Sincronización *Self-Timed*: Protocolo de 4 Fases

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

Escuela Politécnica UAM. Universidad Autonoma de Madrid.
susana.ortega_cisneros@ii.uam.es.
http://www.ii.uam.es

Resumen. En este tutorial se presentan las principales características de la señalización de 4 fases. Se detallan todos los bloques de control necesarios para diseñar un *pipeline*, tales como los módulos *call*, *arbitrer*, *select* y *toggle*. Como marco tecnológico se utilizan FPGAS Xilinx.

1 Introducción

Los circuitos *self-timed* (ST) se basan en dos protocolos de sincronización: 2 y 4 fases. Cada protocolo presenta una determinada figura de área, velocidad, potencia, robustez etc. Este artículo se centra en el protocolo de 4 fases. En este esquema se utilizan básicamente elementos *flip-flops* y células Muller-C [1] para generar señales locales de reloj. El término 4 fases se refiere al número de acciones que involucra la comunicación:

1.-El emisor pone un dato y activa la señal de petición en alto.
2.-El receptor recibe el dato y activa la señal de conocimiento en alto.
3.-El emisor responde poniendo la señal de petición en bajo.
4.-El receptor reconoce esto poniendo la señal de conocimiento en bajo.

Una vez finalizada la última fase, el emisor puede iniciar la siguiente transferencia de datos. Este protocolo es el más utilizado en la actualidad debido a sus ventajas en lo referente a consumo de potencia, emisión de ruido electromagnético y robustez respecto a variaciones en tensión de alimentación y temperatura [2].

2 Señalización de 4 fases

El *handshake* de 4-fases es también conocido como "protocolo de señalización por nivel". 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 las señal de *request*. El receptor reconoce por medio de un cambio de nivel ('0' a '1') de la señal de *acknowledge*. El emisor baja la señal de *request*, la cual es reconocida por el receptor, bajando también la señal de *acknowledge*. Para que se generen correctamente las señales, es necesaria una fase de retorno a cero para que inicialicen al estado que tenían antes de una transferencia. Este esquema utiliza el doble de señalización por evento

que su contraparte de 2 fases. El esquema de transmisión se muestra en la figura 1. El dato puede estar listo para ser transferido del emisor al receptor cuando el primero envía una señal de *request* "Rin". El receptor la recibe, respondiendo con una señal de *acknowledge* "Aout". Sin embargo, la transferencia aún no ha finalizado, el emisor todavía debe regresar la señal de *request* a un nivel inactivo y de igual forma, el receptor deberá hacer el mismo procedimiento con la señal de *ackwnoledge.* Esta acción, que podría considerarse redundante, tiene por función regresar al estado inicial a las señales de *request* y *ackwnoledge*. En el protocolos de 4 fases se utilizan *latches* y *flip flops (FF)* como elementos de almacenamiento en la ruta de datos [3],[4],[5].

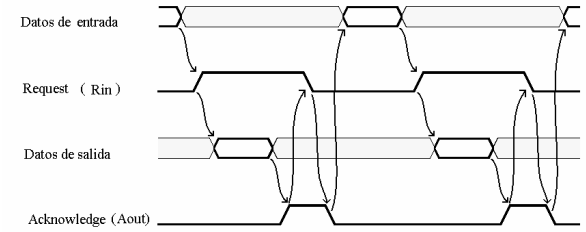

Fig. 1: Protocolo de 4 fases.

En el diseño de los llamados *micropipeline* de 4 fases se manejan tres tipos de esquemas de validación de datos: tempranos, anchos y tardíos.

El diseño del *micropipeline* de 4 fases usa dos sucesivos *handshakes* para completar el proceso de comunicación entre dos etapas vecinas del *pipeline*. Este permite seleccionar el modo de validar los datos: por el flanco de subida o de bajada. En la figura 2 se observan los tres tipos de validación de datos.

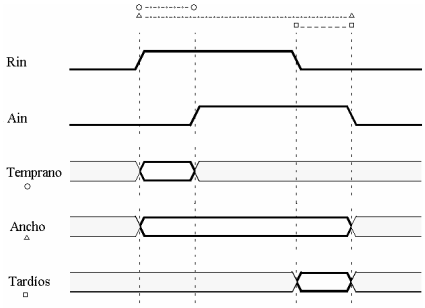

Fig. 2: Esquema de validación de datos de 4 fases

Inicialmente "Rin" y "Ain" están en nivel bajo. En la validación de datos tempranos se utiliza el flanco de subida de "Rin", el cual indica que el dato esta disponible. En el flanco de subida de "Ain" se indica que el dato ha sido capturado. Entonces "Rin" es retornado a cero y posteriormente "Ain" también regresa a cero. El primer *handshake,* en donde "Rin" y "Ain" están en alto, se le denomina fase de procesamiento o evaluación. En él, los datos son validados pudiendo ser cambiados posteriormente. El segundo *handshake* se da cuando

"Rin" y "Ain" están en bajo. Es la llamada fase de *reset*, la cual es redundante para la transferencia de datos, pero necesario para reinicializar al control.

