設計自動化　54－10
(1990.10. 9)

ディジタル装置用ソフトの設計検証のための一手法

小林 康弘，高元 政典，山田 直之，
狩野 泰信(＊)，中村 知治(＊)

(株)日立製作所エネルギー研究所
＊ (株)日立製作所国分工場

あらまし　ディジタル装置のソフトウェアとシーケンス図との間の整合性の検証を支援する手法を開発した。本手法は，シーケンス図の素子に対応するプログラム・モジュールの演算順序の検証を，次の２つのステップにより支援する。1)演算順序がシーケンス図の信号の入出力関係から定まる制約条件を満たすことを確認する。2)フィードバック・ループをカットする必要があった場合，その影響によりクリティカルなエラー事象が発生するための必要条件を記号表現で求める。

A Verificaton Support Method for Programmable Controller Software

Yasuhiro Kobayashi, Masanori Takamoto, Naoyuki Yamada
Yasunobu Kanou(＊) and Tomoharu Nakamura(＊)

Energy Research Laboratory, Hitachi, Ltd.
＊ Kokubu Works, Hitachi, Ltd.

1168 Moriyama-cho, Hitachi-shi, Ibaraki-ken
＊ 1-1-1 Kokubu-cho, Hitachi-shi, Ibaraki-ken

Abstract　A design verificatin support method for digital control equipment has been developed for the consistency between sequential diagram logic and controller software. This method verifies the processing order of program module corresponding to logic gates in a sequential diagram by two steps. Firstly, it confirms that the order satisfies constraints given by the input-oputout relation of gates in a sequential diagram. Secondly, it identifies the necessary condition that feedback loop cuts cause any critical error events in the timing delay propagation process, since logic sequentialization requiers loop cuts.

1．　はじめに

発電所，電力系統等大規模システムの制御・保護装置のトラブルを防止する上で，シーケンス制御回路の設計の高信頼化をはかり，設計段階の不具合を完全に無くすることが鍵となっている。設計の高信頼化のための方策の一つは，シーケンス制御回路を対象とする設計検証技術を確立することである。

シーケンス制御回路を対象とする設計検証手法の例としては，定理証明法を適用して，シーケンス制御回路の設計結果が設計上目標とした機能(設計仕様)を満足することを検証するアプローチ[1-2]が報告されている。

シーケンス制御回路は最近，急速にディジタル化されつつあり，システムの基幹的部分を除いたその大半が，プログラマブル・コントローラというディジタル装置[3]の形で実現される趨勢にある。このようなシーケンス制御回路では，シーケンス図は中間的な設計結果に過ぎず，ディジタル装置上で稼働するソフトウェア（以下ソフトと記す。）が最終的な設計結果となる。

プログラマブル・コントローラの場合，シーケンス制御回路の設計は，(a) 設計仕様を基にシーケンス図の内容を生成する部分である上流側のフェーズ，(b) シーケンス図の並列処理的な内容を基にディジタル装置の逐次処理的なソフトを生成する部分である下流側のフェーズに分けて考えることができる。上記のアプローチは，上流側のフェーズに対する設計検証を対象としている。

本研究の目的は，ディジタル装置で実装されるシーケンス制御回路であるプログラマブル・コントローラの設計の下流側フェーズのための設計検証手法を開発することにある。

2．　ディジタル装置用ソフト

2.1　ソフトＣＡＤシステム

ソフトＣＡＤシステムの入力は，シーケンス図の内容であり，プログラマブル・コントローラの設計の上流側のフェーズから得られる。出力は，適当な計算機言語で記述されたマイクロ・プロセッサのプログラムである。シーケンス図の各素子の信号処理は，それぞれプログラムの対応する部分による情報処理に置き換えられる。ソフト作成の処理手順の第１は，シーケンス図の素子の処理内容に対応させてプログラムのモジュールを作成することである。

また，プログラムではある部分を別の部分の前に処理するようにしている。すなわち，ソフトウェア作成の処理手順の第２は，並列処理的なシーケンス図の内容を，逐次処理的なマイクロ・プロセッサのソフトに移すために必要となるものである。

第１のステップは，シーケンス図の素子に対応するプログラムのモジュールをライブラリー化しておき，シーケンス図をグラフィック的に入力する際に，各素子とモジュールを対応付ける処理である。このステップは，ソフトＣＡＤシステムで自動的に実行できる。

