データパス合成のためのリソース割り付け法

宮崎 敏明†

複雑化，高機能化するVLSIの設計を支援するために抽象度の高い仕様からVLSIを合成するハイレベル・シンセシス技術の研究が盛んになってきた．ハイレベル・シンセシス技術の中心的課題の一つとして，データパス合成問題がある．データパス合成は，通常，入力仕様内の各演算をどの時刻で行うかを決定するスケジューリング問題と，限られたハードウェア資源に入力仕様内の演算／変数を割り付ける問題からなる．後者は，その対象によって，さらに機能ユニット／レジスタ／転送路割り付け問題に分けることができる．本論文では，それぞれの割り付け問題に共通に適応できるアルゴリズムを提案するとともに，個々の問題にそのアルゴリズムを適用する具体的な方法を例題を用いて説明する．また，いくつかの実験結果を示し，提案手法が過去に提案されているクリーク分割法を用いた方法よりも優れていることを示す．

Resource Allocation Method for Data Path Synthesis

Toshiaki Miyazaki†

We propose a new method based on Boolean manipulation for solving the resource allocation problems, which is known as one of the most important subtask to perform a data path synthesis system. In this paper, it is explained that the method is applicable to functional unit, register and interconnection allocation problems in common. It is also shown that the new method subsumes the conventional method utilizing the clique partitioning technique.

1. はじめに

データパス合成問題は，ハイレベル・シンセシスの一分野として，多くの研究がなされてきた．一般に，データパス合成は次のステップを踏んで行われる[1]．

1. 入力アルゴリズムを解析し，データフローグラフ（DFG）を作成する．
2. DFGを用いて各演算のスケジューリングを行う．
3. スケジューリング結果から同時刻に起こらない演算や使用されない変数をマージしてデータパスを生成する．マージの対象は，演算／変数／転送路があり，それぞれ機能ユニット／レジスタ／転送路（バス）の各割り付け問題として独立に解かれるのが一般的である．

図1に，文献2）から抜粋したDFGの例を示す．図中，線で区切られた各区間をコントロールステップ(c-step)と呼ぶ．スケジューリング問題とは，与えられたDFGに図のような線を入れることである．スケジューリングの結果，入力DFGがいくつかのc-stepで実行でき，また各c-stepで何を演算すべきかが決定される．さらに，変数に関して，演算結果をそのステップの終了時刻にレジスタ（ラッチ）に記憶するようにモデル化すると，太線で示した位置にレジスタが最低必要であることがわかる．

ここで，入力DFGをスケジューリングすると，同時刻に計算されるべき演算や，変数のライフタイム，すなわち，その変数値が最初に定義されてから，最後に参照されるまでの時間が求まる．しかし，この段階では演算の実行順序を規定しただけで，具体的なハードウェアを規定しているわけではない．データパスを合成するには，スケジューリングの結果を受けて，具体的なデータパスの要素である機能ユニット，レジスタ，転送路を設ける必要がある．これを以後「リソース割り付け問題」と呼ぶ．

リソース割り付け問題は，「同時刻に使用されない機能はできるだけ共有する」ことを評価基準とし解を見出すのが一般的である．これは，入力DFG内のすべての機能を要素とする集合を最小個の部分集合に分ける一種の最小カバーリング（Minimal covering）問題である．

Tseng[3]は，入力DFG内のすべての機能や変数を

† NTT LSI 研究所
NTT LSI Laboratories

ノードにとり，各ノード同士が同時に使用されることがない場合それらを結んだグラフを作成し，そのグラフを最小個のクリーク（完全グラフ）に分割することでこの問題を解いている．クリーク分割問題は，NP完全であることが知られており，文献 3) でも解法に際してヒューリスティクな方法が使われている．また，レジスタ割り付け問題に限って，レイアウト問題で広く知られている left-edge 法を適用した例もある[4),5)]．

