解　説

5. 最近の CAD システムの話題

5.4 シリコン・コンパイラ†

平　山　正　治††

1. はじめに

1979年の Design Automation Conference において，"Silicon Compiler" という用語がはじめて使われて以来[1]，この概念は VLSI チップの新しい開発手法として注目を集め，世界各国の大学を中心として盛んに研究が進められている．シリコン・コンパイラは，ソフトウェアにおけるコンパイラが高級言語のソースプログラムを入力して計算機の機械語を生成するのと同様に，VLSI チップに対する高位の機能記述を入力し，これから直接 VLSI のレイアウトパターンを生成しようとするものである．

このようなシリコン・コンパイラは，近い将来発生すると予想される VLSI チップの開発コストの増大に対応する有力な手法として活用され，今後の VLSI の設計開発プロセスを大きく変える可能性を含んだ技術と考えられる．しかし，その実現にあたっては，出力されるコードがレイアウトパターンの2次元情報であるため問題の複雑さがソフトウェアのコンパイラと全く異なり，実現が容易でないという意見や，本来，人間が行うべき高位の設計プロセスまでをも計算機に代行させようということじたい不適当であるなどの批判的な意見もある[2]．現在，発表されているシリコン・コンパイラとよばれるシステムにおいても，特定の機能や構造に制限したものや，手書きのチップに比べて性能上の問題を含んだものが多い．

本稿では，シリコン・コンパイラが提案されてきた背景とその特徴，および，シリコン・コンパイラの要素技術である論理合成と自動レイアウトについて概説する．また，現在発表されているシリコン・コンパイラの代表的システムである Bristle Block, MacPitts, FIRST の3システムについて概要を紹介する．

† Silicon Compiler by Masaharu HIRAYAMA (Central Research Lab., Mitsubishi Electric Corp.).

†† 三菱電機(株)中央研究所

2. シリコン・コンパイラとは

近年の VLSI 技術の急速な進展は，大規模システムの VLSI 化の可能性を与えている反面，設計コストの急激な増大という深刻な問題を生じさせている．たとえば，1970年当時，千トランジスタ程度の IC の開発費は3万ドルほどであったが，1990年頃に実現される百万トランジスタ規模の VLSI では1,700万ドルにも達すると予想されている[3]．

このような非現実的な壁を打ち破るための解決策として，1970年代後半にカリフォルニア工科大学の C. Mead と Xerox PARC の L. Conway はこれまで半導体の開発に携わったことのない人に自ら VLSI のレイアウト設計までも行わせるための簡単化した設計規則による新しい VLSI 設計手法を提案した[4]．この設計手法は当初大学での教育に利用されて急速に広まるとともに，λルール，CIF, Silicon Foundry, マルチプロジェクト・チップ等の新しい用語，概念が形成され，半導体産業全体にも大きなインパクトを与えている．この手法によって多くの人々が自ら VLSI チップを設計できるという可能性は大幅に広がったが，実際のチップの設計には半導体に関するある程度の専門知識を必要とし，このためには教育実習体制や設計サポートシステムの確立が必要である．

一方，VLSI 設計に関する各種の知識やノウハウを計算機の内部に取り込んでおき，設計者の意図を理解して，適切な指示を与えるようなエキスパートシステムによって VLSI 設計をサポートすることが可能である．たとえば，モジュール間結合の機能レベル記述から VLSI のレイアウト設計までの各設計ステージを統一的にサポートするようなエキスパートシステムの研究が行われている[5]．

