Universidad Zaragoza
1542

Trabajo de fin de máster

Máster en Ingeniería de Sistemas e Informática

Diseño e implementación de un jugador artificial de Reversi sobre una FPGA

Director: Javier Resano Ezcaray

Autor: Javier Olivito del Ser

Departamento de informática e ingeniería de sistemas

gaZ

Grupo de Arquitectura de Computadores de Zaragoza

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Programa oficial de posgrado en Ingeniería Informática

Curso 2010-2011

Diciembre 2011

Índice

Índice de figuras

Índice de tablas

Diseño e implementación de un jugador artificial de Reversi sobre una FPGA

Resumen

El Field-Programmable Technology Design Competition es un concurso de diseño hardware internacional enmarcado en el International Conference on Field-Programmable Technology, congreso internacional de la región asiática sobre hardware reconfigurable. En su edición de 2010 propuso el desarrollo de un procesador específico para jugar al Reversi sobre una FPGA.

Partiendo de conocimientos nulos acerca de la estrategia subyacente al juego, diseñamos e implementamos en 4 meses un procesador muy superior al software de referencia que suministraba la organización del concurso. El procesador implementa el algoritmo MinMax con poda alfa-beta, búsqueda en profundidad iterativa y ordenación dinámica de nodos para la exploración del espacio de búsqueda, y una evaluación de nodos basada en conceptos fuertemente ligados a la estrategia del juego, tales como movilidad, captura de esquinas o casillas estables.

Posteriormente, desarrollamos una versión software algorítmicamente equivalente con el propósito de establecer comparativas de rendimiento y de consumo FPGA/PC. Los resultados muestran un mayor rendimiento del diseño hardware, fruto principalmente de la explotación del paralelismo y del diseño de una arquitectura a medida, y un consumo sustancialmente inferior, debido principalmente a que el procesador desarrollado trabaja a una frecuencia dos órdenes de magnitud inferior al PC. Como contrapartida, el tiempo de desarrollo del diseño hardware fue claramente superior que el del diseño software equivalente.

El diseño presentado en la sesión del congreso dedicada a la competición fue capaz de batir al resto de finalistas, y por ello fuimos galardonados con el primer premio de la competición. Además, el artículo describiendo el diseño fue publicado en las actas del congreso, siendo accesible a la comunidad científica a través del IEEExplore.

1. Introducción

El FPT Design Competition (en adelante, concurso) es un concurso internacional de diseño hardware que se celebra anualmente como parte del International Conference Field-Programmable Technology, congreso internacional de la región asiática sobre hardware reconfigurable. En su edición de 2010 propuso el diseño e implementación de un jugador artificial de Reversi sobre una FPGA.

Este trabajo tiene los siguientes objetivos:

- Participar en el FPT Design Competition 2010
- Profundizar en el diseño hardware avanzado
- Crear un *benchmark* de soluciones hardware-software algorítmicamente equivalentes e instrumentadas para un posterior estudio comparativo de rendimiento y consumo energético
- Introducirse en las medidas de consumo energético de distintas plataformas de computación

El resto de la memoria sigue la siguiente estructura de contenidos:

La sección 2 describe la inteligencia artificial implementada en el procesador.

La sección 3 detalla la implementación hardware que se ha llevado a cabo.

La sección 4 muestra los resultados en términos de rendimiento y consumo.

La sección 5 expone las conclusiones obtenidas a raíz del trabajo realizado.

La sección 6 contempla las posibles líneas de trabajo futuro.

La sección 7 ilustra la distribución temporal de tareas a lo largo del trabajo.

La sección 8 contiene las referencias más relevantes utilizadas en este trabajo.

Finalmente, el anexo I contiene el artículo que se publicó en el concurso.

1.1 ¿Qué es el Reversi?

Reversi [1] (también conocido como Othello) es un juego de tablero que enfrenta a dos jugadores. Se desarrolla sobre un tablero de 8x8 y con 64 fichas, cuatro de ellas inicialmente colocadas como se muestra en la figura 1. El juego comienza moviendo negras y alternando entre ambos jugadores hasta que ninguno pueda realizar un movimiento legal. Un movimiento legal consiste en colocar una ficha del color propio en una casilla vacía, de manera que se flanqueen una o más fichas del color contrario en cualquiera de las direcciones (horizontal, vertical, diagonal). Aquellas fichas flanqueadas pasan a ser del color del jugador que hizo el movimiento.

El objetivo del juego es tener más fichas que el rival al final de la partida.

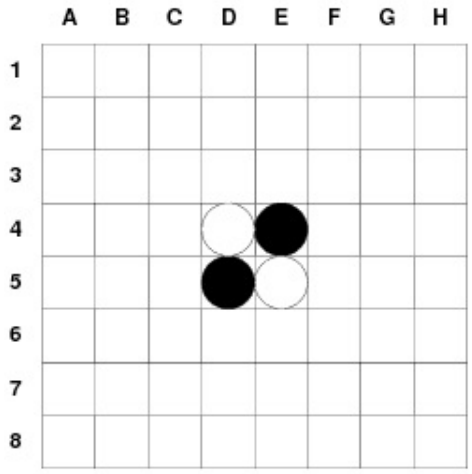


Fig. 1. Disposición inicial de fichas al comienzo de la partida

1.2 ¿Qué es una FPGA?

Una FPGA (*Field Programmable Gate Array*) es un circuito integrado que contiene bloques de lógica, elementos de memoria e interconexiones, todos ellos programables, así como bloques específicos de E/S (figura 2). La configuración de la FPGA mediante la interconexión de los bloques lógicos y la funcionalidad de los mismos, permite generar el sistema lógico deseado.

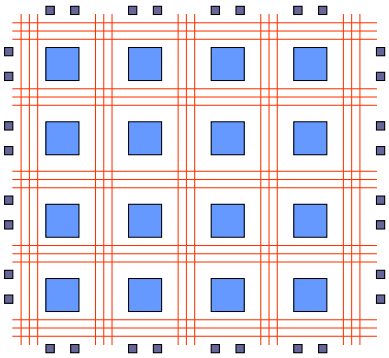


Fig. 2. Estructura típica de una FPGA con bloques de lógica rodeados de elementos de interconexión y celdas de entrada/salida rodeando el chip

La descripción del sistema lógico que se desea diseñar se suele realizar mediante el uso de un lenguaje de descripción de hardware, siendo los más usados VHDL (acrónimo de VHSIC HDL, *Very High Speed Integrated Circuit Hardware Description Language*) y Verilog.

1.3 Especificaciones del concurso

Las especificaciones detalladas más relevantes son:

- Se dispone de un segundo para realizar un movimiento.

- El diseño se evaluará inicialmente contra un software proporcionado por el concurso. Dicho software dispone de tanto tiempo como precise para realizar un movimiento y posee siete niveles de dificultad.

- Las métricas para evaluar el diseño contra el software son:
  - Número de fichas logradas al finalizar la partida para cada nivel de dificultad
  - Número de movimientos realizados para cada nivel de dificultad.

- Los tres mejores diseños evaluados según lo anterior disputarán la final. La final consiste en un torneo round-robin que enfrenta FPGA vs FPGA con un referee de por medio Este torneo se realizará en una de las sesiones del congreso.

- La comunicación de los movimientos se realizará mediante RS-232. Cada envío consta de dos caracteres ASCII de acuerdo a las siguientes convenciones:
  - El juego comienza con el envío por parte del referee de los caracteres “YY” a la FPGA que mueva negras.
  - Un movimiento se comunica indicando la columna y fila del movimiento realizado de acuerdo con la notación utilizada en el tablero de la figura 1.
  - Si no posible mover en algún punto de la partida, se debe enviar “VV” al referee. Si es cierto, el referee lo notificará al oponente enviándole “RR”.
  - Si un jugador realiza un movimiento inválido, el referee le enviará “II” indicándole que ha perdido la partida, y además enviará al rival “XX” notificándole que ha ganado la partida.

- La FPGA sobre la que se implementará el diseño debe ser una de las siguientes:

    - DE2 Development and Education Board
    - XUP Virtex-II Pro Development System
    - Xtreme DSP Starter Platform – Spartan-3A DSP 1800A Edition
    - DE2-70 Development and Education Board
    - Altium NanoBoard 3000

  Nuestro diseño ha sido implementado sobre la XUP Virtex-II Pro Development System (en adelante, FPGA).

1.4 Trabajo relacionado

El Reversi se ha estudiado en diversos trabajos científicos. Destaca especialmente el trabajo realizado por Michael Buro [2] [3] [4] en el área de inteligencia artificial aplicada a juegos de tablero.

En cuanto al desarrollo de un procesador específico para jugar al Reversi, cuando empezamos este trabajo solamente existía una contribución en un congreso del área [5], este trabajo ere interesante por ser la primera vez que se hacía algo así, pero podemos decir que presenta un procesador muy simple, incapaz de realizar un juego avanzado.

