# 入力の一部が不明である場合の論理関数のハードウェアを用いた評価法

井口　幸洋† 笹尾　勤†† 松浦　宗寛†† 伊勢野　総†

# Evaluation of Logic Functions Using Hardware in the Presence of Unknown Inputs

Yukihiro IGUCHI†, Tsutomu SASAO††, Munehiro MATSUURA††, and Atsumu ISENO†

**あらまし**　論理シミュレーションを行うとき，入力の一部が 0 か 1 か不明なままで 2 値論理関数を評価したいことがある．このような場合，再収斂がある回路では，旧来の論理シミュレーションでは必ずしも正確な値を求めることはできない．しかし，与えられた 2 値論理関数に対応した正則 3 値論理関数を評価すれば正確に計算できる．本論文では，正則 3 値論理関数の 2 線式論理での構成法を提案する．提案する実現法は，$n$ 入力論理関数をシミュレーションするのに $O(2^n/n)$ 個の論理セルと $O(n)$ の計算時間を必要とする．我々はソフトウェアシミュレーションの約 100 倍高速な FPGA での実現法を示す．

**キーワード**　論理シミュレーション，3 値論理，Kleene 論理，FPGA

## 1. まえがき

ディジタルシステムの設計過程において，論理シミュレーションによる設計検証は最も重要なステップの一つであり，時間を要するステップでもある [1]〜[3].

論理シミュレーションを行うとき，入力の一部が 0 か 1 か不明（$u$: unknown）な場合に 2 値論理関数を評価したい場合がある [4]．例えば，入力の一部が出力には影響を与えない場合，$u$ は $u$ のまま取り扱いたい．また，回路の初期化のときにも $u$ を取り扱わなければならない． この場合，旧来のキューブに基づく論理シミュレータでは，大規模な回路に対してシミュレーションスピードが遅いという問題がある．また，再収斂（れん）がある回路では必ずしも正確な値が計算できないという問題もある [5]．正確に出力値を計算するには，与えられた 2 値論理関数に対応した正則 3 値論理関数（RT 関数：regular ternary function）を評価すればよいことが知られている [5].

BDD（binary decision diagram）に基づくサイクル・ベース・シミュレータは，旧来の方法に比べ高速ではあるが，出力値を正確に計算するには $u$ を 0 又は 1 に置換してすべての場合をつくさなければならず，$u$ の個数が増えるとシミュレーション時間は指数関数的に増大する [23]．これらに対して Jennings によって提案された Kleene_TDD（Kleene ternary decision diagram）[8] を用いれば，速度の低下なしに正確な値を求められる [23]．しかし，BDD に基づくシミュレータに比べ記憶容量を多く必要とするため，分解などの手法を使わないと実用化が難しい [9].

高速化技法としては，専用ハードウェアによるシミュレーションが有望である．シミュレーション専用エンジンとしては，古くは IBM の YSE や NEC の HAL 等があり [12], [13]，最近では各 CAD ベンダからもシミュレーション専用のプロセッサを多数搭載したものが発表されている．また，FPGA（field programmable gate array）[20] などのプログラマブルデバイスを大量に搭載したエミュレータもある．この場合，実際に回路を論理合成して大量の FPGA で実現するので，エミュレーションは高速だが，準備に相当な時間を必要とする．

本論文の構成は以下のとおりである．**2.** で入力の一部が不明である場合の論理関数の評価方法について概説する．**3.** では正則 3 値論理関数（RT 関数）の 2 線式論理での構成法を提案する．本構成法は，まず与えられた 2 値論理関数を BDD で表現し，次に，各節

† 明治大学理工学部情報科学科，川崎市
Department of Computer Science, Meiji Univ., Kawasaki-shi, 214–8571 Japan
†† 九州工業大学情報工学部電子情報工学科，飯塚市
Dept. of Electronics and Computer Science, Kyushu Institute of Technology, Iizuka-shi, 820–8502 Japan

点を RT モジュールと呼ぶ単純な回路で置換してハードウェア化を行う．また，各 RT モジュールの出力を FF（flip flop）を介して出力させることでパイプライン化し，高速化する方法についても提案する．ソフトウェアでのシミュレーションに比べ最大 100 倍程度高速な FPGA を用いた実現法も示す．ハードウェアで実現した場合の性能評価として FPGA 内で使用する論理ブロックの個数の理論的な見積りを行う．FPGA で回路を実現する場合，配置配線については配線資源と CAD ツールの性能に大きく依存する．そこで配置配線後の回路の良さ（大きさと遅延）については **4.** で実験を行うことにする．**4.** では，ベンチマーク回路に本手法を適用した場合のハードウェア量とスピードに関して実験結果を示す．使用する FPGA 内部の論理ブロックの個数が理論的見積りとほぼ一致していること，また，シミュレーションの速度は *unknown* を正しく扱えるソフトウェアシミュレーションに比べて最大 100 倍程度高速であることが実験によりわかった．

## 2. 入力の一部が不明である場合の論理関数の出力値の評価方法

$B=\{0,1\}$ とする．$n$ 変数の 2 値論理関数 $f$ は，写像：$f: B^n \to B$ を表している．$\boldsymbol{a}=(a_1,a_2,\ldots,a_n)$ を 2 値ベクトルとする，ここで $a_i \in B$．いくつかの $a_i$ が 0 か 1 か未知のまま $\boldsymbol{a}$ に対して $f(\boldsymbol{a})$ を，評価しなければならないことがある [4]．本章では，入力の一部が 0 か 1 か不明の場合に $f$ を評価する方法について概説する．以下では特に断わりのない限り，$n$ は入力変数の個数を示すものとする．

