計算機アーキテクチャ 46－2
（1982. 9. 17）

コンパイラチップの設計と評価

平山正治　瀬尾和男　房岡　瑾
（三菱電機　中央研究所）

1．まえがき

近年のVLSI技術の発展はめざましく，数万～数十万ゲート程度の規模をもつ回路を1チップの中に組み込む事が可能な状況になりつつある．　しかし，これ程の複雑さをもつチップでは，開発時間やコストの大幅な増大とこれに見合うだけの消費量が望めない事から，ユーザが望む機能をもったチップが半導体メーカから汎用的なチップとして供給される事は期待できない．　この結果，計算機のアーキテクトやアプリケーションエンジニアは，彼らが必要とする機能をもつVLSIチップを実現するために，　自らチップの設計，開発に携さわる事が必要となってきている．　このためには，従来，半導体メーカが採ってきたような高度の専門知識を必要とする設計開発アプローチではなく，もっと容易にVLSIチップを開発できるような設計開発アプローチが必要である．　Mead & Conwayは"λ"と呼ぶ基本単位を導入し，これに基づいた非常に簡単化した設計ルールによってVLSIの設計を行なう方法を提案し，(1)　このアプローチは米国内の教育研究機関を中心とした広い範囲で利用されるようになってきている．

我々はこのようなアプローチによって，複雑なソフトウェアをVLSI化する事の可能性，ユーザによるVLSIチップの設計開発の実現性，および，VLSI指向型アーキテクチャの実現を追求する事を目的として，計算機言語PASCALのソースコードを仮想スタックマシンのP-codeにコンパイルする処理を数チップのVLSIで実現する事を試みた．　本稿では，このコンパイラマシンのアーキテクチャと構文解析を行なうParsing Engineチップの設計と評価について報告する．

2．コンパイラチップの意義

近年の計算機利用のほとんどはFORTRAN，PASCAL等の高級言語を利用するものであり，高級言語を直接実行する実用的な高級言語マシンが期待できない現状において，コンパイル処理は今後とも計算機システムに不可欠な処理と考えられる．　また，コンパイル処理は実際の計算処理に対して補足的な処理ではあるが，プログラムの開発時等，その実行頻度は非常に高く，大幅な高速化が望まれる処理のひとつである．　通常，コンパイル処理は本来の計算処理と同様にメインフレームの計算機において実行されているが，コンパイル処理のような記号処理，非数値処理はメインフレームの得意とする処理ではなく，プログラムの編集，デバッグ等とともにマンマシンインターフェイスに属する種類の処理と考えられる．　従って，コンパイラを数チップのVLSIで実現できれば，これを端末装置の内部に取り込んでしまい，プログラムの編集，コンパイル，デバッグ，およびテストランまでの処理をすべて実行するインテリジェント端末が実現可能になる．　このように，マンマシンインターフェイスに関連する処理をメインフレームから取り除き，メインフレームは完成されたオブジェクコードを受け取って本来の計算処理だけを行なうような効率的機能分散システムが将来の計算システムとして有効であると思われる．

一方，コンパイラはかなり複雑なソフトウェアであるが，その構造は以下に示す４つの連続した処理に分解する事ができる．

（1）字句解析 ・・・ 高級言語のソースコードを走査し，内部コードに変換する．

（2）構文解析 ・・・ 内部コードを走査し，その言語の文法規則に適合しているかどうかチェックするとともに，構文に従って生成すべきコンパイラテーブル，またはオブジェクトコードの種類を判定する．

（3）テーブル生成 ・・・ プログラムの宣言部に従って，オブジェクトコードの生成に必要なコンパイル環境（テーブル群）を生成する．

（4）コード生成 ・・・ プログラムの実行部に従って，対象とする計算機のオブジェクトコードを生成する．

以上の処理のうち，字句解析や構文解析は，PASCALのような正規表現の構文規則をもつ言語においては，有限状態マシンのモデルが適用でき，この回路はPLAとスタックとによって簡単にハードウェア化できる．　また，数種の基本セルを組み合わせる事によって正規表現を直接解析するVLSIチップの提案も行なわれている．[2]　一方，テーブル生成とコード生成における処理はコンパイラテーブル群に対する登録，更新，検索処理と考えられ，このようなデータベース操作をSystolic Arrayを用いてVLSI化する試みが提案されている．[3]　さらに，コンパイル処理における上記4個の処理は互いに並行して動作する事ができ，このようなパイプライン構造は並列処理プロセッサによるハードウェア化に適した構造をもっている．　以上のように，コンパイラの処理過程や処理内容はVLSI化に適しており，ソフトウェアのVLSI化の具体例として好適なものと考えられる．

