Aplicaciones aritméticas usando lógica programable

Guillermo A. Jaquenod[1], Marisa R. De Giusti[2], Roberto J. De La Vega[3]

[1]. Facultad de Ingeniería – Univ. Nac.del Centro – Argentina.
chipi@netverk.com.ar. +54.221.479.3142
[2]. Comisión de Investigaciones Científicas de la Pcia. de Buenos Aires – Argentina.
marisadg@volta.ing.unlp.edu.ar. +54.221.4236696
[3]. Facultad de Ingeniería – Univ. Nac.del Centro – Argentina.
rjdlv@fio.unicen.edu.ar. +54.2284.451055

Abstract:

Digital Signal Processing (DSP) has become a cheap, usual and standard tool, with applications in almost every area of electronics. DSP offers big advantages over classic analog processing, like high precision, bounded processing noise, etc., and in DSP algorithms addition, substraction, and multiplication are essential operators.
This paper describes basic arithmetics circuits designed by using Programmable Logic Devices (CPLDs), and shows how bit-serial processing solutions can afford high computing performance with low resources usage.

1.Introducción:

El tratamiento digital de señales (DSP) ha pasado a ser una disciplina cuyo uso se encuentra cada vez en más aplicaciones. Este tipo de procesamiento ofrece enormes ventajas en relación al tratamiento analógico clásico [1,2,3,4], tales como:

- elevada precisión: no existe una limitación física a la precisión de cálculo en las operaciones, siendo usual [1,2] resoluciones de 10 hasta 32 bits.
- independencia ante variaciones térmicas: los números son valores independientes a fenómenos físicos como la temperatura o el envejecimiento, que sí afectan a los circuitos analógicos.
- ruido de procesamiento acotado: esta fuente de distorsión, asociada a los procesos numéricos de truncamiento y redondeo, puede ser modelizada y evaluada, y donde sea necesario minimizada trabajando con mayor precisión.
- otras prestaciones: mediante DSP es posible obtener anchos de banda, frecuencias de corte, bandas pasantes, etc..., imposibles de lograr con componentes analógicos convencionales. Es posible también realizar transformaciones (como la FFT, por ejemplo) mono y multidimensionales, impensables de otro modo.

En este tipo de tareas se encuentra por doquier la multiplicación de datos por coeficientes y la acumulación de estos productos parciales, por lo que los circuitos aritméticos de suma, resta y multiplicación (para números con y sin signo) son herramientas esenciales, siendo también de utilidad circuitos más complejos para cómputo de módulo/fase o funciones trigonométricas [5].

Este articulo describe circuitos aritméticos básicos de suma, resta y multiplicación, y cómo implementarlos usando lógica programable, así como métodos de serialización de datos que posibilitan elevada performance de cálculo con bajo uso de recursos.

2.La suma

La suma es una operación definida entre números, que son elementos abstractos que sirven para representar cantidades. Dado el conjunto de números que componen el universo de elementos de un sistema numérico es posible definir una operación llamada SUMA [6] que opera sobre los elementos de ese sistema y produce como resultado otro elemento, también de ese sistema.

En el sistema decimal que se acostumbra usar en la vida diaria, el conjunto de números va desde menos infinito a más infinito, y la operación SUMA resulta afín al proceso físico de agrupamiento de objetos. Si un sistema numérico no es infinito (por ejemplo, cantidad de dígitos limitada), aparece un fenómeno llamado desborde (overflow); por ejemplo, con tres dígitos decimales sólo es posible representar números entre 000 y 999, allí la operación 222+333=555 resulta lógica pero 999+002=001, si bien correcta, puede resultar extraña.

En un sistema electrónico suele ser conveniente representar los posibles símbolos mediante dos voltajes extremos, a los que se asocian los símbolos ‘0’ y ‘1’; esta convención con sólo dos símbolos es llamada binaria y cada dígito es asociado a la palabra bit (binary digit).

En un sistema numérico de sólo 1 bit, la suma de dos operandos A y B (cada uno de los cuales sólo puede valer ‘0’o ‘1’), está definida mediante

```
0+0 = 0
0+1 = 1
1+0 = 1
1+1 = 0... caso del desborde!
```

Este tipo de función aritmética puede asociarse a la función lógica OR EXCLUSIVO, de donde:

```
SUMA (A,B) == A XOR B
```

La utilidad de esta función es limitada, y suele ser acompañada de una segunda función llamada acarreo (CARRY), cuya salida Cout avisa cuando se produce el desborde en la situación definida por 1+1=0. Al igual que en el caso previo, esta función aritmética puede asociarse a la función lógica AND, de donde:

```
Cout (A,B) == A AND B
```

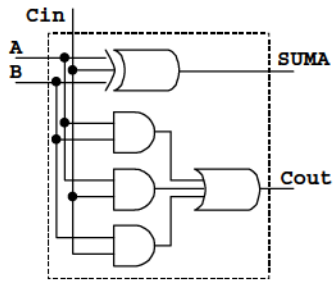
El bloque funcional aritmético compuesto por un operador SUMA más un operador CARRY, con dos salidas Cout y SUMA y con dos entradas A y B, es denominado medio-sumador o ***half-adder***.[3,6,7].

