DISEÑO DE DIVISORES PARALELOS Y SECUENCIALES DE N/N BITS USANDO FPGAS

John M. Espinosa-Duran, Juan G. Catuche-Girón, Mario E. Vera-Lizcano, Jaime Velasco-Medina
Grupo de Bio-nanoelectrónica
Escuela EIEE. Universidad del Valle. A. A. 25360, Cali, Colombia
E-mail: {michaele, juangaca, mario, jvelasco} @univalle.edu.co

ABSTRACT

Este artículo presenta el diseño de divisores paralelos y secuenciales de N/N bits usando FPGAs. En la literatura existen varios trabajos que presentan el diseño de divisores de 2N/N bits usando los algoritmos con y sin restauración, los cuales presentan serias limitaciones, como por ejemplo el rango de valores y tamaño de los operandos. En este contexto, este trabajo presenta el diseño de divisores eficientes que permiten realizar la división entre operandos de N bits, es decir N/N bits. Los diseños son realizados usando una descripción estructural (tipo genérico) en VHDL y son sintetizados sobre un Stratix II EP2S15F484C3 usando QUARTUS II versión 4.1sp2. Con el propósito de verificar el desempeño de los divisores N/N bits, los resultados de simulación son comparados con el "core" de Altera y los divisores 2N/N bits. En este caso, los divisores diseñados son una muy buena alternativa para ser usados en el diseño de sistemas embebidos.

Palabras Clave --- *FPGA, Divisores Paralelos, Divisores Secuenciales, VHDL*

1. INTRODUCCIÓN

La operación de división es más compleja que las operaciones de suma, resta o multiplicación. Inicialmente en algunos procesadores, la división es llevada a cabo mediante rutinas de software, las cuales básicamente realizan la división como una secuencia de restas y desplazamientos. Sin embargo, los procesadores de alto desempeño, disponen de divisores implementados en hardware para incrementar la velocidad de las operaciones aritméticas.

De otro lado, hace algunos años era muy difícil realizar diseños complejos, para volúmenes pequeños o medianos de producción. Sin embargo, el vertiginoso avance de la tecnología permite realizar fácilmente diseños complejos y de bajo costo, es decir, es posible diseñar sistemas embebidos totalmente a la medida para una aplicación dada, y aún para volúmenes muy pequeños, usando dispositivos FPGAs.

En este contexto, los sistemas embebidos son cada día más utilizados para diferentes aplicaciones. Por lo tanto, los principales bloques funcionales de un sistema embebido deben ser optimizados para los diferentes parámetros de diseño. Por ejemplo, en la compresión de video, el bloque funcional que realiza la división debe ser de alto desempeño. Sin embargo, en dispositivos portátiles para telefonía celular, los principales criterios de diseño son relacionados con el área y la potencia disipada [1]. En el caso de diseño digital basado en FPGAs, es importante mencionar que implementar la función de la división utilizando la megafunción lpm_divide de Altera, no es una buena alternativa de diseño, esta es limitada en velocidad.

Teniendo en cuenta las consideraciones anteriores, este artículo presenta el diseño de divisores paralelos y secuenciales de N/N bits y presenta una comparación entre las diferentes arquitecturas básicas en hardware que implementan la división basados en los algoritmos con y sin restauración, considerando los parámetros de diseño tales como área y velocidad.

El artículo esta organizado de la siguiente manera, en la sección 2 se presenta una descripción de los algoritmos con y sin restauración usados para la división. En la sección 3, las arquitecturas básicas en hardware para los divisores paralelos y secuenciales son presentadas. En la sección 4 se presentan los resultados de simulación y las tablas de comparación para los divisores implementados, y finalmente en la sección 5 se presentan las conclusiones y el trabajo futuro.

2. ALGORITMOS BÁSICOS PARA LA DIVISION

En esta sección se presenta una breve descripción de los algoritmos básicos usados para la implementación en hardware de los divisores paralelos y secuenciales.

2.1 Algoritmo de división con restauración de 2N/N bits.

La principal consideración para implementar en hardware los divisores basados en el algoritmo con restauración es que dados dos enteros N y D tal que $N < D$, existen dos enteros únicos Q y R que satisfacen la ecuación: $2N = BQ + R$. La propiedad que justifica la utilización del algoritmo con restauración se fundamenta en la propiedad $N < D$, y entre el residuo y el divisor, $R < D$.

