

計算機を用いた計算機論理のデバッグについて*

加藤 満 左 夫**

戸 田 巖**

中 村 彰**

山 田 正 計**

要 約

新たに設計された計算機論理を製作前に別の計算機を用いてシミュレートし、論理デバッグを行なう方法は、計算機の完成時期をはやめ、調整の人手を節約するのにきわめて有効である。

本文はさきに発表された LSS¹⁾ (Logic Simulation System) を用いて、実用規模の計算機をシミュレーションにより論理デバッグする場合、考慮すべきいくつかの問題点を考察し、これを考慮して進められた計算機設計の一例を述べる。

1. ま え が き

新しいスイッチング回路、たとえばディジタル計算機の論理設計から製作までの一連の過程において、回路の電気的・機械的・不良・不安定などを別として、純粋に論理的な誤りを考えると、

- (1) 論理設計自体の誤り
- (2) 布線表作成における誤り
- (3) 布線作業における誤り

および

- (4) これら各段階を通じての資料転写における誤り、などがある。

これらの誤りをあらかじめ除去すれば、調整段階での労力と時間を節約し、機器完成をはやめる上で大きな効果が期待される。

このような論理的誤りを除去するためには、

(1) に対しては、できれば設計そのものを自動化(回路合成)することが望ましい。しかし、これは現段階では実現にはほど遠い。次善の策としてデバッグを機械化することを考えるのがよからう。デバッグの機械化の一方法としては、設計された回路の動作を、たとえば、既製の計算機でシミュレートしてみて、その応答が設計目的と一致しているかどうかを検査し

て、論理的誤りを発見し、これを除去してデバッグする方法がある^{2),3)}。

(2) に対しては、布線表の作成を計算機を用いて自動的に行なわせる^{3),4)}。

(3) に対しては、布線作業の自動化を考えるか、布線検査を厳重にする。たとえば、検査を試験機を用いて自動的に行なう。

(4) に対しては、転写作業が最少限になるよう設計、製作の手順を工夫し、さらに照合を厳重にする。などの一貫した対策が望まれる。

これらのうち、論理設計は設計者の思考に依存する性質上、誤りを犯しやすい。また製作の基本をなすものであるから、誤りによって起こる影響が大きく、調整段階で大きな時間と労力の浪費を招きやすい。特に先行制御などを採用した計算機では、思いがけない誤りから大幅な設計変更を余儀なくされる場合もある。前述の布線表作成の自動化、布線あるいは布線検査の自動化なども、誤りの多い論理設計資料に基づいて行なったのでは、その効果も半減するであろう。

さらに、積極的に調整段階を能率的にする方法として、シミュレーションの結果得られる論理回路の動作を記述するタイムチャートを利用することも考えられる。

以上の理由によって、われわれは新計算機の設計にあたり、論理設計の誤りの排除と、調整用のタイムチャート作成を第一にとりあげ、LSS¹⁾を用いて論理動作のシミュレーションを行なった。

布線表作成の自動化、布線検査の自動化についても検討されているが、ここでは主として、論理設計に関するものについて、計算機設計の手順とシミュレーションシステム構成の考え方を述べ、さらに実例について報告する⁵⁾。

2. 計算機設計の一手順

設計者の意図は誤りなく製造者に伝えられなくてはならない。そのためには、その表現が正確かつ簡潔で容易に了解されることが必要である。この観点から

* Use of Computers in debugging Computer Logic, by Masao Kato, Iwao Toda, Akira Nakamura and Masakazu Yamada (The Electrical Communication Laboratory)

** 電気通信研究所

は、設計内容を最終的には各素子の入力論理式によって表現するのがよからう。

何人かの設計者によって設計を分担することや、設計終了後の変更の容易さを考慮すると、素子ごとに1枚の設計カードを用いるのがよい。設計カードに設計上のコメント、説明図面番号、通し番号、設計日付などを付記すれば、素子相互の関連を了解しやすくなることができる。

設計の変更は訂正カードを作りこれと原カードを差しかえればよく、資料の更新が正確容易に行なえる。差しかえられた旧カードは訂正の記録として残すことができる。なお、通し番号を付することは、紛失を防ぎ、整理を容易にする。

またこのカードに布線表作成に必要な事項たとえば使用するゲートの種類、入力数、実装位置などを書き加えて設計製作を通してのマスターファイルとして利用すれば、転写による誤りをなくすることができる。

論理設計の内容は、計算機を用いるシミュレーションの結果によりデバッグされる。

それには、まず設計カードの内容を計算機に入力可能な形にする必要がある。たとえばカードの内容を紙テープにさん孔する。これを設計カードの訂正に従って訂正し、マスターテープとして、設計製作を通じて利用することもできる。たとえば計算機を用いてこれから直接布線表を作成する。

