論　文

弱可検査性のための設計目標抽出を利用したデータパス高位合成

東村　剛嗣[†*]　　井上美智子[†]　　藤原　秀雄[†]

High-Level Synthesis for Weakly Testable Data Paths Using Design Objective Extraction

Takeshi HIGASHIMURA[†*], Michiko INOUE[†], and Hideo FUJIWARA[†]

あらまし　非スキャン設計のためのテスト容易性尺度である弱可検査性を考慮したデータパス高位合成手法を提案する．筆者らはこれまで，弱可検査なデータパスの高位合成に関し，合成後のデータパスが弱可検査となるような，ハードウェア要素共有に対する制約に関する十分性を示し，この十分性を満たす制約を設計目標として考慮する高位合成法を提案している．本研究では，まず合成前の動作記述であるデータフローグラフから合成後のデータパスが弱可検査となるための十分条件である設計目標の抽出手法を提案し，高位合成の主な処理であるスケジューリング，バインディングに関して，設計目標と面積をともに考慮する発見的手法を提案する．提案した手法を繰り返し適用することで時間制約のもとで面積が小さくかつ弱可検査なデータパスを合成する手法を提案する．

キーワード　高位合成，弱可検査性，データフローグラフ，データパス，VLSI テスト

1. まえがき

近年の VLSI の高集積化，大規模化に伴い，回路のテストはますます重要かつ困難な問題となってきている．テストのための費用を削減する有効な手段として，設計の初期段階からテスト容易性を考慮することが考えられる．また，VLSI の大規模化に対応できる支援技術として，抽象度の高い動作記述からレジスタ転送レベル（RTL）の回路を合成する高位合成の研究が盛んに行われている．本論文では，テスト容易性を考慮した高位合成法を考察する．

テスト容易性として，本論文ではデータパスの弱可検査性 [1] を考える．弱可検査なデータパスでは，データパス中の各レジスタに対して，外部入力から何らかの値が設定できること，外部出力で何らかの値が観測できることが保証される．このテスト容易性は順序回路テスト生成アルゴリズム（ATPG）を用いてテスト生成を行うデータパスの非スキャン設計のためのものである．

順序回路 ATPG のためのテスト容易性を考慮した高位合成法としては，これまで部分スキャン設計のための手法が多く提案されている [5]〜[9]．これらは，データパス中のフィードバックループをスキャンレジスタを用いて切断し，テスト容易性を向上させるものあり，スキャンレジスタ数を最小化するための多くの発見的手法が提案されている．しかし，スキャン設計では，面積オーバヘッドが大きい，テスト実行時間が長い，回路の通常動作時のクロック速度でのテスト実行（at-speed テスト）ができないなどの問題があり，近年，スキャン設計を行わないテスト容易化設計法が提案されている [1], [10].

筆者らはこれまで，RTL データパスの非スキャン設計のためのテスト容易性尺度として弱可検査性を提案し，弱可検査なデータパスが高故障検出効率を得ることを実験的に示している．また，小さなハードウェアオーバヘッドで与えられたデータパスを弱可検査にするテスト容易化設計法を提案している [1].

文献 [2] では，動作記述であるデータフローグラフ（DFG）を解析し，DFG 上で可制御な要素と可制御でない要素をデータパスでハードウェア共有させることでデータパス全体を弱可検査にする手法を提案している．この中では，合成後のデータパスが弱可検査となるために十分なハードウェア要素共有に関する条

[†] 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–0101 Japan
[*] 現在，日本 IBM 株式会社

件が示され，更に高位合成の基本的な処理であるスケジューリング，バインディングに関し，弱可検査性のための十分条件を設計目標として考慮する手法を提案している．しかし，設計目標の具体的な抽出法は提案していなかった．

本論文では，まず，DFG から設計目標を抽出する方法を新たに提案する．更に，高位合成手法については文献 [2] で提案されたスケジューリング手法とバインディング手法を改善し，動作速度の制約下で設計目標と演算器，レジスタなどのリソースの個数（面積）をともに考慮する発見的手法を提案する．本論文で提案する手法では，設計目標の抽出と抽出された設計目標を設計制約とする合成を繰り返し行うことによって，動作速度，リソース数，弱可検査性を設計制約として同時に考慮し，動作速度，リソース数の最適性を損なわずにテスト容易性の向上を行うことを目的としている．

2. 諸 定 義

合成前の回路の動作記述であるデータフローグラフ及び合成後のデータパスの弱可検査性 [1], [2] について定義する．

2.1 データフローグラフ（DFG）

データフローグラフ（DFG）は有向グラフ $G = (V, E)$ である．節点は，外部入力，外部出力，演算，変数，又は，遅延である．遅延は，外部入力の系列に対し繰り返して処理を行うような動作を記述する際に，ある入力に対して計算された値を次の入力に対する処理で用いるときに値を保持することを意味する．有向枝 $(v_i, v_j) \in E$ は演算あるいは遅延 v_i が変数 v_j を生成する，変数 v_i を演算 v_j が利用する，又は演算 v_i の結果を遅延 v_j が保持することを表す．DFG に対して，外部入力節点を除く任意の節点はそれに入射する辺をもつこと，任意の変数から外部出力までの経路が存在することを仮定する．図 1 に 3rd Lattice Wave Filter の DFG を示す．節点 PI，PO はそれぞれ外部入力，外部出力を，＊，＋ でラベル付けされた節点は，それぞれ乗算，加算を，節点 D は遅延を表している．また，黒で示した節点は変数を表している．

