Capítulo 3

Herramientas de CAD: Flujo de diseño

En este capítulo se resumen, en el primer apartado, las técnicas y las correspondientes herramientas de CAD que van a ser utilizadas para extraer bases de reglas difusas a partir de datos numéricos. A continuación, el segundo apartado resume las técnicas que se han empleado para simplificar bases de reglas. El tercer apartado describe la aproximación que nosotros hemos seguido para facilitar la implementación hardware de las bases de reglas. Un aspecto clave de nuestra aproximación es el uso de una modificación del método de *Quine-McCluskey* para simplificar sistemas lógicos no sólo booleanos sino también difusos. Después se describe la arquitectura de sistemas difusos basadas en reglas activas. El quinto apartado versa sobre la síntesis de sistemas difusos con *System Generator*. Por último, se describe el flujo de diseño que se va a seguir en este Proyecto Fin de Carrera.

3.1. Extracción de bases de reglas a partir de datos numéricos

Las técnicas dedicadas al análisis y extracción de información a partir de datos numéricos son cada vez más necesarias porque son una forma de reducir y compactar información. Desde el punto de vista hardware, la realización de un controlador que respondiera con unas determinadas salidas frente a unas determinadas entradas siempre podría hacerse mediante una tabla de búsqueda (*lookup table*), y, de hecho, hay varios trabajos en los que así se hace. Sin embargo, esta solución sólo tiene sentido para pocos datos (de otro modo, se dispara la complejidad de la tabla) y si no se requiere excesiva continuidad entre los valores que toma la variable de salida. En cualquier otro caso, siempre es mejor recurrir a interpoladores que reproduzcan los datos usando menos recursos de memoria. Dentro de los interpoladores, cobran gran interés los basados en lógica difusa, debido a su capacidad de extraer conocimiento lingüístico de fácil comprensión por personas no expertas.

Debido a la importancia de esta rama, se han desarrollado numerosas técnicas dedicadas a la extracción de bases de reglas difusas a partir de datos numéricos, dentro de las cuales se pueden distinguir las basadas en métodos de agrupamiento o clustering y las basadas en particiones de tipo rejilla (o *grid*) de los universos de discurso de las variables del problema. Todas estas técnicas pueden aplicarse usando la herramienta *Xfdm* de *Xfuzzy*.

3.1.1. Métodos de clustering

Las técnicas de *clustering* se basan en la idea de que, dada una serie de datos (o puntos, si los representamos gráficamente) podemos realizar una división entre ellos, es decir, que podemos realizar una serie de "*agrupamientos*" o colecciones de datos dentro de la nube de puntos que representan los datos. Los datos que están en el mismo grupo deben tener un cierto grado de cercanía o de características similares. A cada uno de esto grupos se les denomina "*cluster*".

Así, un dato cualquiera puede pertenecer bien a un solo *cluster* (por lo que tendría un grado de pertenencia a ese *cluster* de 1) o a varios de ellos (en cuyo caso el grado de pertenencia a ese *cluster* variaría de 0 a 1). Cuando imponemos la necesidad de que un dato o punto sólo pueda pertenecer a un cluster estaríamos hablando de un hard *clustering*. En cambio, si permitimos que un mismo dato o punto pueda pertenecer a varios clusters estaríamos hablando de un *soft* o *fuzzy clustering*. En el caso de *fuzzy* clustering, se puede cuantificar cuán difuso es cada *cluster*. Cuanto más difusos son los *clusters*, cada dato o punto puede pertenecer a más *clusters* a la vez.

La facilidad que nos ofrece este tipo de técnicas es poder trabajar con unos pocos datos en vez de con un gran número de datos, ya que trabajaríamos con los datos prototipos de los *clusters*, que serían el centro de cada *cluster*. Un centro de *cluster* es aquel dato que representa a todos los datos que están incluidos en un mismo *cluster*. Así, en vez de generar un sistema para una serie de puntos, nuestro problema consiste ahora en generar un sistema para una serie de *clusters*.

Cada cluster que se obtiene por una técnica de *clustering* se transforma en una regla mediante su proyección en cada dimensión de las variables de entrada [35]. Por tanto, toda la descripción de la base de reglas se obtiene simultáneamente. La extracción de información mediante *clustering* tiene la ventaja de conseguir pocas reglas, eliminando así el problema denominado "la maldición de la dimensionalidad" que aparece con las técnicas basadas en una partición de tipo grid (*Grid Partitioning*), que consiste en que el número de reglas aumenta exponencialmente con el número de variables de entrada. Como contrapartida, las reglas obtenidas a partir de la proyección de los *clusters* no suelen tener significado lingüístico. Diversas aproximaciones han sido propuestas en la literatura para incrementar el significado lingüístico de estas bases de reglas. Una de ellas es el uso de modificadores lingüísticos, que permiten reducir tanto el número de funciones de pertenencia (muchas de ellas pueden ser obtenidas como modificaciones de otras) como el de las reglas (algunas de ellas pueden combinarse en otra más genérica [36]. Otros autores se centran en simplificar las funciones de pertenencia obtenidas para conseguir un sistema difuso más transparente e interpretable [37].

Las técnicas de *clustering* pueden obtener un número de *clusters* fijado de antemano o ir incrementando este número para que se ajuste mejor al modelo planteado por los datos. Así podemos dividir las técnicas de clustering en técnicas *fixed clustering* (para el caso de un número de *clusters* prefijado) y técnicas *incremental clustering* (para cuando el número de *clusters* varía para conseguir un determinado grado de adaptación a la información numérica).

3.1.1.1. Técnicas de *Fixed Clustering*

Los algoritmos de *clustering* persiguen la minimización de una función objetivo. La función objetivo más utilizada para obtener un número prefijado de c *clusters* es la siguiente:

$$J[U,V;X] = \sum_{i=1}^{c}\sum_{k=1}^{n} u_{ik}^{m} d_{ik}^{2}\left(\vec{x}_k,\vec{v}_i\right) \qquad \textbf{Ecuación 26}$$

Donde:

$V = \left(\vec{v}_1,\vec{v}_2,\ldots,\vec{v}_c\right)$, es la matriz formada por los centros de los c *clusters*. A su vez, cada centro de *cluster* es un vector de p elementos.

- X es la matriz formada por los datos numéricos de partida.
- U es la matriz de grados de pertenencia ($u_{ik} \in [0,1]$). u_{ik} es el grado de pertenencia del dato $\vec{x}_k$ al cluster i.
- $d_{ik}\left(\vec{x}_k,\vec{v}_i\right)$ es la distancia entre el dato $\vec{x}_k$ y el prototipo $\vec{v}_i$.

m es un índice que indica lo difuso que es el *cluster*. Cuando m tiende a 1 los *clusters* tienden a ser *hard* o no-difusos porque entonces los u_{ik} valen 1 ó 0. Mientras que si m tiende a infinito, los u_{ik} tienden a $1/c$, de forma que todos los datos pertenecen con el mismo grado a todos los *clusters*, es decir, que los *clusters* son totalmente difusos. Usualmente se elige $m = 2$ para un *clustering* difuso.

