FUZZYCOMP: UNA HERRAMIENTA PARA LA CONSTRUCCIÓN DE PROTOTIPOS DE SISTEMAS DIFUSOS EN MICROCONTROLADORES DE 8 BITS

JORGE BAENA

Grupo de Microelectrónica. Facultad de Ingeniería. Universidad de Antioquia.

MÓNICA VALLEJO

Grupo de Microelectrónica. Facultad de Ingeniería. Universidad de Antioquia.

JOSÉ AEDO

Coordinador del Grupo de Microelectrónica. Facultad de Ingeniería. Universidad de Antioquia.

Recibido para revisar 15 de Abril de 2002, aceptado 2 de Julio de 2002, versión final 5 de Agosto de 2002.

RESUMEN: En este artículo se presenta el desarrollo de una herramienta que permite la implementación de Sistemas Difusos (SD) en microcontroladores de 8 bits. Similar a un compilador, esta herramienta genera el código en lenguaje ensamblador para la ejecución eficiente del sistema difuso en un microcontrolador de la familia CPU08. El código es generado a partir de una especificación textual del sistema difuso. La herramienta, por lo tanto, disminuye el tiempo requerido para la construcción de prototipos de sistemas difusos en un hardware de bajo costo. La herramienta fue implementada usando C++ y se utilizaron técnicas modernas para la construcción de compiladores en su construcción. Con el fin de verificar su funcionamiento se utilizaron diferentes sistemas difusos "benchmark".

PALABRAS CLAVES: Compiladores, Programación Orientada a Objetos, Lógica Difusa, Sistemas Embebidos.

ABSTRACT: This article presents the development of a software tool that allows the implementation of Fuzzy Systems (FS) on 8 bits microcontrollers. Similar to a compiler, this tool generates the code in assembler for efficient execution of the fuzzy system on a CPU08 microcontroller family. The code is generated from a textual specification of the fuzzy system. Therefore this tool reduces the prototyping time of fuzzy systems in a low cost hardware. The tool was implemented using modern techniques for compiler construction and was built using C++. It was using several fuzzy systems benchmark in order to test the tool.

KEYWORDS: Compiler, Object Oriented Programming, Fuzzy logic, Embedded Systems.

1 INTRODUCCIÓN

El uso de los sistemas difusos se ha venido incrementando considerablemente en los últimos años, principalmente en el área de control. Este aumento en las aplicaciones ha traído la necesidad del desarrollo de herramientas de simulación y de generación de código que simplifiquen el proceso de implementación y disminuyan el tiempo de introducción de nuevos productos al mercado (*Altrock,* 1995*; Altrock,* 1999). En este trabajo se presenta la implementación de una herramienta de software que permite, partiendo de una especificación textual de un sistema difuso, la generación de un código para la construcción de prototipos de

dichos sistemas en un microcontrolador de 8 bits de la familia CPU08 (CPU08, 2001).

En el desarrollo de la herramienta se utilizaron técnicas modernas, basada en objetos, desarrolladas para la construcción de compiladores (*Holmes,* 1995). La gramática para la especificación del sistema difuso se realizó en BNF (*Holmes,* 1995) y se usó el Flex (*Paxson*) y Byacc (*Yacc*) en la construcción del "parser" y del "scanner".

Para la implementación de la herramienta se diseñó una biblioteca de objetos que permite crear una representación de la estructura de un sistema difuso como un "árbol" de objetos. Con base en esta representación inicial se realizan varias transformaciones con el objetivo de generar una nueva representación, también en forma de "árbol", que es más apropiada para la generación del código. Cada nodo (objeto) del árbol en la representación optimizada es dotado con métodos para la generación de código, de tal forma que el proceso de generación es realizado mediante un recorrido controlado por cada uno de los nodos del árbol activando los métodos de generación correspondientes. Todo el proceso de generación es controlado por un objeto que ha sido llamado "controlador". Este esquema facilita la integración de nuevos métodos de generación, permitiendo soporte para nuevos tipos de microcontroladores.

Una característica importante de esta herramienta, en su primera versión (*Home Page Fuzzycomp*), es que genera código optimizado desde el punto de vista de desempeño y del uso de los recursos de memoria (para la familia de microcontroladores CPU08 (*CPU08,* 2001)). La utilización de este tipo de microcontroladores tiene la ventaja de su bajo costo, además cuenta con varios periféricos, lo que los hace efectivos en proyectos sensibles al costo.

