XAPP1236 (v1.0) 2015 年 6 月 15 日

Vivado 高位合成ツールを使用したマルチチャネル分数比サンプル レート変換フィルター デザイン

著者 : Matt Ruan

概要

このアプリケーション ノートでは、Vivado 高位合成 (HLS) ツールを使用したマルチチャネル分数比サンプル レート変換 (SRC) フィルター デザインについて説明します。このツールは、C++ プログラミング言語をソース コードとして使用し、FPGA 向けの高効率で合成可能な Verilog または VHDL コードを生成します。フィルターのパラメーター値 (チャネル数、フィルター タップ数、サンプル レート変換比など) を変更する場合は、C++ ヘッダー ファイルにわずかな変更を加えるだけです。サンプルの SRC フィルター デザインは一般的なアーキテクチャを使用しているため、C++ ソース コードを変更して別のフィルター タイプを簡単に実現できます。

はじめに

サンプル レート変換 (SRC) フィルターは、複数のデータ レートに対応する必要があるデジタル信号処理システムで幅広く採用されています。たとえば、音楽用コンパクト ディスク (CD) では、毎秒 44.1K サウンド サンプルで記録されますが、デジタル ビデオ ディスク (DVD) のサウンド トラックは、毎秒 48K サンプルで再生する必要があります。CD 音楽を DVD サウンド ドラックに追加する前には、ビデオ編集ツールで 44.1Ksps から 48Ksps へサンプル レートを変換する必要があります。

SRC フィルターのもう 1 つの重要な役割は、波形を描写するのに必要なサンプル数を抑えるデータ圧縮機能です。3GPP LTE (Long-Term Evolution) システム [参照 1] の場合、LTE 20MHz 信号の公称サンプル レートは 30.72Msps ですが、有効な信号帯域幅はわずか 18.015MHz です。つまり、サンプル レートを削減することで、ベースバンド チャネル カードとリモートの無線ユニット間のトランスポート帯域幅を最大 30% ～ 40% 削減できます。その結果、オペレーターは既存の光ファイバーを使用して、今までより 30% ～ 40% 多くのユーザーにコンテンツを配信できるようになります。

SRC フィルターが多用途に適用されるようになり、プログラマブル ロジック デバイスで動作が可能で、柔軟性と拡張性があり、リソース効率の良いフィルターを設計する方法が求められています。このアプリケーション ノートでは、この要求に応えることができるザイリンクス Vivado Design Suite を使用して、C++ プログラミング言語でフィルターを定義し、HDL (ハードウェア記述言語) に合成して FPGA に実装する方法を説明します。Vivado Design Suite に統合されている Vivado HLS (HLS) ツール [参照 2] では、与えられた制約に基づいて最適化されたパイプライン アーキテクチャで HDL が自動生成され、HDL コードと C++ コードのビヘイビアーが同じであることを検証するためのテストベンチが作成されます。通常、Vivado HLS で合成されたコードは、経験豊富なロジック エンジニアが手作業で記述する HDL デザインと同等の性能を持ちます。クロック レート、ターゲット FPGA デバイス番号、またはフィルター パラメーター (タップ値、分数変換比など) の変更が必要な場合は、C++ ヘッダー ファイルにわずかな変更を加えるだけです。C++ で記述されたフィルター デザインは IP としてパッケージ化され、新しいアプリケーション用に容易にカスタマイズできます。

動作理論

このセクションでは、サンプル レート変換の基本的な理論と SRC フィルター機能の動作について説明します。フィルターの入力データ サンプル レートと出力データ サンプル レートの比率は P/Q (P と Q は整数) で表します。サンプル レート変換動作は、次のステップで実行されます。

1. 補間 : 2 つの入力サンプルごとに (Q-1) 個の 0 を挿入します。
2. ローパス フィルター処理 : ローパス フィルターを使用して、エイリアシングを削除します。
3. 間引き : すべての P 個のローパス フィルター処理したサンプルに対して 1 つのサンプルを出力します。

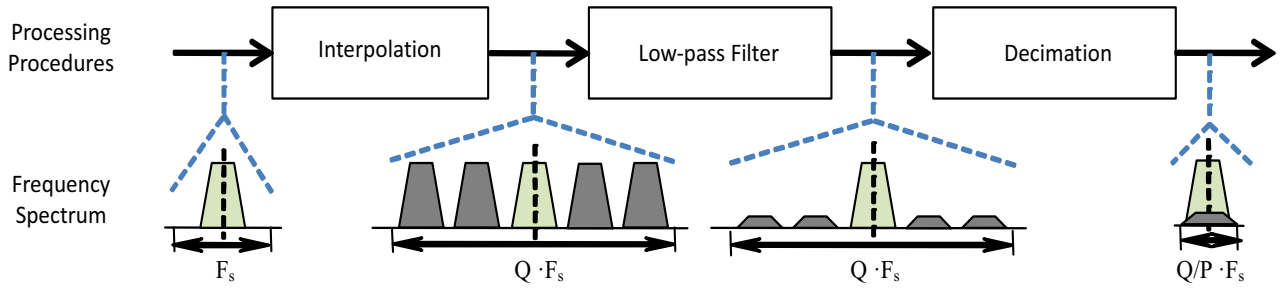


図 1 : 分数サンプル レート変換の概念的な手順

図 1 に、各手順後の信号のスペクトルを示します。必要な信号とそれらの不要なエイリアシングをそれぞれ黄緑色と濃いグレーで示しています。SRC フィルタリングによる最終的な影響は次のようになります。まず、信号の全スペクトルが圧縮または拡大されます ($Q/P \cdot F_s$)。このとき、オリジナルの信号がエイリアシングによってわずかに劣化するため、2 つ目の手順で設計されたローパス フィルターを使用して、信号を許容範囲内に制御する必要があります。

この動作について詳しく説明するため、図 2 に SRC フィルターの計算手順を示します。1 行目のボックスは、0 (白色) が挿入された入力データ ストリームを表しています。この下の行にあるボックスは、畳み込み乗算を実行するために左から右へとスライドするフィルター係数を表しています。これらの各行は 1 つのフィルター係数の位置に対応し、ローパス フィルターにかけられた出力が 1 つ計算されます。破線「--」で表された行では、ローパス フィルターにかけられた結果が間引きされ、「y_n」で表された行のみ SRC フィルターの最終出力として選択されます。

