WP491 (v1.0) 2017 年 3 月 30 日

浮動小数点から固定小数点への変換による消費電力およびコストの削減

著者: Ambrose Finnerty、Hervé Ratigner

ザイリンクスのデバイスおよびツールは、バイナリから倍精度まで、幅広いデータ型をサポートします。UltraScale™ アーキテクチャはサポートする精度にスケーラビリティがあるため、電力とリソースの使用を非常に柔軟に最適化しながら、デザイン パフォーマンスの目標を満たすことができます。

概要

データ センター、航空、防衛、5G ワイヤレス、およびオートモーティブなどの市場では、ADAS、レーダー、深層学習をはじめとするアプリケーションで厳しい温度、消費電力、そしてコストの要件を満たすことが求められています。

目標を達成するための 1 つの非常に効果的な方法は、固定小数点で信号処理チェーンを実装することです。ザイリンクスの FPGA および SoC はネイティブで可変精度をサポートするため、より低い精度を採用するソリューションに向けて日々展開する業界動向に簡単に適合できます。

ザイリンクスは Vivado® 高位合成 (HLS) を含むツール フローを提供しており、固定小数点を使用する、より精度の低い C/C++ デザインの実装を簡単に評価できます。

はじ めに : ザイリンクスがサポート するデータ 型

ザイリンクスの All Programmable デバイスおよびツールは、バイナリから倍精度浮動小数点まで、幅広いデータ型をサポートします。固定小数点での実装の方がリソースと電力の消費量が少ないため、固定小数点で実装されたデザインは、浮動小数点の同等のデザインよりも必ず効率が高くなります。デザインを固定小数点に移行すると、最大で 50% の消費電力とエリアの節約ができることも珍しくありません。

浮動小数点の場合と比較すると、固定小数点データ型では次のような利点があります。

- 使用するロジックリソースの削減
- 消費電力の低減
- BOM コストの削減
- レイテンシの短縮

ザイリンクス デバイスは、浮動小数点データ型が提供するダイナミック レンジ (最大で 7.3TFLOP の単精度浮動小数点 DSP 処理速度) を必要とするカスタマーの要件に対応できます。

業界最先端のザイリンクス ツール スイートは浮動小数点をサポートします。Vivado® 高位合成 (HLS) [参照 1] および System Generator for DSP [参照 2] はどちらも、半精度 (FP16)、単精度 (FP32)、および倍精度 (FP64) を含む多様な浮動小数点精度をネイティブにサポートします。System Generator では、カスタム精度を使用できることでさらに柔軟性が向上しています。これらのツールには、可変固定小数点データ型のネイティブ サポートもあります。

表 1: ザイリンクス ツールでの浮動小数点および固定小数点のデータ型のサポート

ザイリンクス ツール	FP16	FP32	FP64	カスタム FP	固定小数点
Vivado HLS	あり	あり	あり	なし	あり
System Generator for DSP	あり (1)	あり	あり	あり	あり
Floating Point Operator IP	あり	あり	あり	あり	あり (2)

注記:

1. System Generator for DSP には FP16 のネイティブ サポートはありませんが、カスタム サポートで FP16 に対応できます。
2. 浮動小数点演算子コアは、固定から浮動、浮動から固定、および精度が異なる浮動から浮動の変換をサポートします。

ザイリンクス デバイスおよびツールは可変精度データ型をサポートし、業界動向における変化 (たとえば画像分類では、推論の許容精度を保つために INT8 以下の固定小数点計算しか必要ない) [参照 3] [参照 4] にカスタマーが適合できるようする、単純で柔軟なソリューションを提供します。

従来、GPU などの演算負荷の高いワークロードで使用されるほかのデバイスは、単精度浮動小数点のみを効率的にサポートするように設計されてきました。これらを提供するベンダーは、変化し続ける動向に対応するために、製品の再設計を進めています。ザイリンクスのスケーラブルなアーキテクチャを利用すれば、カスタマーは信号処理チェーンの精度を変更することで、変化し続ける業界ニーズに簡単に対応できます。

浮動小数点と固定小数点のどちらの信号処理チェーンを実装するかを選択する場合は、消費電力、コスト、生産性、および精度におけるトレードオフを注意深く評価することが重要です。

ザイリンクスの柔軟性に優れた DSP48E2 スライスは、重要な DSP 計算用のすべてのデータ型で使用できます。DSP スライスをザイリンクスのツール セットと共に使用することで、新しい固定小数点のデザインを実装したり、変換が可能な一部のアプリケーションの既存デザインを浮動小数点から固定小数点に変換したりするときに、大きな利点と柔軟性が得られます [参照 5]。

