IDEAS EN CIENCIAS DE LA INGENIERÍA

Implementación de Autómatas Celulares en una Tarjeta FPGA

Implementation of Cellular Automata on an FPGA Card

González-Contreras, J.[1], Jiménez-López, E.[2]*, Lozoya-Ponce, R.E.[1], Villagrán-Ruiz, J.C.[1]
(*ejimenezl@uaemex.mx)

Recibido: 15-11-2024 Aceptado: 29-01-2025

Resumen

El objetivo de este trabajo es utilizar el modelo de autómata celular en una aplicación específica en una tarjeta FPGA para explorar y visualizar comportamientos complejos. Los autómatas celulares son modelos matemáticos que simulan el comportamiento de sistemas dinámicos mediante reglas locales aplicadas en un espacio geométrico específico. Pueden generar patrones complejos a partir de interacciones simples, lo que los hace útiles para modelar fenómenos naturales y sociales. Una de sus principales ventajas es su capacidad de procesar datos en paralelo, reduciendo el tiempo de simulación en comparación con los enfoques secuenciales. Esto permite simular comportamientos complejos en lapsos de tiempo muy pequeños. Estudiar sistemas dinámicos autónomos en un hardware reconfigurable ofrece una gran cantidad de aplicaciones prácticas en la ingeniería computacional y de software. La arquitectura paralela y reconfigurable de los FPGA's es ideal para explorar aplicaciones avanzadas de autómatas celulares con modificaciones en diseño y reglas de transición, que puede abordar problemas en áreas como criptografía y simulación de sistemas distribuidos.

Palabras clave: Autómatas Celulares, FPGA, Reglas de transición, Programación en Paralelo.

Abstract

The objective of this work is to use the cellular automaton model in a specific application on an FPGA board to explore and visualize complex behaviors. Cellular automata are mathematical models that simulate the behavior of dynamic systems using local rules applied in a specific geometric space. They can generate complex patterns from simple interactions, which makes them useful for modeling natural and social phenomena. One of their main advantages is their ability to process data in parallel, reducing simulation time compared to sequential approaches. This allows simula-

[1]Tecnológico Nacional de México, Instituto Tecnológico de Chihuahua, Ave. Tecnológico 2009, Chihuahua, México, CP 31200.
[2]Centro de Investigación y Estudios Avanzados de la Población, Universidad Autónoma del Estado de México, Paseo Tollocan S/N, Toluca, Estado de México, México, CP 50110.

ting complex behaviors in very small-time spans. Studying autonomous dynamic systems on reconfigurable hardware offers many practical applications in computational and software engineering. The parallel and reconfigurable architecture of FPGAs is ideal for exploring advanced applications of cellular automata with modifications in design and transition rules, which can address problems in areas such as cryptography and distributed systems simulation.

Keywords: Cellular Automata, FPGA, Transition Rules, Parallel Programming.

Introducción

Los autómatas celulares (AC) son modelos matemáticos que simulan el comportamiento de sistemas dinámicos mediante reglas locales aplicadas en un espacio geométrico determinado, normalmente bidimensional o tridimensional. Este espacio se organiza en retículas, estructuras de líneas horizontales y verticales que dividen el área en celdas individuales. Estas celdas interactúan con sus vecinas de acuerdo con reglas específicas, lo que permite observar el comportamiento emergente del sistema completo [1]. La capacidad de los AC para generar patrones complejos a partir de interacciones simples los ha convertido en herramientas útiles para modelar fenómenos naturales y sociales, como procesos físicos, biológicos y sísmicos [2, 3].

Además, los AC pueden emplearse para el modelado de sistemas computacionales y matemáticos, destacando aplicaciones como la simulación de redes neuronales y la optimización de algoritmos de criptografía [4]. La configuración de estos modelos en retículas permite que el comportamiento de cada celda evolucione en función de sus estados previos y las condiciones que la rodean, como ocurre en sistemas distribuidos y adaptativos [5]. En función de los elementos y reglas de interacción, los AC pueden clasificarse en modelos unidimensionales, bidimensionales o tridimensionales, entre los cuales destacan aquellos que simulan sistemas sociales o dinámicas de fluidos [6].

Una de las principales ventajas de los AC es su capacidad para procesar datos en paralelo, lo que permite modelar sistemas complejos y reducir el tiempo de ejecución de simulaciones en comparación con enfoques secuenciales [7]. Esta ventaja ha impulsado el uso de hardware reconfigurable, como las FPGA's, en la implementación de estos modelos. Las FPGA's permiten una configuración de hardware programable que ofrece mejoras significativas en rendimiento y eficiencia, facilitando la ejecución de sistemas de alta complejidad en tiempo real [8].

En este contexto, el presente trabajo se centra en la implementación del autómata celular conocido como "El Juego de la Vida" de Conway en una tarjeta FPGA. Este modelo se ha popularizado por su simplicidad y por la rica variedad de patrones y comportamientos emergentes que puede generar. En "El Juego de la Vida", el comportamiento de cada celda se define en función de sus ocho vecinos, lo que permite observar desde patrones estáticos hasta oscilaciones complejas y estados caóticos [9]. La implementación en FPGA de este tipo de modelos permite simular sus comportamientos en tiempo real, ampliando las aplicaciones de los AC en el estudio de sistemas complejos.

https://doi.org/10.36677/rici.v3i2.25159 Julio 2025 ISSN: 2992-7447 págs. 42-57

Autómata Celular (AC)

Para modelar un proceso de AC, es adecuado interpretar el espacio como una cuadrícula uniforme de dos dimensiones [10]. Los AC se conforman de cuatro características vinculadas a sus elementos fundamentales; (1) La adyacencia de las células situadas en cualquier espacio dimensional deben evidenciar relaciones de proximidad-distancia entre ellas. (2) Los Estados de cada célula tiene la capacidad de adoptar únicamente un estado de un conjunto potencial de estados, en un determinado momento temporal. (3) La condición inicial supone que la condición de las células se basa en los estados y estructuras de otras células, en particular de las más próximas. (4) La reglas de transición se refiere a las directrices que promueven las modificaciones de estado en cada célula en términos de tiempo y espacio [11].

Un arreglo uniforme, cuadriculado, usualmente infinito, compuesto por objetos iguales denominados células, puede tener *n* dimensiones, sin embargo, para propósitos de simulación en la ingeniería, se implementa el sistema en una, dos o tres dimensiones [12]. El AC se ilustra mediante una función tal como la que se muestra en la Ec. (1).

$$C = (L, S, V, \delta) \tag{1}$$

L representa la cuadricula regular, S representa el conjunto limitado de todos los estados que las células pueden adoptar, V representa el conjunto limitado de células que establecen las relaciones espaciales entre ellas (por ejemplo, proximidad o vecindad de un orden específico) y δ es una función de transición que se aplica a las células en cada instante [3].

