
アルテラの FPGA 浮動小数点 DSP デザイン・フロー

第三者機関による分析



Berkeley Design Technology, Inc.

概要

FPGA は、高負荷なデジタル信号処理アプリケーション向けの並列処理エンジンとして使用されることが多くなっています。ベンチマーク結果によれば、FPGA は、高度に並列化可能なワークロードの場合、DSP プロセッサや汎用 CPU よりも高い性能と優れたコストパフォーマンスを実現することができます。しかし、これまではほぼ例外なく固定小数点信号処理デザインのみで使用されています。FPGA は、高性能な浮動小数点演算を必要とするアプリケーション向けの有効なプラットフォームと見なされていません。FPGA の浮動小数点効率と性能は、長い処理レイテンシと配線密集によって制限されます。加えて、従来の FPGA デザイン・フローは、Verilog または VHDL によるレジスタ転送レベルのハードウェア記述に基づいており、複雑な浮動小数点アルゴリズムの実装には適していません。

アルテラは、アルテラ FPGA への浮動小数点デジタル信号処理アルゴリズムの実装プロセスを合理化し、従来よりも高い性能と効率を実現するために、新しい浮動小数点デザイン・フローを開発しました。そのデザイン・フローでは、例えば、乗算、加算、二乗を順次実行するといった、基本的な浮動小数点演算子からなるデータパスを構築するのではなく、浮動小数点コンパイラによって基本的な演算子を単一の関数またはデータパスに統合したフェーズド・データパスを生成します。それにより、従来の浮動小数点 FPGA デザインに存在する冗長性を排除しています。また、アルテラのデザイン・フローは、アルテラの DSP Builder アドバンスド・ブロックセットと MathWorks MATLAB および Simulink ツールによるハイレベル・モデル・ベースのフローです。アルテラは、FPGA 設計者がハイレベルで作業することにより、複雑な浮動小数点アルゴリズムの実装と検証を従来の HDL ベースのデザインよりも素早く行えるようになると期待しています。

BDTI は、アルテラの浮動小数点信号処理デザイン・フローに関して独自に分析を行いました。目的は、高負荷な浮動小数点信号処理アプリケーションに対してアルテラ FPGA で達成できる性能、およびアルテラの浮動小数点信号処理デザイン・フローの使いやすさを評価することでした。本書では、背景および方法論の詳細とともに BDTI の所見を示します。

目次

1. はじめに.....	2
2. 実装.....	3
3. デザイン・フローとツール・チェーン.....	5
4. 性能結果.....	8
5. 結論.....	10
6. 参考文献.....	11

1. はじめに

浮動小数点デザインの例

デジタル・チップの進歩により、従来は研究環境に限られていた複雑なアルゴリズムを組み込みコンピューティング・アプリケーションの分野に応用することが可能になりつつあります。例えば、大規模な計算リソースが利用可能であり、通常はリアルタイム計算を必要としない研究環境では、長い間、主に線形代数が使用されてきました。その具体的な目的は、多数の連立一次方程式を含む系の解を求めることです。大規模な系の解を求めるには、逆行列演算やある種の行列分解が必要です。これらの手法では、計算負荷が非常に高いことに加え、十分に高いダイナミック・レンジを使用しないと、数値的不安定に悩まされることがあります。そのため、そうしたアルゴリズムを効率的かつ正確に実装するには、浮動小数点デバイスが事実上不可欠でした。

アルテラは最近、従来の FPGA デザイン手法に比べて浮動小数点デザインの性能と効率を改善するとともに、アルテラ FPGA 上での浮動小数点 DSP アルゴリズムの実装を簡素化するために、DSP Builder アドバンスト・ブロックセット・ツール・チェーンに浮動小数点機能を導入しました。アルテラのアプローチの有効性を評価するために、BDTI は前世代入と後世代入を併用したコレスキー行列分解アルゴリズムを使用して実装された大規模な逆行列演算問題に焦点を当てました。これら 3 つの処理を組み合わせることにより、エルミート正定値行列の逆行列を求めて連立一次方程式 $\mathbf{Ax} = \mathbf{B}$ のベクトル \mathbf{x} の値を求めるコレスキー・ソルバを構成します。

コレスキー・ソルバは、時空間適応信号処理 (STAP) などの軍用レーダー・アプリケーションでの利用増加に伴って重要性が高まっています。また、ソルバの中核であるコレスキー分解自体も、共分散行列を求める必要がある多くの推定問題や最適化問題に使用されています。コレスキー分解は、計算負荷が非常に高いアルゴリズムであり、高いデータ精度を要求するため、浮動小数点演算が不可欠です。逆行列は、多くのアプリケーションにおいて重要な問題である一方で、広範な浮動小数点 DSP アルゴリズムの例としても役立ちます。例えば、コレスキー分解は、線形代数と有限インパルス応答 (FIR) フィルタを含むさま

表記と定義

\mathbf{M} 太字の大文字は行列を表します。
 \mathbf{z} 太字の小文字はベクトルを表します。

\mathbf{L}^* 行列 \mathbf{L} の共役転置。

\mathbf{I}^* 要素 1 の共役転置。

エルミート行列 自己の共役転置に等しい複素正方行列。これは実対称行列の複素拡張です。

正定値行列 エルミート行列 \mathbf{M} は、すべて 0 でない複素ベクトル \mathbf{z} について $\mathbf{z}^*\mathbf{M}\mathbf{z} > 0$ の場合、正定値です。 \mathbf{M} は本書の目的上、エルミート行列であるため、量 $\mathbf{z}^*\mathbf{M}\mathbf{z}$ は常に実数です。

コレスキー分解 $\mathbf{M} = \mathbf{L}\mathbf{L}^*$ となるように正定値エルミート行列 \mathbf{M} を下三角行列 \mathbf{L} と共役転置 \mathbf{L}^* に分解すること。

F_{max} FPGA デザインの最大周波数。

さまざまなデジタル信号処理アプリケーションに見られるベクトル・ドット積と入れ子ループを使用します。

詳しくはセクション 4 で示しますが、アルテラ Stratix IV FPGA は、クロック速度が 200MHz の場合、本書で説明するコレスキー・ソルバを使用して、サイズが 240x240 の行列に対して 1 秒間に 3,204 回の逆行列演算を、IEEE 単精度浮動小数点演算標準 (IEEE 754) による浮動小数点数表現を上回る精度で実行することができます。

本書で評価するコレスキー・ソルバの例は、ツール・バージョン 11.1 以降の DSP Builder アドバンスト・ブロックセット・ツール・チェーンに付属のデザイン例として、アルテラから提供される予定です。また、www.altera.co.jp/floatingpoint からダウンロードすることもできます。

