ISISTAN
INSTITUTO de SISTEMAS de TANDIL
San Martín ,57 (7000) Tandil
TE:54-293-40363
FAX:54-293-40362

Síntesis de Controladores Difusos Microprogramados

J.-P.Deschamps, G.Bioul,.

Resumen

En este artículo se describe un método de síntesis de controladores difusos basado en una arquitectura predeterminada y parametrizada, y en un conjunto de herramientas informáticas. El resultado de la síntesis es un circuito compuesto de bloques aptos para ser descritos en VHDL, a nivel de transferencias entre registros, y de macrocélulas compilables.

Palabras clave

Sistemas microprogramados, circuitos integrados de aplicación específica (ASIC), lógica difusa.

I. Introducción

1. Presentación del trabajo

Este trabajo forma parte de un proyecto de desarrollo de una Metodología de Diseño de Sistemas. El término Sistemas se refiere principalmente a los (así llamados) sistemas empotrados (embedded systems) en los cuales un conjunto de componentes estándar (microprocesadores, memorias, etc.) y específicos (ASICs), y un conjunto de programas, realizan una función determinada. Por ejemplo: control de procesos industriales, tratamiento de imágenes, reconocimiento de patrones, etc.

Una metodología eficiente debe partir de una especificación a la vez comprensible y no ambigua de la función del sistema. Luego, la descripción se va afinando, etapa tras etapa, hasta llegar a una definición concreta (estructural y no sólo funcional) tanto de los circuitos como de los programas.

Para llevar a la práctica tal metodología existen, incluso a nivel comercial, parte de las herramientas necesarias. Por ejemplo, un circuito digital puede ser descrito y simulado a nivel funcional. Existen herramientas de síntesis de alto nivel capaces de traducir la definición funcional a una descripción estructural describiendo transferencias entre registros, y herramientas de síntesis lógica para generar esquemas con puertas, multiplexores, biestables, etc. Sin embargo, el desarrollo de herramientas de síntesis de alto nivel sigue siendo un tema de investigación. Por otra parte, la generación de

herramientas de codiseño para la definición y simulación del sistema completo, a nivel funcional, y su posterior descomposición en circuitos (hardware) y programas (software) es un tema de investigación relativamente nuevo.

Como primera aproximación a la síntesis de sistemas se eligió el desarrollo de una metodología, con las correspondientes herramientas informáticas, para el diseño de sistemas constando de una ruta de datos (bancos de registros, unidades aritmético-lógicas, conexiones programables, interfaces de entrada y salida) controlada por un microprograma (memoria y secuenciador). Se trata pues de sistemas con un único nivel de programación - el microprograma de control. En el caso de sistemas que ejecutan siempre las mismas funciones es una solución viable y atractiva: viable porque no se trata de computadoras de propósito general que tengan que ser programadas para cada aplicación - dicho de otra manera no ejecutan más que una aplicación; atractiva porque, al diseñar una ruta de datos a medida, se puede esperar que el procesador obtenido tenga mejores prestaciones y menor costo que uno de propósito más general.

Desde luego la viabilidad de tal aproximación al diseño de sistemas digitales depende de la disponibilidad de herramientas informáticas para la traducción de una especificación inicial, de tipo funcional, a la definición de los parámetros de la ruta de datos y a la generación del microprograma.

El punto de partida es la especificación del funcionamiento deseado del sistema; el resultado esperado es el esquema lógico de un procesador a medida y un microprograma. La expresión “a medida” significa lo siguiente: en función de los requerimientos incluidos en las especificaciones - por ejemplo los tiempos de cálculo, los caudales mínimos, el costo - el resultado podría ser un procesador sencillo, de bajo costo, con una única unidad aritmético-lógica, ejecutando un microprograma relativamente largo; a la inversa podría ser también un procesador de alto costo, con varias unidades aritmético-lógicas capaces de efectuar operaciones complejas, ejecutando un microprograma relativamente corto.

En la actualidad se ha desarrollado una metodología completa dentro del marco restringido siguiente: (1) síntesis de sistemas materializando algoritmos sin bifurcaciones (esquemas de cálculo); (2) ruta de datos con una única unidad aritmético-lógica.

Como banco de prueba de la metodología se aplicó la misma al desarrollo de controladores difusos [11]. Esta elección se justifica, en primer lugar, por el hecho de que son sistemas que pueden ser descritos mediante algoritmos sin bifurcaciones y, en segundo lugar, porque existe una demanda para este tipo de controlador. Últimamente empezó a utilizarse esta estrategia de control para diversas aplicaciones concretas, en especial para productos de gran consumo (electrodomésticos, cámaras fotográficas, automóviles, etc.). El controlador puede ser realizado mediante un programa ejecutado por un microprocesador / computador / controlador dedicado o de propósito general, o por un circuito específico (ASIC, FPGA). Esta segunda opción es la que se estudia en este trabajo. Otros ejemplos de materialización se describen en las referencias [1] a [8]. La selección final entre una u otra solución depende de criterios económicos (costo, número de unidades producidas), técnicos (prestaciones, tamaño, consumo) y de estrategia empresarial (protección contra la copia, proveedores disponibles, plazos).

2. Justificación de la arquitectura elegida

Para desarrollar un sistema digital materializando un esquema de cálculo una posible solución consiste en utilizar una herramienta clásica de síntesis de alto nivel, basada en algoritmos de planificación (scheduling) y asignación de recursos y memorias, y una herramienta de síntesis lógica. Como fase preliminar del proyecto se hicieron diversas pruebas con dos programas de síntesis de alto nivel (uno de ellos basado en el algoritmo Famos [9], y el otro formando parte del paquete de síntesis de alto nivel Fidias [10]), y con una herramienta comercial de síntesis lógica. Se diseñaron varios controladores difusos. En el caso de controladores de altas prestaciones, requiriendo el funcionamiento en paralelo de varias unidades aritmético-lógicas, los resultados obtenidos fueron satisfactorios. En cambio, para controladores que procesan un número relativamente reducido de reglas, y no requieren una frecuencia de muestreo (throughput) muy alta - caso de la mayoría de los controladores - los resultados fueron decepcionantes.

Por ese motivo se adoptó una estrategia diferente, más sencilla y que, a la postre, resultó ser más eficiente en el caso de controladores de complejidad media/baja. Para ser más concreto se trataría de controladores para los cuales el producto (número de reglas con dos antecedentes y un consecuente) * (frecuencia máxima de muestreo) no excedería 10^6 (por ejemplo 100 reglas y 10 kHz).

El sistema consta de una ruta de datos parametrizada y de una unidad de control ejecutando un microprograma sin bifurcaciones. El diseño de la ruta de datos se reduce a la colocación (instanciación) y a la interconexión de dos bloques parametrizados descritos en VHDL (unidad aritmético-lógica y puertos de salida) y de dos memorias compiladas (las tablas de las funciones de pertenencia y el banco de registros de doble puerto). La unidad de control contiene un generador de fases (descrito en VHDL e independiente de la aplicación), un contador de microinstrucciones (bloque VHDL parametrizado) y una memoria de microprograma (ROM compilada). El diseño de la unidad de control se reduce prácticamente a la generación del contenido de la ROM de microprograma. Para ello se han desarrollado una serie de herramientas descritas en [12]. Para la síntesis lógica de los bloques descritos en VHDL (unidad aritmético-lógica, puertos de salida, generador de fases, contador de microinstrucciones) se puede utilizar una herramienta comercial. Sin embargo dichos bloques tienen una estructura iterativa (bit slice) con lo cual la relación (número total de puertas) / (número de puertas realmente instanciadas) es muy alta. Por tanto la síntesis lógica puede también ser realizada de forma manual en un tiempo razonable.

Las principales etapas del diseño son las siguientes: (1) definición del esquema de cálculo y generación del microprograma de control; al termino de esta etapa habrán sido definidos los parámetros de los bloques y los contenidos de las ROM compiladas. (2) síntesis lógica de los bloques descritos en VHDL. (3) compilación de las macrocélulas (memorias). (4) ensamblaje de todos los bloques.

