Multi-Gigabit transceiver

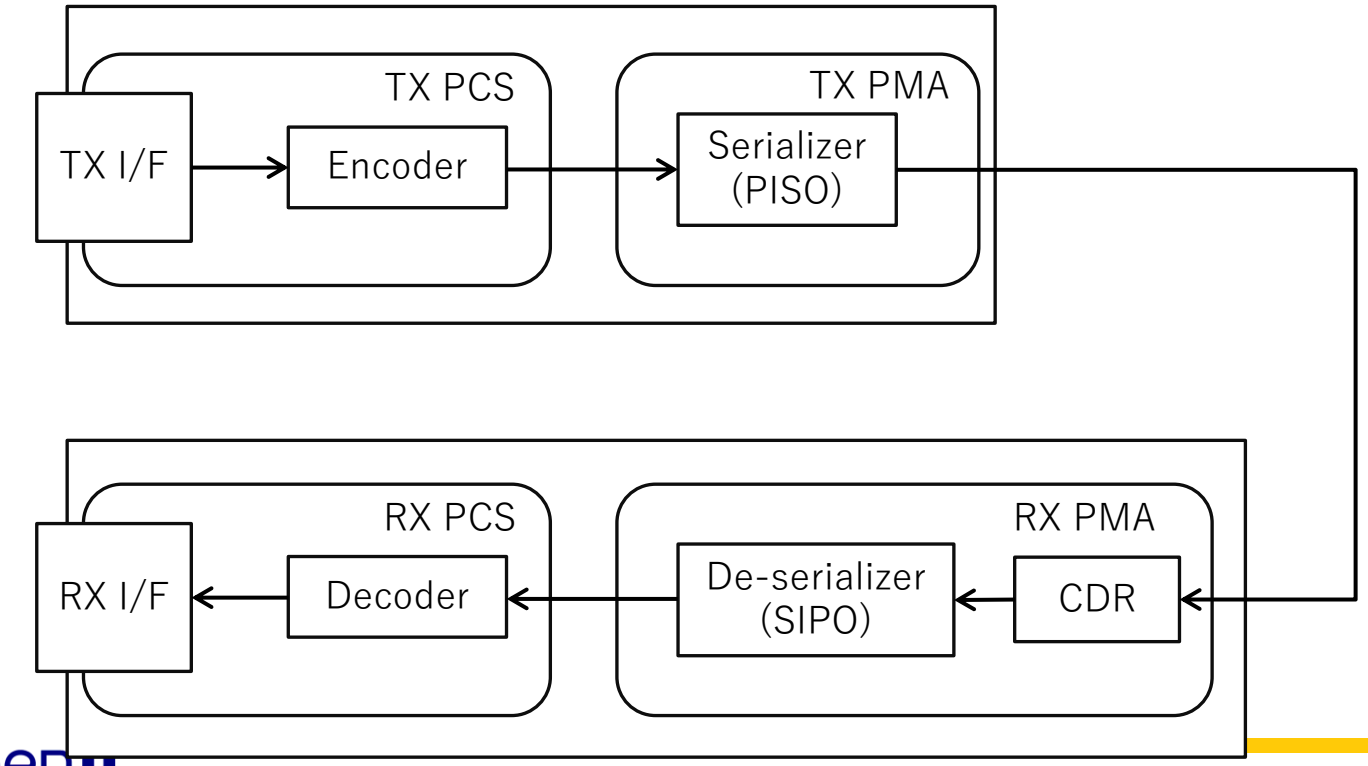
KEK IPNS E-sys
本多良太郎

おさらい

GT channel ⇔ 高速シリアル通信のPHY (物理層を駆動するデバイス)
PCS (Physical Coding Sublayer) 物理符号化副層
- 主にバイナリ/シンボル変換を行う (8b10b encode/decodeなど)

PMA (Physical Medium Attachment) sublayer 物理媒体接続副層
- 主にシリアル/パラレル変換を行う

GTチャンネルの大まかなブロック図
(多くの機能を省略して書いてます)



*GTREFCLKはTXのラインレートを
決定するために使う

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

***CDR**: Clock Data Recovery
データのビット列からクロックを取り出す
(復元) するための回路

GTチャンネルはPHYであるため
通信の最下層の機能しか提供しない。

実際の通信には
EthernetならEthernetのルールに従ったフレームを
AuroraならAuroraのルールに従ったフレームを
組んで流す必要がある。
(ふつうはIPを使う)

この章で学ぶもの

GT quad (物理層)

- Transmitterとreceiverの内部構造 (ここはさらっと)
- クロック関係
  - GTREFCLKとline rateの関係
    - CPLLとQPLL
  - PCS用のクロックとユーザークロックの関係性
  - Elastic bufferの役割とclock correctionの動作

Xilinx Aurora 8b10b (通信プロトコル)

- AXI-4 stream
- このコアで何が出来てどう使うか。
- 実際に実装してみる（発展課題 H2）

用語

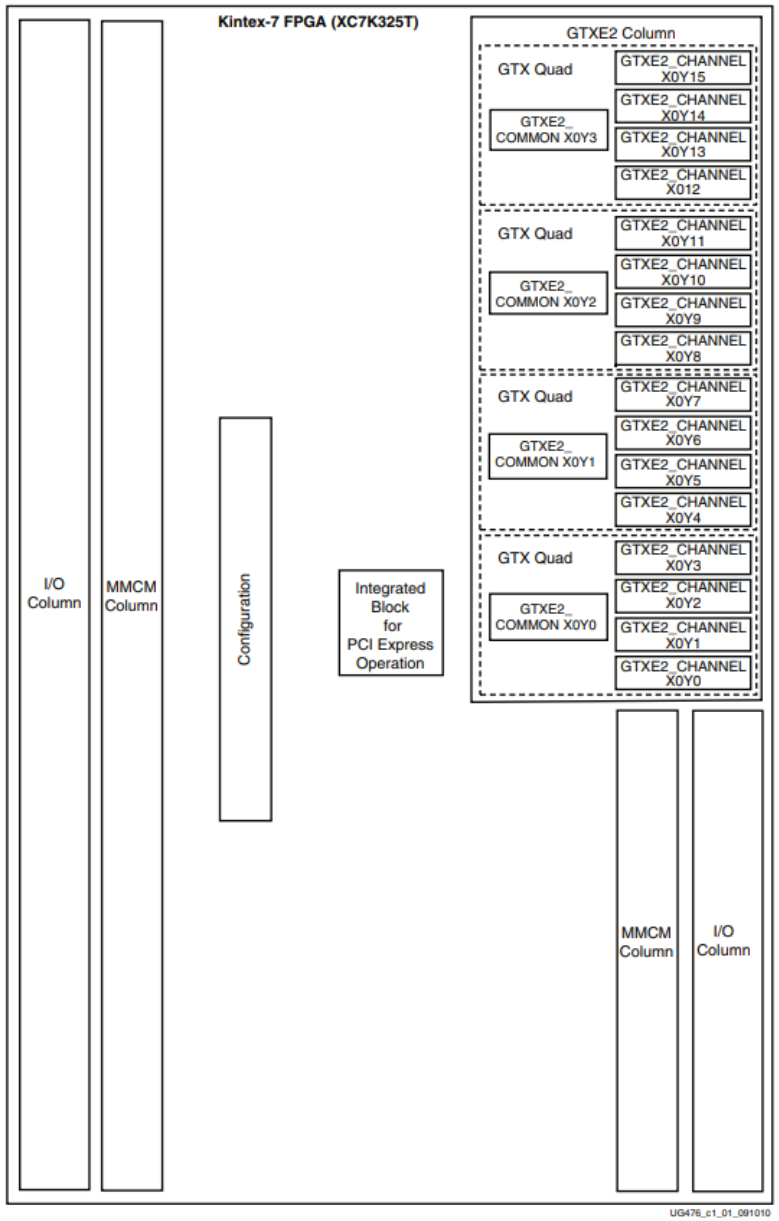


Figure 1-1: **GTX Transceiver Inside Kintex-7 XC7K325T FPGA**

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

- 1つtransmitterと1つのreceiverの組が格納されたブロック。

GT(P...) Quad

- 4つのチャンネルをまとめた1グループ。
- FPGAのバンクに相当する。
- 1つのクアッドは2つのGTREFCLK入力を有する。

GT(P...) Common

- クアッド内のチャンネルに共通のクロックを供給するためのブロック。QPLLが配置されている。

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

- トランシーバリソースが配置されている列

用語

Line rate

- PMAが信号線に流すビットレート。bit-per-second (bps)で書く。
  - ユーザーが送信したいデータの送信レートではない
- Transfer-per-second (T/s)で書くことがあるが、同一。
- Line rateが1.25 Gbpsで8b10b変換を利用している場合、信号線には10-bitシンボルが1.25 Gbpsのレートで流れている。8-bitバイナリのレートは1.0 Gbps。
  - シリアル通信では通信のプロトコルに従いフレームを組むので、ユーザーが送りたいデータ(フレームのペイロード)でみると更に下がる。

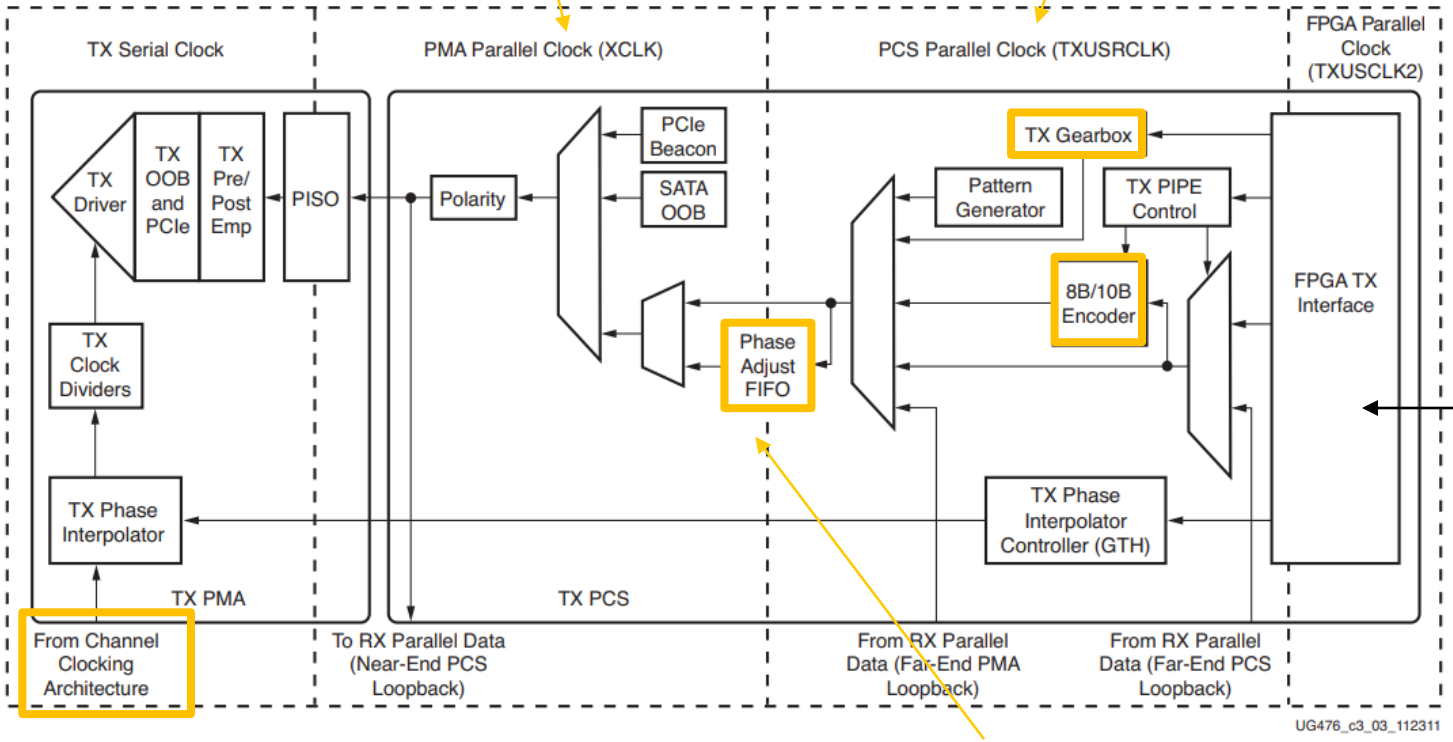
Lane

- 通信の伝送路。Full-duplex (全二重) のばあいTXとRX両方合わせて1つのレーン。

GTXE2 channel内部 (transmitter)

XCLKはGTの基準クロック(GTREFCLK)
から生成される。

TXUSRCLK(2)はユーザー回路のクロック。
GTの基準クロックとは独立である可能性がある。
(USRCLKとUSRCLK2の違いは後程)