En el mismo congreso en el que se presentó nuestro procesador, se presentaron también nuestros dos rivales. De ellos, el trabajo presentado en [6] presenta una aproximación basada en la utilización de procesadores dentro de la FPGA que proporciona un diseño rápido de realizar, pero no muy eficiente. En cuanto al trabajo presentado en [7] es una aproximación probabilística basada en Monte-Carlo que proporciona un rendimiento mejor que el primer rival, pero claramente inferior a nuestro diseño.

Finalmente es interesante destacar que ningún trabajo previo ha realizado una comparación rigurosa del rendimiento y del consumo de un diseño de este tipo implementado en una FPGA comparado con el obtenido con una aproximación basada en un ordenador de sobremesa convencional. En ese sentido consideramos que este trabajo puede resultar muy interesante para la comunidad científica, dado que la ejecución de juegos nunca se ha considerado como una de las posibles aplicaciones de las FPGAs, y sin embargo, tal y como se verá en los resultados experimentales, para este problema en concreto las FPGAs proporcionan un rendimiento y un consumo muy interesantes.

2. Inteligencia artificial implementada en el procesador

2.1 Consideraciones generales

La inteligencia artificial en los juegos de tablero se basa en tratar de explorar estados futuros y evaluarlos de acuerdo a su calidad (esto es, cuantificar lo ventajoso de un tablero para lograr un final ganador).

Este proceso involucra a dos elementos:

1) Algoritmo de búsqueda
2) Función de evaluación

El primero trata de explorar el espacio de estados futuros (árbol de juego) en base a ciertas consideraciones (ver sección 2.2).

El segundo es el estimador que utiliza el algoritmo de búsqueda para elegir cada movimiento (ver sección 2.3).

2.2 Algoritmo de búsqueda

El algoritmo elegido para explorar el árbol de juego es MinMax [8]. La idea subyacente es buscar nuestro mejor movimiento suponiendo que el rival escogerá el más ventajoso para él. El algoritmo realiza una búsqueda primero en profundidad, en la que se evalúan los nodos terminales (aquellos que son nodos fin de partida, o nodos cuya profundidad es la máxima para el árbol en construcción), y el valor se propaga a niveles superiores de manera que en los niveles pares se maximiza la utilidad (elegir el mayor valor de los nodos hijos), y en los niveles impares se minimiza la utilidad (elegir el menor valor de los nodos hijos).

Esta estrategia se topa con el problema de la complejidad computacional del Reversi: el árbol de juego completo consta de $\approx 10^{58}$ nodos. Es inviable por tanto explorarlo por completo. Esto obliga a limitar la búsqueda a cierta profundidad (ver sección 2.2.2). Por otro lado la metodología del algoritmo MinMax abre las puertas a una potente mejora, llamada poda alfa-beta, que evita explorar estados sin variar por ello el resultado de la búsqueda. La sección 2.2.1 la describe con mayor detalle.

2.2.1 Poda alfa-beta

Sea α el valor más alto encontrado hasta el momento en un nivel MIN sin cerrar, y sea β el valor más bajo encontrado hasta el momento en un nivel MAX sin cerrar. La poda alfa-beta [9] evitará explorar aquellos nodos de un nivel MIN que conduzcan a un valor menor o igual a α, y aquellos nodos de un nivel MAX que conduzcan a un valor mayor o igual a β.

La eficacia de la poda alfa-beta depende del orden en el que se generen los sucesores (cuanto antes se generen los mejores sucesores, mayor será la eficacia de la poda). En el mejor de los casos, se consigue una reducción del espacio de estados explorados de $O(b^d)$ a $O(b^{d/2})$, donde b es el factor de ramificación y d la profundidad del árbol. O lo que es lo mismo, se alcanza el doble de profundidad en el mismo tiempo.

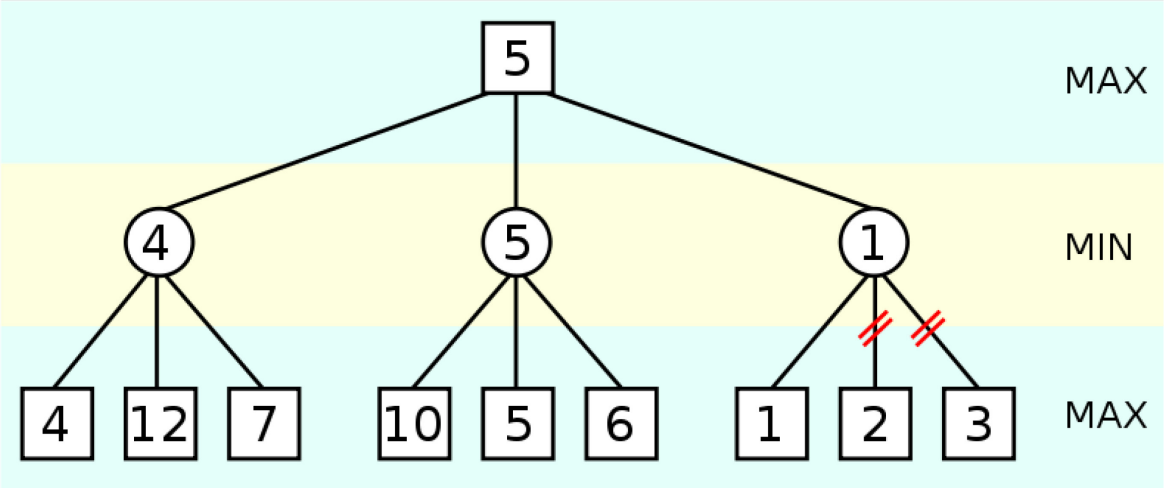


Fig. 3. Poda alfa-beta. Los nodos con marcas rojas no se explorarán

2.2.2 Búsqueda en profundidad iterativa

Tal y como se indica en las especificaciones del concurso, el tiempo máximo para procesar un movimiento es un segundo. Dado que no se conoce a priori el nivel máximo que puede alcanzar una búsqueda para un tiempo máximo prefijado, ya que depende del momento de la partida (esto es, del número de nodos a explorar. En las fases iniciales de la partida el número de sucesores es mayor que en las fases finales, ya que el número de movimientos legales al principio es mayor que al final), una buena aproximación para este problema

consiste en generar árboles de juego de profundidad incremental [10], y devolver como mejor movimiento el del árbol de mayor nivel máximo completamente generado en el momento en el que llegue el *timeout*.

Esta estrategia permite alcanzar la máxima profundidad de búsqueda que permitan las condiciones particulares en cada momento.
La aparente desventaja que supone invertir tiempo en la generación de árboles de profundidad máxima menor al más profundo generado finalmente, no lo es debido a que las búsquedas en los árboles menores sirven de guía a los de mayor profundidad, mejorando la eficacia de la poda alfa-beta.

2.2.3 Ordenación dinámica de nodos

La eficacia de la poda alfa-beta depende fuertemente de la ordenación de los nodos. Cuanto antes se generen los que a posteriori serán los mejores movimientos, mayor será la eficacia de la poda. En nuestro diseño llevamos a cabo dos tipos de ordenación:

a) Independiente del problema: aprovechamos el conocimiento que se infiere de la generación de árboles de profundidad incremental: el mejor movimiento hallado en el árbol limitado a profundidad *d*, será el primero en ser explorado en el árbol limitado a profundidad *d+1*.

b) Dependiente del problema: Puesto que los movimientos en casillas de las esquinas son a priori movimientos prometedores, mientras que movimientos en casillas X (aquellas adyacentes a las esquinas en las diagonales), son a priori malos movimientos, realizamos la siguiente ordenación de sucesores:

A1	H1	A8	H8	resto de casillas	B2	G2	B7	G7

Esto es, exploramos en primer lugar los movimientos en casillas de las esquinas, y en último lugar los movimientos en casillas de tipo X.

2.3 Función de evaluación

Las métricas (o heurísticas) utilizadas para evaluar la calidad de un tablero se definen a continuación. Distinguimos la evaluación de nodos en los que la partida no ha finalizado y aquellos en los que sí ha finalizado.

1) Evaluación de nodos intermedios:
   a) Movilidad: hace referencia a cuantos movimientos legales tiene cada jugador en un momento dado. Cuanto mayor sea la movilidad de un jugador, más probable es que pueda colocar en posiciones ventajosas. La estrategia básica de este juego consiste en obligar a tu rival a colocar una ficha dónde él no quiere. Esto se consigue reduciendo su movilidad, por ello esta métrica es tan importante.

   b) Esquinas y casillas de tipo X: Las esquinas son posiciones muy ventajosas, dado que no pueden ser volteadas, y permiten al jugador que las domine crear una región segura que el rival no puede atacar. Por otro lado, las casillas X son movimientos no deseables, ya que permiten que el rival consiga de forma sencilla la esquina adyacente.

   c) Casillas consolidadas: Son aquellas posiciones que no pueden ser volteadas por el rival. Una casilla está consolidada si sus vecinas inmediatas en al menos un sentido de cada una de las direcciones están consolidadas, dando lugar a la siguiente definición recursiva mutua:

      $C_{i,j}$ *sii* $(C_{i,j-1} \vee C_{i,j+1}) \wedge (C_{i-1,j} \vee C_{i+1,j}) \wedge (C_{i-1,j+1} \vee C_{i+1,j-1}) \wedge (C_{i-1,j-1} \vee C_{i+1,j+1})$

      Donde $C_{i,j}$ significa que la casilla de la fila *i* y columna *j* está consolidada.

