CADPC: UN ENTORNO DE DISEÑO DE CIRCUITOS DIGITALES EN ORDENADORES PERSONALES

Suárez L., Barriga A., Valencia M.

Centro Nacional de Microelectrónica (Edificio CICA)
Universidad de Sevilla
Avda. Reina Mercedes s/n 41012 Sevilla

Tfno: 95-4239923
Fax: 95-4624506
E-mail: barriga@cnm.us.es

RESUMEN

En esta comunicación presentamos cadpc (herramienta cad sobre PC), cadpc es un entorno de diseño de circuitos digitales de carácter eminentemente educativo y de bajo coste. El objetivo básico que pretende cubrir este entorno de diseño es el de acercar el mundo del diseño de los circuitos digitales a un amplio colectivo de personas, especialmente dirigido a los alumnos de Circuitos y Sistemas Digitales, mediante la implementación del entorno sobre la plataforma tipo PC

1. INTRODUCCIÓN

Hoy en día, el diseño de circuitos electrónicos requiere apoyarse de forma masiva en herramientas CAD. Esta situación es necesaria debido, por un lado a la cada vez mayor complejidad de los circuitos, y por otro, a los requerimientos en las prestaciones de los mismos: mayores frecuencias de operación, menor ocupación de área, menor consumo de potencia, reducción del ciclo de diseño debido a una mayor competitividad, etc. Este hecho debe reflejarse en la docencia de las asignaturas ligadas al diseño de los sistemas digitales. Las razones que justifican esto no se refieren solamente en adaptar la docencia a la situación real actual, sino a la necesidad de adoptar las nuevas metodologías de diseño. Es por ello por lo que se requiere el desarrollo de herramientas de ayuda a la docencia del diseño de los sistemas digitales. Este es, precisamente, el objetivo de esta comunicación.

Dentro del conjunto de niveles de descripción de los sistemas digitales, *cadpc* cubre el nivel de conmutación, que corresponde a lo que se conoce como Diseño Lógico [1]. Es en este nivel donde se plantean las cuestiones fundamentales del análisis y síntesis de circuitos digitales:

descripción de celdas básicas a nivel lógico, técnicas de diseño y de optimización, simulación del circuito, etc. El entorno desarrollado soporta las tres tareas principales del proceso: 1) descripción inicial; 2) síntesis; y 3) análisis.

El entorno que presentamos está constituido por un conjunto de herramientas de distinta procedencia. Por un lado, se basa en herramientas de la Universidad de California en Berkeley que operan sobre estaciones de trabajo y miniordenadores bajo sistema operativo UNIX. Otras herramientas están desarrolladas sobre ordenadores personales (PC) bajo MS-DOS. Finalmente, un tercer conjunto corresponde a herramientas diseñadas por nosotros. Este hecho ha requerido un esfuerzo de adaptación y compatibilización del software para que opere bajo el mismo entorno. Finalmente se ha desarrollado una interface que permite una interacción amigable entre el usuario y nuestro entorno.

Actualmente se está integrando este entorno en los estudios de F.P. (rama electrónica) en el Instituto Bahía de Cádiz. Asimismo, ha sido leido en la Facultad de Informática y Estadística de la Universidad de Sevilla un proyecto fin de carrera de Diplomatura consistente en el diseño de un conjunto de prácticas para la asignatura Circuitos y Sistemas Digitales I basadas en *cadpc*.

2. EL CICLO DE DISEÑO EN CADPC

En la Figura 1 se muestra el diagrama de herramientas que componen nuestro ciclo de diseño. En dicho diagrama, dividido en las correspondientes fases del diseño, se indica el origen de cada herramienta según:

- haya sido adaptada de una versión para plataformas UNIX [2].
- haya sido realizada originalmente para nuestro entorno.

