計算機アーキテクチャ　98－11
設　計　自　動　化　65－11
（1993. 1. 21）

プロトコル仕様記述言語からのハードウエア合成手法

小林 哲雄　　宮崎敏明　　星野 民夫
NTT LSI 研究所
〒243-01 神奈川県厚木市森の里若宮 3-1
e-mail: tetsuo@nttica.ntt.jp

あらまし　本論文ではプロトコル仕様記述言語 SAL(Service Additional Language) で記述されたプロトコル仕様記述から制御回路を合成する新しい手法を提案する。本手法は、設計者が状態を陽に意識することなく記述した抽象度の高いメッセージシーケンス記述によるプロトコルの断片的仕様を入力として、ハードウエア記述言語 (HDL) による有限状態機械 (FSM) の記述を生成する。本手法を OSI FTAM(File Transfer, Access and Management) のメッセージシーケンスの例に適用した結果、仕様記述から論理合成可能な HDL 記述を自動生成でき、大幅に設計期間を短縮できることを確認した。

和文キーワード　上位設計支援、制御回路合成、形式的プロトコル記述、有限状態機械合成

An Approach to Interface and State Control Circuits Generation from Formal Protocol Specifications

Tetsuo Kobayashi
Toshiaki Miyazaki
Tamio Hoshino
NTT LSI Laboratories

3-1, Morinosato Wakamiya, Atsugi-Shi Kanagawa Pref., 243-01, JAPAN

Abstract
This paper presents a new high-level design methodology for synthesizing of interface and control circuits from the protocol specification language SAL. We have developed algorithms for mapping sets of partial protocol specifications into finite state machines (FSMs). The FSMs are specifined using either of two register transfer level hardware description languages, UDL/I and SFL. To demonstrate its effectiveness, we have applied our design methodology to the specification and design of the OSI FTAM application layer.

英文 key words　High Level Design, Control Circuit Synthesis, Formal Protocol Description, Finite State Machine Synthesis

1 まえがき

コンピュータネットワークやマルチプロセッサの発展によって、通信制御用LSIが上位設計の重要な対象となっている。これらの上位設計では並列に動作している他のLSIなどとの通信による相互作用を仕様として記述する必要がある。

近年、レジスタ転送レベル(RTL)よりも抽象度の高い入力仕様からLSIの論理回路を合成する研究が盛んに行われている[1]。上位仕様の記述や仕様段階での検証はVLSIの上位設計支援の中心的課題である。しかし、これらの研究の設計対象は、単一のLSIの内部に閉じたデータパス合成や演算器の割り当て問題であり、LSIと外界の通信動作を記述対象とした例は少ない。

インターフェース回路の設計支援の研究では、Borriello[3, 4]がタイミングダイヤグラムを用いて入出力動作の信号間の順序関係と時間制約を記述する方法を提案している。また、Nestor[5]らは各入出力イベント間の時間制約を記述できるようにISPS[2]の拡張を提案している。この記述方法はタイミングダイヤグラムの文字表記と見ることができる。また、この記法から時間制約を満足する回路を合成する処理系が報告されている。

これらのタイミングダイヤグラムによる手法は、入出力動作を直感的に理解しやすいという利点はあるが、

1. 条件分岐や繰り返しなどの制御構造を記述できない
2. 階層的な記述ができないため、大規模な仕様記述が困難
3. 1つの図は一つの場面における動作シナリオのみしか記述できないため、多数の場面からなる仕様を記述するためには場面の数だけダイヤグラムが必要
4. 各場合ごとに記述された動作を一つの仕様として矛盾なくマージする機能の欠如

といった欠点があるため、実際の設計に使われた例は報告されていない。

最近、ペトリネットモデルを拡張したSTG(Signal Transition Graph)による有限状態機械の仕様記述手法[6, 10]が提案されているが、設計者が仕様記述の段階で状態割り当てを意識して明示的に指定しなければならない点において、従来の設計ツールと同じ仕様記述の抽象度に留まっていると言える。

本稿で提案する設計手法は、従来、上位設計の対象としてほとんど考慮されていなかったインターフェース回路やプロトコル制御器など並列実行する複数のハードウエア間の通信の設計支援をより抽象度の高い仕様記述で可能とすることを目標にしたものである。

