FPGA を用いた充足可能性問題の高速計算

金澤 健治

システム情報工学研究科

筑波大学

２０１２年 ３月

目 次

第 1 章　序論 **7**

第 2 章　充足可能性問題 **9**
2.1 問題の定義と表現形式 9
2.2 問題の分類 10
2.2.1 k-SAT 問題 10
2.2.2 3-SAT 問題 11
2.3 問題の応用 11
2.3.1 形式検証 11
2.3.2 その他の応用 13

第 3 章　アルゴリズム **15**
3.1 Complete 型アルゴリズム 15
3.2 Incomplete 型アルゴリズム 17
3.2.1 Walksat アルゴリズム 17

第 4 章　FPGA ソルバー **20**
4.1 FPGA ソルバー向きアルゴリズム 20
4.2 先行研究 21
4.2.1 Complete 型ソルバー 21
4.2.2 Incomplete 型ソルバー 22
4.2.3 先行研究例の分類 23
4.3 ソルバー概要 24
4.3.1 ソルバーのハードウェア構成 24
4.3.2 3-SAT 問題用ソルバー 1 25
4.3.3 3-SAT 問題用ソルバー 2 25
4.3.4 3-SAT 問題用ソルバー 3 26
4.3.5 3-SAT 問題用ソルバー 4 26
4.3.6 形式検証用大規模ソルバー 27

第 5 章　3-SAT 問題用ソルバー 1 **28**
5.1 ハードウェアの特徴 28
5.2 処理の概略 28
5.3 並列/パイプライン処理の詳細 30
5.4 回路の詳細 31
5.4.1 クローズ評価ユニット 31
5.4.2 評価値の計算とクローズの選択 34
5.4.3 問題データのダウンロード時間 35
5.5 評価結果 36
5.6 有効並列度に関する考察 38

第 6 章　3-SAT 問題用ソルバー 2 **39**
6.1 ハードウェアの特徴 39
6.2 処理の概略 40
6.3 並列/パイプライン処理の詳細 42
6.4 回路の詳細 43
6.4.1 Address Translation Table と Clause Table 43
6.4.2 各ユニットの詳細 44
6.4.3 非充足クローズの選択方法とその影響 46
6.4.4 乱数とその影響 46
6.5 評価結果 47

第 7 章　3-SAT 問題用ソルバー 3 **49**
7.1 処理の概略 49
7.1.1 基本構成 49
7.1.2 回路構成 50
7.2 回路の詳細 52
7.2.1 並列処理とマルチスレッド処理 52
7.2.2 各スレッド内の並列性 53
7.2.3 Address Translation Table 54
7.2.4 Unsatisfied Clause Buffer 55
7.3 評価結果 56

第 8 章　3-SAT 問題用ソルバー 4 **58**
8.1 ハードウェアの特徴 58
8.2 外部メモリの使用方法 58

8.3 処理可能な問題規模と期待される性能 59

第 9 章 形式検証用大規模ソルバー 60

9.1 Walksat アルゴリズムの改良 60

9.1.1 論理ゲートの検出方法 61

9.1.2 改良アルゴリズムの詳細 63

9.2 改良アルゴリズムの性能 65

9.3 実装方式の概略 66

9.3.1 データ構造と処理の概略 66

9.3.2 SDRAM を使用した場合の処理並列度 69

9.3.3 データマッピングとハードウェアの詳細 69

9.3.4 並列処理の詳細 73

9.4 性能に関する考察 75

第 10 章 各実装方式の処理性能に関する考察 79

10.1 メモリアクセス速度と処理性能 79

10.2 問題規模と処理性能 81

第 11 章 結論 85

11.1 まとめ 85

11.2 今後の課題 86

11.2.1 アルゴリズムの改良 86

11.2.2 より効率的なデータキャッシュ方法についての検討 86

11.2.3 形式検証以外の応用問題に対する回路の検討 87

謝辞 88

参考文献 89

研究業績 96

図 目 次

2.1 CNF-SAT 9
2.2 DIMACS 書式の CNF-SAT 10
2.3 組合せ回路の等価性検証 12
2.4 有界モデル検査 13

3.1 DPLL アルゴリズム 15
3.2 Walksat アルゴリズム 18

4.1 ソルバーのハードウェア構成 24

5.1 回路のブロック図 (3-SAT ソルバー 1) 29
5.2 マルチスレッド処理 (3-SAT ソルバー 1) 30
5.3 Clause Evaluator の詳細 (3-SAT ソルバー 1) 32
5.4 Current Position の管理手法 33
5.5 Variable Number Arrays 初期化ユニット 36

6.1 回路のブロック図 (3-SAT ソルバー 2) 40
6.2 マルチスレッド処理 (3-SAT ソルバー 2) 42
6.3 Address Translation Table と Clause Table の詳細 (3-SAT ソルバー 2) 43
6.4 Clause Evaluator の詳細 (3-SAT ソルバー 2) 45

7.1 基本構成 (3-SAT ソルバー 3) 49
7.2 回路の詳細 (3-SAT ソルバー 3) 50
7.3 タイミングチャート (3-SAT ソルバー 3) 52
7.4 Address Translation Table(3-SAT ソルバー 3) 54

8.1 回路のブロック図 (3-SAT ソルバー 4) 59

9.1 AND/OR ゲートの CNF 表現 61
9.2 出力信号の検出 63

9.3 改良アルゴリズム 64
9.4 テーブル概略 (形式検証用ソルバー) 66
9.5 usc_buf[] のデータ構造 71
9.6 形式検証用ソルバーのブロック図 72
9.7 v_tbl[] の詳細 73
9.8 ソフトウェアに対する速度向上率 75

10.1 変数の数と処理性能 81
10.2 クローズの数と処理性能 82
10.3 変数の数と LUT 数の関係 83
10.4 変数の数と Block RAM 数の関係 84

表 目 次

4.1 先行研究例の分類 23

5.1 マルチスレッド処理の性能 (3-SAT ソルバー 1) 31
5.2 Unfair Clause Selection によるフリップ回数 34
5.3 性能比較 (3-SAT ソルバー 1) 37

6.1 マルチスレッド処理の性能 (3-SAT ソルバー 2) 43
6.2 性能比較 (3-SAT ソルバー 2) 47

7.1 非充足クローズのキャッシュを行う場合のフリップ回数 55
7.2 性能比較 (3-SAT ソルバー 3) 56

9.1 改良アルゴリズムの性能 65
9.2 各ベンチマークのリテラル数と *clauses list* 長さ 68
9.3 バーストリードの割合，およびクローズの平均リテラル数と *clauses list* 長さの平均 77

10.1 問題規模/メモリアクセス速度と処理性能 79

第1章　序論

充足可能性問題 (Boolean Satisfiability Problem: SAT 問題) は組合せ問題のひとつであり，命題論理式 $F(x_1, x_2, ..., x_n)$ が与えられたとき，F を充足するような，論理変数 $(x_1, x_2, ..., x_n)$ への真偽割当が存在するかどうかを判定する問題である．この問題は，最悪のケースにおいて計算量が指数関数的に増大する，NP 完全問題であることが最初に示された問題であり [1]，ハードウェア・ソフトウェアの形式検証，回路のテストパターン自動生成，プランニング，スケジューリング，回路の配線の探索 (Routing) など，数多くの応用問題に帰着することが可能である [2][3][4][5][6] [10][11][12][13][16][17][19][20][21]．充足可能性問題の解法アルゴリズムは近年急速な進歩を遂げており [3][12][13]，様々な応用問題を充足可能性問題に変換して処理することが現実的なものとなりつつある．一方，実際の応用問題から生成される充足可能性問題の規模は極めて大きく，非常に長い計算時間を必要とする場合がしばしば存在する．ゆえに，充足可能性問題の高速計算にあたっては，効率のよい解法アルゴリズムの開発だけでなく，アルゴリズムを高速に実行することができる専用ハードウェアの開発が望まれる．

Field Programmable Gate Array(FPGA) は，再構成可能な LSI であり，処理する問題に応じて最も適した回路をその都度構成することができる．半導体製造技術の進歩とともに FPGA の大規模化，高性能化もまた進んでおり，単一の FPGA を用いて様々な応用問題を高速に処理することが可能となりつつある．特に対象とする問題に高い並列性が内在する場合には，多くの処理を同時に実行することにより，ソフトウェアと比較して非常に高い性能を得ることができる．

確率的局所探索アルゴリズム (Stochastic Local Search Algorithm) は，充足可能性問題の解法アルゴリズムとして非常に有効であり，Walksat アルゴリズム [29][30][31][28] は，その中で最も性能の高いアルゴリズムのひとつである．Walksat は複雑な制御を必要しないこと，処理の並列性が高いことから，ハードウェアへの実装に適したアルゴリズムであるといえる．

本論文では，FPGA を用いた Walksat アルゴリズムに基づく充足可能性問題の高速計算システムを提案し，その有効性を示す．ハードウェアの設計においては，問題のデータを (1) 全て FPGA 内，(2) 外部メモリに保持するという 2 つの大きな

選択肢がある．前者により，より高い性能を実現することができるが，処理可能な問題の規模はFPGAの回路規模により制限される．後者により，より大規模な問題を処理可能となるが，その性能は外部メモリのデータ転送速度により制限される．

本研究では，複数の外部メモリバンクを有するFPGAボードとホストコンピュータから構成されるシステムにおいて，処理する問題の規模に応じ，異なる5種類の回路実装方式の検討を行った．そして，それぞれの実装方式において，ソフトウェアと比較して高い処理性能が得られることを確認するとともに，FPGAを用いることにより，様々な規模のSAT問題において，ソフトウェアに対してどの程度の高速化が可能であるかを明らかにした．

本論文の構成は以下の通りである．第2章では，充足可能性問題について，その詳細を述べる．第3章では，充足可能性問題の解法アルゴリズムについて説明する．第4章では，過去の研究例について述べ，その課題を明らかにするとともに，本論文で提案する高速計算システムの概要について述べる．第5章から第9章では，提案する異なる5つの回路実装方式について，詳細をそれぞれ述べる．第10章では，各実装方式の性能についての考察を行い，問題規模，メモリアクセス速度と処理速度との関係を明らかにする．最後に，第11章で本論文のまとめを行うとともに，今後の課題について述べ，論文を閉じる．

第2章　充足可能性問題

2.1　問題の定義と表現形式

充足可能性問題 (SAT 問題) は組合せ問題のひとつであり，命題論理式 F の充足解を求める，あるいは F が充足不可能であることを示す問題である．命題論理式 $F(x_1, x_2, \cdots, x_n)$ が与えられたとき，F の値を真にするような論理変数 $(x_1, x_2, ..., x_n)$ への真偽割当が存在するとき，F は充足可能であるといい，そのときの真偽割当を F の充足解という．これに対して，そのような真偽割当が存在しないとき，F は充足不可能であるという．SAT 問題の充足可能性を判定するシステムのことを SAT ソルバーと呼ぶ．

F が和積形式 (Conjunctive Normal Form: CNF) で与えられる問題は CNF-SAT と呼ばれ，多くの場合，SAT 問題は CNF で表現される．図 2.1 に具体例を示す．

$$
\begin{aligned}
F(x_1, x_2, x_3) &= C_1(x_1, x_2, x_3) \wedge C_2(x_1, x_2, x_3) \wedge C_3(x_1, x_2, x_3) \wedge C_4(x_1, x_2, x_3) \\
C_1(x_1, x_2, x_3) &= \neg x_1 \vee x_2 \\
C_2(x_1, x_2, x_3) &= \neg x_2 \vee \neg x_3 \\
C_3(x_1, x_2, x_3) &= x_1 \vee \neg x_2 \vee x_3 \\
C_4(x_1, x_2, x_3) &= x_1 \vee \neg x_3
\end{aligned}
$$

図 2.1: CNF-SAT

上記の式において，C_1, C_2, C_3, C_4 はクローズと呼ばれる．CNF-SAT では，論理式 F はクローズの論理積で表され，個々のクローズはリテラルの論理和で表される．変数 x およびその論理否定 $\neg x$ をリテラルと呼び，前者を正リテラル，後者を負リテラルと呼ぶ．あるリテラル $l(\neg l)$ に対して，その論理否定 $\neg l(l)$ は $l(\neg l)$ の補リテラルと呼ばれる．CNF-SAT が充足可能であるためには，全てのクローズが真でなければならず，そのためには各クローズに含まれるリテラルのうち，それぞれ少なくともひとつが真でなければならない．この例の場合，$(x_1, x_2, x_3) = (1, 1, 0)$ は F の充足解である．

任意の命題論理式は，その充足可能性が同値であるような CNF-SAT 問題に線形時間で変換可能であることから，対象とする問題を CNF-SAT に限定しても一般性を失うことはない [13].

SAT ソルバーの性能を競う競技会である SAT Competitions[67] や SAT Race [68] [69][70] におけるベンチマーク問題は，CNF-SAT を整数列および文字列で表現した DIMACS CNF[71] と呼ばれる書式で与えられており，SAT ソルバーの多くは DIMAC CNF 書式の CNF-SAT を入力として受け付ける．図 2.2 に，図 2.1 の例題の DIMACS CNF 書式を示す.

```
p cnf 3 4
c This is an example instance.
-1 2 0
-2 -3 0
1 -2 3 0
1 -3 0
```

図 2.2: DIMACS 書式の CNF-SAT

DIMACS CNF 書式のベンチマーク問題は，p で始まる問題行 (problem line)，c で始まるコメント行，およびクローズから構成される．問題行には，問題の表現形式，変数の数，クローズの数を記述する．図 2.1 の問題の場合，3 変数 4 クローズの CNF-SAT であることから，問題行は以下のように記述する.

```
p cnf 3 4
```

問題行およびコメント行以外の行には，1 行につき 1 つのクローズを記述する．正リテラルは変数番号で表現し，負リテラルは変数番号の先頭にマイナス符号をつけて表現する．論理和は記述せず，クローズの末尾には 0 をつける.

2.2 問題の分類

2.2.1 k-SAT 問題

CNF-SAT のうち，クローズに含まれるリテラル数が全て k 個のものを k-SAT と呼ぶ． k-SAT($k \geq 3$) は，NP 完全問題であることが最初に証明された問題であ

る [1]．すなわち，解を得るために必要な計算量は，最悪の場合，指数関数的に増加する．

2.2.2 3-SAT 問題

k-SAT のうち，$k = 3$ のものを 3-SAT と呼ぶ．任意の k-SAT は補助的な変数を導入することにより，3-SAT に容易に変換することができる [22]．3-SAT 問題は，変数の数 N_v とクローズの数 N_c との比により，解の存在する確率や，解が求まるまでの計算量が変化することが知られている [8][9]．$N_c/N_v \fallingdotseq 4.3$ のとき，解の存在する確率はおよそ 50%となり，また解が求まるまでの計算量が最も多くなることから，このとき問題は最も困難になるとされている [8][9]．そのため，人工的に生成される 3-SAT 問題は，しばしば $N_c/N_v \fallingdotseq 4.3$ となるように生成される．

2.3 問題の応用

充足可能性問題は，ハードウェア・ソフトウェアの形式検証，回路のテストパターン自動生成，プランニング，スケジューリング，回路の配線の探索 (Routing) など数多くの応用問題と等価である．このため，これらの問題は充足可能性問題に変換し，解くことができる．本節では，代表的な応用例である形式検証，およびその他の応用について，その概要を説明する．

2.3.1 形式検証

形式検証は，システムの設計に誤りがないことを数学的に検証する手法であり，テストパターンを用いたシミュレーション等と比較して網羅性の高い検証が可能であるという長所を持つ [4]．充足可能性問題を用いた形式検証では，与えられた設計と検証仕様から SAT 問題を生成し，その問題の充足可能性を判定することで，その設計に誤りがないかどうかを検証する．形式検証は，2 つの設計が等価であることを検証する等価性検証と，与えられた設計が特定の性質を満足するかどうかを検証するモデル検査に分けられる [3, 4]．以下，ハードウェアの形式検証を例に，それぞれについて説明する．

等価性検証

ハードウェアの等価性検証は，組合せ回路の等価性検証と順序回路の等価性検証とに分けられる．

組合せ回路では，入力が確定すると出力は一意に決定される．そのため，組合せ回路の等価性検証を行う場合は，比較を行いたい回路同士の，同一の入力に対する出力の等価性を判定すればよい．これに対して順序回路の場合は，入力およびその内部状態によって出力が決定されるため，機能的には等価な回路同士であっても，同一の時刻に同一の入力を受け取り，同一の時刻に等価な値を出力するとは限らない．そのため，比較を行いたい回路同士がどの時点において等価であればよいのかといった等価性の定義を，検証者が明示的に与える必要がある [5].

図 2.3 に，組合せ回路の等価性検証の例を示す，まず比較を行いたい 2 つの回路の出力を XOR ゲートで接続する．次に，それを SAT 問題に変換し，得られた問題の充足可能性を判定する．充足可能であれば，その充足解は 2 つの回路が等価でないことを示す反例を表し，充足不可能であれば，2 つの回路が等価であることを示したことになる [3, 4].

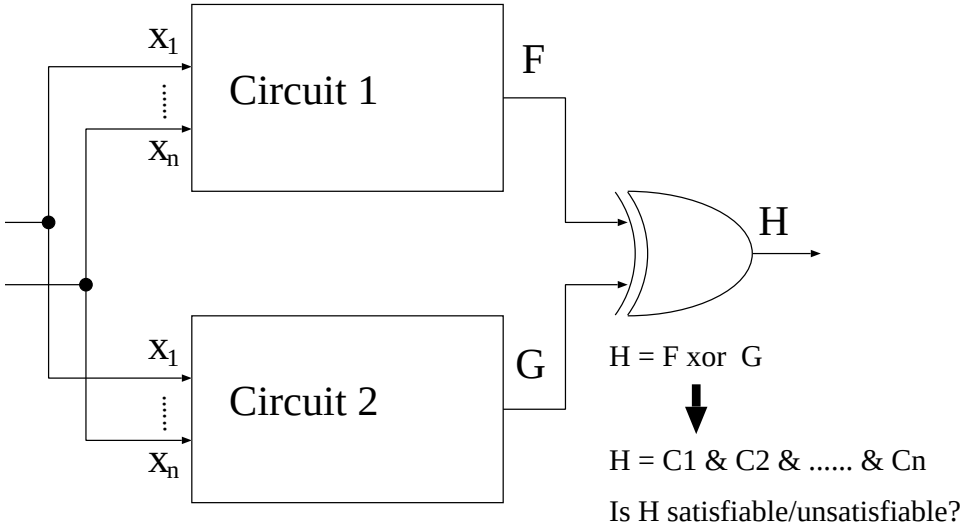

図 2.3: 組合せ回路の等価性検証

モデル検査

与えられた設計が特定の性質や仕様 (プロパティ) を常に満たすかどうかを検証することをモデル検査と呼ぶ．ある設計がプロパティ P を常に満足することを検

証する場合には，その設計を初期状態から到達可能な全ての状態へと展開し，プロパティ P の反例 $\neg P$ を満たす状態が存在するかどうかを検証する [3, 4].

モデル検査のうち，展開される状態の数が有限であるものは有界モデル検査と呼ばれる [3, 4, 5, 6]．図 2.4 に有界モデル検査の概略を示す．まず，与えられた回路を初期状態から有限回数 i だけ展開する．次に，展開されたそれぞれの状態において，プロパティ P の反例 $\neg P$ が満足されたことを検証する回路を接続し，その出力の論理和を取り，これを SAT 問題に変換する．回路が P を常に満たすのであれば，その反例 $\neg P$ が満足されるような状態は存在しないため，生成された SAT 問題が充足されることはない．一方，P を満足しないような状態が存在すれば，生成された SAT 問題は充足され，充足解はその回路が P を満足しない例を表す.

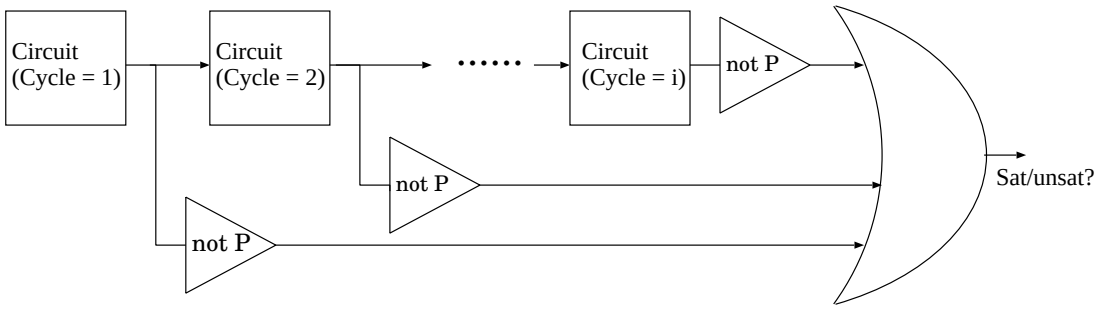

図 2.4: 有界モデル検査

2.3.2 その他の応用

本小節では，充足可能性問題のその他の応用例について簡潔に述べる.

テストパターン自動生成は，回路の故障を検出するための，回路への入力パターンを自動生成する手法である．考えられる入力パターンを全て生成することは計算時間の観点から現実的でないため，いくつかの代表的な故障をモデル化し，その故障モデルに相当する故障を検出するようなテストパターンを生成する．主な故障モデルとしては，(1) 回路中の信号線の値が 0 または 1 に固定される縮退故障，(2) 回路中の信号線同士が短絡する短絡故障，などがある [18]．[19] では，充足可能性問題を用いた縮退故障のテストパターンを生成する方法が提案されている．この方式では，故障の存在しない回路と縮退故障の存在する回路とを排他的論理和で接続したものを SAT 問題に変換し，その充足解を探索する．得られた充

足解のうち，回路の入力信号に該当する変数への真偽割当がテストパターンに相当する．

プランニングは，何らかの目標が存在する場合に，それを達成するためのアクションの選択，およびその実行順序を決定する問題である [7][8]．充足可能性問題を用いたプランニングでは，初期状態，実行可能な全てのアクション，目標をそれぞれ論理式で表現し，それらの論理積を SAT 問題に変換し，充足解の探索を行う [8][16]．得られた充足解は，初期状態から目標へ到達可能なアクションの系列を表す [8][16]．

スケジューリングは，目標を達成するためのアクションと資源，および目的関数 (例えば目的を達成するための総所要時間) が与えられたとき，目的関数が最適化されるような，各アクションに対する資源の割当てを求める問題である [7][8][11]．スケジューリングでは，あるアクション A_i が他のアクション A_j よりも前に完了しなればならないことを表す先行制約，共通の資源を必要とするアクション A_i および A_j を，同時に実行することはできないことを表す資源制約，アクションを開始することのできる時刻，および終了しなければならない時刻が，それぞれ制限条件として与えられる [11][16]．充足可能性問題を用いたスケジューリングでは，これらの制限条件と目的関数の最適値をそれぞれ論理式で表現し，それらの論理積を SAT 問題に変換し，充足解の探索を行う [11][16]．得られた充足解は，目的関数を最適化することができるような，アクションに対する資源の割当てを表す [11][16]．

FPGA Routing は，FPGA 上に実装される回路の配線を探索する問題である．回路のネットリスト，FPGA の配線構造や配線領域などの条件から SAT 問題を生成し，その充足解を求める．得られた充足解は，FPGA 上におけるその回路の配線を表す．同様の問題は Application Specific Integration Circuit(ASIC) にも存在するが，デバイスの内部構造を目的の処理に合わせて作り込むことができる ASIC と異なり，決められた物理構造の制約のもとで回路の配線を決定しなければならない FPGA においては，ASIC と同じ手法を用いることは適切ではなく [20]，FPGA Routing は ASIC Routing とは異なる種類の問題とされている [21]．

第3章　アルゴリズム

充足可能性問題を解くためのアルゴリズムは大きく2つのグループ，すなわちComplete 型と Incomplete 型に分類される．本章では，代表的な Complete 型アルゴリズムである Davis-Putnam-Logemann-Loveland(DPLL) アルゴリズム，および代表的な Incomplete 型アルゴリズムである Walksat アルゴリズムについて，それぞれ説明する．

3.1　Complete 型アルゴリズム

Complete 型のアルゴリズムは，充足可能であるかどうかを必ず判定することができる．Davis-Putnam-Logemann-Loveland(DPLL) アルゴリズム [23][14] は，その中で最も基本的なアルゴリズムのひとつであり，バックトラッキングによる全解探索に， Boolean Constraint Propagation(BCP) と呼ばれる探索空間削減手法を導入したものである．

procedure DPLL()
input: a CNF formula F, partial variable assignment T(initially empty)
begin
　v = decision() //select an unassigned variable and assign a value to it
　boolean_constraint_propagation()
　if **conflict** occurred then return **false**
　if T satisfies all clauses in F then return T
　DPLL$(F \cup \{(v)\}, T \cup \{v\})$
　DPLL$(F \cup \{(\neg v)\}, T \cup \{\neg v\})$
　return “Unsatisfiable.”
end

図 3.1: DPLL アルゴリズム

図 3.1 に，DPLL アルゴリズムの概略を示す．

まず，値の割り当てられていない変数をひとつ選択し，真偽値を割り当てる．この処理は Decision と呼ばれ，Decision により値の割り当てられた変数は Decision Variable と呼ばれる．

Decision により Unit Clause が発生した場合には Boolean Constraint Satisfaction(BCP) を実行する．Unit Clause とは，値の割り当てられていないリテラルの数がひとつだけであり，かつその他のリテラルの値が全て 0 であるようなクローズである．Unit Clause を充足させるためには，未だ値が割り当てられていない残りひとつのリテラル $l_{unassigned}$ に 1 を割り当てる．これにより，$l_{unassigned}$ を含む他の全てのクローズも充足される．Unit Clause を充足させるための，$l_{unassigned}$ に対する値の割り当ては Unit Propagation と呼ばれる．一方，$\neg l_{unassigned}$ の値は 0 となり，これにより，新たな Unit Clause が発生する場合がある．新たな Unit Clause が発生しなくなるまで Unit Propagation を繰り返し実行することにより，探索空間を大幅に削減することが可能となる．Unit Propagation の繰り返しにより変数へ真偽の割り当てを行う操作を BCP と呼ぶ．BCP により値の割り当てることのできる変数が存在しなくなっても，未だ値が割り当てられていない変数が残っている場合には，再度 Decision を行う．Decision と BCP を繰り返すことにより，全てのクローズが充足された場合，そのときの変数への真偽割当は充足解である．

ある Clause を充足させることにより，他の Clause が非充足となることを，Conflict が発生したという．Conflict が発生した場合は，直前の Decision Variable に対し，現在の真偽割当の論理否定を割り当て直す．既に真・偽双方の論理値を割り当てている場合には，更に前の Decision Variable まで遡って割り当てをやり直す．一番初めの Decision Variable まで遡って割り当てをやり直しても Conflict が解消されない場合は，その問題は非充足である．

今日提案されている代表的な Complete 型ソルバー [24][25][26][27] は，この DPLL アルゴリズムに基づいている．GRASP[24] では，Conflict が発生した場合に，その原因を解析する Conflict Driven Clause Learning(CDCL) と呼ばれる手法が導入されている．これにより，Conflict の原因となっている Decision Variable まで一気に遡って値の割り当てをやり直すことが可能となる．これを非辞書式バックトラッキング [3][14][24] と呼ぶ (これに対して，常に直前の Decision Variable まで戻って割り当てをやり直すことを辞書式バックトラッキングと呼ぶ)．また，CDCL の結果に基づき，Learnt Clause と呼ばれるクローズが生成され，もとの問題に追加される．Learnt Clause は，ある変数の組に対してどのような値の割り当てを行った場合に Conflict が発生するかを示すクローズであり，これを問題に追加すること

により，同一の Conflict を繰り返し発生させることを防ぐことができる [3][14][24].

