マルチメディア通信と分散処理　66－9
（１９９４．　７．　７）

LOTOS風言語で書かれた同期式順序回路の要求仕様記述と回路自動合成

黄 莉　安本 慶一　北道 淳司　東野 輝夫　谷口 健一

大阪大学 基礎工学部 情報工学科

あらまし　本稿では，まず，LOTOS風言語LOTOS/HDを定義し，次に，LOTOS/HD仕様からハードウェア回路を自動的に合成する手法を提案する．生成する回路として，同期式順序回路を考える．提案する手法では，設計者がまずLOTOS/HDで同期式順序回路Sの仕様を記述する．この抽象レベルでは，設計者は計算したい内容及び組合せ論理回路で実現可能な関数を用いてそれらの計算過程を記述する．次に，Sのデータ依存関係とイベントの実行順序より，Sを実現する同期式順序回路Cを構成する．Cの変数と関数がそれぞれ同期式順序回路のレジスタと組合せ論理回路に割り当てられる．データパスの割り当て技術を用いて，部品間の接続関係が自動的に決定される．

Synthesis of Synchronous Sequential Circuits from Specifications written in a LOTOS-like Language

Li Huang　Keiichi Yasumoto　Junji Kitamichi　Teruo Higashino　Kenichi Taniguchi

Dept. of Information and Computer Sciences, Osaka University
Toyonaka, Osaka 560, Japan

Abstract: In this paper, we introduce a LOTOS-like language called LOTOS/HD. Then, we propose a technique for synthesizing hardware circuits automatically from LOTOS/HD expressions. As the target circuits, synchronous sequential circuits are considered. In the proposed technique, first, the designers describe a specification S of a synchronous sequential circuit in LOTOS/HD where they only describe which values should be calculated and how such values are calculated using some functions implemented as combinational logic circuits. Next, from the data dependency relations and the temporal order of the events in S, a candidate C of sequential circuits implementing S is derived where the derived circuit C is a correct implementation of S if C satisfies some conditions. The variables and functions in C are allocated to the registers and combinational logic circuits, respectively. Using a data path allocation technique, the connections between their components are decided automatically.

1 まえがき

現在, 多くの通信プロトコルや分散システムの仕様がLOTOSで記述され [3], これらの仕様を実現した通信システム(通信ソフトウェア)も開発されている. 近年のハードウェア技術の進歩により, それらのシステムをハードウェア回路として実現することも可能である. 近年, LOTOSをハードウェア記述言語としてハードウェア回路を設計する方法について, いくつかの研究が行われている [2, 7]. 本論文では, まず, LOTOS風言語LOTOS/HDを提案する. LOTOS/HD仕様の機能としてはBasic LOTOSと殆ど同じであるが, I/Oパラメターを取り扱うことができ, そのLTSが有限である. 次に, 我々はLOTOS/HD仕様からハードウェア回路を自動的に合成する手法を提案する. 生成する回路として, 同期式順序回路を考える.

ハードウェアの研究分野では, 多くのハードウェア記述言語, 例えばVHDL, Verilog HDL, SFLなどが利用されている. また, 多くのハードウェア合成手法も提案されている. これらの合成手法をLOTOS仕様に対して直接適用することも考えられるが, それらの合成技法はほとんどの場合に手続き型言語をハードウェア記述言語として利用している [1, 6, 8]. このため, 抽象レベルの仕様が手続き型プログラムとして記述される場合が多い. もちろん生成される回路では最適化の手法を用いて並列性が考慮されているが, 抽象レベルではあまり並列性は考慮されていない. 一方, LOTOSを用いて抽象レベルの仕様を記述する場合, 同期並列や非同期並列などを用いた制約指向型の記述スタイルが一般的で, 多くの場合, 記述に並列性が含まれる. 従って, LOTOS仕様に対して, ハードウェア合成手法を適用するためには, 従来の合成手法をLOTOS向きに拡張/修正する必要がある.

