Universidad Autónoma de Querétaro

Facultad de Ingeniería

Maestría en Instrumentación y Control Automótico

Diseño e Implementación de un algoritmo genético en FPGA para sintonización de controladores PID

TESIS

Que como parte de los requisitos para obtener el grado de

Maestro en Ciencias en Instrumentación y Control Automótico

Presenta:

Ing. Carlos Ricardo Luna Ortiz

Dirigido por:

Dr. Luis Alfonso Franco Gasca

SINODALES

Dr. Luis Alfonso Franco Gasca		
Presidente		Firma
Dr. Manuel Toledano Ayala		
Secretario		Firma
Dr. Edgar Alejandro Rivas Araiza		
Vocal		Firma
M.C. Gonzalo Macías Bobadilla		
Suplente		Firma
M.I. Jesús Roberto Millán Almaraz		
Suplente		Firma

Dr. Gilberto Herrera Ruiz	Dr. Luis Gerardo Hernández Sandoval
Director de la Facultad	Director de Investigación y Postgrado

Centro Universitario
Querétaro, Qro.
Mayo del 2011
México

Resumen

A pesar de la rápida evolución en los sistemas de control de procesos, el controlador PID (Proporcional-Integral-Derivativo) es el más utilizado en la industria, debido a que sus funciones simples y robustas son suficientes para trabajar en una gran variedad de procesos. Sin embargo, resulta complejo sintonizar de manera adecuada las ganancias del PID, debido a que los procesos no son sistemas lineales, ni estacionarios, de modo que sus características cambian con el transcurso del tiempo. El proceso de sintonización no es trivial, ya que en gran medida depende del proceso a ser controlado, y no existe un método que sea apropiado en todas las aplicaciones. Ante esto surgen los sistemas de control adaptable, donde el controlador realiza una auto-sintonización. Se han diseñado estrategias muy sofisticadas alrededor del PID clásico, entre estas se cuentan las técnicas de inteligencia artificial.En este trabajo se diseña un algoritmo genético capaz de sitonizar controladores PID, dicho algoritmo se implementa en FPGA.

Antes de realizar la implementación del algoritmo genético capaz de sintonizar controladores PID en FPGA se lleva a cabo la etapa de adaptación del algoritmo genético genérico al uno capaz de sintonizar controladores PID, dicha etapa se desarrolla en software.Posteriormente se adapta y migra el algoritmo desarrollado a hardware, es decir, en FPGA. Al final a través de simulaciones se aprecia que la implementación en hardware es más eficiente.Con la finalidad de validar el algoritmo desarrollado, se utiliza un sencillo esquema de control de velocidad de un motor de CD, el cual permite utilizar y el algoritmo genético implementado en FPGA.

(**Palabras clave:** FPGA,Algoritmo Genético,PID,Sintonización)

Summary

In spite the fact that process control systems have evolve really fast, the PID controller is broadly used in industries due to the fact that its functionallity is simple and robust which makes it suitable for tha vast mayority of task at hand. However tuning this controller properlly turns out to be complex because processes most of the times are neither linear nor stationary, which implies that they are time varying.Since the tuning method depends on the process for which it will be tuned achieving a proper tuning is not trivial.One option to tackle this difficulties is using adaptative control systems which are capable of sel-tuned themselves.Really sofisticated strategies have been design using classic PID controller as a baseline, one branch of this strategies are the ones based on artificial intelligence.In this work a genetic algorithm capable of tuning PID controllers is developed and implemented in a FPGA.First of all an adaptation from a generic genetic to one capable of tuning PID controllers is carried out.In this stage all work is done in software.Once software verision is done, the second stage takes place.In this stage an adaptation of the software version to hardware version is performed.At the end of this work simulations presented in both platforms software and hardware show that the latter is more efficient.Last but not least, a control feedback loop using a DC engine is used to show and validate FPGA implementation of genetic algorithm capable of tuning PID controllers.

(**Key words:** FPGA,Genetic Algorithm,PID,Tuning)

A Dios y mis seres queridos

Índice general

Índice de figuras

Indice de Cuadros

1 INTRODUCCIÓN

La variedad de procesos y sistemas que se desean controlar es muy basta, sería muy ambicioso pretender crear un controlador que presente un desempeño aceptable en todos los casos.Sin embargo, en la mayoría de los casos, tal como se menciona en (Aström and Hägglund, 2001) la retroalimentación es un concepto que por si mismo ha revolucionado la manera en como se realiza el control. Y a pesar de que gran parte del esta revolución se debe a la sola idea de la retroalimentación, en ese mismo artículo también se señala que el controlador PID es por mucho la forma más utilizada de control por retroalimentación.Esto se debe a las características que este tipo de controlador presenta.El controlador PID quizá no es el más adecuado para algún proceso en particular, sin embargo,sí es el controlador que prefieren las industrias.

Uno de los aspectos más importantes a considerar del controlador PID es la estructura del mismo, la cual tiene que ver con como es que se procesa el error, de antemano se sabe que todo control PID debe ser capaz de manejar las ganancias proporcional, derivativa e integral, sin embargo, existen diversas maneras de hacerlo, es decir, existen diversas estructuras de PID, que se pueden utilizar y que cumplen con estas características.

Sin importar cual estructura se haya elegido para el controlador, se debe considerar que éste deber ser sintonizado, es decir, se deben elegir valores para cada un de las ganancias del controlador, esto es lo que se conoce como sintonización. Este proceso de sintonización es necesario que se realice con cada nueva planta o cuando el desempeño no es satisfactorio, este característica y necesidad del controlador es lo que lo hace tan versátil.

El proceso de sintonización de un controlador PID no es trivial, sino más bien todo lo contrario, define en gran medida el desempeño del sistema en general.Supóngase que un controlador de tipo PID se utiliza en el control de un sistema o proceso, ya sea que éste sea el controlador más adecuado o no, si se sintoniza de una manera adecuada, los resultados obtenidos, serán más satisfactorios que si la sintonización no es adecuada.

Es importante remarcar que, existen varias técnicas que permiten la sintonización de este tipo de controladores. Cada una de esas técnicas se ha desarrollado teniendo en mente aspectos particulares del PID o los procesos en los cuales se piensa utilizar el controlador PID. A pesar de la variedad de técnicas que se han reportado en la literatura, en las industrias esto no parece haber dado respuesta a sus necesidades.

Las técnicas de sintonización convencionales funcionan bien para casos muy particulares de sistemas o procesos, sin embargo, como dependen mucho del proceso, si este cambia en algún modo, estas técnicas dejan se ser efectivas.

Para resolver el problema de la sintonización de procesos o sistemas que son muy cambiantes, se han propuesto diversas técnicas. Actualmente se estudian de manera seria las técnicas adaptables. Dentro de éstas, existe un tipo de ellas, las basadas en heurística, las cuales según se ha estudiado presentan resultados satisfactorios y un grado de robustez importante, sin embargo, tienen problemas debido a la cantidad de cálculos que se requiere realizar para llevar a cabo este tipo de métodos.Los algoritmos genéticos son una de dichas técnicas heurísticas.La idea general de esta técnica es encontrar una solución a un problema a través de la optimización de un criterio específico, esto se hace utilizando un espacio de búsqueda que se define en un principio y que va evolucionando conforme el algoritmo avanza. Esta idea de evolución toma conceptos de la misma evolución de las especies en la naturaleza. Es importante remarcar que esta técnica presupone que no se tiene la menor idea de cual podrá ser la mejor solución al problema al iniciar el algoritmo.

Los algoritmos genéticos se pueden utilizar para encontrar solución a problemas, de los cuales no se sabe nada a cerca de la misma. Sólo se sabe qué condiciones debe satisfacer la solución. Gracias a que los algoritmos genéticos de alguna manera evolucionan y por lo tanto se adaptan a las condiciones dadas, se pueden utilizar en problemas cuyos parámetros vayan cambiando, tal como sucede en la evolución del ser humano, se adaptan a las condiciones cambiantes del entorno donde vivimos.

Así pues, si se tiene un sistema o proceso que cambia con el tiempo y un controlador PID al cual se le deben sintonizar sus ganancias para ser eficiente ante las nuevas condiciones del problema, entonces es posible aplicar un algoritmo genético que se encargue de la sintonización (adaptación) de dichas ganancias, es decir, que sintonice al PID.

Ya se han reportado artículos donde se utiliza un algoritmo genético para la sintonización de las ganancias de controladores de tipo PID. Los resultados obtenidos en la mayoría de las ocasiones han sido satisfactorios o al menos prometedores. Es importante remarcar que a pesar de que los resultados son satisfactorios, la mayoría de ellos provienen de simulaciones. En los casos donde no provienen de simulaciones, se han realizado implementaciones donde se tiene como plataforma de cálculo una computadora de escritorio, lo cual desde luego es muy válido y en algunos casos quizá deseable, sin embargo si se piensa en desarrollar un sistema de control que no sea una computadora de escritorio, lo cual es común en ambientes industriales donde el control no es centralizado, entonces es deseable contar con este tipo de algoritmos en sistemas embebidos como microcontroladores o FPGAS.

Se debe enfatizar en el problema que se señalaba anteriormente, el cual hacía referencia a que las técnicas heurísticas a pesar de presentar resultados aceptables y ser robustas requieren de una cantidad de cálculos considerables, ésta es quizá una de las razones más importantes por las cuales no se han hecho muchas implementaciones que

se puedan considerar como aplicables en términos prácticos.

Cada vez que se requiere procesar información se piensa en la tecnología con la que se cuenta. Como es bien sabido, no existe una sola plataforma tecnológica de procesamiento de información que sea adecuada para todos los casos.

En la actualidad las tecnologías basadas en procesadores son muy atractivas, sin embargo, cuando se trata de una cantidad de operaciones muy extensa y de propósito específico, como es el caso de los algoritmos genéticos, quizá esta tecnología no sea la más adecuada. Es posible pensar que un procesador dedicado al procesamiento de señales (DSP) sea un candidato para esta tarea, sin embargo, a pesar de que es un procesador dedicado al procesamiento de información, siempre tiene la limitante de su naturaleza secuencial y de su estructura de hardware fija. Estos puntos son importantes, ya que una característica de los algoritmos genéticos es que realizan la busqueda de la mejor solución de forma paralela, por lo cual un DSP no contribuye a este hecho.

Así pues si lo que se requiere es gran capacidad de procesamiento y posibilidad de paralelismo en el mismo, una solución muy natural es un FPGA.

1.1. Descripción del Problema

Uno de los objetivos más importantes de control es poder llevar a cabo la construcción de un controlador para que el proceso o sistema en cuestión, es decir, la planta con la que se cuenta, responda a una entrada en una forma predecible y deseada, y en algunas ocasiones también es importante que esta respuesta se mantenga a pesar de la presencia de señales que se encuentran presentes de manera no deseado, ya se ruido en la entrada o perturbaciones en el entorno.

Para lograr de manera satisfactoria la construcción del controlador anteriormente mencionado, se cuenta con muchas técnicas que, dependen en gran medida de

la planta o sistema que se quiera controlar. Sin embargo, en muchos casos es común el uso de un controlador de tipo PID.

El PID se ha estudiado ya desde hace tiempo, y mucho se ha dicho al respecto. Sin embargo, debido a que los campos donde se ha aplicado el PID son tan diversos, aún en la actualidad siguen surgiendo cosas novedosas acerca de este tipo de controladores.

Uno de los aspectos que de alguna manera son complejos en el controlador PID es la sintonización del mismo, es decir, cuánto deben valer cada una de las ganancias (proporcional, derivativa e integral) para que el controlador tenga el desempeño esperado.

Existen diversas técnicas que permiten la sintonización del PID. Varias de estas técnicas presuponen que la planta exhibe resultados muy similares para entradas muy similares, es decir, que de alguna manera la planta no varía ni con el tiempo ni con el rango de operación.

Cuando se cuenta con un sistema que se considera como invariante en el tiempo, las técnicas convencionales de sintonización de PID son aceptables y hasta cierta medida funcionan de manera eficiente. Sin embargo, estas técnicas no toman en cuenta un posible cambio en la respuesta del sistema a una misma entrada, ya sea por cambio del sistema o de la carga aplicada,es decir, que si el sistema cambia su repuesta, es posible que la sintonización que se realizó de manera previa no sea la adecuada debido al cambio de en la respuesta del sistema.

1.2. Justificación

Cuando se diseña un sistema se desea que el sistema tenga un desempeño satisfactorio. Para poder conseguir un desempeño satisfactorio es necesario cumplir varios requisitos. Uno de ellos es que el problema para el cual fue diseñado el sistema se haya

identificado de manera adecuada, ya que si el problema no se identifica de manera adecuada y se diseña un controlador, es posible que éste no tenga un desempeño y resultado adecuado.

Suponiendo que se ha identificado bien el problema a resolver, otras dificultades se pueden presentar ocasionando que un sistema no tenga un desempeño adecuado. Un ejemplo de ello es el diseño del sistema, es decir, que el sistema no se haya diseñado para atacar el problema de forma adecuada. Es así, como este problema involucra de manera directa el conocimiento y experiencia del diseñador.

El siguiente problema que se puede presentar es que, a pesar de que el diseño sea el correcto, la elaboración del mismo no sea la adecuada. Esto puede deberse a distintas razones, que van desde una mala interpretación del diseño hasta una mala selección de los componentes que conforman el sistema.

Si los problemas previos se han superado y a pesar de ello el sistema no presenta un desempeño como el deseado, es muy probable que lo que suceda sea que no está ajustado o o no cuenta con las opciones adecuadas. Hablando particularmente de un sistema de control esto se refiere al hecho de que el controlador no tenga las características que debiera para poder desempeñar su función de manera satisfactoria.

En un mundo como el actual, es casi imposible pensar en un sistema eficiente que tenga que ser operado de forma manual. Sin embargo, es increíble ver que a pesar de la basta teoría de control con la que se cuenta en la actualidad, existen muchos procesos y aplicaciones que aún operan de forma manual.

No obstante,también existen industrias en las cuales se tienen sistemas que operan de forma automática. Es importante señalar que, de los procesos que no se controlan de manera manual, hablando de industrias de procesos y manufactura, el 90 % de los sistemas o procesos utilizan un controlador de tipo PID (Yu, 2006).

En las industrias de procesos, más del 97 % de los controladores dedicados a la regulación son de tipo PID Yu(Yu, 2006). Se pensaría que a más de 60 años de la publicación de Ziegler y Nichols relacionada con la sintonización del PID, se pudiesen conseguir resultados muy aceptables de diseño de PID en los sistemas, sin embargo, la realidad indica algo muy diferente. Las siguientes cifras dan pauta para reflexionar a cerca de la situación que se vive a ese respecto. Sólo el 20 % de los lazos de control con PID trabajan como se espera. Del 70 % restante el 30 % de los lazos de control tienen un desempeño pobre debido a una sintonización deficiente del controlador.

Aunado de los datos mencionados anteriormente, se encuentra el hecho de que los ingenieros y administradores de empresas de procesos y manufactura, señalan la sintonización de los controladores de tipo PID como un problema difícil de resolver.

Existe evidencia fuerte de que los controladores de tipo PI y PID aún son entendidos de manera muy carente, y que en particular, pobremente sintonizados en muchas aplicaciones. Es claro que las muchas reglas propuestas en la literatura no están teniendo un impacto industrial (O'Dwyer, 2006).

Así pues, contribuir a la solución de este problema es de utilidad a una gran cantidad de industrias, de hecho a casi cualquiera que maneje en sus sistemas lazos de control con controladores PID que presenten perturbaciones en la carga.

Si hablamos de México, sabemos que existe un buen número de empresas que cuentan con sistemas de control y en esos sistemas se emplean controladores PID. Por poner un ejemplo más específico, se puede pensar que en PEMEX existe un lazo de control en el cual se implementa un controlador de tipo PID, y que la carga es variable, este lazo de control es candidato para poder usar lo que se está proponiendo en esta tesis, claro es que la propuesta por si sola no resolvería el problema en su totalidad, sino más bien propone un herramienta que se puede utilizar en un sistema de control

adaptable basado en controlador PID.

Además de las posibles aplicaciones prácticas en las cuales se pueda utilizar este trabajo, también se prestara como un antecedente de la utilización de la inteligencia artificial y la heurística para la incorporación como herramientas para la solución de problemas de control.

1.3. Objetivos

El objetivo principal de este trabajo es diseñar e implementar un algoritmo genético en FPGA capaz de sintonizar un controlador PID digital.Para poder alcanzar el objetivo principal se plantean los siguientes objetivos particulares

- Diseñar un algoritmo genético capaz de sintonizar un controlador PID.
- Probar el algoritmo genético en software para validar su funcionamiento.
- Implementar el algoritmo genético en FPGA para mejorar la velocidad de ejecución.
- Validar la implementación del algoritmo genético en hardware.

1.4. Hipótesis

Es posible diseñar e implementar un algoritmo genético en FPGA para la sintonización de controladores PID, reduciendo el tiempo de cómputo de sistemas análogos basados en software

1.5. Resumen

En el capítulo presente se han expuesto las razones que justifican el desarrollo del presente trabajo, así como los objetivos esperados y la hipótesis central de esta tesis.Esto a través de la introducción a lo sistemas de control retroalimentados y al controlador PID.

En el capítulo dos, se realiza una revisión de literatura para explorar el contexto en donde se ubica este trabajo.En este capítulo se presenta la revisión de documentos

que se encuentran en el estado del arte a cerca de la sintonización de los controladores PID.

Continuando con la secuencia del trabajo, se encuentra el capítulo tres, en el cual se presenta la metodología que se utilizada para alcanzar el objetivo de esta tesis.Para dicho fin, el capítulo se ha dividido en cuatro partes.La primera parte se enfoca en el controlador PID y la versión de este que será implementada en este trabajo.En la segunda parte se presenta una introducción a lo que son los algoritmos genéticos y los principales por menores de los mismos.En la tercera parte se muestra el diseño de un algorimo genético desarrollado para la sintonización de controladores PID, lo cual permite que se enlacen la primera y segunda parte de este trabajo, al mismo tiempo que se toma como preámbulo para la parte medular de esta tesis que es la implementación en hardware de un algoritmo genético capaz de sintonizar controladores PID, dicha tarea se presenta en la parte final de este capítulo.

En el capítulo cuarto y final de este trabajo, se presentan los resultados obtenidos, tanto en software como en hardware de la implementación del algoritmo genético.En primera instancia se presentan simulaciones del algoritmo genético genérico desarrollado,esto es, se muestran las gráficas de como el algoritmo genético es capaz de encontrar el mínimo de una función multivariable.Posteriormente se presentan las gráficas de el algoritmo genético desarrollado para la sintonización de controladores PID, la versión en software.En este caso se presenta los resultados que tiene el algoritmo tanto para una planta lineal como para una no lineal.Y final mente se presentan los resultados obtenidos de la implementación en hardware del algoritmo genético capaz de sintonizar controladores PID.

2 ANTECEDENTES

Los sistemas de control tienen como uno de sus principales propósitos que la salida mostrada por un sistema sea de acuerdo a la referencia que se le impone, en la mayoría de los casos se busca que la salida siga a la referencia.Para poder realizar esto se cuenta con diversas topologías.En un principio se utilizaban esquemas en lazo abierto como el que se muestra en la Figura2.1.

Figura 2.1. Control en lazo abierto

En la Figura 2.1 se puede apreciar que se tiene una referencia que de manera directa se convierte en la entrada de la planta, la cual desde luego produce una respuesta a esta entrada.Es importante hacer notar que en este esquema de control la respuesta de la planta depende de la referencia y de la dinámica del sistema, lo cual trae como consecuencia que la salida quede supeditada en gran medida a la dinámica de la planta.Este esquema sólo es utilizado cuando la dinámica del sistema ofrece respuestas que cumplan con los requerimientos de desempeño que se esperan, sin embargo como en la mayoría de los casos no es así, se buscan nuevas alternativas a esta forma de realizar el control.

Así pues en la Figura 2.2 se muestra una nueva idea de hacer control.En dicha figura se aprecia que por lo pronto la entrada de la planta no es la referencia de manera directa, sino que es la señal de salida de un bloque que se ha denominado como controlador.Además de esta diferencia,quizá más importante aún es el hecho de que la salida se compara con la referencia para generar la entrada al controlador que eventual-

mente se convertirá en la entrada de la planta, después de ser procesada.Y en efecto esta topología es la que se conoce como retroalimentación negativa en lazo cerrado, la cual ha revolucionó la forma de realizar control.

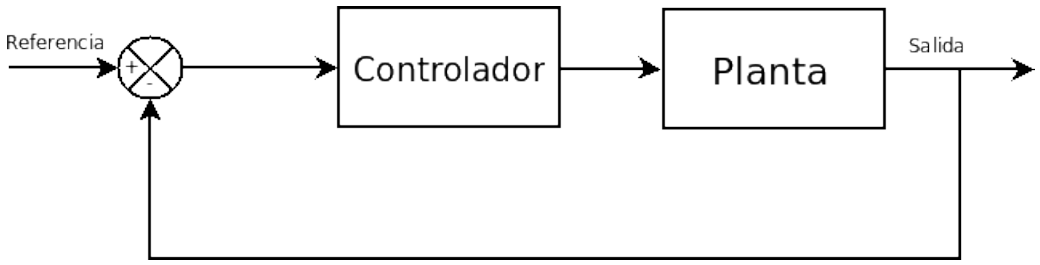

Figura 2.2. Control en lazo abierto

2.1. El controlador PID

2.1.1. Generalidades

Cuando se habla de control automático es casi imposible no pensar en la idea de contar con la retroalimentación. Como señala Astrom (Aström and Hägglund, 2001) la verdad es que como a lo largo de la historia se ha atribuido el éxito del control a cierta tipo de topología de la realimentación, sin embargo, en realidad dicho éxito estriba más bien en la realimentación en sí.

En dicho esquema se tiene que la entrada al controlador es una señal que se conoce como de error, esta señal es la diferencia que existe entre la referencia y la señal medida a la salida de la planta, esta señal ayuda a que de alguna manera el sistema sea consciente de la respuesta, es decir, que si hay discrepancia entre lo deseado y lo obtenido de manera real, el controlador trate de cambiar esta circunstancia.

Es importante señalar que utilizar la realimentación contribuye al problema de control, sin embargo no lo resuelve por completo, claro es, que se debe utilizar un controlador. La decisión de que controlador utilizar no es sencilla, ya que en realidad esta decisión depende del problema en cuestión, esto es, existen controladores de diverso tipos para distintos propósitos.

A pesar de que elegir un controlador u otro no es una decisión trivial y evidentemente no tiene una respuesta única, existe un controlador que se utiliza ampliamente en el control automático, a saber, el controlador PID, por sus siglas en inglés (Proportional Integral Derivative). Este controlador utiliza la señal de error del sistema y la procesa de tres maneras diferentes y muy útiles además de intuitivas, dichas maneras son: lo amplifica o atenúa, lo deriva y lo integra, esto es muy abstracto y matemático, sin embargo si se trata de interpretar el significado de cada una de estas acciones realizadas se puede decir que el estado presente se ve afectado por la parte amplificadora, de alguna manera se tiene idea del pasado, esto mediante la acción integral e incluso se puede decir que de alguna forma se puede predecir lo que sucederá, esto es gracias a la acción derivativa del control.

De manera formal el controlador PID se puede expresar como

$$g_c(t) = K_p e(t) + K_d \frac{de(t)}{dt} + K_i \int e(t)dt \tag{2.1}$$

esto es en su representación temporal y tomando la señal del error como $e(t)$, sin embargo en control es muy común utilizar la representación en s, es decir, utilizando la transformada de Laplace, esta representación es como sigue

$$G_c(s) = K_p E(s) + K_d s E(s) + \frac{K_p E(s)}{s} = \left(K_p + sK_d + \frac{K_i}{s} \right) E(s) \tag{2.2}$$

El controlador PID se utiliza ampliamente en la industria, de hecho como señala O'Dwyer(O'Dwyer, 2006), la habilidad de los controladores PI y PID para compensar la mayoría de los procesos industriales ha contribuido a que tenga una aceptación amplia en la industria.

2.1.2. Sintonización del controlador PID

Introducción

Es importante señalar que a pesar de que el controlador PID es versátil, se utiliza en gran cantidad de procesos industriales, se deben seleccionar los valores las ganancias proporcional, integral y derivativa de forma adecuada, ya que de la selección de los valores de dichas ganancias depende el desempeño del controlador. Seleccionar estas ganancias no es trivial, ya que se puede decir que esto hace que para cada uno de las plantas distintas el PID se pueda utilizar de forma satisfactoria. Así pues, seleccionar las ganancias o *sintonizar*, como comúnmente se le conoce, es una tarea fundamental para el buen desempeño del controlador PID.

Especificaciones

Como se menciona Astrom (Aström and Hägglund, 1995), cuando se resuelve un problema de control es necesario tener en mente cual es el principal objetivo del sistema de control, es común que dicho objetivo sea seguir una referencia dentro de un rango o tener un rechazo a perturbaciones. Además de tener en mente el objetivo que se pretende alcanzar, es importante saber bajo que circunstancias se debe llevar a cabo, es decir, restricciones impuestas por la dinámica del sistema, no linealidades, perturbaciones incluso posible incertidumbre en el proceso.

Cuando se sintoniza un controlador en general y un PID en particular es importante que se tomen en cuenta las especificaciones antes mencionadas, ya que la sintonización depende en gran medida de ellas.

2.1.3. Métodos de sintonización

Métodos Clásicos

A continuación se menciona de manera breve una serie de métodos clásicos de sintonización de controladores PID, esta información se ha tomado de Astrom(Aström and Hägglund, 1995).

Ziegler-Nichols

Son dos los métodos propuestos por Ziegler-Nichols y se utilizan de manera muy extensa, ya sea en su forma original o con algunas modificaciones. Estos métodos se basan en la determinación de algunas características de la dinámica de la planta a controlar y posteriormente los parámetros del controlador se expresan en fórmulas que relacionan los parámetros del controlador con los obtenidos de manera experimental.

A continuación se presentan figuras de como se lleva a cabo la determinación de los parámetros que permite la sintonización del PID. Primero para el método basado en la respuesta en lazo abierto al escalón se tiene que a partir de una figura como la que se muestra en Figura 2.3 como la siguiente se obtienen los parámetros a y L para posteriormente sintonizar el PID.

Con los parámetros obtenidos de la como se muestra en la Figura 2.3 se utilizan las fórmulas mostradas en la Tabla 2.1.

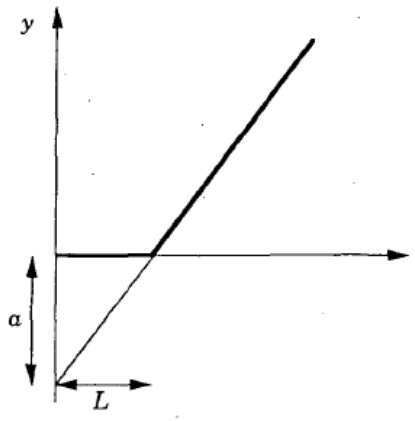

Figura 2.3. Método de Ziegler-Nichols: Respuesta al escalón

De manera similar en la Tabla 2.2 se presenta un resumen de como se eligen los valores para la sintonización de un controlador PID, esto basado en su repuesta en frecuencia. Los parámetros que se utilizan son K_u última ganancia y T_u último periodo, dichos valores se obtiene poniendo el controlador PID en cascada con la planta y en lazo abierto hacer que el sistema oscile sin utilizar las ganancias integral ni derivativa,

Controlador	K	T_i	K_d	T_p
P	$1/a$			$4L$
PI	$0.9a$	$3L$		$5.7L$
PID	$1.2a$	$2L$	$L/2$	$3.4L$

Cuadro 2.1. Método de Ziegler-Nichols: Fórmulas para respuesta al escalón

sólo la proporcional, cuando empiece a oscilar esa es la K_u y el periodo de oscilación es T_u

Controlador	K	T_i	K_d	T_p
P	$0.5K_u$			T_u
PI	$0.4K_u$	$0.8T_u$		$1.4T_u$
PID	$0.6K_u$	$0.5T_u$	$0.125T_u$	$0.85T_u$

Cuadro 2.2. Método de Ziegler-Nichols: Fórmulas para respuesta en frecuencia

Existen algunas versiones modificadas del método de Ziegler-Nichols, que proponen distintas fórmulas dependiendo del desempeño que se desee, por ejemplo Chien,Hrones y Reswick proponen las fórmulas mostradas en a Tabla 2.3.

Sobrepaso	0 %			20 %		
Controlador	K	T_i	K_d	K	T_i	K_d
P	$0.3/a$			$0.7/a$		
PI	$0.6/a$	$4L$		$0.3/a$	$2.3L$	
PID	$0.95/a$	$2.4L$	$0.42L$	$1.2a$	$2L$	$0.42L$

Cuadro 2.3. Método de Chien,Hrones y Reswick

Otros

Existe varios métodos distintos al método de propuesto por Ziegler-Nichols, sin embargo, utilizando éste y sus derivados se obtienen resultados aceptables en la may-

oría de los casos.Debido a que los resultados obtenidos con el método de Ziegler-Nichols son aceptados por los usuarios la mayoría de las ocasiones, poco se menciona de los métodos restantes. A continuación se dará un breve resumen de los métodos que no son derivados del propuesto por Ziegler y Nichols.

