IMPLEMENTACIÓN SOBRE FPGAS DE SISTEMAS DIFUSOS PROGRAMABLES

S. Sánchez-Solano[1], A. Cabrera[2], C. J. Jiménez[1], P. Brox[1], I. Baturone[1], A. Barriga[1]

[1] Instituto de Microelectrónica de Sevilla, Centro Nacional de Microelectrónica,
Avda. Reina Mercedes s/n, 41012-Sevilla, España. <santiago@imse.cnm.es>
[2] Dpto. Automática y Computación, Facultad de Ingeniería Eléctrica, ISPJAE,
Ciudad de la Habana, Cuba. <alex@electrica.ispjae.edu.cu>

ABSTRACT

The number of electronic applications using fuzzy logic–based solutions has increased considerably in the last few years. Concurrently, new CAD tools that explore different implementation technologies for this type of systems have been developed. Among them, the use of specific processing architectures implemented on FPGAs presents as main advantages a good "cost-performance" ratio and an acceptably short development time. The different synthesis facilities provided by the *Xfuzzy* design environment for the implementation of programmable fuzzy systems, which take advantage of the available resources in the current FPGAs families, are analyzed in this paper.

RESUMEN

El número de aplicaciones electrónicas que utilizan soluciones basadas en lógica difusa se ha incrementado considerablemente en los últimos años y, de forma paralela, se han desarrollado nuevas herramientas de CAD que contemplan diferentes técnicas de implementación para este tipo de sistemas. De entre ellas, el uso de arquitecturas específicas de procesado implementadas sobre FPGAs presenta como principales ventajas una buena relación "coste-rendimiento" y un ciclo de desarrollo aceptablemente corto. En esta comunicación se analizan las distintas facilidades de síntesis que proporciona el entorno de diseño Xfuzzy para la implementación de sistemas difusos programables que aprovechen los recursos disponibles en las actuales familias de FPGAs.

IMPLEMENTACIÓN SOBRE FPGAS DE SISTEMAS DIFUSOS PROGRAMABLES[1]

S. Sánchez-Solano[1], A. Cabrera[2], C. J. Jiménez[1], P. Brox[1], I. Baturone[1], A. Barriga[1]

[1] Instituto de Microelectrónica de Sevilla, Centro Nacional de Microelectrónica,
Avda. Reina Mercedes s/n, 41012-Sevilla, España. <santiago@imse.cnm.es>
[2] Dpto. Automática y Computación, Facultad de Ingeniería Eléctrica, ISPJAE,
Ciudad de la Habana, Cuba. <alex@electrica.ispjae.edu.cu>

ABSTRACT

El número de aplicaciones electrónicas que utilizan soluciones basadas en lógica difusa se ha incrementado considerablemente en los últimos años y, de forma paralela, se han desarrollado nuevas herramientas de CAD que contemplan diferentes técnicas de implementación para este tipo de sistemas. De entre ellas, el uso de arquitecturas específicas de procesado implementadas sobre FPGAs presenta como principales ventajas una buena relación "coste-rendimiento" y un ciclo de desarrollo aceptablemente corto. En esta comunicación se analizan las distintas facilidades de síntesis que proporciona el entorno de diseño *Xfuzzy* para la implementación de sistemas difusos programables que aprovechen los recursos disponibles en las actuales familias de FPGAs.

1. INTRODUCCIÓN

Las técnicas de inferencia basadas en lógica difusa proporcionan mecanismos adecuados para tratar la vaguedad e imprecisión típica del lenguaje natural permitiendo, de esta forma, la aplicación de estrategias de razonamiento aproximado a partir del conocimiento experto expresado de forma lingüística. Por otra parte, los formalismos matemáticos en que se basan estas técnicas de inferencia proporcionan procedimientos deterministas y sistemáticos que pueden ser fácilmente implementados mediante programas de ordenador o circuitos micro-electrónicos de aplicación específica. Las dos caracterís-ticas anteriores han motivado el desarrollo en los últimos años de numerosas aplicaciones que utilizan lógica difusa para resolver distintos problemas de control de procesos y toma de decisiones. En estas aplicaciones las soluciones basadas en lógica difusa permiten aplicar técnicas de procesado no lineal en sistemas donde resulta difícil o imposible disponer de un modelo matemático de comportamiento, a la vez que posibilitan el desarrollo de interfaces de usuario basadas en los términos lingüísticos habituales en el lenguaje natural [1-3].

