# 詰将棋専用ハードウェアの作成

堀 洋平[†] 斎藤尚徳[††] 丸山勉[†††]

将棋プログラムの棋力の向上のために，専用ハードウェアシステムの開発は必要不可欠である．本研究ではシステム開発の第1段階として，Field-Programmable Gate Array（FPGA）を使用し詰将棋の専用ハードウェアの作成を行った．FPGAはユーザ自らが回路構成を変更することのできるLSIであり，また内部に大容量のRAMを有するため，きわめて並列度の高い演算をチップ内部で実現することができる．この特長を活かし，詰将棋に適した並列・パイプラインアーキテクチャを開発した．本研究で作成したハードウェアでは，局面情報データを複数のモジュールで並列に生成し，これらのデータをパイプライン処理によって指手データへと変換することで高速な演算を可能にした．また，指手を複数のカテゴリに分類し，これらを並列・パイプライン処理によって生成することによりさらなる高速化を実現した．本論文では，詰将棋ハードウェアにおける指手生成の手法とアーキテクチャについて述べた後，実際に問題局面を解いてハードウェアの性能について議論する．

# Implementation of Tsume Shogi Hardware

Yohei Hori,[†] Hisanori Saito[††] and Tsutomu Maruyama[†††]

Developing dedicated hardware systems is an essential approach to improve play strength of shogi programs. To date, use of programmable devices for shogi hardware has been proposed as a feasible method to resolve the problems of high cost and long developing time of hardware implementation. To devise architecture of shogi hardware, we first implemented a tsume shogi solver on a Field-Programmable Gate Array (FPGA). With the ample hardware resource of an FPGA, we implemented highly parallelized architecture on a single chip and realized high-speed computation of tsume-shogi. In this paper, a procedure to generate moves in tsume shogi hardware and its architecture are described.

## 1. はじめに

ゲームプログラミングの分野において，チェスに代わる研究対象として将棋プログラムが注目されている．将棋は (1) 持駒を再利用できる，(2) 使用する駒の数・種類が多い，(3) 使用する盤面が広い，(4) 成りのルールが複雑で成り駒の種類も多いといった点で，計算機にとってチェスよりも難しいゲームであると認識されている[18]．特に将棋では，持駒の再利用が可能であるため1局面における合法手の総数が多く，その数はチェスが約35であるのに対し，将棋では約80である[22]．

将棋はチェスと同じくタクティカルなゲームであり，最善手の決定のためにはある程度深い探索が要求される．ゆえに将棋においても全幅探索は非現実的であり，ヒューリスティックな前向き枝刈りが行われるのが普通である．しかし，枝刈りによってゲーム木の分岐数をチェスと同程度にした場合，刈り取られる枝の割合が大きく最善手を逃す可能性が高くなると考えられる．精度の高い（最善手を残す確率の高い）枝刈りを実現するためには，正確な局面評価関数と深い探索の実現が必要であるが，将棋のルールの複雑さと合法手数の多さを考えると，チェスに比べてより多くの計算量が要求されると予想される．将棋プログラムの棋力の向上のためには，新しい評価関数や木探索法の開発等のソフトウェア面からのアプローチと，より高速な演算を実現するためのハードウェア面からのアプローチの両方が必要である．

近年，ソフトウェアの開発競争は研究者やメーカあるいは個人によってさかんに進められており，その中で様々なアルゴリズムが提案されて大きな成果をあげている[19)~21),34]．また，将棋やそれ以外も含めてゲーム木の並列探索に関する研究が行われ，その成果

† 筑波大学大学院博士課程工学研究科
Doctoral Program in Engineering, University of Tsukuba
†† ビー・ユー・ジー株式会社
B.U.G., Inc.
††† 筑波大学機能工学系
Institute of Engineering Mechanics and Systems, University of Tsukuba

が報告されている[16),25),27),28),31),35)]．しかし，専用ハードウェアに関する研究は，将棋においてはほとんど行われていない．チェスでは，IBM によってつくられたDeepBlue が，人間の世界チャンピオンである Kasparov に勝利し大きな成功を収めた[29)]．Deep Blue は，その開発過程においても様々なアルゴリズムやアーキテクチャを提案し，チェスプログラムの発展に大きく貢献した[12),13)]．しかし，このような巨大なハードウェアシステムを一般の研究者がつくることは到底不可能である．また，Kasparov との対戦後には解体されてしまい，一般の研究者がこれを使用することはできなかった．アーキテクチャが文献として公表されているとはいえ，これを参考にハードウェアを設計するためには，相当の時間を費やさなければならない．

コンピュータ将棋においては，プログラムの性能を向上させるために，専用ハードウェアの利用がより身近な手段として一般に広まればよいと筆者らは考えている．しかし，アルゴリズムの開発・改良がさかんに行われている段階であり，将棋のハードウェアの研究はそれほど行われていない．将棋のような巨大で複雑なアプリケーションにおいて，早期のハードウェア開発を実現するためには，より多くの研究者によってアーキテクチャやハードウェア・アルゴリズムが提案されてゆく必要がある．また，局面の更新，駒の効きの計算，詰将棋等の典型的な演算やコンセンサスが得られつつあるアルゴリズムについて，ハードウェア・モジュールが Intellectual Property（IP）として提供されることにより，ハードウェアの開発はさらに加速されると考える．

だが，デバイスとして従来のように ASIC を使用した場合，依然としてハードウェア開発における典型的な 2 つの問題 — 時間的・金銭的コストとアーキテクチャの非柔軟性の問題 — に直面する．将棋専用ハードウェアにおけるコストおよび非柔軟性の問題を解決する手段として，Field-Programmable Gate Array（FPGA）の使用が有効である[10),11)]．FPGA は，回路構成をユーザが自由に変更することができる LSI である．FPGA の回路構成は，ハードウェア記述言語（Hardware Description Language: HDL）を使用してテキスト形式のプログラムで記述される．プログラムは計算機上でコンパイルされ，回路構成情報へと変換される．回路の構成および変更は，この回路構成情報をチップにダウンロードすることでただちに実現される．FPGA を用いたハードウェアの開発では，Electronic Design Automation（EDA）ツールを使用することにより，回路の設計から実装までを完全にユーザの環境の中で実現することができる．そのため，開発にかかる時間とコストを大幅に削減することが可能であり，FPGA は多くの研究者の協力によるハードウェア開発を実現するためのキー・デバイスとなるであろう．

我々の目的は，FPGA を用いて作成されたハードウェアの性能を示し，将棋の専用ハードウェア開発における FPGA の有効性を明らかにすることである．その第 1 段階として，まず詰将棋専用ハードウェアの作成を行い，実際に問題を解いてソフトウェアとの性能を比較した．詰将棋は，相手玉を王手の連続で詰ませる手順を見つけ出すパズルであり，特に本将棋における終盤の棋力向上のために重要である．将棋では持駒の使用が可能であるため，終盤において盤上に残っている駒数が多く，同一局面はほとんど現れることがない．ゆえにチェスのように終盤局面をデータベース化することは困難である．将棋では詰手順の発見のために木探索を行っており，詰将棋は本将棋のハードウェア・アーキテクチャを開発する前段階のアプリケーションとして適している．本研究では，証明数を閾値とする多重反復深化を行う PN*[32)] による詰将棋プログラムをハードウェアに実装した．

本論文では，詰将棋ハードウェアのアーキテクチャおよびその性能について説明する．まず 2 章では，これまでに作成された主なチェスのハードウェアについて述べる．3 章では詰将棋のルールを説明し，さらに基本的なアルゴリズムについて述べる．4 章では，ハードウェアにおける指手生成アルゴリズムについて述べる．5 章では，実装されたハードウェア・モジュールのうち,「直接効きデータ生成回路」,「直接王手生成回路」および「木探索回路」を取り上げ，それらのアーキテクチャについて詳しく説明する．6 章では詰将棋ハードウェアの性能について述べる．7 章では本研究の今後の課題について述べ，最後に 8 章において本論文についてまとめる．

## 2. これまでの研究

1 章で述べたように，将棋プログラムのアルゴリズムに関する研究はこれまでにさかんに行われており，また汎用の並列コンピュータを用いた高速化の研究も行われている．しかし，将棋の専用ハードウェアに関する研究は報告されていない．本章では，コンピュータ・チェスにおいて開発された専用ハードウェアについて述べる．将棋プログラムのアルゴリズムに関しては，本論文では説明は省くので，内容については参考

文献を参照されたい．ただし本研究において実装したアルゴリズムについては，3 章において説明する．

本章では，現在までに作成されたチェス専用ハードウェアの中から，Belle，Deep Blue，Brutus およびMBChess-CodeBlue について述べる．Belle は初めてコンピュータチェスの競技会に参加した専用ハードウェアであり，Deep Blue は初めて人間の世界チャンピオンに勝利したチェスマシンである．Brutus とMBChess-CodeBlue は，FPGA を使用したチェスマシンである．ここではハードウェア構成を中心に述べるので，各マシンのハードウェア・アルゴリズム等については，それぞれの文献を参照されたい．

### 2.1 Belle

Belle はベル研究所の Condon と Thompson によって開発され，1977 年に専用ハードウェアとして初めてコンピュータチェスの競技会に参加した[8]．このときのハードウェアは 325 個のチップから構成されていたが，その後も改良が重ねられ，1980 年につくられたBelle は 1,700 個のチップを搭載し 1 秒間に 16 万局面の探索が可能であった．

Belle は 8 × 8 個のロジック・ブロック（chess square）から構成され，各 chess square がチェス盤のマスのそれぞれに対応している．Chess square には transmitter と receiver が 1 つずつ実装されており，これら 2 つが隣接する square と信号の受送信を行う．Belle は，捕獲される駒を見つける “find-victim（FV）” と捕獲する駒を特定する “find-aggressor（FA）” という 2 つのサイクルによって指手の生成を行う．生成される指手のうち，より価値の高い駒（Most Valuable Victim: MVV）をより価値の低い駒（Least Valuable Aggressor: LVA）を使って捕獲する場合ほど評価値は大きい．FV と FA の 2 つのサイクルによって，指手の生成と MVV/LVA に基づく指手の順序付けが行われる．