また，エキスパートシステムのような知的設計システムのひとつとして，機能設計，論理設計，回路設計，レイアウト設計の各設計ステージを人手によってブ

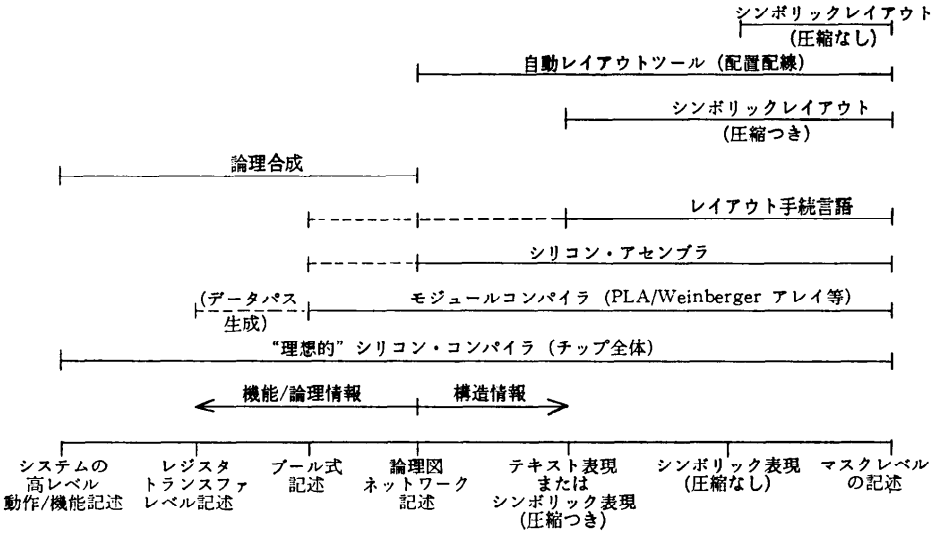


図-1 半導体設計手法の位置付け[2]

レークダウンしていくのではなく，VLSI チップの高位レベルの機能記述から直接レイアウト情報を自動的に出力しようというシリコン・コンパイラが提案された．シリコン・コンパイラ，エキスパートシステムのいずれも内部に保持された VLSI 設計のための知識を利用して，最終的にレイアウト情報を出力するものではあるが以下の点で異なった設計アプローチと考えられる．エキスパートシステムでは数百にもおよぶ詳細で複雑な知識を利用して広範な領域の VLSI 開発を目的とし，設計に必要な知識，ノウハウを提示して設計者の注意を促し，判断を仰ぐかたちで設計のサポートを行うのに対して，シリコン・コンパイラでは比較的少数の規則が適用される制限された領域の中で，より機械的に一括してレイアウトパターンの自動生成を行おうとすることに特徴がある．

ただし，現時点でこの用語に対して明確な定義がされている状況ではなく，最近の学会等においても，シリコン・アセンブラ，モジュール・コンパイラ等の類似した用語が使われ，また，シリコン・コンパイラという用語も各人が異なった意味で使用しているのが現状である．各種の VLSI 設計手法の位置づけを設計レベルとの対比で示してみると，**図-1** のように，理想的な意味でのシリコン・コンパイラとは，設計プロセスの最も上位レベルである機能レベルの記述から実際のチップとして製造可能な半導体のレイアウト情報をダイレクトに変換するシステムと考えられる．

3. シリコン・コンパイラの要素技術

シリコン・コンパイラにおける内部の処理を分解してみると，機能レベルの記述を入力して VLSI の製造プロセスに無関係な論理合成[6]を行う部分と，この結果から各製造プロセスに対応したマスクのレイアウト情報を出力する自動レイアウト[7]という2つの処理に分けられる．したがって，論理回路図，MOS トランジスタ回路図等の適当な中間レベルの記述を設定し，これらの中間レベルを経由した論理合成システムと自動レイアウトシステムを組み合わせることによってシリコン・コンパイラを実現することが可能である．このような2段階の変換過程を含むシステムのほうが製造プロセスに依存する部分が分離されているため，急速な VLSI プロセス技術の発展に容易に対応でき，長期間にわたって有効なシリコン・コンパイラが実現できるという特徴がある．

以下ではシリコン・コンパイラの要素技術である論理合成と自動レイアウトについて，最近の研究開発動向とシリコン・コンパイラを実現するうえでの技術的影響について述べる．

3.1 論 理 合 成

論理合成に関する研究は古く，1960年代から IC 部品 (SSI) によるシステムの自動合成を狙って多くの研究開発がなされてきた．これらの概要は文献8),9) によくまとめられており，また，本特集の中でも述べられているので参照されたい．