Para efectos de verificar el correcto funcionamiento de la herramienta se especificaron varios SD "benchmarks" y se probó su ejecución en un microcontrolador. Los mismos "benchmarks" fueron modelados en el ambiente Matlab (*Matlab,* 1998) con el objetivo de comparar los resultados de la simulación con relación a su ejecución en el microcontolador a partir del código generado.

Este artículo ha sido organizado de la siguiente forma: En la sección 2, se realiza una descripción del sistema difuso empleado. También se muestra la gramática establecida para la definición de los sistemas difusos. En la sección 3 se describe la estructura de la herramienta detallando algunos de los objetos diseñados. En la sección 4 se presenta un ejemplo de cómo usar el compilador y se muestran los resultados de la generación de código de varios sistemas "bench mark" usados para la prueba. Por último, en la sección 5 se presentan las conclusiones y el trabajo futuro.

2 ESTRUCTURA DE LOS SD

Un SD o controlador difuso típico consta de tres etapas: "Difuminado" (Fuzzification, en Inglés), evaluación de reglas y "Concreción" (Defuzzification, en Inglés). Estas tres fases deben ser diseñadas y ejecutadas en forma secuencial teniendo en cuenta los recursos disponibles en el microcontrolador. Dependiendo de la complejidad del sistema difuso los recursos del microcontrolador se pueden saturar. La complejidad del sistema difuso depende directamente del número de entradas y salidas, el número de términos lingüísticos definidos para cada entrada y salida, y el número de reglas (*Mendel,* 1995; *Deitel et al.*, 1995).

La etapa de "difuminado" mapea un valor real de entrada en un conjunto difuso. Existen dos métodos para realizar este proceso: el difuminado basado en "singletons" y le difuminado basada en "non-singleton"(*Mendel,* 1995). En este trabajo se utiliza el primer método por ser el más simple y usado en la actualidad.

Para la definición de conjunto de reglas son establecidos un conjunto de términos lingüistas asociados en cada entrada. Estos términos lingüísticos son modelados con conjuntos difusos, cada uno de los cuales es definido por su función de pertenencia. En este trabajo, con el propósito de optimizar los recursos de memoria, se representan las funciones de pertenencia con base en sus parámetros. De esta forma, se pueden utilizar funciones trapezoidales y triangulares. Cada término es identificado por intermedio de un índice. Para definir las funciones de pertenencia se deben suministrar cada uno de los puntos que la componen. Por

ejemplo para el caso de una función triangular definida por las coordenadas 20, 30, 40, la especificación sería:

Caliente: Input1, shape: Tshape (20, 30, 40)

donde Caliente se refiere al nombre del conjunto difuso. Input1 es el nombre de la entrada a la que pertenece y la forma es del tipo "Tshape" que es la función triangular.

Para el proceso de ejecución del conjunto de reglas, inicialmente se realiza un recorrido por todos los conjuntos difusos definidos para cada entrada verificando si el "singleton" correspondiente a esa entrada tiene una intersección no vacía en cada uno de los términos, en cuyo caso se calcula su valor. El método de cálculo se realiza por medio del punto y la pendiente correspondiente al tramo lineal que define el conjunto difuso correspondiente. Ver Figura 1. Debe notarse que con base en los puntos que definen cada función de pertenencia, la herramienta calcula las pendientes de cada uno de los tramos lineales.

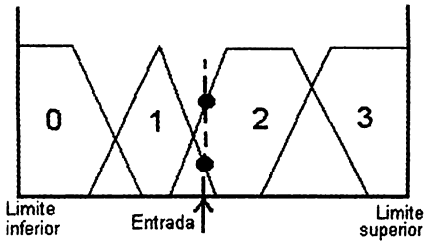

Figura 1. Forma típica de un puerto de un sistema difuso.