提案する手法では, 設計者がまずLOTOS/HDで同期式順序回路の仕様Sを記述する. この抽象レベルでは, 設計者は組合せ論理回路で実現可能な関数を用いて, 計算したい内容とその計算過程だけを記述する. ハードウェア回路の構造については考慮しない. ここで, もしLOTOS/HD仕様CとSに対し, **3.** で述べる模倣関係が成り立つ時, かつその時のみ, 我々はCをSの正しい実現と定義する. 一般に, CはSに比べ, より決定的でよい. 我々はSのデータ依存関係とイベントの実行順序より, Sを実現する同期式順序回路Cを自動生成する. CもLOTOS/HDで記述される. Cがある条件を満たせば, Sの正しい実現となる. 条件を満たさなければ, 設計者がCを修正する. Cの変数と関数は, それぞれレジスタと組合せ論理回路に割り当てられる. データパス割り当て技術 [9] を用いて, レジスタと組合せ論理回路間の接続関係が自動的に決定される.

以下, **2.** ではLOTOS/HDを定義し, ハードウェア仕様の例を与える. **3.** で実現の正しさを形式的に定義する. 与えられたLOTOS/HD仕様から順序回路を生成する技法については **4.** で述べる. まとめと今後の課題を **5.** で述べる.

2 LOTOS/HDによるハードウェアの記述

2.1 LOTOS/HD

本節ではLOTOS/HD言語を定義する.

[定義 2.1]

次の条件を満たすLOTOS仕様をLOTOS/HD仕様と呼ぶ.

(1) ある観測可能なゲートが入力(出力)ゲートとして用いられた場合, そのゲートは出力(入力)ゲートとして用いてはならない.

(2) 各観測可能な入力イベントではg?x:typeのように, 入力変数xの数は一つでなければならない.

(3) 各観測可能な出力イベントはg!yの形で表現されなければならない. ここでyは既定義変数である.

(4) 選択を表す動作式はすべて ([f(y1,...,yn)] → α) [] ([not(f(y1,...,yn))] → β) の形で, 関数f(y1,...,yn)は値域がブールである全域関数である. ガードの付かない選択は許されない. すなわち各選択は決定的でなければならない.

(5) 各プロセスは並列に2つ以上のプロセスを呼び出してはいけない. 結果として, ただ1つのプロセスが呼び出される.

(6) 各観測不可能なイベントはg?y:type[y=f(x1,...,xn)]の形でなければならない. ここで, yは新しい変数で, x1,...,xnは既定義変数或いは定数である. LOTOS/HDでは, 関数名と型だけが定義され, fの内容が記述されない.

(7) 各割り込みは(α [> g?x:type; β)の形に限定し, ゲートgは観測可能で, αに現れてはならない. もしプロセスPが割り込みを表す動作式(α [> g?x:type; β)を含むなら, プロセスPは動作式βから呼び出されてはならない. すなわち可能な割り込みの数は有限でなければならない.

条件(1)から(4)まではハードウェア回路としての制限を示す. ほとんどのハードウェア回路では, 各I/Oポートは入力又は出力のいずれか一方のために利用される. 条件(1)はこれを示した. また, ほとんどのハードウェア回路では, 各ポートからの入力パラメータの数は一つである. これを条件(2)と(3)で表している. 条件(4)は非決定的動作は取り扱わないことを表している. 我々は状態数有限の回路を構成したいので, 条件(5)を与えた. 条件(6)と(7)はハードウェア合成を簡単化するために利用される. 詳細は以下の章で述べる.

2.2 LOTOS/HD仕様の例

本節では, LOTOS/HD仕様の例を与える. 例えば, 例2.1のプロセスP(n:int)はハードウェア仕様である. ここでは, 簡単のため, プロセス定義中のゲート宣言は省略されている.

