

# FPGA 中級トレーニングコース 演習手順書

2022.06.27  
KEK IPNS E-sys  
本多良太郎

## この演習で利用する機材

### AMANEQ

(A main electronics for network oriented trigger-less data acquisition system)

AMANEQ は J-PARC 実験向けに開発された汎用エレクトロニクスです。高機能の FPGA である Xilinx Kintex-7 を搭載し、多数の汎用 IO、DDR3-SDRAM、および 10 Gbps の光ファイバー通信を搭載した回路です。本演習ではこの回路上の FPGA にダウンロードするファームウェアを皆さんに記述してもらい、実際に動作させて動きを確認してもらいます。

AMANEQ の詳しい説明は Open-It プロジェクトページを参照してください。物理学会で利用した資料がアップロードされています。

<https://openit.kek.jp/project/StrHRTDC>

### AMANEQ の写真と演習で利用する機能

図 1 に AMANEQ の写真を示します。この演習では AMANEQ への電源供給は AC/DC アダプタで行います。配布したアダプタをジャックへ接続してください。

AMANEQ にはスピードグレード 2 の Kintex-7 FPGA が搭載されています。EX1 と EX3 では FPGA 外部との通信は行わず、内部の動作を ILA で確認します。AMANEQ の左側に接続した子基板の配線パターンは直接 FPGA の PAD へつながっています。EX2-1 と EX2-2 では子基板上で信号ループバックを作って、IOSERDES の動作を確認します。この回路には 2 つの SFP+ module のスロットがあります。それぞれ、multi-gigabit transceiver (MGT) bank に配線されています。これらを使って、高速シリアル通信の動作を EX4 で学びます。

## 例題における共通事項

Vivado project について

利用 FPGA: xc7k160tffg676-2

(スピードグレード-2 の XC7K-160T FPGA。パッケージは FFG 676 ピン。)

### HDL ファイル

ファイル名を toplevel.vhd もしくは toplevel.v として、階層構造を使わずにコードしてください。(EX4 で example design から移植するモジュールを除く)

### 制約ファイル

pins.xdc、timing.xdc、impl.xdc の 3 種類の制約ファイルを生成し、ピンに関する制約を pins.xdc、タイミング制約を timing.xdc にそれぞれ記述してください。また、impl.xdc を target constraint file に指定し、set up debug 実行時に ILA の記述が書き込まれる制約ファイルを impl.xdc にしてください。

## トップレベルポートピン

CLKP, CLKN

- ・発振器入力です。周波数は 100 MHz。
- ・LVDS\_25 規格を制約してください。

RSTB\_SW

- ・基板上のプッシュボタン(SW2)入力です。
- ・プッシュボタンを押下すると 1 から 0 へ切り替わる信号です。
- ・LVCMOS33 を制約してください。

## リセット論理について

- ・Active high とします。RSTB\_SW 信号はインバートする必要があります。

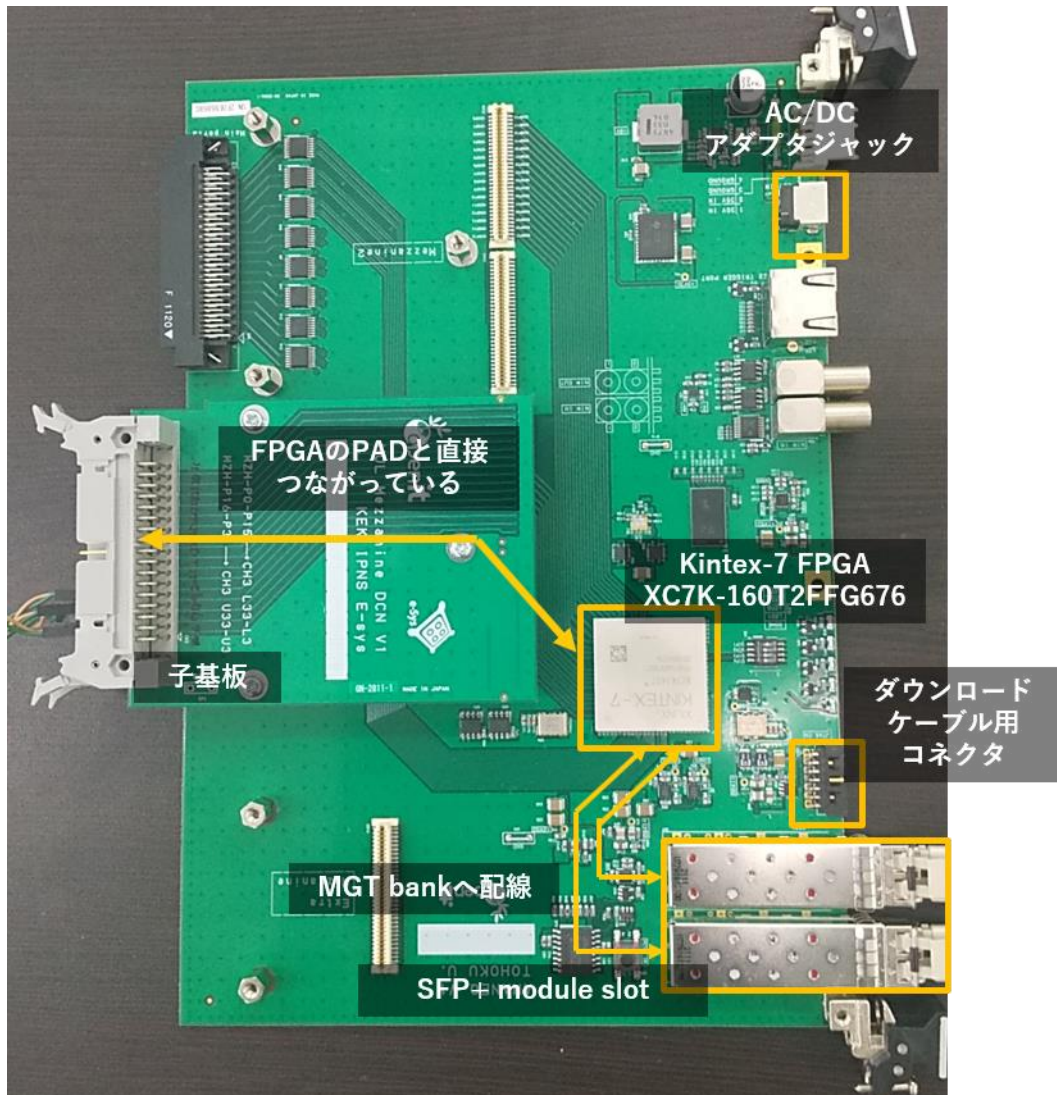


図 1 : AMANEQ の写真と演習で使う機能

# 例題 EX I

## DSP による移動平均フィルタの実装

### ファームウェア構成

図 2 に例題 EX I のブロック図を示します。破線で囲まれた部分はトップレベルポートです。FPGA のパッドに接続されます。RSTB\_SW 信号はインバータを挟んだのち 2 重 FF で同期を取ってください。同期後の信号をリセット信号として各モジュールへ配布してください。

DSP への入力（整数部）は 8-bit のフリーランカウンタで生成してください。適切な遅延素子を挿入した後 DSP の A および D ポートへ接続します。その際、下位 2 bit (小数部) を拡張して 0 埋めしてください。

