Diseño de Control Difuso Usando Promedio de Pesos e Implementado con Lenguaje Verilog

Romeo Urbieta Parrazales
rurbieta@cic.ipn.mx
Profesor del CIC-IPN

Enrique Guzmán Ramírez
eguzman@mixteco.utm.mx
Instituto de Electrónica y Computación, Universidad Tecnológica de la Mixteca

Los controladores que son tradicionalmente implementados en microcontroladores solamente pueden incorporar superficies de control abruptas. El propósito de este trabajo es implementar un control difuso con superficies mas finas dentro de un FPGA. El FPGA ha permitido a los diseñadores de control crear mejores diseños de prueba, hacer modificaciones muy fáciles y rápidas, en un espacio de dimensiones muy reducidas e inmunes al ruido. Esta aproximación de control difuso usa un nuevo concepto de promedio de pesos para tratar de disminuir la tabla de valores difusos, aunque los tamaños de la entrada sean grandes. El hardware del prototipo se implementó a través de un sistema de desarrollo que contiene un FPGA de la familia Spartan IIE tipo PCI y su memoria PROM. Esta fue acoplada por un conector paralelo e interfaz JTAG a la PC, y por el otro lado a un módulo analógico P160. La implementación de software del nuevo algoritmo de control se forma por tres rutinas en lenguaje Verilog. La implementación de software de la superficie de control a suavizar se instaló en la PROM. El resultado final fue un sistema de control difuso con cuatro entradas de siete bits cada una y una salida de dieciséis bits.

I. Introducción

El diseño de controladores difusos basados en FPGAs puede ser muy simple, consistiendo de un FPGA, dos convertidores analógico-digital (A/D), un convertidor digital digital-analógico (D/A) para la salida y un chip ROM. El chip ROM se usa para la tabla de búsqueda difusa (LUT). La razón de tener la LUT en un chip externo es para evitar que el FPGA sea reprogramado si la superficie de control cambia, solamente el "chip" ROM externo necesitará ser cambiado. El diagrama a bloques de un controlador difuso basado en FPGAs se ilustra en la **figura 1**.

La implementación de un sistema difuso tradicional es fácilmente programada dentro de un FPGA y trabaja bien para dos entradas[1]. Sin embargo, si las entradas son incrementadas a tres, la TB (tabla de búsqueda = LUT), usando el método de implementación tradicional, llega a ser muy difícil de manejar.

El tamaño del LUT también crece exponencialmente tanto como entradas sean agregadas[2], siendo este uno de los problemas del control difuso. Se han hecho intentos para corregir el problema al combinar dos o mas entradas para el controlador difuso, tal que el número de entradas permanecerá siempre alrededor dos.

II. Diseño Digital del Algoritmo Promedio de Pesos

Arquitectura interna del nuevo algoritmo de control

La aproximación presentada usa el concepto de promedio de pesos para conservar en forma pequeña el LUT difuso, cuando el tamaño de la entrada se hace grande. Esto se lleva a cabo usando los bits 3 y 4 más significativos de cada entrada para determinar la dirección del LUT[3]. El promedio de pesos elimina los bits

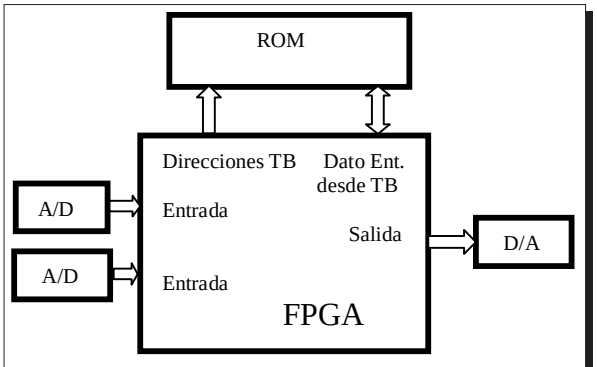


Figura 1. Diagrama a bloques de control difuso con FPGA.

