設 計 自 動 化 56－1
（1991. 2. 12）

CAD統合化のためのプロセス・スケジューリング手法とインタフェース構築ツール

宮崎敏明　星野民夫　遠藤真
NTT LSI 研究所

本論文は、著者らが開発したＣＡＤ統合化システムと、そのサブシステムであるＣＡＤツール・スケジューラおよびインタフェース・ビルダについて述べている。本システムの特徴は、独自のCADツール・スケジューリング手法にある。一度、設計者が専用の「設計フローチャート・エディタ」で、設計手順を定義すると、それぞれのツールが生成する設計データの更新を効率よく行なうように必要最小限のCADツールを自動起動する。また、本システムは、新たなCADツールを取り込むために、いくつかのユーザ・インタフェース構築用ユーティリティとサブルーチン・パッケージを持っている。

A Process Scheduling Technique and an Interface Builder for CAD Tool Integration

Toshiaki Miyazaki (Tanaka)*　Tamio Hoshino　Makoto Endo

NTT LSI Laboratories

3-1, Morinosato Wakamiya, Atsugi-Shi Kanagawa Pref., 243-01, JAPAN

This paper describes an original CAD framework system, and a CAD tool scheduler and an interface builder which are the subsystem of the framework. The main advantage of this system is its unique process control mechanism. Once VLSI designers specify their tasks by using a *flow-chart based graphic editor,* the system could handle the design process and maintain the design data produced by each tool. In addition, this system has several human-interface building tools and subroutine packages that take a new design tool into the system environment easily.

1 はじめに

複雑化するＶＬＳＩの設計を助けるために多くのＣＡＤツールが開発されてきた。今日では、個々のＣＡＤツールをいかにうまく使いこなすかが、設計時間の短縮に大きくかかわってきている。一般に、多くの優れたＣＡＤツールは個別に作られており、他のツールとデータを共有するといった協調性は犠牲にされてきた。

これらを解決するために、フレームワークと呼ばれる統合化システムが最近いくつか発表されている [1]-[6]。しかし、それらシステムは、個々のＣＡＤツールを包含 (encapslate) する機能と、統一したユーザ・インタフェースを提供するものとして開発されたものが多く、いくつかの問題を抱えている。例えば、２つのＣＡＤツールを接続するには、専用のコンバータを作成するか、フレームワークの用意したデータベースに合うように各々のＣＡＤツールを変更しなければならない。一方、設計者は、つねに最新のＣＡＤツールを望むため、フレームワークを使用してもＣＡＤツールを統合する者の仕事は減っていない。

設計者にとってもＣＡＤツール開発者にとっても良いことは、各ＣＡＤツール間のインタフェースを標準化することである。実際、ＣＡＤフレームワークに関する標準化は現在動きだしている [7]。しかし、全てのＣＡＤツールがコンバータなしで互いの設計データを共有するようになるためには、まだ長い時間が必要である。

以上の理由から、我々は CONDUCTOR とよぶ新しいフレームワークを独自に開発した。本論文では、関連研究を述べたのち、CONDUCTOR の主な機能について詳しく述べる。

2 関連研究

CAD フレームワークの研究は近年盛んに行なわれるようになった。その中でも OCT システムは、最初のシステムであると考えられる [1]。OCT は、設計データ管理機能と VEM とよばれるグラフィック・インタフェースをはじめ、多くのデータ・アクセスのためのユーティリティを備えており、VLSI 設計者と CAD ツール開発者に快適な環境を与えている。しかし、OCT は、CAD プロセス制御のための機構は持ち合わせていない。

カーネギー・メロン大学で開発された Ulysses[2] と Cadweld[3] は、VLSI 設計工程を定義するスクリプト言語を持っており、そのスクリプトは、黒板モデルを用いたタスク・コントローラ内で使用される。２つのシステムは、CAD ツール間の複雑な設計データのやり取りを隠し、ユーザ・インタフェースの統一化に成功している。また、スクリプト言語を使用して、新しい CAD ツールを容易に取り込むことができる。しかし、プロセス制御が内部で完全に自動的に行なわれるため、設計者は時として次に起動される CAD ツールを認識できないことがある。

さらに、最近では商品化された CAD フレームワークがいくつか発表されている。CADENCE フレームワーク・システム [6] は、CAD ツールを包含するために *SKILL* とよぶ強力なスクリプト言語を持っている。DEC 社の Powerframe[4] は、Cadweld と同様にオブジェクト指向パラダイムを採用している。両者は、統一したユーザ・インタフェースを実現しているが、ユーザが独自のスクリプト言語で設計工程を書かねばならない。TekWave[5] は、アイコニック・プログラミング環境を提供している。それは、グラフィック画面上で各 CAD ツールを表すアイコンを接続することで CAD ツールの起動順序を定義させようとするものである。しかし、残念なことにプロセス制御自体は単純なものである。

