プロトコル仕様記述言語を入力とする制御回路の機能合成

小林哲雄[†,*] 宮崎敏明[†,**] 星野民夫[†]

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，ハードウェア記述言語 (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．

Behavioral Synthesis for Control Circuits Using Protocol Specification Language

Tetsuo Kobayashi[†,*] Toshiaki Miyazaki[†,**] and Tamio Hoshino[†]

This paper presents a new high-level design methodology for synthesizing 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 specified 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.

1. はじめに

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の上位設計が重要になっている．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]．上位仕様記述からの合成や仕様段階での検証はVLSI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．本稿で，プロトコル仕様記述言語から制御回路を自動合成する手法を提案する．

† NTT LSI 研究所設計システム研究部
LSI Design Systems Laboratory, NTT LSI Laboratories
* 現在 NTT 交換システム研究所伝達装置研究部
Presently with Transport Network Systems Laboratory, NTT Communication Switching Laboratories
** 現在 NTT 伝送システム研究所伝送処理研究部
Presently with Transport Processing Laboratory, NTT Transmission Systems Laboratories

本論文では，まず，従来の上位設計の問題点を述べ，第3章で関連する研究と本手法の特徴を，第4章において仕様のモデル化と記述について説明する．第5章において処理のフローを示し，第6章に実験結果を，最後に考察と今後の課題について言及する．

2. 従来の上位設計の問題点

従来のプロトコル制御や通信制御用 LSI の設計では，LSIの動作要求は方式設計書としてまとめることから始まる．方式設計書では，与えられた入力に対して複数のモジュールが通信しながら並列に動作する様子を，場合分けされた場面ごとに断片的に述べている．この動作記述は，一部は理解を助けるためにメッセージシーケンス図などを用いているほかは，主に自然言語で記述されているものである．

一方，現在実用化されている論理合成システムの入力は有限状態機械 (FSM) の状態遷移表記述やハードウエア記述言語 (HDL) による RTL 記述である．動作は各 FSM ごとに独立に，取りうるすべての状態を記述することによって規定される．

これら入力と方式設計書の間には次のような大きな意味ギャップが存在する．

● 記述の焦点

方式設計書は，場面ごとに分けられた断片的動作記述であり，各場面ごと複数のモジュールの相互作用が記述される．一方，HDL では，動作記述は各 FSM についてそれぞれ，一つのモジュールについての動作という視点で記述される．

● 記述の抽象度

方式設計書では，状態分けを意識せずにメッセージシーケンスや自然言語により動作が記述される．一方，HDL では，状態分けが明確にされた後，記述言語により動作が記述される．

方式設計書から状態遷移表や RTL への書き換えは経験を積んだ設計者の人手に頼っている．そのため，方式設計書以降の設計過程において次のような問題がある．

1. 方式設計書は自然言語による記述のため曖昧な部分や矛盾が残りうる．
2. HDL への書換え段階ではデッドロックなどの仕様誤りの検出が困難である．
3. 方式設計書の不明点や矛盾の解消，場合分けの抜け検出には高度のスキルが必要である．
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．

この方式設計書から状態遷移図を作成する設計過程において，両者の間の意味ギャップを埋める中間的な動作記述として，方式設計書と親和性の良い動作記述から FSM や HDL 記述を生成する手法がこれら通信制御系 LSI の上位設計において求められている．

3. 関連する研究

Borriello[2]は，インタフェース回路の設計支援手法としてタイミングダイヤグラムを用いて入出力動作に伴う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[3]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[5] 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，タイミングダイヤグラムから時間制約を満足する回路を合成する処理系[4]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の記述が不可能．
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. 一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),7)]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いえる．

これら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されているものは皆無に等しい．

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ェアによる仕様の実装を目指した研究がされている[8),9)]．

プロトコルを定義するための形式的記述言語として，LOTOS[10]，SDL[11]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)[12] 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，ハードウェア記述言語 (HDL) による有限状態機械を合成する新しい手法を示す．本手法は，従来の手法に比較して，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 図と同時に RTL 記述を自動合成.
5. 既存の論理合成系と連係する手順を採用し，ハードウェア自動合成により設計期間の短縮.