2.2 データパスの弱可検査性

データパスは外部入力，外部出力，レジスタ，マルチプレクサ，演算器からなるハードウェア要素と，これらを接続する接続信号線から構成される(注1)．ハードウェア要素 H_1 の出力とハードウェア要素 H_2 の入力 X 間に接続信号線がある場合，$H_1 \rightarrow X$，又は，$H_1 \rightarrow H_2$ と表記する．また，演算器 M の入力の集合を $\mathcal{IN}_M$ と表し，演算器 M が M の入力 X の値をそのまま出力する関数をもつとき，X をスルー可能入力と呼び，$thru(X)$ と記す．

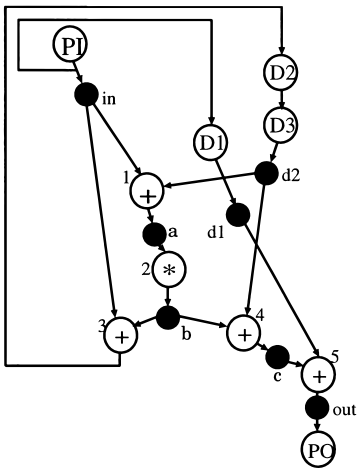


図 1　3rd lattice wave filter の DFG
Fig. 1　DFG of 3rd lattice wave filter.

データパス中の各ハードウェア要素について，弱可制御性，弱可観測性，及びデータパスについて弱可検査性を定義する．直観的には，ハードウェア要素 H の出力に，何らかの値を正当化することができるとき，H は弱可制御であるといい，また，H の出力の何らかの値は外部出力に伝搬することができるとき，H は弱可観測であるという．

［定義 1］ データパスの弱可制御性 [1]

弱可制御なハードウェア要素の集合は以下の条件を満たすハードウェア要素の最小集合 $\mathcal{H}_{wc}$ である．

（1） 任意の外部入力 PI は弱可制御である．すなわち，$PI \in \mathcal{H}_{wc}$.

（2） 任意のレジスタ，又は，マルチプレクサ H は，ある入力が弱可制御なハードウェアと接続していれば弱可制御である．すなわち，$\exists H_{wc} \in \mathcal{H}_{wc}[\ H_{wc} \rightarrow H] \Rightarrow H \in \mathcal{H}_{wc}$.

（3） 任意の演算器 M は，すべての入力，又は，あるスルー可能入力が弱可制御なハードウェア要素と接続していれば弱可制御である．すなわち，$(\forall X \in \mathcal{IN}_M\ [\exists H_{wc} \in \mathcal{H}_{wc},\ H_{wc} \rightarrow X]) \vee$

(注1)：定数は演算器等の組合せモジュールに含まれると考え，ハードウェア要素と考えない．

$(\exists X \in \mathcal{IN}_M\ [thru(X) \wedge \exists H_{wc} \in \mathcal{H}_{wc},\ H_{wc} \rightarrow X])$
$\Rightarrow M \in \mathcal{H}_{wc}$. □

［定義 2］ データパスの弱可観測性 [1]

弱可観測なハードウェア要素の集合は以下の条件を満たすハードウェア要素の最小集合 $\mathcal{H}_{wo}$ である．演算器の集合を $\mathcal{M}$ とする．

（1） 任意の外部出力 PO は弱可観測である．すなわち，$PO \in \mathcal{H}_{wo}$.

（2） 任意のハードウェア要素 H に対し，その出力が演算器以外の弱可観測なハードウェア要素と接続していれば，H は弱可観測である．すなわち，$\exists H_{wo} \in (\mathcal{H}_{wo} - \mathcal{M})[\ H \rightarrow H_{wo}] \Rightarrow H \in \mathcal{H}_{wo}$.

（3） 任意のハードウェア要素 H に対し，その出力が弱可観測な演算器 M の入力 X と接続しており，X がスルー可能入力，又は，X 以外の M の入力がすべて弱可制御であれば，H は弱可観測である．すなわち，$\exists M \in \mathcal{H}_{wo} \cap \mathcal{M}\ [\exists X \in \mathcal{IN}_M\ [H \rightarrow X] \wedge (\forall X' \in (\mathcal{IN}_M - X)[\exists H_{wc} \in \mathcal{H}_{wc}\ [H_{wc} \rightarrow X']] \vee thru(X))] \Rightarrow H \in \mathcal{H}_{wo}$. □

［定義 3］ データパスの弱可検査性 [1]

データパス中のすべてのレジスタが弱可制御，かつ，弱可観測ならば，そのデータパスは弱可検査である． □

データパスに対し，任意のレジスタから外部出力までの経路が存在すると仮定する．このとき，定義より明らかに，データパス中のすべてのレジスタが弱可制御ならば，すべてのレジスタは弱可観測となり，データパスは弱可検査である．よって，以下ではレジスタの弱可制御性のみを考える．

