解　説

2. 方式・機能・論理設計における CAD

2.3 方式・機能・論理シミュレーション†

村上道郎†† 平川和之††

1. はじめに

装置，LSI の設計検証に，論理，及び機能シミュレータを使用することは，現在では当然のこととして受け入れられており，方式レベルシミュレーションも開発され一部で使用されている．

しかし振り返ってみると，1970年代前半までは CPU 等の装置設計に論理シミュレータが使用されてはいたが，処理速度が遅い等により十分な効果をあげているとは言い難かった．その後，論理シミュレータの研究が進み，1～2桁の高速化がなされ，更に機能レベルシミュレーションの出現等により，装置設計の検証として十分にシミュレータが活用されるようになった．

LSI 設計においても，70年後半から数千～数万ゲートの論理 LSI が出現し，内部の複雑なタイミングを検証するためにはシミュレーションが有効であること，また，その LSI を試験するためのテストパターンも，シミュレーションなしでは作成不可能となってきたこと等により，論理シミュレーションが急速に普及してきた．最近では，VLSI 化への対処として，高速化とともに，遅延時間，モデル化等の高精度化が図られている．

本稿では，まず論理装置の設計手順として，方式設計，機能設計，論理設計の各設計ステップを概説し，次に各設計ステップにおけるハードウェア記述言語と，対応するシミュレータについて述べる．特に論理設計に対しては，最近の動向としてのストレングス，及び混合シミュレーションについて解説する．

2. 論理装置設計手順

一般に論理装置の設計は，方式，機能，論理の各設計過程を経て行われる．**図-1** に，設計フローと各設計で作成されるドキュメントを示す．方式設計では，製品企画書等を基にして装置の仕様，ハードとソフトのインタフェース条件，システム構成図等を決定する．装置の仕様とは，処理能力，コスト，信頼性，使用技術，外観図，実装方法・条件等であり，またインタフェース条件とは，データ形式，命令及びその形式等である．システム構成図は，CPU，メモリ，I/O 機器等

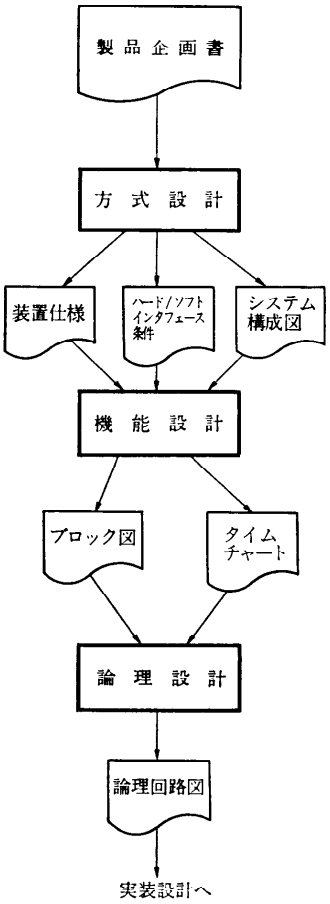


図-1 論理装置設計フロー

† System Level, Functional Level and Logic Level Simulation by Michio MURAKAMI and Kazuyuki HIRAKAWA (OKI Electric Industry Co., Ltd.).

†† 沖電気工業(株)電子デバイス事業部

を使用して，大まかに論理装置の構成を示したものであり，バスのビット幅，ALU，レジスタ，デコーダ等は通常表現されない．**図-2**(a)が，BASYS と名付けられた簡単な計算機システムのシステム構成図である(**3.1** 参照)．この方式設計は，前述の3つの設計の中では，最も人間の発想力を必要とする分野であり，現状の CAD 技術ではサポート範囲が狭く，方式レベルシミュレータが一部で使用されているにすぎない．

機能設計では，システム構成図等を基にして，詳細なブロック図，タイムチャート等を作成する．詳細ブロック図には，各種レジスタ，ALU，RAM，バスのビット幅等が記載されており，システム構成図と比べ，ハードウェアイメージがかなり明確になっている．**図-7**(a)に簡単な計算機のブロック図を示す（**4** 参照)．

ブロック図作成のために，論理装置の動作フローチャート，状態遷移図等を使用することも多く，これらの設計に対する言語[6)~8)]，シミュレータ[9),13),16)]が開発されている．

論理設計では，ブロック図，タイムチャート等を基にして，ゲートレベルの論理回路図を作成する．論理設計におけるシミュレータ，即ち論理シミュレータは，現在ではほとんどの場合に使用されていると言って良い．特に LSI 設計における論理シミュレーションは，シミュレーション用テストベクタが LSI の機能試験用に供せられていることもあり，不可欠なものとなっている．