C/C++ を使用している設計者向けに、ザイリンクスは Vivado HLS を提供し、任意精度型の固定小数点データ型をサポートしています。これにより、固定小数点を使用して簡単にデザインを作成したり、既存の C/C++ デザインを固定小数点に変換したりできます。

浮動小数点から 固定小数点への変換の利点

今日のほとんどすべてのデザインで、消費電力を最小限に抑えることは優先度の高い目標です。ほとんどのアプリケーションは、実稼働用に展開する前に、消費電力と温度に対する厳しい要件を満たす必要があります。

浮動小数点でデザインを作成すると、それより低い精度でデザインを作成した場合と比べて、消費電力が増えることは広く認識されています [参照 6] [参照 7]。浮動小数点 DSP ブロックが FPGA にハード化されている場合や、提供された DSP リソースと追加の FPGA リソースを使用してユーザーがソフト ソリューションを実装する必要がある場合にも、これは当てはまります。浮動小数点の実装は、同等の固定小数点ソリューションよりも多量の FPGA リソースを必要とします。このようにリソース使用量が多いことで、消費電力も増え、最終的にはデザインを実装する総コストが増大します。

浮動小数点のデザインを固定小数点に変換すると、このような難しい仕様に次のように対応できます。

- FPGA リソースの削減
  - 固定小数点データ型を処理する場合は、必要とする DSP48E2、ルックアップ テーブル (LUT)、およびフリップフロップが少なくて済みます。
  - 固定小数点数の場合、格納するために必要なメモリ量が少なくて済みます。
- 消費電力の低減
  - FPGA リソース使用量を削減すると、消費電力は本質的に低減されます。
- BOM コストの削減
  - 同じコストで、アプリケーションのほかの機能に追加リソースを割り当てることができます。
  - リソースの節約により、FPGA 内の演算処理能力を大幅に向上できます。向上した演算処理能力は、機械学習 DNN などの多くのアプリケーションで利点となります。
  - リソース節約により、デザインに必要なデバイスのサイズを縮小できます。
- レイテンシの改善
  - FIR の実装時に (特に DSP48E2 スライスで) リソースを削減することにより、固定小数点デザインでのレイテンシが改善されます。
- 同等のパフォーマンスおよび精度
  - 浮動小数点で達成できるダイナミック レンジを必要としないデザインやアプリケーションでも、固定小数点の実装により、同等の結果や精度が得られます。場合によっては、より良好な結果が得られることもあります。

以前は、浮動小数点から固定小数点へのデザインの変換はツール サポートが限られていたため困難でした。ザイリンクスの All Programmable デバイスをターゲットにしている C/C++ 設計者向けに、Vivado HLS はこの変換における課題を軽減しています。

この変換による利点は非常に大きいため、該当する場合は積極的に検討してください。特に、浮動小数点で可能になるダイナミック レンジや精度を必要としないデザインや、精度が失われることで展開アプリケーションの効率低下を招くことがないと考えられる場合は考察する価値があります。

例 : 浮動小数点 FIR フィ ルター から固定小数点への変換

Vivado HLS で単純な FIR フィルター デザイン [参照 8] を使用し、浮動小数点 FIR デザインから固定小数点バリアントへの変換が、結果として精度は同等のまま、どのようにリソースと消費電力の削減につながるかを示すことができます。

単精度浮動小数点 FIR

C++ FIR 関数コードで、最上位の関数は、FIR.h ヘッダー ファイル内にあるクラス CFir をインスタンシエートします。

```
#include "FIR.h"



// Top-level function with class instantiated

fp_acc_t fp_FIR(fp_data_t x) {

    #pragma HLS PIPELINE

    static CFir<fp_coef_t, fp_data_t, fp_acc_t> fir1;

    return fir1(x);

}
```

CFir クラスはメインの FIR アルゴズムであり、ヘッダー ファイル FIR.h 内で定義されています。

```
// FIR main algorithm
template<class coef_T, class data_T, class acc_T>
acc_T CFir<coef_T, data_T, acc_T>::operator()(data_T x) {
//caller uses #pragma HLS PIPELINE which makes this function pipelined as needed.
#pragma HLS ARRAY_PARTITION variable=c complete dim=1
#pragma HLS ARRAY_PARTITION variable=shift_reg complete dim=1
   int i;
   acc_T acc = 0;
   data_T m;

   loop: for (i = N-1; i >= 0; i--) {
      if (i == 0) {
         m = x;
         shift_reg[0] = x;
       } else {
         m = shift_reg[i-1];
         if (i != (N-1)) {
            shift_reg[i] = shift_reg[i - 1];
         }
      }
      acc += m * c[i];
   }
   return acc;
}
```

