Diseño de circuitos secuenciales en OPAL

Capturar un diseño secuencial en OPAL es muy sencillo y puede hacerse a partir del diagrama de estados de la máquina secuencial que deseamos implementar; eliminando así la necesidad de elaborar y simplificar las ecuaciones para las entradas de cada flip-flop requerido.

Al igual que en el caso de un circuito combinacional, el programa en OPAL comienza con un header en el que puede incluirse la descripción del diseño e inmediatamente después las definiciones del dispositivo, pines y constantes. Como sabemos, un circuito secuencial requiere forzosamente de una entrada de reloj que indique a los elementos de memoria cuando trabajar. En el caso de trabajar con una GAL la entrada de reloj se debe definir como una entrada en el pin 1 del dispositivo. También en esta sección, para un circuito secuencial, es importante definir los bits de estado, los cuales serán los nombres con los que nos referiremos a cada uno de los bits que juntos indicarán el número de estado en el que se encuentra la máquina secuencial que implementemos. Para definir estos bits de estado se utiliza la palabra reservada *STATEBITS* seguida de los nombres que deseemos asignar a cada bit separados por una coma. Es importante mencionar que estos bits de estado también se tomarán como salidas del circuito, por lo que es posible asignarles un pin del mismo modo que se hacía con las salidas combinacionales; es decir con un signo de igual y un número de pin válido justo después del nombre del bit. Esto puede verse en el ejemplo siguiente:

```
BEGIN HEADER
Contador ascendente de 0 a 5 con reset y habilitador
END HEADER
BEGIN DEFINITION
DEVICE GAL22V10;
INPUTS reloj=1,rst=2,cnt=3; { La entrada que servirá de reloj es
                              colocada en el pin 1 }
STATEBITS SB2=23,SB1=22,SB0=21;  { Se declaran los bits de estado
                                   SB2,SB1    y    SB0.    Ellos
                                   determinarán el estado en que
                                   se encuentra el circuito. Se
                                   les asignan los pines de
                                   salida     23,22     y     21
                                   respectivamente }
...
```

Otra definición importante para implementar un circuito secuencial es la de los nombres de estado. Estos nombres identifican a cada uno de los estados y se asocian con un número en decimal. Para definirlos es utiliza la palabra reservada *STATE_NAMES* seguida por los nombres de los estados deseados con su respectivo valor en decimal separados entre sí por comas. Se dice que el circuito se encuentra en uno de estos estados cuando su valor decimal corresponde con el valor en binario de los bits de estado. Esto puede verse con más claridad tomando en cuenta la siguiente línea de código que precedería al ejemplo anterior:

```
...
STATE_NAMES cero=0,uno=1,dos=2,tres=3,cuatro=4,cinco=5;
{ En esta línea se definen los nombres para los estados. Por
   ejemplo, definimos que con la palabra “tres” nos referiremos
   al estado 3 decimal, es decir a que los bits de estado valen
   011 respectivamente }
...
```

Cuando se trabaja con circuitos secuenciales es recomendable agrupar a los bits de estado dentro de un set. Esto no es obligatorio pero facilita la programación al trabajar con ellos. Un set también debe ser declarado dentro de la definición del archivo .OPL como veremos en el próximo ejemplo de código. Con esto terminamos la parte que se refiere a la definición. El siguiente cambio importante entre un diseño combinacional y un secuencial ocurre en el bloque de ecuaciones. Este siempre es requerido al trabajar con un circuito secuencial, no importa si las salidas combinacionales se definen usando una tabla de verdad; esto es porque en este bloque se especifica que entrada se utilizará como reloj. Para ello utilizamos el comando .c aplicado al set de los bits de estado, igualando este con la entrada especificada en el pin uno (el reloj). Esto se muestra a continuación:

```
...
SET cuenta=[SB2,SB1,SB0];   { Se define el set “cuenta” como el
                              conjunto de los bits de estado }
END DEFINITION
BEGIN EQUATIONS
cuenta.c=reloj;   { Se asigna la entrada reloj (pin 1) como el
                    reloj para los bits de estado }
END EQUATIONS
...
```

Con esto hemos terminado de definir la estructura básica de un diseño secuencial en OPAL y a continuación trataremos los puntos necesarios para definir el comportamiento de este circuito a través de un diagrama de estados.