El esquema de datos anchos utiliza flanco de subida de "Rin", el cual indica que el dato esta disponible y el flanco de bajada de "Ain" que se reconoce como "dato capturado". Durante este ciclo, el dato deberá estar presente todo el tiempo para su correcto funcionamiento.

El esquema de datos tardíos usa el flanco de subida de "Rin" que indica "dato disponible" y el flanco de bajada de "Ain" para notar "dato capturado". Al primer *handshake* en donde "Rin" y "Ain" están en alto es la fase de "*preset*", la cual es redundante para transmisión de datos. El segundo *handshake,* con "Rin" y "Ain" en bajo, es llamado fase de procesamiento o evaluación, pues en él los datos son validados. Este último esquema no es muy utilizado [6].

Para describir circuitos ST de 4 fases se utilizan los Grafos de Transición de Señal (STG). Los STG son por definición, una subclase de red de Petri del tipo "libre elección" (*free choice*) interpretada. Básicamente muestran las relaciones de transiciones mediante un diagrama de flujo. Cada transición está conectada con una serie de lugares predecesores y sucesores, que indican respectivamente las condiciones para que se dé la transición y las consecuencias de las mismas. Por lo tanto, un STG muestra la relación de causalidad entre las señales de transición.

En el STG los cambios de señal son representados agregando un signo "+" para representar un cambio de nivel con flanco de subida. Se agrega un signo "—" para indicar un cambio de flanco de bajada. Las flechas indican el sentido de la dependencia con el predecesor y sucesor. Los círculos pequeños sobre las líneas indican el estado actual. En la figura 3 se muestra la descripción STG del circuito de control del protocolo de 4 fases. [7],[8]

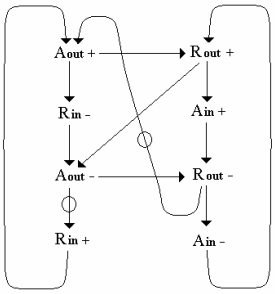

Fig. 3: Grafos de Transición de Señal (STG) en estado $A_{out}\ R_{in}\ A_{in}\ R_{out}$ = 0 0 0 0, para el protocolo de 4 fases

3 Implementación del Bloque de Control Asíncrono (BCA) de 4 fases

En el protocolo de 4 fases sólo se produce una transferencia de datos en los flancos de subida del *request*. Para su realización, se utilizan elementos *flip-flops* tipo RS, componentes Muller C y algunas puertas lógicas adicionales. En la figura 4 se muestra un esquema del FF-RS con *reset*. Tanto R como S son activas a nivel bajo [9].

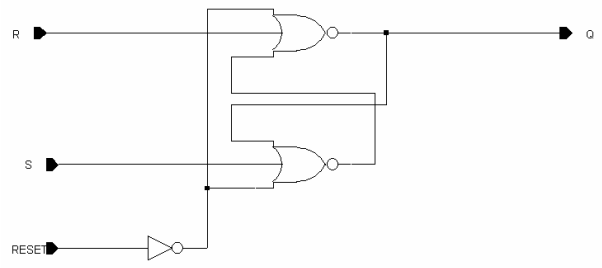

Fig. 4: Diagrama esquemático del *flip-flop* RS.

Como se describe en la simulación de la figura 5, en el primer instante todas las señales están en cero, condición de inicialización. A continuación se activa la señal de "*RESET"*, preparando al circuito para su funcionamiento. Cuando se presenta un cambio de nivel en la señal de *set* "S", la salida del *flip-flop* se activa poniendo a "Q" en alto. Al siguiente cambio de la señal "S" ('1' → '0'), la salida no presenta cambios, permaneciendo con el estado anterior. Al cambiar la señal de "R" ('0'→'1'), se presenta un movimiento en la salida ('1'→'0') pasando a condiciones de reposo del *flip-flop*. Aunque se presente un nuevo cambio en la señal "R", la salida permanece con el estado anterior.

