

1. Descripción del proyecto

Este proyecto tiene varias facetas a desarrollar. Inicialmente lo podemos dividir en tres partes.

Por un lado se trata de desarrollar un entorno gráfico para el diseño y edición de los circuitos.

Por otro lado está la definición de las clases básicas sobre las que se construye un circuito, y que implica la definición de requisitos genéricos para cualquier dispositivo digital.

Finalmente a la hora de diseñar algunos componentes concretos, aparte de el comportamiento genérico indicado en el apartado anterior, puede ser necesario desarrollar elementos específicos. Por ejemplo el componente CPU Indalo III requiere de un ensamblador, un desensamblador, un depurador, etc. y por lo tanto se plantean unos requisitos específicos.

Abordaremos en los siguientes apartados cada una de estas partes del proyecto.

1.1. Diseño y edición de circuitos

Inicialmente en este apartado indicar que solo se han acometido partes concretas de todo el entorno necesario, por lo que simplemente se hará una descripción de los requisitos.

Se pretende disponer de un editor/simulador de circuitos de circuitos digitales. Se dispondrá de un menú de acceso a las funciones del programa, una barra de herramientas para las acciones mas comunes y un panel de montaje donde inicialmente se insertan los componentes y a continuación se trazan las conexiones de los mismos. Los componentes aparecerán en una barra de herramientas jerárquica en la que en el primer nivel dispondremos de diferentes familias: Puertas, circuitos combinacionales, circuitos secuenciales, memorias, ... y cuando se seleccione uno de ellos, se desplegará otra barra de herramientas con todos los componentes de cada familia.

En cuanto a las conexiones, se dispondrá de una herramienta de dibujo que permitirá realizar conexiones entre las diferentes patillas de los circuitos y también con los nodos que se inserten en una línea de conexión, disponiendo de las herramienta clásicas de edición y borrado de cada línea de conexión.

Una vez finalizado el diseño de un circuito se procederá a realizar la simulación de su funcionamiento. Se dispondrá de diferentes elementos de medida (voltímetros, LEDs, osciloscopio, analizador digital,...) para el seguimiento de la evolución del circuito.

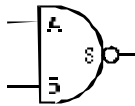
A todos los componentes se le asignará un menú contextual que permitirá visualizar su estado y propiedades. Si un componente está compuesto a su vez por otros componentes, se podrá visualizar su contenido y observar sus líneas internas de conexión y el estado de todos sus elementos.

1.2. Clases básicas para componentes estándar.

Antes de describir estas clases básicas, se enunciará que elementos intervendrán en la simulación y como se relacionan.

Básicamente se dispone de dos elementos fundamentales, por un lado lo que llamaremos *componente* y que se corresponderá con cualquier circuito digital, desde una puerta lógica hasta una CPU o cualquier otro elemento complejo. Y por otro lado lo que denominamos *línea de conexión*, y que como su propio nombre indica sirve para conectar partes de un componente entre sí y/o con otros componentes.

Un *componente* tiene un nombre, una o varias representaciones, un conjunto de *patillas de conexión*, un comportamiento que lo define. Además durante la simulación cada componente tendrá un estado.



Componente puerta AND con las patillas de conexión A, B y S.

Las *patillas de conexión* pertenecen a un *componente*, y son el punto de enlace entre las *líneas de conexión* y los *componentes*. Las *líneas de conexión* pueden ser de *entrada*, *salida* o *entrada/salida*.

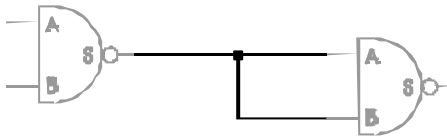
Una *patilla* de salida estándar puede tener solamente los valores '0' ó '1'. Una patilla de salida triestado, puede tener los estados '0', '1' ó 'Z'¹. Cuando se define un componente se especifica de forma clara si tiene las salidas estándar o triestado. Cuando una patilla de salida se conecta a una *línea de conexión*, pone su valor a la línea de conexión con las consideraciones que se harán más adelante.

Una *patilla* de entrada puede estar conectada a una *línea de conexión*, o estar 'al aire' es decir sin conectar. El estado lógico de una línea 'al aire' puede definirse para cada tipo de componente, este puede ser aleatorio, o 1 lógico (como ocurre en las puertas TTL), o como se quiera definir. En esta parte de definición del problema vamos a suponer que es 1 lógico. Una *patilla* de entrada con una *línea de conexión* tendrá el estado de esta línea. Los estados posibles son '0', '1' ó 'Z'. Los dos primeros estados son evidentes, el estado 'Z' es equivalente al de entrada 'al aire'.

Una *patilla* de entrada/salida implica que la salida de esas líneas es de tipo triestado, y que cuando la salida está desactivada, puede funcionar como entrada.

