ICOT Technical Memorandum: TM-0111

TM-0111

シリコン・コンパイラ

丸 山 文 宏 （富士通）

May, 1985

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

4・6　シリコン・コンパイラ

本節では，ＬＳＩ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，設計をサポートするエキスパートシステムの技術について述べる。事例として，第五世代コンピュータプロジェクトの一環として研究が行われている，論理設計支援システム［　］を取り上げる。

4・6・1　システム構成

このシステムの構成を図4・6・1に示す。本システムでは，並列性を表現し易いように設計されたプログラミング言語occam［　］で仕様となるハードウェアの動作アルゴリズムを記述する。この段階では，ユーザはアルゴリズムとそれを実現するハードウェアとの対応を意識する必要はない。ハードウェア化サブシス

ムは，このアルゴリズムからできる限り並列性を抽き出し，ハードウェアと対応付ける。この際，知識ベースに格納されたハードウェア機能設計に関する知識が利用される。ステートマシン最適化サブシステムによって最適化された情報は，ハードウェアの振舞いの記述であり，レジスタトランスファレベルのハードウェア記述言語DDLによるステートマシンとして表現される。このレベルでは，ハードウェアの動作については明確になっているが，それを実現する回路の構造については何も規定されていない。以上が機能設計の過程である。

トランスレータサブシステムは，DDL記述から回路設計に必要な2種類の情報を抽出・整理する。すなわち，データパス情報及びステートマシンに関するステート遷移情報である。データパス情報はデータパス設計サブシステムに渡され，

各ハードウェア要素（フリップフロップ，機能ブロック等）のまわりの回路の設計に利用される。制御回路設計サブシステムは，ステート遷移情報に基づいて，ステートを実現するフリップフロップとそのまわりの制御回路を設計することにより，ステートマシンを実現する。入出力ピン，機能ブロック，フリップフロップは，基本セル割当サブシステムにより，セル・ライブラリに登録された基本セルを組み合せて実現される。一方，組合せ回路は，任意の論理関数を実現できる複合セルによって実現される。実現すべき論理関数は，関数分割サブシステムによって遅延，ファンアウト等を考慮して分割され，それぞれが複合セル設計サブシステムによって複合セルのレイアウト・パタンを与えられる。基本セル及び複合セルに関する情報が回路最適化サブシステムに渡され，回路の冗長な部分が除

表され，最終的にCMOS回路（基本セル，複合セル及びその接続関係）が設計される。以上の回路設計過程までが本システムのカバーする範囲である。この後，更に，配置・配線等の実装設計を経て製造工程へと進む。

このようなシステムにおいては，設計者のノウハウを有効に利用するとともに，従来のCADシステムによって支援されてきたアルゴリズムに基づく処理もうまく組み入れることが重要である。本システムでは，プロダクションシステムのテクノロジーを導入し，論理型言語Prolog上に知識ベースとアルゴリズミックな処理を統合している。

4・6・2 知識ベースと推論

ハードウェア化サブシステムは，例えば，図4・6・2に示す仕様（動作アルゴリ

スム）を与えられて，図4・6のようなハードウェアの機能記述を生成する。設計者が設計を行う場合，全体の仕様を睨みながら段々と具体的な回路を明らかにしていく。このように徐々に設計が出来上がっていく（または詳細化されていく）過程は，計算機においては，その実行に伴う副作用が環境を変化させていくことにより実現できる。本システムでは，Prologのassertを用いて次々と事実（fact）を書き込んでいくことにより実現されている。このために40個程の述語（predicate）が用意されている。図4・6.4にルールの例を示す。これは，occamの変数を1ビットのレジスタで実現する場合のルールである。このような前向き推論の他に，設計を行ううえで様々な条件をチェックする必要がある。このためには後向き推論がふさわしい。後向き推論はPrologの実行メカニズムによ

り自然に実現されている。図4.6に示すルールは，二つのオペレーションがハードウェアで同時に実行できる（compatible）かどうかをチェックするルールである。

設計においては，設計対象間の関係を把握することが重要である。本システムでは，論理型言語の述語によりこういった関係を自然に表現できる，ことを生かした知識表現を行っている。

4・6・3　アルゴリズムに基づく処理

トランスレータサブシステムは，DDL記述を解析し，各レジスタ，ターミナル，ステート毎にそれぞれに対するオペレーションをまとめ，フレーム型の情報を生成する。例えば，あるレジスタに関して，そのレジスタに値をセットするオ