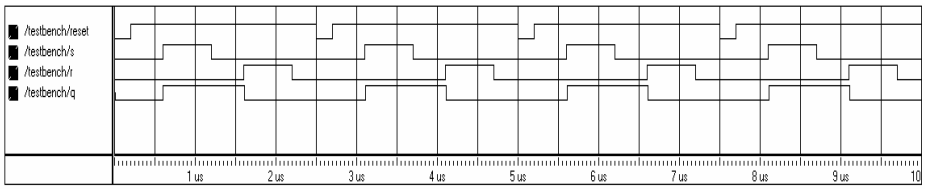

Fig. 5: Simulación lógica del flip-flop RS

Para construir un BCA de 4 fases, se requieren dos componentes *flip-flop* RS y algunas puertas lógicas AND e inversores. Este BCA tiene 3 entradas ("R_I","A_O" y "*RESET*") y 3 salidas ("R_O","A_I" y "X"). El esquema se muestra en la figura 6. [10]

La entrada "R_I" y la salida "A_I" serán conectadas al bloque "I" anterior (no dibujado) y las señales "R_O" y "A_O" al bloque BCA posterior "O" (no dibujado). La señal "XI" es conectada a la señal de captura del *latch*, la cual controlará el paso de los datos. La señal de "*RESET*" pone a '0' todas las salidas en un tiempo inicial, como condición obligatoria de una transmisión correcta.

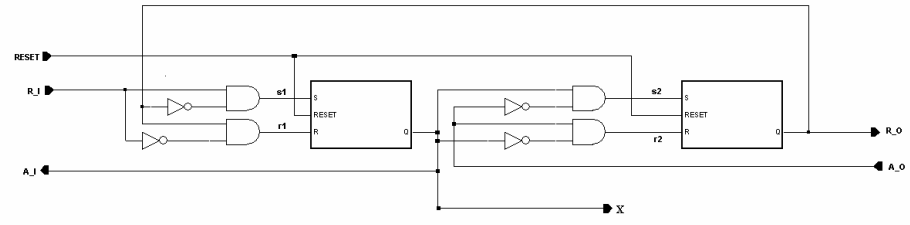

Fig. 6: Diagrama esquemático del BCA de 4 fases, utilizando *filp-flops* RS.

Otro de los elementos que se utilizan en la construcción de BCA de 4 fases es la Muller-C (figura 7). Este bloque se comporta análogamente a una puerta AND pero para eventos. Es decir: a su salida se produce un evento sólo si en todas sus entradas se han producido eventos.

Desde el punto de vista práctico, el comportamiento de una Muller-C se puede resumir de dos maneras equivalentes (figura 8) [11]:

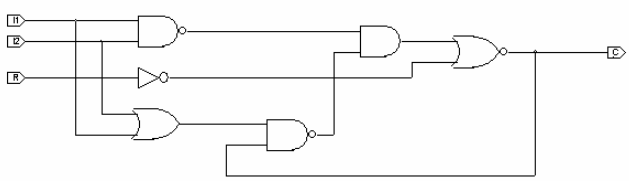

Fig. 7: Diagrama esquemático de la puerta Muller-C

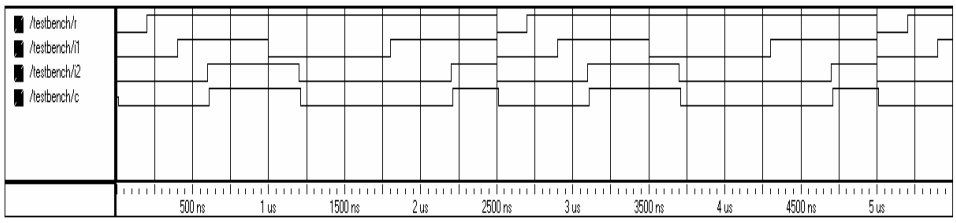

Fig. 8: Simulación lógica de la puerta Muller-C.

Un circuito equivalente para la implementación de una BCA de 4 fases se muestra en la figura 9, este utiliza elementos Muller-C y sigue la secuencia de $R\_in^{+} \rightarrow A\_out^{+} \rightarrow R\_in^{-} \rightarrow A\_out^{-}$, para cada transferencia de datos (figura 10).[12]

