# Multi-Gigabit transceiver

KEK IPNS E-sys
Ryotaro Honda, Yun-Tsung Lai

## TX(RX)USRCLK

Relation between TX(RX)USRCLK and TX(RX)USRCLK2 (from UG476)

- We should use low skew clock resources (**BUFG and BUFR**) to drive TXUSRCLK and TXUSRCLK2.
- TXUSRCLK and TXUSRCLK2's frequencies must be **multiplied or divided from the transmitter reference clock**.

## Review

**GT channel ⇔ High-speed serial transmission's PHY** (**Device which drives the physical layer**)

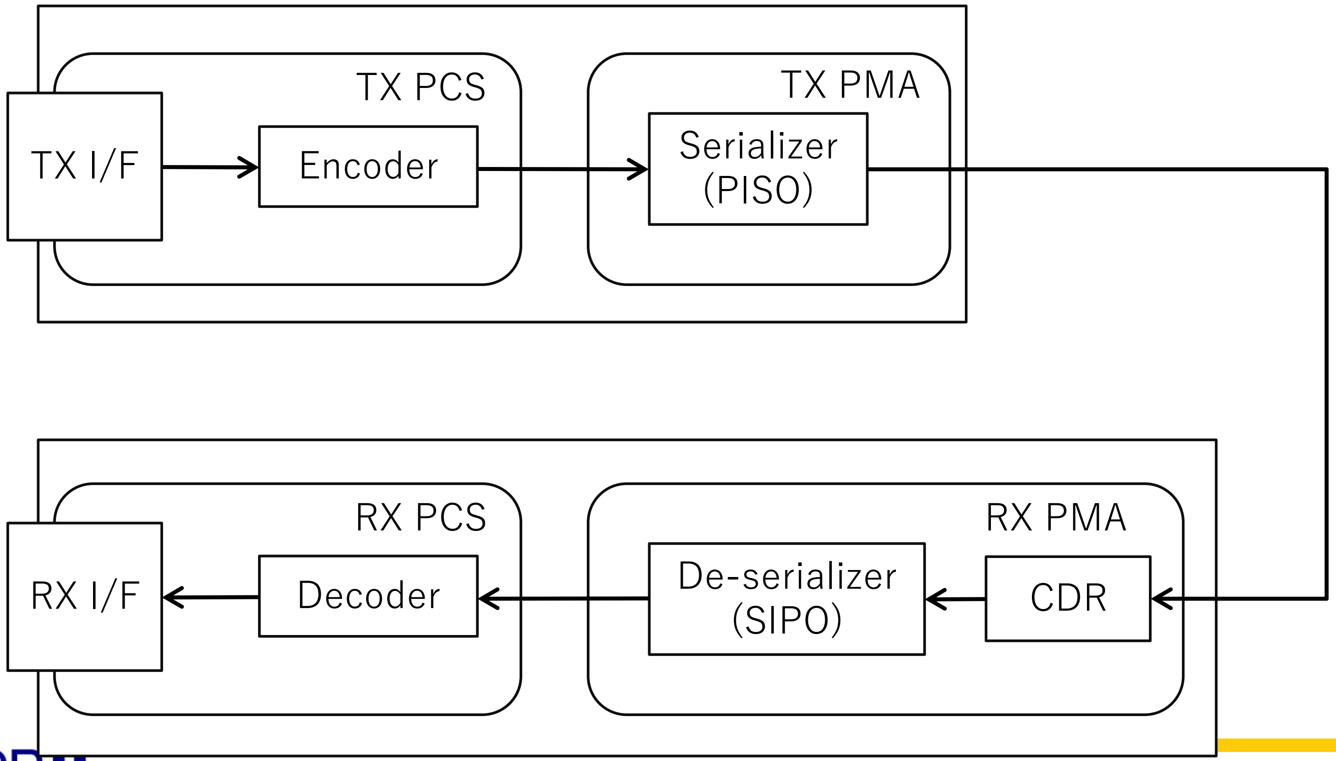
**PCS** (Physical Coding Sublayer)

- Mainly does binary/symbol conversion(8b10b encode/decode, etc)

**PMA** (Physical Medium Attachment) sublayer

- Mainly doss serial/parallel conversion

A rough block diagram of a GT channel
(Many functions are omitted here)



*GTREFCLK is used to determine TX line rate

***SIPO**: Serial In Parallel Out
***PISO**: Parallel In Serial Out

***CDR**: Clock Data Recovery
A circuit to extract (recover) the clock from the data bit stream

Since the GT channel is PHY,
it provides only the lowest-level communication functionalities.

For actual communication
Ethernet frames following Ethernet rules,
Aurora frames following Aurora rules,
It must be assembled and flowed.
(usually IP is utilized)

# Content for this chapter

**GT quad (Physical layer)**

- Transmitter and receiver's internal structure (briefly)
- Clock relation
    - Relation between GTREFCLK and line rate
        - CPLL and QPLL
    - Relation between PCS clock and user clock
    - The role of elastic buffer and operation of clock correction

**Xilinx Aurora 8b10b (transmission protocol)**

- AXI-4 stream
- What to do with this core and how to use it?
- Actual implementation （topic H2)

# Terminology

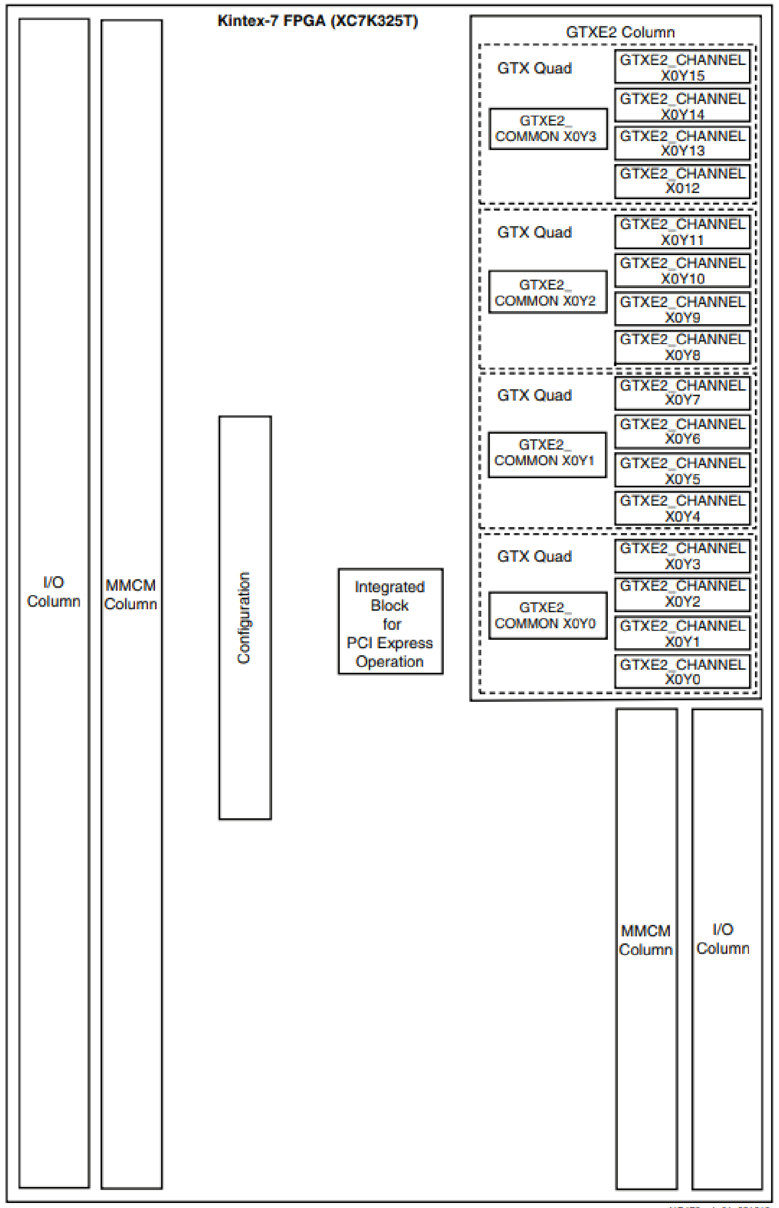


*Figure 1-1:* **GTX Transceiver Inside Kintex-7 XC7K325T FPGA**

## GT(P, X, Y, H)E2 channel

- A block containing a set of 1 transmitter and 1 receiver.

## GT(P…) Quad

- 1 group of 4 channels.
- Corresponds to FPGA banks.
- 1 quad has 2 GTREFCLK inputs.

## GT(P…) Common

- A block to supply a common clock to the channels in the quad. QPLL is placed.

## GT(P, X, Y, H)E2 Column

- The columns where transceiver resources are placed.

## Terminology

**Line rate**

- The bit rate of the PMA sending data to the signal line. Written in bit-per-second (bps)。
    - Not the sending rate of the data which the user wants to send.
- Sometimes written in Transfer-per-second (T/s), but the same.
- If the line rate is 1.25 Gbps with 8b10b conversion, 10-bit symbols are flowing in the signal line at a rate of 1.25 Gbps. 8-bit binary rate is 1.0 Gbps.
    - In serial communication, frames are assembled according to the communication protocol, so the data which the user wants to send (the payload of the frame) is even lower.

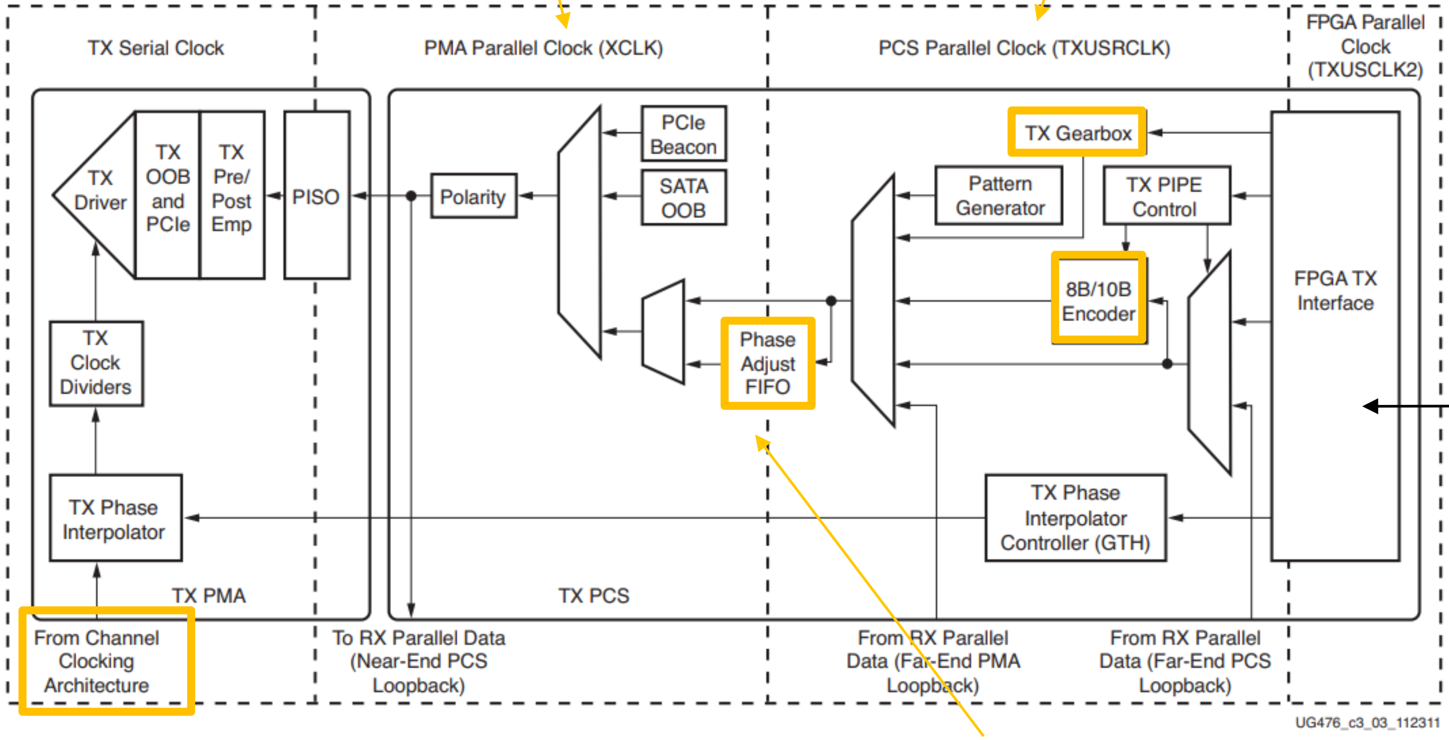
**Lane**

- Communication transmission line. One lane for both TX and RX if full-duplex.

## Inside of GTXE2 channel (transmitter)

XCLK is generated from GT's reference clock (GTREFCLK).

TXUSRCLK(2) is the user circuit clock.
It may be independent of the GT's reference clock.
(The difference between USRCLK and USRCLK2 will be discussed later.)



TX Serial Clock
PMA Parallel Clock (XCLK)
PCS Parallel Clock (TXUSRCLK)
FPGA Parallel Clock (TXUSCLK2)
TX Driver
TX OOB and PCIe
TX Pre/ Post Emp
PISO
Polarity
PCIe Beacon
SATA OOB
Phase Adjust FIFO
TX Gearbox
Pattern Generator
TX PIPE Control
8B/10B Encoder
FPGA TX Interface
TX Clock Dividers
TX Phase Interpolator
TX Phase Interpolator Controller (GTH)
TX PMA
TX PCS
From Channel Clocking Architecture
To RX Parallel Data (Near-End PCS Loopback)
From RX Parallel Data (Far-End PMA Loopback)
From RX Parallel Data (Far-End PCS Loopback)
UG476_c3_03_112311

**8B/10B encoder**
Gives 10-bit symbol.

**TX gearbox**
Conversion other than 8b10b, such as 64b66b.

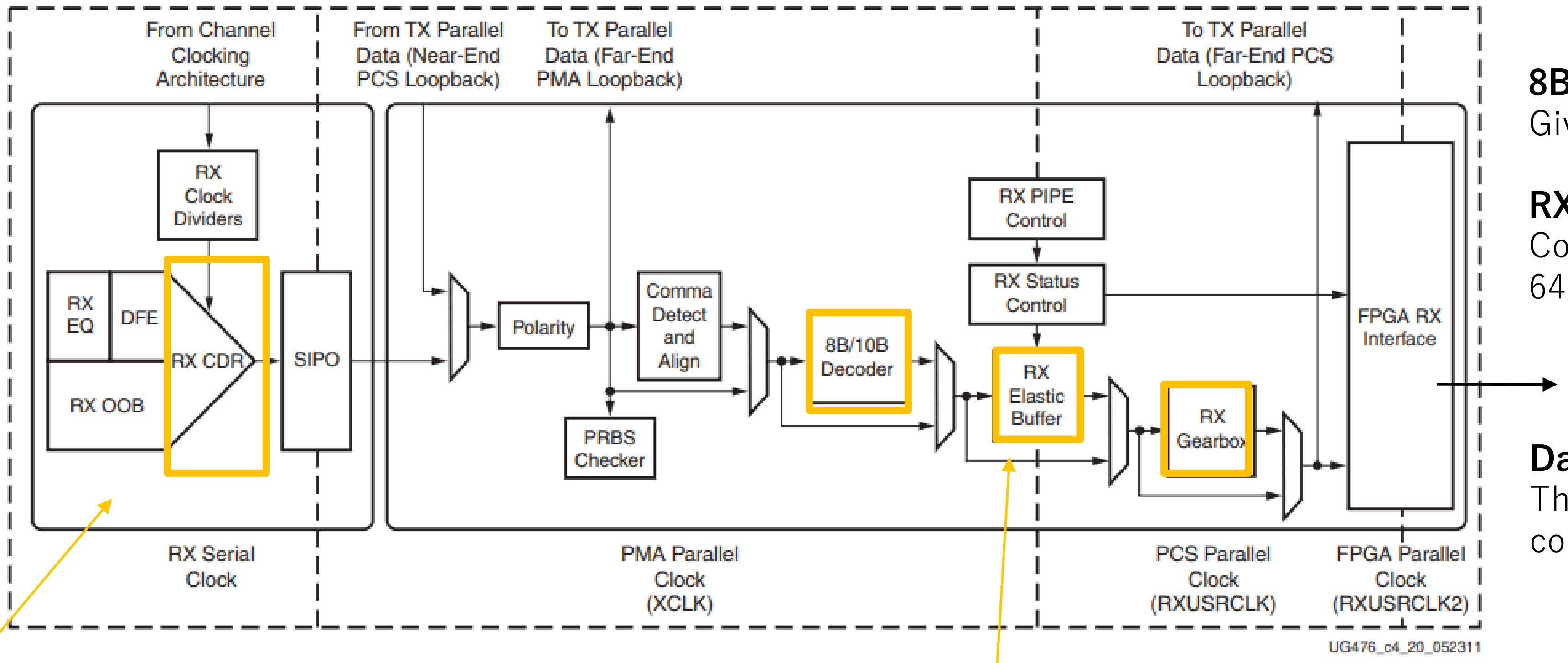
**Data input from upstream circuit**
When generating a IP core for Aurora or Ethernet, the circuit that controls the communication protocol is connected here.
The user circuit should not be directly connected.