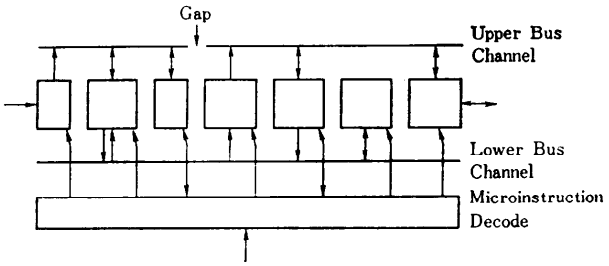
最近の研究成果としては，従来からこの分野において盛んに研究を行っているカーネギーメロン大学における知識ベースシステムによってカスタム VLSI のアルゴリズムレベルの表現からチップの実現技術に依存しないレジスタトランスファレベルの構造を自動生成するシステムが挙げられる[10]．このシステムでは OPS 5 とよぶ汎用知識ベース実現システムを利用し，これに 130 個ほどの VLSI 設計技術者の専門知識を植え付けることによって論理合成エキスパートシステムを実現している．また，IBM でもランダムロジック部分をゲートアレイで実現することを対象として，設計の各ステージでの最適化を会話型で行え，かつ，ここで適用される変換が局所的にしか影響を及ぼさないことによって，VLSI のような大規模回路に対しても実現可能な解を与えることを狙った会話型論理合成システムの開発を行っている[11]．

このように，最近の論理合成の研究動向としては，VLSI チップのような大規模回路を実現の対象とし，設計知識，ノウハウをシステムの内部に取り込んだ知識ベースシステムの形態で，会話型処理を指向したものが多くみられる．この点では，特定のターゲットアーキテクチャに固定し，バッチ処理的形態をとっている現在のシリコン・コンパイラのアプローチとは異なっているが，今後のシリコン・コンパイラの研究として取り組んでいくひとつの方向を示していると考えられる．

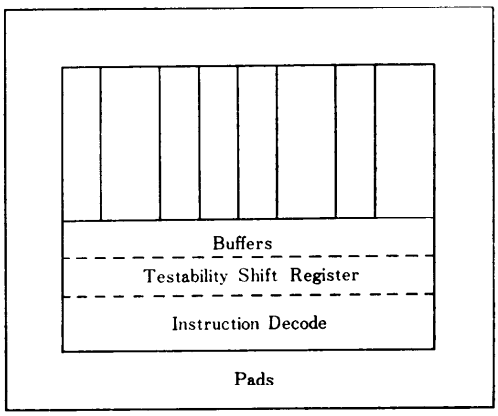
3.2 自動レイアウト

VLSI の設計において設計者が 2 次元的なレイアウト図形を直接入力するのは非常に時間がかかり，大変な労力を要する作業であるため，シンボリックな表現を許したり，規則的な構造を使って設計を容易にする自動レイアウトシステムは従来からよく研究されている．特に，ゲートアレイを利用して VLSI チップを実現する場合には，ユーザの入力する論理図から配置，配線を行って VLSI チップを自動的に生成するシステムが多くの企業で実用化されている．また，ランダムロジックを任意の深さの NOR 回路で実現する Weinberger アレイ[12]や，任意の信号に対しての AND-OR 回路の組合せで実現する PLA (Programmable Logic Array) の場合も，ブール式等の簡単な記述からマスクパターンを自動生成するプログラムが広く利用されている．

最近の自動レイアウトの動向としては，ALU, PLA, ROM 等のある程度大きな機能ブロックを自動生成しようとするセルコンパイラやモジュールコンパイラと呼ばれるシステムが発表されている[13]．これらのシステムによってある程度大きな機能ブロックが容易に実現できれば，後の処理としてこれらのブロックを適当に配置，配線し，全体としての制御構造を生成する処理を付加することによって，シリコン・コンパイラへと発展させることが可能である．ただし，これらの機能ブロックは一般に任意の大きさ，形状をしていると考えられ，このような複数ブロック間の最適配置，配線を行うソフトウェアの開発が必要である．また，機能ブロックが大きくなるに従って，内部に制御回路を含み，自立的に動作する形式になっていくため，チップ全体を簡単なクロックに同期して動作させることが困難になり，複数の非同期要素から構成されるチップの結合方式と制御方式などを検討していく必要がある．



(a) Bristle Block における論理構造



(b) フロアプラン

図-2[14]