El comienzo de un diagrama de estados se especifica en OPAL utilizando la directiva *STATE_DIAGRAM* al igual que los bloques de definición ó de ecuaciones, el diagrama de estados debe estar limitado por las directivas *BEGIN* y *END*. A diferencia de las directivas que marcan el inicio de otros bloques, en el caso de un diagrama de estados es necesario incluir al final el nombre del set que contiene los bits de estado y entre paréntesis el nombre de cada uno de estos en orden de más a menos significativo. Posteriormente es necesario definir lo que ocurre en cada uno de los estados declarados como *STATE_NAMES* al inicio del código. Esto se hace utilizando la palabra reservada *STATE* seguida del nombre del estado y de las instrucciones a realizar cuando los bits de estado alcancen este valor. Si deseamos realizar una instrucción no importando en que estado se encuentra el circuito utilizamos a modo de nombre de estado la palabra reservada *ALL*.

Una ventaja de OPAL que facilita mucho el diseño de un circuito secuencial es el uso de una estructura del tipo *IF...THEN...ELSE...* esto permite analizar las entradas del circuito y si toman determinado valor tener la posibilidad de cambiar a algún otro estado. Esto evidentemente simplifica mucho el trabajo del diseñador, ya que elimina la necesidad de realizar tablas de verdad y simplificar ecuaciones para calcular los estados siguientes en función del estado presente y las entradas. Todo esto se puede observar en el siguiente fragmento de código en el cual se define el comportamiento de un contador ascendente de 0 a 5 con un habilitador (cnt) y un reset (rst):

```
...
BEGIN STATE_DIAGRAM cuenta(SB2,SB1,SB0)  { Inicio del diagrama de
                                           estados. Los bits de estado
                                           en orden de más a menos
                                           significativo se especifican
                                           como SB2,SB1 y SB0 }
STATE ALL: IF rst THEN cero;  { No importa el estado actual, si
                                rst=1 pasa al estado 'cero' }
STATE cero: IF cnt THEN uno;
STATE uno: IF cnt THEN dos;
STATE dos: IF cnt THEN tres; { Si se encuentra en el estado 'dos'
                               y cnt=1 pasa al estado 'tres' }
STATE tres: IF cnt THEN cuatro;
```

```
STATE cuatro: IF cnt THEN cinco;
STATE cinco: IF cnt THEN cero; { Si el estado actual es 'cinco' y
                                 cnt=1, regresa al estado 'cero' }
END STATE_DIAGRAM  { Final del diagrama de estados }
...
```

De esta manera podemos implementar muchos circuitos secuenciales sencillos en OPAL para programarlos en un PLD como una GAL sin la necesidad de realizar un diagrama de estado siguiente y simplificar las ecuaciones para cada flip-flop. Además OPAL nos permite simular nuestro diseño al igual que con un circuito combinacional antes de programar un PLD. Esto se logra con un bloque de vectores de prueba al igual que con un circuito combinacional. La única diferencia importante en este aspecto se encuentra en la forma de declarar la entrada del reloj, para lo cual se utiliza el valor 'c' en lugar de un 1 ó un 0 como se hace con las entradas. Esto se aprecia en el siguiente ejemplo:

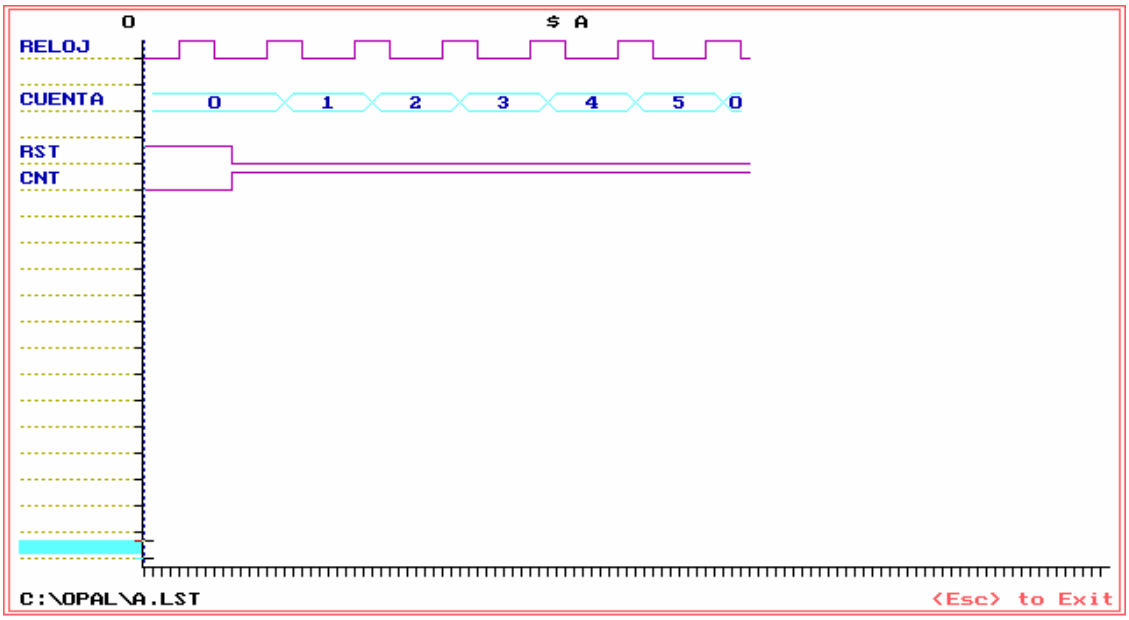
```
...
BEGIN VECTOR
reloj,rst,cnt,cuenta(hex);     { Definimos el orden en que
                         acomodaremos las entradas y agregamos el
                         set de los bits de estado para que OPAL
                         nos muestre su valor en hexadecimal }
c10x  { Estímulos de entrada, especificamos el reloj con el valor
        de 'c'. En este caso se hace un reset }
c01x  { Se realiza una cuenta, el valor del reloj es 'c' }
c01x
c01x
c01x
c01x
c01x
END VECTOR
```

Con esto hemos cubierto todos los puntos importantes del diseño de circuitos secuenciales en OPAL desde el bloque de definición hasta algunos aspectos de la simulación al implementar un circuito sencillo (contador ascendente 0 a 5). Por último es importante mencionar que al programar un PLD como una GAL para que haga la tarea de un circuito secuencial debemos conectar al pin 1 el reloj y tener cuidado de conectar a tierra (0 V) el pin asignado como *Output Enable* ya que de lo contrario nunca obtendremos una salida. En el caso del circuito que diseñamos a lo largo de esta guía, el habilitador de salidas se encuentra en el pin 13, ya que definimos el dispositivo a utilizar como una GAL22V10. Es importante tener cuidado con esto ya que en el archivo .LOG generado por OPAL al sintetizar el circuito no aparece ninguna etiqueta en este pin como se muestra a continuación:

```
                Chip diagram (DIP)

                 .______    ______.
                 |      \__/      |
          reloj  |  1          24 |  VCC
          rst    |  2          23 |  SB2
          cnt    |  3          22 |  SB1
                 |  4          21 |  SB0
                 |  5          20 |
                 |  6          19 |
                 |  7          18 |
                 |  8          17 |
                 |  9          16 |
                 | 10          15 |
                 | 11          14 |
            GND  | 12          13 |
                 |________________|
```

Por último, la siguiente imagen muestra el resultado de la simulación de este circuito ejemplificando nuevamente el despliegue de los datos en hexadecimal, así como el resultado de utilizar el valor ‘c’ para definir una señal de reloj.



OPAL significa Open Programming Architecture Language, es un lenguaje de descripción de hardware que nos permite implementar diseños completos en un solo dispositivo lógico programable (PLD) como por ejemplo una GAL. Esto tiene la ventaja de reducir el número de componentes necesarios de varias compuertas y otros dispositivos a un solo circuito integrado que realiza justo la tarea que nosotros definimos vía software.