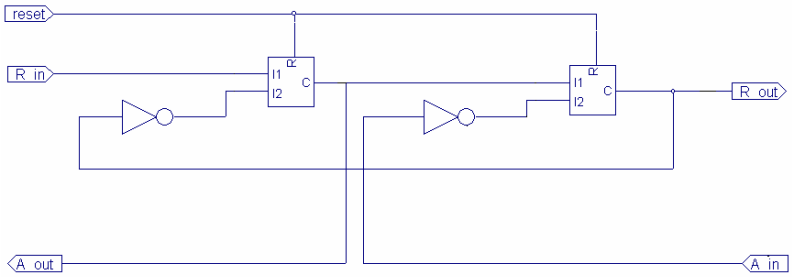

Fig. 9: Diagrama esquemático del BCA de 4 fases, utilizando elementos Muller-C

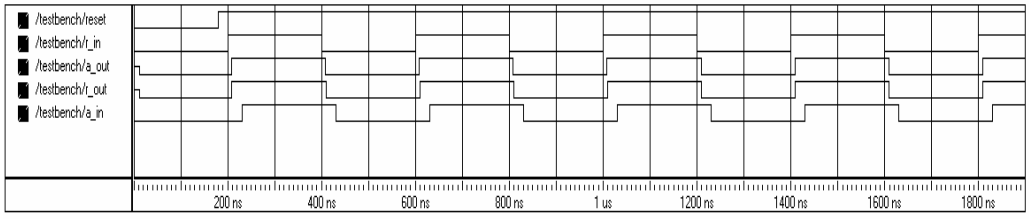

Fig. 10: Simulación lógica del BCA de 4 fases

4 Comportamiento del bloque de control asíncrono de 4 fases

Para ilustrar el mecanismo de sincronización del circuito es conveniente focalizar la atención sobre una etapa genérica *pipeline* (figura 11) cuyo comportamiento es idéntico al resto de los bloques. El análisis se descompone en una sucesión de eventos, resumiéndose a continuación y se representan gráficamente en la figura 12. Donde además se incluyen algunos ciclos adicionales para evidenciar el carácter periódico del funcionamiento del circuito.

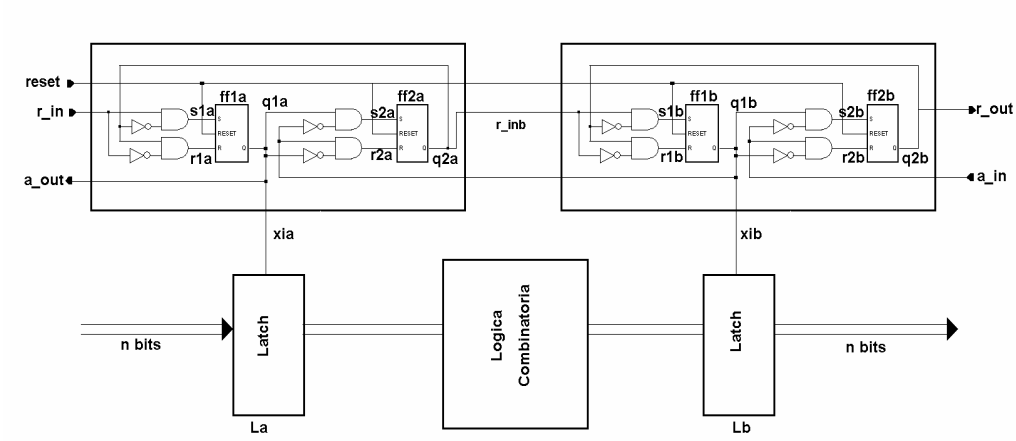

Fig. 11: Etapa genérica de un circuito segmentado ST de 4 fases

Descripción del esquema del protocolo de 4 fases:

1. Se realiza un *reset* inicial de las señales:'r_in','a_out','r1a','r1b','r2b','s1a', 's2a',**'**s1b','s2b','r2a'.
2. La etapa anterior a-1 (no dibujada) impone un dato en la entrada del *latch* 'L_a' y luego produce un evento en 'r_in' (0→1).
3. El cambio de nivel 'r_in' produce un cambio de 0 →1 en la señal de 's1a'.
4. La activación de la señal 's1a' provoca un cambio de nivel en el elemento 'ff1a' (salida 'q1a' en 1) y en la señal 'xia'. (0→1), el *latch* 'L_a' cierra y

captura el dato. Además ocasiona un cambio (0→1) en la señal de reconocimiento "a_out", que va al bloque a-1.

