Implementación de Circuitos *Self-Timed* de 2 y 4 Fases en FPGAs

Ortega S., Raygoza J.J., Sutter G. y Boemo E.

Escuela Politécnica Superior, Universidad Autónoma de Madrid
susana.ortega_cisneros@ii.uam.es
http://www.ii.uam.es

Resumen. Aunque los dispositivos programables tipo FPGAs están diseñados para la implementación eficiente de circuitos síncronos, en la actualidad constituyen la única opción disponible para prototipado rápido de circuitos *self-timed*. En este artículo se presentan algunas ideas para el diseño de estos circuitos en FPGAs, para dos principales protocolo: 2 y 4 fases. Como caso de estudio, se ha elegido la multiplicación binaria. Se ilustra el funcionamiento de estos circuitos y se realiza una comparación entre las dos opciones de sincronización. También se resumen los principales resultados en área, velocidad, retardo de pistas y *fanout*. Como marco tecnológico se utiliza una FPGA Xilinx Virtex II.

1 Introducción

En los circuitos síncronos la transmisión o procesamiento de datos es controlada globalmente por una o más fases de reloj. Por el contrario, en los circuitos *self-timed* (en adelante ST) no existe tal reloj global; los procesos de transmisión y procesamiento de datos se negocian localmente mediante dos líneas de protocolo: *request* (petición) y *acknowledge* (acuse de recibido). Las principales características del protocolo de 2-fases son [1], [2], [3]:

- Se produce una transferencia de datos en cada evento de las señales de *request* y *acknowledge*. La frecuencia de transmisión de datos es el doble que la transferencia de las señales de control. Lo contrario que en un sistema síncrono donde el reloj tiene como mínimo una frecuencia doble que los datos.
- Los eventos en *request* y *acknowledge* se producen alternadamente. Aunque la información esta contenida en los flancos, los niveles de estas señales mantienen siempre la misma relación. Por esta razón es necesario partir del mismo nivel inicial, el cual debe fijarse mediante una señal de *reset*. [4], [5], [6].

El protocolo de 4-fases, también llamado “protocolo de señalización por nivel”[7],[8], utiliza el nivel de las señales para indicar la validación de los datos y su aceptación por el receptor. Cuando el dato esta disponible a ser enviado, el emisor produce un cambio de nivel en la señal de *request*. El receptor contesta por medio de un cambio de nivel (‘0’ a ‘1’) de la señal de *acknowledge*. Como consecuencia, el emisor baja la señal de *request*, la cual es reconocida por el receptor, que también baja la señal de *acknowledge*. Para que se

generen correctamente las señales, es necesaria una fase de retorno a cero que restaure el estado que tenían las señales de control antes de una transferencia. Este esquema utiliza el doble de señalización que su contraparte de 2 fases [9], [10], [11]. Los lectores interesados en la sincronización ST pueden consultar dos artículos de divulgación sobre este tema en la sección "Tutorials sobre Temas de FPGAs" de este libro [12] y [13].

2 Multiplicación ST

El multiplicador elegido como circuito de referencia fue propuesto por H. Guild en su célebre artículo *Full Iterative Fast Array for Binary Multiplication and Addition.* [14]. Para su adaptación a una sincronización ST de 2 y 4 fases, cada etapa del *pipeline* requiere un bloque de comunicación asíncrono (BCA) para negociar la transferencia de datos entre líneas de los procesadores elementales [15], [16]. A nivel de bloque, el multiplicador opera sobre datos que envía un emisor asíncrono y envía los resultados a un receptor asíncrono.

En la figura 1a y b, se muestra el diagrama esquemático de la implementación del multiplicador utilizando los protocolos de 2 y 4 fases respectivamente.

