東海大学紀要情報通信学部
vol.7,No2,2014,pp.9-16

論文

NSLによるBlokus Duo Multi Game AIシステムの実装

玉城 良[*1], 清水 尚彦[*2]

Implementation of Multi Game AI system of Blokus Duo on FPGA with NSL

by

Ryo TAMAKI[*1] and Naohiko SHIMIZU[*2]

(received on Sep.22, 2014 & accepted on Jan.13, 2015)

Abstract

Blokus Duo is a game that two players move pieces on a board of 1414. In the game for two players Blokus Duo, researchers have studied a Game AI from the past years. In general, the game AI is structured by using an game tree search of algorithm. There is the one of issue of game tree search of algorithm is a complicated procedure of game tree. In order to improve this issue, we designed a Game AI system on an FPGA by using game tree search of an $\alpha-\beta$ pruning. This system is designed by using NSL(Next Synthesis Language), and it can design the complicated procedure. For a high-speed, we implemented a parallelized a calculation of the Game AI. As a result, an average calculation time of the process of the three moves game tree search have been 8.24 seconds.

Keywords: *FPGA, NSL(Next Synthesis Language), Blokus Duo, Alpha-Beta Pruning*
キーワード： *FPGA, NSL(Next Synthesis Language), ブロックスデュオ, $\alpha-\beta$ 法*

1. はじめに

チェスや囲碁，将棋などの人工知能分野の中の Game AI の研究は古くから研究されており，1997 年には IBM の Deep Blue[1] と呼ばれるチェス専用のスーパーコンピュータが当時のチェス世界チャンピオンと対戦し，勝利している．コンピュータチェスの研究を経て囲碁・将棋などのゲームが研究され，その中でも近年では，Blokus Duo と呼ぶゲームが盛んに研究されている[5][6][7]．Blokus Duo は 14×14 のサイズの盤面上で行う二人零和有限確定完全情報ゲームである．このゲームは盤面の組み合わせが有限であるため，完全な先読みが可能である．このような二人零和有限確定完全情報ゲームの AI は，ゲーム木と呼ぶデータ構造を探索し，終局まで先読みをして自分が勝利するような指手を選択し，ゲームを導くことによって，完全に勝利することができる．Blokus Duo におけるゲーム木の複雑さは 10^{70} から 10^{80} と試算されている[2]．ゲーム木の複雑さとは，1 試合の内の探索しなければならない局面の数を指す．近年のゲームシステムでは 1 秒間に 100 万局面以上 (10^6) を読むことが可能になってきている．しかし，上述のゲームシステムで完全な先読みを行うためには，限りなく計算時間を要するため，この課題に対する研究が行われている．

[*1] 情報通信学研究科　情報通信学専攻 (修士課程)
Graduate School of Information and Telecommunication Engineering, Course of Information and Telecommunication Engineering, Master's Program
[*2] 情報通信学部 組込みソフトウェア工学科 (教授)
School of Information and Telecommunication Engineering, Department of Embedded Technology, Professor

関連する研究として，以下のように，Blokus Duo Game AI の実装の報告が挙げられる．いずれも，FPGA(Field Programmable Gate Array) 上にシステムを実装している．

- Sugimoto らは，高位合成 Cyber Work Bench を使用し，高速な Blokus Duo Solver を実装している [3]．
- Liu C らは，モンテカルロ法による，play-out[4] を行って最善手を算出している [5]．
- Jiu Cheng Cai らは，高位合成を利用して，複雑な評価関数を多く用意することで，探索空間を狭めても強い Game AI を実現している [6]．
- Kojima らは $\alpha-\beta$ 探索をする二つの CPU コアを載せた Game AI を設計している [7]．

高位合成言語やソフトウェアを用いて Game AI を設計する利点は，ソフトウェアレベルで逐次的にアルゴリズムを記述することで設計が容易になることや，複雑なアルゴリズムを実装しやすいという点である．しかし，ソフトウェアレベルでの実現のみだと，Game AI の計算性能は上がりづらいため，クロックを意識した設計による，高速化が必要となる．

我々の手法では，動作記述言語 NSL(Next Synthesys Language)[9] を用いたゲーム木探索の実現と，クロックを意識した並列探索による高速化を実現する．ゲーム木探索では，再帰的処理により，木の探索が行われるため，ソフトウェアによる実現が一般的であるが，我々は独自のハードウェアスタックを用意することでゲーム木探索を実現する．ハードウェアによるゲーム木探索アルゴリズムの実現の利点は，ハードウェアが基本的に並列での動作となるため，処理の要所に並列的な要素を加えられる点である．NSL は特徴として，Verilog HDL や VHDL などのレジスタ主体の設計と異なり，動作を主体した抽象的な

記述が出来る．抽象的な記述は，複雑なアルゴリズムの設計を助けている．

本研究の目的は，動作記述による Game AI の実現と，それによる高速化の手法の確立を目的としている．高速化を行うことは，Game AI の計算性能を上げ，探索空間が広くなることによって，いずれ Game AI がゲーム木を完全に解くために貢献するためである．

本論文では，2 章で，探索法について紹介し，3 章で，Blokus Duo に，基本の方針としての Blokus Duo に関するデータ・手続き・評価関数について述べる．4 章では，ゲーム木探索を実現する上で，必要になるデータやモジュールについて解説する．5 章では，実際のゲーム木探索の設計について，制御方法やモジュール構成，システムの構成を解説する．6 章では，実装した結果と Game AI の計算性能について述べ，最後にまとめを述べる．