CLKP, CLKN 入力をグローバルクロックとして利用してください。  
星マークの付いた信号線に mark\_debug 制約を与えて ILA へ配線されるようにしてください。

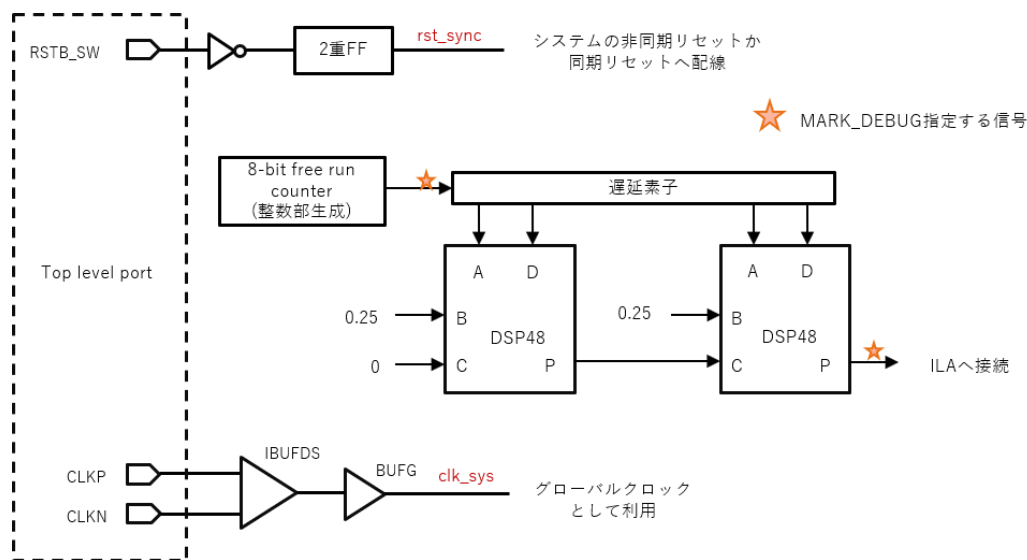


図 2：EX I のブロック図

### 設計条件

DSP 含め同期回路は clk\_sys のドメインに属するとします。入力クロックの制約を timing.xdc に記述してください。

トップレベルポートのピン制約を以下のように与えてください。

ポート名	ピン番号	IO 規格	その他制約
CLKP	F22	LVDS_25	DIFF_TERM
CLKN	E23	LVDS_25	DIFF_TERM
RSTB_SW	H9	LVCMOS33	

DSP への入力オペランド

- ・ 符号付き固定小数点 (整数部 8-bit, 小数部 2-bit)
  - ・ DSP は入力オペランドを符号付きだと認識します。自分で 2 の補数を計算しましょうという意味ではありません。
- ・ 整数部はカウンタで与え、少数部は入力では 0 とします。
  - ・ 余裕があれば他の入力パターンも試してみましょう。

DSP 生成の条件

実装する instruction

- ・  $(A+D)*B+C$

ポート幅

- ・ A, B, D: 10 bit
- ・ C, P: 48 bit

パイプラインレジスタの設定



Custom Pipeline options										
Tier:	1		2		3		4		5	6
	<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>
D	<input type="checkbox"/>	→	<input type="checkbox"/>	→	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>			
A	<input type="checkbox"/>	→	<input type="checkbox"/>	→	<input checked="" type="checkbox"/>					
B	<input type="checkbox"/>	→	<input type="checkbox"/>	→	<input type="checkbox"/>		<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	
CONCAT	→				<input type="checkbox"/>		<input checked="" type="checkbox"/>		<input type="checkbox"/>	
C	<input type="checkbox"/>	→	<input type="checkbox"/>	→	<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>	<input checked="" type="checkbox"/>
CARRYIN	<input type="checkbox"/>	→	<input type="checkbox"/>	→	<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>	
SEL	<input type="checkbox"/>	→	<input type="checkbox"/>	→	<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>	
KEY:	Fabric register									
	DSP register									

図 3 : DSP のパイプラインレジスタの設定

動作チェック

- ・ 移動平均フィルタの出力が期待する値になっているか確認してください。
  - ・ 入力に対する出力の遅延が正しく 5 になっているか確認してください。
- 乗算結果は整数部 20 bit、小数部 4 bit です。出力 P のうち整数部と小数部にわけて ILA へ配線すると確認がしやすいでしょう。

# 例題 EX2

## IOSERDES の実装 (IDELAY 無し)

### ファームウェア構成

例題 EX2 のブロック図を図 4 に示します。この例題では発振器のクロック信号から MMCM を利用して 100 MHz と 400 MHz のグローバルクロックを生成します。生成した 2 つのクロックは OSERDES を駆動するために利用し、100 MHz のクロックはパラレルデータ取り込み、400 MHz のクロックはシリアルデータ送信のためにそれぞれ利用します。OSERDES を DDR モードで駆動するので信号伝送スピードは 800 Mbps です。OSERDES へのパラレルデータ入力へは固定のパターンを入力してください。OSERDES の出力は直接トップレベルポートへ配線します。SDOP (N) は基板上でケーブルにより SDIP (N) へループバックします。

クロックフォワード信号 (CLK\_FWD\_O\_P (N)) も FPGA 外でケーブルにより CLK\_FWD\_I\_P (N) へループバックします。この信号は OSERDES へ入力した 400 MHz のクロックなので、BUFIO と 4 分周設定の BUFR とで 400 MHz と 100 MHz のクロックを生成します。これらのクロックは BUFG を通過していないためグローバルクロックではありません。生成した 100 MHz のリージョナルロックは ISERDES の他に、VIO とエッジ検出回路を駆動します。

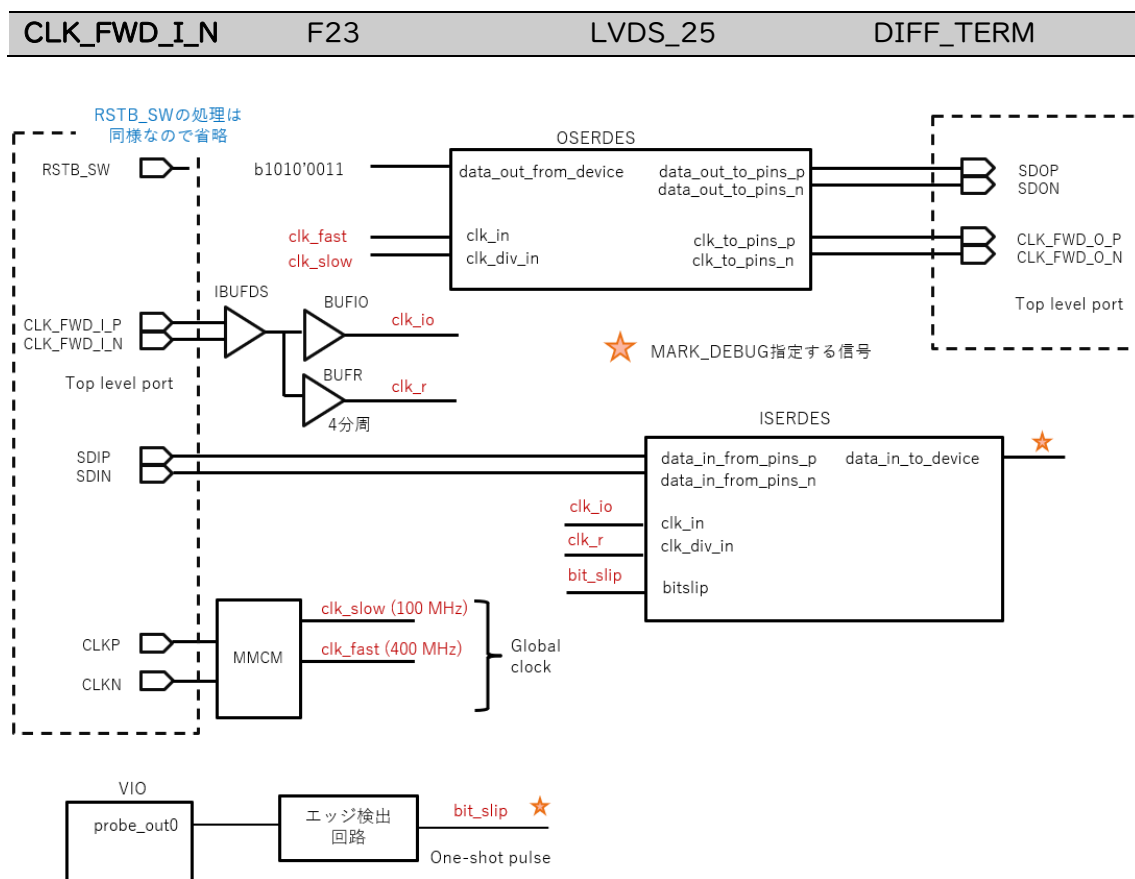
ISERDES は bitslip 入力が 1 の間ビットスリップを実行します。1 回ずつビットスリップの様子を確認するためにはワンショットパルスを入力する必要があります。VIO にはワンショットパルスを出力する機能がないため、probe\_out0 の 1 bit 目の信号の立ち上がりを検出し、エッジを取り出す回路を作成してください。