さらに，プロトコル仕様を，例えばOCCAMのような汎用の並列プログラミング言語を用いた記述[13),14)]や，動作記述可能な HDL である VHDL 記述等からハードウェアを自動合成するというアプローチ[15)~17)]も考えられる．しかしそれらのアプローチは，直接，SDL で仕様を記述するのと同様に，システムすべての動作仕様を把握した上でないと記述ができないという問題が残る．それに比べ，SAL を使用する本方法では，部分仕様の集合として記述が可能であり，段階的な仕様追加が頻発する LSI 上位設計の支援ツールとして優れていると考える.

4. プロトコル処理のモデル化

4.1 SAL のモデル

われわれは，仕様記述言語として SAL (Service Additional Language)[12)]を用いた．ここでは，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.

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.

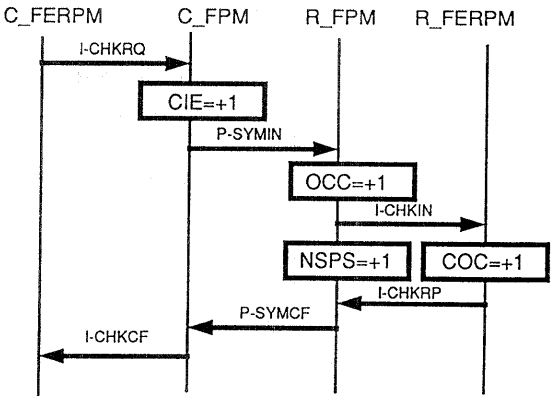

図 1 メッセージシーケンス図の例
Fig. 1 An example of message sequence chart.

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 Borriello[2)]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

図 1 で示したメッセージシーケンスの SAL による記述を**図 2** に示した.

SAL による仕様記述は次の基本形をとる[18)].

〈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)

```
object example_1 () {
 c_ferpm = -c_fpm(i_chkrq:) +c_fpm(i_chkcf:);

 c_fpm   = +c_ferpm(i_chkrq:) .(%{ cie++; %})
           -r_fpm(p_symin:) +r_fpm(p_symcf:)
           -c_ferpm(i_chkcf:);

 r_fpm   = +c_fpm(p_symin:) .(% occ++ ; %})
           -r_ferpm(i_chkin:) .(% nsps++; %})
           +r_ferpm(i_chkrp:) -c_fpm(p_symcf:);

 r_ferpm = +r_ferpm(i_chkin:) .(%{ coc++; %})
           -r_fpm(i_chkrp:);
}
```

図 2 図 1 に対応する SAL による記述例
Fig. 2 The SAL description corresponding to Fig. 1.

+〈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．

仕様を階層的に記述するために，通常のプログラミング言語でのマクロやサブルーチンに相当する記法として，macro と static が準備されている．これらは，それぞれ，
macro マクロ名(引数) {〈イベントシーケンス〉} ;
static マクロ名(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を待って終了する．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ェアの対応

表 1 に示すように，SAL のモデルをハードウェアのモデルに対応づける．

また，合成対象のハードウェ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．

表 1 SAL とハードウェアのモデル対応関係
Table 1 Correspondence between SAL model and hardware.

SAL のモデル	ハードウェアのモデル
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

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 3** 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，ソフトウェ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[19]で行われる．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．

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，つぎの三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を確認する．
- **仕様の両立性検証**
  マージする個々の仕様の起動条件の間に非決定的な条件分岐が存在しないことを調べることで，すでに設計済みの仕様と，新たに追加する仕様とが矛盾なくマージ可能であることを確認する．
- **要求外動作の検出**
  個々の断片的仕様では示されていなかった動作仕様が，仕様のマージによって生成されていないか

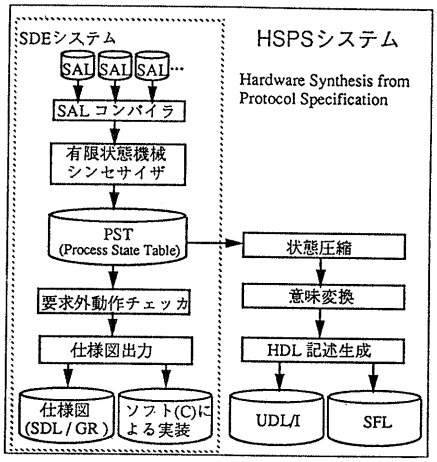

図 3 システムの処理フロー
Fig. 3 An overview of the HSPS system.

