Implementación de Sumadores rápidos Carry-Select en FPGA

Martín Vázquez y Nelson Acosta
INCA/INTIA – Facultad de Ciencias Exactas – UNICEN
Paraje Arroyo Seco, Campus Universitario – Tandil, Argentina
{ mvazquez | nacosta}@exa.unicen.edu.ar

RESUMEN: En este artículo se presenta una implementación eficiente de sumadores *carry-select* en FPGA. Se ajustó el diseño del los sumadores a las restricciones impuestas por las FPGAs pertenecientes a la familia Virtex II de Xilinx. Se analizaron varios sumadores con operandos de hasta 512 bits y con bloques condicionales de diferentes tamaños. Se verificó que dependiendo del tamaño de estos bloques, los sumadores *carry-select* implementados en este trabajo poseen mejores tiempos de computo respecto de los sumadores *ripple-carry*. Además se obtuvo una expresión matemática que puede utilizarse para determinar aproximadamente, en términos de velocidad, el tamaño óptimo de los bloques que conforman al sumador *carry-select*.

Palabras Claves: Sumador *Carry-Select*; RLOC; FPGA

1. Introducción

La alta densidad de las FPGA de hoy ha posibilitado el diseño de sistemas en un solo chip. El uso de núcleos de propiedad intelectual (*core-IP*) [1], en este tipo de sistemas tiene múltiples ventajas: se evita la utilización de componentes extras, se reduce las conexiones fuera del chip y, por último, la solución puede optimizarse respecto de la aplicación, considerando aspectos como frecuencia de operación, precisión, etc. La generación de modelos VHDL sintetizables que permitan implementar las principales funciones aritméticas con FPGA, es una parte esencial en un ambiente de diseño utilizando *cores-IP*.

Los sumadores binarios son fundamentales en una gran variedad de sistemas digitales y se utilizan en todas las operaciones aritméticas [2]. Abarcan un amplio rango de aplicaciones, desde el cómputo de una dirección física, cada vez que la computadora realiza una operación de *fetch*; hasta en algoritmos de encriptación, DSP, etc [3].

Las FPGA son particularmente atractivas para implementar sumadores que utilizan *ripple-carry* debido al uso de conexiones dedicadas que posibilitan la propagación rápida del *carry* [4]. Aquellos sumadores cuya arquitectura incluye *ripple-carry* pueden implementarse eficientemente en FPGAs, tal es el caso de sumadores como *carry-skip* [5], o como se muestra en este trabajo el sumador *carry-select*, entre otros.

Este trabajo presenta una implementación a medida (*full custom*) sobre FPGA de sumadores *carry-select*. Para ello, se realizaron descripciones en VHDL utilizando técnicas de emplazamiento relativo (*RLOC: relative location*) [6], sujetas a las restricciones impuestas por las FPGAs de la familia Virtex II de Xilinx.

Se analizaron varios sumadores, con operandos desde 64 bits hasta 512 bits y con bloques condicionales de diferentes tamaños. También se obtuvo una expresión matemática que posibilita determinar aproximadamente, cuál es el tamaño óptimo de los bloques que conforman el *carry-select* para que sea más rápido. Los resultados adquiridos en el presente trabajo son muy alentadores en el contexto de operaciones rápidas con operandos grandes.

2. Sumador carry-select

El sumador *carry-select* de *n* bits se descompone en grupos en *n/r* bloques sumadores *ripple-carry* de *r* bits. Cada bloque genera un resultado y un *carry*. El sumador *carry-select* (Fig. 1) se basa en la realización de dos sumas de grupos de bits en paralelo: uno con entrada de *carry* en 0 y otro con

entrada de *carry* en 1. Una vez calculado el valor real del *carry* se selecciona el resultado de la suma correspondiente.