### 設計条件

送信側の回路は 100 MHz のグローバルクロック、受信側の回路は 100 MHz のリージョナルクロックドメインに属するとします。発振器の 100 MHz クロックとループバックするフォワードクロック 400 MHz を timing.xdc 内に定義してください。

トップレベルポートのピン制約を以下のように与えてください。

ポート名	ピン番号	IO 規格	その他制約
CLKP	F22	LVDS_25	DIFF_TERM
CLKN	E23	LVDS_25	DIFF_TERM
RSTB_SW	H9	LVCMOS33	
SDOP	D26	LVDS_25	
SDON	C26	LVDS_25	
CLK_FWD_O_P	H23	LVDS_25	
CLK_FWD_O_N	H24	LVDS_25	
SDIP	G24	LVDS_25	DIFF_TERM
SDIN	F24	LVDS_25	DIFF_TERM
CLK_FWD_I_P	G22	LVDS_25	DIFF_TERM



- Interface type: Custom
- Data Bus Direction: Input
- Data Rate: DDR
- Serialization factor: 8
- External Data Width: 1

- ・ I/O signaling: Differential, LVDS\_25
- ・ Clock strategy: Internal clock
- ・ Data delay type: None
- ・ Clock delay type: None

## 基板上の配線

図 5 のように配線し信号ループバックを形成してください。

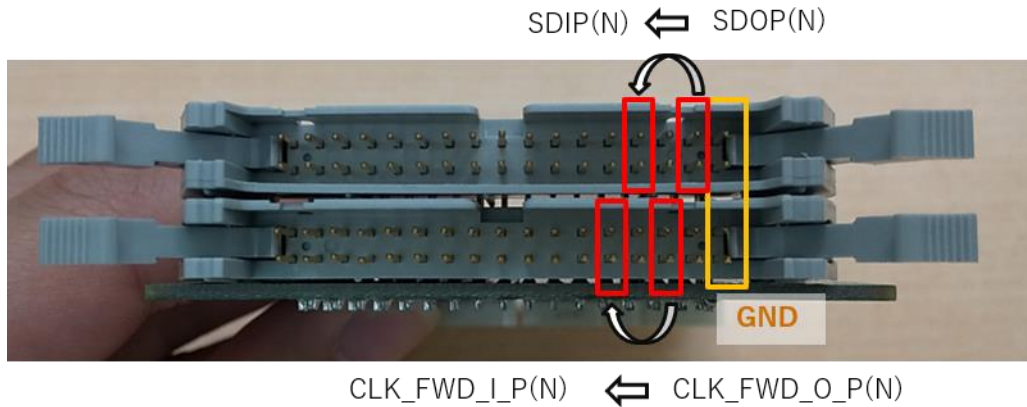


図 5：ループバック経路

## 動作チェック

- ・ FPGA に bit ファイルと Itx ファイルをダウンロード後、ILA が立ち上がることを確認してください。ループバック配線が正しくないと ILA を駆動するリージョナルクロックが喪失するため、ILA が立ち上がらなくなります。
- ・ ISERDES のデータ出力のビットパターンをチェックしましょう。恐らく送信したビットパターンになっていないと思いますが、ビットシフトすれば正しそうなパターンが得られそうでしょうか？
- ・ VIO の probe\_out0 を 0 から 1 へ遷移させるたびに 1 回ずつビットシフトが行われるか確認しましょう。またそのシフトパターンが UG471 に記述されたパターンの通りか確認しましょう。



# 例題 EX3

## メモリリソースへの ECC の実装

### ファームウェア構成

図 6 に例題 E3 のブロック図を示します。この例題では 2 つの ECC 実装方法を試します。SDP モードの block RAM を生成し、入力データ、アドレス、write enable を VIO から与えます。Write enable は VIO probe\_out0 の 3bit 目で与えます。疑似的なデータ破損を injectsbiterr (single bit error)もしくは injectdbiterr (double bit error)で block RAM へ与えます。これらの信号はワンショットパルスである必要があるため、probe\_out0 の 1bit 目および 2bit 目をエッジ検出し、ワンショットパルスを生成してください。また、エラー挿入には write enable が 1 である必要があります。

次に ECC encoder と decoder を実装します。これらは builtin ECC の機能を持たないメモリリソース、例えば分散 RAM や外部の SDRAM チップに対して ECC を付与するために利用します。データ破損を模擬するために VIO からデータ、およびチェックビットへのビット反転パターンを VIO から与え、encoder の出力と XOR をとってください。これにより任意のビット位置ヘシングルもしくは多ビットのエラーを挿入することが出来ます。

### 設計条件

DSP 含め同期回路は clk\_sys のドメインに属するとします。入力クロックの制約を timing.xdc に記述してください。

トップレベルポートのピン制約を以下のように与えてください。

ポート名	ピン番号	IO 規格	その他制約
CLKP	F22	LVDS_25	DIFF_TERM
CLKN	E23	LVDS_25	DIFF_TERM
RSTB_SW	H9	LVCMOS33	

### Block RAM 生成条件

Interface type:	Native
Memory type:	Simple dual port RAM
ECC type:	Builtin ECC
Error injection pins:	Enabled
Port A width:	64
Port A depth:	16