Derivando la función objetivo J respecto a u_{ik} y $\vec{v}_i$ e imponiendo que tales derivadas se anulen, para llegar a un mínimo local, se obtiene que:

$$\vec{v}_i = \frac{\sum_{k=1}^{n} (u_{ik})^m \cdot \vec{x}_k}{\sum_{k=1}^{n} (u_{ik})^m} \quad \textbf{y} \quad u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{d^2_{ik}}{d^2_{jk}} \right)^{1/m-1}} \qquad \textbf{Ecuación 27}$$

Las técnicas de *fixed clustering* parten de unos prototipos $\vec{v}_i$ inicializados aleatoriamente o elegidos en base a unos determinados criterios. A partir de esos $\vec{v}_i$ se calculan los u_{ik} de acuerdo con la *Ecuación 27*. Obtenidos los u_{ik} se vuelven a recalcular los $\vec{v}_i$ de acuerdo con la *Ecuación 27*. Y así sucesivamente hasta que la diferencia entre los nuevos $\vec{v}_i$ y los antiguos sea inferior a un valor que fija la condición de parada del algoritmo.

Tres algoritmos muy conocidos que pertenecen a estas técnicas de *fixed clustering* son el algoritmo *Fuzzy C-Means* [38], el de *Gustafson-Kessel* [39] y el de *Gath-Geva* [40].

El método de *Gath-Geva*, como el de *Gustafson-Kessel*, es preferible cuando los *clusters* tienen forma de hiperelipsoides. Mientras que el algoritmo de *Gustafson-Kessel* no funciona bien con *clusters* de distintos tamaños y orientaciones, el de *Gath-Geva* sí porque puede variar el tamaño de los *clusters*, adaptándose a ellos. En cualquier caso, ambos algoritmos fallan cuando los datos están perfectamente alineados, ya que las matrices asociadas a los clusters tienen determinante cero, lo que impide que se puedan invertir.

3.1.1.2. Técnicas de Incremental Clustering

Para solucionar este tipo de problemas, *Stephen L. Chiu* [41] propuso la técnica *Subtractive Clustering* o *Incremental Clustering*. Esta técnica es una modificación del método de estimación de *clusters* de *Yager y Filev* [42], en la cual se introducía por primera vez la idea de centro de *cluster* potencial o simplemente cl*u*ster potencial. La técnica de *Yager y Filev* partía de una división tipo rejilla o *grid* del espacio de datos. Computaba el valor potencial para cada punto del *grid*, calculando la distancia existente entre un punto del *grid* a los demás. Un punto del *grid* con muchos datos del problema cercanos tendría un potencial alto. El punto del *grid* con el mayor potencial sería elegido como el primer centro de *cluster*. La idea principal en este método es que una vez elegido un centro de *cluster*, los potenciales de todos los demás puntos del *grid* son reducidos acorde a su distancia a este nuevo centro del *cluster*; así los puntos del *grid* cercanos al primer centro de *cluster* verán su potencial altamente reducido. El siguiente centro de *cluster* sería entonces situado en el punto del campo con el siguiente potencial mayor, y así de manera iterativa. Aunque el procedimiento es sencillo y efectivo, tiene el problema de que la computación crece exponencialmente según la dimensión del problema (maldición de la dimensionalidad) como ocurre en todos los problemas que emplean una partición tipo *grid*.

La técnica *Incremental Clustering* evita este problema considerando cada dato del sistema y no cada punto del *grid* como un centro de *cluster* potencial. Usando esta idea, el número de "*puntos del campo*" efectivos a evaluar se simplifica al número de datos del sistema planteado, independientemente de la dimensión del problema. Así, ahora se estudia la distancia de cada dato a los demás. El dato que tuviera más datos cercanos, sería el que tendría un valor potencial más alto, por lo que se convertiría en

el primer centro de *cluster* potencial. Una vez seleccionado el primer centro de *cluster*, todos los datos cercanos a éste disminuirían su valor potencial de ser centros; calculando de nuevo la distancia de un dato a los demás. Esto se realizaría iterativamente hasta que el número de *clusters* sea el impuesto como límite en el algoritmo.

Este método, por lo tanto, nos puede proporcionar cuántos y cuáles son los centros de *clusters* con mayor potencialidad de serlos, para una serie de datos dados, lo cual se podría también aprovechar para realizar una inicialización más correcta de los algoritmos *Fixed Clustering*.

3.1.2. Métodos de tipo *grid*

Los métodos de tipo *rejilla* o *grid* generan una partición de los espacios de las entradas y las salidas antes de crear la base de reglas. Las reglas se obtienen seleccionando las etiquetas más adecuadas para las entradas y las salidas respecto a la base de datos numéricos. Estos métodos consiguen reglas con mucho más significado lingüístico, pero con el coste de obtener un número de reglas mucho mayor. Algunas soluciones reportadas en la literatura para evitar este problema han sido seleccionar las reglas más significativas, como proponen *Nauck* en [43] y *Senhadji* en [44].

El paso de los datos de entrada-salida a un sistema difuso, se puede hacer de una manera directa mediante el algoritmo *Fixed Grid Partitioning* (o algoritmo de *Wang–Mendel*), o de manera iterativa con el algoritmo *Incremental Grid Partitioning* (o de *Higgins–Goodman*), que pasamos a describir a continuación.

3.1.2.1. Algoritmo *Fixed Grid Partitioning* (algoritmo de *Wang y Mendel*)

El algoritmo de *Wang y Mendel* [45] es una de las técnicas más conocidas para construir un sistema de lógica difusa mediante el estudio de un conjunto de ejemplos o datos, y de manera opcional, de la experiencia aportada por un experto humano.

El primer paso de este algoritmo es definir una estructura *grid* para las n variables de entrada, seleccionándose el tipo de funciones de pertenencia (triangulares, gaussianas, …) así como el número de etiquetas en el que se particionará cada universo de discurso. A continuación, se estudia para cada patrón de datos del tipo $(x_{i1},...,x_{ip-1},y_i)$ cuáles son las etiquetas de cada universo de discurso a las que el patrón tiene un mayor grado de pertenencia. A partir de estos datos se pueden generar las reglas difusas.

Puesto que el número de patrones de datos es grande, y cada patrón genera una regla, es bastante probable que se generen reglas contradictorias, es decir, reglas con el mismo antecedente pero distinto consecuente. Para resolver dicho conflicto, a cada regla generada se le asigna un grado, el cual se calcula como el producto de los grados de pertenencia de las variables de entrada y salida como se señala en la *Ecuación 28*:

$$D = \prod_{i=1}^{n} \mu_{IF}(x_i) \cdot \mu_{THEN}(y) \qquad \textbf{Ecuación 28}$$

Una vez calculado el grado de todas las reglas, si en el conjunto de reglas hay más de una regla con los mismos antecedentes, se eliminan las que tengan menor grado, resolviendo así los conflictos que se puedan dar.

