Hardware/Software Co-Design of a High-Speed Othello Solver

Pranav Gangwar, Satvik Maurya, Shubham Garg, Sakshi Goyal, Aditya S Kumar, Preyesh Dalmia, Neeta Pandey
Department of Electronics and Communication Engineering
Delhi Technological University
Delhi, India
{pranavgangwar, satvikmaurya22}@gmail.com

Abstract— This paper presents an improved Othello game solver using the Hardware/Software Co-Design approach. Enhancements in Data Communication, Searching Algorithm, and Evaluation function have been made to beat the prevalent implementations. A novel 64-bit Communication model is proposed which transfers the current board position from the Processing System to the Programmable Logic through a 64-bit change board, instead of the complete 128-bit board. The proposal uses NegaScout searching algorithm along with Transposition Table and Node Ordering; and features like Opening Book, Killer Heuristics, and Bitboards. An Adaptive Evaluation function assigns variable weights to Mobility, Stability, Corner Occupancy, and Disc Differential depending upon the game stage. At a search depth of eight, the proposed communication model improves the communication overhead by 19.5%. Overall, the proposed design demonstrates 75.06% improved performance as compared to the prevalent implementation, and 6.53 times faster than its software implementation on the same platform.

Keywords— Othello, Reversi, hardware/software co-design, high-speed, C, NegaScout, node ordering, game theory, FPGA, ZYNQ

I. Introduction

Board game solvers have been an intriguing topic for researchers [1], and have become substantially sophisticated due to significant advancements in computational power and brute force algorithms. The complexity of games like Chess and Othello lies in the huge number of possibilities for a single move, making it impossible for any machine to evaluate every possible outcome. Therefore, computer programs depend on heuristics, and their effectiveness depends upon the search depth for a particular move in a given time.

Most of the implementations in the literature are targeted for desktop and server grade computers [2]. The computational, and battery-life limitations of portable devices like smartphones and tablets, reduce their effectiveness. The FPGA implementation of some computationally intensive board games like chess [3], Othello [4], Blokus Duo [5], Trax [6], and Connect6 [7] has received considerable attention due to its speed and power efficiency. Although the complete FPGA solution is always desirable, the associated high development cycle and mapping time does not encourage the hardware implementation of the most advanced algorithms. Recently, the Hardware/Software Co-design approach [8] was explored for board games which combined the flexibility and faster development time of software designs, with the high performance and low power consumption of FPGAs. It selectively accelerates the computational hotspots of the software, thereby avoiding the significantly larger development cycle associated with a complete FPGA implementation.

This paper proposes a high-speed Othello game solver based on Hardware/Software Co-design approach, by optimizing: Data Transfer, Searching Algorithm and Heuristics, and Evaluation Function. The Data transactions from software to hardware have been reduced by 50% with the use of a novel 64-bit Communication model. The software design is enhanced by the use of NegaScout algorithm [9], which is aided by heuristics like Transposition Table and Node Ordering [10], Opening Book [11], Killer Heuristics [12], and Bitboards [10]. An Adaptive Evaluation function is implemented on the FPGA, which dynamically adjusts the weights of Mobility, Stability, Corner Occupancy, and Disc Differential depending on the present game stage [13].

The paper is organized as follows: Section II describes the proposed design and its implementation, Section III presents a comprehensive analysis of the proposed design with the prevalent design, and Section IV concludes the paper.

II. Proposed Design

Profiling the software implementation of the Othello solver using the Xilinx TCF Profiler revealed that Evaluation Function takes about 62.88% of total game time, whereas 13.73% of game time is spent on the Move Generation, 4.70% on the Disc Flipping, and the rest 18.69% on Searching Algorithm and Transposition Table. Thus, the Evaluation Function and Move Generation were implemented on hardware. The software, hardware, and the HW/SW Communication components of the proposed design are described below:

A. Software

The NegaScout searching algorithm is employed to evaluate the game tree, due to its superior searching efficiency than the Alpha-Beta Pruning especially at higher search depths with proper node ordering [9]. It assumes the first node as the best searched node, and performs a null window search on all the remaining nodes. A wide window re-search is performed for a node when the null window search fails HIGH, and the node does not belong to the two lowest game tree levels. For randomly ordered nodes, NegaScout takes more time than Alpha-Beta pruning as it re-searches several nodes due to repeated null window search failures. Therefore, a Transposition Table is used to implement node ordering.