3. Entorno de codiseño

Tal como se mencionó antes el objetivo final del proyecto es el desarrollo de una Metodología de Diseño de Sistemas. Para ello es necesario tener la capacidad de especificar y simular sistemas completos incluyendo circuitos, programas, sensores, dispositivos eléctricos y mecánicos, etc. Como primera aproximación a este objetivo se utilizó el entorno de codiseño Ptolemy para la especificación inicial - definición de funciones de pertenencia y de reglas de inferencia. Este mismo entorno se utiliza para el

test de la placa (controlador integrado en un ASIC y circuitos de interfaz). Esta primera experiencia de simulación a nivel de sistema se describe en [14].

II. Arquitectura del circuito

1. Estructura general

La estructura general se muestra en la figura 1. A continuación se describe brevemente la función de los principales bloques:

* El bloque *tablas* contiene las funciones de pertenencia (funciones de las variables de entrada) y las funciones de decodificación (funciones de la variable interna *w*). La variable de control *tabla* selecciona una función particular. El número de bits *m* elegido para codificar el intervalo *[0,1]* define el tamaño de las variables (palabras) internas.

* El banco de registros almacena todas las variables internas del esquema de cálculo. Su organización interna y la definición de las señales de sincronización (ver el apartado 2) están pensadas para que en un mismo ciclo del microprograma se puedan leer el contenido de dos registros *R(j)* y *R(k)* y escribir en un registro *R(i)*. La señal de control *t/ual* permite seleccionar el originen del dato que se escribe en el registro *R(i)*: puede ser el valor de una función de pertenencia o de decodificación (*t/ual* = 1) o el resultado de un cálculo efectuado por la unidad aritmético-lógica (*t/ual* = 0).

* La unidad aritmético-lógica contiene tres biestable *cent, xent* y *s/r* (sumar/restar) controlados por las variables de control *ff(1:0)* y *sh*. Las variables de control *func(3:1)* definen la función realizada por la unidad aritmético-lógica (sumar, restar, etc.). La salida *serie* es utilizada por el algoritmo de división.

* El bloque *puertos* contiene un registro de desplazamiento que convierte el resultado de una división de serie a paralelo, y tantos latches como funciones de salida. La señal de control *sh* se utiliza para añadir un nuevo bit del cociente al contenido actual del registro de desplazamiento, y la señal de control *out* para transferir un dato del registro de desplazamiento al puerto de salida seleccionado. Cuando *out = 00...0* no se realiza ninguna transferencia.

* La memoria de microprograma genera todas las señales de control recogidas en la tabla I.

i: dirección de escritura
j: dirección del primer operando
k: dirección del segundo operando
tabla: selección de una función de pertenencia o de decodificación
t/ual: origen del dato que se escribe
ff: control de los biestables de la unidad aritmético-lógica
sh: control de los biestables de la unidad aritmético-lógica y del registro de desplazamiento del bloque *puertos*
func: operación de la unidad aritmético-lógica
in: control del latch de entrada
out: control de los latches de salida

Tabla I

La especificación completa del circuito se reduce a la definición de los parámetros recogidos en la tabla II, a la generación del contenido de la memoria de microprograma, y a la organización interna y programación del bloque *tablas* (ver apartado 3).

n_d :	número de bits de direccionamiento de los registros
n_t :	número de bits de selección de las funciones de pertenencia y de decodificación
n:	número de entradas
$m_1, m_2, ..., m_n$:	número de bits de las entradas
m:	número de bits de las variables internas
r:	número de salidas
$s_1, s_2, ..., s_r$:	número de bits de las salidas
n_s:	número de bits de selección de las salidas
p:	número de estados del contador de microprograma
q:	número de bits del contador de microprograma

Tabla II

2. Sincronización

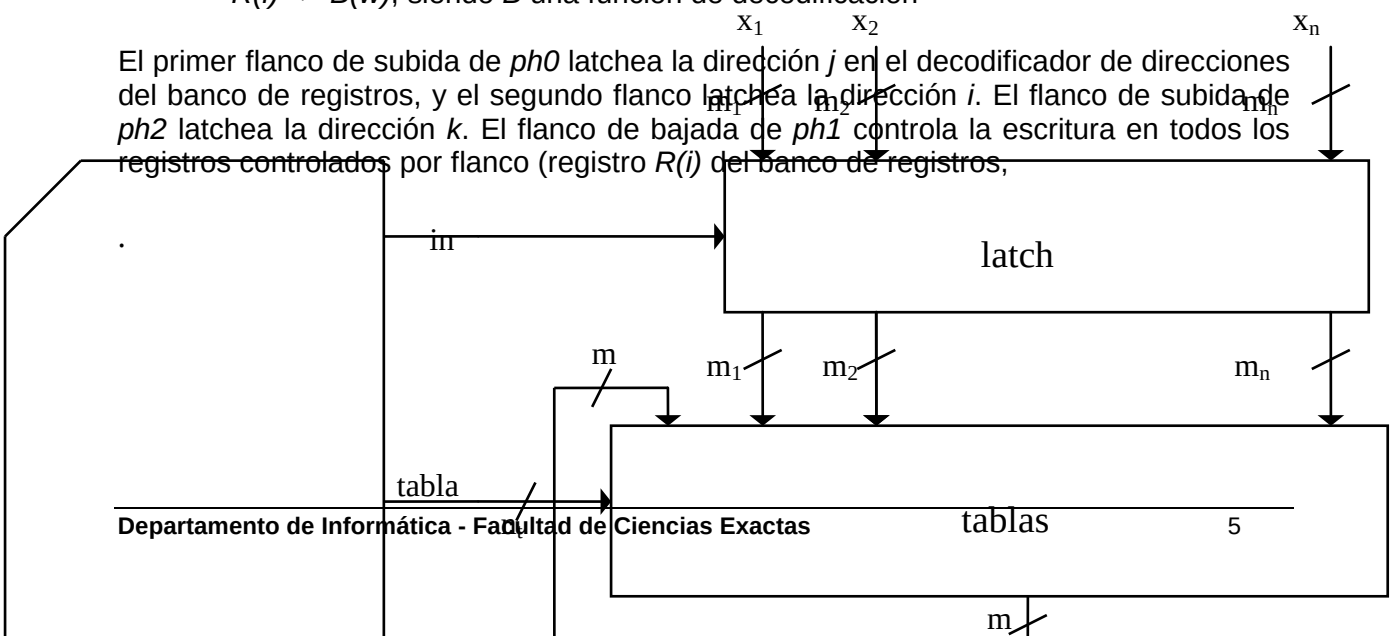
La ejecución de una microinstrucción se descompone en cuatro fases:

* búsqueda de una nueva microinstrucción,
* lectura de los operandos *R(j)* y *R(k)*,
* cálculo o lectura de una tabla,
* escritura del resultado de la microinstrucción en *R(i)* y en el registro de desplazamiento de salida, actualización de los biestables de la unidad aritmético-lógica, actualización del contador de microprograma.

Para ello se generan tres señales de sincronización *ph0, ph1* y *ph2.* En la figura 2 se muestra un cronograma correspondiente a la ejecución de las transferencias funcionales

R(i) <= *F[R(j),R(k)]*, siendo *F* una función de la unidad aritmético-lógica,
R(i) <= $A(x_l)$, siendo *A* una función de pertenencia,
R(i) <= *B(w)*, siendo *B* una función de decodificación

El primer flanco de subida de *ph0* latchea la dirección *j* en el decodificador de direcciones del banco de registros, y el segundo flanco latchea la dirección *i*. El flanco de subida de *ph2* latchea la dirección *k*. El flanco de bajada de *ph1* controla la escritura en todos los registros controlados por flanco (registro *R(i)* del banco de registros,