2) Evaluación de nodos fin de partida:
   a) Número de fichas: el objetivo del juego es tener mayor número de fichas propias al finalizar la partida, por tanto en este caso tan solo es necesario contar las fichas de cada color.

Partiendo de estas métricas, realizamos un ajuste de los pesos en base a la relevancia de cada métrica, así como al *feedback* experimental, resultando la siguiente función de evaluación:

- Nodos intermedios:

$$\begin{aligned} \text{Non-terminalNode}_{ev} = {} & 4*(\text{Corners}_{FPGA} - \text{Corners}_{opponent}) + \\ & 2*(\text{XSquares}_{FPGA} - \text{XSquares}_{opponent}) + \\ & 2*(\text{Mobility}_{FPGA} - \text{Mobility}_{opponent}) + \\ & 1*(\text{Stables}_{FPGA} - \text{Stables}_{opponent}) \end{aligned}$$

- Nodos fin de partida:

$$\text{TerminalNode}_{ev} = \text{Discs}_{FPGA} - \text{Discs}_{opponent}$$

3 Implementación Hardware

La implementación del diseño se lleva a cabo mediante el lenguaje VHDL dentro de entorno Xilinx ISE 10.1.3 [11]. A continuación se muestra la arquitectura del procesador desarrollado y se describe en detalle la implementación de los distintos módulos que lo componen. Todos los módulos se han realizado desde cero con excepción del módulo E/S RS-232 que pudo ser parcialmente reutilizado de un diseño anterior.

3.1 Arquitectura del procesador

El procesador consta de un módulo de búsqueda de mejor movimiento en una tabla de aperturas (*Openings module*), un módulo de búsqueda de mejor movimiento mediante búsqueda en el árbol de juego MinMax (*MinMax move search*) y un módulo de entrada/salida para comunicarse con el oponente (E/S RS-232).
El módulo MinMax move search es el más complejo y se divide a su vez en varios submódulos:

1) Volteador (*Disc flipper*)
2) Movimientos posibles (*Move checker*)
3) Casillas consolidadas (*Stable discs evaluator*)
4) Evaluador (*Evaluator*)
5) Árbol (*Tree tables BRAMs* + *Tree data registers*)

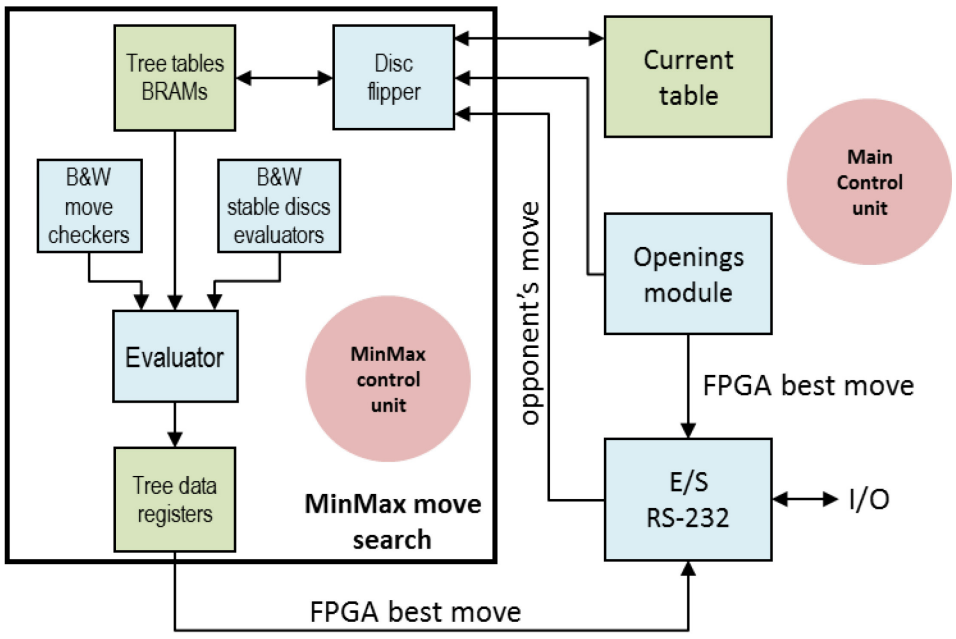


Fig. 4. Arquitectura del procesador

A continuación, se describen los módulos principales con mayor detalle.

3.1.1 Movimientos Posibles (*Move checker*)

Este módulo se encarga de hallar los movimientos legales dado un tablero y un color. Toma como entradas el tablero actual codificado en 128 bits, el turno actual codificado con 1 bit, y devuelve como salida un vector de movimientos legales de 64 bits. Se ha implementado de manera puramente combinacional, implementando una función lógica para cada casilla para averiguar si es un movimiento legal. Una casilla es un movimiento legal si cumple:

1) Está vacía
2) En al menos una dirección, a partir de la casilla que se está analizando existe el siguiente patrón: una o más fichas contiguas rivales seguidas de una ficha propia.

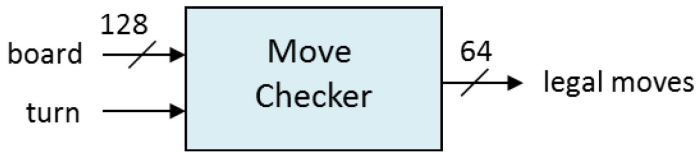


Fig. 5. Interfaz del módulo MoveChecker

Dado que en los nodos terminales se necesita conocer la movilidad de ambos jugadores, este módulo está replicado para poder calcular en paralelo los movimientos legales de cada jugador.

3.1.2 Volteador (*Disc flipper*)

Recibe como entradas el tablero actual, el turno, y el movimiento realizado, y calcula el nuevo tablero resultante. Este módulo se ha implementado mediante una red iterativa matricial bidireccional donde la información se mueve simultáneamente en 4 direcciones (y dos sentidos por dirección). Con esta implementación obtenemos un módulo puramente combinacional capaz de calcular un nuevo tablero en un solo ciclo de reloj. La ventaja de diseñar una solución basada en redes iterativas es que nos permite solucionar un problema muy complejo, casi imposible de formular de forma directa, dividiéndolo en subproblemas mucho más sencillos (en este caso cada casilla es un subproblema). Otra ventaja adicional, a la que aquí no se ha sacado partido, es

que este diseño se puede aplicar de forma directa a cualquier tamaño de tablero, dado que es tan sencillo como instanciar el número de módulos necesario y conectarlos convenientemente.

La figura 7 muestra la arquitectura de una celda de la red. Para cada casilla, se calcula en cada dirección el nuevo patrón a propagar en función del patrón recibido y del contenido de la casilla (ver tabla 1). Así mismo, en caso de que se haya encontrado un patrón que implique una operación de volteo, será la casilla donde se ha colocado ficha la que dará la orden de que se realicen los volteos correspondientes. Para ello analizará todos los patrones que le lleguen, y propagará una señal de *flip* en aquellas direcciones que haya que voltear. El valor de esta señal será igual al número de fichas que hay que voltear en dicha dirección (ver tabla 2). Este valor se decrementará en cada propagación hasta llegar a cero. La figura 6 ilustra con un ejemplo el funcionamiento de la red.

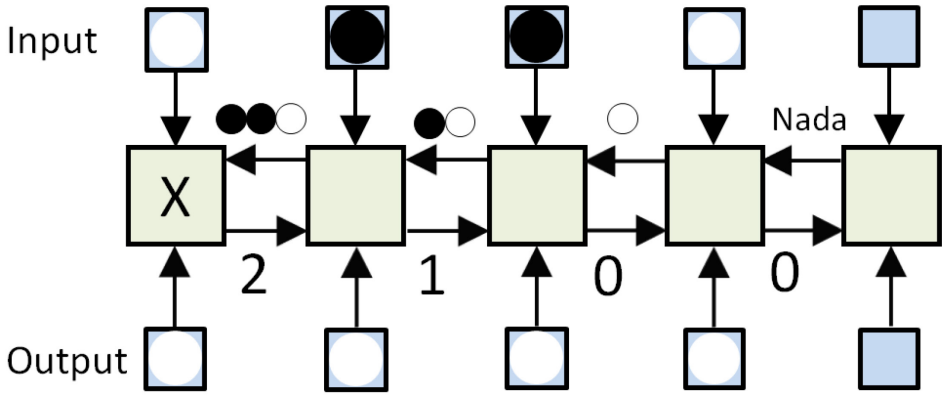


Fig. 6. Ejemplo de funcionamiento de la red iterativa del Disc Flipper en uno de las 8 direcciones de flujo de información. La casilla marcada con una 'X' es aquella donde colocamos la ficha

Patrón recibido	Contenido casilla	Patrón propagado
Nada	ó	Nada
		Nada
		Nada
		Nada
		Nada
		Nada
	ó	Nada

Tabla 1. Propagación de patrones para el volteo de fichas moviendo negras. Para blancas es equivalente.