**The clock which determines the line rate**
(Clock-related circuits exist independently)
A clock that drives the transmission line based on the GT reference clock.
XCLK also branches from here by default.

**Phase Adjust FIFO: TX buffer**
A buffer that absorbs the phase difference between two clocks.
Bypassing this buffer is an advanced usage and it is not covered in this exercise.

# Inside of GTXE2 channel (receiver)

There are also two domains at RX side: XCLK and RXUSRCLK.



From Channel Clocking Architecture
From TX Parallel Data (Near-End PCS Loopback)
To TX Parallel Data (Far-End PMA Loopback)
To TX Parallel Data (Far-End PCS Loopback)
RX Clock Dividers
RX EQ
DFE
RX CDR
RX OOB
SIPO
Polarity
Comma Detect and Align
PRBS Checker
8B/10B Decoder
RX PIPE Control
RX Status Control
RX Elastic Buffer
RX Gearbox
FPGA RX Interface
RX Serial Clock
PMA Parallel Clock (XCLK)
PCS Parallel Clock (RXUSRCLK)
FPGA Parallel Clock (RXUSRCLK2)
UG476_c4_20_052311

**8B/10B decoder**
Gives 8-bit symbol

**RX gearbox**
Conversion other than 8b10b, such as 64b66b.

**Data output to downstream circuit**
The user circuit should not be directly connected just like the case of TX.

**Clock Data Recovery (CDR)**
It recovers the clock from the received signal and it separates the data.
The frequency of the recovered clock depends on the reference clock on the TX side.
XCLK also branches from here by default.

**RX elastic buffer**
A buffer to absorb the small frequency differences and phase differences between two clocks.
It is assumed that XCLK and RXUSRCLK are different.

# Relation between GTREFCLK and line rate

**Point**

- The RX side uses the CDR circuit to restore the clock, so the line rate is determined by the TX side.
- GTREFCLK is the original clock driving the transmission line.

For GTX, GTREFCLK has 2 input per bank (quad)

Requires a dedicated primitive called IBUFDS_GTE2

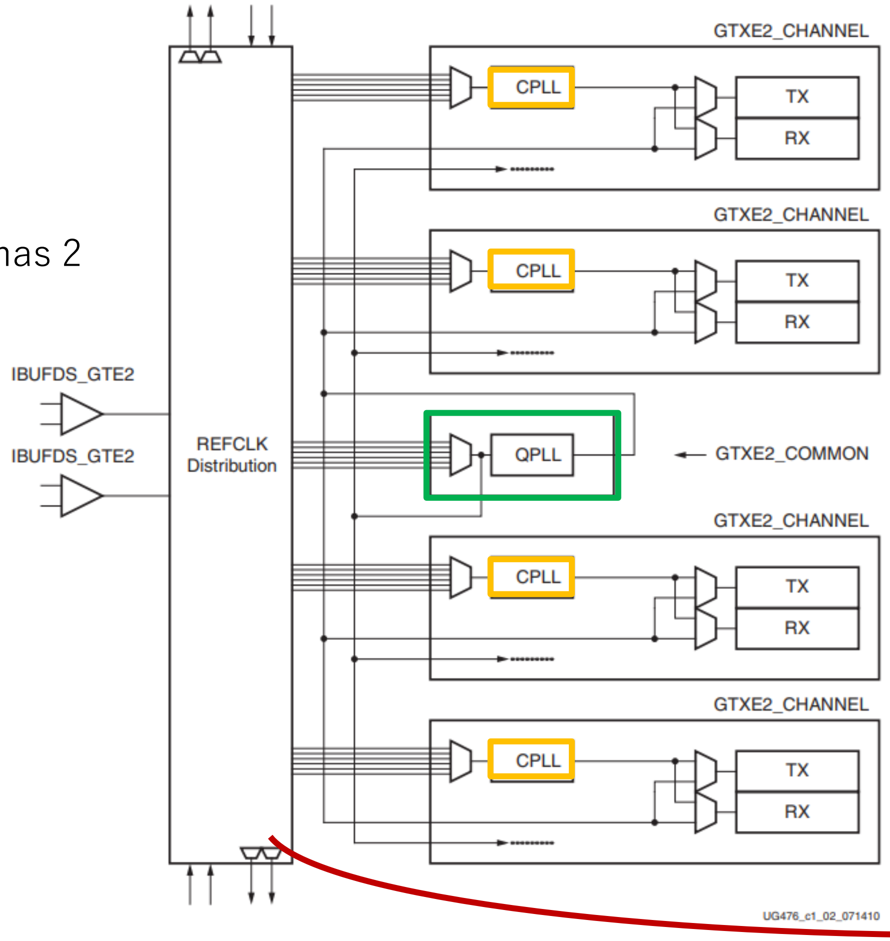


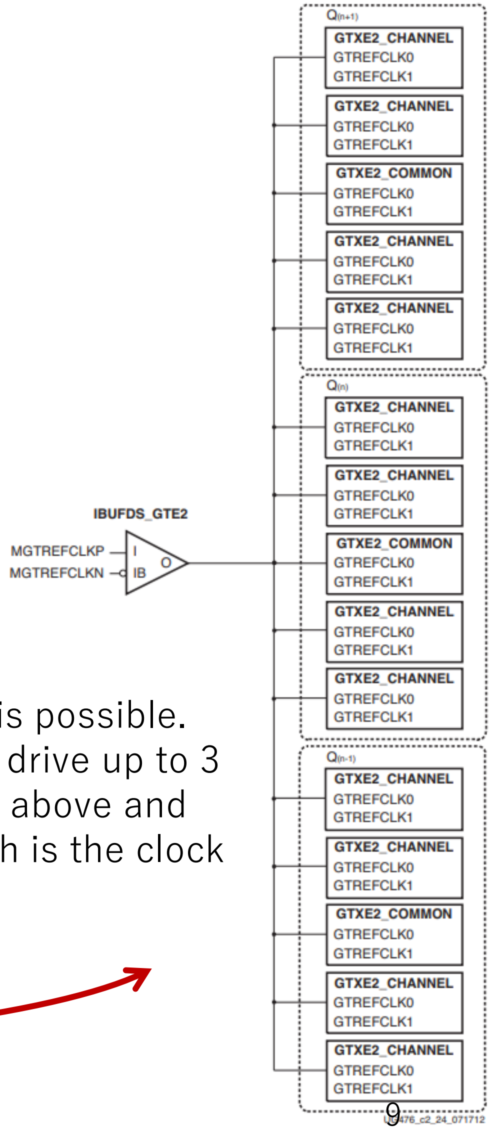
*Figure 1-2:* **GTX Transceiver Quad Configuration**

GTREFCLK input to:

- **Channel PLL (CPLL)**
- **Quad PLL (QPLL)**

within the quad

Vertical clock line is possible. However, it can only drive up to 3 quads in total, one above and below the quad which is the clock source.



## Relation between GTREFCLK and line rate (CPLL)

**Channel PLL**

- The PLL which exists in each channel
- With CPLL, the line rate can be determined for each channel.
- The CPLL setting is done by the wizard when the IP is generated, and it is not common to change it.
  - Manually changing parameters set by IP is an advanced usage (covered in an advanced task).

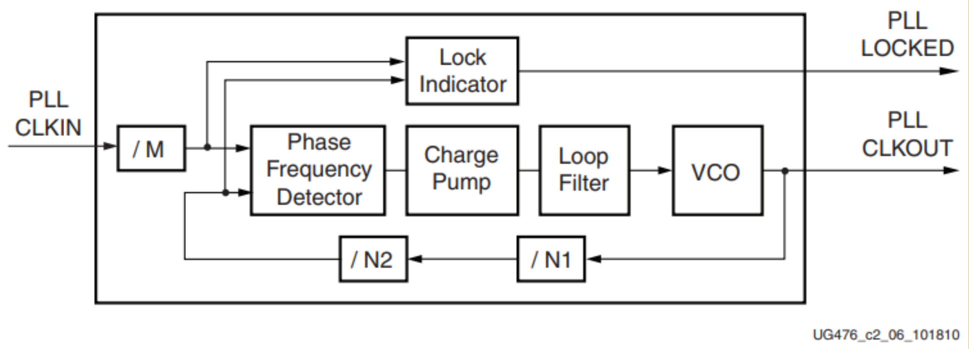


*Figure 2-10:* **CPLL Block Diagram**

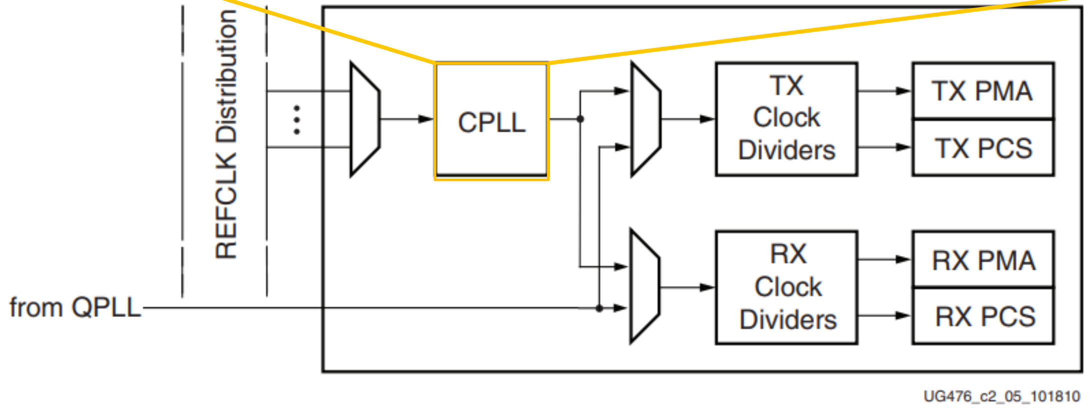


*Figure 2-9:* **Internal Channel Clocking Architecture**

$$f_{PLLClkout} = f_{PLLClkin} \times \frac{N1 \times N2}{M}$$ *Equation 2-1*

$$\boxed{f_{LineRate}} = \frac{f_{PLLClkout} \times 2}{D}$$ *Equation 2-2*

The term "Line rate" appears here

*Table 2-8:* **CPLL Divider Settings**

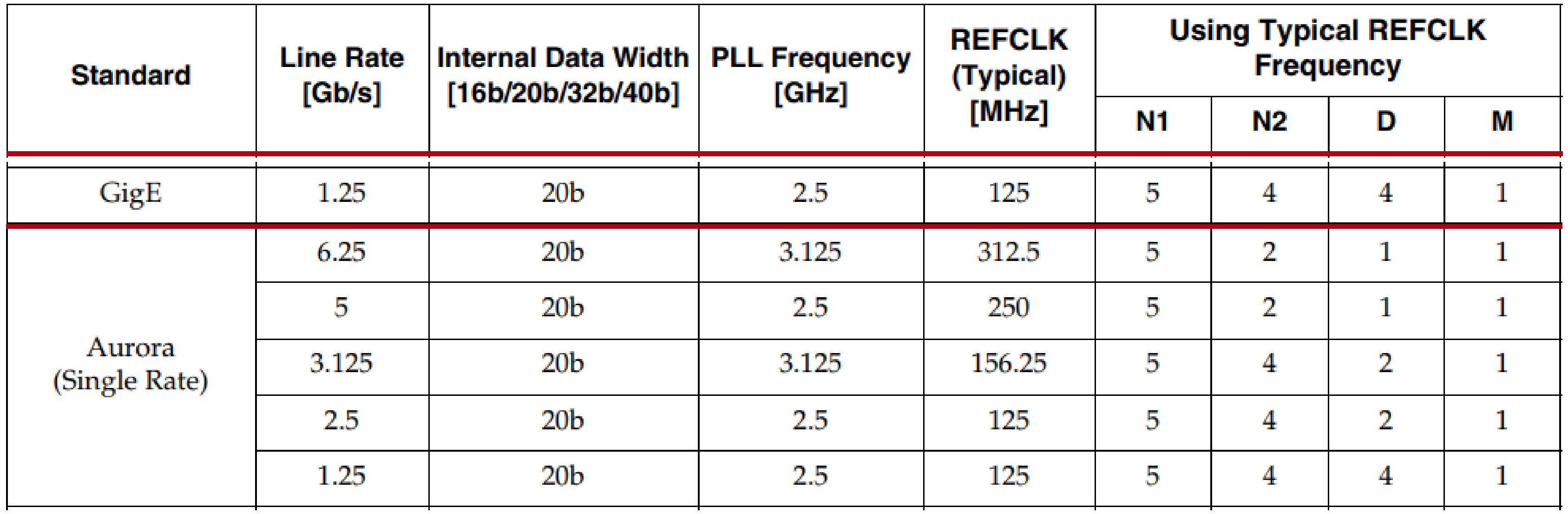
| Factor | Attribute | Valid Settings |
|---|---|---|
| M | CPLL_REFCLK_DIV | 1, 2 |
| N2 | CPLL_FBDIV | 1, 2, 3, 4, 5 |
| N1 | CPLL_FBDIV_45 | 4, 5 |
| D | RXOUT_DIV<br>TXOUT_DIV | 1, 2, 4, 8, 16[1] |

1. TX/RXOUT_DIV = 16 is not supported when using CPLL.

## Relation between GTREFCLK and line rate (CPLL)

The recommended GTREFCLK frequency and CPLL settings are determined for each communication standard.
Here is an example below.

*Table 2-11:* **CPLL Divider Settings for Common Protocols**



| Standard | Line Rate [Gb/s] | Internal Data Width [16b/20b/32b/40b] | PLL Frequency [GHz] | REFCLK (Typical) [MHz] | Using Typical REFCLK Frequency | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | N1 | N2 | D | M |
| GigE | 1.25 | 20b | 2.5 | 125 | 5 | 4 | 4 | 1 |
| Aurora (Single Rate) | 6.25 | 20b | 3.125 | 312.5 | 5 | 2 | 1 | 1 |
| | 5 | 20b | 2.5 | 250 | 5 | 2 | 1 | 1 |
| | 3.125 | 20b | 3.125 | 156.25 | 5 | 4 | 2 | 1 |
| | 2.5 | 20b | 2.5 | 125 | 5 | 4 | 2 | 1 |
| | 1.25 | 20b | 2.5 | 125 | 5 | 4 | 4 | 1 |

UG476

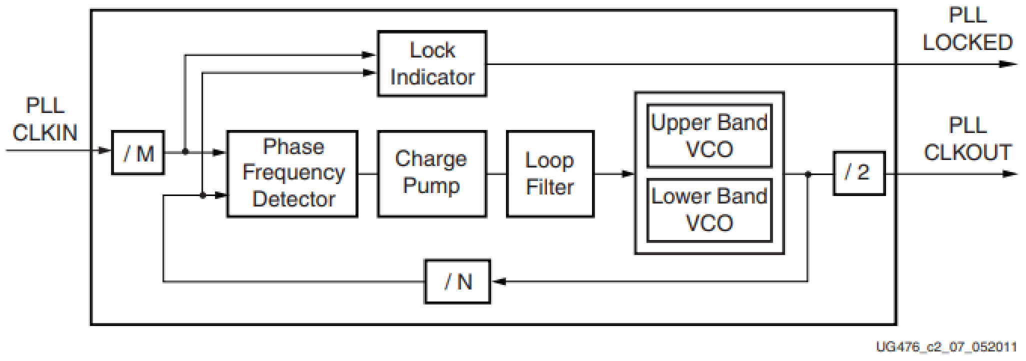
Only the line rates up to 6.25 Gb/s are listed.
GTX should support up to 10 Gb/s and above.
To achieve higher line rates, need to use **QPLL**.

