再構成可能なハードウェアを用いた充足可能性問題の解法

須山 敬之　横尾 真　名古屋 彰

NTT コミュニケーション科学研究所
〒619-0237 京都府相楽郡精華町光台 2-4
Tel: 0774-93-5272　Fax: 0774-93-5285
e-mail: suyama@cslab.kecl.ntt.co.jp

あらまし　本稿では再構成可能なハードウェアを用いた充足可能性問題 (Satisfiability Problems: SAT) の解法について報告する．近年の FPGA に代表される再構成可能な論理素子の技術の進歩により，利用者が手元でかつ簡単に専用の論理回路を実現することが可能となってきている．そこで SAT の個々の問題専用の論理回路を FPGA 上に構成することにより，その問題を高速に解く手法を提案する．SAT とは和積形論理式を真にするような変数への値の割り当てが存在するかどうかを調べる問題であり，代表的な NP 完全問題の一つである．ここではハードウェア上で解くために適した，探索木を削減する新たなアルゴリズムを提案し，その評価，及び実装状況について述べる．
キーワード　充足可能性問題，再構成可能なハードウェア，論理合成，アルゴリズム

Solving Satisfiability Problems using Reconfigurable Hardware

Takayuki Suyama,　Makoto Yokoo,　Akira Nagoya

NTT Communication Science Laboratories
2-4 Hikaridai, Soraku-gun, Seika-cho, Kyoto, 619-0237 Japan
Tel: +81-774-93-5272　Fax: +81-774-93-5285
e-mail: suyama@cslab.kecl.ntt.co.jp

Abstract　This paper presents a report on an approach for solving satisfiability problems (SAT) using Reconfigurable Hardware. Recently, due to advances in Reconfigurable Hardware such as FPGAs, users can now create their own reconfigurable logic circuits. This technology has enabled users to rapidly create logic circuits specialized for solving individual problem instances. Satisfiability problems were chosen because they make up an important subclass of NP-hard problems. We have developed a new algorithm which is suitable for hardware and can reduce the search tree size. We report evaluation results and implementation status.
key words　satisfiability problems, Reconfigurable Hardware, logic synthesis, algorithm

1 はじめに

近年のFPGA (Field Programmable Gate Arrays [1]) に代表される内部論理が再構成可能なハードウェアの技術の発達により, 大規模な専用回路を利用者の手元で手軽に実現できるようになってきた. またハードウェア記述言語からの論理合成システムの利用により, ハードウェアのデザイン自体もより短期間に行うことができるようになってきている. これらの技術を用いることにより, ユーザが使用時にハードウェアの論理的構造を自由に変化させ, より高速な処理を可能とするリコンフィギャラブル・コンピューティングと呼ばれる手法の研究が盛んに行われるようになってきている. この手法により, 何らかの問題を解く際に, 個々の問題毎に特化された論理回路を構成して解くことが可能となる. ここではその例題として充足可能性問題 (Satisfiability Problems: SAT) を用いて本手法の有効性を検証する.

制約充足問題 (constraint satisfaction problem) は様々な問題を定式化できる一般的な枠組みであり, これまでに多くの研究がなされている [2]. その中でも特に和積形論理式の充足可能性問題は代表的なNP完全問題の一つであり, 最初にNP完全であることが発見された問題でもある [3].

本稿では論理合成システムとして, NTTで開発されたPARTHENON[4, 5] を用いて, 充足可能性問題の個々の問題に対してその問題を解くための論理回路を合成し, FPGA上にマッピングすることにより, 問題をハードウェアで高速に解く手法について報告する. またこの方法に適したアルゴリズムを提案する. ソフトウェアにおけるアルゴリズムでは変数の値は逐次的に決定されていくが, 提案するハードウェア向きのアルゴリズムでは全ての制約が同時に評価され, 変数の値が決められる.

