モンテカルロ碁シミュレーション専用ハードウェアの検討

三 好 健 文[†1] 加 藤 英 樹[†1,†2] 竹 内 郁 雄[†1]

専用ハードウェアによってモンテカルロ法による囲碁プレーヤーのシミュレーションエンジンを実現する手法を検討する．専用ハードウェア化することで，内在する並列性を活用した高い性能を実現するこが期待できる．本稿では，ハードウェアによってで囲碁シミュレーションを実現するための手法を検討し，必要となるハードウェアリソース量および信号遅延を評価する．その結果，検討した回路は，現実的なハードウェアリソースで回路を実現することができ，シミュレーション速度は，40 playout/秒である．

Hardware Monte Carlo Simulation Server for Go

Takefumi Miyoshi,[†1] Hideki Kato[†1,†2] and Ikuo Takeuchi[†1]

Hardware Monte Carlo Simulation Server for Go is considered. Specialized hardware can achieve high computational performance by employgin internal parallelism of target application. In this paper, An method for achieving the simulation by hardware is discussed. Here, the required hardware resources and signal delay is also estimated. As the result, the the amount of the hardware resource can be implemented into the actual hardware and the simulation speed is 40 playout/s.

1. はじめに

モンテカルロシミュレーションと UCT アルゴリムを用いたコンピュータ囲碁対局プレーヤは，局面の静的な解析が不要であることから，コンピュータ囲碁の大会において好成績をおさめている．モンテカルロ法では，シミュレーション回数が 2 倍になると ELO レーティングが 50〜100 増加するとの報告があり[1]，シミュレーション回数を増加し，強いプレーヤを作るための各種の高速化手法が考えられている．本研究では，19 路の囲碁の，ゲーム中に存在する高い並列性を活用して専用ハードウェアによってモンテカルロシミュレーションを高速に実行することを目標とする．

既存のシミュレーション速度を考慮し，専用ハードウェアにより 19 路を対象として毎秒 10k から 100k プレイアウトを実現したい．ここで，1 プレイアウトまでの手数を 300 手，ハードウェアの動作周波数が 30MHz のとき，目標とする処理能力と 1 手の計算に許されるサイクル数の関係を**表 1** に示す．

本論文では，このシミュレーション速度を得ることを最終目標として，まずは，モンテカルロシミュレーションに必要なモジュールとその実装手法を検討し，回路の必要とするリソース量および信号遅延を評価する．

表 1 処理能力と 1 手の計算に許されるサイクル数

po/s	サイクル数
1 M	0.1
100 k	1
10 k	10
1 k	100

まず，第 3 節で専用ハードウェア化についての一般論および，実装手段である FPGA および HDL について述べる．第 4 節でハードウェアによって実現するための具体的な手法についての検討を行い，第 5 節で実装した回路の評価結果について示す．

2. 関連研究

専用ハードウェアを用いて強いゲームプレーヤを実現する試みは多数のゲームで行われている．チェスでは，専用 VLSI を搭載した DeepBlue が世界チャンピオンに勝利し，また，将棋では，詰将棋を解く専用ハードウェア[2] の他，A 級指し手リーグ 1 号[3] では完全にスタンドアロンで動作する将棋専用ハードウェアを実現している．一方で，囲碁，特に 19 路盤では回路リソースが大きいことや，終局判定や手の選択が難しいことから，専用ハードウェアによるプレーヤは，まだ多くない．

モンテカルロシミュレーションと UCT アルゴリズ

†1 東京大学大学院情報理工学系研究科創造情報学専攻
Dept. of Creative Informatics, The Univ. of Tokyo
†2 株式会社フィックスターズ
Fixstars Corporation

ムを用いた MC/UCT による囲碁プレーヤは，近年，多く研究されている．これらのプレーヤーでは，マルチプロセッサや，文献[4] のように Cell プロセッサを用いて高速にモンテカルロシミュレーションが実行されている．

3. 専用ハードウェア化

まず，専用ハードウェア化することで得られる効果について述べ，専用ハードウェアの設計方針について述べる．次に，実験として実装に用いる FPGA とハードウェア記述言語について述べる．