Una vez determinados los términos con una intersección no vacía para cada entrada y su valor, se evalúan las reglas que se pueden activar. Siguiendo este procedimiento, primero se realiza un ordenamiento de las reglas y se almacena los consecuentes correspondientes a cada regla. De esta forma se optimizan los recursos de memoria ya que no almacena los antecedentes de las reglas. Es de anotar que para las reglas que no estén definidas se almacena un indicador (0xFF), que le indica al controlador que no es un consecuente válido y no tiene aporte en la salida. El orden utilizado para el almacenamiento de las reglas tiene la estructura observada en la Tabla I. Para el caso de un sistema con dos entradas, la regla que se activa se calcula con la ecuación (1).

$$FUZZYSETS\_In2 * indiceIn\_1 + indiceIn\_2 \quad (1)$$

Donde *FUZZYSETS_In2* es el número de conjuntos difusos que tiene la entrada 2. Por ejemplo, la regla que se activa con los conjuntos (1 2), en el caso mostrado en la Tabla 1, sería 3*1+2=5. Observe que la regla 2 no está definida. De esta manera se puede acceder al consecuente sin almacenar los antecedentes. Debe notarse que las reglas que componen la base de conocimiento deben adaptarse a esta forma estándar previamente a la generación del código correspondiente. Esto constituye uno de los pasos de optimización realizados por la herramienta.

Tabla 1. Tabulación de reglas por medio de índices.

Nº Regla	Antec 1	Antec 2	Consec
0	0	0	4
1	0	1	6
2	0	2	0xFF
3	1	0	8
4	1	1	9
5	**1**	**2**	**4**
6	2	0	2
...	...	...	...

La regla que se activa por los conjuntos 1 y 2 se obtiene con la ecuación 3*1+2=5. Este sería el "offset" usado para ubicar el consecuente de la regla.

En esta versión de la herramienta se utiliza como operador de inferencia el mínimo y para la agregación se utiliza el máximo. Estos métodos se adoptaron por ser los más utilizados y más eficientes desde el punto de vista computacional.

Para el proceso de concreción se utiliza el método del centro de gravedad, el cual es simple cuando los conjuntos de salida son "singletons". Considerando este método el valor de salida del sistema difuso se calculó con la ecuación (2).

$$\text{output} = \frac{\mu_1 * Sg_1 + \mu_2 * Sg_2 + \cdots}{\mu_1 + \mu_2} \quad (2)$$

Donde μ_n es el grado de activación de las reglas y Sg_n es el valor del "singleton" de salida. Debe notarse que a pesar de ser la parte más simple desde el punto vista de programación, esta fase tiene un costo computacional alto por el número de multiplicaciones y la división que se debe realizar. Para mejorar el rendimiento, la herramienta genera un código que aprovecha la instrucción de división de la CPU y no realiza la multiplicación y la suma cuando el grado de activación es cero.

Para la especificación del SD se definió la gramática en BNF (*Holmes,* 1995), que permite definir la estructura del SD para el cual se va a generar el código. Inicialmente se definen las entradas y salidas, luego se definen los términos lingüísticos y, finalmente, se define el conjunto de reglas. En la Figura 2 se muestra la especificación de un sistema difuso de dos entradas, una salida, tres conjuntos por puerto y cinco reglas.

```
Modelo benchl;
Input1 = {-100,100};
Input2 = {-100,100};
Output = {-100,100};
Input1ef (Input1, Input2 >> Output);
term1    : Input1, shape: Sshape (49.64,64.74,100);
term2    : Input1, shape: Tshape (10.07,20.14,64.74);
Term3    : Input1, shape: Zshape (-100,-64.774,-50.35);
Term6    : Input2, shape: Zshape (-100.00,-79.85, -69.78);
Term7    : Input2, shape: Tshape (-79.85,-59.71,-40.28);
Term8    : Input2, shape: Sshape (0,59.71,100.0);
Term11   : output, shape:Singleton (-20.14);
Term12   : output. shape:Singleton (0.0);
Term13   : output, shape:Singleton (60.5);
If {Input1 is term3 and Input2 is term6} then {output is term1};
If {Input1 is term1 and Input2 is term7} then {output is term1};
If {Input1 is term1 and Input2 is term8} then {output is term2};
If {Input1 is term3 and Input2 is term6} then {output is term13};
If {Input1 is term2 and Input2 is term8} then {output is term12};
```

Figura 2. Ejemplo de archivo "prueba1.fuz", el cual contiene la descripción de un sistema difuso

3 ESTRUCTURA DE LA HERRAMIENTA

La herramienta desarrollada tiene similitudes con un compilador. Un compilador tradicional posee típicamente entre dos o cuatro fases como se muestra en la Figura 3, teniendo mayor número de fases cuando es altamente optimizado. De acuerdo con la figura un compilador clásico parte de un programa fuente, definido por una gramática, verifica su sintaxis y lo convierte en una representación intermedia. Luego el programa, en esta representación, es transformado por varias fases de optimización hasta que finalmente es generado el código para una máquina particular.

