Programación VLSI y síntesis de circuitos asíncronos mediante composición de redes de Petri *

Marco A. Peña y Jordi Cortadella
Department d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
08071 - Barcelona

Resumen

Tangram es un lenguaje de programación VLSI traducible automáticamente a redes de módulos asíncronos llamados componentes de sincronización. *En este artículo se usan los* Grafos de Transiciones de Señales *(*STG*s) para describir el comportamiento de estos componentes, y se presentan métodos de recombinación de las descripciones, para generar un único* STG *equivalente para todo el sistema. Se muestra también cómo la eliminación de eventos internos resultantes de la composición, da lugar a* STG*s más sencillos con comportamiento equivalente. Las simplificaciones en los* STG*s generados permitirán la síntesis automática de circuitos eficientes en área.*

1 Introducción

Este artículo describe un sistema de síntesis automática de circuitos asíncronos VLSI, que combina un entorno de diseño de alto nivel, junto con técnicas de síntesis basadas en redes de Petri (PNs) [1]. El uso de descripciones de alto nivel facilita el diseño, mantiene la independencia de la tecnología y reduce el número de errores. Otros autores han desarrollado conceptos similares basándose en lenguajes concurrentes [2], *álgebras insensibles a retardos* [3], o los "*handshake circuits*" [4], entre otros.

Este trabajo utiliza Tangram en conjunción con los *Grafos de Transiciones de Señales* [5, 6], para describir el *comportamiento* de los *componentes de sincronización* usados en la traducción. Junto a ello se han desarrollado métodos automáticos de *composición* de STGs, que recombinan las especificaciones de los componentes y logran eliminar la lógica redundante implícita. El resultado es un único STG que describe todo el circuito, que se sintetizará con herramientas de CAD existentes.

El propósito es que los circuitos procedentes de la traducción automática de Tangram requieran menor coste en área de manera directa, sin recurrir al estudio de optimizaciones particulares (no siempre aplicables) para ciertos patrones de traducción.

*Subvencionado por CYCIT TIC 95-0419 y Departament d'Ensenyament de la Generalitat de Catalunya

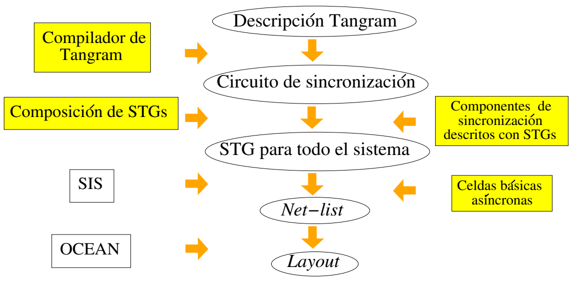


Figura 1: Esquema del proceso de traducción

2 Visión general

La Fig. 1 muestra el esquema global de nuestro proceso de traducción de Tangram hasta el circuito VLSI final. Sombreadas se muestran las herramientas y librerías desarrolladas en este trabajo. Destaca el compilador de Tangram a *circuitos de sincronización*, las herramientas de composición de STGs y la librería de componentes de sincronización descritos con STGs [7]. Cabe resaltar el uso de SIS para la síntesis automática a partir de STGs [8], y OCEAN para la generación y simulación del *layout* [9].

3 Tangram y circuitos de sincronización

En Tangram un circuito se describe como un conjunto de procesos secuenciales que operan concurrentemente y se comunican a través de canales. Un programa Tangram se traduce de forma transparente en términos de *circuitos de sincronización*: una red de componentes asíncronos conectados por canales punto a punto. La interacción entre los *componentes de sincronización* se hace según un sencillo protocolo de intercambio de señales [7].

