設計自動化 41－2
（1988. 2. 18）

図形的ハードウェア記述言語を用いたシリコンコンパイラの設計環境

芳野泰成、奥澤　治、服部一彦、小田原豪太郎
東京大学　工学部

本稿では、ＡＳＩＣ設計を対象としたシリコンコンパイラにおける論理生成部について、設計言語、生成システム、およびその設計環境について述べる。
ＡＤＬ（Algorithm Description Language）は、図形的な表現を用いたアルゴリズムレベルのハードウェア記述言語であり、ハードウェアの知識がない技術者にも設計ができるように開発されたものである。論理生成システムＬＯＳ（ＬOgic generation System）は、ＡＤＬを設計入力言語とし、まずＡＤＬを状態遷移表現であるＳＦＤＬ（Symbolic Functional Description Language）に変換し、これより制御論理を生成する。また、ＡＤＬより、クリーク分割を用いて、資源の割りつけを行ない、データパスを生成する。また、これらのための設計環境として、手書きによる対話的なＡＤＬ入力システムである、論理情報エディタも開発した。入力ツールとして、液晶ディジタイザを用いることによって、思考を妨げられることなく、設計効率が向上する。
本稿では、上記３つの項目について、その特徴、設計例、問題点等について述べる。

A Design Environment of Silicon Compilation Using Graphical Hardware Description Language

Taisei YOSHINO, Osamu OKUZAWA, Kazuhiko HATTORI, Gotaro ODAWARA

Department of Precision Eng., Faculty of Eng., Univ. of Tokyo,
7-3-1, Hongo, Bunkyo-ku, Tokyo, 112, Japan

This paper describes the logic generation system in a silicon compiler for ASIC design.

ADL (Algorithm Description Language) is a tree structured graphical hardware description language, and LOS (LOgic generation System) generates a datapath and translates into SFDL (Symbolic Functional Description Language) which is a register transfer level language. The Circuit Information Processor is a new human machine interface as a design environment with an LCD digitizer.

Some design examples are shown for the performance of ADL and LOS, and some experimental results for CIP are also described.

1．　はじめに

ＬＳＩの適用分野の拡大や、情報処理機器の軽薄短小化に伴い、ＡＳＩＣへのニーズが高まっている。しかし、ＡＳＩＣの仕様を決定するのが、ＬＳＩの熟練設計者であるとは限らない。こういった技術者にとって、レジスタ・トランスファレベル（ＲＴレベル）の記述から回路を設計する従来の手法は、なじみにくいものである。そこで、回路の動作アルゴリズムを、ソフトウェア的に記述できる言語と、アルゴリズム記述を入力とした自動設計システムが望まれている。また、このようなアルゴリズム記述によるハードウェアの設計環境として、設計者が対話的、かつ机上設計と同様に、設計及び計算機への直接入力可能な環境が必要になってきている。

そこで我々は、上記のニーズに答えるべく、ＲＴレベルより高位のハードウェア記述言語から、論理回路図、さらにＬＳＩのレイアウトまでを自動設計するシリコンコンパイラシステムの開発を行なっている。シリコンコンパイラを構成する要素技術として、ＶＬＳＩの製造プロセスに無関係な論理生成の部分と、この結果から各製造プロセスに対応した自動レイアウトの２つに分けられる[1]。前者に対しては、アルゴリズムレベルのハードウェア記述言語ＡＤＬ（ＡlgorithmＤescription　Ｌanguage）[2]と、これを入力設計言語とする論理生成システムＬＯＳ（ＬOgic generation　System）、並びに、その設計環境としての論理情報エディタ[3]の開発を行なっている。また後者については、ゲートアレイをその実現対象として、自動配置・配線システムの開発を行なってきた[4]。

本稿では、我々のシリコンコンパイラシステムにおける論理生成部について、論理情報エディタ、ＡＤＬ、並びに論理生成システムについて述べる。

2．　論理情報エディタ[3]