$T=\{0,1,u\}$ とする．ただし，$u$ は 0 か 1 か不明であることを表す真理値とする．$\vec{\alpha}=(\alpha_1,\alpha_2,\ldots,\alpha_n)$ を，$\alpha_i \in T$ であるような 3 値ベクトルとする．もし，すべての $i$ について $\alpha_i$ が 0 か 1 であるなら，$\vec{\alpha} \in B^n$ である．この場合，$f(\vec{\alpha})$ は，0 又は 1 である．ところが，ある $i$ について $\alpha_i = u$ ならば，$\vec{\alpha} \in T^n - B^n$ となる．このとき，ある $\vec{\alpha}$ に対しては $f(\vec{\alpha})$ は，0 か 1 に定まる場合もあるし，定まらない場合もある．そこで，以下の定義 2.2 のように $f$ を拡張定義した 3 値論理関数：$\mathcal{F}: T^n \to T$ を導入するのが便利である [5]．ここで，$f$ に対して $\mathcal{F}$ は一意的に定まる．

［定義 2.1］ $\vec{\alpha}(\in T^n)$ に対して，$\vec{\alpha}$ におけるすべての $u$ を 0 又は 1 で置換して得られる $B^n$ の元からなる集合を $A(\vec{\alpha})$ で表す．ここで $\vec{\alpha}$ に現れる $u$ の個数を $s$ 個とすると，集合 $A(\vec{\alpha})$ の要素の個数は $2^s$ 個である．

［定義 2.2］ 2 値論理関数 $f$ と $\vec{\alpha} \in T^n$ に対して

$$f(A(\vec{\alpha}))=\{f(\boldsymbol{a}) \mid \boldsymbol{a} \in A(\vec{\alpha})\}.$$

$$\mathcal{F}(\vec{\alpha})=\begin{cases} 0 & \text{if } f(A(\vec{\alpha}))=\{0\}. \\ 1 & \text{if } f(A(\vec{\alpha}))=\{1\}. \\ u & \text{if } f(A(\vec{\alpha}))=\{0,1\}. \end{cases}$$

$\mathcal{F}$ を $f$ の正則 3 値論理関数（RT 関数：regular ternary function）と呼ぶ [5]．

［例 2.1］ 論理式 $f(x_1,x_2,x_3)=\bar{x}_1x_2 \vee x_1x_3$ で表される論理関数 $f$ に対し入力 $\vec{\alpha}_1=(0,0,u)$, $\vec{\alpha}_2=(u,1,1)$, $\vec{\alpha}_3=(u,1,u)$ を加える．このとき，

$$\begin{aligned} f(A(\vec{\alpha}_1)) &= \{f(0,0,0), f(0,0,1)\}=\{0\}, \\ f(A(\vec{\alpha}_2)) &= \{f(0,1,1), f(1,1,1)\}=\{1\}, \\ f(A(\vec{\alpha}_3)) &= \{f(0,1,0), f(0,1,1), f(1,1,0), \\ &\qquad f(1,1,1)\}=\{0,1\} \end{aligned}$$

であるから，定義 2.2 より $\mathcal{F}(\vec{\alpha}_1)=0$, $\mathcal{F}(\vec{\alpha}_2)=1$, $\mathcal{F}(\vec{\alpha}_3)=u$. □

ゲートレベルの論理シミュレーションでは図 1 のように 2 値の AND，OR，EXOR，及び，NOT 演算を 3 値の演算に拡張し論理式を評価する方法をとることが多い [4]．これは Kleene の strong ternary logic と呼ばれている [6]．

［例 2.2］ 例 2.1 の論理式を実現する AND-OR 回路を図 2 に示す．上で述べたようにゲートの各演算を 3 値に拡張し，論理式を評価する方法 [4] に従って，入力 $\vec{\alpha}_2=(u,1,1)$ に対する出力を求めると図 2 に示すように出力は，$u$ となってしまう．しかしながら，$\mathcal{F}(\vec{\alpha}_2)=\mathcal{F}(u,1,1)=1$ なので，実際の回路の出力は

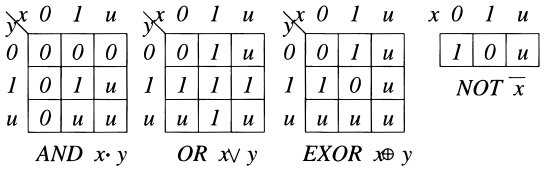



図 1 3 値の AND，OR，EXOR，及び，NOT 演算
Fig. 1 Ternary AND, OR, EXOR, and NOT.

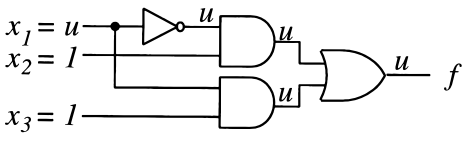



図 2 例 2.1 の論理式を実現する AND-OR 回路
Fig. 2 Realization of expression in Example 2.1.

1 となる．このような方法では必ずしも常に正しい結果を出すとは限らない． □

定義 2.2 に従って入力の一部が不明である場合の出力値を正確に求める方法，つまり，与えられた 3 値ベクトルに対する RT 関数の値を計算する方法がいくつか提案されている． これには，論理関数が積和形論理式で与えられているときの方法 [5]，BDD を用いる方法 [10], [11]，Kleene_TDD を用いる方法 [8], [23] がある．

## 3. RT 関数を実現するハード ウェアの構成法

本章では，まず，決定図（decision diagram）を用いた論理関数の評価法について概説し [1]〜[3]，次に，RT 関数の 2 線式論理での構成法を提案する．

### 3.1 決定図ベースの論理関数評価法

［定義 3.1］[15] 2 分決定グラフ（BDD：binary decision diagram）は，節点の集合 $\boldsymbol{V}$ で表される根付きの有向グラフである．非終端節点 $v$ は，インデックス $index(v) \in \{1, \ldots, n\}$ と二つの子 $low(v)$, $high(v) \in \boldsymbol{V}$ を属性としてもつ． 終端節点 $v$ は，値 $value(v) \in B$ を属性としてもつ．どの非終端節点 $v$ に対しても，もし $low(v)$ が非終端節点であるなら $index(v) < index(low(v))$． 同様に，もし $high(v)$ が非終端節点であるなら $index(v) < index(high(v))$ である．2 値論理関数 $f$ と BDD は次のように関連づけられる．

終端接点 $v$ に対し：

もし，$value(v) = 1$ ならば，$f_v = 1$.

