EC83-29

# 時相論理を用いた論理設計検証システムの実際的使用について

# PRACTICAL USE OF LOGIC DESIGN VERIFICATION SYSTEM

藤田 昌宏　中田恒夫　田中 英彦　元岡 達
Masahiro FUJITA Tsuneo NAKATA Hidehiko TANAKA Tohru MOTO-OKA
（東京大学　工学部）
FACULY OF ENGINEERING UNIVERSITY OF TOKYO

## 1. はじめに

近年の素子技術等の進歩により、ハードウェアシステムは益々大規模・複雑化している。このため、全ての場合についてシミュレーションを行なうことで論理設計を検証することは、もはや不可能となっている。従って、シミュレーションを行なうのは、検証に必要な最低限の場合にのみ限ることができるような手法が必要となる。また、正確に検証を行なうためには、仕様についても従来の自然言語やタイムチャートによるものではなく、数学的基礎がしっかりしたもので記述する必要がある。

そこで、我々は既に、時相論理（temporal logic）を用いて仕様を記述し、ゲート回路、状態遷移図等で設計されたものを処理系にProlog を用いて検証することを提案し、その具体的手法を示した［１、２］。

本論文では、この手法を３００ゲート程度の実際的な回路（ミニコンピュータの共通バスインターフェイス回路）に適用し、そこから得られた問題点やそれを解決するための方法について述べ、本手法が実用的に使用できることを示す。

まず、２．で適用する検証対象について簡単に説明し、３．で仕様記述について述べる。４．で具体的検証例を示し、検証において考慮すべき点や効率について述べる。５．では、時相論理を用いてハードウェアの仕様記述を行なう上での問題点及びその解決法について述べる。

## 2. 検証対象

検証対象は、ミニコンピュータ・PANAFACOM・U－３００のDMAに対する共通バスインターフェイス回路［３］であり、全体の構成を図１に示す。検証は、図１を各ブロックに分け、マニュアル［３］に記載されている具体的回路例に対してそれぞれ行なう。これは、U－３００にユーザ側のデバイスを組込もうとする場合に用いるものである。ユーザのデバイスはデバイスコントロール回路を通して接続され、DMAでデータ転送が行なわれる。本回路は、CPUとユーザデバイス間のバス支配権を主に制御する。

図１中、データバスレシーバ回路、アドレスバスドライバ回路、データバスマルチプレクサドライバ回路は単なる組合せ回路であり、DMA制御回路、割込み制御回路、アドレスセレクト回路が順序回路で回路全体の実行順序を制御する。なお、図１中の各ブロック名のあとのカッコ内はおよそのゲート数（フリップフロップも１ゲートと数える）である。

## 3. 仕様記述

### 3．1 同期部と演算部

一般にシステムは、ALUのように実際に論理・算術演算を行なったり、データ転送を行なう演算部と、各演算間の実行順序を制御し、データ転送の同期をとる同期部に分けて考えることができる。

演算部の簡単な例として、図１中のデータバスマルチプレクサドライバ回路を図２に示す。この回路は６種類の入力信号から各種の制御信号を用いて１つを選択し、出力するものである。これらの制御信号は同期部から適当なタイミングで送られてくる。

また、同期部の例として、図１中のDMA制御回路を図３に示す。これは、図４に示すタイムチャートに従って制御信号の値を決める役割をはたす。図４中の矢印は、信号の因果関係を示しており、タイムチャートの最初の部分は次のことを表現している。まず、ユーザデバイスからのリクエスト信号RQDMAが active になると、応答として$\overline{\text{RQDT}}$を active にする。するとCPUが