我々のシリコンコンパイラシステムにおいては、以下のような設計環境が必要である。

(1) ハードウェアアルゴリズムを図形によって記述し、手書きで入力する。

(2) 設計者は思考を妨げられることなく、設計に集中できる。

(3) 編集・論理検証・シミュレーション等のＥＷＳの基本的な機能がサポートされている。

そこで、このような設計環境を実現するために、液晶ディジタイザを入力とした論理情報エディタを開発した。

2．1　仕様

(1) 入力情報

ＡＤＬ（図形的記述言語）で記述された論理情報。

(2) 出力情報

i) ＡＤＬを液晶パネル上に清書表示する。

ii) ＡＤＬをシリコンコンパイラの内部コードに変換し出力する。

(3) 手書きシンボル認識

i) 認識対象シンボル：英数字・演算記号。

ii) 認識率：99%以上。
　認識時間：0.5 秒／シンボル以下。

(4) 編集機能

消去・挿入等の基本的な編集機能を備える。

(5) テキスト変換機能

入力データを基に、ＡＤＬをシリコンコンパイラの内部コードに変換する。

(6) 辞書更新機能

誤認識が発生した場合、自動的に辞書を更新する。

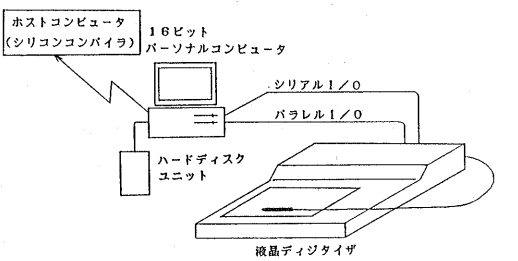


図１　論理情報エディタ構成

2．2　構成と操作法

図１に示すように本エディタは、液晶ディジタイザ、16ビットパーソナルコンピュータ、ハードディスクユニットから成っている。液晶ディジタイザは、ディジタイザの上に 640 x 400 dots (2100 x 1320 ㎜) の液晶パネルを搭載した入力ツールであり、入力と表示を同一平面上で行なうことができる。

本エディタには、図２に示すように入力・編集操作に用いられるコマンドを常駐メニュとして液晶の外部に設置している。入力は、まずシンボルの表示位置をスタイラスペンでポイントした後、内部記述を液晶上任意の位置に手書き入力し、常駐メニュよりシンボルを選択することによって、容易に実行できる。編集機能としては、一文字・文字列・ＡＤＬシンボル・ブロックの各単位で、消去・挿入・複写・移動がサポートされている。誤認識の修正・辞書更新も、該当文字のポイント並びにローカルメニュによって表示されるシンボ

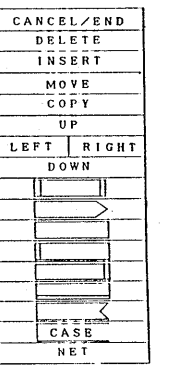


図２　常駐メニュ

ルからの選択によって実行され、設計者は、キーボード・ＣＲＴ等他のデバイスを使用することなく、入力・編集を行なうことが可能である。

３．　ハードウェア記述言語ＡＤＬ[2]

本章では、図形を用いたハードウェア記述言語ＡＤＬについて述べる。ＡＤＬは、論理生成システムによって論理回路に変換されることを前提とした言語で、アルゴリズムの記述に図形を用いている。

３．１　ＡＤＬの特徴および基本概念

本研究では、これまでに述べたことを背景として、システム設計者等にもＬＳＩ設計ができることを目的としたハードウェア記述言語ＡＤＬを開発した。ＡＤＬは以下のような特徴を持っている。

i)記述内容は、動作または内部仕様であり、ハードウェアの接続関係は記述しなくてもよい。

ii）記述レベルはアルゴリズムレベルである。すなわち、状態遷移等を考慮しなければならないレジスタ転送レベルよりも抽象度は高い。

iii)アルゴリズム中の処理、判定条件は、箱の中に記述し、その箱の形と接続（木構造図形）によってアルゴリズムを表現する。

iv）階層的な記述が可能である。すなわち、多クロックにわたる一連の動作をマクロとして定義し、それをコールすることができる。

v)動作記述に際して、ループ等の構造化言語の制御形式を用いることができる。

vi）ハードウェアの動作は、ブロックごとにプロセスという単位で記述される。

図形的にアルゴリズムを表現することにより、処理の流れを一目で追えるようになっている。図形は、ソフトウェアの構造化設計法であるＰＡＤ[5] をベースとしている。ＰＡＤは、フローチャート等に比べPASCAL，Ｃ 言語等の構造化プログラミング言語になじみやすいが、ＡＤＬでは、このＰＡＤのシンボルには特有の動作のためのシンボル、記号を加え、ハードウェア記述言語としている。

また、ハードウェアの基本的な概念として、iv）に述べたように、１ブロックまたは動作の単位をプロセスと呼び、プロセスの中で動作を記述するというスタイルをとる。他のプロセスの起動や、プロセスの終了待ちにより、並列動作や、ブロック間の同期を記述することが可能である。