8B/10B encoder
10ビットシンボルを与える。

TX gearbox
64b66bなど8b10b以外の変換を行う。

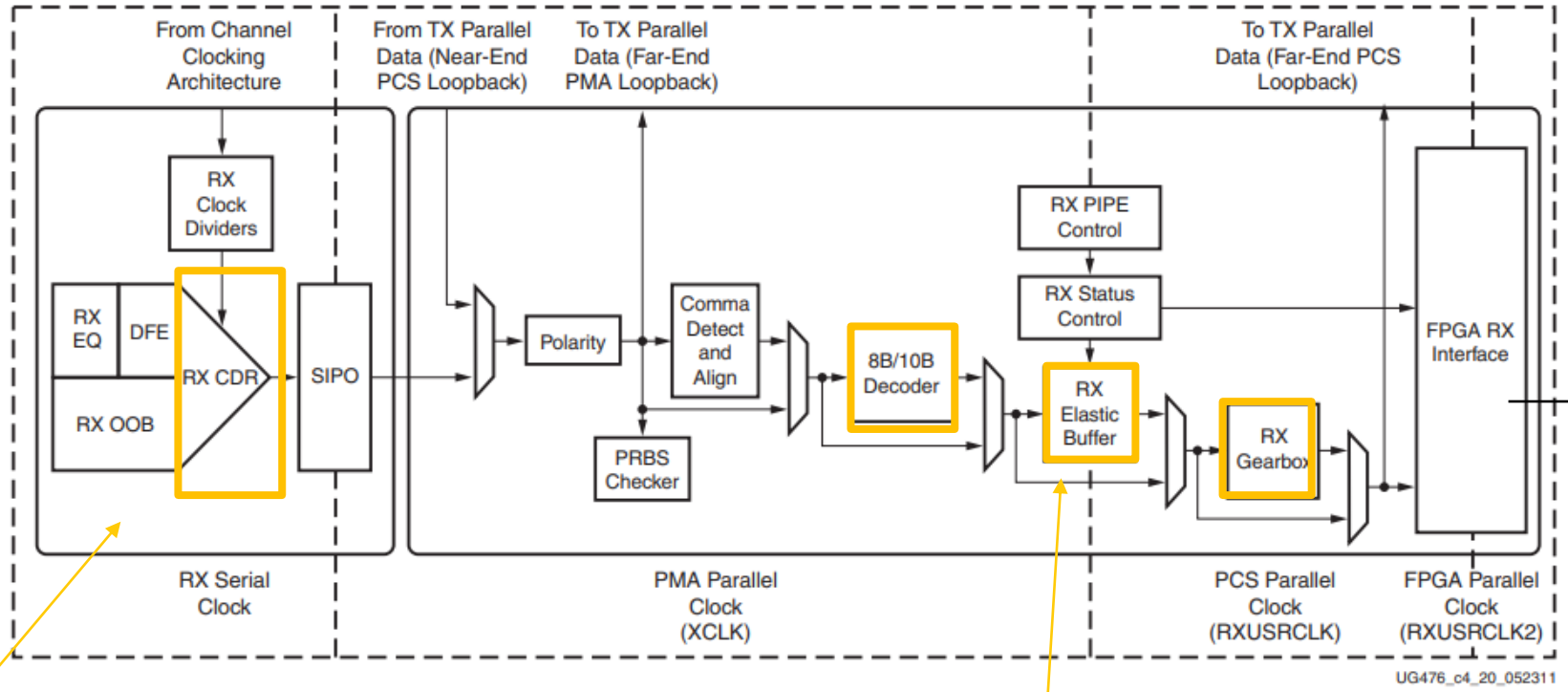
上流の回路からのデータ入力
IPでAuroraやEthernet用のコアを生成するとここには通信プロトコルをつかさどる回路がつながる。
ユーザー回路が直接つながることはないはず。

ラインレートを決めるクロック
(クロック関係回路は独立に存在)
GT基準クロックをもとに伝送路を駆動するクロック。
XCLKもデフォルトではここから分岐する。

Phase Adjust FIFO: TX buffer
2つのクロック間の位相差を吸収するバッファ。
このバッファをバイパスするのは高度な利用法であり
この演習では対象外。

GTXE2 channel内部 (receiver)

RX側にもXCLKとRXUSRCLKの2種類のドメインが存在する。



8B/10B decoder
8ビットバイナリを与える。

RX gearbox
64b66bなど8b10b以外の変換を行う。

下流の回路へのデータ出力
TX同様ユーザー回路は直接つながらないはず。

Clock Data Recovery (CDR)
受信信号からクロックを復元しデータを分離する。
復元されるクロックの周波数はTX側の基準クロックに依存している。
XCLKもデフォルトではここから分岐する。

RX elastic buffer
2つのクロック間の位相差だけでなく微妙な周波数差を吸収するためのバッファ。
XCLKとRXUSRCLKは同一でない事が前提になっている。

GTREFCLKとline rateの関係

ポイント
- RX側はCDR回路によってクロックを復元するのでラインレートを決めるのはTX側。
- GTREFCLKは伝送路を駆動するクロックの元になるクロック。

GTXではGTREFCLKの入力は
バンク（クアッド）に付き2つ

IBUFDS_GTE2という
専用プリミティブが必要

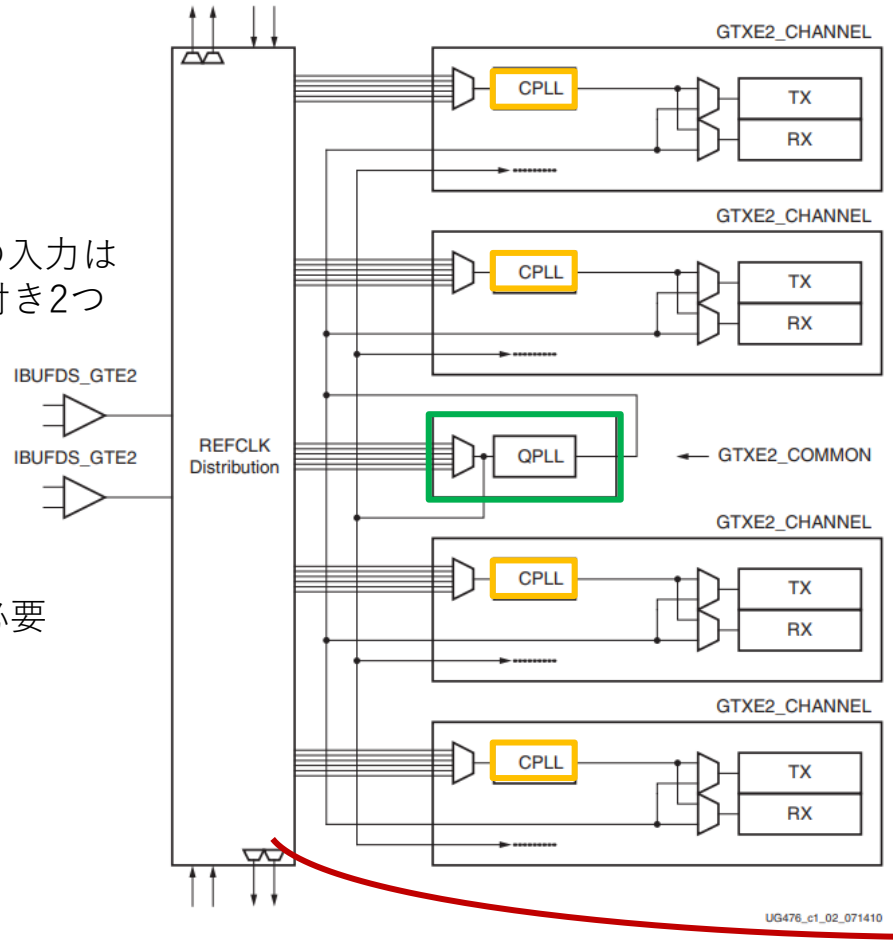
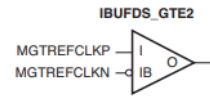
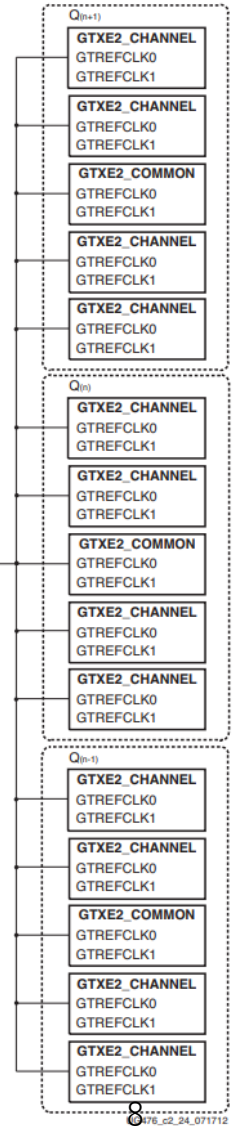


Figure 1-2: **GTX Transceiver Quad Configuration**

GTREFCLKはクアッド内の
- **Channel PLL (CPLL)**
- **Quad PLL (QPLL)**

に入力される。





縦方向のクロックラインにより
こう言う事も可能
ただしクロックソースとなるク
アッドから上下1つず、合計3ク
アッドまでしか駆動できない。

GTREFCLKとline rateの関係 (CPLL)

Channel PLL

- 各channel内に存在するPLL。
- CPLLのおかげでチャンネル毎にラインレートを決定できる。
- CPLLの設定はIPを生成する際にウィザードが行ってくれ、それを変更することは普通は無い。
  - IPが設定したパラメータを手動で変更するのはアドバンスな利用 (発展課題で取り扱います)。

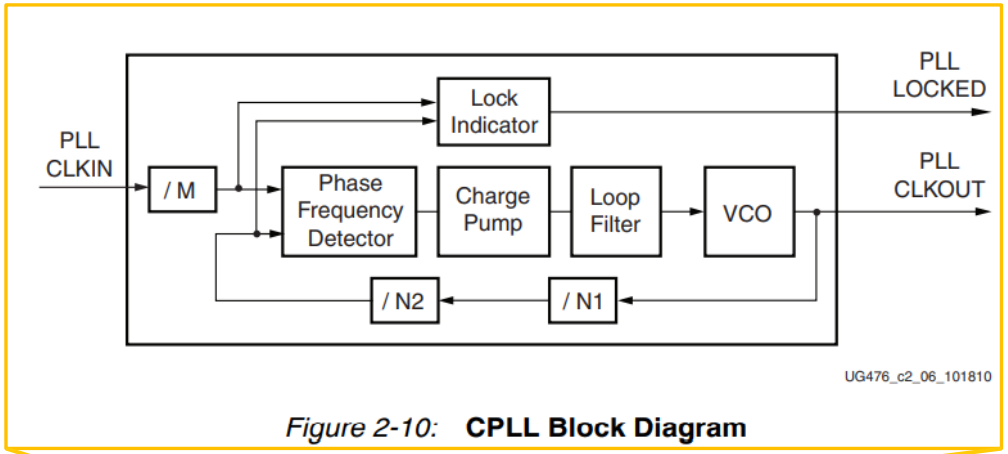


Figure 2-10: **CPLL Block Diagram**

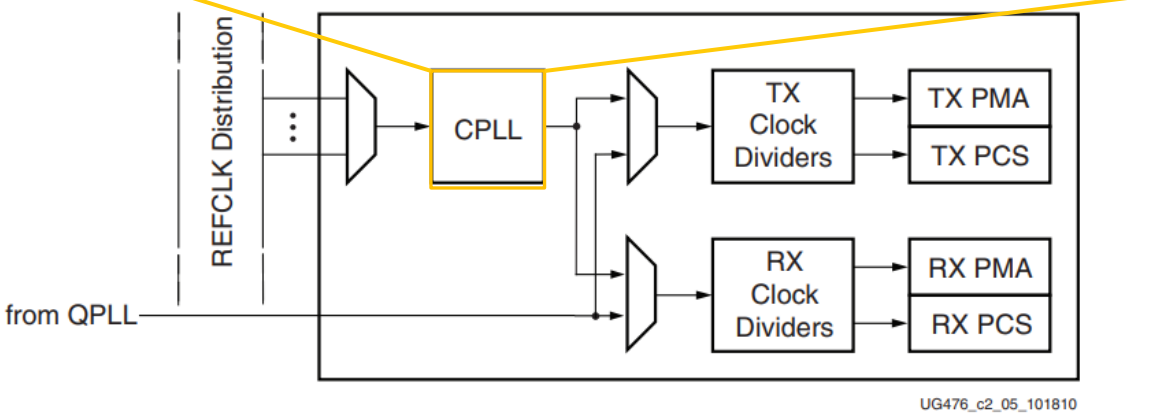


Figure 2-9: **Internal Channel Clocking Architecture**

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

$$f_{LineRate} = \frac{f_{PLLClkout} \times 2}{D} \qquad \textit{Equation 2-2}$$

Line rateという言葉がここで出てくる

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 TXOUT_DIV	1, 2, 4, 8, 16(1)

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

GTREFCLKとline rateの関係（CPLL）

通信規格毎に推奨のGTREFCLK周波数とCPLLの設定が決まっている。
以下は一例。

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

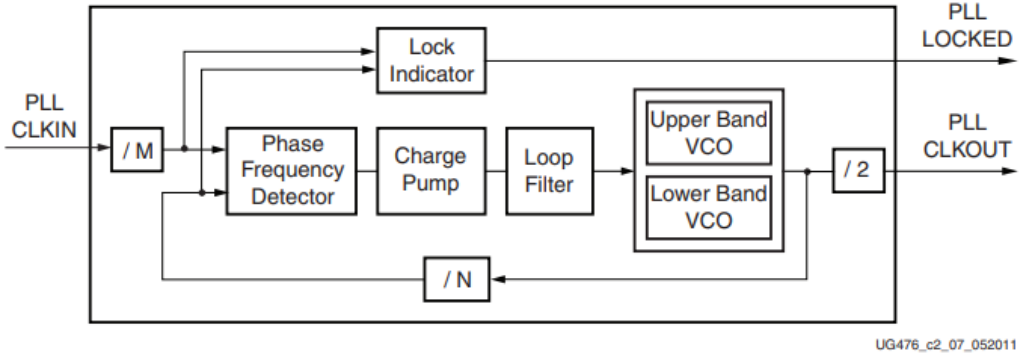
ラインレート 6.25 Gb/sまでしか表に無い。GTXは10 Gb/s以上までサポートしているはず。
より高ラインレートを実現するには**QPLL**を利用する。

UG476より
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**.