本実装の Blokus Duo Game AI は ICFPT(The International Conference on Field-Programmable Technology) のルール [8] に則った仕様となっている．ICFPT では Game AI を FPGA 上に実装しお互い交互に指手を出し合いながら対戦する．ルールに至っては付録で述べている．インターフェースには RS232C シリアル通信を採用した．実装方法に関しては，CPU や IP コア (Altera の Nios II プロセッサ，Xilinx の MicroBlaze MCS 等)，専用ハードウェアなど自由に実装してよい．一方，使用デバイスは制限されており，100MHz 前後の動作周波数の FPGA を載せた Board が指定されている．

実装ツールとして，動作記述言語 NSL, 動作合成に nsl2vl, シミュレーションに gtkwave, vvp, iverilog, 論理合成には Quartus II 13.0，動作テストとして Blokus Duo GUI[10] を用いた．

2. 探索手法

二人零和有限確定完全情報ゲーム (以下二人ゲームとする) において，ゲーム木の複雑さの問題が挙げられている．問題に対して，二人ゲームの Game AI は一定の深さまでの探索，すなわち，先読みした局面において，盤面の評価計算によって，各プレイヤーの有利・不利を判断して，自分に有利にゲームを進めていく．探索の打ち切りと盤面の評価は，実用的な計算速度と Game AI の対戦の強さを求めることができる．

本研究では，ゲーム木の探索手法として，minmax 法を基本とした $\alpha-\beta$ 法のアルゴリズムをハードウェアとして設計していく．ここでは，ゲーム木探索と minmax 法，$\alpha-\beta$ 法について述べる．

2.1 ゲーム木探索

ゲーム木探索は Fig.1 のように相手プレイヤー (Opponent) の指手を含んだ盤面を Root Node(Depth=0) として探索していく．Root Node = Opponent Node であり，Opponent Node の子ノードは必ず Player Node となり，Player Node の親子のノードは Opponent Node となるように，交互に局面を探索していく．ノードが Leaf Node(Depth=3) となったとき，局面の評価を行い，親のノード (Opponent Node) に戻る．木の末端 (Leaf Node) から親ノード (Opponent Node) に戻ったとき，次局面の子ノード (Player Node) に枝分かれする．あるノードにおいて，配置可能な指手の候補が無くなった時点で親ノードに戻り (Depth-1)，Depth=1 の状態で配置可能な指手の候補が無くなった場合，探索を終了する．

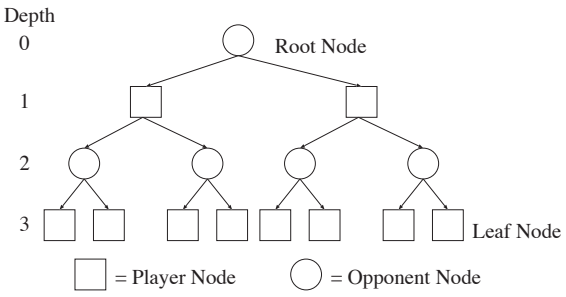


Fig.1 Game Tree Search

2.2 minmax 法

ある局面が自分にとって有利であるかを評価することで，ゲームを有利に進める minmax 法と呼ぶ手法がある．minmax 法ではゲーム木を辿っていき，評価を行ったノードで，先読みを打ちきる．相手の指手を評価する時は，相手にとって不利な盤面になるかを評価する (最小評価)，また自分の指手を評価する時は，自分にとって有利な盤面かを評価する (最大評価)．先読みの数が多ければ多いほど，自分にとってゲームを有利に進めることができる．

2.3 $\alpha-\beta$ 法

$\alpha-\beta$ 法とはゲーム理論における探索アルゴリズムのひとつで，minmax 法の最大最小評価を基本とした探索をする．$\alpha-\beta$ 法の最大の特徴としては，同じような計算結果になる部分のノードを枝刈りすること (部分木の枝刈り) によって，処理数が削減され，結果的に計算速度が高速になることである．加えて，minmax 法と近似した探索結果が得られるため，非常に有用である．Blokus Duo に $\alpha-\beta$ 法を適用した場合，最大で探索すべき局面を平方根まで下げられ，10^{80} の平方根，10^{40} まで探索量を削減できる [2]．

3. 設計の方針

設計の方針として，ハードウェアにおけるゲーム木探索の実現，ゲーム木探索の高速化の順で設計を実施する．Blokus Duo におけるゲーム木探索を実現するためには，ゲームの基本となる盤面やピースのデータと，それらのデータを操作する手続き，評価関数が必要となる．本章では，ゲームを構成するための基本的なデータや手続き，評価関数について述べる．

3.1 ゲームのためのデータ

Blokus Duo では 14×14 マスの盤面上で，21 種類の重複しない形の異なる 1〜5 個の正方形を連結したピースを，お互い交互に出し合って対戦を行う．Blokus Duo の Game AI を構成するためには，基本的に以下に列挙するようなデータが必要となる．