Si queremos implementar un circuito en OPAL, podemos utilizar básicamente tres métodos para describir el comportamiento del mismo: captura por tabla de verdad ó captura por ecuaciones (para circuitos combinacionales) y la captura por diagramas de estados (para circuitos secuenciales). El código para OPAL puede escribirse utilizando cualquier editor de texto simple que permite guardar un archivo con extensión ".OPL". A continuación presentamos los pasos para capturar diseños combinacionales de las dos formas que nos permite OPAL, así como diseños secuenciales. En cada una de estas secciones incluimos una explicación del procedimiento necesario para llevar a cabo una simulación de nuestro circuito dentro del mismo OPAL y observar así el comportamiento que tendrá antes de programarlo en un PLD.

Diseño de circuitos combinacionales en OPAL

Los siguientes son los pasos necesarios para escribir el código en OPAL que permite definir el comportamiento de un circuito combinacional a través de su tabla de verdad. Presentaremos estos pasos a modo de una lista en la que cada punto explica y ejemplifica cada uno de los bloques principales del código de OPAL.

1. BEGIN HEADER
   {Esta directiva nos permite escribir cualquier tipo de información que identifique nuestro código}
   END HEADER

2. BEGIN DEFINITION
   {Esta directiva nos abre el campo que nos permite declarar el dispositivo que utilizaremos, nuestras variables de entrada y de salida.}
   DEVICE GAL22V10;
   {Especificamos el dispositivo en que programaremos nuestro diseño terminando con punto y coma}

INPUTS a=5, b=6, c=7;
{Esta palabra reservada nos permite declarar las variables de entrada de nuestro circuito combinacional, si queremos asignarle pines debemos igualar la variable al número de pin en el que queramos que esté teniendo cuidado de que éste sea válido como entrada}
OUTPUTS(com) x=15, y=16 ,z=17;
{Esta palabra reservada nos habilita la declaración de las variables de salida de nuestro circuito, le agregamos "(com)" para indicar que se tratan de salidas combinacionales, y al igual que en las entradas podemos asignarles un pin}
END DEFINITION

3. BEGIN TRUTH_TABLE
   {Esta directiva nos abre el campo para poder escribir nuestra tabla de verdad}
   TTIN a,b,c;
   TTOUT x,y,z;
   {Las palabras reservadas TTIN y TTOUT nos sirven para especificar cuales son nuestras variables de entrada y de salida respectivamente a nuestra tabla de verdad, así como el orden en que capturaremos sus valores}
   000 000
   001 001
   010 010
   011 011
   100 100
   101 101
   110 110
   111 111
   {En esta parte escribimos tal cual nuestra tabla de verdad iniciando con las variables de entrada y después las de salida en el mismo orden en que las escribimos después de las palabras reservadas TTIN y TTOUT}
   END TRUTH_TABLE

Como podemos ver, el tener la opción de escribir directamente la tabla de verdad en nuestro código nos facilita mucho el trabajo para diseñar un circuito ya que no tenemos que simplificar ecuaciones. Sin embargo su mayor desventaja es que la tabla de verdad crece a razón de 2^n siendo n el número de variables de entrada, de modo que si queremos diseñar un circuito que necesite de 8 entradas nuestra tabla de verdad constaría de 256 combinaciones. Por esto en algunos casos resulta conveniente realizar una captura por medio de ecuaciones. Nuevamente mostramos a modo de lista los pasos necesarios para describir un circuito en OPAL mediante ecuaciones.