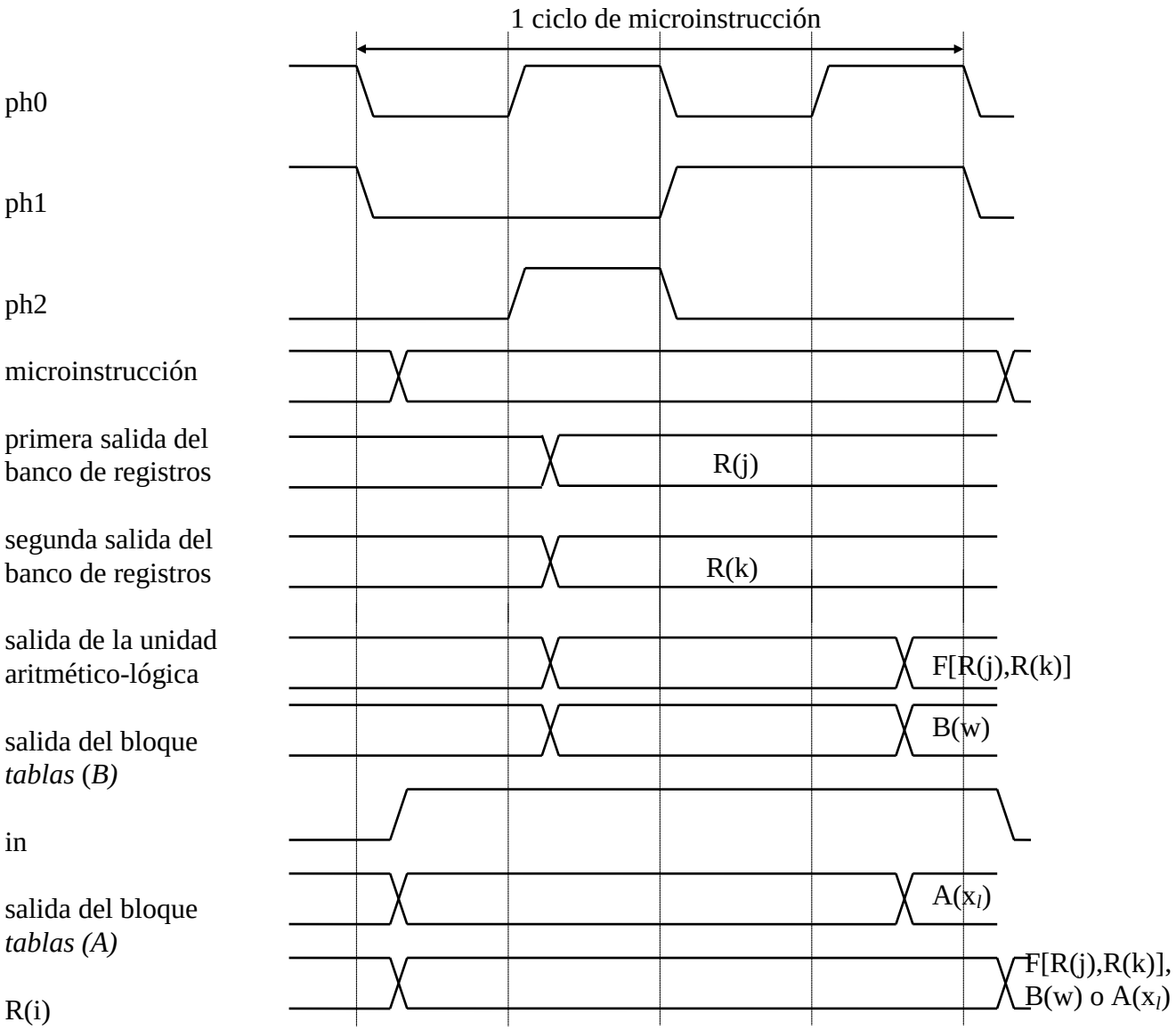


Figura 2 Cronograma de las señales

contador de microprograma, biestables de la unidad aritmético-lógica, registro de desplazamiento de salida).

Se supone que el registro *R(0)* del banco de registros contiene el valor *0.* Es necesario para algunas operaciones, por ejemplo para alinear los operandos en caso de cálculo de un cociente. Una de las primeras microinstrucciones consistirá pues en escribir *0* en *R(0)*, utilizando la operación *cero* de la unidad aritmético-lógica. Por tanto para ejecutar una microinstrucción que no modifique el estado del banco de registros basta con escribir *0* en *R(0)*, razón por la cual no es necesario que el banco tenga una señal de control de la escritura.

3. Tablas

En la figura 3 se muestra la estructura interna del generador de funciones de pertenencia y de decodificación. Cada función (o tabla) ocupa una serie de posiciones contiguas de la memoria. El decodificador asocia a cada número de tabla la dirección inicial de la misma. y el número de la variable a utilizar. Por tanto el decodificador es un circuito combinacional que tiene que ser sintetizado en cada caso. Con esta arquitectura las tablas pueden tener tamaños diferentes (en función del número de bits de la variable correspondiente) y una misma tabla puede ser compartida por varias variables (puede darse el caso de que una misma función de pertenencia sirva para varias variables de entrada).

La memoria puede ser una ROM compilada o una PROM externa (en especial si se integra el controlador en una FPGA).

La ventaja de esta solución es su sencillez: la memoria es una macrocélula compilada; basta con definir su contenido; el sumador y el multiplexor tienen estructuras regulares y pueden ser definidos como bloques VHDL parametrizados; el decodificador es un circuito combinacional que podría ser sintetizado por un programa de síntesis lógica (caso de una FPGA) o integrado en una ROM compilada (caso de un ASIC).

El inconveniente de la solución propuesta es que, a veces, la memoria es muy grande. Una solución alternativa consiste en sintetizar un circuito de cálculo de las funciones de pertenencia y de decodificación en lugar de almacenar sus valores en una memoria.

4. Banco de registros

El banco de registros consta principalmente de una memoria de doble puerto (DPRAM) compilada y de unos cuantos circuitos adicionales (figura 4). El puerto *A* es de sólo lectura; la dirección correspondiente (*j*) se latchea con la señal *ph2*. El puerto *B* es de lectura y escritura; las direcciones correspondientes se latchean con *ph0*; la dirección de lectura (*k*) se selecciona cuando *ph1 = 0* y la dirección de escritura (*i*) cuando *ph1 = 1*. Se necesita un latch adicional que almacene *R(k)* cuando la dirección del puerto *B* pasa de *k* (lectura) a *i* (escritura), es decir, cuando *ph1* pasa de *0* a *1*. La escritura se produce con el flanco de bajada de *ph1*.

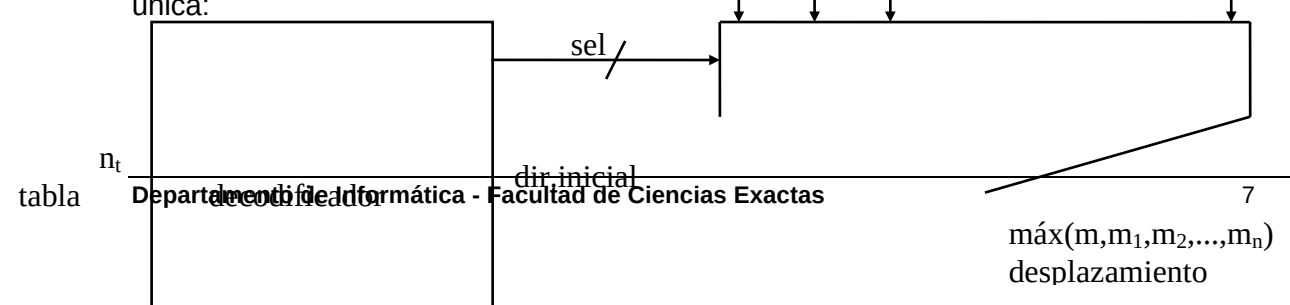
Caso de que no se dispusiera de un compilador de RAM de doble puerto, una solución alternativa consistiría en sintetizar dicha memoria con latches, amplificadores de tres estados y decodificadores de dirección.