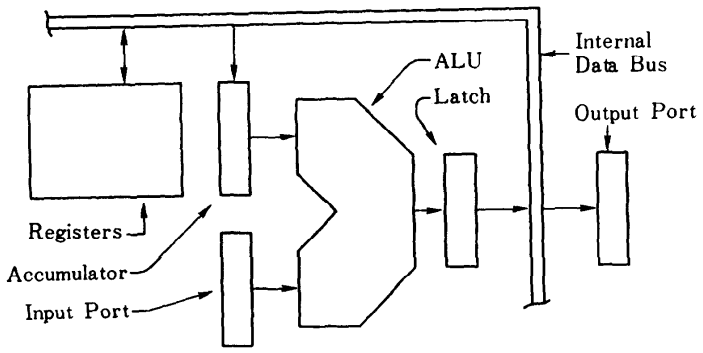
4. シリコン・コンパイラの開発例

4.1 Bristle Block[13),14)]

カリフォルニア工科大学では早くから VLSI の新しい設計手法に関する研究が行われており，1976年からC. Mead によって開始された OM (Our Machine) プロジェクトの中で LSI 化プロセッサシステムの研究開発が行われた．ここで設計された OM 2 データパスチップの経験から，プロトタイプシステムの実現を簡単化，迅速化するためのソフトウェアツールとして Bristle Block と呼ばれる最初のシリコン・コンパイラが1978年に開発された．

Bristle Block の設計思想は，対象とするシステムの動作を特定の言語によって記述したものから実際に製造可能なチップのマスク情報を直接生成することを狙っているが，ターゲットアーキテクチャとして外部から入力されるマイクロ命令によって動作するマイクロプロセッサ形式のデータパスチップだけを実現の対象とし，そのアーキテクチャ，動作タイミング，フロアプランをすべて固定している．

すなわち，ターゲットアーキテクチャは図-2(a)に示すように，メモリ，シフタ，ALU 等の記憶，演算要素が2本の共通バスによって通信しあう形式をしており，外部から入力されるマイクロ命令をデコードすることによって得られる制御信号に応じて動作する構成を規定している．チップ全体の動作タイミングも明確に決められており，重なり区間の存在しない2相クロック $\phi1$, $\phi2$ に同期して，$\phi1$ の区間では2本のバスを利用したデータ転送，$\phi2$ の区間では各記憶，演算要素におけるデータ処理が実行される．また，このような形式のデータパスチップのフロアプランも，図-2(b)に示すように，中央部には各記憶，演算要素が並び，その下側にはマイクロ命



(a) ブロック図

```
NAME SAMPLE 8;

    FIELD
        REG_SELECT<1:3>, ENABLES<4:6>, ALU_OP<7:9>;
    INPUT_PORT INPUT
        LOAD: ALWAYS,
        REGISTER: [WRITE_LOWER: ALWAYS];

    MACRO ADDRESS (ADDR)
  % REGISTER R?ADDR? OPTIONS:
        [READ_UPPER:  REG_SELECT=?ADDR? AND ENABLES=XX1,
         WRITE_UPPER: REG_SELECT=?ADDR? AND ENABLES=XX0,
         REFRESH: ALWAYS]; %
    /* ADDRESS (000) */
    /* ADDRESS (001) */
        ...
    /* ADDRESS (111) */

    PRECHARGE_BOTH PCHG;

    ALU ALU
        INPUT_A: [READ_LOWER: ALWAYS],
        INPUT_B: [READ_UPPER: ENABLES=1XX,
                  REFRESH: ALWAYS],
        OUTPUT_1: [WRITE_UPPER: ENABLES=XX1],
        DECODE: ALU_OP
            0 => OR
            1 => INCREMENT_A
            2 => AND
            3 => SUBTRACT
            4 => XOR
            5 => ADD
            6 => ZERO
            7 => DECREMENT_A;

    OUTPUT_PORT OUTPUT
        REGISTER; [READ_UPPER: ENABLES=X1X,
                   REFRESH: ALWAYS],
        DRIVE: ALWAYS;
END
```

(b) 入力形式

図-3 Bristle Block の例[14)]

令から制御信号を生成するためのデコーダと制御用バッファが置かれ，さらにこれらの外形に沿って入出力用の IO パドが配置される形状に固定されている．

Bristle Block ではターゲットアーキテクチャをマイクロプロセッサシステムのデータパスチップに固定していることから，下記の3種のデータ，

① データパスの幅（ビット長）

② マイクロ命令の構造（フィールド構成）

③ 記憶，演算要素の記述とそのパラメータの指定

をコンパイラに入力することによって，チップを構成する各部のレイアウトパターンが生成される．**図-3**（a）に簡単なデータパスチップのブロック図と，（b）にその入力形式を示す．

