設計自動化 36－4
（1987. 2. 17）

大規模論理回路の論理設計ルール検証方式

室井克信　小迫靖志　橘田光弘　荻原拓冶　村井真一

三菱電機株式会社

大規模論理回路が試験容易化設計ルールを主体とした論理設計ルールに基づいて設計されている場合、この回路がルール通りに作られているかを検証するプログラムを開発した。　プログラムの特徴は検証用のシミュレーションに記号信号値を導入したことにある。これによって、回路が階層化設計されている場合下位階層のモジュールをマクロとして扱えるようになり、大規模回路をトップダウンにもボトムアップにも検証できるようになった。　さらに、論理設計ルールの追加・変更を容易に行えるようにルールベース技術を導入した。この技術と従来プログラムを結び付けることにより、実際の運用にも耐えられることを確認した。

A LOGIC DESIGN RULE CHECK METHOD FOR VERY LARGE CIRCUITS

Katsunobu Muroi , Yasushi Koseko , Mitsuhiro Kitta ,
Takuji Ogihara and Shin-ich Murai

ASIC DESIGN ENGINEERING CENTER , MITSUBISHI ELECTRIC CORP.
5-1-1 OFUNA , KAMAKURA 247 , JAPAN

This paper describes a logic design rule check method for very large circuits which adopts new symbolic simulation technique. If a circuit is designed hierarchically , it is checked both in the top-down and bottom-up manners treating sub-modules on the circuit as black boxes.

Moreover , the rule based technique is introduced in order to add or update the rules easily. We have realized the rule handling in rule based programming and the simulation in conventional programming at the same time.
Consequently the processing time of the program is sufficiently short for production use.

1．まえがき

回路規模の増大に伴って、テストに関わるコストが増化してきている。この問題を解決するために、種々の試験容易化設計が提案されており、特にスキャン設計[3][4]は広く受け入れられている。

スキャン設計法は設計者に設計制約を強いて、回路のテスタビリティーを増す手法である。これらの設計制約は回路を同期化したり、テスト時のタイミング問題を避けるためのものであり、ルールとして明文化されるのが一般的である。（論理）設計ルールチェックプログラムは設計された回路がルールに適合しているかどうかを検証する目的で使われる。

大型計算機のような大規模論理装置においては、実装階層が存在するのが普通である（例えば、ＬＳＩ→マルチチップモジュール →プリント基板→装置 ）。そして、テストはこの各実装階層毎 にボトムアップに行われていく。これに合わせて、テストパターンの生成もボトムアップにおこなわれる。試験容易化設計ルールチェックプログラム[1]はテストパターン生成の前処理として位置づけられるため、やはりボトムアップに使われてきた。

しかし、我々の経験では、下位実装のモジュールでは設計ルール違反がなくても、それらを搭載した上位実装では下位実装モジュール間のインターフェース信号が原因となって設計ルール違反が発生することがままあった。この種の設計ルール違反に伴う再設計のコストは設計の進んだ段階では大きくなる。

本稿では、設計ルール違反の早期発見を実現するために、トップダウンなルールチェックを行う手法について報告する。

当初設計ルールは試験容易化のためのものが主であった。しかし、構造化設計の導入などで設計者間で取り決めた設計指針がルールとして追加されたり、各プロジェクトごとに設計ルールが設定されることが多くなってきている。このため設計ルールをより柔軟に扱う必要が出てきた。本稿では、この問題を解決するために用いたルールベース技術についても報告する。

2．ルールチェック方式の概要

2.1 チェック方式の分類

本節ではルールチェックの基本方針を説明する。設計ルールを分類してみると純粋に回路の構造（トポロジー）に対する制約と論理機能に関する制約に分離することができる。前者の例としては記憶素子以外でのフィードバックループ信号の禁止ルールや特定マクロ後段でのワイアード論理の禁止に関するルール等が考えられる。このようなトポロジカルなルールに対する回路の検証はたんに回路ネット上の探索（回路ネットの観察）ですむ。本稿ではトポロジカルルールに関する検証方式の詳細については触れないことにする。