３．２　ＡＤＬの構成と言語仕様

ＡＤＬは、入出力端子、記憶要素等を宣言する変数宣言部と、ハードウェアの動作を記述する動作記述部からなる。

```
(1) VARIABLE宣言
      /VARIABLE
      <変数名>(ビット幅) [添え字1，添え字2] <-S>
(2) STORAGE宣言
      /STOARAGE
      <変数名>(ビット幅) [アドレス上限，アドレス下限] <-S>
(3) TERMINAL宣言
      /INTERM
      <変数名>(ビット幅) <-S>
      /OUTTERM
      <変数名>(ビット幅) <-S>
(4) BUS宣言
      /BUS
      <BUS名>：<バスに接続される変数名または端子名>
(5) MAINPR宣言
      /MAINPR
      <メインプロセス名>
```

図３　ＡＤＬ変数宣言部

３．２．１　変数宣言部

変数宣言部では、動作記述中で使用される変数（レジスタで実現される）、メモリ、入出力端子等をテキストスタイルで宣言する。変数宣言部は５つの部分からなり、

(1)VARIABLE 宣言：動作記述中の変数にレジスタ（フリップフロップ）を使用することを宣言する。

(2)STORAGE宣言は：動作記述中の変数にアドレスを指定してアクセスする記憶装置（具体的にはメモリ）を使用することを宣言する。

(3)INTERM 宣言：入力端子の宣言。
OUTTERM宣言：出力端子または、双方向端子として宣言するもの。

(4)BUS宣言：上記で指定した中で内部バスに接続されている変数を宣言する。

(5)MAINPR 宣言：全プロセス中最初に起動するプロセスを宣言する。

をそれぞれ記述する。図３に記述仕様を示す。

３．２．２　動作記述部

動作記述部では、ハードウェアの動作を図形を用いて階層的に記述する。ＡＤＬのシンボルとその意味を表１に示す。C,PASCAL等の構造化プログラミング言語と同様に、選択（２分岐・多分岐）や反復（前判定・後判定・ FOR判定）、ラベル等の構造化した制御を扱える。その他、特徴的な点を以下に示す。

(1) 数値

現在は、整数のみを取り扱うものとしている。これは、対象とする回路を画像処理等と考えているためであり、またこれにより、論理生成システムのアルゴリズムもシンプルなものとなる。

(2) 順次動作と並列動作

ＡＤＬでは、動作記述中で処理の並列性を設計者が決定しなければならない。そこで、順次動作と並列動作の区別をするマークを用意している。

i)同期順次動作

同期順次動作では、１つの処理が終了した後、クロックに同期して次の処理を行なう。ＡＤＬでは、その間に記号Ｖを記述し、同期順次動作を表わす。

	意味	図式	内容
定義	プロセス定義	プロセス名	プロセスの定義
	マクロ定義 ファンクション定義	マクロ・ファンクション名	マクロ及びファンクションの定義
実行	処理		1ステップ（1クロック）動作または一連の動作（手続き）
制御動作手順	順次動作 同期	処理1 ▽ 処理2	Vの上の動作（処理1）が完了後（クロックに同期して）下の動作（処理2）を実行する。
	順次動作 非同期	処理1 S 処理2	Sの上の動作（処理1）が完了後（クロックに同期しない）下の動作（処理2）を実行する。
	並列動作	処理1 処理2	各動作（処理1・処理2）を並列に（同時に）実行する。
	反復 前判定	反復条件	ループに入る前に反復条件を調べる。
	反復 後判定	反復停止条件	ループに入った後で反復停止条件を調べる。
	反復 FOR文	I:=M,N(K)	IをMからKずつ増加しながらNまで右側の処理を繰り返す。

	意味	図式	内容
制御	選択 二分岐	条件	条件が成立した場合右上の処理を、不成立の場合右下の処理を行なう。
	選択 多分岐	ラベル（整数） ラベル 変数 ラベル	変数の値と一致するラベルの右側の処理を行なう。
	WAIT文	条件	条件が成立すると右側（又は下）の処理を実行する。
	並列実行レプリケータ	I=M,N	右側の処理をN−M+1個並列に実行する。
基本処理	転送・結合	変数（端子）名:=式	変数（端子）に式の値を代入する。
	他プロセスの起動	>> プロセス名	>> でさされたプロセスを起動する