本稿では，リソース割り付け問題の解をブール式処理を用いて，機械的に見つける計算機向きの手法について提案する．ブール式処理を転送路(バス)割り付け問題に適用した例は，いくつか発表されている[6),7)]．しかし，本稿で提案する手法は，転送路割り付け問題だけでなく，機能ユニット／レジスタ割り付け問題にも共通して適用できる汎用的なものである．

2章でアルゴリズムを提案したのち，提案アルゴリズムを機能ユニット／レジスタ／転送路の各割り付け問題に適用する具体的な方法を3章に示す．4章では，アルゴリズムの実現法について述べる．5章では，実験結果および考察を行う．6章では，まとめと今後の課題について触れる．

2. リソース割り付けアルゴリズム

2.1 定式化

文献 3) では，「同時に使用されないものはマージしてできるだけ少ない数のハードウェア資源（ファシリティ）に割り付ける」ことを基本とし，リソース割り付け問題を解いている．本論文でも，同じ方針をとる．具体的には，同時に使用されるものの組み合わせ（互いにマージできない組み合わせ）を与え，それ以外の組み合わせ（マージ可能な組み合わせ）をまず求め，それを用いて最小個のファシリティに対するリソース割り付けを実現する．

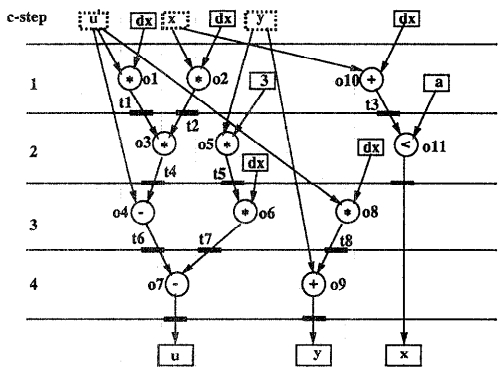


図 1 データフローグラフ（DFG）の例（文献 2) より抜粋）
Fig. 1 An example of a data flow graph (DFG) (See reference 2)).

最初に，用語の定義を行う．与えられた入力 DFG に含まれるすべての演算，変数，および，機能ユニット／レジスタ割り付け後に得られる各ファシリティ間を結ぶ転送路をリソースと呼ぶ．演算，変数，転送路それぞれに対してリソース集合 $R=\{r_1, r_2, \cdots, r_n\}$ (n はリソースの総数）を考え，非両立リソース・ペア (Incompatible Resource Pair : IRP) を以下のように定義する．

$$IRP \equiv \{(a_i, b_i) \mid \forall a_i, \forall b_i \in R;\ i=1, \cdots, m;\ m \textit{ is the total number of elements};\ \textit{Resource } a_i \textit{ and } b_i \textit{ cannot be merged.}\} \quad (1)$$

さらに，非両立リソース関数 (Incompatible Resource Function : IRF) を以下のように定義する．

$$IRF \equiv \begin{cases} \prod_{i=1}^{m} (a_i + b_i) \mid (a_i, b_i) \in IRP & \text{if } IRP \neq \phi \\ 1 & \text{if } IRP = \phi, \end{cases} \quad (2)$$

ここで，＋は論理和を表し，Π は論理積を表す．

アルゴリズム

1. 入力リソース集合 $R=\{r_1, r_2, \cdots, r_n\}$ を与え，割り付けたファシリティの数を表す変数を i とし，$i=1$ と初期化する．
2. 集合 R から関数 IRF を作成する．
3. 関数 IRF を非冗長積和形 (Irredundant Sum-Of-Products : ISOP) に展開する．ISOP から最小個のリテラルを持つ積項を一つ選ぶ．選んだ積項のリテラルを要素とする集合 MC を作成する．ここで，$IRF=1$ のときは，$MC=\phi_0$
4. $A_i=R-MC$ を計算し，A_i を保存する．ここで，A_i は解の一つであり，A_i 内の要素（リソース）は，互いにマージできる．
5. $R=MC$ とし，$R=\phi$ であれば，保存してある解を出力し終了する．その他のときは，$i=i+1$ とし，2へ戻る．

