設 計 自 動 化 71－8
（1994．6．24）

ブロック図によるHDLベース設計環境

野地保† 濱田英幸† 水本勝也† 高坂広之† 青木洋† 中村彰‡

†三菱電機(株) ‡長崎大学工学部

論理合成を主体としたトップダウン設計手法の一形態として，ブロック図による論理合成可能なハードウェア記述言語(HDL)ベース設計環境について述べる．定型的な機能は，ビット幅などのパラメータ入力により，論理合成可能な機能記述を自動生成し，新規の機能記述は論理合成可能なHDL記述へ対話的に変換する．これらをブロック図形式で接続することにより，HDL記述の知識が少ない設計者でも，論理合成を意図した機能設計が視覚的にできる．また論理合成からの後戻りが少なくなり，再利用設計が促進され，設計効率の向上が図れる．評価例では，従来のテキストベースの設計と比較して記述量を22%削減できた．

HDL Based Design Environment using Block Diagram

Tamotsu Noji† *Hideyuki Hamada*† *Katsuya Mizumoto*†
Hiroyuki Kosaka† *Hiroshi Aoki*† *Akira Nakamura*‡

† Mitsubishi Electric Corporation ‡ Nagasaki University

This paper describes top down design methodology by the use of HDL block diagram.This design environment supplies parameterized HDL templates from which new version of HDL description and its block component are generated by overriding default parameter values.Through the block diagram, it is easy to recognize functional designstructures .So, even novice user with HDL can design easily for logic synthesis.This approach avoids some mistakes in HDL description and promotes design re-use.The results show it reduces the amount of HDL description by hand to 78% of fully HDL description by hand.

1 はじめに

ASICの大規模化及び高性能化は，益々進んできている．しかし，ASICの開発期間はその規模の増大や性能向上につれて長くなるわけではなく，むしろ，短くなる傾向にある．このため，ASIC開発には大規模・高性能な論理回路を限られた資源及び期間で，如何に効率良く設計できるかの技術が求められている．この要求 を解決する方法としては，方式・機能設計レベルからレイアウトレベルまでの各工程間の後戻りが少なく，かつ設計効率の良い，論理合成技術を主体としたハードウェア記述言語(HDL)ベースのトップダウン設計手法[1]の確立と，過去の設計資産を容易にかつ有効に活用できる設計環境の構築がある．

本報告では，これら2つの解決策を兼ね備えたブロック図によるHDLベースの設計環境について報告する．また，本環境をKUE-CHIP[2),3)]の設計に適用し，その評価についても述べる．

以下，2章でブロック図によるHDL設計環境の背景，3章でその実現方式，4章でKUE-CHIPへの適用とその評価結果について述べる．

2 背景

従来の論理図入力により論理回路を設計していた設計者にとっては，テキストによるHDLベース設計手法は，ソフトウェア開発と同様にプログラミング技術が必要なため，その設計スタイルを変えることには大きな障壁がある．

また，HDLベースの設計では，設計データをHDLに統一できるため，設計資産の流用やデータベース化が容易という利点があり，論理回路をすべてHDL記述で設計する手法が急速に普及している．しかし，この手法では，(1)各機能間の相互関係が視覚的に判りづらいため，設計ミスが生じやすい，(2)設計者の要求する機能を持ったHDL記述があったとしてもビット幅や一部機能など(以下，ビット幅などと称する)の違いにより利用できないことが多いなどの問題点がある．このため，設計者はHDL記述を所要のビット幅などへ変更する必要がある．このとき，HDL記述した者が他の設計者であれば，その記述内容を理解するだけで多くの時間が費やされることもあり，場合によっては，新しく記述し直す設計者もいる．

これらのことから，従来の手法のままでは，必ずしも設計効率が向上するとは限らない．

このため，機能設計をテキストベースではなく，ブロック図，状態遷移図などによって，視覚的に設計できる手法[4)～6)]，定型的な機能は，ビット幅などのパラメータ値を設定するだけで，論理合成可能なHDL記述を自動生成する手法[6),7)]などが提案されている．

しかし，これらの手法の中には，(1)独自言語の採用により，標準のHDL[3)]を利用することができない[6)]，(2)機能設計の段階で論理合成の可否判定，合成結果の予測，ゲート数の見積り予測などの機能がない[4)~7)]，(3)論理合成可能なHDL記述を出力する[4)~7)]ものもあるが，意図した回路に論理合成されるとは限らない，(4)設計者が記述したHDL記述を論理合成可能な記述に変換する機能がない[4)~7)]，(5)ライブラリ化の方法が開示されていないため，一般の設計者は容易に追加・変更することが困難である[6)]，(6)従来の論理図とHDL記述を混在して設計することができない[4)~7)]などの問題点があり，機能設計からレイアウトまでトップダウンに設計することは困難である．

