Verificación de circuitos independientes de la velocidad con modelos simbólicos de redes de Petri *

Oriol Roig, Enric Pastor y Jordi Cortadella
Dept. d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
08071 - Barcelona

Resumen

Este artículo presenta una metodología para verificar circuitos independientes de la velocidad a partir de su especificación descrita mediante redes de Petri. Se utilizan técnicas basadas en análisis simbólico, en las que tanto el circuito como su especificación se modelan como álgebras booleanas, lo que permite enumerar sus estados utilizando funciones booleanas. Estas funciones se representan de forma compacta mediante Diagramas Binarios de Decisión (BDD). Métodos de síntesis basados en STGs pueden ser verificados directamente, ya que utilizan el mismo formalismo que la técnica de verificación aquí presentada. Nuestra metodología ha sido aplicada a varios ejemplos, probando así su utilidad.

1 Introducción

Durante estos últimos años, los circuitos asíncronos han ganado interés debido a sus prometedoras ventajas, tales como la eliminación de los problemas derivados de un reloj global, circuitos potencialmente más rápidos y que consumen menos, o un alto grado de modularidad. En particular se han estudiado dos clases de circuitos asíncronos: circuitos *independientes de la velocidad*, cuyo correcto funcionamiento no depende de los retardos de las puertas, suponiendo insignificantes los de las conexiones, y circuitos *insensibles a los retardos*, que funcionan correctamente para cualquier retardo acotado de sus conexiones. Hay diversas técnicas de síntesis de circuitos asíncronos, tales como modelos de álgebras de procesos basados en descripciones en CSP (*Communicating Sequential Processes*) [1], modelos de eventos usando redes de Petri [2], y modelos basados en grafos de estados [3].

A pesar de estas técnicas, el diseño de circuitos asíncronos es aún complejo. Un mayor obstáculo para el uso común de circuitos asíncronos lo representa la dificultad de verificarlos, ya que existe la necesidad de probar que las estrategias de diseño generan circuitos que satisfacen la especificación original. Las métodos clásicos utilizados para circuitos síncronos, tales como la simulación exhaustiva, son impracticables debido al gran número de posibles patrones de entrada y a las relaciones temporales entre ellos.

Diversos autores han estudiado el problema de la verificación de circuitos asíncronos. Dill et al. han usado la *teoría de trazas* y la *lógica temporal* [4, 5] para verificar el comportamiento y propiedades del circuito. Moon et al. verifican que el lenguaje generado por el circuito está incluido en el que genera su especificación [6]. Burch et al. han aplicado técnicas basadas en BDDs para representar simbólicamente circuitos y sus grafos de estados y los han verificado contra especificaciones en CTL (*Computation Tree Logic*) [7, 8]. Más recientemente, McMillan [9] ha verificado circuitos asíncronos modelando circuito y entorno como redes de Petri, y evitando el problema de la explosión de estados mediante el desplegado de dichas redes. Finalmente, Kishinevsky et al. utilizan *Change Diagrams* [10] para verificar que sus circuitos son estructuralmente independientes de la velocidad.

Este artículo presenta una técnica completamente automática para verificar circuitos asíncronos basada en la comparación de modelos simbólicos. La especificación está descrita mediante *redes de Petri* [11], ya que son un formalismo general y potente para describir el comportamiento de los circuitos asíncronos. Permiten expresar secuencialidad y concurrencia, elección determinista o no, y también propiedades como la seguridad, vivacidad o ecuanimidad con diferentes niveles de abstracción. El circuito (descrito a nivel de puertas) y su entorno (implícitamente descrito por la red de Petri) se modelan como álgebras booleanas. Las propiedades y el comportamiento de ambos se verifican eficientemente usando técnicas simbólicas basadas en BDDs. Circuito y entorno forman un sistema cerrado, en el que los estados alcanzables se generan simbólicamente, y se verifica que todos los posibles comportamientos del circuito están incluidos en su especificación.

Se han utilizado diferentes ejemplos escalables para mostrar la eficiencia de la técnica aquí pro-

*Subvencionado por CYCIT TIC 91-1036, ACiD-WG (Esprit 7225) and Dept. d'Ensenyament de la Generalitat de Catalunya

puesta. Sin embargo, la regularidad inherente de estos ejemplos no se ha utilizado para simplificar el proceso de verificación, ya que nuestra intención es verificar circuitos en los que no tiene por qué existir regularidad. Por ello los circuitos han sido considerados como un conjunto de puertas sin jerarquía ninguna.