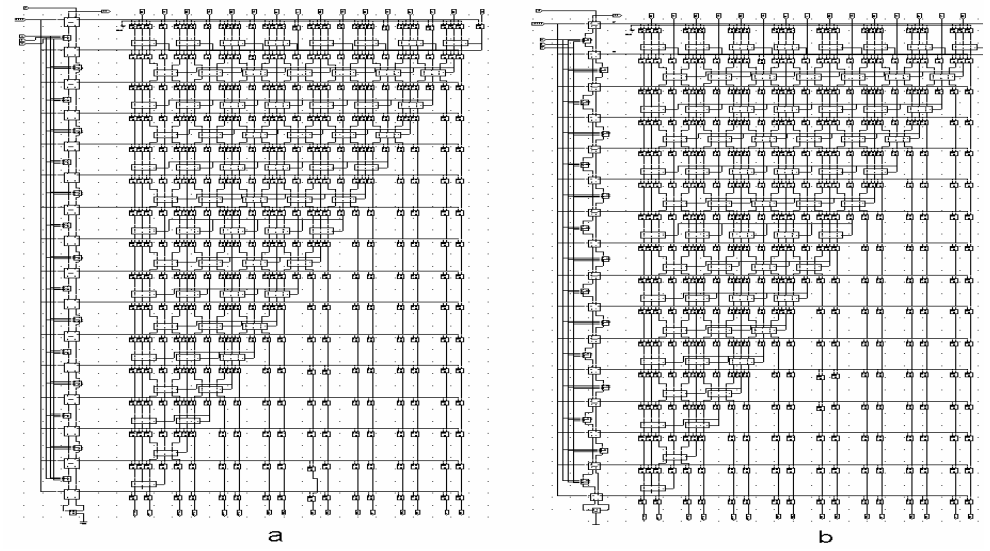

Fig. 1 Multiplicador de Guild con protocolo ST: 2 fases (a) y 4 fases (b).

Los *latches* de cada etapa del *pipeline* son controlados por un BCA. Tanto para el circuito de 2 como el de 4 fases, el multiplicador Guild se transforma en un *pipeline* de 15 etapas y 16 líneas de *latches* (una más para mantener la E/S registrada). Para que la sincronización se realice correctamente el retardo entre el *ack out* y el *req in* de la siguiente etapa [17] debe tener un valor igual al del procesador elemental del multiplicador.

En la figura 2 se ilustra el funcionamiento del circuito para 2 fases. Las señales xi001(0), xi001(1), xi001(2)....xi001(15) muestran los instantes de activación de los *latches*. El resultado correcto (bus "res") de la multiplicación se valida en cada evento del *request* y *ack-*

nowledge (“ri” y “as”). Un *testbench* en VHDL [18] se encarga de introducir de manera asíncrona los datos de entrada (“dato_a” y “dato_b”).

Para el control *self-timed* de 4 fases [19] se presentan los datos de entrada (“a” y “b”) solo en los flancos de subida. El flanco de bajada de la señales de control (“ri” y “a_s”) se utiliza para el restablecimiento del circuito.

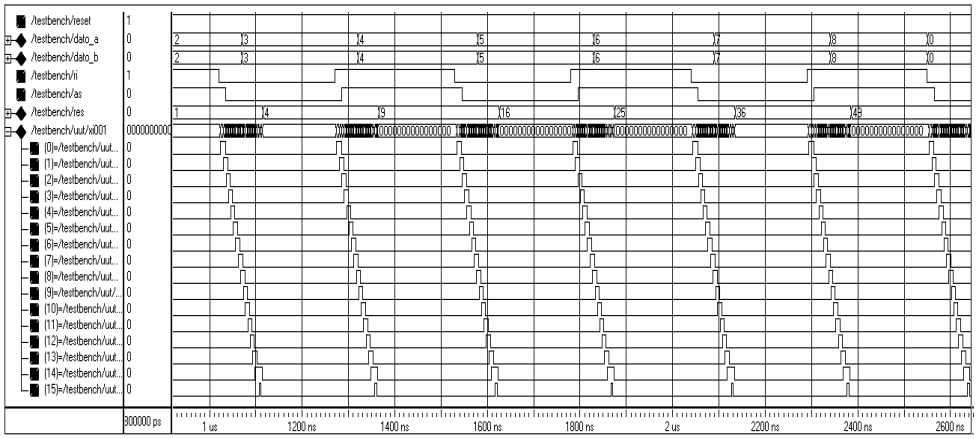

Fig. 2 Simulación *post-layout* del prototipo de 2 fases. Las señal X001 a la X0015,muestran el control de transferencia nivel a nivel del multiplicador.