3.1.2.2. Algoritmo *Incremental Grid Partitioning*

Como acabamos de ver, el algoritmo de *Wang y Mendel* obtiene un sistema difuso cuya partición es fija. El algoritmo *Incremental Grid Partitioning* [46] va un paso más allá, realizando una partición de los universos de discurso de manera gradual, comprobando en cada nueva partición el grado de acercamiento al modelo que estamos estudiando. De esta forma se consigue un particionado gradual o incremental.

El sistema difuso generado por el algoritmo *Incremental Grid Partitioning* es de tipo singleton, de modo que utiliza como método de defuzzificación el *Fuzzy Mean* y la salida del sistema es de la forma de la *Ecuación 38*:

$$\hat{y} = \frac{\sum_i \alpha_i \cdot c_i}{\sum_r \alpha_i} \qquad \textbf{Ecuación 29}$$

Dentro de este algoritmo, pueden encontrarse dos variantes:

- *Incremental Grid Partitioning* sin aprendizaje de los *singletons*.
- *Incremental Grid Partitioning* con aprendizaje de los *singletons*.

Mientras que la primera es el método que hemos comentado anteriormente, la segunda consiste en aplicar, cada vez que se crea una nueva partición, un ajuste de los valores de los consecuentes mediante aprendizaje supervisado. Aunque pueda parecer a primera vista que esto sería siempre lo idóneo, no es así de forma práctica, ya que, a veces, el introducir aprendizaje en los pasos intermedios de generación de determinados sistemas nos lleva a caminos sin salida y/o a soluciones con mayor grado de cómputo y resultados incluso peores. Ejecutar el aprendizaje sólo al final sí que es siempre es ventajoso, como comentamos a continuación.

3.1.3. Ajuste fino a los datos numéricos

Los sistemas difusos que se generan con todas las técnicas de extracción de reglas comentadas anteriormente son de tipo *singleton* (*Takagi-Sugeno* de orden 0) o *Takagi-Sugeno* de orden 1. Por lo tanto, la dependencia de la salida frente a los valores asociados con los consecuentes es lineal, con lo que el error cuadrático medio tiene una forma de hiperparaboloide respecto a ellos. No hay, por tanto, mínimos locales sino un solo conjunto de esos valores que aseguran el valor mínimo global del error cuadrático medio. Con lo cual, es aconsejable como paso final del proceso de extracción de reglas aplicar cualquier método de aprendizaje supervisado sobre los consecuentes.

En este proyecto se va a emplear un método de optimización numérica de tipo *Quasi-Newton*, en concreto el algoritmo de *Levenberg-Marquardt*, ya que está demostrado (experimentalmente) que encuentra ese mínimo global en muy pocas iteraciones y porque calcula un valor muy eficiente para la velocidad de aprendizaje, sin tener que calcular la matriz *Hessiana*. Esta técnica está integrada en la herramienta *Xfsl* de *Xfuzzy*.

3.2. Simplificación de sistemas difusos

Es habitual que un sistema difuso emplee funciones de pertenencia así como reglas 'si-entonces' que pueden reducirse para obtener un sistema con similar, incluso mejor comportamiento. Esto sucede, por ejemplo, cuando extraemos una base de reglas a partir de datos numéricos usando técnicas de clustering. Es habitual que, tras la proyección de los clusters, obtengamos funciones de pertenencia como las indicadas en la *Figura 13a*. Esto sucede también cuando aplicamos aprendizaje supervisado a los

consecuentes de las reglas, como se ilustra en la *Figura 13b*, hayamos obtenido la base de reglas a partir de conocimiento heurístico o numérico. Y no sólo aparece esta redundancia de información en las funciones de pertenencia empleadas sino también en las reglas. Incluso empleando conocimiento heurístico es fácil expresar reglas lingüísticas que son redundantes. Conseguir sistemas difusos simples es muy importante para nuestros dos objetivos: asegurar e incluso aumentar el significado lingüístico del sistema difuso y facilitar su implementación *hardware*. A continuación, resumimos las técnicas que más se han empleado para simplificar funciones de pertenencia y reglas, que se pueden aplicar con la herramienta *Xfsp* de *Xfuzzy*.

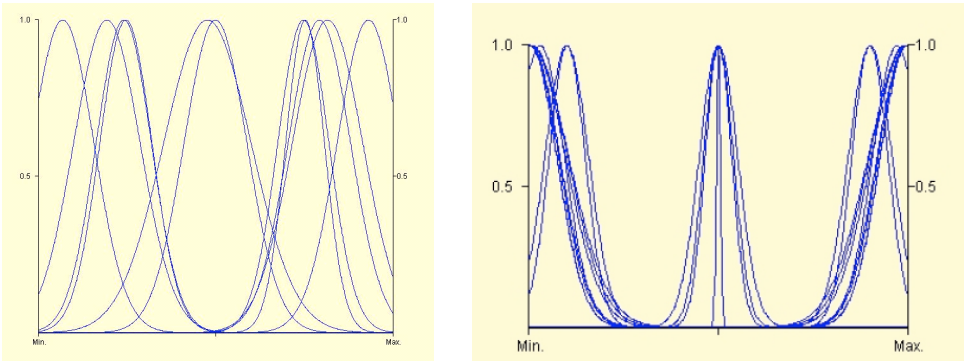

Figura 13. Ejemplos de funciones de pertenencia que presentan claras redundancias

3.2.1. Simplificación de funciones de pertenencia

La existencia de funciones de pertenencia muy parecidas y claramente simplificables en alguna de las variables de un sistema difuso conlleva la utilización de etiquetas lingüísticas innecesarias, pues denotan situaciones semánticamente equivalentes, como bien puede verse en la *Figura 13*. Por lo tanto, dificultan la interpretabilidad del sistema y, potencialmente, pueden también causar la existencia de reglas redundantes.

Los métodos más ampliamente usados para simplificar funciones de pertenencia son la simplificación por clustering, por similitud y por purga.

La simplificación por clustering busca un número reducido de funciones de pertenencia prototipo con los que representar todas las funciones originales. En general, conviene aplicar un algoritmo de clustering hard o no difuso (usualmente el *Hard C-Means*) porque cada función de pertenencia original será remplazada por una sola de las funciones prototipo encontradas. Este algoritmo aplica la minimización de una función objetivo como la *Ecuación 26*, tal y como vimos anteriormente. Los *clusters* se buscan en el espacio formado por los parámetros que definen las funciones de pertenencia. Por ejemplo, las funciones gaussianas se definen mediante sus centros y anchuras.

