修 士 論 文

資源割り当て優先探索に基づく データパス・フロアプラン同時合成

指導教官　金子峰雄 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

大橋 功治

2000 年 2 月 15 日

目 次

図 目 次

表 目 次

第 1 章

はじめに

1.1 本研究の背景

VLSI は数十万，数百万ゲート数を越える数となり，その増加傾向は一層強めている．その恩恵により，複雑なアルゴリズムを処理することができるシステムが 1 つの VLSI で実現することができるようになった．携帯型情報機器やオーディオ装置などに VLSI が用いられ，これらの多くは実時間処理を行うため，リアルタイム性への要求が高く，高速化が必要になってくる．また，軽量なエネルギー源で，多様で複雑な処理を長時間実行を行いたい．

システムの設計においては，VLSI の記述階層をいくつかに分けて，上位レベルの記述言語を下位レベルの記述言語に自動変換する合成技術が用いられる．高位合成は集積回路システム階層設計の上位に位置するものである．要求される仕様の動作もしくはアルゴリズムを直接的に式や関数を使用して表現している動作記述から，レジスタ，演算器，バスなどで構成されたレジスタ・トランスファ・レベル動作記述を生成する設計技術である [1].

演算モジュールの諸特性量が集積システム全体の諸特性量を決める支配的要因である場合，モジュールなどの資源量と回路の実行ステップ数を評価したスケジューリングや資源割り当ての最適化が高位合成の目標となる．このため List scheduling や Force-Directed scheduling に代表されるスケジューラにより，資源量を制約してスケジューリングあるいは資源量の最小化を目的とするスケジューリングが行われ，その結果を受けて，演算，データの演算器，レジスタの割り当て，結線設計，レイアウトへ続く逐次処理が行われることが多い．一方，近年においては微細化の進歩により，配線に起因する動作速度や消費電力などのシステム性能への影響が相対的に増大してきている．しかし，スケジューリン

グを開始点とする逐次的処理では設計の初期段階で配線資源を考慮した意思決定が困難であり，配線資源の最適性は保証することができない．そこで，配線長を制御するためにレイアウトを考慮したアーキテクチャ最適化が必要となっている [2, 3].

1.2 本研究の目的

本研究は，面積やコントロールステップ数などに加えて，信号伝播遅延，消費電力をも同時に最適化できる高位合成システムの構築を目的とし，スケジューリング，資源割り当てといった高位データパス合成問題とモジュール配置問題を同時に取り扱うデータパス合成手法を検討する．

スケジューリング，資源割り当て，レイアウト個々はいずれも $\mathcal{NP}$ 完全なクラスに属する問題でり，データパス合成問題とモジュール配置問題を合成した問題も $\mathcal{NP}$ 完全なクラスに属すると予想される．最適解の探索に当たって，スケジューリング，資源割り当て，レイアウトの全てから構成される解空間は膨大であり，探索空間を縮小する工夫が必要である．ここでは，資源割り当てがスケジューリングに対して同時実行性や実行順序に関する制約を生成し，レイアウトに対して結線情報等を生成することから，資源割り当て空間を優先して探索する手法 [4] による (疑似) 分枝限定手法を採る．枝刈りにはスケジューリングとレイアウトの特性評価を使用する．

モジュール配置問題としては，総配線長最小化や重み付き総配線長最小化があるが，これらは $\mathcal{NP}$ 困難であることが知られている．この総配線長最小化を解く代表的なヒューリスティクな手法として， Simulated Annealing，Simulated Quenching [5] などがある．特に，高位合成の段階でレイアウトを考えた場合に，より繰り返しの少なく，高位合成に適合したヒューリスティックな手法を考案する．

1.3 本論文の構成

第 2章に基本的な高位合成について述べ，第 3章に本研究で検討する高位合成問題を詳細化し，第 4章で高位合成問題への基本的アプローチを述べる．第 5章と第 6章で，提案手法において重要な要素手続きとなるスケジューリング手法，1 次元レイアウト手法について述べる．第 7章で実験とその結果を示し，第 8章でまとめと今後の課題を述べる．

第 2 章

高位データパス合成

2.1 高位データパス合成とは

高位データパス合成は集積回路システム階層設計の上位に位置するものであって，要求される仕様の動作もしくはアルゴリズムを直接的に式や関数を使用して表現している動作記述から，レジスタ，演算器，バスなどで構成されたレジスタ・トランスファ・レベル動作記述を生成する合成技術である．入力は動作記述を示すディペンデンスグラフ (Dependence Gragh) である．図 2.1に示す．多くのディペンデンスグラフの頂点は演算だけを表しているが，ここでは演算とデータを表している．また，枝はデータの流れを表し，演算からデータまたはデータから演算の枝だけあるものとする．

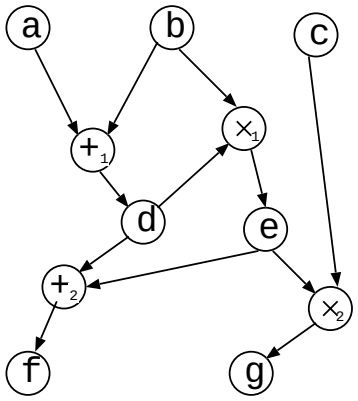


図 2.1: ディペンデンスグラフ (Dependence Gragh)

高位データパスの主な部分問題としては，演算スケジューリングや，演算器，レジスタ

といった資源を割り当てる資源割り当てなどがある．これらの出力をもとにレジスタ・トランスファ・レベル動作記述を生成する．図 2.2に高位データパス合成の構成を示す．

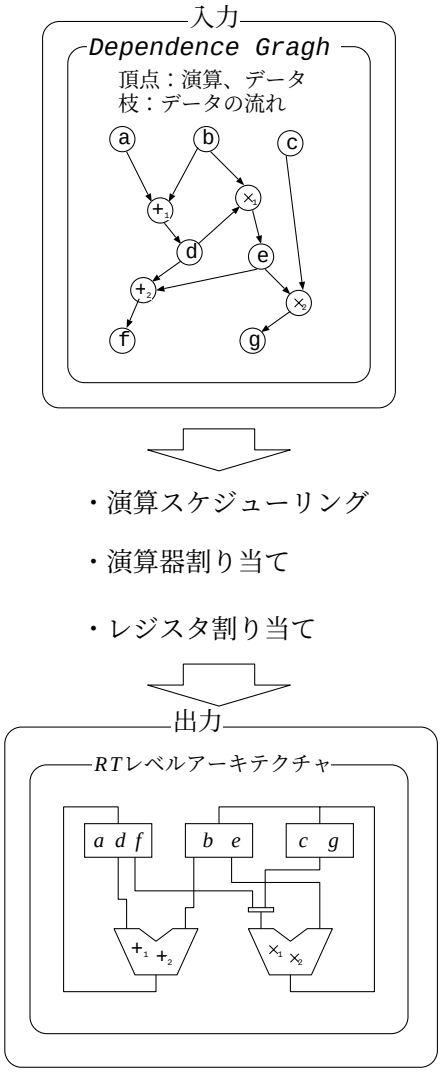


図 2.2: 高位データパス合成

2.2 演算スケジューリング

演算スケジューリングとは，各演算間のデータの依存関係を保ちつつ各演算の実行順序を決定する処理である．図 2.2に示す入力されるディペンデンスグラフの演算スケジューリングの例を図 2.3に示す．この結果では，全ての演算が実行するコントロールステップ

の数は 3 となる．また，はじめに演算 $+_1$ が実行されて，次に演算 $\times_1$ が実行され，最後に，演算 $+_2$ と演算 $\times_2$ が同時に実行されること表している．

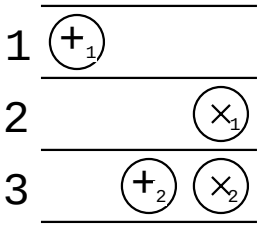


図 2.3: 演算スケジューリング結果

演算スケジューリング問題は,

- 処理時間制約下でのハードウェアー数 (演算器など) 最小化.
- ハードウェアー数 (演算器など) 制約下での処理時間最小化.

大きく 2 つの問題に大別することができる．これらの問題を解くときに，演算器の数に制約を与えずにできるだけ早く演算が終了するようにスケジューリングを行う ASAP スケジューリングや，同様に演算器の数に制約を与えずにできるだけ遅く演算が終了するようにスケジューリングを行う ALAP スケジューリングなどの情報が利用される．また，演算にある優先順序をつけて，優先順序をリストにしそれをもとにスケジューリングを行うリストスケジューリングもよく用いられ，演算スケジューリング問題を解くときに使われる．

2.3 資源割り当て

演算器割り当ては，演算が行われる演算器を決定し，レジスタ割り当ては，データが格納されるレジスタを決定する処理である．演算には加算や乗算など様々な演算があり，加算器や乗算器など演算が可能な演算器を割り当てなければならない．データも同様に様々なビット長のデータがあり，データのビット長より大きいビット長を格納することが可能なレジスタを割り当てなければならない．図 2.4に演算器，レジスタの割り当て結果を示す．この結果では，演算 $+_1$ と $+_2$ が加算器に割り当てられて，演算 $\times_1$ と $\times_2$ が乗算器に割り当てられてることを表している．また，データ a と d と f はレジスタ 1 に，データ b と e はレジスタ 2 に，データ c と g はレジスタ 3 に割り当てられてることを表している．

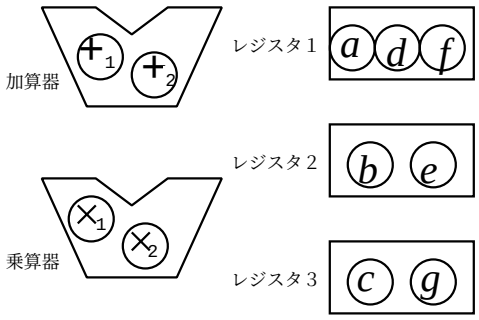


図 2.4: 演算器，レジスタ割り当て結果

演算器，レジスタ割り当ては，基本的には演算スケジューリングの後行われる．演算器，レジスタは同時に 1 つの演算処理または 1 つのデータの格納を行うことしかできないため，ある演算を実行している時間やあるデータを格納している時間は，その演算器またはそのレジスタは使用することができない．この演算器を実行している時間またはレジスタを格納している時間をライフタイムと呼ぶ．このライフタイムが重ならない演算またはデータが同一の演算器またはレジスタを使用することができる．演算器，レジスタの割り当てが決まると，演算器とレジスタ間の結線構造が決定する．演算器，レジスタの割り当ての主な目的は結線数最小化である．

第 3 章

2 次元高位合成問題

従来の多くの高位合成問題は，コントロールステップ数，資源数，結線数などの最小化であった．しかし，配線長は回路面積，実行時間，消費電力を決める重要なファクタであり，レイアウトを加味したアーキテクチャ最適化が必要となっている [2, 3]．レイアウトは本来 2 次元であるが，ここでは簡単化した問題として 1 次元レイアウトを考える．計算処理を時間軸と平面上の 1 軸で構成される 2 次元空間内にマッピングすることから，2 次元高位合成問題と呼ぶ.