3．アーキテクチャ

3.1　コンパイラマシンの全体構成

現在検討中のコンパイラマシンの全体構成は，コンパイルアルゴリズムの各パイプライン・ステージに対応して，4個の処理装置（Lexical Engine，Parsing Engine，Table Generation Engine，Code Generation Engine）と1個の共有メモリ装置（List Memory）が2本の共通バス（P-bus，Q-bus）で結合された構成となっている．（この構成を図1に示す．）基本的にこれらの処理装置は各々1個のVLSIで実現する事をめざし，従って，コンパイラマシン全体は5個の処理チップと数個のメモリチップで構成される．

各処理装置は前ステージの処理装置からのデータパケット（トークン）を受け取り，何らかの処理を行なった後，次のステージの処理装置にデータパケットを送るという単純な実行サイクルをくり返す事により，Lexical Engineに入力されたPASCALのソースコードはCode

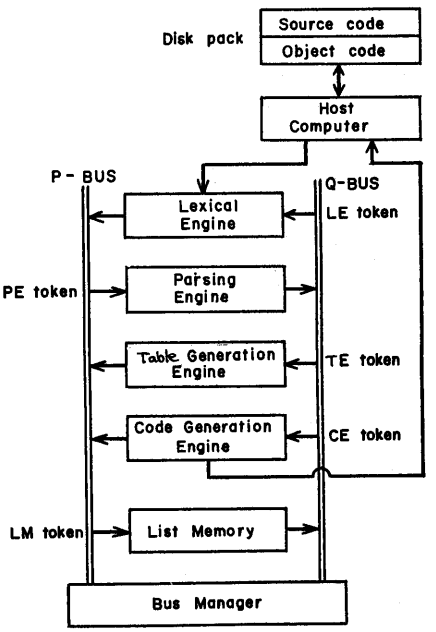

図1　コンパイラマシンの全体構成

Generation Engine から仮想スタックマシンの P-code に変換されて出力される。（各データパケットは転送先の処理装置の名前をとって，LEトークン，CGトークン等と呼ばれる。）

3.2 構成要素の機能概要

(1) Lexical Engine

コンパイラマシンに入力されるPASCALソースプログラムはLexical EngineによってASCIIコードによる表現から内部コードによる表現に変換される。この過程は大きく２つの処理に分ける事ができ，最初の処理では入力された文字列を走査する事によって，特殊記号，予約語，識別名，整定数，実定数，文字定数に分解する。また，この段階で不要なブランク，改行コード，コメント等は取り除かれる。次の段階として，特殊記号と予約語は１B（バイト）長の内部コードに変換される。識別名と各定数はList Memory に転送され，既定義か未定義かのチェックと各テーブルへの登録が行なわれる。各定数の場合はこれらが登録された変換テーブルの番地がList Memoryから転送されるので，これに定数のタイプを示すコードを付加して２B長の内部コードを生成する。識別名の場合は，システムで予め定義されている標準名かユーザ定義の識別名かの２つの可能性があり，List Memoryからの応答に従ってそれぞれ，３B長，２B長の内部コードを生成する。このようにして生成された１～３B長の内部コードはPEトークンとしてParsing Engine に転送される。（図２に内部コードの一覧表を示す。）

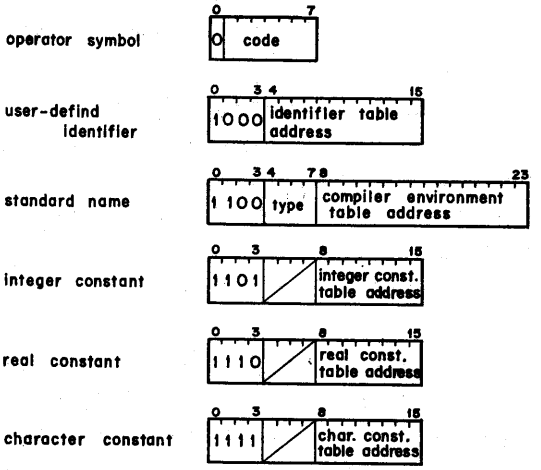

図２ 内部コード一覧表

(2) Parsing Engine

Parsing Engine はLexical Engine から転送されてくる内部コード（PEトークン）を受け取り，これがPASCALの文法に適合しているかどうかチェックし，構文に従ってコンパイラテーブルを生成するためのコマンド（TEトークン）か，あるいはオブジェクトコードを生成するためのコマンド（CEトークン）を発生し，それぞれ Table Generation Engine か Code Generation Engine に送出する。