[例 2.1]

```
P(n:int) :=
  (a?x:int; a?y:int; exit(x, y, any:int, any:int)
  ||| b?z:int; b?w:int; exit(any:int, any:int, z, w))
  >> accept x:int, y:int, z:int, w:int in
       ( [ x+y+z+w=0 ] → c!0; stop
       []
       [ not(x+y+z+w=0) ] →
       d!((x*x−z*z)+(y*y−w*w))/(x+y+z+w) ;
       P(n+1))
```

例2.1では, ゲートaから二つの整数x, yを入力するイベント系列と, ゲートbから別の二つの整数z, wを入力するイベント系列が並行に実行される. もし入力データの和が0であれば, ゲートcから0を出力し, そうでなければ, ゲートdから式((x*x−z*z)+(y*y−w*w))/(x+y+z+w)の値を出力する. この抽象レベルでは, 設計者は出力値がどのように

計算されるかについては記述しない. 一般には, 四つの整数 x, y, z, w に対して, 式 ((x*x−z*z)+(y*y−w*w))/(x+y+z+w) の値を計算する組合せ回路が構成できる. しかし, このような回路は非常に複雑で, 回路を構成するコストも高い. より現実な方法として, 例 2.2 のような変換が考えられる.

例 2.2 中の変数 z5 の値と例 2.1 中の式 ((x*x−z*z)+(y*y−w*w))/(x+y+z+w) の値が等しいことは, 式変形により容易に証明できる. 例 2.2 では, 新しいゲート名 g1,...,g9 と新しい変数 x1,...,x3,y1,...,y3,z1,...,z5 を導入している. g1,...,g9 は計算順序を定義するためだけに利用され, 観測不能なイベントとして取り扱われる. 新しく導入した変数は最終結果を計算するための中間結果を保存するために用いられる.

[例 2.2]

```
P'(n:int) := hide g1, ... , g9 in
  (a?x:int; a?y:int; exit(x, y, any:int, any:int)
  ||| b?z:int; b?w:int; exit(any:int, any:int, z, w))
  >> accept x:int, y:int, z:int, w:int in
      (((g1?x1:int[x1=x+z]; g2?x2:int[x2=x−z];
      g3?x3:int[x3=x1*x2]; exit(x3, any:int)
        ||| g4?y1:int[y1=y+w]; g5?y2:int[y2=y−w];
        g6?y3:int[y3=y1*y2]; exit(any:int, y3))
          >> accept x3:int, y3:int in
          g7?z1:int[z1=x3+y3]; exit(z1, any:int))
      |[g1,g4]| g1?z2:int[z2=x+z]; g4?z3:int[z3=y+w];
      g8?z4:int[z4=z2+z3]; exit(any:int, z4))
  >> accept z1:int, z4:int in
      ( [z4=0] → c!0; stop
      []
      [not(z4=0)] → g9?z5:int[z5=z1/z4];
      d!z5; P'(n+1))
```

3 実現の正しさ

本章では実現の正しさを形式的に定義する. 正しさの一つの定義として弱双模倣等価性 [5] を考えることができる. しかし, 一般的には LOTOS/HD で書かれたハードウェア仕様には並列動作が含まれているが, 生成する回路として, 同期式順序回路を考える場合, 並列性が減少するため, 得られた回路はもとのハードウェア仕様に比べより決定的である. このため, 一般には弱双模倣等価性は成り立たない. そのため, 我々は模倣関係と呼ぶ別の関係を導入する.

模倣関係 $P\ Imp\ Q$ は, (1) プロセス Pはプロセス Q に比べより決定的でよい. (2)Pのすべての実行可能なイベント系列が Q の中でも実行可能でなければならない, ことを意味している. 形式的には, 模倣関係 $P\ Imp\ Q$ を次のように定義する. ここで, Act は観測可能なイベントの有限集合である. ただし, $B \stackrel{a}{\Rightarrow} B'$ の意味は $B(\xrightarrow{i})^* \stackrel{a}{\rightarrow} (\xrightarrow{i})^* B'$とする.

[定義 3.1]

Imp を二つのプロセス間の関係とし, 与えられた二つのプロセス P,Q に対し, $P\ Imp\ Q$ が成り立つとする. 次の条件 (1) が成り立つ時, $P\ Imp\ Q$ を模倣関係と呼ぶ.