第２のステップは，シーケンス図の上で素子に番号付けすることにより，シーケンス図の素子に対応するモジュールの演算順序をユーザが入力する処理である。以下では，この演算順序を「素子の演算順序」と称する。シーケンス図で素子は，信号の流れに従って並行動作的に「演算」をするので，大域的な順序が陽に現われることはない。

素子の演算順序は，シーケンス図における素子の位置を基に，「上から下へ，左から右へ」というような経験的ルールを用いて，ユーザがパターン認識的に決めている。このような経験的ルールに従って，シーケンス図では，各素子が順序付けられ，対応するプログラムでは，対応するモジュールがレイアウトされている。

本研究の対象とする設計検証手法は，ソフトＣＡＤシステムで設計されたプログラムの検証を支援する機能を実現するものと位置付けることができる。このようにして設計されたソフトは，次の製作のステップでディジタル装置に実装され，テストのステップで製品としての品質が確保される。

2.2　ソフト設計の検証

(1) 演算順序によるエラーの例

簡単なシーケンス制御回路を対象に，シーケンス図からマイクロ・プロセッサのプログラムに変換する際に，素子の演算順序によるエラーが発生する例を説明する。そのようなケースの例を図１に示す。

図１(a)がシーケンス図である。３入力，１出力であり，ＡからＧまでの７素子を含む。Ｂ，Ｆのメモリに対しては，リセット信号のフィードバック・ループが存在する。素子の演算順序は，ＡからＧを一列に並べる時の順列であり，

シーケンス図での信号の流れに従わない場合，マイクロ・プロセッサでの処理でスキャン・サイクルが同一のデータを使えず，処理結果に一種の遅れを生ずる。例えば，素子Ｄ，Ｅに注目する。Ｄの後でＥを計算すれば，サイクルｎでのＥの出力は，サイクルｎでのＤの出力から求められる。しかし，Ｄの前でＥを計算すると，サイクルｎでのＥの出力は，サイクルｎ－１でのＤの出力を使って求めることになる。

素子の演算順序をすべて信号の流れに従うように決めるということは，図１(a)の例のようにフィードバック・ループが存在する場合には，不可能である。しかし，素子の演算順序がシーケンス図の信号の流れと整合しないために生ずる出力の遅れは，それだけではエラーとは結びつかない。サイクル時間は，ミリ秒のオーダーであるので，このような遅れがシーケンス図で対象とするシステムの動作の上で重要となるケースは考えにくい。

ここでは，シーケンス図の信号の流れと整合しないために生ずる出力の遅れが発端となって，「出るべき出力が出なくなる」というようなクリティカルなエラー事象を取り上げることにする。

図１(b)に，シーケンス制御回路をプログラムで実現した場合の回路の動作を表わしたタイムチャートを示す。(ｉ)は，素子の演算順序としてＡの前にＤやＦを実行する場合であり，(ii)は，Ａの後にＤやＦを実行する場合である。(ｉ)は，Ａについて見ると明らかなように，シーケンス図の信号の流れに従った正解のケースである。(ii)は，正解のケースとは，単なる時間遅れではない，本質的に異なる挙動を示すエラーのケースとなる。

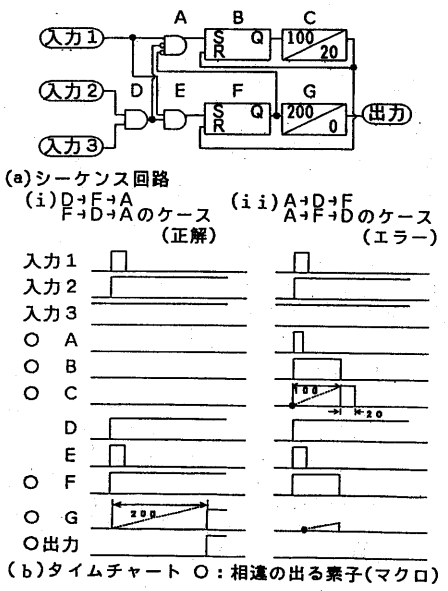


図１　演算順序によるエラー例

ケース(ii)でのエラーの発端は，ＡＮＤ素子Ａの雑音的なパルスを受けてメモリＢがセットされ，タイマーＣで100msの遅れを持った立上りの信号を生ずる所にある。ケース(ｉ)では，これは生じない。一方において，ＡＮＤ素子Ｄ，Ｅを介してメモリＦはセットされ，タイマーＧは200msの遅れを持った立上りの信号を生ずるべく動作する。ここで，ケース(ii)の場合に，タイマーＣの信号立上りが，メモリＦをリセットし，タイマーＧをリセットしてしまう。この結果，ケース(ii)では，タイマーＧで信号が立上がることなく，ケース(ｉ)とは本質的に異なる出力を生ずる。