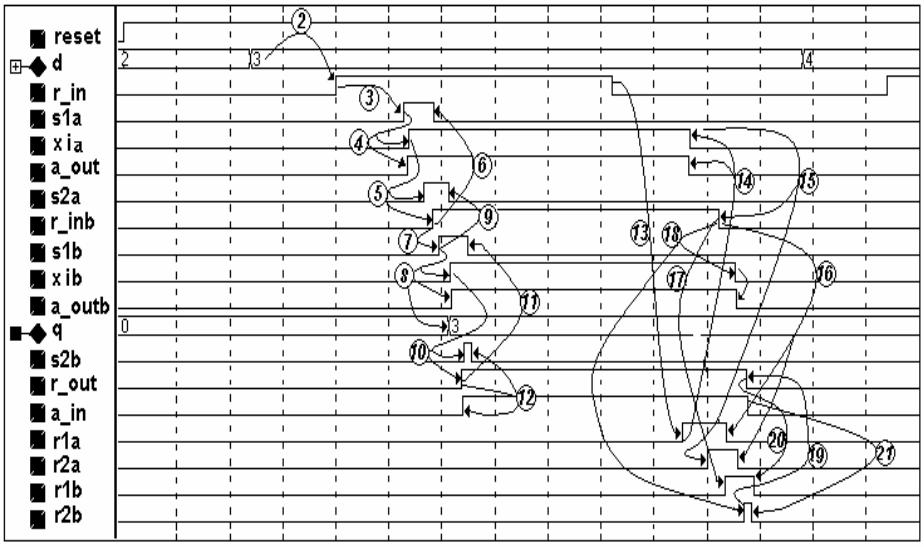

Fig. 12: Diagrama de tiempos de una etapa genérica de un *pipeline* ST. Protocolo de 4 fases

5. La activación de la señal 'xia' ocasiona un cambio en la señal 's2a' (0→1), que es la entrada *set* del elemento 'ff2a' del emisor, provocando el cambio de nivel en la salida 'r_inb' (0→1).
6. Con la activación de 'r_inb' (petición del bloque receptor) se inicia un ciclo similar al ocurrido con el bloque emisor. Dado que esta señal está realimentada a la entrada del bloque BCA emisor, provoca que 's1a' cambie (1→0) casi de forma inmediata. Esto no ocasiona ninguna variación en la salida del elemento 'ff1a', por lo tanto momentáneamente será desatendido.
7. Con la activación de la señal 'r_inb' se genera un cambio en la señal 's1b' del bloque receptor.
8. Al cambiar la señal 's1b' (0→1) se activa el elemento 'ff1b' del bloque receptor. Esto ocasiona un cambio en la salida 'x1b' (0→1). El *latch* 'L_b' cierra y captura el primer dato, al mismo tiempo hay un cambio de nivel en la señal 'a_outb', esta indica al bloque emisor que el dato ha sido capturado.
9. Con el cambio de 's1b' baja la señal 's2a' (1→0) sin ocasionar ningún cambio en la señal 'q2a'.
10. Con la activación de 'xib' cambia la señal 's2b' (0→1), que es la entrada *set* del elemento 'ff2b' del bloque receptor. Esto produce un cambio de nivel en la salida 'r_out' (0→1).
11. 'r_out' es realimentado al elemento 'ff1b' del bloque receptor, ocasionando que la señal 's1b' cambie (1→0) sin que ocurra variación alguna en la salida del *flip flop*.
12. La señal 'r_out' es la petición para la siguiente etapa. Ésta se transmite al bloque posterior, provocando que se capture el dato en dicho bloque, que res-

ponde con una señal de reconocimiento ‘a_in’, con lo cual hay un cambio de nivel en la señal ‘s2b’ (1→0) sin que se ocasione ningún cambio en la salida de ‘ff2b’.

13. Regresando al bloque emisor. El cambio de la señal de reconocimiento ‘a_out’(0→1) ocasiona un cambio de nivel en la señal de petición del bloque anterior a-1 (no dibujado), variando la señal ‘r_in’ (1→0), generandose también un cambio en ‘r1a’ (0→1), que es la entrada de *reset* del elemento ‘ff1a’ de la etapa emisora.
14. Con la activación de la señal ‘r1a’, ‘x1a’ cambia (1→0), y además se deshabilita la señal de conocimiento ‘a_out’ (1→0).
15. Cuando la señal‘x1a’ toma el valor “0”, genera el cambio de nivel ‘r2a’ (0→1) y se deshabilita la salida ‘r_inb’ (1→0) del elemento ‘ff2a’ del emisor.
16. Con la realimentación de la señal ‘r_inb’ se cambia la señal ‘r1a’ (1→0) casi de forma inmediata. Ocurriendo lo mismo para señal ‘r2a’ (1→0).
17. La desactivación de ‘r_inb’ genera un cambio en la señal ‘r1b’ (1→0).
18. Con la activación de ‘r_inb’ se produce la deshabilitación de ‘xib’. Esto ocasiona que ‘r2b’ se active, y además se produce la desactivación de ‘a_outb’ que es la señal de conocimiento del bloque receptor.
19. Con el cambio de ‘r2b’ (0→1) se desactiva la señal ‘r_out’.
20. ‘r_out’ se realimenta al elemento ff1b del bloque receptor desactivando la entrada ‘r1b’ (1→0).
21. Al deshabilitarse ‘r_out’ el bloque posterior deshabilita la señal ‘a_in’ y éste a su vez desactiva ‘r2b’. En este momento el BCA receptor es inicializado y esta listo para aceptar un nuevo dato.