浮動小数点デザイン・フロー

従来、FPGA は高負荷な浮動小数点アプリケーションに適したプラットフォームではありませんでした。FPGA ベンダーは浮動小数点プリミティブ・ライブラリを提供していますが、浮動小数点アプリケーションにおける FPGA の性能は非常に限られています。従来の浮動小数点 FPGA デザインの効率が低い理由の 1 つは、深いパイプライン化と浮動小数点演算子の広い演算構造によって、大きなデータパス・レイテンシと配線密集が生じることです。レイテンシは、データ依存性が高いデザインにおいて、対処が難しい問題をもたらし、最終的に得られるデザインは動作周波数の低いデザインにつながるものが少なくありませんでした。

アルテラ DSP Builder アドバンスト・ブロックセット・ツール・フローは、アーキテクチャ・レベルとシステム・デザイン・レベルの両面からこの問題に対処しています。アルテラの浮動小数点コンパイラは、基本的な浮動小数点演算子の組み立てによってデータパスを構築するのではなく、データパスの大部分を単一の浮動小数点関数に統合します。これは、データパス

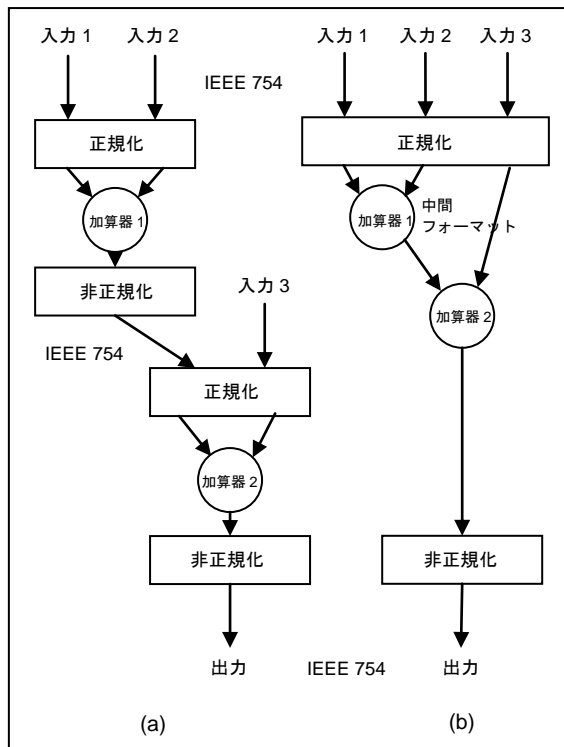


図 1 (a) 従来の浮動小数点実装、
(b) フューズド・データパスによる実装

全体に十分な精度を割り当てて正規化および非正規化ステップをできる限り省くことが目的であり、データパス内のビット増加を分析し、最適な入力正規化を適切に選択することによって行われます。IEEE 754 形式を使用するのはデータパス境界のみで、データパス内ではより大きな仮数幅を使用してダイナミック・レンジを広げ、連続する演算子間での非正規化および正規化処理の必要性を減らします。正規化および非正規化処理の実現には、単精度浮動小数点数の場合、最大 48 ビット範囲のバレル・シフタが必要となります。これは大量のロジックと配線リソースを消費し、FPGA への浮動小数点実装が効率的でない主な原因となります。フューズド・データパス・メソッドは、これらのバレル・シフタの数を大幅に削減します。より高い精度の仮数を必要とする乗算には、Stratix IV および Arria II デバイスの 36x36 乗算器モードが使用されます。図 1(b)は、フューズド・データパス・メソッドによる 2つの加算器チェーンの実装を図 1(a)の従来の実装と比較したものです。図 1(b)のフューズド・データパスでは、第 1 加算器の出力を非正規化し、第 2 加算器の入力を正規化する必要がなくなるため、演算子間の冗長性が排除されます。余分なロジック／ルーティングの排除とハード乗算器の使用により、複雑なデータパス間のタイミングとレイテンシが予測可能になります。単精度と倍精度のいずれの IEEE 754 浮動小数点アルゴリズムも、従来よりも少ないロジックで実装でき、より高い性能が得られま

す。アルテラによれば、フューズド・データパスを使用することにより、基本的な演算子で構成された同等のデータパスに比べて、ロジックの 50%削減とレイテンシの 50%低減につながります[1]。内部データ表現の幅が広がるため、基本的な IEEE 754 浮動小数点演算子を含むライブラリを使用する場合に比べて、全体的なデータ精度が概して高くなります。

アルテラの浮動小数点 DSP デザイン・フローには、アルテラの DSP Builder アドバンスド・ブロックセット、Quartus II 開発ソフトウェア RTL ツール・チェーン、ModelSim シミュレータのほか、MathWorks の MATLAB および Simulink ツールが組み込まれています。Simulink 環境では、複雑なシステムの記述、デバッグ、および検証をアルゴリズム動作レベルで行うことが可能です。DSP Builder アドバンスド・ブロックセットには、データ型伝播やベクトル・データ処理などの Simulink の機能が組み込まれており、アルゴリズム・デザイン・スペースを素早く探索することができます。

このホワイトペーパーでは、アルテラ DSP Builder アドバンスド・ブロックセット・ツール・フローを使用して、複素数データ型の浮動小数点コレスキー・ソルバのデザイン例を検証し、アルテラの浮動小数点デザイン・フローの効率と性能を評価した結果を示します。セクション 2 では、コレスキー・ソルバの実装について説明します。セクション 3 では、デザイン・フローおよびツール・チェーンを実際を使用して気付いた点について述べます。セクション 4 では、ハイエンドの Stratix IV EP4SE360H29C2 デバイスとミッドレンジの Arria II EP2AGX125DF25I3 デバイスという 2 種類のアルテラ FPGA への実装の性能を示します。最後に、セクション 5 で BDTI の結論を示します。

2. 実装

背景

$Ax = b$ という形の一連の一次方程式は、多くのアプリケーションで登場します。線形最小二乗を含む最適化問題にしても、予測問題のためのカルマン・フィルタにしても、MIMO チャネル推定にしても、一連の一次方程式 $Ax = b$ の数値解を求めることが問題であることに変わりありません。行列 A が対称かつ正定値である場合、これらの問題に使用される共分散行列も同様に対称かつ正定値であり、一般にコレスキー分解とコレスキー・ソルバが使用されます。アルゴリズムでは行列 A の逆行列を求め、したがって $x = A^{-1}b$ のベクトル x の値を求めます。コレスキー分解は、LU 分解や QR 分解といった他の方法に比べて 2 倍以上効率的です。これらの分解アルゴリズムはすべて再帰的であり、除算を含むため、行列サイズが大きくなるにつれて広い数値ダイナミック・レンジが必要になります。例えば、行列サイズが 4x4 程度の MIMO チャネル推定の場合、ほとんどの実装では浮動小数点演算