もし，$value(v) = 0$ ならば，$f_v = 0$.

非終端節点 $v$ に対し：

もし，$index(v) = i$ ならば，$f_v$ は次のような関数である．すなわち，

$f_v(x_1, \ldots, x_n) = \bar{x}_i \cdot f_{low(v)}(\vec{X}) \vee x_i \cdot f_{high(v)}(\vec{X})$.

ここで，$\vec{X} = (x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$.

$f_v$ は，非終端節点 $v$ に対応する 2 値論理関数を，根は関数 $f$ それ自身を表していることに注意．

［定義 3.2］[15] BDDが次の条件を満たすときそれを Quasi Reduced Ordered Binary Decision Diagram（QROBDD）という．

（1） $v_1$ を根とした部分グラフと $v_2$ を根とした部分グラフとが等しいような節点 $v_1$，$v_2$ をもたない．

（2） 根から終端節点までのどのパスも，ちょうど $n$ 個の非終端節点を通る．

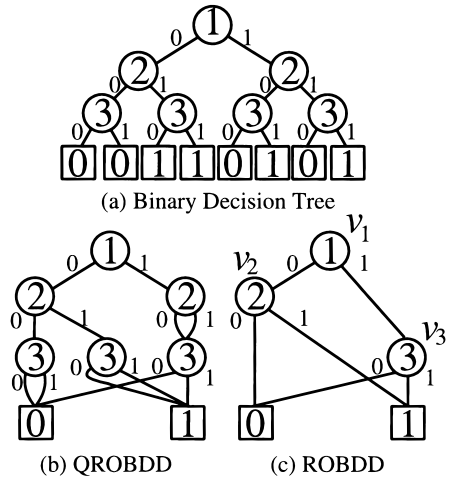



図 3 BDD の例
Fig. 3 Example of BDDs.

［定義 3.3］ 論理関数 $f$ の BDD の節点数を $N(f)$ で表す．

［定義 3.4］[15] BDD が次の条件を満たすときそれを Reduced Ordered Binary Decision Diagram（ROBDD）という．

（1） $low(v) = high(v)$ であるような $v$ を含まない．

（2） $v_1$ を根とした部分グラフと $v_2$ を根とした部分グラフとが等しいような節点 $v_1$，$v_2$ をもたない．

［例 3.1］ 例 2.2 の関数 $f$ の 2 分決定木（binary decision tree）を図 3 (a) に示す．なお，各節点内の数字は変数の添字を，枝についている数字の 0 は $low(v)$ 側を，1 は $high(v)$ 側を意味している．等価な部分グラフを併合し (b) QROBDD を得る．これから更に冗長な節点を削除すると (c) ROBDD を得る． □

BDD ベースの論理シミュレータの動作を例を用いて説明する．

［例 3.2］ 図 3 (c) の ROBDD を用いて例 2.2 の回路に対して $f(x_1, x_2, x_3) = f(0, 1, 1)$ を求める．まず根 $v_1$ のラベルは 1（$x_1$ を表す）である．$x_1$ の値は，0 であるので 0 側に進み節点 $v_2$ に至る．節点 $v_2$ のラベルは 2（$x_2$ を表す）．$x_2$ の値は，1 であるので 1 側に進む．終端節点 1 に到達するので $f(0, 1, 1) = 1$ と決定できる． □

このように，各変数の値に従って根からグラフをたどっていくことで値を求められる．実際には，シミュレータは図 3 (c) から図 4 に示すようなプログラムを生成し，これをコンパイルして実行することになる．

```
int f(int x1,x2,x3){
     if(x1) goto v3;
     else goto v2;
v2:  if(x2) return(1);
     else return(0);
v3:  if(x3) return(1);
     else return(0);
}
```

図 4　BDD よりシミュレータが生成したプログラムの例
Fig. 4　Example of the code generated from the BDD.

| $x$ / $y$ | $0$ | $1$ | $u$ |
|---|---|---|---|
| $0$ | $0$ | $u$ | $u$ |
| $1$ | $u$ | $1$ | $u$ |
| $u$ | $u$ | $u$ | $u$ |

図 5　Alignment 演算 $x \odot y$
Fig. 5　Alignment $x \odot y$.

また，高速化のために直接ネイティブコードを生成するシミュレータもある．$n$ 入力論理関数の 1 回の評価時間は，根からたどる段数に相当するので $O(n)$ となる．実際のサイクルベースシミュレータでは，MDD（Multi-valued Decision Diagram）を用いて段数を数分の 1 に減らすことで更に高速化を図っている．この方法は大規模な回路に対してイベントドリブンシミュレータの約 30 倍から 1000 倍高速であり，Verilog RTL で 10 万ラインの記述量のマイクロプロセッサに対する速度は約 2 けた程度高速であることが知られている [2].