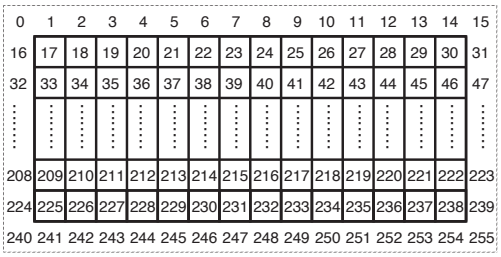
- Board Memory:盤面
- Piece:ピース
- PieceAvailable Memory:各ピースの使用状態

- Move(PieceNum,Rotate,X,Y):指手 (ピース番号, 回転状態, X 座標，Y 座標)
- PlayerNum:プレイヤー番号 (1 or 2)

Board Memory はお互いに交互に出し合った Piece を保持するために用いる．Blokus Duo の盤面は 14 × 14 サイズをとっているが，盤外を Game AI に認識させるために，16 × 16 のサイズのメモリを Board Memory として用意する．16 × 16 サイズにした結果，Y の 0〜15 の値に対して 4bit の左シフト演算をすることで，Board Memory への参照が容易になった．Board Memory への値の参照は Fig.2 上に示されるように，左上からアクセスしていく．Board Memory は一次元配列として用意しており，NSL 記述で値を格納する際は，

$$Board[(y << 4) + x] := DATA; \tag{1}$$

のように行う．Board の内部データは各データが 4bit のビット幅をとり，Fig.2 下のように，盤外を 3, 空きマスを 0, プレイヤー 1 を 1, プレイヤー 2 を 2 として扱う．



Board Memory Map for Blokus Duo

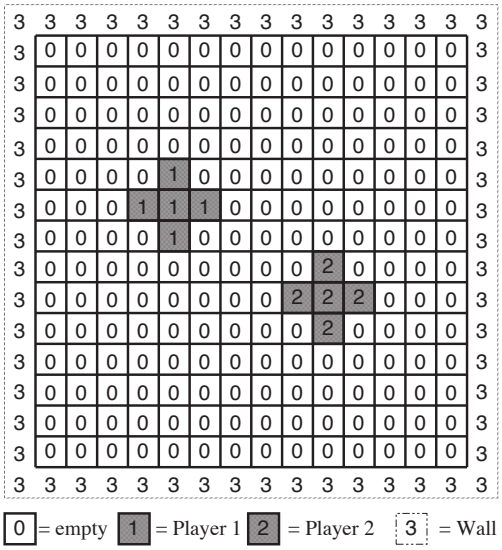
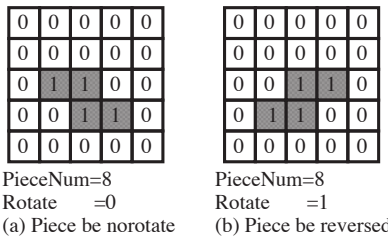


Fig.2　Index map of the board memory for Blokus Duo

Piece は Board Memory に指手を保持する際の実際のピースである．25bit の Fig.3(a) のような無回転状態のピースデータを 21 種類用意し，PieceNum(0〜20) と Rotate(0〜7) を入力するとパターンテーブルによって，Fig.3(b) のように回転状態を適応した Piece が生成される．25bit のうち，0 は空きマス，1 を Piece として扱い，ピースアドレスが 1 を指すとき，各プレイヤーの数字を Board に書き込む．


PieceNum=8
Rotate　=0
(a) Piece be norotate

PieceNum=8
Rotate　=1
(b) Piece be reversed

Fig.3　Example for rotate the pieces

PieceAvailable を 21bit のメモリで用意し，各 bit で 0 を未使用，1 を使用済みとする．PieceAvalable の 21bit はピースの種類の数 (21 種類) を用意したため，1 つのピースの使用状態を 1bit で判断している．指手をサーチする処理の初めに，このデータを読むことで，enable か disable を判断し，指手サーチの処理時間を削減することが出来る．PieceAvailable の値を変更するときは，NSL 記述で，以下のような排他的論理和を用いて，反転したいビットを PieceNum を用いて参照する．

$$Avail[player-1] := Avail[player-1] \bigoplus (0b1 << PieceNum); \tag{2}$$

3.2　Game AI の基本的な手続き

ゲームを正しく進めるために，基本的な手続きを設計する．以下に列挙する基本的な手続きは，動作の追跡のために NSL の構文要素である手続き (Procedure) 記述により設計する．手続き記述は状態遷移やパイプライン，順序回路を用いた制御を提供する構文である [9]．

Select Move:

カウンタにより Move を生成する．Move は x 座標，y 座標，ピース番号，回転状態から成り立つ．Move の Value range として，PieceNum(0〜20),Rotate(0〜7),X(0〜14),Y(0〜14) をとる．

Check Move:

生成した Move が配置可能かチェックする．

Add Node:

配置可能な Move を Piece として Board Memory に追加する．ピース使用状況を更新する．

Evaluation:

ゲーム木の末端 (Leaf Node) の Board Memory を評価する．ノードが Leaf Node の時以外は，評価を行わない．

Remove Node:

配置した Piece を削除する. Board Memory の状態を親ノードに戻す．ピース使用状況を前のノード状態に戻す．

Game AI は上記の各手続きを中心に動作する．ゲーム木探索において，14×14 盤面で 21 種類のピースと 8 種類の回転状態を考慮し，3 手先を 2 プレイヤが読む場合，$\alpha - \beta$ 法によって枝刈りがされないとき，探索すべきノードは最大で約 37800^3 となる．そのため，非常に多い頻度で手続きが動作するため，各手続きにかかるクロック数を限りなく少なくする必要がある．