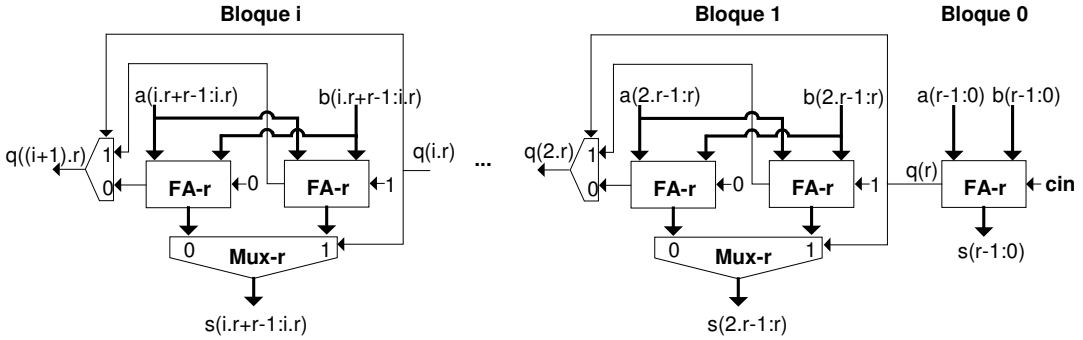

Figura 1. Arquitectura del sumador *carry-select* de *n* bits

El costo y tiempo de computación de un sumador *carry-select* de *n* bits se pueden modelar de la siguiente manera:

$$C_{Carry\text{-}Select}(n) = (2.(n/s\text{-}1) + 1).C_{FA\text{-}r} + (n/s - 1).(C_{Mux\text{-}r} + C_{Mux})$$
$$T_{Carry\text{-}Select}(n) = T_{FA\text{-}r} + (n/s - 1).T_{Mux}$$

en donde *Mux-r* está asociado a un multiplexor 2 a 1 de *r* bits, *Mux* a un multiplexor 2 a 1 de 1 bit y *FA-r* a un sumador completo de *r* bits (*full adder*).

3. Sumador ripple-carry en FPGA

En general, las FPGAs contienen recursos dedicados para generar sumadores rápidos. Los dispositivos programables pertenecientes a la familia Virtex II de Xilinx, incluyen puertas lógicas, multiplexores y tablas *look-up* de propósito general, que posibilitan construir de manera eficiente sumadores *ripple-carry*. En la Fig. 3 se aprecia como se utilizan los recursos en un CLB de Virtex II para implementar un sumador completo *ripple-carry* de 4 bits (CLB *Sum-4*).

La propagación de *carry* se realiza mediante el uso de conexiones dedicadas y el uso de multiplexores rápidos. Las tablas de *look-up* calculan la función de propagación de *carry*

$$p(i) = x(i)\ xor\ y(i)$$

de esta manera

$$q(i+1) = g(i) + p(i).q(i)$$

se deduce

$$g(i) + p(i).q(i) = a(i).b(i) + (a(i)\ xor\ b(i)).q(i) =$$
$$not(a(i)\ xor\ b(i)).a(\mathrm{i}) + (a(i)\ xor\ b(i)).q(\mathrm{i}) =$$
$$not(p(i)).a(\mathrm{i}) + p(i).q(\mathrm{i})$$

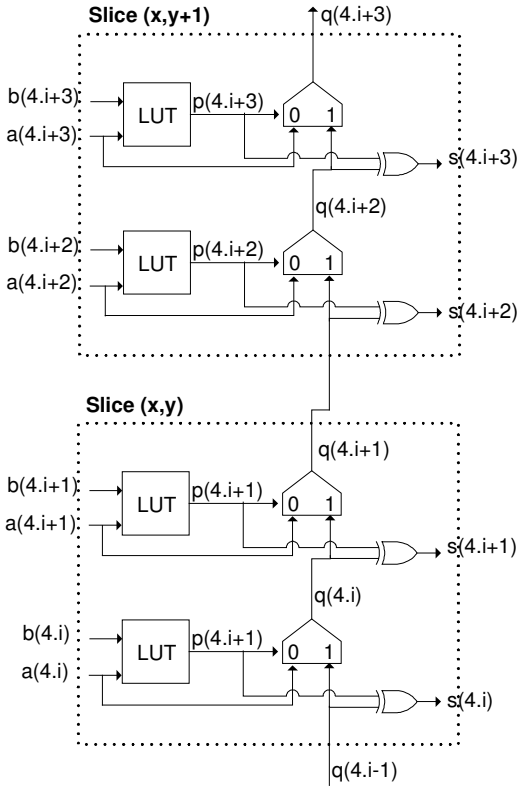

Figura 2. Implementación de un sumador *ripple-carry* de 4 bits en Virtex II

El tiempo de cálculo $T_{Ripple}(n)$ de un sumador *ripple-carry* de n bits es

$$T_{Ripple}(n) = t_{LUT} + n.t_{mux\text{-}cy}$$

donde t_{LUT} es el tiempo de cómputo de una tabla *look-up* y $t_{mux\text{-}cy}$ es el retraso del multiplexor dedicado junto con la conexión al *slice* adyacente.