2.2 アルゴリズムの性質

非両立リソース関数 IRF は，負のリテラルを持たず，単調増大関数である．また，単調増大関数の ISOP は，一意に定まり，全主項の和に等しいことが知られている[12)]．提案したアルゴリズムは，関数 IRF の ISOP

から最小リテラル数の主項を選択していることになる．また，グラフ $G=(V, E)$ を考えると，本アルゴリズムはグラフ G の最小ノードカバー問題を解いているのと等価である．ただし，ノード集合 V は，リソースを表し，$V=R$（入力リソース集合）である．また，エッジ集合 E は，互いにマージできないノード（リソース），すなわち IRP を表している．さらに，これはグラフ G の補グラフ $G'=(V', E')$ の最大クリークを求める問題と考えることができる[8]．ここで，$V'=V$，$E'=\{(x, y) | x, y \in V';\ x \neq y;\ (x, y) \notin E\}$．

図 2 に例を示す．グラフ $G=(V, E)$ は，$V=\{a, b, c, d, e\}$，$E=\{(b, d), (c, d), (c, e)\}$ であるとすると，IRF =(b+d) (c+d) (c+e) である．ここで，その ISOP は，bc+cd+de であり，最小ノードカバー {b, c}(= MC) が求まる．本例では，{c, d} および {d, e} も最小ノードカバーであるが，どれを選ぶかはアルゴリズムの実現方法に依存する．このように最小ノードカバー（リテラル数最小の積項）が，複数存在する場合は，本アルゴリズムでは，任意の一つを選択することになる．このことは，最終解が一意ではなく，解の最小性が保証できないことを意味するが，解くべき問題が NP 完全であることを考えると，計算量削減のために，可能解の探索範囲をあえて限定することにした．

最初の繰り返しの結果，$A_1=V-\{b, c\}=\{a, d, e\}$ が，まず，解として求まる．次に，$V=\{b, c\}$ として，IRF を作成する．エッジ (b, c) は，存在しないから IRF =1．これは，次の最小ノードカバーが空であることを意味し，解は，$A_2=V-\phi=\{b, c\}$ となる．3回目の繰り返しでは，$V=\phi$ であるから，全体の解は，最終的に $\{a, d, e\}$, $\{b, c\}$ として求まる．図 2 は，上記の過程が G の補グラフ $G'=(V', E')$ の最大クリークを求める問題になっていることを示している．

また，アルゴリズム中 ISOP を求めるには，論理式に対する分配律を単純に適用していくことを考えると $2^{|IRP|}$（ここで，$|IRP|$ は IRP の個数）に比例する時間がかかる．しかし，実際には，まともに ISOP を求める必要はなく，最小個のリテラルを持つ積項が一つ求まればよい．これは 4 章で述べる二分決定グラフ (BDD) を用いた実現手法により効率良く求めることができる．さらに，実データでは，$|IRP|$ は，膨大にならないことも 5 章の考察で触れる．

3. 応　　用

提案したアルゴリズムを実際のリソース割り付け問題に適用するには，それぞれの問題にあった適切な関数 IRF を作成しなければならない．

3.1 機能ユニット割り付け

スケジューリングが終了した段階で，使用すべき機能ユニット (FU) の数は決定されるが，実際の割り付けが終わったわけではない．ここでは，入力演算を実際に最小個の FU に割り付けることを考える．まず，**表 1** に示すような両立表 (Compatibility Table : CT) を定義する．CT において，列は入力 DFG 内のすべての演算を表し，行は c-step を表す．また，それぞれの