後者のルールは論理機能に関するもので、例えばレジスタを通常動作モードとスキャン（シフト）モードに自由に切り替えることを保証するものとかレジスタ間の転送に関するものなどがある。これらのルールの検証はシミュレーションの結果を用いてなされる。ルールチェック用のシミュレーション方法としては文献[1]の様に信号値として１、０を使うものと、文献 [2]の様により情報量の多い記号信号値を使うものが考えられる。

我々は、後で述べるように、下位実装モジュールをマクロとして扱う必要性から、記号信号値を採用した。次節ではこの記号信号値を用いたシミュレーションとルールチェック方式について説明する。

2.2 記号シミュレーションとルールチェク

説明のために、対象とする回路はフラットな回路…プリミティブな素子のみからなる回路…であるとする。プリミティブ素子とは、ＡＮＤなどのプリミティブゲート、ワイアード論理、スキャンレジスタ、Ｄラッチ、ＲＡＭ とする。

2.2.1 記号信号

採用した記号信号は、２種類に分類できる。基本記号信号とこれを修飾する付属記号信号である。

(A)基本記号信号

D	:	データ
Cx	:	x相のシステムクロック
G	:	GNDレベルの信号
V	:	Vccレベルの信号
ＳＩ	:	スキャンデータ
ＳＣ	:	スキャンクロック
ＭＳ	:	テスト／ノーマル　モード選択信号

(B)付属記号

データＤやクロックＣxは　より情報量を増すために付属記号を　'．'を用いて連接される。

(B.1)データ

- Ｄ.ＤCx
  ＤCxは Driven by Cx とよむ。
  Ｄ.ＤCxはシステムクロック Cxで駆動される記憶素子の出力データという意味になる。

- Ｄ.ＤCx.ＲＡＭ
  書き込み信号 Cxで駆動されるＲＡＭの出力データを示す。

- Ｄ.ＤCx.ＴＨＲＵ
  システムクロック Cxで駆動されるＤラッチの出力データを示す。

(B.2)クロック

- Cx.GATE
  クロックが制御信号によってストップされる可能性があることを示す。

- Cx.RAM
  この信号がＲＡＭの書き込み信号であることを示す。

基本記号信号と付属信号、あるいはそれらの組合せを用いて複雑な信号値を表現することができる。

2.2.2 記号シミュレーション

先ず、記号シミュレーションとその後のルールチェックの際に用いる２種類の集合（領域）を定義する。

信号値集合（以降、ＳＶＳと記述する）は記号シミュレーション中の記号信号を保持しておく目的で使われる。

制約信号値集合（以降、ＣＳＳと記述する）はルールチェックのときに用いられる集合で、エラーを引き起こす可能性のある記号信号を有している。詳しくは2.3で定義する。

これらの集合は図１に示すように各素子の入出力ピンに付加される。例えば、図１の素子Ｇの入力ピンＡには $SVS_{G,A}$ 、$CSS_{G,A}$ が定義される。

記号シミュレーションを開始する前に一次入力のＳＶＳに初期値が設定される。これが入力パターンとなる。 記号シミュレーションが始まると、一次入力のＳＶＳから内部回路へ記号信号が伝搬される。記号信号が、ある素子Ｇの入力ピンＡに至ると、その記号信号は $SVS_{G,A}$ にセットされる。

素子の出力信号は入力信号と真理値表から決定される。出力記号信号は出力ピンＯの $SVS_{G,O}$ にセットされ、さらに内部回路へ伝搬してゆく。

2.2.3 ルールチェック

記号シミュレーション終了後にルールチェックが行われる。ルール違反は、素子タイプとその入力記号信号から発見することができる。

そこで、各ルール毎に ある素子にルール違反を引き起こす入力記号信号を定義しておく。このように、ルール違反を誘発する記号信号は素子の各ピンのＣＳＳに格納される。

＜定義１：ＣＳＳ＞

素子Ｅのピンについて

$CSS_{E,P}$ ＝ {ｓ | ｓは記号信号であり 素子Ｅのピンに伝搬してはならないもの}

次に簡単な関数ＭＥＭＢＥＲを定義する。

＜定義２：MEMBER＞

if r=s (s ∈ Ｓ) then MEMBER(r,S)→T ;
else MEMBER(r,S)→NIL ;

