APLICACIONES PRÁCTICAS

En este artículo dedicado a las aplicaciones prácticas con nuestro Programador para dispositivos CPLD presentado en la revista Nº269 proponemos nuevos proyectos con Quartus II, desde su creación a la programación del CPLD, llevando así a la práctica los conocimientos adquiridos en las anteriores revistas.

En la **revista Nº285** explicamos los fundamentos básicos de la lógica combinacional y secuencial, los pilares de cualquier proyecto digital.

Han sido innumerables las ocasiones que nuestros proyectos han presentado este tipo de elementos bajo forma de **circuitos integrados**.

Retomamos el artículo anterior presentando en primer lugar los **elementos lógicos** más comunes y utilizados, describiendo su **símbolo**, **lógica** y **circuito interno,** en una de las muchas posibles implementaciones, excluyendo los tratados anteriormente (**NOT**, **AND** y **OR**). Después abordaremos los **bloques combinacionales** más **complejos**.

XOR (eXclusive-OR)

Representa una función lógica de **dos entradas** y **una única salida** que vale **1** cuando **únicamente una sola** de sus **entradas** vale **1**.

Una posible interpretación del comportamiento de esta función lógica, bastante difundida ya que se utiliza mucho en la práctica, es que la

salida es un resultado de la **comparación** de las **dos entradas**.

Si se observa la **tabla de la verdad** la **salida** vale **1** cuando el valor de las **entradas** es **diferente**.

Como veremos posteriormente esta característica se toma como base para la construcción de **comparadores**, **sumadores**, etc.

La **tabla de la verdad** de una **XOR** de **dos entradas** es la siguiente:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR (Exclusive-NOR)

Al igual que las funciones **NAND** y **NOR** son las funciones complementarias (inversas o negadas) de las funciones **AND** y **OR**, también la función **XOR** tiene su función **inversa o complementaria**, la **XNOR.**

La **tabla de la verdad** de una **XNOR** de **dos entradas** es la siguiente:

A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

DECODIFICADOR (binario-decimal)

Es un componente que tiene un número **igual** o **superior** a **1** de **entradas** (**n**) y a lo sumo $\mathbf{2^n}$ **salidas**. En base al valor seleccionado en las entradas se habilita una única salida.

Es un circuito ampliamente utilizado para el **direccionamiento binario**, como en el campo de las **memorias**, ya que es posible seleccio-

con PROGRAMADOR CPLD (II)

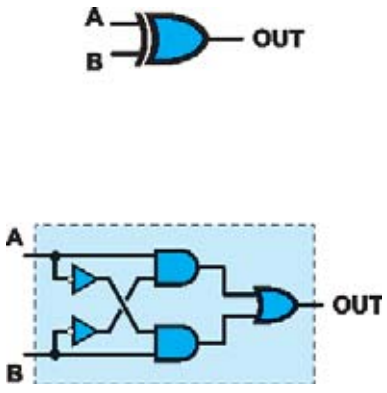


Fig.1 Aquí hemos representado el símbolo y la circuitería interna de una puerta XOR realizada con puertas elementales.

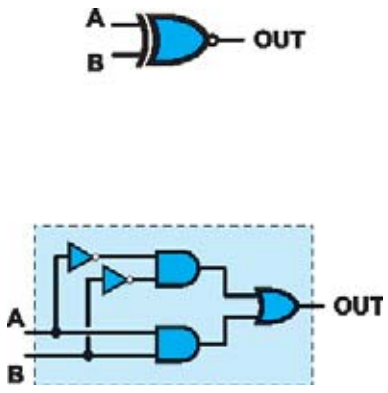


Fig.2 En este caso hemos representado el símbolo y la circuitería interna de una puerta XNOR utilizando también puertas elementales.

nar una de un gran número de salidas con un número exponencialmente más pequeño de entradas.

Por ejemplo, cuando un microprocesador con un **bus de direcciones** de **32 bits** tiene que leer el contenido de una dirección de memoria con "**solo**" **32** señales para **seleccionar una celda** la cantidad de memoria direccionable, gracias al decodificador incluido en las memorias, es de 2^{32} = **4 Gigas**. Por cierto,

Fig.3 Esquema base del componente.

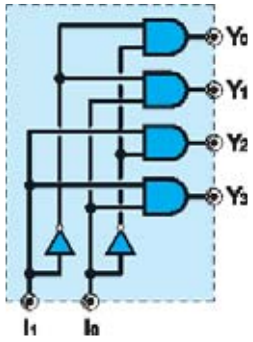


Fig.4 La estructura a interna de un decodificador es un circuito combinacional cuya función es decodificar el código numérico presente en la entrada seleccionando una salida.

este es el motivo que los sistemas operativos diseñados para microprocesadores de 32 bits no pueden direccionar más de **4 GBytes** de **RAM**.