Otra forma de simplificar funciones de pertenencia es por similitud. Definimos el concepto de similitud entre dos conjuntos difusos como el grado en el que dichos conjuntos difusos son iguales. Esta definición está relacionada con los conceptos que representan los conjuntos difusos. Consideremos los conjuntos difusos A y B de la *Figura 14a*. Tienen la misma forma, pero claramente se observa que representan conceptos distintos (un valor bajo y un valor alto de x, respectivamente). Su grado de igualdad es cero y son considerados distintos. Por otro lado, de los dos conjuntos difusos C y D de la *Figura14b* puede decirse que tienen un alto grado de igualdad, incluso aunque tengan forma diferente. Representan conceptos compatibles y son considerados similares.

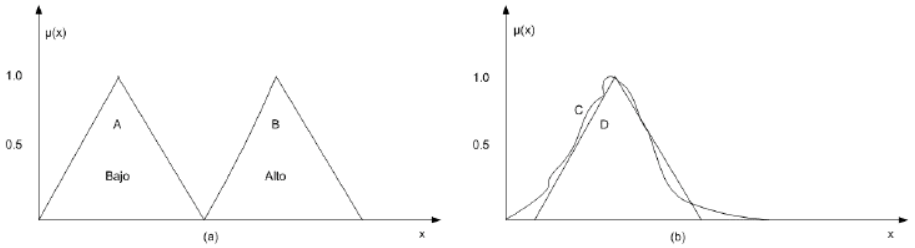

Figura 14. Grado de similitud: (a) Dos conjuntos difusos sin ningún grado de igualdad (b) Conjuntos difusos solapados con un alto grado de igualdad

La simplificación por similitud consiste en identificar los conjuntos difusos que tengan un grado de similitud que está por encima de un valor umbral y sustituirlos por un conjunto difuso común representativo de los originales. El proceso termina cuando no pueden unirse más funciones. Mientras el umbral sea más bajo, mayor será el número de funciones unidas [47].

Por último, la simplificación por purga se trata de una simplificación trivial. Debido a procesos de simplificación de reglas, puede ocurrir que distintas funciones de pertenencia se vuelvan innecesarias, debido a que no son utilizadas en ninguna de las reglas de las bases de reglas del sistema. La simplificación por purga se limita a eliminar dichas funciones de pertenencia.

3.2.2. Simplificación de reglas

En el campo del diseño de sistemas difusos se están llevando a cabo enormes esfuerzos (especialmente en lo que concierne a algoritmos adaptativos de aprendizaje de reglas) para conseguir el mínimo conjunto de reglas difusas. En [48], el compromiso entre un conjunto reducido de reglas y uno adecuado está inscrito en el contexto de la teoría de la información. Para aplicar ese enfoque en el contexto de la lógica difusa, se tiene que asumir la existencia de una expresión de coste mínimo para cada conjunto de reglas candidato.

Es habitual que los procesos de ajuste y simplificación de funciones de pertenencia den lugar a reglas similares que pueden ser eliminadas a continuación de la base de reglas. Las técnicas comúnmente empleadas para ello son las denominadas técnicas '*de podado*'. Consisten en llevar a cabo el tratamiento de una base de reglas para que, dado un conjunto de datos de entrada, determinar el grado de activación de las reglas para esas entradas y poder eliminar las n peores reglas, seleccionar las n mejores o bien imponer un umbral fijo que deba ser superado por el grado de activación de cada regla para que ésta permanezca en la base de reglas.

3.3. Aproximación para facilitar la implementación *hardware*

La técnica que empleamos en este Proyecto Fin de Carrera consiste en partir de una base de reglas tipo *Takagi-Sugeno* (de orden cero) que emplee particiones granulares de las variables del problema y, por tanto, sea lingüísticamente interpretable. Para facilitar la implementación hardware, las particiones de las variables de entrada deben hacerse con triángulos que se solapen hasta el grado de pertenencia 0.5. Este tipo de bases de reglas se obtiene a partir de datos numéricos obtenidos de controladores *PWA* explícitos diseñados conforme a la técnica de *A. Bemporad* (resumida en el apartado 1.3). Para ello empleamos técnicas de tipo *grid,* en concreto, la técnica de *Wang y Mendel*. A continuación, se ajustan los valores asociados con los consecuentes para minimizar el error cuadrático medio. Y se simplifican las funciones de pertenencia para las variables de salida mediante técnicas de *clustering* o similitud. El siguiente paso es reducir la base de reglas. Para ello, además de emplear las técnicas de podado comentadas anteriormente, emplearemos una técnica de simplificación tabular propuesta en [49].

Tras aplicar la simplificación tabular, el último paso que aplicaremos es simplificar las funciones de pertenencia para las variables de entrada mediante la técnica de purga comentada anteriormente, porque es habitual que queden funciones de pertenencia sin usar.

3.3.1. Técnica de simplificación tabular

El método de *Quine-McCluskey* (*Q-M*) es una aproximación tabular a la minimización de funciones booleanas [50][51]. Básicamente, el método *Q-M* tiene dos ventajas sobre la técnica de los mapas de *Karnaugh*. En primer lugar, es un método sistemático sencillo para producir funciones mínimas que es menos dependiente de la habilidad del diseñador para reconocer patrones que el método de los mapas de *Karnaugh*. En segundo lugar, el método es viable a la hora de manejar un número alto de variables en oposición al mapa de *Karnaugh*, que, prácticamente, está limitado a unas cinco o seis variables. En general, el método *Q-M* realiza una búsqueda lineal ordenada en los mintérminos de la función para encontrar todas las combinaciones de mintérminos lógicamente adyacentes. Como se mostrará, también puede ser extendido a funciones de múltiples salidas.

El método de *Quine-McCluskey* comienza con una lista de los mintérminos de n variables de la función y deriva de forma sucesiva todos los implicantes con n_1 variables, los implicantes con n_2 variables, y así sucesivamente hasta que todos los implicantes primos son identificados. Entonces, se calcula una cobertura mínima de la función a partir del conjunto de implicantes primos.

Para poder utilizar el método explicado anteriormente para la minimización de reglas pertenecientes a bases de reglas de sistemas difusos hay que hacer una serie de modificaciones. En primer lugar, en el caso que nos ocupa, la salida de la función, es decir, el consecuente de la regla, no está limitado a dos valores como ocurre con las funciones de conmutación (0 y 1), sino que puede ser cualquier función de pertenencia del tipo al que pertenezca la variable de salida. Es por esto por lo que no tendremos un sólo conjunto de mintérminos, sino varios, en función de qué función de pertenencia de la variable de salida produzcan. La forma de actuar será repetir el proceso para cada una de las posibles funciones de pertenencia de la variable de salida que aparezcan en la base de reglas.