Este artículo está organizado como sigue. En la sección 2 se presenta el isomorfismo entre redes de Petri y álgebras booleanas. La sección 3 explica cómo se representan los circuitos, a nivel de puertas, con álgebras booleanas. La sección 4 muestra la composición de circuito y entorno, los algoritmos para realizar el análisis de la alcanzabilidad, así como las condiciones para detectar errores en la implementación del circuito. A modo de ejemplo, dos implementaciones diferentes de un árbitro asíncrono se verifican en la sección 6. Finalmente se presentan algunas conclusiones.

2 Modelado de redes de Petri seguras mediante álgebras booleanas

Sea $N = \langle P, T, F, m_0 \rangle$ una red de Petri, donde P es el conjunto de lugares, T el conjunto de transiciones, $F \subseteq (P \times T) \cup (T \times P)$ la relación de flujo, y m_0 es el marcado inicial. Los símbolos ${}^\bullet t$ y $t^\bullet$ denotan, respectivamente, los conjuntos de lugares de entrada y salida de la transición t. Una transición está *sensibilizada* cuando todos sus lugares predecesores están marcados. Utilizamos $m_1[t\rangle m_2$ para denotar que el marcado m_2 es alcanzable desde m_1 tras disparar la transición t. El conjunto de todos los marcados alcanzables de N se denota por $[m_0\rangle$. Una introducción más completa a las redes de Petri puede encontrarse en [11].

Un marcado en $[m_o\rangle$ se puede representar mediante un conjunto de lugares m, donde $p_i \in m$ denota que p_i está marcado. Si M_P es el conjunto de todos los marcados de una red de Petri con n lugares ($n = |P|$, $|M_P| = 2^n$), el sistema

$$(2^{M_P}, \cup, \cap, \emptyset, M_P)$$

es el álgebra booleana de conjuntos de marcados. Este sistema es isomorfo con el álgebra booleana de funciones lógicas de n variables, y por tanto existe una correspondencia biunívoca entre marcados de M_P y vértices de B^n [12, 13].

Un marcado $m \in M_P$ está representado por una *función de codificación* $\mathcal{E} : M_P \rightarrow B^n$, donde la imagen de m está codificada como un vértice $(p_1, \ldots, p_n) \in B^n$, tal que:

$$p_i = \begin{cases} 1 & \text{si } p_i \in m \\ 0 & \text{si } p_i \notin m \end{cases} .$$

Cada conjunto de marcados $M \in 2^{M_P}$ tiene una *función característica* $\chi_M^{\mathcal{E}} : B^n \rightarrow B$, que evalúa a 1 para aquellos vértices que se corresponden con marcados en M. Por ejemplo, dada

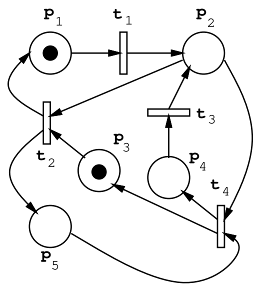

Figura 1: Red de Petri

la red de Petri de Fig. 1, la función característica del conjunto de marcados

$$M = \{ \{p_2, p_5\}, \{p_2, p_3, p_5\}, \{p_1, p_2, p_5\}, \{p_1, p_2, p_3, p_5\}, \{p_1, p_2, p_3, p_4, p_5\} \} ,$$

se calcula como la suma lógica de cada código booleano $\mathcal{E}(m), m \in M$. La función resultante

$$\chi_M = p_1 p_2 p_3 p_5 + p_2 p_4' p_5,$$

representa el conjunto de marcados donde p_1, p_2, p_3 y p_5 tienen marca, o bien p_2 y p_5 están marcados y p_4 no.

La estructura de la red de Petri define las reglas de disparo que determinan el comportamiento de la red. Definimos la *función de transición* de una transición como una función

$$\delta_N : 2^{M_P} \times T \rightarrow 2^{M_P},$$

que transforma, para cada transición t, el conjunto de marcados de M_1 donde t está sensibilizada en un nuevo conjunto de marcados en el que dicha transición ya ha sido disparada:

$$\delta_N(M_1, t) = \{m_2 \in M_P : \exists m_1 \in M_P,\ m_1[t\rangle m_2\} .$$

Este concepto es equivalente a la alcanzabilidad en un paso en redes de Petri, y se denomina *cálculo de la imagen* cuando se usan funciones booleanas.