3.1 設 計 方 針

一般に，専用ハードウェアによって実現することで，全てのアルゴリズムが，汎用プロセッサ上のソフトウェアで記述された場合と比較して高速になるとはいえない．特に，スタックや逐次的なメモリ操作などでは，プロセッサメーカにより設計された汎用プロセッサ上で動作するソフトウェアの方が，容易に高い性能を得ることができる．一方で，専用ハードウェア化のメリットは，アプリケーションに応じて自由にデータ幅や並列に動作する演算ユニットを実装することができる点にある．本論文では，囲碁のシミュレーション中に含まれるデータ並列性およびパイプライン並列性に着目することで，専用ハードウェア化による高速化を実現する．

3.2 FPGA

フィールドプログラマブルゲートアレイ (FPGA) は，論理回路を構成するための多数のルックアップテーブル (LUT) とフリップフロップ (FF) で構成される再構成可能なハードウェアユニットである．近年の FPGA は，LUT や FF の他に，アドレス幅やデータ幅を構成可能なカスタマイズブロック RAM や分散メモリなどの記憶領域を持ち，柔軟な設計が可能である．FPGA は，起動時に自身のハードウェア構成情報を読み込んで構成することができるため，ASIC の設計と比較して，容易に設計開発を行うことができる．その一方で，回路遅延や回路規模は ASIC で作り込む場合より大きくなってしまう．現在市販されている FPGA では，最大 500MHz 程度で動作させることができる．

3.3 ハードウェア記述言語

FPGA を用いたハードウェア設計には，近年では，ハードウェア記述言語 (HDL) が広く用いられ，特に Verilog-HDL あるいは VHDL がよく使われている．ハードウェア記述言語では，信号幅を自在に記述することができる．また並行に動作する回路素子の記述が容易である．本論文では，手順の説明の擬似コードとして VHDL に似た記述を用いる．

4. 囲碁プレーヤーの実装

本節では，本論文で検討する囲碁プレーヤの実装

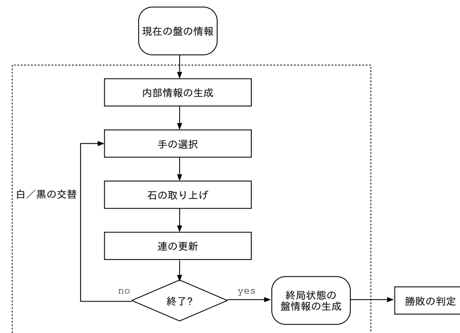

図 1 囲碁プレーヤとハードウェア実装部分

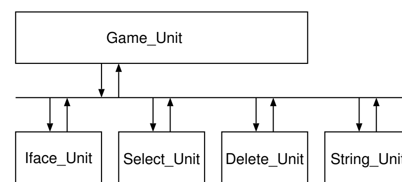

図 2 専用ハードウェアの構成図

について述べる．囲碁のゲームの流れを**図 1** に示す．ここで，本論文では，図 1 中の破線で囲んだ部分を専用ハードウェアによって実現することを考える．専用ハードウェアでは，ゲームの盤情報を与えられた時に，ゲーム終了時点までのゲームをランダムに進める．ゲーム終了状態の盤の情報をホスコンピュータに転送し，勝敗の判定はホストコンピュータによって行う．

ハードウェアユニットとして，実装すべきモジュールは，

(1) ゲーム状態を管理するモジュール (Game_Unit)
(2) ホストコンピュータとのインターフェイス (Iface_Unit)
(3) 手を選択するモジュール (Select_Unit)
(4) 取り上げる石を決定するモジュール (Delete_Unit)
(5) 連の情報を更新するモジュール (String_Unit)

である．各モジュールは，`request` を受けて動作を開始し，動作中は，`busy` 信号をアクティブにする．

ゲーム情報を管理するモジュールを一つにまとめ，そのモジュールでゲームの状態遷移を管理する専用ハードウェア全体の構成図を**図 2** に示す．この構成の場合，各モジュールにおける処理に含まれる並列性を有効に活用することで速度が向上することを期待できる．

一方で，ゲーム情報をストリームとして取り扱う**図 3** に示す構成を考えることもできる．この場合，ゲーム情報を管理するモジュールへのアクセスにおけるボトルネックがなくなるため，パイプライン並列性の活用が期待できる．しかし，ここではハードウェアリソース上の制約から，図 2 に示す，一括してゲーム情報を管理する構成を採用する．

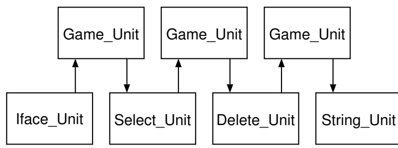

図 3　ハードウェアユニットの構成図

```
signal cWhiteArray: std_logic_vector(360 downto 0);
signal cBlackArray: std_logic_vector(360 downto 0);
signal cSpaceArray: std_logic_vector(360 downto 0);
```

