Universidad Católica Ntra. Sra. de la Asunción

Metodología de Diseño y modelado de circuitos en VHDL

Ingeniería Electrónica

Carlos Alberto González Ayala

Asunción, 2 de Octubre de 2003
Paraguay

Índice General

1. Metodología de designo

1.1. Concepto de herramientas CAD-EDA

En su sentido más moderno, CAD (designo asistido por ordenador, del inglés Computer Aided Design) significa proceso de designo que emplea sofisticadas técnicas gráficas de ordenador, apoyadas en paquetes de software para ayuda en los problemas analíticos, de desarrollo, de coste y ergonómicos asociados con el trabajo de diseño.

En principio, el CAD es un término asociado al dibujo como parte principal del proceso de designo, sin embargo, dado que el designo incluye otras fases, el término CAD se emplea tanto como para el dibujo, o diseño gráfico, como para el resto de herramientas que ayudan al diseño (como la comprobación de funcionamiento, análisis de costes, etc.)

El impacto de las herramientas de CAD sobre el proceso de diseño de circuitos electrónicos y sistemas procesadores es fundamental. No sólo por la adición de interfaces gráficas para facilitar la descripción de esquemas, sino por la inclusión de herramientas, como los simuladores, que facilitan el proceso de diseño y la conclusión con éxito de los proyectos.

EDA (Electronic Design Automation) es el nombre que se le da a todas las herramientas (tanto hardware como software) para la ayuda al diseño de sistemas electrónicos. Dentro del EDA, las herramientas de CAD juegan un importante papel, sin embargo, no sólo el software es importante, workstations cada día más veloces, elementos de entrada de diseño cada vez más sofisticados, etc. son también elementos que ayudan a facilitar el diseño de circuitos electrónicos.

El diseño hardware tiene un problema fundamental, que no existe, por ejemplo, en la producción del software. Este problema es el alto coste del ciclo diseño -prototipación - testeo - vuelta a empezar, ya que el coste del prototipo suele ser, en general, bastante elevado. Se impone la necesidad de reducir este ciclo de diseño para no incluir la fase de prototipación más que al final del proceso, evitando así la repetición de varios prototipos que es lo que encarece el ciclo. Para ello se introduce la fase de simulación y comprobación de circuitos utilizando herramientas de CAD, de forma que no es necesario realizar físicamente un prototipo para comprobar el funcionamiento del

circuito, economizando así el ciclo de diseño. Este ciclo de diseño hardware se muestra en detalle en la figura 1.1.

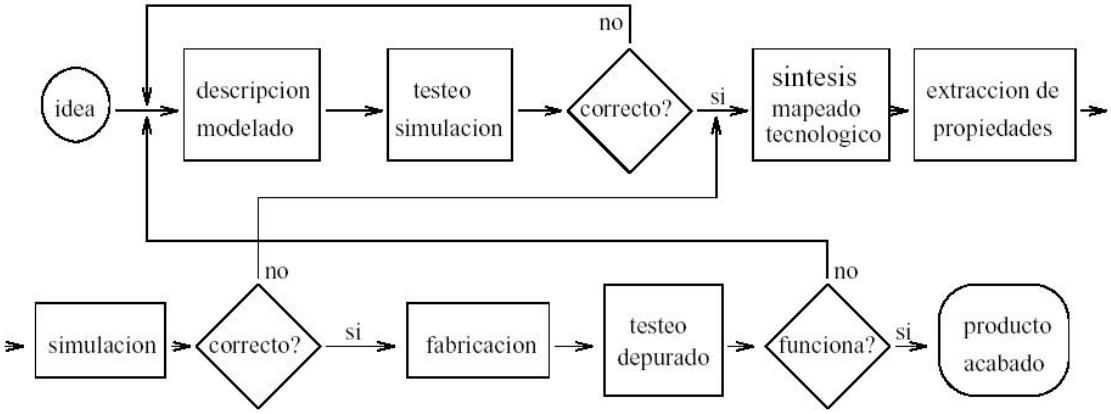

Figura 1.1: Flujo de diseño para sistemas electrónicos y digitales

En el ciclo de diseño hardware las herramientas de CAD están presentes en todos los pasos. En primer lugar en la fase de descripción de la idea, que será un esquema eléctrico, un diagrama de bloques, etc. En segundo lugar en la fase de simulación y comprobación de circuitos, donde diferentes herramientas permiten realizar simulaciones de eventos, funcional, digital o eléctrica de un circuito atendiendo al nivel de simulación requerido. Por último existen las herramientas de CAD orientadas a la fabricación. En el caso de diseño hardware estas herramientas sirven para la realización de PCBs (Printed Circuit Boards o placas de circuito impreso), y también para la realización de ASICs (Application Specific Integrated Circuits) herramientas éstas que nos permiten la realización de microchips así como la realización y programación de dispositivos programables.

Herramientas CAD para el diseño hardware:

- Lenguajes de descripción de circuitos. Son lenguajes mediante los cuales es posible describir un circuito eléctrico o digital. La descripción puede ser de bloques, donde se muestra la arquitectura del diseño, o de comportamiento, donde se describe el comportamiento del circuito en vez de los elementos de los que está compuesto.

- Captura de esquemas. Es la forma clásica de describir un diseño electrónico y la más extendida ya que era la única usada antes de la aparición de las herramientas de CAD. La descripción está basada en un diagrama donde se muestran los diferentes componentes de un circuito.

- Grafos y diagramas de flujo. Es posible describir un circuito o sistema mediante diagramas de flujo, redes de Petri, máquinas de estados, etc. En este caso será una descripción gráfica pero, al contrario que la captura de esquemas, la descripción será comportamental en vez de una descripción de componentes.

- Simulación de sistemas. Estas herramientas se usan sobre todo para la simulación de sistemas. Los componentes de la simulación son elementos de alto nivel como discos duros, buses de comunicaciones, etc. Se aplica la teoría de colas para la simulación.

- Simulación funcional. Bajando al nivel de circuitos digitales se puede realizar una simulación funcional. Este tipo de simulación comprueba el funcionamiento de circuitos digitales de forma funcional, es decir, a partir del comportamiento lógico de sus elementos (sin tener en cuenta problemas eléctricos como retrasos, etc.) se genera el comportamiento del circuito frente a unos estímulos dados.

- Simulación digital. Esta simulación, también exclusiva de los circuitos digitales, es como la anterior con la diferencia de que se tienen en cuenta retrasos en la propagación de las señales digitales. Es una simulación muy cercana al comportamiento real del circuito y prácticamente garantiza el funcionamiento correcto del circuito a realizar.

- Simulación eléctrica. Es la simulación de más bajo nivel donde las respuestas se elaboran a nivel del transistor. Sirven tanto para circuitos analógicos como digitales y su respuesta es prácticamente idéntica a la realidad.

- Realización de PCBs. Con estas herramientas es posible realizar el trazado de pistas para la posterior fabricación de una placa de circuito impreso.

- Realización de circuitos integrados. Son herramientas de CAD que sirven para la realización de circuitos integrados. Las capacidades gráficas de

estas herramientas permiten la realización de las diferentes máscaras que intervienen en la realización de circuitos integrados.

- Realización de dispositivos programables. Con estas herramientas se facilita la programación de este tipo de dispositivos, desde las simples PALs (Programmable And Logic) hasta las más complejas FPGAs (Field Programmable Gate Arrays), pasando por las PLDs (Programmable Logic Devices)

1.2. Diseño Bottom-Up

El término Diseño Bottom-Up (diseño de abajo hacia arriba) se aplica al método de diseño mediante el cual se realiza la descripción del circuito o sistema que se pretende realizar, empezando por describir los componentes más pequeños del sistemas para, más tarde, agruparlos en diferentes módulos, y estos a su vez en otros módulos hasta llegar a uno solo que representa el sistema completo que se pretende realizar. En la figura 1.2 se muestra esta metodología de diseño.

