Arquitectura Básica para Controladores de Lógica Difusa a Programarse en FPGAs

Juan C. Herrera Lozada,
jcrls@ipn.mx
Ma. de Lourdes Olvera Cárdenas,
lolvera@ipn.mx
Ma. Teresa Lozano Hernández.
tlozanoh@ipn.mx
Profesores del CIDETEC-IPN

Este artículo muestra el diseño de una arquitectura básica de un controlador de lógica difusa (FLC – Fuzzy Logic Controller) con modelo orientado a cálculos, a implementarse en un FPGA. La descripción del funcionamiento se realizó alternando entre VHDL y Verilog.

1. Introducción

Las técnicas de la lógica difusa son utilizadas constantemente para resolver problemas no lineales de control, debido a que toma múltiples valores, causando una transición gradual de uno a otro.

El proceso de lógica difusa se divide en tres interfaces, cuya participación en el proceso no cambia de orden: Fuzificación, Inferencia y Defuzificación. En la **Figura 1** se advierte que la Inferencia está supeditada por los valores que entrega la Fuzificación; así mismo, la Defuzificación depende de los propios de la inferencia. Por lo anterior, se afirma que el proceso difuso es secuencial por naturaleza; sin embargo, existe la posibilidad de paralelizar las tareas internas

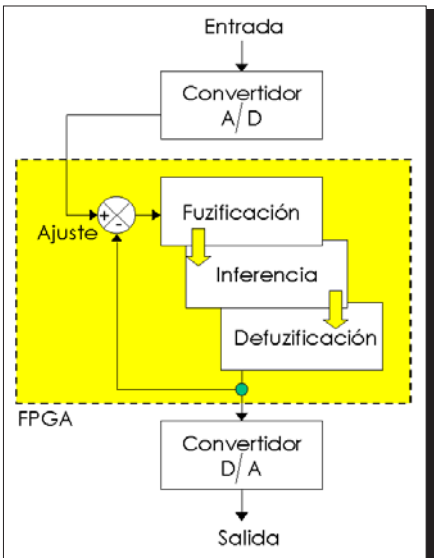

Figura 1. Proceso de Lógica Difusa.

que realiza cada una de las interfaces, propiciando una respuesta más aventajada con respecto a la velocidad de cálculos.

Proponiendo una división modular intrínseca a cada interfaz es posible definir una arquitectura que sea adaptable a cualquier sistema sin importar el campo de diligencia, haciendo extensiva la metodología de diseño.

Los controladores de naturaleza digital presentan arquitecturas determinadas por la forma en la que obtienen sus datos para comenzar las operaciones especializadas: si provienen de tablas con valores de fuzificación y defuzificación precalculados, o si se obtienen en el mismo instante con ayuda de unidades aritméticas en hardware. En este sentido se tienen dos tipos de modelos a seguir: *Modelo Orientado a Memoria* y *Modelo Orientado a Cálculos.*

El *Modelo Orientado a Memoria*, también llamado *Modelo de Obtención Indirecta de Datos Difusos*, presenta la particularidad de utilizar tablas precalculadas de valores, almacenadas en una memoria que puede ser externa o interna al controlador. Los datos pueden ser calculados directamente por el diseñador o como en la mayoría de los casos (incluidas las versiones comerciales) se utiliza un software que realiza los cálculos y crea el ensamblador que configura las características que cumplirá el controlador de acuerdo a los requerimientos propuestos por el diseñador. La **Figura 2** esquematiza un controlador orientado a memoria.

La utilización de un dispositivo de memoria para almacenar las tablas precalculadas implica el diseño de un controlador de memoria y a la vez de un sistema de sincronización por ciclos para la obtención de cada valor.

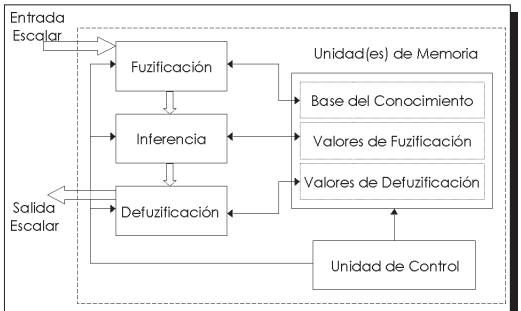

Figura 2. Modelo Orientado a Memoria.

