



XAPP1227 (v1.0) 2014 年 11 月 12 日

Aurora 64B/66B IP コアを使用した非対称レーンの設計

著者 : Venkata Srinadh Utukuru, Sai Krishna Marella

概要

Aurora 64B/66B は、高速シリアル通信向けのスケーラブル、軽量、そして高データレートのリンク レイヤー プロトコルです。Aurora は、直感的なウィザード インターフェイスを使用して、ザイリンクスのトランシーバーを簡単に実装することを目的としています。Aurora プロトコルの仕様は公開されており、リクエストに応じて提供されます。Aurora コアは Vivado® IP カタログから無償で利用可能で、ライセンスを取得してザイリンクスのシリコン デバイスで使用できます。

一般に、Aurora IP コアは特定のプロトコルに依存しない小面積、広帯域、低オーバーヘッドのポイント ツー ポイント通信チャンネルが必要とされるアプリケーションで使用します。そのシンプルなフレーム構造は、既存プロトコルからのデータを容易にカプセル化でき、また電氣的要件も汎用システムと互換性があります。Aurora を使用すると、シンプルな AXI4-Stream ベースのユーザー インターフェイスで高いパフォーマンスを達成できます。ソフトウェアの再開発や、特殊な物理インフラの構築は必要ありません。

このアプリケーション ノートでは、Virtex-7 FPGA VC7203 特性評価プラットフォームでザイリンクス LogiCORE™ IP Aurora 64B/66B コアの非対称レーン デザインを作成するための手順を説明します。

はじめに

Aurora 64B/66B はシンプレックスとデュプレックスの 2 種類のデータ フロー モードをサポートしています。ビデオ プロジェクターなどのアプリケーションでは、メインとなるストリーム データを高スループットのチャンネルで転送し、反対方向の低スループット チャンネルで補助情報や制御情報を転送します。つまり、このようなアプリケーションでは、データ フロー用には、反対方向のチャンネルよりも多くのレーン (高いスループット) が必要となります。

現在 Aurora 64B/66B コアで利用できるデータ フロー モード (シンプレックス/デュプレックス) では、TX と RX で同数のレーンしか設定できません。TX と RX で異なるレーン数としたい場合は、各方向に 2 つの異なる Aurora シンプレックス コアを生成する必要があります。ただし、その場合は各コアが別々の GT トランシーバー (シリアル トランシーバー) を使用するため、GT リソースの使用量が増加します。

このアプリケーション ノートでは、各方向のレーン数が異なる Aurora マルチレーン デザイン (非対称 Aurora デザイン) を作成する際に、共有ロジックを使用して GT リソースの使用量を最小限に抑える方法について説明します。

このアプリケーション ノートで説明する手順を用いて作成したサンプル デザインは、その他のコンフィギュレーションにも適用できます。デザイン作成手順はコアの論理設計を一切変更しないため、コアの機能は Vivado® Design Suite で生成した通常のシンプレックス デザインとまったく同じです。

インプリメンテーションの詳細

非対称データ リンクには 2 つのデザインを作成する必要があります。

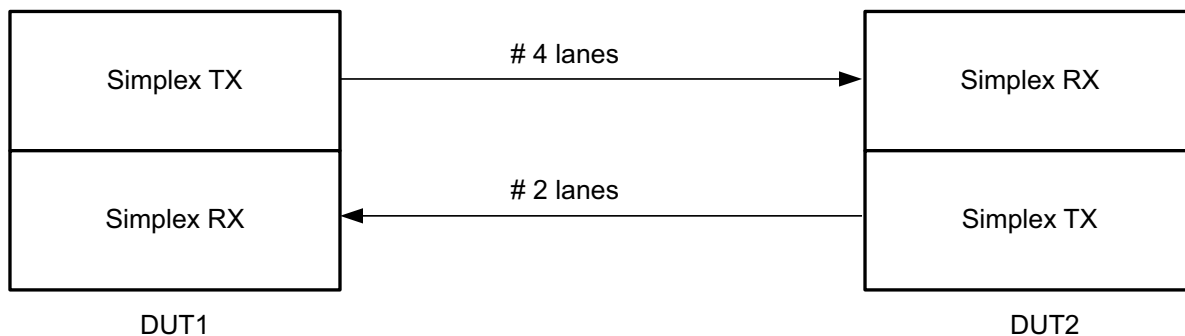
- デザイン 1 (M レーン シンプレックス TX + N レーン シンプレックス RX)
- デザイン 2 (M レーン シンプレックス RX + N レーン シンプレックス TX)

M と N はシンプレックス TX/シンプレックス RX コアのレーン数を表します。更新後のデザインは、インプリメンテーション時に最大 (M, N) 個のトランシーバーを使用します。このセクションでは、M=4、N=2 とします。

- デザイン 1 (4 レーン シンプレックス TX + 2 レーン シンプレックス RX)
- デザイン 2 (4 レーン シンプレックス RX + 2 レーン シンプレックス TX)

この後のセクションでは、非対称レーンのシンプレックス デザインを作成する手順を詳しく説明します。

注記：これらの手順は VC7203 ボード上に 1 つのサンプル コンフィギュレーションを作成する方法を示したもので、レーン数によって内容が異なる場合はそのガイドラインも併せて示します。



X14356

図 1: 非対称レーンのセットアップ

Aurora 64B/66B コアには「共有ロジック」と呼ばれる機能があり、必要なクロックおよびステータス信号を GT のコンフィギュレーションが同じ 2 つのデザインで共有可能な形で生成できます。このアプリケーション ノートで紹介するデザインのインプリメンテーションでは、コア生成時にこの共有ロジック機能を使用します。レーン数の多い方のコアはオプションで [include Shared Logic in core] をオンにして生成し、こマスター コアとします。もう一方のコアは [include Shared Logic in example design] をオンにして生成し、スレーブ コアとします。必要な共有ロジック信号は、マスター コアからスレーブ コアへ供給します。

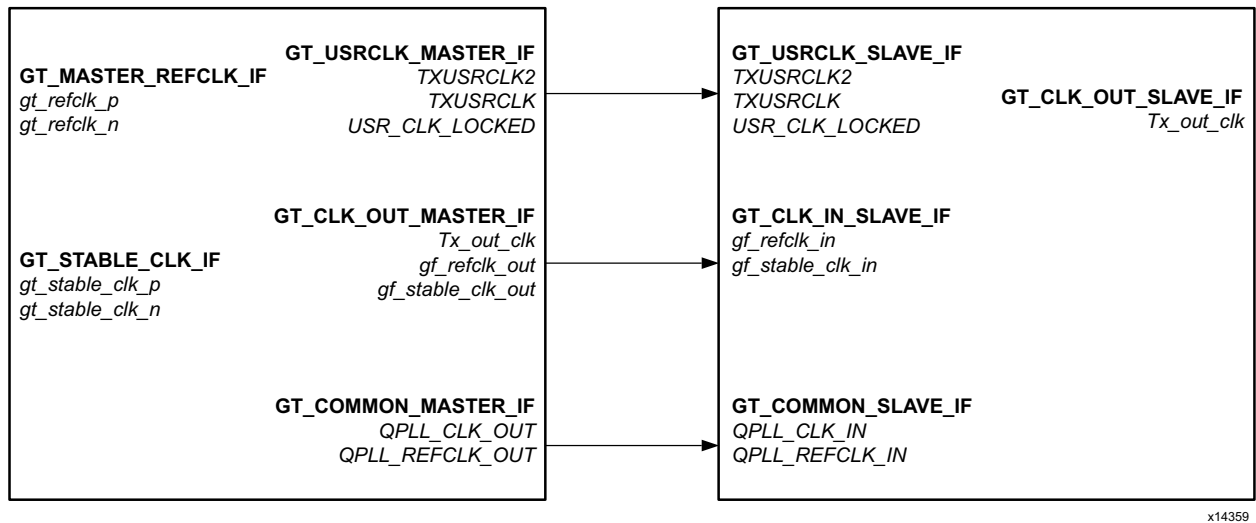


図 2: 共有ロジック

ハードウェアの作成 – デザイン 1 (4 レーン シンプレックス RX + 2 レーン シンプレックス TX)

デザイン 1 の作成に必要な手順は、次のとおりです。

1. 4 レーン シンプレックス RX コアを生成する
2. 2 レーン シンプレックス TX コアを生成する
3. シンプレックス TX Aurora コアの GTX トランシーバー接続を削除する
4. シンプレックス TX の信号をシンプレックス RX コアに接続する

4 レーン シンプレックス RX コアを生成する

次の手順に従って Aurora 64B/66B コアをカスタマイズし、4 レーン シンプレックス RX コアを生成します。

1. Vivado Design Suite を起動します。
2. [Create New Project] をクリックして [Next] をクリックします。
3. プロジェクト名とパスを選択して [Next] をクリックします。
4. [RTL Project] をオンにしてサンプルデザインの実行を許可し、[Do not specify sources at this time] をオンにします。[Next] をクリックします。
5. [XC7VX485TFFG1761-3] または GTX トランシーバーを内蔵したターゲット デバイスをクリックします。
6. [Next] をクリックして [Finish] をクリックします。
7. Flow Navigator の [Project Manager] 下にある [IP catalog] をクリックします。[Group by taxonomy] をクリックして「Aurora 64B66B」を検索します。

Aurora コアは、[Communication & Networking] → [Serial Interfaces] の下にあります。図 3 を参照してください。

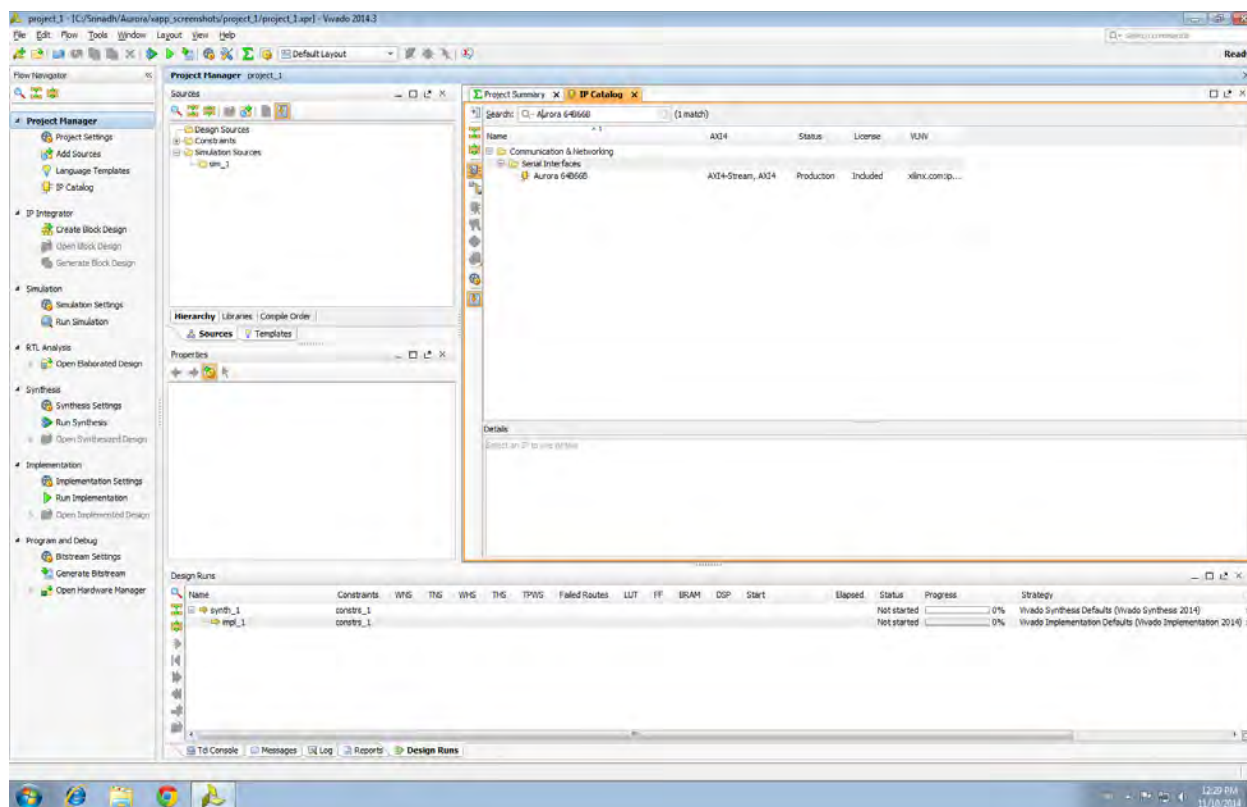


図 3 : Vivado Design Suite の [IP Catalog] タブで Aurora 64B/66B IP を表示

8. [Aurora 64B/66B] を右クリックして [Customize IP] をクリックします。
9. [Customize IP] ダイアログ ボックスの [Core Options] タブで、[Vivado Lab Tools] をオンにします。
10. 図 4 のようにコンフィギュレーション オプションを設定します。

