3L－2

高位合成ツール Archist (2)
--- パイプラインアーキテクチャ合成 ---

東田基樹　大村昌彦　石川淳士
三菱電機(株) システムLSI開発研究所

1.はじめに

我々は、高性能 DSP 向けの高位合成ツール Archist を開発している[1]。ほとんどの DSP アルゴリズムの動作記述には、ループ記述が含まれる。画像処理用 DSP 等の高性能 DSP に要求される性能を実現するには、このループの内部動作をパイプラン実行させ(ループパイプライニング)、高いスループットをもつアーキテクチャを合成する必要がある。本稿では、Archist の特徴と、ループパイプライニングを行なう上での問題点とその解決法を述べる。

2.ループパイプライニング

次のような for ループを含む動作記述を考える。

```
acc = 0 ;
for( i = 0 ; i < N ; i = i+1 ) begin
    tmp1 = f(i) ;
    tmp2 = g(i) ;
    tmp3 = tmp1 * tmp2 ;
    acc = tmp3 + i + acc ;
end
```

図 1 が、この動作記述に対応するコントロール/データフローグラフ (CDG) である。通常の高位合成ツールはこの CDG に基づいて処理を行なう[2]。すなわち、条件判断とループの内部動作の処理を、異なった制御ステップで実行し、これをループさせる。従って、仮に、条件判断に 1 ステップ、ループの内部動作に 5 ステップ必要であるとすれば、ループの全動作に必要なステップ数は、6N+1 となる（比較演算は、N+1 回動作することに注意)。

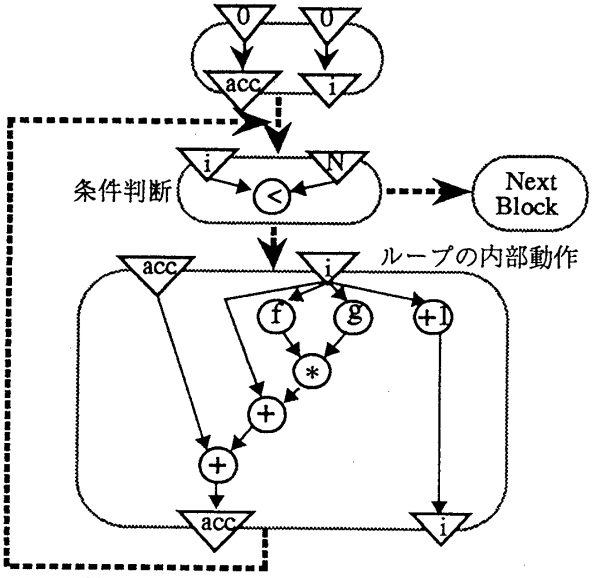

図1 コントロールデータフローグラフ

図 1 の動作記述をより高速に動作させるアーキテクチャを合成する高位合成手法がループパイプライニング[3]である。図 2 に、図 1 の CDG に対してループパイプライニングを行なった時のスケジューリング例を示す。但し、関数 f をパイプライン 演算器 (3.1(2) 参照)F(1),F(2),F(3) で、関数 g,*,<,+1 の演算及び 2 つの加算を、それぞれ演算器　G, MUL, LSS, INC, ADD1, ADD2 で実現している。図 2 から分かるように、ループパイプラインニングでは、条件判断とループの内部動作の実行を、前のループの処理が終るのを待たずにパイプライン状に実行させる。

ループパイプライニングを行なった時、ループの終了までのステップ数は N+5 である。N が十分に大きければ、6 倍の高速化が図れたことになる。

図 2 のスケジューリングにより合成されたパイプラインアーキテクチャを図 3 に示す。

3.高位合成ツール Archist

3.1 パイプラインアーキテクチャ合成機能の特徴

Archist のループパイプライニング機能における特徴を以下に示す。

(1) 1 クロック中に複数のオペレーションを動作させるチェーンニングが可能（例：図 1 のカスケード接続した 2 つの加算を図 2 のように単一ステップにスケジューリングし、図 3 の 2 つの加算器のようにレジスタを介さない構成のデータパスを合成する)。

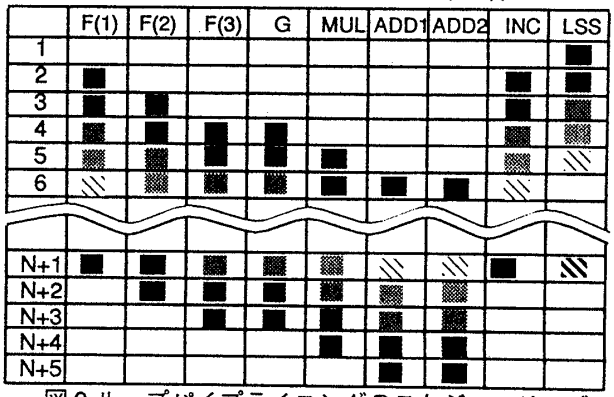

図2 ループパイプライニングのスケジューリング

High-Level Synthesis Tool, Archist (2)
-- Pipeline Architecture Synthesis --
Motoki Higashida, Masahiko Ohmura and Junji Ishikawa
Mitsubishi Electric Corporation, System LSI Laboratory
4-1 Mizuhara, Itami Hyogo 664, Japan

