Sincronización *Self-Timed*: Protocolo de 2 Fases

Boemo E, Ortega S.

Escuela Politécnica Superior, Universidad Autónoma de Madrid, España
eduardo.boemo@uam.es, http://www.ii.uam.es

Abstract. En este artículo se resumen las principales características de la sincronización *Self-Timed* (ST), que da lugar a los circuitos conocidos en la literatura técnica como asíncronos, autotemporizados o *micropipelines.* De las dos opciones principales, 2 y 4 fases, este *tutorial* se centra sólo en la primera. Se explica el funcionamiento de las tres células básicas, el método de sincronización de un *pipeline* ST elemental, y las limitaciones de velocidad de estos sistemas.

1 Introducción

La metodología de diseño de sistemas *Self-Timed* (ST) ha recibido un importante impulso en los últimos años. Entre algunas de sus ventajas [1], se pueden mencionar su inherente funcionamiento en bajo consumo y su inmunidad al *skew* de reloj. Entre las iniciativas recientes en el campo industrial destacan las *smart cards* de Phillips [1], Sun [2][3], Sharp [4], [5]. El tema ha traspasado incluso la propia literatura técnica, mereciendo artículos en el *New York Times* [7], *Forbes* [8] y *The Economist*, [9] probablemente para informar a inversores potenciales sobre los riesgos y oportunidades de las numerosas compañías *start-ups* surgidas al comienzo de este siglo dedicadas principalmente al diseño de microprocesadores ST [10].

En un sistema ST la transferencia de datos es controlada por dos señales *req* (*request*) y *ack* (*acknowledge*) de la forma habitual en cualquier sistema asíncrono: el emisor pone datos en el canal de comunicación y luego activa *req*. Esta señal es utilizada por el receptor para capturar dichos datos. A continuación, éste último activa *ack*, lo cual indica al emisor que la transferencia ha finalizado correctamente. Así, éste envía nuevos datos y el proceso se repite de modo cíclico. La definición del formato de las señales de control da lugar a dos tipos de sincronización: Protocolo de 2 Fases [11] y Protocolo de 4 Fases [12], [13].

Este *tutorial* sólo se centra en los aspectos prácticos del protocolo de 2 fases. La mayor parte del material se ha extraído de dos tesis [14] [15] que siguen el enfoque de la escuela del Prof. Furber [16] de la Universidad de Manchester.

2 Protocolo de 2 Fases

Este protocolo fue propuesto en el célebre artículo *Micropipelines* [11], publicado por Ivan Sutherland en 1989. En la Fig.1 se resume su señalización: el *bus* de datos maneja información codificada en forma convencional mediante niveles lógicos, mientras que las señales *req* y *ack* funcionan por eventos. Un evento es definido como la transición de una

de las señales de control; es decir, un paso de 1 a 0 o de 0 a 1 indistintamente. La figura incluye además el ciclo de transferencia de datos, donde las flechas indican la relación de precedencia entre las señales, de donde se deducen las principales características del protocolo de 2-fases:

1º. Se produce una transferencia de datos en cada evento de *ack* o *req*. Por lo tanto, la frecuencia de transmisión de datos es el doble que la frecuencia de las señales de control. Obsérvese que un sistema síncrono ocurre justamente lo contrario: el reloj tiene como mínimo una frecuencia doble respecto a los datos.

2º. Los eventos en *req* y *ack* se producen alternadamente. Aunque la información está contenida en los flancos, los niveles de *req* y *ack* mantienen siempre la misma relación. Por esta razón es necesario partir del mismo nivel inicial, el cual debe fijarse mediante una señal de *reset*.

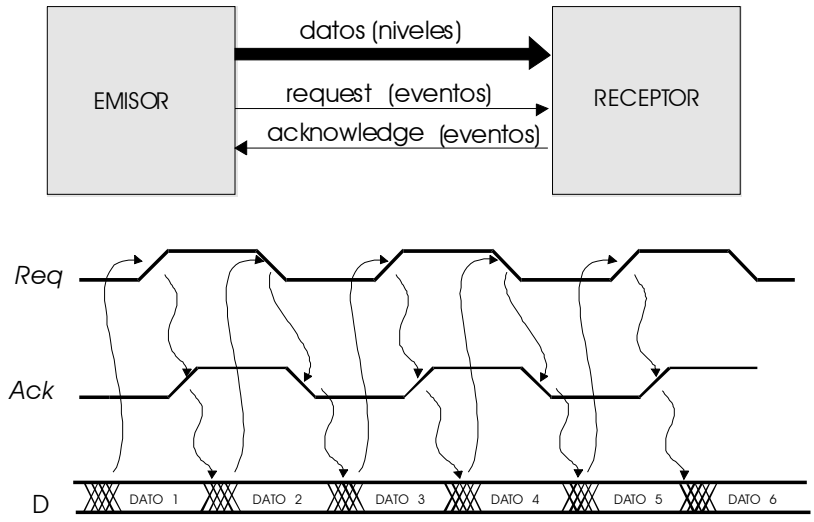

Figura 1: Resumen del esquema de comunicación ST de 2 fases.