但し、rは伝搬記号信号値、ＳはＣＳＳを示す

ＣＳＳとMEMBER関数を用いて、ある素子に設計ルール違反が起きているかどうかを判断できる。つまり、素子ＥのピンＰに信号ｒが伝搬しているとき、

if MEMBER(r, $CSS_{E,P}$) ＝T
then ルール違反あり；
else ルール違反無し；

とする。

例として、”クロックはレジスタ、ラッチのクロックピンに入らねばならない”というルールを考えてみる。図２のレジスタＲＥＧ３のクロックピンＥには予め以下のようにＣＳＳが定義されている。

$CSS_{REG3,E}$ = {D.*, G, V, ＳＩ, MS}

但し、D.* はデータの全てを表す

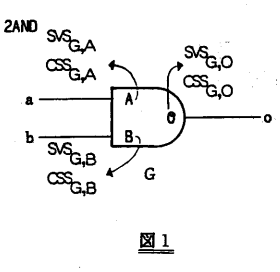

図１

2AND 真理値表

a	b	o
G	*	G
V	V	V
≀	≀	≀
D	D	D
$D.DC_1$	$D.DC_2$	$D.DC_1.DC_2$
D	Cx	Cx.GATE
≀	≀	≀
SI	V	SI
≀	≀	≀

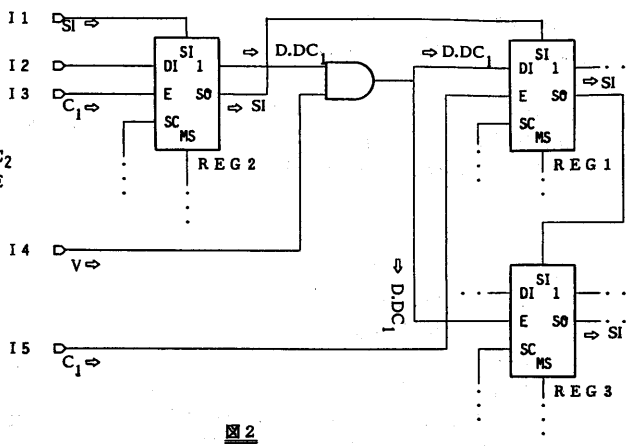

図２

次に記号シミュレーションによって、D.DC1がピンEに到達したとする。すると、

MEMBER（D.DC1, $CSS_{REG3,E}$）→T

であるので、REG3のクロックピンEに関してルール違反が起きていることが判る。

上例で示したように各素子にはCSSが予め設定されている。さらに、シミュレーション途中で、CSSに新たな制約記号信号が付加されることがある。このような制約記号信号は、素子の一方の入力ピンに伝搬信号が到達した場合、その影響で他方の入力ピンのCSSに追加される。

この様子を説明するための例として、別のルールを考えてみる。ルールは、”システムクロックCxで駆動されるラッチの出力信号が、やはりCxで駆動される別のラッチのデータとして入力されてはならない”　とする。図2のレジスタREG1のクロックピンEには予め以下のようにCSSが定義されている。

$CSS_{REG1,E}$= {D.*, G, V, SI, MS}

いま、記号信号D.DC1がREG1のピンDIに到達したとする。すると、例の設計ルールの影響で新しい制約信号C1が$CSS_{REG1,E}$に追加される。

$CSS_{REG1,E}$= $CSS_{REG1,E}$ ∪ {C1}

続いて、信号C1がREG1のEピンに伝搬すると、

$SVS_{REG1,E}$= {C1}

となる。

ルールチェック時に、MEMBER（C1, $CSS_{REG1,E}$）はTを返すので、ルール違反が発見される。

2.2.4　記号信号値のダンプ

ルールチェックが終了し、ルール違反がない場合は各素子のSVSとCSSは以下の4種の状態の何れかに属している。

(1)SVSi,j≠φ　∧　CSSi,j=φ
　伝搬信号がピンjに到達した
　制約信号は定義されていなかった

(2)SVSi,j≠φ　∧　CSSi,j≠φ
　伝搬信号がピンjに到達した
　制約信号は定義されていたが違反はなかった
　CSSi,jはクリアされる

(3)SVSi,j=φ　∧　CSSi,j≠φ
　ピンjには信号は伝搬しなかった
　しかし、制約信号は存在する
　この場合CSSi,jは素子から一次入力に向かって運ばれる。そして到達した一次入力のSVSがやはり空の場合は、運ばれてきた制約信号集合をその一次入力のCSSにマージする。この処理は階層化チェックのために必要になる（3章で理由が述べられる）。