En este caso, el circuito más simple para implementar la división binaria debe acomodar metódicamente el divisor con relación al dividendo y realizar una resta. Si el valor del residuo es cero o positivo, se determina el bit del cociente como 1, y el residuo se amplía con otro bit del dividendo, entonces el divisor se acomoda y se efectúa otra resta. Por otra parte, si el valor del residuo es negativo, se determina el bit

del cociente como 0, y el dividendo se restaura sumándole de nuevo el divisor, y entonces este divisor se acomoda para otra resta [2]. El algoritmo con restauración para la división se puede entender mediante el ejemplo mostrado en la Figura 1.

Dividendo → **11011001** (217), Divisor → **1011** (11)
Cociente → **10011** (19), Residuo → **1000** (8)

```
      10011  Cociente
1011) 11011001   Dividendo
      1011       Divisor
       0101      Dividendo reducido
        0000     Divisor 0 por ser mayor que el
                  dividendo reducido
         1010    Dividendo reducido
         0000    Divisor 0 por ser mayor que el
                  dividendo reducido
         10100   Dividendo reducido
          1011   Divisor desplazado
          10011  Dividendo reducido
           1011  Divisor desplazado
           1000  Residuo
```

Figura 1. División binaria basada en el algoritmo con restauración

Como se observa en el ejemplo de la Figura 1, el algoritmo consiste en: Realizar una resta entre el dividendo y el divisor, pero desde el bit más significativo (MSB). Si el MSB del resultado es uno (1) entonces en el cociente se coloca un cero (0) y se adiciona el siguiente bit del dividendo y se desplaza el divisor, pero si el MSB del resultado es cero (0), entonces se coloca un uno (1) en el cociente, y se desplaza el divisor. Todo lo anterior se hace N (número de bits del divisor) veces.

2.2 Algoritmo de división con restauración de N/N bits.

En este trabajo se propone una modificación al algoritmo anterior para poder realizar la operación entre operandos de N bits, el cual es presentado en la Figura 2.

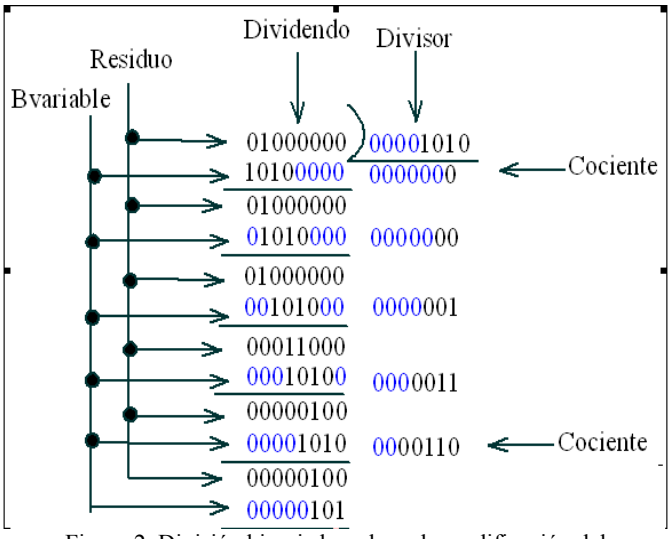


Figura 2. División binaria basada en la modificación del algoritmo con restauración para operándos de N bits

El algoritmo consiste en: inicialmente, se obtiene el divisor modificado (Bvariable inicial), el cual se obtiene detectando el bit más significativo en uno (1) del divisor (en la posición *P*) y desplazando este bit *N-P* veces hacia la posición más significativa (por ejemplo: divisor = 00001010 y Bvariable inicial = 10100000). En la Figura 2, el bit mas significativo en uno (1) del divisor es el bit en la posición 4 (P_4), el cual es desplazado hacia la posición 8 (P_8). Luego, se compara el Bvariable inicial con el dividendo, si el dividendo es mayor, se realiza la resta y se desplaza hacia la izquierda el registro que contiene el cociente con un uno (1), en caso contrario, no se realiza la resta y el registro del cociente se desplaza con un cero (0), esta iteración (comparación, resta, desplazamiento) se realizara hasta que el valor de Bvariable sea menor que el valor del divisor (*P+1* veces). En el ejemplo de la Figura 2, el divisor es 10 y una vez modificado se convierte en 160 (10100000), el cual es mucho mayor que 64 (01000000), por tanto la resta no se realiza y el registro del cociente se desplaza con un cero (0).