設計内容をカードからテープに移す時の誤りは、論理デバッグの段階で人手と時間の浪費を招くので、前もって十分除去しておかなくてはならない。これには、たとえば、同一設計カードから独立に2本のテープを作り、これを照合訂正したものをマスターテープとすればよい。

計算機を用いれば、設計者の不注意による書き誤り設計上の制約条件の逸脱などを簡単に検査できる。これらをシミュレーションに先だち除去しておくことは、能率よくデバッグを進めるうえでも有効である。

論理デバッグは、たとえば次のようにすればよい。まず、シミュレーションの条件を定める。これに従って行なわれたシミュレーションの結果が、設計目的と合致しているかどうか検査する。合致していない場合は、さらにシミュレーション結果を用いて論理の誤りをみつけ出し、これを訂正する。訂正はマスターファイル、マスターテープと順次行なう。マスターファイルにカード方式を取ることににより、訂正は正確容易となり常に最新の資料が利用できる。マスターテープの

訂正は、訂正テープを作り計算機を用いて行なうのがよい。訂正の単位としては、たとえば素子1個をとる。訂正された設計内容に基づき上記のシミュレーション、検査、誤りの訂正を設計目的に合致するまで繰り返す。このような操作を、必要なシミュレーション条件について実行することにより、論理デバッグが進められる。なお、計算機による論理シミュレーションの実行に関しては、次節以下で詳述する。

このようにして、十分にデバッグされた設計資料が得られ、これをマスターファイル、マスターテープとして製作調整に直接利用する。またシミュレーションの結果得られる論理動作を示すタイムチャートは、調整保守に利用することもできる。

3. 計算機シミュレーションのシステム

計算機、電話交換機などの論理設計を複雑な機能を持つオートマトンの実現という面からみる。この立場では論理設計は、方式設計により与えられるオートマトンの機能を実現するための素子間の結合の規定である。この意味で論理設計の結果である論理構成（以下論理構成と略す）は、Completely Specified Sequential Machine とみることができる。一方、方式設計の結果であるオートマトンに対する機能的要求（以下機能的要求と略す）は、オートマトンを完全には規定しえないことが多い。

論理構成は機能的要求を満足するものでなくてはならない。この確認が論理デバッグであるが、一方がオートマトンの機能的表現であるのに対し、他方は素子間の結合を示すもので、相互の直接比較は不可能である。したがって比較のために、なんらかの変換が必要になる。

これには両者をオートマトンの表現——入出力時系列による表現という意味で用いる——にするのが自然であろう。

論理構成を時系列表現に変換することは、前述のように、これが完全に定義されていることから、機械的に実行可能である。これには計算機による論理シミュレーションが効果的である。しかし容易に想像されるように、実用規模の計算機の論理構成をシミュレーションしようとする、この完全なシミュレーション——可能なあらゆる初期条件と外部条件をつくす場合を意味する——を行なうことは、時間的に全く不可能である*。

* 記憶容量1万語；1語12桁の10進計算機を考えると、内部状態数だけでも $10^{1.2 \times 100000}$ となる²⁾。

一方機能的要求をとりあげてみると、これは一般に完全にオートマトンを規定しえないことから、この時系列表現は高々多くの任意性を持った形にしか定まらない。このことは論理構成の機能的要求の時系列による比較を困難にすることはもとより、必要な論理シミュレーションの場合の数が、機能的要求の面からすら定まらないことを示す。

このように完全な論理シミュレーションに基く完全な論理デバッグが、時間的にみて不可能である以上、シミュレーションのシステム構成の問題は、いかにして有効な論理シミュレーションを行ない、デバッグの時間効率を高めようようにするかということに帰する。

ここで一つの論理回路ブロックの論理シミュレーションについて立ち入って考えてみよう。

この論理ブロックにいくつかの素子をつけ加えて、各素子が入力線を有する場合はすべてこのブロックの素子の出力に接続されているようにする。このようにすると、ブロック内の素子は、入力線を有するものおよび全く有しないものの2種類に分けられる。前者はこのブロックの論理構成によって状態が支配されるもので、このブロックの状態を示すものと考えることができるので、これを内部状態素子（以下内部素子と略す）と呼ぶ。後者はこのブロックの論理構成により支配されず、外部よりの入力に相当すると考えられるので、これを外部入力端子素子（以下外部端子と略す）と呼ぶことにする。

同期式論理回路では、初期状態が定まれば、クロックが進むにつれて内部状態は、論理構成および入力時系列に従って変化していく。したがって論理シミュレーションは次の四つの相に分かれる。

- (1) 内部素子の初期状態のセット
- (2) 外部端子の状態のセット
- (3) 内部素子の状態の計算
- (4) 必要な内部素子の状態の取り出し

論理のデバッグは、まず適当な初期状態にセットしたのち、選択された外部端子の条件に従って(2)(3)(4)を必要回数繰り返えし、(4)によって得られる論理シミュレーションの結果が、この論理ブロックの機能的要求を満足するように論理構成を修正するという形で行なわれる。

