3L-11　ＤＤＬからＰｒｏｌｏｇ への自動変換プログラム

藤田　昌宏、　田中　英彦、　元岡　達

（東京大学　工学部）

１．　はじめに

我々は既に、時相論理を用いてハードウェアの仕様を記述し、階層設計を支援する論理設計検証システムをＰｒｏｌｏｇ を用いて作成することを提案し、検証手法を示した［１、２］。本手法では、設計は全て、現在と次の時刻の関係表としてＰｒｏｌｏｇ に変換する。ここでは、状態遷移レベルのハードウェア記述言語であるＤＤＬ［３］のサブセットＤＤＬ－Ｓの記述をＰｒｏｌｏｇ ／ＫＲ［４］の記述に自動変換するプログラムをＰｒｏｌｏｇ ／ＫＲで作成したので報告する。

２．　ＤＤＬ－Ｓ

システムは、一般に、ＡＬＵのように実際に計算を行う演算部（function part ）と、各端子間のデータ転送のタイミングを扱う同期部（synchronization part ）に分けられる。ＤＤＬ－ＳはＤＤＬのサブセットであり、同期部の記述に重点をおいている。シンタックスの主要部を付録１に示す。主な制限は、変数をcontrol とdataに分け、data変数は制限しないがcontrol 変数は１ビットのみとし、ＩＦ文の条件に現れるのはcontrol 変数のみにし、また、ＡＮＤ、ＯＲ等の演算はcontrol 変数には認めるがdata変数には認めないようにしている点である。これらは、記述対象を主に同期部にしぼることを意味する。

図１に示すデータ転送システムについて、状態遷移（ＤＤＬ）レベルの記述を図２に、これをＤＤＬ－Ｓで記述したものを図３に示す。ＤＤＬ－Ｓはリストで表現され、この例の場合２つのオートマトンを記述している。図中、（：<－　ＣＡＬＬ　１）は、control 変数ＣＡＬＬに定数１をデータ転送することを示す。また、ＩＦ文はＩＦの次のリストが条件、その次が then-part、最後がelse-part となっている。

３．　変換法

ＤＤＬ－Ｓの記述は、各オートマトン単位に、各変数に対し現在と次の時刻の関係としてＰｒｏｌｏｇ に変換する。ＩＦ文の条件に使える変数を１ビットに制限したため、変換は容易に行なえる。

図３のようなＤＤＬ－Ｓの記述のうち、Ｓｅｎｄｅｒについては図４のようにＰｒｏｌｏｇ ／ＫＲに変換される。図で＃ではじまるものは変数であり、@のついているものは次の時刻のもので、ないものは現在の時刻のものである。図４中ＦＥＱは変数がある値と等しいかどうかに関する、また、ＴＲＡＮはデータ転送に関するシステムプレディケイトである。

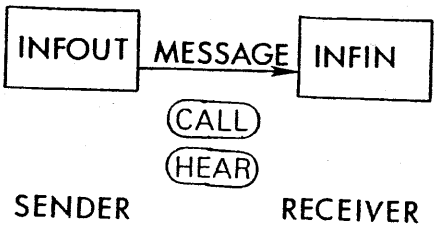
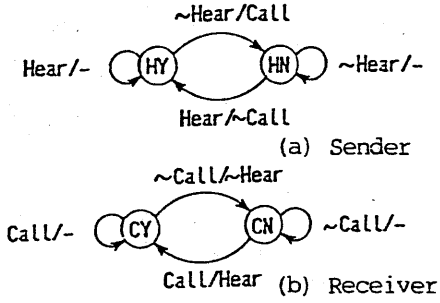


fig.1 data-transfer system




```
LOGIC (in all states):
if Call=1 then Message := Infout

LOGIC:
if Call=1 and Hear=1
        then Infin <- Message
```

fig.2 DDL level design of Sender and Receiver

```
((SYSTEM DATA-TRANSFER)
  (DCL ((CONTROL-REGISTER (CALL HEAR)) (DATA-TERMINAL (MESSAGE))))
  (((AUTOMATON SENDER)
    (DCL ((STATE-NAME (HY HN)) (DATA-REGISTER (INFOUT))))
    ((LOGIC (IF (== CALL 1) (:= MESSAGE INFOUT)))
     (HY (IF (== HEAR 1) (:-> HY) (DO (:<- CALL 1) (:-> HN))))
     (HN (IF (== HEAR 0) (:-> HN) (DO (:<- CALL 0) (:-> HY))))))
   ((AUTOMATON RECEIVER)
    (DCL ((STATE-NAME (CY CN)) (DATA-REGISTER (INFIN))))
    ((LOGIC (IF (AND (== CALL 1) (== HEAR 1)) (:<= INFIN MESSAGE)))
     (CN (IF (== CALL 0) (:-> CN) (DO (:<- HEAR 1) (:-> CY))))
     (CY (IF (== CALL 1) (:-> CY) (DO (:<- HEAR 0) (:-> CN))))))))
```

```
SSERT (SENDER (#STATE #CALL #HEAR #INFOUT)
              (#@STATE #@CALL #@HEAR #@INFOUT)
              (#CL #MESSAGE))
    (IF (EQ #CL 0)
        (TRUE)
        (AND (OR (AND (FEQ #CALL 1) (TRAN #MESSAGE #INFOUT)) (FEQ #CALL 0))
             (OR (AND (= #STATE HY) (OR (AND (FEQ #HEAR 1) (= #@STATE HY))
                                        (AND (FEQ #HEAR 0) (AND (TRAN #@CALL 1) (= #@STATE HN)))))
                 (AND (= #STATE HN) (OR (AND (FEQ #HEAR 0) (= #@STATE HN))
                                        (AND (FEQ #HEAR 1) (AND (TRAN #@CALL 0) (= #@STATE HY)))))))))
```