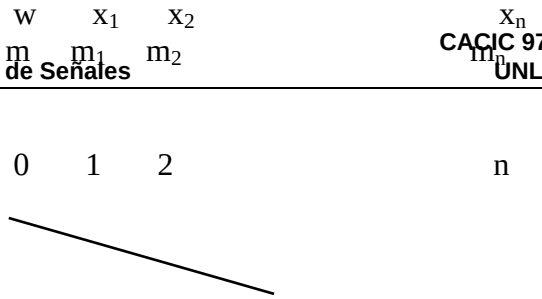
5. Unidad aritmético-lógica

Para la ejecución de un algoritmo de control difuso (ver por ejemplo [13]) se necesitan las operaciones siguientes:

máx(x,y), mín(x,y), x+y, x-y, 2.x+y, 2.x-y

Las dos primeras se utilizan para calcular los pesos w_i asociados a las funciones de decodificación B_i. La suma y la resta sirven para calcular los numeradores y denominadores cuyos cocientes son las funciones de salida; en el caso de una función única:

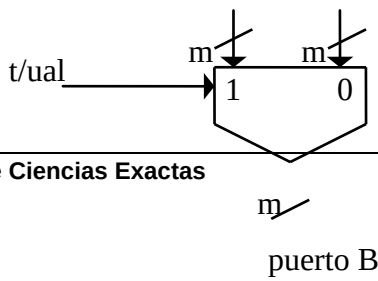




$N = w_1.B_1(w_1) + w_2.B_2(w_2) + ... ; D = w_1 + w_2 + ...$

No es necesario disponer de la multiplicación: en lugar de almacenar en *tablas* las funciones de decodificación $B_1(w), B_2(w), ...$, se almacenan directamente, con doble precisión, los productos $C_1(w) = w.B_1(w), C_2(w) = w.B_2(w), ...$. Por tanto, el cálculo de N se reduce a la lectura de las tablas $C_1 , C_2 , ...$, y al cálculo de una suma. Además, si existen ciertas simetrías entre las funciones de decodificación puede ser útil disponer de la resta: supóngase que $B_j(w) = 2.c - B_i(w)$, siendo c un valor constante (centro de simetría); si se sustituye el rango de salida, por ejemplo $[0, 2^M-1]$, por el rango $[-c, 2^M-1-c]$, entonces las nuevas funciones de decodificación $B_i^*(w)$ y $B_j^*(w)$ tienen la siguiente característica:

$$B_j^*(w) = B_j(w) - c = 2.c - B_i(w) - c = -(B_i(w) - c) = -B_i^*(w).$$



Por tanto basta con almacenar los valores de $C_i^*(w) = w.B_i^*(w)$ y sustituir $... + C_j^*(w_j) + ...$ por $... -C_i(w_j) + ...$. De esta manera se reduce el tamaño de la memoria que almacena las funciones de codificación.

Las dos últimas operaciones sirven para calcular un cociente. Para ello se utiliza el algoritmo de división sin restauración. Está basado en la propiedad siguiente:

Propiedad Dados dos números enteros x e y tales que $-y \le x < y$, existe una única descomposición de $2.x$ en la forma $2.x = (2.q-1).y+r$ donde $q \in \{0,1\}$ y $-y \le r < y$.
La demostración se basa en la siguiente regla de cálculo de q y r: si $x<0$ entonces $q=0$ y $r=2.x+y$; si no: $q=1$ y $r=2.x-y$.

El algoritmo de división consiste en una serie de descomposiciones:

$$2.x = (2.q_0-1).y + r_1$$
$$2.r_1 = (2.q_1-1).y + r_2$$
$$2.r_2 = (2.q_2-1).y + r_3$$
$$....$$
$$2.r_{l-1} = (2.q_{l-1}-1).y + r_l$$

Multiplicando la primera ecuación por 2^{l-1}, la segunda por 2^{l-2}, la tercera por 2^{l-3}, etc., y sumando todas las ecuaciones se obtiene la siguiente relación:

$$2^l.x = (2^l.q_0 + 2^{l-1}.q_1 + 2^{l-2}.q_2 + ... + 2.q_{l-1}).y - (2^l-1).y + r_l$$
$$= (2^l.(q_0 -1)+ 2^{l-1}.q_1 + 2^{l-2}.q_2 + ... + 2.q_{l-1}).y + y + r_l,$$

es decir,

$$x/y \cong (q_0 -1)+ 2^{-1}.q_1 + 2^{-2}.q_2 + ... + 2^{1-l}.q_{l-1} = -\ \bar{q}_0 + 2^{-1}.q_1 + 2^{-2}.q_2 + ... + 2^{1-l}.q_{l-1}$$

número que, en numeración binaria y complemento a 2, se escribe

$$\bar{q}_0\ ,\ q_1\, q_2 \ldots\ q_{l-1}\ .$$

El valor máximo del error es igual a $|(y+r_l)/2^l.y| < 2^{l-1}$ dado que $r_l < y$.

El algoritmo puede ser programado de la forma siguiente:

```
r0 := x;
for i in 0 to l loop
      if ri < 0 then qi := 0; ri+1 := 2.ri + y;
      else qi := 1; ri+1 := 2.ri - y;
      end if;
end loop;
q0 := 1-q0;
```

En el caso del algoritmo de control difuso se sabe que *N/D* es el valor de la función de salida. Si el rango de la misma es *[a,b]*, se sabe de antemano que $a.D \le N \le b.D$ (*D* es siempre positivo). Para poder utilizar el algoritmo de división puede ser necesario sustituir *D* por $D^* = 2^p.D$ de tal manera que $a/2^p \ge -1$ y $b/2^p < 1$, en cuyo caso

$$-D^* \le (a/2^p).D^* \le N \le (b/2^p).D^* < D^*.$$

El algoritmo genera el cociente

$$N/D^* \cong \bar{q}_0\ ,\ q_1\, q_2 \ldots\ q_{l-1}\ .$$

El valor de *f* es

$$f \cong \bar{q}_0\ q_1\, q_2 \ldots\ q_p\ ,\ q_{p+1}\, q_{p+2} \ldots\ q_{l-1}\ .$$

Prácticamente, el cálculo se realiza como sigue:

(1) Alinear los operandos, es decir, sustituir *D* por $D^* = 2^{k.m}.D$ siendo *m* el número de bits de las variables (palabras) internas. Para poder realizar dicha operación se añadió una operación a la unidad aritmético-lógica: generar la palabra de *m* bits *cero = 00 ... 0.* La multiplicación por $2^{k.m}$ es equivalente a la concatenación de *D* con *k* palabras *cero*. Véase un ejemplo: si *D* se expresa con tres palabras almacenadas en *R(7), R(6), R(5)*, y si *R(0)* contiene la palabra *cero*, entonces $2^{3.m}.D = R(7)\ R(6)\ R(5)\ R(0)\ R(0)\ R(0).$
(2) Calcular *r:= x+ cero*.
(3) Ejecutar *l* veces:
si el signo del resultado de la última operación es negativo entonces *q:=0* y *r:=2.r+y*; si no, *q:=1* y *r:=2.r-y* ; entrar *q* en el registro de desplazamiento de salida ($Q := 2.Q + q.s^{1-n}$).

El esquema del cálculo se muestra en la figura 5.

El núcleo de la unidad aritmético-lógica es un circuito combinacional (figura 6) cuyo funcionamiento se describe en la tabla III. Se descompone en *m* módulos (slices) idénticos (figuras 7. y 8). Obsérvese que para calcular *mín(x,y), máx(x,y), x-y (≡ x+ $\bar{y}$+1)* y *2.x-y (≡2.x+ $\bar{y}$+1)* es necesario que las entradas *cent* y *s/r* estén a *1*. Para calcular *x+y* y *2.x+y* estas mismas entradas tienen que estar a *0*. Además, para el cálculo de *2.x+y* y *2.x-y* la entrada *xent* tiene que ser igual a *0*.

func(3:1)	*s/r*	*cent*	*z*
00-	-	-	0
010	1	1	*mín(x,y)*
100	1	1	*máx(x,y)*
110	0	-	*x+y+cent*
110	1	-	*x+ $\bar{y}$+cent*
111	0	-	*2.x+xent+y+cent*
111	1	-	*2.x+xent+ $\bar{y}$+cent*

