AUTÓMATAS CELULARES EVOLUCIONADOS SOBRE FPGA

IVAN DARIO LADINO VEGA

UNIVERSIDAD JAVERIANA
FACULTADDEINGENIERÍA
MAESTRIA ENINGENIERÍAELECTRÓNICA
BOGOTÁ,DC
2012

AUTÓMATAS CELULARES EVOLUCIONADOS SOBRE FPGA

IVAN DARIO LADINO VEGA

Trabajo de Grado de Maestría

Director
Héctor Fernando Cancino de Greiff Ph.D.

UNIVERSIDAD JAVERIANA
FACULTADDEINGENIERÍA
MAESTRIA ENINGENIERÍAELECTRÓNICA
BOGOTÁ,DC
2012

CONTENIDO

1 INTRODUCCIÓN

El objetivo de este proyecto fue el de implementar autómatas celulares sobre FPGA y evolucionarlos sobre PC. Para tal efecto el proyecto se dividió en dos componentes, uno el de la máquina genética sobre el PC, y el otro el correspondiente al autómata implementado sobre la FPGA.

Para el desarrollo de la Máquina Genética se requirió evaluar las alternativas para el proceso de evolución y de medición de aptitud de los individuos de la población, para tal efecto se diseñó una estrategia para el cruce de los cromosomas, en el cual mediante máscaras de cruce y un porcentaje de mutación se generan los hijos. Las ventajas de emplear algoritmos genéticos en el proyecto consisten básicamente en que no se requiere conocer con precisión el problema a resolver, se trabaja con varias soluciones simultáneamente y se emplean operadores estocásticos; en el caso de los autómatas no es necesario determinar matemáticamente la función de transición de las celdas ya que el proceso de evolución permite encontrarla como resultado del proceso de evolución de la población.

Como el mecanismo de evaluación para el autómata correspondió al de filtrado de imágenes, se optó por emplear cromosomas cuya genética se conformará a partir de dos filtros con máscaras de tamaño 3x3, y un patrón de ecualización de los valores de los pixeles de la imagen.

La evaluación de los individuos se realizó por medio del filtrado y finalmente la medición de contraste de la imagen a partir de la varianza del Histograma, es decir que el proceso evolutivo emplea para la medición de aptitud de los individuos el contraste de las imágenes, el cual se mide a partir del histograma. Se dispone de numerosas técnicas para la medición de contraste, pero una de las más sencillas y de bajo computo que permite evolucionar la máquina en menor tiempo, corresponde a la medición de varianza del histograma de la imagen resultante del proceso de filtrado.

Para el filtrado se diseñó un algoritmo propio de FFT en razón a que se intento implementar una transformada basada en teoría de números (NTT), para lo que se requiere un algoritmo rápido similar al de la FFT. La intención de emplear NTT radica en que requiere menos computo y por ende menos tiempo que la FFT.

La máquina genética en pocas palabras evoluciona la población y selecciona los mejores individuos operando los filtros y la ecualización sobre la imagen de prueba, el resultado es entonces un par de máscaras de filtrado y un patrón del ecualizador, dicha información es enviada al autómata sobre la FPGA.

El autómata sobre la FPGA se divide básicamente en tres componentes: uno el autómata lineal, el cual realiza operaciones lineales con base en las máscaras

para vecindades de 3X3; el segundo componente corresponde al del autómata no lineal, el cual realiza operaciones no lineales sobre la información binarizada entregada por el autómata lineal, además mediante una función no lineal de ecualización a la entrada del autómata permite realizar filtrado no lineal a través de la ecualizaciones las imágenes antes del proceso de filtrado.

El último componente sobre la FPGA corresponde a la máquina de control que permite controlar el proceso del Autómata de dos capas implementado. Dicho control se implemento sobre el procesador embebido NIOS II de Altera Corp. (Altera Corporation, 2009). El procesador NIOS dispone de recursos de comunicaciones como puerto seriales, por tanto se implemento la comunicación entre la FPGA y el PC aprovechando los recursos dispuestos en el NIOS II.

El diseño de la máquina genética para el filtrado de imágenes se empleó únicamente para validar el funcionamiento del autómata. Es de aclarar tal como se explicó en el anteproyecto que no fue objetivo del proyecto implementar una mejor o más potente técnica de filtrado de imágenes.

Los resultados alcanzados con el proyecto consistieron básicamente en: un autómata de dos capas sobre la FPGA, una lineal y otra no lineal, una máquina genética que evoluciona individuos para que el filtrado y ecualización mejore el contraste de la imagen para el proceso de filtrado y detección de bordes sobre la FPGA. Se obtuvo un algoritmo propio para la FFT.

De otra parte se logró detectar bordes a partir en una forma muy eficiente con base en la medición de la orientación de la información sobre el autómata lineal y la aplicación de un autómata no lineal.

2 DESCRIPCION DEL PROBLEMA DE INVESTIGACIÓN

Los sistemas digitales para la implementación de las diversas aplicaciones tales como criptografía, simulaciones físicas, procesamiento de señales y de imágenes entre otras, requieren cada vez más, de plataformas concurrentes y de alto computo que permitan obtener tiempos de ejecución inferiores, de tal forma que se reduzcan o eliminen los procesos secuenciales como los implementados sobre plataformas basadas en PC.

Las aplicaciones sobre plataformas digitales involucran actualmente la necesidad de apropiar técnicas relacionadas con la inteligencia artificial y los sistemas bio-inspirados, debido al aumento de la complejidad de los problemas [1], [2], [7]. El concepto de complejidad [3], plantea la necesidad de emplear estrategias que disminuyan la complejidad en dos sentidos: uno el de la disminución de la dependencia entre los requerimientos funcionales, y el otro, la reducción de la

entropía, es decir: el promedio de incertidumbre asociado con las probabilidades de cumplir con los requerimientos funcionales.

Lo anterior significa que se deben apropiar nuevos paradigmas, un conjunto nuevo de conceptos y modelos de las matemáticas, sin los cuales es difícil comprender los conceptos subyacentes y concebir soluciones a problemas prácticos.

Una estrategia para plantear soluciones viables a problemas difíciles, consiste en dividir o descomponer la solución en una colección interrelacionada de componentes mucho más simples, uniformes e independientes, ese es el caso de los Autómatas celulares que se representan básicamente por una colección de celdas elementales conectadas entre sí, con la característica de que para cada celda su estado o valor se actualiza a partir del estado de las celdas circunscritas en su vecindad, aplicando un conjunto de reglas específicas [5]. De tal forma que un autómata en dos dimensiones por ejemplo, se percibe como una malla donde sus nodos corresponden a las celdas, y las aristas o lazos de conexión entre nodos representan la relación de vecindad para cada celda. Cada nodo contiene un valor o estado que cambia a medida que trascurre el tiempo discreto, de acuerdo a la aplicación de las operaciones o reglas a los estados de la celda y sus celdas vecinas.

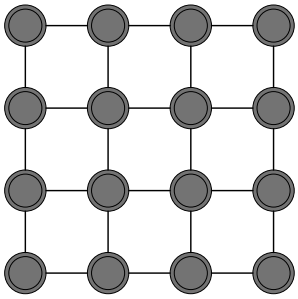

Figura 1. Autómata Celular en dos dimensiones. Fuente: el autor

La estructura de los autómatas de Von Neumann se ha considerado la base de diseño e implementación de las arquitecturas de los sistemas de computación clásicos como: microprocesadores, microcontroladores, y los PCs mismos. Por su parte los autómatas celulares como se conciben en la actualidad, determinan una granularidad mayor, es decir una mayor de densidad de elementos lógicos que la disponible en los sistemas convencionales, dicha densidad o granularidad arrastra consigo los siguientes hechos fundamentales [5]:

- La necesidad de apropiar conceptos y modelos matemáticos que requieren el estudio de algunos temas difíciles propios de las matemáticas formales [8].

- EL autómata celular funciona concurrentemente, ya que cada celda se actualiza simultáneamente con las demás celdas [5].
- La estructura de un autómata de una o dos dimensiones tiene semejanzas muy grandes con la estructura granular de los sistemas digitales discretos [1].
- Los modelos matemáticos de los autómatas celulares para la operación de transición de estado consiste básicamente en un campo de Galois [20].

En consecuencia de lo anterior se percibe que la herramienta ideal serían las CELL MATRIX en razón a que además de su funcionamiento concurrente como el de los autómatas celulares, permite que las celdas se programen a partir de otras celdas y no solo a partir de un único modulo de configuración, ello como lo demostró Sekanina [1], permite reducir el tiempo de configuración secuencial del sistema empleado por los programadores tradicionales. Las CELL MATRIX como se muestra en la figura 2, constan básicamente de una grilla de tablas de "Lookup" conmutables, es decir tablas de búsqueda, con entradas y salidas de datos hacia su vecindad (al igual que los autómatas celulares) y con, entrada y salida de parámetros de configuración.

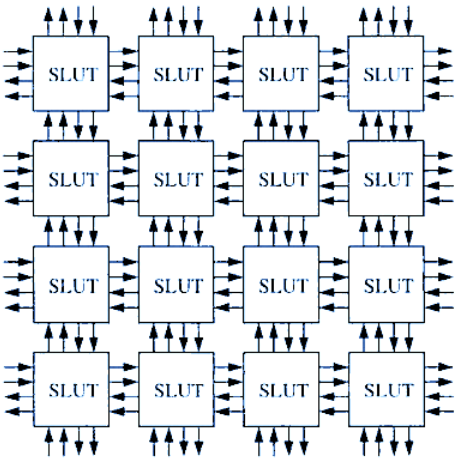

Figura 2. Estructura básica de una Cell Matrix.
Fuente. Tomado de (Sekanina L. , 2004)

El inconveniente es que hoy en día, no se dispone de una herramienta de desarrollo comercial; se esperaba según Sekanina [1]que para el 2012 se dispusiera de las primeras herramientas comerciales, sin embargo a la fecha aún no se dispone de dispositivos ni de herramientas de desarrollo. En consecuencia la alternativa es trabajar con sistemas de hardware reconfigurable como las plataformas de desarrollo basadas en FPGAs, pero es importante subrayar que aunque comparten ciertas características con los autómatas celulares, tiene algunas desventajas que se relacionan a continuación:

- La granularidad de una FPGA es decir la densidad de unidades lógicas es muy inferior a la de un autómata celular o a la de una Cell Matrix [1].
- La memoria de configuración y las interconexiones ocupan el 80 a 90% del espacio en las FPGAs [1].
- La velocidad de una FPGA es aproximadamente 10 veces más baja que la de un ASIC [1].
- La diferencia más grande y sustancial entre las Cell Matrix y las FPGA, consiste en que las celdas se pueden reconfigurar sin necesidad de reprogramar el hardware, simplemente las nuevas reglas se cargan en el Cell matrix como si fuesen datos, mientras que en las FPGA se debe reprogramar cada vez que se requiere cambiar la configuración [1].

Una alternativa a la implementación sobre hardware consiste en simular los autómatas sobre plataformas basadas en PC, lo que implica desarrollar algoritmos eficientes para compensar en alguna medida el proceso secuencial que siguen los procesadores convencionales de los PCs.

Para que el autómata celular desarrolle una tarea específica, como la de filtrado de imágenes, simulación de tráfico, cifrado de datos etc., se debe realizar un trabajo en una serie de etapas: análisis, modelamiento matemático, diseño, implementación y verificación, resolviendo las siguientes preguntas:

- ¿Cuál es el intervalo de valores que representan los estados de las celdas?
- ¿Cuál es la vecindad adecuada para cada celda?
- ¿Cuál es el conjunto de reglas u operaciones sobre los estados de la vecindad para actualizar el valor de cada celda?
- ¿Cuánto instantes de tiempo hacia el pasado del sistema, se valoran los estados anteriores de la vecindad, para obtener el valor actual de cada celda?
- ¿Qué tamaño máximo tendrá el autómata celular teniendo en cuenta las limitaciones de las FPGA?
- Como cambia la estructura de los autómatas de acuerdo a la aplicación.

3 OBJETIVOS

3.1 OBJETIVO GENERAL

Modelar, implementar y evaluar autómatas celulares evolucionados sobre FPGA

3.2 OBJETIVOS ESPECIFÍCOS

- Diseñar y validar el modelo matemático de los autómatas celulares en una y dos dimensiones.

- Diseñar e implementar la máquina genética para evolucionar los autómatas celulares sobre una plataforma basada en PC.
- Implementar los autómatas celulares reconfigurables sobre una herramienta de desarrollo basada en FPGA, sin reprogramar la FPGA.
- Evaluar el desempeño de la implementación en hardware mediante una aplicación orientada al proceso de datos en dos dimensiones.

4 MARCO TEORICO

El marco para el desarrollo del proyecto lo constituyen los sistemas bio-inspirados en razón a que las propiedades de Filogenia, Ontogenia y Epigénesis resultan muy atractivas para el diseño de sistemas digitales ya que permiten imitar la naturaleza en cuanto los rasgos evolutivo y la construcción de sistemas densos a partir de unidades más simples (celdas). Por lo anterior razón se realiza una breve introducción a dicho paradigma así como también a su clasificación.

4.1 SISTEMAS BIO-INSPIRADOS Y MODELO POE

Los sistemas bio-inspirados son sistemas que incorporan características de los sistemas biológicos como lo son la auto-reparación, la auto-replicación, la tolerancia a fallas, el paralelismo masivo, entre muchas otras. Dichos sistemas se organizan en el llamado modelo *POE*, el cual es un modelo diseñado por Sipper entre otros [22,23,24,25,26] en el cual se ubican los sistemas en un espacio tridimensional en donde cada uno de los ejes representa respectivamente la Filogenia, la Ontogenia y la Epigénesis, niveles básicos de organización de la vida, y en donde la técnica objeto de estudio se representa como un punto dentro de ese espacio, siendo su distancia al origen la que indica “que tanto tiene de” cada uno de los ejes.

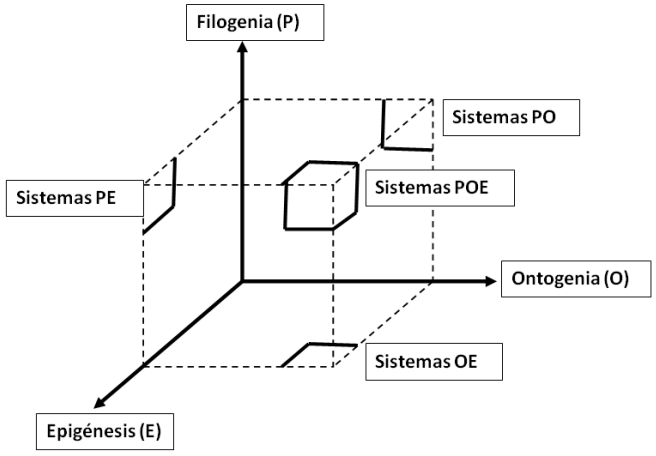

Figura 3. Modelo POE para la clasificación de los sistemas Bio-Inspirados
Fuente. Tomado de (Sekanina L. , 2004). Modificado por el autor.

Tomando como fuente a (Sipper, Sánchez, Mange, Toamssini, Pérez-Uribe, & Stauffer, 1997) y (Stauffer, Sipper, & Pérez-Uribe, Some Applications of FPGA in Bio-Inspired Hardware Systems), el contenido de cada uno de los ejes y su aplicación más típica se describe a continuación:

4.2 FILOGENIA

De manera general, la filogenia es sinónimo de evolución y hace referencia básicamente a la reproducción de los organismos y al pasar de la información genética de los padres a los hijos, proceso en el cual entran en juego operadores genéticos como la mutación y el cruce, que hacen que la información genética de los hijos no sea siempre una copia idéntica de la información genética de sus padres. Este hecho, el cual es gobernado por principios no deterministas juega un papel importante en la supervivencia de las especies, ya que dichas alteraciones hacen que algunas especies se adapten mejor a cambios de su entorno, mientras las que no son capaces de lograr dicha adaptación cesan su existencia; dicha capacidad de adaptación es medida por una cantidad denominada *fitness* (aptitud)

Una forma de visualizar el *fitness* de los individuos es mediante “el modelo de Wright de la superficie adaptativa. Posibles combinaciones de rasgos biológicos definen puntos sobre un espacio de secuencia de varias dimensiones, en donde cada eje coordenado corresponde a cada uno de los rasgos. Una dimensión adicional es usada para modelar los valores de fitness para cada punto en el espacio, reflejando las ventajas selectivas de los correspondientes individuos de manera que cuando el fitness incrementa, la población se mueve colina arriba en la superficie”. Dicho de otra forma, el proceso de mejorar el fitness de una población, se puede denotar, por lo menos a primera vista, como un proceso de optimización.

4.3 ALGORITMOS EVOLUTIVOS

En vista de que “la evolución” es un proceso de optimización que puede ser simulado usando un computador u otro dispositivo para realizar funciones relacionada con la ingeniería [19], es que surgen los denominados algoritmos evolutivos como aplicación más sobresaliente de la filogenia. En general son algoritmos implementados en sistemas de software, que realizan búsqueda estocástica con fines de optimización de variables que hacen las veces de rasgos genéticos. Se basan en la teoría de Darwin sobre evolución de las especies, más precisamente en el llamado Darwinismo molecular.

Los algoritmos evolutivos de acuerdo al aspecto de como el tipo de operador genético que implementen, se dividen principalmente en los denominados algoritmos genéticos, programación genética, estrategias evolutivas y programación evolutiva; pero independiente del tipo de algoritmo en particular que

se use, en general se tiene que consisten de tres componentes: La representación de la información genética, los operadores genéticos utilizados por los algoritmos, la evaluación de las soluciones, llamada *fitness*, y finalmente la selección de las soluciones.

Se puede decir que el proceso que desempeña un algoritmo evolutivo, de acuerdo a [2] y [24], es que dado un problema en particular, una representación adecuada es seleccionada para codificar las soluciones potenciales, luego de esto, un número finito de individuos que forman las soluciones potenciales es seleccionado de forma aleatoria. Seguido a esto la población de individuos es sometida a los procesos de variación genética (cruce, mutación), selección y evaluación, de lo cual resulta una nueva población de individuos más aptos, los cuales se someterán de nuevo a este proceso hasta que un criterio de parada sea alcanzado.

Lo anterior se expresa matemáticamente mediante la siguiente relación [2],

Ecuación 1: evolución

$$P(t+1) = S\left(E\left(V\left(P(t)\right)\right)\right)$$

Fuente: propia

Donde,

$P(t)$*, es la población inicial de N individuos de soluciones potenciales.*
$V(\ \ )$*, es el operador genético, cruce y/o mutación.*
$E(\ \ )$*, es el proceso de evaluación que asigna el fitness a cada individuo*
$S(\ \ )$*, es el proceso de selección de acuerdo al fitness de cada conjunto de soluciones.*
Finalmente, $P(t+1)$ *es la nueva población generada.*

4.4 ONTOGENIA

Biológicamente, la ontogenia hace referencia a la replicación y división celular en el organismo, en otras palabras, al crecimiento y desarrollo del mismo. Dicho proceso es propio de los organismos multicelulares y se puede clasificar en dos etapas, la replicación celular propiamente, y la especialización celular.

La replicación celular, proceso llamado mitosis, es la creación de nuevas células hijas a partir del cigoto, célula diploide resultante de la fecundación, y posee como característica sobresaliente el que cada célula hija posee una copia idéntica (bajo condiciones normales) del ADN de su célula madre, es decir no hay lugar para variaciones en la información genética como ocurre con la reproducción en la filogenia.

La especialización celular es la función en particular que las células desempeñan dentro del organismo, es decir, algunas células serán las encargadas de dar la forma del individuo, y formarán los huesos y músculos, otras se encargarán de la distribución de nutrientes y demás sustancias necesarias y formarán los vasos sanguíneos, entre otros. Dicha especialización resulta de la regulación de la expresión de varias partes del genoma, la cual está determinada por factores como la posición de una célula respecto a sus alrededores.

Sumado al hecho de que todas las células poseen el mismo material genético, se tiene que por lo menos en principio, cualquier célula tiene la capacidad de desempeñar la función de cualquier otra célula, es decir, de reemplazar en dado caso a una célula que ha sufrido daños o que ha muerto.

Lo recién descrito muestra como a partir de un componente relativamente sencillo, el cigoto, se consigue un organismo completamente desarrollado, capaz de lograr tareas de grandísima complejidad, y con capacidad de repararse a sí mismo, lo cual dentro del contexto de los sistemas de hardware resulta de gran atracción para el diseño de sistemas extremadamente complejos y con tolerancia a fallas.

4.5 SISTEMAS BASADOS EN ONTOGENIA

Comparados con los sistemas biológicos, los sistemas basados en la ontogenia han estado bastante atrasados en cuanto al desarrollo de aplicaciones, esto se debe a que la ontogenia es sinónimo de replicación de individuos, lo cual en el campo de ingeniería se podría traducir como creación de más hardware de forma autónoma por el sistema, lo cual hasta el día de hoy es un tema de investigación.

La técnica abordada para el diseño de estos sistemas es hacer uso del llamado Hardware Reconfigurable, que es básicamente una plataforma de hardware con recursos finitos en la cual se pueden construir y reprogramar infinidad de veces circuitos electrónicos. Al utiliza este tipo de dispositivos se logra una aproximación a la replicación observada en la naturaleza.

Lukas Sekanina en su disertación doctoral presentada en el libro: "*Evolvable Components: from theory to hardware implementations* " [1] realiza una revisión del estado del arte de los sistemas bio-inspirados, a partir de la cual se planteó el método para abordar el proyecto desarrollado.

4.6 MEDIDA DE LA ORIENTACIÓN DE LA INFORMACIÓN

Este método fue introducido por Yang Xuan, Liang Dequn [29], y como se expone en su artículo "Multiscale edge detection based on direction information". Consiste básicamente en la aplicación de filtros detectores de bordes con ángulos uniformemente distribuidos en un rango de 180 grados o 360 grados:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix}$$

Figura 4. Filtros detectores de bordes para ángulos de 0, 45, 90 y 135 grados
Fuente. El autor.

Luego de aplicar todos los filtros se selecciona para cada pixel de la imagen aquel que produzca el valor máximo y con ese valor se construye la imagen resultante.

5 ESPECIFICACIONES

Básicamente las especificaciones del proyecto desarrollado corresponde a:

- Diseño de autómatas sobre FPGA
  - Autómata Lineal de 16 x16 celdas de 256 estados cada una
  - Autómata no lineal de 16x16 celdas y de dos estados cada una
  - Implementación de una operación de mapeo no lineal (ecualización) sobre VHDL
  - Máquina de control del autómata de dos capas
- Diseño de la Máquina Genética
  - Algoritmo de generación de la población inicial
  - Algoritmo para el cruce y mutación
  - Algoritmo de filtrado sobre FFT
  - Algoritmo de evolución
  - Algoritmo de Ecualización
  - Algoritmo de evaluación de aptitud
  - Algoritmo de evolución.

6 DESARROLLO

A partir de los objetivos, el desarrollo del proyecto significo dividir el problema en dos componentes principales, uno el de la implementación de los autómatas reconfigurables sobre la FPGA, y el otro, la implementación de la máquina genética sobre la plataforma PC.

Como se mencionó en el marco teórico, la limitante más grande en cuanto a recursos o hardware radica en la densidad de elementos lógicos de la FPGA. En las pruebas realizadas con autómatas en dos dimensiones y de dos estados se construyó en un proyecto anterior [12] un autómata de una capa de 64 x64 celdas que consume 4096 ALUTs (tabla de búsqueda adaptativa) para un total de 4096 celdas sobre una STRATIX II EP2S60 de Altera [13].Sin embargo la implementación de autómatas para ser evaluados en el filtrado de imágenes requiere de un número de estados mucho mayor; si las imágenes en escala de grises emplean una representación típica de 256 niveles (8 bits), cada celda del autómata debe disponer entonces de 256 estados.

La actualización del estado de cada celda puede involucrar operaciones de tipo lineal y no lineal; las de tipo lineal se puede representar básicamente por una convolución, es decir que la actualización del estado de la celda se realiza con base en una máscara aplicada sobre los vecinos (vecindad Moore), mediante una operación como la mostrada en la siguiente ecuación:

Ecuación 2. Actualización de celda por convolución

$$c'(\overrightarrow{n_1, n_2}) = \sum_{p_1, p_1 \in \langle moore\ r=1 \rangle} c(\overrightarrow{p_1, p_2}) h(\overrightarrow{p_1 - n_1}, \overrightarrow{p_2 - n_2},)$$

Fuente: propia.

Con *h* la máscara que se aplica sobre la vecindad, $c(\overrightarrow{n_1, n_2})$ el estado actual de la celda, $c'(\overrightarrow{n_1, n_2})$ el nuevo estado y r el radio de la vecindad.

En el caso de que la operación no sea lineal se pueden emplear reglas como:

Ecuación 3. Actualización de celda por convolución

$$c'(\vec{n}) = f[c(\vec{n} + \overrightarrow{n_1}), c(\vec{n} + \overrightarrow{n_2}), \dots, c(\vec{n} + \overrightarrow{n_m})]$$

Fuente: propia

La función ***f*** denota la operación sobre la vecindad para determinar el nuevo estado. En muchas de las aplicaciones la función ***f*** consiste básicamente en una tabla de búsqueda.

Con base en lo anterior, las limitaciones propias de las FPGAs, las diferencias entre las FPGAs y los autómatas propiamente dichos, los objetivos trazados y el método de evaluación a partir del proceso de filtrado de imágenes, hubo que optar por una estrategia tanto para la implementación del autómata sobre la FPGA, así como también para el diseño de la Máquina Genética sobre el PC.

En cuanto a la implementación del autómata se consideró importante:

- A partir de la herramienta de desarrollo, implementar un autómata en dos dimensiones que permitiese realizar operaciones lineales y no lineales sobre los datos almacenados en el autómata
- Para las operaciones lineales implementar un autómata lineal de un número grande de estados para poder tratar datos de la naturaleza de las imágenes en escala de grises.
- Disponer de una función no lineal para emular filtrado no lineal a partir de filtros lineales.
- Para las operaciones no lineales sobre datos de naturaleza binaria disponer de un autómata con reglas basadas en una tabla de búsqueda.
- Alimentar datos y configuración a través de una interface serial con el PC.

Con base en lo anterior y después de un proceso de evaluación de métodos de implementación de los autómatas, finalmente se determinó que la estrategia para el desarrollo, implementación y validación de los autómatas sobre la FPGA obedeciera a:

- Diseño e implementación de un autómata en dos capas (dos autómatas)
- Implementar un autómata con vecinos Moore de radio r=1 y de 256 estados por celda para operaciones lineales sobre cada celda.
- Implementar un autómata vecinos Moore de radio r=1, de dos estados para operaciones no lineales sobre datos binarios.
- Implementar los dos autómatas de tal forma que empleen la misma plataforma de comunicaciones para ahorrar recursos de hardware.
- Diseñar una máquina de control que controle el proceso de los dos autómatas.
- Diseñar los autómatas para que se reconfiguren (sin volver a programar la FPGA) cada vez que el usuario lo requiera.
- Emplear una vecindad de Moore de r=1, en razón a que con dicha región el número de celdas es 9, mientras que con r=2 se requiere de 25 celdas, ello llevaría a un hardware muy complejo y con un consumo muy alto de recursos.
- Implementar máscaras de orden superior (5X5) operando en cascada dos máscaras de 3X3.
- Reconfigurar los autómatas a partir de máscaras o arreglos que se depositan en la memoria de cada autómata y pueden ser actualizadas frecuentemente.
- La máscara del autómata lineal corresponde a filtros espaciales de 3X3 celdas.
- Para el autómata no lineal la regla corresponde a una tabla de búsqueda.
- La operación no lineal (ecualización) se implementa como una tabla de búsqueda en el mismo proceso de entrada serial de datos.

Con el dispositivo construido a partir de las anteriores estrategias se puede validar el funcionamiento del autómata realizando operaciones de filtrado y detección de bordes en imágenes, ya que ello requiere operaciones de alto cómputo entre pixeles tanto a nivel lineal como no lineal.

La evolución del autómata se realizó sobre una plataforma basada en PC (como se estableció en el anteproyecto), ello significa que las máscaras de los autómatas (filtros y tabla de búsqueda) son el resultado del proceso de evolución de la máquina genética.

En el proceso de diseño de la Máquina Genética, hubo que evaluar diferentes opciones para el proceso de evolución y evaluación de aptitud. Se debió analizar las ventajas de trabajar en el dominio del tiempo o en el dominio de la frecuencia para la evolución de los filtros, así también se evaluó los métodos para evolución de filtros y los mecanismos para realizar filtrado no lineal. En consecuencia se consideró importante:

- Evolucionar la máquina con base en el cruce de filtros y la mutación de una operación no lineal a nivel de pixel.
- Emplear una operación no línea para el mapeo de los valores de los pixeles de las imágenes.
- Realizar el filtrado en el dominio de frecuencia, ya que es más eficiente que en el dominio del tiempo a través de la convolución.
- Emplear una transformada rápida para el proceso de filtrado.
- Para ser compatibles con el autómata sobre la FPGA emplear máscaras de 3x3 pixeles.