論理シミュレーションの時間は、素子1個1回あたりの平均シミュレーション実行時間とシミュレーションする素子の延個数の積で示される。さらに延個数は

シミュレーションブロックの素子数とシミュレーションの場合の数の積に等しく、この場合の数は外部端子条件と初期条件および必要クロック数により決まる。

前述のようにシミュレーションのシステム構成の眼目は、有効なシミュレーションを行ない、論理デバッグの時間効率を上げることにある。これは素子1個あたりの平均シミュレーション実行時間の短縮とシミュレーションの条件すなわちブロック分け、外部端子条件および初期条件に当を得ることに相当する。

前記の論理シミュレーションの四つの相のうちで、(1)(2)(4)はこの両者に関係し、また相互およびデバッグのための結果の検査にも関連している。これに対し(3)は平均シミュレーション時間に関係する。これについてはLSSII⁹⁾に詳しく論じられているのでこれはゆずり、ここでは主として(1)(2)(4)を論理デバッグとの関連において、ブロック分け、外部端子の取り扱い、相互接続などの面から考察する。

3・1 論理シミュレーションのブロックの構成

実用規模を持つ計算機の論理構成全体を、そのまま一度にシミュレーションすることは、それを可能とするほどに大容量の計算機を用いるとしても、望ましい方法とは限らない。これは人手により設計された論理構成が機能的に独立なブロックを組み合わせで構成されている点を利用しにくくなるため、シミュレーションの延素子数の増加を招きやすいからである。

一方細分しすぎると、外部端子の増加により、これのセットと状態計算の二重化によりシミュレーションの延素子数（の総計）は再び増加する。

論理デバッグの面から考えるとシミュレーションブロックは大きく、分割が少ない方がよい。このようにすれば作り出さなければならない外部端子条件とこれに対する結果の検査が容易になるからである。さらにこの二つは人手にたよらざるを得ないので、これらできるだけ少なくして、この段階でのデバッグ渡れをなくするようつとめる必要がある。

また個々の問題として、特に使用される場合の数の多い部分(例、SCC、加算器等)の扱いをどうするか、先行する誤りにより以後のシミュレーションがむだになる——デバッグの初期に多いと予想される——のをどうして少なくするかなどの問題がある。

シミュレーションのブロックの構成は、これらの要素を考慮し、さらに使用する計算機の容量の制約のもとで時間的バランスの面から決められる。一般にある程度大きな機能単位に、できるだけ大きさを揃えて分

割して、シミュレーションの単位とし、このうちで特に場合の数の多いものはさらに別に独立させて扱い、デバッグが進むにつれて、大きなブロックへとまとめていくのが妥当であろう。

3・2 論理シミュレーションの実行

前述のように論理シミュレーションは

- (1) 初期状態のセット
- (2) 外部端子のセット
- (3) 内部状態の計算
- (4) 結果の取り出し

によって行なわれる。これらは、どのような方法で実行するかという面と、どのような条件で実行するかという面の二つの面から考察できる。前者は素子当りのシミュレーションの平均実行時間の短縮の点で、後者はシミュレーションの場合の数の節減の点で、有効なシミュレーションのシステムの構成に関係して来る。

論理構成の機能に着目すれば、試験のためにセットすべき初期状態の個数を減少させることができる。またこれがシミュレーションの中でしばしば現われる場合はセットの方法についても、後述の外部端子のセットのような考慮が必要となる。

外部端子について考察すると、論理構成全体としてみても外部端子として扱うべきものと、シミュレーションのブロック構成により外部端子となったものに分けられる。前者は端子条件を人手により作り出さねばならないのに対し、後者は論理シミュレーションするブロックの周辺ブロックの機能を使用する計算機のプログラムでサブルーチン化すれば機械的にセットが可能となる。前者に対しても、セットの方法に関しては、同様にプログラムにより、「シミュレーションブロックごとに毎回セットする方法」から「シミュレーションブロックの内部状態が外部端子の変化を必要とするたびにセットする方法」に能率化できる。さらにプログラムを用いれば、人手で与える外部端子条件は人が理解しやすい形で与えることができるので、誤りの減少と検査を容易にする効果がある。

外部端子条件の選択はシミュレーションのブロックの構成とも関連して、論理構成に深く立ち入って始めて有効なものとして与えることができる。すなわち、各部分の機能的独立性と論理の流れを十分に生かすことであろう。もちろん、このようにして場合の数を減らせれば当然論理デバッグ洩れが生じうが、デバッグ時間を短縮するためにはやむを得ない。