The Transposition Table is a data structure which improves the search efficiency by preventing redundant re-evaluations. Each traversed node can be categorized into three types: EXACT, LOWER_BOUND, and UPPER_BOUND based on the evaluated values of the sub-tree [10]. These are stored in the Transposition Table along with the score of the node, and the move position of the child nodes in decreasing order of the score value. The key of the Transposition Table is created by merging two Bitboards, and calculating their Zobrist Hash [14]. The ordering of the child nodes is determined from the results of a shallow search. The sub tree search is performed in the node order specified by the Transposition Table to maximize the number of cut-offs. In this design for fixed game tree of depth 8, a shallow search to a depth 4 is done to determine the searching order of the nodes.

In addition, following techniques are also used to speed-up the evaluation of the game tree:

- Bitboards: The Othello game board is processed in the software using Bitboards, which is a 64-bit data type with each bit corresponding to a position on the board. A SET bit signifies that the square is occupied. Two Bitboards are used for the two types of discs in the Othello board i.e., black and white. Board manipulations like playing a move, finding an empty position, counting the number of discs, etc. are performed using shift, AND, and OR operations on Bitboards.
- Opening Book: Othello has several fixed move sequences collectively known as Opening Book, which when played in the beginning of the game can benefit the player at later game stages. It greatly speeds up the execution in the early game stages, since the moves are predetermined.
- Killer Heuristics: Killer Heuristics are the moves which on being played increase the probability of the player winning the game. Capturing the corners of the Othello board are a few such moves, which if available can directly be played without searching the game tree.

B. Hardware

The Othello game tree has approximately 10^{58} nodes, with each node performing numerous computations. This offers the possibility of introducing parallelism by performing the computations on a dedicated hardware. The complete evaluation function of Othello has been implemented on hardware, and the description of various components is given below:

- Mobility: It refers to the legal move positions available to a player. Maximizing mobility is an important strategic consideration based on the premise that a player with low mobility is more likely to be forced to make a bad move, such as giving up a corner. Mobility is implemented as a combinational block for each of the 60 squares (barring the initial 4 pieces) on the 8X8 board. For determining whether a position is a legal move for a white or black player, at most 7 squares are tested in each of the 8 directions (vertical, horizontal and diagonals), as shown in Fig.1. The Boolean expression for finding legal moves for a white player at position (1,1) in the vertically downward direction is given by (1):

$$\mathrm{Mobility}(1,1) = \overline{\mathrm{B}(1,1)} \cdot \overline{\mathrm{W}(1,1)} \cdot \left\{ \begin{array}{c} \mathrm{B}(2,1) \cdot \mathrm{W}(3,1) + \\ \mathrm{B}(2,1) \cdot \mathrm{B}(3,1) \cdot \mathrm{W}(4,1) + \\ \mathrm{B}(2,1) \cdot \mathrm{B}(3,1) \cdot \mathrm{B}(4,1) \cdot \mathrm{W}(5,1) + \\ \mathrm{B}(2,1) \cdot \mathrm{B}(3,1) \cdot \mathrm{B}(4,1) \cdot \mathrm{B}(5,1) \cdot \mathrm{W}(6,1) + \\ \mathrm{B}(2,1) \cdot \mathrm{B}(3,1) \cdot \mathrm{B}(4,1) \cdot \mathrm{B}(5,1) \cdot \mathrm{B}(6,1) \cdot \mathrm{W}(7,1) + \\ \mathrm{B}(2,1) \cdot \mathrm{B}(3,1) \cdot \mathrm{B}(4,1) \cdot \mathrm{B}(5,1) \cdot \mathrm{B}(6,1) \cdot \mathrm{B}(7,1) \cdot \mathrm{W}(8,1) \end{array} \right\} \quad (1)$$

where W(x,y) and B(x,y) represents the presence of white and black disc respectively at position (x,y) on the board.

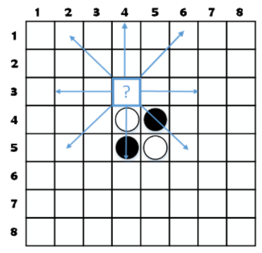

Fig. 1 Determination of Mobility

According to the Boolean expression, a white piece can be placed at position (1,1) if there is no piece already present, and it outflanks at least one black piece.