Primero se encuentran los métodos en los cuales se trata de dar forma al lazo, es decir, se trata de cambiar el valor de la función de transferencia en un punto o serie de puntos mediante los valores otorgados a las ganancias del controlador. Es fácil darse cuenta que sólo se necesitan dos parámetros para forzar a que se tenga un valor deseado a cierta frecuencia, pero como se dispone de tres parámetros en el PID, entonces se dice que se cuenta con un grado de libertad, lo cual se puede aprovechar para darle cierta forma al lazo, y con ello conseguir mejoras en la respuesta.

También existen métodos, en los cuales se encuentra el modelo de la planta y mediante procedimientos analíticos se trata de dar valores a los parámetros del PID para que cuando se combine en cascada con la planta y se realice la retroalimentación, el sistema tenga los polos y ceros de tal forma que el resultado sea el deseado, un ejemplo de estos métodos es del de Haalman, en el cual se puede profundizar en (Aström and Hägglund, 2006).

Además de los métodos mencionados con anterioridad existen los que se basan en la optimización. Dichos métodos tratan de optimizar el criterio de diseño más importante, esto lo logran generando una o varias funciones o desigualdades, en las cuales involucren los parámetros del PID y el criterio de mayor interés en el diseño. A pesar de que estos métodos son una herramienta poderosa, se debe tener cuidado en ciertos aspectos, uno de ellos es el diseño de la función o desigualdad que se utilizará para minimizar, ya que si no se diseña de manera adecuada, es posible que el resultado sea que efectivamente se han encontrado los parámetros que minimizan dicha expresión, sin embargo, ello no implicaría que el sistema tenga el desempeño deseado si la expresión elegida no es la adecuada.Sin olvidar que ya el hecho de minimizar una expresión tiene sus problemas naturales como el de mínimos locales. Además se debe considerar que

estos métodos requieren de una cantidad de cálculos considerable.

Métodos Adaptables

En la sección anterior se presentaron una serie de métodos para sintonizar el controlador PID. Todos los métodos anteriores de forma tácita utilizan el hecho de que la planta o su modelo, bajo ninguna circunstancia, cambiarán con el tiempo, lo cual en la realidad se sabe que no es verdad. Tan sólo es simple hecho de que todo los elementos constitutivos de un sistema tienen un desgaste y tiempo de vida hacen que los parámetros de la planta cambien y por lo tanto la respuesta que pueden tener en un determinado momento a una cierta entrada no sería la misma que presentaran a la misma entrada en otro momento.Bajo estas condiciones los métodos presentados con anterioridad serían válidos mientras el proceso o la planta no cambien, esto es un problema más que se tiene que resolver.

Para entender mejor la situación Astrom (Aström and Hägglund, 1995) pone de manifiesto la situación, y esto se presenta en la Figura 2.4

En la Figura 2.4 se muestra que cuando se tiene una planta con dinámica variables es conveniente utilizar un controlador con parámetros variables, y si este es el caso, se tienen dos posibilidades, la primera es que dichas variaciones de alguna manera se puede predecir, en tal situación lo más conveniente es utilizar un método de asignación de ganancias, pero si no se pueden predecir las variaciones, entonces, es más conveniente utilizar un controlador adaptable. Así pues si se piensa en el peor de los casos, se tendría que la planta tiene una dinámica variable y que dichas variaciones no se pueden predecir.

Antes de mencionar algunos trabajos que se han realizado con relación al control adaptable, y en particular a la sintonización del PID para realizar control adaptable es conveniente que se presente un marco sobre el cual se puedan referenciar los conceptos.

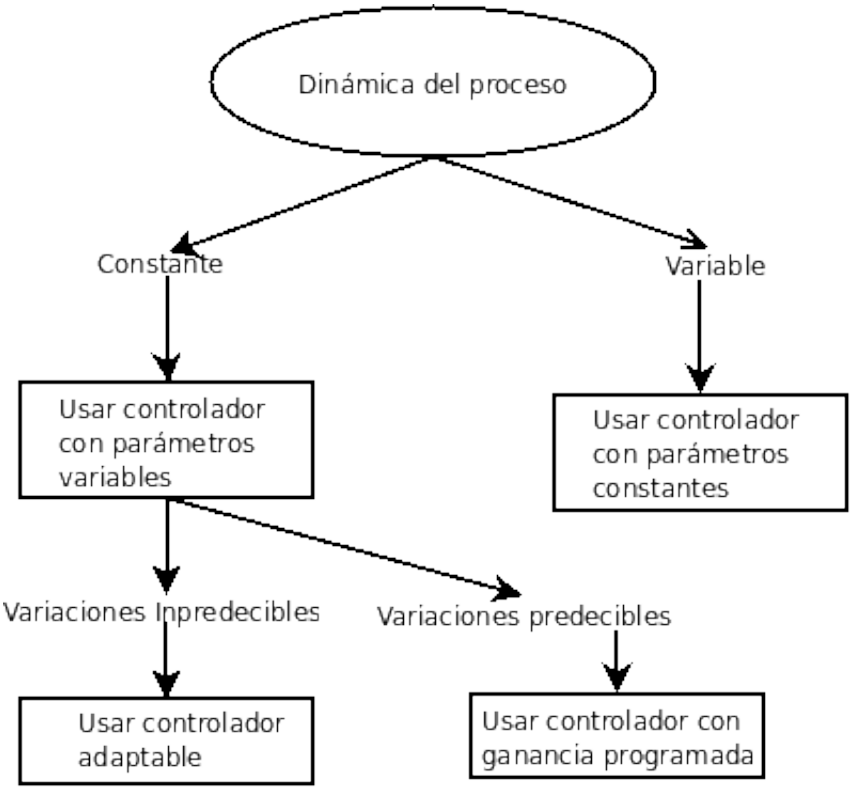

Figura 2.4. Técnicas Adaptables: Cuando utilizarlas

Lo primero es que se debe delimitar la definición de sistema adaptable al campo del control, esta delimitación se señala de la siguiente forma en Bobal(Bobál et al., 2005), donde dice que un sistema de control adaptable es un sistema que tiene la capacidad de adaptar los parámetros o la estructura de una parte del sistema de control, a saber, el controlador, esto cada que existan cambios en los parámetros o la estructura de la planta, dichos cambios se deben dar de tal forma que el sistema en conjunto mantenga un comportamiento óptimo de acuerdo a un criterio seleccionado. En la misma referencia se señala que la adaptación a los cambios que se puedan dar en la planta puede ocurrir principalmente de tres maneras

- Realizando un cambio adecuado en los parámetros del controlador
- Alterando la estructura del controlador
- Generando una señal auxiliar de entrada

Es importante señalar que la retroalimentación de alguna manera compensa los cambios que se puedan presentar, sin embargo por si sola no se considera como un sistema de control adaptable, ya que la manera en la cual procesa el error es fija, es decir, no se adapta a los cambios de manera dinámica.

Una vez que se ha establecido el concepto de sistema de control adaptable, es conveniente hacer una clasificación de los sistemas de control adaptable, de acuerdo a Bobal (Bobál et al., 2005) se pueden clasificar los sistemas de control adaptable como se muestra en la Figura 2.5

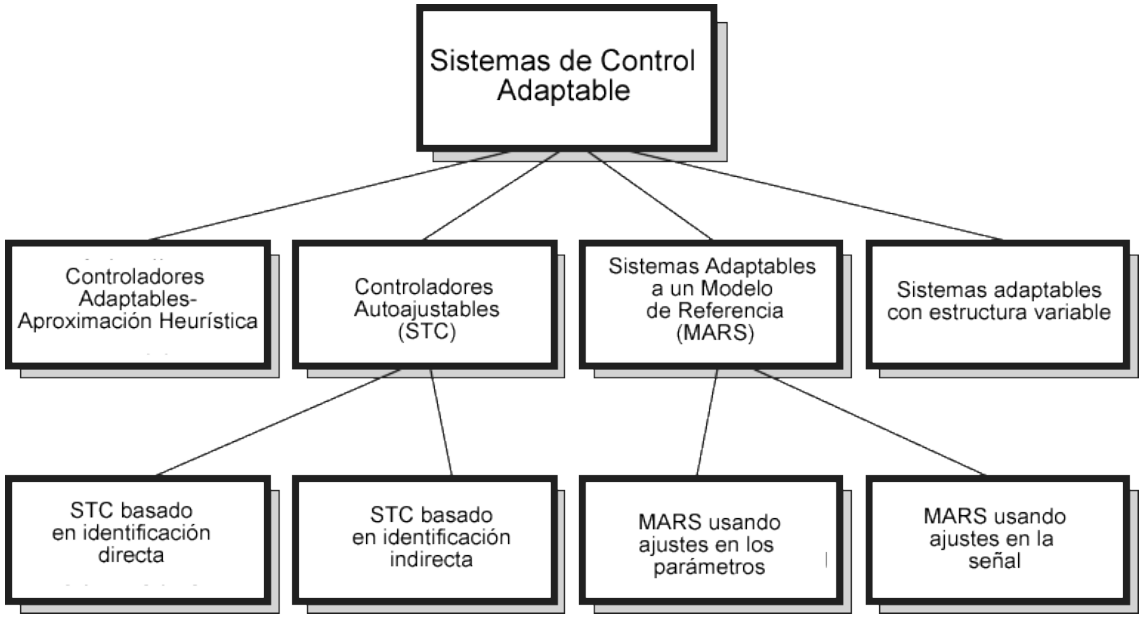

Figura 2.5. Técnicas Adaptables:Clasificación

En la figura se presenta una clasificación de los sistemas de control adaptable, en la cual se presenta como una alternativa los métodos basados en heurística. En este trabajo se utilizará esa aproximación si se desea profundizar en los detalles de los otros métodos se recomienda al lector que utilice la referencia Bobal (Bobál et al., 2005)

Por su parte los métodos basados en heurística según Bobal (Bobál et al., 2005) proveen adaptabilidad de manera directa, ya sea evaluando la salida del sistema de control o el error, o algún otro criterio que indique la calidad del sistema de control. Por lo

regular cuando se utilizan estos métodos no se requiere el modelo de la planta, de hecho en algunos casos no es necesario tomar en cuenta el error o incluir una señal de a la entrada del sistema.A continuación se presenta un esquema en la Figura 2.6 propuesto por la referencia que se ha venido manejando.

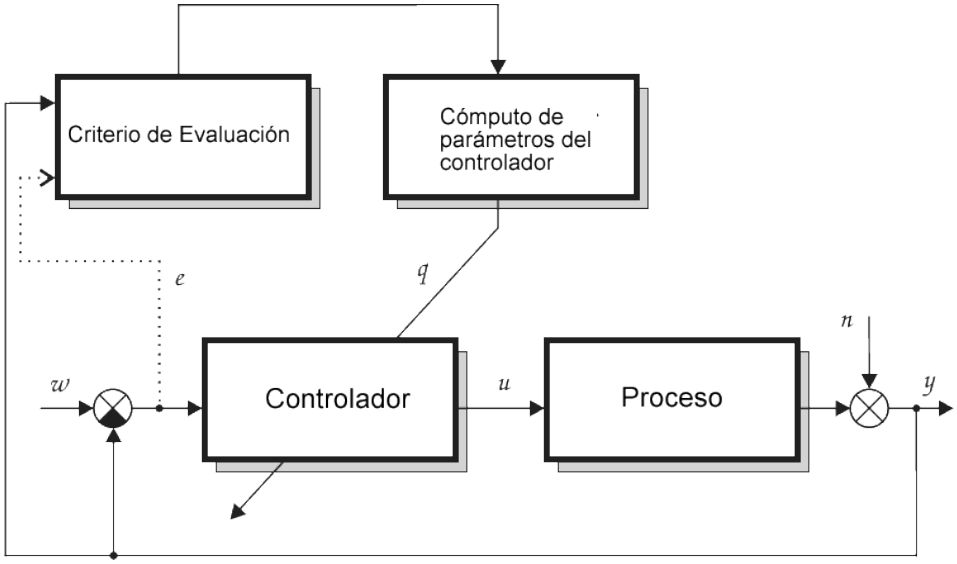

Figura 2.6. Técnicas Adaptables: Aproximación Heurística

De la Figura 2.6 es evidente que no se cuenta con un mecanismo de identificación de la planta. Es importante señalar que cuando se utilizan estos métodos lo que se trata de hacer es optimizar un criterio de desempeño del sistema, esto permite que el sistema sea aplicable a sistemas prácticos y al mismo tiempo le da robustez, sin embargo, la cantidad de cálculos necesarios es grande.

Es importante señalar que el presente trabajo no constituye un sistema de control adaptable per se, si no más bien es una parte constituvia de un sistema de control adaptable, es decir, que puede utilizarse en sistemas de control adaptable pero que por si mismo no tiene la capacidad de ser adaptable.

2.2. Estado del arte

Conforme avanza la humanidad la ciencia y la tecnología van creciendo y evolucionando también. Hoy en día, el surgimiento de problemas otrora inexistentes, requiere de la generación de nuevos métodos de solución, cuya complejidad impide la aplicación de los métodos ya conocidos y utilizados. Estos métodos son producto del refinamiento de los que hasta hoy se vienen utilizando, o en varios casos son producto de la innovación. Esto se debe a que se piensa la solución de los problemas desde un punto de vista distinto al que se venía haciendo, trayendo como resultado un cambio de paradigma. También es posible y común que se comience a desarrollar una teoría que de momento no parece de utilidad práctica, sin embargo, algunas veces resulta que la teoría que se había desarrollado con fines de ciencia básica, termina siendo una herramienta para la solución de problemas prácticos.

A continuación se presentarán una serie de trabajos, en los cuales se han utilizado técnicas heurísticas en general y algoritmos genéticos en particular para diversos propósitos,concluyendo con la utilización de estos para la sintonización de controladores PID.

Los algoritmos genéticos surgieron gracias a Holland, el cual plasmó sus ideas y primer algoritmo genético en su libro *Adaptation in natural and artificial systems* (S.N.Sivanandam and S.N.Deepa, 2008). Esta es una de las técnicas que surgieron como una idea innovadora que en un principio resultó sólo teórica, sin embargo, con el paso del tiempo la tecnología mejoró y la implementación de esta técnica es ahora más factible que en el pasado.

A continuación se mencionan algunos artículos que se basan de alguna manera en algoritmos genéticos.

En el trabajo de Jamali(Jamali et al., 2009) se estudia una forma para obtener un desempeño más robusto y eficiente de algoritmos genéticos para el diseño de contro-

ladores lineales de retroalimentación.

Utilizar algoritmos genéticos en ambientes muy cambiantes es una opción que se considera de manera seria. En (Juang et al., 2008) se propone un esquema de control para guiar un avión en una turbulencia. Dicho sistema de control es un combinación de redes neuronales y algoritmos genéticos.

También podemos encontrar aplicaciones de algoritmos genéticos en disciplinas como la Mecatrónica, tal es el caso del trabajo que se encuentra en (Affi et al., 2007). En este trabajo se diseña un mecanismo de cuatro barras con la ayuda de un algoritmo genético multiobjetivo.

Los algoritmos genéticos tienen la característica de poder ser combinados con otras técnicas para alcanzar resultados más eficientes, tal es el caso del trabajo presentado por Kim(Kim et al., 2007), en el cual se combina un algoritmo genético con otra técnica para resolver un problema de optimización global. Los resultados obtenidos son muy buenos y se consideran más eficientes que los obtenidos usando cualesquiera de las dos técnicas por separado.

Existen trabajos donde se compara a los algoritmos genéticos con otras técnicas.Xu(Xu et al., 2009) propone uno de ellos, en el cual se compara a los algoritmos genéticos con otra técnica. En este artículo se señala que bajo ciertas condiciones el algoritmo genético se desempeña de mejor manera que la otra técnica, sin embargo se señala el problema del esfuerzo computacional requerido por el algoritmo genético.

El campo de aplicación de los algoritmos genéticos como un medio para la sintonización de controladores PID es basto. Por ejemplo, el trabajo realizado por Chen(Chen et al., 2009) utiliza un algoritmo genético para la sintonización de un PID, que de cierta forma administra un sistema de transmisión de paquetes a través del protocolo TCP/IP.

De la misma manera, en el campo de control los algoritmos genéticos han sido muy estudiados. Oh(Oh et al., 2009) presenta un trabajo donde se diseña un controlador en cascada basado en lógica difusa, lo interesante de ello es que, los parámetros del controlador son diseñados por un algoritmo genético, este trabajo se aplica a un conocido problema de control, el péndulo invertido.

En la misma tónica del trabajo anterior se encuentra el realizado por Akın(Akın et al., 2003). En dicho trabajo se presenta un controlador basado en lógica difusa que se utiliza para controlar un motor, en el que los parámetros del controlador son diseñados mediante un algoritmo genético.

En la actualidad existen muchos tipos de controladores, sin embargo, en las industrias se sigue utilizando el controlador de tipo PID, esto se debe a diversas razones como las que se mencionan en Willjuice (Iruthayarajan and Baskar, 2008). Uno de los aspectos importantes y de mayor dificultad de los controladores de tipo PID es la sintonización de los mismos. Existen muchas técnicas para sintonizar este tipo de controladores, desde las convencionales hasta las basadas en técnicas más recientes como las mencionadas por Willjuice. En ese mismo trabajo se utilizan técnicas tipo evolutivas para la sintonización de un controlador de tipo PID. Cabe destacar que los algoritmos genéticos forman parte de esas técnicas, con lo que en base a simulaciones se muestra que éstas presentan mejores resultados en comparación con otros trabajos que se han reportado.

En otros trabajos como el de Altınten(Altınten et al., 2008) se han utilizado algoritmos genéticos para el diseño de controladores tipo PID de forma satisfactoria. Este trabajo en particular es destacable, ya que no sólo se llevó a cabo una simulación, sino que se realizó de manera experimental, arrojando resultados satisfactorios, sin embargo, resulta importante mencionar que el algoritmo genético se implementó en una computadora de escritorio.

En (Jinhua et al., 2009) articulo que se presenta propone un método de sintonización de un controlador de tipo PID, el cual tiene características muy interesantes como que se autoorganiza, además presenta un buen desempeño en lo que a la búsqueda global se refiere. Aunado a estas características se tiene el hecho de que presenta una velocidad aceptable de convergencia. En dicho trabajo se han tenido aportes importantes como la introducción de operadores que hacen que el algoritmo tenga un desempeño aceptable. Principalmente se cuenta con un operador de selección dominante, el cual hace que los individuos más aptos ejerzan mayor acción en el algoritmo. Además del operador anteriormente mencionado, cuentan con el operador mutación. Cabe destacar que, este operador no es el convencional, sino que lo llaman operador de mutación cíclica, el cual permite que la probabilidad de que ocurra una mutación varíe de manera periódica de acuerdo con la generación. Es importante mencionar, que los resultados presentados en dicho trabajo provienen de simulaciones.

Otro trabajo relacionado es el expuesto por Chang(Chang, 2007). En dicho trabajo se diseñan las ganancias de un controlador de tipo PID, con la novedad de que es para un proceso que trabaja con más de una variable. Este trabajo presenta también un aporte interesante al algoritmo genético, dicho aporte consiste en la modificación de uno de los operadores del algoritmo genético. En concreto, se modificó el operador de recombinación para que fuera más versátil.

Los algoritmos genéticos son un técnica heurística que como se da cuenta en (Valarmathi et al., 2009), se pueden utilizar para la identificación de los parámetros de la planta, así como para la sintonización del controlador. En este trabajo en particular los algoritmos genéticos son utilizados en forma secuencial, es decir, primero se aplican para llevar a cabo la identificación de los parámetros de la planta y posteriormente su utilizan para la sintonización del controlador PID. Los algoritmos genéticos se pueden utilizar como cadenas de bits o en números reales. En el trabajo presentado por Valarmathi (Valarmathi et al., 2009), utilizan una codificación del algoritmo genético basada

en números reales. Dos aspectos resultan de importancia en este trabajo, el primero es que el tiempo que el algoritmo genético se tarda en realizar cada tarea, se mide en segundos, es decir, no sería útil en procesos donde se requiera una adaptabilidad en un tiempo menor. El segundo se se relaciona con la implementación, la cual se llevó a cabo en una computadora de escritorio, esto facilitó la codificación con números reales y al mismo tiempo contribuyó a la velocidad con la cual se realizó el cómputo necesario para que los algoritmos genéticos actuaran. En conclusión, a pesar de que los resultados en este caso fueron satisfactorios el hecho de utilizar una computadora hace que la velocidad de cómputo impida que se pudiese utilizar en procesos más rápidos.

Cuando un tema es recurrente, encuentran trabajos como el citado en Alfaro-Cid(Alfaro-Cid et al., 2009). Los algoritmos genéticos son una técnica, la cual se describe mediante un algoritmo muy simple, sin embargo, existen muchos detalles en dicho algoritmo que es muy difícil especificar una manera rigurosa de implementar el algoritmo, por ejemplo, los operadores que maneja el algoritmo genético son variables, es decir, que en principio de cuentas pueden ser obviados como el de la mutación. No sólo pueden ser variables en los operadores, sino que a la hora de aplicar un algoritmo genético se deben tomar en cuenta muchos detalles del problema que se pretende que el algoritmo vaya a resolver. En el mismo trabajo se hace un estudio comparativo de algoritmos genéticos dedicados a la optimización de parámetros de controladores. Como se explica en el trabajo en cuestión, el problema de encontrar los parámetros de un algoritmo genético y los valores de ellos, depende del problema en cuestión Sin embargo, en este trabajo se mencionan cuáles fueron los operadores que hicieron que el algoritmo genético mostrara mejor desempeño para fines de optimización de parámetros.

Existen trabajos como el de Lu(Lu et al., 2007), en el cual se utiliza un algoritmo genético para la sintonización de un PID, pero que además se combinan con otras técnicas, obteniendo resultados eficientes y muy alentadores.

2.2.1. Sintonización de controladores PID utilizando Algoritmos Genéticos

En esta sección se presentan trabajos en los cuales se lleva a cabo una sintonización de controladores PID a través del uso de algoritmos genéticos.

En el trabajo que presenta (Yasue et al., 1999) se pone de manifiesto la posibilidad de utilizar los algoritmos genéticos como una herramienta par la sintonización de un controlador PID. En este trabajo se propone un esquema de control PID basado en algoritmos genéticos y leyes de control de varianza mínima generalizada. El propósito de este trabajo es utilizar este esquema de control en sistemas con parámetros desconocidos o variables. Al final de este trabajo se presentan simulaciones de control con diversas versiones de controladores PID autoajustables, y la que se propuso en ese trabajo resultó ser la de mejor desempeño.

El trabajo anterior expone un sistema donde el controlador es autoajustable, sin embargo, en la publicación de (Dionisio and Joao, 2005) se presenta mas bien un sistema de control adaptable, en el cual se utilizan de igual manera los algoritmos genéticos. En esta ocasión el sistema de control adaptable realiza una identificación de la planta mediante el uso de algoritmos genéticos y posteriormente los utiliza para la sintonización de los parámetros de un controladro PID.Este es un gran trabajo, ya que es el sistema de control adaptable completo.

Existen más trabajos como los presentados por (Ping et al., 2006),(Nazir et al., 2009),(Lin and Liu, 2010) en los que se utilizan algoritmos genéticos como la principal herramienta para llevar a cabo la sintonización del controlador PID.

A pesar de que los algoritmos genéticos son una herramienta poderosa, innovadora y atractiva para resolver problemas, en los trabajos previamente mencionados en esta sección, se analizan desde un punto de vista más teórico y muchos a nivel simulación. Algunos se implementan usando una computadora personal como plataforma de desarrollo, quizá un sistema embebido sería una opción más adecuada como plataforma de implementación.

3 METODOLOGÍA

En este capítulo se presenta el algoritmo genético capaz de sintonizar un controlador PID.En primera instancia se presenta el controlador que se utilizará, es decir, la estructura del controladro PID que se utiliza.Posteriormente se explica en que consiste un algoritmo genético, cuales son sus operadores y finalmente utilizando estos conceptos se desarrolla el algoritmo genético que para la sintonización del PID. En la parte final de este capitulo se presenta la implementación del algoritmo genético en FPGA, es decir, se presentan los detalles de la implementación.

3.1. Controlador PID

El controlador PID en concepto es muy sencillo y es la forma más utilizada de control por realimentación, esto si se toma en cuenta que el control P,PI y PD son casos especiales del control PID.Además de ser sencillo y muy utilizado no es nuevo, es decir, que lleva muchos años ya siendo utilizado, sin embargo, no se puede tratar como un concepto monolítico y simple, más aún, como este controlador presenta resultados aceptables a pesar de desconocer detalles importantes del mismo, muchas veces se pasan por alto muchos detalles finos que de ser tomados en cuenta el control pudiese ser de mayor calidad.

Entre los detalles del control PID se encuentran el diseño del mismo, ya que no existe una única forma de llevar a cabo el concepto de controlador PID.También es muy importante tomar en cuenta el método de sintonización que se utilice para la estructura seleccionada.Y finalmente y quizá mas orientado al aspecto tecnológico que se vive en estos días es el tipo de implementación que se tiene del controlador.Esto se refiere a que más allá de el diseño y técnica de sintonización se hayan decidido aún falta una parte muy importante que es la plataforma en la que se implementará el controlador.No es

noticia el decir que en la actualidad la mayoría de los controladores son implementados de manera electrónicas y más aún de manera digital, lo cual impacta en el desempeño del controlador de manera importante.

En este trabajo no se pretende profundizar en el controlador PID per se, es decir, no se estudiarán el diseño, la sintonización y la implementación de manera asidua, sino más bien se utilizan formas sencillas de cada uno de estos aspectos, a excepción de la sintonización ya que en ello se basa esta tesis, y se parte de ahí para avanzar al tema central que es el método de sintonización basado en algoritmos genéticos.

Con lo anterior dicho se deduce que el controlador utilizado en este trabajo se implementa de manera digital, lo cual se estudia más a profundidad en la siguiente sección.También se recomienda al lector que para un estudio más a minucioso del controlador PID se remita a (Aström and Hägglund, 1995),(Aström and Hägglund, 2001) y (Aström and Hägglund, 2006).

3.1.1. PID Digital

El controlador PID como concepto ha sido el mismo a través del tiempo, sin embargo en cuestiones prácticas ha estado supeditado a las cuestiones de hardware, es decir, que la implementación del mismo ha ido cambiando conforme la tecnología avanza. En un principio el controlador se implementó utilizando técnicas mecánicas, especialmente neumáticas, posteriormente se evolucionó y se migró a técnicas electrónicas pero de naturaleza analógica y actualmente la implementación que más se utiliza el la basada en electrónica digital.

El hecho de que se utilice una implementación de naturaleza digital implica que se deben tomar en cuenta diversos aspectos que más que ligados al concepto del controlador PID, se encuentran ligados a la naturaleza digital.Entre estos aspectos se encuentran el tiempo de muestreo,filtrado anti-alias,desratización y efectos de cuantización,para profundizar en estos temas se pueden utilizar referencias como (Katsuhiko, 1995),(Kannan, 2007) y (Aström and Hägglund, 2006).

Entre las versiones digitales del controlador PID se encuentran las llamada de posición y la incremental o de velocidad. A continuación se presentan cada una de ellas

Forma posicional

$$u(t) = K_p \left[e(t) + \frac{T_c}{T_i} \sum_{i=0}^{t} e(i) + \frac{T_d}{T_c} \left(e(t) - e(t-1) \right) \right]$$

Forma de velocidad

$$\Delta u(t) = u(t) - u(t-1)$$

$$u(t) = u(t-1) + \Delta u(t)$$

$$\Delta u(t) = K_p \left[(e(t) - e(t-1)) + \frac{T_c}{T_i} e(t) + \frac{T_d}{T_c} \left(e(t) - 2e(t-1) + e(t-2) \right) \right]$$

En la figura 3.7 se presenta la idea general de un sistema utilizando un controlador PID discreto en la versión digital

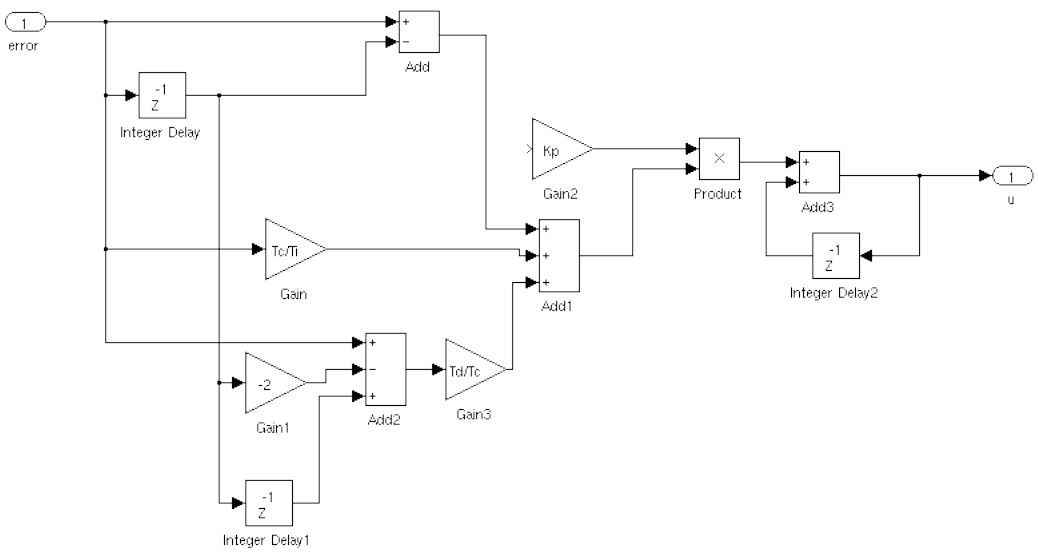

Figura 3.7. PID digital a bloques

Y en la figura 3.8 se muestran los detalles de la implementación digital del controlador PID.

