Performance and energy efficiency analysis of a Reversi player for FPGAs and General Purpose Processors

JAVIER OLIVITO, University of Zaragoza, 0034876555081, jolivito@unizar.es
RUBÉN GRAN, University of Zaragoza
JAVIER RESANO, University of Zaragoza
CARLOS GONZÁLEZ, University Complutense of Madrid
ENRIQUE TORRES, University of Zaragoza

Board-game applications are frequently found in mobile devices where the computing performance and the energy budget are constrained. Since the Artificial Intelligence techniques applied in these games are computationally intensive, the applications developed for mobile systems are frequently simplistic, far from the level of equivalent applications developed for desktop computers.

Currently board games are software applications executed on general purpose processors. However, they exhibit a medium degree of parallelism and a custom hardware accelerator implemented on an FPGA can take advantage of that.

We have selected the well-known Reversi game as a case study because it is a very popular board game with simple rules but huge computational demands. We developed and optimized software and hardware designs for this game that apply the same classical Artificial Intelligence techniques. The applications have been executed on different representative platforms and the results demonstrate that the FPGAs implementations provide better performance, lower power consumption and, therefore, impressive energy savings. These results demonstrate that FPGAs can efficiently deal with this kind of problems.

1. INTRODUCTION

The future of computation demands to increase performance while keeping power consumption as low as possible. This applies especially to mobile devices. With a strongly limited power/energy budget, they strive to approach the performance of desktop computers.

As the number of transistors that can fit in a chip grows, more components can be placed inside, leading to complex on-chip systems with several processors that can be homogeneous (multi-core platforms) or heterogeneous systems. In the latter case, the platform includes some specialized processors, where each one of them has been designed to be very efficient for an application field. This design trend results in what is called System-on-a-Chip (SoC). One recent example of this kind of platforms is Texas Instruments OMAP 5 Processor [1]. It includes two different General Purpose Processors (GPPs), each one with a different power/performance trade-off, and several specific processors to manage 2D and 3D graphics, audio, Digital Signal Processing (DSP), and encryption-decryption.

However, no matter how many transistors are available, it is clear that in general purpose computing platforms it is not possible to add custom hardware for every single application since the number of different applications can be extremely large. Moreover, the actual applications that will run on a given platform are not known at design time, because users can add new applications whenever they want. It is at this point where hardware accelerators implemented in programmable logic play an important role.

Recently, a new family of silicon devices have appeared, combining low-power GPPs with programmable logic. A first multi-chip approach was developed by Intel and Altera, integrating an Intel Atom E6xx Series GPP and an Altera Arria II GX FPGA in a single package [2]. More recently, the major FPGA manufacturers, Xilinx and Altera, have released a complete processor-based SoC and a powerful FPGA integrated in a single chip (Xilinx Zynq-7000 EPP [3], and Altera Cyclone V and Arria V SoC [4]). Both consist in a processor-centric solution with several GPP cores for command and control purposes and to execute all the task that do not demand hardware acceleration, and programmable logic resources for those tasks that demand parallel data processing, have tight real-time restrictions, or must be very energy-efficient.

These platforms allow the software developers to work in a familiar environment, and the logic designers to differentiate their solution from others by adding customized hardware blocks that improve the performance and reduce the energy consumption. In fact, Google has announced that it will include an FPGA in its Project ARA modular smartphones [5]. This is the first time that a mobile phone will include an FPGA.

Artificial intelligence in board games is a field that can take enormous advantage of these new hybrid FPGA/processors platforms. Board games applications such as Chess, Reversi and many others are frequently found in portable devices. The algorithms applied in these games usually involve a heavy computational workload which can benefit from custom hardware. This hardware is able to efficiently analyze the boards not only to improve the performance of the system but also to drastically reduce its energy consumption. The reason is that the size of the boards is especially suitable for a hardware module that processes the board and extracts all its information very fast seamlessly exploiting all its parallelism by processing all the squares in parallel, whereas an equivalent software version executes nested loops that must analyze the board following different access patterns. These loops cannot be profitably parallelized using several threads, because the size of the board is not big enough to compensate for the parallelization overheads. Moreover, they are neither suitable to take advantage of the Single-Instruction Multiple-Data (SIMD) units included in modern processors. These units execute the same arithmetic instructions to several data; however, in board games each board position may demand a different computational treatment.

Although theoretically custom hardware can be extremely efficient for board games applications, the fact is that, in the specialized literature, no previous work can be found presenting an in-depth analysis of this approach taking into account not only the performance, but also the energy and power consumption. For that reason, we have made a performance and energy efficiency analysis between hardware and software implementations of a board game application to quantify the advantages of using programmable logic.

As a case study we have selected a Reversi application, since it is both very popular and very complex. There are many Reversi applications available both for desktop computers and mobile devices. WZebra [6] is probably the most popular application for desktop computers and Reversi Free the most popular for smartphones (it has been installed in more than five millions of Android-based systems). If we compare both of them, WZebra playing in a low level easily defeats Reversi Free playing at its highest level. Reversi Free cannot process millions of boards, as WZebra does, since it would consume a lot of time and quickly drain the battery.

In order to analyze the benefits of a hardware accelerator for Reversi we have developed two algorithmically equivalent versions of a Reversi application. Both of them apply classical AI algorithms for board games, but they differ in the way that the computations are carried out. It is important to point out that the AI algorithms selected are sequential, hence they do not look as a promising target for hardware acceleration. However, these algorithms process millions of boards during a game, and a custom hardware module can process those boards very fast exploiting its data parallelism.

To make a fair comparison, we optimized both designs. In the case of the software version, which was written in C code, the design team included two proficient on code optimization that analyzed its execution in detail in order to identify hot-spots and improve the code. We also parallelized the code and tested compiler optimizations.

Both versions were implemented in the Zynq hybrid FPGA/processor platform and we compared its performance and power/energy consumption. Moreover, we also implemented them on other representative FPGAs and general-purpose processors. The hardware design has been implemented in two high-end FPGAs, Xilinx Virtex II-Pro and Virtex-5, and in one low-cost Xilinx Spartan 6. The software design has been executed on a high-performance Intel i7, and on a low-power Intel Atom.

Our main contributions are:

— Analysis of the benefits of including custom hardware for a board game application, demonstrating that the parallelism granularity of a board nicely fits with current FPGAs architectures.

— Comparison of two algorithmically equivalent hardware and software versions.

— Analysis of the results in several platforms according to three metrics: performance, power consumption, and energy efficiency.

The rest of the paper is organized as follows: Section 2 mentions related work for hardware implementations of the Reversi game, and for performance and energy-efficiency comparisons in different fields. Section 3 describes the rules of Reversi. Section 3 and 4 explains the artificial intelligence techniques implemented. Section 5 motivates hardware acceleration with an example related to Reversi. Sections 6 and 7 analyze the hardware and software designs. Section 8 shows a comprehensive performance analysis. Section 9 finishes the comparison tackling the energy efficiency. Finally, Section 10 weighs the design effort in hardware design and software development, and Section 11 exposes the conclusions of this work.

