Síntesis de Máquinas de Control para Circuitos Asíncronos

Oriol Roig, Enric Pastor, Rosa M. Badia, Jordi Cortadella
Depto. de Arquitectura de Computadores
Universitat Politècnica de Catalunya

Resumen

Este artículo presenta una metodología para la síntesis de circuitos de control en una herramienta de síntesis de alto nivel para circuitos asíncronos. Suponiendo que las fases de planificación *y de* asignación de recursos *ya han sido realizadas, se centra en la especificación y síntesis de las unidades de control. Las unidades funcionales se implementan con tecnología DCVSL, mientras que los circuitos de control se especifican en forma de STGs, y se sintetizan utilizando técnicas de síntesis lógica y eliminación de riesgos.*

1 Introducción

Durante estos últimos años, los circuitos asíncronos han ganado interés en la síntesis de circuitos digitales. Las ventajas inherentes de estos circuitos, eliminación de relojes globales, alta modularidad, bajo consumo, y la posibilidad de obtener tiempos de cálculo promedio, han inducido el desarrollo de herramientas de CAD para su síntesis. Sin embargo, problemas como la metaestabilidad, las carreras y los riesgos, dificultan enormemente el diseño de los circuitos asíncronos.

De entre los trabajos realizados en el campo de la síntesis de circuitos asíncronos cabe remarcar los de Nowick y Dill [1], Yun *et al.* [2], Lavagno *et al.* [3], Moon *et al.* [4] K. van Berkel [5] y Alain Martin [6].

El presente artículo presenta una metodología para la síntesis de controladores locales en la herramienta de síntesis de alto nivel para circuitos asíncronos *Achilles* [7]. Supondremos que las fases de *planificación* y de *asignación de recursos* ya han sido realizadas [8], y nos centraremos en la especificación y síntesis de las unidades de control. Las unidades funcionales se implementan con tecnología DCVSL, mientras que los circuitos de control se especifican en forma de STGs, y se sintetizan utilizando técnicas de síntesis lógica y eliminación de riesgos.

Este artículo está estructurado de la siguiente forma. La sección 2 ofrece una visión general del sistema de síntesis y del modelo de arquitectura utilizado. La sección 3 presenta la técnica de implementación del camino de datos, el diseño de las unidades funcionales, y su relación con las unidades de control. La sección 4 describe como las unidades de control son sintetizadas utilizando STGs. En la sección 5, a modo de ejemplo, se ilustra el proceso de síntesis de un filtro para suavizar una función de una variable. Finalmente, en la sección 6 se presentan algunas conclusiones.

2 Sistemas Asíncronos

A medida que aumenta el tamaño de los circuitos también aumenta la complejidad de su diseño. La ausencia de reloj en los circuitos asíncronos permite una división en módulos de los circuitos sin perjudicar su funcionalidad. La modularidad inherente de los circuitos asíncronos facilita su diseño ya que permite realizar cada etapa considerando en cada momento únicamente aspectos locales.

Un sistema asíncrono puede ser modelizado como un sistema multiprocesador donde los elementos del sistema son procesadores que ejecutan un proceso y que se comunican entre sí de manera totalmente asíncrona [9].

Cada procesador de este sistema está formado por dos componentes bien diferenciados (ver figura 1). Por un lado, un bloque o grupo de bloques del *data-path* (D), cada uno de los cuales es un componente *self-timed* controlado por señales de *handshake* [10]. Esta parte del procesador puede ser tan sencilla como un registro o un sumador o tan compleja como el *data-path* entero del circuito. El otro componente es el *controlador local* (C) del procesador, de manera que el control del sistema multiprocesador está totalmente distribuido en diferentes controladores locales.

Cada controlador local dirige el bloque o grupo de bloques del procesador mediante el intercambio de señales de *handshake*. Estas señales son básicamente la señal *request* y la señal *complete* que indican, respectivamente, que se requiere la ejecución de una operación y que la operación ha terminado. Los controladores locales se comunican de manera asíncrona entre sí cuando es necesario el intercambio de datos entre diferentes procesadores.