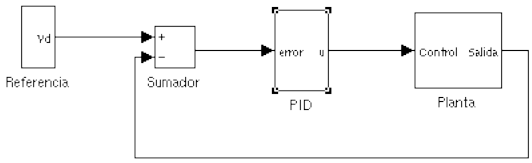

Figura 3.8. Lazo de control con controlador PID digital

3.2. Algoritmos Genéticos

La evolución como la concibió Charles Darwin si se mira con óptica diferente a la puramente biológica se puede conceptualizar como una herramienta poderosa de búsqueda y optimización, ya que la evolución de las especies biológicas ha resuelto problemas tan complejos como el caos, interacciones no lineales y temporalidad.

La computación evolutiva se inspira en el principio de selección natural para resolver problemas de búsqueda y optimización.Entre las principales técnicas de computación evolutiva se encuentran los algoritmos genéticos propuestos por Holland, la programación genética, estrategias evolutivas presentadas por Receuberg y la programación evolutiva a cargo de Forgel.Todos estos paradigmas utilizan el concepto de selección natural, sin embargo, cada uno de ellos los aplica de manera distinta.

De forma general la computación evolutiva ofrece ventajas en muchos problemas de optimización.Algunas de las ventajas mas notorias son: simplicidad de aproximación al problema, respuesta robusta ante cambio de circunstancias en el problema, paralelismo, posibilidad de realizar un híbrido entre técnicas evolutivas y técnicas convencionales o incluso con otras técnicas de la inteligencia artificial como las redes neuronales y el control difuso u otros y quizá como se señala en (S.N.Sivanandam and S.N.Deepa, 2008) la característica que distingue a este tipo de técnicas es que ayuda a resolver el problema de como resolver los problemas.

Cabe señalar que de todas estas alternativas la más representativa es la de los algoritmos genéticos.Los cuales son una herramienta que pertenece a la rama de la

inteligencia artificial, la cual ha probado ser útil en la solución de problemas donde se requiere optimizar una función.Desde el punto de vista del control se puede pensar en esta optimización como una minimización de una función, dicha función por lo regular es el error entre la referencia del sistema y la salida del mismo.

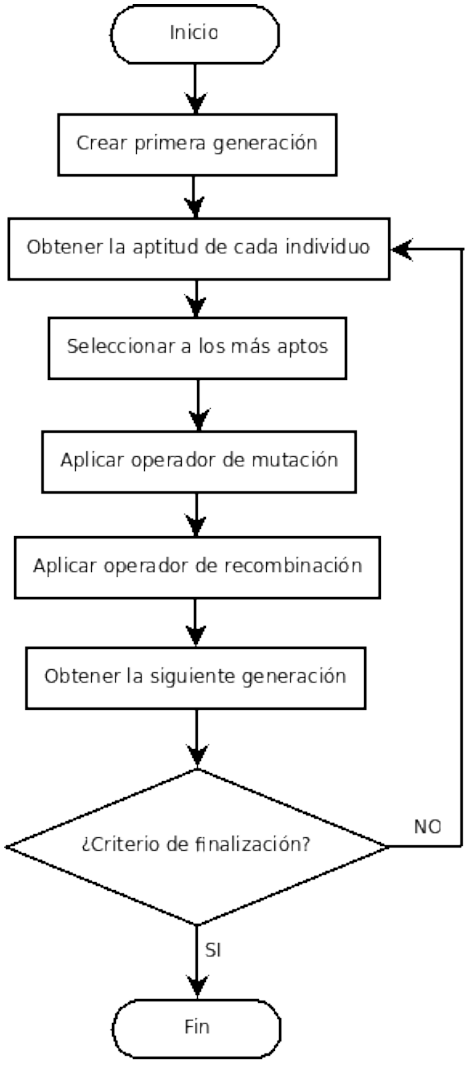

Figura 3.9. Diagrama de flujo de un algoritmo genético genérico

De manera más formal los algoritmos genéticos obedecen el siguiente algoritmo 3.9. En este diagrama de flujo se pone de manifiesto el procedimiento que se lleva a cabo para resolver un problema a través de un algoritmo genético. A continuación se detallan cada una de las etapas del algoritmo genético.

3.2.1. Codificación

Antes de comenzar a describir cada uno de los operadores de los algoritmos genéticos es pertinente mencionar que cada individuo de la población puede representarse o codificarse de dos maneras distintas, una es a través de cadenas de bits y la otra es mediante números reales. Históricamente primero se utilizaron las cadenas de bits como la codificación a elegir, sin embargo, se ha probado que la codificación a través de número reales es la más eficiente.

A pesar de que la codificación utilizando números reales es más eficiente se debe tomar en cuenta que la plataforma que se plantea para la implementación de este algoritmo es un FPGA, lo cual no permite de manera sencilla utilizar número en punto flotante, sin embargo es posible utilizar números en punto fijo que como tal no son reales pero si una aproximación a los mismos.

3.2.2. Creación de la primera generación

Este el primer paso de un algoritmo genético, el cual tiene que ver con el espacio de búsqueda de la solución. Esta definición del espacio de búsqueda debe ser aleatoria. Cada elemento de esta generación se conoce como un individuo. Cada uno de esto individuos es un candidato a la solución más apta para el problema.

Existen dos cuestiones importantes a la hora de crear la primer generación, la primera es que tan aleatoria va a ser la creación y la segunda es el tamaño de la población.

3.2.3. Función de aptitud

Para saber si un elemento de la generación actual cumple con los requerimientos para considerarse la solución buscada se debe contar con un mecanismo, el cual se conoce como función de costo o aptitud. Así pues, cada individuo se somete a la función de aptitud y con ello se logra saber si es un buen candidato para participar en la procreación de nuevas posibles soluciones o incluso se puede dar el caso, que es lo

que se busca al final, de que sea tan apto el individuo que sea la solución buscada.

La función de aptitud no es trivial, de hecho es un punto crítico para el éxito que puede alcanzar el algoritmo genético, es decir, cada problema debe definir una función de aptitud que verdaderamente ponga de manifiesto la aptitud del individuo en cuestión para el problema que se desea resolver.

3.2.4. Selección

Una vez que se han asignado aptitudes a cada uno de los individuos, se procede a la selección de los más aptos. Dicha selección se puede llevar a cabo de diversas formas una de ellas es seleccionar a la mitad de la generación que haya mostrado ser la más apta individuo a individuo, a esta forma se le conoce como elitismo. A pesar de que esto parece ser a primera vista lo más adecuado, algunas veces acarrea problemas de convergencia prematura, por lo cual se utilizan métodos alternativos como la ruleta de selección o selección aleatoria.

3.2.5. Creación de la siguiente generación y criterio de finalización

Antes de crear la nueva generación, primero se revisa si es necesaria dicha creación, es decir, se evalúa el criterio de finalización del algoritmo, en el mejor de los casos, uno de los individuos cuenta con las características pertinentes para ser la solución que se desea, con lo cual el algoritmo termina. A pesar de que es muy posible que se encuentre un individuo que cumpla con los requerimientos necesarios para ser la solución, también existe la posibilidad de que no se encuentre este individuo, así que en muchas ocasiones se plantea como criterio de finalización del algoritmo un cierto número de generaciones, así pues, si se encuentra el individuo solución o se alcanza el límite de generaciones el algoritmo ha finalizado, de otra manera se prosigue con los operadores que se describirán a continuación.

3.2.6. Recombinación

Si es necesario crear una nueva generación, se cuentan con dos operadores principales, los cuales son el de recombinación y el de mutación.

El primero se hace referencia a como a partir de soluciones aptas se pueden generar solucione más aptas, o al meno en teoría eso es lo que se busca.

Existe varias aproximaciones para este operador, una de las más utilizadas es utilizar lo que se conoces como dos padres, individuos de la generación anterior, y a partir de ellos crear dos hijos, esa creación de hijos es lo que se conoce como recombinación.

Utilizando codificación en números reales el operador de recombinación que se utiliza en esta tesis es el que se conoce como operador de recombinación aritmética que es básicamente una combinación lineal de los padres, y tiene la forma que sigue

$$h_1 = rP_1 + (1 - r)P_2$$

$$h_2 = (1 - r)P_1 + rP_2$$

Donde P_x son los padres y h_x son los hijos generados.

3.2.7. Mutación

Este operador es opcional, pero en muchos casos es muy deseable contar con el, ya que en algunos casos permite que no se presente una convergencia prematura.

3.3. Algoritmo genético para la sintonización del controlador PID digital

En las secciones previas se han expuesto el controlador PID digital y los algoritmos genéticos respectivamente,en esta sección se retoman los conceptos de esas secciones para en base a ellos poder desarrollar un algoritmo genético dedicado a la sintonización de un PID.

3.3.1. Consideraciones para el desarrollo del algoritmo genético

Debido a que el algoritmo genético se pretende utilizar para la sintonización de controladores PID, lo primero que se debe tomar en cuenta es el hecho de que el algoritmo presentado hasta el momento es genérico,es decir, sólo se ha mostrado de manera general como es que un algoritmo genético opera, sin embargo, esta forma de

operar se debe adaptar para que sintonice un controlador PID.

Ya teniendo en mente que el objetivo del algoritmo genético es sintonizar un controlador PID es conveniente que desde un principio se tome en cuenta que la plataforma elegida es digital,a saber, un FPGA.Por lo anterior, la adaptación del algoritmo genético genérico a uno capaz de sintonizar un PID se hará para un controlador digital.

A pesar de que desde un principio sería posible tomar en cuenta algunas restricciones mas impuestas por la plataforma final,como por ejemplo el tipo de aritmética que se utilizará o la cantidad de recursos(memoria) con de las cuales se dispondrá para la implementación final, esto no se realiza así en primera instancia, esto con la finalidad de aprovechar la oportunidad para realizar la comparación en desempeño del algoritmo genético implementado en software y con su contraparte en hardware.Es claro que el hecho de no hacer este tipo de consideraciones pues permitirá que la versión en software sea más flexible y alcance un desempeño que va acorde con las posibilidades que el software brinda, sin embargo de igual manera se tendrá que afrontar el problema de la migración de la versión en software a hardware, lo cual no será trivial.

3.3.2. Adaptación del algoritmo genético genérico a uno capaz de sintonizar un controlador PID

Representación y codificación de cada individuo

Debido a que sin importar la estructura del PID que se elija, incluso sin tomar en cuenta si fuese analógico o digital, lo que no varía es que se compone de tres ganancias las cuales son proporcional(K_p), derivativa (K_d) y la integral (K_i).Es por eso que se la codificación de los individuo se deben tomar en cuenta las tres. Así pues cada individuo tendrá la forma de

$$[K_p K_i K_d]$$

Ahora gracias a que la plataforma final de implementación es digital suena tentador que la codificación del individuo se lleve a cabo por cadenas de bits en vez de

que sea de forma real, esto es debido a que las operaciones con cadenas de bits son mas sencillas, sin embargo en este caso se optó por realizar una codificación real, ya que no se contará con la limitaciones fuertes para realizar operaciones aritméticas y además que los resultados en trabajos relacionados sugieres que los resultados alcanzados de esta manera son mejores.

Creación de la primer generación

Es importante señalar que la creación de la primer generación de alguna manera establece el espacio de busqueda, claro que dependiendo del algoritmo genético en cuestión y de la manera en que evolucione, las soluciones pueden, y en algunos casos es deseable, que vayan más allá de este espacio de busqueda. También se debe mencionar que por lo regular esta primer generación se lleva a cabo utilizando métodos aleatorios o pseudo-aleatorios, sin embargo si te tiene una idea aunque sea somera de por donde puede estar la solución, es conveniente que el espacio de busqueda se centre en ese lugar. Así pues en este caso se utiliza un método pseudo-aleatorio para la creación de la primer generación y además se cuenta con un mecanismo para poder delimitar en que rango se generen los individuos.

Función de aptitud

Este es un punto muy crítico en la adaptación del algoritmo genético.De manera común se prueban los algoritmos genéticos con funciones matemáticas conocidas y bien estudiadas, en esos casos la elección de la función de aptitud no sólo no es retadora sino que es directa, es decir, se utiliza la función matemática como función de aptitud o costo, sin embargo en este caso debido a que lo que se requiere es minimizar el error de la referencia con respecto a la salida pues la elección se complica bastante ya que la respuesta del sistema y por ende el error no solo depende del controlador, el cual es bien conocido, sino también de la planta en cuestión.En este caso se optó por tomar la función de aptitud como la integral del valor absoluto del error, esto es cuando se pone el controlador PID en cascada con la planta y en un esquema de retroalimentación típico.

Selección

Para el operador de selección se optó por utilizar elitismo, el cual consiste en escoger a los individuos más aptos, en este caso se selecciona a la mitad de la población más apta.Para esto se ordena los individuos según su aptitud y se procede a tomar los más aptos.

Recombinación

El operador de recombinación que se utilizó es una extrapolación del que se presentó con anterioridad en la sección donde se presentan los operadores genéticos para el algoritmo genético genérico, es decir,

$$p_1 = [P_1 I_1 D_1]$$

$$p_2 = [P_2 I_2 D_2]$$

$$h_1 = [r(P_1) + (1-r)(P_2)r(I_1) + (1-r)(I_2)r(D_1) + (1-r)(D_2)]$$

$$h_2 = [r(P_2) + (1-r)(P_1)r(I_2) + (1-r)(I_1)r(D_2) + (1-r)(D_1)]$$

Donde $0 < r < 1$ es la probabilidad de recombinación y p_x es el padre así como h_x el hijo, además x y P_x,I_x y D_x son las ganancias proporcional, integral y derivativa, respectivamente del individuo en cuestión. Aquí se puede apreciar que el operador utilizado no es más que una extrapolación, es decir, se aplica el operador conocido a cada una de las ganancias por separado.

Mutación

El operador de mutación que se utiliza es como sigue:

$$P'_k = P_k + \Delta(t, {P_k}^U - P_k)$$
$$\Delta(t, y) = yr\left(1 - \frac{t}{T}\right)^b$$

Donde ${P_k}^U$ es el individuo más apto de la generación actual y P_k es el individuo que mutará de esta generación. Además se debe tener en cuenta que t es el número

de generación en la cual se realiza la mutación mientras que T es el número total de generaciones. Por otro lado b es el parámetro de no uniformidad y finalmente r es el coeficiente de probabilidad de mutación.

Siguiente generación y criterio de finalización

El criterio de finalización que se propone es por número de generaciones,es decir, que al cabo de un cierto número de generaciones presumiblemente se tiene a la mejor solución, en este caso, las ganancias adecuadas para el controlador, y claro que mientras no se alcance este número de generaciones lo que sucede es que la nueva generación esta conformada por los padres de esta generación y los hijos de estos.

Versión en software del algoritmo genético para sintonizar controladores PID

En primera instancia se debe mencionar que con propósitos de experimentación y validación del algoritmo, en un principio se utilizó un lenguaje de scripts,Matlab, para el desarrollo de la versión en software.Esto tiene varias implicaciones.Primero como matlab estar orientado a matrices y tiene muchas funciones disponibles, las cuales en ningún momento se dudo utilizarse, el desarrollo de las versiones preliminares y primer versión fue más sencillo.Segundo, precisamente el hecho de que tenga tantas funciones disponibles permitió probar diferentes ideas de manera rápida para ver cual daba mejor resultado.Y tercero, la velocidad de ejecución es más lenta que si se hubiese optado por otro lenguaje de programación.

Sin embargo, una vez que se llegó a un algoritmo adecuado para las necesidades, dicho algoritmo se migró a Hardware para su implementación en FPGA y como en muchos casos sucede el algoritmo se modificó para que la implementación en hardware se diera de la mejor manera.Tomando en cuenta lo que se ha mencionado con anterioridad a cerca de Matlab y debido a que se realizó un comparación entre software y hardware, el algoritmo en Matlab también se migró a un lenguaje de programación un poco más eficiente, es decir C. Además de que es un poco más eficiente, ene este caso el script

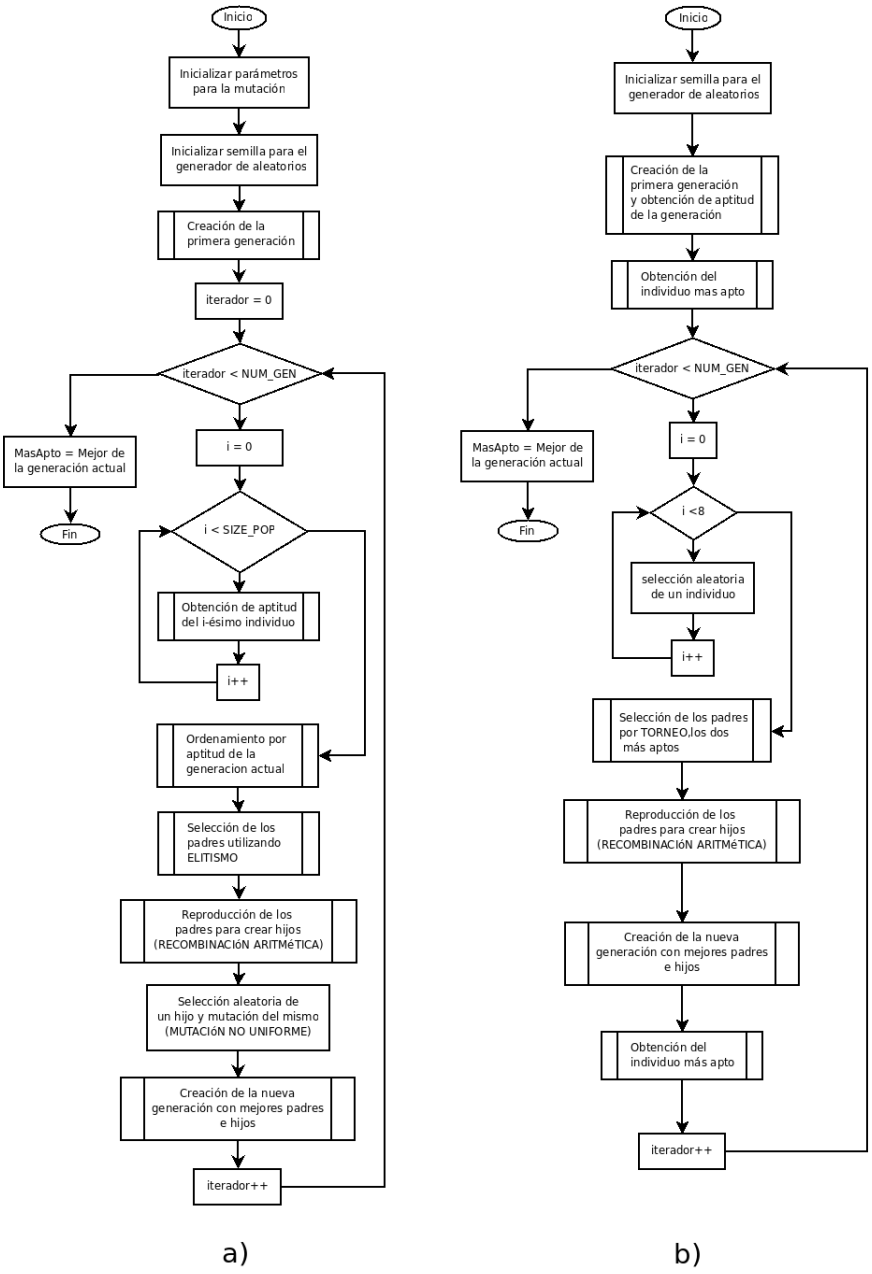

Figura 3.10. Diagrama de flujo del algoritmo genético para sintonizar PID a) en Matlab b)en C

de Matlab y el programa en C no son equivalentes, ya que el programa en C trata de asemejarse en lo más posible a la implementación que se tiene en Hardware para que la comparación sea más apropiada.En la figura 3.10 se presentan los diagramas de flujo de los algoritmos que se implementaron tanto en Matlab como en C.

3.4. Implementación del algoritmo genético en FPGA

En esta sección se presentan los detalles de la descripción en hardware del sistemas dedicado a la sintonización de un controlador PID.Se exponen los motivos por los cuales el algoritmo que se desarrolló en software no pudo ser migrado de manera transparente al hardware. Además se presentan los diagramas de cada una de las partes que conforman la implementación final.

3.4.1. Cambio de Software a Hardware

Es importante señalar que en la versión desarrollada en software los objetivos acotaron de alguna manera el alcance y desempeño de la misma.En principio se eligió realizar una versión en software por la relativa facilidad con la que se pueden realizar cambios, lo cual permitió experimentar con una gran variedad de operadores genéticos hasta que se encontraron los más adecuados para este caso, además se utilizaron las herramientas que se disponen en software a discreción.Es importante destacar que la versión en software es totalmente funcional, sin embargo, nunca se optimizó ya que ese no era el propósito de este trabajo.Debido a lo anteriormente explicado no fue posible migrar de manera transparente el algoritmo a la plataforma en hardware.En cada una de las siguientes secciones se describe la implementación en hardware del sintonizador, y al mismo tiempo se menciona que diferencia hay con respecto a la versión en software.

3.4.2. Descripción general de la implementación en hardware

El sistema de diseño como el que se muestra en la Figura 3.11 .En dicha figura se aprecia como se ha segmentado en módulos el algoritmo genético completo.Cada uno de los módulos desempeña una tarea bien definida y distinta a la de los demás módulos. A pesar de que se ha modularizado el algoritmo debido a su naturaleza y a que los recursos a pesar de ser contundentes pues no son infinitos, no se puede llevar a cabo el algoritmo totalmente de manera paralela, sin embargo, más adelante se explicará de que manera se explota esta modularización para contribuir a la eficiencia del algoritmo.

También se debe notar que se requiere de un secuenciador,es decir, existe una máquina de estados general(secuenciador), la cual se encarga de sincronizar todos y

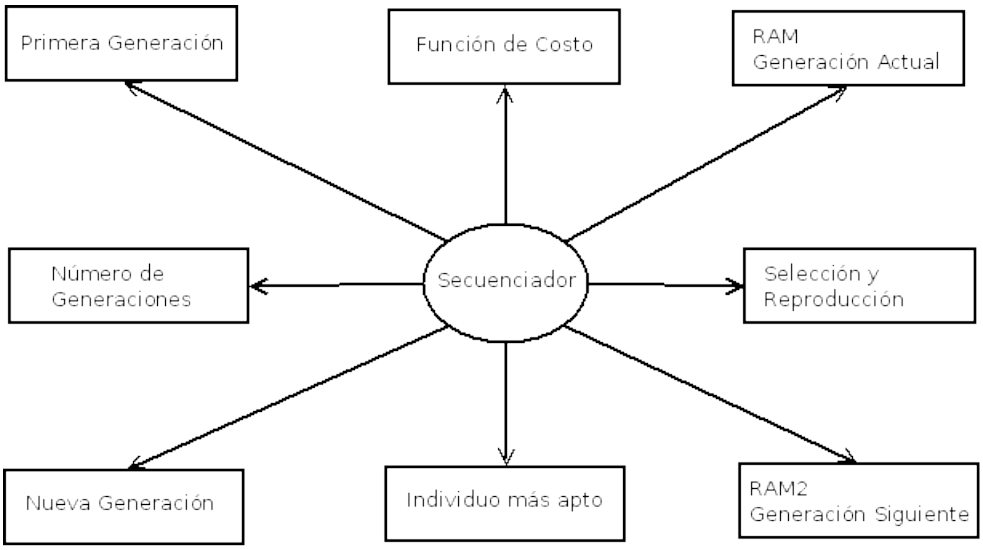

Figura 3.11. Módulos del sistema

cada uno de los módulos para que el sistema funcione de manera adecuada.

3.4.3. Sistema: Los módulos

Después de analizar el algoritmo desarrollado en software y migrarlo a hardware, se decidió dividir el algoritmo en módulos.Los módulos son: GEN1, Fnc_Apt, RAM, Sel_Rep, RAM2, fsm_test_GA, pgr_count, Fittest, NEW_GEN.En conjunto constituyen el sintonizador de controlador PID.

Se puede apreciar que se compone de diez módulos principalmente, esto claro dejando de lado la lógica de apoyo como lo son los circuitos conmutadores.Cada uno de los módulos realiza una tarea específica, salvo el módulo Sel_Rep, el cual como su nombre lo sugiere se encarga de la etapa de selección y reproducción, esto debido a que la forma en como se planteó el sistema es más eficiente si se hace de esta manera, en secciones posteriores se expondrán los detalles de éste y los demás módulos.

Es de destacar que el módulo fsm_test_GA es la máquina de estados que sincroniza todos los módulos, es decir, que se encarga gestionar el flujo de la información de un módulo a otro, con todo lo que ello implica.En las secciones siguientes se describen cada uno de estos diez módulos

3.4.4. Módulo 1: Creación de la primer generación

La finalidad de éste módulo como su nombre lo indica es crear la primer generación, lo cual como ya se ha mencionado con anterioridad define el inicio del espacio de búsqueda del algoritmo genético.

En este caso en particular el espacio de búsqueda se compone de individuos que constan de tres ganancias cada uno por lo cual este módulo es capaz de generar individuos con esa característica.En la Figura 3.12 se muestra este módulo.

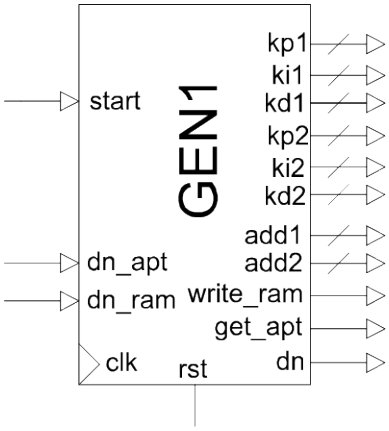

Figura 3.12. Módulo 1: Primera generación

Se puede apreciar cuenta con entradas y salidas que permiten que el secuenciador lo controle, además claro de contar las tres ganancias que son las partes conformantes da cada individuo. Adicional mente a esto se cuenta con una salida denominada *addx*, donde x, va de 1 a 2.Por este puerto se genera una dirección que se utiliza como dirección de una memoria RAM para que se almacene este individuo.

Tanto las ganancias como las direcciones vienen en par,es decir, que estos puertos se encuentran por duplicado.Lo anterior es con la finalidad de dar un poco de paralelismo a la implementación. En otras palabras, el módulo es capaz de crear dos individuos, con sus respectivas direcciones para almacenamiento, a la vez, con esto se

aumenta la velocidad de la creación de la primer generación, lo cual permite que posteriormente se pueda tener mas paralelismo en el algoritmo.

Los algoritmos genéticos son una técnica en la cual es de vital importancia la generación de números aleatorios, esto parece ser trivial, sin embargo no es así.Es muy complicado generar números aleatorios de una manera sistemática, de hecho es impensable.La forma de obtener números aleatorios es muestrear un fenómeno que tenga una cierta aleatoriedad intrínseca y tomando en cuenta que lo que desarrolla es ser humano por lo regular más bien es determinístico pues no es sencillo.Esto se comenta ya que este módulo requiere de un generador de números aleatorios para de alguna manera crear los individuos de la primer generación. En la Figura 3.13 se muestra lo que se conoce como un LFSR, por sus siglas en inglés Linear Feedback Shift Register.

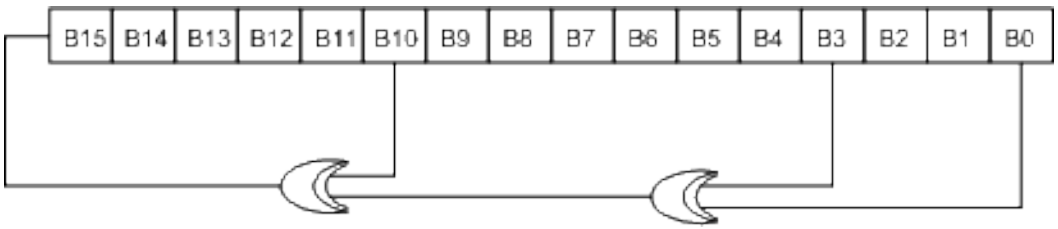

Figura 3.13. Linear Feedback Shift Register

El LFSR permite generar una secuencia de números pseudo aleatorios,como ya se mencionó con anterioridad no son aleatorios completamente ya que el comportamiento es predecible y deterministico, en este sentido se debe decir que a pesar de que el la secuencia es predecible si se elige el primer número de este de manera adecuada los número generados no se repetirán sino hasta después de cierto periodo,es decir, que no habrá una cierta cantidad de números antes de que se obtenga el mismo.

Este módulo esta constituido por dentro de tal manera que de principalmente de LFSRs y una máquina de estados controla su operación, así como de lógica de apoyo.Es conveniente notar que la semilla, o el número donde inician los LFSR que finalmente generan las ganancias provienen de otro LFSR, lo cual pudiese pensarse que no es de

gran ayuda, ya que otra vez ya se conoce la secuencia, sin embargo, los LFSRs que fungen como generadores de semilla toman su semilla de un contador de carrera libre, dicho contador comienza a contar desde el momento en el que el sistema se restablece y no para de hacerlo.Con esto se pretende que la semilla del primer generador de semillas sea de alguna manera no predecible,esto se debe a que no se sabe en que irá la cuenta cuando se genere la semilla, esto depende de cuando se de el comando de crear la primer generación y eso no se sabe a priori.