Fomr UG476:
The CPLL in the GTX transceiver has a nominal operating range between **1.6 GHz to 3.3 GHz**. The CPLL in the GTH transceiver has a nominal operating range between **1.6 GHz to 5.16 GHz**.

# Relation between GTREFCLK and line rate (QPLL)

**Quad PLL**

- 1 GT quad has 1 QPLL. It is wrapped with GTXE2 common.
- It is used when the the transceiver has a line rate exceeding 6.6 Gbps.
- 1 per quad, so it cannot be used to change the line rate for each channel.



$$f_{PLLClkout} = f_{PLLClkin} \times \frac{N}{M \times 2}$$ *Equation 2-3*

$$f_{LineRate} = \frac{f_{PLLClkout} \times 2}{D}$$ *Equation 2-4*

*Table 2-12:* **QPLL Nominal Operating Range**

| Transceiver | Frequency (GHz) | |
|---|---|---|
| GTX | Lower Band | 5.93–8.0 |
| | Upper Band | 9.8–12.5 |
| GTH | 8.0–13.1 | |

*Table 2-13:* **QPLL Divider Settings**

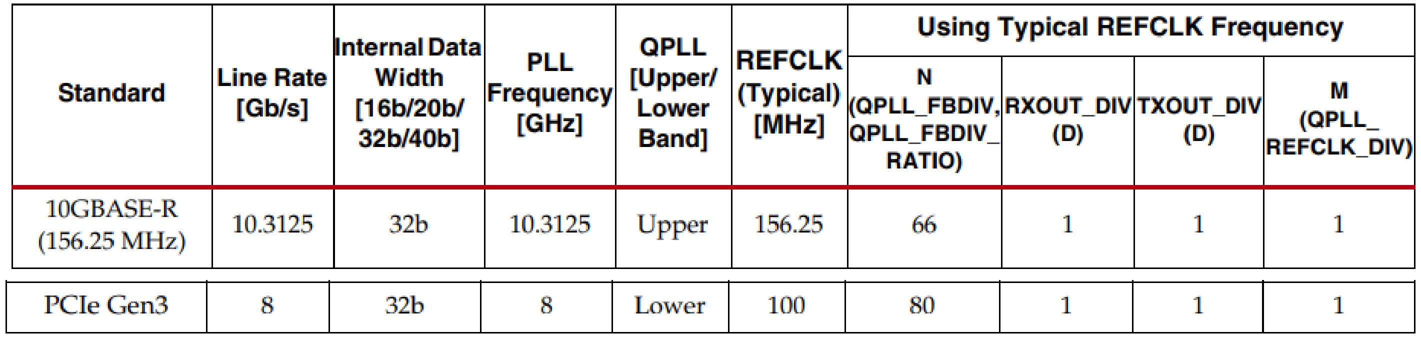
| Factor | Attribute | Valid Settings |
|---|---|---|
| M | QPLL_REFCLK_DIV | 1, 2, 3, 4 |
| N | QPLL_FBDIV<br>QPLL_FBDIV_RATIO | 16, 20, 32, 40, 64, 66, 80, 100<br>(see Table 2-16) |
| D | RXOUT_DIV<br>TXOUT_DIV | 1, 2, 4, 8, 16 |

Setting parameters manually is not common just like using CPLL.

# Relation between GTREFCLK and line rate (QPLL)

It is used in high-speed communication standards with around 10 Gbps.

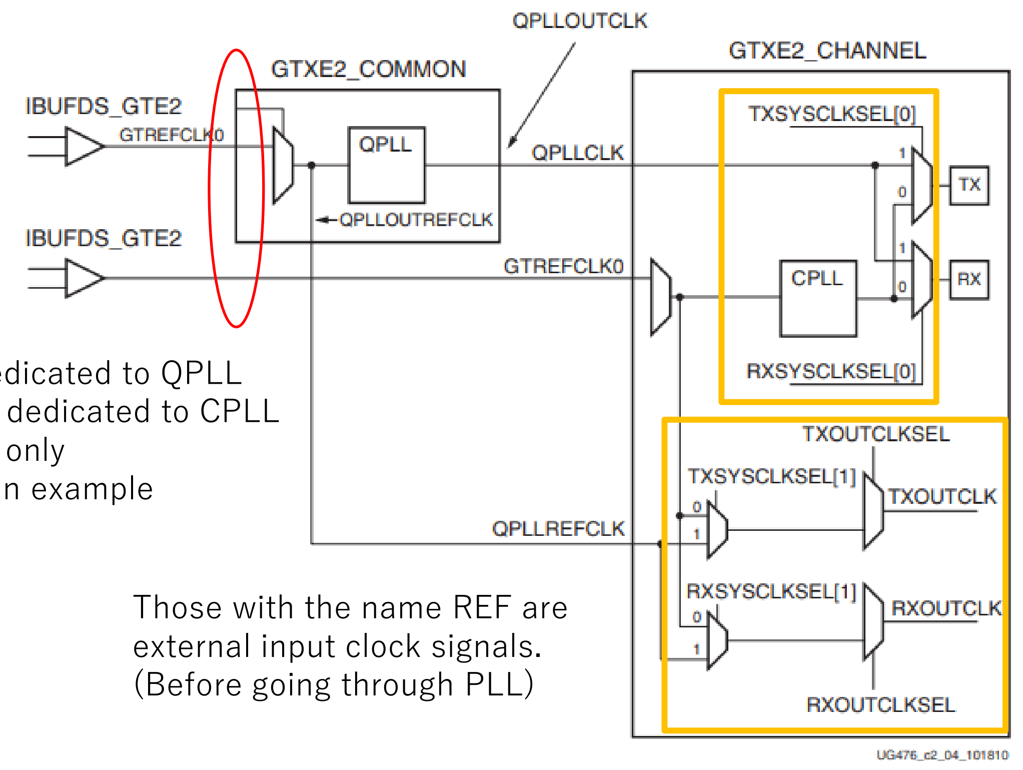
*Table 2-17:* **QPLL Divider Settings for Common Protocols**



| Standard | Line Rate [Gb/s] | Internal Data Width [16b/20b/32b/40b] | PLL Frequency [GHz] | QPLL [Upper/Lower Band] | REFCLK (Typical) [MHz] | Using Typical REFCLK Frequency | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | N (QPLL_FBDIV, QPLL_FBDIV_RATIO) | RXOUT_DIV (D) | TXOUT_DIV (D) | M (QPLL_REFCLK_DIV) |
| 10GBASE-R (156.25 MHz) | 10.3125 | 32b | 10.3125 | Upper | 156.25 | 66 | 1 | 1 | 1 |
| PCIe Gen3 | 8 | 32b | 8 | Lower | 100 | 80 | 1 | 1 | 1 |

## How the PLL's output reaches the channel

A example diagram from UG476



One side dedicated to QPLL
One looks like dedicated to CPLL only
Just an example

Those with the name REF are external input clock signals.
(Before going through PLL)

**TXSYSCLKSEL[0-1]**
**RXSYSCLKSEL[0-1]**
Select CPLL or QPLL

- [0]: Wiring toward PCS/PMA
- [1]: Wiring toward TX(RX)OUTCLK
- [0][1] should contain the same number

**TXOUTCLKSEL**
**RXOUTCLKSEL**

- Will be described later
- Decide which signal to be sent out from GTXE2 channel as TX(RX)OUTCLK. Important.

**Point**
TX(RX)SYSCLKSEL can be estimated from the user's desired line rate.
There are multiple choices for TX(RX)OUTCLKSEL, and it's hard to guess.
It is better to check after generating with IP.

# How the PLL's output reaches the channel

## Transmitter

CPLLREFCLKSE

Unless something special is donw, GTREFCLK0 will be selected.

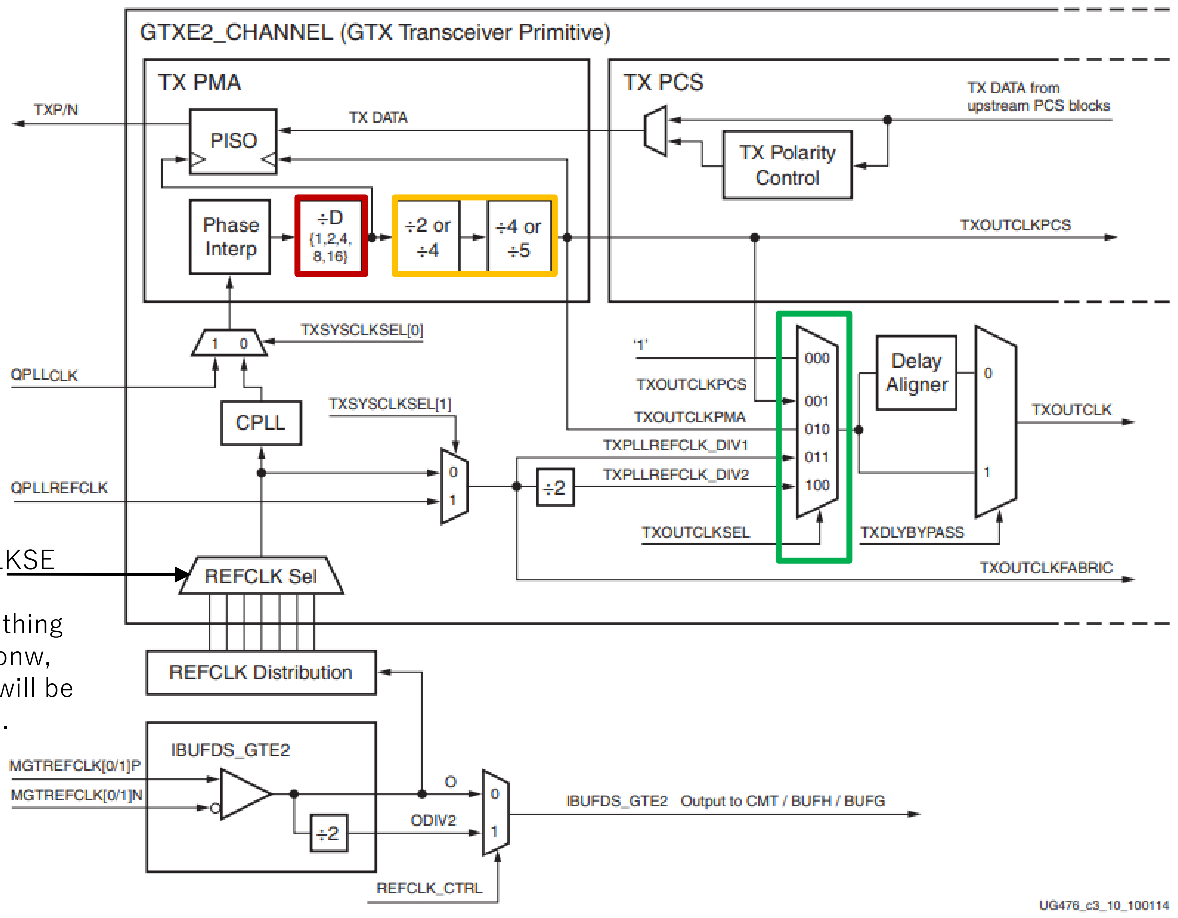


*Figure 3-28:* **TX Serial and Parallel Clock Divider**

**÷D**

PISO is set to the specified rate.
For recommended values, please refer to the CPLL parameter table.

**÷2 or ÷4**

Determined by the value of TX_INT_DATAWIDTH, which determines the internal data width.

- 0 (2-byte width): ÷2
- 1 (4-tybe width): ÷4

1 is always selected for rates above 6.6 Gbps.

**÷4 or ÷5**

Determined by the value of TX_DATA_WIDTH, which determines the data width of the TX (RX) interface

- 4, if 16, 32, 64-bit
- 5, if 20, 40, 80-bit

**TXOUTCLKSEL**

- Decide what to use for the TXOUTCLK signal. Depends on application.
- Since it is the source of TXUSRCLK (will be described later), it turns around and becomes the PCS clock.

# How the PLL's output reaches the channel

Transmitter

When generating PCS/PMA with IP for GbE

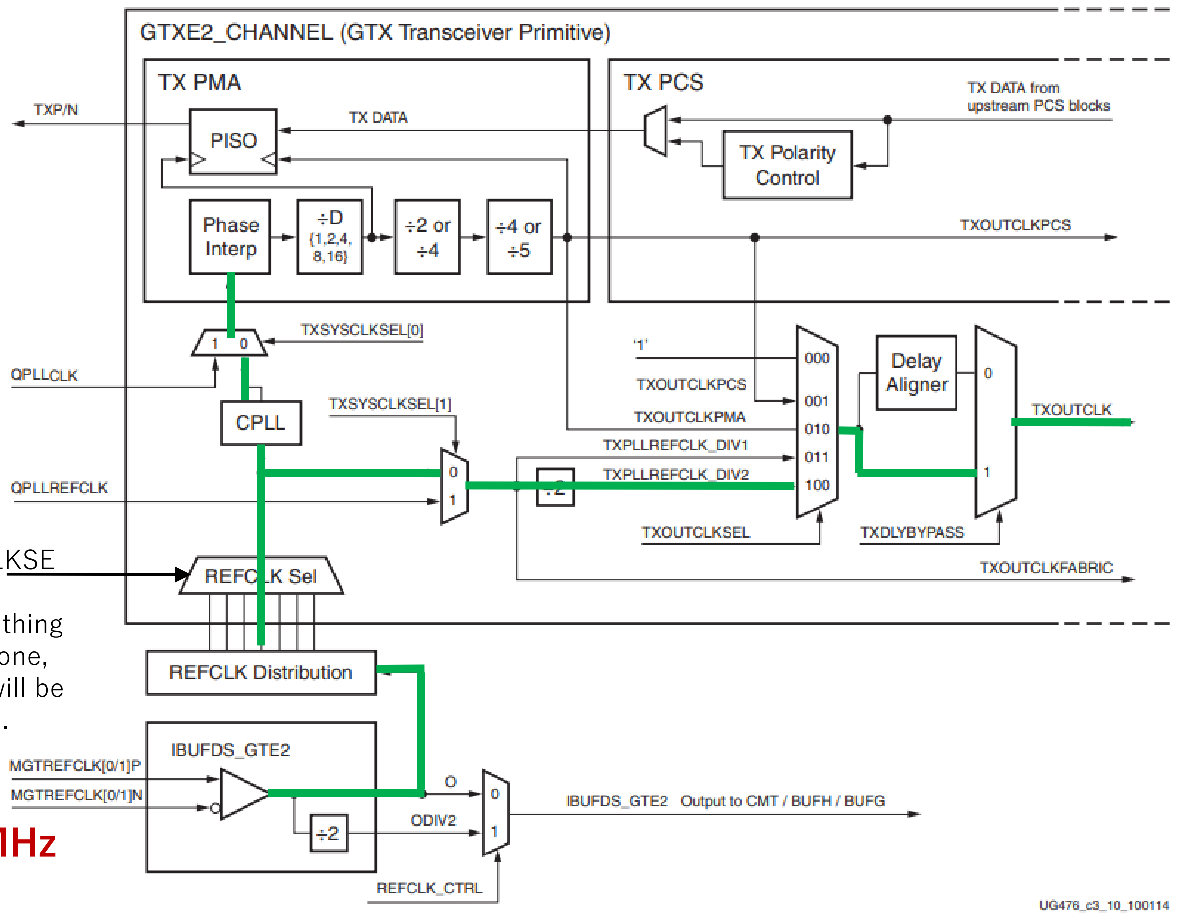


*Figure 3-28:* **TX Serial and Parallel Clock Divider**

CPLLREFCLKSE

Unless something special is done, TREFCLK0 will be selected.

125 MHz

What is the frequency of TXOUTCLK?