La **tabla de la verdad** de un **decodificador binario-decimal** de **dos entradas** y **cuatro salidas** es la siguiente:

I1	I0	Y0	Y1	Y2	Y3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

CODIFICADOR (decimal-binario)

Se trata de un componente que tiene **"n" salidas** y a lo sumo **2^n entradas**, de las cuales solamente una puede activarse en un momento dado.

En la práctica su funcionamiento es exactamente el **contrario** de un **decodificador**, en base a la entrada activa las salidas asumen su forma binaria codificada.

Al igual que en los decodificadores pueden existir **varios tipos** de **codificación** en función de las **combinaciones de bits** utilizadas. Aquí exponemos la más utilizada, la **binaria**.

La **tabla de la verdad** de un **codificador decimal-binario** de **2 bits** es la siguiente:

I3	I2	I1	I0	Y1	Y0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

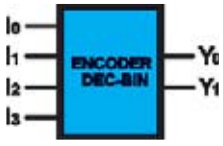


Fig.5 Esquema base del componente.

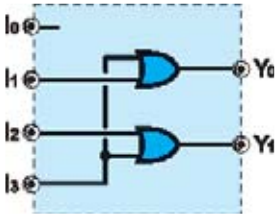


Fig.6 Los codificadores realizan la función inversa de los decodificadores. En este caso se convierte un código decimal de entrada en un código binario en la salida.

Este componente puede tener un **mínimo** de **2 entradas** y un máximo **indefinido.**

Los siguientes elementos que vamos a proponer son **aplicaciones** de los **decodificadores** y de los **codificadores**. De hecho no es extraño que en los **datasheets** de los circuitos se indique que realicen **ambas funciones**, por ejemplo **Decodificador/Demultiplexor**.

MUX (Multiplexor)

Se trata de un componente lógico que tiene **dos** o **más entradas principales**, **una única salida** y un **número** de **entradas** de **control** dependiente biunívocamente del número de entradas principales.

En concreto esta relación es:

N°Entradas = $2^{N°SeñalesControl}$

O lo que es lo mismo, el **número de entradas** es **2** multiplicado por sí mismo **número de señales de control** veces.

Por ejemplo, con **4 señales de control** tenemos:

N°Entradas = 2 x 2 x 2 x 2 = 16

Mediante las señales de control se selecciona una **única entrada** que queda conectada con la salida.

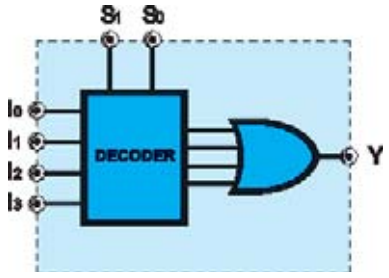


Fig.7 Un multiplexor está constituido internamente por un decodificador con entradas adicionales y con las salidas conectadas a una puerta OR que dispone de un número de entradas igual a las salidas del decodificador.

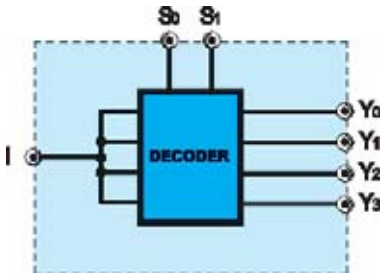


Fig.8 El demultiplexor está formado internamente por un decodificador con sus entradas unidas. Las salidas del decodificador son también las salidas del demultiplexor. La similitud entre un decodificador y un demultiplexor es evidente.

Si, por ejemplo, se tienen **4 entradas principales** son necesarias **2 señales de control** para seleccionar una de las entradas.

Denominando **I0**, **I1**, **I2** e **I3** a las entradas, **S0** y **S1** a las señales de control e **Y** a la salida, la **tabla de la verdad** de un multiplexor de este tipo es la siguiente:

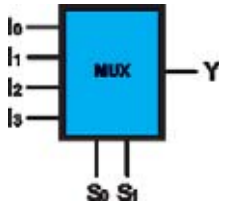


Fig.9 Esquema base del componente.

S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Una de las entradas **I0**, **I1**, **I2** e **I3** se conecta a la salida **Y** en función de los valores asignados a las señales de control **S1 e S0**, tal como se indica en tabla.

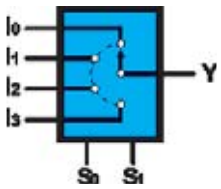


Fig.10 Símbolo del bloque lógico multiplexor. Este componente puede verse como un conmutador que selecciona una de sus entradas a través de las señales de control S0-S1.