内部状態の計算は他の項目と異ってその方法のみが

問題となる。これに関しては LSS¹⁾ に詳述されているので省略するが、使用される場合の数が特に多い部分(例、加算器、SCC、各種レジスタなど)には次のような考慮が有効である。これらはいずれにしる別ブロックとしてデバッグする方が効果的であるので、外部端子の扱いにし、サブルーチンにより機能を代行させる方法をとれば、シミュレーションの時間短縮に役だつ。さらにこのサブルーチンをこの部分のシミュレーション結果の検査に用いれば、デバッグの完全化と機械化のうえで有益である。

結果の取り出しを考えると、結果は人手により検査されることから、人の見やすい形に変換して取り出すのがよい(例レジスタなどは桁ごとにまとめる)。また内部素子のどれどれを結果として取り出すかという点では、一律にせず多重にするのがよい。たとえば常時取り出すものとシミュレーションの結果に疑わしい点とが現われた場合に再シミュレーションで取り出すものとは別だてにする。数百個以上の素子を含む論理ブロックの場合、時間短縮の面から、このような考慮も大きな問題となる。

前述のようにシミュレーション結果の検査は大部分人手にたよらざるを得ないが、できるだけ機械化するのが望ましい。たとえば、シミュレーションのためにある機能ブロックをサブルーチン化した場合次のような方法が考えられる。すなわち、このブロックへの外部入力条件が妥当なものであるかどうかを検査する部分をサブルーチンに付加し、これが接続される論理構成の検査、人手で作った外部端子条件の検査に役だてる。またサブルーチン化したブロックの論理シミュレーションの結果の検査に、この代行サブルーチンを用いることもできる。

3・3 相互接続の検査

論理シミュレーションの時間短縮の点から、あるいは使用する計算機の容量の点から論理構成全体を機能ブロックに分割してシミュレートした場合、これらの相互の接続の検査が必要となる。設計は機能ブロックごとに進められることが多いので、相互の打ち合わせ上での思い違い、または時間的ずれによる思い違いは避けられないであろう。このような誤りの性質上、相互接続の検査は、機械化するのが望ましい。これに必要な論理シミュレーションは相互の接続を確認しうる程度でよい。したがって記憶容量の制約から分割してシミュレートした場合でも、磁気テープなどを使用して十分実現可能である。

以上のような点を考慮してシミュレーションのシステムを構成すれば、任意の大きさの論理構成をシミュレーションにより効果的にデバッグすることができ
る。

4. 实施例

以上の考察に基く実施例として NEAC 2203 を用

いて行なった **CAMA** 大局用計算機の論理デバッグを説明する。

4. 1 計算の手順

(1) 論理設計

設計対象は、CAMA 大局用計算機^{(7)~(9)}（以下 CAMAC と略称する）で、これは記憶容量 1 万語、演算速度、加算 30 μ s のものである。基本論理回路はダ

[illegible]

第1図 設計カードの一例

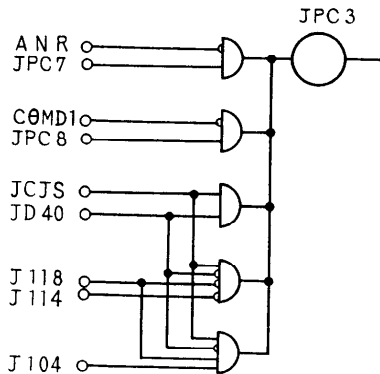
イナミック式; AND-OR ダイオード・トランジスタロジックを用いる。

論理設計の結果はトランジスタ1個あたり1枚の設計カードに、AND-OR 入力論理式として記述する。トランジスタには1個ごとに mnemonic symbol で名称を与える。設計カードの一例を第1図に示す。これの設計内容は第2図のごときものである。これが設計製作を通じてのマスターファイルとなる。

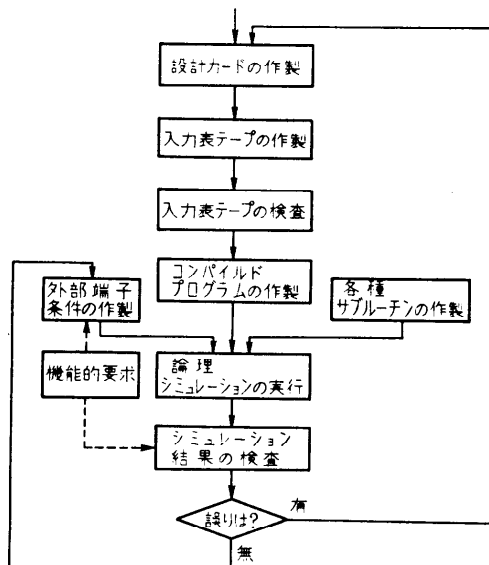
(2) 論理デバッグ

論理設計の結果は論理デバッグする。この手順を流れ図で第3図に示す。

まず設計カードの内容を計算機に入力可能な形に変



第2図 第1図の設計内容



第3図 論理デバックの手順

換する。これには LSS II¹⁾ で指定された形式にさん孔した紙テープを用いた。素子の入力論理式を表現しているの、これを入力表テープと呼ぶ。これの一例を第4図に示す。誤さん孔を除くために同一設計カードから独立に2本のさん孔テープを作り、これを照合訂正して正しい入力表テープとした。

