Capítulo 4

Arquitectura del micro-SIA

En el capítulo 3 se demostró que el micro-SIA puede resolver problemas de optimización numérica global. Aunque al reducir el tamaño de la población resulta obvio que también se reduce el uso de la memoria de datos, resulta interesante analizar cómo se comporta el micro-SIA en una plataforma hardware. En primera instancia, el manejo de pocos individuos parece facilitar la implementación del algoritmo de manera intrínseca en un dispositivo electrónico digital. En esta tesis se dirigieron los esfuerzos hacia dos tipos de dispositivos: un microcontrolador comercial de 8 bits y un dispositivo de lógica reconfigurable (FPGA). El microcontrolador tiene recursos limitados; por otra parte, el FPGA tiene una arquitectura más robusta que la del microcontrolador.

4.1 Micro-SIA en hardware

Para estas realizaciones en hardware se propuso resolver el problema de maximización de unos (*maxone problem*) [34], que consiste en encontrar el número máximo de bits puestos a uno (nivel lógico alto) en una cadena finita. Este problema es clásico en el área del reconocimiento de patrones. Matemáticamente el problema se describe como encontrar un vector $\vec{x} = \{x_1, x_2, \ldots, x_n\}$ en donde $x \in \{0,1\}$ que maximice la ecuación

$$F(\vec{x}) = \sum_{i=1}^{N} x_i \qquad (4.1)$$

En esta ecuación, *N* representa el número de bits de la cadena, por lo que el máximo se obtendrá cuando los *N* bits de la cadena sean unos. Este problema no es sencillo de resolver en hardware, debido a que un número mayor que otro, no necesariamente representa un mejor resultado, por ejemplo, el dato binario 1000 es peor solución que el dato 0011 debido a que su cadena de bits tiene menos unos. A medida que aumenta el número de bits de la cadena, la complejidad de la búsqueda también aumenta tal y como se comenta en [34].

La adecuación general del micro-SIA para las realizaciones en hardware funciona de la siguiente forma:

1. *Generar aleatoriamente 5 anticuerpos de N bits*. En la generación inicial estos anticuerpos se copian directamente a la población inicial para comenzar la convergencia nominal que está controlada por un número de generaciones igual a 5. Obsérvese que a diferencia del micro-SIA utilizado para optimización numérica, ahora se utilizan 5 generaciones en vez de 10, para alcanzar la convergencia nominal. La representación de los individuos ahora es binaria y no real, como sucedió en el capítulo 3 de este trabajo.

2. *Utilizar una selección basada en ranking*. Tal y como se hizo con el micro-SIA para optimización numérica. El anticuerpo de mayor afinidad será el mejor individuo.

3. *Realizar la clonación de todos los anticuerpos*. Para tal propósito se utiliza la ecuación 3.1 del capítulo 3 de esta misma tesis. De igual manera a la versión de optimización numérica, considerando una población de 5 anticuerpos se obtendrá una población de 15 clones.

4. *Realizar la maduración de los clones a través de un proceso de mutación considerando que se manejan cadenas binarias*. La probabilidad de mutación se fijará al principio de la convergencia nominal para cada grupo de clones obtenidos del mismo anticuerpo. Esta probabilidad se determina de manera proporcional a la afinidad del anticuerpo que originó los clones y disminuirá en cada generación, así el grupo de clones del mejor anticuerpo, nombrado *BestAb* en la versión en software, mutará menos que los demás grupos de clones que se generaron de los restantes anticuerpos.

El único clon que se obtuvo del peor anticuerpo tendrá la mayor posibilidad de mutar. Para tal propósito se aplican las mismas ecuaciones 3.2 y 3.3 del capítulo 3. Con respecto a la variación que presenta cada uno de los clones al efectuar la mutación, se utilizó un operador sencillo que invierte un bit en determinada posición de la cadena. Para determinar qué bit debe mutarse se aplica una probabilidad de 0.5 considerando que un bit de una cadena que representa un individuo que es una buena solución tendrá menos posibilidades de cambiar que uno que pertenece a una mala solución.

5. *Efectuar nuevamente una selección basada en ranking*. Esta ocasión, se ordenarán los 15 clones con respecto a su afinidad. Los dos mejores clones se seleccionarán (elitismo) y la nueva población se completará con otros 3 clones seleccionados aleatoriamente de la población de clones maduros. Los restantes clones se eliminarán, proveyendo una auto-regulación dentro de la convergencia nominal.

6. *Aplicar elitismo*. Si se alcanzan las 10 generaciones de la convergencia nominal, se mantendrán los dos mejores clones y se generarán otros tres anticuerpos de manera aleatoria para completar la nueva población y comienza nuevamente la convergencia nominal hasta que el ciclo externo del algoritmo cumpla con la condición de parada.

4.2 Implementación en el microcontrolador

Se decidió dirigir la implementación del diseño hacia un dispositivo comercial, en este caso un microcontrolador de 8 bits de bajo coste. Se utilizó un *PIC16F628A* CUYA arquitectura obliga a que se realice un procesamiento procedural de las instrucciones, además se tiene un número de pines de E/S restringido a dos puertos de 8 bits, así como una memoria de datos que es de sólo 224 bytes. Para este caso, las etapas del diseño incluyen:

1. *Generador de números aleatorios*. Se utiliza el oscilador interno del microcontrolador (4MHz) y un *push button* para accionar el módulo generador que como elemento base tiene un contador binario. Este módulo genera 5 cadenas binarias (anticuerpos) para copiarlos a la población inicial.