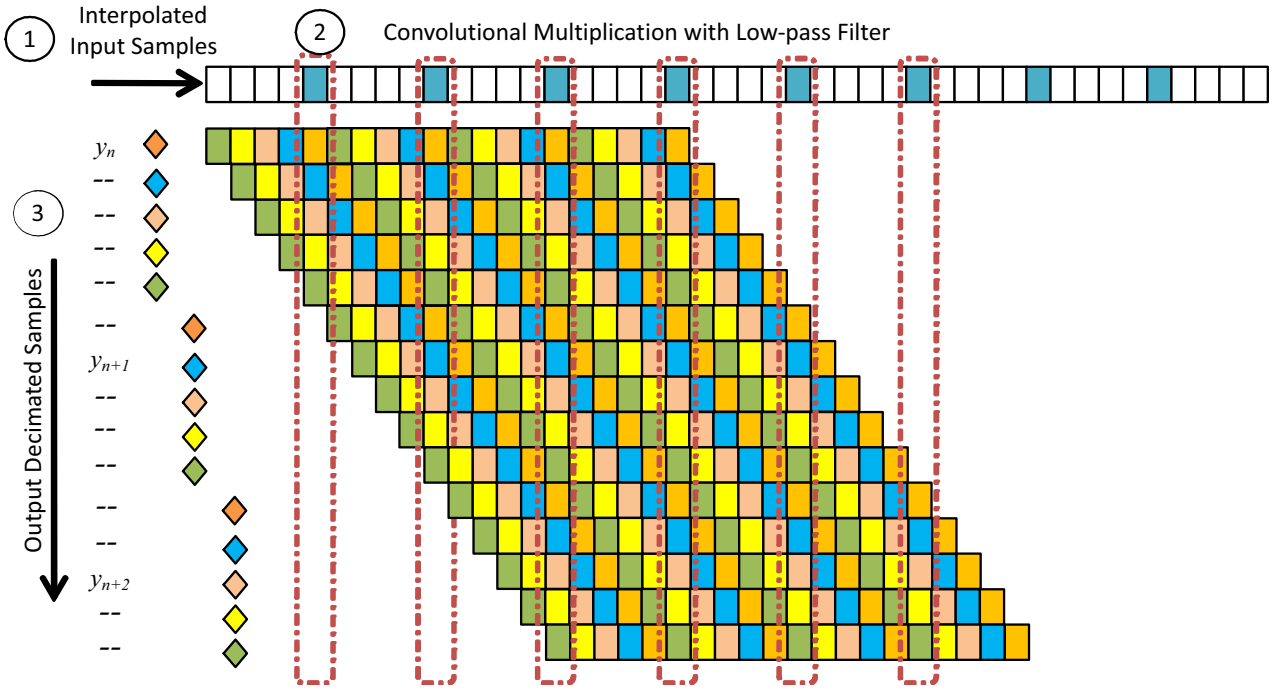


図 2 : 分数サンプル レート変換フィルターの計算手順

図 2 に示す例の SRC フィルターは 20 タップで分数比は 5/6 です。すべてのローパス フィルター出力を計算するには、間近の 4 つの入力サンプルのみ必要です。さらに、係数は、0 以外のサンプルでドット積が求められ、1 つの出力サンプルを計算する場合に異なるセットの係数が使用されないように、5 つのセットに分割できます。これらの観測から次のような事実に法則化できます。

- P 個のフィルター処理されたサンプルから 1 つのみ出力されるため、その他の (P-1) 個のローパス フィルター処理されたサンプルの計算は不要です。
- フィルター タップ数は L で表されます。フィルター係数は Q 個の位相に均等に分割され、各位相には 1 つのローパス フィルター出力の計算に必要となる、最大 ceil(L/Q) 係数が含まれます。
- 1 つのローパス フィルター出力を計算する際は、1 つの位相の係数を使用するドット積用に、直近の ceil(L/Q) 入力サンプルをレジスタに保持する必要があります。

これらの事実を考慮して、図 3 に示すような非常にシンプルな SRC フィルター アーキテクチャが完成します。フィルター係数は、ceil(L/Q) ROM に格納され、入力サンプルとのドット積に必要な係数のみ読み出されて 1 つのフィルダー出力が計算されます。図 3 に示すアーキテクチャは 従来型 FIR フィルターと類似していますが、係数 ROM アドレスを生成してシフト レジスタを適切に管理するために、コントローラーを再設計する必要があります。

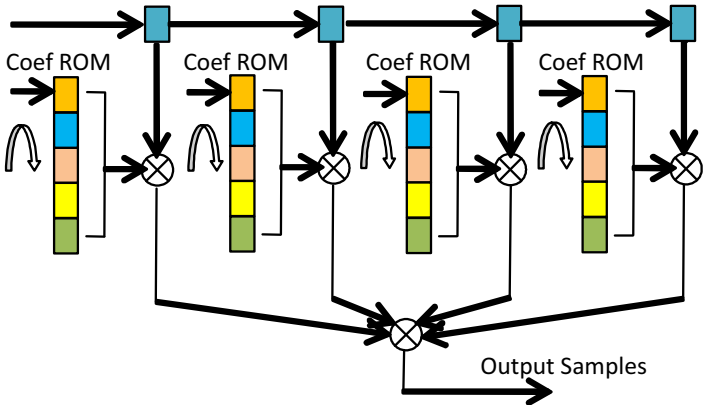


図 3 : ハードウェアの利用効率が良い分数サンプル レート変換フィルターのアーキテクチャ

マルチチャネル SRC フィルターのアーキテクチャ

広帯域のマルチアンテナ LTE システムは、複数のデータ ストリームを処理する必要があり、各データ ストリームには個別に選択されたデータ レートとサンプル レート変換比があります。このアプリケーション ノートでは、通常 200MHz またはそれ以上で動作するザイリンクス 7 シリーズ FPGA にそのようなマルチチャネル/マルチレート SRC フィルターを設計する方法を紹介します。分数リサンプル フィルターの要件を次の表にまとめます。

表 1 : マルチチャネル/マルチレート SRC の要件

	要件	説明
チャネル数	8	8 本のチャネルが 4 つの LTE 20MHz キャリア (80MHz 信号帯域幅) に対応します。
入力サンプル レート	最大 30.72Msps	LTE 20MHz 信号の公称サンプル レート
出力サンプル レート	最大 30.72Msps	LTE 20MHz 信号の標準的なサンプル レート
サンプル変換比	バイパス、3/4、5/8、5/6、4/3、8/5、6/5	3/4、5/8、5/6 は間引き比、4/3、8/5、6/5 は補間比です。1 つのフィルターですべての比率に対応します。

このようなマルチチャネル FIR には、図 4 に示すようなハードウェアの利用効率が良いシストリック マルチ MAC アーキテクチャの使用を推奨します [参照 3]。FPGA 内の豊富なルックアップ テーブル (LUT) を使用して実装できるシフト レジスタに入力データ ストリームを格納することで、高いリソース効率を達成できます。1 つの DSP48E ユニットで各タップの積加算を実現できるように、各 MAC の出力に 1 段のパイプラインが挿入されています。これにより、多くの FPGA リソースを節約でき、タイミングが向上します。