3.3　評価関数

二人ゲームでは，ゲーム木が非常に大きいため，全ての組み合わせを完全に探索することはできない．二人ゲームの Game AI

は，木を一定の深さまで探索したら，局面に対して，勝ち負けでなく有利不利を計算するような静的評価を行う．評価値はプレイヤーの局面が有利であれば大きい値，不利であれば小さい値になる．

我々は有利不利の判断をする評価関数として，Fig.4 に示すような，勢力範囲計算を採用した．勢力範囲計算では，評価計算のために Board Memory と同じサイズの Evaluation Memory を用意する．勢力範囲は Piece を 2bit 左シフトした値を Influence として Evaluation Memory に書き込むことで実現する．Influence の範囲は付録で述べている Blokus Duo の制約に則って，Corner を中心として，上下左右に 2 つ，全斜め方向に 1 つとる．評価値は Influence の個数の合計を計算する．

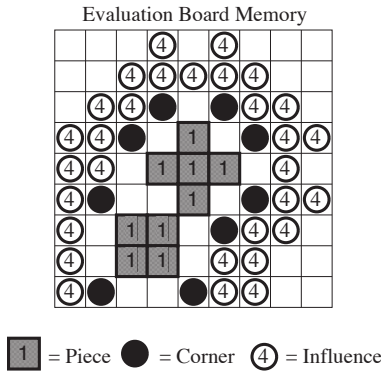


Fig.4　Influence on the Evaluation Memory

4.　ゲーム木探索設計

ある局面に対して，何手か先読みをすることでゲーム木を作成することが出来る．Game AI はゲーム木の末端で局面を評価し，指手の良し悪しを判断する．ゲーム木探索は評価した値を用いて，ルートノード (元となる局面) に対して，最善手を決めるアルゴリズムである．本章では，ハードウェア Game AI におけるゲーム木の設計，および $\alpha - \beta$ 法の設計について述べる．

4.1　ゲーム木探索の設計

ハードウェア動作によるゲーム木探索には 3 章で述べた手続きとゲームのためのデータ，加えて，ゲーム木探索のためのデータが必要になる．ゲーム木探索のためのデータとは Depth, Counter, StackMemory を指す．これらのデータはハードウェアによるゲーム木探索で重要な役割を果たす．

Depth:

Depth はゲーム木探索内でのノードの深さ状態を表す．Depth をインクリメントすることでノードを子ノードに移すことを表し，デクリメントで親ノードに移すことを表す．

Counter:

Move(指手) の生成には Counter モジュールを用い，探索の範囲は Counter の状態で決まる．Fig.5 のように，14×14 盤面の 3 手先を 2 プレイヤが読む場合には，出力として PieceNum,Rotate,x,y を持つ，3 つのカウンタを用意し，入力した Depth の値に応じた Counter が起動する．すなわち Counter[Depth] となる．Counter の内部では X(0〜14), Y(0〜14), PieceNum(0〜20), Rotate(0〜7) の順にカウントする．Rotate のカウントが終了した時点で，Counter は出力信号 fin を High にする．出力信号 fin が High ならば，ノードを親ノードに移す．

Stack Memory:

ノードを子ノードに移す場合，Add Node 手続きによる処理 (Board に Piece を追加する) を行う．また，親ノードに移す場合，追加した指手の情報を保持しておかなければならない．ノードを親ノードに移す処理のために，Fig.6 に示す Stack Memory を用意する．Stack Memory は $(Depth + 1) \times 4$ のサイズをとり，Move(X, Y, PieceNum, Rotate) を格納していく．スタックポインタには Depth を使用し，Depth を始点とした相対アドレスを用いて Move を格納する．親ノードに移る場合，Delete Node 手続きを行う．そのとき，Stack Memory から削除したい Move を取り出す．取り出した Move の PieceNum と Rotate から Piece を生成し，Board に Piece を empty(=0) として入力し，Depth をデクリメントすることで，削除手続き，及び親ノードへの移動が完了する．

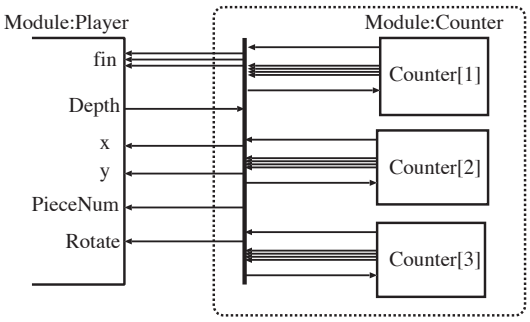


Fig.5　Multiple Counter Modules

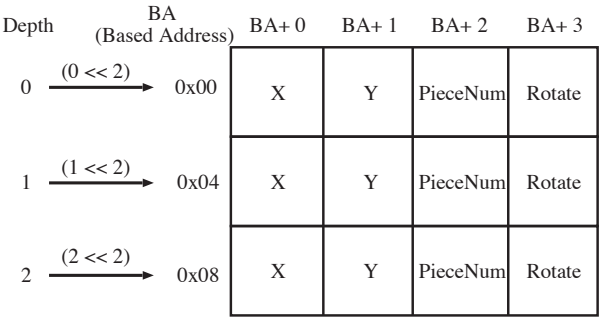


Fig.6　Stack Memory for Bloku Duo

4.2　$\alpha - \beta$ 法の設計

minmax 法は一定の深さまで，すべての指手を探索するため先読み数が増える度に探索空間が指数的に増大する．そのような問題を克服するために，$\alpha - \beta$ 法では，評価値が近似しているノードの枝を刈る．ノードの枝刈りには α カットと β カットを用いる．評価値が β より大きい場合，最善手として β と Move を更新し，ノードを親の親に移す．親の親にノードを移した後，次局面の子ノードに枝分かれをする．これは β による枝刈りである．β カットのみを用いて探索することを NegaMax 探索と