El $t_{mux\text{-}cy}$ es mucho menor que el retraso que se produce cuando se genera un multiplexor con una tabla de *look-up* y se establece una conexión de propósito general.

El costo $C_{Ripple}(n)$ asociado a la cantidad de *slices* que ocupa la implementación el sumador *ripple-carry* de n bits es

$$C_{Ripple}(n) = n/2 \ slices$$

4. Implementación del sumador carry-select en FPGA

Se implementaron varios sumadores en una FPGA de la familia Virtex II de Xilinx (xc2v4000-6ff517), con una matriz de 144x160 slices. La síntesis se realizó con XST (Xilinx Synthesis Technology) [7] y la implementación física con el Xilinx ISE (Integrated System Environment) versión 6.3i [8]. Para aprovechar al máximo los recursos de la FPGA, el diseño se efectuó instanciando componentes de bajo nivel mediante la utilización de técnicas de RLOC.

La implementación de cada bloque i (Blq_i) de r bits (con i>0) que conforma al sumador *carry-select* de n bits, se compone de r/8 subbloques (*SubBlq*) de 8 bits. En la fig. 3 se observa la implementación en Vitrtex II del subbloque j de 8 bits perteneciente a un bloque i de r bits ($SubBlq_{i,j}$). La implementación

de los *slices* del CLB *SumSel-4* son similares a los *slices* del CLB *Sum-4* descriptos en la Fig. 2, la diferencia es que el CLB *SumSel-4* implementa los dos sumadores alternativos, uno para *carry* de entrada en 0 y otro para *carry* de entrada en 1. El CLB *Mux-8* implementa un multiplexor 2 a 1 de 8 bits.