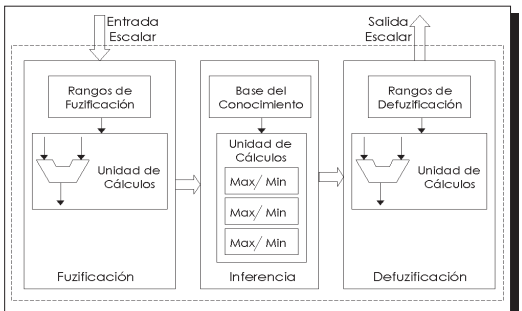

Figura 3. Modelo Orientado a Cálculos.

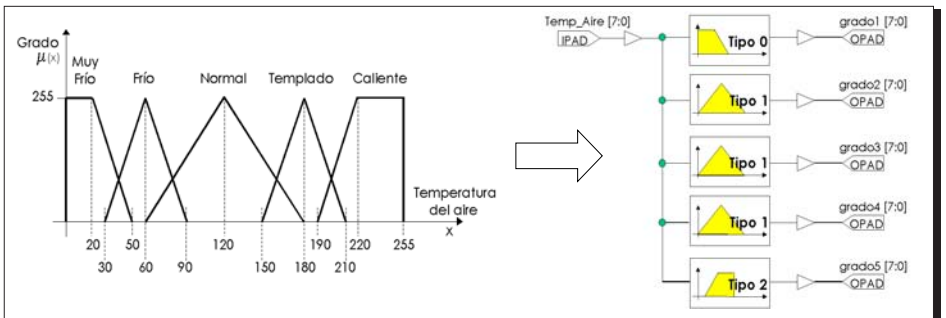

Figura 6: Idea resolutiva de la Interfaz Fuzificadora.

La velocidad de acceso para lectura y escritura, es una limitante física en este tipo de modelos; aunado a que los datos se van obteniendo uno a uno a la vez. Para sistemas mecánicos simples, este tipo de modelo es una buena opción dado que la gran mayoría de los controladores difusos dedicados se diseñan de acuerdo a esta orientación, así el mercado actual contempla un soporte muy importante en lo concerniente al software y dispositivos de propietario.

La **Figura 3** muestra una aproximación a la arquitectura de un *Modelo Orientado a Cálculos*. También se conoce como *Modelo de Obtención Directa de Datos Difusos* y tiene la característica de utilizar unidades aritméticas construidas en software o hardware, que realizan el cálculo a medida que se tiene un valor a procesar. La estimación de las arquitecturas orientadas a este modelo necesitan de algoritmos aritméticos óptimos en la implementación de cada una de las unidades de cálculo. Este modelo no requiere de cálculos previos y específicamente es el tipo de orientación que se utilizó en este documento.

2. *Diseño Planteado*

Para unificar criterios, se premedita el ejercicio de un *controlador de irrigación*. Se propone que una *válvula de irrigación* se controle con dos variables de entrada: *Temperatura del Aire* y *Humedad de la Tierra*. El FLC deberá hacer que la combinación de ambas variables determine el *Tiempo de Irrigación*. Las unidades de medida de cada variable se consideran normalizadas en un rango de *0* a *255*, para un controlador de 8 bits, sin signo y con escalares de punto fijo. Para la realización hardware, se trabaja con diseños independientes para cada interfaz y posteriormente se integran para conformar el controlador completo.

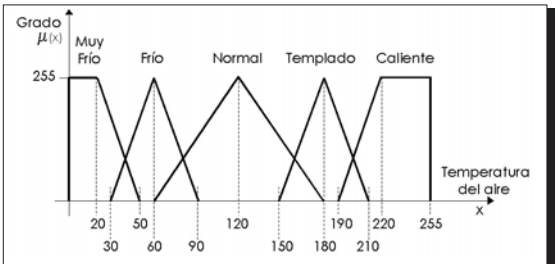

Figura 4. Funciones de pertenencia para la Temperatura del Aire.

2.1 Diseño De La Interfaz De Fuzificación

Las **Figuras 4** y **5**, muestran las funciones de pertenencia para la fuzificación de ambas variables de entrada. Esta interfaz, debe convertir la entrada real en valores difusos (grados de pertenencia), utilizando la ecuación de la línea recta.