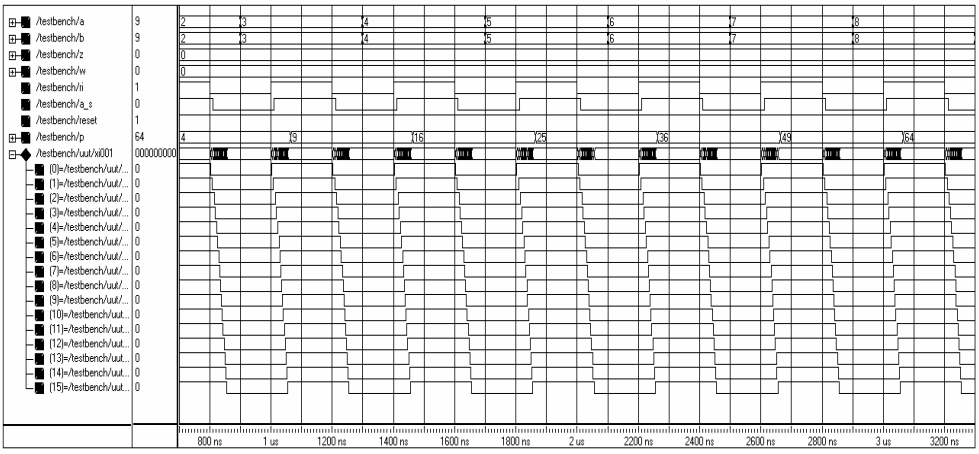

Fig. 3 Simulación lógica del multiplicador de 4 fases. Las señales de acknowledge (“a_s”) y rerequest (“ri”) se presentan sincronizadamente.

La multiplicación se realiza después de que la señal de *reset* es activada. Las señales de control de los *latches* (representadas por el *bus* xi001) se muestran en la figura 3. El pro-

ducto de la multiplicación (bus “p”) se valida en los flancos de subida de las señales “ri” y “a_s”.

Los protocolos de señalización 2 y 4 fases fueron sometidos a pruebas utilizando un *test-bench* que aplica diferentes frecuencias para la señal de activación de “ri” y los datos de entrada “a” y “b”. El multiplicador de 2 fases opera correctamente hasta la frecuencia de 14 MHz y el correspondiente al protocolo de 4 fases hasta 19 MHz, operando adecuadamente con frecuencias abajo de esta misma. Por lo que el multiplicador de 4 fases opera a mas alta velocidad en comparación con su contraparte de 2 fases.

3 Resultados en FPGAs

Los circuitos ST se mapearon sobre una FPGA Xilinx XC2V1000-4FG256 [20]. Los circuitos se definieron en VHDL y se compilaron con la herramienta ISE 5.1. En todos los casos, se utilizaron la opciones por defecto.

En el protocolo de 2 fases, un BCA (bloque de control asíncrono) el elemento básico de un pipeline ST ocupa 1 CLB (4 *slices*, 2 registers y 3 LUTs). En la figura 4 se muestra el layout de un BCA y en la figura 5 el detalle de ocupación. En el ejemplo, el *slice* comp "b1__n0002" (ubicación "*SLICE*_X11Y44”) utiliza la salida F de la LUT para mapear la función lógica de la celula Muller-C:

D=((~A2*((~A1*A3)+~A4))+(A2*(~A1+(A3+~A4)))).

La Muller-C del ejemplo tiene tres entradas: *reset* (“*reset_ibuf”),* la entrada externa (b3_u2_q”) y la entrada de realimentación (“b1_n0002”). El *slice* "b3_u1_q" ("*SLICE*_X18Y54"), mapea dos componentes: un inversor <BYINV>, y un flip-flop <FFY_INIT_ATTR>, que forman la primera parte del componente *toggle*. El *slice* "b3_u2_q" ("*SLICE*_X18Y55") contiene el segundo flip-flop y mapea la segunda parte del *toggle*. El *slice* "I5" ("*SLICE*_X10Y44") contiene la unión de los componentes *toggle* y Muller-C.