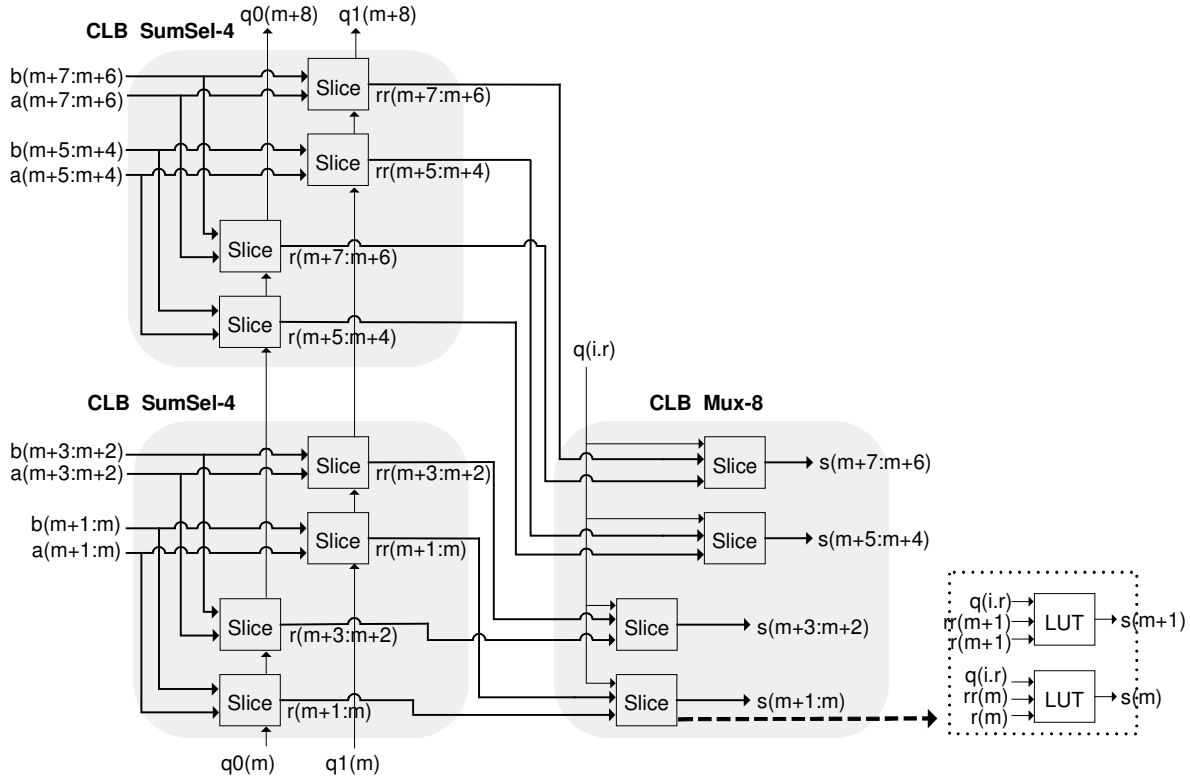

Figura 3. Implementación en Vitrtex II de un $SubBlq_{i,j}$ de 8 bits. Donde $m = i.r + 8.j$

El costo C_{SubBlq} asociado a cada subbloque de 8 bits es de 12 *slices*
La Fig. 4 muestra como se utilizaron los CLBs en Virtex II para implementar un bloque i de r bits. El CLB *Mux-1* sólo utiliza una LUT de un *slice*, la cual describe un multiplexor 2 a 1 de 1 bit.
El costo $C_{Blq\ i}$ de un bloque i (i>0) de r bits está determinado por:

$$C_{Blq\ i} = (r/8).C_{SubBlq} + C_{Mux\text{-}1} = (r/8).12 + 1 = (3/2).r + 1 \ slices$$

El bloque 0 implementa un sumador completo de r bits. El costo $C_{Blq\ 0}$ del bloque 0 de r bits es:

$$C_{Blq\ 0} = r/2 \ slices$$

En la fig. 5 se observa el enplazado en la FPGA de los diferentes bloques que conforman el sumador *carry-select* de n bits.

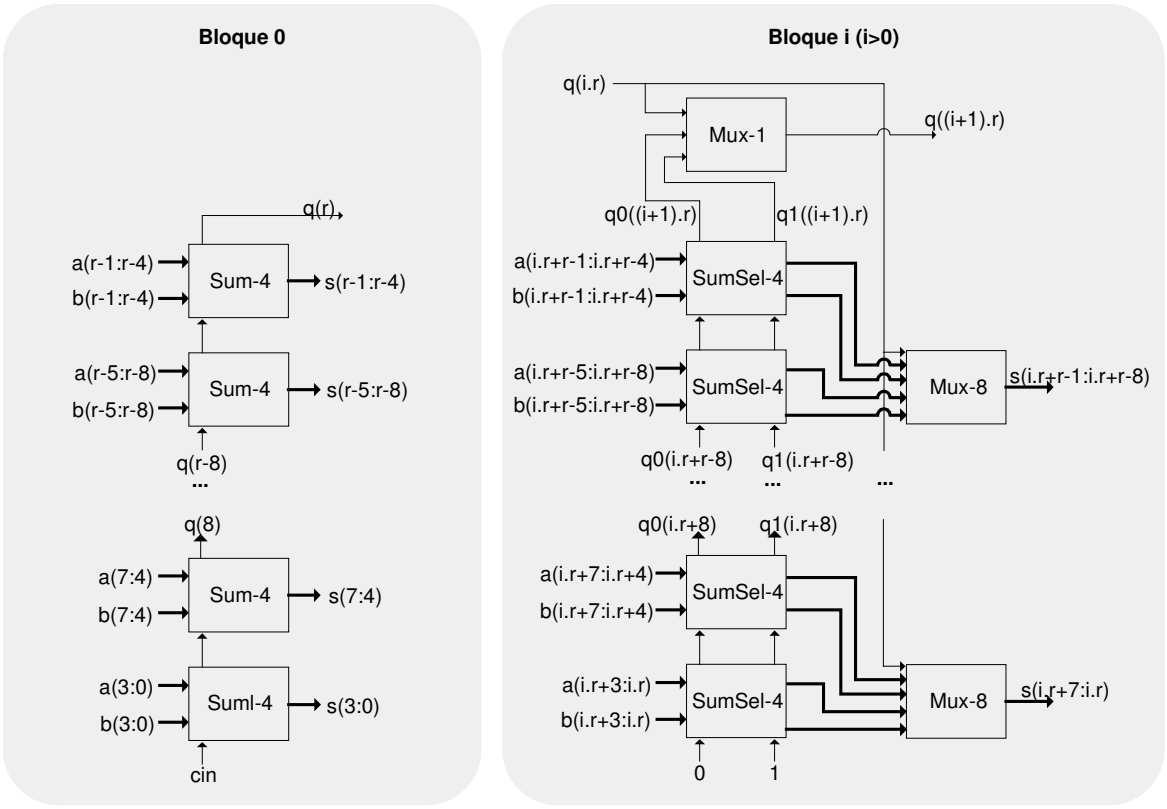

Figura 4. Implementación en Vitrtex II de un bloque *i* de *r* bits.