Existen diversos métodos para generar números aleatorios. Un generador de verdaderos números aleatorios (*RRNG*) en hardware utiliza características no predecibles de elementos físicos (radiación de un núcleo atómico, inestabilidad de un oscilador libre, ruido de una resistencia eléctrica, resonancia de las partículas de un rayo láser, entre otros). Los generadores de números pseudoaleatorios (*PRNG*) son más comunes en hardware debido principalmente, en la mayoría de los casos, a su facilidad de implementación. Los números pseudoaleatorios se generan a partir de una función determinista pero aparentan ser aleatorios. Estos generadores tienen el inconveniente de trabajar con sucesiones periódicas y que a partir de un mismo valor inicial (semilla) se genera siempre la misma sucesión.

En esta tesis se utilizó un *registro de desplazamiento con realimentación lineal* (*LSFR*) como generador de números pseudoaleatorios. Este circuito sirvió tanto para el microcontrolador como para el FPGA, variando la longitud del registro y la función de retroalimentación. Un circuito LFSR genera secuencias pseudoaleatorias con un período garantizado arbitrariamente largo, de forma simple y eficiente, siempre y cuando se inicialicen con valores diferentes de cero. El registro de desplazamiento se implementa con elementos de memoria que comúnmente son *flip-flops tipo D*. La señal común que produce la transferencia de datos entre los flip-flops, está sincronizada con la señal de reloj (CLK) del circuito. Al realizar una combinación de la salida de dos registros y asignándola a la entrada de otro registro, se obtiene un circuito LFSR. Esta realimentación se realiza utilizando compuertas XOR.

Dado que el estado interno de un registro de desplazamiento tiene L bits, un LFSR puede generar secuencias pseudo-aleatorias de tamaño máximo $2^L - 1$ (no 2^L, porque un registro de desplazamiento que se llena con ceros, provoca que la secuencia de salida del LFSR sea una secuencia infinita de ceros). Un LFSR tendrá longitud máxima si su polinomio característico (polinomio de realimentación) es primitivo en $GF(2^n)$, es decir, si se encuentra este polinomio (pudieran existir varios) es posible

determinar cuántas operaciones XOR se requieren para la retroalimentación y en qué etapas se deben conectar para cumplir con todas las combinaciones binarias de 2^n bits sin repetirlas. Existen tablas que indican los polinomios característicos para realizar la conexión de las compuertas XOR, para diferentes tamaños de registros.

El esquema más habitual de un circuito LFSR de 8 bits, se muestra en la figura 4.1. Es el mismo esquema que se empleó en esta tesis para generar los números pseudoaleatorios en el microcontrolador.

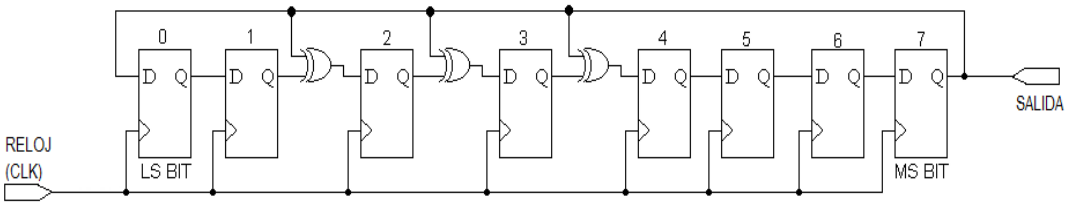


Figura 4.1. LFSR de 8 bits, utilizado para la generación de números pseudoaleatorios.

2. *Módulo de ordenamiento (ranking)*. Este módulo cuenta los unos de cada cadena y asigna una calificación a cada individuo. Posteriormente los ordena de mejor a peor considerando que los mejores individuos son los que tienen el mayor número de unos. Para realizar el ordenamiento se utilizó el conocido método de la burbuja, cuyo pseudocódigo se muestra en la figura 4.2. En éste, *LIMITE* es el número de datos a ordenar.

```
for (i=1; i<LIMITE; i++)
    for j=0 ; j<LIMITE - 1; j++)
          if (vector[j] > vector[j+1])
               temp = vector[j];
               vector[j] = vector[j+1];
               vector[j+1] = temp;
```

Figura 4.2. Pseudocódigo del ordenamiento burbuja.

Para contar los unos de una cadena se utiliza un procedimiento iterativo en donde se recorre un registro de datos (palabra) de un extremo hasta el otro y se incrementa un contador cada vez que se encuentra un *"1"*. El esquema simple utilizado en esta tesis consta de un registro de desplazamiento, que con ayuda de una compuerta XOR y un contador, permite detectar la presencia de un uno y así incrementar el conteo. En el micro-SIA adecuado para hardware, esta tarea es el equivalente a la evaluación de la función objetivo. El mismo número de unos contado es el valor en calificación que se le asigna a cada individuo, equivalente a la aptitud o *afinidad* del SIA. Para ordenarlos de acuerdo a su afinidad, se utiliza un comparador que detecta quién o quiénes tienen el mayor número de unos y los clasifica como los mejores individuos.

3. *Módulo de clonación*. Este módulo se encarga de generar 5 clones del mejor individuo, 4 del segundo mejor y así sucesivamente hasta obtener los 15 clones que caracterizan a nuestro algoritmo.

4. *Módulo de mutación*. Cada uno de los 15 clones tiene probabilidad de mutar, sin embargo, los 5 clones obtenidos del mejor individuo tienen menos probabilidad de mutar que el único clon obtenido del peor individuo, quien tiene la más alta probabilidad de mutar. Toda la cadena de bits de cada clon se recorre completamente para aplicar el operador de mutación en donde corresponda.

Para los 5 clones del mejor anticuerpo se utiliza mutación uniforme, es decir, se invertirá un solo bit de la cadena. La posición de este bit se elige aleatoriamente. Para los 4 clones del segundo mejor anticuerpo se aplica mutación en dos puntos, es decir, se invertirán dos bits en dos posiciones elegidas aleatoriamente. Sucesivamente se utiliza este criterio hasta llegar al peor anticuerpo que sólo obtuvo un clon, a éste se le aplicará mutación en 5 puntos, lo que indica que se invertirán 5 bits de su cadena cuyas posiciones en ésta también se eligen aleatoriamente. En la figura 4.3 se muestra cómo funciona la mutación para el mejor y el peor clon generado.