以上述べた方式設計から論理設計の各レベルにおいて，設計対象となるハードウェアのモデル化言語，即ちハードウェア記述言語が存在する．ハードウェア記述言語には，その構造に着目する構造記述言語と，その動作に着目する動作記述言語とが存在する．これらの詳細については，本特集の「ハードウェア記述言語とその応用」を参照されたい．

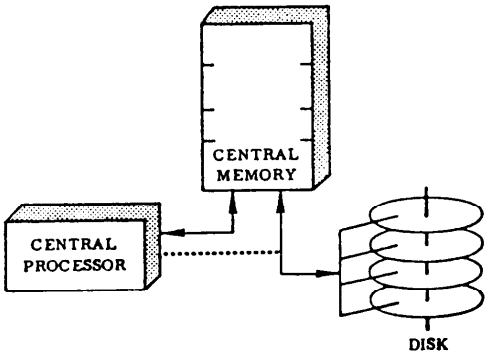
3. 方式設計

方式設計では，データ，命令形式等のハードソフトインタフェース条件，及びシステム構成図等が決定される．方式レベルシミュレーションは，これらの設計結果の検証を行うわけであるが，この時点でのハードウェアイメージは，システム構成図程度でしかない．したがってシミュレーションは，平均命令実行時間等の性能評価，またインタフェース条件の検証を主目的としており，それぞれに対するハードウェア記述言語としては，システムレベル言語とプログラミングレベル言語がある．プログラミングレベル言語はシステムレベルと機能レベルのインタフェース的な言語であり[11)]，方式設計に含めるか機能設計に含めるか意見の分かれるところである．ここでは，ハードとソフトのインタフェース条件の検証を目的とする意味から，方式設計に含めた．

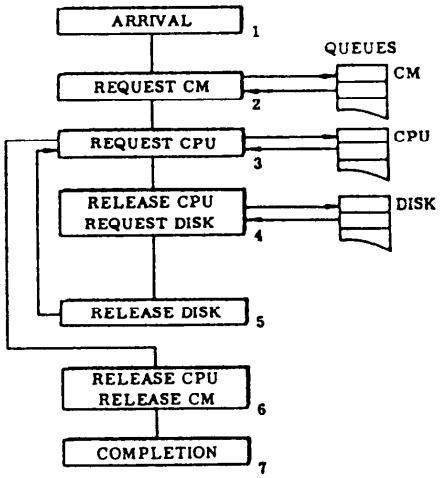
3.1 システムレベル言語

PMS (Processor-Memory-Switch) レベル言語とも呼ばれ，記述されるハードウェアモデルの要素は，プロセッサ，メモリ，スイッチ，周辺装置等である．

システムレベルシミュレーションは，装置全体の特性，例えばメモリ，ディスク等の使用率等の評価を目的とする．例として，文献2) に示されている単純なディスクベースのマルチプログラミング計算機システム，BASYS について説明する．BASYS は，CPU，



(a) 構成図



(b) ジョブ処理フロー

図-2 BASYS システム

メモリ，ディスクで構成されており，BASYS 上のジョブは以下の手順で処理される（**図-2**）．

（1） 複数回の I/O 要求を持つジョブが投入される．

（2） そのジョブに必要な主記憶領域の要求が出される．領域確保が可能であれば，そのジョブを割付ける．否であれば，キューに登録する．

（3） CPU に要求が出される．CPU が空状態であれば，そのジョブが割付けられ，I/O 要求が出されるか実行が完了するまで，そのジョブが CPU を専有する．CPU が空状態でなくビジーであれば，そのジョブはキューに登録される．

（4） ジョブが I/O 要求を出して CPU を解放する．そのとき，CPU キューに登録されている待ち状態のジョブがあれば，それを CPU に割付ける．ディスクが空状態であれば，I/O 処理を行い，ビジーであればキューに登録する．

（5） I/O 処理が完了した時点でディスクを解放し，再び CPU 要求を出す．ディスクが解放されたとき，ディスクキューにジョブが登録されていれば，それをディスクに割付ける．

（6） ジョブが完了すれば，CPU，主記憶を解放する．このとき，CPU キュー，主記憶キューにジョブが登録されていれば，それぞれを割付ける．

（7） ジョブをシステムから解放する．

BASYS の入力は，それぞれ複数回の I/O 要求を持つ複数のジョブである．それらのジョブのシミュレーションのためには，各ジョブごとに，主記憶容量，及びディスク読み書きのための，レコード長，レコード数，CPU 時間等のパラメータ値設定が必要となる．一般に，この設定の良し悪しがシミュレーション結果の信頼性に大きく依存するため，BASYS システムでは，経験等で得られている確率分布に従って，パラメータ値を自動生成している．例えば，**図-3**のように主記憶要求が分布している場合を考える．0 から 1 の間の乱数を発生し，対応する値を得ることで必要主記憶容量を決定する．仮に乱数値が 0.4 であれば，主記憶容量は 35K となる．このように複数のジョブそれぞれに対し，乱数により必要主記憶容量等をパラメータ値として与え，全ジョブ処理完了時の平均待ちジョブ数，平均待ち時間等を評価し，システムが装備すべき主記憶容量，CPU 能力等を決定する．

