論理設計の検証（DDLベリファイア）

上原貴夫　川戸信明　斉藤隆夫　広瀬貞樹　丸山文宏
（富士通研究所）

1. はじめに [1),2)]

新しいアーキテクチャを時機を失せずにVLSI化するためには，誤りのない論理設計を短期間で仕上げることが必要である。一般に，誤りのない設計を実現するためには，その初期の段階で充分な検証を行ってから次の段階へ進むことが大切だといわれている。論理設計においては，回路設計（GATE LEVEL DESIGN）へ進む前に，機能設計（REGISTER TRANSFER LEVEL DESIGN）の段階で検証を行うことが重要である。以下では，形式的方法により機能設計の正しさを証明するDDLベリファイアについて，その原理と実用性の評価について述べ，さらにテンポラルロジックによる仕様記述とその検証にも言及する。

2. DDLベリファイアの原理 [3),4),5),6)]

シミュレータは設計ミスの発見には有効であるが，それが存在しないことを示すには不向きである。ベリファイアは正しい設計が満すべき表明を与えると，これを論理的に証明するものである。図1にDDLベリファイアの原理を示した。

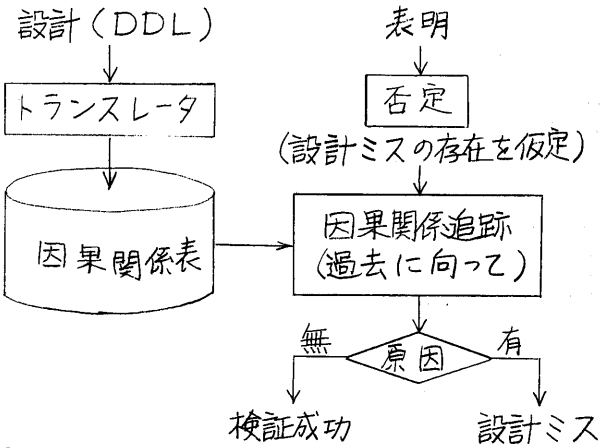
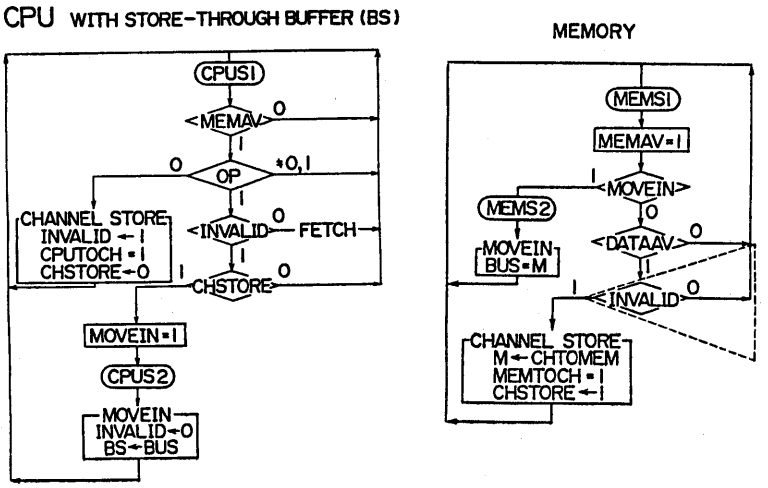


図1　DDLベリファイアの原理

機能設計のDDL記述は，トランスレータにより因果関係を表わす表にまとめられる。使用者が証明すべき表明を論理式で与えると，ベリファイアは背理法を用いて証明を行う。まず，表明の否定をとり，設計ミスが存在して表明に反することが起ったと仮定する。つぎに，この仮定された状況を最終状態として，因果関係を参照しながら時間を過去にさかのぼり，原因となる初期状態が存在するかどうか調べる。その結果，原因が在り得ないことが示せれば，仮定が否定され，最初に与えられた表明が満されていることの証明になる。万一，原因が存在すれば，それは設計ミスに他ならない。

3. 検証例 [6)]