# How the PLL's output reaches the channel

Transmitter

When an IP is generated for Aurora8b10b, linre rate = 1.25 Gbps, 2-byte data width

CPLLREFCLKSE

Unless something special is done, TREFCLK0 will be selected.

156.25 MHz

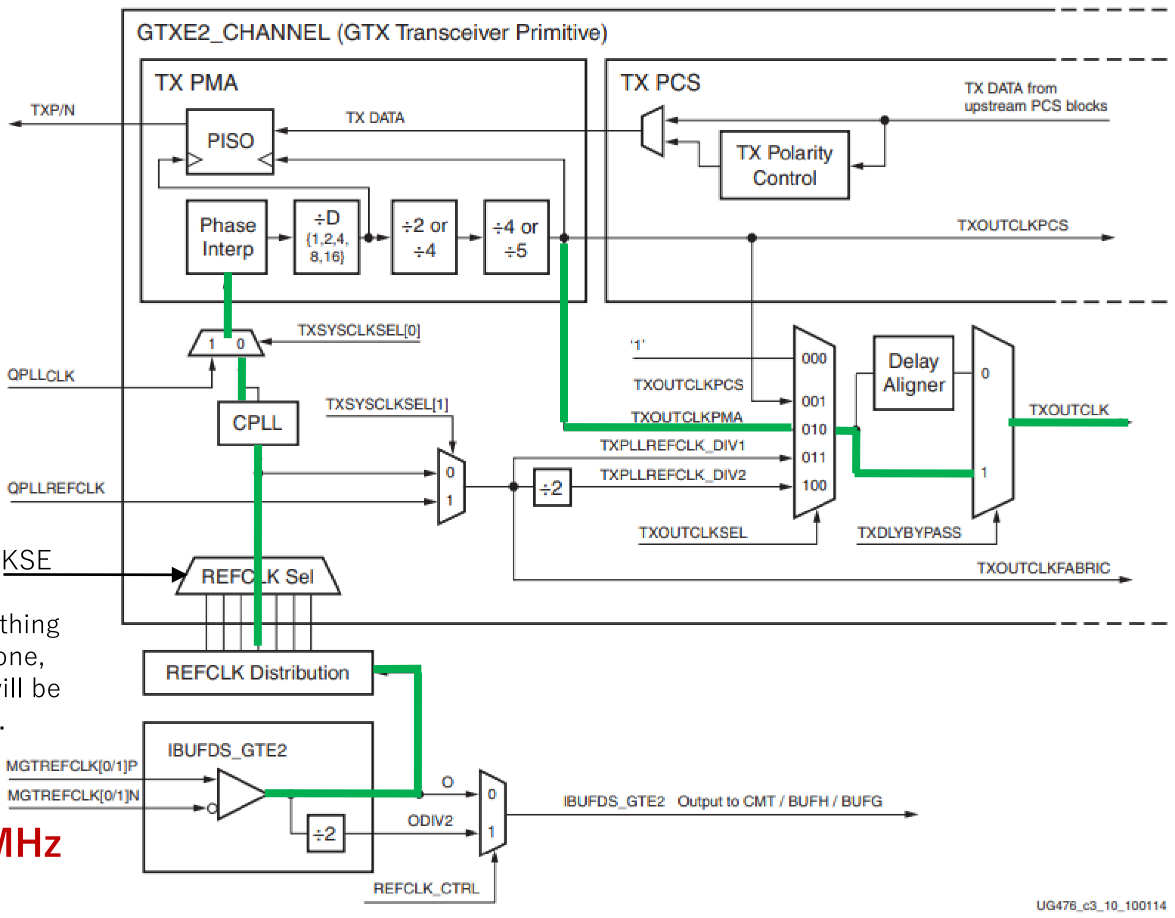


*Figure 3-28:* **TX Serial and Parallel Clock Divider**

CPLL parameter
- M: 1
- N1: 4
- N2: 4
- D: 4

TX_DATA_WIDTH
- 20

TX_INT_DATAWIDTH
- 0

What is the frequency of TXOUTCLK?

# How the PLL's output reaches the channel

Transmitter　　When generating PCS/PMA for 10GbE

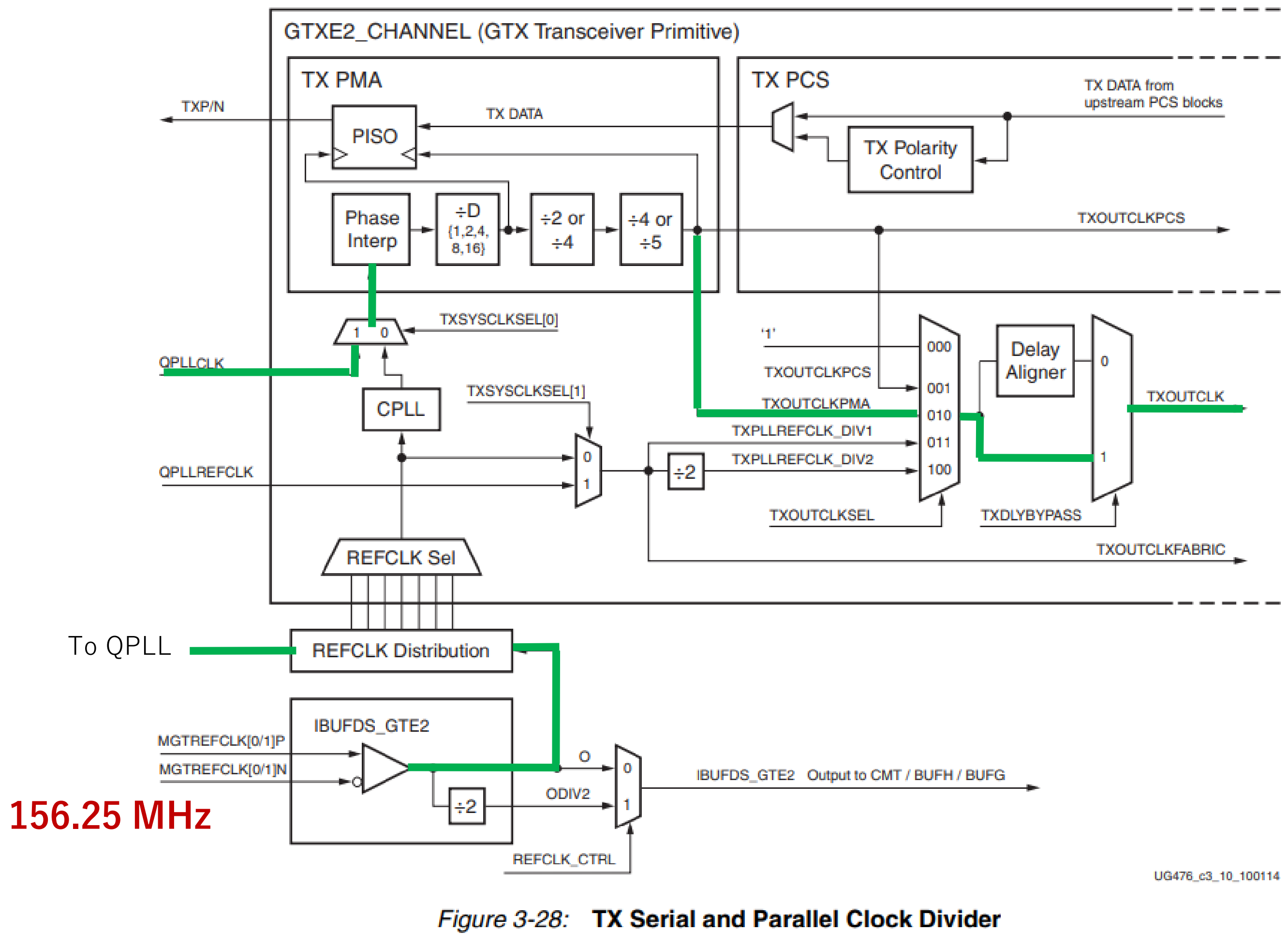


*Figure 3-28:* **TX Serial and Parallel Clock Divider**

QPLL parameter
- M: 1
- N: 66
- D: 1

TX_DATA_WIDTH
- 32

TX_INT_DATAWIDTH
- 1

What is the frequency of TXOUTCLK?

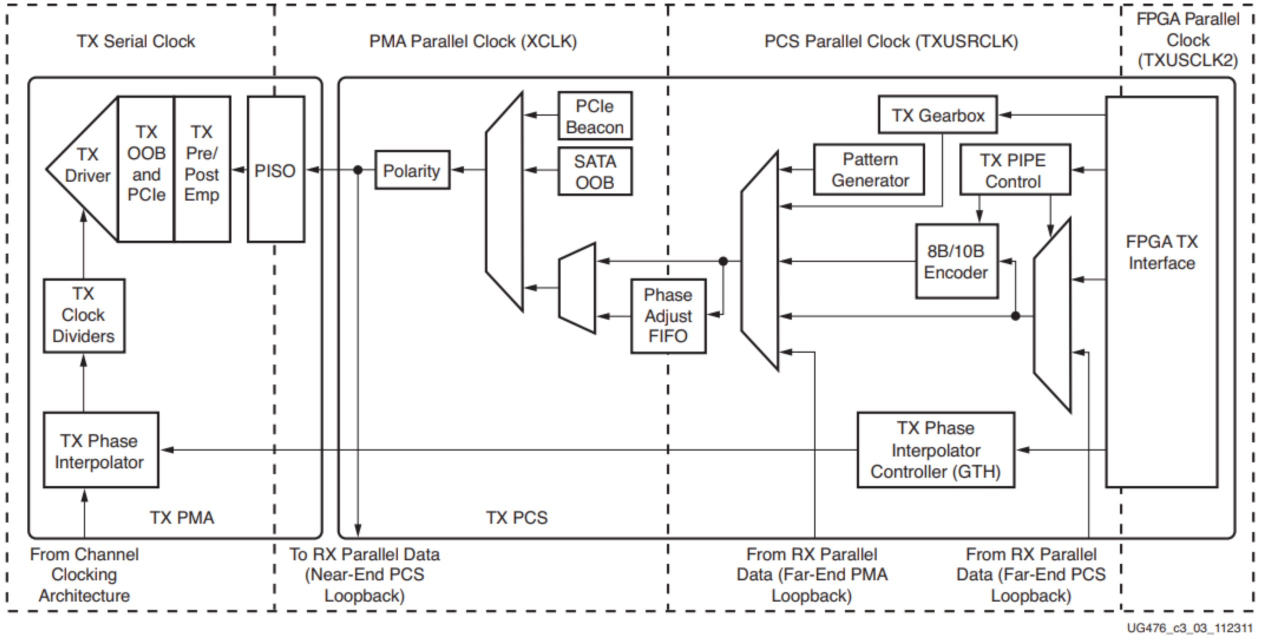
## Summary so far

The required transmitter line rate is obtained using CPLL or QPLL from the GTREFCLK signal.
The frequency of TXOUTCLK is determined.
At this point, the frequency of PCS XCLK before branching is also determined.

Since the line rate of the transmitter has been determined, the frequency of the recovered clock on the receiver side is determined.
Again, the PCS XCLK frequency on the RX side is determined here as well.

Therefore, what is the frequency of TX (RX) USRCLK, which is another clock domain of PCS?

## TX(RX)USRCLK


TX Serial Clock
PMA Parallel Clock (XCLK)
PCS Parallel Clock (TXUSRCLK)
FPGA Parallel Clock (TXUSCLK2)
TX Driver
TX OOB and PCIe
TX Pre/ Post Emp
PISO
Polarity
PCIe Beacon
SATA OOB
Phase Adjust FIFO
TX Gearbox
Pattern Generator
TX PIPE Control
8B/10B Encoder
FPGA TX Interface
TX Clock Dividers
TX Phase Interpolator
TX Phase Interpolator Controller (GTH)
TX PMA
TX PCS
From Channel Clocking Architecture
To RX Parallel Data (Near-End PCS Loopback)
From RX Parallel Data (Far-End PMA Loopback)
From RX Parallel Data (Far-End PCS Loopback)
UG476_c3_03_112311

**TX(RX)USRCLK**
- PCS's internal clock

**TX(RX)USRCLK2**
- TX（RX） interface clock

The frequency of USRCLK is uniquely determined

$$TXUSRCLK\ Rate = \frac{Line\ Rate}{Internal\ Datapath\ Width} \quad \text{式 3-1}$$

In the other words, similar to XCLK。
USRCLK2 is？

*Table 3-3:* **TXUSRCLK2 Frequency Relationship to TXUSRCLK**

| FPGA Interface Width | TX_DATA_WIDTH | TX_INT_DATAWIDTH | TXUSRCLK2 Frequency |
|---|---|---|---|
| 2-Byte | 16, 20 | 0 | $F_{TXUSRCLK2} = F_{TXUSRCLK}$ |
| 4-Byte | 32, 40 | 0 | $F_{TXUSRCLK2} = F_{TXUSRCLK}/2$ |
| 4-Byte | 32, 40 | 1 | $F_{TXUSRCLK2} = F_{TXUSRCLK}$ |
| 8-Byte | 64, 80 | 1 | $F_{TXUSRCLK2} = F_{TXUSRCLK}/2$ |

USRCLK2 is the same as, or half of USRCLK

## TX(RX)USRCLK

This is a summary of the above.
In other words, XCLK and TXUSRCLK are the same in the transmitter.
And, if multiple GTXE2 channels are implemented, the representative TXOUTCLK should be used.
(Of course, it is limited to the case with the same communication standard)

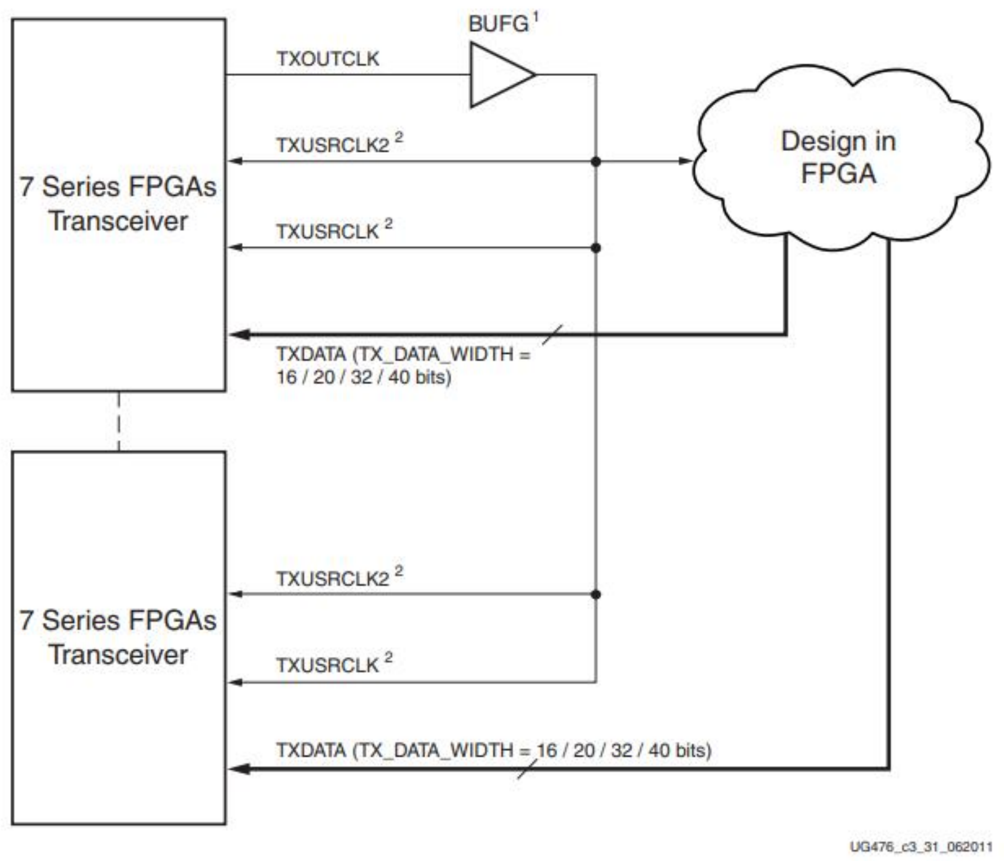

BUFG 1
TXOUTCLK
TXUSRCLK2 2
TXUSRCLK 2
7 Series FPGAs Transceiver
Design in FPGA
TXDATA (TX_DATA_WIDTH = 16 / 20 / 32 / 40 bits)
TXUSRCLK2 2
TXUSRCLK 2
7 Series FPGAs Transceiver
TXDATA (TX_DATA_WIDTH = 16 / 20 / 32 / 40 bits)
UG476_c3_31_062011