PASCALの文法は Greibach's normal form と呼ばれる Z → αXY という形式の書きかえ規則によってほとんど記述する事ができる。（ここで X，Y，Z は非端末記号，α は handle と呼ばれる端末記号である。）このような性質をもつ言語に対しては top-down に構文解析を行なう事が容易であり，これを実現するためにはスタックつきの有限状態マシンを構成すればよい。すなわち，X，Y，Z を〈状態〉，α を〈入力シンボル〉とすれば，現在の状態 Z においてシンボル α を入力した時，次の状態 X，Y を生成するようなマシンを考えればよい。ここで状態 X，Y に応じて，以下のようなスタック操作と次の状態の決定を行なう必要がある。

(Z, a) ⟶ (X, Y)

if X = nil and Y = nil then pop up stack → Z

if X ≠ nil and Y = nil then X → Z

if X ≠ nil and Y ≠ nil then X → Z, Y → push down stack

(3) Table Generation Engine

Table Generation Engine は Parsing Engine から送出される TE トークンを受け取り，この指示に従ってオブジェクトコードの生成に必要な変数テーブル，タイプテーブル，関数テーブル等のコンパイラテーブル群を作成する。しかし，前段階の Parsing Engine において構文解析が完了しているため，これらの処理は比較的簡単なものである。ここで行なわれるテーブル群の作成操作としては，PASCAL プログラムの宣言部の構文に対応して，プログラムパラメータの登録，ラベルの登録，定数名の登録とその値のセット，タイプ名の登録とその構造のセット，変数名の登録とそのタイプのセット，および関数名とプロシジャ名の登録とその仮引数の変数名，タイプ等のセットがあり，これらの操作をプログラム，または各プロシジャの宣言範囲に従って行なう。

以上の操作を行なうために Table Generation Engine は，TE トークンを解釈して簡単なデータベース操作を行なう LM トークンに展開し，これを List Memory に送出して応答を受け取るという処理をくり返す。この結果，Code Generation Engine から参照されるすべてのコンパイラテーブル群が List Memory 中に生成される。

(4) Code Generation Engine

Code Generation Engine は Parsing Engine から送出される CE トークンを受け取り，Table Generation Engine によって List Memory 中に生成されたコンパイラテーブル群を参照しながら，仮想スタックマシンの P-code を生成する。ここで実行される主な操作は，プログラムの実行部中における <statement> の構文に従って，変数への値の代入，関数，プロシジャの呼出し，くり返し構文の条件判定や飛び先ラベル等の P-code の生成，および，<expression> の構文に従って，各変数間の種々の演算を行なう P-code の生成を行なう事である。この操作においても，Code Generation Engine はコンパイラテーブル群を参照するための LE トークンを発生し，List Memory からの応答に従ってコード生成を行なう。

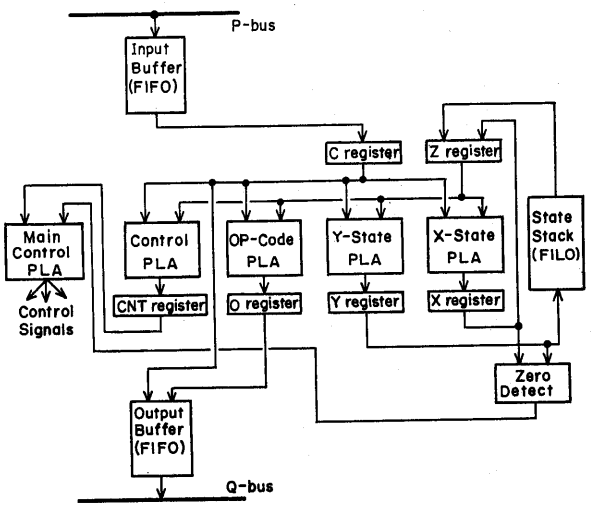

図3 Parsing Engine の構成

3.3 Parsing Engine の構成

Parsing Engine における構文解析を行なう有限状態マシンは，2個の PLA (X-state PLA, Y-state PLA)；状態スタック，および，