En segundo lugar, las variables de entrada de la base de reglas tampoco toman estrictamente dos valores, sino que pueden tomar como valor cualquier función de pertenencia del tipo al que pertenezcan. En este caso, lo más conveniente es asignar de forma unívoca un número a cada función de pertenencia dentro de cada tipo. De este modo, para realizar la primera agrupación de mintérminos no se agruparán aquellos que tengan el mismo número de unos en su representación binaria (lo que no tiene sentido ahora), sino aquellos mintérminos para los que la representación numérica de sus funciones de pertenencia sumen lo mismo. La combinación de mintérminos se hará entre aquellos cuya suma se diferencie en una unidad y que tengan las mismas funciones de pertenencia para todas sus variables salvo para una. Es importante que la numeración de las funciones de pertenencia se haga de forma creciente según su orden en el cubrimiento del universo de discurso (*Figura 15*).

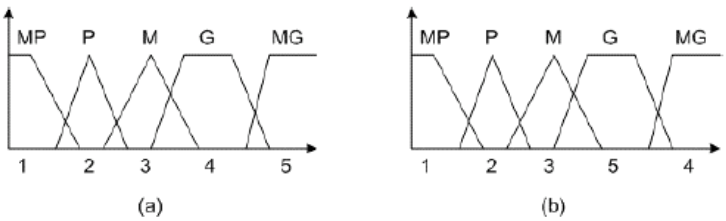

Figura 15. Numeración para las funciones de pertenencia de un tipo: (a) Numeración correcta; (b) Numeración incorrecta

Esta técnica permite simplificaciones como las que se ilustran en la *Figura 16*.

x1 \ x2	(1) MP	(2) P	(3) M	(4) G	(5) MG
(1) MP	Z	Z	Z	Z	Z
(2) P	PS	PS	Z	NS	NS
(3) M	Z	Z	Z	Z	Z
(4) G	N	N	Z	P	P
(5) MG	N	N	Z	P	P

(a)

x1 \ x2	(1) MP	(2) P	(3) M	(4) G	(5) MG
(1) MP	Z	Z	Z	Z	Z
(2) P	PS	PS	Z	NS	NS
(3) M	Z	Z	Z	Z	Z
(4) G	N	N	Z	P	P
(5) MG	N	N	Z	P	P

(b)

Figura 16. Base de reglas de partida (a) y agrupación de implicantes primos (b)

En este ejemplo, las 25 reglas (en forma tabular) mostradas en la *Figura 16(a)*, quedan simplificadas en las siguientes 5 reglas:

Si (x_1 *es* MP *o* x_1 *es* M *o* x_2 *es* M), *entonces* $y = Z$;
Si (x_1 *es* P *y* x_2 *es menor o igual que* P), *entonces* $y = PS$;
Si (x_1 *es* P *y* x_2 *es mayor o igual que* G), *entonces* $y = NS$;
Si (x_1 *es mayor o igual que* G *y* x_2 *es menor o igual que* P), *entonces* $y = N$;
Si (x_1 *es mayor o igual que* G *y* x_2 *es mayor o igual que* G), *entonces* $y = P$;

Si, tras este mecanismo de simplificación, aplicamos la técnica de purga para simplificar las funciones de pertenencia de las variables de entrada podemos observar, en este ejemplo, cómo se reducen las funciones de pertenencia de la variable x_1, de 5 a 4 (MP, P, M, G) y las de x_2, de 5 funciones iniciales a 3 (P, M, G).

Esta técnica de simplificación tabular de reglas se puede aplicar con la herramienta *Xfsp* de *Xfuzzy*.

3.4. Arquitectura de sistemas difusos basados en reglas activas

En este apartado se describe la arquitectura de sistemas difusos que se ha empleado en las implementaciones realizadas en este Proyecto Fin de Carrera. El objetivo fundamental de esta arquitectura es conseguir una alta velocidad de operación y reducir al máximo los recursos necesarios para su implementación [52], [53]. Dicha arquitectura (*Figura 17*) se caracteriza por presentar una estructura modular y un alto grado de configurabilidad, debido a la posibilidad de implementar cada uno de los bloques de la figura a partir de diferentes realizaciones de circuito.

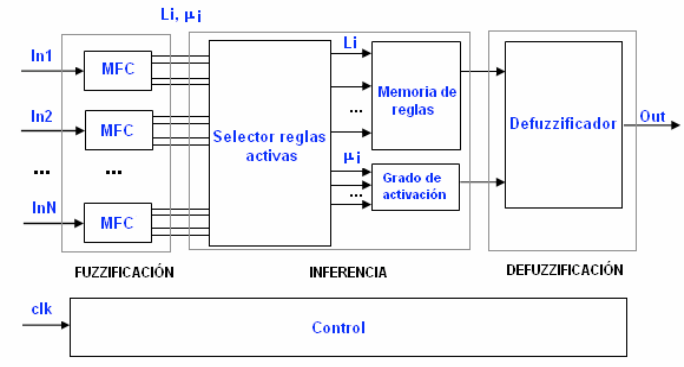

Figura 17. Estructura general de la arquitectura

Se basa en los siguientes principios:

- **Inferencia dirigida por reglas activas:** Son las arquitecturas que obtienen una mejor relación entre las prestaciones y los recursos requeridos en comparación con las arquitecturas que evalúan las bases de reglas en paralelo o de forma secuencial.

- **Limitación del grado de solapamiento de las funciones de pertenencia de las entradas:** Si limitamos a N el grado de solapamiento se reduce a NI el número de reglas activas a procesar (donde I es el número de entradas). También se reducen los recursos necesarios para calcular los grados de pertenencia de cada entrada a los conjuntos difusos correspondientes.

- **Empleo de métodos de defuzzificación simplificados:** Estos métodos realizan operaciones sólo sobre cada una de las reglas. Si además, tenemos en cuenta que se evalúan únicamente las reglas activas, para obtener el valor de salida se requiere solamente operaciones sobre las reglas activas. Esto supone un incremento de velocidad y una reducción en los recursos requeridos.

En la *Figura 17* aparecen las tres etapas típicas en el cálculo de una inferencia difusa: fuzzificación, inferencia y defuzzificación. Los circuitos generadores de funciones de pertenencia (*MFC*) suministran dos parejas de datos etiqueta-grado de pertenencia (L_i,μ_i) para cada valor de las entradas del sistema. En la etapa de inferencia se procesan secuencialmente cada una de las reglas activas, utilizándose para conseguir esto, un circuito selector de reglas activas. En cada ciclo de reloj, se combinan los grados de pertenencia μ_i de cada entrada a través del conectivo de antecedentes para calcular el grado de activación de la regla (α^r), mientras que la combinación de las respectivas etiquetas de los antecedentes direcciona la posición de memoria que contiene el consecuente. En la etapa de defuzzificación, un bloque obtiene el valor de salida adecuado utilizando los grados de activación y consecuentes calculados en la etapa previa. Un bloque de control se encarga de regular la temporización del proceso.