Las técnicas de implementación de los sistemas difusos empleadas en muchas aplicaciones se basan en el uso de software sobre procesadores programables de propósito general. Sin embargo, para aplicaciones que requieren alta velocidad de inferencia y bajo consumo de área y potencia es necesario recurrir a la utilización de hardware específico que implemente una arquitectura eficiente [4]. Algunos aspectos claves para conseguir sistemas de inferencia difusos de bajo coste y alta velocidad son la limitación de los tipos de funciones de pertenencia usadas en las reglas, el uso de métodos de defuzzificación simplificados y el empleo de estrategias de cálculo que evalúen solo la contribución de las reglas activas [5].

Los principales inconvenientes asociados con la realización microelectrónica de sistemas difusos provienen del elevado coste en términos económicos y en tiempo de desarrollo que conlleva el diseño y fabricación de un circuito integrado. En este sentido, dos factores que pueden minimizar en gran medida estos inconvenientes son el empleo de herramientas de CAD que aceleren los procesos de descripción verificación y síntesis automática, y el uso de dispositivos programables que simplifiquen las etapas de prototipado y fabricación de los sistemas. En particular, el uso de arquitecturas específicas de procesado implementadas sobre FPGAs proporciona una excelente relación "coste-rendimiento" y un ciclo de desarrollo extremadamente corto [6].

En esta comunicación se describe un flujo de diseño de sistemas difusos sobre FPGAs que se apoya en el uso combinado de herramientas específicas para el desarrollo de sistemas basados en lógica difusa [7-8] y herramientas genéricas de simulación y síntesis a partir de lenguajes de descripción de hardware. Asimismo, se analizan los diferentes estilos de descripción y opciones de implementación que permiten sacar partido a los recursos de memoria que ofrecen los dispositivos actualmente disponibles en el mercado.

[1] Este trabajo ha sido parcialmente financiado por el proyecto CICYT TIC2001-1726

2. COMPONENTES DE UN SISTEMA DIFUSO

El diagrama de bloques básico de un sistema difuso que utiliza el método de defuzzificación de la media difusa se muestra en la Figura 1. Los circuitos generadores de funciones de pertenencia (MFCs) generan los conjuntos difusos utilizados en los antecedentes de las reglas. Para cada valor de las entradas, los MFCs suministran tantos pares (etiqueta, valor de activación) como grado de solapamiento se haya previsto en el sistema. El máximo número de reglas activas (reglas con contribución no nula) queda limitado al fijar un grado de solapamiento. La etapa de inferencia se encarga de procesar secuencialmente cada una de dichas reglas, utilizándose para ello un array de multiplexores controlados por un contador. En cada ciclo del contador los grados de pertenencia son combinados a través del operador MIN o Producto para calcular el grado de activación de la regla (α^i), mientras que las etiquetas de los antecedentes direccionan la posición de la memoria de reglas que contiene el consecuente correspondiente a dicha regla (c^i). Finalmente, la etapa de defuzzificación calcula la salida del sistema como la media de los consecuentes de las reglas ponderada por sus grados de activación, de acuerdo con la ecuación:

$$y = \Sigma_r \alpha^i \, . \, c^i \, / \, \Sigma_r \alpha^i \qquad (1)$$

En un sistema de I variables de entrada con N bits de precisión y grado de solapamiento 2, que emplee las etapas de pipeline mostradas en la Figura 1, la velocidad de inferencia viene limitada por la mayor de las tres cantidades siguientes: el número de ciclos de reloj necesarios para calcular los antecedentes; el número de reglas activas (2^I); y el número de ciclos empleados en la división. Para precisiones de 8-10 bits y frecuencias de 30-50 MHz la velocidad de inferencia obtenida con esta arquitectura de procesado puede ser superior a varios MFLIPS (millones de inferencias difusas por segundo).