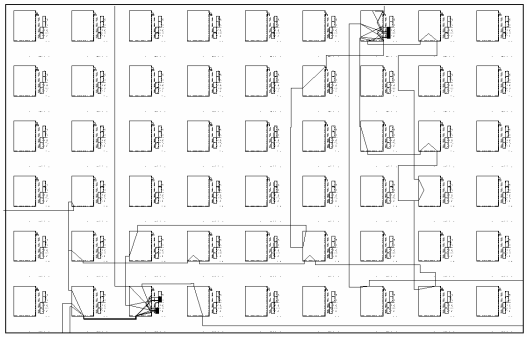

Fig. 4. BCA de 2 fases: *Layout* en Virtex II.

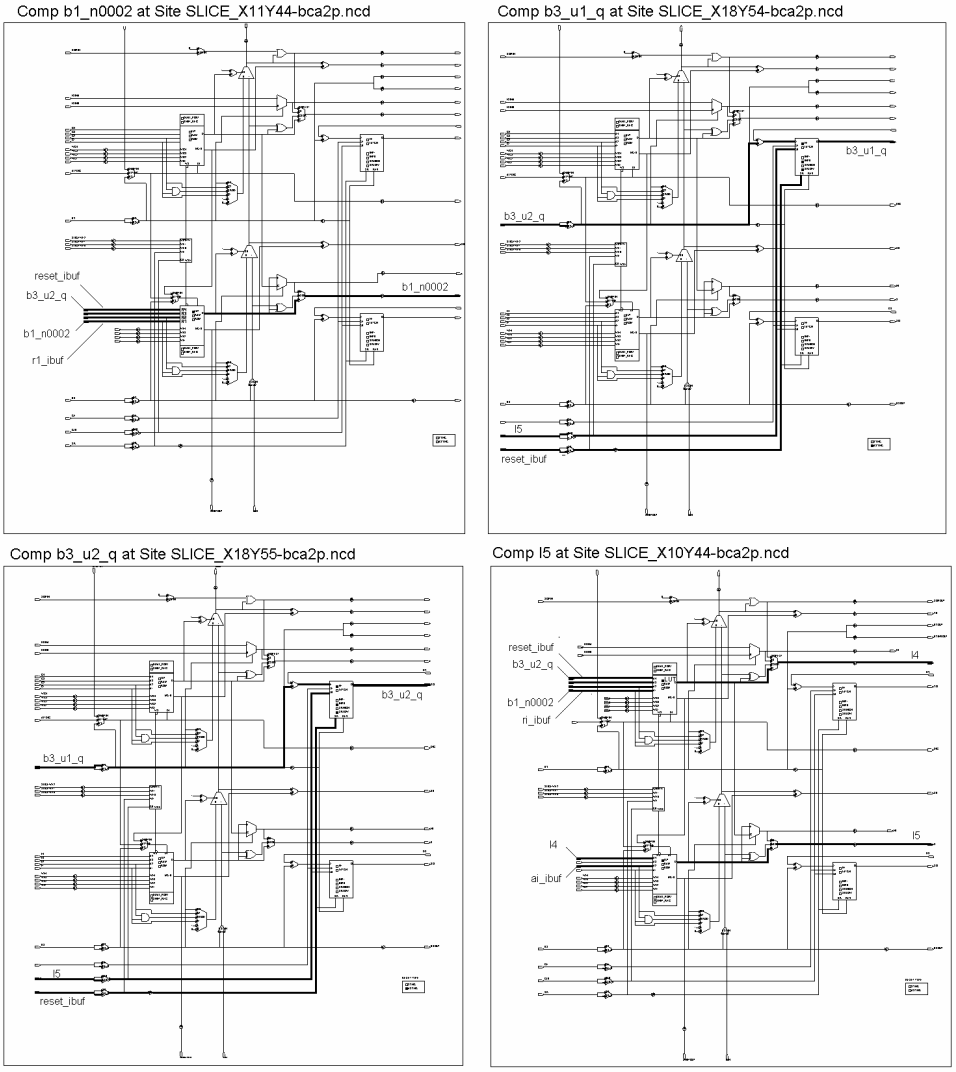

Fig. 5 BCA de 2 fases en VIRTEX-II.