図2にバッファ記憶をもつコンピュータの機能設計の例を示した。これは，CPUとMEMORYとの2つの部分から成る。一般に，ユニット間のインタフェースには設計ミスが生じやすく，ベリファイアが検証の効果を発揮する。図3は，トランスレータによって得た因果関係表であり，その内容の一般的な解釈を図4に示した。図5は，表明の例であり，その検証過程を図6に示した。




```
00010     <SYSTEM> SAMPLE:
00020       <TIME> P<(100)>.
00030         <ENTRANCE> DATAAV,CHTOMEM(16).
00040         <TERMINAL> CPUTOCH,MEMTOCH,MEMAV,BUS(16),MOVEIN,FETCH.
00050         <REGISTER> OP(16),INVALID,CHSTORE.
00060         <AUTOMATON> CPU: P:
00070           <REGISTER> BS(16).
00075           <BOOLEAN>  FETCH=CPUS1 & MEMAV & (OP(0:1):=1) & ~INVALID.
00080           <STATES>
00090             CPUS1: MEMAV: ? OP(0:1) #0   INVALIDX-1,
00100                                          CPUTOCH=1,
00110                                          CHSTORE<-0,
00120                                          ->CPUS1
00130                                    /* CHANNEL STORE */
00140                                  #1   :* INVALID *:
00150                                          :* CHSTORE *:
00160                                             MOVEIN=1,
00170                                             ->CPUS2;
00180                                             ->CPUS1.;
00190                                          ->CPUS1.
00200                                    /* FETCH */
00210                                  ;    ->CPUS1..
00220             CPUS2: INVALIDX-0,BS<-BUS,->CPUS1.
00230           <END>.
00240         <END> CPU.
00250         <AUTOMATON> MEMORY: P:
00260           <REGISTER> M(16).
00270           <STATES>
00280             MEMS1: MEMAV=1,:* MOVEIN *: ->MEMS2;
00290                                :* DATAAV & INVALID *:
00300                                   M<-CHTOMEM,
00310                                   MEMTOCH=1,
00320                                   CHSTORE<-1.,
00330                                ->MEMS1..
00340             MEMS2: BUS=M,->MEMS1.
00350           <END>.
00360         <END> MEMORY.
00370     <END> SAMPLE.
```

図2　機能設計の例

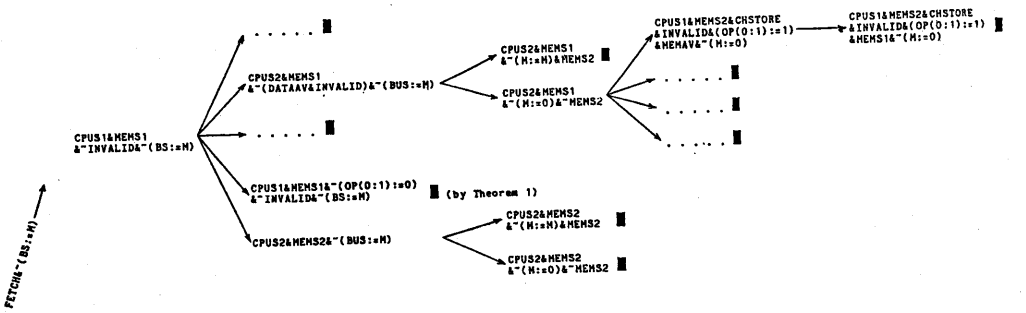


図6　検証過程

SYSTEM : SAMPLE

1 BUS	(0: 15)	16 BIT(S) TERMINAL
NO. : SINK RANGE :	SOURCE :	CONNECTION CONDITION
1- 1 : (0:15)	: MEMORYSIM(0:15)	: MEMORYSMEMS2

2 CPUTOCH	(0: 0)	1 BIT(S) TERMINAL
NO. : SINK RANGE :	SOURCE :	CONNECTION CONDITION
2- 1 : (0)	: 1	: (OP(0:1) := 0) & (CPUSCPUS1 & MEMAV(0))