Tabla III

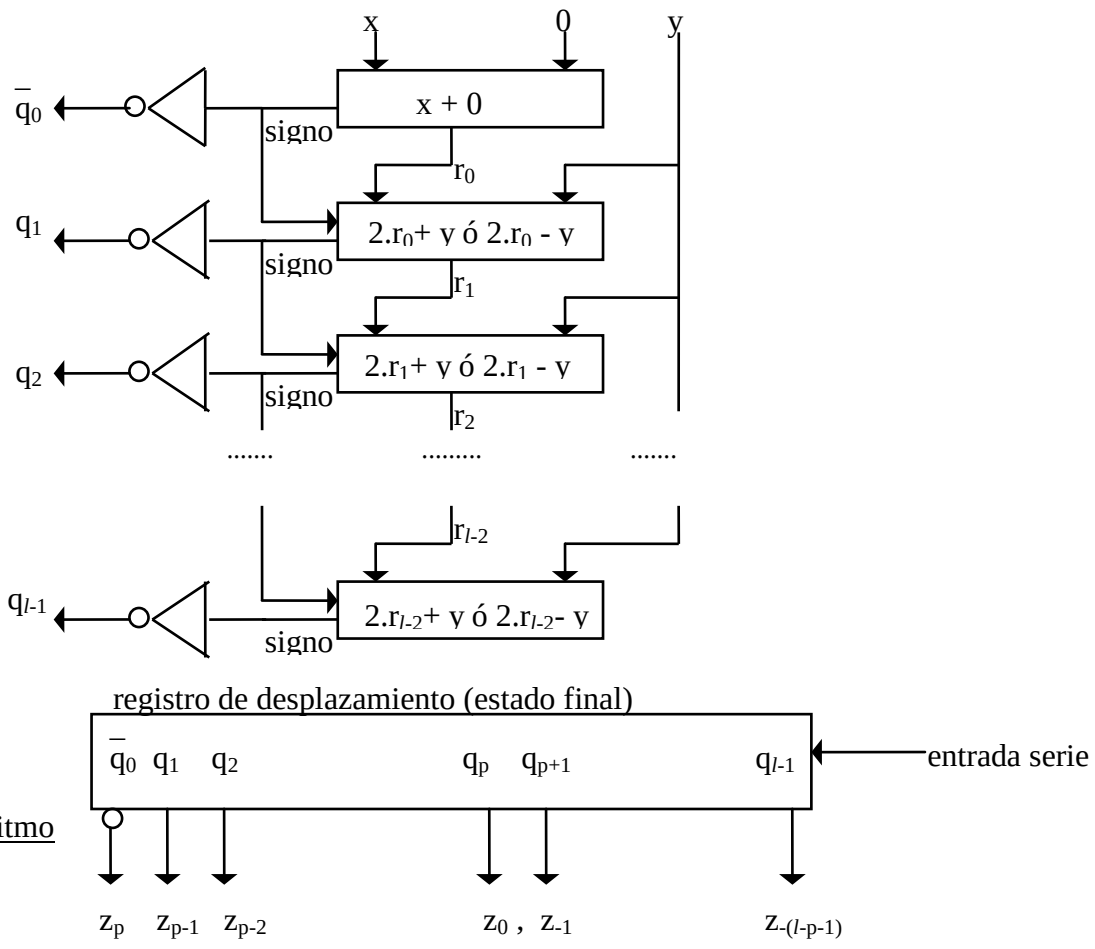


Figura 5 Algoritmo de división

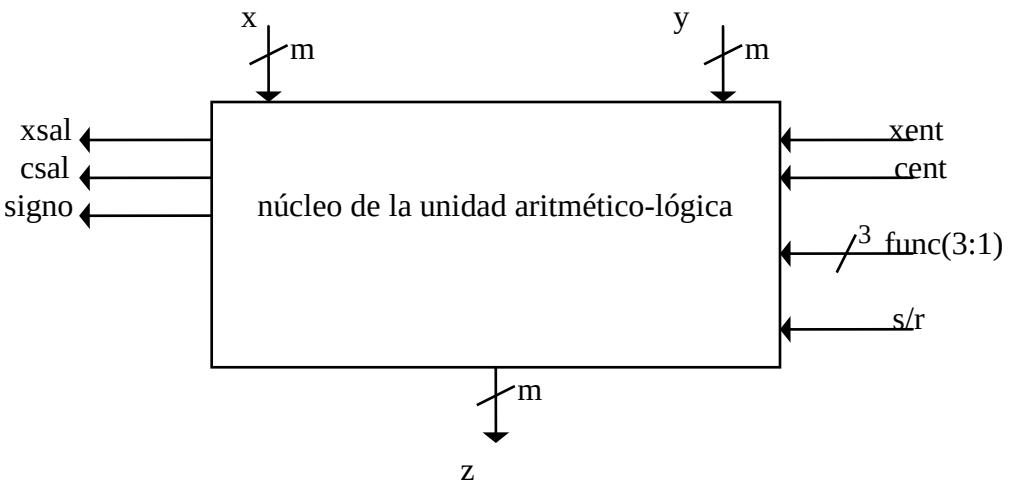


Figura 6 Núcleo de la unidad aritmético-lógica

Para poder realizar operaciones de precisión múltiple, así como poder elegir entre suma y resta en función del resultado de la operación anterior, se necesitan tres biestables que almacenan *xent, cent* y *s/r*. En la tabla IV se define la manera en que se actualizan los tres biestables en función del valor de las variables de control *ff(1:0)* y *sh*.

ff(1:0)	*sh*	*xent*	*cent*	*s/r*
00	0	*xent*	*cent*	*s/r*
01	0	0	0	0
10	0	0	1	1
11	0	*xsal*	*csal*	*s/r*
10	1	0	*/signo*	*/signo*

Tabla IV

La primera línea corresponde a microinstrucciones que no modifican el estado de los biestables. La segunda a la generación del estado inicial para realizar una suma. La tercera a la generación del estado inicial para realizar una resta. La cuarta corresponde a la generación del estado inicial para la segunda, tercera, etc., parte de operaciones de precisión múltiple. La quinta a la generación del estado inicial para realizar las operaciones *2.x+y* o *2.x-y* en función del signo de la operación anterior. El circuito se muestra en la figura 9.

Véanse dos ejemplos. El siguiente microprograma (tabla V) controla la ejecución de las transferencias funcionales *R(8) <= máx[R(5),R(6)]; R(8) <= máx[R(8),R(7)]:*

n°	*i*	*j*	*k*	*tabla*	*t/ual*	*ff*	*sh*	*func*	*in*	*out*
0:	0	-	-	-	0	10	0	00-	0	00...0
1:	8	5	6	-	0	00	0	100	0	00...0
2:	8	8	7	-	0	00	0	100	0	00...0

