積和演算ライブラリを用いた CyberWorkBench®高位合成フロー

酒井完[†1]　青山哲也[†1]　高橋渡[†1]　本田晋也[†2]　中本幸一[†3]　若林一敏[†1]

本稿では，積和演算ライブラリを高位合成ツール上で実現するフローを提案する．まず，既存の高位合成ツール上の FIR フィルタ専用ライブラリで実現されているハードウェア構造を一般化し，本ハードウェア構造は従来適用対象であったパイプライン回路だけでなく順序回路においても使用可能であること，また DSP ブロックの適切な利用により高性能なデザイン生成が可能であることを示した．高位合成ツール上で本ハードウェア構造を実現する為，積和演算ライブラリ関数の呼出し部を適切な単位で分割し，各分割単位を多サイクル入力型のパイプライン演算器として合成するフローを提案した．本フローを高位合成ツール CyberWorkBench®上に構築し，より高性能なデザインが得られていることを確認した．

Multiply-Accumulate Class Library for CyberWorkBench® High-level Synthesis

TAMOTSU SAKAI[†1]　TETSUYA AOYAMA[†1]　WATARU TAKAHASHI[†1]
SHINYA HONDA[†2]　YUKIKAZU NAKAMOTO[†3]
KAZUTOSHI WAKABAYASHI[†1]

This paper proposes a high level synthesis design flow using the class library for multiply-accumulate (MAC) operation. Firstly, we generalize the hardware structure which is synthesized from the library dedicated for FIR filter on conventional high-level synthesis tool. We then indicate this generalized structure can be used in sequential circuit design as well as pipelined circuit design and a high performance design can be gained by taking full advantage of functionality provided in the embedded DSP blocks. To synthesize this generalized structure from high level synthesis tool, we propose high level synthesis design flow: MAC library function call is divided into several units and each division unit is synthesized as a pipelined operator with input at multiple cycles. Applying this proposed flow to high level synthesis tool CyberWorkBench®, high-performance designs in a various range of applications can be gained.

1. 背景

近年，複雑かつ大規模化するシステムにおいて高性能な処理が要求される中で，高い並列構造を持つデバイスである FPGA が注目されている．FPGA においては，従来 VHDL, Verilog といったハードウェア記述言語(HDL)を用いてシステムを記述するのが一般的だったが，近年は C/C++/SystemC といったより抽象度の高い言語を用いて記述し，それらを高位合成ツールに入力して HDL を生成する設計手法(ESL)が主流になりつつある．

高位合成ツールにより C/C++/SystemC 等で記述された複雑かつ大規模なシステムの FPGA 化が容易となったが，高性能なデザイン生成においてはより一層の向上余地があると言える．

本稿では積和演算(multiply-accumulate operation)に注目し，より高性能なデザインを生成する為の高位合成フローを提案する．積和演算はディジタル信号処理や音声・画像処理，ニューラルネットワークなど様々な応用領域で用いられる基本的な演算である．積和演算が用いられるアルゴリズムを合成し，高性能なデザインを生成する為には，FPGA 上にハードマクロとして存在する DSP ブロックを適切に利用することが重要であると考えられる．具体的には，

(1)　DSP ブロックのカスケード接続構造の実現
(2)　DSP ブロック中のアキュムレータ利用

により，高性能なデザインの生成が可能であると考えられる．

現状の高位合成ツールにおいては，FIR フィルタ専用の高位合成用ライブラリが用意されており，上記(1), (2)の実現により，高性能なデザイン生成が可能となっているが，固定化されたパイプライン用のハードウェア・アーキテクチャである為，順序回路において用いることはできない．また，より汎用的な位置付けとなる積和演算向けに上記(1),(2)を実現する機構を実現出来ていない為，高性能なデザイン生成は容易ではなかった．

そこで本稿では，積和演算専用の高位合成用ライブラリを用意し，多様な回路（パイプライン回路と順序回路）において利用可能であり，また上記(1), (2)の実現により高性能なデザイン生成を可能とする高位合成フローを提案する．