3 FETCH	(0: 0)	1 BIT(S) TERMINAL
NO. : SINK RANGE :	SOURCE :	CONNECTION CONDITION
3- 1 : (0)	: CPUSCPUS1(1) & MEMAV(0)) & (OP(0:1) := 1) & ~ INVALID(0)	

4 MEMAV	(0: 0)	1 BIT(S) TERMINAL
NO. : SINK RANGE :	SOURCE :	CONNECTION CONDITION
4- 1 : (0)	: 1	: MEMORYSMEMS1

5 MEMTOCH	(0: 0)	1 BIT(S) TERMINAL
NO. : SINK RANGE :	SOURCE :	CONNECTION CONDITION
5- 1 : (0)	: 1	: (DATAAV(0) & INVALID(0)) & ~MOVEIN(0) & MEMORYSMEMS1

6 MOVEIN	(0: 0)	1 BIT(S) TERMINAL
NO. : SINK RANGE :	SOURCE :	CONNECTION CONDITION
6- 1 : (0)	: 1	: CHSTORE(0) & INVALID(0) & (OP(0:1) := 1) & (CPUSCPUS1 & MEMAV(0))

7 CHSTORE	(0: 0)	1 BIT(S) REGISTER	
NO. : SINK RANGE :	SOURCE :	TRANSFER CONDITION	: CLOCK
7- 1 : (0)	: 0	: (OP(0:1) := 0) & (CPUSCPU S1 & MEMAV(0))	: P
7- 2 :	: 1	: (DATAAV(0) & INVALID(0)) & ~MOVEIN(0) & MEMORYSMEM S1	: P

8 CHTOMEM	(0: 15)	16 BIT(S) REGISTER

9 DATAAV	(0: 0)	1 BIT(S) REGISTER

10 INVALID	(0: 0)	1 BIT(S) REGISTER	
NO. : SINK RANGE :	SOURCE :	TRANSFER CONDITION	: CLOCK
10- 1 : (0)	: 0	: CPUSCPUS2	: P
10- 2 :	: 1	: (OP(0:1) := 0) & (CPUSCPU S1 & MEMAV(0))	: P

11 OP	(0: 15)	16 BIT(S) REGISTER

図3 因果関係表

SYSTEM : SAMPLE
AUTOMATON: MEMORY

NO. :	NEXT STATE :	PRESENT STATE :	TRANSITION CONDITION
12- 1 :	MEMS1	MEMS1	: ~MOVEIN(0)
12- 2 :		MEMS2	
13- 1 :	MEMS2	MEMS1	: MOVEIN(0)

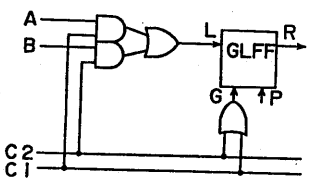
14 MEMORYSM	(0: 15)	16 BIT(S) REGISTER	
NO. : SINK RANGE :	SOURCE :	TRANSFER CONDITION	: CLOCK
14- 1 : (0:15)	: CHTOMEM(0:15)	: (DATAAV(0) & INVALID(0)) & ~MOVEIN(0) & MEMORYSMEM S1	: P

SYSTEM : SAMPLE
AUTOMATON: CPU

NO. :	NEXT STATE :	PRESENT STATE :	TRANSITION CONDITION
15- 1 :	CPUS1	CPUS1	: (OP(0:1) := 0) & (MEMAV(0))
15- 2 :		CPUS1	: (~MEMAV(0))
15- 3 :		CPUS1	: ~(OP(0:1) := 0 : OP(0:1) := 1) & (MEMAV(0))
15- 4 :		CPUS1	: ~INVALID(0) & (OP(0:1) := 1) & (MEMAV(0))
15- 5 :		CPUS1	: ~CHSTORE(0) & INVALID(0) & (OP(0:1) := 1) & (MEMA V(0))
15- 6 :		CPUS2	
16- 1 :	CPUS2	CPUS1	: CHSTORE(0) & INVALID(0) & (OP(0:1) := 1) & (MEMAV (0))

17 CPUSBS	(0: 15)	16 BIT(S) REGISTER	
NO. : SINK RANGE :	SOURCE :	TRANSFER CONDITION	: CLOCK
17- 1 : (0:15)	: BUS(0:15)	: CPUSCPUS2	: P