La operación del sistema difuso viene determinada por el contenido de su base de conocimiento, es decir, por el conjunto de reglas que la definen. Según el diagrama de bloques de la Figura 1, dicha base de conocimiento se encuentra repartida entre los MFCs y la memoria de reglas. Dependiendo del tipo de aplicación y de la tecnología elegida para realizar el circuito, existen diferentes opciones tanto arquitecturales como de implementación para construir cada uno de estos bloques.

Como se ilustra en la Figura 2, los MFCs pueden ser implementados mediante técnicas vectoriales o aritméticas. En el primer caso, las memorias de antecedentes que almacenan los distintos grados de pertenencia son direccionadas por las palabras binarias correspondientes a los valores de las entradas. El tamaño de cada una de las I memorias es 2^N palabras de $(2*D + \lceil \log_2 F \rceil)$ bits, donde D es el número de bits de precisión con que se almacenan los grados de pertenencia y F el número de etiquetas utilizado. El empleo de MFCs basados en memoria permite la definición de funciones de pertenencia arbitrarias, aunque puede condicionar la realizabilidad del circuito cuando se incrementa el número de entradas o su precisión.

El diagrama de bloques de la Figura 2b muestra el esquema de un circuito aritmético capaz de generar un conjunto de funciones de pertenencia triangulares a partir de los coeficientes que definen los valores de las posiciones y las pendientes de los conjuntos difusos. La memoria necesaria para almacenar los coeficientes requiere en este caso $\lceil \log_2 F \rceil$ palabras de I*(N+P) bits, donde N y P son los bits empleados para guardar los coeficientes de los centros y las pendientes, respectivamente. La simplicidad de este circuito permite incrementar considerablemente el número de funciones utilizadas en una determinada aplicación. Una ventaja adicional es que las funciones de pertenencia generadas se encuentran normalizadas, en el sentido de que su suma es siempre 1. Esta característica permite eliminar la división de la etapa de defuzzificación cuando se utiliza el producto como conectivo de antecedentes.

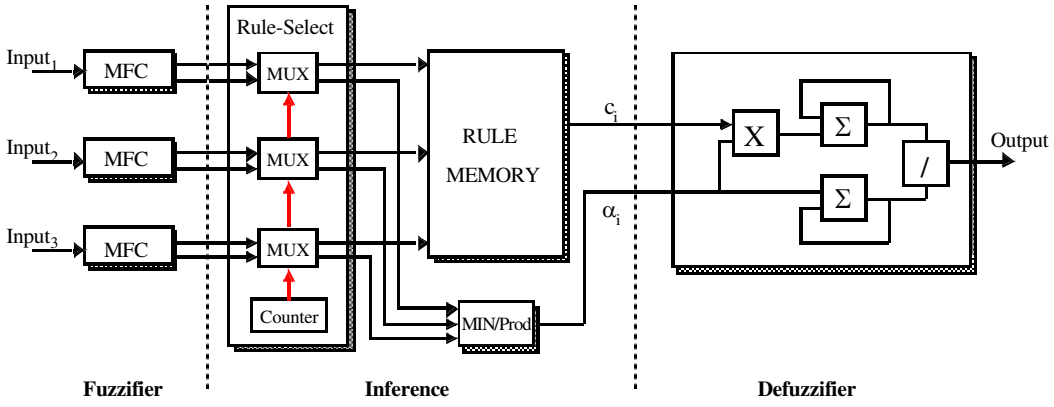

Figura 1: Esquema de evaluación de reglas activas en sistemas de inferencia basados en lógica difusa.