2. RELATED WORK

There are other works proposing hardware implementations for the Reversi game. C.K. Wong et al. presented in 2004 and FPGA-based implementation with a simple AI [7]. This design did not evaluate the game board dynamically but only apply a static weight for each possible move, which is a very weak approach. In any case, it was an interesting proof of concept that demonstrated that FPGAs can be used for board games. Later in 2010, Reversi was the target of a digital hardware design competition organized by the International Conference on Field Programmable Technology [8]. A previous version of our hardware design was awarded with the first prize [9]. This version provide similar performance than our current version but it was less efficient, needing 30% more hardware resources to carry out the same computations. The main reason is that it was designed very fast in order to participate in the competition. Moreover, in that competition the only objective was to play as well as possible. Hence we did not spend time optimizing the hardware resources. In this competition, the design presented by T. Mabuchi et al. was awarded with the second prize [10], and the design presented by M. Smerdis et al. [11] with third prize.

The design presented in [10] proposed to use the FPGA to implement a multiprocessor system with six specific processors specifically designed to carry out the operations needed in this game. The design described in [11] follows a Monte-Carlo method that generates several random games for each possible move and selects the move that provides better score (more random games won). These two designs follow completely different paradigms than our hardware design. Hence it is complex to compare the results. Fortunately there are objective metrics that can be used to compare them. Firstly, our design defeated these two other designs in a live session of the International Conference on Field Programmable Technology. In that session, our design won all the games, two against the design presented in [10] and two against the design from [11]. Secondly, as a preliminary step of this competition all the designs were tested against a reference software that was provided by the organizers. Each design played fourteen games (the software had seven different skill levels, and each design played one game with black stones, and another with white stones). The average results are presented in the following table. The score represents the score of the hardware design minus the score of the reference software. The maximum score that can be achieved is +64, which means that all the games finished 64-0 (64 stones for the hardware design, and 0 stones for the software opponent). As it can be seen in the table our design almost reached that value. The other hardware designs also provided very good results, but they scores were significantly worse.

Table I. Results of the ICFPT '10 designs against the reference software

Design	Average result
Our design [9]	+63
Othello solver [10]	+35
CarlOthello [11]	+50

Previously, several relevant works have analyzed the benefits of using FPGAs to implement custom accelerators. For instance, Lopez et al. show the use of FPGAs for Remote Sensing applications [12,13], Cope et al. analyze performance speedups obtained for several image processing algorithms implemented on Graphics Processing Unit (GPUs) and FPGAs [14], and J. Cong et al. present a case study for a medical application [15]. These works prove that FPGAs provide excellent performance for a wide range of applications. However, this is not enough for mobile platforms, where energy efficiency is compulsory.

3. REVERSI

Reversi [16] is a two-player board game played on an 8x8 board using black and white discs. The game begins with the movement of the black player. Then, both players alternate their movements. A legal move consists in placing a disc outflanking at least one opponent's disc in any direction (horizontal, vertical, diagonal). Every opponent's disc outflanked turns over its color. When a player has no legal moves, the turn comes back to the other player, and when both have no legal moves the game ends. The goal is to have more discs than the opponent at the end of the game.

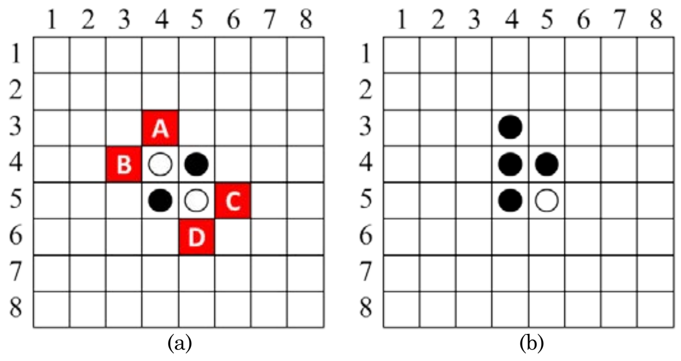


Fig. 1. (a) Initial Reversi board depicting legal moves for black player in red
(b) Board after black player played 'A'

Fig. 1a depicts the initial board and the four possible dark-player legal moves (A, B, C and D). Fig. 1b shows the board after the dark player played 'A'. Although the rules are very simple, Reversi has a huge computational complexity, as the game-tree has approximately 10^{58} nodes. Currently, it still remains as an unresolved game.

4. BOARD EVALUATION METRICS

Our design uses four metrics in order to estimate the quality of a board.

(1) Mobility. It refers to the number of legal moves of a player. This is a very important strategic concept of the game. A good player tries to maximize its mobility while limiting the opponent's one. This usually forces the opponent to make undesirable movements.

(2) Stable discs. Stable discs are those that once they are placed, they cannot be flipped anymore. They are generally desirable because they allow dominating its surrounding squares and definitely contribute to the final score. We have identified two situations where a disc is stable:

i. A disc is stable if at least one neighbor in each direction is stable.

This is formally defined as follows:

$$S_{i,j} \textit{ if } (S_{i,j-1} \vee S_{i,j+1}) \wedge (S_{i-1,j} \vee S_{i+1,j}) \wedge (S_{i-1,j+1} \vee S_{i+1,j-1}) \wedge (S_{i-1,j-1} \vee S_{i+1,j+1})$$

Where $S_{i,j}$ means that the square located in row i and column j is stable. This definition implies mutual recursion since a square needs to check if a neighbor is stable while the latter also needs to check if the former is stable. Implementing mutual recursion algorithms is always a complex issue, since it can easily lead to large performance degradations.

ii. A disc is stable if its row, column, and diagonals are fully filled.

(3) Corners and x-squares. Corners are considered valuable squares because they are stable and they are the key to get more stable discs around the corner. On the contrary, x-squares (corners adjacent squares placed on the major diagonals) are generally undesirable squares because they facilitate the opponent to reach the corners. Note that this applies only when the corresponding corner is empty.

(4) Number of discs. This metric evaluates end-game boards.

Our evaluation function is a linear combination of these metrics. The weights of the metrics were assigned based on the relative strategic importance of each metric, and later they were empirically tuned analyzing its behavior against several opponents. More accurate evaluation functions can be easily included in the system but it is not the focus of this work.

5. SEARCH TECHNIQUES

We have chosen the Minimax algorithm to explore the game-tree because it is the most common search algorithm for board games. Minimax is a depth-first search algorithm that looks for the best movement assuming that the opponent will play his best movement as well. Additionally, we included the following techniques to improve the performance of the game-tree exploration:

(a) Alpha-beta pruning. In a Minimax search, it is very common to find situations where it is possible to stop generating new successors from a node because they are irrelevant in the search outcome. The efficiency of alpha-beta pruning depends on the order in which nodes are generated. The sooner the best movements are generated the higher the pruning efficiency [17].