Este módulo sólo se utiliza al principio del algoritmo genético, ya que su función es crear la primer generación, y como su nombre lo indica es la primera, de ahí en adelante no se utiliza más sino hasta que se inicie el algoritmo genético una vez más.

3.4.5. Módulo 2: Función de Aptitud

Este módulo es el que se encarga de obtener la aptitud de cada individuo, ya sea que este pertenezca a la primera generación o las subsecuentes.

Para que se cumpla el objetivo de este módulo es necesario conocer el error que existe entre la referencia y la salida del sistema en lazo cerrado.Cuándo se hace referencia al sistema se debe tomar en cuenta que éste incluye a el controlador PID y la planta en cascada, esto claro en lazo cerrado.Como ya se ha comentado con anterioridad el controlador PID se conoce de antemano así que es sencillo conocer su salida a una entrada dada, sin embargo, la planta no es conocida, lo cual constituye un obstáculo en el diseño, ya que sin conocer la planta no es posible realizar la sintonización.Así pues es necesario es utilizar un método para que realice la identificación de la planta para tener esta información y con ella poder conocer la salida del sistema y que el algoritmo genético tenga la información necesaria para poder trabajar.

En este trabajo la identificación de la planta se llevó a cabo fuera de línea utilizando el método de mínimos cuadrados.De esta manera se consigue tener un modelo

matemático de la planta que permite completar el sistema y por ende es posible conocer la respuesta del sistema y el error que es lo que finalmente se busca.

La aptitud de cada individuo se obtiene de manera sencilla, simplemente es la suma del valor absoluto del error que se obtiene el sistema a una entrada fija y con un cierto número de iteraciones.

Entrando un poco más a detalle en el módulo, se puede apreciar que se compone de siete módulos.Estos módulos básicamente están interconectados de tal manera que el resultado sea un sistema en lazo cerrado, y además que el error se vaya acumulando, para lo cual se tiene un módulo que se encarga de realizar esta tarea. Es claro que de igual manera se necesita un contador que sea tenga la cuenta de cuantas iteraciones. Como este sistema se compone de una cantidad considerable de módulos se necesita una maquina de estados que los sincronice todos para un funcionamiento correcto, es máquina es básicamente otro secuenciador.

Algunos de estos módulos desempeñan una tarea muy sencilla por lo cual no se explicarán, sin embargo hay algunos otros que merecen especial atención.En primera instancia se analizará el módulo de la planta.

En la Figura 3.14 se muestra al módulo de la planta. Debido a que se tomó de una función de transferencia, su implementación pudo ser menos costosa en cuanto a hardware se refiere, sin embargo, en este caso se decidió por la implementación como se muestra.Esta implementación permite que para cada entrada exista una salida en tan solo un ciclo de reloj, si bien es cierto que esto es costoso en recursos también es cierto que en cuestión de velocidad se mejora bastante.

Por otro lado en la Figura 3.15 se presenta el módulo de un controlador PID, el cual es, como su nombre lo indica, un controlador PID, que es totalmente funcional.A pesar de que este módulo es un PID funcional, al igual que el módulo de la planta se

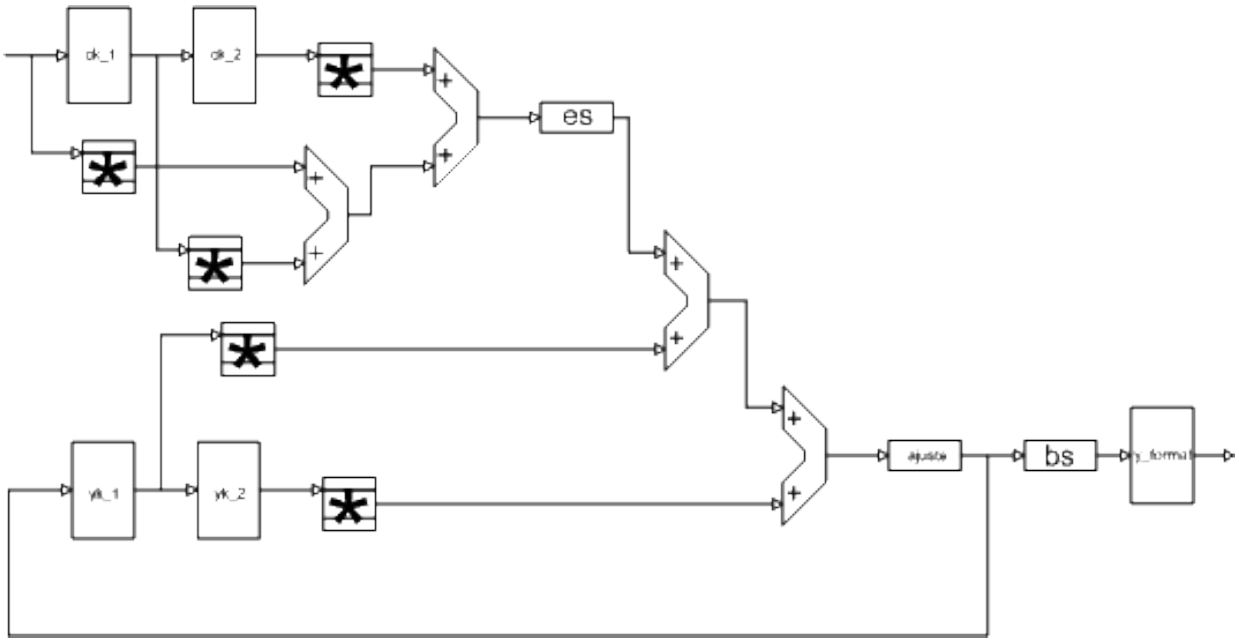

Figura 3.14. Planta del al función de aptitud

construyó de manera que para cada entrada se tenga una salida en tan sólo un ciclo de reloj.Las ventajas y desventajas de la forma de implementar esté módulo, son las mismas que tiene el de la planta.

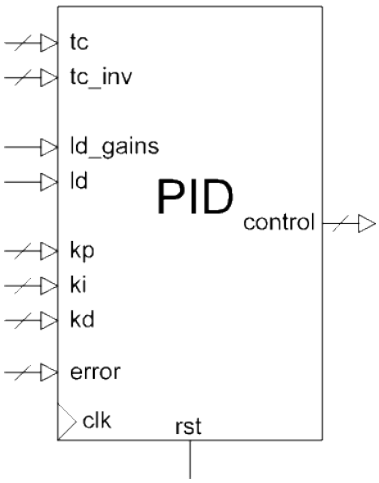

Figura 3.15. PID de la función de aptitud

Como se aprecia en la Figura 3.16 el módulo PID fue implementado de manera paralela, también se debe tomar en cuenta que los parametros de tiempo de muestreo se

tomaron de acuerdo al tiempo de muestreo de la planta en cuestión.También se puede apreciar que este módulo es la versión en hardware del PID en modo velocidad presentado con anterioridad.

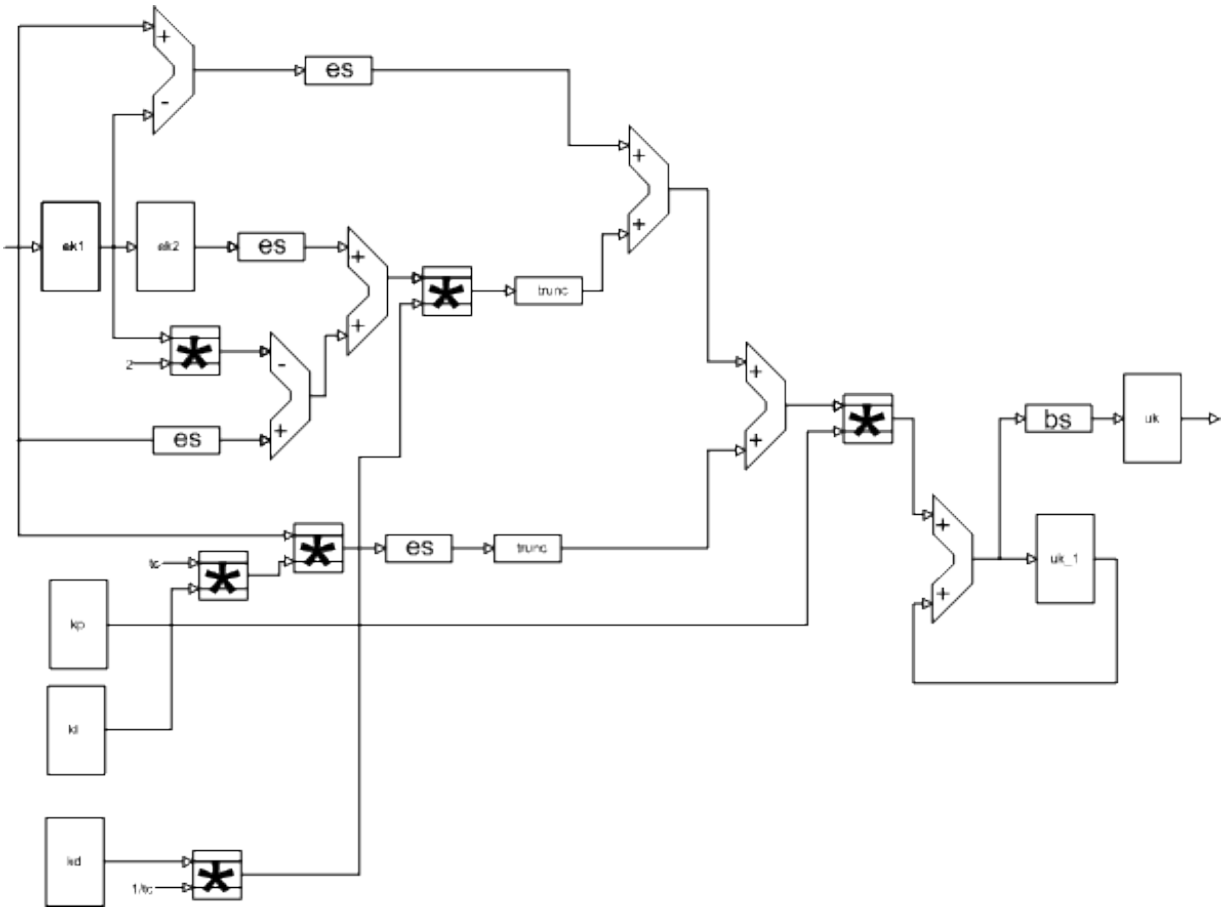

Figura 3.16. PID de la función de aptitud:arquitectura

Es conveniente señalar que la máquina de estados que funge como secuenciador asume que cada nuevo ciclo de reloj ha pasado un tiempo de muestreo, es decir, que no existe una base de tiempo como sería el caso si este mismo módulo se aplicar a una lazo de instrumentación real, esto es porque realmente se está emulando el lazo con el fin de que se realice lo más rápido posible.

Finalmente se debe notar que en el diagrama de mayor jerarquía se cuenta con dos módulos de que obtiene la aptitud de un individuo.Ambos módulos son idénticos y el

hecho de que existan dos módulos con la misma funcionalidad permite que el algoritmo se lleve a cabo en un poco más en paralelo.Es cierto que si este módulo se multiplica por el número total de individuos por generación, entonces se reduciría enormemente el tiempo en el cual se ejecuta el algoritmo genético, sin embargo, esto sería muy costoso en cuanto a recursos se refiere, por esta razón sólo de optó por utilizar un par de estos módulos.

3.4.6. Módulo 3: Generación Actual

Esté módulo se encarga de almacenar la generación presente o actual.Se implementa esencialmente como una serie de memorias RAM.

En la Figura 3.17 se muestra la implementación de este módulo.Consta básicamente de cuatro memorias RAM de dos puertos de escritura y lectura cada una, esto con la finalidad que continuar con el paralelismo que se había ganado al tener dos módulos de obtención de aptitud.

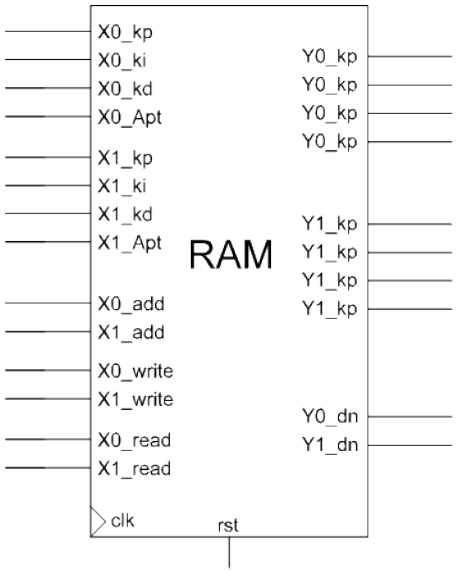

Figura 3.17. Módulo 3: Generación Actual

Las memorias RAM de este módulo almacenan una parte de cada individuo, tres de ellas almacenan al un individuo, es decir, las ganancias proporcional,integral y

derivativa.Y la memoria RAM restante almacena la aptitud de este individuo.

Además este módulo cuenta con un par de pequeñas máquinas de estados, las cuales tienen por objeto encargarse de las señales de control que permiten escribir y leer de las memorias RAM.Cómo las memorias RAM son de dos puertos se requieren dos máquinas de estados,una para cada puerto.

3.4.7. Módulo 4: Selección y Reproducción

Debido a la manera en que se decidió llevar a cabo estas dos operaciones se optó por poner ambas en un sólo módulo.De no haber unido estos dos operadores genéticos para que se lleven a cabo al mismo tiempo hubiese sido necesario utilizar una memoria RAM auxiliar que almacenara a los individuos seleccionados para ser reproducidos, lo cual consumiría más recursos del FPGA.

En la Figura 3.18 se presenta el módulo de selección y reproducción.

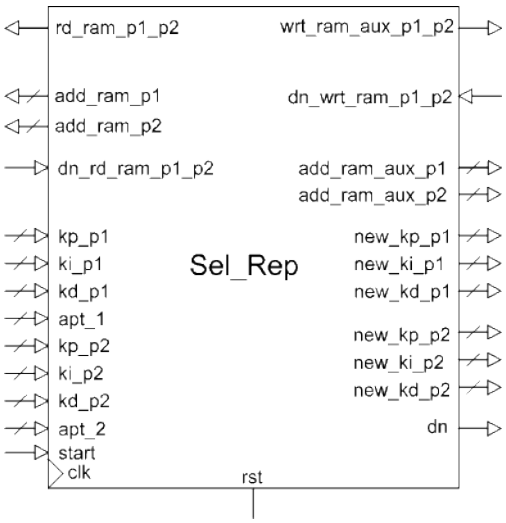

Figura 3.18. Módulo 4: Selección y Reproducción

Las entradas de éste módulo tienen que ver principalmente con la interacción con el secuenciador y con el módulo de creación de la primer generación,ya que es de

este módulo donde se seleccionan los individuos para ser reproducidos.Por otro lado las salidas se relacionan más bien con el operador genético de reproducción que tiene que ver con la generación de los nuevos individuos o hijos.

En la Figura 3.19 se presenta la arquitectura de este módulo.En dicha arquitectura destacan los módulos de: selección aleatoria,el de torneo, reproducción aritmética, generador de direcciones y el secuenciador de este módulo.

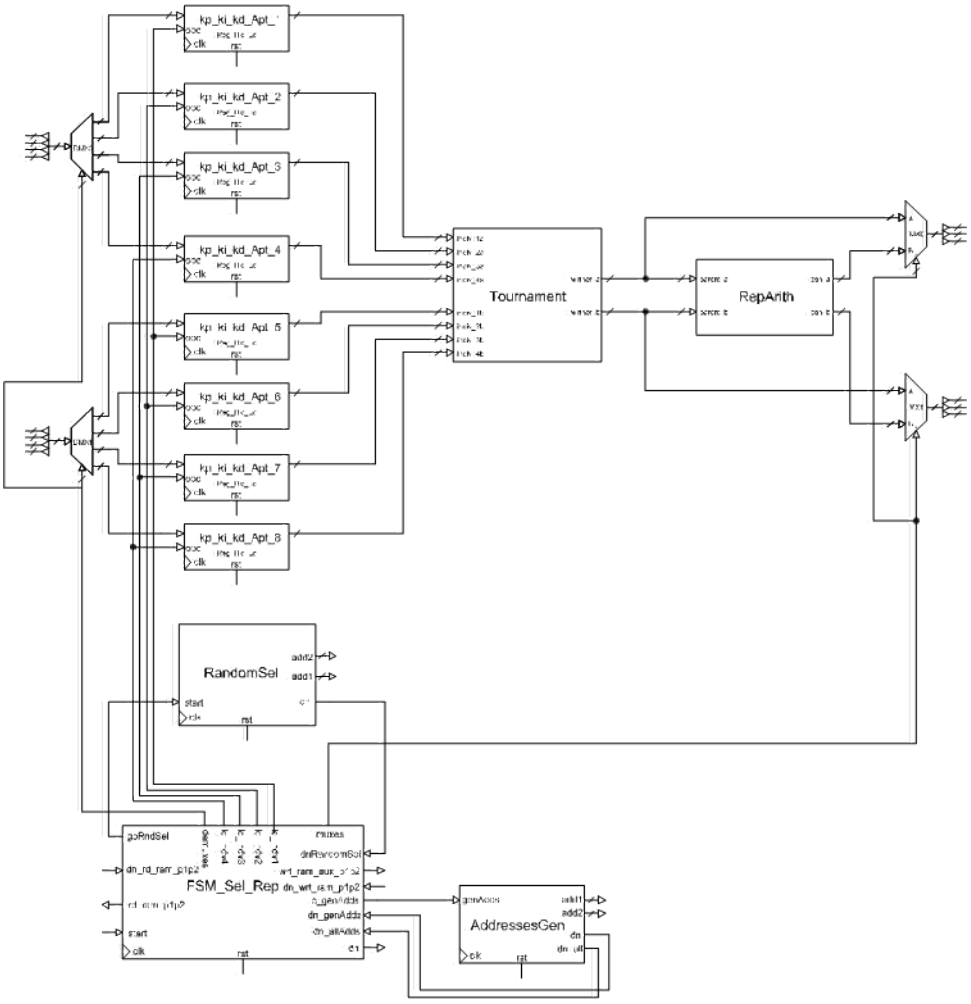

Figura 3.19. Arquitectura del módulo de selección y reproducción

El módulo de selección aleatoria se compone de dos módulos generadores de semillas, como los que se utilizan en el módulo de de creación de primer generación

además de dos LFSRs, los cuales a diferencia de los que se encuentran en el módulo de creación de la primer generación son de menor tamaño debido a que los números pseudo-aleatorios que deben generar deben de encontrarse en el rango de la memoria donde se almacena la primera generación, a saber de 0 a 256.Además de esto cuenta con una máquina de estados que se encarga de dar la orden de generación de semilla y cargarla para la generación de los números pseudo-aleatorios que finalmente serán un par de direcciones las cuales contienen un par de individuos que son sometidos a un torneo para ver si son reproducidos.

Por otra parte el módulo de torneo como su nombre lo sugiere es un torneo en el que compiten ocho individuos y los dos que resultan ser más aptos son reproducidos.La arquitectura de este módulo consta principalmente de una serie de comparadores y conmutadores.

Los ocho individuos se dividen en dos grupos de cuatro.En cada uno de estos grupos se compara la aptitud de cada uno de los individuos y de igual manera en cada grupo se tiene un ganador.Los ganadores de cada grupo son los individuos que se seleccionan de manera definitiva en esta ronda para ser reproducidos.

Al igual que el módulo de torneo, el de reproducción aritmética es puramente combinacional.En este módulo es donde se generan los hijos de los ganadores del torneo.La manera en como se generan los hijos es análoga a la contraparte en software, es una simple combinación lineal de los padres.

El módulo generador de direcciones se encarga de generar las direcciones donde se almacenarán los individuos de la nueva generación,el cual es una memoria RAM diferente a la de donde se tomaron los padres.Cabe destacar que las direcciones se generan en pares para poder almacenar de dos en dos individuos.A pesar de que el número de individuos que pertenecerá a la siguiente generación por cada ciclo de este módulo es cuatro,no es conveniente generar cuatro direcciones ya que esto consumiría más recursos

del FPGA y como la memoria con la que se cuenta es de dos puertos,de todas formas se tendrían que almacenar de dos en dos.

De la discusión anterior es fácil ver que esta serie de procesos deben estar bien sincronizados para que el funcionamiento del módulo en conjunto sea apropiado, por lo cual este módulo cuenta con una máquina de estados que hace las veces de secuenciador.

3.4.8. Módulo 5: Nueva generación

Este módulo se encarga de almacenar a los individuos que conformaran la siguiente generación.

En realidad la arquitectura de este módulo es muy similar a la del módulo que almacena la generación actual, con la diferencia de que se necesita una memoria RAM menos, ya que los que serán los individuos de la siguiente generación aún no cuentan con la aptitud.

3.4.9. Módulo 6: Contador de generaciones

Como su nombre lo indica este es un módulo que permite llevar la cuenta de las iteraciones que lleva el algoritmo, en otras palabras, el numero de generación en la que va el algoritmo.

En la figura 3.20 se muestra este módulo.

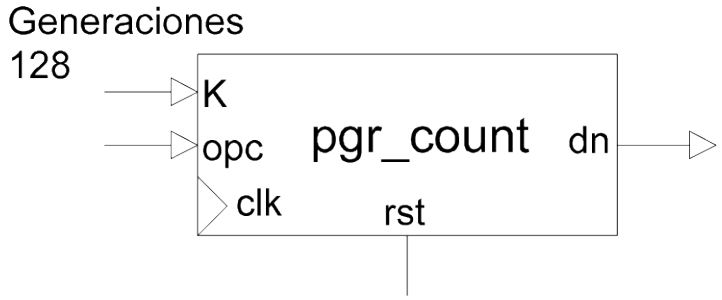

Figura 3.20. Módulo 6: Contador de generaciones

A pesar de que este es un módulo muy sencillo permite que se modifique el número de generaciones del algoritmo genético.

3.4.10. Módulo 7: El más apto

La misión de este módulo es simple pero muy crítica, se trata de almacenar al individuo más apto de encontrado hasta el momento a través de las generaciones.

En la Figura 3.21 se puede apreciar la arquitectura de este módulo.Consta principalmente de una serie de registros y un comparador.En los registros se tiene almacenado al individuo más apto de la generación presente, mientras que el comparador se encarga de poner a competir el individuo más apto hasta la generación anterior con el de la generación actual, esto claro para obtener al más apto de todas las generaciones.

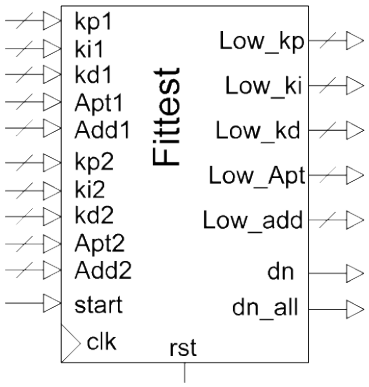

Figura 3.21. Módulo 7: Individuo más apto

3.4.11. Módulo 8: Direcciones de la nueva generación

La arquitectura de este módulo es muy simple.Consta básicamente de un contador y lógica combinacional de apoyo que le permite generar las direcciones de donde se irán tomando los individuos de la siguiente generación.Una vez que se lean los individuos de la siguiente generación se procede a obtener la aptitud de los mismos.Esta operación se lleva a cabo utilizando los módulos de obtención de aptitud anteriormente descritos.

3.4.12. Módulo 9: El secuenciador

Este módulo constituye una parte fundamental del sistema, ya que se encarga de ordenar a cada módulo que inicie su operación y está al pendiente de cuando termina la tarea asignada para poder continuar con el algoritmo.

En la Figura 3.22 se presenta el diagrama del secuenciador.

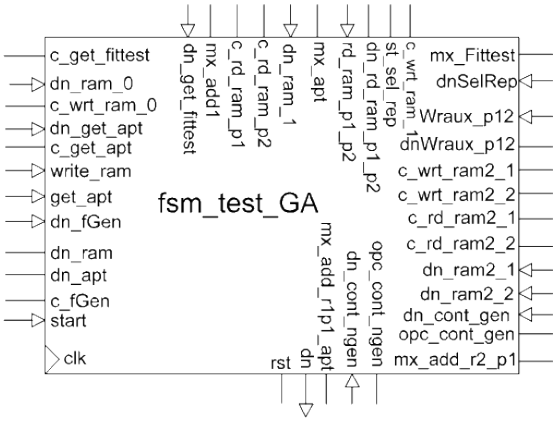

Figura 3.22. Módulo 9: Secuenciador

El secuenciador desempeña una función crucial para el correcto funcionamiento del sistema, ya que coordina a los demás módulos para que en conjunto se alcance el objetivo final. En la figura 3.23 se presenta un diagrama de flujo que presenta de manera lógica la tarea del secuenciador.

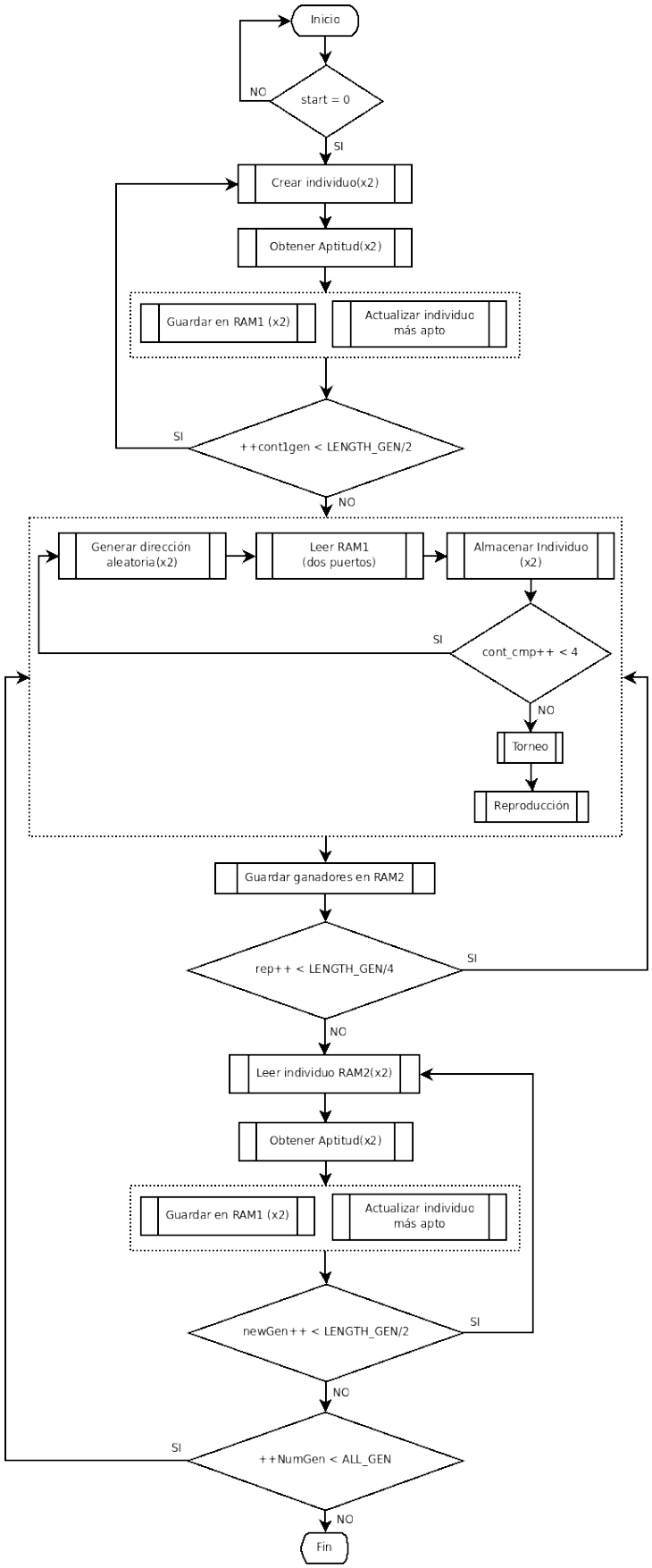

Figura 3.23. Diagrama de flujo de la operación del Secuenciador

4 RESULTADOS

En esta sección se presentan los resultados obtenidos, es decir, se utiliza el algoritmo genético diseñado para sintonizar un controlador PID. En la primera parte de este capitulo se prueba el diseño en software

En primera instancia se presenta el algoritmo genético corriendo en Matlab. Posteriormente se utiliza un algoritmo similar a este pero que se implementó en el lenguaje de programación C.Al final de esta sección se presenta una tabla comparativa entre estas dos versiones.

En la segunda parte de este capítulo se presentan los resultados obtenidos al utilizar el algoritmo genético para sintonizar un controlador PID, pero esta ocasión teniendo una FPGA como plataforma de implementación.

En la parte final de este capítulo se presenta un sistema de control de velocidad de un motor de corriente directa con la finalidad de validar el algoritmo.

4.1. Software

En la parte de software, como ya se ha mencionado con anterioridad, se diseñó e implementó un algoritmo genético capaz de sintonizar un controlador PID.Las primeras implementaciones se realizaron en Matlab.Se realizaron varias simulaciones de las implementaciones del algoritmo y a continuación se presentan algunas que se seleccionaron.

