Automatización del problema de asignación de estados en el diseño de sistemas secuenciales síncronos

Juan Fº Sanjuan Estrada, I. García Fernández, J. A. Alvarez Bermejo
Dpto. de Arquitectura de Computadores y Electrónica
Universidad de Almería
04120 Almería
e-mail: jsanjuan@ual.es, inma@ace.ual.es, jaberme@ace.ual.es

Resumen

Cuando un alumno se introduce en el mundo del diseño de sistemas secuenciales síncronos se tropieza con dos problemas, minimización y codificación de estados, tanto más significativos cuanto menor número de puertas deseemos. Para la resolución del problema de minimización de estados existe un algoritmo, tan sencillo como largo y tedioso, que esta implementado en la mayoría de los entornos computacionales de diseño digital. Sin embargo, para la resolución del problema de asignación de estados, existen una serie de reglas difíciles de aplicar.

Por este motivo se ha programado una herramienta, que a partir de la tabla de transiciones de estados minimizada, permite al alumno obtener los circuitos con menor número de puertas para los distintos tipos de biestables deseados.

1. Motivación

Las técnicas de diseño de sistemas secuenciales síncronos son cruciales en el aprendizaje que reciben los alumnos de primer curso de Ingeniería Técnica en Informática, correspondiente al temario de asignaturas tales como Tecnología de Computadores y Laboratorio de Estructura y Tecnología de Computadores.

En la docencia teórica interesa que el alumno conozca la metodología de diseño de sistemas secuenciales síncronos con ejemplos, y afiance sus conocimientos con la realización de problemas y ejercicios. Sin embargo, en las clases prácticas de laboratorio el alumno debe aprender a simular e implementar sistemas digitales con circuitos integrados comerciales. El alumno aumenta su rendimiento cuando los circuitos a realizar contienen un menor número de puertas posibles, de ahí que el alumno encuentre un gran apoyo en el software que le permita resolver tareas tales como la minimización de estados, minimización de funciones, y asignación de estados.

La mayoría de las herramientas de diseño de sistemas digitales, disponibles en el mercado, omiten el problema de codificación de estados, permitiendo que el diseñador elija el código de asignación optando por una codificación intuitiva y sencilla, o en el mejor de los casos emplea un algoritmo heurístico que pretende predecir rápidamente la mejor codificación, con resultados dudosos. Sin embargo, la codificación de estados es un problema combinatorio, cuya complejidad aumenta rápidamente al incrementarse el número de estados, por lo que la solución óptima se alcanza resolviendo un problema NP [6].

Este articulo se estructura en los siguientes aspectos, primeramente se realiza una breve introducción a los autómatas de estados finitos (sección 2), continuando con una descripción global de la metodología de diseño de sistemas secuenciales (sección 3). Posteriormente, se comentan el problema de minimización de estados (sección 4) y el problema de asignación de estados (sección 5). En la sección 6 se describe la herramienta de asignación de estados CCEDAF [8], para finalizar con las mejoras previstas que se implementarán en un futuro próximo (sección 7) y conclusiones finales (sección 8).

2. Autómata de estados finitos

Un sistema secuencial, también denominado autómata de estados finitos, se caracteriza porque

la salida no sólo depende de la entrada, sino también del estado en que se encuentra. Es decir, un sistema secuencial debe ser capaz de recordar el estado actual, por eso se dice que un sistema secuencial es aquel que tiene memoria (ver figura 1).

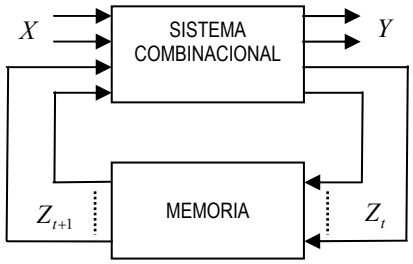

Figura 1. Esquema básico de un sistema secuencial.