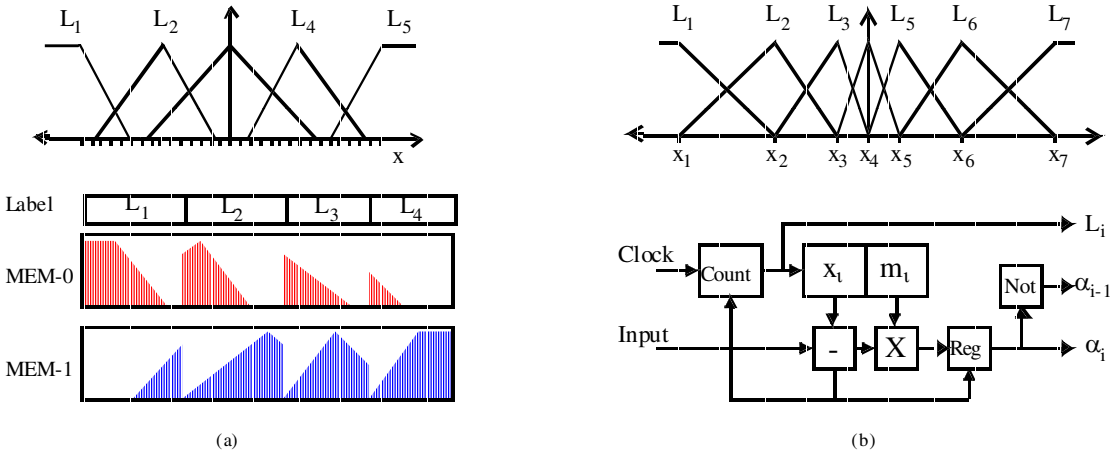

Figura 2: Implementación de MFCs mediante técnicas vectorial (a) y aritmética (b).

Desde el punto de vista de implementación, tanto la memoria de antecedentes (independientemente del tipo de MFC utilizado) como la memoria de reglas admiten diferentes opciones: memoria RAM o ROM (según interese o no modificar la base de conocimiento del sistema de forma concurrente con su operación), o circuitos combinacionales que permitan reducir el tamaño de la lógica necesaria.

A diferencia de una implementación ASIC, donde la funcionalidad del sistema queda totalmente definida una vez fabricado el circuito, determinados tipos de FPGAs, como las suministradas por *Xilinx*, mantienen la configuración interna del dispositivo en una memoria RAM, de forma que dicha funcionalidad puede cambiarse sin más que "reprogramar" el circuito. Esta característica de programabilidad intrínseca permite, entre otras cosas, actualizar la base de conocimiento del sistema difuso con independencia de que se hayan empleado memorias ROM o bloques combinacionales para almacenar los valores de los antecedentes y los consecuentes. La disponibilidad de herramientas automáticas de diseño que permitan obtener cómoda y rápidamente el fichero de configuración de la FPGA a partir de las especificaciones de alto nivel del sistema facilita en gran medida la realización de este proceso.

Cuando el cambio en la base de conocimiento debe ser concurrente con la propia operación del sistema (por ejemplo, para sistemas adaptativos) es preciso utilizar memoria RAM. Los bloques básicos combinacionales de ciertas FPGAs están formados por pequeñas memorias que actúan como tablas de búsqueda capaces de implementar cualquier función de un determinado número de entradas. Estas memorias pueden combinarse para formar bloques de RAM distribuida utilizables por el diseñador. Adicionalmente, algunas familias de FPGAS como la Spartan2E de *Xilinx* ponen a disposición del diseñador bloques específicos de memoria que admiten diferentes configuraciones. Las herramientas de diseño suministradas por el fabricante permiten inferir el uso de memoria a partir de descripciones VHDL genéricas en la etapa de síntesis y seleccionar el tipo de memoria a utilizar en la etapa de implementación.

En el resto de esta comunicación se describen las modificaciones realizadas en las herramientas de síntesis de *Xfuzzy* para sacar partido de estas características, así como los resultados obtenidos al utilizar dichas herramientas para realizar una exploración del espacio de diseño de sistemas difusos.