(b) Iterative Deepening. Typically the movement must be selected before a timeout arrives. However, the search depth that can be reached for a given time depends on the board. Hence, fixing the depth search may lead to returning no move, if the system runs out of time before finishing the analysis, or suboptimal decisions, if a small depth is chosen. Iterative deepening solves this issue making incremental depth searches until a timeout arrives. Moreover, it allows inferring knowledge about the quality of the movements, so it can be used to increase the pruning efficiency. This is discussed below.

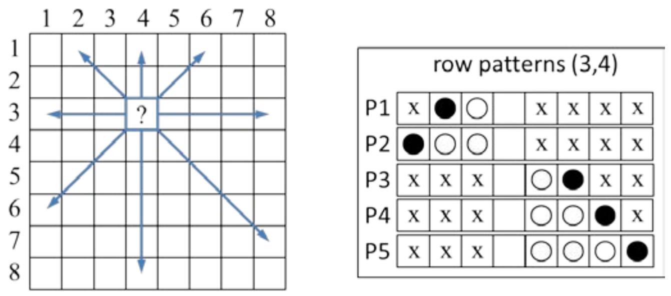
(c) Dynamic node ordering. As we have exposed before, exploring first the best movements increases the pruning efficiency. We have implemented two kinds of dynamic node ordering that attempt to explore first the most promising moves.

The first one is based on strategic concepts of Reversi, and consists in analyzing first the movements on corners, and analyzing last the movements that will help the opponent to get a corner (i.e. x-squares).

The second one uses the inferred knowledge acquired throughout incremental searches. In a search of depth *d+1,* the first movement explored is best movement of the previous search, i.e. depth *d*.

6. HARDWARE ACCELERATION

Board games offer many sources of parallelism. Parallelism is classified as fine-grained when the tasks to do in parallel require a little amount of work and have to be performed many times. The current trend in GPPs is to exploit parallelism by adding more cores. However, this scheme fails to exploit fine-grained parallelism because of communication overheads. We can illustrate this with an example from the Reversi game. The most frequent task is identifying the legal moves of a board. Fig. 2 illustrates the squares to analyze in order to determine if the square (3,4) corresponds to a legal move for the dark player, and also the specific patterns to find out whether it is a legal move due to its row configuration. These comparisons check if any of the patterns is satisfied, and the same analysis must be done for its corresponding column and diagonals. Finally, if any pattern correspondence is found, the square is a legal move.



row patterns (3,4)

P1	x	●	○		x	x	x	x
P2	●	○	○		x	x	x	x
P3	x	x	x		○	●	x	x
P4	x	x	x		○	○	●	x
P5	x	x	x		○	○	○	●

Fig. 2. Legal move analysis of the square (3,4). 'x' means 'do not care'

We carried out these computations in software and hardware as follows. Algorithm 1 shows a brief pseudo-code that finds all the legal moves of a board. It traverses the board looking for a legal move pattern in each square. To this end, four small loops are nested in such a way that the function analyzes each square in all directions and try to identify up to six different patterns. The same function is implemented in hardware using an array of combinational blocks which consists of 1920 2-bits comparators that seamlessly exploit all the data parallelism. This corresponds to a fully unroll of the four nested loops by executing every single iteration in parallel. Fig. 3 depicts a piece of hardware of one of these blocks that determines if the row satisfies the pattern 5 for the square (3,4).

ALGORITHM 1. Legal moves computation

```
Input: Current board and color.
Output: Legal moves for the input color.
function legalMove(in row, in col) returns boolean
    for each direction do
        for each pattern do
            if pattern_match(row, col, direction, pattern) then
                return true;
            end if
        end for
    end for
    return false;
end function


function allLegalMoves(out legalMoves)
    for each row do
        for each col do
            legalMoves(row, col) = legalMove (row, col);
        end for
    end for
end function
```

The execution of this function executed on an Intel i7 processor takes about 750 ns on average after both algorithmic and compilation optimizations. The code cannot be profitably parallelized because the little amount of work to parallelize (about 6,000 assembly instructions per invocation) is not enough to make up for multi-thread overheads. Using SIMD instructions is not feasible because each square requires slightly different computations.

In contrast, a hardware implementation on a FPGA exploits all the available parallelism devoting one hardware block per board square. With the 64 blocks working in parallel it just takes 7 ns to find all legal moves on a Virtex-5 FPGA, which yields a speedup factor of above 100x over the Intel i7 processor.

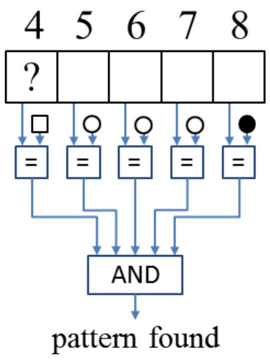


Fig. 3. Hardware architecture to identify the pattern 5 of Fig. 2

7. HARDWARE DESIGN

In this section, we present the Reversi player hardware architecture (see Fig. 4), and a description of the most relevant modules.

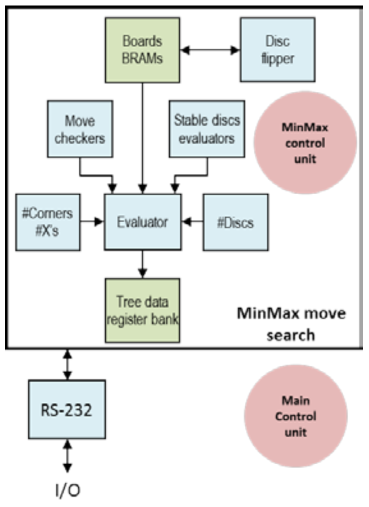


Fig. 4. Hardware architecture of the Reversi player

7.1 Move checker

This module identifies the legal moves of a board. It is a purely combinational module which implements a logic function for each square that compares its row, column, and diagonals with every legal move pattern.

7.2 Disc flipper

This module returns the new board after making a legal move. It has been implemented as an iterative network composed by 64 cells, one per each board square. This network works as follows:

(1) The cell corresponding to the selected movement begins the pattern propagation to its neighbors in every direction.
(2) Each cell propagates a new pattern in each direction based on the content of its square and its input pattern.

(3) If a cell identifies a flip pattern, it returns a flip signal in the opposite direction. Every cell that receives this signal turns its color and keeps propagating this signal until it arrives to the cell corresponding to the movement.

Table II shows the patterns propagation scheme after a movement of the white player. Three different patterns are considered: *no relevant pattern*, *one white disc*, and *one white disc followed by at least one black disc*. For the black player the design follows a similar scheme with the opposite patterns (*one black disc* and *one black disc followed by at least one white disc*).