A partir de las consideraciones expuestas, y de la evaluación de diferentes posibilidades para la máquina genética se determinó establecer las siguientes estrategias:

- Emplear el filtrado y el ajuste del histograma para el proceso de evolución ya que permite combinar operaciones lineales y no lineales.
- Construir los cromosomas de la población a partir de dos máscaras de convolución de 3X3 y un patrón de ecualización de histograma.
- Emplear dos máscaras de 3x3 para poder emular máscaras de 5x5.
- Emplear ecualización de pixeles y filtrado para la emulación de filtrado no lineal.
- Generar la función de ecualización de histograma a partir de polinomios de Lagrange por medio de la matriz de Vandermonde.
- Emplear el histograma de las imágenes como la información para el cálculo de la función de aptitud.
- Construir una función de aptitud que mida contraste de la imagen.
- Realizar el proceso de filtrado en espectro y no por medio de la calculo directo de la convolución para reducir el número de operaciones.

Con base en lo expuesto, la Máquina Genética evoluciona filtros y ecualizadores hasta obtener un par de filtros y un ecualizador que aumente el contraste de la imagen en un número dado de iteraciones. La evolución se basa en el cruce y mutación de los cromosomas y se evalúa su aptitud midiendo el contraste de la imagen ecualizada y filtrada por medio de la FFT. Los filtros y el ecualizador de histograma se transmiten al autómata en la FPGA, en el cual se realiza el proceso de filtrado ecualización y además se realiza la detección de bordes empleando la medición de información direccional y el autómata no lineal.

6.1 METODOLOGÍA

Cada objetivo específico se desarrolló a partir de una serie de actividades como las expuestas en el anteproyecto en el planteamiento metodológico. El resultado de cada conjunto de actividades corresponde a un conjunto de productos para el cumplimiento de los objetivos.

El marco metodológico corresponde al mostrado en la siguiente tabla:

Tabla 1. Desarrollo Metodológico

Objetivo	Actividad	Producto
Diseñar y validar el modelo matemático de los autómatas celulares en una y dos dimensiones.	• Determinar las herramientas necesarias de la Topología matemática, para comprender los modelos de los autómatas celulares.	Modelo matemático de un autómata celular general
	• Generación de autómatas celulares de prueba en una dimensión	Modelo de un autómata en una dimensión
	• Generación de autómatas celulares de prueba en dos dimensiones	Autómata celular de dos dimensiones implementado sobre una herramienta de simulación en PC.
Diseñar e implementar la máquina genética para evolucionar los	• Diseño de la máquina genética sobre una plataforma basada en PC.	Máquina genética básica

autómatas celulares sobre una plataforma basada en PC.	• Diseño de un autómata celular	Modelo funcional y matemático del autómata celular.
	• Evolución del autómata por medio de la máquina genética	Autómata celular final para ser implementado en la FPGA
	• Pruebas de desempeño de prototipo con el autómata de juego de la vida.	Resultados de las pruebas
Implementar los autómatas celulares reconfigurables sobre una herramienta de desarrollo basada en FPGA, sin reprogramar la FPGA.	• Implementación en VHDL de la interface entre el PC y la FPGA para la transferencia de configuración y datos.	Interface entre el PC y la herramienta de desarrollo.
	• Implementación en VHDL y sobre la FPGA del autómata celular de tal forma que las reglas se puedan reconfigurar sin la necesidad de reprogramar la FPGA.	Autómata celular evolucionado reconfigurable sobre la FPGA.
Evaluar el desempeño de la implementación en hardware mediante una aplicación orientada al proceso de datos en dos dimensiones.	• Diseño de la función de aptitud (fitness), para la generación de reglas para la implementación de filtros de suaviza miento y detección de bordes en una matriz que representa una imagen.	Máquina genética para la generación de reglas para autómatas celulares para el filtrado de imágenes.

	• Evaluación del autómata celular, mediante la prueba con la implementación de filtros de suaviza miento y detección de bordes	Comparación de resultados con los obtenidos con las técnicas tradicionales basadas en PC, en cuanto a tiempo de ejecución y eficacia en el proceso de filtrado.

Con base en la tabla anterior se expone a continuación los desarrollos y productos de cada actividad en el marco de cada objetivo específico.

6.2 DESARROLLO DEL PRIMER OBJETIVO:

"Diseñar y validar el modelo matemático de los autómatas celulares en una y dos dimensiones".

6.2.1 HERRAMIENTAS DE LA TOPOLOGÍA – MODELO MATEMATICO

"Determinar las herramientas necesarias de la Topología matemática, para comprender los modelos de los autómatas celulares".

Un autómata consiste básicamente en un arreglo de celdas en una o varias dimensiones, donde el estado de cada celda se puede representar mediante un conjunto de valores, por ejemplo un anillo o un campo sobre los enteros; en el caso más simple el estado puede ser binario. La dinámica de las celdas y por ende el del autómata está gobernada por una función de transición que actualiza cada celda en relación a los valores de los estados de un conjunto de celdas vecinas contenidas en una región.

Definición: Un autómata celular uniforme y finito se define matemáticamente como (Kari, Spring - 2011),(Sekanina L. , 2004):

Ecuación 4: Autómata celular

$$A = d, S, N, f, c_0, B$$

Fuente: propia

En donde:

$d \in$ N, Es la dimensión del autómata

S, es el conjunto de estados (finito) en los cuales puede estar el autómata

$N = (\vec{n}_1, \vec{n}_2, \ldots, \vec{n}_m)$, Es el vector de vecindades con m vecinos

$f : S^m \to S$, Es la función de transición local o regla del autómata

c_0, Es la configuración inicial del autómata

B, es el conjunto de condiciones de frontera

En la definición y notación anterior, no se especifica la evolución en el tiempo del autómata, ni las pautas para la implementación de los autómatas ya sea en sistemas de software o de hardware, tampoco se determina que celdas componen la vecindad de cada una de las celdas. Por lo anterior, como primera medida, para hacer evidente la transición en el tiempo, se define una configuración (***c***) del autómata celular como el estado en el que se encuentran todas las celdas del autómata en un momento dado.

Ecuación 5. Estados

$$c: \mathbb{Z}^d \longrightarrow S$$

Fuente: propia

El estado de cada celda $\vec{n} \in \mathbb{Z}^d$ se denota como, $c(\vec{n})$. La aplicación iterada de la regla del autómata genera un cambio global en la configuración del mismo, desde una configuración ***c*** a una configuración ***c'*** de la siguiente manera:

Ecuación 6: transición de estados

$$c'(\vec{n}) = f[c(\vec{n} + \overrightarrow{n_1}), c(\vec{n} + \overrightarrow{n_2}), \ldots, c(\vec{n} + \overrightarrow{n_m})]$$

Fuente: Propia

Dicha transformación, a saber, el ir de una configuración dada a la siguiente se logra mediante una función de transición global ***G*** y se define como una función de la siguiente forma:

Ecuación: 7. Función de transición

$$G: S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$$

Fuente: propia

La aplicación repetida de la regla del autómata, en últimas la aplicación repetida de la función de transición global ***G*** es lo que genera la evolución, así para n iteraciones se tiene:

Ecuación 8. Evolución de la configuración

$$c \longmapsto G(c) \longmapsto G^2(c) \longmapsto G^3(c) \longmapsto \cdots \longmapsto G^n(c)$$

Fuente: propia

En donde ***c*** se denomina condición inicial del autómata celular.

Vecindad: En cuanto a los vecinos de cada celda del autómata, clásicamente se definen dos tipos de vecindades, vecinos Newmann y vecinos Moore. La vecindad Moore $d - dimensional$, denotada como $\boldsymbol{M_r^d}$ con dimensión $\boldsymbol{d}$ y radio $\boldsymbol{r}$ se define como:

Ecuación 9. Vecindad Moore

$$M_r^d = \{(k_1, k_2, \dots, k_d) \in \mathbb{Z}^d \; tales \; que \; |k_i| \leq r \; para \; i = 1,2, \dots, d\}$$

Fuente: propia

El vector de vecinos Moore contiene $(2r+1)^d$ elementos.

La vecindad Newmann $d - dimensional$, denotada como $\boldsymbol{V_r^d}$ con dimensión $\boldsymbol{d}$ y radio $\boldsymbol{r}$ se define como:

Ecuación 10: Vecindad Newmann

$$V_r^d = \left\{(k_1, k_2, .., k_d) \; \in \mathbb{Z}^d \; tales \; que \; \sum_{i=1}^{d} |k_i| \leq r\right\}$$

Fuente: propia

Para el caso bidimensional, en la figura se muestran los dos tipos de vecindad: (a) vecindad de Moore y (b) vecindad Newmann:

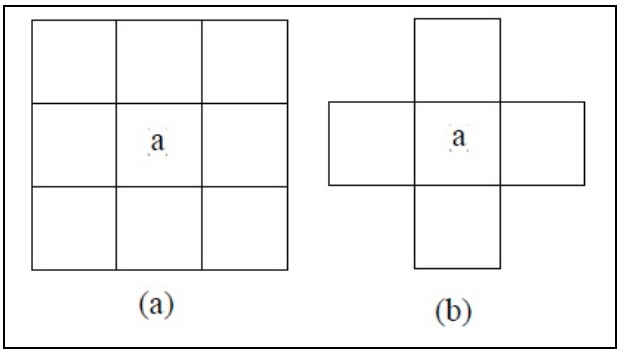

Figura 5. Vecindades Clásicas en los Autómatas Celulares
Fuente. Autor.

Un último aspecto fundamental a la hora de definir los autómatas con fines de implementación es que en general, o más exactamente, matemáticamente, los autómatas celulares se consideran de extensión infinita en sus dimensiones, este hecho resulta evidente de la definición al ver que el espacio celular es el conjunto de los enteros $\mathbb{Z}$, pero dicha condición es difícilmente lograda en implementaciones físicas de cualquier tipo, por lo tanto, para lidiar con esta situación entran en juego las denominadas condiciones de frontera, las cuales de algún modo relatan que sucede, o que hacer con el autómata en sus fronteras.

6.2.2 GENERACION DE AUTOMATAS CELULARES DE PRUEBA EN UNA DIMENSION

Se desarrolló con base en el modelo matemático un autómata muy simple de dos estados y en una dimensión sobre la plataforma de Matlab (figura 6); el autómata genera una imagen fractal conocida como triangulo de Sierpinski (figura 7).La evolución del autómata transcurre en el tiempo, lo que genera una imagen a partir de los estados de las celdas para cada instante. El tiempo transcurre hacia abajo en la figura.

```
H=ones(256,256);                    % estados iniciales de la celda
H(1,2: 110)=0;                      % semilla inicial del autómata

for j=1:254                         % número de pasos de evolución
for i=253:-1:2                      % evolución celda por celda
      v=H(j,i-1:i+1);
      PV=dot(v,[4 2 1]);
switch PV                     % reglas de evolución
case 0 %111 0
            H(j+1,i)=1;
case 1 %110 1
            H(j+1,i)=0;
case 2 %101 1
            H(j+1,i)=1;
case 3 %100 0
            H(j+1,i)=0;
case 4 %011 0
            H(j+1,i)=0;
case 5 %010 1
            H(j+1,i)=1;
case 6 %001 1
            H(j+1,i)=0;
case 7 %000 0
            H(j+1,i)=1;
end
end
end
```

Figura 6. Autómata de una dimensión
Fuente: propia

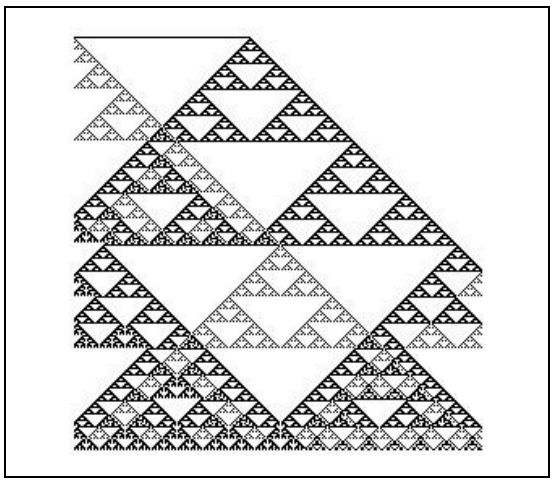

Figura 7. Triangulo de Sierpinski
Fuente: propia

6.2.3 GENERACION DE AUTOMATAS CELULARES DE PRUEBA EN DOS DIMENSIONES

Para la implementación de un autómata en dos dimensiones se optó por implementar el autómata conocido como bola8. El código se encuentra en el anexo 1 y muestra la forma de implementar el autómata. En la figura 8 se observa el proceso de evolución.

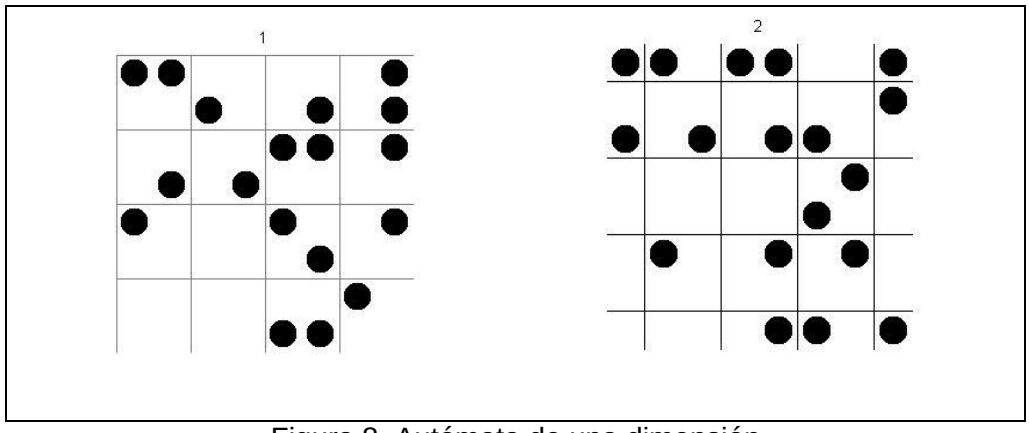

Figura 8. Autómata de una dimensión
Fuente: propia

6.3 DESARROLLO DEL SEGUNDO OBJETIVO:

"Diseñar e implementar la máquina genética para evolucionar los autómatas celulares sobre una plataforma basada en PC."

6.3.1 DISEÑO DE LA MÁQUINAGENÉTICA SOBRE UNA PLATAFORMA BASADA EN PC

Para el diseño e implementación de la máquina genética se siguieron las estrategias expuestas al inicio del numeral 6, y su composición se puede representar mediante el diagrama en bloques mostrado en la figura 9. A continuación se explicará cada componente.

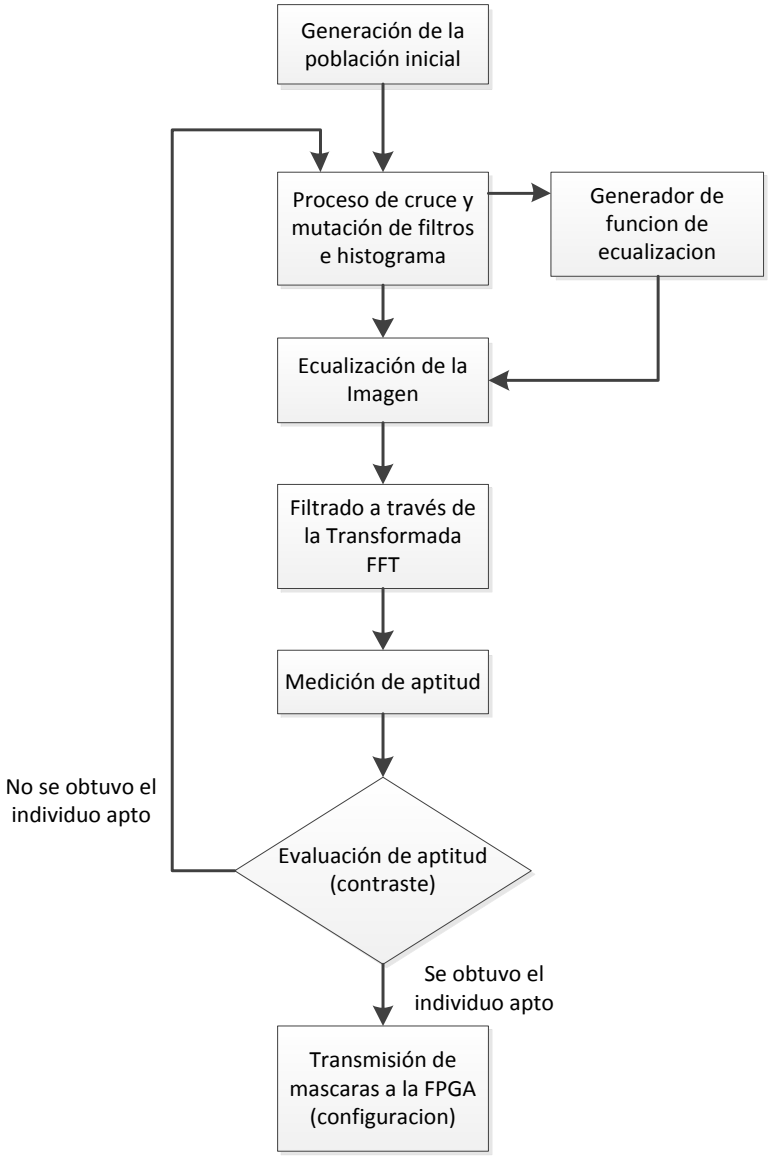

Figura 9. Máquina Genética
Fuente: propia

6.3.1.1 GENERACION DE LA POBLACIÓN INICIAL

La población inicial se construye a partir de cinco filtros básicos 3x3: pasabajos, pasa todo, y pasa altos; los filtros corresponden a:

Ecuación 11. Filtros

$$\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

$$\begin{bmatrix}-1 & 0 & -1\\ 0 & 4 & 0\\ -1 & 0 & -1\end{bmatrix}\begin{bmatrix}-1 & -1 & -1\\ -1 & -8 & -1\\ -1 & -1 & -1\end{bmatrix}$$

$$\begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}$$

Fuente: propia

Se mezclan y se producen seis cromosomas iníciales con la siguiente estructura:

Ecuación 12. Cromosoma

$$\begin{bmatrix}a & b & c\\ d & e & f\\ g & h & k\end{bmatrix} \quad \begin{bmatrix}l & m & n\\ o & p & q\\ r & s & u\end{bmatrix} \quad [y_1 \quad y_2 \quad y_3]$$

Fuente: propia

Los dos primeros campos obedecen al primer y segundo filtro (f_1 y f_2), mientras que el último campo corresponde al de las semillas para el ecualizador.

Las semillas consisten básicamente en cinco valores en la abscisa vertical que se corresponde con cinco valores fijos en la abscisa horizontal: [0 1/4 2/4 3/4 1].A partir de los puntos descritos por los valores verticales se generan aleatoriamente curva de ecualización en las que los puntos correspondientes a los extremos ((1,0), (1,1)) permanecen fijos. El resultado da como resultado curvas como las mostradas en la figura 10.

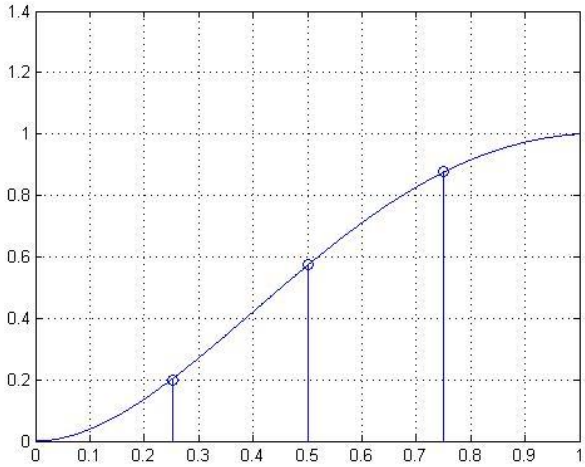

Figura 10. Ecualizador
Fuente: propia

El mecanismo para generar la curva obedece al procedimiento de generación del polinomio de Lagrange:

Ecuación 13. Sistema lineal

$$\begin{array}{l} a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_2 x_0^n = y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_2 x_1^n = y_1 \\ a_0 + a_1 x_2 + a_2 x_2^2 + \cdots + a_2 x_2^n = y_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_2 x_n^n = y_n \end{array}$$

Fuente: propia

Los valores en x (abscisa horizontal) son fijos y corresponden a [0 1/4 2/4 3/4 1], para los valores en y (abscisa horizontal) son fijos los extremos y variables los otros tres: [0 r_1 r_2 $r_3$1].

Con base en el sistema de ecuaciones se obtiene la siguiente representación:

Ecuación14. Representación matricial

$$\begin{bmatrix} 1 & x_o x_0^2 & \ldots . & x_0^n \\ 1 & x_1 x_1^2 & \ldots . & x_1^n \\ & \cdots & & \\ 1 & x_n x_n^2 & \ldots . & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \cdots \\ a_n \end{bmatrix}$$

Fuente: propia

La matriz de los valores de x se conoce con el nombre de Matriz de Vandermonde.

Como se tienen los valores de x (fijos) y los de y (variables excepto los dos extremos), se despeja la ecuación matricial para obtener los coeficientes del

polinomio $[a_0 a_1 a_2 \ \ldots \ a_n]$ (en este caso n=4), ello lleva a obtener el polinomio generador de la ecualización:

Ecuación 15. Representación matricial

$$y = a_0 \quad a_1 x \quad + a_2 x^2 \quad + a_3 x^3 \quad + a_4 x^4$$

Fuente: propia

El algoritmo para generar la población inicial y la inversa de la matriz de Vandermonde (MI) se muestra a continuación:

```
function  [c,MI] = poblacion_0(J)
% creación de la población inicial
% se crean los filtros básicos
% la imagen entra en J para el cálculo de la función de aptitud
a=0;                                % se cargan los 5 puntos en x para el histograma
b=1/4;
g=1/2;
d=3/4;
e=1;
M=[1 a a^2 a^3 a^4; 1 b b^2 b^3 b^4; 1 g g^2 g^3 g^4; 1 d d^2 d^3 d^4; 1 e e^2 e^3 e^4 ];
MI=M^(-1);                          % se obtiene la matriz para el polinomio-histograma

h=zeros(3,3,5);                     % banco de filtros-----------------------------
h(:,:,1)=[1 1 1; 1 1 1; 1 1 1];
h(:,:,2)=[1 2 1;2 4 2; 1 2 1];
h(:,:,3)=[0 -1 0; -1 4 -1; 0 -1 0];
h(:,:,4)=[-1 -1 -1; -1 8 1; -1 -1 -1];
h(:,:,5)=[0 0 0; 0 1 0; 0 0 0];

for i=1:4                           % normalización de los filtros----------
   m=abs(mean(mean(h(:,:,i)))*9);
if m==0
      m=1/16;
end
    h(:,:,i)=h(:,:,i)*(1/m);
end

%población inicial-----------------------------------------------------------------
c(1).f1=h(:,:,1);
c(1).f2=h(:,:,5);
c(1).ht=[1/4 2/4 3/4];
c(1).ap=0;

c(2).f1=h(:,:,2);
c(2).f2=h(:,:,5);
c(2).ht=[1/4 2/4 3/4];
```

```
c(2).ap=0;

c(3).f1=h(:,:,3);
c(3).f2=h(:,:,5);
c(3).ht=[1/4 2/4 3/4];
c(3).ap=0;

c(4).f1=h(:,:,4);
c(4).f2=h(:,:,5);
c(4).ht=[1/4 2/4 3/4];
c(4).ap=0;

c(5).f1=h(:,:,1);
c(5).f2=h(:,:,3);
c(5).ht=[1/4 2/4 3/4];
c(5).ap=0;

c(6).f1=h(:,:,2);
c(6).f2=h(:,:,4);
c(6).ht=[1/4 2/4 3/4];
c(6).ap=0;

HJ=imhist(J);

% cálculo de la función de aptitud con varianza (inversa de contraste)
for k=1:6
    h1=c(k).f1;                          % se lee el filtro f1
    h2=c(k).f2;                          % se lee el filtro f2
    y=conv_2DD(h1,J);                    % se aplica el primer filtro
    x=conv_2DD(h2,y);                    % se aplica el segundo filtro
    HX=imhist(x);
    [apt,vj]= aptitud(HX,HJ);            % se aplica la función de aptitud
    c(k).ap=apt;
end
```

Figura 11. Generador población inicial
Fuente: propia

6.3.1.2 CRUCE Y MUTACIÓN

El procedimiento de cruce entre padres emplea las siguientes dos máscaras de cruce:

Ecuación 16. Máscaras de cruce

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Fuente: propia

La selección de las mascaras de cruce se origino en la necesidad de que las dos mascaras sean ortogonales y además no sean direccionales; de otra parte el tamaño de la máscara se limito a una estructura de 3X3 para reducir el tamaño del autómata celular implementado sobre la FPGA.

El proceso de cruce consiste en: se escoge un par de padres al azar, al primer padre se le multiplica elemento a elemento el primer filtro por la primera máscara obteniéndose un filtro g_1, la segunda máscara se emplea para el primer filtro del segundo padre y se obtiene un filtro g_2. Una vez se obtienen los dos filtros se genera un número aleatorio p (entre 0 y 1) y se realiza la siguiente operación:

Ecuación 17. Filtro hijo

$$h_1 = g_1 * P + g_2 * (1 - p)$$

Fuente: propia

El filtro obtenido es el resultado del cruce entre los dos primeros filtros de los dos padres, el procedimiento para los dos segundos filtros de los dos padres es el mismo. El proceso de mutación consiste en mover aleatoriamente el valor del centro de los dos filtros del hijo.

Padre 1	**[f_1]**	**[f_2]**	$[y_1 \quad y_2 \quad y_3]$
Padre 2	**[f_1]**	**[f_2]**	$[y_1 \quad y_2 \quad y_3]$
hijo	**[h_1]**	**[h_2]**	$[r_1 \quad r_2 \quad r_3]$

Figura 12. Cruce. Fuente: propia

Los tres puntos $[r_1 \quad r_2 \quad r_3]$se obtienen como un promedio de las semillas $[y_1 \quad y_2 \quad y_3]$de los dos padres, el proceso de mutación consiste en mover aleatoriamente uno de los tres valores de la semilla.

El algoritmo de cruce y mutación se muestra a continuación:

```
function d=cruce_mut(c)
nh=zeros(2,3);                  % matriz para guardar histogramas
m1=[0 1 0; 1 1 1; 0 1 0];       % máscara de cruce 1
m2=[1 0 1; 0 0 0; 1 0 1];       % máscara de cruce 2
rc1=randi(6);                   % se selecciona el primer padre
rc2=randi(6);                   % se selecciona el segundo padre
```

```
p=rand;                                  % se selecciona al azar el porcentaje del padre
f1= c(rc1).f1.*m1*p + c(rc2).f1.*m2*(1-p);
f1(2,2)= f1(2,2) -0.5 +0.5*rand;         % mutación del centro de f1
   m=abs(mean(mean(f1))*9);
if m==0
      m=1/16;
end
d.f1=f1*(1/m);
p=rand;                                  % se selecciona al azar el porcentaje del padre
f2= c(rc1).f2.*m1*p + c(rc2).f2.*m2*(1-p);
f2(2,2)= f2(2,2) -0.5 +0.5*rand;         % mutación del centro de f2
   m=abs(mean(mean(f2))*9);
if m==0
      m=1/16;
end
d.f2=f2*(1/m);
nh(1,:)=c(rc1).ht;
nh(2,:)=c(rc2).ht;                       % se tienen los dos histogramas
nht=mean(nh);                            % se obtiene el promedio de las semillas
eh=randi(3);
gh=nht(eh);
if abs(0.5 - gh)<0.2                     % se realiza la mutación sobre las semillas
nht(eh)=gh -0.2 +0.4*rand;
elseif (0.5-gh)>0
nht(eh)=0.3 -0.2 +0.4*rand;
else
nht(eh)=0.7 -0.2 +0.4*rand;
end
d.ht=nht;
```

Figura 13. Algoritmo de cruce
Fuente: propia

6.3.1.3 ECUALIZACIÓN

El procedimiento de ecualización consiste básicamente en una mapeo pixel por pixel a través de la ecuación obtenido mediante el polinomio de Lagrange:

Ecuación 18: Polinomio de Lagrange

$$y = a_0 \quad a_1 x \quad + a_2 x^2 \quad + a_3 x^3 \quad + a_4 x^4$$

Fuente: propia

El código para este procedimiento es:

```
function II=ecu_im(J,A)
[N,M]=size(J);
II=zeros(N,M);
for i=1:N
for k=1:M
      r=J(i,k);
```

```
        p=[1 r r^2 r^3 r^4];
        II(i,k)=dot(p,A');
end
end
```

Figura 14. Código de ecualización
Fuente: propia

6.3.1.4 FILTRADO A TRAVEZ DE LA TRANSFORMADA FFT

Desde el punto de vista del cómputo, el número de operaciones necesarias para efectuar el filtrado en espacio, es mayor que en el filtrado en frecuencia cuando se emplea una transformada rápida como la FFT, sin embargo en el escenario de un algoritmo evolutivo el número de veces que se efectúa la transformada puede ser muy alto, elevando los tiempos de convergencia a valores muy altos.