En la segunda iteración, el divisor se desplaza a la derecha una posición y el nuevo valor del divisor es 80 (01010000), el cual es mayor que el dividendo, entonces la resta no se realiza y el registro del cociente se desplaza con un cero (0). Para la tercera iteración, el divisor tendrá un valor de 40 (00101000), el cual es menor que el dividendo, entonces la resta se realiza y el registro del cociente se desplaza con un uno (1). En la cuarta iteración, el nuevo valor del divisor es 20 (00010100), el cual es menor que el obtenido de la resta anterior (24), entonces se realiza la resta y el registro del cociente se desplazara con un uno (1).

Las iteraciones terminan cuando se detecte que el valor de Bvariable es menor que el valor inicial del divisor, entonces el valor del residuo es el resultado de la ultima resta. En el ejemplo, el resultado de la ultima resta es 4.

2.3 Divisor sin restauración de 2N/N bits.

La principal consideración para implementar en hardware divisores basados en el algoritmo sin restauración es que dados dos enteros N y D tal que $-D \le N < D$, existen dos enteros, Q y R que satisfacen la ecuación: $2N = (2Q-1)B + R$. Esta ecuación puede ser re-escrita como: $2N + 2D = 2BQ + B + R$.

Es conocido por el teorema de la división de enteros que, dados dos enteros (2N + 2D) y 2D, se puede encontrar un único cociente Q y un único residuo (B + R) que satisfagan la ecuación anterior y reflejando la propiedad al residuo ($-D \le R < D$). En conclusión, la principal consideración para este algoritmo es que los dos números (enteros positivos) satisfagan la siguiente condición: $0 \le N < D$ [3].

Es posible mejorar la división basada en el algoritmo sin restauración evitando restaurar el dividendo después de una resta sin éxito, es decir, si el resultado es negativo.

En la Figura 3 se muestra la división basada en el algoritmo sin restauración. Esta figura presenta un ejemplo en el cual se observa que en el primer ciclo se realiza la resta entre el dividendo y el divisor, pero desde el bit MSB. Si el bit MSB del resultado es uno (1), entonces se coloca un cero (0) en el cociente y se suma el resultado con el divisor desplazado; si es

cero (0), entonces se coloca un uno (1) en el cociente y se resta el resultado con el divisor desplazado. Después de realizar N desplazamientos, si el bit MSB es uno (1), entonces se suma el resultado y el divisor no es desplazado, pero si es cero (0) no se hace ninguna operación [4].

Dividendo → **1000** (8), Divisor → **11** (3)
Cociente → **10** (2), Residuo → **10** (2)

```
          0010  Cociente
00011) 00001000   Dividendo
       00011      Resta (1er ciclo)
       111100     MSB = 1, entonces 0 en cociente y
                  se suma, desplazando el Divisor
        00011     Divisor desplazado (2do ciclo)
        111110    MSB = 1, entonces 0 en cociente y
                  se suma, desplazando el Divisor
        00011     Divisor desplazado (3do ciclo)
        000010    MSB = 0, entonces 1 en cociente y
                  se resta, desplazando el Divisor
        00011     Divisor desplazado (4to ciclo)
        11111     MSB = 1, entonces 0 en coc. y se
                  suma el Divisor. Si MSB = 0,
        00011     entonces 1 en cociente y no se hace
                  ninguna operación.
        00010     Residuo
```

Figura 3. División binaria basada en el algoritmo sin restauración.

3. ARQUITECTURAS HARWARE PARA LA DIVISION

En esta sección se presentan las arquitecturas implementadas en hardware para los divisores paralelos y secuenciales de 2N/N y N/N bits, los cuales son basados en los algoritmos con y sin restauración.

3.1 Divisores paralelos de 2N/N y N/N bits

3.1.1 Divisor paralelo con restauración de 2N/N bits

Una arquitectura para el divisor con restauración se muestra en la Figura 4. En este caso, se utiliza la celda RC como celda básica, la cual se muestra en la Figura 5 [3], donde las ecuaciones que controlan este restador son:

$$P = \overline{A}.B + \overline{A}.C + B.C = \overline{A}.(B + C) + B.C \quad (1)$$

$$S = AD + A.\overline{B}.\overline{C} + ABC + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.\overline{B}.C.\overline{D} \quad (2)$$
$$= A.\left(D + \overline{(B \oplus C)}\right) + \overline{A}.\overline{D}(B \oplus C)$$

Donde se observa que:

$S = A \oplus B \oplus C$, si D = 0 (3)
$S = A$, si D = 1 (4)