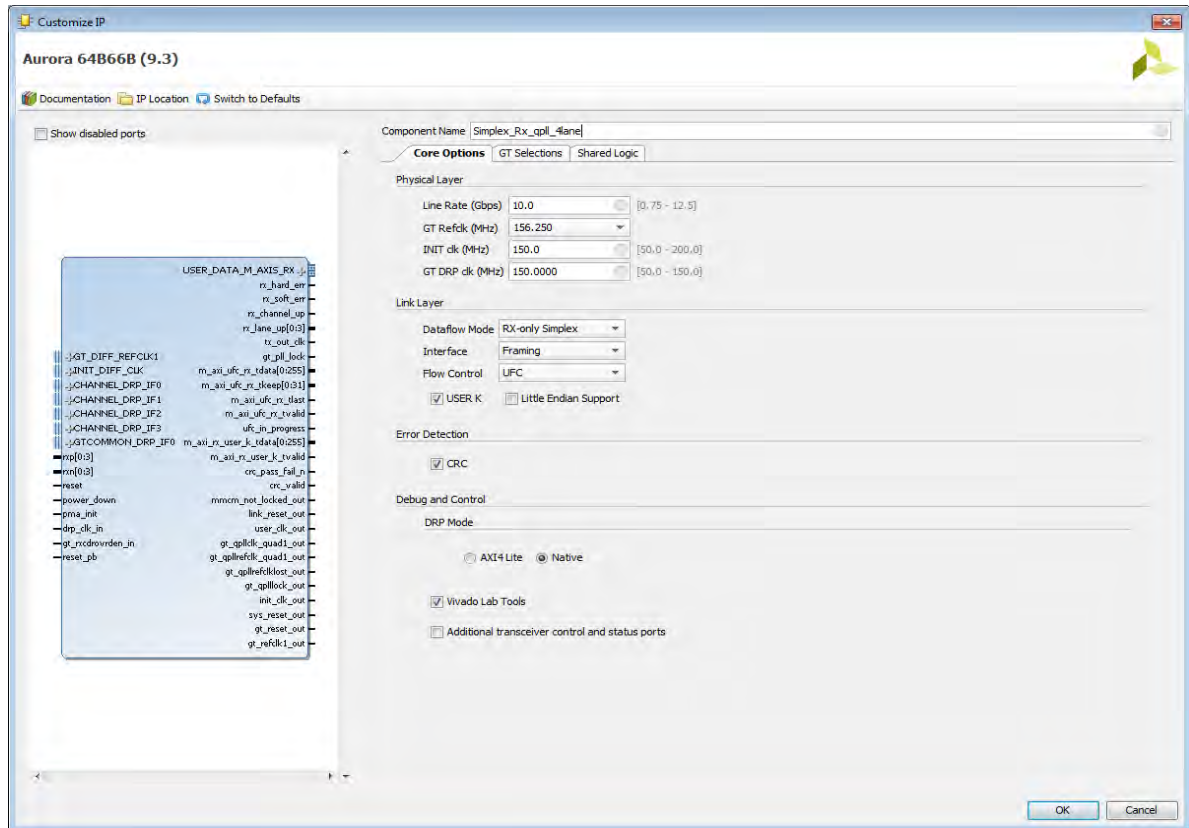


図 4 : Aurora 64B/66B コアを 4 レーン シンプレックス RX コアとして生成する場合の Vivado IDE の設定

11. [GT Selections] タブをクリックします。ターゲット ボードに従って GT トランシーバーを選択します。

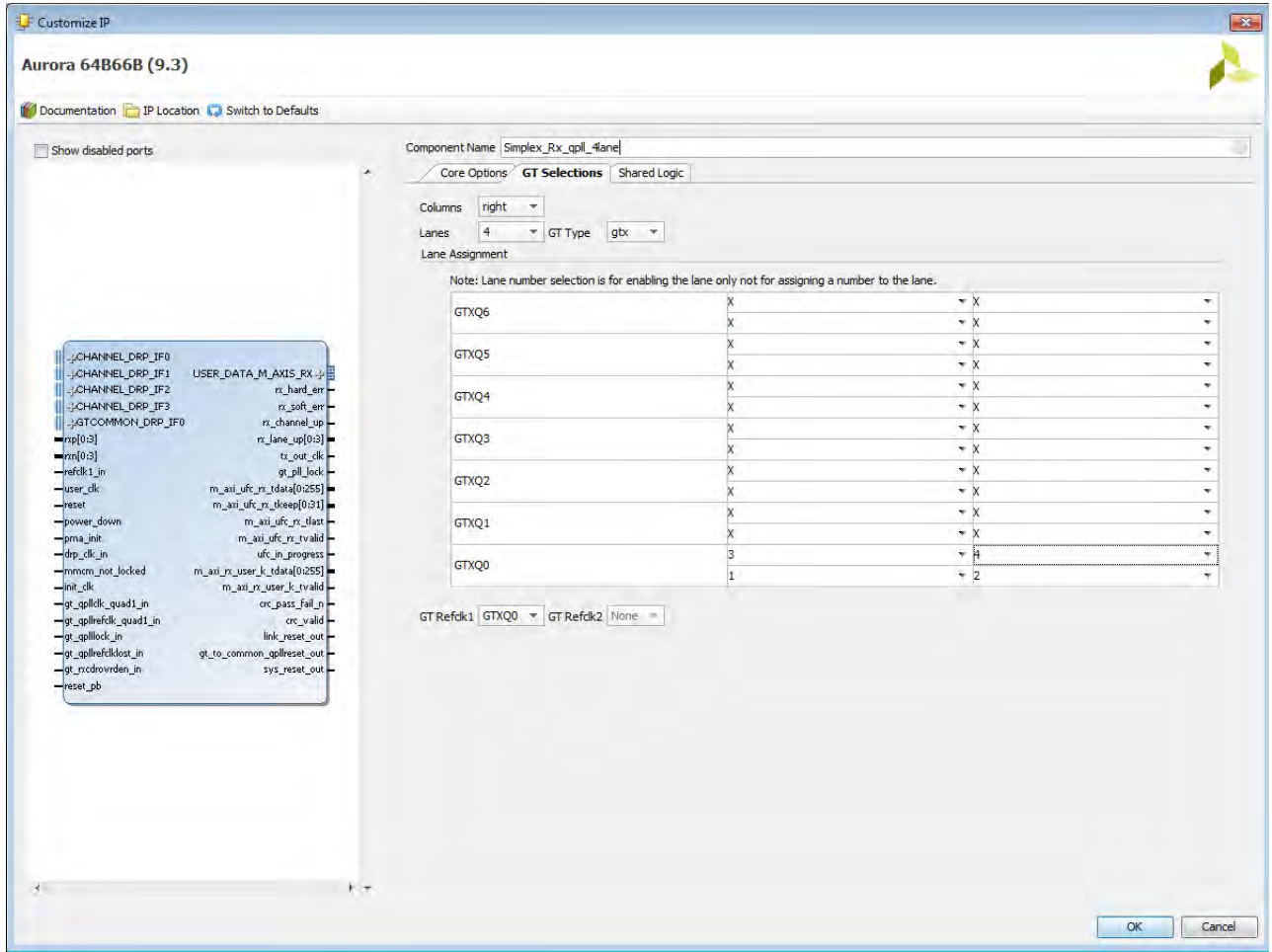


図 5 : [GT Selections] タブ

12. [Shared Logic] タブをクリックします。[include Shared Logic in core] をオンにして [OK] をクリックします。図 6 を参照してください。

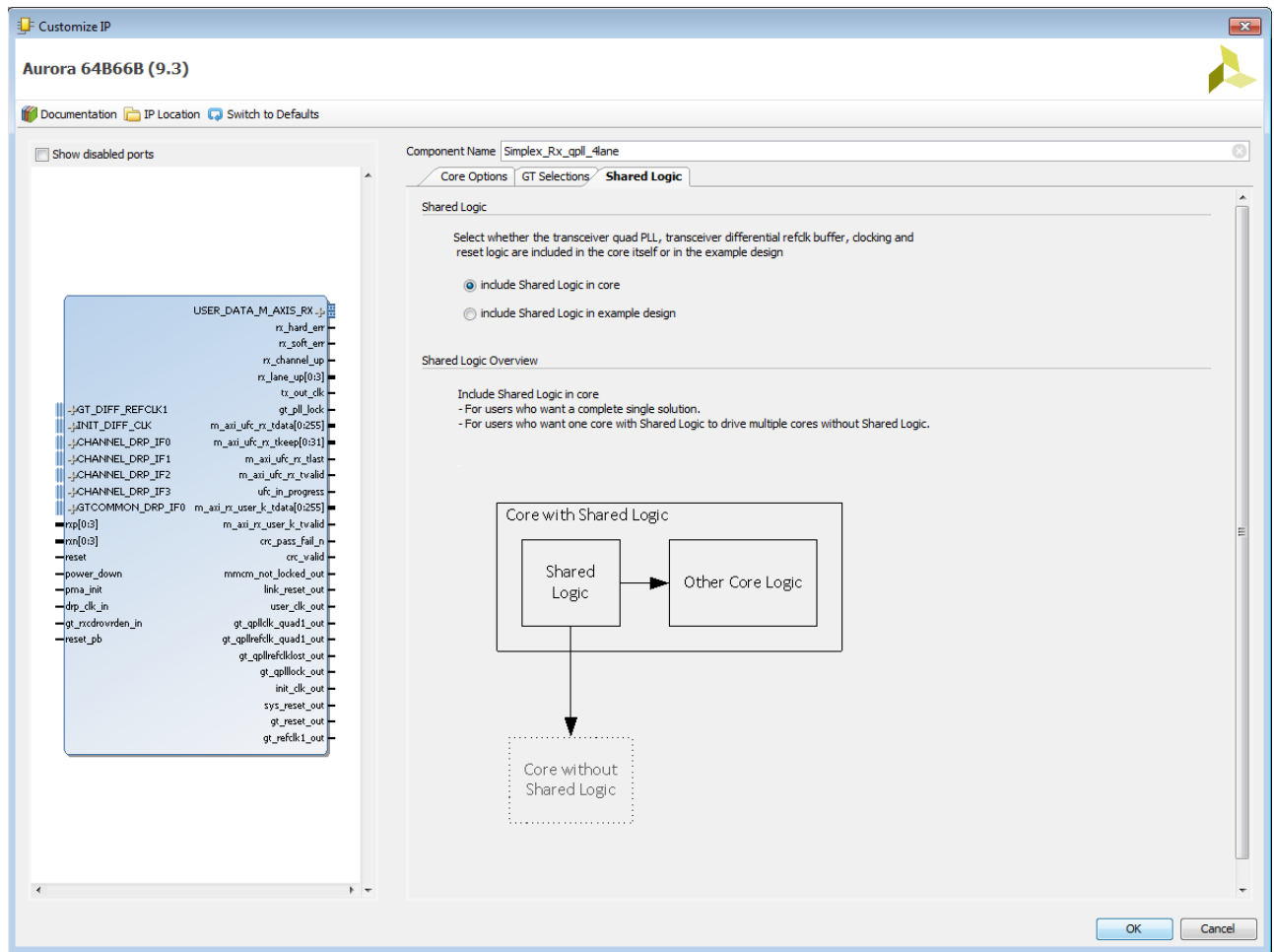


図 6 : 4 レーン シンプレックス RX コアに必要な共有ロジック設定

13. [Generate Output Products] ダイアログ ボックスで [Out-of-Context Settings] をクリックし、[Out-of-Context Settings] ダイアログ ボックスのチェック ボックスをオフにします。続いて [OK] をクリックします。次に、[Generate Synthesized Checkpoint] ダイアログ ボックスが表示されます。

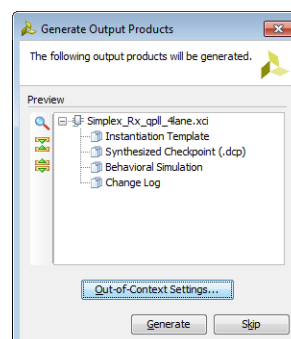


図 7 : [Generate Output Products] ダイアログ ボックス

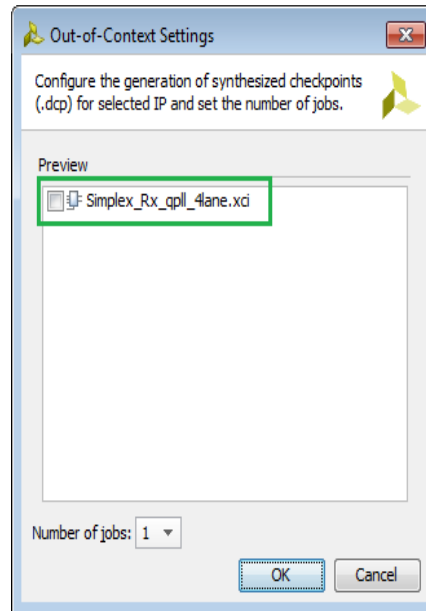


図 8 : [Out-of-Context Settings] ダイアログ ボックス

14. [Generate Synthesized Checkpoint] ダイアログ ボックスで [OK] をクリックします。

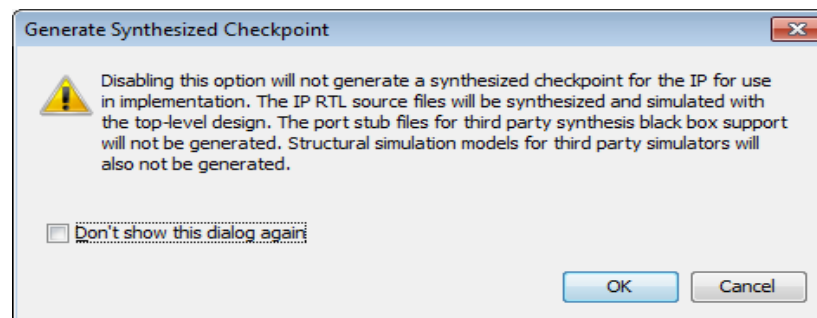


図 9 : [Generate Synthesized Checkpoint] ダイアログ ボックス

15. [.xci] を右クリックして [Open IP Example Design] をクリックします。[Open IP Example Design] ダイアログ ボックスでターゲット ディレクトリを選択し、[OK] をクリックします。

2 レーン シンプレックス TX コアを生成する

次の手順に従って Aurora 64B/66B コアをカスタマイズし、2 レーン シンプレックス TX コアを生成します。

1. Flow Navigator の [Project Manager] 下にある [IP catalog] をクリックします。[Group by taxonomy] をクリックして「Aurora 64B66B」を検索します。

Aurora コアは、[Communication & Networking] → [Serial Interfaces] の下にあります。図 3 を参照してください。

2. [Aurora 64B66B] を右クリックして [Customize IP] をクリックします。
3. [Customize IP] ダイアログ ボックスの [Core Options] タブで、[Vivado Lab Tools] をオンにします。
4. 図 10 のようにコンフィギュレーション オプションを設定します。各オプションは、[Dataflow Mode] を除き 4 レーン RX デザインと同じ設定にしてください。

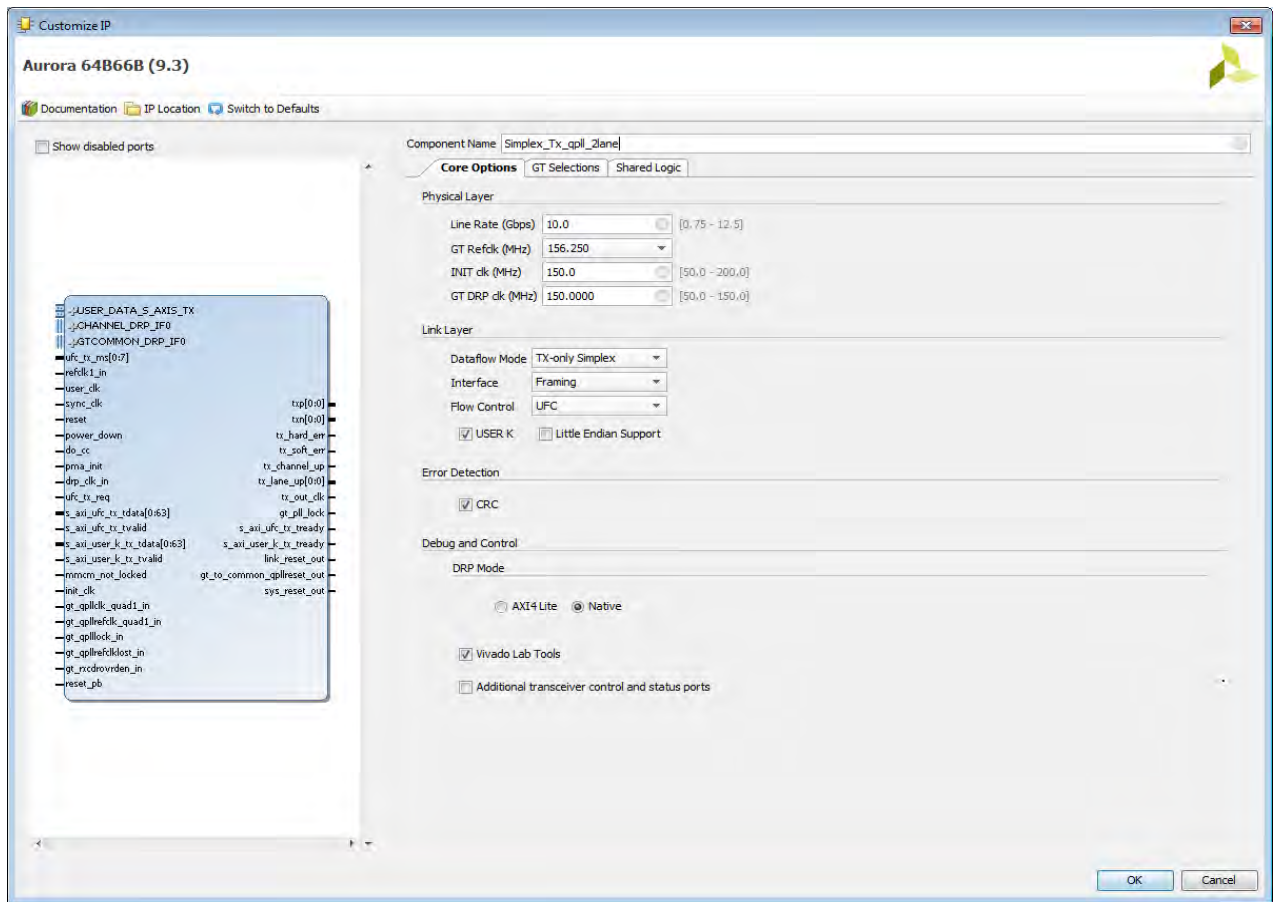


図 10 : Aurora 64B/66B コアを 2 レーン シンプレックス TX コアとして生成する場合の Vivado IDE の設定

5. [GT Selections] タブをクリックします。ターゲット ボードに従って GT トランシーバーを選択します。
6. [Shared Logic] タブをクリックします。[include Shared Logic in example design] をオンにして [OK] をクリックします。
7. [Generate Output Products] ダイアログ ボックスで [Out-of-Context Settings] をクリックし、チェック ボックスをオフにします。続いて [OK] をクリックします。次に、[Generate Synthesized Checkpoint] ダイアログ ボックスが表示されます。
8. [.xci] を右クリックして [Open IP Example Design] をクリックします。[Open IP Example Design] ダイアログ ボックスでターゲット ディレクトリを選択し、[OK] をクリックします。