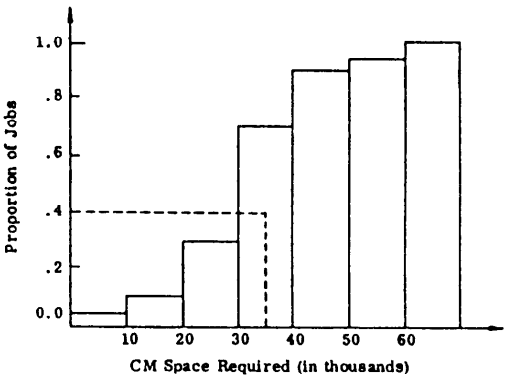


図-3 主記憶要求分布

以上，BASYS 用のシステムレベルシミュレータを例として説明したが，一般のシステムレベルシミュレータも基本原理は同じである．なお，シミュレータ作成のためのプログラム言語としては，GPSS[3]，SIMSCRIPT[4]や一般的な FORTRAN 等が使用される．

3.2 プログラミングレベル言語

計算機システムの動作仕様を，マイクロインストラクションレベルの命令として定義する言語である．代表的なものに ISP[1]，LCD[5]等があり，これらはプログラミングレベルだけではなく，それぞれ機能レベル，論理レベルまでの表現が可能である．LCD によるプログラミングレベルのシミュレーションは，次に続く機能設計検証のために行われる．即ち**図-4**に示すように，プログラミングレベルのシミュレーション結果と機能レベルのシミュレーション結果を比較し，両者が等しければ機能設計が正しいとするものである．LCD によるシミュレーション例を，**図-5**に示す．**図-5**(a)がインストラクションの定義であり，(b)が，(a)で定義したインストラクション群を

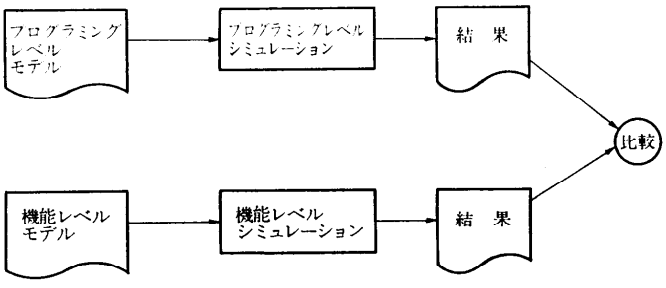


図-4 機能設計の検証

```
[21] MODEL HILEV; CMT----------;
[22]
[23] WHILE CYCLE=FETCH;
[24]  INST:=MAIN(IAR);
[25]  IAR:=INCR(IAR);
[26]  CYCLE:=EXECUTE;
[27] END;
[28] WHILE CYCLE=EXECUTE&OP=PL;
[29]  CYCLE:=FETCH;
[30]  ACC:=MAIN(EA(XR(R),IAR,INST));
[31] END;
[32] WHILE CYCLE=EXECUTE&OP=PLI;
[33]  CYCLE:=FETCH;
[34]  IF R='000'; ACC:=D/0,0,0,0,0,0,0,0,0:7/;
[35]   ELSE; XR(R):=D/0,0,0,0,0,0,0,0,0:7/;
[36]  END;
[37] END;
[38] WHILE CYCLE=EXECUTE&OP=PST;
[39]  CYCLE:=FETCH;
[40]  MAIN(EA(XR(R),IAR,INST)):=ACC;
[41] END;
[42] WHILE CYCLE=EXECUTE&OP=PA;
[43]  CYCLE:=FETCH;
[44]  ACC:=SUM(ACC,MAIN(EA(XR(R),IAR,INST)));
[45] END;
[46] WHILE CYCLE=EXECUTE&OP=PAI;
[47]  CYCLE:=FETCH;
[48]  IF R='000';
[49]   ACC:=SUM(ACC,D/0,0,0,0,0,0,0,0,0:7/);
[50]  ELSE;
[51]   XR(R):=SUM(XR(R),D/0,0,0,0,0,0,0,0,0:7/);
[52]  END;
[53] END;
```