このような，課題を解決するため本環境では，トップダウン設計手法の一形態として，過去の設計資産を容易にかつ有効に活用でき設計効率の向上を図れる設計環境を提案する．

3 実現方式

本環境はトップダウン設計手法の一形態であり，本環境を適用する一般的な設計フローを図1に示す．また，本環境の構成を図2に示す．

3.1 実現する機能

(1) 機能単位のHDL記述(以下，HDLマクロ

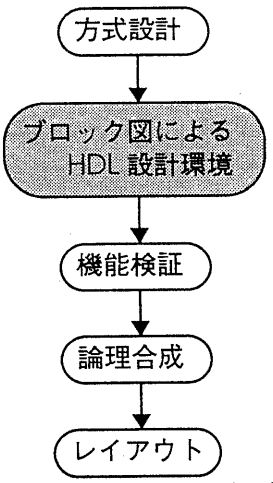

図1 設計フローにおける本環境の位置付け

と称す)をブロックとして表現することにより，テキストベースではなく，ブロックを組み合わせて視覚的に設計を行なう機能．これにより，HDLの言語仕様に関する知識が少ない設計者であってもブロックを組み合わせることで設計ができ，各機能間の相互関係は視覚的にしかも容易に把握できる．

(2) 定型的な機能のHDLマクロは，ビット幅などのパラメータ化を行ない，設計者がパラメータ値を設定するだけで，論理合成可能なHDL記述の自動生成を行なう機能．パラメータ化したHDLマクロの作成は，Verilog HDL，VHDLなどの標準的なHDLをベースとして行なう．これにより，定型的な機能のHDLマクロに関しては，設計者がHDLマクロの記述内容を変更するといった作業がなくなる．

(3) 合成予測[8]を適用し，設計者が新たに記述したHDL記述であっても，機能設計の段階で，論理合成向けへの変換，論理合成後の回路予測，ゲート数の見積り予測を行なう．これにより，論理合成やレイアウトからの後戻りが少なくできる．

3.2 ブロック図の入力形態

ブロック図の入力形態としては次の2種類があり，これらを併用して設計を行なう．

3.2.1 概略ブロック図を描いてからの入力

設計者は，直接概略のブロック図を描いてから，各ブロックに対して設計を行なう．この入力形式では，設計者が設計に対してあいまいな状態であっても，設計の概略となるブロック図を入力し，それから，各機能のブロックを詳細につめていくことができる．これにより，設計者の思考に近い形で設計することができる．