2.1 Células elementales para arrays ST

En [11] y [17] se definen distintos tipos de células de control para una sincronización orientada a eventos. Sin embargo, este *tutorial* sólo se centra en las tres células básicas necesarias para construir un *pipeline* ST, llamadas OR de eventos, AND de eventos y *Toggle*. Sus símbolos se muestran en la Fig.2.

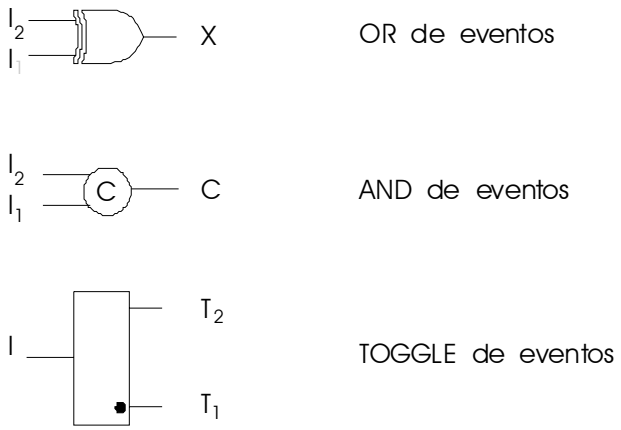

Figura 2: Células elementales para la sincronización ST

La OR de eventos actúa análogamente a una puerta OR, pero para eventos. Es decir: en la salida se produce un evento, cada vez que se produce un evento en cualquiera de sus entradas. En la Fig.3 se muestra su diagrama de tiempos. Se representa con el símbolo de la or-exclusiva por una razón simple: desde el punto de vista lógico, el comportamiento de una OR de eventos es idéntico al de aquella puerta.

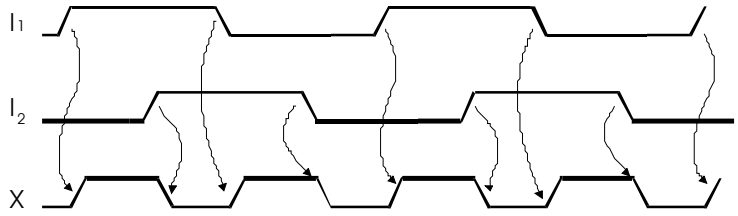

Figura 3: Cronograma de una OR de eventos.

La AND de eventos, también llamado Muller-C, se comporta análogamente a una puerta AND pero para eventos. Es decir: la salida produce un evento solo si en todas sus entradas se han producido eventos.

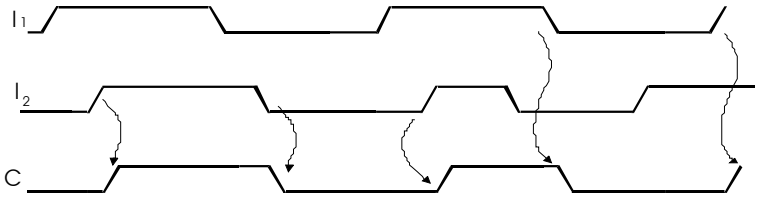

Figura 4: Diagrama de tiempos de una MULLER-C.

La llegada del primer evento en cualquiera de las entradas no modifica la salida. Esto solo ocurre después de la llegada de un segundo evento. En la Fig.4 se muestra el diagrama

de tiempos para dos entradas. Por ser un elemento secuencial, una AND de eventos debe incorporar un *reset* inicial.

Desde un punto de vista práctico, el comportamiento de una Muller-C se puede resumir de dos maneras equivalentes:

1º. Si ambas entradas están en el mismo nivel, la salida copia el valor de la entrada, mientras que si son distintas, la salida se mantiene en el nivel anterior [11].

2º. La salida pasa a "1" solo si todas las entradas están a "1", mientras que la salida pasa a "0" solo si todas las entradas están en "0" [18].

Diversos autores aportan implementaciones de esta célula, cuya función lógica para k entradas es:

$$C(t+1) = I_1\, I_2\, I_3 \ldots I_k + (I_1 + I_2 + I_3 + \ldots + I_k)\, C(t)$$

C(t) es el estado previo y C(t+1) el siguiente estado. En la Fig.5 se muestra su esquema general mediante puertas, aunque también es posible realizarla a partir de árboles de elementos-C de menor tamaño [19].

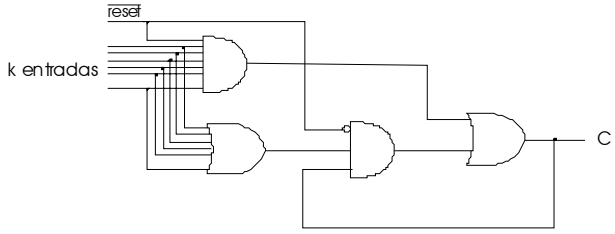

Figura 5: Esquema general de una Muller-C de k entradas.