Table II. Pattern propagation scheme

Input pattern	Square content	Output pattern
-	Don't care	-
○	●	○●1+
	○ or □	-
○●1+	●	○●1+
	○ or □	-

Fig. 5 shows a simplified example with four squares in one direction. A white disc is placed in the cell marked with an 'x'. This cell begins the pattern propagation by sending the corresponding pattern to its neighbor. The next cell propagates a new pattern according to its input pattern and the content of its square (see Table II). The third cell does the same propagation. The fourth cell completes a flip pattern, so it returns a flip signal in the opposite direction that is propagated until the starting cell is reached. All the intermediate cells that receive this signal flip their discs.

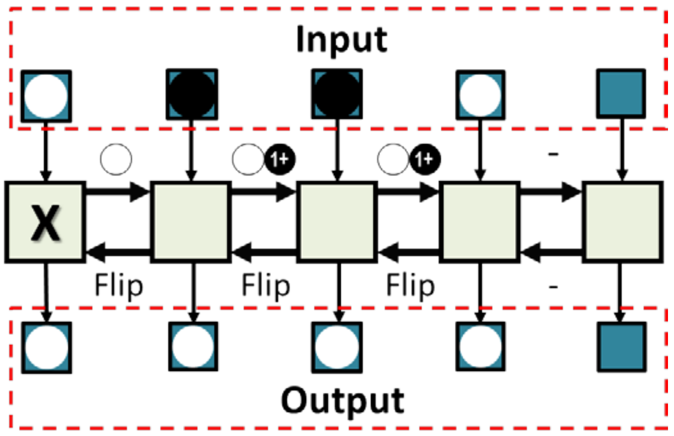


Fig. 5. Toy example of the Disc flipper network

7.3 Stable discs evaluator

This module returns the number of stable discs of a board. We implemented it as an iterative network. In this case, it was necessary to add a flip-flop to each cell because implementing it as a purely combinational module leads to combinational loops that strongly downgrade the performance by reducing the maximum frequency the design might work.

The cells corresponding to the border of the board are a special case and they can be easily evaluated using combinational logic. The layout of the module is shown in Fig. 6. The dashed lines represent the layers of the network due to the flip-flops added to each cell. This network works as follows:

(1) The combinational logic for the border squares determines which ones of them are stable discs in the first cycle.

(2) If any stable disc is detected, in the following cycles, the intermediate results are propagated in a round trip wave fashion until the results converge (i.e. no output has changed within the last cycle).

The second step takes a variable number of cycles (from 1 to 5), so we have added a 1-bit comparator to each cell that compares the current network status with the previous cycle status in order to identify when the evaluation finishes.

Finally, once the stable discs have been identified, two 64-bit tree adders return how many of them have been found for each color.

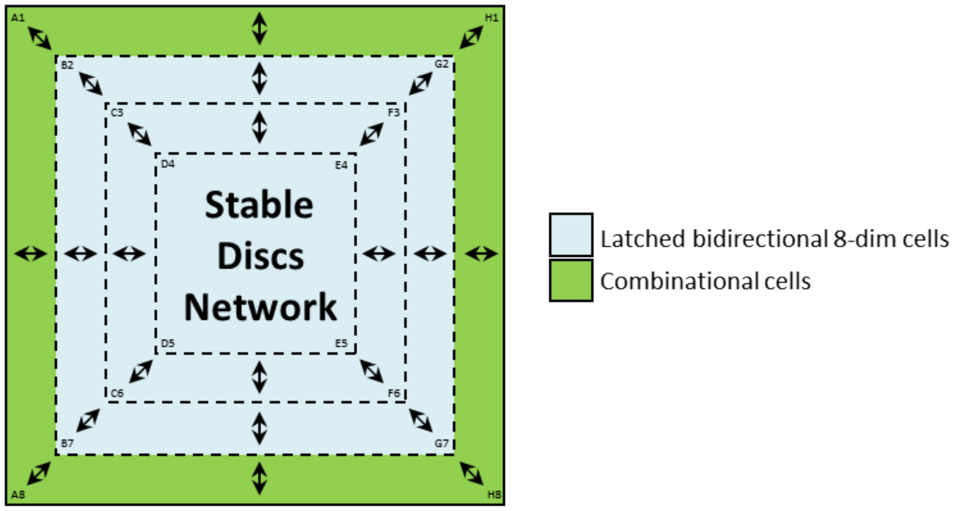
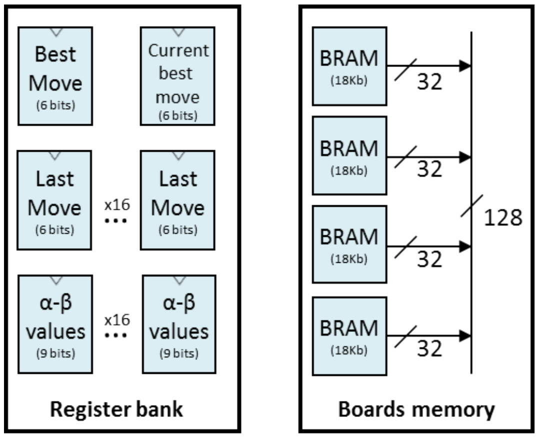


Fig. 5. Stable Discs Network layout

7.4 Boards memory and tree data register bank

These are the modules that storage the information concerning each game-tree level. *Best move* register stores the best move of the last fully explored game-tree. *Current best move* register stores the best move of game-tree under analysis. *Last move* registers store the last move analyzed in each level. *α-β values* registers store the scores propagated from the leaf nodes to upper levels according to the Minimax algorithm.

Boards memory stores all the boards of the open branch under analysis. It has been designed using several memory modules in parallel in such a way that a board can be read or written in just one clock cycle. Fig. 6 shows these two storage elements.



(a) (b)

Fig. 6. (a) Register bank module. (b) Boards memory module

7.5 Minimax control unit

Minimax control unit has been implemented as a Finite-State Machine (FSM) (see Fig. 7). State 'Find next move' performs legal moves calculation, movement reordering, and move selection. State 'Generate new board' generates the new board after placing a disc. State 'Evaluate board' evaluates a board and manages the flow control of Minimax.

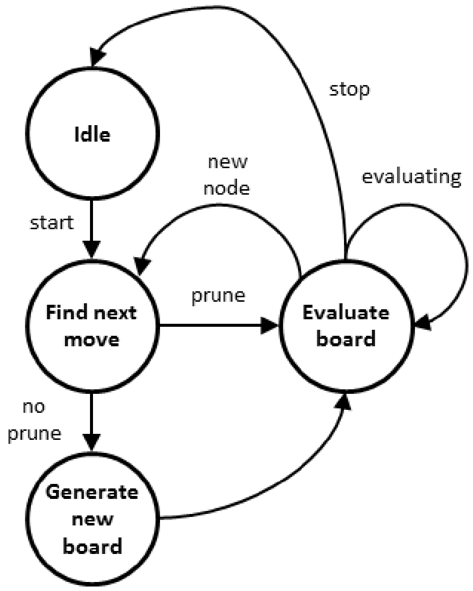


Fig. 7. Minimax FSM

8. SOFTWARE VERSION

We have developed an algorithmically equivalent software solution to be executed in general purpose processors for performance and power consumption evaluation. It has been written in C code and compiled with GCC both for Intel and ARM processors.