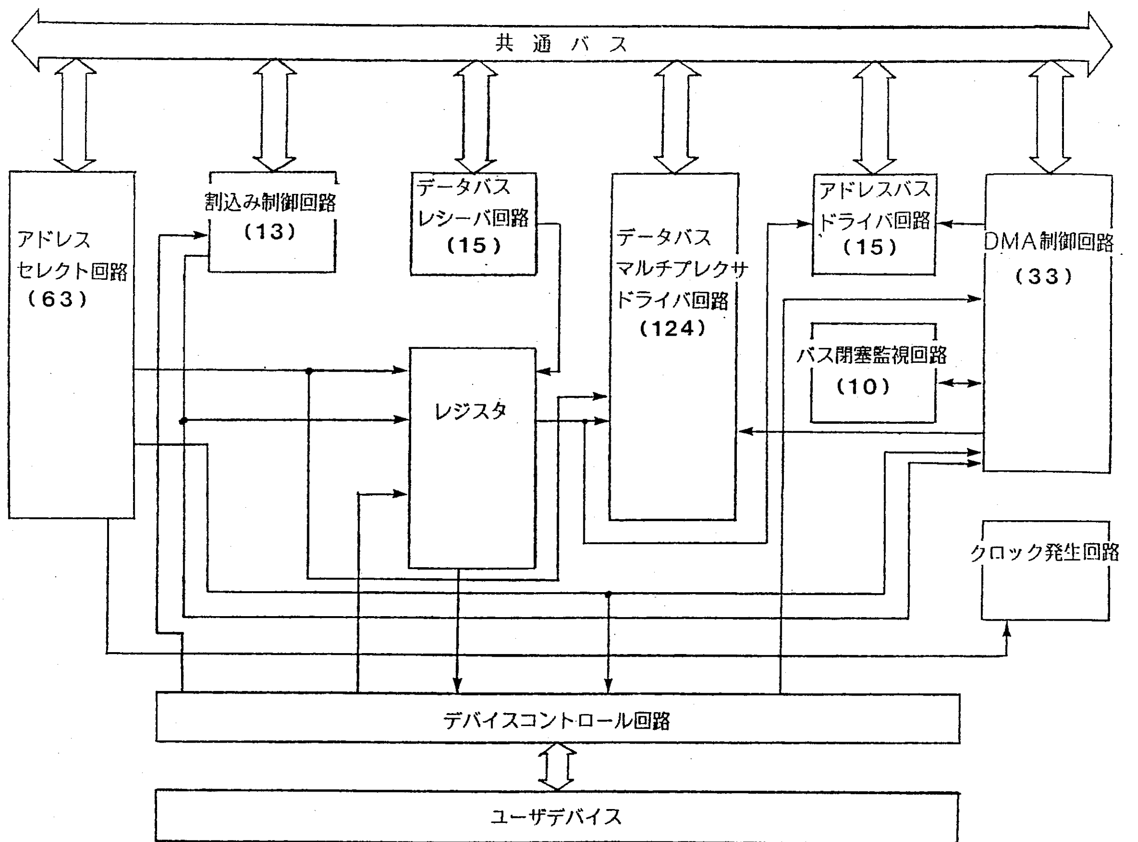


図1　共通バスインターフェイス回路（検証対象）のブロック図

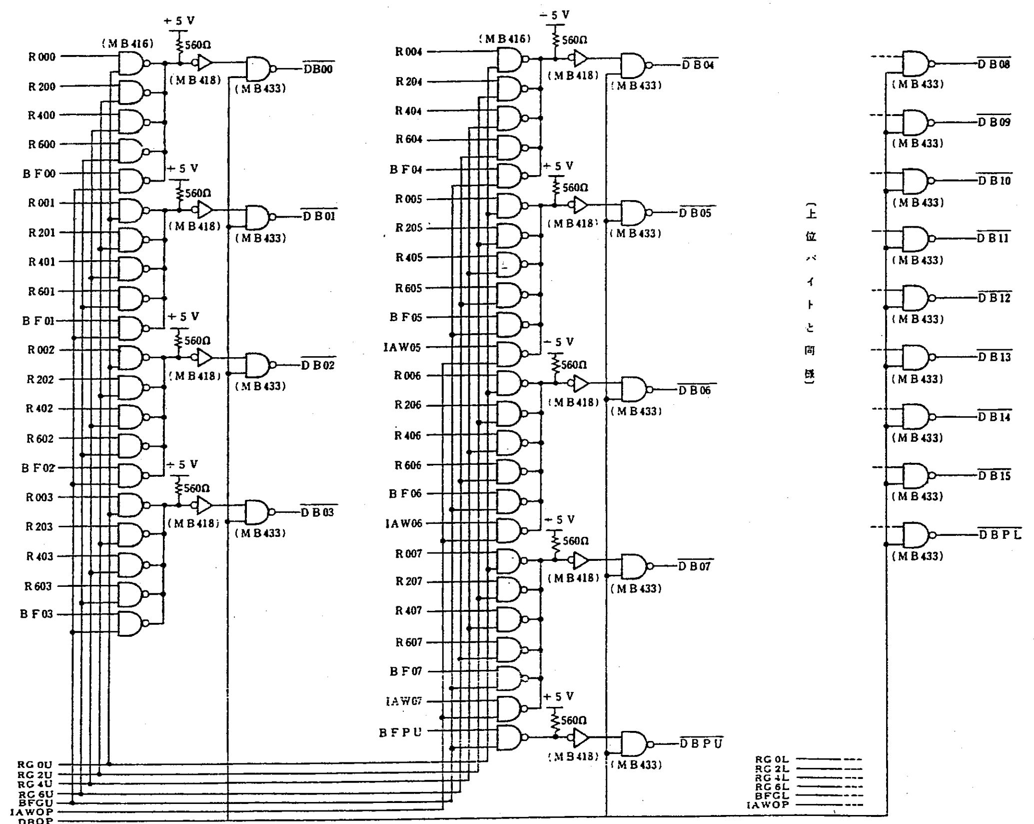


図2　同期部の例：　データバスマルチプレクサ回路
演算

$\overline{\mathrm{ACDT}}$信号を active にし、それがDMA制御回路に伝わって$\overline{\mathrm{RQDT}}$信号をinactiveにする。……

このように、図4は外部回路の制御信号も含めた各信号の順序関係を表わしている。

演算部では、主に与えられた入力データを用い、指定された時間で指定された計算（図2では入力信号線の選択）を終了し得るか否かが問題となり、細かいタイミングはあまり問題とならず、人にとっても比較的考えやすい。

一方、同期部では、各種制御信号を指定通り送出しているか否かが問題となり、並列に動作するもの全体を考慮しなければならず、人にとって誤りも起こしやすい。従って特に自動検証が望まれる。

ここでは時相論理を用いて仕様記述を行なうが、同期部は命題論理の範囲で記述し、決定手続き［4］を用いて自動検証を行なう。また、演算部は述語論理の範囲で記述し、人が対話的に検証していくことにする。