Cuando se desea sumar números binarios de varios dígitos (varios bits) se emplea un sistema posicional, donde en cada numero el peso de cada bit está asociado a su posición (al igual que en decimal, donde un 3 en las unidades representa 3, pero un 3 en las decenas representa 30 = 3 x 10). Usando esta convención es posible representar números naturales.

$$N = \sum_{i=0}^{n-1} a_i.2^{\mathrm{i}}$$

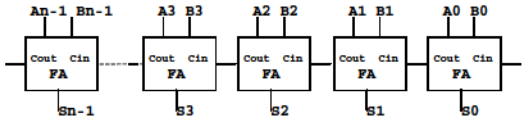
Si se indica con subíndices crecientes (desde 0 en adelante) la posición de bits de peso numérico creciente, es claro que dados dos operandos, para sumar los bits menos significativos (posición 0), basta un ***half adder***, pero para aquellos en la posición 1 y sucesivas falta algo; esto sucede porque para cada posición, además de los respectivos bits de los operandos A y B se hace necesario también sumar el acarreo proveniente de la etapa previa.

El circuito digital que realiza esta operación posee ahora tres entradas (las conocidas A, B y una nueva llamada Carry de entrada –o Cin) y dos salidas (SUMA y Cout), es denominado sumador completo, o ***full adder***, y las ecuaciones lógicas que describen su operación son:




```
SUMA (A,B,Cin) = A XOR B XOR Cin

Cout (A,B,Cin) = (A AND B)
                 OR (A AND Cin)
                 OR (B AND Cin)
```

Un ***full-adder*** puede ser también visto [8] como un **compresor 3:2**, o **contador 3:2**, pues dadas sus tres entradas, en ellas puede haber desde ningún hasta tres unos, y la dupla (Cout,SUMA) indica justamente cuántos unos son (de igual modo, un ***half-adder*** puede ser visto como un **contador 2:2**). Un sumador paralelo de dos palabras A y B de N bits requiere entonces de un ***half-adder*** y (N-1) ***full-adders***, donde la salida Cout de cada etapa se conecta a la entrada Cin de la etapa siguiente. Si este sumador debe poder aceptar acarreos desde etapas previas, serán necesarios entonces N ***full-adders***, como en la figura siguiente.



Este circuito es plenamente operativo y ampliamente usado, y el método empleado para la propagación de la señal de acarreo existente desde el bit menos significativo al más significativo es llamado ***"ripple carry adder (RCA)"*** [3,4,6,7]. Su única debilidad surge cuando se requiere realizar operaciones a muy alta velocidad, dado que es necesario esperar que el acarreo se propague a través de todos los ***full-adders*** (dos niveles de compuerta de retardo por cada bit de suma) para recién contar con un resultado válido.

3.La resta y los números negativos

Mediante sistemas numéricos que emplean codificación binaria es también posible representar números enteros. Existen varios métodos (signo y módulo, complemento a uno, y complemento a dos) siendo el último el empleado en la casi totalidad de los casos.

En el caso del complemento a dos, la cantidad representada por cada bit del número coincide con el caso de los números sin signo, excepto para el caso del bit más significativo, cuyo peso es de (-2^(N-1)).

$$N = -a_{n-1}.2^{n-1} + \sum_{i=0}^{n-2} a_i.2^i$$

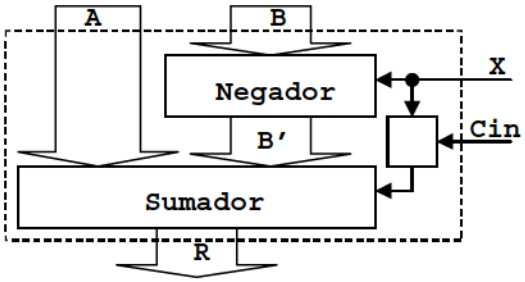
Por razones conocidas [6] el complemento a dos de un número puede calcularse como el complemento a uno de dicho número más uno, siendo lo interesante que el complemento a uno de un número es ese número negado bit a bit (es decir donde cada cero se reemplaza por un uno, y viceversa). Si se usa el signo "!" para indicar la negación bit a bit, para resolver R=A-B surge:

```
R = A - B = A + (-B)
  = A + (!B + 1) = A + !B + 1
```

Donde el '1' a sumar puede ser incorporado poniendo a '1' la línea Cin del ***full-adder*** correspondiente al bit menos significativo.

Un sumador / restador (esquema primigenio de lo que es la ALU en cualquier microprocesador) que permita

resolver tanto R=A+B como R=A-B en base a una línea de control X (X=0: SUMA; X=1: RESTA) podrá ser resuelto mediante un esquema como el siguiente:



Donde el bloque negador pone en su salida B' el valor B (si X=0) o !B (si X=1).

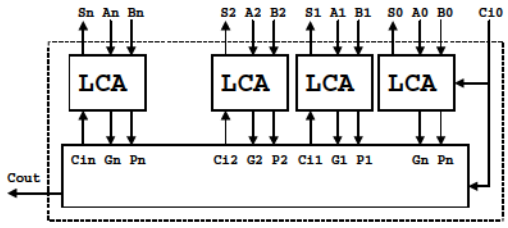
4.Mejora de los tiempos de cómputo

Para acelerar el proceso de la suma de números de muchos bits, existen varias soluciones tecnológicas:

Look Ahead Carry (LAC): en vez de esperar la propagación de la cadena de CARRY es posible que en cada etapa "i" se generen dos funciones extra, llamadas G(Generate) y P(Propagate), cada una de las cuales requiere un único nivel de retardo.

```
Gi = Ai AND Bi        Pi  = Ai OR Bi
```

En base a estas señales las entradas Cin a los sumadores pueden calcularse como sigue:

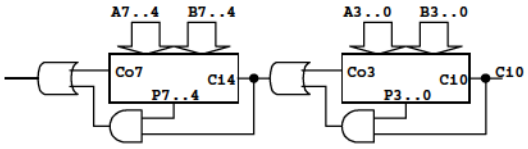



```
Ci1 = G0 OR (P0 AND Ci0)
Ci2 = G1
      OR (G0 AND P1)
      OR (Ci0 AND P1 AND P0)
Ci3 = G2
      OR (G1 AND P2)
      OR (G0 AND P2 AND P1)
      OR (Ci0 AND P2 AND P1 AND P0)
Ci4 = G3
      OR (G2 AND P3)
      OR (G1 AND P3 AND P2)
      OR (G0 AND P3 AND P2 AND P1)
     OR (Ci0 AND P3 AND P2
         AND P1 AND P0)
Ci5 = ...
Ci6 = ...
```

Si se toma como ejemplo Ci4, se nota que usando LCA su valor puede ser generado con un retardo de sólo tres niveles frente a los ocho niveles de un RCA.
El uso de LAC es común en circuitos de alta velocidad [3,4,6,7], y su única desventaja es que la ecuación de cálculo de sucesivos Cin se hace cada vez más compleja; por ello es habitual el empleo de circuitos híbridos, formados por bloques que emplean internamente LAC, pero que son conectados entre sí en cascada.

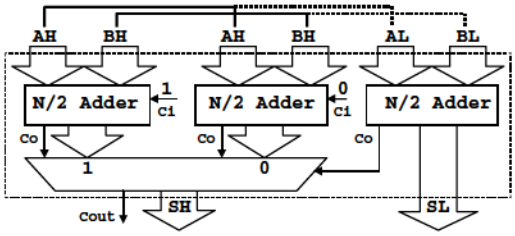
Carry Skip Adder (CSKIP): El cálculo de P (Propagate) es mucho más simple que el de Generate. La solución Carry Skip emplea bloques (P.Ej: de 4 bits) que usan internamente RCA, y que sólo generan una función simple P a nivel de bloque.

```
P = P0 AND P1 AND P2 AND P3
  =      (A0 OR B0) AND (A1 OR B1)
    AND (A2 OR B2) AND (A3 OR B3)
```



Mediante circuitos de este tipo es posible ganar tiempo, por cuanto si un bloque propaga el acarreo (P==1) y tiene acarreo en su entrada ya puede saberse que producirá acarreo en su salida [3].

Carry Select (CSELECT): Esta solución [9] permite reducir a casi la mitad los retardos de cálculo, a costa del incremento de algo más de un 50% en la complejidad del circuito.



Dados dos sumandos A y B de N bits se los divide en dos partes: una parte (AL y BL) compuesta por los N/2 bits menos significativos, y una parte (AH y BH) compuesta por los N/2 bits más significativos restantes. Para la suma AL+BL se usa un único sumador "L" de N/2 bits, en tanto que para la suma de AH+BH se emplean dos sumadores, H0 y H1: H0 con Cin=0, y H1 con Cin=1, usándose luego la señal de salida Cout del sumador "L" para optar entre las salidas de los dos sumadores Hi.

Carry Save Adder (CSA): Cuando es necesario sumar más de dos operandos es posible derivar por distintos caminos a las señales SUMA y Cout [4] (siempre que se respeten sus pesos binarios) para optimizar los tiempos de retardo.

5.La suma de múltiples términos

Cuando en una aplicación el resultado surge de la suma de más de dos sumandos, es prudente analizar la forma de realizar dicha suma, para minimizar uso de recursos y retardos.

Si en una suma múltiple se analiza individualmente cada columna (que contiene términos de igual peso), 3

términos de peso 2^n pueden ser comprimidos a la forma de 2 señales, una de peso 2^n y otra de peso 2^(n+1).; de igual modo, 7 términos de una misma columna pueden comprimirse a la forma de 3 señales (de peso 2^n, 2^(n+1) y 2^(n+2)), o 15 términos en 4 señales, etc [6].

Otra consideración importante es acomodar los sumadores de modo de tratar de evitar lo más posible que una etapa, con datos de entrada estables, deba esperar el acarreo de una etapa previa para poder resolver su salida.

Como ejemplo obvio: para sumar A+B+C+D, la forma ineficiente es realizar la operación como A+(B+(C+D)) que requiere 3 sumadores y produce 3 retardos de suma, en tanto que hacerla del modo (A+B)+(C+D), con idéntico consumo de recursos, sólo produce 2 retardos de suma.

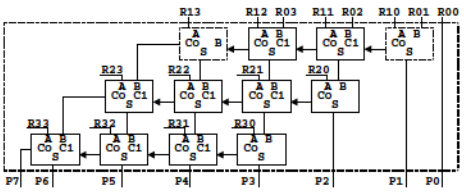
6.La multiplicación

La multiplicación binaria bit a bit, es muy simple, por cuanto solo si ambos multiplicandos valen 1 el producto es 1, lo que coincide con la función AND. El problema de su realización surge cuando se multiplican números de varios bits, por cuanto cada bit del resultado surge de una función lógica compleja de los bits de los multiplicandos.

Dados, por ejemplo, dos operandos de 4 bits indicados por B3B2B1B0 y A3A2A1A0, y si se denomina Rij = Ai & Bj, su producto está dado por

```
                          A3  A2  A1  A0
                        X B3  B2  B1  B0
=================================
pp0   0    0    0    0   R30 R20 R10 R00
pp1  +0    0    0   R31  R21 R11 R01   0
pp2   0    0   R32  R22  R12 R02   0    0
pp3   0   R33  R23  R13  R03   0    0    0
=================================
     P7   P6   P5   P4   P3   P2   P1   P0
```

En este caso el producto puede ser calculado mediante el esquema siguiente[3], con doce *full-adders*, y cuyo peor retardo de cálculo es ocho niveles de *full-adder*

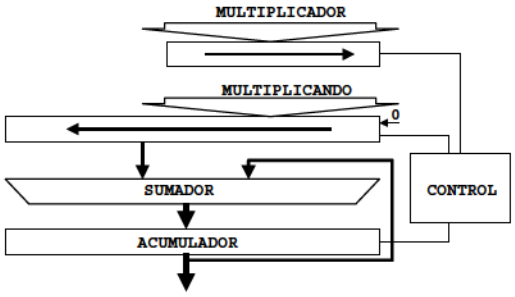


Otros métodos de reagrupamiento de las sumas parciales (Wallace-tree multiplier, Dadda multiplier [4]) permiten obtener mejoras adicionales en el uso de recursos y en la velocidad de cómputo.