2.3 高位合成

データパス高位合成とは，動作記述である DFG を RTL の記述であるデータパスに変換することである．具体的には以下の部分問題を解く．

（1） スケジューリング：DFG 中の各演算をそれが実行される制御ステップに割り当てる．

（2） バインディング：DFG 中の演算を演算器に，変数をレジスタに割り当てる．演算器とレジスタ間の相互接続関係を決定し，必要があればマルチプレクサの割当てを行う．

3. 弱可検査性を考慮した高位合成

3.1 概　　略

本論文では，与えられた DFG と動作速度に関する時間制約から，少ないハードウェアで，かつ弱可検査なデータパスを実現する高位合成法を提案する．ここでは，簡単のため，各演算の実行遅延は 1 制御ステップとする．また，ある演算に割り当てることのできる演算器の種類は一意に決定できるものとし，各遅延にはそれぞれ専用のレジスタを割り当てる．データパスを構成するハードウェア要素のうち演算器とレジスタをリソースと呼ぶ．データパスの面積は，それに含まれるリソースに要する面積に支配されると考え，以降では各リソース数を最小にすることを考える．

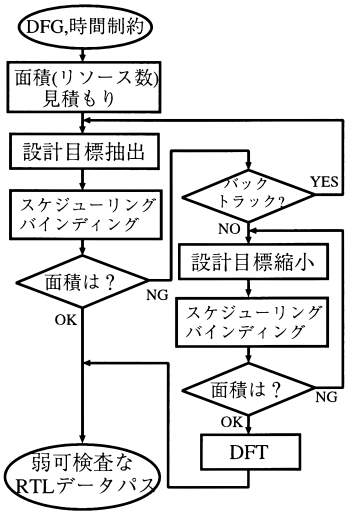


図 2 弱可検査なデータパスの合成
Fig. 2 High-level synthesis for weakly testable data path.

本手法のフローを図 2 に示し，その概要を述べる．まず最初に必要な各リソース数を見積もる．これには時間制約のもとでリソース数を最小にする既存のスケジューリング法であるフォースディレクティッド法 [3]（FD 法）を利用する．スケジュールされた DFG（SDFG）に対しては，ハードウェア要素の各型に対し，1 制御ステップで同時に必要とされる個数の最大値が，その SDFG から合成されるデータパスでの，その型のハードウェア要素の必要最小数である．

次に，合成後のデータパスが弱可検査となるためのリソース共有に関する十分条件である設計目標 [2] を DFG から抽出する．ここでは，設計目標の実現可能性を表す評価尺度を導入し，評価値の高い順に設計目標を抽出する．

次に，抽出された設計目標を設計制約とするスケジューリング，バインディングを行う．スケジューリ

ング完了後とバインディング完了後に，それぞれ必要なリソース数を求め，先に求めた見積り値を超える場合には，再び設計目標を抽出する．ただし，このバックトラックの回数には上限値を設け，上限値を超える場合は，設計目標から要素をいくつか削除し，縮小した割当て情報に対して合成を行う．合成されたデータパスが弱可検査でない場合は，文献 [1] で提案された，テスト容易化設計手法を用いてデータパスを弱可検査にする．以下では，まず，設計目標を定義し，設計目標の抽出，スケジューリング，バインディング，設計目標の縮小を説明する．

3.2 設計目標

DFG の要素に対して，データパスのハードウェア要素と同様の弱可制御性を考える．すなわち，DFG の要素に外部入力から何らかの値を設定できるとき，その要素は弱可制御であるとする．このとき，DFG の要素が弱可制御であれば，その要素が割り当てられたリソースも必ず弱可制御となる．また，DFG のある要素が弱可制御であれば，その要素とリソースを共有する要素もまた弱可制御であると考える．リソースの共有に関するある条件のもとで，DFG のすべての変数節点が弱可制御であれば，その条件を満たすデータパスのすべてのレジスタはすべて弱可制御となり，データパスは弱可検査となる．このような，リソースの共有に関する弱可検査性のための十分条件を設計目標といい以下のように定義する [2].

割り当てられるリソースの型が同一である演算又は変数からなる集合を共有集合と呼ぶ．また，共有集合の集合を割当て情報と呼ぶ．割当て情報は，それに属する各共有集合に対し，共有集合内のすべての要素が同じ演算器又はレジスタを共有する条件を表す．ある割当て情報に対して，ある DFG の要素が二つ以上の共有集合に属するならば，それらは一つの共有集合にすべきである．このことから，割当て情報に属する共有集合は互いに素であるとする．

［定義 4］ DFG の弱可制御性 [2]

DFG$G = (V, E)$ と割当て情報 $\mathcal{B}$ に対し，以下の条件を満たす節点の最小集合を $\mathcal{G}_{wc}$ とする．

（1） 任意の外部入力 I_g は弱可制御である．すなわち，$I_g \in \mathcal{G}_{wc}$.

（2） 任意の節点 v に対し，入射辺で隣接するすべての節点が弱可制御であれば，v は弱可制御である．すなわち，$\{u \in V | (u, v) \in E\} \subseteq \mathcal{G}_{wc} \Rightarrow v \in \mathcal{G}_{wc}$.

（3） 割当て情報 $\mathcal{B}$ に属するある共有集合 B に，弱可制御な要素が存在すれば B に属するすべての要素も弱可制御である．すなわち，$\exists B \in \mathcal{B}[\exists v \in B[v \in \mathcal{G}_{wc}] \Rightarrow B \subseteq \mathcal{G}_{wc}$.