3.1 アーキテクチャモデル

ここでは，レジスタ・トランスファ・レベル記述のモデルとしてマルチプレクサタイプのアーキテクチャを対象とする．構成要素は演算器，レジスタ，マルチプレクサ，資源間結線である．図 3.1にアーキテクチャモデルを示す．演算器とレジスタ間の配線は，モジュールの入力端子直前に適当な位置に挿入されたマルチプレクサを通してなされる．また，横幅のサイズが異なる演算器やレジスタは，配線長に関する評価に従って 1 次元上にレイアウトされる．

ここで，1 つのレジスタから出力されたデータが，演算器を通らずに別のレジスタに転送されることはないものとし，また chaining と呼ばれる 1 コントロールステップ内に 2 つ以上の演算がレジスタにデータを格納することなく実行される処理はないものとする．

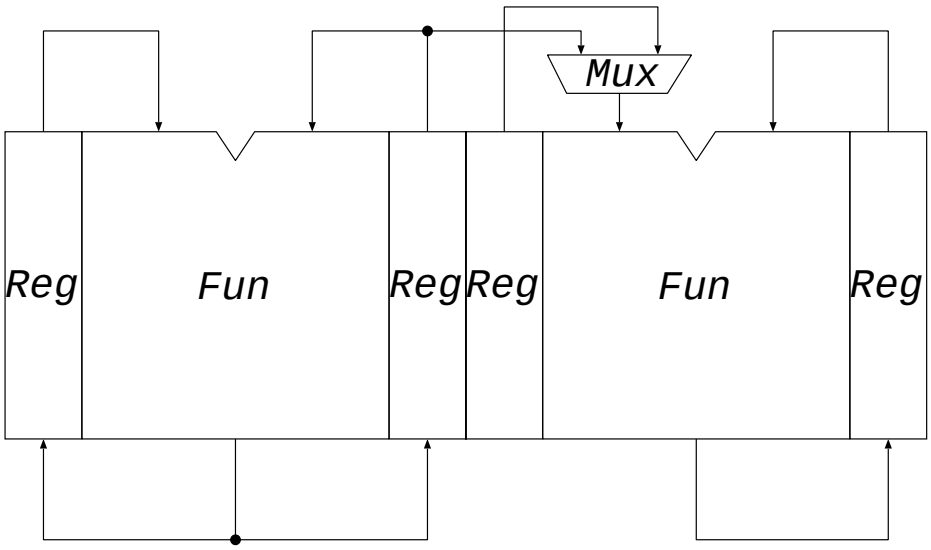


図 3.1: アーキテクチャモデル

3.2 問題の記述

[2 次元高位合成]

入力

- ディペンデンスグラフ (Dependence Gragh) $G = (V_G, A_G)$

  V_G：演算集合 V_O とデータ集合 V_D の和集合.

  A_G：データの入力の集合 A_I と出力の集合 A_O の和集合．ここで，$A_I \subseteq V_D \times V_O$, $A_O \subseteq V_O \times V_D$.

  $p(v)$：$v \in V_G$ の直前の集合.

  $s(v)$：$v \in V_G$ の直後の集合.

- ユニットライブラリ (Unit Library)

  * 演算器

    $F_i = \{f_{i,1}, f_{i,2}, \cdots, f_{i,M_{f,i}}\}$：タイプ i の演算器の集合.

    $a_{f,i}$：タイプ i の演算器の面積.

    $p_{f,i}$：タイプ i の演算器の消費電力.

    $t_f(f_{i,m})$：タイプ i の演算器の処理時間.

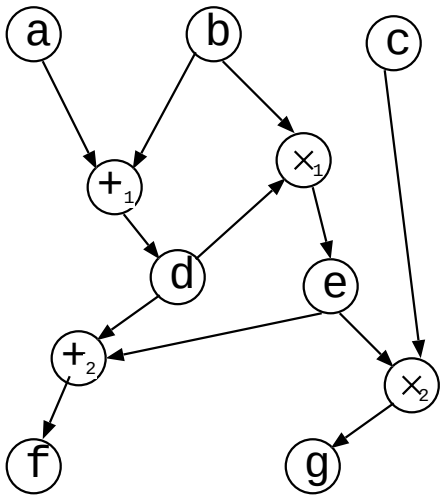


図 3.2: ディペンデンスグラフ (Dependence Gragh)

$e(f_{i,j})$：タイプ i の演算器の処理に想定するサイクル数.

$c_{f,i}$：タイプ i の演算器の入力容量.

* レジスタ

$R_j = \{r_{j,1}, r_{j,2}, \cdots, r_{j,M_{r,j}}\}$：タイプ j のレジスタの集合.

$a_{r,j}$：タイプ j のレジスタの面積.

$p_{r,j}$：タイプ j のレジスタの消費電力.

$t_r(r_{j,m})$：タイプ j のセットアップタイム.

$c_{r,j}$：タイプ j のレジスタの入力容量.

* マルチプレクサ

$a_{m,k}$：タイプ k のマルチプレクサの面積.

$p_{m,k}$：タイプ k のマルチプレクサの消費電力.

$t_{m,k}$：タイプ k のマルチプレクサの通過時間.

* 配線

C_a：単位長当りの配線面積.

r：単位長当りの配線抵抗.

c：単位長当りの配線容量.

C_p：単位容量当りの配線消費電力.

出力

- 演算スケジューリング

  演算スケジューリングは演算集合をコントロールステップに割り当てる.

  $\sigma : V_O \to Z_+$

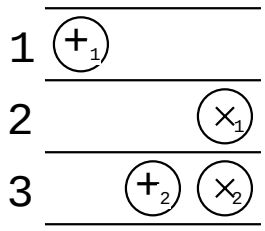


図 3.3: 演算スケジューリング

- 演算器の割り当て

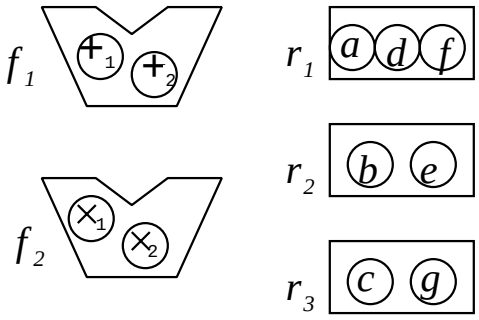
  演算器の割り当ては演算集合を演算器に割り当てる.

  $\rho : V_O \to \mathcal{F}$, 但し $\mathcal{F}$ は演算器モジュール集合

- レジスタの割り当て

  レジスタの割り当てはデータ集合をレジスタに割り当てる.

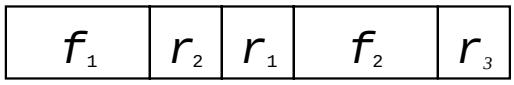
  $\xi : V_D \to \mathcal{R}$, 但し $\mathcal{R}$ はレジスタモジュール集合



- 1 次元レイアウト

  1 次元レイアウトは割り当てられた演算器またはレジスタの順番を決める.

  $\varphi : \mathcal{F} \cup \mathcal{R} \to Z_+$



レジスタの数，タイプ k のマルチプレクサの数である．C_a は単位長の配線面積で，l_n がネット n の長さである．

式 (3.1) の配線に関する項を詳細に書くと以下のように表現できる．

$$C_a \sum_{E_n \in E_H} l_n = C_a \sum_{E_n \in E_H} [\max\{\lambda_\varphi(v_i) | v_i \in E_n\} - \min\{\lambda_\varphi(v_j) | v_j \in E_n\}] \tag{3.2}$$

3.3.2 計算処理時間

設計対象となる計算処理の実行時間 T は，処理に要するコントロールステップの数 C と 1 コントロールステップの時間であるクロックピリオド D とを掛けたものとして与えられる．

$$T = C \times D \tag{3.3}$$

コントロールステップ数 C は開始コントロールステップから終了コントロールステップまでの数である．クロックピリオド D は各演算で構成されるレジスタからレジスタへのパスについての最大信号遅延で与えられ，次のように評価する．(図 3.6)

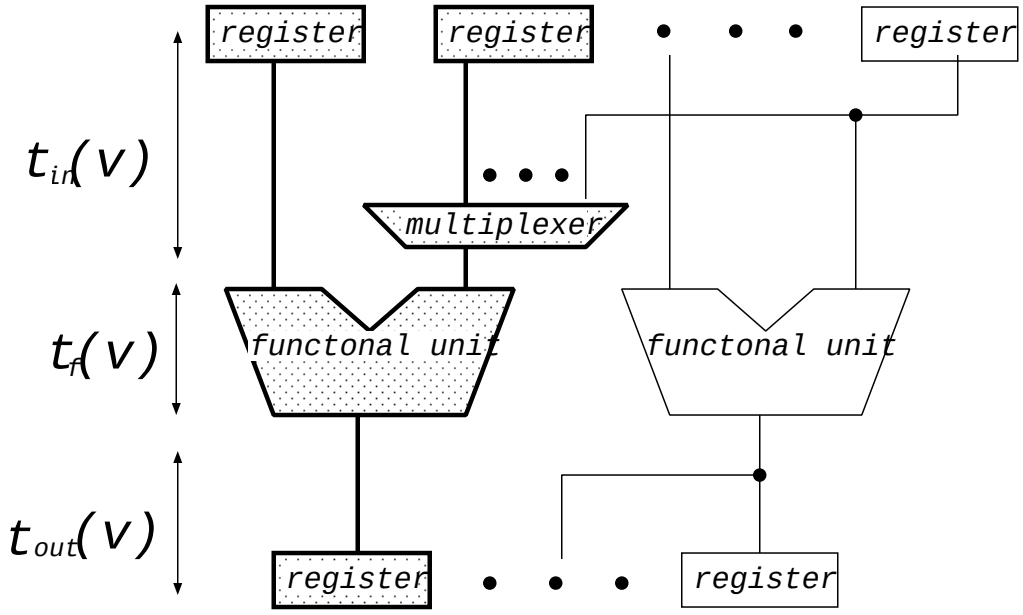


図 3.6: クロックピリオド

$$D = \max_{v \in V_O} \frac{t_f(v) + t_{in}(v) + t_{out}(v)}{e(v)} \tag{3.4}$$

$$
\begin{aligned}
t_{in}(v) &= \max_{p_i(v)}[t_r(p_i(v)) + t_l(p_i(v), v) + t_{m,k}(p_i(v))] \\
t_{out}(v) &= t_l(v, s(v)) + t_{m,k}(v)
\end{aligned}
$$

ここで，$t_f(v)$ は v が割り当てられた演算器の処理時間である．$t_r(p_i(v))$ は v の入力データが格納されるレジスタのセットアップタイム，$t_l(p_i(v), v)$ は入力データが格納されているレジスタから v が割り当てられた演算器までの配線による遅延で，マルチプレクサがあればその通過時間を加えた最大遅延を与えるものをレジスタから演算器まで遅延 $t_{in}(v)$ である．$t_l(v, s(v))$ は演算器から出力データが格納されるレジスタまでの配線遅延で，マルチプレクサがあればその通過時間を加えたものを演算器からレジスタまでの遅延 $t_{out}(v)$ である．$e(v)$ を v が割り当てられた演算器の処理に想定するコントロールステップ数である．