Otra característica típica en los sistemas secuenciales es el sincronismo, que hace que el sistema secuencial genere la secuencia de forma ordenada en el tiempo, según una señal de reloj. Gran cantidad de sistemas necesitan de una evolución ordenada. Además, el diseño y análisis de circuitos secuenciales síncronos es más sencillo y seguro que el correspondiente a circuitos secuenciales asíncronos [2].

Un sistema secuencial síncrono tiene como elemento básico al flip-flop por flanco, de tal forma que el sistema en su conjunto evoluciona al ritmo del reloj y produce una secuencia de salida en función de la entrada y del estado almacenado en los flip-flops.

3. Procedimiento de diseño

El diseño genérico de un autómata se estructura en las siguientes fases [1]:

1. Diseño del diagrama de estados
2. Creación de la tabla de transiciones
   2.1 Minimización de estados
   2.2 Codificación de estados
3. Creación de la tabla de excitaciones
4. Simplificación e implementación

La calidad final del circuito dependerá de la experiencia e intuición del diseñador, además de la complejidad del propio sistema. La experiencia se alcanza con ejemplos y con el propio diseño de autómatas. Sin embargo en el diseño de autómatas de estados finitos surgen dos grandes problemas, por un lado la minimización de estados y la codificación de estados. Para la resolución de ambos problemas se deben aplicar procedimientos sistemáticos, cuya correcta aplicación de estos métodos deriva en un circuito mínimo y funcional [2].

A partir de la descripción textual del sistema a diseñar, que puede ser incompleta y/o incoherente, se crea un diagrama de estados (ver figura 2) que explique de una manera gráfica el funcionamiento del autómata.

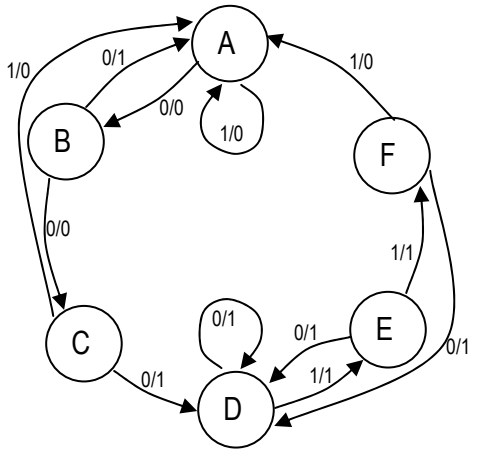

Figura 2. Ejemplo de un diagrama de estados.

En esta fase la comprensión, experiencia e intuición del diseñador son fundamentales para un diseño correcto.

	x = 0	x = 1
A	B, 0	A, 0
B	C, 0	A, 0
C	D, 1	A, 0
D	D, 1	E, 1
E	D, 1	F, 1
F	D, 1	A, 0

Tabla 1. Ejemplo de una tabla de transiciones.

La tabla de transición de estados (ver tabla 1) no es más que la reordenación en forma de tabla de la información gráfica del diagrama de estados. Es decir, es una transformación de gráfico a tabla, de hecho el diagrama de estados no es necesario, aunque sí es cómodo su uso para aclarar el diseño.

En esta fase, el alumno deberá de resolver dos problemas que surgen si se desea obtener un circuito con el menor número de puertas. Nos referimos a la minimización y codificación de estados, que estudiaremos con más detalle en apartados posteriores.

En la tabla de excitación (tabla 2) primero reordenamos en filas el panel de la tabla de transición de estados y finalmente asociamos los valores de las entradas de los biestables, según la tabla de excitación del biestable elegido (tabla 3).

	$y_2^n\ y_1^n\ y_0^n$			$D_2\ D_1\ D_0$						Salida, z	
				X=0			x=1			X=0	x=1
A	0	0	0	0	0	1	0	0	0	0	0
B	0	0	1	1	0	0	0	0	0	0	0
-	0	1	1	-	-	-	-	-	-	-	-
-	0	1	0	-	-	-	-	-	-	-	-
D	1	1	0	1	1	0	1	1	1	1	1
E	1	1	1	1	1	0	1	0	1	1	1
F	1	0	1	1	1	0	0	0	0	1	0
C	1	0	0	1	1	0	0	0	0	1	0

Tabla 2. Ejemplo de una tabla de excitación con biestables tipo D.