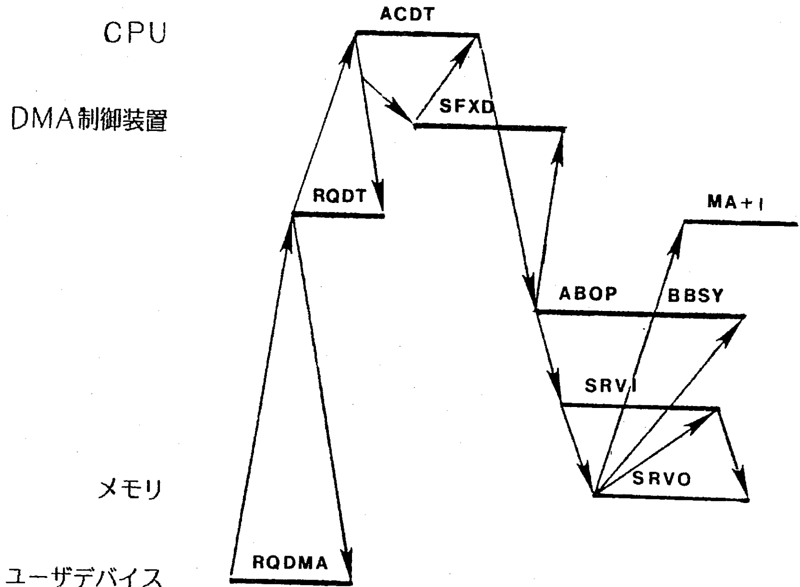


図4　DMA制御回路に対するタイムチャート

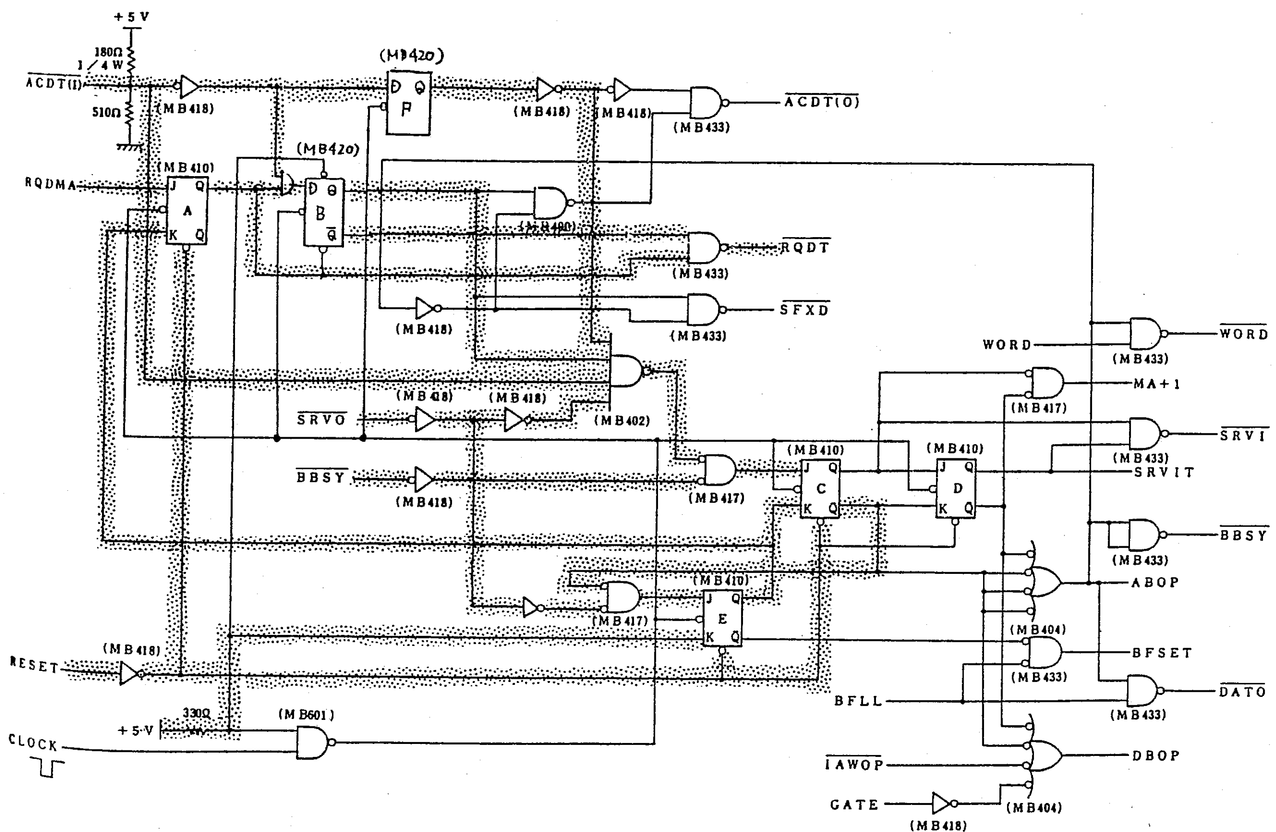


図3　演算部の例：　DMA制御回路

同期

## ３．２　同期部の仕様記述

仕様記述には、時相論理（temporal logic）を用いる。時相論理にもいくつかの種類があるが、ここでは、基本的に linear time temporal logic ［４、５、６］と呼ばれるものをハードウェアを記述しやすいように少し拡張して用いることを考える。拡張については５．で述べ、本節では元のままの形で記述する。 linear time temporal logic は古典論理に、□（always）、▽（sometime）、U（until ）、○（next）の４つの時相演算子を付け加えたもので、時間軸上のシーケンス（状態遷移列）を記述することができる。また、命題論理の範囲では決定手続き［４］が存在し、自動検証が可能である。

図４のタイムチャート中のＲＱＤＭＡ信号がくると、$\overline{\mathrm{RQDT}}$信号が active になるというのは時相論理では、

$$\Box(\mathrm{RQDMA} \to \nabla\overline{\mathrm{RQDT}}) \quad \cdots①$$

という形で表せる。ここで、$\overline{\mathrm{RQDT}}$の上のバーは負論理であることを示し、$\overline{\mathrm{RQDT}}$は active （つまり“０”）、$\sim\overline{\mathrm{RQDT}}$は inactive （つまり“１”）を表している。

条件①は、確かにＲＱＤＭＡ信号がくると、いつか$\overline{\mathrm{RQDT}}$信号が active になることを保証するが、それ以外の時にも active になってしまうかもしれない。従って、$\overline{\mathrm{RQDT}}$信号が active になるのは、ＲＱＤＭＡ信号が active になった時に限るという記述が必要になる。これは、U演算子を用いて次のように表現する。

$$\Box(\sim\overline{\mathrm{RQDT}} \to (\sim\overline{\mathrm{RQDT}}\ U\ \mathrm{RQDMA})) \quad \cdots②$$

これは、一旦$\sim\overline{\mathrm{RQDT}}$になるとＲＱＤＭＡ信号がくるまで inactive でいることを保証する。このように、①、②の記述をペアにして一般に、各制御信号の立上りや立下りの順序を記述していく。

