ENTORNO DE DISEÑO Y SIMULACIÓN DE CIRCUITOS LÓGICOS

V.Soler, J.Roig
Vicenç Soler
Unitat de Microelectrònica
Departament d'Informàtica
Edifici C
08193 Bellaterra (Barcelona)
Tel: 93-581-21-65
FAX: 93-581-30-33
e-mail: soler@cnm.es

RESUMEN.- Este trabajo es un simulador lógico de fácil uso y conocimiento para que los alumnos puedan tocar aspectos a un nivel que en un simulador lógico comercial no se pueden tocar. Permite, además, el poder ampliarlo, retocarlo, sacar cualquier tipo de estadísticas de la simulación y ver cómo funciona realmente un simulador de este tipo.

1.-INTRODUCCIÓN

Es bien sabido que los alumnos que empiezan a trabajar en el diseño de circuitos integrados lo hacen con herramientas que ponen todas las comodidades posibles para el diseñador. Pero de cara al alumno, tan importante es el hecho de hacer buenos diseños como el conocer cómo funcionan los simuladores que los simulan.
Este proyecto, pues, pretende ser un simulador lógico en el que el alumno pueda sacar estadísticas de rendimiento, conocer todos los formatos de los ficheros y, en general, saber cómo funciona realmente.

2.-DISEÑO

Es un simulador discreto dirigido por eventos programado en Borland C++ 3.1 (con programación orientada a objetos).

Los eventos están formados por tres elementos:

- Tiempo: Es el tiempo en que el evento se ejecutará.
- Acción: Es la acción a ejecutar.
- Parámetros: Son los parámetros que se le pasa a la acción.

El tipo de lenguaje utilizado para la definición de los eventos y de las acciones es del tipo en los eventos irán rodeadas por paréntesis y las acciones llevarán siempre los parámetros entre paréntesis. Por tanto, si se quiere mandar una orden al simulador será mediante un evento, como el del siguiente ejemplo:

(2 PLACE (NOT a b))

en el tiempo 2 se ejecutará la acción PLACE que lleva como parámetros: "NOT a b". De hecho, quiere decir que en tiempo 2 coloque en el circuito un símbolo NOT entre los nodos a y b.
Las unidades de tiempo usadas son ns, ya que las especificaciones de las Data Sheets siempre están expresadas en ns. No obstante, se le puede cambiar la unidad a una especificación temporal cualquiera si se le especifica, excepto si se trata del tiempo en que se ejecuta un evento.
Las unidades de capacidad usadas son pF, ya que, al igual que las unidades de tiempo, las especificaciones de las Data Sheets están especificadas en pF. Al igual que en el tiempo, se le puede cambiar la unidad si se le especifica.

3.-EL LENGUAJE

Mediante un lenguaje definido, el usuario se define una librería de símbolos, que normalmente ya se le proporcionará al alumno. Al símbolo se le definen todas aquellas características que programan su funcionamiento, es decir, los datos que se pueden hallar en las Data Sheets. Así pues, las características que se pueden definir en un símbolo son:

- INPUTS: Define el número, orden y nombre de las entradas al símbolo. Las entradas se pueden especificar en forma de bus.

Definición:
INPUTS '(' <nombre$_1$> <nombre$_2$> ... <nombre$_n$> ')'

- OUTPUTS: Define el número, orden y nombre de las salidas del símbolo. Las salidas se pueden especificar en forma de bus.

Definición:
OUTPUTS '(' <nombre$_1$> <nombre$_2$> ... <nombre$_n$> ')'

- FANIN: Define las capacidades intrínsecas de las entradas. El simulador asocia las capacidades con cada entrada, según el orden en que se entraron en INPUTS.

Definición:
FANIN '(' núm_real ... núm_real ')'

- FANOUT: Define las capacidades intrínsecas de las salidas. El simulador asocia las capacidades con cada salida, según el orden en que se entraron en OUTPUTS.

Definición:
FANIN '(' núm_real ... núm_real ')'

- TRUTHTABLE: Define una tabla de verdad para el símbolo. En esta tabla se podrán definir todos los casos posibles, incluso flancos. Los casos que se permiten son: L, H, 1, 0, X (cualquier opción), Z (Alta Impedancia), R (Flanco de subida), F(Flanco de

bajada) y S (mantener el estado). Los tres últimos casos sólo se usan para definir símbolos secuenciales.

Definición:

TRUTHTABLE '('

<val_in$_{11}$> <val_in$_{12}$> ... <val_in$_{1n}$> <val_out$_{11}$> <val_out$_{12}$> ... <val_out$_{1m}$>

.