を，送信と受信のイベントの対応関係を調べることで確認する．

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ってしまうという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[19]．

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が，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[20]．

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4に示すように，二つのプロセス p と q の間

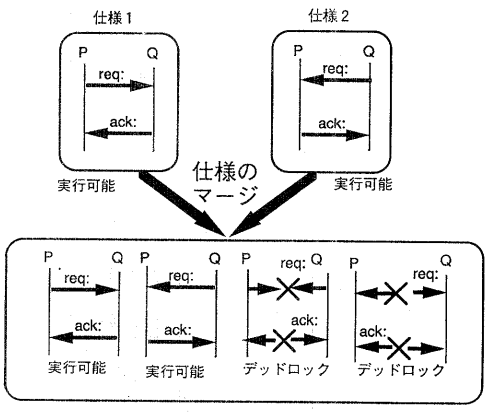

図 4 検出可能な要求外動作の例
Fig. 4 An example of detection of unacceptable behaviors.

で p から q を起動する動作と，q から p を起動するという二つの動作仕様はそれぞれ単独では実行可能な仕様である．この二つの動作仕様をマージした場合，p と q が同時に相手を起動するという実行不可能な仕様が得られてしまう．このようなマージによって生成される実行不可能な仕様を要求外動作という．本システムでは，送信と受信イベントの対応関係を調べることで，マージした FSM に要求外動作が含まれているか否かを検出することが可能である．

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ェア自動合成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おのおの一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．ソフトウェアによる 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，ハードウェアによる場合，速度や回路規模の増大に直結する．よって，SAL 記述による仕様をハードウェア記述言語（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，状態変数値の書き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想定していないため，連続する通信イベントが同時に発生する場合も順序発生する場合も同じ記述である．よって，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．

以上述べたように，人工言語理論で用いられている FSM の状態と本システムで扱う状態の意味は異なる．このため，従来から研究されている状態数圧縮手法[21),22)]が本システムへ必ずしも適用可能ではない．

本システムでは並列に動作する FSM 間の送信・受信イベントに着目して状態圧縮する．本状態圧縮法では，シーケンスの前後でのみ圧縮するため，別々の動

作条件下で起動される同様の処理に対して別々の状態を割り当ててしまうことがある．そのため，条件分岐した処理を広域的に解析することでさらに状態を圧縮できる可能性が残されている．

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ェ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ェ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[23]および SFL[24]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5**に示すテンプレートにそって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おのおの表している．

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

```
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1 1 ;
      実行文1 2 ;
        .....

      ->    END;
    .....
END;
```