図４のタイムチャート主要部を時相論理で表現したものを図５に示す。図５では簡単のため、最初にＲＥＳＥＴ信号が active になってリセットがかけられた状態を仮定し、そこからのシーケンスについて記述してある。図４の$\overline{\mathrm{ACDT}}$、$\overline{\mathrm{BBSY}}$、ＧＡＴＥはＣＰＵから、ＲＱＤＭＡ、ＢＦＬＬはユーザデバイスから、$\overline{\mathrm{SRVO}}$はメモリから、$\overline{\mathrm{IAWOP}}$はデータバスマルチプレクサからの信号であり、ＤＭＡ制御回路にとっては入力信号となる。これらの信号の制御は、ＤＭＡ制御回路からみれば、外部回路が行なうもので、ＤＭＡ制御回路の仕様には一見関係しないようであるが、実際の設計は外部回路に大きく依存して行なわれる。従って、外部仕様（環境の記述）として、入力信号の制御についても記述しておく必要がある。これらは図４中の<environmant >の部分にあたり、検証を行なう際にも利用される。図５では②の形の記述をいつも入れなければならないので、少し面倒になっている。また、図中の▽演算子は実際には、『いつか』ではなく、『何クロック後までに』という形で記述すべきものもある。これらの解決法については、５．で説明する。

<spec>

$$\Box(\mathrm{RQDMA} \to \nabla\overline{\mathrm{RQDT}}), \Box(\sim\overline{\mathrm{RQDT}} \to (\sim\overline{\mathrm{RQDT}}\ U\ \mathrm{RQDMA})),$$
$$\Box(\overline{\mathrm{ACDT}} \to (\nabla\sim\overline{\mathrm{RQDT}} \wedge \nabla\overline{\mathrm{SFXD}})), \Box(\overline{\mathrm{RQDT}} \to (\overline{\mathrm{RQDT}}\ U\ \overline{\mathrm{ACDT}})),$$
$$\Box(\sim\overline{\mathrm{SFXD}} \to (\sim\overline{\mathrm{SFXD}}\ U\ \overline{\mathrm{ACDT}})),$$
$$\Box(\sim\overline{\mathrm{ACDT}} \to \nabla(\mathrm{ABOP} \wedge \overline{\mathrm{BBSY}}),$$
$$\Box(\sim(\mathrm{ABOP} \wedge \overline{\mathrm{BBSY}}) \to (\sim(\mathrm{ABOP} \wedge \overline{\mathrm{BBSY}})\ U \sim\overline{\mathrm{ACDT}})),$$
$$\Box(\mathrm{ABOP} \to \circ(\mathrm{SRVI} \wedge \sim\overline{\mathrm{SFXD}})),$$
$$\Box(\sim(\mathrm{SRVI} \wedge \sim\overline{\mathrm{SFXD}}) \to (\sim(\mathrm{SRVI} \wedge \sim\overline{\mathrm{SFXD}})\ U\ \mathrm{ABOP})),$$
$$\Box(\overline{\mathrm{SRVO}} \to (\nabla \mathrm{MA+1} \wedge \nabla(\sim\mathrm{ABOP} \wedge \sim\overline{\mathrm{BBSY}}) \wedge \nabla\sim\mathrm{SRVI})),$$
$$\Box(\sim\mathrm{MA+1} \to (\sim\mathrm{MA+1}\ U\ \overline{\mathrm{SRVO}})),$$
$$\Box(\sim(\sim\mathrm{ABOP} \wedge \sim\overline{\mathrm{BBSY}}) \to (\sim(\sim\mathrm{ABOP} \wedge \sim\overline{\mathrm{BBSY}})\ U\ \overline{\mathrm{SRVO}})),$$
$$\Box(\mathrm{SRVI} \to (\mathrm{SRVI}\ U\ \overline{\mathrm{SRVO}})),$$
$$\Box(\mathrm{MA+1} \to \circ\sim\mathrm{MA+1})$$

<environment>

$$\Box(\overline{\mathrm{RQDT}} \to (\nabla\overline{\mathrm{ACDT}} \wedge \nabla\sim\mathrm{RQDMA})),$$
$$\Box(\sim\overline{\mathrm{ACDT}} \to (\sim\overline{\mathrm{ACDT}}\ U\ \overline{\mathrm{RQDT}})), \Box(\mathrm{RQDMA} \to (\mathrm{RQDMA}\ U\ \overline{\mathrm{RQDT}})),$$
$$\Box(\overline{\mathrm{SFXD}} \to \nabla\sim\overline{\mathrm{ACDT}}), \Box(\overline{\mathrm{ACDT}} \to (\overline{\mathrm{ACDT}}\ U\ \overline{\mathrm{SFXD}})),$$
$$\Box(\mathrm{SRVI} \to \nabla\overline{\mathrm{SRVO}}), \Box(\sim\overline{\mathrm{SRVO}} \to (\sim\overline{\mathrm{SRVO}}\ U\ \mathrm{SRVI})),$$
$$\Box(\sim\mathrm{SRVI} \to \nabla\sim\overline{\mathrm{SRVO}}), \Box(\overline{\mathrm{SRVO}} \to (\overline{\mathrm{SRVO}}\ U \sim\mathrm{SRVI}))$$

図５　時相論理によるＤＭＡ制御回路の仕様記述

この他に信号Aが active になるのは、信号Bが active になった後である（A after B）は、A after B＝（○～A U B）…③のようにU演算子を用いて表現することができる。③の形を用いることで、演算Xを実行してから演算Yを実行するという sequential な制御も記述できる。

以上のようにして、命題論理の範囲で同期部を記述する。

## ３．３　演算部の仕様記述

演算部は一般的に、扱うデータのデータ幅に比例して、同じパターンの繰り返しの形で、ハードウェア規模が増大する。このため、演算部の仕様を命題論理の範囲で記述して自動検証を行なうことは、計算機のスピード・スペースの点で不可能であるだけでなく、効率から考えても適した方法ではない。従ってここでは、述語論理を用いて入出力間の関係を関数的に記述し、述語論理に対する proof checker［７、８］を用いて、人が対話的に検証を行なうことを考える。もともと演算部では細かいタイミングは問題とならず、ソフトウェアの sequential program に対する検証手法を適用することも割に容易であり、比較的簡単な記号シミュレーションを行なうことで検証できる場合も多い。