(1) 任意の観測可能なイベント $a \in Act$ に対して, もし $P \stackrel{a}{\Rightarrow} P'$が成り立てば, ある Q'が存在し, $Q \stackrel{a}{\Rightarrow} Q'$ と P' Imp Q' が成り立つ.

次に, 与えられた LOTOS(LOTOS/HD) 仕様を Basic LOTOS 仕様に変換する関数 $Proj(P)$ を定義する.

[定義 3.2]

与えられた LOTOS(LOTOS/HD) 仕様 Pに対して, P_Gは Pのすべてのガード付き選択 ($[C] \rightarrow \alpha$) を内部アクション i を用いて (i; α) に変形した LOTOS(LOTOS/HD) 仕様であるとする. P_Gに含まれるすべての入出力変数と選択条件を無視して得られた Basic LOTOS 仕様を $Proj(P)$ で表す.

例えば, 例 2.1 中のプロセス Pに対して,

```
Proj(P) := ( a; a; exit ||| b; b; exit ) >>
                (i; c; stop [ ] i; d; Proj(P) )
```

となる.

次に, 我々は模倣関係を用いて, 実現の正しさを定義する.

[定義 3.3]

二つの LOTOS(LOTOS/HD) 仕様 S,Cに対して, もし模倣関係 Proj(C) Imp Proj(S) が成り立てば, Cは Sの正しい実現であると定義する.

例えば, 例 2.1 と例 2.2 の 2 つの仕様 Pと P'に対して, 模倣関係 $Proj(P')\ Imp\ Proj(P)$ が成り立つ. よって, P'は P の正しい実現である.

ここで, 我々が $Proj(P)$ を考える理由を述べる. 次の LOTOS 仕様 Rに対して,

```
R := a?x:int; ([x=0] → b!x; stop [ ]
                [not(x=0)]→ c!x; stop)
```

文献 [3] に基づくラベル付き遷移システム (LTS) を構成すると, 図 1(a) のようになる. この場合, 入力イベント $a?x$ が実行される時点で, x の値によりイベント $b?x$ あるいは $c?x$ のいずれが実行されるかが決まる. これは LOTOS の通常の意味定義に従っている. しかし, ハードウェア回路を考えた場合, x の値を入力した後でプロセス内部で $b?x$ と $c?x$ のいずれを実行すべきかを観測不能な内部動作の実行で決定した方が自然である. つまり, 図 1(b) のような LTS の方がより適切に上述の状況を表現していると考えられる. よって, 我々は図 1(b) のような LTS が得られるように $Proj(R)$ を定義し, それを用いて実現の正しさを定義している.

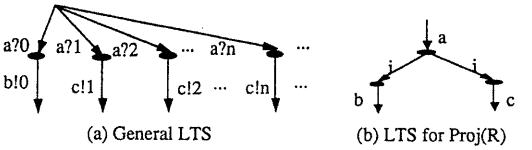

図 1: 2 種類の LTS

4 順序回路への変換

ここでは, 与えられた LOTOS/HD 仕様 Sをどのように同期式順序回路 Cに変換するかについて述べる. 簡単のため, **4.1** と **4.2** では, (1) 割り込みについては考慮しない, (2) 与えられた LOTOS/HD 仕様にはただ一つのプロセスしかない, を仮定する.

例 2.2 の仕様 $P'(n:int)$ は上述の条件を満たす. これらの制約の除去法については **4.3** で述べる.