Q	$\overline{Q}$	R	S
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Q	$\overline{Q}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Q	$\overline{Q}$	D
0	0	0
0	1	1
1	0	0
1	1	1

Q	$\overline{Q}$	T
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 3. Tabla de excitaciones de los biestables

A partir de la tabla de excitación deberemos de simplificar, por ejemplo con el método de Veitch-Karnaugh, cada una de las funciones correspondientes a las entradas de los biestables y a las salidas del sistema secuencial. Ver ejemplo con las siguientes funciones lógicas minimizadas:

$$D_2 = \overline{y_2} y_0 \bar{x} + y_2 \bar{x} + \overline{y_1}$$

$$D_1 = y_2 \bar{x} + y_1 \overline{y_0}$$

$$D_0 = \overline{y_2}\,\overline{y_1}\,\overline{y_0}\,\bar{x} + y_1 x$$

$$z = y_1 + y_2 \bar{x}$$

Finalmente el alumno, en prácticas de laboratorio, analizará mediante un software de simulación a fin de comprobar su correcto funcionamiento, para posteriormente implementar el circuito con chips comerciales. Por este motivo, la obtención de un circuito lo más reducido posible es crucial para una sencilla simulación, fácil implementación y rápida localización de errores.

4. Problema de minimización de estados

De un autómata sólo nos interesa la salida que va produciendo, no el número de estados que necesita para hacerlo, es decir, nos interesa la salida, no por cuántos estados distintos ha pasado. Por lo tanto, es claro el interés de minimizar la tabla de estados para obtener el número mínimo de estados sin degradar el autómata, en el mejor de los casos podríamos disminuir el número de biestables necesarios, simplificando tanto el proceso de creación de la tabla de excitación como el circuito final.

Para realizar esta minimización de la tabla de estados existen dos métodos: el método de Huffman y el método de las tablas de implicación [2]. Cabe decir que minimizar supone aplicar un algoritmo tan sencillo como largo y tedioso; que está implementado en la mayoría de los entornos computacionales de diseño digital, incluyendo el BOOLE-DEUSTO [7].

Una vez minimizada la tabla de estados, hay que decidir el número de variables de estado. La cuantía de las variables de estado se establece de forma que haya las menos posibles. Por tanto, si en la tabla a sintetizar hay m estados, el número n de variables de estado vendrá dado por la relación:

$$2^{n-1} < m \leq 2^n \qquad (1)$$

5. Problema de asignación de estados

En cualquier caso, una vez decidido el número de variables de estado, se ha de codificar cada uno de ellos; este proceso de codificación es el que se conoce como ***asignación de estados***. Para situarnos en el problema, vamos a considerar la asignación de estados correspondiente a la tabla de transiciones (Tabla 1). En esta tabla hay seis estados, con lo que resultan tres variables de estado, según la expresión (1).

5.1. Problema combinatorio

Utilizando resultados de combinatoria se obtiene que el número de combinaciones (N_1) posibles de códigos binarios asignados a n variables de estados responde a la expresión:

$$N_1 = \frac{2^n!}{(2^n - m)!} \qquad (2)$$

Sin embargo, este número de combinaciones se puede reducir considerablemente, si tenemos en cuenta la *adyacencia* entre estados [4], de tal forma, que entre dos estados adyacentes solo se pueden asignar códigos binarios adyacentes, con lo que se podrán descartar el resto de combinaciones no adyacentes. La adyacencia de estados permite descartar aquellas codificaciones binarias que difieren en el orden de las variables o en la complementación de alguna variable; por lo que el número de combinaciones realmente necesarias viene dado por la expresión (3).

$$N_2 = \frac{(2^n - 1)!}{(2^n - m)!n!} \qquad (3)$$

En la tabla 4 se muestran algunos valores de N_1 y N_2 obtenidos a partir de las ecuaciones (2) y (3) respectivamente, y es claro que este valor se dispara en cuanto m crece ligeramente.

m	n	N_1	N_2
3	2	24	3
4	2	24	3
5	3	6.720	140
6	3	20.160	420
7	3	40.320	840
8	3	40.320	840
9	4	$4.151_1347.200$	$10_1810.800$
10	4	$29.059_1430.400$	$75_1675.600$

Tabla 4. Valores de posibles asignaciones.