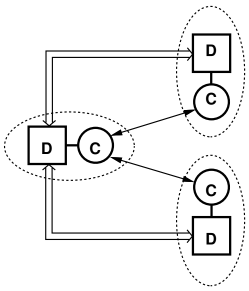

Figura 1: Sistema Asíncrono

Cada procesador ejecuta un proceso secuencial formado por primitivas de comunicación (*send* y *receive*) y operaciones a ejecutar por los componentes del *data-path*. Las primitivas de comunicación describen las transacciones de datos que se han de realizar entre los diferentes procesadores. Estas transacciones vienen determinadas por las dependencias de datos entre operaciones que se ejecutan en procesadores diferentes.
En un sistema de síntesis de alto nivel como el descrito en [7] el proceso secuencial a ejecutar por cada proceso puede ser obtenido tras las fases de *planificación* y de *asignación de recursos*.
Por otro lado, el comportamiento de los diferentes controladores locales puede ser descrito con un STG, y los circuitos que los implementan, generados con alguna de las diferentes técnicas para sintetizar circuitos libres de riesgos a partir de STGs.

3 Componentes del Data-path

Los componentes del *data-path* (camino de datos) son módulos que permiten seleccionar, almacenar u operar datos de n bits. Los más usuales son multiplexores, registros y unidades funcionales (sumadores, multiplicadores, etc.).

3.1 Componentes síncronos

En los circuitos síncronos el retardo de un módulo viene determinado por su camino crítico. Esto es un inconveniente, especialmente cuando se aplica a las unidades funcionales, ya que las diferencias entre el mejor y el peor caso pueden ser muy importantes.

3.2 Componentes "self-timed"

Los circuitos *self-timed* [10] poseen un mecanismo que permite indicar cuando acaba la operación que se estaba realizando, y por tanto, las señales de salida son válidas. La sincronización con estos bloques se realiza mediante las señales R (*request*, señal de entrada) y C (*complete*, de salida). El protocolo usado en la ejecución de una operación es un *handshake* [10] de cuatro fases, que consta de los siguientes pasos:

- R^- solicita el *reset* del bloque [1].
- C^- indica que el *reset* ha terminado.
- R^+ solicita la ejecución de un calculo.
- C^+ indica que el cálculo ha finalizado.

Los bloques necesitan un cierto tiempo de *reset* antes de poder ejecutar la siguiente operación. Por otro lado, el hecho que el retardo dependa de los datos de entrada los hace muy atractivos, ya que se podrá hablar de un tiempo medio de ejecución, en lugar del peor de los casos.
La conexión entre varios de estos componentes se realiza de la siguiente forma: mientras que los datos se intercambian directamente entre los distintos módulos, las señales R y C se conectan con los controladores locales que gestionan la secuencia de las operaciones.
Hay varias posibilidades a la hora de diseñar bloques *self-timed*, de las cuales explicaremos las dos más utilizadas:

- *Bundled data* [11]: C simplemente es la señal R retardada (figura 2). Esto sólo es adecuado para aquellos módulos con tiempos "constantes", es decir, que el mejor y el peor caso son muy parecidos. Usar una solución de este tipo simplificará la lógica del circuito si la comparamos con la de la siguiente opción.

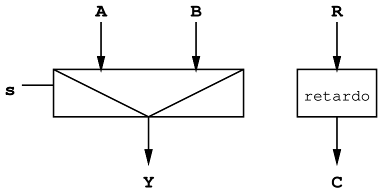

Figura 2: Multiplexor *bundled data*

- *Dual-rail* [12]: Cada bit de datos A está representado por dos señales A^T y A^F. En la tabla 1 se ve la codificación más común.

  De este modo se permite detectar fácilmente el final de la operación cuando todas las salidas están complementadas, por lo que esta opción es válida para todos los componentes del *data-path*.

[1] R^- indica una transición $1 \rightarrow 0$ de la señal R, R^+ indica una transición $0 \rightarrow 1$.