Sin embargo, todos los métodos de multiplicación paralela de números de N bits requieren, con ligeras diferencias, una cantidad de sumadores que crece como N^2, y para minimizar este problema, una alternativa posible es descomponer el proceso de multiplicación en forma de una suma secuencial de productos parciales, proceso que desde el punto de vista algorítmico puede ser descripto (usando C, por ejemplo) como:

```
unsigned mundo, mudor; unsigned pp = 0;

for (i=0, i< nbits; i++) {
  if ((mudor & 0x0001) != 0)
    pp += mundo;
  mundo <<= 1;
  mudor >>= 1;
}
```

En este proceso (a veces llamado ***shift/add***), cada producto parcial puede ser nulo (si el correspondiente bit del multiplicador es cero) o igual al multiplicando (con un desplazamiento según qué bit del multiplicador sea), y un esquema de una posible realización física es:



En el caso de números con signo la multiplicación debe considerar dos nuevos elementos: la extensión de signo del multiplicando y la diferencia entre el peso numérico del bit más significativo (MSB) del multiplicador y los restantes, pues este bit MSB representa una cantidad negativa.

Uno de los métodos más conocidos es llamado “algoritmo de BOOTH” [3], donde los bits del multiplicador son analizados de a pares, y en función de ellos el multiplicando desplazado es sumado o restado para ir generando los productos parciales. Si se considera, por ejemplo, un multiplicando A y un multiplicador B, ambos de 5 bits y en complemento a 2, y donde B se indica de la forma B4B3B2B1B0, dicha cantidad B puede ser asociada a la suma:

```
B = B4.(-16)+B3.8+B2.4+B1.2+B0.1
  =  B4.(-16) + B3.(16-8)
   + B2.(8-4) + B1.(4-2) + B0(2-1)
  = -16.(B4-B3)
    - 8.(B3-B2)
    - 4.(B2-B1)
    - 2.(B1-B0)
    - 1.(B0-0)
= -16.C4 - 8.C3 - 4.C2 - 2.C1 - 1.C0
```

Dado que los **Bi** sólo pueden valer 0 o 1, los términos **Ci** sólo pueden valer -1, 0, o +1, definiendo en cada iteración si el multiplicando A, desplazado a izquierda a medida que se analiza desde el LSB al MSB, es sumado, ignorado, o restado, respectivamente..

7.Consideraciones de diseño con FPL

Para pasar de la concepción teórica a la realización práctica de un dado diseño deben tenerse en cuenta la

tecnología a usar (P.Ej: ASICs o lógica programable), e incluso en este caso el empleo de soluciones basadas en términos producto, tablas de lookup o multiplexores llevan a planteos muy distintos [4]. Por ello al encarar un diseño se impone considerar previamente el tipo de recursos disponibles y las restricciones que plantea. En el caso de las familias FLEX de ALTERA [8] debe tenerse en cuenta su arquitectura:

Los elementos lógicos (LEs): son los elementos constructivos de la familia FLEX, y se componen de un bloque combinatorio basado en una tabla de lookup (LUT) de 16 bits seguida de un registro opcional.

- Cada elemento lógico tiene un abanico de entrada (fan-in) de sólo 4 líneas, elemento importante a tener en cuenta cuando se requieren funciones de muchas entradas.
- Los elementos lógicos poseen el llamado "modo aritmético", donde la LUT se descompone en dos subtablas de 8 bits y donde sólo se dispone de 2 entradas a cada LE (la línea restante proviene de una de las subtablas de la etapa previa). Esta función, llamada CARRY CHAIN, permite alta velocidad de operación y eficiencia en el aprovechamiento de los LEs aunque a costo de imponer mayores exigencias al proceso de asignación de celdas (*fitting*) al compilador.
- Los elementos lógicos también poseen una función llamada CASCADE CHAIN que permite realizar el AND lógico de las salidas de LEs contiguos, y así calcular ciertas funciones con mayor fan-in. Al igual que el CARRY CHAIN, esta función permite alta velocidad de operación y eficiencia en el uso de recursos a costa de exigencias de *fitting*.

Un diseño que aproveche la estructura de las familias FLEX deberá hacer uso mesurado de las cadenas de CARRY y CASCADE, y buscar aquella descomposición de las funciones a sintetizar que aproveche módulos de bajo fan-in, para óptimo uso de la LUT de cada LE.

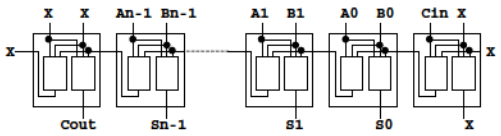
Los LABs: ocho LEs, con facilidades especiales de conexión entre sí, forman un LAB (Logic Array Block); el agrupamiento de ciertas funciones en un LAB permite el uso de estos recursos locales de conexión, con mejoras de velocidad y menor compromiso de los recursos globales

La interconexión FAST TRACK: es un recurso de conexión global, en forma de matriz, que provee canales de conexión rápidos entre LABs y con las patas del dispositivo.

Los EABs: intercalados con los LABs, existen bloques de memoria RAM rápida de 2048 bits, que pueden ser configurados como RAM o ROM, y con ancho de palabra entre 1 y 8 bits.