*Figure 3-3:* **Multiple Lanes—TXOUTCLK Drives TXUSRCLK2 (2-Byte or 4-Byte Mode)**

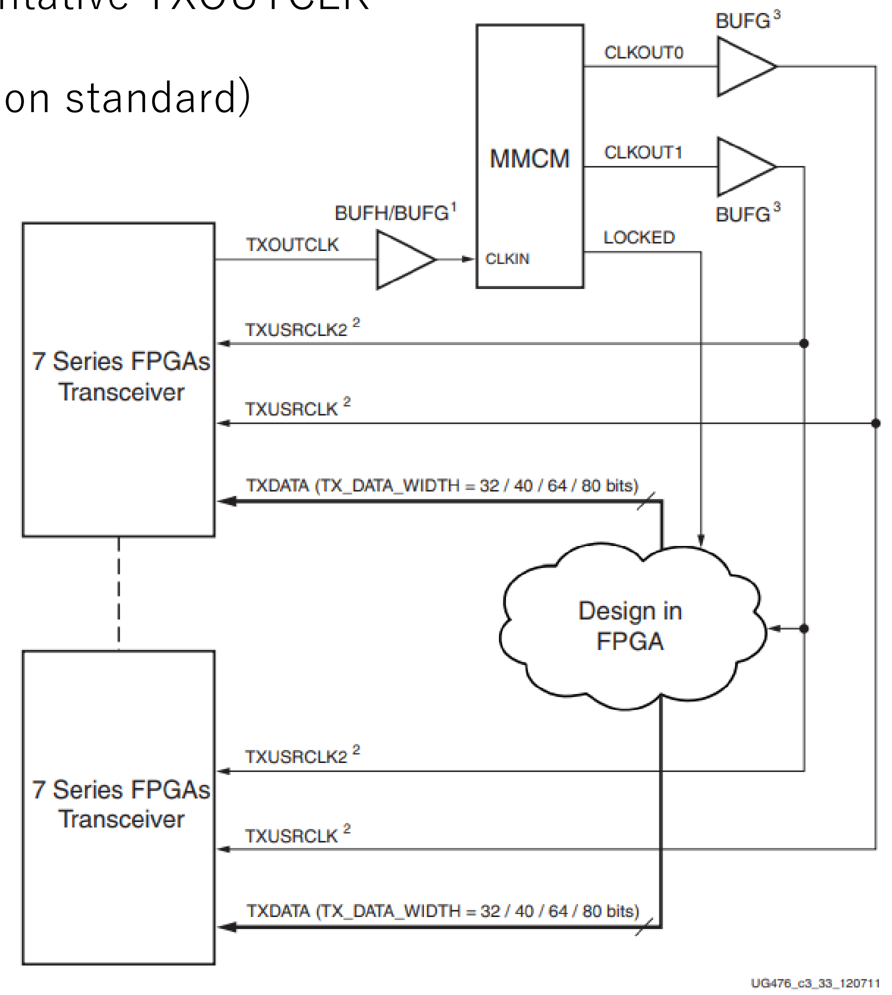
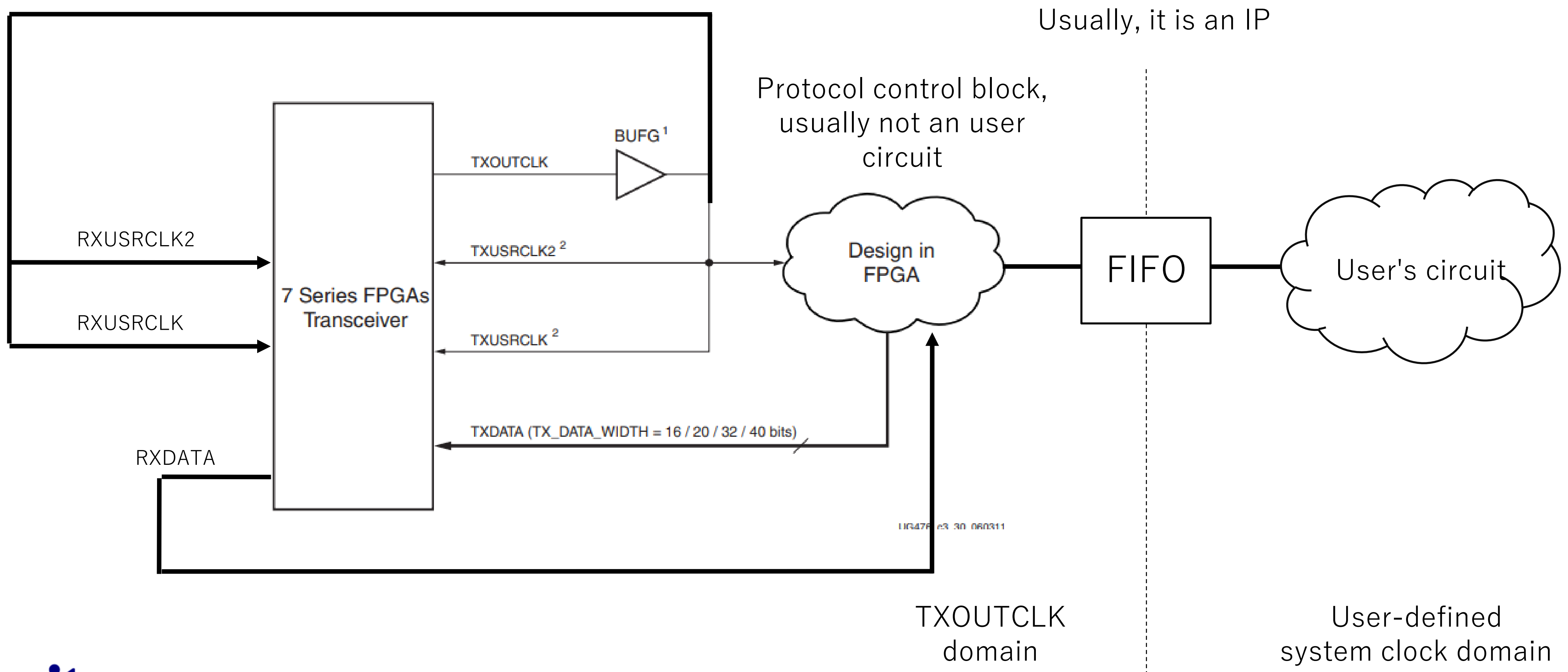

BUFG 3
CLKOUT0
MMCM
CLKOUT1
BUFG 3
BUFH/BUFG 1
LOCKED
TXOUTCLK
CLKIN
TXUSRCLK2 2
7 Series FPGAs Transceiver
TXUSRCLK 2
TXDATA (TX_DATA_WIDTH = 32 / 40 / 64 / 80 bits)
Design in FPGA
TXUSRCLK2 2
7 Series FPGAs Transceiver
TXUSRCLK 2
TXDATA (TX_DATA_WIDTH = 32 / 40 / 64 / 80 bits)
UG476_c3_33_120711

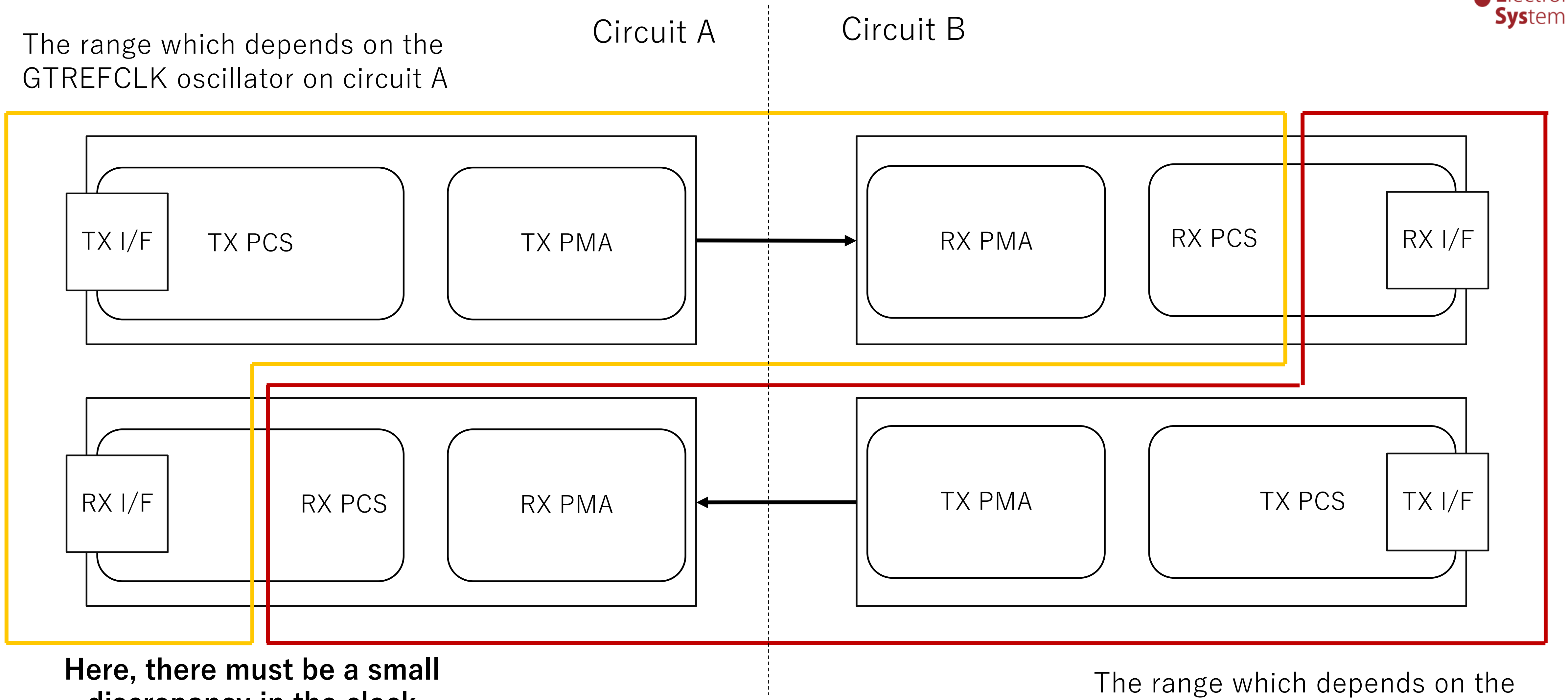
*Figure 3-5:* **Multiple Lanes—TXOUTCLK Drives TXUSRCLK2 (4-Byte or 8-Byte Mode)**

# TX(RX)USRCLK

The line rate is the same for transmission and reception, including RX.
The clock of the protocol-controlling block will not be divided between TX and RX.
Therefore, it would be natural to have RXUSRCLK and RXUSRCLK2 based on TXOUTCLK.


Usually, it is an IP
Protocol control block, usually not an user circuit
BUFG 1
TXOUTCLK
RXUSRCLK2
TXUSRCLK2 2
7 Series FPGAs Transceiver
RXUSRCLK
TXUSRCLK 2
Design in FPGA
FIFO
User's circuit
TXDATA (TX_DATA_WIDTH = 16 / 20 / 32 / 40 bits)
RXDATA
UG476_c3_30_060311
TXOUTCLK domain
User-defined system clock domain

## GTREFCLK's effect range



The range which depends on the GTREFCLK oscillator on circuit A

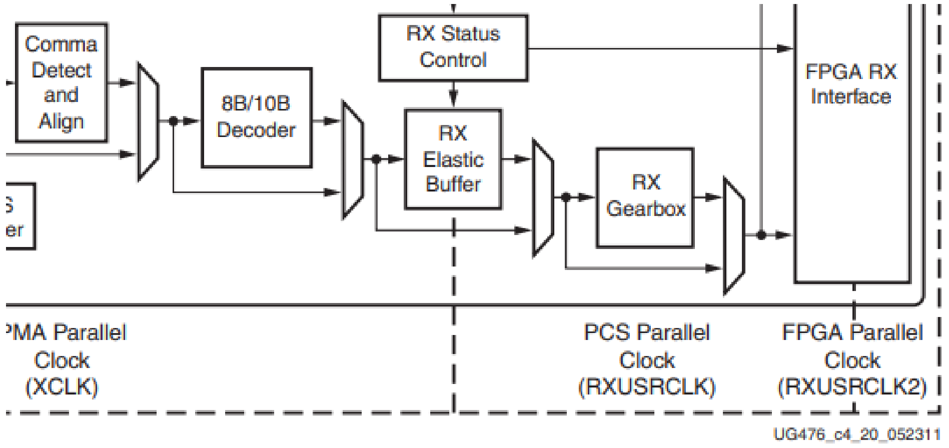
**Here, there must be a small discrepancy in the clock frequency**

**Need elastic buffer**

The range which depends on the GTREFCLK oscillator on circuit AB

## Elastic buffer and clock correction

Clock correction uses IP-specified settings, and it is not recommended to change it by yourself.
Although it is a black box, it's a basic element of serial communication, so you should know how it works.



**RXUSRCLK is faster than XCLK**

- The buffer will eventually underflow.

**RXUSRCLK is slower than XCLK**

- The buffer will eventually overflow.

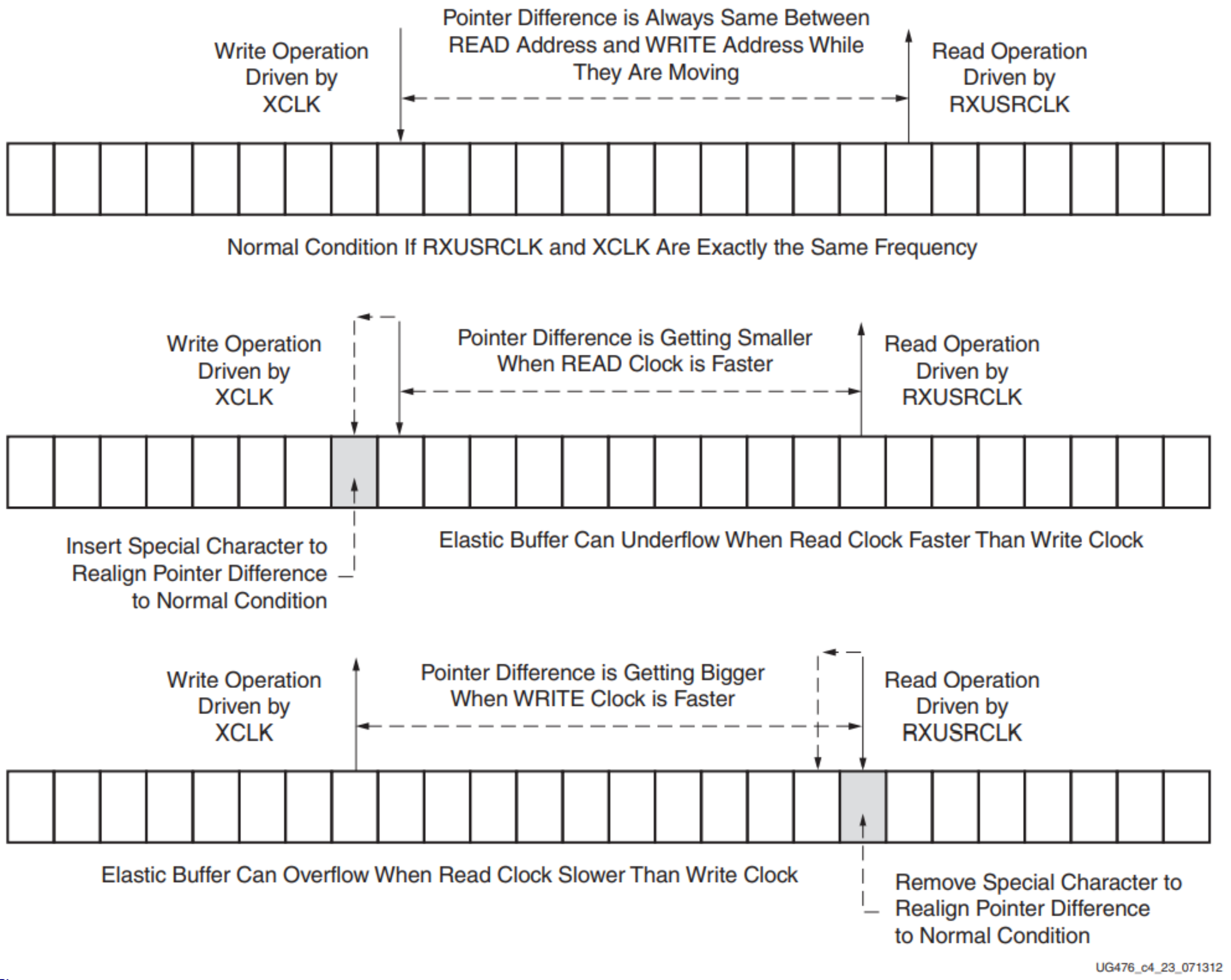
**Then, what should we do?**

TX side sends a special data in a regular intervals (interrupts the user's data stream)

- Duplicate the special data if buffer is about to underflow
- Remove the special data if buffer is about to underflow

It's called "clock correction". Elastic buffer provides this function.
At the user circuit side, the received data rate appears to match its TX line rate within a certain range.
Xilinx UG mentions that the GT reference clock between circuits is aligned within ±100 ppm .