しかし，このままでは 0 と 1 の 2 値しか取り扱えない．$u$ を取り扱うためには，定義 2.2 にあるように $u$ に 0 と 1 とを代入して関数の値を求める必要がある． これを定義 3.5 に示す Alignment 演算を導入して定式化する．

［定義 3.5］ 図 5 の *Alignment* 演算は，$x, y \in T$ とすると，

$$x \odot y = \begin{cases} x & x = y \text{ のとき}, \\ u & \text{上記以外のとき}. \end{cases}$$

を表している．

［定理 3.1］

$\mathcal{F}(u, x_2, \ldots, x_n) = \mathcal{F}(0, x_2, \ldots, x_n) \odot \mathcal{F}(1, x_2, \ldots, x_n).$

［例 3.3］ 図 3 (c) の ROBDD を用いて例 2.2 の回路に対して $\mathcal{F}(x_1, x_2, x_3) = \mathcal{F}(u, 1, 1)$ を求める．$\mathcal{F}(u, 1, 1)$ は，定理 3.1 により $\mathcal{F}(0, 1, 1) \odot \mathcal{F}(1, 1, 1)$ として求められる．例 3.2 で求めたように $\mathcal{F}(0, 1, 1) = 1$． 同様に $\mathcal{F}(1, 1, 1) = 1$． したがって，$\mathcal{F}(u, 1, 1) = \mathcal{F}(0, 1, 1) \odot \mathcal{F}(1, 1, 1) = 1 \odot 1 = 1$ となる． □

これは，入力変数の値が $u$ であるような節点で *low* 側と *high* 側の両方をたどることを意味する． 出力が $u$ となる場合は評価途中で値を決定することができる場合もある．それ以外の場合，すべての場合について評価を行わねばならない． この場合，$s$ を入力が $u$ である入力変数の個数とすると，評価に要する時間は，$O(2^s \cdot n)$ となる． 一方，のちに述べるアルゴリズム 3.1 を用いて， RT 関数をまず図 8 のような論理回路で構成し，それをソフトウェアでシミュレーションすれば $O(N(f))$ の時間で評価できる．しかし，FPGA のようなハードウェアで実現するよりは時間がかかる．Kleene_TDD を用いて $O(n)$ の時間で求める方法が提案されているが，この場合節点数が BDD に比べ $O(3^n/n)$ に増大するという問題があり，実用化には分解などの技法の整備など課題がある [9].

以下では入力値に $u$ がある場合でも，ハードウェア量は BDD と同じく $O(2^n/n)$，評価時間は $O(n)$ で求めることのできるハードウェアの構成法を示す． また，パイプライン化による高速化についても述べる．

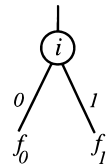
### 3.2　RT 関数を実現するハードウェア

RT 関数をハードウェアで実現する方法を本節で提案する． RT 関数は書換え可能なデバイスで実現することになる． 書換え可能なデバイスには，CPLD や FPGA などがある．ただし，CPLD では $2^n$ 個の積項を必要とするので（定理 3.3 で後述），ここでは LUT 型の FPGA で実現する．

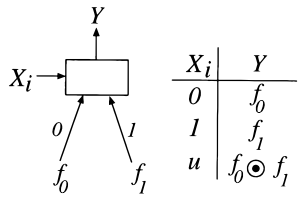
まずは，実現手法について考える．BDD をたどるとき，ある節点ラベルに対応する変数の値が 0(1) ならば *low*(*high*) 側に進み，$u$ のとき *low* 側と *high* 側の両方を評価し，それらの値の Alignment 演算を行えば正しく計算できる． つまり図 6 (b) の表に示す演算を実現するモジュールを BDD の節点 1 個（図 6 (a)）に対応させればよい． 0，1，及び，$u$ の 3 値を表 1 に示すように，それぞれ $(1, 0)$，$(0, 1)$，及び，$(1, 1)$ に割り当てて図 6 (b) のモジュールを実現すると，2 線式論理で実現された回路（図 7）が得られる． これを RT モジュールと呼ぶことにする．

［アルゴリズム 3.1］ RT 関数を実現するハードウェアの構成法

（1） 2 分決定グラフの 1 個の節点（図 6 (a)）を RT モジュール（図 7）1 個で置換する． 各 RT モジュー




(a) A node of BDD.




(b) RT module.

図 6　RT モジュール
Fig. 6　RT module.

表 1　RT モジュールの真理値表
Table 1　Truth table for an RT module.

| $X_{iL}$ | $X_{iH}$ | $f_{0L}$ | $f_{0H}$ | $f_{1L}$ | $f_{1H}$ | $Y_L$ | $Y_H$ |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | – | – | 1 | 0 |
| 1 | 0 | 0 | 1 | – | – | 0 | 1 |
| 1 | 0 | 1 | 1 | – | – | 1 | 1 |
| 0 | 1 | – | – | 1 | 0 | 1 | 0 |
| 0 | 1 | – | – | 0 | 1 | 0 | 1 |
| 0 | 1 | – | – | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | otherwise | | | | 1 | 1 |
| otherwise | | | | | | – | – |

– : don't care

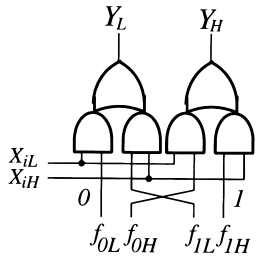
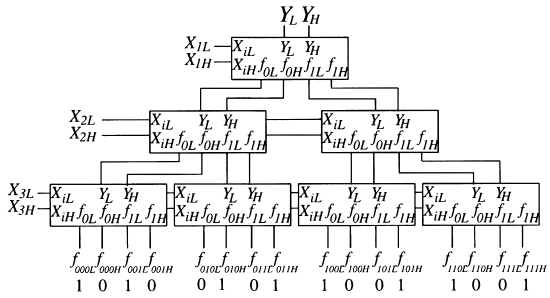



図 7　RT モジュールの 2 線式による実現
Fig. 7　Double-rail realization of an RT module.