Para pequeños valores de m (por ejemplo, 3 ó 4) y n (por ejemplo, 2), sólo hay tres asignaciones estrictamente diferentes; es decir, se pueden encontrar otras asignaciones, pero con igual número de puertas, convirtiéndose en una de las tres mediante operaciones de permutación y/o complementación de las variables. Dado que no es un número muy elevado, se pueden probar las tres asignaciones, para escoger la mejor. Para cuatro o menos estados se podrían probar manualmente las tres posibles asignaciones, pero para más estados es prácticamente imposible.

Por otro lado, como se puede observar en la tabla 4, para más de ocho estados, el número de combinaciones posibles se dispara enormemente, cuya resolución es el objetivo de muchos investigadores [3].

5.2. Reglas para la adyacencia de estados

Para disminuir el problema de codificación de estados no se dispone de un algoritmo, sino de una serie de reglas o recomendaciones a seguir en el proceso de asignación de variables, que se ordenan a continuación según su importancia relativa:

1. Los estados que tengan los mismos estados siguientes en todas las columnas recibirán asignaciones adyacentes. Si la coincidencia en los estados siguientes ocurre en todas las columnas menos en una, los estados correspondientes también deberán de ser adyacentes, pero con menos prioridad que en el caso anterior. Y así sucesivamente para coincidencias en todas las columnas menos en dos, etc...
2. Los estados siguientes de uno dado situados en columnas adyacentes recibirán asignaciones adyacentes.
3. Las asignaciones deberán simplificar las funciones de salida (es decir, estados con las mismas salidas para determinadas

combinaciones de entrada recibirán asignaciones adyacentes).

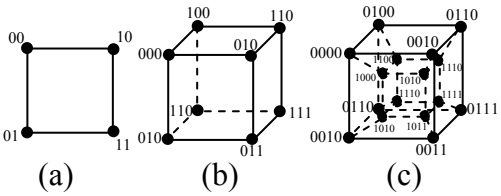

Figura 3. Adyacencia entre estados: (a) 2 variables, (b) 3 variables, y (c) 4 variables.

Finalmente, a aquellos estados adyacentes se les asigna una codificación adyacente (figura 3), donde se observan las asignaciones adyacentes de dos variables (a), de tres variables (b) y de cuatro variables (c).

Sin embargo, la aplicación de estas reglas no siempre genera unas recomendaciones claras, y con frecuencia no se pueden cumplir todas las recomendaciones. En términos generales, si varios diseñadores tratan de realizar una misma tabla de estados, lo normal es que cada uno utilice una asignación diferente, aún aplicando todos correctamente los puntos anteriores, y las realizaciones serán prácticamente del mismo coste, sin garantizar que se trate del mínimo número de puertas posibles.

5.3. Asignación de estados automática

El objetivo fundamental es obtener el circuito digital óptimo, es decir, aquel cuyo bloque de memoria disponga de un menor número de biestables y cuyo bloque combinacional contenga el menor número de puertas lógicas.