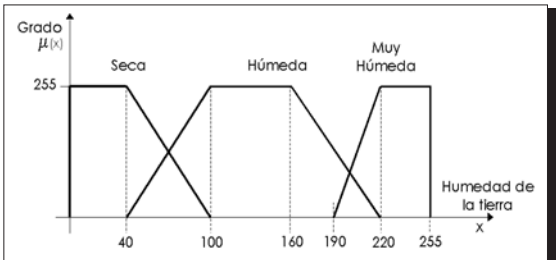

Figura 5. Funciones de pertenencia para la Humedad de la Tierra.

Una variable a fuzificar con *n* funciones de pertenencia diferentes se resuelve paralelamente con *n* diferentes bloques fuzificadores, donde cada uno calcula de manera autónoma su grado de pertenencia, tal y como se aprecia en la **Figura 6**.

Cada bloque fuzificador tiene puntos gráficos característicos que se modifican de manera simple en código. Supóngase la función de pertenencia *Frío* de la **Figura 4**. Una aproximación al código en Verilog que modela su funcionamiento se lista a continuación.

```
always  @(Temp_Aire)
begin
pendiente = 255/30;
 if (Temp_Aire<=30 || Temp_Aire>=90)
grado = 0;
 else if (Temp_Aire<=60)
grado = pendiente*(Temp_Aire - 30);
 else
grado= pendiente*(90 - Temp_Aire);
end
```

El formato textual y comprensible de la sintaxis permite modelar funciones de pertenencia de los tipos habituales (triangulares y trapezoidales) con pendientes simétricas y asimétricas, a excepción del Gaussiano debido a que su solución se apega más a medios analógicos que digitales. La **Figura 7** (siguiente página), presenta el algoritmo en hardware, codificado anteriormente.

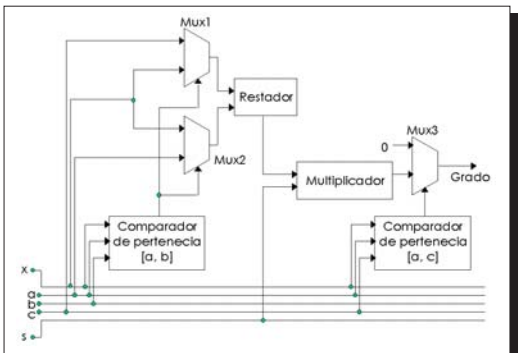

Figura 7. Módulo fuzificador individual.

2.2. Diseño de la Interfaz de Inferencia

Para modelar la interfaz de Inferencia se propone la solución Mamdani MAX-MIN, donde las reglas individuales se evalúan con el operador del mínimo de ambos grados y si comparten el consecuente, se evalúan con el máximo. Una idea abstracta se muestra en la **Figura 8**, pensando que cada una de las dos unidades fuzificadoras planteadas de inicio, aporta su respectivo grado calculado. Las reglas se evalúan todas contra todas en un esquema matricial, como lo indica la **Tabla 1**.

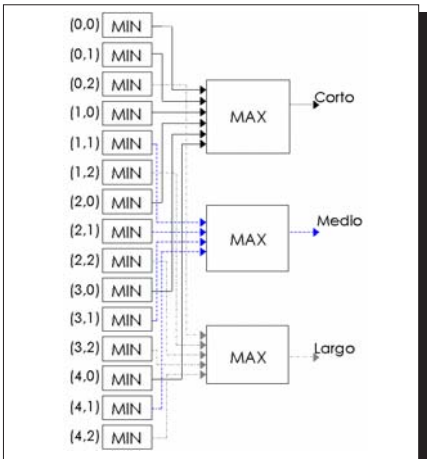

Figura 8. Idea resolutiva de la Interfaz de Inferencia.

Cada regla se conecta a su respectivo consecuente de manera personalizada, con la ventaja que los bloques MIN y MAX, no cambian de estrategia en la codificación. La **Figura 9**, representa un bloque individual.

Tiempo de Irrigación		*Temperatura del Aire*				
		Muy Frío	Frío	Normal	Templado	Caliente
Humedad de la Tierra	Muy Húmeda	Corto	Corto	Corto	Corto	Corto
	Húmeda	Corto	Medio	Medio	Medio	Medio
	Seca	Largo	Largo	Largo	Largo	Largo

Tabla 1. Reglas de Inferencia.

Una aproximación a la codificación del bloque anterior en VHDL se lista a continuación. Para describir un bloque MIN sólo basta modificar la comparación.

```
process (A,B)
begin
  if (A<B) then
    consecuente <= A;
  else
    consecuente <= B;
  end if;
end process;
```

2.3. Diseño de la Interfaz de Defuzificación

Suponiendo las tres funciones Singletons de defuzificación planteadas en la **Figura 10**, se procede a utilizar el método del Promedio de los Centros para entregar el valor real.