非辞書式バックトラッキングと，Learnt Clause の追加による探索の効率化は，のちに開発された他の Complete 型ソルバー [25][26][27] に引き継がれ，今日の Complete 型ソルバーの基礎をなす技術となっている．

3.2 Incomplete 型アルゴリズム

Incomplete 型のアルゴリズムは，主にヒューリスティックなアルゴリズムであり，必ずしも解を見出すとは限らない．このため, 解が見出されなかった場合，その問題には解がないのか，単に発見できなかっただけなのかを区別することはできない．確率的局所探索アルゴリズム (Stochastic Local Search Algorithm: SLS) は，代表的な Incomplete 型アルゴリズムのひとつであり，問題が充足可能である場合に，非常に高速に充足解を発見することができる [28]．また，限られた時間の中で，より多くのクローズが真となるような変数の値を求める問題 (MAX-SAT 問題) を解くこともできる．代表的な SLS アルゴリズムである Walksat アルゴリズムについて述べる．

3.2.1 Walksat アルゴリズム

Walksat[29][30] は，SLS アルゴリズムのうち，最も高性能なアルゴリズムのひとつであり，複雑な制御を必要とせず，また処理の並列性が高いことから，ハードウェアを用いた実装に適している [55]．図 3.2 に，Walksat の処理手順の概略を示す．まず，(1) 乱数を用いて変数の初期値を生成する．そして，(2) 充足していないクローズをランダムにひとつ選び出し，(3) そのクローズに含まれる変数の中から，ヒューリスティクスを用いてひとつ変数を選択し，(4) その変数の値をフリップする (変数の値が 0 ならば 1 とし，1 ならば 0 とする)．(2) から (4) を繰り返すことにより解を探索する．図 3.2 において，*MAX-TRIES* は繰り返し回数の最大値 (一般に，1 回の試行を長く行うのではなく，乱数の初期値を変え複数回の試行が行われる)，*MAX-FLIPS* は 1 回の試行における変数のフリップ回数の最大値である．[30] では，変数を選択するためのヒューリスティクスが 6 つ挙げられている．ここではそれらのうち，次の 2 つのヒューリスティクスについて述べる．

Walksat/G: 確率 p で，フリップする変数を選択されたクローズ内からランダムに選ぶ．確率 $1-p$ でクローズ内の全ての変数のフリップを一時的に行い (以下，一時的フリップと呼ぶ)，その結果，充足されたクローズの数 (*score*

```
procedure Walksat
input a CNF formula F, MAX-FLIPS and MAX-TRIES
begin
for i in 1 to MAX-TRIES
    T = randomly generated truth assignment        (1)
    for j in 1 to MAX-FLIPS
      if T satisfies F then return T
      c = a random unsatisfied clause              (2)
      v = a variable in c chosen by a heuristic H  (3)
      T = T with v flipped                         (4)
    end for
end for
    return "no satisfying assignment found"
end
```

図 3.2: Walksat アルゴリズム

と呼ぶ) が最大となる変数を選択し，実際にその変数の値をフリップする．p はノイズパラメータであり，0 から 1 の間の値をとる．

Walksat/SKC: ある変数をフリップしたとき，値が 1 から 0 になる (非充足となる) クローズの数を *break-value* と呼ぶ．選ばれたクローズに含まれる変数全てについて，一時的フリップを行い，*break-value* を求める．*break-value* が 0 となるものがあれば，そのなかのいずれかをランダムに選択する．そうでない場合，確率 p でランダムに変数を選択し，確率 $1-p$ で，*break-value* が最小となるものを選択する．

Walksat アルゴリズムの欠点として，局所解からの脱出が困難であることが挙げられる．これを克服するため，新しい変数選択ヒューリスティクスの開発 [30][31][32][33] も含め，様々な改良手法が提案されている．

[31] では，Walksat アルゴリズムにおけるノイズパラメータの動的調整が提案されている．これは，変数のフリップにより非充足クローズの数が減少する場合にはノイズパラメータを減少させる一方，一定回数だけ変数のフリップを行っても非充足クローズの数が減少しない場合には局所解に陥ったとみなし，ノイズパラメータの値を増加させるものである．ノイズパラメータを減少させることにより，充足解への収束を速めることを企図するとともに，ノイズパラメータを増加

させることにより，ランダムに変数をフリップする確率を高くし，それにより局所解からの脱出が容易となることを企図している．

[34][35] では，Dynamic Local Search(DLS) に基づく Incomplete アルゴリズムが提案されている．DLS は，局所解や解の存在しない領域に陥ることを避けるために，局所探索法における評価関数自体を変形する，というアイディアに基づいて提案された手法である [34]．具体的には，まず与えられた CNF-SAT の各クローズに重み付けを行う．次に，フリップの候補となる各変数の一時的フリップを行い，フリップされる前とフリップされた後における，非充足クローズの重みの総和をそれぞれ計算し，その差分が最も少ない変数を優先的に選択する (フリップの候補となるのは，非充足クローズ内に出現する変数である [34][35])．フリップを行った後，充足されていないクローズの重みの増加を行う．

[40][41] では，SAT 問題の簡略化手法 [36][37][38][39][40] をプリプロセッサとして導入することにより，Incomplete 型アルゴリズムの性能を向上させることが可能であることが報告されている．

[42, 43] では，SLS アルゴリズムと CDCL 付き DPLL アルゴリズムを統合した，SATHYS アルゴリズムが提案されている．SATHYS アルゴリズムでは，まず SLS アルゴリズムを用いた解探索を行う．[31] と同様な手法で，一定回数変数のフリップを行っても充足されるクローズの数が増加しない場合には局所解に陥ったと判断され，DPLL アルゴリズムによる探索に切り替わる．それまでの変数への真偽割当は保留される．DPLL アルゴリズムは，Decision および BCP を 1 ステップ実行する (Conflict が発生した場合には CDCL を実行し，Learnt Clause を問題に追加するとともに，Decision をやり直す)．その後，保留されていた変数への真偽割当のうち，DPLL により値の割り当てられた変数への真偽割当を，DPLL により割り当てられた値に更新し，SLS による探索に戻る．このように，SATHYS アルゴリズムは，CDCL 付き DPLL を用いて SLS における局所解からの脱出を図ったアルゴリズムであると言える．

第4章　FPGAソルバー

FPGAを用いたSATソルバー(FPGAソルバー)においては，優れた探索能力を持ち，かつ並列性の高いアルゴリズムを，その並列性を有効に活用しながら実行することが重要である．実現可能な並列度は，問題規模により変化する．小規模な問題であれば，全てのデータをFPGA上に保持することにより，並列度を最大化することが可能である．一方，問題が大規模になるにつれ，より多くのデータを外部メモリ上に置かなければならなくなり，並列度は外部メモリのデータ転送速度により制限される．

本章では，まず，FPGAソルバーに向いたアルゴリズムについての考察を行う．次に，先行研究の紹介とそれらの分類を行い，先行研究においてどのような問題点が残されているかを明らかにする．最後に，筆者らの提案するFPGAソルバー(3-SAT問題用ソルバー1〜4，形式検証用大規模ソルバー)について，その概略を説明する．

4.1　FPGAソルバー向きアルゴリズム

既に述べたように，充足可能性問題を解くアルゴリズムはComplete型とIncomplete型に大別される．

代表的なComplete型アルゴリズムであるDPLLアルゴリズムはバックトラッキングによる全数探索アルゴリズムであるが，変数の数N_vに対して$O(2^{N_v})$の探索空間を持つ充足可能性問題においては，そのままでは高速な処理を実現することは難しい．一方，DPLLの高速化手法であるCDCLは複雑な制御を必要とするため，ハードウェアによる実装が困難である[55].

Incomplete型アルゴリズムのうちDLSアルゴリズムは，(1)クローズの重みの更新のため頻繁にテーブルを参照する必要がある，(2)重みの更新のための乗算器が多数必要となり，回路規模が大きくなる，(3)フリップの候補となる変数に対して，その変数を含む非充足クローズの重みの総和をそれぞれ計算し，その総和の順に変数をソートしなければならない，など，並列処理を阻害する要因が多い．

これらに対して，Walksat アルゴリズムは，複雑な制御回路が不要でありながら探索能力が高く，処理の並列度も高いことから，ハードウェアシステムに適したアルゴリズムであると言える．

4.2 先行研究

これまでに数多くの FPGA ソルバーが提案されている [55][63][64][65][66]．FPGA ソルバーは，(1) 問題毎に回路構成データを生成し，回路をその都度書き換える方式，(2) 回路の全面的な書き換えは行わず，ホストコンピュータ等によりデータを生成し，それを FPGA にダウンロードする方式 に大別される [55]．前者により，問題に合わせた最適な回路構成を取ることが可能となり，処理性能を最大化することができるが，回路構成データの生成に非常に長い時間を要するため，ソフトウェアに対して速度向上を実現することは難しい．後者は，前者と比較して非常に短い時間で問題データの変更を行うことができるが，回路上の各ユニットにデータのダウンロードのためのロジックと配線が必要となるため，前者と比較して回路規模は増大し，その分処理の性能 (並列度) は制限される．本節では，後者の方式のソルバーの中から代表的なものを取り上げ，それぞれ概略を述べる．

4.2.1 Complete 型ソルバー

[63] では，DPLL アルゴリズム [23] を Celoxica 社製 FPGA ボードである RC1000 に実装しており，同ボードに搭載されている 1 チップの XC2V2000E と 2MB の外部 SRAM 4 バンクを用いることにより，7168 変数/165036 クローズまでの問題を解くことができる．このシステムでは，与えられた問題をホストコンピュータ上で 3-SAT に変換し，それを FPGA 上の回路にマッピングする．問題が FPGA に収まり切らない場合には，複数の回路構成データに分けてマッピングを行い，それらを外部 SRAM バンクに格納する．そして，FPGA，外部 SRAM バンク，およびホストコンピュータ間の調停を行う CPLD を介して，回路構成データが適宜 FPGA 上にダウンロードされ，処理される．

本システムの動作周波数は 40MHz であり，Complete 型アルゴリズムである GRASP[24] を Pentium II 300MHz，メインメモリ 192MB 上で実行した場合との比較の結果，50 変数/100 クローズの 3-SAT 問題 [73] において 250 倍の速度向上率が達成されている．一方，より大規模な問題である 435 変数/1027 クローズの k-SAT 問題 [73] (561 変数/1405 クローズの 3-SAT 問題に変換される) における速

度向上率は0.12倍であり，ソフトウェアに対して速度向上を実現することはできていない．これより大規模な問題における処理の性能は報告されていない．

[64]では，GRASP[24]をXilinx社製FPGAであるXC2VP30と，総容量512MBのDDR SDRAMバンクを用いて実装しており，8K変数/31Kクローズまでの問題を解くことが可能である．与えられた問題は，FPGAで処理可能なサイズとなるように，ホストコンピュータ上で幾つかの小問題に分割され，DDR SDRAMバンクに格納される．各データはFPGA上の組み込みプロセッサを介してDDR SDRAMバンクから読み出され，FPGAのメモリ領域であるBlock RAMにキャッシュしながら処理される．より大規模なFPGAであるXC4VFX140を用いて10K変数/280Kクローズまでの問題を解くことが可能であると報告されている．

処理性能については，XC4VFX140を用いることによって，Complete型アルゴリズムであるMinisat[27]をPentium-4 3.2GHz，メインメモリ3GB上で実行した場合と比較して，約17倍の速度向上が見込まれると報告されているが，その根拠については詳しく言及されていない．

4.2.2 Incomplete型ソルバー

以下の議論において，N_vは与えられた論理式の持つ変数の数，N_cはクローズの数を表す．

[65]では，pure random Walksat アルゴリズム (Walksat/G において $p = 1.0$ としたもの．変数は常にランダムに選ばれる) を Xilinx 社製 FPGA XCV300 を用いて実装しており，50 変数/170 クローズまでの 3-SAT 問題を扱うことができる．動作周波数は33MHz，回路の性能は363.7K flip per second (fps) であり，Sun SPARCstation 20上で同一のアルゴリズムを用いて50変数/170クローズの3-SAT問題を解いた場合と比較して，最大3.3倍の速度向上率が実現されている．このハードウェアでは，50個の変数が，ROMとANDゲートから構成された170個のクローズ評価ユニットにブロードキャストされ，170個のクローズが並列に評価される．ブロードキャストされた50個の変数の値は，4個ずつに分割されて，ROM(1b(幅) ×16(深さ)) のアドレスとして用いられる．各ROMの出力のANDがとられ，クローズが充足されているかどうかが調べられる．このシステムでは，問題毎に，FPGAの回路構成データの一部をFPGAにダウンロードし，FPGA内のROMの部分を書き換えることにより，様々な問題を扱うことができるようになっている．回路構成データの生成に0.45sec，それをFPGAへダウンロードするのに1.46secを要する．この方式では，$N_c \times (N_v - 1)/3$ 個 (N_v が 4^n のとき) のLookup Table (LUT: FPGAの基本演算素子) を必要とするため，大規模な問題を

扱うことは難しい．

[66] では，pure random Walksat と Walksat/G を，Xilinx 社 FPGA XCV1000 上に実装しており，100 変数/220 クローズまでの 3-SAT 問題を解くことができる．動作周波数は 20MHz であり，pure random Walksat(パイプライン処理あり) で 20Mfps，Walksat/G (パイプライン処理は実現されていない) で 2.2Mfps の性能を実現している．Pentium 4 1.5GHz 上で同一のアルゴリズムを実行した場合との比較の結果，100 変数/200 クローズの 3-SAT 問題において，前者の回路で最大 24.93 倍，後者の回路で最大 3.06 倍の速度向上率が実現されている．このハードウェアでは，[63] と同様にクローズのデータをホストコンピュータ上で生成し，それを FPGA にダウンロードしている．クローズデータの生成と，FPGA ボード上の SRAM へのダウンロードに 7.6msec，そのデータを FPGA が読み込むのに 220×16 クロックサイクルを要する．クローズ評価モジュールの大きさは，[65] と同様に $N_c \times (N_v - 1)/3$ (N_v が 4^n のとき) に比例する．より大規模な FPGA である XC2V6000 を用いることにより 100 変数/600 クローズまでの 3-SAT 問題を解くことが可能であると報告されている．

いずれのシステムも，変数のフリップの際に全てのクローズを並列に評価するような回路構成をとっている一方で，各クローズの評価を行うクローズ評価ユニットの規模が大きいため，処理可能な問題規模は大変小さい．

4.2.3 先行研究例の分類

表 4.1: 先行研究例の分類

	(N_v, N_c)	FPGA	問題形式	アルゴリズム	性能
Tavares et al.[1]	(7168,165036)	XC2V2000E	k-SAT	DPLL	N/A
Gulati et al.[2]	(8K,31K)	XC2VP30	k-SAT	GRASP	N/A
	(10K,280K)	XC4VFX140	k-SAT	GRASP	N/A
Leong et al.[3]	(50,170)	XCV300	3-SAT	random Walksat	363.7Kfps
Yap et al.[3]	(100,220)	XCV1000	3-SAT	random Walksat	20Mfps
	(100,220)	XCV1000	3-SAT	Walksat/G	2.2Mfps
	(100,600)	XC2V6000	3-SAT	Walksat/G	N/A

[1] 2MB×4 の外部 SRAM バンクを使用.

[2] 合計 512MB の外部 DDR SDRAM バンクを使用.

[3] 外部メモリバンクは使用していない.

表 4.1 に，先行研究例の分類を示す．前 2 者は Complete 型アルゴリズムのソルバー，後 2 者は Incomplete 型のソルバーである．Complete 型のソルバーにお

いては，比較的大規模な k-SAT 問題を処理可能であるが，その性能については 435 変数/1027 クローズまでの問題における処理性能が報告されているのみであり，それより大規模な問題における処理性能は報告されていない．これに対して，Incomplete 型のソルバーにおいては，ソフトウェアに対して比較的高い速度向上率が実現されているが，処理可能な問題の形式は 3-SAT に限定されており，その問題規模も小さい．

以上のように，従来の FPGA ソルバーでは，数千変数/数万クローズを超えるような大規模な問題に対する高い処理性能は実現されていない．実応用問題においては，その規模は最大 10M 変数/100M クローズにのぼるため，処理可能な問題規模の更なる拡大が必要である．次節では，本論文において提案する，問題規模に応じた異なる 4 つの 3-SAT 問題用ソルバー (ソルバー 1〜4)，および，実応用問題のひとつである形式検証問題から生成された問題を対象とした大規模ソルバーについて，それぞれ概略を述べる．提案手法により，先行研究で扱える範囲の 3-SAT 問題に対して更なる高速化が実現される．また，先行研究例では扱うことのできなかった問題においても高い処理性能を実現することが可能となる．

4.3 ソルバー概要

4.3.1 ソルバーのハードウェア構成

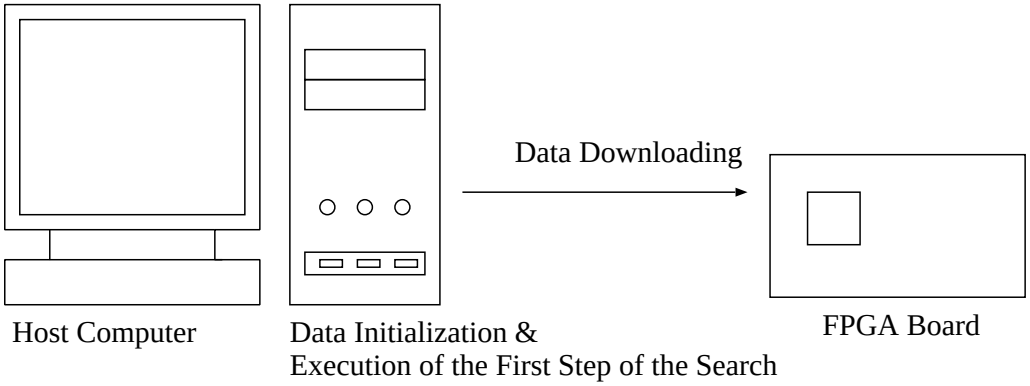

図 4.1: ソルバーのハードウェア構成

図 4.1 に本 FPGA ソルバーのハードウェア構成を示す．本ソルバーでは，回路のポータビリティを保つため，クローズデータ等の各データをホストコンピュー

タ上で生成し，それをFPGAにダウンロードする．また，3-SAT問題用ソルバー1を除く各ソルバーにおいては，解探索の最初のステップ(変数の初期値をランダムに生成し，それに対する全てのクローズの値を評価する)はホストコンピュータ上で実行される．評価結果は，クローズデータ等の各データとともにFPGAにダウンロードされる．

4.3.2 3-SAT問題用ソルバー1

本ソルバー[56, 57]の特徴は，(1)SAT問題におけるクローズの評価モジュールの小型化による高い並列性の実現，(2) 同一パイプライン上で，異なる4つの試行を並行して実行するマルチスレッド処理による高速化，(3)SATの各問題のハードウェアシステムへのダウンロード時間の短縮による高速化，にある．(1)，(2)により，より多くのクローズの並列処理による，より高速な処理が可能となるとともに，より大規模な問題を扱うことも可能となる．また，(3)により，処理時間の短い簡単な問題においても高速な処理を実現することが可能となる．

本ソルバーでは，Xilinx社製FPGAであるXC2V6000[49]を用いることにより，最大 256変数/2048クローズ，または 4096変数/1792クローズまでの問題が処理可能であり，個々の問題のホストコンピュータ上でのデータ生成，および，そのダウンロード時間も含めて，Walksat/G および Walksat/SKCにおいて，最大21Mfpsの処理速度を達成している．

4.3.3 3-SAT問題用ソルバー2

本ソルバー[58, 59, 62]の特徴は，(1) クローズの値を評価する際，変数のフリップにより値が変化する可能性のあるクローズのみを並列に評価することによる回路規模の小型化，(2) 複数の FIFO tree および Buffer を用いた非充足クローズの管理に基づくマルチスレッド処理による解探索の高速化(ソルバー1同様，4つの異なる試行を同時に行うことができる)にある．(1)により，処理の有効並列度を落とすことなく回路規模の小型化を行うことが可能であり，これにより，より大規模な問題を処理可能となるとともに，(2)により，より高速な処理が可能となる．2章2.2.2節で述べたように，多くの3-SAT問題は，解の探索をより困難なものとするため，変数の数N_vとクローズの数N_cの比がほぼ$1:4.3$となるように生成されている．この場合，Walksatアルゴリズムにおいて変数をフリップする際に評価しなければならないクローズの数は，平均で6，最大20未満となる．ソルバー2では，並列に評価することのできるクローズの数を20に限定することにより，処

理の有効並列度を維持したまま回路規模の小型化を行っている．このソルバーでは，XC2V6000 を用いて，2048 変数/8500 クローズまでの問題を処理することが可能であり，動作周波数は 85.2MHz，ホストコンピュータ上でのデータの生成およびそのダウンロード時間も含め，最大 28Mfps の処理速度を実現している．

4.3.4 3-SAT 問題用ソルバー 3

本ソルバー [60, 62] は，ソルバー 2 の回路方式における非充足クローズバッファ，およびクローズの評価を行う際に参照する Clause Table を，FPGA ボード上の外部メモリバンクにそれぞれ格納することにより，ソルバー 2 の回路方式において，処理可能な問題規模の拡大を図ったものである．また，外部メモリバンクのデータ転送速度の制約によるボトルネックを緩和するため，非充足クローズバッファと Clause Table の一部を，それぞれ FPGA 内のメモリ領域である Block RAM にキャッシュしている．ソルバー 2 と同様，変数をフリップする際に評価されるクローズの数は最大 20 であるが，外部メモリバンクのデータ転送速度の制約により，並列に評価することのできるクローズの数は，最大 8 に限定される．ソルバー 1, 2 と同様，マルチスレッド処理により異なる 4 つの試行を同時に実行可能であるが，外部メモリの参照による動作周波数の低下，およびクローズの評価に必要となるクロック数の増加により，処理速度は低下する．本ソルバーでは，XC2V6000 と，32 ビット ×1M の Synchronous SRAM(SSRAM) バンクを 6 つ搭載した Alpha Data 社製 FPGA ボードである ADM-XRC-II に，32 ビット ×2M の付加 SSRAM バンクを 2 つ追加したハードウェア構成 [47][48] を用いて，32K 変数/128K クローズまでの問題を処理することが可能である．動作周波数は 67.2MHz であり，ホストコンピュータ上でのデータの生成およびそのダウンロード時間も含めた処理速度は，最大 19.2Mfps である．

4.3.5 3-SAT 問題用ソルバー 4

本ソルバー [59, 62] は，ソルバー 3 と同様に非充足クローズバッファと Clause Table を外部メモリバンクにそれぞれ格納したものであるが，ソルバー 3 と異なり，Clause Table の Block RAM へのキャッシュは行っていない．ソルバー 2，3 と同様，変数をフリップする際に評価されるクローズの数は最大 20 であり，ADM-XRC-II を用いて 128K 変数/512K クローズまでの問題を処理可能である．一方，クローズ評価に必要となるクロック数の更なる増加と (並列に評価することのできるク

ローズ数は 5 となる)，マルチスレッド処理における実行可能なスレッド数が減少することから (スレッド数 2)，期待される性能はソルバー 2 のおよそ 1/4 となる．

4.3.6 形式検証用大規模ソルバー

本ソルバー [61] は，実応用問題のひとつである形式検証を対象としたソルバーである．Xilinx 社製 FPGA である XC4VLX160[50] と，データ幅 32 ビット，容量 4G ビットの DDR3 SDRAM バンクを 4 つ搭載したハードウェア構成 (データ幅 16 ビット，容量 2G ビットの DDR3 SDRAM を 8 個搭載した Alpha Data 社製 FPGA ボード ADM-XRC-6T1 の外部メモリバンク構成に相当する [52]) において，1M 変数/16M クローズまでの問題を処理可能である．

本ソルバーの検討にあたり，Walksat/SKC に対し，形式検証問題において局所最適解からの脱出が容易となるよう，以下のような改良を加えた．

- 与えられた問題のクローズが，どの論理ゲートを表しているのかを検出し，更にその論理ゲートの出力信号に相当するリテラルを特定する．Walksat アルゴリズムにおいて選択された非充足クローズ内のリテラルのうち，*break-value* が 0 となるものが存在しない場合には，そのクローズの対応する論理ゲートにおける出力信号に相当するリテラルを優先的にフリップする (優先度はノイズパラメータにより決定される)．
- (B) 非充足クローズの増減に応じてノイズパラメータの動的調整 [31] を行う．

上記のハードウェア構成において，DDR3 SDRAM バンクのスループットとアクセス遅延，および FPGA の動作周波数をそれぞれ変化させた場合の性能を，ソフトウェアシミュレータを用いて検証し，ソフトウェアに対してどの程度の高速化が可能であるかの考察を行った．

第5章　3-SAT問題用ソルバー 1

5.1　ハードウェアの特徴

本ハードウェアは次のような特徴を持つ．

1. クローズ評価モジュールが小さい．N_c にほぼ比例する程度の回路規模である．モジュールを構成するユニットは，22 個 (256 変数まで) または 25 個 (4096 変数まで) の LUT によって構成される．これは，[65][66] における $N_c \times (N_v - 1)/3$ と比べて大変小さい．

2. 回路は完全にパイプライン化されている．また，異なる複数の試行を同時に実行するマルチスレッド処理による高速化がなされている．

3. データのダウンロードが高速である．256 変数/1024 クローズの問題の場合，1024(クローズの定義，即ち各クローズに含まれている変数表)+384(各変数の初期値) クロックサイクルでダウンロードが終了する．このダウンロード時間は N_c にほぼ比例する．

4. 回路の書き換えなしで，2 つのヒューリスティクス (Walksat/G および Walksat/SKC) を実行することができる．Walksat/G よりも Walksat/SKC のほうが高性能であるが，Walksat/SKC では，変数をフリップする前と，フリップした後のクローズ値をそれぞれ保持する必要がある．そのため，従来のハードウェアシステム [65][66] では Walksat/SKC を実装することができなかった．本ハードウェアでは，pure random Walksat は実装していない．これは，pure random Walksat は，実装自体は容易であるが，探索能力が低く，大規模な問題を解くことができないためである．

5.2　処理の概略

本ハードウェアのブロック図を図 5.1 に示す．図において，まず，選択されたクローズ内の 3 つの変数番号が Clause Table から順番に読み出される．これらの