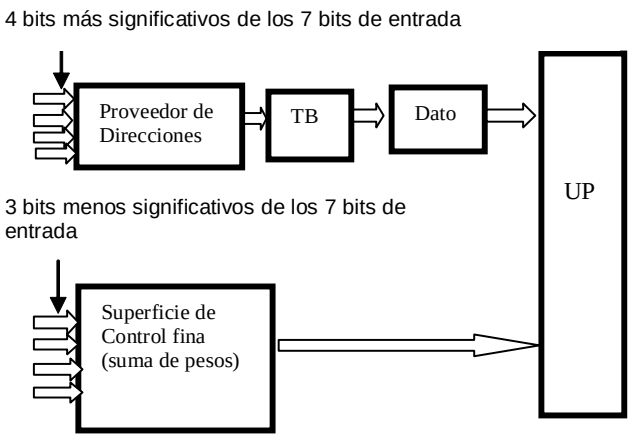


Figura 2. Diagrama a bloques de la aproximación propuesta.

remanentes para eliminar lo escarpado o abrupto de la superficie de control. El diagrama a bloques del sistema se muestra en la **figura 2**. Este nuevo método puede causar un decrecimiento en la velocidad (la cual es no crucial si se implementa en un FPGA) pero se obtiene una superficie suave, y además se puede diseñar para un gran número de entradas. En este proyecto se planteó para cuatro.

Como se ve en esta figura, hay cuatro entradas. Esto es una buena prueba para determinar si la aproximación promedio de pesos trabaja con más de dos entradas. La parte dificil del proyecto es la implementación de la nueva aproximación de suma de pesos.

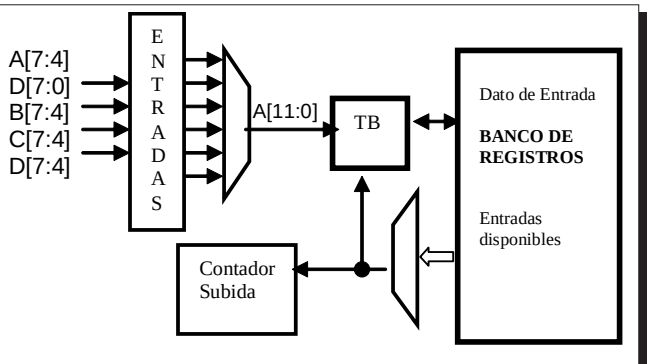


Figura 3 Diagrama del Proveedor de direcciones.

PROVEEDOR DE DIRECCIONES DE LA TABLA DE BÚSQUEDA

Con el concepto de promedio de pesos, la siguiente preocupación fue determinar si los ejes de esta superficie de control acoplarían las superficies de control vecinas. El bloque de direcciones [4] de la **figura 2** puede ser visto ahora en la **figura 3**.

Este diagrama a bloques tiene un multiplexor grande para las entradas. Para esto se empleó el software de Xilinx, que sería capaz de ubicarlo y rutearlo eficientemente. El diseño de la **figura 4** fue creado para implementar eficientemente el multiplexor, al dividirlo en cuatro multiplexores más pequeños.

PROMEDIO DE PESOS

Con esta parte del diseño, la siguiente meta fue diseñar la parte del promedio de pesos, así, aquellos valores podrían ser multiplicados por la tabla y entonces acumulados para la respuesta. El diagrama a bloques del promedio de pesos está en la **figura 5**. Aquí, los bloques de entrada son los bits menos significativos, los cuales son invertidos o pasados a través de un patrón específico.

UNIDAD DE PROCESAMIENTO

El propósito del bloque de procesamiento es multiplicar los valores promedio de los pesos propios por los valores de la tabla de búsqueda TB (LUT). El proceso entonces suma todos los

Figura 4. Diagrama a Bloques para la Estructura de Multiplexores de Entrada.

valores para crear la salida. El primer diseño del bloque de procesamiento intentado fue de 16 multiplicadores paralelos seguido por 15 sumadores. Esto causó que el diseño fuera muy grande, lo cual no es muy recomendable. La segunda aproximación de diseño al bloque de procesamiento fue hecho en una TB, tomando los dos valores de entrada y concatenándolos juntos para crear una dirección de TB construida.