GTREFCLKとline rateの関係（QPLL）

Quad PLL

- GT quadにつき1つ存在するPLL。GTXE2 commonでラップする。
- トランシーバで6.6 Gbpsを超えるラインレートを出す場合に利用する。
- クアッドにつき1つなのでチャンネル毎にラインレートを変える用途には使えない。



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

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

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 QPLL_FBDIV_RATIO	16, 20, 32, 40, 64, 66, 80, 100 (see Table 2-16)
D	RXOUT_DIV TXOUT_DIV	1, 2, 4, 8, 16

CPLLと同様にパラメタを手動で設定することは
普通はしない。

GTREFCLKとline rateの関係（QPLL）

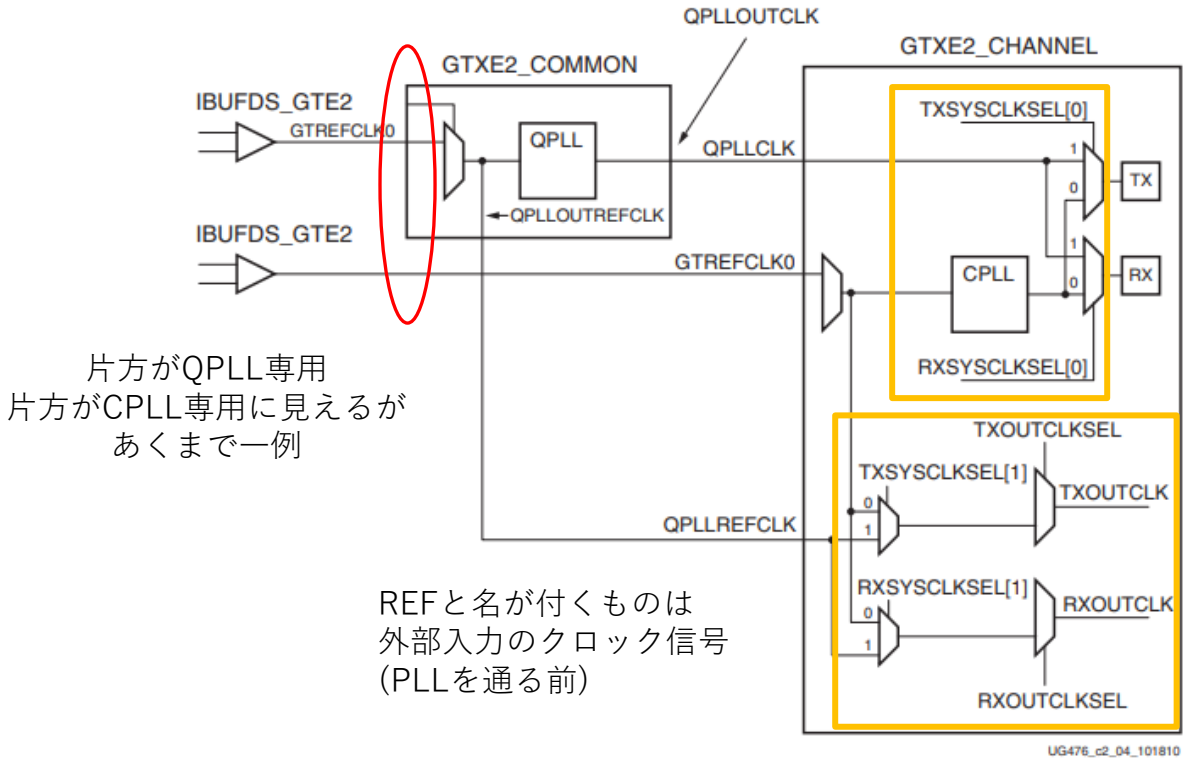
10 Gbps付近の高速通信規格で利用する。

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

PLLの出力がどのようにchannelへ到達するか

UG476に例として描かれている図



片方がQPLL専用
片方がCPLL専用に見えるが
あくまで一例

REFと名が付くものは
外部入力のクロック信号
(PLLを通る前)

TXSYSCLKSEL[0-1]
RXSYSCLKSEL[0-1]
CPLL/QPLLの選択を行う
- [0]: PCS/PMAへの配線
- [1]: TX(RX)OUTCLKへの配線
- [0][1]は同じ数字が入るべき

TXOUTCLKSEL
RXOUTCLKSEL
- 後述
- どの信号をTX(RX)OUTCLKとして
  GTXE2 channelの外に出すかを決める。
  重要。

ポイント
TX(RX)SYSCLKSELは自分が利用したいラインレートから推測できる
TX(RX)OUTCLKSELは複数の選択肢があり推測しきれない。
IPで生成したら確認した方がよい。