3.2.2 登録ブロック配置による入力

ライブラリデータベースには，HDLマクロ，

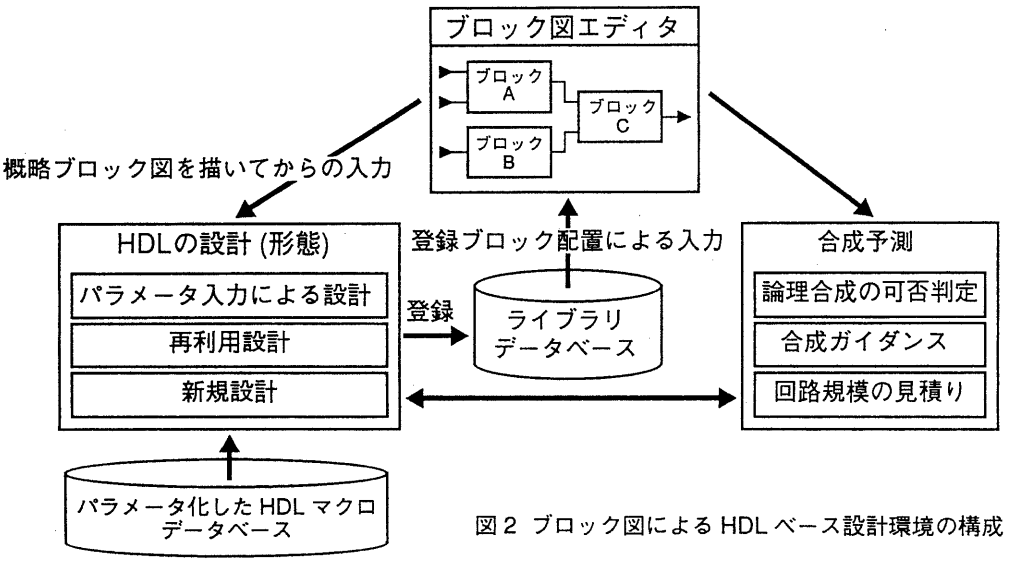

図2 ブロック図による HDL ベース設計環境の構成

論理図及びそれらのブロックが登録されている．この入力形式では，ライブラリデータベースに登録されているHDLマクロ及び論理図のブロックを配置することによって，設計を行なう．このため，設計者はブロックを描く必要がない．この入力形式は，予めブロックに対して詳細な仕様が決まっているときに有効である．

3.3 HDLの設計形態

ブロック図の各ブロックであるHDLマクロ及びHDL記述の設計には，再利用設計，パラメータ入力による設計及び新規設計の3つの設計形態がある．

データパス系の設計は，HDL記述をしなくとも，再利用設計やパラメータ入力による設計で，ほぼすべて実現できる．また，制御系の設計は新規設計により実現できる．

3.3.1 再利用設計

再利用設計は，過去の設計データや頻繁に使用する機能を論理合成可能な機能記述のHDLマクロとしてライブラリデータベースに登録して再利用する設計形態であり，設計効率の向上に効果的な設計手法である．HDLマクロをライブラリデータベースに登録する際には，登録するHDLマクロのブロックを自動的に生成し，HDL記述と共にライブラリデータベースに登録する．なお，再利用設計では，HDLマクロのみではなく，過去に設計した論理図も扱える．これにより，HDLマクロと論理図を混在して設計することも可能である．

3.3.2 パラメータ入力による設計

カウンタ，シフトレジスタ，ALUなどの定型的な機能については，ビット幅などのパラメータ化を実現し，設計者がパラメータ設定を行なうだけで，HDL記述とそのブロックが自動生成されるHDLマクロを開発した．自動生成されたHDL記述は論理合成可能な機能記述であり，設計者は、自動生成されたHDL記述の内容を確認できる．また，自動生成されたHDL記述に合成予測を適用することで，論理合成された場合の概略ゲート数を論理合成することなく見積れる．なお，頻繁に使用するパラメータ設定後の機能については，再度，HDLマクロとしてライブラリデータベースに登録すれば，再利用設計に用いることができる．このように定型的な機能に関してはHDL記述の自動生成が行なわれるため，HDLベースの設計に不慣れな設計者はもちろんのこと，熟練したHDLベースの設計者にとっても，設計期間の短縮及び設計負荷削減が実現できる．

パラメータ化したHDLマクロは，本環境独自の言語によって作成するのではなく，Verilog HDL，VHDLなどの標準的なHDLを利用して作成する．このため，HDLベース設計の経験がある設計者であれば，独自言語を覚えることなく，パラメータ化したHDLマクロを作成し追加することが可能である．

3.3.3 新規設計

新規設計は，設計者がHDL記述によって設計する形態であり，次のような場合である．

(1) 新たにHDL記述する場合．

(2) 既存のHDL記述を流用し，変更して用いる場合．

(3) HDLマクロとして登録されているHDL記述を流用し，変更して用いる場合．

新規設計は，ブロック図入力形態に合わせて，次の2つの設計形態から成る．

・概略ブロック図を描いてからの入力

設計者が描いたブロックをもとに，HDLで記述された雛型を生成する．なお，雛型には，module名，入出力宣言などが記述されている．設計者はその雛型に対してHDL記述を追加したり，他のHDL記述を読み込んで変更を行なう．

なお，新規設計で記述・変更したHDL記述に対しては，合成予測を適用することができる．このため，論理合成実行前に論理合成の可否，概略ゲート数の予測が可能である．

・登録ブロック配置による入力

設計者が新たに記述したHDL記述，または変

更したHDL記述をもとにブロックを自動生成する．そして，設計者が記述・変更したHDL記述と自動生成されたブロックを一緒にライブラリデータベースへ登録する．

3.4 合成予測

回路規模オーバ，意図しない回路への論理合成，論理合成向きでないHDL記述などによる論理合成からの後戻りを少なくし，論理合成の実行時間を削減する方法として，合成予測を適用した．