を使用して実行されます。軍用アプリケーションに見られるもののように高いスループットを必要とする大規模システムの場合、必要な浮動小数点演算の比率が高いことから、一般に組込みシステムでは対応できませんでした。設計者は、アルゴリズム全体をあきらめて準最適なソリューションを採用するか、複数の高性能

浮動小数点プロセッサの使用に頼ることが多く、結果としてコストとデザイン作業の増加につながります。

アーキテクチャの概要

ここでのデザイン例では、コレスキー・ソルバをパイプライン化し、並列動作する2つのサブシステムとして FPGA に実装します。最初のサブシステムは、コレスキー分解と前進代入（囲み欄「アルゴリズム」のステップ 1 および 2）を実行します。2 番目のサブシステムは後退代入（囲み欄のステップ 3）を実行します。入力行列はエルミート行列であり、この分解によって共役転置の関係にある三角行列が生成されます。そこで、入力行列 A の半分のみをロードし、下三角行列のみを生成し、使用後の A の要素を上書きすることにより、メモリ使用率が最適化されます。どちらのサブシステムもパイプライン化されており、入力ステージと処理ステージを利用して、あるメモリ領域で処理を実行し、残りのメモリ領域を新規データのロードに使用することができます。図 2 に示すように、分解および前進代入パイプライン・ステージの出力は後退代入の入力ステージに入ります。

数学的観点から見ると、方程式(6)の前進代入は、 l_{jk}^* の共役を除いて分解方程式(3)の一部と考えられるため、実装ではベクトル b の転置を行列 A の最後の行に付加することにより、大きな処理レイテンシを生じることなく単一の処理に結合されます。

分解の中核は、方程式(3)および(4)の複素ベクトル・ドット積エンジン（ベクトル乗算器とも呼ばれます）です。ベクトル・サイズ (VS) は、デバイス内の利用可能な DSP エレメント数によって制限され、Stratix IV SE360 デバイスでは最大 60 個、Arria II GX125 デバイスでは最大 30 個の複素数要素からなるベクトル・サイズを実装することができます。また、ベクトル・サイズは、クロック・サイクルごとにドット積エンジンに新規データ・セットを供給するのに必要な並列メモリ読み出し数にも対応し、したがって内部で使用されるデュアル・ポート・メモリの幅および分割を決定します。実装上の理由から、所与の行列サイズに対し、メモリを $\text{ceil}(N/VS)$ バンクに分割します ($\text{ceil}()$ はシーリング関数、 N は行列のサイズ)。この評価で使用される最大行列は、240x240 サイズのエルミート行列です。行列のサイズは、デバイス内の利用可能なメモリによって制限されます。

この分解は、左上隅から垂直方向にジグザグ状に右下隅まで、列単位で 1 要素ずつ実行します。まず各列の対角線要素を計算し、続いて同じ列内の下にある非対角線要素をすべて計算した後、右隣列の一番上の対角線要素に移動します。イベントおよび繰り返しのスケジュールは、3 レベルの入れ子 *for* ループによって制御します。DSP Builder アドバンスド・ブロックセットの「For Loop」ブロックは、繰り返しループの実装に最適です。このブロックによって、ハンド・

アルゴリズム

$Ax = b$ のベクトル x の値を求めるための再帰的コレスキー・アルゴリズムには、以下の3つのステップがあります。

ステップ 1: 分解 ($A = LL^*$ として下三角行列 L の解を求める)

$$l_{11} = \sqrt{a_{11}} \quad (1)$$

For $i = 2$ to n ,

$$l_{i1} = a_{i1} / l_{11} \quad (2)$$

For $j = 2$ to $(i-1)$,

$$l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} \times l_{jk}^*) / l_{jj} \quad (3)$$

end

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} (l_{ik} \times l_{ik}^*)} \quad (4)$$

end

上記の方程式における依存関係に注意してください。方程式(4)の対角線要素は、同じ行の左側の要素にのみ依存します。非対角線要素は、同じ行の左側の要素、および上の対応する対角線要素の左側の要素に依存します。

ステップ 2: 前進代入 (方程式 $Ly = b$ の y の値を求める)

$$y_1 = b_1 / l_{11} \quad (5)$$

For $i = 2$ to n ,

$$y_i = (b_i - \sum_{k=1}^{i-1} y_k \times l_{ik}) / l_{ii} \quad (6)$$

end

ステップ 3: 後退代入 (方程式 $L^*x = y$ の x の値を求める)

$$x_n = y_n / l_{nn}^* \quad (7)$$

For $i = n-1$ to 1 ,

$$x_i = (y_i - \sum_{k=i+1}^n x_k \times l_{ik}^*) / l_{ii}^* \quad (8)$$

end

ここで、

n = 行列 A の次元

l_{ij} = 行列 L の行 i 列 j の要素

a_{ij} = 行列 A の行 i 列 j の要素

y_i = ベクトル y の行 i の要素

b_i = ベクトル b の行 i の要素

x_i = ベクトル x の行 i の要素

ステップ 1 の出力はコレスキー分解、ステップ 3 の出力は一次方程式 $Ax = b$ の解 x です。このアルゴリズムでは、 $x = A^{-1}b$ を解くために行列 A の逆行列を間接的に求めます。