MAKE コマンド [8] は、UNIX の世界ではソフトウエア開発で頻繁に使用されている。*MAKE* コマンドは、Makefile 内に書かれた各ファイル間の依存関係とその依存関係を保つためのコマンド情報をもとに、各ファイルのタイム・スタンプ (更新された日時) をチェックし、矛盾が無いようにコンパイラ等のコマンドを自動起動する。*MAKE* コマンドは、大規模なプログラム開発時などでは大変強力なツールであるが、残念なことに単純な UNIX のファイルしかあつかうことができない。

一方、ソフトウエア開発の立場から CAD ツールをみると、インタラクティブなツールが多くなるにつれて、「ソフトウエアの爆発」が問題になってきている。実際、インタラクティブなアプリケーション・プログラムの５０％以上は、ユーザ・インタフェース部であるといわれている [9]。「ソフトウエアの爆発」を避けるために、いくつかのインタフェース構築用ユーティリティが考えられている [9][10]。それらのユーティリティは、ユーザ・インタフェースのためのソース・コード量を削減するために役立っている。しかし、それらユーティリティを CAD ツール統合化のために使用しようとすると、各々独自の構造や制約を持っているために、ほとんど全ての CAD ツールに手をいれなければならなくなるのが現状である。

以上、述べてきた関連研究を考慮して、CONDUCTOR は以下の機能を持つようにした。

- VLSI 設計者が、特別な言語を用いずに、設計作業手順を定義できる設計フローチャート・エディタ。本エディタは、作業手順を入力するためだけでなく、実際に起動されている CAD ツールを点滅させて示すことができ、設計者は設計作業を目で容易に確認できる。
- 不必要な CAD ツールの起動を避ける機能。本機能は、*MAKE* コマンドと同様な機能であるが、単純ファイルだけでなくバージョンや設計レベル等の管理情報を持った設計データもあつかうことができる。
- 簡単に使用できるように設計されたユーザ・インタフェース構築用ツールとサブルーチン・パッケージ。

3 CAD ツール統合環境

3.1 システム・アーキテクチャ

CONDACTOR のシステム構成図を図 1 に示す。本システムは、メイン・パネル (Main Panel: MP)、CAD ツール・スケジューラ (CAD Tool Scheduler: CTS) および設計データ管理ツール (Design Data Maintainer: DDM) の三つのサブシステムからなる。MP は、トップ・レベルのユーザ・インタフェースを提供するものであり、アイコン・ボタンやポップアップ・メニュ等を持っている。VLSI 設計者は、各 CAD ツールを MP 上のボタンやポップアップ・メニュを使用して起動できる。MP は、CAD ツールが作り出す設計データを格納する作業ディレクトリも管理できる。これにより、設計

者は過去に作成した設計データを見失うことを避けることができる。CTS は、VLSI 設計者によって定義された設計手順に従って各 CAD ツールを自動起動する。そのため、設計者は複雑な各 CAD ツールの起動手順を覚えることから解放される。DDM は、設計データの管理を行なう。設計者は、DDM の操作パネルを使用して設計データを単純ファイルと同じようにあつかうことができる。全ての CAD ツールおよびサブシステムは、MP から起動することができる。

新たな CAD ツールを CONDUCTOR 内に取り込むには、簡単な定義を「環境ファイル」に書き込むだけでよい。CONDUCTOR は、起動されると「環境ファイル」を読み込み、取り込むべき CAD ツールを認識する。

また、CONDUCTOR は、新たな CAD ツールを包含するためのいくつかのユーティリティを持っている。汎用リスト・ブラウザ (List Browser: LB) は、各ツールが本来コンソール・ターミナルに出力する情報を取得するために用いられる。グラフィックを用いたマクロ・プロセッサ (WINC) は、種々の入力パラメータを必要とする CAD ツールに適用される。また、新たに CAD ツールを開発するときに統一したユーザ・インタフェースを簡便に作成できるように、グラフィック・パネル作成ツール (DF) および設計データ・アクセス・パッケージが用意されている。これら全てのユーティリティおよびパッケージを総合してインタフェース・ビルダ (Interface Bulider) と呼ぶ。