.

<val_in$_{c1}$> <val_in$_{c2}$> ... <val_in$_{cn}$> <val_out$_{c1}$> <val_out$_{c2}$> ... <val_out$_{cm}$>

')'

donde val_in es el valor en la entrada correspondiente (por orden de definición en INPUTS) y val_out el valor en la salida, para el caso correspondiente.

- SPECIFICATIONS: Define una función de comportamiento para el símbolo. Es otra manera de especificar una tabla de verdad, más sencilla para algunos símbolos. La especificación se hace mediante un lenguaje que tiene las tres primitivas básicas de las operaciones lógicas: AND, OR y NOT.

Definición:

SPECIFICATIONS '('

<nombre$_1$> '=' action_operator$_1$ '(' op$_{11}$ ... op$_{1n}$ ')'

.

.

<nombre$_m$> '=' action_operator$_m$ '(' op$_{m1}$ ... op$_{mn}$ ')'

')'

Ejemplo: Especificación del funcionamiento de una puerta XOR.

Y=OR(AND(x1 NOT(x2)) AND(NOT(x1) x2))

- TIMING CONSTRAINTS: Define las restricciones temporales (valores de SETUP, HOLD, PWL y PWH) para un símbolo de tipo secuencial (Flip-Flop, etc.).

Definición:

TIMING_CONSTRAINTS '('

<constr$_1$> <from_port$_1$> <to_port$_1$> min$_1$ typ$_1$ max$_1$ mil$_1$

.

.

<constr$_n$> <from_port$_n$> <to_port$_n$> min$_n$ typ$_n$ max$_n$ mil$_n$

')'

- PROPAGATION DELAYS: Define la tabla de retardos de propagación (de baja a alta y de alta a baja) de los símbolos desde las entradas hasta las salidas.

Definición:

PROPAGATION_DELAYS '('

<delay$_1$> <from_port$_1$> <to_port$_1$> min$_1$ typ$_1$ max$_1$ mil$_1$

.

.

<delay$_n$> <from_port$_n$> <to_port$_n$> min$_n$ typ$_n$ max$_n$ mil$_n$

')'

Los símbolos que se definen con estas propiedades son los símbolos básicos, que son los que normalmente se encuentran en una librería de símbolos (ES2, por ejemplo).

El símbolo será definido mediante la acción SYMBOL, a la que se le pasa el nombre del símbolo y todas las características citadas anteriormente.

Todas las especificaciones temporales tienen 4 valores, que corresponden a los casos Mínimo, Típico, Máximo y Militar, respectivamente.

El símbolo será colocado en el circuito mediante la acción PLACE, a la que se le tiene que indicar el nombre del símbolo y a qué nodos del circuito van conectadas sus entradas y salidas.

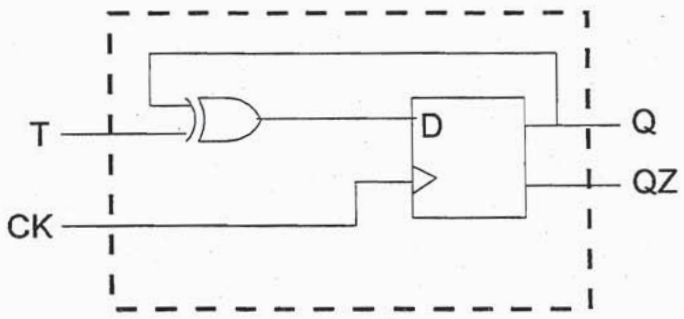
Pero si un alumno desea definirse nuevos símbolos a partir de símbolos ya creados, lo tiene que hacer mediante la acción NEW_SYMBOL. En esta acción se define únicamente el circuito que forma el nuevo símbolo. Para especificar este circuito y sus entradas y salidas generales se ofrecen las siguientes acciones:

- NEW_INPUTS: Define, al igual que INPUTS para SYMBOL, el número, orden y nombre de las entradas al nuevo símbolo, que se pueden especificar en forma de bus.
- NEW_OUTPUTS: Define, al igual que OUTPUTS para SYMBOL, el número, orden y nombre de las salidas al nuevo símbolo, que se pueden especificar en forma de bus.
- NEW_PLACE: Define la red de símbolos que forma el circuito interno del nuevo símbolo. Su especificación es idéntica a PLACE, pero teniendo en cuenta que los nodos definidos son locales al nuevo símbolo (no son nodos del circuito global).