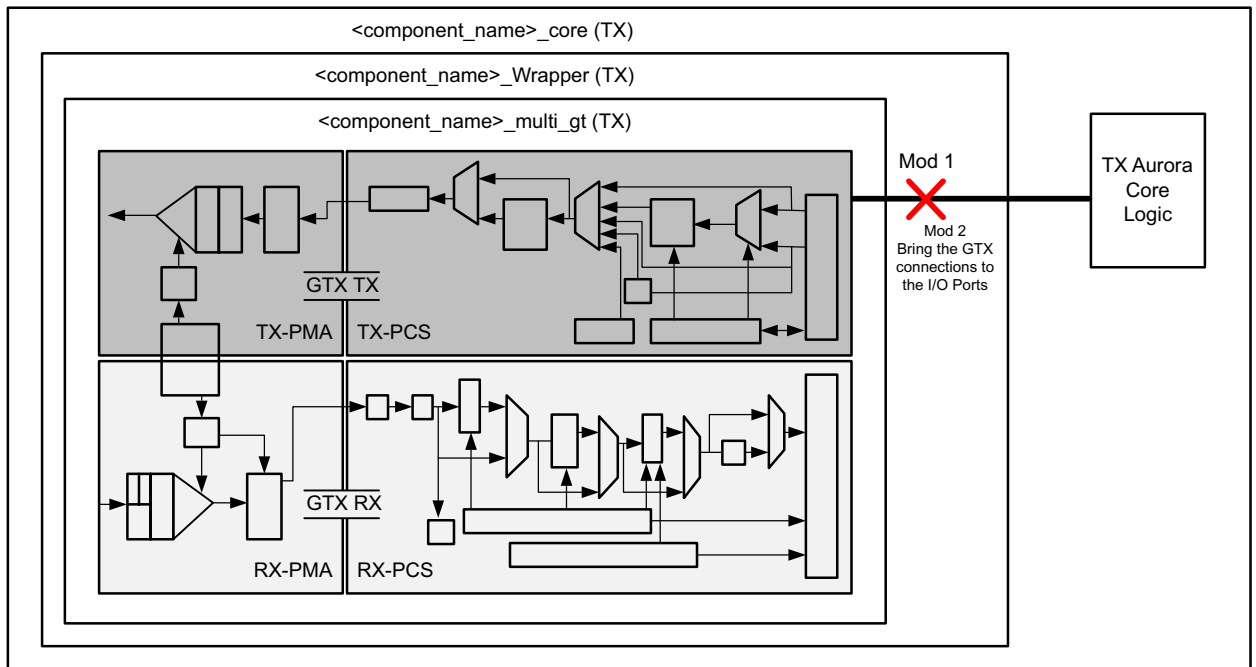
シンプレックス TX Aurora コアの GTX トランシーバー接続を削除する

シンプレックス TX Aurora デザインから GTX トランシーバーのインスタンスを削除し、シンプレックス RX Aurora デザインにインスタンス化された GTX トランシーバー (TX 側) に必要な信号を接続するには、`<tx_component_name1>TX_only_wrapper.v` ファイルに次の変更を加える必要があります。

変更内容

シンプレックス 4 レーン RX デザインとシンプレックス 2 レーン TX デザインは GT リソースを共有しているため、`<tx_component_name1>TX_only_wrapper.v` ファイルで次のコンポーネントをコメントアウトしてシンプレックス 2 レーン TX デザインから GT のインスタンスを削除します。

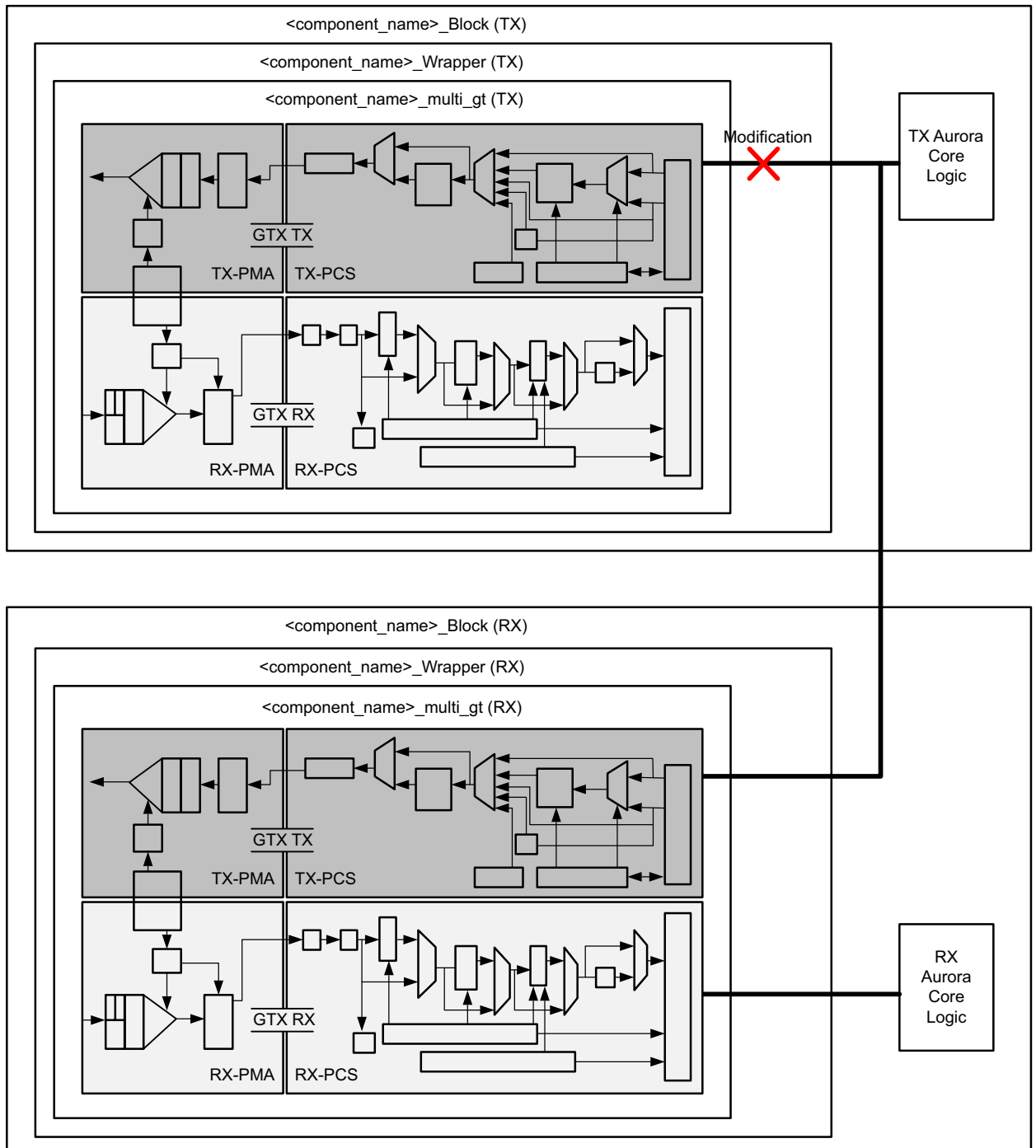
```
<tx_component_name1>_MULTI_GT
```



X14360

図 11: シンプレックス TX デザインの GTX インスタンスを削除

シンプレックス TX の信号をシンプレックス RX コアに接続する



x14358

図 12: シンプレックス TX の信号をシンプレックス RX コアに接続

GT インスタンスをコメントアウトしたら、シンプレックス 2 レーン TX デザインの GTX トランシーバー インスタンスに接続されている GTX トランシーバーの入力/出力をシンプレックス 2 レーン TX デザインの最上位のサンプル デザインへ接続し、これらをシンプレックス 4 レーン RX デザインの GTX トランシーバー インスタンスに接続する必要があります。この手順では、2 レーン シンプレックス TX から GTX トランシーバー入力にシンプレックス RX コアへ接続します。

シンプレックス TX コアに必要な GTX トランシーバー出力を、シンプレックス RX Aurora コアの最上位に接続した後、シンプレックス TX コアに接続します。図 12 に、この手順で実行する変更を示します。

この手順では、シンプレックス TX コアの次のファイルを編集します。

- <tx_component_name1>_wrapper.v
- <tx_component_name1>_core.v
- <tx_component_name1>.v
- <tx_component_name1>_support.v
- <tx_component_name1>_exdes.v

この手順では、シンプレックス RX コアの次のファイルを編集します。

- <rx_component_name1>_gtx.v
- <rx_component_name1>_wrapper.v
- <rx_component_name1>_core.v
- <rx_component_name1>.v
- <rx_component_name1>_exdes.v

次に、これらの各ファイルで必要な変更内容について説明します。

シンプレックス TX コアの <tx_component_name1>_wrapper.v の編集

<tx_component_name1>_wrapper ファイルに必要な変更箇所は 2 つあります。

変更内容 1

GTX トランシーバーの入力および出力すべてに対するバスを作成します。これらのバス信号はデザインの最上位に伝搬します。

<tx_component_name1>_wrapper モジュールに次のバス信号を追加します。これらの信号は GTX トランシーバー専用で、シンプレックス RX コアにインスタンス化された GTX トランシーバーの TX 側のポートに接続するために使用します。

```
output [140:0] GT_IN_SMPX_RX_IN_TX_OUT;  
input [1:0] GT_OUT_SMPX_RX_OUT_TX_IN;
```

注記: 上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

新規に作成したバスに GTX トランシーバーのインターフェイス信号を assign 文で接続します。必要な変更内容を、次のコードに示します。

```
assign GT_IN_SMPX_RX_IN_TX_OUT=
  {gttxreset_t,txuserddy_t,tx_hdr_r,scrambled_data_i,txsequence_i,tx_hdr_lane1_r,
  scrambled_data_lane1_i};

assign tx_resetdone_i = GT_OUT_SMPX_RX_OUT_TX_IN[0];
assign tx_resetdone_lane1_i = GT_OUT_SMPX_RX_OUT_TX_IN[1];
```

次の信号は共通で、スレーブ コアから独立しています。

- gttxreset_t
- txuserddy_t
- txsequence_i

次の信号は、スレーブ コアのレーンごとにマスター コアに接続する必要があります。

- scrambled_data_i - GT へのスクランブル済みデータ入力
- tx_hdr_r - GT へのヘッダー入力
- tx_resetdone_i - GT からの RESETDONE 出力

シンプレックス TX コアの <tx_component_name1>_core.v の編集

<tx_component_name1>_core.v ファイルに必要な変更箇所は 2 つあります。

変更内容 1

前述のとおり、バス信号はデザインの最上位に伝搬します。このため、これらの信号を _core.v ファイルでも定義する必要があります。

<tx_component_name1>_core モジュールに次のバス信号を追加します。

```
output [140:0] GT_IN_SMPX_RX_IN_TX_OUT;
input [1:0] GT_OUT_SMPX_RX_OUT_TX_IN;
```

注記: 上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

手順 4.1 でラッパー モジュールに新規ポートを追加しているため、core.v ファイルのラッパー インスタンスレーションにも同じ変更が必要です。これらのバス信号を <tx_component_name1>_wrapper モジュールのインスタンスレーションの同じバス信号に接続します。必要な変更内容を、次のコードに示します。

```
<tx_component_name1>_WRAPPER  #
(
.....
)
<tx_component_name1>_wrapper_i
(
.....
.GT_IN_SMPX_RX_IN_TX_OUT      (GT_IN_SMPX_RX_IN_TX_OUT),
.GT_OUT_SMPX_RX_OUT_TX_IN     (GT_OUT_SMPX_RX_OUT_TX_IN),
.....
);
```

シンプレックス TX コアの <tx_component_name1>.v の編集

<tx_component_name1>.v の変更内容は、手順 4.2 と同様です。<tx_component_name1>.v ファイルに必要な変更箇所は 2 つあります。

変更内容 1

<tx_component_name1>.v モジュールに次のバス信号を追加します。

```
output [140:0] GT_IN_SMPX_RX_IN_TX_OUT;
input [1:0]    GT_OUT_SMPX_RX_OUT_TX_IN;
```

注記：上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

変更内容 1 で定義したポートを、<tx_component_name1>_core.v モジュールのインスタンスレーションの各バス信号に接続します。必要な変更内容を、次のコードに示します。

```
<tx_component_name1>_core  inst
(
.....
.GT_IN_SMPX_RX_IN_TX_OUT      (GT_IN_SMPX_RX_IN_TX_OUT),
.GT_OUT_SMPX_RX_OUT_TX_IN     (GT_OUT_SMPX_RX_OUT_TX_IN),
.....
);
```

シンプレックス TX コアの <tx_component_name1>_support.v の編集

GTX トランシーバーの入力/出力バス信号に加え、マスター コア (この場合はシンプレックス RX コア) からの共有ロジック関連の信号も接続する必要があります。すべての共有ロジック信号をシンプレックス TX コア (スレーブ) に接続するためには、新しいバスを定義する必要があります。共有ロジック信号は最上位モジュールに作成され、これらすべての信号がシンプレックス TX コアの <tx_component_name1>_support.v モジュールに接続されます。