El cálculo de la imagen para transiciones se puede realizar eficientemente usando la información topológica de la red de Petri. Para ello presentaremos las funciones características de algunos conjuntos importantes relacionados con la transición $t \in T$:

$$\begin{aligned}
\mathrm{E}_t &= \prod_{p_i \in {}^\bullet t} p_i && (t \text{ sensibilizada}), \\
\mathrm{NPM}_t &= \prod_{p_i \in {}^\bullet t} p_i' && (\text{ningún predecesor marcado}), \\
\mathrm{ASM}_t &= \prod_{p_i \in t^\bullet} p_i && (\text{todo sucesor marcado}), \\
\mathrm{NSM}_t &= \prod_{p_i \in t^\bullet} p_i' && (\text{ningún sucesor marcado}).
\end{aligned}$$

Dadas estas funciones características, el cálculo de la imagen para transiciones se reduce a calcular:

$$\delta_N(M, t) = (M_{\mathrm{E}_t} \cdot \mathrm{NPM}_t)_{\mathrm{NSM}_t} \cdot \mathrm{ASM}_t .$$

El cálculo de la imagen para toda la red de Petri se realiza de la siguiente forma:

$$\Delta_N(M) = \bigcup_{t \in T} \delta_N(M, t).$$

Usando el ejemplo de Fig. 1, calcularemos $M' = \delta_N(M, t_1)$ dado el conjunto

$$M = p_1 p_2' p_3 p_4' p_5' + p_1' p_2 p_3 p_4' p_5' + p_1 p_2' p_3' p_4' p_5.$$

Primero, $M_{\mathrm{E}_{t_1}}$ (cofactor de M respecto $\mathrm{E}_{t_1} = p_1$) selecciona aquellos marcados donde t_1 está sensibilizada y elimina sus lugares predecesores de la función característica:

$$M_{\mathrm{E}_{t_1}} = p_2' p_3 p_4' p_5' + p_2' p_3' p_4' p_5.$$

El producto por $\mathrm{NPM}_{t_1} = p_1'$ elimina las marcas de los lugares predecesores:

$$M_{\mathrm{E}_{t_1}} \cdot \mathrm{NPM}_{t_1} = p_1' p_2' p_3 p_4' p_5' + p_1' p_2' p_3' p_4' p_5.$$

A continuación, el cofactor respecto $\mathrm{NSM}_{t_1} = p_2'$ elimina todos los lugares sucesores:

$$(M_{\mathrm{E}_{t_1}} \cdot \mathrm{NPM}_{t_1})_{\mathrm{NSM}_{t_1}} = p_1' p_3 p_4' p_5' + p_1' p_3' p_4' p_5.$$

Finalmente, el producto por $\mathrm{ASM}_{t_1} = p_2$ añade una marca a los sucesores de t_1:

$$M' = p_1' p_2 p_3 p_4' p_5' + p_1' p_2 p_3' p_4' p_5.$$

3 Modelado de circuitos independientes de la velocidad

Un circuito no es más que un conjunto de nodos o señales, donde los nodos que no son entradas del circuito, son la salida de alguna puerta. Denotamos por S_I, S_O and S_H los conjuntos de señales de entrada, de salida e internas, respectivamente. En un circuito síncrono el estado está completamente especificado por las señales de entrada y algunos biestables. Por el contrario, el estado de un circuito independiente de la velocidad depende de todas sus señales. Como los circuitos independientes de la velocidad funcionan para cualquier retardo de sus puertas, el estado futuro del circuito depende únicamente del valor de sus señales en el presente estado, y no de los retardos que sufran dichas señales.

El conjunto de todos los posibles estados C_S de un circuito con v señales se modela simbólicamente como una *álgebra booleana*, en la que cada señal $s_i \in S$ ($S = S_I \cup S_O \cup S_H$) está representada por una variable. De ahora en adelante, usaremos s_i indistintamente para referirnos tanto a la señal como a la variable que la representa. El valor de las variables de entrada, internas y de salida determina el estado c del circuito:

$$c = (s_1, \ldots, s_v), \quad s_j \in \{0, 1\}, \; 1 \leq j \leq v \; .$$

Cada estado c se representa como un *minterm* de una función de v variables, y un conjunto de estados C como la suma de los *minterms* de esos estados.