表1
ADLのシンボル

ii）非同期順次動作

非同期順次動作では1つの処理が終了したのち、クロックに関係なく次の処理を行なう。ADLでは、上記のVの代わりにSを用いてこれを表わす。

iii)並列動作

並列動作では、記述された処理を並列に行なう。ADLでは、シンボルどうしを縦線で接続し、V、Sのマークを何もつけなければそれらの処理は並列処理とみなされる。

(3) プロセス間の記述

他プロセスの起動によって指定されたプロセスは、動作を始める。ADLで設計され、生成されるハードウェアは、各プロセスに対応する有効フラグ（フリップフロップ）を持っている。また、あるプロセスが動いていれば偽、停止していれば真となるEOP関数を用意しており、これを参照することによって、他のプロセスの状態を確認でき、プロセス間の通信等を行なうことが可能である。

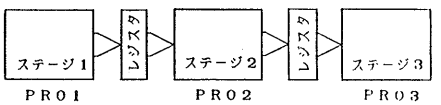
(4) マクロとファンクション

マクロは、一連の動作のまとまりを手続き化するものである。ただし、プロセスと違い動作の主体とはならないため、プロセスでコールされるだけにとどまる。また、引き数はとれない。コールされたマクロは、論理生成システムのプリプロセッサによって展開される。

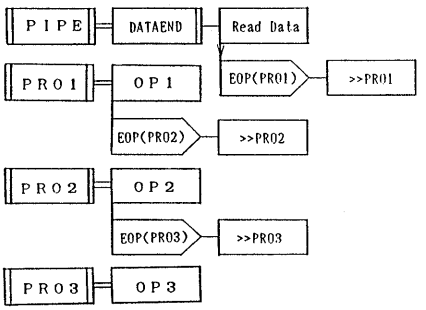
ファンクション定義では、組み合わせ論理で実現可能な演算機能を定義する。マクロと異なる点は、単一ステップで完了する動作の集合であること、すなわち、多クロック要する動作は記述できない点と、引き数をとれる点である。

3．3　ADLの記述例

本節では、パイプライン処理の記述を例にして、ADLの記述性能、特にプロセス間の同期の取り方について述べる。ADLを用いてパイプライン処理を記述しようとする場合、各ステージの処理を1プロセスとして記述する。図4に3ステージからなるパイプライン処理のモデルとそれに対するADL記述を示す。全体を制御するプロセスがPIPEで、これが最初のステージであるPRO1に次々と起動をかける。各ステージは、直前のステージに起動をかけられ、処理を始める。処理の終わりには次のステージの処理が完了しているか否かを確認し、完了していれば起動をかけ、そうでなければ完了するまで待つ。この記述は、WAIT文とEOP関数の組み合わせで行なっている。このように、パイプライン処理のような並列処理の手法に対してもADLで記述が可能であり、しかも、明確に示せることがわかる。



(a) パイプライン



(b) ADL

図4　ADLの記述例

４．　論理生成システムＬＯＳ

本章では、先のＡＤＬを設計入力言語とする論理生成システムＬＯＳについて述べる。

４．１　全体構成

本生成システムは、図５に示すように論理情報エディタ、ＡＤＬベリファイア、ＡＤＬ基本構造生成、状態遷移表現生成、データパスの割り付け・統合化、制御論理生成、ゲート生成の７つより構成される。

論理情報エディタより入力されたＡＤＬは、ＡＤＬ基本構造生成により、ＡＤＬ基本構造に変換される。そして、その結果より、データパスの割り付け・統合化によりデータの流れを表わすデータパスを作成し、また、状態遷移表現生成によりＳＦＤＬ(Ｓymbolic Ｆunctional Ｄescription　Ｌanguage)[6]と呼ぶ状態遷移を表わす言語に変換する。次に、制御論理生成が、ＳＦＤＬよりそれぞれの動作を制御するためのブール式を生成し、最後にゲート生成が、ブール式とデータパスとを合わせて論理回路を生成する。また、ＡＤＬベリファイアは、入力されたＡＤＬが所望の動作をしているかどうかを確認する検証ツールである。

さて、本システムはインプリメント中であり、現在ＡＤＬの入力、基本構造生成、状態遷移表現生成、及びデータパス生成の基本的な部分のインプリメントを完了している。以下、この生成のフローに沿って、各部の手法を述べる。