3. DISEÑO DE SISTEMAS DIFUSOS CON XFUZZY

El entorno de desarrollo *Xfuzzy* proporciona un conjunto de herramientas que facilitan las diferentes etapas de diseño de sistemas difusos [7]. Siguiendo el diagrama de flujo de la Figura 3, el proceso de diseño de un sistema comienza con la definición de su especificación en el lenguaje XFL. Las distintas herramientas del entorno permiten completar la especificación, modelar el sistema y ajustar su definición a las características de la aplicación considerada.

Una vez obtenida una especificación funcionalmente correcta, puede comenzar la fase de síntesis VHDL. La herramienta *xfvhdl* toma como entrada la especificación XFL de un sistema difuso y genera un conjunto de ficheros VHDL que implementan el motor de inferencia. Para ello los componentes correspondientes a las distintas opciones arquitecturales han sido previamente definidos e incluidos en una librería de celdas parametrizables que cumplen las restricciones impuestas por las herramientas de síntesis disponibles [8].

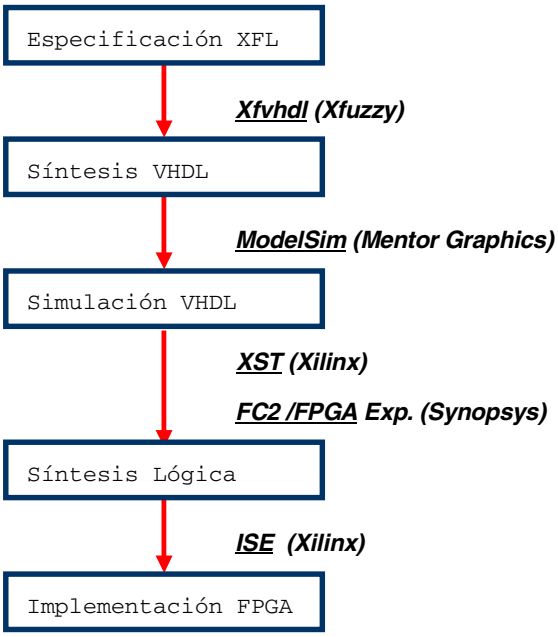

Figura 3: Flujo de diseño de sistemas difusos con *Xfuzzy*.

La descripción VHDL constituye el punto de partida de las etapas de simulación (simulador *ModelSim* de *Mentor Graphics*) y síntesis lógica (*XST* de *Xilinx* o *FPGA Compiler 2* y *FPGA Express* de *Synopsys*). Con objeto de acelerar la realización de estas dos etapas de diseño, *xfvhdl* proporciona dos salidas adicionales: un fichero de *testbench* para facilitar la simulación del sistema difuso y procedimientos de comandos que permiten realizar en batch los procesos de síntesis e implementación de la FPGA sobre dispositivos *Xilinx*.

Al ejecutar la herramienta *xfvhdl* el diseñador puede seleccionar las siguientes opciones arquitecturales y de implementación:

Opciones arquitecturales:
- MFCs basados en memoria
- MFCs aritméticos

Base de conocimiento:
- Predeterminada (ROM)
- Programable (RAM)

Implementación de Memoria:
- RAM: Distribuida o Tipo bloque
- ROM: Distribuida o Lógica combinacional

La selección de la estrategia empleada para almacenar la base de conocimiento condiciona el estilo utilizado en las descripciones VHDL de los diferentes módulos del sistema. La Figura 4 muestra una porción del código correspondiente a un MFC basado en memoria y que va a ser implementado mediante ROM o lógica combinacional. La elección entre estas dos últimas alternativas depende de la herramienta de síntesis lógica empleada. Las herramientas de *Synopsys* (*FPGA Compiler 2* y *FPGA Express*) siempre implementarán el MFC mediante lógica combinacional. La herramienta de síntesis de *Xilinx* (*XST*), por el contrario, es capaz de detectar la estructura de la memoria ROM, de forma que las herramientas de implementación puedan configurar adecuadamente las bloques básicos de las FPGAs de las familias XC4000 y Spartan2E (CLBs y Slices, respectivamente).

