A Threat Based Connect6 Implementation on FPGA

Kizheppatt Vipin, Suhaib A. Fahmy
School of Computer Engineering
Nanyang Technological University
Nanyang Avenue, Singapore
vipin2@e.ntu.edu.sg, sfahmy@ntu.edu.sg

***Abstract*—Connect6 is a new generation k-in-a-row game, which has drawn great interest not only from game enthusiasts but also from researchers, due to its unique characteristics such as fairness and high state-space complexity. In this paper we describe the design and implementation of an FPGA-based Connect6 player that can compete against opponents by communicating through a serial interface. Our algorithmic implementation utilises only basic FPGA building blocks such as LUTs and flip-flops and does not include any IP cores or hardware macros, making it portable across different FPGA platforms without design modifications. The design has been implemented and validated on a Xilinx Spartan-3A FPGA board, and a Xilinx Spartan-6 board. The algorithm uses a powerful threat-based placement strategy, which maximises the FPGA's winning opportunity while reducing the opponent's options. Extended simulation results and evaluation based on software and human players confirm that our FPGA based implementation performs well and the algorithm used in the design leads to a high probability of success.**

I. Introduction

Connect6 was introduced in 2003 by Professor I-Chen Wu at Department of Computer Science and Information Engineering of National Chiao Tung University, Taiwan. It is a form of k-in-a-row game played between two players usually on a 19×19 cell board using black and white stones. The player with black stones starts the game, placing a single stone. In all subsequent moves, each player places two stones. The single-stone first move is designed to increase fairness and avoid any sort of advantage. The first player to place 6 stones in a row (horizontally, vertically or diagonally) wins the game. Connect6 has attracted great interest due to its fairness and higher state-space complexity compared to other variations of k-in-a-row games. Although several software based solutions have been developed [1], a complete FPGA logic-based implementation is challenging, mainly due to the large solution space.

Like other k-in-a-row games, there are two aspects to winning a Connect6 game. First is the offensive strategy, which should ensure that each stone placed increases the chance of winning, not just locally, but considering the global solution too. Hence, each stone needs to be placed in such a manner that it contributes to a current winning sequence. At the same time, if the current sequence is defended by the opponent, it should still contribute to some future sequences.

The other strategy is for defence. The player must defend sequences that would immediately cause the opponent to win, while also monitoring global placements by the opponent, which may create serious threats in the future. A good algorithm uses both offensive and defensive moves in proportion and leads to a high probability of success.

In this paper we describe the design and implementation of an FPGA based Connect6 player, as a submission to the FPT 2011 Design Competition. Our algorithm uses a threat based strategy for finding a suitable location for placing each stone. This design does not use any embedded or soft processors in the design; it is implemented entirely using logic elements such as LUTs and flip-flops. The design is coded using the Verilog hardware description language and has been extensively simulated using Mentor Graphics' Modelsim simulator to ensure it is free of glitches. We have also implemented a Tcl/Tk based GUI that integrates with the simulator, allowing human players to play against the FPGA logic. Presently the design is implemented and hardware validated on a Xilinx Spartan-3 700A-4C FPGA on a Spartan-3A evaluation board [2] using the Xilinx ISE 13.2 design tool chain. The highly pipelined and parallel implementation helps the design achieve a high clock frequency of 79 MHz, even on this relatively low-end FPGA. The algorithm takes a very short time to make a move (7 ms in worst case), which also helps, by giving the opponent minimal time to perform any pre-calculations in a time-limited game.

The rest of this paper is organised as follows: Section II discusses the algorithm, Section III describes the architecture of our system, Section IV presents the simulation results and performance of our implementation, and Section V concludes the paper.

II. Playing Strategies

This section describes the playing strategies adopted by our algorithm. The algorithm used is highly effective for both defensive and offensive moves.

A. Threat

In Connect6, a player is said to have t threats, if the opponent has to place t stones to prevent the player from winning in his next move. Fig. 1 shows three example threats for a player with black stone. The dotted circles represent the locations, where the opponent has to place stones to preventing black from winning. Since in each turn a player can place a maximum of two stones, they can defend only up to two threats. Hence, the basic winning strategy in this game is to create at least three threats in one turn.

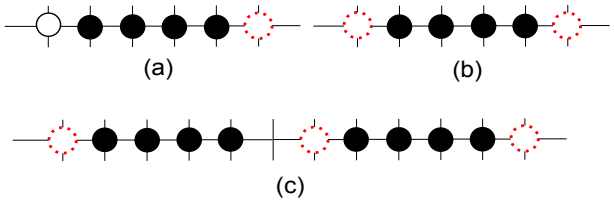

Fig. 1. (a) Single threat, (b) Two threats, (c) Three threats

A simple algorithm is used for detecting threats. Firstly a one-dimensional window with 6 cells is selected. This is called a *threat window*. Defensively, if there are 4 or more opponent stones and no player stones present within this window, a threat is present. This is because in the next move, by placing stones in the empty locations, the opponent can win. The total number of threats present on the board can be detected by sliding the threat window in all directions and counting the threats.