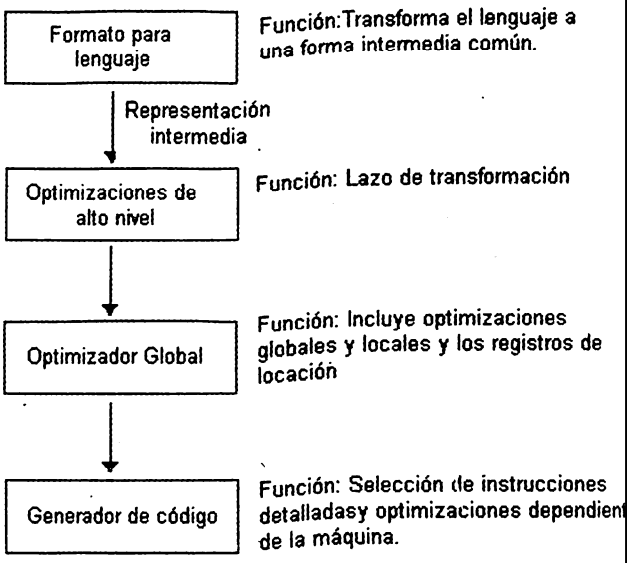

Figura 3. Esquema de un compilador clásico.

Una fase es simplemente un proceso en el cual se transforma una representación del programa en otra equivalente. Las fases de optimización son diseñadas opcionalmente y pueden obviarse cuando el objetivo es una compilación más rápida con una calidad del código aceptable.

Por otra parte, el uso de metodologías basadas en objetos simplifica de manera considerable el proceso de implementación de compiladores. Una técnica para la implementación consiste en definir una representación intermedia como un árbol de objetos, cuyos nodos son las instrucciones definidas en el lenguaje de entrada (*Matlab,* 1998).

Siguiendo este lineamiento, el primer paso para el diseño de la herramienta es la definición de la gramática para la especificación de los sistemas difusos. Para generar el "parser" (*Altrock,* 1995), la gramática para la descripción de los sistemas difusos se especificó utilizando el pseudolenguaje BNF (*Holmes,* 1995), y a partir

de esta descripción se utilizó el programa Byacc para generar el "parser". Un segmento de la especificación de la gramática se muestra en la Figura 4.

```
%token MODELOTK
%token SHAPETK
%token TSHAPE
%token EQTK /* = */
%token SHIFT/* >> */
%token SCTK /*;**/
%startModel
%%
Model: MODELOTK Ident SCTKfuzzy_part
        {$$ = new ModeloCls($2,$4);}
   | MODELOTK erro
   {soften("ERROR: Incomplete model description");}
   | error Ident SCTKfuzzy_part DOTTK
   {soften("ERROR: missing head of model");}
   | MODELOTK Ident SCTK fuzzy_part DOTK error
   {soften("ERROR: missing final point");}
;
Fuzzy_part:
InputoutputlstSCTKinoutdefSCTKlabellstSCTKrulesIstSCTK
   {$$ = new FuzzyPartCls($1,$3,$5,$7,)M;}
   | inputoutputlsterrorinoutdef
    {soften("ERROR: missing ",");}
   | inputoutputlstSCTKinoutdef error
    {soften("ERROR: missing ",");}
   | inputoutputlstSCTKinoutdefSCTK ruleslst error
    {soften("ERROR: missing ",");}
;
```

Figura 4. Segmento del archivo con la especificación de la gramática

El paso siguiente para el diseño de la herramienta es la creación del "scanner" (*Holmes,* 1995), el cual recorre el archivo fuente y obtiene los diferentes lexemas presentes en el código fuente. Se utilizó el programa Flex para generar el "scanner", como una función en lenguaje C a partir de un archivo de especificación que contiene la información de los tokens (*Altrock,* 1995) definidos dentro de la gramática. Las funciones generadas, el "scanner" y el "parser" se encapsulan (*Deitel et al.,* 1995) en la clase ScanParseCls, formando así el objeto encargado del realizar estos procesos. Ver Figura 5.