El estado de una puerta está determinado por el valor de sus entradas $(s_1, \ldots, s_n)$ y su salida (s_k). Dada una puerta que calcula la función $s_k = f_{s_k}(s_1, \ldots, s_n)$, definimos sus *funciones de excitación positiva y negativa*, respectivamente, como:

$$\begin{array}{l} f_{s_k}^+(s_1, \ldots, s_n) = s_k' \cdot f_{s_k}(s_1, \ldots, s_n), \text{ y} \\ f_{s_k}^-(s_1, \ldots, s_n) = s_k \cdot f_{s_k}'(s_1, \ldots, s_n). \end{array}$$

Las funciones $f_{s_k}^+$ y $f_{s_k}^-$ representan las funciones características de aquellos estados en que la puerta puede realizar una transición $0 \rightarrow 1$ y $1 \rightarrow 0$, respectivamente. A continuación mostramos las funciones de excitación de una puerta AND y de un elemento C de Muller:

$$\begin{array}{ll} s_k = f_{s_k}(s_i, s_j) & = s_i \cdot s_j \\ \left\{ \begin{array}{l} f_{s_k}^+(s_i, s_j, s_k) \\ f_{s_k}^-(s_i, s_j, s_k) \end{array} \right. & \begin{array}{l} = s_k' \cdot s_i \cdot s_j \\ = s_k \cdot (s_i' + s_j') \end{array} \end{array}$$

$$\begin{array}{ll} s_k = f_{s_k}(s_i, s_j, s_k) & = s_k \cdot s_i + s_k \cdot s_j + s_i \cdot s_j \\ \left\{ \begin{array}{l} f_{s_k}^+(s_i, s_j, s_k) \\ f_{s_k}^-(s_i, s_j, s_k) \end{array} \right. & \begin{array}{l} = s_k' \cdot s_i \cdot s_j \\ = s_k \cdot s_i' \cdot s_j' \end{array} \end{array}$$

Definimos la *función de transición* de una señal interna o de salida como la función

$$\delta_C : 2^{C_S} \times (S_O \cup S_H) \longrightarrow 2^{C_S},$$

que dado un conjunto de estados y una señal calcula el nuevo conjunto de estados después de conmutar la puerta que controla s_k. El conjunto de estados alcanzables tras conmutar todas las señales del circuito se obtiene calculando:

$$\Delta_C(C) = \bigcup_{s_k \in S_O \cup S_H} \delta_C(C, s_k),$$

donde la función de transición de s_k se obtiene utilizando las funciones de excitación positiva y negativa:

$$\delta_C(C_1, s_k) = (C_1 \cdot f_{s_k}^+)_{s_k'} \cdot s_k + (C_1 \cdot f_{s_k}^-)_{s_k} \cdot s_k'.$$

Para ilustrar esto, calcularemos el nuevo conjunto de estados C_2 tras conmutar una puerta AND con entradas s_1 y s_3 y salida s_4, desde el conjunto $C_1 = s_1 s_2' s_3 s_4' s_5 + s_1' s_2 s_3 s_4 s_5' + s_1' s_2' s_3' s_4' s_5$. Para ello usaremos la función de transición $\delta_S(C_1, s_4)$. Los productos

$$\begin{array}{lll} C_1 \cdot f_4^+ = & C_1 \cdot s_4' s_1 s_3 & = s_1 s_2' s_3 s_4' s_5, \\ C_1 \cdot f_4^- = & C_1 \cdot s_4 (s_1' + s_3') & = s_1' s_2 s_3 s_4 s_5', \end{array}$$

evalúan los estados donde s_4 puede cambiar a 1 ó 0, respectivamente. El conjunto final de estados C_2 es la unión de los estados en que s_4 ha conmutado, y se calcula de la siguiente forma:

$$\begin{array}{ll} C_2 & = (C_1 \cdot f_4^+)_{s_4'} \cdot s_4 + (C_1 \cdot f_4^-)_{s_4} \cdot s_4' \\ & = (s_1 s_2' s_3 s_4' s_5)_{s_4'} \cdot s_4 + (s_1' s_2 s_3 s_4 s_5')_{s_4} \cdot s_4' \\ & = s_1 s_2' s_3 s_4 s_5 + s_1' s_2 s_3 s_4' s_5'. \end{array}$$

