A-08

動的再構成デバイス用の回路自動生成の一手法

A Circuit Synthesis for Dynamic Reconfigurable Device

荒木 統行†　　神戸 尚志†

Nobuyuki Araki　　Takashi Kambe

1. はじめに

近年、チップ内部の構成を動作中に変更することのできる動的再構成可能プロセッサは、DAPDNA2、DRP など多くの半導体メーカ、大学の研究機関から提案・発表されている。動的再構成可能プロセッサは、必要に応じて動作を変更することが可能であるので、その応用について注目されている。

しかし、動的再構成プロセッサは従来のプロセッサとは異なり、PE(Processing Element)を用いた粗粒度構成をとる場合が多いため、PE を効率よく用いることができる専用の設計環境が重要である。ベンダーからは、GUI を用いて対話的に設計するツール、専用言語による記述、C 言語から回路合成などのツールが提供されているが、制約が多く、PE についての詳細な知識が必要とするなどの問題もある。本文では動的再構成プロセッサ DAPDNA2 を対象に、高級プログラミング言語を入力とし、for、while、if などの制御文を効率的に PE に割りつけることにより、PE 単位のパイプライン回路を自動合成するアルゴリズムについて提案する。

2. DAPDNA-2

DAPDNA-2 は主にシーケンシャルな処理を担当する 32 ビット RISC プロセッサコア DAP(Digital Application Processor)と、2 次元アレイ構造で再構成可能な部分に当たる DNA(Distributed Network Architecture)からなる。

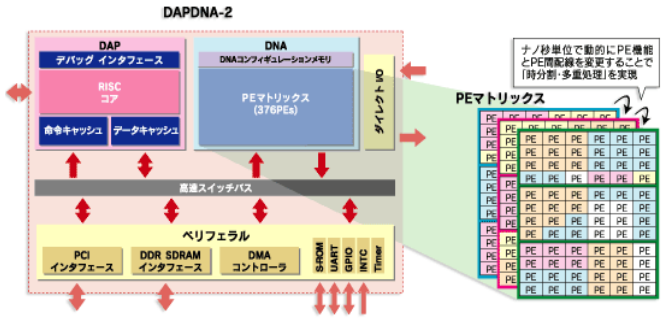


図 1　DAPDNA-2 のブロック図

32 ビット RISC プロセッサである DAP の動作周波数は、この LSI のほかの回路ブロックと同じ 166MHz であり、DNA に対してマスターとなる。用途としては、一般的な汎用プロセッサとしての利用や、PE マトリックスの動的再構成の制御などに使用する。

DNA は PE によるマトリックス構造をとる。PE マトリックスは、各 PE の動作設定と PE 間の接続設定によって定義され、これを 1 面の DNA コンフィギュレーションとしてプログラムされる。DNA コンフィギュレーションは実行バンクと 3 つのバックグランドバンクに格納されており、DNA コンフィギュレーションを動的に切り替えることにより、複雑な処理や大規模な回路を実現する。

DAPDNA-2 には、異なる処理に対応できるようにいくつかの種類の PE が用意されている。PE の一覧を表 1 に示す。

表 1　PE 一覧

PE	機能
EXE	各種演算
Delay	遅延
RAM	内部メモリ
カウンタ	カウンタとして動作
LDB/LDX	DNAへの入力部
STB/STX	DNA外部への出力部

データ処理用の PE には EXE、Delay、RAM などがあるが、ここでは演算やパイプライン化を行う上で重要な役割を持つ EXE エレメントについて述べる。EXE エレメントの内部構造を図 2 に示す。

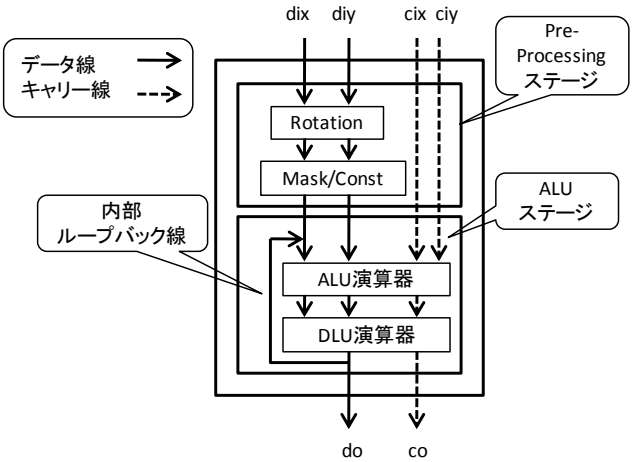


図 2　EXE エレメントの内部構造

DAPDNA-2 では処理値を表す 32bit の「データ」と、主に条件判断の結果に利用され PE の動作を変える 2bit の「キャリー」を扱う。EXE エレメントは Pre-Processing ステージと ALU ステージの 2 ステージから構成されており、各

†近畿大学　理工学部　電気電子工学科

演算器はパイプライン構造を持ち、内部ループバック線は ALU ステージの出力結果を再び ALU ステージに入力し、次のサイクルの演算で使用可能になり、ループ内でのデータの依存関係がある場合に使用される。PE 単位のパイプライン化は、PE 数は殆ど増加させずに処理の高速化を実現でき、低消費電力化にも有効である。一方、演算器によって実行可能な演算が異なり、一つのコンフィグレーション中の PE 数に制限があるため、演算の実装方法によって処理性能と使用リソースが大きく変化する。このため、演算の PE への割り付けに工夫が必要である。