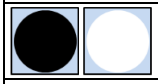
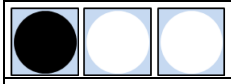
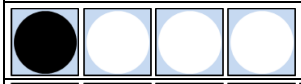
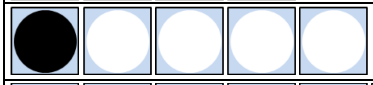
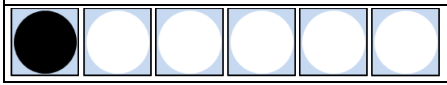
Patrón recibido	Salida Flip
	1
	2
	3
	4
	5

Tabla 2. Propagación de señal de flip a partir de la casilla donde se ha colocado ficha. Para blancas es equivalente.

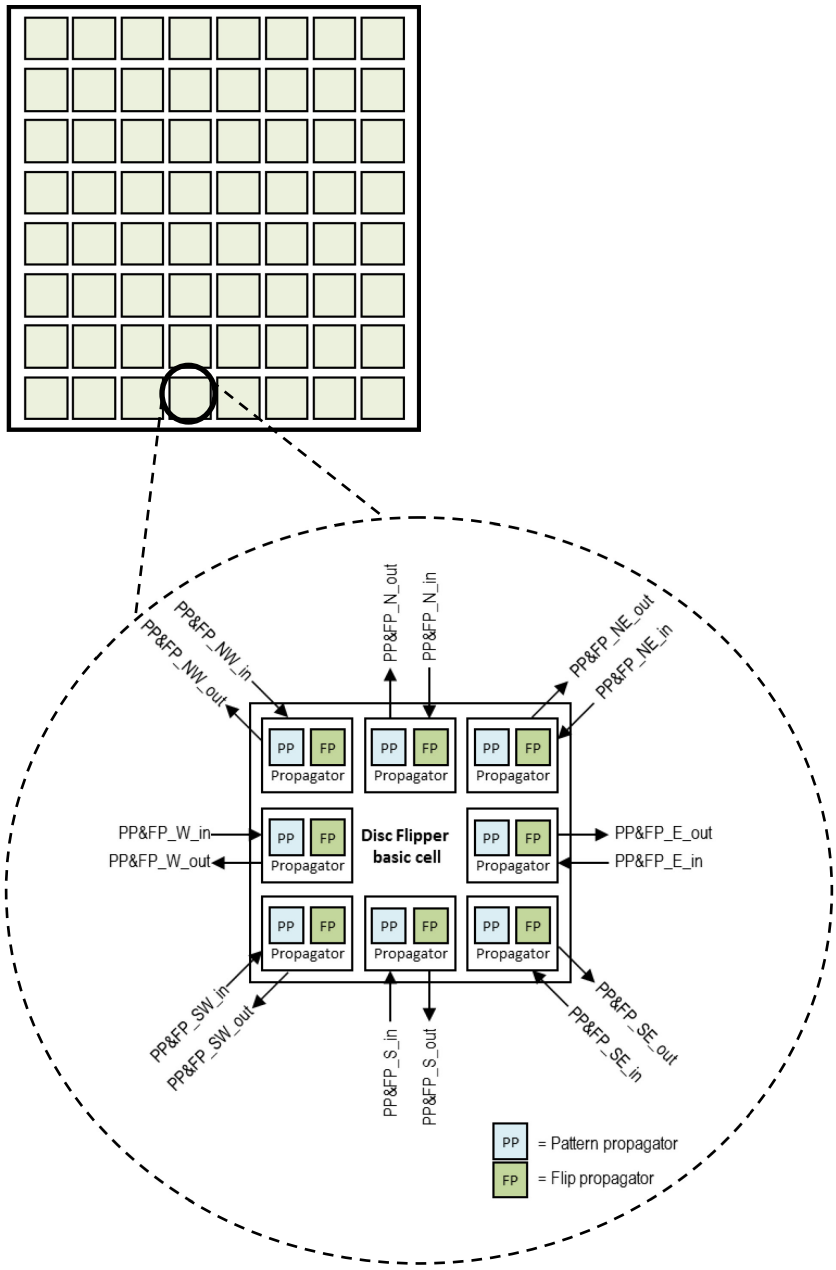


Fig. 7. Arquitectura e interfaz de una celda de la red iterativa del Disc Flipper.

3.1.3 Casillas consolidadas (*Stable discs evaluator*)

Como se ha indicado en la sección 2.3.c, el cálculo de las casillas consolidadas se realiza mediante una función recursiva mutua. Una implementación puramente combinacional deriva por tanto en bucles

combinacionales que conllevan una reducción drástica de la frecuencia máxima del procesador. Por este motivo, este módulo ha sido implementado mediante una red iterativa matricial bidireccional secuencializada mediante la inclusión de un biestable en cada celda de la red (salvo en las casillas del contorno).

Esta red, al igual que el volteador, trabaja simultáneamente en las 4 direcciones, y en los dos sentidos de cada dirección. Las casillas correspondientes al contorno se calculan combinacionalmente mediante funciones lógicas. El resto del tablero se secuencializa por capas tal y como muestra la figura 8, la información generada en cada capa se almacena en un conjunto de biestables. El módulo resultante calcula las casillas en un número de ciclos variable, y por ello el módulo posee una salida *done*, tal y como se puede ver en la figura 9. Esta señal se activará cuando ningún biestable cambie con respecto al ciclo anterior. Dado que esta operación es algo más costosa que el resto de operaciones para evaluar un nodo terminal, se añadió una sencilla lógica adicional que comprueba si todas las esquinas están vacías, en cuyo caso no es posible que haya casillas consolidadas, y por tanto se omite el cálculo de las mismas.

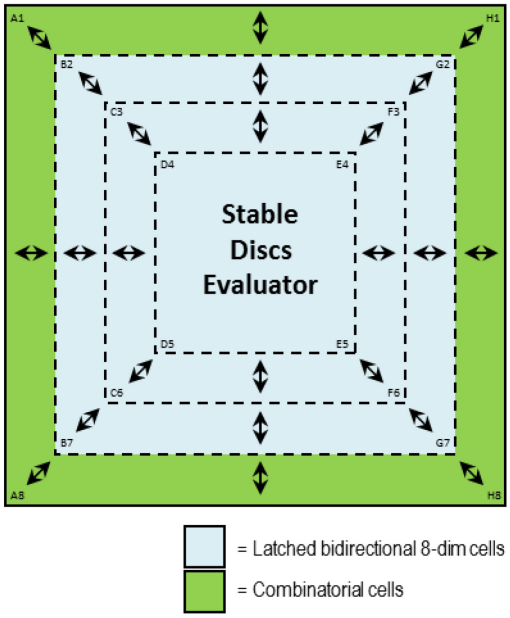


Fig. 8. Arquitectura del módulo de cálculo de las casillas estables.

El módulo contiene además un sumador en árbol de 64 bits que devuelve el número de casillas consolidadas. La interfaz del *Stable Discs Evaluator* se muestra en la figura 9.

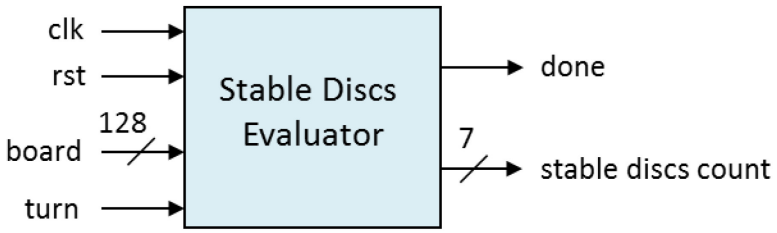


Fig. 9. Interfaz del módulo Stable Discs Evaluator

Dado que en los nodos terminales se necesita conocer el número de casillas estables de ambos jugadores, este módulo está replicado para poder calcular ambos en paralelo.

3.1.4 Tabla de aperturas (*Openings table*)

En los juegos de tablero es habitual guiar el juego en su fase inicial por secuencias de movimientos que, si bien pueden no ser óptimos, sí sabemos que son buenos movimientos en base a la experiencia y al conocimiento de los conceptos referentes a la estrategia del juego.

En el procesador se hizo uso de una tabla de 51 aperturas. Dada la simetría horizontal y vertical del tablero, se calcularon las correspondientes aperturas simétricas en cada dirección, resultando un total de 204 aperturas.

Para la confección del módulo de gestión de las aperturas se han utilizado 4 BRAMs de doble puerto y ancho del bus de datos de 9 bits. En la figura 10 se muestra la arquitectura del módulo.