演算部の仕様記述の例として、図２の回路を表現したものを図６に示す。図６では、各信号線をそのビット幅に応じた大きさの配列で表現し、出力信号（$\overline{\mathrm{DBO}}$）と入力信号間の関係を表現している。図２の回路が図６を満しているか否かの検証は、文献［９、１０］と同じようにして、proof checker を用いることにより簡単に行なえる。詳細は文献［９、１０］を参照されたい。

## ３．４　全体の仕様記述

３．２、３．３では、同期部、演算部に分けてそれぞれの仕様記述について述べた。図１全体に対する仕様記述も各信号線を同期部が制御するものと、演算部が制御するものに明確に分けて記述する。このように記述されていれば、時相論理で記述されているものに対しても、同期部については自動的に検証することができる。しかし、階層構造化設計を一貫して支援する際、設計のかなり初期の段階では必ずしも明確に同期部と演算部を分けて記述することは容易ではない。従って、設計者が同期部と演算部に明確に分けて記述しなくても、検証システム側で自動的に分けて整理した形に変換し、処理できることが望まれる。現在、各々の演算は入出力間の関係を表わす関数の形でとらえ、各関数間の順序関係を時相論理で記述するような仕様記述言語を考え、それを用いた各設計段階間の検証について検討している。

## ４．　検証手法及び検証例

本節では、３．２で示したDMA制御回路について、いくつかの検証結果を示し、それに関して問題点とその解決法について述べる。まず、最初に検証法について簡単に説明する。詳しくは、参考文献［１、２］を参照されたい。

<spec>

$0 < i < 15$

$$\Box((\mathrm{RG0U} \wedge \sim\mathrm{RG2U} \wedge \sim\mathrm{RG4U} \wedge \sim\mathrm{RG6U} \wedge \sim\mathrm{BFGU} \wedge \sim\mathrm{IAWOP} \wedge \mathrm{DBOP}) \rightarrow \overline{\mathrm{DB0(i)}} = \mathrm{R00(i)})$$

$$\Box((\sim\mathrm{RG0U} \wedge \mathrm{RG2U} \wedge \sim\mathrm{RG4U} \wedge \sim\mathrm{RG6U} \wedge \sim\mathrm{BFGU} \wedge \sim\mathrm{IAWOP} \wedge \mathrm{DBOP}) \rightarrow \overline{\mathrm{DB0(i)}} = \mathrm{R20(i)})$$

$$\Box((\sim\mathrm{RG0U} \wedge \sim\mathrm{RG2U} \wedge \mathrm{RG4U} \wedge \sim\mathrm{RG6U} \wedge \sim\mathrm{BFGU} \wedge \sim\mathrm{IAWOP} \wedge \mathrm{DBOP}) \rightarrow \overline{\mathrm{DB0(i)}} = \mathrm{R40(i)})$$

$$\Box((\sim\mathrm{RG0U} \wedge \sim\mathrm{RG2U} \wedge \sim\mathrm{RG4U} \wedge \mathrm{RG6U} \wedge \sim\mathrm{BFGU} \wedge \sim\mathrm{IAWOP} \wedge \mathrm{DBOP}) \rightarrow \overline{\mathrm{DB0(i)}} = \mathrm{R60(i)})$$

$$\Box((\sim\mathrm{RG0U} \wedge \sim\mathrm{RG2U} \wedge \sim\mathrm{RG4U} \wedge \sim\mathrm{RG6U} \wedge \mathrm{BFGU} \wedge \sim\mathrm{IAWOP} \wedge \mathrm{DBOP}) \rightarrow \overline{\mathrm{DB0(i)}} = \mathrm{BF0(i)})$$

図６　時相論理によるデータバスマルチプレクサ回路の仕様記述

検証は次の４つのステップにより実行される。

（１）与えられた検証すべき仕様の否定をとる。
（２）その否定を時相論理の決定手続きを用いて、現在に対する条件と次の時刻以降に対する条件に分ける形で展開し、状態遷移表現に変換する。
（３）できた状態遷移表現をゲート回路や状態遷移図による設計と同じようにProlog に変換する。
（４）（３）でてきたProlog の記述と検証すべき設計から得られたProlog の記述とをＡＮＤ条件で結び、全体として時間の進む向き（順方向）または、その反対の向き（逆方向）に状態遷移させる。もし、ループになって恒久的に満される状態遷移列があれば、反例として印刷する。

以上のうち、検証すべき仕様に対して変化するのは、（１）～（３）の結果得られるProlog の記述のみであり、（４）を実行するプログラムはいつも同じでよい。

また、ステップ（２）の展開は、次に示すような時相論理の恒等式を用いて行なわれる。

$\nabla F = F \vee \bigcirc \nabla F$

$\Box F = F \wedge \bigcirc \Box F$

$F1 \ U \ F2 = F2 \vee (F1 \wedge \bigcirc (F1 \ U \ F2))$

$\sim \nabla F = \sim F \wedge \bigcirc \sim \nabla F$

$\sim \Box F = \sim F \vee \bigcirc \sim \Box F$

$\sim (F1 \ U \ F2) = \sim F2 \wedge (\sim F1 \vee \bigcirc \sim (F1 \ U \ F2))$

例えば、$A \rightarrow (B \ U \ C)$ を検証するには、まず否定をとり、

$\sim (A \rightarrow (B \ U \ C)) = A \wedge \sim (B \ U \ C)$ …④

となる。従って、上記の関係から、

④$= A \wedge (\sim C \wedge (\sim B \vee \bigcirc \sim (B \ U \ C)))$
$= (A \wedge \sim B \wedge \sim C)$
$\vee (A \wedge \sim C \wedge \bigcirc \sim (B \ U \ C))$ …⑤

と展開される。これから図７のような状態遷移表現が得られる。以上のように任意の時相論理の式は状態遷移表現に展開できる。また、時相論理で記述されたものを検証する際にも、時相論理の記述を上のように展開してProlog に変換することにより、ゲート回路や状態遷移図と同じように検証を行なうことができる。