The application was analyzed with Intel VTune Amplifier XE 2011 [18], which is a powerful profiling software.

An initial analysis of the hotspots revealed that finding the legal moves is the most time-consuming task, so we focused on reducing its impact. Initially this function was computing the legal moves every time that a node was analyzed, as it is done in the hardware design. However, after the results of this analysis, we decided to reduce the number of invocations to this function by storing the legal moves of every open node in a specific data structure, in such a way that they are computed only when the node is initially generated. The remaining times the code loads the stored data instead of computing it again. This reduced the hotspot by 44%.

Additionally, we implemented a data structure that reduces the number of pattern checks by marking as potential legal moves only the squares that have at least one disc in any of its surrounding squares. This improvement reduced the computation by an additional 9%.

We have also compiled it with Intel C Composer XE [19], getting no relevant improvements even after tuning the compiler optimizations for each processor.

Finally, we tried to parallelize the legal moves function by using automatic parallelization feature of the GCC compiler, and by manually modifying the code using two thread libraries (POSIX Threads and Intel Threading Building Blocks). In the first case, the compiler was unable to parallelize the code. In the second case, the result was a strong performance downgrade, as expected, due to the threads management overhead.

9. PERFORMANCE ANALYSIS

We have used the Zynq hybrid FPGA/processor platform, three FPGAs, and two GPPs to evaluate the performance of hardware and software solutions. Tables III and IV summarize their main features. Note that in the case of FPGAs, the working frequency depends on the design, and in this case it is low due to the delay of the large combinational blocks.

Table III. Main features of General Purpose Processors Platforms

Platform	Processor	Tech (nm)	Year	Clock
Atom	Intel Atom D525	45	2010	1.8 GHz
i7	Intel i7-2600	32	2011	3.4 GHz
Zynq (GPP)	ARM Cortex-A9	28	2012	667 MHz

Table IV. Main features of Programmable Logic Platforms.
Clock Column indicates the design working frequency

Platform	Board	Tech (nm)	Year	Logic cells	BRAM (KB)	Clock (MHz)
Virtex II-Pro	[20]	90	2002	30K	306	32
Virtex-5	[21]	65	2006	110K	666	60
Spartan-6	[22]	45	2009	43K	261	32
Zynq (FPGA)	[23]	28	2012	85K	560	32

Table V. FPGA resources utilization

FPGA	Slice Registers	Slice LUTs	BRAMs
Virtex II-Pro	634 (2%)	8250 (30%)	8 (5%)
Virtex-5	599 (<1%)	5837 (8%)	4 (2%)
Spartan-6	622 (1%)	8251 (30%)	8 (6%)
Zynq	637 (<1%)	8834 (16%)	6 (4%)

Table V shows the occupied resources for each FPGA. Even for low-cost FPGAs, like the Spartan-6, the design occupies less than one third of the FPGA hence our design can be implemented in very small FPGAs. Regarding the cost of a software solution, if a processor is already present in the system the only cost will be to store the application in the memory subsystem. Since its memory footprint is very small (less than 500KB) any memory subsystem will provide enough storage capacity. Hence, using the software version reduces the cost of the system, but as we will see in the following sections, it also provides worst performance and more energy consumption.

9.1 Overall performance

We have used two metrics to measure the performance: boards analyzed per second, and average computing time per move.

Depending on what kind of comparison we want to carry out, we have used two slightly modified versions:
(a) Time-constrained. These designs decide the next movement in no more than one second, and their performance is measured as *boards analyzed per second*. This metric allows analyzing the behavioral differences between the software and hardware implementations during a game.

As Fig. 8 depicts, in the software version the number of boards analyzed per second increases during the game. The reason is that as the board becomes fuller of discs, the number of squares which are potential legal moves decreases, and therefore analyzing a board requires fewer instructions. On the contrary, the hardware version exhibits the opposite trend. In this case all the positions are analyzed in parallel. However, the number of boards analyzed per second slightly decreases during the game due to the stable discs evaluation. Boards without stables discs are evaluated in one cycle, whereas boards with stable discs need up to five cycles. In fact, the more stable disc the more cycles it takes to identify them.

The hardware version provides two order of magnitude better performances than the software version. It is interesting to point out that a Virtex II-Pro, which is FPGA from 2002, offers from 15 to 40 times better performance than a last-generation multi-processor chip such as the i7-2600. These results demonstrate that FPGAs are extremely efficient analyzing boards.

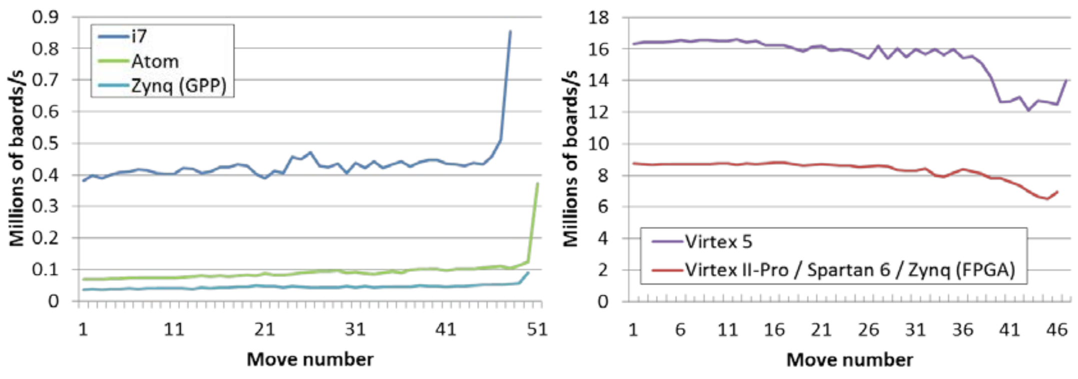


Fig. 8. Boards analyzed per second in software and hardware

(b) Fixed depth. These designs explore always the same number of movements in advance, in such a way that each movement demands the same useful workload.