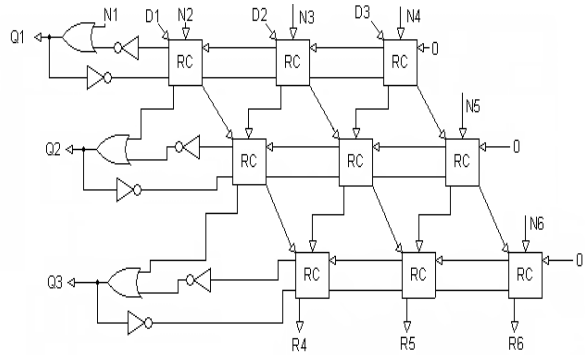


Figura 4. Divisor paralelo con restauración para N=3 bits

El circuito calcula los cocientes $Q = 0.Q_0Q_1Q_2$ y los residuos $R = 0.R_0R_1R_2R_3R_4R_5$ de $N=0.N_0N_1N_2N_3N_4N_5$ para $D=0.D_0D_1D_2$ bajo la premisa que N < D. Por ejemplo, para 3 bits, N=0.100011 y D=0.101, entonces Q=0.111 y R=0.000000.

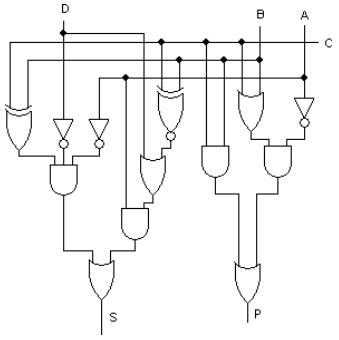


Figura 5. Celda básica RC para la división con restauración

3.1.2 Divisor paralelo con restauración de N/N bits

Una arquitectura para el divisor basado en el algoritmo de restauración modificado se muestra en la Figura 6. En este caso, se utiliza la celda básica que muestra en la Figura 7.

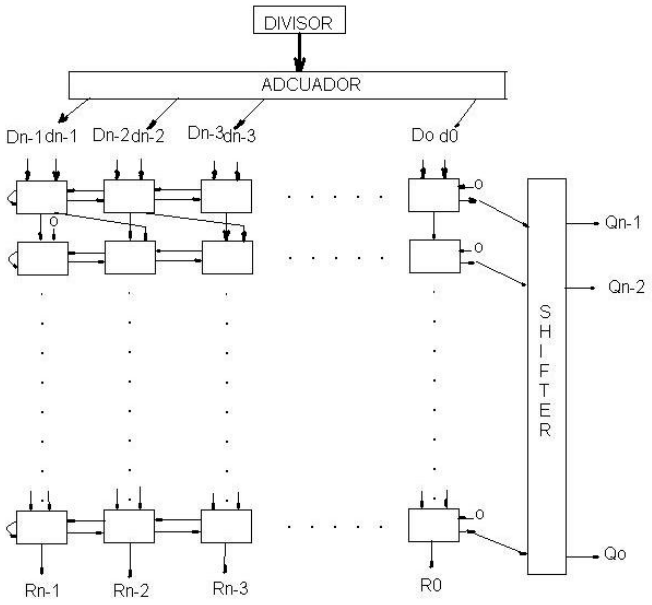


Figura 6. Divisor paralelo con restauración para N-bits

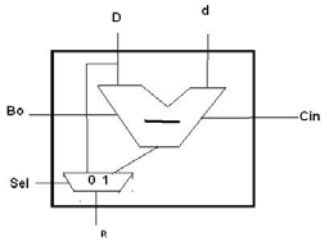


Figura 7. Celda básica para la división con restauración, vista interna.

3.1.3 Divisor sin restauración de 2N/N bits.

En la Figura 8 se muestra un arreglo que implementa la división basada en el algoritmo de la Figura 3. El diagrama mostrado en la Figura 9 es la celda básica CD usada por el divisor basado en el algoritmo sin restauración.