- Stability: Stable discs are impossible to flip after placement. They can change the dynamics of the game by flipping the unstable discs, hence contribute significantly to the final score. Stability of a disc is calculated using the following Boolean expression (2):

$$S_{i,j} = (S_{i,j-1} \,|S_{i,j+1}) \,\&\, (S_{i-1,j} \,|S_{i+1,j}) \,\&\, (S_{i-1,j+1} \,|S_{i+1,j-1}) \,\&\, (S_{i-1,j-1} \,|S_{i+1,j+1}) \quad (2)$$

where (i, j) denotes row and column number of the board and $S_{i,j} = 1$ denotes that the disc placed at (i,j) is stable.

- Corner Occupancy: Corners are strategically the most powerful positions in the game, because they cannot be flipped once captured. They control the horizontal, vertical as well as the diagonal positions on the board. Capturing a corner is always advantageous in the game; therefore, it is given a high weightage in the evaluation function.
- Disc Differential: The game is won by capturing higher number of discs than the opponent therefore; disc differential is an important heuristic for the evaluation function in the later stages of the game.

The features described above affect the outcome of the game at different stages. Hence, the evaluation function has been made adaptive to accommodate the dynamic nature of the game, by changing the feature weights depending on the game stage. Moreover, the feature weights have been used in the power of 2 to minimize the data path delay in the FPGA, and achieve a high operating frequency. The score is calculated using relation (3):

$$\begin{aligned} \mathrm{Score} = {} & W_{mobility} * (Mobility_{player} - Mobility_{opponent}) + \\ & W_{Stability} * (Stability_{player} - Stability_{opponent}) + \\ & W_{Corners} * (Corners_{player} - Corners_{opponent}) + \\ & W_{Disc\ difference} * (Discs_{player} - Discs_{opponent}) \end{aligned} \quad (3)$$

The weightage given to the various features is decided using Table I. The early stage corresponds to less than 20 discs on the board. The lower mid-half stage is for 21-36 discs, while the upper mid-half corresponds to 37-53 discs on the board. The endgame strategy takes effect after 54 discs on the board.

TABLE I. EVALUATION STRATEGIES FOR DIFFERENT GAME STAGES

Stages	**Strategies**			
	Mobility	**Stability**	**Corners**	**Disc Difference**
Early	High	Low	Moderate	Low
Lower Mid-Half	High	Moderate	High	Low
Upper Mid-Half	Moderate	High	High	Moderate
Endgame	0	0	0	High

C. *HW/SW Communication*

The HW/SW Communication architecture used in [8] (hereafter referred to as the Standard Communication Model) requires four AXI-Lite transactions for sending the complete 128-bit game board from the Processing System (PS) to the Programmable Logic (PL).

The data transfer from PS to PL is accomplished in 64 bits which requires two AXI-Lite transactions, as compared to the Standard Communication Model. This is achieved by the following procedure and a block diagram of the architecture used is shown in Fig. 2:

- **Changeboard Generation**: A 64-bit Changeboard is generated in the PS by determining the changes in the current 128-bit board with respect to the parent board, with all the affected positions represented by 1. The Changeboard is transmitted to a specific AXI Interface register depending on the player's identity, and the difference between the game tree depths of the last sent board and the current board.
- **Address Decoder**: All the ancestor 128-bit boards of the current board are stored in the PL's memory. The parent board of the transmitted Changeboard is fetched from this memory, whose relative address from the current address depends on the game tree's depth difference between the last sent board and the current board. The address of the AXI register storing the Changeboard is determined by the AXI interface's control signals, which decides the relative address of the Changeboard.
- **New Board Generation**: The New Board Generator requires the Changeboard, the reference board, and the player's identity to reconstruct the complete current board, which is then stored in the memory.