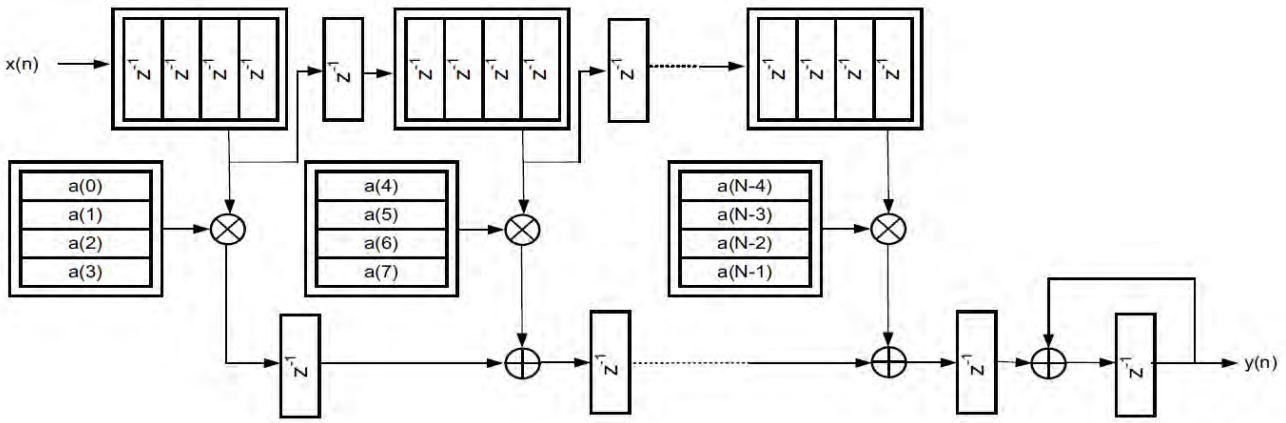


図 4 : 推奨されるシストリック マルチ MAC アーキテクチャ ([?? 3] の図 3-10)

図 3 に示すように、SRC フィルターの中心的な計算モジュールはシストリック MAC ブロックであるため、一般的なシストリック マルチ MAC アーキテクチャを変更することで SRC フィルターを含むさまざまなアプリケーションに対応できます。SRC フィルターの注意すべき部分は常に、係数 ROM のアドレス生成やデータ シフト レジスタの管理を行う制御ロジックです。

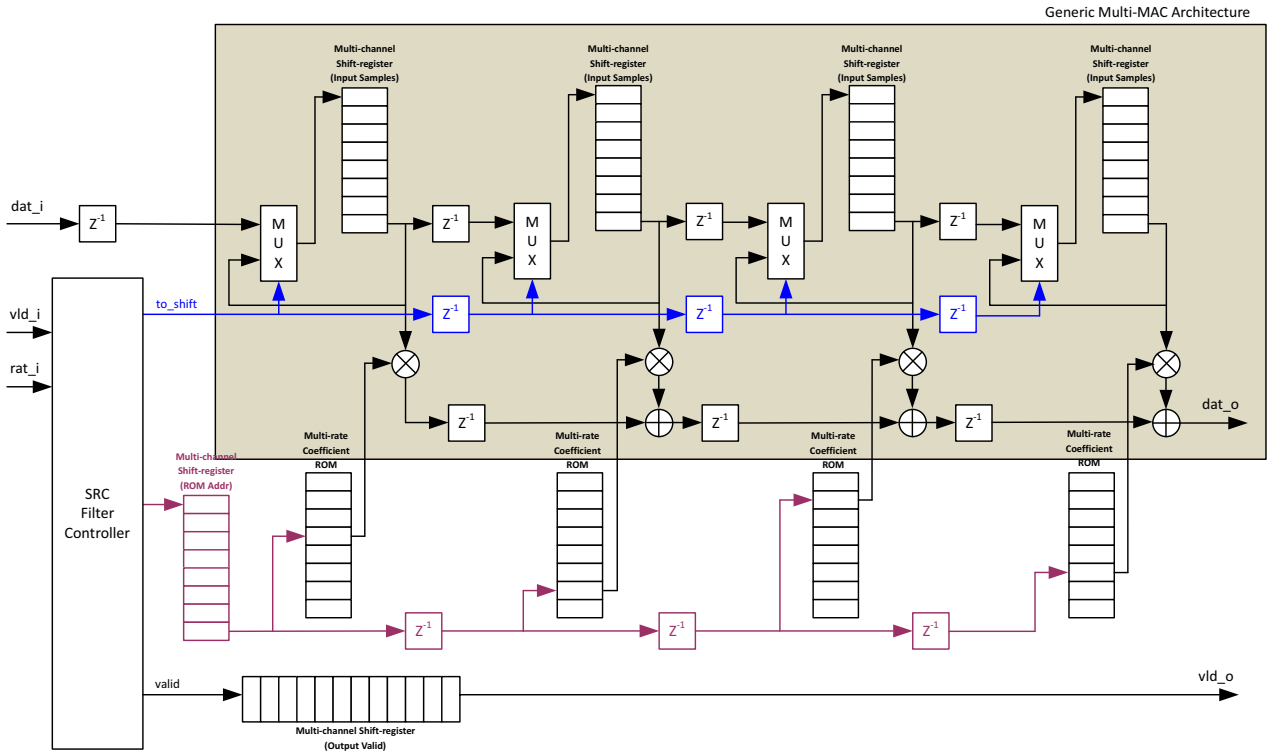


図 5 : 8 チャネル 4 タップのマルチレート SRC フィルターの簡略ブロック図

図 5 に、マルチチャネル/マルチレート SRC フィルターのブロック図を示します。影付き部分は、ラウンドロビンの時分割方式で処理される独立チャネルのデータを格納するシフト レジスタを備えた一般的なマルチ MAC 演算ユニットを示しています。valid_i がアサートされると、シフト レジスタが新しい入力データを取り入れて、古いデータをシフト アウトします。現チャネルに新しい入力データがない場合は、チャネルのステートが変わらないように、シフト レジスタの最も古いデータが書き戻されます。これにより、各チャネルが F_c/N_{ch} で与えられた上限 (F_c はメインのクロック周波数、N_{ch} はチャネル数) を超えない限り、SRC フィルターは多様なサンプル レートに対応できます。

係数 ROM の構造を図 6 に示します。各アドレスは、与えられたサンプル変換比の係数セットに対応します。すべての係数 ROM には、1 つのフィルター タップの全係数が含まれます。アーキテクチャがパイプライン化されているため、最初の ROM アドレスのみ計算されて、それがその他のフィルター タップに対して遅延されます。