Davis-Putnamの基本的な手法 [6] をハードウェア上に実装することにより, この手法が有効であることが先ず [7, 8] で示された. その後, 同様の研究がなされ, 処理速度が改善されるなど多くの研究結果が報告されている [9, 10, 11, 12, 13].

本稿ではシミュレーションにより, このアルゴリズムによる探索に必要な時間のオーダに関してMaximum Occurrences in clauses of Minimum Size Heuristic (MOM)[14, 15] による手法をハードウェア化した手法 [12] と今回提案する手法の比較を行う. また, アルゴリズムの違いによる必要なハードウェア量の比較検討を行い, FPGA上への実装状況について述べる.

2 問題の定義

充足可能性問題は与えられたブール式の値を1とするような変数の値の割り当て方が存在するかを判定する問題である.

2値変数 x_i は真か偽のいずれかを取る (それぞれを1と0で表わす). 変数の否定 $\overline{x_i}$ は変数の値が0の時1, 1の時0となる. 変数およびその否定をリテラル (literal) と呼ぶ. リテラル $l_k(k = 1,\ldots,m)$ の論理和 $l_1 + l_1+,\cdots,+l_m$ は節 (clause) と呼ばれる. 論理和 (+) により, 節はリテラルが一つでも1である場合には1となり, そうでない場合は0となる. 和積形論理式とは, 節を $C_n(n = 1,\ldots,p)$ とすると, 節の論理積 $(C_1 \cdot C_1\cdot,\cdots,\cdot C_p)$ である. 論理積 $(\cdot)$ により, 論理式はは全ての節が1であるときに1となり, それ以外の時は0となる. 充足可能性問題は与えられた論理式を真 (すなわち1) とする変数への値の割り当てが存在するかどうかを調べる問題であり, 1とする変数への割り当てを充足解と呼ぶ.

これ以降は問題が各節がちょうど3個のリテラルを含む和積標準のブール式で表わされる場合 (3-SATと呼ばれる) に限定して話を進める. 3-SATもNP完全であることが証明されている.

3-SATを解くためのアルゴリズムはただ一つの解を求めることを目的としている場合がほとんどであるが, ここでは論理式が充足可能である場合, 全ての充足解を求めるものとし, また充足可能でない場合は, 充足解が存在しないことを保証する.

3 処理の流れ

ソフトウェアを用いて問題を解く場合, 用いられるアルゴリズムは汎用計算機上で逐次的に実行されることを前提としている. また, メモリ上に格納されているデータの参照にはポインタなどが用

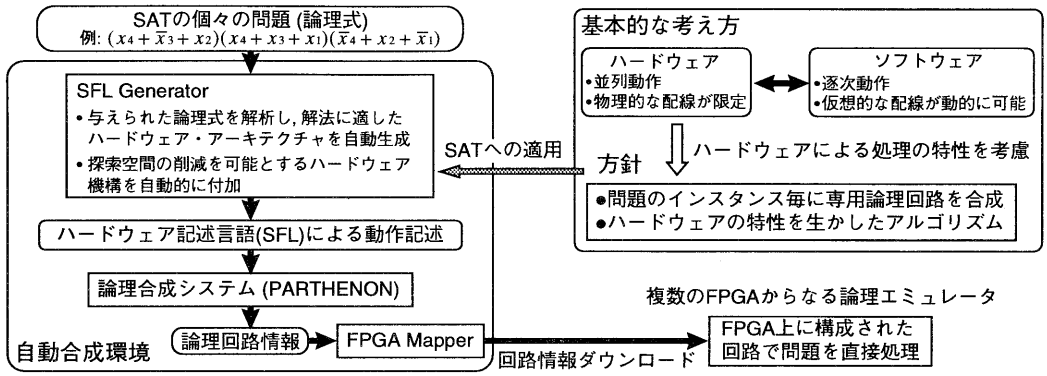

図 1: SAT を解くためのハードウェア合成の流れ

いられ, メモリ上のデータにはほぼ制約がなくアクセスが可能である. それに対し, ハードウェアは並列に動作することを基本としており, またデータの参照を行う場合はデータのやり取りを行う間を物理的に接続する必要がある. これらの違いを考慮し, 以下のような方針を取ることとする.

