# FPGA Implementation of a Strong Reversi Player

Javier Olivito [#1], Carlos González [*2], Javier Resano [#3]

[#] *DIIS-I3A, University of Zaragoza*
*Zaragoza, Spain*
[1] creoeneoli@gmail.com
[3] jresano@unizar.es

[*] *Architecture Department, Complutense University of Madrid*
*Madrid, Spain*
[2] carlosgonzalez@fdi.ucm.es

***Abstract*—In this article, we present a design of a Reversi player submitted to the FPT'10 Design Competition and implemented on a XC2VP30 Virtex-II Pro FPGA.**

**Our player applies several techniques to explore the solution space attempting to look as many moves forward as possible for the given time, and uses several metrics to evaluate the quality of a given board. The most important metric is the mobility, basically our player attempts to maximise its available moves whereas minimising the opponent moves.**

**With these techniques our player easily defeats the competition software opponent.**

## I. Introduction

Reversi (aka Othello) [1] is a strategy board game played between two players. It is played using a square 8 x 8 board and 64 discs. The discs are coloured black on one side and white on the opposite side. Each player shall be assigned to play a colour. The goal of the game is to have the majority of colour discs on the board at the end of the game.

The game will start with black making a move. Play then alternates between white and black until one of the following occurs: There are no moves that the player can make to outflank the opponent's disc(s) (the player is then said to have no valid moves) or both players have no valid moves.

When a player has no valid moves, he forfeits his turn and the opponent continues to move. A player is not allowed to voluntarily forfeit his turn. The game ends when both players have no valid moves or when the entire board has been played. Therefore, it is possible for a game to end before all 64 squares are filled.

Designing a Reversi player is a challenging problem for a digital logic designer due to the huge computational complexity [2]. In fact designing a perfect player is unfeasible and the design must rely on evaluation heuristics and search techniques that cannot guarantee that the best movement is selected. There are two key issues when designing a Reversi player. On the one hand, the player evaluates the possible movements using an evaluation function. This is a complex problem since in the Reversi game there is not any straightforward way of knowing whether you are winning or losing. On the other hand a good player needs to look forward in the game as far as possible in order to evaluate which will be the consequences of each movement.

**978-1-4244-8983-1/10/$26.00 ©2010 IEEE**

In this paper, we present the design and implementation of a Reversi player. Our player explores the solution space (looking forward) using a minimax approach with alpha-beta pruning. It also applies an iterative deepening approach, analysing first four levels forward, then five, then six, and continuing until we reach a configurable timeout. The minimax exploration search is optimised applying a simplified dynamic node ordering that explores the best movement of the previous level first in order to improve pruning. Regarding the evaluation function, we have defined several metrics in order to evaluate a given board. The most important is mobility (i.e. the number of legal movements). During the game we attempt to maximize our mobility, and to minimize the mobility of the opponent. We also attempt to get the square corners, and to maximize the stable discs that are those that cannot be flipped. Finally, if we reach the end of the game we only take into account the number of discs of each colour.

Our design has been implemented on a XC2VP30 Virtex-II Pro FPGA included in the XUP Development System [3]. This FPGA includes two embedded PowerPCs (not used in this design), 13969 Slices and 2448 Kb Block RAMs. We have developed our design using VHDL language for the specification and the Xilinx ISE environment [4].

We do not consider that this design is completely finished. On the contrary, we have identified several possible optimizations. Nevertheless, we believe that our player has achieved a very good level, and only expert players can defeat it.

The rest of the paper is organized as follows: Section II explains the implemented techniques and the outline of our solution strategy. Section III focuses on the proposed architecture. Section IV provides a report on the results of our benchmark evaluation. Section V explains our conclusions and indicates some lines for future work.

## II. Implemented Techniques

This section outlines the techniques applied by our Reversi player.

### A. Minimax Search Algorithm

The minimax algorithm selects the 'best' next move for a computer player in a two player game. The algorithm makes a tree of all possible moves for both players. This algorithm is

called minimax simply because the computer makes moves that bring it maximum gain, while assuming the opponent makes moves that bring the computer minimum gain. Because the players alternate moves, the algorithm alternates between minimizing and maximizing levels of the recursive search tree.

There are two players involved, MAX and MIN. A search tree is generated, depth-first, starting with the current game position up to the end game position when the final game position is evaluated. Afterwards, the inner node values of the tree are filled bottom-up with the evaluated values. The nodes that belong to the MAX player receive the maximum value of its children. The nodes for the MIN player will select the minimum value of its children.