En la Fig.6 se muestra una realización de la Muller-C de dos entradas [20]. En [13] se presenta una versión equivalente pero incluyendo la inversión de la entrada I2 (necesaria en los esquemas anteriores) y utilizando biestables RS con entradas activas a nivel bajo. Otros ejemplos de esta célula son desarrollados en [21] y [22].

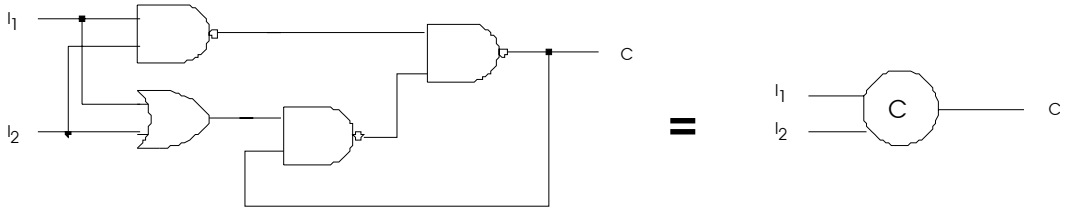

Figura 6: Muller-C de dos entradas.

La última célula básica es la llamada *toggle* de eventos. 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 Fig.7 se resume su diagrama de tiempos.

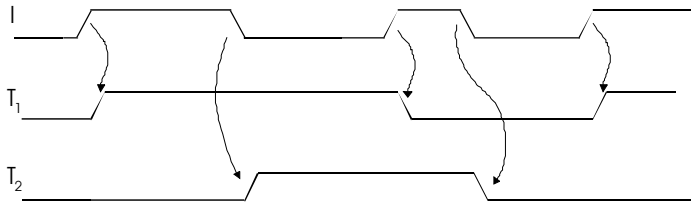

Figura 7: *Toggle* de eventos. Diagrama de tiempos.

Un *toggle* de evento se puede realizar a partir de Muller-C, tal como se muestra en la Fig.8. En tecnologías basadas en LUTs, tal como ocurre en las FPGAs, la implementación del *toggle* en un CLB también puede incluir otras células asíncronas sin costo adicional en área.

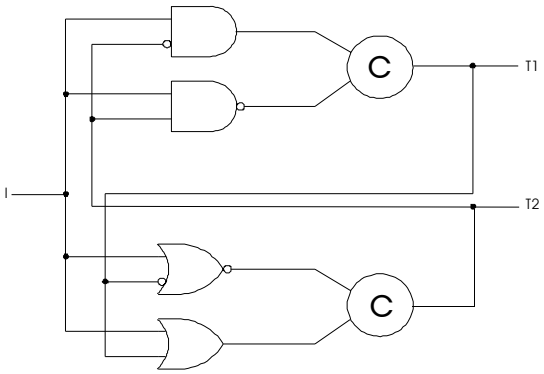

Figura 8: *Toggle* de eventos a partir de células Muller C.

3 Sincronización de un Pipeline

Con las tres células anteriores se puede sincronizar un *pipeline.* El esquema del circuito se puede observar en la Fig.9 [11] [14]. Las tres diferencias con su homólogo síncrono son:

a. La línea de reloj es reemplazada por una cadena de bloques de control asíncronos (en adelante, BCA).

b. Toda la interconexión es local. La única excepción es la señal general de *reset*, omitida en la figura.

c. Se utilizan *latches* en lugar de registros. En los ejemplos siguientes, se considera que estos *latches* son transparentes con 0 y cierran con 1.

La figura también descubre el punto más débil de los sistemas ST: la necesidad de agregar un retardo *d* (d_1, d_2, etc. en la figura), de manera que la señal *req* de cada etapa (que se genera a partir del *ack* anterior) no llegue al *latch* antes que los datos a capturar.

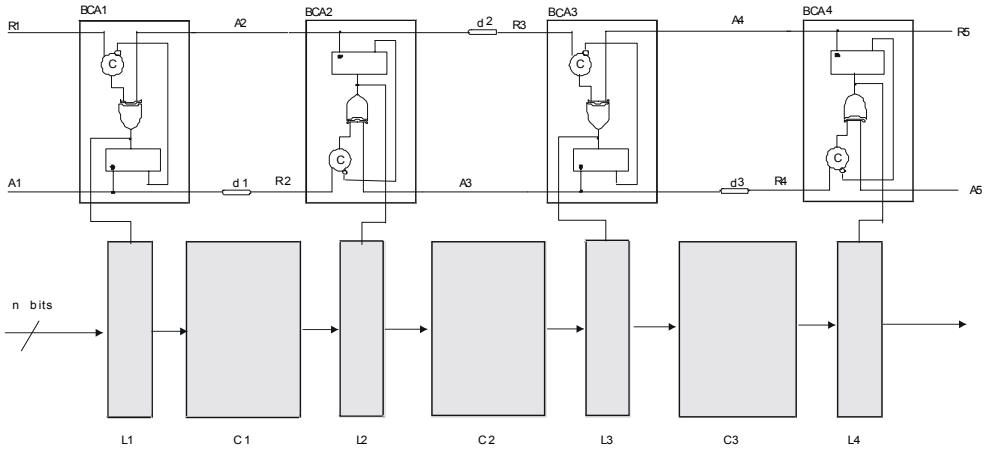

Figura 9: Esquema de un *pipeline* ST.

