社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

並列ブランチング・プログラム・マシンを用いた順序回路の模擬について

中原　啓貴† 　笹尾　勤† 　松浦　宗寛† 　川村　嘉郁††

† 九州工業大学 情報工学部 〒820–8502 福岡県飯塚市大字川津 680–4
†† ルネサステクノロジ 〒100–0004 東京都千代田区大手町 2-6-2

あらまし　順序回路を模擬するブランチング・プログラム・マシン (BM) を基本演算要素とし, BM を 128 台並列に並べたマシン (PBM128) を試作した. PBM128 は 128 台の BM とそれらを接続するプログラマブル接続回路からなる. 論理関数を 4 値の決定グラフで表現し, 3 アドレス方式 4 分岐命令で評価する. PBM128 上に種々の多出力ベンチマーク関数を実現し, Intel 社の Core2Duo と比較を行った. PBM128 が必要とするメモリ量は Core2Duo の約 4 分の 1 であり, 速度は 21.4〜96.1 倍であった.

キーワード　組込みシステム, ブランチング・プログラム・マシン, マルチプロセッサ, BDD

Emulation of Sequential Circuits by a Parallel Branching Program Machine

Hiroki NAKAHARA†, Tsutomu SASAO†, Munehiro MATSUURA†, and Yoshifumi KAWAMURA††

† Department of Computer Science and Electronics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan
†† Renesas Technology Corp., Tokyo, 100-0004, Japan

Abstract　The parallel branching program machine (PBM128) consists of 128 branching program machines (BMs) and a programmable interconnection. To represent logic functions on BMs, we use quaternary decision diagrams. To evaluate functions, we use 3-address quaternary branch instructions. We realized many benchmark functions on PBM128, and compared its memory size and computation time with the Intel's Core2Duo microprocessor. PBM128 requires approximately quarter of the memory for the Core2Duo, and is 21.4-96.1 times faster than the Core2Duo.

Key words　Embedded System, Branching Program Machine, Multi-Processing, BDD

1. はじめに

二分決定グラフ (Binary Decision Diagram: BDD) を模擬するブランチング・プログラム・マシン (以下 BM と略記) [2], [3], [21] は 2 種類の命令 (分岐命令と出力命令) を有するプロセッサである. 従って, BM は汎用プロセッサよりもアーキテクチャが単純である. また, 制御回路などに多く用いる条件分岐を専用命令で実行するので, 特定のアプリケーションでは汎用プロセッサよりも高速に処理できる. BM のアプリケーションは, 制御回路 [3], [21], 回路シミュレータ [1], [8], [16], ネットワーク機器の経路表やパケット分類 [19], 及びパターンマッチングなどが挙げられる.

本論文では BM を 128 台用いた並列ブランチング・プログラム・マシン (以下 PBM128 と略記) について述べる. 性能を向上させ命令ステップ数を削減するため, 連続する 2 ビットを同時に評価する命令を新たに追加する. また, 複数の BM を並列に動作させる手法について述べる.

第 2 章では順序回路を模擬するブランチング・プログラム・マシンについて述べる. また, 連続する 2 ビットを同時に評価する命令について述べる. 第 3 章では, 並列ブランチング・プログラム・マシンについて述べる. 第 4 章では, PBM128 を FPGA 上に実装し, 汎用プロセッサと比較した結果を述べる. 第 5 章で, 本論文のまとめと今後の課題について述べる.

2. 順序回路を模擬するブランチング・プログラム・マシン

本章では順序回路を模擬する BM について述べる. 本論文では図 1 に示す順序回路を模擬することを考える. 組合せ論理回路部を決定グラフで表現し, BM の命令に変換する. 次に, その命令を BM で実行する. 順序回路を模擬するために, 状態変数を保持するレジスタを用いる.

本章では, まず多出力論理関数を表現する MTBDD (Multi-Terminal Binary Decision Diagram) とそれを模擬する命令について述べる. 次に, ステップ数と実行時間を削減するため, MTQDD (Multi-Terminal Quaternary Decision Diagram) とそれを模擬する命令について述べる. そしてこれらの命令を実行する BM について述べる.

本論文では BM の命令長を 32 ビットとした. これは組込みシステムのデータ長と試作に用いた FPGA の組込みメモリの

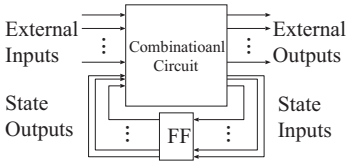


図 1　順序回路のモデル.

ビット幅に適合しているためである.

2.1 MTBDD と MTQDD

任意の n 入力論理関数は **BDD (Binary Decision Diagram)** [4] で表現できる. 多出力論理関数を表現する BDD は **MTBDD** [10] と呼ばれ, 高々n 回のテーブル参照で多出力を同時に評価できる. テーブル参照の回数の平均値を**平均パス長**と呼ぶ. BDD の評価時間は平均パス長に比例する [5].