このとき，$g \in \mathcal{G}_{wc}$ なる要素 g は弱可制御である．

□

［定義 5］ DFG の弱可検査性 [2]

DFG G と割当て情報 $\mathcal{B}$ に対し，G のすべての変数節点が弱可制御であるとき，2 項組 $(G, \mathcal{B})$ は弱可検査であるという． □

DFG G に対し，$(G, \mathcal{B})$ が弱可検査となる割当て情報 $\mathcal{B}$ を満たすデータパスは必ず弱可検査となる．このように，合成後のデータパスが弱可検査となるための十分条件である割当て情報を設計目標と呼ぶ．

3.3 設計目標の抽出

一般に，与えられた DFG に対して設計目標となるような割当て情報は複数存在する．リソース数を最小に抑えながら弱可検査性を達成するには，実現されやすい設計目標を抽出する必要がある．本論文では，設計目標の実現の容易さを示す尺度である重複可能度を導入し，重複可能度の小さい，すなわち，実現の容易な設計目標を順に抽出する発見的手法を提案する．

3.3.1 割当て情報の重複可能度

2 演算，又は 2 変数がリソースを共有するためには，それらが同じ制御ステップを利用しないことが必要となる．ここでは，時間制約から演算，変数が割り当て可能な制御ステップの範囲を求め，同じ制御ステップを利用する可能性を考える．

時間制約及びデータの依存関係を守りながら，DFG 上の演算をできるだけ早い時刻の制御ステップへ割り当てるスケジューリング（ASAP）と，できるだけ遅い時刻の制御ステップへ割り当てるスケジューリング（ALAP）から，演算が実行可能な制御ステップの範囲が求められる．ASAP で割り当てられる制御ステップから ALAP で割り当てられる制御ステップまでを演算の割当て可能範囲と呼ぶ．ある変数が生成される制御ステップ（生成時刻）から，その変数が最後に使用される制御ステップ（消滅時刻）までの範囲をその変数のライフタイムという．ただし，変数が生成される制御ステップとは，その変数を生成する演算が実行された次の制御ステップとする．また，ASAP によって求まるライフタイムを LT_{ASAP}，ALAP によって求まるライフタイムを LT_{ALAP} と表す．LT_{ASAP} の生成時刻から，LT_{ALAP} の消滅時刻までを変数の割当て可能範囲と呼ぶ．

2 演算あるいは変数が，同じ制御ステップに割り当てられる可能性を重複可能性と呼ぶ．2 演算に関する重複可能性は，割当て可能範囲が重複すれば 1，重複しなければ 0 である．ただし，両 2 演算の割当て可能範囲の大きさがともに 1 で重複する場合は ∞ とする．2 変数に関する重複可能性は，割当て可能範囲が重複すれば 1，割当て可能範囲が重複しなければ 0 である．ただし，両 2 変数それぞれの LT_{ASAP}，LT_{ALAP} の四つのライフタイムに共通の重複部分があれば，∞ とする．重複可能性の値は，2 演算又は 2 変数の利用する制御ステップが，1 のとき重複する可能性があること，0 のとき必ず重複しないこと，∞ のとき必ず重複することを表す．

共有集合 B_i に対して，B_i 中のすべての要素対 $\{v_p, v_q\} \subseteq B_i$ の重複可能性の和を共有集合の重複可能度とする．また，割当て情報中のすべての共有集合の重複可能度の和を割当て情報の重複可能度と呼ぶ．

［例］ 演算 1，2，3 に対して，割当て可能範囲がそれぞれ，$[1,3]$，$[2,4]$，$[4,5]$ であるとする．変数 a，b，c に対して，割当て可能範囲がそれぞれ，$[1,3]$ ($LT_{ASAP} = [1,2]$，$LT_{ALAP} = [2,3]$)，$[3,5]$ ($LT_{ASAP} = [3,4]$,$LT_{ALAP} = [4,5]$)，$[4,5]$ ($LT_{ASAP} = [4,5]$,$LT_{ALAP} = [4,5]$) であるとする．このとき，演算 1，2，演算 2，3 の重複可能性はともに 1，演算 1，3 の重複可能性は 0 であり，共有集合 $(1,2,3)$ の重複可能度は 2 である．また，変数 a，b の重複可能性は 1，変数 b，c の重複可能性は ∞ である．よって，割当て情報 $\{(1,2,3),(a,b)\}$ の重複可能度は 3，割当て情報 $\{(1,2,3),(b,c)\}$ の重複可能度は ∞ となる．

3.3.2 設計目標抽出法

重複可能度が小さい順に設計目標を抽出する発見的手法を提案する．まず，重複可能度が最小の設計目標を抽出するグリーディな発見的手法を示す．設計目標の抽出は，初期値が空集合である割当て情報が設計目標となるまで，すなわち，割当て情報に対して DFG が弱可検査となるまで，以下に示す拡大を繰り返す．ここで，割当て情報の拡大とは以下のいずれかである．現在の割当て情報を $\mathcal{B} = \{B_1, \cdots, B_i, \cdots, B_m\}$ とする．