呼ぶ．NegaMax 探索ではお互いに最大の評価値のみを求めるため，β 値を越える評価値が算出されるまで探索をし続ける．ここで，最大探索を打ちきるために，α カットを導入する．もし，評価値が α より小さい場合，枝刈りを行う．α は β の更新された値を受け継ぐ．したがって α は β にとっての最小値となり，

$$(EvaluatedValue < \alpha) \bigvee (\beta < EvaluatedValue) \quad (3)$$

が成り立ったとき，枝刈りを行う．また，最善手の更新，及び最大値の更新は評価値が β より大きいときのみ行う．

5. ハードウェア Game AI の設計

ゲーム木探索は各手続きと必要なデータによって状態を遷移する．NSL の手続き記述 Procedure 構文は状態遷移によるフロー制御機能を提供する．

我々は Fig.7 に示した状態遷移を制御して $\alpha-\beta$ 法を適応した Game AI を設計した．Fig.7 の状態遷移図は，ゲーム木探索のためのデータと手続きによるゲーム木探索の状態遷移を示しており，手続き内の具体的な処理は，図中では省略する．

5.1 状態遷移によるゲーム木探索の制御

ここで，状態遷移によるゲーム木探索について述べる．Opponent Move が入力された時，ゲーム木探索を開始するために，相手の指手を Board Memory に追加 (Root Node の作成) する．Opponent Move から Select Move へ遷移する時，Depth をインクリメントする (子ノードに移す)．Select Move では Piece の生成を行い，枝刈り状態 Pruing と Counter モジュールの fin 信号を遷移条件とする．Select Move の遷移条件がすべて Low の場合，Check Move へ遷移する．Check Move では Piece のチェックを行い，Board Memory へ配置可能 (enable) なら Add Node，配置不可能 (disable) なら Select Move へ遷移する．Add Node では Board Memory に Piece を書き込み，Depth をスタックポインタとして指手を Push する．Evaluation では，$Depth == Leaf$ の場合，評価値計算を行い，Delete Node へ遷移する．また，$Depth < Leaf$ の場合，Depth をインクリメントして，Select Move へ遷移する．

枝刈り操作は Select Move の遷移条件を用いる．Select Move の遷移条件の Pruing, Counter[2].fin, Counter[3].fin が High の場合，Pruing を Low にして，Delete Node に遷移する．Delete Node では EvaluationValue と alpha,beta が条件を満たした時，Pruing を High に，Depth をデクリメント，Pop を行い，Select Move へ遷移する．深さ 1 のカウンタ (Counter[1]) の出力信号 fin が High ならば，Send Move へ遷移する．Send Move では評価値や alpha, beta, カウンタなどを初期化し，最後に上位モジュールに Move を出力しゲーム木探索を終了する．

5.2 ゲーム木探索モジュールの構成

ゲーム木探索を行うモジュールの構成を Fig.8 に示す．モジュール内部ではそれぞれローカルメモリ (Board Memory, Backup Board Memory, Available Memory, Stack Memory) を用意する．それぞれのローカルメモリは 3, 4 章で述べた通りである．また，新たに加えた Backup Board Memory は Add opponent move 手続きを行う時に Root Node の盤面状態を保持しておき，最後に

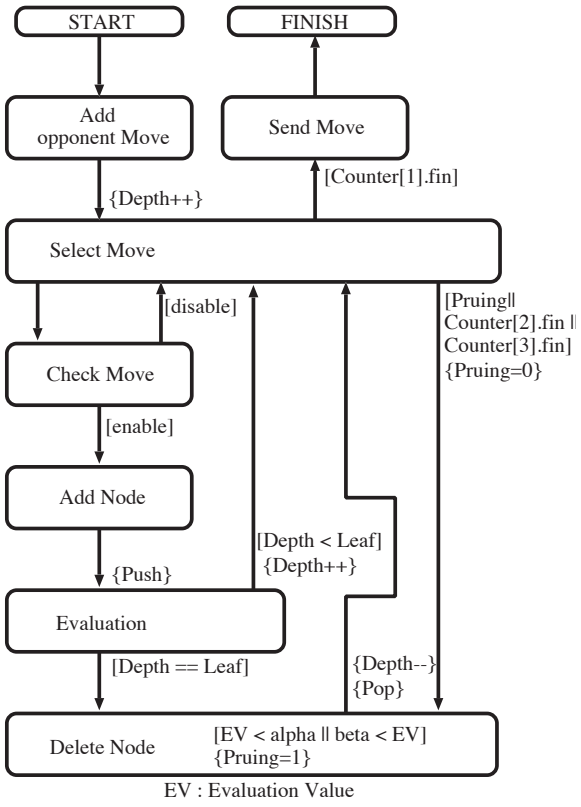


Fig.7　State transitin diagram of Game AI

最善手の Move を追加するための Board Memory のバックアップとして使用する．

それぞれの手続きは上記のローカルメモリを用いて計算を行う．Evaluation モジュールは 3 章で述べた EvaluationMemory をローカルメモリとして用意している．