5 Módulos de control de 4 fases

En sistemas asíncronos o *micropipelines*, será necesario implementar módulos de control que realicen las funciones de regulación de flujos de peticiones y reconocimientos. Los bloques asíncronos principales son: *CALL, ARBITRER, SELECT* y *TOGGLE* [13], [14], [15].

El bloque *SELECT* envía los eventos de entrada a una de las dos salidas, de acuerdo al valor booleano de la señal de control. Una implementación circuital [16] se muestra en la figura 13.

El comportamiento lógico del bloque *SELECT* se muestra en la figura 14. Se activa la señal de *‘reset’* para llevar al módulo a un estado inicial. La señal de entrada “call_in” representa los eventos de entrada, los cuales pueden ser derivados a una de las salidas seleccionadas: “falso” o “verdadero”, dependiendo del estado de la señal de “control”. Si ésta tiene nivel ‘1’, los eventos se derivan a la selección “verdadero” y, con nivel lógico ‘0’, los eventos se derivan a la selección “falso”.

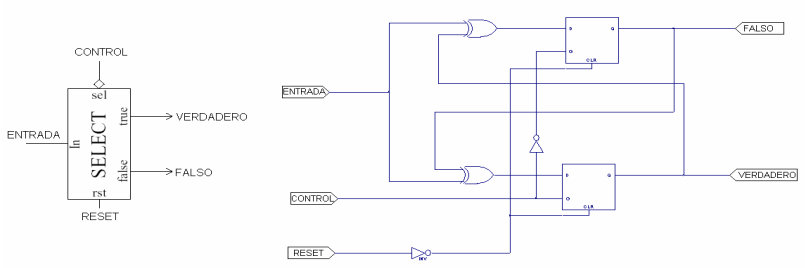

Fig. 13: Esquemático del módulo de control *SELECT*

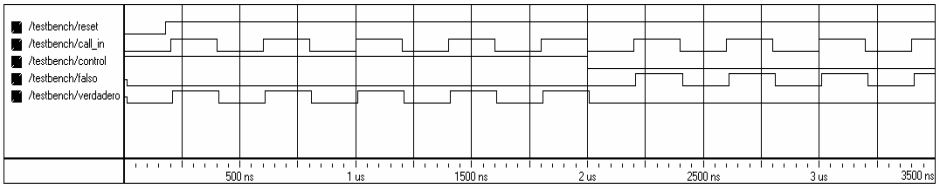

Fig. 14: Simulación lógica del módulo de control *SELECT*

Otro de los módulos de control utilizados en *micropipelines* de 4 fases es el *TOGGLE* cuyo símbolo y diagrama esquemático se muestra en la figura 15. Este elemento tiene una entrada "I" y dos salidas "T1" y "T2". Cada evento en la entrada produce alternadamente un evento en cada salida, comenzando por T1, (salida marcada con un punto en el símbolo de la célula). En la figura 16, se muestra su comportamiento [17],[18],[19].

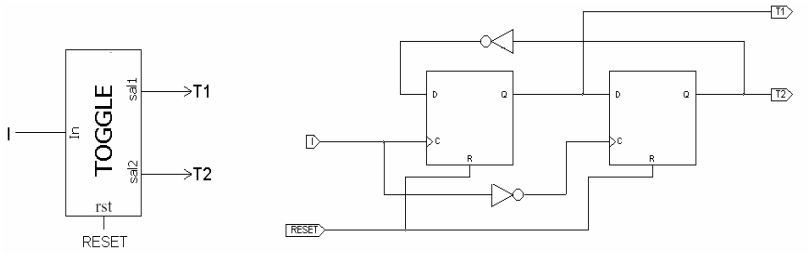

Fig. 15: Esquemático del módulo de control *TOGGLE*.