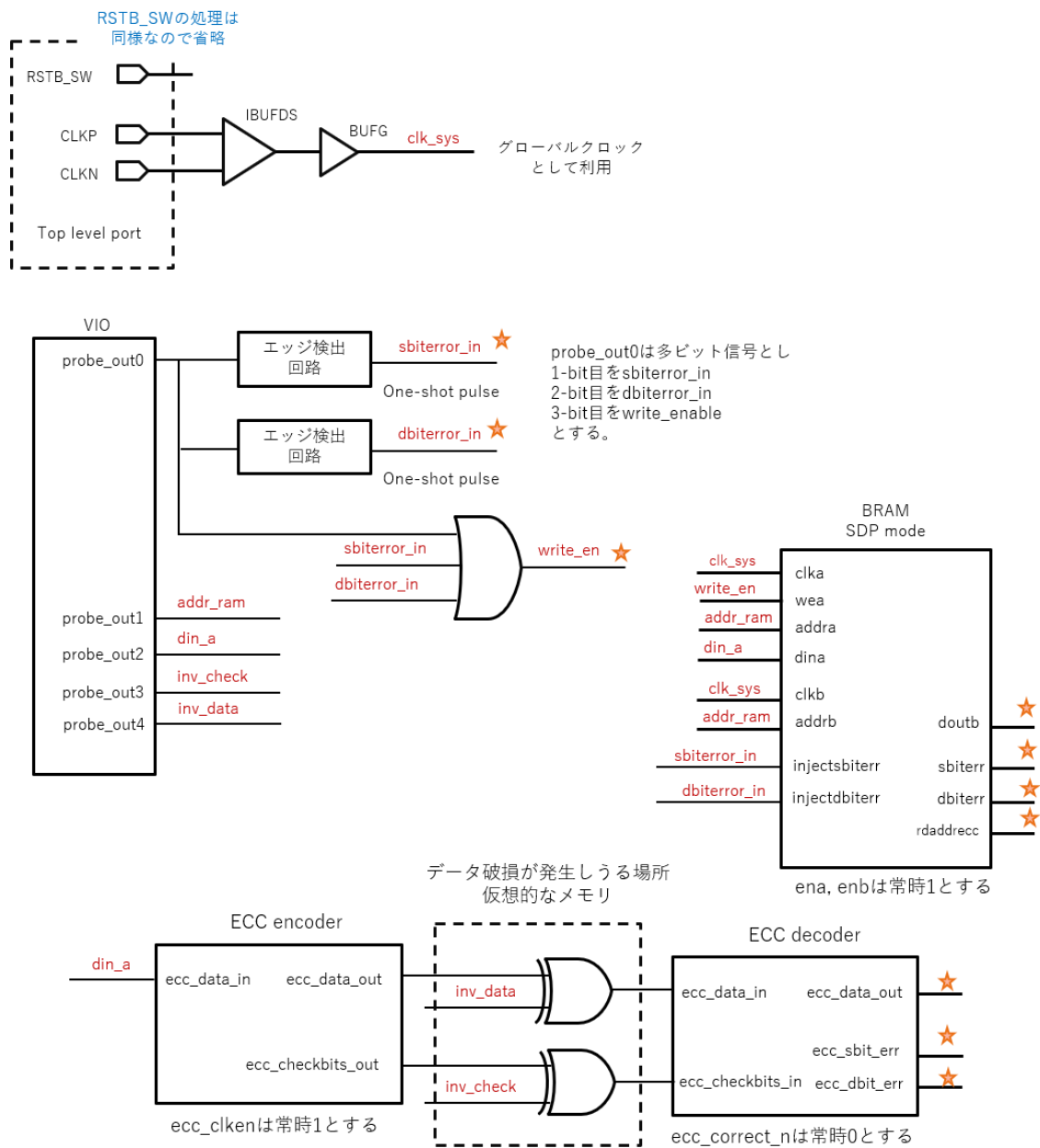


図 6 : E3 のブロック図

## ECC encoder 生成条件

Algorithm Options:	Hamming
ECC mode:	Encoder
Data width:	64
Enable input registering:	enabled
Enable output registering:	enabled

## ECC decoder 生成条件

Algorithm Options:	Hamming
ECC mode:	Decoder
Data width:	64
Enable input registering:	enabled
Enable output registering:	enabled

## 動作チェック

- ・injects(d)biterr をアサートするとアドレスで指定した位置に格納されているメモリコンテンツにデータ破損が発生させます。エラーが発生させた後、doutb、sbiterr、dbiterr、rdaddrecc の値がどのような遷移するか確認しましょう。また、ECC はデータ出力を補正しますがメモリコンテンツはそのままです。一度、他のエラーが発生していないアドレスを指定した後、再びデータ破損が発生しているアドレスを指定して読み出しを行うと、どのようなことが起きるか確認しましょう。
- ・ECC encoder/decoder では inv\_data、inv\_check のパターンを変化させる前に encoder に与えたデータが正しく decoder から出力されることを確認しましょう。その後、inv パターンを変化させ ECC decoder の信号がどのように変化するか確認しましょう。

## 発展課題 HI

### IOSERDES の実装 (IDELAY あり)

#### ファームウェア構成

図 7 に発展課題 HI のブロック図を示します。EX2 との違いは ISERDES が IDELAY 制御用のポートを有することと、IDELAYCTRL 用の基準クロックが必要なことです。MMCM で 200 MHz のクロックを生成し、ISERDES の ref\_clock ポートへ入力してください。この例題では IDELAY 制御に in\_delay\_tap\_in を用います。必要なタップ数をこのポートへ VIO から与えます。in\_delay\_tap\_in は in\_delay\_reset が 1 になった時に反映されます。VIO の probe\_out0 の 2bit 目をエッジ検出し、load\_idelay 信号としてこのポートへ接続してください。