La transformada rápida de Fourier se opera sobre los complejos, ello implica que en la programación se deben manipular estructuras con dos campos, uno para los reales y otro para los imaginarios. De otra parte, si se logra operar sobre un campo de enteros se reduce la complejidad del algoritmo de la transformada, además que se consigue una mayor precisión que en la FFT.

Por lo anterior se intento desarrollar una transformada en teoría de números sobre un campo de Galois de 255 valores, para ello se hizo indispensable desarrollar las siguientes actividades:

- Desarrollar un algoritmo eficiente para la FFT en una dimensión
- Desarrollar un algoritmo para la construcción de un campo de Galois de 255 valores.
- Construir el algoritmo NTT con base en la estructura de la FFT de una dimensión y con las raíces ubicadas en el campo de Galois.

Los algoritmos para la FFT se encuentran en los archivos del anexo 3, y sirven como marco para el diseño de la NTT.

El generador del campo de Galois construye el campo con base en el siguiente polinomio primitivo:

Ecuación 19. Polinomio característico

$$1 + x^2 + x^3 + x^4 + x^8$$

Fuente propia

El algoritmo implementado sobre Matlab se encuentra en el anexo 2.

En cuanto a la transformada NTT sobre el campo generado por el algoritmo mostrado en el anexo 2, se empleo el algoritmo propio de la transformada FFT (anexo 3), empero los resultados evidenciaron los siguientes inconvenientes:

- El proceso de la transformación sobre la NTT, requiere que el número de puntos sea divisor de 255 es decir de 2^n -1, pero el algoritmo desarrollado se concibió para una FFT RADIX2 es decir que el tamaño de la muestra es de naturaleza par y por ende no es divisor de 255.
- De otra parte según lo expuesto por Trifonov y Fedorenko [28] no siempre es posible obtener la NTT para todos los campos de Galois.

Sin embargo los trabajos realizados por Trifonov y Fedorenko [28] muestran que la cantidad de cómputo se puede reducir como lo evidencia la siguiente tabla de los autores:

Tabla 2: comparación algoritmos para el cálculo de la FFT

Complexity of some FFT algorithms

	Horner's method		Goertzel's algorithm		Algorithms from [7, 8]		Zakharova's method [10]		Suggested method	
n	N_{mul}	N_{add}	N_{mul}	N_{add}	N_{mul}	N_{add}	N_{mul}	N_{add}	N_{mul}	N_{add}
7	36	42	12	42	9	35	6	26	6	25
15	196	210	38	210	20	70	16	100	16	77
31	900	930	120	930	108	645	60	388	54	315
63	3844	3906	282	3906	158	623	97	952	97	805
127	15876	16002	756	16002	594	5770	468	3737	216	2780
255	64516	64770	1718	64770	1225	4715	646	35503	586	7919
511	260100	260610	4044	260610	4374	—	—	—	1014	26643

fuente: Trifonov y Fedorenko [28]

En la tabla *Nmul* representa el número de multiplicadores y *Nadd* el número de sumadores.

6.3.1.5 MEDICIÓN DE APTITUD

La función para evaluar la aptitud de cada uno de los individuos de la población, corresponde a la medición de contraste. Una imagen con alto contraste tiene un histograma más uniforme que otra con inferior contraste. La herramienta empleada fue la de varianza, un histograma con una varianza baja representa un contraste alto, y por lo contrario una imagen con varianza alta en su histograma significa un contraste bajo.

El código para la medición de aptitud se encuentra en el anexo 4.

6.3.2 DISEÑO DE UN AUTÓMATA CELULAR

Se implementó un autómata de Wolfram [5] para la detección de bordes, el cual consiste básicamente en una tabla de lookup. El código se encuentra en el anexo 5.El autómata de Wolfram se empleó en la detección de bordes mediante el siguiente algoritmo:

```
%% Autómata celular para suavizado de imágenes sin alteración de bordes %%
% -- Lee la imagen -- %
I=imread('einstein.jpg');
I=rgb2gray(I);
I=double(I);
z1=I;
% ------------------------------------- %
% -- Iteraciones del Autómata celular -- %
% ------------------------------------- %
n=4;
for i=1:n
    i
% Halla la matriz de medida de orientación de la información
    M=orient_info(I,3);
                                % Halla la matriz de marca, binariza M
    [Var Th]=umbral(M);         % Por medio de cálculos de varianza sobre la matriz M
          % se determina un umbral con un margen de variación del
          % 10% a partir de cual se binariza la imagen.
    B=M./Th;
    BB=B;
    B=im2bw(B);
    B=double(B);                % Matriz Binarizada
                                % Aplica un filtro de Promedio a la imagen original
    h=(1/9).*ones(3,3);         % Máscara del filtro de suavizado
    [a,b]=size(h);
    Ip=conv_2D(h,I);            % Filtra
    [A,C]=size(Ip);
    Ip=Ip(((a-1)/2)+1:A-((a-1)/2),((b-1)/2)+1:C-((b-1)/2));    % Ajusta dimensiones

% aplicación de filtrado a imagen sin bordes
    R=[0 0 0 0 1 1 1 1 1 0];
    B=automata(B,R,1);
    i1=I.*B;
    i2=Ip.*imcomplement(B);
    res=i1+i2;
    zm(:,:,i)=B;
    zi(:,:,i)=res;
    I=res;
end
```

Figura 15. Autómata de suavizado Fuente: Propia

6.3.3 EVOLUCIÓN DEL AUTÓMATA POR MEDIO DE LA MÁQUINA GENÉTICA

El proceso de evolución del autómata se realizó tal como se muestra en el diagrama de bloque de la figura 9, es decir se siguen los siguientes 6 pasos:

- Lectura de la imagen
- Creación de la población inicial
- Cruce y mutación de individuos
- Ecualización de imagen
- Filtrado
- Calculo de la aptitud de los individuos

El código para la máquina genética se muestra en la figura 16; en él se evidencia los 6 pasos descritos. El criterio para la detención del proceso consiste en la medición de aptitud, cuando la varianza medida del histograma resultante está por debajo del 50% de la varianza del histograma de la imagen original. El umbral para la parada del algoritmo de evolución se puede ajustar de acuerdo al tiempo disponible para el proceso.

```
% se lee la imagen de entrada al proceso
I=imread('einstein.jpg');
I=rgb2gray(I);
J=double(I)*(1/256);
[N,M]=size(J);
HJ=imhist(J);
[c,MI]=poblacion_0(J);                  % generación de la población 0
for k=1:100                             %  número máximo de evoluciones
      d=cruce_mut(c);                   % se obtiene el nuevo organismo
      sht=d.ht;                         % se lee las semillas del histograma
      A=gen_pol(sht,MI);                % se generan los coeficientes del polinomio
      II=ecu_im(J,A);                   % se ecualiza la imagen J con el polinomio
      h1=d.f1;                          % se lee el filtro f1
      h2=d.f2;                          % se lee el filtro f2
      y=conv_2DD(h1,II);                % se aplica el primer filtro
      x=conv_2DD(h2,y);                 % se aplica el segundo filtro
      HX=imhist(x);
      [apt,vj]= aptitud(HX,HJ);         % se aplica la función de aptitud
      apto=zeros(1,6);
for q=1:6
          apto(q)=c(q).ap;
end
      [minap,IN] = max(apto);
if apt<minap
          d.ap=apt;
          c(IN)=d;
end
if apt<0.5*vj                           % criterio de convergencia de la aptitud
```

```
break                          % se detiene cuando la aptitud es inferior al 50% del
end                            % valor de la varianza del histograma inicial
end

for q=1:6
      apto(q)=c(q).ap;
end
[minap,IN] = min(apto);
 sht=c(IN).ht;            % se lee las semillas del histograma
 A=gen_pol(sht,MI);                  % se generan lo coeficientes del polinomio
 II=ecu_im(J,A);          % se ecualiza la imagen J con el polinomio
 h1=c(IN).f1;      % se lee el filtro f1
 h2=c(IN).f2;                        % se lee el filtro f2
 y=conv_2DD(h1,II);                  % se aplica el primer filtro
 x=conv_2DD(h2,y);                   % se aplica el segundo filtro
 HX=imhist(x);                       % se obtiene el histograma de la imagen
 c(IN).ap                            % se obtiene el valor de aptitud
```

Figura 16. Algoritmo de evolución.
Fuente: propia

6.3.4 PRUEBAS DE DESEMPEÑO DEL PROTOTIPO CON EL JUEGO DE LA VIDA

Las pruebas se realizaron directamente con el autómata de Wolfram en lugar del juego de la vida, ya que el autómata detecta bordes y entrega resultandos más coherentes que los obtenidos con el juego de la vida. En la figura 17 se observa a la izquierda la imagen sin filtrar, en el centro la imagen resultante del proceso de filtrado no lineal, y por último en la derecha se observa el resultado del proceso de detección de bordes del autómata.

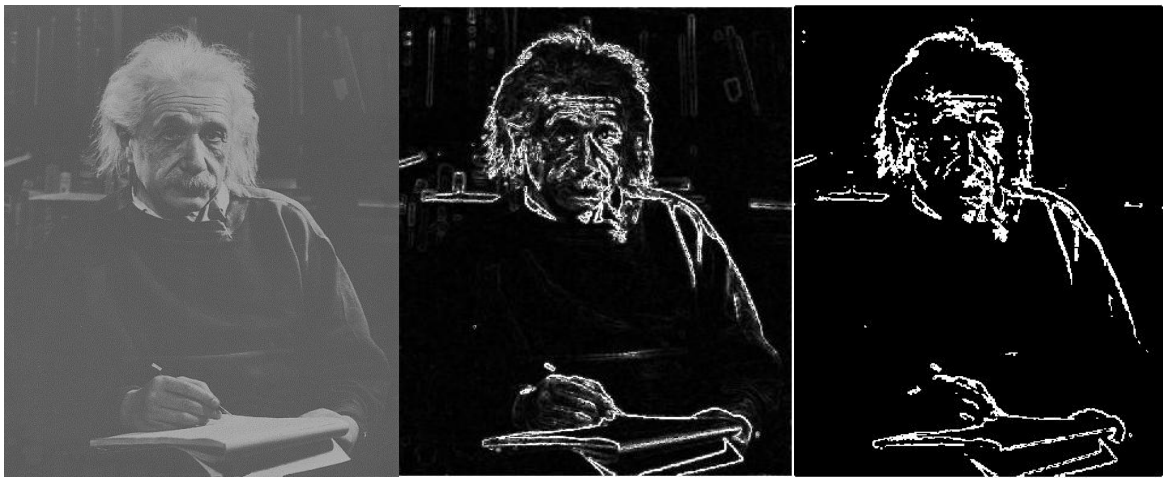

Figura 17. Prueba del autómata de filtrado. Fuente: propia

6.4 DESARROLLO DEL TERCER OBJETIVO

“Implementar los autómatas celulares reconfigurables sobre una herramienta de desarrollo basada en FPGA, sin reprogramar la FPGA.”

El diagrama general del sistema se muestra en la figura 18. Se compone básicamente de tres componentes:

- El autómata celular con una capa lineal y la capa no lineal.
- La unidad o máquina de control.
- La interfaz serial (UART) con el PC.

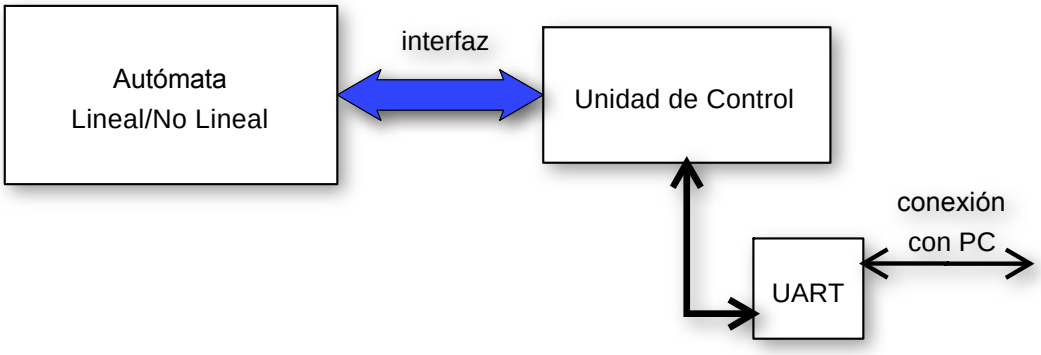

Figura 18. Diagrama en Bloques del Sistema. Fuente: propia

6.4.1 IMPLEMENTACIÓN EN VHDL DE LA INTERFACE ENTRE EL PC Y LA FPGA PARA LA TRANSFERENCIA DE CONFIGURACIÓN Y DATOS.

La implementación consiste básicamente en una UART sobre la FPGA para el proceso de descarga y subida de imágenes desde el PC, y para que se descarguen hacia la FPGA las configuraciones de los autómatas implementados.

Se empleo una UART para minimizar el consumo de hardware ya que el autómata lineal y el no lineal consumen la mayoría de recursos de la FPGA

El diseño de la UART se involucró directamente dentro del procesador NIOS el cual realiza el papel de la unidad de control, en razón que a partir del procesador NIOS se infiere directamente la interfaz sin necesidad de diseñar hardware adicional.

6.4.2 IMPLEMENTACIÓN EN VHDL Y SOBRE LA FPGA DEL AUTÓMATA CELULAR DE TAL FORMA QUE LAS REGLAS SE PUEDAN RECONFIGURAR SIN LA NECESIDAD DE REPROGRAMAR LA FPGA.

El autómata celular consiste de dos grandes módulos (figura 19): uno, el autómata celular en sí, que es básicamente un arreglo bidimensional de celdas idénticas (más las de condiciones de frontera) cuya operación en paralelo conforma un sistema de gran complejidad y versatilidad para la implementación de funciones tanto lineales como no lineales, que además ofrece paralelismo masivo; el segundo módulo corresponde al sistema de control que gobierna las acciones del autómata celular. Este último es implementado usando el procesador embebido NIOS II de Altera Corp.(Altera Corporation, 2009).

Los datos que ingresan al autómata desde el exterior (PC) lo hacen por medio de una UART implementada a partir del procesador NIOS, y de allí son enviados al autómata para su procesamiento.

El esquema básico del autómata celular se puede observar en la figura 19. La unidad de control implementada sobre el procesador se comunica con el autómata a través de cinco grupos de señales:

- Datos de entrada
- Datos de salida
- Dirección de escritura
- Control autómata lineal
- Control autómata no lineal

El autómata consiste de una matriz de 16 x 16 celdas, en razón a que en el consumo de recursos de la FPGA llego aproximadamente al 90 % lo que impidió implementar un autómata más grande; en el caso de los multiplicadores contenidos en la FPGA se emplearon en su totalidad.

Además de la matriz de 16X16 celdas fue necesario implementar 68 celdas adicionales para las condiciones de frontera en operaciones con mascaras de 3X3, lo que llevo a un autómata de 18 X18 celdas.

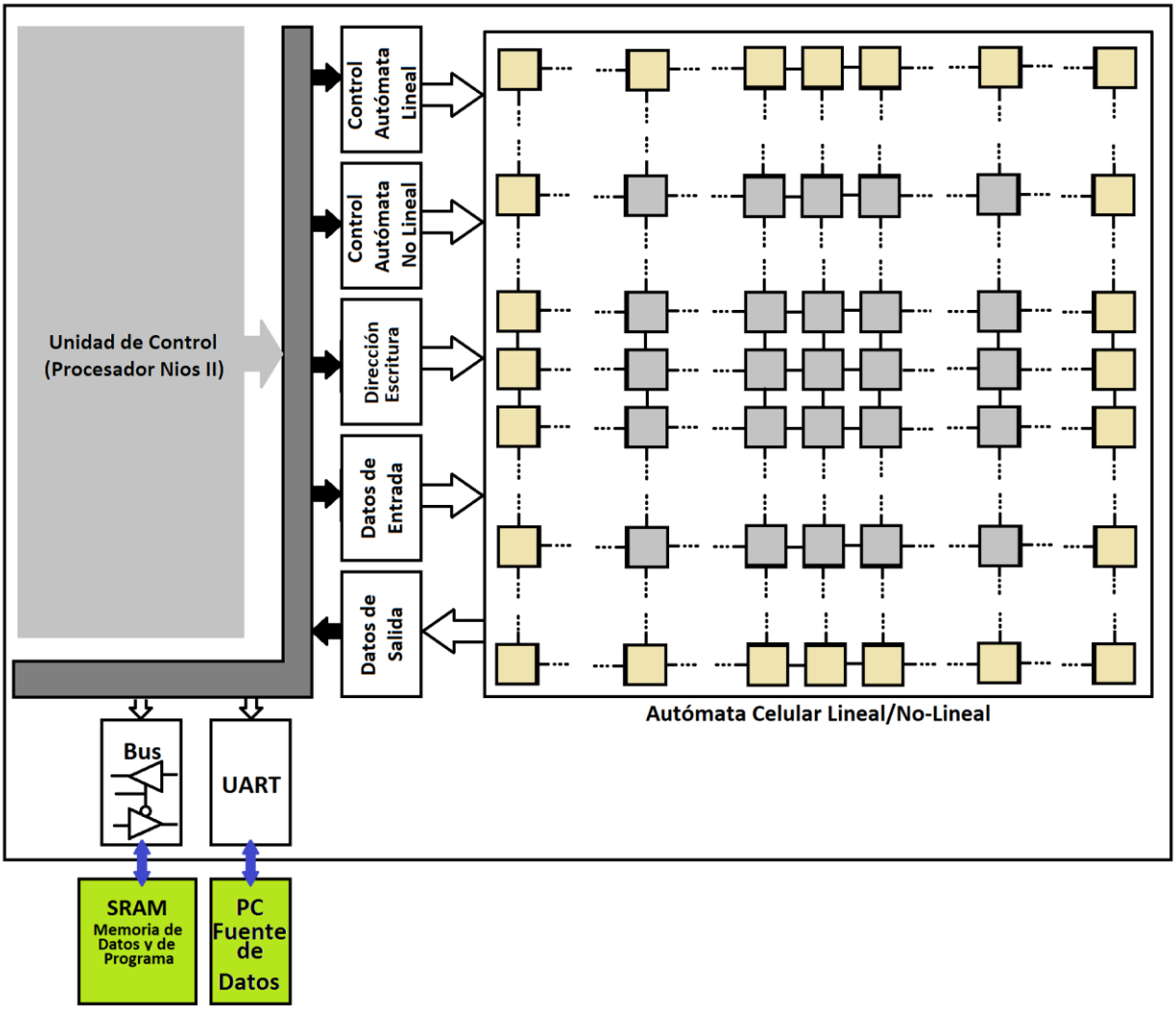

Figura 19. Autómata celular lineal/no lineal. Fuente: propia

6.4.2.1 DESCRIPCIÓN EN VHDL DEL AUTÓMATA CELULAR – CAPA LINEAL

La capa lineal del autómata celular realiza básicamente convolución en dos dimensiones entre los datos ingresados desde el exterior, en este caso los píxeles de la imagen, y diferentes máscaras dependiendo de la parte del proceso que se esté llevando a cabo. Las máscaras son descargadas desde el exterior (el PC) y almacenadas dentro de memoria presente en cada celda del autómata.

La herramienta en donde se lleva a cabo la implementación de los autómatas celulares es la FPGA Stratix II EP2S60 de Altera, cuya arquitectura a groso modo consiste de:

- Tablas de búsqueda adaptativas (ALUT)
- Bloques de memoria RAM distribuida

- Bloques de 512 Bits (M512)
- Bloques de 4096 bits (M4K)
- Bloques de 512Kbits (M-RAM)
- Bloques DSP
- Multiplicadores
- Red jerárquica de relojes y PLL
- Recursos de entrada y salida (I/O)
- Red de inter-conexionado

En el proceso de la convolución de los datos, la parte crítica es la multiplicación de los datos, esto debido a que cada dato es una palabra de 8 bits, y como la idea es aprovechar al máximo el paralelismo que caracteriza a los autómatas, el implementar las multiplicaciones haciendo uso de los recursos lógicos de la FPGA queda descartado como opción, ya que dicha aproximación al diseño de la celda hace que la misma consuma demasiados recursos lo cual reduce la cantidad de celdas a implementar. Por lo anterior, la multiplicación es realizada haciendo uso de los multiplicadores presentes en los bloques DSP de la FPGA, los cuales, para esta celda en particular suman 288(Altera Corporation), luego el tamaño máximo de celdas es 256, organizadas en forma de arreglo rectangular de 16 filas por 16 columnas.

Table 2–5. DSP Blocks in Stratix II Devices *Note (1)*

Device	DSP Blocks	Total 9 × 9 Multipliers	Total 18 × 18 Multipliers	Total 36 × 36 Multipliers
EP2S15	12	96	48	12
EP2S30	16	128	64	16
EP2S60	36	288	144	36
EP2S90	48	384	192	48
EP2S130	63	504	252	63
EP2S180	96	768	384	96

Figura 20. Bloques DSP y Multiplicadores en la FPGA Stratix II
Fuente. Altera Corporation

Se tiene además que como las máscaras con las que se va a realizar la convolución son máscaras de 3x3, entonces se requieren 9 multiplicaciones por celda para, por lo tanto una convolución toma en principio 9 ciclos de reloj, sin tener en cuenta el tiempo requerido para poner en marcha dicha operación, es decir las operaciones de control.

a) FUNCIONAMIENTO DE LA CELDA

Como el autómata celular es uniforme, todas las celdas funcionan de la misma manera, por lo tanto se realiza la descripción del funcionamiento de una sola celda. La unidad que determina cuando una celda realiza determinada acción, o cuando el autómata celular funciona como un todo es la unidad de control, que será descrita más adelante.

Básicamente, las celdas[1] de la capa lineal del autómata realizan convolución en dos dimensiones con máscaras (filtros) de 3×3 píxeles. Dicho proceso de filtrado consiste a la vez de dos subprocesos. El primero de estos consiste en realizar un proceso de filtrado lineal con dos máscaras que son resultado de la evolución de una máquina genética.

El segundo subproceso es un filtrado no isotrópico denominado *Medida de La Orientación de la Información [11]*, el cual resalta los bordes de la imagen. Seguido a esto la imagen se binariza y es enviada a la capa no lineal del autómata celular.

Adicional a esto, se tiene que los datos son descargados a las celdas desde el procesador NIOS[2] en tres procesos, uno que descarga los coeficientes de los filtros tanto los de filtrado lineal isotrópico como los no isotrópicos, y almacena dichos coeficientes en la celda. El segundo proceso carga las condiciones iníciales de la celda, las cuales se ven como el estado actual de la celda. Finalmente el tercer proceso realiza la convolución de los datos y el resultado final es visto como el estado siguiente de la celda.

Desde el exterior de la celda, la misma se observa como una caja negra con entradas de control y entradas de datos y la salida respectiva de la celda, las cuales se detallan a continuación, junto con una descripción breve de su funcionamiento.

En la figura 21 se tiene el detalle de la celda completa, a partir de ella se explicará cada uno de los componentes de los autómatas lineal y no lineal.

[1]N.del.A. En esta sección, salvo se diga lo contrario de forma explícita, siempre que se haga referencia a la celda, se refiere a los componentes y funcionamiento lineal de la misma, los aspectos del autómata no lineal se explicarán posteriormente.

[2]N.del.A. Cuando se haga referencia a la unidad de control, se usarán indistintamente los términos Nios II o unidad de control.

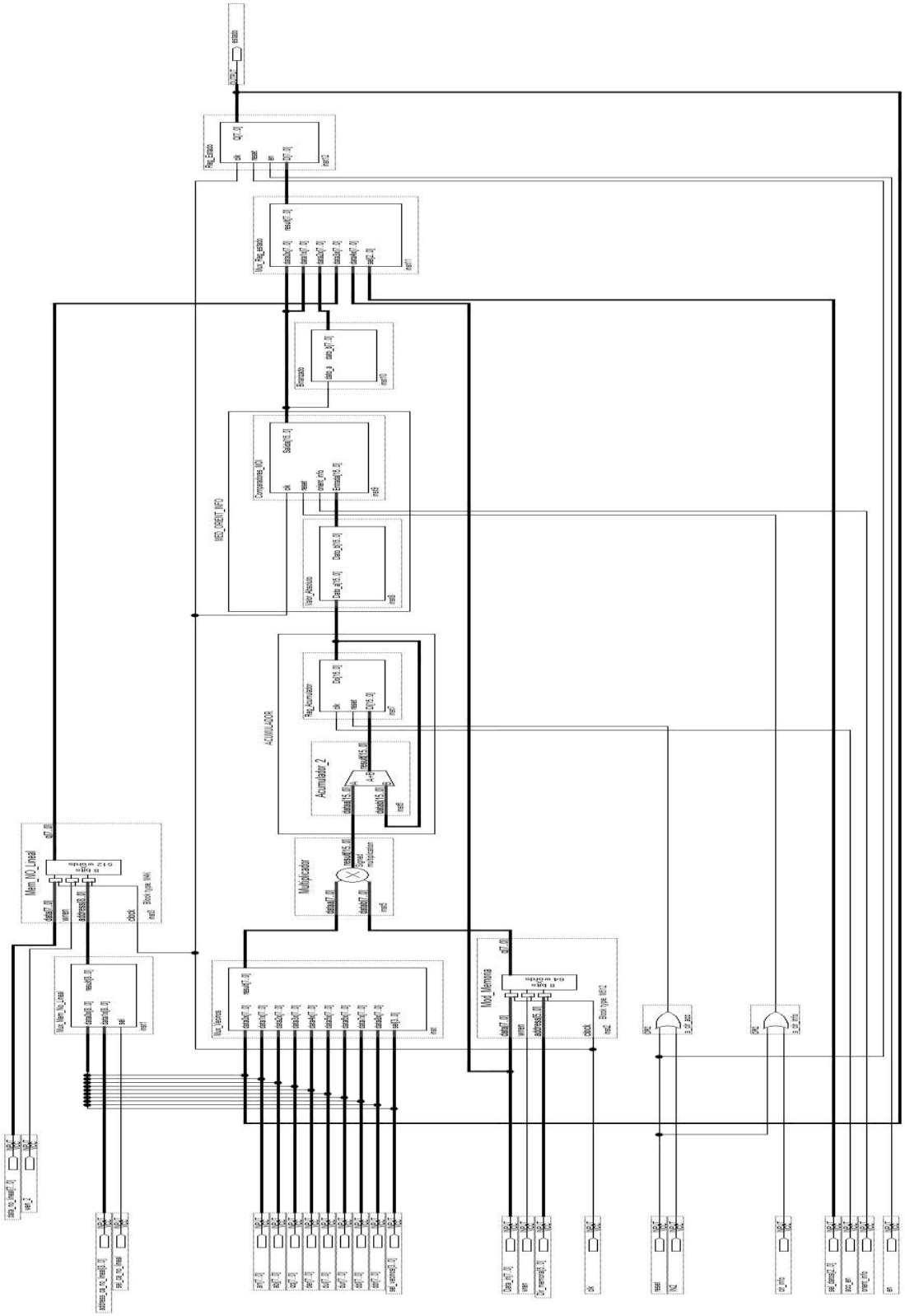

Figura 21. Diagrama Esquemático de la Celda. Fuente. Autor

En la tabla 11, se muestra el detalle de las entradas de datos de la celda.

Tabla 11. Entradas de la Celda[3]

Entradas de Datos	
Data_in (8 bits)	Entrada de datos desde el exterior de la celda, provenientes de la unidad de Control.
arr (8 bits)	Estado de la celda superior
abj (8 bits)	Estado de la celda inferior
izq (8 bits)	Estado de la celda a la izquierda
der (8 bits)	Estado de la celda a la derecha
dul (8 bits)	Estado de la celda de la esquina superior izquierda
dur (8 bits)	Estado de la celda superior derecha
ddl (8 bits)	Estado de la celda inferior izquierda
ddr (8 bits)	Estado de la celda inferior derecha
Entradas de Control	
clk	Reloj global
reset	Reset asíncrono global
sel_datos	Selecciona que datos se verán a la salida de la celda
sel_vecinos	Cambia entre los estado de las celdas vecinas para realizar la convolución.
acc_en	Habilita el módulo acumulador
clr_acc	Borra el módulo acumulador
clr_info	Borra el registro del módulo que realiza la parte final del filtrado ni isotrópico (*Med_orient_info*)
orien_info	Habilita el módulo *Med_orient_info*
dir_memoria	Similar a *sel_vecinos* pero intercambia los coeficientes de la memoria con los cuales se realiza la convolución
wen	Señal de habilitación de escritura en la memoria de coeficientes
sel_ca_no_lineal	Selecciona entre los estados de las celdas y la dirección del exterior (*address_ca_no_lineal*) como apuntador de entrada a la memoria del celda no lineal
wen_2	Habilitador de escritura en la memoria de la celda no lineal
address_ca_no_lineal	Dirección (apuntador) a la memoria de la celda no lineal
data_no_lineal	Datos de entrada a la memoria de la celda no lineal

Fuente. Autor

[3]Los puertos en negrilla están relacionados con la capa no lineal de la celda, que serán explicados posteriormente.

b) ESTRUCTURA DE LA CELDA

En cuanto a la arquitectura de la celda se tiene que cada celda se compone de los siguientes componentes[4].