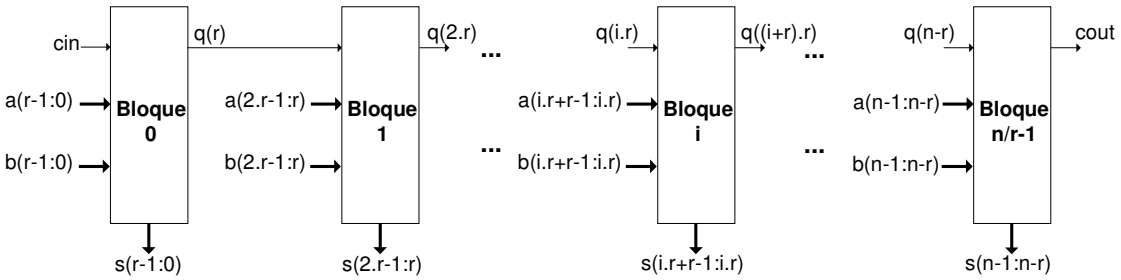

Figura 5. Emplazado del sumador *carry-select* de *n* bits.

El costo $C_{Carrry\text{-}Select}$ de implementar el sumador *carry-select* de *n* bits con bloques de *r* bits en una Virtex II es

$$C_{Carrry\text{-}Select}(n) = C_{Blq\ 0} + (n/r\text{-}1).C_{Blq\ i} = r/2 + (n/r\text{-}1).((3/2).r + 1) \ \ slices$$

$$C_{Carrry\text{-}Select}(n) = (3/2 + 1/r\ ).n - r - 1 \ \ slices \qquad \textbf{(1)}$$

Expresando el costo $C_{Carrry\text{-}Select}(n)$ en función del tamaño de bloque se obtiene

$$g(r) = C_{Carrry\text{-}Select}(n) = (3/2 + 1/r\ ).n - r - 1$$

se observa que para sumadores del mismo tamaño, el costo disminuye a medida que crece r.
Para el análisis de tiempos que se realiza a continuación se consideran los siguientes aspectos: a) t_{cnx1} está asociado al tiempo de conexión promedio entre un *slice* del CLB *SumSel-4* a un *slice* del CLB *Mux-8*, o al tiempo de conexión de las señales $a(i)$ y $b(i)$, o al tiempo entre un *slice* del CLB *SumSel-4* y un *slice* correspondiente al CLB *Mux-1* (Fig. 3 y Fig. 4); b) t_{cnx2} está asociado al tiempo de conexión promedio entre la salida asociada a $q(i.r)$ y un *slice* de *Mux-1*; y c) t_{cnx3} es el tiempo de conexión promedio entre la salida asociada a $q(i.r)$ y un *slice* correspondiente a *Mux-8*. Se puede verificar fácilmente que $t_{cnx3} > t_{cnx2} > t_{cnx1}$. Se asume además, que t_k es el tiempo de computo que insume calcular la salida k.
Para el bloque 0:

$$\begin{aligned} t_{q(r)} &= T_{Ripple} + t_{cnx1} \\ &= t_{LUT} + r.t_{mux\text{-}cy} + t_{cnx1} \\ t_{s(r-1:0)} &= t_{LUT} + r.t_{mux\text{-}cy} + t_{cnx1} \end{aligned}$$

Para el bloque 1:

$$\begin{aligned} t_{q(2.r)} &= \max\{\ t_{q(r)} + t_{cnx2},\ T_{Ripple} + 2.t_{cnx1}\} + t_{LUT} \\ &= 2.t_{LUT} + r.t_{mux\text{-}cy} + t_{cnx2} + t_{cnx1} \\ t_{s(2.r-1:r)} &= \max\{\ t_{q(r)} + t_{cnx3},\ T_{Ripple} + 2.t_{cnx1}\} + t_{LUT} \\ &= 2.t_{LUT} + r.t_{mux\text{-}cy} + t_{cnx3} + t_{cnx1} \end{aligned}$$