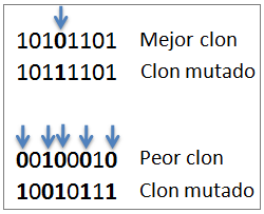


Figura 4.3. Mutación aplicada a los clones.

5. *Módulo de ordenamiento de clones*. Este módulo funciona de manera similar al primer módulo de ordenamiento para 5 anticuerpos, con la observación de que ahora debe clasificar 15 clones. Tras haber mutado, los individuos en el *ranking* han cambiado su calidad.

6. *Módulo de remplazo*. Este módulo se encarga de mantener a los dos mejores clones de los 15 generados y completa la población de trabajo con otros tres que se generan de manera aleatoria. Este módulo hace uso del módulo generador de números aleatorios para cumplir con su encomienda. Este mismo módulo lleva el control de la convergencia nominal por lo que al alcanzar las 5 generaciones del ciclo interno, mantendrá a los dos mejores individuos, que a su vez son las dos mejores soluciones, y los copia a la población inicial para comenzar nuevamente un ciclo externo del micro-SIA.

En la figura 4.4 se pueden observar los módulos concebidos para la implementación embebida, que complementan la explicación anterior. Las etapas del micro-SIA implementado en hardware son las mismas que se presentaron en la figura 3.2 del capítulo 3 de esta tesis.

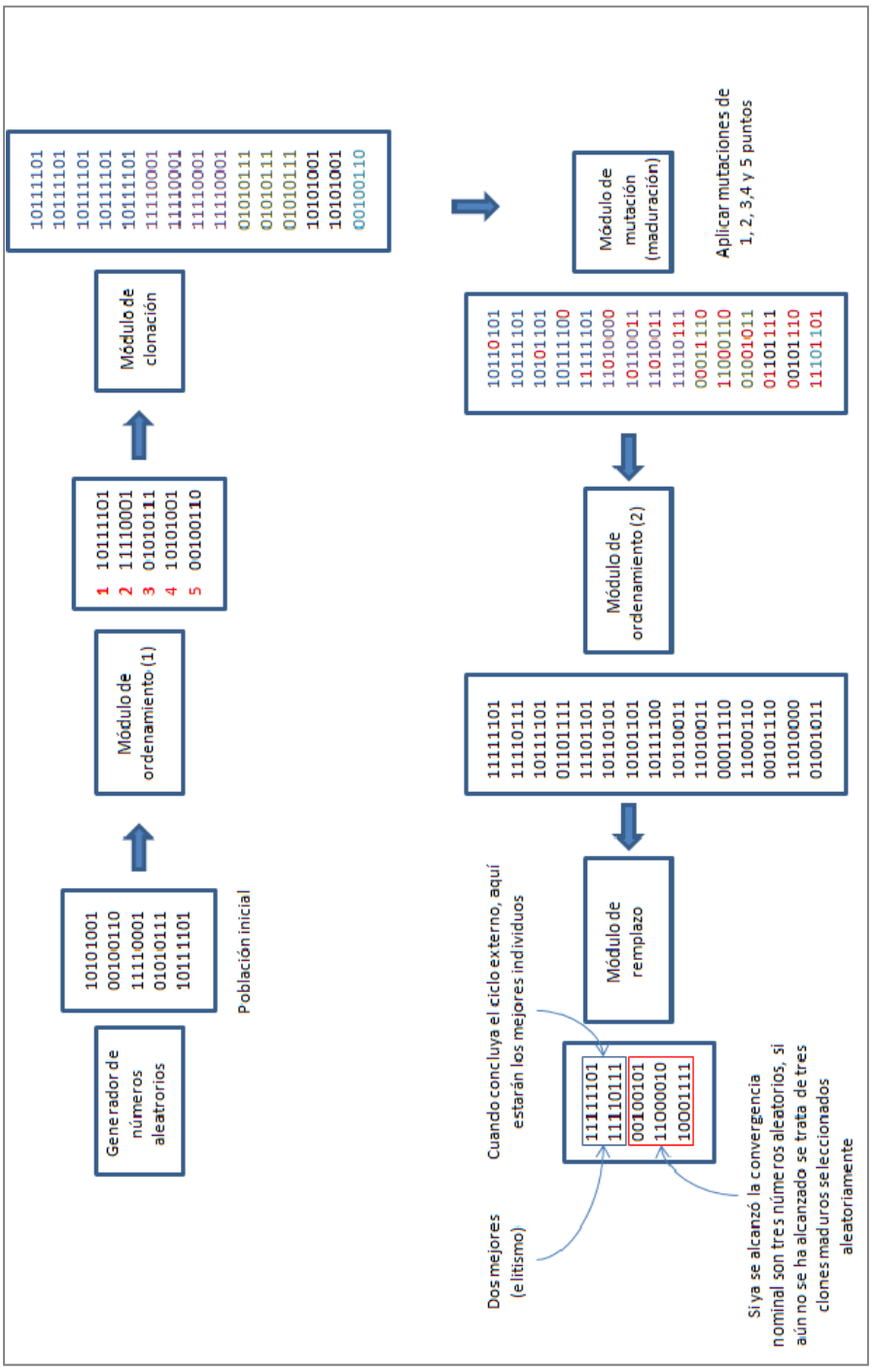


Figura 4.4. Diagrama a bloques del micro-SIA implementado en hardware, para solucionar el problema de la maximización de unos en una cadena de 8 bits.