Esta aproximación de diseño no es muy recomendable tampoco. La tercera aproximación de diseño fue para reemplazar los multiplicadores en paralelo con un simple multiplicador y usarlo secuencialmente. El diagrama a bloques de esta aproximación a la unidad de procesamiento puede ser visto en la **Figura 6**.

Los recursos compartidos eliminan muchas compuertas. Otra aproximación que debería conciderarse es posicionar algunos de los sumadores a lo largo de los multiplicadores, si el espacio fuera todavía suficiente.

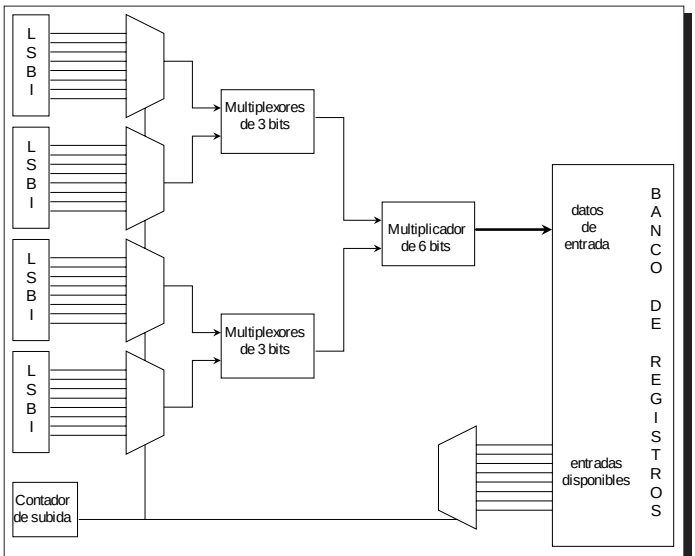


Figura 5. Diagrama a bloques del bloque de promedio de pesos.

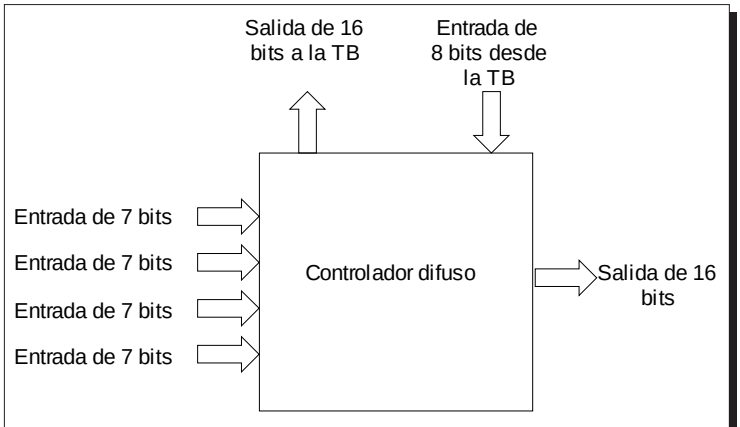


Figura 7. Diagrama final de las entradas/salidas del FPGA.

III. Implementacion del Algoritmo de Promedio de Pesos en Verilog (Fpga)

Diseño Final de Control

Este diseño necesita 68 patas disponibles de Entrada/Salida. Hay cuatro entradas de 7 bits, dos líneas de salida de 16 bits y una entrada más de 8 bits. Las cuatro entradas de 7 bits son entradas para el diseño. La primera salida de 16 bits es la línea de direccionamiento para la tabla de búsqueda y la entrada de 8 bits es la línea de datos desde la tabla de búsqueda. La salida final de 16 bits es la salida del diseño. Un diagrama a bloques de la estructura de entrada/salida se puede ver en la **figura 7**. El chip seleccionado para el diseño es el XC2S50E de 144 patas[5], con tecnología de montaje de superficie.

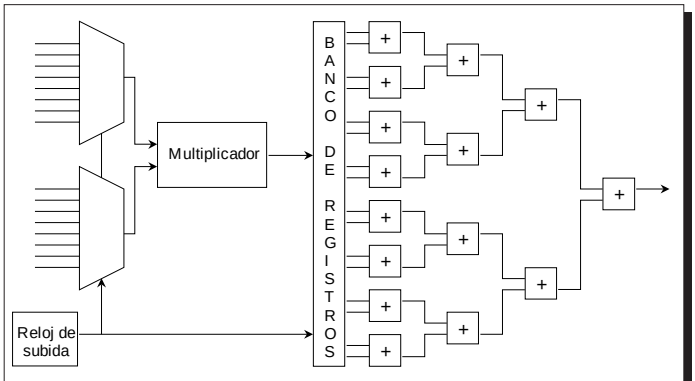


Figura 6. Diagrama de la Unidad de Procesamiento.