Para el bloque 2

$$\begin{aligned} t_{q(3.r)} &= \max\{\ t_{q(2.r)} + t_{cnx2},\ T_{Ripple} + 2.t_{cnx1}\} + t_{LUT} \\ &= 3.t_{LUT} + r.t_{mux\text{-}cy} + 2.t_{cnx2} + t_{cnx1} \\ t_{s(2.r+r-1:2.r)} &= \max\{\ t_{q(2.r)} + t_{cnx3},\ T_{Ripple} + 2.t_{cnx1}\} + t_{LUT} \\ &= 3.t_{LUT} + r.t_{mux\text{-}cy} + t_{cnx2} + t_{cnx3} + t_{cnx1} \end{aligned}$$

Para el bloque 3

$$\begin{aligned} t_{q(4.r)} &= t_{q(3.r)} + t_{cnx2} + t_{LUT} \\ &= 4.t_{LUT} + r.t_{mux\text{-}cy} + 3.t_{cnx2} + t_{cnx1} \\ t_{s(3.r+r-1:3.r)} &= t_{q(3.r)} + t_{cnx3} + t_{LUT} \\ &= 4.t_{LUT} + r.t_{mux\text{-}cy} + 2.t_{cnx2} + t_{cnx3} + t_{cnx1} \end{aligned}$$

Para el bloque i

$$\begin{aligned} t_{q((i+1).r)} &= (i+1).t_{LUT} + r.t_{mux\text{-}cy} + i.t_{cnx2} + t_{cnx1} \\ t_{s(i.r+r-1:i.r)} &= (i+1).t_{LUT} + r.t_{mux\text{-}cy} + (i-1).t_{cnx2} + t_{cnx3} + t_{cnx1} \end{aligned}$$

Considerando $m = n/r$, los tiempos asociados a las salidas de último bloque (m-1) son:

$$\begin{aligned} t_{q(m.r)} &= m.t_{LUT} + r.t_{mux\text{-}cy} + (m-1).t_{cnx2} + t_{cnx1} \\ t_{q(n)} &= (n/r).t_{LUT} + r.t_{mux\text{-}cy} + (n/r-1).t_{cnx2} + t_{cnx1} \end{aligned}$$

$$\begin{aligned} t_{s(m.r-1:m.r-r)} &= m.t_{LUT} + r.t_{mux\text{-}cy} + (m-2).t_{cnx2} + t_{cnx3} + t_{cnx1} \\ t_{s(n-1:n-r)} &= (n/r).t_{LUT} + r.t_{mux\text{-}cy} + (n/r-2).t_{cnx2} + t_{cnx3} + t_{cnx1} \end{aligned}$$

Por consiguiente el tiempo de cómputo máximo $T_{Carry\text{-}Select}(n)$ asociado a la implementación del sumador *carry-select* de n bits con bloques de r bits en una Virtex II es

$$T_{Carrry\text{-}Select}(n) = (n/r).t_{LUT} + r.t_{mux\text{-}cy} + (n/r\ -2).t_{cnx2} + t_{cnx3} + t_{cnx1} \qquad \textbf{(2)}$$

Asumiendo $C = t_{cnx3} + t_{cnx1} - 2.t_{cnx2}$, la ecuación anterior (2) se puede expresar de la siguiente manera

$$T_{Carrry\text{-}Select}(n) = (n/r).(t_{LUT} + t_{cnx2}) + r.t_{mux\text{-}cy} + C$$

Expresando el tiempo de cómputo $T_{Carrry\text{-}Select}(n)$ en función del tamaño de bloque

$$f(r) = T_{Carrry\text{-}Select}(n) = (n/r).(t_{LUT} + t_{cnx2}) + r.t_{mux\text{-}cy} + C$$