1. 問題のインスタンス毎にその問題専用の論理回路を合成.

2. ハードウェア向けのアルゴリズムを開発, 適用.

第一の方針はハードウェアの可変性の少なさによるものである. FPGA の技術により論理の可変性は従来より増しているが, 依然としてソフトウェアより劣っているため, SAT の全ての問題に対応するハードウェアを合成するよりも, 個々の問題に対して合成する方が効率がよい. 第二の方針はソフトウェアとハードウェアの違いに起因するものである. すなわち, ハードウェアの並列性を活かすため, 並列性を重視したアルゴリズムを開発し, 適用する.

図 1に処理の流れを示す. 与えられた SAT の問題は SFL generator(C プログラム) により解析され, ハードウェア向けのアルゴリズムを含んだその問題を解くためのハードウェア記述言語による記述が作成される. その記述を入力として PARTHENON により論理回路 (ネットリスト) が論理合成される. 論理合成により得られた回路は FPGA のマッピングプログラムに入力され, それにより FPGA 用のダウンロードデータが得られる. そのデータを実際に FPGA に転送し, 動作させることにより, 与えられた問題を解くことができる.

4 アルゴリズム

本稿で提案するアルゴリズムは Davis-Putnam による手続きに拡張を加えたものである. 変数を選択する過程に於いて Experimental Unit Propagation [16] を用いる. 木を探索するようなアルゴリズムを実装する場合, 汎用計算機ではスタックが用いられる事が多い. それに対して, 等価なアルゴリズムをハードウェアのみで実装する場合, スタックを用いる方針は, 大きなメモリを FPGA 上に構成することが難しい, メモリへのアクセスがボトルネックとなる等の点で不利となる. ここではそれらを避けるため, 各変数毎にその変数の状態を保持するためのレジスタを用意し, そのレジスタの値を変更することにより, 各変数が決定されているかどうか, また探索木中の位置等を記憶するようにする. その詳細を以下に示す.

4.1 アルゴリズムの詳細

アルゴリズムの詳細について述べる．以下では x_i が真である時，$(x_i, 1)$ とする．

- 各変数 x_i に対して $depth(x_i)$ が定義される．初期値は 0．値が決定された時点での探索木の深さを示す．
- 各変数 x_i に対して $determined(x_i)$ が定義される．初期値は 0．1 は値が決定されている事を示す．
- 各変数 x_i に対して $branch(x_i)$ が定義される．$branch(x_i) = 0$ であれば，x_i の値は unit resolution で決定されたことを，$branch(x_i) = 1$ であれば x_i の値が branching で決定されたことを示す．
- グローバル変数 *current_depth* が定義される．初期値は 1．
- 各節に関して，状態を not-satisfied, satisfied, unit, other に分類する．
- 複数の unit の節があり，同じ変数に違う値を割り当てる場合，これらの unit 節は矛盾であると言う．

提案するアルゴリズムは "Main Procedure" と "Experimental Unit Propagation Procedure" から成っている．Experimental Unit Propagation Procedure は Main Procedure 中の **Branching** ステートに於いて，次に値が割り当てられるべき変数を決定するために用いられる．初期状態では *current_depth* は 1，各変数の x_i, $determined(x_i)$ 及び $branch(x_i)$ は 0 である．