Para ilustrar el mecanismo de sincronización del circuito sólo es necesario focalizar la atención sobre una etapa genérica del *pipeline* de la Fig.9. El análisis se puede descomponer en una sucesión de eventos, cuya secuencia se resume en la Tabla 1. Se representa gráficamente en la Fig.10, donde además se incluyen algunos ciclos adicionales para evidenciar el carácter periódico del funcionamiento del circuito. Por simplicidad, se utilizará la siguiente nomenclatura: R: req, A: ack, C: señal de salida de la Muller-C, X: señal de control del *latch,* transparente con "0", T1 y T2: señales de salida del *toggle.*

4 Memoria FIFO ST

Una memoria FIFO ST [23] se obtiene simplemente eliminando la parte combinacional del esquema de la Fig.9. Para esta aplicación, la técnica ST destaca por su sencillez y eficacia, haciendo innecesarios los circuitos para mantener los punteros de lectura y escritura y los generadores de dirección de las FIFOs síncronas.

Evento	Efecto sobre el circuito
reset inicial	Las señales R, A, C, X, T_1 y T_2 de todos los BCA pasan a 0.
A	La etapa i-1 (no dibujada) impone el dato 1 en la entrada del *latch* L_i y luego produce un evento en R_i (paso de 0 a 1).
B	El evento en R_i produce un evento en la salida C_i de la Muller-C (paso de 0 a 1), pues sus dos entradas están a 1 (obsérvese que la otra entrada de la Muller-C está negada).
C	Se produce un evento en la salida X_i de la XOR (paso de 0 a 1), dado que A_j=0. Con X_i=1 el *latch* L_1, cierra y captura el dato 1.
D	El evento en X_i hace que el TOGGLE, produzca un evento (paso de 0 a 1) en la salida $T_{1i} = A_i$. Así, el *ack* para el bloque i-1 es generado.
E	Como consecuencia del *ack* anterior, el bloque i-1 pone el dato 2 en el *bus* y envía un nuevo R_i.
F	Este nuevo *req* no produce cambio en la salida C_i, y por lo tanto, momentáneamente será desatendido.
A'	La señal A_i retrasada un tiempo d_i igual al retardo del bloque combinacional C_{ij}, constituye la señal R_j.
C'	El *latch* j cierra y captura el dato 1, pues el evento en R_j (paso de 0 a 1) ha marcado el comienzo de un proceso idéntico al descripto, pero para la etapa j.
D'	Se produce el *ack* hacia la etapa i. Las señales en j han quedado en los estados: C_j=1, X_j=1 y A_j=1.
A"	El ack anterior retrasado constituye el req hacia la etapa k.
G	El evento en A_j (paso de 0 a 1) produce un evento en X_i (paso de 1 a 0). El *latch* L_i queda nuevamente transparente y se produce la entrada de los datos 2.
H	El evento en X_i hace que el $TOGGLE_i$ produzca un evento en T_{2i} (paso de 0 a 1).
I	El evento en $T2_i$ produce un nuevo evento en la salida C_i de la MULLER-C (paso de 1 a 0), pues sus dos entradas han pasado a 0.
J	El evento en C_i produce un nuevo evento en la salida de la XOR (paso de 0 a 1), dado que A_j=1. El *latch* L_i se cierra y captura el dato 2.

Tabla 1: Evolución de un pipeline ST.