2 従来の上位設計の問題点

RTLより抽象度の高い仕様記述法、仕様の検証法、および、仕様からの回路合成法の3つは、上位設計支援技術における主要問題である。

従来、LSIの設計において、状態遷移表の作成を設計者が人手で行なっていた。このため、従来のプロトコル制御や通信制御用LSIの方式設計以降の設計過程において次のような問題点がある。

1. 方式設計書から状態遷移表までの設計過程がすべて手作業
2. 方式設計書は自然言語による記述のため曖昧な部分が残る
3. 仕様書の矛盾解消には高度なスキルが必要
4. 状態割当を人手で行うため時間がかかり誤りが混入しやすい

この設計過程において、設計者が人手で書いていた状態遷移表の作成以降の設計過程を支援する設計支援ツールが最近出はじめたばかりであり、状態割当や状態遷移表の作成より上流の設計に使用可能な設計支援ツールは皆無に等しい。

一方、プロトコル工学の分野では並列に動作するプロセス間の通信仕様を形式的に記述したものを入力として、仕様の検証やソフトウエアによる仕様の実装を目指した研究がされている[11, 12]。

方式設計書と親和性がよいメッセージシーケンス記述をもとに有限状態機械を合成するツールを提供できれば、これら通信制御系LSIの上位設計に大きな助けとなるものと考えられる。

3 本設計手法の特長

プロトコルを定義するための形式的記述言語として、LOTOS[8]、SDL[9]などが存在する。今回、筆者らはプロトコル仕様記述言語SAL (Service Additional Language) [14]を選び、LSIの上位仕様記述への適用を試みた。

メッセージシーケンス記述による入出力動作の記述から、ハードウエア記述言語(HDL)による有限状態機械を合成する新しい設計手法を示す。本手法は、従来の設計手法に比較して、次のような特長がある。

1. メッセージシーケンスから有限状態機械を自動合成するため、従来の人手設計と比較し、記述の抽象度をより高くすることが可能。
   このため、設計者は状態を意識せずに仕様記述を行なえる。
2. 断片的な動作仕様を入力として部分仕様の集合として仕様記述が可能。
   LOTOSやSDLなど他のプロトコル仕様記述言語では全体仕様を知らないと仕様記述ができないという問題がある。
3. 仕様段階でデッドロック検出や実行可能性検証が可能。
4. 仕様記述からCCITT準拠のSDL図を出力でき仕様図として利用できる。

4 プロトコル処理のモデル化

4.1 SAL のモデル

SAL の記述対象となるモデルを単純な記述例を用いて説明する。
SAL では並列実行する複数のプロセスが互いに FIFO チャネルを介してメッセージを送受信し合う有限状態機械としてモデル化される [14]。

従来、このような並列プロセスの動作を記述するには各プロセスごとの動作を状態遷移としてそれぞれ記述するという方法が取られることが多かった。しかし、この場合、システム全体の動作を理解しづらいという問題がある。そのため、仕様の理解を助けるために図 1 に示したような、プロセス間の信号のやり取りを表現するメッセージシーケンス図を同時に用いることが多い。個々のプロセス動作を、時間経過を上から下へとった縦軸で表し、有限状態機械（プロセス）間のメッセージのやり取りを横軸方向の矢印と信号名で表現する。通信以外の動作はボックスとして表す。

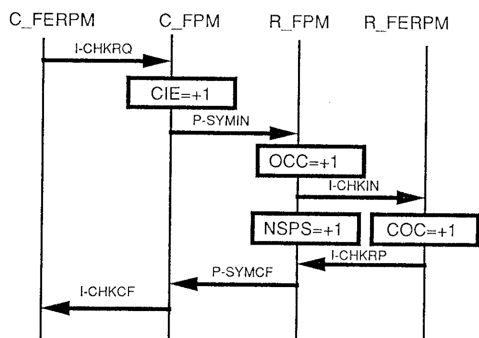

図 1: メッセージシーケンス図の例

SAL の基本的なアイディアは、断片的に記述されたシステムの動作を規定するメッセージシーケンス図からシステム全体のプロセス状態遷移を自動生成することである。

従来の Borrielo などの方法では、条件分岐、繰り返し等の記述や大規模な仕様を階層的に記述することができないため、正常動作と各種のエラー回復動作など類似した動作を複数記述する際に記述の重複が多く、仕様記述から状態遷移を自動生成するシステムの入力としてメッセージシーケンス図を用いるのは困難であった。SAL では、仕様の階層的記述や条件分岐、繰り返しなどの制御構造の記述子を導入し、メッセージシーケンス図の概念を直接言語形式で定義可能にしている。