Las relaciones de vecindad espacial entre las células pueden ser representadas de la siguiente forma $c \in L$ es $V(c) = \{k_1, k_2, \cdots, k_n | k_j \in L, j = 0, 1, \ldots, n\}$. La relación de vecindad en una célula c se refiere al grupo de células que consideran a la célula como el centro de su área de influencia [11, 13]. Para un AC de una dimensión (es decir, espacio lineal), la vecindad de la célula estudiada se compone de las células vecinas situadas en su lado derecho e izquierdo, lo que se conoce como vecindad de radio r, tal como se muestra en la Ec. (2).

$$V(c) = (c_{i-r}, \cdots, c_{i-1}, c_i, c_{i+1}, \cdots, c_{i+r}) \tag{2}$$

Lo esencial de un AC son las reglas de transición, también conocidas como instrucciones, que simbolizan la progresión del proceso que se pretende modelar. Las normas pueden ser deterministas o estocásticas, de carácter sencillo o muy detallado.

Usualmente, las reglas son estables, sin embargo, algunos trabajos incorporan reglas que evolucionan para representar un proceso específico y es en este momento cuando los AC se convierten en agentes de redes. Es importante destacar que las reglas se fundamentan en la forma en que se establece la proximidad entre las células. En un espacio bidimensional, la adyacencia más habitual es de cuatro células que es llamado el criterio de von Neumann, o de ocho células que es llamado el criterio de Moore. Por lo tanto, en un espacio unidimensional (por ejemplo, lineal) las células próximas pueden ser únicamente tres, las cuales necesariamente tienen frontera compartida, lo que puede generar únicamente ocho tipos de adyacencia.

Si k representa la cantidad de estados celulares (0 y 1) y n representa la cantidad de adyacencias, la cantidad de reglas de transición requeridas para el progreso de un AC de tres bits en un espacio lineal es: $k^n = 2^8 = 256$. Es decir, se cuenta con 256 reglas de transición disponibles para simular el progreso de un modelo de AC de tres bits en el espacio lineal [3].

En la Figura 1, se observa el comportamiento con ciertas reglas que se generan con tres bits. Existe un patrón de células que se repiten en espacios y lapsos de tiempo determinados, estructuras espaciotemporales en diferentes formas geométricas. Cada forma se puede observar como un fractal con una regla determinada.

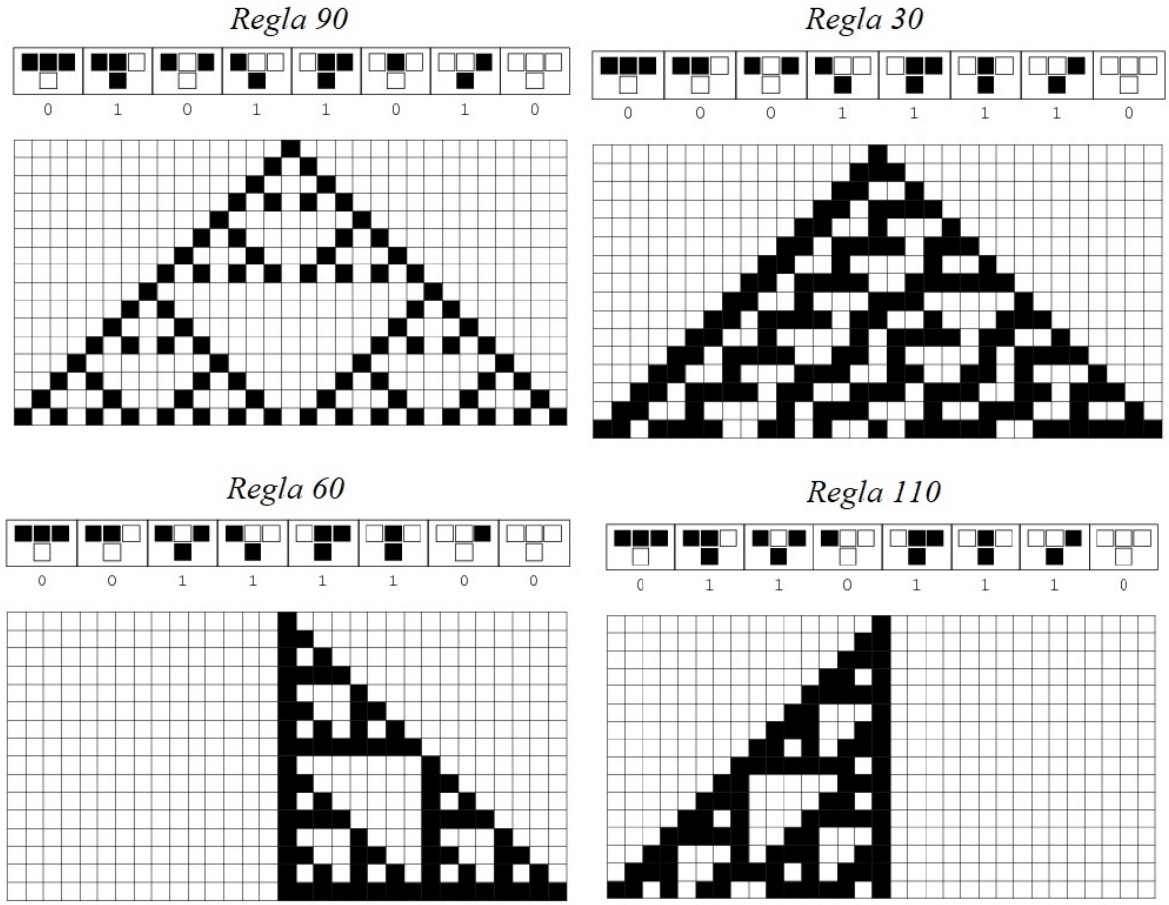

Figura 1: *Comportamiento de reglas Fractales.*

El Juego de la Vida es un autómata celular ideado por John Horton Conway (Liverpool, 26 de diciembre de 1937-Princeton, Nueva Jersey, 11 de abril de 2020) en1970. Combina la noción de autómatas de estado con un concepto de cuadrícula bidimensional y un conjunto específico de reglas para la evolución de su estado. El tipo particular de Vida pertenece a la categoría de AC totalistas no isotrópicos de tipo II [14].