本稿で報告する成果は以下の通りである．

(1)　以下の特徴を持つ C++ベースの積和演算ライブラリを用いた高位合成フローを提案した．

†1 日本電気株式会社
NEC Corporation.
†2 名古屋大学
Nagoya University
†3 兵庫県立大学
University of Hyogo

- 多用な回路(パイプライン回路と順序回路)に適用可能
- 性能指標(スループット，レイテンシ，面積)に応じてカスタマイズ可能

(2)　実データへの適用を行い，従来得られなかった高性能なデザインが高位合成ツール CyberWorkBench®から生成されることで，提案する高位合成フローの有効性を示した．

本稿の構成は以下の通りである．

2 章では，用語定義と前提知識を与える．

3 章では，本稿で提案する積和演算ライブラリを用いた高位合成フローに言及する．

4 章では，3 章で提案した高位合成フローを高位合成ツール CyberWorkBench®上に構築した際の評価結果を報告する．

最後に 5 章にて結論を述べる．

2. 準備

2.1 用語定義

本節では，本稿で用いる用語の定義を与える．

積和演算または **mac 演算**(multiply-accumulate operation)は乗算の結果を順次累積していく演算またはその集合体を示し，一般には，$y = \sum_i a_i \cdot b_i$ 等といった演算式で表現される．一方，**fma 演算**(fused multiply-add operation)は，乗算の結果を加算する演算(d = a・b + c)を指すものとする．

2.2　積和演算と FPGA

積和演算は，ディジタル信号処理や音声・画像処理，ニューラルネットワークなど様々な応用領域で用いられる基本的な演算である．

多くの汎用プロセッサにおいては，積和演算命令が追加されており，高性能な積和演算処理が可能である一方，消費電力や大きさに制限がある携帯機器等の場合においては，ディジタル・シグナル・プロセッサ(DSP)のような安価かつ低電力でありながら高性能を提供するデバイスが開発されてきた[1]．近年は，複雑化かつ大規模化するシステムにおいて高性能処理が要求される中で，コンフィギュラブル・ロジックブロック(CLB)，メモリの他に専用リソースである DSP ブロックを有する高い並列構造を持ち，低電力でありながら高い性能を実現するデバイスである FPGA が注目されている[2]．

2.3 FPGA における DSP ブロック

FPGA における DSP ブロックは各 FPGA ベンダーにおいて提供されている．例えば，Intel 社が提供されている DSP ブロックの構造は図 1 のようなものである[3]．

本ブロックにおいては，専用乗算器と専用加算器を含んでおり，乗算結果と外部入力との加算を高速に実行することができる．そして外部から与えるフラグ(accumulate フラグ)によってその演算結果を蓄積するアキュムレータ機能を実現することができる．このように DSP ブロックはディタル信号処理をはじめとする積和演算処理に適した構造と言える．