For offence, this same algorithm can also be used to detect sequences for immediate success. In this case the algorithm checks for a window with 4 or more of the player's stones and no opponent stones. By placing stones in the empty cells, the player can immediately win the game.

A threat can be defended by disrupting the continuity of the opponent's stone placements. By placing a stone at the rightmost empty cell within a threat detected window, it can be effectively defended. This lemma is proven in [3].

B. Weight Function

Our algorithm places stones based on the weight function of the cells. In each turn, a stone is placed at a cell having the maximum weight. A cell which will lead to immediate success has the maximum weight value. A cell which can defend a threat has the next highest weight. This ensures that if there is an opportunity for immediate success, the FPGA will win the game. If it is not possible to win immediately, the algorithm will defend all threats (up to a maximum of two).

If there is neither scope for immediate success nor any existent threats, placement of stones becomes non-trivial. A weight function is calculated for each cell and the cell with maximum weight value is selected for placement. The weight value for each cell is calculated by considering several factors.

As shown in Fig.2, placing a stone at a location influences several other cells. In the ideal case, a placement affects 5 nearby cells in all 8 directions. This is because of the fact that in Connect6, 6 adjacent placements are needed to win. A cell is given a high weight value, if it can create an additional threat to the opponent. Threats are critical, since creating three or more will definitely lead to success. Moreover, creating threats causes the opponent to defend them, rather than attacking. The next consideration is preventing the opponent from creating serious threats in future. For this purpose, the opponent weight function is calculated as follows. For a cell, the directions which can contribute to future success of the opponent are selected. These directions can be found using the previously described *threat window* concept. For a direction, if there are

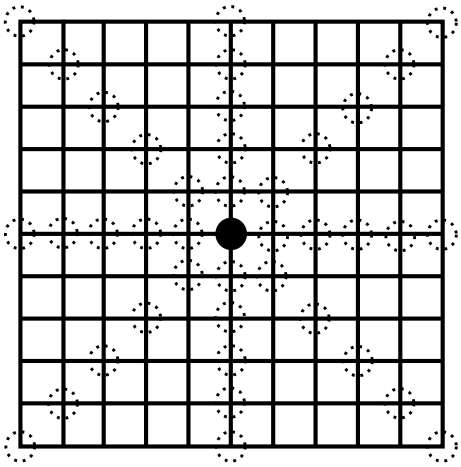

Fig. 2. Calculation cell weight functions.

no player's stones present with in a *threat window*, a future success is possible for the opponent in that direction.

The total weight of the i^{th} cell is calculated as,

$$W_i = \sum_d \sum N_d * W_d; \tag{1}$$

where d = 1,2,... the total number of success possible directions, N_d = the number of opponent stones in direction d, W_d = the weight for direction d

The weight for a direction (W_d) is proportional to the number of opponent stones present in that direction. This makes sure that consecutive placement of several stones in a single direction will cause a large weight value. In our algorithm, we consider only the four principle directions, row, column and the two diagonals for the weight calculation. The four principle directions are created by merging the collinear directions among the possible 8 directions. The cell with maximum weight value due to opponent placement is selected for placement, only if the opponent weight value is above a threshold value. This makes sure that the FPGA defends critical opponent placements and at the same time is not always stuck with defensive moves.

If the opponent weight value is below the threshold, we consider the player weight function. This is calculated similarly, but considering the player's current placement. Placement is done at the cell with the maximum weight value. This placement helps in generating future threats to the opponent.

When selecting cells, there can be more than one cell with the same weight value. In this case a tie-break is required, based on *board weight* values. Each cell in the board has a pre-defined *board weight*. The cells towards the centre of the board have higher weight and this decreases towards the edges. This is because towards the centre of the board, there are more degrees of freedom in placement directions compared to the edges. In both defensive and offensive moves, if more than one cell has the same weight value, the cell with higher *board weight* is selected for placement.

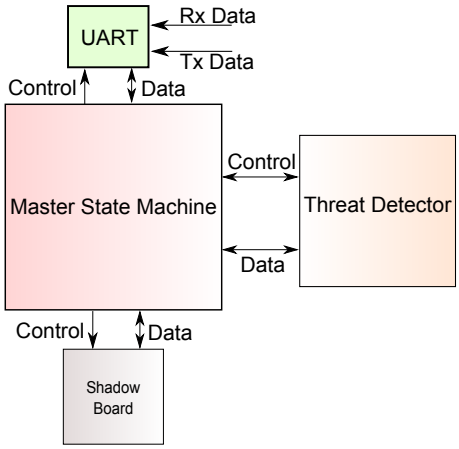

Fig. 3. Connect6 system architecture.