Belle における transmitter-receiver 構造および FV/FA アルゴリズムは，その後開発されたDeep Blue やMBChess-CodeBlue 等にも採用されており，チェス・マシンにおける基本方針になっているといえる．

### 2.2 Deep Blue

DeepBlue は IBM の Hsu を中心に開発され，初めて人間の世界チャンピオンに勝利したコンピュータである．Deep Blue は，32 ノードの IBM RS/6000 SP にチェス専用チップを搭載したスーパコンピュータである[6),9),14)]．1 ノードには 1 個の IBM Power2 Super Chip と 32 個の Application Specific Integrated Circuit（ASIC）が搭載されている．2 つのノードは主に I/O の制御に用いられており，実質的なチェスの計算は残りの 30 ノード（480 ASICs）によって実行されている．これら ASIC のハードウェアパワーにより，Deep Blue は 1 秒間に最大 2 億局面の評価が可能である．

### 2.3 Brutus

Brutus は Donninger によって開発されたチェス専用ハードウェアであり，デバイスとして FPGA を使用している．Brutus は 2002 年に開かれた Computer Olympiad のチェス部門に参加し，3 位の成績を収めた．Brutus は ChessBase という商業プロジェクトから支援を受けており，詳細に関する文献は筆者らの知る限り公表されていない．ChessBase によると，使用したハードウェアは Alpha Data 社製の 64 bit PCI ボード 1 枚であり，これには Xilinx 社製の FPGA である Virtex-V405E が搭載されている[7)]．

### 2.4 MBChess-CodeBlue

MBChess-CodeBlue は，Boulè によって開発されたハードウェアシステムであり，チェスプログラム MBChess の指手生成モジュールが FPGA に実装されている[3)~5)]．ハードウェアには 8×8 の chess square が実装されており，Belle と同様の transmitter-receiver 構造を持つ．Code-Blue は PCI ボードに搭載された Xilinx 社製の XCV800 に実装されている．MBChess-CodeBlue は，AMD K6-2 450 MHz 上のMBChess に対してレーティングを 150 から 200 程度向上させることができた．

### 2.5 将棋との比較

将棋の駒は 8 種類であり，このうち 6 種類が成ることができる．成駒のうち 4 種類は「金」とまったく同じ動きをするが，持駒の使用が可能であるためこれらを単純に「金」として扱うことができない．よって実質上 14 種類の駒すべてを区別する必要がある．敵味方を合わせると，将棋の 1 マスには最大で 12 方向から駒が移動してくる．将棋において Belle と同様の構造をとる場合，各種類の駒の入力が各方向からあるため，1 つのロジック・ブロックはチェスと比べて非常に大きく複雑になると考えられる．また，持駒による指手の生成や，複雑なルール下での成り生成も，ロジックを複雑にする大きな要因である．そのためブロック間の配線量が増加し，ハードウェアの性能が低下することが懸念される．また，将棋盤は 9 × 9 とチェス盤に比べて大きいので，より多くのハードウェア・リソースが必要となる．

これらの理由から，筆者らは 9 × 9 のロジック・ブ

ロックを用いる方法ではなく，独自のアーキテクチャによって詰将棋回路を実装した．そのアルゴリズムとアーキテクチャについて，4 章以降で説明する．

## 3. 詰 将 棋

本章では，まず詰将棋のルールについて説明し，次いで詰将棋プログラムのアルゴリズムについて述べる．さらに，本論文で用いるいくつかの用語を定義する．

### 3.1 ル ー ル

詰将棋は，王手の連続で相手玉を詰ませるパズルである．詰将棋では先手が攻め方となるため，先手の指手は必ず王手でなければならない．後手は受け方となり，詰みまでの手数ができるだけ長くなるように王手をかわす．ただし，後手は延命のために「無駄合」をしてはならない（先手が合駒を取り，その駒を使わなくても相手玉を詰ますことができる場合，その合駒は無駄合であるという[15]）．

また，詰将棋は芸術作品としての要素が大きく，攻め方に無駄な持駒があってはならない，先手の手順は一意に決まらなければならない等，作品の完全性を定義する様々な決まりがある．詰将棋のルールに関する詳しい説明は，文献 30) を参照されたい．

### 3.2 アルゴリズム

詰将棋プログラムは，指手を枝，局面を節点とするゲーム木を作成し，これを探索することで解を求める．詰みである局面の評価値を “1”，不詰である局面の評価値を “0” とすれば，詰将棋のゲーム木は先手局面において論理和，後手局面において論理積をとる AND/OR 木となる[17]．

本研究では，木探索アルゴリズムとして Seo らによる PN*[32] を採用し，ハードウェアに実装した．PN* は，現在における最長手詰問題である「ミクロコスモス」（1525 手詰）を初めて解くことに成功したアルゴリズムである．PN*は Allis らによる pn-search[1] を発展させた探索法であり，証明数（Proof Number: PN）を閾値とする多重反復深化による縦型探索を行う．証明数とは，AND/OR 木におけるある節点を評価するために，他のいくつの節点を評価する必要があるかを数値で表したものである．証明数の考えは，McAllester による共謀数[23] という概念に基づいている．ある節点の証明数が大きいということは，その節点の評価値を更新するためにより多くの他節点を評価する必要が生じるということである．詰将棋の場合，証明数は後手の自由度とほぼ等しいと考えられる．ゆえに詰将棋における PN*は，後手の応手が少ない指手を優先して展開する手法であるといえる．

反復深化では，評価値が閾値を超えない限り縦型の探索を行う．ある閾値で解が発見されなかった場合，閾値を増やして再び探索を実行する．多重反復深化では，初めて展開された節点において，その先の探索での閾値を最小値に設定し直してから反復深化を行う．多重反復深化では，同じ節点を何度も往来することがあるため，一度展開した節点について重複して演算を実行することを避けるため，ハッシュ表を利用する．

PN*を発展させた探索法として，長井らによる df-pn アルゴリズム[24] がある．Df-pn では，証明数とともに反証数（disproof number）を使用し，現在知られている 300 手詰以上の長手詰問題をすべて解くことに初めて成功した．本研究で作成したハードウェア・システムは，現時点では反証数を用いた探索を行っておらず，今後改良すべき点の 1 つである．

### 3.3 用語の定義

本論文では便宜上，王手，防手および駒の効きをいくつかのカテゴリに分類している．本節において，それらのカテゴリについて説明する．

#### 3.3.1 王手の分類

詰将棋では，先手の指手は必ず王手でなければならない．本研究では王手をその特徴によって以下の 3 つのカテゴリに分類した．

- 直接王手
  ある駒が移動し，かつその駒自身が王手を掛けている場合，その指手は「直接王手」であるとする．
- 持駒王手
  持駒を使用して王手を掛けた場合，その指手は「持駒王手」であるとする．
- 開き王手
  ある駒が移動した結果，飛，角，香のいずれかの効きが後手玉に到達することによって王手が生じた場合，その指手を「開き王手」とする．また，直接王手と開き王手が同時に起きる指手を一般的に「両王手」と呼ぶが，二重に指手が生成されるのを防ぐため，両王手となる場合は直接王手のみが生成される．

#### 3.3.2 防手の分類

後手は，先手によって掛けられた王手を回避する手を指さなければならない．このような後手の応手を「防手」と呼ぶことにする．今回，防手をその特徴によって以下のような 3 つのカテゴリに分類した．

- 玉移動防手
  後手玉が移動することによって王手を回避している場合，その指手を「玉移動防手」とする．
- 捕獲防手

王手を掛けている先手の駒を，後手の駒が取ることによって王手を回避している場合，その指手を「捕獲防手」とする．ただし，指手が重複して生成されるのを防ぐため，玉が先手の駒を取る場合は「玉移動防手」とする．

- 合駒防手
  王手を掛けている先手の跳駒の効きを，後手の駒が遮ることによって王手を回避している場合，その指手を「合駒防手」とする．この防手には，後手の駒が移動する場合と，持駒を使う場合が考えられる．

#### 3.3.3 駒の効きの分類

あるマスがある駒の可動範囲内にあるとき，そのマスには「駒の効きがある」あるいは「駒が効いている」という．本論文では，便宜上以下の 2 つの「効き」を定義した．

- 直接的効き
  ある駒 P があるマス S へ移動できる場合,「S には P の直接的効きがある」あるいは「S には P が直接効いている」という．ただし，単に「効き」といった場合は直接的効きを指すものとする．
- 間接的効き
  ある駒 P が移動することにより，跳駒 Q がマス S へ移動可能な状態になるとき,「S には Q の間接的効きがある」あるいは「S には Q が間接的に効いている」という．

## 4. 詰将棋のハードウェア・アルゴリズム

筆者らの研究の目的は，現時点で利用可能なハードウェアを用いて，従来研究されてきた優れたアルゴリズムを高速に実行するためのハードウェア・システムを構築することである．しかし，既存のソフトウェアのコードをそのままハードウェア記述言語に置き換えていく方法では，高い性能を実現することはできない．アルゴリズムをハードウェアに実装する場合，データの依存関係を考慮し，どのような順番で演算を行うか，またどの演算が並列処理・パイプライン処理可能であるかを注意深く検討する必要がある．さらに，ハードウェアの並列処理性能を最大限に活かすために，ソフトウェアと同じ結果を保証しつつ，ソフトウェアとはまったく異なる方法で計算を行うことが必要な場合もある．

本章では，まず王手および防手がどのような流れで指手が生成されるかを大まかに説明する．次に，ハードウェアにおいて木探索がどのように実現されているか説明する．

### 4.1 王手の生成

ソフトウェアにおいては，後手玉の位置から上下左右，斜め，桂馬方向に向かって盤面をスキャンし，先手の駒が発見された場合に王手を生成するという方法が一般的であろう（盤面のスキャンとは，対象となるマスのデータをメモリから次々と読み出すことである）．この方法をハードウェアで実現することは可能である．しかし，後手玉の位置は一定ではないので，読み出すメモリのアドレスや読み出しの順番が毎回変わるためハードウェアが複雑になる．また，各方向へのスキャンを順番に行っていては高い性能を得ることができない．