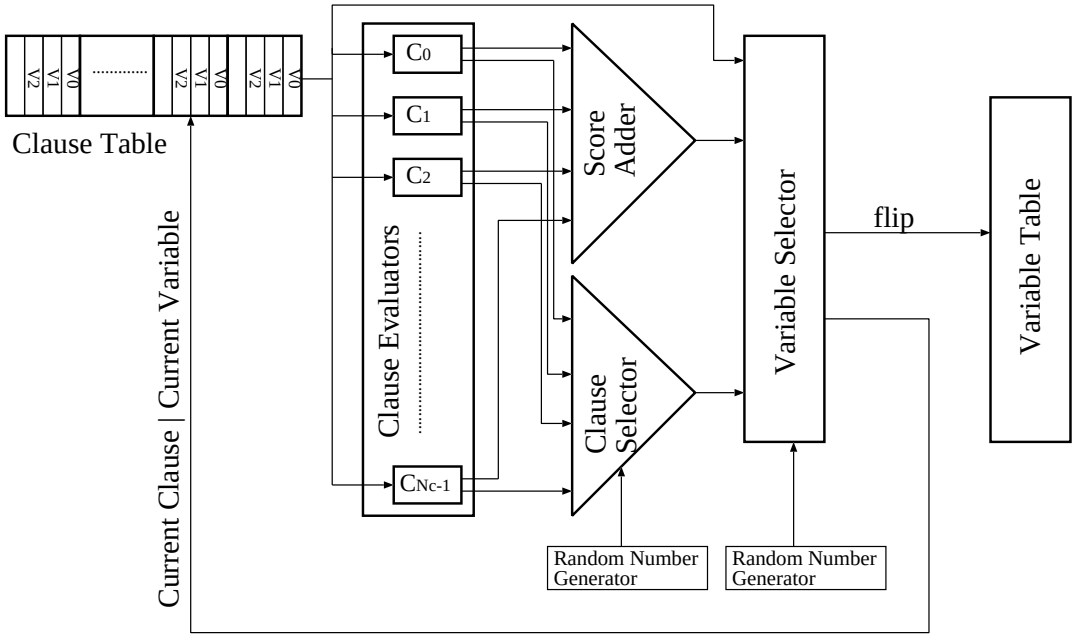

図 5.1: 回路のブロック図 (3-SAT ソルバー 1)

変数は，Clause Evaluators (クローズ評価モジュール) にブロードキャストされ，全てのクローズが並列に評価される．Clause Table の値は，各問題の実行時に，各問題に合わせて初期化される．本ハードウェアでは，変数番号のみがブロードキャストされ，クローズ内のリテラルが否定であるかどうかの情報はリテラルの正負はブロードキャストされない．これは，変数番号が Clause Evaluator 内に保持されたリテラル値をフリップするためだけに用いられるためである．各 Clause Evaluator は，送られた変数番号と，保持している変数番号を比較し，等しければ保持しているリテラル値をフリップする (一時的フリップ)．そして，その値を用いて *score* (または *break-value*) が Score Adder によって計算される．同時に，充足されていないクローズが， Clause Selector によって，ひとつ選択される．このようにして，3 つの変数それぞれに対する 変数番号，*score* と充足されていないクローズ番号が Variable Selector 内に保持される．Variable Selector は，これら 3 つの *score* を比較し，アルゴリズムに従って一組の { 変数番号，充足されていないクローズ番号 } を選択し，残りの 2 つを捨てる．そして，その選択された変数番号に対して，各 Clause Evaluator 内のリテラル値の変更が実際に行われ，また，Variable Table 内の変数の値がフリップされる (このテーブルの値は，最終的に解を出力する時にのみ用いられる)．選択されたクローズに対して，以上の処理を繰り返すことにより，全てのクローズを充足可能な変数の値を探索する．

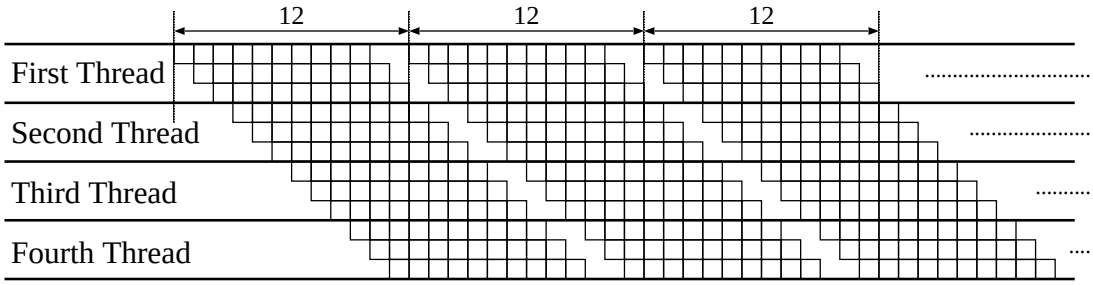

図 5.2: マルチスレッド処理 (3-SAT ソルバー 1)

5.3 並列/パイプライン処理の詳細

本ハードウェアは10段のパイプラインにより構成されている．既に述べたように，1回の変数のフリップに際し，3回の一時的フリップが必要となる．この3回の一時的フリップのそれぞれに10ステージ (10クロック) の処理が必要であり，それらが同一のパイプライン上で処理されるため，1回のフリップに必要なクロック数は12クロックとなる．従って，あるひとつの試行において，変数のフリップは12クロックに1回しか実行できないことになる (あるフリップの計算結果が，パイプラインから出力されるのは12クロック後であるため，それまで，次のフリップを行うことができない)．このことは，パイプライン上において，9クロックサイクル分のアイドル時間があることを意味する (パイプラインのうち3段分 (3クロック分) は1回のフリップで占有されるため)．この9クロックサイクルを用いると，同一パイプライン上で，更に3つの変数のフリップを行うことができる．この9クロックサイクルを活用するために，4つの独立した試行 (図3.2に示したように，解の探索は，変数の初期値を最大で *MAX-TRIES* 回変えて繰り返される) をパイプライン上で連続して行う (マルチスレッド処理 [56, 57])．図5.2は，本マルチスレッド処理の流れを示したものである．

Walksat アルゴリズムでは，ランダムに与えられる変数の初期値，乱数を用いた変数の選択により，解を得るまでのフリップ回数が大きく変わる．乱数によっては，非常に少ない回数で解を得ることができることもあれば，非常に大きな回数を要する場合もある．マルチスレッド処理により，N 回の試行を同一パイプライン上で並行して行い，N 回の試行のうち最も少ないフリップ回数で解に到達した試行の解を採択することで，解を得るまでのフリップ回数の期待値をより小さくすることができる．解が見つかった時点で (フリップ回数 $= K$ とする)，他の試行は全て打ち切られる．このときまでに，パイプライン上では，$K \times N$ 回のフ

表 5.1: マルチスレッド処理の性能 (3-SAT ソルバー 1)

Number of Threads	uf225-960		BMSk3n100	
	flip 回数	speedup	flip 回数	speedup
1	78882	-	73473	-
2	40891	1.93	36332	2.02
4	20162	3.91	18183	4.04
8	10305	7.65	9222	7.97
16	4974	15.85	4706	15.61

リップが行われたことになる．複数の回路を用いた並列処理でも同様の効果を期待することができるが，マルチスレッド処理では，ハードウェアの追加を殆ど必要としない．

マルチスレッド処理の効果はシミュレーションによって調べることができる．表 5.1 は異なるスレッド数における解を得るまでのフリップ回数の比較を示したものである (表 5.1 の値は 50 回実行した時の平均値である)．表 5.1 に示した値は，最も速く解に到達した試行のフリップ回数である (従って，パイプライン上では，この値のスレッド数倍のフリップが実行されたことになる)．アルゴリズムは，Walksat/SKC を用いている．評価には，ベンチマークプログラムである uf225-960(225 変数/960 クローズ，計 100 題)，および BMSk3n100 (100 変数/クローズ数は様々，計 500 題) を用いた [73]．表 5.1 からわかるように，マルチスレッド処理を用いて互いに無関係な N 回の試行を並行して行うことにより，ほぼ N 倍の速度向上率が得られている．

5.4 回路の詳細

5.4.1 クローズ評価ユニット

図 5.3 にクローズ評価ユニット (Clause Evaluator) の詳細を示す．ひとつの Clause Evaluator は，2 個の Variable Number Array と (256 変数の場合)，1 個の Literal Value Table からなる 3 対により構成されている．Variable Number Array と Literal Value Table は，それぞれ distributed RAM (FPGA 内の小規模メモリであり，大きさは $1b$(幅) $\times 16$ (深さ)) により構成されている．Variable Number Array と Literal Value Table の値は，ホストコンピュータから問題のデータがダ

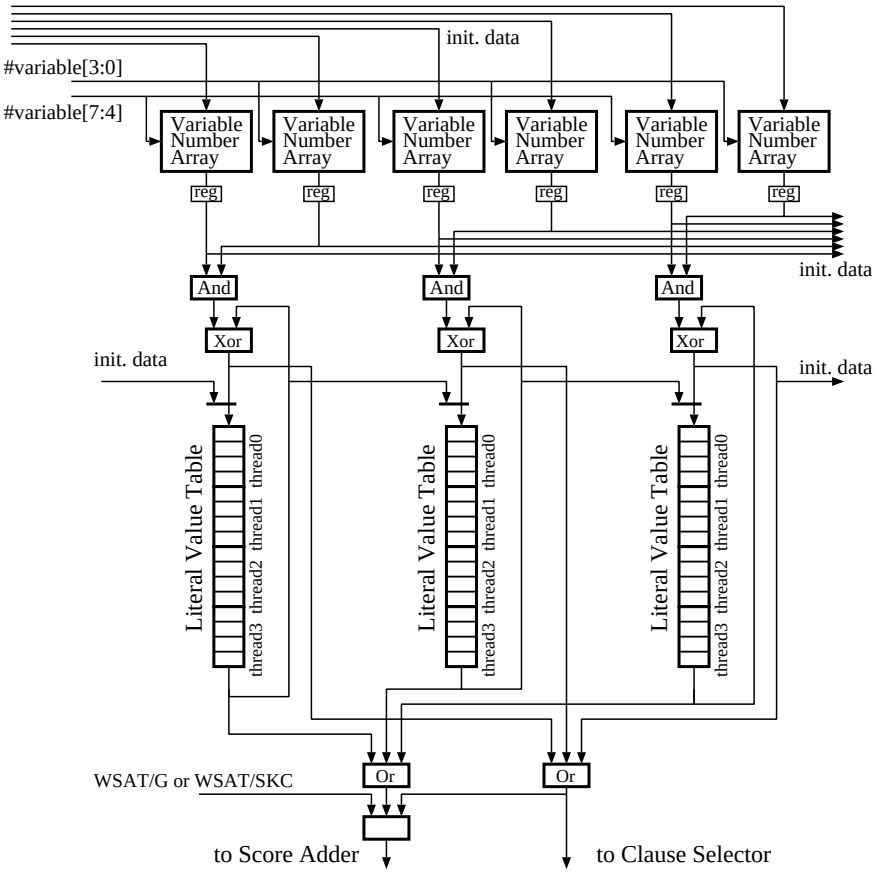

図 5.3: Clause Evaluator の詳細 (3-SAT ソルバー 1)

ウンロードされる際に初期化される．Clause Table から送られてきた変数番号は 4 ビットずつに分けられ，Variable Number Array のアドレスとして使われる．

図 5.3 において，変数番号が Clause Evaluator に送られてくると，3(各クローズの持つリテラルの数)× 2 (ひとつのリテラルに対する Variable Number Array の数．ひとつの Variable Number Array は 4 ビット入力であり，変数の最大数が 256 の場合は 8 ビット必要であるため 2 個となる) 個の Variable Number Array が読み出され，その中に送られてきた変数番号が含まれているかが調べられる．リテラル 1 個あたりに Variable Number Array をもうひとつ付け加えることで (1 クローズあたり 3 個となる)，4096 変数 (2^{12}) まで扱えるようになる．Literal Value Table は，dual port distributed RAM (2 つの LUT からなり，dual port access が可能) から構成されている．2 つの Variable Number Array の出力の AND が true となった時，Clause Evaluator に送られてきた変数番号と，2 つの Variable Number Array

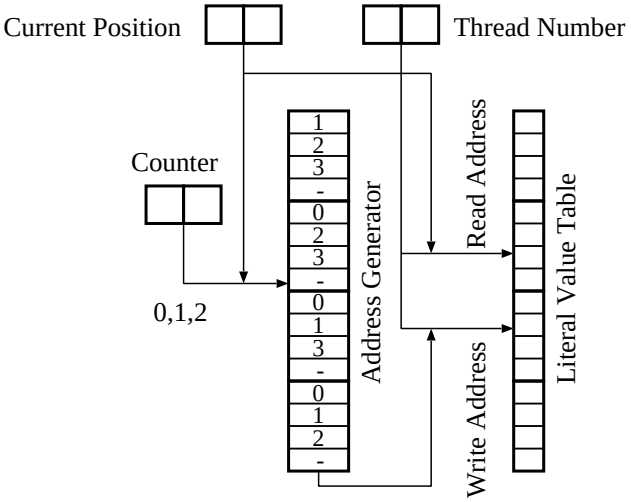

図 5.4: Current Position の管理手法

に保持されていた変数番号が等しいことを意味する．このとき，対応する Literal Value Table 内の値がフリップされる (一時的フリップ)．Literal Value Table の 16 ビットのうち，4 ビットがひとつのスレッドに用いられている．4 ビットのうちの 1 ビットにリテラルの現在値が保持され，残りの 3 ビットに，Clause Table よりブロードキャストされて来た 3 つの変数に応じて更新された新たなリテラル値 (一時的フリップにより生成された値) が保持される．最終的に，それら 3 つの値のいずれかが選択され，新しいリテラル値として用いられることになる．

図 5.4 に，リテラルの現在値を格納する場所 (Current Position) の管理方法を示す．Current Position は 2 ビットのレジスタにより保持され，Counter によって生成される 2 ビットの値 (0,1,2) と組み合わされた後 (Current Position が上位 2 ビット，Counter の値が下位 2 ビットとなる)，Address Generator をアクセスするのに用いられる．Address Generator の出力は，Current Position を避けた，メモリ上の残りの 3 個所となっている．Thread Number と組み合わされた Current Position は (Thread Number が上位となる)，Literal Value Table 中のリテラルの現在値を読み出すのに用いられ，Address Generator の出力と組み合わされた Thread Number は (やはり Thread Number が上位となる)，各クローズ中の 3 つの新しいリテラル値を Literal Value Table に格納するために用いられる．これにより，最終的にフリップする変数を選択する際に，Current Position の値を変更するだけで，リテラル値の更新を行うことができる．

表 5.2: Unfair Clause Selection によるフリップ回数

	uf225-960		BMSk3n100	
algorithm	fair	unfair	fair	unfair
Walksat/SKC	20051	20162	18270	18183
Walksat/G	161334	166013	50995	49977

5.4.2 評価値の計算とクローズの選択

図 5.3 において，3 つの Literal Value Table 中の各リテラルの現在値 (計 3 個)，およびそのリテラル値の一時的フリップにより生成された新たなリテラル値 (計 3 個) の OR がそれぞれ計算され，これらの値を用いて，Score Adder により，*score* (または *break-value*) が計算される．Score Adder は単純な木構造を持つ加算器であり，$N_c = 1024$ の場合，2 ビットの加算器 ×512 個，3 ビット加算器 256 個，4 ビット加算器 128 個等から構成されている．

Clause Selector は，充足されていないクローズをランダムにひとつ選択する．ここでの問題は，全ての充足されていないクローズが等しい確率で選ばれるようにすることがパイプライン化されたハードウェア上では大変難しいことである．本ハードウェアでは，2 分木状に接続されたセレクタを用いてクローズを選択しているため各クローズが選択される確率が均等ではない．例えば，$N_c = 1024$ のとき，0-511 番目のクローズのうち n 個，512-1023 番目のクローズのうち m 個が充足されていないと仮定すると，たとえ $n \ll m$ であっても，n 個のクローズの中から選択される確率は 1/2，m 個のクローズの中から選択される確率は 1/2 となる．表 5.2 は，fair selection (乱数により全ての候補が等しい確率で選択される) を用いた場合と，unfair selection (本論文で用いた手法) を用いた場合に，解を得るのに要するフリップ回数をシミュレーションにより比較したものである (マルチスレッド処理を用いている)．表 5.2 の値は，ベンチマークプログラム uf225-960, BMSk3n100 の全ての問題をそれぞれ 50 回ずつ実行した場合の平均値である．表 5.2 から，本研究で用いた unfair selection でも，探索性能には，ほとんど差異がないことがわかる．

5.4.3 問題データのダウンロード時間

各問題のデータのダウンロードに要する時間は，問題が小規模であったり，簡単に解ける問題の場合には重要である．これは，これらの時間を，ハードウェアシステムによる実行時間の一部として考慮する必要があるからである．本ハードウェアシステムでは，新たな問題に対して書き換える必要があるのは以下の 4 箇所である．

1. Clause Table 1 個
2. Variable Number Array $N_c \times 3 \times 2$ 個
3. Literal Value Table $N_c \times 3$ 個
4. Variable Table 1 個

Clause Table, Variable Table および Literal Value Table の値は，ホストコンピュータ上で生成され，ダウンロードされる．Variable Number Array の値は，ダウンロード時間を短縮するために，Clause Table のデータを用いて，FPGA 上で生成される．

Clause Table のサイズは，$N_c \times 3 \times logN_v$ ビット (N_c が 1024，N_v が 256 の場合は 1024×3 バイト) であり，Variable Table の サイズは，$N_v \times 4$ (スレッド数) ビット (N_v=256 のときは 1024 ビット) である．この 2 種類のデータは，同時にダウンロードされる (ホストコンピュータとのデータ転送幅は 32b であるため，空いている 8b が Variable Table のために用いられる)．従って，これらのテーブルの初期化に必要なデータ転送量は，1024 ワードとなる．

Literal Value Table の値は，Variable Table の値に応じて決まる．ダウンロード時間を最小にするために，Literal Value Table は 32 個のグループに分けられ，グループ内の Literal Value Table はシフトレジスタのように接続される．N_c =1024 の場合，$N_c \times 3 \times 4/32$ ワード，即ち 384 ワードのデータがダウンロードされる．

ホストコンピュータから Variable Number Array にデータをダウンロードする場合，$N_c \times 3 \times 2 \times 16$ ビット ($N_c = 1024$ のとき，3072 ワード) のデータのダウンロードが必要となる．そこで，本研究では，Variable Number Array へのデータを Clause Table のデータから生成するユニットを構築した．図 5.5 にユニットの概要を示す．ひとつのクローズのデータ (変数 3 個) がダウンロードされると，Plane-A 上の single port distributed RAM (FPGA 内の小規模メモリ，1b(幅) ×16(深さ)) 6 個 (3 変数 × 各変数に 2 個) のそれぞれ 1 ビットが，変数番号のうちの 4 ビットに応じてセットされる．この操作を 32 回繰り返すことにより，192 個の distributed

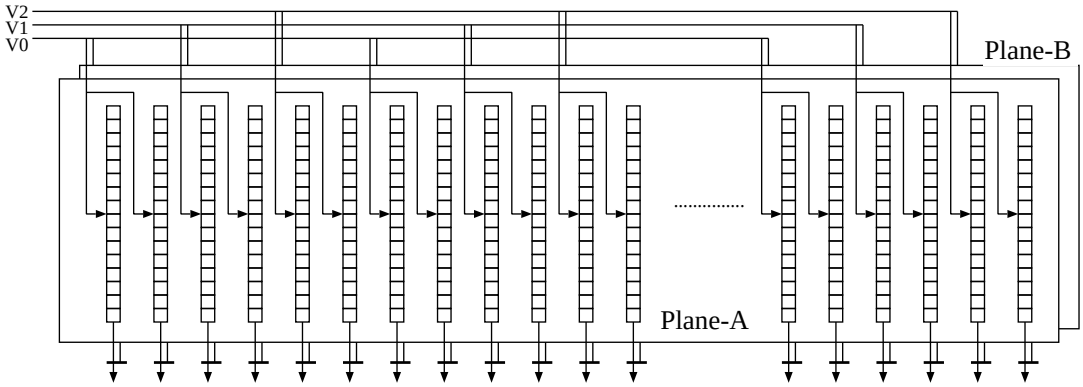

図 5.5: Variable Number Arrays 初期化ユニット

RAM の値がセットされる．そして，これらの値は並列に読み出され，Variable Number Array に送られる．Variable Number Array は 192 のグループに分けられ，シフトレジスタのように接続されている．Variable Number Array は single port distributed RAM から構成されているので，16 ビットのデータをシフトするのに 32 クロックサイクルを要する．これと並行して，Plane-B 上の 192 個の distributed RAM がセットされる．そして，Plane-A と Plane-B の役割が入れ替えられる．このユニットを用いて，本ハードウェアでは，Clause Table にデータをダウンロードしながら，Variable Number Array を同時に初期化することができる．初期化には $1024 + 32$ クロックサイクルを要する．

従って，N_c =1024 のとき，Clause Table のダウンロード時間と Literal Value Table のダウンロード時間の和である 1024+384 クロックサイクルで，全てのデータの初期化を行うことができる (他のデータの初期化は，このとき間内に，これらのダウンロードと並行して行うことができる).

5.5 評価結果

本研究では，XC2V6000 を 1 チップ搭載した Alpha Data 社製 FPGA ボード ADM-XRC II を用いて 256 変数/1024 クローズの問題の処理が可能なハードウェアを実装した．回路の動作周波数は 66MHz，回路使用量は 45% であり，Block RAM 144 個のうちの 5 つを使用した．XC2V6000 1 チップにより，256 変数/2048 クローズ (または 4096 変数/1792 クローズ) までの回路を実装することが可能である．今回，256 変数/1024 クローズまでの回路を用いて評価を行ったのは，256 変数/2048 クローズ (または 4096 変数/1792 クローズ) 程度の一般的なベンチマーク

表 5.3: 性能比較 (3-SAT ソルバー 1)

Instance	Software		Our FPGA Solver						
			without multi-thread			with multi-thread			
	#flips	Kfps	#flips	Kfps	Speedup	#flips	total	Kfps	Speedup
uf225-087	1691	780	1936	331	0.37	814	3256	576	0.38
uf225-026	2231	669	2024	345	0.57	842	3368	596	0.59
uf225-028	10922	780	11723	1536	1.83	3801	15204	2456	2.26
uf225-091	16347	761	14962	1820	2.62	5010	20040	3126	3.35
uf225-032	1027208	776	1330132	5378	5.35	280742	1122968	19860	23.4
uf225-039	2726196	770	2690011	5439	7.16	600537	2402148	20945	30.9

プログラムが無かったためである [73].

表5.3に，AMD Athlon MP2200+ 上でのソフトウェア (WalkSAT Version 35[72]) の実行結果と本ハードウェアの実行結果 (フリップ回数 (#flip)，平均フリップレート (Kfps = Kflips/second)) および，その処理速度の比較を示す．マルチスレッド処理 (スレッド数は 4) におけるフリップ回数は，4 回の試行のうち最も速く解に到達した試行のものである (パイプライン上では，この 4 倍 (スレッド数) の回数のフリップ (total) が行われており，平均フリップレートの算出には，その値を用いた)．速度向上率は，実行時間より算出されており，フリップ回数 (マルチスレッドの場合は total) を平均フリップレートで割った値が実行時間となる．ベンチマークプログラムは uf225-960(225 変数/960 クローズ) を使用し，アルゴリズムは Walksat/G よりも高性能である Walksat/SKC を用いた．また，図 3.2 に示した Walksat アルゴリズムにおいて，MAX-FLIPS には十分大きな値 (2^{28}) を固定値として与えた．これにより，今回用いたベンチマークプログラムにおいては，全ての初期値に対して，MAX-FLIPS 以内に解を得ることができる．このため，再試行が行われることはなく，MAX-TRIES は 1 に固定されている (マルチスレッド処理が行われているため，実際にはスレッド数分の TRY が行われる)．

本ハードウェアにダウンロードするデータを AMD Athlon MP2200+ 上で生成するのに約 2.7msec，FPGA 上にダウンロードするのに約 2.8msec を要する．PCI バスのデータ幅が 32 ビット，動作周波数が 66MHz であることから，DMA 転送のための準備時間がダウンロード時間の大部分を占めていると考えられる (この問題はドライバソフトの改良によって改善されると考えられる)．

表5.3に示したように，本ハードウェアは，uf225-032 や uf225-039 のような，解を得るまでに多くのフリップ回数を必要とするような問題においては，ソフトウェアを大きく上回る性能を示す．しかし，フリップ回数が少ない問題 (簡単な問題) を

解く場合には，大きな速度向上率を得ることは難しい．これは，ホストコンピュータ上でのダウンロードデータの生成とダウンロードに要する時間の全実行時間に占める割合が大きくなること，また，マルチスレッド処理による性能向上の効果が薄れることによるものである．簡単な問題に対する平均フリップレートは，[66] で報告されている結果よりも低くなっている．理由は次のとおりである．本ハードウェアでは，マルチスレッド処理と Walksat/SKC を用いているため，解を見つけるのに要するフリップ回数が [66] と比較して非常に少ない ([66] では，回路の制約上，フリップ前の各クローズの値が必要となる Walksat/SKC を実現することができない)．このため，簡単な問題に対しては，ダウンロードデータの生成とダウンロードに要する時間 (本ハードウェアでは [66] よりも若干少ない) が，実行時間全体の大部分を占める．従って，より少ないフリップ回数で解を見出すほど，言い換えればより高い探索性能を示すほど，平均フリップレートは低くなる．

今回の実装では，256 変数/1024 クローズまでの回路をひとつ FPGA 上に実装し，評価を行ったが，FPGA の書き換え可能性を利用することにより，問題の規模に合わせて，実装する回路の数を変更し，より高速化をはかることができる．例えば，今回評価に用いた FPGA の場合，256 変数/1024 クローズの回路であれば，2 個同時に実装することができる (回路使用率が回路ひとつ当たり 45% であるため)．この場合，スレッド数は合計 8(1 回路あたり 4) となり，表 5.1 に示したように，1 回路の場合の約 2 倍の高速化を期待することができる．

5.6 有効並列度に関する考察

本ハードウェアでは，変数をフリップする際，全てのクローズが並列に評価される．これは，ハードウェアの制御 (特にパイプライン処理の制御等) を容易にするためである．しかし，ソフトウェアでは，フリップする変数を含むクローズだけが評価され，全てのクローズが評価される訳ではない．ゆえに，処理の有効並列度は，クローズの総数ではなく，フリップする変数を含んでいるクローズの数によって決定される．例えば，変数がフリップされるときに評価されるクローズの数は，uf225-039 の場合，平均で約 12 個である．そのため本ハードウェアにおいては，非常に高い並列性が実現されていながら，ソフトウェアに対してそれほど高い速度向上率を得られていない．

SAT 問題における処理の有効並列度には，クローズの総数ではなく，フリップされる変数を含んでいるクローズの数が関係している．そのようなクローズだけを評価するハードウェアを構築することにより，限られた FPGA の回路量でより大規模な問題を処理することが可能となる．

第6章　3-SAT問題用ソルバー 2

6.1　ハードウェアの特徴

本ソルバーでは，[30] で挙げられているヒューリスティクスのうち，Walksat/SKC のみを実装している．これは，Walksat/SKC が，Walksat/G や pure random Walksat と比べて高い探索能力を持つためである．本ソルバーの特徴は次のとおりである．

1. 回路規模の小型化のため，変数をフリップする際，全てのクローズ値を評価するのではなく，フリップ操作によって非充足となる可能性のあるクローズのみを並列に評価する．Walksat/SKC における *break-value* を計算するためには，変数をフリップした際，値が真から偽へと変わるクローズの数のみを数えればよい．例えば，変数 v_k の値が, フリップにより 0 になる場合，リテラル v_k を含むクローズのみを評価すればよい (値が 1 から 0 に変化する可能性のあるクローズは，リテラル v_k を含むクローズのみであるため)．同様に，フリップにより， v_k の値が 1 になる場合，リテラル $\neg\, v_k$ を含むクローズのみを評価すればよい．このように，変数をフリップする際，全てのクローズの値を評価するのではなく，フリップにより非充足となる可能性のあるクローズのみを評価することにより，処理の有効並列度を落とすことなく回路規模の小型化を図ることができる．今回評価に用いたベンチマーク問題では，フリップにより値が変化するクローズの数は，高々20 未満 (平均約 6) であった．これは，これらの問題において $N_v : N_c \fallingdotseq 1 : 4.3$ であることによる (既に述べたように，[8][9] によれば，このとき 問題が最も困難なものとなる)．