今，$E_n = \{v_{-N}, \cdots, v_{-1}, v_0, v_1, \cdots, v_M\} \in E_H$, かつ $\lambda_p(v_i) < \lambda_p(v_{i+1}), -N \leq i < M$ とする．また，v_0 を信号のソース，他をシンクとする．v_0 から $v_j, 0 < j$ までの信号伝播遅延 $t_l(v_0, v_j)$ は，エルモアの遅延モデルに基づけば以下の通り．

$$
\begin{aligned}
t_l(v_0, v_j) &= r_d \left[c\{\lambda_p(v_M) - \lambda_p(v_{-N})\} + \sum_{v_k \in E_n} C_{v_k} \right] \\
&\quad + \sum_{i=0}^{j-1} r(\lambda_p(v_{i+1}) - \lambda_p(v_i)) \left[\frac{c(\lambda_p(v_{i+1}) - \lambda_p(v_i))}{2} \right. \\
&\quad \left. + c\{\lambda_p(v_M) - \lambda_p(v_{i+1})\} + \sum_{l=i+1}^{M} C_{v_l} \right] \qquad (3.5)
\end{aligned}
$$

ここで，r_d は出力駆動抵抗で，c は単位長の配線容量で，r は単位長の配線抵抗である．C_{v_i} は v_i をソースとしたときのシンクまでの配線容量とシンクの容量の合計である．現時点では配線に関する 2 次項を無視し，式 (3.5) の第 1 項だけを信号伝播遅延 $t_l(v_0, v_j)$ として用いている．

$$
t_l(v_0, v_j) \approx r_d \left[c\{\lambda_p(v_M) - \lambda_p(v_{-N})\} + \sum_{v_k \in E_n} C_{v_k} \right] \qquad (3.6)
$$

3.3.3 消費電力

総消費電力 P は，各ハードウエア資源の使用回数に関係して決まり，次の式で評価できる．

$$
P = \sum_{i,n} p_{f,i} u_{f,i,n} + \sum_{j,n} p_{r,j} u_{r,j,n} + \sum_{k,n} p_{m,k} u_{m,k,n} + C_p \sum_{n} c l_n u_n \qquad (3.7)
$$

ここで，$p_{f,i}, p_{r,j}, p_{m,k}$ はタイプ i の演算器の消費電力，タイプ j のレジスタの消費電力，タイプ k のマルチプレクサの消費電力である．$u_{f,i,n}, u_{r,j,n}, u_{m,k,n}, u_n$ はタイプ i の n 番の演算器の利用回数，タイプ j の n 番のレジスタの書き込み回数，タイプ k の n 番のマルチプレクサの利用回数で，ネット n の使用回数である．l_n はネット n の長さで，c は単位長の配線容量で,C_p は単位配線容量の消費電力である．

式 (3.7) の配線に関する項を詳細に書くと以下のように表現できる．

$$C_p \sum_{E_n \in E_H} c l_n u_n = C_p \sum_{E_n \in E_H} c[\max \lambda_\varphi(v_i) | v_i \in E_n - \min \lambda_\varphi(v_j) | v_j \in E_n] u_n \tag{3.8}$$

第 4 章

資源割り当て優先探索

4.1 従来の逐次的な手法と問題点

演算モジュールの諸特性量が集積システム全体の諸特性量を決める支配的要因である場合，モジュールなどの資源量と回路の実行ステップ数を評価したスケジューリングや資源割り当ての最適化が高位合成の目標となる．このため List scheduling や Force-Directed

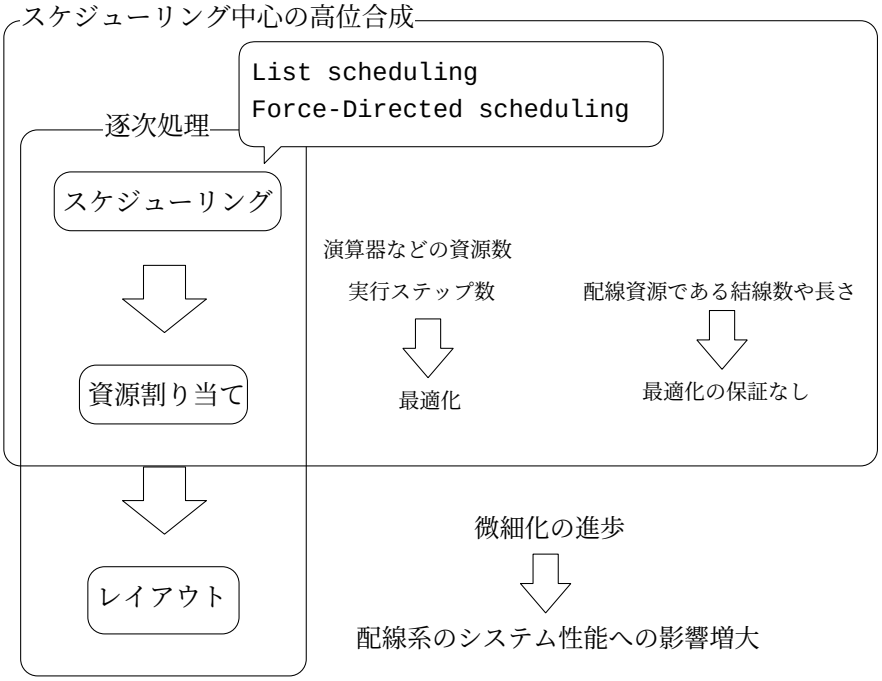


図 4.1: 逐次的な手法

scheduling に代表されるスケジューラにより，資源量を制約してスケジューリングあるいは資源量の最小化を目的とするスケジューリングが行われ，その結果を受けて，演算，データの演算器，レジスタの割り当て，結線設計，レイアウトへ続く逐次処理 (図 4.1) が行われることが多い．一方，近年においては微細化の進歩により，配線に起因する動作速度や消費電力などのシステム性能への影響が相対的に増大してきている．しかし，スケジューリングを開始点とする逐次的処理では設計の初期段階で配線資源を考慮した意思決定が困難であり，配線資源の最適性は保証することができない．

4.2 提案手法

スケジューリング σ，資源割り当て ρ，ξ，レイアウト φ の個々はいずれも $\mathcal{NP}$ 困難なクラスに属する問題であり，2 次元高位合成問題も $\mathcal{NP}$ 困難なクラスに属すると予想される．最適解の探索に当たって，σ，ρ，ξ，φ の全てから構成される解空間は膨大であり，探索空間を縮小する工夫が必要である．

ここでは，資源割り当てがスケジューリングに対して同時実行性や実行順序に関する制約を生成し，レイアウトに対して結線情報等を生成することから，資源割り当て空間を優先して探索する手法を採る [4]．具体的には，資源割り当て空間の (擬似) 分枝限定法に基づく探索を行い，分枝操作としては，1 つの演算またはデータを 1 つの演算器またはレジスタに割り当てたときに，その割り当てまでの部分解に対するスケジューリングと，レイアウトによる特性評価を行なう．その際に，スケジューリングとレイアウトによる特性評価を枝刈りに使用する (図 4.2).

以下に，提案する 2 次元高位合成問題を解く (擬似) 分枝限定法の概要を示す.

主ルーチン；

F： 資源 (モジュール) の集合

QB & B(V_G);

1. V_G が空ならば特性を評価し，必要に応じて最適解を更新．後 return.
2. V_G から一つの要素 v を選択.
3. F の各要素 f_i に関して以下を繰り返す.
   - 3.1 v が f_i に割り当てできなければ，次の f_i へ
   - 3.2 v を f_i に割り当て.

3.3 割り当て済みの情報からモジュール接続関係と同時実効性や実行順序のスケジューリングに関する制約を生成.

3.4 1 次元レイアウトと総配線長，最大信号遅延，重み付き総配線長の特性評価.

3.5 スケジューリングと併せて，回路面積，実行時間，消費電力の特性を優先順位で評価.

3.6 更なる探索が必要ならば，QB&B$(V_G - \{v\})$ を実行.

3.7 f_i を割り当てる前の 1 次元レイアウトとスケジューリング状態に戻し，次の f_i へ.

4. return

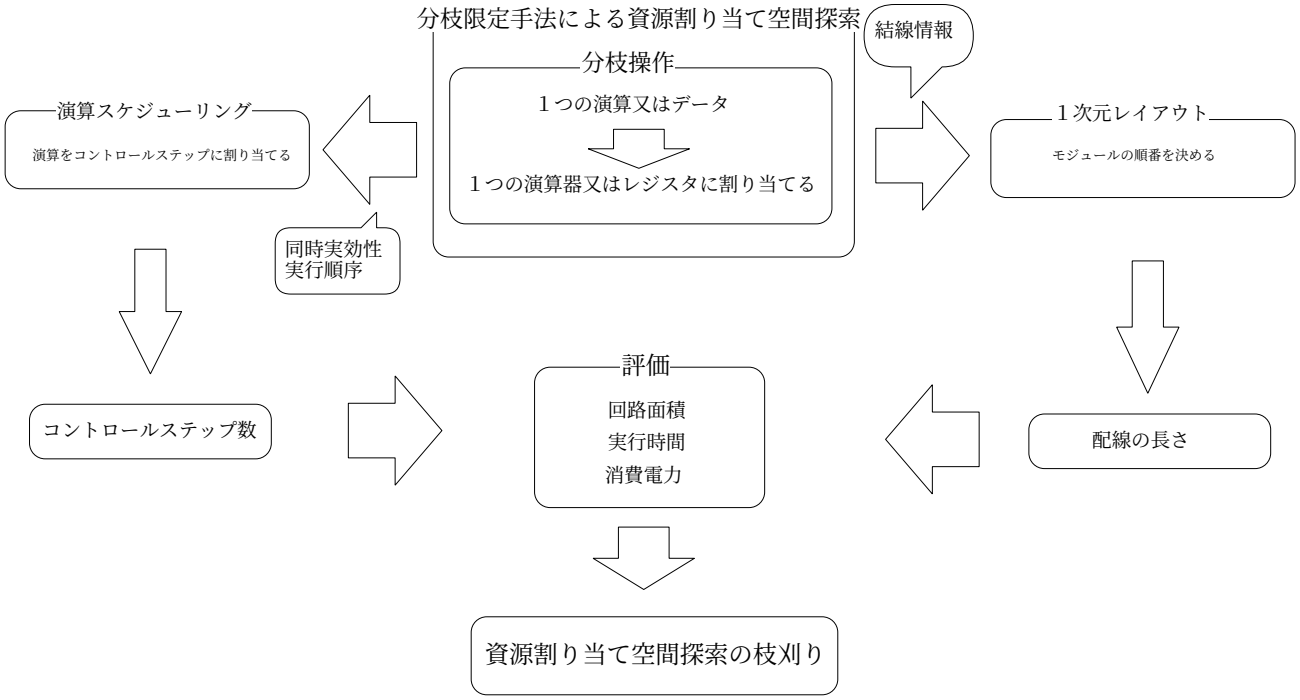


図 4.2: 提案手法

第 5 章

スケジューリング

スケジューリングは，入力となる Dependence Gragh にて指定されるデータ依存関係を保って，各演算の開始時刻 (コントロールステップ) を決定する問題である．特に，演算，データの演算器，レジスタへの割り当てが指定される時は，同一演算器へ割り当てられた演算の実行順序及び同一レジスタに割り当てられたデータの生存区間 (生成されてから使用し終えるまでの時区間) の順序を決める問題に帰着される [4].