El término totalista representa la propiedad de las aplicaciones de reglas que consideran el estado de

https://doi.org/10.36677/rici.v3i2.25159 Julio 2025 ISSN: 2992-7447 págs. 42-57

cada celda con un peso igual para el estado de cada celda. De hecho, las aplicaciones de reglas se centran únicamente en el recuento de celdas vecinas activas. De acuerdo con esta configuración local, el estado de la celda central se actualiza. Los autómatas totalista no isotrópica de tipo II satisfacen por tanto las siguientes propiedades:

1. Vecindades locales con recuentos de células y pesos independientes para cada estado de célula.
2. Pesos constantes para todos los estados de célula.
3. No hay distinción entre el estado de célula 0-1 y el estado muerto "vacío".
4. Coherencia entre el uso de la simetría y las rotaciones.
5. Pesos y acciones temporalmente acotados.

La función principal del sistema que se introduce a la FPGA es acelerar los cálculos secuenciales y no paralelos necesarios para simular muchas generaciones en cuadrículas bidimensionales muy grandes. Como función extra, se añade una generación de visualización para ayudar a visualizar los resultados.

En el Juego de la Vida, el conjunto de reglas es el siguiente:

a. Cualquier célula viva con menos de dos vecinas vivas muere, como si fuera por subpoblación, como se presenta en la parte superior de la Figura 3(a).

b. Cualquier célula viva con dos o tres vecinas vivas sobrevive a la siguiente generación, como se observa en la Figura 3(b).

c. Cualquier célula viva con más de tres vecinas vivas muere, como si fuera por superpoblación, como se observa en la parte inferior de la Figura 3(c).

d. Cualquier célula muerta con exactamente tres vecinas vivas se convierte en una célula viva, como si fuera por reproducción, como se muestra en la Figura 3(d).

Es la simplicidad de estas reglas lo que ha atraído a un gran número de entusiastas de la Vida, que han encontrado configuraciones estables, oscilantes infinitas. Debido a la simplicidad del juego, ha encontrado aplicaciones en una variedad de ciencias principalmente en la ingeniería de software, software que se ejecuta en servidores de red, grupos y sociedades que impulsa la utilidad de los fractales en la ciencia.

En el Juego de la Vida de Conway, se muestra en la Figura 2, la vecindad de una célula se determina típicamente considerando las ocho células que la rodean inmediatamente: arriba, abajo, izquierda, derecha y las cuatro diagonales. Estas ocho células son vecinas directas de la célula central en el manejo.

Las reglas del Juego de la Vida de Conway son bastante simples, pero generan una sorprendente complejidad. (1) Nacimiento: se reemplaza una célula muerta por una viva si dicha célula tiene tres vecinos vivos como se muestra en la Figura 3(d). (2) Muerte: se reemplaza una célula viva por una muerta si dicha célula no tiene más de un vecino vivo (muerte por aislamiento) o si tiene más de 3 vecinos vivos (muerte por sobrepoblación) como se muestra en la Figura 3(c). (3) Supervivencia: una célula viva permanecerá en ese estado si tiene dos o tres vecinos vivos como se muestra en la Figura 3 (b).

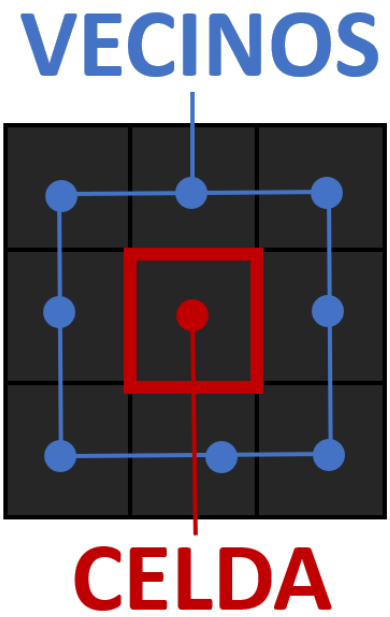

Figura 2: *Vecindad de la célula en el Juego de la Vida.*

Aplicación de FPGA en la implementación del Juego de la vida

Si bien los estudios avanzados y las simulaciones que se basan en AC para fines científicos pueden aprovechar los microprocesadores modernos de alto rendimiento, la investigación basada en matrices de AC a menudo enfrenta los siguientes desafíos:

(1) El tamaño de la cuadrícula que podemos simular está limitado por la memoria del microprocesador en uso, ya que cada estado de celda requiere espacio de memoria.

(2) Los modelos basados en reglas de transición dan como resultado una gran cantidad de operaciones de comparación local para la evaluación y transiciones de estado para el seguimiento fino del espacio vital, lo que resulta en una gran carga temporal para la programación y procesamiento de los cálculos.

(3) Se necesita una respuesta extremadamente detallada a los datos y las operaciones de control para una implementación del sistema eficiente y de alto rendimiento.

(4) Un esquema de acceso que minimice los retrasos de comunicación entre celdas vecinas.

Los FPGA modernos proporcionan una cantidad significativa de elementos lógicos y bloques especializados para operaciones matemáticas. Es posible crear una matriz de más de mil celdas por FPGA con un rendimiento que depende del mecanismo de transición de celdas y de los FPGA. También es posible implementar casi cualquier regla de transición de celdas utilizando lenguajes de descripción de hardware. Utilizando el ejemplo de adición de acarreo de ondulación, la tasa de actualización de celdas del modelo