合成予測と実際の論理合成との実行時間の比較では，合成予測の方が100倍以上の速さで実行でき，設計効率の向上が図られる．

なお，適用した合成予測の機能は，論理合成の可否判定，合成ガイダンス及び回路規模の見積りである．

3.4.1 論理合成の可否判定

論理合成の可否判定は，設計者が新たに記述したHDL記述が論理合成可能か否かについて判定する．

可否判定は，次の2つのステップを満たさなければ，論理合成不可と判定する．

[Step1] HDL記述が論理合成ツールの記述仕様を満足していること．

[Step2] Step1が満足していれば，設計ノウハウをもとに作成されたHDL記述のテンプレートと合っていること．

これにより，設計者は論理合成を実行することなく，論理合成の可否を知ることができる．なお，検出する主な項目は次のとおりである．

(1) ライブラリに存在しないラッチ，フリップフロップの参照

(2) 不定値の代入

(3) メモリ素子　など

3.4.2 合成ガイダンス

合成ガイダンスは，設計者が記述したHDL記述を設計者の意図する回路に論理合成できるように変換する．変換作業は，HDL記述とその論理合成結果を設計ノウハウをもとにしたHDL記述のテンプレートに従って設計者と対話形式で行なわれるため，設計者はどの部分がどのように変換されたかを把握することができる．これにより，HDL記述に不慣れな設計者であっても，短期間で論理合成可能なHDL記述を作成することができる．

主な変換項目は次のとおりである．

(1) 不要なラッチとフリップフロップの除去

(2) プライオリティエンコーダとセレクタの入れ換え

(3) 非同期セット/リセットと同期セット/リセットの入れ換え　など

3.4.3 回路規模の見積り

回路規模の見積りは，HDL記述から論理合成後の回路を推定することにより，ゲート数を概算する．見積方法は，HDL記述を共通ライブラリで論理回路にマッピングし，それをテクノロジライブラリの内部ゲートに換算する．

論理合成後のゲート数との精度比較では±10%程度であり，実用範囲であると考えられる．

これにより，設計者は論理合成前に概略ゲート数が把握できる．

3.5 設計手順例

設計手順例として，図3のブロック図について述べる．

図4は，登録ブロック配置による入力及び概略ブロック図を描いてからの入力によって，各ブロックの配置/入力を行なったものである．

図4において，REG（ビット幅32のレジスタ），SEL（ビット幅32の2入力1出力のセレクタ），ALU(ビット幅32)，SFT REG（ビット幅32のシフトレジスタ）の各ブロックは，登録ブロック配置による入力であり，既にこの段階で各ブロックの詳細な設計は完了している．

一方，CNT（制御部）ブロックは，概略ブロック図を描いてからの入力であり，詳細な設計は行なわれていない．

図4では配線を行なっていないが，この段階で，ブロックの配線を行なってもかまわない．

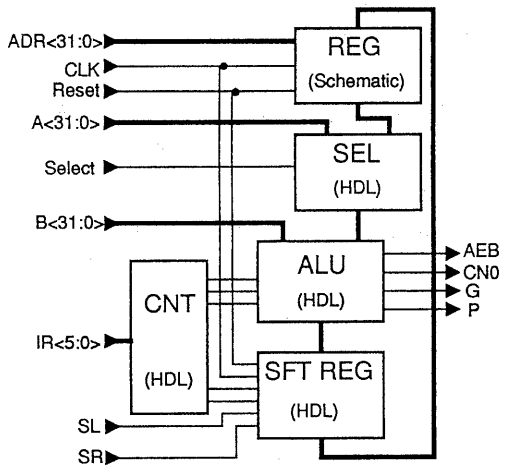

図3 ブロック図例

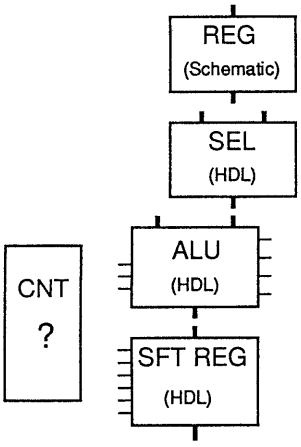

図4 ブロック配置/入力例

以下，設計手順を説明する．

(1) REGブロックは，過去に設計した論理図を用いた再利用設計である．

(2) SFT REGブロックは，パラメータ入力により設計を行ない，次の手順で設計する．

パラメータ化されているシフトレジスタのHDLマクロを選択し，図5のパラメータウインドウに対して，ビット幅を32，ブロック名（セル名）を SFT REG，Input Pin Order Output Pin Order で各入・出力ピンの並びを設定する．

図5の設定内容から，図6に示すビット幅32の論理合成可能なHDL記述とそのブロックが自動生成され，このブロックを用いてブロック図入力（配置）を行なう．このとき，必要であれば，合成予測により，概略ゲート数も把握できる．