# Elastic buffer and clock correction



Write Operation Driven by XCLK
Pointer Difference is Always Same Between READ Address and WRITE Address While They Are Moving
Read Operation Driven by RXUSRCLK
Normal Condition If RXUSRCLK and XCLK Are Exactly the Same Frequency
Write Operation Driven by XCLK
Pointer Difference is Getting Smaller When READ Clock is Faster
Read Operation Driven by RXUSRCLK
Insert Special Character to Realign Pointer Difference to Normal Condition
Elastic Buffer Can Underflow When Read Clock Faster Than Write Clock
Write Operation Driven by XCLK
Pointer Difference is Getting Bigger When WRITE Clock is Faster
Read Operation Driven by RXUSRCLK
Elastic Buffer Can Overflow When Read Clock Slower Than Write Clock
Remove Special Character to Realign Pointer Difference to Normal Condition
UG476_c4_23_071312

Clock correction is a function that keeps the difference between the write pointer and read pointer inside the elastic buffer constant.
In other words, elastic buffers have a read latency within a certain range.
The latency could be a problem in the trigger circuit

In the experiment synchronizing all clocks including GTREFCLK, one of the purposes is to bypass the RX elastic buffer.

In contrast, if the latency can be tolerated, clock correction will maintain synchronization up to a certain level, so a perfect clock synchronization may not be necessary.

## Summary so far

(Default configuration when generating IP by the wizard)

- TXOUTCLK is generated from GTREFCLK, and various user clocks are based on this.
- The effect range of GTREFCLK is up to XCLK of the destination RX PCS.
- The RX elastic buffer absorbs the difference in frequency and synchronizes within a certain range.
- RX elastic buffer has read latency.

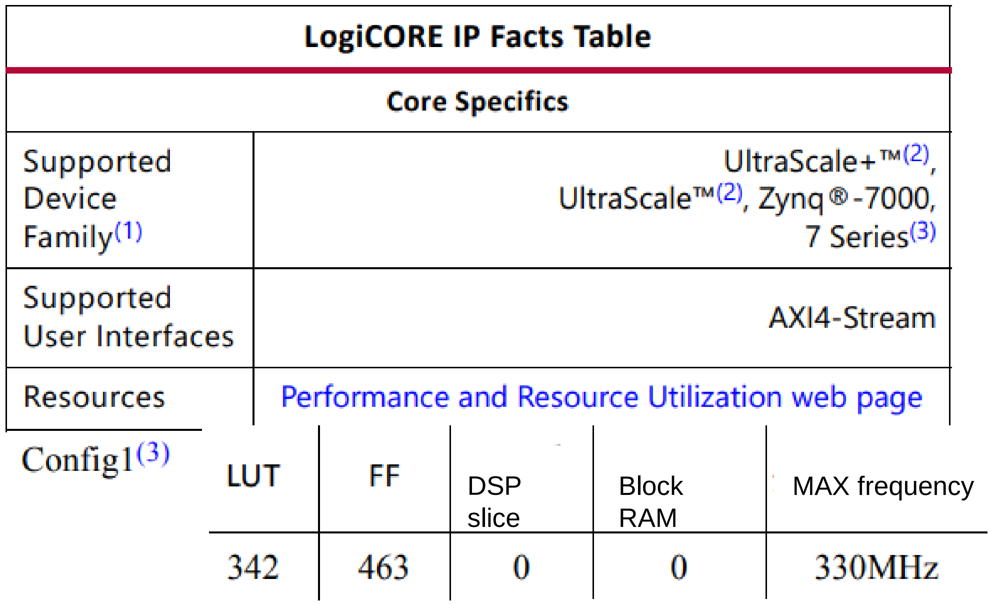
# Try to use Aurora 8b10b

**What is Aurora**

*Aurora is a LogiCORE IP designed to enable easy implementation of AMD transceivers while providing a light-weight user interface on top of which designers can build a serial link. Aurora 8B/10B is a scalable, lightweight, link-layer protocol for high-speed serial communication. The protocol specification is open and available upon request. The IP is free from the IP Catalog on AMD silicon devices.*

From Xilinx's webpage

Since PCS/PMA is a PHY, conversation cannot take place without some kind of protocol.
Aurora is a lightweight and simple protocol to use transceivers.
It supports both framing data and streaming data.



| LogiCORE IP Facts Table | |
|---|---|
| **Core Specifics** | |
| Supported Device Family[1] | UltraScale+™[2], UltraScale™[2], Zynq®-7000, 7 Series[3] |
| Supported User Interfaces | AXI4-Stream |
| Resources | Performance and Resource Utilization web page |

| Config1[3] | LUT | FF | DSP slice | Block RAM | MAX frequency |
|---|---|---|---|---|---|
| | 342 | 463 | 0 | 0 | 330MHz |

PG046

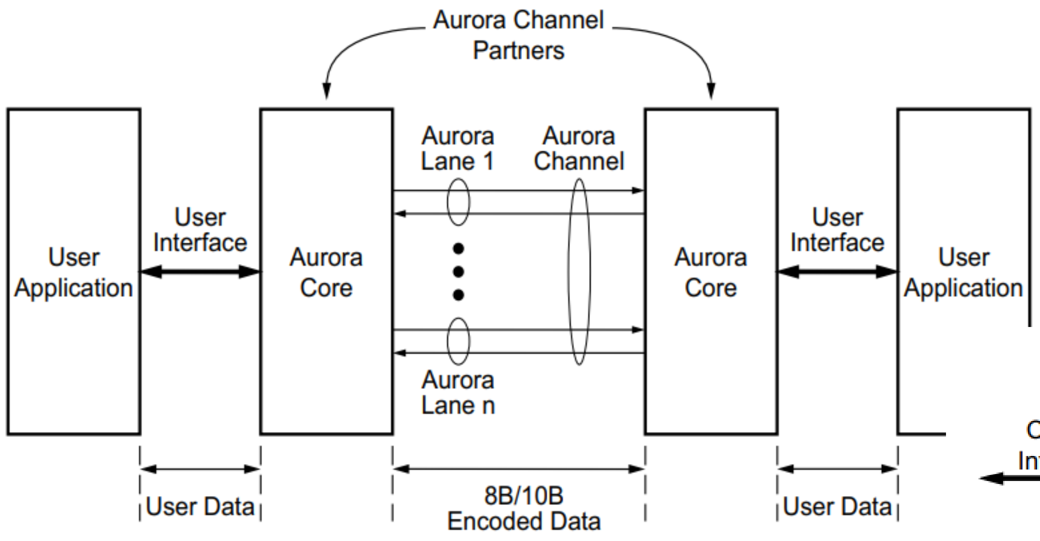
# Aurora 8b10b core

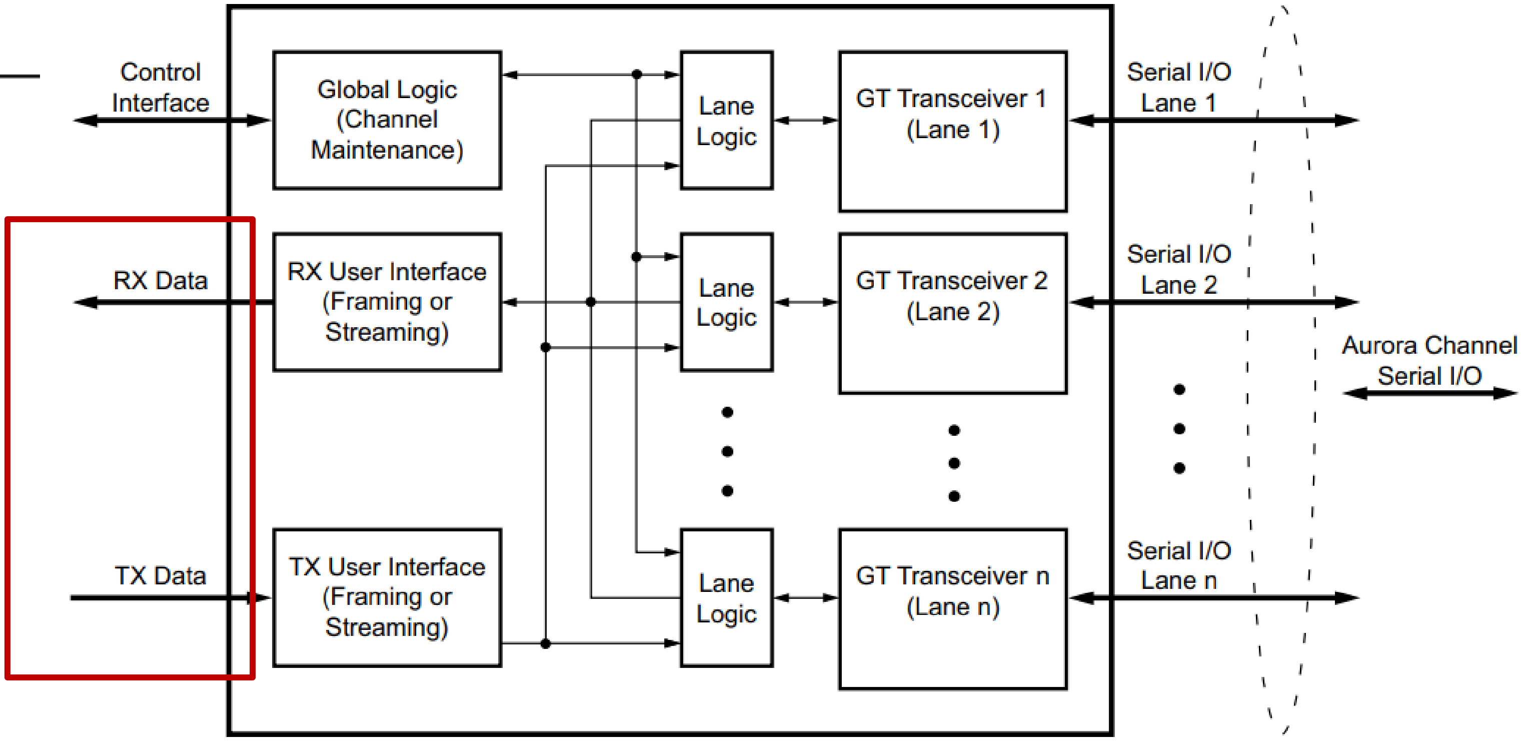
Aurora's IP core can configure one communication block (channel) by bundling multiple GT channels (lanes).

EX: 4 lanes operating with a 2-byte data width are bundled to provide an 8-byte user interface.

⇒ From the user circuit, the throughput seems to be improved.

⇒ Enable the data rates which cannot be reached with a single transceiver.





Control Interface
RX Data
TX Data
Global Logic (Channel Maintenance)
RX User Interface (Framing or Streaming)
TX User Interface (Framing or Streaming)
Lane Logic
Lane Logic
Lane Logic
GT Transceiver 1 (Lane 1)
GT Transceiver 2 (Lane 2)
GT Transceiver n (Lane n)
Serial I/O Lane 1
Serial I/O Lane 2
Serial I/O Lane n
Aurora Channel Serial I/O

*Figure 2-1:* **Aurora 8B/10B Core Block Diagram**

Aurora's data bus uses an on-chip bus standard, called "AXI4-Stream"

It is used by other Xilinx IPs, and it is worth remembering.
Especially, it's important when talking to a processor, such as Zynq.

# AXI4-stream

A bus standard belonging to the AXI interconnect protocol family of AMBA 4.0ヌ
AMBA was originally a bus standard for ARM and it is released with a free license.

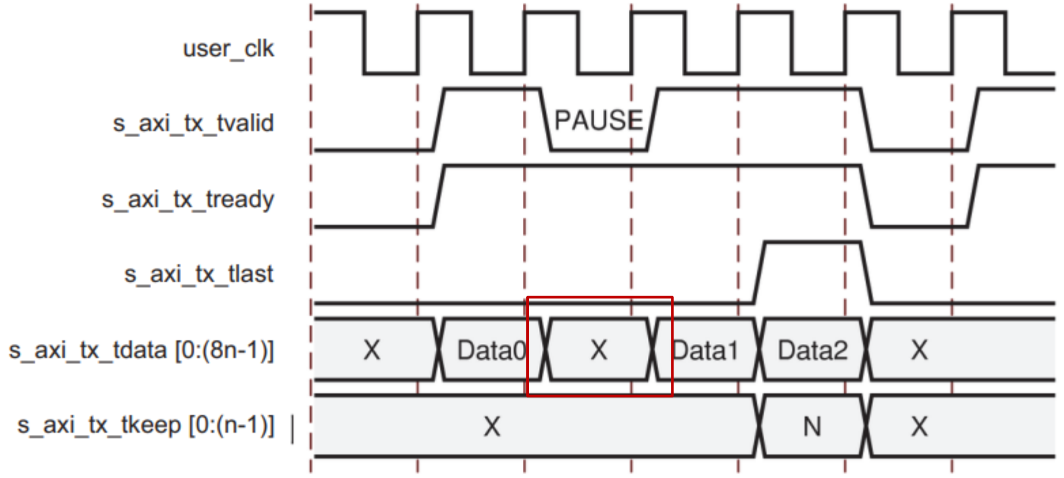
This time, I will explain the mode where the Aurora protocol has a frame structure.
AXI4-Stream has the concept of master/slave, and the one outputting tvalid is the master.

For TX, the data source user circuit is the master.
For RX, Aurora core is the master.

| Signal | Explanation |
|---|---|
| tvalid | A signal indicating that the AXI4-Stream signal from the master is valid. (Active High)<br>It indicates that tdata is valid for Aurora's data interface. |
| tready | A signal indicating that the slave is ready to receive data. (Active High)<br>If tvalid is high and tready is low, the master should stretch the data.<br>It exists only at TX side. |
| tlast | It marks the end of the user data frame.(Active High) |
| tkeep | It indicates the valid byte contents of the last data in the frame. It is valid only when tlast is high. You can think of it as byte write enable for TX and byte read valid for RX. |
| tdata | User data. |

# AXI4-stream

### Timing chart of transmitting data

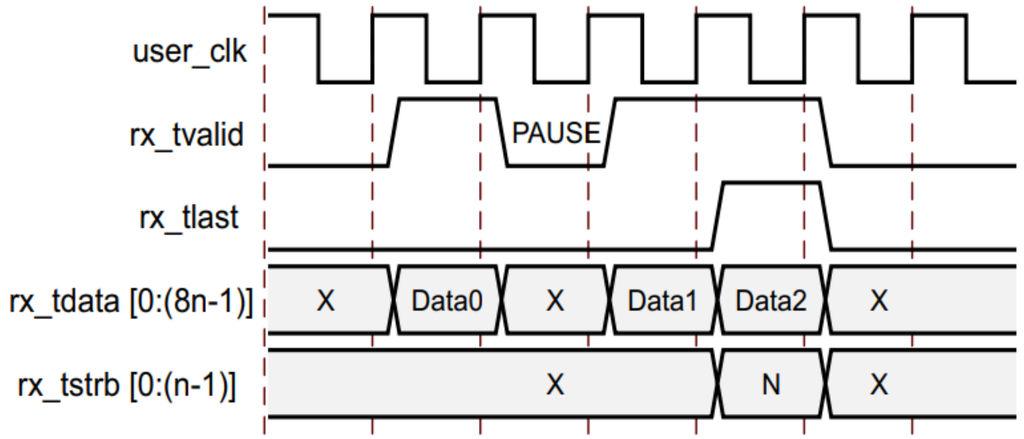


During frame transmission, if tvalid is de-asserted,
it indicates a pause.
In the middle of frame transmission, the transmission data
is changed to idle characters.