最初の素子Ａの雑音的なパルスは，素子Ａの演算順序が，シーケンス図の信号の流れに従わないために生じたものに過ぎず，次の演算サイクルでは，本来の値に復帰している。しかし，このパルスは，メモリーＢやタイマーＣを介して，タイマーＧに伝わり，あってはならないタイマーのリセットを生じている。

(2) ソフト設計検証

ソフト作成の手順を設計段階でのエラーを防止するという面から見る。第１のステップでは，シーケンス図の素子に対応するプログラムのモジュールを作成できるので，この結果を改めて検証する必要はない。以下，このモジュールをマクロと称する。第２のステップでは，ユーザの経験的な判断に依る所が大きく，場合によっては，上記のようなエラーを生ずる可能性がないとは言えない。したがって，ソフトＣＡＤシステムにおける課題の一つは，第２のステップでユーザが決めた「素子の演算順序」をいかに検証するかという点にある。

(3) 設計検証へのアプローチ

今回採用した「素子の演算順序」を検証するアプローチは，エラー発生メカニズムに注目し，ユーザの入力データをチェックし，エラーの診断に有益なデータを提供するという実際的な支

援をすることである。素子の演算順序という点からは，次のようなエラーだけを問題とすればよい。エラーは，演算のサイクルのズレが生ずるという起因事象に端を発し，このズレがタイマーのセット・リセットのタイミングに重なるというようなクリティカルなエラー事象に発展することにより生ずる。したがってこの種のエラーの対策は次のようになる。

(a) 起因事象を少なくすること

素子の演算順序ができるだけ信号の流れに沿うように決定することである。これはシーケンス図にフィードバック・ループがない場合には，完全に実行できる。フィードバック・ループがある場合には，ループをどこかで見掛け上カットして実行する。

(b) ループのカットの位置を固定すること

メモリーのリセットのループであれば，リセット信号の入力部でカットする。この位置を固定することにより，潜在的なエラーの起因事象を限定できる。

(c) エラーの診断に有益なデータを得ること

ループのカットによる影響がクリティカルなエラー事象が発生するための必要条件を記号表現で求める。

3. 演算順序の検証手法

3.1 検証アルゴリズムの全体フロー

素子の演算順序を検証するためのアルゴリズムのフローを図2に示す。この図では一体になっているが，処理の内容を2つの検証ステップに対応させて，2分することができる。

(a) シーケンス図の信号の流れに基づいて，必要があればフィードバック・ループをカットした上で，素子の演算順序が満たすべき制約条件を抽出し，与えられた演算順序がそれらの制約を満たすことを確認する処理。これについては，3.2節で説明する。

(b) フィードバック・ループをカットした場合，出力の遅れの影響によりクリティカルなエラー事象が発生するための必要条件を求める処理。この部分は，図2において一番下のブロックに示してある。これについては，3.3節で扱う。

3.2 信号の流れに関する制約条件

(1) 制約ネットワーク

図2において，前節の(a)に対応する部分で中心となるのは，「順序に関する制約ネットワーク作成」，「演算順序と制約条件の照合」の両ブロックである。素子の演算順序がシーケンス図の信号の流れに従うための制約条件は，図2の右上に掲げたような制約ネットワークで記述できる。図によれば素子Qの演算には，素子I4，N，Vが先行する必要がある。矢印は，シーケンス図の素子の入出力関係，すなわちマクロの演算における順序関係を示している。

制約ネットワークがループを含まない場合，ネットワークで表わされ制約条件すべてを満足した演算順序を得ることができる。このような演算順序は，一般には，一義的に決まるわけではない。演算順序の検証は，解の一つが求まっているか否かを確認する問題である。