2. 回路構成の簡単化のため，解探索の最初のステップ (変数の初期値をランダムに生成し，それに対する全てのクローズの値を評価する) はホストプロセッサ上で実行される．評価結果は，与えられた問題のクローズデータとともに回路にダウンロードされる (ソルバー 1 ではクローズの評価は行われない)．

3. 回路は完全にパイプライン化されている．また，より高い探索性能を得るため，マルチスレッド処理 [56, 57, 58, 59, 62] による高速化が行われている．

6.2 処理の概略

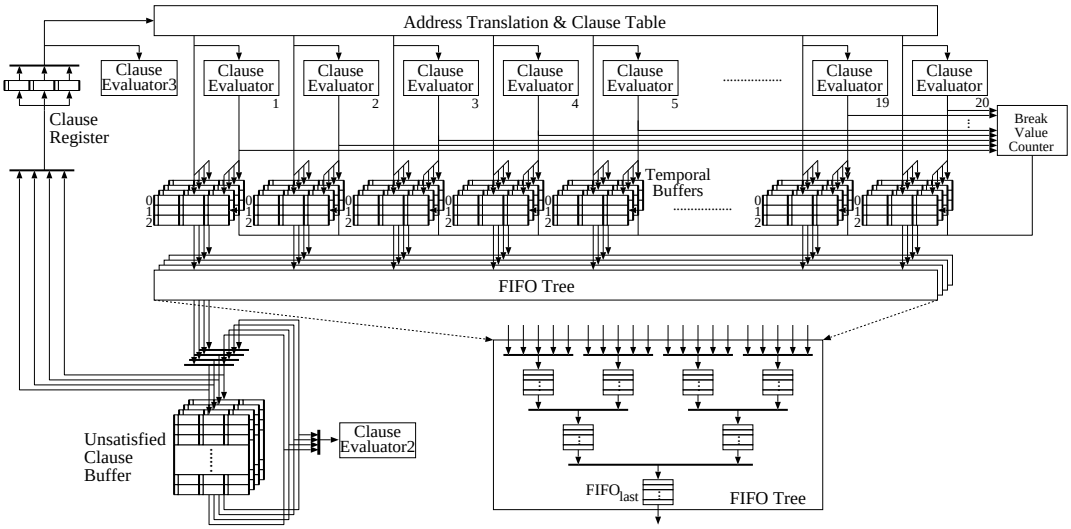

図 6.1: 回路のブロック図 (3-SAT ソルバー 2)

本ハードウェアにおける処理の概略を以下に示す．

1. まず，図 6.1 において，Unsatisfied Clause Buffer からクローズ C_k がランダムにひとつ読み出され，Clause Register にセットされる．

2. 次に，C_k 内のリテラル $L_i (i = 1, 2, 3)$ に対する *break-value* を以下の手順でそれぞれ計算する．

   - リテラル L_i が，Address Translation Table および Clause Table へ順に送られ，リテラル $\neg L_i$ を含むクローズのデータと *valid bit* が並列に出力される (簡略化のため，図 6.1 では *valid bit* を省略している)．
   - 各 Clause Evaluator は，これらのクローズが，L_i の一時的フリップにより非充足となるかを並列に評価する．L_i の一時的フリップによって $L_i = 1$ となるため (一時的フリップを行う前は，$C_k = 0$ であるため，$L_i = 0$ (i=1,2,3) である)，Clause Evaluator は与えられたクローズを

$\neg L_i = 0$ として評価する．各 Clause Evaluator の出力 (*brk*) は 1 ビットの値であり，Clause Evaluator に与えられたクローズが，リテラルの一時的フリップによって非充足となったかどうかを表す (*brk*=1 ならば非充足，*brk*=0 ならば充足)．

- *brk*=1 となったクローズデータは Temporal Buffer に送られる．各 Temporal Buffer には，各リテラル $L_i(i = 1, 2, 3)$ にそれぞれ対応する非充足クローズデータが格納される．
- Break-Value Counter は，$L_i(i = 1, 2, 3)$ に対する *break-value* (値が 1 の *brk* の数) を計算する．*break-value* を用いて，Walksat/SKC アルゴリズムに従い，L_i (i=1,2,3) の中からひとつを選択する．

3. 選ばれたリテラル L_m が実際にフリップされ，リテラル L_m に対応する非充足クローズデータが，各 Temporal Buffer から *FIFO tree* に送られる．

4. *FIFO tree* はこれらの非充足クローズを集め，Unsatisfied Clause Buffer に格納する．各 *FIFO tree* を構成する各 FIFO は，入力として与えられたクローズデータをランダムな順序で格納する．

本回路においては，探索の最中に Unsatisfied Clause Buffer に格納されているクローズが充足されることがある．これは，他の非充足クローズを充足させるための変数のフリップにより, Unsatisfied Clause Buffer 中のクローズも充足することがあるためである．そのため，Unsatisfied Clause Buffer からクローズを選択する際，(1) Unsatisfied Clause Buffer からランダムにクローズを選択し，Clause Evaluator 2 を用いて評価を行い，(2) もし充足していたならば，*FIFO tree* 内の *FIFO last* の出力を代わりに用いる．(3) もし *FIFO last* の出力がない場合には，非充足クローズの選択は行なわれず，回路はアイドル状態となる．筆者らの評価によると，全非充足クローズのうち 20〜25%が，非充足クローズ内に格納されている間に充足される．また, 選択されたクローズ中の各リテラルの一時的フリップを行うと同時に Clause Evaluator 3 を用いてそのクローズの評価を行い，そのクローズが本当に非充足であるかどうかを確認する．充足されていた場合には，回路はアイドル状態となる (筆者らが評価に用いたベンチマークでは，全実行時間の 3〜5%がアイドル時間となった)．

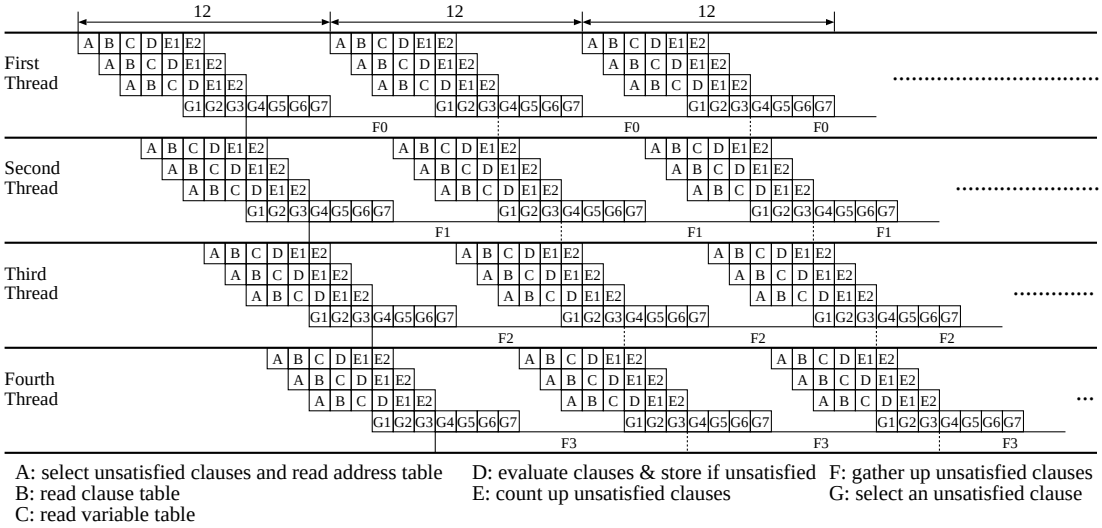

図 6.2: マルチスレッド処理 (3-SAT ソルバー 2)

6.3 並列/パイプライン処理の詳細

本ハードウェアは 12 段のパイプラインにより構成されているため，あるひとつの試行において，変数のフリップは 12 クロックに 1 回しか実行できない (あるフリップの計算結果が，パイプラインから出力されるのは 12 クロック後であり，それまで，次のフリップを行うことができない)．既に述べたように，1 回の変数のフリップに際し 3 回の一時的フリップが必要である．このことは，パイプライン上において，9 クロックサイクル分のアイドル時間があることを意味する．この 9 クロックサイクルを用いると，同一パイプライン上で，更に 3 つの変数のフリップを行うことができる．この 9 クロックサイクルを活用するために，4 つの独立した試行 (図 3.2 に示したように，解の探索は，変数の初期値を変えながら, 最大で *MAX-TRIES* 回繰り返される) を同一パイプライン上で実行する (マルチスレッド処理 [56, 57, 58, 59, 62])．図 6.2 に，本マルチスレッド処理の流れを示す．図 6.2 において，*FIFO tree* を用いて非充足クローズを集める処理 (F) は，その他の処理と並行して，全てのクロックにおいて行われている．これは，*break-value* の最大値, つまり変数のフリップにより非充足となるクローズの最大値は 12 以上であり，その場合，12 クロックサイクル以内にそれら全ての非充足クローズを集めることができないためである．このような状態が続くと，*FIFO tree* 内の各 FIFO はオーバフローを起こすが，32 ワードの FIFO により *FIFO tree* を構成すれば，今回評価した規模の問題ではオーバフローが起こることはない．

表 6.1: マルチスレッド処理の性能 (3-SAT ソルバー 2)

problem	#var	#cls	the number of threads			
			1	2	4	8
uf225-028	225	960	1.00	1.84	3.14	4.71
f600	600	2550	1.00	1.89	3.34	4.90
f1000	1000	4250	1.00	1.84	3.04	4.95
f2000	2000	8500	1.00	2.05	3.60	5.58

マルチスレッド処理の効果はシミュレーションにより調べることができる．表 6.1 は，[73] に収録されているベンチマーク問題のうち，4 つの問題を用いて性能向上率を評価したものである (各問題に対して 1000 回の探索を実行した場合の平均をとっている)．#var は変数の数，#cls はクローズの数である．表 6.1 に示したように，マルチスレッド処理を行うことにより，大規模な問題になるほど探索性能は向上する．

6.4 回路の詳細

6.4.1 Address Translation Table と Clause Table

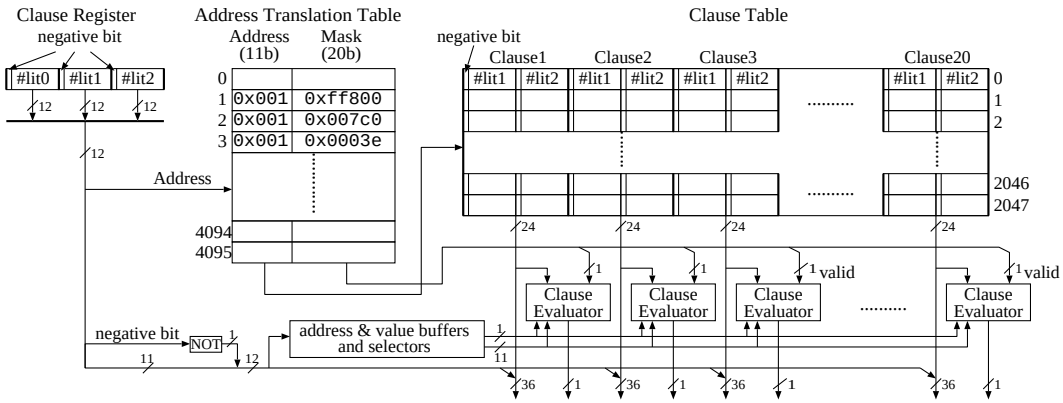

図 6.3: Address Translation Table と Clause Table の詳細 (3-SAT ソルバー 2)

図 6.3 に Address Translation Table と Clause Table の詳細を示す．Clause Table

には, 各リテラル L_i に対して, $\neg L_i$ を含む全てのクローズが並列に読み出せるように格納されている．これは既に述べたように，L_i の一時的フリップにより非充足となるクローズは, $\neg L_i$ を含むクローズのみであるからである．また，本ハードウェアでは，クローズは3つのリテラルの組で表される．このため，Clause Table には $\neg L_i$ を除く2つのリテラルの組がクローズデータとして格納されている．これにリテラル $\neg L_i$ を加えることにより，$\neg L_i$ を含む3つのリテラルからなるひとつのクローズを得ることができる．Clause Table からクローズデータを読み出すため，まず Clause Register に格納されているリテラル L_i が，Address Translation Table に対してアドレスとして与えられる．Address Translation Table には，Clause Table に対するアドレスおよびマスクデータが格納されている．Address Translation Table は最大で 4096 通りのデータを保持する．これは，本ハードウェアの対象としている問題が，最大 2048 個の変数 (最大 4096 通りのリテラル) を持つためである．

ひとつのリテラルを含むクローズ数は，今回評価の対象とした規模の問題の場合，最大で 19，平均では 6 である．最大 19 以上のクローズを並列に処理できるように，Clause Table の幅を 20 クローズ分としたため，Clause Table の記憶領域を有効利用するために，Clause Table のひとつのエントリには複数のリテラルに対するクローズデータが書き込まれる．そのため，マスクデータを用いて目的のクローズデータとそうでないクローズデータを区別している．例えば，図 6.3 では，Address Translation Table の 1,2,3 番地の Address フィールドの値は，いずれも 0x001 であり，これは，リテラル 1,2,3 に対するクローズデータが，全て Clause Table の 0x001 番地に格納されていることを意味する．このとき，リテラル 1,2,3 に対応するクローズデータは，Mask フィールドで示された値 0xff800, 0x007c0, 0x0003e の位置，すなわち，左から 9 個分，次の 5 個分，更に次の 5 個分に格納されている．図 6.3 では，Address Translation Table の 1，2 および 3 番地に対応するリテラルの否定を含むクローズデータは，Clause Table 内の同一のアドレス上に格納されている．そのため，マスクデータを用いて目的のクローズデータとそうでないクローズデータを区別している．

本ハードウェアの対象としている規模の問題では，Clause Table の深さは最大 1340 あれば十分であるため，本ハードウェアでは Clause Table の深さを 2048 とした．

6.4.2 各ユニットの詳細

図 6.4 に Clause Evaluator の詳細を示す．Clause Evaluator は，2 つの Variable Table を持つ．Variable Table は，FPGA の内部メモリである Block RAM(1b ×

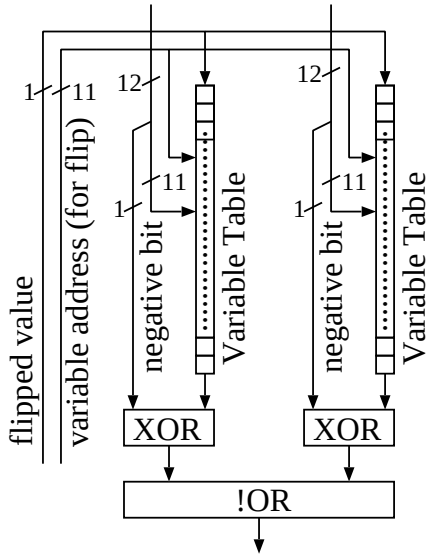

図 6.4: Clause Evaluator の詳細 (3-SAT ソルバー 2)

16K) により構成され，各変数に対して 4 通りの値の割当を保持している (スレッド数が 4 であるため).

Clause Evaluator は 2 つの Variable Table から読み出された値を用いてクローズの評価を行う ($\neg L_i$ の値は L_i のフリップにより 0 となるため). Clause Evaluator 2 および Clause Evaluator 3 は，Clause Evaluator とほぼ同一であるが，Clause Evaluator とは異なり 3 つの Variable Table を持っており，Unsatisfied Clause Buffer または FIFO tree から読み出されたクローズが非充足であるかどうかを評価するために用いられる.

Break-Value Counter は，リテラル L_i のフリップにより非充足となったクローズの数 (*break-value*) を計算するモジュールである．Break-Value Counter への入力データは，$\neg L_i$ を含む全てのクローズのクローズ値であり，最大 20 ビットである．これらの入力データは 5 ビットごとに 4 つのグループに分けられる．各 5 ビットのデータは Break-Value Counter 内の ROM に対してアドレスとして与えられ，ROM は与えられたアドレスに対応する *break-value* を出力する．このようにして得られた 4 つの *break-value* は，2 分木状に接続された加算器により足し合わされる．以上の処理は，非充足クローズ C_k の持つ 3 つのリテラル $L_i (i = 1, 2, 3)$ に対してそれぞれ実行され，これにより得られた 3 通りの *break-value* のうち，最も値の小さなものが選択される.

6.4.3 非充足クローズの選択方法とその影響

本ハードウェアシステムでは，Unsatisfied Clause Buffer の中からクローズをランダムにひとつ選択する必要がある．以下の議論では，Unsatisfied Clause Buffer に格納されているクローズの数を m とする．

クローズをランダムにひとつ選択するために，まずランダムな整数 $r(0 \leq r < m)$ を次のように生成する．Random Number Generator を用いて，18 ビットのランダムな整数 N_R を生成し，$1/m$ をかける．$1/m(m = 1, 2, ...)$ の値は，Block RAM により構成されるテーブル ($1/m$ Table) に格納されている．この値に m をかけたものを，N_R から引くことにより，m 未満のランダムな整数 r を生成することができる．

$(r = N_R - [N_R/m] \times m = N_R \bmod m)$

次に，Unsatisfied Clause Buffer の r 番地に格納されているクローズデータが読み出され，Clause Register にセットされる．

最後に，Unsatisfied Clause Buffer の r 番地を埋めるため，$FIFO_{tree}$ 内の $FIFO_{last}$ から読み出されたデータを Unsatisfied Clause Buffer の r 番地に書き込む．$FIFO_{last}$ が空の場合には，Unsatisfied Clause Buffer の最後尾のデータが代わりに書き込まれる．

以上の処理によって読み出されたクローズが，読み出された時点ですでに充足されていた場合，$FIFO_{last}$ に格納されているクローズが代わりに選択される．$FIFO_{last}$ が空の場合には，クローズの選択は行われず，回路はアイドル状態となる．また，選択されたクローズ中の各リテラルのフリップを行うと同時に，Clause Evaluator 3 を用いて，そのクローズが本当に非充足であるかどうか評価を行う．評価の結果，充足されていれば，その間回路はアイドル状態となる．

本研究で用いたベンチマーク問題について筆者らの調べたところによると，m の最大値は 1157 であった．そのため，本ハードウェアでは Unsatisfied Clause Buffer の深さを 2048 に設定した．

6.4.4 乱数とその影響

本システムにおいては，(1) フリップすべき変数の選択等 (2) 非充足クローズの選択等 において乱数が用いられる．(1) の乱数は，{0,1} および {0,1,2} の範囲となる．{0,1} の乱数列に関しては，大きさ 251 と 241 の ROM 2 個に予め {0,1} の乱数列を生成しておき，これらの ROM を同時に繰り返し読み出し，その値の組合せから {0,1} を生成している (周期は 251×241)．{0,1,2} の乱数列も同様である．(2) には 乗積法 $r_{i+1} = (2^8 + 1) \times r_i \ (mod\ 2^{18})$ を用いた．この方式により，

表 6.2: 性能比較 (3-SAT ソルバー 2)

			Software Solver			Our FPGA Solver				
problem	N_v	N_c	#flip	Mfps	time	#flip	Mfps	time	init	speed up
uf225 - 028	225	960	11612	0.766	0.0152	3449	3.31	0.0044	0.0029	3.4
f600	600	2550	133210	0.785	0.1697	41705	14.72	0.0118	0.0057	14.4
f1000	1000	4250	627292	0.761	0.8248	185943	22.13	0.0350	0.0077	23.6
f2000	2000	8500	4242254	0.568	7.6484	915152	25.82	0.1470	0.0133	50.8

Linux 上の *rand()*，*drand48()*，*random()* 等を用いた場合とほぼ同様の結果 (1000 回の試行の平均値の差が 5%以下) が得られることを，本ハードウェアと同じ動きをするソフトウェアシミュレータにより，複数の例題において確認した．

6.5 評価結果

本回路方式に基づき，XC2V6000 を 1 チップ搭載した Alpha Data 社製 FPGA ボード ADM-XRC II を用いて，2048 変数/8500 クローズまでの問題を処理可能な回路を実装した．回路の動作周波数は 85.2 MHz，回路使用量は 51 %であり，FPGA 内の全ての Block RAM のうち 90 %が用いられた．

表 6.2 に，本ハードウェアと，ソフトウェア (WalkSAT Version 35 [72]) を Pentium 4 3.2GHz，メインメモリ 2GB の PC 上で実行した場合との処理速度の比較を示す．ベンチマーク問題には，[73] から 4 つを用いた．表 6.2 に示されているのは，問題の規模 (#vars:変数の数, #cls：クローズの数)，解の発見に要したフリップ回数 (#flip)，平均フリップレート (M flips per second(Mfps))，解の発見に要した実行時間 (tm)，ホストプロセッサ上でのデータ生成とそれらを回路へダウンロードするために要した時間 (init)，全実行時間に対して回路がアイドル状態 (6.4.3 節を参照) となる割合 (idle)，ソフトウェアに対する本ハードウェアの速度向上率 (X) である．これらの値は 1000 回の試行の平均値である．

マルチスレッド処理が有効に機能するため，本ハードウェアにおいて解を見つけるまでに要するフリップ回数 (#flip) は，ソフトウェアにおいて必要となるフリップ回数より少ない．既に述べたように，今回の評価に用いたベンチマーク問題の場合，Unsatisfied Clause Buffer に格納されている非充足クローズのうち 20-25%が, 他のクローズを充足させるための変数のフリップにより充足される. 表 6.2 に示したように，回路がアイドル状態になる割合は，解探索にかかる全実行時間の 3-5%程度である．このことは，非充足クローズの 20%程度は $FIFO_{last}$ から選択されることを意味する．#flip は非充足クローズが $FIFO_{last}$ から選択される割合に依存して変化する．この依存関係については，より多くのベンチマーク問題

を用いて更に詳しく評価する必要があり，適切な割合で $FIFO_{last}$ から非充足クローズを選択すれば，#flip をより減らすことができると考えられる．

既に述べたように，ハードウェアシステムにおいては，大規模な問題をより高速に解くことが重要である．表 6.2 に示すように，ソフトウェアシステムにおける解探索の性能 (Mfps) は，問題が大規模になるほど悪くなっている．これは，ソフトウェアシステムの場合，問題が大規模になることによりキャッシュメモリのヒット率が低くなるためと考えられる．反対に，本ハードウェアの性能は，問題が大規模になるほど良くなっている．これは，全実行時間 (tm) に占める init の割合が減ること，マルチスレッド処理が大規模な問題に対して有効なことによる (表 6.1)．本ハードウェアでマルチスレッド処理を用いて解探索を行った場合，ソフトウェアに対する解探索の速度向上率は，ベンチマーク問題 f2000 においては 50 倍に達する (並列動作するユニット数は 20 であるが，各ユニット内において，2 つの Variable Table の読み出し，その論理和演算，Variable Table の更新，非充足クローズの生成/管理等が同時に行われるため，実際の最大並列度は数 100 を越える)．また，本ハードウェアの最大性能は，init の処理時間を無視できるような十分大きな問題に対しては，28Mfps(動作周波数/3) となる．

本ソルバーにおいて用いられている Block RAM の数は，SAT 問題に含まれるクローズの数にほぼ比例する．ゆえに，より大規模な FPGA を用いることにより，さらに大規模な問題を解くことができる回路の構築が可能となる．現在 最大規模の FPGA は，今回評価に用いた XC2V6000 の 3 倍強の Block RAM を持つため，約 3 倍の規模の問題をこのままの方式で処理することができる．このときの性能としては，本節で述べたものと同程度のものを期待することができる．

本ソルバーの構築には，FPGA ボード上の外部メモリバンクを使用していない．外部メモリバンクを用いることにより，更に大規模な問題を解くことができる回路を構築することができるが，外部メモリバンクのデータ転送速度が制約となり，処理速度は低下する．7 章および 8 章では，本ソルバーと同じ回路方式において外部メモリバンクを用いた異なる 2 つのソルバーを提案し，それぞれの回路における処理可能な問題規模とそのときの処理速度について述べる．そして，本回路方式における，処理可能な問題規模と性能とのトレードオフについて明らかにする．

第7章　3-SAT問題用ソルバー 3

7.1　処理の概略

7.1.1　基本構成

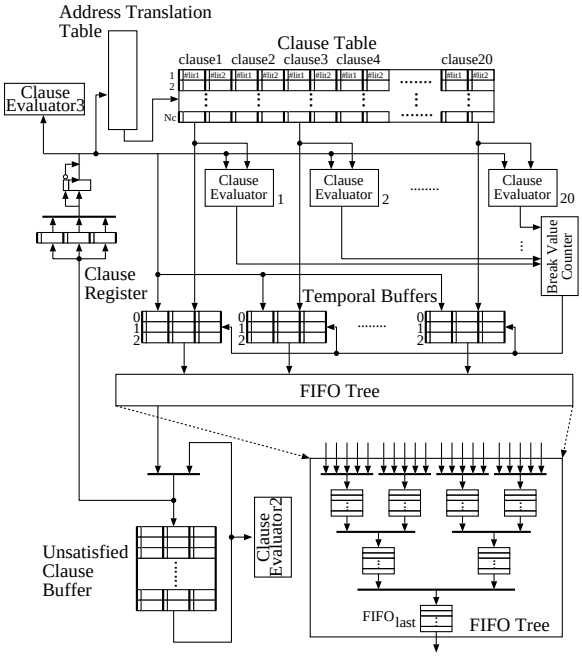

図 7.1: 基本構成 (3-SAT ソルバー 3)

図 7.1 に，本ソルバーの基本構成を示す．ソルバー 2 と同様に，クローズデータ (Address Translation Table と Clause Table)，変数に対する 4 通りの初期値とそれらに対する非充足クローズはホストコンピュータ上で生成され，FPGA にダウンロードされる．また，本ソルバーの評価に用いたベンチマークにおいても，変数のフリップが行われるときに評価されるクローズの数は 20 未満である．処理可能な問題規模の拡大のため，ソルバー 3 では，Clause Table と非充足クローズバッ

ファのを外部メモリバンクに格納している．そのため，外部メモリバンクのデータ転送速度がボトルネックとなり，処理の性能は制限される．これを緩和するため，Clause Table と非充足クローズバッファのそれぞれ一部は FPGA 内の Block RAM に格納される．

処理の概略はソルバー 2 と同様であり，まず非充足クローズバッファからクローズが読み出され，そのクローズ内の各リテラル番号により Address Translation Table へのアクセスおよび Clause Table からのクローズの読み出しが行われる (Address Translation Table は，FPGA 上の Clause Table へのアクセスのみに用いられる．外部 SSRAM 上の Clause Table は，リテラル番号を用いて直接参照される)．読み出されたクローズはクローズ評価モジュールを用いて評価され，その評価結果から各リテラルに対する *break-value* が算出され，Walksat/SKC アルゴリズムに基づき，変数のフリップが行われる．フリップにより新たに生じた非充足クローズがあれば，それらは FIFO Tree により収集され，非充足クローズバッファに格納される．