En la Figura 4.24 se aprecia como el error va disminuyendo conforme el algoritmo genético evoluciona, es decir, conforme avanzan las generaciones.

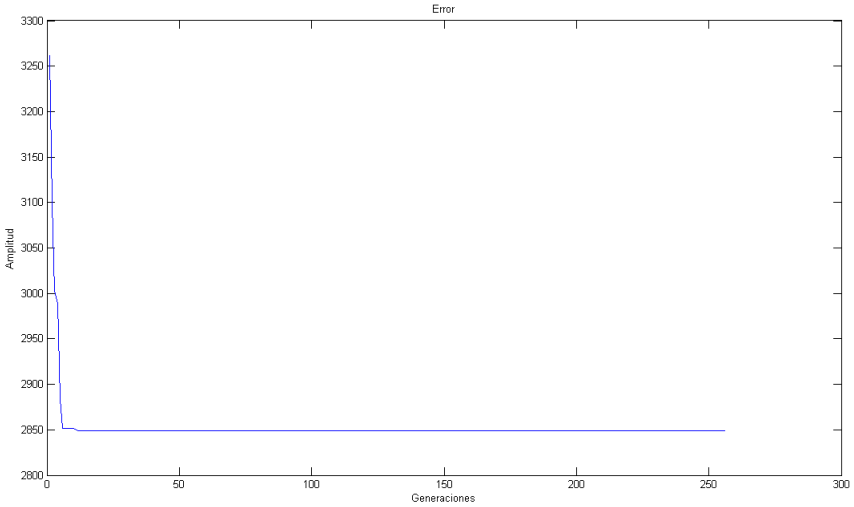

Figura 4.24. AG Matlab: Minimización del error

Por otro lado, la Figura 4.25 deja ver como evolucionan las ganancias conforme avanzan las generaciones.

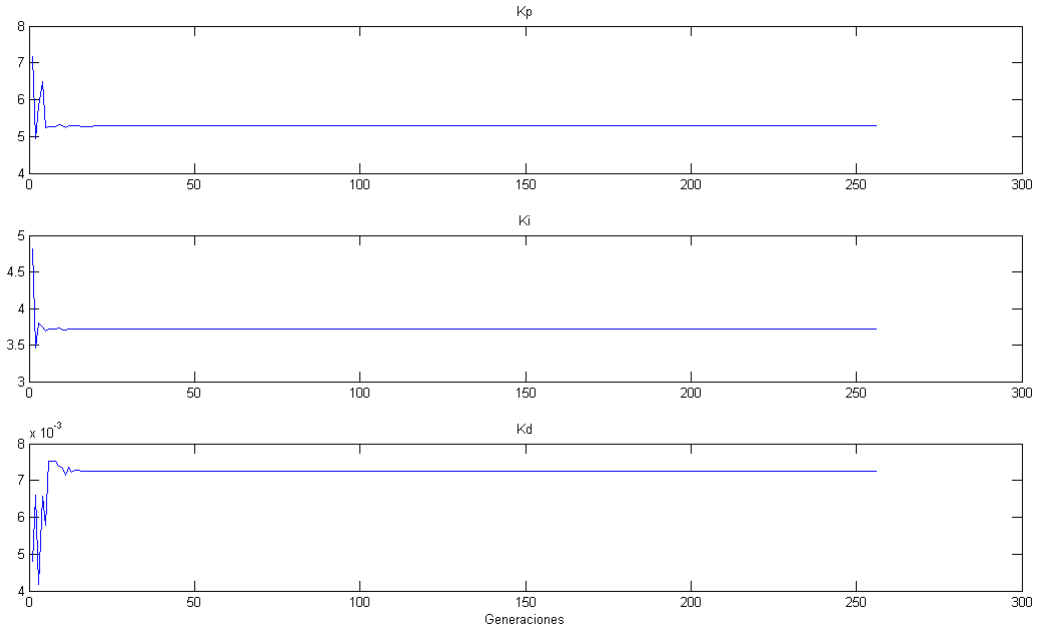

Figura 4.25. AG Matlab: Evolución de las ganancias

Ahora en la Figura 4.26 se presenta la respuesta de la planta sintonizada con las ganancias que encontró el algoritmo genético.

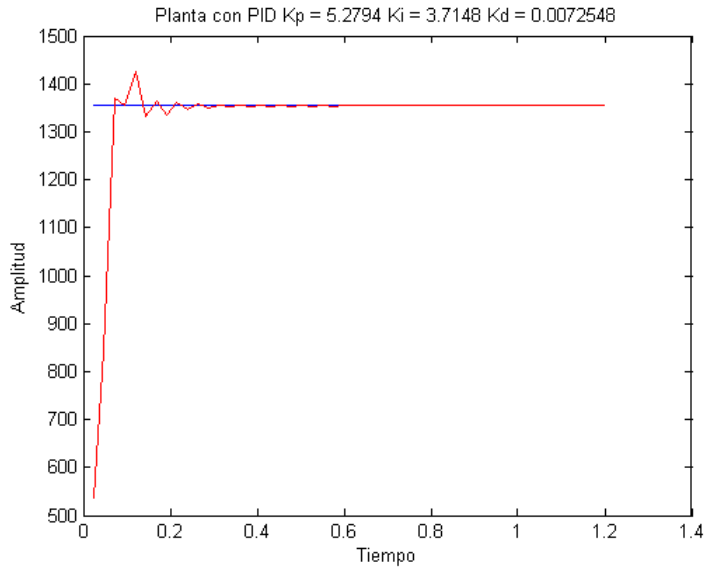

Figura 4.26. AG Matlab: Respuesta de la planta en lazo cerrado

Las tres figuras anteriores muestran la respuesta del algoritmo genético diseñado e implementado en Matlab. Es importante mencionar que el tiempo de ejecución fue de aproximadamente 22 segundos.

También es importante señalar que este algoritmo fue el resultado de hacer diversas pruebas con operadores genéticos, la finalidad de este algoritmo no fue ser óptimo, sino simplemente que tuviese la capacidad de sintonizar un controlador PID, sin embargo, con el objetivo de optimizar el desempeño del algoritmo en software se migro de lenguaje de Matlab a C.Además de la migración, el algoritmo se adaptó un poco, esto con la finalidad de poder compararlo más tarde con la implementación en hardware.

En la Figura 4.27 se presenta la respuesta de la planta con el algoritmo genético adaptado y compilado en C.

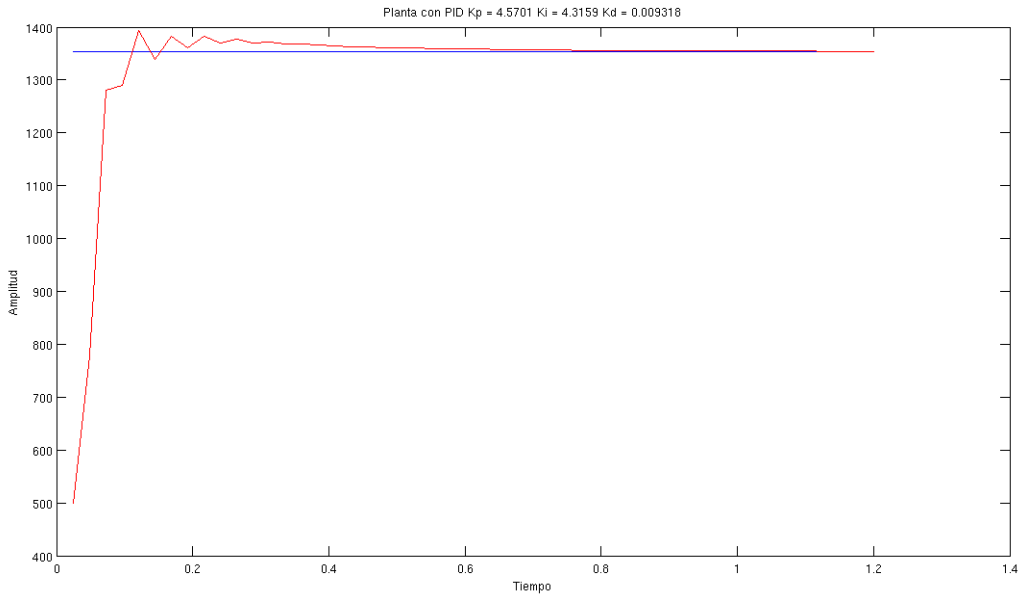

Figura 4.27. AG Matlab: Minimización del error

Se puede apreciar que los resultados son parecidos, a pesar de las adaptaciones realizadas, sin embargo, se debe recordar que los algoritmos genéticos son una técnica heurística y se basan fuertemente en procesos aleatorios, por lo cual aun siendo el mismo algoritmo es muy complicado que se tenga la misma salida cada que se ejecuta el algoritmo,a pesar de lo anterior de antemano se sabe que la respuesta será mejor que la primera que se proponga, esto por la naturaleza de la técnica.

Cabe destacar que efectivamente se optimizó el algoritmo diseñado previamente ya que en este caso el algoritmo se completó en un tiempo de aproximadamente 100 milisegundos.

4.2. Hardware

En esta sección se presentan los resultados que se obtienen al utilizar el algoritmo genético que se implementó en el FPGA.Cabe destacar que este algoritmo no es el mismo que está programado en Matlab ya que fue adaptado.De hecho con el fin de poder realizar las comparaciones pertinentes entre el programa en software y la implementación en hardware se migró el algoritmo al lenguaje de programación C.Aunque es

necesario recordar que el algoritmo que se programó en C no es exactamente el mismo que se programó en Matlab, sin embargo la implementación en hardware si es análoga al programa en C.

En la Figura 4.28 se presenta una simulación del sistema implementado en el FPGA.En esta simulación se realiza la sintonización de un controlador PID.En esta figura es posible apreciar la evolución de las ganancias conforme transcurre el tiempo.De igual manera se puede notar que el algorimo se completa en un tiempo de aproximadamente 38 milisegundos.

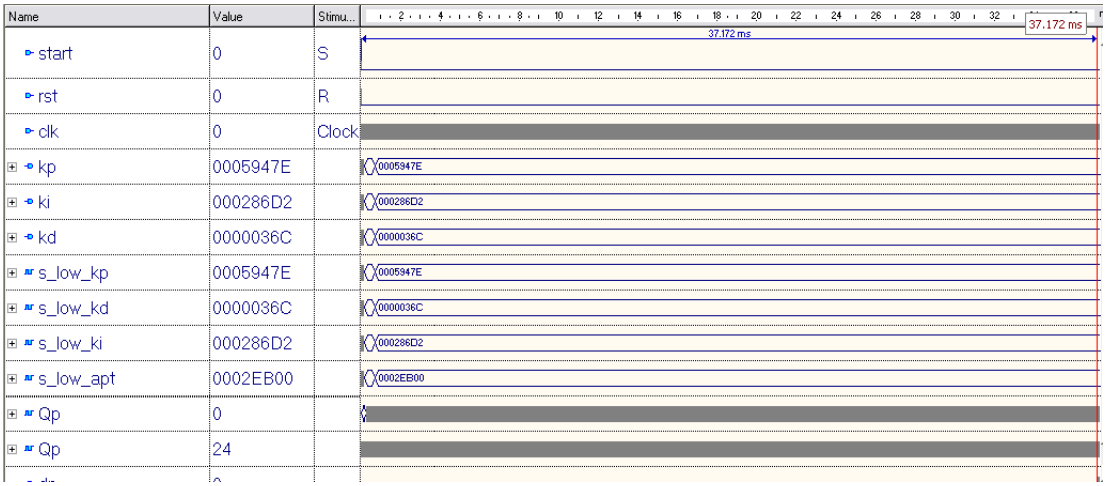

Figura 4.28. Primera simulación en FPGA: Evolución ganancias

En la Figura 4.29 es un acercamiento a la figura anterior.En este acercamiento se aprecia que la duración de cada generación es de aproximadamente 145 microsegundos a excepción de la primera que toma casi 300 microsegundos.

De igual manera se puede apreciar que en la generación 251 las ganancias son las que el algoritmo genético, en esta corrida, obtiene como las mejores.

Con las ganancias que se encontraron al finalizar el algoritmo se realizó una simulación de la planta en lazo cerrado con el controlador PID.La respuesta se presenta

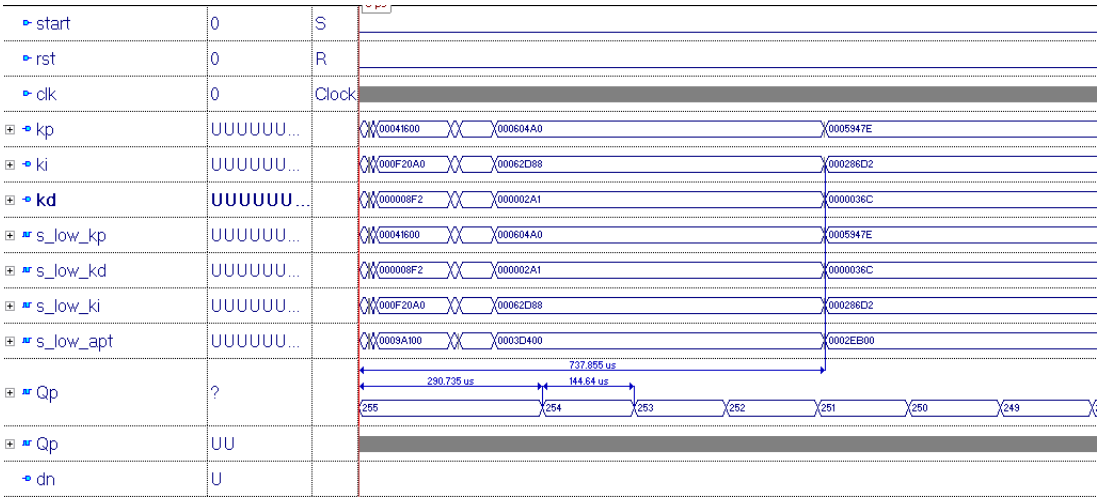

Figura 4.29. Primera simulación en FPGA: Detalles de la evolución de las ganancias en la Figura 4.30.

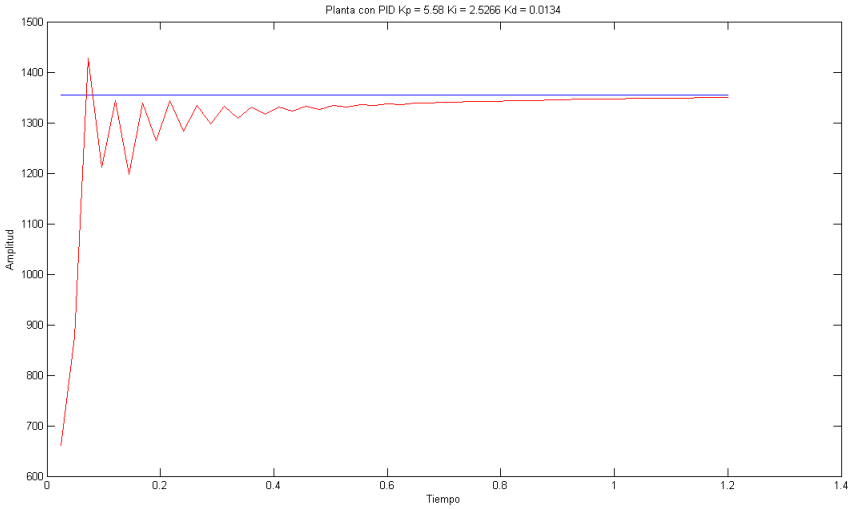

Figura 4.30. Primera simulación en FPGA: Respuesta de la planta en lazo cerrado

Por otro lado en la Figura 4.31 se presenta una segunda simulación del sintonizador.A partir de esta figura es posible darse cuenta que el tiempo en que tarda en ejecutase el algoritmo genético es el mismo que en la simulación anterior, lo cual era de esperarse ya que el sistema se diseño de esa manera.

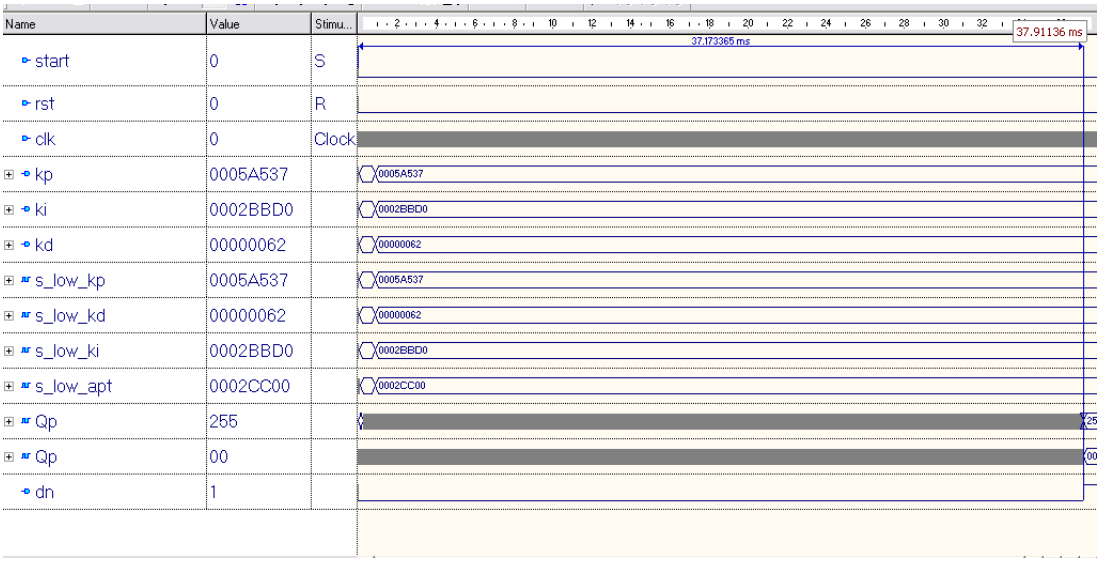

Figura 4.31. Segunda simulación en FPGA: evolución de las ganancias

En la Figura 4.32 se presenta un acercamiento a la Figura 4.31, en este acercamiento se nota que el tiempo que tarda el algoritmo genético en realizar una generación completa es el mismo que en la primera simulación.

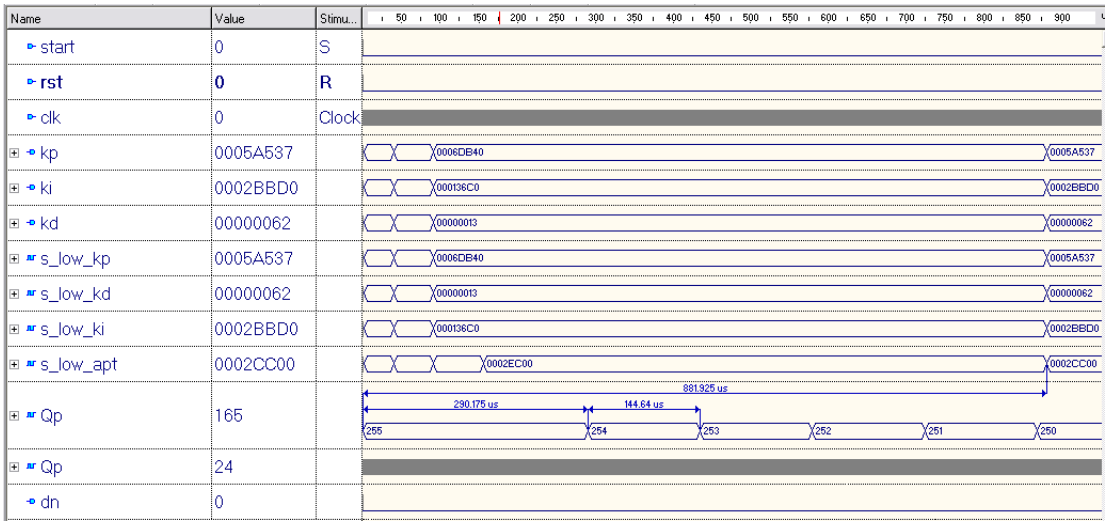

Figura 4.32. Segunda simulación en FPGA: Detalles de la evolución de las ganancias

En los acercamientos que se han realizado a las simulaciones se nota que la solución se encuentra en las primeras generaciones y que a pesar de que el tiempo de

termino del algoritmo genético es de aproximadamente 38 milisegundos, las soluciones en este caso se encuentran antes de un milisegundo.

Para esta segunda simulación la respuesta de la planta en lazo cerrado es la que se muestra en la Figura 4.33.

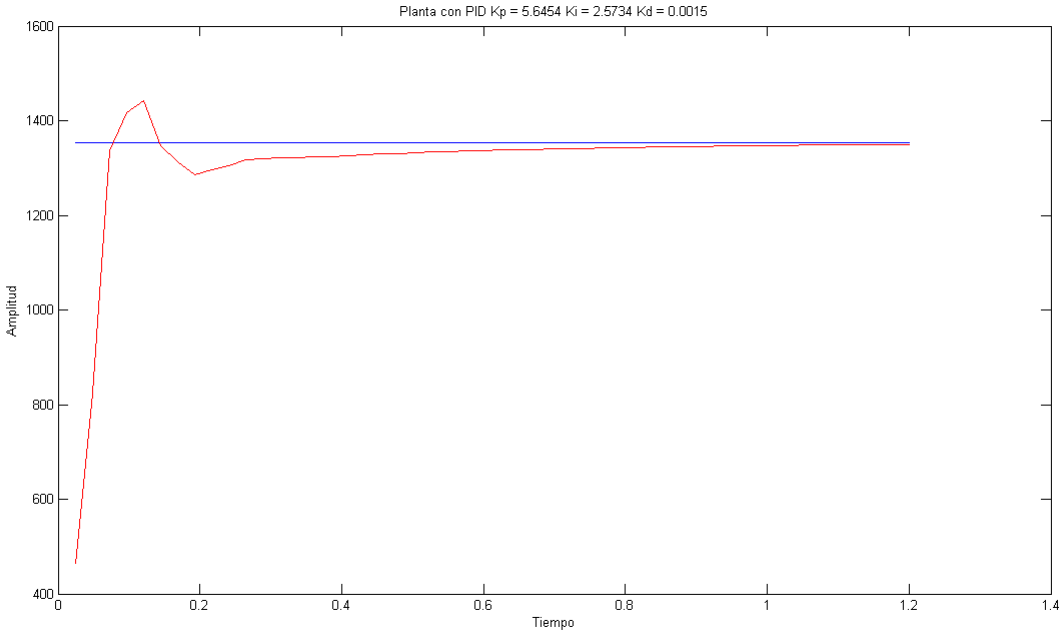

Figura 4.33. Segunda simulación en FPGA: Respuesta de la planta en lazo cerrado

Se realizó una tercera simulación.El resultado de esta se presenta en la Figura 4.34.A diferencia de las simulaciones anteriores en este caso la evolución del algoritmo es ligeramente distinta, ya que al principio se nota una evolución más dinámica de las ganancias.Además a pesar de que durante muchas generaciones el algoritmo genético parece que ya ha encontrado la solución llega un momento en el una ganancia evoluciona una ves más.

En la figura 4.35 se puede apreciar como la evolución de las ganancias es más dinámica en el principio del algoritmo genético.También se debe apreciar que que el

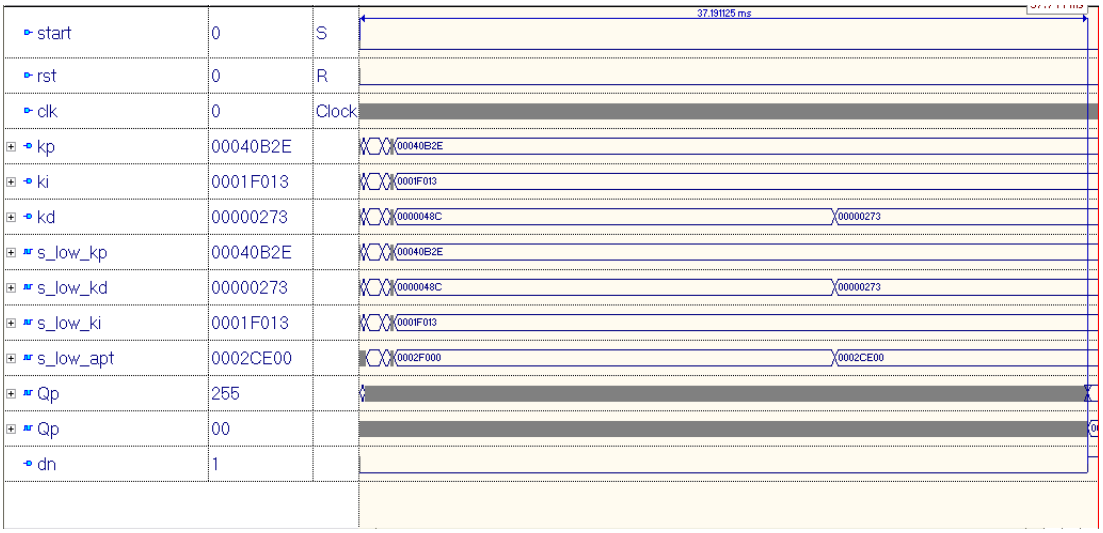

Figura 4.34. Tercera simulación en FPGA: Detalles de la evolución de las ganancias

tiempo en que tarda en realizarse cada generación es constante e igual al de las simulaciones anteriores.

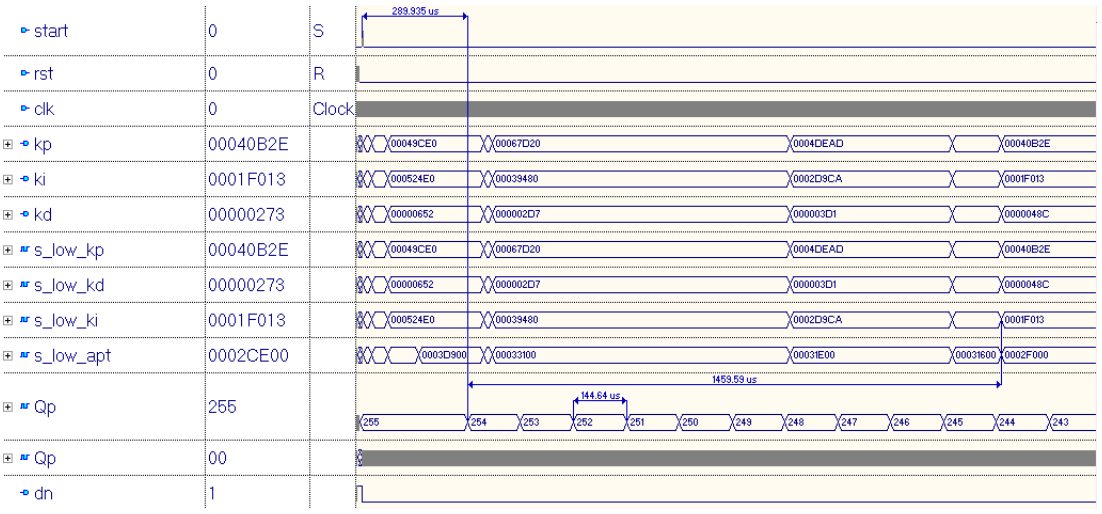

Figura 4.35. Tercera simulación en FPGA: Detalles de la evolución de las ganancias

Como ya se comentó con anterioridad las ganancias permanecen sin evolucionar a lo largo de un periodo considerable a lo largo de un buen periodo en esta simulación, sin embargo llega un momento en el algoritmo que una de ellas vuelve a evolucionar, los detalles de esta evolución se presentan en la Figura 4.36.

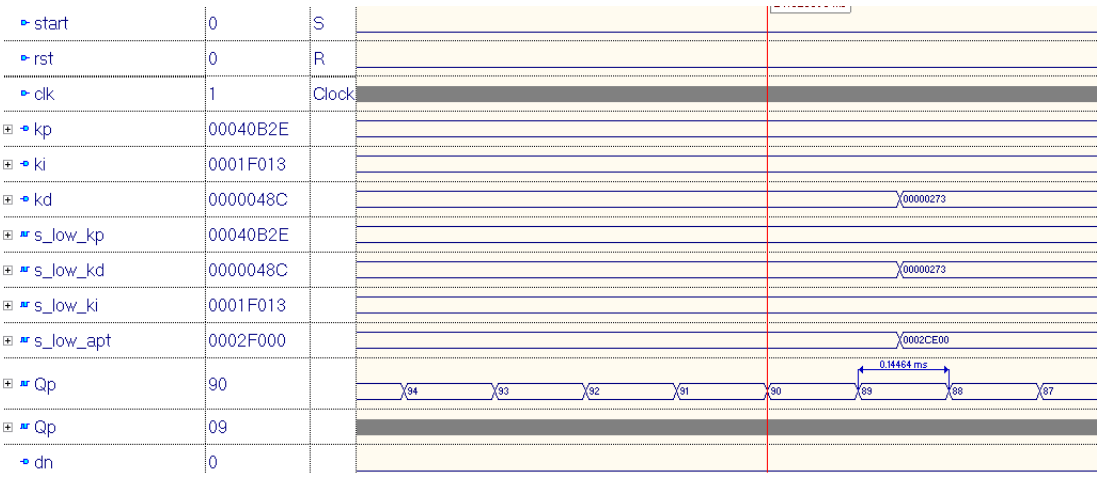

Figura 4.36. Tercera simulación en FPGA: Las ganancias evolucionan otra vez

Y en la Figura 4.37 se presenta la respuesta de la planta en lazo cerrado con las ganancias que el algoritmo genético encontró.