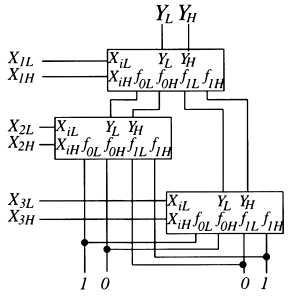
ル間は BDD の枝に対応させて配線する．例えば，図 3 (a) を図 8 (a) に，図 3 (c) を図 8 (b) の回路に変換する．

（2） 終端節点 0 を $(f_L, f_H) = (1, 0)$ に，終端節点 1 を $(f_L, f_H) = (0, 1)$ に対応させる．

（3） 各変数 $X_i$ に計算したい値を入力する．ただし，0，1，$u$ は各々 $(X_{iL}, X_{iH}) = (1, 0), (0, 1), (1, 1)$




(a)




(b)

図 8　RT 関数の実現
Fig. 8　Realization of RT function.

で表す．

（4） 出力 $(Y_L, Y_H)$ の値が求める RT 関数の値となる．ただし，$(Y_L, Y_H)$ の値は $(1, 0), (0, 1), (1, 1)$ が各々 0，1，$u$ を表す．

［定理 3.2］ $n$ 入力 2 値論理関数 $f$ の RT 関数は，上記の構成法に従って，$N(f)$ 個の RT モジュールを用いて実現できる．

（証明） 構成法より明らか． □

次に RT 関数を CPLD で実現する場合のコストについて考える．まず，図 8 (a) の $Y_L, Y_H$ の論理式を以下に示す．

$$
\begin{aligned}
Y_L \;=\; & f_{000L}X_{1L}X_{2L}X_{3L} \vee f_{001L}X_{1L}X_{2L}X_{3H} \\
& \vee\; f_{010L}X_{1L}X_{2H}X_{3L} \vee f_{011L}X_{1L}X_{2H}X_{3H} \\
& \vee\; f_{100L}X_{1H}X_{2L}X_{3L} \vee f_{101L}X_{1H}X_{2L}X_{3H} \\
& \vee\; f_{110L}X_{1H}X_{2H}X_{3L} \vee f_{111L}X_{1H}X_{2H}X_{3H}. \\
Y_H \;=\; & f_{000H}X_{1L}X_{2L}X_{3L} \vee f_{001H}X_{1L}X_{2L}X_{3H} \\
& \vee\; f_{010H}X_{1L}X_{2H}X_{3L} \vee f_{011H}X_{1L}X_{2H}X_{3H} \\
& \vee\; f_{100H}X_{1H}X_{2L}X_{3L} \vee f_{101H}X_{1H}X_{2L}X_{3H} \\
& \vee\; f_{110H}X_{1H}X_{2H}X_{3L} \vee f_{111H}X_{1H}X_{2H}X_{3H}.
\end{aligned}
$$

［定義 3.6］ 関数 $f$ の MSOP（最小論理和形）の積項数を $\tau(MSOP, f)$ で表す．

［定理 3.3］ $\tau(MSOP, Y_L) + \tau(MSOP, Y_H) = 2^n$．

（証明） 3 変数の場合について考える．論理式 $Y_L$，$Y_H$ において係数の組 $(f_{000L}, f_{000H})$ のとり得る値は $(1,0), (0,1)$ のいずれかである．どちらの場合でも積項 $X_{1L}X_{2L}X_{3L}$ は，$Y_L, Y_H$ のうちどちらか一方の論理式に一つだけ必要である．残りの係数についても同様である．どの積項についても互いに距離は 2 以上なのでこれ以上の簡単化はできない．4 変数以上の場合も同様に証明できる． □

定理 3.3 から RT 関数を CPLD のような論理和形を必要とするデバイスで実現すると，$2^n$ 個の積項が必要であることがわかる．一方，定理 3.2 から，RT 関数を，BDD の非終端節点を RT モジュールで置換し，それを LUT（Look-Up Table）型の FPGA にマッピングすると，セル数が $O(2^n/n)$ で抑えられることがわかる．

XILINX 社の FPGA（XC4000 シリーズ）は，三つの再定義可能な要素：CLB（configurable logic block），IOB（input/output block），及び，配線領域から構成されている [19], [20]．XC4000 シリーズの CLB は，2 個の 4 入力 LUT と 1 個の 3 入力 LUT，及び，2 個の D-FF を含んでいる． RT モジュールの出力 $Y_L$ と $Y_H$ は，4 入力論理関数で表せるので それぞれを CLB 内の 4 入力 LUT で実現できる．したがって，CLB1 個で RT モジュール 1 個を実現できる．この場合，回路は BDD から直接変換できるので，論理合成は不要で配置配線のみが必要となる．

詳細については立ち入らないが，RT 関数の次の性質は， 例えばハードウェアの故障検査などに有用である．

［定理 3.4］ 図 8 (a)，(b) で各変数 $X_{iL}, X_{iH}$ にそれぞれ $\bar{x_i}, x_i$ を代入すると $Y_L, Y_H$ はそれぞれ $\bar{f}, f$ を表す．

（証明） 3 変数の場合について考える．図 8 (a)，(b) で各変数 $X_{iL}, X_{iH}$ にそれぞれ $\bar{x_i}, x_i$ を代入すると $Y_L, Y_H$ は次のようになる．

$$
\begin{aligned}
Y_L \;=\; & f_{000L}\bar{x_1}\bar{x_2}\bar{x_3} \vee f_{001L}\bar{x_1}\bar{x_2}x_3 \\
\vee\; & f_{010L}\bar{x_1}x_2\bar{x_3} \vee f_{011L}\bar{x_1}x_2x_3 \\
\vee\; & f_{100L}x_1\bar{x_2}\bar{x_3} \vee f_{101L}x_1\bar{x_2}x_3 \\
\vee\; & f_{110L}x_1x_2\bar{x_3} \vee f_{111L}x_1x_2x_3.
\end{aligned}
$$

$$
\begin{aligned}
Y_H \;=\; & f_{000H}\bar{x_1}\bar{x_2}\bar{x_3} \vee f_{001H}\bar{x_1}\bar{x_2}x_3 \\
\vee\; & f_{010H}\bar{x_1}x_2\bar{x_3} \vee f_{011H}\bar{x_1}x_2x_3 \\
\vee\; & f_{100H}x_1\bar{x_2}\bar{x_3} \vee f_{101H}x_1\bar{x_2}x_3 \\
\vee\; & f_{110H}x_1x_2\bar{x_3} \vee f_{111H}x_1x_2x_3.
\end{aligned}
$$