FIG.5 IC,
 ## ※※ICI,
 01 01 ※IC00,
 02 05 ※※ICB -※I074 ※MCE1 -MCS01
 -MCS02,
 03 03 ※※ICB -※ID60 -※I074,
 04 03 ※IC04 MCS01 -※I074,
 05 03 ※ICMI -※MC45 -SC130,
 06 03 ※IC07 ※MC45 -※I074,
 07 01 ※IC03,
 08 02 ※※ICI -※IC26,
 ## ※ ICMI,
 01 04 ※IC01 ※MCE1 -SC130 -※JPC5,
 02 02 ※IC02 -※JPC5,
 ## ※※ICB,
 01 03 ※ICMI ※MC45 -※JPC5,
 02 03 ※ICMI SC130 -※JPC5,

第4図 入力表テープの一例

訂正された入力表テープは次の段階で LSS II¹⁾ により検査する。検査の項目としては次のようなものがある。

(i) 論理設計の回路的制約条件の逸脱。たとえば入力ゲート数が規定数以内か、入力数の指定が実際の入力の数と一致しているか、素子が禁じられた組み合わせを含むことはないかなど。

(ii) 入力論理式のトランジスタ名称の誤り。たとえば名称の書き誤りはないか、シミュレーション単位外のトランジスタ名称を含むことはないかなど。

(iii) さん孔形式の誤り。たとえば入力表テープの照合訂正洩れにより LSS II¹⁾ の規定に反しているものはないかなど。

このようにして検査訂正された入力表テープが設計製作を通じてのマスターテープとなる。次にマスターテープから LSS II¹⁾ のコンパイラによりコンパイルプログラムを作る。これが実際に計算機でシミュレーションを実行するプログラムである(この詳細は文献1参照)。

一方論理シミュレーション実行のための各種サブルーチンおよび外部端子条件を作成する。これらと、

LSS II¹⁾の制御ルーチンにより論理シミュレーションが実行される(この部分の詳細は後述する)。

シミュレーションの結果は、論理設計者が検査する。誤りを発見したならば新たに設計カードを作り、原カードと差しかえてマスターファイルを訂正する。上記の手順で訂正部分についてのみのコンパイルドプログラムを作り、これを原コンパイルドプログラムの訂正として付加し、これを用いてシミュレーションの実行、結果の検査を行ない誤りがなくなるまで繰り返す。なお訂正はトランジスタ1個を単位とした。誤りがなければ外部端子条件を変えて論理シミュレーションの実行、結果の検査、誤りの訂正を続け、論理デバッグを進める。

(3) 設計資料の利用

設計製作を通じてのマスターファイルである設計カードは常に最新の形で保存され、十分に論理デバッグされる。これは人が見やすい形になっているので、人手で作業する場合に利用する。

入力表テープはマスターファイルの写しにあたり、計算機に入力可能な設計資料で、これをマスターテープとして計算機など機械により作業する時に利用する。

このように資料を用いれば、設計製作の各段階で、資料の転写により起る誤りを排除できる。

また論理シミュレーションで得られるタイムチャートは、調整保守の資料としても有益である。さらに論理シミュレーションでは、素子の状態の履歴が残るので、これを積極的に利用して、調整保守用の資料の作成を検討している。

4・2 計算機シミュレーションのシステム

(1) シミュレーションのブロック構成

全体を大きな機能単位に分割し、主制御部、演算制御部、割り込みおよび手動制御部、入出力制御部などを各々一つのシミュレーションの単位とした。これらはいずれも数百個のトランジスタを含み、LSS II¹⁾で一度にシミュレーションできる範囲にあり、相互の接続線なども妥当な数である。

さらにこのシミュレーション単位の中でのサブブロックについて、主制御部の例をあげると SCC、インデックスアダー、番地比較器などを別ブロックとし各々独立に論理デバッグを行なった。主制御部全体のシミュレーションでは、これらはいずれもサブルーチンにより機能を代行させた。