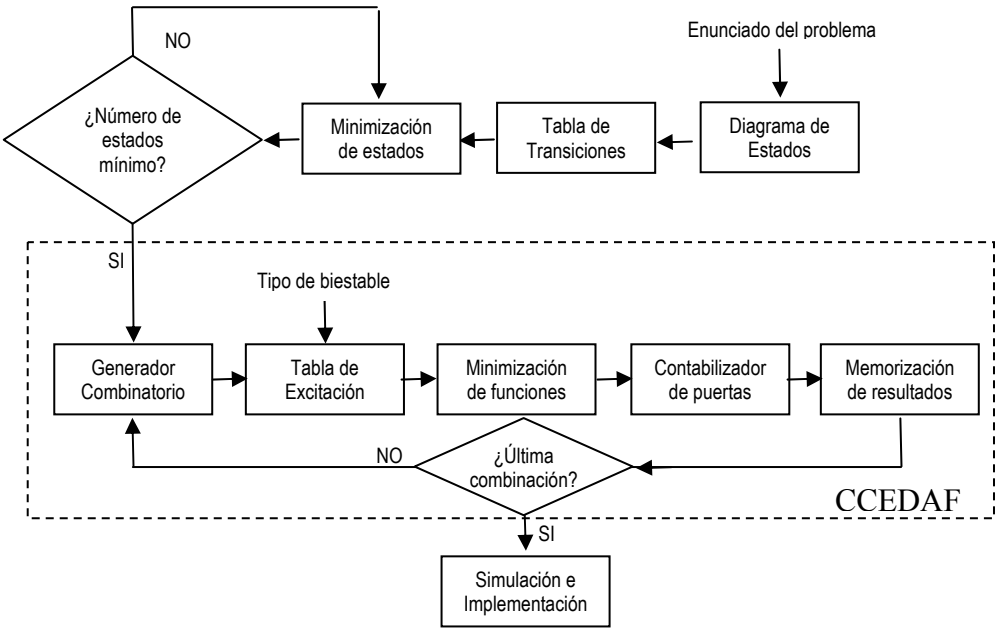

Figura 4. Esquema general del procedimiento para el diseño de circuitos secuenciales síncronos, especificando las tareas realizadas por la herramienta software CCEDAF.

Por un lado, el menor número de biestables requeridos viene prefijado por el número de estados minimizados (problema de minimización de estados). Mientras que el número de puertas lógicas necesarias vienen impuestas por el problema de codificación de estados.

La forma más exacta de resolver el problema de codificación es evaluando todas las combinaciones posibles de asignación de

estados, sin embargo, el tiempo de computación necesario aumenta rápidamente.

6. Herramienta de asignación de estados

Se ha diseñado una herramienta software, denominada CCEDAF – *Codificador Combinatorio de Estados para el Diseño de Autómatas Finitos*, que permite a partir de una tabla de transiciones minimizada evaluar todas las posibles combinaciones de codificación de estados, para los diferentes biestables existentes: JK, RS, D y T, contabilizando el número de puertas AND, OR y NOT de dos entradas necesarias en cada asignación.

En la figura 3 se observa la estructura interna de la herramienta software CCEDAF dentro del esquema general para el diseño de autómatas finitos. Como se puede observar el alumno se responsabiliza de la creación del diagrama de estados y la tabla de transiciones a partir del enunciado del problema. Esta fase es la menos automatizable, y su éxito depende mucho de la experiencia e intuición del diseñador, así como de la complejidad del propio sistema.

Posteriormente, el alumno puede emplear diferentes herramientas, tales como BOOLE-DEUSTO, para la obtención de la tabla de transiciones minimizada.

La herramienta CCEDAF internamente se estructura en cinco fases: generador combinatorio, creación de la tabla de excitación, minimización de funciones, contabilizador de puertas y memorizador de resultados.

El generador combinatorios se encarga de generar todas las combinaciones posibles de la codificación de estados. A partir del número de estados *m*, disponibles en la tabla de transiciones proporcionada por el alumno, calcula el número de bits *n* necesarios para la codificación de todos los estados, sabiendo que se debe cumplir la expresión (1).

Una vez generada la combinación actual, se estudia la adyacencia de estados, con lo que el número de combinaciones se puede reducir considerablemente, según la expresión (2).

A partir del tipo de biestable (JK, RS, D, T) escogido por el alumno, CCEDAF crea la tabla de excitación del sistema asignando la codificación actual de estados proporcionada por el generador combinatorio.