４．２　ＡＤＬ基本構造生成

ＡＤＬ基本構造とは、ハードウェアのアルゴリズムを処理と選択のシンボルのみで記述したものである。従って、ＡＤＬ基本構造生成では、まず使用されているマクロを展開した後、入力されたＡＤＬに使用されている反復制御、ＷＡＩＴ文を２分岐の選択を使用したループに展開する。図６に、例として前判定反復の展開を示す。

４．３　状態遷移表現生成

状態遷移表現生成のサブプログラムでは、ＡＤＬ基本構造から状態遷移表現（レジスタ転送レベルの記述）であるＳＦＤＬを生成する。以下、ＳＦＤＬ、生成アルゴリズムについて述べる。

４．３．１　ＳＦＤＬ

ＳＦＤＬは、我々が開発した図形的機能記述言語である。記述内容は機能記述で、レベルはレジスタ転送レベルである。木構造図形を用いる点は、ＡＤＬと同じであるが、以下の点でＡＤＬと異なる。

(1) 動作記述の基本単位は、プロセス定義でなく状態定義である。

(2) ループ、ＷＡＩＴ文はなく、処理と選択のみが許される。

(3) 他プロセスの起動、ジャンプ文はなく、他へ制御を移すのは、状態遷移文である。

図７に、ＳＦＤＬの例とこれに相当する状態遷移図を示す。状態遷移図と１対１なので、これより制御論理を組むことが可能である。

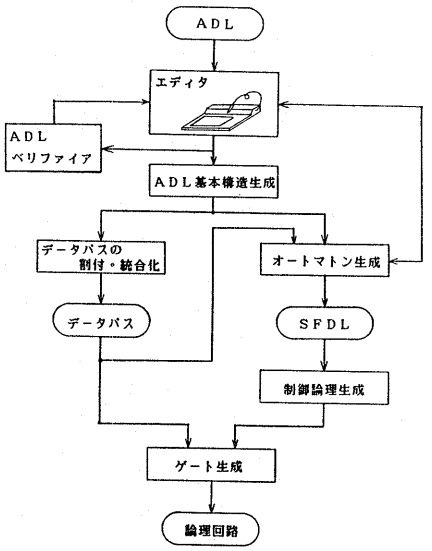


図５　論理生成システム構成

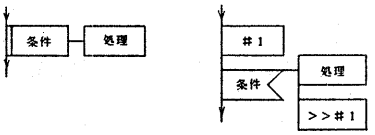
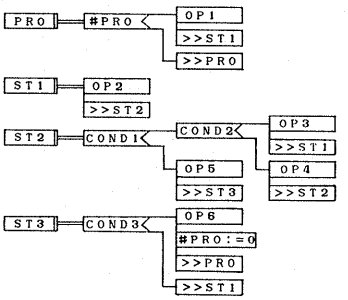
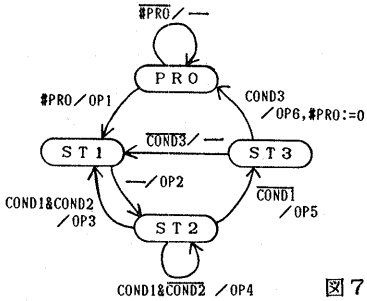


図６　ＡＤＬ基本構造生成（前判定処理）



(a) ＳＦＤＬ



(b) 状態遷移図

図７　ＳＦＤＬ

４．３．２　生成アルゴリズム

まず、ＡＤＬ基本構造生成で生成されたＡＤＬ基本構造をステップ毎に分割する。つまり、ＡＤＬの記述でシンボル接続記号Ｖによって接続されている処理を分割する。そして、分割した各々に１つの状態を割り付けることによって、ＳＦＤＬを生成する。図８に、ＡＤＬ基本構造から、ＳＦＤＬを生成した例を示す。　さて、この状態遷移表現生成では、上記の分割に際し、プロセス１つに対し、１つのプロセス有効フラグレジスタを自動的に生成する。このプロセス有効フラグレジスタは、１ビットのレジスタであり、これによりプロセスの制御が可能である。つまり、ＡＤＬで他のプロセスを起動する場合の記述は、ＡＤＬの他プロセス起動コマンドとその目的とするプロセス名を記述するが、このプロセス有効フラグレジスタを導入することにより、実際の操作は、目的とするプロセスのプロセス有効フラグレジスタに’１’を代入することになる。また、プロセスを停止させるためには、目的とするプロセスのプロセス有効フラグレジスタに’０’を代入する。このため、プロセス有効フラグレジスタの生成と同じ時に、各プロセスに対してプロセス起動処理、プロセス終了処理を行なっている。（図８参照）