Tabla V

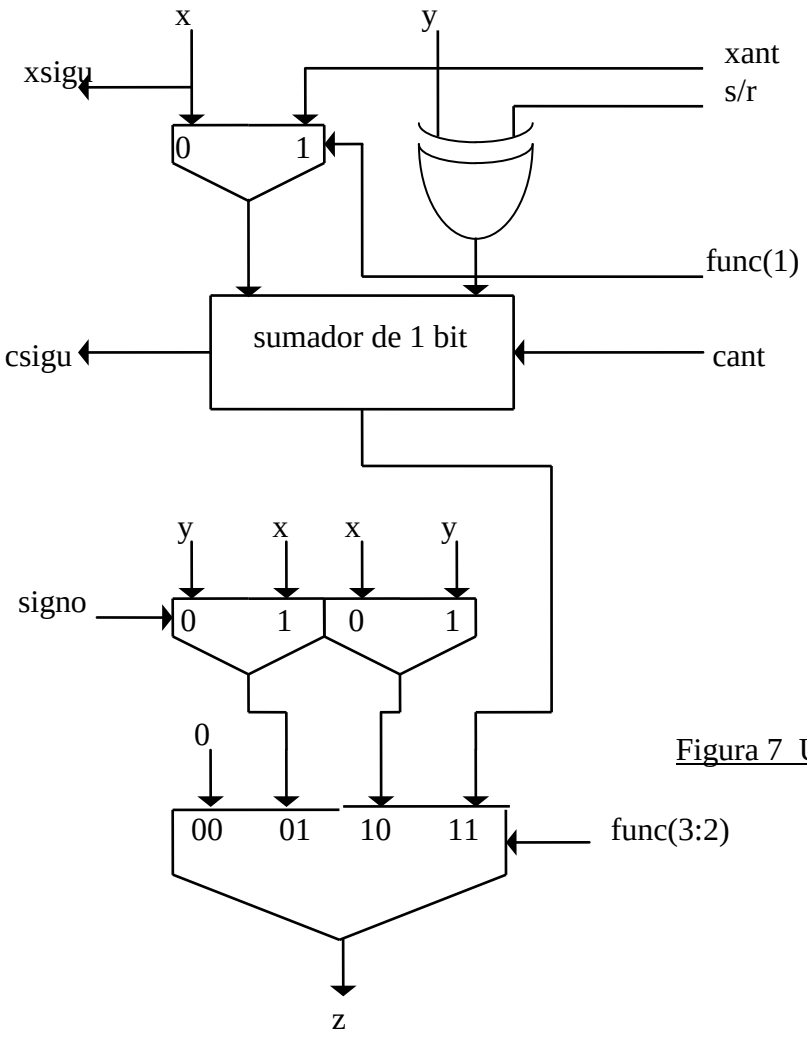


Figura 7 Unidad aritmético-lógica de 1 bit

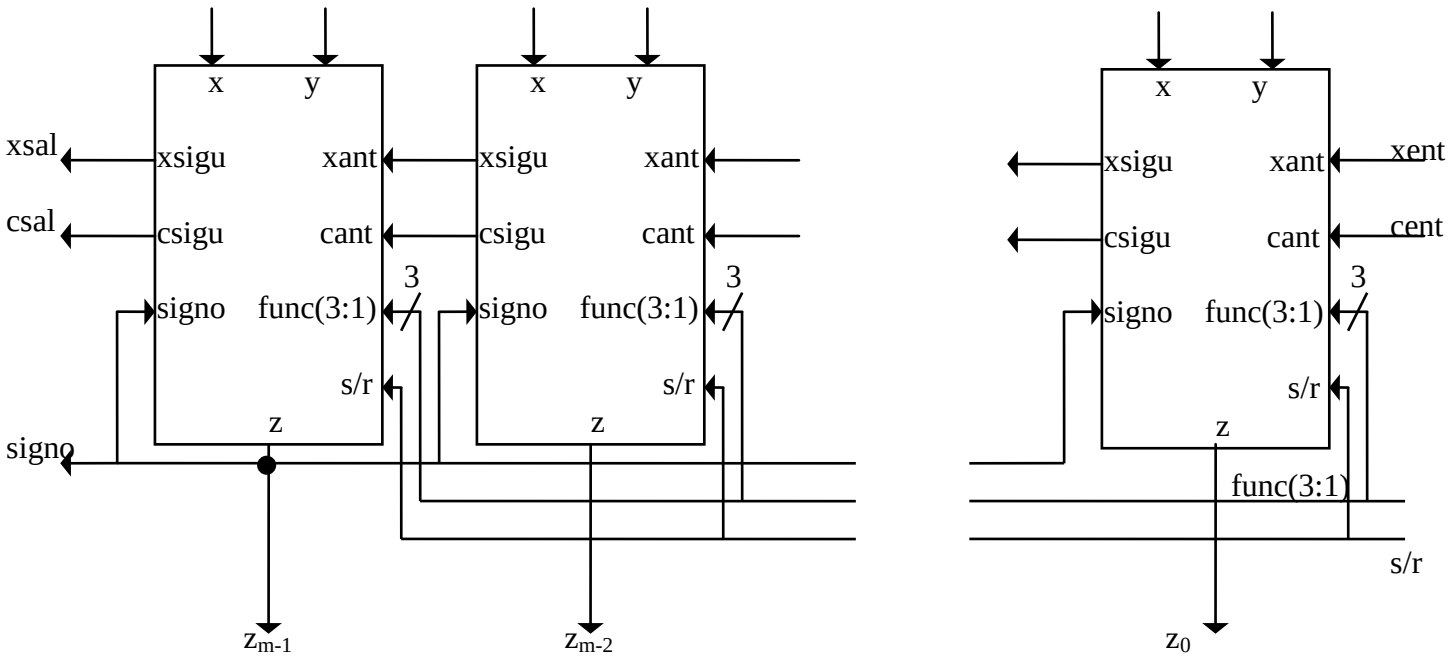


Figura 8 Unidad aritmético-lógica de *m* bits

La microinstrucción n^o 0 sirve para la inicialización de los biestables para realizar la operación *máx*: *xent* indiferente (en este caso *0*), *cent = 1, s/r = 1*; la microinstrucción n^o 1 controla la ejecución de la transferencia *R(8) <= máx[R(5),R(6)]* sin modificar el estado de los biestables; la microinstrucción n^o 2 controla la ejecución de la transferencia *R(8) <= máx[R(8),R(7)].*

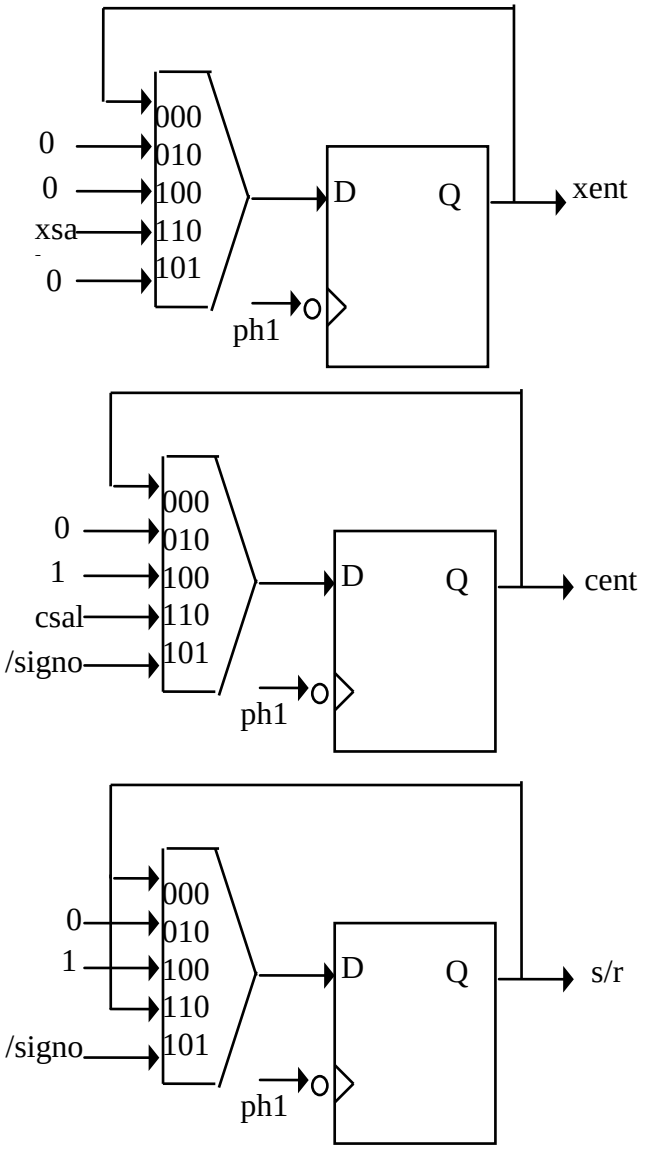


Figura 9 Control de los biestables

Como segundo ejemplo se describe el microprograma de división (tabla VI) de *N* por *D*, suponiendo que *N* consta de tres palabras almacenadas en *R(3), R(2)* y *R(1)*, y *D* de dos

palabras almacenadas en *R(5)* y *R(4)*. Para alinear los operandos es necesario multiplicar *D* por 2^m. Por tanto el cálculo que se lleva a cabo es *R(3) R(2) R(1) / R(5) R(4) R(0)*:

nº	*i*	*j*	*k*	*tabla*	*t/ual*	*ff*	*sh*	*func*	*in*	*out*
0:	0	-	-	-	0	01	0	00-	0	0...00
1:	3	3	0	-	0	10	1	110	0	0...00
2:	1	1	0	-	0	11	0	111	0	0...00
3:	2	2	4	-	0	11	0	111	0	0...00
4:	3	3	5	-	0	10	1	111	0	0...00
5:	1	1	0	-	0	11	0	111	0	0...00
6:	2	2	4	-	0	11	0	111	0	0...00
7:	3	3	5	-	0	10	1	111	0	0...00
8:	1	1	0	-	0	11	0	111	0	0...00
9:	2	2	4	-	0	11	0	111	0	0...00
10:	3	3	5	-	0	10	1	111	0	0...00
11:	1	1	0	-	0	11	0	111	0	0...00
12:	2	2	4	-	0	11	0	111	0	0...00
13:	3	3	5	-	0	10	1	111	0	0...00
14:	0	-	-	-	0	00	0	00-	0	0...01