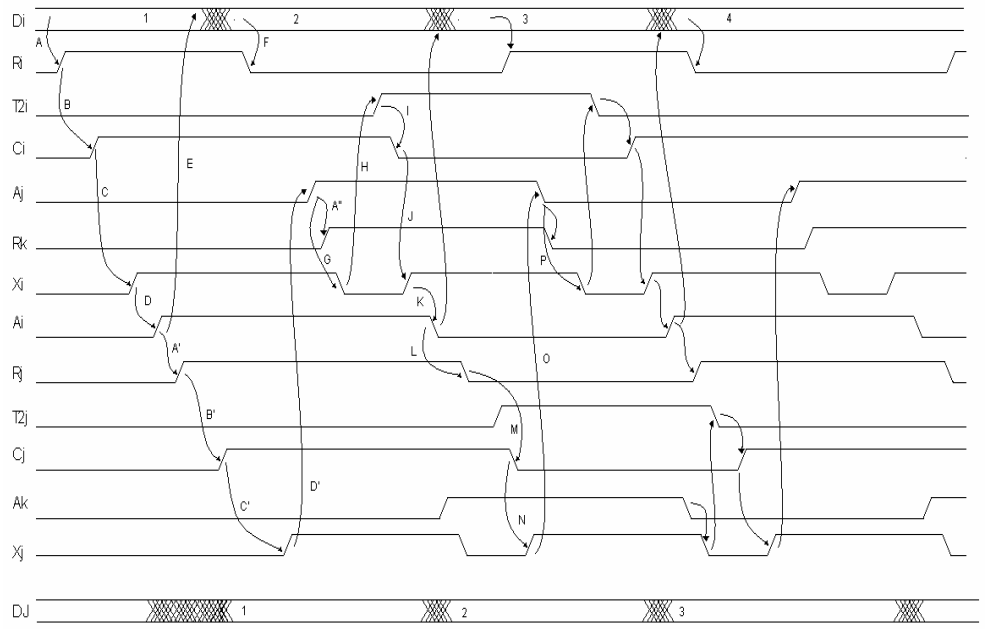

Figura 10: Diagrama de tiempos de una etapa genérica de un *pipeline* ST con protocolo de 2 fases.

En una FIFO ST, las señales de lectura/escritura (R/W) son reemplazadas por A_O (el *ack* de la última etapa) y R_I (el *request* de la primera etapa). Para almacenar un dato, el emisor debe producir un evento en R_I que hace la función de señal de escritura. A partir de dicho evento, el dato es movido automáticamente hasta el otro extremo de la FIFO, hasta quedar contiguo al dato último dato ingresado. En el otro extremo, el receptor solo debe producir un evento en A_0 (señal de lectura) para extraer un dato. Nuevamente, este evento hace que los siguientes datos se muevan automáticamente una posición hacia la salida. Obsérvese que no son necesarias las señales "vacía" (*empty*) ni "llena" (*full*): la primera condición equivale a la ausencia de un nuevo evento en R_O, después de la extracción de un dato por el receptor. La segunda equivale a la ausencia de eventos en A_I después de un requerimiento del emisor.

5. Sincronización ST de un *array*

La aplicación de técnicas ST en *array* 2D presenta algunas ventajas potenciales que deben ser analizadas para cada tecnología en particular. Algunos antecedentes sobre este tema son: [24], [14] y [25], todos orientados a multiplicadores ST.

La diferencia con los casos anteriores es que en un array, los datos de entrada provienen de diferentes procesadores elementales (PEs), y lo mismo ocurre con los resultados. Así, para sincronizarlos se debe establecer una dupla *req-ack* por cada canal de comunicación entre PEs. En la Fig.11 se muestra el esquema general de un BCA(p,q), un bloque para gestionar la comunicación de un PE con p canales de entrada y q canales de salida.

A diferencia de los casos anteriores, ahora el BCA incluye células Muller-C con p y q líneas de entrada. Su comportamiento sigue correspondiendo a una AND para eventos: produce un evento en su salida solo cuando se ha producido un evento en todas sus entradas. La comunicación ST entre un PE genérico con el resto del *array* se puede sintetizar con tres reglas:

1º El PE puede comenzar el procesamiento solo si los p datos de entrada están disponibles. Para detectar esta situación utiliza una Muller-C con p líneas de *req*.

2º Antes de aceptar nuevos datos, el PE debe averiguar si los q PEs receptores han capturado el resultado anterior. Para detectar esta situación utiliza una Muller-C para procesar los q *acks*, y producir el evento que lo saca del estado de espera.

3º Cuando el PE termina el proceso, debe generar un único *Ack* (que envía a los p emisores) y un único Req (que envía a los q receptores).

6 Análisis temporal de la sincronización ST

El análisis temporal de un circuito ST no difiere de su homólogo síncrono; los puntos de interés son: doble captura y captura nula, ambos definidos en [26], violaciones del *setup* y *hold*, y finalmente frecuencia máxima de operación.

Un circuito segmentado ST es inmune a la doble captura. La célula *toggle* tiene por función justamente evitar ese fenómeno bloqueando la Muller-C. La ocurrencia de doble captura equivaldría a que un segundo *req* (instante 3 de la Fig.12) abriera nuevamente el *latch* L_i. Así, nuevos datos se superpondrían a los que aún no ha capturado el *latch* L_j.

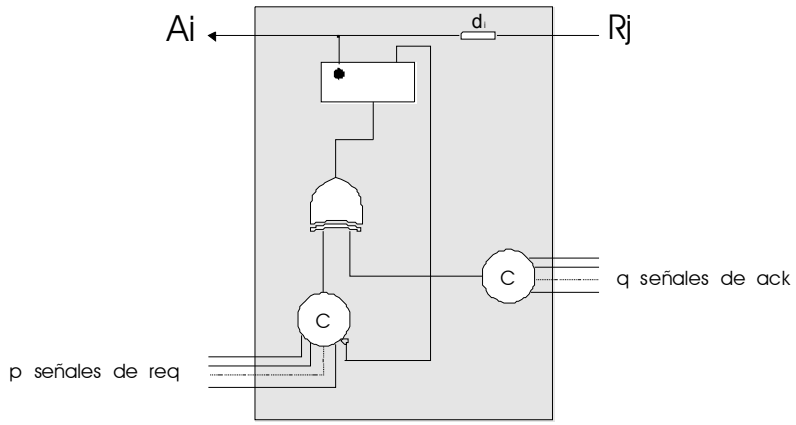

Figura 11: Bloque de control asíncrono con capacidad para la gestión de p canales de entrada y q canales de salida.