われわれのハードウェアでは，王手を生成する前に，まず盤上の先手の駒の効きをすべて調べる．このために，横 1 段の 9 マスのデータを同時に読み出しながら，1 段目から 9 段目および 9 段目から 1 段目に向かって盤面のスキャンを行う．この 2 方向からのスキャンは並列に実行される（2 方向のスキャンが必要な理由は 5 章で述べる）．また，各段のデータはパイプライン回路によって処理されるため，盤面のデータの読み出しは 9 サイクル連続で行われる．この方法では，王手の生成にまったく関係のない駒の効きも生成されるが，ハードウェアにおいては，求める駒の効きの数によって所要クロック数が変わることはない．2.5 節で述べたように，本研究では回路が複雑化するのを防ぐため $9 \times 9$ のロジック・ブロックを用いる方法は採用していないが，上で述べたような 9 マスのデータの並列処理，9 段のデータのパイプライン処理によって高速な演算が実現されている．

駒の効きとは駒の可動範囲のことであるから，駒の効きはすべての可能な指手を表している．しかし，詰将棋においては先手の指手は王手でなければならないため，すべての指手の中から王手となるものだけを抜き出す必要がある．われわれのハードウェアでは，王手のみを選択するために王手マスクを使用する．王手マスクは局面が更新されるたびに新たに作成する必要があり，計算量の観点からは冗長的であるが，マスクの作成は駒の効きの計算と並列に実行されるので，所要クロック数は増加しない．このように，局面に依存しない手法を用いることにより，ハードウェアの制御が単純化され，より高速な動作周波数の実現が可能となる．

駒の効きとマスクが得られた後，これらのデータを用いることで王手のみが生成される．3 章において述べたように，本研究では先手の指手を「直接王手」「間接王手」「持駒王手」に分類し，これらの王手を別々

の回路で並列に生成している．それぞれの王手の生成のためにどのようなデータが必要であるかは，5 章において詳しく説明する．

生成された王手はマルチプレクサを経由して木探索回路へと送られ，スタックに保存される．このスタックをどのような順番で読み書きするかによって，様々な木探索の実現が可能となる．木探索回路によって選択された王手によって局面が更新されると，手番は後手側に移り，続いて防手の生成が行われる．

ハードウェアにおいて王手を生成する手順は，以下のようにまとめられる．

(1) 駒の効きの計算，マスクの生成
(2) 直接王手，持駒王手，開き王手の生成
(3) マルチプレクサを経由した王手のスタックへの書き込み
(4) 木探索回路による指手の決定，局面の更新

### 4.2 防手の生成

防手の場合も，王手生成の場合と同様な盤面のスキャンによって駒の効きを求め，その後各カテゴリの防手を並列に生成する．3 章において述べたように，防手は「玉移動防手」「捕獲防手」「合駒防手」に分類されており，これらが並列に生成される．ハードウェアにおいて防手を生成する手順は基本的に王手の生成と同様であり，以下のようにまとめられる．

(1) 後手の直接効きデータの生成
(2) 玉移動防手，捕獲防手，合駒防手の生成
(3) マルチプレクサを経由した防手のスタックへの書き込み
(4) 木探索回路による指手の決定，局面の更新

### 4.3 木探索アルゴリズム

本研究では PN*アルゴリズムによって木探索を行う．PN*では，先手局面では王手の中から 1 つを選択して展開する．後手局面では，現在の証明数が閾値を超えていない場合は防手の中から 1 つを選択して展開し，証明数が閾値以上である場合はゲーム木を 1 段後退する．ハードウェアでは，このようなゲーム木の展開・後退はステートマシンによって制御されている．**図 1** に，木探索回路におけるステートマシンの状態遷移図を示す．また木探索回路では，ハッシュ表等の指手生成回路とは独立したストラクチャのデータを使用している．使用するデータのストラクチャについては，5.5 節で詳細に述べる．ステートマシンの状態遷移は複雑であるため，ここでは**図 2** を例に簡単な説明にとどめる．図 2 の局面 A は先手局面であり，ステートマシンは BMG の状態にあるとする．また，現在の閾値は 2 とする．図 2 では以下のようにステートマシンが遷移する．

(1) **BMG（王手生成）**
王手の生成を回路に指示する．ここでは ☗3二金 ☗2二金 ☗2二歩成 ☗2二歩不成 の 4 つの王手が生成され，状態は **EXP** へ移る．

(2) **EXP（展開）**
王手の中から 1 つを選択し，ゲーム木を 1 段進める．ここでは ☗3二金 が選択され局面 B に移行し，状態は **WMG** へ移る．

(3) **WMG（防手生成）**
防手の生成を回路に指示する．ここでは ☖1一玉 ☖1二玉 ☖3二玉 の 3 つの防手が生成され，証明数が 3 となり閾値を超える．よって ☗3二金 は現時点では不詰であると判断され，状態は **STP** へ移る．

(4) **STP（生成中止）**
ハードウェアでは，ハッシュ表の参照は指手の生成と並列に実行されている．ハッシュ表には，その節点の証明数と局面データが登録されている．先手局面では，この値が多重反復深化における閾値を超えていた場合に指手の生成を中止し，ただちに次の状態 **FCK** に移る．後手局面では，登録されている兄弟節点の証明数の和が閾値以上である場合に指手の生成を中止し，**FCK** に移る．

(5) **FCK（詰フラグチェック）**
詰フラグは 2 bit のデータであり，その局面が「詰」「不詰」または「未展開」であることを示す．ここではまだ詰が見つかっておらず，かつ未展開の指手があるため，フラグは「未展開」となっており，状態は **NXT** へ移る．

(6) **NXT（節点移動）**
節点を次の兄弟節点に移す．ここでは ☗2二金 が選択され，状態は **UDP** へ移る．

(7) **UDP（局面更新）**
選択された指手をもとに，局面を更新する．ここでは ☗2二金 が実行されて局面 C が生成され，状態は **WMG** へ移る．

(8) **WMG**
防手の生成を回路に指示する．局面 C では，後手がどう指しても王手をかわすことができず，防手は生成されない．このため状態は **NMV** へ移る．

(9) **NMV（指手なし）**
先手局面ならば「不詰」であり，ハッシュ表に $\infty$（証明数の最大値）を登録する．後手局面な

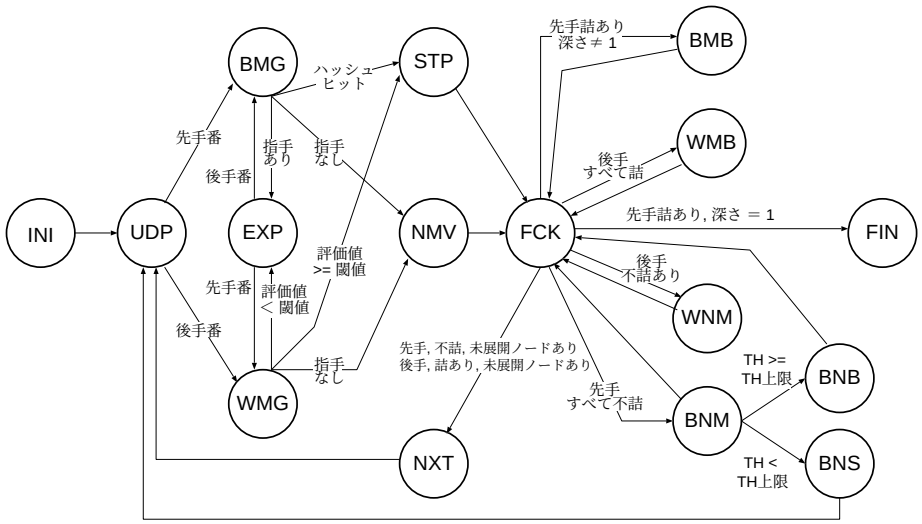



**図 1** 木探索回路の状態遷移図
Fig. 1 The state transition diagram of the tree search circuit.

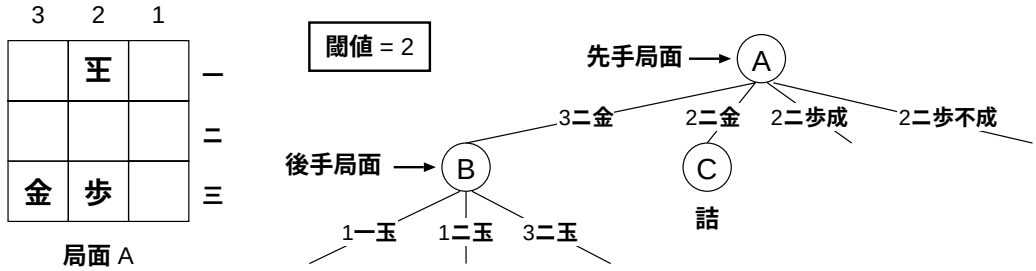



**図 2** 詰将棋の局面の例
Fig. 2 An example position of Tsume-Shogi.

らば「詰」であり，ハッシュ表に 0 を登録する．ここでは後手局面であるから「詰」とし，状態は **FCK** に移る．

(10) **FCK**
局面 A の深さが 1 であり，さらに詰が見つかっているため，状態は **FIN** へ移り計算は終了する．

図 2 ではいくつかの状態が現れなかったが，それらの状態では以下のような処理を行っている．

**初期化（INI）** レジスタ，メモリの初期化を行う．

**BMB（先手詰）** 局面が詰であった場合に，指手データをスタックから読み出して詰フラグデータを記入し，再びスタックに戻す．また，ハッシュ表に証明数 0 を登録する．

**WMB（後手詰）** BMB と同様．

**BNM（先手不詰）** 局面が不詰であった場合に，現在の閾値とその反復における閾値の上限を比較する．

**BNB（先手不詰戻る）** 現在の閾値がその反復における閾値の上限以上であった場合，探索木を 1 段戻る．

**BNS（先手不詰不戻）** 現在の閾値がその反復における閾値の上限未満であった場合，現在の閾値に 1 を加え，兄弟節点の詰フラグをすべて「未展開」に設定し直して再び探索を行う．

## 5. FPGA による実現

本研究ではデバイスとして Field-Programmable Gate Array（FPGA）を使用し，詰将棋専用ハードウェアを実現した．本章では，まず詰将棋ハードウェアにおいて使用されるデータのストラクチャについて述べる．次いで詰将棋ハードウェアの全体像を説明し，その後，王手生成に関連のあるモジュールである「先手直接効き生成回路」「直接王手生成回路」「木探索回路」について説明する．なお，防手生成に関連するモジュールについては本論文では説明を割愛するが，基