- Mod_memoria
- Mux_vecinos
- Mux_mem_no_lineal
- Mem_NO_lineal
- Multiplicador
- **Acumulador**
  - **Acumulador_2**
  - **Reg_Acumulador**
- **Med_Orient_Info**
  - **Valor_Absoluto**
  - **Comparadores_MOI**
- Binarizado
- Mux_Reg_Estado
- Reg_estado

A continuación se describe el funcionamiento de cada unidad, su código en VHDL se encuentra en los anexos. En cuanto a la descripción en VHDL, la misma se abordó teniendo como objetivo la minimización en el consumo de los recursos lógicos de la FPGA, por lo tanto, muchas de las descripciones se hacen en bajo nivel, accediendo a las "primitivas" de la FPGA, obteniendo así un mayor control sobre el hardware a ser inferido por el sintetizador de la suite Quartus II de Altera Corporation.

Mux_vecinos

Este bloque es un multiplexor que selecciona uno de los estados de los vecinos a la vez, para ser multiplicado por el respectivo coeficiente de la máscara. Dicho bloque se implementó haciendo uso de las denominadas "MegaWizard Plugin-Manager" de la suite Quartus II de Altera Corp. La escogencia de un multiplexor de "MegaWizard Plugin-Manager" y no uno creado en VHDL radica en el hecho de que tal el bloque, puede considerarse como hardware "genérico" y no se obtiene ninguna ganancia al hacer el mismo eh VHDL.

Para el uso de la herramienta "MegaWizard Plugin-Manager" es requerida la "Librería de Módulos Parametrizados – LPM" de Altera Corp.(Altera Corporation, 1996).

[4]Los componentes en negrilla forman parte la capa no lineal, que será explicada posteriormente.

La descripción en VHDL del bloque se muestra en el anexo 6, su diagrama esquemático (RTL[5]) se muestra en la figura a continuación:

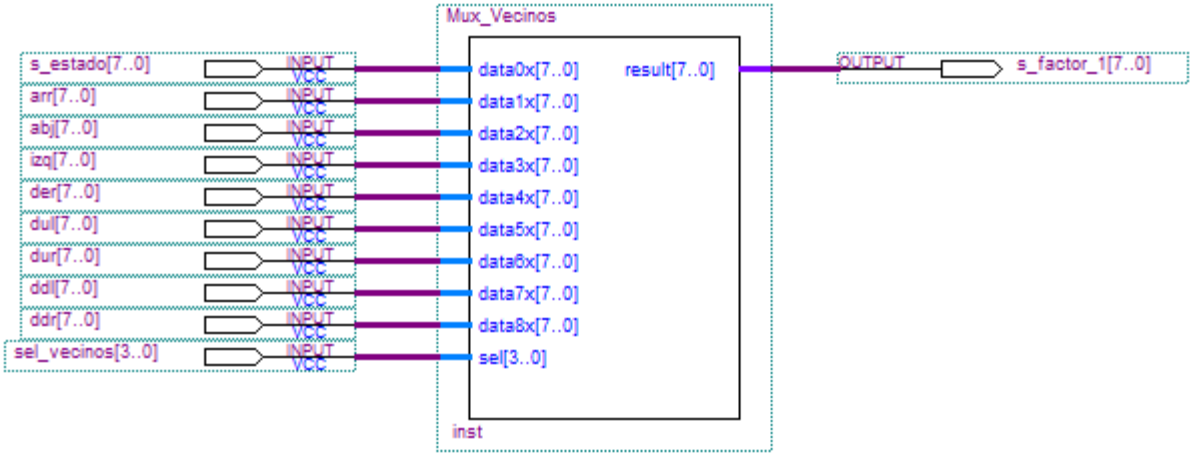

Figura 22. Diagrama Esquemático del "Mux_Vecinos"
Fuente. Autor, obtenido con Quartus II

Mod_memoria

Como se requiere realizar convolución con diferentes máscaras, y éstas deben estar almacenadas en cada celda, ya que forman parte de la regla de transición del autómata, se implementa memoria distribuida valiéndose de los bloques RAM presentes en la FPGA. Dicho módulo debe tener capacidad de almacenar 54 datos cada uno de 8 bits, esto ya que cada filtro es una máscara de 9 datos y son en total los 2 filtros provenientes de la máquina genética, y los 4 filtros de la Medida de la Orientación de la Información.

Por lo anterior, se hace uso de los bloques de memoria de 512 bits (M512 RAM), organizada como arreglo de 64 datos de 8 bits cada uno e implementados con la asistencia de la herramienta “MegaWizard Plugin-Manager”. La descripción en VHDL se encuentra en el anexo 7 y el diagrama esquemático RTL del módulo se presentan a continuación.

[5]RTL – Del inglés *Register Transfer Level*

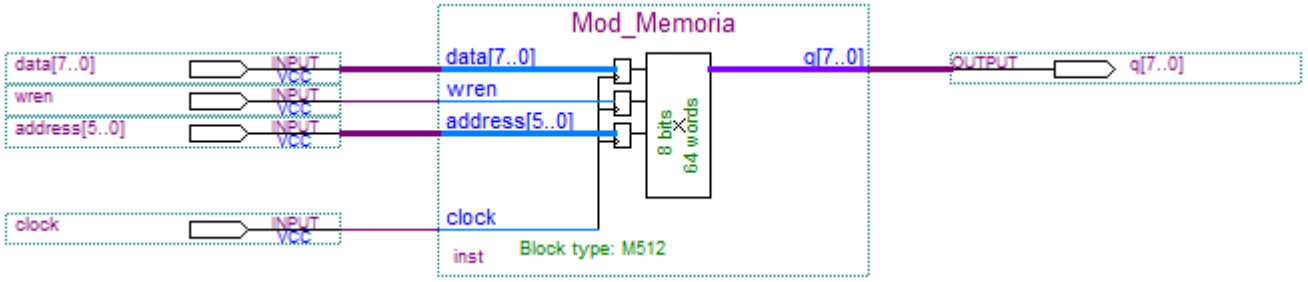

Figura 23. Diagrama Esquemático del Módulo "Mod_Memoria"
Fuente. Autor, obtenido con Quartus II.

Multiplicador

La operación de multiplicación y acumulación, denominada MAC[6] es el núcleo de la convolución, y la FPGA Stratix II posee elementos especializados para llevar a cabo esta operación, los bloques DSP. La desventaja de hacer uso de esta facilidad de la FPGA es que agota más rápidamente los multiplicadores presentes en cada bloque, haciendo que la cantidad de celdas a implementar disminuya, por lo tanto, se divide la operación MAC en dos partes, una dedicada exclusivamente a la multiplicación, la cual hará uso de los multiplicadores presentes en los bloques DSP, y otra parte dedicada exclusivamente a la acumulación.

Además de lo anterior, se observa que los coeficientes de los filtros pueden ser negativos, luego resulta necesario que las operaciones aritméticas dentro cada celda soporte manipulación de cantidades negativas, es decir, la representación de los datos dentro de cada celda estará dada en complemento a dos. En cuanto a la implementación, de nuevo se hace uso de la herramienta "MegaWizard Plugin-Manager" para crear los multiplicadores en complemento a dos, esto ya que al hacer uso de la herramienta se garantiza que durante el proceso de sintetizado y ruteo de hardware, se infieran e implementen los bloques multiplicadores existentes en la FPGA. El código en VHDL del multiplicador se encuentra en el anexo 8, el esquemático se muestran a continuación.

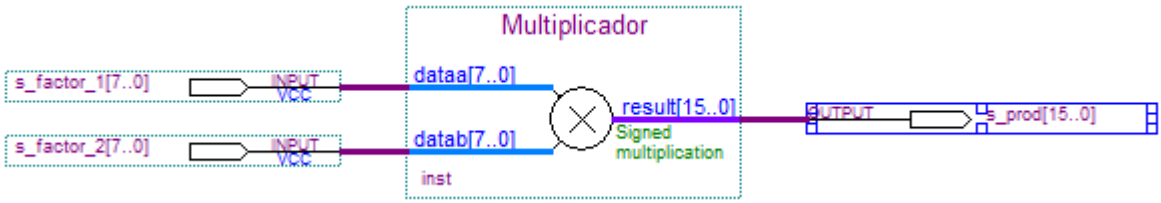

Figura 24. Esquemático del Módulo "Multiplicador"
Fuente. Autor

[6] MAC – del inglés Multiply-Accumulate

Acumulador

El acumulador es un registro que va sumando el resultado de la multiplicación y el dato presente anteriormente en el registro. De nuevo, el sumador realiza la operación en complemento a dos para permitir la manipulación de cantidades negativas. La implementación de dicho módulo se subdivide en dos módulos, a saber, el módulo que realiza la acumulación (suma) denominador "*Acumulador_2"* y el registro que va almacenando el dato, denominado "*Reg_Acumulador*".

El acumulador se implementó con asistencia de la herramienta "MegaWizard Plugin-Manager" y el registro se implementó con primitivas de hardware de Altera, a saber, como múltiples registros conectados en paralelo. El código de los dos módulos se encuentra en el anexo 8, el esquemático RTL se muestra en la figura 25.

El diagrama esquemático del acumulador se muestra en la figura siguiente.

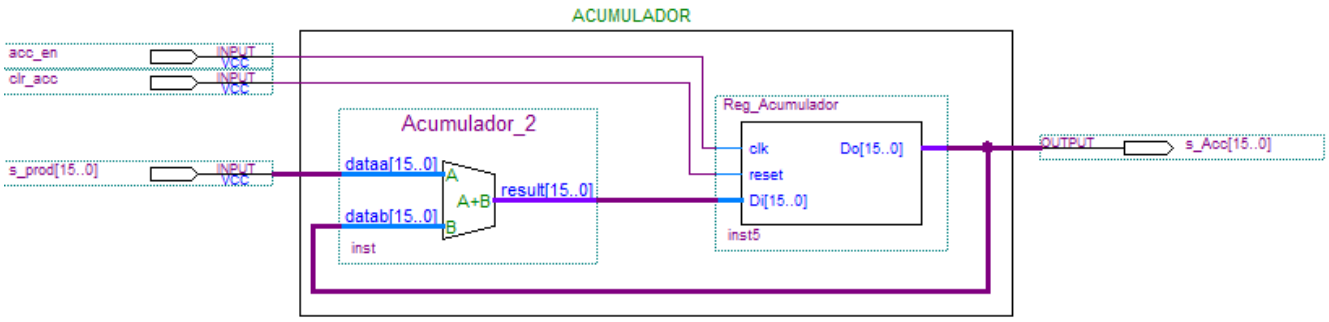

Figura 25. Esquemático RTL del Acumulador
Fuente. Autor, Obtenido con Quartus II

Med_orient_Info

Este módulo realiza la parte final del filtrado no isotrópico, el cual como ya se mencionó anteriormente consiste en tomar el valor absoluto del resultado de cada convolución y comparar entre los cuatro valores absolutos obtenidos (uno por cada máscara MOI) para seleccionar el mayor como resultado final del proceso. Para el cálculo del valor absoluto, en vista de que los datos se encuentran en representación complemento a dos, se observa el bit más significativo (que hace las veces de bit de signo) y si el mismo es '1' se toma el complemento a dos del dato como entrada al comparador, de lo contrario el dato pasa al comparador sin alterarse.

El comparador consiste de un circuito comparador y un registro, de tal manera que se aproveche la propiedad de transitividad de la relación de orden, y se optimice el

uso de recursos lógicos de la FPGA. Es decir, inicialmente se comparan los datos frente a cero, lo cual causa que independiente del dato que ingrese, dicho dato ocupe la posición de máximo; y a medida que ingresen nuevos datos, se comparen con el que hasta ahora es el máximo.

Los códigos en VHDL para tanto el Valor Absoluto, como para el comparador se encuentran en el anexo 9, el esquemático se muestra en la figura a continuación.

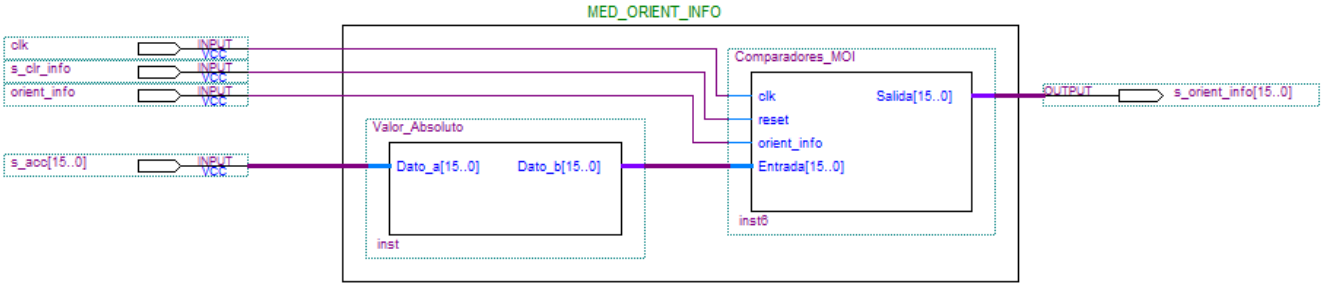

Figura 26. Esquemático - Módulo "Med_Orient_Info"Fuente. Autor

Proceso de Binarización

Una vez finalizada la etapa de filtrado usando los filtros de Medida de Orientación de la Información, se requiere una imagen binaria como entrada al autómata no lineal; este proceso se logra estableciendo un umbral en la imagen, a partir del cual el estado de la celda se establece en ‘1’, y por debajo del cual el estado se hace ‘0’. Como ejemplo del “binarizado”, estableciendo el umbral en el bit 7, de un dato de 16 bit Little-endian, lo cual se traduce en un nivel de umbral de aproximadamente el 50%.

MSB															**LSB**
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

El código en VHDL se encuentra en el anexo 10

Mux_Reg_Estado

El multiplexor previo al registro de estado de la celda, selecciona los datos a mostrar a la salida de la celda, es decir el estado de la misma, dicho multiplexor permite además que se puedan exportar todos los resultados parciales del proceso de la celda, a saber, el filtrado isotrópico para mejora de contraste, el filtrado con las máscaras MOI, la imagen binarizada, y la imagen luego de pasar por la celda no lineal. De nuevo, la implementación del multiplexor se realiza con la herramienta “MegaWizard Plugin-Manager”, como se muestra en el anexo 11.

El diagrama esquemático RTL se aprecia en la siguiente figura:

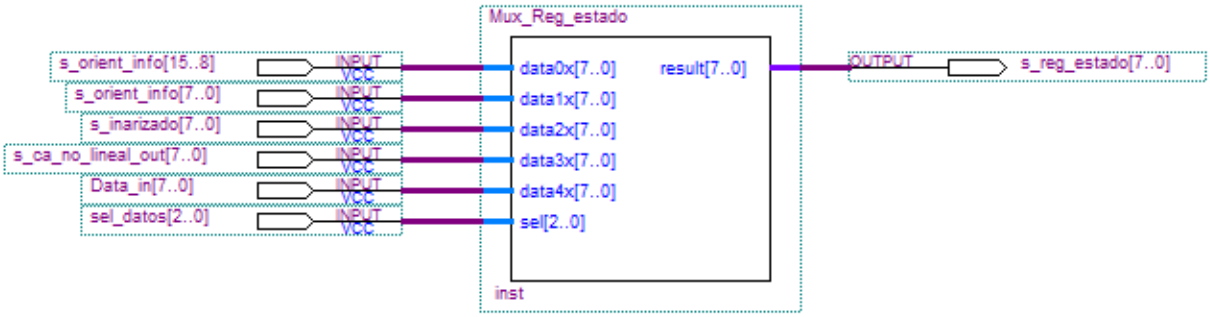

Figura 27. Diagrama Esquemático Mux_Reg_Estado
Fuente: autor.

Reg_estado

El registro de estado es un registro de 8 bits a la salida de la celda del autómata que hace evidente la transición en el tiempo de los estados del autómata, ya que este solo habilitará su salida una vez culminado el proceso de convolución, y lo hará de manera similar para todas las celdas del autómata y de forma simultánea. La descripción en VHDL del registro de estado se muestra a continuación, cabe aclarar que la descripción de este bloque fue realizada en base a primitivas de flip-flop de altera presentes en la FPGA. El código en VHDL se encuentra en el anexo 12.

6.4.2.2 DESCRIPCIÓN EN VHDL DEL AUTÓMATA CELULAR – CAPA NO LINEAL

Luego de realizados los procesos de filtrado y de haber realizado el proceso en donde se mide la orientación de la información de la imagen, se obtiene como resultado una imagen binarizada con los bordes de la imagen original. Esta imagen binaria de bordes es procesada por la capa no lineal del autómata con el fin de mejorar algunas características de la imagen, conectar bordes por ejemplo.

La capa no lineal consiste básicamente de una tabla de búsqueda a la cual se descarga desde el exterior (el PC), es decir la regla de transición, y que dependiendo de la misma cambia el estado del autómata.

Desde el punto de vista de implementación, la capa no lineal es sencillamente un bloque más dentro de la celda del autómata mencionado en la sección anterior, esto con el fin de facilitar la replicación de celdas y de hacer un uso eficiente de los recursos de ruteo de la FPGA. Además al estar esta capa dentro de la celda ya existente, se gana el tener solo unidad de control maestra sobre todas las celdas.

Cada celda de este autómata recibe como entradas el estado actual del autómata (resultado obtenido de la medida de la orientación de la información) y con base en una regla cargada en la celda, cambia el valor de la celda, cambiando así el estado del autómata cuando se ven todas las celdas a la vez. La implementación de dicho autómata se hace de manera similar que con la capa lineal, es decir, haciendo uso de memoria distribuida usando los bloques de memoria, en este caso los bloques M4K RAM.

De esta forma, se vuelve sencillo el instanciar este autómata como un bloque dentro del autómata anterior (lineal) y no se hace uso de los recursos lógicos de la FPGA, consiguiéndose así un sistema complejo en cuanto a función y muy sencillo en cuanto a estructura, lo cual es completamente coherente con el autómata celular como modelo matemático.

La descripción en VHDL de la capa no lineal es básicamente una copia del módulo de memoria ***Mod_memoria*** usado en la celda de la capa lineal del autómata. La diferencia radica en el tamaño del bus de direcciones para hacer que el sintetizador de Quartus infiera los bloques de memoria deseados, es decir, los bloques *M4K RAM*.

Los módulos que componen la celda no lineal, como se señaló en secciones anteriores aparecen resaltados en rojo y se rescriben a continuación.

- Mux_mem_no_lineal
- Mem_NO_lineal

Mux_mem_no_lineal

Similar al multiplexor de vecinos, este multiplexor selecciona entre los la dirección de memoria, proveniente del NIOS, o el estado de las celdas vecinas del autómata, en este último caso, toma solo un bit por celda, para armar una señal de nueve bits como apuntador a la memoria. En el anexo 13 se muestra la descripción en VHDL del módulo.

Mem_NO_Lineal

Similar a la memoria de la celda lineal, dicha celda permite el almacenamiento de los coeficientes de la regla del autómata no lineal, y al igual que para su equivalente en la celda lineal, la implementación se lleva a cabo con la herramienta “MegaWizard Plugin-Manager”, en la cual se crea una memoria *M4K RAM*, configurada como 512 palabras de 8 bits cada una.

En cuanto al funcionamiento de dicho módulo se tiene que, inicialmente, la memoria es escrita desde el exterior (desde el NIOS), lo cual carga la regla del

autómata NO lineal en los espacios de memoria de las celdas. Una vez cargados los datos, y con las celdas en funcionamiento se inhabilita la escritura en dicha memoria dejando como dirección de lectura los estados de las celdas vecinas, creando así un autómata celular no lineal.

En principio, dada la configuración de la memoria, dicho autómata soporta 8 reglas binarias diferentes, una por cada bit del dato de salida de la memoria, esto, sumado a que los datos son realimentados al autómata, forman un autómata no lineal de ocho capas. El código en VHDL se encuentra en el anexo 14, el esquemático se muestra en la figura a continuación:

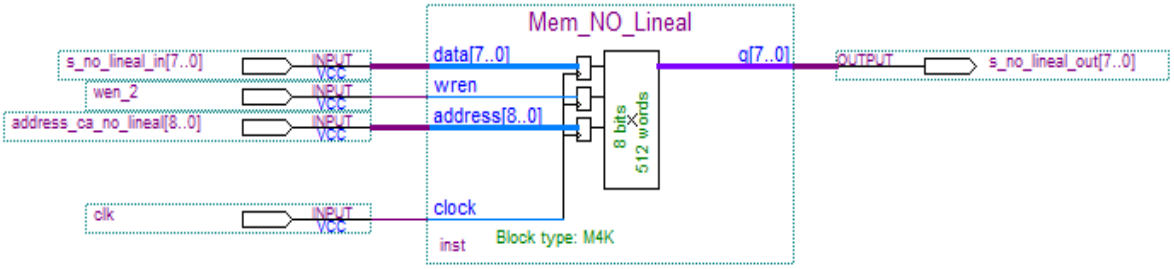

Figura 28. Diagrama Esquemático - Mem_NO_Lineal
Fuente. Autor

Como se observa en el código del anexo 14, la memoria de la celda no lineal es inicializada con una regla precargada ("*Mem_no_lineal.mif*"). La regla usada fue la denominada *Regla 56*, la cual en una imagen binaria detecta bordes(Tamanaha Goi, 2003). Dicha regla funciona de la siguiente manera, cuenta el número N de los píxeles que estén en estado '1' de las celdas vecinas (junto con su propio estadio), si tal número $4 < N < 6$ entonces el estado nuevo de la celda será '1', de lo contrario el nuevo estado de la celda es '0'.

Expresado matemáticamente se tiene que: definida una vecindad Moore de radio 1 sobre la celdas, es decir el estado de la celda en cuestión y el estado de sus ocho celdas adyacentes vecinas, se tiene que las celdas pueden estar vivas, es decir tener estado '1' o muertas, es decir tener estado '0', el siguiente estado de las celdas queda determinado por:

Ecuación 19. Regla autómata no lineal

$$e_k(t+1) = \begin{cases} 0, & 0 \le s_k(t) \le 3 \text{ o } 6 \le s_k(t) \le 9 \\ 1, & \text{en otros casos} \end{cases}$$

Fuente. Autor.

Donde,

$e_k(t) :=$ Estado de la celda k en el instante t

$e_k(t+1) :=$ Estado de la celda k en el instante $t+1$

$s_k(t) :=$ Número de celdas "vivas" vecinas de la celda $e_k(t)$ en el instante t

6.4.2.3 CONEXIÓN DEL AUTÓMATA CELULAR

Como el autómata celular es uniforme y regular, lo cual significa que todas las celdas son idénticas, en la conexión de las celdas esta resulta ser una característica atractiva del autómata, ya que facilita la conexión convirtiendo este proceso en un proceso de replicación de estructuras iguales, y dotando así de una generalidad al autómata adicional a la que explicó en el diseño de la celda. La ganancia de generalidad radica en el hecho de que al ser un proceso de replicación de celdas, en principio el autómata puede ser de tamaño variable limitado solo por los recursos presentes en el dispositivo, como es el caso que se discute en este documento.

En el anexo 15 se encuentra la conexión por VHDL correspondiente al diagrama esquemático de la celda (entiéndase por celda = lineal + no lineal) mostrado en la figura 21.

El autómata celular está formado por 256 celdas más una capa exterior de registros que hacen las veces de condiciones de frontera, y la conexión de las celdas para la construcción del autómata celular se realiza mediante la manipulación de arreglos multidimensionales. Se crean tres arreglos descritos a continuación. Con la descripción funcional y el diagrama de bloques respectivo, y una versión condensada del código en VHDL para la implementación de los mismos, por condensada se hace referencia a que solo se muestra su creación e inmediatamente después la implementación del mismo, lo cual no es usualmente correcto en el lenguaje VHDL

a) Arreglo 1. Habilitación de escritura en las celdas

Dicho arreglo “desenrolla” el vector de direcciones Dir y direcciona cada bit a una celda del autómata. Es una arreglo 1Dx1D de 17 datos de 17 bit cada uno, en la figura 29 se muestra un diagrama de dicho arreglo, el código en VHDL se encuentra en el anexo 16.

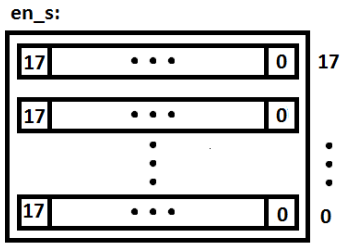

Figura 29. Diagrama del arreglo para habilitación de escritura en las celdas
Fuente. Autor

b) Arreglo 2. Matriz de conexión de las celdas

Este arreglo es una matriz en hardware, en donde cada entrada i, j de la matriz corresponde a una celda. La implementación del mismo en VHDL (anexo 17) se realiza con un arreglo anidado dentro de otro arreglo, que a la vez se anida en un arreglo final, es decir, el arreglo resultante es de tipo 1Dx1Dx1Dx. El código en VHDL del anexo 17 muestra de manera condensada la forma en la cual se creó la matriz de las celdas, en el mismo se omitieron los detalles de la arquitectura de cada celda. El diagrama se muestra en la figura 30.

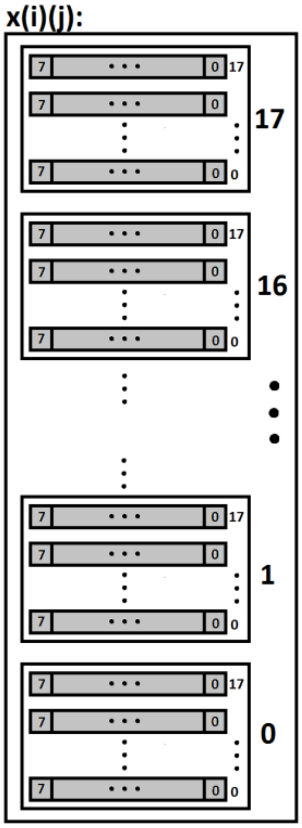

Figura 30. Diagrama de la "matriz" de celdas - Arreglo 2. Fuente: autor

c) arreglo 3. Datos de salida

Este último arreglo reordena cada uno de los estados de salida de las celdas almacenados en las posiciones $x(i)(j)$ del Arreglo 2 y los convierte en un vector 1Dx1D de 256 entradas cada una de 8 bits. Este arreglo ya no tiene en cuenta la salida de las celdas correspondientes a condiciones de frontera, solo mapea las celdas del bloque de 16×16 píxeles de la imagen a procesar. El código en VHDL se encuentra en el anexo 18, mientras que el diagrama se muestra en la figura 31.

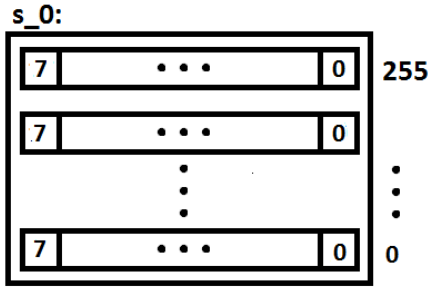

Figura 31. Diagrama del Arreglo 3

Fuente. Autor

Finalmente, cada una de las 256 entradas del vector *s_0* es la entrada de un multiplexor de $256:1$ a la salida del autómata y el cual es controlado por la señal *Addr* como se mencionó anteriormente.

6.4.2.4 UNIDAD DE CONTROL – PROCESADOR NIOS II

La unidad de control está formada por el procesador NIOS II de Altera Corp.(Altera Corporation, 2009)(Altera Corporation, 2009), el cual es un microprocesador embebido de 32 bits, y de módulos añadidos al microprocesador, a saber, una UART para la comunicación con el exterior (PC) y un controlador de memoria SRAM (externa a la FPGA), la cual hace las veces de memoria de datos y de memoria de programa. Dichos módulos junto con el procesador son implementados con la herramienta "SOPC Builder"(Altera Corporation, 2009).

La figura 32 muestra una imagen que describe a groso modo la arquitectura interna del procesador NIOS II.