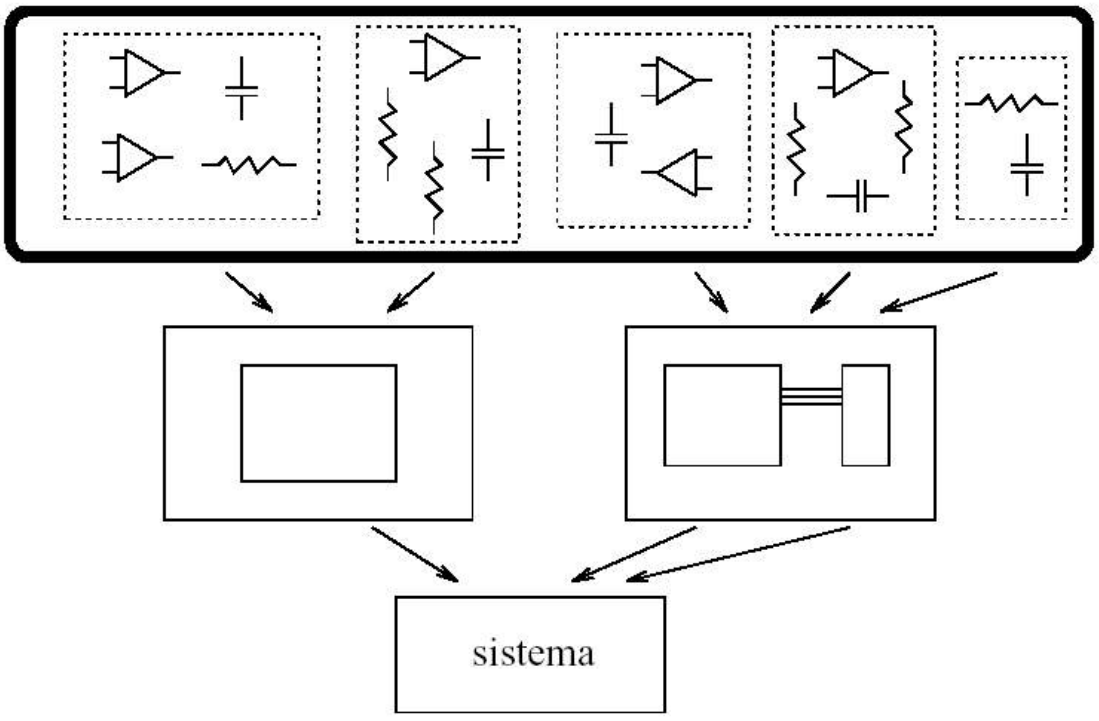

Figura 1.2: Metodología de diseño Bottom-Up

Esta metodología de diseño no implica una estructuración jerárquica de los elementos del sistema. Esta estructuración, al contrario de lo que ocurre en el diseño top-down que se verá después, se realiza una vez realizada la descripción del circuito, y por tanto no resulta necesaria.

En un diseño bottom-up se empieza por crear una descripción, con esquemas por ejemplo, de los componentes del circuito. Estos componentes pertenecen normalmente a una librería que contiene chips, resistencias, condensadores, y otros elementos que representan unidades funcionales con significado propio dentro del diseño. Estas unidades se las puede conocer por el nombre de primitivas puesto que no es necesario disponer de elementos de más bajo nivel para describir el circuito que se pretende realizar.

En general, esta forma de diseñar no es muy buena, ya que es un flujo de diseño bastante ineficiente. Para diseños muy grandes, como los actuales, no se puede esperar unir miles de componentes a bajo nivel y pretender que el diseño funcione adecuadamente. El hecho de unir un número elevado de componentes entre si sin una estructura más elevada que permita separarlos en bloques hace que sea complejo el análisis del circuito, lo que provoca dificultades a la hora de detectar fallos en el circuito, anomalías de funcionamiento, etc. Con esto, la probabilidad de cometer errores de diseño se hace más elevada. Para poder encontrar errores de diseño, o saber si el circuito realizará la función para la que ha sido diseñado, es necesario perder mucho más tiempo en lo que es la definición, diseño y análisis en alto nivel para ver entonces si funciona como deseamos.

1.2.1. Ejemplo práctico

Un ejemplo bastante práctico sería el proceso de diseño de circuito analógicos.

Primeramente se realiza un estudio del proyecto y una solución para el problema (comúnmente utilizando teoría de circuitos) prácticamente si la ayuda de herramientas CAD. Posteriormente, utilizando componentes prefabricados, se procede a la simulación del sistema en alguna herramienta diseñada para tal propósito, que incluye librerías estándar, y/o que permite incluir otras librerías.

El ORCAD Capture y Layout permite, por ejemplo, la simulación de circuitos analógicos en entornos gráficos, o preparación de PCBs, como se muestra en la gráfica:

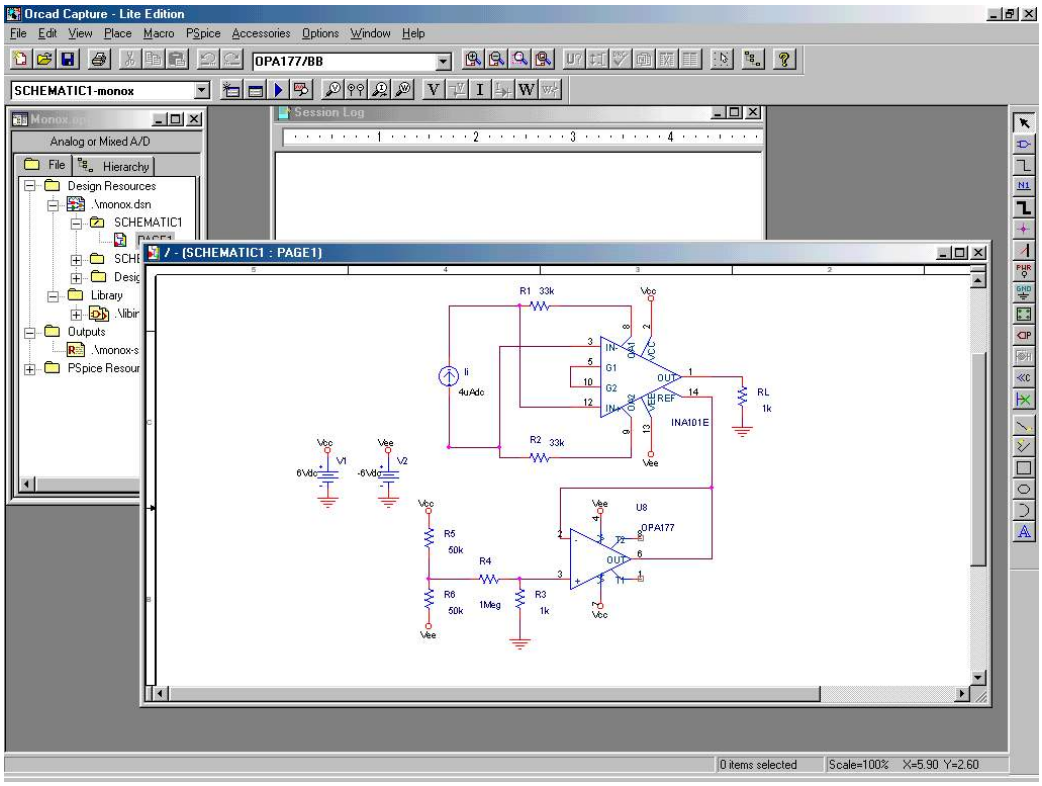

Figura 1.3: Entorno de trabajo del Orcad Capture

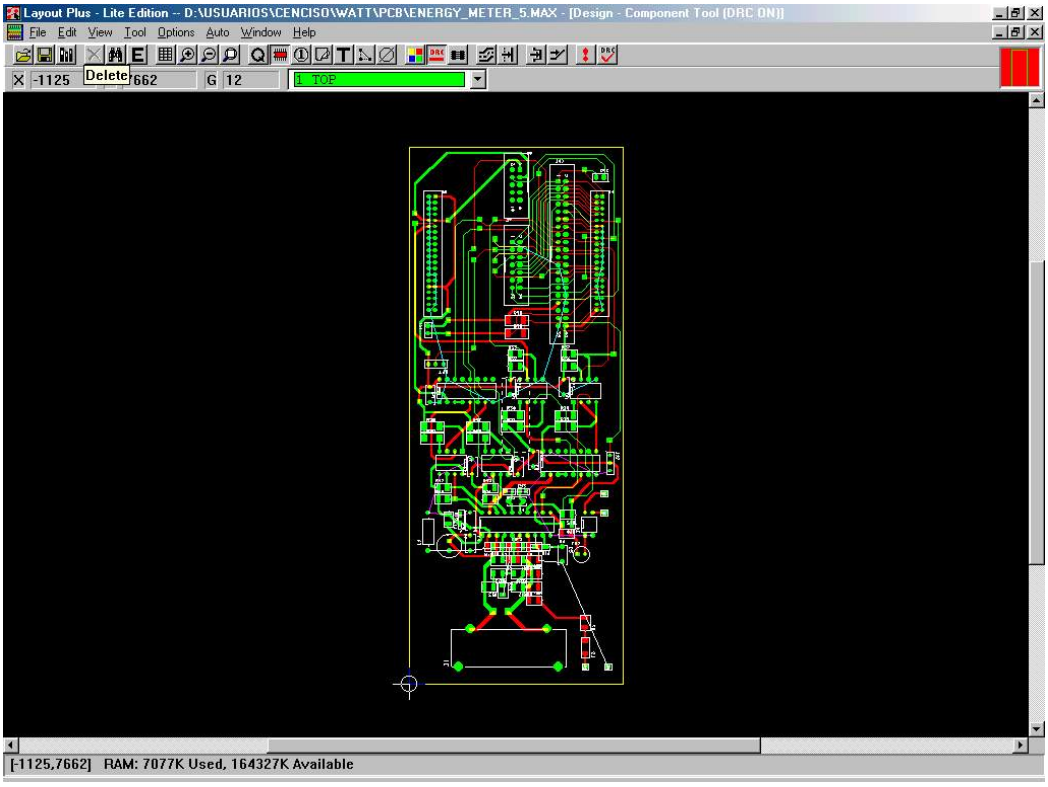

Figura 1.4: Entorno de trabajo del Orcad Layout

1.3. Diseño Top-Down

El diseño Top-Down es, en su más pura forma, el proceso de capturar una idea en un alto nivel de abstracción, e implementar esa idea primero en un muy alto nivel, y después ir hacia abajo incrementando el nivel de detalle, según sea necesario. Esta forma de diseñar se muestra gráficamente en la figura 1.5 donde el sistema inicial se ha dividido en diferentes módulos, cada uno de los cuales se encuentra a su vez subdividido hasta llegar a los elementos primarios de la descripción.

Los años 80 trajeron una revolución en las herramientas para el diseño por ordenador. Aunque esto no modificó la forma de diseñar sí que mejoró la facilidad de hacerlo. Así, mediante el software disponible por ordenador, se podrían diseñar circuitos más complejos en, comparativamente, cortos periodos de tiempo (aunque se siguiera utilizando el diseño bottom-up).

Pero hoy en día, nos encontramos en un marco en que es necesario hacer diseños más y más complicados en menos tiempo. Así, se puede descubrir que el flujo de diseño bottom-up es bastante ineficiente. El problema básico del diseño bottom-up es que no permite acometer con éxito diseños que contengan muchos elementos puesto que es fácil conectarlos de forma errónea. No se puede esperar unir miles de componentes de bajo nivel, o primitivas, y confiar en que el diseño funcione adecuadamente.

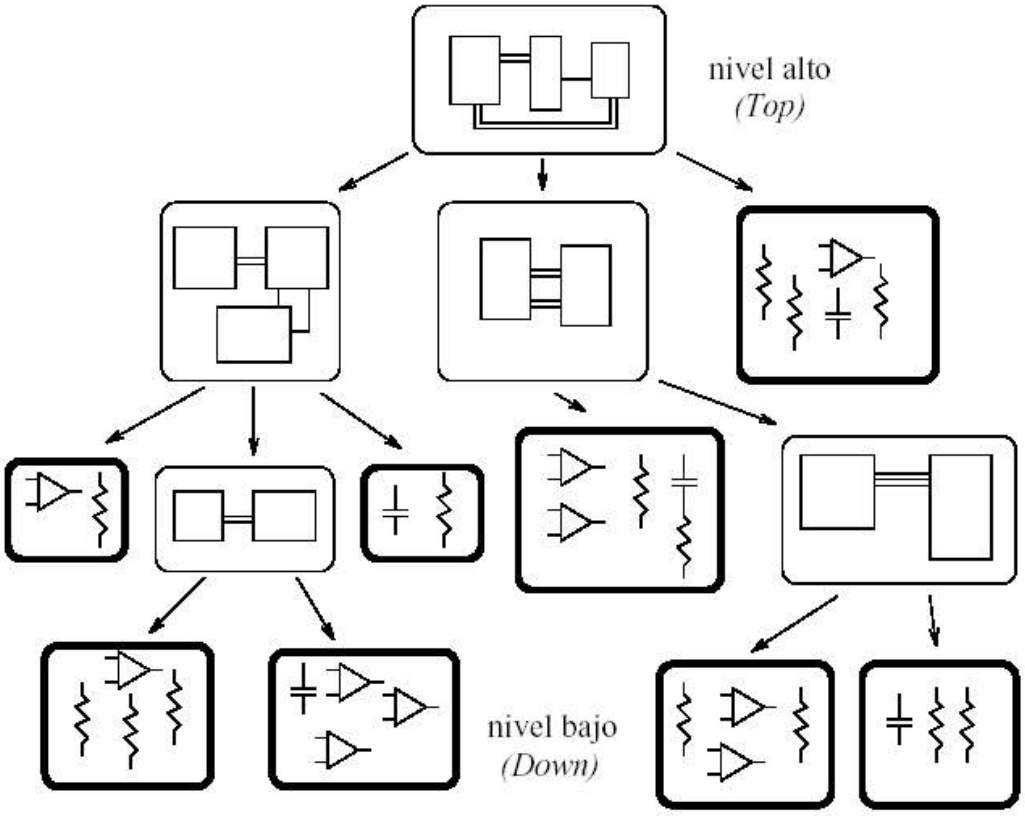

Figura 1.5: Metodología de diseño Top-Down

Para esto existe la metodología Top-down que sigue un poco el lema de "divide y vencerás", de manera que un problema, en principio muy complejo, es dividido en varios subproblemas que a su vez pueden ser divididos en otros problemas mucho más sencillos de tratar. En el caso de un circuito esto se traduciría en la división del sistema completo en módulos, cada uno de los cuales con una funcionalidad determinada. A su vez, estos módulos, dependiendo siempre de la complejidad del circuito inicial o de los módulos, se pueden dividir en otros módulos hasta llegar a los componentes básicos del circuito o primitivas.