資源割り当て駆動スケジューリング問題は，ジョブショップスケジューリング問題を部分問題として含んでおり，一般に $\mathcal{NP}$ 困難である．

5.1 スケジューリンググラフ

入力として与えられる Dependence Gragh からスケジューリンググラフの変形を考える．その際に，2 つの予備の演算頂点 o_{init} と o_{quit} を導入する．o_{init} は初めの入力データのための出ていく枝を持ち，入ってくる枝はない．o_{quit} は出力データからの入ってくる枝を持ち，出ていく枝はない．スケジューリンググラフ $G_s = (V_s, A_s)$ は $V_s = V_{\lfloor\rfloor} \cup V_{\lceil\rceil}, A_s = A_c \cup A_{\leq} \cup A_h$ として構成される．

- $V_{\lfloor\rfloor}$ は演算の開始時間に相当する開始演算の集合．
- $V_{\lceil\rceil}$ は演算の終了時間相当する終了演算の集合．
- A_c は 定数制約アークの集合と呼び，開始演算と終了演算間のアークである．アークの重みは，(その演算の処理時間に相当するコントロールステップ数) - 1 である．もし，$(u, v) \in A_c$ で，c の重みを持っているなら，$u + c = v$ を満たされることを要求している．

- $A_{\leq}$ は上限制約アークの集合と呼び，ある終了演算とある開始演算間のアークである．アークの重みは 1 である．もし，$(u, v) \in A_{\leq}$ なら，$u + 1 \leq v$ を満たすことを要求している．

- A_h は仮のアークの集合と呼び．仮のアークは多数ソース/単一デスティネーションまたは単一ソース/多数デスティネーションのどちらかを持つ．それぞれ，$(u_1, \cdots, u_i : v)$ と $(u : v_1, \cdots, v_j)$ と記す．仮のアークはいつも組で現れる．初期スケジューリングラフでは $\emptyset$ である．

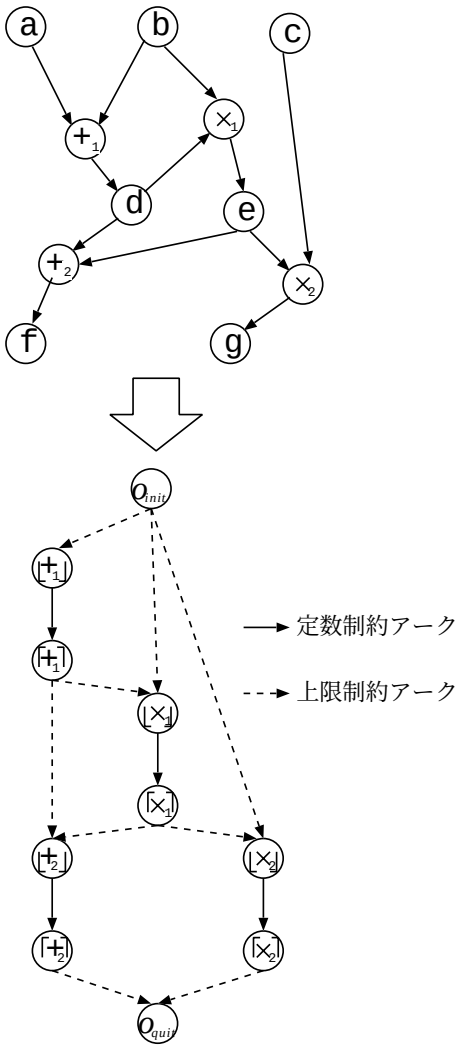


図 5.1: Dependence Gragh と Scheduling Gragh

5.2 スケジューリングと割り当ての相互作用

2 つの演算 o_i と o_j が同じ演算器に割り当てられるとき，スケジューリンググラフに仮のアーク $(\lceil o_i \rceil : \lfloor o_j \rfloor)_1$ と $(\lceil o_j \rceil : \lfloor o_i \rfloor)_1$ の組を加える．ここで，あとつづりはアークの重みである．

2 つのデータ d_k と d_l が同じレジスタに割り当てられるとき，スケジューリンググラフに仮のアーク $(\lceil s_1(d_k) \rceil, \cdots, \lceil s_i(d_k) \rceil : \lceil p(d_l) \rceil)_0$ と $(\lceil s_1(d_l) \rceil, \cdots, \lceil s_j(d_l) \rceil : \lceil p(d_l) \rceil)_0$ の組を加える．

5.3 リダクション

仮のアークはいつも組で現れて，仮のアークの 1 つを取り除いて，他方の方を上限制約アークにする操作をリダクションという．

5.3.1 順序関係制約

$(x_1, \cdots, x_i : y)_\delta$ と $(u_1, \cdots, u_j : v)_\delta$ を仮のアークの 1 つの組とする．ここで δ はアークの重みで 0 か 1 である．もし，$x_1, \cdots, x_i$ の 1 つから y までに定数制約アークと上限制約アークだけ含むパスがあるなら，$(x_1, \cdots, x_i : y)_\delta$ を上限制約アークとする．

5.3.2 時間制約

σ_{asap} と σ_{alap} は ASAP スケジューリングと ALAP スケジューリングで頂点に正の正数を割り当てることとする．$(x_1, \cdots, x_i : y)_\delta$ と $(u_1, \cdots, u_j : v)_\delta$ を仮のアークの 1 つの組とする．もし，

$$\forall i', 1 \leq i' \leq i \quad : \quad \sigma_{asap}(x'_i) + \delta \leq \sigma_{alap}(y) \tag{5.1}$$

$$\exists j', 1 \leq j' \leq j \quad : \quad \sigma_{asap}(u'_j) + \delta > \sigma_{alap}(v) \tag{5.2}$$

$(x_1, \cdots, x_i : y)_\delta$ を上限制約アークとする．

もし，

$$\exists i', 1 \leq i' \leq i \quad : \quad \sigma_{asap}(x'_i) + \delta > \sigma_{alap}(y) \tag{5.3}$$

$$\exists j', 1 \leq j' \leq j \quad : \quad \sigma_{asap}(u'_j) + \delta > \sigma_{alap}(v) \tag{5.4}$$

与えられた資源割り当てでは，コントロールステップに割り当てることができない．

5.3.3 ヒューリスティク

$(x_1, \cdots, x_i : y)_\delta = a_y$ と $(u_1, \cdots, u_j : v)_\delta = a_v$ を仮のアークの 1 つの組とする．次のようなものを計算する．

$$Slack(a_y) = \sigma_{alap}(y) - \left(\max_{i'}[\sigma_{asap}(x'_i)] + \delta\right) \tag{5.5}$$

$$Slack(a_v) = \sigma_{alap}(v) - \left(\max_{i'}[\sigma_{asap}(u'_i)] + \delta\right) \tag{5.6}$$

$Slack$ が大きい方を上限制約アークとする．

例えば図 5.2の場合，$Slack(a_y) = 2$ で $Slack(a_v) = 1$ となり，仮のアーク (x,y) を上限制約アークとする．

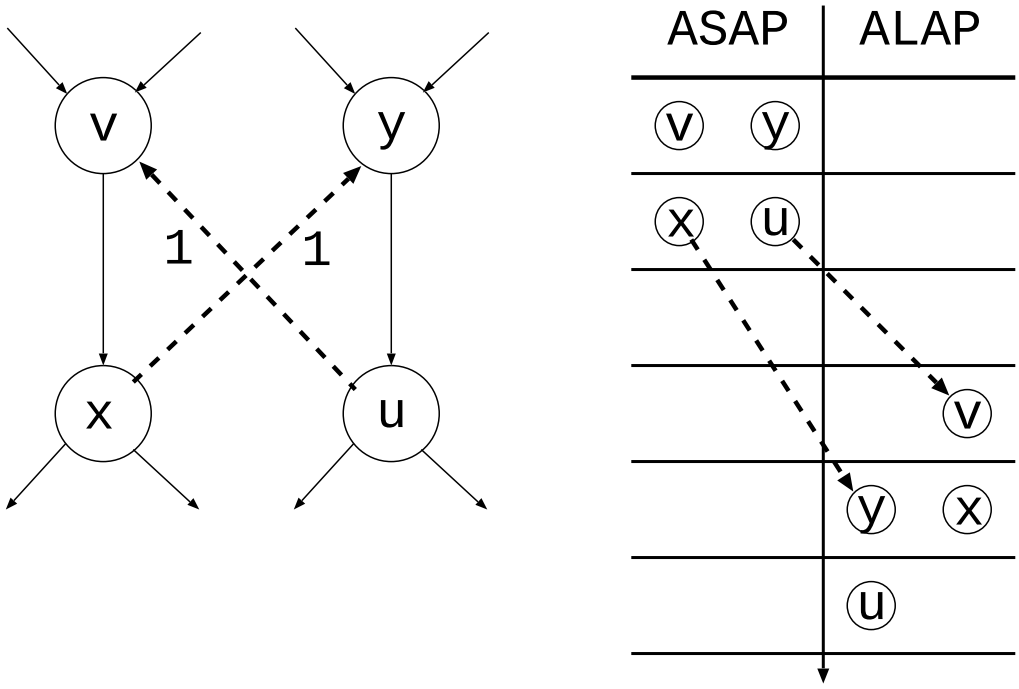


図 5.2: Slack の例

5.4 分枝限定法におけるリダクション操作

分枝限定手法による探索途中の部分割り当てに対しては，実行順序制約，時間制約を可能な限り実行し，その結果に対して最長パス長を求め，スケジューリング長の下限として

探索の枝刈に用いる．また，全ての割り当てが定められた探索木の葉においては，実行順序制約，時間制約を可能な限り実行し，未決定順序がこれらの制約で解決できなくなった時点でヒューリスティックな手法を行なって未決定順序の中の 1 つを決めて再び実行順序制約，時間制約の手続きを，全ての順序が決まるまで繰り返す．これによって求められる結果に対して最長パス長を求め，その割り当て解に対するスケジューリング長とする．

第 6 章

1 次元レイアウト

1 次元レイアウトにおける総配線長最小化は $\mathcal{NP}$ 困難であることが知られている．この総配線長最小化を解く代表的なヒューリスティクな手法として， Simulated Annealing, Simulated Quenching [5] などがある．しかし，ここでの 1 次元レイアウトの使用形態を