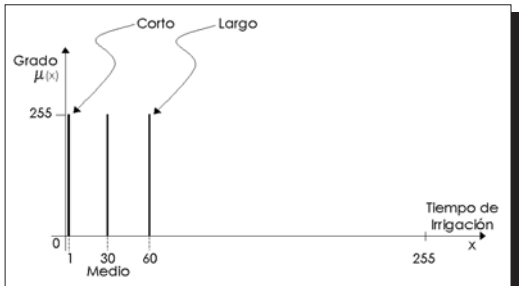

Figura 10. Singletons para la defuzificación.

La **Figura 11**, muestra la modularización del proceso defuzificador. Los consecuentes entregados por la interfaz de Inferencia (Figura 7) se multiplicarán de forma individual con su respectivo peso (valor Singleton).

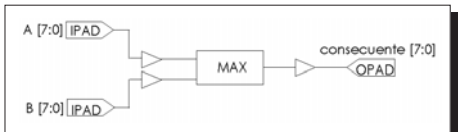

Figura 9. Bloque básico MAX, dos entradas.

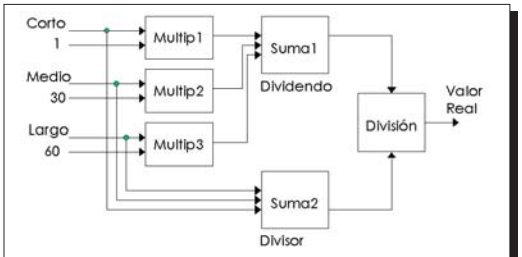

Figura 11. Idea resolutiva de la Interfaz Defuzificadora.

Describir un multiplicador en VHDL o Verilog es elemental tal y como lo muestra el listado siguiente en VHDL. Es importante mantener el tamaño del resultado.

```
Process(A,  B)
 prod <= Conv_Std_Logic_Vector(A*B)),16);
End Process;
```

Una aproximación al listado de un sumador en Verilog se lista parcialmente a continuación.

```
always @ (A or B or C)
begin
divisor = A + B + C;
end
```

Para la unidad de división por hardware, se optó por realizar una unidad síncrona y otro más asíncrona. La primera realiza restas sucesivas, una por cada ciclo de reloj, por lo que si se tiene un dividendo de 255 y un divisor de 4, tendrán que pasar 60 ciclos de reloj para visualizar el resultado. El listado parcial, en la siguiente página) muestra una aproximación a la codificación de este algoritmo de división, haciendo hincapié en el ciclo While que realiza las restas sucesivas.

Opción 1

```
always @(dividendo or divisor)
begin : divide
cont = 0;
minuendo = dividendo;
sustraendo = divisor;
if (!minuendo || !sustraendo)
      begin
      resul = 0;
      disable divide;
      end
else if (minuendo == sustraendo)
      begin
            resul = 1;
            disable divide;
      end
else
      begin
      while (minuendo >= sustraendo)
            begin
            resta = minuendo - sustraendo;
            cont = cont + 1;
            minuendo = resta;
            end
      resul = cont;
      disable divide;
      end
end
endmodule
```

Opción 2

```
always @(dividendo or divisor)
begin  :divide
integer i;
reg_cor={17'b0, dividendo};
c_divisor={9'b0, divisor};
  for (i=0; i<16; i=i+1)
  begin
  reg_cor= reg_cor<<1;
  reg_cor[32:16]= reg_cor[32:16]-c_divisor;
  guarda[15-i]= !reg_cor[32];
  if(reg_cor[32])
  reg_cor[32:16]=reg_cor[32:16]+c_divisor;
  resul=guarda[7:0];
  end
disable divide;
end
```

En la segunda opción, se utiliza el algoritmo de corrimientos constantes que permite obtener el dato de manera inmediata.

3. *Integración e Implementación*

La implementación del controlador se realizó sobre un FPGA XC4010XL de Xilinx con resultados de cálculo satisfactorios. La **Figura 12** representa una aproximación al diagrama esquemático, indicando puntualmente la modularización del diseño.