Señal A^T	Señal A^F	Bit A
0	0	*Reset* = No está listo
0	1	Evalúa FALSO
1	0	Evalúa CIERTO
1	1	No usado = Nunca ocurrirá

Tabla 1: Codificación *dual-rail*

3.3 Familia DCVSL

DCVSL (*Differential Cascode Voltage Switch Logic*) [13] es una familia lógica que utiliza la codificación *dual-rail* en los datos de entrada y salida. Las puertas generan las funciones z^T y z^F precargando dos transistores pmos durante la fase de *reset*, y descargando sólo uno de ellos a través de una de las dos redes de *pull-down* en la fase de ejecución. Para construir el protocolo de *handshake* antes mencionado, se debe conectar la señal R a los transistores de precarga, mientras que se obtiene C como la suma lógica de z^T y z^F. En la figura 3.a se observa una puerta AND realizada con esta familia.
La red de *pull-down* que calcula z^T es la misma red de transistores nmos usada en la puerta análoga en CMOS estático, mientras que la que calcula z^F es isomórfica a la de transistores pmos. Esta es la forma habitual de construir puertas con lógica DCVSL, pero en ciertos casos, con puertas complejas, alguno de los transistores de ambas redes se pueden compartir, con lo que se reduce su número. Sirva como muestra la puerta XOR de la figura 4.

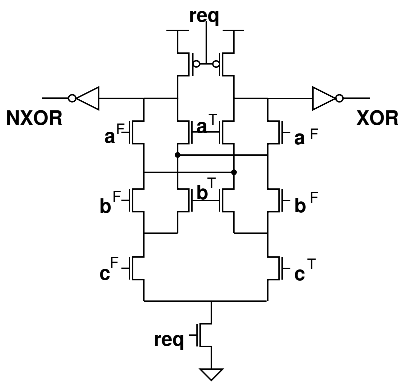

Figura 4: Puerta XOR DCVSL

Otra importante ventaja de esta familia es el hecho que el resultado calculado se mantiene aunque todas las entradas pasen a valer cero de nuevo. Esto permite hacer el *reset* de un módulo cuando aquellos que utilizan su resultado han terminado, ya que estos últimos seguirán manteniendo los mismos valores de salida cuando sus entradas valgan cero.
En la figura 3.b se observa cómo se puede detectar el final de la ejecución de un módulo con salidas multiples. Se puede utilizar un árbol de elementos C de Muller para detectar cuando TODAS las salidas son válidas o cuando NINGUNA de ellas lo es.

4 Máquinas de Control Asíncronas

Diferentes técnicas han sido propuestas para sintetizar circuitos de control asíncronos. Entre ellas destacan las basadas en la especificación de *tablas de flujo* siguiendo el modelo de Huffman [14], y los circuitos *self-clocked* con un reloj local que produce un pulso siempre que un estado o señales de salida deben modificarse [1].
La técnica que ha sido escogida para este trabajo son los STGs, introducidos en [15] y [16]. Un STG es una red de Petri [17] que modeliza el comportamiento de un circuito, y cuyas transiciones son interpretadas como cambios en las señales de entrada y salida del circuito. Las transiciones positivas y negativas de una señal t_i se indican t_i^+ y t_i^- respectivamente.

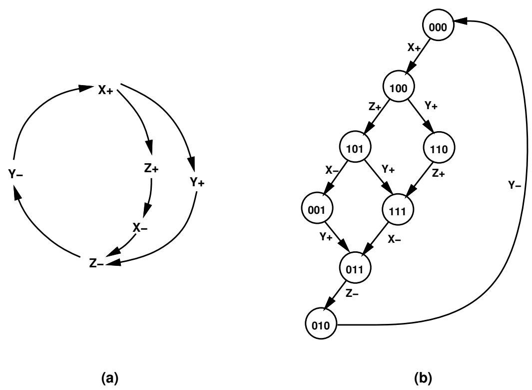

Figura 5: (a) Ejemplo de STG; (b) Grafo de Estados derivado del STG

El *grafo de estados* (SG) de un STG, es un *autómata finito*, en el cual los estados corresponden a los distintos marcajes del STG. Una arista etiquetada con t_i^+, desde el estado s_j hasta el estado s_k, indica que m_k puede ser *alcanzado* desde m_j disparando la transición t_i^+. La figura 5.a muestra un ejemplo de STG presentado en [18] con dos señales de entrada x e y, y una señal de salida z. La figura 5.b presenta el grafo de estados derivado del STG.
Los procedimientos de síntesis basados en STGs, usan las señales del circuito como señales de estado [18] [3] [4]. Dado un STG, cada estado s_j del SG tiene asignado un vector $v_j \in \{0,1\}^n$ acorde con las transiciones del STG (donde n es