図 4　盤情報を構成するレジスタ

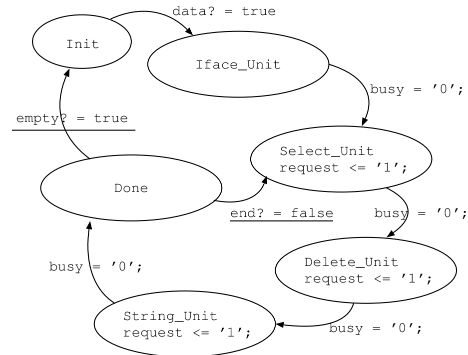

図 6　モジュールの動作を管理する状態遷移

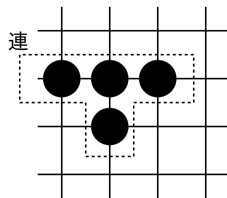

図 5　連

次の小節では，ゲーム情報の管理手法および，各モジュールの詳細について述べる．

4.1　ゲーム情報管理モジュール

ゲーム情報管理モジュールでは，ゲーム盤上の情報および，ゲーム状態のための状態遷移を管理する．ゲーム盤上の情報は，各交点における石の情報および盤上の石によって構成される連の情報から成る．

4.1.1　各交点における石の情報

N 路の囲碁のゲーム盤上の N^2 個のゲーム盤上の交点には，{ 黒石，白石，空 } の 3 状態の可能性がある．各交点の石の状態は，ゲーム中で頻繁に使用する．従って高速にアクセスすることができるようレジスタとして実装することを考える．ここで専用ハードウェアでは，レジスタ幅を自由に決定することができることを利用する．盤上の交点数 N^2bit 幅のレジスタを 3 本用いることで，上記の 3 状態をワンホットエンコーディング，すなわち各状態に 1bit を割り当てて表現することができる．19 路の盤を考えるとき，全ての目の状態を表現するためには，各状態に対応した 361bit 幅のレジスタを用意すればよい (**図 4**)．　交点 (x, y) に対応する石の状態は，各レジスタの $y \times 19 + x$bit 目を検査することで得られる．

4.1.2　連 の 情 報

囲碁のゲームを進めるために，連の情報は，石の取り上げや合法手の判定を行う際に大変有用である．連は**図 5** に示すように，盤上で縦横に継がっている石の集合を指す．　従って N 路の囲碁における，連の最大個数は最大でも N^2 個であり，また連に属する石の個数もまた，最大 N^2 個である．従って，19 路の場合，各連の情報として

連の周囲の空点 (呼吸点) の個数　9bit
連に属する石の個数　9bit
連の色　4bit
連に属する石の座標　9bit

を考え，ユニークな識別子 (ID) による連情報の保持領域を 361 エントリ用意すればよい．また，各交点に置かれた石が，どのエントリの連に属しているかを決定するためのテーブルがあると，石の取り上げや連の更新に有用である．これは，連の個数が最大で N^2 個であることから 19 路の場合，各交点の属する連に対して 9bit のレジスタで表現できる．

4.1.3　ゲーム状態の管理

ゲーム状態として管理すべき情報は，図 2 に示した各モジュールを，順次アクティブにするための状態遷移および，白/黒の手番，取りあげられた石の個数である．各モジュールの動作は，`request` と `busy` によって制御できる．ここでは，**図 6** に示す状態遷移によって各モジュールの動作を管理する．

図 6 中の各楕円が状態を示している．各状態は，各モジュールの動作に対応し，各状態遷移直後に，対応するモジュールの `request` をアクティブにし，モジュールが動作中であることを示す信号である `busy` が 0 になるのを待つ．初期状態は，`Init` で，ホストコンピュータからデータが入力されると，`Iface_Unit` に状態を遷移させる．`Iface_Unit` で入力データをデコードし，内部に格納後，`Select_Unit`，`Delete_Unit`，`String_Unit`，`Done` のループでゲームをシミュレーションする．ここで状態 `Done` では，ゲームの終了状態を判定する．終了状態でなければ (`end? = false`)，再び `Select_Unit` に状態は遷移する．終了状態は，白と黒が連続して 2 回パスした場合，あるいはシミュレーションを規定回数繰り返した場合とする．ここで，白と黒の手番を交代させる．終了状態の時 (`end? = true`) には，最終局面をホストコンピュータがアクセス可能なメモリに格納し，`Init` に遷移する．