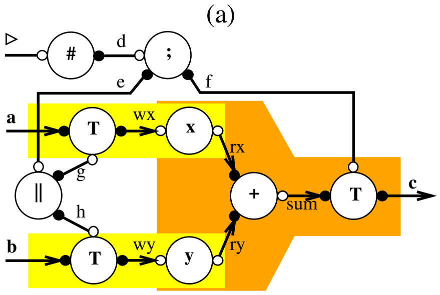
3.1 Tangram

Como lenguaje de programación VLSI, Tangram posee una serie de propiedades que lo hacen apto para el diseño de circuitos. Es de propósito general, ofrece un adecuado nivel de abstracción de la tecnología, y permite desarrollar con facilidad múltiples soluciones para una especificación dada.

Tangram, es similar a los lenguajes de programación habituales (Pascal, C, . . .), pero ofrece además

```
( a,b:? int act & c:! int act )
begin
   block
      x,y: var int
   action
      #[ (a?x || b?y); c!x+y ]
   fblock
end
```

(a)



(b)

Figura 2: Programa Tangram para un circuito sumador de enteros (a), y circuito de sincronización obtenido por traducción dirigida por la sintaxis (b)

particularidades propias de los lenguajes de comunicación de procesos como CSP [10]: declaración de puertos para la interacción con el entorno, composición secuencial y concurrente de procesos, comunicación y sincronización entre ellos, etc.

El programa de la Fig. 2 (a) describe un circuito que recibe dos valores a través de los puertos de entrada **a** y **b**, y emite su suma por **c**. El programa consta de un *comando de bloque* que introduce las variables x e y, y define su ámbito de validez. El comando en el bloque indica la infinita repetición de dos acciones de entrada `a?x` y `b?y` en *paralelo*, seguidas *secuencialmente* por la acción de salida `c!x+y`.

Tangram ofrece diversos comandos elementales como la asignación ('$x := E$', donde E es una expresión), entrada ('$a?x$') y salida ('$a!E$') de datos, y la sincronización entre comandos conectados a un canal ('$a \sim$'). Además ofrece mecanismos para construir nuevos comandos a partir de otros existentes: como la composición secuencial ('$C_1; C_2$') y paralela ('$C_1 \parallel C_2$'), la repetición infinita ('$\#[C]$'), el comando de elección **case**, así como los comandos **if**, **do** y **sel** basados en *acciones protegidas* [11], entre otros.

3.2 Circuitos de sincronización

La sincronización entre procesos a través de un canal se realiza según un protocolo asíncrono de intercambio de señales (petición/respuesta), de 4 fases[1] (Fig. 3). Los procesos de Tangram se implementan como redes de un reducido conjunto de componentes que interaccionan en estos términos (aproximadamente existe un componente de sincronización para cada concepto primitivo del lenguaje). La interfaz de estos componentes consiste en una serie de *puertos* activos o pasivos dependiendo de su papel en el intercambio de señales. Tales puertos pueden ser "no dirigidos" cuya función es la sincronización

[1] El uso de 2 fases es igualmente posible.

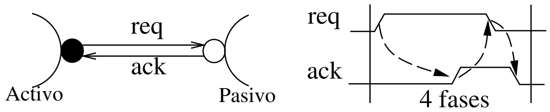


Figura 3: Canal de sincronización entre procesos y protocolo de 4 fases

(como en los canales g y h de la Fig. 2), o "dirigidos" si se usan para la entrada/salida de datos (como en los canales wx y wy, por ejemplo).

3.3 De Tangram a componentes de sincronización

El primer paso en la traducción de Tangram a circuitos VLSI, es la transformación del programa en un circuito de sincronización equivalente. Nuestro compilador [7] utiliza traducción *dirigida por la sintaxis*, donde cada construcción sintáctica se transforma de manera directa en un subcircuito. Esta relación entre programa y circuito permite el diseño abstracto, así como mantener control sobre la eficiencia del circuito desde el nivel de programa.