(a) インストラクション定義

```
[228] RUN FETCH.R; CMT----------;
[229]  CYCLE<0>:=FETCH; STOP CYCLE=EXECUTE;
[230] ENDRUN;
[231] CONSTANT REG/3/=UNIQUE,DISP/8/;
[232] RUN PL.R;
[233]  CYCLE<0>:=EXECUTE; STOP CYCLE=FETCH;
[234]  SELECT; INST<0>:=PL,REG,DISP;
[235]         INST<0>:=PL,'000',DISP;
[236]  END;
[237] ENDRUN;
```

(b) シミュレーション実行コマンド

```
WHILE CYCLE<0>=FETCH;
 INST<1>:=MAIN<0>(IAR<0>);
 IAR<1>:=INCR(IAR<0>);
 CYCLE<1>:=EXECUTE; END;
WHILE CYCLE<0>=EXECUTE&INST<0>/0:4/=PL&
                ¬(INST<0>/5:7/='000');
 CYCLE<1>:=FETCH;
 ACC<1>:=MAIN<0>(SUM(XR<0>(INST<0>/5:7/),
                INST<0>/8,8,8,8,8,8,8,8,8:15/));
END;
WHILE CYCLE<0>=EXECUTE&INST<0>/0:4/=PL&
            INST<0>/5:7/='000';
 CYCLE<1>:=FETCH;
 ACC<1>:=MAIN<0>(SUM(IAR<0>, INST<0>/8,8,8,8,8,8,8,8,8:15/));
END;
WHILE CYCLE<0>=EXECUTE&INST<0>/0:4/=PLI&
                ¬(INST<0>/5:7/='000');
 CYCLE<1>:=FETCH;
 XR<1>(INST<0>/5:7/):= INST<0>/8,8,8,8,8,8,8,8,8:15/;
END;
WHILE CYCLE<0>=EXECUTE&INST<0>/0:4/=PLI&
              INST<0>/5:7/='000';
 CYCLE<1>:=FETCH;
 ACC<1>:=INST<0>/8,8,8,8,8,8,8,8,8:15/;
END; WHILE CYCLE<0>=EXECUTE&INST<0>/0:4/=PST&
                ¬(INST<0>/5:7/='000');
 CYCLE<1>:=FETCH;
 MAIN<1>(SUM(XR<0>(INST<0>/5:7/),
       INST<0>/8,8,8,8,8,8,8,8,8:15/)):=ACC<0>;
END;
```

(c) シミュレーション結果

図-5 LCD

検証するための，シミュレーション実行コマンド群である．まず(a)を説明すると，各インストラクションは，WHILE から END 間に定義されている．例えば第23～27行目には，FETCH サイクルでの処理内容が示されている．即ち[24]は，インストラクションアドレスレジスタ IAR で示す主記憶の内容を，インストラクションレジスタ INST に転送することを，[25]は IAR の内容を1増加させることを，そして[26]はサイクルを EXECUTE モードへ切換えることを，それぞれ意味する．次に，図-5(b)を説明する．各コマンドは，RUN から ENDRUN の間に定義される．例えば，[228]～[230]は，FETCH を検証するコマンドであり，[229]は，まず時刻0における CYCLE を FETCH モードに設定し，EXECUTE モードになるまで実行することを意味する．

図-5(c)が，シミュレーション結果である．同図からわかるように，LCD のシミュレーション結果は，どのインストラクションが，いつ，どのように実行されたかのリストである．

4. 機能設計

機能設計では，方式設計で作成されたシステム構成図等を基にして，ブロック図，タイムチャート等を作成する．その際に，状態遷移図，動作フローチャートを利用するため，機能レベルにおける動作記述言語としては，状態遷移，または動作フローを基にする言語が多い．前者の代表的な言語が，DDL[6),7)]である．DDL は，機能レベルの記述を本来の目的としているが，ゲートレベル等，他のレベルの記述も可能となっている．一方，動作フローを基にする言語には，代表的なものは特にないが，一例としては IDL[8)]がある．これらの他にも種々の言語が発表されており，それぞれのシミュレータも開発されているが[9)~15)]，ここでは，文献16)に述べられている DDL シミュレータについて説明する．図-6が，シミュレータの構成図である．記述された DDL は，DDL コンパイラにより PL/I に変換される．また，初期値の設定，シミュレーション終了条件等の制御は，SCL 言語によって記述され，これも SCL コンパイラにより PL/I に変換される．DDL リンカは，DDL コンパイラ，SCL コンパイラが作成した結合情報に基づき，PL/I のメインプログラムを作成する．2つのコンパイラが作