PLLの出力がどのようにchannelへ到達するか

Transmitter

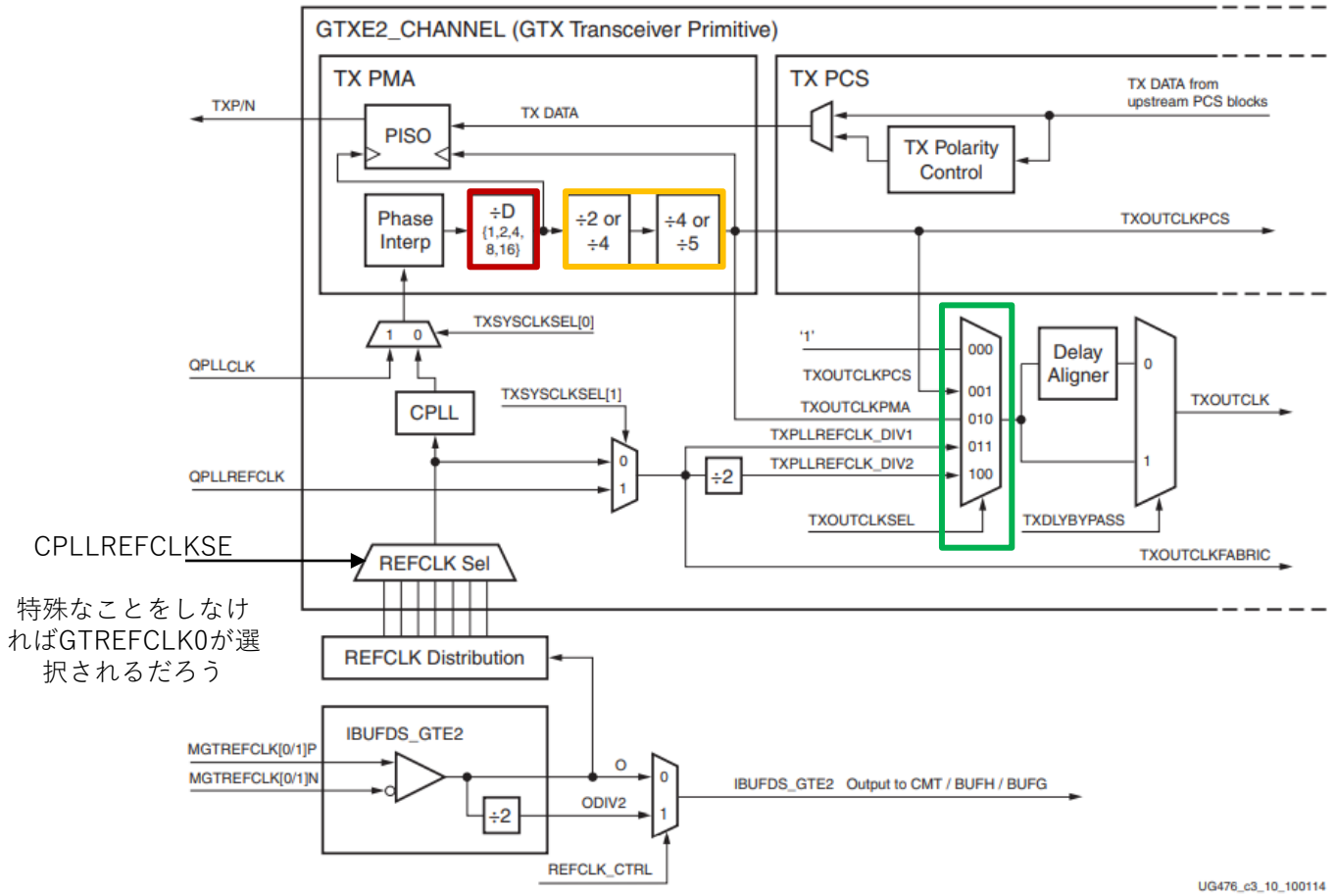


Figure 3-28: **TX Serial and Parallel Clock Divider**

特殊なことをしなければGTREFCLK0が選択されるだろう

÷D

PISOが指定のライレートになるように設定される。
推奨値はCPLLパラーメータテーブル参照。

÷2 or ÷4

内部データ幅を決めるTX_INT_DATAWIDTHの値によって決まる。

- 0 (2-byte幅): ÷2
- 1 (4-tybe幅): ÷4

6.6 Gbps以上のレートでは必ず1が選択される。

÷4 or ÷5

TX (RX) インターフェースのデータ幅を決める
TX_DATA_WIDTHの値によって決まる

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

TXOUTCLKSEL

- TXOUTCLK信号に何を使うか決める。アプリーケーションに依存。
- 後述のTXUSRCLKのソースであるため回りまわってPCSのクロックになる。

PLLの出力がどのようにchannelへ到達するか

Transmitter　　GbE用にIPでPCS/PMAを生成した場合

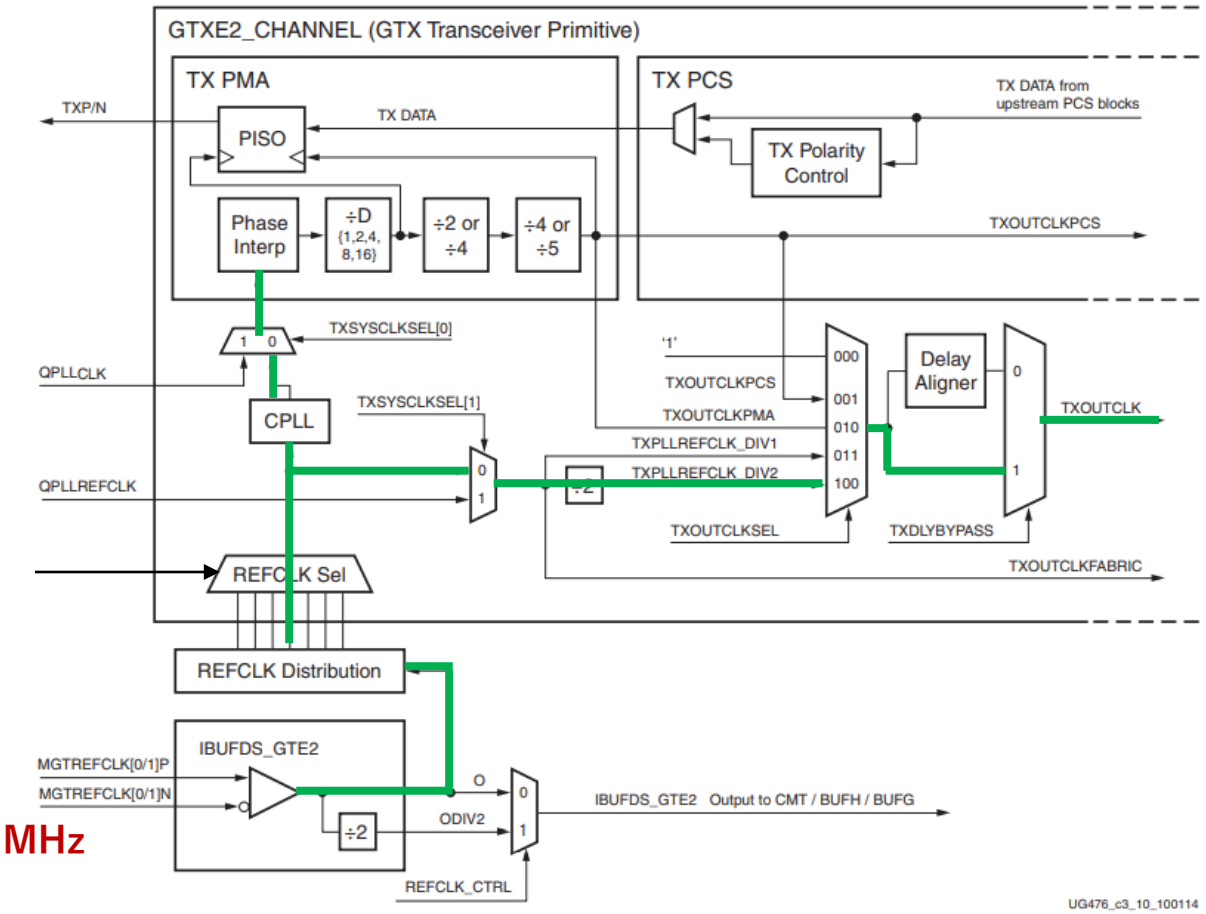


Figure 3-28: **TX Serial and Parallel Clock Divider**

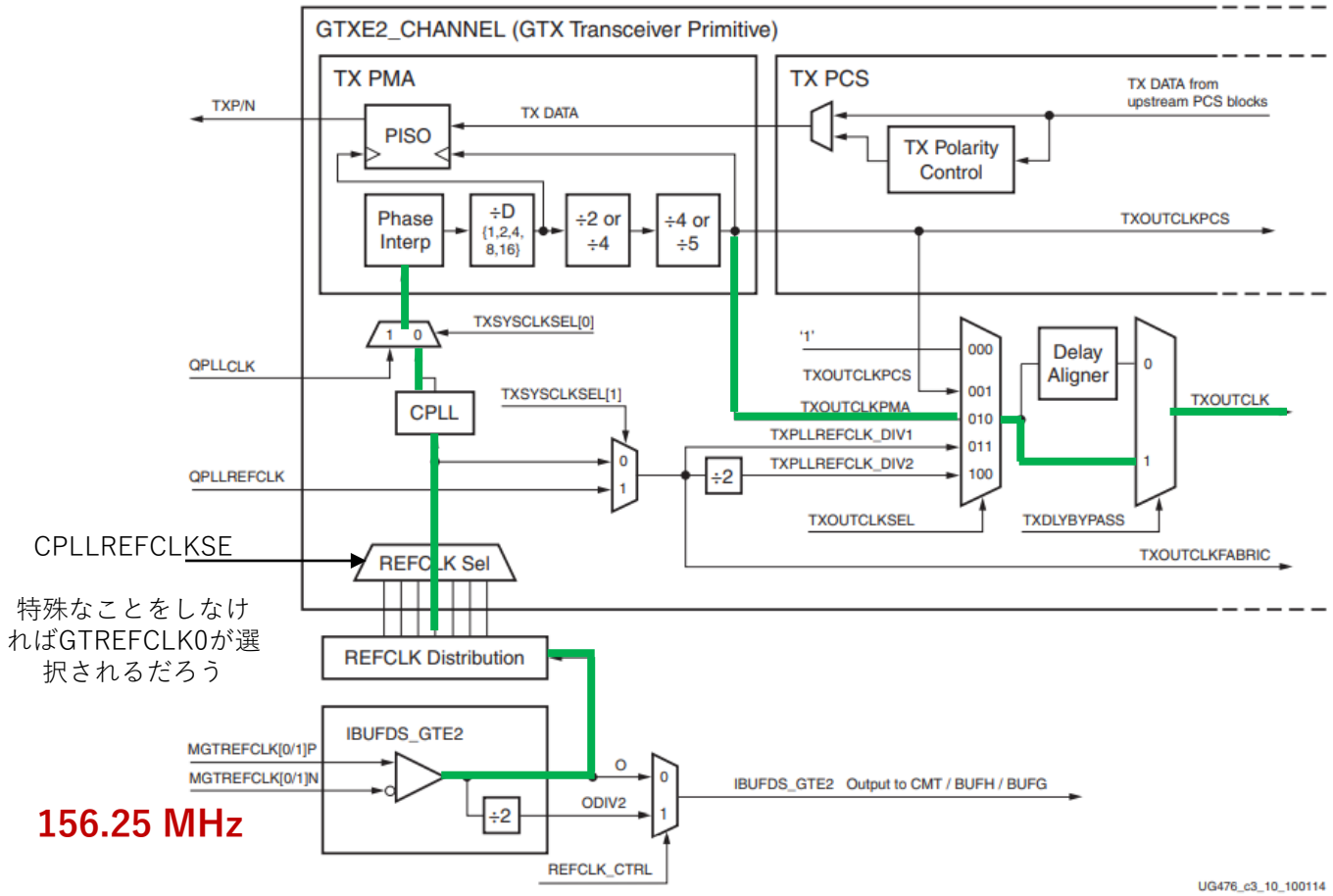
特殊なこと
ればGTREF(
択される

125 MHz

TXOUTCLKの周波数は?

PLLの出力がどのようにchannelへ到達するか

Transmitter　　Aurora8b10b, ラインレート1.25 Gbps, 2-byteデータ幅用にIPで生成した場合



特殊なことをしなければGTREFCLK0が選択されるだろう

156.25 MHz

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

CPLLパラメータ
- M: 1
- N1: 4
- N2: 4
- D: 4

TX_DATA_WIDTH
- 20

TX_INT_DATAWIDTH
- 0

TXOUTCLKの周波数は?

PLLの出力がどのようにchannelへ到達するか

Transmitter　　10GbE用にPCS/PMAを生成した場合

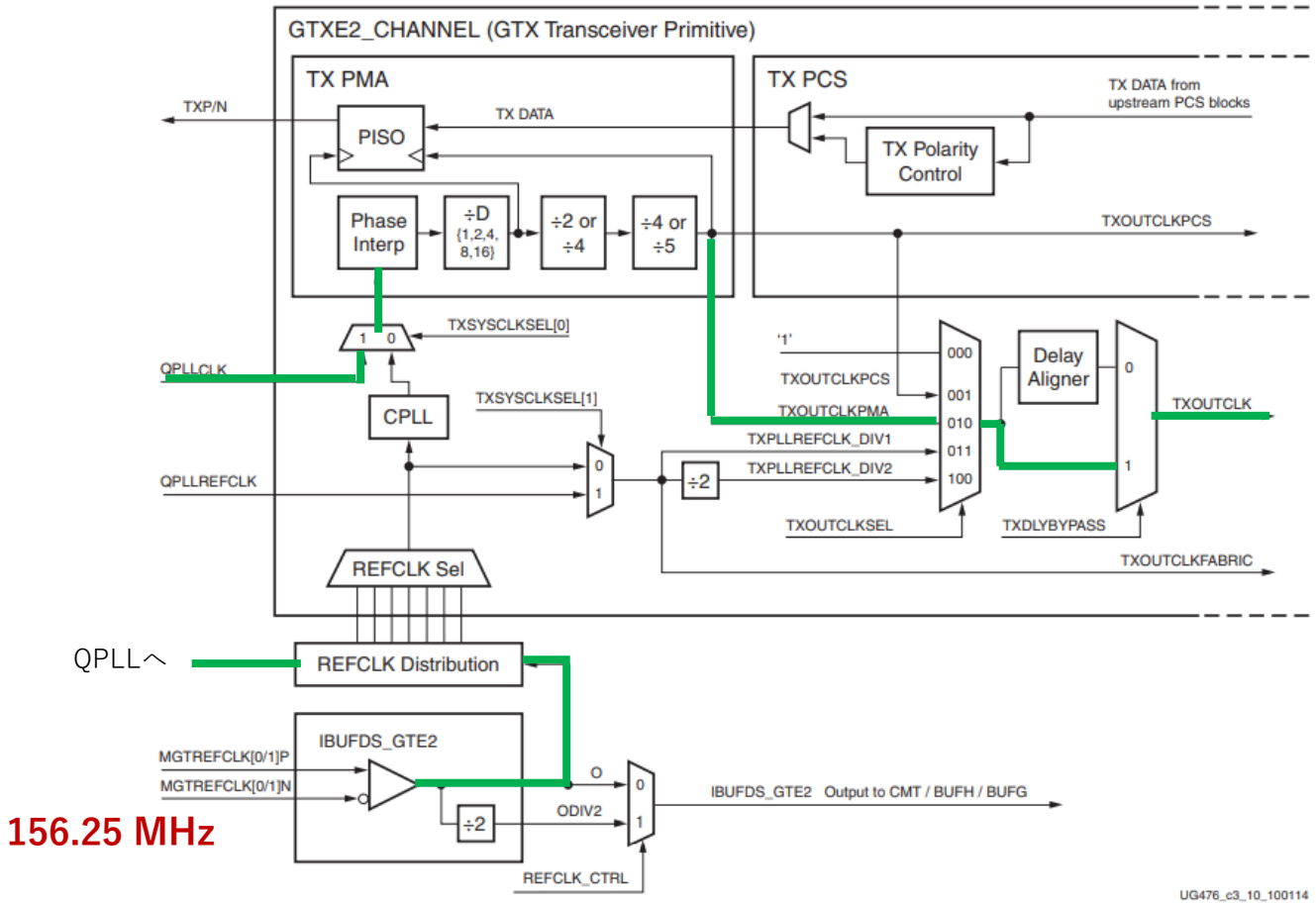


Figure 3-28: **TX Serial and Parallel Clock Divider**

QPLLパラメータ
- M: 1
- N: 66
- D: 1

TX_DATA_WIDTH
- 32

TX_INT_DATAWIDTH
- 1

TXOUTCLKの周波数は?

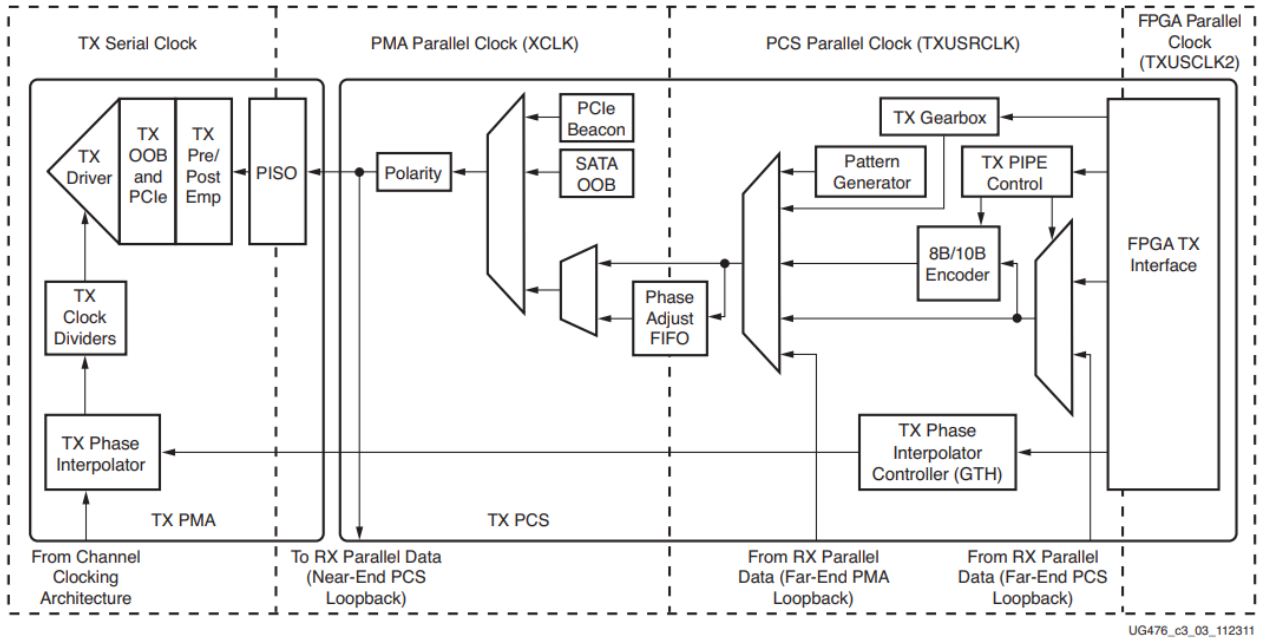
途中まとめ

GTREFCLKの信号からCPLLかQPLLを利用して
必要なtransmitterのラインレートが得られる。
TXOUTCLKの周波数が定まる。
この時点で分岐前のPCS XCLKの周波数も決まっている。

Transmitterのラインレートが決まったので
receiver側の復元クロックの周波数が決まる。
やはりここでもRX側のPCS XCLK周波数も決まっている。

ではPCSのもう一つのクロックドメインであるTX (RX) USRCLKの周波数は？

TX(RX)USRCLK



TX(RX)USRCLK
- PCSの内部クロック

TX(RX)USRCLK2
- TX (RX) インターフェースクロック

USRCLKの方は一意に周波数が決まる

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

すなわちXCLKも同様。
USRCLK2は？

表 3-3 : TXUSRCLK2 と TXUSRCLK の周波数関係

FPGA インターフェイス幅	TX_DATA_WIDTH	TX_INT_DATAWIDTH		TXUSRCLK2 の周波数
2 バイト	16、20	0	2-byte	$F_{TXUSRCLK2} = F_{TXUSRCLK}$
4 バイト	32、40	0		$F_{TXUSRCLK2} = F_{TXUSRCLK}/2$
4 バイト	32、40	1	4-byte	$F_{TXUSRCLK2} = F_{TXUSRCLK}$
8 バイト	64、80	1		$F_{TXUSRCLK2} = F_{TXUSRCLK}/2$

USRCLK2はUSRCLKと同一か
2分周の関係にある

TX(RX)USRCLK

TX(RX)USRCLKとTX(RX)USRCLK2の関係性 (UG476より)

- スキューが小さいクロックリソース (**BUFG および BUFR**) を使用してTXUSRCLKおよびTXUSRCLK2を駆動する必要があります。

- TXUSRCLK および TXUSRCLK2 は、**トランスミッターの基準クロックを逓倍または分周した周波数クロック**にする必要があります。

TX(RX)USRCLK

ここまでの事をまとめるとこのようになる。
すなわちtransmitterではXCLKとTXUSRCLKは同一
複数GTXE2 channelを実装する場合代表のTXOUTCLKを利用すればよい
(もちろん同一の通信規格の場合に限る)