再び図2の例を用いて説明する。レジスタREG2のピンDIに伝搬信号が無かったとする。そうすると、

$CSS_{REG2,DI}$= {D.DC1, G, V, Cx.*, SI, SC, MS}

は入力側へ移動しCSS_{I2}とマージされる。

CSS_{I2}= CSS_{I2} ∪ $CSS_{REG2,DI}$

(4)SVSi,j=φ　∧　CSSi,j=φ
　制約信号も設定されておらず、かつ伝搬信号もない。

設計ルール違反がない場合、一次入出力のSVSとCSSはデータベースへダンプされる。ただし、この段階で各一次入出力にはSVSかCSSのどちらかのみが設定されている。ダンプされた記号信号は上位の階層の設計ルールチェックに使われる。

3．階層的ルールチェック

本章では回路を階層的に設計ルールチェックする手法について説明する。いま、モジュールMは上位実装レベルの回路であるとし、モジュールmはMに搭載されている下位実装レベルのモジュールの一つとする。そしてここでは、モジュールmはすでに設計ルールチェックを終えているとする。前章で述べたとおり、ルールチェック後のmの一次入出力ピンの記号信号はデータベースに格納されている。そこで、モジュールMのルールチェックの間はモジュールmはブラックボックス（マクロ）として扱うことにする。

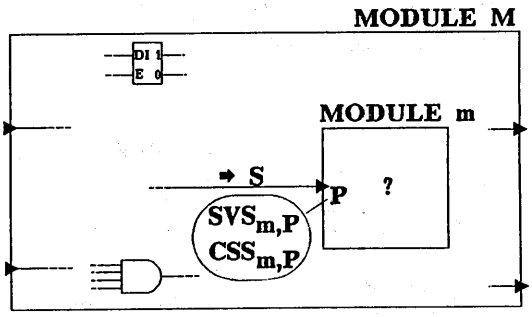

図3

上で述べた状況のもとでモジュールMのルールチェックを考える。

先ず第一に、モジュールMの一次入出力に記号信号を設定する。また、モジュールmの入出力ピンの記号信号もデータベースからSVS，CSSへロードする（実際には、モジュールMについてもデータベースからロードする）。

次に、記号シミュレーションを開始する。２章ではモジュールの一次入力から信号伝搬を始めると述べたが、ブラックボックスの下位実装モジュールがあるときはその出力ピンからも信号の伝搬を行う。このこと以外は下位実装モジュールとプリミティブ素子は同様な扱いをする。上位実装モジュールMの一次出力と下位実装モジュールmの入力ピンのＳＶＳは特別な役割を持つことがある。２章ではＳＶＳはシミュレーション時に伝搬信号値の保持の目的で使われると述べた。しかし、この両者のＳＶＳは、記号信号のロード時点で、回路の期待値を持っている可能性がある。そこでこれらのＳＶＳは信号が伝搬してきても、それによって破壊されないこととする。

モジュールMのプリミティブ素子に対する設計ルールチェックは２章で述べた方法と同じである。ここでは、Mの一次出力とmの入力ピンに対するルールチェック方式について説明する。これらはアルゴリズム上では全く同一の扱いとなる。ここでは、モジュールmの入力ピンｐについて考える。ピンｐには伝搬信号ｓが到達したとする。設計ルール違反はｓ、$SVS_{m,p}$、$CSS_{m,p}$から判断される。考慮すべき状態は全部で４種類ある。

(1)$SVS_{m,p} \neq \phi$ ∧ $CSS_{m,p} = \phi$ ∧ $s \neq null$

```
if  s∈SVSm,p    then 違反無し  ;
                else 違反あり  ;
```

伝搬信号が期待値と不一致のときエラーとなる

(2)$SVS_{m,p} = \phi$ ∧ $CSS_{m,p} \neq \phi$ ∧ $s \neq null$

```
if  MEMBER(s,CSSm,p)=T
    then  違反あり  ;
    else  違反無し  ;
```