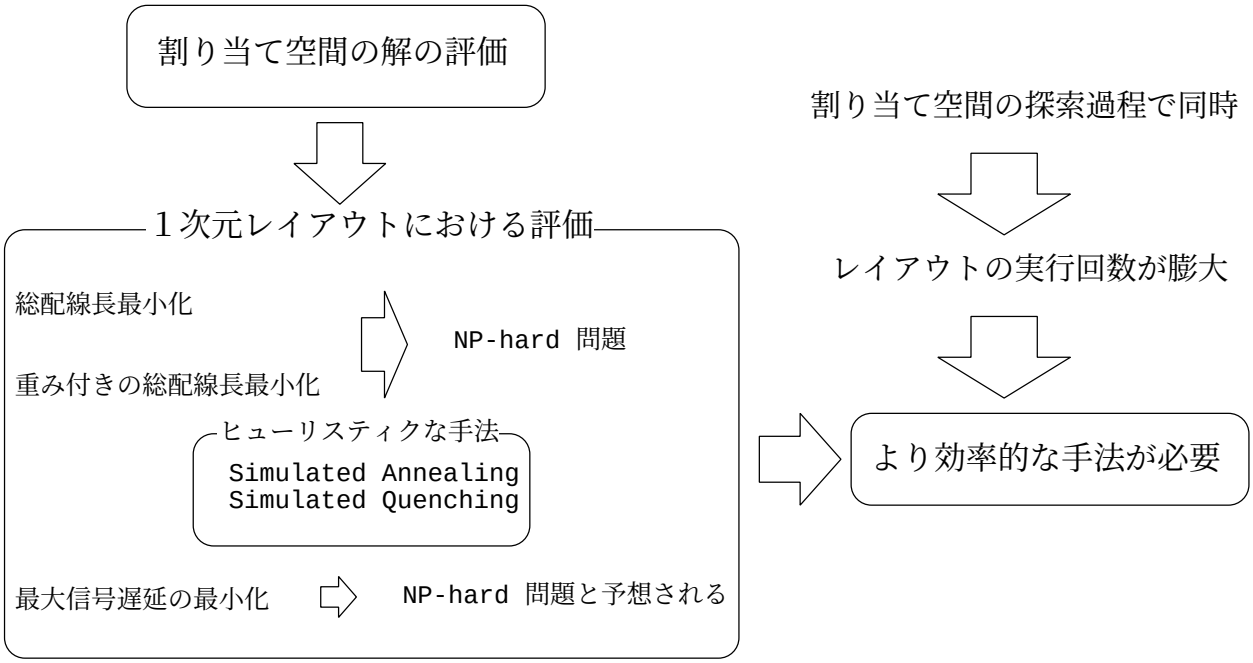


図 6.1: 1 次元レイアウト問題

考えた場合，繰り返しの少ないより効率的な手法が求められる．また，割り当て空間探索そのものとの適合性も考慮に入れる必要がある．

6.1 リップアップとリプレース

本来分枝限定法では，部分解において，それから派生する解の持つ特性量の下限を見積もって枝刈りを行う．ここで採用している分枝操作では，モジュール間の結線が追加的に増加するため，部分解に対する最適レイアウトによる特性量は，派生する解の特性量の下限となるが，先に述べた通り，最適解を求めるには大きな計算量が必要となる．また現在までに得られている下限見積もりは，自明で精度の悪いものとなっている．そこでここでは，部分解に対する発見的レイアウトによる特性量評価を持って枝刈りを実行する擬似的分枝限定法を用いる．

ヒューリスティックなレイアウト手法として構成的手法と局所的改善が考えられるが，今回は後者を採用している．特にここでは，割り当て空間探索における初期のレイアウトを空とし，部分解においては，その親の部分解に対するレイアウトを初期解として，それに対する局所的改善を行って，その部分解に対するレイアウトとしている．

部分割り当て解の，その親の部分解に対する接続関係の差異は，高々 1 つのモジュールの追加と高々モジュール数のモジュール間接続要求 (既に存在するモジュール間接続の再使用要求) の追加のみである．ここで，新たに追加された接続 (再使用) 要求のそれぞれに対して，接続が要求される 2 つのモジュールを 1 次元配置から取り出す．ここでは**リップアップ**と呼ぶ，これら 2 つのモジュール以外の配置順序を変えずに，リップアップしたモジュールを再配置する．ここでは**リプレース**と呼ぶ．2 つのモジュールに対して，全ての可能なリプレース候補について，総配線長，最大信号伝播遅延，重み付き総配線長を評価し，位置を求めている．

この 2 つのモジュールのリップアップ・リプレース操作を変化した結線要求のすべてに対して行ない，レイアウト結果を更新する．

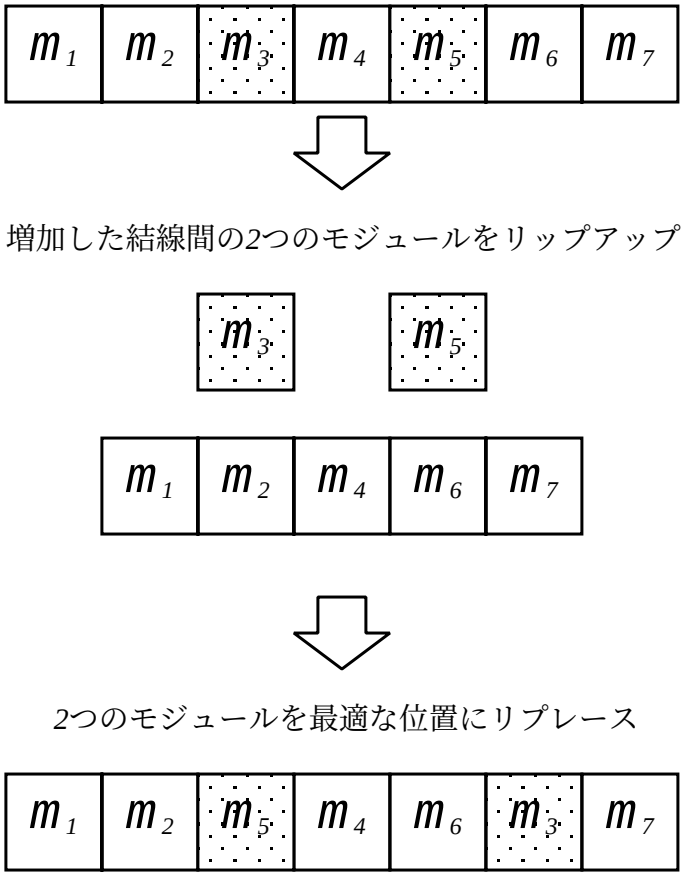


図 6.2: 2 つのモジュールのリップアップとリプレース

6.2 1 次元レイアウト手法

分枝限定手法の中で採用しているレイアウト手法は，以下の方法と同等な手法となっている．

E：2 つのモジュール間の結線要求の集合 (データ転送に使われる回数を反映して重複を許す)
$E_{done} = \emptyset$
π：任意の初期レイアウト

1. E が空ならば終了
2. E から一つの要素 e を選び，E_{done} に加える．

3. e に接続する両方ののモジュール m_1 と m_2 を選ぶ.

4. E_{done} の結線要求のみを考慮して，m_1 と m_2 を最適な位置へ移動する.

5. E から e を削除して，1. へ戻る.

6.3 1 次元レイアウトにおける計算量

分枝操作が行われたときの 1 次元レイアウトにおける計算量を示す．ここで，Dependence Gragh における演算頂点数を n，モジュールの個数を m とする．リップアップした 2 つのモジュールが，リプレースされる位置の候補は $m \times (m-1)$ である.

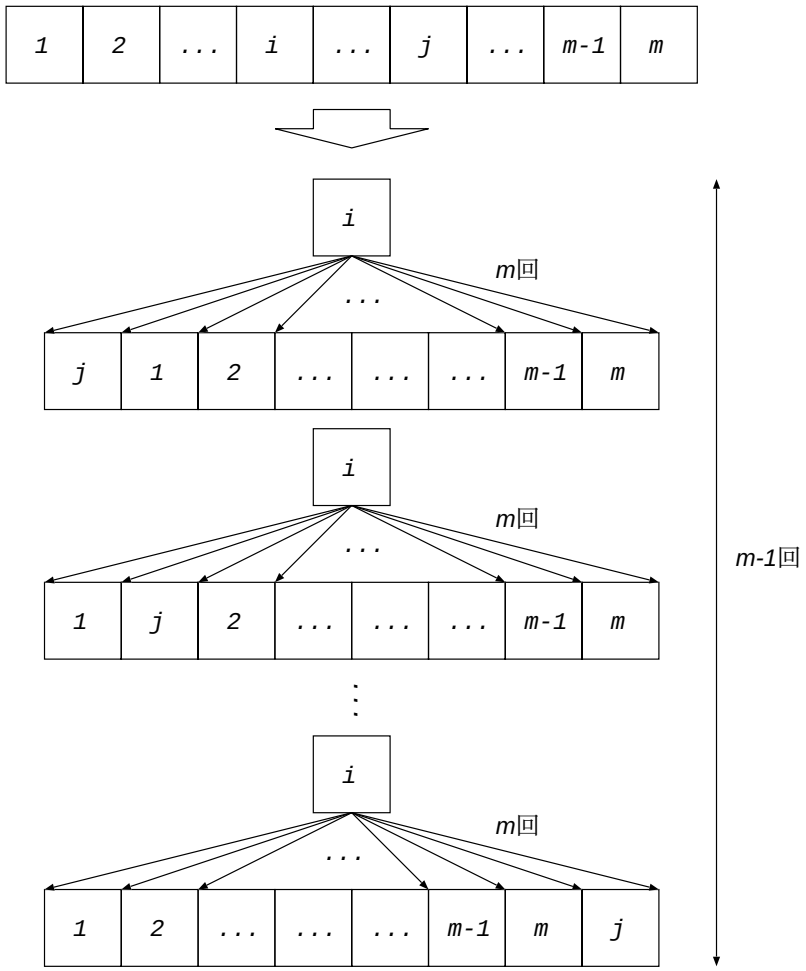


図 6.3: 2 つのモジュールのリプレース操作回数

1 つのリプレース候補に対して，総配線長，最大信号伝播遅延，重み付き総配線長の特性評価を行なうのに要する計算量は $O(n)$ となる．また，2 つのモジュールのリップアップ・れプレース操作は，1 回の分枝操作で増加した結線要求の数分行われる．変化する結線要求の数はたかだかモジュールの個数 m である．

以上より，分枝操作 1 回当たりのレイアウトの計算量は

$$O(n \times m(m-1) \times m) = O(nm^3)$$

となる．

第 7 章

設計実験

7.1 設計実験モデル

提案手法の有効性を確認するために，ワークステーション上にC言語を用いて実装を行った．入力の Dependence Gragh として図 7.1に示す微分方程式の例と図 7.2に示すディジタルフィルタの例をを用い，そのループの本体を使った．その他の設計評価に必要な入力パラメータとして，表 7.1に示すユニットライブラリを与えた．