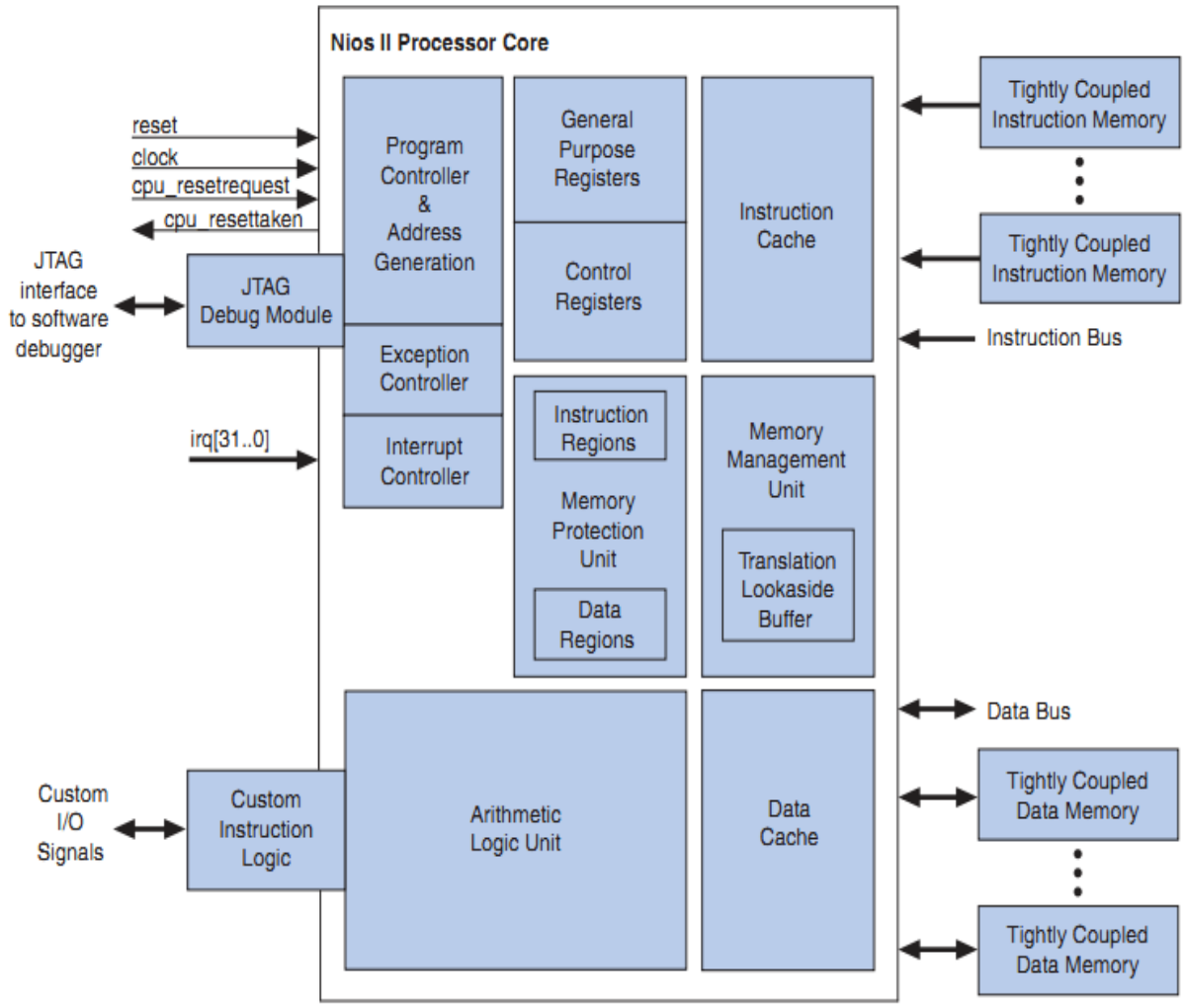

Figura 32. Arquitectura del Procesador NIOS II
Fuente. Tomado de (Altera Corporation, 2009)

Una vez instanciado el procesador, solo resta conectar los dos grandes bloques (Autómata celular y Unidad de control), y diseñar la rutina de software que gobernará el funcionamiento de todas las celdas, es decir, del autómata e general. La imagen que se muestra a continuación, muestra el detalle de la conexión de dichos bloques.

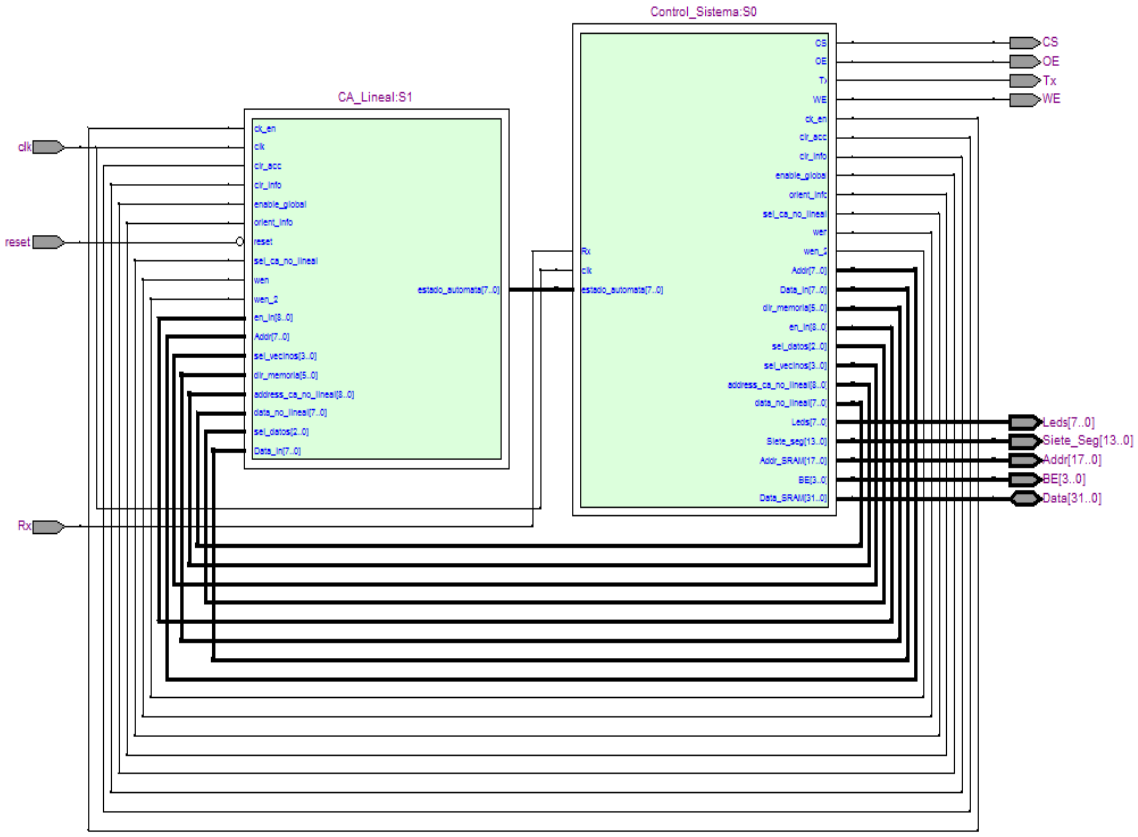

Figura 33. Autómata Celular Híbrido sobre FPGA
Fuente. Autor

En cuanto a las rutinas de software para controlar al autómata, se tiene que las funciones desempeñadas por el autómata son en orden descendente:

1) Leer los datos desde la UART
    a. Coeficientes para la ecualización
    b. Máscaras provenientes de la máquina genética (2 filtros de 3x3 píxeles)
    c. Regla del Autómata No Lineal
    d. Coeficientes de los MOI (4 filtros de 3x3 píxeles que son fijos)
    e. Imagen a filtrar
2) Ecualizar con los datos entrantes al sistema
3) Almacenar los datos en la memoria SRAM
4) Escribir la regla de transición en el autómata no lineal
5) Escribir los datos de configuración
    a. Filtros de la máquina evolutiva (mejora de contraste)
    b. Filtros MOI
6) Escribir los píxeles de la imagen a filtrar en bloques de 16x16 píxeles (tamaño máximo de celdas del autómata

7) Leer y transmitir el resultado de cada bloque de la imagen procesada
    a. Leer resultado del procesamiento MOI
    b. Leer la imagen binarizada
    c. Leer el resultado de aplicar la regla del autómata no lineal
8) Repetir lo anterior (pasos 1-8) una cantidad n de veces en donde n representa la cantidad de bloques de 16x16 píxeles que forman la imagen
9) Esperar por un nuevo proceso, y en dado caso volver al paso 1

PROGRAMACIÓN DEL PROCESADOR NIOS II

La secuencia de pasos recién descrita se traduce en 7 archivos .c que gobiernan el funcionamiento del sistema, los cuales se muestran en la tabla siguiente junto con una descripción breve de los mismos.

Tabla 4. Programas para el Procesador NIOS II

Programas – Procesador NIOS II	
Programa	**Descripción**
Control.c	Hace las veces de archivo cabecera, y es en donde se describen los pasos del proceso mencionados anteriormente, hace las veces del main(void)
Adicional.h	Librería de las funciones que realizan procesamiento y control sobre el autómata
Adicional.c	Funciones que gobiernan el autómata
Automata.h	Librería para la lectura y escritura de datos del autómata
Automata.c	Funciones de lectura y escritura de datos del autómata
Manejo_UART.h	Librería para la comunicación con el exterior (PC)
Manejo_UART.h	Funciones de lectura y escritura de datos desde y hacia el exterior (PC)

Fuente. Autor

Interfaz de Comunicación – MATLAB

Finalmente, adicional a los programas que gobiernan en NIOS, y en últimas al autómata entero, se deben acondicionar los datos que provienen desde el exterior, es decir, dar el formato adecuado la imagen y dividir la misma en bloques tal que se transmitan bloques de 16x16 píxeles más las condiciones de frontera, y leer los datos y armar los boques de las imágenes respectivas, en últimas, en MATLAB solo se realiza acondicionado y reformado de la imagen más no procesamiento, que se deja sujeto al autómata. Esta interfaz consiste de tres archivos .m que se encuentran en el anexo 19.a continuación se muestra una breve descripción de los mismos.

Interfaz de Comunicación MATLAB – Autómata	
Programa	**Descripción**
Interfaz.m	Crea el puerto serial y establece las propiedades del mismo
Filtrado.m	Escribe en el puerto serial el segmento de imagen junto con los filtros a usar
Comunicación.m	Lee la imagen, acondiciona el formato de la misma, arma los bloques de 16x16 píxeles, y determina las condiciones de frontera de cada bloque de acuerdo a los bloques vecinos y a la posición del bloque respecto a la imagen completa. Luego de la escritura, lee los datos del puerto serial y arma las imágenes resultado

6.5 DESARROLLO DEL CUARTO OBJETIVO

"Evaluar el desempeño de la implementación en hardware mediante una aplicación orientada al proceso de datos en dos dimensiones."

6.5.1 DISEÑO DE LA FUNCIÓN DE APTITUD (FITNESS), PARA LA GENERACIÓN DE REGLAS PARA LA IMPLEMENTACIÓN DE FILTROS SE SUAVIZAMIENTO Y DETECCIÓN DE BORDES EN UNA MATRIZ QUE REPRESENTA UNA IMAGEN.

6.5.1.1 MEDICION DE APTITUD

La función para evaluar la aptitud de cada uno de los individuos de la población, corresponde a la medición de contraste. Una imagen con alto contraste tienen un histograma más uniforme que otra con inferior contraste. La herramienta empleada fue la de varianza, un histograma con una varianza baja representa un contraste alto, y por lo contrario una imagen con varianza alta en su histograma significa un contraste bajo.

El código para la medición de aptitud corresponde a:

```
function [apt,vj]=aptitud(HX,HJ)
% ingresa el histograma de la imagen original HJ
% ingresa el histograma de la imagen resultante HX
mx=mean(HX);
mj=mean(HJ);
vx=dot((HX-mx),(HX-mx))/(mx^2);
vj=dot((HJ-mj),(HJ-mj))/(mj^2);
apt=vx;
```

Figura 34. Función de "*Fitness*"
Fuente: propia

6.5.2 EVALUACIÓN DEL AUTÓMATA CELULAR, MEDIANTE LA PRUEBA CON LA IMPLEMENTACIÓN DE FILTROS DE SUAVIZA MIENTO Y DETECCIÓN DE BORDES

Al correr la máquina evolutiva y entregar los datos al autómata se obtiene los siguientes resultados de la ecualización del histograma.

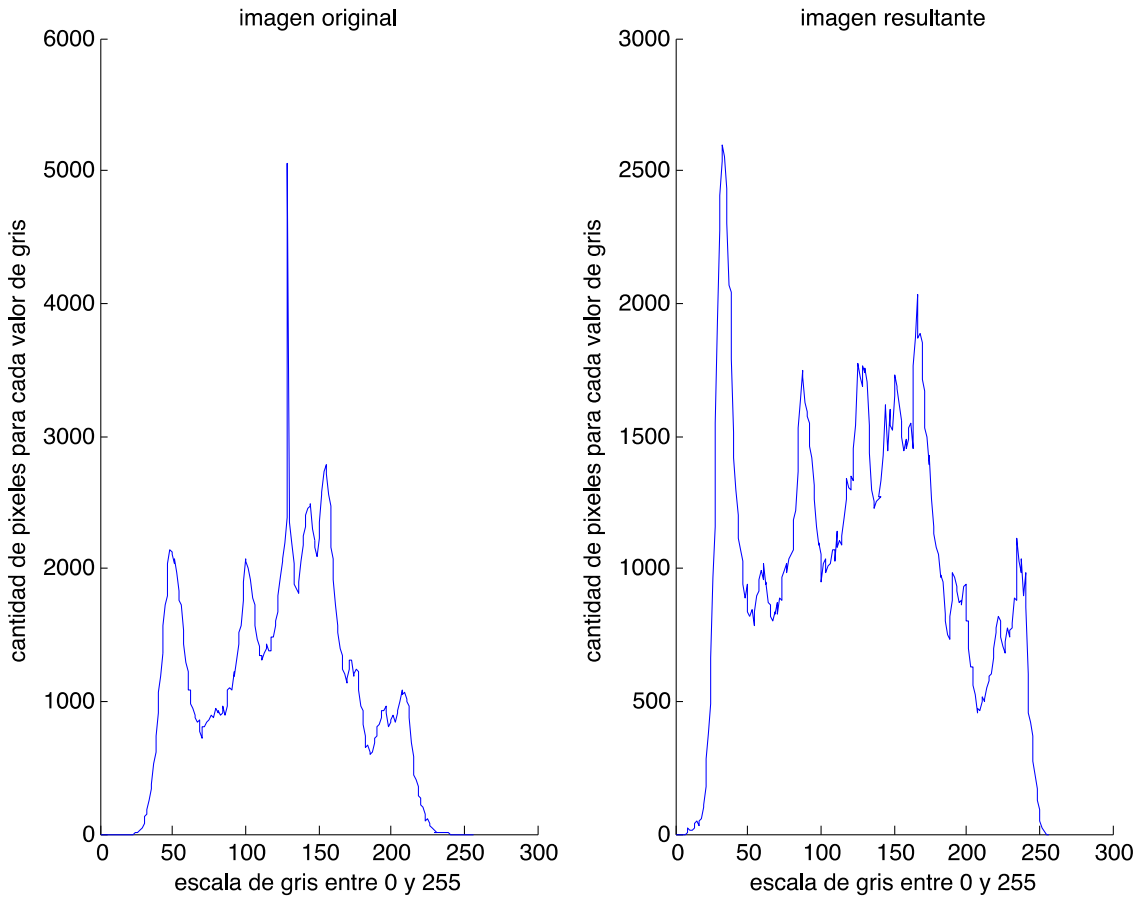

Figura 35. Medición de histograma- mejora de contraste para la imagen de la figura 36
Fuente: propia

En la figura anterior se contrastan los histogramas para la imagen de entrada y para la imagen que resulta del proceso de filtrado lineal y no lineal. En el histograma de la izquierda se observa que tiene un valor pico alrededor de 5000, mientras que la imagen de la derecha tiene un valor pico de aproximadamente 2500, de otra parte el rango de los valores de gris en el histograma de la derecha es mayor que en el histograma de la imagen original, ello evidencia una mejora en el contraste de la imagen.

Las imágenes resultantes del proceso se observan en la imágenes mostradas en la figura 36. La imagen de la esquina superior izquierda (imagen original) muestra la imagen correspondiente al histograma de la izquierda de la figura 35, mientras que en la esquina superior derecha se aprecia el resultado de la mejora de contraste.

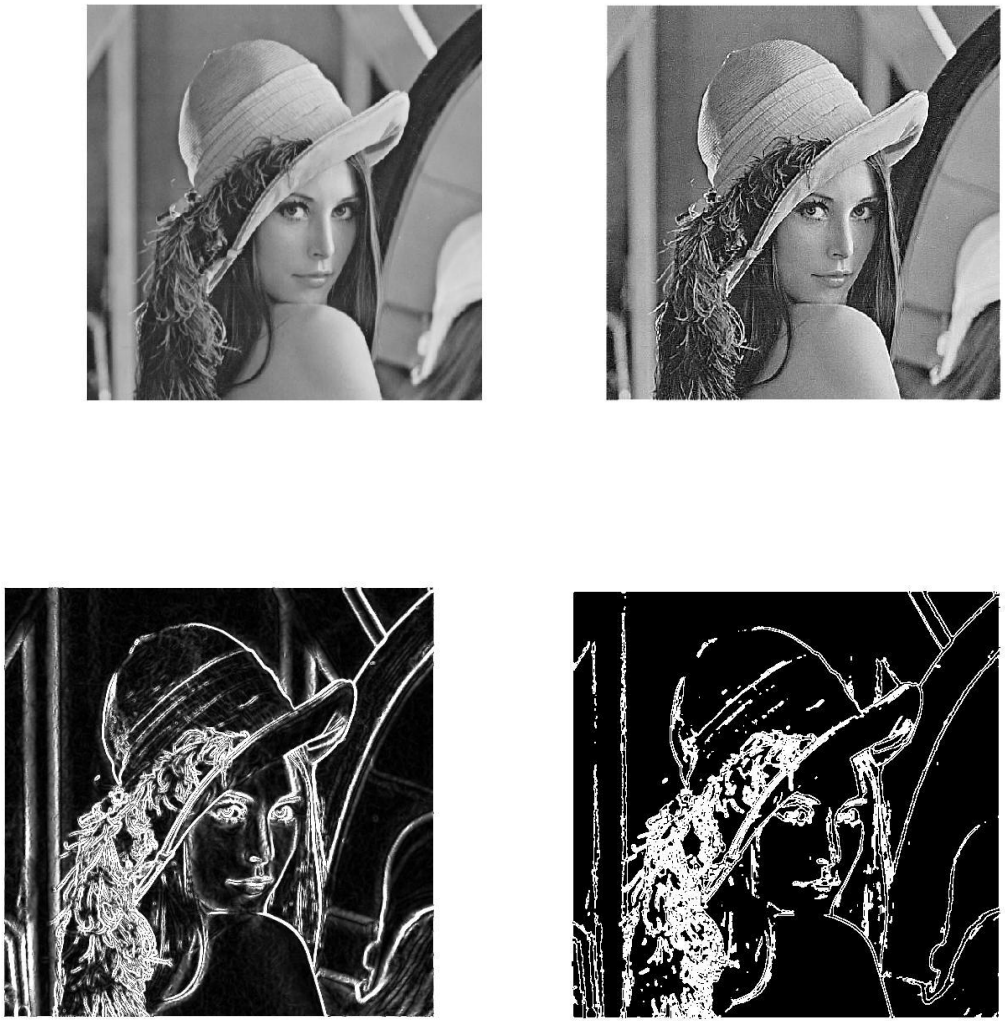

Figura 36. Resultados del filtrado
Fuente. Autor

En la parte inferior izquierda se tiene la imagen con la información de orientación, y finalmente a la derecha en la parte inferior se tienen los bordes detectados por el autómata.

En este proceso la máquina genética entrego las siguientes mascaras para los filtros:

h1 =

0.2649	0.0005	0.2649
0.0005	-0.0616	0.0005
0.2649	0.0005	0.2649

h2 =

0	0	0
0	1	0
0	0	0

En las figuras mostradas a continuación se muestra el resultado con otras imágenes. En la figura 37 se realiza el mismo proceso de mejora de contraste para la imagen de una moneda, mientras que en la figura 39 se muestra el resultado para la imagen de otro rostro.

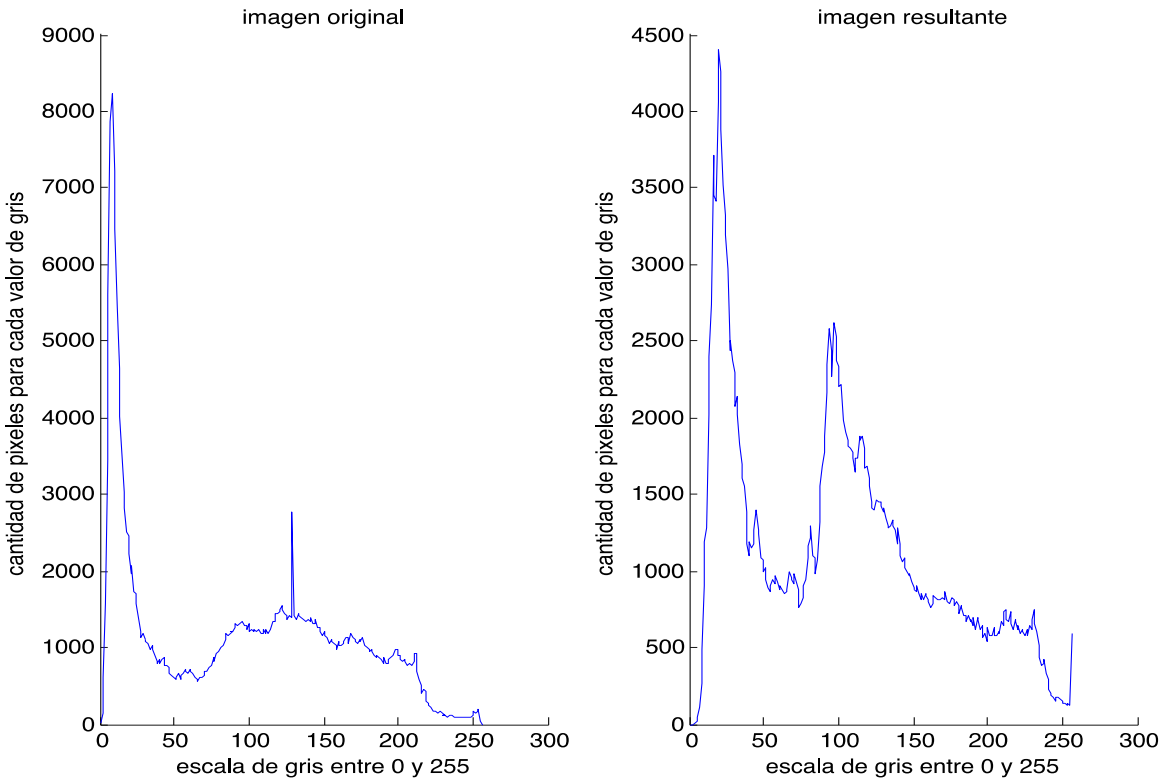

Figura 37. Mejora de contraste para la imagen de la figura 38
Fuente. Autor

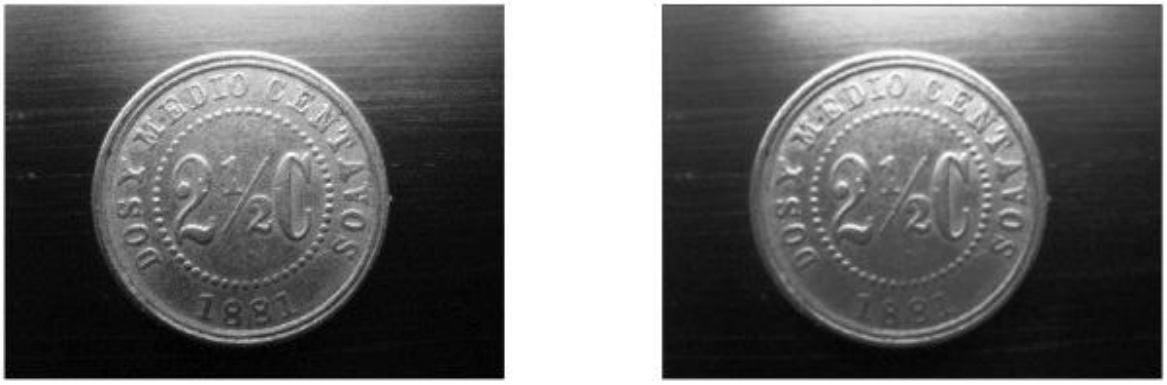

Figura 38. Mejora de contraste. Derecha: imagen original. Izquierda: imagen resultante

En este caso las mascaras corresponden a:

h1 =

0.2725	0.0000	0.2725
0.0000	-0.0899	0.0000
0.2725	0.0000	0.2725

h2 =

0.0010	0.0005	0.0010
0.0021	1.0000	0.0021
0.0010	0.0005	0.0010

En las figuras 39 y 40 se observan los resultados del mismo proceso para la imagen de otro rostro.

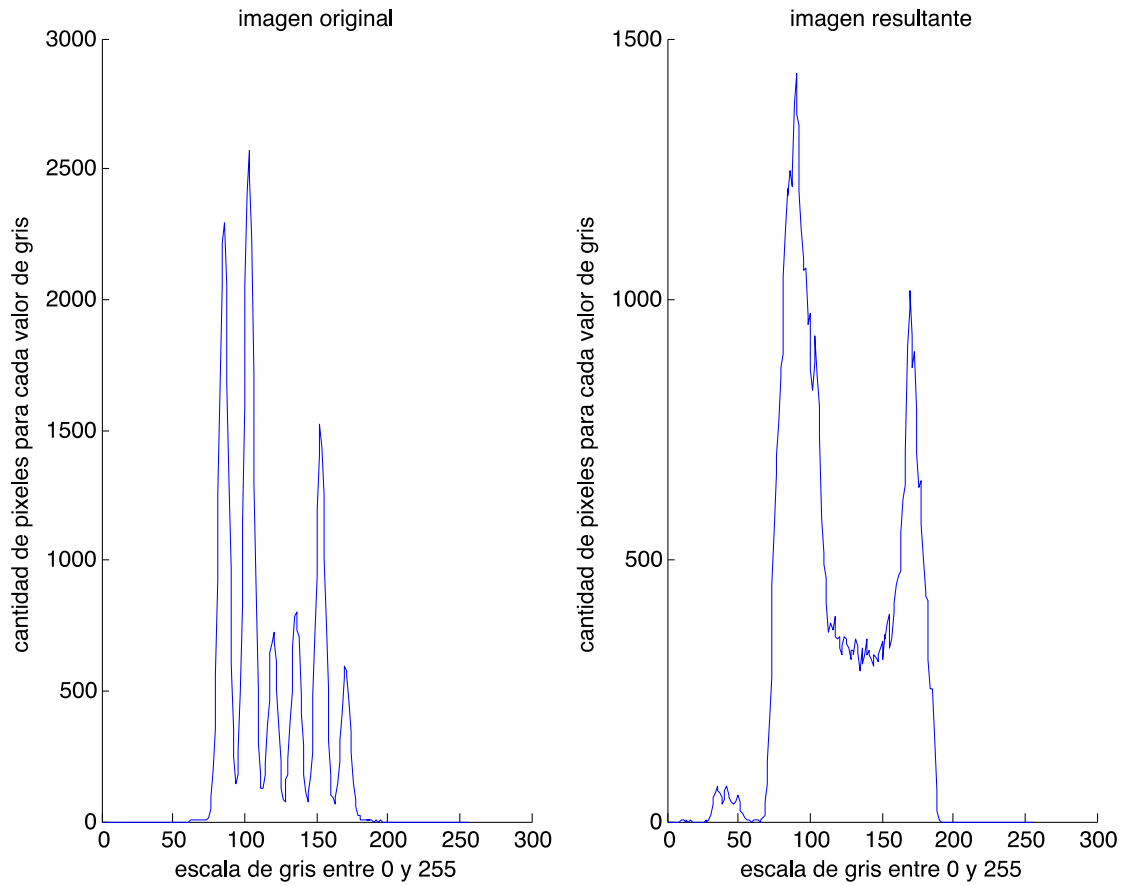

Figura 39. Mejora de contraste para la imagen de la figura 40
Fuente. Autor

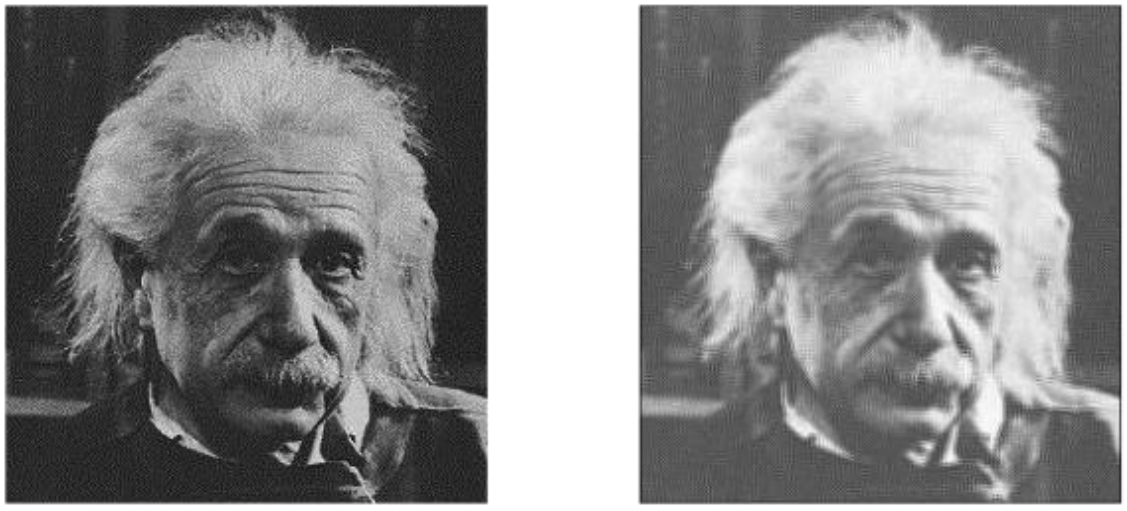

Figura 40. Mejora de contraste. Derecha: imagen original. Izquierda: imagen resultante

En este caso las mascaras corresponden a:

h1 =

0.2865	0.0039	0.2865
0.0039	-0.1615	0.0039
0.2865	0.0039	0.2865

h2 =

0	0	0
0	1	0
0	0	0

7 ANALISIS DE RESULTADOS

De acuerdo a los objetivos trazados se considera que se alcanzaron las metas para cada objetivo y se justifica en la explicación dada a continuación:

7.1 PRIMER OBJETIVO

Se planteó el modelo matemático general de los autómatas celulares basado en las referencias bibliográficas citadas, lo cual permite entender el funcionamiento de los autómatas tanto de naturaleza lineal como no lineal; de otra parte la representación de la función de actualización de cada celda depende completamente de la aplicación en particular y de ello depende que el análisis de los autómatas requiera de herramientas de las matemáticas más complejas.