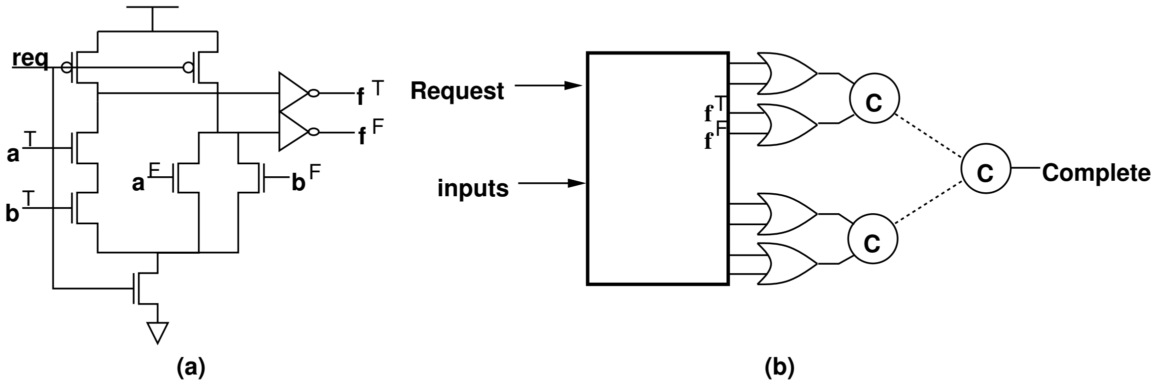

Figura 3: (a) Puerta AND DCVSL; (b) Detección de finalización con C de Muller

el número de señales en el circuito). Para una arista $s_j \longrightarrow s_k$ etiquetada t_i^+, la señal t_i debe valer 0 en v_j y 1 en v_k. Si la arista está etiquetada t_i^-, t_i debe valer 1 en v_j y 0 en v_k. Por el contrario, si no está etiquetada con ninguna transición de t_i, v_j y v_k deben tener el mismo valor para t_i.

El siguiente algoritmo deriva la función *siguiente-estado* (f_i) para cada señal t_i utilizando los códigos v_j calculados para cada estado.

Algoritmo:

`Sea` t_i `una señal de estado y` f_i `su función` *siguiente-estado*. `Sea` s_j `un estado del SG y` v_j `su código. Para cada estado` s_j `del SG hacer:`

> `(a)`
>
> `si hay una arista` $s_j \xrightarrow{t_i^+} s_k$ `entonces` $f_i(v_j) = 1$
>
> `(b)`
>
> `si hay una arista` $s_j \xrightarrow{t_i^-} s_k$ `entonces` $f_i(v_j) = 0$
>
> `(c) en otro caso` $f_i(v_j) = v_j^i$
>
> `(d) todo código` v_j `sin un estado en el SG` $f_i(v_j) =$ `x (`*don't care*`)`

La función *siguiente-estado* f_i definida por este algoritmo está *bien definida* si para cada código v_j, la función f_i está definida de forma unívoca, es decir, no existen dos estados s_j, s_k con el mismo código ($v_j = v_k$) de forma que $f_i(v_j) \neq f_i(v_k)$.

Si la función *siguiente-estado* está *bien definida* para todas las señales de estado, entonces el STG tiene la propiedad denominada *Complete State Coding* (CSC). CSC es una condición suficiente y necesaria en un STG para generar circuitos libres de riesgos. Si un STG no cumple la propiedad CSC, mediante la inserción de señales de estado, se puede obtener un nuevo STG con el mismo comportamiento que el original, pero que cumple la propiedad CSC [19][20]. Una vez calculada la función *siguiente-estado*, se obtiene una implementación aplicando técnicas de síntesis lógica [21][22].