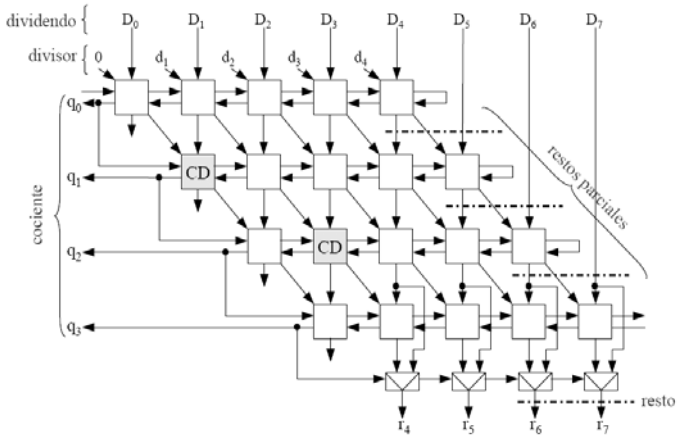


Figura 8. Divisor paralelo sin restauración: N=4bits

La celda básica CD de la Figura 9, es en esencia un circuito sumador-restador, el cual utiliza la señal C como señal de control para seleccionar el tipo de operación a realizar. La operación (suma o resta) a transferir en el paso i–ésimo es determinada por el valor lógico de la señal C que está disponible en el paso i–1.

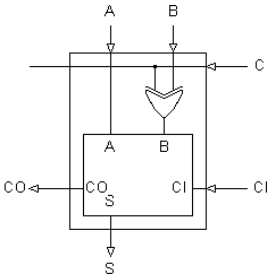


Figura 9. Celda básica CD para la división sin restauración

3.1.4 Divisor Parametrizado: Megafunción lpm_divide.

Altera provee una librería de Megafunciones conocida como Librería de Módulos Parametrizados (LPM). Para la operación de división, dispone de la Megafunción lpm_divide, la cual es un bloque parametrizado y es descrito en un lenguaje de alto nivel [5]. El diagrama de la Megafunción se puede observar en la Figura 8.

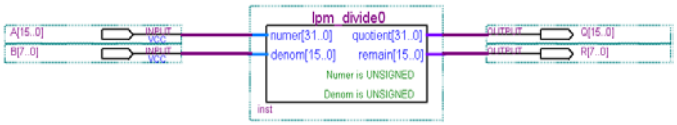


Figura 10. Divisor parametrizado: megafunción lpm_divide

3.2 Divisores secuenciales de 2N/N y N/N bits

En la Figura 11, se muestra el diagrama de bloques del *datapath* para un divisor secuencial [6]. Con este datapath es posible implementar en hardware divisores secuenciales basados en el algoritmo con y sin restauración.

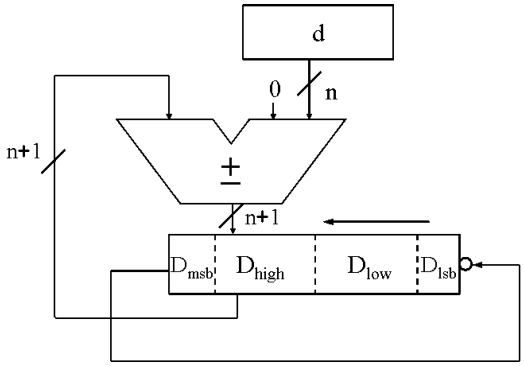


Figure 11 *Datapath* del divisor secuencial con y sin restauracion

3.2.1Divisor secuencial con restauración de 2N/N

El *controlpath* del divisor secuencial basado en el algoritmo con restauración utiliza el algoritmo presentado en la Figura 12.

Estado0 :	Cargar (0, dividendo) en *D;* cargar divisor en *d*
Estado1:	*D*esplazar *D* a la izquierda
Estado2:	*Dhigh* ← *Dhigh*-(0, *d*)
Estado3:	Si *Dmsb* = 0 entonces *Dlsb* ← 1
	Si *Dmsb* = 1 entonces *Dlsb* ← 0, *Dhigh*←*Dhigh* + (0, *d*)
	Si *Estado1-Estado3* no se han repetido *n* veces ir a

Figura 12. Descripción de la FSM para el divisor secuencial basado en la división con restauración

3.2.2 Divisor secuencial sin restauración de 2N/N

El *controlpath* del divisor secuencial basado en el algoritmo sin restauración utiliza el algoritmo presentado en la Figura 13.

Estado0:	Cargar (0, dividendo) en D, cargar divisor en d
Estado1:	Desplazar D a la izquierda
Estado2:	Dhigh ← Dhigh-(0, d)
Estado3:	Si Dmsb = 0 entonces *Dlsb* ← 1
	Si Dmsb = 1 entonces *Dlsb* ← 0
Estado4:	Desplazar D a la izquierda
Estado5:	Si Dmsb = 0 entonces Dhigh ← Dhigh-(0, d)
	Si Dmsb = 1 entonces Dhigh ← Dhigh+ (0, d)
	Si *Estado3-Estado5* no se han repetido n-1 veces ir a *Estado3*
Estado6:	Si Dmsb = 0 entonces *Dlsb* ← 1

Figura 13. Descripción de la FSM para el divisor secuencial basado en la división sin restauración

3.2.3 Divisor secuencial con restauración de N-N

El datapath del divisor secuencial basado en el algoritmo con restauración modificado de N/N bits se muestra en la Figura 14 y el algoritmo para la unidad de control es presentado en la Figura 15.