https://doi.org/10.36677/rici.v3i2.25159 Julio 2025 ISSN: 2992-7447 págs. 42-57

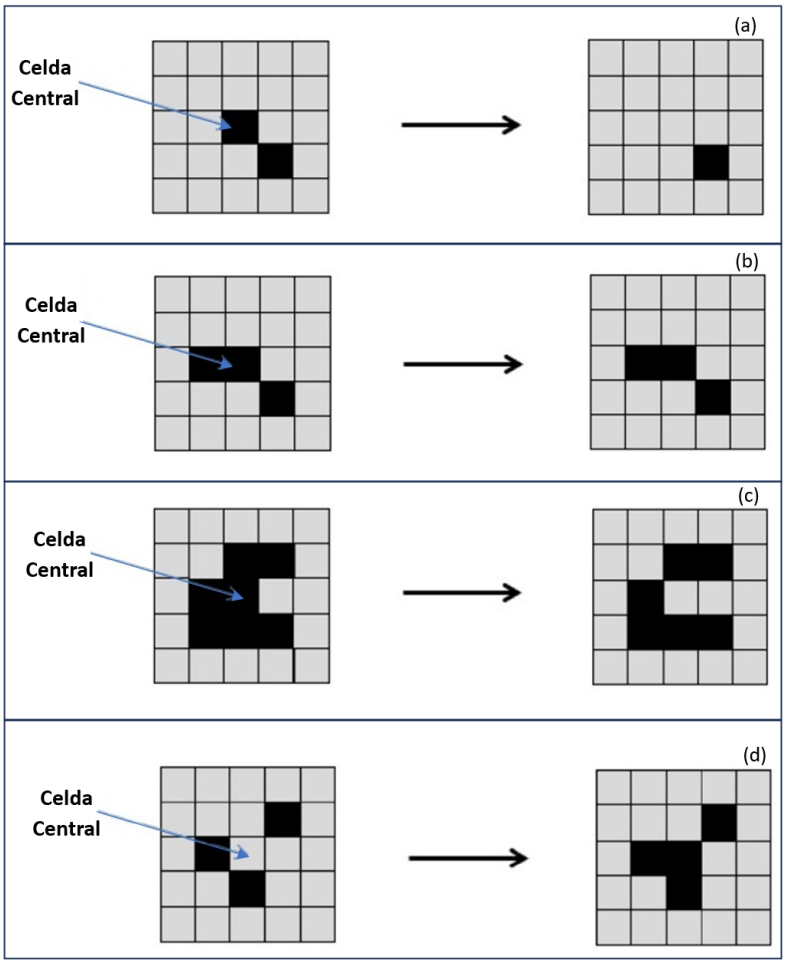

Figura 3: *Patrones del Juego de la Vida.*

de red sin escala fue de más de 10 GHz por grupo de 512 celdas de FPGA, lo que demuestra un alto grado de paralelismo. Gracias al pequeño requisito de área necesario para la memoria dedicada del FPGA y al posible paralelismo extremo de las celdas que puede emplear un FPGA normal, la cantidad de memoria requerida en un procesador de AC basado en FPGA puede ser significativamente menor que la de un sistema paralelo de microprocesadores modernos.

La tarjeta que utiliza este trabajo es la DE0-CV de Intel/Altera. El FPGA es un cyclone V, modelo 5CEBA4F23C7N. Dicho FPGA posee 49,000 elementos lógicos programables, 3080 Kbits de memoria embebida y 4 PLLs fraccionales. Para configurar la tarjeta, esta posee un dispositivo EPCS64 de configuración serial en el FPGA, interfaz USB tipo B en la tarjeta. Soporta modos de configuración JTAG y AS. En cuanto a memoria, posee 64 MB de memoria SDRAM con un bus de datos de 16 bits. Esta es una tarjeta de desarrollo, por lo que contiene una serie de entradas y salidas para realizar proyectos [15].

El programa Quartus II es utilizado para realizar Ingeniería de Software, siendo herramienta de desarrollo de Intel, para dispositivos FPGA. Cuenta tanto con opciones de diseño como de depuración, simulación y programación. En este entorno se pueden desarrollar proyectos de ingeniería de software de varias maneras, tanto en forma esquemática (i.e. con diagramas de bloques) como mediante lenguajes de descripción de hardware. Quartus II provee todas las herramientas necesarias para realizar proyectos en distintos FPGA, por lo que, si se desea cambiar de un FPGA a otro, únicamente es necesario cambiar el dispositivo seleccionado en la configuración, y asignar los pines correspondientes en las entradas y salidas del nuevo dispositivo.

Al modelo de ingeniería que se propone se agrega Verilog está definido formalmente por el estándar de la IEEE 1364. Es un lenguaje de descripción de Hardware (HDL), mediante el cual es posible configurar dispositivos electrónicos programables. Al ser un HDL, la función de Verilog es describir el hardware que será sintetizado en el dispositivo electrónico, así como sus interconexiones, para conseguir el comportamiento deseado. Por ser un lenguaje, tiene una sintaxis específica a utilizar para describir el comportamiento deseado; es importante recalcar que Verilog distingue diferencias entre minúsculas y mayúsculas, por lo que tener cuidado en este sentido es indispensable, ya que puede provocar tanto errores de compilación, como errores lógicos. Otra característica importante en Verilog a tomar en cuenta, es que es ejecutado de manera paralela, a diferencia de lenguajes de programación estructurales, como “C” que se ejecutan secuencialmente [17].

La Matriz de gráficos de video (Video Graphics Array, VGA) es un estándar de gráficos de computadora introducido por IBM en 1987. Proporciona una resolución de pantalla de 640x480 pixeles con hasta 16 colores visibles simultáneamente en su paleta de 256 colores. La VGA ha sido un estándar ampliamente adoptado en la industria de la informática personal y ha evolucionado a través de varios estándares posteriores [17].

Una señal de video VGA contiene 5 señales activas: para sincronización de video: horizontal y vertical; y para asignar color: rojo (Red), verde (Green) y azul (Blue). Al cambiar los niveles analógicos de las tres señales RGB, se generan otros colores. Se realiza un rastreo sobre la pantalla de visualización en una secuencia de líneas horizontales para generar una imagen. La información de color RGB en la señal de video se utiliza para controlar la intensidad de los haces de electrones. Este proceso comienza en el origen (0,0) y repite el pintado de los píxeles de izquierda a derecha y de arriba hacia abajo tantas veces como sea necesario, hasta que el ojo humano no perciba saltos visuales.

El Juego de la Vida se implementó en una matriz de celdas, se utiliza una matriz de 64x48 celdas,

https://doi.org/10.36677/rici.v3i2.25159 Julio 2025 ISSN: 2992-7447 págs. 42-57

donde cada celda está representada por un bit como se muestra en la Figura 4.