para encontrar el valor aproximado de r que minimice la expresión $f(r)$

$$f\,'(r) = -(n/r^2).(t_{LUT} + t_{cnx2}) + t_{mux\text{-}cy} = 0$$

entonces

$$r = [n.(t_{LUT} + t_{cnx2})/t_{mux\text{-}cy}]^{½}$$

Experimentalmente se verificó que

$$(t_{LUT} + t_{cnx2})/\ t_{mux\text{-}cy} \approx 31$$

de este modo

$$r\ \approx 5{,}5\ .(n)^{½} \qquad \textbf{(3)}$$

5. Resultados Experimentales

En la tabla 1 se listan los peores retrasos obtenidos por la implementación de los sumadores *carry-select*, para diferentes tamaños de operandos y bloques (n y r respectivamente). Para el caso que el tamaño de los operandos es igual al tamaño de los bloques (n=r), se obtiene un sumador *ripple-carry* convencional. Nótese que existen celdas en las cuales no existe información, esto es porque n no es divisible por r, o bien, en el caso de (*), porque la implementación propuesta en este trabajo para el sumador con estos parámetros (n y r) no se puede emplazar, ya que el número de columnas de CLBs para esa cantidad de bloques (n/r) excede el límite de la FPGA utilizada (xc2v4000-6ff517).

n \ *r*	**8**	**16**	**32**	**64**	**80**	**96**	**112**	**128**	**160**	**192**	**224**	**256**	**320**	**384**	**448**	**512**
64	10,9	7,2	**5,7**	6	-	-	-	-	-	-	-	-	-	-	-	-
128	19	11,1	8,5	**7,5**	-	-	-	8,7	-	-	-	-	-	-	-	-
192	28	16,1	10,4	**9,5**	-	9,6	-	-	-	11,3	-	-	-	-	-	-
256	(*)	19	12,3	**9,9**	-	-	-	11,5	-	-	-	14	-	-	-	-
320	(*)	23,8	14,8	11,2	**11,2**		-	-	13,7	-	-	-	19,2	-	-	-
384	(*)	(*)	16,8	12,9	-	**12,5**	-	12,6	-	15,3	-	-	-	22,8	-	-
448	(*)	(*)	19	14	-	-	**12,8**	-	-	-	15,8	-	-	-	26,3	-
512	(*)	(*)	20,9	15	-	-	-	**14,9**	-	-	-	17,9	-	-	-	29,7

Tabla 1. Resultados experimentales: Perores tiempos de cálculo en ns.

La tabla 2 muestra la evaluación de la expresión (3) para los diferentes tipos de operandos y la aproximación al tamaño real del bloque (r). Se tomó como aproximación, el r divisor de n más cercano al resultado producido por (3). Se observa que la expresión matemática desarrollada para determinar el tamaño de bloque óptimo de manera que el sumador *carry-select* de n bits sea más rápido, produce un valor que aproximado a r_{aprox}, coincide con el tamaño de r resaltado para cada n en la tabla 1.

n	$5{,}5.(n)^{1/2}$	r_{aprox}
64	44	32
128	62	64
192	76	64
256	88	64
320	98	80
384	108	96
448	116	112
512	124	128

Tabla 2. Validación de la expresión que determina *r* optimo para sumador más rápido

En tabla 3 se aprecia el área ocupada por la implementación de los diferentes sumadores. El número de *slices* ocupado por cada sumador está determinado por (1), se observa que el sumador *ripple-carry* (n=r) presenta la menor ocupación de área.

n \ r	**8**	**16**	**32**	**64**	**80**	**96**	**112**	**128**	**160**	**192**	**224**	**256**	**320**	**384**	**448**	**512**
64	95	83	65	**32**	-	-	-	-	-	-	-	-	-	-	-	-
128	199	183	163	129	-	-	-	**64**	-	-	-	-	-	-	-	-
192	303	283	261	226	-	193	-	-	-	**96**	-	-	-	-	-	-
256	(*)	383	359	323	-	-	-	257	-	-	-	**128**	-	-	-	-
320	(*)	483	457	420	403	-	-	-	321	-	-	-	**160**	-	-	-
384	(*)	(*)	555	517	-	483	-	450	-	385	-	-	-	**192**	-	-
448	(*)	(*)	653	614	-	-	563	-	-	-	449	-	-	-	**224**	
512	(*)	(*)	751	711	-	-	-	643	-	-	-	513	-	-	-	**256**

Tabla 3. Resultados experimentales: Ocupación de *slices*.

