3N-1　Prologを用いた論理設計の検証

藤田　昌宏　、　田中　英彦　、　元岡　達
（東京大学　工学部）

1．　はじめに

筆者らは、機能レベル以上のレベルから、階層設計を支援するCADシステムについて提案を行ない＜1＞、シミュレータを組み入れたシステムを作成した＜2＞。　ここでは、CADシステム内の設計検証について、以下の点を考慮し、いくつかの例について検討を行なったので、報告する。

①階層設計を支援する。
②仕様記述にテンポラルロジック＜3、4＞を用いる。
③上位レベルと下位レベル間の検証を行なう。
④Prolog／KR＜5＞を用いて作成する。

2．　CADシステムの構成

全体の設計の流れを、data-trnsfer system を例にとって、図1に示す。設計は階層的に行なわれ、1つの設計サイクルは、

①テンポラルロジックによる仕様記述
②サブモジュールにモジュール分け、または、
　DDLやprimitive gates による設計
③②のサブモジュールや設計が上位
　レベルの仕様を満たしているかの検証
と、なる。

このサイクルは、全てのモジュールがDDLやprimitive gates で記述されるまで続けられる。　検証は、アサーションとその答えという形で行なわれる。（図2）

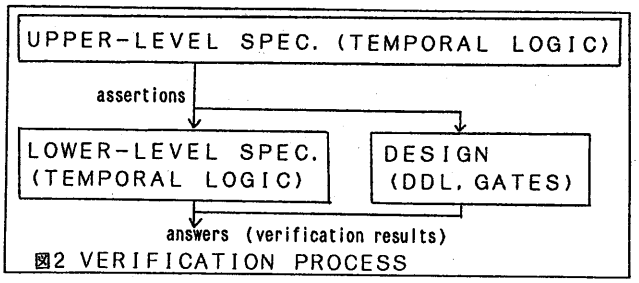

図2 VERIFICATION PROCESS

3．　ハードウェアの仕様記述、及び、設計の例……ハンドシェイク（図1）

ハンドシェイクによるデータ伝送を例にとって説明すると、top-level では、仕様として、適当なinitial-condition が成り立てば、データが正しく伝送されることを記述する。　2nd-levelの設計を行なう時にハンドシェイクについての知識を用いて、テンポラルロジックによる仕様（設計）を得る。　3rd-levelでは、DDLによる設計を行なっている。また、もし必要なら、4th-levelの設計として、gateによる設計を行なう。