4.1 同期式順序回路の構成

4.1.1 入出力変数間の依存関係

与えられたLOTOS/HD仕様に書かれたイベントの実行順序は構成する同期式順序回路でも保存されなければならない．実行順序に関する制約条件は整数線形不等式で記述する．各イベントに対して，識別子を用いて，構成する順序回路でどのステップで実行すべきかを表現できる．入力イベントの各変数はその実行ステップを表現する識別子として利用できる．そこで，P_x を入力イベント g?x を実行すべきステップを表現する変数とする．各出力イベントを，適当な識別子を用いて区別する．例えば，例2.2の二つの出力イベント c!0, d!z5 を，識別子 P_c1, P_d1 で区別する．

制約条件は次のように与える．まず，観測可能なイベントについての制約条件を与える．例えば，観測可能なイベント a?x と b?y があり，a?x が b?y より先に実行される必要があるなら，次の制約条件を与える．

$$(O-1)\quad P\_x < P\_y$$

一方，観測不能なイベント同士，或いは，一方が観測可能でもう一方が観測不能であるようなイベント同士については，制約条件をデータ依存関係により記述する．例えば，仕様 "g1?x[...]; ... ; g2?y[...]; g3?z[z=f(x,y)]; ..." が与えられ，イベント g1,g2,g3 が観測不能なイベントであるなら，次の制約を与える．

$$(U-1)\quad P\_x < P\_z \qquad (U-2)P\_y < P\_z$$

この制約条件は，変数 z の値を計算するためには，g3!z[z=f(x,y)] が実行される前に，まず変数 x, y の値が定義されなければならないことを表している．ここでは，もし変数 x, y 間にデータ依存関係がなければ，P_x < P_y は必ずしも成り立たない．最後に，同期並列イベント間の制約条件を与える．もしLOTOS仕様 "... g1?x; ... |[G]| ... g1?w; ..." が与えられ，ゲート g1 が G に属する場合，次の制約を与える．

$$(S-1)\quad P\_x = P\_w$$

この制約条件は，二つのイベント g1?x と g1?w が同じステップで実行されなければならないことを表している．

Max({x1,...,xn}) は変数 x1,...,xn の最大値で，Cons(P) はプロセス P に利用されるすべての識別子の集合を表すとする．もし Max(Cons(P)) の値を最小にすることができれば，構成する順序回路のステップ数も最小になる．すなわち，Max(Cons(P)) の最小値が k であれば，LOTOS/HD 仕様 P はステップ0からステップ k までの順序回路によって実現できる．Cons(P) 間のすべての制約は整数線形不等式によって記述されているので，Max(Cons(P)) を目的関数として，整数線形計画問題を解く手続きを利用して，最小値 k を得ることができる．

ハードウェアの研究分野では，集合 Cons(P) の変数に具体値を定める問題はスケジューリングの問題と呼ばれている [4]．スケジューリングの手法はいろいろある．ASAP(As Soon As Possible) はすべてのイベントをできるだけ速く実行させる手法である．すなわち，可能な最小値を集合 Cons(P) の変数に割り当てる．本論文では，ASAP 手法を用いて集合 Cons(P) の変数の値を決める．例2.2のプロセス P' に対して，

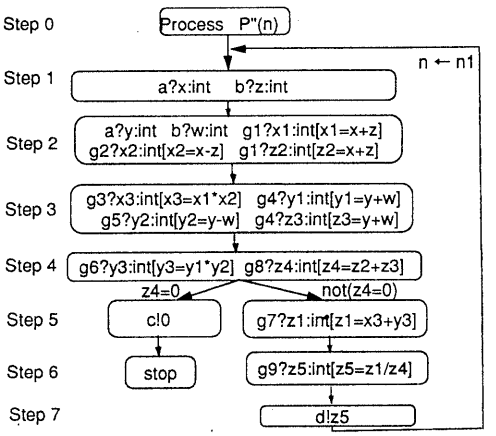

図 2: 変数を用いる状態図

Max(Cons(P')) の最小値は7である．従って，LOTOS/HD 仕様 $P'(n:int)$ はステップ0からステップ7までの順序回路によって実現できる．図2の順序回路は Cons(P') の制約条件を満たす実現の中で最も効率よい実現の一つである．