このシステムを使って実際のチップの自動生成を試みた結果，$6.20\times6.25\,mm^2$ の大きさをもつ16ビット幅のデータパスチップは7分，また，$4.34\times4.95\,mm^2$ のシーケンスコントローラは4分のコンパイル時間で済み，このとき生成されたチップの大きさは人手によってレイアウトされたものに比べて10%程度の増加でしかなかったと報告されている．

4.2 MacPitts[15),16)]

マサチューセッツ工科大学の Lincoln Laboratory で開発されたシリコン・コンパイラ MacPitts は高位言語によるアルゴリズムレベルの記述から VLSI チップを生成しようとするシステムである．MacPitts によって生成されるチップは通常のマイクロプロセッサと同様にデータパス部と制御部からなるターゲットアーキテクチャをもち，固定的なフロアプランのうえに実現されるが，対象とするチップの記述はより機能的であり，かつ，柔軟なバス構造や並列プロセスが容易に実現できる等の特徴をもったシステムである．例えば，$\sqrt{a^2+b^2}$ の近似値を求めるのに**図-4**（a）に示すアルゴリズムを使うとすれば，これを実現するチップは**図-4**（b）に示すプログラムのように記述される．この記述例のように MacPitts では Lisp の PROG 形式と同様の表現形式によって，レジスタトランスファレベルの動作が手続き的に記述されている．

MacPitts によって生成されるチップのデータパス部は演算要素とレジスタ，および必要に応じてこれらに対する入力データを切り換えるマルチプレクサとが複数のデータ転送バスによって結合された構成になっているため，複数の演算ユニットが異なったソースデータに対して同時に演算を行うことが可能である．また，データパス部のレイアウトにおいては，“organelle” と呼ばれる1ビット単位の演算，記憶要素間を専用バスとマルチプレクサで接続したビットスライスのデータパスを構成し，これを必要なビット数だけ並べて任意の語長のシステムを実現する方式を採っている（**図-5** 参照）．

また，生成されるチップの制御部には，データパス部の各要素の動作を制御する信号を生成するために，Weinberger アレイによって実現される有限状態マシンモデルが適用されている．ここで生成される制御信号は入力された動作記述を解析することによって得られるものであるが，通常の Lisp 処理系とは異なって，複数の COND の条件式を同時に評価し，同時実

〔絶対値の近似計算〕

$$\sqrt{a^2+b^2}\approx\max\left(g,\ \frac{7}{8}g+\frac{1}{2}l\right)$$

ここで　$g=\max(|a|,|b|),\ l=\min(|a|,|b|)$

（a）アルゴリズム

```
(program genmag4
    (def msb constant 3)
    (def a port input (2 3 4 5))
    (def b port input (6 7 8 9))
    (def res port output (10 11 12 13))
    (def aab-g register)
    (def bab-l-sqs register)
        ...
    (process compmag
loop
; set aab & bab
        (cond ((bit msb a) (setq aab-g (- 0 a)))
              (t (setq aab-g a)))
        (cond ((bit msb b) (setq bab-l-sqs (- 0 b)))
              (t (setq bab-l-sqs b)))
;set g & l
        (cond ((not (unsigned -> aab-g bab-l-sqs))
               (setq aab-g bab-l-sqs)
               (setq bab-l-sqs aab-g)))

; sqs := 7/8g + 1/2l
        (setq bab-l-sqs (+ (- aab-g (3>>f aab-g))
                           (>> bab-l-sqs)))
; max of sqs & g
        (cond ((unsigned -> aab-g bab-l-sqs)
               (setq res aab-g))
              (t (setq res bab-l-sqs)))
        (go loop)))
```

（b）入力形式

図-4 MacPitts の例

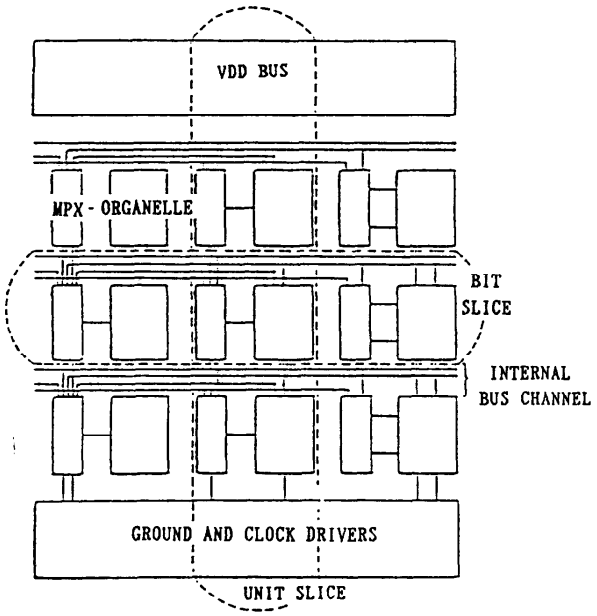


図-5 MacPitts におけるデータパス部の構成[16]