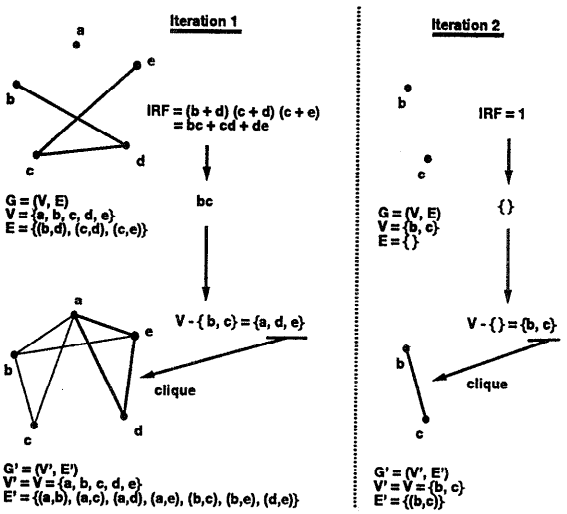


図 2 最小ノードカバー問題とクリーク分割問題との関係
Fig. 2 Relationship between minimum node covering and clique partitioning problems.

表 1 演算に対する両立表
Table 1 Compatibility table for operations.

type	mul	mul	mul	alu	mul	mul	alu	mul	alu	alu	alu
c-step	o 1	o 2	o 3	o 4	o 5	o 6	o 7	o 8	o 9	o 10	o 11
1	x	x								x	
2			x		x						x
3				x		x		x			
4							x		x		

演算が実行されている c-step に対応する欄に“x”を書き入れるものとする．表1に示した CT は，図1で示した DFG のスケジューリング結果から得られる．また，表には，各演算に対応した割り付けられるべき FU のタイプも示されている．“*mul*”と“*alu*”は，それぞれ乗算器と ALU (+, −, <) を表す．

CT を1行目から最終行まで順に横方向に見ていき，“x”が二つ以上存在するか，または，“x”の付いた演算の FU タイプと異なる FU タイプを持つ演算が存在する場合，それらは互いにマージできない．すなわち IRP である．例えば，表1中，演算 o1 に対して，(o1, o2), (o1, o10), (o1, o4), (o1, o7), (o1, o9), (o1, o11) が IRP である．このように，CT を利用することにより IRP を機械的に見つけることができ，関数 IRF を簡単に求めることができる．

割り付けアルゴリズムによって，すべての入力演算は，FU1＝{o4, o9, o10, o11}，FU2＝{o7}，FU3＝{o1, o3, o6}，FU4＝{o2, o5, o8} の四つの機能ユニットに割り付けられた．この機能ユニットの数は，もちろん，スケジューリングの結果と一致する．

3.2 レジスタ割り付け

表2に示すような CT を機能ユニット割り付け時と同様に作成する．ただし，列は入力 DFG 内のすべての変数とする．ここでは，各変数がその値を保持しておかねばならないすべての c-step に“x”を埋める．この CT は，各変数のライフタイムを示しており，一般に，「ライフタイムテーブル」とも呼ばれる．表2の11個の変数は，図1内に出現するすべての変数を表している．ここで，変数‘u’, ‘x’, ‘y’は，便宜上 c-step 0 で定義されるとし，また，定数‘3’, ‘a’, ‘dx’は，常時参照可能であると仮定している．11個の変数は，表2より作成した IRF を用いて，6個のレジスタ (u, x, y, R1, R2, R3) に割り付けられた．

図3に，機能ユニットおよびレジスタ割り付け結果をもとに描いたデータパスを示す．ここで，定数は，定数レジスタとして表現している．

3.3 転送路割り付け

転送路割り付けの目的は，機能ユニットとレジスタ割り付けの結果を受けて，最小個のバスや配線を見出すことである．

表 2 変数に対する両立表 (ライフタイムテーブル)
Table 2 Compatibility table for variables (Lifetime table).

c-step	u	x	y	t1	t2	t3	t4	t5	t6	t7	t8
0	x	x	x								
1	x		x	x	x	x					
2	x	x	x				x	x			
3		x	x						x	x	x
4	x	x	x								