### *B. Alpha-beta Pruning*

The minimax search tree grows too fast, and soon becomes unfeasible to analyse it. The alpha-beta Pruning scheme allows minimax to do the same analysis more efficiently, without losing any information. The idea is to identify and skip (prune) all the nodes that cannot modify the best value.

### *C. Iterative Deepening Approach*

Our player must decide the movement meeting a given deadline (1s for the Design Competition). Then the question is, how many levels can be analyzed during this time? The answer is that it depends a lot on the current table. One option was to use a worst-case approach and analyse few levels, but this will decrease the quality of our player. Hence we decided to apply an iterative deepening approach. Basically, we start analysing 4 levels, then 5, then 6, until a timeout signal reports that we are running out of time. When this happens we select the solution found in the last analysed level.

### *D. Dynamic Node Ordering*

The efficiency of the alpha-beta procedure depends on the order in which successors of a node are examined. If the max levels of the minimax search start with the best movements, and the min levels start with the worst movements, the search will be much faster.

Since we are applying an iterative deepening approach, when we start the analysis of level $n$ we already have some information (obtained during the analysis of the level $n$-1) of the quality of the movements. And this information can be used to optimize the search. We did not have time to carry out a complex optimization. Hence, we have only included a very simple optimization: our minimax search starts analysing the best movement found in the previous level.

### *E. Evaluation Function*

There are several concepts taken into account in the evaluation of a leaf node, depending on whether the node is terminal (end of the match) or not.

In the early and mid-game, the evaluation of a leaf node is based on:

- Mobility
- Corners and X-squares
- Stable discs

Mobility refers to the number of legal moves of each player. Mobility is a crucial concept; the objective is forcing our opponent to make undesirable moves. This goal can be achieved limiting as much as possible the opponent's mobility.

Corners and valuable squares because discs placed on them are stable, and because generally make easy to get more stable discs (stable discs are those ones that cannot be flipped). On the contrary, X-squares (corner adjacent squares placed on the major diagonals) are not valuable squares. The reason for that is that X-squares give our opponent free access to corners in many ways, so placing a disc in an X-square will probably cause to loss its corresponding corner. Note that X-squares are undesirable squares only while its corresponding corner is empty.

As a result of all of this, our evaluation function for non-terminal nodes is:

$$\begin{aligned} Non-terminalNode_{ev} = Corners_{FPGA} * 16 - \\ Corners_{opponent} * 16 + XSquares_{FPGA} * 4 - \\ Xsquares_{opponent} * 4 + mobility_{FPGA} - \\ mobility_{opponent} + stables_{FPGA} - stables_{opponent} \end{aligned}$$

where *Corners* is the number of discs on the corners, *Xsquares* is the number of discs on the X-square places, *mobility* is the number of posible movements and *stables* is the number of discs that cannot be flipped. And for terminal nodes:

$$TerminalNode_{ev} = Discs_{FPGA} - Discs_{opponent}$$

where *Discs* is the number of discs on a board for a given player. In the latter case our player simply counts the number of discs of each colour, since at the end obtaining as many discs as possible of your colour if the final objective of the game.

## III. Proposed Architecture

The proposed architecture consists of the following modules:

### *A. Move Checker*

This combinatorial module has a 128-bit entry, representing the current board, and a 1-bit entry indicating the current turn (black or white). It applies a logic function for each square. As output it returns the legal moves as a 64-bit vector (see Fig. 1).

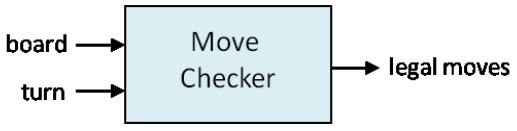


Fig. 1. Move Checker interface

### B. Disc Flipper

When a move is made, one or more discs have to be flipped. Disc Flipper is a purely combinatorial module that takes the current board, the current turn, and the desired move, and returns the new flipped board (see Fig. 2).

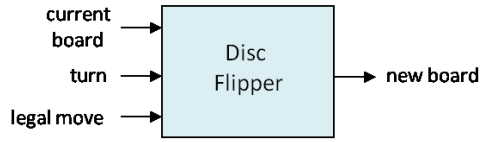


Fig. 2. Disc Flipper interface

Disc flipper is a complex module, and has been implemented as a bidirectional and two-dimension iterative network, composed of 64 basic cells, one per each board square. First the network looks for a pattern that involves flipping and if found it is propagated to the neighbours. Each cell receives a pattern in each direction, updates it taking into account its entry and finally propagates its result to its neighbour in every direction (W, NW, N, NE, E, SE, S, SW). Afterwards, when the cell that corresponds to the new placed disc receives these patterns, it will send a flip signal pointing out how many discs have to be flipped in each direction. The flip value depends on the previous propagation pattern received. Those cells which receive a flip signal, will flip its disc, and then decrement the flip signal and propagate it to its following neighbour in the proper direction.