En el protocolo de 4 fases, un BCA ocupa menos elementos de 1 CLB: 1 *slice* y 2 LUTs. La figura 6a muestra su *layout*, y en la figura 6b, la ocupación de elementos de la FPGA (*slice* comp "a_s_obuf", posicionado en "*SLICE*_X0Y6"). En este caso, las 2 LUTs se usan para mapear los FF “RS” más la lógica auxiliar del BCA. Las funciones son:

D=(A2*((~A1*(A4*~A3))+(A1*(A4+~A3)))), para la LUT superior.
D=(A3*((~A1*(A2*~A4))+(A1*(A2+~A4)))), para la LUT inferior.

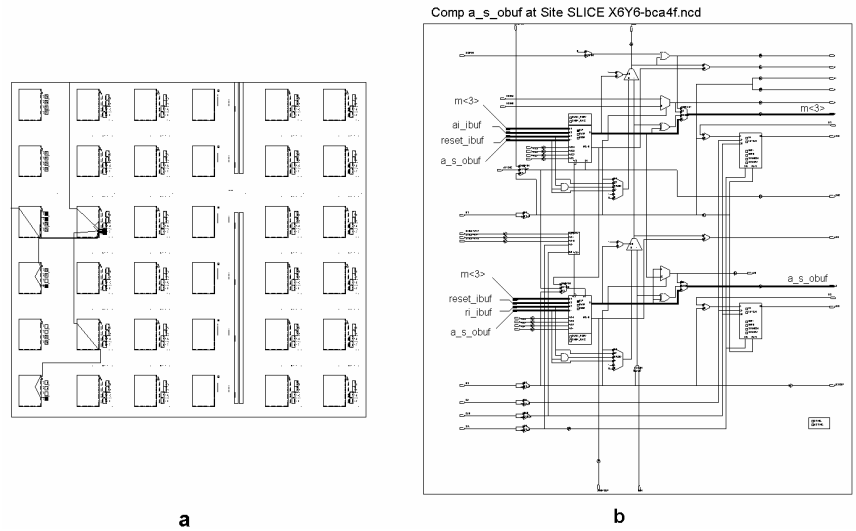

Fig. 6 BCA de 4 fases en VIRTEX-II.

Para realizar el retardo se utilizo la técnica llamada compensación de retrasos "*matched delays*" [21]. Para cada etapa del pipeline, se implementa un elemento de retardo cuyo valor es similar al de la etapa. Así, cuando la señal de *request* (entrada al elemento de retraso) se activa, los datos en la entrada del bloque de computación son válidos. En este caso dicho bloque requiera un retardo de 9.3ns. Para generar este retardo se diseño un macro con FPGA editor (Figura 7) utilizando una LUT y la ecuación: D=A2.

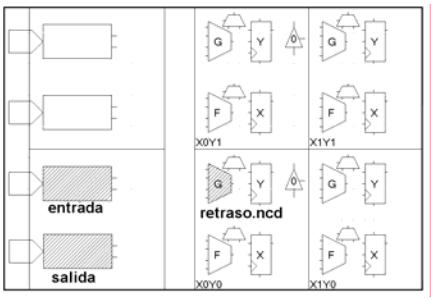

Fig. 7 Macro del retardo en VIRTEX-II.

En la figura 8 se muestran los resultados de ocupación para tres circuitos multiplicadores Guild de 8 bits y granularidad fina: 2 fases, 4 fases y finalmente sincronización clásica de fase única (TSPC o *true single-phase clocking*). El circuito de 2 fases ocupa 376 de los 5,120 slices (7%) mientras que el de 4 fases ocupa 326 slices (6%). Finalmente, el circuito síncrono sólo ocupa 280 slices (5%). En la figura 8b se muestra la cantidad de *latches* o FF utilizados en cada caso. En la figura 8c, la cantidad de *LUTs*. [22]