図5 UDL/I によるオートマトン記述のテンプレート
Fig. 5 Generic automaton specification in UDL/I.

```
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 6 SFL によるオートマトン記述生成の例
Fig. 6 An example of automaton description in SFL.

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 6** 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二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，他はすべて構文的な変換だけである．

6. 実験および考察

本章では，本合成手法の有効性を示すため，OSI FTAM の仕様書[25]に基づき，ファイルプロトコルの動作仕様を入力とした場合の実験結果を示す．

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

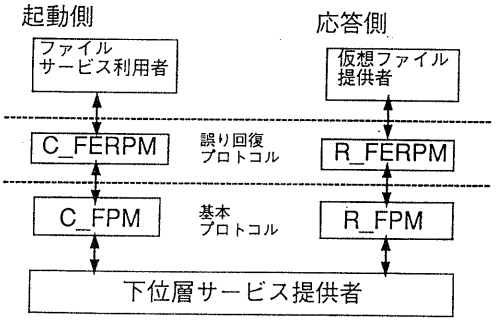

図 7 FTAM の機能モデル
Fig. 7 FTAM functional model.

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．

FTAM の機能モデルを**図 7** 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によって規定されるサービスプリミティブを介して行われる．

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）という二つの並列動作する 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．

これら四つの FSM のほかに，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った．

FTAM の動作シーケンスでは，基本（正常時）プロトコルのほかに，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 状態数圧縮効果

表 2 と**表 3** に 5.5 節で述べた状態数圧縮方法の実験結果を示す．これは，FTAM の動作のうち，正常時のプロトコルから二つ例を取り，ソフトウェアでの実現

表 2 状態圧縮の効果 [WRITE 操作 (正常時動作) の場合]
Table 2 State compaction performance. (WRITE operation in normal mode)

FSM 名	SAL 記述行数	PST 表現状態数	変換後状態数	状態の圧縮率 (%)	UDL/I 行数	SFL 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
FSM 合計	265	398	149	63	223	281

表 3 状態圧縮の効果 [READ 操作 (正常時動作) の場合]
Table 3 State compaction performance. (READ operation in normal mode)

FSM 名	SAL 記述行数	PST 表現状態数	変換後状態数	状態の圧縮率 (%)	UDL/I 行数	SFL 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
FSM 合計	265	407	163	60	239	254

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ってハードウェアのモデルに変換した後の状態数を測定したものである．また，合成した UDL/I, SFL 記述の行数もあわせて示す．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表している．

表 2 および表 3 中の r_fpm と c_fpm の FSM について状態数圧縮の効果が大きい．これは，仕様記述のなかで，メッセージの送信側から受信側の FSM に対して，同一方向に連続して通信をするプロトコルが多いためである．

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．

6.3 断片的仕様からの合成

プロトコル処理 LSI 仕様においては，正常時の動作記述以外に，その数倍から十数倍の分量のエラー発生時の例外処理の仕様が与えられるという特徴がある．断片的に与えられるこれらのメッセージシーケンスから共通な動作記述部分を抽出しまとめることが，仕様から現実的な HDL 記述への変換を行う際に必要である．

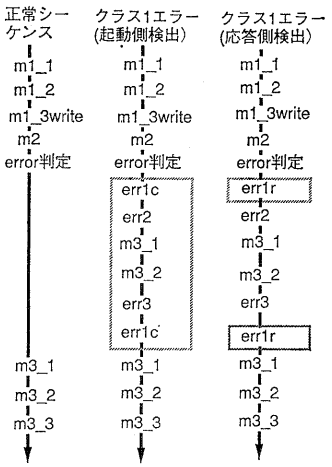

図 8 仕様マージ前のシーケンス
Fig. 8 Sequences to be merged in the FTAM WRITE operation.

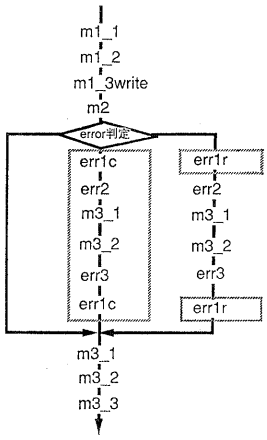

図 9 仕様マージ後のシーケンス
Fig. 9 Sequences after merge operation in the FTAM WRITE operation.

図 8 に示すような FTAM 仕様内の WRITE 処理を例に，正常時，回復可能なエラーを処理起動側で検出した場合，および，回復可能なエラーを応答側で検出した場合の三つの場合の動作シーケンスをマージして一つの FSM の動作記述を得る実験を行った．それぞれの共通部分を抽出することにより図 9 に示す動作シーケンスをもつ FSM を得た．マージの効果を確認するため，それぞれの動作シーケンス単独の記述から

表 4 仕様マージの効果 [WRITE 操作の場合]