### C. Move Selector

Here resides the IA of the processor. It implements the minimax algorithm with alpha-beta pruning and iterative deepening and its node evaluation (see subsection III-D). Moreover, a simplified dynamic node ordering has been implemented. Dynamic ordering takes advantage of iterative deepening knowledge generation: the best move determined in the previous tree generated determines the first sub-tree explored in the next deeper tree. It potentially increases the alpha-beta pruning performance, since it is likely that the best move determined in the previous tree will be also the best one in the next deeper tree generated.

### D. Evaluator

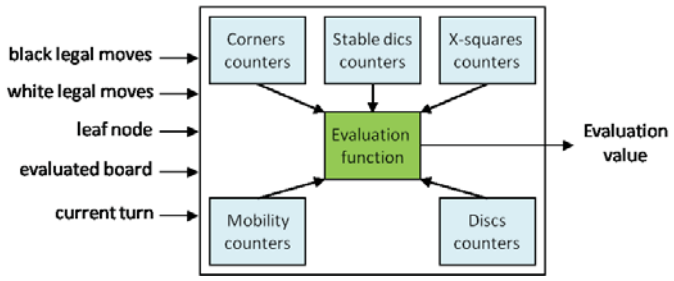


Fig. 3. Evaluator architecture

Fig. 3 describes the Evaluator. This module implements the evaluation functions described in the previous section. It uses the output of the Move Checker module to identify the $mobility_{FPGA}$ and $mobility_{opponent}$ values. In fact, there are two Move Checker modules in order to identify in the same cycle the mobility of the white and the black player. The value of $Corners_{FPGA}$, $Corners_{opponent}$, $XSquares_{FPGA}$ and $Xsquares_{opponent}$ are obtained from the current table using very simple combinatorial logic. Stable discs are a much more complex issue.

Any disc is stable if at least one of its neighbours in every direction (horizontal, vertical, diagonal) is stable. Applying directly this definition to design a combinatorial function will lead to a mutual recursion problem, so we did not implement a combinatorial solution this time.

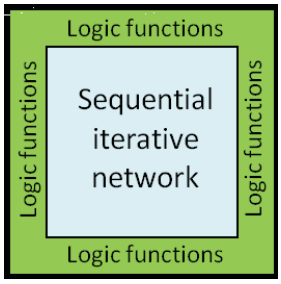


Fig. 4. Stable discs evaluation method according to the board areas

Our stable discs evaluator has been implemented using combinatorial logic functions for the border squares and using a sequential iterative network for the rest of the squares (see Fig. 4). This network includes latches to break the mutual recursion problem, and also 1-bit comparators to identify when the computations have finished (i.e. when there are no changes in the latches of the network).

In the end-game, the evaluation of a leaf (and terminal) node is based exclusively on the number of discs.

## IV. Results

Our design has been tested against the software provided for this competition. The competition rules specify that the maximum computation time for each movement must be one second. But we have also tested our design using less time: 0.1s and 0.000032s. The results are shown in Tables I, II, III.

TABLE I
Results Against Competition SW with Computation Time = 1s

| Skill | User moves black | | | Computer moves black | | |
|---|---|---|---|---|---|---|
| | Score | Moves | | Score | Moves | |
| | | User | SW | | User | SW |
| 1 | Won 63-1 | 32+0 | 31+3 | Won 63-0 | 31+0 | 31+3 |
| 2 | Won 64-0 | 33+0 | 32+5 | Won 60-4 | 31+0 | 31+2 |
| 3 | Won 64-0 | 31+0 | 30+1 | Won 64-0 | 33+0 | 33+6 |
| 4 | Won 64-0 | 31+0 | 30+1 | Won 63-0 | 31+0 | 31+3 |
| 5 | Won 64-0 | 33+0 | 32+5 | Won 64-0 | 30+0 | 30+0 |
| 6 | Won 64-0 | 31+0 | 30+1 | Won 63-0 | 31+0 | 31+3 |
| 7 | Won 64-0 | 33+0 | 32+5 | Won 64-0 | 32+0 | 32+4 |

Table I shows our results against the competition software including the final score, the number of moves, expressed as *number of moves placing a disc + number of times forced to pass*, and the maximum computation time used by our design in a move. These results show how our design defeats very easily the competition software.