Conclusiones

El diseño trabaja muy bien y las salidas son muy suaves. El nuevo método de promedio de pesos es muy superior a los métodos tradicionales de control difuso. El sistema tradicional provee unas superficies rugosas y abruptas, los cuales pueden causar que el sistema ha controlar se vuelva inestable. Con superficies más suaves el sistema controlado por el nuevo controlador difuso de promedio de pesos es más estable en comparación con los métodos tradicionales. En sistemas difusos tradicionales el número de entradas son usualmente de dos a tres entradas. Este método permitió un número de entradas más grandes, en este caso fueron cuatro entradas. El método de promedio de pesos ayuda a resolver los problemas de crecimiento exponencial y complejidad de la tabla de búsqueda, permitiendo de esta manera superficies más suaves con pequeñas tablas de búsqueda.

Esta nueva tecnología de sistemas digitales es de bajo consumo de voltaje y corriente y de frecuencias muy elevadas con respecto de aquellos circuitos integrados rígidos (hechos en la fábrica), lo que lo hacen inmunes al ruido. La nueva tecnología empleada se podría pensar en similitud con aquellas personas que envían a ser un traje a la medida, así, estos circuitos pueden incorporar algoritmos de control difusos más sencillos hasta los mas complejos en una forma específica que no se encuentra en el mercado.

Referencias

[1] Pedro Diniz and Joonseok Park. *"Automatic Synhesis Data Storage and Control Structures for FPGA-based Computing Engines"*. University of Southern California.

[2] S K Kaul, R Koul, C L Bhat, I K Kaul and A K Tickoo.*"Use of a 'look-up' table improves the accuracy of a low-cost resolver-based absolute shaft encoder"*. Bhabha Atomic Research Centre, Nuclear Research Laboratory, Trombay, Mumbai 400 085, India. December 1996.

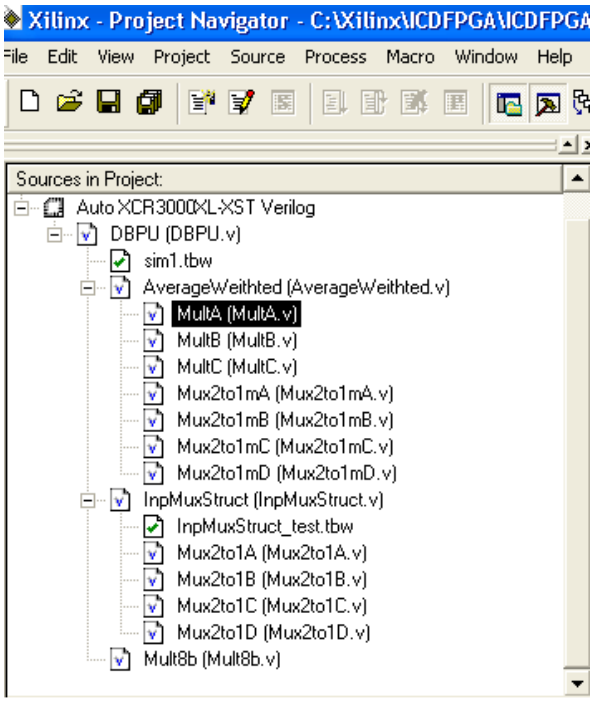
[3] Chr. W. Frey, A. Jacubasch, H.-B. Kuntze, R. Plietsch. *"Smart Neuro-Fuzzy Based Control of a Rotary Hammer Drill"*. Fraunhofer Institute for Information and Data Processing IITB. Fraunhoferstraße 1, D-76131 Karlsruhe, Germany.

[4] John M. Yarbrough. " Digital Logia". Decoders. PWS Publishing Company. Pag. 171-184. 1997.