Observamos además la distinción de tres grandes bloques en el entorno. La zona de descripción de alto nivel corresponde al bloque de especificaciones del sistema a diseñar. En este caso se detalla la funcionalidad del circuito que puede ser descrito como un circuito combinacional (*min2eqn*) o secuencial (*peg* o *meg*). En cualquier caso se traduce dicha descripción de entrada a un formato común de ecuaciones lógicas que son entrada a una herramienta (*eqntott*) que traduce dicha descripción al formato de tabla de verdad. Un segundo bloque corresponde a las herramientas de síntesis. Estas reciben la descripción de entrada y, una vez optimizada, suministran la implementación del circuito. Finalmente, el tercer bloqueo corresponde a las interfaces con otras herramientas. Así, *avst* permite enlazar con un simulador lógico, *apcb* enlaza con herramientas de diseño del placa de circuito impreso, y *avstlib* incluye el diseño en una librería de celdas. Estos tres tipos de *netlist* de salida (simulación, placa y librería) tienen el formato de entrada correspondiente al producto OrCAD (*vst*, *pcb* y *modelpro*, respectivamente).

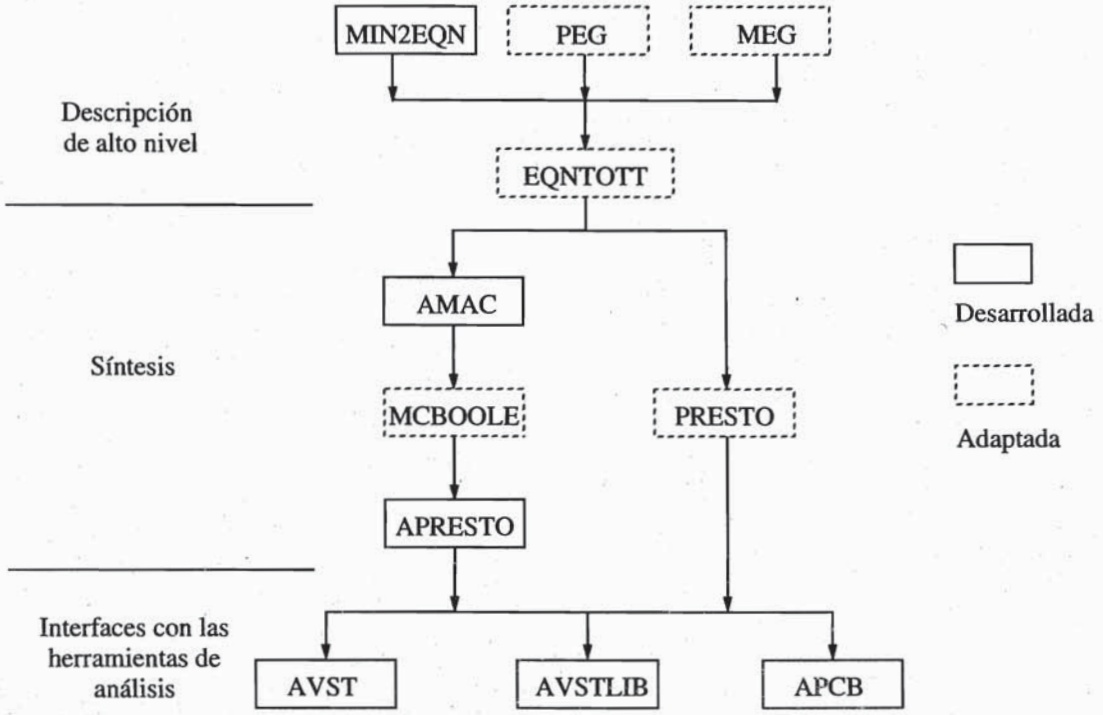
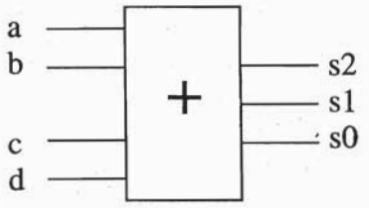


Figura 1: Las herramientas de cadpc, fases y origen.

A continuación pasamos a describir la función y uso de cada herramienta, las cuales han sido agrupadas según la fase de diseño en la que se utilizan: descripción inicial de alto nivel, optimización y análisis. En el caso de las herramientas que son entrada de usuario al entorno, mediante descripción de funciones (*min2eqn*), descripción de máquinas de estados (*peg*, *meg*) y descripción de ecuaciones lógicas (*eqntott*) se ilustra su uso con un ejemplo de fichero de entrada. El caso de las tablas de verdad, formato de entrada de los minimizadores, queda ilustrado con las salidas de *eqntott*.