表-1 MacPitts とスタンダードセル方式の比較[17]

チップ名	スタンダードセル方式による設計とレイアウト	MacPitts による設計とデータパス部のレイアウト，スタンダードセル方式による制御部のレイアウト	MacPitts による設計とレイアウト
HOB 3	8.09	39.37 (4.99)	60.36 (17.22)
DTMF 3	23.65	171.41 (16.74)	318.02 (99.32)

単位：mm²，（ ）内は制御部の大きさ

行可能な処理を並行して実行させるための制御信号が生成される．

MacPitts を利用して信号処理チップを自動生成した場合と従来のスタンダードセル方式のレイアウトによった場合とを比較した結果，表-1 に示すように，シリコン・コンパイラによる自動生成では10倍前後もの大きな面積を要している[17]．ただし，制御部のランダムロジックの部分をスタンダードセル方式でレイアウトすればこの面積増加を約半分にすることができる．ここで示された結果は，Bristle Block 場合に面積が10%しか増加しなかったのに比べて大きな相違を示しているが，これは MacPitts がそれだけ汎用的な入力の記述を許しているためと思われる．一方，ここで示した DTMF 3 チップはスタンダードセル方式の設計で6人月を要したのに対して，MacPitts ではたったの8時間で設計を完了できたとされている．

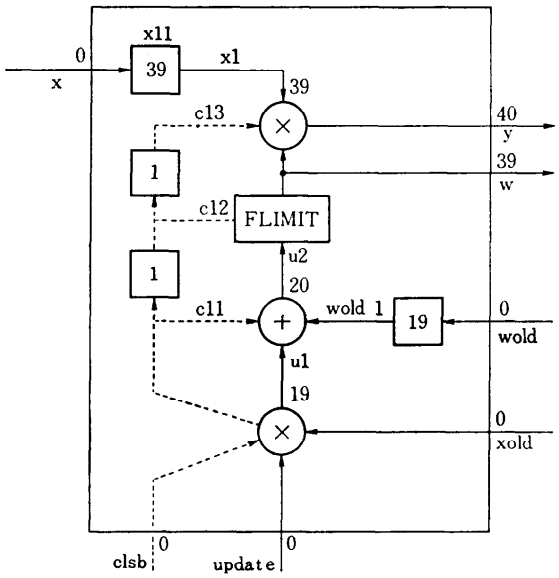
4.3 FIRST[18),19)]

エジンバラ大学においては，素人が，極短期間で，正しく動作する信号処理用のカスタム VLSI チップを開発する目的で，FIRST (Fast Implementation of Real-time Signal Transforms) と呼ぶシリコン・コンパイラを開発し，これを用いていくつかの信号処理用チップの開発を行っている．このシステムにおいては VLSI の応用分野を信号処理に限定することによって，処理の対象となるデータ信号は固定小数点形式でビットシリアルに入出力され，また，このような信号データが内部に記憶要素を含まない信号処理演算要素間を順次流れていく中で所定の処理が行われる形式のチップだけを対象としている．

このシステムを利用して信号処理チップを設計しようとする人は，所定の機能を実現するために必要な基本演算要素をライブラリとして登録されているものの中から選択し，これら演算要素間を流れる信号のデータフローを入力言語によって記述する．このとき，チップの外部からの入力信号を含めて，演算要素に入ってくる複数の信号はすべて同時に入力されるものとしているため，適当な長さの遅延要素をチップの各所に挿入して演算要素に到着する信号のタイミングを取ることが必要である．たとえば，X＝A＊B+C の演算を行うとき，A，B，Cとも同じタイミングでデータが入力されるため，Cの信号に対してA＊Bの乗算時間と同じ長さの遅延要素を挿入して，加算器へのデータ入力を遅らせる必要がある．