4.1.2 順序回路を表現する LOTOS/HD

本節では，LOTOS/HD により生成される順序回路の意味を定義する．例 4.1 の LOTOS/HD 仕様 P"(n:int) は図2の順序回路を表現し，順序回路の各ステップにおけるイベントが並行に実行される．

```
[例 4.1]
P"(n:int) := hide g1, ... , g9 in
    (a?x:int; exit(x, ...)
        ||| b?z:int; exit(..., z, ...))
  >> accept x:int, y:int in
    (a?y:int; exit(y, ...) |||
        b?w:int; exit(..., w, ...)
        ||| g2?x2[x2=x-z]; exit(..., x2, ...)
        ||| (g1?x1:int[x1=x+z]; exit(..., x1, ...)
        |[g1]| g1?z2:int[z2=x+z]; exit(..., z2)))
  >> .....
  >> accept y3:int, z4:int in
    (([z4=0] → (c!0; exit) >> (stop))
    []
    ([not(z4=0)] →
    (g7?z1:int[z1=x3+y3]; exit(z1))
    >> accept z1:int in
    (g9?z5:int[z5=z1/z4]; exit(z5))
    >> accept z5:int in
    (d!z5; exit ) >> P"(n+1)))
```

例 2.2 の LOTOS/HD 仕様 P'(n:int) と例 4.1 の LOTOS/HD 仕様 P''(n:int) に対して，模倣関係 Proj(P''(n:int)) Imp Proj(P'(n:int)) が成り立つ．よって，P''(n:int) は P'(n:int) の正しい実現である．LOTOS/HD 仕様 P''(n:int) では，入力

イベント $a?x:int$ は入力イベント $b?w:int$ の前に実行しなければならないとしている．一方，LOTOS/HD 仕様 $P'(n:int)$ では，$b?w:int$ は $a?x:int$ の前に実行されることがありうる．そのため，LOTOS/HD 仕様 $P''(n:int)$ は LOTOS/HD 仕様 $P'(n:int)$ と比べ，より決定的であるといえる．

4.1.3 正しい順序回路を生成する条件

4.1.1 で述べた方法により生成される順序回路は回路として正しくないかもしれない．

同じゲートでの二つの入出力イベントが同じステップに割り当てられた場合を考える．例えば，LOTOS/HD 仕様 "(a?x:int; ... ||| a?y:int; ...)" が与えられ，イベント a?x:int, a?y:int が同じステップに割り当てられる場合がある．しかし，生成する順序回路では，1ステップで，各ゲートに1つしか入力は許されない．従って，もしこのような入出力イベントが同じステップに割り当てられれば，設計者がそれらを異なるステップに割り当てらなければならない．

また，もし "(g?x:int[x=f1(...)]; ...|[g]|... g?y:int[y=f2(...)]; ...)" が与えられ，g?x, g?y が同時に実行され，関数 f1(...),f2(...) は異なれば，デードロックが起こりうる．この時，設計者は他の同期が可能かどうかを考えなければならない．

もし生成する回路の状態遷移図が正しければ，生成する回路も正しい．順序回路を構成する際にいくつかのヒューリスティックを用いた場合，生成した順序回路に対応する各 LOTOS/HD 仕様 S と C に対して，設計者は Proj(C) *Imp* Proj(S) が成り立つかどうかを確認しなければならない．

4.2 データパス割り当て

与えられた LOTOS/HD 仕様の変数の値を保存するために，いくつかのレジスタを割り当てらなければならない．簡単な回路を構成するためには，できるだけレジスタの数を少なくしなければならない．変数 x の値がステップ i で定義され，ステップ j で参照され，ステップ j 以降で参照されない場合，我々は変数 x の生存期間を [i, j−1] と定義する．もし x がステップ j で出力されれば，その変数の生存期間を [i, j] と定義する．もし二つの変数 x, y が同期イベントとして利用されれば，x, y よりなる変数の組の生存期間を定義する．例えば，図 2では，変数 x1, z2 の生存期間はそれぞれ [2, 2], [2, 3] であるので，変数の組 x1&z2 の生存期間は [2, 3] と定義する．