**表 1** 各駒のビット割当て
Table 1 Bit assignment of *piece data*.

| 先手 | | | | 後手 | | | |
|---|---|---|---|---|---|---|---|
| 王 | 101000 | | | 王 | 111000 | | |
| 飛 | 100001 | 龍 | 101001 | 飛 | 110001 | 龍 | 111001 |
| 角 | 100010 | 馬 | 101010 | 角 | 110010 | 馬 | 111010 |
| 金 | 100011 | | | 金 | 110011 | | |
| 銀 | 100100 | 成銀 | 101100 | 銀 | 110100 | 成銀 | 111100 |
| 桂 | 100101 | 成桂 | 101101 | 桂 | 110101 | 成桂 | 111101 |
| 香 | 100110 | 成香 | 101110 | 香 | 110110 | 成香 | 111110 |
| 歩 | 100111 | と | 101111 | 歩 | 110111 | と | 111111 |
| 駒なし | 000000 | | | 駒なし | 000000 | | |

本的には王手生成と同様の回路となっている．

### 5.1 データストラクチャ

以下では，詰将棋ハードウェアにおいて使用される主要なデータについて述べる．ここで説明するデータストラクチャは，盤データ，直接効きデータ，直接王手マスクおよび指手データである．その他のデータについては，本論文では説明を省く．

#### 5.1.1 盤データ

盤上のどの位置にどの駒があるかという情報を表すデータを「盤データ」と呼ぶ．今回作成したハードウェアでは，**表 1** のように 1 つの駒を 6 ビットで表す．最上位 bit は駒の有無，5 bit 目は先手/後手，4 bit 目は成/不成，1–3 bit は駒の種類を表している．本研究では，1 クロックで横 1 段（9 マス）の盤データの読み出し/書き込みを行い，並列計算を実現している．1 つの駒は 6 bit，盤のサイズは $9 \times 9$ であるから，盤データは 54 bit 幅の RAM のアドレス 1 から 9 までに保存される．

#### 5.1.2 直接効きデータ

どのマスに対してどの駒の直接的な効きがあるかという情報の集合を「直接効きデータ」と呼ぶ．

あるマスに対して移動可能な先手/後手の駒の数は，持駒を除けばそれぞれ最大 10 枚（上下左右，斜め 4 方向，桂 2 方向から）である．そこで，1 マスに対し 10 個のレジスタファイルを用意し，これに到達可能な駒の種類（成/不成の状態を含む 4 bit）を保存する．また到達可能な駒が跳駒である場合，跳駒の位置を特定するために，角ならば $x$ 座標，香ならば $y$ 座標を同時に保存する．飛車の場合は，左右から移動してくる場合は $x$ 座標，上下から移動してくる場合は $y$ 座標を保存する．$x, y$ 座標はそれぞれ 4 bit で表せるため，1 個のレジスタファイルは 8 bit となる．ただし，桂跳ねの 2 方向から到達可能な駒は桂だけであるから，桂の効きの有無は 1 bit で表す．

以上より，1 マスに対する直接効きデータは 66 bit

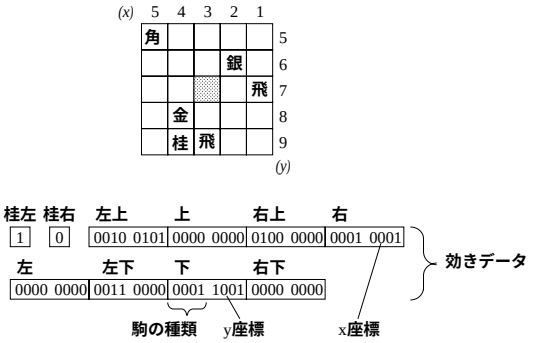



**図 3** 直接効きデータのストラクチャ
Fig. 3 The structure of *direct piece cover data*.

（$= 8 \times 8 + 2 \times 1$）であり，横 1 段では 594 bit となる．ゆえに直接効きデータは，594 bit 幅の RAM のアドレス 1 から 9 に保存される．

ある 1 マスに対する直接効きデータのストラクチャの例を**図 3** に示す．図 3 では，$(3, 7)$ の位置に $(5, 5)$ の角が到達可能である．ゆえに，左上方向に対応するレジスタに，駒の種類を表す “0010” と角の $x$ 座標を表す “0101” が保存されている．他の方向から到達する駒の効きも，これと同様に保存されている．

#### 5.1.3 直接王手マスクデータ

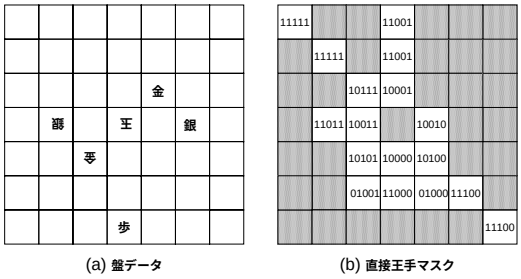
あるマスから見て後手玉がどの位置にあるかという情報の集合を「直接王手マスク」と呼ぶ．後手玉の方向は**表 2** のように数値化されている．1 マスに対し 5 bit が割り当てられているため，横 1 段では 45 bit となる．ゆえに直接王手マスクは，45 bit 幅の RAM のアドレス 1 から 9 までに保存される．

表 2 を用いて作成された直接王手マスクの例を**図 4** (b) に示す．図中の黒いマスは，そこから王手をかけることのできる駒は存在しないことを表しており，実際は “00000” というデータである．

直接王手マスクは，後手玉の位置にすべての駒の可動範囲を合わせ持つ仮想的な駒を置き，この駒の効き

**表 2** 直接王手マスクにおける後手玉の方向の数値化
Table 2 Bit assignment of white's king direction for *direct check mask*.

| 玉の方向 | 隣接するマス | 離れたマス |
|---|---|---|
| 上 | 10000 | 11000 |
| 下 | 10001 | 11001 |
| 左 | 10010 | 11010 |
| 右 | 10011 | 11011 |
| 左上 | 10100 | 11100 |
| 右上 | 10101 | 11101 |
| 左下 | 10110 | 11110 |
| 右下 | 10111 | 11111 |
| 桂左上 | | 01000 |
| 桂右上 | | 01001 |
| 桂左下 | | 01010 |
| 桂右下 | | 01011 |



(a) 盤データ (b) 直接王手マスク

**図 4** 直接王手マスクの例
Fig. 4 An example of *direct check mask*.

**表 3** 指手データのビット割当て
Table 3 Bit assignment of *move data*.

| 名称 | サイズ | 意味 |
|---|---|---|
| sx | 4 bit | 移動元の $x$ 座標 |
| sy | 4 bit | 移動元の $y$ 座標 |
| dx | 4 bit | 移動先の $x$ 座標 |
| dy | 4 bit | 移動先の $y$ 座標 |
| type | 4 bit | 移動する駒の種類 |
| capt | 3 bit | 取る駒の種類 |
| prom | 1 bit | 成/不成の区別 |

を求める方法で得ることができる．仮想的な駒の効きは，具体的には，玉・飛車・角および後手の桂馬の効きを合わせたものである．ゆえに直接王手マスクを求める場合も，直接効きデータを求める際の 9 マスの並列処理と 9 段のパイプライン処理が適用でき，駒の効きの計算と同一の制御回路を利用することができる．

#### 5.1.4 指手データ

どの位置の駒がどこへ移動するか，移動後に駒が成るかどうかという情報を「指手データ」と呼ぶ．相手の駒を取る場合は，その駒の種類も指手データの中に含まれる．指手データのビット割当ては，**表 3** のようになっている．ただし持駒を使用する指手の場合，先手ならば $sx$，$sy$ の値をともに “1110”，後手ならば “1111” とする．

### 5.2 ハードウェアのブロック図

詰将棋回路のブロック図を**図 5** に示す．図 5 中のモジュールにつけられている番号はハードウェア処理のステージを表しており，4.1 節および 4.2 節で述べた手順中の番号に対応している．各ステージのモジュールはすべて並列に動作する．さらに，各モジュールはその内部において並列・パイプライン処理が行われており，高速な指手の生成が実現されている．

上で述べたように，駒の効きと王手マスクは，9 段分の盤データのパイプライン処理を 9 マス並列に実行するという共通の計算方式によって求めることができる．これらの中の「先手直接効きデータ生成回路」のアーキテクチャについて，5.3 節で詳細に述べる．また第 2 ステージの 6 つの指手生成回路でも，9 マス分データを同時に読み出し，パイプライン処理によって 9 段分の指手を次々と生成することによって高速化を図った．これらの中の「直接王手生成回路」のアーキテクチャについて 5.4 節で詳細に述べる．5.5 節では，今回実装した木探索回路のアーキテクチャについて説明する．

### 5.3 先手直接効きデータ生成回路

#### 5.3.1 直接効きデータの生成手順

専用ハードウェアを用いる場合，81 マスの盤データを同時に読み出して，1 クロックですべてのマスに対する効きを求めるのが理想的である．しかし，1 クロックで跳駒の効きを求めるためには複雑なロジックを組まなければならず，遅延時間が大きくなって回路全体の性能を下げる原因となる．今回実装した回路では，同時に読み出す盤データを 9 マス（横 1 段）とし，1 クロックで 9 マスの効きデータを並列に求めている．跳駒の効きは，1 クロック経過するたびに隣接するマスに伝播する．この方法では，すべての盤データを読み出すのに 9 クロックかかるが，複雑なロジックを必要としないため遅延時間が小さく，回路全体は高速に動作する．