シリコン・コンパイラ FIRST を使った信号処理チップの開発例として，LMS 適応フィルタに使われる LMS 演算モジュールのブロック図と入力形式を図-6 (a)，(b)に示す．この例では，記述の最初の行でこの演算要素の名前，パラメータ，入出力される制御信号，データ信号が宣言され，続く2行でこのチップの内部で使われる制御およびデータ信号が宣言される．以下の行はこのチップの中で使われる基本演算要素の宣言と，これらに対して入出力される信号をパラメータとして指定する記述によって図-6(a)に示される機能ブロック図のデータフローが表現されている．

FIRST コンパイラが用意している基本演算要素の



(a) ブロック図

```
OPERATOR LMSPART1 [sw1] (clsb) x,update,xold,wold -> y,w

    SIGNAL x1,u1,u2,wold1,x11
    CONTROL c11,c12,c13

    CBITDELAY [1] (c11 -> c12)
    CBITDELAY [1] (c12 -> c13)

    BITDELAY [19] x -> x11
    BITDELAY [20] x11 -> x1
    BITDELAY [19] wold -> wold1

    MULTIPLY [1,12,0,0] (c13 -> NC) w,x1 -> y,NC
    FLIMIT [sw1,1,0] (c12) u2 -> w
    ADD [1,0,0,0] (c11) u1,wold1,GND -> u2,NC
    MULTIPLY [1,12,0,0] (clsb -> c11) update,xold -> u1,NC

END
```

(b) 入力形式

図-6 FIRST の例

ライブラリとしては乗算，加算，ソート等に加えて，いくつかの遅延要素があり，また，これらを組み合わせたフィルタやバタフライ演算等のマクロ機能がサブルーチン的に利用できる．これらは5μm ルールのnMOS プロセスで設計されており，最大 8MHz のクロックで動作することが可能である．FIRST によって生成されるチップのフロアプランは，図-7 に示すように，チップの中央部分に全配線領域を集中し，これに対して各演算要素を背中合わせに配置する構成を採っている．信号処理の場合，演算要素に入出力される各信号に対して1本の配線ですみ，また，各演算要素の配置も単に入力に記述される順序に配置するだけであるため FIRST コンパイラにおける配置，配線処理は比較的単純である．

5. おわりに

本稿ではシリコン・コンパイラに関する概要と現在発表されているシリコン・コンパイラの代表的な3システムについて述べた．ここで示したシステムはいずれもターゲットアーキテクチャや対象領域を固定しており，汎用的な VLSI チップの機能記述を許すような理想的シリコン・コンパイラとは言い難いが，実用的なシリコン・コンパイラの開発を行ったという点で高く評価できる．

シリコン・コンパイラはまだ研究段階にある技術分野であるが，これによって実際の VLSI 開発をサポートしようとする企業も現れている．英国エジンバラの Lattice Logic Ltd. は MODEL とよぶ設計記述言語による階層的な回路記述から自動的にゲートアレイへの展開を行う自動レイアウトツールを開発しているし[20]，米国カリフォルニアの VLSI Technology Inc. は駆動能力，スピード等がパラメータで自由に変えられる前述のセルレイアウトコンパイラによって，ユーザの要求に最適な VLSI チップの実現をサポートしている[13]．同じくカリフォルニアの Silicon Compiler Inc. はシリコン・コンパイラによってローカルエリアネットワークの制御用チップの開発を行ったり，DEC 社のスーパミニコン VAX-11 の VLSI 化を7カ月間で行ったと発表している[21]．

シリコン・コンパイラによって，面積，速度の観点から最高の性能を追及した VLSI チップを実現しようとするのは将来的にも困難と考えられるが，少量，多品種のカスタム VLSI の開発では短期間で正しく動作するチップを実現することが最も重要であり，この目的のために有効なシリコン・コンパイラの開発が