[例 2.1]　図 2 に MTBDD の例を示す.　(例終)

BDD の評価時間を削減するために, **MDD (Multi-valued Decision Diagram)** [11] を用いる. k 個の 2 値変数を組にして 1 つの変数と考えるとそれは 2^k 値の**超変数**となる. 2^k 値の超変数で表現した決定グラフを $MDD(k)$ で表す. 従って, BDD は MDD(1) と等価である. 論理関数を MDD(k) で表現すると, BDD で表現した場合に比べメモリ参照回数をしばしば $\frac{1}{k}$ まで削減できる [9]. 従って, k を増加させると評価時間を短縮できる. しかし, 1 つの節点を表現するメモリ量は $O(2^k)$ で増加する. 多くのベンチマーク関数において, $k = 2$ の時 MDD(k) の総メモリ量が最小になる [12]. 従って, 論理関数の評価では MDD(2) のほうが BDD よりも適している. MDD(2) は 4 分岐を持つため, 本論文では MDD(2) を **QDD (Quaternary Decision Diagram)** と表現する. 各グループにおける 2 値変数の個数が全て k である MDD を**ホモジニアス MDD(k)** と呼ぶ. 一方, 各グループにおける 2 値変数の個数が k 以下である MDD を**ヘテロジニアス MDD(k)** と呼ぶ. ヘテロジニアス MDD は変数のグルーピングを工夫するとホモジニアス MDD よりも参照回数と節点数を削減できる [13]. 以降, 本論文ではヘテロジニアス MTQDD を単に QDD と呼ぶ.

[例 2.2]　図 3 に図 2 で示した MTBDD を MTQDD に変換した結果を示す. 最上位の節点のみ 2 分岐であり, 他の節点は 4 分岐となっている.　(例終)

2.2 BDD を評価する命令セット

MTBDD を評価するために, 終端節点と非終端節点を模擬する 2 つの命令を用いる. **2 アドレス方式 2 分岐命令 (B_BRANCH)** は非終端節点を評価し, **データセット命令 (DATASET)** は終端節点を評価する. 以下に, これらのニーモニックと内部表現を示す.

```
B_BRANCH (ADDR0, ADDR1), INDEX
```

31 29	28 22	21 16	15 8	7 0
111		INDEX	ADDR1	ADDR0

```
DATASET DATA, REG, Jump ADDR
```

31 30	29 24	23 16	15 0
10	REG	ADDR	DATA

B_BRANCH 命令は INDEX で指定された変数の値が 0 であれば ADDR0 に指定されたアドレスにジャンプし, そうでなければ ADDR1 に指定されたアドレスにジャンプする. *DATASET* 命令は REG で指定されたレジスタに DATA (16 ビット) を書込む. そして ADDR で指定されたアドレスにジャンプする. 1 命令の長さを削減するため, **1 アドレス方式 2 分岐命令**も提案されている [3], [7], [15], [21]. しかし, 1 アドレス方式を用いると実行時間がしばしば増加する. そこで, 本論文では 2 アドレス方式の 2 分岐命令を用いる.

[例 2.3]　図 2 に示した MTBDD を模擬するコードを以下に示す.

```
A0:  B_BRANCH (A1,A7),x0
A1:  B_BRANCH (A2,A3),x1
A2:  DATASET 01,0,A0
A3:  B_BRANCH (A4,A5),x2
A4:  DATASET 10,0,A0
A5:  B_BRANCH (A4,A6),x3
A6:  DATASET 00,0,A0
```

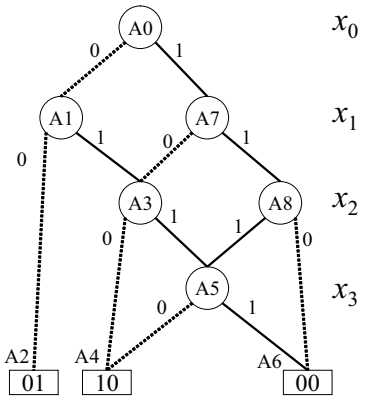


図 2　MTBDD の例.

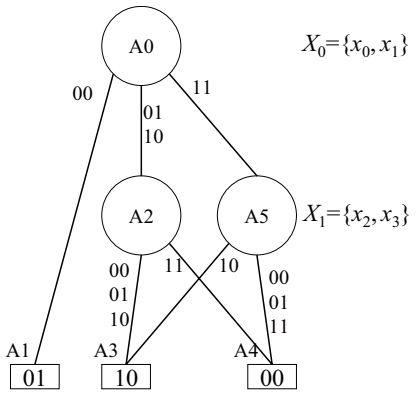


図 3　図 2 の MTBDD を MTQDD に変換した結果.

```
A7:  B_BRANCH (A3,A8),x1
A8:  B_BRANCH (A6,A5),x2
```

(例終)

2.3 QDD を評価する命令セット [18]

QDD において, 2 値変数を 2 個同時に評価し, 分岐先アドレスを 4 個に増やすことで **4 アドレス方式 4 分岐命令**を得る. 4 アドレス方式 4 分岐命令は 2 分岐命令に比べ, 評価時間を最大半分に削減できる. しかしながら, この方式は長い命令語を必要とし, 現在の組込みプロセッサの標準的な語長 (32 ビット) には適さない. 従って本論文では **3 アドレス方式 4 分岐命令 (Q_BRANCH)** を用いる. 3 アドレス方式 4 分岐命令のニーモニックと内部表現を示す.

```
Q_BRANCH (ADDR0,ADDR1,ADDR2),INDEX, SEL
```

31 30	26 25	24 23	16 15	8 7	0
0	INDEX	SEL	ADDR0	ADDR1	ADDR2

Q_BRANCH 命令は 4 つの分岐先アドレスのうち 1 つのアドレスにジャンプする. 4 つの分岐先アドレスのうち, 3 つは *ADDR0, ADDR1, ADDR2* で指定し, 残り 1 つの分岐先アドレスは現在の命令のアドレスの次のアドレス (PC+1) とする. 評価時間とステップ数削減のため, 4 種の *Q_BRANCH* 命令を用いる (図 4). *Q_BRANCH* 命令の SEL はそのうちの 1 つを指定する. i を INDEX で指定された変数の値とすると, SEL=i のとき, PC+1 にジャンプする. アドレスの割付けを工夫することでステップ数や実行時間を最適化できる. 現在の命令のアドレスの次のアドレスが他の *Q_BRANCH* 命令の PC+1 と競合した場合, **無条件ジャンプ命令**が必要となる. 無条件ジャンプは *B_BRANCH* 命令を流用する. 次の例で無条件ジャンプ命令が必要となる場合を示す.