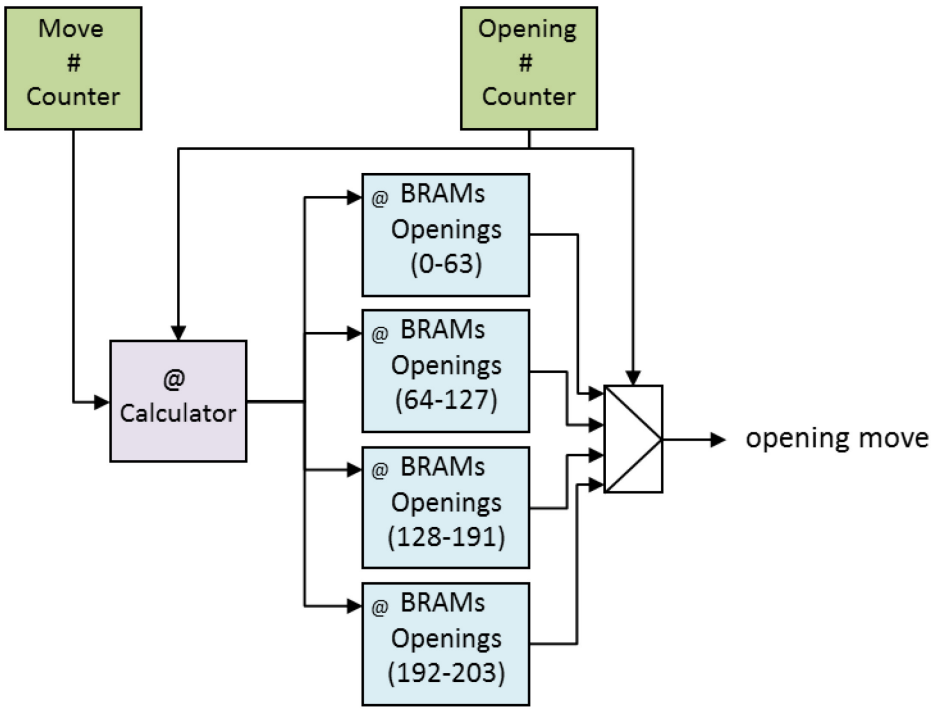


Fig. 10. Arquitectura del módulo de gestión de aperturas

Al comenzar el juego se sigue siempre la primera apertura encontrada que se adapte a la situación actual. Cuando la situación de la partida no se adapte a ninguna de las aperturas este módulo dejará de utilizarse. Por lo general, esto ocurre tras 4 o 5 movimientos.

3.1.5 Árbol

La implementación hardware del árbol de juego consta de los siguientes submódulos:

a) Almacenes del árbol
b) Máquina de estados
c) Módulo de ordenación dinámica de nodos

El módulo del árbol de juego ha sido dimensionado para alcanzar una profundidad máxima de 16 niveles. A continuación se describen en detalle los submódulos:

a) Almacenes del árbol

- Es necesario almacenar el tablero abierto por cada nivel del árbol. Cada tablero se codifica con 128 bits (2 bits por casilla). Esto

requiere un total de 128 bits/tablero x 16 niveles x 1 tablero/nivel = 2048 bits.
Para que la lectura de un tablero se realice en un ciclo de reloj, se han utilizado 4 BRAMs de 36 bits de ancho en paralelo, resultando una memoria con un ancho de datos de 144 bits.

- La información relativa al algoritmo MinMax con poda alfa-beta se almacena en un banco de registros diseñado a medida que consta de lo siguiente:
    1) 16 registros de 6 bits que almacenan el último movimiento explorado en cada nivel abierto
    2) 16 registros de 9 bits que almacenan los valores α - β de cada nivel abierto
    3) Un registro de 6 bits que almacena el mejor movimiento que ha encontrado el algoritmo para el nivel de profundidad máximo actual
    4) Un registro de 6 bits que almacena el mejor movimiento que ha encontrado el algoritmo en el árbol de mayor profundidad completado

b) Máquina de estados

La generación de nodos se ha dividido en dos etapas para alcanzar una mayor frecuencia de trabajo. La primera etapa se corresponde con el estado “Estado Genera Nodo 1”, y en ella se calculan los movimientos legales y se elige el siguiente movimiento legal a explorar. La segunda etapa se corresponde con el estado “Estado Genera Nodo 2”, y en ella se genera el nuevo tablero correspondiente a realizar el movimiento legal elegido en la etapa previa y voltear las fichas que proceda.

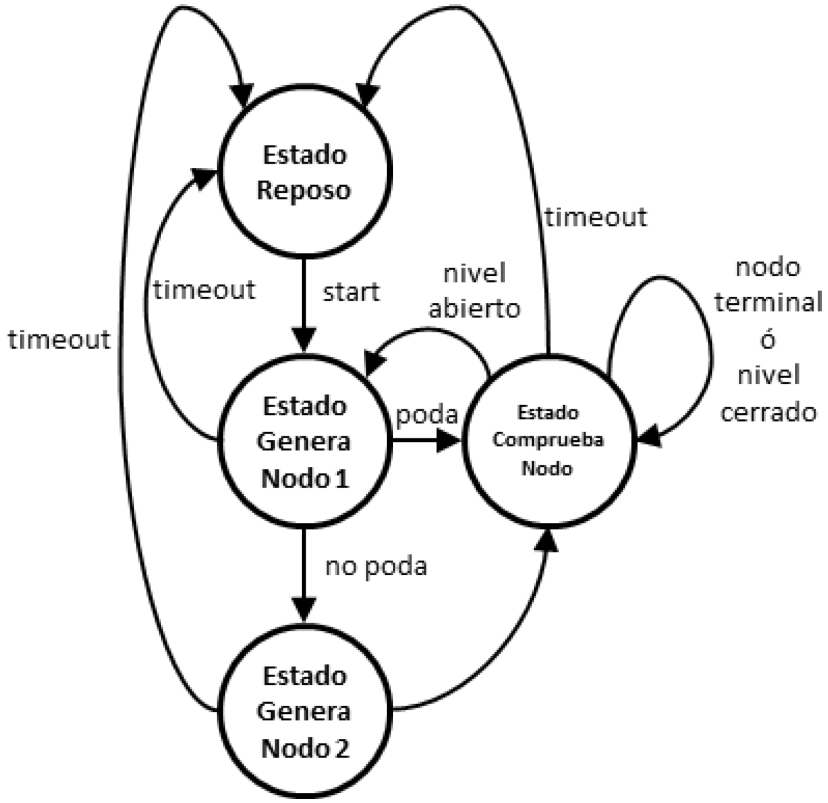


Fig. 11. Máquina de estados del módulo Árbol

c) Módulo de ordenación dinámica de nodos

Aplicamos una modificación a la salida del módulo *Movimientos Posibles* para reordenar los movimientos según lo descrito en la sección 2.2.3, esta reordenación la realizan los módulos de la figura 12, *Mapping* y *Last move mapping*. Una vez reordenados los movimientos posibles, seleccionamos el siguiente movimiento a explorar. Para ello se utiliza un codificador con prioridad enmascarado. La máscara se aplica en función del último movimiento explorado. De esta forma la salida de este módulo es el movimiento más prioritario según nuestra reordenación que todavía no hemos explorado. Finalmente un mapeado inverso traduce el movimiento elegido en el vector reordenado al correspondiente movimiento en su orden natural.

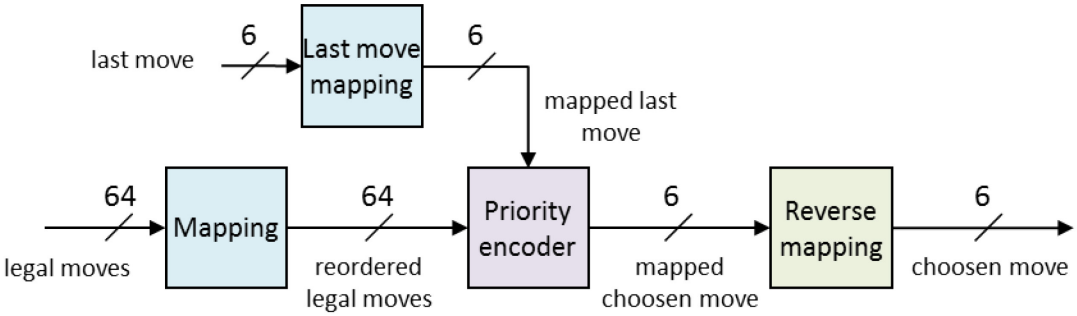


Fig. 12. Arquitectura del módulo de ordenación dinámica de nodos

3.2 Recursos utilizados

El diseño ha sido implementado sobre una Virtex II-Pro (XC2VP30-FF896). Este modelo dispone de un total de 13.696 slices y 2.448 Kb de BRAM. La tabla 3 muestra los recursos utilizados por la implementación del procesador. Como se puede ver utilizamos menos de la mitad de los recursos de la FPGA, a pesar de que es una FPGA relativamente antigua (de hace unos 10 años), mucho más pequeña que las FPGAs de altas prestaciones actuales. Por tanto nuestro diseño tiene un coste hardware muy razonable.