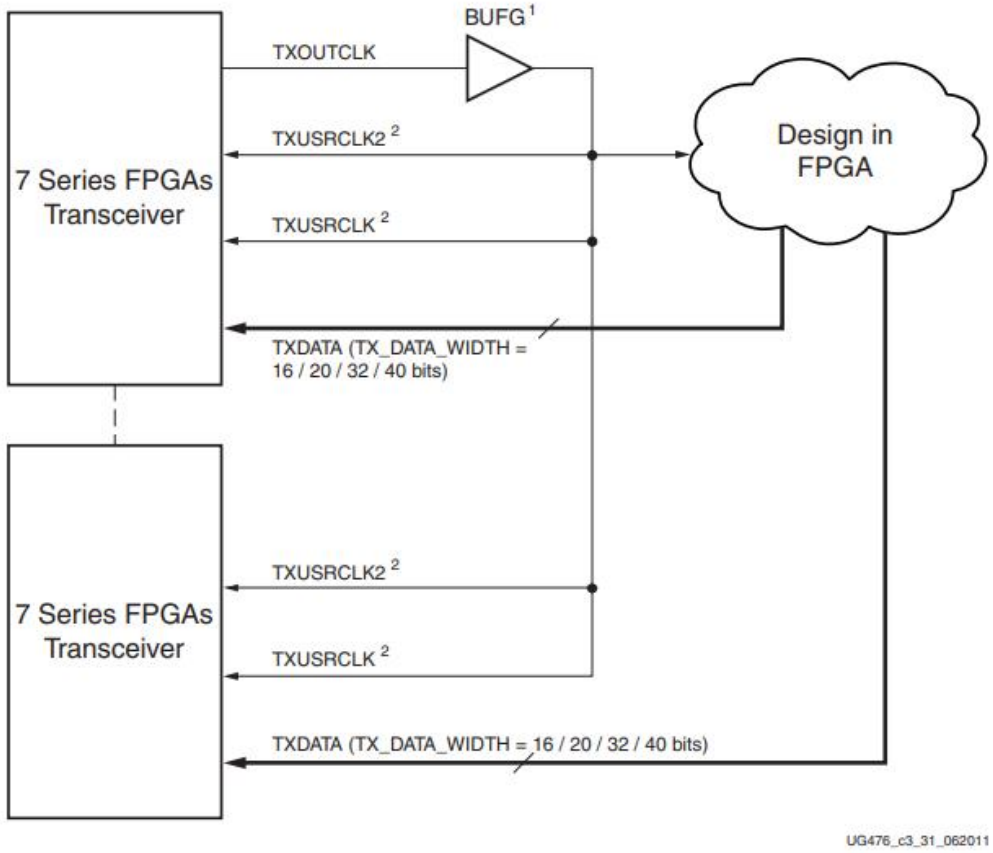


図 3-3 : マルチ レーン - TXOUTCLK を使用して TXUSRCLK2 を駆動
(2 バイトまたは 4 バイト モード)

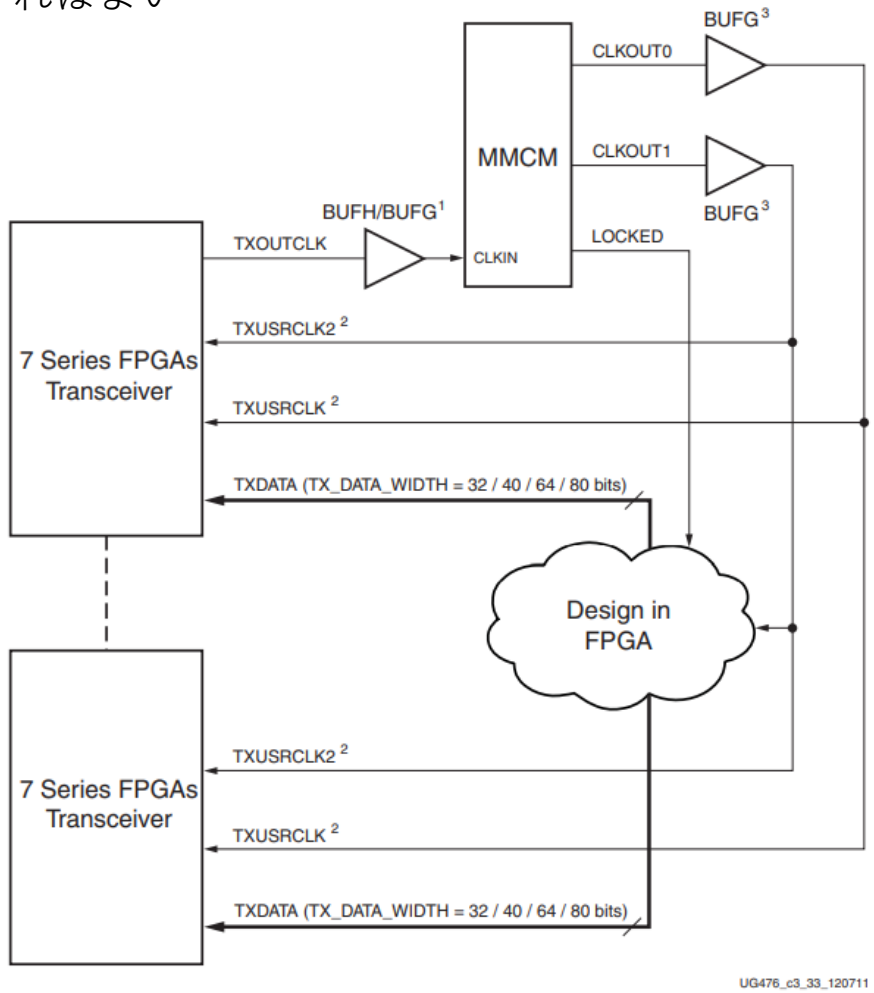
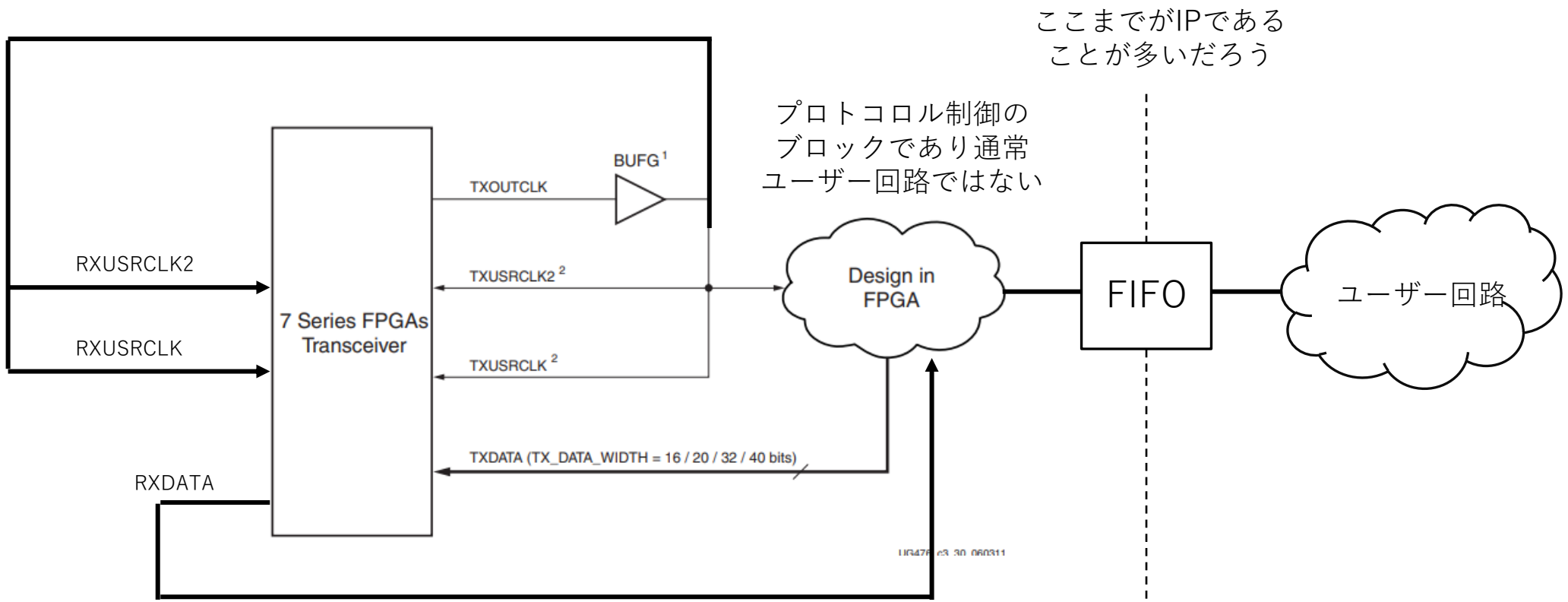


図 3-5 : マルチ レーン - TXOUTCLK を使用して TXUSRCLK2 を駆動
(4 バイトまたは 8 バイト モード)

TX(RX)USRCLK

RXを含めて考えてみる
送受信でラインレートは同様。
プロトコルを司るブロックのクロックもTXとRXで分ける事はないだろう。
そのため、RXUSRCLKとRXUSRCLK2はTXOUTCLKを元にするのが自然だろう。

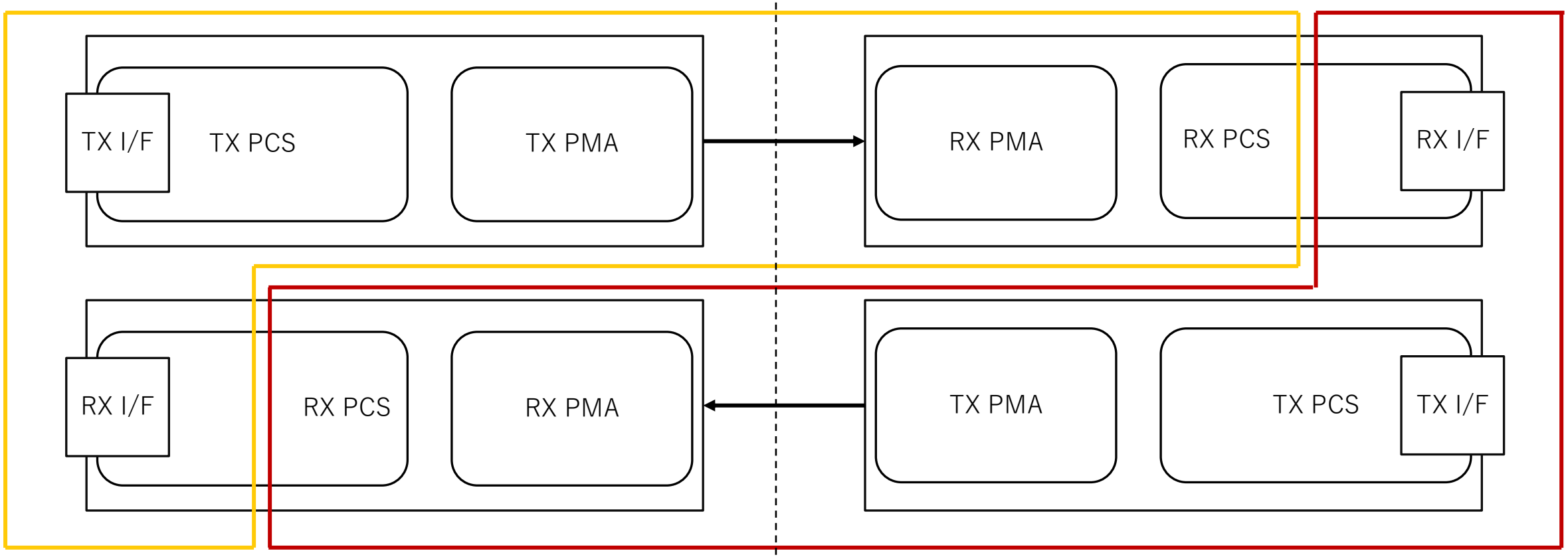


GTREFCLKの影響範囲

回路A上のGTREFCLK発振器に
依存する範囲

回路A

回路B



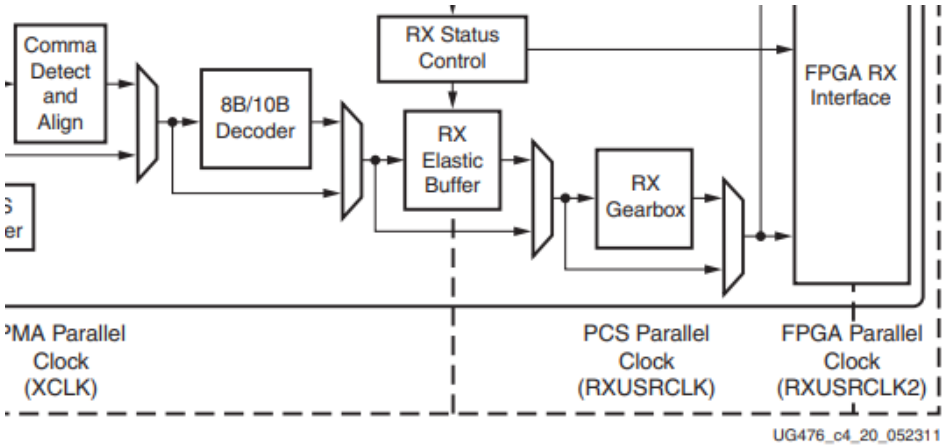
**ここで必ずクロック周波数の
微妙な不一致が発生する**

回路B上のGTREFCLK発振器に
依存する範囲

Elastic bufferの登場

Elastic bufferとclock correction

Clock correctionはIPが指定した設定を利用し自分で変更することは非推奨。
ブラックボックスでも良いがシリアル通信の基本要素なので動作は知っておきましょう。