<tx_component_name1>_support.v ファイルに必要な変更箇所は 4 つあります。

変更内容 1

次のバス ポートを追加します。

```
output [140:0] GT_IN_SMPX_RX_IN_TX_OUT;
input [1:0]    GT_OUT_SMPX_RX_OUT_TX_IN;
input  [7:0]   SHARED_LOGIC_MASTER_TO_SLAVE
```

注記: 上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

共有ロジック信号を各信号に assign 文で接続します。

```
assign user_clk_i           = SHARED_LOGIC_MASTER_TO_SLAVE[0];
assign sync_clk_i          = SHARED_LOGIC_MASTER_TO_SLAVE[1];
assign INIT_CLK_i         = SHARED_LOGIC_MASTER_TO_SLAVE[2];
assign refclk1_in_i       = SHARED_LOGIC_MASTER_TO_SLAVE[3];
assign gt_qpllclk_quad1_i = SHARED_LOGIC_MASTER_TO_SLAVE[4];
assign gt_qpllrefclk_quad1_i = SHARED_LOGIC_MASTER_TO_SLAVE[5];
assign gt_qplllock_i      = SHARED_LOGIC_MASTER_TO_SLAVE[6];
assign pll_not_locked_i   = SHARED_LOGIC_MASTER_TO_SLAVE[7];
assign mmc_m_not_locked_i = pll_not_locked_i;
assign refclk1_in         = refclk1_in_i;
```

変更内容 3

次のモジュール インスタンスエーションをコメントアウトして、シンプレックス 4 レーン RX デザインと重複しているロジックを削除します。

- <tx_component_name1>_gt_common_wrapper
- IBUFDS_GTE2
- <tx_component_name1>_CLOCK_MODULE

変更内容 4

変更内容 1 で定義した、GTX トランシーバー入力/出力用に作成したポートを <tx_component_name1>.v モジュールのインスタンス化の同じバス信号に接続します。必要な変更内容を、次のコードに示します。

```
<tx_component_name1> <tx_component_name1>_i
(
.....
.GT_IN_SMPX_RX_IN_TX_OUT      (GT_IN_SMPX_RX_IN_TX_OUT),
.GT_OUT_SMPX_RX_OUT_TX_IN     (GT_OUT_SMPX_RX_OUT_TX_IN),
.....
);
```

シンプレックス TX コアの <tx_component_name1>_exdes.v の編集

<tx_component_name1>_exdes.v ファイルに必要な変更箇所は 2 つあります。

変更内容 1

次のバス信号を追加します。

```
output [140:0] GT_IN_SMPX_RX_IN_TX_OUT;
input  [1:0]   GT_OUT_SMPX_RX_OUT_TX_IN;
input  [7:0]   SHARED_LOGIC_MASTER_TO_SLAVE
```

注記：上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

次のコードに示すように、新規に作成した I/O ポートを <tx_component_name1>.v モジュールのインスタンス化の各ポートに接続します。

```
<tx_component_name1>_support <tx_component_name1>_block_i
(
.....
.GT_IN_SMPX_RX_IN_TX_OUT      (GT_IN_SMPX_RX_IN_TX_OUT),
.GT_OUT_SMPX_RX_OUT_TX_IN     (GT_OUT_SMPX_RX_OUT_TX_IN),
.SHARED_LOGIC_MASTER_TO_SLAVE (SHARED_LOGIC_MASTER_TO_SLAVE),
.....
);
```

シンプレックス RX コアの <rx_component_name1>_gtx.v の編集

シンプレックス RX コアの TX_DATA_WIDTH パラメーターはデフォルトで 32 に設定されます。GT の TX インターフェイスはシンプレックス TX に使用し、64 ビット データ インターフェイスが必要なため、このパラメーター値を 64 に変更する必要があります。

シンプレックス RX コアの <rx_component_name1>_wrapper.v の編集

シンプレックス TX コアの <rx_component_name1>_wrapper.v モジュールでグループ化したすべての GTX トランシーバー入力/出力信号を、シンプレックス RX コアの <rx_component_name1>_wrapper.v の GTX トランシーバーの TX インターフェイス ポートに assign 文で接続します。これらのバス信号以外に、シンプレックス RX コアからシンプレックス TX コアへいくつかの GT TX 信号を接続する必要があります。

<rx_component_name1>_wrapper ファイルに必要な変更箇所は 3 つあります。

変更内容 1

<rx_component_name1>_wrapper モジュールに次のバス信号を追加します。これらの信号は GTX トランシーバー専用で、シンプレックス RX コアの GTX トランシーバーに接続するために使用します。

```
input  [140:0] GT_IN_SMPX_TX_OUT_RX_IN;
output [1:0]   GT_OUT_SMPX_TX_IN_RX_OUT;
output                TXN_OUT;
output                TXP_OUT;
output                TXN_OUT_lane1;
output                TXP_OUT_lane1;
input                TXUSRCLK2_IN;
input                TXUSRCLK_IN;
```

注記: 上記のポート幅は TX コアのレーン数によって決まります。

次のワイヤ信号を宣言します。

```
wire [1:0]          tx_hdr_r;
wire [1:0]          tx_hdr_lane1_r;
wire [63:0]         scrambled_data_i;
wire [63:0]         scrambled_data_lane1_i;
```

変更内容 2

次に示すように、新規に作成したポートに GT インターフェイス信号を assign 文で接続します。

```
assign scrambled_data_lane1_i = GT_IN_SMPX_TX_OUT_RX_IN[63:0];
assign tx_hdr_lane1_r         = GT_IN_SMPX_TX_OUT_RX_IN[65:64];
assign txsequence_i          = GT_IN_SMPX_TX_OUT_RX_IN[72:66];
assign scrambled_data_i      = GT_IN_SMPX_TX_OUT_RX_IN[136:73];
assign tx_hdr_r              = GT_IN_SMPX_TX_OUT_RX_IN[138:137];
assign txuserddy_t           = GT_IN_SMPX_TX_OUT_RX_IN[139];
assign gttxreset_t           = GT_IN_SMPX_TX_OUT_RX_IN[140];

assign GT_OUT_SMPX_TX_IN_RX_OUT = {tx_resetdone_lane1_i,tx_resetdone_i};
```

変更内容 3

すべての GT TX インターフェイス ポートを <rx_component_name1>_MULTI_GT モジュールに接続します。このモジュールのインスタンス化に必要な変更内容を、次のコードに示します。

```
<rx_component_name1>_MULTI_GT#
(
    .WRAPPER_SIM_GTRESET_SPEEDUP (SIM_GTXRESET_SPEEDUP)
)
<rx_component_name1>_multi_gt_i
(
    .....
    //----- TX Initialization and Reset Ports -----
    .GT0_GTTXRESET_IN          (gtxreset_t),
    .GT0_TXUSERRDY_IN         (txuserddy_t),
    //----- Transmit Ports - 64b66b and 64b67b Gearbox Ports -----
    .GT0_TXHEADER_IN          (tx_hdr_r),
    //----- Transmit Ports - FPGA TX Interface Ports -----
    .GT0_TXUSRCLK_IN          (TXUSRCLK_IN),
    .GT0_TXUSRCLK2_IN         (TXUSRCLK2_IN),
    //----- Transmit Ports - TX Data Path interface -----
    .GT0_TXDATA_IN            (scrambled_data_i),
    //----- Transmit Ports - TX Driver and OOB signaling -----
    .GT0_GTXTXN_OUT           (TXN_OUT),
    .GT0_GTXTXP_OUT           (TXP_OUT),
    .GT0_TXSEQUENCE_IN        (txsequence_i),
    .GT0_TXRESETDONE_OUT      (tx_resetdone_i),
    //----- TX Initialization and Reset Ports -----
    .GT1_GTTXRESET_IN          (gtxreset_t),
    .GT1_TXUSERRDY_IN         (txuserddy_t),
    //----- Transmit Ports - 64b66b and 64b67b Gearbox Ports -----
    .GT1_TXHEADER_IN          (tx_hdr_lanel_r),
    //----- Transmit Ports - FPGA TX Interface Ports -----
    .GT1_TXUSRCLK_IN          (TXUSRCLK_IN ),
    .GT1_TXUSRCLK2_IN         (TXUSRCLK2_IN),
    //----- Transmit Ports - TX Data Path interface -----
    .GT1_TXDATA_IN            (scrambled_data_lanel_i),
    //----- Transmit Ports - TX Driver and OOB signaling -----
    .GT1_GTXTXN_OUT           (TXN_OUT_lanel),
    .GT1_GTXTXP_OUT           (TXP_OUT_lanel),
    .GT1_TXSEQUENCE_IN        (txsequence_i),
    .GT1_TXRESETDONE_OUT      (tx_resetdone_lanel_i),
    .....
);
```

シンプレックス RX コアの <rx_component_name1>_core.v の編集

シンプレックス RX コアの <rx_component_name1>_exdes モジュールで新規に作成したすべてのポートおよびバス信号は、階層内のさまざまなモジュールを経由してラッパー モジュールまで伝搬する必要があります。このため、<rx_component_name1>_core.v ファイルに次の 2 つの変更が必要です。

変更内容 1

<rx_component_name1>_core モジュールに次のポートを追加します。

```
input  [140:0] GT_IN_SMPX_TX_OUT_RX_IN;
output [1:0]   GT_OUT_SMPX_TX_IN_RX_OUT;
output                TXN_OUT;
output                TXP_OUT;
output                TXN_OUT_lane1;
output                TXP_OUT_lane1;
input                TXUSRCLK2_IN;
input                TXUSRCLK_IN;
```

注記: 上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

次のコードに示すように、新規に作成した I/O ポートを <rx_component_name1>_wrapper モジュールのインスタンスエーションの各ポートに接続します。

```
<rx_component_name1>_WRAPPER  #
(
.....
)
<rx_component_name1>_wrapper_i
(
.....
    .GT_IN_SMPX_TX_OUT_RX_IN          (GT_IN_SMPX_TX_OUT_RX_IN),
    .GT_OUT_SMPX_TX_IN_RX_OUT        (GT_OUT_SMPX_TX_IN_RX_OUT),
    .TXN_OUT                          (TXN_OUT),
    .TXP_OUT                          (TXP_OUT),
    .TXN_OUT_lane1                   (TXN_OUT_lane1),
    .TXP_OUT_lane1                   (TXP_OUT_lane1),
    .TXUSRCLK_IN                     (TXUSRCLK_IN),
    .TXUSRCLK2_IN                    (TXUSRCLK2_IN),
.....
);
```

シンプレックス RX コアの <rx_component_name1>_support.v の編集

<rx_component_name1>_support.v ファイルに必要な変更箇所は 3 つあります。

変更内容 1

<rx_component_name1>_support モジュールに次のポートを追加します。

```
input  [140:0] GT_IN_SMPX_TX_OUT_RX_IN;
output [1:0]   GT_OUT_SMPX_TX_IN_RX_OUT;
output                TXN_OUT;
output                TXP_OUT;
output                TXN_OUT_lane1;
output                TXP_OUT_lane1;
output                sync_clk_out;
```

変更内容 2

<rx_component_name1>_core モジュールのインスタンス化の各信号に接続します。必要な変更内容を、次のコードに示します。

```

<rx_component_name1>_core <rx_component_name1>_core_i
(
....

    .GT_IN_SMPX_TX_OUT_RX_IN      (GT_IN_SMPX_TX_OUT_RX_IN),
    .GT_OUT_SMPX_TX_IN_RX_OUT    (GT_OUT_SMPX_TX_IN_RX_OUT),
    .TXN_OUT                      (TXN_OUT),
    .TXP_OUT                      (TXP_OUT),
    .TXN_OUT_lane1               (TXN_OUT_lane1),
    .TXP_OUT_lane1               (TXP_OUT_lane1),
    .TXUSRCLK_IN                 (sync_clk_i),
    .TXUSRCLK2_IN                (user_clk_i),

    ...
);

```

変更内容 3

次の assign 文を追加して sync_clk をシンプレックス TX コアへ伝搬させます。

```
assign sync_clk_out = sync_clk_i;
```

シンプレックス RX コアの <rx_component_name1>.v の編集

<rx_component_name1>.v ファイルに必要な変更箇所は 2 つあります。

変更内容 1

<rx_component_name1>.v モジュールに次のポートを追加します。

```

input  [140:0] GT_IN_SMPX_TX_OUT_RX_IN;
output [1:0]   GT_OUT_SMPX_TX_IN_RX_OUT;
output                TXN_OUT;
output                TXP_OUT;
output                TXN_OUT_lane1;
output                TXP_OUT_lane1;
output                sync_clk_out;

```

注記: 上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

これらのポートを <rx_component_name1>_support.v モジュールのインスタンス化の各信号に接続します。必要な変更内容を、次のコードに示します。

```
<rx_component_name1>_support inst
(
.....
.GT_OUT_SMPX_TX_IN_RX_OUT      (GT_OUT_SMPX_TX_IN_RX_OUT),
.GT_IN_SMPX_TX_OUT_RX_IN      (GT_IN_SMPX_TX_OUT_RX_IN),
.TXN_OUT                        (TXN_OUT),
.TXP_OUT                        (TXP_OUT),
.TXN_OUT_lane1                 (TXN_OUT_lane1),
.TXP_OUT_lane1                 (TXP_OUT_lane1),
.sync_clk_out                  (sync_clk_out),
.....
);
```

シンプレックス RX コアの <rx_component_name1>_exdes.v の編集

シンプレックス 4 レーン RX デザインの最上位ファイルで 3 つのワイヤを定義し、共有ロジック、およびシンプレックス 2 レーン TX インスタンス化とシンプレックス 4 レーン RX インスタンス化間の GTX トランシーバー TX/RX 信号と接続する必要があります。また、RESET 信号がシンプレックス TX コアとシンプレックス RX コアの両方にあります。したがって、このモジュールの既存の RESET 入力を削除して、これら両方のコアの RESET 入力に接続するための 2 つの RESET ポートを新規に作成します。また、シンプレックス TX コアからのすべてのステータス信号をデザインの最上位モジュール (このデザインではシンプレックス RX コアの <rx_component_name1>_exdes.v) に追加します。

<rx_component_name1>.v ファイルに必要な変更箇所は 4 つあります。

変更内容 1

次のワイヤ信号を作成します。

```
wire [140:0] gt_in_smpx_tx_out_rx_in_i;
wire [1:0]   gt_out_smpx_tx_in_rx_out_i;
wire [7:0]   shared_logic_master_to_slave;
```

次のポートを削除します。

RESET

次のポートを作成します。

```
input          TX_RESET;
input          RX_RESET;
output         TX_HARD_ERR;
output         TX_SOFT_ERR;
output [0:1]   TX_LANE_UP;
output         TX_CHANNEL_UP;
output         TXN_OUT;
output         TXP_OUT;
output         TXN_OUT_LANE1;
output         TXP_OUT_LANE1;
```