Descripción de alto nivel

min2eqn: traduce un conjunto de funciones lógicas descritas como suma de mintérminos o producto de maxtérminos (F[a,b,c]=Σ(0,2,5,7)) a su equivalente en formato de ecuaciones booleanas. Por ejemplo, la descripción en formato de entrada a min2eqn de un sumador asíncrono de dos números de dos bits sería la de la Figura 2 .



s2 [a, b, c, d] = E (7, 10, 11, 13, 14, 15)
s1 [a, b, c, d] = E (2 ,3, 5, 6, 8, 9, 12, 15)
s0 [a, b, c, d] = E (1, 3, 4, 6, 9, 11, 12, 14)

Figura 2: Símbolo y funciones de un sumador en formato min2eqn.

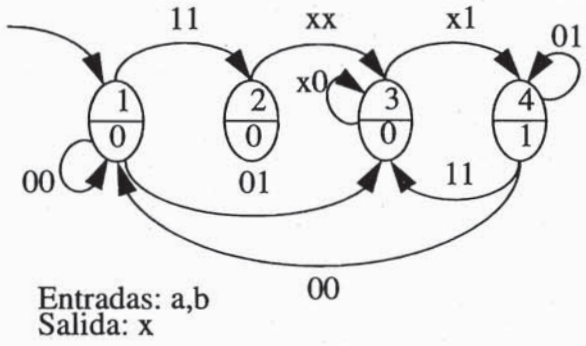
```
s2 = !a&b&c&d | a&!b&c&!d | a&!b&c&d | a&b&!c&d
 | a&b&c&!d | a&b&c&d ;

s1 = !a&!b&c&!d | !a&!b&c&d | !a&b&!c&d | !a&b&c&!d  |
a&!b&!c&!d | a&!b&!c&d | a&b&!c&!d | a&b&c&d ;

s0 = !a&!b&!c&d | !a&!b&c&d | !a&b&!c&!d |!a&b&c&!d  |
a&!b&!c&d | a&!b&c&d | a&b&!c&!d | a&b&c&!d ;
```

Figura 3: Ecuaciones de un sumador de dos números de dos bits.

Al ejecutar *min2eqn* sobre la descripción de entrada de la Figura 2 se obtienen las expresiones en formato de ecuaciones lógicas. La Figura 3 muestra la salida suministrada por min2eqn. En ella el símbolo ! representa la negación lógica, | la operación OR y & la operación AND.

peg: traduce la descripción de alto nivel de una máquina de estados según el modelo de Moore al conjunto de ecuaciones booleanas equivalentes. Permite obtener además un fichero de resumen de las características de la máquina generada (número de entradas, de salidas, de vectores eliminados, etc.) y un informe con información relativa a la tabla de estados interna que usa *peg* para generar las ecuaciones de salida.

En la Figura 4-a se muestra un ejemplo de un diagrama de estados. La descripción de dicho diagrama para *peg* se muestra en la Figura 4-b. Entre las principales características de esta descripción se observa: sencillez en las especificaciones de entradas y salidas, sentencias de control sobre una línea de entrada tipo IF-THEN-ELSE, sentencias de control múltiple CASE, saltos al estado actual LOOP, activación de señales asociadas a los estados ASSERT, saltos incondicionales GOTO, etc.