いくつかのラッチで実現できる。また，構文解析回路の他に，次段の処理装置に送出するTE，CEトークンの生成，データ転送制御情報の生成，および，このEngine全体の制御回路もそれぞれ1個ずつのPLA（OP-code PLA，Control PLA，Main Control PLA）を必要とする。これらのPLAは表1に示す入力数，プロダクト項数，出力数をもつ事がPASCALの構文規則の解析から得られた。（詳細は省略する。）これらの構成要素の他に，他の処理装置とのデータ通信を円滑に行なうためのFIFO形式の入出力バッファを備えている。これらの構成要素からなるParsing Engineは，図3に示されるように非常に簡単化された構成になっている。

表1 Parsing処理用PLAのサイズ

	Inputs	Product Terms	Outputs
X-state PLA	16	347	8
Y-state PLA	16	157	8
OP-code PLA	16	276	8
Control PLA	16	298	6
Main Control PLA	21	44	15

4. Parsingチップの設計

4.1 VLSI開発サポートシステム

VLSIチップの設計を行なう上では，十分な使いやすさをもった開発サポートシステムの確立が必須である。我々はMead & Conwayアプローチにのっとって MITで提案されたDPL（Design Procedure Language）[4]を中心とした開発サポートシステム（"会話型FDPLシステム"と呼ぶ）を開発し，これを利用してVLSIチップのレイアウトを行なっている。

DPLはLispベースのシステムであり，object-orientedなVLSIレイアウト言語である。このDPLにおいては，"defcell"と呼ぶ関数によって，オブジェクトの"class"が定義され，"partq"と呼ぶ関数によって，あるclassに属する1個の"instance"が生成される。基本的なclassとしては"rectangle"があり，このパラメータにはその大きさを決めるための"width"と"length"，および，この図形が属すマスクの層を指定するための"layer"がある。DPLではこの他に，他のinstanceとの相対的な位置関係や，回転，対称等の座標変換を指定する関数があり，これらの関数によってチップのレイアウトを階層的に行なっていく事ができる。

図4 DPLによるPLAの記述（部分）

```
(defcell cell ()
        (partq m1 m-bus (width 14))
        (partq m2 m-bus (width 14)
               (center (pt-above (>> center m1) 7)))
        (partq c1 contact
               (center-left (>> center-left m1)))
        (partq c2 contact
               (center-left (>> center-left m2)))
        (partq p1 p-bus (length 14)
               (bottom-center (pt (+ 4 (>> x bottom-center c1))
                                  (+ -1 (>> y bottom-center c1)))))
        (partq p2 p-bus (length 14)
               (center (pt-to-right (>> center p1) 6)))
        (partq d1 d-bus (length 14)
               (center (pt-to-right (>> center p1) 3)))
        )

(defcell input ()
        (partq pb1 p-bus (width 14))
        (partq mb1 m-bus (width 14)
               (center (pt-below (>> center pb1) 12)))
        (partq mb2 m-bus (width 14)
               (center (pt-below (>> center mb1) 19)))

        . . .  )

(defcell pla ((in 6) (pterm 8) (out 4) (table *table*))
        (partq in-1 input)
        (replicateq in-row array (list (>> in) 1) (list (>> in-1)))

        (partq gnd-1 ground)
        (replicateq gnd-row1 array (list (>> in) 1) (list (>> gnd-1)))
        (align (>> gnd-row1) (>> top-center gnd-row1)
               (>> bottom-center in-row))

        (partq cell-1 cell)
        (replicateq cell-matrix1 array (list (>> in) (quotient (>> pterm) 2))
                    (list (>> cell-1)))
        (align (>> cell-matrix1) (>> top-center cell-matrix1)
               (pt-above (>> bottom-center gnd-row1) 1))

        . . .  )
```

我々の所で開発された会話型FDPLシステムは，MITのDPL仕様とほぼ同等の基本レイアウト機能を備えているが，キーボードから入力されるDPLの記述をカラーディスプレイ装置に表示する機能，および，ディスプレイ装置とタブレットを会話型で使用して直接図形を入力する機能やこれをDPLの記述に逆変換する機能も備えている．　また，表示図形の一部の拡大，縮小等も容易に行なえ，これらの機能によって新しいclassの定義や修正，およびレイアウトを構成する各要素間の配線をディスプレイ画面を見ながら会話的に行なえるという特徴をもっている．　さらに，4.3節で述べる"vgen"関数の導入等，いくつかの機能拡張を行なった非常に有効なVLSIレイアウトシステムが利用できる体制が確立されている．