3.5. Síntesis de sistemas difusos con System Generator

En [54] se ha propuesto una nueva técnica de implementación de sistemas de inferencia difusos basada en el desarrollo, mediante la herramienta *System Generator*, de una librería de celdas que implementa cada uno de los bloques que aparecen en la arquitectura descrita en la sección 3.4. Para cubrir las etapas de *fuzzificación*, inferencia y *defuzzificación*, se han diseñado un conjunto de bloques encargados de la generación aritmética de las funciones de pertenencia, el procesado secuencial de las reglas activas, el suministro de las señales de control, el cálculo del grado de activación de las reglas, el almacenamiento de los consecuentes de la base de reglas y la realización de tareas de acumulación y división para diversos métodos de defuzzificación. En la F*igura 18* se puede observar la versión actual de la librería *XfuzzyLib* (versión 3), que consta de un total de 9 grupos de bloques.

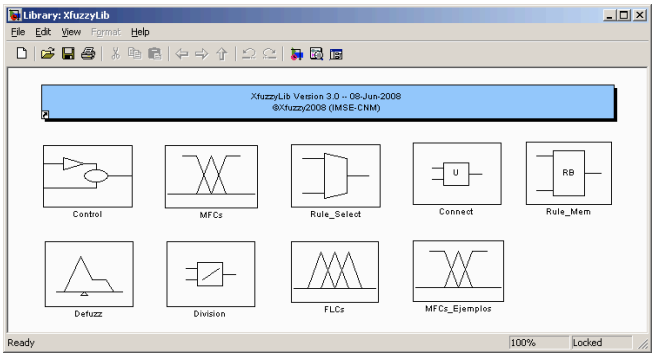

Figura 18. Conjunto de bloques de la librería de módulos *XfuzzyLib*

A continuación se describe de forma resumida el contenido de cada uno de los grupos de celdas que aparecen en la figura anterior [54].

Generadores de funciones de pertenencia (*MFCs*): Estos bloques se encargan de generar las funciones de pertenencia de los antecedentes de las reglas de un sistema difuso utilizando técnicas aritméticas. El uso de estas técnicas presenta muchas ventajas desde el punto de vista de su realización hardware [55]. Las funciones de pertenencia generadas tienen formas triangulares, salvo la primera y la última función que pueden ser de tipo 'S' y 'Z', respectivamente. Presentan grado de solapamiento 2 y están normalizadas. Una vez realizado el diagrama de bloques de un nuevo módulo de la librería, se encapsula como un 'subsistema' y se añade una 'máscara' para identificar los distintos parámetros que caracterizan al módulo. De esta forma, cuando dicho módulo se use en un sistema jerárquico superior, estos parámetros pueden asignarse mediante valores numéricos o mediante variables a las que se les asigna un valor numérico en la ventana de comandos de Matlab o a través de un fichero '.m'. Los parámetros que identifican a un *MFC* son el número de bits de las entradas, los valores de los puntos de corte, los valores de las pendientes y su número de bits, el número de funciones de pertenencia y el número de bits del grado de pertenencia. La librería incluye también varios *MFCs* de ejemplo con distribuciones típicas de funciones de pertenencia. En la *Figura 19* se puede observar el símbolo del subsistema correspondiente al generador de funciones de pertenencia, así como los parámetros necesarios para dimensionar ese bloque.

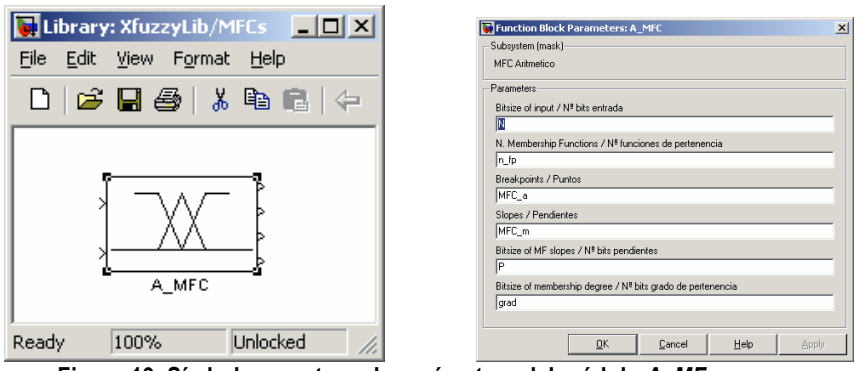

Figura 19. Símbolo y ventana de parámetros del módulo *A_MF*

- **Selectores de reglas activas (*Rule_Select*):** La librería incluye módulos de selección de reglas activas para módulos de inferencia con una, dos y tres entradas. Dicho circuito está realizado usando dos multiplexores por cada entrada.

- **Conectivos de antecedentes (*Connect*):** Incluye conectivos Mínimo y Producto con diferente número de entradas. El único parámetro que se utiliza para su dimensionado es el número de bits del grado de pertenencia.

- **Memoria de reglas (*Rule_Mem*):** La versión actual de la librería proporciona tres tipos de memoria de reglas. En todos los casos la memoria dispone de un parámetro llamado 'Nº bits bus de direcciones', que se calcula con una determinada expresión que permite distinto número de funciones de pertenencia en cada una de las entradas. El número de parámetros a almacenar depende del método de *defuzzificación*. Cuando es *FuzzyMean* o *MaxLabel* sólo hay que almacenar los consecuentes de las reglas. Para *WeightedFuzzyMean*, *Quality* o *GammaQuality*, hay que almacenar también los valores de los pesos correspondientes, que tienen en cuenta el área o el soporte de los conjuntos difusos de salida. Por último, si el método es *Takagi-Sugeno* de orden 1 hay que almacenar tres parámetros para el caso de dos entradas.

- **Defuzzificadores (*Defuzz*):** Se han implementado los métodos de defuzzificación *FuzzyMean*, *WeightedFuzzyMean* (*Quality* y *GammaQuality* son equivalentes), *MaxLabel* y *Takagi-Sugeno* de orden 1. Para todos ellos el único parámetro de dimensionamiento es el número de bits de la salida. En la *Figura 20* se puede ver el diagrama de bloques del defuzzificador *FuzzyMean*.

- **División:** Proporciona un módulo divisor basado en el algoritmo '*resto sin restauración*' [56] que puede ser utilizado por cualquiera de los bloques descritos anteriormente. El parámetro que dimensiona este bloque es el número de bits de la salida.

- **Bloques de control (Control):** Se ha diseñado un módulo encargado de generar el contador de ciclos de operación y diversas señales necesarias para el control de las diferentes etapas del sistema difuso. Entre estas señales se encuentran las del selector de reglas activas y del *defuzzificador*. El parámetro necesario para el dimensionamiento del módulo es el número de ciclos, que vendrá determinado, en el caso más general, por el máximo del número de reglas activas, el número de funciones de pertenencia y el número de bits del divisor.

