Define:**
  - N = number of GMII bytes encoded into block
  - Bytes numbered n = 0, 1, 2, …, N-1. Byte 0 is the first one presented on GMII.
  - TC[n] = 0 if byte n is data byte on GMII, 1 if byte n is control byte on GMII
  - TC[-1] = 1 by definition
  - TD[n][0:7] = GMII byte n TXD[0:7] if TC[n] = 0
  - TD[n][5:7] = 010 – IPG, 101 – LPI, 001 – TX Error if TC[n] = 1. TD[n][0:4] is undefined.
  - B[0:8N] is the 8N+1 block. Bit 0 transmitted first.
  - OR(p) = Bitwise OR of TC[p:N-1]
  - NEXT(p)[0:3] = bit position of lowest bit in TC[p:N-1] that is a 1. Bit 3 is MSB. Value is invalid if NEXT(p)[4] is 0
  - NEXT(p)[4] = Bitwise AND of TC[p:N-1].
  - NEXT(N)[4] = 0 by definition
- **B[0] = OR(0)**
- **B[8n+1:8n+4] =**
  - TD[n][0:3] – if OR(n) = 0
  - NEXT(n)[0:3] – if OR(n) = 1 AND TC[n-1] = 1
  - TD[n-1][3:6] – if OR(n) = 1 AND TC[n-1] = 0
- **B[8n+5] =**
  - TD[n][4] – if OR(n) = 0
  - NEXT(n+1)[4] – if OR(n) = 1 AND TC[n-1] = 1
  - TD[n-1][7] – if OR(n) = 1 AND TC[n-1] = 0
- **B[8n+6:8n+8] =**
  - TD[n][5:7] – if OR(n) = 0
  - TD[n][5:7] – if OR(n) = 1 AND TC[n] = 1
  - TD[n][0:2] – if OR(n) = 1 AND TC[n] = 0

Proposal

- Use pointer based encoding instead of block type field encoding as formally described in slide 18
- Adopt Clause 55 scrambling
- Leave things at 64/65 for now but be flexible if a different 8n / 8n + 1 works better with the FEC

Summary of Benefits

- Preserves compatibility with existing gigabit Ethernet MACs
- Constant latency through encoder
- Flexibility to change block length to optimize PCS if needed
- Lower bandwidth overhead if a larger block is chosen

THANK YOU