4.2.1 Experimentación con el microcontrolador

Se realizó una interfaz que permitió comunicar el microcontrolador con una PC, vía el puerto serial (RS-232). Esta interfaz sólo se utilizó para monitorear (leer) los datos entregados por el microcontrolador, dado que el micro-SIA se ejecuta intrínsecamente. El circuito de prueba que se construyó se muestra en el diagrama de la figura 4.5. El *push-button* permite iniciar la ejecución del algoritmo. La terminal en la PC recibe los datos iniciales y finales del algoritmo; estos datos son: los 5 primeros individuos generados aleatoriamente, el número de generaciones (ciclo externo del micro-SIA) de la convergencia y la solución obtenida.

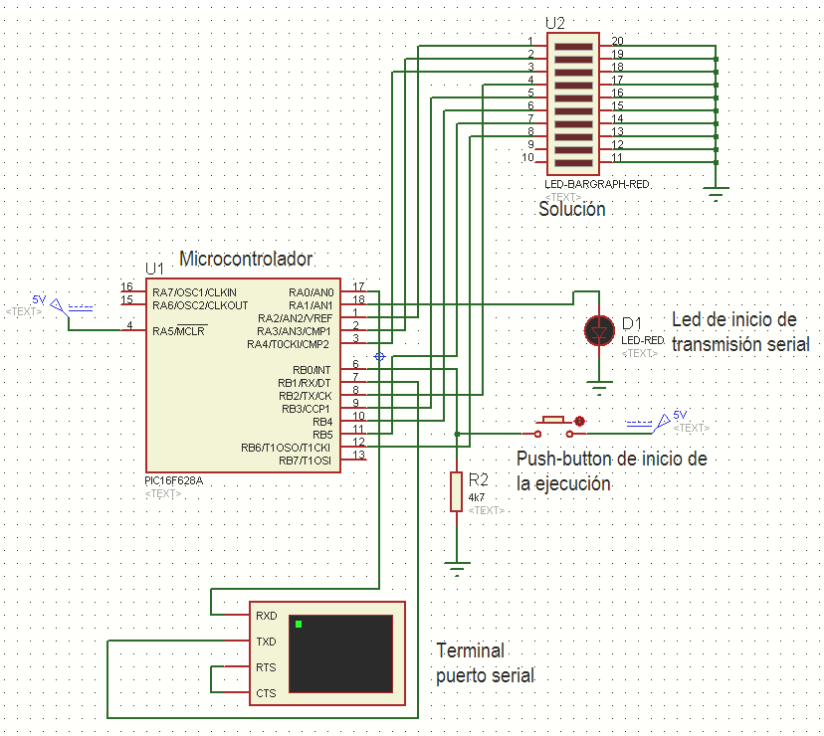


Figura 4.5. Implementación del micro-SIA en el microcontrolador PIC16F628, para resolver el problema de maximización de unos.

En la tabla 4.1 se reportan los 5 anticuerpos que el microcontrolador generó para la población inicial, en 5 corridas diferentes. En esta misma tabla se listan las

generaciones necesarias para lograr el máximo número de unos en la cadena de 8 bits, así como el tiempo de convergencia. En este primer experimento se buscó el número máximo de generaciones para converger al óptimo (11111111, con afinidad 8) y el tiempo utilizado por el microcontrolador. La frecuencia de trabajo del PIC16F628A fue de 4MHz correspondiente al oscilador interno del dispositivo.

Se observa que en todos los casos se convergió al óptimo, lo cual demuestra en forma práctica que los operadores de mutación y de elitismo se seleccionaron adecuadamente para la micro-población y el crecimiento controlado a sólo 15 clones. El tiempo de ejecución de la corrida 5 fue el menor y el de la corrida 1 resultó el más alto. Si se revisan los anticuerpos generados en la población inicial de la corrida 5, se puede advertir que desde el principio se obtuvo un muy buen candidato con afinidad 7, el cual guió la búsqueda y aceleró la convergencia.

Tabla 4.1. Población inicial del micro-SIA implementado en el microcontrolador, generaciones y tiempo de convergencia.

Corrida	*Población inicial*	*Afinidad inicial*	*Generaciones necesarias*	*Mejor encontrado*	*Tiempo (segundos)*
1	10101011 11101001 00110101 11001000 00010010	5 5 4 3 2	1346	11111111	238
2	00110000 00010010 10111101 11001111 10011010	2 2 6 6 4	1102	11111111	194
3	10000011 10010010 00001001 10000001 00101111	3 3 2 2 5	1210	11111111	214
4	11101001 10111101 11000001 11000011 00111110	5 6 3 4 5	1322	11111111	234
5	11111011 01100110 00000001 11101101 10110110	7 4 1 6 5	746	11111111	132

El siguiente experimento realizado con el microcontrolador, consistió en mantener un número fijo de generaciones en el ciclo externo del algoritmo y observar cuál fue el mejor resultado obtenido al alcanzar el criterio de paro general. Para este experimento se realizaron 10 corridas del micro-SIA con un número de generaciones preestablecido y seleccionado considerando los resultados reportados en la tabla 4.1.

En la tabla 4.2 se reportan los resultados obtenidos en este experimento. Entre 1000 y 1500 generaciones se convergió al óptimo. Cuando se utilizaron pocas generaciones (100 y 500) no se obtuvieron buenas soluciones, no obstante, es notorio que el micro-SIA comienza a maximizar (en la particularidad de este problema) desde que inicia su ejecución, dado que hay un crecimiento estable de la afinidad al paso de las generaciones.

Tabla 4.2. Resultados experimentales del micro-SIA implementado en el microcontrolador con diferente número de generaciones.