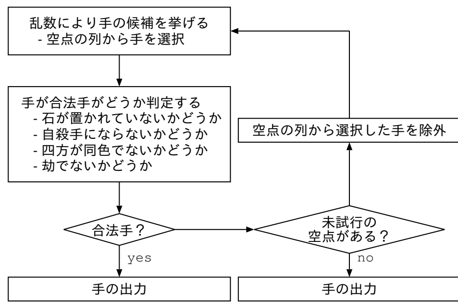

図 7　手の選択

4.2 手の選択

ランダムでゲームをシミュレーションするためには，乱数による手の候補の選択と，囲碁として許される手の選別の必要がある．本論文では，手の選択を**図 7** に示す手順で行う．

手の選択では，まず乱数で手の候補を得る．乱数の生成には，ハードウェアでも記述が容易な XORSHIFT 法[5]を用いる．得られた乱数を使い，空点の列から手を選択する．次に，この手が合法であるかどうかを判定する．手が合法である場合には，その手を出力して手の選択を終了する．合法でない場合，未試行の空点が存在するならば，選択した手を空点の列から除外して，再び乱数による手の選択に戻る．ここで，未試行の空点が存在しない場合，パスを選択する．

4.3 合法手判定器

まずは，簡単にゲームをすすめることができることを目標に，次の条件を満たす手を合法手とする．

- まだ石が置かれていない交点
- 自殺手ではない
- 劫をとる手ではない
- 四方を同色の石に囲まれない．

最後の条件は囲碁のルールとして定められている事項ではない．しかし，このルールを加えないと，自身の目をつぶしてしまい現実的なシミュレーションを実現することができない．

ここで，候補となる交点における石の有無および四方の交点の状態に関しては，レジスタ中の該当する bit の値を検査することで判定できる．また，自殺手でないかどうかの判定は，四方の各交点に対して，同色か同色でないかの条件に従って，**図 8** に示す手法で判定できる．四方の全てにおいて自殺手の可能性がある場合，その候補は自殺手である．

4.4 石の取り上げ

ゲームを進めるためには，置かれた石に応じて，適切に石を取り上げる必要がある．石の取り上げは，自殺手の判定と同様に，置かれた石の四方の連の呼吸点の個数によって判定することができる．すなわち，四方の点について**図 9** に示す判定手法を適用する．

石の取り上げは，各連がその連に属している石の座標をビット列として保持しているため，そのビット列を検査することで取り上げ対象の石を得ることができる．取り挙げ対象の石の座標の状態レジスタを適切な値にセットすることが，取り上げの処理である．iStoneArray が，取り上げ対象の連に属する石の座標を保持するビット列，BlackArray，WhiteArray，SpaceArray がそれぞれ取り上げ後の各座標の交点の状態を示すビット列，pBlackArrayOut，pWhiteArrayOut，pSpaceArrayOut がそれぞれ現在の各座標の交点の状態を格納するレジスタであるとするとき，取り上げの処理は，**図 10** に示す処理となる．ここで，各石，すなわち各ビットに対する処理には依存関係がないため，専用ハードウェアとして実装する場合には，このループは完全に展開できる．

```
if 盤外である
  自殺手の可能性有．
elsif 同色の石がおいてある場合
  if その石の属す連の呼吸点 = 1 then
    -- 呼吸点が一つしかない．自殺手の可能性有
  else
    -- 呼吸点が複数あるので自殺手ではない
  end if;
elsif 異色の石がおいてある場合 then
  if その石の属す連の呼吸点 = 1 then
    -- 呼吸点が一つ．取り上げられる (=自殺手ではない)
  else
    -- 呼吸点が複数．取り上げられない (=自殺手の可能性有)
  end if;
elsif 石のない点 then
  自殺手ではない
end if;
```

図 8　自殺手の可能性を判定する擬似コード

```
if 「石が同色でない」 かつ 「石の属す連の呼吸点数 = 1」 then
  この連に属す石を取り上げる
else
  石の取り上げはない
end if;
```