4.2 SAL の記法

SAL による仕様記述は次の基本形をとる [13]。

```
<サービス名> {
   <プロセス名1> = <イベントシーケンス>;
   <プロセス名2> = <イベントシーケンス>;
         .....
   <プロセス名n> = <イベントシーケンス>;
}
```

サービスとは、複数のプロセスが相互に通信を行いながら実行するシステムの動作記述である。イベントシーケンスは、左辺のプロセスの動作（イベント）を記述し、左から右へ順に実行される。プロセス間の通信には送信イベントと受信イベントがあり、

− <送信先プロセス名>(メッセージ)

＋ <受信先プロセス名>(メッセージ)

として記述する。−記号はメッセージの送信を、＋記号は受信を意味し、メッセージにはキーワードまたは値を記述することができる。

通常のソフトウエアのプログラミングにおけるサブルーチンやマクロ呼び出しなどに相当する通信以外のイベントの呼び出しは

.<イベント名>(引数)

として記述する。

これらの動作は別途、

```
macro イベント名 (引数) {<イベントシーケンス>};
```

または、

```
static イベント名 (引数) {<イベントシーケンス>};
```

として定義される。

イベントシーケンスの記述において、繰り返しを表現するために.loop や.while などの記述子が準備されている。

```
.loop{<イベントシーケンス>}<受信イベント>
.while(<終了条件式>){<イベントシーケンス>}
```

は、{ } で囲まれたイベント列が繰り返されることを意味する。繰り返しの終了は、.loop の場合 { } の次の受信イベントで指定された信号の到着まで、一方、.while では終了条件式が真である限り繰り返しを実行する。

条件分岐は.if によって記述する。

```
.if(<条件式>){<イベントシーケンス>}
```

条件式が真の場合、イベントシーケンスを実行する。偽の場合は、なにも実行せず次のイベントへ制御を移す。

4.3 SAL のモデルとハードウエアの対応

表 1 に示すように、SAL のモデルをハードウエアのモデルに対応つける。

表 1: SAL とハードウエアのモデル対応関係

SAL のモデル	ハードウエアのモデル
プロセス	有限状態機械
チャネル	ネット
送信イベント	出力線
受信イベント	入力線

また、合成対象のハードウエアモデルには以下に示す仮定を設けた。

1. 信号線は単方向であり、双方向バスは存在しない。
2. 通信ポートおよび信号線は専用に設けられ、ポート資源などの排他制御は必要としない。
3. 単相クロックで各動作は1クロック以内に完了する。

5 システムの処理フロー

本稿で提案するHSPS(Hardware Synthesis from Protocol Specification) システムの処理フローを図2に示す。

まず、正常動作仕様、エラー回復処理仕様などSALで個別に記述された動作記述は、SALコンパイラに入力され、仕様矛盾の検出や実行可能性の検証を経て、ソフトウエアでの実現を念頭に置いたプロセス遷移表現(PST)と呼ばれる有限状態機械記述へ変換される。ここまでの処理は、別途開発されたSDEシステム[15]で行なわれる。その後PST表現は、状態圧縮、意味変換過程を経てHDL記述となる。以下にそれぞれの処理を述べる。

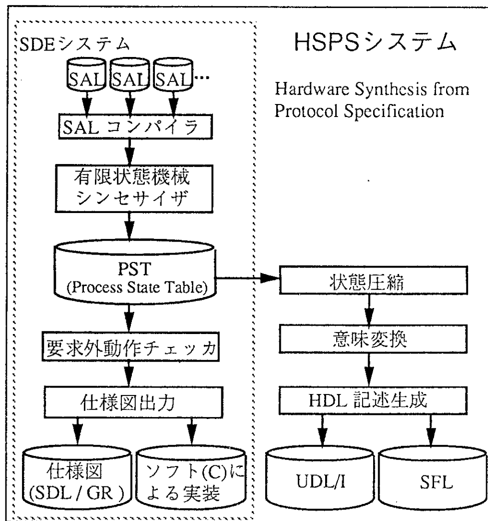

図2: システムの処理フロー

5.1 仕様検証

仕様からHDLによる有限状態機械の記述を生成する過程で、仕様記述が正しいか否かを確認するために、つぎの3つの検証を行うことができる。

- **実行可能性検証**
  各プロセスの記述された要求仕様が途中で停止せず実行可能なこと。
- **仕様の両立性検証**
  既存の設計仕様と追加の仕様が矛盾なくマージできるかを非決定的な分岐が存在しないことを確認する。
- **要求外動作の検出**
  要求仕様に示されていない動作をしないこと。