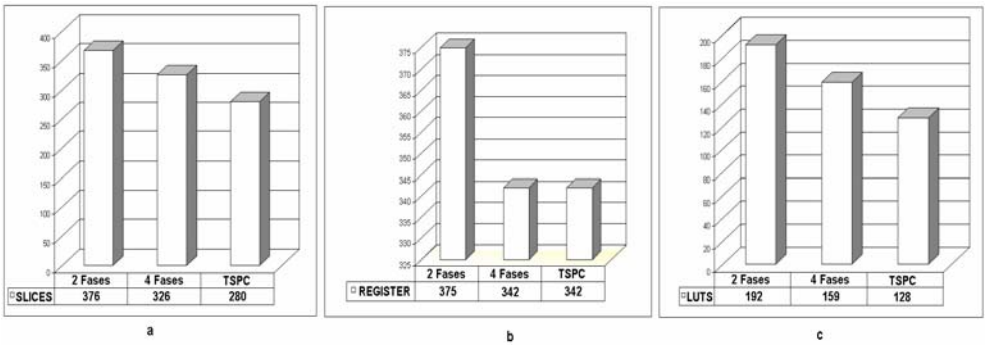

Fig. 8 Ocupación del multiplicador de granularidad fina en el FPGA.

La segmentación de grano fino divide a las líneas globales de datos de entrada en un subconjunto de líneas con menor carga. Como consecuencia, la frecuencia de operación se incrementa por un doble efecto: reducción de la profundidad de lógica y reducción en la capacidad en cada nodo. [15]

En la Tabla 1 se resumen los resultados de ocupación de recursos de los mismos circuitos pero segmentando cada dos etapas (granularidad 2). En los tres multiplicadores se obtiene una reducción en área de un 50% aproximadamente. En la figura 9 se muestran los resultados de la implementación para los multiplicadores de 2 fases (figura 9a) y 4 fases (figura 9b).

Protocolo	SLICEs	REGISTERs	LUTs
2 Fases	225	187	164
4 Fases	196	168	145
TSPC	162	168	128

Tabla 1. Ocupación de recursos de los multiplicadores de granularidad 2.

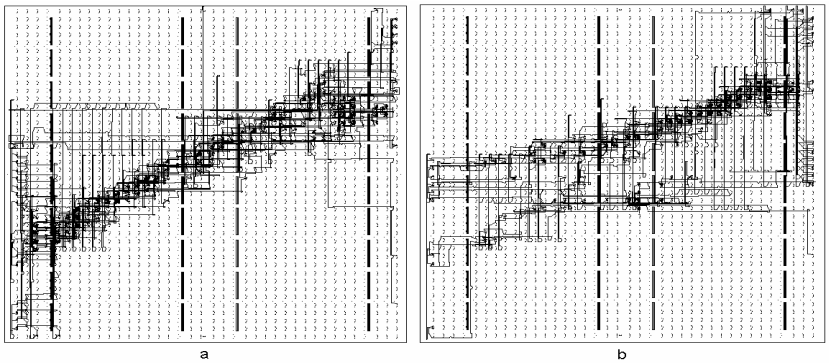

Fig. 9 Vista de rutado de los multiplicadores ST con granularidad 2 en el FPGA

4 Predicción del retardo de pistas en un prototipo ST en FPGA

Uno de los problemas en la construcción de circuitos ST en tecnologías con un alto grado de automatización del proceso de *placement* y *routing* es la dificultad de conocer anticipadamente los retardos de interconexión. Estos datos son fundamentales para dimensionar el retardo asíncrono que se debe asignar al *request* de cada etapa del *pipeline*.

En las figuras 10,11 y 12 se muestran los retardos en función del *fanout* de cada nodo, para los multiplicadores de 2 y 4 fases (granularidad 2). También se presentan los histogramas de retardo pistas. Se puede observar que éstos últimos son muy similares para 2 y 4 fases, salvo para la señal de *reset* que tiene el *fanout* más elevado, el retardo de pista puede considerarse menor que 3 ns. En general, más del 80 % de las pistas tiene un retardo menor que 1 ns, siguiendo una distribución de Pareto-Levy [23].