Como se muestra en la Figura 5 se diseño un objeto controlador denominado "MainControlCls" para administrar el proceso de transformación del programa fuente y el proceso de generación de código. Este objeto controlador inicialmente realiza un llamado al objeto ScanParseCls, que desarrolla las fases de "scanner" y "parser" y paralelamente realiza la creación de la representación intermedia en forma de árbol (los nodos del árbol son objetos derivados del objeto PtreeCls). Posteriormente el objeto MainControlCls realiza el llamado al árbol para que realice los métodos de optimización y generación de código.

La clase ErrorCls está relacionada con el objeto ScanParseCls y permite manejar los errores de sintaxis y de consistencia en la descripción del SD.

El objeto LexTokCls almacena los identificadores detectados por el "scanner" y el objeto OptionsCls almacena las diferentes opciones dadas a la herramienta en la línea de comandos al ser ejecutada. Adicionalmente se diseñaron clases que otorguen ciertos atributos especiales a objetos por medio de herencia (*Paxson*), por ejemplo la estructura de listas enlazadas la cual es necesaria en casi todos los nodos del árbol. Todos los objetos del árbol (nodos) son derivados de una clase base llamada "PTreeCls" que posee las características básicas del nodo. En la Figura 6 se muestra la estructura de varios objetos diseñados para establecer la representación del SD, los cuales son descendientes de "PTreeCls".

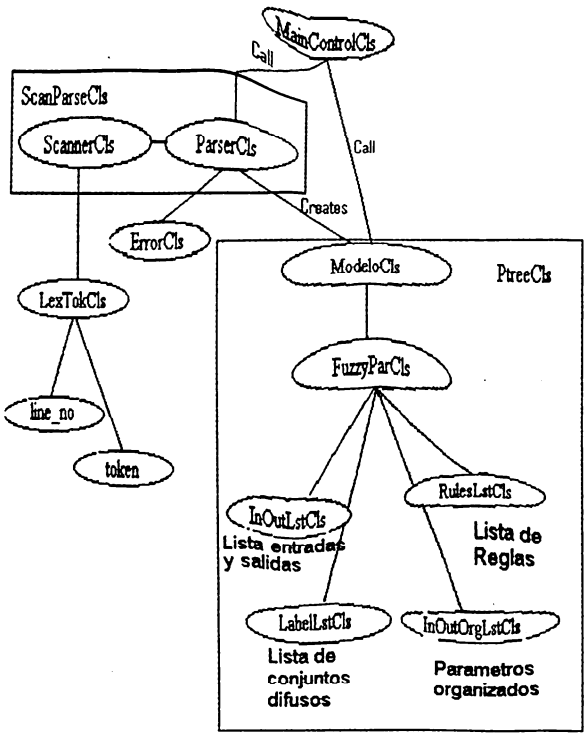

Figura 5. Esquema del compilador fuzzy utilizando objetos

Como se observa en la Figura 5, la estructura del SD se establece en el objeto ModeloCls. Este objeto tiene como dato principal un objeto FuzzyParCls, el cual es el más importante del árbol ya que contiene todos los parámetros del sistema fuzzy por medio de cuatro objetos diferentes definidos internamente. La clase InOutLstCls contiene la definición de los puertos del sistema con sus rangos válidos. La clase LabelLstCls contiene los conjuntos difusos definidos, con su correspondiente tipo y parámetros. La clase RuleLstCls almacena las reglas definidas. Por último la clase InOutOrgLstCls contiene los conjuntos ordenados por puertos y en orden numérico. Estas clases se pueden observar con mayor detalle en la Figura 6.