El código de la Figura 5 corresponde a un MFC basado en memoria RAM. Las versiones actuales de las herramientas de *Synopsys* no son capaces de inferir el uso de memoria a partir de código VHDL como el mostrado en la figura. Sin embargo, la herramienta de *Xilinx* sí es capaz de inferir la memoria e implementarla como una memoria distribuida (en la mayor parte de las familias de FPGAs actuales) o como memoria de tipo bloque (en las FPGAs más modernas como las de las familias Spartan2E y Virtex).

La selección de la herramienta de síntesis y de las distintas opciones de implementación se realiza mediante parámetros en la llamada a *xfvhdl*. La opción –C permite elegir entre *Xilinx-XST* (x), *Synopsys-FPGA Express* (e) o *Synopsys-FPGA Compiler 2* (f). El parámetro –M determina la implementación de las memorias mediante RAM distribuida (d), RAM de tipo bloque (b), ROM (o) o lógica combinacional (l). El comando *xfvhdl* admite parámetros adicionales para dimensionar el sistema difuso, seleccionar el dispositivo FPGA, definir el nivel de

```
entity AntecedentMem_1 is
  port( addr:  in std_logic_vector(N-1 downto 0);
        do  : out std_logic_vector(M-1 downto 0));
end AntecedentMem_1;

architecture ROM of AntecedentMem_1 is
  subtype ROM_WORD is std_logic_vector (M-1 downto 0);
  type ROM_TABLE is array (0 to fil-1) of ROM_WORD;
  constant ROM: ROM_TABLE := ROM_TABLE'(
          ROM_WORD'("0000000000000"),
          ROM_WORD'("0001100000110"),
          ROM_WORD'("0001000101101"),
           . . .
          ROM_WORD'("1010000011111"));

  begin
    do <= ROM(conv_integer(addr));
end ROM;
```

Figura 4: Descripción de MFC implementado mediante Memoria ROM o lógica combinacional.

```
entity AntecedentMem is
  port( clk : in std_logic;
        we  : in std_logic;
        addr: in std_logic_vector(N-1 downto 0);
        di  : in std_logic_vector(M-1 downto 0);
        do  : out std_logic_vector(M-1 downto 0));
end AntecedentMem;

architecture RAM of AntecedentMem is
  type ram_type is array (fil-1 downto 0)
              of std_logic_vector(M-1 downto 0);
   signal RAM: ram_type;
  signal read_a: std_logic_vector (N-1 downto 0);

begin
   process(clk)
   begin
     if (clk'event and clk = '1') then
          if (we = '1') then
               RAM(conv_integer(addr)) <= di;
             end if;
             read_a <= addr;
        end if;
      end process;
      do <= RAM(conv_integer(read_a));
end RAM;
```

Figura 5: Descripción de MFC implementado mediante Memoria RAM.

esfuerzo y el objetivo (área o tiempo) a minimizar durante la síntesis y para ejecutar de forma automática los procesos de síntesis e implementación.

4. EXPLORACIÓN DEL ESPACIO DE DISEÑO

Con objeto de comparar los recursos utilizados por las distintas opciones arquitecturales y de implementación se ha realizado una exploración del espacio de diseño de un sistema de dos entradas con 7 funciones de pertenencia por entrada y 49 reglas, utilizando las herramientas de *Synopsys* y *Xilinx* para FPGAs de las familias Spartan2E y XC4000XL, considerándose precisiones entre 5 y 12 bits. Los resultados más representativos de este estudio se incluyen a continuación.

Las gráficas de la Figura 6 muestran el número de Slices utilizados en una FPGA xc2s200e frente al número de bits de precisión para las distintas opciones de implementación de los elementos de memoria del sistema difuso. La gráfica superior corresponde a la opción de MFCs basados en memoria (FLCM). Puede observarse que el crecimiento del sistema es exponencial, salvo en el caso de utilizar RAM de tipo bloque, lo que imposibilita la realización de sistemas con precisiones superiores a 9 bits. Por el contrario, como se muestra en la gráfica inferior, la reducción en el tamaño de la memoria de antecedentes que supone el empleo de MFCs aritméticos (FLCA) repercute en la cantidad de recursos utilizados y se traduce en un incremento lineal de éstos, lo que permite implementar sistemas con precisiones mayores. En todos los casos, con la utilización de RAM tipo bloque (RAMB) se consigue reducir el número de Slices requeridos, mientras que el empleo de RAM distribuida (RAMD) requiere más recursos que el uso de ROM o lógica combinacional, ya que es preciso incorpora los buses de entrada y las señales de control que permitan modificar los contenidos de las memorias.