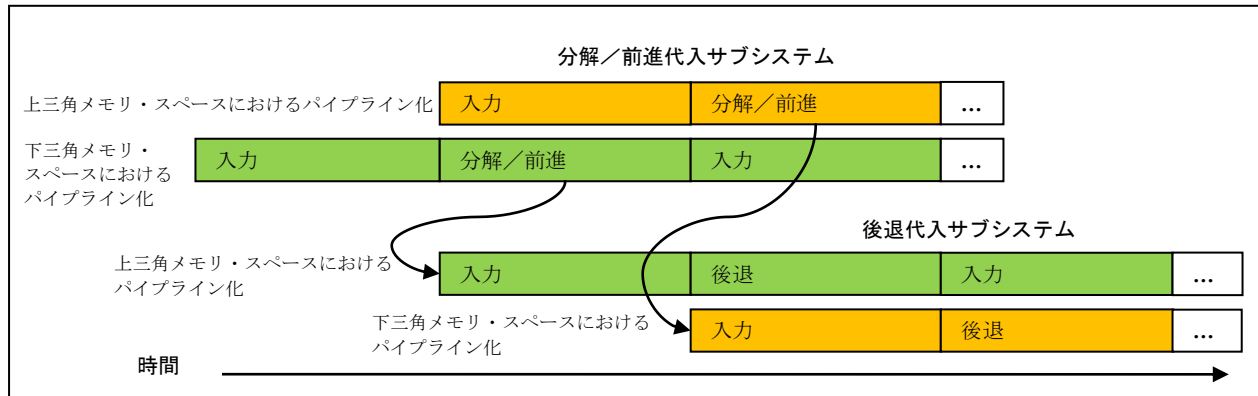


図2 プロセスのパイプライン化とメモリの再利用

コーディングによる RTL 実装の場合に必要なとされる複雑な制御信号が抽象化されるため、設計者はアルゴリズム自体に集中することができます。最外ループは $j=1 \sim N$ の列単位の処理を実行し、中央ループは $bank=1 \sim \text{ceil}(N/VS)$ のバンク単位の処理を実行し、最内ループは $i=1 \sim N$ の行要素を処理します。図 3(a) に処理シーケンスを示します。

ドット積エンジンは行列の行に対して動作し、方程式(3)、(4)、および(6)の加算項でベクトル・サイズまでの乗算を1つのサイクルで同時に計算します。ベクトル・ドット積がベクトル・サイズより短い場合、使用しない項をマスクし、加算に含めないようにします。ドット積がベクトル・サイズにより長い場合、部分積和を計算し、バンク境界に保存します。そして、その行に含まれる所与の要素のバンク出力がすべて得られた時点で、総和を実行します。バンク出力の総和は、DSP Builder アドバンスド・ブロックセットの浮動小数点加算器ブロックを使用して、単一のアキュムレータ・ループで実行します。このフィードバック・ループには、13 サイクルのレイテンシがあります。アキュムレータの処理完了を待つ間にこの大きなストールが発生するのを回避するために、3 レベルの入れ子ループの中央ループは、従来のソフトウェア実装において使用される「for Rows」ループではなく、「for Banks」ループになっています。このスワップを実行することにより、浮動小数点アキュムレータのレイテンシが隠され、ハードウェア使用率が向上します。現在のところ、DSP Builder アドバンスド・ブロックセットは、このタイプのレイテンシに対処するためのループ遅延エレメントの自動追加は行いません。これは、遅延エレメントを追加するとデジタル信号処理フィードバック・ループの機能が誤って変更される可能性があるためです。したがって、この値は、設計者が指定する必要があります。ただし、遅延値が不足している場合、DSP Builder によってエラー・メッセージが生成されます。設計者は、実験によって必要な値を見つけることができます。図 3(b) に、対角線要素 e_{ij} の計算を示します。この要素を生成するドット積は、ベクトル・サイズの2つのバンク全体と

3 番目の部分バンクにまたがります。for Banks ループは中央ループであるため、 e_{ij} 要素の完全な値は、 $j=VS$ および $j=2*VS$ 境界で、その下にあるすべての行の部分積和がすべて処理された時点で得られます。これらの部分積和は、内部 FIFO に一時的に保存されます。

ベクトル・サイズを選択は、ハードウェア効率とシステム・レイテンシに影響します。ベクトル・サイズが行列サイズに比べて大きい場合、ベクトル・サイズより大きい列番号に達するまでドット積内の多くの項が使用されないため、ハードウェア効率が低下します。行列およびベクトル・サイズを選択がレイテンシとハードウェア使用率に与える影響の詳細な分析については、セクション4を参照してください。

2 番目のサブシステムは後退代入を実行します。このサブシステムは、固有の入力および出力メモリ・ブロックを備えており、コレスキー分解/前進代入サブシステムと同様に入力ステージと処理ステージにパイプライン化されています。後退代入の複雑度は分解の N^3 に対して N^2 程度であるため、ドット積にはベクトル処理を使用せず、単一の複素数乗算器を使用します。これで、コレスキー分解/前進代入サブシステムに十分に対応可能です。

3. デザイン・フローとツール・チェーン

この評価のために、DSP Builder アドバンスド・ブロックセットを使用して作成されたコレスキー・ソルバの実装をアルテラから提供していただきました。BDTI のエンジニアは、評価に必要なアルテラおよび MATLAB のツールをインストールした後、デザイン・フローをよく理解するために簡単なトレーニング・クラスを受講しました。その上で、アルテラのデザインの検討、シミュレート、合成、および ModelSim RTL シミュレーションをいずれも Simulink 環境で行いました。その過程で、アルテラのデザイン・フローとコレスキー・ソルバ・サンプル・デザインの性能を評価しました。ツール・チェーンのインストールは、苦もなく直観的に行えました。

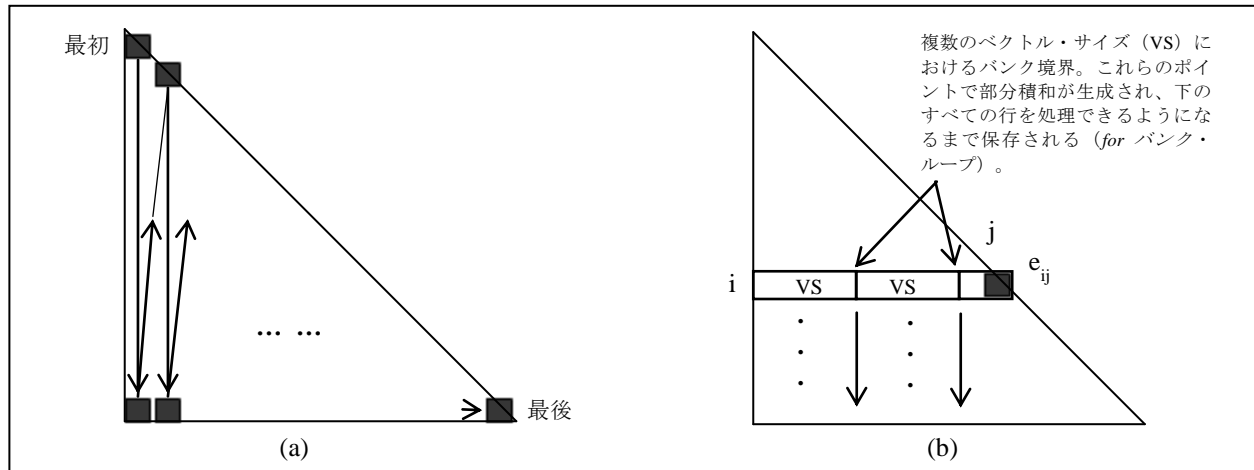


図3 (a) 処理シーケンス、(b) 対角線要素 e_{ij} の計算には、 $j=VS$ と $j=2*VS$ における2つの部分積和のほか、最後に残るドット積部分の加算を含む