変更内容 2

シンプレックス RX コアの <rx_component_name1>_exdes.v は、シンプレックス 4 レーン RX デザインとシンプレックス 2 レーン TX デザインで構成されるデザイン 1 の最上位 HDL ソース ファイルです。使いやすさを考慮し、この最上位ファイルはシンプレックス 4 レーン RX サンプル デザインの最上位ファイルをベースに作成し、この中にシンプレックス 2 レーン TX サンプル デザインの最上位ファイルをインスタンスシートしています。

シンプレックス TX コアの <tx_component_name1>_exdes をインスタンスシートし、必要な信号モジュールを接続します。必要な変更内容を、次のコードに示します。

```
<tx_component_name1>_exdes #
(
  .EXAMPLE_SIMULATION (EXAMPLE_SIMULATION),
  .USE_CORE_TRAFFIC   (USE_CORE_TRAFFIC),
  .USE_LABTOOLS       (USE_LABTOOLS)
)
<tx_component_name1>_exdes_inst
(
  // User IO
  .TX_HARD_ERR   (TX_HARD_ERR),
  .TX_SOFT_ERR   (TX_SOFT_ERR),
  .TX_LANE_UP    (TX_LANE_UP),
  .TX_CHANNEL_UP (TX_CHANNEL_UP),
  .INIT_CLK_P    (tied_to_ground_i),
  .INIT_CLK_N    (tied_to_ground_i),
  .PMA_INIT      (PMA_INIT),
  .GTXQ0_P       (tied_to_ground_i),
  .GTXQ0_N       (tied_to_ground_i),

  // GTX I/O
  .TXP(),
  .TXN(),
  .GT_IN_SMPX_RX_IN_TX_OUT   (gt_in_smpx_tx_out_rx_in_i),
  .GT_OUT_SMPX_RX_OUT_TX_IN  (gt_out_smpx_tx_in_rx_out_i),
  .SHARED_LOGIC_MASTER_TO_SLAVE (shared_logic_master_to_slave),

  .RESET(TX_RESET)
);
```

変更内容 3

次のコードに示すように、SHARED_LOGIC_MASTER_TO_SLAVE バスを利用してシンプレックス RX からの共有ロジック信号をシンプレックス TX へ assign 文で接続します。必要な変更内容を、次のコードに示します。

```
assign shared_logic_master_to_slave[0] = user_clk_i;
assign shared_logic_master_to_slave[1] = sync_clk_i;
assign shared_logic_master_to_slave[2] = INIT_CLK_i;
assign shared_logic_master_to_slave[3] = refclk1_in_i;
assign shared_logic_master_to_slave[4] = gt_qpllclk_quad1_i;
assign shared_logic_master_to_slave[5] = gt_qpllrefclk_quad1_i;
assign shared_logic_master_to_slave[6] = gt_qplllock_i;
assign shared_logic_master_to_slave[7] = pll_not_locked_i;
```

変更内容 4

バス信号を <rx_component_name1> モジュールのインスタンスエーションの同じバス信号に接続します。必要な変更内容を、次のコードに示します。

```
<rx_component_name1> <rx_component_name1>_block_i
(
.....
.GT_IN_SMPX_TX_OUT_RX_IN      (gt_in_smpx_tx_out_rx_in_i),
.GT_OUT_SMPX_TX_IN_RX_OUT    (gt_out_smpx_tx_in_rx_out_i),
.TXN_OUT                      (TXN_OUT),
.TXP_OUT                      (TXP_OUT),
.TXN_OUT_lane1               (TXN_OUT_LANE1),
.TXP_OUT_lane1               (TXP_OUT_LANE1),
.gt_qplllock_out             (gt_qplllock_i),
.gt_qpllrefclk_quad1_out     (gt_qpllrefclk_quad1_i),
.gt_qpllclk_quad1_out        (gt_qpllclk_quad1_i),
.gt_refclk1_out              (refclk1_in_i),
.reset_pb                    (RX_RESET),
.sync_clk_out                 (sync_clk_i),
.....
);
```

ハードウェアの作成 – デザイン 2 (4 レーン シンプレックス TX + 2 レーン シンプレックス RX)

デザイン 2 の作成手順は、デザイン 1 の場合とほぼ共通です。特に明記のない限り、変更が必要な理由も同じです。

デザイン 1 の作成に必要な手順は、次のとおりです。

1. 4 レーン シンプレックス TX コアを生成する
2. 2 レーン シンプレックス RX コアを生成する
3. シンプレックス RX Aurora コアの GTX トランシーバー接続を削除する
4. シンプレックス RX コアからの GTX トランシーバー ポート接続をシンプレックス TX コアに配線する

4 レーン シンプレックス TX コアを生成する

次の手順に従って Aurora 64B/66B コアをカスタマイズし、4 レーン シンプレックス TX コアを生成します。

1. Vivado Design Suite を起動します。
2. [Create New Project] をクリックして [Next] をクリックします。
3. プロジェクト名とパスを選択して [Next] をクリックします。
4. [RTL Project] をオンにしてサンプルデザインの実行を許可し、[Do not specify sources at this time] をオンにします。[Next] をクリックします。
5. [XC7VX485TFFG1761-3] または GTX トランシーバーを内蔵したターゲット デバイスをクリックします。
6. [Next] をクリックして [Finish] をクリックします。
7. Flow Navigator の [Project Manager] 下にある [IP catalog] をクリックし、「Aurora 64B66B」を検索します。

Aurora コアは、[Communication & Networking] → [Serial Interfaces] の下にあります。図 3 を参照してください。

8. [Aurora 64B66B] を右クリックして [Customize IP] をクリックします。
9. [Customize IP] ダイアログ ボックスの [Core Options] タブで、[Vivado Lab Tools] をオンにします。コアのオプションは、すでに生成したほかのコアと同様に設定してください。

10. 図 13 のようにコンフィギュレーションオプションを設定します。

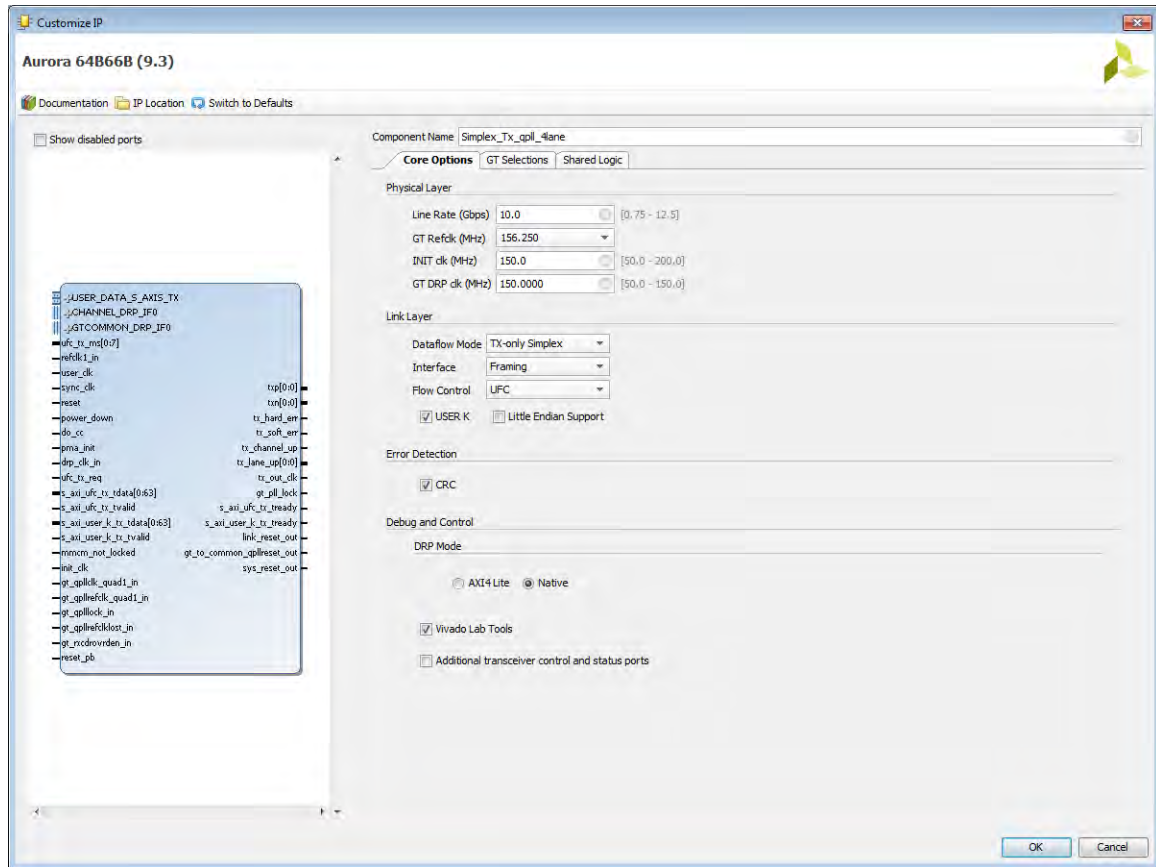


図 13 : Aurora 64B/66B コアを 4 レーン シンプレックス TX コアとして生成する場合の Vivado IDE の設定

11. [GT Selections] タブをクリックします。ターゲット ボードに従って GT を選択します。
12. [Shared Logic] タブをクリックします。[include Shared Logic in core] をオンにして [OK] をクリックします。

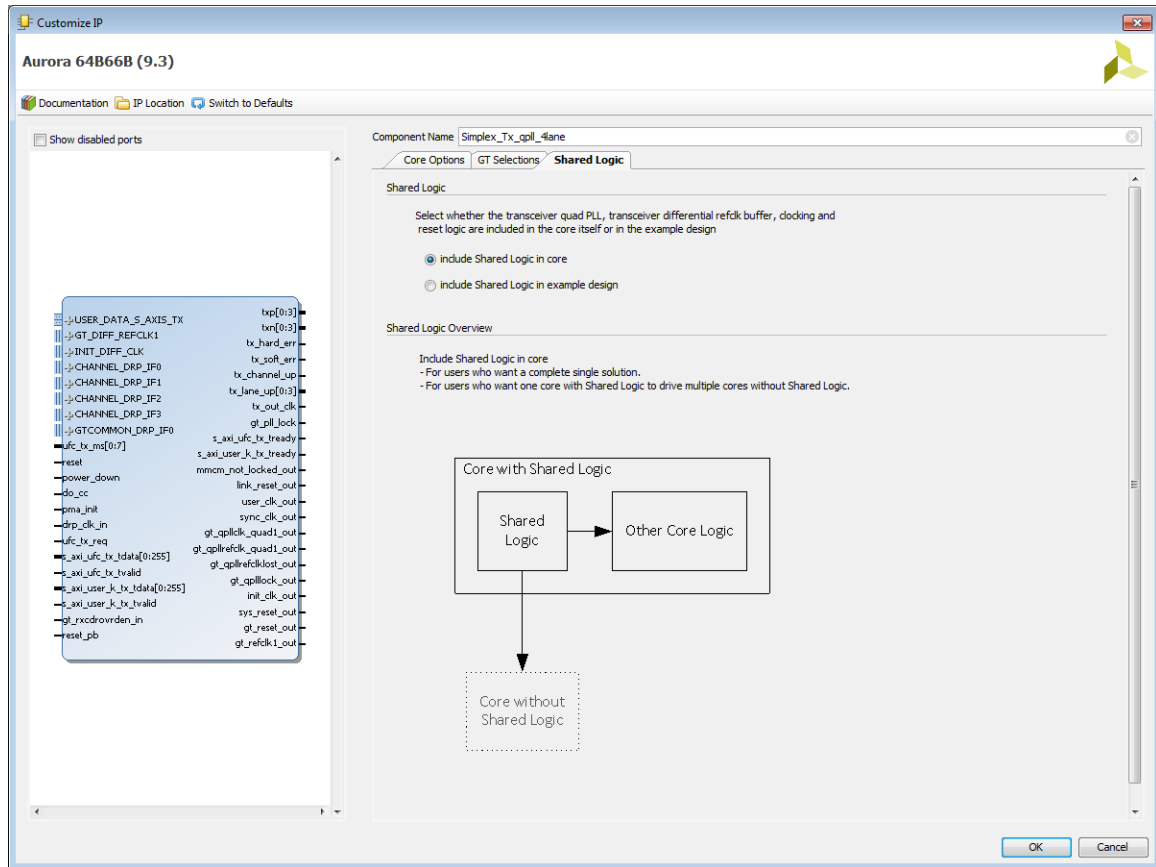


図 14 : 4 レーン シンプレックス TX コアの共有ロジック設定

13. [Generate Output Products] ダイアログ ボックスで、[Generate Synthesized checkpoint (.dcp)] がオフであることを確認して [Generate] をクリックします。
14. [Generate Synthesized Checkpoint] ダイアログ ボックスで [OK] をクリックします。
15. [.xci] を右クリックして [Open IP Example Design] をクリックします。[Open IP Example Design] ダイアログ ボックスでターゲット ディレクトリを選択し、[OK] をクリックします。

2 レーン シンプレックス RX コアを生成する

次の手順に従って Aurora 64B/66B コアをカスタマイズし、2 レーン シンプレックス RX コアを生成します。

1. Flow Navigator の [Project Manager] 下にある [IP catalog] をクリックし、「Aurora 64B66B」を検索します。

Aurora コアは、[Communication & Networking] → [Serial Interfaces] の下にあります。図 3 を参照してください。

2. [Aurora 64B66B] を右クリックして [Customize IP] をクリックします。
3. [Customize IP] ダイアログ ボックスの [Core Options] タブで、[Vivado Lab Tools] をオンにします。
4. 図 15 のようにコンフィギュレーション オプションを設定します。各オプションは、[Dataflow Mode] を除き 4 レーン TX デザインと同じ設定にしてください。