6.1. Minimización de funciones

Las funciones de entrada de cada uno de los biestables, así como las salidas del sistema digital, son minimizadas por el método de Quine-McCluskey [2], más fácil de programar que el método gráfico de Veitch-Karnaugh.

6.2. Contabilizador de puertas

A partir de las funciones de entrada de biestables y salidas del sistema, ya minimizadas, se contabilizan el número de puertas AND, OR y NOT de dos entradas necesarias para su implementación. El coste de una ecuación booleana B_i representada en forma de suma de productos, viene dada por la ecuación (4).

$$C(B_i) = \sum_{j=0}^{k-1} P_j(B_i) + O(B_i) \qquad (4)$$

donde k es el número de términos producto en la ecuación booleana.

$$P_j(B_i) = \begin{cases} m & \text{Si el término } j \text{ de } B_i \text{ tiene } m \text{ variables} \\ 0 & \text{Si el término } j \text{ de } B_i \text{ tiene 1 variable} \end{cases}$$

$$O(B_i) = \begin{cases} m & \text{Si } B_i \text{ tiene } m \text{ términos} \\ 0 & \text{Si } B_i \text{ tiene 1 término} \end{cases}$$

Por lo tanto, dado un conjunto de ecuaciones booleanas $S = \{ B_0, B_1, \ldots, B_{n-1} \}$, su costo es calculado por la ecuación (5).

$$C(S) = \sum_{i=0}^{n-1} \left[C(B_i) - \sum_{j=0}^{k_i-1} R_j(B_i) \right] \qquad (5)$$

donde $C(B_i)$ es el coste de la ecuación B_i, y k_i es el número de términos producto en la ecuación B_i, con $i > 1$.

$$R_j(B_i) = \begin{cases} P_j(B_i) & \text{Si el término } j \text{ de } B_i \text{ se repite en otro término distinto.} \\ 0 & \end{cases}$$

Finalmente, el coste de un circuito secuencial síncrono es igual al coste del conjunto de ecuaciones booleanas formadas por las ecuaciones que generan los próximos estados más las ecuaciones de salidas del circuito.

Es importante destacar que se busca el número de puertas totales, es decir, si una función requiere la misma puerta lógica que la de otra función, esta puerta lógica solo se contabiliza una vez.

6.3. Memorizador de resultados

Finalmente, el software CCEDAF almacena un número de resultados establecido por el alumno, que generan un menor número de puertas, descartando de la lista las codificaciones que requieren un mayor coste

Al diseñar sistemas secuenciales normalmente existen varias combinaciones con el mismo número de puertas, por este motivo, CCEDAF le ofrece al alumno la posibilidad de indicar el número de resultados óptimos.

El alumno puede ejecutar varias veces el programa CCEDAF, para distintos tipos de biestables, y escoger los mejores resultados, para finalmente realizar la fase de simulación o implementación.

7. Rendimiento del alumno

Es obvio que el tiempo de simulación e implementación de circuitos disminuye cuando el circuito se simplifica, sin embargo, esto repercute en un aumento del tiempo de diseño del sistema digital. Con este articulo, proponemos incrementar el tiempo de diseño, aumentando el tiempo de computación, de tal forma que disminuya el tiempo de simulación e implementación del circuito, por lo que aumentará el rendimiento del alumno en la realización de prácticas de laboratorio.

7.1. Ejecución

La herramienta CCEDAF esta programada en el lenguaje orientado a objetos C++, pudiéndose ejecutar tanto en Linux como en Windows. El alumno puede ejecutar simultáneamente varias veces CCEDAF con la misma tabla de transiciones pero con distinto tipo de biestable, y seleccionar la solución más adecuada para realizar la simulación e implementación.

7.2. Reducción del tiempo de computación

La evaluación de todas las posibles asignaciones de estados del problema combinatorio se traduce en un aumento del tiempo de computación. Sin embargo, el tiempo de computación oscila entre varios minutos para menos de ocho estados, hasta varios días para sistemas más complejos.