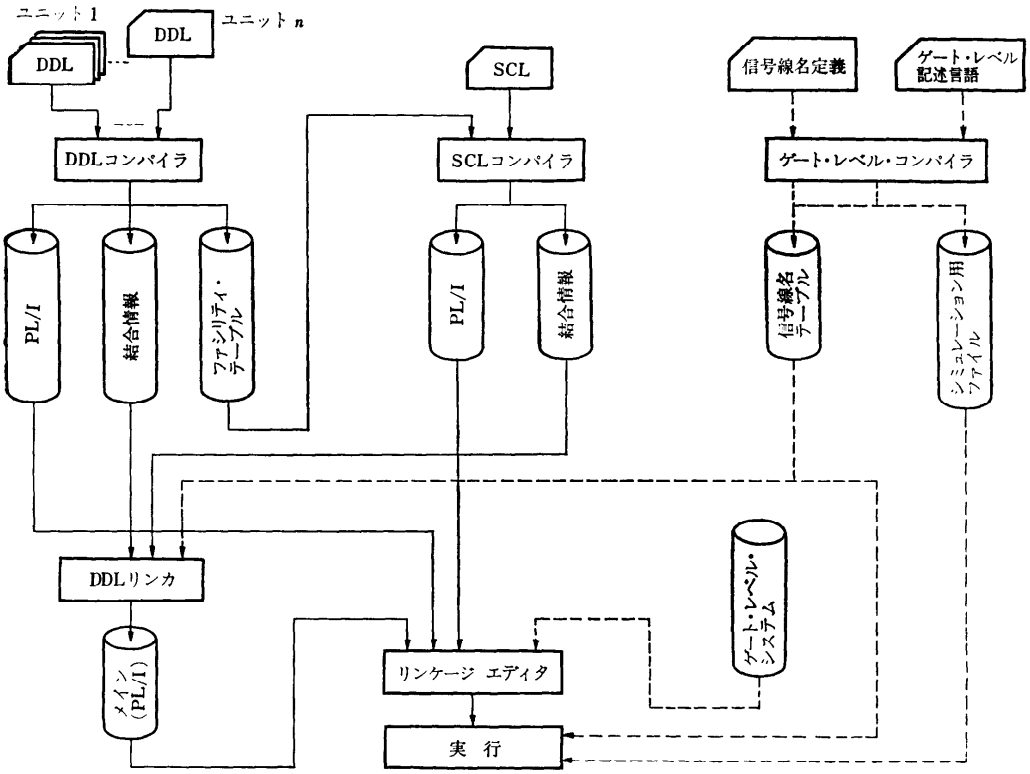
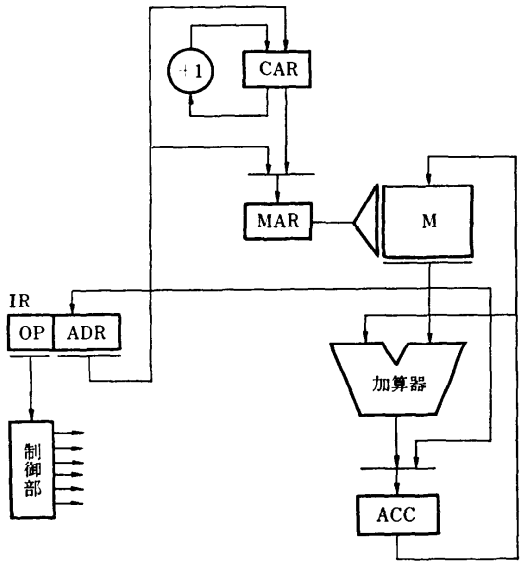


図-6 DDL シミュレータ構成図

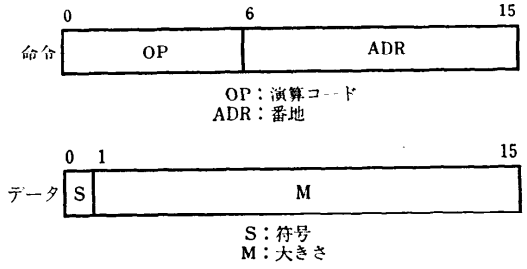
成した PL/I プログラム，及び DDL リンカが作成したメインプログラムは，通常のリンケージエディタで結合され，作成されたロードモジュールがシミュレーション実行用となる．この DDL シミュレータは，ゲートレベルシミュレータとの結合が可能である．この場合，ゲートレベルシステムと前述の PL/1 プログラムが，リンケージエディタにより1つのロードモジュールに合成される．