期待値の設定がなく、伝搬信号がある場合
制約信号との関係からルール違反が見つかる
ただし、ｓが$SVS_{m,p}$に設定される

この状態は2.2.4の(3)で述べた処理によって引き起こされる。ここでは、ＣＳＳ移動の必要性について説明する。　再び図２を用いる。図２全体の回路をモジュールｍとする（図４）。上位モジュールMのシミュレーションで信号Ｄ.ＤＣ1がモジュールｍの入力ピンＩ2に到達するとしよう。もし、モジュールｍがプリミティブ素子に展開されていると、設計ルール違反は以下の結果から発見できる。

MEMBER $(D.DC1, CSS_{REG2,DI}) \rightarrow T$

なぜなら、2.2.4で示したように、

$CSS_{REG2,DI}$ = {D.DC1, G, V, C. *, SI, SC, MS}

であるから。

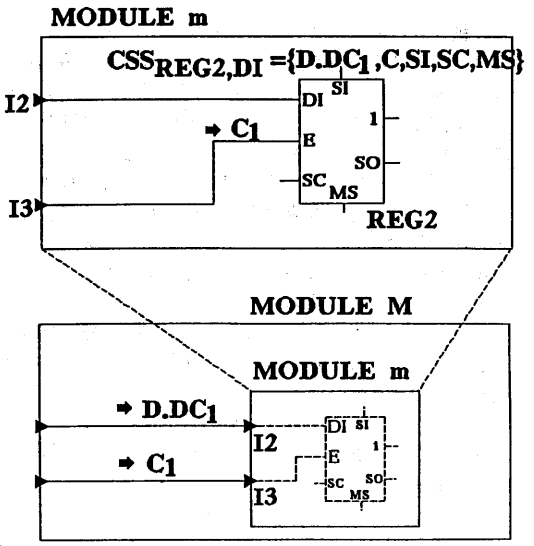

図４

しかしながら、モジュールｍがブラックボックスとして扱われるなら、ｍの入力ピンＩ2ではルール違反を発見することはできない。そこで、モジュールｍのルールチェック時点で、ＲＥＧ２のＤＩピンから入力Ｉ2に向かって$CSS_{REG2,DI}$の移動をしておく必要がある。もし、この処理がなされているならば、

MEMBER（Ｄ．ＤＣ1,CSS_{I2}）→Ｔ　からルール違反を発見することが可能になる。

(3)$SVS_{m,p} = \phi$ ∧ $CSS_{m,p} \neq \phi$ ∧ $s = null$

伝搬信号が無く、制約信号が設定されている状態である。この場合は、ＣＳＳを入力側へ伝搬させる。

(4)$SVS_{m,p} = \phi$ ∧ $CSS_{m,p} = \phi$

どの様な処理も行われない。

モジュールMのルールチェックが終了し、もしルール違反がなかった場合はMの一次入出力のＳＶＳ，ＣＳＳがデータベースにダンプされる。また、ｍについてもＳＶＳ，ＣＳＳに更新が有れば再びデータベースに格納される。

ここで説明した方法は下位実装モジュールｍが既にルールチェック済みで、記号信号がデータベースにダンプされていることが仮定されていた。つまりこれはボトムアップなルールチェック方式であった。しかし我々の方法はたとえ下位実装モジュールの内部回路が知られていなくとも、その一次入出力ピンの記号信号さえ判っていれば上位実装モジュールの設計ルールチェックを行える方法である。そのためには、設計者が　未設計であった

り詳細設計が終了していない下位実装モジュールの入出力ピンに記号信号を割り当ててデータベースに格納しておく必要がある。しかしながら、回路の一次入出力に漏れなく記号信号を設定しておくことは、特に詳細設計未終了の回路では、設計者に多大の負荷を強いることになる。そこで我々は記号信号を図面から自動的に抜き出してくることにした。階層化設計がなされている場合は、たとえ下位実装モジュールがブラックボックスでも、そのインターフェース信号は確定している。このインターフェース信号の命名規則を設けて、後ほど信号名から記号信号を抽出できるようにした。この様な機能を盛り込むことにより、設計ルールチェックをトップダウンにも実施できるようになった。