ただし，1 段目から 9 段目まで順に盤データを読み出して効きを生成した場合，跳駒の効きは 9 列目に向かって伝播するが 1 列目に向かっては伝播しない．跳駒の効きを求めるためには，盤面を上から下へ，および下から上へ向かってそれぞれスキャンする必要がある．また，飛車の横方向の効きは上下のスキャンでは求めることができない．ゆえに，飛車の発見された段のみ左右方向のスキャンを行う．各方向のスキャンによって，どの部分の効きを求めるかを**図 6** に示す．こ

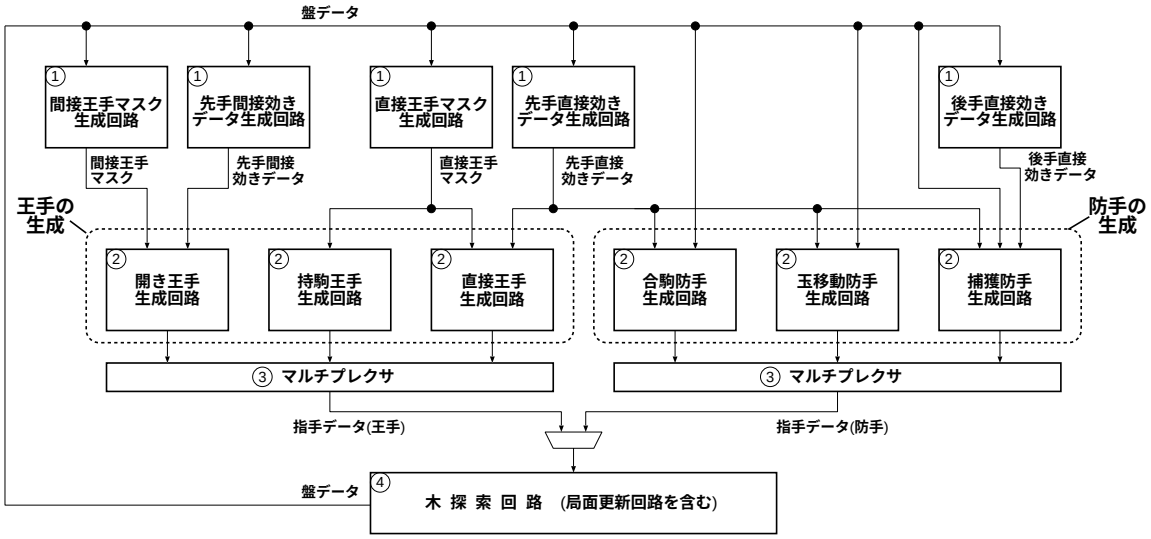



**図 5** 詰将棋回路のブロック図
Fig. 5 The structure of tsume shogi hardware.

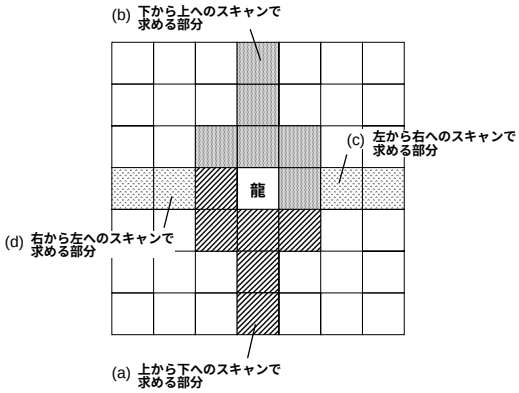



**図 6** 各方向のスキャンで効きが求められる場所
Fig. 6 *Piece cover data* caluculated by each scan.

れら 4 方向のスキャンが終了後，各方向のスキャンで得られた効きデータの論理和を計算することで，盤全体の効きデータを得ることができる．これら 4 方向のスキャンは並列に実行可能であり，処理時間は単方向のスキャンのみを行った場合と変わらない．

駒の効きの計算方法は，以下のようにまとめられる．

(1) 盤面のスキャン
  (a) 盤面を上から下に向かってスキャンし，図 6 の (a) の部分の効きを求める．求められた効きデータは，FPGA 内部のメモリに保存される．
  (b) 盤面を下から上に向かってスキャンし，図 6 の (b) の部分の効きを求める．効きデータは FPGA 内部のメモリに保存される．
  (c) 飛車のいる段を左から右に向かってスキャンし，図 6 の (c) の部分の効きを求める．
  (d) 飛車のいる段を右から左に向かってスキャンし，図 6 の (d) の部分の効きを求める．
  効きデータはレジスタに保持される．
(2) 各方向のスキャンによって得られた効きデータの論理和を計算する

手順 (1) の中で，(a) から (d) のスキャンは並列に実行される．また，(1) および (2) は粗粒度のパイプラインにより実行される．

### 5.3.2 回 路 構 成

先手の直接効きデータ生成回路の構成を**図 7** に示す．図 7 中のモジュールの働きを以下に示す．

- Top Down Scanner
  盤面を上から下へ向かってスキャンする．
- Bottom Up Scanner
  盤面を下から上へ向かってスキャンする．
- Rook Cover Scanner
  飛車の横方向の効きを求める．飛車は 2 枚あり，それぞれの効きを別々の回路で求めるため，Rook Cover Scanner は 2 つある．
  - Left-to-Right Scanner
    飛車の存在する 1 段を左から右へ向かってスキャンする．
  - Right-to-Left Scanner
    飛車の存在する 1 段を右から左へ向かってスキャンする．

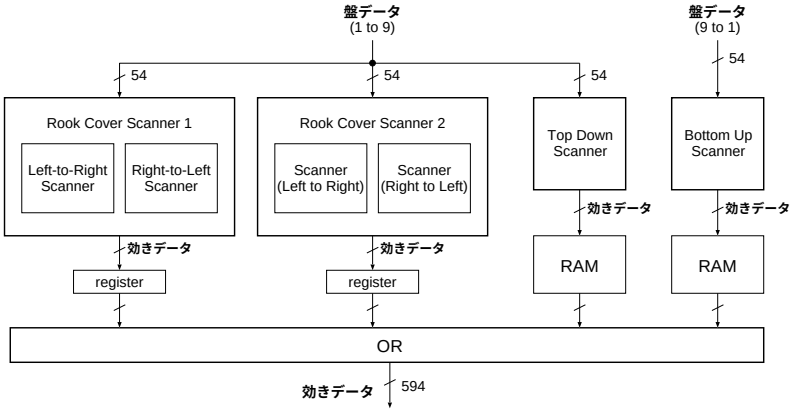



**図 7** 先手直接効きデータ生成回路の構成
Fig. 7 The structure of *Black Direct Cover Generator*.

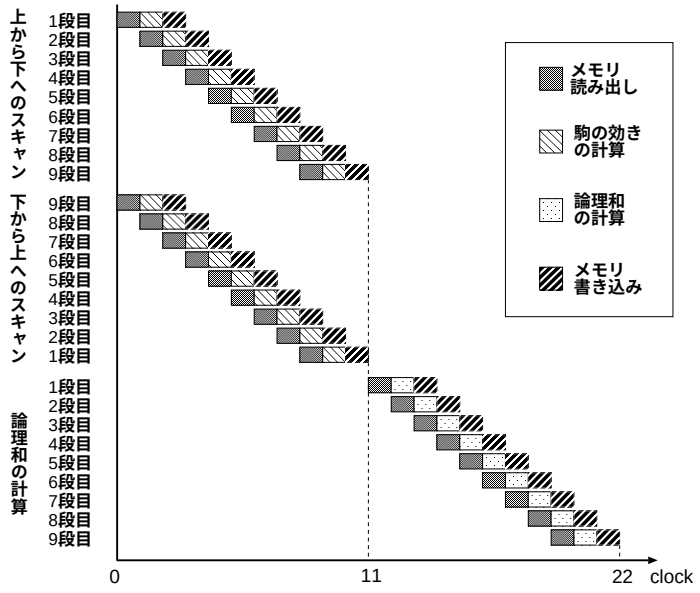



**図 8** 先手直接効きデータ生成回路におけるパイプライン処理
Fig. 8 Pipeline processing in *Black Direct Cover Generator*.

これらのモジュールはすべて並列に動作する．またそれぞれのモジュールはその内部においてさらに並列・パイプライン処理が行われており，高速な演算が実現されている．

### 5.3.3 性　　能

すでに述べたように，駒の効きの計算回路では，複雑なロジックによって動作周波数が下がるのを防ぐために上下左右のスキャンを組み合わせている．これらのスキャンを並列に実行することで，処理クロック数を増やすことなく高速な回路を実現している．また上下のスキャンでは，1 クロックで横 1 段（9 マス）の効きデータを計算している．このように複数のマスの効きデータを並列に計算することで，演算にかかる時間を大幅に短縮している．さらに上下のスキャンでは，**図 8** に示すように，メモリからデータを読み出し，効きを計算してメモリに書き込むという 3 つの行程をパイプラインで処理することにより，9 段分の演算を 11 クロックで完了することができる．

盤全体の効きデータは，各方向のスキャンで得た効きデータの論理和をとることによって得られるが，この論理和の計算も図 8 が示すようなパイプライン処理によって 11 クロックで終了する．ゆえに，盤データが与えられてから最終的な効きデータが得られるまでの時間は，わずか 22 クロックである．