Main Procedure

1. **Not Satisfied:** *not-satisfied* の節がある場合，もしくは矛盾する unit 節がある場合，5 に行く．
2. **Satisfied:** すべての節が satisfied の場合，現在の値の割り当てを解として出力．5 に行く．
3. **Unit:** *unit* が存在し，それらが矛盾しない場合，各 unit 節中の $determined(x_i)$ が 0 の変数 x_i に関して，x_i の値を節で指定される値 v_i に，$determined(x_i)$ を 1 に，$branch(x_i)$ を 0 に，$depth(x_i)$ を *current_depth* に設定，1 に戻る．
4. **Branching:** その他の場合，max_eval_min=0, max_eval_sum=0 として，$determined(x_i) = 0$ である すべての変数に x_i に関して以下を実行．
   (a) *eval_0* に Experimental Unit Propagation Procedure$(x_i, 0)$ の実行結果をセットする．*eval_0* の値が not-satisfied の場合は，$determined(x_i) = 1$, x_i=1, $branch(x_i) = 0$, $depth(x_i) = current\_depth$ とし，1 に行く (他の変数に関する処理は abort)．
   (b) *eval_1* に Experimental Unit Propagation Procedure$(x_i, 1)$ の実行結果をセットする．*eval_1* の値が not-satisfied の場合は，$determined(x_i) = 1$, x_i=0, $branch(x_i) = 0$, $depth(x_i) = current\_depth$ とし，1 に行く (他の変数に関する処理は abort)．
   (c) $max\_eval\_min < min(eval\_0, eval\_1)$, または $max\_eval\_min = min(eval\_0, eval\_1)$ かつ $max\_eval\_sum < eval\_0 + eval\_1$, ならば *best_pos* に x_i をセット，max_eval_min に $min(eval\_0, eval\_1)$ を，max_eval_sum に $eval\_0 + eval\_1$ をそれぞれセットする．

   branching End *best_pos* で指定される変数 x_i に関して，x_i=0, $determined(x_i) = 1$, $branch(x_i)$ を 1, $depth(x_i)$ を $current\_depth + 1$ に設定，*current_depth* を 1 増加．1 に戻る．
5. **Backtracking:**
   5.1 *current_depth* が 1 ならアルゴリズムを終了．

5.2 その他の場合，各変数 x_i に関して，$depth(x_i) = current\_depth$ かつ $branch(x_i) = 0$ ならば，$determined(x_i)$ を 0，$depth(x_i)$ を 0 に設定．$depth(x_i) = current\_depth$ かつ $branch(x_i) = 1$ ならば，x_i の値を 1 に，$branch(x_i)$ を 0 に，$depth(x_i)$ を $current\_depth - 1$ に設定，

5.3 $current\_depth$ を $current\_depth - 1$ に設定，1 にもどる．

Experimental Unit Propagation($x_i, value$) Procedure

先ず count=0，$x_i = val$，$branch(x_i)$ を 1 に，$determined(x_i) = 1$，$depth(x_i) = current\_depth+1$，$current\_depth = current\_depth+1$ とする．

1. **Not Satisfied:** *not-satisfied* の節がある場合，もしくは矛盾する unit 節がある場合，$result = not\_satisfied$ として 5 に行く．
2. **Satisfied:** すべての節が satisfied の場合，$result = n$ として 5 に行く．
3. **Unit:** *unit* が存在し，それらが矛盾しない場合，各 unit 節 中の $determined(x_i)$ が 0 の変数 x_i に関して，x_i の値を節で指定される値 v_i に，$determined(x_i)$ を 1 に，$branch(x_i)$ を 0 に，$depth(x_i)$ を $current\_depth$ に設定，$count$ を 1 増加 (複数の unit 節がある場合でも 1 増加するだけで良い)，1 に戻る．
4. **Branching:** その他の場合，$result = count$ として 5 に行く．
5. **Backtracking:** 各変数 x_i に関して，$depth(x_i) = current\_depth$ ならば，$determined(x_i)$ を 0 に，$branch(x_i)$ を 0 に，$depth(x_i)$ を n+1 にする．$current\_depth$ を 1 減らす．$result$ を返す．