[5] Spartan II 200 PCI Developtment Board User´s Guide. Version 1.0 April 2002.

Anexo I

Programa principal de Control difuso usando promedio de pesos en lenguaje Verilog.



Subrutinas de promedio de pesos

MultA

```
module MultA (OutmA, OutmB, MAB);

   input  [2:0] OutmA, OutmB;
   output [5:0] MAB;

   assign MAB = OutmA * OutmB;

endmodule
```

MultC

```
module MultC (MAB, MCD, MABCD);

   input  [5:0]  MAB, MCD;
   output [11:0] MABCD;

   assign MABCD = MAB * MCD;

endmodule
```

MultB

```
module MultB (OutmC, OutmD, MCD);

   input  [2:0] OutmC, OutmD;
   output [5:0] MCD;

   assign MCD = OutmC * OutmD;

endmodule
```

Mux2to1mA

```
module Mux2to1mA(A, OutmA, Counter);

   input  [1:0] Counter;      // Se ge
   input  [7:0] A;
   output [2:0] OutmA;

   reg    [2:0] OutmA;

  always @(Counter or A)
     assign OutmA = A[3:1];          /,

endmodule
```

Mux2to1mB

```
module Mux2to1mB(B, OutmB, Counter);

   input  [1:0] Counter;      // Se
   input  [7:0] B;
   output [2:0] OutmB;

   reg    [2:0] OutmB;

  always @(Counter or B)
     assign OutmB = B[3:1];

endmodule
```

Mux2to1mC

```
module Mux2tolmC(C, OutmC, Counter);

   input  [1:0] Counter;      // Se
   input  [7:0] C;
   output [2:0] OutmC;

   reg    [2:0] OutmC;

  always @(Counter or C)
     assign OutmC = C[3:1];

endmodule
```

Mux2to1mD

```
module Mux2tolmD(D, OutmD, Counter);

   input  [1:0] Counter;      // Se
   input  [7:0] D;
   output [2:0] OutmD;

   reg    [2:0] OutmD;

  always @(Counter or D)
     assign OutmD = D[3:1];

endmodule
```

SUBRUTINAS DE DIRECCIONAMIENTO DE LA PROM

Mux2to1A

```
module Mux2tolA (A, OutA, Counter);

   input  [1:0] Counter;      // Se
   input  [7:0] A;
   output [7:0] OutA;

   reg    [7:0] OutA, AuxA;

  always @(Counter or A or AuxA)
    begin

     assign AuxA = A[7:4] + 1;

      case (Counter)
         2'd0 : OutA = AuxA;  // ADI
         2'd1 : OutA = A[7:4];
      endcase
    end    // Fin del always
```

Mux2to1B

```
module Mux2tolB (B, OutB, Counter);

   input  [1:0] Counter;      // Se
   input  [7:0] B;
   output [7:0] OutB;

   reg    [7:0] OutB, AuxB;

  always @(Counter or B or AuxB)
    begin

     assign AuxB = B[7:4] + 1;

      case (Counter)
         2'd0 : OutB = AuxB;  // ADI
         2'd1 : OutB = B[7:4];
      endcase
    end    // Fin del always

endmodule
```

Mux2to1C

```
module Mux2tolC (C, OutC, Counter);

   input  [1:0] Counter;      // Se
   input  [7:0] C;
   output [7:0] OutC;

   reg    [7:0] OutC, AuxC;

  always @(Counter or C or AuxC)
    begin

     assign AuxC = C[7:4] + 1;

      case (Counter)
         2'd0 : OutC = AuxC;  // ADD
         2'd1 : OutC = C[7:4];
      endcase
    end    // Fin del always

endmodule
```

Mux2to1D

```
module Mux2tolD (D, OutD, Counter);

   input  [1:0] Counter;      // Se
   input  [7:0] D;
   output [7:0] OutD;

   reg    [7:0] OutD, AuxD;

  always @(Counter or D or AuxD)
    begin

     assign AuxD = D[7:4] + 1;

      case (Counter)
         2'd0 : OutD = AuxD;  // ADI
         2'd1 : OutD = D[7:4];
      endcase
    end    // Fin del always

endmodule
```