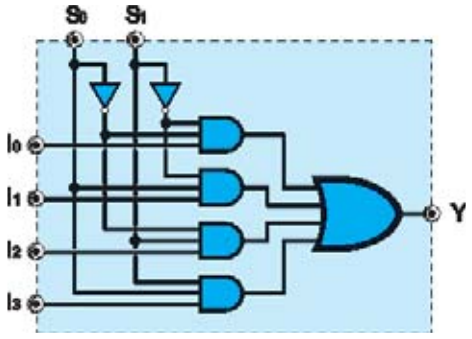


Fig.11 Circuitería interna de un multiplexor utilizando puertas elementales.

Aquí hemos expuesto un multiplexor de **4 entradas**. En general este valor puede estar entre un **mínimo** de **2** y un **máximo indeterminado**.

Evidentemente al **aumentar** el número de **entradas** también **aumenta** de forma proporcional el número de **señales de control**.

DEMUX (DeMultiplexor)

Como se puede intuir por el nombre se trata de un **multiplexor invertido**.

Dispone una única **entrada principal**, una o más **señales de control** y un número de **salidas** dependiente biunívocamente de las señales de control según la siguiente relación:

N°Salidas = $2^{N°SeñalesControl}$

Para un demultiplexor con **dos señales de control** (**S0**, **S1**) y **cuatro salidas** (**Y0**, **Y1**, **Y2**, **Y3**) la **tabla de la verdad** es la siguiente:

S1	S0	Y0	Y1	Y2	Y3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Las **señales de control** permiten seleccionar la **salida** que queda conectada a la **entrada**.

En este componente se parte desde un **mínimo** de 2 salidas hasta un **máximo indeterminado**.

También al **aumentar** el número de **salidas** **aumenta** de forma proporcional el número de **señales de control**.

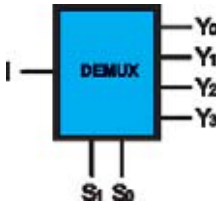


Fig.12 Esquema base del componente.

EL NUEVO PROYECTO

Ahora vamos a diseñar una de las funciones lógicas vistas anteriormente con **Quartus II** partiendo desde cero, un **decodificador binario-decimal** de **2 bits**.

El objetivo de este ejercicio es adquirir cierta destreza en la utilización de las herramientas y utilidades para diseñar con un **CPLD** cualquier función lógica más o menos compleja.

Por ejemplo, diseñando este decodificador habréis reconstruido la mitad de un **74LS139**, un circuito integrado de **16 terminales** que incluye **dos decodificadores** de **2 bits**.

Para simplificar las cosas se ha creado un proyecto vacío que sólo contiene los **ajustes base** para **configurar** correctamente el **CPLD** y los **terminales** de entrada y salida del hardware del kit.

Como en los demás proyectos utilizados anteriormente hay que copiar la carpeta "**decoder**" del CDROM en el disco duro, por ejemplo en una carpeta denominada **C:\CPLD\decodificador.**

Después, una vez lanzado **Quartus II**, hay que abrir el archivo utilizando la función del menú en **FileOpen Project**.

Al hacer click sobre el icono **"decoder**" situado en la parte izquierda de la pantalla la situación ha de ser similar a la mostrada en la Fig.15.

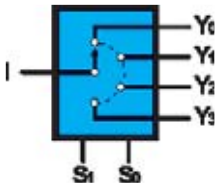
Aquí están presentes exclusivamente los **terminales I/O** y las **salidas** conectadas a **Vcc** ya que, recordando que estamos en lógica negativa, de esta forma permanecen **apagados** todos los diodos LED y el zumbador.

ATENCIÓN: Modificar el **pinout** (**distribución de terminales**) del esquema o intercambiar las entradas con las salidas puede ser destructivo para el **CPLD** ya que el compilador no puede determinar como se conectan los terminales en la placa de circuito impreso**.** Nuestra **distribución** de **entradas** y **salidas** protege de posibles errores de este tipo.

Si cambiamos las configuración de los terminales **Quartus II** podría no señalar errores que en realidad son nuestros porque no hacemos coincidir el diseño con el hardware del circuito impreso, es decir de la **tarjeta LX.1686**.

La salida **beeper** tiene que ser controlada mediante una onda cuadrada, como en el ejemplo **"Self_Test**". Esto es así porque el zumbador emite la frecuencia de control, pero si en entrada tiene un valor bajo fijo se comporta como un cortocircuito entre **Vcc** y

Fig.13 Símbolo del bloque lógico demultiplexor. Este componente puede verse como un conmutador que selecciona una de sus salidas a través de las señales de control S0-S1.