[例 2.4]　図 5 は図 3 に示した MTQDD に 3 アドレス方式 4 分岐命令のアドレス割当てを行った結果である. SEL は図 4 に示したものと同一である. A6 が無条件ジャンプ命令である. ここでは *B_BRANCH* 命令で代用している. MTQDD を模擬するコードを示す.

```
A0:  Q_BRANCH (A2,A2,A5),X0,00
A1:  DATASET 01,0,A0
```

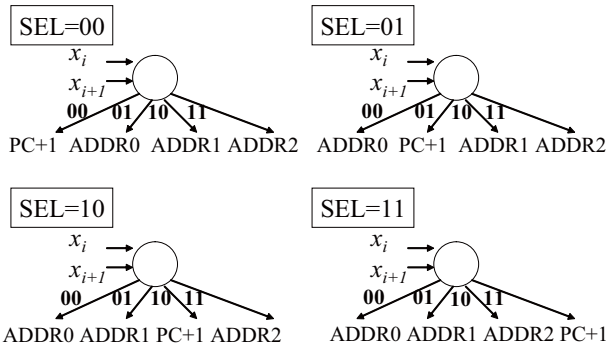


図 4　3 アドレス方式 4 分岐命令における分岐先アドレスの組合せ.

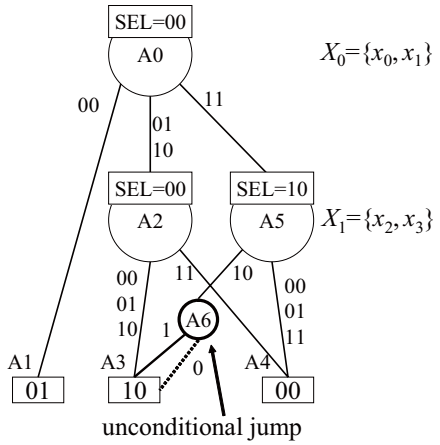


図 5　3 アドレス方式 4 分岐命令のアドレスを割当てた結果.

```
A2:  Q_BRANCH (A3,A3,A4),X1,00
A3:  DATASET 10,0,A0
A4:  DATASET 00,0,A0
A5:  Q_BRANCH (A4,A4,A4),X1,10
A6:  B_BRANCH (A3,A3),--
```

(例終)

2.4　順序回路を模擬するブランチング・プログラム・マシン

順序回路を模擬する BM を図 6 に示す(注1). **命令メモリ**は語長が 32 ビットの BM 命令を 256 個格納する. **命令デコーダ**は命令メモリから読み出した命令を解釈し, 実行を行う. **プログラムカウンタ (PC)** は現在のプログラムカウンタを保持する. 後述する RUN 信号が有効のときのみプログラムカウンタの更新を行う. **レジスタファイル**は出力レジスタや状態レジスタ, 及び**入力選択レジスタ (ISR)** や**フラグレジスタ**と呼ばれる特殊なレジスタで構成される. レジスタの更新は *DATASET* 命令で行う. 出力及び状態レジスタは**ダブルランク・フリップフロップ** [17] と呼ばれる特殊なレジスタで構成する. フラグレジスタは BM の動作を指定する特殊なレジスタであり, 各ビットは固有の役割りを持つ. **RUN ビット**は 1 のときのみ BM のプログラムカウンタの更新が行われる. **IRQ ビット**は 1 のとき割込み信号が外部に送られる. **RCOPY ビット**は 1 のときダブルランク・フリップフロップのデータ転送が行われる. **ACT ビット**は 1 のとき後述する BM を接続する回路を起動する. **FETCH ビット**は 1 のとき入力レジスタに値が取り込まれる. 図 7 にダブルランク・フリップフロップを示す. L_1, L_2 は D ラッチである. *DATASET* 命令は 16 ビット毎にしか値を更新できないため, 16 ビットよりビット数の大きな出力や状態変数を持つ回路を評価する場合, 前後の状態の値が混ざってしまう. そこで, ダブルランク・フリップフロップを用いて, 状態遷移信号 (S_Clock) が High のときに一度に値を更新する. BM はレジスタの出力をフィードバックし順序回路を実現できる. **入力レジスタ**にはこれらのフィードバックされた値や外部入力や隣接した BM からの出力が入る. これらは入力選択レジスタの値を用いて選択できる.

フラグレジスタの未設定のビットはユーザが自由に使用できる. フラグレジスタのみ**ビットセット命令 (BITSET)** を用いて特定のビットを書換え可能である. *BITSET* 命令のニーモニックと内部表現を以下に示す.

(注1)：図中, 信号線の幅が 1 の場合幅を省略することに注意.