Generaciones	*Mejor encontrado*	*Afinidad del mejor encontrado*	*Peor encontrado*	*Afinidad del peor encontrado*	*Media de la afinidad*
100	01101001	4	10000001	2	3
500	11001111	6	01001110	4	5.25
750	01111111	7	10110111	6	6.5
1000	11111111	8	11110111	7	7.8
1500	11111111	8	11111111	8	8

4.3 Implementación en el FPGA

El micro-SIA para el problema de maximización de unos, se implementó en un FPGA *Spartan 3A*, con número de parte *3S700A* del fabricante *Xilinx*. Los módulos diseñados en el numeral 4.2 para el microcontrolador son los mismos que se utilizan para el FPGA, con la observación de que en este último dispositivo es posible generar una arquitectura más robusta.

Los módulos fueron realizados en VHDL que es uno de los lenguajes descriptores de hardware más utilizados actualmente. Para este diseño se consideraron cadenas binarias de 16 bits, por lo que aplica el diagrama a bloques de la figura 4.4., asumiendo registros de datos de 16 bits. En la figura 4.6 se muestra una aproximación al código en VHDL que implementa el LFSR que genera los números pseudoaleatorios de 16 bits.

```
ARCHITECTURE arch_lfsr  OF lfsr_pseudo IS
 SIGNAL   q     : std_logic_vector(15 DOWNTO  0);
 SIGNAL   reset : std_logic;
 CONSTANT semilla  : std_logic_vector(15 DOWNTO  0):= (OTHERS  => '1');
BEGIN
 lfsr : PROCESS(clk,reset)
 BEGIN
  IF(reset='0') THEN
   q <= semilla;                        -- Inicio
  ELSIF (clk'EVENT AND  clk='1') THEN
   q(0) <= q(15);                       -- Realimentación
   q(1) <= q(0);
   q(2) <= q(1) XOR q(15);              -- Etapa 1
   q(3) <= q(2) XOR q(15);              -- Etapa 2
   q(4) <= q(3) XOR q(15);              -- Etapa 3
   q(15 DOWNTO 5)<= q(14 DOWNTO 4);
  END IF;

 END PROCESS lfsr;
END ARCHITECTURE arch_lfsr;
```

Figura 4.6. Arquitectura en VHDL para un LFSR de 16 bits.

En la figura 4.7 se presenta la arquitectura para el micro-SIA implementada en el FPGA. Obsérvese que las entidades diseñadas corresponden a las explicadas en el numeral 4.2. La síntesis lógica se realizó utilizando *ISE Webpack 12.4i* de *xilinx*. El paso de los individuos entre cada entidad se realiza de manera paralela.

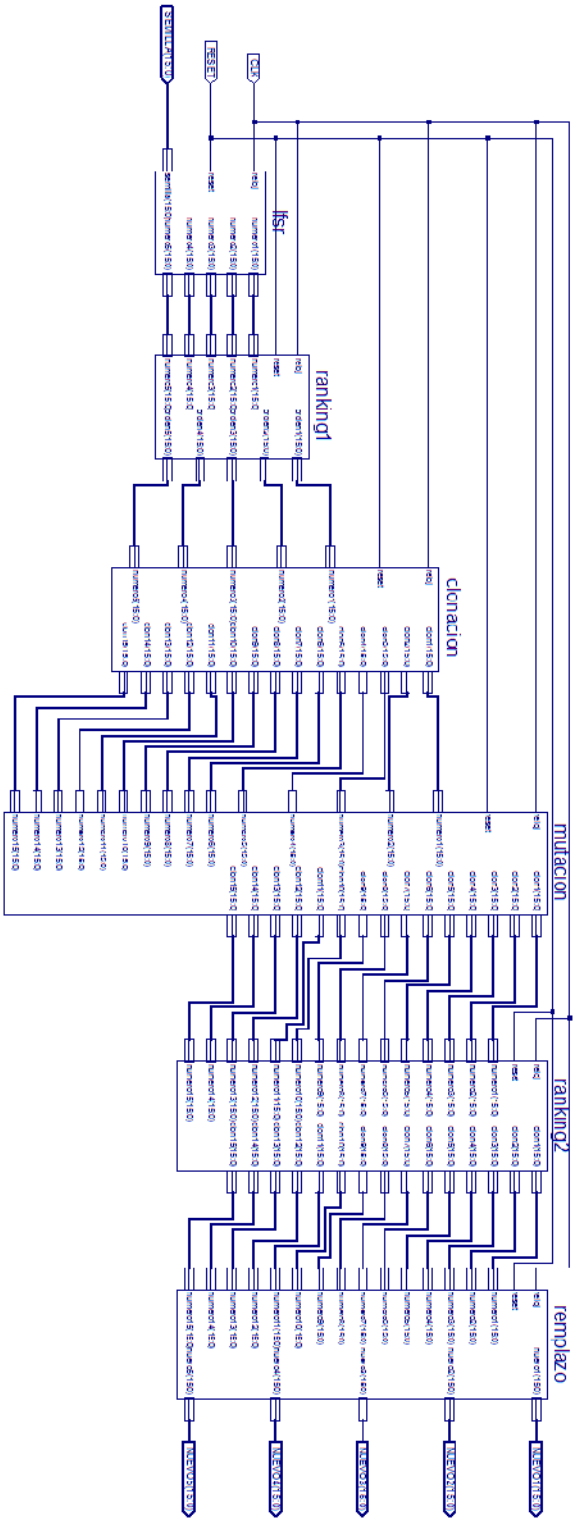


Figura 4.7. Arquitectura del micro-SIA en el FPGA.

4.3.1 Experimentación con el FPGA

El FPGA *3S700A* de Xilinx es un dispositivo con tecnología SRAM y 700,000 compuertas lógicas. La síntesis lógica del diagrama de la figura 4.7 utilizó el 32% de los recursos del FPGA y con el controlador para la comunicación con el puerto serial de la PC, designado para el monitoreo de datos, se utilizó el 37% de los recursos. De igual forma que en el caso del microcontrolador, se predispuso el diseño de una interfaz de comunicación serial (RS-232) sólo para leer los datos entregados por el FPGA. Del lado de la PC se habilitó una terminal que se sincronizó a 9600 baudios con el FPGA.