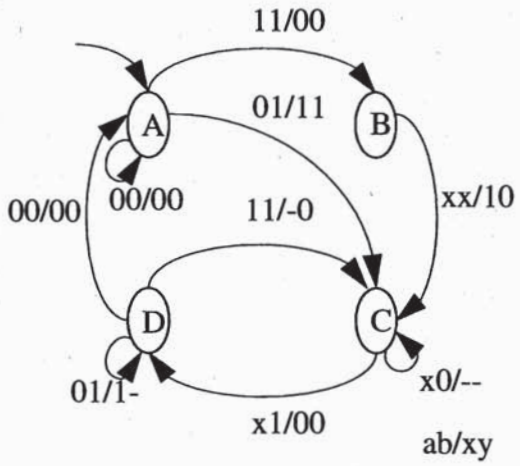


a)

```
INPUTS : RESET a b ;
OUTPUTS : x ;
uno:    CASE (a b)
            11 => dos ;
            01 => tres ;
        ENDCASE => LOOP;
dos: GOTO tres ;
tres: IF b THEN cuatro ELSE LOOP ;
cuatro : ASSERT x ;  -- activa salida
        CASE (a b)
            1 1 => tres ;
            0 0 => uno ;
        ENDCASE => LOOP ;
```

b)

Figura 4: a) Diagrama de estados. b) Descripción en formato de *peg* del diagrama de estados.



a)

```
INPUTS : a b start;
OUTPUTS : x y ;
RESET ON start TO A ;
A : CASE ( a b )
      1 1 => B ;
      0 1 => D (x y) ;
      0 0 => LOOP ;
      ENDCASE => ANY ;
B : GOTO C ( x y ) ;
C : CASE ( a b )
      ? 0 => LOOP ( x=? y=?) ;
      ? 1 => D ( x=0 y=0 ) ;
      ENDCASE => ANY ;
D : CASE ( a b )
      1 1 => C ( x=? ) ;
      0 1 => LOOP ( x y=? ) ;
      0 0 => A ;
      ENDCASE => ANY ;
```

b)

Figura 5: a) Diagrama de estados a implementar con *meg*. b) Descripción meg del diagrama de estados.

meg: equivalente Mealy a *peg*, genera igualmente un fichero de ecuaciones booleanas, uno de resumen y un informe sobre la tabla de estados. En la Figura 5 se muestra una realización de un diagrama de estados. Observamos características similares al caso de *peg*, con la diferencia de la activación de señales en las transiciones, en lugar de asociadas al estado. Por ejemplo, en el estado A, con entradas (a,b)=(0 1) pasamos al estado D, activando las líneas de salida x e y. Podemos ver también la facilidad con la que incluimos una función de RESET al estado que queremos (A) según el valor de una entrada de inicialización (start) de la forma RESET ON start TO A ;

eqntott: herramienta para traducir las ecuaciones lógicas de salida de los tres programas anteriores al formato de tabla de verdad necesario por los minimizadores de la siguiente fase de diseño. En el fichero de salida figura, además de la tabla, una cabecera con los nombres de las variables de entrada y salida. Además, pueden generarse tablas de ecuaciones lógicas más complejas, en la Figura 6 vemos un caso de una ecuación con agrupación por paréntesis y su tabla de verdad asociada. El formato de la tabla tiene una fácil lectura: cuatro líneas de entrada (.i 4), una de salida (.o 1), nombres de las entradas (a, b, c, d), nombre de la salida (f), la tabla en sí, donde se especifican los 1's, 0's y *don't care* (-), y el delimitador de fin de tabla (.e).

```
f = ( a & !b ) | (  ( b&c ) | !( c | !d ) | a ) | !( !a & !b );

                .i 4
                .o 1
                .ilb  a b c d
                .ob  f
                .p 5
                --01  1
                -1--  1
                -11-  1
                1---  1
                10--  1
                .e
```

Figura 6: Ecuación y tabla de verdad asociada.

Optimización

presto: minimizador de tablas de verdad según el algoritmo Quine-McCluskey [3] [4] [5]. Toma como entrada la tabla de verdad de salida generada por *eqntott* y genera su equivalente minimizada.

amac: este filtro traduce el formato de tabla de verdad de salida de *eqntott* al equivalente de entrada al minimizador *mcboole*, más exhaustivo que *presto*, pero que pierde la información relativa a los nombres de las variables de entrada, de las funciones de salida y número de términos producto.

mcboole: minimizador de tablas de verdad según un algoritmo propio [6].

apresto: recupera de la tabla de salida de *mcboole* el formato *presto*, con lo que seguimos arrastrando los nombres de variables y funciones, a los cuales podremos seguir refiriéndonos tanto en la simulación, edición de placa de circuito impreso e inclusión de nuevo componente a librería de la siguiente fase del ciclo de diseño.