La frecuencia de trabajo del procesador es de 32 MHz. Esta FPGA está pensada para trabajar a 100 MHz, sin embargo en este diseño hemos apostado por hacer redes iterativas capaces de hacer operaciones muy complejas en tan sólo un ciclo de reloj. Estas redes son las que limitan la frecuencia de reloj del sistema. Sin embargo también son las que nos proporcionan nuestro buen rendimiento, por lo que, tal y como se verá más adelante, la perdida de frecuencia de reloj es completamente aceptable.

Versión	BRAMs	Slices
Finalistas	4 (2%)	5.325 (38%)
Ganadores	8 (5%)	5.830 (42%)

Tabla 3. Recursos de la FPGA utilizados. La versión *Finalistas* es la que nos clasificó para la final. La versión Ganadores fue la presentada en la final del concurso

4 Resultados

Para evaluar nuestro diseño se han realizado diversas mediciones y comparaciones estudiando tanto su rendimiento, su consumo de energía, y el tiempo de desarrollo.

4.1 Rendimiento

El rendimiento se ha comparado con cuatro rivales:

1) SW de referencia proporcionado por el concurso
2) Diseños realizados por los otros dos equipos de finalistas
3) SW equivalente que aplica exactamente las mismas técnicas
4) SW profesional de Reversi WZebra, considerado uno de los mejores SW de Reversi existentes y desarrollado y entrenado por maestros del juego

4.1.1 Versus SW de referencia

La organización del concurso proporcionó un software contra el cual testar y evaluar los diseños participantes. Este software tiene las siguientes características:

- Algoritmo de búsqueda MinMax con poda alfa-beta
- Evaluación de nodos terminales basada en número de fichas y puntuaciones estáticas de las casillas del tablero
- 7 niveles de dificultad, que se corresponden con la profundidad alcanzada en el árbol de juego
- Tiempo de cómputo variable: el necesario para construir y evaluar el árbol correspondiente al nivel de dificultad elegido

Los resultados de la confrontación del diseño implementado en la FPGA con este software se muestran en la tabla 4. El software fue ejecutado en un PC equipado con un procesador Intel i7-2600 @ 3,4GHz y 8 GB de memoria RAM.

Nivel de dificultad SW	FPGA mueve negras			SW mueve negras		
	Puntuación partida (HW-SW)	Movimientos		Puntuación partida (HW-SW)	Movimientos	
		HW	SW		HW	SW
1	63-1	32+0	31+3	63-0	31+0	31+3
2	64-0	33+0	32+5	60-4	31+0	31+2
3	64-0	31+0	30+1	64-0	33+0	33+6
4	64-0	31+0	30+1	63-0	31+0	31+3
5	64-0	33+0	32+5	64-0	30+0	30+0
6	64-0	31+0	30+1	63-0	31+0	31+3
7	64-0	33+0	32+5	64-0	32+0	32+4

Tabla 4. Resultados de la confrontación SW de referencia vs FPGA con tiempo de cómputo limitado a 1 segundo

Se aprecia una gran superioridad del jugador artificial implementado sobre la FPGA sobre el SW de referencia.

Adicionalmente, testeamos nuestro diseño de manera más agresiva limitándole el tiempo de cómputo para ver qué resultados obteníamos contra el SW de referencia. La tabla 5 muestra los resultados para un tiempo de cómputo limitado a 0,1 segundos, y la tabla 6 para un tiempo de cómputo limitado a 3,2 * 10^{-5} segundos (1024 ciclos de reloj).

Nivel de dificultad SW	FPGA mueve negras	SW mueve negras
	Puntuación partida (HW-SW)	Puntuación partida (HW-SW)
1	57-7	64-0
2	64-0	60-0
3	64-0	64-0
4	64-0	59-4
5	64-0	61-1
6	64-0	59-4
7	60-3	59-5

Tabla 5. Resultados de la confrontación SW de referencia vs FPGA con tiempo de cómputo limitado a 0,1 segundos

Nivel de dificultad SW	FPGA mueve negras	SW mueve negras
	Puntuación partida (HW-SW)	Puntuación partida (HW-SW)
1	42-22	51-3
2	52-12	44-20
3	42-22	54-10
4	60-0	38-26
5	63-0	47-17
6	63-1	38-26
7	63-0	*

* El SW realizó un movimiento inválido

Tabla 6. Resultados de la confrontación SW de referencia vs FPGA con tiempo de cómputo limitado a $3{,}2 * 10^{-5}$ segundos

A la vista de los resultados de las tablas 5 y 6, se puede apreciar cómo incluso para un tiempo de cómputo realmente restrictivo ($3{,}2 * 10^{-5}$ segundos), nuestro diseño es capaz de ganar en todos los casos.

4.1.2 Versus diseños finalistas

En la final disputada en una sesión del congreso, nuestro diseño se enfrentó a los otros dos diseños finalistas (FPGA vs FPGA) en formato de competición round-robin. Jugamos un total de cuatro partidas, dos contra un equipo de una universidad griega, y dos contra un equipo de una universidad japonesa, resultando ganadores en todas ellas, por lo que obtuvimos el primer premio del concurso.

4.1.3 Versus SW equivalente

Desarrollamos una solución software algorítmicamente equivalente al diseño hardware. Ha sido desarrollada en C, y consta de 2 hilos de ejecución, uno para el programa principal y otro para la gestión del *timeout*.

FPGA mueve negras			SW mueve negras		
Puntuación partida (HW-SW)	Nivel máximo promedio		Puntuación partida (HW-SW)	Nivel máximo promedio	
	HW	SW		HW	SW
42-22	9,64	7,63	52-12	9,85	8,16

Tabla 7. Resultados de la confrontación SW equivalente vs FPGA.

Los resultados de la confrontación del diseño implementado en la FPGA con este software se muestran en la tabla 7. El software fue ejecutado en un PC equipado con un procesador Intel i7-2600 @ 3,4GHz y 8 GB de memoria RAM.

La FPGA gana al SW, tal y como se puede apreciar, debido a que es capaz de alcanzar mayor profundidad en el árbol de juego para un mismo tiempo de cómputo.

Es importante mencionar que durante la realización de este software se aplicaron los conocimientos adquiridos en la asignatura del máster "Programación orientada a prestaciones". Para ello, la versión SW inicial fue analizada con el programa de *profiling* de Intel, "*VTune"*, en busca de los *hotspots.*
En este análisis se constató que la función que haya los movimientos legales (*legalMoves*) es el principal *hotspot* de la aplicación, suponiendo un 81% del total de tiempo de cómputo de movimientos.
En base a este resultado, se optimizó dicha función añadiendo una *cache* de movimientos legales, que evita recalcularlos para cada movimiento de un nodo abierto. La aplicación se compiló con la versión 4.1.2 del compilador GCC, aplicando el flag "-O3". También se intentó la paralelización automática, si bien el compilador no fue capaz de sacar partido a esta posibilidad.
En todo caso, estamos en proceso de paralelizar manualmente la función legalMoves mediante directivas OpenMP, si bien en el momento de redacción de este documento no se ha concluido la tarea.

4.1.4 Versus SW profesional (WZebra)

Consideramos interesante evaluar a nuestro diseño contra un software profesional de Reversi. Elegimos el WZebra [12], dado que es gratuito y está considerado como uno de los más potentes creados hasta el momento.
WZebra implementa MultiProb-Cut, un sofisticado algoritmo de búsqueda superior a MinMax, una extensa tabla de aperturas, capaz de generar nuevas variantes en base al conocimiento adquirido en las partidas que disputa, y una función de evaluación basada en inspección de patrones de distintas zonas del tablero.

Todo esto le convierte en un rival extremadamente fuerte, y como tal, es superior a nuestro diseño, si bien para profundidades de búsqueda bajas (menor de 4 niveles), nuestro diseño gana casi todas las partidas, y para niveles medios (4-6) de WZebra, nuestro diseño es capaz de batirle en ciertas ocasiones. WZebra exhibe un comportamiento no determinista, y de un total de 22 partidas disputadas contra él, nuestro diseño fue capaz de batirle en 9 ocasiones (≈40%). A partir del nivel 7 WZebra ganó a nuestro diseño todas las partidas que disputamos.

4.2 Consumo energético

Actualmente, el problema de la disipación de calor es un factor crítico en el diseño de procesadores. La capacidad de cálculo de muchos sistemas se ve necesariamente coartada por este factor, especialmente en los sistemas empotrados.

Para la evaluación utilizamos como instrumento de medida un vatímetro Yokogawa WT210 [13]. El esquema de medida del consumo de la FPGA y del PC se muestra en la figura 13.