制約ネットワークがループを含む場合，取りあえずループを除いて，制約ネットワークが，演算順序に関して首尾一貫した制約条件を表わせるようにする。図2の制約ネットワークには，素子Tに関連して，(i) N → Q → T → N，(ii) V → Q → T → Vというループが含まれている。この場合，もとのシーケンス図で素子N，Vがメモリであり，(i)の場合T → Nが，(ii)の場合T → Vがリセット信号となっていたので，この部分をカットし，ループを含まない制約ネットワークを得ている。

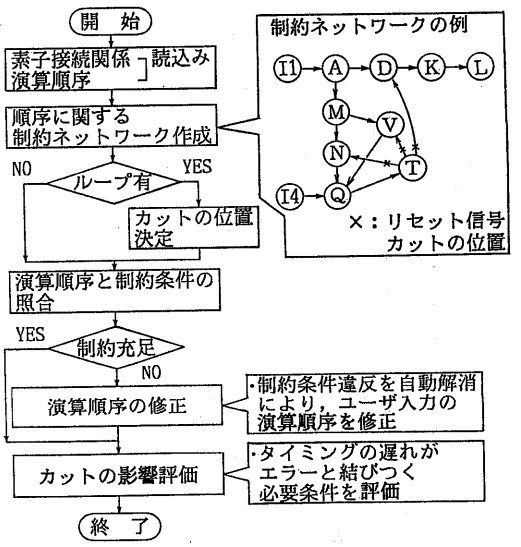


図2 演算順序を検証するための手順のフロー

(2) 演算順序と制約条件の照合

上記(a)のステップの内容をまとめる。素子に対応するマクロの演算を，シーケンス図の素子間の入出力関係を厳密に保ってサイクル時間内に実行できれば，マイコンの逐次処理的論理をシーケンスで表現した並列処理的論理に対応させることができる。シーケンス図がフィードバック・ループを含まない場合，マクロの演算順序が入出力関係の制約条件を満足することを検証すれば良い。シーケンス図の素子間の入出力関係から決まる処理の順序を制約ネットワークで表し，与えられた演算順序を照合する。この場合，入出力関係の制約条件を満足するようにマクロの演算順序を決めることが可能であり，与えられた演算順序が満足しない場合，演算順序を修正する手順を適用する。

シーケンス図がフィードバック・ループを含む場合，サイクル時間内のマクロの演算において，シーケンス図の素子間の入出力関係を保てないマクロが存在する。ループが戻る位置でカットすれば，それ以外の部分では，上記の手順を採ることができる。この場合，カットの位置のマクロの演算結果は，前述のように，シーケンスに対してサイクル時間だけ一種の「遅れ」を生ずることになる。この影響については，ステップ(b)で評価する。

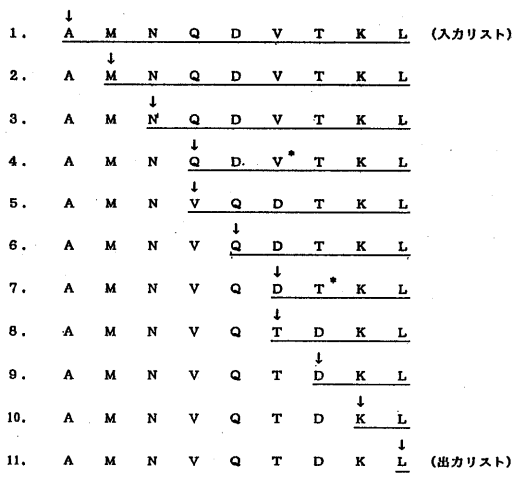


図3　演算順序リストの変化

(3) 演算順序の修正

演算順序の修正の手順は，制約ネットワークに表わされた制約条件を満足するか否かを評価するステップを含んでいる。入力データは，ユーザが指定した演算順序であり，出力結果は制約条件すべてを満たす演算順序である。入力した演算順序が適切であれば，それがそのまま出力される。

演算順序の修正の手順では，演算順序リストに含まれるマクロの名称をノードとして，今どのノードに注目し，制約ネットワークとの照合をしているのかを管理している。そのノードを注視ノードと称し，入力したリストの中で早い順に選んでいく。照合の処理は，演算順序リストを早い順から見ていくことに対応し，注視ノードよりも制約ネットワークで上流側にあるノードがすべて注視済ノードリストに含まれていることを確認することに他ならない。