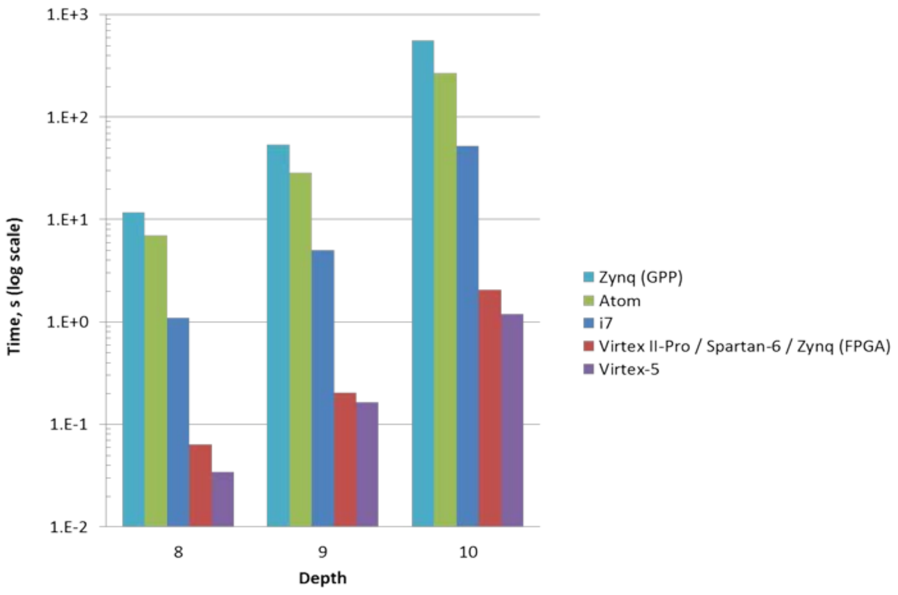


Fig. 9. Average computation time per movement

Performance for Virtex II-Pro, Spartan 6 and Zynq FPGA is exactly the same because they implement the same hardware design, running at the same frequency. Virtex 5 FPGA enables an implementation at a higher frequency; therefore it provides the best performance.

In the case of GPPs, the performance not only depends on frequency but also many other architectural details. The results evidence that i7 processor is about seven times faster than Atom, and about ten times faster than Zynq GPP.

Comparing FPGAs with GPPs, even a low-cost FPGA, Spartan 6, achieves a remarkable speedup of 25 over a high-performance processor (i7).

9.2 Task-level performance

In order to better understand the previous results, it is interesting to identify the tasks where a design achieves greater performance and the reasons for that. To carry out this analysis we have used the fixed-depth designs, in order to guarantee that software and hardware designs have the same useful workload (i.e. they have to explore the same game-tree).

A task performance analysis requires knowing the execution time of each task. For Intel processors, we profiled our application with Intel VTune in order to gather accurate execution statistics of every function. For ARM processor, we manually instrumented the code using hardware counters included in the system for accurate time measuring.

For this analysis, we have split the application in three tasks: *Find next move*, *Generate new board* and *Evaluate board*. Fig. 10 shows the average computation time for each task exploring eight moves in advance during a game.

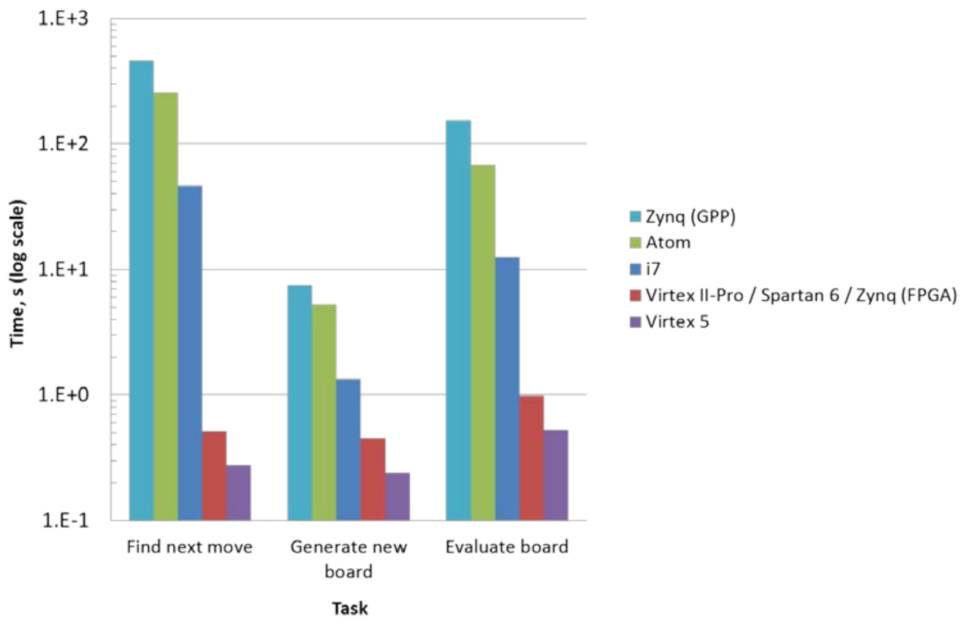


Fig. 10. Computation time of each task in hardware and software during a game

The graph makes clear that the hardware design obtains the highest speedup finding the next moves, achieving an impressive speedup of roughly 90 if we compare the high-performance processor with the low-cost FPGA. Moreover, this in not only the task where the hardware achieves the higher speedup, but this is also the most time-consuming task in software, so we conclude that this is the key to explain the performance advantage of the hardware design over the software.

The hardware exploits perfectly all the parallelism since the 64 squares of a board are analyzed in parallel, and this is done for both players also in parallel, while the software cannot profitably take advantage of data parallelism, neither by multi-threading nor by executing SIMD instructions.

10. POWER CONSUMPTION AND ENERGY EFFICIENCY

Power consumption was measured with a Yokogawa WT210 digital power meter, which is an accepted device by Standard Performance Evaluation Corporation (SPEC) for power efficiency benchmarking [24].

The power consumed by a system can be divided in two different terms: static and dynamic power. On the one hand, the static power is the power consumed by the system even when no useful work is carried out. The only way to avoid this consumption is to turn off the system. On the other hand, the dynamic power is the power consumed due to the computations carried out by the system.

Table IV shows the power consumption data of the evaluated platforms. The column 'Total Power' is the average power consumption of the whole system executing our application. Hence, it includes both the static and the dynamic terms. The column 'ΔPower' only includes the dynamic power consumption, i.e. the average increase in the power consumption due to the execution of our application. This is the most important metric since 'Total Power' includes the static power consumption of all the elements of the platform, even those that are not used.

Table IV. Average power consumption executing our Reversi application

Platform	Total Power (W)	ΔPower (W)
i7	68.31	24.19
Atom	34.20	1.07
Zynq (GPP)	4.61	0.1
Virtex II-Pro	4.79	0.42
Virtex-5	7.31	0.15
Spartan-6	3.02	0.06
Zynq (FPGA)	2.90	0.02

Energy efficiency measures useful work done per unit of energy. In this application, we define the useful work as the number of boards analyzed; therefore, combining the performance metric *boards analyzed per second* with the power data, we obtain the energy efficiency metric as shown in Eq 1.

$$Energy\ efficiency = \frac{performance}{power} = {\frac{kBoards}{s}}\Big/{watt} = \frac{kBoards}{J} \qquad (1)$$

Fig. 11 shows the energy efficiency results. Columns named as 'System' show the energy efficiency taking into account the power consumption of the whole system, while columns named as 'ΔEnergy' only take into account the dynamic power consumption.