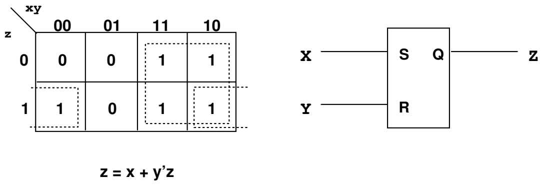

Figura 6: Circuito libre de riesgos de la señal z

Dada la implementación de una señal, existe un riesgo estático cuando se puede producir una transición ($0 \rightarrow 1 \rightarrow 0$), denominado 0-riesgo, o ($1 \rightarrow 0 \rightarrow 1$) 1-riesgo, cuando no estaba permitida ninguna transición de esa señal. Un riesgo dinámico ocurre cuando se producen múltiples transiciones ($0 \rightarrow 1$) ($1 \rightarrow 0$) y sólo estaba permitida una transición de esa señal. Todos los riesgos de un circuito deben ser eliminados, de otra forma se pueden producir comportamientos anómalos. Las técnicas existentes para eliminar los riesgos se basan en la utilización de lógica redundante y la inserción de retardos [23][24]. La figura 6 presenta una implementación libre de riesgos para la señal z del ejemplo.

5 Ejemplo

5.1 Descripción de comportamiento

Se quiere suavizar los valores m_i procedentes del muestreo de una determinada señal mediante un filtro que realiza la suma ponderada de tres valores, de forma que al valor central (m_i) se le da el doble de peso que a sus dos valores adyacentes. La fórmula a aplicar es la siguiente:

$$s_i = \frac{m_{i-1} + 2m_i + m_{i+1}}{4}$$

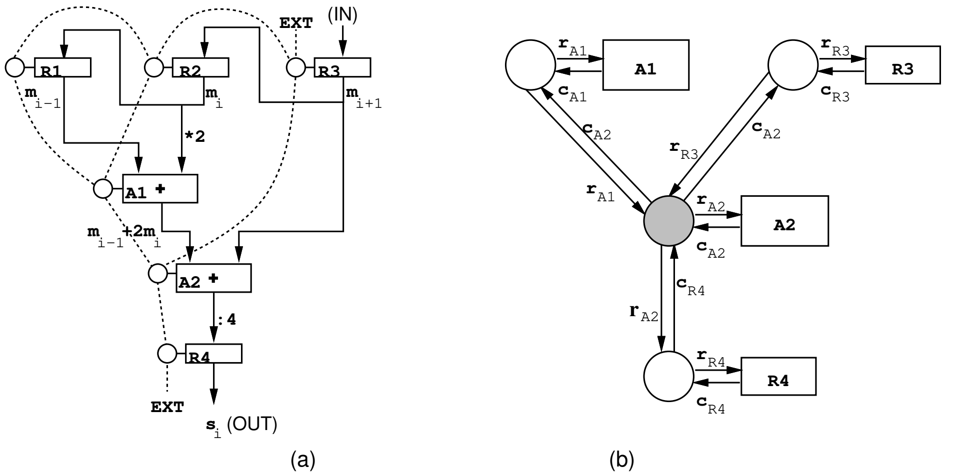

Figura 7: (a) Diagrama de bloques de la función s_i; (b) Señales para el controlador del sumador A2

5.2 Data-path

La figura 7.a muestra el esquema de bloques que implementa el filtro. Se utilizan los registros R1 a R4, y los sumadores A1 y A2. También está representada mediante líneas punteadas la comunicación entre las unidades de control de cada módulo.

5.3 Control

En un data-path síncrono únicamente necesitaríamos dar un tiempo de ciclo tan largo como el retardo del camino crítico. En cambio, si suponemos los componentes realizados con lógica DCVSL, el controlador local de cada módulo actuará según las señales que reciba de los otros controladores. Básicamente, el comportamiento de un bloque se puede resumir así:

- Se hace el *reset* del módulo (R^-) cuando sus sucesores han finalizado las operaciones, cosa que indicarán con la transición C^+. Al hacer el *reset*, las salidas pasan a ser todas cero, así que, gracias a la propiedad vista en la sección 3.3, los sucesores seguirán manteniendo los valores de salida que tenían.
- Se puede solicitar el *request* (R^+) en el momento en que tanto el propio módulo como los sucesores han completado su *reset* (lo indicarán con la transición C^-) y los predecesores han empezado a efectuar su operación (indicado con R^+). El hecho de activar la señal R únicamente permite que se efectúe el cálculo, pero no se obtendrán resultados válidos (complementados) de salida hasta que también lo sean los de la entrada.