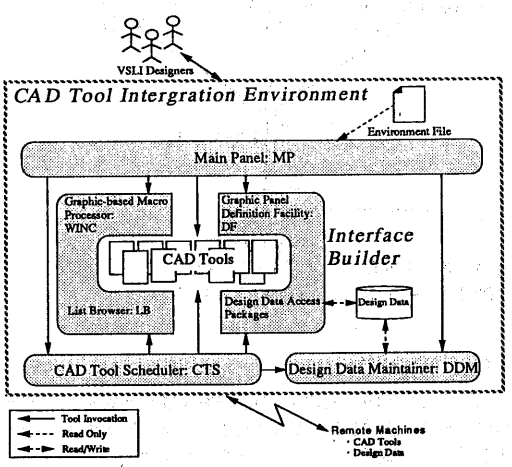


図 1: CONDUCTOR のシステム構成図。

3.2 CAD ツール・スケジューラ: CTS

一般に、VLSI 設計作業は多くの CAD ツールを使用して行なわれる。しかし、現状のフレームワークでは、各 CAD ツールの協調作業をを設計者に馴染みのない個々のフレームワークが用意した独自言語を用いて記述しなければならない。そのため、既存フレームワーク上で設計工程や CAD ツールの起動順序を定義することは設計者にとって、それほど簡単な作業ではない。CAD ツール・スケジューラ (CTS) は、上記の作業を軽減するために開発された。CTS は、CONDUCTOR のサブシステムであり、VLSI 設計者によって定義された CAD ツールと設計データの関係をもとに各ツールを起動する。設計者は、CTS のユーザ・インタフェースである設計フローチャート・エディタを用いて、ツールやデータの関係を定義、変更できる。最初に、CTS は、定義された設計フローチャートから CAD ツールや設計データの関係を抽出し、タスク依存グラフ (Task Dependency Graph: TDG) を作る。つぎに、CTS は、TDG を参照し、保守が必要な設計データを見つけ適切な CAD ツールを起動する。そのため、たとえ設計フローチャート上で使用されるように定義された CAD ツールであっても、設計データの更新に寄与しないものは起動されない。この機構は、設計データを再作成する時間を短縮するのに役立っている。設計フローチャートは、もちろん、セーブして複数の設計者間で共有することができ、設計工程管理が可能となる。CTS は、ある CAD ツール起動中、設計フローチャート内の関係するツール・アイコンを点滅させる。それにより、設計者は、現在の CAD ツール起動の状況を容易に把握できる。

3.2.1 設計フローチャート・エディタとタスク依存グラフ

図 2に、CTS の画面イメージを示す。VLSI 設計者は、起動する CAD ツールの順序とその入出力データを CTS を用いて設定する。ツール、データおよびテキスト・ファイルは、各々「矩形」、「ディスク」および「カード」アイコンで示される。個々のツールやデータの依存関係は、設計フローチャート・エディタ内で関係するアイコン間を結線することによって確立される。各ディスク・アイコンは、一つ以上の「ハードウエア・モジュール」を示している。例えば、数百個のエンド・レベル・セルを含むセル・ライブラリは、一個のディスク・アイコンで示すことができる。全てのアイコンは、パラメータ設定ウインドウを持っており、ツールを起動する前ならば、いつでもパラメータを変更することができる。設定できるパラメータは、アイコンによって異なる。例えば、一般的に矩形アイコンすなわち CAD ツールには、処理対象の「ハードウエア・モジュール」名とツールに対するオプションが設定できる。本機能は、単純ではあるが非常に大切である。なぜならば、フレームワーク・システムにおいて、包含されている CAD ツールに対してユーザがオプション等のパラメータが自由に設定できないと、設計者は不満を持つからである。設定されたパラメータの一つは、各対応するアイコンの左上に表示される。

CTS の機能を詳しく述べるために、ここで以下の設計作業を仮定する。「モジュール M はサブモジュール A と B を持っている。設計作業のゴールは、論理シミュレータとレイアウト・ツールのために階層展開された論理レベルの設計データを作成することである。ここで、モジュール M は RTL 記述されており、サブモジュール A および B は論理レベルの記述 (ネットリスト) が用意されているとする。」このとき、以下に示す手順でゴールに達するものとすると設計フローチャートは図 2に示したようになる。

- ツール HSLFX は、モジュール M の RTL 記述をコンパイルし、出力を設計データ bobj01 に格納する。
- ツール ANGEL は、bobj01 を用いて、モジュール M の論理レベルの設計データを合成し、結果を設計データ sobj01 に格納する。

- ツール HSL はモジュール A と B の論理レベルの記述をコンパイルし、結果を設計データ lib に格納する。
- ツール EXPANDH は、sobj01 と lib を入力として、モジュール M を階層展開し、展開結果を設計データ sobj04 に格納する。