Addr	ROM 0	ROM 1	ROM 2	ROM M
0	Rat1_Coef_0	Rat1_Coef_3	Rat1_Coef_6	Rat1_Coef_L-2
1	Rat1_Coef_1	Rat1_Coef_4	Rat1_Coef_7	Rat1_Coef_L-1
2	Rat1_Coef_2	Rat1_Coef_5	Rat1_Coef_8	Rat1_Coef_L
3	Rat2_Coef_0	Rat2_Coef_5	Rat2_Coef_10	Rat2_Coef_L-4
4	Rat2_Coef_1	Rat2_Coef_6	Rat2_Coef_11	Rat2_Coef_L-3
5	Rat2_Coef_2	Rat2_Coef_7	Rat2_Coef_12	Rat2_Coef_L-2
6	Rat2_Coef_3	Rat2_Coef_8	Rat2_Coef_13	Rat2_Coef_L-1
7	Rat2_Coef_4	Rat2_Coef_9	Rat2_Coef_14	Rat2_Coef_L

図 6 : 係数 ROM データ構造の図

図 5 に示すように、SRC フィルター コントローラーで入力データ (to_shift フラグ、valid フラグ、ROM アドレス) が計算された後は非常に単純で、シンプルな演算ブロックで実行できます。ただし、コントローラーが複数チャネルのサンプル レートおよび変換比に対応できるロジックを設計する作業は単純ではありません。このようなコントローラーの記述には、C++ などの高レベル プログラミング言語が有効です。

インプリメンテーションの詳細

図 7 に、C++ 関数のデータ フローを示します。最上位の関数は MultiSRC で、その中の 3 つのサブ関数 (srcCtrl、srcMac、srcCoef) は図 5のコントローラー、MAC 演算ユニット、および係数 ROM に相当します。シフト レジスタは、HLS のデータ構造 ap_shift_reg で実現されます [参照 2]。srcMac 関数および srcCtrl 関数の C++ ソース コード、および必要な合成結果を得るために使用する Vivado HLS の指示子については、次のセクションで説明します。

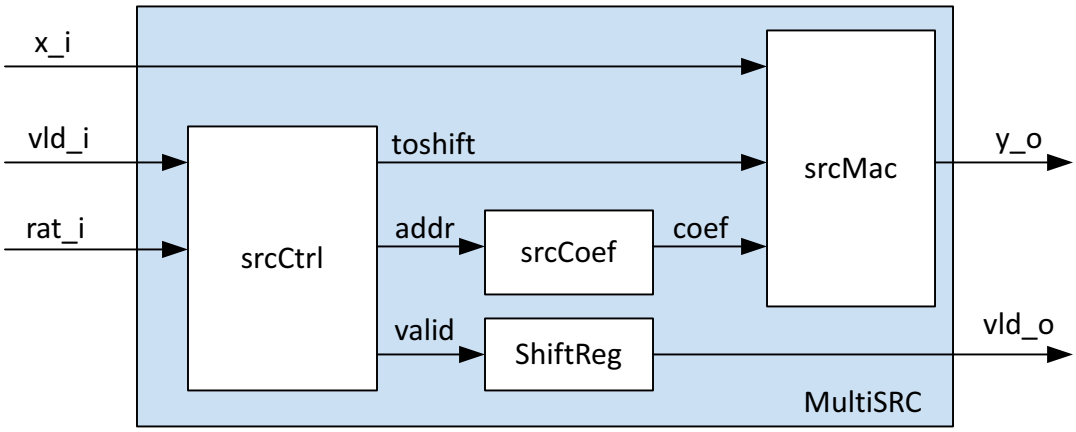


図 7 : C++ 関数のデータ フロー

srcMac.cpp

この関数は、一般的なマルチチャネル シストリック MAC ユニットを実装します。図 8 に、複数チャネルをサポートする 1 つのフィルター タップの MAC 演算ブロックを示します。このような MAC モジュールをいくつかインスタンシエートして、1 つのマルチタップ フィルターを生成できます。フィルター係数が MAC の入力として提供されることが前提で、一般的な MAC アーキテクチャを使用してあらゆる種類のフィルター (SRC フィルターに限定されない) に対応できます。

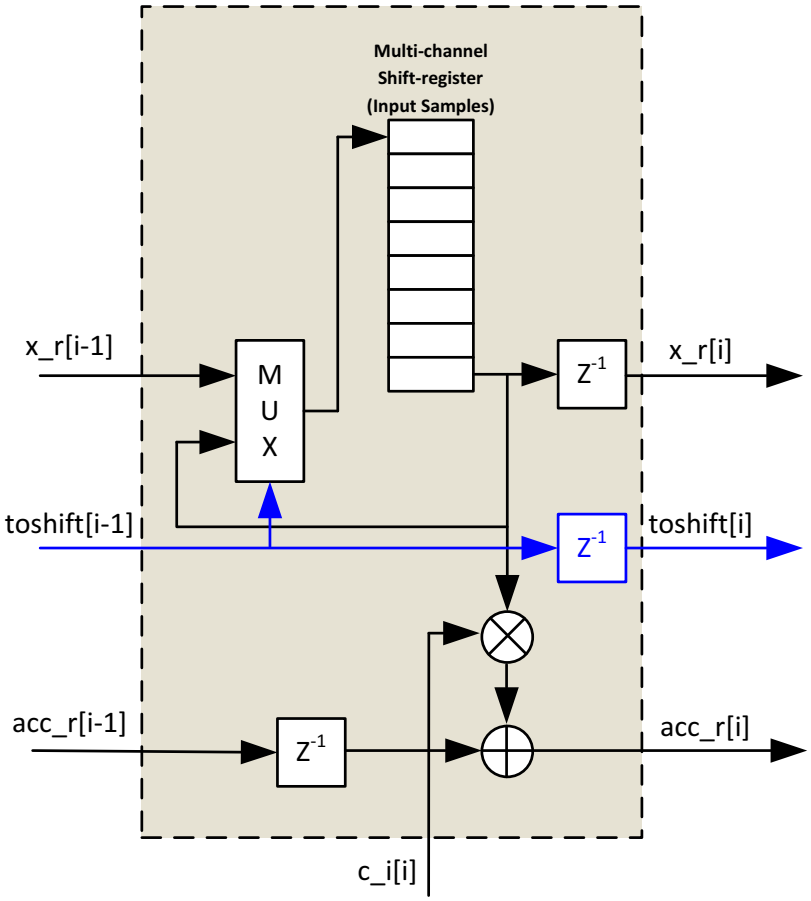


図 8 : 1 つのフィルター タップの MAC ブロック

マルチチャネル MAC は、次の C++ コードで定義されます。

```
// loop for all the taps
MULTMACLOOP: for(int i=NumTap-1; i>=1; i--){

  // multiplier with registered output
  mult_t mul_temp = x_r[i] * c_i[i];

  // acc_o is expected to have one clock delay here
  acc_r[i] = mul_temp + acc_r[i-1];

  // read out data from shift register
  x_r[i] = shift_reg[i].read(ChMux-1);

  // mux to determine whether shift or not
  // if not shift, then write back the data read from shift register
  x_mux = toshift_r[i-1]? x_r[i-1] : x_r[i];

  // shift in and out
  shift_reg[i].shift(x_mux);
  toshift_r[i]=toshift_r[i-1];
```

```
}

// multiply for the first shift reg
acc_r[0] = x_r[0] * c_i[0];

// read out data for next
x_r[0] = shift_reg[0].read(ChMux-1);


// mux to determine whether shift or not
// if not shift, then write back the data read from shift register
toshift_r[0]=toshift_i;
x_mux = toshift_r[0]? x_i : x_r[0];
shift_reg[0].shift(x_mux);
```