Este modo de operar se puede representar mediante un STG. En la figura 8 está descrito el funcionamiento del controlador del sumador A2. La

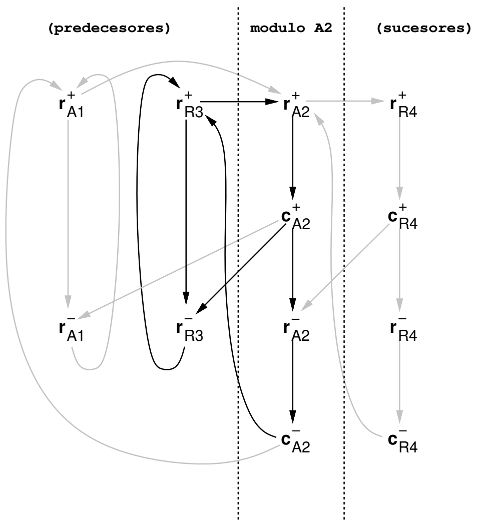

Figura 8: STG del controlador del sumador A2

unidad de control del sumador A2 se comunica con las de los módulos A1, R3 (predecesores) y R4 (sucesor), y el intercambio de señales entre estos módulos está representado gráficamente en la figura 7.b.
Este STG es un caso típico de un bloque con entradas de varios módulos. La comunicación predecesor-sucesor es idéntica para todos los módulos, por lo tanto, si un módulo tiene un mayor número de predecesores y/o sucesores, únicamente será necesario replicar los patrones de la figura 8 (dibujados en gris) tantas veces como sucesores o predecesores tenga.

5.4 Síntesis de los STG de control

El STG de la figura 8, con el marcaje inicial especificado, verifica las condiciones necesarias para su síntesis. A partir de él se puede derivar la función *siguiente-estado* de la señal r_{A2} ($f_{r_{A2}}$), que tiene por entradas r_{A1}, r_{R3}, c_{R4}. La ecuación de dicha función es

$$f_{r_{A2}} = r_{A1}\ r_{R3}\ \overline{c_{R4}} + (\overline{\overline{c_{R4}} + r_{R3} + r_{A1}})\ f_{r_{A2}}$$

La función $f_{r_{A2}}$ está *bien definida* y el STG cumple la propiedad *Complete State Coding*. Finalmente, aplicando algoritmos de síntesis libre de riesgos se obtiene el circuito detallado en la figura 9

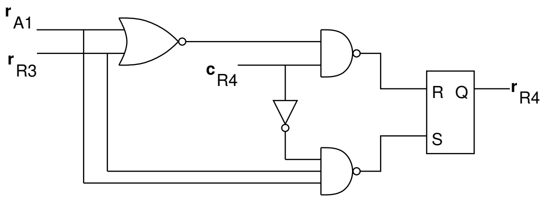

Figura 9: Circuito libre de riesgos para r_{A2}

6 Conclusiones

Este artículo ha presentado una metodología para automatizar la síntesis de circuitos asíncronos. Partiendo de una descripción en un lenguaje de alto nivel, supuestas las fases de *planificación* y *asignación de operaciones* han sido realizadas, y utilizando STGs, nos hemos centrado en la definición e implementación de los autómatas de control para unidades funcionales construidas con tecnología DCVSL.

A modo de ejemplo, se ha aplicado la metodología a la síntesis de un filtro para el procesado de señal, y se ha mostrado cómo se generan sus circuitos de control.

Referencias

[1] S. M. Nowick y D. L. Dill. "Synthesis of asynchronous state machines using a local clock". En *Proc. of the Int. Conf. on Computer Design*, páginas 192–197, 1991.

[2] K. Y. Yun, D. L. Dill, y S. M. Nowick. "Synthesis of 3d asynchronous state machines". En *Proc. of the Int. Conf. on Computer Design*, páginas 346–350, 1992.