En la gráfica de la figura 11 se muestra que la mayoría 423 (98%) de las pistas tienen un *fanout* de que oscila en el rango [0, 34], con retardos de entre 0 y 7 ns. Con respecto a la gráfica de la figura 12 se muestra que 391 (98%) de las pistas oscilan entre 0 a 34, con retados de entre 0 a 3 ns.

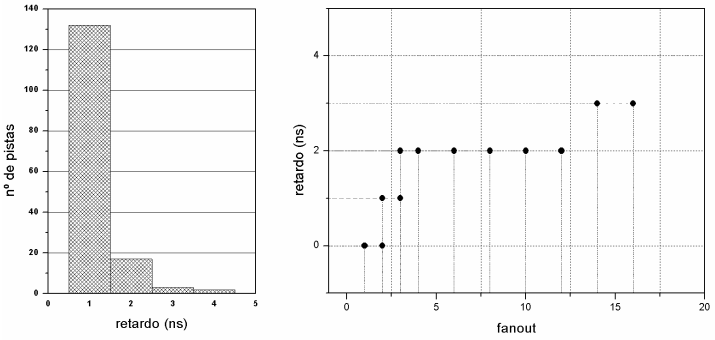

Fig. 10 Retardo vs fanout multiplicador 8x8 TSPC grano2 FPGA.

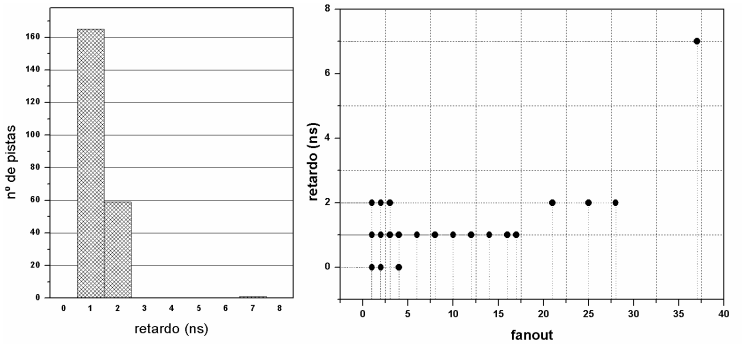

Figura 11 Retardo vs fanout multiplicador 8x8 2 fases grano2 FPGA.

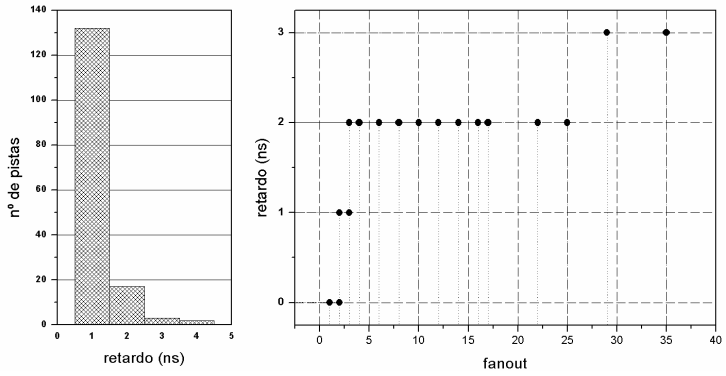

Figura 12 Retardo vs fanout multiplicador 8x8 4 fases grano2 FPGA.

5 Conclusiones

En este trabajo se han dado algunas ideas y resultados numéricos útiles para el prototipado rápido de circuitos *self-timed* de 2 y 4 fases en FPGAs. La velocidad de operación del protocolo de 4 fases es mayor que la del protocolo de 2 fases, observándose más robustez de esta técnica. Finalmente, en los circuitos presentados la ocupación de los circuitos *self-timed* es 30% mayor que los *pipelines* síncronos equivalentes.