Table 4 Merge operation performance (WRITE operation).

仕様名	SAL 記述行数	PST 表現状態数	圧縮後状態数	状態の圧縮率 (%)	UDL/I 行数	SFL 行数
正常シーケンス	265	398	149	63	223	281
クラス1エラー (起動側検出)	464	901	347	61	390	490
クラス1エラー (応答側検出)	473	919	354	61	398	499
単純合計	—	2218	850	62	1011	1270
マージ後 HDL 生成	—	1464	562	62	667	838

得られた PST の状態数，および，各場合の記述をマージ後得られた PST の状態数を**表 4** に示す．

正常時の動作シーケンスと2種類のエラー発生時の回復動作シーケンスのマージを試みた結果，エラー回復処理本体以外の前処理と後処理のシーケンスはほぼ共通しており一つの記述としてマージすることができた．このマージ処理により，各シーケンスごとに HDL を合成する方法と PST 状態数で比較し 34% の状態数を削減することができた．

エラーの起動側検出の場合と応答側検出の二つのシーケンスではエラー回復処理の内部も類似しているが，これ以上のマージ処理はできなかった．これは，現在使用している共通部分抽出アルゴリズムがマージ可能部分の先頭として受信イベントまたは条件分岐のみを許しているという制約からくるものである．

仕様記述段階で状態割り当てを意識せずに済み，メッセージシーケンスからシステムが自動で状態割り当てと状態遷移表の作成をするメリットは，小規模な記述対象よりも，むしろ，本論文で示したような状態数が数十以上の中規模の通信制御 FSM の設計で発揮される．仕様変更の際に新たな状態を挿入したり，状態名の書き換えの手間がすべて自動化されるので人的エラーの発生する余地が減少し上位設計の効率化に役立つものである．

7. おわりに

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ェアを合成する手法を提案した．本手法には，従来の上位設計手法では困難であった HDL 変換前の仕様段階で，実行可能性検証や仕様の矛盾検出が可能となるという利点がある．

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，ハードウェ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．

ここでは，LSI 上位仕様の記述手段としてメッセージシーケンスレベルの記述を用いた．この記述では，上位仕様の記述に混在するイベント間の実時間制約を記述しないため，リソース競合やタイムアウトに起因する実行可能性のエラーを検出することはできないとの問題がある．この改善は今後の課題である．

謝辞 日頃，有益な助言をいただく LSI 研究所設計システム研究部 安達徹氏に感謝します．また，SAL の言語とモデルについて指導をいただいた NTT ソフトウエア研究所 市川晴久氏，加藤順氏，荒川則泰氏に感謝します．

参考文献

1) Camposano, R. and Wolf, W.: *High-Level LSI Synthesis*, Kluwer Academic Publishers (1991).
2) Borriello, G. et al.: Synthesizing Transducers from Interface Specifications, *VLSI '87*, pp. 403-418 (1988).
3) Nestor, J. A. et al.: Behavioral Synthesis with Interfaces, *ICCAD '86*, pp. 112-115 (1986).
4) Borriello, G.: *Specification and Synthesis of Interface Logic*, Kluwer Academic Pubulishers (1991).
5) Barbacci, M. R.: Instruction Set Processor Specifications (ISPS): The Notation and Its Applications, *IEEE Trans. Comput.*, Vol. C-30, No. 1, pp. 24-40 (1981).
6) Vanbekbergen, P. et al.: A Generalized State Assignment Theory for Transformations on Signal Transition Graphs, *ICCAD '92*, pp. 112-117 (1992).
7) Lin, K. J. et al.: On the Verification of State-Coding in STGs, *ICCAD '92*, pp. 118-122 (1992).
8) Zafiropulo, P. et al.: Towards Analyzing and Synthesizing Protocols, *IEEE Trans. Comm.*, Vol. COM-28, No. 4, pp. 651-660 (1980).
9) Gouda, M. G. et al.: A Technique for Proving Liveness of Communicating Finite State Machines with Examples, *Proc. ACM Symp. on Principles of Distributed Computing 3rd*, pp. 38-49 (1984).
10) ISO: OSI-LOTOS—A Formal Description Technique Based on the Temporal Ordering of Observational Behevior, ISO 8807 (1989).
11) CCITT: SDL Recommendation Z 100-Z 104

(1984).