図 3は図 2の順序回路で用いられるすべての変数の生存期間を表している．

図 3には，変数 x2 と y2 の生存期間は重なっていない，このような変数の値は同じレジスタで保存できる．図 4は一つのレジスタ割り当てを表している．図 4では，レジスタ $R1$ はプロセスパラメタを保存するために利用される．

4.3 回路の構成

本節で，与えられたデータパス割り当てから回路を構成する方法について述べる．まず，図 2のような変数を含む状態遷移図に対して，これらの変数を適切なレジスタに割り当てることにより，変数をレジスタに置き換えた新しい状態遷移図を構成する．新しい状態遷移図で，各観測不能なイベント，例えば，g?x[x=f(y,z)] を Rx ← f(Ry,Rz) のような代入文に変形する．ここでは，Rx は x の値を保存するレジスタである．図 5は図 4のレジスタ割り当てにより変形された状態遷移図である．例えば，図 5 のステップ 2 で，イベント a?y, g2?x2[x2=x+y] が実行されている．対応する動作はそれぞれ a?R2, R5 ← R2−R3 である．

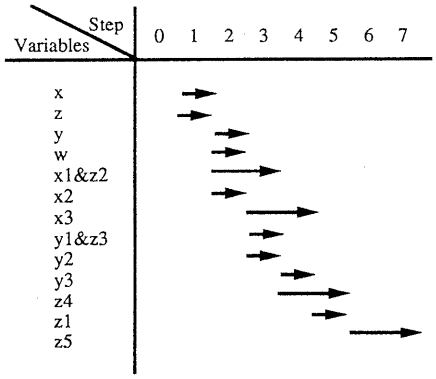

図 3: 生存期間

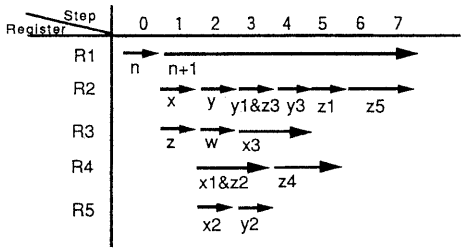

図 4: レジスタ割り当て

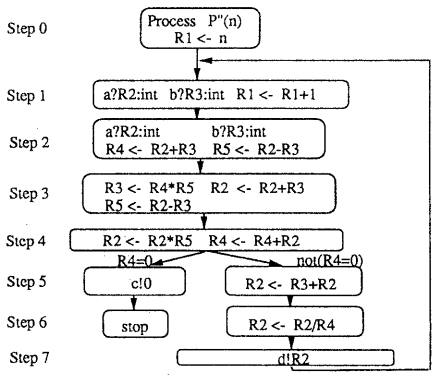

図 5: 割り当てられたレジスタを用いた状態遷移図

次に，割り当てられたレジスタを用いた状態遷移図から具体的な回路を構成する．代入文 $Ri \leftarrow F(R1,\ldots,Rn)$ に対して，レジスタ $R1$,...,Rn の出力をそれぞれ関数 F の値を計算する組合せ回路の入力に接続する．$R1$,...,Rn に定数があれば，対応する定数を入力につなぐ．次に組合せ回路の出力をレジ

スタ Ri の入力に接続する．複数の組合せ回路の出力が一つのレジスタ Ri に入力される場合は，マルチプレクサで入力制御を行う．複数のレジスタの出力が一つの組合せ回路に入力される場合も同様に，マルチプレクサで入力制御を行う．また，各出力ゲートにデータを送り出すかどうかを制御するためにゲート (tri-state buffer) を用いる．

図 6はこの方法により生成した回路を表す．図 6中の記号 MPX と G はそれぞれマルチプレクサとゲートを表す．各マルチプレクサ MPX はその制御信号 $mpxi$ の値により一つの入力を選択する．各ゲート G はその信号 $sigi$ が 1 である時のみその入力を出力する．各レジスタ Ri はロード信号 $loadi$ が 1 である時のみ入力をロードする．順序回路の状態遷移図の条件分岐のどちらを選択するかを決定するために，選択信号 $sel1$ を用いる．この選択信号は変数 $z4$ の値が 0 である時のみ 1 を出力する．