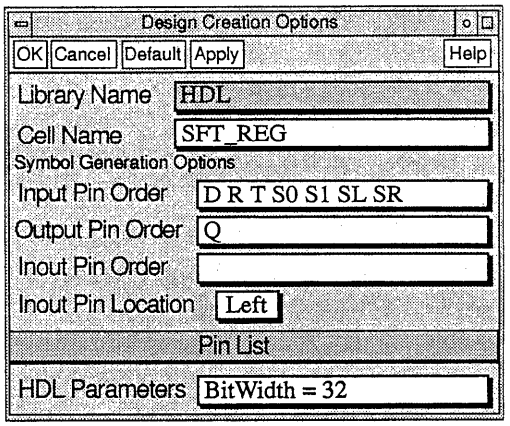

図5 パラメータ設定例

```
module SFT_REG(Q,R,S0,S1,T,SR,SL,D);
//    n-bit Bidirectional Universal
//    Shift Register with Direct Reset
parameter BitWidth = 32;
output [BitWidth-1:0] Q;
input R,  S0,  S1,  T,  SR,  SL;
input [BitWidth-1:0] D;
                    ⋮
wire [1:0] S = {S0,  S1};
reg [BitWidth-1:0] Q;
                    ⋮
always @(posedge T or negedge R)
 begin
   if (!R)
     Q = 0;
   else
     case (S) // ··· parallel_case full_case
       2'b10 : Q = ·····;
       2'b01 : Q = ·····;
       2'b11 : Q = D;
       2'b00 : Q = Q;
     endcase
 end
                    ⋮
endmodule
```

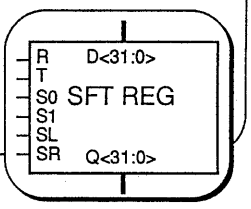

図6 自動生成された32ビットシフトレジスタのHDL記述とブロック

(3) SEL及びALUブロックは，SFT REGブロックと同様にパラメータ入力により設計を行なう．

(4) CNTブロックは，新規設計で行なうため，設計者が仕様に従ってHDL記述を行なう．既にCNTのブロックが作成されておれば，HDL記述の雛型が自動生成され，設計者はそれに対してHDL記述を追加する．

HDL記述が終了すれば，設計者は記述したHDLが論理合成可能か否かについて，合成予測を実行する．合成予測は，論理合成の可能性についてチェックし，論理合成が可能であれば，概略ゲート数を算出し，設計者に対してその結果を出力する．一方，論理合成が不可能の場合は，その箇所をエラーメッセージとして，設計者に出力する．

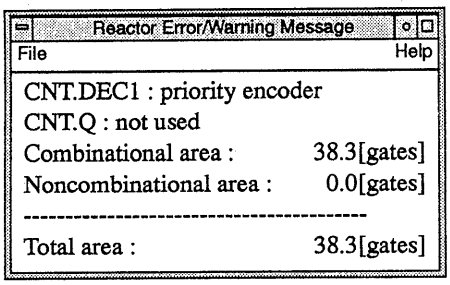

図7 合成予測結果例

図7は，CNTブロックに合成予測を適用した結果例である．CNTブロックの記述から，設計者はデコーダを設計したにもかかわらず，プライオリティエンコーダが合成されることを示し，記述中のQは宣言されているだけで使用されていないことを示している．また，概略ゲート数は，組み合わせ回路部分が38.3，順序回路部分が0，合計が38.3であることを予測している．

なお，合成予測機能は，対話的に予測し修正することも可能である．

(5) 各ブロックの設計が終了し，接続されていないブロック間を接続することにより，設計が完了する．

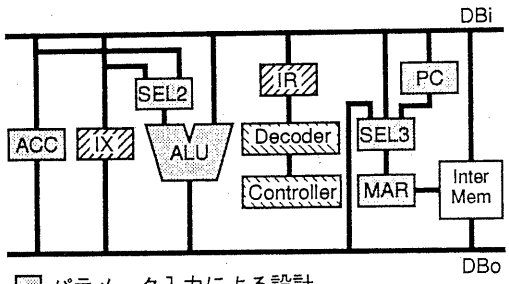

図8 KUE-CHIPの構成

4 適用と評価

KUE-CHIPの設計に本環境を適用した．その結果，設計者が新たにHDL記述を行なったところは制御系のみであり，データパス系は，再利用及びパラメータ入力による設計により，ブロックを組み合わせて，視覚的に設計できた．図8にKUE-CHIPの構成と各ブロックに適用した設計形態を示す．なお，図8においてIX,IRブロックは，パラメータ入力で作成したACCブロックの再利用である．