5 実装方法

第 4.1節で述べたアルゴリズムは有限状態マシンとして実装することが可能である．

Main Procedure (以下 Main) と Experimental Unit Propagation Procedure (以下 EUP) は手続きとして共通である部分があり，かつ，Main と EUP は同時に行われることがないため，ハードウェアを共用することが可能である．それぞれの動作状態にある時，それを Main mode, EUP mode と呼ぶ．*evaluation*, *unit* 及び *backtracking* の各状態は Main mode, EUP mode の両方で使われるが，その動作はこれまで述べたように mode によりやや異なっている．その区別を行うため，*eup* と呼ぶフラグを設け，それにより状態の認識，制御を行う．

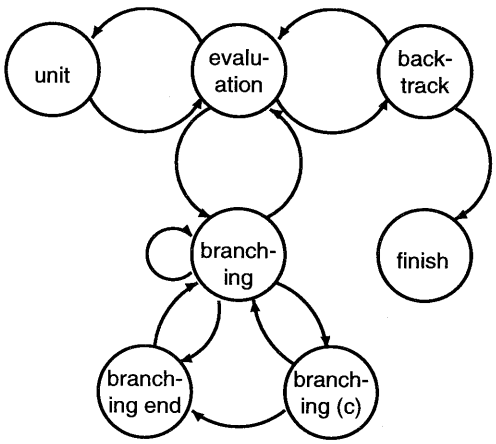

図 2: 状態遷移図

図 2に状態遷移の様子を示す．各ステートは以下のような動作を行なう．*evaluation* ステートでは各節の状態を並列にチェックする．例えば節が *not-satisfied* かどうかのチェックは以下のような回路を用いて行なうことが可能である．

$$(determined(x_i) \cdot determined(x_j) \cdot determined(x_k)) \cdot ((x_i \oplus v_i) \cdot (x_j \oplus v_j) \cdot (x_k \oplus v_k))$$

ここで $\oplus$ は Exclusive-OR

backtracking では，各変数毎に $depth(x_i)$ と *current_depth* を比較し，変数値，*determined*, *depth* 等を決定する．これらの動作も各変数毎に並列に行なわれる．

unit では，その変数が unit 節に含まれる場合，変数値が決定される．

状態が *evaluation* ステートから *branching* ステートに遷移する時，*eup* が 1 にセットされ，mode が Main から EUP に変化する．*branching* ステート *branching end* ステート等では，アルゴリズムに基づきそれぞれに対応する動作が行なわれる．

6 評価

6.1 シミュレーション結果

ランダムに作成された 3-SAT の問題を用いて，シミュレーションにより本アルゴリズムの評価を行なった．節の数は変数の数の 4.3 倍とした．これは乱数で問題を作成した場合，難しい問題ができる値であるとされている [17].

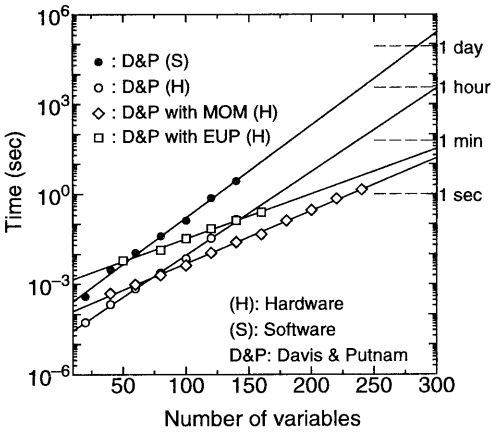

図 3: ランダムな 3-SAT を解く場合に必要な時間

図 3に評価結果を示す．ハードウェアでは動作クロックを従来の結果との比較のため 1MHz としている．ランダムに作成された問題では解の個数が非常に大きくなる場合があるため，100 個の解が見つかった段階でシミュレーションを終了している．MOM を用いた場合 [12], 探索木の大きさが変数の数を n とした時 $O(2^{n/17.3})$ で大きくなるのに対し，EUP を用いた場合，それが $O(2^{n/20.0})$ で大きくなることが分かる．これは変数の数が大きくなった場合，提案する手法が有利であることを示している．