ペレーション（レジスタ転送オペレーション）をすべて集め，そのソース，実行条件毎にまとめた情報をそのレジスタのビット数等の情報とともにフレームに格納する（図4・6・6）。Prologにとって言語の解析は得意な分野であり，効率的にトランスレータが実現されている。

フリップフロップ，機能ブロックに関するフレームは，入力されるソース・条件に関する情報をすべて持っているから，その入力部分の回路を設計することは容易である。データパス設計サブシステムは，フレーム情報から各レジスタ，機能ブロックのまわりの組合せ回路を設計する（図4・6・7）。正確に言えば，各レジスタ，機能ブロックのソース，実行条件となっている（広義の）論理式を操作し，複合セルで実現すべき論理関数を決定する。

複合セルは，任意の組合せ回路を実現する機能を持つ．ところが，図4・6・8に示されているように，等価な回路であっても，入力端子に対応するゲートの並べ方の順序によって，左のように隙間（セパレーション）を多く取らなければならなかったり，右のようにセパレーションのないコンパクトなレイアウトが可能になったりする．本システムでは，この複合セルのレイアウトにかかる理論的なヒューリスティックアルゴリズムを導入し，Prologによって実現されている．

以上見て来たように，設計対象間の関係を述語で自然に表現する枠組みによる知識ベースとアルゴリズムに基づく処理が，Prolog上に統合されている．

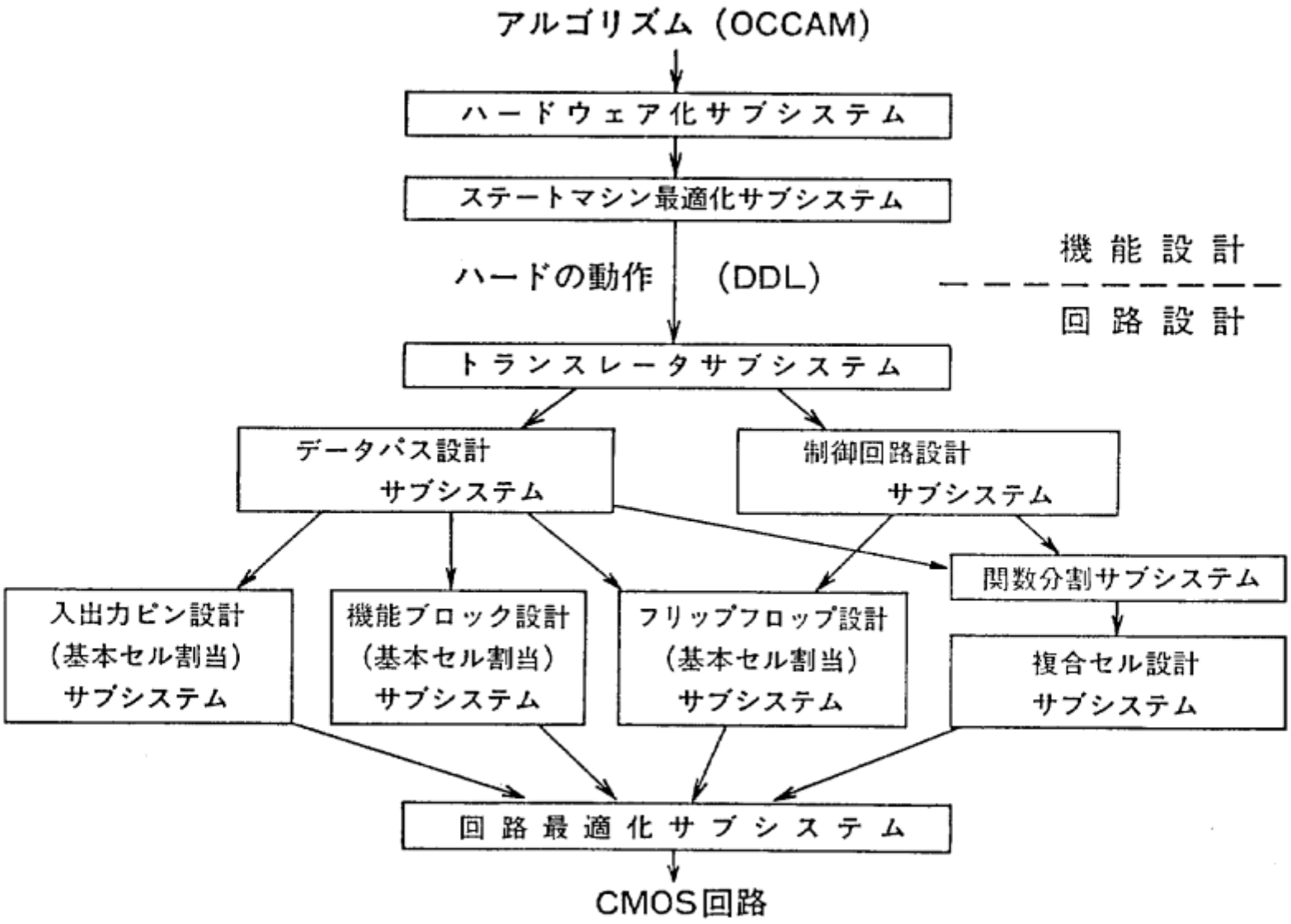

アルゴリズム（OCCAM）
ハードウェア化サブシステム
ステートマシン最適化サブシステム
ハードの動作
（DDL）
機能設計
回路設計
トランスレータサブシステム
データパス設計
サブシステム
制御回路設計
サブシステム
関数分割サブシステム
入出力ピン設計
（基本セル割当）
サブシステム
機能ブロック設計
（基本セル割当）
サブシステム
フリップフロップ設計
（基本セル割当）
サブシステム
複合セル設計
サブシステム
回路最適化サブシステム
CMOS回路

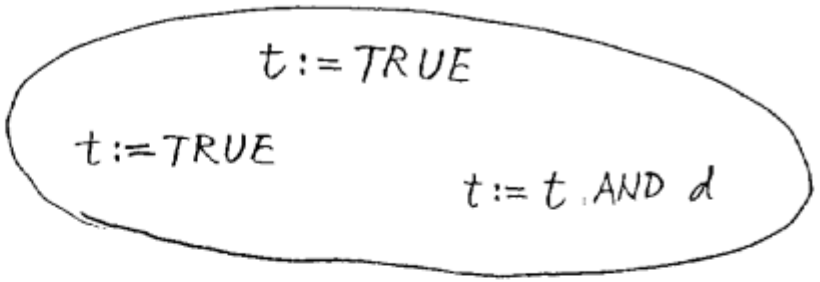
図4.4.1　システム構成

```