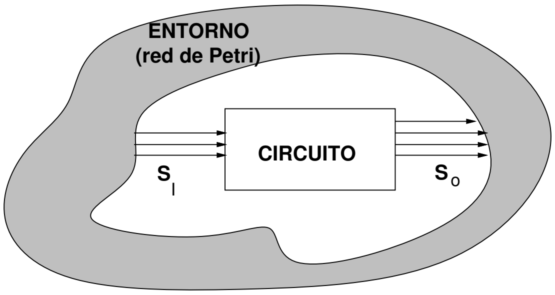

Figura 2: Sistema circuito-entorno

4 Composición de circuito y entorno

Tal como muestra Fig. 2, consideraremos el sistema formado por un circuito y la red de Petri que modela a la vez su comportamiento y su entorno. En dicho sistema las señales de entrada (S_I) y salida (S_O) del circuito tienen asociadas transiciones en la red de Petri. Dada una señal $s \in S_I \cup S_O$, denotamos por T_{s^+} (T_{s^-}) el conjunto de transiciones de la red de Petri que especifican una transición ascendente (descendente) de s.

El conjunto de estados Q_G del sistema circuito-entorno es un subconjunto del producto cartesiano de los conjuntos de estados de cada subsistema: $Q_G \subseteq 2^{M_P \times C_S}$. Sin embargo, el número de estados de Q_G es habitualmente menor que el de dicho producto, ya que el comportamiento de un subsistema queda restringido por el otro. El estado del sistema está definido por el par ordenado (m, c), donde m representa un marcado de la red de Petri y c un estado del circuito.

La red de Petri y el circuito deben comunicarse de forma síncrona. Esto significa que el circuito conmutará las señales de entrada cuando lo indique el entorno y, a su vez, que el entorno debe estar listo para aceptar los cambios de las puertas de salida del circuito. Así, cuando la red de Petri dispara una transición $t \in T_{s^+}$, con $s \in S_I$, al mismo tiempo la señal s pasará a valer 1. Cuando la señal $s \in S_O$ cambie a 1, todas las transiciones $t \in T_{s^+}$ sensibilizadas deberán dispararse. De igual modo se comportarán las transiciones descendentes. Las señales internas del circuito (S_H) pueden conmutar sin ningún tipo de sincronización con el entorno, ya que no tienen correspondencia en éste, y lo mismo sucede con las transiciones internas de la red de Petri, caso de que las haya.

En cualquier caso, el estado del sistema cambiará cuando conmute alguna señal o se dispare alguna transición. Las fórmulas para el cálculo de la imagen antes definidas, δ_N y δ_C, se pueden extender para usarlas en el sistema circuito-entorno:

$$\begin{aligned} \delta_N: &\quad 2^{Q_G} \times T \rightarrow 2^{Q_G} \\ \delta_C: &\quad 2^{Q_G} \times S \rightarrow 2^{Q_G} \end{aligned}$$

Ahora, δ_N puede ser utilizada para disparar las transiciones internas de la red de Petri y δ_C para conmutar las señales internas del circuito. Sin embargo, es necesario definir nuevas funciones de transición para las señales de entrada y salida del circuito, con el fin de "simular" la sincronización entre una señal s y los conjuntos T_{s^+} y T_{s^-}. Dichas funciones de transición se definen, respectivamente, como:

$$\begin{aligned} \delta_I: &\quad 2^{Q_G} \times S_I \rightarrow 2^{Q_G} \\ \delta_O: &\quad 2^{Q_G} \times S_O \rightarrow 2^{Q_G} \end{aligned}$$

Dado el conjunto de estados de Q donde s está excitada, δ_I y δ_O calculan un nuevo conjunto de estados en que s ha conmutado y una transición de $T_{s^+} \cup T_{s^-}$ se ha disparado. La función de transición para las señales de entrada se calcula como

$$\begin{aligned} \delta_I(Q_1, s_k) = &\left(\bigcup\nolimits_{t \in T_{s_k^+}} \delta_N(Q_1, t)_{s'_k} \cdot s_k \right) \cup \\ &\left(\bigcup\nolimits_{t \in T_{s_k^-}} \delta_N(Q_1, t)_{s_k} \cdot s'_k \right) , \end{aligned}$$

y para las señales de salida como

$$\delta_O(Q_1, s_k) = \bigcup_{t \in T_{s_k^+} \cup T_{s_k^-}} \delta_C(Q_1, s_k) \cdot \delta_N(Q_1, t) \ .$$

Cabe remarcar que en cada nuevo estado la transición disparada de los conjuntos T_{s^+} y T_{s^-} puede ser diferente.