(2) 論理シミュレーションの実行

例として主制御部をとる。これの規模は次のようなものである。

シミュレーション・トランジスタ数	767個
コンパイルド・プログラム	7,400語
各種サブルーチン	700語
シミュレーション時間	0.5~1分*/クロック

前節で述べたようにシミュレーションのシステム構成上、大きな問題点である外部端子のセットは次のようにした。シミュレーションに用いる計算機の中に擬似メモリを設けここに CAMAC のプログラムを置く。CAMAC のメモリ制御機能をサブルーチン化したプログラムにより CAMAC のメモリ使用の要求があるごとに指定された番地の内容を CAMAC 主制御部のメモリ端子にセットし、またはその逆を実行する。シミュレートされる計算機のプログラムを実際にシミュレーションで実行することにより、機能的要求への見通しがよくなる。外部端子条件は擬似メモリに置かれる命令とその組み合わせにより示され、効果的にデバッグを進めることが可能となる。

使用したサブルーチンは次の7種である。

- (i) 擬似メモリへの読み込みルーチン
- (ii) 擬似メモリ制御ルーチン
- (iii) 初期状態のセット・ルーチン
- (iv) 演算制御ルーチン
- (v) SCC, インデックスアダー, 番地比較器ルーチン
- (vi) プリント・ルーチン
- (vii) バックおよびアンバック・ルーチン

(i)(ii)は上述の擬似メモリ用ルーチンであり、(iii)は各トランジスタを CAMAC の起動状態にするプログラム (iv)は主制御部の周辺ブロックとして最も関係の深い演算制御部の機能を代行するプログラム、(v)は主制御内のサブブロックの機能を代行するプログラム (vi)はシミュレーションの結果を人が見やすい形にして取り出すプログラムで必要なレジスタなどを印刷する。(vii)は上記の各サブルーチンの中でビットごとになっている情報を桁ごとに集め、またはその逆を行なうたびに用いるいわばサブ・サブルーチンである。

論理シミュレーションは、まず LSS II¹⁾ ルーチンの読み込み CAMAC 論理構成のコンパイルドプログラムの読み込み、各種サブルーチンの読み込み、制御プログラムの読み込みを行う。制御プログラムはシミ

* 結果の取り出し状況により変化する。

シミュレーション実行プログラムであるコンパイルドプログラム、各種サブルーチンなどの全体の制御を行なうものである。この一例を第5図に、これにより指定される実行内容を第6図に示す。

```

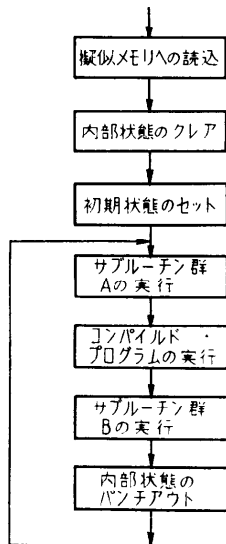
ZC
EXCT Y, Y, 8250 Y.
CLWD.
EXCT Y, Y, 8450 Y.
EXCT Y, Y, 7500 Y.
SIML 1 Y, Y.
EXCT Y, Y, 8500 Y.
PALS.
JUMP -4 X.
※※※※
ZE

```

第5図 制御プログラムの一例

[註]

サブルーチン群 A
・演算制御ルーチン
・SCC, インデックスアダー
および番地比較器ルーチン
サブルーチン群 B
・擬似メモリ制御ルーチン
・プリントルーチン



第6図 第5図の実行内容

各種プログラムの読み込みののちに制御プログラムによりシミュレーションが実行される。第6図のように、まず擬似メモリに置かれる CAMAC のプログラムが読み込まれる（これの一例を第7図に示す）。以後サブルーチンコン群パイルドプログラムが指定の順

TP-0017 MR TEST

000 Z	300083210002 Y
300000000007 Y	300083214002 Y
300000004007 Y	-300093210004 Y
-300000000008 Y	-660000000001 Y
300083210002 Y	987654321987 Y
300083214002 Y	00010001 Z
-300093210004 Y	300000000007 Y
660000004000 Y	300000004007 Y
1234567891 3 Y	-300000000008 Y
000000000005 Y	300083210002 Y
000000000004 Y	300083214002 Y
0004000 Z	-300093210004 Y
300000000007 Y	011000000000 Y
300000004007 Y	246813579246 Y
-300000000008 Y	660 Z

第7図 CAMAC のプログラムの一例

序で行なわれ、シミュレーションが実行される。結果は論理デバッグに便利のように、主要レジスタなどはプリントルーチンにより見やすい形に10進変換してクロックごとに取り出す。この一例が第8図である。また必要に応じて全部または一部のシミュレーショントランジスタの状態を紙テープにパンチアウトしておき、オフラインで見やすい形に変換する。これには、タイムチャートの形、または各クロックごとにオンのトランジスタ名称の形などを用い、結果の検査に役だてた。これらの一例を第9図、第10図に示す。