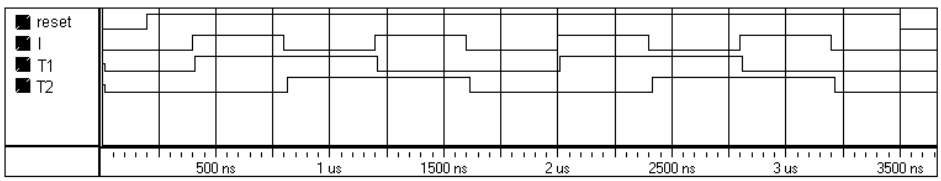

Fig. 16: Simulación lógica de la célula *TOGGLE*.

El bloque *CALL*, proporciona una función equivalente a una "llamada a subrutina", recuerda que emisor realizo un *request* (R_1 o R_2) con lo cual genera un evento en la salida R_sub. Cuando la subrutina se ha completado la etapa receptora devuelve el *acknowledge* (activación y desactivación sucesiva de R_sub y D_sub, acorde con el protocolo). Para que la operación sea correcta el ciclo deberá completarse antes de que ocurra el siguiente *request*, por lo que las dos señales R_1 y R_2 deben ser mutuamente exclusivas. En la figura 17 se muestra el símbolo y el diagrama esquemático de este elemento, y en la figura 18 el comportamiento lógico [20],[21].

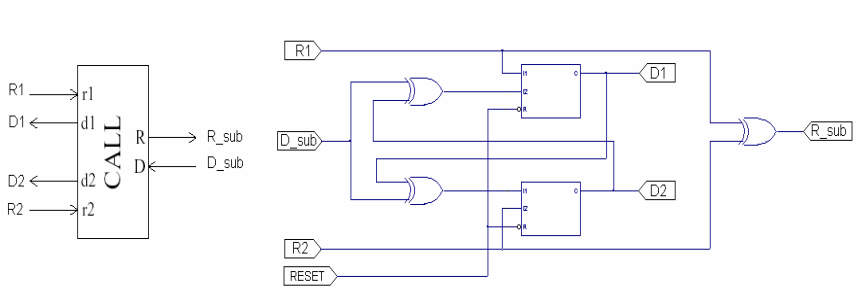

Fig. 17: Esquemático del módulo de control *CALL*.

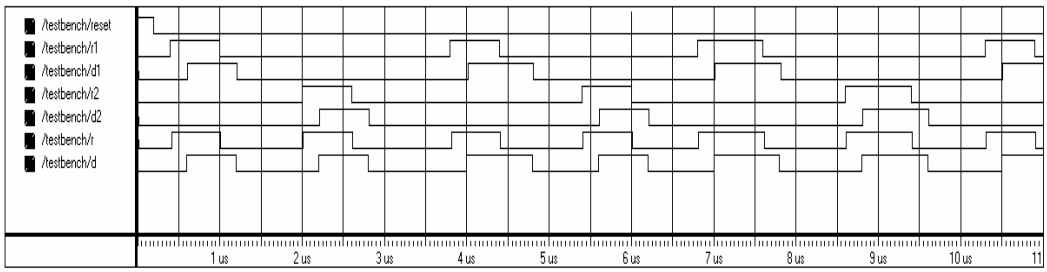

Fig. 18: Simulación lógica del módulo de control *CALL*.

El bloque *ARBITER* tiene como núcleo el circuito denominado *MUTEX* (MUTual EXclusión) cuya implementación se muestra en la figura 19. En si mismo el mutex es un arbitrador compuesto de dos etapas, un *latch* RS y un par de inversores. El *latch* RS almacena las peticiones provenientes de sus entradas, mientras que la siguiente etapa asegura los valores lógicos estables en su salida; aunque el *latch* RS entre en *metaestabilidad* (caso de peticiones simultaneas). Los *latches* de salida del arbitrador almacenan las peticiones y garantizan que permanecen almacenadas hasta que se completa el uso del recurso [8].

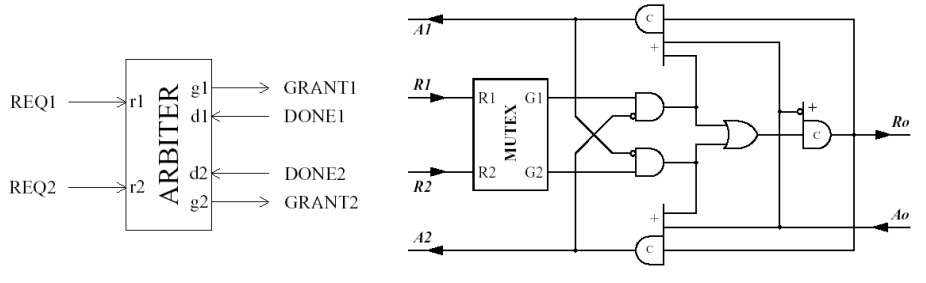

Fig. 19: Esquemático del módulo de control *ARBITER*.