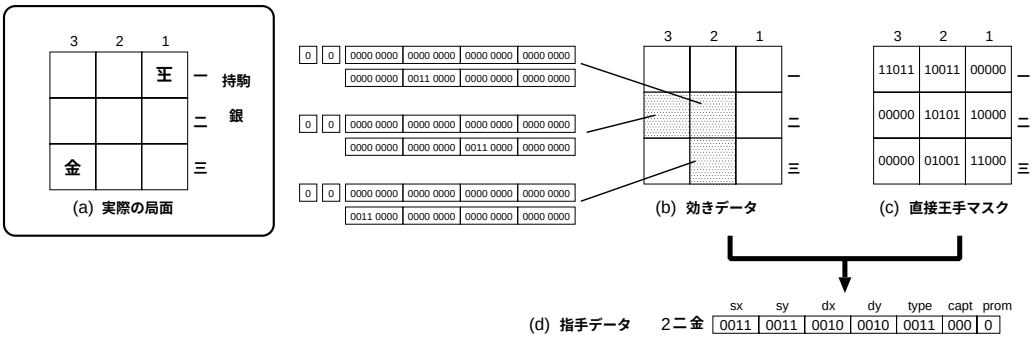



**図 9** 直接王手の生成
Fig. 9 Generation of *direct check*.

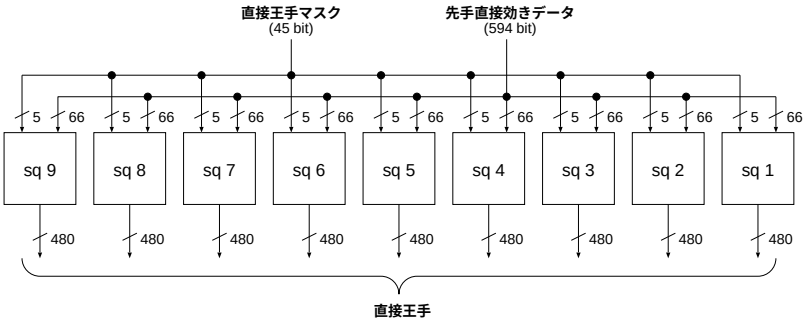



**図 10** 直接王手生成回路の構成
Fig. 10 The structure of *Direct Check Generator*.

### 5.4 直接王手生成回路

#### 5.4.1 直接王手生成アルゴリズム

直接王手は，先手の直接効きデータと直接王手マスクを参照することで生成される．あるマスにおけるこれら 2 つのデータがともに 0 でない場合，直接王手が生成される可能性がある．今，あるマス $(x, y)$ に駒 P の効きが働いているとする．このとき，駒 P の移動可能な方向と後手玉のいる方向が一致すれば，P が $(x, y)$ に移動する手は直接王手である．

例として，**図 9** (a) のような局面が与えられたとする．これに対する先手の直接効きデータおよび直接王手マスクは，図 9 (b)，(c) のようになる．この局面では，直接効きデータと直接王手マスクが共に 0 でないマスが存在し，それらは（2, 二）と（2, 三）である．

（2, 二）の直接効きデータを参照すると，先手の金の効きが働いていることが分かる．また直接王手マスクは “10101” であり，右上のマスに相手玉がいることが分かる．ここで金は右上に移動可能な駒であるから，直接王手として☗2二金 が生成される．

同様にして（2, 三）にも金の効きが働いていることが分かるが，直接王手マスクが “01001” であるため桂以外の駒は王手をかけることができない．よって☗2三金 が直接王手として生成されることはない．

#### 5.4.2 回 路 構 成

直接王手生成回路の構成を**図 10** に示す．図中の *sq X* $(X = 1, 2, \cdots, 9)$ は，それぞれ $X$ 列目のマスに移動する直接王手を生成するモジュールである．これらのモジュールを並列に動作させ 9 マス分の指手を同時に生成するために，回路には 1 クロックで横 1 段の先手直接効きデータ（594 bit）と直接王手マスク（45 bit）が入力される．

図 10 の中の *sq X* には 1 マス分の効きデータとマスクデータが入力される．5.1 節で述べたように，1 マス分の効きデータは 10 方向から到達しうる駒のデータの集合である．これら 10 方向のデータが並列に参照され，到達する駒の移動可能方向と後手玉のいる方向が一致すれば直接王手が生成される．1 マスには 10 方向から駒が到達し，これらの成・不成があるため，*sq X* からはそれぞれ 20 個の指手データが出力される．

#### 5.4.3 性 能

図 10 に示すように，直接王手回路ではきわめて並列度の高い演算が行われており，横 1 段についての直接王手の生成は，直接効きデータと直接王手マスクが入力されてから 2 ステップで終了する．さらに回路で

**表 4** 指手スタックに格納されるデータ（形式 1）
Table 4 The data format 1 stored to the move stack.

| 名称 | 幅 (bit) | 内容 |
|---|---|---|
| データ形式フラグ | 1 | 形式 1 の場合は “0”. |
| 詰手数 | 11 | 詰が発見された時点の探索木の深さ |
| 詰フラグ | 2 | “00” で「未展開」. “01” で「不詰」. “10” で「詰」. |
| 指手データ | 24 | 指手生成回路より送られてくるデータ. |

**表 5** 指手スタックに格納されるデータ（形式 2）
Table 5 The data format 2 stored to the move stack.

| 名称 | 幅 (bit) | 内容 |
|---|---|---|
| データ形式フラグ | 1 | 形式 1 の場合は “1”. |
| 前指手アドレス | 14 | 1 つ前の指手が格納されている場所のアドレス. |
| CN 上限 | 8 | その反復における閾値の上限. |
| 指手数 | 8 | 同一局面から生成された指手の数. |
| padding | 7 | 使用されない. |

は 2 段のパイプライン処理が実現されており，9 段すべてについての演算は 10 ステップで終了する．

### 5.5 木探索回路

ここでは，まず木探索回路で使用されるデータのストラクチャについて述べ，次いで木探索回路の構成について説明する．

#### 5.5.1 データストラクチャ

木探索回路で使用するデータのストラクチャは，実装するアルゴリズムによって異なる．ただし木探索回路は指手生成回路とは独立して動作するため，木探索アルゴリズムを変更する際に，指手生成回路のアーキテクチャを変更する必要はない．本項では，今回実装した木探索アルゴリズムにおいて使用される主なデータのストラクチャについて述べる．

木探索回路では，生成された指手はすべてスタックに格納される．ハードウェアによるゲーム木の展開は，このスタックを制御することによって実現されている．また，実装したアルゴリズムは多重反復深化を行うので，同じ局面を何度も展開することを避けるためにハッシュ表を利用している．以下では，指手スタックとハッシュ表のストラクチャについて説明する．

##### 5.5.1.1 指手スタック

指手スタックは，幅 38 bit，アドレス空間 14 bit のメモリである．指手生成回路から送られてきた指手データは，木探索回路において，節点の深さ，詰の有無といった情報が付加されてスタックに格納される．このとき格納されるデータを「形式 1」とする．また，ある局面における指手の生成が終わると，生成された指手の数，その反復における閾値の上限，1 つ前の指手のアドレスがスタックに格納される．このとき格納されるデータを「形式 2」とする．指手スタックの形式 1 のデータストラクチャを**表 4** に，形式 2 のデータストラクチャを**表 5** に示す．

##### 5.5.1.2 ハッシュ表

多重反復深化では，同じ節点を何度も往来する可能性があるため，同一局面の評価値の計算の重複を避けるためにハッシュ表が利用される．証明数を用いた多重反復深化における局面の評価値とは，先手局面ならば子節点の証明数の最小値，後手局面ならば子節点の証明数の和である．

ハッシュ表には，その局面の評価値,（詰み局面ならば）詰手数が記録されるとともに，圧縮された局面データが記録される．圧縮局面は，ハッシュ表の衝突が起きた場合の誤作動を防止するために利用する．ある局面がハッシュ表にヒットした場合，その局面が登録されている圧縮局面と同一であれば，格納されている評価値を使用する．その局面と圧縮局面が同一でない場合は，格納されている評価値を現在の局面の評価値で上書きする．

ハッシュ表は，幅 108 bit，アドレス空間 20 bit のメモリである．ハッシュ表はより大きな方が望ましいが，現在使用している FPGA ボードのメモリ搭載量の制約によりこの値とした．**表 6** に，ハッシュ表に格納されるデータのストラクチャを示す．

#### 5.5.2 回路構成

木探索回路の構成は**図 11** のようになっている．図 5 に示したように，木探索回路に入力される指手データはマルチプレクサからの出力であり，木探索回路から出力される盤データは第 1 ステージの各回路と第 2 ステージの回路の一部に入力される．各モジュールの動作について，以下で説明する．

**指手スタック** 詰フラグ，詰手数の情報が加えられた

**表 6** ハッシュ表のデータストラクチャ
Table 6 The data structure of the hash table.

| 名称 | 幅 (bit) | 内容 |
|---|---|---|
| フラグ | 1 | 登録データの有無. |
| 評価値 | 8 | その局面の評価値. |
| 詰手数 | 11 | 詰が発見された時点の探索木の深さ. |
| 圧縮局面 | 88 | ハッシュ表の衝突による誤作動を防止する. |

**表 7** 詰将棋回路のリソース使用率と動作周波数
Table 7 Hardware resource usage and frequency of the tsume-shogi circuit.

| モジュール | Slice 使用数 | (使用率) | RAM 使用数 | (使用率) | 動作周波数 [MHz] |
|---|---|---|---|---|---|
| 先手回路 | 13,460/33,792 | (39%) | 66/144 | (45%) | 53.348 |
| 後手回路 | 8,007/33,792 | (23%) | 25/144 | (17%) | 46.553 |
| 木探索回路 | 1,544/33,792 | (4.6%) | 43/144 | (29%) | 69.633 |
| 回路全体 | 23,011/33,792 | (68%) | 134/144 | (93%) | 46.553 |

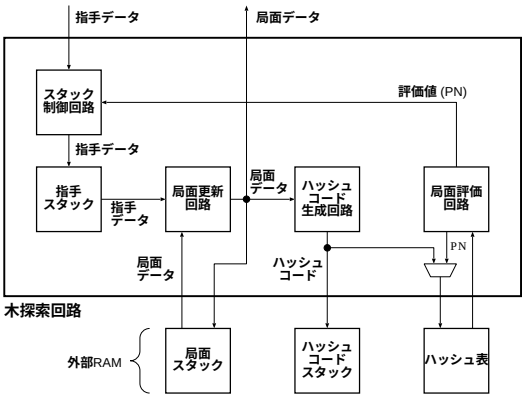



**図 11** 木探索回路の構成
Fig. 11 The structure of the tree search circuit.

指手データを格納するメモリ.