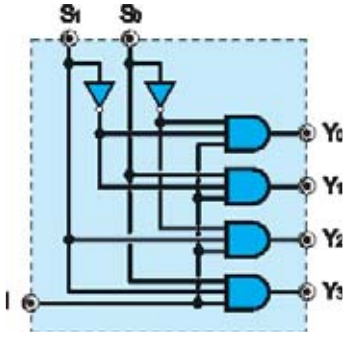


Fig.14 Circuitería interna de un demultiplexor utilizando puertas elementales.

GND que se cierra por el **CPLD,** por lo que podría destruirse el terminal del pin que debería proporcionar una alta corriente.

Dicho esto, empezamos a implementar el circuito del decodificador binario eligiendo, por ejemplo, los pulsadores **P4** y **P5** como entradas y los diodos LED **DL1**, **DL2**, **DL3**, **DL4** como salidas.

Como ya se ha aprendido hay que empezar a insertar los símbolos utilizados para la realización del proyecto, en particular **AND** y **NOT** (se puede insertar uno solo de cada tipo y luego utilizar la acción **copiar-pegar**).

En primer lugar tomamos las **NOT** y **negamos** las entradas y las salidas para trabajar en **lógica positiva**. El resultado se muestra en la Fig.16.

Siguiendo el esquema de un decodificador estándar y considerando por comodidad visual de asociación de los componentes del kit, ponemos **P5** como **bit0**, **P4** como **bit1** y **DL1-4** conectados a las salidas habilitadas del decodificador de forma creciente. El esquema resultante debería ser parecido al mostrado en la Fig.17.

Una vez completado y programado el **CPLD** deberíamos ver:

DL1: Encendido cuando ninguno de los dos pulsadores es accionado. Indica la selección de la primera salida que corresponde a la combinación de entrada "**00**".

DL2: Se enciende al accionar únicamente **P5**. Corresponde a la combinación de entrada "**01**".

DL3: Se enciende al accionar únicamente **P4**. Corresponde a la combinación de entrada "**10**".

DL4: Se enciende al accionar simultáneamente **P4** y **P5**. Corresponde a la combinación de entrada "**11**".

Hay que tener presente que la **señal de reloj** (**entrada clock**) está físicamente presente en cuanto se conecta el **CPLD** al circuito impreso, si bien en este proyecto no se utiliza al tratarse de un **circuito puramente combinacional**.

En lo concerniente a las puertas **XOR** y **XNOR** proponemos un ejemplo sencillo mediante el cual se pueden probar las tablas de la verdad.

Utilizando como entradas los pulsadores **P2** y **P3** se puede verificar la **tabla de la verdad** de la puerta **XOR**. De forma similar se puede proceder con la puerta **XNOR** utilizando los pulsadores **P4** y **P5.**

Las salidas de ambas puertas, **XOR** y **XNOR**, se pueden visualizar, respectivamente, sobre los diodos LED **DL1 y DL2.**

Ejercicio 1

A menudo los bloques lógicos en general, y los decodificadores en particular, disponen de una señal de control denominada **Output Enable** que **valida** o **inhibe** el **bloque**.

Aquí proponemos **modificar** el circuito del decodificador para que disponga de la señal de control **Output Enable**.

Sugerencia: Las puertas **AND** de **2 entradas** pueden ser **sustituidas** por puertas **AND** de **3 entradas** y así tener una señal adicional para controlar la lógica de funcionamiento.

Ejercicio 2

Implementar un **codificador binario** de **2 bits** utilizando **P2-P3-P4-P5** para las entradas y **DL1-DL2** para las salidas.

Los estados han de ser los siguientes: Si se activa **P5** los dos diodos LED han de estar **apagados**, si se acciona **P4** se ha de encender únicamente **DL1**, si se acciona **P3** se ha de encender únicamente **DL2** y, por último, si se activa **P2** se han de **encender** los dos diodos LED.

¿Cómo funciona Quartus II?

En este epígrafe vamos a detallar las fases y operaciones a realizar con **Altera Quartus II** para **programar adecuadamente** un **CPLD**.