1. BEGIN HEADER
   {Esta directiva nos permite escribir cualquier tipo de información que identifique nuestro código a modo de encabezado}
   END HEADER

2. BEGIN DEFINITION
   {Esta directiva nos abre el campo que nos permite declarar el dispositivo que utilizaremos, nuestras variables de entrada y de salida.}
   DEVICE GAL22V10;
   {Especificamos nuestro dispositivo terminando con punto y coma}
   INPUTS a=5, b=6, c=7, d=8;
   {Esta palabra reservada nos permite declarar las variables de entrada que nos servirán para formar nuestras ecuaciones, si queremos asignarle algún pin en especial, debemos igualar la variable al número de pin en el que queramos que esté.}
   OUTPUTS(com) w=15, x=16, y=17, z=18;
   {Esta palabra reservada nos habilita la declaración de las variables de salida, es decir, a lo que serán igualadas nuestras ecuaciones, le agregamos "(com)" para indicarle al sintetizador que nuestro circuito es combinacional.}
   SET in1=[a,b];
   SET in2=[c,d];
   {OPAL nos permite declarar set's de variables. Estos son nombres con los que nos referimos a un conjunto de variables y nos ayuda a trabajar con múltiples entradas ó salidas al mismo tiempo}
   END DEFINITION
   {Como podemos ver hasta ahora, el código es idéntico al que utilizamos para la tabla de verdad}

3. BEGIN EQUIATIONS
   {Esta directiva nos abre el campo para poder escribir nuestras ecuaciones}
   Nodo1=a*b*c;
   {OPAL nos permite declarar nodos en nuestro programa, los cuales sólo se usan dentro de nuestro dispositivo, no tienen una salida en ningún pin. Si utilizamos nodos, es recomendable declararlos antes de nuestras ecuaciones para llevar un orden, pero esto no es un requisito}
   w=a+b;
   x=(a+b)$(/c*b);
   y=(a==c);
   {OPAL permite las operaciones lógicas: AND (*), OR (+), XOR ($), NOT (/) y comparaciones como igual (==), diferente (!=), menor (<), mayor (>), menor ó

igual (<=) y mayor ó igual (>=); así como agrupar términos con el uso de paréntesis}
z=(in1==in2);
{También es posible trabajar con set's en lugar de variables. De esta manera se facilita la captura de las ecuaciones ya que podemos comparar varios bits a la vez}
END EQUATIONS

Como podemos ver, OPAL nos permite especificar el comportamiento de un dispositivo mediante ecuaciones con mucha facilidad, pero limitándonos únicamente al uso de operaciones lógicas y comparaciones.

Además de estos dos métodos de captura para describir circuitos combinacionales, OPAL nos permite realizar una simulación del comportamiento de nuestro circuito antes de programarlo en un PLD y de realizar el cableado necesario. Para llevar a cabo esta simulación debemos incluir al final de nuestro código un vector de prueba utilizando la directiva *VECTOR*. A continuación se muestra un ejemplo de cómo podría utilizarse esta directiva asumiendo que nuestro circuito tiene las entradas A, B y C y las salidas X, Y y Z. Este fragmento de código debe incluirse al final del resto del programa.

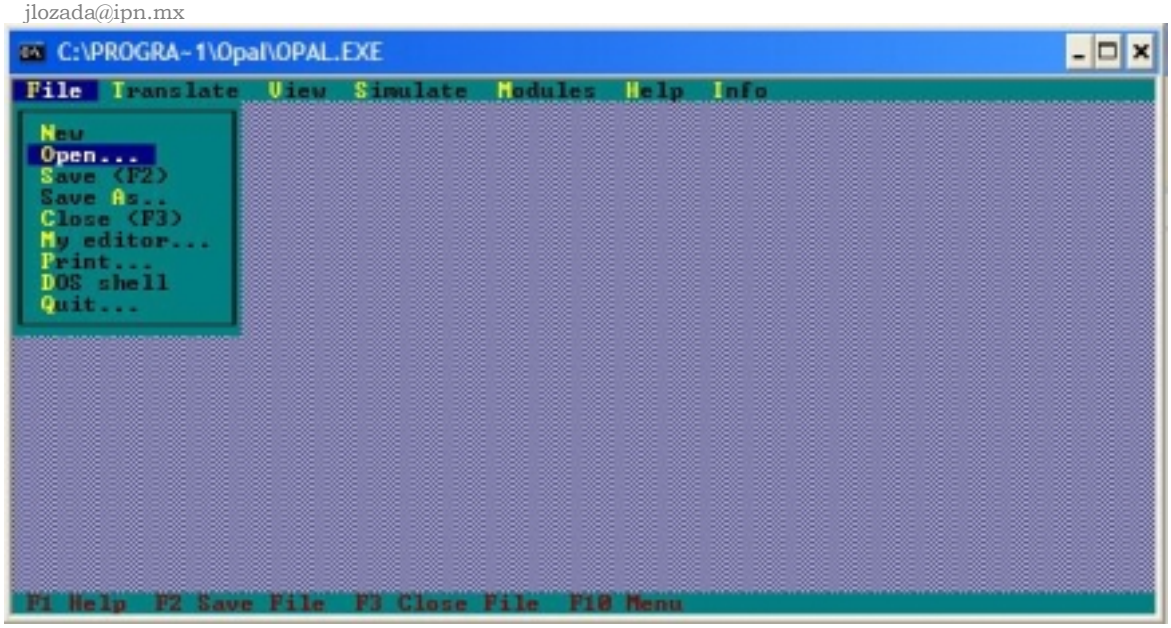
```
BEGIN VECTOR
A,B,C;
000
010
011
101
111
END VECTOR
```