1.3.1. Ventajas del diseño Top-Down

- Incrementa la productividad del diseño. Este ujo de diseño permite especificar funcionalmente en un nivel alto de abstracción sin tener que considerar la implementación del mismo a nivel de puertas lógicas. Por ejemplo se puede especificar un diseño en VHDL y el software utilizado

generaría el nivel de puertas directamente. Esto minimiza la cantidad de tiempo utilizado en un diseño.

- Incrementa la reutilización del diseño. En el proceso de diseño se utilizan tecnologías genéricas. Esto es, que no se fija la tecnología a utilizar hasta pasos posteriores en el proceso. Esto permite reutilizar los datos del diseño únicamente cambiando la tecnología de implementación. Así es posible crear un nuevo diseño de uno ya existente.

- Rápida detección de errores. Como se dedica más tiempo a la definición y al diseño, se encuentran muchos errores pronto en el proceso de descripción del circuito.

1.4. Ingeniería concurrente

En los años ochenta, los suministradores de productos EDA se preocuparon sobre todo de realizar herramientas más veloces y workstations más rápidas especialmente pensando en un entorno de diseño donde un producto es diseñado en serie. La competencia entre las diversas compañías se basaba en lo rápido que cada paso de la cadena de diseño podía realizarse.

En los noventa, la competencia se encuentra, no en lo rápido en que se puedan completar los diferentes pasos de un diseño, sino en que se pueda realizar ingeniería concurrente. La ingeniería concurrente permite que se puedan utilizar datos de un paso en el proceso de diseño antes de que el paso previo haya sido completado. Esto implica la existencia de monitores dentro del sistema de diseño para comunicar adecuadamente la actividad de diseño hacia todos los pasos del proceso.

La forma más sencilla de obtener un sistema concurrente es que todos los pasos del proceso de diseño compartan la misma base de datos. De esta manera, diferentes herramientas correspondientes a diferentes pasos en el proceso de diseño, comparten los mismos datos. Un cambio realizado con una herramienta tiene efectos inmediatos sobre la ejecución de otra herramienta.

En general hay dos tipos diferentes de ingeniería concurrente:

- Ingeniería concurrente personal. Viene referida a la posibilidad de realizar cambios en el diseño (esquema) sin tener que abandonar el análisis o

simulación, o las herramientas de diseño de circuitos impresos, por ejemplo.

- Ingeniería concurrente de grupo. Este tipo permite, a los diferentes equipos de expertos que trabajan en un diseño, el solapar la creación, análisis, y trazado de un diseño. Por ejemplo, un equipo puede estar simulando un circuito que otro equipo acaba de modificar, etc.

En general, el elemento más importante de un sistema EDA que permita diseño concurrente, es la base de datos. En esta base de datos, cada elemento es común a todas las herramientas que componen el sistema. Las diferencias entre una herramienta y otra vendrán de lo que la herramienta ve del elemento. Así, cada elemento de la base de datos estará compuesto por distintas vistas cada una asociada generalmente a una herramienta del sistema.

En una herramienta de CAD, donde se incluyan diferentes fases del proceso de diseño como captura de esquemas, simulación, etc, existe siempre la operación por la cual las herramientas posteriores del flujo de diseño (como simulación o diseño de PCBs) conocen los resultados de los pasos previos (como la captura de esquemas). A esta operación se le conoce con el nombre de preanotación o forwardannotation y consiste en que las herramientas anteriores dentro del flujo de diseño, informan a las herramientas posteriores de los cambios realizados en el diseño.

En el caso de herramientas con capacidad para ingeniería concurrente se debe permitir una operación adicional. Esta operación, muy importante dentro de la ingeniería concurrente, es la retroanotación o backannotation. Uno de los objetivos de la ingeniería concurrente es la posibilidad de trabajar en fases del proceso de diseño sin haber completado previamente las fases anteriores. Para conseguir esto, no es únicamente necesario disponer de una base de datos única, sino también, disponer de los mecanismos necesarios para que, herramientas asociadas a fases anteriores del proceso de diseño, puedan saber de los cambios realizados por herramientas posteriores e incorporarlos a su visión especial del diseño. Para esto existe el mecanismo de backannotation que simplemente sirve para que herramientas pertenecientes a fases finales del proceso de diseño puedan anotar cambios a las fases iniciales del diseño.

Por ejemplo, en un esquema podemos especificar el encapsulado de un chip, pero puede que en la fase inicial del diseño no se sepa todavía. Es posible que en el proceso de diseño de las pistas de un circuito impreso, que sería una fase

posterior, ya se conozca dicho encapsulado. En este caso, la herramienta que realiza el diseño del circuito impreso puede backanotar la información del encapsulado a la herramienta de captura de esquemas.

2. Descripción del diseño

La primera tarea a realizar dentro del flujo de diseño electrónico, después de concebir la idea, es realizar una descripción de lo que se pretende hacer. Los ordenadores ofrecen hoy día herramientas especiales para la creación y verificación de diseños. Con dichas herramientas es posible describir tanto un sencillo circuito, que represente una simple puerta lógica, como un complejo diseño electrónico.

En un principio, las herramientas de CAD se limitaban a servir de meros instrumentos de dibujo para poder realizar el diseño; el diseñador de circuitos realizaba la descripción a bajo nivel sobre un papel, utilizando símbolos y componentes básicos, que luego trasladaba al computador para obtener una representación más ordenada. Con la incorporación de herramientas de fabricación de PCBs, o circuitos integrados, o simuladores, etc. la descripción del circuito empezaba a jugar un papel más importante ya que servía como entrada de información a las herramientas posteriores en el flujo de diseño. Esto, unido a la metodología Top-down de diseño de circuitos, llevó a la aparición de herramientas de descripción que permitieran al diseñador definir el problema de una forma abstracta de manera que fuera el ordenador quien se ocupara de realizar la concretización de la idea.

Teniendo en cuenta esta evolución, las herramientas de CAD actuales permiten las siguientes posibilidades de abordar la descripción de una idea o diseño electrónico:

- Descripción estructural. Consiste en enumerar los componentes de un circuito y sus interconexiones. Dependiendo de la herramienta que se utilice hay dos formas de hacerlo:

  - Esquemas. Es la forma tradicional en que los circuitos han sido diseñados desde que la electrónica existe. Consiste en la descripción gráfica de los componentes de un circuito.

  - Lenguaje. Se realiza una enumeración de los componentes de un circuito así como su conexionado.

- Descripción comportamental. Es posible describir un circuito electrónico (generalmente digital) simplemente describiendo cómo se comporta. Para

este tipos de descripción también se utiliza un lenguaje de descripción hardware específico.

2.1. Captura de esquemas

Con captura de esquemas se entiende el proceso de descripción, mediante un dibujo, de un circuito eléctrico. El dibujo del esquema puede incluir más que un simple diagrama de líneas. Puede incluir también información sobre tiempos, instancias, cables, conectores, notas, y muchas otras propiedades importantes y valores necesarios por el resto de aplicaciones para la interpretación del mismo.

Un esquema viene especificado en la base de datos por dos partes fundamentales: las hojas y los símbolos. En principio, un esquema puede estar formado por varias hojas que es donde se dibujan los diversos componentes o símbolos que forman el circuito. En las hojas se especifican también las interconexiones así como informaciones adicional es para el uso posterior del esquema en otras aplicaciones.