7.1.2 回路構成

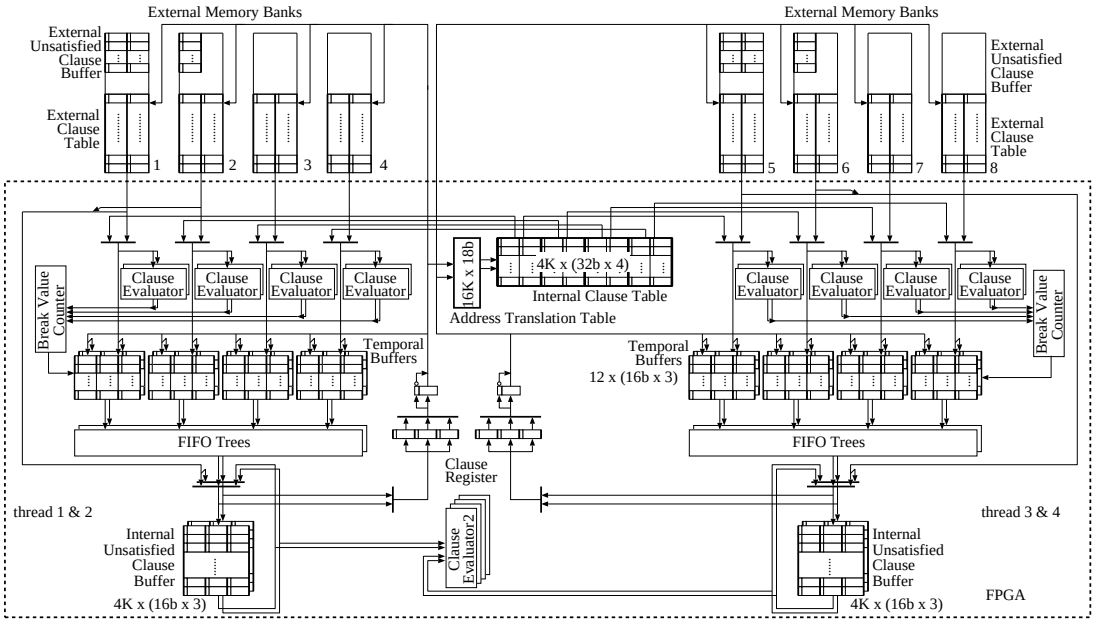

図 7.2: 回路の詳細 (3-SAT ソルバー 3)

本研究では，Alpha Data 社製 FPGA ボードである ADM-XRC-II を用いてソル

バー 3 の実装を行った．ADM-XRC-II には，XC2V6000 1 チップと，8 つの外部 SSRAM バンクが搭載されている (付加メモリボードの 2 バンクを含む)[47][48]．各バンクのデータ幅は 32 ビットである．

本ソルバーでは，Clause Table の各エントリに格納される 20 クローズのうち 8 クローズ分を外部 SSRAM バンクに割り当てる．その理由は以下の通りである．

- 1 つのリテラルは 16 ビットで表現される (論理否定の有無に 1 ビット，変数番号に 15 ビット ($2^{15} = 32K$))．
- Clause Table には，クローズに含まれる 3 つのリテラルのうち 2 つの組が，1 つのクローズデータとして格納される (これにより，各外部 SSRAM バンクの 1 エントリに 1 つのクローズが格納される．クローズデータの詳細は 6.4 節を参照されたい)．

以下の議論において，N_{c_l} は，ある特定のリテラルを含んでいるクローズの数を表す．

本ソルバーの評価に用いたベンチマークの場合，N_{c_l} の平均値は 6 である．そのため，多くのリテラルに対して，クローズデータは全て 8 つの外部 SSRAM バンクに格納することができる．外部 SSRAM バンクに格納し切れなかった場合には，その分を FPGA 内の Block RAM に格納することより，図 7.1 に示した本回路方式におけるクローズ評価の最大並列度を実現することができる．しかし，このような回路構成の場合，回路内の他のテーブルのサイズが大き過ぎるため，XC2V6000 上に実装することができない．例えば，この回路構成では Clause Evaluator を構成する Variable Table に 160 個の Block RAM が必要となるため，144 個の Block RAM しか持たない XC2V6000[49] にこの回路を実装することはできない．そのため，クローズ評価の最大並列度を削減して，Clause Evaluator の数を減らす必要がある．

図 7.2 に，$(N_v,N_c)=(32K, 128K)$ のときの回路構成の詳細を示す．その特徴は以下の通りである．

1. 同一の構造を持った 2 つのユニットが，Address Translation Table と Block RAM 上の Clause Table を共有し，同時に動作する．
2. 各ユニットは．同一パイプライン上で 2 つの独立した試行を行う (マルチスレッド処理)．
3. 各ユニットには，4 つの外部 SSRAM バンクが外部 Clause Table(および非充足クローズバッファ) としてそれぞれ割り当てられる．

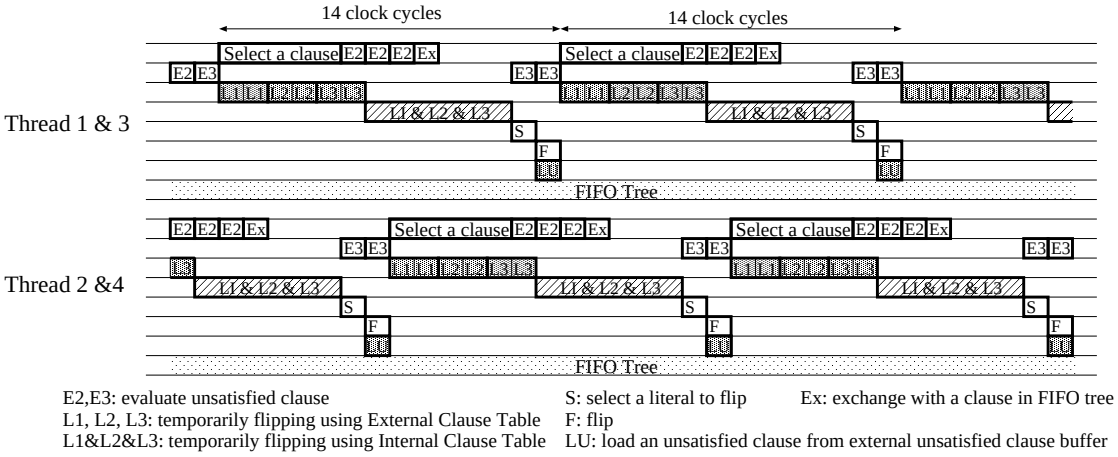

図 7.3: タイミングチャート (3-SAT ソルバー 3)

4. 各スレッドには，それぞれ 4 つの Clause Evaluator が割り当てられる．すなわちスレッドあたりのクローズ評価の最大並列度は 4 である．

5. 変数のフリップに 14 クロック (うち 3 つのリテラルの一時的フリップに 12 クロック) を要する (ソルバー 2 では変数のフリップに 12 クロック，そのうちリテラルの一時的フリップに 3 クロックである)．

以降の節では，これらの特徴について，その背景を説明する．

7.2 回路の詳細

7.2.1 並列処理とマルチスレッド処理

限られた容量の Block RAM で，大規模な問題に対して高い処理性能を実現するためには，Block RAM の dual port access 機能を最大限活用することが重要である．Block RAM 上に構成される各種のテーブルは，dual port access 機能を用いて，異なるスレッドにより同時にアクセスされる．図 7.2 において，Address Translation Table と Block RAM 上の Clause Table(内部 Clause Table) は，2 つのユニットにより共有され，同時に動作する．また，処理の遅延を隠蔽するため，各ユニット内において同一パイプライン上で 2 つのスレッドが実行される (マルチスレッド処理)．各ユニットには 4 つの外部 SSRAM バンクが割り当てられ，外部 Clause Table および非充足クローズバッファとして使用される．

7.2.2 各スレッド内の並列性

既に述べたように，本回路方式のクローズ評価の最大並列度 (N_{c_l} の最大値) は 20 である．ソルバー 2 では，スレッド数分 (4 つ) の Variable Table を 1 つの Block RAM (1 ビット $\times$ $16K$) に格納し (2048 変数のとき，$N_v \times 4 = 16K$)，それを各スレッド間で共有することにより，最大並列度を実現している．しかし，$N_v = 32K$ の場合には Variable Table をスレッド間で共有することができない (1 スレッド分の Variable Table に 2 つの Block RAM が必要となる)．ゆえに，$N_v = 32K$，4 スレッドの回路において，全ての Variable Table を構成するために必要となる Block RAM の総数は,

$$ClauseEvaluator \text{ の個数} \times 2 \times \text{スレッド数}$$

となる．他のテーブルおよびバッファに用いる Block RAM の数を考慮し，スレッドあたりのクローズ評価の並列度は 4 とした．これにより，クローズ評価の最大並列度はソルバー 2 の 1/5 となるが，N_{C_l} の平均は 6 であるため，それほど大きな性能低下の要因とはならない．

図 7.2 において，各スレッドに対して 4 つの Clause Evaluator が割り当てられ，4 クローズが同時に評価される．N_{c_l} 個のクローズのうち 8 つは SSRAM バンク上の外部 Clause Table 上に置かれ，2 クロックサイクルで読み出される．残りのクローズは内部 Clause Table に格納される．

以下に，リテラルの一時的フリップの手順を述べる．ランダムに選択された非充足クローズの持つ各リテラルに対し，まず外部 Clause Table から，クローズが計 6 クロックで読み出される (図 7.3 の $L1$，$L2$，および $L3$ がこれに該当する)．次に，内部 Clause Table からクローズが計 6 クロックで読み出される．内部 Clause Table のアクセスに用いられる 6 クロックは，各リテラルに対して動的に割り当てられる．これは次の理由による．

1. 今回評価に用いたベンチマークでは，クローズ内の 3 つのリテラルに対する N_{C_l} の合計は $48(= 4 \times (6 + 6))$ 未満である．

2. そのため，3 つのリテラルに対するクローズの評価は 12 クロック以内に完了する．

3. $1 \leq N_{C_l} \leq 19$ であり，外部 Clause Table にはそのうち最大 8 個のクローズが格納されるため，内部 Clause Table に格納する必要のあるクローズの数

は 0〜11 である．ゆえに，リテラルあたりの内部 Clause Table のエントリ数は 0〜3 である．

4. 内部 Clause Table からのクローズの読み出しに必要なステップ数は，0〜3 の間で変化する．

以上の 12 クロックによるリテラルの一時的フリップに加えて，*break-value* の算出とそれに基づくリテラルの選択，および選択されたリテラルのフリップのために，更に 2 クロックを必要とする．そのため，1 回の変数のフリップにつき 14 クロックが必要となる．

本ソルバーでは，4 つのスレッド，すなわちスレッド 1 および 2(図 7.2 における左側半分)，スレッド 3 および 4(図 7.2 における右側半分) が並行して動作する．図 7.3 に本ソルバーのタイミングチャートを示す．スレッド 1 および 3 は同時に動作し，これより 7 クロック遅れてスレッド 2 および 4 が同時に動作する．

7.2.3 Address Translation Table

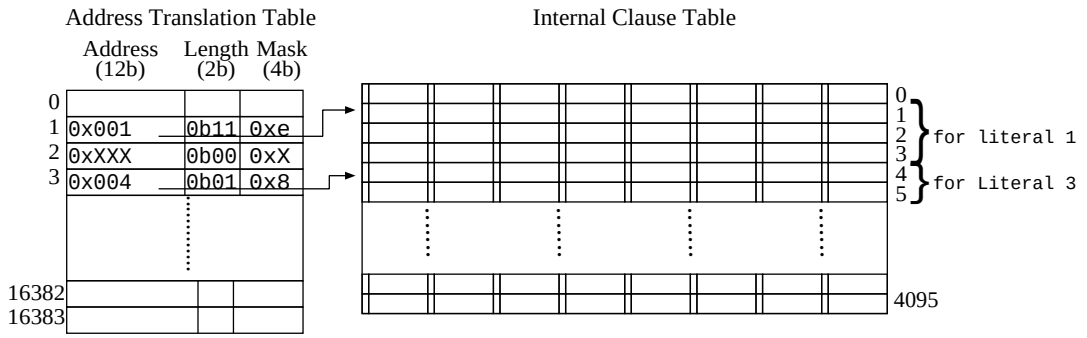

図 7.4: Address Translation Table(3-SAT ソルバー 3)

図 7.4 に本ソルバーの Address Translation Table の詳細を示す．Address は，内部 Clause Table における先頭のエントリのアドレス，Length は内部 Clause Table 内のエントリ数である．Mask は，ソルバー 2 の Address Translation Table における Mask と同じ役割を果たすが，ソルバー 2 とは異なり，リテラル L_k に対する内部 Clause Table 内のエントリのうち，最後に読み出されるエントリに対してのみ使用される．Length の最大値は 3 である．すなわち，1 リテラルあたり最大 12 クローズが内部 Clause Table に格納される．図 7.4 において，リテラル 1 に対応す

るクローズデータは，内部 Clause Table の 1 番地から，3 番地の第 3 列目までに，リテラル 3 に対応するクローズデータは，内部 Clause Table の 4 番地から，5 番地の第 1 列目までに，それぞれ格納されている．リテラル 2 は内部 Clause Table のエントリを持たないリテラルである (補リテラルであるリテラル $\neg 2$ が内部 Clause Table のエントリを持つ．これについては後述する)．

$N_v = 32K$ のとき，全てのリテラルに対して Address Translation Table のエントリを用意すると，必要とされる Block RAM 数は 72($= (12 + 2 + 4)$ ビット $\times$ $32K \times 2/(1$ ビット $\times 16K))$ となり，XC2V6000 の Block RAM の半数が Address Translation Table に割り当てられることになる．

Address Translation Table の小型化のため，本ソルバーでは $N_{C_l} > 8$ であるリテラルに対して Address Translation Table のエントリを用意する．ただし，Address Translation Table へのアクセスを簡略化するため，$N_{C_l} \leq 8$ であるリテラルであっても，その補リテラル (リテラルの論理否定) が $N_{C_l} > 8$ である場合にはエントリを用意する (ただし，内部 Clause Table のエントリは補リテラルにのみ用意される)．変数番号の大きさで Address Translation Table にエントリを持つリテラルと持たないリテラルとを区別できるようにするため，正負両リテラルの N_{C_l} の合計が小さい変数に，より小さい変数番号が割り当てられるよう，ホスト PC 上で変数番号の入れ換えが行われる．

以上により，$N_v = 32K$ のとき，Address Translation Table のエントリ数を $16K$ まで削減することが可能であり，これにより Address Translation Table に必要とされる Block RAM の個数は 18($= (12 + 2 + 4)$ ビット $\times 16K/(1$ ビット $\times 16K))$ と，大幅に削減される．

7.2.4 Unsatisfied Clause Buffer

表 7.1: 非充足クローズのキャッシュを行う場合のフリップ回数

	$4K$ クローズ分	無制限
sat04-903	596317	596543
sat04-904	5788391	6445473
sat04-910	2308837	2102668
v32k-c128k	2277993	2259274

非充足クローズの総数は，通常，解探索の初期に最大となり，探索の進行に伴って減少する．$N_c = 128K$ のとき，変数の初期値に対する非充足クローズ数の期待値は $128K \times (1/2)^3$ であり，その全てを XC2V6000 上に格納することはできない．

本ソルバーでは，非充足クローズの一部を Block RAM 上の非充足クローズバッファ(内部バッファ) にキャッシュし，それ以外は外部メモリバンク上の非充足クローズバッファ(外部バッファ) に格納する．非充足クローズの選択は，内部バッファの中からランダムに行われる．外部バッファ上の非充足クローズは，解の探索が進み，内部バッファ内の非充足クローズが減少するのに伴って，徐々に内部バッファへ移動される．

表 7.1 は，上記の方式において内部バッファの容量を $4K$ クローズ分とした場合と，内部バッファの容量が無制限である場合とにおける，解が得られるまでのフリップ回数を比較したものである．表 7.1 に示すように，どちらの場合も解が得られるまでのフリップ回数に有意差は存在しない．

7.3 評価結果

表 7.2: 性能比較 (3-SAT ソルバー 3)

			Software Solver			Our FPGA Solver			
problem	N_v	N_c	#flip	Mfps	time	#flip	Mfps	time	speedup
sat04-903	18000	48060	1283091	0.433	2.864	596317	11.47	0.208	13.8
sat04-904	18000	48060	16104531	0.470	33.914	5788391	17.96	1.289	26.3
sat04-910	22000	58740	9364619	0.421	21.637	2308837	15.81	0.584	37.0
v32k-c128k	32768	131072	2963965	0.134	21.968	2277994	12.48	0.730	30.1

本ソルバーの性能評価について述べる．回路の動作周波数は 67.2MHz，回路使用量は 88%，Block RAM の使用量は 90% であった．表 7.2(各項目は 6.2 と同一である) に，本ソルバーと，ソフトウェアとの処理速度の比較を示す．評価に使用したソフトウェアとその動作プラットフォームはソルバー 2 と同一である．ベンチマーク問題には，[74] から 3 つ，およびベンチマーク生成ソフトウェア [75] を用いて生成したものをひとつ用いた．各問題の初期化時間は，sat04-903 および sat04-904 が 0.084 秒，sat04-910 が 0.104 秒，v32k-128k が 0.256 秒であった．これらの値は 100 回の試行の平均値である．

Solver 2 と同様，本ソルバーにおいてもマルチスレッド処理の効果により，ソフトウェアと比較して解を発見するまでに必要とするフリップ回数は少なくなって

いる．表 7.2 に示すように，ソフトウェアの性能 (Mfps) は，問題規模が大きくなるほど悪くなっている．これも，Solver 2 の場合と同様，問題の大規模化によりキャッシュミスヒット率が上昇するためであると考えられる．最大性能は 19.2Mfps であり，ソルバー 2 と比較してやや低下している．これは，外部 SSRAM バンクの制御のため回路が複雑化しており，その分動作周波数が低下することと，1 回の変数のフリップに必要なクロック数が増加していることによるものである (ソルバー 2 では 12 クロック，本ソルバーでは 14 クロックを要する)．

本ソルバーの回路実装方式において処理可能な問題の規模は，Variable Table に用いることのできる Block RAM の数によって決まる．外部メモリバンクに配置するデータ量を増やすことにより，Variable Table に用いることのできる Block RAM の数を増やすことができるが，外部メモリのアクセスによるオーバヘッドを隠蔽することができなくなるため，処理の性能は低下する．次章では，外部メモリを全面的に使用した場合における処理可能な問題規模，および期待される処理性能について考察する．

第8章　3-SAT問題用ソルバー 4

8.1　ハードウェアの特徴

本ソルバーは，ソルバー2およびソルバー3の回路方式において，Clause Tableを全て外部メモリバンク上に置いたものである (Address Translation Tableは使用せず，リテラル番号により直接Clause Tableを参照する)．これにより，ADM-XRC-II(付加メモリボード上の2バンクを含む) を用いた場合，128K変数/576Kクローズまでの問題を処理することが可能となる (ソルバー2およびソルバー3の評価に用いた問題と同様，$N_v : N_c \fallingdotseq 4.3$ の場合．現在 最大規模のFPGA，および，より大規模なメモリバンクを用いれば，この3倍程度の規模まで処理可能となる)．一方，外部メモリのデータ転送速度の制約により，処理性能はソルバー2および3と比較して大きく低下する．

以下，本ソルバーにおける外部メモリの使用方法および処理の性能について，その詳細を述べる．

8.2　外部メモリの使用方法

1. 6バンク (計 1M × 192b (= 32b × 6)) を Clause Table に用いる．128K変数の時，リテラルのデータ幅は18bとなるため，192bで10リテラル，すなわち5クローズ分のデータを同時に読み出すことができる．

2. 2バンク (付加メモリボード上のバンク; 各 2M × 32b) を Unsatisfied Clause Buffer (1M × 32b) および $1/m$ Table (512K × 32b) に用いる．非充足クローズには3リテラル (54b) が必要であるため，クローズ当たり2ワードを用いる．

このとき，パイプラインは以下のようになる．

1. 1リテラルの評価 (最大20個のクローズの評価) に4クロックサイクル (各クロックサイクル毎に5クローズを並列に評価．有効でないクローズは，2つのリテラルの値が共に0となっている)．従って，1フリップに12クロックサイクルを要する．

2. 非充足クローズの収集，管理，選択に 12 クロックサイクル (外部メモリを用いるため FPGA 内部メモリの場合より長くなる).

総パイプライン段数は 24 段，うち 12 段が変数のフリップとなるため，実行可能なスレッド数は 2(=24/12) となる．各スレッド毎に Unsatisfied Clause Buffer & $1/m$ Table の 1 バンクを用いる．図 8.1 にこのときの回路構成を示す.

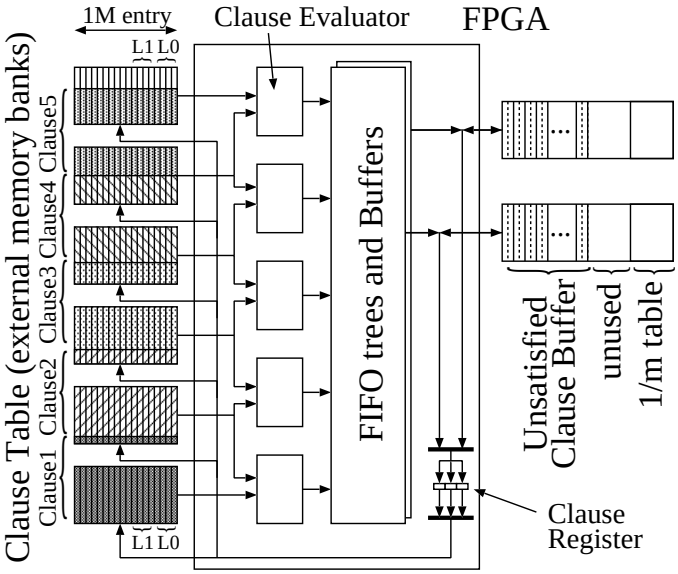

図 8.1: 回路のブロック図 (3-SAT ソルバー 4)

8.3 処理可能な問題規模と期待される性能

処理可能な問題の規模は Variable Table に用いることのできる Block RAM の数 および メモリバンクの大きさによって決まる．非充足クローズの FPGA 内での一時的な保持に 28 個の Block RAM が必要なため (54b × 4K のバッファを 2 個)，Clause Evaluator 2, 3 も含めて，1 ユニット当たり 16 個，1 スレッドあたり 8 個 (16K × 8 = 128K 変数) を Variable Table に使用することができる (dual port access を利用)．また，128K 変数の時，Clause Table の大きさは，128K × 2 (否定の分を含む) × 4 (リテラル当たり 4 エントリ) = 1M エントリ となる.

このとき期待される性能は，ソルバー 2 の $1/4 = 12/24 \times 1/2$(1 クローズの評価時間が 12 から 24 クロックに増加，かつ，マルチスレッド処理の効果が約 1/2) となる．ソフトウェアシミュレータを用いた評価の結果，66MHz で動作する場合 (付加メモリボード参照のために 66MHz とした)，[75] を用いて生成した 128K 変数/512K クローズの問題における処理の性能は，ソフトウェアの 16.4 倍となった.

第9章　形式検証用大規模ソルバー

本章では，形式検証問題を対象とした大規模ソルバーについて述べる．形式検証は，ハードウェアやソフトウェアの設計の正しさを数学的に検証するための手法であり，充足可能性問題と等価な応用問題のうち，最も重要なもののひとつである．形式検証問題から生成される SAT 問題は，その殆どが k-SAT 問題であり，本論文で提案する 3-SAT 問題用ソルバーの手法をそのまま用いることはできない．補助変数の導入により 3-SAT へ変換して解く方法も考えられるが，その場合，問題規模が大幅に増加するため，処理可能な問題規模が限定される要因となる．また，形式検証の問題規模は大変大きく，データの格納には SDRAM が必要となる．この場合，ソルバーの性能は SDRAM のデータ転送速度により制限される．

本章では，そのような制約のもと，SDRAM のスループットとアクセス遅延，FPGA の動作周波数をパラメータとして，FPGA ソルバーにおいて，形式検証から生成される SAT 問題に対してどの程度の高速化が可能であるか考察する．

9.1　Walksat アルゴリズムの改良

充足可能性問題を用いた形式検証では，設計と検証仕様が与えられ，それらをもとに SAT 問題が生成される．形式検証問題から生成された SAT 問題においては，その CNF 表現から，もとの設計における論理ゲートやその接続関係を推定することが可能であり [76][77]，そのような構造を把握し，それ利用した探索を行うことにより，SAT 問題を効率よく解くことができる [78]．

筆者らの知見によれば，Walksat アルゴリズムにおいて，論理ゲートの出力信号に相当するリテラルを優先的にフリップすることにより，探索は局所最適解に向かって急速に収束する (論理回路の入力から出力方向に向かって探索することに相当するため，本論文ではこれを前向き探索と呼ぶ)．これに加えて入力信号に相当するリテラルのフリップを適宜行うことにより (論理回路の出力から入力方向に向かって探索することに相当するため，本論文ではこれを後向き探索と呼ぶ)，局所最適解から容易に脱出することができる．本研究では，この前向き探索と後向

き探索を Walksat/SKC アルゴリズムに導入した．これにより，形式検証問題から生成された SAT 問題を効率良く解くことが可能となる．

9.1.1 論理ゲートの検出方法

$o \leftarrow AND(i_1, i_2, ..., i_{n-1})$:

$$(i_1 \vee \neg o) \wedge (i_2 \vee \neg o) \wedge ... \wedge (i_{n-1} \vee \neg o) \wedge (\neg i_1 \vee \neg i_2 \vee ... \vee \neg i_{n-1} \vee o)$$

$o \leftarrow OR(i_1, i_2, ..., i_{n-1})$:

$$(\neg i_1 \vee o) \wedge (\neg i_2 \vee o) \wedge ... \wedge (\neg i_{n-1} \vee o) \wedge (i_1 \vee i_2 \vee ... \vee i_{n-1} \vee \neg o)$$

図 9.1: AND/OR ゲートの CNF 表現

図 9.1 に AND ゲートおよび OR ゲートの CNF 表現を示す．AND/OR ゲートは，信号数分 ($= n$ 個) のリテラルを含むひとつの n-ary クローズと，入力信号数個 ($= n - 1$ 個) のバイナリクローズで表される．n-ary クローズは，全ての入出力信号に相当するリテラルから構成され，各バイナリクローズは，出力信号に相当するリテラルと，入力信号のうちどれかひとつに相当するリテラルから構成される．また，バイナリクローズ内の各リテラルは，同一ゲートに対応する n-ary クローズにおいて，補リテラルとして出現する[1]．

出力信号は全てのクローズに含まれるため，出力信号を特定するためには，ひとつの n-ary クローズと，その n-ary クローズと同一のゲートに対応する，少なくとも 2 つのバイナリクローズを検出すればよい．また，出力信号として検出された変数と，n-ary クローズに含まれるその他の変数との組合せから，そのゲートに対応する他のバイナリクローズを特定することもできる．

以上のことから，次のようにして，論理ゲート，およびその出力信号の検出を行う．

1. 同一のゲートに対するバイナリクローズおよび n-ary クローズの検出

   各バイナリクローズに対して，そのクローズ内のリテラルの補リテラルを全て含む n-ary クローズを探す．n-ary クローズが複数検出された場合は，どのクローズがそのバイナリクローズに対するクローズであるかわからないため，そのバイナリクローズに対する n-ary クローズの特定は後回しにする．

[1] これらは NAND，NOR ゲートにも同様に当てはまる．

2. 出力信号の検出

   対応する 2 つのバイナリクローズが検出された n-ary クローズ，およびそれら 2 つのバイナリクローズについて，出力信号を検出する．

3. 未検出のバイナリクローズの検出

   検出された出力信号と，n-ary クローズに含まれるリテラルから，同一ゲートに対応する未検出のバイナリクローズを検出する．

検出の例

図 9.2 の回路における，出力信号の検出の手順を述べる．バイナリクローズ (A, C), (B, C), (B, E), (C, E), (D, E), (D, F), (E, F) に対して，対応する n-ary クローズをそれぞれ特定すると，

1. (A, C) の補リテラルを全て含む ((A, C) に対応する) のは $(-A, -B, -C)$ のみ．
2. (B, C) に対応するのは，$(-A, -B, -C), (-B, -C, -D, -E)$.
3. $(B, E), (C, E)$ に対応するのは $(-B, -C, -D, -E)$ のみ．
4. $(D, F), (E, F)$ に対応するのは $(-D, -E, -F)$ のみ．
5. (D, E) に対応するのは $(-B, -C, -D, -E), (-D, -E, -F)$.