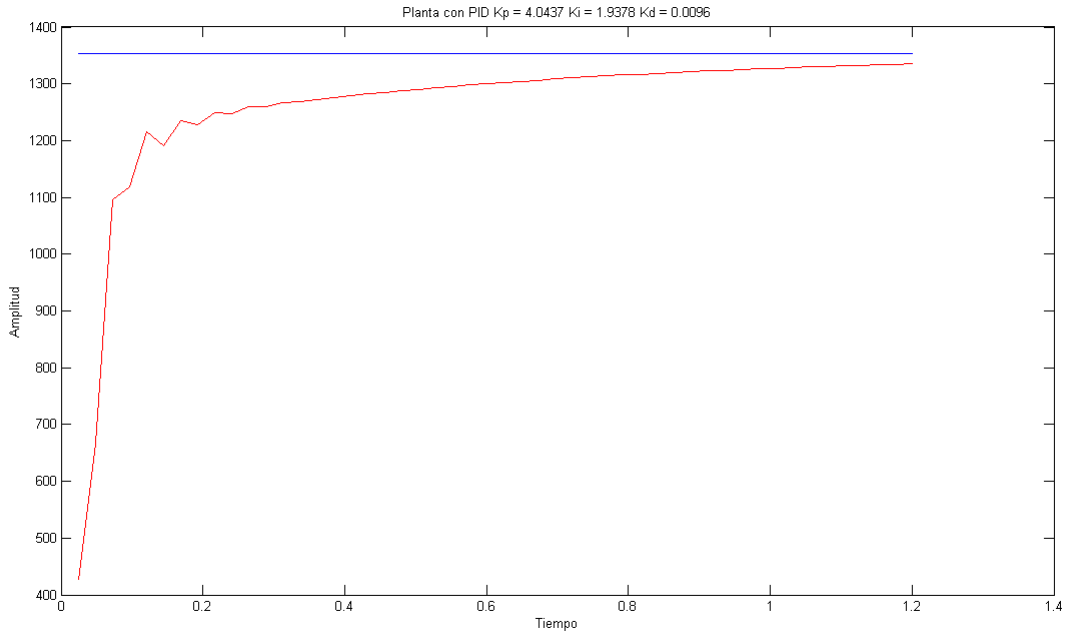

Figura 4.37. Tercera simulación en FPGA: Respuesta de la planta en lazo cerrado

4.3. Sistema de control

En esta sección se presentan los resultados obtenidos al utilizar el algoritmo genético para la sintonización del controlador PID, en este caso se controló la velocidad de un motor de corriente directa.

En la Figura 4.38 se presenta la idea general del sistema de control utilizado para validar el algoritmo genético.

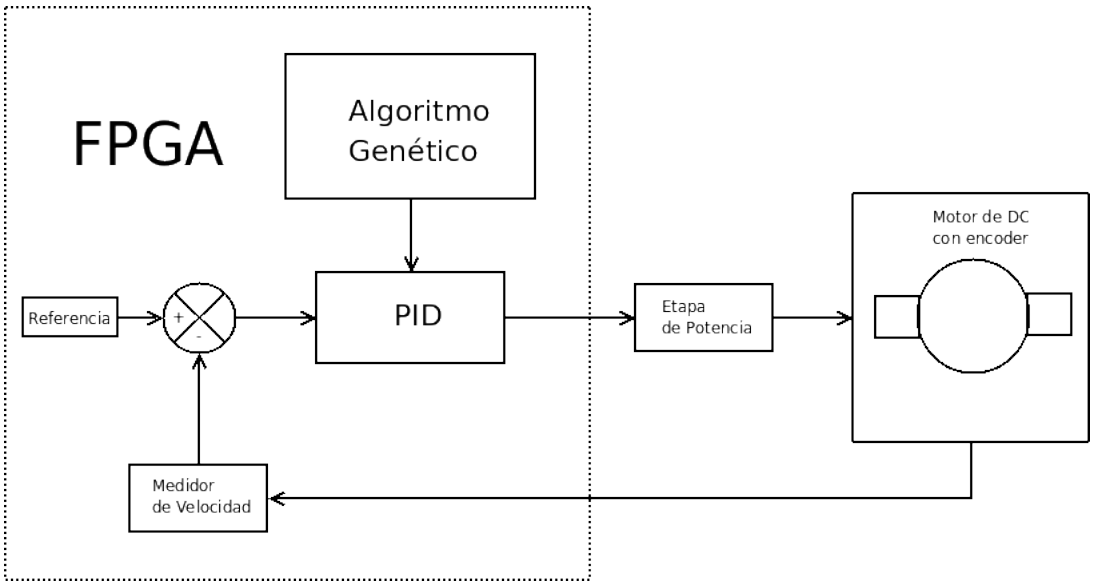

Figura 4.38. Sistema de control para validación del algoritmo genético

En esta figura es posible darse cuenta que el sistema completo, tanto el control como la instrumentación quedó implementado en el FPGA, claro que la etapa de potencia es la única que es externa al FPGA.

Ahora la en Figura 4.39 se muestra la respuesta del motor para el sistema de control que se mostró con anterioridad.

4.3.1. Comparativa entre plataformas

Después de haber desarrollado el algoritmo genético capaz de sintonizar controladores PID se migró de plataforma y de lenguaje de programación.La primer migración

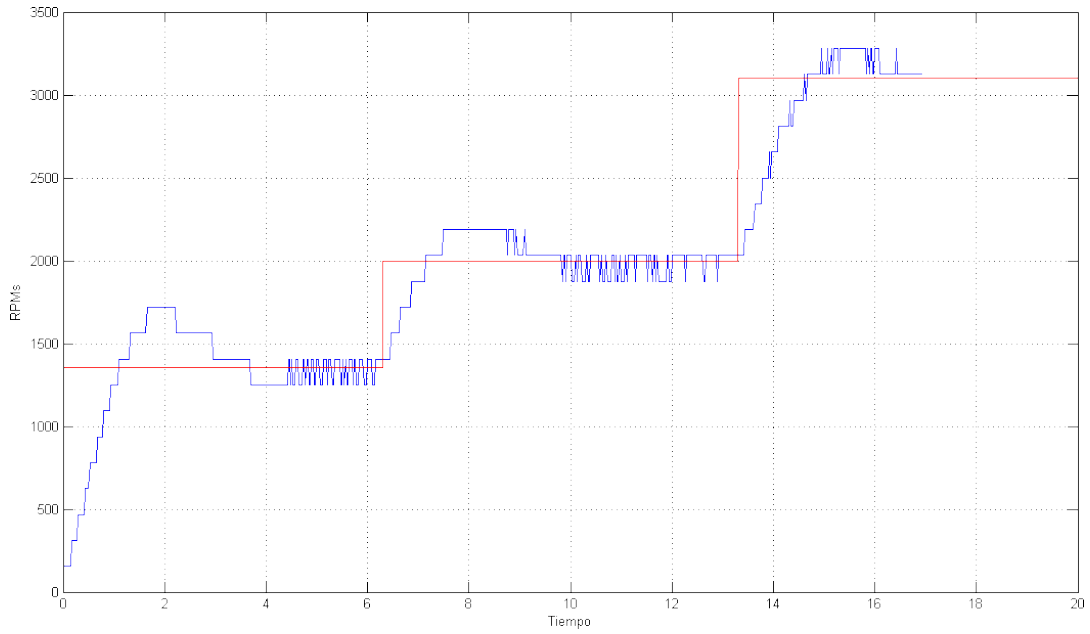

Figura 4.39. Respuesta del motor para validación del algoritmo genético

se debió al cambio de software a hardware, mientras que la segunda tuvo su motivación en hecho de que se realizara una comparación de tal manera que se aprovechara más el poder de computo del procesador para la versión en software y poner más de manifiesto el funcionamiento del algoritmo en hardware.

A continuación se presenta una tabla en la que se resume el desempeño obtenido.

Plataforma	Sistema Operativo	Lenguaje	Frecuencia	Tiempo de ejecución
Intel Dual Core Inside	Linux	Matlab	1.42 GHz	22 seg
Intel Dual Core Inside	Linux	C	1.42 GHz	110 ms
Virtex II Pro	No aplica	VHDL	100 MHz	37 ms

Cuadro 4.4. Comparación entre plataformas del desempeño del algoritmo genético

4.4. Conclusiones

En principio es posible aceptar la hipótesis de este trabajo, ya que después del desarrollo del mismo se ha logrado implementar un algoritmo genético capaz de sintonizar un controlador PID.

De este trabajo también se conluye que las versiones en software de los algoritmos a pesar de que pueden no ser la última respuesta al problema, esto debido al desempeño, si constituyen una herramienta de fundamental para la solución de problemas, ya que dan un primer acercamiento a la solución de estos.Sin embargo se debe tomar en cuenta que los beneficios que proporciona el utilizar herramientas de software tienen un precio; en principio el desempeño la mayoría de las veces es mejor en hardware que en software. Además si lo que se busca es utilizar las versiones en software par su posterior migración al hardware se debe tener en cuenta que plataforma de hardware se va utilizar, ya que si no dispone de los recursos necesarios para una migración intacta del algoritmo se debe evaluar cuanto afectará esta falta de recursos.Y aún suponiendo que se cuenten con los recursos necesarios para hacer la adaptación intacta, se debe valorar si vale la pena o si se están desperdiciando algunas ventajas que la descripción en hardware nos brinda como por ejemplo el paralelismo.

Es necesario hacer énfasis en que en este trabajo se utilizó como plataforma de hardware un FPGA, lo cual permitió un libertad y flexibilidad en el diseño, que al final se vio reflejado en el desempeño conseguido.Todo esto gracias a las bondades del FPGA como lo son su alta densidad de integración y capacidad para desarrollar estructuras de procesamiento en paralelo.

Del mismo modo ha de señalarse que a pesar de que en la actualidad existen lenguajes de alto nivel se utilizan para describir hardware, en esta ocasión se utilizo el lenguaje VHDL, el cual permitió tener más control de como se llevó a cabo la descripción del algoritmo a más bajo nivel lo cual trajo como consecuencia una alta eficiencia en los recursos utilizados en el FPGA.

Después de los resultados obtenidos en este trabajo es posible aceptar la hipótesis del mismo,y a pesar de que éste no es un sistema de control adaptable per se, es posible utilizar los resultados del mismo para elaborar uno, es decir, que la parte de adaptación de un sistema de control puede ser cubierta por el sintonizador genético

desarrollado en este trabajo, con lo cual queda como reto importante un trabajo en el que se pueda llevar a cabo una identificación de la planta, lo cual por supuesto no es un trabajo trivial.

5 APENDICE

5.1. Código del Algoritmo Genético en Matlab

```
function [ Gen X0 X1 X2 X3 ] = GENERIC_PID_GA4( SIZE_POP,NUM_GEN,r,aX,
                                                 bX,aY,bY,aZ,bZ)

%   Carlos Ricardo Luna Ortiz
%   Name:
%       Multivariable Genetic Algorithm
%   Description:
%       This programs shows implements a multivariable genetic algorithm
%       for the minimization of a given function
clc;
%%clear all;

%Formato de despliege de numeros
%format short
format long

%Parametros del algoritmo genetico

%Tamaño de la poblacion
%SIZE_POP = 128;

%Numero de Generaciones
%NUM_GEN = 50;

%Constante de recombinacion
%r=0.8;

%Mutacion
b_nonuniformity = 2;
r_mutation = 0.6;


%Creacion de las generaciones

%Creacion de la poblacion inicial
tic
new_generation = zeros(SIZE_POP,4);

%Se propone el metodo de generacion de numeros aleatorios y
% se da una semilla que depende del reloj
rand('twister',sum(100*clock))

%Para la primera variable
new_generation(:,2) = aX + (bX-aX).*rand( SIZE_POP , 1 );
%Para la segunda variable
new_generation(:,3) = aY + (bY-aY).*rand( SIZE_POP , 1 );
%Para la tercera variable
new_generation(:,4) = aZ + (bZ-aZ).*rand( SIZE_POP , 1 );

%Initial_population = zeros(2,SIZE_POP);
%for i = 0:SIZE_POP
%    Initial_population(i) = a + (b-a).*rand(100,1);
%end

iterator = 1;
%criterio de finalizacion del algoritmo
```

```
done = 0;
%ASIGNACION DE LA APTITUD DE CADA INDIVIDUO, el minimo es el más apto
while( iterator <= NUM_GEN  && done ~= 1)

    for i=1:SIZE_POP
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %Se crea el individuo actual
        x = new_generation(i,2);
        y = new_generation(i,3);
        z = new_generation(i,4);
        idividual = [x y z];
        apt = evaluate_sys3(0.024,50,x,y,z);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        new_generation(i,1) = apt;
    end
    %ORGANIZAR DE ACUERDO A LA APTITUD, se crean lo padres
    [ sorted_by_apt IX ] = sort(new_generation(:,1));
    for i=1:SIZE_POP
        %Se copia la aptitud del individuo
        parents(i,1) = new_generation( IX(i) , 1);
        %Se copia el individuo como tal
        parents(i,2) = new_generation( IX(i) , 2);
        parents(i,3) = new_generation( IX(i) , 3);
        parents(i,4) = new_generation( IX(i) , 4);
    end
    %COPIAR ELEMENTOS A LAS GENERACIONES
    Generations(:,:,iterator) = parents;
    %Seleccionar a los padres de la siguiente generacion
    %Elitismo
    parents = Generations(1:SIZE_POP,2:4,iterator);
    parents2=parents;

    %Ruleta
    %indices = round(1 + ((SIZE_POP/4)-1).*rand( SIZE_POP/2 , 1 ));
    %for i = 1:SIZE_POP/2
    %    parents(i,:) = Generations(indices(i),2:3,iterator);
    %end
    %Generacion de los hijos
    indices = round(1 + ((SIZE_POP/4)-1).*rand( SIZE_POP/2 , 1 ));
    for i = 1:SIZE_POP/2
        parents2(i,:)= parents( indices(i),: );
    end
    for i = 1:SIZE_POP/4
        %Primer cromosoma
        children((2*i)-1,2) =
                   r*parents2( (2*i)-1 ,1) +(1-r)*( parents2( (2*i) ,1 ));
        children((2*i),2) =
                   r*parents2( (2*i) ,1 ) + (1-r)*(parents2( (2*i)-1 ,1) );
        %Segundo cromosoma
        children((2*i)-1,3) =
                     r*parents2( (2*i)-1 ,2) +(1-r)*parents2( (2*i) ,2 ) ;
        children((2*i),3) =
                      r*parents2( (2*i) ,2 ) + (1-r)*parents2( (2*i)-1 ,2);
        %tercer cromosoma
        children((2*i)-1,4) =
                      r*parents2( (2*i)-1 ,3) +(1-r)*parents2( (2*i) ,3 ) ;
        children((2*i),4) =
```

```
                    r*parents2( (2*i) ,3 ) + (1-r)*parents2( (2*i)-1 ,3);
    end
    %Mutacion
    %Puede mutar un hijo
    rand('twister',sum(100*clock));
    index_to_mutate = ceil(1 + (SIZE_POP/2 -1).*rand( 1 , 1 ));
    y_mutation =
              Generations(1,2:4,iterator) - children(index_to_mutate,2:4);

    delta_mutation =
     y_mutation*r_mutation*power(1-((iterator-1)/NUM_GEN),b_nonuniformity);
    mutant = children(index_to_mutate,2:4)+delta_mutation;
    children(index_to_mutate,2:4) = mutant;

    %Copiar la nueva generacion
    %new_generation = [ parents(1:SIZE_POP/2,:) ];
    for i = 1:SIZE_POP/2
        %aptitud desconocida
        new_generation( i,1) = 0;
        %Asignacion de cada uno de los cromosomas
        new_generation( i,2) = parents( i,1);
        new_generation( i,3) = parents( i,2);
        new_generation( i,4) = parents( i,3);
    end
    %new_generation = [ parents(1:SIZE_POP/2,:) ];
    for i = 1:SIZE_POP/2
        new_generation( (SIZE_POP/2) +i,1:1) = 0;
        new_generation( (SIZE_POP/2) +i,2:4) = children( i,2:4 );
    end
    %Se almacena al mejor individuo de cada Genracion
    Best_Individuals_apt(iterator,1)=Generations(1,1,iterator);
    Best_Individuals_x(iterator,1) = Generations(1,2,iterator);
    Best_Individuals_y(iterator,1) = Generations(1,3,iterator);
    Best_Individuals_z(iterator,1) = Generations(1,4,iterator);
    Best_Individuals_total(iterator,:) = Generations(1,2:4,iterator);
   %se incrementa el contador de generaiones
    iterator = iterator+1;
end
toc

%Se muestra la evolucion de los individuos a traves de las generaciones
figure(3);
subplot(3,1,1);
plot(Best_Individuals_x);
title('Kp');
subplot(3,1,2);
plot(Best_Individuals_y);
title('Ki');
subplot(3,1,3);
plot(Best_Individuals_z);
title('Kd');
xlabel('Generaciones');


%Best_Individuals_apt
X0 = Generations(1,1,iterator-1);
X1 = Generations(1,2,iterator-1);
```

```
X2 = Generations(1,3,iterator-1);
X3 = Generations(1,4,iterator-1);
Gen = Generations;
figure(1);
plot(Best_Individuals_apt);
title('Error');
xlabel('Generaciones');
ylabel('Amplitud');
%stem(Best_Individuals_apt);
%figure(2)
%ezsurf(fnc,[-1 1])
%evaluate_sys_noL2(0.1,100,X1,X2,X3);
evaluate_sys3_graf(0.024,50,X1,X2,X3);
disp('dummy');
%evaluate_sys2(0.024,500,3.4,12.59,0.034);
```

5.2. Código del Algoritmo Genético en C

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <math.h>
#define LENGTH_GEN 128
#define SIM_TIME   50
#define NUM_GEN   128
#define AX 0
#define BX 16
#define AY 0
#define BY 7
#define AZ 0
#define BZ 1
#define AW 0
#define BW LENGTH_GEN
#define R 0.7

//prototipos de funciones
double get_time();

double evaluate_sys(double Ts, double Kp,double Ki, double Kd);

void tournament(double* Apts,int* indx_winners,int a1,int a2,int a3,
               int a4,int b1,int b2,int b3,int b4);

void crosover(double* son_a_Kp,double* son_a_Ki,double* son_a_Kd,
             double* son_b_Kp,double* son_b_Ki,double* son_b_Kd,
             double Kp_pa,double Ki_pa,double Kd_pa,double Kp_pb,
             double Ki_pb,double Kd_pb);


int main(int argc, char** argv)
{
 double time_start;
 double time_end;
 ////////////////////////////////////
 int i,j,jj,k,kk;
 int best_add;
 double Kp[LENGTH_GEN];
 double Ki[LENGTH_GEN];
 double Kd[LENGTH_GEN];

 double Kp_new[LENGTH_GEN];
 double Ki_new[LENGTH_GEN];
 double Kd_new[LENGTH_GEN];
 double Apt[LENGTH_GEN];

 int challengers_indxs[8];
 int winners_indx[2];

 double son_a_Kp,son_a_Ki,son_a_Kd;
 double son_b_Kp,son_b_Ki,son_b_Kd;

 time_start = get_time();

 srand48(time(NULL));
```

```
//Primera generacion
for (i=0 ; i < LENGTH_GEN ; i++)
{
 Kp[i] = AX + (BX-AX)*((double)drand48());
 Ki[i] = AY + (BY-AY)*((double)drand48());
 Kd[i] = AZ + (BZ-AZ)*((double)drand48());
 //obtener aptitud
 Apt[i] = evaluate_sys(0.024,Kp[i],Ki[i],Kd[i]);
 //Seleccionar al mejor
 if( i == 0 )
 {
  best_add = i;
 }
 else
 {
  if( Apt[best_add] < Apt[i] )
  {
   best_add = best_add;
  }
  else
  {
   best_add = i;
  }
 }
}
for( kk = 0; kk<NUM_GEN;kk++ )
{
 //Seleccion y reproduccion
 for(j = 0; j<8; j++)
 {
  challengers_indxs[j] = AW + (BW-AW)*drand48();
 }
 //Se realiza el torneo para encontrar a los dos individuos mas aptos
  tournament(&(Apt[0]),winners_indx,challengers_indxs[0],challengers_indxs[1],
             challengers_indxs[2],challengers_indxs[3],challengers_indxs[4],
             challengers_indxs[5],challengers_indxs[6],challengers_indxs[7]);

 //Se generan los hijos de los dos individuos mas aptos
 //En la primera iteracion se utiliza al mas apto
 for( k = 0; k<LENGTH_GEN;k = k+4 )
 {
  if( k == 0 )
  {
   crosover(&son_a_Kp,&son_a_Ki,&son_a_Kd,&son_b_Kp,&son_b_Ki,&son_b_Kd,
            Kp[best_add],Ki[best_add],Kd[best_add],Kp[winners_indx[1]],
            Ki[winners_indx[1]],Kd[winners_indx[1]]);
   Kp_new[k]   = Kp[best_add];
   Kp_new[k+1] = Kp[winners_indx[1]];
   Kp_new[k+2] = son_a_Kp;
   Kp_new[k+3] = son_b_Kp;

   Ki_new[k]   = Ki[best_add];
   Ki_new[k+1] = Ki[winners_indx[1]];
   Ki_new[k+2] = son_a_Ki;
   Ki_new[k+3] = son_b_Ki;
   Kd_new[k]   = Kd[best_add];
   Kd_new[k+1] = Kd[winners_indx[1]];
   Kd_new[k+2] = son_a_Kd;
   Kd_new[k+3] = son_b_Kd;
  }
  else
  {
   crosover(&son_a_Kp,&son_a_Ki,&son_a_Kd,&son_b_Kp,&son_b_Ki,&son_b_Kd,
            Kp[winners_indx[0]],Ki[winners_indx[0]],Kd[winners_indx[0]],
            Kp[winners_indx[1]],Ki[winners_indx[1]],Kd[winners_indx[1]]);
```

```
    Kp_new[k]   = Kp[winners_indx[0]];
    Kp_new[k+1] = Kp[winners_indx[1]];
    Kp_new[k+2] = son_a_Kp;
    Kp_new[k+3] = son_b_Kp;

    Ki_new[k]   = Ki[winners_indx[0]];
    Ki_new[k+1] = Ki[winners_indx[1]];
    Ki_new[k+2] = son_a_Ki;
    Ki_new[k+3] = son_b_Ki;

    Kd_new[k]   = Kd[winners_indx[0]];
    Kd_new[k+1] = Kd[winners_indx[1]];
    Kd_new[k+2] = son_a_Kd;
    Kd_new[k+3] = son_b_Kd;
   }
  }

  //Obtener la aptitud de la nueva generacion y copiar a la gen actual
  for(jj = 0;jj<LENGTH_GEN;jj++)
  {

   Kp[jj]  = Kp_new[jj];
   Ki[jj]  = Ki_new[jj];
   Kd[jj]  = Kd_new[jj];
   Apt[jj] = evaluate_sys(0.024,Kp[jj],Ki[jj],Kd[jj]);
   //Seleccionar al mejor
   if( jj == 0 )
   {
    best_add = jj;
   }
   else
   {
    if( Apt[best_add] < Apt[jj] )
    {
     best_add = best_add;
    }
    else
    {
     best_add = jj;
    }
   }
   //printf("Best ADD 2 %d\n",best_add);
  }

 }
 time_end = get_time();
 printf("Best Kp = %f, Ki = %f, Kd = %f, Apt = %f\n",Kp[best_add],
                          Ki[best_add],Kd[best_add],Apt[best_add]);
 printf("ERROR :%f",evaluate_sys(0.024,19,4,0.0002));
 printf("This program has run for: %f seconds.",  time_end - time_start);
 return 0;
}

double get_time()
{
 struct timeval t;
 gettimeofday(&t, NULL);
 double d = t.tv_sec + (double) t.tv_usec/1000000;
 return d;
}

double evaluate_sys(double Ts, double Kp,double Ki, double Kd)
{
 //iteradores
 unsigned int k = 0;

 //parametros de la planta
 //double a0 = 1.0;
 double a1 = 0.584208;
 double a2 = 0.303395;
 double b0 = 0.054;
 double b1 = 0.029;
```

```
double b2 = 0.040;

//Error inicial
double i_error = 0.0;

//Vector de error
double e[SIM_TIME];


//entrada
double u[SIM_TIME];

//salida
double y[SIM_TIME];

//Delta del controlador
double delta_c[SIM_TIME];

//Controlador
double c[SIM_TIME];

//Tiempo de acuerdo al tiempo de muestreo
unsigned int KTs[SIM_TIME];

//Referencia
double ref = 100.0;

for( k = 0; k< SIM_TIME; k++ )
{
 KTs[k] = k*Ts;

 //Entrada
 u[k] = ref;

 //Error
 if( k == 1 )
 {
  e[k] = u[k];
 }
 if( k > 1 )
 {
  e[k] = u[k-1] - y[k-1];
 }
 //Controlador
 if( k == 1 )
 {
  delta_c[k] = Kp*(( e[k] ) + (Ki*Ts*e[k]) + ((Kd/Ts)*e[k]));
 }
 else if(k == 2)
 {
  delta_c[k] = Kp*(( e[k]-e[k-1] ) + Ki*Ts*e[k] + (Kd/Ts)*(e[k]-2*e[k-1]));
 }
 else if( k > 2)
 {
  delta_c[k] = Kp*(( e[k]-e[k-1] ) + Ki*Ts*e[k] + (Kd/Ts)*(e[k]-2*e[k-1]+e[k-2]));
 }
 if (k == 1)
 {
  c[k] = delta_c[k];
 }
    if ( k > 1 )
 {
  c[k] = c[k-1]+delta_c[k];
 }

 //Salida
 if ( k == 1 )
 {
  y[k] = b0*c[k];
 }
 if( k == 2 )
 {
  y[k] = b0*c[k]+b1*c[k-1]+a1*y[k-1];
 }
 if ( k > 2 )
 {
```

```
   y[k] = b0*c[k]+b1*c[k-1]+b2*c[k-2]+a1*y[k-1]+a2*y[k-2];
  }
  i_error = i_error + fabs(e[k]);
 }
 return i_error;
}
void tournament(double* Apts,int* indx_winners,int a1,int a2,int a3,
                int a4,int b1,int b2,int b3,int b4)
{
 int inter1a,inter2a,inter1b,inter2b;
 int winnern_a_indx,winnern_b_indx;
 if( Apts[a1] > Apts[a2])
 {
  inter1a = a2;
 }
 else
 {
  inter1a = a1;
 }
 if( Apts[a3] > Apts[a4])
 {
  inter2a = a4;
 }
 else
 {
  inter2a = a3;
 }
 if( Apts[b1] > Apts[b2])
 {
  inter1b = b2;
 }
 else
 {
  inter1b = b1;
 }
 if( Apts[b3] > Apts[b4])
 {
  inter2b = b4;
 }
 else
 {
  inter2b = b3;
 }
 ////////////////////////////////////////////////
 if( Apts[inter1a] > Apts[inter2a] )
 {
  winnern_a_indx = inter2a;
 }
 else
 {
  winnern_a_indx = inter1a;
 }
 if( Apts[inter1b] > Apts[inter2b] )
 {
  winnern_b_indx = inter2b;
 }
 else
 {
  winnern_b_indx = inter1b;
 }
 ////////////////////////////////////////////////
 *(indx_winners)   = winnern_a_indx;
 *(indx_winners+1) = winnern_b_indx;
}
void crosover(double* son_a_Kp,double* son_a_Ki,double* son_a_Kd,
              double* son_b_Kp,double* son_b_Ki,double* son_b_Kd,
              double Kp_pa,double Ki_pa,double Kd_pa,double Kp_pb,
              double Ki_pb,double Kd_pb)
{
 *(son_a_Kp) = (R*Kp_pa) + ((1-R)*Kp_pb);
 *(son_a_Ki) = (R*Ki_pa) + ((1-R)*Ki_pb);
 *(son_a_Kd) = (R*Kd_pa) + ((1-R)*Kd_pb);
```

```
 *(son_b_Kp) = (R*Kp_pb) + ((1-R)*Kp_pa);
 *(son_b_Ki) = (R*Ki_pb) + ((1-R)*Ki_pa);
 *(son_b_Kd) = (R*Kd_pb) + ((1-R)*Kd_pa);
}
```

5.3. Código del Algoritmo Genético en VHDL: Bloque de Mayor Jerarquía