ここで，図-7(a)のブロック図，及び，図-7(b)の命令形式を持つ簡単な計算機を例とし，DDL 記述，SCL 記述，及びシミュレーション結果を示す．この計算機は，1ワード16ビット構成であり，命令部6ビット，アドレス部10ビットとなっている．図-7(c)に命令セット一覧を示す．図-8(a)が，この計算機の DDL 表現である．1行目はオートマトン名 CPU，クロック名 CLK の定義である．2行目から23行目に，この計算機の持つ状態と各状態における処理内容について定義している．たとえば3～4行目は，状態が ADS の場合における処理であり，レジスタ CAR の内容をレジスタ MAR へ転送すると同時に，CAR の内容を1増加させ，状態を IFT に遷移させることを意味する．また8～12行目は OP，即ちレジスタ IR の0～5ビットの内容により，それぞれ指定された状態へ遷移することを意味する．図-8(b)が，シミュレーション実行制御データ SCL 記述の例である．この例では，2～6行目が3番地の内容と4番地の内容を加算し，その結果を5番地にストアし，更に3番地と4番地に初期データとしてそれぞれ7と9を設定するプログラムを，時刻0でM(0)～M(4)に登録することを示す．また，7行目はその実行過程のトレースとして，CAR，MAR，IR，ACC，M(5)の各内容の出力を，また8行目は CAR の内容が4になればシミュレーションを停止することを，それぞれ意味する．

図-8(a)の DDL 記述と，図-8(b)の SCL 記述によるシミュレーション結果を図-8(c)に示す．(c)は，12クロック目に M(5) の内容が16進数で 0010，10進数で 16 となり，演算が正しく行われたことを示



(a) ブロック図



(b) 命令形式

略称	演算コード	演算内容
LDA	000100	ACC に M(ADR) の内容をロードする.
STA	001000	M(ADR) に ACC の内容をストアする.
ADD	010000	ACC の内容と M(ADR) の内容を加算して, ACC にしまう.
BRA	100000	無条件で ADR に分岐する.
BRP	100001	ACC の内容が正か 0 ならば, ADR に分岐する.

ACC：アキュームレータ
ADR：命令の番地指定部分の内容
M(ADR)：メモリの ADR 番地

(c) 命令セット

図-7 DDL シミュレーションモデル

している. 即ち, この命令セットに関する限り, (a) の DDL 記述は正しかったことを意味している.

このシミュレータのアルゴリズムについては, 文献16)に詳述されているので興味があれば参考にすると良い.

```
<AUTOMATON>CPU: CLK:
    <STATES>
        ADS: MAR←CAR, CAR←CAR+1,
             →IFT.
        IFT: IR←M(MAR),
             →DEC.
        DEC: MAR←ADR,
             ?OP  #4 →LDA
                  #8 →STA
                  #16→ADD
                  #32→BRA
                  #33→BRP..
        LDA: ACC←M(MAR),
             →ADS.
        STA: M(MAR)←ACC,
             →ADS.
        ADD: ACC←ACC+M(MAR),
             →ADS.
        BRA: CAR←ADR,
             →ADS.
        BRP: |*¬ACC(0)*|CAR←ADR.,
             →ADS.
    <END>.
<END>CPU.
```

(a) DDL 記述

```
SCL EXAMPLE;
    INIT AT Ø   M(Ø)=X'1ØØ3';   /*LDA  3  */
    INIT AT Ø   M(1)=X'4ØØ4';   /*ADD  4  */
    INIT AT Ø   M(2)=X'2ØØ5';   /*STA  5  */
    INIT AT Ø   M(3)=X'ØØØ7';   /*   7    */
    INIT AT Ø   M(4)=X'ØØØ9';   /*   9    */
    TRACE (CLK) CAR, MAR, IR, ACC, M(5);
    END ON CAR=X'ØØ4';
LCS;
```

(b) SCL 記述

CLK	CAR	MAR	IR	ACC	M(5)	STATE
Ø	ØØØ	ØØØØ	ØØØØ	ØØØØ	ØØØØ	/* ADS */
1	ØØ1	ØØØØ	ØØØØ	ØØØØ	ØØØØ	/* IFT */
2	ØØ1	ØØØØ	1ØØ3	ØØØØ	ØØØØ	/* DEC */
3	ØØ1	ØØØ3	1ØØ3	ØØØØ	ØØØØ	/* LDA */
4	ØØ1	ØØØ3	1ØØ3	ØØØ7	ØØØØ	/* ADS */
5	ØØ2	ØØØ1	1ØØ3	ØØØ7	ØØØØ	/* IFT */
6	ØØ2	ØØØ1	4ØØ4	ØØØ7	ØØØØ	/* DEC */
7	ØØ2	ØØØ4	4ØØ4	ØØØ7	ØØØØ	/* ADD */
8	ØØ2	ØØØ4	4ØØ4	ØØ1Ø	ØØØØ	/* ADS */
9	ØØ3	ØØØ2	4ØØ4	ØØ1Ø	ØØØØ	/* IFT */
1Ø	ØØ3	ØØØ2	2ØØ5	ØØ1Ø	ØØØØ	/* DEC */
11	ØØ3	ØØØ5	2ØØ5	ØØ1Ø	ØØØØ	/* STA */
12	ØØ3	ØØØ5	2ØØ5	ØØ1Ø	ØØ1Ø	/* ADS */
13	ØØ4	ØØØ3	2ØØ5	ØØ1Ø	ØØ1Ø	/* IFT */

AT SIM. TIME=13Ø SIMULATION IS ENDED ON CONDITION CAR=X'ØØ4'.