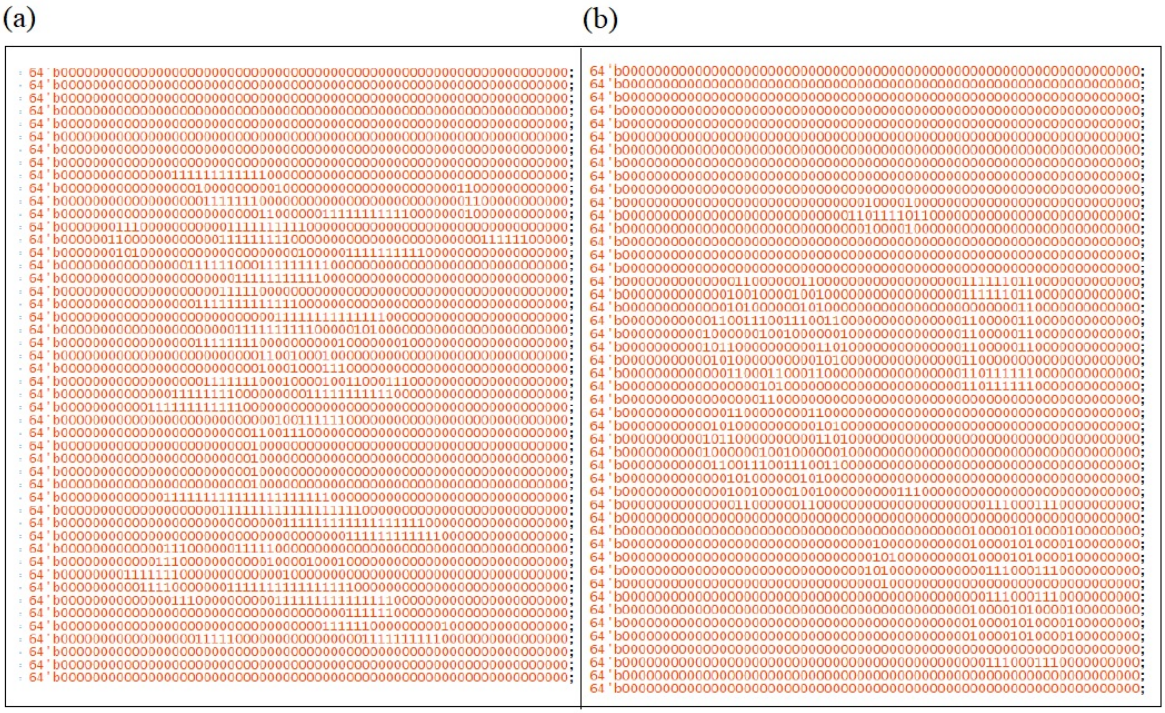

Figura 4: *(a) Matriz inicial precargada. (b) Patrones oscilatorios.*

Se cargaron en las matrices de algunos patrones como se muestra en la Figura 4 (condiciones iniciales). Con un botón externo se realizan cambios de patrones para ver el comportamiento de cada uno de ellos, para esta implementación se colocaron seis condiciones iniciales. La parte medular del Juego de la Vida es la lógica que actualiza en el estado de cada celda con base en sus vecinos. Para cada celda, se contaron los números de vecinos vivos. Esto se hizo utilizando bucles anidados que iteran sobre las posiciones adyacentes (incluyendo diagonales) como se presenta en la Figura 5 incisos (a) y (b).

Dentro del módulo se incluye un segundo divisor de frecuencia que controla el tiempo de transiciones para cada una de las celdas, parte de la ingeniería de software, como se muestra en la Figura 6.

Al contar los vecinos vivos, se aplica una regla del juego de la vida, para determinar el estado futuro de la celda y guardarlo en una matriz temporal. Se actualiza la matriz principal (ver Figura 7).

Se crea un submódulo que determina la posición en un par ordenado (x, y). Se selecciona el color blanco para celda viva y color negro en caso contrario (Figura 8).

En resumen, el proceso de sinterización de la ingeniería de software se implementa con un consumo promedio del 50 % de los recursos totales de la tarjeta FPGA. Esto se debe al uso de sistemas secuenciales y arreglos dimensionales en hardware. Al generar el Modelo RTL (Register Transfer Level) del sistema en Quartus después de la síntesis, muestra el diagrama de la Figura 9.

Una vez diseñada la ingeniería de la aplicación y reconocidas las características de las estructuras y su comportamiento en la tarjeta FPGA, se destacan varios aspectos clave de la aplicación estudiada. En

(a)

```
genvar i;
generate
    for (i = 0; i < 64 * 48; i = i + 1) begin : CELLS
        wire [7:0] neighbours;
        if (i == 0) begin
            // Top-left square
            assign neighbours[0] = cells[64*48 - 1];
            assign neighbours[1] = cells[64*48 - 64];
            assign neighbours[2] = cells[64*48 - 64 + 1];
            assign neighbours[3] = cells[i      + 64 - 1];
            assign neighbours[4] = cells[i      + 1];
            assign neighbours[5] = cells[64     + 64 - 1];
            assign neighbours[6] = cells[64];
            assign neighbours[7] = cells[64     + 1];
        end else if (i == 63) begin
            // Top-right square
            assign neighbours[0] = cells[64*48 - 1 - 1];
            assign neighbours[1] = cells[64*48 - 1];
            assign neighbours[2] = cells[64*48 - 64];
            assign neighbours[3] = cells[i      - 1];
            assign neighbours[4] = cells[0];
            assign neighbours[5] = cells[i      + 64 - 1];
            assign neighbours[6] = cells[i      + 64];
            assign neighbours[7] = cells[i      + 1];
        end else if (i == 64*48 - 64) begin
            // Bottom-left square
            assign neighbours[0] = cells[i - 1];
            assign neighbours[1] = cells[i - 64];
            assign neighbours[2] = cells[i - 64 + 1];
            assign neighbours[3] = cells[i + 64 - 1];
            assign neighbours[4] = cells[i + 1];
            assign neighbours[5] = cells[0 + 64 - 1];
            assign neighbours[6] = cells[0];
            assign neighbours[7] = cells[0 + 1];
        end else if (i == 64*48 - 1) begin
            // Bottom-right square
            assign neighbours[0] = cells[i - 64 - 1];
            assign neighbours[1] = cells[i - 64];
            assign neighbours[2] = cells[i - 64 - 64 + 1];
            assign neighbours[3] = cells[i - 1];
            assign neighbours[4] = cells[i - 64 + 1];
            assign neighbours[5] = cells[0 + 64 - 1 - 1];
            assign neighbours[6] = cells[0 + 64 - 1];
            assign neighbours[7] = cells[0];
        end else if (i < 63) begin
```