この手順の中心部は演算順序を更新するステップである。注視ノードよりも上流側にあるノードで，注視済みリストに含まれていないノードがある場合，演算順序のリストでノードの入れ替えを行う。ここでは，まず，そのノードを注視ノードの直前に移すとともに，そのノードを注視ノードにして演算順序の更新の処理を再帰的に呼び出す。これによって，制御が元の注視ノードに戻るときには，前述の制約違反のノードに関連した修正はすべて完了している状態になっているので，確実に処理を進めることができる。図3に演算順序リストの変化を示す。

入力した演算順序が不適切であると見なせるケースとしては，単なる順序の違いの他に，リストに含まれるマクロの重複や欠落も考えられる。重複については，入力したリストをチェックすることで，欠落については，制約ネットワークの情報を用いてチェックすることで対応できる。原理的に，制約条件を満足するように演算順序を決めることができるので，欠落したものを追加するに当たってリスト中での位置は問わない。極端な場合として，演算順序のリストが空である場合にも対応できるので，この手順は演算順序を生成する手段としても使用できる。

3.3　エラー発生のための必要条件

(1) フィードバック・ループのカット

図2において，3.1節の(b)に対応する部分は，「カットの位置の決定」と「カットの影響評価」の両ブロックである。制約ネットワークがループを含まない場合，これらのブロックはバイパスすることになる。

シーケンス図がフィードバック・ループを含む場合，制約ネットワークはループを含む。制約ネットワークのループをカットする位置は，フィードバック・ループで戻る位置とするのが自然である。図2の制約ネットワークの例の場合，カットの位置は，メモリのリセット信号の位置に対応している。

(2) カットの影響評価

ループのカットにより，その部分のマクロの演算結果が，ループ内の他のマクロの演算結果に対して時間的な遅れを生ずる。これが，シーケンス図で表現した並列処理的論理とマイクロ・プロセッサーの逐次処理的論理の相違という形で，設計エラーに結びつくための条件を求める。

一次事象であるメモリーの立下り遅れは，スキャン・サイクルの時間のオーダーであり，シーケンス図に現れるタイマーのセット・リセット時間のオーダーに比し小さいので，この遅れが直接原因となり，出力にエラーを生ずる可能性はきわめて低いと言える。カットの影響評価のステップは，基本的には，カットの影響が原因で万一エラーが生じた場合，その診断のために役立つデータを収集しておくことである。したがって，直接ソフトの検証を支援するわけではない。

ループのカットについては，すべてのループを評価する必要はない。一定時間長さのパルス信号を発生させるための回路のようなローカルで閉じたフィードバック・ループは，評価の対象としない。

カットの影響評価のプロセスは，スキャン・サイクルの時間オーダーでの単純な遅れとは異なるクリティカルなエラー事象を同定するステップ，その事象が生ずるための必要条件を記号表現で求めるステップから成る。例えば，カットの直接的な影響による「メモリーのリセット遅れ」は，クリティカルなエラー事象ではないが，これがタイマー動作と結び付いた「タイマーのセット・リセット誤動作」は，クリティカルなエラー事象である。

(3) クリティカルな事象の同定

ループをカットしたメモリーに注目し，クリティカルなエラー事象を同定する。メモリーから生ずるエラー事象としては，次に示すようなものを挙げることができる。

セット入力との関連がない場合:

①リセット信号の立上り遅れにより単純に出力の立下りが遅れる事象

セット入力との関連がある場合:

②立上り遅れにより，リセット信号が次のセット信号にマスクされて，出力に一時的な「立下り－立上り」パターンが現れない事象

③立上り遅れにより，セット信号にマスクされるはずのリセット信号が現れて，出力をoffにしてしまう事象

ここで，①は，クリティカルではなく，タイマーのセット・リセット誤動作に発展して，初めてクリティカルなエラー事象となる。

②③は，それ自体クリティカルなエラー事象である。

一次事象が発生するための条件を，一次事象発生条件リストとして記号表現しておく。

(4) タイマーへの一次事象の伝播とクリティカルなエラー事象の発生

上記①のケースでは，評価対象のメモリーに対し出力側にあるタイマーを選択し，一次事象の誤動作を伝播させ，タイマーのセット誤動作あるいはリセット誤動作の発生が生ずるための条件を求める。このプロセスは，経路活性化法[4]での「D駆動操作」に似ている。ここでは，メモリー出力での一次事象を保存しつつタイマーまで伝播させる。