3. 回路生成アルゴリズム

C 言語で記述されたソフトウェアアルゴリズムを動的再構成可能プロセッサで実現するためには、制御文の取り扱いが最も重要である。回路生成アルゴリズムの全体フローを図 3 に示す。本文では主にループ文と条件文に対する DNA コンフィギュレーションの自動生成法について述べる。

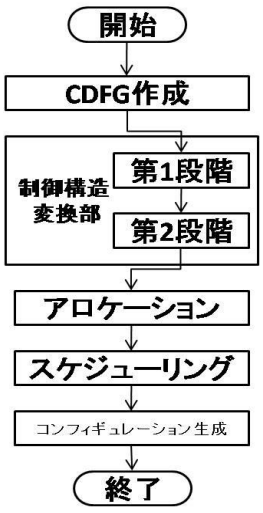


図 3　全体フロー

(第 1 段階) ループ文の変換

まず、ループ文に対して、パイプライン可能かどうかの判定を行う。次に、判定結果に基づいて、Bach C-CDFG(以降、CDFG とする)から D2-CDFG へ変換する。

(第 2 段階) 条件文の変換

条件式の比較演算に対して、投機的実行を行えるかどうかの判定を行う。次に、投機的実行が行えていない場合は投機的実行可能な CDFG に変換し、条件文に当たる比較演算・処理を D2-CDFG に変換する。

3.1　コントロールデータフローグラフ

本アルゴリズムは、2 種類のコントロールデータフローグラフ「CDFG」及び「D2-CDFG」を用いる。

CDFG は、Bach C 高位合成システムが生成した CDFG であり、「ノード」は演算処理を表し、ノード ID、親ノード ID、ノードの種類、ポート情報などを持つ。ポート情報には、ポート ID、ポートの型・幅、接続するノード ID・ポート ID などがある。ノードはその種類から 33 種類の「演算ノード」、7 種類の「ブロックノード」の 2 つに分類される (表 2 参照)。「演算ノード」とは、入力されたデータに対し加算や減算などの演算を行うノードである。また、定数値の生成には、Const ノードを用い、CtrlStart ポートに制御信号を入力することで定数値を出力する。

「ブロックノード」とは、複数ノードの集合であり、「ブロックノード」を用いることで演算ノードに階層を表現する。例えば、ループ文の演算部などを「ブロックノード」であらわす。

表 2　CDFG のノード一覧

演算ノード	ブロックノード
Const (定数)	body (演算ノードのまとまり)
＋ (加算)	Par(並列処理)
＜ (比較)	Branch (条件分岐)
・ ・ ・	CTO-B (Constant-Time-Operation) (条件分岐をまとめる際に取り扱う)
	Loop (ループ)
LAr-W (配列の書き込み)	Continue (Continue)
LAr-R (配列の読み込み)	Exit (ループの終了)

表 3 には、ノードの持つポート情報の一覧を示す。ポートは、その種類によってデータの入出力、制御の開始・終了などを行う。

表 3　ポート一覧

表示	ポートの種類	機能
○	DataIn	ノードへのデータ入力
	DataOut	ノードからのデータ出力
●	CtrlStart	ノードへの制御開始
	CtrlFinish	ノードへの制御終了
□	DataCont	ループノードでデータの折り返し
■	CtrlCont	ループノードで制御の折り返し
▲	Selector	分岐制御の選択

図 4 に簡単なソースコードに対する CDFG の例を示し、図 5 に図 4 のノード ID:2 の内部情報を示す。

図 4 の CDFG において、各ノードの上辺にある○印は DataIn ポート、下辺にある○印は DataOut ポートである。また、Const ノードにある●印は、ノードの制御を行う CtrlStart ポートである。

また、D2-CDFG とは、CDFG を DAPDNA-2 用に変換した CDFG である。このグラフのノードは、DAPDNA-2 の演算器に対応し、制御とデータの流れを表現したグラフである。D2-CDFG では、データの流れを実線エッジとキャリーの流れを点線エッジで表現する。

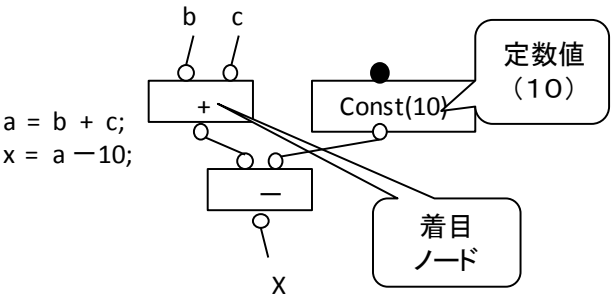


図 4　CDFG 例

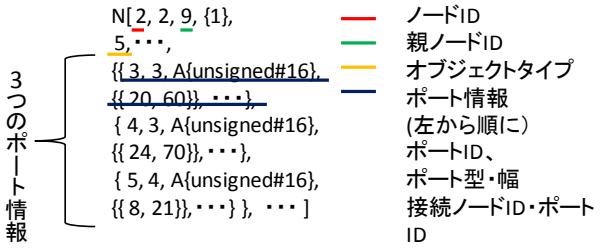


図 5　ノード情報例

3.1.1　ループ文の CDFG 表現

Bach C システムは、ループ文記述から例えば図 6 に示す CDFG を生成する。

ループ文の CDFG は、Loop ノード、Branch ノード、Exit ノード、body ノードの 4 つのブロックノードで構成される。Loop ノードは、DataCont ポートを持ち、このポートは i 回目の処理結果を i+1 回目の処理(Loop ノードの DataIn ポート)にデータを受け渡す。Branch ノードは、Selector ポートを持ち、このポートに入力される値(0,1)により、Exit、body ノードのどちらに遷移するかを表す。Exit ノードは、ループの終了を表し、ループを終了した際、次の処理に必要なデータを受け渡す。body ノードは、ループ文内の処理を行うブロックノードであり、body ノード内にはループ文内の演算処理を表す演算ノードがある。