(b)

```
            // Top row
            assign neighbours[0] = cells[64*48 - 64 + i - 1];
            assign neighbours[1] = cells[64*48 - 64 + i];
            assign neighbours[2] = cells[64*48 - 64 + i + 1];
            assign neighbours[3] = cells[i - 1];
            assign neighbours[4] = cells[i + 1];
            assign neighbours[5] = cells[i + 64 - 1];
            assign neighbours[6] = cells[i + 64];
            assign neighbours[7] = cells[i + 64 + 1];
        end else if (i > 64*48 - 64) begin
            // Bottom row
            assign neighbours[0] = cells[i - 64 - 1];
            assign neighbours[1] = cells[i - 64];
            assign neighbours[2] = cells[i - 64 + 1];
            assign neighbours[3] = cells[i - 1];
            assign neighbours[4] = cells[i + 1];
            assign neighbours[5] = cells[0 + i - 1];
            assign neighbours[6] = cells[0 + i];
            assign neighbours[7] = cells[0 + i + 1];
        end else if (i % 64 == 0) begin
            // Leftmost column
            assign neighbours[0] = cells[i - 1];
            assign neighbours[1] = cells[i - 64];
            assign neighbours[2] = cells[i - 64 + 1];
            assign neighbours[3] = cells[i + 64 - 1];
            assign neighbours[4] = cells[i + 1];
            assign neighbours[5] = cells[i + 64 + 64 - 1];
            assign neighbours[6] = cells[i + 64];
            assign neighbours[7] = cells[i + 64 + 1];
        end else if ((i + 1) % 64 == 0) begin
            // Rightmost column
            assign neighbours[0] = cells[i - 64 - 1];
            assign neighbours[1] = cells[i - 64];
            assign neighbours[2] = cells[i - 64 - 64 + 1];
            assign neighbours[3] = cells[i - 1];
            assign neighbours[4] = cells[i - 64 + 1];
            assign neighbours[5] = cells[i + 64 - 1];
            assign neighbours[6] = cells[i + 64];
            assign neighbours[7] = cells[i + 1];
        end else begin
            // Middle squares
            assign neighbours[0] = cells[i - 64 - 1];
            assign neighbours[1] = cells[i - 64];
            assign neighbours[2] = cells[i - 64 + 1];
            assign neighbours[3] = cells[i - 1];
            assign neighbours[4] = cells[i + 1];
            assign neighbours[5] = cells[i + 64 - 1];
            assign neighbours[6] = cells[i + 64];
            assign neighbours[7] = cells[i + 64 + 1];
        end
        // Create the module for the cell
        life_cell(neighbours, Clk, KEY[2], cells_reset_state[i], cells[i]);
    end
endgenerate
```

Figura 5: *Programación de la matriz precargada, condiciones iniciales.*

```
wire Clk;
clock(Clk, CLOCK_50,start);
```



```
module clock(Clk, Clk_50,start);
        input Clk_50;
        input start;
        output Clk;
        reg [31:0] counter;

        always @(posedge Clk_50) begin
            if (counter == 1250000)
                counter <= 0;
            else

                if(start) begin
                    counter <= counter + 1;
                end

        end
        assign Clk = (counter == 1250000);
endmodule
```

Figura 6: *Descripción del Divisor.*

https://doi.org/10.36677/rici.v3i2.25159 Julio 2025 ISSN: 2992-7447 págs. 42-57

```
// Create the module for the cell
life_cell(neighbours, clk, KEY[2],

cells_reset_state[i], cells[i]);
```

```
module life_cell(neighbours, clk, Rst, initial_state, state);
    input [7:0] neighbours;
    input clk;
    input Rst;
    input initial_state;
    output reg state;

    wire [3:0] population;
    wire next_state;

    // population is the count of neighbouring cells that are alive
    assign population = neighbours[7] +
                        neighbours[6] +
                        neighbours[5] +
                        neighbours[4] +
                        neighbours[3] +
                        neighbours[2] +
                        neighbours[1] +
                        neighbours[0];

    // next_state is the next state (alive or dead) of this cell
    assign next_state = (population == 2 & state) | population == 3;

    always @(posedge clk or negedge Rst) begin
        if (!Rst) begin
            // When reset fires, return this cell to its initial sta
            state = initial_state;
        end else begin
            // When the clock fires, bring this cell to its next sta
            state = next_state;
        end
    end
endmodule
```

Figura 7: *Submódulo y descripción para determinar el futuro de la celda.*

```
display(cells, x, y, r, g, b);
```

```
module display(cells, x, y, r, g, b);
    input [0:64*48-1] cells;
    input [9:0] x, y;
    output [9:0] r, g, b;


    reg [29:0] RGB;
    always @(x or y) begin
        if (cells[(y / 10) * 64 + (x / 10)-12]) begin
            // The cell that covers this (x, y) coordi
            RGB = 30'b111111111111111111111111111111;
        end else begin
            // The cell that covers this (x, y) coordi
            RGB = 30'b000000000000000000000000000000;
        end
    end
    assign r = RGB[11:8];
    assign g = RGB[7:4];
    assign b = RGB[3:0];

endmodule
```

Figura 8: *Submódulo para determinar posición y color.*

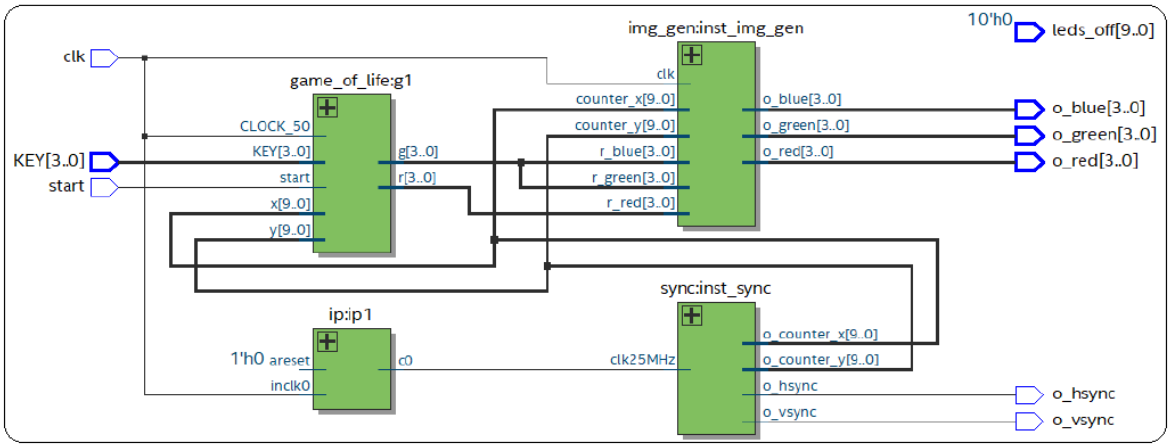

Figura 9: *Diagrama RTL obtenido.*

primer lugar, los autómatas celulares (AC) pueden modelarse para una implementación concurrente en circuitos secuenciales, lo que permite aprovechar las capacidades inherentes de los FPGA en términos de flexibilidad, procesamiento paralelo y velocidad. Estas características, como lo han señalado Sepúlveda et al [18] y Ladino Vega y Cancino de Greiff [19], convierten a los FPGA en dispositivos idóneos no solo para el diseño de ASICs, sino también para aplicaciones avanzadas como simulación, depuración y procesamiento de señales digitales, donde destacan casos de uso como video, que es el enfoque principal de este trabajo.