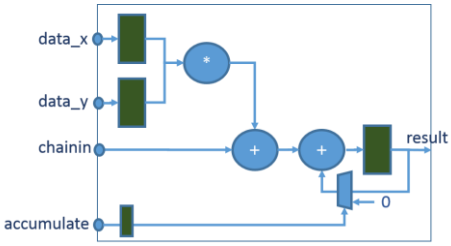


図 1 DSP ブロック構造

2.4 高性能なハードウェアを実現する構造

積和演算が用いられるアプリケーションにおいて高性能なハードウェアを構成する為には，

1. DSP ブロックのカスケード接続構造の実現
2. DSP ブロック中のアキュムレータ利用

により，高性能なデザインの生成が可能であると考えられる．FIR フィルタの例を用いて具体的に説明する．

図 2 は，タップ数 8 の FIR フィルタのブロック構成図であり，8 個の DSP ブロックがカスケード接続された構造となっている．

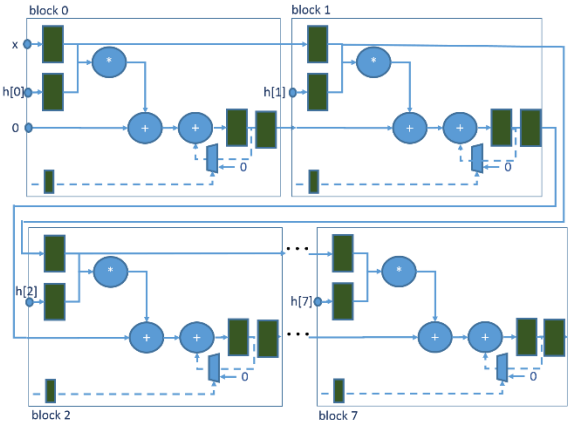


図 2 DSP のカスケード接続構造

この構造は Systolic Architecture と呼ばれ，II=1[1]で動作するパイプライン回路[4]であり，クリティカルパスが DSP ブロック内の専用乗算器と専用加算器の合計遅延となる為，非常に高い周波数で動作可能な構造として知られている．

一方，タップ数の増加に比例し，DSP ブロックのリソース数とパイプライン・レイテンシが増加する点が短所として挙げられる．

レイテンシの短縮化・面積の低減策としては、II を 2 以上としてスループットを抑制する構造が考えられる．

図 3 は図 2 で実現した FIR フィルタを II=2 で実現した場合のハードウェア構造を，タイミングチャートを用いて概念的に説明した図である．すなわち，積和演算を 2 個(=II 個)に分割し，分割された積和演算(以降，分割積和演算と呼

1 Initiation Interval の略．パイプライン処理の開始サイクル間隔を指す．

ぶ)を同一のリソースで使用可能なように，1 サイクルずらして処理する．本例では，分割積和演算は 4 個の DSP ブロックで構成されており，最後の DSP ブロックにおける accumulate フラグを調整することにより，分割積和演算の演算結果を積算し，元の積和演算結果と等価な結果が得られるようにする．具体的には，最初の分割積和演算においては，以前の積算結果を用いない為，最後の DSP ブロックにおける accumulate フラグは OFF にし，以降の分割積和演算においては，前の分割積和演算の結果を積算する必要がある為，最後の DSP ブロックの accumulate フラグを ON にする．尚，本構造は Poly-phase FIR フィルタ[5]等のハードウェア構成で用いられているものである．

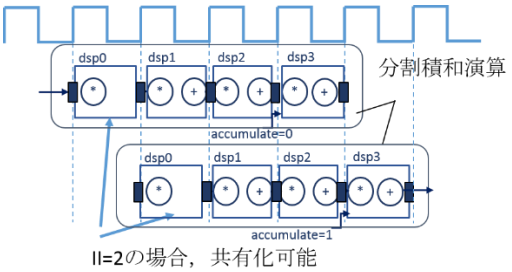


図 3　タイミングチャートを用いた，タップ数 8 の FIR フィルタ(II=2)のハードウェア構成に関する概念図

さて，図 2 の II=1 のパイプライン回路において，レイテンシは 10 サイクル，DSP ブロック使用数 8 である．一方，図 3 の II=2 のパイプライン回路では，レイテンシは 7 サイクル，DSP ブロック使用数は 4 となり，スループットは 1/2 に低下するが低レイテンシ・省面積で実現されることになる．このように，DSP ブロック内のアキュムレータ構造を利用することで，高い動作周波数を維持しつつ，レイテンシ・面積の短縮化が可能となる．

2.5 高位合成ツールにおける課題点

既存の高位合成ツールにおいては，動作合成可能な FIR フィルタ専用のライブラリが提供されており，図 2 または図 3 のハードウェア構造を実現し，要件 1,2 を満たす高性能なデザイン生成が可能であるが，以下が課題点として挙げられる．

・データ入力が最初の DSP ブロックに入力される(h[0]～h[7])為，定数値)ストリーム I/F のパイプライン構造となっている為，順序回路には対応できない．