Los símbolos son cajas que se interconectan unas con otras en la hoja de diseño. Un símbolo es un objeto que contiene un conjunto de modelos usados para describir los aspectos funcionales, gráficos, temporales, y tecnológicos del diseño electrónico.

Hay dos tipos de símbolos. El primer tipo está formado por los símbolos que representan componentes básicos, o primitivas. Estos componentes definen un elemento que se encuentra en el nivel más bajo de la jerarquía de diseño. Así, este tipo de componentes serían las resistencias, condensadores, transistores, puertas lógicas, procesadores, chips de memoria, etc.

Un segundo tipo de símbolos son aquellos que especifican, no un elemento simple, sino otro circuito completo, compuesto a su vez por símbolos, etc. Es decir, este segundo tipo de símbolos son elementos que están por encima de los símbolos básicos dentro de la jerarquía. Normalmente este tipo de símbolos suelen tener asociados una hoja que es la que describe sus componentes, aunque, con la aparición de las descripciones mediante lenguaje, es posible encontrar que dentro del símbolo en un esquema tenemos una descripción mediante lenguaje en vez de una hoja que sería lo esperable. Las posibilidades de las herramientas de descripción actuales son tales que

permiten, sin demasiados problemas, juntar en un mismo diseño descripciones mediante gráficos y descripciones mediante lenguaje.

El método clásico para la interconexión de los distintos símbolos de una hoja son los hilos o nets. Un hilo en el esquema tiene una correspondiente inmediata con el circuito real, se trata de un cable físico que conecta un pin de un chip con un pin de otro. Sin embargo, dado que un esquema puede representar un nivel de abstracción elevado dentro de una jerarquía, un cable puede representar una conexión con un sentido más amplio, como por ejemplo una línea telefónica, o un enlace de microondas a través de satélite.

Un cable en un esquema es un elemento que indica conexión, y en principio, puede ser tanto un hilo de cobre, como una pista en un circuito impreso, como un conjunto de hilos, como un cable de una interface serie, etc. Sin embargo, en los comienzos del diseño electrónico, donde los esquemas correspondían en la mayoría de los casos al nivel más bajo de una jerarquía, los cables eran siempre hilos conductores, y para representar un conjunto de hilos conductores se introdujo otro elemento adicional, el bus. Un bus es una conexión que une dos componentes al igual que un cable, sin embargo se caracteriza por representar, no un único hilo, sino múltiples. La introducción de este elemento fue inmediata a partir del desarrollo de circuitos digitales, donde la conexión entre procesadores, memorias, etc. era fácilmente agrupable.

Actualmente, dada la gran complejidad de los diseños electrónicos, con miles de conexiones en una misma hoja, se hace necesario el uso de otras técnicas de interconexión de componentes. Una posibilidad que ofrecen la mayoría de herramientas de CAD es la utilización de etiquetas. Es posible poner etiquetas a los pines o a los cables, de manera que dos pines o cables con la misma etiqueta o nombre están físicamente interconectados. Esto evita el tener que trazar múltiples conexiones entre componentes evitando así una aglomeración de hilos que harían ilegible cualquier esquema.

Otro elemento importante dentro de una hoja o esquema son los puertos. Los puertos son conexiones al exterior de la hoja, y realizan la labor de interface del circuito con el mundo exterior. En general, un esquema se puede ver como una caja negra donde los puertos son la única información visible. Esta caja negra, junto con sus puertos, forma un componente que puede ser usado en otra hoja, que a su vez es un componente que puede formar parte de otra hoja

y así sucesivamente. Los puertos pueden ser de entrada, de salida, o de entrada/salida, dependiendo de la dirección del flujo de la información.

2.2. Generación de símbolos

Como se ha comentado anteriormente, el concepto de símbolo tiene un sentido amplio; puede representar tanto un componente físico, como un transistor o un chip, o puede representar un elemento abstracto, como un sistema, etc. que a su vez se encuentra formado por distintos elementos o símbolos.

Los símbolos suelen estar formados por dos elementos fundamentales, el cuerpo y los puertos. El cuerpo es simplemente un dibujo que en su forma más genérica puede ser una simple caja. Los puertos son los elementos que realmente definen el componente ya que indican la comunicación con el exterior. Un componente puede no tener cuerpo (aunque en principio resulta difícil trabajar con un elemento que no se ve) mientras se pueda interconexionar con otros elementos, no se necesita nada más.

Los símbolos más simples, que corresponden a un elemento físico, se encuentran normalmente agrupados en librerías de símbolos. Para ser usados únicamente se necesita copiarlos de la librería y meterlos en el diseño. Hay otros elementos que no representan primitivas, sino que representan otros esquemas en un nivel más bajo de la jerarquía; en estos casos, los símbolos no suelen estar agrupados en librerías sino que forman parte de la base de datos que contiene el diseño completo.

Las herramientas que soportan la metodología de diseño bottom-up o top-down, deben proveer algún mecanismo para convertir las hojas en símbolos, es decir, tomar un esquema, con unas entradas y salidas, y generar una caja con las mismas entradas y salidas que pueda ser usado como un símbolo más en otras partes del diseño. En principio, es muy sencillo generar un símbolo a partir de un esquema, siempre y cuando en el esquema se especifiquen adecuadamente los puertos de interconexión. Estos símbolos, generados a partir de esquemas, sirven para la realización de diseños jerárquicos ya que pueden ser usados en otros esquemas y así sucesivamente.

Aunque un símbolo sólo necesita los puertos y un cuerpo, hay otra serie de elementos que resultan de mucha utilidad. Un elemento muy importante es el nombre, ya es una forma de identificar el símbolo y resulta de utilidad para leer un esquema. Dependiendo de la utilización del símbolo puede ser

interesante la adición de otros elementos o propiedades, así, para símbolos que representen chips, es muy interesante añadirles información sobre el encapsulado del chip, una referencia para identificar individualmente a cada componente dentro del circuito, etc. Para otros componentes, dedicados a simulación por ejemplo, puede ser interesante añadirles propiedades sobre el retraso de la señal, etc.

Un mismo símbolo puede representar varias cosas dentro de un diseño. Lo que un símbolo representa depende de la herramienta particular que se esté utilizando. Supongamos el caso muy simple de un contador. El símbolo del contador será una caja cuadrada, con una serie de entradas y salidas, pero ¿qué representa realmente? Si por ejemplo se está realizando un circuito impreso, este símbolo del contador representa el encapsulado con sus diferentes patillas, y las partes de cobre asociadas. Si por el contrario, queremos realizar una simulación para ver el comportamiento del contador, en realidad el símbolo estará haciendo referencia a una descripción del comportamiento del circuito. Y aun pueden haber más representaciones, el mismo símbolo del contador puede representar a su vez una descripción estructural (realizada con un lenguaje de descripción hardware como VHDL) o incluso otro esquema formado por símbolos más simples como puertas lógicas o incluso transistores. El mismo símbolo representa muchas cosas que conviven de forma concurrente en la misma base de datos. Lo que se ve del símbolo dependerá de la tarea que se realice en cada momento, así como de la herramienta que se esté utilizando.

2.3. Diseño modular

El flujo de diseño top-down, ofrece una ventaja adicional, y es que la información se estructura de forma modular. El hecho de empezar la realización de un diseño a partir del concepto de sistema, hace que las subdivisiones se realicen de forma que los diferentes módulos generados sean disjuntos entre sí y no se solapen. De esta forma, el diseño modular sería la realización de diseños realizando divisiones funcionalmente complementarias de los diversos componentes del sistema, permitiendo de esta manera una subdivisión clara y no solapada de las diferentes tareas dentro del diseño.