図 9　石の取り上げを判定する擬似コード

4.5 連情報の更新

ゲームの進行に伴う最後に連の情報の更新を行う．更新の対象となる連は，

連の統合　打った石の周辺の連で同色の連
連の更新　連の呼吸点の個数の更新

である．まず連の統合について述べ，次に更新について述べる．

4.5.1 連の統合

図 11 に連の統合についての例を示す．図 11 中，丸印のように石が配置された盤上で，黒がその手番において，星印の交点に石を打つとする．このとき，独立する連 A および B は，共に，星印におかれた石と同じ連に属すこととなる．これを連の統合という．

```
for i in 0 to 360 loop
  if iStoneArray(i) = ’1’ then
    -- 取り上げの対象
    BlackArray(i) := ’0’;
    WhiteArray(i) := ’0’;
    SpaceArray(i) := ’1’;
  else
    -- 取り上げの対象ではない
    BlackArray(i) := pBlackArrayOut(i);
    WhiteArray(i) := pWhiteArrayOut(i);
    SpaceArray(i) := pSpaceArrayOut(i);
  end if;
end loop;  -- i
```

図 10　石の取り上げを判定する擬似コード

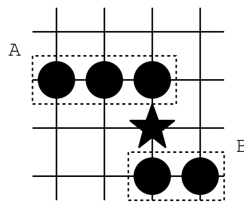

図 11　連の統合

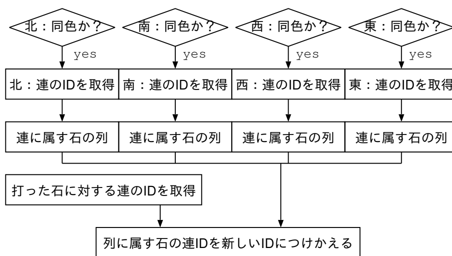

図 12　連の統合の手順

図 12 は，ハードウェアユニットによる連の統合の手順を示す．まず，四方の石が打った石の色と同色であるか判定する．同色である場合には，その石の属する連に含まれているすべての石が統合の対象となる．各石が属している連の ID は，連の情報として保持されているテーブルを引くことで得られ，さらに，得た連の ID から，その連に属している石の列が得られる．この列に含まれる石が連の統合の対象である．新しい連の ID は，打った手の座標 (x, y) から $y \times 19 + x$ で一意に定められる ID とする．打った石および対象となる石を新しい連に登録し，打った石および対象となる石が属す連を新しい連の ID としてテーブルに登録する．

4.5.2　連 の 更 新

連の統合，石の取り上げによって，連周辺の空白の交点の個数が変化する．**図 13** に示す例では，星印の位置に手をうつことで，囲まれた白丸は取り上げられ，A，B および C の連はそれぞれ連周辺の空点の個数の情報を更新する必要がある． 連周辺の空点の個数を

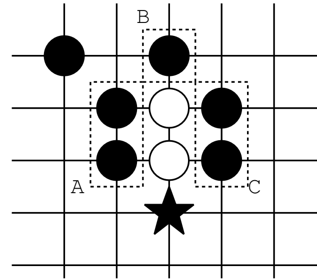

図 13　連の更新の例

```
vDame := 0;
vSize := 0;
for i in 0 to 18 loop
  if i > 0 then
    vSlideN := pStoneArray(19*i-1 downto 19*(i-1));
  end if;
  if i < 18 then
    vSlideS := pStoneArray(19*(i+2)-1 downto 19*(i+1));
  end if;
  vSlideW := "0" & pStoneArray(19*(i+1)-1 downto 19*i+1);
  vSlideE := pStoneArray(19*(i+1)-2 downto 19*i) & "0";
  for j in 0 to 18 loop
    if (vSlideN(j) = ’1’ or
        vSlideS(j) = ’1’ or
        vSlideW(j) = ’1’ or
        vSlideE(j) = ’1’) and
       pSpaceArray(i*19+j) = ’1’ then
      vDame := vDame + 1;
    end if;
    if pStoneArray(i*19+j) = ’1’ then
      vSize := vSize + 1;
    end if;
  end loop;  -- j
end loop;  -- i
```

図 14　連周辺の空点の個数の数え上げ

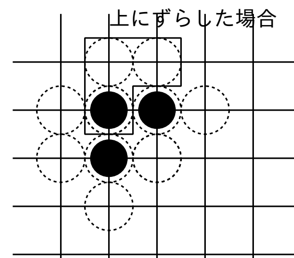

図 15　連周辺の空点の個数の数え上げの例

数え上げる手順を**図 14** に示す． この処理は，更新の対象とする連を上下左右に 1 点ずつシフトし，シフトした位置における空点の個数を数え上げる．**図 15** に例を示す．黒丸が対象とする連で，破線の丸は，図 15 中の「上にずらした場合」のように，上下左右にずらした石を示している．ここで，破線のうち，空点の個数は 7 個で，これは連の持つ呼吸点の個数に等しい．ここで，ループ本体の処理にはループをまたいだ依存関係を持たないため，このループは完全に展開され並