この関数には、デザインのすべての実装で II = 1 (繰り返し間隔 1) [参照 9] を確実に指定する、重要なARRAY_PARTITION プラグマが含まれています。さらに、PIPELINE プラグマが最上位の関数呼び出しに適用されます。

これらのプラグマは、並行した積の実装とそれに続く累積実行用の加算器ツリーと共に、データ型に関係なく FIR 関数全体を通して II = 1 を維持したまま、最小のレイテンシを実現します。

fp_FIR 関数で、fp_coef_t、fp_data_t、および fp_acc_t はすべて、float (つまり C++ のネイティブの単精度浮動小数点データ型) として定義されます。

```
// float

typedef float fp_coef_t;
typedef float fp_data_t;
typedef float fp_acc_t;
```

フィルター係数は、ヘッダー ファイル内の include コマンドによりロードされます。

```
template<class coef_T, class data_T, class acc_T>
const coef_T CFir<coef_T, data_T, acc_T>::c[] = {
   #include "FIR_fp.inc"
};
```

この係数は対称 FIR フィルターを作成しますが、この例では、DSP48E2 スライスの前置加算器は使用されません。前置加算器の使用により、効率はさらに上がります。

次の結果は、XCVU9P-2FLGB2104 デバイスで 400MHz クロック (2.5ns クロック周期) を目標とし、Vivado HLS で C 合成とインプリメンテーションを実行して 85 タップの FIR フィルター作成したものです。表 2 を参照してください。

表 2: 単精度浮動小数点 FIR のインプリメンテーション後の結果

単精度浮動小数点 (FP32)	
F_{MAX}	500MHz
レイテンシ (クロック サイクル)	91
繰り返し間隔 (II)	1
DSP48E2	423
LUT	23,101

この例では、単精度浮動小数点 FIR を実装するために、423 の DSP48E2 と約 23K の LUT が必要です。この結果として、レイテンシは 91 クロック サイクルとなり、F_{MAX} は 500MHz となります (400MHz という目標を大きく超える)。

固定小数点 FIR フィルターへの変換

DSP の効率を最大限にするために、固定小数点への変換では DSP スライスのバス幅のサイズ (27x18 ビット乗算器と 48 ビット アキュムレータ) を考慮する必要があります。これらのバス幅をデザインで許容される最小値まで削減すると、リソースと消費電力の節約の観点でその恩恵は最大になります。

この FIR フィルターの例では、次の固定小数点データ型は、DSP48E2 スライスのバス サイズと一致するように定義されています。つまり、18 ビットの係数に整数部 1 ビットと小数部 17 ビット、27 ビットのデータに整数部 15 ビットと小数部 12 ビット、48 ビットのアキュムレータに整数部 19 ビットと小数部 29 ビットなどと定義されています。

```
// fixed points
#include <ap_fixed.h>


typedef ap_fixed<18,1> fx_coef_t;
typedef ap_fixed<27,15> fx_data_t;
typedef ap_fixed<48,19> fx_acc_t;
```

Vivado HLS のネイティブの ap_fixed データ型を使用するには、`ap_fixed.h` ヘッダー ファイルを組み込んで、任意の固定小数点データ型を定義する必要があります [参照 9]。

さらに、400MHz クロック (2.5ns クロック周期) と XCVU9P-2FLGB2104 デバイスをターゲットにすると、固定小数点 FIR デザインの C 合成とインプリメンテーションの結果は、表 3 に示すようになりました。

表 3: 両方のデザインの実装後の結果比較

	単精度浮動小数点	固定小数点	固定小数点の利点
F_{MAX} (インプリメンテーション後)	500MHz	580MHz	16% 高速
レイテンシ	91	12	約 7.5 分の 1 に短縮
繰り返し間隔 (II)	1	1	–
DSP48E2	423	85	5 倍の DSP 効率
LUT	23,106	1,973	11 倍のロジック効率

結果が示すとおり、レイテンシと FPGA リソース使用量に特に注目すると、かなりの向上が見られます。

UltraScale アーキテクチャでは、必要であれば、複数の DSP48E2 スライスをカスケード接続することでさらに大きなバス幅をサポートできます。固定小数点のデザインおよびカスケード接続される DSP48E2 スライスを使用すれば、浮動小数点の実装と比較して、リソースと消費電力が大幅に向上します。