En comparación con otras plataformas, como procesadores tradicionales o sistemas puramente basados en software, los FPGA sobresalen debido a su capacidad para realizar cálculos en paralelo directamente en hardware, lo que elimina los cuellos de botella asociados al procesamiento secuencial en CPU o GPU. Además, el uso de lenguajes de descripción de hardware como Verilog facilita la personalización de diseños específicos, permitiendo a los desarrolladores optimizar tanto la velocidad como el consumo energético, tal como lo destacan ambos estudios [18, 19].

Otra ventaja clave del uso de FPGA, como lo demuestra esta implementación, es el manejo eficiente de salidas VGA, que permite controlar y actualizar píxeles de manera precisa y en tiempo real. Esto asegura una experiencia visual fluida y un rendimiento óptimo, incluso en aplicaciones complejas como la simulación de AC. En el modelo propuesto, este diseño permite realizar cálculos paralelos fuera de la PC y controlar dispositivos con un mayor nivel de detalle y precisión en los AC seleccionados. Las capturas mostradas en la Figura 10 ejemplifican el funcionamiento del sistema implementado en una pantalla VGA.

CONCLUSIONES

La implementación del autómata celular conocido como el “El Juego de la Vida” en una tarjeta FPGA ha demostrado ser una herramienta efectiva para explorar y visualizar los comportamientos complejos que pueden emerger de reglas locales simples. Este proyecto no solo ha permitido estudiar la dinámica de estos sistemas en un entorno de hardware reconfigurable, sino que también ha mostrado el potencial de los AC para aplicaciones prácticas en el ámbito de la ingeniería.

https://doi.org/10.36677/rici.v3i2.25159 Julio 2025 ISSN: 2992-7447 págs. 42-57

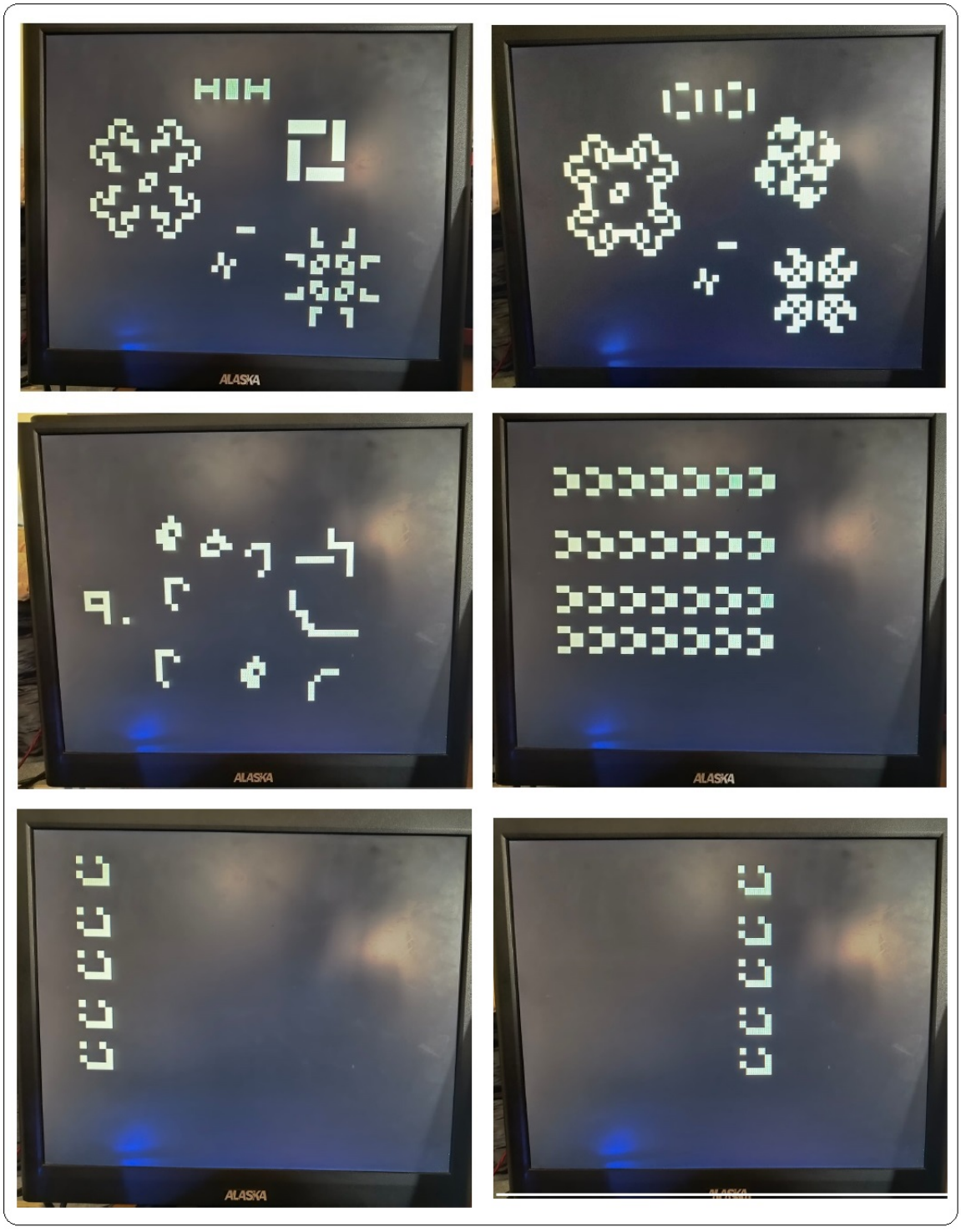

Figura 10: *Patrones aleatorios en el Juego de la Vida.*

Además, este tipo de implementaciones resulta especialmente beneficioso en el contexto educativo, donde proporciona a los estudiantes de ingeniería electrónica una experiencia práctica y visual que facilita la comprensión de conceptos fundamentales de AC. La posibilidad de observar la evolución de patrones en tiempo real en un FPGA permite a los alumnos desarrollar una intuición más profunda sobre la interacción de celdas y el impacto de las reglas locales en el comportamiento global del sistema.