The ARM Cortex-A9 (Zynq GPP) demonstrates to be the most energy-efficient processor whereas Intel i7 is the least efficient because it is a performance-oriented processor. In the case of FPGAs, the energy efficiency increases as the scale integration improves.

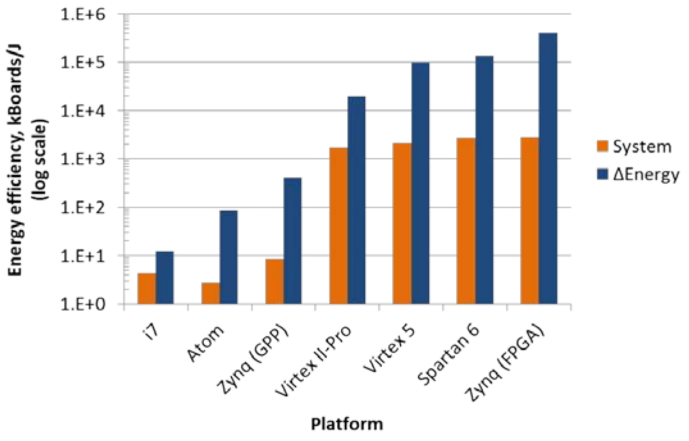


Fig. 11 Energy efficiency of each platform executing the Reversi application

It is remarkable the fact that FPGAs implementations achieves energy efficiencies from two up to four orders of magnitude higher than the software running on GPPs. Hence adding a customize hardware module for the Reversi game not only leads to an important performance improvement, but also to remarkable energy savings.

11. DEVELOPMENT EFFORT

The previous results demonstrate the utility of FPGAs to improve both the performance and the power consumption of a board game application. However, using FPGAs has some drawbacks.

The first drawback is the way to code the design. Hardware designs are usually written in a Hardware Description Language (HDL), typically VHDL or Verilog, and writing a hardware design is usually more complex than writing a software application that has the same functionality. However, if the design team has a good command of HDL, writing an initial version of the hardware design requires just a little more time than writing a software application. Moreover, FPGA vendors are doing an important effort to simplify the hardware design process. For instance Xilinx has developed a C/C++ to HDL compiler [25] that can be used to directly map a C or C++ code to an FPGA, and they have reported good results for several digital processing applications

There are two additional drawbacks associated to FPGA hardware design. The first one is the compiling time. Extracting the FPGA configuration which implements a design requires much more time than compiling software code. For large designs it may take even hours, and this is done several times during the development process, significantly slowing down the development pace. The last drawback is the debugging complexity. Once the initial design has been written, the testbenchs will likely identify several malfunctions, but debugging large hardware designs with several modules interacting and working in parallel becomes much more complex than debugging a sequential software code. Once again, FPGA vendors provide several tools to make this step more affordable, such as powerful simulators, or specific support to monitor some FPGA internal signals. Anyway, debugging a parallel system is always more complex than debugging a sequential one.

Hybrid hardware/software platforms offer a well-balanced solution to minimize the impact of these drawbacks. Those tasks that are not critical for the system performance may run on the GPP, and only very demanding tasks would be mapped to the FPGA. This approach enables to design a custom SoC with an affordable development effort.

12. CONCLUSIONS

In this article we have analyzed the performance and energy efficiency of a Reversi player implemented on FPGAs and GPPs. Board games offer many sources of fine-grained parallelism, and the experimental results demonstrate that FPGAs can seamlessly exploit it in order to speed up the execution and reduce the power consumption of the critical functions that analyze the boards.

Moreover, new hybrid platforms that tightly couple FPGAs and GPPs enable energy-efficient implementations with a reasonable design effort by implementing the core functions in hardware and the remaining functionality in software. Hence we believe that FPGAs can play an important role in general-purpose systems dealing with any kind of application that exhibits either data or task parallelism, especially for battery-dependent devices where energy efficiency is compulsory. In fact, as we have shown in our case study, they can be the key to provide the performance of powerful desktop processors in low-power devices.

As future work, it would be interesting to explore different co-design possibilities, such as managing all the computations on the programmable logic, and leaving the I/O in the processor side, or using the programmable logic to process the boards while the processor executes the search algorithm. We believe that it will be especially interesting to carry out a comprehensive evaluation of the communication schemes between the processor and the programmable logic in order to decide the best co-design alternative.

ACKNOWDELGEMENTS

This work was supported in part by grants TIN2010-21291-C02-01 (Spanish Gov. and European ERDF), Consolider CSD2007-00050 (Spanish Gov.), gaZ: T48 research group (Aragón Gov. and European ESF).

REFERENCES

[1] OMAP™ Mobile Processors: OMAP™ 5 Platform. Retrieved March 13, 2014 from www.ti.com/lsds/ti/omap-applications-processors/omap-5-processors-products.page?paramCriteria=no
[2] Intel® Atom™ Processor E6x5C Series. Retrieved March 13, 2014 from http://www.intel.com/p/en_US/embedded/hwsw/hardware/atom-e6x5c/overview
[3] Zynq-7000 Extensible Processing Platform. Retrieved March 13, 2014 from http://www.xilinx.com/products/zynq-7000/extensible-virtual-platform.htm
[4] Arria V SoC FPGA Hard Processor System. Retrieved March 13, 2014 from http://www.altera.com/devices/fpga/arria-fpgas/arria-v/hard-processor-system/arrv-soc-hps.html
[5] Google Project ARA. Retrieved March 24, 2014 from http://www.projectara.com/
[6] Zebra – Gunnar Andersson's Homepage. Retrieved March 24, 2014 from http://www.radagast.se/othello/
[7] C.K.. Wong, K.K.. Lo and P.H.W. Leong. 2004. *"An FPGA-Based Othello Endgame solver"*. International Conference on Field-Programmable Technology 2004 (ICFPT '04). Brisbane, NSW, Australia, pp. 81–88.
DOI:http://dx.doi.org/10.1109/FPT.2004.1393254
[8] FPT '10 Design Competition. Retrieved March 13, 2014 from http://cas.ee.ic.ac.uk/people/as999/FPTDesignComp/
[9] J. Olivito, C. González and J. Resano. 2010. *"FPGA implementation of a strong Reversi player"*. International Conference on Field-Programmable Technology 2010 (ICFPT '10). Beijing, China, pp. 507–510.
DOI:http://dx.doi.org/http://dx.doi.org/ 10.1109/FPT.2010.5681469
[10] T. Mabuchi, T. Watanabe, R. Moriwaki, Y. Aoyama, A. Gundjalam, Y. Yamaji, H. Nakada and M. Watanabe. 2010. *"Othello Solver based on a soft-core MIMD processor array"*. International Conference on Field-Programmable Technology 2010 (ICFPT '10). Beijing, China, pp. 511–514.