Para determinar el tamaño de bloque que maximice la relación tiempo de cómputo y ocupación de *slices* (tiempo-área),se consideró la siguiente expresión

$$\delta(n,r) = 1/ [\ T(n,r) * C(n,r)\ ] \qquad \textbf{(4)}$$

donde $T(n,r)$ y $C(n,r)$ son el tiempo de cómputo y el costo en área respectivamente, asociados al sumador de n bits con tamaño de bloque de r bits. De este modo, para un determinado n, el r óptimo es

$$r_{opt}(n) = \max\ (\delta(n,r)) \ \text{ para todo } r \text{ divisor de } n \qquad \textbf{(5)}$$

La tabla 4 muestra la relación tiempo-área (δ) para cada sumador implementado. Para realizar el cálculo de (4), se expresó el tiempo de cómputo $T(n,r)$ en microsegundos. Se observa que para todos los tamaños de operandos estudiados, el sumador *ripple-carry* (r_{opt}=n) posee la mejor relación tiempo-área.

n \ *r*	8	16	32	64	80	96	112	128	160	192	224	256	320	384	448	512
64	0,97	1,67	2,7	**5,21**	-	-	-	-	-	-	-	-	-	-	-	-
128	0,26	0,49	0,72	1,03	-	-	-	**1,8**	-	-	-	-	-	-	-	-
192	0,12	0,22	0,37	0,47	-	0,54	-	-	-	**0,92**	-	-	-	-	-	-
256	-	0,14	0,23	0,31	-	-	-	0,34	-	-	-	**0,56**	-	-	-	-
320	-	0,89	0,15	0,21	0,22	-	-	-	0,23	-	-	-	**0,33**	-	-	-
384	-	-	0,11	0,15	-	0,17	-	1,77		0,17	-	-	-	**0,23**	-	-
448	-	-	0,08	0,12	-	-	0,14	-	-	-	0,14	-	-	-	**0,17**	
512	-	-	0,06	0,09	-	-	-	0,1	-	-	-	0,11	-	-	-	**0,13**

Tabla 4. Relación tiempo-área (δ).

6. Conclusiones

En el presente trabajo se realizó la implementación eficiente en FPGA de diferentes sumadores *carry-select*, con operandos de hasta 512 bits y bloques de diferentes tamaños. Para ello se utilizó la técnica de emplazado relativo.

Se obtuvo que los sumadores *carry-select*, para ciertos tamaños de bloque, son más rápidos que los sumadores *ripple-carry*. En cuanto al costo, se verificó que éste disminuye a medida que aumenta el tamaño de bloque, obteniéndose que el sumador *ripple-carry* posee el menor costo.

Además se determinó que el sumador *ripple-carry* es el que posee la mejor relación tiempo-área.

Se desarrolló y validó experimentalmente una expresión matemática que posibilita determinar a priori, el tamaño de bloque que permite que un sumador *carry-select* de *n* bits sea más eficiente en términos de velocidad o tiempo de cómputo.

7. Referencias

[1] G. Spivey, S. Bhattacharyya, K. Nakajima, "Logic Foundry: Rapid Prototyping for FPGA-Based DSP Systems". EURASIP Journal on Applied Signal Processing 2003, pp 565-579. Hindawi Publishing Corporation, 2003

[2] P. Corsonello, V. Kantabutra, S. Perri, "Fast, Low-Cost Adders Using Carry Strength Signals", SSGRR 2000, l'Aquila, Italia, Julio 31-Aug 6.

[3] E. Jamro, K. Wiatr, "Convolution operation implemented in FPGA structures for real-time image processing" Ernest Jamro, Kazimierz Wiatr, ISPA 2001 : proceedings of the 2nd international symposium on Image and Signal Processing and Analysis : in conjunction with 23nd international conference on Information technology interfaces, Junio 2001

[3] P. Corsonello, V. Kantabutra, S. Perri, "Fast, Low-Cost Adders Using Carry Strength Signals", SSGRR 2000, l'Aquila, Italia, Julio 31-Aug 6.

[4] Xilinx, inc. "Virtex-II Platform FPGAs: Complete Data Sheet ", Marzo 2005, disponible en http://support.xilinx.com

[5] G.Bioul, J-P.Deschamps, G. Sutter, "Efficient FPGA Implementation of Carry-Skip Adders", III JCRA, España, Sept 2003. ISBN 84-600-9928-8. Pp 81-90.

[6] Xilinx, inc. "Constraints Guide – ISE6.3i", capítulo 2 Relative Location (RLOC), Agosto 2004, disponible en http://support.xilinx.com.

[7] Xilinx, inc. "Xilinx Synthesis Technology (XST) User Guide", 2004, available at http://support.xilinx.com

[8] Xilinx, inc. "ISE6.3i Documentation", 2004, disponible en http://support.xilinx.com