Las operaciones fundamentales son comunes a todos los programas que tienen como entrada el **código de descripción del**

Nota: Se puede obtener una **actualización** del programa **Altera Quartus II** en:
https://www.altera.com/support/software/download/sof-download_center.html

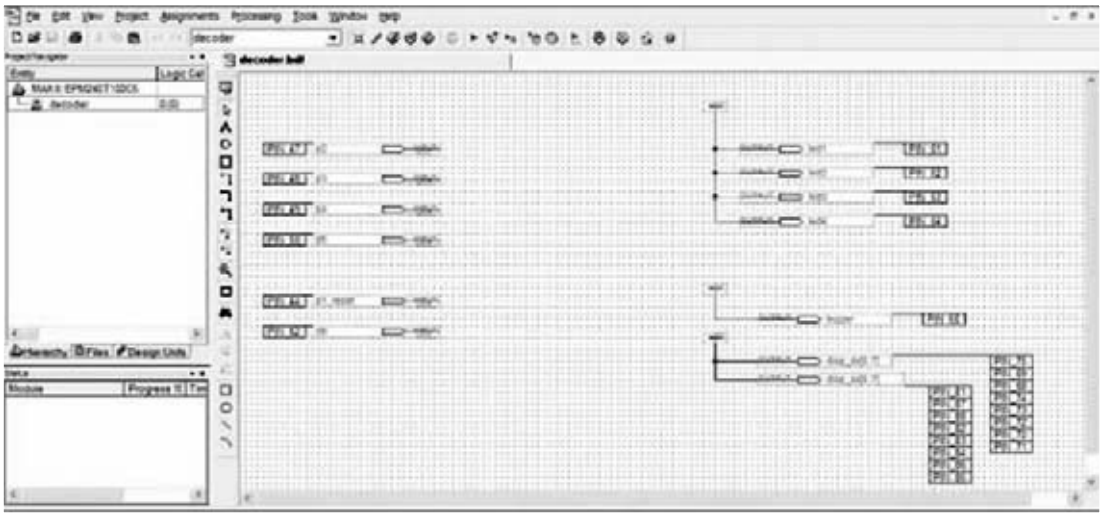


Fig.15 Distribución de las entradas y salidas dentro del proyecto en Quartus II. Esta distribución de entradas y salidas corresponde a la tarjeta LX.1686, por lo tanto, tomando este punto de partida, podéis comenzar a crear vuestros proyectos.

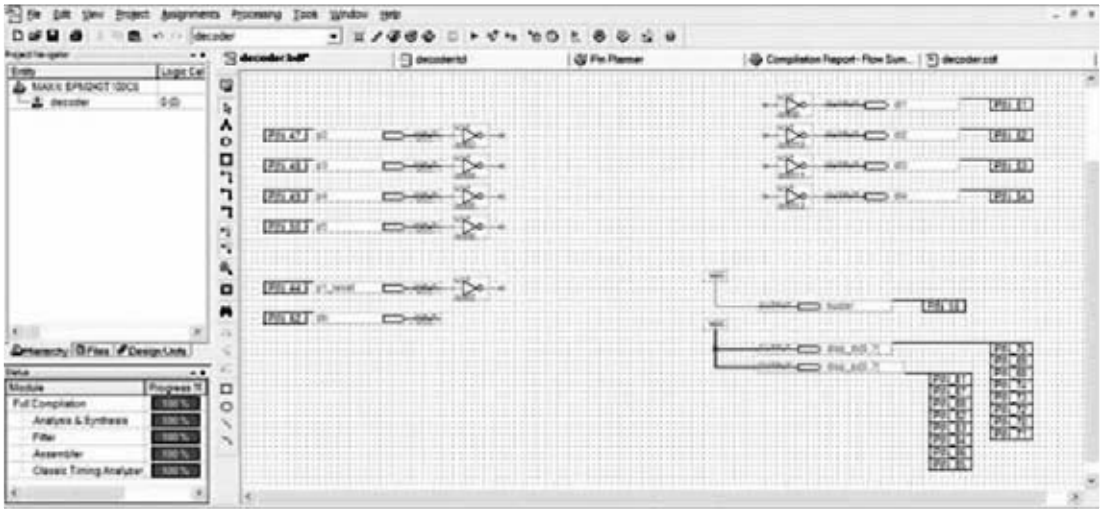


Fig.16 Aquí se muestra la inserción de las puertas NOT en las entradas y en las salidas. De esta forma se puede trabajar con lógica positiva.