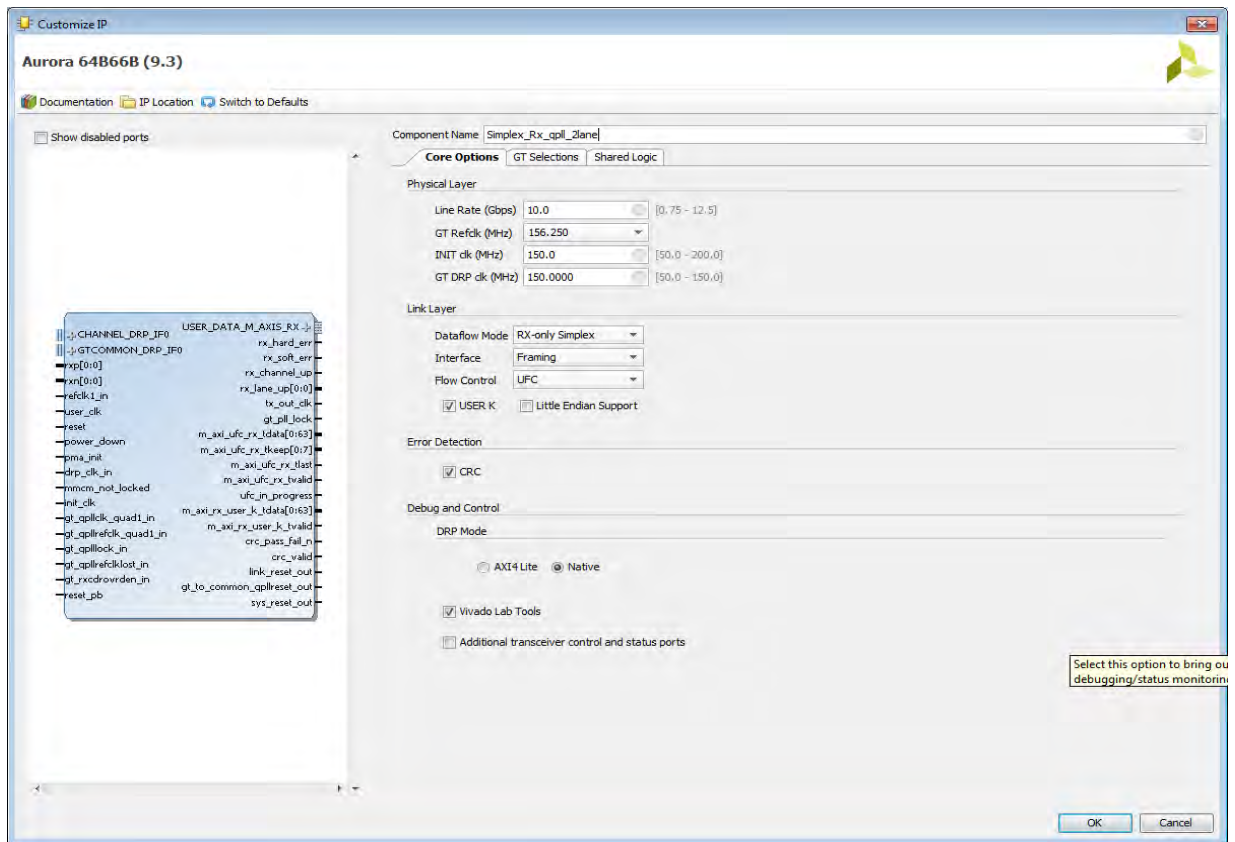


図 15 : Aurora 64B/66B コアを 2 レーン シンプレックス RX コアとして生成する場合の Vivado IDE の設定

5. [GT Selections] タブをクリックします。ターゲット ボードに従って GT を選択します。
6. [Shared Logic] タブをクリックします。[include Shared Logic in example design] をオンにして [OK] をクリックします。
7. [Generate Output Products] ダイアログ ボックスで、[Generate Synthesized checkpoint (.dcp)] がオフであることを確認して [Generate] をクリックします。
8. [Generate Synthesized Checkpoint] ダイアログ ボックスで [OK] をクリックします。
9. [.xci] を右クリックして [Open IP Example Design] をクリックします。[Open IP Example Design] ダイアログ ボックスでターゲット ディレクトリを選択し、[OK] をクリックします。

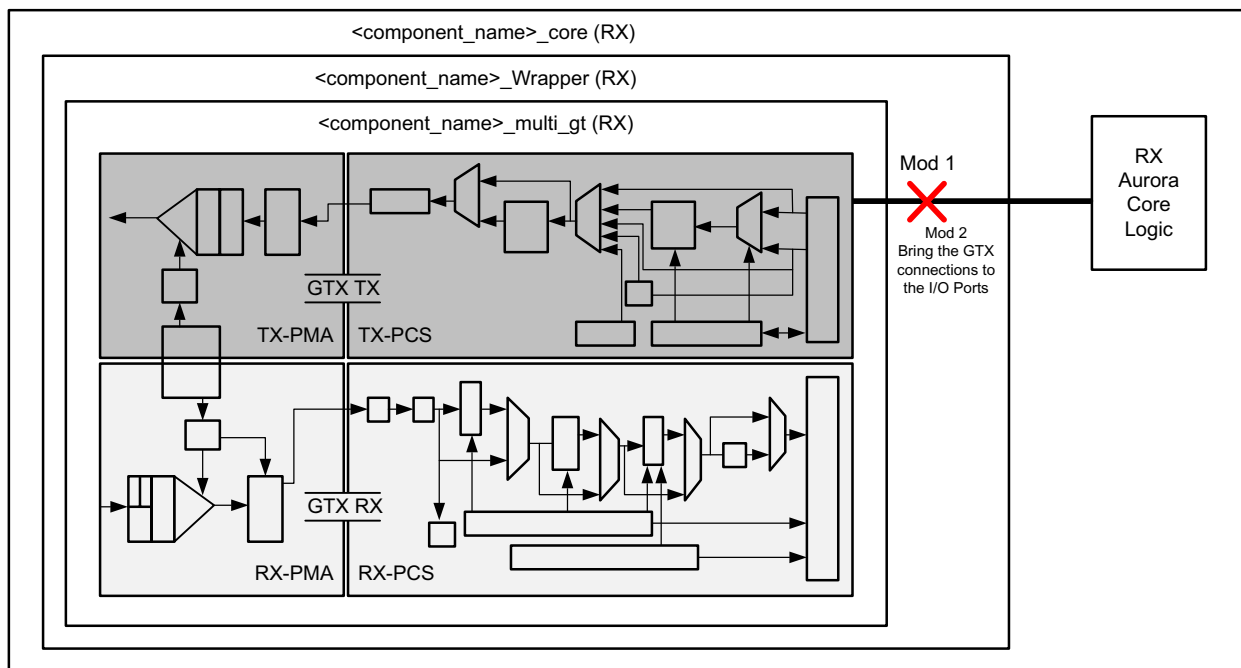
シンプレックス RX Aurora コアの GTX トランシーバー接続を削除する

基本的な考え方として、シンプレックス RX Aurora デザインから GTX トランシーバーのインスタンスエーションを削除し、シンプレックス TX Aurora デザインにインスタンスエーションされた GTX トランシーバー (RX 側) に必要な信号を接続します。<rx_component_name2>_wrapper.v ファイルには次の変更が必要です。

変更内容

GTX トランシーバーのインスタンスエーションを削除するには、<rx_component_name2>_wrapper.v ファイルで次のコンポーネントをコメントアウトします。

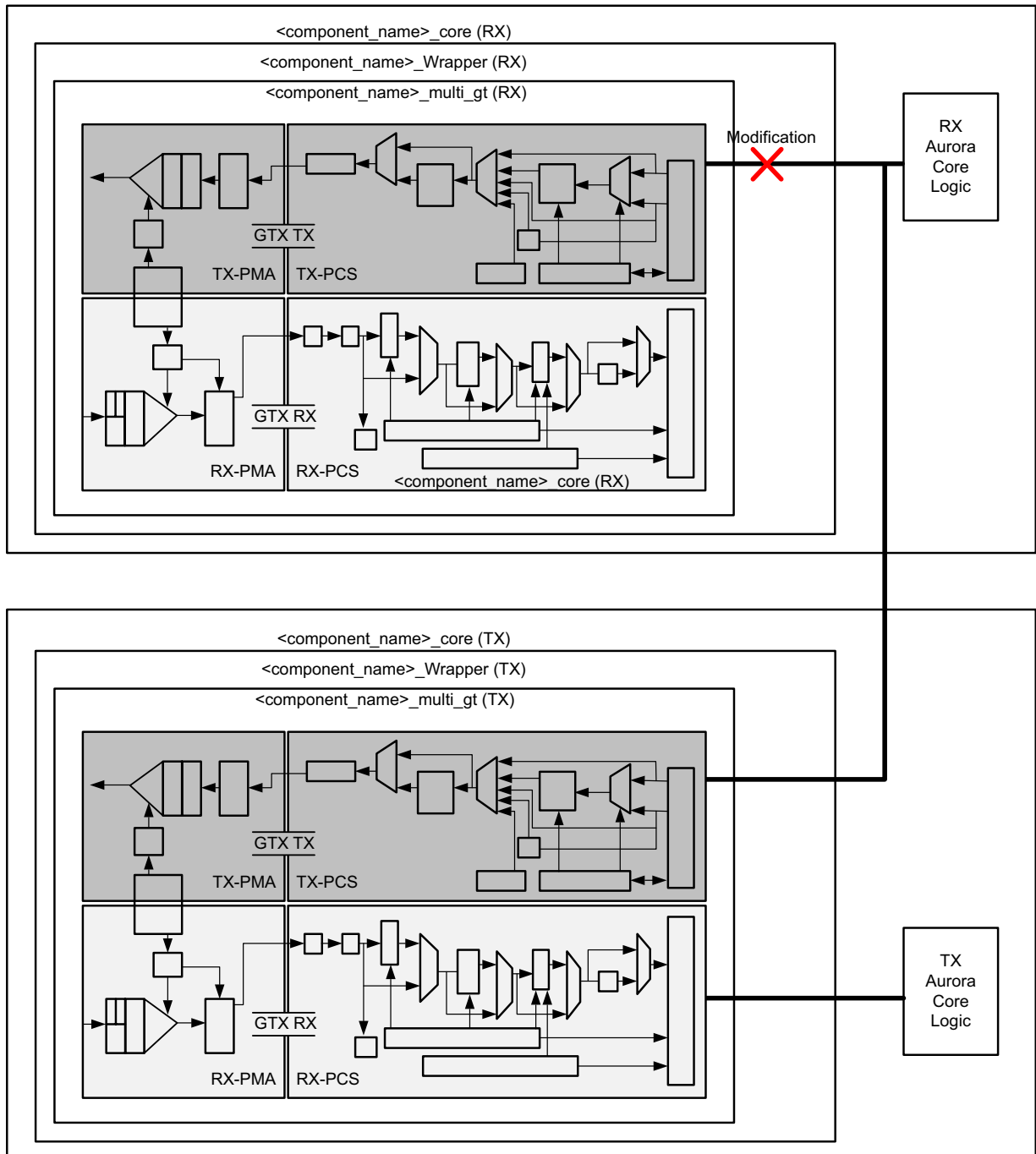
```
< rx_component_name2>_MULTI_GT
```



X14360

図 16: シンプレックス RX デザインの GTX インスタンスエーションを削除

シンプレックス RX コアからの GTX トランシーバー ポート接続をシンプレックス TX コアに接続する



X14357

図 17: シンプレックス RX 2 レーン コアからの信号をシンプレックス TX 4 レーン コアに接続

ここでは、シンプレックス RX コアからの GTX トランシーバー入力をシンプレックス TX コアに接続します。シンプレックス RX コアに必要な GTX トランシーバー出力を、シンプレックス TX Aurora コアの最上位に接続した後、シンプレックス RX コアに接続します。図 17 に、この手順で実行する変更を示します。

この手順では、シンプレックス RX コアの次のファイルを編集します。

- <rx_component_name2>_wrapper.v
- <rx_component_name2>_core.v
- <rx_component_name2>.v
- <rx_component_name2>_support.v
- <rx_component_name2>_exdes.v

この手順では、シンプレックス TX コアの次のファイルを編集します。

- <tx_component_name2>_gtx.v
- <tx_component_name2>_wrapper.v
- <tx_component_name2>_core.v
- <tx_component_name2>.v
- <tx_component_name2>_exdes.v

次に、これらの各ファイルで必要な変更内容について説明します。

シンプレックス RX コアの <rx_component_name2>_wrapper.v の編集

<rx_component_name2>_wrapper ファイルに必要な変更箇所は 2 つあります。

変更内容 1

<rx_component_name2>_wrapper モジュールに次のポートを追加します。これらのポートは GTX トランシーバー専用で、シンプレックス RX コアの GTX トランシーバーに接続するために使用します。

```
input  [81:0]    GT_OUT_SMPX_TX_OUT_RX_IN;
output [6:0]    GT_IN_SMPX_TX_IN_RX_OUT;
```

注記: 上記のポート幅は RX コアのレーン数によって決まります。

次のワイヤ信号を宣言します。

```
wire [2:0] gt0_rxbufstatus_out;
wire [2:0] gt1_rxbufstatus_out;
```

変更内容 2

次のコードに示すように、GT インターフェイス信号を新規に作成したポートに assign 文で接続します。

```

assign pre_rxddata_from_gtx_i      = GT_OUT_SMPX_TX_OUT_RX_IN[31:0];
assign gt0_rxbufstatus_out         = GT_OUT_SMPX_TX_OUT_RX_IN[34:32];
assign rxrecclk_from_gtx_i        = GT_OUT_SMPX_TX_OUT_RX_IN[35];
assign pre_rxddatavalid_i         = GT_OUT_SMPX_TX_OUT_RX_IN[36];
assign pre_rxheader_from_gtx_i    = GT_OUT_SMPX_TX_OUT_RX_IN[38:37];
assign pre_rxheadervalid_i        = GT_OUT_SMPX_TX_OUT_RX_IN[39];
assign rx_resetdone_i             = GT_OUT_SMPX_TX_OUT_RX_IN[40];
assign pre_rxddata_from_gtx_lane1_i = GT_OUT_SMPX_TX_OUT_RX_IN[72:41];
assign gtl_rxbufstatus_out        = GT_OUT_SMPX_TX_OUT_RX_IN[75:73];
assign rxrecclk_from_gtx_lane1_i  = GT_OUT_SMPX_TX_OUT_RX_IN[76];
assign pre_rxddatavalid_lane1_i   = GT_OUT_SMPX_TX_OUT_RX_IN[77];
assign pre_rxheader_from_gtx_lane1_i = GT_OUT_SMPX_TX_OUT_RX_IN[79:78];
assign pre_rxheadervalid_lane1_i  = GT_OUT_SMPX_TX_OUT_RX_IN[80];
assign rx_resetdone_lane1_i       = GT_OUT_SMPX_TX_OUT_RX_IN[81];

assign GT_IN_SMPX_TX_IN_RX_OUT = {rxuserddy_t, sync_rx_polarity_r,
rxrecclk_to_fabric_i, rxgearboxslip_i, gtrxreset_i, sync_rx_polarity_lane1_r,
rxgearboxslip_lane1_i};

```

次の信号は共通で、スレーブ コアから独立しています。

- gtrxreset_t
- rxuserddy_t
- rxrecclk_to_fabric_i

次の信号は、スレーブ コアのレーンごとにマスター コアに接続する必要があります。

- pre_rxddata_from_gtx - GT からの RX データ出力
- rxbufstatus_out - RX バッファ ステータス
- rxrecclk_from_gtx - GT からリカバリしたクロック
- pre_rxddatavalid - RX データ Valid 信号
- pre_rxheader_from_gtx - GT からの RX ヘッダー出力
- pre_rxheadervalid - RX ヘッダー Valid 信号
- rx_resetdone - RX リセット Done 信号

シンプレックス RX コアの <rx_component_name2>_core.v の編集

<rx_component_name2>_core.v ファイルに必要な変更箇所は 2 つあります。

変更内容 1

<rx_component_name2>_core モジュールに次のポートを追加します。

```
input   [81:0]   GT_OUT_SMPX_TX_OUT_RX_IN;
output  [6:0]    GT_IN_SMPX_TX_IN_RX_OUT;
```

注記: 上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

新規に作成したポートを <rx_component_name2>_wrapper モジュールのインスタンスエーションの各バス信号に接続します。必要な変更内容を、次のコードに示します。

```
Simplex_RX_qpll_2lane_WRAPPER  #
(
.....
)
Simplex_RX_qpll_2lane_wrapper_i
(
.....
.GT_OUT_SMPX_TX_OUT_RX_IN      (GT_OUT_SMPX_TX_OUT_RX_IN),
.GT_IN_SMPX_TX_IN_RX_OUT      (GT_IN_SMPX_TX_IN_RX_OUT),
.....
);
```