図３の回路に対し、図６の仕様の１つである①を時間軸に対して順方向（時間の進む向き）に推論する場合の処理時間は、上記の検証プログラムをそのまま使用するとＣＰＵ時間で１８０秒以上かかる（ＴＳＳのＣＰＵ時間制限が１８０秒のため、その制限内では処理が終了しなかった。なお、処理系には大型計算機Ｍ－２８０Ｈ上のProlog ／ＫＲ［１１、１２］を用いた。）。

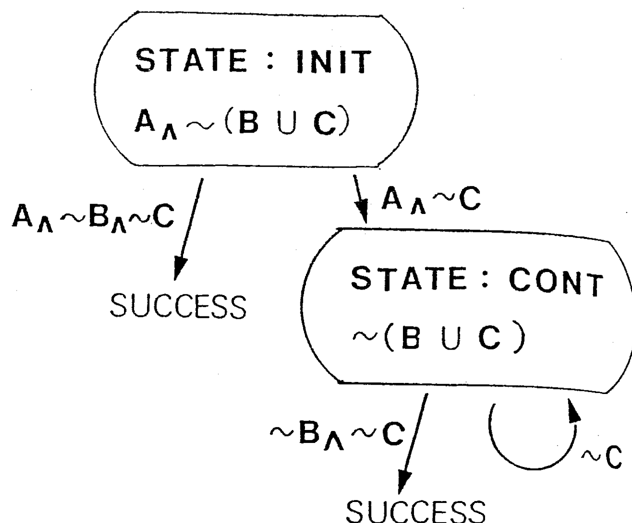


図7　～（A→（B　U　C））を展開して得られる状態遷移表現

そこで、検証を効率化するため、調べる場合の数を減らすことを考える。これには、次のようなことが考えられる。

（１）外部回路の仕様から、ある入力信号線の値が決定できる場合には、その信号線の値を固定する。
（２）検証プログラムにおいて、１度調べた状態を記憶しておき、同じことを２度調べないようにする。
（３）回路の構造から現在行なっている検証に関係する信号線とは明らかに無関係である入力信号線の値を固定する。
（４）（３）を進めて、現在行なっている検証に関係する部分の回路のみを抽出し、その抽出したものに対して検証を行なう。

以下、具体的に説明する。
（１）については、例えば図４のタイムチャートから分かるように、ＲＱＤＭＡ信号がきて、$\overline{RQDT}$信号が active になるまでの間は、ＣＰＵが制御する信号である$\overline{ACDT}$は inactive のままである。これは、図6の仕様中の<environment >の中の、

$\Box (\sim \overline{ACDT} \rightarrow \sim \overline{ACDT} \ U \ \overline{RQDT})$ …⑥

からも分かる。従って、①の検証においては、$\Box \sim \overline{ACDT}$（または、⑥）を条件に付け加えることによって調べる場合の数を減らすことができる。また、後の例で示すように、実際の回路設計は適当な外部回路の仕様を考慮して行なわれており、適当に外部回路の条件

を加えないと反例がでることも多い。このような時相論理の条件を検証に付け加えるには、上記の検証プログラム作成の時と同じように、与えられた時相論理の式を決定手続きを用いて状態遷移表現に展開したのち、Prolog に変換すればよい。なお、図6の外部回路の条件を全てProlog に変換して検証すれば、調べる場合の数が最も減少するはずであるが、条件があまり多いとそちらを参照するのに時間がかかるようになるため、現在行なっている仕様に関係ないものは除いた方がよい。

（２）については、もとの検証プログラムでは、Prolog の処理系のもつ自動バックトラック機能のみを用いて、しらみつぶしに調べているので、同じ状態を何度も調べてしまう。そこで１度調べた状態（仕様や外部回路の条件を表わす時相論理の式の状態と各フリップフロップの内部状態）をProlog のプレディケイトとして記憶しておき、２度同じ状態を処理しないようにすれば、検証プログラムの手間が増えまたメモリの消費量が増大するが、検証時間の短縮化が期待できる。

（３）は図３で考えると、①の検証には、WORD、BFLL、$\overline{\mathrm{IAWOP}}$、GATEの４つの信号線が影響を及ぼさないことは、少し回路を解析することで容易に分かる。従ってこれらの信号を０か１に固定して検証を行なっても問題はなく調べる場合の数をかなり減らすことができる。

（４）は、回路をさらに解析し、①の検証に関係するゲートのみを抽出して、それについて検証を行なうものである。回路の抽出は次のようにして行なう。まず、検証すべき仕様に現れる信号線のうち、外部入力信号線を取り除く。こうすると今の例の場合は$\overline{\mathrm{RQDT}}$のみが残る。次に残った各信号線について、回路を出力から入力へトレースしていき、全てのパスが（ａ）外部入力端子に到達するか、（ｂ）既に通った回路に到達するまで行なう。このようにトレースされたゲートの集合が、その仕様の検証に関係する部分の回路となる。実際に図３の回路に対してこれを行なうと、図中の　　のついた部分が抽出される。検証は　　のついた部分のみに対して行なうことで、調べ場合の数を大きく減らすことができる。

①の検証に対して以上の手法をいろいろ組合せて用いた場合の処理時間をまとめたものが表１である。
表１には、

表１、２は次のようになる。

（ａ）

| | α | ～α |
|---|---|---|
| ～β | 32.1 | >180.0 |
| β | 16.5 | >180.0 |

（ｂ）

| | α | ～α |
|---|---|---|
| ～β | 6.3 | 87.8 |
| β | 3.9 | 24.4 |

（ｃ）

| | α | ～α |
|---|---|---|
| ～β | 2.2 | 30.3 |
| β | 1.3 | 8.1 |

α：記憶あり
～α：記憶なし
β：□（～$\overline{\mathrm{ACDT}}$）あり
～β：　〃　　なし

表１　□（RQDMA→▽$\overline{\mathrm{RQDT}}$）の検証に要するCPU時間
（単位・秒、M－２８０H上のProlog ／KR［１１、１２］による）