- ある $B_i \in \mathcal{B}$ に要素を追加して，$B_i + \{v\}$ に置き換える．ここで，v は現在の割当て情報に対して弱可制御でなく，かつ，割り当てられるリソースの型が B_i 中のすべての要素と同一である．
- B に共有集合 (v_i, v_j) を追加する．ここで，現在の割当て情報 $\mathcal{B}$ に対し，v_i は弱可制御あり，v_j は弱可制御でない要素である．また，v_i，v_j が割り当てられるリソースの型は同一である．

可能な拡大の中から，以下の拡大を選択する．

（1） 拡大後の重複可能度が最小であるものを選択する．複数ある場合は，

（2） 新たに弱可制御となるリソースの型の個数が最多であるものを選択する．複数ある場合は，

（3） 新しく弱可制御となる節点数が最多であるものの中から任意の拡大を選択する．

設計目標抽出後，それに続きくスケジューリング，バインディングでリソース数見積りを達成できない場合，次の設計目標を抽出する．この場合，前回の抽出での最後の拡大を取り消して，次に優先度の高い拡大を行い，設計目標が得られるまで拡大を繰り返す．

3.4 スケジューリング

設計目標を考慮したスケジューリング法を提案する．遅延節点を削除した無閉路な DFG に対するスケジューリングを行う．提案手法では設計目標が設計制約となるよう DFG を拡張し，次に，この拡張 DFG に対して FD 法を拡張した手法でスケジューリングを行う．以降，DFG 及び拡張 DFG 上の経路の長さを，その経路上に現れる演算数とする．

設計目標中の共有集合に属する 2 要素がハードウェア要素を共有するためには，2 要素を異なる制御ステップに割り当てることが必要である．そこで，DFG 上の演算間に有向枝を追加して，演算の実行順序に制約を加え，共有集合に属する 2 要素が必ず異なる制御ステップに割り当てられるようにする．各共有集合に対し，以下のように有向枝の追加を行う．

共有集合が演算からなる場合，共有集合に属する 2 演算で，重複可能性が 1 である演算間の一方の向きに演算制約枝と呼ぶ有向枝を追加する．演算制約枝 (v_i, v_j) は，演算 v_i が実行される制御ステップより後の制御ステップで演算 v_j が実行されることを示し，2 演算 v_i，v_j は必ず異なる制御ステップに割り当てられる．演算制約枝の向きは両方向を考慮する必要があるが，ここでは簡単のため，2 演算のうち，外部出力までの最長経路長が大きい方を始点とする．

共有集合が変数からなる場合，共有集合に属する 2 変数で，重複可能性が 1 である 2 変数に対し，一方の変数を利用する各演算と他方の変数を生成する演算間に変数制約枝と呼ぶ有向枝を追加する．変数制約枝

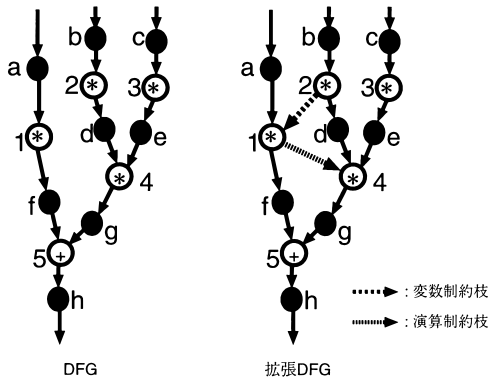


図 3　拡張 DFG
Fig. 3　Extended DFG.

(v_k, v_l) は，演算 v_k が実行される制御ステップ以降の制御ステップで演算 v_l が実行されることを示す．変数 v_k を利用する各演算と変数 v_l を生成する演算間に変数制約枝を追加すると，v_l は v_k が消滅した以降に生成されるので，v_k と v_l は必ず異なる制御ステップに割り当てられる．変数制約枝の向きに関しても，簡単のため，2 変数のうち，外部出力までの最長経路長が大きい方の変数を利用する演算が始点となる向きとする．

［例］ 図 3 に設計目標 {{1,4},{b,f}} に対する例を示す．演算に関する共有集合 {1,4} に対しては演算制約枝（1,4）を，変数に関する共有集合 {b,f} に対しては変数制約枝（2,1）を追加する．

FD 法では，DFG 又は部分的にスケジュールされた DFG に対して，ある演算をある制御ステップに割り当てる処理を繰り返し行う．各繰返しでは，ASAP と ALAP をもとにフォースという尺度を演算と制御ステップの各対に対して計算する．本手法では，ASAP と ALAP を求める際に，演算制約枝，変数制約枝を考慮するように拡張する．ASAP と ALAP が求まった後のフォースの算出方法は同様である．

3.5　バインディング

レジスタバインディング，演算器バインディングでは設計目標を最優先して行う．演算器バインディングでは内部接続コストも考慮する．

まず，各遅延にそれぞれ専用のレジスタを割り当てる．次に，変数をレジスタに割り当てる．変数を頂点とし，ライフタイムが重複しない変数間，すなわち，同一のレジスタに割当て可能な変数間に辺を設けたグラフをレジスタコンパティビリティグラフ（RCG）と呼ぶ．レジスタバインディングは，RCG に対するクリーク分割として解くことができる．ここでは，まず，変数に関する共有集合に対し，その共有集合に属するすべての変数をマージして一つの頂点とする．ここで，マージとは，隣接する 2 頂点 u，v を一つの頂点 w に置き換え，u，v がともに隣接する頂点と w 間に辺を加える操作である．スケジューリングの結果，設計目標中の同じ共有集合に現れる変数のライフタイムは必ず重複しないので，RCG では同じ共有集合内の任意の 2 変数間に必ず辺が存在する．最後に，マージ後の RCG に対して最小クリーク分割 [4] を行い，分割後の各クリークに対し，レジスタを割り当てる．