En el caso particular de los autómatas implementados, el modelo explica su funcionamiento y permite implementarlo en plataformas de software y hardware.

La validación del modelo matemático se consigue a partir de la implementación en la FPGA del autómata hibrido compuesto de uno lineal y otro o lineal.

7.2 SEGUNDO OBJETIVO

La máquina genética implementada en la plataforma de Matlab, permitió a partir de una población inicial evolucionar la población final para obtener individuos con cromosomas que cumplieran con la función de aptitud establecida. El proceso de evolución ocurrió a partir de las estrategias de cruce y mutación diseñadas.

La evolución de la máquina genética toma en algunas ocasiones un tiempo considerable sobre el entorno de Matlab. Ello se debe que todas las operaciones se realizan en modo secuencial, sin embargo la implementación de los algoritmos en un lenguaje de bajo nivel como C/C++ permitirían acelerar el proceso de la evolución.

La evolución de la máquina no siempre converge a valores pequeños de varianza (altos valores de contraste). Esto se debe a que la evolución de los filtros en el dominio del espacio se traduce en grande cambios en el espectro. De otra parte la

función de aptitud escogida para reducir el computo no evalúa los contenidos espectrales simplemente evalúa el contraste a partir de la varianza del histograma.

El autómata sobre la FPGA se diseñó de tal forma que se reconfigura sin ningún inconveniente cada vez que se envía los fenotipos desde el PC, no se requiere reprogramar la FPGA para reconfigurar el Hardware.

7.3 TERCER OBJETIVO

El tercer objetivo se cumplió mediante el diseño y la implementación en la FPGA de los dos autómatas, el lineal y el no lineal, la implementación de la máquina de control y de más periféricos para la comunicación con el PC.

Los dos autómatas se implementaron directamente sobre la FPGA, mientras la máquina de control se diseño sobre el procesador NIOS de Altera, lo que permitió corregir los errores que se presentaron en el primer prototipo.

La implementación de autómatas consume bastante recursos sobre una FPGA, ya que la arquitectura no está directamente orientada a este tipo de aplicaciones, en este caso, sobre una FPGA Stratix II de altera, se implementaron 256 celdas del autómata lineal con 256 estados. Y 256 celdas del autómata no lineal. La limitante del hardware radicó en el número de multiplicadores disponibles en la STRATIX II, a pesar de que esta FPGA está diseñada para aplicaciones de DSP dispone de un número considerable de multiplicadores (255), no son insuficientes para aplicaciones de alta densidad como los autómatas.

El consumo Total de recursos de la FPGA fue de aproximadamente el 80%, ello evidencia en primera instancia que la estructura y granularidad de la FPGA aunque es concurrente y dispone de muchos bloques funcionales, no es lo suficientemente densa para la implementación de autómatas de alta densidad.

Es importante subrayar que le procesador NIOS implementado sobre la FPGA, al ser secuencial hace el lento el proceso total del sistema, sin embargo permitió un mecanismo más sencillo para el proceso de comunicación y control general del autómata.

Debido a la ralentización producida por el NIOS, no sería la herramienta adecuada para la implementación de la máquina genética dentro de la FPGA, en su lugar se debe implementar la máquina genética directamente en primitivas sobre la FPGA.

7.4 CUARTO OBJETIVO

La validación del autómata se logró a partir de la implementación de un proceso que requiera de operaciones masivas en dos dimensiones, para ello se tomó el proceso de filtrado de imágenes como mecanismo de validación, sin embargo cabe anotar que entre los objetivos del proyecto no se tiene el de mejorar o validar

las técnicas de filtrado de imágenes, simplemente se emplea como mecanismo de prueba del autómata.

Los resultados obtenidos por el autómatas son muy similares a los obtenidos directamente por Matlab, sin embargo el número de pulsos de reloj que requiere el autómata es muy pequeño en comparación con las empleadas por una máquina secuencial, para el caso del autómata lineal de 256 estados solo requiere 9 pulsos de reloj, mientras que para el autómata no lineal consume apenas dos pulsos.

8 CONCLUSIONES

- La selección de la función de aptitud determina que la población converja y por tanto que la máquina llegue a obtener un individuo óptimo que cumpla con los requerimientos funcionales del autómata.
- En el caso de operaciones lineales que involucren multiplicaciones y un número considerable de estados, la implementación consume bastantes recursos sobre la FPGA, en cuyo caso se debe trabajar con FPGAs orientadas al procesamiento de señales.
- Cuando se trata de autómatas binarios o de dos estados, las implementaciones pueden ser de un tamaño mucho mayor, en el caso de la STRATIX puede significar un total de 498000 celdas.
- Es posible implementar hardware reconfigurable sobre la FPGA sin la necesidad de reprogramar el dispositivo, sin embargo seleccionar la configuración de las tablas de búsqueda puede resultar en una tarea muy compleja.
- Se concluye que los autómatas celulares se constituyen en una herramienta poderosa para operaciones de gran complejidad como las involucradas en el procesamiento de señales.
- La FFT no es la única alternativa para realizar procesos de filtrado, la NTT se constituye en una alternativa muy poderosa con la ventaja de que al trabajar sobre enteros permite implementaciones directas sobre hardware digital.
- Para el procesamiento de señales sobre Autómatas celulares implementados sobre FPGAs es crucial implementar una interface de comunicación de mayor velocidad que un puerto serial ya que los procesos seriales se convierten en un cuello de botella para alimentar el autómata.
- El Autómata implementado permite realizar funciones no isomorfas, lo que permitiría aplicar filtros distintos en cada región.
- El autómata no lineal además permite realizar operaciones no linéales como la ecualización.

9 BIBLIOGRAFIA

[1] Sekanina L. “Evolvable Components: from theory to hardware implementations”. Ed. New York: Springer, 2004.

[2] Greenwood G., Tyrrel A. “Introduction to evolvable hardware”. Ed. New Jersey: Wiley Interscience, 2007.

[3] El-Haik B., Yang K. “The components of complexity in engineering design”. *IIE Trans.*, vol. 31, No 10, pp. 925-934, Aug. 1998.

[4] Sipper M., Sánchez E., Manege D., Tomassini M., Pérez-Uribe A., Stauffer A. “A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired hardware Systems”. IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, pp. 83-97, Abr. 1997.

[5] Joel L. Schiff, “Cellular Automata: A Discrete View of the World”. New York Wiley 2008

[6] Olu M. “Cellular Autómata Transforms: Theory and Applications in multimedia Compression, Encryption and Modeling”. Ed. Boston Dordrecbt London : Kluwer Academic Publishers, 2000.

[7] Haykin S., Kosko B. “Intelligent signal Procesing”. Ed. New Jersey: Wiley- IEEE, 2001.

[8] Martin O., Odlizko A., Wolfram S. “Algebraic properties of cellular automata”. Communications in Mathematical Physics. 93, pp. 219-258. 1984.

[9] Sipper, M. (1997). Cellular Automata. En M. Sipper, Evolution of Parallel Cellular Machines, The Cellular Programming Approach, pp. 3. Springer.

[10] Xiaonong Ran and K. J. Ray Liu, Senior Member, IEEE. Fast Algorithms for 2-D Circular Convolutions and Number Theoretic Transforms Based on Polynomial Transforms Over Finite Rings. IEEE transactions on signal processing. vol. 43. no. 3. march 1995

[11] Chun-Ling Chang, Yun-Jif, Zhang, Yun-Yin Gdong. Cellular autómata for edge detection of images. Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, 26-29 August 2004

[12] Hilton Tamanaha Goi. An Original Method of Edge Detection Based on Cellular Automata. Homework for EE817 Emerging Computing Technologies, Prof. Marek Perkowski. June 3rd 2003

[13] G.A. Jullien. Number Theoretic Transforms - Extract from Number Theoretic Techniques in Digital Signal Processing Book Chapter Advances in Electronics and Electron Physics, Academic Press. Inc., (Invited), vol. 80, Chapter 2, pp. 69-163, 1991.

[14] Tood K. Moon. Error Correction Coding. Wiley Press 2005

[15] John G. Proakis, Dimitris K Manolakis. Digital Signal Processing (4th Edition). Prentice Hall 2006.

[16] Steve Kilts. Advanced FPGA Design: Architecture, Implementation, and Optimization. Wiley-IEEE Press; 1 edition -June 29, 2007.

[17] S.N. Sivanandam, S.N. Deepa. Introduction to genetic Algorithms. Springer 2010

[18] Sáenz Cabezas, Brayan Jair. Jairo Jonny Hidalgo Obando. Implementación de autómatas celulares sobre FPGA. Proyecto de Grado Fundación Universitaria Los Libertadores. Facultad de Ingeniería. Programa de Ingeniería Electrónica, 2011.

[19] Fogel, D. B. (Abril de 1997). Evolutionary Computation: A New Transactions. IEEE Transactions on Evolutionary Computation , Vol. 1 (Issue 1.).

[20] Kari, J. (Spring - 2011). Celullar Automata. University of Turku.

[21] Newmann, J. v. (1966). Theory of Self-Reproducing Automata. Illinois: Univ. of Illinois Press.

[22] Przemyslaw, P., & Lindemayer, A. (1996). The Algorithmic Beauty of Plants. Spinger-Veralag.

[23] Purves, W. K., Sadava, D., Orians, G. H., & Heller, H. C. (2003). Life: The Science of Biology. W.H. Freeman.

[24] Salem Zebulum, R., Pacheco, M. A., & Vellasco, M. M. (2001). Evolutionary Electronics (Primera Edición ed.). CRC Press.

[25] Schiff, J. L. (2007). Cellular Autómata - A Discrete View of the World. John Wiley & Sons.

[26] Sipper, M. (1997). Cellular Automata. En M. Sipper, Evolution of Parallel Cellular Machines, The Cellular Programming Approach (pág. 3). Springer.

[27] Pollard, J. M. (1971). The Fast Fourier Transform in a finite field. Mathematic on Computation, volumen 25, number 114.

[28] Trifonov, J. V., Fedorenko S. V. (2003). A Method for Fast Computation of the Fourier Transform over a Finite Field. Problems of Information Transmision, volumen 39, number 3.

[29] Yang Xuan, Liang Dequn (1997). Multiscale edge detection based on direction information. Journal of Xidian University. Vol. number 4.

10 ANEXOS

Anexo 1: Figura 8. Autómata en dos dimensiones.

```
W=ones(32,32);                          %matriz de inicio
n=8;                                    %número de divisiones de la malla
A=ones(n*size(W));                      %matriz imagen
AA=ones(n,n);                           %matriz auxiliar
N=30;                                   %cantidad de imágenes
L=2*(length(W));                        %=64
%**********************construcción de las bolas***********************
%************************centro de cada bola**************************
ro=2*n-(n/2);
xo=2*n;
yo=2*n;
%bola
for i=1:length(W)
for j=1:length(W)
      p=(i-xo)^2;
      r=(j-yo)^2;
if (p+r)<ro^2
         W(i,j)=0;
end
end
end
%**************  grafica los círculos en posiciones aleatorias (primera imagen)********
z=randn(n);
for i=1:n
for j=1:n
if abs(z(i,j))<0.3
         AA(i,j)=0;
end
end
end
A=relleno(A,AA,W,n);
A=m_par(A,L);
figure(1),imshow(A)
title(1)
%*****************gráfica del resto de imágenes*****************************
for i=2:N
   A=ones(size(A));                     %inicializo A
if mod(i,2)==0                          %grafica con malla impar
for k=1:n/2                             %siguiente posición en matriz auxiliar
for l=1:n/2
            a=AA(2*k-1:2*k,2*l-1:2*l);
            b=next1(a);
            AA(2*k-1:2*k,2*l-1:2*l)=b;
```

```
end
end
        A=relleno(A,AA,W,n);
        A=m_impar(A,L);
else                        %grafica con malla par
        BB=ones(size(AA)+2);
        BB(2:n+1,2:n+1)=AA;
for k=1:n/2+1
for l=1:n/2+1
                a=BB(2*k-1:2*k,2*l-1:2*l);
                b=next1(a);
                BB(2*k-1:2*k,2*l-1:2*l)=b;
end
end
        AA=BB(2:n+1,2:n+1);
        A=relleno(A,AA,W,n);
        A=m_par(A,L);
end
    pause(1)
    imshow(A)
    title(i)
end
```

Anexo 2: Algoritmo generador del campo de Galois

```
function [Ab,Ad,Ae] = G_galois_F(poldec)

% Esta función genera la matriz del campo de Galois para un campo 2^m y con
% polinomio irreducible, el polinomio entre en notación decimal "poldec"
% la salida se entrega como una matriz ordenada desde -, a^0,a^1,a^2,...a^(2^(m)-2)
% Ad -> Elementos del campo en representación decimal
% Ab -> Elementos del campo en representación binaria
% Ab -> Matriz de potencias

pol=dec2bin(poldec);          % se obtiene el polinomio en binario a partir de poldec
r=length(pol);                % se obtiene la longitud de pol
m=r-1;                        % m el número de bits del código
M=2^(m);                      % M el número de filas del código
resp=bitxor(M,poldec);        % Xor binaria entre M y poldec, resultado en decimal
res=dec2bin(resp);            % res = binario de resp

% ------------------------------------------------- %
% -- Crea los elementos en representación binaria
% ------------------------------------------------- %
Ab=char(48*ones(M,m));        % Se crea una matriz de Mxm para el código de salida en
representación binaria
a=1;                          % alfa a^0=1
```

```
b=a;                          % b = copia de a
Ab(2,:)=dec2bin(b,m);         % se guarda el segundo dato (alfa) en la tabla del código
for i=3:M                     % se inicia a llenar la tabla del código desde i=3
if Ab(i-1,1)=='0';            % si el valor del código anterior, en la posición más pesada
es 0
      b= bin2dec(Ab(i-1,:));   % (para poder realizar la operación shift es necesario
convertir a dec)
      x=bitshift(b,1);        % se corre la palabra - equivale a multiplicar por alfa (a)
      Ab(i,:)=dec2bin(x,m);   % se guarda en la tabla
else
      d=Ab(i-1,:);            % se lee el dato anterior de la tabla;
      d(1,1)='0';             % se coloca cero en la posición más pesada
      dd=bin2dec(d);          % se convierte a decimal a c
      dds=bitshift(dd,1);     % se corre a la derecha un bit a cd
      dc= bitxor(dds,resp);   % Xor entre cds y el residuo del polinomio resp
      Ab(i,:)= dec2bin(dc,m); % se convierte a binario a cc en m bits y se guarda en C
end
end
% ------------------------------------------------------------------ %
% -- Crea la matriz de los exponentes de los elementos del campo -- %
% ------------------------------------------------------------------ %
D=zeros(M,1);                 % se crea la matriz de potencias
for i=2:M
   d=bin2dec(Ab(i,:));        % se lee el código y se convierte a decimal
   D(d+1)=i-2;
end
% Ae=D(2:M);                  % Matriz de los exponentes (salvo el elemento cero)
Ae=(0:M-2)';
% ---------------------------------------------------- %
% -- Elementos del campo en representación decimal
% ---------------------------------------------------- %
Ad=zeros(M,1);                % Se crea la matriz para almacenar la representación
decimal de los elementos
for i=1:M
   Ad(i)=bin2dec(Ab(i,:));    % Pasa a decimal los elementos del campo
end
% ---------------------------------------------------- %
% -- Convierte a 'double' para manipulación numérica -- %
% ---------------------------------------------------- %
Ab=double(Ab);                % Se pasa de 'char' a 'double' la matriz de representación
binaria para la manipulación numérica
Ab=Ab-48;                     % 48 es la representación decimal del cero en ASCII
end
```

Anexo 3:Código FFT

```
function Y=fftivan(x)
```

```
% transformada de Fourier FFT radix 2 para señales de longitud 2^k
% la salida se entrega en una matriz de N/2 filas por dos columnas
N=length(x);                    %N = longitud del vector x debe ser de la forma 2^k
xx=decit(x);                    % el algoritmo decit realiza la declinación de x
k=log2(N);
W2=[1 1; 1 -1];                 % primera etapa de las mariposas para FFT2
yy=(W2*xx')';                   % se calcula la primera etapa de la FFT

for i=1:(k-1)                   % ejecuta (k-1) etapas adicionales para la FFT
   W=GWN(N,i);                  % Genera la matriz de factores Twidle de cada etapa
   yy=W.*yy;                    % aplica los factores Twidle a cada salida de la etapa
anterior
   M=GMM(N,i);                  % genera las mariposas de cada etapa
   yy=M*yy;                     % obtiene la siguiente transformada
end
Y=yy;
```

Código de diezmado

```
function xx = decit(x)
% algoritmo de diesmado, la entrada debe ser un vector fila de tamaño N=2^p (con p
entero)
% la salida xx corresponde a una matriz de tamaño (N/2,2)
% cada fila (de dos elementos) es la entrada para las Butterfly
% en ap se crean los apuntadores para guardar x decimado en xx
N=length(x);
xx=zeros(N/2,2);
k=log2(N/2);
ap=zeros(N/2,k);
for i=1:N/2
   ap(i,k:-1:1) = de2bi(i-1,k);  % se crean los apuntadores y se invierten
end
for i=1:N/2
   xx(i,1) = x(bi2de(ap(i,:))+1);% se leen los datos de x y se guardan en xx (primera
columna)
   xx(i,2) = x(bi2de(ap(i,:))+1 +N/2); % se guardan los datos de x para la segunda columna
end
```

Código generador de matrices de Twidle

```
function W = GWN(N,I)
% función para la generación de las matrices de Twidle
% N corresponde al tamaño del vector x de entrada,
% I es la potencia o etapa de la FFT a desarrollar
Aux=zeros(N/2,2);
W=zeros(N/2,2);
```

```
M=2^(I+1);                          % se calcula el M para el factor de Twidle
R=2^(I-1);                          % R = número de repeticiones de la matriz básica
WN=exp(-j*2*pi/M);                  % factores de Twidle

for i=1:N/2
    Aux(i,1)=mod(floor((i-1)/R),2);  % Aux se emplea para mostrar la matriz
    Aux(i,2)=mod((i-1),R)*2;
    r=Aux(i,1);
    k=Aux(i,2);
    W(i,1)=WN^(r*k);
    W(i,2)=WN^(r*(k+1));
end
```

Construccion de las “Butterfly”

```
function W = GMM(N,I)
% función para la generación de la matriz MM (mariposas)
M=N/2;
A=2^I;
B=A/2;
W=zeros(M,M);
for i=1:M/2
      ap=A*floor((i-1)/B)+mod((i-1),B)+1;
      W(ap,ap)=1;
      W(ap,ap+B)=1;
      W(ap+B,ap)=1;
      W(ap+B,ap+B)=-1;
end
```

Anexo 4: Algoritmo de aptitud

```
function [apt,vj]=aptitud(HX,HJ)
% ingresa el histograma de la imagen original HJ
% ingresa el histograma de la imagen resultante HX
mx=mean(HX);
mj=mean(HJ);
vx=dot((HX-mx),(HX-mx))/(mx^2);
vj=dot((HJ-mj),(HJ-mj))/(mj^2);
apt=vx;
```

Anexo 5: Autómata de Wolfram

```
function y=automata(B,R,Q)
% B es la imagen binaria de entrada
% se tratan los bordes con autómata celular según la regla R
% la regla entra en forma de un polinomio binario
% la regla es un string de 10 bits - máscara 3 bits
% Realiza la convolución en 2 dimensiones y
% se actualiza el valor de la celda según la regla
% en Q entra en el número de evoluciones
[N,M]=size(B);
w=zeros(N+(2),M+(2));
w(2:N+1,2:M+1)=B;
y=zeros(N,M);
for k=1:Q

for i=2:N+1
for j=2:M+1
          sa=sum(sum(w(i-1:i+1, j-1:j+1)));
switch sa
case 0
                y(i-1,j-1)=R(1);
case 1
                y(i-1,j-1)=R(2);
case 2
                y(i-1,j-1)=R(3);
case 3
                y(i-1,j-1)=R(4);
case 4
                y(i-1,j-1)=R(5);
case 5
                y(i-1,j-1)=R(6);
case 6
                y(i-1,j-1)=R(7);
case 7
                y(i-1,j-1)=R(8);
case 8
                y(i-1,j-1)=R(9);
case 9
                y(i-1,j-1)=R(10);
end
end
end
   w=zeros(N+(2),M+(2));
   w(2:N+1,2:M+1)=y;
end
```

Anexo 6: Descripción en VHDL del Módulo "Mux_vecinos". Fuente. Autor, obtenido con “MegaWizard Plugin-Manager” – Quartus II

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;
--------------------------------------------------------------------------------
-----------
entity Mux_Vecinos is
	Port (	data0x		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		data1x		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		data2x		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		data3x		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		data4x		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		data5x		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		data6x		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		data7x		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		data8x		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		sel		: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
		result		: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
end Mux_Vecinos;
--------------------------------------------------------------------------------
-----------
ARCHITECTURE SYN OF mux_vecinos IS

--	type STD_LOGIC_2D is array (NATURAL RANGE <>, NATURAL RANGE <>) of
STD_LOGIC;

	SIGNAL sub_wire0	: STD_LOGIC_VECTOR (7 DOWNTO 0);
	SIGNAL sub_wire1	: STD_LOGIC_VECTOR (7 DOWNTO 0);
	SIGNAL sub_wire2	: STD_LOGIC_2D (8 DOWNTO 0, 7 DOWNTO 0);
	SIGNAL sub_wire3	: STD_LOGIC_VECTOR (7 DOWNTO 0);
	SIGNAL sub_wire4	: STD_LOGIC_VECTOR (7 DOWNTO 0);
	SIGNAL sub_wire5	: STD_LOGIC_VECTOR (7 DOWNTO 0);
	SIGNAL sub_wire6	: STD_LOGIC_VECTOR (7 DOWNTO 0);
	SIGNAL sub_wire7	: STD_LOGIC_VECTOR (7 DOWNTO 0);
	SIGNAL sub_wire8	: STD_LOGIC_VECTOR (7 DOWNTO 0);
	SIGNAL sub_wire9	: STD_LOGIC_VECTOR (7 DOWNTO 0);
	SIGNAL sub_wire10	: STD_LOGIC_VECTOR (7 DOWNTO 0);

BEGIN

sub_wire10    <= data0x(7 DOWNTO 0);
sub_wire9    <= data1x(7 DOWNTO 0);
sub_wire8    <= data2x(7 DOWNTO 0);
sub_wire7    <= data3x(7 DOWNTO 0);
sub_wire6    <= data4x(7 DOWNTO 0);
sub_wire5    <= data5x(7 DOWNTO 0);
sub_wire4    <= data6x(7 DOWNTO 0);
sub_wire3    <= data7x(7 DOWNTO 0);
result    <= sub_wire0(7 DOWNTO 0);
sub_wire1    <= data8x(7 DOWNTO 0);
sub_wire2(8, 0)    <= sub_wire1(0);
sub_wire2(8, 1)    <= sub_wire1(1);
sub_wire2(8, 2)    <= sub_wire1(2);
sub_wire2(8, 3)    <= sub_wire1(3);
sub_wire2(8, 4)    <= sub_wire1(4);
```

```
sub_wire2(8, 5)    <= sub_wire1(5);
sub_wire2(8, 6)    <= sub_wire1(6);
sub_wire2(8, 7)    <= sub_wire1(7);
sub_wire2(7, 0)    <= sub_wire3(0);
sub_wire2(7, 1)    <= sub_wire3(1);
sub_wire2(7, 2)    <= sub_wire3(2);
sub_wire2(7, 3)    <= sub_wire3(3);
sub_wire2(7, 4)    <= sub_wire3(4);
sub_wire2(7, 5)    <= sub_wire3(5);
sub_wire2(7, 6)    <= sub_wire3(6);
sub_wire2(7, 7)    <= sub_wire3(7);
sub_wire2(6, 0)    <= sub_wire4(0);
sub_wire2(6, 1)    <= sub_wire4(1);
sub_wire2(6, 2)    <= sub_wire4(2);
sub_wire2(6, 3)    <= sub_wire4(3);
sub_wire2(6, 4)    <= sub_wire4(4);
sub_wire2(6, 5)    <= sub_wire4(5);
sub_wire2(6, 6)    <= sub_wire4(6);
sub_wire2(6, 7)    <= sub_wire4(7);
sub_wire2(5, 0)    <= sub_wire5(0);
sub_wire2(5, 1)    <= sub_wire5(1);
sub_wire2(5, 2)    <= sub_wire5(2);
sub_wire2(5, 3)    <= sub_wire5(3);
sub_wire2(5, 4)    <= sub_wire5(4);
sub_wire2(5, 5)    <= sub_wire5(5);
sub_wire2(5, 6)    <= sub_wire5(6);
sub_wire2(5, 7)    <= sub_wire5(7);
sub_wire2(4, 0)    <= sub_wire6(0);
sub_wire2(4, 1)    <= sub_wire6(1);
sub_wire2(4, 2)    <= sub_wire6(2);
sub_wire2(4, 3)    <= sub_wire6(3);
sub_wire2(4, 4)    <= sub_wire6(4);
sub_wire2(4, 5)    <= sub_wire6(5);
sub_wire2(4, 6)    <= sub_wire6(6);
sub_wire2(4, 7)    <= sub_wire6(7);
sub_wire2(3, 0)    <= sub_wire7(0);
sub_wire2(3, 1)    <= sub_wire7(1);
sub_wire2(3, 2)    <= sub_wire7(2);
sub_wire2(3, 3)    <= sub_wire7(3);
sub_wire2(3, 4)    <= sub_wire7(4);
sub_wire2(3, 5)    <= sub_wire7(5);
sub_wire2(3, 6)    <= sub_wire7(6);
sub_wire2(3, 7)    <= sub_wire7(7);
sub_wire2(2, 0)    <= sub_wire8(0);
sub_wire2(2, 1)    <= sub_wire8(1);
sub_wire2(2, 2)    <= sub_wire8(2);
sub_wire2(2, 3)    <= sub_wire8(3);
sub_wire2(2, 4)    <= sub_wire8(4);
sub_wire2(2, 5)    <= sub_wire8(5);
sub_wire2(2, 6)    <= sub_wire8(6);
sub_wire2(2, 7)    <= sub_wire8(7);
sub_wire2(1, 0)    <= sub_wire9(0);
sub_wire2(1, 1)    <= sub_wire9(1);
sub_wire2(1, 2)    <= sub_wire9(2);
sub_wire2(1, 3)    <= sub_wire9(3);
sub_wire2(1, 4)    <= sub_wire9(4);
sub_wire2(1, 5)    <= sub_wire9(5);
sub_wire2(1, 6)    <= sub_wire9(6);
sub_wire2(1, 7)    <= sub_wire9(7);
sub_wire2(0, 0)    <= sub_wire10(0);
```