El otro fallo de sincronización, la captura nula, ocurre cuando los datos capturados en el *latch* L_i son procesados, pero llegan a L_j después que éste se ha cerrado. A diferencia que en el caso anterior esta situación es posible y constituye uno de los puntos débiles de los circuitos ST. Este inconveniente se salva habitualmente retardando la salida T_1 un tiempo *d*, de manera de asegurar que los datos atraviesan el segmento combinacional antes de la llegada del *req*.

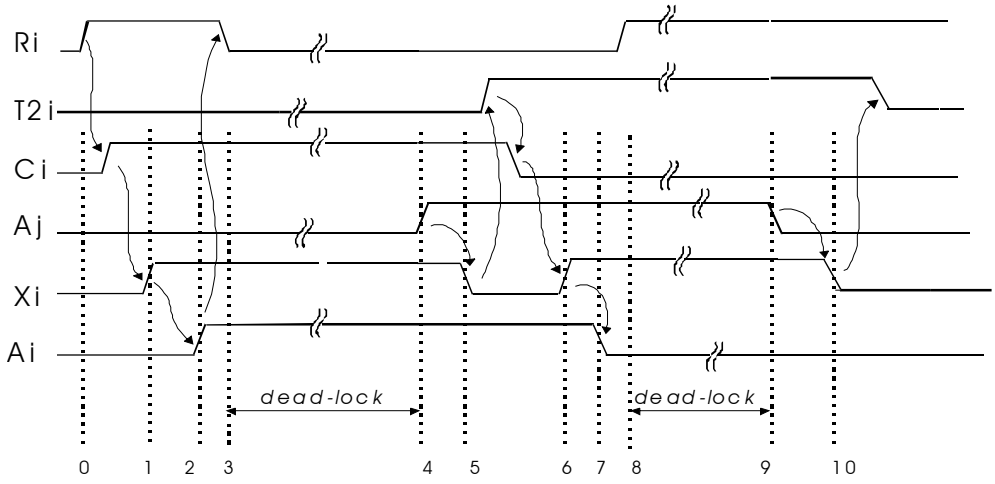

Figura 12: Captura nula y doble captura en sistemas ST

El valor óptimo del retardo *d* depende del resultado de la carrera entre los nuevos datos y la señal R_j, cuyo instante de partida lo marca el evento en X_i que cierra el *latch* L_i. Si se asigna t=0 a este último, el dato está disponible para su captura en el *latch* L_j en el peor caso después de transcurrir un tiempo:

$$t_{datos} = \delta_{Ri,max} + C_{ij,max} + \text{Setup}_{\text{ latch}}$$

Donde δ_{Rimax} y C_{ijmax} son el retardo máximo del *latch* y la etapa combinacional respectivamente. Mientras, la señal X_j cierra el *latch* L_j en (peor caso) en el tiempo correspondiente al camino de eventos K-L-M-N que conecta las subidas de X_i y X_j, las señales de control de dos *latches* consecutivos. Es decir,

$$t_{control} = \delta_{Ti,\,mín} + d_{i,mín} + \delta_{Mj,mín} + \delta_{Oj,mín}$$

Donde $\delta_{Ti,\,mín}$, $\delta_{Mj,mín}$ y $\delta_{Oj,mín}$ son los retardos mínimos del *toggle*, Muller y OR de eventos respectivamente. $d_{i,mín}$ es la tolerancia mínima del retardo d. Considerando el peor caso para cada célula se pueden eliminar los subíndices, quedando la condición para evitar captura nula como:

$$d > (\delta R + C + \text{Setup}_{\text{ latch}}) - (\delta_T + \delta_M + \delta_O)$$

La ecuación anterior se considera que los retardos de interconexión están incluidos dentro del retardo de cada bloque. Por lo tanto, para FPGAs, el valor de *d* debe ser tolerante a variaciones en los resultados de un PPR (*partitioning, placement, routing*) automático.

t	Situación en el circuito
t_0	En t_0 se produce un evento en R_i (paso de 0 a 1) y el circuito evoluciona normalmente hasta t_2.
t_3	Aparece un nuevo R_i. La entrada de la Muller-C es 01, por lo tanto la salida C_i se mantiene invariable en 1. La XOR_i y $TOGGLE_i$ tampoco cambian su salida y el sistema entra en un *dead-lock*.
t_4	El evento en A_j desbloquea el circuito y renace la actividad en las etapas involucradas.
t_5-t_6	Se produce la captura del dato 2. El evento en A_j hace pasar X_i a 0. Este cambio hace bascular nuevamente al TOGGLE, que ahora produce un evento en T_{2i}, que se realimenta a la Muller-C. C_i pasa a 0 y X_i a 1. El *latch* 1 se cierra nuevamente.
t_7	El evento en X_i produce el *ack* correspondiente al *req* del t_3.
t_8	Hay un nuevo *req* y el proceso se repite.