5.2 SALコンパイラ

SALで記述された動作仕様を各プロセスごとのメッセージシーケンス記述へコンパイルする。この段階で、プロセス間の信号のやり取りの対応関係を調べ、お互いに相手からの信号を待つデッドロックを検出する。すべてのプロセスが仕様の終了まで遷移するかを調べることで仕様の実行可能性を検証する[15]。

5.3 有限状態機械シンセサイザ

ここでは、有限状態機械(FSM)を生成する。部分仕様のマージや既存の動作仕様から合成されたFSMと新たに追加する仕様からコンパイルされたメッセージシーケンスをマージして各プロセスごとにFSMを生成する処理を行う。このFSMはPST(Process State Table)と呼ばれる形式で生成される。矛盾なくマージできるか、仕様の両方が共に実行可能かをこのPST生成時に検証する[16]。

仕様のマージは既存および追加のメッセージシーケンスがお互いに受信イベントである箇所、または、お互いに排他的な条件による分岐である箇所において可能である。仕様が両立するためには、複数の状態への遷移において、次状態が決定的であることが必要である。仕様が非決定的な遷移を含む場合は実現不能となる。

5.4 要求外動作チェッカ

既存のFSMへ新たに仕様を追加しマージする際に、実現したい仕様以外の動作を含むことがある。例えば、図3に示すように、2つのプロセスpとqの間でpから相手を起動する場合とqから相手を起動するという、仕様は単独では実行可能である2つの仕様をマージした場合、両者が同時に相手を起動する場合という動作がマージによって得られてしまう。この場合、送信と受信イベントに過不足が生じるため、要求外動作がマージしたFSMに含まれていることを検出することが可能となる。

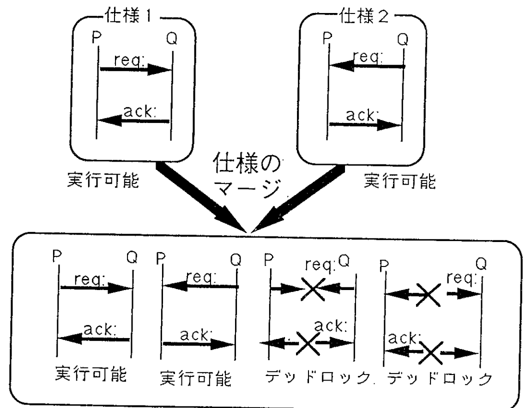

図3: 検出可能な要求外動作の例

5.5　PST からの状態圧縮

生成された FSM の表現 PST は、本来、プロトコル記述からの通信ソフトウエア自動仕様を目的としたものであり、仕様の動的検証の容易化のため、受信イベントや送信イベントに各々１つの状態を割り当てている。そのため、SAL による簡単な仕様記述でも PST ファイルでは状態数が多くなる。ソフトウエアによる FSM の実現では、状態数の多さは性能にほとんど影響を与えないが、ハードウエアによる場合、面積や回路規模の増大に直結する。そのため、SAL 記述による仕様をハードウエア記述言語 (HDL) へ変換するためには状態数の削減が不可欠である。

HDL では、状態変数値の書換えという明示的方法で状態を表現する場合を除いて、一般には記述されたテキストの実行位置が状態に対応している。そのため、受信イベント条件成立待ち状態は個別の状態として扱うが、それ以外の通信を伴わない演算処理のシーケンスは一つの状態としてまとめても動作仕様 本来の意味を損なわない。

連続する受信イベントについては、メッセージの到着順序に依存せずに一連の受信イベントが到着することを待つものとし、連続する送信イベントは、すべて同時に実行されるものとして、HDL の状態を割り当てる。SAL の記述では、通信イベント間の経過時間を規定していないため、連続する通信イベントが同時に発生する場合も順次発生する場合も同じ記述であるため、状態圧縮においてこのような取扱いをしてもモデル上の問題は生じない。