Simulink は、MATLAB フレームワークを基礎に構築されているため、MATLAB フレームワークが必要です。コレスキー・モデルの入力ステイミュラスは、MATLAB m ファイル・スクリプトによって生成します。入力行列は、エルミート正定値を保証するために合成によって生成します。主な入力パラメータは、行列サイズとベクトル・サイズの2つです。ベクトル・サイズは、合成されたデザインにおけるベクトル・ドット積エンジンのサイズと最大許容行列サイズを決定するデザイン・パラメータです。この実装では、最大行列サイズはベクトル・サイズの4倍に設定します。行列サイズはユーザー変数であり、最大許容行列サイズ以下の任意のサイズにすることができます。

MATLAB m ファイル・スクリプトにより、合成によって生成された L および x と計算されたベクトル y を倍精度浮動小数点形式で保存し、これを Simulink モデルおよび合成された RTL デザインの誤差性能の評価基準とします。

Simulink 環境では、コレスキー・ソルバ・デザインはアルテラ DSP Builder アドバンスト・ブロックセットのブロックを使用します。これは DSP Builder スタンダード・ブロックセットとは別のブロックセットです。DSP Builder アドバンスト・ブロックセットは、ブロック・ベースの DSP アルゴリズムおよびデータパスのデザインを対象としており、より一般的かつ基本的なファンクション・ブロックを使用する DSP Builder スタンダード・ブロックセットに比べ、抽象化レベルが高くなっています。このライブラリには、標準のものよりも複雑な高速フーリエ変換 (FFT) および FIR フィルタ・ビルディング・ブロックのほか、50 種類以上の一般的な三角関数、算術関数、およびブール関数が含まれています。スタンダード・ブロックセットの要素とアドバンスト・ブロックセットの要素を同じ階層レベルのデータパス構造内で併用することはできません。浮動小数点コンパイラをサポートしているのは、DSP Builder アドバン

スト・ブロックセットのブロックのみであり、スタンダード・ブロックセットのブロックは浮動小数点処理に最適化されていません。また、スタンダード・ブロックセットは、ハンド・コーディングされた HDL のインポートに対応していますが、アドバンスト・ブロックセットの場合、ツールが HDL レベルでの最適化を実行できないため対応していません。一般に、ブロック・ベースのデザイン・エントリ手法は DSP アルゴリズムに適していますが、制御ベースかつステート・マシンを含むデザインの場合は、テキスト・ベースの手法の方が直観的に行えます。

Simulink でシミュレーションを開始すると、モデルがコンパイルされ、アルテラ Quartus II 開発ソフトウェア環境用の HDL コードと制約が生成され、ModelSim 環境用のテスト・ベンチとスクリプト・ファイルが作成された後、Simulink モデル・シミュレーションが実行されます。単一のコレスキー・ソルバのシミュレーションの実行に必要な時間は、行列サイズによって1分未満から6分程度です。Simulink によるシミュレーションでは、Quartus II 開発ソフトウェア・コンパイラの実行なしに、詳細なリソース使用量見積もりが生成されるため、さまざまなアルゴリズム変更に必要なデバイス・サイズを素早く決定できます。

アルゴリズム探索および対応する HDL 生成の容易性を評価するために、モデルで実験を行いました。ステイミュラス・ブロックおよびシミュレーション実行でベクトル・ドット積サイズ、行列サイズ、データ型などの入力パラメータを変更したところ、いずれの場合も数分以内に正しい RTL コードが生成され、シミュレーション出力は保存した MATLAB 評価基準と一致しました。

開発の抽象化レベルが上がると、アルゴリズム・スペースの探索、シミュレーション、および最適化された RTL 生成の時間を短縮することができます。しかし、この新しいアプローチによってもたらされる柔軟

性を十分に得るには、ハイレベルのブロック・ベースのデザインを開始する前にモデルの構造を熟慮する必要があります。この例において、ベクトル・サイズや行列サイズなどの入力パラメータによってデザイン・スペースを柔軟に探索できるようにするために、理解し、従う必要があるデザイン・メソッドがあります。また、このモデルの浮動小数点アキュムレータ・セクションに例示するように、良好なスループット・レートとリソース使用量を実現するには、いくつかのハードウェア・デザインを理解することも必要です。

7つのベクトル・サイズ/行列サイズの構成について、FPGA リソース使用量、達成可能な最大クロック・レート、スループット、および機能的な正確性を評価しました。クロック・レート、デバイス選択、スピード・グレードなどの FPGA デザイン制約は、Simulink 環境で指定します。

構成の合成はすべて、Simulink 環境から直接起動できる Quartus II 開発ソフトウェアで行いました。Quartus II 開発ソフトウェアによって報告された結果によれば、構成に応じて 154MHz~203MHz のクロック・レートが1回の配線で達成されます。さらに高いクロック・レートを達成するために、デザイン・スペース・エクスプローラ・ツールを使用しました。デザイン・スペース・エクスプローラは、Quartus II 開発ソフトウェアの一部として利用可能で、パスごとに異なるシードを使用して複数のルータ・パスを自動的に実行します。そして、最も高いクロック・レートが得られる配線を保存します。これはユーザーの介入が不要な自動プロセスであり、コレスキー・ソルバ・デザインの最適化された実装の収束には 4~6 時間を要しました。得られた改善は、使用した行列サイズ/ベクトル・サイズに応じて 8%~23%の範囲でした。例えば、行列サイズが 60x60、ベクトル・サイズが 60 の場合、1回の配線フィット時の 154MHz に対し、デザイン・スペース・エクスプローラを使用すると 202MHz が達成されました。

FPGA リソース使用量は、こうしたデザインに対する予想と一致しています。メモリ使用は行列サイズの二乗に比例し、乗算器使用率はベクトル・サイズとともに直線的に増加します。FPGA におけるメモリのグラニュラリティのため、メモリ要件を正確に決定することは一般に困難です。例えば、合成ツールは信号遅延に合わせてメモリ・ブロックの使用を選択することができますが、この例ではメモリの大部分は行列データ・ストレージによって消費されます。それに対し、乗算器使用率の予測は比較的容易です。ベクトル乗算器には複素数値の浮動小数点乗算ごとに 16 個の 18x18 DSP エレメントが必要です。ベクトル・ドット積エンジンには、ベクトル・サイズを 60 個の複素浮動小数点値とすると 960 個の 18x18 DSP エレメントが必要です。7つの構成ごとのリソース要件の内訳については、セクション 4 を参照してください。