4.2　各構成要素のレイアウト

Parsing Engineのチップレイアウトを行なう最初の段階として，このチップを構成するいくつかの構成要素のレイアウトを行なう．　これらの構成要素としては，図3からも明らかなように，PLA，stack，FIFOバッファ，ラッチ，セレクタ，およびI／Oパド等がある．　これらの構成要素のレイアウトにおいては適当なパラメータを設定し，後で適当な値を与える事によって任意のサイズのものが得られるように設計している．　たとえば，PLAの場合であれば，入力数，プロダクト項数，出力数，および，PLA書込みテーブルをパラメータとして設定している．

また，最も下位の簡単なセルの記述から，これらを組み合わせて上位レベルのセルを順次構成していくというDPLの階層設計の手法が有効に利用され，比較的小規模の記述によって大規模な構成要素のレイアウトを容易に行なう事ができる．　たとえば，PLAの場合であれば，これを構成するPLAセルや入力ドライバ等の基本セルのclassを定義し，これを行，列，またはアレー状に配列する事によりPLAセルマトリックス，入力ドライバ列等を作り，これらを適当な位置関係で配置する事によりPLA全体のレイアウトを行なっている．　PLAにおけるDPLによる記述の一部と最終的なプロット図を図4，図5に示す．

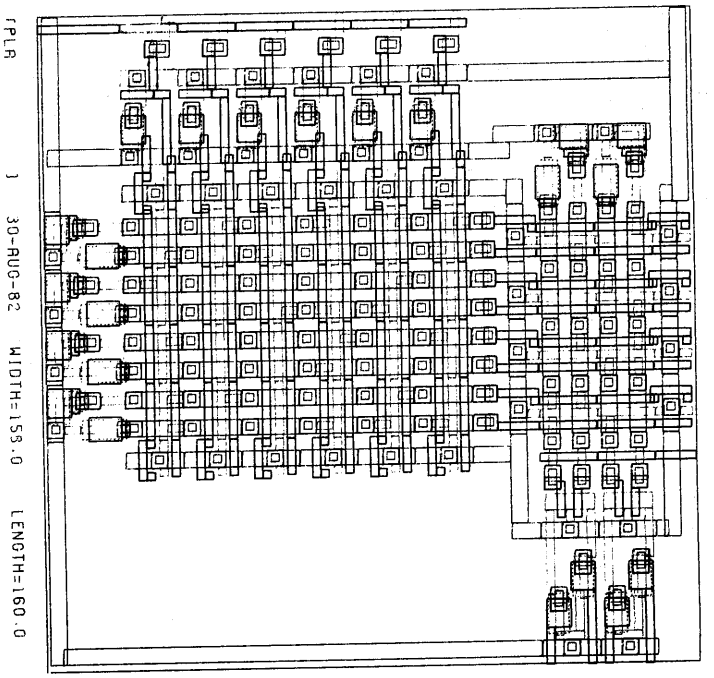

図5　PLAのプロット図

4.3 フロアプラン

チップを構成する各要素のレイアウトが完了した時点で，これら要素のチップ全体に対する配置を決定する．　PLA等の各構成要素はいくつかのパラメータによってその外形が決定されるため，この外形，入出力端子の位置，および各構成要素間の接続関係によって，最小の面積で最短の配線長をもつような最適配置を求める．　この後，I/Oパドの配置を決め，最後に電源，接地バス，データバス，およびその他の各要素間の配線パターンをレイアウトして全チップのレイアウトを完了する．

上記の最適配置を決定する操作は，現在，ディスプレイ装置上に表示したものを見ながら試行錯誤的に行なう必要があるが，この時に全構成要素の詳細パターンを保持しながら行なうと非常に大きな処理時間と表示時間を要してしまう．　我々の会話型FDPLシステムでは"vgeng"という関数を導入し，これを適当なinstanceに適用する事により，このinstanceから入出力端子以外のすべてのパターンを取り除き，外形と入出力端子だけをもつinstanceに変換してしまう．　この関数によって内部処理時間と表示時間が大幅に高速化されるだけでなく，入出力端子の位置が明らかとなって構成要素間の配線が容易になるという効果もある．