El cálculo de la imagen para todas las señales de entrada y salida del circuito se obtiene así:

$$\begin{aligned} \Delta_I(Q) = &\ \bigcup\nolimits_{s_k \in S_I} \delta_I(Q, s_k) \ , \\ \Delta_O(Q) = &\ \bigcup\nolimits_{s_k \in S_O} \delta_O(Q, s_k) \ . \end{aligned}$$

Aquellos estados en que una señal de salida ($s \in S_O$) está positivamente (negativamente) excitada, mientras que ninguna transición $t \in T_{s^+}$ (T_{s^-}) está sensibilizada se denominan *estados erróneos*. Dichos estados modelan la situación en que el circuito genera una transición no esperada por el entorno, es decir, un *riesgo*. $\Delta_O(Q)$ está definida correctamente si Q no contiene estados erróneos, por lo que es necesario detectarlos antes de realizar el cálculo de la imagen. Estos errores en un conjunto de estados Q se detectan cuando se cumple alguna de las siguientes ecuaciones:

$$\begin{aligned} \bigcup_{t \in T_{s^+}} Q \cdot f_s^+ \cdot E'_t &\neq \emptyset \\ \bigcup_{t \in T_{s^-}} Q \cdot f_s^- \cdot E'_t &\neq \emptyset \end{aligned}$$

5 Recorrido del sistema

El problema de la comparación de modelos se resuelve calculando todos los estados alcanzables del sistema circuito-entorno, y comprobando que

atravesar_circuito_y_entorno (S, N) {
  $Alcanzados = Desde = estado\_inicial$;
  repetir {
    $Nuevos = propagar\_internas(Desde, S_H)$;
    $acabar\_si\_estados\_err\acute{o}neos(Nuevos, S_O)$;
    $Hasta = \Delta_I(Nuevos, S_I) \cup \Delta_O(Nuevos, S_O)$;
    $Desde = Hasta - Alcanzados$;
    $Alcanzados = Alcanzados \cup Nuevos \cup Hasta$;
  } hasta_que $(Desde = \emptyset)$;
}

Figura 3: Algoritmo para recorrer el sistema circuito-entorno

no se producen estados erróneos. Entonces podemos decir que el circuito es una implementación independiente de la velocidad de su especificación.

Fig. 3 muestra el algoritmo para generar todos los posibles estados del sistema circuito-entorno (*Alcanzados*), similar al algoritmo *Breadth First Search* utilizado para atravesar máquinas de estados [14]. Dado un conjunto de estados (*Desde*), el algoritmo calcula los nuevos estados alcanzados tras conmutar todas las puertas internas (*Nuevos*). Desde este nuevo conjunto se comprueba que no haya estados erróneos y se disparan, síncronamente, las señales de entrada y salida del circuito y las transiciones sensibilizadas (*Hasta*). El conjunto *Desde* para la siguiente iteración será el conjunto *Hasta*, pero eliminando aquellos estados ya visitados. Finalmente, el algoritmo termina cuando ya no se generan nuevos estados.

Los cambios en las señales de salida se producen sólo tras haber propagado las transiciones en las entradas del circuito a lo largo de éste. Conmutar todas las puertas internas hasta que no haya más cambios internos ahorra muchos cálculos, ya que de no hacerse así, el algoritmo calcularía las funciones δ_I y δ_O para un gran número de estados en los que no se producirían cambios. Además, el tamaño de los BDDs a lo largo de este "subrecorrido" es menor. Por tanto, esta optimización resulta en menores tiempo de ejecución y tamaño de BDD.

6 Ejemplos

Esta sección muestra la potencia de nuestra metodología para verificar circuitos de tamaño moderado contra su especificación. Los ejemplos escogidos son dos diferentes implementaciones escalables de un árbitro que controla el acceso de N usuarios a un recurso compartido.