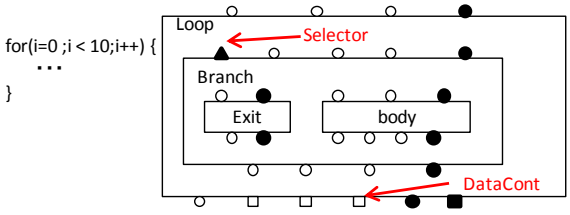


図 6　ループ文に対しての CDFG 構成

また、2 重以上のループ文の CDFG は body ノード内に入れ子となるループ文の CDFG が作成される。

3.1.2　条件文の CDFG

Bach C システムでは、条件文記述は図 7 の例に示す CDFG が生成される。

条件文の CDFG は、Sel ノードと演算ノードから構成される。図 7 の Sel ノードにある左端の DataIn ポート(Carry-DataIn ポートと呼ぶ)の入力値は 0,1 のみである。Carry-DataIn に 1 が入力されると、右側 DataIn ポート(True-DataIn ポートと呼ぶ)の値が DataOut ポートに出力され、0 が入力されると、中央の DataIn ポート (False-DataIn ポートと呼ぶ)の値が出力される。図 7 では条件式が真であれば、y + x の値が出力され、偽であれば、y − x の値が出力される。

一般に、分岐先の処理をあらかじめ実行し、条件によって値を変化させる実行を「投機実行」と呼ぶ。条件式や処理内容が複雑な場合は Sel ノードを複数使用し、投機実行を行わない CDFG が生成される(図 8, 9)。

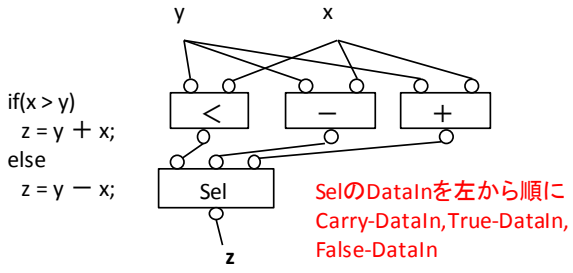


図 7　条件文の CDFG 例

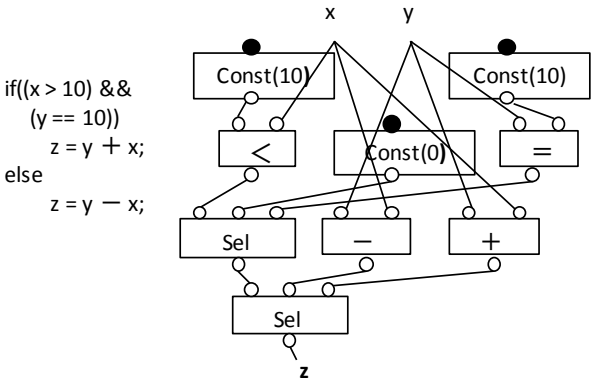


図 8　複数の Sel ノードで構成される CDFG

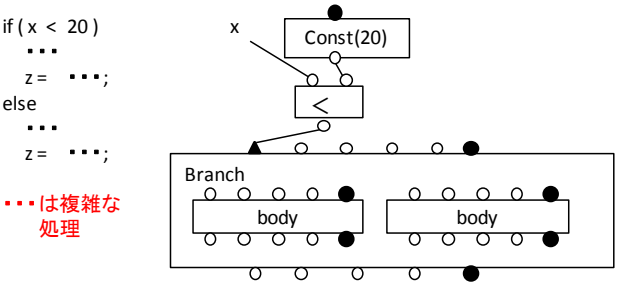
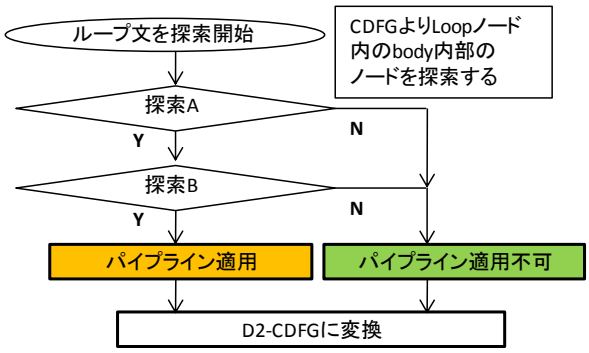


図 9　投機実行を行わない CDFG

3.2　ループパイプライン化判定

DAPDNA のパイプライン化を行うためにループ文内の処理内容についてパイプライン可能かどうかの判定を行う。図 10 にパイプライン化適用判定の処理フローを示す。



探索A・・・ループ文が定数ループかどうかの判定。
探索B・・・ループ文処理が i 回目から i + 1 回目にデータの受け渡しが全て内部ループバック適用どうかの判定。