演算器バインディングでは，まず，演算器の型ごとに変数と同様にして演算コンパティビリティグラフ（OCG）を作成する．次に，設計目標中の演算に関する共有集合に対し，RCG と同様のマージを行う．演算に関する共有集合に対するマージ後の OCG に対し，以下のようにクリーク分割をして演算器バインディングを行う．クリーク分割は OCG から極大なクリークを逐次的に選択することで行う．極大クリークの選択では，マルチプレクサなどの内部接続コストを抑えるために，入力又は出力となるレジスタを共有するような演算が同じクリークに属することを優先する．

3.6　設計目標の縮小

設計目標の変更回数が制限回数を超えた場合に設計目標を縮小する．ここでは，既に抽出された設計目標から，最小個の要素を削除して実現可能な割当て情報を得ることを目標とする．まず，最初に抽出された設計目標に対し，以下で述べる縮小を行い割当て情報を得る．ここで，縮小とは，設計目標に属するある共有集合から要素を一つ削除することである．得られた割当て情報に対し合成を行い，合成結果がリソース数見積りを達成できなかった場合，更に次に抽出された設計目標に対して縮小を試みる．これを，リソース数見積りを達成するまで繰り返す．リソース数見積りが達成できない場合は，既に縮小された割当て情報を更に縮小する．割当て情報が空集合となるまで縮小されれば，見積もられたリソース数で合成できるので，必ずリソース数見積り内での合成が行える．

設計目標の縮小法を説明する．まず，縮小前の設計目標（又は割当て情報）に対して，リソース数見積りを達成できない原因となる演算制約枝，又は変数制約枝を求める．このような制約枝は，リソース制約のもとでスケジューリングを行った場合，時間制約を超え

るスケジューリング結果の原因となるような経路上にあると考えられる．このような経路を探すために，リソース制約のもとで時間を最小化する発見的手法であるリストスケジューリング法 [11]（LS 法）を利用する．LS 法は，リソース制約に矛盾しない範囲で，第 1 制御ステップから順に演算を割り当てていく．ある制御ステップに DFG で定める依存関係に矛盾せずに割当て可能な演算数が，リソース制約を超える場合，ある優先度に従って，優先度の高い演算から順に割当てを行う．このとき，リソース制約による演算間の依存関係を表す有向枝（制約依存枝と呼ぶ）をその制御ステップに割り当てられた演算から割り当てられなかった演算への向きに挿入する．スケジューリングされた DFG から，経路上のどの演算もリソース制約に矛盾しない範囲で他の制御ステップに割当てを変更することができないようなクリティカル経路を求める．以下，便宜上，第 1 ステップの前に外部入力節点が割り当てられ，最終ステップの後に外部出力節点が割り当てられると考える．具体的には，クリティカル経路は，外部入力を始点とし外部出力に至る以下の有向枝だけからなる経路である．

- 制御ステップの境界をたかだか一つしか超えない有向枝，又は，
- 制御ステップの境界を二つ以上超える有向枝 (v_1, v_2) であり，$v1$ と $v2$ の間の制御ステップには必ず $v2$ を終点とする制約依存枝の始点が存在するもの．

［例］ 図 4 (a) は，時間制約 4 でスケジューリングされた DFG が，加算器のリソース数見積り 1 を達成していない例である．有向枝 $(3,5)$ は演算制約枝である．変数節点は簡単のため省略している．同図 (b) は同じ DFG をリソース制約のもとでスケジューリングし，制約依存枝 $(4,5)$ を追加した結果である．ここで，有向枝 $(3,5)$ の終点 5 は，制約依存枝 $(4,5)$ の終点であり上記のクリティカル経路上の有向枝の条件を満たしている．この例では，二つの経路 $(PI1, 1, 3, 4, 5, 6, PO1)$ と $(PI1, 1, 3, 5, 6, PO1)$ がクリティカル経路となり，演算制約枝 $(3,5)$ がリソース数見積りを達成できない原因となる制約枝であると考えられる．

クリティカルパス経路上の制約枝に対応する演算及び変数の中から，設計目標から削除した場合重複可能度が最小となる要素を削除する．条件を満たす制約枝がない場合は，設計制約に現れるすべての要素のうち，それを削除することにより重複可能度が最小となるものを削除する．

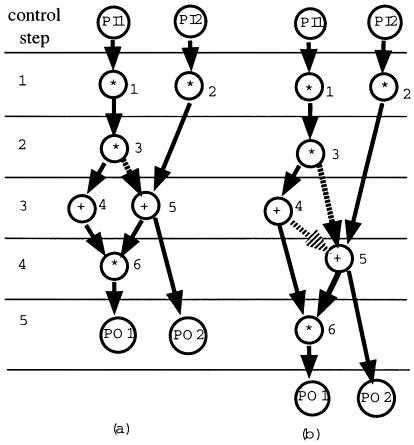


図 4　スケジューリング例：(a) 時間制約下，(b) リソース制約下

Fig. 4　Example of scheduling: (a) under time constraint, (b) under resource constraint.