```
sub_wire2(0, 1)    <= sub_wire10(1);
sub_wire2(0, 2)    <= sub_wire10(2);
sub_wire2(0, 3)    <= sub_wire10(3);
sub_wire2(0, 4)    <= sub_wire10(4);
sub_wire2(0, 5)    <= sub_wire10(5);
sub_wire2(0, 6)    <= sub_wire10(6);
sub_wire2(0, 7)    <= sub_wire10(7);

lpm_mux_component : lpm_mux
GENERIC MAP (       lpm_size => 9,
                    lpm_type => "LPM_MUX",
                    lpm_width => 8,
                    lpm_widths => 4)
PORT MAP (   sel => sel,
             data => sub_wire2,
             result => sub_wire0);
-------------------------------------------------------------------------------
-----------
end SYN;
```

Anexo 7: Descripción en VHDL del Módulo "Mod_Memoria". Fuente. Autor, obtenido con “MegaWizard Plugin-Manager” – Quartus II

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera_mf;
USE altera_mf.all;
-------------------------------------------------------------------------------
-----------
ENTITY Mod_Memoria IS
       PORT (       address      : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
                    clock        : IN STD_LOGIC ;
                    data         : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    wren         : IN STD_LOGIC ;
                    q            : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END Mod_Memoria;
-------------------------------------------------------------------------------
-----------
ARCHITECTURE SYN OF mod_memoria IS

       SIGNAL sub_wire0    : STD_LOGIC_VECTOR (7 DOWNTO 0);

       COMPONENT altsyncram
       GENERIC (    clock_enable_input_a             : STRING;
                    clock_enable_output_a            : STRING;
                    intended_device_family           : STRING;
                    lpm_type                         : STRING;
                    numwords_a
                    operation_mode                   : STRING;
                    outdata_aclr_a                   : STRING;
                    outdata_reg_a                    : STRING;
                    power_up_uninitialized           : STRING;
                    ram_block_type                   : STRING;
                    widthad_a
                    width_a
                    width_byteena_a
       PORT ( wren_a       : IN STD_LOGIC ;
              clock0       : IN STD_LOGIC ;
```

```
              address_a     : IN STD_LOGIC_VECTOR (5 DOWNTO 0);
              q_a           : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
              data_a        : IN STD_LOGIC_VECTOR (7 DOWNTO 0));
       END COMPONENT;

BEGIN

q    <= sub_wire0(7 DOWNTO 0);

altsyncram_component : altsyncram
GENERIC MAP (       clock_enable_input_a => "BYPASS",
                    clock_enable_output_a => "BYPASS",
                    intended_device_family => "Stratix II",
                    lpm_type => "altsyncram",
                    numwords_a => 64,
                    operation_mode => "SINGLE_PORT",
                    outdata_aclr_a => "NONE",
                    outdata_reg_a => "UNREGISTERED",
                    power_up_uninitialized => "FALSE",
                    ram_block_type => "M512",
                    widthad_a => 6,
                    width_a => 8,
                    width_byteena_a => 1)
PORT MAP (   wren_a => wren,
             clock0 => clock,
             address_a => address,
             data_a => data,
             q_a => sub_wire0);
--------------------------------------------------------------------------------
-----------
END SYN;
```

Descripción en VHDL del Módulo "Multiplicador" Fuente. Autor, obtenido con “MegaWizard Plugin-Manager” – Quartus II

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.all;
--------------------------------------------------------------------------------
ENTITY Multiplicador IS
       PORT ( dataa        : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    datab        : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    result       : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
END Multiplicador;
--------------------------------------------------------------------------------
ARCHITECTURE SYN OF multiplicador IS

SIGNAL sub_wire0    : STD_LOGIC_VECTOR (15 DOWNTO 0);

COMPONENT lpm_mult
GENERIC (    lpm_hint            : STRING;
             lpm_representation  : STRING;
             lpm_type            : STRING;
             lpm_widtha          : NATURAL;
             lpm_widthb          : NATURAL;
             lpm_widthp          : NATURAL);
PORT ( dataa  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
```

```
      datab  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      result : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
END COMPONENT;

BEGIN

result    <= sub_wire0(15 DOWNTO 0);

lpm_mult_component : lpm_mult
GENERIC MAP (       lpm_hint                                                    =>
"DEDICATED_MULTIPLIER_CIRCUITRY=YES,MAXIMIZE_SPEED=5",
                    lpm_representation => "SIGNED",
                    lpm_type => "LPM_MULT",
                    lpm_widtha => 8,
                    lpm_widthb => 8,
                    lpm_widthp => 16)
PORT MAP (    dataa => dataa,
              datab => datab,
              result => sub_wire0);
-------------------------------------------------------------------------------
END SYN;
```

Anexo 8. Cableado por código de los módulos componentes. Acumulador y registro del acumulador.

Cableado de los módulos

```
-- Module Name : Acumulador de 16 bits
-------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-------------------------------------------------------------------------------
-----------
entity Acumulador is
      Port (       reset : in STD_LOGIC;
                   ck_en : in STD_LOGIC;
                   x : in STD_LOGIC_VECTOR(15 downto 0);
                   Acc : out STD_LOGIC_VECTOR(15 downto 0));
end Acumulador;
-------------------------------------------------------------------------------
-----------
architecture arq_Acumulador of Acumulador is

-- ============================= --
-- == Componentes del Módulo == --
-- ============================= --
--> Sumador : Módulo que realiza la suma entre los datos que ingresan al módulo
component Acumulador_2 is
      Port (       dataa        : IN STD_LOGIC_VECTOR (15 downto 0);
                   datab        : IN STD_LOGIC_VECTOR (15 downto 0);
                   result       : OUT STD_LOGIC_VECTOR (15 downto 0));
end component;
--> Registro del Acumulador : Almacena los datos que resultan de la acumulación
component Reg_Acumulador is
      Port (       clk : in STD_LOGIC;
                   reset : in STD_LOGIC;
                   Di : in STD_LOGIC_VECTOR(15 downto 0);
```

```
                    Do : out STD_LOGIC_VECTOR(15 downto 0));
end component;
-- ========================= --
-- == Señales de Conexión == --
-- ========================= --
signal acc_reg : STD_LOGIC_VECTOR(15 downto 0);
signal acc_next : STD_LOGIC_VECTOR(15 downto 0);

begin

A0 : Acumulador_2
port map (   dataa => x,
             datab => acc_reg,
             result => acc_next);
--> Registro Acc
Accum: Reg_Acumulador
port map (   clk => ck_en,
             reset => reset,
             Di => acc_next,
             Do => acc_reg);
Acc<=acc_reg;
-------------------------------------------------------------------------------
-----------
end arq_Acumulador;
```

Código en VHDL del módulo "Acumulador_2"

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.all;
-------------------------------------------------------------------------------
-----------
ENTITY Acumulador_2 IS
      PORT (       dataa        : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                   datab        : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                   result       : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
END Acumulador_2;
-------------------------------------------------------------------------------
-----------
ARCHITECTURE SYN OF acumulador_2 IS

      SIGNAL sub_wire0    : STD_LOGIC_VECTOR (15 DOWNTO 0);
      COMPONENT lpm_add_sub
      GENERIC (    lpm_direction       : STRING;
                   lpm_hint            : STRING;
                   lpm_representation        : STRING;
                   lpm_type            : STRING;
                   lpm_width           : NATURAL);
      PORT (       dataa  : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                   datab  : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                   result : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
      END COMPONENT;

BEGIN

result    <= sub_wire0(15 DOWNTO 0);

lpm_add_sub_component : lpm_add_sub
```

```
GENERIC MAP (       lpm_direction => "ADD",
                    lpm_hint => "ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO",
                    lpm_representation => "SIGNED",
                    lpm_type => "LPM_ADD_SUB",
                    lpm_width => 16)
PORT MAP (   dataa => dataa,
             datab => datab,
             result => sub_wire0);
--------------------------------------------------------------------------------
-----------
END SYN;
```

Para hacer uso de las primitivas de altera, es necesaria la librería de primitivas del altera, denominada “altera.altera_primitives_components”, como se observa en las primeras líneas del código mostrado a continuación.

Descripción en VHDL del Registro del Acumulador

```
-- Module Name : Registro_ACC
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
LIBRARY altera;
USE altera.altera_primitives_components.all;
--------------------------------------------------------------------------------
-----------entity Reg_Acumulador is
       Port (       clk : in STD_LOGIC;
                    reset : in STD_LOGIC;
                    Di : in STD_LOGIC_VECTOR(15 downto 0);
                    Do : out STD_LOGIC_VECTOR(15 downto 0));
end Reg_Acumulador;
--------------------------------------------------------------------------------
-----------architecture arq_Reg_Acumulador of Reg_Acumulador is

signal s_reset : STD_LOGIC;

begin

s_reset <= not reset;

--> Registro para el estado actual y el siguiente estado de la celda
R: for i in 0 to 15 generate
begin
       C: DFF
       port map (   d => Di(i),
                    clk => clk,
                    clrn => s_reset,
                    prn => '1',
                    q => Do(i));
end generate;
--------------------------------------------------------------------------------
-----------end arq_Reg_Acumulador;
```

Anexo 9: Módulo Valor Absoluto - Código en VHDL, Comparadores MOI.

```
-- Module Name : Valor Absoluto - Medida de la Orientación de la Información
----------------------------------------------------------------------------------
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
----------------------------------------------------------------------------------
--
entity Valor_Absoluto is
      port (       Dato_a : in STD_LOGIC_VECTOR(15 downto 0);
                   Dato_b : out STD_LOGIC_VECTOR(15 downto 0));
end Valor_Absoluto;
----------------------------------------------------------------------------------
--
architecture arq_Valor_Absoluto of Valor_Absoluto is

-- ========================= --
-- == Señales de Conexión == --
-- ========================= --
constant uno : STD_LOGIC_VECTOR(15 downto 0):=(0=>'1',others=>'0');
signal carry : STD_LOGIC_VECTOR(16 downto 0);
signal suma : STD_LOGIC_VECTOR(15 downto 0);
signal x : STD_LOGIC_VECTOR(15 downto 0);

begin

--> Remueve el complemento a dos del dato negando el mismo y sumándole 'uno'
x <= not Dato_a; -- Niega el dato

Sum: for i in 0 to 15 generate
begin
      -- Realiza la suma, implementada como un sumador completo de 16 bits
      carry(0)<='0';                          -- carry inicial de la suma
      suma(i)<=x(i) xor uno(i) xor carry(i);  -- Valor de la suma
      carry(i+1)<=(x(i)  and  uno(i))  or  (carry(i)  and  x(i))  or  (carry(i)  and
uno(i));
                                              -- Valor del carry
end generate;

--> Examina el bit de signo para aplica o no la remoción del complemento a dos
with Dato_a(15) select
      Dato_b <=     Dato_a when '0',    -- El dato es positivo, luego para sin
                                        -- alterar
                    suma when others;   -- El dato es negativo, luego remueve el
                                        -- complemento a dos
----------------------------------------------------------------------------------
--
end arq_Valor_Absoluto;
```

Código en VHDL de los Comparadores MOI

```
-- Module Name : Comparadores - Medida de la Orientación de la Información
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library altera; -- Librerías para el uso de primitivas de hardware altera
```

```
use altera.altera_primitives_components.ALL;
-------------------------------------------------------------------------------
---------
entity Comparadores_MOI is
	port (		clk : in STD_LOGIC;
			reset : in STD_LOGIC;
			orient_info : STD_LOGIC;
			Entrada : in STD_LOGIC_VECTOR(15 downto 0);
			Salida : out STD_LOGIC_VECTOR(15 downto 0));
end Comparadores_MOI;
-------------------------------------------------------------------------------
---------
architecture arq_Comparadores_MOI of Comparadores_MOI is

-- ------------------------- --
-- == Señales de Conexión == --
-- ------------------------- --
signal s_reset : STD_LOGIC;
signal s1 : STD_LOGIC_VECTOR(15 downto 0);
signal s2 : STD_LOGIC_VECTOR(15 downto 0);

begin

s_reset <= not reset;

-- ------------ --
-- == Máximo == --
-- ------------ --
--> Comparador, comprara entre la entrada y la señal del registro, 's1'
process(Entrada,s1)
begin
if(Entrada>=s1) then
	s2<=Entrada;
else
	s2<=s1;
end if;
end process;
-- -------------- --
-- == Registro == --
-- -------------- --
-- Crea el registro mediante la implementación de primitivas de hardware, flip-
flops tipo D
-- con enable, todos conectados en paralelo. Se crea un registro de 16 bits
R0: for i in 0 to 15 generate
begin
	C0: DFFE
	port map (	d => s2(i),
			clk => clk,
			clrn => s_reset,
			prn => '1',
			ena => orient_info,
			q => s1(i));
end generate;
Salida <= s1; -- Señal de Salida (Máximo de las Máscaras MOI)
-------------------------------------------------------------------------------
---------
end arq_Comparadores_MOI;
```

Anexo 10. Descripción en VHDL del Módulo de Binarizado

```
-- Module Name : "Binarización" de loa datos
----------------------------------------------------------------------------------
----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
----------------------------------------------------------------------------------
-----------
entity Binarizado is
       Port (       dato_a : in STD_LOGIC;
                    dato_b : out STD_LOGIC_VECTOR(7 downto 0));
end Binarizado;
----------------------------------------------------------------------------------
-----------
architecture arq_Binarizado of Binarizado is

begin

--> El 'dato_a' es el bit de entrada que determina si la salida es uno o cero
with dato_a select
       dato_b <=     (others=>'0') when '0',          -- El dato está por debajo
del umbral
                     (others=>'1') when others; -- El dato supera el umbral
----------------------------------------------------------------------------------
-----------
end arq_Binarizado;
```

Anexo 11. Multiplexor de Salida - Descripción en VHDL. Fuente. Autor, obtenido con “MegaWizard Plugin-Manager” – Quartus II

```
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 9.0 Build 132 02/25/2009 SJ Full Version
-- ************************************************************
--Copyright (C) 1991-2009 Altera Corporation
--Your use of Altera Corporation's design tools, logic functions
--and other software and tools, and its AMPP partner logic
--functions, and any output files from any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
--to the terms and conditions of the Altera Program License
--Subscription Agreement, Altera MegaCore Function License
--Agreement, or other applicable license agreement, including,
--without limitation, that your use is for the sole purpose of
--programming logic devices manufactured by Altera and sold by
--Altera or its authorized distributors.  Please refer to the
--applicable agreement for further details.
----------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;
```

```
--------------------------------------------------------------------------------
ENTITY Mux_Reg_estado IS
       PORT (       data0x       : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data1x       : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data2x       : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data3x       : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data4x       : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    sel          : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
                    result       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END Mux_Reg_estado;
--------------------------------------------------------------------------------
ARCHITECTURE SYN OF mux_reg_estado IS

--     type  STD_LOGIC_2D  is  array  (NATURAL  RANGE  <>,  NATURAL  RANGE  <>)  of
STD_LOGIC;

SIGNAL sub_wire0    : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL sub_wire1    : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL sub_wire2    : STD_LOGIC_2D (4 DOWNTO 0, 7 DOWNTO 0);
SIGNAL sub_wire3    : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL sub_wire4    : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL sub_wire5    : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL sub_wire6    : STD_LOGIC_VECTOR (7 DOWNTO 0);

BEGIN

sub_wire6    <= data0x(7 DOWNTO 0);
sub_wire5    <= data1x(7 DOWNTO 0);
sub_wire4    <= data2x(7 DOWNTO 0);
sub_wire3    <= data3x(7 DOWNTO 0);
result       <= sub_wire0(7 DOWNTO 0);
sub_wire1    <= data4x(7 DOWNTO 0);
sub_wire2(4, 0)    <= sub_wire1(0);
sub_wire2(4, 1)    <= sub_wire1(1);
sub_wire2(4, 2)    <= sub_wire1(2);
sub_wire2(4, 3)    <= sub_wire1(3);
sub_wire2(4, 4)    <= sub_wire1(4);
sub_wire2(4, 5)    <= sub_wire1(5);
sub_wire2(4, 6)    <= sub_wire1(6);
sub_wire2(4, 7)    <= sub_wire1(7);
sub_wire2(3, 0)    <= sub_wire3(0);
sub_wire2(3, 1)    <= sub_wire3(1);
sub_wire2(3, 2)    <= sub_wire3(2);
sub_wire2(3, 3)    <= sub_wire3(3);
sub_wire2(3, 4)    <= sub_wire3(4);
sub_wire2(3, 5)    <= sub_wire3(5);
sub_wire2(3, 6)    <= sub_wire3(6);
sub_wire2(3, 7)    <= sub_wire3(7);
sub_wire2(2, 0)    <= sub_wire4(0);
sub_wire2(2, 1)    <= sub_wire4(1);
sub_wire2(2, 2)    <= sub_wire4(2);
sub_wire2(2, 3)    <= sub_wire4(3);
sub_wire2(2, 4)    <= sub_wire4(4);
sub_wire2(2, 5)    <= sub_wire4(5);
sub_wire2(2, 6)    <= sub_wire4(6);
sub_wire2(2, 7)    <= sub_wire4(7);
sub_wire2(1, 0)    <= sub_wire5(0);
sub_wire2(1, 1)    <= sub_wire5(1);
sub_wire2(1, 2)    <= sub_wire5(2);
sub_wire2(1, 3)    <= sub_wire5(3);
```

```
sub_wire2(1, 4)    <= sub_wire5(4);
sub_wire2(1, 5)    <= sub_wire5(5);
sub_wire2(1, 6)    <= sub_wire5(6);
sub_wire2(1, 7)    <= sub_wire5(7);
sub_wire2(0, 0)    <= sub_wire6(0);
sub_wire2(0, 1)    <= sub_wire6(1);
sub_wire2(0, 2)    <= sub_wire6(2);
sub_wire2(0, 3)    <= sub_wire6(3);
sub_wire2(0, 4)    <= sub_wire6(4);
sub_wire2(0, 5)    <= sub_wire6(5);
sub_wire2(0, 6)    <= sub_wire6(6);
sub_wire2(0, 7)    <= sub_wire6(7);

      lpm_mux_component : lpm_mux
      GENERIC MAP (       lpm_size => 5,
                          lpm_type => "LPM_MUX",
                          lpm_width => 8,
                          lpm_widths => 3)
      PORT MAP (    sel => sel,
                    data => sub_wire2,
                    result => sub_wire0);
--------------------------------------------------------------------------------
END SYN;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Anexo 12: Código en VHDL del Registro de Estado

```
-- Module Name : Registro de Estado
--------------------------------------------------------------------------------
----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library altera;
use altera.altera_primitives_components.all;
--------------------------------------------------------------------------------
-------
entity Reg_Estado is
      Port (       clk : in STD_LOGIC;
                   reset : in STD_LOGIC;
                   en : in STD_LOGIC;
                   D : in STD_LOGIC_VECTOR(7 downto 0);
                   Q : out STD_LOGIC_VECTOR(7 downto 0));
end Reg_Estado;
--------------------------------------------------------------------------------
-------
architecture arq_Reg_Estado of Reg_Estado is

signal s_reset : STD_LOGIC;

begin

s_reset <= not reset;

R : for i in 0 to 7 generate
begin
      F : DFFE
      port map (   clk => clk,
```

```
                    clrn => s_reset,
                    prn => '1',
                    ena => en,
                    d => D(i),
                    q => Q(i));
end generate R;
-------------------------------------------------------------------------------
-------
end arq_Reg_Estado;
```

Anexo 13: Descripción en VHDL del "Mux_mem_no_lineal". Fuente. Autor, obtenido con “MegaWizard Plugin-Manager” – Quartus II

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;
-------------------------------------------------------------------------------
ENTITY Mux_Mem_No_Lineal IS
       PORT (       data0x       : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                    data1x       : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                    sel          : IN STD_LOGIC ;
                    result       : OUT STD_LOGIC_VECTOR (8 DOWNTO 0));
END Mux_Mem_No_Lineal;
-------------------------------------------------------------------------------
ARCHITECTURE SYN OF mux_mem_no_lineal IS

--     type  STD_LOGIC_2D  is  array  (NATURAL  RANGE  <>,  NATURAL  RANGE  <>)  of
STD_LOGIC;

SIGNAL sub_wire0    : STD_LOGIC_VECTOR (8 DOWNTO 0);
SIGNAL sub_wire1    : STD_LOGIC ;
SIGNAL sub_wire2    : STD_LOGIC_VECTOR (0 DOWNTO 0);
SIGNAL sub_wire3    : STD_LOGIC_VECTOR (8 DOWNTO 0);
SIGNAL sub_wire4    : STD_LOGIC_2D (1 DOWNTO 0, 8 DOWNTO 0);
SIGNAL sub_wire5    : STD_LOGIC_VECTOR (8 DOWNTO 0);

BEGIN

sub_wire5           <= data0x(8 DOWNTO 0);
result              <= sub_wire0(8 DOWNTO 0);
sub_wire1           <= sel;
sub_wire2(0)        <= sub_wire1;
sub_wire3           <= data1x(8 DOWNTO 0);
sub_wire4(1, 0)     <= sub_wire3(0);
sub_wire4(1, 1)     <= sub_wire3(1);
sub_wire4(1, 2)    <= sub_wire3(2);
sub_wire4(1, 3)    <= sub_wire3(3);
sub_wire4(1, 4)    <= sub_wire3(4);
sub_wire4(1, 5)    <= sub_wire3(5);
sub_wire4(1, 6)    <= sub_wire3(6);
sub_wire4(1, 7)    <= sub_wire3(7);
sub_wire4(1, 8)    <= sub_wire3(8);
sub_wire4(0, 0)    <= sub_wire5(0);
sub_wire4(0, 1)    <= sub_wire5(1);
sub_wire4(0, 2)    <= sub_wire5(2);
sub_wire4(0, 3)    <= sub_wire5(3);
sub_wire4(0, 4)    <= sub_wire5(4);
```

```
sub_wire4(0, 5)    <= sub_wire5(5);
sub_wire4(0, 6)    <= sub_wire5(6);
sub_wire4(0, 7)    <= sub_wire5(7);
sub_wire4(0, 8)    <= sub_wire5(8);

lpm_mux_component : lpm_mux
GENERIC MAP (      lpm_size => 2,
                   lpm_type => "LPM_MUX",
                   lpm_width => 9,
                   lpm_widths => 1)
PORT MAP (   sel => sub_wire2,
                   data => sub_wire4,
                   result => sub_wire0);
--------------------------------------------------------------------------------
END SYN;
```

Anexo 14: Descripción en VHDL del Módulo de memoria del autómata NO Lineal. Fuente. Autor, obtenido con "MegaWizard Plugin-Manager" – Quartus II

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera_mf;
USE altera_mf.all;
--------------------------------------------------------------------------------
-
ENTITY Mem_NO_Lineal IS
      PORT (       address      : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                   clock        : IN STD_LOGIC ;
                   data         : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                   wren         : IN STD_LOGIC ;
                   q            : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END Mem_NO_Lineal;
--------------------------------------------------------------------------------
-
ARCHITECTURE SYN OF mem_no_lineal IS

SIGNAL sub_wire0    : STD_LOGIC_VECTOR (7 DOWNTO 0);
COMPONENT altsyncram
GENERIC (          clock_enable_input_a      : STRING;
                   clock_enable_output_a     : STRING;
                   init_file                 : STRING;
                   intended_device_family    : STRING;
                   lpm_hint                  : STRING;
                   lpm_type                  : STRING;
                   numwords_a                : NATURAL;
                   operation_mode            : STRING;
                   outdata_aclr_a            : STRING;
                   outdata_reg_a             : STRING;
                   power_up_uninitialized    : STRING;
                   ram_block_type            : STRING;
                   widthad_a                 : NATURAL;
                   width_a                   : NATURAL;
                   width_byteena_a           : NATURAL);
PORT (       wren_a       : IN STD_LOGIC ;
             clock0       : IN STD_LOGIC ;
             address_a    : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
             q_a          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
             data_a       : IN STD_LOGIC_VECTOR (7 DOWNTO 0));
END COMPONENT;
```

```
BEGIN
q    <= sub_wire0(7 DOWNTO 0);

altsyncram_component : altsyncram
GENERIC MAP (clock_enable_input_a => "BYPASS",
             clock_enable_output_a => "BYPASS",
             init_file => "Mem_no_lineal.mif",      -- Archivo de inicialización
       de la memoria
                                                    -- dicho de otra foma, regla incial
del
                                                    -- autómata no lineal
             intended_device_family => "Stratix II",
             lpm_hint => "ENABLE_RUNTIME_MOD=NO",
             lpm_type => "altsyncram",
             numwords_a => 512,
             operation_mode => "SINGLE_PORT",
             outdata_aclr_a => "NONE",
             outdata_reg_a => "UNREGISTERED",
             power_up_uninitialized => "FALSE",
             ram_block_type => "M4K",
             widthad_a => 9,
             width_a => 8,
             width_byteena_a => 1)
PORT MAP (          wren_a => wren,
                    clock0 => clock,
                    address_a => address,
                    data_a => data,
                    q_a => sub_wire0);
--------------------------------------------------------------------------------
-
END SYN;
```

Anexo 15: Conexión de los componentes de la Celda correspondiente al diagrama de la . Fuente. Autor

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- Module Name : Celda
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--------------------------------------------------------------------------------
entity Celda is
       Port (       -- Señales Globales
                    clk : in STD_LOGIC;
                    reset : in STD_LOGIC;
                    en : in STD_LOGIC;
                    sel_datos : in STD_LOGIC_VECTOR(2 downto 0);
                    sel_vecinos : in STD_LOGIC_VECTOR(3 downto 0);
                    Data_in : in STD_LOGIC_VECTOR(7 downto 0);
                    --> Acumulador
                    acc_en : in STD_LOGIC;
                    clr_acc : in STD_LOGIC;
                    --> Orientación de la Información
                    clr_info : in STD_LOGIC;
                    orient_info : in STD_LOGIC;
```

```
                    --> Mod Memoria - Filtros
                    dir_memoria : in STD_LOGIC_VECTOR(5 downto 0);
                    wen : in STD_LOGIC;
                    --> Autómata NO Lineal
                    sel_ca_no_lineal : in STD_LOGIC;
                    wen_2 : in STD_LOGIC;
                    address_ca_no_lineal : in STD_LOGIC_VECTOR(8 downto 0);
                    data_no_lineal : in STD_LOGIC_VECTOR(7 downto 0);
                    --> Vecinos
                    arr : in STD_LOGIC_VECTOR(7 downto 0);
                    abj : in STD_LOGIC_VECTOR(7 downto 0);
                    izq : in STD_LOGIC_VECTOR(7 downto 0);
                    der : in STD_LOGIC_VECTOR(7 downto 0);
                    dul : in STD_LOGIC_VECTOR(7 downto 0);
                    dur : in STD_LOGIC_VECTOR(7 downto 0);
                    ddl : in STD_LOGIC_VECTOR(7 downto 0);
                    ddr : in STD_LOGIC_VECTOR(7 downto 0);
                    --> Estado de la celda
                    estado : out STD_LOGIC_VECTOR(7 downto 0));
end Celda;
--------------------------------------------------------------------------------
architecture arq_Celda of Celda is

-- ================================== --
-- == Módulos que componen la celda == --
-- ================================== --
--> Memoria RAM 512bits
component Mod_Memoria is
      Port ( address            : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
                    clock       : IN STD_LOGIC ;
                    data        : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                    wren        : IN STD_LOGIC ;
                    q           : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
end component;
--> Multiplexor selector de vecinos
component Mux_Vecinos is
      Port ( data0x       : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data1x      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data2x      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data3x      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data4x      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data5x      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data6x      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data7x      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data8x      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    sel         : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                    result      : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
end component;
--> Multiplexor para direcciones de memoria del autómata No Lineal
component Mux_Mem_No_Lineal is
      Port ( data0x       : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                    data1x      : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                    sel         : IN STD_LOGIC ;
                    result      : OUT STD_LOGIC_VECTOR (8 DOWNTO 0));
end component;
--> Autómata NO Lineal
component Mem_NO_Lineal is
      Port ( address            : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                    clock       : IN STD_LOGIC ;
                    data        : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
```

```
                    wren          : IN STD_LOGIC ;
                    q             : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
end component;
--> Multiplicador
component Multiplicador is
      Port ( dataa        : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    datab         : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    result        : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
end component;
--> Acumulador
component Acumulador is
      Port ( reset : in STD_LOGIC;
                    ck_en : in STD_LOGIC;
                    x : in STD_LOGIC_VECTOR(15 downto 0);
                    Acc : out STD_LOGIC_VECTOR(15 downto 0));
end component;
--> Meddida de Orientación de la Información
component Med_Orient_Info is
      port ( clk : in STD_LOGIC;
                    reset : in STD_LOGIC;
                    orient_info : STD_LOGIC;
                    Entrada : in STD_LOGIC_VECTOR(15 downto 0);
                    Salida : out STD_LOGIC_VECTOR(15 downto 0));
end component;
--> "Binarización" de la imagen
component Binarizado is
      Port ( dato_a : in STD_LOGIC;
                    dato_b : out STD_LOGIC_VECTOR(7 downto 0));
end component;
--> Multiplexor de entrada al registro de estado
component Mux_Reg_estado is
      Port ( data0x       : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data1x        : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data2x        : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data3x        : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    data4x        : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                    sel                 : IN STD_LOGIC_VECTOR(2 downto 0);
                    result        : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
end component;
--> Registro de Estado
component Reg_Estado is
      Port ( clk : in STD_LOGIC;
                    reset : in STD_LOGIC;
                    en : in STD_LOGIC;
                    D : in STD_LOGIC_VECTOR(7 downto 0);
                    Q : out STD_LOGIC_VECTOR(7 downto 0));
end component;
-- ======================== --
-- == Señales de Conexión == --
-- ======================== --
signal s_clr_acc : STD_LOGIC;
signal s_clr_info : STD_LOGIC;
signal s_factor_1 : STD_LOGIC_VECTOR(7 downto 0);
signal s_factor_2 : STD_LOGIC_VECTOR(7 downto 0);
signal s_prod : STD_LOGIC_VECTOR(15 downto 0);
signal s_acc : STD_LOGIC_VECTOR(15 downto 0);
signal s_orient_info : STD_LOGIC_VECTOR(15 downto 0);
signal s_reg_estado : STD_LOGIC_VECTOR(7 downto 0);
signal s_estado : STD_LOGIC_VECTOR(7 downto 0);
signal s_binarizado : STD_LOGIC_VECTOR(7 downto 0);
```

```
signal s_no_lineal_mux : STD_LOGIC_VECTOR(8 downto 0);
signal s_no_lineal_in : STD_LOGIC_VECTOR(8 downto 0);
signal s_no_lineal_out : STD_LOGIC_VECTOR(7 downto 0);

begin

-- ================================= --
-- == Conexión de los Componentes == --
-- ================================= --
C0: Mux_Vecinos
port map (   data0x => s_estado,
                    data1x => arr,
                    data2x => abj,
                    data3x => izq,
                    data4x => der,
                    data5x => dul,
                    data6x => dur,
                    data7x => ddl,
                    data8x => ddr,
                    sel => sel_vecinos,
                    result => s_factor_1);

C1: Mod_Memoria
port map (   address => dir_memoria,
                    clock => clk,
                    wren => wen,
                    data => Data_in,
                    q => s_factor_2);

s_no_lineal_in      <=     s_estado(0)  &      arr(0) &      abj(0) &
                                izq(0) &      der(0) &      dul(0) &
                                dur(0) &      ddl(0) &      ddr(0);


C2: Mux_Mem_No_Lineal
port map (   data0x => s_no_lineal_in,
                    data1x => address_ca_no_lineal,
                    sel => sel_ca_no_lineal,
                    result => s_no_lineal_mux);

C3: Mem_NO_Lineal
port map (   clock => clk,
                    wren => wen_2,
                    address => s_no_lineal_mux,
                    data => data_no_lineal,
                    q => s_no_lineal_out);

C4: Multiplicador
port map (   dataa => s_factor_1,
                    datab => s_factor_2,
                    result => s_prod);

s_clr_acc <= clr_acc or reset;

C5: Acumulador
port map (   reset => s_clr_acc,
                    ck_en => acc_en,
                    x => s_prod,
                    Acc => s_acc);
```

```
s_clr_info <= reset or clr_info;

C6: Med_Orient_info
port map (   clk => clk,
                    orient_info => orient_info,
                    reset => s_clr_info,
                    Entrada => s_acc,
                    Salida => s_orient_info);

C7: Binarizado
port map (   dato_a => s_orient_info(7),
                    dato_b => s_binarizado);

C8: Mux_Reg_estado
port map (   data0x => s_orient_info(15 downto 8),
                    data1x => s_orient_info(7 downto 0),
                    data2x => s_binarizado,
                    data3x => s_no_lineal_out,
                    data4x => Data_in,
                    sel => sel_datos,
                    result => s_reg_estado);

C9: Reg_Estado
port map (   clk => clk,
                    reset => reset,
                    en => en,
                    D => s_reg_estado,
                    Q => s_estado);
--> Datos de Salida
estado <= s_estado;
-------------------------------------------------------------------------------
end arq_Celda;
```