フィルター精度の比較

System Generator for DSP の Vivado HLS ブロック (ザイリンクス ブロックセットに含まれている) を使用すると、FIR フィルターの 2 つのインプリメンテーションを、MATLAB®/Simulink® 環境内で比較できます。図 1 を参照してください。

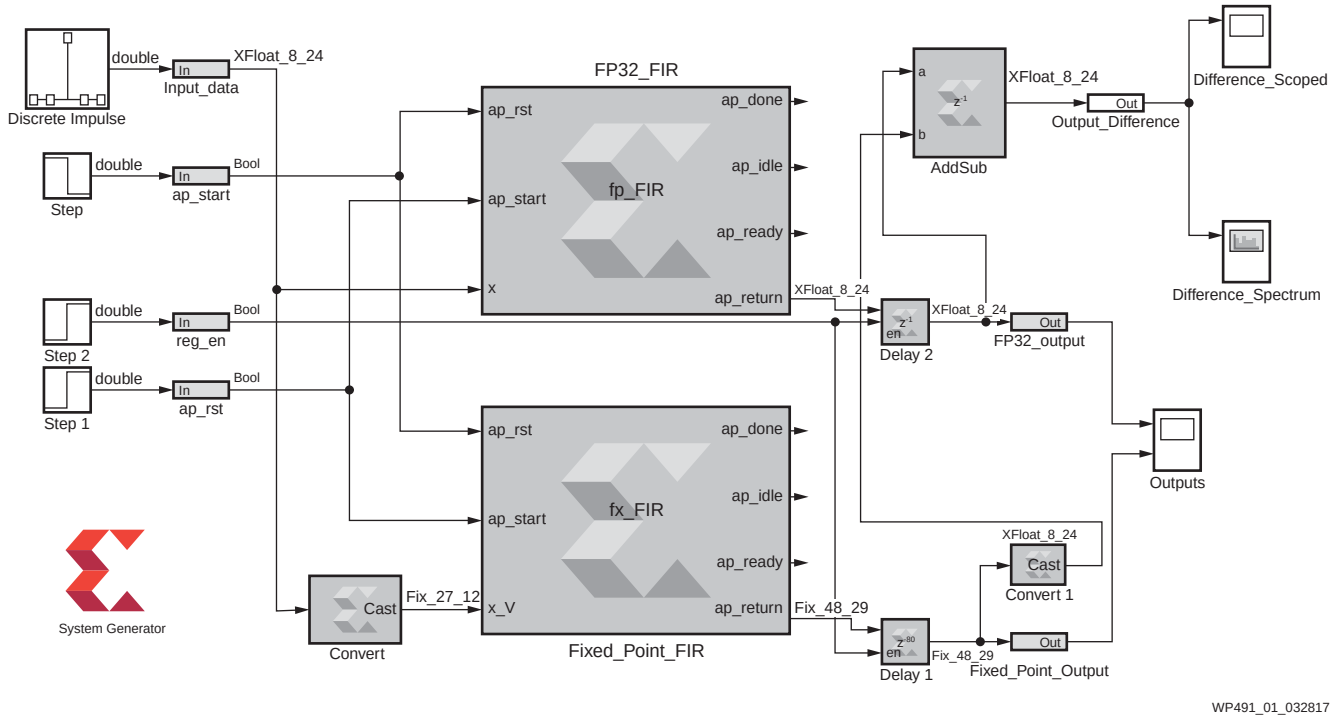


図 1: System Generator for DSP モデル - 2 つの HLS ソリューションの比較

System Generator モデルは、Vivado HLS が提供する単精度浮動小数点 (FP32) と固定小数点の FIR ソリューションを組み込むように構成されている、2 つの Vivado HLS ブロックから成ります。両方のブロックには同じ入力が適用されますが、インパルス信号は別個であり、各 FIR からの出力は Simulink スコープ上で比較されます。図 2 を参照してください。