Una *línea de conexión* está compuesta de uno o varios *segmentos* que interconectan *patillas* de componentes. Un *segmento* puede estar constituido por varias rectas que comienzan o finalizan en una *patilla de conexión* o en un nodo. Un *nodo* es un punto de encuentro de tres o cuatro *segmentos*. Una línea de conexión tiene un estado definido, que será '0', '1' ó 'Z'.

¹ El estado 'Z' o de alta impedancia equivale a el aislamiento entre la salida de una puerta y la línea a la que está conectada. A veces se describe como si la salida de la puerta tuviera un interruptor, si este interruptor está cerrado, la puerta funciona de forma convencional (vale '0' ó '1'). Si el interruptor está abierto se dice que la salida está en estado 'Z'. Las salidas que pueden trabajar de esta forma, se denominan salidas *triestado* y son especialmente adecuadas para trabajar con *buses de conexión*. En un *bus* pueden 'escribir' (poner '1' ó '0') varios dispositivos. Si se trata de salidas estándar, unas valdrán '0' y otras '1', y el estado del bus será indefinido. Si se usan salidas triestado, se ponen todas en estado 'Z' excepto la que en un momento determinado se seleccione para 'escribir' en el bus. Los circuitos con salidas triestado disponen de una línea de entrada a la que se suele denominar *Enable* y que sirve para activar las salidas del circuito, o dejarlas en estado 'Z'.



Línea de conexión con tres segmentos y un nodo.

En una *línea de conexión* solamente puede ‘escribir’ información un dispositivo en un momento determinado, es decir, no puede darse el caso que la salida de un componente este poniendo un ‘0’, y otro componente quiera poner un ‘1’. De hecho debería generarse un error cuando se pretendiera conectar dos *patillas* de salida a una misma línea de conexión en la fase de edición del circuito. Esto no es aplicable para salidas triestado. En este caso puede haber varias *patillas* de salida conectadas a la misma *línea de conexión*, pero en un momento determinado solo puede estar activa una única salida. Las situaciones posibles son:

- Que todas las salidas estén en estado de alta impedancia. En este caso el estado de la *línea de conexión* es ‘Z’.
- Que todas las salidas estén en estado de alta impedancia excepto una. En este caso el estado de la *línea de conexión* es el mismo que tiene la salida activada (‘0’ ó ‘1’).
- Que dado el caso anterior, se active la salida de otra puerta. En este caso se debe producir un error, pues dos salidas no pueden escribir en una misma línea. En un circuito real esto pasaría desapercibido si las dos salidas tienen el mismo estado ‘0’ ó ‘1’, y en caso contrario el circuito funcionaría de forma impredecible, ocasionando incluso una avería en una o ambas salidas si la situación se prolonga en el tiempo. Para controlar esta situación de error, sería conveniente que la *línea de conexión*, además de almacenar su estado, si es diferente de ‘Z’, debería almacenar cual es la salida que esté poniendo este valor. Cuando se quiera escribir en la línea, si es la salida activa, se procede a anotar el nuevo valor. Si es otra salida, se dispara el error. Este error se puede mostrar destacando la *línea de conexión*, o por medio de consola de salida.

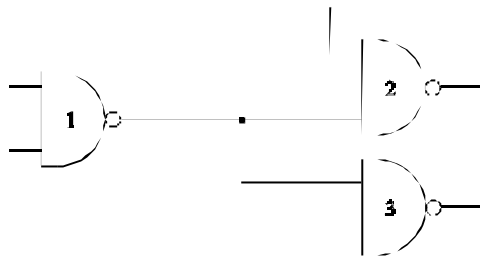
1.3. Modo de funcionamiento de los componentes

Una vez definidos los elementos básicos que entran en juego, vamos a describir como es su interacción. Cada *componente* tiene definido un comportamiento que indica los valores que tendrán sus salidas en función del valor de sus entradas y si corresponde, de su estado anterior. Un componente recalcula el valor de sus salidas solamente en el caso de que cambie alguna de sus entradas (de forma inmediata o retardada como se verá más adelante).

Partiendo de esas premisas, se establecerá un sistema de notificación que avise a un componente cada vez que cambie alguna de sus entradas. La responsabilidad de los avisos recaerá en las *líneas de conexión*. Cuando una línea de salida cambia de estado, lo notifica a la *línea de conexión* que corresponda. Esta *línea de conexión* conoce las entradas a la que está conectada y procede a avisar a cada una de ellas de forma secuencial.