4. 実験結果

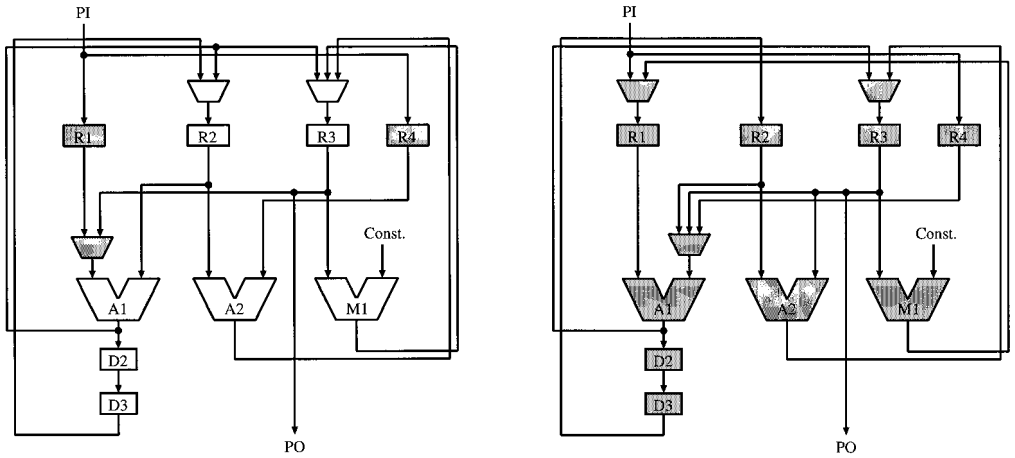
提案した高位合成手法の有効性を評価するために，提案手法を高位合成システムの評価用ベンチマークとして利用されている 4 種類のディジタル信号処理プロセッサ 3rd Lattice Wave Filter（3rdLWF）（図 1），4th IIR Cascade Filter（4thIIR），4th Jaumann Wave Filter（4thJWF），5th Elliptic Wave Filter（5thEWF）の DFG に対する適用結果を表 1 に示す．表 1 の *time constraint* は時間制約となる制御ステップ数，*design objective* は本手法で合成した場合を○，設計目標抽出を行わずに合成した場合を×で表す．*#reg*，*#add*，*#mul* は各 DFG の合成結果におけるレジスタ，加算器，乗算器の個数，*#backtrack* はバックトラックが起こり設計目標を変更した回数である．*weak testability* は合成したデータパスが弱可検査である場合を○，そうでない場合を×で表す．

また，合成したデータパス（ビット幅：8 bit）の VHDL 記述を Mentor Graphics 社の論理合成ツール AutoLogic II でゲートレベルネットリストに変換し，Sunrise 社の順序回路テスト生成ツール TestGen を用いてテスト生成を行った．表 1 の *fault eff.*，*CPU* はそれぞれ，故障検出効率，テスト生成時間を表す．使用した計算機は SunUltra2（CPU: 300 MHz×2，Memory: 256 MB）である．

高位合成に要する時間は，設計目標抽出に要する時間を含めてすべて数秒以内であった．実験結果による

表1 実験結果
Table 1 Experimental result.

circuit	*time constraint*	*design objective*	data path *#reg*	data path *#add*	data path *#mult*	*#back-track*	*weak testability*	*fault eff.* [%]	*CPU* [s]
3rdLWF	4	×	4	2	1	–	×	23.29	17754
		○	4	2	1	0	○	99.88	44
	5	×	4	1	1	–	×	24.44	15988
		○	4	1	1	0	○	99.93	13
4thIIR	6	×	7	2	3	–	×	24.48	36725
		○	7	2	3	0	○	99.74	385
	8	×	7	2	2	–	×	39.35	29164
		○	7	2	2	0	○	99.88	117
4thJWF	8	×	8	2	2	–	×	18.38	33158
		○	8	2	2	1	○	99.93	49
	11	×	8	2	1	–	×	20.08	33120
		○	8	2	1	0	○	99.90	221
5thEWF	15	×	10	4	2	–	×	15.11	60212
		○	10	4	2	0	○	99.38	1057
	20	×	10	3	1	–	×	15.55	50502
		○	10	3	1	0	○	99.77	30



(a) 設計目標を考慮しない場合　　(b) 設計目標：{{in,c},{1,5}}

図5 合成された 3rd lattice wave filter のデータパス
Fig. 5 RT-level data path of 3rd lattice wave filter.