srcCtrl.cpp

この関数は、マルチチャネル/マルチレート SRC フィルターの制御ロジックを実装します。図 9 にブロック図を示します。チャネルが機能するとき、そのステータスが読み出されて、サンプル レート変換比、データ有効フラグなどの入力パラメーターに基づいて処理されます。新しい位相や係数 ROM アドレスの計算が完了次第、次にアクセスされる前に、そのチャネルのステータス メモリがアップデートされる必要があります。

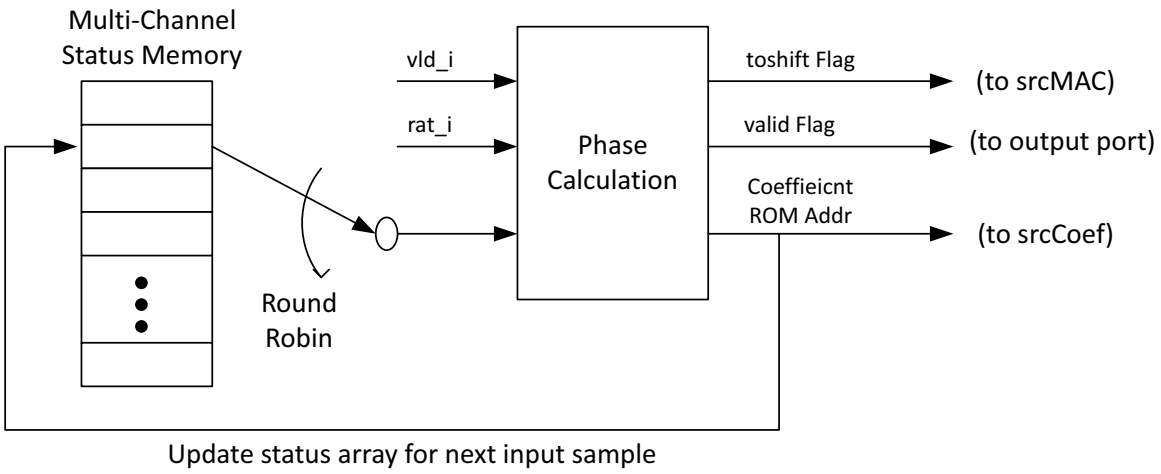
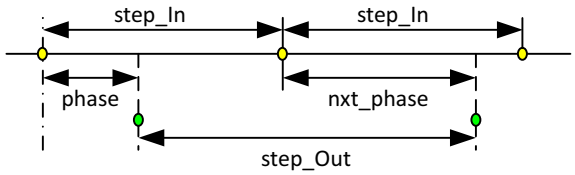
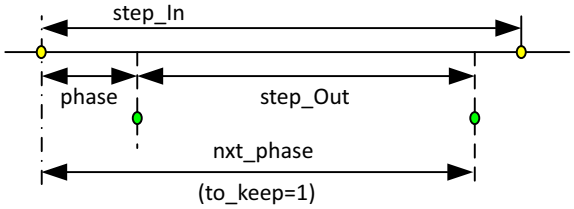


図 9 : 1 つの位相計算エンジンが時分割方式で複数チャネルに対応

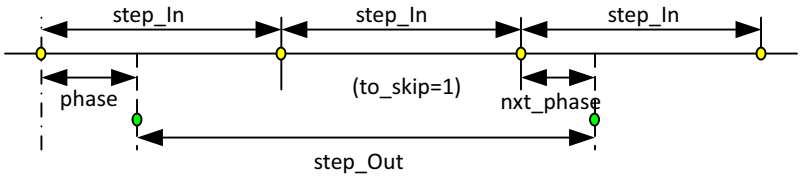
図 10 に、位相計算のプロセスを示します。入力サンプル周期は step_in で定義され、出力サンプル周期は step_out で定義されます。黄色のドットは入力サンプルを表し、緑色のドットは出力サンプルを表しています。位相とステップは、現入力サンプルの位置に対して定義されます。図 10 (a) に、次の出力が次の 2 つの入力の間に生じる通常の位相アップデート プロセスを示します。図 10 (b) のように、次の入力が生じる前に次の出力がまだ生じている場合は、to_keep フラグをアサートして、MAC のシフト レジスタに現在のステータスを保持するように強制する必要があります。間引きの場合、図 10 (c) のように次の出力が次の 2 つの入力サンプルの後に生じるため、to_skip フラグをアサートして、次の入力サンプルに対して出力を計算する必要がないことを示す必要があります。



(a) 通常条件、to_keep=false、to_skip=false



(b) (phase+step_Out<step_In) の場合、to_keep=true、to_skip=false



(c) (phase+step_Out>2 step_In) の場合、to_keep=false、to_skip=true

図 10 : 位相計算およびフラグ設定の条件

上記の制御ロジックは、次の C コードで定義できます。

```
// update phase and flag for next input
   if (vld_o){
   phase+=step_out;
   if(phase<step_in){ // it means that no new data is required for next output
     tokeep=true;
     isskip=false;
   }else{ // new data is needed
     tokeep=false;
     isskip=( (phase>>1) >=step_in);
     phase-=step_in;
   }
  }else if (vld_i){ // a new input is available, but it is to be skipped
    tokeep=false;
    isskip=( (phase>>1) >=step_in);
    phase-=step_in;
  }
```

指示子

C++ コードを HDL コードに変換する際、Vivado HLS は、1 つのループを完了するために必要なクロック サイクル数やモジュールは古い入力がすべて処理される前に新しい入力を受け入れることができるかなどのパラメーターを記述した追加情報を必要とします。これらの指示子はデザインに欠かせない情報であり、C++ コードが適切なビヘイビアーで HDL に合成されるように指示します。既存のデザインを新しいアプリケーションに移植する場合、通常は指示子をわずかに変更するだけでよく、C++ コードを変更する必要はありません。