In this example, the three words, Data0, Data1, and Data2,
are a set of user data, which are
encapsulated in Aurora's frame.

The frame length is arbitrary and the Aurora frame's
terminator is not inserted until tlast is asserted.

### Timing chart of receiving data



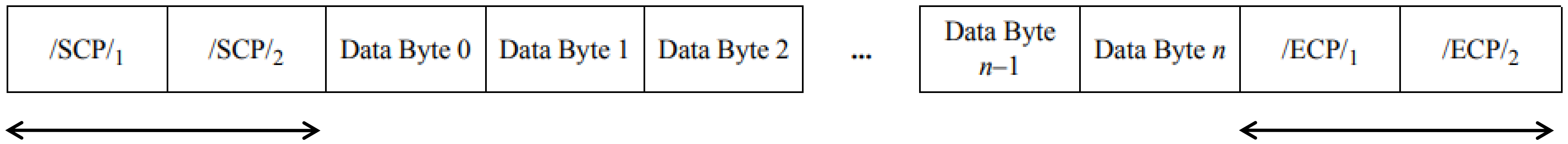
tdata is valid when rx_tvalid is high.
tlast is asserted at the end of the user frame.

# Aurora 8b10b frame

The Aurora frame is very simple.

Standard channel frame

表 3-6 : 標準的なチャネル フレーム



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| /SCP/$_1$ | /SCP/$_2$ | Data Byte 0 | Data Byte 1 | Data Byte 2 | ... | Data Byte *n*–1 | Data Byte *n* | /ECP/$_1$ | /ECP/$_2$ |

K character

K character

inserted with a tlast assert

# Aurora 8b10b core

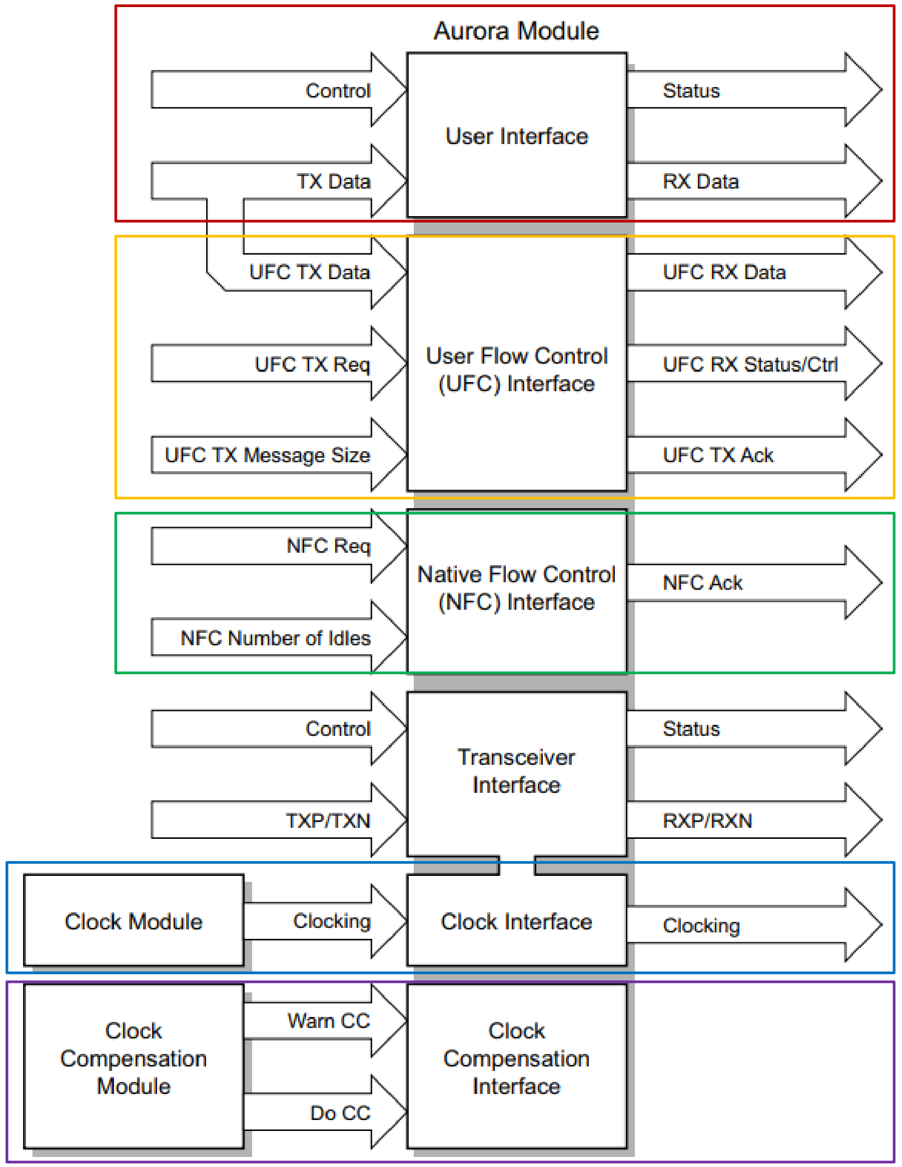


図 2-3 : 最上位インターフェイス
Upper-most interface

What has been explained so far.

An optional function the can be selected in the framing mode.
Other short messages can interrupt the transmission of user data.
Data written to UFC is prior to flow to lane.

An optional function the can be selected in the framing mode.
A function controlling the rate on the receiving side by inserting idle characters of a specified length at the timing of the request during data transmission.
It is utilized when the processing capacity of the receiving side is not high.

With the knowledge of PCS/PMA, it can be describe correctly,
It is safer to follow the example design first.

It controls clock correction.
It must be inside the core.

# Aurora's option

**CRC (Cyclic redundancy check)**

- A type of error detection code. Data corruption is detected when sending and receiving data.

**Scrambler/Descrambler**

- Scramble the data into random numbers using a pseudo-random sequence (TX side)
- Undo the scrambling on the data (RX side)

- Avoid the increasing EMI at a specific frequency due to continuation of specific bit patterns.
- For 8b10b, the running disparity maintains DC balance, but does not avoid specific patterns.

The slides for exercise from here
Skipped for the first day

## Try to implement Aurora8b10b

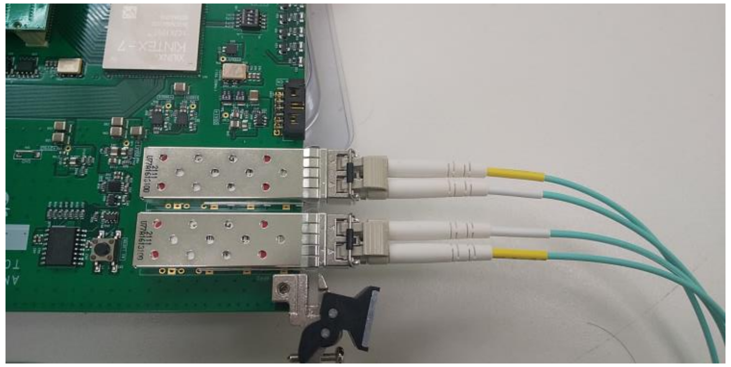
**Specifications for implementation**

- Connect the two optical modules inserted in AMANEQ with a fiber for loop back.
- Select Simplex to generate one TX-only core and one RX-only core.
  - Select sidebands for back channel and connect TX and RX.
  - (What happens during initialization inside the full-duplex core can be seen.)
- TX always transmits incremental data.
- Assert tlast every 8 clocks with 16-byte as a user frame's unit.

**Aurora core's conditions**

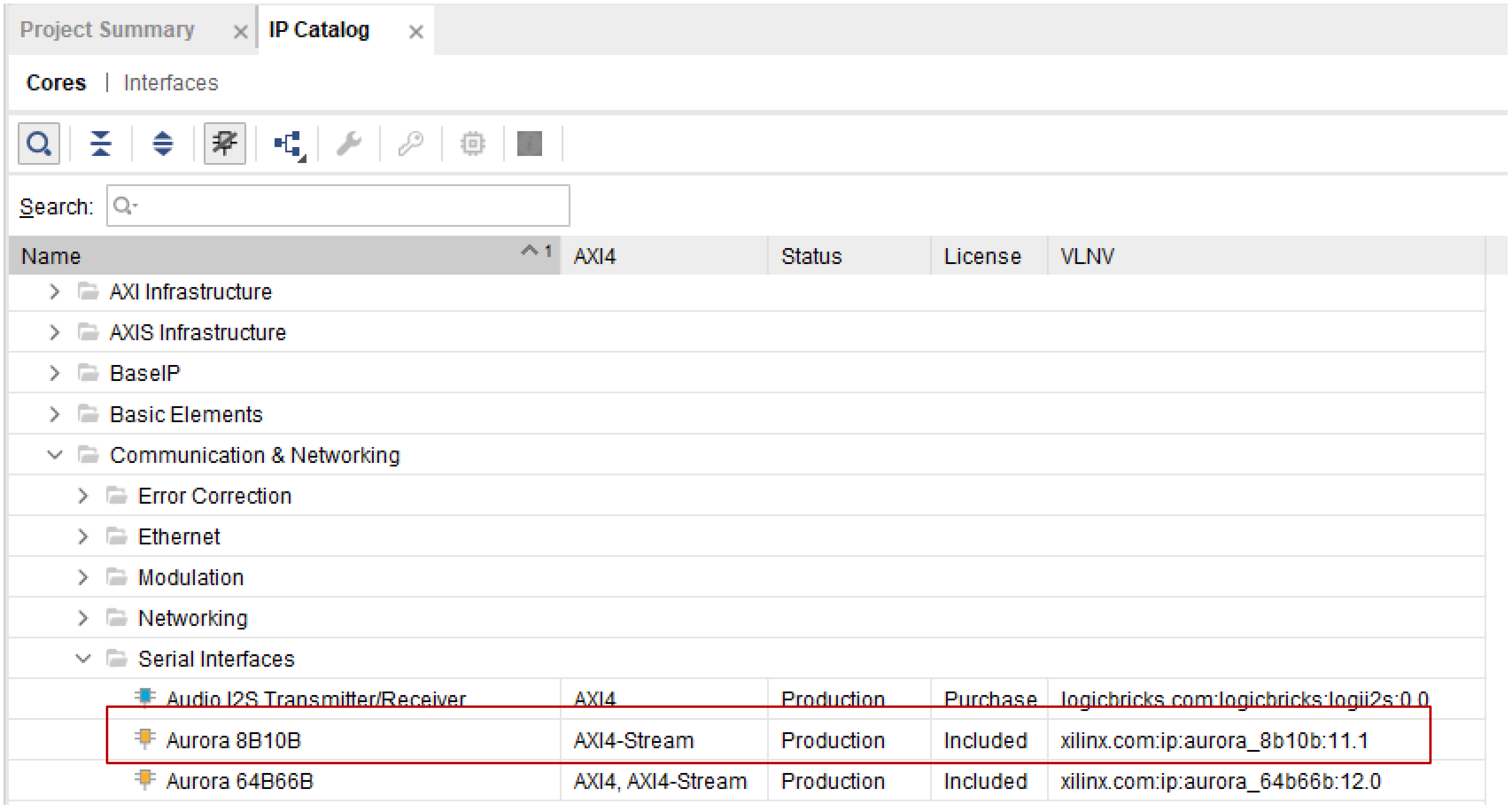
| | |
|---|---|
| • GTREFCLK frequency: | 156.25 MHz |
| • Data width: | 2-byte |
| • Line rate: | 1.25 Gbps |
| • INIT CLK: | 100 MHz |
| • DRP CLK: | 100 MHz |

- No UFC, NFC, scrambling, CRC



It might be difficult to implement it for a beginner of GT.
I'll try it halfway through.

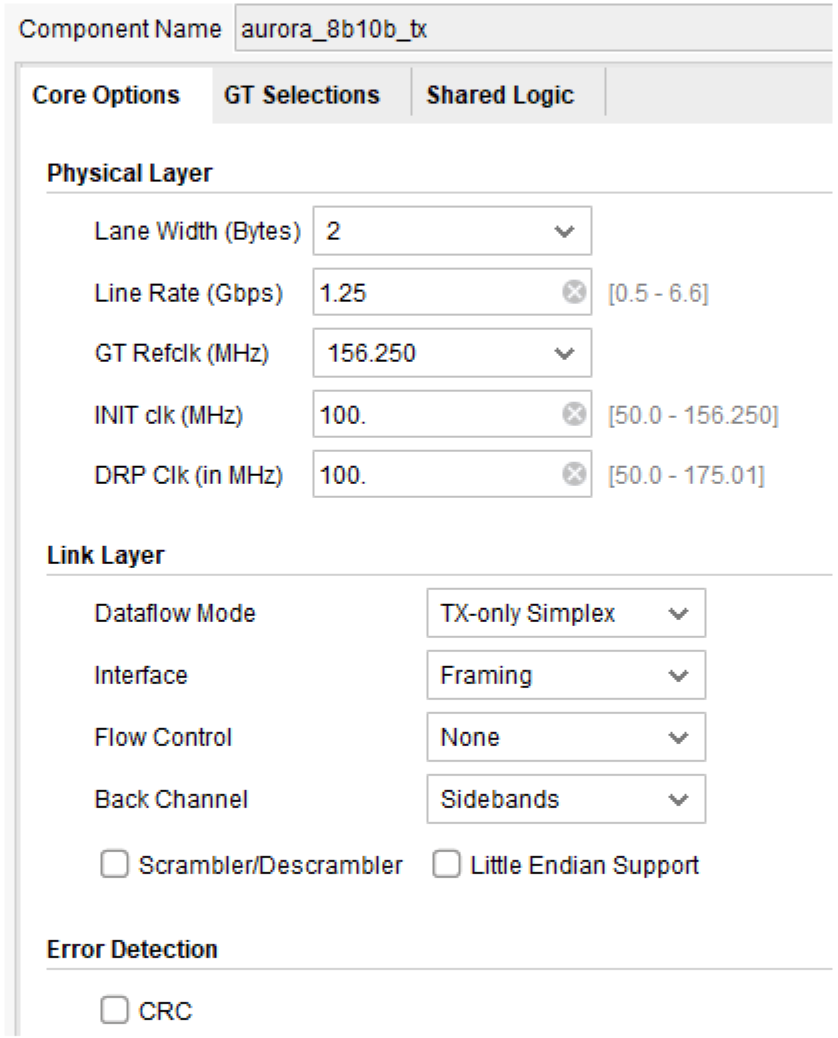
# Try to implement Aurora8b10b



| Name | AXI4 | Status | License | VLNV |
|---|---|---|---|---|
| AXI Infrastructure | | | | |
| AXIS Infrastructure | | | | |
| BaseIP | | | | |
| Basic Elements | | | | |
| Communication & Networking | | | | |
| Error Correction | | | | |
| Ethernet | | | | |
| Modulation | | | | |
| Networking | | | | |
| Serial Interfaces | | | | |
| Audio I2S Transmitter/Receiver | AXI4 | Production | Purchase | logicbricks.com:logicbricks:logii2s:0.0 |
| Aurora 8B10B | AXI4-Stream | Production | Included | xilinx.com:ip:aurora_8b10b:11.1 |
| Aurora 64B66B | AXI4, AXI4-Stream | Production | Included | xilinx.com:ip:aurora_64b66b:12.0 |