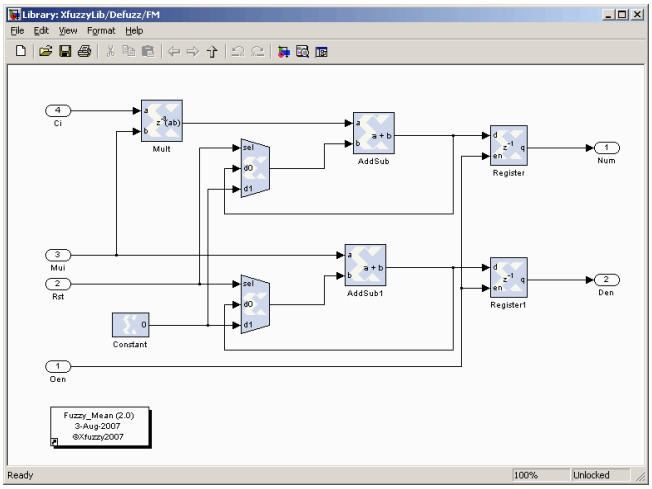

Figura 20. Diagrama de bloques del defuzzificador *FuzzyMean*

- **Arquitecturas tipo de sistemas de inferencia difusos (FLCs):** La librería contiene diferentes descripciones de arquitecturas tipo de sistemas de inferencia, a las que denominamos '*FLCs*' y que se diferencian entre sí en el conectivo de antecedentes y método de defuzzificación que utilizan. En las *Figura 21 y 22* se puede observar el esquema de dos *FLC* de 2 entradas que usan producto como conectivo de antecedentes y *FuzzyMean* simplificado o *Weihted- FuzzyMean* como *defuzzificador*, respectivamente. Estos *FLCs* han sido realizados, a partir de la interconexión de los bloques explicados anteriormente, con el objetivo de facilitar el desarrollo de sistemas difusos de mayor complejidad. Como se puede observar en la *Figura 23*, cada uno de estos bloques es totalmente parametrizable. Básicamente hay dos tipos de parámetros. Los que corresponden al dimensionado del sistema, como son el número de bits de entrada, salida y grado de pertenencia. Y los que definen las funciones de pertenencia y la base de reglas. Los distintos parámetros corresponden a variables que se pueden inicializar a través de un fichero de configuración '.m' de *Matlab*.

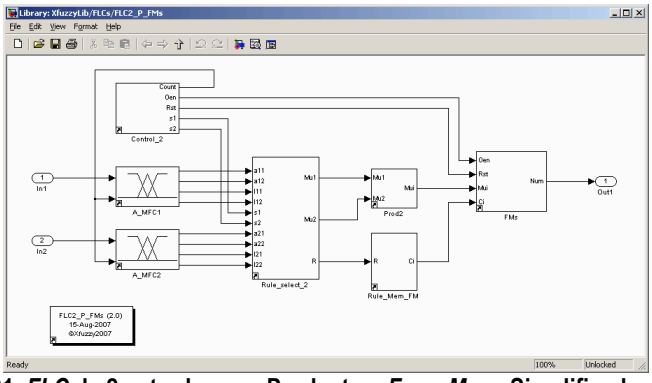

Figura 21. *FLC* de 2 entradas con Producto y *FuzzyMean* Simplificado

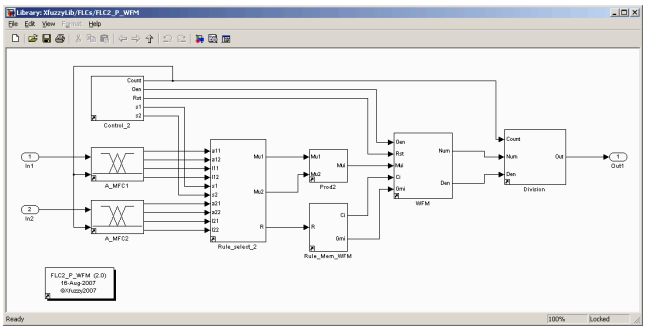

Figura 22. FLC de 2 entradas con Producto y *WeightedFuzzyMean*

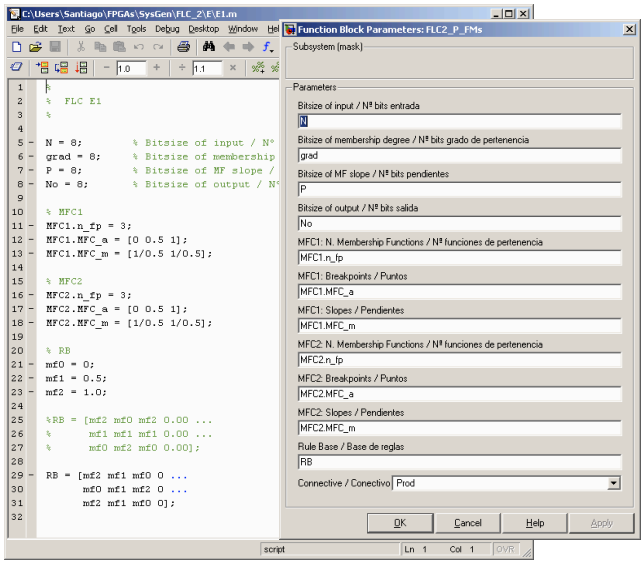

Figura 23. Parámetros de un *FLC2_P_FMs* y definición mediante fichero '.m'

Utilizando la librería *XfuzzyLib*, la construcción de un sistema difuso requiere elegir los componentes adecuados, interconectarlos e inicializar los parámetros que dimensionan cada bloque correctamente. Todo esto se puede hacer de una manera sencilla con *Simulink* (la herramienta interactiva para el modelado, la simulación y el análisis de sistemas dinámicos integrada en *Matlab*). Cada uno de los bloques que hemos descrito anteriormente está realizado utilizando la librería de módulos '*Xilinx Blockset*' de *Simulink*, que incluye la herramienta *System Generator*. La herramienta *System Generator* permite trasladar el modelo matemático descrito en *Simulink* a un modelo *hardware* descrito en lenguaje *HDL*, optimizado en términos de área y velocidad. Dicha descripción constituye la entrada a las herramientas de síntesis e implementación de *FPGAs*.

Todo este proceso de síntesis puede iniciarse desde *Xfuzzy* con la herramienta *Xfsg*.

3.6. Flujo de diseño

El flujo de diseño utilizado en este Proyecto Fin de Carrera combina el uso de las herramientas específicas de desarrollo de sistemas difusos del entorno *Xfuzzy*, las herramientas de modelado y simulación de *Matlab* y las herramientas de síntesis e implementación de FPGAs de *Xilinx*. De acuerdo con dicho flujo de diseño, el desarrollo del sistema de control se realizará en dos etapas con niveles de abstracción diferentes como se muestra en la *Figura 24*.