El diseño bottom-up, no ofrece tanta facilidad para la división del diseño en partes funcionalmente independientes. Al partir de los elementos básicos de los que se compone el sistema, no resulta tan sencillo agruparlos de forma coherente. Esta es otra de las desventajas del ujo de diseño bottom-up, el

resultado final puede resultar bastante confuso al no estar modularmente dividido.

2.4. Diseño jerárquico

Un complejo diseño electrónico puede necesitar cientos de miles de componentes lógicos para describir correctamente su funcionamiento. Estos diseños necesitan que sean organizados de una forma que sea fácil su comprensión. Una forma de organizar el diseño es la creación de un diseño modular jerárquico tal y como se ha venido viendo cuando se explicaba el ujo de diseño top-down.

Una jerarquía consiste en construir un nivel de descripción funcional de diseño debajo de otro de forma que cada nuevo nivel posee una descripción más detallada del sistema. La construcción de diseños jerárquicos es la consecuencia inmediata de aplicar el flujo de diseño top-down.

En la creación de diseños jerárquicos es muy útil la realización de bloques funcionales o módulos. Un bloque funcional es un símbolo que representa un grupo de elementos en alto nivel. Se puede pensar que un bloque funcional son particiones del diseño original con descripciones asociadas a las pequeñas unidades.

2.5. El netlist

El netlist es la primera forma de describir un circuito mediante un lenguaje, y consiste en dar una lista de componentes, sus interconexiones y las entradas y salidas. No es un lenguaje de alto nivel por lo que no describe como funciona el circuito sino que simplemente se limita a describir los componentes que posee y las conexiones entre ellos.

2.5.1. El formato EDIF

Dada la gran proliferación de lenguajes para la comunicación de descripciones del diseño entre herramientas, fue necesario crear un formato que fuera estándar y que todas las herramientas pudieran entender. Así es como apareció el formato EDIF.

El formato EDIF (Electronic Design Interchange Format) es un estándar industrial para facilitar el intercambio de datos de diseño electrónico entre

sistemas EDA (Electronic Design Automation). Este formato de intercambio está diseñado para tener en cuenta cualquier tipo de información eléctrica, incluyendo diseño de esquemas, trazado de pistas (físicas y simbólicas), conectividad, e información de texto, como por ejemplo las propiedades de los objetos de un diseño.

El formato EDIF fue originalmente propuesto como estándar por Mentor Graphics, Motorola, National Semiconductor, Texas Instruments, Daisy Systems, Tektronix, y la Universidad de California en Berkeley, todos ellos embarcados cooperativamente en su desarrollo. Desde entonces, el EDIF ha sido aceptado por más y más compañías. Fue aprobado como estándar por la Electronic Industries Association (EIA) en 1987, y por el American National Standards Institute (ANSI) en 1988.

La sintaxis de EDIF es bastante simple y comprensible, sin embargo, no se pretende que sea exactamente un lenguaje de descripción de hardware con el cual los diseñadores puedan definir sus circuitos, aunque hay algunos que lo utilizan directamente como lenguaje de descripción. La filosofía del formato EDIF es más la de un lenguaje de descripción para el intercambio de información entre herramientas de diseño que un formato para intercambio de información entre diseñadores. En cualquier caso, siempre es posible describir circuitos utilizando este lenguaje.

Un ejemplo de cómo sería el fichero EDIF que describiría un símbolo, de nombre "pruotro", con una entrada llamada \in" y una salida llamada \out", se puede ver a continuación:

```
(edif EDIFFILENAME (edifVersion 2 0 0)
        (edifLevel 0)
        (keywordMap (keywordLevel 0))
        (status
                (written
                        (timeStamp 1995 2 20 18 2 40)
                        (author "Mentor Graphics Corporation")
                        (program "ENWRITE" (version "v8.4fi2.1"))
                )
        )

        (library (rename &fifasstfipardofimentor "/fasst/pardo/mentor")
                (edifLevel 0)
                (technology
                        (numberDefinition
                                (scale 1 (e 1 -6) (unit distance)))
                )
                (cell pruotro (cellType generic)
```

```
                (view pruotro (viewType netlist)
                        (interface
                                (port in (direction INPUT)
                                        (property pin (string "in"))
                                        (property pintype (string "in"))
                                )
                                (port out (direction OUTPUT)
                                        (property pin (string "out"))
                                        (property pintype (string "out"))
                                )
                        )
                )
        )
        (design pruotro (cellRef pruotro (libraryRef &fifasstfipardofimentor)))
)
```

Una de las características de este formato es la gran cantidad de información que se puede recoger en un único texto. En realidad, en el ejemplo anterior se mostraba el EDIF de un único símbolo con un pin de entrada y otro de salida. Todas las sentencias iniciales son para la definición de librerías, mientras que sólo las últimas sirven para describir el símbolo. Esta descripción empieza con la sentencia (cell pruotro) (cellType generic) donde se indica que se va a describir una célula llamada internamente pruotro. A continuación vendría la sección de interface donde se indican las entradas y salidas. Estas entradas y salidas se indican mediante la sentencia port donde se indica además si el puerto es de entrada o salida. En cada descripción de puerto vienen además sentencias indicando propiedades del port. Por ejemplo el primer pin tiene dos propiedades, una que indica el nombre, llamada pin y otra que indica el tipo llamada pintype. Tanto el nombre de las propiedades como su valor son definibles por el usuario. Estas propiedades son importantes ya que sirven para que otras herramientas de diseño puedan extraer información adicional sobre el circuito. Por ejemplo, en la misma descripción de puerto se podría haber incluido otra propiedad que fuera retraso, de manera que esta información pudiera ser utilizada por una herramienta de simulación por ejemplo.

2.5.2. Otros formatos de Netlist

Aunque el EDIF es el formato de intercambio estándar, dada su complejidad se utilizan a veces otros lenguajes de Netlist mucho más sencillos. Esto lo suelen hacer así los fabricantes ya que les resulta más sencillo interpretar una descripción especialmente pensada para sus herramientas que el formato EDIF que es tan genérico que no es sencillo tener una interface. Lo que suelen hacer los fabricantes es utilizar un lenguaje propio y proveer los programas

necesarios para pasar de su lenguaje al EDIF y viceversa, de esta manera se aseguran la compatibilidad con el resto de herramientas del mundo, y las suyas propias son más sencillas de realizar.

Un ejemplo de lenguaje de descripción lo tenemos en el Tango, cuyo lenguaje de netlist es muy simple y contempla muchas posibles descripciones, incluida la inclusión de propiedades. Tango es un entorno de trabajo para PC que incluye herramientas de descripción y diseño de PCBs. Más adelante se verá un ejemplo de esta descripción.

Otro formato de netlist, este muy usado directamente y no a partir de esquemas, es el formato de descripción de Spice. Spice es un simulador eléctrico, es decir, simula transistores, resistencias, etc. aunque también permite la simulación eléctrica de circuitos digitales. Este lenguaje es utilizado por el simulador para saber exactamente como es el circuito a simular. Está solamente indicado para ser utilizado con este programa por lo que está limitado su uso para otros propósitos. Como ejemplo de las limitaciones que presenta se puede decir que no permite la inclusión de propiedades en el diseño.