$Y_H$ は $f$ を，$Y_L$ は $\bar{f}$ を主加法標準形で表現したものにほかならない．したがって，定理が成立する． □

### 3.3 回路の分解

本構成法に従って回路を構成し，FPGA で実現する場合を考える． 1 個の FPGA で収容できない大規模な関数は，複数の FPGA に分けて収納する必要がある． この場合の制約条件は，FPGA の CLB の個数と端子数となる． また，実際には多出力論理関数を実現しなければならない． このような場合は，与えられた多出力論理関数に対応する共有 2 分決定グラフ（SBDD: shared binary decision diagram）を構成すればよい．

使用する FPGA に含まれる CLB の個数を $\#CLBs$ で，入出力数の合計を $\#I/O$ で表すとする． もし $(N(f) \leqq \#CLBs) \wedge ((\#inputs + \#outputs) * 2 \leqq \#I/Os)$，であるならば，RT 関数は 1 個の FPGA で実現できる． その他の場合，回路を各出力ごとに分解すればよい． この問題はビンパッキング問題として定式化されている [20]．

もし $(N(f) > \#CLBs) \vee ((\#inputs + \#outputs) * 2 > \#I/Os)$ であるならば関数分解 [9], [18] を行う必要がある．

### 3.4 パイプライン化による高速化

通常，論理シミュレーションを行う場合，一つの入力ベクトルに対してのみ出力値を評価するのではなく，大量の入力ベクトルに対して効率良く計算できることが望まれる．本方式では，図 9 に示すように，RT モジュールの出力に D-FF を入れたモジュールを使用す

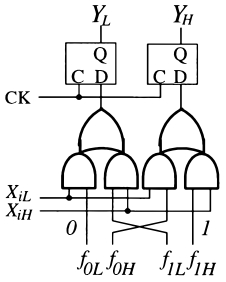



図 9 パイプライン化した RT モジュール
Fig. 9 An pipelined RT module.

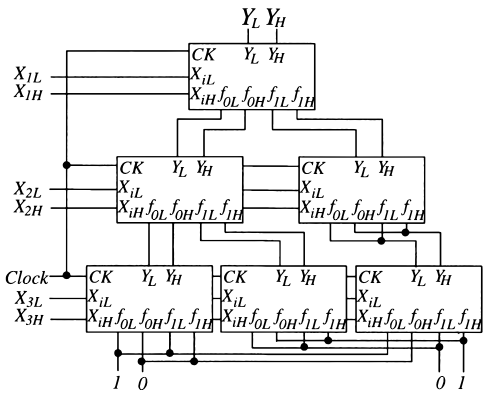



図 10　パイプライン化した RT モジュールを用いた RT 関数の実現
Fig. 10　Realization of RT function with pipelined RT modules.

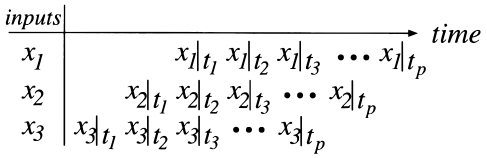



図 11　入力のスケジューリング例
Fig. 11　Example of inputs' schedule.

ることで，簡単にパイプライン化できる．この場合，根から終端節点までのすべての経路の段数をそろえる必要があるので，ROBDD ではなく，QROBDD を使用する必要がある．図 3 (b) の各節点をパイプライン化した RT モジュール（図 9）で置換した例を図 10 に示す．このモジュールも CLB 1 個で実現できる．

パイプライン化する場合は，入力ベクトルをずらして入れる必要がある．時刻 $t_p$ のときの入力 $x_i$ の値を $x_i|_{t_p}$ で表現する．例えば，図 10 に時刻 $t_1$ から $p$ 個の入力ベクトル $\alpha_1 = (x_1|t_1, x_2|t_1, x_3|t_1), \alpha_2 = (x_1|t_2, x_2|t_2, x_3|t_2), \ldots, \alpha_p = (x_1|t_p, x_2|t_p, x_3|t_p)$ に対し計算をする場合の入力のスケジューリングを図 11 に示す．

## 4. 実験結果

**3.** では RT 関数を FPGA で実現する方法について述べ，ハードウェア量のうち CLB の個数について理論的に解析を行った．しかし，配置配線後のハードウェア量，及び，信号遅延は配線資源，CAD ツールによって選ばれた配線経路に大きく依存する．そこで，本章ではこれらの値についてはベンチマーク関数に対して計算機実験を行うことで評価する．実験結果を表 2 に示す．ここで，多出力関数は SROBDD（shared reduced ordered BDD），及び，SQROBDD（shared quasi reduced ordered BDD）で表現している．BDD の入力変数の順序付けは発見的手法 [21], [22] を使用した．各関数に対して，node の欄上段には SROBDD，下段には SQROBDD の非終端節点数を示す．これらの BDD より提案手法を用いたハードウェアを実現するデバイスには，XILINX 社の FPGA を想定した．開発システムには XILINX 社の Foundation Series 1.4 を用いた．XILINX 社の FPGA は，CLB と呼

表 2　実験結果
Table 2　Experimental results.