La primera etapa, que tiene como objetivo la descripción y verificación funcional del sistema de control, se llevará a cabo con ayuda de las distintas facilidades de *Xfuzzy*. El sistema de control difuso

será descrito como una especificación *XFL3* que puede combinar módulos difusos (para tareas de aproximación de funciones o toma de decisiones) y módulos crisp (que implementan operadores aritméticos y lógicos de propósito general). Las bases de conocimiento se pueden generar directamente a partir de la experiencia de un experto (*Xfedit*), o bien mediante el empleo de datos numéricos de entrenamiento y herramientas de identificación (*Xfdm*) y aprendizaje supervisado (*Xfsl*).En este Proyecto Fin de Carrera se ha optado por seguir el segundo de los caminos, empleando como datos numéricos los que se obtienen con la herramienta *Hybrid Toolbox* que permite diseñar controladores *PWA*. La descripción identificada y aprendida se simplificará mediante la herramienta *Xfsp*. La verificación funcional se llevará a cabo analizando la relación entrada/salida del sistema (*Xfplot*) así como simulando su comportamiento en lazo cerrado cuando se combina con un modelo de caso de estudio codificado en lenguaje Java (*Xfsim*). Una vez validada la especificación *XFL3* del sistema de control, la herramienta de síntesis hardware incluida en *Xfuzzy* (*Xfsg*) generará los ficheros necesarios para iniciar la segunda etapa.

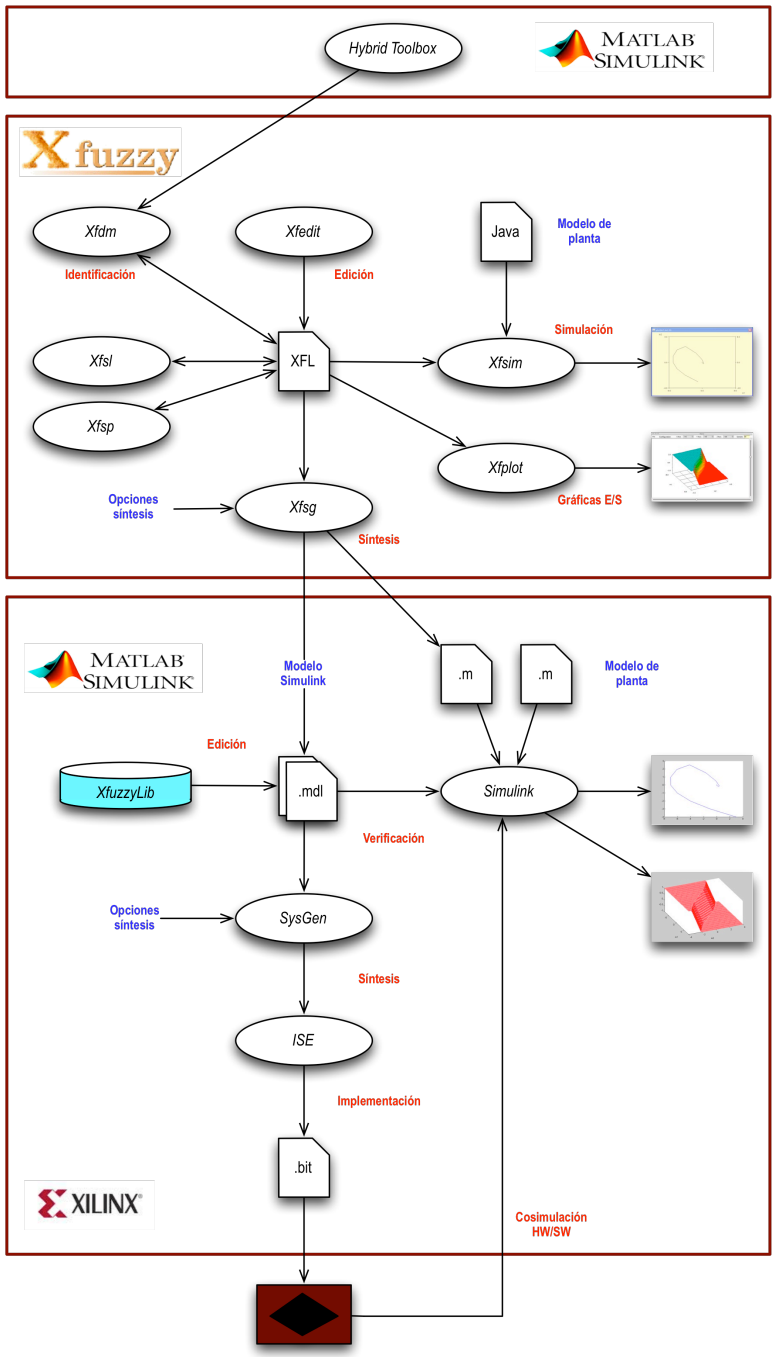

Figura 24. Flujo de diseño

La segunda etapa de desarrollo conlleva la ejecución, desde el entorno *Matlab*, de diferentes herramientas de *Xilinx* que facilitan la síntesis e implementación del sistema de control difuso sobre una *FPGA*. En esta etapa de diseño el sistema se describe mediante un modelo *Simulink* que combina distintos componentes de la librería *XfuzzyLib*. El modelo del sistema de control puede construirse a partir de cero o con ayuda de los ficheros (*.mdl* y *.txt*) generados por *Xfsg*. El uso de los diferentes elementos de generación y visualización de señales que proporcionan las distintas toolboxes de *Simulink* facilita la verificación del sistema a nivel lógico, permitiendo analizar en detalle aspectos tales como el número de bits de precisión o el tipo de aritmética usada en los diferentes componentes del sistema. La inclusión de un modelo *Simulink* de la planta bajo control permitirá, asimismo, verificar la operación del sistema de control en lazo cerrado con el mismo nivel de detalle. En ambos casos, la funcionalidad del sistema difuso viene definida por el fichero '.m' generado por *Xfsg*. Una vez validado el modelo *Simulink*, comenzará la fase de síntesis e implementación seleccionando la opción deseada con ayuda de la interfaz gráfica de *System Generator* incluida en el modelo. Entre las posibles opciones se incluye: traslación a código *HDL*, generación del fichero de configuración de la *FPGA* (bitstream) y configuración del entorno para la realización de *cosimulación hardware/software*. En este último caso la verificación final del sistema de control puede llevarse a cabo mediante la integración en un lazo cerrado de cosimulación del modelo software de la planta descrito en *Matlab* y la implementación hardware del controlador sobre la *FPGA*.

3.7. Conclusiones

Las herramientas de *CAD* integradas en el entorno *Xfuzzy* nos facilitan el diseño de controladores difusos en general y controladores *PWA* en particular. En el siguiente *Capítulo* veremos cómo se han empleado estas herramientas en el diseño de controladores *PWA* basados en módulos difusos.