プロセス有効フラグレジスタは、値の入力により目的とするプロセスを制御するだけでなく、目的とするプロセスのプロセス有効フラグレジスタの内容を参照することにより、そのプロセスの動作状態を示すことができる。すなわち、目的とするプロセスのプロセス有効フラグレジスタの内容が’１’であれば、そのプロセスは動作中であり、’０’であればそのプロセスは動作していないことがわかる。つまり、ＡＤＬで使用しているＥＯＰ関数は、このプロセス有効フラグレジスタの否定をとったものである。

また、ＡＤＬ基本構造には、頻繁にラベルを使用しているが、ＳＦＤＬでは、このラベルの使用を許していない。ラベルは、ＪＵＭＰの飛び先であり、状態を割りあてることにより、状態を定義することができる。従って、このようなラベルを状態定義のシンボルと統合するためのラベル統合処理も状態遷移表現生成で行なっている。

４．４　データパス生成

本節では、ＡＤＬ基本構造を基に、回路のデータ転送路を自動設計するデータパス生成サブプログラムについて述べる。データパス生成では、演算のスケジューリングとハードウェア資源の割りつけの２つが問題となる。以下、これらについて概説する。

４．４．１　演算のスケジューリング

ＡＤＬでは、ハードウェアで実現されるアルゴリズムを、並列動作、順次動作に分けて記述することにより、ハードウェアの実行速度に関する問題を言語の記述に反映できる。従って、演算の順序、並列性は、ＡＤＬの記述により、明確に定められるため、演算のスケジューリングをシステムが行なう必要はない。このように、設計者の意向を処理速度に関して細かく反映できるという長所をもつ反面、効率のよいハードウェアを実現するには、ある程度練習が必要となる。

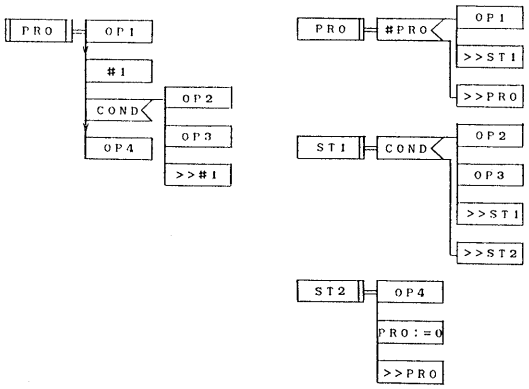


図８　ＳＦＤＬ生成例

４．４．２　ハードウェア資源の割りつけ

ハードウェアの資源の割りつけ問題については、記憶要素、演算要素、結合要素に分けて述べる。

(1) 記憶要素

記憶要素は、ＡＤＬでは、変数宣言部に宣言される。変数は、/VARと/STORAGEに分かれるが、/VARで宣言されたものはレジスタで、/STORAGEで宣言されたものはメモリで実現される。現在のシステムでは、１変数に対し、１つの記憶要素を対応させており、記憶要素に対する最適化は行なっていない。

(2) 演算要素

演算要素は、自動的にハードウェアの割りつけが行なわれる。動作記述中の算術演算子や論理演算子は、ハードウェアでは演算ユニットとして実現される。さて、この演算要素については、演算の行なわれるタイミングと演算の種類・入力変数を考慮して、演算子の統合を行なっている。本システムでは、これにグラフを用いている。各々の演算子をグラフのノードとし、同時に使用されないものどうしをエッジで結ぶ。このグラフから、部分完全グラフ（クリーク）を取り出し、それを１つの演算ユニットとすれば、最小の数の演算ユニットで済む。グラフのクリーク分割をハードウェア資源の割りつけに応用する考えは、ＣＭＵのＦacetシステムの論文[7] にみられ、本システムでは、その近似法をベースとしている。クリーク分割およびその近似解法については、文献[7] を参照されたい。

(3) 結合要素

記憶要素、演算要素が決定した後、それらは転送路で結ばれる。１つの要素に多数の入力があるときは、マルチプレクサが自動生成される。また、バスを形成するか否かについては、現在設計者の指示に従っており、ＡＤＬの変数宣言部中の/BUS宣言において、バスに接続するレジスタやメモリを指定している。このように本システムでは、設計者の考えをデータパスに直接反映できるようになっている。しかし、逆にＡＤＬにてハードウェアを設計する段階において、ある程度データパスを意識して設計を行なわなければならない。