RXUSRCLKの方がXCLKより速い
- バッファはいずれアンダーフローする

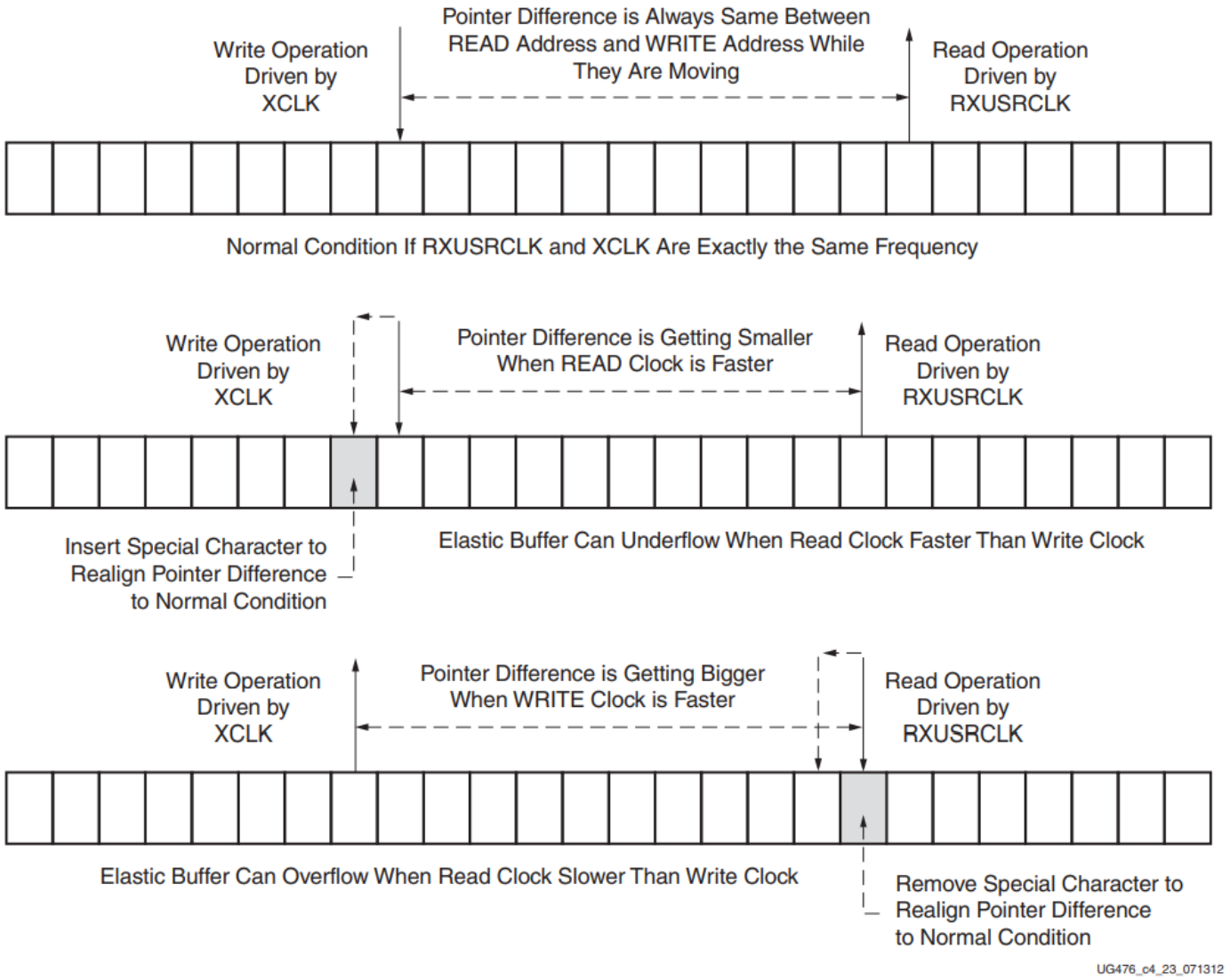
RXUSRCLKの方がXCLKより遅い
- バッファいずれオーバーフローする

ではどうするか?

TX側が特殊なデータを一定間隔で送信 (ユーザーのデータ列に割り込ませる)
- バッファがアンダーフローしそうなら**特殊データを複製する**
- バッファがオーバーフローしそうなら**特殊データを削除する**

この動作をclock correctionを呼び、その機能を提供するのがelastic bufferである。
ユーザー回路側では受信データレートが自身のTXラインレートとある範囲内で一致しているように見える。
XilinxはUGにおいて回路間のGT基準クロックを±100 ppm内に揃える、と表現している。

Elastic bufferとclock correction



Clock correctionはelastic buffer内部のwrite pointerとread pointerの差を一定に保とうとする機能。
すなわち、elastic bufferにはある範囲内のread latencyが存在する。
Trigger回路ではこのlatencyが問題となる場合がある

GTREFCLKを含めてすべてのクロックの同期を取っている実験はRX elastic bufferをバイパスするのが1つの目的だろう。

逆にlatencyを許容できるのであればclock correctionによりある程度の同期性が保たれるため完全なクロック同期は必ずしも必要ではないだろう。

ここまでのまとめ

(ウィザードでIPを生成するとデフォルトで選択される構成)

- GTREFCLKからTXOUTCLKが生成され各種ユーザークロックはこれがもととなる。
- GTREFCLKの影響範囲は通信先のRX PCSのXCLKまで。
- RX elastic bufferが周波数の違いを吸収しある範囲内で同期がとられる。
- RX elastic bufferにはread latencyが存在する。

Aurora 8b10bを使ってみる

Auroraとは

Aurora 8B/10B は、ザイリンクス トランシーバーを簡単にインプリメントし、軽量ユーザー インターフェイスを提供してその上にシリアル リンクを構築できるようにする LogiCORE IP です。Aurora 8B/10B は、高速シリアル通信向けのスケーラブルで軽量なリンク レイヤー プロトコルです。

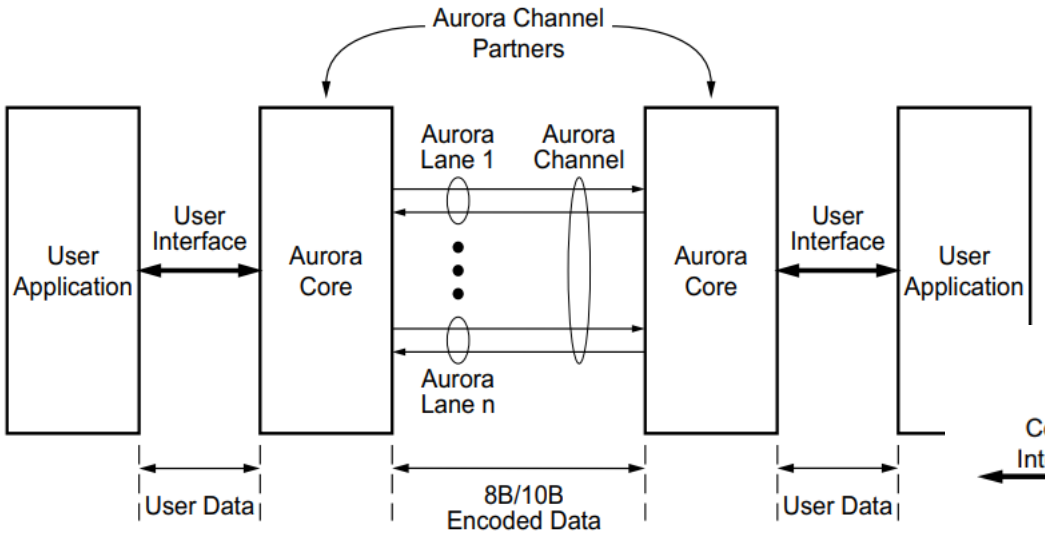
Xilinx のウェブページより

PCS/PMAはPHYであるため何かしらプロトコルがないと会話ができない。
Auroraはトランシーバを利用するための軽量でシンプルなプロトコル。
フレーム構造をもったデータとデータストリーミングの両方に対応している。

コアの概要					
サポートされるデバイスファミリ(1)	UltraScale アーキテクチャ、Zynq-7000、7 シリーズ				
サポートされるユーザー インターフェイス	AXI4-Stream				
リソース(2)	LUT	FF	DSP スライス	ブロック RAM	最大周波数(3)
Config1(3)	342	463	0	0	330MHz

PG046

Aurora 8b10b core



AuroraのIP coreは複数のGT channel (lane) を束ねて1つの通信ブロック (channel) を構成することができる。

例: 2-byteデータ幅で動いているlaneを4つ束ねて8-byteのユーザーインターフェースを提供する。
⇒ ユーザー回路から見たスループットが向上する。
⇒ 単一のトランシーバで到達できないデータレートも可能に。

AuroraのデータバスはAXI4-Streamというオンチップ・バス規格を採用している

Xilinxの他のIPで使われており覚えて損はない。
Zynq等でプロセッサと会話するときに特に重要。

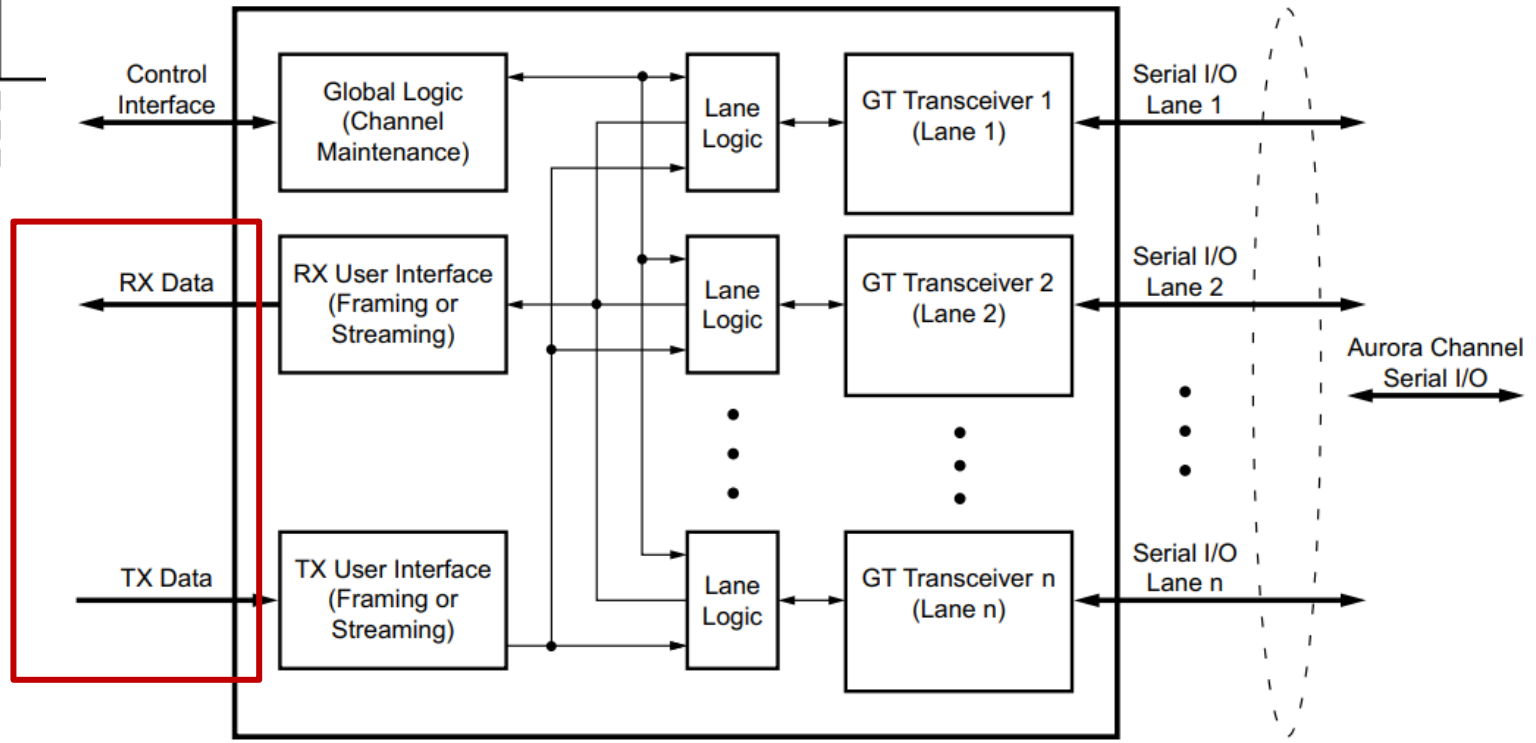


図 2-1 : Aurora 8B/10B コアのブロック図

AXI4-stream

AMBA 4.0のAXIインタコネクタ・プロトコル・ファミリに属するバス規格
AMBAは元々ARM用のバス規格であり無償ライセンスで公開されている。

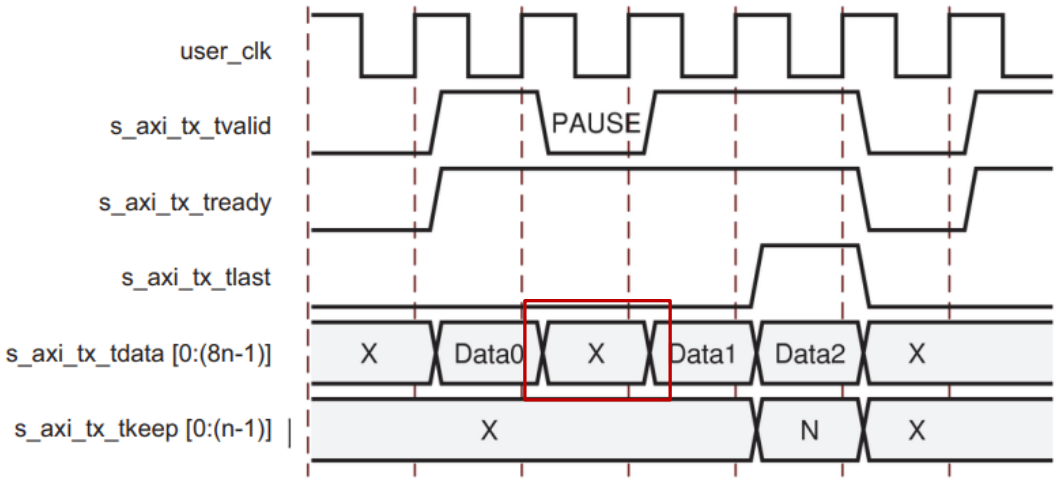
今回はAuroraプロトコルにフレーム構造があるモードについて説明する。
AXI4-Streamにはmaster/slaveの考えがありtvalidを出力する方がmaster。