（ａ）

| α | ～α |
|---|---|
| 1.7 | >180.0 |

（ｂ）

| α | ～α |
|---|---|
| 0.8 | 16.6 |

（ｃ）

| α | ～α |
|---|---|
| 0.3 | 5.8 |

α：記憶あり
～α：記憶なし

表２　□（$\overline{\mathrm{ACDT}}$→▽～$\overline{\mathrm{RQDT}}$）の検証に要するCPU時間

（ａ）図３の回路をそのまま使う、
（ｂ）４つの検証に関係ない外部入力信号を固定する（上記の（３）に対応する）、
（ｃ）検証に関係のある部分のみの回路を抽出して使用する（上記の（４）に対応する）
の３通りのそれぞれの場合について、
（α）上記の（２）に対応する１度処理した状態を記憶し２度調べることがないようにする否か、
（β）（１）に対応する□～$\overline{\mathrm{ACDT}}$を付け加えるか否か、
の計４通りの処理時間が示してある。なお、表中“＞１８０”となっているのは、検証が１８０秒では終了しなかったことを示す。表から、（１）により２～４倍、（２）により６～１４倍、（３）により４～７倍、（４）により１６～２０倍程度検証が速くなることが分かる。

（１）については、適当に外部回路の条件を与えないと反例がでる場合もある。例えば、図３の回路に対し、
□（$\overline{\mathrm{ACDT}}$→▽～$\overline{\mathrm{RQDT}}$）　…⑦
を検証した結果が表２であるが（（ａ）、（ｂ）、（ｃ）の意味は表１と同じ）、これは、□（～RQDMA）を仮定しないと反例がでる。このことから、外部回路の条件が自由に検証システムに与えられるようになっている必要があることが分かる。
また（２）については、実際に記憶された状態の数は、

だいたい２０～１００程度であった。従って、回路規模があまり大きくならない（言いかえれば、調べる回路の範囲をしぼれる場合）には、必要なメモリ量もそれほど多くなく非常に有効であると言える。

表１、２以外の仕様や他の回路に対しても、ほぼ同じような検証結果が得られており、（１）～（４）の手法はそれぞれかなり調べなければならない場合の数を減らし、検証時間を短くすると言える。

## ５．　時相論理を基礎とした仕様記述言語

３節では、システムを同期部と演算部に分け、時相論理を用いて仕様記述をすることを述べたが、その際に次のような考慮すべき点があることが分かった。

（１）実際のハードウェアの応答等を記述する際には、▽演算子を用いて記述している部分でも、『いつか』ではなく、『何クロック後までに』という意味を表現しなければならないこともある。
（２）演算Ｘを行なってから演算Ｙを行なうという sequential な制御を記述する場合には、演算Ｘ、Ｙそれぞれに対し、実行中であることを示すフラグ$F_x$、$F_y$ を考え、それらに③式に示すような条件を付けることになる。従って演算の数がかなり大きくなると、これらを記述するのが面倒になってくる。また、同じことを繰り返すようなことも表現しにくい。
（３）非同期回路を扱う際には、ゲートの遅延を陽に指定したい場合もある。
（４）現在のところ仕様は、論理式の形で記述されているので、仕様に記述されていないことについては、何が起ってもよいことになっている。従って厳密に仕様を記述するためには、設計者はかなり細かいところまで考慮しなければならず、負担が大きい。

以下にこれらについて検討する。
（１）については、○演算子を用いることで同期回路について、『n クロック後まではＰでありつづける』（$\boxed{n}$Ｐとする）と、『n クロック後までにはＰとなる』（$\nabla_n$Ｐとする）を表現することができる。
今、$\textcircled{n}$Ｐを

$$\textcircled{n}P = \overbrace{\bigcirc\cdots\bigcirc}^{n\text{個}} P \quad \cdots⑧$$

と定義すると、□Ｐ、▽Ｐはそれぞれ、

$$\boxed{n}P = P \wedge \bigcirc P \wedge \textcircled{2}P \wedge \cdots \wedge \textcircled{n}P \quad \cdots⑨$$

$$\nabla_n P = P \vee \bigcirc P \vee \textcircled{2}P \vee \cdots \vee \textcircled{n}P \quad \cdots⑩$$

と表現できる。⑨を用いると、『信号Ｐがn クロック続けて active にならないと信号Ｑは active にならない』は、

$$(\sim Q)\ \ U\ \ (\boxed{n}P) \quad \cdots⑪$$

のようにして表現できる。

（２）については、例えば、図４を正確に記述する
かなりの記述量になってしまう。これには次の２つの
決案が考えられる。
（ａ）各演算が実行中であることを示すフラグを用い
が sequential な制御等のようによく用いるものについては、マクロとしてあらかじめ登録しておくようにする
こうすれば、実際に設計者が記述しなければならない
は減少する。
（ｂ）Moszkowski ［１３、１４］のinterval temporal logic を用いることにする。これは時間をいくつかのintervalというものに分け、その interval ことに□、▽、○等の演算子に対応するもので記述していくものである。また、分けられた interval はさらにいくつかのサブinterval に分けて記述することができ、時間に関して抽象的なものから具体的なものへの詳細化を円滑に支援できる。論理としての表現能力ということでは、文献［１３、１４］に示されているハードウェアの記述は、linear time temporal logicでもほぼ記述できる。ただ、interval temporal logic で記述されたものの方が理解しやすい。しかし、自動検証という観点から考えると、（ｂ）のように記述されているよりも（ａ
のように記述されている方がやりやすい。これは、（ａ
のように記述すると、各演算が実行中であることを示すフラグが一種の内部状態を示すものであるため、検証に際し調べるべき場合の数をかなり減らす役割を果すからである。（逆に、内部状態を表すものまで記述しなければならないから（ａ）は記述しにくいとも言える。）
結局、sequentialなもののみの記述ならば、（ｂ）の方がよいが、複数の並列に動作するもののinteraction を記述する際には、（ａ）のような考え方も必要であり、（ａ）、（ｂ）の両方を用いるのがよいと考える。