2.5.3. Ejemplo de diferentes Netlist

Se presenta a continuación un circuito y su descripción usando los tres formatos que se acaban de comentar. El circuito que se pretende describir aparece en la figura 2.1 y se trata de un esquema que ha sido generado a partir de la herramienta de captura de esquemas de Tango.

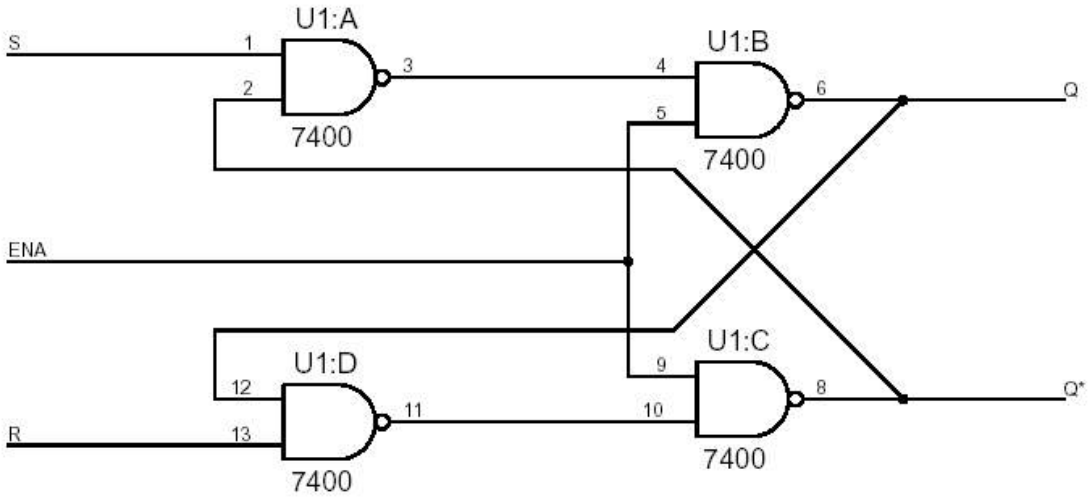

Figura 2.1: Ejemplo de esquema para su descripción Netlist

En primer lugar se presenta la descripción EDIF de este simple circuito:

```
(edif TI
        (edifVersion 2 0 0) (edifLevel 0) (keywordMap (keywordLevel 0))
        (status
                (written
                        (timeStamp 1996 2 22 19 40 43)
                        (dataOrigin "TANGO Schematic" (Version "1.30"))
                        (comment "Copyright (C) 1990 ACCEL Technologies Inc.")
                )
        )
        (Design ROOT
                (CellRef TI
                        (LibraryRef TIfiLIB))
        )
        (Library TIfiLIB (EdifLevel 0)
                (technology (numberDefinition (scale 1 (E 254 -7) (unit DISTANCE))))
                (cell U1 (cellType GENERIC)
                        (property Type (string "7400"))
                        (view S (viewType SCHEMATIC)
                                (interface
                                        (Port A (Designator "1")
                                                (Direction INPUT )
                                        )
                                        (Port B (Designator "2")
                                                (Direction INPUT )
                                        )
                                        (Port Y (Designator "3")
                                                (Direction OUTPUT )
```

```
                        )
                        (Port A (Designator "4")
                                (Direction INPUT )
                        )
                        (Port B (Designator "5")
                                (Direction INPUT )
                        )
                        (Port Y (Designator "6")
                                (Direction OUTPUT )
                        )
                        (Port GND (Designator "7")
                                (property ElecType (string "Power") )
                        )
                        (Port Y (Designator "8")
                                (Direction OUTPUT )
                        )
                        (Port A (Designator "9")
                                (Direction INPUT )
                        )
                        (Port B (Designator "10")
                                (Direction INPUT )
                        )
                        (Port Y (Designator "11")
                                (Direction OUTPUT )
                        )
                        (Port A (Designator "12")
                                (Direction INPUT )
                        )
                        (Port B (Designator "13")
                                (Direction INPUT )
                        )
                        (Port VCC (Designator "14")
                                (property ElecType (string "Power") )
                        )
                )
        )
)
(cell TI (cellType GENERIC)
        (view N
                (viewType NETLIST)
                (interface)
                (Contents
                        (net ENA
                                (joined (portRef &B (viewRef S (cellRef U1)))
                                        (portRef &A (viewRef S (cellRef U1)))
                                )
                        )
                        (net GND
                                (joined (portRef &GND (viewRef S (cellRef U1)))
                                )
                        )
                        (net NETfi002
                                (joined (portRef &B (viewRef S (cellRef U1)))
                                        (portRef &Y (viewRef S (cellRef U1)))
                                )
                        )
```

```
                    (net NETfi004
                        (joined (portRef &Y (viewRef S (cellRef U1)))
                            (portRef &A (viewRef S (cellRef U1)))
                        )
                    )
                    (net Q
                        (joined (portRef &Y (viewRef S (cellRef U1)))
                            (portRef &A (viewRef S (cellRef U1)))
                        )
                    )
                    (net Q*
                        (joined (portRef &B (viewRef S (cellRef U1)))
                            (portRef &Y (viewRef S (cellRef U1)))
                        )
                    )
                    (net R
                        (joined (portRef &B (viewRef S (cellRef U1)))
                        )
                    )
                    (net S
                        (joined (portRef &A (viewRef S (cellRef U1)))
                        )
                    )
                    (net VCC
                        (joined (portRef &VCC (viewRef S (cellRef U1)))
                        )
                    )
                )
            )
        )
    )
)
```

Se ve que es una descripción bastante larga para lo pequeño del circuito, pero esto es necesario si se pretende una completa compatibilidad con el resto de herramientas que existen. La primera parte es el encabezamiento y contiene básicamente comentarios. A continuación viene la parte de librería donde se definen los componentes que aparecen. En este caso hay un único componente, representado por cuatro símbolos, que corresponde al 7400 de Texas Instruments y que contiene 4 puertas NAND. El componente viene definido por sus puertos, cada uno con un nombre diferente y una dirección de entrada o salida. Además, en este caso, cada puerto contiene un designator que sirve para indicar la posición del pin dentro del encapsulado que es un DIP14 en este caso. Hay que destacar que aunque en el esquema no aparecen las tensiones de alimentación, sí que aparecen en cambio en el Netlist. Esto es debido a que muchas veces estas tensiones, tanto VCC como GND, aparecen ocultas en los símbolos del esquema, pero son necesarias en el Netlist ya que éste debe contener absolutamente toda la información, incluida la alimentación del circuito.

Después de la definición del componente viene la definición del interconexionado. El emplazamiento del componente se realiza mediante la instrucción cell. Las interconexiones se realizan en el bloque indicado por la palabra clave Contents. Cada conexión se indica en un bloque net donde se indica el nombre y qué cosas conecta mediante el bloque joined. Por ejemplo, la primera conexión (ENA) conecta los puertos A y B del chip U1 que es el 7400 especificado en la librería. Y así se van conectando los diferentes elementos. Hay conexiones que no tienen nombre en el esquema, pero todas las conexiones en un Netlist deben tener un nombre, así que lo que hace la herramienta en estos casos es inventarse nombres. Este es precisamente el caso de las conexiones NET 002 y NET 004 que son los nombres que la herramienta se ha inventado.