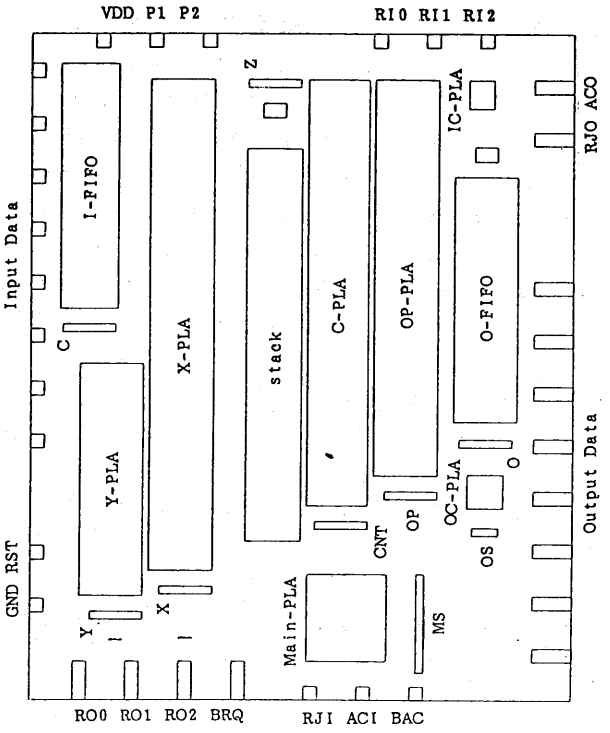

図6　Parsing Chipの全体レイアウト

この関数を使って全構成要素の配置を完了した時点でのParsing Engine全体のレイアウト図を図6に示す．（この図では入出力端子の表示を省略している．）このレイアウト図においては3.3節で示した構成要素の他に，入出力データ用に各8個，バス転送制御用に12個，および電源，グラウンド，基本クロック信号等の入力用に5個の合計33個のI/Oパドがチップの外形に沿って配置されている．

5. 評価

Parsing Engineを構成する7個の主な要素について，それぞれのパラメータ，外形の幅，長さ，トランジスタ数，および長方形の数を表2に示す．　また，これらの構成要素を用いて，図6のようにParsing Engineの全体をレイアウトした場合の各構成要素の占有面積，トランジスタ数，長方形の数，および，これらの全体に対する割合を表3に示す．　このParsingチップは全体で2,766λ×3,426λの大きさをもち，37,372個のトランジスタ，212,571個の長方形で構成されている．　現在のプロセス技術からλを2μmと仮定すれば，このチップは5.6mm × 6.9mm ＝ 36.6 mm² のサイズをもつVLSIとなり，

十分に1チップ化が可能なものであると考えられる．

このParsing Engineは上記の非常に少ない構成要素から構成され，ランダム回路をほとんど含まない事から明らかなように，このチップは非常に高いregularityをもっている．この事は約38,000個のトランジスタ，220,000個の長方形から構成されるチップの記述に約3,000行のHPLコードしか要さなかった事からも推定される．このチップではPLA部とメモリ部の占有面積がチップ全体の38.7％になり，コントロール部の3.0％に比べて高い値を示し，チップの多くが実質的な処理に使用されているのがわかる．一方，このチップでは各所に8ビット幅や16ビット幅のデータバスを配線する必要があり，このための占有面積が23.9％とかなり大きな部分を占めている．また，各構成要素のチップ全体に対する配置を人手によって行なったため，チップの未使用領域が31.3％も存在している．

このチップの設計には，チップレイアウトやプロセス技術に関してほとんど予備知識をもたなかった2人の人が実質的に約1.5ヶ月でチップ全体のレイアウトまでを完了しており，このチップの設計所要時間は約3人月と推定される．

表2 構成要素の規模

component	parameters	width (λ)	length (λ)	transistors	rectangles
PLA	I, P, O	14I + 7O + 46	7P + 104	I*P + O*P + 3I + 4O + P	5.5I*P + 2.75P*O + 48I + 24P + 35.5O + 18
stack	B, W	63W + 29	21B + 116	8B*W + 32W + 2	70B*W + 252W + 6B + 49
FIFO memory	B, W	35W + 138	28B + 70	6B*W + 10W + 38	47B*W + 100W + 6B +316
latch	B	34B	41.5	6B	53B
selecter	B	10B + 38	7B + 19	2B + 4	13B + 51
input pad	-	72	72	1	52
output pad	-	70	202	16	315