Sin embargo, la mayoría de los circuitos expuestos tanto en libros de texto, como en las prácticas propuestas en el laboratorio, pueden ser resueltas a través de tablas de transiciones minimizadas con un máximo de ocho estados, lo que permitirá al alumno obtener el circuito mínimo para distintos tipos de biestables en poco tiempo (diez o quince minutos).

En el caso que el alumno desee diseñar circuitos con más de ocho estados, el tiempo de computación se puede reducir considerablemente si en vez de evaluar todas las combinaciones posibles N_1 (varios días o incluso semanas), solo analiza aquellas combinaciones realmente distintas N_2 (varias horas incluso días) que cumplan la adyacencia de estados.

7.3. Simulación e implementación

La información de salida de CCEDAF contiene los mejores resultados con menor coste para distintos tipos de biestables, donde el alumno puede estudiar todos los sistemas de expresiones booleanas en forma de producto de sumas, y seleccionar la mejor opción para una posterior simulación e implementación.

8. Mejoras futuras

La actual versión de CCEDAF sufrirá actualizaciones en un futuro próximo, entre las que se tiene previsto realizar, destacamos las siguientes:

1. Incluir una fase de preprocesamiento que permita minimizar la tabla de transiciones, permitiendo reducir el número de estados totales. De esta forma la aplicación

CCEDAF será capaz de resolver los dos problemas propios del diseño de sistemas secuenciales sincronos.

2. Programar una versión paralela de CCEDAF, que permita repartir la carga computacional entre varios procesadores, para de esta forma reducir enormemente el tiempo de computación.
3. Diseñar un entorno gráfico más amigable, que le permita al usuario, introducir correctamente las tablas de transiciones de estados a través del diagrama de estados.

9. Conclusión

Con la herramienta software CCEDAF se pretende que durante el diseño de sistemas secuenciales sincronos en prácticas de laboratorio, el alumno dedique un mayor tiempo a la simulación e implementación de circuitos, dejando que los cálculos de diseño de autómatas finitos los realice en las horas de ejercicios.

Sin esta aplicación, el alumno dedica excesivo tiempo al diseño teórico del sistema, reduciendo tiempo de simulación, y casi siempre generan circuitos con un gran número de puertas, lo que repercute en una imposible implementación y sobre todo en una mayor dificultad de localización de errores, tanto para el alumno como para el profesor.

Referencias

[1] Angulo Usategui, J.M. y García Zubía, J. *Sistemas digitales y tecnología de computadores.* Paraninfo, 2002.

[2] Lloris Ruíz, A., Prieto Espinosa, A., y Parrilla Roure, L. *Sistemas digitales*. McGraw-Hill, 2003.

[3] Nelson Amaral, J. Kagan Tumer and J. Ghosh. *Designing genetic algorithms for the state assignment problem*. IEEE Transactions on systems man and cybernetics. Vol. 20, No. 10, 1999.

[4] S. Devadas and R. Newton. *Exact algorithms for output encoding, state assignment, and four-level boolean minimization*. IEEE Transactions on computer-aided design. Vol. 10, No. 1, January, 1991.

[5] L. Benini and G. De Micheli. *State assignment for low power dissipation*. IEEE Journal of solid-state circuits. Vol. 30, No. 3, March 1995.

[6] T. Villa and Sangiovanni-vincentelli, A. *NOVA: State assignment of finite state machines for optimal two-level logic implementation*. IEEE Transactios on computer-aided design. Vol. 9, No. 9, September 1990.

[7] J. García Zubia, J. Sanz Martinez, y B. Sotomayor. *BOOLE-DEUSTO, la aplicación para sistemas digitales*. VII Jornadas de Enseñanza Universitaria de la Informática (JENUI), 2001.

[8] J.F. Sanjuan Estrada, I. García Fernández, J. A. Alvarez Bermejo. *CCEDAF: Codificador Combinatorio de Estados para el Diseño de Autómatas Finitos*. JENUI. 2004.