と，すべての回路において，本手法により，設計目標を抽出しない場合と同数の演算器数及びレジスタ数で弱可検査性を満たす合成が可能となっている．ほとんどの場合において，最初に抽出された設計目標を用いた合成が可能となっており，抽出される設計目標の実現可能性は高いものであると考えられる．設計目標変更のバックトラックが生じているのは時間制約が 8 制御ステップのときの 4thJWF の合成時で，そのバックトラック回数は 1 回のみである．

弱可検査でないデータパスでは故障検出効率が総じて低い値であるのに対し，弱可検査なデータパスの故障検出効率はいずれも 99%台であり，テスト容易性として弱可検査性の有効性が示されている．

よって，本手法により，リソース数の最適性を損なわないような設計目標が抽出され，その設計目標により，弱可検査なデータパスが合成できていることが示されているといえる．

また，3rdLWF（図 1）に対し，時間制約が 4 制御ステップの場合の本手法による結果，設計目標を考慮しない場合の結果を図 5 に示す．抽出された設計目標は {{in,c},{1,5}} である．図 5 で黒色のハードウェア要素は弱可制御なハードウェア要素を表す．また，

A，M，R（D）でラベル付けされたハードウェア要素はそれぞれ加算器，乗算器，レジスタを表す．

5. むすび

本論文では，与えられたデータフローグラフと動作速度に関する時間制約から，少ないハードウェアで，かつ弱可検査なデータパスを実現する高位合成法を提案した．また，本論文で提案した高位合成手法をベンチマークに適用し，本手法の有効性を示した．

実験に用いたベンチマーク回路では，設計目標の縮小を行わずに合成を行うことができたが，設計目標の縮小が必要となるような規模の大きい回路での実験的評価が今後の課題である．更に，今後の課題として，条件分岐構造を含むコントロールデータフローグラフに対する弱可検査性を考慮した高位合成法の提案も挙げられる．

謝辞　本研究に際し，多くの貴重な意見をいただいた本学の増澤利光助教授，井上智生助手，並びに，情報論理学講座の諸氏に深く感謝します．本研究は一部，（株）半導体理工学研究センター（STARC）との共同研究，及び，文部省科学技術研究費補助金・基盤研究B(2)（代表者：藤原秀雄，課題番号 09480054）の研究助成による．

文　献

[1] K. Takabatake, M. Inoue, T. Masuzawa, and H. Fujiwara, “Non-scan design for testable data paths using thru operation,” Proc. Asia and South Pacific Design Automation Conf., pp.313–318, 1997.

[2] M. Inoue, K. Noda, T. Higashimura, T. Masuzawa, and H. Fujiwara, “High-Level Synthesis for Weakly Testable Data Paths,” IEICE Trans. on Information and Systems, vol.E81-D, no.7, pp.645–653, 1998.

[3] P.G. Paulin and J.P. Knight, “Force-directed scheduling for the behavioral synthesis of ASIC’s,” IEEE Trans. on CAD, vol.8, no.6, pp.661–679, 1989.

[4] C. Tseng and D.P. Siewiorek, “Automated synthesis of data paths in digital systems,” IEEE Trans. CAD, vol.5, no.3, pp.379–395, 1986.

[5] D.H. Lee and S.M. Reddy, “On determining scan flip-flops in partial-scan designs,” Proc. Intl. Conf. on CAD, pp.322–325, 1990.

[6] T.-C. Lee, W. Wolf, and N.K. Jha, “Behavioral synthesis for easy testability in data path scheduling,” Proc. Intl. Conf. on Computer Design, pp.616–619, 1992.

[7] S. Dey, M. Potkonjak, and R. Roy, “Exploiting hardware sharing in high-level synthesis for partial scan optimization,” Proc. Intl. Conf. on Computer-Aided-Design, pp.20–25, 1993.

[8] M. Potkonjak, S. Dey, and R. Roy, “Behavioral synthesis of area-efficient testable designs using interaction between hardware sharing and partial scan,” IEEE. Trans. on CAD, vol.14, no.5, pp.1141–1154, 1995.

[9] V. Fernandez and P. Sanchez, “Partial scan high-level synthesis,” Proc. European Design and Test Conf., 1996.

[10] S. Dey and M. Potkonjak, “Non-scan design-for-testability of RT-level data paths,” Proc. Intl. Conf. on Computer-Aided-Design, pp.640–645, 1994.

[11] M.C. McFarland, A.C. Parker, and R. Camposano, “Tutorial on high-level synthesis,” Proc. Design Automation Conf., pp.330–336, 1988.

（平成 10 年 3 月 30 日受付，7 月 27 日再受付）

東村　剛嗣　（学生員）

平 5 阪大・工・生産加工卒．同年（株）リコー入社．平 10 奈良先端科学技術大学院大学博士前期課程了．同年日本 IBM(株）入社．テスト容易化設計，高位合成の研究に従事．

井上美智子　（正員）

昭 62 阪大・基礎工・情報卒．平 1 同大大学院博士前期課程了．同年（株）富士通研究所入社．平 7 阪大大学院博士後期課程了．同年奈良先端科学技術大学院大学助手，現在に至る．分散アルゴリズム，グラフ理論，テスト容易化設計，高位合成の研究に従事．工博．IEEE，情報処理学会，人工知能学会各会員．

藤原　秀雄　（正員）

昭 44 阪大・工・電子卒．昭 49 同大大学院博士課程了．阪大工学部助手，明治大理工学部教授を経て，現在，奈良先端科学技術大学院大学教授．論理設計，テスト容易化設計，テスト生成，高位合成，フォールトトレランスの研究に従事．著書「Logic Testing and Design for Testability」（MIT Press）など．工博．IEEE，情報処理学会各会員，IEEE Fellow，IEEE Golden Core Member．