SRC フィルターの場合、次の指示子を使用して必要なビヘイビアーを実現します。

```
# The function takes pipelined architecture and accepts new inputs every clock cycle
set_directive_pipeline -II 1  multiSRC

# The functions can be inlined to avoid hand shaking
set_directive_inline -region -recursive multiSRC

# The loops in Multi-MAC calculator should be unrolled and run in parallel
set_directive_unroll -skip_exit_check   srcMac/MULTMACLOOP

# The 2D array of filter coefficients should be partitioned into small ROMs for each tap
set_directive_array_partition -dim 1 srcCoef coef_rom
```

合成結果

ザイリンクスの Vivado HLS は、すべてのデザイン ファイルを解析し、その後、設計者が合成の指示子で指定したターゲット クロック スピードやデータ スループットを満たす適切なハードウェア アーキテクチャを自動的に選択します。C コンパイル完了後には、合成された HDL の基本情報を参照して、デザイン目標と照らし合わせて結果を評価できます。図 11 に、Vivado HLS で生成された SRC フィルターの合成レポートを示します。

想定どおり、47 タップのマルチチャネル SRC フィルターは 47 個の DSP48E を使用して、各入力データあたり 1 クロック サイクルのスループットを達成しています。デザインの予想動作は 1/3.55ns、つまり 281.6MHz であるため、250MHz の要件を満たしています。レポートによると、フィルター係数 ROM とベース アドレス テーブルは、分散メモリで実装されています。FPGA の配置配線が完了するまでデザインのタイミングは固定されないため、C 合成レポートにあるクロック スピードの予測値は、最終的な結果とは異なる可能性があります。

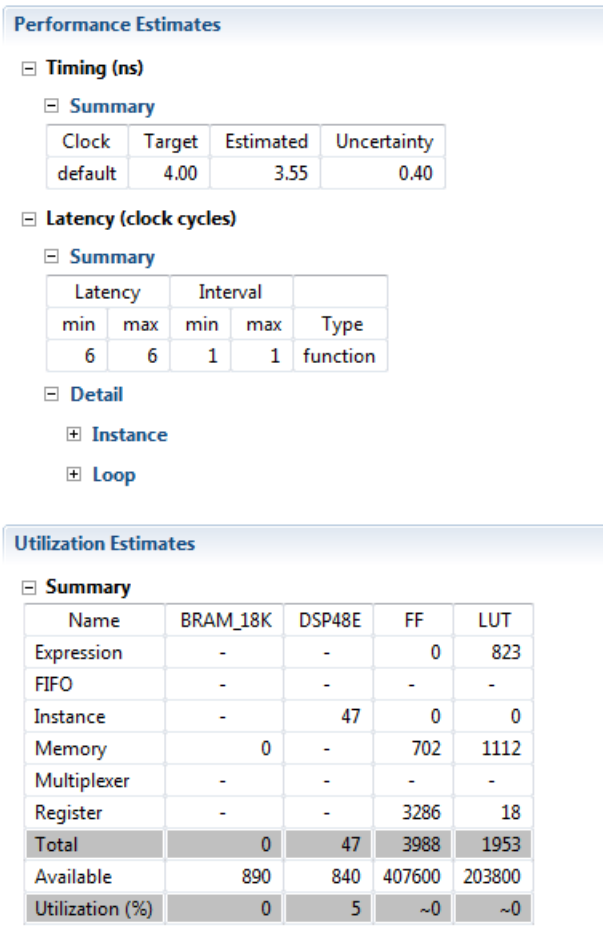


図 11 : C 合成レポート

検証結果

Vivado HLS デザイン フローの機能検証は、2 つの手順で実行します。

最初の手順は、C++ コードを検証するための C++ の機能検証です。テストベンチを C++ コードでマニュアル記述する必要があります。ただし、C++ ライブラリで提供されるリッチ ファイル形式の I/O 関数を使用する場合は、あらかじめ保存されている入力および出力テスト ベクターに基づいてテストベンチを記述することは難しくありません。マルチチャネル SRC の場合、テストベンチが入力テスト ベクターを読み込み、C++ 関数を呼び出してデータを処理します。その後、C++ 関数の出力をあらかじめ保存されているゴールデン テスト ベクターの出力と比較します。検証用に、異なるサンプル レートと変換比を持つ 8 つの独立したチャネルを備えた 1 つのテスト ケースが構築されています (表 2 参照)。

表 2 : 検証用に構築された SRC フィルターのテスト ケース

チャネル ID	キャリア サンプル レート (Msps)	変換比
0	30.72	5/8
1	23.04	5/6
2	15.36	3/4
3	5.76	バイパス
4	15.36	4/3
5	7.68	8/5

表 2 : 検証用に構築された SRC フィルターのテスト ケース (続き)

チャネル ID	キャリア サンプル レート (Msps)	変換比
6	11.52	6/5
7	19.2	バイパス

C++ ビヘイビアーが検証されると、C++ 関数が HDL に合成されて、Vivado HLS は C++ テスト コードに従って自動的に HDL テストベンチを生成できるようになります。この手順を「C/RTL 協調シミュレーション」と呼び、HDL のビヘイビアーと C++ の機能が一致することを確認します。図 12 に示すように、Vivado HLS は C/RTL 協調シミュレーションに複数のシミュレータおよび HDL コードをサポートしています。