For black player's current move,

$$(Board_b)_{i+1} = (Board_b)_i \mid Changeboard \tag{4}$$

$$(Board_w)_{i+1} = (Board_w)_i \& (\sim Changeboard) \tag{5}$$

For white player's current move,

$$(Board_w)_{i+1} = (Board_w)_i \mid Changeboard \tag{6}$$

$$(Board_b)_{i+1} = (Board_b)_i \& (\sim Changeboard) \tag{7}$$

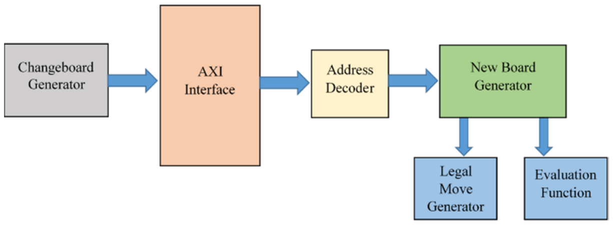

Fig. 2. Proposed Communication Model

III. Results

The HW/SW Co-Design architecture presented in this paper is implemented on Digilent's Zybo Development board with Xilinx's Zynq Z-7010 SoC[15], using Xilinx Vivado Design Tool [16]. The AXI4 Lite [17] mode is used for communication between the PS and the PL, with code compilation performed using the GNU GCC Compiler by activating the Maximum Optimization Flag.

The proposed design implemented as Design A is compared with two designs: Design B and Design C. Design B employs the same software and hardware as Design A, but with the Standard Communication Model. It is designed to measure the exact speed-gain achieved by the 64-bit communication model, considering only the software and the hardware overheads. Design C is based on [8] which is designed to measure the overall speed-up achieved by the proposed design. It uses Alpha-Beta Pruning with node ordering till depth 3 for the game tree search, and transmits the complete board to the hardware by first converting the character board to the 128-bit board. The hardware returns the encoded value of stability and mobility in 32 bits requiring one AXI transaction, which is used to calculate the score in software.

The performance of the three designs are compared on the basis of Total Execution Time (calculated by making the solver play against itself to account for the worst scenario, and measuring the total time taken to finish the game), and the Hardware/Software Communication Time, as shown in Table II. The Hardware computation time is negligible for all the designs, because the time taken to initialize the AXI protocol before transferring the data to the software is greater than the hardware computation time. For fair comparison, all the designs were run using the maximum clock frequencies (estimated by Vivado after P&R) which have been listed in Table III.

The proposed HW/SW Co-Design architecture attains a performance gain of more than 4X as compared to [8], owing to the optimizations made in the Hardware/Software communication as well as the searching algorithm. A detailed analysis demonstrates that the communication overhead is reduced from 64.92% of the Total Execution Time in Design C, to 31.06% in Design B, and finally to 26.89% in Design A. Hence, the novel 64-bit Communication model improves the communication overhead of the game solver by 19.5%. The vast improvement in the overhead of designs A and B over design C can also be attributed to better searching algorithms used, which reduces the number of AXI transactions.

TABLE II. COMPARISON OF EXECUTION TIME AND COMMUNICATION OVERHEAD OF ALL THREE DESIGNS

Design	Execution time (in seconds, max. search depth = 8 moves)		
	ARM Cortex A9	HW/SW Co-Design	
		Total Time	Communication overhead
Design A	98.2	15.02	4.04
Design B	98.2	16.16	5.02
Design C	306.4	60.23	39.1