TABLE II
RESULTS AGAINST COMPETITION SW WITH COMPUTATION TIME = 0.1S

| Skill | User moves black | Computer moves black |
|---|---|---|
| | Score | Score |
| 1 | Won 57-7 | Won 64-0 |
| 2 | Won 64-0 | Won 60-0 |
| 3 | Won 64-0 | Won 64-0 |
| 4 | Won 64-0 | Won 59-4 |
| 5 | Won 64-0 | Won 61-1 |
| 6 | Won 64-0 | Won 59-4 |
| 7 | Won 60-3 | Won 59-5 |

Table II shows our results against the same opponent, but limiting our computation time to 0.1 seconds. Our design still obtains very good results.

TABLE III
RESULTS AGAINST COMPETITION SW WITH COMPUTATION TIME = 0.032MS

| Skill | User moves black | Computer moves black |
|---|---|---|
| | Score | Score |
| 1 | Won 42-22 | Won 51-3 |
| 2 | Won 52-12 | Won 44-20 |
| 3 | Won 42-22 | Won 54-10 |
| 4 | Won 60-0 | Won 38-26 |
| 5 | Won 63-0 | Won 47-17 |
| 6 | Won 63-1 | Won 38-26 |
| 7 | Won 63-0 | Won * |

*Computer made an invalid move

Table III shows the results limiting computation time to only 0.032 milliseconds. Even with such a little computation time, our design is able to win in all cases.

Moreover, we have found interesting to test our design against WZebra. WZebra [5] is one of the most powerful Reversi game softwares. WZebra engine has sophisticated search algorithms, very accurate and powerful heuristics, and a huge opening table, so it is a really big challenge to defeat it.

We have tested our design against WZebra for search depth levels from one up to six. For search depth level one, our design has always won. For higher search depth levels, our design won about a 40% of the matches. WZebra exhibits non-deterministic behaviour, and sometimes we have found funny results. For instance, we have won almost all of the games when playing against level 6, but we have lost almost all the games against level 4, and some of the games for levels 2, 3 and 4 that should be all weaker opponents. As a conclusion WZebra is clearly more powerful, but we are very proud of being able to compete against such a powerful opponent for several levels.

TABLE IV
FPGA RESOURCES UTILIZATION

| Slices | | BRAMs | |
|---|---|---|---|
| Number | Percentage | Number | Percentage |
| 5325 | 38 | 4 | 2 |

Table IV presents the resource utilization of our design: 38% of the slices and 2% of the BRAM memory resources. Hence this design can be loaded even on very small FPGAs. The design is currently using a 32 MHz clock. The reason is that we have included huge combinatorial blocks that cannot work faster. However, since these blocks are used almost every cycle, we believe that this has been a good design decision.

## V. CONCLUSIONS AND FUTURE WORK

We have almost obtained the optimal results when playing against the competition software opponent. Taking into account that have designed our Reversi player in a few months, and that none of the design team members had any previous experience with the Reversi game, we consider that these are very good results and that these results demonstrate that FPGAs can be very efficient for this kind of 'general purpose' computations. We believe that the results for a timeout of just 32 $\mu$s are impressive and that it is unlikely that a software solution could achieve comparable results with such a tight timeout.

Our design efficiently search the design space, and our evaluation function has demonstrated that it can steer that search toward good movements. Of course Reversi is an extremely complex game that has been studied in depth for a long time. Hence it is possible to find stronger players as WZebra.

Of course it is also possible to improve our design. The hardware cost is quite affordable, and we can simply replicate the design and apply a parallel approach during the search. In addition the cost function was tuned playing against the competition opponent. If we tune it playing against a stronger opponent we believe that we can improve our player without increasing the hardware cost. Moreover, we did not have time to fully apply the dynamic node ordering technique, that can be very powerful to increase the look-ahead search level. Finally, introducing opening tables and a sophisticated search algorithm, such as NegaScout or MultiProb-Cut, will lead to better results.

## REFERENCES


[1] Brian Rose, "Othello: A Minute to Learn... A Lifetime to Master". 2005. http://othellogateway.com/
[2] S. Iwata and T. Kasai, "The Othello game on an n*n board is PSPACE-complete", Theoretical Computer Science, vol. 123, pp. 329-340, January 1994.
[3] Xilinx Web Site. http://www.xilinx.com/univ/xupv2p.html
[4] Xilinx Inc., Available online: http://www.xilinx.com
[5] http://radagast.se/othello/