（３）については、遅延時間の最小単位を○演算子における次の時刻に対応させれば記述できる。しかし、遅延時間が一定ではなく幅があるような場合には、記述自体は問題なくできるが、検証については、○演算子で展開していたのでは、非常に能率が悪い。従って何らかの工夫が必要であり、現在の検証プログラムで各ゲートの遅延まで考慮した検証を行なうのは、計算機スピードの点でむつかしい。

（４）は、実用的な運用においては、問題となる。解決法としては、各信号線にレジスタ、ターミナルという属性を設け、レジスタ属性の信号は、仕様に記述されている条件でしか変化しないことにするというのが考えられる。例えば、レジスタＱに対し、

$$P \to \circ Q \quad \cdots ⑫$$

は、「もし今Pなら次の時刻にレジスタQをセットする。もし今～Pなら次の時刻Qの値は変化しない。」を意味する。（これは、

$$(\sim Q) \to ((\circ \sim Q) \quad U \quad P) \quad \cdots ⑬$$

が暗黙の内に仮定されていることに対応する。）

このようにすれば、仕様の記述量をかなり減らすことができ、また設計者にとっても分かりやすいものとなる。図5の仕様をこのような条件のもとで書き直したものが図8である。図5に比べ非常に分かりやすくなっている。図8を用いて検証する際には、検証システム内部で自動的に⑬に対応するものを導きだし（これは各レジスタの変化する条件を整理すればよく、自動的にできる。）、図5の形にして検証できる。なお、時相論理の決定手続を用いて、仕様から状態遷移図を自動合成する際には、決定手続を一部変更することにより、図8から直接合成することができる。詳しくは、文献［15、16］を参照されたい。

<spec>

$\square(RQDMA \to \nabla \overline{RQDT})$,
$\square(\overline{ACDT} \to (\nabla \sim\overline{RQDT} \wedge \nabla \overline{SFXD}))$,
$\square(\sim\overline{ACDT} \to \nabla(ABOP \wedge \overline{BBSY})$,
$\square(ABOP \to \circ(SRVI \wedge \sim\overline{SFXD}))$,
$\square(\overline{SRVO} \to (\nabla MA+1 \wedge \nabla(\sim ABOP \wedge \sim\overline{BBSY}) \wedge \nabla \sim SRVI))$,
$\square(MA+1 \to \circ \sim MA+1)$

<environment>

$\square(\overline{RQDT} \to (\nabla \overline{ACDT} \wedge \nabla \sim RQDMA))$,
$\square(\overline{SFXD} \to \nabla \sim\overline{ACDT})$,
$\square(SRVI \to \nabla \overline{SRVO})$,
$\square(\sim SRVI \to \nabla \sim\overline{SRVO})$

図8　各変数をレジスタと考えたDMA制御回路の仕様記述

現在以上のことを考慮し、検証ということに重点をおいた、時相論理を基礎にしたハードウェア仕様記述言語を検討中であり、まとまりしだい適当な機会に報告することとしたい。

## 6. おわりに

時相論理とProlog を用いた検証システムを実際的な回路に適用した結果を報告した。

システムを同期部と演算部に分け、同期部は命題論理で記述し自動検証を、演算部は述語論理で記述し人との対話的な検証を行なうことで、実用性のある検証システムとすることができる。また、3．の（1）～（4）で述べたような工夫を行なうことによって、検証に際し、調べなければならない場合の数を大きく減少できることが分かった。

時相論理を用いて効率よく仕様記述を行なうには、5．で示した様々な工夫も必要となる。今後はこれらを考慮し、かつ階層構造化設計を円滑に支援できる、時相論理を基礎とした仕様記述言語を考え、それを検証するシステムを開発していきたい。

## 7. 参考文献


［1］　藤田、田中、元岡："テンポラルロジックによるハードウェア仕様記述とその検証"、電気学会電子デバイス研究会資料、EDD83－38、1983

［2］　藤田、田中、元岡："テンポラルロジックとProlog ／KRを用いたハードウェア論理設計の検証"、ロジックプログラミングコンファレンス、1983

［3］　"PANAFACOM・Uシリーズ　ユーザ装置設計手引書"、09HS－0080－1、富士通

［4］　P. Wolper ："Temporal Logic Can Be More Expressive ", 22nd Annual Symposium on Foundation of Computer Science, 1981

［5］　A. Pnueli , S. Shelah , J. Stavi : "On the Temporal Analysis of Fairness ", 7th Annual Symposium on Principle of Programming Language , 1980

［6］　Z. Manna : "Verification of Sequential Programs : Temporal Axiomatization ", Dept. of Computer Science, Stanford University , Report No.STAN－CS－81－877, 1981

［7］　R. W. Weyhrauch : "A Users Manual for FOL", Dept. of Computer Science, Stanford University , Report No.STAN－CS－77－432, 1977

［8］　M. Gordon , R. Milner , C. Wadsworth : "Edinburgh LCF", Lecture Note in Computer Science, Vol. 78, Springer-Verlag , 1979

[9] T. J. Wagner : “Hardware Verification ”, Dept. of Computer Science, Stanford Universoty , Report No.STAN−CS −77−632, 1977
[10] 藤田、田中、元岡：“定理証明法によるハードウェア検証の検討”、情報処理学会設計自動化研究会夏期シンポジウム、1983
[11] 中島：“Prolog ／KR User ´s Manual ”、東京大学工学部、テクニカルレポートME TR82−4、1982
[12] 中島：“Prolog ”、コンピューター・サイエンス・ライブラリー、産業図書、1983
[13] B. Moszkowski : “A Temporal Logic for Multi- Level Reasoning about Hardware ”, Dept. of Computer Science, Stanford University , Report No.STAN−CS−82−952, 1982
[14] J. Halpern, Z. Manna, B. Moszkowski : “A Hardware Semantics Based on Temporal Intervals”, Dept. of Computer Science, Stanford University , Report No.STAN−CS−83−963, 1983
[15] 濱中、藤田、田中、元岡：“テンポラルロジックによるードウェア仕様記述と状態遷移図の合成”、ロジックプログラミングコンファレンス、1983
[16] 濱中：“テンポラルロジックによる論理設計法”、東京大学工学部電気工学科卒業論文、1983