R	BIT REGISTER
SOURCE	TRANSFER CONDITION
A	C1
B	C2



結果　　原因

R at time t ⇔ (A∧C1)∨(B∧C2)∨(R∧¬(C1∨C2)) at time t-1

図4 因果関係表の解釈

「読出時には，バッファの内容はメモリの内容と一致している。」

FETCH⊃ BS := M

図5 表明の例

4. 設計の信頼度 [7),8)]

図7にDDLを用いた論理設計の手順を示した．図8に，この手順に従って設計したものと，従来手法によるものとの信頼度を比較した．この例では，DDL記述を検証した段階で発見できなかった設計ミスは，1000IC当り4件であり，従来手法に比べて一桁少ないことがわかる．また，DDLベリファイアにより10件の検証を行ったが，いずれも高々1～2分（M180II CPU時間）で検証に成功した．

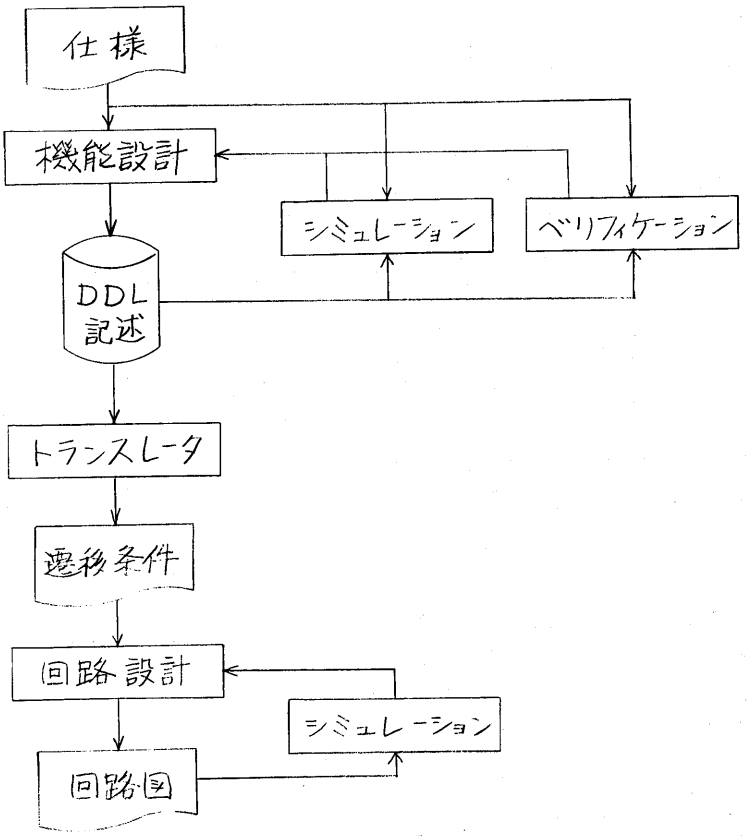


図7　DDLを用いた論理設計の手順

	DDL記述	回路図
サブシステム1（DDL使用）	0.004	0.016
サブシステム2（従来手法）	——	0.079

図8　残存設計ミス（単位…件/IC）

5. テンポラルロジック [9], [10], [11]

前記の実験の結果，DDLベリファイアで用いた背理法と後向推論は探索空間を限定する効果があり，大規模システムの検証が実行可能であることが明らかになった．一方，仕様記述に用いた命題論理は，ハードウェアの動作仕様を記述するのには不充分であった．そこで，仕様の記述にテンポラルロジックを導入することを提案した．これによって，「悪いことが起らない」こと（SAFETY）のみでなく，良いことが起る」こと（LIVENESS）を表明できるようになった．ここでは，例題を用いて，表明の与え方および背理法と後向推論による検証の原理を示す．