8.Circuitos de suma usando FLEX:

8.1.Sumador paralelo simple: Usando dispositivos FLEX y la opción CARRY CHAIN un sumador requiere tantos LEs como bits de precisión sean necesarios, más un LE inicial y uno final para ingresar Cin y sacar Cout de la cadena de CARRY, es decir (N+2) LEs [2]. El esquema circuital de un sumador de N bits se observa en la figura siguiente, junto con su descripción mediante el lenguaje de descripción de hardware AHDL.




```
PARAMETERS (N);

SUBDESIGN n_adder (
 a[N-1..0],b[N-1..0],cin : INPUT;
 s[N-1..0],cout : OUTPUT;)

VARIABLE
 ctmp[N..0] : NODE;

BEGIN
 ctmp[0] = cin;
 FOR i IN 0 TO N-1 GENERATE
  s[i] = a[i]
     XOR b[i]
     XOR CARRY(ctmp[i]);
  ctmp[i+1] =  (a[i] AND b[i])
     OR (a[i] AND CARRY(ctmp[i]))
     OR (b[i] AND CARRY(ctmp[i]));
 END GENERATE;
 cout = CARRY(ctmp[N]);
END;
```

El operador ***CARRY(_)*** permite indicar al compilador cómo resolver la conexión entre LEs; a su vez, la definición de N como parámetro permite que esta descripción mediante AHDL sea independiente de la cantidad de bits de los sumandos.

8.2.Sumador restador paralelo: Un circuito capaz de sumar y restar mediante dispositivos FLEX requiere el agregado de una etapa capaz de negar o no los bits del sustraendo B[i] así como en acarreo de entrada. En este nuevo circuito, la descripción en AHDL es:

```
PARAMETERS (N);

SUBDESIGN n_adder (
 a[N-1..0],b[N-1..0],cin,ctl:INPUT;
 s[N-1..0],cout : OUTPUT;)

VARIABLE
 ctmp[N..0] : NODE;
 bx [N-1..0] : NODE;

BEGIN
 ctmp[0] = cin XOR ctl;
 FOR i IN 0 TO N-1 GENERATE
  bx[i] = b[i] XOR ctl;
  s[i] =   a[i]
    XOR bx[i]
    XOR CARRY(ctmp[i]);
  ctmp[i+1] =  (a[i] AND bx[i])
    OR (a[i] AND CARRY(ctmp[i]))
    OR (bx[i] AND CARRY(ctmp[i]));
 END GENERATE;
 cout = CARRY(ctmp[N]) XOR ctl;
END;
```

Dado que los LEs, cuando usan la opción CARRY CHAIN, sólo pueden resolver en cada LE dos funciones de 3 variables de entrada, y que ahora cada etapa del sumador restador requiere 4 entradas (A[i],B[i], Cin[i], ctl) es que el consumo de recursos de un sumador / restador paralelo de N bits pasa a ser de (2N+2) LEs.

9.Circuitos de multiplicación con FLEX

9.1.Multiplicador paralelo: El multiplicador paralelo usa en forma intensa al sumador de múltiples términos. Al observar la expresión de un producto de 4 X 4 bits se nota que:

- P0 copia R00 y es solo el AND de A0 y B0, y no puede generar acarreo
- P1 sólo requiere sumar R01+R10 y puede generar un acarreo hacia P2
- P2 requiere sumar cuatro términos: R20, R11, R02 más el posible acarreo desde P1, y puede generar acarreos hacia P3.
- P3 requiere la suma de cinco términos: R30, R21, R12, R03 más el posible acarreo desde P2, y puede generar acarreos hacia P4 y P5.

La forma óptima de particionar una suma múltiple es fuertemente dependiente de la capacidad de fan-in, los recursos de cableado, y de las prestaciones "especiales" propias de la tecnología FLEX, como las cadenas de CARRY.

9.2.Multiplicador iterativo para números sin signo: La realización mediante FLEX de un circuito de multiplicación tipo shift-add requiere:

- un registro de desplazamiento de 2N bits, donde se carga inicialmente el multiplicando MO, y que es desplazado a izquierda en cada ciclo de iteración
- un registro de desplazamiento de N bits para el multiplicador MR, que se desplaza a derecha en cada ciclo de iteración, y cuyo bit menos significativo indica en cada ciclo si el multiplicando MO debe ser o no sumado al producto parcial acumulado en AC
- un acumulador AC de 2N bits, donde se almacena el resultado parcial
- un sumador ADDER de 2N bit para realizar la eventual suma de AC con MO
- circuitos de cuenta y control de la iteración.

Esto lleva a un consumo de recursos de alrededor de 6N LEs, con un retardo de procesamiento de N+1 o N+2 ciclos de reloj.

9.3.Multiplicador iterativo para números con signo: Este tipo de circuito es muy parecido al previo, y sólo requiere un sumador / restador de 2N bits en vez del sumador, que al cargase el multiplicando MO en la parte alta del registro se copie su signo, y el agregado de un registro extra a la derecha del registro de desplazamiento del multiplicador MR, para contar con los bits de decisión que requiere el algoritmo de BOOTH. Por ello el consumo de recursos es ahora de alrededor de 8N LEs, con idéntico retardo de procesamiento.