12) Ichikawa, H. et al.: Communications Software Management with Verification and Transformation into CCITT Specification and Description Language, *The Transaction of the IECE of Japan*, Vol. 69-B, No. 4, pp. 524-535 (1986).
13) Mano, T., Maruyama, F., Hayashi, K., Kakuda, T., Kawato, N. and Uehara, T.: OCCAM to CMOS Experimental Logic Design Support System, *7th Computer Hardware Description Languages and Their Applications (CHDL '85)*, pp. 381-390 (Aug. 1985).
14) Maruyama, F., Mano, T., Hayashi, K., Kakuda, T., Kawato, N. and Uehara, T.: Prolog-Based Expert System for Logic Design, *Proc. of the International Conference on Fifth Generation Computer Systems 1984*, pp. 563-571 (1984).
15) Stoll, A. and Duzy, P.: High-Level Synthesis from VHDL with Exact Timing Constraints, *Proc. 29th DAC*, pp. 188-193 (June 1992).
16) Wolf, W., Takach, A., Huang, C. Y., Manno, R. and Wu, E.: The Princeton University Behavioral Synthesis System, *Proc. 29th DAC*, pp. 182-187 (June 1992).
17) Gajiski, D., Dutt, N., Wu, A. and Lin, S.: *High-Level Synthesis—Introduction to Chip and System Design*, Kluwer Academic Publishers (1992).
18) 市川晴久ほか：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，Vol. 70-B, No. 5, pp. 565-575 (1987).
19) 加藤 順ほか：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, Vol. 38, No. 11, pp. 1249-1256 (1989).
20) 伊藤正樹ほか：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，Vol. 69-B, No. 5, pp. 449-459 (1986).
21) Hopcroft, J. E.: An $n \log n$ Algorithm for Minimizing States in a Finite Automaton, *Theory of Machines and Computations*, Academic Press, Now York (1971).
22) Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, MA (1974).
23) UDL/I Committee: UDL/I Language Reference Manual Version 2.0.1, Japan Electric Industry Development Association (1993).
24) Nakamura, Y. et al.: An RTL Behavioral Description Based Logic Design CAD System with Synthesis Capability, *IFIP CHDL '85* (1985).
25) 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 S 004 (V 2.0) FTAM 実装規約 (1990).

(平成5年 3 月 1 日受付)
(平成5年11月11日採録)

小林 哲雄（正会員）

1985年北海道大学工学部電気工学科卒業．1987年同大学大学院電気工学専攻修士課程修了．同年 NTT 入社．以来，NTT LSI 研究所において，オブジェクト指向モデルによる機能シミュレーション，並列計算機の性能評価，LSI 上位設計支援技術の研究開発に従事．現在，NTT 交換システム研究所伝達システム研究部研究主任．ACM 会員．

宮崎 敏明（正会員）

昭和56年電気通信大学電気通信学部応用電子工学科卒業．昭和58年同大学大学院修士課程修了．同年日本電信電話公社入社．以来，厚木電気通信研究所（現 NTT LSI 研究所）において，信号処理 LSI 用 CAD，CAD フレームワーク，論理設計支援技術および上位設計支援技術の研究開発に従事．現在 NTT 伝送システム研究所伝送処理研究部主任研究員．IEEE，電子情報通信学会各会員．

星野 民夫（正会員）

1976年群馬大学大学院工学研究科電子工学専攻修士課程修了．同年日本電信電話公社（現在 NTT）に入社．以来，NTT 研究所において，設計言語，設計データベース，論理合成システムなどの研究を行う．現在 NTT LSI 研究所設計システム研究部主幹研究員（グループリーダ）．IEE，電子情報通信学会各会員，(社)日本電子工業振興協会の LSI 設計用記述言語標準化委員．1985年計算機ハードウエア記述言語国際シンポジウム (CHDL 85) において最優秀論文賞受賞．