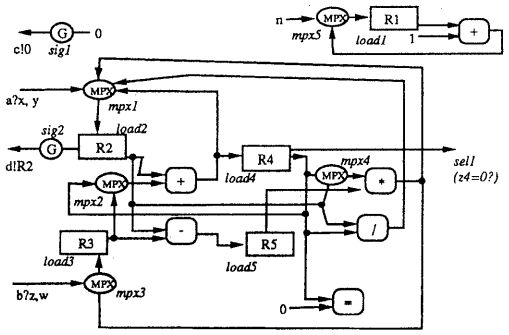

図 6: 生成する回路

この回路を実行するためには，制御回路も設計しなければならない．すなわち，回路を実行する際の各制御信号の値を決定しなければならない．一般的な技法を用いて，制御回路とそのマイクロプログラムは自動生成できる．

4.1 と **4.2** では与えられた LOTOS/HD 仕様には一つのプロセスしかないと仮定している．もし二つのプロセス P, Q が利用されていれば，P, Q に対して状態遷移図を作って，それらを一つの状態遷移図に連接し，その連接された状態遷移図から，具体的な回路を構成できる．

また,**4.1** と **4.2** では，割り込みを取り扱っていない．もし割り込み $P[> Q$ が存在すれば，P, Q に対して状態遷移図を作って，割り込みが発生しない時，Pの状態遷移図を実行し，割り込みが発生する時，Q の状態遷移図を実行する．割り込みは任意の時点で発生できるので，Pの状態遷移図から Q の状態遷移図への制御移行機能が必要である．

5 まとめ

本稿では，LOTOS のサブクラスである LOTOS/HD で記述された LOTOS 仕様からハードウェア回路を合成する手法を提案した．現在，我々は提案する手法に基づいて，与えられた LOTOS/HD 仕様からレジスタ転送レベル (RTL) の回路を導出するツールを開発している．レジスタ転送レベル (RTL) の記述から実際の IC レベルの回路への変換には，NTT で開発された自動生成ツール PARTHENON を利用する．PARTHENON は SFL[6] をハードウェア記述言語として用いている．

今後の課題として，このツールを用いて提案する手法の有用性を検討することなどがあげられる．

参考文献

[1] S. W. Director, A. C. Parker, D. P. Siewiorek and D.E. Thomas : "A Design Methodology and Computer Aids for Digital Systems", IEEE Trans. on Circuits & Systems, 28, 7, pp.634-645, 1981.

[2] M. Faci and L. Logrippo : "Specifying Hardware Systems in LOTOS", Proc. of Computer Hardware Description Languages and their Applications XI (CHDL'93), pp.305-312, North-Holland, 1993.

[3] ISO : "Information Processing System, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", IS 8807, 1989.

[4] M C. McFarland, A. C. Parker and R. Composano : "Tutorial on High-Level Synthesis", Proc. of 25th Design Automation Conf., pp.330- 336, June 1988.

[5] R. Milner : "Communication and Concurrency", Prentice-Hall, 1989.

[6] Y. Nakamura : "An Integrated Logic Design Environment Based on Behavioral Description", IEEE Trans. on Computer-Aided Design Integrated Circuits & Systems, 6, 3, pp.322-336, 1987.

[7] K. J. Turner : "An Engineering Approach to Formal Methods", Proc. 13th IFIP WG 6.1 Symp. on Protocol Specification, Testing and Verification (PSTV-XIII), North Holland, pp.357-380, 1993.

[8] R.A. Walker and D.E. Thomas : "Behavioral Transformation for Algorithmic Level IC Design", IEEE Trans. on Computer-Aided Design Integrated Circuits & Systems, 8, 10, 1989.

[9] C. Tseng and D. P. Siewiorek : "Automated Synthesis of Data Paths in Digital Systems", IEEE Trans. on Computer-Aided Design Integrated Circuits & Systems, 5, 3, pp.379-395, 1986.