データパス生成例を、図９に示す。

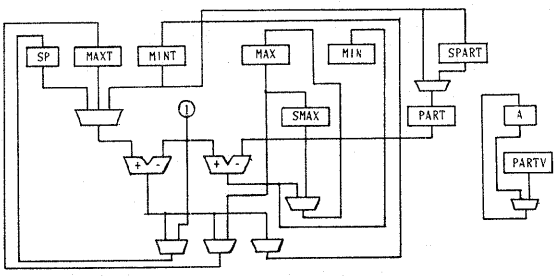


図９　データパス生成例　(ソート回路)

４．５　制御論理生成

この制御論理生成では、状態遷移表現生成により生成されたＳＦＤＬより、それぞれの状態で実行される処理の条件をＳＦＤＬの下の階層から上にたどりながら検索する。そして、その結果より処理が実行されるための条件式を導き出す。図１０に制御論理生成の例を示す。転送先Ａ、転送元Ｂ＋Ｃの処理（ＳＦＤＬの記述中では、Ａ：＝Ｂ＋Ｃ）の場合、処理のシンボルから矢印のように下の階層から上へ、その処理が選択される条件を全て検索し、検索された条件全てとその処理が実行される時の状態と論理積をとることにより、条件式を得ることができる。また、同じ処理が複数ある場合には、まずそれぞれの処理が実行されるための条件式を導き出し、その導き出された全ての条件式の論理和をとることにより、その処理の実行されるための条件式を導き出す。

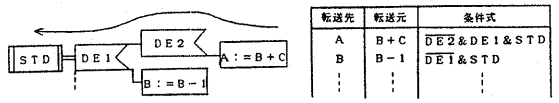


図１０　制御論理生成

表２　エディタの認識率

使用辞書	熟練者（２人）	初心者（２人）
標準辞書	９７．６％	９１．５％
個人辞書	９９．５％	９７．９％

表３　設計工程時間

	机上設計（分）	本エディタによる設計（分）
ＰＩＴ１	２１０	１５０
ＰＩＴ２	３００	１８０
平均	２５５	１６５

５．　評価

本章では、論理情報エディタ、ＡＤＬ、並びに論理生成システムについての評価結果を示す。

５．１　論理情報エディタに関する評価

表２は、認識率を示したものである。個人辞書の作成により、認識率の向上が認められ、シンボルの入力には支障のない結果が得られている。また認識時間も、１シンボルあたり0.3 秒以下であり、仕様を満足している。操作性に関しては、仕様から内部コードを生成するまでの全設計工程時間を評価した（表３）。設計対象は、プロセッサの周辺装置である Programmable Interval Timer(約3Kｹﾞｰﾄ) である。本エディタの使用により、机上で紙と鉛筆を用いた場合よりもスムーズに設計が行なわれ、また、図形による記述と手書き入力の実現により、思考を妨げられずに設計に集中できた。ページング・マルチウィンドウ等の画面制御コマンドのサポートが、今後の課題である。

表４　ＤＳＰの仕様

データ幅	データ：16 bit　命令語：32 bit
演算機能	並列乗算器、ＡＬＵ、バレルシフタ
入出力機能	シリアルポート２、パラレルポート１、ＤＭＡ２
メモリ	インストラクションＲＯＭ：32bit 2kw データＲＡＭ：16bit 4kw
方　式	３サイクルのパイプライン方式

５．２　ＡＤＬに関する評価

評価にあたり、画像処理を想定した仮想的なＤＳＰを、ＡＤＬを用いて設計した。本ＤＳＰの設計仕様を表４に示す。このＤＳＰは、松下電子のMN1901 [8] に準拠した命令を持っている。

さて、ＡＤＬの記述例を図１１に示す。MAINP は、初めに起動されるプロセスで、PR1,PR2,PR3 は、それぞれフェッチサイクル、メモリアクセスサイクル、実行サイクルである。これらより、テキスト形式の言語に比べＡＤＬを用いる利点として、

(1) 並列に行なわれる処理が明確である。
(2) 各処理の実行される条件が明確である。
(3) 反復して行なわれる処理の範囲・反復の条件が明確である。

以上の３点が挙げられる。

また、ＡＤＬの記述量（Box 数）は、ハードウェアの規模よりも、命令や条件の複雑さで決まる。

５．３　論理生成システムに関する評価

前節で述べたＤＳＰについての、論理生成性能を表５に示す。これは、ＡＤＬ・ＳＦＤＬの記述量（Box数）と、ＳＦＤＬにおける状態レジスタの数を、回路ブロックごとにまとめたものである。表５より、生成されるＳＦＤＬは、ＡＤＬに比べ約 30% 多いことがわかる。これよりＡＤＬが、ＲＴレベルの言語に比べ、ある程度抽象度が上がっていることがわかる。