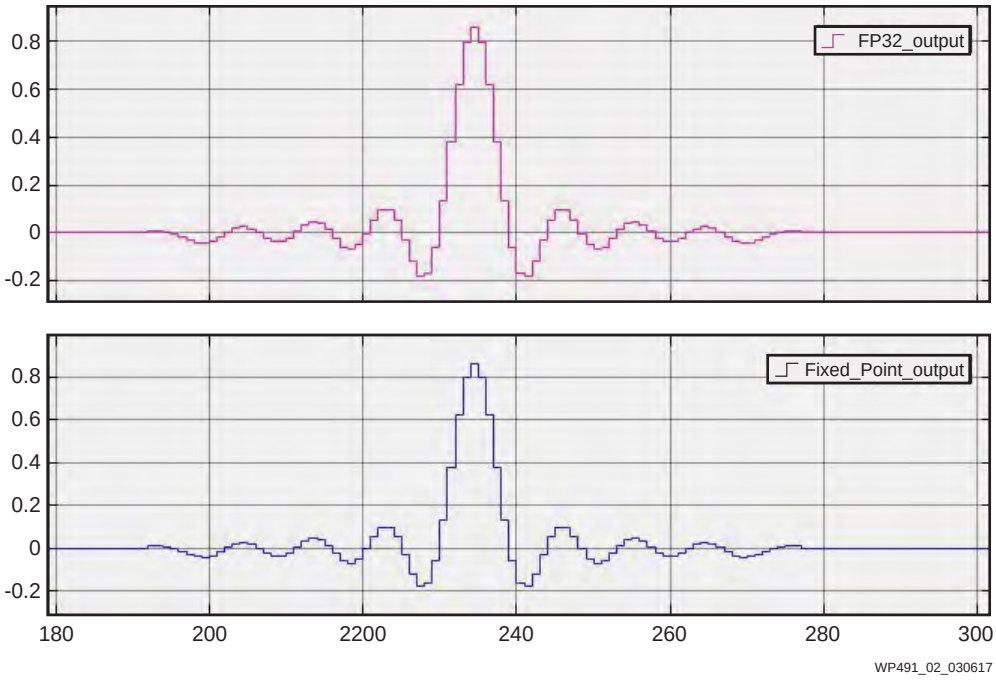


図 2: System Generator の 2 つのHLS デザインの出力

出力を簡単に比較するために、2 つのソリューション間のレイテンシ差を揃えるために、固定小数点側の結果を遅延させることが必要でした。

予測どおり、どちらの FIR フィルターもほとんど同じ結果を出し、差はごくわずかです。

信号をさらに分析するために、両出力が互いに差し引かれました。図 3 に示すスペクトラム分析プロットによると、結果の信号では、-100dBm ～ -160dBm の範囲内でごくわずかに精度が失われることが示されました。

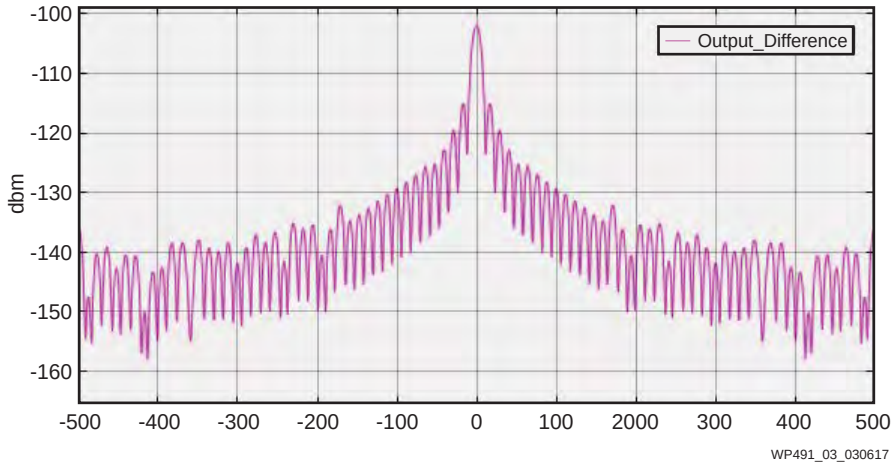


図 3: 2 つの出力間の差異の dB プロット

主要な利点

元の単精度浮動小数点 FIR フィルターと、変換された固定小数点 FIR フィルターの結果を比較します。固定小数点のデザインは、リソース削減とレイテンシ改善の両方を示しており、デザイン F_{MAX} は維持もしくは改善しています。図 4 を参照してください。