Este trabajo también puede servir como base para proyectos más complejos y especializados. La arquitectura paralela y reconfigurable de los FPGA's es idónea para explorar aplicaciones avanzadas de AC, tales como sistemas de encriptación y cifrado, generación de secuencias pseudoaleatorias y modelado de redes neuronales. Con modificaciones en el diseño y en las reglas de transición, es posible expandir el modelo para abordar problemas en áreas como la criptografía y la simulación de sistemas distribuidos.

Cabe destacar que la implementación de AC en FPGA's no solo contribuye al conocimiento en ingeniería de sistemas y hardware, sino que también ofrece un enfoque didáctico y adaptable para la enseñanza y el desarrollo de proyectos avanzados, potenciando tanto la comprensión académica como la innovación aplicada en el ámbito de la ingeniería electrónica.

DECLARACIÓN DE CONTRIBUCIÓN DE AUTORES Y COLABORADORES

Jazmín González Contreras: Conceptualización, Metodología, Software, Redacción-Preparación del borrador original y Edición. **Eduardo Jiménez López**: Conceptualización, Investigación, Metodología, Software, Curación de datos, Redacción-Preparación del borrador original y Edición. **Ricardo Eliú Lozoya Ponce**: Conceptualización, Investigación, Metodología, Software, Curación de datos, Redacción-Preparación del borrador original y Edición. **Julio César Villagrán Ruiz**: Supervisión, Revisión y Edición.

REFERENCIAS

[1] D. Alejandro and R. Gómez. Descripción y Aplicaciones de los Autómatas Celulares. 2011. Disponible en: https://delta.cs.cinvestav.mx/~mcintosh/cellularautomata/Summer_Research_files/Arti_Ver_Inv_2011_DARG.pdf.

[2] B. L. Benoso and C. Yáñez Márquez. Autómatas celulares alfa-beta. *Computación y Sistemas*, 16(4), pp. 471–479, 2012. Disponible en: https://www.redalyc.org/articulo.oa?id=61524670009.

[3] E. Jiménez, T. Chávez, and C. Garrocho. Modelando la expansión urbana con autómatas celulares: aplicación de la Estación de Inteligencia Territorial (Christaller). *Geografía y Sistemas de Información Geográfica, Geosig*, 12, pp. 1–26, 2018. Disponible en: https://acortar.link/7gMbwZ.

[4] I. Dario and L. Vega. AUTÓMATAS CELULARES EVOLUCIONADOS SOBRE FPGA. Pontificia Universidad Javeriana Bogotá, 2012. Disponible en: https://repository.javeriana.edu.co/bitstream/handle/10554/12733/LadinoVegaIvanDario2012.pdf?sequence=1&isAllowed=y.

https://doi.org/10.36677/rici.v3i2.25159 Julio 2025 ISSN: 2992-7447 págs. 42-57

[5] M. H. Kirkeby and M. Schoeberl. Towards Comparing Performance of Algorithms in Hardware and Software. 2022. Disponible en: http://arxiv.org/abs/2204.03394.

[6] B. Martin and P. Solé. Pseudo-random Sequences Generated by Cellular Automata. 2008, pp. 401–410. Disponible en: https://hal.science/hal-00305407v1.

[7] R. Rechtman. Una introducción a autómatas celulares. *Ciencias*, 24, pp. 23–29, 1991. Disponible en: https://www.revistacienciasunam.com/es/172-revistas/revista-ciencias-24/1572-una-introducci%C3%B3n-a-aut%C3%B3matas-celulares.html.

[8] A. C. Rojas and Á. Rojas Matas. Autómatas celulares y aplicaciones. *UNIÓN Revista Iberoamericana de Educación Matemática*, 46, pp. 33–48, 2016. Disponible en: http://www.fisem.org/web/unionhttp://www.revistaunion.org.

[9] T. Smilkstein, K. K. Tati, P. Barve, M. L. Hai, K. Sajjapongse, and D. K. Sharma. An evolutionary algorithm testbed for quick implementation of algorithms in hardware. En *2009 IEEE Workshop on Evolving and Self-Developing Intelligent Systems*, pp. 51–57. IEEE, 2009. doi: https://doi.org/10.1007/978-3-540-48302-1_31.

[10] E. Jiménez-López and L. A. López-Rivera. Artificial neural networks in the application of the growth of the urban sprawl. *Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI*, 11(21), pp. 109–119, 2023. doi: https://doi.org/10.29057/icbi.v11i21.10565.

[11] E. Jiménez López. Deep Learning in the Expansion of the Urban Spot. En *Study of Complex Systems and their Applications Conference*, pp. 37–51. Cham: Springer Nature Switzerland, 2023. doi: https://doi.org/10.1007/978-3-031-51224-7_3.

[12] E. Jiménez-López. Inverse filter in the growth of urban sprawl with cellular automata model. En *Complex Systems and Their Applications: Second International Conference (EDIESCA 2021)*, pp. 231–247. Cham: Springer International Publishing, 2022. doi: https://doi.org/10.1007/978-3-031-02472-6_12.

[13] E. Jiménez López. Cadenas de Markov espaciales para simular el crecimiento del Área Metropolitana de Toluca, 2017–2031. *Economía, sociedad y territorio*, 19(60), pp. 109–140, 2019. doi: https://doi.org/10.22136/est20191324.

[14] S. A. Sanfélix, M. B. Camps, and M. Christodoulou. Landscape Design Methodology: Pattern Formation Through the Use of Cellular Automata. *Temes de Disseny*, (35), pp. 26–41, 2019. doi: https://doi.org/10.46467/TdD35.2019.26-41.

[15] Terasic Technologies. *DE0-CV user manual*. Disponible en: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=163&No=921&PartNo=4.

[16] E. Schkufza, M. Wei, and C. J. Rossbach. Just-in-time compilation for Verilog: A new technique for improving the FPGA programming experience. En *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 271–286, 2019. doi: https://doi.org/10.1145/3297858.3304010.

[17] D. G. Costa. Visual sensors hardware platforms: A review. *IEEE Sensors Journal*, 20(8), pp. 4025–4033, 2019. doi: https://doi.org/10.1109/JSEN.2019.2952447.

[18] J. Sepúlveda, C. Camargo, and S. Bolívar. Metodología de Implementación de Autómatas Celulares en FPGA. 2016. Disponible en: https://silo.tips/download/metodologia-de-implementacion-de-automatas-celulares-en-fpga.

[19] I. D. Ladino Vega and H. F. Cancino de Greiff. Autómatas Celulares Evolucionados sobre FPGA. Pontificia Universidad Javeriana, 2012. Disponible en: https://repository.javeriana.edu.co/handle/10554/12733.