・FIR フィルタ専用の為，一般的な積和演算（乗算両オペランドが変数の場合は考慮外）

2.6 一般的な DSP カスケード接続構造

より一般的な DSP カスケード接続構造として，図 4 の構造を考える．図 2 の構造と異なる点としては，全てのデータ x[0]～x[7]，y[0]～y[7]が入力端子となっており，また，各端子の入力タイミングが異なっている点である．本構造は各入力端子が異なるインターフェース(メモリ・FIFO 等)に異なるタイミングで接続することが可能である為，より柔軟な設計が可能である．また，積和演算を適切な単位で分割することで，図 3 で提示した，任意スループットのパイプライン回路に適用可能であるし，順序回路にも適用可能である．また FIR 専用ライブラリと異なり，両方のオペランドが変数であるケースを想定していることから，一般の積和演算に適用可能である為， モータ制御やニューラルネットワーク等，幅広いアプリケーションでの利用が可能となる．

本稿では，図 4 で示した一般的なハードウェア構造を実現する為，積和演算ライブラリを用いた高位合成フローを提案する．詳細は 3 章にて言及する．

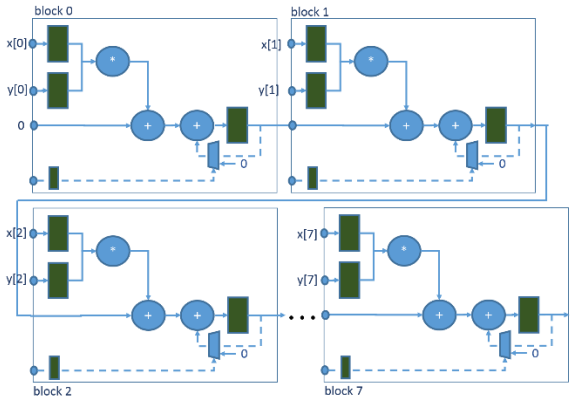


図 4 一般的な DSP カスケード接続構造

3. 提案フロー

本章では，積和演算ライブラリを用いた高位合成フローを提示する．図 5 に提案フローの全体図を示す．

以下では，主要構成部位である，積和演算ライブラリ，C/C++/SystemC パーサ，高位合成ツール，RTL 出力/論理合成環境生成ツールについて 3.1 節~3.4 節にてそれぞれ述べる．