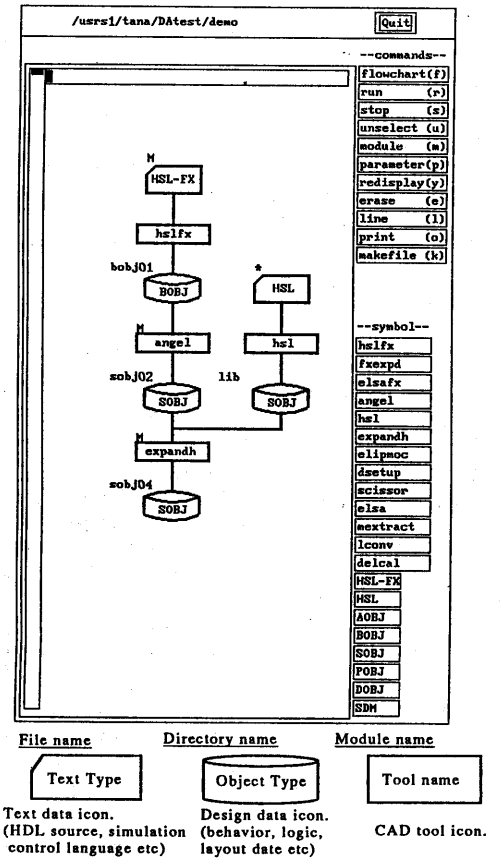


図 2: CAD ツール・スケジューラ (CTS) の画面イメージ。

設計者は、このような設計フローチャートを、通常の描画ツールを使うような感覚で作成することができる。フローチャート・エディタは、例えば「2つの設計データは直接接続することはできない」といった CAD ツールと設計データに対する情報を持っている。それにより、CTS は、明らかに不当な設計手順の設定を、設計フローチャート作成時にチェックできる。作成された設計フローチャートは、ドキュメントとして PostScript†プリンタに出力することもできる。

設計者が、設計フローチャート・エディタ内の 'run' ボタンをクリックすると、CTS は描かれているフローチャートからツールやデータの依存関係を抽出し、タスク依存グラフ TDG を作成する。TDG は、有向グラフであり、ノードは、CAD ツールまたは設計データを示し、エッジは、設計データの流れを示している。図 3に TDG の例を示す。TDG 内で一番下の設計データをターゲット・データ (TD) と定義する。TD は、一つの TDG 内では唯一である。ここで、TDG 内に含まれる全ての設計データの集合を D とすると、レベル $Level(d)$ $\forall d \in D$ は以下のように定義される。

$$
\begin{aligned}
Distance(d) &\equiv d \text{ と } TD \text{ の間にある CAD ツールの数。} \\
Level(TD) &\equiv L \equiv \max_{\forall d \in D}(Distance(d)). \\
Level(d) &\equiv L - Distance(d).
\end{aligned}
$$

例えば、図 3で、$Level(D_{11}) = 1$, and $Level(D_{41}) = Level(TD) = 4$. である。

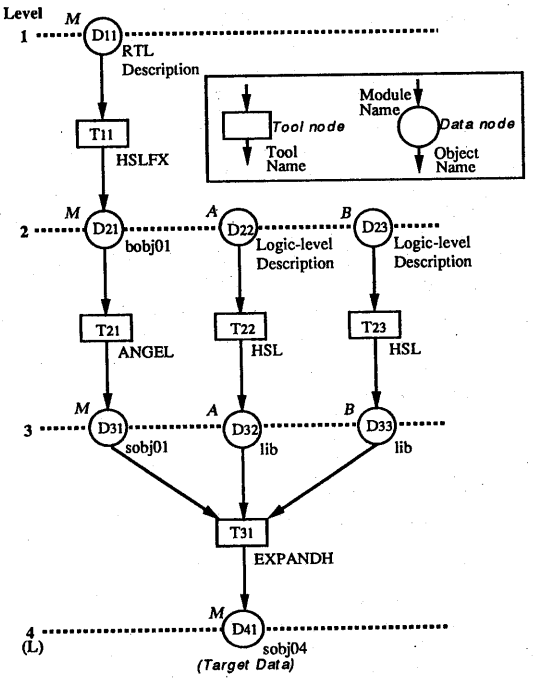


図 3: タスク依存グラフ (TDG) の例。このグラフは、図 2 の設計フローチャートから作成される。

3.2.2 ジョブ制御

CTS は、TDG を用いて設計作業手順を制御する。制御のゴールは、矛盾がないように設計データを作成することである。ジョブ制御アルゴリズムを以下に示す。

for $l := 1$ **to** $L - 1$ **do**
  for *each data* : n *at* l **do**
    if $time\_stamp(D_{ln}) \geq time\_stamp(child(D_{ln}))$ **then**
      invoke $CAD\_tool(D_{ln})$;

where

$L \equiv Level(TD)$.
$time\_stamp(D_{ln}) \equiv D_{ln}$ のタイム・スタンプを得る。大きいほど、新しいことを示す。
$CAD\_tool(D_{ln}) \equiv D_{ln}$ を入力データとする CAD ツール。
$child(D_{ln}) \equiv CAD\_tool(D_{ln})$ の出力データ。