La Fig. 2 (b) muestra la traducción para el programa sumador, donde se aprecia cómo el circuito refleja la estructura de la descripción original. El circuito consta de 9 componentes de sincronización, 10 canales (d, e, f, $\ldots$) y 4 puertos (**a**, **b**, **c** y ▷ de activación del circuito). Los canales conectan un puerto pasivo (círculo blanco) con un puerto activo (círculo negro). La comunicación a través de ellos (Fig. 3) la inicia la parte activa con la señal de petición ($req \uparrow$). La parte pasiva responde con la señal de aceptación al concluir el cálculo ($ack \uparrow$). Finalmente ambas líneas vuelven al estado de reposo ($req \downarrow$, $ack \downarrow$).

El componente etiquetado ';' es un *sequencer*. Activado por d realiza *sincronizaciones* secuencialmente a través de e y f, antes de responder con la aceptación a través de d. El comportamiento del *parallelizer* ('$\parallel$') es ovbio.

El *repeater* ('#') implementa la infinita sucesión de sincronizaciones a través del puerto d, una vez activado por ▷. Nunca retornará la señal de aceptación.

El componente 'x' es una variable. La escritura se realiza enviando un valor por el canal wx. La señal de aceptación en este canal finaliza la acción. Una petición por rx inicia la acción de lectura, que concluye con el envío por el canal del valor almacenado.

La función del *transferrer* etiquetado 'T' es transferir un dato desde el entorno al interior del circuito ('$\mathbf{a} \rightarrow wx$', '$\mathbf{b} \rightarrow wy$') o al revés ('$sum \rightarrow \mathbf{c}$').

La Fig. 4 muestra mediante un Diagrama de Tiempos, el protocolo de 4 fases en el comportamiento de un *parallelizer*. Se observa la concurrencia en los eventos, los puntos de sincronización, etc. Las señales subrayadas corresponden a señales de entrada.

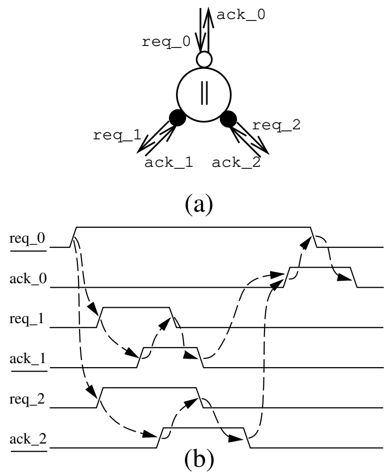


Figura 4: *Parallelizer*: (a) interfaz, (b) comportamiento expresado con un *Diagrama de Tiempos*

4 Descripción de componentes de sincronización con STGs

La manera más directa de obtener el circuito final, sería implementando cada uno de los componentes de sincronización, y uniendo los diversos módulos resultantes [4]. En este trabajo especificamos el comportamiento de dichos componentes con STGs y desarrollamos mecanismos de composición para ellos, de forma que las descripciones se compartan y recombinen entre sí. El objetivo es generar un único STG simplificado con comportamiento equivalente para todo el sistema, que sintetizaremos con herramientas de CAD. El circuito obtenido será menos costoso en área gracias a las optimizaciones globales a nivel de STGs.

Una red de Petri es una tupla $N = \langle P, T, F, M_o \rangle$, donde P es el conjunto de *lugares*, T es el conjunto de *transiciones*, $F \subseteq (P \times T) \cup (T \times P)$ es la *relación de flujo* (arcos) y M_o es el *marcado* (estado) inicial. Dado un nodo $x \in P \cup T$, los conjuntos de *predecesores* y *sucesores* de éste se representan por ${}^\bullet x$ y $x^\bullet$ respectivamente. Una transición $t \in T$ está *sensibilizada* en un marcado M ($M[t\rangle$), cuando todos los lugares en ${}^\bullet t$ están marcados. Al *dispararse* una transición sensibilizada en M, se elimina una marca de los lugares en ${}^\bullet t$ y se añade una marca a los lugares en $t^\bullet$, para alcanzar un nuevo marcado M' ($M[t\rangle M'$).