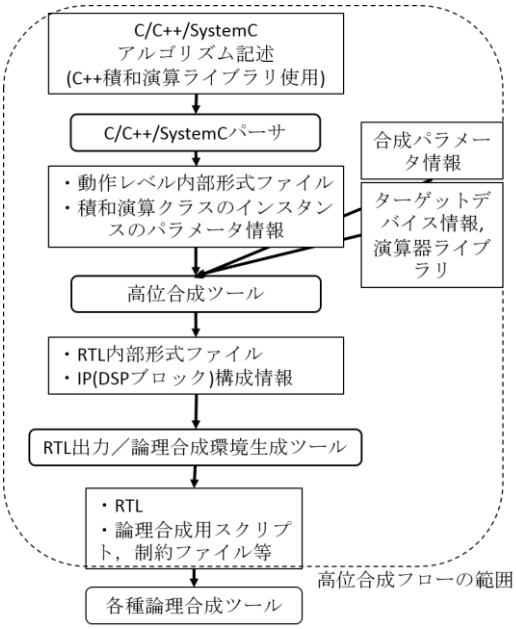


図 5 積和演算ライブラリを用いた高位合成フロー

3.1 積和演算ライブラリ

積和演算ライブラリは C++上で実現されるテンプレート・クラスである．以下の図 6 に，積和演算ライブラリの使用例(C++記述)を示す．本例では，2x4 の二次元配列 a と b の各配列要素の乗算結果を積算しており，II=2 のパイプライン回路を実現する．12~13 行目にあるように，積和演算クラスにテンプレート引数を指定することにより，本クラスをインスタンシエートしている．また 23 行目では，calc 関数を呼出すことにより，積和演算の実行が可能である．

```
1  //SystemCのデータ型を使用
2  #include <systemc.h>
3  //積和演算ライブラリのインクルード
4  #include <cwb_cpp.h>
5  /*Cyber folding=2 */ // II=2のパイプライン回路を生成
6  sc_fixed<19,4> filter() {
7    //積和演算クラスをインスタンシエートする
8    // 第一引数：積和演算に使用する配列の次元および要素情報
9    // 第二，第三引数：乗算の左，右オペランドのデータ型
10   // 第四引数：積和演算出力データ型
11   // 第五引数：分割積和演算の数(II=2以上で処理可能)
12   static cwb::cwb_mac<CWB_2D(2,4), sc_fixed<8,1>,
13                       sc_fixed<8,1>, sc_fixed<19,4>, 2> m;
14   ...
15   for (int i=0;i<2;i++) {
16     for (int j=0;j<4;j++) {
17       //mem_a[0]～mem_a[3]は別々のメモリ
18       a[i][j] = mem_a[j][adr_a];
19       //mem_b[0]～mem_b[3]は別々のメモリ
20       b[i][j] = mem_b[j][adr_b];
21     }
22   }
23   return m.calc(a, b);//積和演算 ΣΣa[i][j]*b[i][j]の実行
24 }
```

図 6　積和演算ライブラリ使用例

3.2 C/C++/SystemC パーサ

C/C++/SystemC パーサは，ユーザ記述を解析し，動作レベル内部形式ファイルの出力と同時に，インスタンシエートされた積和演算クラスのテンプレート・パラメータ情報を出力する．パラメータ情報には，配列要素数，次元数，演算データ型，分割積和演算の数(以後，分割数)が含まれる．

3.3 高位合成ツール

高位合成ツールは，C/C++/SystemC パーサが出力する動作レベル内部ファイルおよびパラメータ情報と，ユーザが合成時に指定するターゲット・デバイス情報(ライブラリ情報)や合成パラメータ情報を基に，回路を合成する．尚，パイプライン回路や順序回路といった回路種別の指定は，合成パラメータ情報で与えるものとする．

さて，積和演算処理を高位合成ツールにおいて合成可能とする為には，積和演算を複数の分割積和演算(→2.4 節)に変換する．具体的には，記述中の calc 関数呼出し部を，分割積和演算を行う関数(以下，分割積和演算関数)の呼出しに変換する．その後，高位合成フローの工程に従い，CDFG 作成，スケジューリング，バインディング等を実施していく[6][7]．以下では各工程について詳述する．

3.3.1 分割積和演算関数への変換

積和演算処理部である calc 関数呼出し部を変換し，分割数回の分割積和演算関数呼出しに変更する．分割積和演算関数の引数は，各分割積和演算に対応する乗算オペランドと accumulate フラグで構成される．積和演算結果は，最後の分割積和演算関数呼出しの戻り値である．

例えば図 6 の例では，分割数を 2 としている為，2 個の分割積和演算関数呼出しが生成される．1 回目の呼出しでは，前回呼出し時の分割積和演算結果を積算する為，accumulate フラグを OFF(=0)にし，2 回目の呼出しでは，前回呼出し時の分割積和演算結果を積算する為，accumulate フラグを ON(=1)にする．以上の変換結果を図 7 に示す．

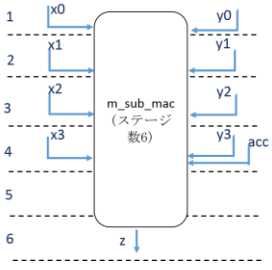
```
sum = m.calc(a, b)
      ↓
(void) m_sub_mac(a[0][0], b[0][0], a[0][1], b[0][1],
                 a[0][2], b[0][2], a[0][3], b[0][3], 0);
sum = m_sub_mac(a[1][0], b[1][0], a[1][1], b[0][1],
                 a[1][2], b[1][2], a[1][3], b[1][3], 1);
```

図 7 分割積和演算関数呼出しの生成

3.3.2 パイプライン演算器生成

分割積和演算はパイプライン動作可能であり，前節で生成した分割積和演算関数は多サイクル入力のパイプライン演算となる．高位合成ツールは，演算を，演算器ライブラリに登録してある演算器に割当てて合成する機構である為，スケジューリング前段階において分割積和演算関数に対応する多サイクル入力型パイプライン演算器を用意し，演算器ライブラリに追加しておく必要がある．以後，このパイプライン演算器を分割積和演算器と呼ぶことにする．