表 1に各問題を解くために必要なステート数および時間を示す．これらの問題は DIMACS Challenge benchmark problems[18] に含まれる問題である．ここでは動作クロックを 6.2節で動作が確認された 12MHz としている．比較のため，POSIT[15] プログラムを Sun Ultra 30 Model 300 (UltraSPARC-II 296MHz) 上で動作させた場合の cpu タイムを示す．この表より提案手法がソフトウェアより高速に解を求めることができることが分かる．

6.2 実装状況

ALTERA 社の FLEX10K250 を用いて，アルゴリズムを実装した結果を示す．FLEX10K250 は 12,160 個の Logic Cells (LCs) を内蔵し，使用可能ゲート数は 149k から 310k 程度である．この FPGA を一つ用いて，“aim-100-2_0-no-1.cnf” を実装することができた．これは 100 変数，200 節を含んだ DIMAC benchmark の問題である．実装に際して用いた LC 数は 9,464 で，全体の 77% を使用，シミュレーションによる遅延の評価では 12.07MHz の動作が可能であった．また，200 変数 320 節の問題である “aim-200-1_6-yes1-1.cnf” も FPGA 上にマッピングすることができた．ただし，21 個の FLEX10K シリーズのチップに分割された．これは LC 間の接続に多くリソースを必要としたためと考えられる．LC の使用率は 13% であった．

図 4に “aim-{50,100,200}-1_6-yes1-1.cnf” の論理合成時のゲート数を示す．アルゴリズム自体が必要とするゲート数の比較のため，論理合成時の初期回路の等価ゲート数を用いている．今回提案した EUP によるアルゴリズムは MOM による場合より約 30% 使用ゲート数が減少している．これはアルゴリズム中に共用できる部分があるため，

表 1: "aim-<#variables>-2_0-no-1.cnf" を解くために必要なステート数と時間

problem	# of states	time (s) @ 12MHz	cpu time of POSIT (s)
aim-100-2_0-no-1	182,766,201	15.2	92.2
aim-100-2_0-no-2	154,293,446	12.8	50.2
aim-100-2_0-no-3	137,816,329	11.5	18.7
aim-100-2_0-no-4	415,784,455	34.6	43.3
average	222,665,107	18.5	51.5

branching 用に別の回路が必要な MOM に比べて少ない回路ですむためである.

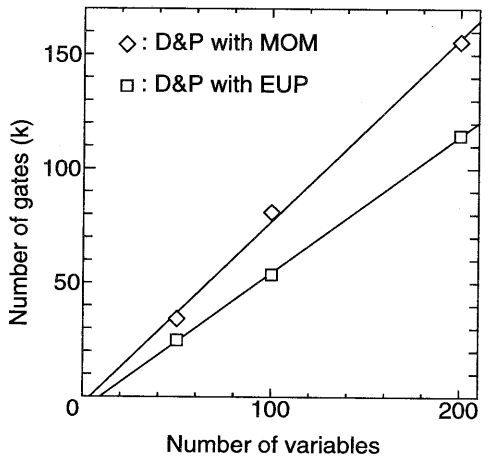

図 4: "aim-<#variables>-1_6-yes1-1.cnf" を実装するために必要なゲート数

7 まとめ

本稿では FPGA を用いて充足可能性問題を解く手法について述べた. Davis-Putnam の手法を基本とし, Experimental Unit Propagation により拡張を加え, それをハードウェア向けに改良したアルゴリズムについて述べ, その手法の有効性について述べた. その手法を用いることにより MOM の手法より約 30% ゲート数を削減でき, 探索時間のオーダも改善することができた. また, 実際に 100 変数の問題に関して実装し, 更に 200 変数の問題に関しても FPGA にマッピングが可能なことを確認した.

今後は更にアルゴリズムを改良し, PCA[19] 等の動的再構成可能なハードウェア上で実行可能な手法について提案する予定である.