1. より，(A, C) および $(-A, -B, -C)$ の出力信号は A, C のどちらか一方であるとわかる．

2. より，どちらの n-ary クローズが (B,C) に対応するのかわからないため，検出は後回しにする．

3. より，クローズ $(-B, -C, -D, -E)$ と，同一のゲートに対するバイナリクローズが 2 つ判明しているため $((B, E), (C, E))$，これら 3 つのクローズに共通の変数である E が，出力信号として検出される．

4. より，3. と同様にして，F が出力信号として検出される．

5. は 2. と同様であるが，4. より，$(-D, -E, -F)$ に対応するバイナリクローズは全て判明しているため，(D, E) に対応する n-ary クローズは $(-B, -C, -D, -E)$ と判明する．これにより $(-B, -C, -D, -E)$ と，これに対するバイナリクローズが全て検出されたため，(B, C) に対応する n-ary クローズは $(-A, -B, -C)$ のみとなる．これと 1. から，$(-A, -B, -C)$ に対するバイナリクローズが全て検出され，B が出力信号として検出される．

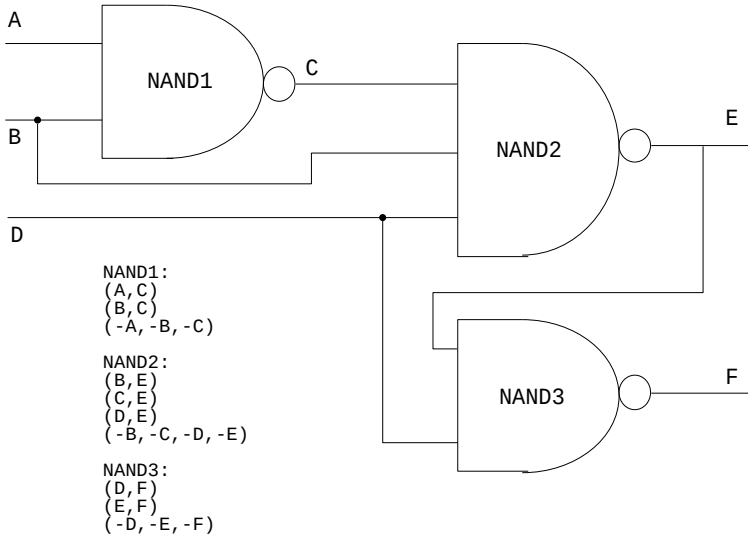

図 9.2: 出力信号の検出

9.1.2 改良アルゴリズムの詳細

図 9.3 に改良アルゴリズムの詳細を示す．局所最適解からの脱出を容易にするため，本アルゴリズムではノイズパラメータの動的調整を導入した [31]．[31] では，次のようにノイズパラメータ p の動的調整を行っている．

- 変数のフリップを $\theta \times m$ 回行っても充足されるクローズの個数が増加しなければ，次のように p を増加させる (ここで，m は与えられた問題のクローズの総数，θ は任意の自然数，ϕ は任意の実数である)．

  $$p = p + (1 - p) \times \phi$$

- 変数のフリップにより充足されるクローズの個数が増加すれば，次のように p を減少させる．

  $$p = p - p \times \phi/2$$

改良アルゴリズムにおける，フリップされる変数を決定するためのヒューリスティックは次の通りである．

1. ランダムに選択された非充足クローズ内の各変数に対して，*break-value* をそれぞれ計算する．

2. *break-value* $= 0$ となるような変数があれば，その中からランダムにひとつ選択する．

```
Walksat-ex(F, MAX-TRIES, MAX-FLIPS)
 for (i = 0; i < MAX-TRIES; i++) {
   T = randomly generated truth assignment;
   for (j = 0; j < MAX-FLIPS; j++) {
    if T satisfies F return T;
    c = an unsatisfied clause selected at random;
    v = a variable in c chosen by the Heuristic(c);
    T = T with v flipped;
    Noise parameter tuning();
   }
 }
}

the Heuristic(c) {
 L is empty;
 for each literal li in c {
   count break-value;
   if (break-value = 0) L=L ∪ li;
 }
 if (L is not empty) {
   flip any of variables in L randomly;
 } else {
  with probability p, flip any variable randomly;
  with probability 1-p, flip the variable which
                    corresponds to the output signal of c;
 }
}

Noise parameter tuning() {
 if (there is no improvement) {
   step = step + 1;
   if (step > 100) {
    p = p + (1 − p) × φ; step = 0;
   }
 } else {
   p = p − p × φ/2; step = 0;
 }
}
```

図 9.3: 改良アルゴリズム

3. *break-value* $= 0$ となるような変数がなければ,

   (a) 確率 p で，ランダムに変数を選択する．

   (b) 確率 $1-p$ で，そのクローズに対応する論理ゲートの出力信号に相当する変数を選択する．

上記の処理手順のうち，ステップ 3 が，本アルゴリズムと Walksat/SKC との差分である．ステップ 3 において，(a) は後向き探索に，(b) は前向き探索に相当する (ただし，(a) においても，出力信号が選択された場合は前向き探索となる)．ノイズパラメータの値により，出力信号の選択の優先度が決定される．出力信号は，プリプロセッサにより予め検出される．本アルゴリズムにおけるノイズパラメータの自動調整は，[31] を簡略化し，$\theta \times m$ を 100 に固定したものである．

9.2 改良アルゴリズムの性能

表 9.1: 改良アルゴリズムの性能

Instances	size		Our algorithm			RSAPS		
	N_v	N_c	flips (M)	t (sec)	ratio	flips (M)	t (sec)	ratio
velev-fvp-sat-3.0-b18	35853	1012271	1.66	4.45	1.0	92.20	191.80	0.7
12pipe_bug8	117526	8760516	0.54	1481.34	1.0	2.04	87.82	1.0
velev-vliw-2.0-b6	235926	8076539	112.70	3774.30	0.9	9.10	229.64	1.0
velev-vliw-4.0-b8	521179	13378617	10.76	2881.74	1.0	122.16	5916.20	0.5

表 9.1 に改良アルゴリズムと RSAPS アルゴリズム [34] との性能比較を示す．RSAPS アルゴリズムは，Incomplete 型アルゴリズムのうち，最も高い性能を示すもののひとつであり，応用問題から生成された SAT 問題に対して高い性能を示す．ベンチマーク問題には，応用問題における SAT ソルバーの性能を競うコンテストである SAT-Race[68, 69, 70] より，形式検証から生成されたものを 4 題用いた．表 9.1 において，N_v は変数の数，N_c はクローズの数を表し，*flip* は 10 回の試行における解が求まるまでのフリップ回数の平均値，t は 10 回の試行における実行時間の平均値，*ratio* は 10 回の試行のうち解を発見することができた試行の割合 (全ての試行において解を発見することができた場合は 1.0 となる) を表す．評価のプラットフォームは Intel Core i7-920 2.79GHz，メインメモリ 4GB であり，各々の試行におけるフリップ回数の上限は 2^{28} とした．

表 9.1 に示したように，改良アルゴリズムは，今回評価に用いたベンチマークのうち，2 題において RSAPS よりも高い性能を示している．他の 2 題においては，実行時間では RSAPS を下回っているが，解を発見することのできた試行の割合

では上回っている．これらのことから，改良アルゴリズムは，その単純なヒューリスティックにもかかわらず，形式検証問題においてRSAPSと比較して遜色ない性能を示していると言える．

9.3 実装方式の概略

9.3.1 データ構造と処理の概略

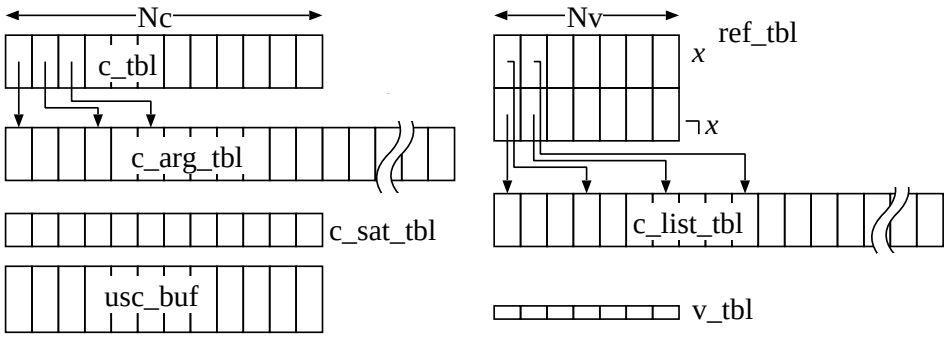

図 9.4: テーブル概略 (形式検証用ソルバー)

図 9.4 に，本ソルバーにおいて用いられるテーブルの概略を示す．Clause Table(c_tbl[]) のエントリは，c_arg_tbl[] を参照するためのポインタである．c_arg_tbl[] は，クローズ内のリテラルを保持するテーブルである．c_sat_tbl[] は，クローズ内のリテラルのうち，真であるリテラルの数を保持する[2]．c_sat_tbl[] のエントリの値が 0 の場合，そのエントリに対応するクローズは，与えられた問題の変数に対する現在の真偽割当に対して非充足であることを意味する．usc_buf[] は，非充足クローズを保持するバッファである．c_list_tbl[] は，リテラル x および $\neg x$ を含むクローズのリスト (以下，それぞれ *clauses list* x および *clauses list* $\neg x$ と呼ぶ) を保持するテーブルである．ref_tbl[] は 2 ワード幅の配列であり，変数 x に対して，c_list_tbl[] における *clauses list* x および *clauses list* $\neg x$ のアドレスを保持する．v_tbl[] は，変数の値を保持するテーブルである．

以下に本ソルバーにおける処理手順を示す．まず，ホストコンピュータ上で以下の処理を行う．

1. 与えられた問題に対する c_tbl[] と c_arg_tbl[] を生成する．

[2]後述するように，c_sat_tbl[] のエントリは，リテラル数 4 以上のクローズのみに用意される．

2. 生成された c_tbl[] と c_arg_tbl[] をスキャンし，
   (a) 各クローズにおける出力信号を検出する．
   (b) 各変数 $x_i(i = 1, ..., N_v)$ に対して，*clauses list* x_i および *clauses list* $\neg x_i$ を生成する．
3. 変数の初期値をランダムに生成する．
4. 生成された初期値に対するクローズの値を評価し，
   (a) 各クローズにおける充足されたリテラルの数を数え，それを c_sat_tbl[] に書き込む．
   (b) 非充足クローズを usc_buf[] に格納する．
5. 生成された各テーブルを FPGA にダウンロードする．

次に，FPGA 上で以下の処理を行い，解を探索する．

1. usc_buf[] から非充足クローズ c をひとつ選択する (c が充足されていた場合は，これを破棄して他のクローズを選択する．非充足クローズが選択されるまでこの処理を繰り返す)．
2. c に含まれるリテラルを，c_tbl[c] をアドレスとして，c_arg_tbl[] から読み出す．
3. c に含まれる各リテラル l(c は非充足クローズであるため，全ての l は非充足である) に対して，以下の処理を行う．
   (a) *break-value* を検出するため，ref_tbl[l] から *clauses list* $\neg l$ のアドレスを読み出し，それを用いて c_list_tbl[] から *clauses list* $\neg l$ を読み出す (これは，l をフリップすることにより，$\neg l$ を含むクローズが非充足となるためである)．
   (b) *clauses list* $\neg l$ 内のクローズのうち，c_sat_tbl[] の値が 1 であるクローズの数 (=*break-value*) を数える (これらのクローズは，リテラル l をフリップすることにより新たに非充足となるクローズである．ゆえに，その総数はリテラル l に対する *break-value* に一致する)．
   (c) *break-value* が 0 であれば，l を集合 L に追加する．
4. 集合 L が空でなければ，その中からランダムにリテラルを選択する．

   L が空であれば，

- 確率 p で，c の中からランダムに選択する．
- 確率 $1-p$ で，c の出力信号に該当するリテラルを選択する．

5. 選択されたリテラル l_f をフリップする．
6. *clauses list* $\neg l_f$ を読み出し，リスト内の各クローズに対する c_sat_tbl[] の値をデクリメントする．値が 0 となったクローズがあれば，そのクローズを usc_buf[] に追加する．
7. *clauses list* l_f を読み出し，リスト内の各クローズに対する c_sat_tbl[] の値をインクリメントする．
8. 以上の 1〜7 を，usc_buf[] が空になるまで繰り返す．

以上の処理手順において，c_list_tbl[] へのアクセスが最も連続的であり，SDRAM のバーストリード [44, 45][46] を用いてこのテーブルへのアクセスを高速化することにより，ソルバーの性能を向上させることができる．

表 9.2 に，本ソルバーの評価に用いたベンチマークにおける，各クローズのリテラル数，および *clauses list* の長さに関するデータを示す (ベンチマーク名はそれぞれ短縮している)．項目#args のうち，2(%) および 3(%) は，それぞれ，リテラル数が 2 および 3 のクローズの，全クローズに占める割合 (%) である．表 9.2 に示すように，リテラル数 3 以下のクローズが全クローズの大部分を占め，リテラル数 4 以上のクローズの占める割合はわずかである．しかしながら，例えば 4.0-b8(velev-vliw-4.0-b8 の省略である) においては，リテラル数 4 以上のクローズの数は 454660 個と大変多い．今回評価に用いたベンチマークでは，各クローズにおけるリテラル数の最大値は 134 から 6689 である．*clauses list* の長さは，平均では 75 から 220 程度であるが，その最大値は 110,000 を超える．このことは，特定のクローズにおいては，フリップされるリテラルを決定するために非常に長い計算時間を必要とすることを意味する．

表 9.2: 各ベンチマークのリテラル数と *clauses list* 長さ

	#args			length of clause list	
	2(%)	3(%)	max	ave	max
3.0-b18	50.9	47.0	134	83.1	1083
bug8	93.7	5.2	340	221.9	37863
2.0-b6	42.6	55.3	169	101.1	24572
4.0-b8	65.1	31.5	6689	75.2	110469

9.3.2 SDRAM を使用した場合の処理並列度

本ソルバーのアルゴリズムには高い並列性が内在するが，実際の処理の並列度は，FPGA ボードの外部メモリバンクのデータ転送速度により制限される．これは，テーブルの大部分を外部メモリバンクに置かなければならないためである．Xilinx 社の Virtex-6 シリーズ FPGA の DDR3 SDRAM インタフェース [53] は最大 1066Mbps(DDR-1066) のデータ転送速度をサポートしており，32 ビット幅のメモリバンクを 4 つ使用した場合，266MHz(133MHz) で動作する FPGA に対して 16(32) ワードのデータを並列に転送することができる．本ソルバーにおけるテーブルのデータ幅は 21 ビットから 26 ビット (usb_buf[](64 ビット)，c_sat_tbl[](最大 13 ビット)，v_tbl[](1 ビット) を除く) であり，32 ビットワードの SDRAM バンクを容易に用いることが可能である．ゆえに，32 ビットワードのメモリバンクを 4 つ使用した場合，処理の最大並列度 (SDRAM インタフェースにより供給されるワード数 L) は，16(または 32) となる (L はメモリインタフェースの動作周波数に依存する．FPGA 内部のテーブルへのアクセスコンフリクトを回避する必要があるため，実効値はこれよりも低くなる)．

9.3.3 データマッピングとハードウェアの詳細

Block RAM，および外部 SDRAM バンクへのデータマッピングについて述べる．

v_tbl[] のデータ幅は 1 ビットであり，このテーブルへのアクセスはランダムに行われる．1 ビット $\times$ 16K の Block RAM を 64 個用いることにより，1M 変数分の値を保持することができる．これにより，本ソルバーでは，[69] に挙げられているベンチマーク問題のうち，11,483,525 個の変数を持つ 1 題を除き，他の全ての問題を処理することが可能である．他のテーブルは規模が大きいため，その全てを FPGA 上に格納することはできない．しかしながら，それらテーブルのうち c_list_tbl[](このテーブルは read-only である) へのアクセスは連続的であるため (平均 75〜220 ワードが連続してアクセスされる)，c_list_tbl[] に対しては SDRAM のバーストリードが有効に機能する．そのため，c_list_tbl[] は外部 SDRAM バンク上への格納に適している．その他のテーブルのうち，c_tbl[]，c_arg_tbl[]，および ref_tbl[] もまた外部 SDRAM バンク上にマッピングされる．その理由は次の通りである．

1. ref_tbl[] へのアクセスが発生すると，続いて c_list_tbl[] へのアクセスが発生し，それにより 75〜220 個のクローズが読み出される．

2. c_arg_tbl[] からリテラルが読み出されると (c_tbl[] から読み出されたデータがアドレスとして用いられる)，読み出された各リテラルに対して上記の 1 が繰り返し実行される．

以上の理由により，c_tbl[]，c_arg_tbl[]，および ref_tbl[] へのアクセスにより発生するオーバヘッドは，SDRAM のバーストアクセスを用いることにより，比較的小さく抑えることができる．

c_sat_tbl[] は最も頻繁にアクセスされるテーブルであり，このテーブルへのアクセスはランダムに発生する．そのため，次のような手法を用いて c_sat_tbl[] へのアクセス遅延の隠蔽を行った．

1. 全クローズの大部分 (95%以上) を占めるリテラル数 3 以下のクローズに対してはクローズ番号は割り当てられず，クローズ内のリテラル番号の組をクローズ番号として使用する．例えば，c_list_tbl[] では，リテラル l を含むクローズのうち，リテラル数 3(または 2) のクローズは，l を除く残り 2 つ (または 1 つ) のリテラル番号が，クローズ番号として用いられる (これらのクローズに l が含まれているのは自明であるため，クローズ番号に l を入れる必要はない)．これらのクローズの値は，各リテラルの変数番号で v_tbl[] を参照し，読み出された値，および negation の有無からリテラル値を計算し，求められた各リテラル値の論理和を取ることにより算出される．そのため，リテラル数 3 以下のクローズの値は，c_sat_tbl[] には保持されない．

2. リテラル数 4 以上のクローズにはクローズ番号が割り当てられる．c_sat_tbl[] はクローズ番号を用いてアクセスされる．c_sat_tbl[] のエントリのうち，一部は FPGA の Block RAM に配置される．筆者らの評価では，より多くのリテラルを含むクローズほど，頻繁に参照される傾向にあった．そのため，全てのクローズをリテラル数によって予めソートし，c_sat_tbl[] のエントリのうち，より多くのリテラルを含むクローズに対するエントリを Block RAM に配置した．

usc_buf[] は非充足クローズを格納するバッファであり，そのごく一部が FPGA 上に置かれ，残りは外部 SDRAM バンクに置かれる．c_list_tbl[] の場合と同様に，リテラル数 3 以下のクローズに対しては，各クローズ内のリテラル番号の組がクローズ番号として用いられるが，c_list_tbl[] とは異なり，全てのリテラル番号が格納される．図 9.5 に usc_buf[] のデータ構造を示す．エントリの幅は 64 ビットであり，最上位 2 ビット (または最上位 1 ビット) は，リテラル数を識別するための tag フィールドとして使用され，他のフィールドはクローズ番号 (リテラル番号) として使用される．詳細は次の通りである．

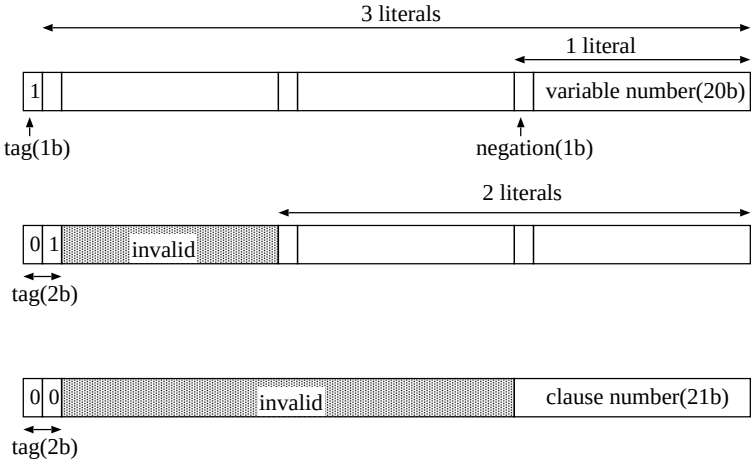

図 9.5: usc_buf[] のデータ構造

- 最上位 1 ビットが 1 であるエントリはリテラル数 3 のクローズを表し，他の 63 ビットで 3 つのリテラルを表す (1M 変数のとき，変数番号に 20 ビット，negation の有無に 1 ビットを必要とするため，1 リテラルにつき 21 ビット，3 リテラルで 63 ビットとなる).
- 最上位 2 ビットが 01 であるエントリはリテラル数 2 のクローズを表し，下位 42 ビットで 2 つのリテラルを表す.
- 最上位 2 ビットが 00 であるエントリはリテラル数 4 以上のクローズを表し，下位 21 ビットでクローズ番号を表す.

図 9.6 に，本ソルバーのブロック図を示す (外部 SDRAM バンク数は 4，バンクあたりのデータ幅は 32 ビットであり，本ブロック図では，回路の動作周波数は SDRAM のデータ転送速度の 1/4 である)．本ソルバーでは，4 つのメモリバンクを 1 つの大きなメモリバンクとして用いる (c_sat_tbl[] へのアクセスを除く．これについては後述する)．FPGA 上の c_sat_tbl[] は，図 9.6 の c_sat_tbl[] の上側に示したように，$16K \times 6$ ビットおよび $2K \times 18$ ビットのテーブルの組合せから構成される．この組合せは並列処理可能なワード数 L 分 (本ブロック図の場合は $L = 16$) 用意される．今回評価に用いたベンチマークでは，c_sat_tbl[] に必要とされるデータ幅の最大値は 13 ビットであるが，6 ビットを超えるデータ幅を必要とするクローズは，$32K$ 個未満であり，5 または 6 ビットのデータ幅を必要とするクローズは，$256K$ 個未満である．図 9.6 において，128 個の Block RAM が c_sat_tbl[] として使用される．更に 64 個の Block RAM ($16K \times 4$ ビット $\times 16$) を追加し，図 9.6 の

c_sat_tbl[] の下側に示すような構成とすることにより，今回評価に用いたベンチマークにおいては c_sat_tbl[] を全て Block RAM 上に置くことが可能となる．

図 9.6 のアーキテクチャでは，最大 16 ワードを並列に処理することができる．ゆえに，変数のフリップにより最大 16 個の非充足クローズが一度に生成され (平均値はこれよりもずっと小さくなる)，深さ 32 の FIFO に送られる．各 FIFO は多段接続網のように接続されており，これにより，それぞれの FIFO 内の非充足クローズの数が平滑化される．この FIFO の多段接続網からの出力は，Block RAM 上の usc_buf[] へ送られる．

回路の単純化のため，c_list_tbl[] には，データは 2 ワードバウンダリでのみ格納することが可能である (既に述べたように，リテラル数 3 のクローズは 2 つのリテラル番号で表現される)．また，使用されないエントリも存在する．

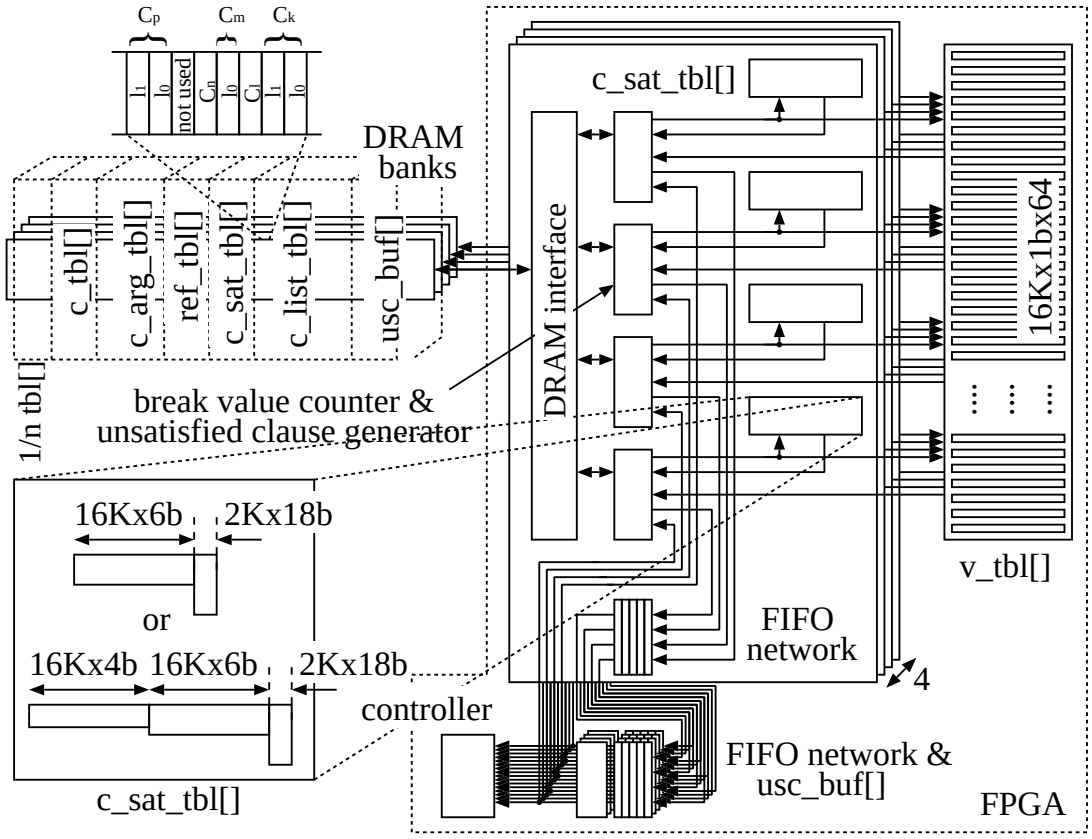

図 9.6: 形式検証用ソルバーのブロック図

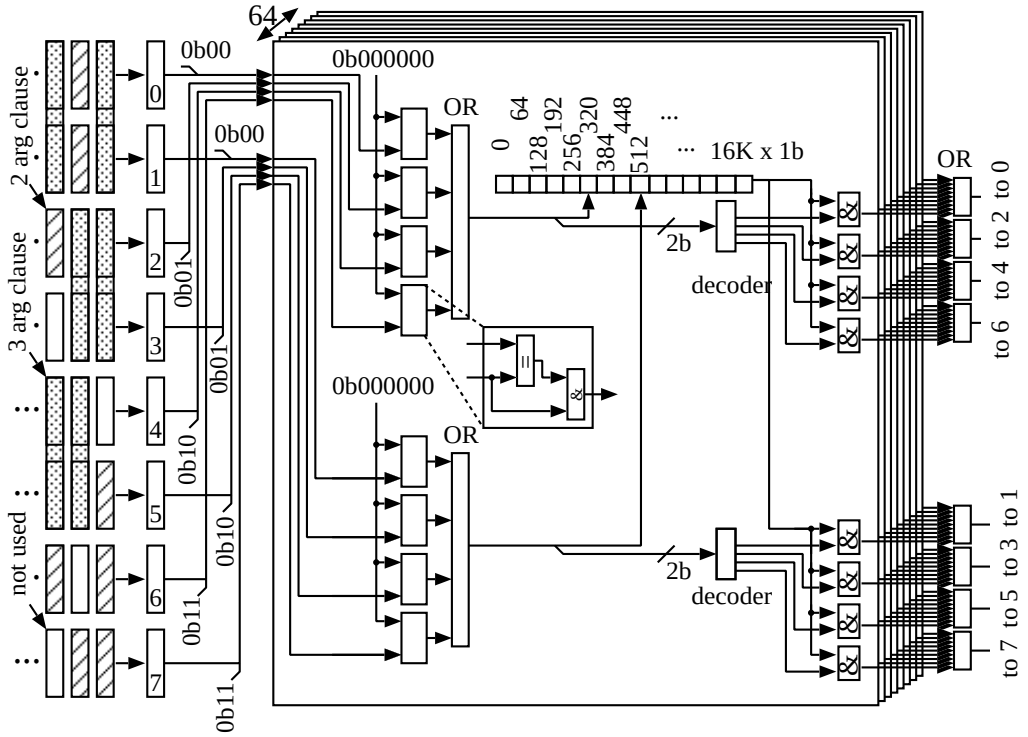

図 9.7: v_tbl[] の詳細

9.3.4 並列処理の詳細

本ソルバーではメモリアクセス制御を簡略化するため，全ての外部メモリバンクをひとつの大きなメモリバンクとみなしてアクセスする．ゆえに，本ソルバーの処理の並列度は，主に *clauses list* へのアクセスにより制限される．*clauses list* へのアクセスは，クローズ内のリテラル数+2 回，繰り返し実行される．この操作は本ソルバーにおける全計算時間の大部分を占める．

clauses list 内のクローズの列は，次の 3 つにより構成される．

1. リテラル数 2 および 3 のクローズ
2. c_sat_tbl[] が Block RAM 上に置かれるクローズ (*cached clause* と呼ぶ)
3. c_sat_tbl[] が外部 SDRAM バンク上に置かれるクローズ (*long clause* と呼ぶ)