(c) シミュレーション結果

図-8 DDL シミュレーション

5. 論理設計

論理設計におけるシミュレーション，即ち論理シミュレーションについては，これまで数多く発表されており，その詳細な解説論文も存在する[17)~19)]．そこで，ここではストレングス，混合シミュレーションについて述べることとする．

5.1 ストレングス

論理シミュレータが扱う状態値としては，従来は1，0，X（0でも1でも良い状態），Z（ハイインピダンス）が一般的であった．これらの値は，通常のインバータ，AND，OR，フリップフロップ等をシミュレートするには十分であるが，MOS 回路でしばしば現れるワイヤード結合，トライステート出力，双方向トランスファゲート等を扱うには不十分であり，誤動作等やっかいな問題を生じていた．この主な原因は，状態値にその強さ（ストレングス）を持っていないからであり，たとえば同じ1でも出力インピダンスの低い強い1と，出力インピダンスの高い弱い1との区別ができないためである．

インピダンスの観点から考えると，従来の状態値0，1，X，Zの中では，Zのみがその概念を持ち，概念のない0，1，Xと，混在していることになる．そこで，論理値（レベルとも言う）とストレングスを明確に分離し，両者のペアーで状態値（ステート）を表すことにより，正確なシミュレーションを行うことが考え出された．

ストレングスの種類には，たとえば次のようなものがある．

（1） エクスターナル (External)

外部から，特定の端子の状態値を強制値に設定するために用いられる．

（2） フォーシング (Forcing)

端子が直接，電源あるいはグランドとつながっている状態．

（3） レジスティブ (Resistive)

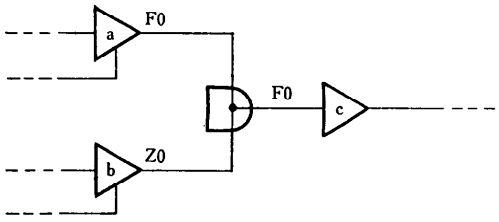
端子が，抵抗を介して電源あるいはグランドとつながっている状態．

（4） ハイインピダンス (High Impedance)

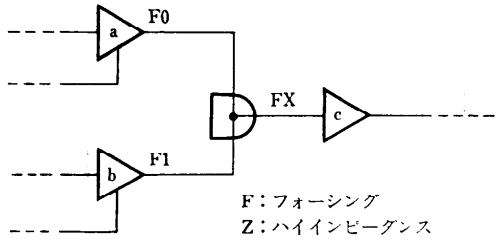
高抵抗状態．

以上の4つのストレングスは，強い順序に示されている．即ち，エクスターナルが最も強く，ハイインピダンスが最も弱い．

ストレングスは，MOS 回路のワイヤード演算時に極めて有効である．この場合，そのネットの状態値としては，そのネットに出力を供給している素子の出力状態値の中で最強のストレングスとそれに対応する論理値が採用される．ただし，最強のストレングスを持つ状態値が複数個存在し，かつ，それらに対応する論理値が異なる場合は，そのネットの論理値はXとなる．たとえば**図-9**のように，トライステートバッファa，bがワイヤード結合され，cに入力している回路を考える．この場合，a，bの出力状態値がそれぞれF0（フォーシング0），Z0（ハイインピダンス0）であれば，cの入力状態値としてはF0となる（**図-9**（a））．また，F0とF1であれば，cの入力状態値はFX（フォーシングX）となる．



（a） ストレングスが異なる場合



（b） ストレングスが同じ場合

図-9 ワイヤード結合の演算

5.2 混合シミュレーション

混合シミュレーション[20),21)]とは，機能レベルと論理レベルの混在等，異なるレベルで表現されたモデルに対し，シミュレーションを行う方式である．またそれは，トップダウン設計，LSI 設計等の際に有効となる．トップダウン設計とは，設計の上流から下流に向って，各ステップの無謬性を検証しつつ設計を進める手法である．たとえば論理設計の完了したブロックの無謬性検証のためには，ブロック単体のシミュレーションも必要であるが，他の部分とのインタフェース検証のために装置全体でのシミュレーションも必要となることがある．このような場合，そのブロックを論理レベル，その他を機能レベルでシミュレーションす