表 7.1: ユニットライブラリ

モジュール名	乗算器	加算器	レジスタ
ステップ数	2	1	×
処理時間	20	10	1
サイズ	30	15	1
入力容量	1	1	1
出力駆動抵抗	1	1	1
単位長の配線抵抗	1		
単位長の配線容量	1		

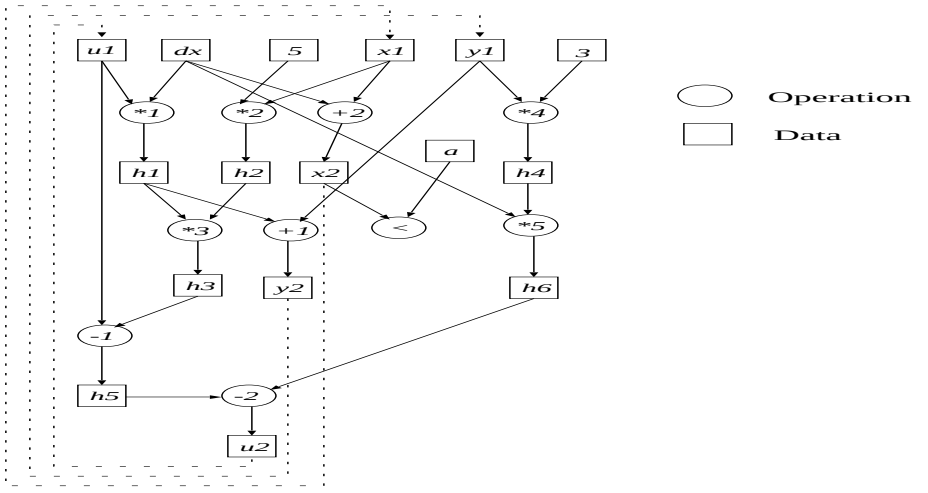
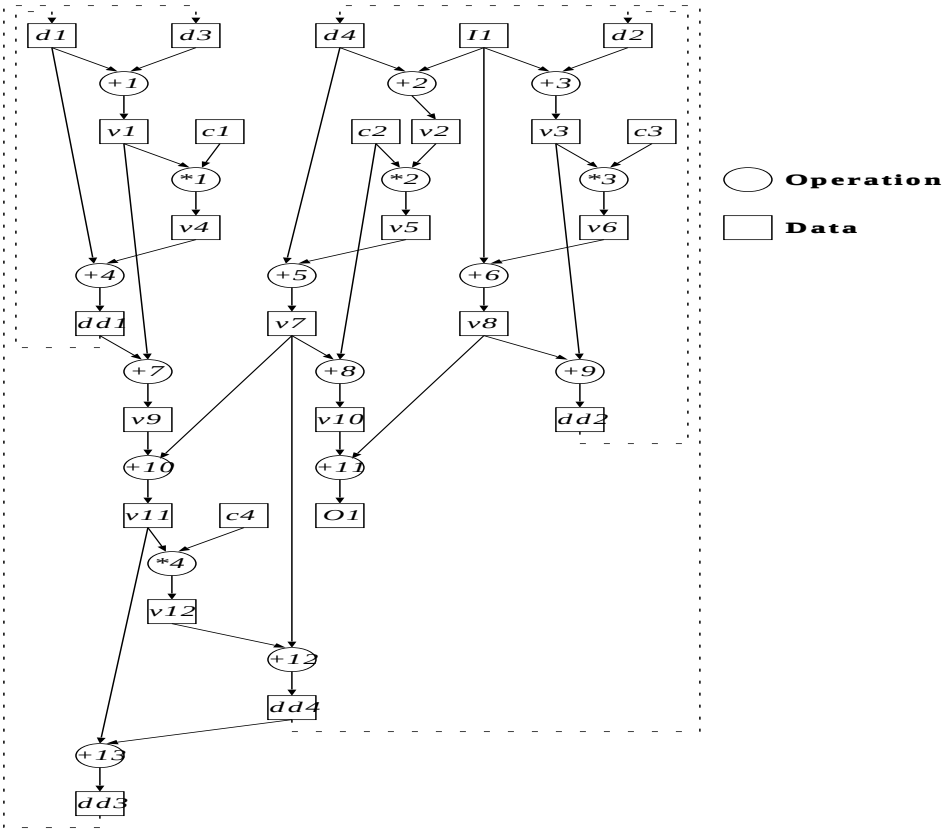


図 7.1: 微分方程式の Dependence Gragh



7.2 実験結果

微分方程式の例に対しては，設計は表 7.2に示す 3 種類の資源制約とディジタルフィルタに対しては，資源制約 4 の下で，最適化の順位を変えながら行なった．得られた結果に対しての特性量の比較を表 7.3に示す．制約 1 と制約 2 の下で，逐次的な処理となる結線数最小化とする資源割り当てを行なったあと，レイアウトを総配線長，最大信号遅延，重み付き総配線長の評価を全探索でそれぞれ求めた特性量も同時に示す．

表 7.2: 設計制約

	入力	乗算器	加算器	レジスタ	ステップ数
制約 1	DE	3	1	10	9
制約 2	DE	2	1	13	15
制約 3	DE	1	1	13	15
制約 4	DF	2	3	20	11

DE:微分方程式，DF:ディジタルフィルタ

制約条件を 1，2，3 と変化させても最も優先順位が高い回路面積，実行時間，消費電力の評価項目が，他の評価項目に比べて最小の解を生成することが分かる．制約 1 では許容コントロールステップ数を 9 に制限していたため，スケジューリングに制限ができてしまう．しかし，制約 2 で許容コントロールステップ数を 15 としたとき，乗算器，加算器を 1 個づつ用いて，制約 1 より消費電力，回路面積の解がよくなっていることが分かる．また，逐次的な手法と比較しても提案手法の方が良い解を生成することが分かる．これは，結線数やコントロールステップ数を最小化する資源割り当ての解を生成することで，レイアウト時の総配線長最小化や最大信号遅延最小化や重み付きの総配線長最小化する資源割り当ての解を逃しているためである．提案手法では資源割り当て段階で，レイアウトの総配線長，最大信号遅延，重み付き総配線長の評価を行なっているので，これらの評価を最小化する資源割り当ての解を逃しにくく，最終的により良い解を生成できることを確認した．

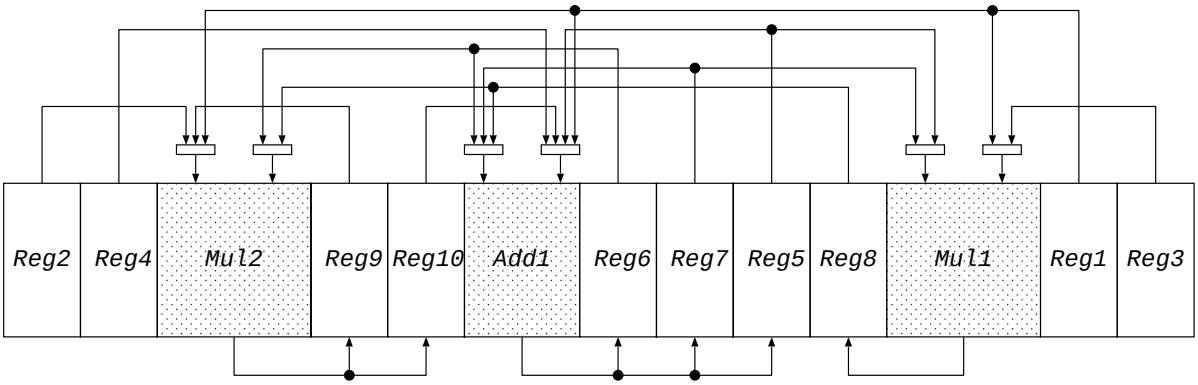
消費電力，実行時間，回路面積のそれぞれ優先順序を最も高くした実験結果である RT レベルアーキテクチャーとスケジューリングを図 7.3，図 7.4，図 7.5 及び図 7.6，図 7.7，図 7.8 に示す．

入力であるディジタルフィルタの Dependence Gragh に対して，提案手法では解を生成するのに多大な時間を要する．したがって，資源制約を変えた比較が容易に行なうこと

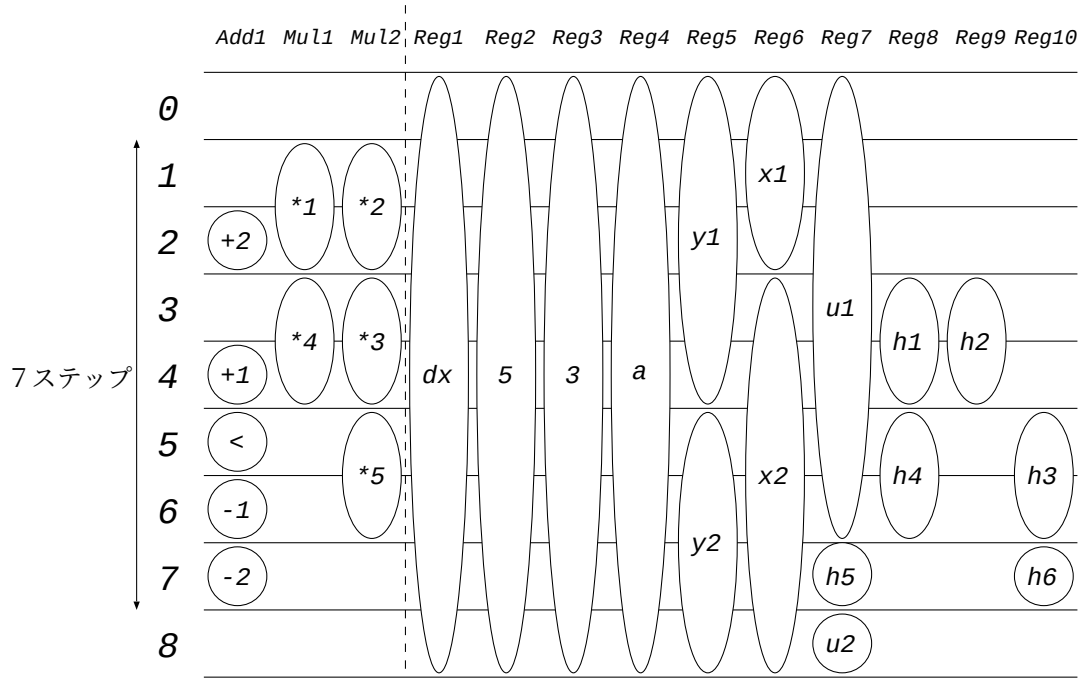
表 7.3: 設計結果の特性比較

設計制約	電力	時間	面積	逐次的手法
制約 1 電力優先	**329**	644	321	366
制約 1 時間優先	422	**392**	280	392
制約 1 面積優先	380	406	**250**	250
制約 2 電力優先	**307**	704	256	366
制約 2 時間優先	422	**392**	280	392
制約 2 面積優先	372	583	**233**	250
制約 3 電力優先	**307**	704	256	-
制約 3 時間優先	420	**572**	251	-
制約 3 面積優先	372	583	**233**	-
制約 4 電力優先	**538**	640	317	-
制約 4 時間優先	639	**500**	308	-
制約 4 面積優先	611	600	**254**	-