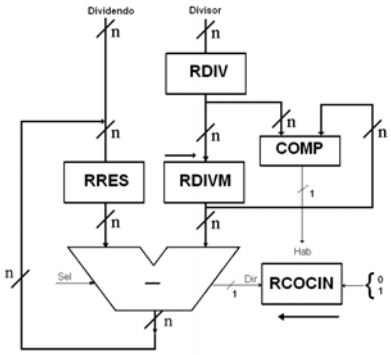


Figure 14. D*atapath* del divisor secuencial N/N

Estado0: Cargar (dividendo) en RRES, cargar divisor en RDIV para adecuación
Estado1: Cargar el divisor modificado en RDIVM para obtener el resultado de la comparación mediante el carry
Estado2: Sel ← carry, dir ← carry
Si carry = 0 entonces RRES ← RRES-RDIVM, RCOCIN←1
Si carry = 1 entonces RCOCIN←0
Estado3: RDIVM←0, cargar los datos para el comparador (COMP)
Estado4: Si hab = 1 entonces volver al Estado 0
Si hab = 0 entonces COCIENTE←RCOCIN , RESIDUO←RRES

Figura 15. Descripción de la FSM para el divisor secuencial N/N

4. Resultados de Simulación

En esta sección, se presentan los resultados de simulación para los divisores implementados en hardware para N=16 bits. En este caso, los parámetros de diseño tenidos en cuenta son: velocidad o frecuencia máxima y área o número de ALUTs (*Arimethic Look-Up Table)*. El dispositivo utilizado es el Stratix II EP2S15F484C3 de Altera y la herramienta de síntesis usada fue QUARTUS II 4.1sp2 Web Full Edition

4.1 Divisores Paralelos:

Los resultados de simulación considerando la optimización en velocidad y área, son presentados en las Tabla I y II.

Tabla I. Divisores paralelos optimizados para velocidad

Implementación	Area (# ALUTs)	Velocidad (Mhz)
Con restauración 2N-N	917	21.1
Con restauración N-N	995	11.9
Sin Restauración 2N-N	875	20.1
Altera	1171	14.7
Altera: *pipeline*	1049	55.8

Tabla II. Divisores paralelos optimizados para area

Implementación	Area (# ALUTs)	Velocidad (Mhz)
Con restauración 2N-N	431	7.4
Con restauración N-N	740	9.47
Sin Restauración 2N-N	519	16.5
Altera	847	16.1
Altera: *pipeline*	872	52.4

Desde la Tabla I, sin tener en cuenta los resultados de la Megafunción *lpm_divide* implementada con arquitectura pipeline, podemos concluir que el divisor con restauración y sin restauración con respecto a la Megafunción *lpm_divide* de Altera, son una excelente alternativa para realizar la operación de división, pues tienen una velocidad superior en casi en un 50%, además estas implementaciones presentan una menor área. También es importante notar que la megafunción de Altera con *pipeline* presenta un resultado no coherente con respecto al área utilizada, es decir que esta utiliza menor área que la megafunción sin *pipeline*. Lo anterior se justifica por que la arquitectura de los ALUTs tiene flip-flops incluidos. Si observamos la Tabla II, la implementación del divisor con restauración presenta la mejor alternativa en área ocupada, pero la implementación sin restauración es una buena opción si se necesita un poco mas de velocidad. Los resultados de simulación para los divisores paralelos son mostrados en la Figura 16.