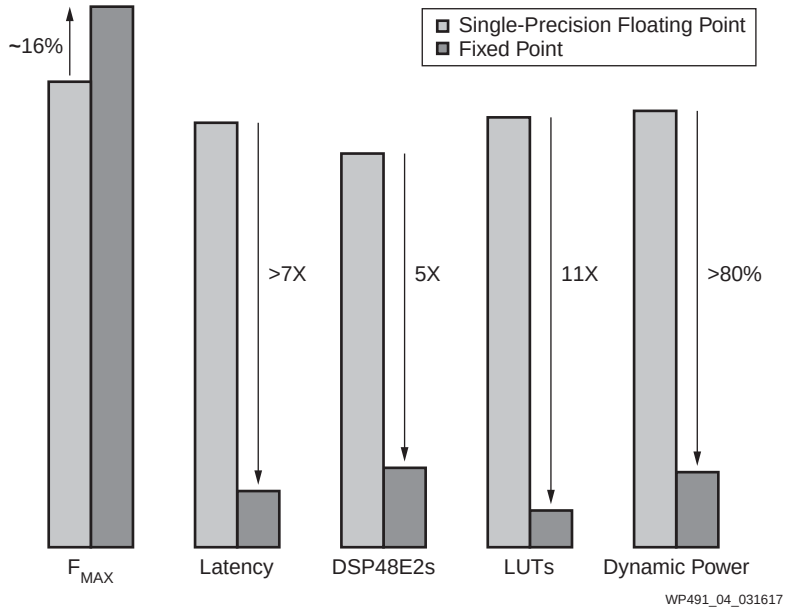


図 4: 固定小数点 - レイテンシ、リソース、および消費電力を削減、パフォーマンスはほぼ同じ

FPGA リソース使用量の大幅な削減

この例の固定小数点 FIR は、元の浮動小数点 FIR の 5 分の 1 未満です。

バス幅は、ハードウェアの DSP48E2 スライスへの最適なマッピングに合わせて選択されています。これにより、1 つの DSP48E2 スライスで、85 の各係数について並行して乗算を実行できます。結果、DSP48E2 スライスの使用量は浮動小数点ソリューションの場合の 20% まで削減されます。

さらに、固定小数点の実装では、浮動小数点演算を作成するために追加の LUT を必要としないため、FPGA ファブリック内の LUT も大幅に節約されます (約 90%)。

デザインにこのような FIR フィルターが 10 個あれば、デザインにおける推定消費電力はそれに合わせて変動します。表 4 は、10 個の FIR フィルター デザインの、単精度と固定小数点両方の実装における XCVU9P FPGA リソース使用量を示しています。このデザインで単精度浮動小数点と固定小数点の実装時を比較すると、リソースに著しい差異があります。

表 4: 2 つのデータ型ソリューションにおける 10 個の FIR フィルターで使用されるリソース

	DSP48E2		LUT	
	リソース数	デバイス使用量	リソース数	デバイス使用量
単精度浮動小数点	4,230	62%	231,060	20%
固定小数点	850	12%	19,730	2%

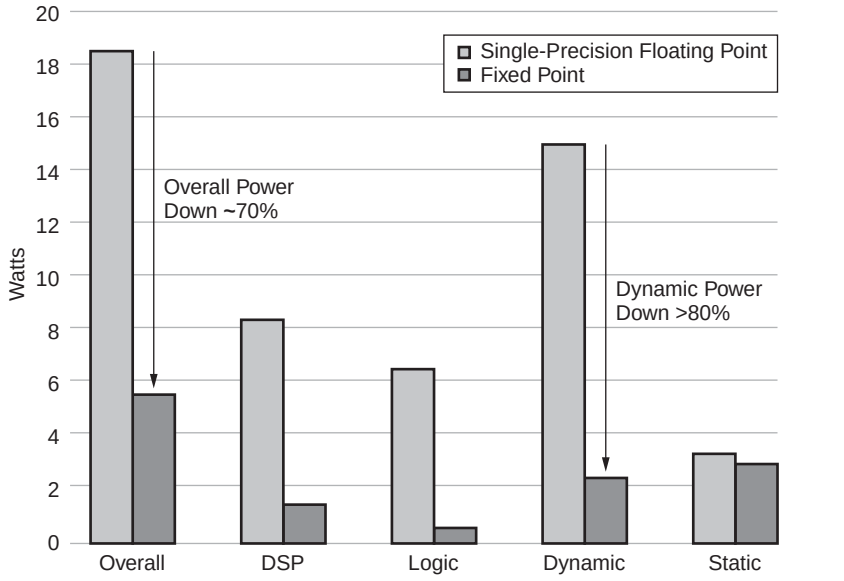
大幅なリソース節約により、デザインの機能セット、消費電力、パフォーマンス、およびコストに関して、設計者は多くの利点とこの広範囲にわたる効果を得ることができます。