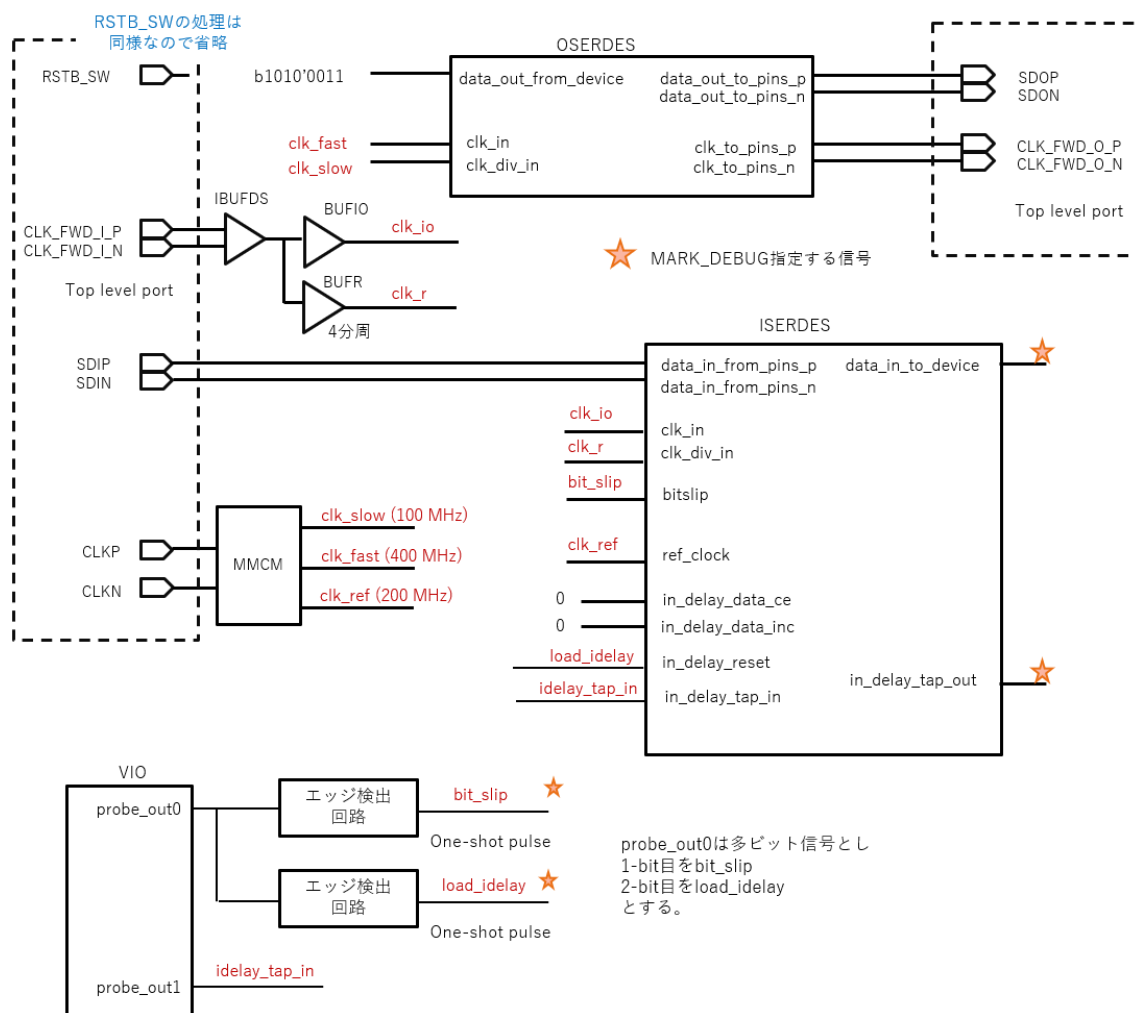


図 7: HI のブロック図

## 設計条件

制約ファイルへの記述は EX2 と同様です。

## OSERDES の生成条件

- Interface type: Custom
  - Data Bus Direction: Output
  - Data Rate: DDR
  - Serialization factor: 8
  - External Data Width: 1
  - I/O signaling: Differential, LVDS\_25
- 
- Clock strategy: Internal clock
  - Clock forwarding: Enabled

## ISERDES の生成条件

- Interface type: Custom
- Data Bus Direction: Input
- Data Rate: DDR
- Serialization factor: 8
- External Data Width: 1
- I/O signaling: Differential, LVDS\_25
- Clock strategy: Internal clock
- Data delay type: Variable loadable
- Clock delay type: None
- Include IDELAYCTRL: Enabled

## 動作チェック

- IDELAY の 1 tap の遅延量は 100 ps 以下です。すぐには data\_in\_to\_device のパターンは変化しません。in\_delay\_tap\_out が load\_idelay アサート後に変化していれば正しく IDELAY は制御できています。そのことをまず確認しましょう。
- IDELAY の最大 tap 数は 32 です。どこかで 1 度 data\_in\_to\_device の出力パターンが安定しなくなる tap 設定が見つかるはずで、それを見つけましょう。

# 発展課題 H2

## Aurora8b10b の実装

### ファームウェア構成

図 8 に発展課題 H2 のブロック図を示します。Aurora8b10b への配線をすべて書きき  
る事が難しいため、このブロック図ではクロック配線のみを記述します。その他の信号線は、  
Aurora8b10 IP の example design や、本演習の教材ソースコードを参照にしつつ配線し  
てください。Aurora8b10b コアは GTREFCLK とそれとは独立のクロックを必要とします。  
100 MHz のクロックを CLKP (N)から生成し GTREFCLK と独立のクロックとして利用し  
てください。Aurora8b10 コアのユーザーインターフェースは user\_clk に同期しています。  
また、GTX channel 用に sync\_clk も必要です。これらは両方とも tx\_out\_clk が起源です。  
user\_clk はグローバルクロックであり、Aurora8b10 の周辺回路もこのクロックで駆動し  
てください。

gt\_common\_wrapper と tx\_clock\_module、およびこの図には記述がないですが  
support\_reset\_logic は Aurora8b10b の example design から移植してください。  
support\_reset\_logic の下位モジュールである cdc\_sync\_exdes も移植が必要です。

### 設計条件

GTREFCLK を timing.xdc 内で定義してください。AMANEQ の GTREFCLK 発振器の周  
波数は 156.25 MHz です。また、false\_path の指定が必要です。Aurora8b10 コアの  
example design の imports ディレクトリ内に参考 xdc ファイルが存在します。それを参  
考にしつつ記述を移植してください。もし分からない場合、教材ソースコード(答え)を参  
照してください。

トップレベルポートのピン制約を以下のように与えてください。

ポート名	ピン番号	IO 規格	その他制約
CLKP	F22	LVDS_25	DIFF_TERM
CLKN	E23	LVDS_25	DIFF_TERM
RSTB_SW	H9	LVCMOS33	
GTREFCLK_P	D6		
GTREFCLK_N	D5		
GTX_TXP	D2		
GTX_TXN	D1		
GTX_RXP	G4		
GTX_RXN	G3		