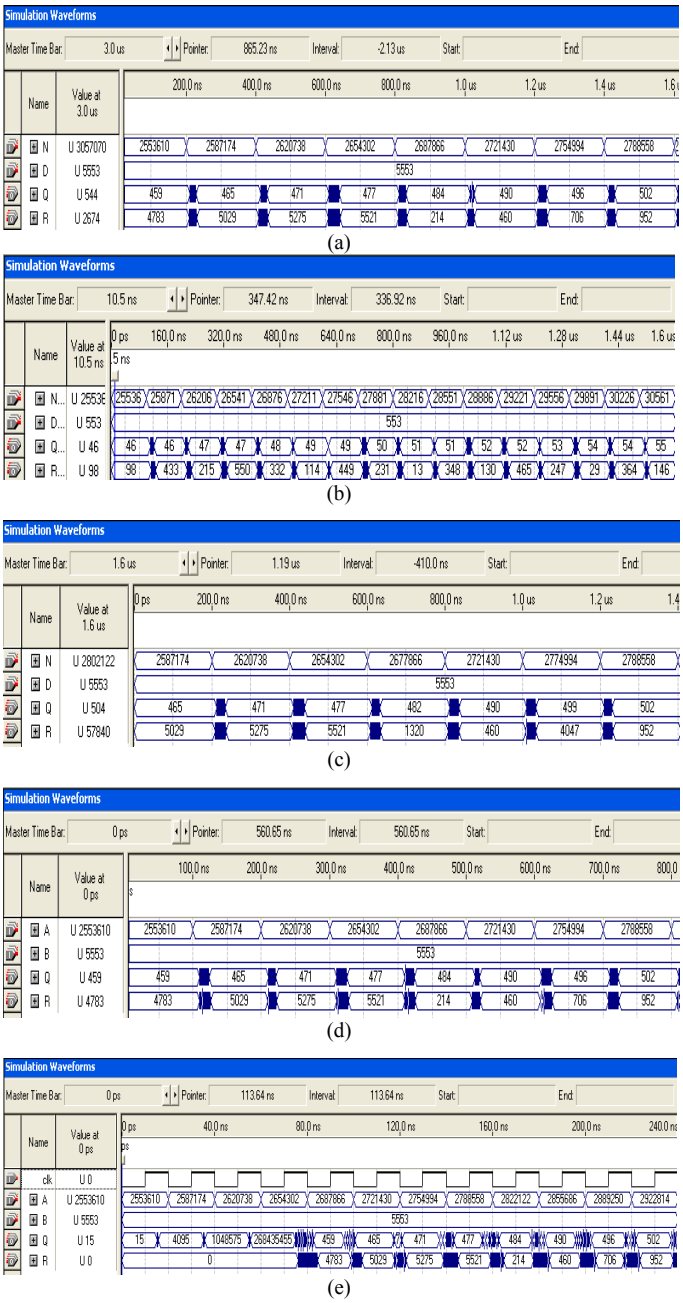


Figura 16. Resultados de simulación del divisor paralelo (a) con

restauración (b) con restauración N bits (c) sin restauración (d) Altera (e) Altera con *pipeline* de 4 etapas

4.2 Divisores Secuenciales

Los resultados de simulación muestran el resultado obtenido de la división considerando optimización en velocidad y área, son presentados en las Tabla III y IV. Para el caso del divisor con restauración de N/N-bits, donde "pos" es la posición del bit más significativo en uno (1) del divisor.

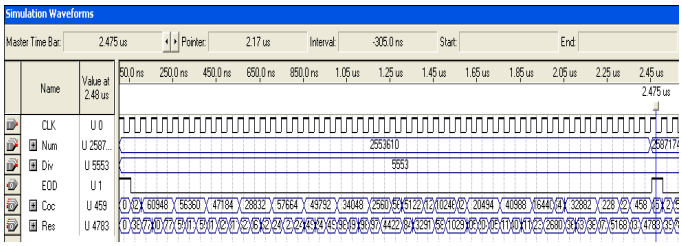
TABLA III. DIVISORES SECUENCIALES OPTIMIZADOS PARA VELOCIDAD

Algoritmo	Area (# ALUTs)	Velocidad (Mhz)
Con restauración 2N-N	120	302.5
Sin Restauración 2N-N	139	263.9
Con restauración N-N	358	313.28

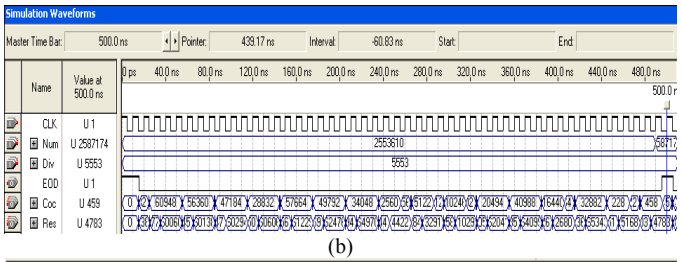
TABLA IV. DIVISORES SECUENCIALES OPTIMIZADOS PARA AREA