KUE-CHIPをテキストベースで設計した場合と本環境で設計した場合とのHDL記述のステップ数(;ごとに1ステップと換算)について表1に示す．HDL記述は，設計者により，同じ機能でも，記述ステップ数は異なる．このため，本環境の制御系のHDL記述のステップ数は，テキストベースで設計したときのHDL記述のステップ数を使用した．表1からKUE-CHIPのテキストベースのHDL記述は制御系で293ステップ，データパス系及びその他で84ステップで

表1 記述ステップ数の比較

	テキストベース	ブロック図によるHDL設計環境
制御系	293 step	293 step
データパス系	70 step	0 step
その他	14 step	0 step
総記述数	377 step	293 step

あり，総記述数377ステップ中，制御系が約78%，データパス系・その他が22%占めている．本環境では，データパス系・その他の回路をすべて再利用/パラメータ入力で設計でき，テキストベースで作成したときに比べて，総記述ステップ数にして約22%削減できたことを示している．

KUE-CHIPは，比較的データパス系の記述が少ない回路であったが，データパス系の記述比率が高い回路においては，さらに大きな効果が期待できる．また，自動生成されたHDLマクロについては，個々の機能が保証されているため，設計・検証・解析作業が容易になると考えられる．

本環境により，データパス系の設計効率の向上が実現できた．

5 おわりに

本報告では，ブロック図によるHDLベース設計環境の実現方式及び評価について述べた．

本環境では，定型的な機能に関しては，パラメータの設定のみで，論理合成可能なHDL記述を作成することが可能である．また，HDLベースの設計がブロックを組み合わせて行なえるため，視覚的にしかも容易に設計できる．さらに，合成予測を適用することにより，論理合成前に論理合成の可否，回路規模の概略見積りが可能となり，論理合成やレイアウトからの後戻りを少なくすることができる．

これらトップダウン設計手法と設計資産であるHDLマクロを有効に利用することで，ASICの大規模化・高性能化に対応できると考えられる．

今回，評価対象としたKUE-CHIPのように比較的制御系の HDL 記述が多い回路については，制御系もブロック図で取り扱えるように考慮する必要がある．

今後は，ブロック図をデータパス系及び制御系の両方が扱える汎用的なグラフィクスベースの入力形態に拡張していくことが必要である．また，ブロックとして扱えるものをHDLマクロのみではなく，汎用LSI，ソフトウェアなども扱えるように考慮する必要がある．

参考文献

1) 野地保，清水圭典，小山雅行，国岡美千子:トップダウン設計手法と論理合成，情報処理学会 第44回全国大会講演論文集 (6), 6-149 (1992).

2) 神原弘之，安浦寛人:計算機教育用コンピュータの開発とその応用，情報処理，Vol33，No. 2，pp.118-127 (1991).

3) 神原弘之，安浦寛人，Pankaj Kukkal，Hideaki Kobayashi，野地保，小栗清:ハードウェア記述言語の比較，情報処理，Vol.33，No. 11，pp.1269-1283 (1992)

4) David Harel,Hagi Lachover,Amnon Naamad,-Amir Pnueli, Michal Politi,Rivi Sherman,Aharon Shtull-Trauring,and Mark Trakhtenbrot: STATEMATE:A working environment for the development of complex reactive systems,IEEE Transactions of Software Engineering,16(4):403-413 (1990)

5) 松本道弘，高井裕司，岩崎知恵，村岡道明:グラフィック入力による動作機能設計の一手法，情報処理学会研究報告，93-ARC-98, 93-DA-65，pp.73-80(1993)

6) W.Bruce Culbertson,Toshiki Osame,Yoshisuke Otsuru,J.Barry Shackleford,and Motoo Tanaka: The HP Tsutsuji Logic Synthesis System,Hewlett-Packard Journal,August,pp.38-51 (1993)

7) 近藤芳人，山崎孝雄，岩瀬清一郎:拡張Verilog-HDLを用いたデータパス・ライブラリの構築法，情報処理学会 DAシンポジウム'93論文集，pp.69-72 (1993).

8) 野地保，濱田英幸，清水圭典，高坂広之，中村彰:機能記述からの合成予測，電子情報通信学会 第7回 回路とシステム軽井沢ワークショップ論文集,pp.303-308 (1994).