□

†PostScript は、Adobe System Inc. の登録商標である。

まず、CTS はレベル 1 から設計データのタイム・スタンプ (作成更新日時) を調べる。次に、入力データがその出力データよりも新しい CAD ツールを見つけ、そのツールを起動する。以上の手順をレベル L まで続ける。例えば、図 3 において、D_{11} と D_{23} が各々独立に更新されたと仮定すると、CTS はデータ D_{21}、D_{31}、D_{33}、D_{41} を更新するためにツール T_{11}、T_{21}、T_{23} および T_{31} を順番に起動する。D_{22} は、更新されなかったので、T_{22} は、起動されない。

ここで、前述のジョブ制御アルゴリズムよると、D_{31} と D_{33} が更新されているため、D_{41} を更新しようとして T_{31} は二度起動されるように思える。しかし、実際の起動は一回しか行なわれない。それは、D_{41} と D_{31} の間のタイム・スタンプの矛盾を解決するために T_{31} を起動することで D_{41} は一回更新されるが、更新されたのちは D_{41} と D_{33} のタイム・スタンプの矛盾は既に解決されているからである。これが、設計 TAT を短縮するためのキー・メカニズムである。

CTS は、起動しているツールと関連するアイコンをそのツールの処理が終るまで点滅させ、終了後は各ツールからのメッセージを表示するウインドウを開く。

3.3 設計データ管理ツール： DDM

我々は、3.4 節で示すように特別なデータ管理機構を開発し使用している。個々の設計データは、データベースとしても使用できるように多くの管理項目を持っている。一方、HDL 記述のように単体のファイルも設計環境に存在する。そのために、効率良く設計データやファイルを管理するためのツールやコマンドが必要になった。DDM は、グラフィック画面を用いて、データを統一的に管理するツールである。DDM 上では、設計データも単純ファイルもほとんど区別なくあつかうことができる。

DDM は、もちろん単体で使用できるツールであるが、前記の CTS 上からも起動でき、選択した設計データのブラウザとしても使用できる。

3.4 設計データ構造とアクセス・パッケージ

CONDUCTOR は、色々な設計レベルをサポートする CAD ツールを統合化しなければならない。それらのツールは、一般に階層化設計とデータのバージョン管理機能を持っている。そのために、CONDUCTOR 内で使用する統一した設計データ構造は、高速アクセスばかりでなく、拡張性を十分考慮する必要があった。

CONDUCTOR 内の設計データは、管理テーブル (MNGTBL) と実データから構成されている。個々の実データは、階層設計されたハードウエア・モジュールの情報と２０を越える管理項目からなり、MNGTBL からポインティングされている。一方、MNGTBL は、個々の実データが持つ管理項目を重複して持っている。これにより、実際に実データの参照や更新が起こるまで、実データをアクセスする必要がなくなるため高速化に寄与している。また、計算機上のディスク容量を節約するために、各々のデータは書き込みの際に自動的にパッキングされる。

それぞれの実データは、一定の大きさのテーブルからなり、管理項目を格納したテーブル以外は、任意の情報を格納できる。そのため、新たな情報を格納したくなった場合は、新しいテーブルを一つ設けるだけでよい。もちろん、その場合、新たに作成したテーブルおよびそこに格納した情報は作成したものが責任を持って管理しなければならない。

設計データ・アクセス用パッケージは、用途に合わせて階層的に用意されている。最も基本的なパッケージは、テーブルを意識してデータをアクセスするようなっている。しかし、それではアプリケーション層からあつかいにくいので、テーブル構造を全く意識しないアクセス・パッケージも個別の設計データに合わせて用意されている。

4 インタフェース構築ツール

4.1 グラフィック・パネル作成ツール： DF

アプリケーション・プログラムでは、頻繁にユーザ・インタフェース部の改善が行なわれる。そのため、グラフィック・パネルを独立して設計できるように、ユーザ・インタフェースに関わる部分をアプリケーション本体とを切り離すことにした。

グラフィック・パネル作成ツール (DF) は、パネル上のクリック・ボタンやポップアップ・メニュといったファシリティの位置や項目などの属性を、本体のアプリケーション・プログラムを再コンパイルすることなしに変更することができるパッケージである。DF は、パネル定義言語 (DF 言語) と、そのパネル定義と本体のアプリケーション・プログラムを結び付けるために必要なサブルーチン・ライブラリから構成される。DF 言語記述は、ファシリティの定義とそのファシリティに結合されるアクション名のみであり、実際の動作に関するものは一切含まない。また、DF 言語は、マルチ・パネル定義機能を有しており、一つの DF 言語記述ファイル内に複数の違ったパネルに対する定義が可能である。