ができないので，改良点としては，大きな入力にも対応させることである．

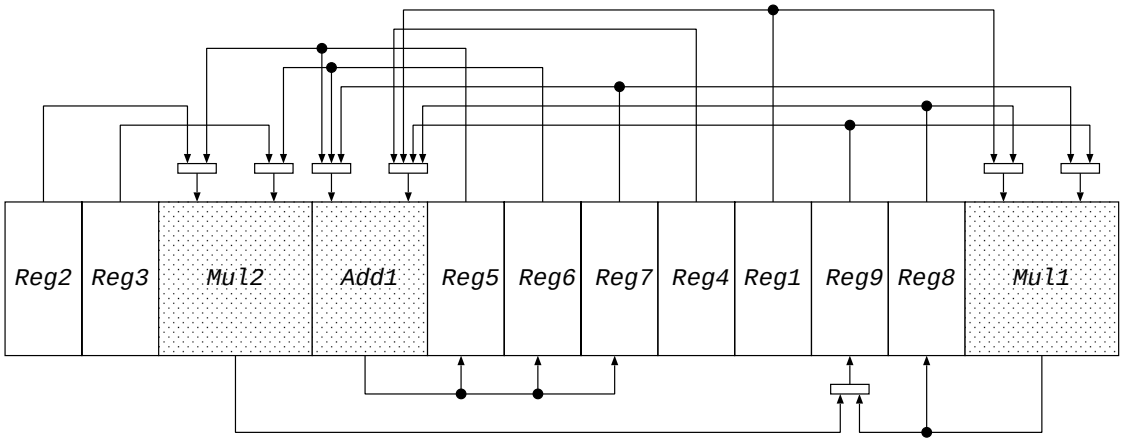


(a)RT レベルアーキテクチャ

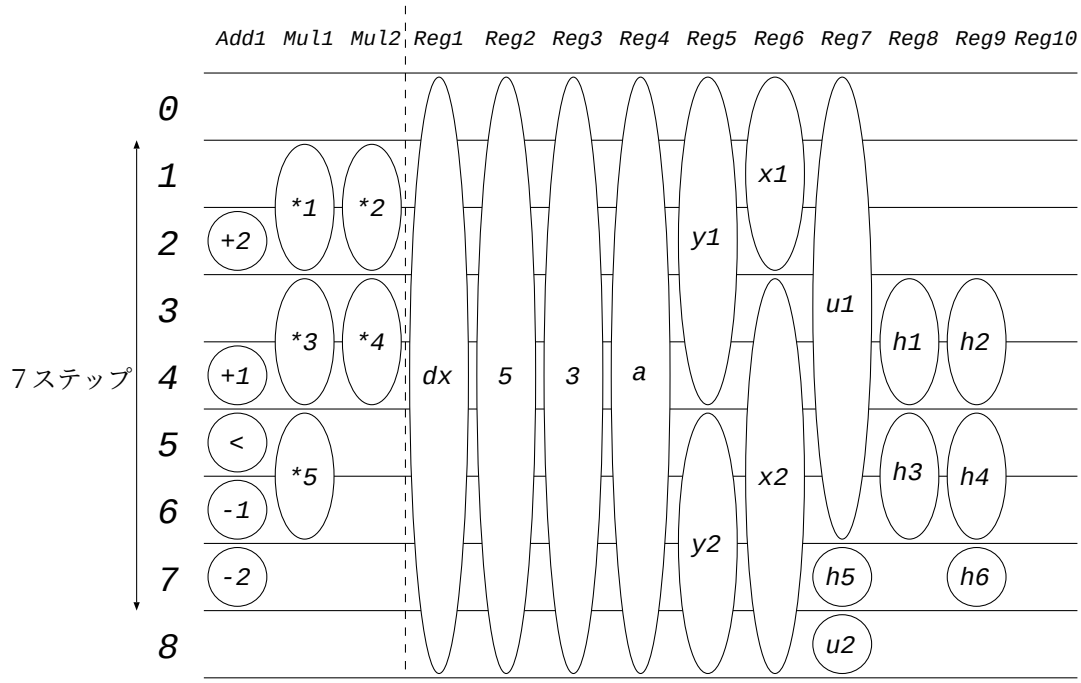


(b) スケジューリング

図 7.3: 制約 1；実験結果 (消費電力優先)

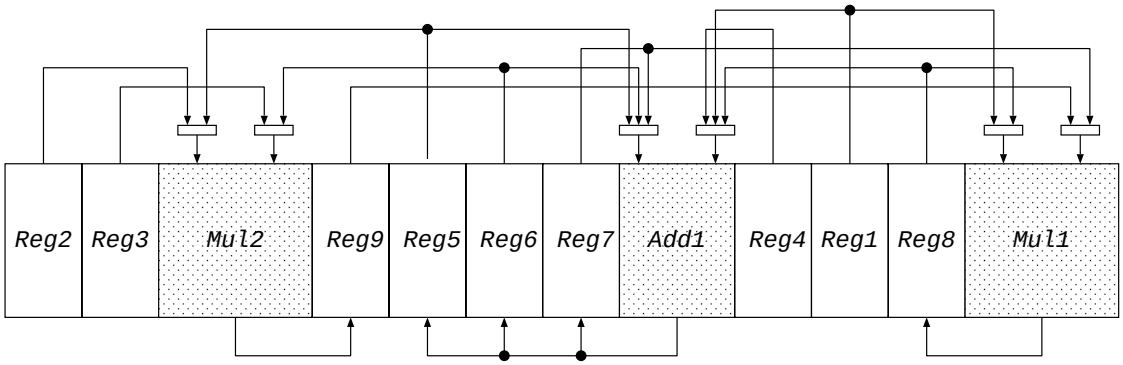


(a)RT レベルアーキテクチャ

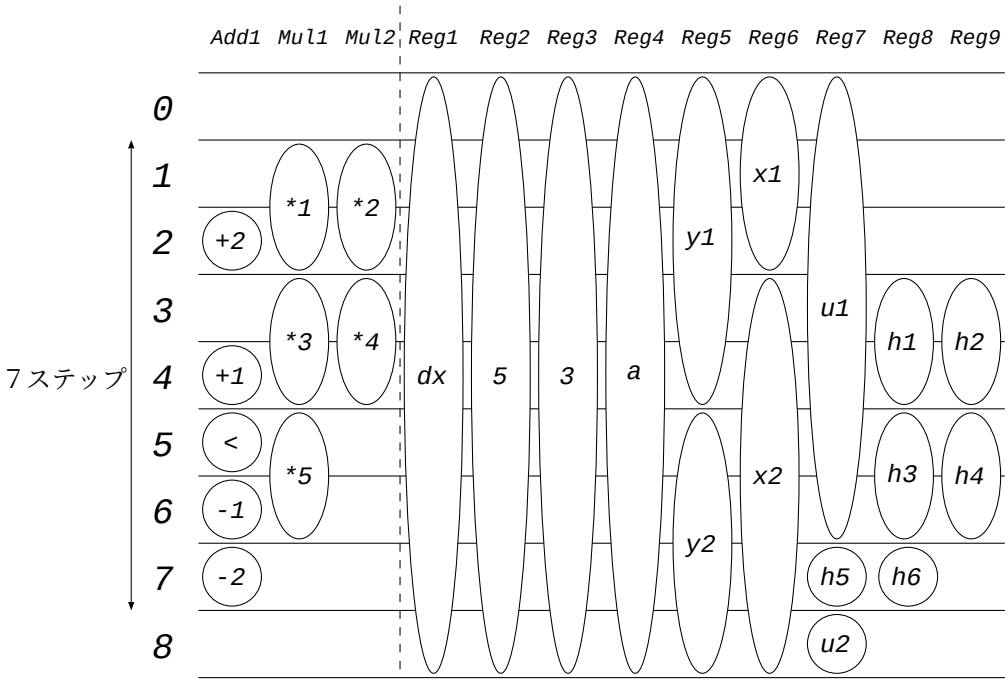


(b) スケジューリング

図 7.4: 制約 1,2；実験結果 (実行時間優先)

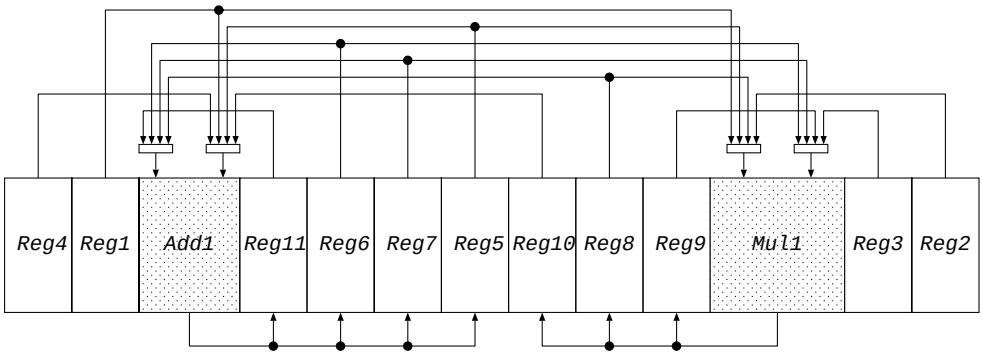


(a)RT レベルアーキテクチャ

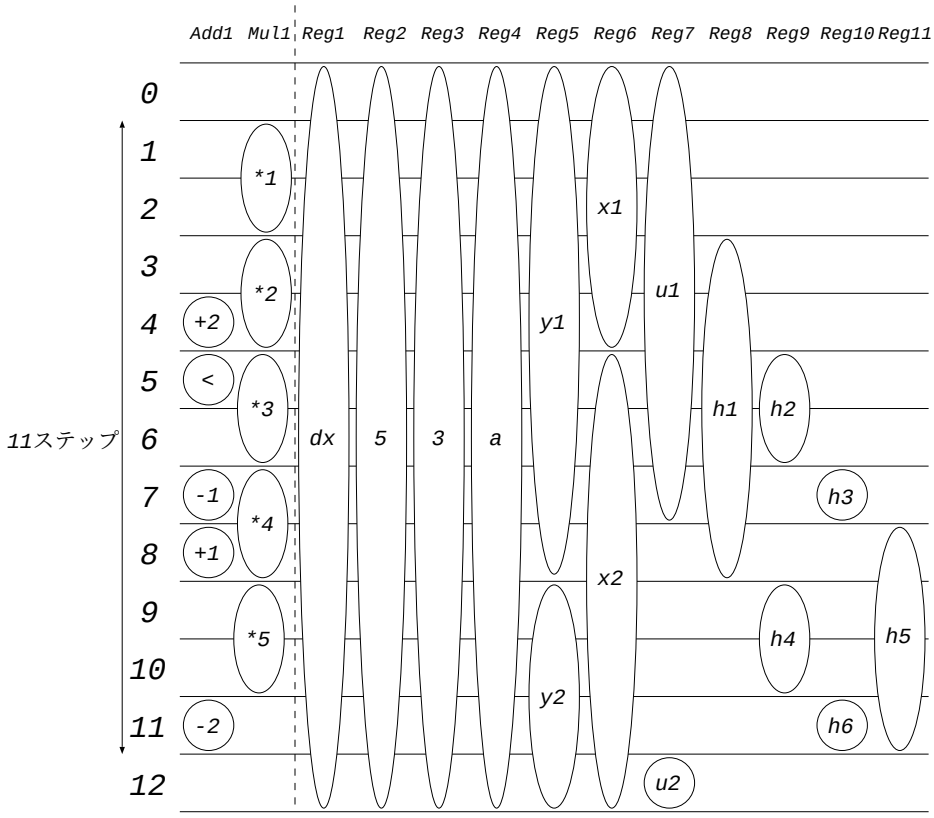


(b) スケジューリング

図 7.5: 制約 1；実験結果 (回路面積優先)