れば，すべてのブロックの論理設計完了を待つ必要もなく，かつ高速なシミュレーションが可能となる．またLSI化論理回路の検証の場合は，その部分は論理レベルで，その他の周辺回路等は機能レベルでモデルを表現することにより容易，かつ，高速に装置全体のシミュレーションが可能となる．たとえばマイクロプロセッサの論理設計の場合，マイクロプロセッサは論理レベルで，ROM，制御回路等は機能レベルで表現し，ROM中のマイクロプログラムをテストデータとして利用した例も報告されている[20]．

6. おわりに

論理装置の設計手順，方式，機能の各シミュレーション，及び論理シミュレーションとして，ストレングス，混合シミュレーションについて述べた．このようなソフトシミュレータに対し，最近出現したハードウェアシミュレータは，現在でこそゲートレベル回路を対象としたものが多いが，今後急速な改良，普及が予想され，ワークステーションの出現とともに，従来汎用の大型計算機に頼ってきた論理設計用CADに大きなインパクトを与え，その開発方向，運用形態を変化させるであろうと思われる．

参考文献

1) Bell, C. G. and Newell, A.: The PMS and ISP Descriptive Systems for Computer Structures, Spring Joint Comput. Conf., Vol. 36, pp. 351-374 (1970).
2) Macdougall, M. H.: Computer System Simulation: An Introduction, Computer Aided Design Tools for Digital Systems, 2nd Edition, pp. 117-135, IEEE Comput. Soc. (1979).
3) Efron, R. and Gordon, G.: General Purpose Digital Simulation and Examples of Its Applications, IBM Syst. J., Vol. 3, pp. 22-34 (1964).
4) Teicherow, D. and Lublin, J. F.: Computer Simulation-Discussion of the Technique and Comparison of Language, Comm. ACM, Vol. 9, No. 10, pp. 723-741 (1966).
5) Evangelisti, C. J. et al.: Designing with LCD: Language for Computer Design, 14th DA Conf., pp. 369-376 (1977).
6) Dietmeyer, D. L.: Introduction DDL, Computer Aided Design Tools for Digital Systems, 2nd Edition, pp. 55-59, IEEE Comput. Soc. (1979).
7) Duley, J. R. and Dietmeyer, D. L.: A Digital System Design Language, IEEE Trans. Comput., Vol. C-17, No. 9, pp. 850-861 (1968).
8) Maissel, L. I. and Ostapko, D. L.: Interactive Design Language: A Unified Approach to Hardware Simulation, Synthesis and Documentation, 19th DA Conf., pp. 193-201 (1982).
9) Parasch, G. J.: Development and Application of A Designer Oriented Cyclic Simulator, 13th DA Conf., pp. 48-53 (1976).
10) Druian, R. L.: Functional Model for VLSI Design, 20th DA Conf., pp. 506-514 (1983).
11) Nash, D. et al.: Functional Level Simulation at Raytheon, 17th DA Conf., pp. 634-641 (1980).
12) Armstrong, J. R. and Devlin, D. E.: GSP: A Logic Simulator for LSI, 18th DA Conf., pp. 518-524 (1981).
13) Arndt, R. L. and Dietmeyer, D. L.: DDLSIM-A Digital Design Language Simulator, Proc. of NEC, Vol. 26, pp. 116-118 (1970).
14) Hollander, Y.: Using an RTL Simulator to Simplify VLSI Design, VLSI DESIGN, Sep., pp. 60-66 (1983).
15) Chappell, S. G. et al.: Function Simulation in the Lamp System, 13th DA Conf., pp. 42-47 (1976).
16) 上原貴夫他：大型コンピュータなどの論理設計に適用できるCADシステム，日経エレクトロニクス，1979. 11. 12, pp. 104-130 (1979).
17) 村井真一：ゲートレベル論理シミュレーション，情報処理学会論文誌，Vol. 22, No. 8, pp. 762-769 (1981).
18) 菅野卓雄監修：カスタムLSI応用設計ハンドブック，リアライズ社，pp. 242-257 (1984).
19) Szygenda, S. A. and Thompson, E. W.: Digital Logic Simulation in a Time-Based, Table-Driven Environment Part 1. Design Verification, Computer, pp. 24-36 (Mar. 1975).
20) 白木 昇他：BINALY：階層構造モデルによる論理シミュレータ，沖電気研究開発110, Vol. 47, No. 1, pp. 33-38 (1980).
21) Sasaki, T. et al.: MIXS: A Mixed Level Simulator for Large Digital System Logic Verification, 17th DA Conf., pp. 626-633 (1980).

(昭和59年7月5日受付)