図 10　パイプライン化判定フロー

3.2.1　探索 A

DAPDNA のパイプライン構造は各 PE において入力されたデータ数だけ演算を行うデータフロー型なので、パイプライン処理可能なのは、ループ回数 X が定数回であり、「定数ループ」と呼ぶ。また、パイプライン処理が出来ないループとは、ループ回数が記述よりループ回数 X が求めることができないループであり、「不定ループ」と呼ぶ(表 4)。

表 4　定数・不定ループの定義

	定数ループ	不定ループ

記述例	i = a; //初期値(a = 0,･･･,m)
	j = b; //初期値(b = 0,･･･,m)
	while(i < j) {
	･･･
	i += p; //p = 1,･･･, n)
	j += q; //q = 0, 1,･･･, n − 1) }
		i = a; //初期値(a = 0,･･･,m)
		j = b; //初期値(b = 0,･･･,m)
		while(i < j) {
		･･･
		i += data[i]; // i の増加が不定
		j += q; //q = 0, 1,･･･, n) }

ループ回数 X(回)	i, j の値が一定値で変化するのでループ回数Xが求めることができる。 $a + pX \geq b + qX$ $X \geq \frac{b-a}{p-q}$(回)	i の値の変化がループ処理回数によって値が不定であるのでループ回数は求めることができない。

3.2.2　探索 B

DAPDNA でのパイプライン処理を行うためには、i 回目の処理結果が i + 1 回目の処理内容に必要なデータの演算ノードが全て内部ループバック処理に適用可能かどうかを判定する。内部ループバック処理とは、i 回目の EXE エレメントの ALU ステージの出力結果を、再び ALU ステージに入力して、i + 1 回目の演算での使用を可能にする処理である。

演算ノードが内部ループバック処理を適用すれば、その演算ノードに関して 1 つの PE でループ文処理を形成することが可能である(図 11)。ループパイプライン化を適用するため、i 回目の処理結果が i + 1 回目の処理内容に必要な演算ノードが全て内部ループバック処理が適用可能か判定する。

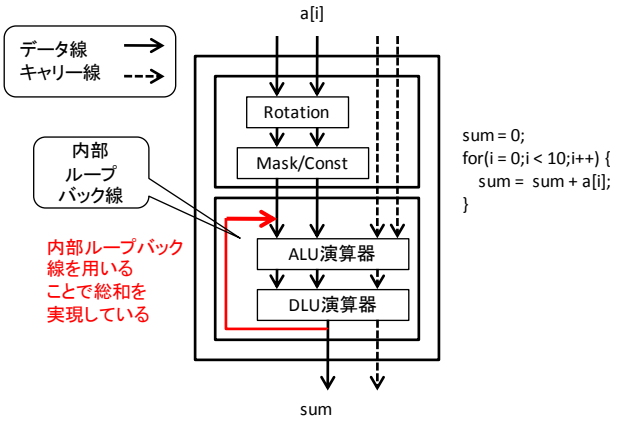


図 11　内部ループバックを用いた処理

この判定は CDFG において loop ノード内の body ノードに着目し、body ノードの DataOut ポートに接続される演算ノードは次のループ処理に必要な演算ノードのみが接続されていることが内部ループバックによるパイプライン化の適用条件である。接続されている演算ノードが全て内部ループバック可能であれば、DAPDNA でパイプライン適用可能だからである。接続される演算ノードのうち、ループ文の終了を行うために条件式の演算ノード(< 、== などであり条件式ノードとする)、配列にデータを書き込むための配列ノード(LAr-W)は、探索を行う必要がない。これは、パイプライン適用には関係のない演算ノードだからである。

図 12 に探索 B で行う判定処理フローを示す。

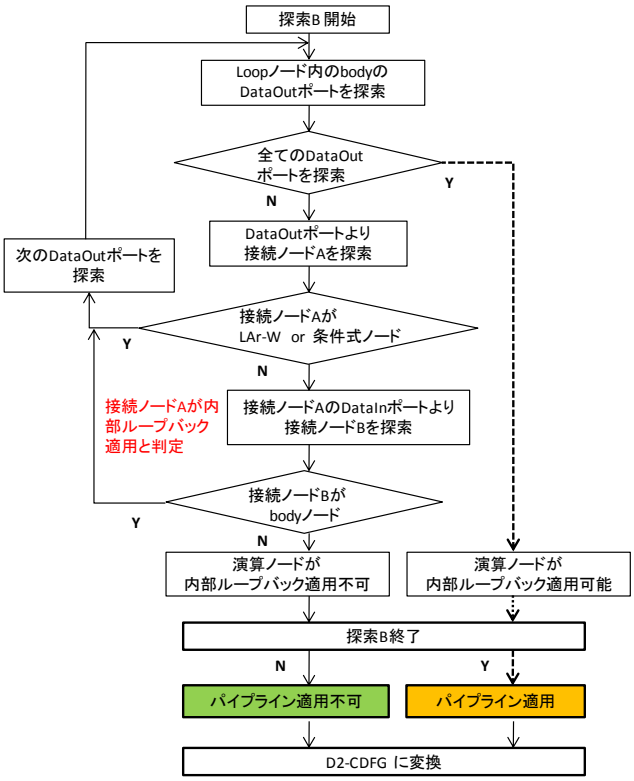


図 12　探索 B 判定処理フロー

図 12 に示すように、探索する演算ノード A が内部ループバック可能と判定するには、探索ノードの DataIn の接続ノード B が body ノードの DataIn ポートと接続されているか判定を行う。接続される場合、探索ノードは DAPDNA の EXE の演算において内部ループバックを用いた処理が適用と判定する。