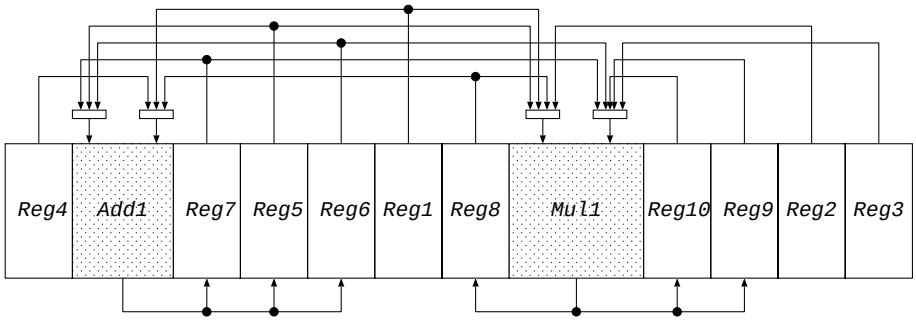


(a)RT レベルアーキテクチャ

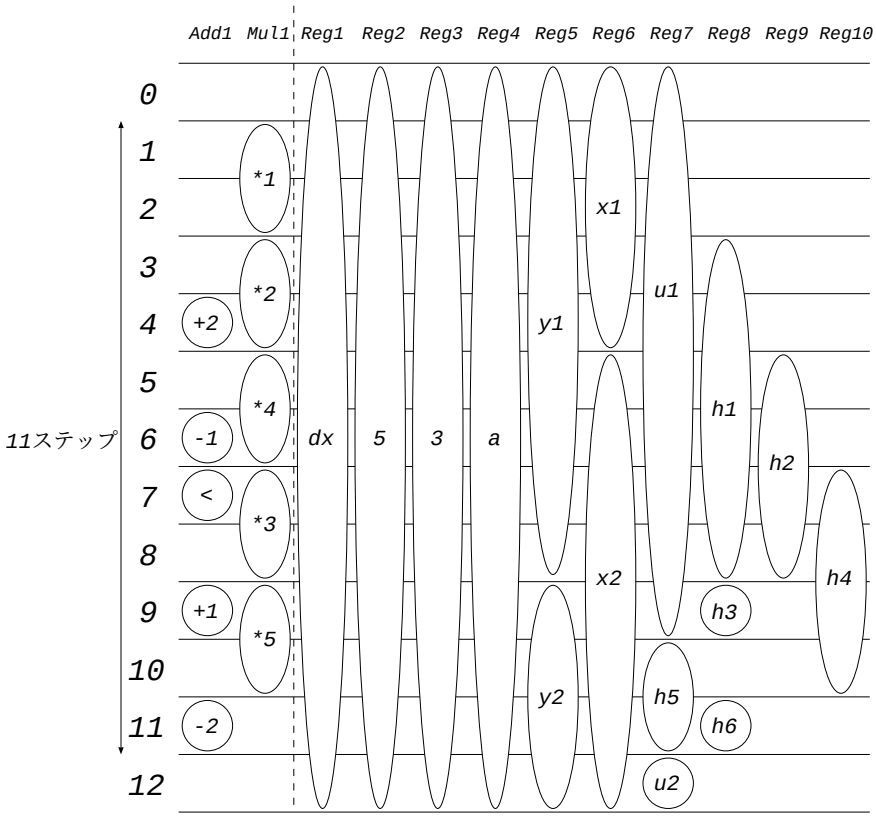


(b) スケジューリング

図 7.6: 制約 2,3；実験結果 (消費電力優先)

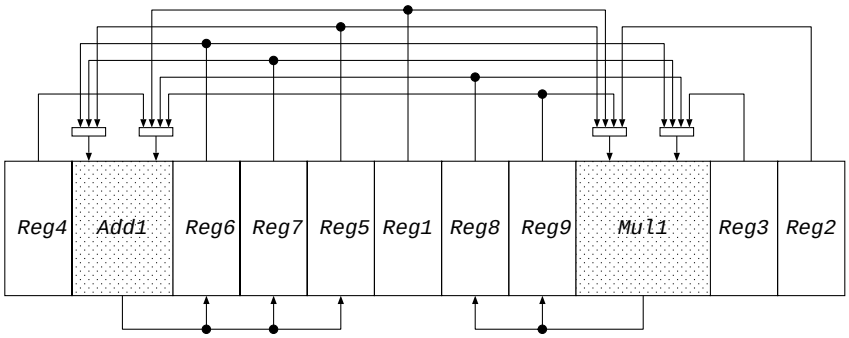


(a)RT レベルアーキテクチャ

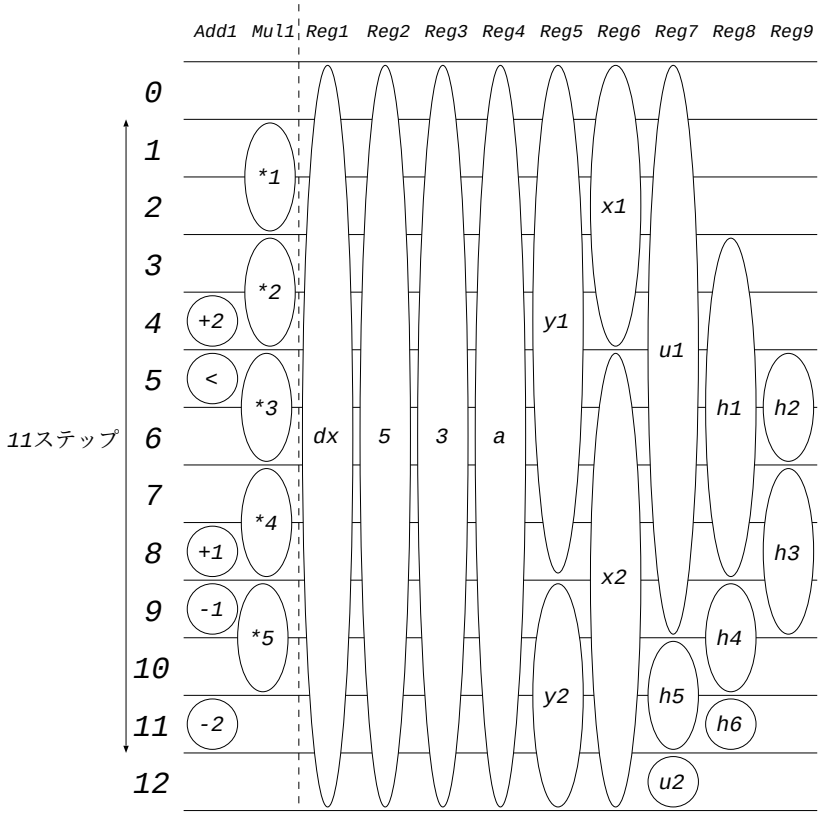


(b) スケジューリング

図 7.7: 制約 3；実験結果 (実行時間優先)



(a)RT レベルアーキテクチャ



(b) スケジューリング

図 7.8: 制約 2,3；実験結果 (回路面積優先)

第 8 章

むすび

8.1 まとめ

これまでのデータパス合成問題を対象とするヒューリスティックな手法の多くは，コントロールステップ数，演算器やレジスタなどの資源数，資源間の結線数最小化が目的であった．しかし，配線に起因する実行時間，消費電力が大きいため，高位合成でその評価を行う必要がある．本研究は，回路面積，実行時間，消費電力を最小化を目的としたスケジューリング，資源割り当てといった高位データパス合成問題とモジュール配置問題を同時に取り扱うデータパス合成手法を提案した．提案手法では，資源割り当て空間優先探索による分枝限定法を用いている．資源割り当てがスケジューリングに対して同時実行性や実行順序に関する制約を生成し，レイアウトに対して結線情報等を生成する．その際に，スケジューリングではコントロールステップ数，1 次元レイアウトでは配線に関する総配線長，最大信号遅延，重み付き総配線長の評価を基に回路面積，実行時間，消費電力の集積システムの評価を優先順序で行う．この評価を分枝限定法の枝刈りに利用している．本研究の提案手法を微分方程式の例を用いて設計実験を行い，提案手法は回路面積，実行時間，消費電力の評価に対して，最適解または最適解に近い解を生成することが確認できた．また，結線数を最小化にする高位合成を行なったあと，回路面積，実行時間，消費電力のそれぞれを最小化するレイアウトを行なう逐次的手法では最終的には解を悪化させてしまう．この考えのもとに，提案手法では高位データパス問題とモジュール配置問題を同時に取り扱うことで，最終的な解としては逐次的手法と比べて，優れた解を生成することを確認し有効性を実証した．

8.2　今後の課題

提案した資源割り当て空間の分枝限定法による探索では，基本的には全探索と同じであるため，大きな入力に関しては膨大な計算時間を要し，現実的時間内に解を出すことが非常に困難である．より効率的な資源割り当て空間探索手法の検討が今後の課題として残されている．

謝辞

本研究を行なうにあたり，日頃から温かくご指導いただいてくれた金子峰雄助教授ならびに田湯智助手に深く感謝致します．また，有益な御助言や御討論いただいた金子研究室の皆様に感謝致します.

参考文献

[1] P.Michel, U.Lauther, P.Duzy, "The synthesis approach to digital system design," Kluwer Adademic, 1992.

[2] J.P.Weng, A.C.Parker, "3D scheduling: High–Level Syntheis with floorplaning," Proc. 28th Design Automation Conf., ACM/IEEE, 1991.

[3] Y–M.Jiang, T–F.Lee, T–T.Hwang, Y–L.Lin, "Performance–Driven Interconnection Optimization for Microarchitecture Synthesis," Trans. on CAD, Vol.13, No.2, pp137–149, 1994.

[4] Y.Nishio, M.Kaneko, S.Tayu, "Assignment Based Approach to High Level Synthesis for Net Relevant Design Criteria", Technical report of IEICE, VLD98–147, pp.49–56, 1999.

[5] S.Sato, "Simulated Quenching: a New Placement Method for Module Generation," Proc. ICCAD, pp.538–541, 1997.