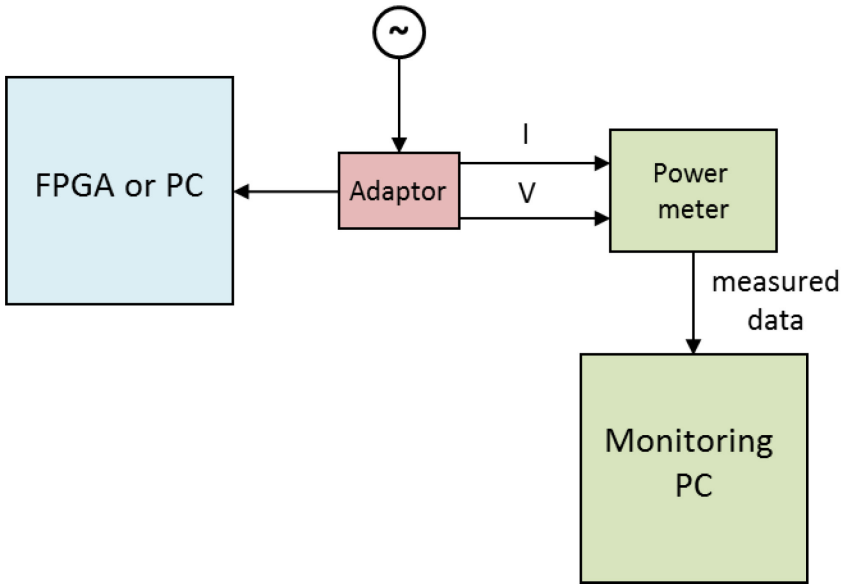


Figura 13. Montaje para la medida de consumo mediante un vatímetro Yokogawa WT210

El vatímetro viene acompañado de un software que procesa el muestreo de las medidas y ofrece gráficas de una gran variedad de magnitudes de medida. En nuestro caso, nos interesa mostrar la potencia instantánea en función del tiempo. La figura 14 muestra, para una partida FPGA vs SW equivalente, la correspondiente gráfica de consumo para el PC, y la figura 15 para el consumo de la FPGA. Los picos que se observan en las gráficas se corresponden a los intervalos de procesamiento/espera propios del juego dado que tanto el PC como la FPGA alternan periodos de actividad, en los que tienen que decidir qué movimiento quieren realizar, con periodos de inactividad, en los que están esperando a que el rival mueva. En el caso del PC, realiza una espera activa con lo que sigue ejecutando instrucciones y disipando potencia.

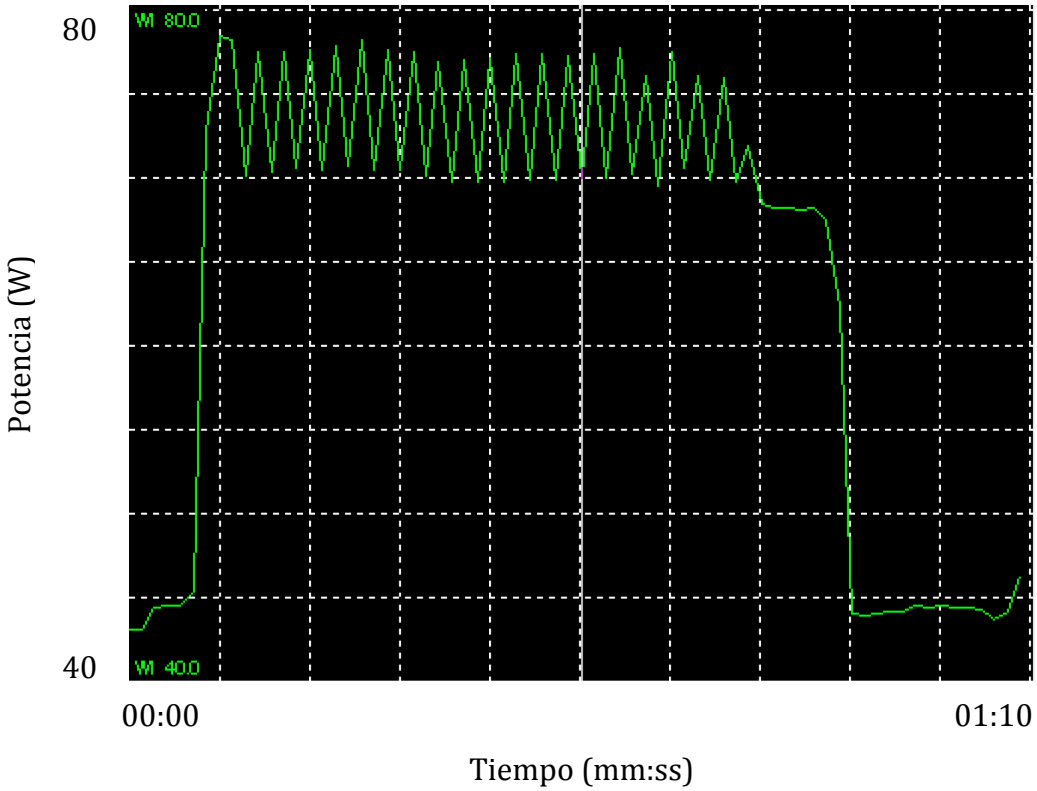


Figura 14. Consumo de potencia del PC corriendo el SW equivalente durante el trascurso de una partida de Reversi

En la figura 14 se puede ver cómo el consumo del PC en reposo ronda los 45 W. Durante la ejecución del SW equivalente, se distinguen los tramos de tiempo en los que el SW espera el movimiento del rival, con un consumo en torno a los 70 W, y los tramos en los que el SW está calculando su próximo movimiento, lo cual eleve el consumo a cerca de 80 W. El promedio de potencia durante la ejecución del SW es de casi 75 W.

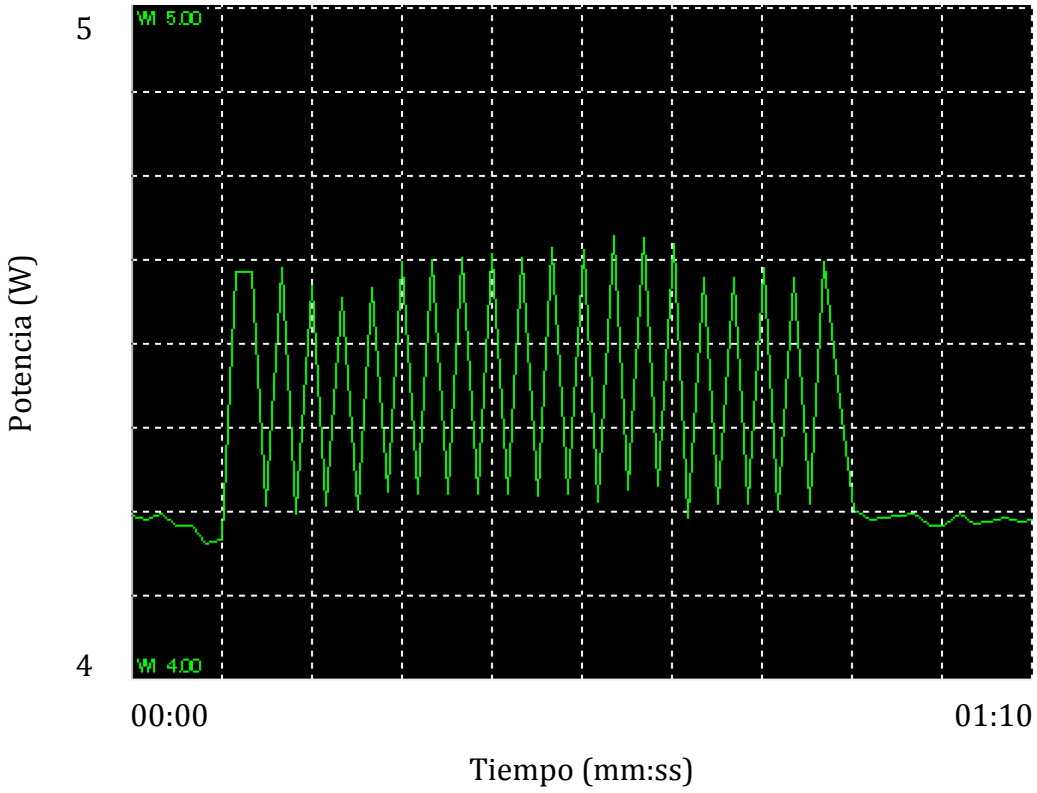


Figura 15. Consumo de potencia de la FPGA durante el trascurso de una partida de Reversi

En la figura 15 se aprecia cómo el consumo de la FPGA en reposo ronda los 4,25 W. Durante el transcurso de la partida, la FPGA consume en torno a 4,6 W en los tramos donde calcula su próximo movimiento, y en torno a 4,3 W en los tramos donde espera el movimiento del rival.

El promedio de potencia durante la partida se sitúa en unos 4,45 W, unas 17 veces menos que el PC.

4.3 Tiempo de desarrollo

Como se verá en la sección 7, el tiempo de desarrollo del diseño HW ha sido sustancialmente superior al del diseño SW. Esto es debido principalmente a dos motivos:

- El tiempo de diseño HW incluyó también la afinación de los pesos de la función de evaluación, para obtener mejores resultados contra el SW de referencia
- El tiempo de desarrollo de una solución HW es superior al del desarrollo de una solución equivalente SW, debido a que, si bien el

diseño y la implementación puedan suponer tiempos relativamente similares, la depuración de desarrollos HW es mucho más laboriosa que a la de desarrollos SW, teniendo un gran impacto en el tiempo total necesario para su desarrollo.