表3 Parsing Chipの規模

component	conditions	area ($K\lambda^2$)		transistors		rectangles	
X-PLA	16*347*8	841	8.9%	9796	26.2%	47720	22.4%
Y-PLA	16*157*8	399	4.2%	4005	10.7%	22242	10.5%
OP-PLA	16*276*8	663	7.0%	6980	18.7%	38054	17.9%
C-PLA	16*298*6	683	7.2%	6926	18.5%	39292	18.5%
(PLA total)		2286	24.1%	27707	74.1%	147290	69.3%
stack	32 bytes	573	6.1%	3074	8.2%	12348	5.8%
I-FIFO	32 bytes	370	3.9%	1894	5.1%	15596	7.3%
O-FIFO	32 bytes	370	3.9%	1894	5.1%	15596	7.3%
latches	49 bits	69	0.7%	294	0.8%	2597	1.2%
(memory total)		1382	14.6%	7156	19.2%	46137	21.7%
Main-PLA	21*44*15	183	1.9%	1751	4.7%	9668	4.5%
IC-PLA	4*8*4	22	0.2%	112	0.3%	1038	0.5%
OC-PLA	7*10*10	37	0.4%	241	0.6%	1609	0.8%
latches	27 bits	38	0.4%	162	0.4%	1431	0.7%
(control total)		280	3.0%	2266	6.1%	13746	6.5%
in pad	19	98	1.0%	19	0.1%	988	0.5%
out pad	14	198	2.1%	224	0.6%	4410	2.1%
wire	-	2264	23.9%	0	0.0%	-	-
(others total)		2560	27.0%	243	0.7%	5398	2.5%
(unused)	-	2967	31.3%				
total		9475	100.0%	37372	100.0%	212571	100.0%

6. あとがき

本稿では，複雑なソフトウェアをVLSI化する事の可能性やユーザによるVLSIチップの実現性を追求するために，PASCALコンパイラマシンのParsing処理を行なうVLSIチップの設計を試みた。この結果，PASCAL程度の文法をもつ言語の構文解析とその結果によるコンパイラテーブル生成とコード生成のための指示を出力する処理は，PLAを主体とした回路構成によって，5.6mm×6.9mmの大きさをもち，約38,000個のトランジスタを含むVLSIチップで実現できる事が示された。また，この程度の複雑さをもつチップでも，Mead & Conwayアプローチの簡単化された設計ルールと，階層型レイアウト言語HPLを中心とした効率的開発サポートツール（会話型FHPLシステム）の利用によって，VLSIに関する専門知識をほとんどもたない人でも約3人月の開発期間でチップの基本的な設計が可能である事が示された。

今後の問題としては，設計ルールチェック，論理，回路シミュレーション等のVLSI検証ソフトウェアを含めた総合的VLSI設計サポートシステムの確立，および，このようにして設計されたVLSIチップを少量でも低コスト，短期間でファブリケートするSilicon Foundry[5]の確立が強く望まれる。コンパイラマシンに関しては，VLSI化が容易なコンパイラテーブル群の構成法とこれを利用したコンパイルアルゴリズムの研究開発が必要であり，これを実現するVLSIの開発を今後検討していきたいと考えている。

《 謝辞 》

本研究を行なうにあたって多くの有益な御助言を頂き，会話型FHPLシステムの実現に御努力頂いた三菱電機中央研究所システム研究部 杉本明氏に心から感謝いたします。

《 文献 》

1) C.Mead, L.Conway: Introduction to VLSI Systems, Addison-Wesley (1980)
2) M.Foster, H.Kung: Recognize Regular Languages with Programmable Building-Blocks, VLSI81, Academic Press (1981)
3) H.Kung, P.Lehman: Systolic (VLSI) Arrays for Relational Database Operations, Carnegie-Mellon University, Technical Report CMU-CS-80-114
4) J.Batali, A.Hartheimer: The Design Procedure Language Manual, MIT AI Memo No.598 (1980)
5) W.Jansen, D.Fairbrain: The Silicon Foundry: Concepts and Reality, Lambda, Vol.II, No.1 (1981)