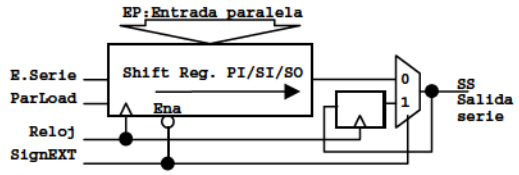
10.Aritmética serial usando FLEX

En muchos casos de procesamiento de señales, la frecuencia de entrada de los datos a procesar es muy inferior a la máxima frecuencia de operación secuencial de los dispositivos FPGA disponibles. En ese caso, en vez de procesar los datos de N bits en paralelo, es posible procesar esos mismos datos en forma serie [4,5], de a un bit por vez, aunque a una frecuencia N veces mayor, obteniendo usualmente importantes ahorros de recursos; ventaja que se evidencia aún más en dispositivos ricos en registros, como es el caso de los FLEX.

Por ejemplo: en audio estéreo calidad CD los datos se muestrean de a pares, con resolución de 16 bits, a razón de sólo 44.1 Ksps, y su serialización significa pasar a procesar un flujo de bits de algo más de 1,41 megabits / segundo, que está entre 50 y 100 veces por debajo de la máxima frecuencia de operación de los dispositivos FLEX más lentos. Otro ejemplo de la utilidad de los circuitos mixtos serie / paralelo en aplicaciones VLSI es su uso en el área de microcontroladores, tal como sucede en la línea COP8SAx de National, donde todo el flujo interno de datos es realizado en forma serie.

Los circuitos de tratamiento serie permiten habitualmente procesar datos de N bits a velocidades de 1 dato cada N o 2N ciclos de reloj más algún ciclo extra de control, y con una latencia (retardo entrada→salida) de unos pocos ciclos de reloj. A estos circuitos suele añadirse una etapa global de control que administra los procesos de serialización, inicialización y sincronismo.

10.1.Sobre la tarea del serializador y el agregado de demoras: Al analizar los distintos circuitos se observa que algunos de ellos deben convertir datos en paralelo al formato serie, y que otros presentar en su salida datos retardados una cierta cantidad de ciclos, o en ciertos momentos deben realizar la extensión de signo del dato saliente.



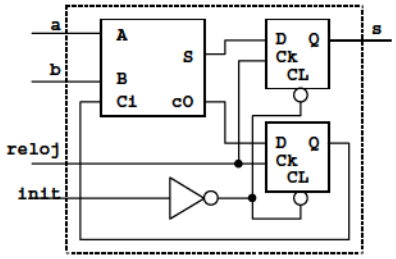
La conversión de datos expresados en forma paralela a un formato serie sólo requiere del uso de registros de desplazamiento PSI/SO (*Parallel & Serial Input Serial Output*), más una etapa de extensión de signo.

11.Circuitos de suma serie con FLEX:

11.1.Un sumador serial: En el circuito siguiente, de tipo Carry Save [5], con solo dos LEs es posible sumar dos operandos A[] y B[] de la longitud que se desee, que son presentados a su entrada desde el bit menos significativo (LSB) al más significativo (MSB).

Siendo su descripción en AHDL:

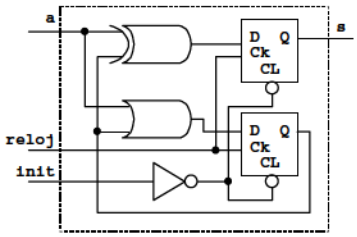
```
SUBDESIGN addser (
 a,b,init,reloj : INPUT;
 s : OUTPUT;)

VARIABLE
 st,cyt : DFF;

BEGIN
 st.clk  = reloj; st.clrn  = !init;
 cyt.clk = reloj; cyt.clrn = !init;
 cyt.d =   (a AND b) OR (a AND cyt.q)
        OR (b AND cyt.q);
 st.d = a XOR b XOR cyt.q;
 s = st.q;
END;
```



En este circuito hay varios puntos a notar:

- que la señal cout[i] generada en cada ciclo es salvada para ser sumada a los bits inmediato superiores A[i+1]+B[i+1] en el ciclo siguiente.
- que la lógica combinatoria a resolver en ambos LE es sólo función de tres variables.
- que la inclusión de flipflop produce un retardo de procesamiento de un ciclo de reloj
- que es necesaria una señal de inicialización, que ponga a '0' al fliflop de Carry Save
- que dados dos operandos de N bits, el resultado debe considerar N+1 bits

11.2.Un complementador serial: Para realizar el complemento a dos de un número es posible aprovechar un conocido algoritmo de conversión [6]: inspeccionar el número desde el bit LSB al MSB, copiandolo en forma textual hasta encontrar el primer '1'(inclusive); y a partir de allí invertir los bits restantes. Un esquema circuital posible es:



Y su descripción en AHDL:

```
SUBDESIGN ca2 (
 a,init,ctl,reloj : INPUT;
 s : OUTPUT;)
VARIABLE
 pasneg,det1 : DFF;
BEGIN
 det1.clrn = !init;
 pasneg.clrn = !init;
 det1.clk = reloj;
 pasneg. clk = reloj;

 IF (ctl = GND) THEN det1.d = GND;
  ELSE det1.d = det1.q OR a;
  END IF;
 pasneg.d = a XOR det1.q;
 s = pasneg.q;
END;
```

Un complementador serial requiere dos LEs: uno copia textualmente o invierte el bit entrante, en tanto el otro detecta y enclava el primer '1'; aquí también:

- es necesario contar con una señal de inicialización.
- se introduce una latencia de un ciclo de reloj.

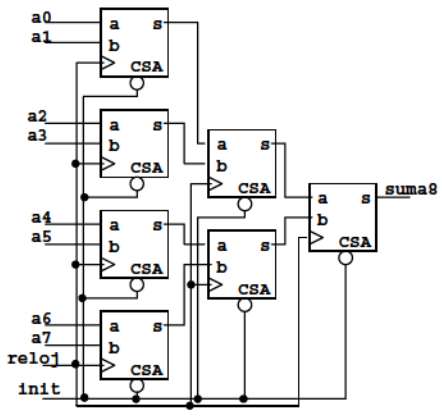
En este caso en el AHDL se ha agregado una señal de control opcional (ctl) que valiendo '0' no niega al dato entrante y valiendo '1' si lo hace.