消費電力の大幅な削減

大幅なリソース節約により、関連する消費電力も削減されます。

このホワイト ペーパーで説明されている単一 FIR フィルターで、2 つのインプリメンテーションにおける推定消費電力を比較すると、固定小数点 FIR では消費電力が 1.4W 抑えられています。どちらの場合も、デバイスのスタティック消費電力はちょうど 3W を超えています。個々の単精度浮動小数点 FIR デザインの総消費電力は 4.7W です。これは、3.3W を使用する固定小数点 FIR のデザインで、ダイナミック消費電力が 80% 以上節約されることを示します。

10 個の FIR フィルター デザインについて、2 つのインプリメンテーションの推定消費電力は、Xilinx Power Estimator (XPE) と表 4 で算出されているリソースを使用して確認できます。節約量の比較を図 5 に示します。



Note: 2016.4 XPE, assuming a worst case 25% toggle rate for the DSP48E2 slice and F_{MAX} 400MHz

WP491_05_031617

図 5: 10 個の FIR フィルターの例: 固定小数点の使用による大幅な消費電力節約

この 10 個の FIR フィルターの例では、デザインが固定小数点データ型を使用するように変換されると、全体的に 70% の消費電力節約を実現できます。多くの FPGA リソースを使用し、大量の浮動小数点信号処理を伴うデザインの場合、一部またはすべての浮動小数点信号処理チェーンを固定小数点に変換することで、消費電力が大幅に削減されます。

BOM コストの削減

浮動小数点デザインを固定小数点の実装に変換することで、FPGA リソース使用量は大幅に削減されます。この削減より、BOM コストを削減できる可能性があります。これは次の 3 とおりの方法で実現できます。

1. アプリケーション機能セットは、新しく使用可能になった FPGA リソースを利用して拡張できます。
2. FPGA の全体的な演算処理能力は、多くの FPGA リソース使用量削減とデータパスによる F_{MAX} の改善により、大幅に向上できます。
3. 必要な FPGA リソース数が少なくなったため、デザインをより小規模なザイリンクス FPGA に移行できるようになります。

同等の精度

単一 FIR フィルター デザインの 2 つのインプリメンテーションの出力を比較すると、固定小数点では、-100dBm ～ -160dBm の範囲で精度が失われているだけで、消費電力とコストは節減されており、フィルターとして同等の精度を提供していることがわかります。

固定小数点の実装で同じダイナミック レンジにすることはできません。そのようにするとデザインで精度が失われる可能性があります。ただし、多くのデザインでは要求されるのは最小の標準精度のみであるため、これは問題ではありません。単一 FIR の例と同様、多くのデザインが固定小数点への変換対象として適しています。

さらに高い精度が要求される値を使用するデザインでは、信号処理チェーンの中間値を浮動小数点から固定小数点に変換できる場合があります。この方法では、デザインの全部ではなく特定の部分を固定小数点に変換できます。つまり、必要な場合はダイナミック レンジを維持し、データパスでは確実に精度を維持しながら、固定小数点の実装のいくつかの利点を活用できるようになります。

レイテンシの改善

単一 FIR デザインの例では、レイテンシはフィルターにより、固定小数点の実装で 12 クロック サイクル、浮動小数点デザインで 91 クロック サイクルまで改善されます。リソース (特に DSP48E2 スライス) が削減されると 、レイテンシの改善を期待できます。

単一 FIR の例のように、レイテンシだけでなく F_{MAX} も向上できる可能性があります。単一 FIR ではインプリメンテーション後に、F_{MAX} が 16% 向上しました。

まとめ

ザイリンクスの All Programmable デバイスおよびツールは、複数の精度の浮動小数点や固定小数点を含む、多くのデータ型をサポートします。浮動小数点のデザインは、それが FPGA を (または GPU などのほかのアーキテクチャを) ターゲットとしているかどうかに関係なく、固定小数点の同じデザインよりも多くのリソースと消費電力を使用します。

業界動向は、アプリケーションによっては浮動小数点データ型を使用しない方向へと明確にシフトしていることを示しています。たとえば深層学習推論のワークロードでは、可能であれば INT8 以下の精度が使用されています。

今日の厳しいデザイン環境で温度と消費電力の要件を満たすことは一層困難になってきており、設計する側は消費電力を削減するためのあらゆる手段を評価する必要があります。そのような選択肢の 1 つが、浮動小数点のデザインを固定小数点に変換することです。