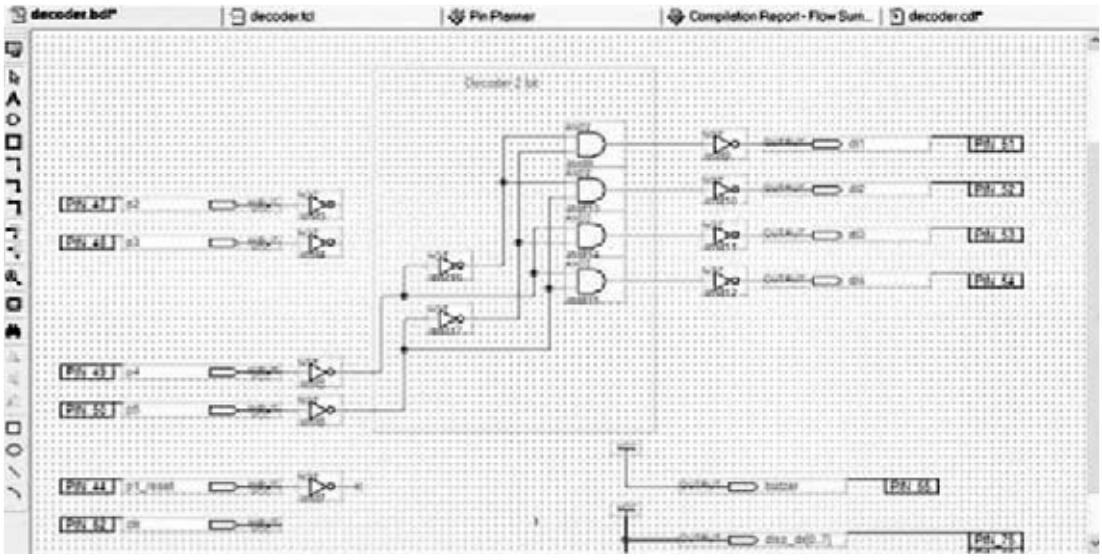


Fig.17 Aspecto del esquema del decodificador binario-decimal de 2 bits diseñado con Quartus II. A partir de la combinación lógica de entrada se enciende un único diodo LED respondiendo a la lógica de la tabla de la verdad.

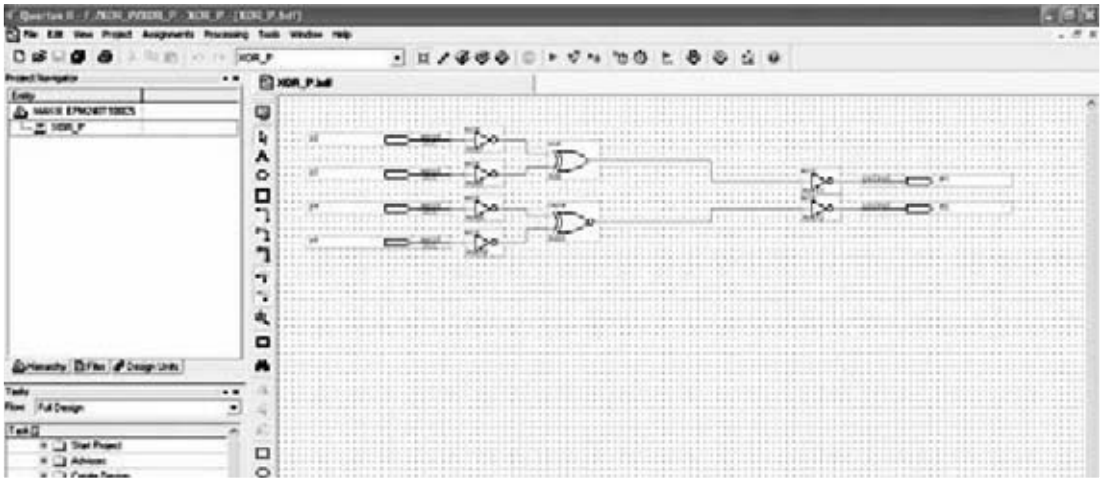


Fig.18 Una vez realizado el circuito, y mapeado en el CPLD, se puede probar su funcionamiento. Se pueden utilizar pulsadores para proporcionar los valores lógicos 0-1 y ver sobre diodos LED el resultado de las operaciones lógicas.

hardware y devuelven como salida los **archivos de programación** para **CPLD** y **FPGA**.

El proceso está compuesto por **5 fases**, que se detallan seguidamente.

1. Creación del proyecto

En este paso se define el desarrollo del proyecto.

Hay que tener presente que el circuito se puede describir de **tres formas diferentes**, no excluyentes y perfectamente integradas:

- Mediante un **esquema**, insertando e interconectando gráficamente bloques simples, tales como puertas lógicas elementales **AND, OR**, **NOT**, etc., y elementos más complejos como **registros**, **contadores**, **bloques aritméticos**, etc.

- Mediante **megafunciones** o **M**ódulos **P**arametrizables de **L**ibrería (**LPM**), insertando macrobloques sintetizados y optimizados de Altera que realizan funciones complejas. Estos módulos permiten parametrizar potentes funciones que se pueden utilizar en nuestros proyectos.