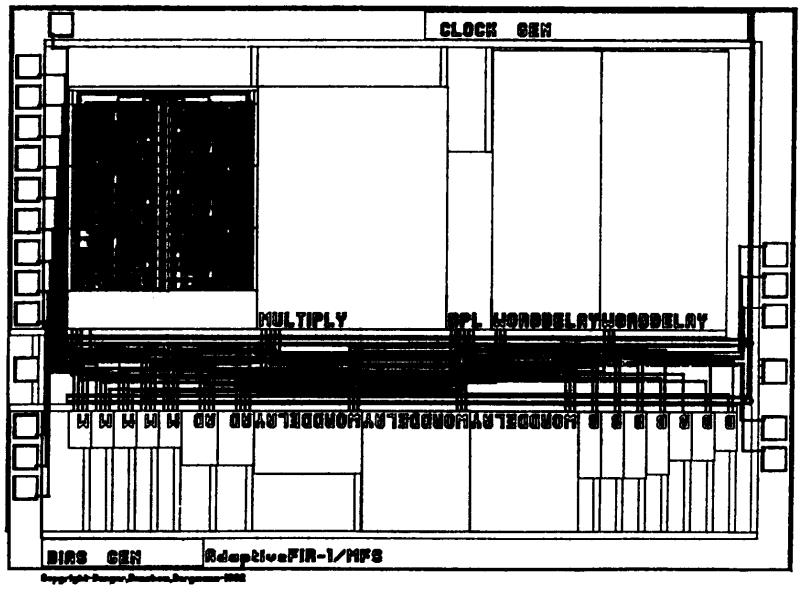


図-7 FIRST におけるフロアプラン[18]

ますます重要になってくると考えられる.

参 考 文 献

1) Johannsen, D.: Bristle Blocks: A Silicon Compiler, Proc. of the 16th Design Automation Conference (1979).
2) Werner, J.: The Silicon Compiler: Panacea, Wishful Thinking, or Old Hat?, VLSI Design (Sep./Oct. 1982).
3) LSI Opportunities, Anderson/Bogert, CA. (1981).
4) Mead, C. and Conway, L.: Introduction to VLSI Systems, Addison-Wesley, Mass. (1980).
5) Brown, H. et al.: Palladio: An Exploratory Environment for Circuit Design, Computer, Vol. 16, No. 12 (Dec. 1983).
6) Werner, J.: Progress Toward the "Ideal" Silicon Compiler, Part 1: The Front End, VLSI Design (Sep. 1983).
7) Werner, J.: Progress Toward the "Ideal" Silicon Compiler, Part 2: The Layout Problem, VLSI Design (Oct. 1983).
8) Thomas, D. E.: The Automatic Synthesis of Digital Systems, Proc. IEEE, Vol. 69, No. 10 (Oct. 1981).
9) Shiva, S. G.: Automatic Hardware Synthesis, Proc. IEEE, Vol. 71, No. 1 (Jan. 1983).
10) Kowalski, T. J. and Thomas, D. E.: The VLSI Design Automation Assistant: Prototype System, Proc. of the 20th Design Automation Conference (1983).
11) Darringer, J. A. et al.: Logic Synthesis Through Local Transformations, IBM J. Res. Dev., Vol. 25. No. 4 (July 1981).
12) Weinberger, A.: Large Scale Integration of MOS Complex Logic: A Layout Method, IEEE J. of Solid-State Circuits, Vol. SC-2, No. 4 (Dec. 1967).
13) Nance, S. et al.: Cell-Layout Compilers Simplify Custom-IC Design, EDN (Sep. 15, 1983).
14) Johannsen, D. L.: Silicon Compilation, Tech. Rep. 4530, California Institute of Technology (1981).
15) Siskind, J. M. et al.: Generating Custom High Performance VLSI Designs from Succinct Algorithmic Descriptions, MIT Conference on Advanced Research in VLSI (1982).
16) Southard, J. R.: MacPitts: An Approach to Silicon Compilation, Computer, Vol. 16, No. 12 (Dec. 1983).
17) Fox, J. R.: The MacPitts Silicon Compiler: A View from the Telecommunications Industry, VLSI Design (May/June 1983).
18) Denyer, P. B. et al.: A Silicon Compiler for VLSI Signal Processors, ESSCIRC '82 Digest of Technical Papers (Sep. 1982).
19) Smith, S. G.: Case Studies in System Design with a Silicon Compiler, Report No. ISG/83/11, University of Edinburgh (Jan. 1983).
20) Designing with Gate Array, Lattice Logic Ltd., Edinburgh (1982).
21) Louie, G. et al.: The Micro VAX I Data-Path Chip, VLSI Design (Dec. 1983).

(昭和59年5月31日受付)