シンプレックス RX コアの <rx_component_name2>.v の編集

<rx_component_name2>.v ファイルに必要な変更箇所は 2 つあります。

変更内容 1

<rx_component_name2>.v モジュールに次のポートを追加します。

```
input   [81:0]   GT_OUT_SMPX_TX_OUT_RX_IN;
output  [6:0]    GT_IN_SMPX_TX_IN_RX_OUT;
```

注記: 上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

新規に作成したポートを <rx_component_name2>_core.v モジュールのインスタンスエーションの各バス信号に接続します。必要な変更内容を、次のコードに示します。

```
<rx_component_name2>_core inst
(
.....
.GT_OUT_SMPX_TX_OUT_RX_IN      (GT_OUT_SMPX_TX_OUT_RX_IN),
.GT_IN_SMPX_TX_IN_RX_OUT      (GT_IN_SMPX_TX_IN_RX_OUT),
.....
);
```

シンプレックス RX コアの <rx_component_name2>_support.v の編集

<rx_component_name2>_support.v ファイルに必要な変更箇所は 4 つあります。

変更内容 1

次のポートを追加します。

```
input  [7:0]    SHARED_LOGIC_MASTER_TO_SLAVE,
input  [81:0]   GT_OUT_SMPX_TX_OUT_RX_IN,
output [6:0]    GT_IN_SMPX_TX_IN_RX_OUT
```

注記：上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

新規に作成したポートを <rx_component_name2>.v モジュールのインスタンスエーションの適切なバス信号に接続します。必要な変更内容を、次のコードに示します。

```
<rx_component_name2> <rx_component_name2>_i
(
.....
.GT_OUT_SMPX_TX_OUT_RX_IN      (GT_OUT_SMPX_TX_OUT_RX_IN),
.GT_IN_SMPX_TX_IN_RX_OUT      (GT_IN_SMPX_TX_IN_RX_OUT),
.....
);
```

変更内容 3

共有ロジック信号を各信号に assign 文で接続します。

```
assign user_clk_i           = SHARED_LOGIC_MASTER_TO_SLAVE[0];
assign sync_clk_i          = SHARED_LOGIC_MASTER_TO_SLAVE[1];
assign INIT_CLK_i          = SHARED_LOGIC_MASTER_TO_SLAVE[2];
assign refclk1_in_i        = SHARED_LOGIC_MASTER_TO_SLAVE[3];
assign gt_qpllclk_quad1_i  = SHARED_LOGIC_MASTER_TO_SLAVE[4];
assign gt_qpllrefclk_quad1_i = SHARED_LOGIC_MASTER_TO_SLAVE[5];
assign gt_qplllock_i       = SHARED_LOGIC_MASTER_TO_SLAVE[6];
assign pll_not_locked_i    = SHARED_LOGIC_MASTER_TO_SLAVE[7];
assign mmcm_not_locked_i   = pll_not_locked_i;
assign refclk1_in          = refclk1_in_i;
```

変更内容 4

次のモジュール インスタンス化をコメントアウトします。

- Simplex_RX_qpll_2lane_gt_common_wrapper
- IBUFDS_GTE2
- Simplex_RX_qpll_2lane_CLOCK_MODULE

シンプレックス RX コアの <rx_component_name2>_exdes.v の編集

<rx_component_name2>_exdes.v ファイルに必要な変更箇所は 2 つあります。

変更内容 1

次のバス信号を追加します。

```
input   [7:0]      SHARED_LOGIC_MASTER_TO_SLAVE;
input   [81:0]     GT_OUT_SMPX_TX_OUT_RX_IN;
output  [6:0]      GT_IN_SMPX_TX_IN_RX_OUT;
```

注記 : 上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

新規に作成したポートを <rx_component_name2>_support.v モジュールのインスタンス化の適切なバス信号に接続します。必要な変更内容を、次のコードに示します。

```
<rx_component_name2>_support <rx_component_name2>_support_block_i
(
.....
.GT_OUT_SMPX_TX_OUT_RX_IN      (GT_OUT_SMPX_TX_OUT_RX_IN),
.GT_IN_SMPX_TX_IN_RX_OUT      (GT_IN_SMPX_TX_IN_RX_OUT),
.SHARED_LOGIC_MASTER_TO_SLAVE (SHARED_LOGIC_MASTER_TO_SLAVE),
.....
);
```

シンプレックス TX コアの <tx_component_name2>_wrapper.v の編集

<tx_component_name2>_wrapper ファイルに必要な変更箇所は 3 つあります。

変更内容 1

<tx_component_name2>_wrapper モジュールに次のポートを追加します。これらのポートは GTX トランシーバー専用で、シンプレックス RX コアの GTX トランシーバーに接続するために使用します。

```
output [81:0] GT_OUT_SMPX_RX_IN_TX_OUT;
input  [6:0]  GT_IN_SMPX_RX_OUT_TX_IN;
input          RX1N_IN;
input          RX1P_IN;
input          RX1N_IN_LANE1;
input          RX1P_IN_LANE1;
```

注記 : 上記のポート幅は TX コアのレーン数によって決まります。

次のワイヤ信号を宣言します。

```
wire [1:0]          pre_rxheader_from_gtx_i;
wire [31:0]        pre_rxdata_from_gtx_i;
wire [31:0]        pre_rxdata_from_gtx_lane1_i;
wire [1:0]          pre_rxheader_from_gtx_lane1_i;
wire rx_resetdone_lane1_i;
wire pre_rxheadervalid_lane1_i;
wire pre_rxdatabavalid_lane1_i;
wire rxrecclk_from_gtx_lane1_i;
wire rx_resetdone_i;
wire pre_rxheadervalid_i;
wire pre_rxdatabavalid_i;
wire rxrecclk_from_gtx_i;
wire [2:0] gt0_rxbufstatus_out;
wire [2:0] gt1_rxbufstatus_out;
```

変更内容 2

次に示すように、新規に作成したポートに GT インターフェイス信号を assign 文で接続します。

```
assign rxgearboxslip_lane1_i      = GT_IN_SMPX_RX_OUT_TX_IN[0];
assign sync_rx_polarity_lane1_r   = GT_IN_SMPX_RX_OUT_TX_IN[1];
assign gtrxreset_i                = GT_IN_SMPX_RX_OUT_TX_IN[2];
assign rxgearboxslip_i            = GT_IN_SMPX_RX_OUT_TX_IN[3];
assign rxrecclk_to_fabric_i       = GT_IN_SMPX_RX_OUT_TX_IN[4];
assign sync_rx_polarity_r         = GT_IN_SMPX_RX_OUT_TX_IN[5];
assign rxuserddy_t                = GT_IN_SMPX_RX_OUT_TX_IN[6];

assign GT_OUT_SMPX_RX_IN_TX_OUT   = {rx_resetdone_lane1_i, pre_rxheadervalid_lane1_i,
pre_rxheader_from_gtx_lane1_i, pre_rxdatabavalid_lane1_i, rxrecclk_from_gtx_lane1_i,
gt1_rxbufstatus_out, pre_rxdata_from_gtx_lane1_i, rx_resetdone_i, pre_rxheadervalid_i,
pre_rxheader_from_gtx_i, pre_rxdatabavalid_i, rxrecclk_from_gtx_i, gt0_rxbufstatus_out,
pre_rxdata_from_gtx_i};
```

変更内容 3

シリアルポート RXP/RXN を <tx_component_name2>_MULTI_GT モジュールからの出力に接続します。必要な変更内容を、次のコードに示します。

```

<component_name>_MULTI_GT#
(
    .WRAPPER_SIM_GTRXRESET_SPEEDUP (SIM_GTXRESET_SPEEDUP)
)
<component_name>_multi_gt_i
(
    .....
    //----- Receive Ports - RX Gearbox Ports -----
    .GT0_RXGEARBOXSLIP_IN      (rxgearboxslip_i),
    .GT1_RXGEARBOXSLIP_IN      (rxgearboxslip_lane1_i),

    //----- RX Initialization and Reset Ports -----
    .GT0_RXUSERRDY_IN          (rxuserddy_t),//(tied_to_ground_i),
    .GT0_RX_POLARITY_IN        (sync_rx_polarity_r),//(tied_to_ground_i),

    .GT1_RXUSERRDY_IN          (rxuserddy_t),//(tied_to_ground_i),
    .GT1_RX_POLARITY_IN        (sync_rx_polarity_lane1_r),
    .GT0_RXUSRCLK_IN           (rxrecclk_to_fabric_i),
    .GT0_RXUSRCLK2_IN          (rxrecclk_to_fabric_i),
    .GT1_RXUSRCLK_IN           (rxrecclk_to_fabric_i),
    .GT1_RXUSRCLK2_IN          (rxrecclk_to_fabric_i),
    .GT0_RXDATA_OUT            (pre_rxd_data_from_gtx_i),
    .GT1_RXDATA_OUT            (pre_rxd_data_from_gtx_lane1_i),

    //----- Receive Ports - RX Initialization and Reset Ports -----
    .GT0_GTRXRESET_IN          (gtrxreset_i),
    .GT1_GTRXRESET_IN          (gtrxreset_i),

    //----- Receive Ports - RX Gearbox Ports -----

    //----- Receive Ports -RX Initialization and Reset Ports -----
    .GT0_RXRESETDONE_OUT       (rx_resetdone_i),
    .GT1_RXRESETDONE_OUT       (rx_resetdone_lane1_i),

    //----- Receive Ports - RX Gearbox Ports -----
    .GT0_RXDATAVALID_OUT       (pre_rxd_datavalid_i),
    .GT0_RXHEADER_OUT          (pre_rxheader_from_gtx_i),
    .GT0_RXHEADERVALID_OUT     (pre_rxheadervalid_i),
    .GT1_RXDATAVALID_OUT       (pre_rxd_datavalid_lane1_i),
    .GT1_RXHEADER_OUT          (pre_rxheader_from_gtx_lane1_i),
    .GT1_RXHEADERVALID_OUT     (pre_rxheadervalid_lane1_i),

    //----- Receive Ports - RX Fabric Output Control Ports -----
    .GT0_RXOUTCLK_OUT          (rxrecclk_from_gtx_i),
    .GT1_RXOUTCLK_OUT          (rxrecclk_from_gtx_lane1_i),

    //----- Receive Ports - RX Buffer Bypass Ports -----
    .GT0_RXBUFSTATUS_OUT       (gt0_rxbufstatus_out),
    .GT1_RXBUFSTATUS_OUT       (gt1_rxbufstatus_out),

    .GT0_GTXRXN_IN             (RX1N_IN),
    .GT0_GTXRXP_IN             (RX1P_IN),
    .GT1_GTXRXN_IN             (RX1N_IN_LANE1),
    .GT1_GTXRXP_IN             (RX1P_IN_LANE1),
    .....

```

```
);
```

シンプレックス TX コアの <tx_component_name2>_core.v の編集

<tx_component_name2>_core.v ファイルに必要な変更箇所は 2 つあります。

変更内容 1

<tx_component_name2>_core モジュールに次のポートを追加します。

```
output [81:0] GT_OUT_SMPX_RX_IN_TX_OUT;
input [6:0] GT_IN_SMPX_RX_OUT_TX_IN;
input      RX1N_IN;
input      RX1P_IN;
input      RX1N_IN_LANE1;
input      RX1P_IN_LANE1;
```

注記：上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

新規に作成したポートを <tx_component_name2>_core モジュールのインスタンスエーションの適切なバス信号に接続します。必要な変更内容を、次のコードに示します。

```
<tx_component_name2>_WRAPPER#
(
.....
)
<tx_component_name2>_WRAPPER_i
(
.....
.GT_OUT_SMPX_RX_IN_TX_OUT          (GT_OUT_SMPX_RX_IN_TX_OUT),
.GT_IN_SMPX_RX_OUT_TX_IN           (GT_IN_SMPX_RX_OUT_TX_IN),
.RX1N_IN                            (RX1N_IN),
.RX1P_IN                            (RX1P_IN),
.RX1N_IN_LANE1                      (RX1N_IN_LANE1),
.RX1P_IN_LANE1                      (RX1P_IN_LANE1),
.....
);
```

シンプレックス TX コアの <tx_component_name2>_support.v の編集

<tx_component_name2>_support.v ファイルに必要な変更箇所は 2 つあります。

変更内容 1

<tx_component_name2> モジュールに次のポートを追加します。

```
output [81:0] GT_OUT_SMPX_RX_IN_TX_OUT;
input  [6:0]  GT_IN_SMPX_RX_OUT_TX_IN;
input          RX1N_IN;
input          RX1P_IN;
input          RX1N_IN_LANE1;
input          RX1P_IN_LANE1;
```

変更内容 2

新規に作成したポートを <tx_component_name2>_core モジュールのインスタンスエーションの信号に接続します。必要な変更内容を、次のコードに示します。

```
<tx_component_name2>_core <tx_component_name2>_core_i
(
.....
    .GT_OUT_SMPX_RX_IN_TX_OUT          (GT_OUT_SMPX_RX_IN_TX_OUT),
    .GT_IN_SMPX_RX_OUT_TX_IN          (GT_IN_SMPX_RX_OUT_TX_IN),
    .RX1N_IN                           (RX1N_IN),
    .RX1P_IN                           (RX1P_IN),
    .RX1N_IN_LANE1                    (RX1N_IN_LANE1),
    .RX1P_IN_LANE1                    (RX1P_IN_LANE1),
.....
);
```

シンプレックス TX コアの <tx_component_name2>.v の編集

<tx_component_name2>.v ファイルに必要な変更箇所は 2 つあります。

変更内容 1

<tx_component_name2>.v モジュールに次のポートを追加します。

```
output [81:0] GT_OUT_SMPX_RX_IN_TX_OUT;
input  [6:0]  GT_IN_SMPX_RX_OUT_TX_IN;
input          RX1N_IN;
input          RX1P_IN;
input          RX1N_IN_LANE1;
input          RX1P_IN_LANE1;
```

注記: 上記のポート幅は TX コアのレーン数によって決まります。

変更内容 2

新規に作成したポートを <tx_component_name2>.v モジュールのインスタンス化の適切なバス信号に接続します。必要な変更内容を、次のコードに示します。

```
<tx_component_name2>_support inst
(
.....
.GT_OUT_SMPX_RX_IN_TX_OUT      (GT_OUT_SMPX_RX_IN_TX_OUT),
.GT_IN_SMPX_RX_OUT_TX_IN      (GT_IN_SMPX_RX_OUT_TX_IN),
.RX1N_IN                        (RX1N_IN),
.RX1P_IN                        (RX1P_IN),
.RX1N_IN_LANE1                 (RX1N_IN_LANE1),
.RX1P_IN_LANE1                 (RX1P_IN_LANE1),
.....
);
```