謝辞

本研究を進めるにあたり, 御指導, 御助言頂いている NTT コミュニケーション科学研究所服部部長, メリーランド大学中島和生教授に感謝致します.

参考文献

[1] Brown, S. D.; Francis, R. J.; Rose, J.; and Vranesic, Z. G. *Field-Programmable Gate Arrays.* Kluwer Academic Publishers, 1992

[2] Mackworth, A. K. Constraint satisfaction. In Shapiro, S. C., ed., *Encyclopedia of Artificial Intelligence.* New York: Wiley-Interscience Publication. pp. 285–293, 1992

[3] Cook, S. The complexity of theorem proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computation*, pp.151–158, 1971

[4] Camposano, R.; and Wolf, W. *High-level VLSI synthesis.* Kluwer Academic, 1991

[5] Nakamura, Y.; Oguri, K.; Nagoya, A.; Yukishita, M.; and Nomura, R. High-level synthesis design at NTT systems labs. *IEICE Trans. Inf & Syst.* E76-D(9): pp. 1047–1054, 1993

[6] Davis, M.; and Putnam, H. A computing procedure for quantification theory. *Journal of the ACM* 7: pp. 201–215, 1960

[7] Suyama, T.; Yokoo, M.; and Sawada, H. Solving satisfiability problems on FPGAs. In *Proceedings of the 6th International Workshop on Field Programmable Logic and Applications*, pp. 136–145, Springer, 1996

[8] 須山 敬之, 横尾 真, 澤田 宏, 名古屋 彰: FPGA と論理合成システムを用いた充足可能性問題の解法, 信学技報, VLD96-86,CPSY96-98, pp. 167–174, 1996

[9] Abramovici, M.; and Saab, D. Satisfiability on Reconfigurable Hardware. In *International Workshop on Field Programmable Logic and Applications*, pp. 448–456, 1997

[10] Hamadi, Y.; and Merceron D. Reconfigurable Architectures: A New Vision for Optimizing Problem. In *Proceedings of Third International Conference on Principles and Practice on Constraint Programming (CP'97)*. pp. 209–221, 1997

[11] Rashid, A.; Leonard, J.; and Mangione-Smith, W. H. Dynamic Circuit Generation for Solving Specific Problem Instances of Boolean Satisfiability. In *Proc. of IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 196–204, 1998

[12] Suyama, T., Yokoo, M. and Sawada, H. Solving Satisfiability Problems Using Logic Synthesis and Reconfigurable Hardware. *Proc. of the 31st Annual Hawaii International Conference on System Sciences (HICSS-31)*, Vol. VII, pp. 179–186, 1998

[13] Zhong, P.; Martonosi, M.; Ashar, P.; and Malik S. Accelerating Boolean Satisfiability with Configurable Hardware. In *Proc. of Symposium on Field-Programmable Custom Computing Machines*, pp. 186–195, 1998

[14] Dubois, O.; Andre, P.; Boufkhad, Y.; and Carlier, J. Can a very simple algorithm be efficient for solving the SAT problem? In *Proceedings of the DIMACS Challenge II Workshop*, 1993

[15] Freeman, J. W. *Improvements to propositional satisfiability search algorithms.* Ph.D. Dissertation, the University of Pennsylvania, 1995

[16] Li, C. M.; Anbulagan. Heuristics Based on Unit propagation for Satisfiability Problems. In *Proceedings of 15th International Joint Conference on Artificial Intelligence.* pp. 366–371, 1997

[17] Mitchell, D.; Selman, B.; and Levesque, H. Hard and easy distributions of SAT problem. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 459–465, 1992

[18] ftp://dimacs.rutgers.edu/pub/challenge/satisfiability

[19] 小栗 清, 伊藤 秀之, 小西 隆介, 名古屋 彰: 布線論理による汎用計算機構 (プラスティックセルアーキテクチャ), 信学技報, CPSY98-54, pp. 45–52, 1998.