Tabla 2. Análisis de doble captura.

Tiempo de *Hold* y Ancho Mínimo de Pulso: En el punto anterior se ha incluido el efecto del *setup* y retardo de propagación de los *latches* sobre la temporización del circuito. Solo resta analizar las restricciones que imponen los otros dos parámetros: el tiempo de *hold,* H, y el ancho mínimo de pulso W. Para el primero debe cumplirse que:

$$\delta_{Tj,mín} + \delta_{Oi,mín} + \delta_{Ri,mín} + C_{ij,máx} > H$$

Donde $C_{ij,máx}$ es el máximo retardo combinacional de una etapa del *pipeline*. Es decir, el lapso entre el cierre de Lj y la llegada de un nuevo a su entrada debe ser mayor que H. En cuanto al ancho mínimo del pulso W, debe cumplirse que:

$$\delta_{O,mín} + \delta_{T,mín} + \delta_{M,\,mín} > W$$

El tiempo de apertura del latch depende del retardo de propagación de los eventos en el bucle XOR-Toggle-Muller de cada BCA. Ninguna de las dos condiciones constituye una seria restricción en una implementación del circuito.

Frecuencia máxima de operación: En un sistema ST, el intervalo entre dos eventos sucesivos (para *ack*) o dos flancos de bajada sucesivos (para la señal X, de control de los *latches*) marcan la entrada de un nuevo dato. Este último corresponde por ejemplo, al camino H-I-J-K-L-M-N-O-P de la Fig.10. En el peor caso, se pueden eliminar los subíndices quedando el período de operación limitado por:

$$T > d_{máx} + 3\,\delta_{T,máx} + 3\,\delta_{O,máx} + 2\,\delta_{M,\,máx}$$

La comparación entre la expresión anterior y la homóloga para un sistema síncrono permite predecir, para una determinada tecnología, el valor de *skew* para el cual la opción ST superaría en velocidad a la síncrona.

7 Conclusiones

Se han presentado las principales características del protocolo ST de 2 fases. El costo en área de este tipo de sincronización es importante; para un ASIC, debe evaluarse si alcanza a compensar la reducción de área asociada a la eliminación del árbol de distribución de reloj. En cuanto a la velocidad, la expresión de arriba indica que el período de un sistema está fuertemente limitado por el retardo de las células básicas de control.

Una de las aplicaciones donde un sistema ST puede encontrar su nicho, es en el diseño de *Smart Cards* inalámbricas y etiquetas inteligentes, donde, la energía para activar el circuito se obtiene de una fuente de radiofrecuencia. En estos casos, la potencia pico disponible es muy baja y podría no ser suficiente para alimentar un sistema síncrono en el instante posterior a un flanco de reloj. Por el contrario, la ausencia de reloj de un sistema ST hace que los flancos de sincronización (y por lo tanto los picos de consumo) no coincidan en el tiempo, disminuyendo la distancia entre el valor medio y pico de corriente de alimentación [27].

Las ventajas de la temporización asíncrona cuando el tamaño del circuito es "grande", es un argumento que figura en la introducción de prácticamente todos los artículos sobre circuitos *self-timed*. Sin embargo, usualmente no adjunta ningún pronóstico sobre el tamaño límite de un sistema síncrono. Una excepción es el trabajo de Kappler et al [28] sobre FCCMs para la comparación de cadenas genéticas, donde los autores pronostican máquinas de hasta 25000 chips. Para un sistema de tal tamaño, la técnica ST debería tenerse en cuenta.

Para finalizar, los lectores interesados en sistemas ST pueden comenzar por dos excelentes libros de texto, uno de ellos en castellano: [29] [30]. Una lista *on-line* permanentemente actualizada de artículos sobre ST es mantenida en la *Technische Universiteit Eindhoven* [31].