逆に、DataIn ポートが body ノードの DataIn ポートに接続されない場合、その演算ノード A は内部ループバック処理が適用できないと判定し、不定ループの場合と同様な D2-CDFG に変換する。また、body の DataOut ポートより全ての演算ノードを探索するので、1 つでも演算ノードが適用できない場合はパイプライン処理が行えないので、その時点で探索 B は終了する。

図 13 に示す CDFG は、body の DataOut に接続されるノードのうち条件式ノード、配列ノード以外の演算ノード(i, sum)が内部ループバック適用であるのでパイプライン適用と判定できる。また、図 13 は簡単のため body 内の演算ノードの一部と Exit ノードを省略している。

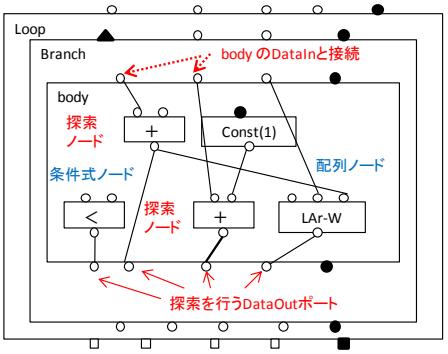
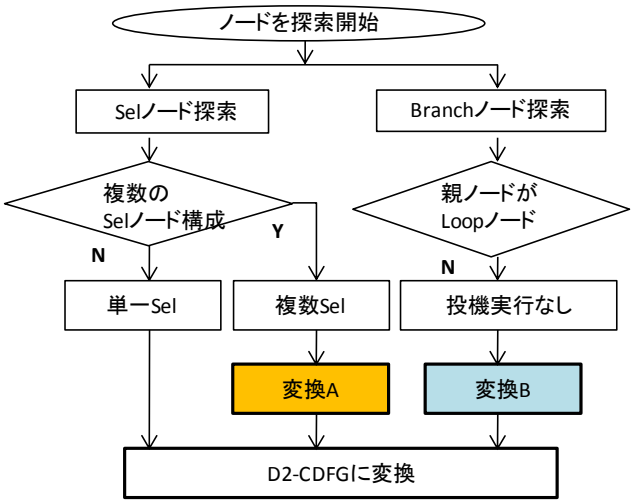


図 13　パイプライン処理可能 CDFG

3.3　制御構造変換の第 2 段階（条件判断）

第 2 段階では第 1 段階で変換したループ文の CDFG を変換した D2-CDFG 部を含む CDFG に対し、図 7～9 に示した条件式文の CDFG を D2-CDFG に変換する。このアルゴリズムを図 14 に示す。



探索A･･･　Selノードを接続関係よりタイプ分けを行う。
探索B･･･　投機実行を行えるように変換を行う。

図 14　条件式変換判定フローチャート

3.3.1　単一 Sel ノード

「単一 Sel ノード」とは、Sel ノード 1 つで条件式文を表す場合を言う。

単一 Sel ノードの場合、Carry-DataIn の入力値(0,1)の値により DataOut の出力値は True-DataIn・False-DataIn の入力値の片方が出力されるため、例えば、図 7 の Sel ノードの Carry-DataIn は、DAPDNA-2 の EXE エレメントより条件判断の結果に利用され PE の動作を変える「キャリー」に相当し、True-DataIn・False-DataIn は、「データ」に相当するので、D2-CDFG に変換する際、単一 Sel ノードは図 15 の例のように D2-CDFG に変換できる。

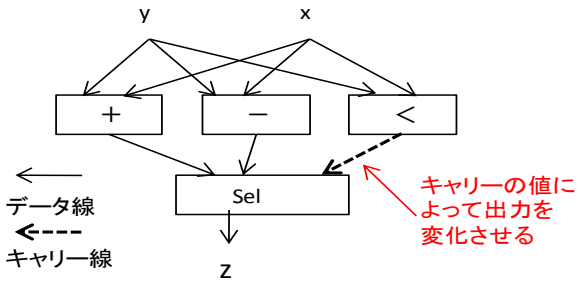


図 15　単一 Sel ノード処理の D2-CDFG

3.3.2　複数 Sel ノード(変換 A)

「複数 Sel ノード」とは、Sel ノードの DataIn もしくは DataOut ポートに他の Sel ノードが接続される場合である。図 8 に示す CDFG は複数 Sel ノードを持つ。

図 16 に示す変換アルゴリズムによって、着目する Sel ノードを D2-CDFG に変換する。まず、各 Sel ノードに対して「Data-Sel 型」、「Array-Sel 型」、「And-Sel 型」、「Or-Sel 型」の 4 つの型に場合分けを行う。次に、「Array-Sel 型」のみを「Data-Sel 型」に変換する。

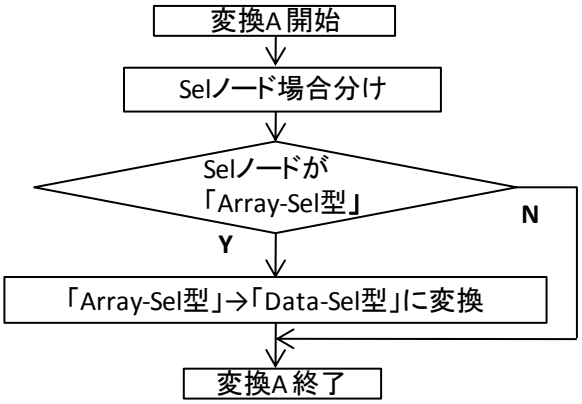
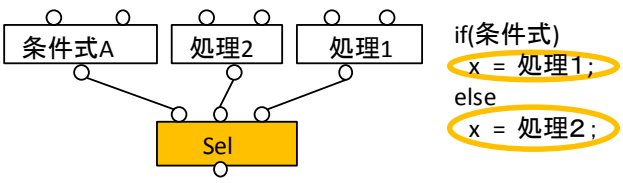