5. Conclusiones

Los distintos resultados obtenidos y la experiencia adquirida durante el desarrollo del diseño expuesto en este trabajo permiten extraer las siguientes conclusiones:

Alto rendimiento del diseño hardware:

Los resultados expuestos en la sección 4.1.3 muestran cómo el diseño HW es capaz de procesar un mayor número de nodos por unidad de tiempo que un SW equivalente que aplica los mismos algoritmos y las mismas optimizaciones. En promedio, el diseño HW es capaz de alcanzar casi 2 niveles más de profundidad en el árbol de juego. Esto supone una gran diferencia, ya que el número de nodos a explorar crece exponencialmente con la profundidad alcanzada.

El principal motivo de esta sustancial diferencia se encuentra en el paralelismo. El cálculo de los movimientos legales de un tablero posee un alto grado de paralelismo, ya que es posible calcular en paralelo cada casilla del tablero.

El *Move Checker* del diseño HW en efecto calcula en paralelo todas las casillas del tablero. No así la solución SW, que ejecuta el mismo algoritmo pero de manera secuencial, casilla a casilla.

Es importante señalar además que estamos enfrentando a tecnologías con un importante desfase temporal: La FPGA sobre la que ha sido implementado el diseño data de 2002, mientras que el procesador sobre el cual corre la solución SW equivalente data de 2011. Por tanto el procesador que se ha desarrollado en este proyecto implementado en una FPGA que ya está casi obsoleta trabajando a 32 MHz es capaz de proporcionar un rendimiento muy superior que un PC actual. La razón es que este diseño HW ha sido optimizado para sacar el máximo partido al paralelismo de los distintos problemas a tratar.

Bajo consumo del diseño hardware:

El incremento de consumo de potencia del PC ejecutando la solución SW equivalente es dos órdenes de magnitud superior al correspondiente incremento de consumo de la FPGA. Dado que ambas trabajan el mismo tiempo, el consumo de energía también seria dos órdenes mayor para el PC. Pero en realidad, si

realizase el mismo trabajo (explorar el mismo número de nodos), el consumo de energía sería todavía mayor en el PC dado que el PC en el mismo tiempo explora menos niveles.

Inteligencia artificial en el Reversi:

El Reversi actualmente es un juego no resuelto (en tamaño 8x8). Por lo tanto, es necesario utilizar heurísticas que evalúen con la mayor fidelidad posible la calidad de los tableros.

El software proporcionado por el concurso implementa una primera aproximación consiste en evaluar a partir de valores estáticos para las casillas y del número de fichas.

Nuestro diseño implementa un segundo paradigma basado en conceptos tales como la movilidad, casillas estables y captura de las esquinas.

Finalmente, existe un tercer paradigma fundamentado en la inspección de patrones en el tablero. Programas profesionales de Reversi tales como WZebra, implementan este paradigma.

Los distintos enfrentamientos entre los 3 paradigmas evidencian que la estrategia de evaluación implementada en el diseño HW es claramente superior a la implementada en el software proporcionado por el concurso, pero se ve ampliamente superada por la inspección de patrones implementada en WZebra.

6. Trabajo futuro

Los objetivos iniciales del proyecto se han alcanzado en su totalidad por lo que podría parecer que este es un trabajo cerrado. Sin embargo quedan algunas tareas interesantes por realizar.

En primer lugar queremos buscar la forma de realizar un análisis de consumo más detallado, siendo capaces de ver no sólo el consumo total sino también el consumo interno de cada parte de nuestro procesador, así como de la versión SW equivalente. De esta forma podremos identificar que partes de nuestro procesador son más eficientes.

Por otro lado hemos seguido trabajando en el diseño de otro procesador para otro juego (en este caso el Connect-6 [14]), y planeamos hacer un análisis conjunto de los resultados y enviarlo a una revista científica. Consideramos que este análisis resultaría muy interesante para la comunidad científica dado que muy pocas veces se han realizado comparaciones HW/SW rigurosas y justificadas que incluyan el

consumo de potencia y energía. Además, la utilización de FPGAs para acelerar la ejecución de juegos de tablero es un campo prácticamente inexplorado.

Por último, para que nuestro análisis sea más interesante planeamos ampliarlo a otras plataformas. En concreto nos gustaría incluir plataformas para sistemas móviles, y plataformas tipo GPGPU (General-Purpose Computing on Graphics Processing Units).

7. Planificación

Comenzamos el trabajo el 25 de Junio de 2010, y el *deadline* del concurso, 15 de Octubre de 2010, limitaba el tiempo del que inicialmente disponíamos.

En este espacio de tiempo desarrollamos el diseño HW que evaluamos contra el SW de referencia, descrito en [15].

Posteriormente, la organización habilitó a los finalistas a mejorar sus diseños para la final que se celebró el 10 de Diciembre de 2010. En este periodo incorporamos dos optimizaciones adicionales al diseño HW: tabla de aperturas y ordenación dinámica de nodos independiente del problema.

Más adelante, del 5 de Junio de 2011 al 11 de Agosto de 2011, desarrollamos la solución SW equivalente e instrumentada, y realizamos el *profiling* y la optimización de la misma.

Finalmente, en el mes de Octubre de 2011, se puso en marcha la plataforma de medición de consumo de potencia, y se realizaron las medidas oportunas.

El total de horas empleadas en el trabajo es de 810. La figura 16 muestra la distribución de las mismas.

Es importante mencionar que la tarea “Diseño e implementación HW” incluye el diseño y ajuste de la inteligencia artificial incorporada al diseño, facilitando de manera importante el desarrollo de la versión SW.

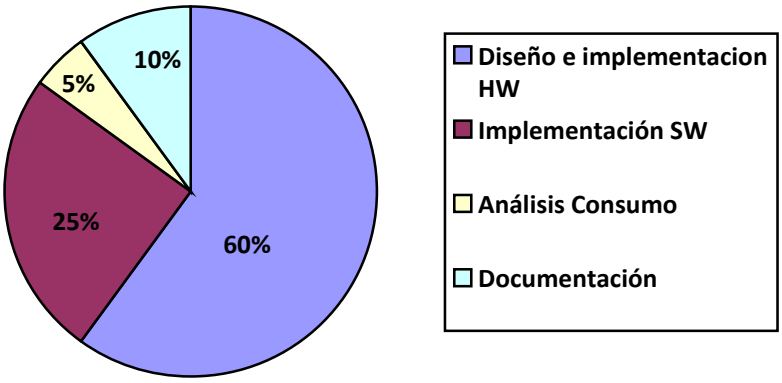


Figura 16. Distribución del tiempo invertido en cada tarea del trabajo

8. Referencias

[1] Brian Rose (2005) Othello: A Minute to Learn… A Lifetime to Master. http://othellogateway.com/

[2] M. Buro (1999) How Machines have Learned to Play Othello, IEEE Intelligent Systems J. Vol 14(6), pag: 12-14

[3] M. Buro (2000) Experiments with Multi-ProbCut and a New High-Quality Evaluation Function for Othello, *Games in AI Research*, ISBN: 90-621-6416-1

[4] M. Buro (2003) The Evolution of Strong Othello Programs. *Entertainment Computing - Technology and Applications*, Kluwer, pag. 81-88

[5] Wong, C.K.; Lo, K.K.; Leong, P.H.W. (2004) An FPGA-based Othello Endgame Solver. *Proceedings IEEE International Conference on Field-Progammable Technology (FPT 2004)*, pag: 81-88.

[6] Mabuchi, T.; Watanabe, T.; Moriwaki, R.; Aoyama, Y.; Gundjalam, A.; Yamaji, Y.; Nakada, H.; Watanabe, M. (2010) Othello Solver based on a soft-core MIMD processor array. *Proceedings IEEE International Conference on Field-Progammable Technology (FPT 2010)*, pag: 511-514.

[7] Smerdis, M.; Malakonakis, P.; Dollas, A. (2010) CarlOthello : An FPGA-Based Monte Carlo Othello player. *Proceedings IEEE International Conference on Field-Progammable Technology (FPT 2010)*, pag: 515-518.

[8] Minimax, *Wikipedia: The Free Encyclopedia,*
http://en.wikipedia.org/wiki/Minimax

[9] Alpha-Beta pruning, *Wikipedia: The Free Encyclopedia,*
http://en.wikipedia.org/wiki/Alpha-beta_pruning

[10] Iterative deepening depth-first search, *Wikipedia: The Free Encyclopedia*,
http://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search

[11] Xilinx Inc., http://www.xilinx.com

[12] Gunnar Andersson Othello Website, http://radagast.se/othello/

[13] Yokogawa Electric Corporation Website, http://www.yokogawa.com/

[14] Connect-6, *Wikipedia: The Free Encyclopedia,*
http://en.wikipedia.org/wiki/Connect6

[15] Olivito, J.; González, C.; Resano, J. (2010) FPGA Implementation of a Strong Reversi Player. *Proceedings IEEE International Conference on Field-Progammable Technology (FPT 2010)*, pag: 507-510.