Este modo de funcionamiento presenta algunos inconvenientes que se manifiestan sobre todo en circuitos en los que se usa ‘realimentación’ que se produce cuando la salida de un circuito se conecta a una de sus entradas o a circuitos que directa

o indirectamente están conectados con alguna de sus entradas. Este problema se ve claramente en el ejemplo que se muestra en la siguiente figura:



Supongamos que en un momento dado, la salida de la puerta 1 es '0' esto implica que la salida de la puerta 2 es '1', porque hay un '0' en una de sus entradas. Lo mismo ocurrirá en la puerta 3. Si por un cambio en sus entradas, cambia la salida de la puerta 1 a valer '1', esto se notifica a la *línea de conexión* que a su vez deberá avisar a las puertas 2 y 3. Suponiendo que avisa en primer lugar a la puerta 2, el cambio de su entrada a '1', hará que la salida valga '0', este cambio se notificará a su salida que a su vez lo comunicará a la entrada a la que está conectada, y por lo tanto recibirá un '0' que hará que vuelva a cambiar la salida a '1', que cambiará la entrada y así sucesivamente poniéndose la puerta 2 a 'autooscilar'. El programa entrará en un bucle infinito y nunca se avisará del cambio a la puerta 3. Se podría buscar una solución al problema haciendo que en primer lugar una *línea de conexión* avise primero a todas las entradas conectadas, y después los circuitos que hayan tenido un cambio modifiquen su estado, pero a poco que se haga complejo el circuito se tendrían que generar enormes listas de componentes pendientes de activar que podrían generar problemas.

Por otro lado también este modo de trabajo presenta un problema, en el sentido de que en el mundo real, un circuito no cambia su salida de forma inmediata, sino que siempre existe un *tiempo de respuesta* que es el que transcurre desde que se produce un cambio en sus entradas, y este cambio se manifiesta en sus salidas. Cuando se trabaja con circuitos con diferentes *tiempos de respuesta*, esto afecta al comportamiento del circuito y el simulador lo debe reflejar.

Por las dos razones descritas, nos decidimos a incorporar un mecanismo que permita controlar los tiempos de respuesta de los diferentes circuitos. La solución adoptada consiste en la inclusión de un *temporizador* que básicamente consiste en una tabla que contiene una entrada por cada 'instante de tiempo' de nuestro sistema. En principio no hace falta asignar una duración concreta a estos instantes de tiempo, sino que simplemente se puede decir que si dos puertas reciben cambios en sus entradas en un instante determinado, y una es más 'rápida' que la otra. En el *temporizador* se reflejará que en un instante determinado se avisará a una puerta, y uno o varios instantes más tarde se avisará a la otra.

Con este nuevo elemento la secuenciación de los sucesos podría ser la siguiente: En primer lugar vamos a suponer que las dos puertas lógicas tienen un tiempo de respuesta de '10'. Cuando la puerta 1 cambia su salida a '1', se notifica a la puerta 2 dicho cambio. La puerta 2 no cambia de estado de forma inmediata, sino que simplemente deja un aviso al *temporizador* para que la avise dentro de 10 unidades de tiempo. En este punto finaliza el proceso de la puerta 2 y por tanto ahora se notifica el cambio a la puerta 3, que hará lo mismo que 2, es decir, dejará el mismo aviso al *temporizador*. El circuito seguirá evolucionando, y transcurriendo unidades de tiempo. Cuando hayan transcurrido 10 unidades, el *temporizador* avisará a la puerta 2. Esta consultará el valor de las entradas y modificará su salida. Esto hace que vuelva a

cambiar una entrada en puerta 2, la cual volverá a dejar un aviso pendiente en el *temporizador*. Al finalizar el proceso de la puerta 2, el *temporizador* avisará a puerta 3 y así sucesivamente irá evolucionando el circuito, con una simulación más parecida al mundo real.

Cabe destacar que la existencia del *temporizador*, no significa que dentro del mismo haya bucles de espera para que pasen las unidades de tiempo de forma más rápida o más lenta, de hecho el circuito no trabajará de manera uniforme. En el ejemplo descrito, en las unidades de tiempo en las que se pasan notificaciones a los *componentes* transcurren consumiendo más tiempo de proceso que en aquellas unidades en las que no se debe avisar a nadie.

En las pruebas realizadas con diversos prototipos, este sistema ha funcionado perfectamente. Además a partir de este punto los componentes pueden clasificarse como de dos tipos: De tiempos de respuesta simples y de tiempos de respuesta complejos. En el primer grupo, se pueden incluir las puertas lógicas por ejemplo. En el segundo grupo se incluyen circuitos más complejos, en los que su comportamiento debe ser descrito por medio de cronogramas en los que se precisa como deben secuenciarse las activaciones de sus entradas y salidas. Un ejemplo sería el de una memoria, en la que se tienen en cuenta los tiempos transcurridos desde que se introduce una dirección de memoria estable, o se activa alguna línea de control, o se deben poner datos en la salida, etc. También en las pruebas realizadas, el sistema funciona correctamente.