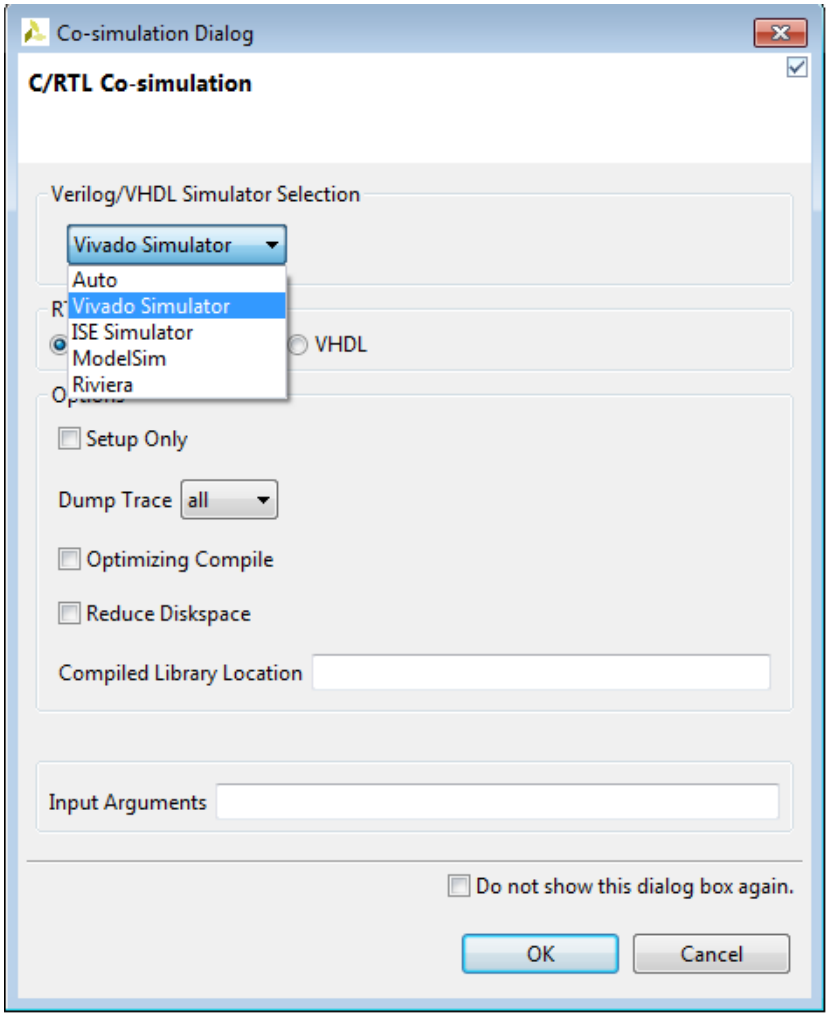


図 12 : C/HDL 協調シミュレーションのウィザード

HDL デザインの出力と C リファレンス モデルの出力を比較して、正しく機能していることを確認します。シミュレーション終了後、ツールは次のような検証結果を出力します。

```
......

## save_wave_config multiSRC.wcfg
## run all
$finish called at time : 33650400 ps : File
"C:/0510/ProjMultiSRC/SolutionX/sim/verilog/multiSRC.autotb.v" Line 443
run: Time (s): cpu = 00:00:06 ; elapsed = 00:00:06 .Memory (MB): peak = 85.535 ; gain =
0.012
```

```
## quit
INFO: [Common 17-206] Exiting xsim at Sun May 10 21:53:06 2015...
@I [SIM-316] Starting C post checking ...

[0] Total 625 Output Samples 0 Errors.
[1] Total 625 Output Samples 0 Errors.
[2] Total 375 Output Samples 0 Errors.
[3] Total 188 Output Samples 0 Errors.
[4] Total 667 Output Samples 0 Errors.
[5] Total 400 Output Samples 0 Errors.
[6] Total 450 Output Samples 0 Errors.
[7] Total 625 Output Samples 0 Errors.
@I [SIM-1000] *** C/RTL co-simulation finished: PASS ***
```

波形での手動デバッグのために、信号トレースをダンプするオプションもあります。次に示すシミュレーションの波形は、さまざまなサンプル レートと変換比を使用する 8 チャネルのテスト ケースです。使用されたシミュレータは、Vivado シミュレータ 2015.1 です。図 13 に示すこれらの波形から、マルチチャネル SRC のレイテンシが 244ns/4ns = 61 クロック サイクルであり、想定値と一致することが確認できます。

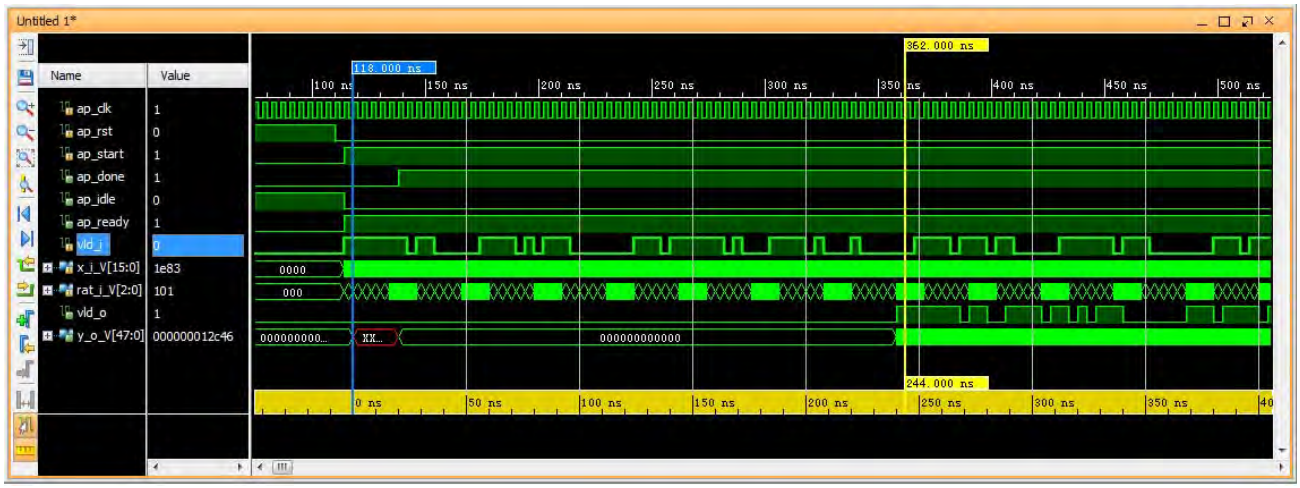


図 13 : C/HDL 協調シミュレーションの波形

インプリメンテーション結果

ザイリンクスの Vivado HLS は、C++ 関数の HDL コードを生成するだけではなく、HDL を IP にパッケージ化する多数のオプションを提供するため、Vivado Design Suite (System Generator、EDK、IP インテグレーターなど) を使用して、これを大規模なデザインに容易に統合できます。ここでは例示目的で、サンプルのリファレンス デザイン用に IP Catalog が選択されています。

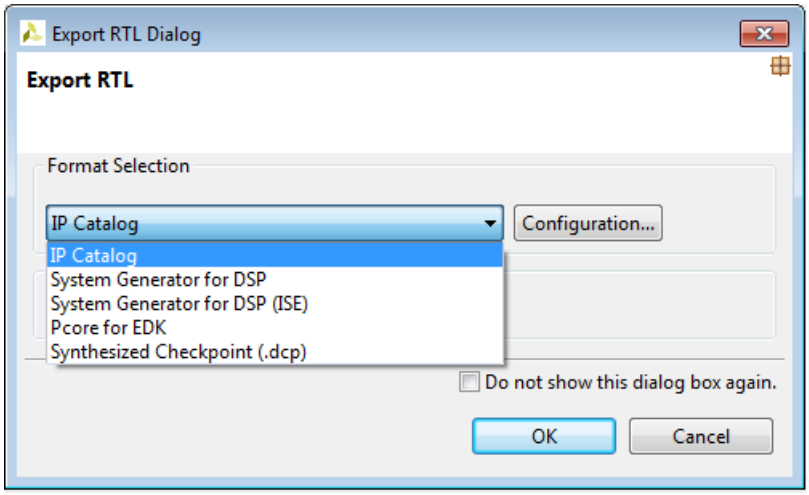


図 14 : [Export RTL] ダイアログ ボックス

Vivado HLS ツールは、Vivado プロジェクトを自動的に作成してすべての HDL コードを合成し、インプリメンテーションの性能を検証します。SRC フィルターの最終的なインプリメンテーション レポートは次のとおりです。

```
Implementation tool: Xilinx Vivado v.2015.1
Device target:       xc7k325tffg900-2
Report date:         Sun May 10 21:59:30 +0800 2015

#=== Resource usage ===
SLICE:          643
LUT:           1542
FF:            2592
DSP:             47
BRAM:             0
SRL:            391
#=== Final timing ===
CP required:    4.000
CP achieved:    3.660
Timing met
```