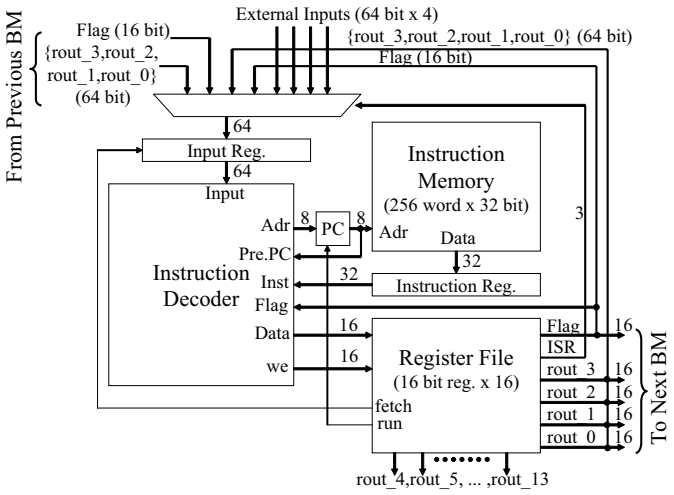


図 6　順序回路を模擬するブランチング・プログラム・マシン.

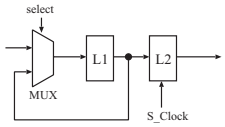
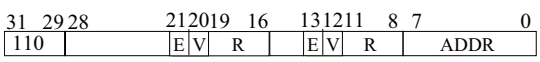


図 7　ダブルランク・フリップフロップ.

```
BITSET E1,V1,R1, E2, V2, R2, ADDR
```



BITSET 命令は $E_i = 1$ のとき, フラグレジスタの R_i 番目のビットに V_i で指定した値を書込む. $E_i = 0$ のとき, 書込みが行われず現在の値を保持する. ここで, $i = 1, 2$ である. そして, ADDR で指定したアドレスにジャンプする. *BITSET* 命令 1 つでフラグレジスタの任意の 2 ビットを書き換え可能である.

試作した BM では 1 命令を処理するのに 2 クロック使用している. 各クロックの用途は以下の通りである.

第 1 クロック: 命令メモリから命令を読み出す

第 2 クロック: プログラムカウンタに次のアドレスを取り込む

3.　並列ブランチング・プログラム・マシン

本章では 2 章で説明した順序回路を模擬するブランチング・プログラム・マシン (BM) を 128 個用いた並列ブランチング・プログラム・マシン (PBM128) について述べる. 128 個の BM を単純に直接接続し, 相互に通信可能とした場合, 接続回路は非常に大きくなり, 現実的ではない. よって, PBM128 では階層的な接続回路を用いる. 本論文では, BM を 8 台並べた **8_BM** を 1 ユニットとし, それらを 16 個並べ階層構造を構成する. 順序回路を模擬する場合, 状態遷移を行うため他の 8_BM を参照する. この際, 8_BM 間の接続には**プログラマブル接続回路**を用いる. 8_BM 内の BM 間の通信は 1clock で行える. 一方, 異なる 8_BM 間の通信は 8_BM 内の通信時間と比較して 4clock と遅い.

3.1　8_BM

図 8 に 8_BM を示す. 8_BM は 8 台の BM から構成される. BM の出力値とフラグレジスタの値はカスケード接続された**プログラマブル・ルーティングボックス**を経由し, 隣接する BM のレジスタに格納される. レジスタに格納された値は, フィードバックされ各 BM の入力に送られる. また, 256 ビットの外部入力は各 BM にブロードキャストされる. 8_BM の出力をフィードバックして各 BM で参照することもできる. 従って, 各 8_BM を独立に動作させることもできる.

プログラマブル・ルーティングボックスはコンフィギュレーションを変えることで前段の BM の出力と現在の BM の出力

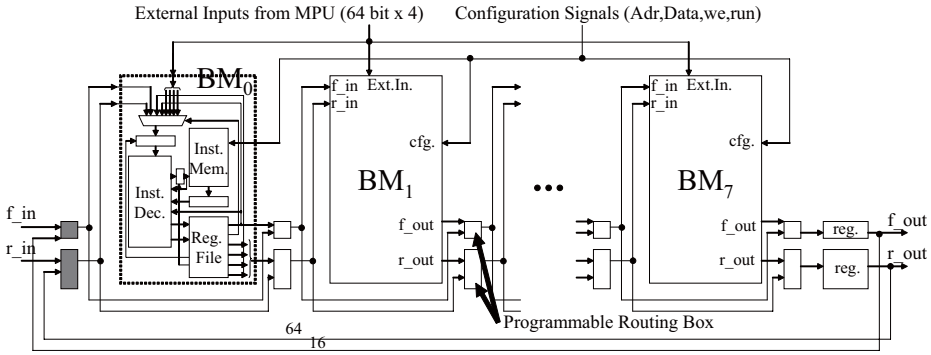


図 8　8_BM.