Un Grafo de Transiciones de Señales es la tripleta $G = \langle N, S, \Lambda \rangle$, donde N es una PN, $S = I \cup O \cup H$ es el conjunto de señales (entrada, salida e internas), y $\Lambda : T \longrightarrow S \times \{+, -\}$ es la función de etiquetado, donde las transiciones se interpretan como cambios de valor en las señales del circuito. Las transiciones positivas y negativas de la señal a se representan por a_+ y a_- respectivamente. a_* representa una transición genérica. Distinguiremos las múltiples transiciones de una señal mediante índices ($a_{1*}, a_{2*}, \ldots$).

Al implementar los componentes de sincronización con STGs, se ha seguido el protocolo de 4 fases. En la codificación de datos se ha empleado el código au-

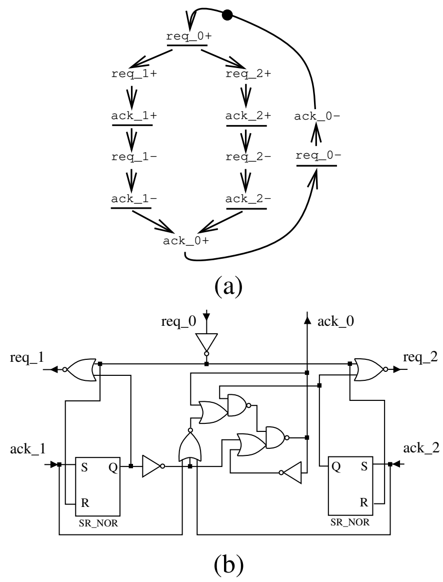


Figura 5: *Parallelizer*: (a) STG para su comportamiento, y (b) circuito obtenido por síntesis automática

tosincronizado de "*doble-vía*" [12]. La Fig. 4 muestra la interfaz detallada y el comportamiento del componente *parallelizer*. Se observa que cada canal de sincronización requiere dos hilos: petición (*req*) y aceptación (*ack*). El STG correspondiente y la implementación obtenida con SIS [8] se muestra en la Fig. 5. La marca en el arco '$ack\_0_- \longrightarrow req\_0_+$' indica el estado inicial.

5 Composición de STGs

En otros trabajos sobre la composición de STGs para la síntesis de circuitos asíncronos, los métodos presentados requieren un coste cuadrático en el número de transiciones ([13], [14]). Tal coste se debe al objetivo de mantener la *equivalencia de las trazas* de eventos, tanto si son observables externamente como si son internos.

Nuestro objetivo es la síntesis de un circuito asíncrono, por lo que sólo nos interesa la *equivalencia de las trazas* de eventos externos, que corresponden a las señales de entrada y salida del circuito. Esto nos permitirá realizar la composición de los STGs con coste lineal en el número de transiciones, sin restringirnos a un tipo de redes de Petri concreto. Y más aún, permitirá eliminar eventos internos resultantes de la composición, con lo que el nuevo STG se simplificará y los circuitos que se obtengan requerirán menos área.

Al componer dos STGs $G_1 = \langle N_1, S_1, \Lambda_1 \rangle$ y $G_2 = \langle N_2, S_2, \Lambda_2 \rangle$ para componentes de sincronización, el primer paso es identificar los puertos implicados en la conexión. De ello resultan las señales en *correspondencia*. Las señales $a \in S_1$ y $b \in S_2$ están en *correspondencia* ($a \; \Xi \; b$) si en la conexión de los circuitos para cada componente ambas estarían conec-