Otras acciones son:
-CLOCK: Permite definir un reloj. Se le tiene que especificar el nodo al cuál va conectado, la frecuencia y la duración del reloj. La duración del reloj determinará la duración de la simulación.
- IFILE: Permite especificar el fichero de estímulos de entrada al circuito. La especificación de los símbolos es la típica de un simulador lógico y no vale la pena comentarla aquí.
- START: Hace que la simulación comience a funcionar.

4.-EJEMPLO

En este ejemplo se podrá observar cómo se define un símbolo nuevo para un Flip-Flop T (ver Figura 1), teniendo en la librería de símbolos ya definidos un FFD y una puerta XOR.



Primero de todo, se especificarán los símbolos de la librería (se supone que estos símbolos ya vendrán definidos en una librería, pero se muestran aquí para poder observar cómo han sido definidos).

```
(0 symbol ( DFF
   inputs (D CK)
   outputs (Q QZ)
   fanin (0.025 0.035)
   fanout (1.25 1.16)
   truthtable (
         1 R 1 0    {D CK Q QZ}
         0 R 0 1
         x 0 s s
         x 1 s s
   )
   timing_constraints (
         setup d ck 2.26 1.89 0 0   {min, typ, max, mil}
         hold d ck 1.94 1.62 0 0
         pwl ck ck 2.90 2.43 0 0
         pwh ck ck 2.58 2.16 0 0
   )
   propagation_delays (
         tplh CK Q 0.65 1.32 2.70 3.23
         tphl CK Q 0.69 1.40 2.86 3.42
         tplh CK QZ 0.45 0.92 1.87 2.24
         tphl CK QZ 0.49 0.99 2.03 2.43
         dtplh CK Q 0.45 0.92 1.87 2.24
         dtphl CK Q 0.34 0.69 1.41 1.69
         dtplh ANY QZ 0.49 1.00 2.04 2.43
         dtphl ANY QZ 0.36 0.73 1.50 1.79
   )
   )
)
```

donde en la especificación de la tabla de verdad:

- "1 R 1 0 {D CK Q QZ}" quiere decir que cuando D esté a alta y haya un flanco de subida en CK, Q tiene que valer 1 y QZ un 0.

y

- "x 0 s s" quiere decir que si CK está a baja y D no importa lo que valga, en Q y QZ se mantienen los estados.

Definición de la XOR:

```
(0 symbol( XOR
   inputs(A B)
   outputs(Y)
   fanin(0.050 0.028)
   fanout(1.10)
   specifications(
         y=or(and(not(a) b) and(a not(b))
   )
   propagation_delays(
         tplh A Y 0.31 0.64 1.31 1.56
```

```
                tphl A Y 0.46 0.94 1.93 2.31
                tplh B Y 0.35 0.72 1.47 1.76
                tphl B Y 0.46 0.93 1.89 2.26
                dtplh ANY Y 0.52 1.05 2.15 2.58
                dtphl ANY Y 0.38 0.77 1.57 1.88
        )
        )
    )

{ Definición del símbolo TFF.}
    (1 new_symbol( tff
       new_inputs(T CK)
  new_outputs(Q QZ)
       new_place(XOR T Q a)
       new_place(DFF a CK Q QZ)
     )
    )

{ Y para simularlo: }
   { Se coloca el FF T en el circuito (es el único símbolo del circuito)
    (1 PLACE (tff A CK Q QN) )

    { Comienza la simulación }
    (2 START( ) )
```

5.-CONCLUSIÓN

Este simulador sirve para que los alumnos puedan ejercitar y practicar con un simulador en el que conozcan perfectamente su funcionamiento y formatos de ficheros. Además, este simulador permitirá el que se puedan hacer proyectos final de carrera que amplíen en diferentes herramientas la potencia de este simulador. De hecho ya se ha desarrollado una herramienta gráfica para definir los circuitos y otra para ver las waveforms.

6.-REFERENCIAS

[1] Stroustrup, B. "The C++ Programming Language". Addison-Wesley Publishing, Reading, MA, 1986.
[2] "Turbo C++". Borland International Inc., Scotts Valley, CA, 1990.
[3] Abramovici, M.A., Breuer, M., Friedman, A.D. "Digital Systems: Testing and Testable Design", 1990.
[4] B.P.Zeigler, S.Chi "Symbolic Discrete Event System Specification", IEEE Transactions on Systems, Man and Cybernetics, Nov/Dec., pp. 1428-1443, 1992.
[5] R.F.Garzia, M.R. Garzia, and B.P. Zeigler, "Discrete-event Simulation", IEEE Spectrum, Dec., pp.32-36, 1986.

El material empleado ha sido un PC Pentium y los compiladores Borland C++ 3.1 y Borland C++ 4.0.