図 4 に例を示す。図 4(a) は、DF 言語記述の一部である。この記述は、パネル上のクリック・ボタンやテキスト入力フィールド等のファシリティとキャンバス・ウインドウ上のポップアップ・メニュを定義している。図 4(b) は、C 言語で書かれたメイン・プログラムの概要である。(a) および (b) を使用して作成されたウインドウを図 4(c) に示す。この図は、図 4(a) の２５ - ５２行目で定義されたポップアップ・メニュが表示されているところである。DF は、グラフィック・パネルのカスタマイズやデッバグに特に便利であり、既に１０以上のグラフィック・インタフェースを持つアプリケーション・プログラムの開発に利用されている。実際、全ての CONDICTOR のサブシステムは、DF を用いて作成されている。

4.2 グラフィックを用いたマクロ・プロセッサ： WINC

フレームワークでは、CAD ツールを順に起動したりするためにシェル・スクリプトやコマンド・ファイルがよく作成される。それらのファイルは、いくつかの変数やパラメータを除いてテンプレート化できることが多い。WINC は、形式がある程度決まったファイルを作成するために開発されたマクロ・プロセッサである。WINC の構造概要を図 5 に示す。WINC は、'cpp'[11] や 'm4' [12] のような通常のマクロ・プロセッサと同等な機能を持っているだけでなく、グラフック・パネルからパラメータを入力できる機能を持っている。WINC は、起動されるとまず「テンプレート・ファイル」を読み、その情報に従ってパラメータ設定ウインドウを開く。

ユーザがウインドウを通してパラメータを設定したのち、'do' ボタンをクリックすると、WINC は、テンプレート・ファイル内に記述された内容をマクロ展開し、結果をファイルに出力する。

テンプレート・ファイルは、パネル定義部と被マクロ展開部からなる。WINC は、パネル定義部の記述に従ってパラメータ設定ウインドウを作成し、画面上にマップする。被マクロ展開部は、WINC が処理するマクロ変数を含んだテキスト群からなる。WINC は、外部のファイルやコマンド実行結果を展開結果に取り込むこともできる。

図 6(a) にテンプレート・ファイルの例を示す。%%(7 行目) で区切られた上部はパネル定義部であり、下部は被マクロ展開部である。図 6(b) は、1 - 7 行目の記述によって WINC が作成したパラメータ設定ウインドウである。'do' ボタンがクリックされると、WINC は、図 6(c) に示すリストを生成する。

WINC は、汎用波形エディタから論理シミュレータを起動するための制御ファイルの作成等に適用されている。

4.3 汎用リスト・ブラウザ: LB

フレームワークを意識せずに作成された CAD ツールの多くは、ターミナル上に種々の情報を表示する。このような情報は、フレームワーク上に CAD ツールが取り込まれると他の情報と混ざったり失われたりすることがある。汎用リスト・ブラウザ (LB) は、重要な情報が失われることを防ぐために開発された。LB は、本来コンソール・ターミナルに出力される情報を切替えて、LB のウインドウ上に表示させる機能を持っている。また、表示内容はファイルにセーブすることも可能である。

5 実現方法

CONDUCTOR フレームワークは、UNIX ワークステーション上に実現されている。いくつかの設計データ・アクセス・パッケージは、メイン・フレームでも使用できる。移植性を考慮し、グラフィックは X window を用い、プログラミング言語は C 言語を用いた。

6 まとめ

CONDUCTOR とよぶ新し CAD フレームワーク・システムを紹介した。その中で、設計データを作成するために状況に合わせて必要最小限の CAD ツールしか起動しない機構を実現した。この機構は、設計作業の TAT 化に効果がある。また、設計者が設計データ作成手順を特別な言語を覚えることなく定義できるように専用のグラフィック・エディタを用意した。

CONDUCTOR は、新たな CAD ツールを取り込むためにいくつかのユーザ・インタフェースに関するユーティリティおよびパッケージを持っている。

CONDUCTOR は、専用グラフィック・エディタで定義された設計データ作成手順をを共有することで設計プロジェクト管理ができる。しかし、より大きなプロジェクト管理を実現するには、「グループウエア」の概念を取り入れる必要があると考える。

謝辞

日頃、有益な助言を頂く LSI 研究所 設計システム研究部 安達徹主幹研究員に感謝します。

参考文献

[1] D. Harrison et al., "Data Management and Graphics Editing in the Berkeley Design Environment," Proc. ICCAD Conference, pp.240-27, Santa Clara, CA, November 1986.

[2] M. Bushnell and S. W. Director, "Automated Design Tool Execution in the Ulysses Design Environment," IEEE Trans. on Computer-Aided Design, Vol.8, No.3, March 1989.