- Mediante **escritura de código**, fundamentalmente en **VHDL** y **VERILOG**. Utilizaremos principalmente el **VHDL** por múltiples razones: Es **universal** y **portable**, en el sentido que no está ligado a un entorno de desarrollo concreto o a un **PLD específico**, si se escribe correctamente es sintetizable para cualquier dispositivo. Además es más **potente** y **flexible** que un **esquema**, ya que es preciso conocer los componentes que forman el circuito a realizar, basta con describir su comportamiento.

Una vez completada la descripción del proyecto se pueden definir **vínculos** para el **compilador**, como por ejemplo la disposición de los terminales.

Al lanzar la **compilación** se efectúan **automáticamente** el resto de **pasos necesarios**, siempre y cuando el código esté escrito correctamente.

2. Análisis y síntesis

Se realiza una verificación de la **corrección formal** y **léxica** del código, además de la **compatibilidad** de las conexiones para evitar conflictos hardware.

Si no se detectan errores el proyecto queda **sintetizado**, o bien es interpretado y sintetizado con funciones booleanas simples, siendo por tanto **implementable** dentro del **PLD** seleccionado.

Una vez realizada esta fase se generan una serie de **informes** sobre las características del circuito, sobre los registros necesarios, sobre la lógica combinacional y sobre la interpretación del código.

3. Conexiones y enrutamiento

Las herramientas utilizadas en esta fase se basan en complejos algoritmos que buscan formar con la estructura interior del **CPLD** el hardware descrito, de forma óptima y utilizando el menor número de **LE** (**L**ogic **E**lements) en un espacio físico lo más pequeño posible.

Además se busca minimizar los retardos de propagación de la señal y las capacidades distribuidas debidas a las líneas de comunicación internas.

Esta es la fase más crítica, ya que aquí se crea efectivamente el circuito definitivo.

Existen una serie de estrategias para reducir el esfuerzo que tiene que realizar el compilador para realizar las conexiones, incrementando así las prestaciones y optimizando el proyecto.

Por ejemplo, se podrían sintetizar de forma separada los bloques más críticos y luego insertarlos en un proyecto contenedor como macrobloques “intocables”.

Hay más métodos de optimización de los circuitos, si bien tratar este tema no está contemplado en el objetivo de este artículo.

4. Simulación

Llegado este punto se pueden realizar **simulaciones funcionales** y **análisis de tiempos** del circuito.

Mediante la **simulación funcional** el usuario verifica que el código escrito desarrolla efectivamente las funciones para las que ha sido ideado.

Suele precisar **tiempo extra de desarrollo**, ya que se han de proporcionar las señales en las entradas del circuito para verificar que las salidas ofrecen los valores esperados.

El **análisis de tiempos** (**Timing Analyzer**) suele generar una serie de informes sobre los tiempos de retardo de las señales presentes en el circuito.

Se pueden observar **retardos registro-registro**, incluyendo valores máximos y mínimos, **retardos de propagación** máximos entre un terminal de **entrada** y un terminal de **salida**, **tiempo de respuesta** entre el frente del **reloj** y la aparición de la señal en la salida de un registro, etc.

Estos retardos determinan la **frecuencia máxima** a la que puede trabajar el **PLD**, dato sin duda tremendamente importante.

5. Programación y configuración

Una vez realizadas las **simulaciones funcionales** y los **análisis de tiempos** se puede programar el circuito realizado sobre el **PLD** utilizando un **Programador**.

En nuestro caso el cable **ByteBlaster II** de **Altera** se conecta el **puerto paralelo** del **PC** y al **Programador CPLD LX.1685** con la **Tarjeta test LX.1686** que incluye el **CPLD MAX II** en modo **JTAG** (**J**oint **T**est **A**ction **G**roup).

ByteBlaster II es esencialmente un **cable bidireccional** para la comunicación entre el puerto paralelo del **PC** y el **Programador.**

El estándar **JTAG** está constituido principalmente por una programación en serie de todos los registros del **CPLD**.

Fig.19. Proceso de realización de un CPLD con herramientas de desarrollo tipo Quartus II.

SOLUCIONES EJERCICIOS REVISTA 285

Ejercicio 1: En base a la fórmula utilizada para el cálculo del período y de la frecuencia, y con el esquema a mano, determinar las **frecuencias correspondientes** a la activación de los **pulsadores**.
Solución: Puesto que un pulsador hace pasar el **bit 11** del contador y el otro el **bit 12**, recordando además que la numeración de los bits comienza por **0**, tendremos que:

F1 = 1/T1 = F_clock/2^{12} = 20.000.000/4.096 = 4.882 Hz

F2 = 1/T2 = F_clock/2^{13} = 20.000.000/8.192 = 2.441 Hz

Ejercicio 2: Modificar el esquema de forma que los **diodos LED** conectados a las señales **q[25]**, **q[26]** y **q[27]** permanezcan **siempre apagados**. Probar la solución.