**スタック制御回路** 指手スタックへの入力データ，アドレス，イネーブルの生成を行う.

**局面更新回路** 現在の局面を，入力された指手データによって更新する.

**ハッシュ生成回路** 入力された局面データより，ハッシュコードを生成する.

**評価値計算回路** ハッシュ表を読み出し，現在の局面の評価値を求める.

**局面スタック** 探索木を戻るために，生成した局面を保存しておくメモリ.

**ハッシュコード スタック** 探索木を 1 段戻ってハッシュ表に評価値を登録する際，1 度訪れた局面のハッシュコードを生成し直さないで済むように，ハッシュコードを保存しておくメモリ.

**ハッシュ表** 局面の証明数を登録するメモリ.

## 6. 詰将棋回路の性能

本研究では，Alpha Data 社製の FPGA ボード ADM-XRC-2 に詰将棋回路を実装し，性能の評価を行った．ADM-XRC-2 は，Xilinx 社 Virtex-II シリーズの FPGA である XC2V6000 と，8 バンク 40 MB（6 バンク × 4 MB，2 バンク × 8 MB）の SSRAM モジュールを搭載している[2]．XC2V6000 は，論理回路を実現する Slice 33,792 個，メモリを実現する 18 Kb の Block RAM144 個から構成されている[36]．また，ハードウェアの設計には，Xilinx 社製の EDA ツールである Foundation 4.1 を使用した.

実装された詰将棋回路のハードウェアリソースの使用率，および CAD により報告された最大動作周波数を**表 7** に示す．表 7 中の「先手回路」は，先手の指手を生成するために必要な効き計算回路，マスク生成回路，直接王手生成回路，持駒王手生成回路，開き王手生成回路，およびマルチプレクサを含む回路である．同様に「後手回路」は，後手の指手を生成するために必要な効き計算回路，玉移動防手生成回路，捕獲防手生成回路，合駒防手生成回路およびマルチプレクサを含む回路である.

表 7 が示すように，最新の FPGA を用いることにより，詰将棋という大規模なアプリケーションを 1 チップ上に実装することができた．これはシステムの高性能化と低価格化を考えた場合，非常に重要である．回路全体の動作周波数は，CAD によって 46.553 MHz と求められた．ただし，CAD によって求められる動作周波数は，大量生産されたすべての FPGA 上で当該回路が安全に動作するための最低の数値であり，われわれの FPGA 上では 50 MHz で問題なく動作することを確認した.

**表 8** 詰将棋問題において展開されたノード数，分岐数および実行時間
Table 8 The number of the nodes expanded in the tsume-shogi problems.

| 手数 | 問題数 | 平均展開ノード数 | 先手平均分岐数 | 後手平均分岐数 | SW 平均 [msec] | HW 平均 [msec] | 性能比 |
|---|---|---|---|---|---|---|---|
| 3 手詰 | 2 | 1,940 | 7.24 | 6.22 | 55 | 5.3 | 10.3 |
| 5 手詰 | 9 | 5,494 | 5.04 | 4.38 | 138 | 20.5 | 6.72 |
| 7 手詰 | 17 | 3,282 | 5.59 | 4.47 | 78 | 10.2 | 7.64 |
| 9 手詰 | 23 | 12,452 | 5.23 | 4.65 | 270 | 44.8 | 6.01 |
| 11 手詰 | 23 | 19,901 | 4.98 | 4.66 | 376 | 68.1 | 5.52 |
| 13 手詰 | 17 | 28,389 | 4.84 | 4.79 | 445 | 97.9 | 5.33 |
| 15 手詰 | 6 | 60,889 | 4.46 | 4.20 | 988 | 181 | 5.48 |
| 17 手詰 | 1 | 37,801 | 5.14 | 5.80 | 570 | 114 | 5.02 |
| 全問題 | 98 | 19,874 | 5.14 | 4.64 | 303 | 59.3 | 5.30 |

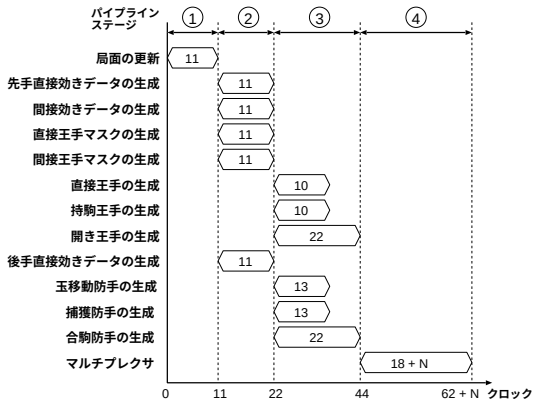



**図 12** 詰将棋回路のタイミングチャート
Fig. 12 The timing chart of the tsume-shogi circuit.

**図 12** は，ハードウェアにおける指手生成のタイミングチャートである．各行程中の数値は演算のステップ数を表し，マルチプレクサの所要ステップ中の $N$ は 1 局面に対する指手の数を表す．マルチプレクサは，各指手生成回路の出力より王手・防手のみを選択しメモリへ送るモジュールであり，この処理は逐次的にしか実行することができないため，所要ステップ数は生成される指手の数に依存する．

本研究では，内藤による詰将棋選集[26] の 100 題の中から双玉問題を除く 98 題をテスト問題として使用し，ハードウェアおよびソフトウェアによるゲーム木のノードの展開能力を比較した．ハードウェアによる解手順は FPGA 上の指手スタックに保存されており，この内容は PC のメモリに送られ，PC 上に表示される．実験の結果，これら 98 題すべてについて，解手順を発見することができた（だたし，7 章で述べるように，完全性のチェックが行われていないため必ずしも最適解とはなっていない．作意と同じ手順が得られたのは全体の 45%であった）．**表 8** に，各手数の問題において展開されたノード数の平均，先手節点と後手節点の平均分岐数，およびソフトウェア（SW）とハードウェア（HW）による実行時間の平均を示す．ハードウェアでの処理時間を PC 上のソフトウェアから測定することは可能であるが，きわめて短時間であるため正確な数値を求めることは難しい．また，実測された処理時間にはソフトウェアとハードウェアの通信時間が含まれるが，通信時間は PC の動作状態等により測定するたびに変わるため，純粋にハードウェア処理のみにかかった時間を算出することは困難である．ゆえにハードウェアの実行時間は，所要クロック数と動作周波数から計算した．専用ハードウェアでは，ある問題に対する所要クロック数は毎回必ず同じであるため，処理時間を正確に算出することができる（本システムでは FPGA ボード上のメモリのみを使用し，PC 上のメモリを使用しないため，同一の問題におけるメモリアクセスのタイミングもつねに一定である）．なお，システムのクロックは 50 MHz とした．

ソフトウェアは性能比較のために独自に作成したものであり，実行環境は，CPU が Pentium4-2.53 GHz，OS は Windows2000 上の Unix エミュレータ Cygwin である．使用したコンパイラは gcc 3.2 であり，最適化レベルは 2 とした．ソフトウェアにもハードウェアと同様の木探索アルゴリズムとハッシュ表が実装されているが，跳駒の位置を覚えて指手の高速化を図るなど，できる限りのチューンナップがされている．ただし，生成される指手の順序およびノードの展開順序はハードウェアとまったく同じくなるように設計されている．今回我々が作成した回路には，木探索を効率化するためのいくつかの手法が実装されていないが，これについては 7 章で述べる．また現段階では，ハッシュ表の大きさの制約や効率化アルゴリズムの未実装により長手詰の問題が解けていないが，ここでは同じアルゴリズムによるノードの展開能力に注目する．

表 8 が示すように，本研究で作成した詰将棋回路はソフトウェアに対して 5 倍以上の性能を得ることができた．またハードウェアのノード展開能力は，1 秒間

あたり約 34 万局面であった．表 8 においては，手数が長いほどハードウェアのソフトウェアに対する性能比が下がる傾向にあるが，先手・後手の平均分岐数を見ると分かるように，長手数の問題では可能な指手が比較的限られているためである（17 手詰の問題はこの傾向の例外であるが，問題数が 1 題のみであるため，より多くの問題を用いて検討する必要がある）．

図 12 より，局面の更新が開始されてから全王手または全防手が生成されるまでのステップ数は $62 + N$ であるから，指手の生成に要する時間は

$$(62 + N) \times \frac{1}{46.553} \qquad (1)$$

となる．Seo らによって求められた詰将棋のゲーム木の分岐数の平均は 5.23 であり[32]，上式 (1) において $N = 5$ を代入すると，指手 5 個の生成にかかる演算時間は 1.44 $\mu$sec となり高速な指手の生成が実現されていることが分かる．

実装した詰将棋ハードウェアでは，並列度の高い演算によって指手の生成が行われており，全指手の生成に要する時間は生成する指手の数にはあまり依存しない．たとえば生成する指手数 $N$ が 10 である場合，ソフトウェアによる逐次処理での所要時間は $N = 5$ である場合の約 2 倍と予想されるが，ハードウェアによる並列処理では，式 (1) よりわずか 1.07 倍増加するのみである．ゆえに，1 局面あたりの指手数が多い複雑な問題ほど，ハードウェアのソフトウェアに対する性能は向上する．詰将棋は芸術作品としての側面があるため，先手に与えられる持駒が必要最低限であり，さらに詰局面において持駒が残ってはならないという制約がある．実際の将棋の終盤では，詰ませるためには必要のない持駒が存在するなど，1 局面あたりに可能な指手数は詰将棋よりも多くなると考えられる．ゆえにハードウェアは，実戦局面においてさらに高い性能を示すと考えられる．

## 7. 今後の課題

本研究で作成したハードウェアは，探索木の展開能力についてソフトウェアに対して優れた性能を示した．しかし，FPGA ボードのハードウェア的な制約や未実装のアルゴリズムがあり，改良を加えることでさらに高い性能が得られると期待される．以下に今後の課題についてまとめる．

今回のハードウェアには木探索の効率化を行ういくつかの機能が実装されておらず，かなり無駄な計算を実行していると考えられる．今後実装することが課題とされるアルゴリズムとして以下のようなものがあげられ，これらの実装によって探索木の展開能力の向上が期待される．