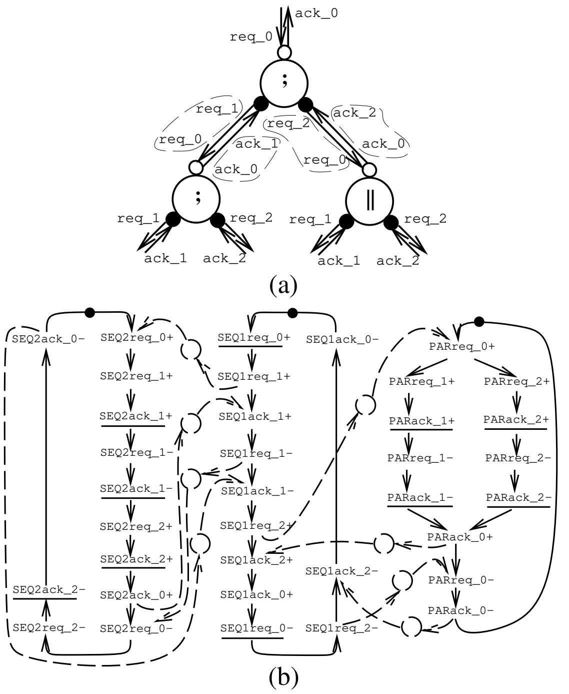


Figura 6: Interconexión de componentes de sincronización: (a) esquema de componentes; (b) STG equivalente por "Inserción de Lugares"

tadas (sólo se admite el caso en que, o bien $\mathsf{a} \in O_1$ y $\mathsf{b} \in I_2$, o bien $\mathsf{a} \in I_1$ y $\mathsf{b} \in O_2$). Definiremos el conjunto $SC_{1/2} \subseteq S_1$ como el conjunto de señales de G_1 en *correspondencia* con señales de G_2 (análogamente se define $SC_{2/1} \subseteq S_2$.

El conjunto de pares de eventos en *correspondencia* se define como:

$$TC = \{(\mathsf{a}_+, \mathsf{b}_+), (\mathsf{a}_-, \mathsf{b}_-) \mid \mathsf{a} \in O_1 \cup O_2 \wedge \mathsf{b} \in I_1 \cup I_2 \wedge \mathsf{a} \,\Xi\, \mathsf{b}\}$$

5.1 Inserción de lugares

Es un método de composición intuitivo que explicita en el nuevo STG la relación causal: "una transición de una señal de salida en uno de los componentes, provocará una transición en la señal de entrada en *correspondencia*, del otro componente". La Fig. 6 muestra la aplicación de esta idea. El nuevo STG mantiene la estructura de los originales y añade nuevos lugares entre los eventos en *correspondencia*. Las señales en *correspondencia* pasan a ser internas.

La composición consiste en la inserción de un lugar p para establecer la relación causal entre los eventos en *correspondencia*. Así, se definen los nuevos conjuntos de lugares:

$$P' = \bigcup_{(\mathsf{x}_*, \mathsf{y}_*) \in TC} \{p_{x*y*}\}$$

y de relaciones de flujo:

Interna	Eliminada
$SEQ1req\_1$	$SEQ2req\_0$
$SEQ1ack\_1$	$SEQ2ack\_0$
$SEQ1req\_2$	$PARreq\_0$
$SEQ1ack\_2$	$PARack\_0$

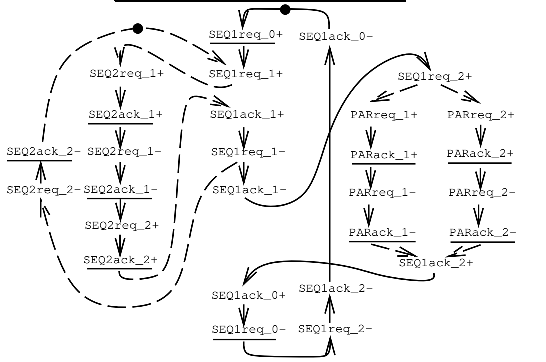


Figura 7: Composición por "Identificación de Eventos". STG para el ejemplo anterior

$$F' = \{(\mathsf{x}_{i*} \to p_{x*y*}), (p_{x*y*} \to \mathsf{y}_{j*}) \mid (\mathsf{x}_*, \mathsf{y}_*) \in TC\}$$

El nuevo STG resultante de la composición $G = \langle N, S, \Lambda \rangle$, donde $N = \langle P, T, F, M_o \rangle$, se define como:

$T = T_1 \cup T_2$
$P = P_1 \cup P_2 \cup P'$
$F = F_1 \cup F_2 \cup F'$
$M_o = M_{o1} \cup M_{o2}$
$S = I \cup O \cup H$, donde
$\quad I = (I_1 \setminus SC_{1/2}) \cup (I_2 \setminus SC_{2/1})$
$\quad O = (O_1 \setminus SC_{1/2}) \cup (O_2 \setminus SC_{2/1})$
$\quad H = H_1 \cup H_2 \cup SC_{1/2} \cup SC_{2/1}$
$\Lambda = \Lambda_1 \cup \Lambda_2$

5.2 Identificación de Eventos

El método de composición anterior considera las transiciones ocurridas en los extremos del hilo de conexión como eventos distintos. Sin embargo, al unirse los subcircuitos para formar uno solo, tales eventos que ahora son internos, pueden asumirse como el mismo. Así, para cada par de señales en *correspondencia*, $\mathsf{a} \in S_1$ y $\mathsf{b} \in S_2$, tal que $\mathsf{a} \,\Xi\, \mathsf{b}$, una de ellas pasará a ser interna mientras que la otra se eliminará.

Supongamos que a pasa a ser interna y b se elimina. Entonces $\forall (\mathsf{a}_*, \mathsf{b}_*) \in TC$, todos los arcos hacia transiciones b_{j+} (b_{j-}) en G_2, cambiarán su destino en G hacia las transiciones a_{i+} (a_{i-}). Los arcos con origen en transiciones b_{j+} (b_{j-}) partirán ahora de las correspondientes a_{i+} (a_{i-}).

La aplicación de esta idea al ejemplo de la Fig. 6 (a) puede verse en la Fig. 7, donde se muestra el sencillo STG resultante y la tabla con la identificación de señales realizada. Las trazas de eventos externos expresadas por el nuevo STG son equivalentes a las obtenidas con el método anterior.

En [7] pueden encontrarse más detalles sobre la implementación de este método para casos complejos.

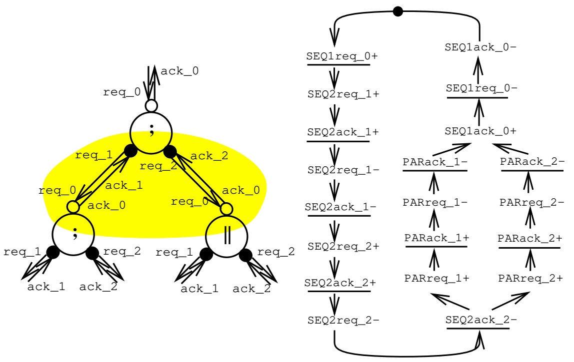


Figura 8: Simplificación del circuito elminando eventos internos a nivel de STG

Tabla 1: Comparación de resultados según el método de composición

	Transiciones	Lugares	Estados	Area
Conexión directa[2]	-	-	-	146
Inserción Lugares	36	37	52	186
Identificación Eventos	28	29	44	158
Eliminación señales	20	21	36	122

5.3 Eliminación de señales internas

En la síntesis de los STGs resultantes de la composición, la implementación de las nuevas señales internas incrementa innecesariamente el área del circuito final. Si se eliminan estas señales, el STG se simplifica considerablemente y se obtiene un circuito más simple y con menor coste en área, manteniendo el comportamiento observable por el entorno (Fig. 8).

Para eliminar estas señales se ha usado la herramienta '`petrify`' [15], basada en la obtención simbólica del *Grafo de Estados* a partir de una PN y su posterior transformación en un "*Elementary Transition System*" (ETS). A partir de él puede extraerse una nueva PN con el mínimo número de transiciones, que mantiene el isomorfismo del *Grafo de Alcanzabilidad* respecto al ETS original. Dada su generalidad, esta herramienta también permite realizar otras transformaciones y simplificaciones en los STGs resultantes.