スループットおよび性能は、修正版のデザインを使用して RTL レベルで評価しました。処理効率を測定するために、Simulink 回路図入力環境で、ベクトル乗算器のアクティブ信号によってイネーブルされるカウンタをデザインに追加しました。コレスキー・ソルバの実際の処理時間は、ModelSim シミュレーション環境での測定によって算出しました。Simulink による処理時間の計算は、いずれの構成も ModelSim での測定と一致しました。

セクション 4 で示す性能結果は、デザイン・スペース・エクスプローラ・ツールによって達成されたものであり、手動による最適化またはフロア・プランニングは一切行っていません。詳しく分析したところ、デザインにおけるワースト・ケースの遅延は、ロジックのチェインではなく配線によるものであることがわかりました。これは、ツールがデータパスを効率的にパイプライン化したことを示しています。ツール・チェインにより、ロー・レベルのデザイン変更やフロア・プランニングなしで、実用になる速度とリソース使用量が達成されました。

シミュレーション後のスクリプトにより、Simulink による IEEE 754 単精度浮動小数点出力と、合成によって生成された MATLAB 倍精度浮動小数点基準との差を計算します。同様に、DSP Builder アドバンスド・ブロックセットを使用して作成した単精度小数点合成 RTL と、合成によって生成した MATLAB 倍精度基準の ModelSim シミュレーション出力の差も計算します。誤差性能結果については、セクション 4 を参照してください。

DSP Builder アドバンスド・ブロックセット・デザイン・フローに関するトレーニングには、アルテラによる 4 時間のクラスのほか、約 10 時間のオンライン・チュートリアルおよびデモが必要です。また、BDTI が実際にコレスキー・ソルバ・モデルを探索し、変更を行うのに費やした時間は 90 時間程度でした。ツール・チェインで素早く作業するのに必要な時間と労力は、設計者のスキルと経歴によって左右されます。Simulink のブロック・ベースのデザインと FPGA ハードウェア・デザイン経験の両方に熟練したエンジニアであれば、DSP Builder アドバンスド・ブロックセット・アプローチを効率的かつ容易に使用できるものと思われます。MATLAB と Simulink の知識がほとんどない FPGA 設計者にとって、高い抽象化レベルでの設計は新しい考え方であるため、当初は困難が伴い、習熟するまでにかかなりの時間を要するかもしれません。しかし、いったん習熟すれば、HDL アプローチよりもはるかに速いデザイン・サイクルを実現できるはずで、なぜなら、パイプライン化などのハードウェア・デザインの詳細について心配する必要がなくなり、アルゴリズムの実装に集中できるようになるからです。最初に機能検証の大部分を Simulink 環境で終えた後に、完全に機能するシミュレーション

デバイス	構成 (行列サイズ/ ベクトル・ サイズ)	使用される ロジック・ エレメント (LE / 全体に占める 割合)	使用される DSP ブロック (18×18 乗算器/ 全体に占める割合)	メモリ (サイズ / 全体に占める割合)			クロック・ レート (F _{max} , MHz)
				M144K (ブロック)	M9K (ブロック)	MLAB (64 ビット・ ブロック)	
Stratix IV EP4SE360H29C2	240×240 / 60	162K / 57%	1,014 / 98%	0 / 0%	899 / 72%	13.4K / 9%	218
	180×180 / 60	133K / 47%	1,014 / 98%	0 / 0%	771 / 60%	4.7K / 3%	224
	120×120 / 60	131K / 46%	1,014 / 98%	0 / 0%	276 / 60%	4.6K / 3%	225
	60×60 / 60	143K / 50%	1,014 / 98%	0 / 0%	66 / 5%	8.1K / 6%	202
Arria II EP2AGX125 DF25I3	120×120 / 30	63K / 64%	534 / 93%	N/A	440 / 60%	1.3K / 3%	214
	60×60 / 30	64K / 65%	534 / 93%	N/A	284 / 39%	1.2K / 3%	228
	30×30 / 30	67K / 68%	534 / 93%	N/A	226 / 31%	3.0K / 6%	207

表 1 リソース使用量とクロック速度

を ModelSim で実行できるため、シミュレーション時間は大幅に短縮されます。

システム・レベルのデザインの経験はあるものの、ハードウェア・デザインのスキルがほとんどないエンジニアの場合、習熟には長い時間を要する可能性があります。ツール・チェーンでは、ハードウェア・コンパイル、合成、配線、および自動スクリプト生成が Simulink 環境内に統合され、パイプライン化や信号ベクトル化などの多くの複雑なデザイン・コンセプトが抽象化されていますが、実装を行うにはやはりハードウェア・デザインの知識がある程度必要です。

4. 性能結果

このセクションでは、アルテラのコレスキー・ソルバ浮動小数点実装例に関する BDTI の独立評価の結果を示します。

すべてのデザインは、アルテラの DSP Builder アドバンスト・ブロックセット 11.0 と MathWorks MATLAB 7.10 および Simulink 7.5 を使用して実装し、Quartus II 開発ソフトウェア 11.0 によって構築しました。RTL シミュレーションは、アルテラ ModelSim 6.6d を使用

して実行しました。デザインは、-2 スピード・グレードのハイエンド Stratix IV EP4SE360H29C2 デバイスと、-3 スピード・グレードのミッドレンジ Arria-II EP2AGX125DF25I3 という 2 種類のアルテラ 40nm FPGA デバイス (各デバイスの最速グレード) 向けに構築しました。いずれの場合も、デザイン・スペース・エクスプローラを使用してクロック・レート (F_{max}) を最適化しました。

ベクトル・サイズを 60 とした 4 種類の行列サイズと、ベクトル・サイズを 30 とした 3 種類の行列サイズという合計 7 つのケースをシミュレートし、構築しました。リソース使用量、性能、および精度の結果をケースごとに記録しました。表 1 に、達成されたリソース使用量とクロック速度を構成ごとに示します。

コレスキー・ソルバ・デザインは、行列サイズ・パラメータを備えており、最大デザイン・サイズより小さい行列サイズを実行時に使用することができます。表 1 に示したリソース使用量の結果に関して、テスト対象の行列サイズによって消費される実際のリソースを得るために、評価対象の行列サイズに等しい最大行列サイズ・パラメータで各構成を合成しました。