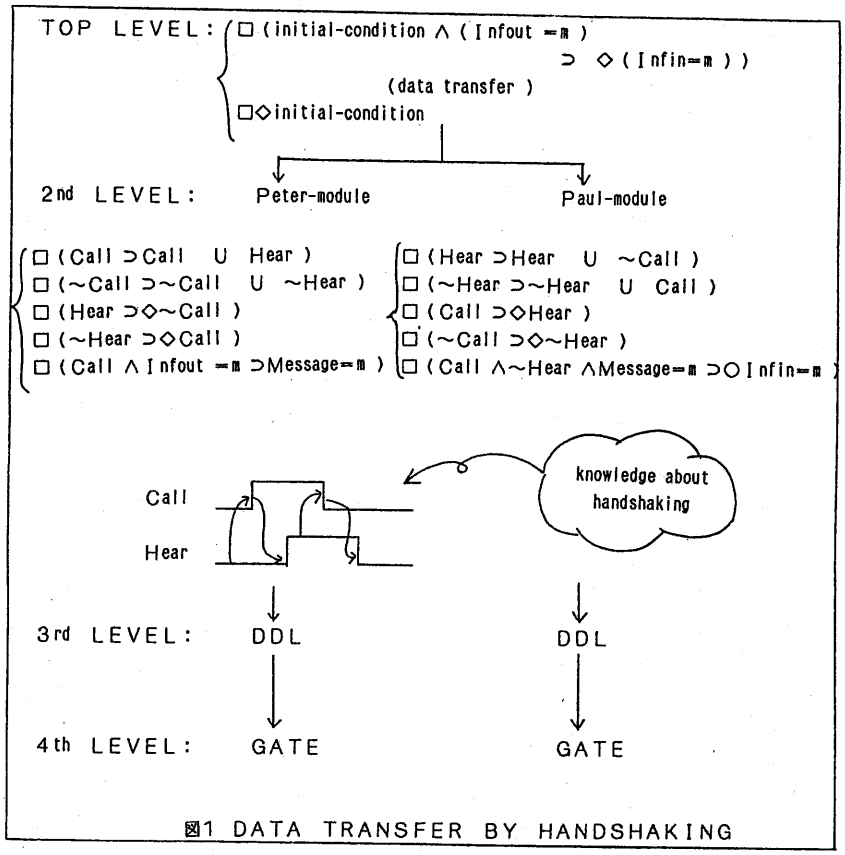

図1 DATA TRANSFER BY HANDSHAKING

4．　Prologを用いた、DDL、及び、ゲートレベルのベリフィケーション

次の6つのタイプのアサーションについて、処理プログラムを作成し、検討した。意味については参考文献＜4＞参照。
□（A⊃○B）、□（A⊃◇B）、□（A⊃□B）、
□（A⊃□◇B）、□（A⊃◇□B）、□（A⊃BUC）

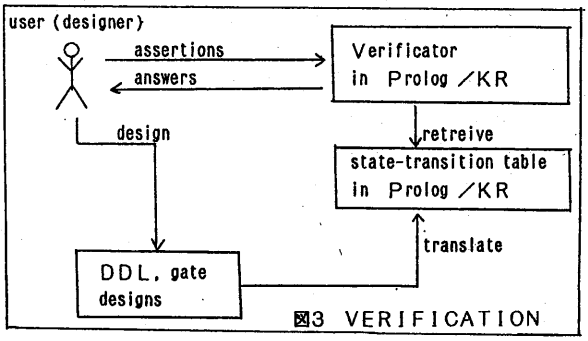

図3 VERIFICATION

これらを用いて様々なタイミングチャートを表わすことができる。 assertion は、forward reasoning でも、backward reasoningでもどちらでも処理できる。

①ＤＤＬ記述やprimitive gates による記述をＰrolog ／ＫＲのstate transition tableに直す。つまり、現在の状態と次の状態との関係に直す。

②①のtable を用いてシステム全体の状態遷移を可能にする。

③背理法で証明する。

以上を図に示すと、図３（前ページ）のようになる。

```
(ASSERT (VERI11 *0 *1) (STTRAN *0 *1) (VERIIII *0 *1))
(ASSERT (VERIIII *B *P)
        (STTRAN *P *A) (IF (LOGICB *P) (FALSE))
        (IF (STEQU *B *P) (AND (PRINT (*B TYPE<) *P)) (FALSE)))
        (VERIIII (*P *B) *A))
(ASSERT (VERIBK11 *0 *1) (STTRAN *1 *0) (VERIBK1 *0 *1))
```

図4 FORWARD REASONING

・forward reasoning

初期状態から始め、ループか、エラーが起こるまで次の状態を求めていくことを繰り返す。もしループか、エラーが起こったら、強制的にbacktrack をかけ、全ての状態遷移について調べる。 A⊃◇B について、図４にＰrolog ／ＫＲによるプログラムを示す。この他のアサーションも同じように処理できる。

・ ゲートレベルのベリフィケーション

data-transfer systemのＰaul-moduleについて、ベリフィケーション例を示す。図５がゲート回路のＰrolog ／ＫＲでの表現である。ここで、組み合せ回路にはディレイがないとする。図５中の変数のうち、“@”で始まるものは次の状態における各変数の値であり、その以外のものは今の状態における変数の値である。

```
:(VERI11 (*PETER *PAUL *CALL 1 *INFOUT *INFIN) *1)
(((HY CY 1 1 0 0) (HY CN 1 1 0 *INFIN_3487)) TYPE<) (HY CY 1 1 0 0))
((HY CY 1 1 0 *INFIN_3487) TYPE<) (HY CY 1 1 0 *INFIN_3487))
NIL                                                    反例
:(VERI11 (HY CN 0 1 *INFOUT *INFIN) *1)
NIL
```

図8 EXAMPLE Hear ⊃◇~Call

```
(ASSERT (FDFF 0 0 1 0 1))
(ASSERT (FDFF 1 0 1 1 0))
(ASSERT (FDFF 0 1 0 0 1))
(ASSERT (FDFF 1 1 0 1 0))
          D入力  Q  Q̄  Qn  Q̄n
```