CHAN pattern[6]:
CHAN string [6]:
CHAN data   [5]:
CHAN end    [6]:
CHAN wild   [6]:
CHAN result [6]:
PROC comp(CHAN pin,sin,pout,sout,dout) =
  VAR p,s:
  SEQ
    PAR
      p:=0
      s:=0
    WHILE TRUE
      SEQ
        PAR
          pout ! p
          sout ! s
        PAR
          pin ? p
          sin ? s
        dout ! p=s:
PROC acc(CHAN xin,lin,rin,din,xout,lout,rout) =
  VAR d,x,l,r,t:
  SEQ
    PAR
      x:=FALSE
      l:=FALSE
      r:=FALSE
      t:=FALSE
    WHILE TRUE
      SEQ
        PAR
          xout ! x
          lout ! l
          rout ! r
        PAR
          din ? d
          xin ? x
          lin ? l
          rin ? r
        IF
          l=TRUE
            SEQ
              r:=t
              t:=TRUE
          l=FALSE
            t:=t^(x d):
PAR i=[1 FOR 5]
  PAR
    comp(pattern[i-1],string[5-i],pattern[i],
         string[6-i],data[i-1])
    acc(wild[i-1],end[i-1],result[5-i],data[i-1],
        wild[i],end[i],result[6-i])
```