構成 (行列サイズ/ ベクトル・サイズ)	Simulink によって 報告された スループット (行列 / 秒)	ModelSim によって 報告された スループット (行列 / 秒)	ベクトル乗算器使用率	
			Simulink による報告	ModelSim による報告
240×240 / 60	3,204	3,204	88%	88%
180×180 / 60	6,113	6,113	78%	78%
120×120 / 60	12,680	12,680	58%	58%
60×60 / 60	28,998	28,998	27%	27%
120×120 / 30	9,921	9,921	69%	69%
60×60 / 30	27,886	27,886	33%	33%
30×30 / 30	59,665	59,665	14%	14%

表 2 コレスキー・ソルバの性能

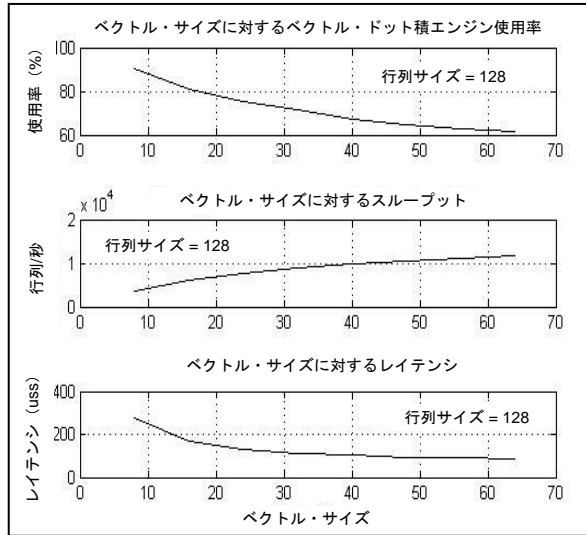


図4 ベクトル・サイズの影響

ここで注目すべき点は、1個または数個の浮動小数点演算子からなるごく単純なデザインの場合、表1に示したコレスキー・ソルバの例で達成されるクロック・レートよりもはるかに高速に動作可能であることです。しかし、浮動小数点デザインが複雑になるに従って、配線密集によって性能が大きく制限される傾向にあるため、これは特に意味のある比較ではありません。

表2に、7つの構成ごとにコレスキー・ソルバ実装の性能を示します。ベクトル乗算器使用率は、コレスキー・ソルバによって消費される合計サイクルでベクトル乗算器のアクティブ・サイクルを除算することによって計算しました。これらはいずれも Simulink による報告に基づいています。ただし、入力行列が三角行列であることから、この使用率にはドット積エンジン内の無効項が考慮されておらず、単一の作業単位としてのドット積エンジンの使用率を示しています。性能は、200MHz のクロック・レートを前提としていま

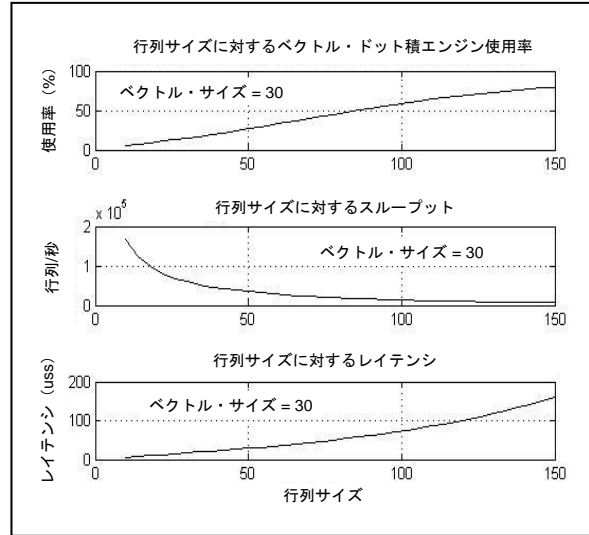


図5 行列サイズの影響

す。表に示すように、Simulink と ModelSim による計算は一致しています。スループットは、コレスキー・ソルバの実行につき消費されたサイクルで 200MHz を除算することによって計算しました。後退代入サブシステムは、コレスキー分解よりも低いレイテンシで並列実行されるため、全体のスループットには影響しません。各ケースのレイテンシはスループットの逆数です。コレスキー分解および前進代入のドット積エンジンはアルゴリズムの中核であり、表2の使用率は実装の効率の正しい評価基準となります。ここで、行列サイズとベクトル・サイズの比率が 1:1 に近づくにつれて効率が低下している点に注目してください。仮に分散行列が小さい場合、複数の行列の分解をインターリーブし、同じデータパスで計算することができます。行列数と行列サイズを掛け合わせた数値が1回の分解のレイテンシより大きい限り、この時分割多重によって効率が向上します。

行列サイズ/ ベクトル・ サイズ	機能	MathWorks Simulink IEEE 754 単精度 浮動小数点 (ノルム / 最大誤差)	アルテラ DSP Builder による 合成 RTL 単精度 浮動小数点 (フューズド・データパス) (ノルム / 最大誤差)
240×240 / 60	コレスキー分解	3.53e-5 / 2.84e-6	2.01e-5 / 2.71e-6
	前進代入	5.06e-4 / 9.44e-5	1.87e-4 / 2.88e-5
	後退代入	2.29e-5 / 3.74e-6	1.04e-5 / 1.27e-6
60×60 / 60	コレスキー分解	8.89e-6 / 1.28e-6	3.97e-6 / 6.20e-7
	前進代入	9.35e-5 / 3.41e-5	2.26e-5 / 5.70e-6
	後退代入	1.98e-5 / 6.18e-6	4.13e-6 / 1.15e-6
120×120 / 30	コレスキー分解	1.38e-5 / 1.20e-6	8.70e-6 / 1.41e-6
	前進代入	1.26e-4 / 2.65e-5	5.95e-5 / 1.21e-5
	後退代入	1.17e-5 / 2.65e-6	5.80e-6 / 1.07e-6
30×30 / 30	コレスキー分解	3.33e-6 / 7.68e-7	1.80e-6 / 3.37e-7
	前進代入	2.20e-5 / 9.44e-6	6.90e-6 / 2.24e-6
	後退代入	8.59e-6 / 4.09e-6	2.62e-6 / 1.09e-6