11.3.Un sumador / restador serial: Como la lógica propia de los dos LEs que componen un sumador serial es función de 3 variables, el agregado de una señal ctl (0:suma; 1:resta), para realizar la suma / resta es directa, y no requiere uso de recursos adicionales. La descripción en AHDL es:

```
SUBDESIGN addsubser (
 a,b,init,ctl, reloj : INPUT;
 s : OUTPUT;)

VARIABLE
 st,cyt : DFF;

BEGIN
 st.clk  = reloj; st.clrn = !init;
 cyt.clk = reloj; cyt.clrn = !init;

 st.d = a XOR b XOR cyt.q;
 IF (ctl == GND)
  THEN
   cyt.d = (a AND b) OR (a AND cyt.q)
         OR (b AND cyt.q);
  ELSE
   cyt.d = (a AND !b) OR (a  AND cyt.q)
         OR (!b AND cyt.q);
  END IF;
 s = st.q;
END;
```

11.4.Sumador serial para términos múltiples

Usando el mismo esquema Carry Save del sumador serie

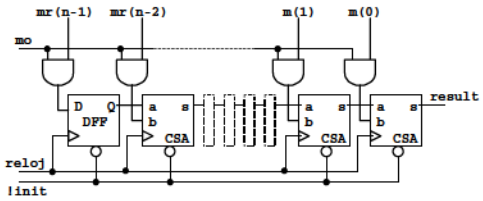


es posible realizar un circuito en forma de árbol, que vaya sumando datos que provienen en forma serial de más de dos fuentes [5].

Este circuito introduce tantos retardos de reloj y agrega tantos bits de precisión al resultado como ceil(log2(K)), donde K es la cantidad de sumandos simultáneos, y consume 2(K-1) elementos lógicos.

12.Multiplicación serial con FLEX

12.1.Multiplicador serial para números sin signo: En muchas aplicaciones de DSP la multiplicación es entre un multiplicando MO dinámicamente variable (datos) y un multiplicador MR estático o cuasi estático (coeficientes) [1,2]. En este caso es posible realizar un circuito serial de multiplicación para palabras de N bits, basado en la suma de productos parciales en un esquema Carry Save, tal como se observa en la figura.



Cada bit de la cadena de producto es el resultado de la multiplicación (AND) del bit ingresante del multiplicando con el correspondiente bit del multiplicador, más el producto parcial previo, más el acarreo salvaguardado de la suma previa, y como se trata de funciones de sólo 4 términos, para la cadena de producto se requieren sólo N LEs; en el caso del bit más significativo no hay transporte desde una etapa previa, por lo que sólo es necesario resolver una función AND de 2 variables.

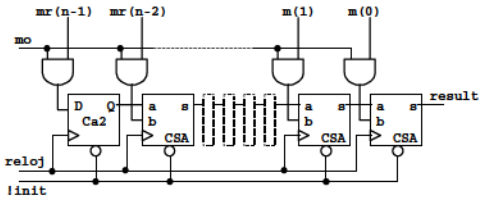
Los N-1 LEs restantes son necesarios para la salvaguarda del acarreo cuando se realiza esta suma, siendo necesarios un LE menos porque en la etapa más significativa, al no haber transporte desde una etapa previa, nunca se genera Carry y basta un único flipflop.

En consecuencia:

- el producto de 2 términos de N bits genera un resultado de 2N bits, por lo que luego del ingreso del MO deben ingresarse N bits en '0'.
- la latencia es de sólo un ciclo de reloj desde el ingreso del primer bit de MO hasta la salida del primer bit del producto
- todo el circuito requiere 2N-1 elementos lógicos

12.2.Un multiplicador serial para números con signo: Para multiplicar números con signo el circuito previo requiere sólo ligeros cambios.

- En el caso del multiplicando MO que ingresa en modo serie, debe realizarse la extensión de signo.
- En el caso del multiplicador MR debe considerarse que un '1' en el bit de signo de MR implica restar al producto el valor de MO x 2(N-1). Para ello, en la etapa que corresponde al bit MSB del producto basta agregar un circuito complementador entre la AND ya existente y el flipflop.



En AHDL la descripción de este multiplicador es:

```
PARAMETERS (N);
SUBDESIGN mulsign (
 mr[N-1..0],mo,sex : INPUT;
 init,reloj: INPUT;
 resul: OUTPUT;)

VARIABLE
 r[N-1..0],sgd: DFF;
 cs[N-2..0],ext: DFF;
 pp[N-1..0],mox: node;

BEGIN
 r  [].clk = reloj; sgd[].clk = reloj;
 cs [].clk = reloj; ext[].clk = reloj;
 r  [].clrn = !init;
 sgd[].clrn = !init;
 cs [].clrn = !init;

 IF (sex==GND)
  THEN mox =mo;ELSE mox =ext.q;END IF;
 sext.d = mox;

 FOR i IN 0 TO N-1 GENERATE
  pp[i] = mr[i] AND mox;
 END GENERATE;

---------- complementador
 sgd.d    = sgd.q OR pp[N-1];
 r[N-1].q = pp[N-1] XOR sgd.q;

---------- N-1 etapas carry save
 FOR i IN 0 TO N-2 GENERATE
  r[i].d =   pp[i] XOR r[i+1].q
         XOR cs[i].q;
  cs[i].d = (pp[i]   AND r[i+1].q)
         OR (pp[i]   AND cs[i].q)
         OR (cs[i].q AND r[i+1].q);
 END GENERATE;
 resul = r[0].q;
END;
```

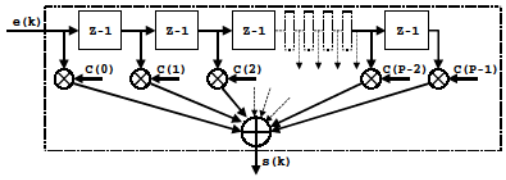
Y requiere 2N+2 LEs.