シンプレックス TX コアの <tx_component_name2>_exdes.v の編集

<tx_component_name2>_exdes.v ファイルに必要な変更箇所は 4 つあります。

変更内容 1

次のワイヤ信号を作成します。

```
wire [81:0]  GT_OUT_SMPX_RX_IN_TX_OUT;
wire [6:0]   GT_IN_SMPX_RX_OUT_TX_IN;
wire [7:0]   shared_logic_master_to_slave;
```

次のポートを削除します。

RESET

次のポートを作成します。

```
input      TX_RESET;
input      RX_RESET;
output     RX_HARD_ERR;
output     RX_SOFT_ERR;
output [0:1] RX_LANE_UP;
output     RX_CHANNEL_UP;
input      RX1N_IN;
input      RX1P_IN;
input      RX1N_IN_LANE1;
input      RX1P_IN_LANE1;
output [0:7] DATA_ERR_COUNT;
output     UFC_ERR;
output     USER_K_ERR;
output     CRC_PASS_FAIL_N;
output     CRC_VALID;
```

変更内容 2

<rx_component_name2>_exdes モジュールをインスタンス化します。必要な変更内容を、次のコードに示します。

```
<rx_component_name2>_exdes <component_name>_exdes_inst (
    .RX_HARD_ERR          (RX_HARD_ERR),
    .RX_SOFT_ERR          (RX_SOFT_ERR),
    .UFC_ERR              (UFC_ERR),
    .USER_K_ERR           (USER_K_ERR),
    .RX_LANE_UP           (RX_LANE_UP),
    .RX_CHANNEL_UP        (RX_CHANNEL_UP),
    .INIT_CLK_P           (tied_to_ground_i),
    .INIT_CLK_N           (tied_to_ground_i),
    .PMA_INIT              (PMA_INIT),
    .GTXQ0_P              (tied_to_ground_i),
    .GTXQ0_N              (tied_to_ground_i),
    .DATA_ERR_COUNT       (DATA_ERR_COUNT),
    .RXP                   (),
    .RXN                   (),
    .CRC_PASS_FAIL_N      (CRC_PASS_FAIL_N),
    .CRC_VALID            (CRC_VALID),
    .RESET                 (RX_RESET),
    .SHARED_LOGIC_MASTER_TO_SLAVE (shared_logic_master_to_slave),
    .GT_OUT_SMPX_TX_OUT_RX_IN (GT_OUT_SMPX_RX_IN_TX_OUT),
    .GT_IN_SMPX_TX_IN_RX_OUT (GT_IN_SMPX_RX_OUT_TX_IN)
);
```

変更内容 3

次に示すように、SHARED_LOGIC_MASTER_TO_SLAVE バスを利用してシンプレックス TX からの共有ロジック信号をシンプレックス RX に assign 文で接続します。必要な変更内容を、次のコードに示します。

```
assign shared_logic_master_to_slave[0] = user_clk_i;
assign shared_logic_master_to_slave[1] = sync_clk_i;
assign shared_logic_master_to_slave[2] = INIT_CLK_i;
assign shared_logic_master_to_slave[3] = refclk1_in_i;
assign shared_logic_master_to_slave[4] = gt_qpllclk_quad1_i;
assign shared_logic_master_to_slave[5] = gt_qpllrefclk_quad1_i;
assign shared_logic_master_to_slave[6] = gt_qplllock_i;
assign shared_logic_master_to_slave[7] = pll_not_locked_i;
```

変更内容 4

新規に定義したポートを <component_name> モジュールのインスタンス化の各バス信号に接続します。必要な変更内容を、次のコードに示します。

```
<tx_component_name2> <tx_component_name2>_block_i
(
    .....
    .GT_OUT_SMPX_RX_IN_TX_OUT (GT_OUT_SMPX_RX_IN_TX_OUT),
    .GT_IN_SMPX_RX_OUT_TX_IN (GT_IN_SMPX_RX_OUT_TX_IN),
    .gt_qplllock_out          (gt_qplllock_i),
    .gt_qpllrefclk_quad1_out (gt_qpllrefclk_quad1_i),
    .gt_qpllclk_quad1_out     (gt_qpllclk_quad1_i),
    .gt_refclk1_out           (refclk1_in_i),
    .reset_pb                 (TX_RESET),
    .RX1N_IN                  (RX1N_IN),
    .RX1P_IN                  (RX1P_IN),
    .RX1N_IN_LANE1            (RX1N_IN_LANE1),
    .RX1P_IN_LANE1            (RX1P_IN_LANE1),
    .....
);
```

設計に関する考察事項

このデザインをシステムで使用する際は、次の点に注意してください。

- Aurora シンプレックス TX およびシンプレックス RX コアへの RESET 入力はデザイン内でデカップルされます。ただし、PMA_INIT 入力はデカップルされません。
- このアプリケーション ノートで説明した方法は、すべてのシンプレックス コアのライン レートが同じであることを前提としています。

リファレンス デザイン

このリファレンス デザインは Vivado Design Suite 2014.3 を使用して作成し、2 つの VC7203 ボードを接続してテストしています。このサンプル デザインは、Aurora 64B/66B 内蔵のリセット FSM を使用して反復的リセットによってリンク機能をテストしています。リセットを 4000 回実行してリンクの機能をテストし、すべてのリセット後にリンクが正しく動作することを確認しています。次の図は、4000 回のリセット実行後の結果を示したものです。ここで説明した手順は、一部を変更してほかのバージョンのコアにも適用できます。

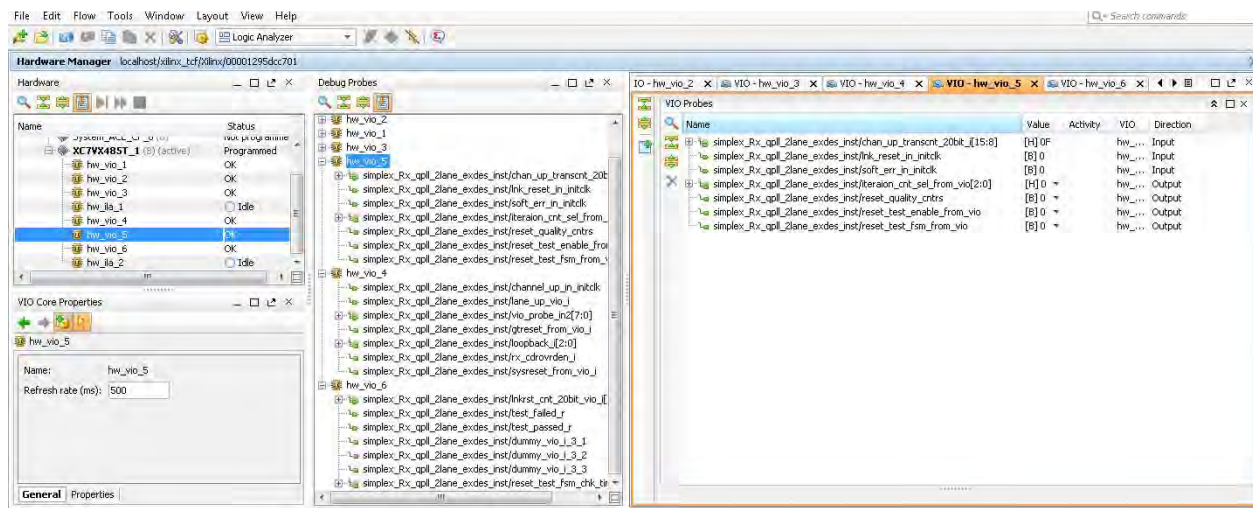


図 18 : 4000 回リセット後のチャネル アップ回数

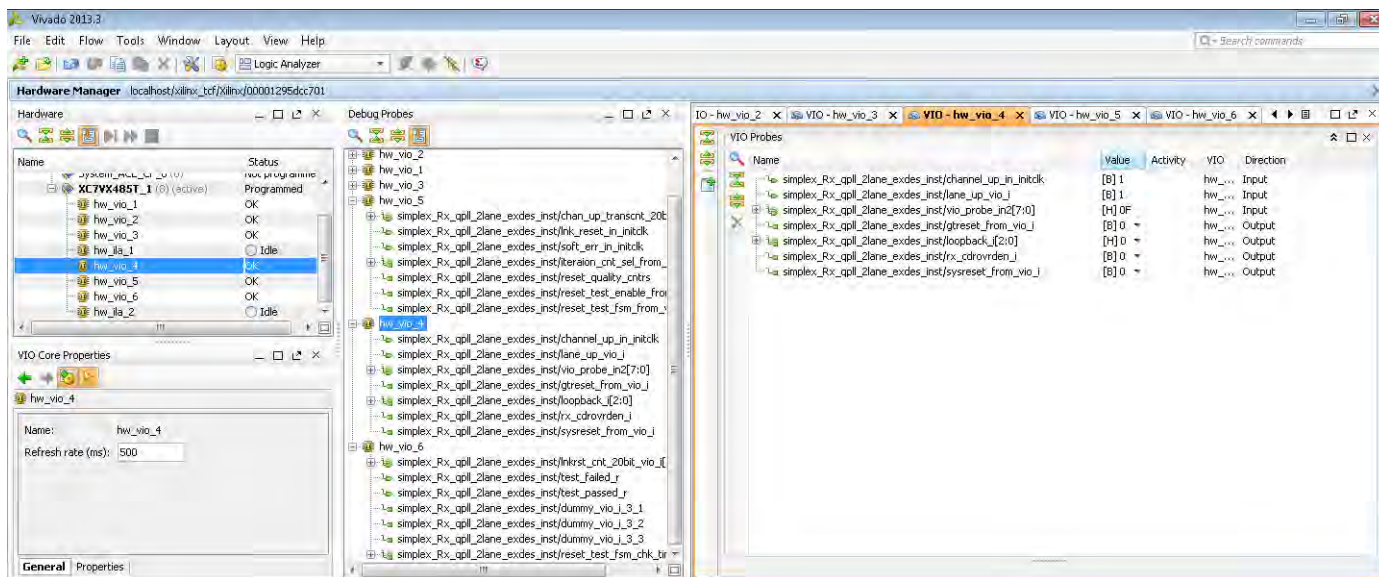


図 19: 4000 回リセット後のチャンネルアップ ステータス

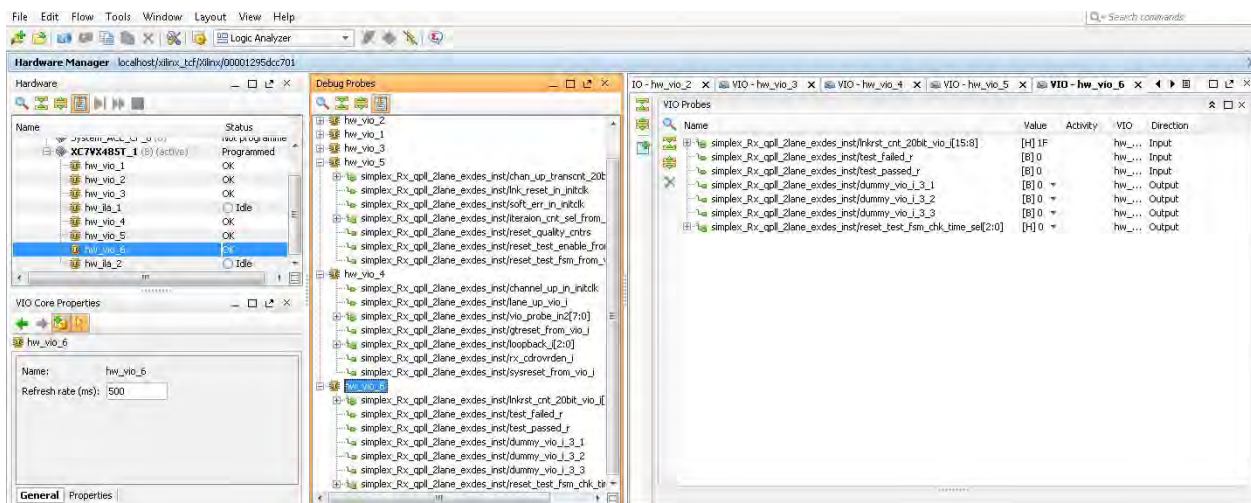


図 20: 4000 回リセット後のリンクリセット回数

このアプリケーション ノートの [リファレンス デザイン ファイル](#) は、ザイリンクスのウェブサイトからダウンロードできます。

表 1 に、リファレンス デザインの詳細を示します。

表 1: リファレンス デザインの詳細

パラメーター	説明
全般	
開発者	Venkata Srinadh Utukuru, Sai Krishna Marella
ターゲット デバイス	XC7VX485TFFG1761-2
ソース コードの提供	あり
ソース コードの形式	Verilog
既存のザイリンクス アプリケーション ノート/リファレンス デザイン、サードパーティ、Vivado ツールからデザインへのコード/IP の使用 (使用した場合はその詳細)	(4 レーン シンプレックス TX + 2 レーン シンプレックス RX)、(4 レーン シンプレックス RX + 2 レーン シンプレックス TX) となるようにカスタマイズした Aurora 64B/66B シンプレックス コア
シミュレーション	
論理シミュレーションの実施	あり
タイミングシミュレーションの実施	なし
論理シミュレーションおよびタイミングシミュレーションでのテストベンチの利用	あり
テストベンチの形式	Verilog
使用したシミュレータ/バージョン	Questa SIM 10.2a
SPICE/IBIS シミュレーションの実施	N/A
インプリメンテーション	
使用した合成ツール/バージョン	Vivado 合成
使用したインプリメンテーション ツール/バージョン	Vivado Design Suite 2014.3
スタティック タイミング解析の実施	なし
ハードウェア検証	
ハードウェア検証の実施	あり
使用したハードウェア プラットフォーム	VC7203 特性評価ボード

参考資料

- 『LogiCORE IP Aurora 64B/66B 製品ガイド』([PG074](#))
- 『KC705 評価キットで Aurora 64B66B コア (全二重) を使用するシステムを設計』([XAPP1192](#))
- 『7 シリーズ FPGA GTX/GTH トランシーバー ユーザー ガイド』([UG476](#))
- 『Vivado Design Suite ユーザー ガイド : IP を使用した設計』([UG896](#))
- 『Vivado Design Suite ユーザー ガイド : プログラムおよびデバッグ』([UG908](#))
- 『VC7203 IBERT スタートアップ ガイド』([UG847](#))

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	内容
2014年11月12日	1.0	初版

法的通知

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>.

Automotive Applications Disclaimer

XILINX PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS APPLICATIONS RELATED TO: (I) THE DEPLOYMENT OF AIRBAGS, (II) CONTROL OF A VEHICLE, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE (WHICH DOES NOT INCLUDE USE OF SOFTWARE IN THE XILINX DEVICE TO IMPLEMENT THE REDUNDANCY) AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR, OR (III) USES THAT COULD LEAD TO DEATH OR PERSONAL INJURY. CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF XILINX PRODUCTS IN SUCH APPLICATIONS.

© Copyright 2014 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

この資料に関するフィードバックおよびリンクなどの問題につきましては、jpn_trans_feedback@xilinx.com まで、または各ページの右下にある [フィードバック送信] ボタンをクリックすると表示されるフォームからお知らせください。いただきましたご意見を参考に早急に対応させていただきます。なお、このメールアドレスへのお問い合わせは受け付けておりません。あらかじめご了承ください。