図 16　変換 A 処理フローチャート

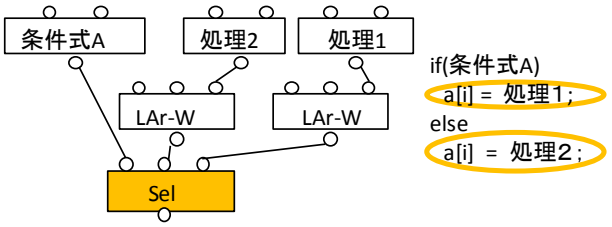
3.3.2.1　Sel ノードの場合分け

まず、Sel ノードに対して「Data-Sel 型」、「Array-Sel 型」、「And-Sel 型」、「Or-Sel 型」の 4 つの型に場合分けを行う。この場合分けを行うことにより、D2-CDFG に変換する際、着目する Sel ノードの出力とそれに接続されるノードがデータ線で接続するかキャリー線で接続されるかを判別する。

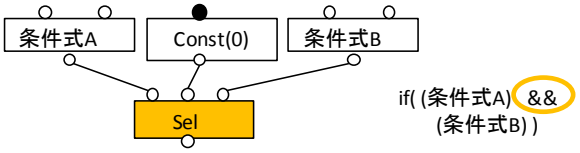
この判別方法は、着目する Sel ノードの DataIn・DataOut ポートの接続ノード・ポートを探索し、各接続関係により 4 つの型に場合分けを行う（図 17）。



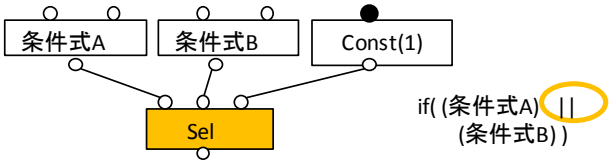
(a) Data-Sel 型



(b) Array-Sel 型



(c) And-Sel 型



(d) Or-Sel 型

図 17　Sel ノードの接続関係の分類

また、C 言語記述が複数の else if 、または条件式が複雑な場合、処理 1, 2、条件式 A,B のノードさらに Sel ノードが接続され、図 17 の分類ができない場合があるが(図 18)、着目する Sel ノードの DataOut ポートが必ず Sel ノードに接続されるので、接続されるノードの DataIn ポートが Carry-DataIn に接続されているかの判定を行うことで Sel ノードの場合分けを行うことが出来る。Carry-DataIn ポートに接続される場合、Sel ノードは And・Or-Sel 型の 2 つに絞られ、接続関係より Const(0)と接続されている場合は Sel ノードが And-Sel 型と判定出来る。逆に、True-DataIn・False-DataIn ポートに接続される場合は Data・Array-Sel 型の 2 つに絞られ、着目する Sel ノードの True-DataIn,False-DataIn に接続されるノードに LAr-W ノードが接続される場合は、Array-Sel 型と判定でき、その他の演算ノードの場合は Data-Sel 型と判定できる。

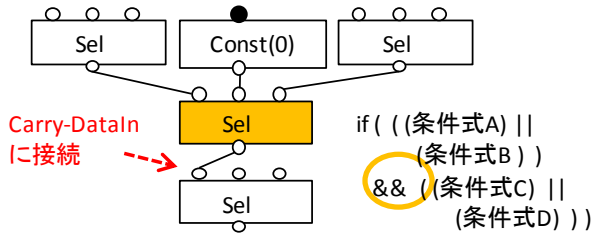


図 18　DataIn の接続より判定不可の例

図 19 に「And-Sel 型」と判定した場合(図 17-(c))の D2-CDFG の例を示す。また、「Data-Sel 型」の判定した場合の D2-CDFG を図 15 に示す。

図 19 の D2-CDFG の Sel ノードは図 15 に示した単一 Sel ノードの D2-CDFG とは違い、入力値・出力値にデータ線を用いず、キャリー線のみで表現する。図 17-(c)の条件式 A の演算結果(0,1)を図 19 の Sel ノードのキャリー線(cix)に入力し、条件式 B の演算結果(0,1)を Sel ノードのキャリー線(ciy)に入力する。Sel ノードの演算内容は、「And-Sel 型」であるので cix,ciy の入力値を論理積で求め、「キャリー線」を用いて出力する。

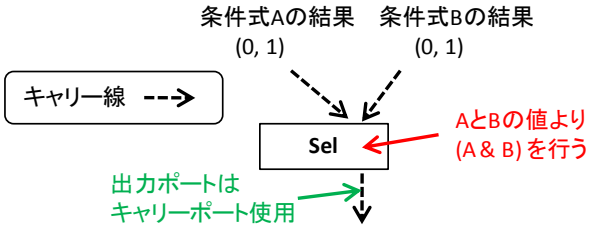


図 19　「And-Sel 型」の D2-CDFG

3.3.2.2　「Array-Sel 型」から「Data-Sel 型」への変換

「Array-Sel 型」とは、図 17-(b) に示すように Sel ノードの True・False-DataIn に接続されるノードに配列の書き込みノード(LAr-W)が接続される Sel の型である。LAr-W ノードは、例えば、図 20 のように配列 X[10](ビット幅 320bit)の要素 5 番目にデータ 10(ビット幅 32bit)を書き込む場合に用いる。