La frecuencia de trabajo de la arquitectura bioinspirada en el FPGA, es la misma frecuencia del oscilador incorporado en la tarjeta de desarrollo, es decir 50MHz.

El primer experimento con el micro-SIA en el FPGA consistió en revisar la probabilidad de convergencia a través del número de generaciones. Para tal propósito se realizaron 5 corridas del micro-SIA buscando el óptimo de la maximización de unos para una cadena de 16 bits.

En la tabla 4.3 se reportan los resultados de este experimento. En la última columna de esta misma tabla se anotó el tiempo que le tomó al FPGA resolver el problema. A diferencia de la tabla 4.1, referente al experimento equivalente con el microcontrolador, en la tabla 4.3 no se anotó la mejor solución obtenida, esto se hizo arbitrariamente debido a que todas las corridas convergieron al óptimo, que en este caso fue *1111111111111111*, es decir, con afinidad 16.

Este ejercicio considera 65535 combinaciones a razón de 2^{16}, por lo que no es un ejercicio sencillo para el micro-SIA en el FPGA. En la tabla 4.3 se aprecia que en la corrida 3, el micro-SIA convergió más rápidamente al óptimo, sin embargo, en la corrida 1 se utilizó el mayor número de generaciones y por ende el mayor tiempo de convergencia. Al igual que como se precisó en la primera experimentación con el microcontrolador, cuando se obtienen soluciones potenciales en la población inicial, la convergencia es más rápida.

Tabla 4.3. Resultados experimentales del micro-SIA implementado en el FPGA: generaciones y tiempo de convergencia.

Corrida	*Población inicial*	*Afinidad inicial*	*Generaciones necesarias*	*Tiempo (segundos)*
1	0111000000000001 1010100100110101 1001101000110101 1000100010100001 0001111001111101	4 8 8 5 10	55226	135
2	1100111101110101 0101001010101100 0000110010010000 0101000000010000 0001000000001111	11 7 4 3 5	10014	25
3	1001111110111101 0001001011111101 0001100110001100 1111000111111101 0110011110000000	12 9 6 12 6	7230	18
4	0100100100110101 0000001000001000 1100100000000001 1100100000111111 0010011100111110	7 2 4 9 9	12378	31
5	0011111011111011 1011111111011111 0000000110000000 0110101010101001 1010110010010011	7 14 2 8 8	17783	45

Para observar el efecto de las generaciones en el ciclo externo, se decidió realizar un segundo experimento. Cabe mencionar que tanto en el microcontrolador como en el FPGA, no se modificaron las características de la convergencia nominal, manteniendo 5 generaciones para alcanzarla; esto se debe a que por tratarse de parámetros sensibles, la variación de ambos números de generaciones repercutiría en resultados que no sería sencillo interpretar.

El micro-SIA para optimización numérica global que se diseñó en el capítulo 3, utiliza 10 generaciones para alcanzar la convergencia nominal; en contraparte, el micro-SIA en hardware sólo utiliza 5 generaciones. Se decidió fijar este número de generaciones debido a que experimentalmente se obtuvieron excelentes resultados.

En la tabla 4.4., se reportan los resultados obtenidos para este experimento. El micro-SIA se ejecutó 10 veces para cada número de generaciones preestablecidas, como se indica en la primera columna de la tabla.

Tabla 4.4. Resultados experimentales del micro-SIA implementado en el FPGA con diferente número de generaciones.

Generaciones	*Mejor encontrado*	*Afinidad del mejor encontrado*	*Peor encontrado*	*Afinidad del peor encontrado*	*Media de la afinidad*
100	0001001110010001	6	0001000000010001	3	3.75
500	0111000101010011	8	1000001000010100	4	4.5
1000	1111000011001100	8	0011001110010001	6	6.4
5000	1010011111100010	9	0011110011000000	6	7.75
10000	1111110011001111	12	0000110000111111	8	10.5
15000	1011111111111101	14	1101010111101000	9	12.8
20000	1111111111111111	16	1011111100111101	12	14.25

En los resultados de la tabla 4.4., se observa que entre las 15000 y 20000 generaciones, el micro-SIA convergió al óptimo. Existe un proceso de maximización gradual al paso de las generaciones, similar a lo que se observó en el ejercicio equivalente con el microcontrolador. Se tiene un comportamiento lineal que acelera la búsqueda en las últimas generaciones del algoritmo, tal y como se aprecia cuando se revisa la media de la corrida con 20000 generaciones en el ciclo externo del micro-SIA.

4.3.2 Aplicación para el reconocimiento de caracteres

El SIA ha sido aplicado con éxito dentro del área del reconocimiento de patrones y específicamente en el reconocimiento de caracteres. Por tal motivo se decidió realizar una aplicación que permitió la ejecución intrínseca del micro-SIA en un FPGA con el propósito de reconocer 14 caracteres predefinidos: 1, *2*, *3*, *4*, *5*, *6*, *7*, *8*, *9*, *A*, *E*, *I*, *O*, *U*. Nótese que no se consideró el caracter correspondiente al número cero (0), para no confundirse con el caracter de la letra *O*.

Se utilizó una matriz de 7 filas y 5 columnas (35 puntos) para dibujar cada caracter. Se predispuso esta configuración dado que en el mercado nacional de componentes electrónicos es común el uso de matrices de LEDs que concuerdan con estas características.

La matriz es fácilmente codificable en una cadena de 35 bits de la forma $Ag =< Ag_1, Ag_2, ..., Ag_L >$, en donde Ag representa los antígenos o caracteres preestablecidos, y en contraparte, para el caracter a reconocer se codifica de la forma $Ab =< Ab_1, Ab_2, ..., Ab_L >$, en donde Ab representa un anticuerpo. En la figura 4.8 se muestra un ejemplo de codificación para el caracter E.