III. System Architecture

The overall system architecture is represented in Fig.3.

A. Master State Machine (MSM)

The MSM module manages the overall functionality of the system. It interfaces with the UART module, and manages data reception and transmission operations. It also controls the internal data storage and gives instructions to the main algorithmic state machine to calculate different weight functions. The sequence of operations performed by the MSM is described in Fig.4. Initially the system waits for the referee's signal to start the game. If the FPGA is playing with black stones, a single stone is placed at the location with the maximum weight function. Since at this point of time, no previous placements are present, both opponent and FPGA cell weights will be zero. Hence the placement will be at a location with maximum *board weight*, i.e. *(10,10)*. If the FPGA is playing white, initial placement is done very close to the opponent placement. For all the subsequent placements, weight functions are calculated and placement is performed as described in section II.

B. UART

The UART module is used for receiving and transmitting data through an RS232 interface. It is configured at a baud rate of 115200, with 8-bit data, no parity and 1 stop bit. This module also contains an internal buffer to preventing data overflow.

C. Shadow Board

The shadow board is a representation of the actual playing board. It contains a 19×19 array of distributed memory elements, with each element being capable of storing two bits. A memory element contains the information such as whether the corresponding cell in the playing board is vacant, occupied by an FPGA stone or occupied by an opponent stone. Whenever the FPGA receives or transmits information about placing a stone, it is updated in the shadow board by the MSM.

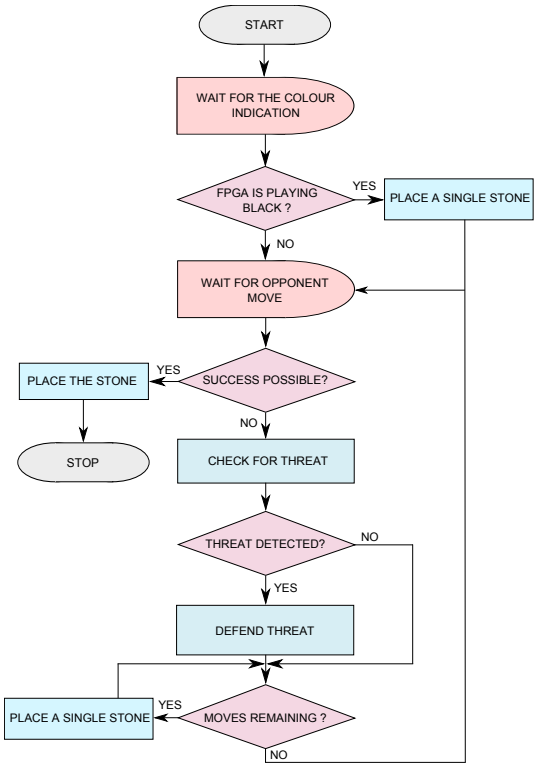

Fig. 4. Connect6 flowchart.

D. Threat Detector

The threat detector (TD) implements the main algorithm. The TD detects immediate winning sequences, immediate threats as well as the weight value for each cell. Taking advantage of parallel execution of hardware elements, immediate winning sequences and immediate threats are detected concurrently. If there is an immediate winning sequence, TD indicates this to the MSM with the corresponding cell locations. In the absence of this sequence, it indicates the locations of the cells, where defensive placements are required. If both the cases are absent, it finds the cell with maximum weight value and intimates the MSM.

IV. Performance

A. Implementation

The design has been implemented on different FPGA platforms to confirm its portability, using Xilinx ISE-13.2 tool chain. Implementation results are shown in Table-I. For the Spartan-6 FPGA, required area is smaller due to its 6-input LUT architecture, while the improved fabric results in a better maximum frequency. Currently it is hardware-validated on Spartan-3A evaluation board to demonstrate its capabilities even on a low end board. On this platform, the design uses 37% of the available Slices in the FPGA. Due to pipelined implementation, our design is able to achieve a high clock frequency of 79MHz. Currently the FPGA is running at the on-board clock frequency of 50MHz.

B. Simulation

The design has been simulated and verified using Mentor Graphics' Modelsim-SE 6.6 simulator. To check the entire

TABLE I
IMPLEMENTATION RESULTS ON DIFFERENT FPGAS

FPGA	Resouce type			F_{max}(MHz)
	FFs	LUTs	Slices	
Spartan 3 700A-4	3329	1615	2211	79
Spartan 6 LX45-3	1405	2392	789	130
Virtex 5 FX70T-3	1471	2168	943	175

board for the presence of a winning sequence or threats, the system takes 1.66 ms (8300 clock cycles). To calculate the weight value of every cell on the board, the system takes 2.56 ms (12800 clock cycles). Hence, the worst case scenario – when the FPGA has to place two stones based on the weight values of the cells, due to the absence of winning sequences and threats – is 6.78 ms (33900 clock cycles). The weight values are recalculated prior to placement of the second stone, since the first placement can influences up to 40 other cells. A possible improvement would be to restricting the updating of weights to within the influenced region.