fig.4 Sender in Prolog/KR translated from DDL-S

変換の主な作業は、次のようになる。

① オートマトン名をプレディケイト名とし、引数として、外部ターミナル、レジスタ、内部状態等を付ける。この際、次の形にする。

(automaton 名
レジスタと内部状態の現在の値のリスト
レジスタと内部状態の次の値のリスト
外部ターミナルの現在の値のリスト)

つまり、現在と次の時刻の間の各変数の関係とする。

② オートマトン内の状態の記述が現れるごとにORで、1つの状態内の各処理をANDで結び、バックトラックにより全ての状態を調べられるようにする。このように、Prolog /KRにORがあるため変換が容易になっている。(通常のProlog に変換することも可能である。)

③ actionごとに変換する。レジスタへのデータ転送は、そのレジスタの次の時刻の変数(#の後に@のついているもの)と転送元の変数をパターン照合させ、ターミナルへのデータ転送は、そのターミナルの現在の変数(@のないもの)とパターン照合させる。IF文の処理は、then側とelse側にそれぞれ、then側の満すべき条件(IF文の条件)、else側の満すべき条件(then側の条件の否定)をANDで結んだものをORで結ぶように変換し、バックトラックした時にthen側、else側どちらのパスもとるようにして、全ての場合処理できるようにしている。

④ クロックに対する処理をつけ加える。

(IF (= #CL 0) (TRUE) …)

をつけ加えて、そのモジュールにクロック(#CL)がこなければ(#CL=0)何もせず、もしくれば(#CL=1)状態遷移をするようにしている。

この変換プログラムもProlog /KRでかかれており(約300行)、処理時間は、図3を変換する場合M-200Hで約0.1秒である。

4. おわりに

同期部の記述に重点をおいた状態遷移レベルのハードウェア記述言語DDL-Sを設定し、その記述のProlog /KRの記述への自動変換プログラムについて述べた。Prolog での表現は、ゲート回路に対するものと同じになっており、DDL-Sだけでなくゲート回路と混合したものも検証することができる。

5. 参考文献

[1] 藤田他、第26回情報処理全国大会3K-1
[2] 濱中他、第26回情報処理全国大会3K-2
[3] IEEE tran. computer C-17 pp850 ～
[4] 中島"Prolog /KR User 's Manual"

Appendix DDL-S syntax

```
ddl-des ::= (system-def declare-list descriptions)
system-def ::= (SYSTEM system-name)
declare-list ::= (DCL ({declare}*))
declare ::= (CONTROL-REGISTER control-register-list) |
            (DATA-REGISTER data-register-list) |
            (CONTROL-TERMINAL control-terminal-list) |
            (DATA-TERMINAL data-terminal-list)
descriptions ::= {automaton-des}*
automaton-des ::= (automaton-def declare-list-in-automaton
                   automaton-des-body)
automaton-def ::= (AUTOMATON automaton-name)
declare-list-in-automaton ::=(DCL({declare-in-automaton}*))
declare-in-automaton ::=
          (CONTROL-REGISTER control-register-list) |
          (DATA-REGISTER data-register-list) |
          (CONTROL-TERMINAL control-terminal-list) |
          (DATA-TERMINAL data-terminal-list) |
          (STATE-NAME state-name-list)
automaton-des-body ::= ({logic-des} {state-des}**)
logic-des ::= (LOGIC action)
state-des ::= (state-name action)
action ::= (:<= data-register source2) |
           (:= data-terminal source2) |
           (:<- control-register sourcel) |
           (:- control-terminal sourcel) |
           (IF condition action-then action-else) |
           (DO {action}*) |
           (:-> state-name)
action-then ::= action
action-else ::= action
sourcel ::= bit | control-register | control-terminal
source2 ::= const | data-register | data-terminal
condition ::= (NOT condition) |
              (AND {condition}*) |
              (OR {condition}*) |
              (== control-var bit) |
              (== control-var control-var)
control-var ::= control-register | control-terminal
bit ::= 0 | 1
```