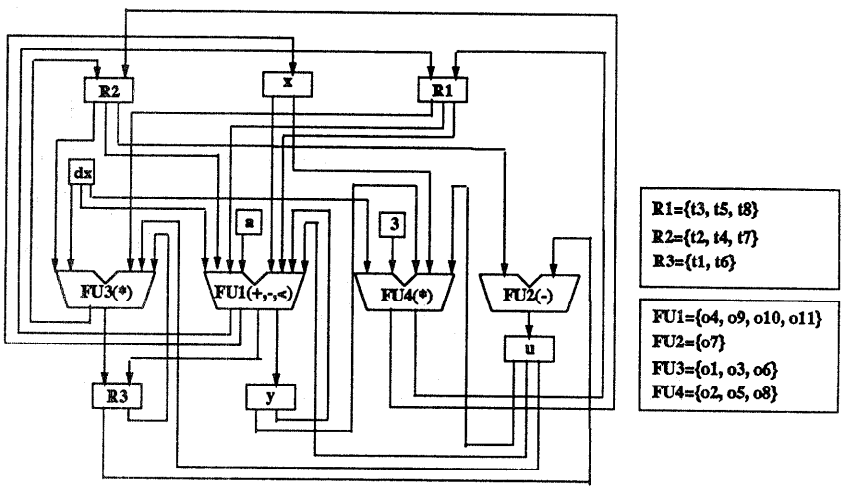


図 3 機能ユニットおよびレジスタ割り付け終了後に得られた図1に対するデータパス
Fig. 3 Data path for Figure 1 after the functional unit and register allocations.

最初に，前述の割り付け問題と同様に，CT を作成する．ここでは，出力が二つ以上ある FU やレジスタ等のファシリティを列にとり，c-step を行にとる．出力が一つでも使用されている c-step に“x”を書き入れる．**表 3** は，図 3 を用いて作成した CT である．本 CT より，IRF を作成し，提案アルゴリズムで処理すると，(x, R 1) および (u, R 3) の組がマージできる．これは，それぞれの組のファシリティの出力ポートから出ている配線が互いにマージできることを示しており，その結果二つのバスが割り付けられる．

図 4 に，転送路割り付け結果を経て最終的に合成されたデータパスを示す．複数入力のあるファシリティの入力ポートにはマルチプレクサを挿入している．

表 3 転送路に対する両立表
Table 3 Compatibility table for interconnections.

c-step	u	x	y	FU 3	R3	FU 4	R2	FU 1	R1
1	x	x		x		x		x	
2			x	x	x	x	x	x	x
3	x			x		x	x	x	x
4			x		x		x	x	x

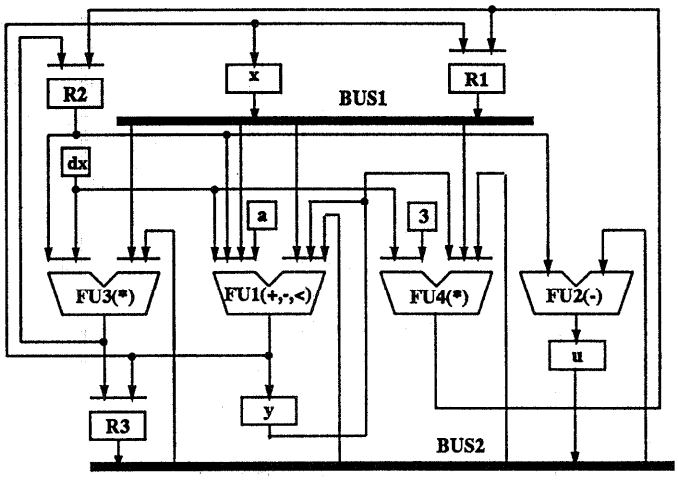


図 4 最終的に生成された図 1 に対するデータパス
Fig. 4 Final generated data path for Figure 1.