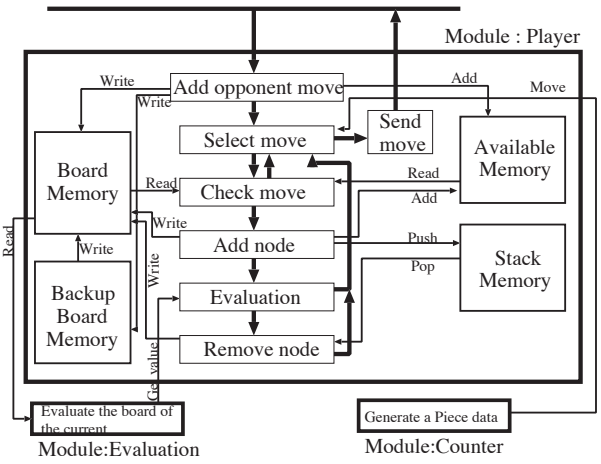


Fig.8　Block diagram of Player module

5.3 システム設計

我々は $\alpha-\beta$ 法に基づいた探索を行う Game AI を Player モジュールとして設計した．Player モジュールをコアとして組み込んだ Game AI システムを，ブロック図として Fig.9 に示す．本システムは RS232C シリアル通信をインターフェースとして

搭載しており，シリアル通信を経由して他プレイヤーとの対戦を行う． 今回，他プレイヤーと対戦するための host システムとして，Blokus Duo GUI を使用した． Blokus Duo GUI は，PC 上で動作するソフトウェアプログラムである． Blokus Duo GUI は，Blokus Duo の対戦用インターフェース機能だけでなく，指手を選ぶ時間を計測する機能を持っており，Game AI システムのテストだけでなく，計算速度測定にも用いた． 動作速度測定は，シリアル通信の送受信に要する時間も含める． また，シリアル通信の通信速度は 115, 200[bps] である．

Not Accelerated Single Game AI システム

計算速度テストの結果，Fig.9 に示した Single Game AI システムの計算速度は，先読み回数 (=3) の時，1 試合における平均計算時間は 502[s] となった．

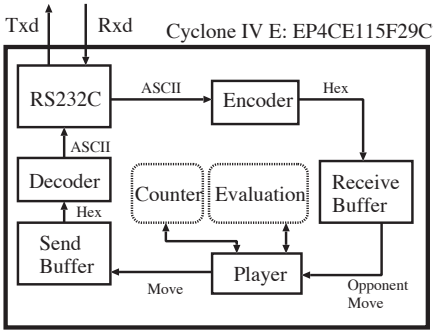


Fig.9 Block diagram of Single Game AI on FPGA

Accelerated Single Game AI システム

我々は，先読み回数 (=3) の時の，計算時間を実用的な速度にするために，Game AI システムの高速化を行った． 初めに，処理のボトルネックとなったのが，Evaluation モジュールである．Evaluation モジュールはローカルな EvaluationMemory を用意しており，Fig.10(a) のように，局面の評価のために Player モジュールの Board Memory から盤面データを 256 クロックかけて読み出しを行っている． 加えて，評価中に EvaluationMemory を走査する処理で 256 クロックかけており，Evaluation の処理には約 520 クロック必要となる． Evaluation は局面の数だけ処理を行うため，クロック数の削減は高速化処理のために必要である．

我々は Evaluation のクロック数の削減のために，Fig.10(b) のように，Board Memory を読み出すための入力端子を 16 本用意した． Board Memory の読み出し用入力端子を 16 本用意した結果，Evaluatoin にかかるクロック数は約 280 クロックに削減した．

また，Add Node の時の，Board Memory への書き込みに Piece のサイズ分ループしていたため，毎回 25 クロックかけて書き込みを行っていた． 書き込み処理に対しても，書き込み用端子を 25 本用意し，ループを展開したことによって，25 クロックから 1 クロックにクロック数を削減した． 信号線を増やしたことによるループの展開を行った結果，先読み回数 (=3) の時の，1 試合における平均計算時間が，Not Accelerated Single Game AI システムのときは 502[s] だが，今回は 36.13[s] となり 13 倍の速度向上となった．