のビットワイズ AND やビットワイズ OR 演算ができる．また, 定数出力もできる．ビットワイズ AND 演算を行う場合, 未端 (図 8 で灰色で示された部分) のプログラマブル・ルーティングボックスに定数 1 を出力させる．一方, ビットワイズ OR 演算を行う場合, 定数 0 を出力させる．

8_BM では, 各 BM 間の通信時間はレジスタを 1 つ経由するだけなのでデータレジスタに値を書込んだ後, 1 クロック後に参照可能である．BM は 1 命令の処理に 2 クロック使用しており, 8_BM 内であれば命令レベルの通信時間の遅延は発生しない．

3.2　並列ブランチング・プログラム・マシン (PBM128)

図 9 に 128 個の BM を用いた並列ブランチング・プログラム・マシン (PBM128) を示す．PBM128 は 16 ユニットの 8_BM とそれらを接続するプログラマブル接続回路から構成されている．256 ビットの外部入力とコンフィギュレーション信号は各 8_BM にブロードキャストされる．コンフィギュレーション信号は, アドレス信号, 書込みイネーブル信号, プログラマブル接続回路の**接続指定信号**からなる．各 8_BM の 64 ビットの出力は上位 32 ビットが外部に直接出力され, 下位 32 ビット (状態変数) はプログラマブル接続回路を通してフィードバックされ, 指定された 8_BM に接続される．また, フラグレジスタの値はビットワイズ AND またはビットワイズ OR され, 各 8_BM にフィードバックされる．これらはコンフィギュレーション信号で指定する．全ての 8_BM の演算が終った後に状態遷移を行うため, 各フラグレジスタの ACT ビットの AND をとった**接続信号**をプログラマブル接続回路に転送する．接続信号が 1 になると, プログラマブル接続回路が起動する．

3.3　プログラマブル接続回路

プログラマブル接続回路は複雑な信号選択を行うため, 多段のマルチプレクサを必要とする．単純な構成では, 8_BM 選択回路がクリティカルパスとなり全体の動作周波数が極端に低下する．よって, 8_BM 選択回路内にはパイプライン・レジスタを挿入し, 動作周波数を向上させている．ただし, パイプライン・レジスタを挿入したため, プログラマブル接続回路の出力を確定させるために 4 クロック必要である．PBM128 では 1 命令につき 2 クロック使用しているため, プログラマブル接続回路を通してデータを転送するには 2 命令待つ必要がある．コード生成時, この遅延を考慮しなければならない．

4.　実 験 結 果

4.1　並列ブランチング・プログラム・マシンの実装

PBM128 を FPGA 上に実装した．PBM128 のコンフィギュレーション, 制御, 入出力の受渡しのため, Altera 社の組込みプロセッサ NiosII/f を組込んだ．また, コンフィギュレーションデータは外部に SD-Card を取付け, FPGA 内に組込んだ SD-Card コントローラで読み込むようにした．実装に用いた FPGA と開発ツールを表 1 に示す．また, 実装結果のハードウェア量を纏めたものを表 2 に示す．実装結果より, PBM128 は最大 132MHz の動作周波数で動作することが確認できた．PBM128 は ALUT を 67817 個使用しており, これは実装に用いた FPGA の全 ALUT の 64%に相当する．8_BM1 個当り PBM128 で費した ALUT の 5.6%を費した．また, 16 個の 8_BM で PBM128

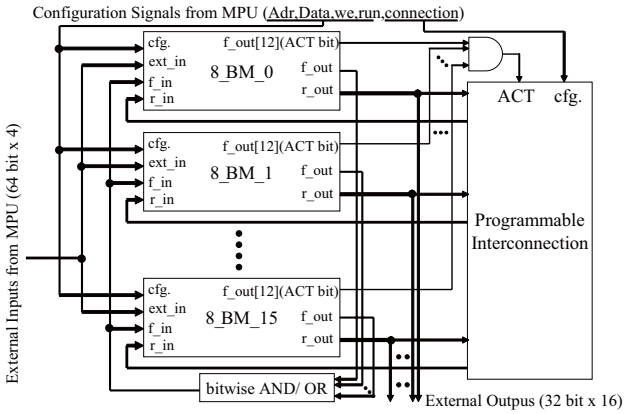


図 9　並列ブランチング・プログラム・マシン (PBM128).

表 1　実装に用いた FPGA と開発ツール.

FPGA Board: PowerMedusa EM300-MU130
FPGA: Altera StratixII EP2S130F1508C4 ALUTs: 106032 User I/Os: 1126 Memory bits: 6747840 (M4Ks): 609
Development Tools: Altera Quartus II ver.7.2

表 2　PBM128 のハードウェア資源.

	ALUTs	M4Ks
PBM128	67817(100%)	256
(8_BM)	3778(5.6%)	16
(BM)	455(0.6%)	2
(プログラマブル接続回路)	6307(9.3%)	0
Max.Freq	132.73[MHz]	

で費した ALUT の 88%を費している．又, プログラマブル接続回路は PBM128 で費した ALUT の 9.3%を費している．なお, ハードウェア量に関しては NiosII/f と SD-Card コントローラの部分を除いている．

4.2　Intel Core2Duo との比較

様々なベンチマーク関数 [20] を並列ブランチング・プログラム・マシン (PBM128) 上で動作させ, ステップ数と実行時間について汎用プロセッサと比較を行った．比較に用いた汎用プロセッサは Intel 社の Core2Duo U7600(1.2GHz, Cache L1 data 32KB, L1 inst. 32KB, L2 2MB) である．コンパイラは gcc を用い, 最適化オプションは O3 とした．ベンチマーク関数は MCNC ベンチマーク関数を用いた．選択したベンチマーク関数は入出力数が多く, 単一の MTQDD で表現できない．よって, 出力を分割し, 複数の MTQDD で表現し, BM のコードに変換した．出力分割法はなるべく入力変数が類似するように分割する方法を用いた [14]．PBM128 で用いたデータ構造は MTQDD である．一方, Core2Duo で実現したデータ構造は MTBDD を用いた．各 MTBDD の非終端節点を if-then-else 分岐命令に置換えたコードを生成した．これは汎用プロセッサの場合, BDD の方がコードが単純であり, キャッシュにヒットしやすく, QDD