4. 実 現 方 法

提案したアルゴリズムは，UNIX マシン上に二分決定グラフ (BDD)[9] を用いて実現した．BDD は，大きなブール式をコンパクトに表現できる形式であり，近年，多くのブール式最適化手法で採用されている．**図 5** は，関数 IRF＝bc＋cd＋de を表した BDD の一例である．BDD は通常終端に 0 と 1 を示す二つの葉 (leaf) を持つ有向グラフである．

今，BDD 上で“1”-path とは，根から“1”-leaf へ至るパスであると定義する．また，“1”-path の「重み」を，そのパス上にある“1”-edge の個数であると定義すると，最小個のリテラルを持つ積項は，対応する BDD から最小の重みを持つ“1”-path を見つける問題となる[10]．例えば，積項 *bc* は，図 5 上で，太線で表したパスであり，その重みは 2 である．

これより，2.1 節で述べたアルゴリズム内で ISOP を実際に求める必要はなく，上記の BDD が作成できれば，最小個のリテラルを持つ積項を BDD 上のパス探索によって求めることができる．

また，BDD の大きさは入力変数の順序付けに大きく依存することが知られている[9]．ここでは，単純に変数の出現順とした．この順序付けの妥当性を探るために，予備実験として**表 4** に示したそれぞれのデータ

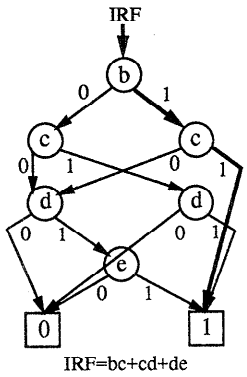


図 5 二分決定グラフ (BDD) の例
Fig. 5 An example of a Binary Decision Diagram (BDD).

に対してランダムに変数順序を決定した場合と比較した．その結果，出現順に変数の順序付けをしたほうが平均 70% 小さい BDD が作成できた．

5. 実験結果および考察

実験結果を，クリーク分割による方法[3]と比較して，表 4 に示す．表中，上方が提案手法による結果であ

り，下方がクリーク分割法による結果である．それぞれのデータは，Force-Directed 法[2]によりスケジューリングされた結果を用いている．

提案手法とクリーク分割法による割り付け結果に大きな違いはみられない．また，表はそれぞれの方法で問題を解いたときの時間と，問題に対応したグラフのエッジ数も合わせて記してある．エッジ数は，問題の複雑さと関係しており，特に提案手法に対するグラフのエッジは，式(1)で示した IRP の個数 ($|IRP|$) と同じである．また，提案手法については，BDD の大きさも示した．

大きな問題に対しては，提案手法がクリーク分割法に比べ高速であることがわかる．例えば，表中に太線で示したように，データ "EWF" と "Seki" の転送路割り付けにおいて，われわれの手法は約5倍クリーク分割法にくらべ高速である．ここで，V と E は，問題グラフ $G=(V, E)$ のノードとエッジであるとすると，クリーク分割法の計算量は，$O(E^2)+O(V^2E)+O(V^3)$ である[3]．一方，提案手法が採用している BDD は，最悪の場合，二分木となりその作成に要する時間は指数的に増加する．しかし，いったん作成された後は，探索等の操作は，BDD の大きさに対して線形であることが知られている[9]．

2.2 節で述べたように，提案手法で問題を解くことと，クリーク分割法でその問題の補グラフを解くこととは等価である．よって，クリーク分割法で大きな問題は，提案手法では小さな問題となる．言い替えると，リソース割り付けにおいては，提案手法で問題を解くほうがクリーク分割法で解くよりも，問題を小さくできることを示している．これは，表中で，問題の大きさを示すエッジの数を見ればわかる．これより，提案手法は，従来のクリーク分割法に比べ，リソース割り付けに対しては適した方法であると言える．

表 4 リソース割り付け手法に対する実験結果
Table 4 Experimental results of the resource allocation method.