Tabla VI

Al término de la ejecución de este microprograma se habrán calculado cinco bits del cociente, se habrán introducido en serie en el registro de desplazamiento de salida y se habrá transferido el resultado al puerto de salida nº 1. La microinstrucción nº 0 sirve para poner *cent* y *s/r* a *0*. La microinstrucción nº 1 para sumar *R(3)* con *0* y generar así el bit de signo del cociente; dicho bit se introduce en el registro de desplazamiento de salida (*sh = 1*) y los biestables *cent* y *s/r* se inicializan con el valor de */signo*. La microinstrucción nº 2 realiza la transferencia *R(1) <= 2.R(1)±R(0)* manteniendo *s/r* a su valor anterior y transfiriendo *xsal* y *csal* a los biestables *xent* y *cent*. La microinstrucción nº 3 realiza la transferencia *R(2) <= 2.R(2)±R(4)* controlando los biestables de la misma manera que la anterior. La microinstrucción nº 4 realiza la transferencia *R(3) <= 2.R(3)±R(5)* y actualiza *cent* y *s/r* en función del resultado; al mismo tiempo se añade un bit del cociente al registro de desplazamiento de salida. Con las microinstrucciones nº 5 a 7, 8 a 10 y 11 a 13 se generan otros tres bits del cociente. La última microinstrucción transfiere el contenido del registro de desplazamiento al puerto de salida seleccionado.

6. Puertos de salida

Según el algoritmo descrito en el apartado 5 el valor de una función particular *f* se obtiene en la forma

$$f = \bar{q}_0 \; q_1 \, q_2 \ldots q_p \, , \, q_{p+1} \, q_{p+2} \ldots q_{l-1}$$

siendo *l* el número de bits generados por el microprograma. Por otra parte se supone que en las especificaciones del controlador se define el formato deseado de la función *f*; por ejemplo:

$$f = z_t\ z_{t-1}\ \ldots\ z_0\ ,\ z_{-1}\ z_{-2}\ \ldots\ z_{-(s-t-1)}\ .$$

Por tanto se tienen que cumplir las siguientes desigualdades:

$$p \geq t\ ,\ l-p \geq s-t.$$

La primera se cumple eligiendo para p el primer múltiplo $k.m$ de m tal que $p = k.m \geq t$. Una vez elegido el valor de p, el valor de l se deduce de la segunda desigualdad: $l = p+s-t$.

En conclusión, el número de bits que tiene que generar el microprograma, para la función de salida f_i, es igual a

$$l_i = p_i + s_i - t_i\ .$$

Observación: caso de que $p_i = t_i$, el número l_i de bits generados por el microprograma es igual al número s_i de bits de la representación deseada. En este caso el primer bit (el de signo) tiene que ser invertido ($z_t = \bar{q}_0$). Al contrario, si $p_i > t_i$ el número de bits generados por el microprograma es mayor que el número de bits de la representación deseada. Ello significa que se tienen que descartar los $p_i - t_i$ primeros bits, en particular el bit q_0, con lo cual no se invierte el primer bit. En el ejemplo del capítulo 7 la función de salida tiene un valor comprendido entre -32 y 31, y se expresa como un número entero de 6 bits: $t+1 = 6,\ s-t-1 = 0$, es decir, $t = 5$ y $s = 6$. Por otra parte, sabiendo que $-32 \leq N/D < 31$, es necesario multiplicar D por, como mínimo, 32 para que se cumpla la condición $-1 \leq N/D^* < 1$. Sin embargo, sólo se puede multiplicar por múltiplos de 2^8, siendo 8 el número de bits por palabra interna. Por consiguiente, en lugar de multiplicar D por 32, se multiplica por 256, razón por la cual los tres primeros bits generados por el algoritmo de división pueden ser descartados. Efectivamente en este caso los valores de los parámetros l, p, s y t son:

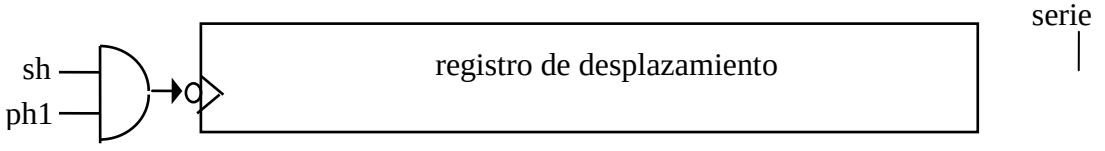
$$t = 5,\ s = 6,\ p = 8,\ l = 8+6-5 = 9.$$

El microprograma genera 9 bits $q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8$ de los cuales sólo se utilizan los seis últimos:

$$f \cong q_3\ q_4\ q_5\ q_6\ q_7\ q_8\ .$$

En realidad $\bar{q}_0 = q_1 = q_2 = q_3 = signo$, con lo cual se pueden eliminar $\bar{q}_0$, q_1 y q_2 .

El circuito se muestra en la figura 10. Para algunos de los latches puede ser necesario invertir el bit de signo, es decir, utilizar la salida $\bar{Q}$ del biestable correspondiente.



máx(s_1, ..., s_r)

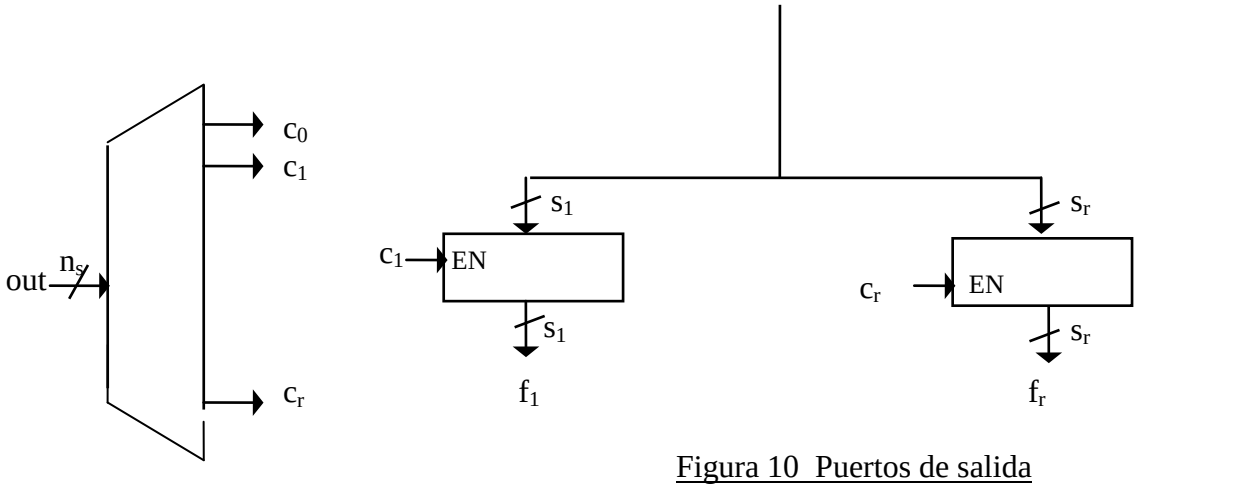


Figura 10 Puertos de salida

7 Cálculo de las funciones de pertenencia y de decodificación

Muchas veces las funciones de pertenencia tienen una forma trapezoidal (ver la figura 11). En lugar de almacenar los valore de $A_i(x)$ en una memoria, se pueden calcular. Bastaría con almacenar los valores de *a, b, c, d* así como los coeficientes angulares k_1 y k_2 que corresponden a los intervalos *[a,b]* y *[c,d]*.