DOI:http://dx.doi.org/http://dx.doi.org/10.1109/FPT.2010.5681470

[11]M. Smerdis, P. Malakonakis and A. Dollas. 2010. *"CarlOthello: An FPGA-Based Monte Carlo Othello player"*. International Conference on Field-Programmable Technology 2010 (ICFPT '10). Beijing, China, pp. 515–518.
DOI:http://dx.doi.org/10.1109/FPT.2010.5681471

[12]N. Aranki, D. Keymeulen, A. Bakhshi and M. Klimesh. 2010. "*Hardware Implementation of Lossless Adaptive and Scalable Hyperspectral Data Compression for Space*". *NASA/ESA Conference on Adaptive Hardware and Systems (AHS '09).* San Francisco, California, 315–322.
DOI:http://dx.doi.org/10.1109/AHS.2009.66

[13]Y. Guoxia, T. Vladimirova and M. N. Sweeting. 2009. FPGA-based on-board multi/hyperspectral image compression system. *IEEE International Geoscience and Remote Sensing Symposium (IGARSS '09).* Cape Town, South Africa, Vol 5, 212–215.
DOI:http://dx.doi.org/ http://dx.doi.org/10.1109/IGARSS.2009.5417693

[14]B. Cope, P.Y.K. Cheung, W. Luk, and L. Howes. 2010. Performance Comparison of Graphics Processors to Reconfigurable Logic: A Case Study. *IEEE Transactions on Computers*. Vol. 59, no. 4, 433–448.
DOI:http://dx.doi.org/ http://dx.doi.org/10.1109/TC.2009.179

[15] J. Cong, et al., 2011. Customizable Domain-Specific Computing. *IEEE Design and Test of Computers*. Vol. 28, no. 2, 6–15.

[16] Othello: A Minute to Learn... A Lifetime to Master. Retrieved March 13, 2014 from http://othello.federation.free.fr/livres/othello-book-Brian-Rose.pdf

[17]S.J. Russell and P. Norvig. 2003. Artificial Intelligence: A Modern Approach. Upper Saddle River, New Jersey: Prentice Hall.

[18] Intel® VTune™ Amplifier XE 2011 | Intel® Developer Zone. Retrieved March 13, 2014 from http://software.intel.com/en-us/intel-vtune-amplifier-xe

[19] Intel® Composer XE 2011 | Intel® Developer Zone. Retrieved March 13, 2014 from http://software.intel.com/en-us/intel-composer-xe

[20] Xilinx University Program Virtex-II Pro Development System. Retrieved March 13, 2014 from http://www.xilinx.com/products/boards-and-kits/XUPV2P.htm

[21] Xilinx University Program XUPV5-LX110T Development System. Retrieved March 13, 2014 from http://www.xilinx.com/univ/xupv5-lx110t.htm

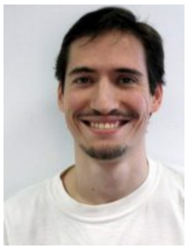
[22] Atlys™ Spartan-6 FPGA Development Board. Retrieved March 13, 2014 from http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,836&Prod=ATLYS&CFID=266512&CFTOKEN=74324583

[23] ZedBoard Zynq™-7000 Development Board. Retrieved March 13, 2014 from https://www.digilentinc.com/Products/Detail.cfm?Prod=ZEDBOARD

[24] WT210/WT230 Digital Power Meters. Retrieved March 13, 2014 from http://tmi.yokogawa.com/products/digital-power-analyzers/digital-power-analyzers/wt210wt230-digital-power-meters/#tm-wt210_01.htm

[25] 2014. Xilinx Vivado Design Suite. Retrieved March 13, 2014 from http://www.xilinx.com/products/design-tools/vivado/index.htm

Javier Olivito received the MS degree in computer engineering in 2010 from the University of Zaragoza, Spain. Currently he is PhD student in the GAZ research group, from University of Zaragoza. His research has been focused in hardware/software co-design, acceleration of artificial intelligence algorithms in board games and dynamic reconfiguration. His FPGA designs have received several international awards including the first prize in the Design Competition of the IEEE International Conference on Field-Programmable Technology in 2009 (FPT'09) and in 2010 (FPT'10).



Javier Resano received the Bachelor Degree in Physics in 1997, a Master Degree in Computer Science in 1999, and the Ph.D. degree in 2005 at the Universidad Complutense of Madrid, Spain. Currently he is Associate Professor at the Computer Eng. Department of the Universidad of Zaragoza, and he is a member of the GHADIR research group, from Universidad Complutense, and the GAZ research group, from Universidad de Zaragoza. He is also member of the Engineering Research Institute of Aragon (I3A). His research has been focused in hardware/software co-design, task scheduling techniques, Dynamically Reconfigurable Hardware and FPGA design. He have designed hardware accelerators for different fields, including remote sensing and artificial intelligence, and his designs have received several international awards including the first prize in the Design Competition of the IEEE International Conference on Field Programmable Technology in 2009 and in 2010 and the second prize in 2012.

Carlos González received the M.S. and Ph.D. degrees in computer engineering from the Complutense University of Madrid, Madrid, Spain, in 2008 and 2011, respectively.

He is currently a Teaching Assistant in the Department of Computer Architecture and Automation of the Universidad Complutense Madrid. As a research member of GHADIR group, he mainly focuses on applying run-time reconfiguration in aerospace applications. His research interests include remotely sensed hyperspectral imaging, signal and image processing, and efficient implementation of large-scale scientific problems on reconfigurable hardware. He is also interested in the acceleration of artificial intelligence algorithms applied to games.

Dr. González won the Design Competition of the IEEE International Conference on Field Programmable Technology in 2009 (FPT'09) and in 2010 (FPT'10). He received the Best Paper Award of an Engineer under 35 years old in the International Conference on Space Technology in 2011.

Enrique Torres received the MS degree in computer science from the Polytechnic University of Catalunya in 1993, and the PhD degree in computing science from the University of Zaragoza in 2005. He was an assistant professor in the Polytechnic Schools of the University of Girona. He is an assistant professor in the Computer Science and Systems Engineering Department (DIIS) at the University of Zaragoza, Spain. He was also on sabbatical leave for study and research at the University of California in Berkeley, where he was a member of the International Computer Science Institute (ICSI). His research interests include processor microarchitecture, memory hierarchy, parallel computer architecture heterogeneous computing with FPGAs & GPUs. He is a member of the IEEE Computer Society. He is also a member of the Aragón Institute of Engineering Research (I3A) and the European HiPEAC NoE. More details about his research and background can be found at http://webdiis.unizar.es/gaz/miembros.html.

Rubén Gran graduated in Computer Science from the University of Zaragoza (Spain) and held his PhD in 2010 from the Polytechnic University of Catalonia (UPC, Spain). Since then, he is assistant professor in the Department of Computer Science and Systems Engineering at the University of Zaragoza. His research interests are hard real-time systems, hardware for reducing worst-case execution time and energy consumption, microarchitecture and effective programming for parallel and heterogeneous systems.