まず，誤動作のパターンを分類する。次の4種類を考えれば良い。

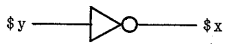
- up　　：変わるべきでない時にoffからonに変わったエラー
- down　：変わるべきでない時にonからoffに変わったエラー
- noup　：offからonに変わるべき時に変わらなかったエラー
- nodown：onからoffに変わるべき時に変わらなかったエラー

誤動作の伝播は，ここでは，メモリーの出力結線からタイマーの入力結線までの経路に沿っ

て，次に示す伝播プロセスを記述するルールを順方向に適用し，実行する。このとき，誤動作が伝播されるための中間結線での条件を，誤動作保存条件リストとして記号表現しておく。

(5) 誤動作の伝播ルール

誤動作が伝播するプロセスを記述するルールとは，例えば，次のようなものである。



(up $y $t) ⟶ (down $x $t)

[NOTのinputに(ある時点で) up のエラーがあれば，outputに(同じ時点で) down のエラーを生ずる。]

ルールは，エラー伝播に関する入出力間の関係として，outputでのエラーのタイプ4種類と素子のタイプ5種類の組合せで整理できる。表1に，エラー伝播に関する入出力間の関係を示す。ルールには，次の2種類が含まれている。

(a)原因となる場合：メモリーのルールでoutputにnodownエラーがあるケースは，メモリーでタイプ①の一次事象が生ずることに対応する。

タイマーについては，outputにupエラーがあるとき，inputのエラーによりセット誤動作を生じ，noupエラーがあるとき，リセット誤動作を生ずる。これらは，タイマーでのクリティカルなエラー事象に対応する。上記の選択されたタイマーでは，(4)の伝播において，「原因となる場合」に対応する。このときの前提条件をクリティカル事象発生条件リストに記号表現しておく。

② 原因とならない場合：表中にある他の組合せのルールである。これらのルールが対象とするプロセスは，原因に関係せず，エラーを伝えることである。

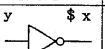
(6) クリティカルなエラー事象が生ずるための必要条件の決定

誤動作が伝播されるための中間結線での条件は，入力側に伝播させて書き換えることができる。このプロセスは，経路活性化法での「後方操作」に似ている。しかし，テストのための入力データをきめる場合と異なり，中間結線での条件を入力結線での条件までブレークダウンする必要は必ずしもない。

クリティカルなエラー事象が生ずるための条件は，つぎのようになる。

表1 エラー伝播に関する入出力間の関係

——：時間軸　……：制約条件

素子 / エラー	NOT	AND	OR	メモリ	タイマー
					
エラー 真 (up $x $t)	(down $y $t)	・$y₁ だけにエラー (and (on $y₂ $t) (up $y₁ $t)) ・両方にエラー ①$t で両方 up ②up と down の組合せ	・$y₁ だけにエラー (and (off $y₂ $t) (up $y₁ $t)) ・両方にエラー ①$t で両方 up	・メモリ原因ではない (and (off $z $t-1) (up $s $t))	・スルーの場合 (up $y $t-$t_1$) ・原因となる場合 (and (off $y $t-$t_1$-1) (on $y $t-$t_2$) (nodown $y $t-1))
(down $x $t)	(up $y $t)	・$y₁ だけにエラー (and (on $y₂ $t) (down $y₁ $t)) ・両方にエラー ①$t で両方 down	・$y₁ だけにエラー (and (off $y₂ $t) (down $y₁ $t)) ・両方にエラー ①$t で両方 down ②down と noup の組合せ	・メモリ原因ではない (and (on $z $t-1) (up $r $t))	・スルーの場合 (down $y $t-$t_2$)
(noup $x $t)	(nodown $x $t)	・$y₁ だけにエラー (and (on $y₂ $t) (noup $y₁ $t)) ・両方にエラー ①$t で両方 noup ②noup と down の組合せ	・$y₁ だけにエラー (and (off $y₂ $t) (noup $y₁ $t)) ・両方にエラー ①$t で両方 noup	・メモリ原因ではない (and (off $z $t-1) (down $s $t))	・スルーの場合 (noup $y $t-$t_2$) ・原因となる場合 (and (off $y $t-$t_2$-1) (on $y $t-$t_2$) (down $y $t')) $t-t_2 < $t' < $t-$t_1$+$t_2$
(nodown $x $t)	(noup $x $t)	・$y₁ だけにエラー (and (on $y₂ $t) (nodown $y₁ $t)) ・両方にエラー ①$t で両方 nodown	・$y₁ だけにエラー (and (off $y₂ $t) (nodown $y₁ $t)) ・両方にエラー ①$t で両方 nodown ②up と nodown の組合せ	・メモリー原因と成りうる。条件は次の通り。 (and (off $r $t-1) (on $r $t) (on $z $t) (on $s $t))	・スルーの場合 (nodown $y $t-$t_2$)