テンポラルロジックでは，□Pは「以後常にPが真である」こと，◇Pは「将来いつかPが真になる」ことを表わし，

$$\sim \Diamond P \equiv \Box \sim P$$

である．

［例1］SAFETYの例として，□～Pの検証を考える．背理法を用いることとし，表明を否定して◇Pと仮定する．もし，$\circ^n P$（n時刻後にP）であるための必要条件C_nが恒等的に偽であることが示せれば，仮定が否定され表明が検証されたことになる．

例えば，図5の表明は，□～(FETCH ∧ ～(BS := M)) なので，◇(FETCH ∧ ～(BS := M)) と仮定して，図6のように時刻を逆のぼりながらその必要条件をしらべている．

［例2］LIVENESSの例として，◇Pの検証を考える．背理法を用いることとし，表明を否定して□～Pと仮定する．もし，$\circ^n \sim P$であるための必要条件をC_nとしたとき，$C_0 \wedge C_1 \wedge C_2 \wedge \cdots \wedge C_n$が恒等的に偽であることが示せれば，仮定が否定され表明が検証されたことになる．

例えば，図2の設計において，メモリがいつか使用可能になることは，つぎのように表明できる．

$$\Diamond \mathrm{MEMAV}$$

これを背理法で証明するために，表明を否定し，

$$\Box \sim \mathrm{MEMAV}$$

すなわち，ずっと使用可能にならないと仮定する．

～MEMAVであるための必要条件をC_0とすれば，図3の4-1行より，

$$C_0 = \sim \mathrm{MEMS1}$$

である．さらに，～MEMS1であるための1時刻前における必要条件をC_1とすれば，図3の13-1行より，

$$C_1 = \mathrm{MEMS1} \wedge \mathrm{MOVEIN}$$

となる．したがって，恒等的に

$$\sim (C_1 \wedge C_2)$$

であるから，2時刻つづけて～MEMAVではあり得ないことがわかり，

$$\Diamond \mathrm{MEMAV}$$

であることが証明された．

［例3］次式は，メモリの使用可能性をより的確に表明している．

$$\Box \Diamond \mathrm{MEMAV}$$

例2の証明には，特定の時刻が現われなかったので，それは例3の証明になる．

6. おわりに

DDLシステムによる設計と検証に関して実験を行い，設計の信頼性を1桁上げるという目標の達成に対して明るい見通しを得た．今後，仕様記述の形式化と人工知能を応用した回路設計の自動化を推進することにより，VLSIの無誤性を達成できるものと考える．

参考文献

1) 三浦，上原：VLSIと高信頼化技法，情報処理，Vol 23，No. 4，pp. 283－291（1982）．
2) 上原：機能設計，情報処理，Vol. 22，No. 8，pp. 757－761（1981）．
3) KAWATO, SAITO, MARUYAMA and UEHARA: DESIGN AND VERIFICATION OF LARGE SCALE COMPUTERS BY USING DDL, 16TH DESIGN AUTOMATION CONFERENCE, pp. 375-381 (1979).
4) 丸山：ハードウェアの機能設計段階における検証，情報処理，Vol. 21，No. 5（1980）．
5) MARUYAMA, UEHARA, KAWATO and SAITO: HARDWARE VERIFICATION AND DESIGN DIAGNOSIS, FTCS 10, pp. 59-64 (1980).
6) UEHARA, MARUYAMA, SAITO and KAWATO: DDL VERIFIER, CHDL'81, pp. 51－61（1981）．
7) 丸山，川戸，上原：DDLベリファイアによる論理設計の検証とその評価，情報処理学会第23回全国大会，pp. 1071－1072（1981）．
8) 広瀬，川戸，丸山，斉藤：DDLシステムによるハード設計とその評価，57年度電通学会全国大会，pp 6.83（1982）．
9) 上原，丸山：TEMPORAL LOGICによるハードウェアの仕様記述と検証，情報処理学会第23回全国大会，pp. 1073－1074（1981）．
10) 上原，斉藤：テンポラルロジックによる表明と機能設計の検証，情報処理学会第24回全国大会，pp. 1203－1204（1982）．
11) 上原，斉藤：テンポラルロジックとハードウェア検証，57年度電通学会全国大会，pp. 6.82（1982）．