En este fragmento de código debemos inicialmente rescribir las entradas a nuestro circuito, en el orden en que deseamos trabajar con ellas. En los siguientes renglones colocamos varios valores de los estímulos que queremos ver en las variables de entrada. Al momento de sintetizar nuestro código como veremos más adelante, OPAL generará automáticamente un archivo .LST donde se encontrará la información de la simulación. Podemos utilizar la información de este archivo para que OPAL nos muestre un diagrama de tiempos con las salidas de nuestro circuito y sus entradas con los estímulos especificados en los vectores de simulación. En este ejemplo la entrada A tendrá un valor de 0 los 3 primeros intervalos y de 1 los últimos 2, mientras que la entrada C tendrá un 0 los primeros 2 intervalos y un 1 los últimos 3.

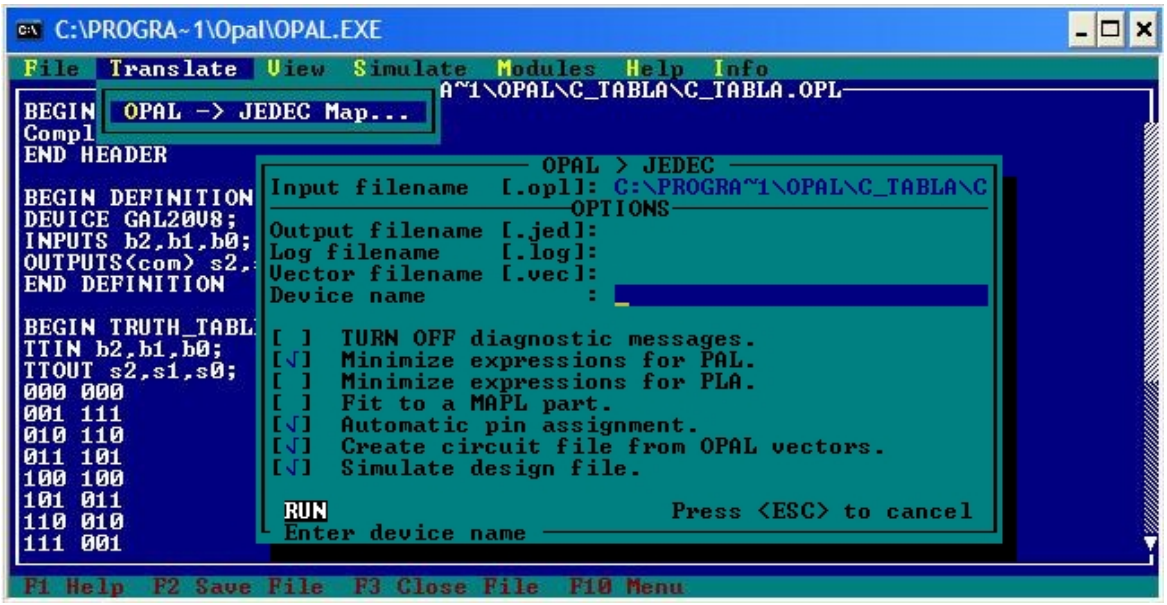
Este tipo de simulación resulta muy conveniente para verificar el funcionamiento de algunos circuitos sencillos ya que nos evita la necesidad de programar el dispositivo antes de saber si cometimos algún error al simplificar las ecuaciones ó durante la captura del programa. Sin embargo, en algunos casos resulta muy incomodo tener que seguir los valores de todo un grupo de variables como en el caso de un sumador ó de un multiplicador. Para estos casos es posible obtener una simulación en la cual todo un set de variables aparezca representado en hexadecimal sobre el mismo diagrama de tiempos. Para ello debemos definir un set con las salidas correspondientes ordenadas de más a menos significativa al inicio de nuestro programa (en el bloque de definition). Después debemos incluir este set como una de las variables de nuestro vector de prueba, colocando su nombre seguido por la palabra "hex" entre paréntesis. Por último es necesario indicar el estímulo que correspondería al set de salidas como "no importa" que se declara como una x. El siguiente ejemplo muestra esto suponiendo un circuito con las entradas A y B y un set de salidas S:

```
BEGIN VECTOR
A,B,S(hex);
00x
01x
10x
11x
END VECTOR
```