13.Ejemplo de aplicación: un filtro digital tipo FIR

Un filtro digital tipo FIR genera su salida S(k) en un instante k mediante la suma de P muestras de datos de entrada e(k) demoradas en el tiempo, cada una de ellas afectadas por un dado coeficiente C(i).

$$s(k) = \sum_{i=0}^{P-1} e(k-i).C(i)$$

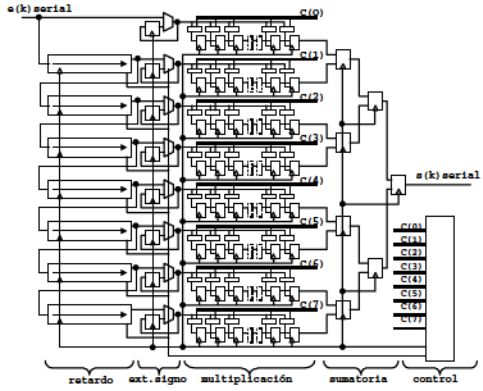
Este proceso corresponde a una convolución discreta, y un esquema global de solución puede ser:



Dado un filtro de P etapas (o taps) que procese datos de N bits, mediante aritmética serial es posible resolverlo usando:

- N.(P-1) LEs para las etapas de demora
- 2.P LEs para las etapas de extensión de signo
- 2.P.N LEs para los multiplicadores
- 2.P-1 LEs para sumar los términos
- P.N LEs para almacenar los coeficientes
- algunos LEs más para las tareas de control

Para datos de entrada con signo de N bits, el resultado de cada producto puede tener 2N-1 bits significativos, y la suma de P términos incrementa dicho resultado en ceil(log2(P)) bits significativos; a su vez la etapa de multiplicación agrega una latencia de un ciclo y la etapa de suma final una de ceil(log2(P)) ciclos, con lo que, para un caso hipotético de N=10 y P=8 esta solución requeriría alrededor de 350 elementos lógicos, con una latencia de 4 ciclos de reloj y aceptación de un nuevo dato cada 21 ciclos de reloj.



13.1.Posibles mejoras: Así como existen filtros con todos los C(i) distintos, se encuentran usualmente [1,2] filtros simétricos donde C(0)=C(P-1), C(1)=C(P-2), ..., etc. En este caso los datos afectados por coeficientes similares pueden ser sumados antes de la multiplicación, con importante ahorro de recursos. Para el ejemplo previo (P taps con datos de N bits), se necesitarían:

- N.(P-1) LEs para las etapas de demora
- 2.((P/2)-1) LEs para las etapas de extensión de signo
- P LEs para la suma de los pares de datos (P/2 sumadores Csave con 2 LEs c/u)
- (P/2).2.N LEs para los multiplicadores
- (P/2).N LEs para almacenar los coeficientes
- 2(P/2)-1 LEs para sumar los términos
- algunos LEs más para las tareas de control

Que resulta en alrededor de 220 LEs.

Finalmente, si los coeficientes no cambian en el tiempo, es también posible eliminar los registros necesarios para su almacenamiento; resultando para el ejemplo un consumo de recursos de menos de 200 LEs.

14.Conclusiones

En este artículo se ha mostrado la facilidad de diseño de módulos aritméticos de alta performance mediante el uso de lógica programable y cómo las soluciones seriales permiten ahorro de recursos con menores exigencias de conexionado, haciendo más eficiente el proceso de PPR (Partition, Place & Route).

A su vez la simplicidad circuital de los bloques combinatorios colocados entre registros minimiza los retardos y posibilita una elevada frecuencia de reloj, haciendo que el aparente costo de "más ciclos de reloj" sea en muchos casos compensado por ciclos más rápidos y circuitos más económicos.

15.Bibliografía

[1].Harris. ***"DIGITAL SIGNAL PROCESSING DATA BOOK"***. Harris Semiconductor, USA, 1994, DB302B.

[2].Harris. ***"SIGNAL PROCESSING NEW RELEASES"***. Harris Semiconductor, USA, 1995, DB314.

[3].Pollard, L.H. : ***"COMPUTER DESIGN AND ARCHITECTURE"***. Prentice Hall, USA, 1990. ISBN 0-13-167255-X.

[4].Smith, M.J.S.: ***"APPLICATION SPECIFIC INTEGRATED CIRCUITS"***. Addison Wesley, USA, 1997. ISBN 0-201-50022-1.

[5].Andraka, R. ***"BUILDING A HIGH PERFORMANCE BIT SERIAL PROCESSOR IN A FPGA"***. Proceedings of Design SuperCon'96, Juanuary 1996, pp.5.1-5.21

[6].Wakerly, J.F.: "DISEÑO DIGITAL: PRINCIPIOS Y PRÁCTICAS". Prentice Hall Hispanoamericana, México, 1992, ISBN 968-880-244-1.

[7].Hennessy,J. and Patterson, D.: ***"ARQUITECTURA DE COMPUTADORES. Un enfoque cualitativo".*** Mc.Graw Hill, USA, 1993. ISBN 84-7615-912-9.

[8].Altera Corp. ***"1999 DEVICE DATA BOOK",*** A-DB-0599-01, Altera Corp., 1999, USA.