```
library IEEE;
use ieee.std_logic_1164.all;

entity GA4 is
 port(
  rst: in std_logic;
  clk: in std_logic;
  ------------------
  start: in  std_logic;
  kp:    out std_logic_vector(31 downto 0);
  ki:    out std_logic_vector(31 downto 0);
  kd:    out std_logic_vector(31 downto 0);
  dn:    out std_logic
 );
end GA4;

architecture estructural of GA4 is

component GEN1
 port(
  rst:    in  std_logic;
  clk:    in  std_logic;
  ------------------------------------------
  start:  in  std_logic;
  ------------------------------------------
  kp1:   out std_logic_vector(31 downto 0);
  ki1:   out std_logic_vector(31 downto 0);
  kd1:   out std_logic_vector(31 downto 0);
  ------------------------------------------
  kp2:   out std_logic_vector(31 downto 0);
  ki2:   out std_logic_vector(31 downto 0);
  kd2:   out std_logic_vector(31 downto 0);
  ------------------------------------------
  get_apt: out std_logic;
  dn_apt:  in  std_logic;
  ------------------------------------------
  add1:          out std_logic_vector(6 downto 0);
  add2:          out std_logic_vector(6 downto 0);
  ------------------------------------------
  write_ram:   out std_logic;
  dn_write_ram: in  std_logic;
  ------------------------------------
  dn:     out std_logic
 );
end component;

component Apt_Fnc
 port(
  rst:         in  std_logic;         --reset maestro
  clk:         in  std_logic;         --reloj maestro
  ------------------------------------------------------------
  start:       in  std_logic;         --Bandera nuevas ganancias controlador
  iteraciones: in std_logic_vector(7 downto 0);
  ------------------------------------------------------------
  referencia:  in  std_logic_vector(15 downto 0);
  ------------------------------------------------------------
  kp:          in  std_logic_vector(31 downto 0);
  ki:          in  std_logic_vector(31 downto 0);
  kd:    in  std_logic_vector(31 downto 0);
  ------------------------------------------------------------

  kp_ldd:   out std_logic_vector(31 downto 0);
  ki_ldd:   out std_logic_vector(31 downto 0);
  kd_ldd:   out std_logic_vector(31 downto 0);
  apt:   out std_logic_vector(31 downto 0);
  dn:          out std_logic
 );
```

```
end component;
component RAM1A
 port(
  rst: in std_logic;
  clk: in std_logic;
  --------------------
  rw:  in std_logic;
  wa: in std_logic;
  wb: in std_logic;
  ra: in std_logic;
  rb: in std_logic;
  ---------------------------------------
  kp_0i:  in std_logic_vector(31 downto 0);
  ki_0i:   in std_logic_vector(31 downto 0);
  kd_0i:  in std_logic_vector(31 downto 0);
  apt_0i: in std_logic_vector(31 downto 0);
  ---------------------------------------
  kp_1i:  in std_logic_vector(31 downto 0);
  ki_1i:   in std_logic_vector(31 downto 0);
  kd_1i:  in std_logic_vector(31 downto 0);
  apt_1i: in std_logic_vector(31 downto 0);
  ---------------------------------------
  aa:  in std_logic_vector(6 downto 0);
  ab:  in std_logic_vector(6 downto 0);
  ---------------------------------------
  a0:  in std_logic_vector(4 downto 0);
  a1:  in std_logic_vector(4 downto 0);
  a2:  in std_logic_vector(4 downto 0);
  a3:  in std_logic_vector(4 downto 0);
  a4:  in std_logic_vector(4 downto 0);
  a5:  in std_logic_vector(4 downto 0);
  a6:  in std_logic_vector(4 downto 0);
  a7:  in std_logic_vector(4 downto 0);
  ---------------------------------------
  kp_0o:  out std_logic_vector(31 downto 0);
  ki_0o:  out std_logic_vector(31 downto 0);
  kd_0o:  out std_logic_vector(31 downto 0);
  apt_0o: out std_logic_vector(31 downto 0);
  ---------------------------------------
  kp_1o:  out std_logic_vector(31 downto 0);
  ki_1o:  out std_logic_vector(31 downto 0);
  kd_1o:  out std_logic_vector(31 downto 0);
  apt_1o: out std_logic_vector(31 downto 0);
  ---------------------------------------
  kp_2o:  out std_logic_vector(31 downto 0);
  ki_2o:  out std_logic_vector(31 downto 0);
  kd_2o:  out std_logic_vector(31 downto 0);
  apt_2o: out std_logic_vector(31 downto 0);
  ---------------------------------------
  kp_3o:  out std_logic_vector(31 downto 0);
  ki_3o:  out std_logic_vector(31 downto 0);
  kd_3o:  out std_logic_vector(31 downto 0);
  apt_3o: out std_logic_vector(31 downto 0);
  ---------------------------------------
  kp_4o:  out std_logic_vector(31 downto 0);
  ki_4o:  out std_logic_vector(31 downto 0);
  kd_4o:  out std_logic_vector(31 downto 0);
  apt_4o: out std_logic_vector(31 downto 0);
  ---------------------------------------
  kp_5o:  out std_logic_vector(31 downto 0);
  ki_5o:  out std_logic_vector(31 downto 0);
  kd_5o:  out std_logic_vector(31 downto 0);
  apt_5o: out std_logic_vector(31 downto 0);
  ---------------------------------------
  kp_6o:  out std_logic_vector(31 downto 0);
  ki_6o:  out std_logic_vector(31 downto 0);
  kd_6o:  out std_logic_vector(31 downto 0);
  apt_6o: out std_logic_vector(31 downto 0);
  ---------------------------------------
  kp_7o:  out std_logic_vector(31 downto 0);
```

```
  ki_7o:  out std_logic_vector(31 downto 0);
  kd_7o:  out std_logic_vector(31 downto 0);
  apt_7o: out std_logic_vector(31 downto 0);
  ---------------------------------------
  ya_dn: out std_logic;
  yb_dn: out std_logic
 );
end component;
component Fittest
 port(
  rst: in  std_logic;
  clk: in  std_logic;
  ---------------------------------------------
  Kp1:  in  std_logic_vector( 31 downto 0 );
  Ki1:  in  std_logic_vector( 31 downto 0 );
  Kd1:  in  std_logic_vector( 31 downto 0 );
  Apt1:  in  std_logic_vector( 31 downto 0 );
  Add1:    in  std_logic_vector( 6 downto 0 );
  ---------------------------------------------
  Kp2:  in  std_logic_vector( 31 downto 0 );
  Ki2:  in  std_logic_vector( 31 downto 0 );
  Kd2:  in  std_logic_vector( 31 downto 0 );
  Apt2:  in  std_logic_vector( 31 downto 0 );
  Add2:    in  std_logic_vector( 6 downto 0 );
  ---------------------------------------------
  start:  in  std_logic;
  ------------------------------
  low_Kp:  out std_logic_vector( 31 downto 0 );
  low_Ki:  out std_logic_vector( 31 downto 0 );
  low_Kd:  out std_logic_vector( 31 downto 0 );
  low_Apt: out std_logic_vector( 31 downto 0 );
  low_Add: out std_logic_vector( 6 downto 0 );
  dn:   out std_logic;
  dn_all:  out std_logic
 );
end component;
component SelRep
  port(
  rst:   in std_logic;
  clk:   in std_logic;
  -------------------------
  rd_ram_p1_p2:     out std_logic;
  ----------------------------------------------------
  add_ram_p0:       out std_logic_vector(4 downto 0);
  add_ram_p1:       out std_logic_vector(4 downto 0);
  add_ram_p2:       out std_logic_vector(4 downto 0);
  add_ram_p3:       out std_logic_vector(4 downto 0);
  add_ram_p4:       out std_logic_vector(4 downto 0);
  add_ram_p5:       out std_logic_vector(4 downto 0);
  add_ram_p6:       out std_logic_vector(4 downto 0);
  add_ram_p7:       out std_logic_vector(4 downto 0);
  ----------------------------------------------------
  dn_rd_ram_p1_p2:  in std_logic;
  ----------------------------------------------------
  kp_p0:            in std_logic_vector(31 downto 0);
  ki_p0:            in std_logic_vector(31 downto 0);
  kd_p0:            in std_logic_vector(31 downto 0);
  apt_0:     in std_logic_vector(31 downto 0);
  kp_p1:            in std_logic_vector(31 downto 0);
  ki_p1:            in std_logic_vector(31 downto 0);
  kd_p1:            in std_logic_vector(31 downto 0);
  apt_1:     in std_logic_vector(31 downto 0);
  kp_p2:            in std_logic_vector(31 downto 0);
  ki_p2:            in std_logic_vector(31 downto 0);
  kd_p2:            in std_logic_vector(31 downto 0);
  apt_2:     in std_logic_vector(31 downto 0);
  kp_p3:            in std_logic_vector(31 downto 0);
  ki_p3:            in std_logic_vector(31 downto 0);
```

```
  kd_p3:            in std_logic_vector(31 downto 0);
  apt_3:     in std_logic_vector(31 downto 0);
  kp_p4:            in std_logic_vector(31 downto 0);
  ki_p4:            in std_logic_vector(31 downto 0);
  kd_p4:            in std_logic_vector(31 downto 0);
  apt_4:     in std_logic_vector(31 downto 0);
  kp_p5:            in std_logic_vector(31 downto 0);
  ki_p5:            in std_logic_vector(31 downto 0);
  kd_p5:            in std_logic_vector(31 downto 0);
  apt_5:     in std_logic_vector(31 downto 0);
  kp_p6:            in std_logic_vector(31 downto 0);
  ki_p6:            in std_logic_vector(31 downto 0);
  kd_p6:            in std_logic_vector(31 downto 0);
  apt_6:     in std_logic_vector(31 downto 0);
  kp_p7:            in std_logic_vector(31 downto 0);
  ki_p7:            in std_logic_vector(31 downto 0);
  kd_p7:            in std_logic_vector(31 downto 0);
  apt_7:     in std_logic_vector(31 downto 0);
  ------------------------------------------------------
  start:            in std_logic;
  ------------------------------------------------------
  wrt_ram_aux_p1_p2:  out std_logic;
  dn_wrt_ram_p1_p2:   in std_logic;
  add_ram_aux_p1:     out std_logic_vector(6 downto 0);
  add_ram_aux_p2:  out std_logic_vector(6 downto 0);
  new_kp_p1:   out std_logic_vector(31 downto 0);
  new_ki_p1:          out std_logic_vector(31 downto 0);
  new_kd_p1:   out std_logic_vector(31 downto 0);
  new_kp_p2:   out std_logic_vector(31 downto 0);
  new_ki_p2:   out std_logic_vector(31 downto 0);
  new_kd_p2:   out std_logic_vector(31 downto 0);
  dn:     out std_logic
 );
end component;
component RAM2
 port(
  rst:     in  std_logic;
  clk:     in  std_logic;
  ----------------------------------------
  x0_Kp:   in std_logic_vector(31 downto 0);
  x0_Ki:  in std_logic_vector(31 downto 0);
  x0_Kd:  in std_logic_vector(31 downto 0);
  ----------------------------------------
  x1_Kp:   in std_logic_vector(31 downto 0);
  x1_Ki:  in std_logic_vector(31 downto 0);
  x1_Kd:  in std_logic_vector(31 downto 0);
  ----------------------------------------
  x0_add:   in  std_logic_vector(6 downto 0);
  x1_add:   in  std_logic_vector(6 downto 0);
  ----------------------------------------
  x0_write: in  std_logic;
  x1_write: in  std_logic;
  x0_read:  in  std_logic;
  x1_read:  in  std_logic;
  ----------------------------------------
  y0_Kp:   out std_logic_vector(31 downto 0);
  y0_Ki:  out std_logic_vector(31 downto 0);
  y0_Kd:  out std_logic_vector(31 downto 0);
  ----------------------------------------
  y1_Kp:   out std_logic_vector(31 downto 0);
  y1_Ki:  out std_logic_vector(31 downto 0);
  y1_Kd:  out std_logic_vector(31 downto 0);
  ----------------------------------------
  y0_dn: out std_logic;
  y1_dn: out std_logic
 );
end component;
component fsm_test_GA
 port(
```

```
  rst: in std_logic;
  clk: in std_logic;
  --------------------------------
  c_get_fittest:  out std_logic;
  dn_ram0:  in  std_logic;
  c_wrt_ram_0: out std_logic;
  dn_get_apt:  in  std_logic;
  c_get_apt:  out std_logic;
  write_ram:  in  std_logic;
  get_apt:  in  std_logic;
  dn_fGen:  in  std_logic;
  dn_ram:   out std_logic;
  dn_apt:   out std_logic;
  c_fGen:   out std_logic;
  start:   in  std_logic;
  --------------------------------
  dn_get_fittest:   in  std_logic;
  mx_add1:    out std_logic;
  c_rd_ram_p1:   out std_logic;
  c_rd_ram_p2:   out std_logic;
  dn_ram_1:    in  std_logic;
  mx_apt:     out std_logic;
  rd_ram_p1_p2:   in  std_logic;
  dn_rd_ram_p1_p2:  out std_logic;
  st_sel_rep:    out std_logic;
  --------------------------------
  mx_Fittest:    out std_logic;
  dnSelRep:    in  std_logic;
  Wraux_p12:    in  std_logic;
  dnWraux_p12:   out std_logic;
  c_wrt_ram2_1:  out std_logic;
  c_wrt_ram2_2:  out std_logic;
  c_rd_ram2_1:   out std_logic;
  c_rd_ram2_2:   out std_logic;
  dn_ram2_1:    in  std_logic;
  dn_ram2_2:    in  std_logic;
  dn_cont_gen:   in  std_logic;
  opc_cont_gen:  out std_logic_vector(1 downto 0);
  mx_add_r2_p1:  out std_logic;
  --------------------------------
  opc_cont_ngen:   out std_logic_vector(1 downto 0);
  dn_cont_ngen:    in  std_logic;
  mx_add_r1p1_apt: out std_logic;
  ld_this_fittest: out std_logic;
  c_wrt_ram_1:  out std_logic;
  dn:        out std_logic
 );
end component;
component Mux2_1_n
 generic(
  n: integer := 8
 );
 port(
  I0,I1: in  std_logic_vector(n-1 downto 0);
  S:        in  std_logic;
  Y:        out std_logic_vector(n-1 downto 0)
 );
end component;

component pgr_cntr_zero
 generic(
  n: integer :=8
 );
 port(
  rst: in  std_logic;
  clk: in  std_logic;
  opc: in  std_logic_vector( 1 downto 0 );
  cta: in  std_logic_vector( n-1 downto 0 );
  dn:  out std_logic
```

```
 );
end component;
component prgr_count_cta
 generic (
 n: integer :=8
 );
 port (
 RST:  in  std_logic;
 CLK:  in  std_logic;
 W:    in  std_logic_vector(n-1 downto 0);
 OPC:  in  std_logic_vector(1 downto 0);
 zero: out std_logic;
 q:    out std_logic_vector (n-1 downto 0)
 );
end component;
component new_gen
 port(
  rst:  in  std_logic;
  clk:  in  std_logic;
  opc:  in  std_logic_vector(1 downto 0);
  add1: out std_logic_vector(6 downto 0);
  add2: out std_logic_vector(6 downto 0);
  dn:   out std_logic
 );
end component;
component reg_n
   generic(n : integer:= 8  -- Definicion generica con n=8
      );
   port(
     rst: in  std_logic;
  clk: in  std_logic;
       ld:  in  std_logic;
       D :  in  std_logic_vector(n-1 downto 0);
      Q :  out std_logic_vector(n-1 downto 0)
     );
end component;
component Better
 port(
  Apt_now: in  std_logic_vector(31 downto 0);
  Apt_old: in  std_logic_vector(31 downto 0);
  ----------------------------------------
  mx_best: out std_logic
 );
end component;

--seniales
--fsm
signal s_c_get_fittest,s_c_wrt_ram_0,s_c_get_apt,
       s_dn_ram,s_dn_apt,s_c_fGen:                 std_logic;

signal s_mx_add1,s_c_rd_ram_p1,s_c_rd_ram_p2,s_mx_apt
           ,s_dn_rd_ram_p1_p2,s_st_sel_rep:        std_logic;

signal s_mx_Fittest,s_dnWraux_p12,s_c_wrt_ram2_1,
       s_c_wrt_ram2_2,s_c_rd_ram2_1,s_c_rd_ram2_2: std_logic;
signal s_mx_add_r2_p1,s_mx_add_r1p1_apt,s_ld_this_fittest,
                        s_c_wrt_ram_1,s_rw_ram1a: std_logic;
signal s_opc_cont_gen,s_opc_cont_ngen: td_logic_vector(1 downto 0);

--------------------
--gen1
signal s_gen_gwrite_ram,s_gen1_get_apt,s_gen1_dn: std_logic;
signal s_gen1_all_gains1,s_gen1_all_gains2: std_logic_vector(95 downto 0);
signal s_gen1_kp1,s_gen1_ki1,s_gen1_kd1:    std_logic_vector(31 downto 0);
signal s_gen1_kp2,s_gen1_ki2,s_gen1_kd2:    std_logic_vector(31 downto 0);
```

```
signal s_gen_add1:                    std_logic_vector(6 downto 0);
signal s_gen_add2:                    std_logic_vector(6 downto 0);
--fnc_apt1
signal s_fnc_apt1_dn: std_logic;
signal s_kp_ldd_1,s_ki_ldd_1,s_kd_ldd_1,s_fnc_apt_Apt_1: std_logic_vector(31 downto 0);
--fnc_apt2
signal s_fnc_apt2_dn: std_logic;
signal s_kp_ldd_2,s_ki_ldd_2,s_kd_ldd_2,s_fnc_apt_Apt_2: std_logic_vector(31 downto 0);
--fittest
signal s_low_kp,s_low_kd,s_low_ki,s_low_apt: std_logic_vector(31 downto 0);
signal s_low_add:                            std_logic_vector(6 downto 0);
signal s_fittest_dn,s_fittest_dn_all:        std_logic;
signal s_fittest_individual_no_apt:          std_logic_vector(95 downto 0);
signal s_fittest_individual:                 std_logic_vector(127 downto 0);
--ram
signal s_ram_y0_kp,s_ram_y0_ki,s_ram_y0_kd,s_ram_y0_apt: std_logic_vector(31 downto 0);
signal s_ram_y1_kp,s_ram_y1_ki,s_ram_y1_kd,s_ram_y1_apt: std_logic_vector(31 downto 0);
signal s_ram_y2_kp,s_ram_y2_ki,s_ram_y2_kd,s_ram_y2_apt: std_logic_vector(31 downto 0);
signal s_ram_y3_kp,s_ram_y3_ki,s_ram_y3_kd,s_ram_y3_apt: std_logic_vector(31 downto 0);
signal s_ram_y4_kp,s_ram_y4_ki,s_ram_y4_kd,s_ram_y4_apt: std_logic_vector(31 downto 0);
signal s_ram_y5_kp,s_ram_y5_ki,s_ram_y5_kd,s_ram_y5_apt: std_logic_vector(31 downto 0);
signal s_ram_y6_kp,s_ram_y6_ki,s_ram_y6_kd,s_ram_y6_apt: std_logic_vector(31 downto 0);
signal s_ram_y7_kp,s_ram_y7_ki,s_ram_y7_kd,s_ram_y7_apt: std_logic_vector(31 downto 0);
signal s_ram_y0_dn,s_ram_y1_dn: std_logic;
signal s_ram_individual: std_logic_vector(127 downto 0);
--SelRep
signal s_rd_ram_p1_p2,s_wrt_ram_aux_p1_p2,s_dn_SelRep: std_logic;
signal s_add_ram_aux_p1,s_add_ram_aux_p2: std_logic_vector(6 downto 0);
signal s_add_ram_p0,s_add_ram_p1,s_add_ram_p2,s_add_ram_p3,s_add_ram_p4,
       s_add_ram_p5,s_add_ram_p6,s_add_ram_p7: std_logic_vector(4 downto 0);
signal s_new_kp_p1,s_new_ki_p1,s_new_kd_p1,s_new_kp_p2,
                    s_new_ki_p2,s_new_kd_p2: std_logic_vector(31 downto 0);
signal s_add_ram_p1_adj,s_add_ram_p2_adj: std_logic_vector(6 downto 0);
--ram2
signal s_ram2_y0_kp,s_ram2_y0_ki,s_ram2_y0_kd,s_ram2_y1_kp,s_ram2_y1_ki,
      s_ram2_y1_kd: std_logic_vector(31 downto 0);
signal s_ram2_y0_dn,s_ram2_y1_dn:               std_logic;
signal s_ram2_y0_all_gains,s_ram2_y1_all_gains: std_logic_vector(95 downto 0);

--mx0
signal s_mx0_y:                          std_logic_vector(95 downto 0);
signal s_mx0_y_kp,s_mx0_y_ki,s_mx0_y_kd: std_logic_vector(31 downto 0);
--mx1
signal s_mx1_y:                          std_logic_vector(95 downto 0);
signal s_mx1_y_kp,s_mx1_y_ki,s_mx1_y_kd: std_logic_vector(31 downto 0);
--mx2
signal s_mx2_y: std_logic_vector(6 downto 0);
--mx3
signal s_mx3_y: std_logic_vector(6 downto 0);
--mx4
signal s_mx4_y: std_logic_vector(6 downto 0);
--mx5
signal s_mx5_y: std_logic_vector(6 downto 0);
--mx6
signal s_mx6_y:   std_logic_vector(127 downto 0);
signal s_mx6_y_kp,s_mx6_y_ki,s_mx6_y_kd,
              s_mx6_y_apt: std_logic_vector(31 downto 0);
--mx7
signal s_mx7_y: std_logic_vector(6 downto 0);
--mx8
signal s_mx8_y: std_logic_vector(6 downto 0);
--mx9
```

```
signal s_mx9_y: std_logic_vector(95 downto 0);

-------------------------------------------------------------------------
-------------------------------------------------------------------------
--cont generations
signal s_dn_cont_gen: std_logic;
--new generation
signal s_dn_cont_ngen: std_logic;
signal s_add1_ngen,s_add2_ngen: std_logic_vector(6 downto 0);
--fittest old
signal s_Apt_fittest_individual_old: std_logic_vector(31 downto 0);
signal s_fittest_individual_old_no_apt: std_logic_vector(95 downto 0);
signal s_fittest_individual_old:        std_logic_vector(127 downto 0);


--Better
signal s_mx_best: std_logic;
--and apts
signal s_fncs_apt_dn: std_logic;
--------Unconnected signals
--ram
signal s_ram_x1_kp,s_ram_x1_ki,s_ram_x1_kd,
       s_ram_x1_Apt: std_logic_vector(31 downto 0);
signal s_ram_x1_write: std_logic;

begin
 U0:  GEN1 port map(rst,clk,s_c_fGen,s_gen1_kp1,s_gen1_ki1,s_gen1_kd1,
             s_gen1_kp2,s_gen1_ki2,s_gen1_kd2,s_gen1_get_apt,s_dn_apt,
             s_gen_add1,s_gen_add2,s_gen_gwrite_ram,s_dn_ram,s_gen1_dn);
 U1:  Apt_Fnc  port map(rst,clk,s_c_get_apt,"00110010","0000000100000000",
                    s_mx0_y_kp,s_mx0_y_ki,s_mx0_y_kd,s_kp_ldd_1,s_ki_ldd_1,
                    s_kd_ldd_1,s_fnc_apt_Apt_1,s_fnc_apt1_dn);
                    --referencia formato 8.8
 U2:  Apt_Fnc  port map(rst,clk,s_c_get_apt,"00110010","0000000100000000",
               s_mx1_y_kp,s_mx1_y_ki,s_mx1_y_kd,s_kp_ldd_2,s_ki_ldd_2,
                   s_kd_ldd_2,s_fnc_apt_Apt_2,s_fnc_apt2_dn);
 U3:  RAM1A port map(rst,clk,s_rw_ram1a,s_c_wrt_ram_0,s_c_wrt_ram_1,
            s_c_rd_ram_p1,s_c_rd_ram_p2,s_kp_ldd_1,s_ki_ldd_1,s_kd_ldd_1,
            s_fnc_apt_Apt_1,s_kp_ldd_2,s_ki_ldd_2,s_kd_ldd_2,
            s_fnc_apt_Apt_2,s_mx4_y,s_mx5_y,s_add_ram_p0,s_add_ram_p1,
            s_add_ram_p2,s_add_ram_p3,s_add_ram_p4,s_add_ram_p5,
            s_add_ram_p6,s_add_ram_p7,s_ram_y0_kp,s_ram_y0_ki,s_ram_y0_kd,
            s_ram_y0_apt,s_ram_y1_kp,s_ram_y1_ki,s_ram_y1_kd,
            s_ram_y1_apt,s_ram_y2_kp,s_ram_y2_ki,s_ram_y2_kd,
            s_ram_y2_apt,s_ram_y3_kp,s_ram_y3_ki,s_ram_y3_kd,s_ram_y3_apt,
            s_ram_y4_kp,s_ram_y4_ki,s_ram_y4_kd,s_ram_y4_apt,s_ram_y5_kp,
            s_ram_y5_ki,s_ram_y5_kd,s_ram_y5_apt,s_ram_y6_kp,
            s_ram_y6_ki,s_ram_y6_kd,s_ram_y6_apt,s_ram_y7_kp,
            s_ram_y7_ki,s_ram_y7_kd,s_ram_y7_apt,s_ram_y0_dn,s_ram_y1_dn);


 U4:  SelRep port map (rst,clk,s_rd_ram_p1_p2,s_add_ram_p0,s_add_ram_p1,
             s_add_ram_p2,s_add_ram_p3,s_add_ram_p4,
             s_add_ram_p5,s_add_ram_p6,s_add_ram_p7,
             s_dn_rd_ram_p1_p2,s_mx6_y_kp,s_mx6_y_ki,
             s_mx6_y_kd,s_mx6_y_apt,s_ram_y1_kp,
             s_ram_y1_ki,s_ram_y1_kd,s_ram_y1_apt,
             s_ram_y2_kp,s_ram_y2_ki,s_ram_y2_kd,
             s_ram_y2_apt,s_ram_y3_kp,s_ram_y3_ki,
             s_ram_y3_kd,s_ram_y3_apt,s_ram_y4_kp,
             s_ram_y4_ki,s_ram_y4_kd,s_ram_y4_apt,
             s_ram_y5_kp,s_ram_y5_ki,s_ram_y5_kd,
             s_ram_y5_apt,s_ram_y6_kp,s_ram_y6_ki,
             s_ram_y6_kd,s_ram_y6_apt,s_ram_y7_kp,
             s_ram_y7_ki,s_ram_y7_kd,s_ram_y7_apt,
             s_st_sel_rep,s_wrt_ram_aux_p1_p2,
```

```
            s_dnWraux_p12,s_add_ram_aux_p1,
            s_add_ram_aux_p2,s_new_kp_p1,s_new_ki_p1,
            s_new_kd_p1,s_new_kp_p2,s_new_ki_p2,
            s_new_kd_p2,s_dn_SelRep);


 U5:  RAM2 port map(rst,clk,s_new_kp_p1,s_new_ki_p1,s_new_kd_p1,
                 s_new_kp_p2,s_new_ki_p2,s_new_kd_p2,s_mx7_y,
                 s_mx8_y,s_c_wrt_ram2_1,s_c_wrt_ram2_2,
                 s_c_rd_ram2_1,s_c_rd_ram2_2,s_ram2_y0_kp,
                 s_ram2_y0_ki,s_ram2_y0_kd,s_ram2_y1_kp,
                 s_ram2_y1_ki,s_ram2_y1_kd,s_ram2_y0_dn,
                 s_ram2_y1_dn);
 U6:  fsm_test_GA port map(rst,clk,s_c_get_fittest,s_ram_y0_dn,
                 s_c_wrt_ram_0,s_fncs_apt_dn,s_c_get_apt,
                 s_gen_gwrite_ram,s_gen1_get_apt,s_gen1_dn,
                 s_dn_ram,s_dn_apt,s_c_fGen,start,s_fittest_dn,
                 s_mx_add1,s_c_rd_ram_p1,s_c_rd_ram_p2,
                 s_ram_y1_dn,s_mx_apt,s_rd_ram_p1_p2,
                 s_dn_rd_ram_p1_p2,s_st_sel_rep,s_mx_Fittest,
                 s_dn_SelRep,s_wrt_ram_aux_p1_p2,s_dnWraux_p12,
                 s_c_wrt_ram2_1,s_c_wrt_ram2_2,s_c_rd_ram2_1,
                 s_c_rd_ram2_2,s_ram2_y0_dn,s_ram2_y1_dn,
                 s_dn_cont_gen,s_opc_cont_gen,s_mx_add_r2_p1,
                 s_opc_cont_ngen,s_dn_cont_ngen,
                 s_mx_add_r1p1_apt,s_ld_this_fittest,
                 s_c_wrt_ram_1,dn);
 U7:  pgr_cntr_zero  generic map(8) port map(rst,clk,s_opc_cont_gen,
                                        "11111111",s_dn_cont_gen);
 U8:  Fittest port map(rst,clk,s_kp_ldd_1,s_ki_ldd_1,s_kd_ldd_1,
              s_fnc_apt_Apt_1,s_mx4_y,s_kp_ldd_2,
              s_ki_ldd_2,s_kd_ldd_2,s_fnc_apt_Apt_2,
              s_mx5_y,s_c_get_fittest,s_low_kp,s_low_ki,
              s_low_kd,s_low_apt,s_low_add,s_fittest_dn,
              s_fittest_dn_all);
U9:  new_gen port map(rst,clk,s_opc_cont_ngen,s_add1_ngen,
                      s_add2_ngen,s_dn_cont_ngen);
 U10: Better   port map(s_low_apt,s_Apt_fittest_individual_old,s_mx_best);
 U11: reg_n    generic map(128) port map(rst,clk,s_ld_this_fittest,
               s_fittest_individual,s_fittest_individual_old);