なお、上位実装に搭載されている下位モジュール全てがブラックボックスである必要はなく、一部が展開されていてもよい。

4．構成

図5はルールチェックプログラムの実現環境を示している。

先ず、EWSから入力された図面から回路ネット情報がデータベースに格納される。

次に、このデータベースから記号信号を抽出するプリプロセッサが働く。生成された記号信号は必要ならエディタにより追加変更を受ける。

設計ルールチェックプログラムはデータベース上の記号信号を用いて実行され、変更された記号信号は再びデータベースに返される。

さらに、テストパターン生成プログラム[5]の入力となるデータも記号信号から自動的に作成される。

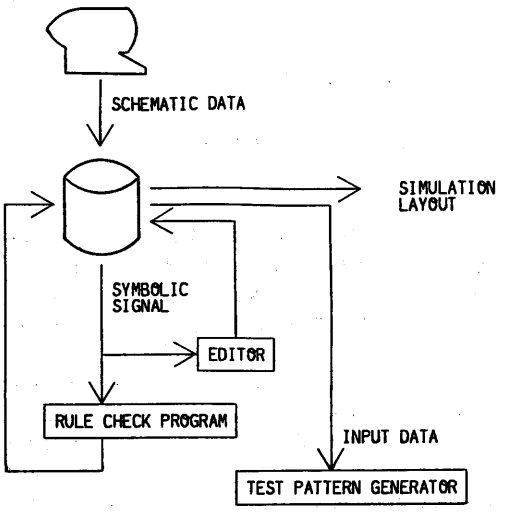

図5

設計ルールチェックプログラムに必要となる計算機時間は問題とはならないが、参考のため次にサンプル回路での実行結果を示しておく。

	実装レベル	LSI数	素子数	時間
①	LSI	−	1577	369sec
②	プリント基板	10	17400	52sec

6.5MIPSマシンを使用
②の素子数は展開した場合

表1

5．ルールベース化

上述した論理設計ルールチェックプログラムはFORTRANで記述されておりプログラム中に設計ルールが埋め込まれているため、プロセス技術等の進歩に伴う設計ルール更新に対し柔軟に対応できず更新のたびにプログラムの改修が生じる。たとえ僅かな設計ルールの更新であってもプログラム全体に渡る改修や最悪の場合作り直しが生じるため、この状況はプログラム管理の面で好ましくないだけでなく、論理装置の開発期間の短縮化を阻害する原因ともなる。

この問題を解決するためのひとつの方法は、プログラム中に埋め込まれていた設計ルールをルールベースとして独立させることである。これによりルールベースのみを変更すればよく、処理部を改修することなしにプログラムの更新が可能となる[6][7]。

まずPROLOGを用いてチェックプログラムを試作した。その結果、設計ルール更新に柔軟なプログラム実現の可能性を確認したが、その反面現在のハードウェアでは実行効率が非常に悪く実用化が難しいという結論を得た。

そこで、柔軟性を保ちつつ高速化を計るため、FORTRANとリンク可能なエキスパートシステム構築用言語を採用し、再びチェックプログラムの作成を試みた。FORTRANには比較的処理量の多いシミュレーションを担当させ、エキスパートシステムにはシミュレーション結果に基づいた設計ルールのチェック及びプログラム全体の制御を担当させる。検証する設計ルールはエキスパートシステム構築用言語により提供されているルール表現を利用して記述する。

我々は、この様な方針に基ずいてチェックプログラムの開発を行い、2種類の回路について4つの基本的な設計ルールの検証を行った。その実行結果を表2に示す。これは実用に耐えられるチェックプログラム実現の可能性を示している。また検証した設計ルール及び実行例を図6に示す。この図からルール表現した設計ルールの記述性・判読性が比較的良いことがわかる。これは陽にプログラム内の処理・制御を考慮せずに設計ルールを表現できることが大きな要因であると考えられる。

表2　実行結果

	素子数	レジスタ数	実行時間（秒）
回路1	161	9	0．2
回路2	1892	70	3．3

6．あとがき

本報告では、大規模論理回路の設計ルールチェック方式について述べた。第一の結論は、記号信号値を用いたシミュレーション手法の採用により、階層化設計回路をボトムアップにもトップダウンにもチェックできるようになったことである。これにより設計者はルール違反を早期に発見できるようなった。