この状態圧縮結果は、次のような中間言語形式で表現される。

```
(<module 名>  <state-card> {<state-card>})

<state-card> ::=
  (<状態 ID> (<event-list>
              {<event-list>}) <次状態 ID>)

<event-list> ::=
    − <送信先 module 名> (<message>)
 |  ＋ <受信先 module 名> (<message>)
 |  . 非通信イベント (<引数>)
 |  .IF (<条件式>) (<event-list>)
 |  .loop{<event-list>}<受信イベント>
 |  .while(<終了条件式>){<event-list>}

<message> ::= (<信号線名> <信号値>)
```

ただし は括弧の内容の 0 回以上の繰り返し記述が可能であることを表す。

5.6　意味変換

前節の状態圧縮に引き続いて、各種の HDL 記述の生成を容易化するために、さらにいくつかの意味変換を行う。

SAL の記述は非同期の仕様記述であるが、各状態での動作は１クロック以内で完了するものと仮定して、同期式 FSM の記述へ変換する。

この変換と同時に、SAL のモデルをハードウエアの実装モデルに対応させる。具体的には、SAL のメッセージ送信や受信を直接ハードウエアで実現するのではなく、送信イベントに対しては出力端子を宣言し、その端子に対して値を代入するという記述へ変換する。また、受信イベントに対しては入力の値を監視して所定の値の到着を待つものとして変換する。

SAL での if 文および if 文に後続する受信イベントは、HDL 上では一つの状態の始まりを判定する条件式へ変換する。そのため中間言語表現では、if 文は各状態の最初にしか現れない。ここでは、「一つの状態中では、各イベントの実行は仕様記述の順序に従う」というモデルを仮定し FSM を生成する。

これらの変換後、中間言語は以下のように形成される。

```
(<module 名> <state-card> {<state-card>})

<state-card> ::=
   (<状態 ID> ( <event-list> ) <次状態 ID>)

<event-list> ::=
  @IF (<条件式>{<論理演算子><条件式>})
                              (<event-list>)
 | @非通信イベント (<引数>)
 | @loop{<event-list>}<受信イベント>
 | @while(<終了条件式>){<event-list>}

<条件式> ::=
   (<信号線名> <値>)
 | (<信号線名> <論理演算子> <値>)
```

ただし は括弧の内容の 0 回以上の繰り返し記述が可能であることを表す。

5.7　HDL 記述生成

ここでは中間言語から目的の RTL 記述を出力する。本システムは、複数の HDL 記述を出力できる。以下、UDL/I[17] および SFL[18] について述べる。

まず、中間言語から UDL/I によるオートマトン記述へ変換する方法を説明する。UDL/I ではオートマトンを記述するための記述子が提供されている。また、前段までの処理で、状態名の付け変えや意味変換が完了しているため、UDL/I と中間言語でのオートマトン記述との間に意味的な相違点はほとんどない。そのため、図 4 に示すテンプレートにそって単純に変換するという単純な構文変換だけで生成することができる。.rst はリセット端子名、.clk は動作クロック端子名を各々表している。

次に、中間言語から SFL によるオートマトン記述へ変換する方法を説明する。オートマトン記述専用の記述子を提供する UDL/I とは若干異なり、SFL では state や par という記述子と、first_state と goto という制御の順序を指定する道具だてを図 5 に示す例のように組み合わせてオートマトンを記述することになる。

中間言語から SFL への変換は、中間言語の一つの state にだけ着目して行なうことができる。外部端子への出力を

意味する代入文を前節で述べたように2つの代入文に変換することを除いては意味レベルの変換は必要なく、他は全て構文的な変換だけである。

```
AUTOMATON : オートマトン名 : .rst : .clk ;
  状態名1: WAIT( オートマトン起動条件式1 ) :
    BEGIN
      実行文11;
      実行文12;
        .....

      -> 次の状態名;
    END;

  状態名2: WAIT( オートマトン起動条件式2 ) :
    BEGIN
      実行文21;
      実行文22;
        .....

      -> 次の状態名;
    END;
    .....
END;
```