よりも高速であったためである. 10 万個のランダムテストベクトルを評価する時間を測定し, 1 ベクトルを評価する平均時間を求めた. PLL を安定に動作させるため, PBM128 は 100[MHz] で動作させた. 一方, Core2Duo は 1.2[GHz] で動作させた. メモリ量としては, Core2Duo は BDD の実行コードのみを考慮し, テストベクトル生成部と時間測定の実行コードは無視した. 一方, PBM128 はステップ数から必要なメモリ量を求めた. メモリ量と実行時間を比較した結果を表 3 に示す. 表 3 において, *Name* はベンチマーク関数名, *In* は入力数, *Out* は出力数, *FF* は状態変数の個数, *Code* は実行コードのメモリ量を表し単位は [KBytes] である. *Time* は実行時間を表し単位は [nsec] である. *Total Grp* は分割したグループ数を表し, *BNode* は MTBDD のノード数の総和を表し, *QNode with UJ* は MTQDD のノード数と無条件ジャンプ命令の和を表し, *Total APL* は回路分割後の各平均パス長 (APL) の総和を表し, *Max. APL* は各グループの APL の中で最大の APL を表す. 表 3 より, PBM128 の実行コードのメモリ量は Core2Duo の約 4 分の 1 であり, 速度は 21.4〜96.1 倍であることがわかる.

a） メモリ量の解析

汎用プロセッサで BDD を実現する場合, 非終端節点を表現する C コードとアセンブラコードは図 10 に示すコードとなる.

```
// C-code                  // Assembly(x86)-code
A_1:                       A_1:
if( x[2] & 0x001)          movl %eax,_x+8
goto A_2;                  testb %al,$1
else                       je A_10
goto A_10;                 A_2:
A_2:                       movl %eax,_x+4
if( x[1] & 0x002)          testb %al,$2
goto A_4;                  jne A_4
else
goto A_3;
```

図 10 C-code and Assembly-code for BDD.

BM の 2 分岐命令を汎用プロセッサで実現した場合

1. 入力変数を読み込む
2. マスクをかけ, 指定した入力値を取り出す
3. 入力値に応じて指定先にジャンプ

である. これは 1 アドレス方式 2 分岐命令で BDD を評価しているといえる.

Est.Step.MPU を MPU のステップ数の見積り値とする. MPU は MTBDD を評価するので, MTBDD の 1 節点につき 3 命令必要である. 従って,

$$Est.Step.MPU = BNode \times 3 \tag{1}$$

である. ここで *BNode* は表 3 で示した値である. *Est.Step.PBM* を PBM128 のステップ数の見積り値とする. PBM は MTQDD を評価するので, ステップ数は MTQDD の節点数と無条件ジャンプ命令数の和に比例する. 従って,

$$Est.Step.PBM = QNode\ with\ UJ \tag{2}$$

である. *QNode with UJ* は表 3 で示した値である. *Est.Ratio.Code* をコードサイズの比率の見積り値とする. ステップ数に 4 を掛けるとコードサイズ [Byte] が得られるので式 (1), (2) より,

$$\begin{aligned} Est.Ratio.Code &= \frac{Est.Step.MPU \times 4}{Est.Step.PBM \times 4} \\ &= \frac{BNode}{QNode\ with\ UJ} \times 3 \end{aligned} \tag{3}$$

である. 表 3 より, *Est.Ratio.Code* は 3.99〜4.53 であり, *Act.Ratio.Code* は 4.00〜4.79 であった. よって *Est.Ratio.Code* は実際の比率を見積もれていることがわかる. つまり, Core2Duo は PBM128 よりも約 4 倍ステップ数を必要とする.

b） 実行時間の解析

評価時間の見積り値 *Est.Time* を以下のように求めた.

$$Est.Time = ETPI \times IPN \times APL + TST \tag{4}$$

式 (4) において, *ETPI* は 1 命令を実行するのに必要な時間 [nsec/inst], *IPN* は 1 節点を評価するのに必要な命令数 [inst/node], *APL* は 1 つの入力ベクトルを評価するために訪れる平均節点数 (Average Path Length: APL)[node], *TST* は状態遷移を行うのに必要な時間 [nsec] を表す.

PBM128 の実行時間の見積りを *Est.Time.PBM* とする. PBM128 では 1 命令を実行するのに 2 クロック使用している. 実験では 100[MHz](=10[nsec]) で動作させたので, $ETPI_{PBM} = 20[nsec/inst]$ である. PBM128 は 1 節点を 1 命令で評価できるので, $IPN_{PBM} = 1.0[inst/node]$ である. また, PBM128 は分割した QDD を並列に動作させるので, 実行時間は各グループの中で最大の APL に拘束される. 従って, $APL_{PBM} = \max\{QAPL_i\} = Max.APL$ である. なお, $QAPL_i$ はグループ i を表現する QDD の平均パス長であり, $Max.APL$ は表 3 で示した値である. PBM128 ではプログラマブル接続回路が 4 クロックで状態変数のフィードバックを行う. 従って, $TST_{PBM} = 40[nsec]$ である.