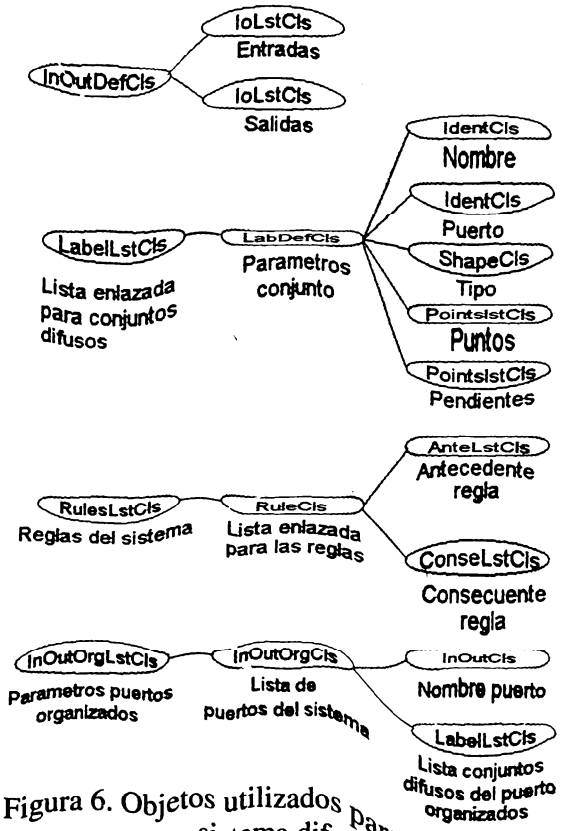

Figura 6. Objetos utilizados para representar el sistema difuso.

Cuando se crea el objeto tipo FuzzyParCls este crea sus objetos miembro y realiza verificación de su consistencia. Cuando ya tiene todos sus datos crea el objeto tipo InOutOrgLstCls, el cual es el árbol optimizado necesario para generar el código de salida adecuadamente.

Además de realizar un ordenamiento y adecuación de los parámetros, la herramienta realiza un análisis del sistema difuso, de modo que obtiene características relevantes para la generación de código. Este análisis permite realizar diversas optimizaciones al código que se va a generar. Por ejemplo, la forma de la función de pertenencia de los conjuntos difusos (triangular, trapezoidal, etc.) determina una generación de código diferente, al igual que si el punto de cruce entre dos funciones de pertenencia establece cierta simetría o no.

4 PRUEBA DEL COMPILADOR

La herramienta *Fuzzycomp Rev1.2* está disponible en ambiente Linux y presenta las siguientes restricciones: traslape máximo de dos, capacidad para funciones de pertenencia triangulares, trapezoidales, S y Z en la entrada y sólo "singletons" en la salida.

Para realizar comprobación del código generado se utilizaron varios SD de prueba. En la Figura 2 se muestra una porción de la descripción de uno de estos sistemas. Los términos "modelo", "inoutdef", "if", "is", "and", "then", "Sshape", "Zshape", "Tshape", "TRshape", "Singleton" son las palabras reservadas del lenguaje de descripción. Según este lenguaje, la descripción del sistema consta de cinco secciones principales: definición del modelo, definición de puertos y su rango, definición de entradas y salidas, definición de conjuntos difusos y por último definición de las reglas. Cada sección termina en punto y coma. En el caso que no se sigan estas reglas o exista inconsistencia en los datos, la herramienta genera un mensaje de error, indicando el tipo de error y la línea donde fue encontrado. El archivo ASCII con la especificación del SD debe poseer extensión ".fuz".

La herramienta genera un archivo "nombre.asm" donde "nombre" corresponde al identificador del modelo compilado. Por ejemplo, para generar el código del SD descrito la figura 2 se ejecuta en la consola "*fuzzycomp – as prueba1.fuz*". Una vez ejecutado se genera el archivo *bench1.asm*, el cual se puede ensamblar con cualquier ensamblador para CPU08. Un

segmento de este archivo es mostrado en la Figura 7. La opción "–as" indica al compilador que genere el archivo asm y le adicione código para comunicación por puerto serial, de modo que se pueda probar inmediatamente en un microcontrolador que tenga módulo para comunicación serial (*Valvano,* 2000) con cualquier aplicación tipo terminal.