(2) 1クロック周期以上の遅延をもつ演算器を、1クロック周期以下の遅延となるブロックに分割し、パイプラインラッチを介して接続したパイプライン演算器の利用が可能。(例:関数 f に対応する図 3 の演算器 F(1),F(2),F(3))。
(3) 複数箇所でループパイプライニングを行なった場合、演算器の共有化が可能。
(4) ユーザ定義モジュールの使用が可能。
(5) パイプラインピッチ (隣あったループ動作の開始間隔) は 1 に限定。

3.2 パイプラインレジスタ数の最小化

ループパイプライニングを行なった場合、1 つのレジスタが 2 以上の制御ステップに渡って値を保持することはできない。もし、2 以上の制御ステップに渡って値を保持する必要がある場合、図 3 の i ～ i(4) のレジスタ群のように、パイプラインレジスタを構成する必要がある。

このため、ループパイプライニングを行なった場合の特別な問題として、パイプラインレジスタ数の最小化問題が発生する。例えば、図 2 では G のスケジューリングを 4 ステップ目に設定しているが、この例では、G のスケジューリングは 2 ～ 4 の間で自由に設定できる。しかし、もし 2 ステップ目に設定すれば、G の値が使用されるのは 5 ステップ目なので、2 ～ 5 ステップの間、演算結果の値を保持する必要が生ずる。従って、この場合、図 3 のレジスタ tmp2 を 3 段のパイプラインレジスタにしなければならず、2 段分余分のレジスタが必要となる。

Archist では、パイプラインレジスタの個数が少なくなるようにスケジューリングを行なっている。

3.3 複数のパイプラインブロックによる演算器の共有化

多機能の DSP を合成しようとした時、複数の独立なループに対してパイプライニングを行なわせたい場合がよくある。各パイプラインブロック内では、オペレーション毎に個別の演算器を割り当てる必要があり、演算器の共有はできない。しかし、異なったパイプラインブロックの演算器の共有は可能である。

この時、できるだけ転送路が単純になるように演算器を共有化する必要がある。例えば、他のパイプラインブロックにて、acc2 = acc2 + tmp4 のような演算があったと仮定し、これを図 3 のデータパス上の演算器と共有することを考える。図 3 には加算器が 2 つ (ADD1,ADD2) あり、そのどちらを共有しても構わない。しかし、変数 acc2 をレジスタ acc に割り付け、加算を ADD2 に割り付けた方が、データパス系が単純になることが推測できる。

Archist では、演算器とレジスタの接続関係を考慮して、データパスが単純になるようリソースバインディングを行なっている。

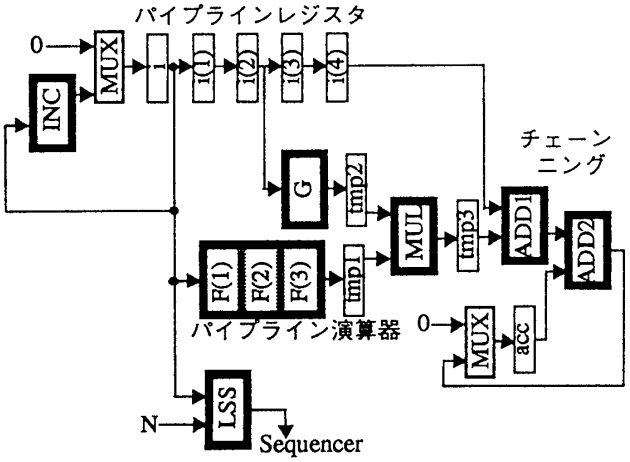

図 3 合成されたパイプラインアーキテクチャ

3.4 ユーザ定義演算器の採用

Archist では、ユーザ定義演算器の使用が可能である。具体的には、図 1 の例にあるように、ユーザが設計した演算器の機能を関数として表現し、動作記述で関数呼び出しを行なうことにより使用できる。この時、Archist は、この関数をプリミティブな演算子と解釈して合成を行ない、演算子に対応する演算器としてユーザの設計した演算器を用いる。

ループパイプライニングの動作部は、アーキテクチャの良し悪しを決定づける部分であることが多い。Archist では、このような部分に対するデータパスについては、ユーザが最適な設計を行ない、そのデータパスを用いて Archist が全体のアーキテクチャ設計を行なうといった思想をとっている。

また、複数のパイプラインブロックがある場合において、演算器の共有化を制御する目的にもこの機能は有用である。この場合、共有化させたいデータパスをユーザ定義演算器としておき、それぞれのループ中で、同じ関数を呼び出すようにする。

4. おわりに

本稿では、高位合成ツール Archist のパイプラインアーキテクチャ合成機能について紹介した。特に、ループパイプライングを行なった時に問題となる、パイプラインレジスタの最小化と、複数のパイプラインブロックがあった場合の、共有化の手法について述べた。

[参考文献]

[1] 大村他、「高位合成ツール Archist(1)-- ユーザインタフェース --」 本予稿集 3L-02 (1994-9).
[2] D.Gajski, et al., HIGH-LEVEL SYNTHSIS, Kluwer Academic Publishers,(1992).
[3] C.Hwang, et al."A Scheduler for Pipeline Synthesis", IEEE trans. on CAD Vol.12, No.9, (1993-9).