汎用プロセッサの実行時間の見積りを *Est.Time.MPU* とする. 汎用プロセッサの 1 命令を実行する時間 $ETPI_{MPU}$ はキャッシュアクセス時間に影響を受ける. つまり, 同じ命令やデータを処理する場合でも異なるキャッシュにアクセスすれば, 実行時間も異なる. キャッシュの違いによるアクセス時間の影響を調べる実験を行ない, L1 キャッシュの平均アクセス時間は約 3[nsec] であり, L2 キャッシュの平均アクセス時間は約 15[nsec] であることを求めた. アセンブラコードの解析結果から, BDD の節点を評価するには変数読出し (mov), 変数のマスク (test), 条件分岐 (jump) の 3 命令が必要であることを確かめた. BDD の条件分岐先はほぼランダムであるから, 最初の変数読み出しでキャッシュミスが起こると考えられる. 連続する残りの命令はプリフェッチされるので, キャッシュミスは隠蔽される. L1 と L2 キャッシュのアクセス時間を T_{L1}, T_{L2} とし, L1 キャッシュの容量を M_{L1} とし, BDD のメモリ量を M_{BDD} とする. 1 節点を評価する 3 つの命令のうち, キャッシュミスが考えられる load 命令の実行時間を T_{mov} とする. T_{mov} はそれぞれのキャッシュアクセス時間とヒット率を掛けた値の和であるから

$$T_{mov} = \frac{T_{L2}(M_{BDD} - M_{L1}) + T_{L1}M_{L1}}{M_{BDD}} \tag{5}$$

となる. 従って, $ETPI_{MPU}$ は

$$ETPI_{MPU} = \begin{cases} T_{L1} & (M_{BDD} \leqq M_{L1}) \\ \frac{T_{mov}+2\times T_{L1}}{3} & (M_{BDD} > M_{L1}) \end{cases} \tag{6}$$

となる. 上の式は BDD が L1 キャッシュに収まる場合の ETPI を表し, 下の式は BDD が L1 キャッシュに収まらない場合の ETPI を表す. 下の式において, 分子の第 2 項は test と jump 命令の実行時間の見積りを表す. アセンブラコードの解析から $IPN_{MPU} = 3.0$ を得た. また, 汎用プロセッサでは分割した BDD をシングルコアで逐一評価するため $APL_{MPU} = \sum_{i=0}^{g-1} BAPL_i = Total.APL$ である. ここで, $BAPL_i$ はグループ i を表現する BDD の平均パス長であり, g は分割したグループ数であり, $Total.APL$ は表 3 で示した値である. 汎用プロセッサは状態遷移を各変数毎に逐一行わなければならない. 状態変数は 3[nsec] でアクセスできる L1 キャッシュに収まると仮定し, $TST_{MPU} = \#FF \times 3$ とした. ここで, $\#FF$ は表 3 で示した状態変数の個数を表す.

Est.Ratio.Time を実行時間の比率の見積り値とすると

$$Est.Ratio.Time = \frac{Est.Time.MPU}{Est.Time.PBM} \tag{7}$$

である. 表 3 より, *Est.Ratio.Time* は 19.1〜98.9 であり, *Act.Ratio.Time* は 21.4〜96.1 であった. よって, *Est.Ratio.Time* は実際の比率を見積もれていることがわかる.

これらの解析より, PBM128 が MPU より高速である理由は

表 3　メモリ量と実行時間の比較.

Name	In	Out	FF	Total Grp.	Core2Duo@1.2GHz				PBM128@100MHz				Ratio (Core2Duo/PBM128)			
					BNode	Total APL	Code [KB]	Time [ns]	QNode with UJ	Max. APL	Code [KB]	Time [ns]	Ratio.Code		Ratio.Time	
													Est.	Act.	Est.	Act.
s5378	35	49	164	126	5702	703.9	74.6	12030	4131	13.1	17.8	323	4.14	4.19	49.1	37.2
s9234	36	39	211	117	10963	590.7	148.6	13450	7613	14.6	33.4	352	4.32	4.44	46.9	38.2
dsip	229	197	224	120	7907	649.3	112.1	17500	5342	6.1	24.8	182	4.44	4.52	95.0	96.1
bigkey	263	197	224	125	9971	831.7	149.5	19170	6876	8.0	33.9	220	4.35	4.41	98.9	87.1
apex6	135	99	0	99	1535	297.1	23.0	3700	1016	5.0	4.8	163	4.53	4.79	21.8	22.6
cps	24	102	0	102	3035	242.5	33.9	3468	2121	5.1	8.3	162	4.29	4.08	19.1	21.4
des	256	245	0	124	8952	770.3	123.1	16560	6730	12.4	30.7	308	3.99	4.00	50.2	53.7
frg2	143	139	0	116	3161	529.9	40.0	6390	2226	7.7	9.2	215	4.26	4.34	28.0	29.7

1. キャッシュミスを回避させたこと
2. 特殊な *Q_BRANCH* 命令を用意したこと
3. 128 台の BM を並列に動作させたこと,

によると言える.

5. まとめと今後の課題

本論文では, 128 台のブランチング・プログラム・マシン (BM) を用いた並列ブランチング・プログラム・マシン (PBM128) について述べた. 順序回路を模擬する BM を 8 台並列に並べたものを 8_BM とし, PBM128 は 16 台の 8_BM とそれらを接続するプログラマブル接続回路からなる. PBM128 の性能向上とステップ数削減のため, 3 アドレス方式 4 分岐命令を実装した. ベンチマーク関数を実現し, Intel 社の Core2Duo と比較を行った結果, PBM128 の実行コードのメモリ量は Core2Duo の約 4 分の 1 であり, 速度は 21.4〜96.1 倍であった.

PBM128 が Core2Duo よりも高速な理由は以下の通り.