Bibliografia

[1] A. Peeters and K. van Berkel, "Single-rail handshake circuits", in *Asynchronous Design Methodologies*, pp. 53–62, IEEE Computer Society Press, May 1995.
[2] Arechavala, L. "Design *Self-Timed*", thesis University of Manchester. Department of-Computer Science UK, 1994.
[3] I.E. Sutherland, "Micropipelines", *Communications of the ACM*, Vol. 32 No. 6, June 1989, pp. 720-738.
[4] Al Davis,Steven M. "An Introduction to Asynchonous Circuit design", UUCS-97-013, Computer Science University of Utah. September 19, 1997.
[5] O.A. Petlin and S.B. Furber, "Built-in self-testing of micropipelines", in *Proc. InternationalSymposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 22–29, IEEEComputer Society Press, Apr. 1997.
[6] Kees van Berkel and Arjan Bink. "Single-track handshaking signaling with application to micropipelines and handshake circuits", In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 122-133. IEEE Computer Society Press, March 1996.
[7] H. Kagotani, T. Okamoto, and T. Nanya, "Synthesis of four-phase asynchronous control circuits from pipeline dependency graphs", in *Proc. of Asia and South Pacific Design Automation Conference*, pp. 425–430, Feb. 2001.

[8] S.B. Furber and J. Liu, “Dynamic logic in four-phase micropipelines”, in *Proc. InternationalSymposium on Advanced Research in Asynchronous Circuits and Systems*, IEEE Computer Society Press, Mar. 1996.
[9] M. Renaudin, B.E. Hassan, and A. Guyot, “New asynchronous pipeline scheme: Applicationto the design of a self-timed ring divider”, *IEEE Journal of Solid-State Circuits*, vol. 31,pp. 1001–1013, July 1996.
[10] Liu, J., “Arithmetic and Control Components for an Asynchronous System”, Ph.D. thesis, Department of Computer Science, University of Manchester, UK (1997)
[11] V.W.-Y. Sit, C.-S. Choy, and C.-F. Chan, “A four-phase handshaking asynchronous static RAM design for self-timed systems”, *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 90–96, Jan. 1999.
[12] E. Boemo y Ortega S, “Protocolo Self-Timed de 2 fases: un Tutorial”, *Proc. JCRA 2003*, Universidad Autónoma de Madrid: 2003.
[13] Ortega S.,Raygoza J.J. y E. Boemo, “Protocolo Self-Timed de 4 fases: un Tutorial”, *Proc. JCRA 2003*, Universidad Autónoma de Madrid: 2003.
[14] H. Guild “Full Iterative Fast Array for Binary Multiplication and Addition”, *Electronics Letters*, pp-263, Vol. 5, Nº 12, Jun. 1969.
[15] Boemo E. “Contribución al diseño de *arrays* VLSI con paralelismo de grano fino”, Ph.D. Tesis. Universidad Politécnica de Madrid, Noviembre 1995.
[16] X. Liao and J.-S. Chiang, “A novel asynchronous control unit and the application to a pipelined multiplier”, in *Proc. International Symposium on Circuits and Systems*, 1998.
[17] Kelly R. “Asynchronous Design Aspects of High-Performance Logic”, Thesis. University of Manchester. *Department of Computer Science* UK, 1995
[18] Xilinx. “Concept-HDL Interface Guide,” 2.1i. Copyright 1991-1999 Xilinx, Inc.
[19] Sun-Yen-Tan. “High-Level Modelling of micropipelines”, MSc.Thesis University of Manchester. *Department of Computer Science* UK, 1992
[20] Xilinx “Virtex-II Platform FPGA User Guide”, 1-800-255-7778 UG002 (v1.5) 2 Dec. 2002. www.xilinx.com
[21] A. Acosta, A. Barriga, M. Bellido y J. Valencia, *“Temporización en circuitos integrados CMO”* , Editorial Marcombo, Cap. 3 pp. 167-169.
[22] Virtex™-II “Platform FPGAs:Introduction and Overview”, DS031-1 (v1.9) September 26, 2002 Advance Product Specification
[23] E. Boemo, S. Lopez-Buedo, E. Todorovich, and N. Acosta, “Logic Depth and the Determinism of Place-Route Tools. Some Experiments on FPGAs”, *Proc. VI Iberchip Workshop*, pp.280-285, Sao Paulo, Brazil, 16-18 March, 2000.