Interfases

avst: a partir de una tabla de verdad en formato *presto* y de un fichero donde se especifican tanto la tecnología como los encapsulados a usar, así como la configuración deseada (AND-OR, NAND-NAND, etc.) genera un *netlist* en formato EDIF [7] de entrada al simulador de OrCAD.

apcb: una vez comprobada mediante la simulación el correcto funcionamiento de nuestro diseño podemos obtener, con los mismos parámetros de ejecución usados en *avst*, el *netlist* para *pcb*, que es el editor de placas semiautomático de circuito impreso de OrCAD.

avstlib: nos permite, tras modelar retrasos, añadir como componente a las librerías de OrCAD [8] un diseño de salida minimizado en formato *presto*; este componente podrá ser utilizado como unidad funcional independiente en la captura de esquemas de OrCAD y posterior simulación de nuevos circuitos.

3. EL INTERFACE DE USUARIO - DOCUMENTACIÓN

El uso de un interface de usuario común a todas las herramientas pasaba por la unificación de criterios de ejecución entre las herramientas adaptadas del sistema operativo UNIX y aquellas desarrolladas para completar el entorno. Tras eliminar de las adaptadas características propias de su ejecución bajo UNIX (perfiles de ejecución, manejo de variables de entorno, etc.) se procedió a añadir al código C opciones para el uso de ficheros en lugar del manejo de entrada/ salida estándar mediante redireccionamiento.

Una vez realizada la unificación de criterios de ejecución se adaptó un shell de utilidades DOS para ser usado por nuestro entorno, el cual nos brindaba las siguientes posibilidades:

- Ayuda *on-line* de ejecución.
- Acceso a ayuda externa mas completa.
- Abierto a modificaciones de usuario.
- Llamada a otros comandos del S.O. y otros programas.

En la Figura 7 puede verse un ejemplo del uso del entorno: la agrupación en fases de las herramientas, la ayuda en pantalla, etc.