 --mx0
 U12: Mux2_1_n generic map(96)  port map(s_gen1_all_gains1,
               s_ram2_y0_all_gains,s_mx_apt,s_mx0_y);
 --mx1
 U13: Mux2_1_n generic map(96)  port map(s_gen1_all_gains2,
                       s_ram2_y1_all_gains,s_mx_apt,s_mx1_y);
 --mx2
 U14: Mux2_1_n generic map(7)   port map(s_gen_add1,s_add_ram_p1_adj,
                                s_mx_add1,s_mx2_y);
 --mx3
 U15: Mux2_1_n generic map(7)   port map(s_gen_add2,s_add_ram_p2_adj,
                                s_mx_add1,s_mx3_y);
 --mx4
 U16: Mux2_1_n generic map(7)   port map(s_mx2_y,s_add1_ngen,
                                s_mx_add_r2_p1,s_mx4_y);
 --mx5
 U17: Mux2_1_n generic map(7)   port map(s_mx3_y,s_add2_ngen,
                                s_mx_add_r2_p1,s_mx5_y);
 --mx6
 U18: Mux2_1_n generic map(128) port map(s_ram_individual,s_fittest_individual,
                                s_mx_Fittest,s_mx6_y);
 --mx7
 U19: Mux2_1_n generic map(7)   port map(s_add_ram_aux_p1,s_add1_ngen,
```

```
                                s_mx_add_r2_p1,s_mx7_y);
 --mx8
 U20: Mux2_1_n generic map(7)   port map(s_add_ram_aux_p2,s_add2_ngen,
                                s_mx_add_r2_p1,s_mx8_y);
 --mx9
 U21: Mux2_1_n generic map(96)  port map(s_fittest_individual_old_no_apt,
                               s_fittest_individual_no_apt,s_mx_best,s_mx9_y);


 --De/Concatenations
 --Gen1
 s_gen1_all_gains1 <= s_gen1_kp1 & s_gen1_ki1 & s_gen1_kd1;
 s_gen1_all_gains2 <= s_gen1_kp2 & s_gen1_ki2 & s_gen1_kd2;
 --MX0
 s_mx0_y_kp <= s_mx0_y(95 downto 64);
 s_mx0_y_ki <= s_mx0_y(63 downto 32);
 s_mx0_y_kd <= s_mx0_y(31 downto 0);
 --MX1
 s_mx1_y_kp <= s_mx1_y(95 downto 64);
 s_mx1_y_ki <= s_mx1_y(63 downto 32);
 s_mx1_y_kd <= s_mx1_y(31 downto 0);
 --MX6
 s_mx6_y_kp  <= s_mx6_y(127 downto 96);
 s_mx6_y_ki  <= s_mx6_y(95 downto 64);
 s_mx6_y_kd  <= s_mx6_y(63 downto 32);
 s_mx6_y_apt <= s_mx6_y(31 downto 0);
 --RAM
s_ram_individual <= s_ram_y0_kp & s_ram_y0_ki & s_ram_y0_kd & s_ram_y0_apt;
s_ram_x1_write <= '0';

 --RAM2
 s_ram2_y0_all_gains <= s_ram2_y0_kp & s_ram2_y0_ki & s_ram2_y0_kd;
 s_ram2_y1_all_gains <= s_ram2_y1_kp & s_ram2_y1_ki & s_ram2_y1_kd;

 --FITTEST
 s_fittest_individual <= s_low_kp & s_low_ki & s_low_kd & s_low_apt;
 s_fittest_individual_no_apt <= s_low_kp & s_low_ki & s_low_kd;
 --Fittest old
 s_Apt_fittest_individual_old <= s_fittest_individual_old(31 downto 0);
 s_fittest_individual_old_no_apt <= s_fittest_individual_old(127 downto 32);

 --Best individual of genertions so far
 kp <= s_mx9_y(95 downto 64);
 ki <= s_mx9_y(63 downto 32);
 kd <= s_mx9_y(31 downto 0);
 --dn of aptitude functions
 s_fncs_apt_dn <= s_fnc_apt1_dn and s_fnc_apt1_dn;
end estructural;
```

5.4. Código del Algoritmo Genético en VHDL: Secuenciador

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity fsm_test_GA is
 port(
  rst: in std_logic;
  clk: in std_logic;
  -------------------------------
  c_get_fittest:  out std_logic;
  dn_ram0:  in  std_logic;
  c_wrt_ram_0: out std_logic;
  dn_get_apt:  in  std_logic;
  c_get_apt:  out std_logic;
  write_ram:  in  std_logic;
```

```
  get_apt:  in  std_logic;
  dn_fGen:  in  std_logic;
  dn_ram:   out std_logic;
  dn_apt:   out std_logic;
  c_fGen:   out std_logic;
  start:   in  std_logic;
  --------------------------------
  dn_get_fittest:   in  std_logic;
  mx_add1:    out std_logic;
  c_rd_ram_p1:   out std_logic;
  c_rd_ram_p2:   out std_logic;
  dn_ram_1:    in  std_logic;
  mx_apt:     out std_logic;
  rd_ram_p1_p2:   in  std_logic;
  dn_rd_ram_p1_p2:  out std_logic;
  st_sel_rep:    out std_logic;
  --------------------------------
  mx_Fittest:    out std_logic;
  dnSelRep:    in  std_logic;
  Wraux_p12:    in  std_logic;
  dnWraux_p12:   out std_logic;
  c_wrt_ram2_1:  out std_logic;
  c_wrt_ram2_2:  out std_logic;
  c_rd_ram2_1:   out std_logic;
  c_rd_ram2_2:   out std_logic;
  dn_ram2_1:    in  std_logic;
  dn_ram2_2:    in  std_logic;
  dn_cont_gen:   in  std_logic;
  opc_cont_gen:  out std_logic_vector(1 downto 0);
  mx_add_r2_p1:  out std_logic;
  --------------------------------
  opc_cont_ngen:   out std_logic_vector(1 downto 0);
  dn_cont_ngen:    in  std_logic;
  mx_add_r1p1_apt: out std_logic;
  ld_this_fittest: out std_logic;
  c_wrt_ram_1:     out std_logic;
  ram1a_rw:        out std_logic;
  dn:        out std_logic
 );
end fsm_test_GA;

architecture maquina of fsm_test_GA is
signal Qp,Qn: std_logic_vector( 5 downto 0 );
begin
 comb: process(Qp,dn_ram0,dn_get_apt,write_ram,get_apt,
               dn_fGen,start,dn_get_fittest,dn_ram_1,
               rd_ram_p1_p2,dnSelRep,Wraux_p12,
               dn_ram2_1,dn_ram2_2,dn_cont_ngen)
 begin
  case Qp is

   when "000000" =>
    --Salidas

    c_fGen          <= '0';
    dn_apt          <= '0';
    dn_ram          <= '0';
    c_get_apt       <= '0';

    c_wrt_ram_0     <= '0';
    c_wrt_ram_1     <= '0';
    c_get_fittest   <= '0';
    mx_add1         <= '0';
    c_rd_ram_p1     <= '0';

    c_rd_ram_p2     <= '0';
    mx_apt          <= '0';
    dn_rd_ram_p1_p2 <= '0';
    st_sel_rep      <= '0';

    mx_Fittest      <= '0';
    dnWraux_p12     <= '0';
    c_wrt_ram2_1    <= '0';
    c_wrt_ram2_2    <= '0';
```

```
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "10";--LD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "00";--LD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '1';

 --Estado siguiente
 if( start = '0' ) then
  Qn <= Qp;
 else
  Qn <= Qp+1;
 end if;

when "000001" =>
 --Salidas

 c_fGen          <= '1';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 if( get_apt = '0' ) then
  Qn <= Qp;
 else
  Qn <= Qp+1;
 end if;

when "000010" =>
 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '1';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
```

```
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 Qn <= Qp+1;
when "000011" =>
 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 if( dn_get_apt = '0' ) then
  Qn <= Qp;
 else
  Qn <= Qp+1;
 end if;
when "000100" =>
 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '1';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
```

```
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 if( write_ram = '0' ) then
  Qn <= Qp;
 else
  Qn <= Qp+1;
 end if;

when "000101" =>
 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '1';
 c_wrt_ram_1     <= '1';
 c_get_fittest   <= '1';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '1';
 dn              <= '0';

 --Estado siguiente
 Qn <= Qp+1;

when "000110" =>

 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
```

```
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 if( dn_ram0 = '0' ) then
  Qn <= Qp;
 else
  if( dn_ram_1 = '0' )then
   Qn <= Qp;
  else
   if( dn_get_fittest = '0' )then
    Qn <= Qp;
   else
    Qn <= Qp+1;
   end if;
  end if;
 end if;

when "000111" =>

 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '1';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 Qn <= Qp+1;

when "001000" =>

 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
```

```
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 Qn <= Qp+1;
when "001001" =>

 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 if( dn_fGen = '0' ) then
  Qn <= "000001";
 else
  Qn <= Qp+1;
 end if;


when "001010" =>

 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';
```

```
 c_rd_ram_p2     <= '0';
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '1';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '1';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 Qn <= Qp+1;
when "001011" =>

 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '1';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 if( rd_ram_p1_p2 = '0' ) then
  Qn <= Qp;
 else
  Qn <= Qp+1;
 end if;

 when "001100" =>

  --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '1';
```

```
 c_rd_ram_p2     <= '1';
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '1';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 Qn <= Qp+1;
when "001101" =>

 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '0';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '1';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 dn              <= '0';

 --Estado siguiente
 if( dn_ram0 = '0' ) then
  Qn <= Qp;
 else
  if( dn_ram_1 = '0' ) then
   Qn <= Qp;
  else
   Qn <= Qp+1;
  end if;
 end if;
when "001110" =>

 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
```

```
 mx_add1          <= '1';
 c_rd_ram_p1      <= '0';
 c_rd_ram_p2      <= '0';
 mx_apt           <= '0';
 dn_rd_ram_p1_p2  <= '1';
 st_sel_rep       <= '0';
 mx_Fittest       <= '1';
 dnWraux_p12      <= '0';
 c_wrt_ram2_1     <= '0';
 c_wrt_ram2_2     <= '0';
 c_rd_ram2_1      <= '0';
 c_rd_ram2_2      <= '0';
 opc_cont_gen     <= "01";--HD
 mx_add_r2_p1     <= '0';
 opc_cont_ngen    <= "01";--HD
 mx_add_r1p1_apt  <= '0';
 ld_this_fittest  <= '0';--HLD
 ram1a_rw         <= '0';
 dn               <= '0';
 --Estado siguiente
 Qn <= Qp+1;
when "001111" =>
 --Salidas
 c_fGen           <= '0';
 dn_apt           <= '0';
 dn_ram           <= '0';
 c_get_apt        <= '0';
 c_wrt_ram_0      <= '0';
 c_wrt_ram_1      <= '0';
 c_get_fittest    <= '0';
 mx_add1          <= '1';
 c_rd_ram_p1      <= '0';
 c_rd_ram_p2      <= '0';
 mx_apt           <= '1';
 dn_rd_ram_p1_p2  <= '1';
 st_sel_rep       <= '0';
 mx_Fittest       <= '1';
 dnWraux_p12      <= '0';
 c_wrt_ram2_1     <= '0';
 c_wrt_ram2_2     <= '0';
 c_rd_ram2_1      <= '0';
 c_rd_ram2_2      <= '0';
 opc_cont_gen     <= "01";--HD
 mx_add_r2_p1     <= '0';
 opc_cont_ngen    <= "01";--HD
 mx_add_r1p1_apt  <= '0';
 ld_this_fittest  <= '0';--HLD
 ram1a_rw         <= '0';
 dn               <= '0';
 --Estado siguiente
 if(  Wraux_p12 = '0' ) then
  Qn <= Qp;
 else
  Qn <= Qp+1;
 end if;
when "010000" =>
 --Salidas
 c_fGen           <= '0';
 dn_apt           <= '0';
 dn_ram           <= '0';
 c_get_apt        <= '0';
 c_wrt_ram_0      <= '0';
 c_wrt_ram_1      <= '0';
 c_get_fittest    <= '0';
```

```
 mx_add1          <= '1';
 c_rd_ram_p1      <= '0';
 c_rd_ram_p2      <= '0';
 mx_apt           <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep       <= '0';
 mx_Fittest       <= '1';
 dnWraux_p12      <= '0';
 c_wrt_ram2_1     <= '1';
 c_wrt_ram2_2     <= '1';
 c_rd_ram2_1      <= '0';
 c_rd_ram2_2      <= '0';
 opc_cont_gen     <= "01";--HD
 mx_add_r2_p1     <= '0';
 opc_cont_ngen    <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw         <= '0';
 dn               <= '0';
 --Estado siguiente
 Qn <= Qp+1;

when "010001" =>
 --Salidas
 c_fGen           <= '0';
 dn_apt           <= '0';
 dn_ram           <= '0';
 c_get_apt        <= '0';
 c_wrt_ram_0      <= '0';
 c_wrt_ram_1      <= '0';
 c_get_fittest    <= '0';
 mx_add1          <= '1';
 c_rd_ram_p1      <= '0';
 c_rd_ram_p2      <= '0';
 mx_apt           <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep       <= '0';
 mx_Fittest       <= '1';
 dnWraux_p12      <= '0';
 c_wrt_ram2_1     <= '0';
 c_wrt_ram2_2     <= '0';
 c_rd_ram2_1      <= '0';
 c_rd_ram2_2      <= '0';
 opc_cont_gen     <= "01";--HD
 mx_add_r2_p1     <= '0';
 opc_cont_ngen    <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw         <= '0';
 dn               <= '0';
 --Estado siguiente
 if( dn_ram2_1 = '0' ) then
  Qn <= Qp;
 else
  if( dn_ram2_2 = '0' ) then
   Qn <= Qp;
  else
   Qn <= Qp+1;
  end if;
 end if;
when "010010" =>
 --Salidas
 c_fGen           <= '0';
 dn_apt           <= '0';
 dn_ram           <= '0';
 c_get_apt        <= '0';
```

```
 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '1';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';
 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 if(  Wraux_p12 = '0' ) then
  Qn <= Qp;
 else
  Qn <= Qp+1;
 end if;
when "010011" =>
 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';
 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '1';
 c_wrt_ram2_2    <= '1';
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';
 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 Qn <= Qp+1;

when "010100" =>
 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
```

```
 c_get_apt       <= '0';
 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';
 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 if( dn_ram2_1 = '0' ) then
  Qn <= Qp;
 else
  if( dn_ram2_2 = '0' ) then
   Qn <= Qp;
  else
   Qn <= Qp+1;
  end if;
 end if;
when "010101" =>
 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';
 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '1';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';
 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 Qn <= Qp+1;

when "010110" =>   --Espera a leer 8 por segunda vez sin el
```

```
                    --mas apto a la salida del mux
  --Salidas
  c_fGen           <= '0';
  dn_apt           <= '0';
  dn_ram           <= '0';
  c_get_apt        <= '0';
  c_wrt_ram_0      <= '0';
  c_wrt_ram_1      <= '0';
  c_get_fittest    <= '0';
  mx_add1          <= '1';
  c_rd_ram_p1      <= '0';
  c_rd_ram_p2      <= '0';
  mx_apt           <= '0';
  dn_rd_ram_p1_p2  <= '0';
  st_sel_rep       <= '0';
  mx_Fittest       <= '0';
  dnWraux_p12      <= '0';
  c_wrt_ram2_1     <= '0';
  c_wrt_ram2_2     <= '0';
  c_rd_ram2_1      <= '0';
  c_rd_ram2_2      <= '0';
  opc_cont_gen     <= "01";--HD
  mx_add_r2_p1     <= '0';
  opc_cont_ngen    <= "01";--HD
  mx_add_r1p1_apt  <= '0';
  ld_this_fittest  <= '0';--HLD
  ram1a_rw         <= '0';
  dn               <= '0';
  --Estado siguiente
  if( rd_ram_p1_p2 = '0' ) then
   Qn <= Qp;
  else
   Qn <= Qp+1;
  end if;

 when "010111" =>
  --Salidas
  c_fGen           <= '0';
  dn_apt           <= '0';
  dn_ram           <= '0';
  c_get_apt        <= '0';
  c_wrt_ram_0      <= '0';
  c_wrt_ram_1      <= '0';
  c_get_fittest    <= '0';
  mx_add1          <= '1';
  c_rd_ram_p1      <= '1';
  c_rd_ram_p2      <= '1';
  mx_apt           <= '0';
  dn_rd_ram_p1_p2  <= '0';
  st_sel_rep       <= '0';
  mx_Fittest       <= '0';
  dnWraux_p12      <= '0';
  c_wrt_ram2_1     <= '0';
  c_wrt_ram2_2     <= '0';
  c_rd_ram2_1      <= '0';
  c_rd_ram2_2      <= '0';
  opc_cont_gen     <= "01";--HD
  mx_add_r2_p1     <= '0';
  opc_cont_ngen    <= "01";--HD
  mx_add_r1p1_apt  <= '0';
  ld_this_fittest  <= '0';--HLD
  ram1a_rw         <= '0';
  dn               <= '0';
  --Estado siguiente
  Qn <= Qp+1;
```

```
when "011000" =>
 --Salidas
 c_fGen           <= '0';
 dn_apt           <= '0';
 dn_ram           <= '0';
 c_get_apt        <= '0';
 c_wrt_ram_0      <= '0';
 c_wrt_ram_1      <= '0';
 c_get_fittest    <= '0';
 mx_add1          <= '1';
 c_rd_ram_p1      <= '0';
 c_rd_ram_p2      <= '0';
 mx_apt           <= '0';
 dn_rd_ram_p1_p2  <= '0';
 st_sel_rep       <= '0';
 mx_Fittest       <= '0';
 dnWraux_p12      <= '0';
 c_wrt_ram2_1     <= '0';
 c_wrt_ram2_2     <= '0';
 c_rd_ram2_1      <= '0';
 c_rd_ram2_2      <= '0';
 opc_cont_gen     <= "01";--HD
 mx_add_r2_p1     <= '0';
 opc_cont_ngen    <= "01";--HD
 mx_add_r1p1_apt  <= '0';
 ld_this_fittest  <= '0';--HLD
 ram1a_rw         <= '0';
 dn               <= '0';
 --Estado siguiente
 if( dn_ram0 = '0' ) then
  Qn <= Qp;
 else
  if( dn_ram_1 = '0' ) then
   Qn <= Qp;
  else
   Qn <= Qp+1;
  end if;
 end if;
when "011001" =>
 --Salidas
 c_fGen           <= '0';
 dn_apt           <= '0';
 dn_ram           <= '0';
 c_get_apt        <= '0';
 c_wrt_ram_0      <= '0';
 c_wrt_ram_1      <= '0';
 c_get_fittest    <= '0';
 mx_add1          <= '1';
 c_rd_ram_p1      <= '0';
 c_rd_ram_p2      <= '0';
 mx_apt           <= '0';
 dn_rd_ram_p1_p2  <= '1';
 st_sel_rep       <= '0';
 mx_Fittest       <= '0';
 dnWraux_p12      <= '0';
 c_wrt_ram2_1     <= '0';
 c_wrt_ram2_2     <= '0';
 c_rd_ram2_1      <= '0';
 c_rd_ram2_2      <= '0';
 opc_cont_gen     <= "01";--HD
 mx_add_r2_p1     <= '0';
 opc_cont_ngen    <= "01";--HD
 mx_add_r1p1_apt  <= '0';
 ld_this_fittest  <= '0';--HLD
```

```
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 Qn <= Qp+1;
when "011010" =>

 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '1';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 if(  Wraux_p12 = '0' ) then
  Qn <= Qp;
 else
  Qn <= Qp+1;
 end if;
when "011011" =>

 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '1';
 c_wrt_ram2_2    <= '1';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
```

```
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 Qn <= Qp+1;

when "011100" =>

 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 if( dn_ram2_1 = '0' ) then
  Qn <= Qp;
 else
  if( dn_ram2_2 = '0' ) then
   Qn <= Qp;
  else
   Qn <= Qp+1;
  end if;
 end if;

 when "011101" =>

 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '1';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
```

```
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';
 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 if(  Wraux_p12 = '0' ) then
  Qn <= Qp;
 else
  Qn <= Qp+1;
 end if;
when "011110" =>
 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';
 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '1';
 c_wrt_ram2_2    <= '1';
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';
 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 Qn <= Qp+1;

when "011111" =>
 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';
 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
 c_rd_ram2_1     <= '0';
```

```
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 if( dn_ram2_1 = '0' ) then
  Qn <= Qp;
 else
  if( dn_ram2_2 = '0' ) then
   Qn <= Qp;
  else
   Qn <= Qp+1;
  end if;
 end if;

when "100000" =>

 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '1';
 c_rd_ram_p1     <= '0';

 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';

 mx_Fittest      <= '0';
 dnWraux_p12     <= '1';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';

 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '0';

 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '0';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';

 --Estado siguiente
 if( dnSelRep = '0' ) then
  if( rd_ram_p1_p2 = '0' ) then
   Qn <= Qp;
  else
   Qn <= "010111";
  end if;
 else
  Qn <= Qp+1;
 end if;

when "100001" =>

 --Salidas

 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';

 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
```

```
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
 c_rd_ram2_1     <= '1';
 c_rd_ram2_2     <= '1';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '1';
 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '1';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 Qn <= Qp+1;
 ---Se comienza la obtencion de aptitud de la nueva generacion

when "100010" =>
 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';
 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '1';
 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '1';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 if( dn_ram2_1 = '0' ) then
  Qn <= Qp;
 else
  if( dn_ram2_2 = '0' ) then
   Qn <= Qp;
  else
   Qn <= Qp+1;
  end if;
 end if;
when "100011" =>
 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
```

```
 dn_ram          <= '0';
 c_get_apt       <= '1';
 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '1';
 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '1';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 Qn <= Qp+1;

when "100100" =>
 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';
 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '1';
 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '1';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 if( dn_get_apt = '0' ) then
  Qn <= Qp;
 else
  Qn <= Qp+1;
 end if;
when "100101" =>
 --Salidas
 c_fGen          <= '0';
```

```
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';
 c_wrt_ram_0     <= '1';
 c_wrt_ram_1     <= '1';
 c_get_fittest   <= '1';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '1';
 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '1';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 Qn <= Qp+1;

when "100110" =>
 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';
 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '1';
 opc_cont_ngen   <= "01";--HD
 mx_add_r1p1_apt <= '1';
 ld_this_fittest <= '0';--HLD
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 if( dn_ram0 = '0' ) then
  Qn <= Qp;
 else
  if( dn_ram_1 = '0' ) then
   Qn <= Qp;
  else
   if( dn_get_fittest = '0' ) then
    Qn <= Qp;
   else
```

```
    if( dn_cont_ngen = '0' ) then
     Qn <= Qp+1;
    else
     if( dn_cont_gen = '0' ) then
      Qn <= "101000";--
     else
      Qn <= "000000";
     end if;
    end if;
   end if;
  end if;
 end if;
when "100111" =>
 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';
 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "01";--HD
 mx_add_r2_p1    <= '1';
 opc_cont_ngen   <= "11";--DEC
 mx_add_r1p1_apt <= '1';
 ld_this_fittest <= '1';--HLD-----------------
 ram1a_rw        <= '0';
 dn              <= '0';
 --Estado siguiente
 Qn <= "011111";

when others =>
 --Salidas
 c_fGen          <= '0';
 dn_apt          <= '0';
 dn_ram          <= '0';
 c_get_apt       <= '0';
 c_wrt_ram_0     <= '0';
 c_wrt_ram_1     <= '0';
 c_get_fittest   <= '0';
 mx_add1         <= '0';
 c_rd_ram_p1     <= '0';
 c_rd_ram_p2     <= '0';
 mx_apt          <= '1';
 dn_rd_ram_p1_p2 <= '0';
 st_sel_rep      <= '0';
 mx_Fittest      <= '0';
 dnWraux_p12     <= '0';
 c_wrt_ram2_1    <= '0';
 c_wrt_ram2_2    <= '0';
 c_rd_ram2_1     <= '0';
 c_rd_ram2_2     <= '0';
 opc_cont_gen    <= "11";--DEC
```

```
    mx_add_r2_p1    <= '1';
    opc_cont_ngen   <= "11";--DEC
    mx_add_r1p1_apt <= '1';
    ld_this_fittest <= '1';--HLD---------------------
    ram1a_rw        <= '0';
    dn              <= '0';
    --Estado siguiente
    Qn <= "001001";
   end case;
 end process comb;
 sec: process(rst,clk)
 begin
  if( rst = '1' ) then
   Qp <= (others => '0');
  elsif(clk'event and clk = '1') then
   Qp <= Qn;
  end if;
 end process sec;
end;
```

LITERATURA CITADA

Z. Affi, B. EL-Kribi, and L. Romdhane. Advanced mechatronic design using a multi-objective genetic algorithm optimization of a motor-driven four-bar system. *Mechatronics*, 2007.

E. Akın, M. Kaya, and M. Karakose. A robust integrator algorithm with genetic based fuzzy controller feedback for direct vector control. *Computers and Electrical Engineering*, 29:379–394, 2003.

E. Alfaro-Cid, E. McGookin, and D. Murray-Smith. A comparative study of genetic operators for controller parameter optimisation. *Control Engineering Practice*, (17): 185– 197, 2009.

A. Altınten, F. Ketevanliolu, S. Erdoan, H. Hapoglu, and M. Alpbaz. Self-tuning pid control of jacketed batch polystyrene reactor using genetic algorithm. *Chemical Engineering*, (138):490–497, 2008.

K. J. Aström and T. Hägglund. The future of pid control. *Control Engineering Practice*, (9):1163–1165, 2001.

K. J. Aström and T. Hägglund. *Advanced PID Control*. Instrument Society of America, 2006.

K. J. Aström and T. Hägglund. *PID Controllers: Theory, Design and Tuning*. Instrument Society of America, 1995.

V. Bobál, J. Böhm, J. Fessl, and J. Machácek. *Digital Self-tuning Controllers. Algorithms, Implementation and Applications.* Springer-Verlag, 2005.

W.-D. Chang. A multi-crossover genetic approach to multivariable pid controllers tuning. *Expert Systems with Applications*, (33):620–626, 2007.

C.-K. Chen, H.-H. Kuo, J.-J. Yan, and NewAuthor3. Ga-based pid active queue management control design for a class of tcp communication networks. *Expert Systems with Applications*, 36:1903–1913, 2009.

P. Dionisio and P. Joao. Genetic algorithm based system identification and pid tuning for optimum adaptive control. In *International Conference on Advanced Intelligent Mechatronics*, 2005.

M. W. Iruthayarajan and S. Baskar. Evolutionary algorithms based design of multivariable pid controller. *Expert Systems with Applications*, (36):9159–9167, 2008.

A. Jamali, A. Hajiloo, and N.Ñariman-zadeh. Reliability-based robust pareto design of linear state feedback controllers using a multi-objective uniform-diversity genetic algorithm (muga). *Expert Systems with Applications*, 2009.

Z. Jinhua, Z. Jian, D. Haifeng, and W. Sun'an. Self-organizing genetic algorithm based tuning of pid controllers. *Information Sciences*, (179):1007–1018, 2009.

J.-G. Juang, H.-K. Chiou, and L.-H. Chien. Analysis and comparison of aircraft landing control using recurrent neural networks and genetic algorithms approaches. *Neurocomputing*, pages 3224–3238, 2008.

M. Kannan. *Digital Control*. Wiley, 2007.

O. Katsuhiko. *Discrete-Time Control Systems*. Prentice Hall, Inc, second edition, 1995.

D. H. Kim, A. Abraham, and J. H. Cho. A hybrid genetic algorithm and bacterial foraging approach for global optimization. *Information Sciences*, 2007.

G. Lin and G. Liu. Tuning pid controller using adaptive genetic algorithms. In *The 5th International Conference on Computer Science & Education*. IEEE, 2010.

H.-C. Lu, J.-C. Chang, and M.-F. Yeh. Design and analysis of direct-action cmac pid controller. *Neurocomputing*, (70):2615–2625, 2007.

A. J. A. Nazir, Gautham, R. Surajan, and Binu.L.S. A simplified genetic algorithm for online tuning of pid controller in labview. In *World Congress on Nature & Biologically Inspired Computing*. IEEE, 2009.

A. O'Dwyer. *Handbook of PI and PID controller tuning rules*. Imperiall College Press, second edition, 2006.

S.-K. Oh, S.-H. Jung, and W. Pedrycz. Design of optimized fuzzy cascade controllers by means of hierarchical fair competition-based genetic algorithms. *Expert Systems with Applications*, (36):11641–11651, 2009.

Z. Ping, M. Yuan, and H. Wang. Self-tuning pid based on adaptive genetic algorithms with the application of activated sludge aeration process. In *Proceedings of the 6th World Congress on Intelligent Control and Automation*. IEEE, 2006.

S.N.Sivanandam and S.N.Deepa. *Introduction to Genetic Algorithms*. Springer-Verlag, 2008.

K. Valarmathi, D. Devaraj, and T. Radhakrishnan. Real-coded genetic algorithm for system identification and controller tuning. *Applied Mathematical Modelling*, (33): 3392–3401, 2009.

T. Xu, H. Wei, and G. Hu. Study on continuous network design problem using simulated annealing and genetic algorithm. *Expert Systems with Applications*, 2009.

M. Yasue, Y. Toru, and K. Masahiro. A design of self-tuning pid controllers using a genetic algorithm. In *Proceedings of The American Control Conference*, 1999.

C.-C. Yu. Autotuning of pid controllers.a relay feedback approach. *Springer*, 2006.