ここでは，この分割積和演算器を生成する過程について説明する．生成においては，積和演算クラスのインスタンスのテンプレート・パラメータ情報とターゲット・デバイス情報[2]を用いる．図 6 の記述を例に，分割積和演算器の生成方法を説明する．本記述では，2x4 の 2 次元配列同士の積和演算が定義されており，計 8 回の fma 演算が行われる．分割数を 2 と指定している為，分割積和演算関数呼出し 1 回当たり 4 回の fma 演算が行われる．合成に指定したターゲット・デバイスがサポートしている DSP ブロックが図 1 の構造である場合，DSP ブロックが 4 段カスケード接続されている構造に相当する為，図 8 のような 6 ステージの 9 入力パイプライン演算器が生成されることとなる．



2 高位合成時に指定するものである．図 5 を参照．

図 8 分割積和演算器例(ステージ数 6)

3.3.3 CDFG 生成・時間制約生成

時間制約は，演算や入出力が実行されるサイクルを限定する制御手段である．一般に高位合成におけるスケジューリング工程では，演算器数制約や時間制約等の制約に基づいて各演算や各入出力の実行タイミングが決定される[6].

さて，CDFG 生成工程では，分割積和演算器に相当するノードを生成し，ノード間には時間制約を設定する．2.4 節で言及したように，パイプライン回路時においては，分割積和演算間で同一のリソースを共有する為に，各分割積和演算の処理開始間隔を 1 サイクルとすることが必要である．一方，順序回路合成時においては，同様に各分割積和演算間に処理開始間隔を与えることは必要であるが，処理開始間隔は 1 サイクル以上となり，比較的緩い制約条件となる．これは，パイプライン回路の場合と異なり，順序回路の場合においては，任意の異なる状態間で共有が可能であるからである．図 9 に CDFG 生成例を示す.

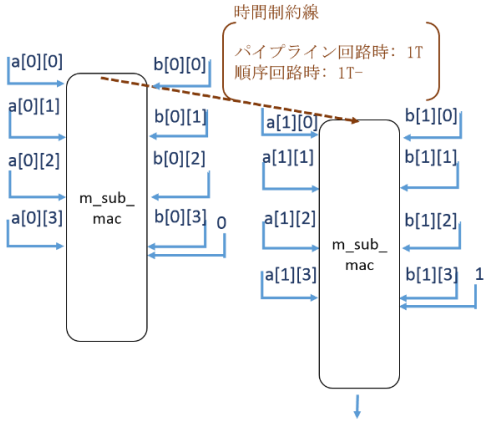


図 9 CDFG 生成例

3.3.4 スケジューリング

スケジューリングの工程においては，3.3.3 節で設定した時間制約に基づいてスケジューリングを行う.

図 9 の例のスケジューリング結果の一例を図 10 に示す.

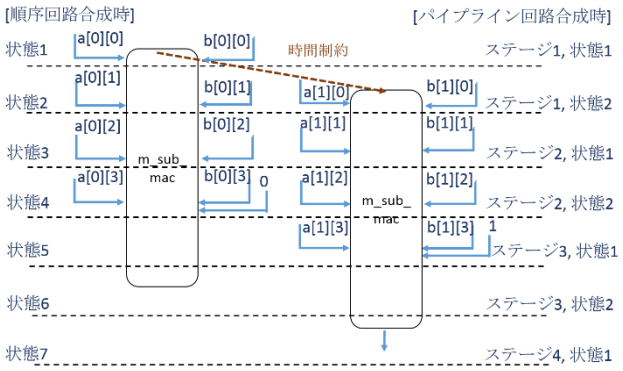


図 10 スケジューリング例

3.3.5 バインディング

バインディングは，入力記述上の演算子を演算器に，変数や配列をレジスタ・メモリにマッピングする工程である[6]. 具体的には，時間的・論理的排他性を考慮し，CDFG 上のノードを物理的な演算器やレジスタ, メモリと結びつける．ここでは CDFG 上の分割積和演算ノード(図 9)を分割演算器に結びつける．積和演算クラスの一つのインタンスにおける積和演算実行(calc 関数の実行)の結果として生じる各分割積和演算は，同一の DSP ブロックの実体に割り当てられることが必要である為，各分割積和演算ノードを同一の分割積和演算器に結びつけるようにする．