[3] J. Daniell and S. W. Director, "An Object Oriented Approach to CAD Tool Control Within a Design Framework," Proc. 26th Design Automation Conference, pp.197-202, Las Vegas, NV, June 1989.

[4] "Powerframe manual," Digital Equipment Corp.

[5] "TekWave manual," Tektronics Corp.

[6] "Design Framework manual," Cadence Design Systems Inc.

[7] "CFI Newsline," CAD Framework Initiative, Inc. Vol.1, Issue 3, November 1989.

[8] S. I. Feldman, "MAKE - A Program for Maintaining Computer Programs," Software Practice and Experience, Vol.9, No.4, pp.255-265, April 1979.

[9] M. R. Szczur and P. Miller, "Transportable Application Environment (TAE) PLUS - Experiences in 'Object'ively Modernizing a User Interface Environment," Proc. OOPSLA Conference, pp.58-71, September 1988.

[10] A. Weinand, E. Gamma and R. Marty, "ET++ - An Object-Oriented Application Framework in C++," Proc. OOPSLA Conference, pp.46-57, September 1988.

[11] "CPP - C Pre-Processor," UNIX manual.

[12] "M4 - a Macro Processor," UNIX manual.

```
   /* Definition for the facilities on the upper panel window.     */
1  window: sample1                    /* First DF definition        */
2  panelfile:                         /* Start panel definition.    */
3  cmap "paneseg" { red   255,080;    /* Color map for the panel.   */
4                   green 255,100;
5                   blue  150,020;
6  }
7  button {                           /* Define a click button.     */
8    string "button";                 /* Button name.               */
9    xcod   10; ycod   10;            /* XY coordination.           */
10   proc   myproc1;                  /* Activated function name.   */
11   menu   buttonmenu;               /* Bind pull-down menu.       */
12 }
13 menu buttonmenu {                 /* Define a pull-down menu.   */
14   item { string "btn menu 1";      /* Menu item &                */
15          proc myproc1;      }      /* activated function name.   */
16   item { string "btn menu 2";
17          proc myproc2;      }
18 }
19 field  {                          /* Define a text field.       */
20   string "text";                   /* Title name.                */
21   xcod   80; ycod   10;            /* XY coordination.           */
22   length 15;                       /* Max text length.           */
23   proc   myproc2;                  /* Activated function name.   */
24 }
```

```
   /* Definition for the pop-up menu on the lower canvas window.   */
25 window: sample2                    /* Second DF definition.      */
26 menufile:                          /* Start menu definition.     */
27 cmap "canseg" { red   200,000;     /* Color map for the canvas.  */
28                  green 255,000;
29                  blue  255,000;
30 }
31 command "a" myproc1;               /* Define fast paths.         */
32 command "b" myproc2;               /* When type the character    */
33 command "e" myexit;                /* keys, call the function.   */
34 command "q" myexit;                /* As to select a menu item.  */
35 menu main {                        /* Define a pop-up menu.      */
36   title "   canvas menu     ";     /* Menu title.                */
37   item { string "canvas menu EXIT"; /* Menu item &                */
38          proc  myexit;          }  /* activated function name.   */
39   item { string "canvas menu 1   ";
40          proc  myproc1;         }
41   item { string "canvas menu 2   ";
42          proc  myproc2;
43          pullright plmenu;      } /* Call a pull-right menu.    */
44 }
45 menu plmenu {                      /* Define a pull-right menu.  */
46   item { string "pull right EXIT";
47          proc myexit;           }
48   item { string "pull right 1";
49          proc myproc1;          }
50   item { string "pull right 2";
51          proc myproc2;          }
52 }
```

図 4 (a): DF 言語記述例。(一部省略してある)

```
1 #include<Xdf_map.h>

2 void proc1(), proc2(), proc3()
3 DF_RTN rtntb[]={                  /* Define the table of activated   */
4                  proc1, "myproc1",  /* functions for DF.              */
5                  proc2, "myproc2",  /* Pairs of 'function name' and   */
6                  proc3, "myexit" ,  /* 'proc" name in DF file'.       */
7                      0,        0};

8 main() {
9   df_file("sample.df");          /* Read a DF file and initialize.   */

10  /* Initialize X-window. */

11  panel=XtCreateWidget("Panel", ...);  /* Create a panel window.     */
12  df_panel(panel, "sample1");          /* Bind DF facilities to the panel. */

13  canvas=XtCreateWidget("Canvas", ...); /* Create a canvas window.   */
14  df_menu(canvas, "sample2");          /* Bind DF facilities to the canvas. */

15  XtMainLoop();                        /* Start event dispatching.  */
16 }

17 /* Define activated functions. */
18 void proc1() {...}
19 void proc2() {...}
20 void proc3() {...; exit(0);}
```