Anexo 16: Código en VHDL para la implementación del Arreglo 1

```
--======================================== --
-- == Arreglo para la habilitación de escritura inicial en cada celda == --
--======================================== --
--> Crea el arreglo
subtype word_0 is STD_LOGIC_VECTOR(17 downto 0);
type matriz_0 is array(17 downto 0) of word_0;
signal en_s : matriz_0; --> Para la señal 'EN'

--> Reordena el vector 'Dir' en forma de matriz
Dir_0: for i in 0 to 17 generate
begin
       Dir_1: for j in 0 to 17 generate
       begin
              en_s(i)(j)<=Dir((18*i)+j);
       end generate Dir_1;
end generate Dir_0;
```

Anexo 17: Código en VHDL del Arreglo 2

```
-- ========================================== --
-- == Arreglo para la conexión de las celdas == --
-- ========================================== --
subtype word_1 is STD_LOGIC_VECTOR(7 downto 0);
type matriz_1 is array (17 downto 0) of word_1;
type matriz_2 is array(17 downto 0) of matriz_1;
signal x : matriz_2; -- Señal para manipulación del arreglo
-- ============================== --
-- == Crea la matriz de celdas == --
-- ============================== --
Matriz_Celdas: for i in 0 to 17 generate
begin
	--> Fila 0 (superior)
	Fila_0 : if i=0 generate
	begin
		Columnas_Fila_0 : for j in 0 to 17 generate
		begin
			R: Registro_8_Bits --> Condiciones de frontera
		end generate Columnas_Fila_0;
	end generate Fila_0;
	--> Filas centrales
	Filas_centrales: if i>=1 and i<=16 generate
	begin
		Columnas_Filas_centrales: for j in 0 to 17 generate
		begin
			--> Columna 0
			Filas_centrales_Columna_0: if j=0 generate
			begin
				R: Registro_8_Bits --> Condiciones de frontera
			end generate Filas_centrales_Columna_0;
			--> Columnas centrales
			Filas_centrales_Columnas_centrales: if j>=1 and j<=16 generate
			begin
				H: Celda --> Implementa todas las celdas del autómata -
16x16 celdas
			end generate Filas_centrales_Columnas_centrales;
			--> Columna 17
			Fila_centrales_Columna_17: if j=17 generate
			begin
				R: Registro_8_Bits --> Condiciones de frontera
			end generate Fila_centrales_Columna_17;
		end generate Columnas_Filas_centrales;
	end generate Filas_centrales;
	--> Fila 17 (inferior)
	Fila_17: if i=17 generate
	begin
		Columnas_Fila_17: for j in 0 to 17 generate
		begin
			R: Registro_8_Bits --> Condiciones de frontera
			end generate Columnas_Fila_17;
	end generate Fila_17;
end generate Matriz_Celdas;
```

Anexo 18: Código en VHDL del Arreglo 3. Fuente: autor.

```
-- ===================================== --
-- == Arreglos para los datos de salida == --
-- ===================================== --
subtype word_2 is STD_LOGIC_VECTOR(7 downto 0);
type datos_salida is array (255 downto 0) of word_2;
signal s_0 : datos_salida;--> Señal de manipulación del arreglo
--> Desarma la matriz
Sal_0: for i in 0 to 15 generate
begin
	Sal_1: for j in 0 to 15 generate
	begin
		s_0((16*i)+j)<=x(i+1)(j+1);
	end generate Sal_1;
end generate Sal_0;
```

Anexo 18: códigos implementados sobre el procesador NIOS

Manejo_UART.h – Manejo_UART.c

```
/*****************************************
 **** FUNCIONES PARA EL MANEJO DE LA UART ***
 *****************************************/
#ifndef MANEJO_UART_H_
#define MANEJO_UART_H_

#include "system.h"		// Librería de direcciones de los registros del sistema
#include "alt_types.h"	// Define tipos de datos (alt_u8, alt_u32, etc.)
#include "altera_avalon_uart_regs.h"	// Registros de la UART

/* -> Inicializa la UART habilitando interrupciones por recepción de datos*/
void inicializacion_uart(alt_u32 enable_int);
/* -> Lee el registro Rx de la UART */
alt_u8 lectura_Rx();
/* -> Escribe Datos en la UART */
void escritura_uart(alt_u32 dato);

#endif
```

```
/*****************************************
 **** FUNCIONES PARA EL MANEJO DE LA UART ***
 *****************************************/
#include "Manejo_UART.h"  // Librería en donde se declaran las funciones

/* -> Inicializa la UART habilitando interrupciones por recepción de datos*/
void inicializacion_uart(alt_u32 enable_int)
{
    // Habilita interrupciones de Rx en el registro de control
    IOWR_ALTERA_AVALON_UART_CONTROL(UART_BASE,enable_int); }
/* -> Lee el registro Rx de la UART */
alt_u8 lectura_Rx()
{
```

```
    alt_u8 dato_leido;
// Lee y enmáscara el registro de recepción de datos de la UART
    dato_leido=(alt_u8)(0x000000FF & IORD_ALTERA_AVALON_UART_RXDATA(UART_BASE));
    return dato_leido;
}
/* -> Escribe Datos en la UART */
void escritura_uart(alt_u32 dato)
{
    volatile alt_u32 escr_ready=0;
    while(escr_ready==0)
    {
        // Lee el bit TRDY para determinar la siguiente escritura
        escr_ready=(alt_u32)(0x0040 & IORD_ALTERA_AVALON_UART_STATUS(UART_BASE));
        escr_ready=escr_ready>>6;             // Desplza 6 bits para asuntos de
formato
    }
    IOWR_ALTERA_AVALON_UART_TXDATA(UART_BASE,(0x000000FF & dato)); }
```

Automata.h – Automata.c

```
/*************************************************************
 *** FUNCIONES PARA COMUNICACIÓN CON EL AUTÓMATA CELULAR ***
 *************************************************************/
#ifndef AUTOMATA_H_
#define AUTOMATA_H_

#include <unistd.h>
#include "system.h"
#include "alt_types.h"
#include "altera_avalon_pio_regs.h"

/* -> Lee datos del autómata  */
alt_u32 lectura_de_ca(alt_u32 direccion);

#endif
```

```
/*************************************************************
 *** FUNCIONES PARA COMUNICACIÓN CON EL AUTÓMATA CELULAR ***
 *************************************************************/
#include "Automata.h"

/* -> Lee datos del autómata  */
alt_u32 lectura_de_ca(alt_u32 direccion)
{
       // Lee el puerto de entrada 'estado_automata', enmáscara los datos
       // para evitar la lectura de datos incorrectos y retorna el dato leído
    IOWR_ALTERA_AVALON_PIO_DATA(ADDR_BASE,direccion);
    return (0x000000FF & IORD_ALTERA_AVALON_PIO_DATA(ESTADO_AUTOMATA_BASE));
}
```

Adicional.h – Adicional.c

```
/* ************************************************************
   **** PATRÓN DE LEDS PARA COMPROBAR ESCRITURA EN MEMORIA ****
   ************************************************************ */
#ifndef ADICIONAL_H_
#define ADICIONAL_H_

#include <unistd.h>
#include "system.h"
#include "alt_types.h"
#include "altera_avalon_pio_regs.h"

// --> Condiciones Iniciales
void condiciones_iniciales(void *pdatos);
// --> Carga de Coeficientes de los filtros
void carga_configuracion(void* pdatos);
// Convolución de los datos
void convolucion(void* pdatos, volatile alt_u32* pfilt);

#endif
```

```
/* ************************************************************
   **** PATRÓN DE LEDS PARA COMPROBAR ESCRITURA EN MEMORIA ****
   ************************************************************ */
#include "Adicional.h"
#include "Manejo_UART.h"

// --> Carga de Coeficientes de los filtros
void carga_configuracion(void* pdatos)
{
// Carga los coeficientes de los filtros //

    // Cambia el formato de la variable
    volatile alt_u32 *p_datos_2 = (volatile alt_u32*)(pdatos);
alt_u32 i=0;
    alt_32 dato=0;  // Contador de datos a escribir

// Habilitación de escritura en memorias de las celdas
IOWR_ALTERA_AVALON_PIO_DATA(WEN_BASE,0x00000001);
    for(i=0;i<36;i++)
{
        dato = (alt_u32)*(p_datos_2+i);

       // Dirección de escritura
        IOWR_ALTERA_AVALON_PIO_DATA(DIR_MEMORIA_BASE,i);

  // Dato a escribir en memmorias
        IOWR_ALTERA_AVALON_PIO_DATA(DATA_IN_BASE,dato);
    }
// Reestablece el bit 'wen' (Se escribe con flanco de bajada)
    IOWR_ALTERA_AVALON_PIO_DATA(WEN_BASE,0x00000000);
}
// --> Carga las condiciones inicales
void condiciones_iniciales(void *pdatos)
```

```
{
    // Escribe los datos iniciales en cada celda del autómata (el valor del píxel
en cada
    // celda) y las condiciones de frontera
    volatile alt_32 *p_datos_1 = (volatile alt_u32*)(pdatos);

// Cambia el formato del apuntador de la variable 'pdatos'
    alt_u32 i=0;
    alt_u32 dato=0;
    IOWR_ALTERA_AVALON_PIO_DATA(SEL_DATOS_BASE,0x00000004);
    for(i=0;i<=323;i++)
    {
        dato = (alt_u32)*(p_datos_1+i+36);

       // Escribe datos en las celdas del autómata
         IOWR_ALTERA_AVALON_PIO_DATA(DATA_IN_BASE,dato);
       // Determina en cual celda se escribe y actualiza el estado de la misma
       IOWR_ALTERA_AVALON_PIO_DATA(EN_IN_BASE,i+0x00000001);
    }
    IOWR_ALTERA_AVALON_PIO_DATA(SEL_DATOS_BASE,0x00000000);

// Reestablece el bus de habilitación de escritura en las celdas
    IOWR_ALTERA_AVALON_PIO_DATA(EN_IN_BASE,0x00000000);

    // Borra el bus de datos
    IOWR_ALTERA_AVALON_PIO_DATA(DATA_IN_BASE,0x00000000);
}
// --> Realiza la convolución 2D
void convolucion(void* pdatos, volatile alt_u32* pfilt)
{
    volatile alt_u32 i=0;
    for(i=0;i<=8;i++)
{
        volatile alt_u32 k = (alt_u32)*pfilt;
        IOWR_ALTERA_AVALON_PIO_DATA(CK_EN_BASE,0x0000000);
        IOWR_ALTERA_AVALON_PIO_DATA(DIR_MEMORIA_BASE,i+k);
        IOWR_ALTERA_AVALON_PIO_DATA(SEL_VECINOS_BASE,i);
        IOWR_ALTERA_AVALON_PIO_DATA(CK_EN_BASE,0x0000001);
    }
IOWR_ALTERA_AVALON_PIO_DATA(CK_EN_BASE,0x0000000); }
```

Control.c (unidad de control)

```
/*********************************
 ***   Module Name : Control.c  ***
 *********************************/
#include <unistd.h>
#include <sys/alt_irq.h>
#include "system.h"
#include "alt_types.h"
#include "altera_avalon_uart_regs.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_timer_regs.h"
#include "Manejo_UART.h"
#include "Automata.h"
```

```
#include "Adicional.h"

// ************************* //
// *** Variables Globales *** //
// ************************* //
volatile alt_u8 flag=0x00;      // Variable que determina la función a realizar
volatile alt_u8 *pflag=&flag;   // Apuntador a la varable flag

volatile alt_u32 filt=0;        // Variable que determina que filtro usar
volatile alt_u32 *pfilt=&filt;  // Apuntador a la variable filt

volatile alt_u32 ind=0;         // Contador de datos que llegan desde la UART
volatile alt_u32 datos[360];    // "Buffer" en donde se almacenan los datos de la
UART
void *pdatos=(void*)&datos[0];  // Apuntador de 'datos'
// ************************************************************** //
// *** Rutina de Interrupción - ISR - para lectura de Comandos **** //
// ************************************************************** //
// --> Descripción:
//     Captura los datos provenientes desde la UART
//     Llena el buffer con dichos datos y va contando la cantidad de los mismos
static void lectura_comandos_isr(void *context, alt_u32 id)
{
    // Da formato a la variable 'bandera' que determina la operacion a realizar
    volatile alt_u32 *p_isr = (volatile alt_u32*)(context);     volatile alt_u32
temp;
// Variable temporal para almacenamiento de datos
    temp=(alt_u32)lectura_Rx();  // Lee los datos de la UART
    *(p_isr+ind)=temp;              // Escribe en 'buffer' el dato leído
    ind++;                          // Incrementa el contador de datos recibidos
    if(ind==360)
    {
        ind=0;                    // Reestablece el contador de datos
      *pflag = 0x01;               // Habilita la primera función a realizar
    }
}
// ************************************* //
// *** Funciones Usadas en el Programa *** //
// ************************************* //
// --> Inicializacion del sistema:
void inicializacion(void)
{
// Inicializa el módulo UART

// Configura la UART para generar interrupciones por recepción de datos
    inicializacion_uart(0x00000080);
    // Borra puertos
    IOWR_ALTERA_AVALON_PIO_DATA(ORIENT_INFO_BASE,0x00000000);
    IOWR_ALTERA_AVALON_PIO_DATA(CLR_INFO_BASE,0x00000001);
IOWR_ALTERA_AVALON_PIO_DATA(CLR_ACC_BASE,0x00000001);
IOWR_ALTERA_AVALON_PIO_DATA(CK_EN_BASE,0x00000000);
IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_GLOBAL_BASE,0x00000000);
IOWR_ALTERA_AVALON_PIO_DATA(SEL_DATOS_BASE,0x00000000);
IOWR_ALTERA_AVALON_PIO_DATA(WEN_BASE,0x00000000);
IOWR_ALTERA_AVALON_PIO_DATA(ADDR_BASE,0x00000000);
IOWR_ALTERA_AVALON_PIO_DATA(DATA_IN_BASE,0x00000000);
IOWR_ALTERA_AVALON_PIO_DATA(DIR_MEMORIA_BASE,0x00000000);
```

```
IOWR_ALTERA_AVALON_PIO_DATA(EN_IN_BASE,0x00000000);
IOWR_ALTERA_AVALON_PIO_DATA(SIETE_SEG_BASE,0x0000FFFF);
IOWR_ALTERA_AVALON_PIO_DATA(DATA_NO_LINEAL_BASE,0x00000000);
    IOWR_ALTERA_AVALON_PIO_DATA(WEN_2_BASE,0x00000000);
    IOWR_ALTERA_AVALON_PIO_DATA(ADDRESS_CA_NO_LINEAL_BASE,0x00000000);
    IOWR_ALTERA_AVALON_PIO_DATA(SEL_CA_NO_LINEAL_BASE,0x00000000);
}
// --> Función Principal
void funcion_principal(volatile alt_u8 *pflag, void *pdatos)
{
// ========================================== //
    // == INICIALIZACIÓN Y CONFIGURACIÓN INICIAL == //
    // ========================================== //
    if(*pflag==0x01)
    {
// --> Carga Configuración y Datos
        carga_configuracion(pdatos);
        condiciones_iniciales(pdatos);
        *pflag^=0x03;
}
    else
    {
        if(*pflag==0x02)
        {
// --> Filtros
            IOWR_ALTERA_AVALON_PIO_DATA(CLR_INFO_BASE,0x00000000);
            volatile alt_u32 i = 0;
            for(i=0;i<4;i++)
{
                IOWR_ALTERA_AVALON_PIO_DATA(CLR_ACC_BASE,0x00000000);
                *pfilt = (alt_u32)(9*i);
                convolucion(pdatos,pfilt);
                IOWR_ALTERA_AVALON_PIO_DATA(ORIENT_INFO_BASE,0x00000001);
                IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_GLOBAL_BASE,0x00000001);
                IOWR_ALTERA_AVALON_PIO_DATA(ORIENT_INFO_BASE,0x00000000);
                IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_GLOBAL_BASE,0x00000000);
                IOWR_ALTERA_AVALON_PIO_DATA(CLR_ACC_BASE,0x00000001);
}
            *pflag ^= 0x06;
        }
        else
        {
            if(*pflag==0x04)
{
                volatile alt_u32 dir=0;
                volatile alt_u32 i=256;
                while(i!=0)
                {
// --> Lee los resultados filtros MOI (Parte Alta)
                    // Lee los datos del autómata y los escribe en la UART
                    escritura_uart(lectura_de_ca(dir));
                    dir++;
i--;
                }
                *pflag^=0x0C;
            }
            else
```

```
            {
if(*pflag==0x08)
                {
                    IOWR_ALTERA_AVALON_PIO_DATA(SEL_DATOS_BASE,0x00000001);
                    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_GLOBAL_BASE,0x00000001);
                    IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_GLOBAL_BASE,0x00000000);
                    IOWR_ALTERA_AVALON_PIO_DATA(SEL_DATOS_BASE,0x00000000);
                    volatile alt_u32 dir=0;
                    volatile alt_u32 i=256;
                    while(i!=0)
                    {
                        // --> Lee los resultados filtros MOI (Parte Baja)
                        // --> Lee los resultados de la imagen binarizada
                        escritura_uart(lectura_de_ca(dir));
                    // Lee los datos del autómata y los escribe en la UART
                        dir++;
                        i--;
                    }
             *pflag^=0x18;
                }
                else
                {
                    if(*pflag==0x10)
{
                        IOWR_ALTERA_AVALON_PIO_DATA(SEL_DATOS_BASE,0x00000002);

IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_GLOBAL_BASE,0x00000001);

IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_GLOBAL_BASE,0x00000000);
                        IOWR_ALTERA_AVALON_PIO_DATA(SEL_DATOS_BASE,0x00000000);
                        volatile alt_u32 dir=0;
                        volatile alt_u32 i=256;
                        while(i!=0)
                        {
// --> Lee los resultados de la imagen binarizada
                            escritura_uart(lectura_de_ca(dir));
                           // Lee los datos del autómata y los escribe en la UART
                            dir++;
                            i--;
                        }
                    *pflag^=0x30;
                    }
                    else
                    {
                        if(*pflag==0x20)
{
IOWR_ALTERA_AVALON_PIO_DATA(SEL_DATOS_BASE,0x00000003);

IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_GLOBAL_BASE,0x00000001);

IOWR_ALTERA_AVALON_PIO_DATA(ENABLE_GLOBAL_BASE,0x00000000);
                        IOWR_ALTERA_AVALON_PIO_DATA(SEL_DATOS_BASE,0x00000000);
                        volatile alt_u32 dir=0;
                        volatile alt_u32 i=256;
                        while(i!=0)
                        {
```

```
// --> Lee los resultados de la imagen luego del CA NO Lineal
             // Lee los datos del autómata y los escribe en la UART
                           escritura_uart(lectura_de_ca(dir));
                           dir++;
                           i--;
                        }
                        IOWR_ALTERA_AVALON_PIO_DATA(CLR_ACC_BASE,0x00000001);
                        IOWR_ALTERA_AVALON_PIO_DATA(CLR_INFO_BASE,0x00000001);
                        *pflag^=0x20;
                        }
                    }
                }
            }
        }
}
////////////////////////////////////////////////////////////////////////////
/////////////// ********************* //
// *** Función main() *** //
// ********************* //
int main()
{
    alt_u32 a=0x00000055;
// Inicialización del sistema
    inicializacion();
// Registra la rutina de interrupción para la lectura de datos de la UART en el
Nios
alt_irq_register(   UART_IRQ,
pdatos,
                        lectura_comandos_isr);
// Bucle principal del programa //
    *pflag=0x00;
    *pfilt=0;
    while(1)
    {
        IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE,a);// Patrón de LEDS
        usleep(50000);
        a^=0x000000FF;
        funcion_principal(pflag,pdatos);
    }
    return 0;
}
```

Anexo 19: códigos en Matlab para la interface con el autómata implementado sobre la FPGA

Configuración serial de la Interfaz serial

```
function s=interfaz()
% ================================== %
% ==  Interfaz Comunicación Serie == %
```

```
% ================================= %
% -> Crea el puerto serial
s=serial('COM4');
% -> Configura propiedades de la comunicación
set(s,'baudrate',115200)   % 38400 Bauds
set(s,'databits',8)        % 8 Bits de datos
set(s,'parity','none')     % Sin bit de paridad
set(s,'stopbits',1)        % 1 Bit de stop
set(s,'FlowControl','none')      % Desactiva el control de flujo / Hadshaking
set(s,'Terminator','LF')   % Caracter fin de paquete de transmisión
% -> Propiedades asociadas con la escritura de datos
set(s,'OutputBufferSize',350)    % Crea un buffer de N Bytes de memoria para
                                    % escritura
% -> Propiedades asociadas con la lectura de datos
set(s,'InputBufferSize',262144) % Crea un buffer de N Bytes de memoria para
lectura
end
```

Transmisión de los mascaras de filtrado y de los segmentos de imagen de 16x16

```
function filtrado(img,s)
% Filtrado(Img,Filt)
% Escribe el segmento de imagen junto con el filtro a usar

% Crea un timer para espera de datos y evitar pérdidas durante la
% comunicación
t=timer;
set(t,'StartDelay',0.7,'Period',0.7,'TimerFcn',@(x,y)disp('.'));
% Filtros MOI (Medida de la Orientación de la Información)
filt1=int8(1.*[0 1 -1 0 0 1 1 -1 -1]);
filt2=int8(1.*[0 1 -1 1 -1 1 0 0 -1]);
filt3=int8(1.*[0 0 0 1 -1 1 -1 1 -1]);
filt4=int8(1.*[0 -1 1 1 -1 0 -1 1 0]);
% Arma un vector con los datos a escribir
x=reshape(img,1,324);
% Transmisión de coeficientes del Filtro No.1
fwrite(s,filt1,'int8');
% Transmisión de coeficientes del Filtro No.2
fwrite(s,filt2,'int8');
% Transmisión de coeficientes del Filtro No.3
fwrite(s,filt3,'int8');
% Transmisión de coeficientes del Filtro No.4
fwrite(s,filt4,'int8');
% Transmision de Datos
fwrite(s,x,'int8');
% Incia un timer para evitar la pérdida de datos
start(t);
wait(t);
stop(t);
end
```

Comunicación entre el PC y la FPGA

```
% ==================================== %
% == Comunicación Serial con la FPGA = %
% ==================================== %
% --> Lee la imagen
img=imread('imagen.jpg');
% --> Cambia el formato de la misma a 'int8' (complemento a dos)
img=double(img);
img=img-128;
img=int8(img);
% --> Arma los bloques de 16x16 píxeles a partir de la imagen leída
I=cell(8,8);
Img=cell(8,8);
for i=1:8
for j=1:8
        I{i,j}=img(16*i-15:16*i,16*j-15:16*j);
end
end
% --> Prepara la imagen y las condiciones de frontera
% Determina las condiciones de frontera de cada bloque dependiendo de su
% posición respecto a la imagen completa
for i=1:8
for j=1:8
        x=zeros(18,18);
        x=int8(x);
        x(2:17,2:17)=I{i,j};
if((i-1)>0)
            arr=I{i-1,j};
else
            arr=zeros(16,16);
end
if((i+1)<9)
            abj=I{i+1,j};
else
            abj=zeros(16,16);
end
if((j-1)>0)
            izq=I{i,j-1};
else
            izq=zeros(16,16);
end
if((j+1)<9)
            der=I{i,j+1};
else
            der=zeros(16,16);
end
if((i-1)>0 && (j-1)>0)
            dul=I{i-1,j-1};
else
            dul=zeros(16,16);
end
if((i-1)>0 && (j+1)<9)
            dur=I{i-1,j+1};
else
            dur=zeros(16,16);
```

```
end
if((i+1)<9 && (j-1)>0)
            ddl=I{i+1,j-1};
else
            ddl=zeros(16,16);
end
if((i+1)<9 && (j+1)<9)
            ddr=I{i+1,j+1};
else
            ddr=zeros(16,16);
end
        x(1,2:17)=arr(16,:);
x(18,2:17)=abj(1,:);
        x(2:17,1)=izq(:,16);
        x(2:17,18)=der(:,1);
x(1,1)=dul(16,16);
        x(1,18)=dur(16,1);
        x(18,1)=ddl(1,16);
        x(18,18)=ddr(1,1);
        Img{i,j}=x;
end
end
% --> Crea el puerto de comunicaciones y transmite los datos
s=interfaz();       % Crea la interfaz de puerto serial
fopen(s)            % Abre el puerto para la comunicación
for i=1:8
for j=1:8
        filtrado(Img{i,j},s);
end
end
% Lectura de Datos
B1=s.BytesAvailable;
z=zeros(1,B1);
z=fread(s,B1,'uint8');
% Borra el puerto
fclose(s)
delete(s)
clear s
% Arma las matrices de datos leídos
for i=1:64
    k=4*i-3;
    m=4*i-2;
    n=4*i-1;
    l=4*i;
    a=z(256*k-255:256*k);
    b=z(256*m-255:256*m);
    c=z(256*n-255:256*n);
    d=z(256*l-255:256*l);
    a=reshape(a,16,16);
    b=reshape(b,16,16);
    c=reshape(c,16,16);
    d=reshape(d,16,16);
    A{i}=a;
    B{i}=b;
    C{i}=c;
    D{i}=d;
end
```

```
A=reshape(A,8,8);
A=A';
B=reshape(B,8,8);
B=B';
C=reshape(C,8,8);
C=C';
D=reshape(D,8,8);
D=D';
for i=1:8
for j=1:8
        Ma(16*i-15:16*i,16*j-15:16*j)=A{i,j}; % Parte alta procesado MOI
        Mb(16*i-15:16*i,16*j-15:16*j)=B{i,j}; % Parte baja procesado MOI
        Bi(16*i-15:16*i,16*j-15:16*j)=C{i,j}; % Imagen Binaria
Rf(16*i-15:16*i,16*j-15:16*j)=D{i,j}; % Resultado autómata no Lineal
end
end
```