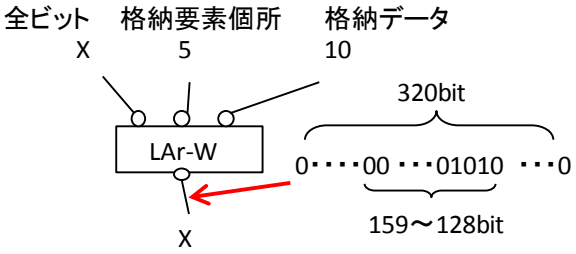
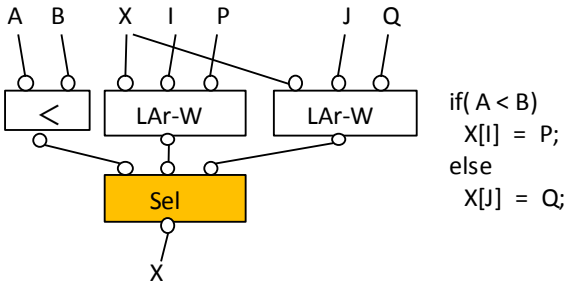


図 20　LAr-W ノード CDFG 構成

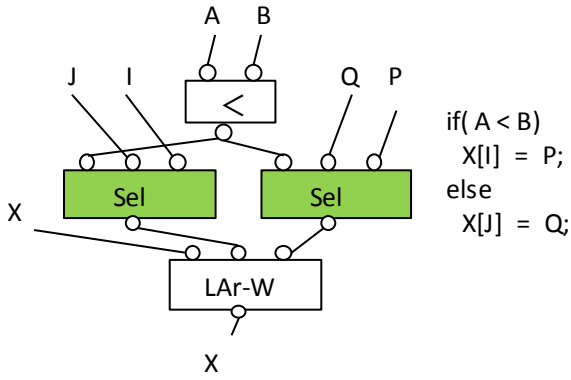
DAPDNA-2 で配列 X にデータを書き込む場合、RAM エレメントを用いる。RAM エレメントにデータを書き込む際、メモリアドレスとデータがある。RAM の出力ポートには LAr-W ノードと異なり、アドレス指定することにより、32 ビット分のデータのみを出力する。図 17-(b)に示す CDFG の場合、1 つの LAr-W ノードを 1 つの RAM エレメントに割りつけるので、C 言語記述上同じ配列に 2 つの RAM エレメントを用いることになる。それぞれの RAM エレメント内に、if 側、else 側の処理結果内容が残り、次の処理にこの配列を使用する場合、どちらの RAM エレメントを使用するかは不定である。このため、「Array-Sel 型」を「Data-Sel 型」に変換する。これより、Sel ノードが「Data-Sel 型」と判定され、D2-CDFG に変換可能となる。例えば、図 21-(a)の「Array-Sel 型」は、図 21-(b)のように「Data-Sel 型」に変換される。

この変換手順は、まずそれぞれの LAr-W ノードの全ビット(X)を探索し、LAr-W ノードを 1 つにまとめる。次に、格納要素個所がそれぞれ異なるか探索を行う。異なる場合、図 21-(b)のように要素個所にも Sel ノードを挿入する。また、異ならない場合、Sel ノードは挿入する必要はなく、

格納要素個所のデータ線を LAr-W ノードに接続する。同様に、格納データについてもそれぞれ探索を行い、Sel ノードを挿入する。次に、条件式の出力値を挿入した Sel ノードの Carry-DataIn と接続させ、それぞれの True・False-DataIn に格納要素個所・格納データの接続を行う。最後に、着目する Sel ノードを削除することで変換を行うことができ、挿入した Sel ノードは「Data-Sel 型」と判定される。



(a) 「Array-Sel 型」変換前



(b) 「Array-Sel 型」変換後
図 21 「Array-Sel 型」の変換

3.3.3 投機実行への変換(変換 B)

投機実行を行えない場合とは、条件式の結果より if 側か else 側の処理のどちらか片方を行う処理であり、図 9 に例を示す。ここでは条件式ノード(<)の出力値を Branch ノードの Selector ポートに入力することにより、Branch ノードが Branch ノードの内の２つの body ノードのどちらかを制御する。これを投機実行に行う変換アルゴリズム「変換 B」を適用する(図 22)。

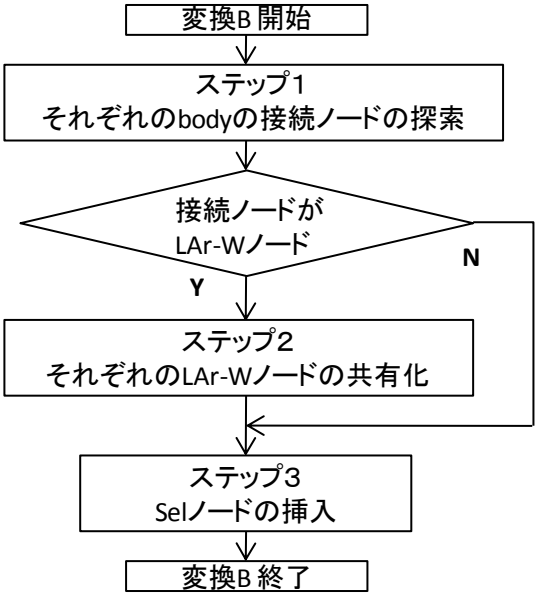
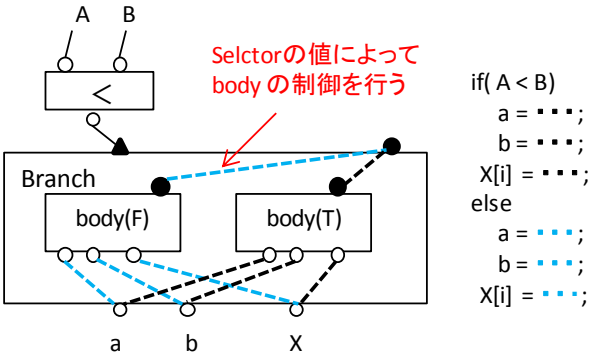