Algoritmo	Area (# ALUTs)	Velocidad (Mhz)
Con restauración 2N-N	93	163.4
Sin Restauración 2N-N	102	155.7
Con restauración N-N	351	310.46

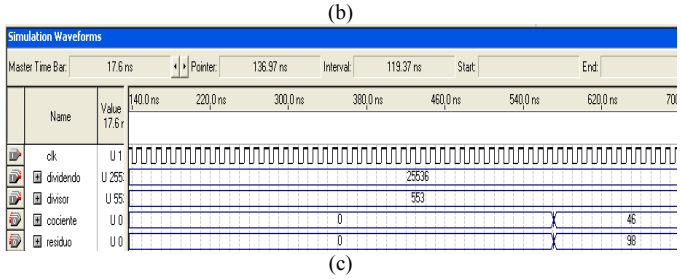
Desde la Tabla III podemos concluir que la implementación en hardware del divisor secuencial con restauración es casi 40Mhz (20%) más rápida que la implementación con el algoritmo sin restauración, con un ahorro de área del 18%. Los datos de la Tabla IV, nos muestran una reducción en la velocidad de casi el 60% de la velocidad obtenida en la optimización para velocidad, con un ahorro de área de aproximadamente el 25%, siendo la mejor solución la que utiliza el algoritmo con restauración. Los resultados de simulación son mostrados en la Figura 17.



(a)



(b)



(c)

Figura 17. Resultados de simulación del divisor secuencial (a) con restauración (b) sin restauración (c) con restauración N bits

5. Conclusiones y Trabajo Futuro

Este trabajo presenta el diseño de divisores paralelos y secuenciales de N/N bits y presenta una comparación entre las arquitecturas en hardware de divisores paralelos y secuenciales de 2N/N bits. Los divisores paralelos y secuenciales, fueron diseñados usando QUARTUS II versión 4.1sp2 y las arquitecturas fueron sintetizadas sobre el Stratix II. *EP2S15F484C3.*

En los casos donde el área y/o velocidad sean criterios de diseño primordial, el uso de la megafunción *lpm_divide* de Altera para implementar la división no es la más adecuada.

Los divisores paralelos con y sin restauración, siempre utilizan menor área, que la megafunción lpm_divide y los divisores secuenciales, presentan una ventaja con respecto a los paralelos, utilizan poca área y trabajan a mayor velocidad para la señal de reloj. Sin embargo, el resultado de la división es obtenido después de un número de ciclos de reloj.

El divisor implementado a partir de la megafunción *lpm_divide* usando arquitectura pipeline de 4 etapas, presenta el mejor desempeño desde el punto de vista de velocidad. Sin embargo, presenta la desventaja que usa una gran cantidad de área.

Los resultados de algunas arquitecturas de divisores paralelos muestran que son una opción viable para sistemas embebidos, con bajo consumo de potencia y que justifican la escasa perdida de velocidad con respecto a la megafunción *lpm_divide* con *pipeline* de 4 etapas.

El trabajo futuro, será orientado a diseñar divisores de alta velocidad considerando nuevos algoritmos y arquitecturas tales como divisores basados en el algoritmo SRT [7] y en usar pequeñas tablas de búsqueda [8].

6. AGRADECIMIENTOS

Este trabajo ha sido patrocinado por Altera Corporation a través del Programa Universitario. Los autores dan especial agradecimientos a Mrs. Ralene Marcoccia de Altera Corporation.

Referencias

[1] V.C. Hamacher, Z.G. Vranesic y S.G.Zaky, *Organización de Computadoras*, Mc Graw Hill.

[2] W. Stallings, *Organización y Arquitectura de Computadores*, Prentice Hall.

[3] M. Davio, *Digital Systems with Algorithm Implementation*, John Wiley & Sons.

[4] M. Morris Mano. *Arquitectura de Computadoras*, Prentice Hall, Prentice Hall.

[5] "lpm_divide Megafunction – User Guide". Altera Corporation. September 2004

[6] D.A. Paterson y S.L. Hennessy, *Computer Organization and Design: the Hardware / Software Interface*, Morgan Kaufmann Publishers.

[7] David L. Harris, Stuart F. Oberman. Mark A. Horowitz. "SRT Division Architectures and Implementations". 2000.

[8] Jong-Chul Jeong, Woo-Chan Park, Woong Jeong, Tack-Don Han, Moon-Key Lee, "A Cost-Effective Pipeline Divider with a Small Lookup Table". *IEEE Transactions on Computers.* Vol. 56, No. 4. April 2004.