	Proposed resource allocation											
	FU				Register				Interconnection			
data	#	time	edge	BDD	#[1]	time	edge	BDD	#[2]	time	edge	BDD
DiffEq[2]	3	1.1	10	28	6+3	2.6	28	28	9+0	4.6	362	165
Tseng[3]	3	1.4	9	22	6+0	2.9	37	69	6+5	5.5	135	144
EWF[2]	4	1.6	100	56	14+1	6.9	251	255	9+4	**67.7**	362	4296
Seki[11]	-	-	-	-	-	-	-	-	5+2	**231.1**	252	47598

	Clique partitioning[3]											
	FU				Register				Interconnection			
data	#	time	edge	BDD	#[1]	time	edge	BDD	#[2]	time	edge	BDD
Diffeq	3	0.2	45	-	6+3	0.1	27	-	9+0	3.8	214	-
Tseng	3	0.3	57	-	6+0	0.5	68	-	6+5	5.7	243	-
EWF	4	27.6	530	-	12+1	12.5	344	-	10+4	**428.3**	1849	-
Seki	-	-	-	-	-	-	-	-	5+2	**1163.3**	2523	-

(*Note*)
1) (Register)+(Constant register).
2) (Bus)+(dedicated wire).
3) time = elapse time [sec.]. (SparcStation 1+, 24MB).

6. おわりに

ブール式処理を使用したハードウェア資源の割り付けアルゴリズムを提案した．また，両立表という表現形式を考案し，提案したアルゴリズムが，機能ユニット，レジスタ，転送路割り付けに対して共通に適用できるようにした．

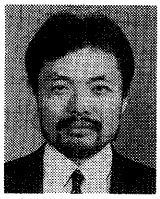
データパス合成は，入力データフローグラフのスケジューリングの結果に大きく依存する．提案した手法は，スケジューリング結果を入力としているが，スケジューリングそのものも検討する必要がある．

参考文献

1) McFarland, M. C., Parker, A. C. and Camposano, R.: Tutorial on IIigh-Level Synthesis, *Proc. 25th Design Automation Conference*, pp. 330-336 (1988).
2) Paulin, P. G. and Knight, J. P.: Force-Directed Scheduling in Automatic Data Path Synthesis, *Proc. 24th Design Automation Conference*, pp. 195-202 (1987).
3) Tseng, C. J.: Automated Synthesis of Data Paths in Digital Systems, Ph. D thesis CMU (1984).
4) Kurdahi, F. J. and Parker, A. C.: REAL: A Program for REgister ALlocation, *Proc. 24th Design Automation Conference*, pp. 210-215 (1987).
5) Tanaka, T., Kobayashi, T. and Karatsu, O.: HARP: Fortran to Silicon, *IEEE Trans. CAD*, Vol. 8, No. 6, pp. 649-660 (1989).
6) Mathialagan, A. and Biswas, N. N.: Optimal Interconnections in the Design of Microprocessors and Digital Systems, *IEEE Trans. Comput.*, Vol. C-29, No. 2, pp. 145-149 (1980).
7) Crover, D., Singh, H. and Kanda, N. K.: Datapath Optimization in Mordorn Computers, *INT. J. Electronics*, Vol. 53, No. 1, pp. 17-24 (1982).
8) Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, p. 387 (1974).
9) Bryant, R. E.: Graph-based Algorithms for

Boolean Function Manipulation, *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677-691 (1986).
10) Lin, B. and Somenzi, F.: Minimization of Symbolic Relations, *Proc. ICCAD-90*, pp. 88-91 (1990).
11) Sekikawa, H. et al.: An Efficient Algorithm for Interconnect Elements Assignment, *Trans. IEICE*, Vol. J 74-A, No. 7, pp. 1031-1040 (1991).
12) Muroga, S.: *Logic Design and Switching Theory*, *Reading*, John Wiley & Sons (1979).

(平成4年7月20日受付)
(平成5年4月 8 日採録)



宮崎 敏明（正会員）

昭和 56 年電気通信大学電気通信学部応用電子工学科卒業．昭和 58 年同大学大学院修士課程修了．同年日本電信電話公社入社．以来，厚木電気通信研究所（現 NTT LSI 研究所）において，信号処理 LSI 用 CAD, CAD フレームワーク，論理設計支援技術および上位設計支援技術の研究開発に従事．現在 NTT 伝送システム研究所伝送処理研究部主任研究員．IEEE，電子情報通信学会各会員．