Las gráficas de la Figura 7 ilustran con mayor detalle los recursos necesarios para implementar los MFCs con los dos tipos de memoria disponibles en las FPGAs Spartan2E. La figura de la izquierda muestra el número de Slices de la FPGA configurados como bloques básicos de memoria para conformar una memoria distribuida. De nuevo puede observarse que el crecimiento es exponencial para MFCs basados en memoria y lineal para MFCs aritméticos. La gráfica de la derecha muestra el porcentaje de recursos específicos de memoria tipo bloque utilizado por las dos opciones de MFCs. El crecimiento poco gradual que aparece en la gráfica se debe a que estas memorias solo admiten determinadas configuraciones.

La efectividad de las diferentes herramientas de síntesis puede compararse mediante los resultados que se muestran en la Figura 8. Puesto que *FPGA Compiler 2* no es capaz de inferir memorias, se incluyen solo datos correspondientes a sistemas no programables. Para MFCs basados en memoria, al realizar la síntesis mediante *XST* se ha considerado tanto la opción de inferir ROM como la de sustituir su operación mediante lógica combinacional. La

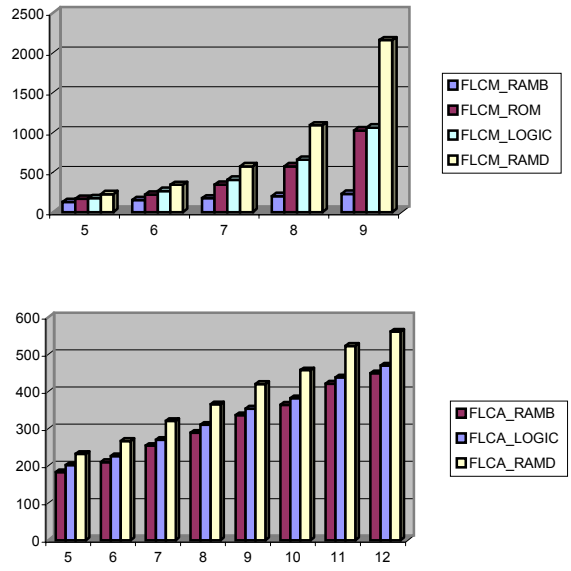

Figura 6: Número de Slices utilizados por sistemas difusos con MFCs basados en memoria (a) y aritméticos (b).

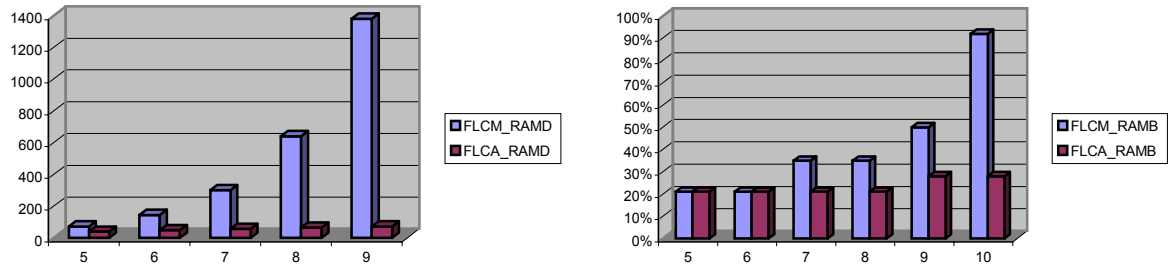

Figura 7: Utilización de memoria en las distintas opciones de implementación de MFCs.

herramienta de *Synopsys* solo admite esta última opción, si bien puede comprobarse que los resultados obtenidos superan ligeramente a los de la herramienta de *Xilinx*. Dado el tamaño y la estructura de la memoria necesaria para almacenar los coeficientes de los MFCs aritméticos, *XST* no infiere en ningún caso el uso de memoria ROM para implementar los sistemas difusos que emplean esta opción arquitectural.