Una vez declarado el set de simulación, para observar el resultado primero debemos sintetizar nuestro código. Este procedimiento también generará los archivos .JED que contiene la información para programar el dispositivo utilizando el TopMax ó algún otro programador; y el .LOG que contiene información importante acerca de la utilización del dispositivo y un diagrama con las asignaciones de los pines correspondientes a nuestro diseño. Para sintetizar un código en OPAL lo primero que debemos de hacer es guardar nuestro programa con la extensión .OPL. Después debemos abrir el ejecutable de OPAL y desde ahí seleccionar File → Open… Y luego buscar el archivo .OPL en el directorio en que lo hayamos guardado como se muestra en la siguiente figura:



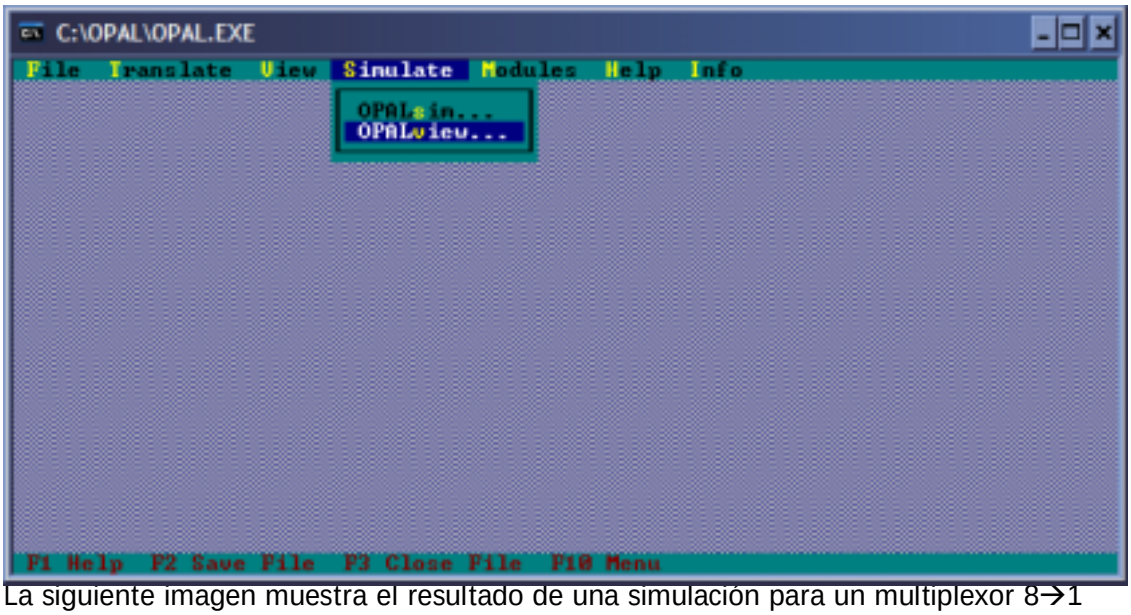
Una vez abierto el archivo con nuestro código, debemos entrar al menú Translate y seleccionar la opción "OPAL → JEDEC Map". Esto mostrará una ventana con una serie de opciones para el sintetizador. Normalmente dejaremos todas estas opciones con su valor por omisión y simplemente elegiremos la opción de RUN. Esta ventana se muestra en la siguiente figura:



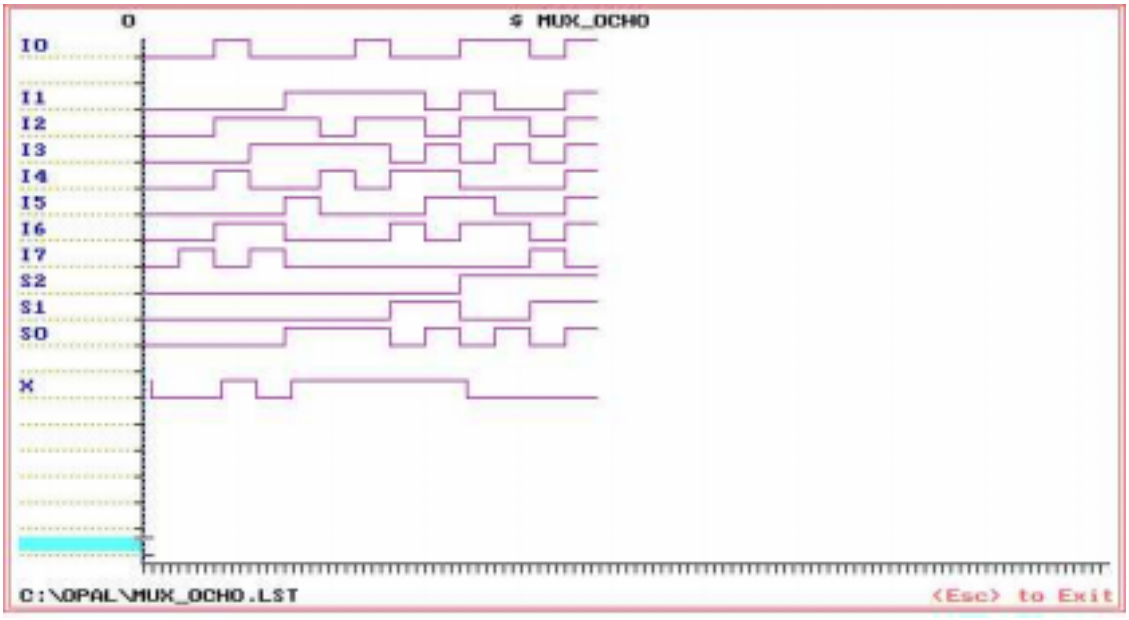
Al presionar RUN se sintetizará nuestro código indicándonos si existe algún error ó generando los archivos finales en caso contrario. De estos archivos los más importantes son como ya mencionamos, el .JED que nos servirá para programar el dispositivo en que

implementemos nuestro diseño; el .LST con la información sobre la simulación de acuerdo con los vectores de prueba que hayamos definido y por último el .LOG con los datos del porcentaje de utilización del dispositivo y el diagrama que muestra las asignaciones de pines.

Por último, si deseamos observar la simulación de nuestro diseño debemos de ir al menú Simulate dentro de OPAL y de ahí seleccionar la opción OPALview... como se muestra en la figura. Luego debemos seleccionar el archivo .LST correspondiente a la simulación que deseamos observar y con esto el diagrama de tiempos aparecerá en la pantalla completa.



La siguiente imagen muestra el resultado de una simulación para un multiplexor 8→1



Y la siguiente ejemplifica como se ve la salida en hexadecimal. Esta simulación es para un multiplicador 3x3.