図4.5.2 occam による仕様（動作アルゴリズム）

```
<SYSTEM> pm.
  <TIME> clk.
  <ENTRANCE> pin(8), sin(8), xin, lin, rin.
  <TERMINAL> pout(8), sout(8), dout, xout, lout,
             rout, send1.
  <AUTOMATON> comp: clk:
    <REGISTER> p(8), s(8).
    <STATES>
      init: p<-0, s<-0, ->idle.
      idle: pout=p, sout=s,
            p<-pin, s<-sin, ->state1.
      state1: send1=1, dout=(p:=s), ->idle.
    <END>.
  <END>comp.
  <AUTOMATON> acc: clk:
    <REGISTER> d, x, l, r, t.
    <STATES>
      init: x<-0, l<-0, r<-0, t<-1, ->idle.
      idle: send1: xout=x, lout=l, rout=r,
                   x<-xin, l<-lin, r<-rin,
                   d<-dout, ->state1.
      state1: |* l *| r<-t, t<-1
                    ; t<-(t&(x|d))., ->idle.
    <END>.
  <END> acc.
<END> pm.
```

図4・6・3 DDLによる機能記述



変数tに対するオペレーション

```
implement_variable(Var, ···):-
  sources_by_assignment(Var, Assign_source_list),
  truth_value(Assign_source_list),
  sources_through_channel(Var, Input_source_list),
  truth_value(Input_source_list),
        .  .  .
  assert(implementation(Var, register, 1, ···)).
```

<REGISTER>　t(1), ···　　1ビットレジスタ t

図4・6・4　前向き推論のルール

SEQ
　r := t
　t := TRUE　　ここ⇒ 逐次

```
compatible(Operation1, Operation2) :-
    store_operation(Operation1, Var1, Source1, ···),
    store_operation(Operation2, Var2, Source2, ···),
    followed_by(Operation1, Operation2),
    Var1 \== Var2,
    implementation(Var1, register, ···),
    implementation(Var2, register, ···),
    not(referred_to(Var1, Source2)).
```

r←t, t←1　　ここ⇒ 並列

図4.[illegible]　後向き推論のルール

register : t

bits : [0, 0]

automaton : acc

source : [1, 1, t & d]

condition : [init, state1 & l, state1 & ¬l]