Tab. 1 y Tab. 2 presentan los resultados obtenidos verificando árbitros con diferente número de usuarios. Para cada árbitro se muestran el número de señales y de estados, el tamaño de los BDDs que representan dichos estados, el número

Tabla 1: Resultados del anillo de *DMEs*

usr	señ.	estados	BDD final	iter.	CPU
2	36	5.0×10^2	542	5	5
3	54	6.6×10^3	1945	6	36
4	72	7.5×10^4	5501	7	191
5	90	8.0×10^5	13946	8	838
6	108	8.2×10^6	33336	9	2566
7	126	8.2×10^7	76951	10	7881

Tabla 2: Resultados del árbitro en árbol

usr	señ.	estados	BDD final	iter.	CPU
2	10	52	160	9	1
3	18	4.4×10^2	765	16	3
4	26	38×10^3	2802	17	15
5	34	3.0×10^4	9048	24	71
6	42	2.3×10^5	25590	25	271
7	50	1.8×10^6	81449	32	1163
8	58	1.4×10^7	211973	33	3703

de iteraciones y tiempo de ejecución, en segundos, del algoritmo de Fig. 3. Los tiempos de CPU se han obtenido en una estación de trabajo Sun SPARC 10 con 64Mb de RAM.

El primer ejemplo es un anillo de celdas DME (*Distributed Mutual Exclusion*), propuesto por Martin [15] (ver Fig. 4), y posteriormente estudiado por otros autores Dill [5, 4], McMillan [9], Burch [8], mediante diferentes técnicas.

El circuito que implementa un árbitro de N usuarios compartiendo un recurso es un anillo de N celdas. Cada celda está conectada a un usuario, y a los vecinos de su derecha e izquierda. La exclusión mutua esta garantizada mediante un único *token* que se mueve alrededor del anillo. La celda que tiene el *token* puede conceder el recurso a su usuario o pasárselo al vecino de la derecha. Fig. 5.a muestra un anillo con dos usuarios, mientras que Fig. 5.b muestra la red de Petri que describe el comportamiento del circuito. Cada usuario sigue un protocolo de cuatro fases con el circuito para pedir y abandonar el recurso.

Mirando los resultados de la tabla 1, se observa que el tamaño de los BDDs que representan el sistema circuito-entorno se mantiene moderado, y los tiempos de verificación son aceptables, teniendo en cuenta el gran número de estados del sistema. El número de iteraciones da idea de la

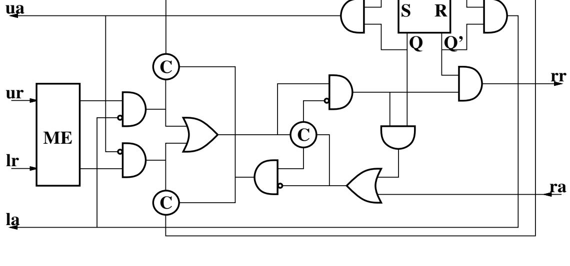

Figura 4: Celda DME propuesta por Dill

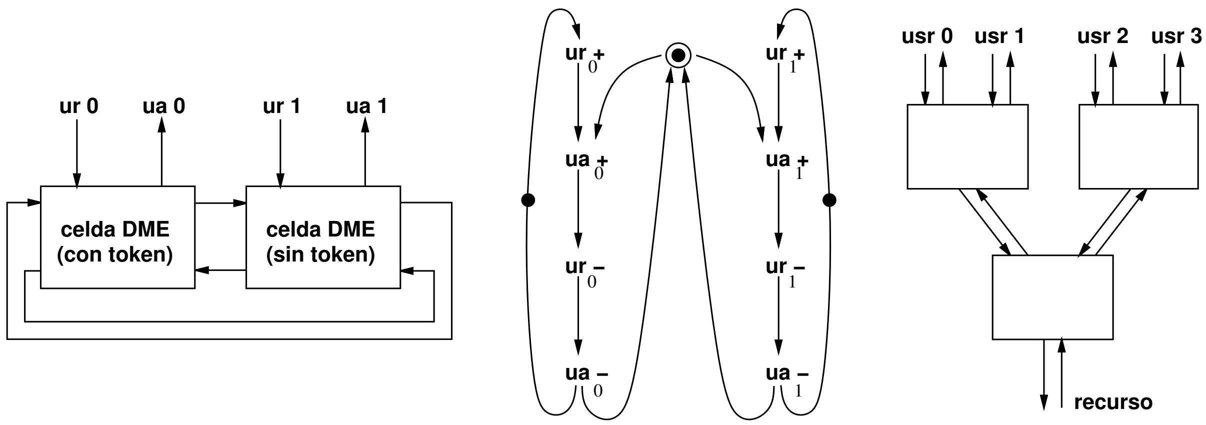

Figura 5: (a) Anillo con dos usuarios. (b) Red de Petri para el entorno. (c) Árbol con cuatro usuarios.

profundidad de la red de Petri y del circuito.