[3] L. Lavagno, K. Keutzer, y A. Sangiovanni-Vincentelli. "Algorithms for synthesis of hazard-free asynchronous circuits". En *Proc. of the 28th. Design Automation Conference*, páginas 302–308, Junio, 1991.

[4] C. W. Moon, P. R. Stephan, y R. K. Brayton. "Synthesis of hazard-free asynchronous circuits from graphical specifications". En *Proc. of the Int. Conf. on Computer-Aided Design*, páginas 322–325, Noviembre, 1991.

[5] K. van Berkel. "Handshake circuits: an intermediary between communicating processes and VLSI". Phd thesis, Technische Universiteit Eindhoven, Mayo, 1992.

[6] A. J. Martin. "Compiling communicating processes into delay-insensitive VLSI circuits". *Distributing Computing*, Vol. 1:226–234, 1986.

[7] J. Cortadella, R. M. Badia, E. Pastor, y A. Pardo. "Achilles: A high-level synthesis system for asynchronous circuits". En *Sixth International Workshop on High-Level Synthesis*, páginas 307–311, Noviembre, 1992.

[8] R. M. Badia y J. Cortadella. "High-level synthesis of asynchronous systems: Scheduling and process synchronization". En *Proc. of the European Design Automation Conference*, Febrero, 1993.

[9] J. Cortadella y R. M. Badia. "An asynchronous architecture model for behavioral synthesis". En *Proc. of the European Design Automation Conference*, páginas 307–311, Marzo, 1992.

[10] C. L. Seitz. "Introduction to VLSI Systems", capítulo System Timing. Mead & Conway, Addison-Wesley, 1980.

[11] I. E. Sutherland. "Micropipelines". *Communications of the ACM*, 32(6):720–738, Junio, 1989.

[12] D. B. Armstrong, A. D. Friedman, y P. R. Manon. "Design of asynchronous circuits assuming unbounded gate delays". *IEEE Trans. on Computers*, Vol. C-18, Diciembre, 1969.

[13] L. G. Heller y W. R. Griffin. "Cascode voltage switch logic: A differential CMOS logic family". En *1984 IEEE ISSCC Digest of Technical Papers*, Febrero, 1984.

[14] S. Unger. "Asynchronous sequential switching circuits with unrestricted input changes". *IEEE Trans. on Computers*, Vol. C-20(12), Diciembre, 1971.

[15] V. I. Varshavsky. "Self-Timed Control of Concurrent Processes". Kluwer Academic Publishers, 1990.

[16] T.-A. Chu. "Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications". Ph.d. thesis, MIT, Junio, 1987.

[17] T. Murata. "Petri Nets: Properties, analysis and applications". *Proceedings of the IEEE*, Vol. 77(4):541–574, Abril, 1989.

[18] T. H.-Y. Meng, R. W. Brodersen, y D. G. Messerschmitt. "Automatic synthesis of asynchronous circuits from high-level specifications". *IEEE Trans. on CAD*, Vol. 8(11):1185–1205, Noviembre, 1989.

[19] L. Lavagno, C. W. Moon, R. K. Brayton, y A. Sangiovanni-Vincentelli. "Solving the state assignment problem for signal transition graphs". En *Proc. of the 29th. Design Automation Conference*, Junio, 1992.

[20] E. Pastor y J. Cortadella. "An efficient unique state coding algorithm for signal transition graphs". En *Proc. of the Int. Conf. on Computer Design*, Octubre, 1993.

[21] R. K. Brayton, G. D. Hachtel, C. T. McMullen, y A. L. Sangiovanni-Vincentelli. "Logic Minimization Algorithms for VLSI Synthesis". Kluwer Academic Publishers, 1984.

[22] E. Pastor y J. Cortadella. "Polynomial algorithms for the synthesis of hazard-free circuits from signal transition graphs". En *Proc. of the Int. Conf. on Computer-Aided Design*, Noviembre, 1993.

[23] E. B. Eichelberger. "Hazard detection in combinational and sequential switching circuits". *IBM Journal*, Marzo, 1965.

[24] L. Lavagno y A. Sangiovanni-Vincentelli. "Linear programming for optimum hazard elimination in asynchronous circuits". En *Proc. of the Int. Conf. on Computer Design*, páginas 275–278, 1992.