3.3.6 後処理

合成対象となった各分割積和演算は，バインディング工程の結果，各分割積和演算器との対応関係が決定されたが，最終的には各 FPGA ベンダーが提供する IP である DSP ブロックが適切にコンフィギュレーションされた状態にした上で，その DSP ブロックのカスケード構造を実現する必要があるし，また，分割演算器の各端子は，各 DSP ブロックの入出力データ端子または入力アキュムレータ端子に適切に接続されるように必要がある．

本工程では，後工程のツールに渡す為の情報として，DSP ブロック関連の情報(分割積和演算器の各端子と DSP ブロックの対応関係や DSP ブロックのコンフィギュレーション情報(パイプライン・レジスタの位置，演算器の使用有無など))を RTL 相当の内部形式情報出力処理において出力する．

表 1 は，図 8 で示した分割積和演算器の各端子と，実際に割り当てられる DSP ブロックのインスタンスとの対応関係の例を示したものである．

表 1 分割積和演算器の各端子と DSP ブロックのインスタンスとの対応関係

端子名	DSP ブロックインスタンス ID
x0, y0	1
x1,y1	2
x2,y2	3
x3, y3, accumulate,z	4

3.4 RTL 生成ツール/論理合成環境生成ツール

RTL 生成ツールは, 高位合成ツールが出力した RTL 相当の内部形式情報に含まれる，DSP ブロック関連情報(→3.3.6 節)を解析し，DSP ブロックのインスタンス化を行い，RTL を出力する．また，論理合成環境生成ツールも同様に，DSP ブロック関連情報を解析し，DSP ブロック IP のコンフィギュレーション情報を，論理合成用スクリプトと共に出力する．

4. 評価結果

本章では，前章で提示した高位合成フローを高位合成ツール CyberWorkBench®上に構築した際の評価結果を報告する．

4.1 FIR フィルタ

本節では，32 タップの FIR フィルタへの適用事例を報告する．合成回路は II=2 のパイプライン回路であり，32 タップ

のフィルタが 2 つの単位に分割され，それぞれ 1 サイクルずつずらして処理が行われる．本試行においては，Intel 社のデバイス ArriaV を使用しているが，ArriaV の DSP アーキテクチャ[8]は，DSP1 個あたり 2 回の積和演算実行が可能な為，表 2 に示される通り，使用 DSP 数は 8 となる．図 11 に論理合成後に得られた DSP カスケード構造図を示す．尚，使用した論理合成ツールは Quartus Prime Design Software 17.0.0 である．

表 2　32 タップ FIR フィルタ回路の性能指標

Latency	Stages	II	ALMs	Regs	DSPs	Fmax (MHz)
21	11	2	201	100	8	265.25

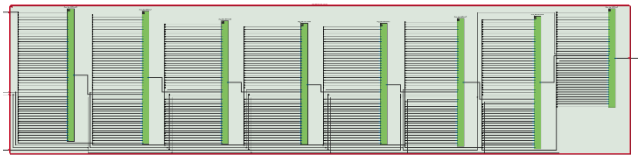


図 11 実現された DSP カスケード構造図

4.2 QR 分解

QR 分解(householder 変換使用)は，ノルム計算や行列積計算が行われる等，内積演算が多く使用されるアルゴリズムである．表 3 は，本積和演算ライブラリを使用し，QR 分解アルゴリズム(C 記述)を高位合成した結果，生成されたデザイン(順序回路)の性能指標を示すものである．本例では，QR 分解対象となる行列のサイズは 5x3, 乗算のデータ型は符号付き固定小数点型(整数部 10 ビット，小数部 8 ビット)とした．使用した論理合成ツールは Quartus Prime Design Software 17.0.0，ターゲット・デバイスは Arria10 である．