**無駄合判定** 今回のハードウェアでは，無駄合の判定を行っていない．そのため，跳駒がある問題を解くのに時間がかかる場合がある．無駄合判定アルゴリズムは柿木によって報告されている[15]．

**局面の優越関係・証明駒** Seo による PN*を用いた詰将棋プログラムでは，木探索を効率化するために，局面の優越関係の利用や証明駒の利用を行っている[33]．

**反証数・反証駒** PN*を改良した df-pn アルゴリズムが長井らによって報告されている[24]．長井らのプログラムでは，証明数，証明駒とともに反証数，反証駒の概念を用い，探索の効率化を行っている．

また詰将棋においては，先手は最短，後手は最長の手を指さなければならない．現在のハードウェアでは後手の指手はすべて展開されているが，先手に関しては，詰であることが分かった時点で残りの指手の展開を行わない．解の完全性を保証するアルゴリズムが実装されるべきである．

またハードウェア面での改善点として，

- 大容量メモリを使用したハッシュ表のアドレス空間の拡張
- 複数のチップによるゲーム木の並列探索
- 複数の FPGA ボードによる演算システムの構築

があげられる．たとえば，DRAM の拡張ボードの導入によりハッシュ表のサイズを大きくすることで，木探索の性能を向上させることができるであろう．また，ゲーム木の並列探索では使用するハードウェア・リソース量にともなって性能が向上する．近年，大規模な FPGA やそれらを搭載した PCI ボードが安価に手に入るため，複数チップ，あるいは複数ボードを使用した低価格で高性能な演算システムの構築が期待される．

われわれの最終的な目的は，本将棋の専用ハードウェアの開発である．本研究において開発された詰将棋ハードウェアでは，局面の更新回路，駒の効きの演算回路，王手・防手生成回路等の本将棋にそのまま使用可能なモジュールも含まれる．木探索回路のアーキテクチャは，本将棋に応用することが可能であろう．今後は，詰将棋ハードウェアのアーキテクチャを本将棋へ拡張する．

## 8. おわりに

本研究では，将棋専用ハードウェアの開発の第 1 段階として，FPGA を用いた詰将棋専用ハードウェアの

作成を行い，そのアーキテクチャと性能について述べた．本研究では，先手の指手を「直接王手」「持駒王手」「開き王手」，後手の指手を「玉移動防手」「捕獲防手」「合駒防手」というカテゴリに分類しこれらを別々の回路で並列に生成すること，また，これらの指手の計算に用いられる各データの生成回路を並列に動作させることで高速演算を実現した．さらに，それぞれの回路はその内部において並列・パイプラインアーキテクチャが実装されている．これにより，きわめて並列度の高い処理によって高速な指手の生成が可能となった．また FPGA を使用することで，チップ内部に任意の幅の RAM を構成することが可能となり，このような高並列処理におけるメモリバンド幅ネックの問題点を解決することができた．

実際の詰将棋の問題を解いてソフトウェアと処理時間を比較したところ，5 倍以上の性能を得ることができた．ハードウェア処理は，複雑な詰将棋の問題や候補手のきわめて多い本将棋において高い性能を示すことが期待される．

今後，未実装のアルゴリズムのハードウェア化を進めるとともに，詰将棋回路を利用して本将棋ハードウェアの開発を行う．

## 参 考 文 献


1) Allis, L.V., van der Meulen, M. and van den Herik, H.J.: Proof-number search, *Artificial Intelligence*, Vol.66, No.1, pp.91–124 (1994).
2) Alpha Data Parallel Systems: *ADM-XRC-II User Manual, ver.1.5*, Edinburgh, UK (2002).
3) Boulé, M.: An FPGA Move Generator for the Game of Chess, Master's Thesis, McGrill University, Montreal (2002).
4) Boulé, M. and Zilic, Z.: An FPGA Based Move Generator for the Game of Chess, *IEEE Custom International Circuits Conference*, pp.71–74 (2002).
5) Boulé, M. and Zilic, Z.: An FPGA Move Generator for the Game of Chess, *ICGA Journal*, Vol.25, No.2, pp.85–95 (2002).
6) Campbell, M., Hoane Jr., A.J. and Hsu, F.-h.: Deep Blue, *Artificial Intelligence*, Vol.134, No.1–2, pp.57–83 (2002).
7) ChessBase. http://www.chessbase.com
8) Condon, J.H. and Thompson, K.: Belle Chess Hardware, *Advances in Computer Chess 3*, Clarke, M.R.B. (Ed.), pp.45–54, Pergamon Press, Oxford (1982).
9) Hamilton, S. and Lee, G.: Deep Blue's Hardware-Software Synergy, *Computer*, Vol.30, No.10, pp.29–35 (1997).
10) Hori, Y., Seki, M., Grimbergen, R., Maruyama, T. and Hoshino, T.: A Shogi Processor with a Field Programmable Gate Array, *Computers and Games*, pp.297–314 (2000).
11) Hori, Y., Sonoyama, M. and Maruyama, T.: An FPGA-Based Processor for Shogi Mating Problems, *IEEE International Conference on Filed-Programmable Technology*, pp.117–124 (2002).
12) Hsu, F.-h.: A Two-Million Moves/s CMOS Single Chip Chess Move Generator, *IEEE Solid-State Circuits*, Vol.5, pp.841–846 (1987).
13) Hsu, F.-h.: Large Scale Parallelization of Alpha-Beta Search: An Algorithmic and Architectural Study with Computer Chess, Technical Report CMU-CA-90-108, Carnegie Mellon University, Pittsburgh, PA (1990).
14) Hsu, F.-h.: IBM's Deep Blue Chess Grandmaster Chips, *IEEE Micro*, Vol.19, No.2, pp.70–81 (1999).
15) 柿木義一，小谷善行ほか：将棋プログラム K1.5 の思考アルゴリズム，コンピュータ将棋，pp.80–100, サイエンス社 (1990).
16) 笠田洋和，山田雅之，松波功二，世木博久，伊藤英則：詰将棋におけるゲーム木の並列探索とその評価，情報処理学会論文誌，Vol.36, No.11, pp.2531–2539 (1995).
17) 小谷善行，吉川竹四郎，柿木義一，森田和郎：コンピュータ将棋，サイエンス社 (1990).
18) Matsubara, H. and Grimbergen, R.: Differences between Shogi and western Chess from a computational point of view, *Board Game in Academia* (1997).
19) 松原　仁（編）：コンピュータ将棋の進歩，共立出版 (1996).
20) 松原　仁（編）：コンピュータ将棋の進歩 2，共立出版 (1998).
21) 松原　仁（編）：コンピュータ将棋の進歩 3，共立出版 (2000).
22) 松原　仁，半田剣一：ゲームとしての将棋のいくつかの性質について，情報処理学会人工知能研究会資料 96-3, pp.21–30 (1994).
23) McAllester, D.A.: Conspiracy numbers for min-max search, *Artificial Intelligence*, Vol.35, No.3, pp.287–310 (1988).
24) 長井　歩，今井　浩：df-pn アルゴリズムの詰将棋を解くプログラムへの応用，情報処理学会論文誌，Vol.43, No.6, pp.1769–1777 (2002).
25) 長島紀子，中山泰一，野下浩平：ゲーム木の並列探索のための分散共有ハッシュ機構の設計と実現，情報処理学会論文誌，Vol.39, No.6, pp.1581–1586 (1998).
26) 内藤國雄：内藤詰将棋選集，日本将棋連盟 (2002).

27) 中山泰一：並列に詰将棋を解くプログラム — 多数の UNIX ワークステーションを利用して，コンピュータ将棋の進歩 2，松原 仁（編），pp.22–31，共立出版 (1998).
28) 中山泰一，赤澤忠文，野下浩平：ゲーム木の並列探索のための分散的実行管理機構，電子情報通信学会論文誌，Vol.J79-D-I, No.9, pp.572–575 (1996).
29) Newborn, M.: *Kasparov Versus Deep Blue: Computer Chess Comes of Age*, Springer Verlag (1997).
30) 野下浩平：詰将棋を解くプログラム T2，コンピュータ将棋の進歩，松原 仁（編），pp.50–70，共立出版 (1996).
31) 佐藤信弘，新藤雅也，野下浩平，中山泰一：並列ゲーム木探索のための分散共有ハッシュ法の評価，情報処理学会論文誌，Vol.42, No.5, pp.1198–1206 (2001).
32) Seo, M., Iida, H. and Uiterwijk, J.W.H.M.: The PN*-search algorithm: Application to tsume-shogi, *Artificial Intelligence*, Vol.129, No.1–2, pp.253–277 (2001).
33) 脊尾昌宏：詰将棋を解くアルゴリズムにおける優越関係の効率的な利用について，ゲーム・プログラミング ワークショップ，pp.129–136 (1999).
34) Takizawa, T. and Grimbergen, R.: Review: Computer Shogi through 2000, *Computers and Games*, Hamamatsu, Japan, pp.433–442, Springer (2000).
35) 立松靖朗，山田雅之，世木博久，伊藤英則：詰将棋におけるプロセッサ稼働率を考慮したゲーム木並列探索，情報処理学会論文誌，Vol.37, No.9, pp.1745–1748 (1996).
36) Xilinx, Inc.: *Virtex-II 1.5V Field Programmable Gate Arrays v1.7*, San Jose, CA (2001).

(平成 15 年 7 月 4 日受付)
(平成 16 年 1 月 6 日採録)

**堀 洋平**（学生会員）

1976 年生．1999 年筑波大学第三学群工学システム学類卒業．同年筑波大学大学院博士課程工学研究科知能機能工学専攻入学，現在に至る．FPGA を用いた将棋専用ハードウェアの開発/研究に従事．

**斎藤 尚徳**

1980 年生．2003 年筑波大学第三学群工学システム学類卒業．同年（株）ビー・ユー・ジー入社．現在，（株）ビー・ユー・ジー開発本部に所属．ネットワーク組込系機器の開発に従事．

**丸山 勉**（正会員）

1958 年生．1987 年東京大学大学院工学系研究科情報工学専門課程博士課程修了．同年日本電気（株）入社．並列オブジェクト指向言語，並列遺伝的アルゴリズム，並列マシン Cenju の開発/研究に従事．1997 年より筑波大学機能工学系助教授．書き換え可能なハードウェアを用いた計算の高速化に関する研究に従事．