El segundo ejemplo es un árbitro en árbol propuesto por Seitz [16] y modificado por Dill [4]. Fig. 5.c muestra un árbol de tres celdas que controla el acceso de cuatro usuarios a un mismo recurso. Este ejemplo es peor que el primero en términos de tamaño de BDDs que representan el sistema. Posiblemente los resultados mejorarían escogiendo un mejor orden de variables.

7 Conclusiones

En este artículo se ha presentado una nueva metodología para verificar circuitos independientes de la velocidad a partir de su especificación descrita mediante redes de Petri. La principal ventaja de esta técnica es la posibilidad de verificar el circuito con su especificación original. Además se pueden verificar propiedades a nivel de red de Petri.

Modelar redes de Petri con álgebras booleanas, que consecuentemente permite el uso de potentes técnicas que usan BDDs para representar conjuntos de marcados, abre nuevas posibilidades para extender estas técnicas a otras áreas donde se requieren síntesis y verificación.

Las técnicas que utilizan modelos simbólicos, aunque interesantes, todavía consumen un tiempo de CPU considerable. Por ello se requiere más investigación para encontrar optimizaciones que permitan usar BDDs en un amplio rango de aplicaciones (heurísticas de ordenación de variables, explotación de la regularidad, y métodos para reducir el número de estados).

Referencias

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

[2] L. Lavagno, K. Keutzer, y A. Sangiovanni-Vincentelli. "Algorithms for Synthesis of Hazard-free Asynchronous Circuits". En *Proc. of DAC*, páginas 302–308, Junio, 1991.

[3] P. Beerel y T. Meng. "Automatic Gate-level Synthesis of Speed-independent Circuits". En *Proc. of the ICCAD*, Noviembre, 1992.

[4] D. L. Dill. "Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits". PhD thesis, CMU, Febrero, 1988.

[5] D. L. Dill y E. M. Clarke. "Automatic verification of asynchronous circuits using temporal logic". En *Proceedings of the Institution of Electrical Engineers*, páginas 276–282, Septiembre, 1986.

[6] C. W. Moon. "Synthesis and Verification of Asynchronous Circuits from Graphical Specifications". PhD thesis, University of California at Berkeley, 1992.

[7] J. R. Burch, E. M. Clarke, y D. E. Long. "Representing circuits more efficiently in symbolic model checking". En *Proc. of DAC*, páginas 403–407, San Francisco, CA, Junio, 1991.

[8] J. R. Burch, E. M. Clarke, D. Long, K. L. McMillan, y D. L. Dill. "Symbolic model checking for sequential circuit verification". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Junio, 1992.

[9] K. McMillan. "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits". En *Proc. of CAV*, Junio, 1992.

[10] M. Kishinevsky, A. Kondratyev, A. Taubin, y V. Varshavsky. "Concurrent Hardware. The Theory and Practice of Self-timed Design". Series in Parallel Computing. Wiley, 1994.

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

[12] F. M. Brown. "Boolean Reasoning: The Logic of Boolean Equations". KAP, 1990.

[13] E. Pastor, O. Roig, J. Cortadella, y R. Badia. "Petri net Analysis Using Boolean Manipulation". En *International Conference on Application and Theory of Petri Nets*. Springer-Verlag, Junio, 1994.

[14] O. Coudert, C. Berthet, y J. C. Madre. "Verification of sequential machines using boolean functional vectors". En *Proc. IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, páginas 111–128, Leuven, Belgium, Noviembre, 1989.

[15] A. J. Martin. "The design of a self-timed circuit for distributed mutual exclusion". En *Chapel Hill Conference on VLSI*, 1985.

[16] C. L. Seitz. "System timing". En *Introduction to VLSI Systems*, capítulo 7. Mead & Conway, Addison-Wesley, 1980.