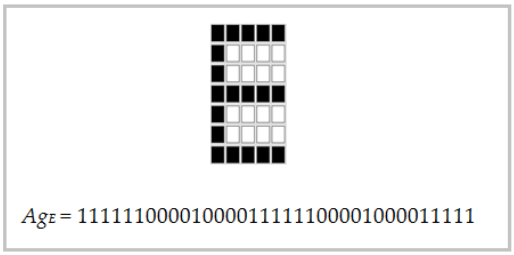


Figura 4.8. Ejemplo de codificación para la matriz de 35 puntos.

La aplicación diseñada conforma un sistema simple para el reconocimiento de los caracteres antes indicados. De inicio, se introduce un carácter corrupto que puede tener grandes variaciones con respecto a los caracteres originales, con ruido aditivo o sustractivo generado de manera arbitraria. Utilizando una distancia de *Hamming*, implementada con compuertas XOR, es sencillo comparar el caracter corrupto con cada uno de los caracteres predefinidos y asignar una medida de afinidad para cada una de estas comparaciones. En este sentido, este procedimiento de comparación es equivalente a la evaluación de la función objetivo.

En la figura 4.9 se ejemplifica el procedimiento para determinar la afinidad de un anticuerpo o caracter corrupto, es decir, cómo se le califica para saber qué tan bueno es el individuo como solución. Nótese que la comparación entre las cadenas de 35 bits se realiza bit por bit con una compuerta XOR, por lo que cuando los bits sean iguales se obtendrá un 0 lógico y cuando sean diferentes se obtendrá un 1 lógico. Se trata de un problema de minimización, dado que el óptimo de una comparación se obtendrá

cuando la cadena del resultado de comparación obtenga sus 35 bits ($C_{34},\ldots,C_0$) puestos a 0 lógico, que determina que ambos caracteres son idénticos.

El *módulo de conteo de unos* se utiliza para asignar el valor numérico de la afinidad, por ejemplo, la cadena del resultado de comparación (considerando los 35 bits) 00000111110000011111000001111100000 tiene una afinidad igual a 15; la cadena 00000000000000011111000000000000000 tiene una afinidad de 5 y sería una mejor solución.

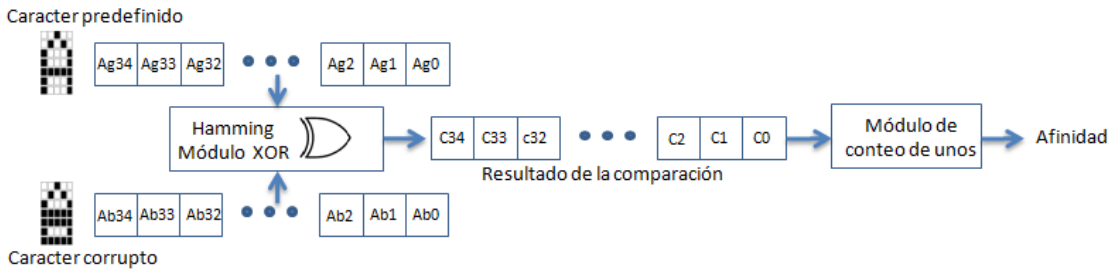


Figura 4.9. Procedimiento para asignar un valor de afinidad en la aplicación de reconocimiento de caracteres.

Se compara entonces el caracter corrupto con los 14 caracteres predefinidos y se obtendrá un valor de afinidad para cada comparación. Las comparaciones que obtienen las dos afinidades menores, serán las dos mejores soluciones. Para conformar la población inicial del micro-SIA se utilizan estos dos mejores individuos con otros tres que se generan aleatoriamente. La encomienda del micro-SIA es encontrar la mejor solución para el reconocimiento del caracter corrupto. En la figura 4.10 se presenta una aproximación al sistema de reconocimiento implementado.

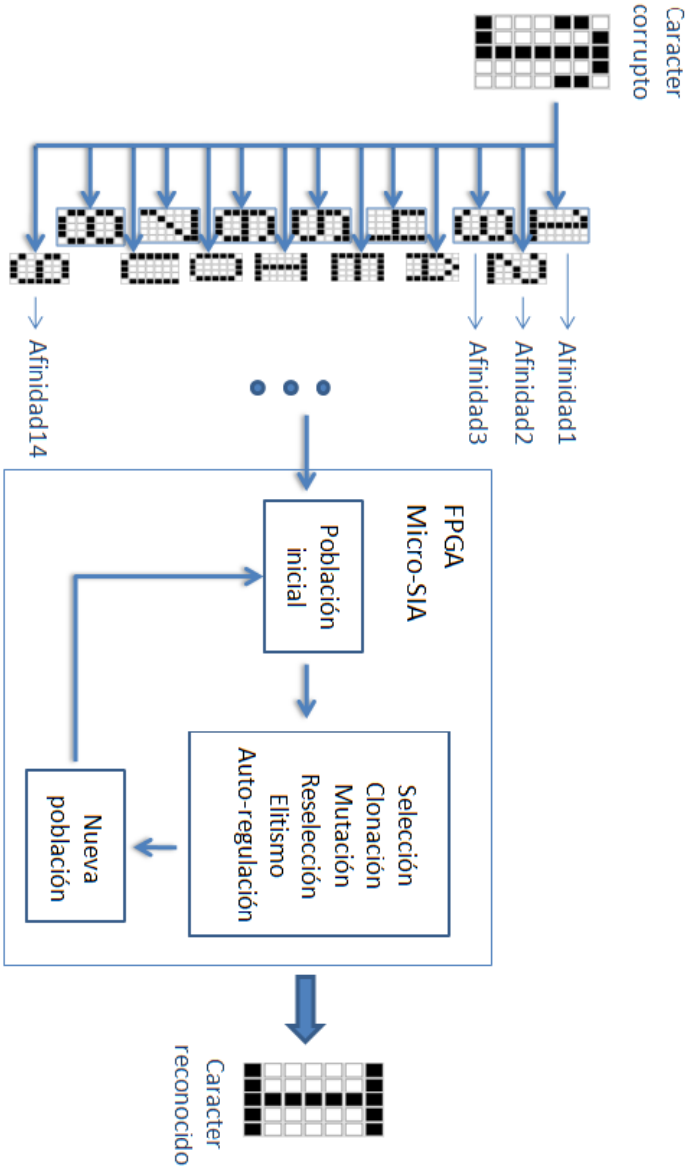


Figura 4.10. Reconocimiento de caracteres con el micro-SIA con ejecución intrínseca.