本ソルバーでは，SDRAM インタフェースから供給される L ワードのデータ列 ($L = 16$ または 32．以下，*L-words* と呼ぶ) は並列に処理される．しかしながら，Block RAM 上のテーブルへのアクセスコンフリクトを回避しなければならないた

め，常に最大並列度で処理を行うことができるわけではない．処理の並列度をできる限り高くするため，プリプロセッサを用いて読み出されるクローズの順序のスケジューリングが行われる．c_list_tbl[] において，クローズを表現するために必要とされるワード数は，リテラル数 3 のクローズの場合は 2，リテラル数 2 の場合は 1 である．*L-words* 内の各スロットに，まずリテラル数 3 のクローズが 2 ワードバウンダリで配置される．同一の *L-words* に格納すると v_tbl[] へのアクセスコンフリクトを起こすクローズ同士は，異なる *L-words* に格納される．同一の *L-words* に配置することのできる，リテラル数 3 のクローズがない場合には，リテラル数 2 のクローズが配置される．

図 9.7 に，$L = 8$ のときの v_tbl[] の詳細を示す (図の簡略化のため，$L = 8$ とした)．図 9.7 において，*L-words* 内の各データは，その配置場所に従って 2 つのグループ (奇数グループおよび偶数グループ) に分かれ，v_tbl[] を構成する全ての Block RAM にそれぞれブロードキャストされる．ここで，リテラル数 3 のクローズ内の 2 つのリテラル同士は，常に異なるグループに割り当てられ，その値は Block RAM の dual port access を用いて同時に計算される．v_tbl[] へのアクセスコンフリクトを回避するため，同一の Block RAM にアクセスするリテラルを持ったクローズは異なる *L-words* に配置される．クローズのスケジューリングに際して，クローズ間のデータ依存性は存在しないため，クローズ数が十分大きい問題では，*L-words* の大部分を埋めることが可能である．リテラル数 3 のクローズの場合，最大 $L/2$ 個が並列に処理され，リテラル数 2 のクローズの場合は，最大 L 個が並列に処理される．

図 9.6 に示したように，外部 SDRAM バンクと FPGA 上の c_sat_tbl[] との接続関係は固定されている．そのため，c_list_tbl[] 内の各クローズは，c_sat_tbl[] を構成する Block RAM のうち，特定のものにのみアクセスすることが可能である．c_sat_tbl[] に正しくアクセスすることができるようにするため，*cached clause* は，c_list_tbl[] において，*L-words* 内の常に同一の位置に配置されるように，格納される．*cached clause* における，並列に処理可能なクローズの個数の最大値は L である．

long clause の各クローズに対する c_sat_tbl[] の値は外部 SDRAM バンク上に配置される．これらのクローズについては，c_sat_tbl[] のベースアドレスにクローズ番号を加算した値をアドレスとして，外部 SDRAM バンクから値が読み出される．各クローズに対する c_sat_tbl[] のエントリは，外部 SDRAM バンクのいずれかひとつに格納される．バンク数 4，クローズ c_l に対するエントリが i 番目の外部 SDRAM バンクに格納されている場合，c_l は，*L-words* 内の $(4k + i - 1)$ 番目に配置される $(k = 0, 1, ..., L/4 - 1)$．

以上のスケジューリングは，ホストコンピュータ上でプリプロセッサにより予め実行される．

9.4 性能に関する考察

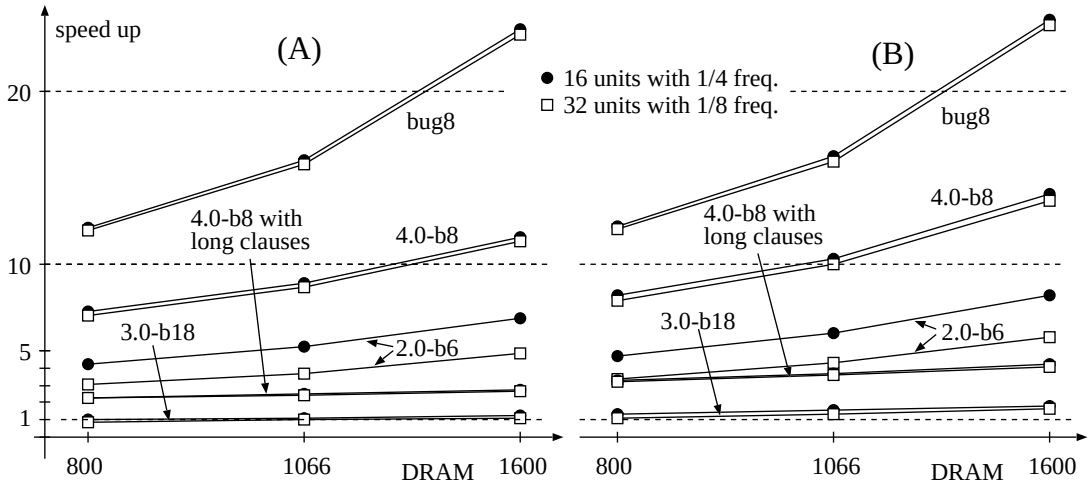

図 9.8: ソフトウェアに対する速度向上率

Xilinx 社の Virtex-4 シリーズ FPGA[50][51] を用いた場合，本ソルバーの実装に必要となる Block RAM 数は，225($L = 16$) または 239 個 ($L = 32$) である (v_tbl[] に 64 個，c_sat_tbl[] に 128 個，*break-value* の保持に 7 個，c_arg_tbl[] に 11 個，$1/m$ Table に 1 個，usc_buf[] に 14($L = 16$) または 28 個 ($L = 32$)，それぞれ必要となる)．必要とされる LUT の数は，L にほぼ比例する．$L = 32$ のとき，およそ $90K$ 個の LUT が必要となる (usc_buf[] に非充足クローズを格納するための FIFO の多段接続網に $22K$ 個，v_tbl[] を構成する各 Block RAM に並列にアクセスするための回路に $45K$ 個が必要となる)．以上の Block RAM 数，および LUT 数は，今日の FPGA のハードウェアリソース量 [50][51] に対して妥当な数量である．

次の 3 通りの外部 DDR3 SDRAM のスループットにおいて，メモリアクセス遅延，L(並列処理可能なワード数) をパラメータとし，シミュレーションによる性能評価を行った．

1. DDR-800(ADM-XRC-6T1 に搭載されている DDR3 SDRAM のスループット [52])

2. DDR-1066(Virtex-6 シリーズ FPGA で実装可能な DDR3 SDRAM インタフェースのスループットの最大値 [53])

3. DDR-1600(今回の評価に用いたソフトウェアの動作プラットフォームであるホスト PC における DDR3 SDRAM のスループット [54])

SDRAM には内部バンクが存在するため，バーストリードが継続する場合には，アイドル時間なしにデータを読み出すことができる．回路の動作周波数は，外部 DDR3 SDRAM のデータ転送速度の 1/4 または 1/8(100MHz から 400MHz) である．1/4 の場合，4 つの外部 SDRAM バンクから 16 ワードが ($L = 16$)，1/8 の場合，外部メモリバンクから 32 ワードが ($L = 32$)，並列に読み出されて処理される．

図 9.8 に Core i7-920 2.79GHz，メインメモリ 4GB に対する速度向上率を示す．(A) は，DDR3 SDRAM インタフェースの遅延時間 (アドレス生成とデータの受け取り) が 100nsec，(B) は，遅延時間 50nsec の場合の速度向上率のグラフである．評価には，1sec あたりのフリップ回数 (fps) を用いた (それぞれ 10 回の試行の平均値である)．これは，解の発見に要する全実行時間が乱数により大きく変動するため，これによる正確な性能比較が困難である一方，fps は乱数の影響をそれほど大きく受けないためである．図 9.8 は，横軸に DDR3 SDRAM のスループットを，縦軸に速度向上率をとり，それぞれのベンチマークにおける，DDR-800，DDR-1066，DDR-1600(回路の動作周波数は，$L = 16$ のとき，200MHz，266MHz，400MHz，$L = 32$ のとき，100MHz，133MHz，200MHz) の場合の速度向上率をプロットしたものである．2.0-b6 を除き，回路の動作周波数が DDR3 SDRAM のデータ転送速度の 1/4，1/8 のいずれの場合もほぼ同等の速度向上率となった．2.0-b6 において速度向上率に差が見られた理由は定かではないが，幾つかのリテラルに対して，*clauses list* があまり効率良くスケジューリングされていないこと，およびそれらの *clauses list* へのアクセスが頻発しているためと考えられる．

処理の性能は，問題によって有意な差が見られた．最も小規模な問題である 3.0-b18 においては，スルーレートが DDR-800 で，メモリアクセス遅延が 100ns のとき，本ソルバーの処理性能はソフトウェアの処理性能を下回った．本ソルバーの処理性能が最も高かったベンチマークは bug8 であり，DDR3-1600 で，FPGA の動作周波数が 200MHz(32 並列) のとき，ソフトウェアに対する速度向上率は 23.8 倍となった．DDR-1066 で，FPGA の動作周波数が 266MHz(16 並列) の場合には，bug8 の速度向上率は 16.0 倍であった．図 9.8 において，'with long clauses' は，4.0-b8 において，c_sat_tbl[] の一部が外部 SDRAM バンクに格納される場合の速度向上率を示している (この場合，c_sat_tbl[] のほぼ半数が外部 SDRAM に置かれる．4.0-b8 に対して c_sat_tbl[] を全て FPGA 上に配置するためには，更に 64 個の Block RAM が必要である)．他のベンチマークにおいては，c_sat_tbl[] は全て Block RAM に置くことができる．c_sat_tbl[] の一部が外部 SDRAM バンクに置かれることにより，速度向上率は大きく低下する．

表 9.3: バーストリードの割合，およびクローズの平均リテラル数と *clauses list* 長さの平均

	#units&freq	3.0-b18	bug8	2.0-b6	4.0-b8
burst ratio	32&1/8	0.561	0.992	0.892	0.896
	16&1/4	0.696	0.996	0.942	0.943
ave.num.of.args		10.6	41.4	12.2	77.1
ave.len.of.c_list		115.0	16012.4	1113.1	881.4

表 9.3 に，全実行時間に占める，DDR3 SDRAM バンクをバーストリードしている時間の割合を示す．bug8 においては，全実行時間に占める DDR3 SDRAM バンクのバーストリードの時間がほぼ 100%に達している．これが，bug8 において最も高い速度向上率を示している理由である．表 9.3 に示した通り，bug8 における，回路上で実際に評価されるクローズの平均リテラル数は 41.4 であり (全てのクローズの平均リテラル数は 2.98 である)，回路上で実際に評価されるリテラルの *clauses list* の平均の長さは 16012.4(全ての *clauses list* の平均の長さは 221.9) である．ゆえに，変数のフリップに際して 41.4 × 16912.4 クローズがバーストリードされる．3.0-b18 においては全実行時間に占める DDR3 SDRAM のバーストリード時間の割合は低いため，メモリアクセス遅延が大きい場合には，処理の性能はソフトウェアと比較して低くなる．これが，bug8 において DDR3 SDRAM のスループットが上がることによって高い性能が得られる一方，3.0-b18 においてはそのような傾向が見られないことの理由である．

図 9.8(B) に示したように，メモリインタフェース遅延 (メモリアクセス遅延) が小さくなることによる性能の向上はあまり見られない．bug8 では，全実行時間の大部分がバーストリードの実行時間であり，メモリアクセス遅延の改善による性能向上率は，わずか 1.4%である．3.0-b18 の場合は 40%の性能向上が見られたが，それでもソフトウェアと比較してわずかに速い程度である．メモリアクセス遅延が小さくなることにより最も性能が向上するのは，4.0-b8 'with long clauses' であり，およそ 50%の性能向上が見られた (しかしながら，ソフトウェアに対する速度向上率は，DDR3-1066 の場合で 2.5 倍から 3.7 倍に上昇したに過ぎない)．一方，4.0-b8 において c_sat_tbl[] が全て Block RAM に置かれた場合は，メモリアクセス遅延が小さくなることによる性能向上率は，15%であった．メモリアクセス遅延の削減による性能向上は，ソフトウェアに対してそれほど高い速度向上率が得られていない問題に対してはある程度効果的であるのに対し，高い速度向上率が得られている問題に対してはあまり効果的ではない．これは，SDRAM を用いた場合に高い性能を得るためには，非常に高い割合でバーストアクセスを行う必要が

あり，そのような場合には，メモリアクセス遅延はあまり影響を与えないためである．

以上の議論において，ホストコンピュータ上での処理時間と FPGA へのダウンロード時間は含まれていない (それぞれ 0.30，8.72，9.47，15.8sec である)．これは，これらのオーバヘッドの全実行時間に占める割合が小さく，無視しても殆ど影響がないためである．

第10章　各実装方式の処理性能に関する考察

表 10.1 に，本論文において提案するソルバーにおける処理可能な問題規模，メモリアクセス速度，および処理性能を示す．小規模な問題であれば，全てのデータを FPGA 上に保持することにより，処理の並列度を最大化することが可能であり，最も高い処理性能を実現することができる．一方，問題が大規模になるにつれ，外部メモリにより多くのデータを置かなければならなくなり，処理性能は外部メモリのデータ転送速度により制限される．本章では，提案した FPGA ソルバーにおけるメモリアクセス速度と処理性能，および問題規模と処理性能との関係について考察する．

表 10.1: 問題規模/メモリアクセス速度と処理性能

ソルバー	(N_v, N_c)	外部メモリ	メモリアクセス速度 (Gb/sec)	処理性能 (Mfps)
3-SAT 用 1	(256,2048)or (4096,1792)	なし	-	～21
3-SAT 用 2	(2048,8500)	なし	40.9Gb/sec	～28
3-SAT 用 3	(32K,128K)	SSRAM 40MB	17.2Gb/sec	～19.2
3-SAT 用 4	(128K,512K)	SSRAM 40MB	12.7Gb/sec	～5.4(estimated)
形式検証用	(1M,16M)(estimated)	DDR3 SDRAM 2GB	～136.4Gb/sec[1] (問題により様々)	メモリアクセス速度の実効値による

10.1　メモリアクセス速度と処理性能

3-SAT 用ソルバー 1 では，Block RAM は非充足クローズ中のリテラルの読み出し，および変数の現在値の保持のみに使用され，Block RAM へのアクセス速度が

[1]スループット DDR-1066 のときの最大値

処理性能を制限することはない．また，外部メモリバンクは使用しない．ゆえに，ソルバー 1 において，メモリアクセス速度が処理性能を制限することはない．

3-SAT 用ソルバー 2 では，Address Translation Table，Clause Table，$1/m$ Table (非充足クローズの選択のために使用する乱数を生成するためのテーブル)，および非充足クローズバッファに Block RAM を使用している．このうち処理の並列度を制限するのは Clause Table である．$Nc/Nv \fallingdotseq 4.3$ の 3-SAT 問題において，クローズ評価の最大並列度は 20 であり，そのときのデータ幅は $(1+\lceil \log_2 N_v \rceil) \times 2 \times 20$ ビットである．ゆえに，完全にパイプライン化された回路においては，その動作周波数を f とおくと，最大並列度を実現するために必要とされるメモリアクセス速度は，$(f \times (1 + \lceil \log_2 N_v \rceil) \times 2 \times 20)$[ビット/sec] となる．外部メモリバンクを使用しないソルバー 2 においては，Clause Table は割り当てられた Block RAM の容量の範囲内で任意のデータ幅を取ることができる．ゆえに，ソルバー 2 ではクローズ評価の最大並列度を実現することが可能であり，ソルバー 1 と同様，メモリアクセス速度が処理性能を制限することはない．

3-SAT 用ソルバー 3 では，Clause Table，および非充足クローズバッファのそれぞれ一部を Block RAM にキャッシュし，残りを外部メモリバンクに配置することにより，処理可能な問題規模の拡大を図っている．ソルバー 2 同様，処理の並列度は Clause Table により制限される．Clause Table のデータ幅は外部メモリバンクのデータ幅 W_{bank} およびバンク数 N_{bank} により制限され，$W_{bank} \times N_{bank}$ となる (Block RAM を併用し，Clause Table のデータ幅を補うことにより最大並列度を実現することができるが，Variable Table に用いることのできる Block RAM が不足するため，そのような構成を取ることはできない)．このとき，$(1 + \lceil \log_2 N_v \rceil) \times 2 \times 20$ ビット のデータを読み出すのに必要なクロックサイクル数は，$(1 + \lceil \log_2 N_v \rceil) \times 2 \times 20/(W_{bank} \times N_{bank})$ となり，総パイプライン段数の増加と，それによる処理性能の低下を招く．

3-SAT 用ソルバー 4 では，Clause Table は全て外部メモリバンクに配置される．Clause Table のデータ幅は，Clause Table に割り当てることのできるメモリバンクの数 $N_{avl}(\leq N_{bank})$ およびメモリバンクのデータ幅 W_{bank} より，$N_{avl} \times W_{bank}$ となる．$(1 + \lceil \log_2 N_v \rceil) \times 2 \times 20$ ビット のデータを読み出すのに必要なクロックサイクル数は，$(1 + \lceil \log_2 N_v \rceil) \times 2 \times 20/(W_{bank} \times N_{avl})$ となり，総パイプライン段数の更なる増加と，いっそうの性能低下を招く．

形式検証用ソルバーでは，外部メモリとして DDR3 SDRAM を用いている．DDR3 SDRAM のデータ転送速度は，回路の動作周波数の 4 倍または 8 倍となる．そのため，データアクセス速度の最大値は非常に高速であるが，実効値は全メモリアクセスに占めるバーストアクセスの割合によって大きく変化する．形式

検証用ソルバーにおける実効アクセス速度と処理性能の関係については，9 章 9.4 節を参照されたい．

10.2 問題規模と処理性能

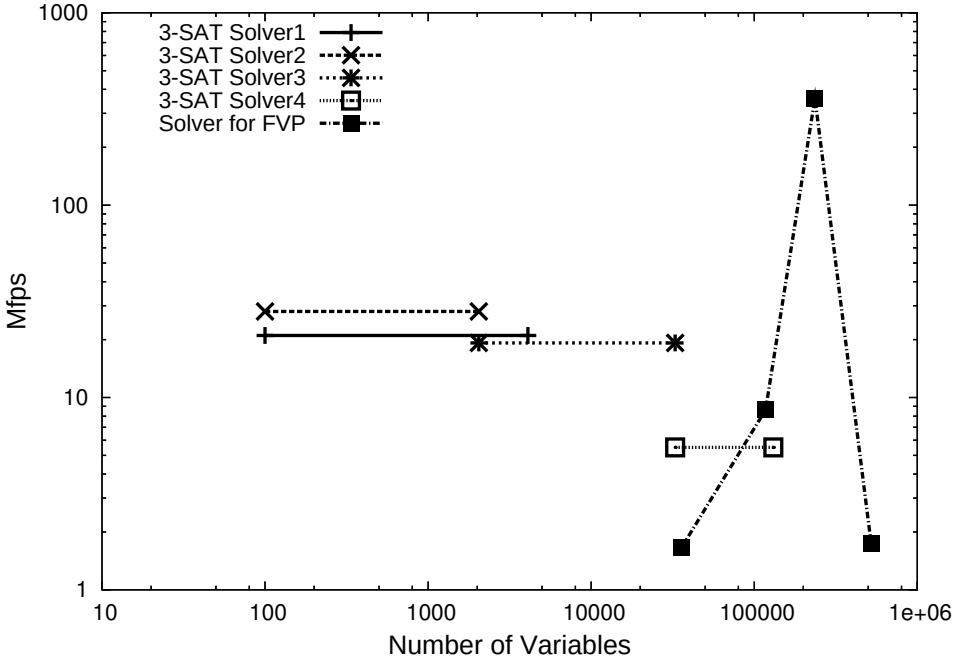

図 10.1: 変数の数と処理性能

図 10.1，図 10.2 に，3-SAT 用ソルバー 1〜4，および形式検証用ソルバー (Solver for FVP) における変数の数と処理性能，およびクローズ数と処理性能の関係を示す[2]．

3-SAT 用ソルバー 1 では，処理可能な問題規模はクローズ評価モジュールに用いるのことのできる LUT の数で決まる．変数の数を N_v，クローズの数を N_c とすると，クローズ評価モジュールに必要な LUT の数は，$N_c \times (16 + 3/4 \times \log_2 N_v)$ となり，ほぼ N_c に比例する．

一方，$N_c/N_v \leq 4.3$ の 3-SAT 問題に限定することにより，3-SAT 問題用ソルバー 2 を用いて，より大規模な問題を処理することができる．図 10.3 に，ソルバー 1 および 2 における，Nv と LUT 数の関係を示す ($N_c/N_v \fallingdotseq 4.3$ の場合)．ソ

[2]形式検証用ソルバーについては，それぞれ DDR-1066 の場合の性能であり，9 章 9.4 節における評価で用いた 4 つのベンチマークにおける値である．

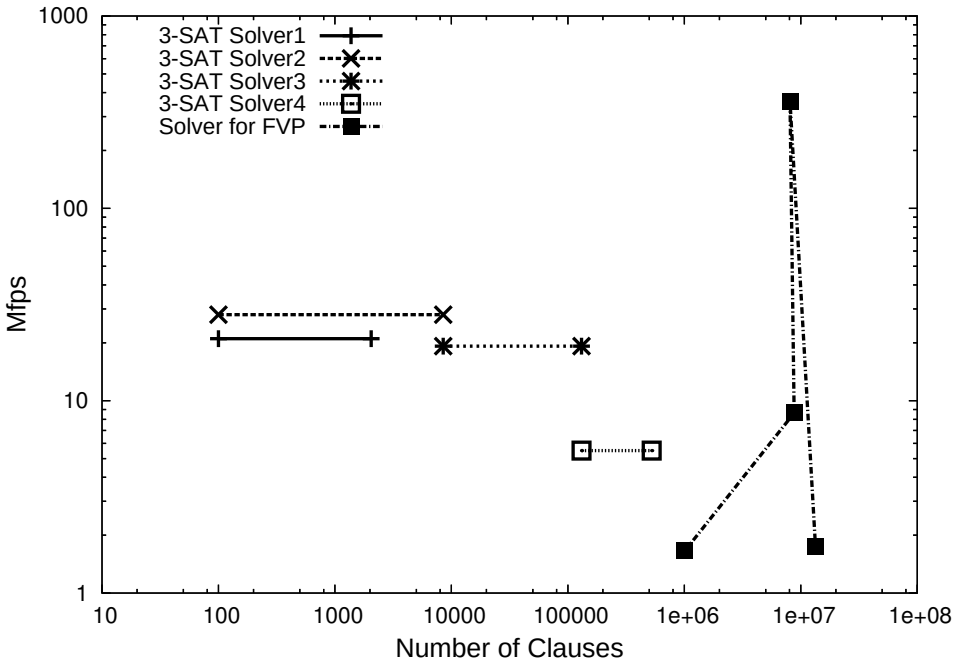

図 10.2: クローズの数と処理性能

ルバー 1 において，LUT は主にクローズ評価モジュールにより消費され，その量は $N_v \times \log_2 N_v (N_v \fallingdotseq N_c/4.3)$ にほぼ比例する．これに対して，ソルバー 2 では，LUT は主に Temporal Buffer と FIFO Tree により消費される．これらの回路規模は，リテラルのデータ幅である $1 + \log_2 N_v$ にほぼ比例する．ゆえに，ソルバー 2 においては，処理可能な問題規模に対する LUT 消費量の影響は少ない．ソルバー 3 および 4 も同様である．形式検証用ソルバーにおいては，LUT 使用量は問題規模に拘らず固定である．

図 10.4 に，ソルバー 2〜4，および形式検証用ソルバーにおける，N_v と Block RAM 数の関係[3]を示す．データを全て FPGA 上に置くソルバー 2 における Block RAM 使用率が最も高く，問題規模が大きくなり，外部メモリを使用する割合が大きくなるに従い，Block RAM 使用率は低下する．形式検証用ソルバーにおいては，Block RAM の使用量は問題規模によらず固定である．

3-SAT 問題用ソルバー 2 では，Variable Table，Address Translation Table と Clause Table，および非充足クローズバッファに Block RAM を使用しており，処理可能な問題規模はこれらのテーブルおよびバッファに用いることのできる Block