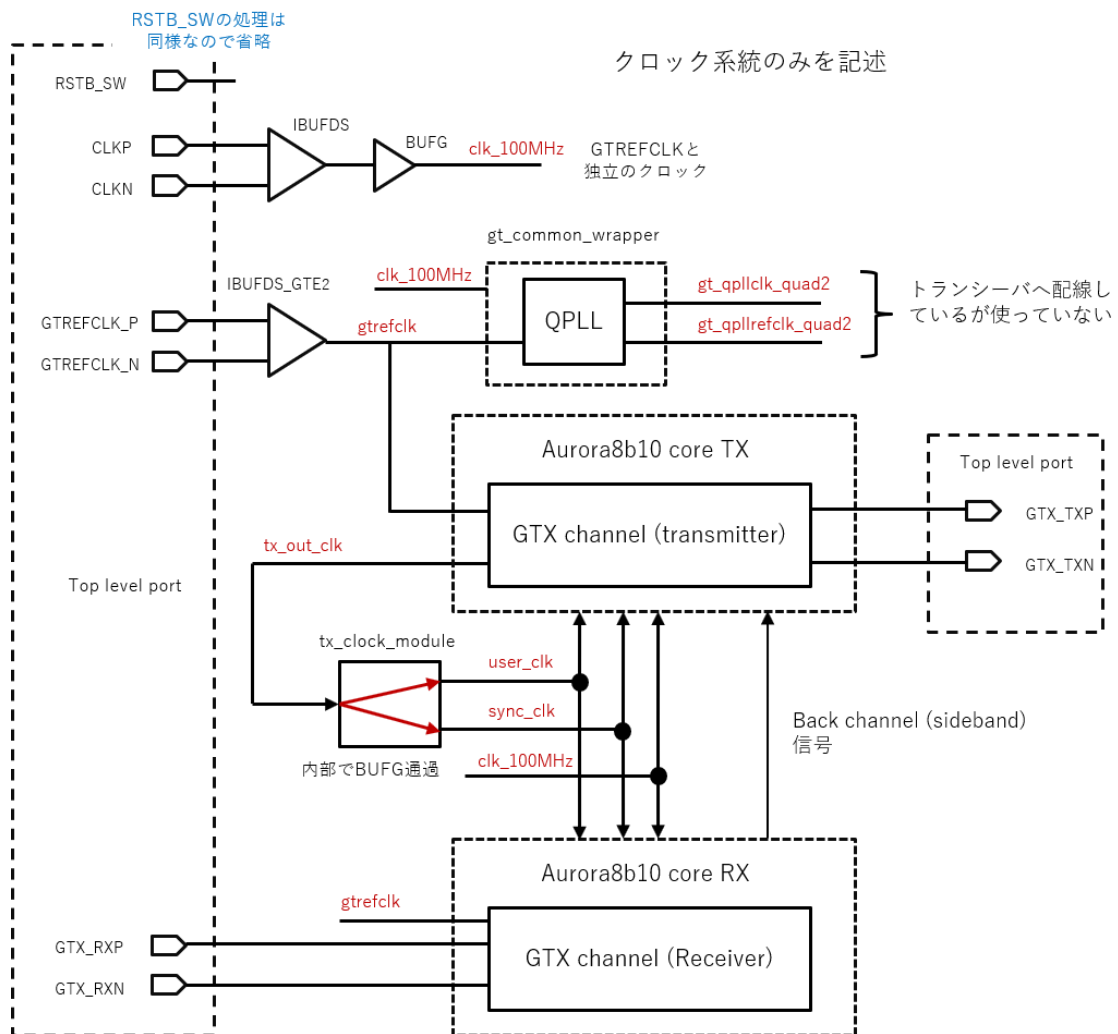


図 8 : H2 のブロック図

## Aurora8b10 コアの生成条件

Lane Width: 2

Line rate: 1.25 Gbps

GT refclk: 156.25 MHz

INIT clk: 100 MHz

DRP clk: 100 MHz

Dataflow mode: Simplex

Interface: Framing

Flow control: None

Back channel: Sidebands

Shared logic については example design に含める方を選択してください。

## AXI4-Stream の制御

s\_axi\_tx\_tdata に 16-bit カウンタの出力を接続してください。TX 側のユーザーインターフェースへフレーム構造をもったデータを入力するためには、tvalid、tlast、tkeep の制御が必要です。tvalid に tx\_channel\_up を接続してください。これによって、コアがデータ通信可能になった時点で tvalid が 1 へ遷移します。今回 16-byte を 1 つのフレームと定義することにします。8 回データをコアに入力するごとに tlast を 1 へ遷移させてください。tlast が 1 へ遷移するごとにコアは Aurora フレームの開始と終了を示す制御文字を挿入します。制御文字を挿入している間は tready が 0 になり、ユーザーデータの取り込みを行いません。データは tvalid と tready 両方が 1 の時にコアに取り込まれるので、tready が 0 の間は 16-bit カウンタのインクリメントを停止してください。tkeep には常に 0 を入力してください。

## Back channel の必要性

RX 側は自身が初期化プロセスに入った場合、そのことを back channel を通じて TX 側へ伝えます。TX 側はそれを元にレーンに流すデータを制御し、コアの初期化プロセスを走らせます。Full-duplex の場合これを通信の両端で行う事で、正しくトランシーバとコアの初期化を行う事が可能になります。Simplex の場合一方通行のためお互いの状態が分かりません。Simplex のコアの初期化は full-duplex に比べて難しいです。

今回の例題ではあえて 2 つの GTX channel を利用し、simplex で実装したコアの間を back channel でつなぎました。Full-duplex ではこのような接続がコアの内部で行われています。

余裕があれば back channel の接続を切って、何が起きるか確認してみてください。レーンは立ち上がるがコアは初期化できなくなります。この事から back channel の重要性和 simplex の難しさが分かると思います。

## 難しいと感じた場合

配布した教材ソースコードと Aurora8b10b example design の imports 内部のソースコードを見比べて、次の点を考察してみましょう。

- imports 内部のどのモジュールが教材ソースコードへ移植されているか。
- 移植するにあたってどの部分が変更されているか。

Aurora8b10b を駆動するだけであれば example design のソースコードが殆どそのまま使える事に気づくはずですが。どのモジュールを移植すればよいか、どういった変更をなぜ加えないといけないのか。これらを理解すれば演習 H2 はこなせるはずです。

## 動作チェック

- ・ TX および RX の lane\_up と channel\_up がそれぞれ 1 になっていることを確認してください。channel\_up が 0 のままの場合、back channel の接続忘れが考えられます。
- ・ TX が送信したデータと RX が受信したデータが一致する事、またフレーム構造に合わせて正しく RX 側の tlast が遷移することを確認してください。TX と RX のデータが一致しない (RX 側でデータが飛ぶ) 場合、tx\_tready の監視とそれに対する回路応答が正し



くありません。

## 発展演習 H3

### GTREFCLK が 156.25 MHz の場合の GbE 通信と IP の改変方法

この演習は普段 SiTCP を利用している方向けです<sup>1</sup>。この演習では自分で HDL コードを書くのではなく、配布されたコードを元に GTREFCLK の周波数と FPGA 内部のクロック周波数の関係を考察し、それに基づいて IP の改変を行います。

この演習では IP の中身をユーザーが書き換える方法を学びます。正しくない IP の変更は重大なトラブルにつながるリスクがある行為である事を十分に理解したうえで取り組んでください。

#### SiTCP と PCS/PMA の関係

図 9 に MGT を利用して GbE 通信を行う際の、PCS/PMA と SiTCP の関係を示します。GbE では 8b10b 変換を利用しているため GbE ではシリアル通信部のラインレートは 1.25 Gbps と決まっています。まず、PMA-TX がこのラインレートでデータを出力できるようにする必要があります。次に、PCS と SiTCP の間では GMII でデータのやり取りが行われます。ここでは 1 Gbps のデータレートを出すために 125 MHz のクロックが必ず必要になります。GbE 通信を行う際にはこの 2 点を抑える必要があります。

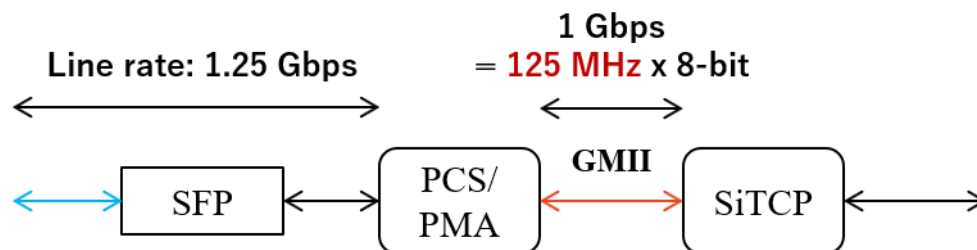


図 9: SiTCP と PCS/PMA コアの関係

#### GTREFCLK の周波数と生成クロック周波数の関係

多くの場合 GbE 用の PCS/PMA は IP カタログで生成するでしょう。GTREFCLK は 125 MHz が入力されることを想定しており、TX 側のクロック配線は図 10 に示すようなルートが指定されます。ここから TX ラインレートと TXOUCLK の両方が GTREFCLK に依存することが想定されますが、TX ラインレートは CPLL の設定を変更することで変える事が出来そうです。では TXOUTCLK はどうでしょうか。

ここからは配布コードの H3 プロジェクトを参照してください。複数のモジュールが toplevel.vhd 内に記述されていますが、この演習で必要な物は GtClockDistributor2、GbEPcsPma、SiTCP です。これらクロック配線図を図 11 に示します。ここからが演習課

<sup>1</sup> SiTCP は BBT の製品です <https://www.bbtech.co.jp/sitcp/>

[illegible]