6 Conclusiones

En este tutorial se muestran los principales circuitos de control *Self-Timed* de 4 fases y se describe el *handshake* de dicho protocolo. Se implementa y define el Bloque de Control Asíncrono (BCA) y se muestra un conjunto de módulos de control, utilizados para formar circuitos complejos con protocolo de 4 fases.

Agradecimientos

Este trabajo ha sido financiado por el Proyecto TIC2001-2688-C03-03 del Ministerio de Ciencia y Tecnología de España. La participación de S. Ortega Cisneros ha sido financiada por el Consejo Nacional de Ciencia y Tecnología (CONACYT) de México.

Bibliografía

1 S.B. Furber and P. Day: "Four phase micropipelined latch control circuits", *IEEE Transations on VLSI Systems*, vol 4 , no 2, June (1996), pp. 247-253.

2 Sparso J., Furber S.,"Principles of asynchronous circuit design a systems perspective, European Low-power initiative for electronic system design".,Dimes editor, (2001).

3 KyoungKeun Y., "The Design of a Self-Timed Low Power FIFO using a word-slice structure", M.P. Thesis, *Department of Computer Science*, University of Manchester, UK (1998)

4 Sun-Yen T."High level modelling of micropipelines", M.P. Thesis, Department of Computer Science, University of Manchester, UK (1992)

5 Kelly R.,"Asynchronous Design Aspects of High-Performance Logic Architectural Modelling of a Bipolar Asynchronous Microprocessor", M.C. Thesis, *Department of Computer Science*, University of Manchester, UK (1995)

6 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.*

7 Murata T. : "Petri nets: Propierties, analysis and applications", *Proceedings of the IEEE*, vol 77, num. 4, April (1989), pp 541-580.

8 A. Acosta, A. Barriga, M. Bellido y J. Valencia, “Temporización en circuitos integrados CMOS” , *Editorial Marcombo*, (2000).
9 Herber T.,”Electrónica Digital y Diseño de microprocesadores”, McGraw-Hill, 1991
10 Jacobs G.M. and Brodersen R.W., “Self-timed integrated circuits for digital signal processing applications,” in *VLSI Signal Processing*, III (R. W. Brodersen and H. S. Moscovitz, eds.), IEEE Press, 1988.
11 Sutherland I.E., “Micropipelines”, *Communications of the ACM*, Vol. 32 No. 6, June 1989, pp. 720-738.
12 Tan S., Furber S., and Yen W., “The design of an asynchronous VHDL synthesizer”, en Proc. Design, Automation and Test in Europe (DATE), *IEEE Computer Society Press*, Feb.1998.
13 Theodoropoulos G., “Strategies for the modelling and simulation of asynchronous computer architectures”, Phd. Thesis, *Department of Computer Science*, University of Manchester, UK (1995).
14 Liu, J., “Arithmetic and Control Components for an Asynchronous System”, Ph.D. thesis, *Department of Computer Science*, University of Manchester, UK pp 97-128. (1997)
15 Petlin A., “Random Testing of Asynchronous VLSI Circuits”, M.C. thesis, *Department of Computer Science*, University of Manchester, UK pp 97-128. (1994)
16 Furber S., “La informática sin relojes, lógica asíncrona de bajo consumo”, *Informatica-Inteligencia Artificial* (1995)
17 Endecott P., “Processor Architecture for Power Efficiency and Asynchronous Implementation”,M.C. thesis, *Department of Computer Science*, University of Manchester, UK pp 97-128. (1994)
18 Yun K. Y., Beerel P. A., and J. Arceo, “High-performance two-phase micropipeline building blocks: double edge-triggered latches and burst-mode select and toggle circuits”, IEE Proceedings, Circuits, Devices and Systems, vol. 143, pp. 282–288, Oct. 1996.
19 Chambers P., “The ten commandments of excellent design VHDL code examples”, *Engineering fellow VLSI technology.*
20 Hormdee D., “Copy-back cache organisation for an asynchonous microprocessor”, Phd. thesis submitted to the University of Manchester, *Department of Computer Science* ,2002.
21 Brunvand E., “Using FPGAs to Implement Self-Timed Systems”, University of Utah, *Journal of VLSI Signal Processing* , p. 173-190, July1992.