| function | in | out | nodes | CLBs | nsec/vector |
|---|---|---|---|---|---|
| misj[1] | 35 | 14 | 41 | 48 | 28 |
| | | | 286 | 89 | 13 |
| rckl[1] | 32 | 7 | 190 | 178 | 95 |
| | | | 374 | 229 | 18 |
| x6dn[1] | 39 | 5 | 219 | 328 | 84 |
| | | | 986 | 272 | 31 |
| apex1[3] | 45 | 45 | 1275 | 1057 | 147 |
| | | | 4274 | 1362 | 89 |
| apex2[1] | 39 | 3 | 59 | 51 | 62 |
| | | | 127 | 91 | 12 |
| apex5[5] | 117 | 88 | 1077 | 1058 | 202 |
| | | | 10310 | 1139 | 66 |
| apex7[3] | 49 | 37 | 262 | 252 | 73 |
| | | | 1747 | 348 | 30 |
| exep[3] | 30 | 63 | 601 | 383 | 69 |
| | | | 1429 | 691 | 38 |
| accpla[4] | 50 | 69 | 1587 | 1290 | 169 |
| | | | 5822 | 1701 | 67 |
| mish[4] | 94 | 43 | 103 | 130 | 22 |
| | | | 1956 | 237 | 13 |
| signet[5] | 39 | 8 | 1440 | 1295 | 216 |
| | | | 3894 | 1487 | 92 |
| xparc[5] | 41 | 73 | 1875 | 1440 | 358 |
| | | | 4749 | 1983 | 99 |
| c432[5] | 36 | 7 | 1848 | 1777 | 340 |
| | | | 3040 | 3082 | 105 |
| c432 (1)[1] | 18 | 1 | 18 | 19 | 35 |
| | | | 87 | 123 | 12 |
| c432 (2)[1] | 27 | 1 | 73 | 73 | 76 |
| | | | 191 | 227 | 14 |
| c432 (3)[3] | 36 | 1 | 265 | 263 | 99 |
| | | | 470 | 506 | 33 |
| c432 (4)[3] | 36 | 1 | 273 | 245 | 86 |
| | | | 522 | 558 | 37 |
| c432 (5)[3] | 36 | 1 | 384 | 363 | 108 |
| | | | 651 | 687 | 37 |
| c432 (6)[3] | 36 | 1 | 460 | 445 | 114 |
| | | | 782 | 818 | 44 |
| c432 (7)[3] | 36 | 1 | 521 | 510 | 115 |
| | | | 877 | 913 | 54 |

[1]XC4010E-1, [2]XC4036XL-1, [3]XC4052XL-1, [4]XC4062XL-09, [5]XC40125XV-1

ばれる論理関数を実現するブロックがアレー状に配置されている．例えば，実験に用いた中で規模が最大の XC40125XV-1 は，4624（68×68）個の CLB をもつ [19]．各ベンチマーク関数を実現するのに必要な CLB 数は 4624 より少なく，また，CAD による配置配線の結果，すべての関数が XC40125XV-1 1 個にマッピングできた． CLB 数，及び，シミュレーション時間は，上段に非パイプラインのものを下段にパイプライン化した場合を示す．シミュレーション時間は，スタティックタイミングアナライザ，及び，タイミングシミュレーションを用いて本方式の 1 ベクトルをシミュレーションするに要する最大時間を求めた．ただし，パイプライン化したものはスループットとなっていることに注意．

例えば，C432 は，36 入力，7 出力の関数である．この関数を SROBDD で表現すると非終端節点の個数は，1848 個であり，これを本方式でハードウェア化し，XC4010E-1 で実現すると 1777 個の CLB を必要とする．1 ベクトル当りのシミュレーション時間の最大値は，340 ns である． 一方，QROBDD で表現すると非終端節点の個数は，3040 個であり， これをパイプライン化し，同じく XC4010E-1 で実現すると 3082 個の CLB を必要とする． 1 ベクトル当りのシミュレーション結果は，105 ns ごとに出力される．

CLB の個数は，BDD の接点数より大きくなることも小さくなることもある．ファンアウトが大きい場合は，バッファが自動的に挿入され，そのために，CLB 数が増加する．一方，終端節点の定数 0，及び，1 に直接つながる RT モジュールは，簡単化される．更に，この影響により，次段以後の回路も簡単化される場合もある．この場合は，CLB 数が減少する．

決定図技法を用いた高速なソフトウェアシミュレーションに対して， C432 の場合で，非パイプラインのもので約 10 倍，パイプライン化したもので約 30 倍の高速化が図れた [2]．ただし，このソフトウェアシミュレータは，*unknown* を取り扱えない．一方，本方式は入力に *unknown* があっても，速度の低下なしに正しくシミュレーションできる．*unknown* を正しく扱えるソフトウェアシミュレータに比べ，約 100 倍の高速化が図れた [23].

パイプライン化した方式が，パイプライン化しないものに比べ約 2〜5 倍程度の動作速度の改善が見られる．高速化の度合が少ないのは，QROBDD では節点数が数倍〜10 倍程度増加しているため FPGA の配線資源のうち比較的遅いスイッチマトリクス経由の配線が増加することが原因と思われる．信号遅延の大きいものを解析した結果，配線資源が足りなくなって比較的遅い配線を使用しなければならなくなって遅延が増加していること，特にパイプライン化したものの信号遅延の 98% が配線によるものであり，CLB での遅延は無視できる程度に小さいことがわかった． また，規模の小さな回路を大きな FPGA で実現すると配線が遠くを迂回して遅延が増大する場合があることがわかった．C432 を 1 出力ごとに分解し C432 (1)〜(7) とし，もとのものより規模の小さな FPGA で実現した場合の実験を行った．分解前に比べ 2〜3 倍程度高速になった．

開発システムや FPGA の改良が盛んに行われているので，今後，性能は更に改善されると思われる．

## 5. むすび

本論文では，入力の一部に 0 か 1 か不明な値がある場合の論理関数の評価法を概説した．論理シミュレーションで不明値 $u$ を正確に求めるには，RT 関数を評価すればよい．我々は，RT 関数を 2 線式論理を用いて実現する方法を提案した． この方法は，$O(2^n/n)$ のロジックセル，$O(n)$ の時間を必要とする．実験を行った結果，FPGA での実現はソフトウェアシミュレーションより最大で約 100 倍の高速化を実現できることを示した．

## 文　献


[1] P. Ashar and S. Malik, “Fast functional simulation using branching programs,” ICCAD ’95, pp.408–412, Nov. 1995.