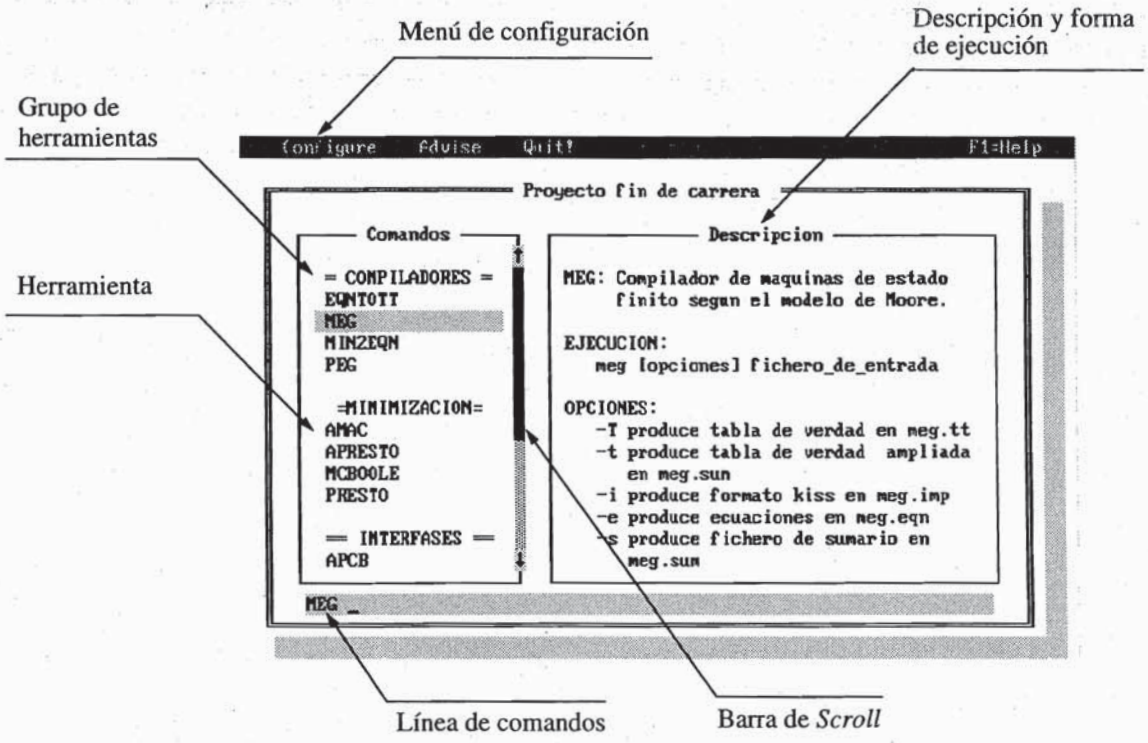


Figura 7: El shell de usuario.

El uso de este *shell* es opcional ya que es posible ejecutar las diferentes herramientas desde la línea de comandos del MS-DOS. Para obtener ayuda existen dos métodos: si la herramienta se ejecuta indicando parámetros accederemos a la ayuda ejecutándola sin éstos, si la herramienta se ejecuta de forma interactiva, accederemos mediante el cualificador *-man* del comando correspondiente a la herramienta.

El entorno está apoyado por un manual de usuario compuesto de tres partes. La primera dedicada a la instalación y puesta a punto del entorno. La segunda está compuesta por un conjunto de tutoriales que explican paso a paso el uso de cada herramienta con ejemplos. La tercera parte es una guía de referencia rápida de las herramientas; esta guía se encuentra integrada en el propio entorno de diseño y puede ser consultada *on-line* como ya indicamos anteriormente.

De entre los factores tenidos en cuenta durante el desarrollo del entorno merece especial interés, desde el punto de vista evolutivo, la claridad, no sólo en el uso de las diferentes herramientas, sino también de las estructuras de los ficheros de datos con un objetivo claro: facilitar la creación e integración de nuevos programas al entorno. Esto, junto con la facilidad de añadir los ejecutables y sus correspondientes ficheros de ayuda al *shell* de usuario hacen de *cadpc* un entorno de diseño abierto a sucesivas mejoras.

4. CONCLUSIONES

Hemos presentado un entorno de diseño centrado en el nivel de conmutación, para lo cual se ha desarrollado un amplio conjunto de herramientas que han venido a cubrir las diferentes etapas por las que pasa un circuito a través del ciclo de diseño. Estas herramientas parten de las especificaciones, cubren el proceso de optimización e implementación del circuito correspondiente, y permite enlazar con otras herramientas (simulador lógico, diseño de placas, etc).

El entorno ha sido implementado sobre PC bajo sistema operativo MS-DOS, lo que permite acercar el mundo del CAD aplicado al diseño digital a un mayor número de personas, destacando su uso como herramienta docente. El entorno es fácil de usar y dispone de ayuda *on-line* y un cómodo sistema de acceso a una guía de referencia rápida.

Finalmente, es un entorno abierto de diseño, esto es, permite incluir con facilidad nuevas herramientas que complementen y/o aumente su funcionalidad.

5. BIBLIOGRAFIA

[1] C. G. Bell y A. Newell, "Computer Structures : Readings and Examples". McGraw Hill, 1971.

[2] "1986 VLSI Tools". Computer Science Division EECS Department University of California at Berkeley, 1986.

[3] E. J. McCluskey, "Minimization of Boolean functions". *Bell Syst. Tech.* J., vol. 35 p. 1417, Abril. 1956

[4] D. W. Brown, "A state Machine synthesizer SMS". *Proc. 18th Design Automation Conf.*, p. 301, Junio 1981.

[5] W. V. Quine, "A way to simplify truth functions". *Amer. Math. Monthly*, vol. 62, p. 627, Noviembre. 1955.

[6] M. R. Dagenais, "Mcboole: a new procedure for exact logic minimization". *IEEE Transactios on Computer-aided Design*, Vol. CAD-5, NO 1, 1986.

[7] "Electronic Design Interchange Format Version 1 0 0". Texas Instruments, Dallas, Texas, 1985.

[8] "Modelling Language. OrCAD VST User's Guide". Pág 102-157, 1989.