5.4 Comparación

En la Tabla 1 pueden verse los resultados obtenidos para el ejemplo de la Fig. 6 (a) con los métodos presentados. Se observa la reducción de área obtenida por la combinación de la composición con la eliminación de eventos internos, ya que se comparte lógica entre los componentes y se reduce la complejidad del STG.

6 Resultados

La Tabla 2 muestra resultados obtenidos en la síntesis de varios circuitos. En cada caso se ha partido de una descripción mediante un programa Tangram, y se han aplicado los procesos descritos (Fig. 1) hasta la obtención del circuito final. Los ejemplos, aunque sencillos, permiten observar la reducción en área obtenida con nuestro método. Entre ellos destacan: un servidor de un recurso compartido en un sistema de Exclusión Mutua Distribuida (DME) en anillo (se muestran 3 versiones de diversa complejidad), una FIFO de tres posiciones, un filtro de Sobel para la extracción de contornos en imágenes, una célula de ordenación sistólica de bloques (BUBBLE) [4], un generador de C.R.C., etc.

Tabla 2: Resultados experimentales

	Cont. / Aritm.	Area TDS	STG Lug./Tran.	STG Area	Red. %
DME 1	19 / -	352	237 / 149	296	16
DME 2	55 / -	1798	932 / 588	1438	20
DME 3	32 / -	806	456 / 280	660	18
3-FIFO	20 / -	390	234 / 143	332	15
Sobel filter	12 / 31	700	136 / 177	594	15
Bubble	20 / 20	826	400 / 323	694	16
C.R.C.	52 / -	2830	1085 / 670	2066	27
Comparator	40 / 13	1338	810 / 647	942	30
M.c.d.	10 / 25	282	186 / 155	234	17
Mean filter	29 / 21	1852	675 / 549	1390	25

Los datos de los STGs y circuitos obtenidos, corresponden únicamente a la parte de control y manejo de datos booleanos (1 bit). Esto es así porque los componentes aritméticos no se han diseñado con STGs, sino que se han usado implementaciones eficientes ya existentes. La primera columna muestra el número de componentes de sincronización (control-booleanos y aritméticos) que forman el circuito tras la traducción dirigida por la sintaxis (TDS). En la segunda columna se muestra el área (transistores) requerida por la conexión directa (TDS) de los subcircuitos para cada uno de los componentes. La tercera columna muestra los datos del STG obtenido por composición (lugares y transiciones), así como el área del circuito generado a partir de él. Finalmente en la última columna se muestra el porcentaje de reducción obtenido para los diversos ejemplos, que oscila entre un 15 y un 30 %.

Como se ha visto, el proceso descrito genera un único STG para todo el sistema, a partir del cual se obtiene el circuito mediante síntesis automática. Las herramientas de síntesis a partir de STGs utilizan como paso intermedio el Grafo de Estados, lo cual implica un coste exponencial que limita el tamaño de los circuitos sintetizables. Por esta razón en algunos ejemplos, se ha recurrido al *particionado manual* de la red de componentes de sincronización, reduciendo así la complejidad del problema. El circuito final se ha obtenido conectando los subcircuitos generados para cada partición. Esto significa que los datos de área obtenidos podrían mejorarse si las herramientas de síntesis admitiesen redes de Petri mayores.

7 Conclusiones

Se ha descrito un modelo para la traducción del lenguaje Tangram a redes de Petri.

En primer lugar, hemos visto como traducir una

[2] Conexión de los subcircuitos para cada componente. El número de transistores corresponde a la suma de los de cada uno de ellos: *sequencer* (34) y *parallelizer* (78).

descripción Tangram en una red *insensible a retardos* de módulos asíncronos, llamados *componentes de sincronización*, que interaccionan entre sí a través de canales punto a punto.

El comportamiento de dichos componentes y su interacción mediante el protocolo de sincronización de 4 fases, se ha descrito con redes de Petri (concretamente STGs), obteniéndose una librería de descripciones [7].