各種のサブルーチンには入力条件を検査する機能を持たせた。たとえば擬似メモリ制御ルーチンでは同時

TP-0020 JUMP INTERRUPTION

	I	J	P	SC	AW	AR	IM
00	+000000000000	+000000000000	+000000000000	0	00000	00000	
01	+000000000000	+000000000000	+000000000000	0	00000	00000	
02	+000000000000	+000000000000	+000000000000	0	00000	00000	
03	+300000000011	+000000000000	+000000000000	0	00000	00000	
04	+300000000011	+000000000000	+000000000000	0	00001	00000	
05	+300000000011	+300000000011	+000000000000	0	00001	00001	
06	+300000000011	+300000000011	+000000000000	0	00001	00011	
07	+660000000003	+300000000011	+000000000000	0	00001	00011	
08	+660000000003	+300000000011	+000000000000	0	00002	00011	
09	+660000000003	+300000000011	+620300009923	0	00002	00002	00
10	+660000000003	+660000000003	+620300009923	0	00002	00002	00011
11	+660000000003	+660000000003	+620300009923	0	00003	00002	00011

[註] I: 命令レジスタ 1, J: 命令レジスタ 2, P: オペランドレジスタ
SC: SCC, AW: アドレススイッチ, AR: アドレス レジスタ

第8図 主要レジスタの印刷例

TP-0003 JUMP-1

	※※ I J	※※※ J J	※※
	※※ CC	※※ J CC	SM
	I I WW	J J PWW	TC
	CC 2 2	CCC 3 3	RE
	I B 1 2	PX 5 1 2	Q 1
00	X . . X	 X	. .
01	X . . X	 X	. X
02	. . . X	 X	. .
03	. X . X	 X	. .
04	X . X X	 X	. .
05	X . . .	X . . . X	. X
06		X . . . X	. .
07	. X . .	X . . . X	. .
08	X . X .	. . . X X	. .
09	X . X X		. .
10	X . . .	. . X . .	. X
11	X . . X		. .
12	X . . X	 X	. X
13	. . . X	 X	. .
14	. X . X	 X	. .
15	. X . X	 X	. X
16	X . X X	 X	. .
17	X . . .	. . X . X	. X
18	X . . X	 X	. .
19	X . . X	 X	. X
20	. . . X	 X	. .
21	. X . X	 X	. .
22	X . X X	 X	. .
23	X . . .	 X	. X
24		 X	. .
25	. X . .	. . X . X	. .
26	X . . X	 X	. .
27	X . . X	 X	. X
28	. . . X	 X	. .
29	. X . X	 X	. .
30	X . X X	 X	. .
31	X . . .	 X	. X
32		 X	. .
33	. X . .	. . X . X	. .
34	X . . X	 X	. .
35	X . . X	 X	. X
36	. . . X	 X	. .
37	. X . X	 X	. .
38	X . X X	 X	. .

第9図 主要トランジスタのタイムチャート例

に CAMA 主制御の2個以上から取り出し要求が出ていないかなどの検査である。

また加算器などでは論理シミュレーション結果を機能代行サブルーチンの結果と比較することにより検査の自動化を計った。

このような方法は、デバッグの機械化と完全化の上

0 0	I CW 2 3	J CW 2 2	J CW 3 2	※ ADVW	※ IC 0 1
	※ IC 1 1	※ IC 1 2	※ IC K 8	※ IG K 2	※ J CPN
	※ IC 0 0	※ IC 4 5	※ IC I	STRQN	
0 1	I CW 2 3	J CW 2 2	J CW 3 2	MGE 3 1	PGD 0 2
	PGD 0 3	PGD 0 4	PGD 0 5	PGD 0 6	PGD 0 7
	PGD 0 8	PGD 0 6	PGD 1 0	PGD 1 1	PGD 1 2
	※ ADVW	※ IC 0 1	※ IC 0 6	※ IC 1 1	※ IC 1 2
	※ IC 2 6	※ IC K 8	※ ID 1 0	※ ID 1 1	※ ID 5 2
	※ IG K 2	※ J CPN	※ JD 1 0	※ JD 1 1	※ JD 1 2
	※ MC 0 0	※ MC 0 4	※ MC 4 5	※ MCC 3	※ MCE 1
	※ MR 1 1	※ MRN 2	※ MRN 3	※ IC I	SCCA 5
	STRQN				
0 2	I CW 2 3	J CW 2 2	J CW 3 2	MGE 3 1	※ ADVW
	※ IC 0 6	※ IC 1 1	※ IC 1 2	※ IC K 8	※ IC M 1
	※ ID 1 0	※ ID 1 1	※ IC 5 2	※ IG K 2	※ J CPN
	※ JD 1 0	※ JD 1 1	※ JD 1 2	※ M 0 2 2	※ M 1 0 1
	※ M 1 0 4	※ M 1 1 8	※ M 1 2 2	※ MC 4 5	※ MC 5 7
	※ MCC 3	※ MRN 2	※ MRN 3	※ IG 3	SCCA 5
	STRQN				
0 3	AWG 1 1	I CW 2 3	IDG 1 0	IDG 1 1	J CW 2 2
	J CW 3 2	MGE 3 1	※ ADVW	※ I 0 2 2	※ I 1 0 1
	※ I 1 0 4	※ I 1 1 8	※ I 1 2 2	※ IG 0 6	※ IC 1 1
	※ IC 1 2	※ IC K 8	※ ID 1 0	※ ID 1 1	※ ID 5 2
	※ ID 5 9	※ IG K 2	※ J CPN	※ JD 1 0	※ JD 1 1
	※ JD 1 2	※ MC 0 1	※ MC 4 5	※ MCC 3	※ MRN 2
	※ MRN 3	※ IC B	SCCA 5	SCCS 1	STRQN
0 4	I CW 2 1	J CW 2 2	J CW 3 2	MGE 3 1	※ ADVW
	※ I 0 2 2	※ I 1 0 1	※ I 1 0 4	※ I 1 1 8	※ I 1 2 2
	※ IC 0 1	※ IC 0 6	※ IC 1 0	※ IC 1 1	※ IC J 8
	※ IC W 2	※ ID 2 2	※ ID 2 5	※ ID 2 6	※ ID 5 2
	※ ID 5 9	※ IG K 2	※ J CPN	※ JD 1 0	※ JD 1 1
	※ JD 1 2	※ MC 0 0	※ MC 4 5	※ MCC 3	※ MRN 2
	※ MRN 3	※ IC I	※ I G 2	※ J G 4	SCO 1 1
	SCCA 5	STRQN			