第二の結論は、ルールベース技術の導入によりルールの管理が行い易くなる見通しを得たことである。

当初、論理設計ルールチェックプログラムは回路が試験容易化設計されているかどうかの検証を行っていた。しかし、設計ルールとして種々の論理設計制約や設計指針が含まれてくるにつれ、ルールチェックプログラムは一般的な検証ツールとなってきている。ルールチェックプログラムは設計回路のより構造的な側面を検証するものとして、機能検証用のツールである論理（機能）シミュレータと並存していくであろう。

今後はルールチェックプログラムの検証ツールとしての充実を目指し、設計ルール違反原因回路の指摘や自動変更機能についてもサポートしてゆくつもりである。

7．参考文献

[1]Godoy,H.C.,G.B.Franklin and P.S.Bottorff, "Auromated Checking of Logic Design Structure for Compliance with Testability Ground Rules," proc. 14th DA Conference pp469-478

[2]Bhavsar,D.K., "DESIGN FOR TEST CALCULUS : AN ALGORITHM FOR DFT RULES CHECKING," proc. 20th DA Conference pp300-307

[3]Eichelberger,E.B. and T.W.Williams , "A Logic Design Structure for LSI Testing," proc. 14th DA Conference pp462-468

[4]Funatsu,S. ,et al., "Design Digital Circuits with Easily Testable Considerations," 1978 Test Conference

[5]Ogihara,T. ,et al., "TEST GENERATION FOR SCAN DESIGN CIRCUITS WITH TRI-STATE MODULES AND BIDIRECTIONAL TERMINALS ," proc. 20th DA Conference pp71-78

[6]Horstmann,P.W. and E.P.Stabler , "Computer Aided Design Using Logic Programming ," 1984 DA Conference pp144-151

[7]Kyushik Son, "RULE BASED TESTABILITY CHECKER AND TEST GENERATION," 1985 Test Conference

[8]Muroi,K. ,et al., "A HIERARCHICAL LOGIC DESIGN CHECK PROGRAM FOR SCAN DESIGN CIRCUITS," 1986 ICCAD'86


```
rule non_clock_supplied_register
{
   &1(net  type=REGISTER; io=IN; pin=E;                                    );
   &2(atr  signal=&1.signal; ( get(clock,@.value)=&NON_CLOCK );            );
  -->
   call print_msg(|non_clock_supplied_register|);
   call print_elm(&1);
   call print_val(&2);
};

rule same_phase_transfer
{
   &1(net  type=REGISTER; io=IN; pin=DI;                                   );
   &2(atr  signal=&1.signal;                                               );
   &3(net  element=&1.element; type=REGISTER; io=IN; pin=E;                );
   &4(atr  signal=&3.signal; ( get(clock,@.value)<>&NON_CLOCK );
                             ( eqr(phase,@.value,&2.value)    );           );
  -->
   call print_msg(|same_phase_transfer|);
   call print_elm(&1);
   call print_val(&2);
   call print_val(&4);
};

rule clock_stop
{
   &1(net  type=REGISTER; pin=E;                                           );
   &2(atr  signal=&1.signal; ( get(gated,@.value)=&YES );                  );
  -->
   call print_msg(|clock_stop|);
   call print_elm(&1);
   call print_val(&2);
};
```

図6 設計ルール及び実行例 （続く）

```
rule register_with_wired_or
{
   &1(net  type=REGISTER; io=OUT;                                    );
   &2(net  signal=&1.signal; io=IN;                                  );
   &3(net  element=&2.element; type=WOR;                             );
  -->
   call print_msg(|register_with_wired_or|);
   call print_elm(&1);
   call print_elm(&2);
};
```

```
***** error *****     non_clock_supplied_register
    element=B_REG_4
       type=REGISTER
        pin=E
      value={LOW}

***** error *****     clock_stop
    element=A_REG_1
       type=REGISTER
        pin=E
      value={C03U(GATE)}

***** error *****     clock_stop
    element=B_REG_3
       type=REGISTER
        pin=E
      value={C01U(GATE)}

***** error *****     clock_stop
    element=C_REG_1
       type=REGISTER
        pin=E
      value={C04U(GATE)}
```

図6 設計ルール及び実行例 （続き）