TXならデータソースのユーザー回路がmaster。
RXならAurora coreがmaster。

信号種別	説明
tvalid	MasterからのAXI4-Stream信号が有効である事を示す信号。(Active High) Auroraのデータインターフェースの場合tdataが有効であることを示す。
tready	Slaveがデータを受信可能であることを示す信号。(Active High) tvalidがhighでこの信号がlowであった場合masterはデータのストレッチを行う必要がある。 TX側にのみ存在。
tlast	ユーザーデータフレームの終わりを示す。(Active High)
tkeep	フレーム中の最後のデータのうち有効なバイトコンテンツを示す。tlastがhighの時だけ有効。 TXならbyte write enable, RXならbyte read validと思えばよい。
tdata	ユーザーデータ

AXI4-stream

データ送信のタイミングチャート

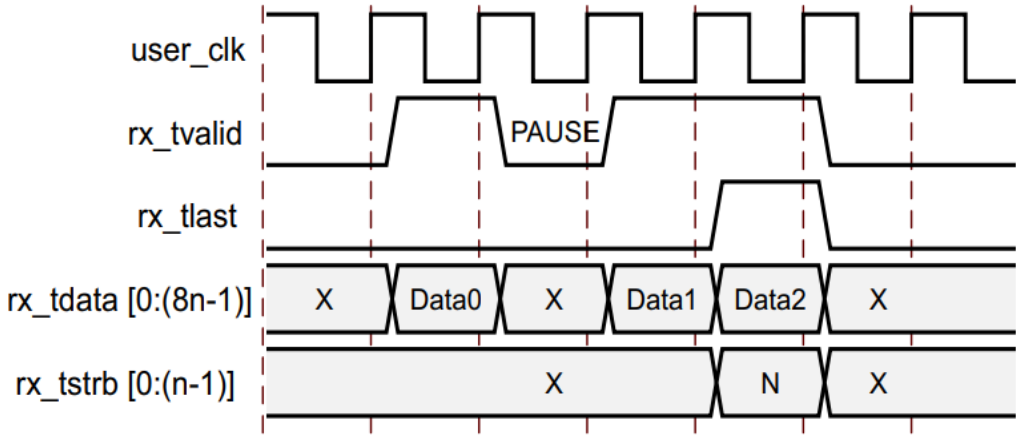


フレーム送信中のtvalidディアサートはポーズを示す。
フレーム送信の途中だが送信データがアイドル文字に
変更される。

この例ではData0, Data1, Data2の3ワードが
ユーザーデータの1セットであり
Auroraのフレームにカプセル化される。

フレーム長は任意でありtlastがアサートされるまで
Auroraフレームの終端文字は挿入されない。

データ受信のタイミングチャート

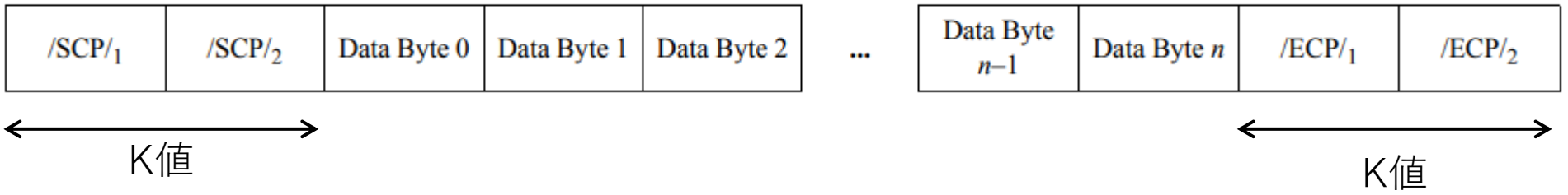


rx_tvalidがhighの時、tdataが有効である。
ユーザーフレームの終わりにtlastがアサートされる。

Aurora 8b10b frame

Auroraのフレームはとてもシンプル。

表 3-6 : 標準的なチャネル フレーム



tlastアサートで挿入される

Aurora 8b10b core

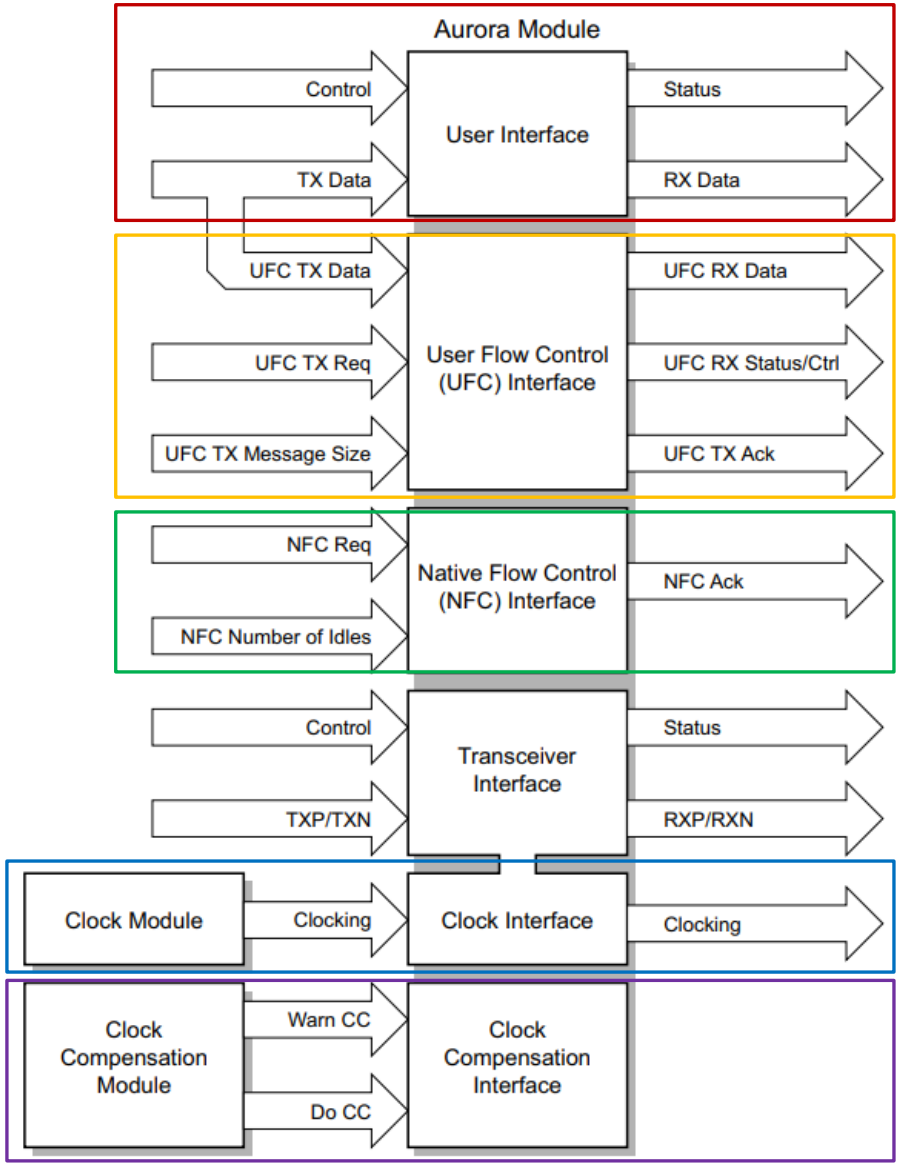


図 2-3 : 最上位インターフェイス

今説明した部分

Framing modeの時選択できるオプション機能
ユーザーデータを送信中に他の短いメッセージを割り込ませることが出来る。
UFCに書き込んだデータは優先的にlaneに流れる。

Framing modeの時選択できるオプション機能
データ送信中にリクエストのあったタイミングで指定長さアイドル文字を挿入し受信側のレートをコントロールする機能。
受信側の処理能力が高くない時に利用する。

PCS/PMAの知識があれば正しく記述できるが、
example designにまずは従うのが無難。

Clock correctionの制御を行う。
Coreの外には出てこない。

Auroraのオプション

CRC (巡回冗長検査)
- 誤り検出符号の一種。データ送受信時のデータ破損検出。

スクランブラ・デスクランブラ
- 疑似ランダム系列を用いてデータを乱数に攪拌する (TX側)
- 攪拌されたデータを元に戻す (RX側)

- 特定のビットパターンが連続することにより特定周波数のEMIが強くなることを避ける。
- 8b10bのランニングディスパリティによってDCバランスはとれているが特定パターンを避けられているわけではない。

ここから発展演習用のスライド
初日は一旦スキップ

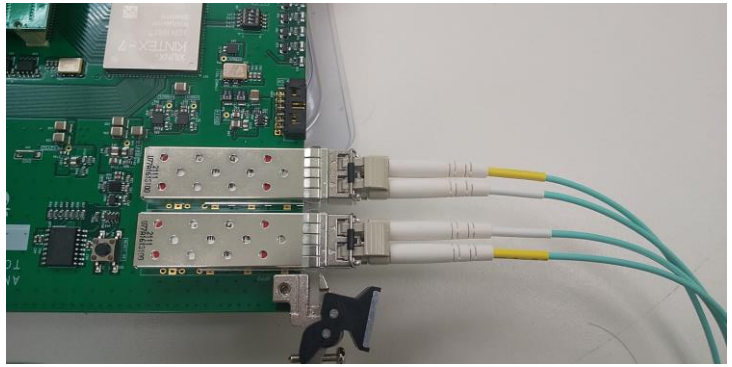
Aurora8b10bを実装してみましょう

実装仕様

- AMANEQに刺さっている2つの光モジュールをファイバでつなぎループバックさせる。
- Simplexを選択し、TXのみのコアとRXのみのコアを1つずつ生成する。
  - Back channelにsidebandsを選択しTXとRXを接続する。
  - (Full-duplex core内部で初期化時に起きていることを確認することが出来る)
- TXからはインクリメンタルデータを常時送信する。
- 16-byteをユーザーフレーム単位とし、8クロックごとにtlastをアサートする。

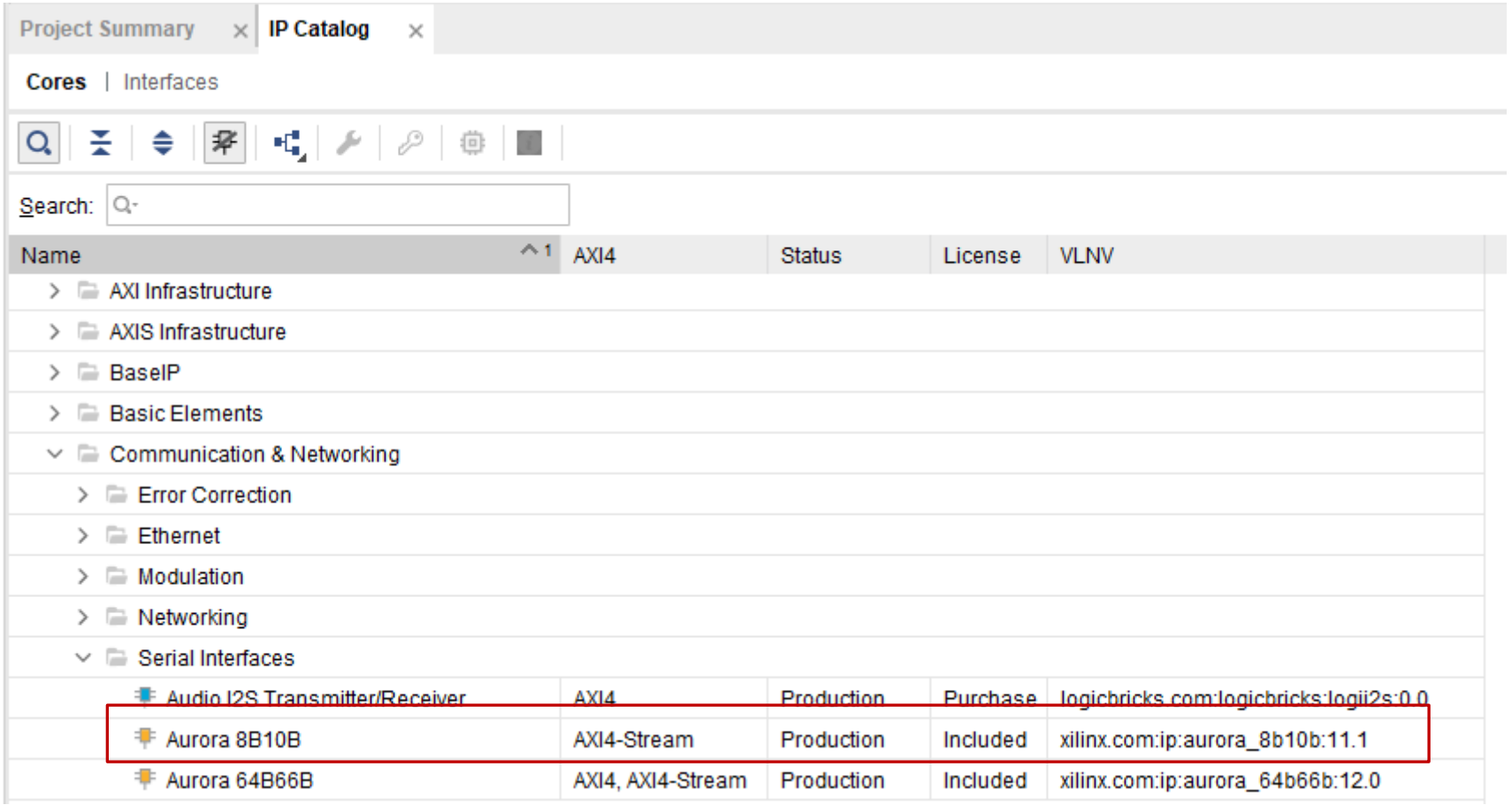
Aurora coreの条件

- GTREFCLKの周波数: 156.25 MHz
- データ幅: 2-byte
- ラインレート: 1.25 Gbps
- INIT CLK: 100 MHz
- DRP CLK: 100 MHz
- UFC, NFC, スクランブル, CRCは無し