合成結果の品質を検証するために、ザイリンクスの FIR Complier v7.2 をベースに 2 つのベンチマーク フィルターが構築されています。1 つ目のベンチマークはマルチチャネルの多相 FIR フィルターで、8 つの独立チャネル、32 位相、および各位相で 47 タップをサポートします。この多相フィルターの MAC アーキテクチャは、SRC の MAC アーキテクチャと類似していますが、提示された SRC フィルターのようにリアルタイムにサンプリング レートを変更することをサポートしないため、制御ロジックは非常にシンプルです。表 3 に示すように、HLS で実装された SRC フィルターのリソース使用数は、FIR フィルターのリソース使用数よりわずかに多い程度です。つまり、Vivado HLS 合成の結果は、FPGA リソースという点でかなり効率が良いといえます。

2 つ目のベンチマーク フィルターは、8 つのインスタンスのシングルパス多相 FIR で構成されており、1 つの出力サンプルを計算するために 8 クロック サイクルを使用します。その他の 2 つのソリューションと公正に比較するため、フィルター係数は分散メモリに格納する必要があります。マルチチャネル パイプライン アーキテクチャのメリットを示す表 3 では、Vivado HLS で実装された SRC は LUT の効率が非常に良いことがわかります。つまり、マルチチャネル アーキテクチャは係数 ROM が 1 セットのみ必要であるのに対し、シングルチャネル アーキテクチャは 7 セットも余分に必要になります。同様に、位相の累算では 1 つのフィルターに対して DSP が 1 つ余分に必要になるため、完全なパラレル アーキテクチャよりリソースの使用効率が低下します。

表 3 : ベンチマーク フィルターの合成結果

	LUT	FF	DSP	BRAM18
Vivado HLS で実装された SRC	1542	2592	47	0
マルチチャネル多相 FIR	1283	2043	47	0
8 インスタンスのシングルチャネル多相 FIR	4480	3056	56	0

まとめ

このアプリケーション ノートでは、C++ コードを入力とし、FPGA デザインに合成可能な HDL コードを生成する Vivado HLS ツール チェーンを使用して、マルチキャリア/マルチチャネル SRC フィルターを構築する方法を説明しました。C++ ソース コードは維持が容易な上に、さまざまな FPGA デバイス、出力サンプル レート、システム クロック周波数にスケーラブルに対応可能です。C++ で記述された制御ロジックに変更を加えるだけで、一般的なマルチ MAC アーキテクチャをベースに別の種類のフィルターを簡単に構築できます。

リファレンス デザイン

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 4 に、リファレンス デザインの詳細を示します。

表 4 : 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	Matt Ruan (ザイリンクス)
ターゲット デバイス	7K325T、7Z045 など
ソース コードの提供	あり
ソース コードの形式	C++、テスト ベクター、合成スクリプト
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、またはサードパーティからデザインへのコード/IP の使用	なし
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	C++

表 4 : リファレンス デザインの詳細 (続き)

パラメーター	説明
使用したシミュレータ/バージョン	Vivado シミュレータ 2015.1
SPICE/IBIS シミュレーションの実施	なし
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 HLS 2015.1
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 2015.1
スタティック タイミング解析の実施	あり
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	ザイリンクス ZC706 EVB

デザイン ファイルの階層

最上位フォルダーの下のディレクトリ構造は次のとおりです。

```
\src
| このフォルダーには C++ デザイン ファイルとヘッダー ファイルが含まれます。
|
\tb
| このフォルダーには、テストベンチとして機能する C++ デザイン ファイルが含まれます。
|
\tv
| このフォルダーには、検証目的に使用される入力および出力の
| ゴールデン テスト ベクターが含まれます。
|
\boardtest
|
+----- \src
|   このフォルダーには、オンボード テスト用の Vivado プロジェクト ファイルが含まれます。
```

インストールと操作の手順

サンプル デザインを実行するために FPGA を設定します。

1. ザイリンクスの Vivado 2015.1 またはそれ以降のバージョンをインストールします。
2. デザイン ファイルを空のディレクトリに解凍します。
3. Vivado HLS のコマンド ライン ウィンドウで次を実行します。
   a. `cd` で、デザインのルート ディレクトリに移動します。
   b. 「`vivado_hls run.tcl`」と入力します。
   c. 合成されたデザインが想定を満たしていることを確認します。
4. Vivado の Tcl コマンド ウィンドウで次を実行します。
   a. `cd` で、`boardtest` ディレクトリへ移動します。
   b. `source boardtest.tcl` と入力します。
   c. インプリメンテーション結果が想定を満たしていることを確認します。
5. FPGA 上でリファレンス デザインを実行します。
   a. デザインを ZC706 ボードにダウンロードします。
   b. ZC706 の真ん中のプッシュボタンを押して、デザインをリセットします。

c. 右側のプッシュボタンを押して、テストを実行します。

d. LED2 が点灯していることを確認します。

e. 手順 c と d を数回繰り返して、確実にテストに合格することを確認します。

参考資料

注記：日本語版のバージョンは、英語版より古い場合があります。

1. 3GPP TS 36.211 -「3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation (Release 12)」2014 年 7 月
2. 『Vivado Design Suite ユーザー ガイド：高位合成』(UG902 : 英語版、日本語版)
3. 『FIR Compiler v7.2 LogiCORE IP 製品ガイド』(PG149)

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
2015 年 6 月 15 日	1.0	初版

法的通知

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。フィードバックは日本語で入力可能です。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。

自動車用のアプリケーションの免責条項

ザイリンクスの製品は、フェイルセーフとして設計されたり意図されてはおらず、また、フェイルセーフの動作を要求するアプリケーション (具体的には、(I) エアバッグの展開、(II) 車のコントロール (フェイルセーフまたは余剰性の機能 (余剰性を実行するためのザイリンクスの装置にソフトウェアを使用することは含まれません) および操作者がミスをした際の警告信号がある場合を除きます)、(III) 死亡や身体傷害を導く使用、に関するアプリケーション) を使用するために設計されたり意図されたりもしていません。顧客は、そのようなアプリケーションにザイリンクスの製品を使用する場合のリスクと責任を単独で負います。