(ⅰ)メモリーのエラー事象①の場合
一次事象発生条件リスト，クリティカル事象発生条件リスト，誤動作保存条件リストに含まれるすべての条件
(ⅱ)メモリーのエラー事象②③の場合
一次事象発生条件リストに含まれるすべての条件

ここで求めた条件は，次のような意味で設計エラーが生ずる必要条件であって，十分条件ではない。まず，ここで想定したクリティカルなエラー事象が，必ずしも出力側のエラーと結び付かない。次に，クリティカルなエラー事象が生ずるための条件に矛盾が含まれるため，クリティカルなエラー事象が成立しない場合が有りうる。

中間結線の条件を入力結線の条件に書き換える処理は，クリティカルなエラー事象が成立する条件に矛盾が含まれるか否かを確認する上で有効である。 前述のようにカットの影響評価は，一般には，検証に寄与しない。しかし，次のような場合には，問題としているループのカットに関してエラーが生じないないことを確認できる。

(ⅰ)クリティカルなエラー事象が生ずるための条件に矛盾が含まれ，条件リストが空リストとなる。
(ⅱ)入力結線の条件が，使用条件で想定できないようなものとなる。

3.4 適用例

開発した手法に基づいてプロトタイププログラムを作成した。プログラムのフローは，図3に示したものである。プログラムは，Common-Lispで書かれている。作成したプログラムを，素子数30のシーケンス制御回路に試験的に適用し，ユーザが与えたマクロの演算順序に関して人為的に設計エラーを加えたケースを用いて，下記の機能を確認した。

演算順序を修正する機能を用いて，種々のエラーを加えたケースに対し，入力した演算順序を修正することができた。3.2節でも述べたように，演算順序リストとして全くの空リストを与えた場合にも，制約条件を満足する演算順序を出力することができる点についても確認した。しかし，結線関係だけから自動的に演算順序を生成した場合，ユーザがＣＡＤシステムでシーケンス図における素子のレイアウトを見ながら，入力した演算順序と見かけ上大幅に異なるものが得られる可能性がある。開発した機能は，ユーザが入力した演算順序に近く，かつ制約条件を満足する演算順序を求めることができている。この意味では，自動生成よりは，ユーザにとって使い勝手の面で良いと考える。

カットの影響でエラーが生ずる必要条件を求める機能については，(a)メモリー単独でエラーを生ずる条件，(b)メモリーのリセット遅れがタイマーのセット・リセット動作エラーに結びつくための条件を，上記回路のメモリーに対して求めた。

4．ま と め

ディジタル装置のソフトウェアとシーケンス図との間の整合性の検証を支援する手法を開発した。本手法は，シーケンス図の素子に対応するプログラム・モジュールの演算順序の検証を，エラーの発生メカニズムに注目して支援するもので，次の２ステップから成る。

(i) 演算順序がシーケンス図の信号の流れに従うように設定されていること，すなわち入出力関係から定まる制約条件を満たすことを確認する。フィードバック・ループが存在する場合は，指定箇所でループをカットし，上記事項を確認する。
(ii) フィードバック・ループをカットした場合，その影響が，エラーの発生メカニズムにおいてクリティカルなエラー事象にまで発展するための必要条件を記号表現で求める。

開発した手法に基づいてプロトタイプ・プログラムを作成し，テスト問題として選定したシーケンス回路に適用し，その機能を確認した。

［参 考 文 献］

[1] 松田ほか：シーケンス制御回路の設計検証ステム(Ⅰ)，情報処理学会第36回全国大会，3U-9 (1989)
[2] 山田ほか：シーケンス制御回路の設計検証システム(Ⅱ)，情報処理学会第36回全国大会，3U-10 (1989)
[3] 長谷川：シーケンス制御総論，計測と制御，Vol.27，No.8，20 (1988)
[4] 樹下ほか：VLSIの設計検証Ⅱ－論理とテスト，岩波書店 (1985)