図 4 (b): (a) の DF 言語記述を用いるときの C 言語プログラムの概要。

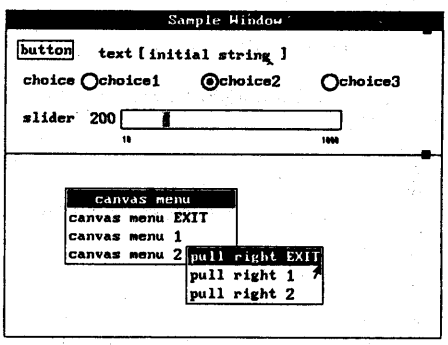


図 4 (c): (a) と (b) から生成されたウインドウ画面。

図 4: グラフィック・パネル作成ツール (DF) の例。

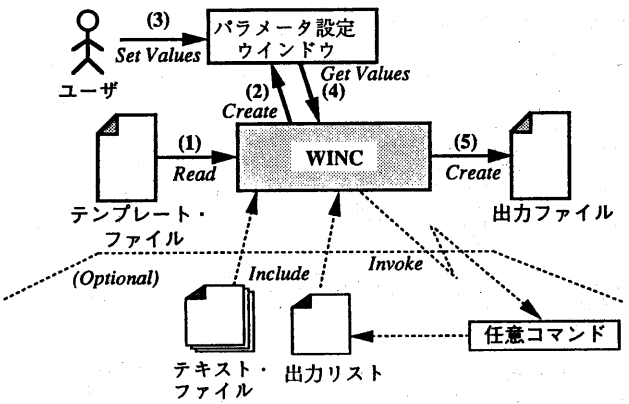


図 5: WINC マクロ・プロセッサの構造概要。

```
000001  %SET    Command1 = 'who';
000002  %SET    Command2 = 'ls';
000003  %SWITCH Switch ;
000004  %CYCLE   Cycle   ('C1','C2','C3','C4');
000005  %CHOICE  Choice  ('CH1','CH2','CH3','CH4');
000006  %TOGGLE  Toggle  ('TG1','TG2','TG3','TG4','TG5'/' ');
000007  %%
000008
000009  ***** Selection is as follows *****
000010  Command1 ·      :%Command1
000011  Command2        :%Command2
000012  Switch          :%Switch
000013  Cycle           :%Cycle
000014  Choice          :%Choice
000015  Toggle          :%Toggle
000016  **********************************
000017  %IF ( Switch )
000018  ----- Output from "%Command1" -----
000019  %EXEC(%Command1)
000020  %ELSE
000021  ----- Output from "%Command2" -----
000022  %EXEC (%Command2)
000023  %ENDIF
000024
000025  %PSET Tg ('Start '+%Toggle+' End')
000026  %FOREACH X ( %Tg )
000027  TG contains "%X".
000028  %ENDFOR
```

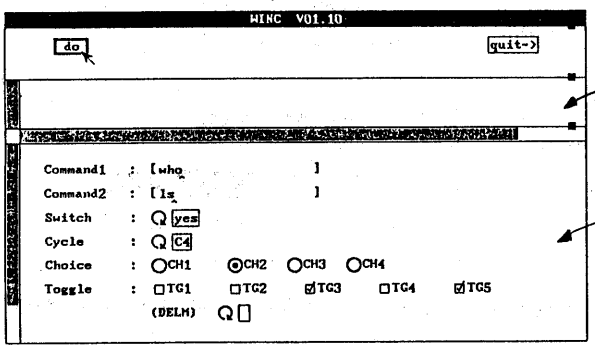
Default values.

Panel definition part.

Expansion part.

Concatenate text strings.

図6 (a): WINCテンプレート・ファイルの例。



Error message area.

Parameter setting area
(Produced by the panel definition part of the template.)

図6 (b): パラメータ設定ウインドウの例。

```
000001
000002  ***** Selection is as follows *****
000003  Command1        :who
000004  Command2        :ls
000005  Switch          :YES
000006  Cycle           :C4
000007  Choice          :CH2
000008  Toggle          :TG3 TG5
000009  **********************************
000010  ----- Output from "who" -----
000011  tana       console      Mar 20 09:10
000012  tana       pts1         Mar 20 09:10
000013  tana       pts3         Mar 20 09:29
000014
000015  TG contains "Start".
000016  TG contains "TG3".
000017  TG contains "TG5".
000018  TG contains "End".
```

From 'FOREACH' statement.

図6 (c): 出力リストの例。(a) のテンプレートを用い、(b) の設定のときに出力される。

図6 : WINCのテンプレート、パラメータ設定ウインドウおよび出力リストの例。