図4.6.6 フレーム情報

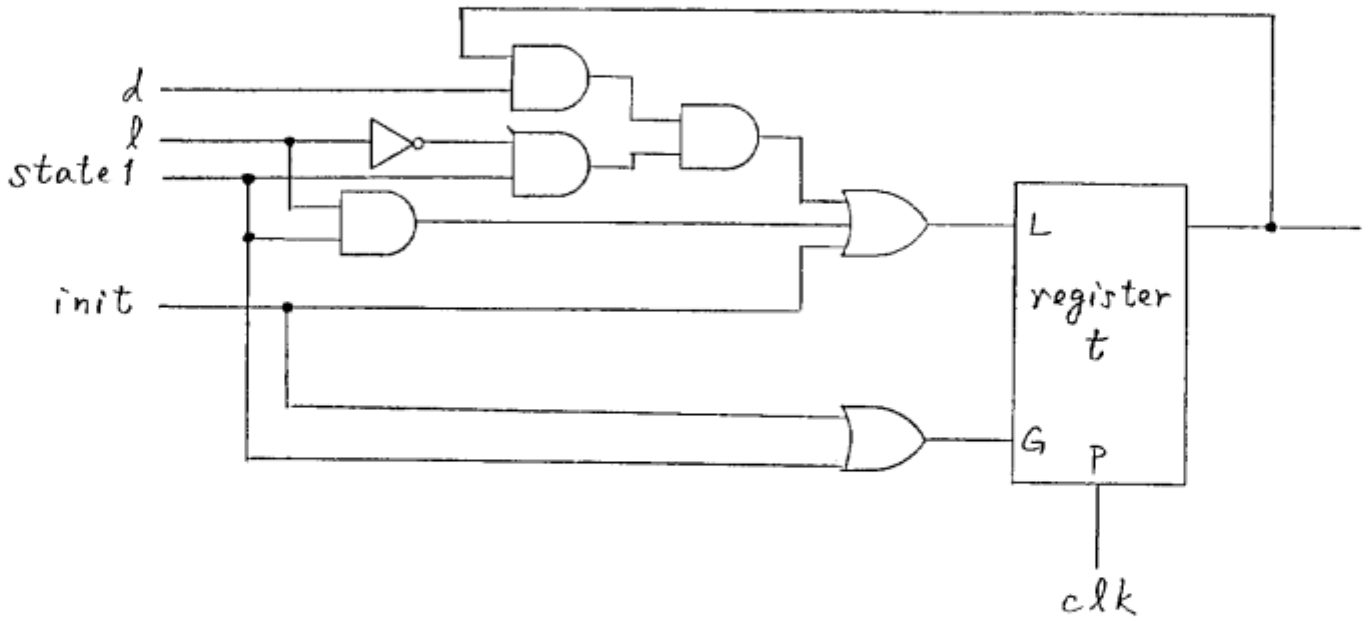


図4・6・7　レジスタのまわりの回路

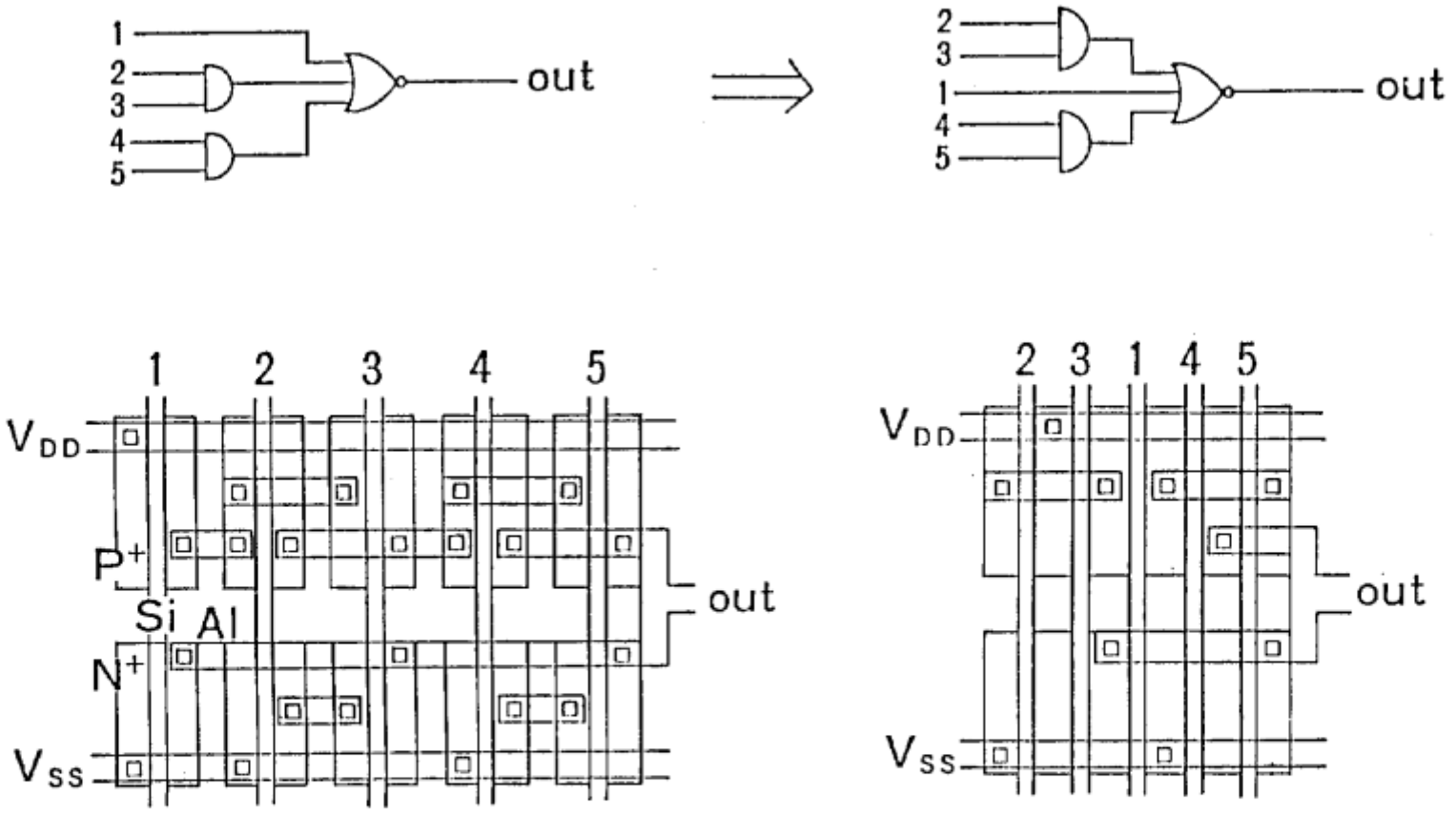


図4・6・5　複合セルのレイアウト