図 10: GbE 用に PCS/PMA を IP カタログから生成した際の TX クロックルート

配布ソースコードを読み解いて図 11 の配線図を完成させてください。

GTREFCLK が 125 MHz と 156.25 MHz の場合について TXOUTCLK、RXOUTCLK、USERCLK、USERCLK2、GMII TX CLK の周波数を記入してください。

AMANEQ に搭載されている GTREFCLK 用の発振器は 156.25 MHz です。GbE 通信を行うためにはどこの設定をどう変更する必要があるでしょうか。

ヒント：2 つ存在します。

問 3 を正解したうえで問 4 へ進んでください。

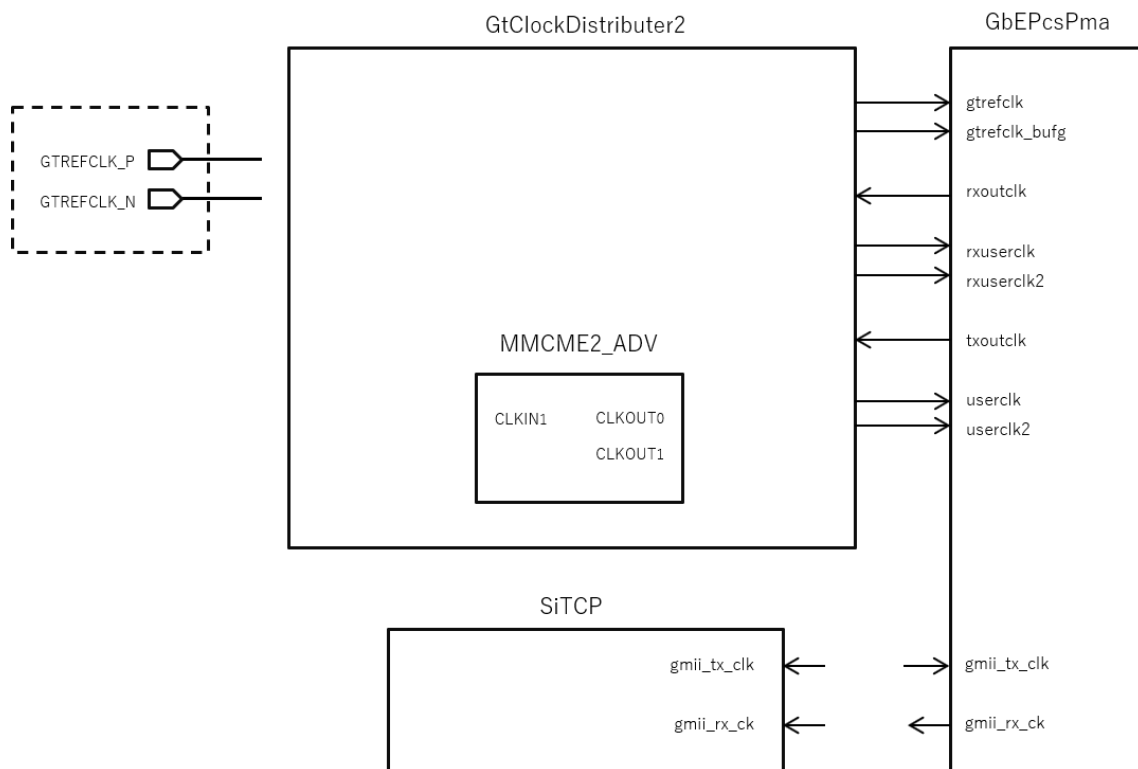


図 11: 配布コードのクロック配線図

#### 問 4

MMCM と CPLL の設定を変更して合成し bitstream ファイルを生成しましょう。CPLL の設定を変更する必要があることがわかりましたが、CPLL 設定は IP 内部で記述されておりプログラマは変更することが出来ません。以下を参考にして IP の内容をユーザーが変更できるようにして、カスタム IP を生成してください。

HDL コードだけでなく IP の XDC も変更しなければなりません。GTREFCLK の周波数に依存している部分を探し出し、全てを変更してください。

#### 問 5

生成した bitstream ファイルを AMANEQ ヘダウンロードして ping が通ることを確認してください。問 5 をやる際には本多に声をかけてください。

### IP の変更方法

CPLL のパラメータは GTXE2\_CHANNEL の generic port で与えます。GTXE2\_CHANNEL は IP が生成する gig\_ethernet\_pcs\_pma\_gtwizard\_gt.vhd 内部で実体化されています。この HDL コードは Vivado の管理下にあるため、内容を変更しても IP

の最終生成物へは反映されません。ユーザーによるカスタム IP を生成するためには Vivado による管理を外す必要があります。そのために次の tcl コマンドを Tcl console から実行します。

```
set_property IS_MANAGED false [get_files <ip_name>.xci]
```

<ip\_name>は今回の場合 gig\_ethernet\_pcs\_pma です。

コマンドが実行されると図 12 のように IP の前にスラッシュが付きます。この状態になった IP は Vivado の管理を離れユーザー管理状態となります。必要な HDL や XDC を変更し、IP の最終生成物へ反映させるために、次の tcl コマンドを実行して IP 合成の run を手動で実行します。













```
>   clk_wiz_sys (19)
>   fmp_rd_fifo (12)
>   fmp_wd_fifo (12)
>   gig_ethernet_pcs_pma (38)
>   sem_controller (10)
>   xadc_sys (11)
```

図 12: ユーザー管理となった IP

```
reset_run <ip_name>_synth_1
```

このコマンドを実行すると、元の IP の最終生成物が破棄されます。

```
launch_run <ip_name>_synth_1
```

このコマンドを実行すると、変更を加えた部分の内容を反映した IP が生成されます。うまくいっているかどうかは dcp が更新されたかどうかで確認することが出来ます。

ここまで終了したらあとは通常通りプロジェクトの合成と配置配線を実行して bitstream を生成してください。

## 発展演習 H4

### ISERDES を用いた 1 ns 精度 TDC の実装

FPGA による time-to-digital converter (TDC) の実装方法は複数の種類が存在しますが、TDC LSB が 100-1000 ps 程度の精度の場合多相クロックによる実装を用いる事が一般的です。例えば 4 相の 250 MHz を用いれば 1 ns 精度の TDC を実装可能です。この手法は良く知られているものの、サンプリングを行う D-FF までの配線長の調整が必要であったり、システムクロックとの分周比に合わせたクロックドメイン乗り換えのブロックが必要であったりと、ある程度実装ノウハウが必要になります。

ISERDES プリミティブをもちいると 1 ns 精度程度 (BUFG が扱える最大周波数に依存) の TDC であれば多相クロックを用いた実装に比べて格段に少ないコード量で実装可能です。この演習では 1 ns 精度でパルスの立ち上がりと立下りを測定する TDC を実装し、入力パルスのパルス幅を測定してみます。