しかし、現在状態レジスタを１状態あたり１つ生成する単純な方法を採用しているが、状態レジスタ数を削減するための改善が必要である。また、データパス生成におけるバスの発生等は、設計者が指定するものとしており、当初の目的であるハードウェアの知識が乏しい技術者にとっては、性能のよいハードウェアを設計することは困難である。このような生成回路の性能については、まだ設計者自身に委ねられている。

６．　おわりに

本稿では、ＡＳＩＣ設計を目標としたシリコンコンパイラにおける論理生成システムについて、その設計環境、ハードウェア記述言語ＡＤＬ、並びに論理生成システムについて、報告した。図形的な表現であるＡＤＬの記述容易性が確認され、またそれを入力とする論理生成システムによって、データパス並びに制御論理を生成できた。今後は、ゲート生成までのシステムの完成と、生成回路の性能を規定するライブラリ等の検討を行なう予定である。

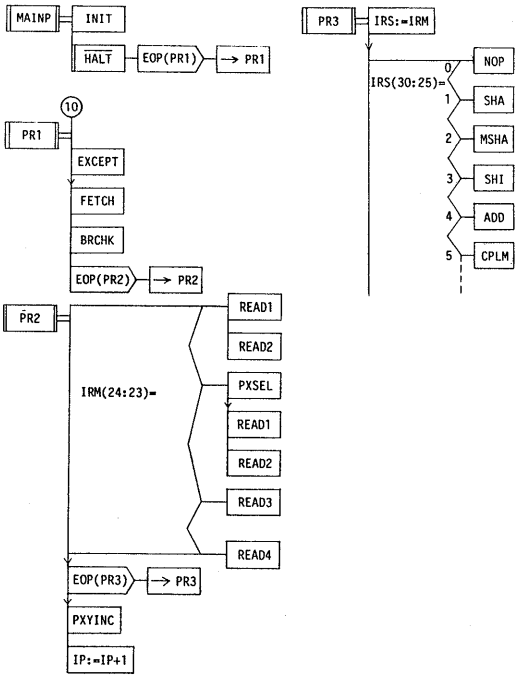


図１１　ＤＳＰのＡＤＬによる設計

表５　ＡＤＬと生成結果

ブロック	ＡＤＬ	ＳＦＤＬ	状態レジスタ	回路ブロック
Ｆetch	33	47	3	
ＭＡ Ｄecode	177	190	4	加算器　２ バス　３
実　行 ＡＬＵ ＡＬＵＭＵＬ ＳＨＩＦＴ ＭＯＶ	 376 224 26 139	 501 293 36 183	 67 39 6 20	 ＡＬＵ 並列乗算器 バレルシフタ バス　１
ＭＷ	127	127	0	
Ｉ／Ｏ	92	173	15	減算カウンタ２ シフタ２ バッファ３
合　計	1,194	1,550	154	

参考文献

[1] 平山：シリコン・コンパイラ，情報処理学会誌，Vol.25 No.10，pp.1153-1160，1984.

[2] 奥沢他：ハードウェア記述言語ＡＤＬの開発と論理生成手法，情報処理学会第34回全国大会講演論文集，pp.2005-2006，1987.

[3] 奥沢他：ハードウェア記述言語ＡＤＬの入力エディタの開発，情報処理学会第35回全国大会講演論文集，pp.2257-2258，1987.

[4] G. Odawara et al. : Partitioning and Placement Technique for C-MOS Gate Arrays, IEEE Trans.on CAD, Vol.6 No.3, pp.355-363, 1987.

[5] 二村：プログラム技法－ＰＡＤによる構造化プログラムング，オーム社，1984.

[6] G.Odawara et al. : A Symbolic Functional Description Language, Proc. of 21st Design Automation Conf., pp.73-80, 1984.

[7] C.J. Tseng et al. : A Procedure for the Automated Synthesis of Digital Systems, Proc. of 20th Design Automation Conf., pp.490-496, 1983.

[8] MN1901/MN1909 ﾕｰｻﾞｰｽﾞ ﾏﾆｭｱﾙ,松下電子工業，1985.