Hemos presentado métodos de composición automática de redes de Petri con coste lineal respecto al número de transiciones. Dichos métodos se han aplicado a la conexión de los componentes de sincronización, lo que permite obtener un STG con comportamiento equivalente a la descripción original en Tangram.

Combinando la composición con la eliminación de eventos internos, se logra la recombinación de las descripciones, se comparte lógica común y se eliminan redundancias y transiciones no observables. Esto da lugar a STGs simplificados para todo el sistema, a partir de los cuales se obtienen automáticamente circuitos más sencillos en términos de área.

Todo el proceso de obtención de una red de Petri a partir de una descripción Tangram, se ha implementado en un conjunto de herramientas [7].

El trabajo futuro se centrará en resolver el problema del particionado automático de las redes de componentes de sincronización, de forma que para cada partición pueda obtenerse un STG manejable por las herramientas de síntesis actuales. Por otro lado, está previsto extender los métodos descritos a otros lenguajes de descripción como CSP.

Referencias

[1] T. Murata. "Petri Nets: Properties, Analysis and Applications". *Proceedings of the IEEE*, 77(4):541–574, Abril, 1989.

[2] A. J. Martin. "Programming in VLSI: From communicating processes to delay-insensitive circuits". En C. Hoare, editor, *Developments in Concurrency and Communications*, páginas 1–64. UT Year of Programming Series, Addison-Wesley, 1990.

[3] M. Josephs y J. Udding. "Delay -insensitive circuits: An algebraic approach to their design". En J. Baeten y J. Klop, editors, *Theories of Concurrency: Unification and Extension, volume 458 of Lecture Notes in Computer Science*, páginas 342–366. CONCUR'90, Springer-Verlag, Agosto, 1990.

[4] K. van Berkel. "Handshake Circuits: an Asynchronous Architecture for VLSI Programming", volumen 5 de *International Series on Parallel Computation*. Cambridge University Press, 1993.

[5] L. Rosenblum y A. Yakovlev. "Signal graphs: From self-timed to timed ones". En *International Workshop on Timed Petri Nets*, páginas 199–206, 1985.

[6] T. Chu. "Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications". Tesis Doctoral, MIT, Junio, 1987.

[7] M. A. Peña. "Síntesis de circuitos asíncronos mediante la traducción de circuitos de sincronización a redes de Petri". Tesina, Facultat d'Informàtica de Barcelona (UPC), Barcelona, Spain, Abril, 1995.

[8] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, y A. Sangiovanni-Vincentelli. "SIS: A system for sequential circuit synthesis". Technical Report UCB/ERL M92/41, U.C. Berkeley, Mayo, 1992.

[9] P. Groeneveld y P. Stravers. "Ocean: the Sea-of-Gates Design System". Report de investigación, Delft University of Technology, faculty of Electrical Engineering, Noviembre, 1993.

[10] C. Hoare. "Communicating Sequential Processes". Prentice Hall International, 1989.

[11] E. Dijkstra. "Guarded Commands, Nondeterminacy and Formal Derivation of Programs". *Communications of the ACM*, 18(8):453–457, Agosto, 1975.

[12] C. L. Seitz. "System timing". En C. Mead y L. Conway, editors, *Introduction to VLSI Systems*, capítulo 7. Addison-Wesley, 1981.

[13] G. de Jong y B. Lin. "A Communicating Petri Net Model for the Design of Concurrent Asynchronous Modules". En *Proc. ACM/IEEE Design Automation Conference*, páginas 49–55, Junio, 1994.

[14] I. Reicher y M. Yoeli. "Net-based modeling of communicating parallel processes with applications to VLSI design". Report de investigación 532, Technion-Israel Institute of Technology, Diciembre, 1988.

[15] J. Cortadella, M. Kishinevsky, L. Lavagno, y A. Yakovlev. "Synthesizing Petri Nets from State-Based Models". En *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD'95)*, Noviembre, 1995.