Solución: Es suficiente con renombrar las señales **q[25]**, **q[26]**, **q[27]** a **vlow**, de esta forma los diodos LED están siempre desactivados. Una vez realizado hay que **recompilar** y **programar** el **CPLD** para verificar el funcionamiento.

Ejercicio 3: Insertar el símbolo "**vcc**" y asignarle un nombre, por ejemplo "**vhigh**". Modificar el esquema de forma que los **diodos LED** conectados a las señales **q[25]**, **q[26]** y **q[27]** permanezcan **siempre encendidos**. Probar la solución.

Solución: Hay que seguir los mismos pasos realizados para insertar el símbolo "**gnd**".

En un espacio vacío del esquema hay que hacer click con el botón secundario del ratón, seleccionar "**insert"** y, a continuación, "**symbol**".

Ahora, en la parte izquierda de la pantalla hay que seleccionar la **rama** adecuada dentro del **directorio de instalación** (por ejemplo "**c:\programas\altera\70\quartus\libraries**"), seleccionando "**primitives**".

Acto seguido hay que hacer click sobre "**others**" y luego sobre "**vcc**". En la parte derecha se **previsualiza** el **símbolo**. Ya sólo queda pulsar en **OK** y **posicionar el símbolo** en el lugar deseado del **esquema**.

Si ahora acercamos el **puntero** a la **terminación** del símbolo "**vcc**" recién insertado se notará como el puntero cambia de forma. Haciendo click y arrastrando el ratón se creará un **cable de conexión** .

Una vez asignada una **etiqueta** al cable de conexión se pueden **realizar conexiones** utilizando el **nombre** de la **etiqueta asignada**.

Por ejemplo, asignando el nombre "**vhigh**" al cable conectado al "**vcc**" recién insertado **todas las señales** con la denominación "**vhigh**" quedarán automáticamente **conectadas a Vcc**.

Una vez realizado hay que **recompilar** y **programar** el **CPLD** para verificar el funcionamiento.

Ejercicio 4: **Borrar** la **AND2** (**inst3**) asociada a los pulsadores "**p4**" y "**p5**" y **sustituirla** por una **OR** de **dos entradas** (**OR2**). Probar los resultados.

Solución: En primer lugar hay que hacer click sobre **AND2** y pulsar la tecla "**supr**" del teclado. A continuación hay que proceder a insertar un símbolo, como en la creación de "**vcc"** y "**gnd"**, haciendo click con el botón secundario del ratón, seleccionar "**insert"** y, a continuación, "**symbol**".

Ahora, en la parte izquierda de la pantalla hay que seleccionar la **rama** adecuada dentro del **directorio de instalación** (por ejemplo "**c:\programas\altera\70\quartus\libraries**"), seleccionando "**primitives**", "**logic"** y localizar **OR2.**

Ya sólo queda hacer click en **OK** y posicionar el componente en el lugar del **AND2** anteriormente borrado.

Recompilando y programando el **CPLD** los dos pulsadores encenderán el diodo LED siguiendo la tabla de la verdad de una **OR** en lugar de una **AND.**

Se puede experimentar de forma análoga con otras funciones como **XOR**, **XNOR**, etc.

PRECIO de REALIZACIÓN

LX.1685: Todos los componentes necesarios para realizar el **Programador CPLD** publicado en la **revista Nº 269**, **incluyendo** circuito impreso, integrados, conector para el puerto paralelo, manguera de conexión de 10 hilos para conectar el programador a la Tarjeta de prueba y el **CD-ROM CDR1685** .....**60,90€**

NOTA: El **CD-ROM CDR1685** contiene el programa **Quartus II**, es decir el paquete completo para la escritura del código de programación, para ensamblar y para programar los dispositivos **CPLD**. Además contiene los proyectos **counter.qpf, test.qpf, xor_p.qpf** y **decoder.qpf**.

LX.1686: Todos los componentes necesarios para realizar la **Tarjeta de prueba CPLD incluyendo** circuito impreso, integrados, cuarzo, dos displays, diodos LED, zumbador, pulsadores y la tarjeta SMD **KM.1686** con el chip CPLD **MAX II EPM240T100C5N** ....**120,90€**

CS.1685: Circuito impreso LX.1685.....**5,50€**

CS.1686: Circuito impreso LX.1686.....**25,25€**

ESTOS PRECIOS NO INCLUYEN I.V.A.