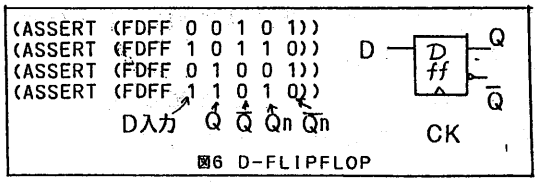

図6 D-FLIPFLOP

現在の状態の値と、次の状態の値は、図６のように、D-flipflop （Ｐrolog ／ＫＲでは、ＦＤＦＦ）のtable によって互いに結びつけられている。table の第一項は、Ｄ－入力を示し、次の二項は、現在の内部状態を示し、最後の二項は、次の内部状態を示している。組み合せ回路は、基本ゲートを用いて記述される。 これで、現在の状態と次の状態の関係がＰrolog ／ＫＲの記述として得られた。図７にＰaul-moduleのspecification をassertion として用いた、ベリフィケーション例を示す。

```
:(VERI51 (*MESSAGE *CALL 1 *INFIN *CALL_YES *CALL_NO) *1)
((0 0 1 0 1 0) (0 1 0 0 0 1))
((0 0 1 0 1 0) (1 1 0 0 0 1))
((0 0 1 1 1 0) (0 1 0 0 0 1))
((0 0 1 1 1 0) (1 1 0 0 0 1))
((0 0 1 0 0 1) (0 1 0 0 0 1))
((0 0 1 0 0 1) (1 1 0 0 0 1))   反例……Call の立ち上りでは、
((0 0 1 1 0 1) (0 1 0 0 0 1))   満たされないことが分かる。
((0 0 1 1 0 1) (1 1 0 0 0 1))
((1 0 1 0 1 0) (0 1 0 0 0 1))
((1 0 1 0 1 0) (1 1 0 0 0 1))
((1 0 1 1 1 0) (0 1 0 0 0 1))
((1 0 1 1 1 0) (1 1 0 0 0 1))
((1 0 1 0 0 1) (0 1 0 0 0 1))
((1 0 1 0 0 1) (1 1 0 0 0 1))
((1 0 1 1 0 1) (0 1 0 0 0 1))
((1 0 1 1 0 1) (1 1 0 0 0 1))
NIL
:
```

図7 EXAMPLE Hear ⊃Hear U~Call

・ ＤＤＬレベルのベリフィケーション

ゲートレベルの時と同じように、まず、現在の状態と次の状態の関係を表わすＰrologのプログラムを作る。これは、ＤＤＬの場合、forward reasoning の方はすぐに得られ、backward reasoningの方は、ＤＤＬ translatorを用いるとすぐに得られる。ベリフィケーションの方法は、ゲートレベルの時と同じである。図に例を示す。

・ 混合レベルのベリフィケーション

ＤＤＬ記述と、gateによる設計の両方を持ったモジュールの検証も可能である。

```
(ASSERT (PAULC (*MESSAGE *CALL *HEAR *INFIN *CALL_YES *CALL_NO)
               (*@MESSAGE *@CALL *@HEAR *@INFIN *@CALL_YES *@CALL_NO))
        (FAND *MESSAGE *CALL *N4) (FAND *N4 *CALL_NO *N5)
        (FDFF *N5 *INFIN *~INFIN *@INFIN *@~INFIN)
        (FDFF *CALL *CALL_YES *CALL_NO *@CALL_YES *@CALL_NO)
        (FAND *CALL_YES *HEAR *N1) (FOR *N1 *CALL_NO *N2) (FAND *CALL *N2 *N3)
        (FDFF *N3 *HEAR *~HEAR *@HEAR *@~HEAR))
```

図5 GATE DESIGN OF PAUL

5. 最後に

Ｐrolog ／ＫＲとテンポラルロジックを用いた、論理設計の検証について、具体例を用いて説明を行なった。ここで示した方法により、仕様記述のレベルから、ゲートレベルまで、かなり統一的に検証を行なっていくことができる。 今後、ここで述べた手法を用いた論理設計検証システムを作成し、評価、及び、高能率化を進めていくと共に、ＤＤＬの自動合成も検討していく予定である。

＊参考文献

＜１＞藤田 他、『入出力に図形出力を用いた要求仕様及び、…』、第２３回情処全大

＜２＞藤田 他、『入出力に図形表示を用いた機能レベル…』、第２４回情処全大

＜３＞上原 他、『テンポラルロジックによる表明と機能設計の検証』、第２４回情処全大

＜４＞P. Wolper : “Temporal Logic Can Be More Expressive”, 22nd FCS,1981

＜５＞中島、『Prolog ／KR User ´s Manual』、東京大学、METR82－4、1982