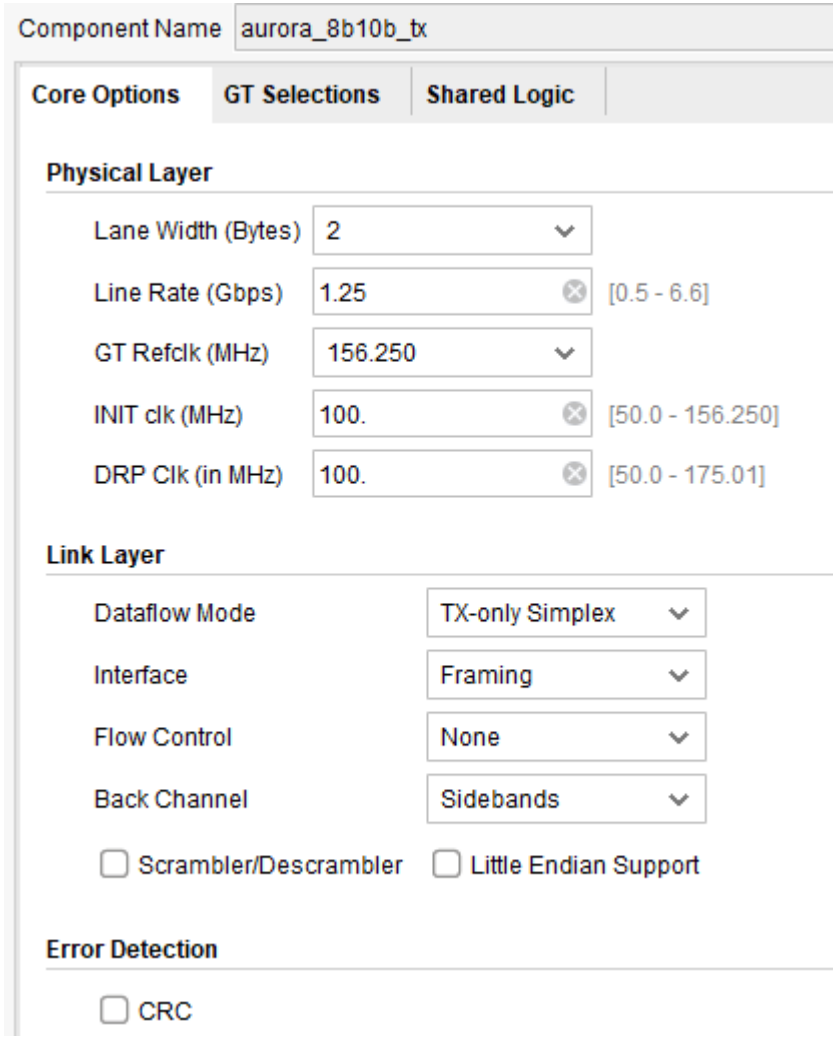
GTトランシーバが初めての人は自分一人で実装するのはちょっと難しいと思います。
途中までやってみます。

Aurora8b10bを実装してみましょう



Aurora8b10bを実装してみましょう

まずはTX側



INIT clk

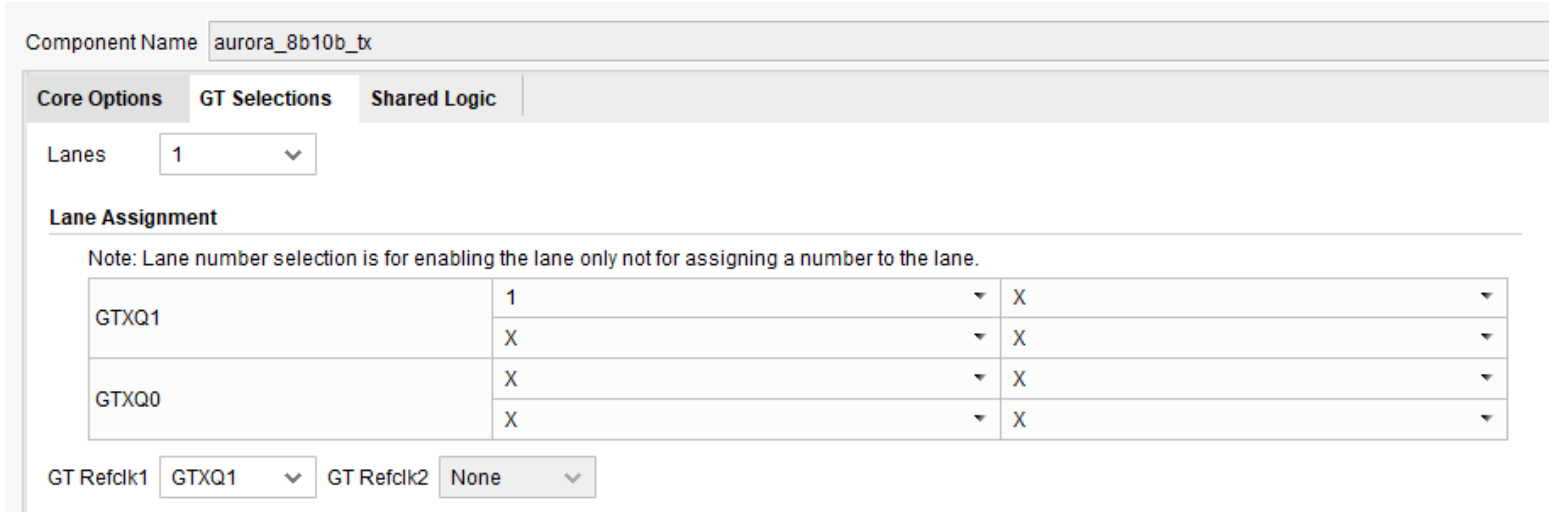
- トランシーバリセットが入るとTXOUTCLKが失われてしまう。その間使われるクロック。
- ホットプラグ機能をサポートするために必要。
- トランシーバと独立のクロックソースにつなぐ。
- 今回は発振器に直付けする。

DRP clk

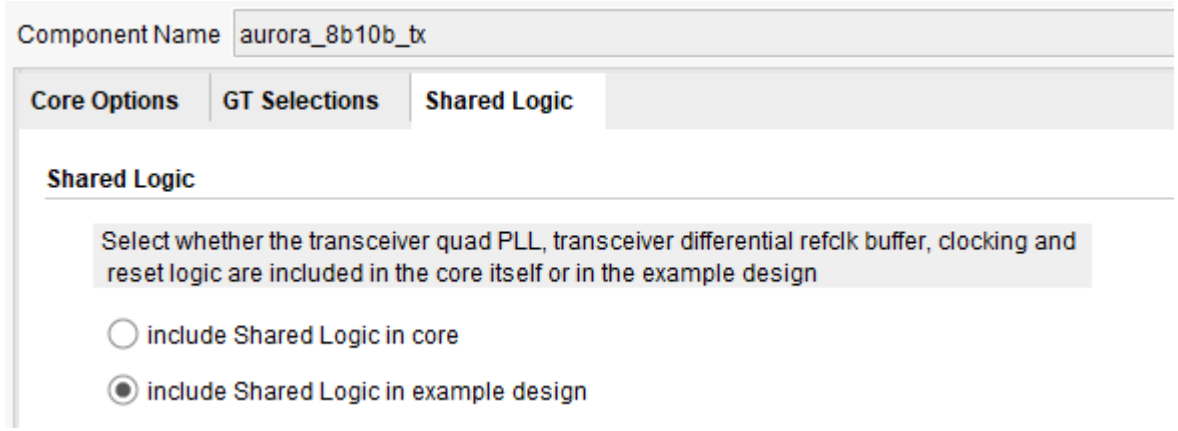
- GTX channelの制御ポート。
- 使わずとも通信できる。
- 今回はINIT clkと同じクロックをつないでおく。

Aurora8b10bを実装してみましょう

まずはTX側



今回channelあたり1 laneしか実装しないため深く考えなくてよい
後から制約でリソース位置の指定は行う。



AuroraのIP coreをサポートする様々な周辺回路をIP coreに入れてしまうか、
モジュールにして自分で実装するか。
自分でいじれるように後者を選んだ方がよい。

Aurora8b10bを実装してみましょう

RX側

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.

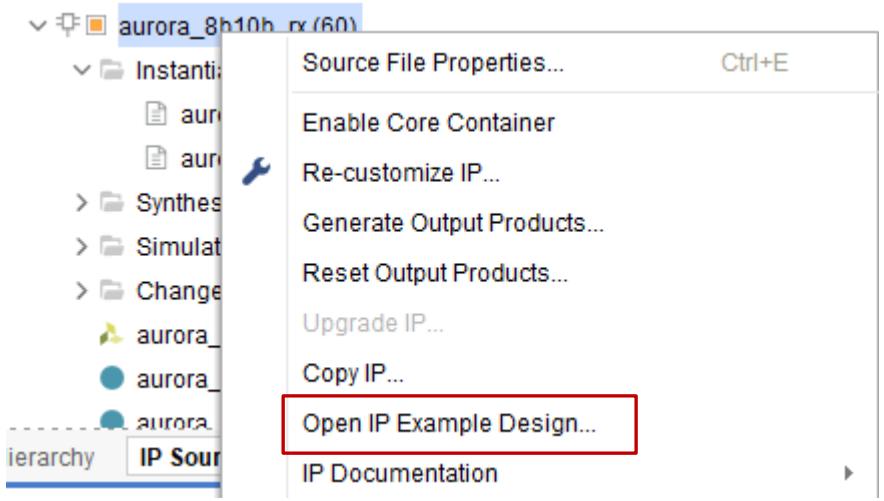
GTXQ1	1	X
	X	X
GTXQ0	X	X
	X	X

GT Refclk1: GTXQ1 GT Refclk2: None

Aurora8b10bを実装してみましょう

Instantiation templateから実装してみましょう
と言いたいところですが…出来る気がしますか？
多分無理だと思います。

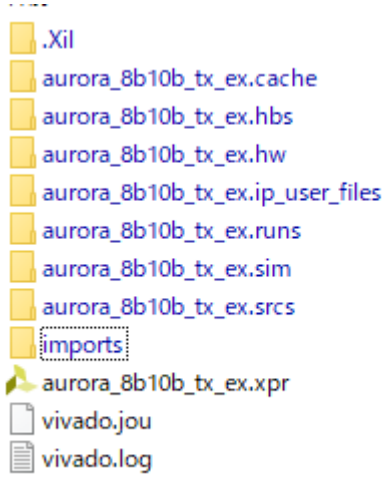
トランシーバ系のIPはexample designを生成して
それを参考にするのがよいでしょう。



Example designのVivadoプロジェクトを生成します。
生成先を指定するとVivadoがもう1つ立ち上がります

Aurora8b10bを実装してみましょう

Example designの中身



.srcs

- IPのHDLコード。
- Auroraはcoreの全てのコードが生成されブラックボックスが一切ない。
- ソースコードを見ると分かるように、FPGAの専用リソースを使っておらずすべてファブリックで実装できるようになっている。
- 自作のプロトコロルを作る時に参考になるだろう。

imports

- Example designのソースコード。
- 何やらたくさんあるがsupportと書かれたコードが最も重要。
- supportモジュールに接続されているモジュールが実際動かすのに必要なリソースであり、それらを探して移植すればよい。
- ある程度自分の回路に合わせてコードの変更は必要。

発展課題H2で何をしたらいいのか分からない人へ

- importsの中から何をコピーしてくれば良いかリストしましょう
- 演習手順書H2のブロック図を参照しコピーしたコードのどこを変更するべきか検討しましょう
- 手順書のブロック図の通りになるようにクロックを配線しましょう
- それでも正解の配線が分からない場合教材ソースコード（答え）を参照しましょう