[3]Block RAM 1 つあたりの容量は 16K としている．

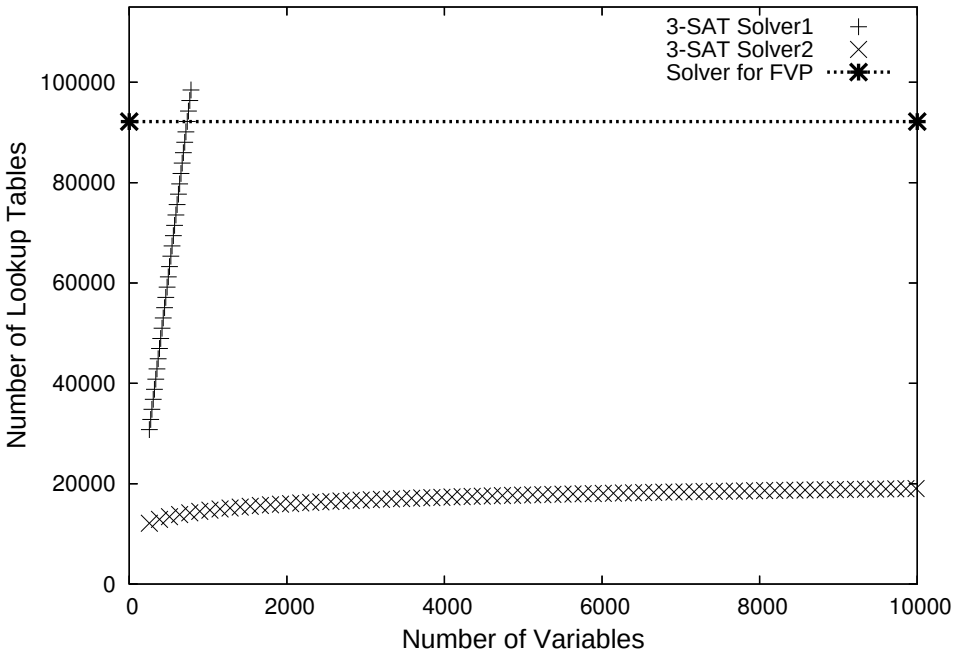

図 10.3: 変数の数と LUT 数の関係

RAM の数により制限される．Variable Table の構成に必要な Block RAM 数 $N_{B_{vtbl}}$ は，N_v，スレッド数 N_{th}，およびクローズ評価の最大並列度 P_{max} によって決まり，$N_{B_{vtbl}} = \lceil N_{th} \times N_v/16K \rceil \times 2 \times P_{max}$ となる．Address Translation Table の構成に必要な Block RAM 数 $N_{B_{at}}$ は，N_v，P_{max}，および Clause Table の深さ D によって決まり，$N_{B_{at}} = \lceil (P_{max} + \log_2 D) \times N_v \times 2/16K \rceil$ となる．D は N_v，P_{max}，およびクローズ評価の平均並列度 P_{avg} により決まり，$D = \lceil 2 \times N_v \times P_{avg}/P_{max} \rceil$ となる．Clause Table の構成に必要な Block RAM 数 $N_{B_{ct}}$ は，P_{max}，D，および N_v によって決まり，$N_c/N_v \fallingdotseq 4.3$ のとき，$N_{B_{ct}} = \lceil D \times (1 + \log_2 N_v) \times 2 \times P_{max}/16K \rceil$ となる．非充足クローズバッファに必要とされる Block RAM$N_{B_{ucb}}$ は，$N_{B_{ucb}} = \lceil N_{th} \times (1 + \log_2 N_v) \times 3 \times N_c/2^3/16K \rceil$ となる．

$N_{B_{vtbl}} + N_{B_{at}} + N_{B_{ct}} + N_{B_{ucb}}$ が Block RAM の総数を上回るときは，全てのデータを FPGA 上に置くことはできず，データの一部を外部メモリバンクに置く必要がある．ソルバー 3 では，Clause Table および非充足クローズバッファのうち，それぞれ一部を外部 SSRAM バンクに置くことにより，$N_{B_{at}}$ および $N_{B_{ct}}$ による制限が緩和されるため，ソルバー 2 と比較して，より大規模な問題を処理可能となる．一方で，$N_{B_{vtbl}}$ の増加を抑制しなければならなくなるため，処理の並列度を最大化することはできず，その性能は低下する．$N_{B_{vtbl}}$ は N_v，N_{th}，およびクロー

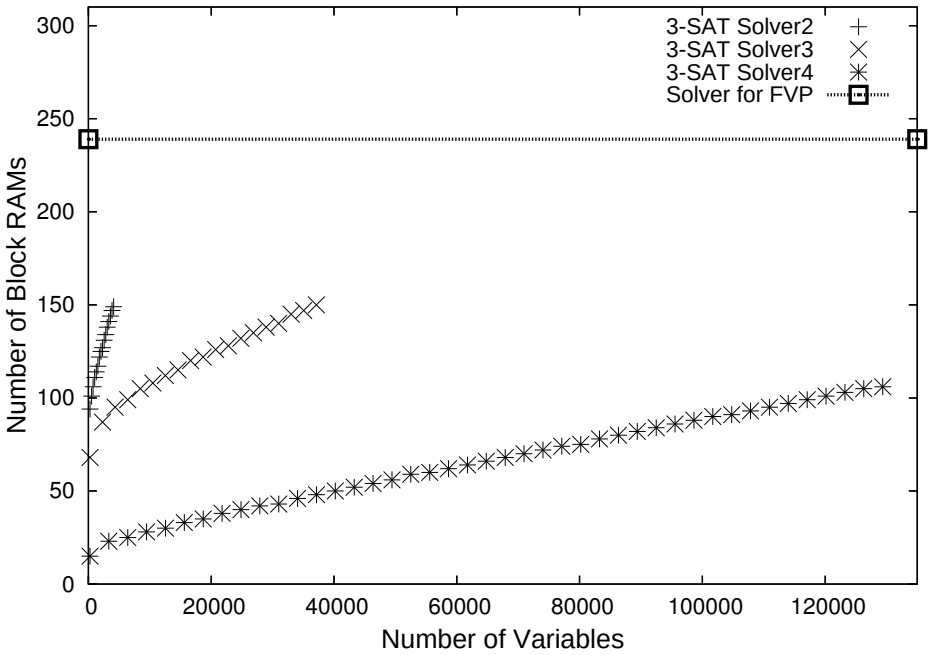

図 10.4: 変数の数と Block RAM 数の関係

ズ評価の並列度 P によって決まり，$N_{B_{vtbl}} = \lceil N_{th} \times N_v \times P/16K \rceil$ となる．

3-SAT 用ソルバー 4 では，Clause Table を全て外部 SSRAM バンクに配置している．これにより $N_{B_{at}}$ による制限がなくなること，また，ソルバー 3 と同様に $N_{B_{ct}}$ による制限が緩和されることから，より大規模な問題を処理可能となる．このとき処理可能な問題規模は，$N_{B_{vtbl}}$，および外部メモリバンクの容量により決まる．$N_{B_{vtbl}}$ はソルバー 3 と同様である．外部メモリバンクは主に Clause Table によって消費され，その消費量は $P_{max} \times N_v \times \log_2 N_v$ にほぼ比例する．一方，前節で考察したように，処理の並列度は外部 SSRAM バンクのアクセス速度により制限され，性能は更に低下する．

実応用問題から生成される大規模な問題を処理する場合には SDRAM を用いてデータを保持する必要があり，そのときの性能は，SDRAM の実効データ転送速度により決まる．9 章 9.4 節において考察したように，実応用問題から生成される SAT 問題では，リテラル数の非常に大きいクローズや，非常に多くのクローズに含まれるリテラルが存在し，それらに起因する参照の局所性が，SDRAM の実効データ転送速度を上昇させる要因となる．一方，そのような特徴を持たない 3-SAT 問題の場合は SDRAM の実効データ転送速度を上昇させることはできず，処理の性能は大幅に低下するものと考えられる．

第11章　結論

11.1　まとめ

本論文では，FPGA を用いた充足可能性問題の高速計算システム (FPGA ソルバー) について述べた．SAT ソルバーの構築にあたっては，効率の良いアルゴリズムの開発だけでなく，アルゴリズムを高速に実行可能なハードウェアの開発が重要である．Walksat アルゴリズムは，充足不可能性を示すことはできない “incomplete” なアルゴリズムであるが，充足可能な SAT 問題に対して高い探索性能を持っている．また，複雑な制御を必要とせず，処理の並列度が高いことから，ハードウェア化に適している．データを全て FPGA 上に保持することができるような小規模な問題においては，アルゴリズムの有効並列度を最大限に引き出すことにより，最も高い処理性能を実現することができる．一方，問題規模が大きくなるにつれ，外部メモリ上に，より多くのデータを配置せざるを得なくなるため，処理の性能は，外部メモリのデータ転送速度により制限される．

本論文では，そのような制約のもと，1 チップの FPGA および複数の外部メモリバンクが搭載された FPGA ボードとホストコンピュータから構成されるシステムにおいて，Walksat アルゴリズムに基づく 3-SAT 問題用ソルバー，および Walksat の改良アルゴリズムに基づく形式検証問題用ソルバーの提案をそれぞれ行った．

3-SAT 問題用ソルバーにおいては，外部 SSRAM バンクを搭載した FPGA ボードにおいて，問題規模に応じ，FPGA のみを使用した異なる 2 つの回路方式と，外部 SSRAM バンクを使用した異なる 2 つの回路方式をそれぞれ提案し，どのような回路方式で，どの程度の規模の問題を処理することができ，その際どの程度の性能を実現可能であるかを明らかにした．

形式検証問題用ソルバーにおいては，SAT の実応用問題のひとつであり，3-SAT 問題を大きく上回る規模を持った形式検証問題を対象とし，まず Walksat アルゴリズムの改良を行った．そして，FPGA および外部 SDRAM バンクを搭載した FPGA ボードにおいて，本改良アルゴリズムに基づく回路方式の検討を行い，SDRAM バンクのアクセス遅延とスループットの制約のもとで，どの程度までの規模の問題を処理可能であり，その際どの程度の処理性能を実現可能であるかを明らかに

した．

その結果，提案手法により，従来のFPGAソルバーで扱える範囲の問題において，更なる高速化を実現するとともに，従来のFPGAソルバーでは扱うことのできなかった大規模な問題を高速に処理可能であることを示し，それとともに，FPGAは，様々な規模の3-SAT問題，および形式検証問題において，ソフトウェアに対して高い処理性能を実現可能であることを示した．

11.2 今後の課題

11.2.1 アルゴリズムの改良

9章9.1節において述べたように，実応用問題から生成されるSAT問題には，もとの問題に内在する構造が存在し，そのような内部構造を利用することにより，アルゴリズムの処理性能を大幅に向上させることが可能である[78]．

本研究では，応用問題のひとつであるハードウェアの形式検証問題から生成されたSAT問題の持つ構造に着目し，Walksatアルゴリズムの改良を行った．これにより，Walksatアルゴリズムの高い並列性を落とすことなく，形式検証における処理の性能を向上させることができ，9章で述べたように，FPGAソルバーにおいて大規模な形式検証問題を処理することが可能であることを示した．

本改良アルゴリズムは，論理ゲートのCNF表現形式から論理ゲートとその入出力関係を検出し，その結果から，クローズ内のどのリテラルがどの論理ゲートの出力信号に相当するか特定しているが，この方法ではNOTゲートやXORゲートを検出することはできない．これらの論理ゲートを検出するためには，検出された論理ゲートとその入出力関係から，まだ検出されていない論理ゲートとその入出力関係をトラッキング等により検出する必要があり，これは今後の課題のひとつである．

また，非充足クローズの中から，どのリテラルがどの論理ゲートの出力信号に相当するか特定できなかったクローズが選択された場合，そのクローズの中から出力信号に相当する変数を選択することはできない．このような場合における変数選択ヒューリスティックの検討も，今後の課題のひとつである．

11.2.2 より効率的なデータキャッシュ方法についての検討

大規模な問題においてはSDRAMの使用は不可避である．DDR3 SDRAM等，最新のSDRAMでは，バーストアクセスによるデータ転送速度は高く，バースト

アクセスの割合を高めることができる場合には，データ転送速度の実効値は高くなる．その一方，ランダムアクセスが頻繁に発生する場合には，その高いデータ転送速度を利用することはできない．

9章9.4節で述べたように，本研究による形式検証用ソルバーにおいては，c_sat_tbl[]を全てFPGA上にマッピングした場合と，一部を外部SDRAMに配置した場合とで，処理の性能に大きな差が見られた．また，より多くのリテラルを含むクローズが頻繁に選択される傾向にあるなど，データ参照の局所性も存在した．既に述べたように，実際の応用問題から生成されるSAT問題は，もともとの問題における内部構造を保持しているため，頻繁に参照されるクローズや変数にも局所性が存在すると考えられる．ゆえに，そのような参照の局所性等を考慮し，バーストアクセスによる高いデータ転送速度を実現できるよう，より効率的なデータキャッシュ方法の検討を行う必要がある．

11.2.3 形式検証以外の応用問題に対する回路の検討

2章2.3節において述べたように，充足可能性問題は，形式検証の他にもテストパターン自動生成，プランニング，スケジューリング，FPGA Routingなど，多くの応用問題と等価である．形式検証以外の応用問題に対する回路の検討を行うことにより，充足可能性問題と等価である様々な応用問題における高速計算が可能になるとともに，実応用問題に対するFPGAの有用性がいっそう高まることが期待される．

謝辞

筑波大学システム情報系・知能機能工学域 教授の丸山勉博士には，工学システム学類 4 年次の研究室配属以来，博士課程修了までの間，指導教員として研究全般に渡りご指導，ご鞭撻をいただきました．大変興味深い研究テーマとそれを探究する機会を与えていただき，計算機工学全般における先生の深い御見識の一端に触れるとともに，問題の発見とその解決のためのものの見方や考え方について学ばせていただきました．ここに深く感謝致します．

筑波大学システム情報系知能機能工学域 教授の白川友紀博士，同・講師の延原肇博士，筑波大学システム情報系情報工学域 教授の山口喜教博士，同・講師の山口佳樹博士には，本論文の審査を快くお引き受けくださるとともに，本研究に対する数々の貴重なご助言を賜りました．深く感謝致します．また，山口佳樹博士には繰り返し囚人のジレンマゲームの高速計算の研究に携わる機会を与えていただくなど，本研究以外の面においても大変お世話になりました．深く感謝致します．

博士課程在籍の間，丸山研究室の皆様には大変お世話になりました．ここに感謝致します．

博士課程への進学，および研究への従事にあたっては，ブラザー工業株式会社・電子システム開発部長の鈴木祥五氏，同・プロジェクトマネージャーの杉本輔氏，元・ブラザー工業株式会社・第 1 開発部長で，現在は兄弟高科技 (深セン) 有限公司・総経理の桑原悟氏，他多くの関係者の皆様より御理解・御協力をいただきました．心より御礼申し上げます．

最後に，支えてくださった両親，兄，親戚に深く感謝致します．

参考文献

[1] Stephen A. Cook, “The Complexity of Theorem-Proving Procedures”, 3rd Annual ACM Symposium on Theory of Computing, pp.151–158, 1971

[2] João P. Marques-Silva and Karem A. Sakallah, “Boolean Satisfiability in Electronic Design Automation”, 37th Design Automation Conference, pp. 675–680, 2000

[3] 藤田 昌宏，“[特別講演]SAT アルゴリズムとその形式的検証への応用”，情報処理学会研究報告，2006-SLDM-127，pp. 103–108，2006

[4] 藤田 昌宏，“VLSI 設計工学 SoC における設計からハードウェアまで”，数理工学社，2009

[5] 藤田 昌宏，“システム LSI 検証技術入門 第 19 回 (最終回) 形式的検証技術の現状と今後の動向”，デザインウェーブマガジン 2007 年 2 月号，pp. 151–156

[6] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu, “Symbolic Model Checking without BDDs”, 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 193–207, 1999

[7] 宮下 和雄，“プランニングとスケジューリング”，人工知能学会誌，16 巻，第 5 号，pp. 611–616，2001

[8] Stuart J. Russell and Peter Norvig(古川 康一 監訳)，“エージェントアプローチ人工知能 第 2 版”，共立出版，2008

[9] Bart Selman, David G. Mitchell, and Hector J. Levesque, “Generating Hard Satisfiability Problems”, Artificial Intelligence, Volume 81, Issue 1–2, pp. 17–29, 1996

[10] Jun Gu, Paul W. Purdom, John Franco, and W. Wah, “Algorithms for the Satisfiablity(SAT) Problem: A Survey”, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 35, pp. 19–152, 1996

[11] James M. Crawford and Andrew B. Baker, “Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems”, 12th National Conference on Artificial Intelligence, pp. 1092–1097, 1994

[12] 人工知能学会誌，25 巻，第 1 号，2010

[13] 井上 克己，田村 直之，“SAT ソルバーの基礎”，人工知能学会誌，25 巻，第 1 号，pp. 57–67，2010

[14] 鍋島 英知，宋 剛秀，“高速 SAT ソルバーの原理”，人工知能学会誌，25 巻，第 1 号，pp. 68–76，2010

[15] 田村 直之，丹生 智也，番原 睦則，“制約最適化問題と SAT 符号化”，人工知能学会誌，25 巻，第 1 号，pp. 77–85，2010

[16] 鍋島 英知，“SAT によるプランニングとスケジューリング”，人工知能学会誌，25 巻，第 1 号，pp. 114–121，2010

[17] 番原 睦則，田村 直之，“SAT によるシステム検証”，人工知能学会誌，25 巻，第 1 号，pp. 122–129，2010

[18] 元岡 達，“岩波マイクロエレクトロニクス 4:VLSI の設計 2 論理とテスト”，岩波書店，1987

[19] Tracy Larrabee, “Test Pattern Generation Using Boolean Satisfiablity”, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Volume 11, No.1, pp. 4–15, 1992

[20] R. Glenn Wood and Rob A. Rutenbar, “FPGA Routing and Routability Estimation Via Boolean Satisfiability”, IEEE Transactions on Very Large Scale Integration(VLSI) Systems, Volume 6, Issue 2, pp. 222-231, 1998

[21] Andreas Kuehulmann, “The Best of ICCAD: 20 Years Excellence in Computer-Aided Design”, Springer, 2003

[22] 岩間 一雄，“情報系教科書シリーズ 4 アルゴリズム理論入門”，昭晃堂，2001

[23] Martin Davis, George Logemann, and Donald Loveland, "A Machine Program for Theorem Proving", Communications of the ACM, Volume 5, Issue 7, 1962

[24] João P. Marques-Silva and Karem A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability", IEEE Transactions on Computers, Volume 48, Issue 5, pp. 506–521, 1999

[25] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik, "Chaff: Engineering an Efficient SAT Solver", 38th International Design Automation Conference, pp. 530–535, 2001

[26] Evguenii I. Goldberg and Yakov Novikov, "Berkmin: A Fast and Robust SAT Solver", 2002 Design, Automation and Test in Europe, pp. 142–149, 2002

[27] Niklas Eén and Niklas Sörensson, "An Extensible SAT-Solver", 6th International Conference on Theory and Applications of Satisfiability Testing, pp. 502–518, 2003

[28] Holger H. Hoos and Thomas Stützle, "Local Search Algorithms for SAT: An Empirical Evaluation", Journal of Automated Reasoning, Volume 24, Number 4, pp. 421–481, 2000

[29] Bart Selman, Henry A. Kautz, and Bram Cohen, "Noise Strategies for Improving Local Search", 12th National Conference on Artificial Intelligence, pp. 337–343, 1994

[30] David A. McAllester, Bart Selman, and Henry A. Kautz, "Evidence for Invariants in Local Search", 14th National Conference on Artificial Intelligence, pp. 321–326, 1997

[31] Holger H. Hoos, "An Adaptive Noise Mechanism for WalkSAT", 18th National Conference on Artificial Intelligence, pp. 655–660, 2002

[32] Steven D. Prestwich, "Random Walk with Continuously Smoothed Variable Weights", 8th International Conference on Theory and Applications of Satisfiability Testing, pp. 203–215, 2005

[33] Dave A. D. Tompkins and Holger H. Hoos, "Dynamic Scoring Functions with Variable Expressions: New SLS Methods for Solving SAT", 13th International Conference on Theory and Applications of Satisfiability Testing, pp. 278–292, 2010

[34] Frank Hutter, Dave A.D. Tompkins, and Holger H. Hoos, "Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT", 8th International Conference of Principles and Practice of Constraint Programming, pp. 233–248, 2002

[35] John Thoronton, Duc N. Pham, Stuart Bain, and Valnir Ferreira Jr, "Additive versus Multiplicative Clause Weighting for SAT", 19th National Conference on Artificial Intelligence, pp. 191–196, 2004

[36] Ronen I. Brafman, "A Simplifier for Propositional Formulars with Many Binary Clauses", 17th International Conference of Artificial Intelligence, pp. 515–522, 2001

[37] Fahiem Bacchus and Jonathan Winter, "Effective Preprocessing with Hyper-Resolution and Equality Reduction", 6th International Conference on Theory and Applications of Satisfiability Testing, pp. 341–355, 2003

[38] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan, "NiVER: Non Increasing Variable Elimination Resolution for Preprocessing SAT Instances", 7th International Conference on Theory and Applications of Satisfiability Testing, pp. 276–291, 2004

[39] Niklas Eén and Armin Biere, "Effective Preprocessing in SAT Through Variable and Clause Elimination", 8th International Conference on Theory and Applications of Satisfiability Testing, pp. 61–75, 2005

[40] Anbulagan, Duc N. Pham, John K. Slancy, and Abdul Satter, "Old Resolution Meets Modern SLS", 20th National Conference on Artificial Intelligence, pp. 354–359, 2005

[41] Anbulagan, Duc N. Pham, John K. Slancy, and Abdul Satter, "Boosting SLS Performance by Incorporating Resolution Preprocessor", 3rd International Workshop on Local Search Tequniques in Constraint Satisfaction, pp. 43–57, 2006

[42] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Sais, "Integrating Conflict Driven Clause Learning to Local Search", 6th International Workshop on Local Search Techniques in Constraint Satisfaction, pp. 55–68, 2009

[43] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Sais, "Boosting Local Search Thanks to CDCL", 17th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, pp. 474–488, 2010

[44] "J0123N81(Ver.8.1) SDRAM の使い方 - ユーザーズマニュアル", http://www.elpida.com, 2009

[45] "J1503E10(Ver.1.0) DDR3 SDRAM 新機能の説明 - ユーザーズマニュアル", http://www.elpida.com, 2009

[46] "2Gb: x4, x8, x16 DDR3 SDRAM Features Rev. K", http://www.micron.com, 2006

[47] "ADM-XRC-II Datasheet", http://alpha-data.co.uk, 2010

[48] "XRM ZBT Datasheet", http://alpha-data.co.uk, 2009

[49] "DS031(Ver.3.5) Virtex II Platform FPGAs: Complete Datasheet", http://www.xilinx.com, 2007

[50] "Virtex-4 Family Overview", http://www.xilinx.com, 2010

[51] "Virtex-4 FPGA User Guide", http://www.xilinx.com, 2008

[52] "ADM-XRC-6T1 Datasheet", http://www.alpha-data.co.uk, 2010

[53] "DS186(Ver.1.7) Virtex-6 FPGA Memory Interface Solutions", http://www.xilinx.com, 2011

[54] "GA-EX58-UD3R User's Manual", Giga-byte Technology co, ltd., 2009

[55] Iouliia Skliarova, António de B. Ferrari, "Reconfigurable Hardware SAT solvers: A survey of Systems", IEEE Transactions on Computers, Volume 53, Issue 11, pp. 1449–1461, 2004.

[56] Kenji Kanazawa and Tsutomu Maruyama, “An FPGA Solver for WSAT Algorithms”, 15th International Conference on Field Programmable Logic and Applications, pp. 83–88, 2005

[57] 金沢 健治，丸山 勉，“FPGA を用いた WSAT アルゴリズムの高速計算”，電子情報通信学会論文誌，Volume J89-D, Issue 6, pp.1173–1181, 2006

[58] Kenji Kanazawa and Tsutomu Maruyama, “An FPGA Solver for Large SAT Problems”, 16th International Conference on Field Programmable Logic and Applications, pp. 303–308, 2006

[59] 金沢 健治，丸山 勉，“FPGA を用いた大規模な充足可能性問題の高速計算”，電子情報通信学会論文誌，Volume J90-D, Issue 10, pp. 2713–2722, 2007

[60] Kenji Kanazawa and Tsutomu Maruyama, “An FPGA Solver for Very Large SAT Problems”, 17th International Conference on Field Programmable Logic and Applications, pp. 493–496, 2007

[61] Kenji Kanazawa and Tsutomu Maruyama, “An FPGA Solver for SAT-encoded Formal Verification Problems”, 21st International Conference on Field Programmable Logic and Applications, pp. 38–43, 2011

[62] Kenji Kanazawa and Tsutomu Maruyama, “An Approach for Solving Large SAT Problems on FPGA”, ACM Transactions on Reconfigurable Technology and Systems, Volume 4, Issue 1, 2010

[63] C. J. Tavares, C. Bungardean, G. M. Matos, José T. de Sousa, “Solving SAT with a Context-Switching Virtual Clause Pipeline and an FPGA Embedded Processor”, 14th International Conference on Field Programmable Logic and Applications, pp. 344–353, 2004

[64] Kanupriya Gulati, Suganth Paul, Sunil P. Khatri, Srinivas Patil, and Abhijit Jas, “FPGA-Based Hardware Acceleration for Boolean Satisfiability”, ACM Transactions on Design Automation of Electronic Systems, Volume 14, Issue 2, 2009

[65] Philip H. W. Leong, C. W. Sham, W. C. Wong, H. Y. Wong, Ying S. Yuen and Monk-Ping Leong, “A Bitstream reconfigurable FPGA implementation of the WSAT algorithm”, IEEE Transactions on Very Large Scale Integration Systems Volume 9, Issue 1, pp. 197–201, 2001.

[66] Roland H. C. Yap, Stella Wang and Martin Henz, "Hardware Implementations of Realtime Reconfigurable WSAT Variants", 13th International Conference on Field Programmable Logic and Applications, pp. 488–496, 2003

[67] http://www.satcompetition.org

[68] http://fmv.jku.at/sat-race-2006/

[69] http://baldur.iti.uka.de/sat-race-2008/

[70] http://baldur.iti.uka.de/sat-race-2010/

[71] DIMACS Challenge. Satisfiability: Suggested format, 1993

[72] http://www.satlib.org/solvers.html

[73] http://www.satlib.org/benchm.html

[74] http://www.satlive.org

[75] http://www.is.titech.ac.jp/~watanabe/gensat/

[76] Richard Ostrowski, Éric Grégoire, Bertrand Mazure, and Lakhdar Sais, "Recovering and Exploiting Strutural Knowledge from CNF Formulas", 8th International Conference on Principles and Practice of Constraint Programming, pp. 185–199, 2002

[77] Éric Grégoire, Richard Ostrowski, Bertrand Mazure, and Lakhdar Sais, "Automatic Extraction of Functional Dependencies", 7th International Conference on Theory and Applications of Satisfiability Testing, pp. 122–132, 2004

[78] Duc N. Pham, John Thornton, Abdul Sattar, "Building structure into local search for SAT", 20th International Joint Conference on Artificial Intelligence, pp. 2359–2364, 2007

研究業績

論文誌 (筆頭)

- Kenji Kanazawa and Tsutomu Maruyama, “An Approach for Solving Large SAT Problems on FPGA”, ACM Transactions on Reconfigurable Technology and Systems, Volume 4, Issue 1, 2010
- 金沢 健治，丸山 勉，“FPGA を用いた大規模な充足可能性問題の高速計算”，電子情報通信学会論文誌，Volume J90-D, Issue 10, pp. 2713–2722, 2007
- 金沢 健治，丸山 勉，“FPGA を用いた WSAT アルゴリズムの高速計算”，電子情報通信学会論文誌，Volume J89-D, Issue 6, pp. 1173–1181, 2006

査読付国際会議 (筆頭)

- Kenji Kanazawa and Tsutomu Maruyama, “An FPGA Solver for SAT-encoded Formal Verification Problems”, 21st International Conference on Field Programmable Logic and Applications, pp. 38–43, 2011
- Kenji Kanazawa and Tsutomu Maruyama, “An FPGA Solver for Very Large SAT Problems”, 17th International Conference on Field Programmable Logic and Applications, pp. 493–496, 2007
- Kenji Kanazawa and Tsutomu Maruyama, “An FPGA Solver for Large SAT Problems”, 16th International Conference on Field Programmable Logic and Applications, pp. 303–308, 2006
- Kenji Kanazawa and Tsutomu Maruyama, “An FPGA Solver for WSAT Algorithms”, 15th International Conference on Field Programmable Logic and Applications, pp. 83–88, 2005

査読付国際会議 (共著)

- Yoshiki Yamaguchi, Kenji Kanazawa, Yoshiharu Ohke and Tsutomu Maruyama, "An Acceleration Method for Evolutionary Systems Based on Iterated Prisoner's Dilemma", 3rd International Workshop on Applied Reconfigurable Computing, pp. 358–364, 2007