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. Hauck, S. “Asynchronous Design Methodology: An Overview”, Proceedings of the IEEE, Vol.83, No 1, Enero 1995.
1. Yoshida, J “Philips gambit: Self-timing's time is here”, *EE Times* Marzo 31, 2003.
2. Sutherland I , Lexau, J, “Designing fast asynchronous circuitos”, *Proc ASYNC 2001*, Univ. de Utah: Marzo 2001.
3. Johnson C, “Scrap system clock, Sun exec tells Async”, *EE Times*, 19 de Marzo de 2001.
4. Brunvand, E.; Nowick, S.; Yun, K.; Practical advances in asynchronous design and in asynchronous/synchronous interfaces”, *Proc. 36th Design Automation Conference*, page(s): 104 - 109, 21-25 June 1999.
5. Terada, H.; Miyata, S.; Iwata, M.; “DDMPs: self-timed super-pipelined data-driven multimedia processors”, *Proceedings of the IEEE,* Volume: 87 Issue: 2 , Page(s): 282 -295, Feb. 1999.
6. D. Edwards y W. Toms, “The Status of Asynchronous Design in Industry”, Feb 2003. Disponible en: http://www.scism.sbu.ac.uk/ccsv/ACiD-WG/AsyncIndustryStatus.pdf
7. Markoff J, "Computing Pioneer Challenges the Clock”, *New York Times*, Marzo 5, 2001.
8. Tristram C, “It's Time for Clockless Chips”, *Forbes*, Octubre 2001.
9. *The Economist*, “Old tricks for new chips”, Abril 19, 2001. Disponible en: http://www.cs.columbia.edu/ async/misc/economist/Economist_com.htm.
10. Cataldo A, “Iconoclast designers choose MIP”, EE Times May, 2002.
11. I. Sutherland, "Micropipelines", *Communication of the ACM*, vol.22, nº6, pp.720-734. Jun. 1989.
12. T. Meng, R. Brodersen y D. Messerschmitt, "Automatic Synthesis of Asynchronous Circuits from High-Level Specificacions", *IEEE Transactions on Computer-Aided Design*, vol.8, n11, Noviembre 1989.
13. G. Jacobs y R. Brodersen, "A Fully Asynchronous Digital Signal Processor Using Self-Timed Circuits", *IEEE Journal of Solid-State Circuits*, Vol.25, Nº6. Diciembre 1990.
14. J. Arechabala, *"Test Interfase for Micropipelined Units",* Master Sc. Thesis, Dept. of Computer Science, Universidad de Manchester, UK, Octubre 1992.
15. Boemo E, *“Contribución al Diseño de Arrays VLSI con Paralelismo de Grano Fino”,* Tesis de Doctorado, ETSI Telecomunicación, UPM, 1985.
16. Grupo de Diseño Asíncrono, Univ. de Manchester: http://www.cs.man.ac.uk/apt/index.html
17. Brunvard E, "Using FPGAs to Implement Self-Timed Systems", *Journal of VLSI Signal Processing*, nº6, pp.173-190. Boston: Kluwer Academic Publishers. 1993.
18. G. Jacobs y R. Brodersen, "Self-Timed Integrated Circuits for Digital Signal Processing Applications", en *VLSI Signal Processing III,* editado por R. Brodersen y H. Moscovitz. New York: IEEE Press, 1988.
19. J. Tierno, A. Martin, D. Borkovic y T. Lee, "A 100-MIPS GaAs Asynchronous Microprocesor", *IEEE Design and Test of Computers*, pp.43-49, verano 1994.

20. I. David, R. Ginosar y M. Yoeli, "An Efficient Implementation of Boolean Functions as Self-Timed Circuits", *IEEE Trans. on Computers*, pp.2-11, vol.41, No.1, Enero 1992.
21. T. Wuu y S. Vrudhula, "A Design of a Fast and Area Efficient Multi-Input Muller C-element", *IEEE Trans. on VLSI Systems*, Vol.1, nº2, Jun. 1993.
22. S.L. Lu, "Improved design of CMOS multiple-input Muller-C-elements", *Electronic Letters*, Pag.1680-1682, Vol.29, Nº19, 16 de Septiembre de 1993
23. Sutherland; I, "Asynchronous first-in-first-out register structure", Patente US1987000118997, Nov. 10, 1987
24. L. Lau, D. renshaw y J. Mavor, "A Self-Timed Wavefront Multiplier", *Proc. ISCAS´89*, pp.138-140. IEEE Press 1989.
25. A. Acosta, M. Bellido, A. Barriga y M. Valencia, "Arquitectura para el Diseño de Circuitos Autotemporizados Bidimensionales. Realización de Multiplicadores", *Proc. VIII Congreso de Diseño de Circuitos Integrados*, pp.179-184. Málaga, Noviembre 1993.
26. J. Fishburn, "Clock Skew Optimization", *IEEE Trans. on Computers,* Vol.39, Nº7, pp.945-951, July 1990.
27. Kessel J. "Asynchronous Circuits in Contactless Smart Cards", *Proc. 4th ACiD (Asynchronous Circuit Design) Workshop,* Grenoble: TIMA Laboratory, Feb.2000
28. C. Kapper, J. Oldfield y K. Shen, "Genetic String Distance Evaluation with an Array of Self-Timed FPGA Chip", en *More FPGAs*, Editores W. Moore y W. Luk, pp.435-442. Abingdon EE&CS Books: Oxford, 1994.
29 J. Sparsø and S. Furber (eds.), "*Principles of asynchronous circuit design - A systems perspective"*, Kluwer Academic Publishers, 2001.
30. A. Acosta, A. Barriga, M. Bellido y J. Valencia, *"Temporización en circuitos integrados CMOS"*, Editorial Marcombo, 2000.
31. Ad Peeters, "The Asynchronous Bibliography", http://www.win.tue.nl/async-bib/doc/async.pdf