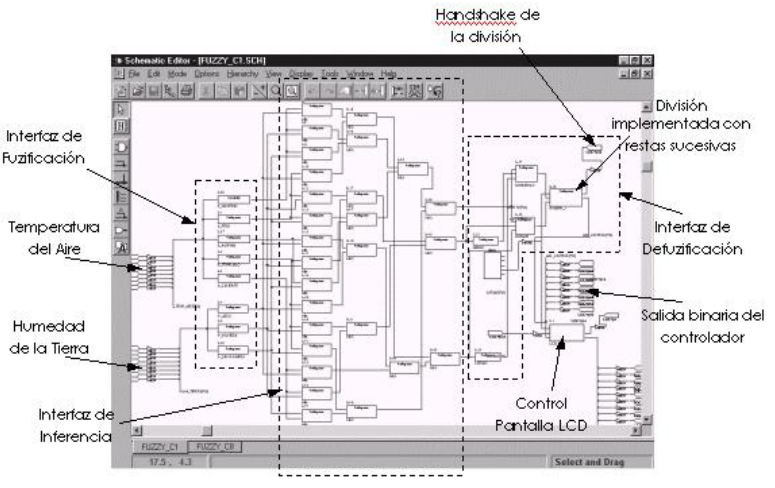

Figura 12. Diagrama esquemático del controlador diseñado.

La **Figura 13** muestra la tarjeta conectada con sus interfaces de entrada y salida. Para introducir los datos y validar resultados se utilizaron ADCs e interruptores convencionales y una pantalla LCD genérica. La unidad adquisitiva de datos, así como los acoplamientos exteriores del FLC, no se consideran elementos de discusión en los alcances de este trabajo.

Los códigos diseñados se sintetizaron con ayuda del software Foundation de Xilinx en su versión 4.1i, obteniéndose 331 CLBs (Módulos Lógicos) para el controlador con división síncrona y 840 CLBs para el propio de la división asíncrona. La frecuencia de trabajo es de hasta 6MHz, siendo suficientes 16 KHz para elementos de respuesta mecánica.

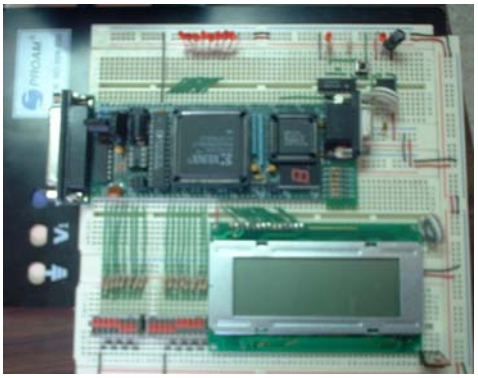

Figura 13. Tarjeta con interfaces de entrada y salida.

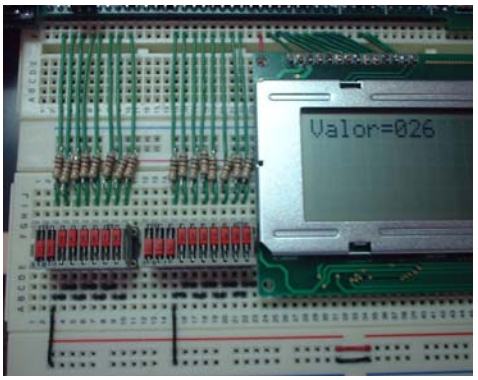

Figura 14. comprobación Experimental

La **Figura 14** muestra una validación de resultados, comprobados experimentalmente.

4. Conclusiones

Se logró implementar un controlador basado en lógica difusa, que presenta una arquitectura orientada a cálculos paralelos, que es flexible a modificaciones inmediatas.

Todo el proceso difuso se puede ejecutar de manera asíncrona o bien, síncrona, en función del tipo de unidad divisora que se diseñe. En el caso de una unidad síncrona, se necesitan tantos ciclos de reloj como restas realice. La gran desventaja de la unidad de división mediante la segunda opción es que el hardware se incrementa en un 300% en comparación a la versión síncrona; sin embargo, es capaz de entregar el resultado de manera inmediata.

5. Referencias

[1] Kandel Abraham, "Fuzzy Hardware", Kluwer Academic Publishers, 1998.

[2] Sebastian Michael J., *"Application – Specific Integrated Circuits"*, Addison Wesley, 1998.

[3] *"Synopsis, FPGA Express with Verilog HDL and VHDL, Reference Manual"*, Xilinx in line, 1999.