La estructura del código generado por la herramienta consta de una rutina principal que va llamando otras subrutinas. Este esquema permite agregar fácilmente otras subrutinas al código sin producir errores en el algoritmo difuso.

```
*********MAIN PROGRAM********
        org         ROM_INIT
MAIN:               lhx     #RAM_TOP
        txs
        jsr         CONFIGURE
        jsr         WELCOME
FUZZY_SYSTEM:
        jsr         GET_INPUT
        sta         INPUT_0
        jsr         GET_INPUT
        sta         INPUT_1
        jsr         FUZZY_ENGINE
        lda         OUT_0
        jsr         PUT_OUTPUT
        bra         FUZZY_SYSTEM
```

Figura 7. Segmento archivo "bench1.asm", generado por la herramienta Fuzzycomp con la descripción de la Figura 2.

Se realizaron pruebas con SD con diferentes tipos de estructura. Se graficaron las superficies de control de cada uno de los sistemas y se compararon con las obtenidas utilizando Matlab, con el objeto de verificar su correcto funcionamiento. En las Figuras 8, 9, 10 y 11 se muestran las superficies obtenidas con dos de los SD utilizados en las pruebas.

En la Tabla 2 se muestra una comparación de desempeño y recursos de memoria en cuatro sistemas de prueba, con relación a una implementación de un algoritmo similar generado con un compilador de C (*Home Page Fuzzycomp*).

Tabla 2. Medidas de rendimiento de Fuzzycomp.

Modelo	Número entradas	Conjunto entradas	Reglas	Memoria ROM (bytes)	Ciclo CPU (Max)
Benc	2	5-5	20	843	845
Benc	2	5-5	17	549	843
Benc	3	7-5-3	105	916	1297
Benc	2	7-5	35	568	739
Bench 3 en C	2	7-5	35	1973	3437

Los ciclos de la CPU pueden variar ligeramente. La frecuencia máxima de la CPU08 es de 8 MHz (*CUP08, 2001*).

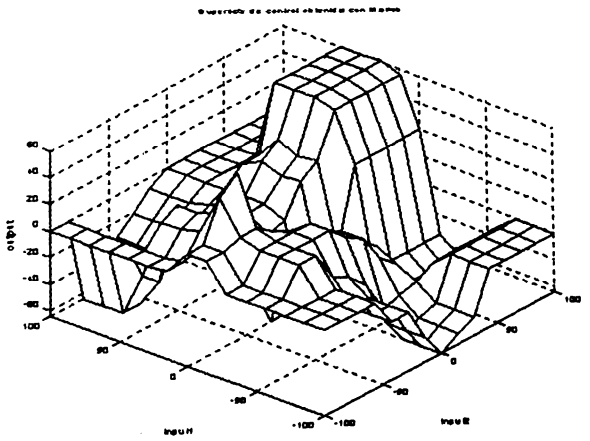

Figura 8. Superficie de control para el modelo bench1 obtenida con Matlab.

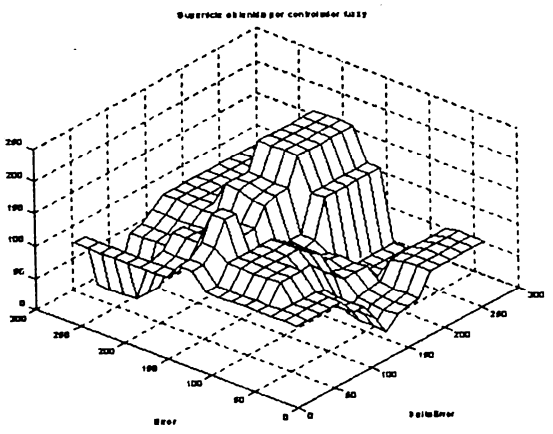

Figura 9. Superficie de control para el modelo bench1 obtenida con el código generado por el compilador.

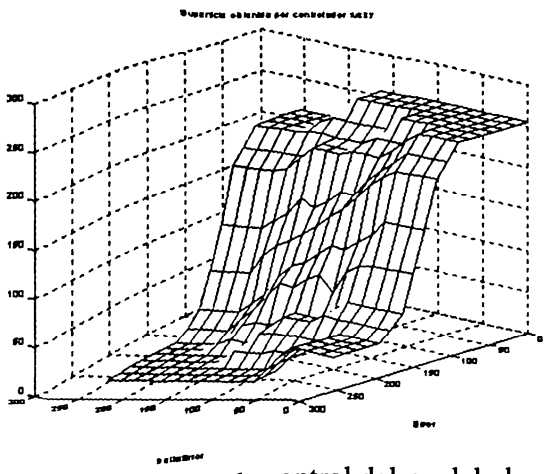

Figura 10. Superficie de control del modelo bench3 obtenida con el código generado por el compilador.