表3 Simulink モデルと合成された RTL のエラー性能

行列サイズに対するベクトル・サイズを選択は妥協点を見つけることになり、アプリケーションによって異なります。ベクトル・サイズが行列サイズよりもはるかに小さいデザインでは、レイテンシが大きくなる代わりにリソース効率が高くなります。それに対し、ベクトル・サイズが行列サイズと同等であるデザインでは、リソース使用量が多くなる代わりにレイテンシが短くなります。一般に、ベクトル・サイズを行列サイズの 1/4 に合わせるのが良好な妥協点となります。図 4 および図 5 に、リソース使用量、スループット、およびレイテンシに対する各種ベクトル・サイズおよび行列サイズの影響を示します。

表 3 に、単精度浮動小数点演算を使用する Simulink シミュレーションおよび RTL 実装におけるコレスキー・ソルバの誤差性能を示します。誤差は、Simulink および ModelSim RTL シミュレーションの各出力を、合成によって生成された倍精度浮動小数点基準 L 、 x 、および y と比較することによって計算したものです。多くの場合、RTL 実装はフェーズド・データパス・メソドロジの恩恵を受けており、列(3)および(4)のノルムを比較すれば明らかのように、標準の IEEE 754 単精度実装よりも高い精度を達成しています。結果の行列（またはベクトル）における全体的な誤差の大きさを測定するために、フロベニウス・ノルムを使用しました。これは次式によって計算されます。

$$\|E\|_F = \sqrt{\sum_{i=1}^N \sum_{j=1}^N |e_{ij}|^2}$$

ここで、
 N は行列のサイズ、
 i, j はそれぞれ行列の行および列番号、
 e_{ij} は行列要素 (i,j) 内の誤差です。

ベクトルのフロベニウス・ノルムも行列の場合と同様ですが、総和は一方方向にのみ実行されます。最大誤差は、行列内のすべての要素 e_{ij} 全体または列ベクトル内のすべての要素 e_{ij} 全体の最大絶対誤差です。

5. 結論

本書では、アルテラの DSP Builder アドバンスト・ブロックセット・デザイン・フローによる FPGA への浮動小数点 DSP アルゴリズムの実装に対する新しいアプローチを評価しました。このアプローチでは、設計者が Simulink 環境においてアルゴリズム動作レベルで作業することが可能です。ツール・チェーンにより、アルゴリズムのモデリング/シミュレーション、RTL 生成、合成、配置配線、およびデザイン検証ステージが Simulink 環境内に統合されます。この統合により、アルゴリズム・レベルと FPGA レベルの両面で迅速な開発とデザイン・スペースの素早い探索が可能になり、ひいては全体的な設計作業の削減に

つながります。アルゴリズムをハイレベルでモデル化してデバッグした後、デザインを合成し、任意のアルテラ FPGA をターゲットにすることが可能です。

この評価の目的上、アルテラ DSP Builder アドバンスト・ブロックセットを使用して Simulink でモデル化した単精度複素数データ IEEE 754 浮動小数点コレスキー・ソルバをデザイン例として使用しました。このデザインは、200MHz で動作する Stratix IV S360 FPGA デバイスの場合、サイズが 240x240 の行列に対して 1 秒当たり 3,204 回の複素浮動小数点コレスキー・ソルバ実行という性能を達成しました。この性能は、手動の最適化またはフロア・プランニングを一切行わずに、アルテラ DSP Builder アドバンスト・ブロックセット・ツール・フロー (Quartus II 開発ソフトウェアおよびデザイン・スペース・エクスプローラ・ツールを含む) を使用して達成されたものです。Simulink によるハイレベルのブロック・ベースのデザインから、ツール・チェーンによってデザインのパイプライン化、時間最適化、および合成が自動的に実行された結果、実用になる速度とリソース使用量が達成されたということです。

アルテラの浮動小数点 DSP デザイン・フローには、アルテラの DSP Builder アドバンスト・ブロックセット、Quartus II 開発ソフトウェア RTL ツール・チェーン、ModelSim シミュレータのほか、MathWorks の MATLAB および Simulink ツールが組み込まれています。Simulink 環境では、複雑なシステムの記述、デバッグ、および検証をアルゴリズム動作レベルで行うことが可能です。DSP Builder アドバンスト・ブロックセットには、データ型伝播やベクトル・データ処理などの Simulink の機能が組み込まれており、アルゴリズム・デザイン・スペースを素早く探索することができます。

アルテラの浮動小数点デザイン・フローでは、単一のプラットフォームの下でツールを合理化することにより、FPGA への複素浮動小数点 DSP アルゴリズムの実装プロセスが単純化されています。また、フェーズド・データパス・メソドロジにより、従来の限界を超える高い性能と効率で複素浮動小数点データパスを実装することが可能です。

その一方で、この新しいアプローチでは、DSP Builder アドバンスト・ブロックセットを使いこなすまでかなりの時間が必要であることも事実です。これは、MATLAB と Simulink に慣れてない設計者に特に言えることです。ブロック・ベースのデザイン・エントリ手法は、従来のハードウェア設計者には当初理解しにくいかもしれません。また、Simulink が持つ柔軟性を十分に生かすために理解し、従うべきデザイン・メソドロジもあります。本書で紹介した例で行列サイズやベクトル・サイズについて行ったようなアルゴリズム・スペースの探索を行えるようにするには、事前にモデルの構造を十分に熟慮する必要があります。

現在、**DSP Builder** アドバンスト・ブロックセットを使用したデザインは、ブロックセットによって最適化された性能が得られるように提供されるエレメントに限られています。一方、**DSP Builder** スタンダード・ブロックセットのエレメントは、浮動小数点コンパイラに最適化されておらず、同じ階層レベルのアドバンスト・ブロックセットと併用することはできません。また、ハンド・コーディングされた **HDL** ブロックは、スタンダード・ブロックセットにしかインポートできません。さらに、**DSP Builder** アドバンスト・ブロックセットは、**DSP** 実装を対象にしており、高負荷な制御やステート・マシンを必要とする設計への使用は制限されることも考えられます。

現在サンプル出荷中の次世代 **28nm Stratix V** および **Arria V FPGA** デバイスは、豊富な乗算器とメモリ容量を備えています。新しい可変精度 **DSP** ブロックと、より高精度の **27x27** 乗算器モードにより、浮動小数点乗算に必要なリソースは現在の **Stratix IV** および **Arria II** デバイスよりも少なくなるはずで、これらの機能強化と潜在的に優れた浮動小数点性能により、次世代 **FPGA** では浮動小数点デザインの実装が一層容易になることが期待されます。

6. 参考文献

[1] S.S. Demirsoy, M. Langhammer, 2009. "Fused datapath floating point implementation of Cholesky decomposition." *FPGA '09*, February, 2009.