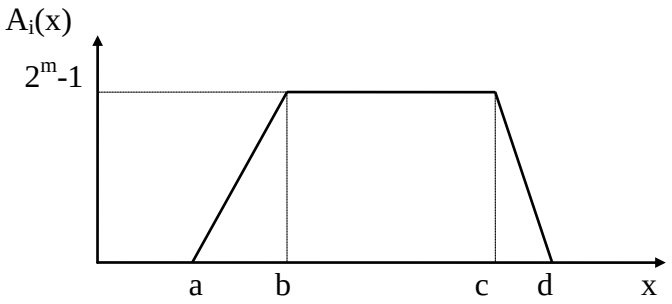


Figura 11 Función de pertenencia trapezoidal

Por otra parte, si se utiliza un método simplificado de decodificación en el cual cada B_j tiene un valor constante (singleton method; ver por ejemplo [11]), entonces en lugar de almacenar los valores de $C_j(w) = w.B_j$ dichos valores pueden ser calculados. En resumen, el circuito *tablas* tiene que ejecutar el siguiente cálculo:

if función_de_pertenencia_i then
if $x \leq a_i$ *or* $x \geq d_i$ *then* $A_i = 0$;
elsif $a_i \leq x \leq b_i$ *then* $A_i = (x-a_i).k_{i1}$;
elsif $c_i \leq x \leq d_i$ *then* $A_i = (d_i-x).k_{i2}$;
else $A_i = 2^m-1$;
end if;

elsif función_de_decodificación_j then $C_j=w.B_j$*;*
end if;

El circuito correspondiente se muestra en la figura 12. El decodificador selecciona *w* o una de las entradas x_i (de la misma manera que en el circuito de la figura 3). Una ROM almacena el valor de los parámetros *a, b, c, d,* k_1 (para las funciones de pertenencia), *B* (para las funciones de decodificación) y k_2 . Para las funciones de decodificación el decodificador selecciona *x=w* y la memoria contiene los valores siguientes: *a=0,* $b=c=d=2^m-1$. Por tanto $a \le x \le b$ con lo cual el valor calculado por el multiplicador es $(x-a).k_1/B = w.B.$ El selector de salida sirve de interfaz entre la salida del multiplicador y el banco de registros. En el caso de las funciones de pertenencia los operandos de la multiplicación son números reales expresados con cierta precisión y el resultado final (salida del circuito) tiene que ser un entero comprendido entre *0* y 2^m-1; el circuito de interfaz selecciona los *m* bits del resultado que corresponden a la parte entera. Para las funciones de decodificación los operandos de la multiplicación son *w* (entero comprendido entre *0* y 2^m-1) y *B* (número real expresado con cierta precisión); el resultado es un número real expresado con (posiblemente) precisión múltiple; el circuito de interfaz permite seleccionar sucesivamente la primera palabra, la segunda palabra, etc. del resultado; ello significa que si *w.B* se representa con *k* palabras, a la función de decodificación *B* corresponden *k* números de tablas (pasaría lo mismo si se eligiera la solución de la figura 3). Todas las filas de la ROM que corresponden a las tablas de una misma función de decodificación *B* tienen el mismo contenido.

La ROM y el multiplicador son macrocélulas compiladas. Basta con definir el contenido de la ROM y el tamaño del multiplicador. El comparador, el restador, los multiplexores y el selector (conjunto de multiplexores) tienen estructuras regulares y pueden ser definidos como bloques VHDL parametrizados. El decodificador es un circuito combinacional que puede ser sintetizado por un programa de síntesis lógica o integrado en otra ROM compilada.

8. Referencias

[1] Masaki Togai and Hiroyuki Watanabe, “Expert System on a Chip: An Engine for Real-Time Approximate Reasoning”, IEEE EXPERT, vol. 1, nº 3, pp. 55 - 62, August 1986.
[2] Hiroyuki Watanabe, Wayne D. Dettlof and Kathy E. Yount, “A VLSI Fuzzy Logic Controller with Reconfigurable, Cascadable Architecture”, IEEE Journal on Solid-State Circuits, vol. 25, nº 2, April 1990.
[3] Ansgar P. Ungering, Karsten Thuener and Karl Goser, “Architecture of a PDM VLSI Fuzzy Logic Controller with Pipelining and Optimized Chip area”, Proceedings of IEEE International Conference on Fuzzy Systems, pp. 447 - 452, 1993.
[4] Mamuro Sasaki, Fumio Uenoand Takahiro Inoue, “7.5 MFLIPS Fuzzy Microprocessor Using SIMD and Logic-in-Memory Structure”, Proceedings of IEEE International Conference on Fuzzy Systems, pp. 527 - 534, 1993.
[5] A.Costa, A.De Gloria, P.Faraboschi, A.Pagni and G.Rizzotto, “Hardware Solution for Fuzzy Control”, Proceedings of the IEEE, vol. 83, nº 3, March 1995, pp. 422 - 434.
[6] J.M.Mendel, “Fuzzy Logic Systems for engineering: A Tutorial”, Proceedings of the IEEE, vol. 83,nº 3, March 1995, pp. 345 - 377.

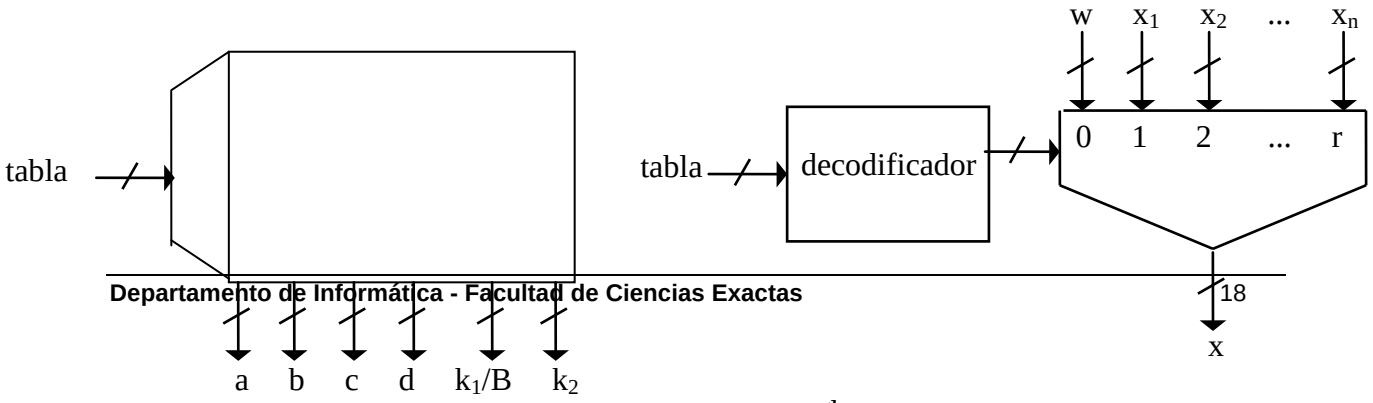


Figura 12 Cálculo de las funciones de pertenencia y de decodificación

[7] J.-P.Deschamps y J.I.Martínez Torre, “Metodología para la Materialización de Algoritmos Borrosos: de una Especificación de Alto Nivel a un ASIC”, V Congreso Español sobre Tecnología y Lógica Fuzzy (ESTYLF'95), Murcia 20 - 22 de Septiembre de 1995, pp. 251 - 256.

[8] D.L.Hung, “Dedicated Digital Fuzzy Hardware”, IEEE Micro, vol. 15, nº 4, August 1995, pp.31 - 39.

[9] In-Cheol Park and Chong-Min Kyung, FAMOS: An Efficient Scheduling Algorithm for High-Level Synthesis, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol, 12, number 10, October 1993, pp. 1437 - 1448.

[10] R.Moreno, R.Hermida, M.Fernández, J.M.Mendías, J.Septien, Global Analysis for Integrated Scheduling and Module Allocation, Proceedings of APCHDL 94, pp.103 - 109, 1994.

[11] B.Kosko, “Neural Network and Fuzzy Systems”, Prentice Hall, 1992.

[12] N.Acosta, J.-P.Deschamps y G.Sutter, “Herramientas para la Materialización de Controladores Difusos Microprogramados”, *Tercer Workshop Iberchip*, CINVESTAV, México D.F., 19 - 21 de Febrero de 1997, pp. 164 - 173.

[13] J.-P.Deschamps y G.Bioul, “Metodología de Síntesis de Controladores Difusos”, *Tercer Workshop Iberchip,* CINVESTAV, México D.F., 19 - 21 de Febrero de 1997, 154 - 163.

[14] G.Rigotti y M. Tosini, “Codiseño y Testeo de Controladores Difusos”, *Tercer Workshop Iberchip,* CINVESTAV, México D.F., 19 - 21 de Febrero de 1997, pp. 174 - 183.