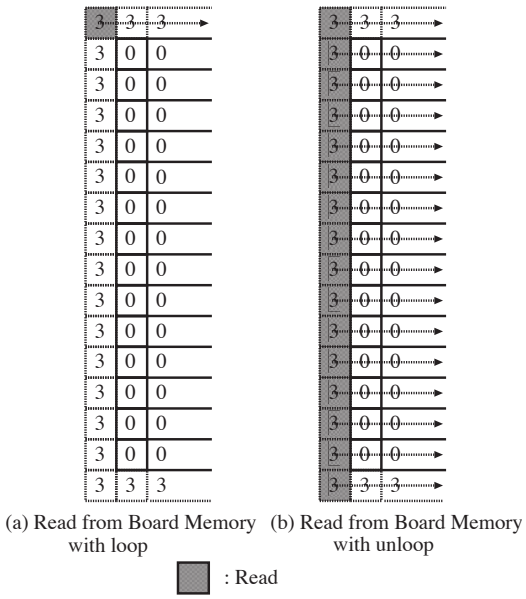


Fig.10 Read from Board Memory

Multi Game AI システム

我々は，計算時間をさらに短縮するために Fig.11 に示すように，Game AI の並列化を行った． 今回, 3 つの GameAI を FPGA 上に搭載することで，GameAI の並列処理化を実現した. Multi Game AI システムでは各 Player モジュールで探索範囲を分担しており，今回はゲーム木を 3 分割し，並列に探索した． Multi Game AI システムでの指手の選択は最終的に Compare に評価値を送り，それぞれ値を比較し更新していくことで指手を選択する． Compare は評価値を各 Player モジュールの算出した評価値を比較し，最大，もしくは最小となる評価を求め，その評価に対応した指手を更新する． 各 Player モジュールはそれぞれ，最後に選択した指手がお互いに異なるため，Compare 処理が終了した時，最善手となる指手を各 Player モジュールの Board Memory に書き込むことが必要である．

Multi Game AI システムを実装した結果，先読み回数 (=3) の時，平均計算時間を約 8.24[s] に短縮した．

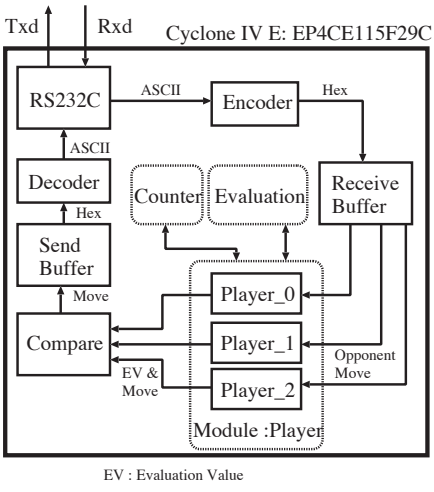


Fig.11 Block diagram of Multi Game AI on FPGA

Table 1　Result of Implementation

Device	Cyclone IV E EP4CE115	
Accelerated Single Game AI	Total logic elements	21,291/114,480(19%)
	Dedicated logic registers	4,246/114,480(4%)
Multi Game AI	Total logic elements	96,411/114,480(85%)
	Dedicated logic registers	7,426/114,480(6%)

6. 実装結果と性能

6.1 実装結果

我々は動作合成言語 NSL を用いて，Altera Cyclone IV E EP4CE115F29C7 上に Game AI システムを実装した．論理合成には Quartus II version 13.0 を用いた．Game AI システムは 50MHz で動作する．複数の Game AI をコアとして組み込んだ Multi Game AI システムと，単一の Game AI をコアとした Single Game AI システムを論理合成した結果を，Table.1 にデバイスのリソース使用率として示す．LE(logic elements) は Cyclone シリーズにおける，組み合わせ回路や順序回路などを構成する論理リソースの最小単位である．Single Game AI システムの実装結果として合計 21,291 の TLE(Total logic elements) を使用した．また，Multi Game AI システムでは Single の約 4 倍となる合計 96,411 TLE を使用した．TLE の結果から，Player モジュールはシステムの中で多くのリソースを使用することを示す．加えて，Counter モジュールと Evaluation モジュールは Player モジュールの下位モジュールになっており，Player モジュールと同じ数の Counter と Evaluation モジュールを用意した．この下位モジュールと Player モジュールの構成は，更なるリソース使用率の増加の原因になっている．本手法では，計算速度の高速化に向けて，より多くの Game AI を FPGA に搭載するために，リソースの使用率を抑えるべきである．リソース使用率を抑えるために，Player モジュールの構成を見直す必要がある．加えて，実装したシステムは並列計算のために単純に Player モジュールを複数用意したが，複数用意しなくてもよい処理を選定し，Game AI 内部の構成を見直す必要がある．

6.2 システム性能

実装したシステムの平均計算時間を Fig.12 に示す．Fig.12 の各システムは，それぞれ先読み (=3) まで探索する．Fig.12(a) の高速化しない通常の Single Game AI システムの平均計算時間は 502[s] となっている．Fig.12(b) の Accelerated Game AI システムの平均計算時間は 36.13[s] となり，(a) のシステムを大幅に高速化する結果となった．この結果は，ひとつのノードに対する処理を短縮したことによって，処理全体の性能が上がったことを示す．Fig.12(c) の並列でゲーム木を探索する Multi Game AI システムの平均計算時間は 8.24[s] となり，ほぼ人間の思考速度と同等の応答性能になったため，実際に対戦ができるレベルまで，Game AI を高速化することができた．