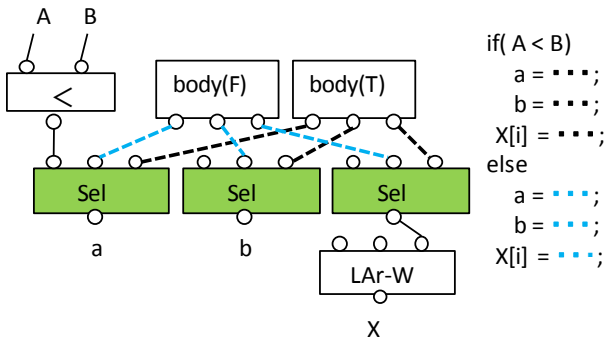
図 22 変換 B 処理フロー

(ステップ 1) body ノードの接続ノード探索
まず、各 body の DataOut ポートと接続されているノード(body ノード内)が LAr-W ノードであるかどうかを探索する。
(ステップ 2) LAr-W ノードの共有化
ステップ 1 において接続ノードが配列の書き込みノード(LAr-W)であった場合、RAM エレメントを 1 つにする。3.3.3.2 の変換と同様に一方の LAr-W ノードの全ビットより、もう一方の LAr-W ノードを見つけ、共有化を行う。また、ステップ 3 のために、各格納要素個所と格納データと接続されるノードを探索する。

(ステップ 3)Sel ノードの挿入
各 body ノードの DataOut ポートに Sel ノードを挿入し、投機実行に変換する。LAr-W ノードがある場合は、Array-Sel 型から Data-Sel 型への変換を行う。図 23 に変換前(a)と変換後(b)の CDFG 例を示す。



(a) 投機実行処理を行わない CDFG



(b) 投機実行処理に変換した CDFG

図 23　投機実行処理への変換

4.　PE のアロケーションとスケジューリング

D2-CDFG に変換されたグラフは、ノードの PE への割り付け（アロケーション）と、ディレイノードの挿入によるタイミング調整（スケジューリング）を行い、DNA コンフィギュレーションを生成する。

まず、各ノードの演算子に対し、割り当て可能な演算器候補の情報を生成する。各ノードの割り当て可能な演算器の数を「割り当て自由度」と呼ぶ。この「割り当て自由度」の低いノードを優先しつつ、各ノードをデータフローに沿って PE に割り当てる

DAPDNA-2 では入力から出力まで使用する PE の数によって最大遅延が決定される。アロケーション後の D2-CDFG では演算器単位でノードを構成していたが、PE 単位にノードの表現単位を変換する。ノードの表現単位が PE であり、各ノードが PE の遅延情報を持つグラフを「タイミンググラフ」と呼ぶ。ディレイノードを追加・移動する手順を以下に示す。

① タイミンググラフについて、対象ノードの入力タイミング（根～親ノードまでの遅延数の合計）を算出

② 対象ノードの双方の親ノードの入力タイミングを比較して、同じ遅延数になるように、ディレイノードを挿入

5.　まとめ

動的再構成プロセッサのコンフィギュレーション自動生成アルゴリズムの提案を行った。ループ処理のパイプライン化と投機的実行の変換に焦点を当て、デバイスの特性を生かしたコンフィグレーションの自動設計を行う手法である。今後、本文のアルゴリズムに基づいたプログラムの設計と、大規模なアルゴリズムに対する適用とその評価を行う。

参考文献

[1] 天野英晴. ″最近のリコンフィギャラブルシステム, 動的リコンフィギャラブルシステムの動向″ 電子情報通信学会技術研究報告. **105**(36): 31-36 (2005).

[2]T. Toyo, H. Watanabe, and K. Shiba. ``Implementation of Dynamically Reconfigurable Processor DAP/DNA-2,'' Proc, of IEEE VLSI-TSA Int. Sympo. on VLSI Design, Automation & TEST, pp. 321-322, 2005.

[3] H. Nakano, T. Shindo, T. Kazami, and M. Motomura, ``Development of Dynamically Reconfigurable Processor Lsi,'' NEC Technical Journal, Vol. 56, No. 4, pp. 99-102, 2003.

[4]八並 泰一郎, et al. "動的部分再構成技術を用いたJPEGデコーダの機能分割実装" 情報処理学会研究報告. pp43-148, 2008.

[5]Okada, K., A. Yamada, et al. ″Hardware Algorithm Optimization Using Bach C(Special Section of Selected Papers from the 14th Workshop on Circuits and Systems in Karuizawa).″ IEICE transactions on fundamentals of electronics, communications and computer sciences **85**(4): 835-841, 2002.