1　2 変数を同時に評価する命令を用意したこと
2　128 台の BM を並列動作させたこと
3　分散メモリとメモリ量を削減する命令を用意し, キャッシュミスを回避したこと

本論文で用いたコード生成ツールは既存の手法を流用したものであり, 必ずしも PBM128 に適したものではない. 今後の課題は PBM128 に適したコード生成ツールを考案することである. また, PBM128 は順序回路の模擬以外の用途にも対応可能である. PBM128 に適した他の応用の開発も今後の課題である.

6. 謝　　辞

本研究は, 一部, 日本学術振興会・科学研究費補助金, および, 文部科学省・知的クラスター創成事業 (第二期) の補助金による. 日立情報制御ソリューションズ梶原久志氏には有益な助言を頂いた.

文　　献

[1] P. Ashar and S. Malik, "Fast functional simulation using branching programs," *Proc. Internaitional Conference on Computer Aided Design,* pp. 408-412, Nov. 1995.

[2] P. C. Baracos, R. D. Hudson, L. J. Vroomen, and P. J. A. Zsombor-Murray, "Advances in binary decision based programmable controllers," *IEEE Transactions on Industrial Electornics,* Vol. 35, No. 3, pp. 417-425, Aug., 1988.

[3] R. T. Boute, "The binary-decision machine as programmable controller," *Euromicro Newsletter,* Vol. 1, No. 2, pp. 16-22, 1976.

[4] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677-691, Aug. 1986.

[5] J. T. Butler, T. Sasao, and M. Matsuura, "Average path length of binary decision diagrams," *IEEE Trans. Compt.*, Vol. 54, No. 9, pp. 1041-1053, Sep. 2005.

[6] C. H. Clare, *Designing Logic Systems Using State Machines,* McGraw-Hill, New York, 1973.

[7] M. Davio, J.-P Deschamps, and A. Thayse, *Degital Systems with Algorithm Implementation,* Jhon Wiley & Sons, New York, 1983, p.368.

[8] Y. Iguchi, T. Sasao, M. Matsuura, and A. Iseno, "A hardware simulation engine based on decision diagrams," *Asia and South Pacific Design Automation Conference'2000,* Yokohama, Japan, pp. 73-76, Jan. 26-28, 2000.

[9] Y. Iguchi, T. Sasao, and M. Matsuura, "Implementation of multiple-output functions using PROMDDs," *30th International Symposium on Mulitple-Valued Logic,* Portland, Oreon, U.S.A, pp. 199-205, May 23-25 2000.

[10] Y. Iguchi, T. Sasao, and M. Matsuura, "Evaluation of multiple-output logic functions," *Asia and South Pacific Design Automation Conference'2003,* Kitakyushu, Japan, pp. 312-315, Jan. 21-24, 2003.

[11] T. Kam, T. Villa, R. K. Brayton, and A. L. Sagiovanni-Vincentelli, "Multi-valued decision diagrams: Theory and Applications," *Multiple-Valued Logic,* Vol. 4, No. 1-2, pp. 9-62, 1998.

[12] S. Nagayama, T. Sasao, Y. Iguchi, and M. Matsuura, "Area-time complexities of multi-valued decision diagrams," *IEICE Transactions on Fundamentals of Electronics,* Vol. E87-A, No. 5, pp. 1020-1028, May 2004.

[13] S. Nagayama, and T. Sasao, "On the optimization of heterogeneous MDDs," *IEEE Transactions on CAD,* Vol. 24, No. 11, pp. 1645-1659, Nov. 2005.

[14] H. Nakahara, T. Sasao, and M. Matsuura, "A Design algorithm for sequential circuits using LUT rings," *IEICE Transactions on Fundamentals of Electronics*, Communications and Computer Sciences, Vol. E88-A, No. 12, pp. 3342-3350, Dec. 2005.

[15] D. Mange, "A high-level-language programmable controller: Part I-II," *IEEE Micro,* Vol. 6, No. 1, pp. 25-41 (Part I), Vol. 6, No. 2, pp. 47-63 (Part II), Feb/Mar, 1986.

[16] P. C. McGeer, K. L. McMillan, A. Saldanha, A. L. Sangiovanni-Vincentelli, and P. Scaglia, "Fast discrete function evaluation using decision diagrams," *Proc. International Conference on Computer Aided Design,* pp. 402-407, Nov. 1995.

[17] T. Sasao, H. Nakahara, M. Matsuura and Y. Iguchi, "Realization of sequential circuits by look-up table ring," *The 2004 IEEE International Midwest Symposium on Circuits and Systems,* Hiroshima, pp.I:517-I:520, July 25-28, 2004.

[18] T. Sasao, H. Nakahara, M. Matsuura, Y. Kawamura, and J.T. Butler," "A quaternary decision diagram machine and the optimization of its code," *39th International Symposium on Multiple-Valued Logic (ISMVL 2009)*, (accepted).

[19] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Computing Curveys,* Vol. 37, Issue 3, pp. 238-275, Sep. 2005.

[20] S. Yang, "Logic synthesis and optimization benchmark user guide version 3.0," *MCNC*, Jan. 1991.

[21] P.J.A.Zsombor-Murray, L.J. Vroomen, R.D. Hudson, Le-Ngoc Tho, and P.H. Holck, "Binary-decision-based programmable controllers, Part I-III," *IEEE Micro* Vol. 3, No. 4, pp. 67-83 (Part I), No. 5, pp. 16-26 (Part II), No. 6, pp. 24-39 (Part III), 1983.