# Try to implement Aurora8b10b

First, TX side



| | |
|---|---|
| Component Name | aurora_8b10b_tx |

Core Options | GT Selections | Shared Logic

**Physical Layer**

| | | |
|---|---|---|
| Lane Width (Bytes) | 2 | |
| Line Rate (Gbps) | 1.25 | [0.5 - 6.6] |
| GT Refclk (MHz) | 156.250 | |
| INIT clk (MHz) | 100. | [50.0 - 156.250] |
| DRP Clk (in MHz) | 100. | [50.0 - 175.01] |

**Link Layer**

| | |
|---|---|
| Dataflow Mode | TX-only Simplex |
| Interface | Framing |
| Flow Control | None |
| Back Channel | Sidebands |

☐ Scrambler/Descrambler ☐ Little Endian Support

**Error Detection**

☐ CRC

## INIT clk

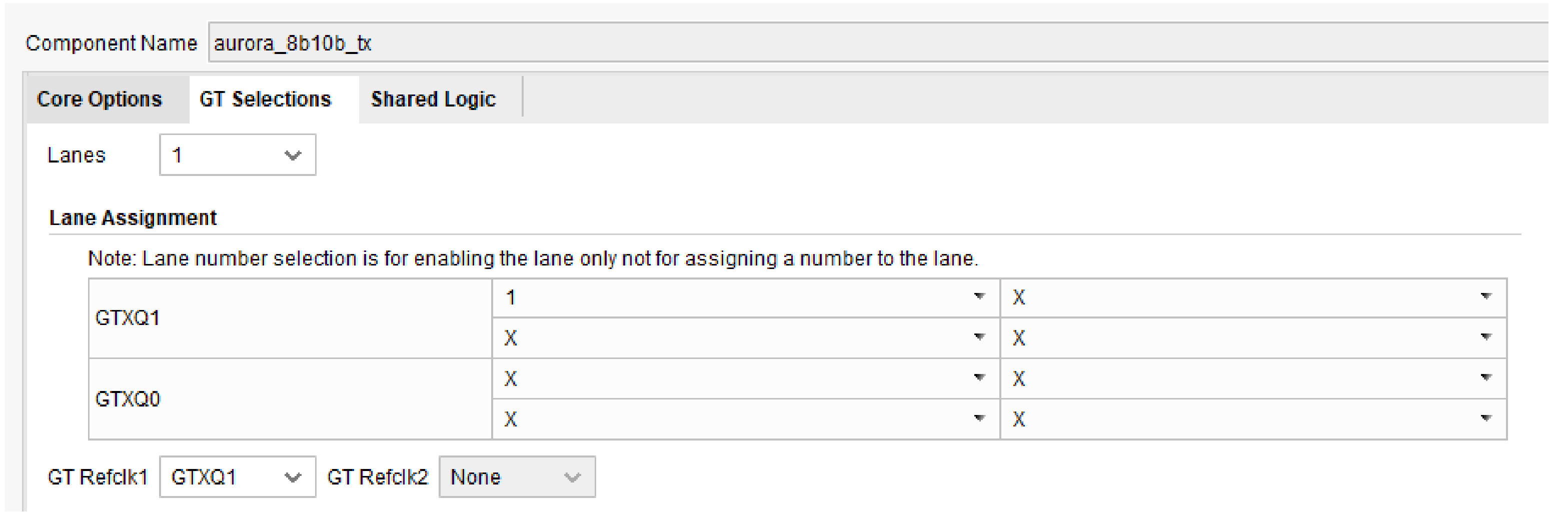
- TXOUTCLK is lost when a transceiver reset is asserted. During that moment, this clock is used.
- It is required to support hotplug functionality.
- Use a clock source independent of the transceiver.
- This time, it is attached directly to an oscillator.

## DRP clk

- GTX channel's control port.
- It still works without it.
- This time, connect the same clock as INIT clk.

# Try to implement Aurora8b10b

First, TX side



Component Name: aurora_8b10b_tx

Core Options | GT Selections | Shared Logic

Lanes: 1

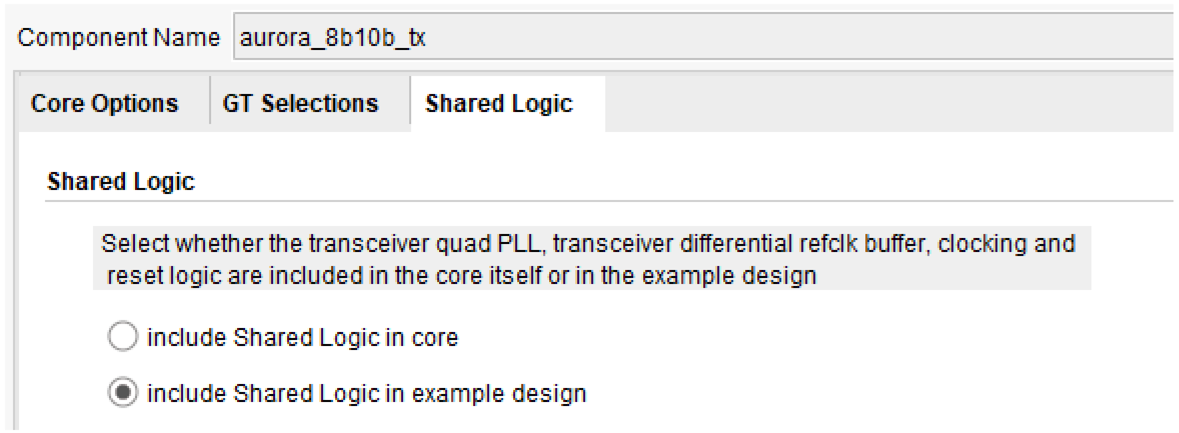
**Lane Assignment**

Note: Lane number selection is for enabling the lane only not for assigning a number to the lane.

| | | |
|---|---|---|
| GTXQ1 | 1 | X |
| | X | X |
| GTXQ0 | X | X |
| | X | X |

GT Refclk1: GTXQ1 GT Refclk2: None

This time, only 1 lane is implemented per channel, so no need to think too much about it.
Specify the resource location with constraints later.



Component Name: aurora_8b10b_tx

Core Options | GT Selections | Shared Logic

**Shared Logic**

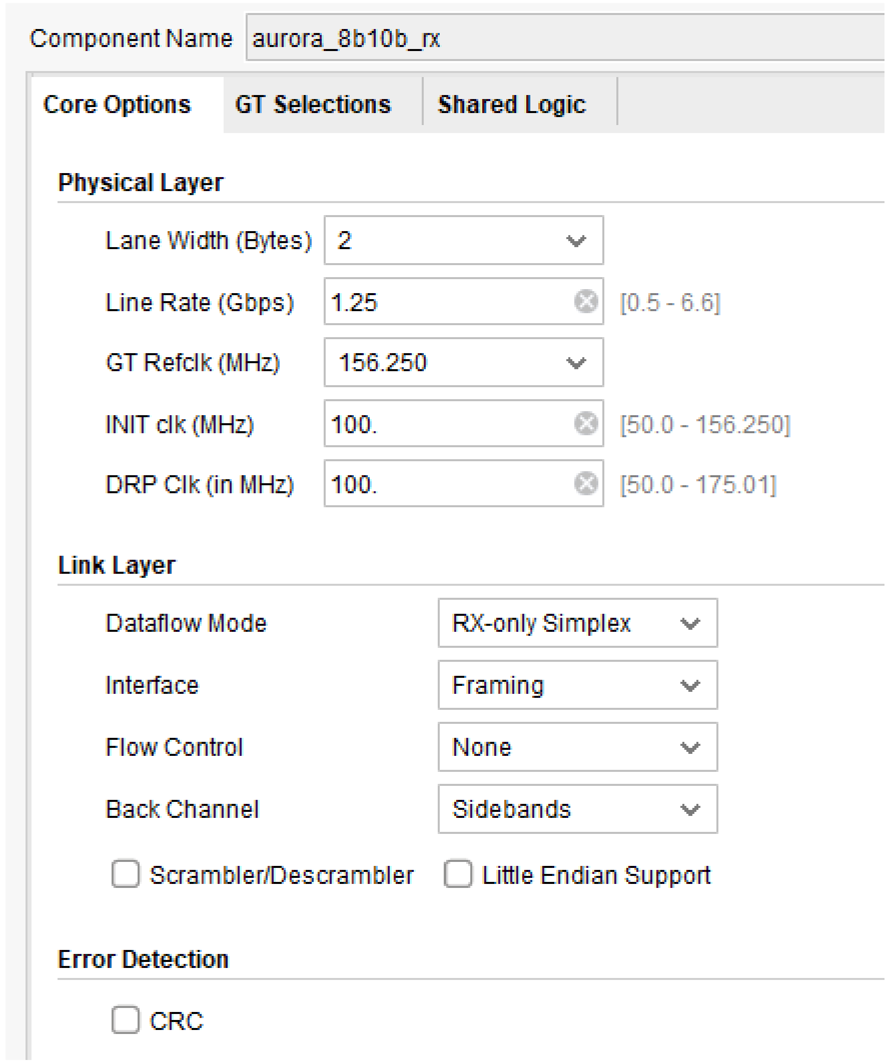
Select whether the transceiver quad PLL, transceiver differential refclk buffer, clocking and reset logic are included in the core itself or in the example design

○ include Shared Logic in core
● include Shared Logic in example design

To put various peripheral circuits that support Aurora's IP core:
Into the IP core,
Or make them into modules and implement them yourself?
It is better to choose the latter so that you can play around with it by yourself.

# Try to implement Aurora8b10b

## RX side



Component Name: aurora_8b10b_rx

Core Options | GT Selections | Shared Logic

**Physical Layer**

| | | |
|---|---|---|
| Lane Width (Bytes) | 2 | |
| Line Rate (Gbps) | 1.25 | [0.5 - 6.6] |
| GT Refclk (MHz) | 156.250 | |
| INIT clk (MHz) | 100. | [50.0 - 156.250] |
| DRP Clk (in MHz) | 100. | [50.0 - 175.01] |

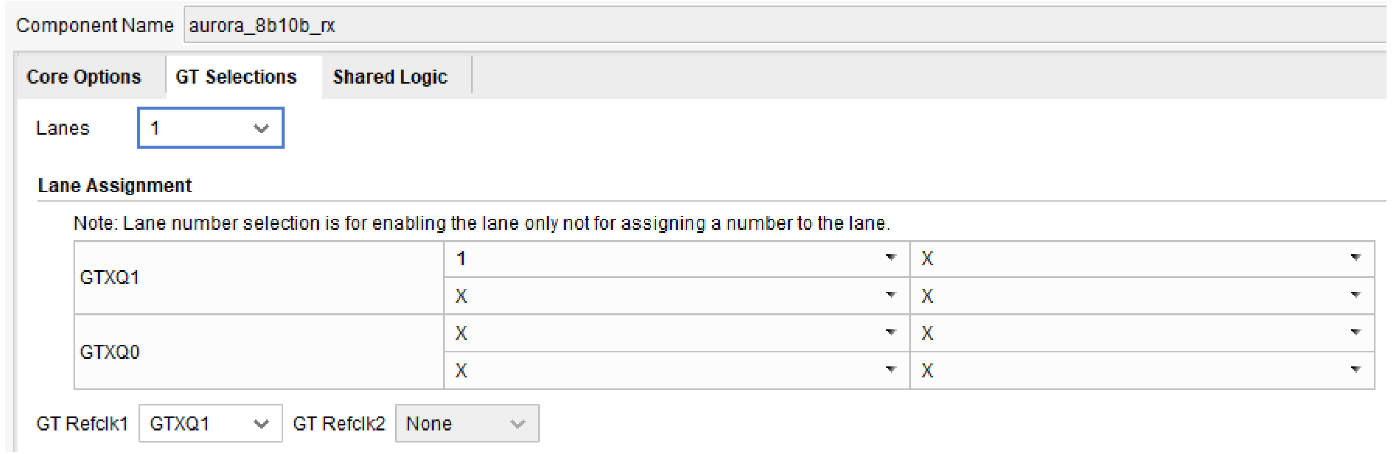
**Link Layer**

| | |
|---|---|
| Dataflow Mode | RX-only Simplex |
| Interface | Framing |
| Flow Control | None |
| Back Channel | Sidebands |

☐ Scrambler/Descrambler ☐ Little Endian Support

**Error Detection**

☐ CRC



Component Name: aurora_8b10b_rx

Core Options | GT Selections | Shared Logic

Lanes: 1

**Lane Assignment**

Note: Lane number selection is for enabling the lane only not for assigning a number to the lane.

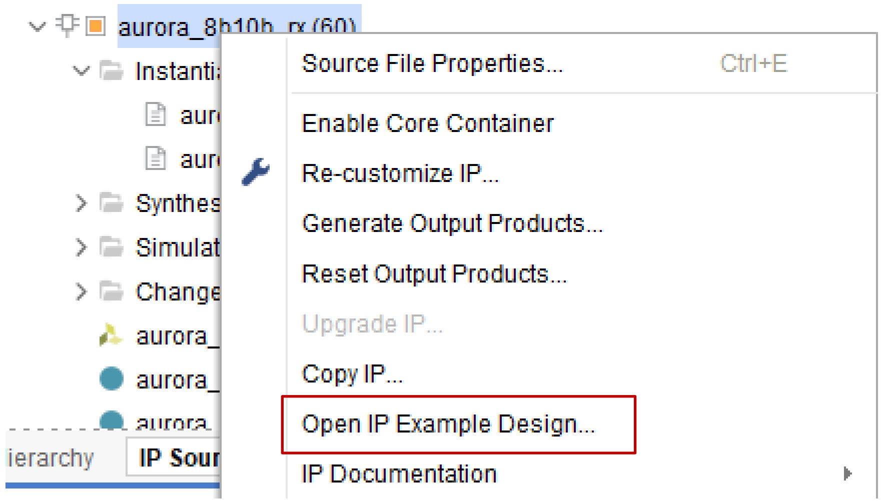
| | | |
|---|---|---|
| GTXQ1 | 1 | X |
| | X | X |
| GTXQ0 | X | X |
| | X | X |

GT Refclk1: GTXQ1  GT Refclk2: None

# Try to implement Aurora8b10b

I would like to suggest that we will implement it from the instantiation template, but it's probably impossible.

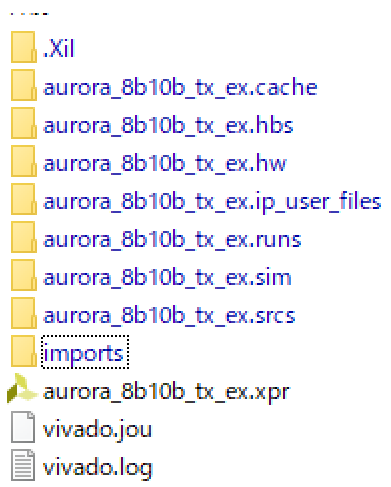
It is a good idea to generate an example design for the transceiver IP and use it as a reference.



Generate a Vivado project for the example design.
Another Vivado will launch when with specifying the destination of generation.

# Try to implement Aurora8b10b

## Inside the example design



.Xil
aurora_8b10b_tx_ex.cache
aurora_8b10b_tx_ex.hbs
aurora_8b10b_tx_ex.hw
aurora_8b10b_tx_ex.ip_user_files
aurora_8b10b_tx_ex.runs
aurora_8b10b_tx_ex.sim
aurora_8b10b_tx_ex.srcs
imports
aurora_8b10b_tx_ex.xpr
vivado.jou
vivado.log

**srcs**

- IP's HDL code.
- For Aurora, all core codes are generated without black boxes.
- As in the source code, it does not use FPGA dedicated resources and can be implemented entirely in fabric.
- It is helpful when making your own protocol.

**imports**

- Example design's source code.
- Many things here, but the code written as support is the most important.
- We need to run the modules connected to the support module. Just need to find them and port them.
- It is necessary to change the code to some extent based on your own circuit.

**For those who don't know what to do in H2**

- List what to copy from imports.
- Let's consider where to change the copied codes by referring to the block diagram in the practice manual of H2.
- Let's wire the clock so that it is just like the block diagram of the manual.
- If you still don't know the correct wiring, please refer to the teaching material source code (the answer).