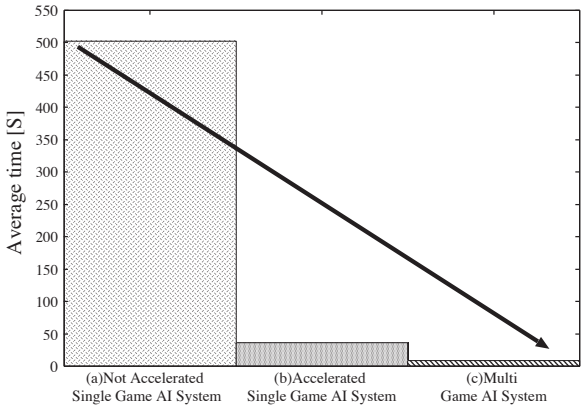


Fig.12　Average calculation speed of the systems

7. おわりに

我々は Blokus Duo における Game AI システムの実装について述べた．Game AI の探索アルゴリズムとして，$\alpha-\beta$ 法を用い，ゲーム木探索を行った．ゲーム木探索のアルゴリズムは動作合成言語 NSL の手続き記述によって，容易に実現できたことを確認した．動作時間測定の結果，Game AI システムを高速化するためには，ひとつのノードに対する処理時間を短縮することが，ゲーム木探索の処理性能を上げることを示した．さらに，Game AI の計算性能の向上のために，複数の Game AI を並列で動作させる Multi Game AI システムを実装した．Multi Game AI システムは並列動作によって，3 手読みを平均計算時間 8.24[s] という実用的な速度になった．

6 章で述べた通り，Multi Game AI システムは多くのリソースを使用した．リソース使用の増大はシステムの不具合となる問題の他，本システムの計算性能を越えるハードウェアを設計するためにも，アーキテクチャの再構成が必須である．

また，評価関数の精度や，ゲーム AI の強さについては議論していない．本稿の結果から，適用可能性のある，Negascout や反復深化深さ優先探索などの複雑なゲーム木探索アルゴリズムを適用し，計算性能や，ゲーム AI の強さを比較し検討することで，ゲーム AI の実装技術において，今後の発展に期待できる．

参考文献

1) IBM (2007)「新たな科学技術の発見のためのエンジンとなった IBM Blue Gene」, http://www-06.ibm.com/jp/press/20070516001.html 2014 年 9 月 18 日アクセス.
2) 小谷 善行編 (2010)「ゲーム計算メカニズム」コロナ社.
3) Sugimoto, N. ; Miyajima, T. ; Kuhara, T. ; Katuta, Y.more authors, (2013) "Artificial intelligence of Blokus Duo on FPGA using Cyber Work Bench ", Field-Programmable Technology (FPT), 2013 International Conference on, pp.498-501, IEEE.
4) 美添 一樹 (2012)「モンテカルロ木探索およびコンピュータ囲碁の進歩について」, http://www-

erato.ist.hokudai.ac.jp/docs/summer2012/compgo_plan_pub.pdf, 2014 年 9 月 19 日アクセス.
5) Liu, C. (2013) ”Implementation of a highly scalable blokus duo solver on FPGA”, Field-Programmable Technology (FPT), 2013 International Conference on, pp.482-485, IEEE.
6) Jiu Cheng Cai; Ruolong Lian ; Mengyao Wang ; Canis, A. ; Jongsok Choi ; Fort, B. ; Hart, E. ; Miao, E. ; Yanyan Zhang ; Calagar, N. ; Brown, S. ; Anderson, J (2013) ”From C to Blokus Duo with LegUp high-level synthesis”, Field-Programmable Technology (FPT), 2013 International Conference on, pp.486-489, IEEE.
7) Kojima (2013) ”An implementation of Blokus Duo player on FPGA”, Field-Programmable Technology (FPT), 2013 International Conference on , pp.506-509, IEEE.
8) ICFPT (2013) ”Rules of Blokus Duo”, http://lut.eee.u-ryukyu.ac.jp/dc13/rules.html 2014-09-18 accessed.
9) オーバートーン株式会社 (2012)「NSL リファレンスマニュアル」, http://www.overtone.co.jp/binaries/documents/reference /NSL_Language_Reference_ver1.4.pdf 2014 年 9 月 18 日アクセス.
10) 青山 卓也, 玉城 良, 廣瀬 翔太, 清水 尚彦 (2013)「NSL による Blokus Duo のハードウェアシステム・GUI 実装」, 第 39 回パルテノン研究会, Vol.39, pp.43-46, パルテノン研究会.

付録

Blokus Duo のルール

Blokus Duo では 14×14 マスの盤面上で，お互い交互に 21 種類の形の異なるピースをひとつずつ置いていく．最終的にお互いがピースを盤面上に置けなくなった時点の，残ったピースの総面積が少ない方が勝利となる．また，ルール上の制約として，新しくピースを置く時は，Fig.13 左のように角を一つ以上自分のピースと接しなければならず，Fig.13 右のように自分のピースと辺で接してはならない．相手のピースに対しては，マスが被らなければ自由に接してよい．

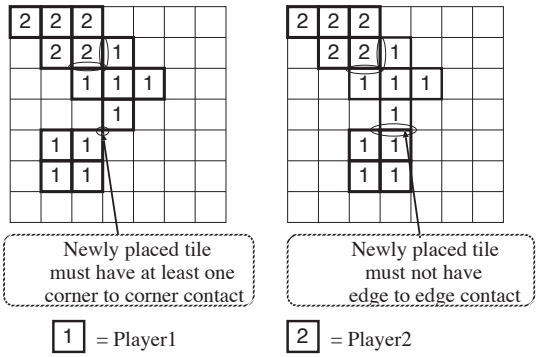


Fig.13　Corner to Corner Contact