表 2　XC5VLX50 の持つリソース

項目	値
スライス数	7200
ロジックセル数	46080
CLB Flip-Flops	28800
分散 RAM(Kbits)	480
ブロック RAM(各 32Kbits)	48

表 3　合成結果

項目	使用数	最大数	使用率
スライスレジスタ数	7,806	28,800	27%
スライス LUT 数	16,154	28,800	56%
スライス数	6,559	7,200	91%
メモリ使用数	306	1,728	17%

表 4　信号遅延

項目	遅延 (ns)
ロジック	27.7
配線	142.3
合計	170.0

列に処理される．

5. 評　　価

Xilinx Vertex-5 ファミリの XC5VLX50[6] に実装することを仮定し，必要となるハードウェアリソース量や信号遅延について評価する．実装には VHDL を用い，合成および配置配線は，Xilinx ISE Webpack 10.1[7] を用いて行った．XC5VLX50 の持つリソースを**表 2** に示す．

実装した回路の必要とするリソースを**表 3** に示す．連の情報および，各石が属する連の ID を保持するテーブルは，リソースの都合により，すべて RAM として実装した．そのため各連へのアクセスは排他的である必要があり，全ての連にアクセスするためには 400 サイクルの時間が必要になる．

また，信号遅延を**表 4** に示す．この結果より実装した回路は 5MHz 程度で動作する． 今回は，設計を容易にするために各モジュールを粗に結合したことで，配線による遅延がかなり大きい．

以上の実装結果より，今回検討した手法では，ハードウェアリソース量の点では，FPGA を用いて専用ハードウェアを実装することが可能であることがわかった．ただし，リソース量の都合により，連の情報をメモリを用いて保持していることから，各連へのアクセスは逐次的に行う必要がある．そのため，すべての連にアクセスするためには，最小でも 500 サイクル程度の時間が必要となる．また，検討した回路では，信号遅延が大きく，動作周波数を高くできないことが分かった．この実装の場合，1 プレイアウトにかかる時間は，25m 秒程度であり，プレイアウト速度は，毎秒 40 程度である．

6. ま と め

本論文では，モンテカルロ法に基づく囲碁プレーヤの試行部分を専用ハードウェアによって実現するために必要な回路ユニットおよびそれらの実装方法について述べた．検討した回路を VHDL を用いて記述し，Virtex-5 を対象に実装することで，回路規模および信号遅延についての評価を行った．この結果，検討した回路は，ハードウェアリソース量は FPGA 上で実装することができる程度であり，また，その信号遅延は 170n 秒であった．これは，1 プレイアウトに必要とする時間が 25m 秒程度で，毎秒あたりのプレイアウト数 40 程度に相当する．

今後の課題としては，まずモジュール設計の最適化により信号遅延を小さくし，動作周波数の向上を行う必要がある．また，現在の回路では，まだ十分パイプライン並列性をひきだしていないため，これを考慮することで速度の向上を目指す．さらに，スタンドアロンでの動作を実現するために，考慮していないルールの追加および勝敗判定モジュールを実装することが必要となる．手の絞り込みや UCT 探索と組み合わせを行うことで強い囲碁プレーヤの実現を目指す．

参 考 文 献

1) Dailey D. scalability studies with uct. *computer-go mailing list*, 2006.
2) 堀洋平, 斎藤尚徳, 丸山勉. 詰将棋専用ハードウェアの作成. 情報処理学会論文誌, Vol.45, No.3, pp. 1014–1031, 20040315.
3) A 級指し手リーグ 1 号. `http://aleag.cocolog-nifty.com/`.
4) 加藤英樹, 竹内郁雄. 並列 mc/uct アルゴリズムの実装. ゲームプログラミングワークショップ 2007(GPW2007).
5) François Panneton and Pierre L'ecuyer. On the xorshift random number generators. *ACM Trans. Model. Comput. Simul.*, Vol. 15, No. 4, pp. 346–361, 2005.
6) Virtex-5 マルチプラットフォーム FPGA. `http://japan.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/index.htm`.
7) ISE WebPack ソフトウェア. `http://japan.xilinx.com/ise/logic_design_prod/webpack.htm`.