TABLE III. CLOCK FREQUENCIES FOR THE THREE DESIGNS

Design	Clock Frequency (MHz)
Design A	64.06
Design B	72.35
Design C	78.65

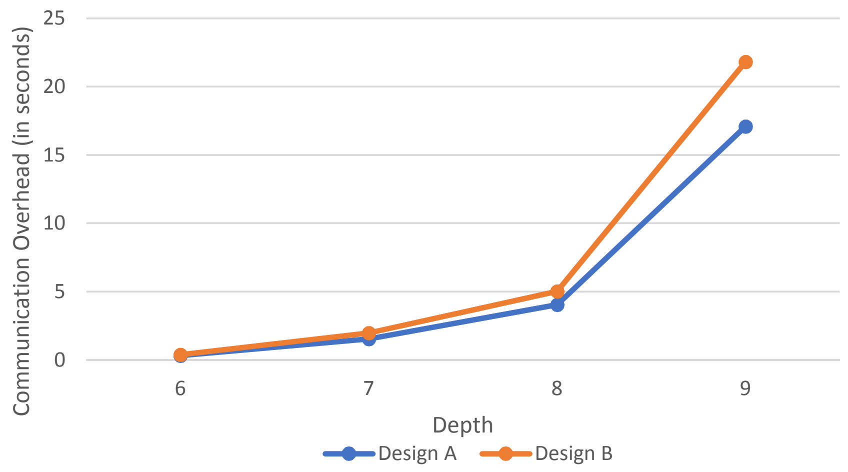

Fig. 3. Comparison of the Communication Overhead of Designs A and B

Figure 3 shows the increase in the communication overhead with search depths for the proposed Design A, and Design B which uses the Standard Communication Model. It can be concluded that the communication overhead will increase at a much lower rate for the proposed 64-bit Communication model.

IV. Conclusions

A high-speed Othello game solver using the HW/SW Co-Design architecture has been presented in this paper. The game solver is optimized by using a novel HW/SW communication model, NegaScout search algorithm, Node Ordering, Transposition Table, Opening Book and Killer Heuristics, and Bitboard. The proposed design shows a performance improvement of more than 4X when compared to the prevalent implementations. Furthermore, the reduction of the communication overhead has a significant impact on the performance of the game solver, thus demonstrating its effectiveness in applications employing HW/SW Co-Design architectures. The overall speed and strength of the evaluation function makes this solver a good match for various game solvers used in mobile applications.

References

[1] C. Shannon, Programming a Computer for Playing Chess, New York: Springer, New York, NY, 1988.

[2] M. Buro, "Logistello: A strong learning othello program," in *Annual Conference Gesellschaft für Klassifikation*, 1995.

[3] M. Boule and Z. Zilic, "An FPGA Based Move Generator for the Game of Chess," in *IEEE Custom Integrated Circuits Conference*, Orlando, FL, USA, 2002.

[4] C. Wong, K. L. Lo and P. Leong, "An FPGA-based Othello endgame solver," in *IEEE International Conference on Field-Programmable Technology* , Brisbane, NSW, Australia, 2004.

[5] A. Kojima, "An Implementation of Blokus Duo player on FPGA," in *International Conference on Field-Programmable Technology (FPT)*, Kyoto, Japan, 2013.

[6] A. Kojima, "An Implementation of Trax player using programmable SoC," in *International Conference on Field Programmable Technology (FPT)*, Queenstown, New Zealand, 2015.

[7] T. Watanabe, R. Moriwaki, Y. Yamaji, Y. Kamikubo, Y. Torigai, Y. Nihira, T. Yoza, Y. Ueno, Y. Aoyama and M. Watanabe, "An FPGA Connect6 Solver with a two-stage pipelined evaluation," in *International Conference on Field-Programmable Technology*, New Delhi, India, 2011.

[8] J. Olivito, J. Resano and J. L. Briz, "Accelerating Board Games Through Hardware/Software Codesign," *IEEE Transactions on Computational Intelligence and AI in Games,* vol. 9, no. 4, 2016.

[9] A. Reinefeld, "An Improvement to the Scout Tree Search Algorithm," *ICGA Journal,* vol. 6, no. 4, pp. 4-14, 1983.

[10] J. Chen, "Applications of Artificial Intelligence and Machine Learning in Othello," Computer Systems Lab at Thomas Jefferson High School of Science & Technology, 2010.

[11] S. MacGuire, "Strategy Guide for Reversi & Reversed Reversi," [Online]. Available: http://www.samsoft.org.uk/reversi/strategy.htm.

[12] M. Buro, "The Evolution of Strong Othello Programs," in *Entertainment Computing*, vol. 112, Springer, Boston, MA, 2003, pp. 81-88.

[13] J. Olivito, C. González and J. Resano, "FPGA implementation of a strong Reversi player," in *International Conference on Field-Programmable Technology*, Beijing, China, 2010.

[14] A. L. Zobrist, "A new hashing method with application for game playing," *ICCA journal*, vol. 13, no. 2, pp. 69–73, 1970.

[15] Zynq-7000 SoC & Zynq UltraScale+ MPSoc, http://www.xilinx.com/products/silicon-devices/soc.html

[16] Vivado Design Suite - HLx Editions. Xilinx, 2017, http://www.xilinx.com/products/design-tools/vivado.html

[17] AMBA AXI4 Interface Protocol, http://www.xilinx.com/ipcenter/axi4.html