### ISERDES を用いた TDC の概念図

図 13 に概念図を示します。ISERDES は 500 MHz DDR モードで入力パルスをサンプリングし、125 MHz クロックでパラレルデータを出力するとしましょう。この 2 つのクロックは 4 分周の関係にあるため、出力されるデータ幅は 8-bit です。このとき、ISERDES は入力パルスの走行のスナップショットを 1 ns 精度で 8 ns (1/125 MHz) 毎に取得していることになります。ちょうどパルスがやってきたタイミングでは図に示すように 0 と 1 が混ざったデータが出力されるはずであり、このようなデータを温度計コード (thermometer code) と呼びます。これをバイナリエンコードすることにより TDC の下位 3-bit である fine count を得ます。温度計コードは 8 ns 周期で更新されるため、TDC の上位ビットには 125 MHz でカウントアップするフリーランカウンタの値を用いればよいでしょう。たったこれだけで 3-bit fine count + N-bit coarse count のデータ構造を持つ TDC データが得られます。

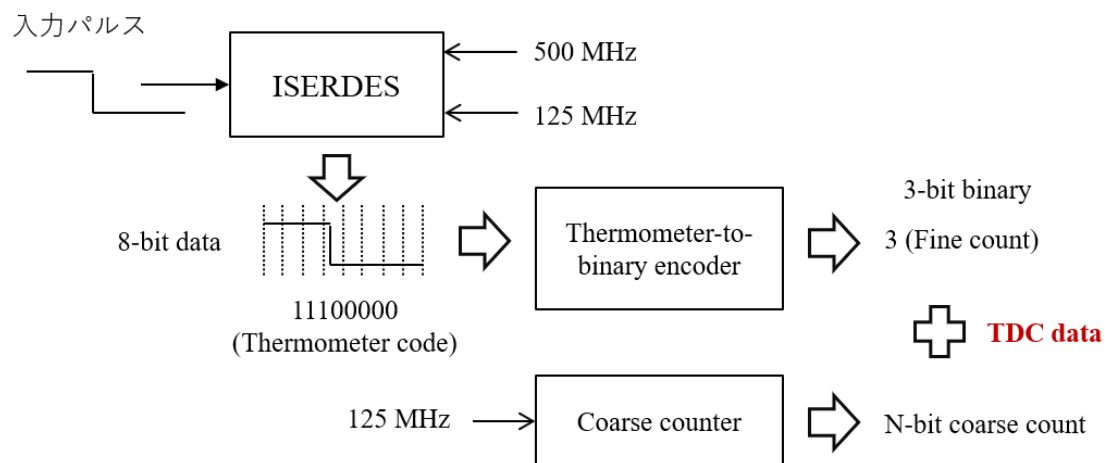


図 13 : ISERDES TDC の概念図

## ファームウェア構成

図 14 にブロック図を示します。H4 は発展演習のため例題のように全ての配線は書き下していません。この演習ではパルスは NIM 入力ポートから入力します。入力パルスはファンクションジェネレータで生成します。この課題では速いクロックと入力データがそもそも同期していないため、遅いクロックとの間の位相を気にする必要がありません。そのため、bitslip は使用しません。

パルス幅を測定するために leading edge と trailing edge 両方を測定する必要があります。Leading edge は 0→1 遷移、trailing edge は 1→0 遷移をそれぞれ発見すればよいのですが、コードの再利用性を高めるためにビット反転を取り、立下りを立ち上がりに見せかけてバイナリエンコーダへ入力しましょう。エンコーダではバイナリ値を出力する他ヒット信号 (data valid) も同時に出力するのが良いでしょう。ヒット信号を使ってレジスタにコースカウンタの値と一緒に記録し、減算を取ってパルス幅を得ましょう。

動作確認のために適当な場所の信号を ILA で監視するようにしてください。この課題では VIO は利用しません。

## 設計条件

トップレベルポートのピン制約を以下のように与えてください。

ポート名	ピン番号	IO 規格	その他制約
CLKP	F22	LVDS_25	DIFF_TERM
CLKN	E23	LVDS_25	DIFF_TERM
RSTB_SW	H9	LVC MOS33	
NIM_IN	V8	LVC MOS15	

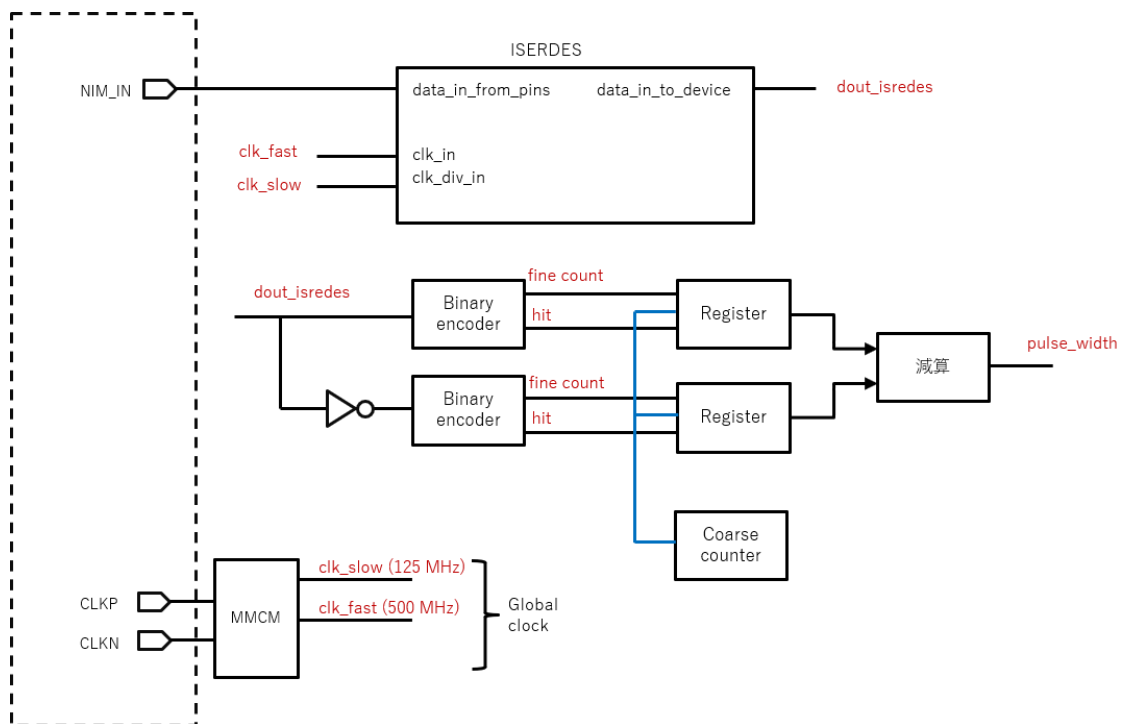


図 14 : H4 ブロック図

### 問 1

Thermometer-to-binary encoder を設計してください。

検討事項 1 : パルスの立ち上がり (0→1) は ISERDES の Q0 から現れるでしょうか? Q7 から現れるでしょうか?

検討事項 2 : ヒット信号はワンショットパルスである必要があります。どのようにヒット判定をするのがよいでしょうか? ヒント: パルス入力の前は必ず ISERDES の出力がすべて 0 です。

### 問 2

Coarse counter を実装しましょう。

検討事項 1 : 入力信号幅は 1 us 以内とします。ビット幅はいくつにすればよいでしょうか。

検討事項 2 : TDC データは fine count と coarse count の単純なベクトル連結で作るとします。この定義に沿って TDC データを生成する場合、coarse count はカウントアップとカウンドダウンのどちらで生成すべきでしょうか? また、それはどうしてでしょうか?

### 問 3

作製した TDC を AMANEQ へダウンロードし、実際にパルスを入力してパルス幅が測れることを確認しましょう。ファンクションジェネレータを使うので講師へ声をかけてください。