Las características del micro-SIA embebido en el FPGA son las siguientes: micro-población de 5 individuos (anticuerpos), individuos de 35 bits, 10 generaciones para alcanzar la convergencia nominal, 50 generaciones para la convergencia general (criterio de paro del algoritmo), 15 clones, mutación uniforme, elitismo para las dos mejores soluciones y ruido estocástico integrando tres clones elegidos aleatoriamente dentro de la convergencia nominal o tres individuos generados aleatoriamente si se

alcanzó la convergencia nominal. Puede observarse que se mantienen los parámetros y operadores utilizados en los experimentos anteriores, con la excepción de las 10 generaciones para alcanzar la convergencia nominal.

Para enviar el caracter corrupto al FPGA y leer los datos resultantes, se implementó una aplicación que permite desde una PC la interacción con el FPGA. Lo anterior se logra vía el puerto serial (RS-232). En la figura 4.11 se muestra la aplicación en software en tiempo de ejecución. Esta aplicación permite codificar el caracter, enviar la cadena de 35 bits por el puerto serial y decodificar una cadena de 35 bits que el FPGA devuelve como resultado del reconocimiento de caracteres.

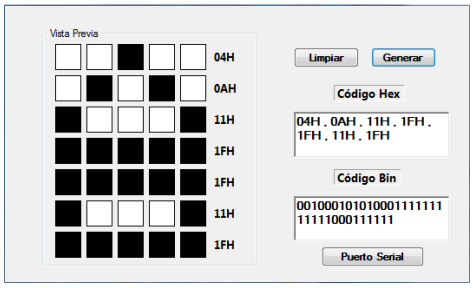
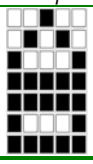
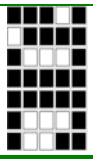
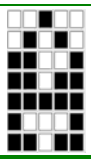
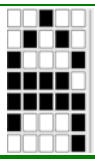
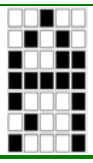
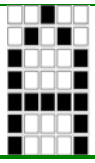


Figura 4.11. Aplicación para codificar y decodificar que permite la interacción con el FPGA.

En la tabla 4.5 se muestra a detalle cómo el micro-SIA evoluciona un caracter. En la primera columna se aprecia el caracter corrupto y las columnas posteriores muestran las modificaciones sufridas en las generaciones indicadas. Estas generaciones hacen referencia al ciclo externo del micro-SIA.

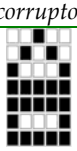
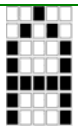
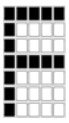
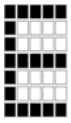
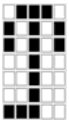
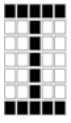
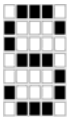
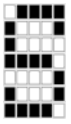
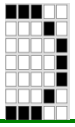
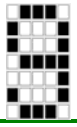
Tabla 4.5. Evolución de un caracter por medio del micro-SIA.

Caracter corrupto	*Generación 10*	*Generación 20*	*Generación 30*	*Generación 40*	*Generación 50 Salida*
					

El micro-SIA convergió a las 50 generaciones, encontrando la mejor solución. El caracter corrupto utilizado para este experimento tiene ruido aditivo generado de manera arbitraria.

En la tabla 4.6 se reportan los resultados de 5 caracteres corruptos introducidos al sistema de reconocimiento. Se realizaron 5 ejecuciones de cada uno de éstos y los resultados fueron siempre los mismos. Los caracteres utilizados tienen ruido aditivo o sustractivo, también generado de manera arbitraria sin alguna tendencia. El último de los caracteres es muy distinto a cualquiera de los caracteres predefinidos en el sistema, sin embargo, el caracter reconocido fue reiteradamente el mismo que se reporta en la tabla 4.6.

Tabla 4.6. Resultados experimentales para 5 caracteres corruptos.

Carácter corrupto	*Generación 50 Salida*
	
	
	
	
	

Para estos experimentos no se midió el tiempo de ejecución debido a que intencionalmente se incluyeron retardos para monitorear las modificaciones de los caracteres al paso de las generaciones. De acuerdo con los resultados reportados en la tabla 4.6 el micro-SIA aplicado al reconocimiento de caracteres funcionó correctamente al obtener el caracter más parecido a uno corrupto. Aún tratándose de una cadena de 35 bits, con 2^{35} posibles combinaciones diferentes, el micro-SIA encontró muy buenas soluciones. Resultó ventajoso que desde la primera población generada se utilizó elitismo, las mejores soluciones son prometedoras y guían la búsqueda, acelerando la convergencia sin la necesidad de explorar la totalidad de las combinaciones posibles.

La arquitectura bioinspirada que se diseñó es robusta y permite cambios sencillos y drásticos, ya sea para reducir o aumentar el número de caracteres predefinidos. La ejecución paralela en el manejo de datos se conserva en similitud al diagrama de la figura 4.7. La única limitante es la cantidad de recursos internos del FPGA, que para este diseño y considerando la interfaz con el puerto serial, se utilizó el 63% de la capacidad total del dispositivo.