Un último apunte relativo a los tiempos empleados por las distintas herramientas para realizar la síntesis de los sistemas. En este sentido cabe mencionar que *XST* es ligeramente más rápida que *FPGA Compiler 2* y que en la mayoría de los casos el tiempo empleado por ambas se mantiene siempre por debajo del minuto de CPU.

6. CONCLUSIONES

La disponibilidad de un flujo de diseño apoyado en herramientas de CAD que permiten pasar en cuestión de minutos desde la especificación de un sistema difuso hasta su implementación en una FPGA nos ha permitido explorar el espacio de diseño de este tipo de sistemas, analizando la influencia de distintas alternativas arquitecturales y diferentes opciones de implementación con objeto de optimizar el uso de los recursos que proporcionan los dispositivos actuales. Los resultados de este estudio resultan básicos para afrontar el desarrollo de las nuevas facilidades de síntesis hardware que incorporará la próxima versión del entorno *Xfuzzy*.

7. REFERENCIAS

[1] Passino, K. M., Yurkovich, S., *Fuzzy Control*, Addison-Wesley, 1998.

[2] Terano, T., Asai, K., Sugeno, M., Eds., *Applied Fuzzy Systems*, Academic Press, 1994.

[3] Yen, J., Langari, R., Zadeh, L. A., Eds., *Industrial Applications of Fuzzy Logic and Intelligent Systems*, IEEE Press, 1995.

[4] Baturone, I., Barriga, A., Sánchez-Solano, S., Jiménez, C. J. López, D. R., *Microelectronic Design of Fuzzy Logic-Based Systems*, CRC Press, 2000.

[5] C. J. Jiménez, S. Sánchez-Solano, A. Barriga, "Hardware Implementation of a General Purpose Fuzzy Controller", *IFSA'95*, pp. 185-188, July 1995.

[6] E. Lago, M. A. Hinojosa, C. J. Jiménez, A. Barriga y S. Sánchez-Solano, "FPGA implementation of fuzzy controllers", *DCIS'97*, pp. 715-720, Nov. 1997.

.

[7] D. R. López, C. J. Jiménez, I. Baturone, A. Barriga, S. Sánchez-Solano, "Xfuzzy: A design environment for fuzzy systems", *FUZZIEEE'98*, pp. 1060-1065, May 1998.

[8] E. Lago, C. J. Jiménez, D. R. López, S. Sánchez-Solano, A. Barriga, "Xfvhdl: A Tool for the Synthesis of Fuzzy Logic Controllers", *DATE'98*, pp. 102-107, Paris, 1998.

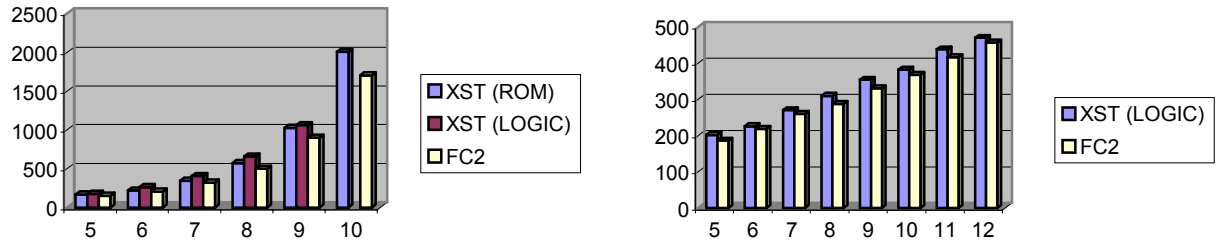

Figura 8: Comparación del número de Slices obtenido por distintas herramientas de síntesis para MFCs basados en memorias (a) y MFCs aritméticos (b).