表 3　積和演算ライブラリ使用時の性能指標(QR 分解)

Design	Latency	ALMs	Regs	DSPs	Fmax (MHz)	Performance
1-1	252	10,649	7,998	47	**203.54**	1.238085
2-1	252	10,672	8,173	59	**198.65**	1.268562
3-1	252	10,523	8,633	68	**196.46**	1.282703
4-1	252	10,355	9,008	71	**198.93**	1.266777
5-1	252	10,939	9,908	80	**198.14**	1.271828

比較対照として，積和演算ライブラリを使用せず，積和演算部をそのまま高位合成して得られたデザインの性能指標を表 4 に示す．この場合，CyberWorkBench®において演算ツリーをバランス化する機能が働く為，生成デザインのレイテンシが小さくなる傾向がある．一方，DSP のカスケード構造は実現されない為，CLB と DSP ブロック間の配線遅延がボトルネックとなる傾向となり，本ライブラリ使用時と比べると，十分な Fmax が得られない．トータルの性能比較としては積和演算ライブラリ使用時の方が，未使用時と比べて平均 7.3%改善する結果となった．

表 4　積和演算ライブラリ未使用時の性能指標(QR 分解)

Design	Latency	ALMs	Regs	DSPs	Fmax (MHz)	Performance
2-1	240	9,242	8,232	48	181.29	1.323845
2-2	240	9,235	8,080	57	173.25	1.385281
2-3	240	10,078	8,465	79	179.6	1.336302
2-4	240	9,721	8,062	87	172.27	1.393161
2-5	240	9,896	8,062	102	172.83	1.388647

5. 結論

本稿では，積和演算ライブラリを用いた高位合成フローを提案した．本ライブラリは，FIR フィルタ専用のライブラリの一般形に位置づけられ，本フローはより汎用性の高いアルゴリズムの合成において有効とある．本フローは，順序回路・任意スループットのパイプライン回路いずれにおいても使用可能であり，またライブラリ使用時に与えるパラメータにより，スループット・面積・レイテンシ制御が可能であることから，システムの要求性能に対して柔軟に対応できる．積和演算処理は，FPGA のハードマクロである DSP ブロックの性能を最大限に生かす形で実現される為，本フローにおいては，高位合成ツールはより高性能なデザインを出力することが可能となる．

本提案フローを高位合成ツール CyberWorkBench®上に構築し，従来と比べて高性能なデザインが得られることを示した．

謝辞　本研究の一部は，国立研究開発法人　新エネルギー・産業技術総合開発機構（NEDO）の委託業務の結果，得られたものです．

参考文献

1) Takao Nishitani, Yuichi Kawakami, Rikio Maruta and Akira Sawai, “LSI Signal Processor Development for Communications Equipment”, IEEE Proc. of ICASSP, pp.386-389, (1980).
2) Using FPGAs to solve tough DSP design challenges https://www.eetimes.com/document.asp?doc_id=1279776
3) Cyclone V Device Handbook, https://www.altera.co.jp/ja_JP/pdfs/literature/hb/cyclone-v/cv_5v2.pdf
4) A 1D Systolic FIR, http://www.iro.umontreal.ca/~aboulham/F6221/Xilinx%20A%201D%20systolic%20FIR.htm
5) Polyphase Filters, http://www.ws.binghamton.edu/fowler/fowler%20personal%20page/EE521_files/IV-05%20Polyphase%20FIlters%20Revised.pdf
6) 若林一敏：ソフトウェアプログラムからハードウェア記述を合成する高位合成技術，電子情報通信学会 基礎・境界ソサイティ，Vol6, No.1, pp.37-50 (2012).
7) Kazutoshi Wakabayashi, Cyber: High-Level Synthesis Systems from Software into ASIC, Kluwer Academic Publishers, pp. 127-151, (1991).
8) Arria V Device Handbook, https://www.altera.com/en_US/pdfs/literature/hb/arria-v/av_5v2.pdf
9) CyberWorkBench®, https://www.nec.com/en/global/prod/cwb