C. Testing

The FPGA implementation has been tested against several opponents. Against the provided software host, our FPGA implementation wins in all cases, playing as both black and white, as shown in Table-II. Note that the number of moves

TABLE II
FPGA LOGIC VS HOST SW

No. of Games	FPGA Col.	FPGA Win	SW Win	Avg. Moves
25	Black	25	0	58
25	White	25	0	57

represent the total number of (both FPGA and opponent) stones present on the board, when the game ends. Due to the random placement nature of the software, the number of moves required is different in each game. In the best case, the FPGA was able to beat the software in 22 moves and in the worst case it took 206 moves. On average, the FPGA takes 58 moves to beat the host software whatever colour it plays.

For more extensive testing, a Tcl/Tk GUI was developed (Fig.5), which was configured to directly interface with the simulator. This allows humans to play against the FPGA logic. The design was tested against a Java-based Connect6 implementation [4] that has different expert levels. The FPGA design lost only against an advanced level opponent, and only when playing with white stones, as shown in Table-III.

TABLE III
FPGA LOGIC VS JAVA SW

SW Level	FPGA Col.	Winner	Avg. Moves
Random	Black	FPGA	15
Random	White	FPGA	11
Simple	Black	FPGA	25
Simple	White	FPGA	23
Hard	Black	FPGA	53
Hard	White	FPGA	26
Advanced	Black	FPGA	35
Advanced	White	S/W	35

With the help of the GUI, a number of students of different expertise levels were encouraged to play against the FPGA logic, including some seasoned experts. In most cases, the FPGA design was able to win.

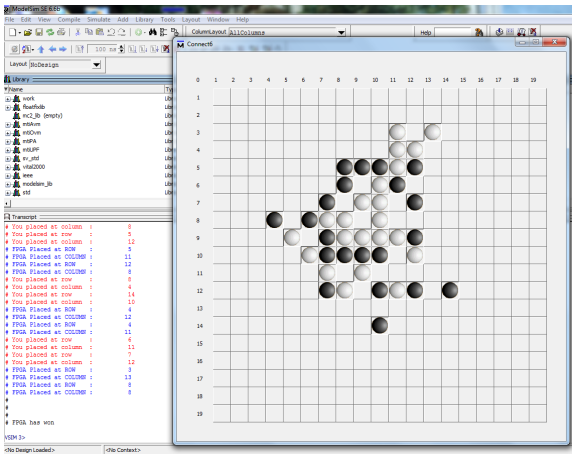

Fig. 5. Connect6 GUI.

TABLE IV
FPGA LOGIC VS HUMAN PLAYERS

No. of Games	FPGA Col.	FPGA Win	Human Win
25	Black	20	5
25	White	22	3

As a final test, one instance of the FPGA design was played against another instance of the same design. This led to a tie, with all the cells in the board getting filled. This proves that FPGA logic can play equally well using both black and white stones.

V. CONCLUSION

In this paper we described the implementation of a Connect6 player on an FPGA. The threat-based algorithm used proves to be very effective in achieving a high rate of success, and is amenable to hardware acceleration. One immediate aim is to improve the speed and performance of the system. In the current design, the shadow board (SB) implementation using distributed memory results in a large combinational logic block. This logic is the main limiting factor in achieving higher clock speed. By moving the SB to Block RAMs, we expect to improve the maximum clock frequency with minimal effect on architecture dependency, as modern FPGAs all have Block RAMs. Increasing system frequency above the board default oscillator using a Digital Clock Manager (DCM) can further reduce the total time taken for placements. We are interested in seeing how we could incorporate non-determinism in the selection of the moves to play in a future version, in order to allow more paths to winning. In a future implementation, we intend to incorporate advanced cell searching methods such as the one described in [5].

REFERENCES

[1] connect6 org. [Online]. Available: http://www.connect6.org/

[2] Xilinx Inc., *ug334*, June 2008. [Online]. Available: http://www.xilinx.com/

[3] I. C. Wu and D. Y. Huang, "A new family of k-in-a-row games," in *Proceedings of The 11th Advances in Computer Games (ACG11)*, 2006.

[4] K. Verhoef. [Online]. Available: http://kevinverhoef.nl/connect6.htm

[5] I. C. Wu and P. H. Lin, "Relevance-zone-oriented proof search for connect6," *IEEE Transactions on computational intelligence and AI in games*, vol. 2, pp. 191–207, 2010.