C/C++ で作業している場合は、Vivado HLS などのザイリンクス ツールを使用することで、この変換プロセスを簡単に実行できます。

設固定小数点データ型への変換に関連するトレードオフを十分に調査し、この変換によって得られる非常に大きな利点を十分に理解してください。

浮動小数点を使用し続けることで市場への投入は容易になるかもしれませんが、コストは高くつきます。固定小数点への変換に時間と労力を投資することは、リソース、コスト、および消費電力を低減するという観点からは非常に大きな利点があり、パフォーマンスが損なわれることはほとんどありません。

詳細は、ザイリンクス ウェブサイトの DSP ページ (https://japan.xilinx.com/products/technology/dsp.html) を参照してください。

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. ザイリンクス ウェブサイト: 『Vivado 高位合成』
2. ザイリンクス ウェブ サイト: 『System Generator for DSP』
3. 『Hardware-oriented Approximation of Convolutional Neural Networks』、ICLR 2016、Gysel ほか
4. 『Deep Compression: Compressing Deep Neural Networks With Pruning, Trained Quantization And Huffman Coding』、ICLR 2016、Han ほか
5. 『ザイリンクスデバイスでの INT8 に最適化した深層学習の実装』(WP486: 英語版、日本語版)
6. 『Deep Learning with Limited Numerical Precision』、Gupta ほか
7. 『Reducing Power by Optimizing the Necessary Precision/Range of Floating-Point Arithmetic』、Ying Fai Tong ほか
8. Github: デザイン ファイル
9. ザイリンクス ソフトウェア マニュアル: 『Vivado Design Suite ユーザー ガイド: 高位合成』

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
2017 年 3 月 30 日	1.0	初版

免責事項

本通知に基づいて貴殿または貴社 (本通知の被通知者が個人の場合には「貴殿」、法人その他の団体の場合には「貴社」。以下同じ) に開示される情報 (以下「本情報」といいます) は、ザイリンクスの製品を選択および使用することのためにのみ提供されます。適用される法律が許容する最大限の範囲で、(1) 本情報は「現状有姿」、およびすべて受領者の責任で (with all faults) という状態で提供され、ザイリンクスは、本通知をもって、明示、黙示、法定を問わず (商品性、非侵害、特定目的適合性の保証を含みますがこれらに限られません)、すべての保証および条件を負わない (否認する) ものとします。また、(2) ザイリンクスは、本情報 (貴殿または貴社による本情報の使用を含む) に関係し、起因し、関連する、いかなる種類・性質の損失または損害についても、責任を負わない (契約上、不法行為上 (過失の場合を含む)、その他のいかなる責任の法理によるかを問わない) ものとし、当該損失または損害には、直接、間接、特別、付随的、結果的な損失または損害 (第三者が起こした行為の結果被った、データ、利益、業務上の信用の損失、その他あらゆる種類の損失や損害を含みます) が含まれるものとし、それは、たとえ当該損害や損失が合理的に予見可能であったり、ザイリンクスがそれらの可能性について助言を受けていた場合であったとしても同様です。ザイリンクスは、本情報に含まれるいかなる誤りも訂正する義務を負わず、本情報または製品仕様のアップデートを貴殿または貴社に知らせる義務も負いません。事前の書面による同意のない限り、貴殿または貴社は本情報を再生産、変更、頒布、または公に展示してはなりません。一定の製品は、ザイリンクスの限定的保証の諸条件に従うこととなるので https://japan.xilinx.com/legal.htm#tos で見られるザイリンクスの販売条件を参照してください。IP コアは、ザイリンクスが貴殿または貴社に付与したライセンスに含まれる保証と補助的条件に従うことになります。ザイリンクスの製品は、フェイルセーフとして、または、フェイルセーフの動作を要求するアプリケーションに使用するために、デザインされたり意図されたりしていません。そのような重大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は、貴殿または貴社が単独で負うものです。https://japan.xilinx.com/legal.htm#tos で見られるザイリンクスの販売条件を参照してください。

自動車用のアプリケーションの免責条項

オートモーティブ製品 (製品番号に「XA」が含まれる) は、ISO 26262 自動車用機能安全規格に従った安全コンセプトまたは余剰性の機能 (「セーフティ設計」) がない限り、エアバッグの展開における使用または車両の制御に影響するアプリケーション (「セーフティ アプリケーション」) における使用は保証されていません。顧客は、製品を組み込むすべてのシステムについて、その使用前または提供前に安全を目的として十分なテストを行うものとします。セーフティ設計なしにセーフティ アプリケーションで製品を使用するリスクはすべて顧客が負い、製品の責任の制限を規定する適用法令および規則にのみ従うものとします。

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。