[2] P.C. McGeer, K.L. McMillan, A. Saldanha, A.L. Sangiovanni-Vincentelli, and P. Scaglia, “Fast discrete function evaluation using decision diagrams,” ICCAD ’95, pp.402–407, Nov. 1995.

[3] R. Murgai and M. Fujita, “Some recent advances in software and hardware logic simulation,” 10th Int’l Conf. on VLSI Design, pp.232–238, Jan. 1997.

[4] M. Abramovici, M.A. Breuer, and A.D. Friedman, “Digital Systems TESTING and Testable DESIGN,” pp.43–46, Computer Science Press, 1990.

[5] M. Mukaidono, “Regular ternary logic functions — ternary logic functions suitable for treating ambiguity,” IEEE Trans. Comput., vol.C–35, no.2, pp.179–183, Feb. 1986.

[6] S.C. Kleene, Introduction to Metamathematics, Wolters-Noordhoff, North-Holland Publishing, 1952.

[7] Y. Iguchi, T. Sasao, and M. Matsuura, “On properties of Kleene TDDs,” IEICE Trans. INF. & SYST.,

vol.E81-D, no.7, pp.716–723, July 1998.
[8] G. Jennings, "Symbolic incompletely specified functions for correct evaluation in the presence of indeterminate input values," 28th Hawaii Int'l Conf. on System Science (vol.I: Architecture), pp.23–31, Jan. 1995.
[9] Y. Iguchi, T. Sasao, and M. Matsuura, "On decomposition of Kleene TDDs," Sixth Asian Test Symposium, Proc. ATS '97, pp.234–239, Nov. 1997.
[10] R.E. Bryant, "Boolean analysis of MOS circuits," IEEE Trans. on CAD of Integrated Circuits, vol.CAD-6, no.4, pp.634–649, July 1987.
[11] R.E. Bryant, "Symbolic simulation—techniques and applications," Proc. of 27th Design Automation Conf., pp.517–521, June 1990.
[12] 小池誠彦, "CAD マシン," オーム社, 1989.
[13] T. Nakata and N. Koike, "Design automation machine," Design Methodologies, ed., S. Goto, North-Holland, pp.465–499, 1986.
[14] R.E. Bryant, "Graph-based algorithms for boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, Aug. 1986.
[15] T. Sasao and M. Fujita (ed.), Representations of Discrete Functions, Kluwer Academic Publishers, 1996.
[16] S. Minato, "Graph-based representation of discrete functions," in [15].
[17] T. Sasao, "Ternary decision diagrams and thier applications," in [15].
[18] 笹尾 勤, "論理設計：スイッチング回路理論 [第 2 版]," 近代科学社, 1998.
[19] "XC4000E and XC4000X Series Field Programmable Gate Arrays," http://www.xilinx.com/partinfo/databook.htm#xc4000, Nov. 1997.
[20] S.D. Brown, R.J. Francis, J. Rose, and Z.G. Vranesic, "Field Programmable Gate Arrays," Kluwer Academic Publishers, Boston, 1992.
[21] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchanges of variables," ICCAD-91, pp.472–475, 1991.
[22] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," ICCAD-93, pp.42–47, 1993.
[23] 井口幸洋, 笹尾 勤, 伊勢野総, "2 線式論理を用いた正則 3 値論理関数の実現法," 信学 '98 総全大.

（平成 10 年 10 月 30 日受付，11 年 2 月 8 日再受付）

**井口 幸洋** （正員）

昭 57 明大・工・電子通信卒．昭 62 同大大学院博士課程了．同年同大・工・助手．平 5 同大・理工・専任講師，現在に至る．工博．この間，平 8 年 4 月より 1 年間九州工業大学情報工学部にて私学研修員．論理設計，スイッチング理論，再構成可能なシステムに興味をもつ．情報処理学会会員．

**笹尾 勤** （正員）

1972 阪大・工・電子卒．1977 同大大学院博士課程了．工博． 同年同大・工・助手．1988 九州工大・情報・助教授，1993 同教授，現在に至る．この間，1982 年 2 月より 1 年間 IBM ワトソン研究所客員研究員，1990 年 1 月より 3 か月間米国海軍大学院教授．1979 年丹羽記念賞，1987 年 ISMVL 論文賞．論理設計，スイッチング理論，多値論理などの研究に従事．最近は，EXOR 論理回路，BDD，論理回路の分解に興味をもつ．著書『論理設計とスイッチング理論—LSI，VLSI の設計基礎—』（共立出版，共訳），『スイッチング理論演習』（朝倉書店，共著），『PLA の作り方使い方』（日刊工業），『論理回路：スイッチング回路理論』（近代科学社），「Logic Synthesis and Optimization」，「Representations of Discrete Functions」（Kluwer，編著），「Switching Theory for Logic Synthesis」（Kluwer，1999），など．IEEE Transactions on Computer Editor， IEEE フェロー．

**松浦 宗寛**

平 3 より九州工大技官．論理設計アルゴリズム，決定図技法，EXOR 論理回路，論理回路の分解に興味をもつ．

**伊勢野 総**

平 8 明大・理工・物理卒．平 10 同大大学院博士前期課程了．現在同大学院博士後期課程情報科学系在学中．論理設計，再構成可能システムに興味をもつ．