第10図 クロックごとのオン・トランジスタの印刷例

で有効である。

(3) 相互接続の確認

(1)で述べたように CAMAC 全体を機能単位に分割して論理デバッグした。この分割は設計の分担とも一致しているので、相互接続の確認が問題となる。相互接続上の誤りは打ち合わせ上の思い違いなどに基因することから、この確認は計算機を用いて行なうのが望ましい。

LSS II¹⁾ そのままでは、記憶容量の制約から CAMAC 全体をまとめて同時に論理シミュレーションできないので磁気テープの利用を考えている。現在LSSグループにより、LSS II¹⁾を発展させた形でこれの実現が計られている。これの完成をまって、全体をまとめて論理シミュレーションを行ない相互接続の確認を行なう予定である。

5. む す び

既製の計算機を用いてシミュレーションにより新計算機の論理デバッグを行なう方法は、製作前に十分にデバッグできる点で効果的である。

論理デバッグの初期には書き誤りなどの単純な誤り

が多数発見される。次第にデバッグが進むにつれ特別な条件でのみ起るような誤りが少数ではあるが発見される。

CAMAC の制御部には先行制御方式⁶⁾ が採用されている。このためシミュレーション結果を用いて状態の履歴をたどることにより発見可能となった誤りもある。

主制御を例にとると設計カードの訂正は仕様変更を含めて約15%，ひととおりのデバッグに要した期間は約1カ月であった。ただし設計終了後人手による検査を全然行わずに、直ちに本稿にのべたデバッグ法を用いた結果である。

また論理シミュレーションを単に論理デバッグに用いるのみにとどめず、計算機の設計、製作、調整、保守などの過程に一貫して取り入れ、さらに効果的に利用することを検討している。

6. 謝 辞

計算機を用いる計算機論理のデバッグを行なうにあたり、御協力下さった日本電気株式会社第一計算機課金田課長始め同課の方々、実行の機会を与えられた当研究所新堀次長、終始御指導御討論をいただき、また

LSS¹⁾ 作成に努力された岸上調査役および LSS グループの方々に、深甚なる謝意を表する次第である。

参 考 文 献

- 1) 高島，津田他；論理構成のシミュレーションプログラム，電子計算機研究会資料，1962年10月
- 2) 伊吹；ディジタル機械論理設計への計算機の応用，電子計算機専門委員会資料，1960年10月
- 3) Leiner, Weinber, Coleman: Using Digital Computers in the Design and Maintenance of New Computers, IRE Trans. EC-10 No.4 (1961-12) p. 680
- 4) 淵，西野；電子計算機に関する自動データ処理，情報処理，1 No. 4 (1960-12) p. 213.
- 5) C. E. Shannon, J. McCarthy; Automata Studies, Princeton University Press.
- 6) 加藤他；計算機を用いた計算機のチェックについて，昭和 38 年電気 4 学会連合大会.
- 7) 岸上他；CAMA 大局用電子計算機の構成，同上.
- 8) 加藤他；CAMA 大局用電子計算機の命令システム，同上
- 9) 加藤他；CAMA 大局用電子計算機のマスターコントロールについて，同上

(昭和 38 年 2 月 8 日受付)