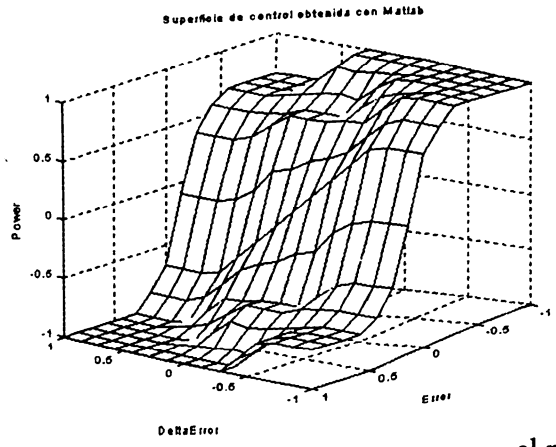

Figura 11. Superficie de control para el modelo bench3 obtenida con Matlab.

5 CONCLUSIONES

La herramienta *Fuzzycomp* ha demostrado su efectividad para la construcción rápida de prototipos de sistemas difusos en hardware de bajo costo.

En el proceso de desarrollo de una aplicación basada en lógica difusa, los cambios realizados en el conjunto de reglas no implican grandes esfuerzos en el rediseño del prototipo, ya que la generación de código se realiza de manera automática.

El código generado por la herramienta presenta un buen desempeño y exige pocos recursos de memoria, lo que permite el desarrollo de aplicaciones donde se requiere satisfacer este tipo de exigencias.

La herramienta, se implementó usando programación orientada a objetos y su estructura permite la integración con gran facilidad de generadores de código para otras familias de microcontroladores. Esta mejora se haría integrando los métodos apropiados para cada nodo del árbol previamente diseñado.

Como trabajos futuros se prevé la integración de otras familias de procesadores, la integración de otro tipo de funciones de pertenencia y la generación automática de un modelo de simulación en C del SD implementado. También se pretende mejorar la interfaz con el usuario, de modo que se vuelva un entorno de desarrollo completo.

REFERENCIAS

Altrock C. *Fuzzy logic and neurofuzzy applications explained.* Prentice all, Englewood Cliffs, USA, 1995.

Altrock C. *Fuzzylogic and neuro-fuzzy technologies in appliances.* Embedded Systems Conference, http://www.esc.com, USA, 1999.

CPU08 Central processor Unit, Reference manual Digital DNA from Motorola. Motorola Inc. 2001.

Deitel H.M., Deitel P.J. *Como programar en C/C++.* Prentice Hall. México, 1995.

Forero M.G. *Introducción al procesamiento digital de imágenes.* Universidad Nacional de Educación a Distancia. UNED. Madrid. España. 2001.

Forero M.G., González M. *Fusión automática de imágenes a través de la obtención de sinogramas.* Revista Ingeniería e Investigación, No. 40. Universidad Nacional de Colombia. Agosto 1998.

Hearn D., Baker P. *Gráficas por computadora.* Prentice Hall Hispanoamericana. Segunda Edición. 1995.

Holmes. J. *Object-oriented compiler construction.* Prentice Hall, 1995.

HomePage Fuzzycomp: http://microe.udea.edu.co/proyectos/fuzzycomp

Matlab *Fuzzy Logic toolbox.* User guide, The

Math Work Inc, 1998.
Mendel. M.J. *Fuzzy Logic Systems for Engineering: a Tutorial.* Proceedings of the IEEE, 83, (3), 1995.
Paxson V. *Flex – fast lexical analyzer generator.* Manual en el sistema operativo Linux.
Pittman, T., Peters, J. *The art of compiler design.* Prentice Hall. U.S.A. 1992.
Valvano, J. *Embedded Microcomputer Systems.* Brooks/Cole Thomson Learning, U.S.A. 2000.
Yacc – *an LALR parser generator.* Manual en el Sistema operativo Linux.