図4: UDL/Iによるオートマトン記述のテンプレート

```
stage interface{
    state_name state1;
    state_name state2;
    .....
    state_name state32;

    state state1 par{
        operational_o_reg := 0b1;
        operational_out   = operational_o_reg;
        .....

        goto state2;
    }
    state state2 par{
        any{( operational_in) : par{
            bus_o_reg := 0b1;
            bus_out   = bus_o_reg;
            .....

            goto state3;
            }
        else : par{
            goto state2;
        }
    }
    .....
```

図5 SFLによるオートマトン記述の例

6 実験

本合成手法の有効性を示すため、OSI FTAMの仕様書[19]に基づき、ファイルプロトコルの動作仕様を記述した。

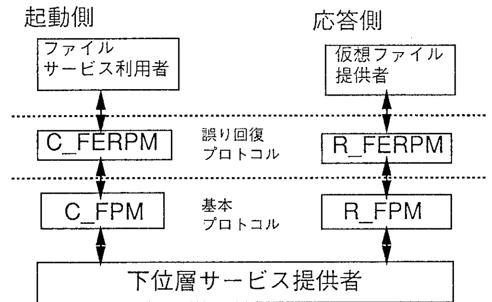

図6: FTAMの機能モデル

6.1 記述対象

FTAMはファイルに対する各種の操作、つまり、ファイル転送、アクセス(読み出し、書き込み、書き換え)および管理(生成、削除、オープン、クローズ)に関するプロトコルである。通常、ファイルに対するこれらの操作はコンピュータシステムに依存するものであるが、FTAMは、異機種上にあるファイルをあたかもローカルファイルを使うように、遠隔から操作できるような手段を提供するものである。

FTAMの機能モデルを図6に示す。このモデルではファイル操作を要求する起動側と、その要求に応じて仮想ファイルを操作する応答側の間の通信は、プレゼンテーション層より下位のレイヤーによって規定されるサービスプリミティブを介して行なわれる。

FTAMはOSIのプロトコル階層ではアプリケーション層に位置するものであり、タイムアウトによる接続断などの下位レイヤーの実現方法はプレゼンテーション層以下で規定されるため、本仕様書の範囲外である。

起動側、応答側はそれぞれ、基本ファイルプロトコル機械(FPM)とファイル誤り回復プロトコル機械(FERPM)という2つの並列動作するFSMから構成される。FERPMは、FPMが提供するサービスを利用して誤り回復機能を実現する。したがって回復や再開は、FTAMのユーザには見えない。

これら4つのFSMの他に、FTAMを利用するファイルサービスユーザ(FSU)と、起動側の要求に応じて応答側で仮想ファイルの操作を行なうファイルシステム(FS)の動作もFSMとしてモデル化し記述を行なった。

FTAMの動作シーケンスでは、基本(正常時)プロトコルの他に、実行時に発見した誤りの種類に応じて、処理の中断や回復の動作を12通りの誤り回復プロトコルとして規定している。これら誤り回復プロトコルは基本(正常時)プロトコルや他の誤り回復プロトコルと全面的に異なる動作仕様ではなく、類似するプロトコル部分が多くそれぞれの場合により部分的に異なる動作をするものとして記述されている。

6.2 結果

表2と表3に実験結果を示す。表中、FSM名の前のC_は起動側であることを、R_は応答側のFSMであることをそれぞれ示す。FTAMの動作のうち、正常時のプロトコルから2つ例を取り、ソフトウエアでの実現を指向したFSMであるPST表現上での状態数と、それを前述した状態圧縮および意味変換を行なってハードウエアのモデルに変換した後の状態数を示す。また、合成したUDL/I、SFL記述の行数もあわせて示す。

表2: 状態圧縮の効果 [WRITE操作(正常時動作)の場合]

FSM名	SAL記述	PST表現	変換後	状態の圧縮率	UDL/I	SFL
	行数	状態数	状態数	%	行数	行数
c_fsu	25	35	8	77	22	25
c_ferpm	48	85	42	51	52	67
c_fpm	54	76	27	65	43	51
r_fpm	61	80	22	73	37	50
r_ferpm	45	85	41	52	52	67
r_fs	32	37	9	74	17	21

表3: 状態圧縮の効果 [READ操作(正常時動作)の場合]

FSM名	SAL記述	PST表現	変換後	状態の圧縮率	UDL/I	SFL
	行数	状態数	状態数	%	行数	行数
c_fsu	28	37	16	57	25	27
c_ferpm	51	85	41	52	51	53
c_fpm	60	81	25	69	52	53
r_fpm	55	82	27	67	39	43
r_ferpm	45	83	41	51	52	54
r_fs	26	39	13	67	20	24

仕様記述のなかで、特定の送信モジュールが決まった受信モジュールに対して連続して通信をするプロトコルが多い場合には特に状態圧縮の効果が大きいことがわかる。また、一般に、上位合成では抽象度の高い入力仕様からHDL記述を生成した場合、記述量は増加するが、本システムの場合、入力SAL記述と出力HDL記述の行数はほぼ等しい。これは上述の圧縮の効果が大きいと思われる。生成されたHDL記述は、論理合成ツールにより論理合成可能であることを確認した。

7 まとめ

陽に状態遷移を意識することなく、しかも、断片的に仕様が記述されたプロトコル仕様記述からハードウエアを合成する手法を提案した。

本手法をOSI FTAMのプロトコル仕様に適用した結果、仕様の曖昧さや誤りの混入を防ぎ、仕様段階でその無矛盾性を検証できるとともに、ハードウエア自動合成により設計効率の向上をはかれることを確認した。

謝辞

SALの言語とモデルについて指導をいただいたソフトウエア研究所 市川晴久氏、加藤順氏、荒川則泰氏、ならびに日頃、有益な助言を頂くLSI研究所 設計システム研究部 安達徹氏に感謝します。

参考文献

[1] R. Camposano, W.Wolf, "High-Level LSI Synthesis", Kluwer Academic Publishers, 1991.

[2] M. R. Barbacci, "Instruction Set Processor Specifications(ISPS): The Notation and Its Applications", IEEE Transactions on Computers C-30(1), pp.24-40, 1981.

[3] G. Borriello et al., "Synthesizing Transducers from Interface Specifications", VLSI '87, pp.403-418, 1988.

[4] G. Borriello, "Specification and Synthesis of Interface Logic", Kluwer Academic Pubulishers, 1991.

[5] J. A. Nestor et al., "Behavioral Synthesis with Interfaces", ICCAD '86, 1986.

[6] P.Vanbekbergen et al., "A Generalized State Assignment Theory for Transformations on Signal Transition Graphs", ICCAD '92, 1992.

[7] K.J.Lin et al., "On the Verification of State-Coding in STGs", ICCAD '92, 1992.

[8] ISO, "Definition of the Temporal Ordering Specification Language LOTOS", ISO/TC97/WG1 N299

[9] CCITT,"SDL Recommendation Z100-Z104", 1984.

[10] K.J.Lin et al., "On the Verification of State-Coding in STGs", ICCAD '92, 1992.

[11] P. Zafiropulo et al., "Towards Analyzing and Synthesizing Protocols", IEEE Trans. Comm. COM-28(4), pp.651-660, 1980.

[12] M.G. Gouda et al., "A Technique for Proving liveness of Communicating Finite State Machines with Examples", Proc. ACM Symp. on Princeples of Distributed Computing 3rd, pp.38-49, 1984.

[13] 市川 晴久 他、「並行処理の監視性と通信プロトコル実装への適用」、信学会論文誌B、Vol70-B, No.5、pp.565-575、1987年5月.

[14] H. Ichikawa et al. "Communications Software Management with Verification and Transformation into CCITT Specification and Description Language", The Transaction of the IECE of Japan, Vol69-B, No.4、pp.524-535, 1986.

[15] 加藤 順 他、「通信サービスの拡大を支援するソフトウエア作成環境(SDE)」、NTT R&D, Vol38, No.11, pp.1249-1256, 1989.

[16] 伊藤 正樹 他、「並行プロセスを基本とした交換プログラム仕様の階層的検証法」、信学会論文誌B、Vol69-B, No.5、pp.449-459、1986年5月.

[17] UDL/I Committe, " UDL/I Language Reference Draft Version 1.0h4", Japan Electric Industry Development Association, 1991.

[18] Y. Nakamura et. al., " An RTL Behavioral Description Based Logic Design CAD System with Synthesis Capability", IFIP CHDL '85, 1985.

[19] 日本規格協会、"開放型システム間相互接続の基本参照モデル JIS X 5003-1987 参考 S004(V2.0) FTAM 実装規約", 1990.