El listado que viene a continuación corresponde a la descripción del mismo circuito pero utilizando un netlist propio de Tango:

```
[
U1
DIP14
7400
]

(
ENA
U1-5
U1-9
)
(
GND
U1-7
)
(
NET_002
U1-10
U1-11
)
(
NET_004
U1-3
U1-4
)
(
Q
U1-6
U1-12
)
(
Q*
U1-2
U1-8
)
(
R
U1-13
)
(
S
U1-1
)
(
VCC
U1-14
)
```

Se observa que esta descripción es mucho más simple y fácil de entender que la anterior. Ello es debido a que este Netlist no necesita ser estándar ni ser exportado a ninguna otra herramienta, sino que debe servir únicamente para el entorno de Tango, por lo que es posible simplificar mucho más su descripción.

En la cabecera, las primeras líneas encerradas entre corchetes, se encuentra la parte de definición de los elementos. Simplemente viene el nombre del chip (7400), su referencia dentro del esquema (U1) y una propiedad adicional que

en el formato EDIF no se encontraba, y es la propiedad que indica el tipo de encapsulado del símbolos; en este caso, el valor de la propiedad de encapsulado es DIP14 que indica un encapsulado Dual Inline Package de catorce pines. Esto es necesario en Tango puesto que este netlist va a ser leído tal cual por la herramienta de diseño de PCBs por lo que es interesante saber de antemano el encapsulado.

Después de la definición de los elementos que componen el esquema vienen las interconexiones. éstas están agrupadas entre paréntesis. La primera conexión, net, cable, etc., es ENA y se conoce porque es el primer nombre después de abrir el paréntesis. A continuación del nombre vienen todos los nodos a los que está conectado. En el caso de ENA se ve claramente que está conectado a U1-5 y U1-9, es decir, ENA es una conexión que conecta los pines 5 y 9 del chip U1 que es el único en el esquema. Y el resto de interconexiones se realizan de la misma manera.

El último ejemplo corresponde a la descripción para Spice del mismo circuito. Como vamos a ver es la descripción más simple de todas ya que sólo tiene un objetivo, y es la de ser utilizada como entrada para un programa en concreto, el simulador Spice:

```
* TI CIRCUIT FILE
U1 S Q* 4 4 ENA Q 0 Q* ENA 2 2 Q R VCC 7400
.END
```

Toda la información del circuito se encuentra en la línea segunda, con lo que todavía es más simple de lo que parece. La primera es un comentario que además hace de título del netlist. En la segunda se encuentra la descripción, y la última indica que se acabó la descripción.

La sintaxis es bien simple (línea segunda). La primera palabra indica el nombre, U1, y como empieza por la letra U, Spice ya sabe que se trata de un chip o componente. Además sabe que todos los nombres que siguen corresponden a nombres de nodos o conexiones y se corresponden con las entradas del chip. Sólo el último nombre indica de qué chip se trata, en este caso el 7400. En Spice dos nodos con el mismo nombre están conectados, así es fácil ver que la conexión ENA conecta los pines 5 y 9 del componente porque las posiciones quinta y novena del chip están marcadas como ENA.

Se han mostrado en esta sección diversos tipos de Netlist y se han sacado algunas conclusiones. La más importante es que el netlist es un formato de intercambio de información a nivel de herramientas cuya descripción se basa en enumerar los componentes del circuito y sus interconexiones. Otra conclusión importante es que existe un formato estándar que sirve casi para cualquier herramienta, como es el formato EDIF. Otra cosa que se ha visto es que la complejidad en la sintaxis depende de la generalidad del lenguaje utilizado. Así, el formato EDIF es el más complejo puesto que es el más genérico que existe. El resto de lenguajes, específicos para cada herramienta, pueden ser mucho mas simples, pero se pierde generalidad, ya que con la simplificación se está eliminando mucha información que puede ser útil para determinado tipo de herramientas.

3. Introducción al lenguaje VHDL

Se vio en el capítulo anterior, que la forma más común de describir un circuito era mediante la utilización de esquemas que son una representación gráfica de lo que se pretende realizar. Con la aparición de herramientas de EDA cada vez más complejas, que integran en el mismo marco de trabajo tanto las herramientas de descripción, síntesis y realización, apareció también la necesidad de disponer de una descripción del circuito que permitiera el intercambio de información entre las diferentes herramientas que componen la herramienta de trabajo.

En principio se utilizó un lenguaje de descripción que permitía, mediante sentencias simples, describir completamente un circuito. A estos lenguajes se les llamó Netlist puesto que eran simplemente eso, un conjunto de instrucciones que indicaban el interconexionado entre los componentes de un diseño, es decir, se trataba de una lista de conexiones.

A partir de estos lenguajes simples, que ya eran auténticos lenguajes de descripción hardware, se descubrió el interés que podría tener el describir los circuitos directamente utilizando un lenguaje en vez de usar esquemas. Sin embargo, se siguieron utilizando esquemas puesto que desde el punto de vista del ser humano son mucho más sencillos de entender, aunque un lenguaje siempre permite una edición más sencilla y rápida.

Con una mayor sofisticación de las herramientas de diseño, y con la puesta al alcance de todos de la posibilidad de fabricación de circuitos integrados y de circuitos con lógica programable, fue apareciendo la necesidad de poder describir los circuitos con un alto grado de abstracción, no desde el punto de vista estructural, sino desde el punto de vista funcional. Existía la necesidad de poder describir un circuito pero no desde el punto de vista de sus componentes, sino desde el punto de vista de cómo funcionaba.

Este nivel de abstracción se había alcanzado ya con las herramientas de simulación. Para poder simular partes de un circuito era necesario disponer de un modelo que describiera el funcionamiento de ese circuito, o componente. Estos lenguajes estaban sobre todo orientados a la simulación, por lo que poco importaba que el nivel de abstracción fuera tan alto que no fuera sencillo una realización o síntesis a partir de dicho modelo.

Con la aparición de técnicas para la síntesis de circuitos a partir de un lenguaje de alto nivel, se utilizaron como lenguajes de descripción precisamente estos lenguajes de simulación, que si bien alcanzan un altísimo nivel de abstracción, su orientación es básicamente la de simular, por lo que los resultados de una síntesis a partir de descripciones con estos lenguajes no es siempre la más óptima. En estos momentos no parece que exista un lenguaje de alto nivel de abstracción cuya orientación o finalidad sea la de la síntesis automática de circuitos, por lo que todavía, de hecho se empieza ahora, se utilizan estos lenguajes orientados a la simulación también para la síntesis de circuitos.

3.1. El lenguaje VHDL

VHDL, viene de VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. VHDL es un lenguaje de descripción y modelado diseñado para describir (en una forma que los humanos y las máquinas puedan leer y entender) la funcionalidad y la organización de sistemas hardware digitales, placas de circuitos, y componentes.

VHDL fue desarrollado como un lenguaje para el modelado y simulación lógica dirigida por eventos de sistemas digitales, y actualmente se lo utiliza también para la síntesis automática de circuitos. El VHDL fue desarrollado de forma muy parecida al ADA debido a que el ADA fue también propuesto como un lenguaje puro pero que tuviera estructuras y elementos sintácticos que permitieran la programación de cualquier sistema hardware sin limitación de la arquitectura. El ADA tenía una orientación hacia sistemas en tiempo real y al hardware en general, por lo que se lo escogió como modelo para desarrollar el VHDL.

VHDL es un lenguaje con una sintaxis amplia y flexible que permite el modelado estructural, en flujo de datos y de comportamiento hardware. VHDL permite el modelado preciso, en distintos estilos, del comportamiento de un sistema digital conocido y el desarrollo de modelos de simulación.

Uno de los objetivos del lenguaje VHDL es el modelado. Modelado es el desarrollo de un modelo para simulación de un circuito o sistema previamente implementado cuyo comportamiento, por tanto, se conoce. El objetivo del modelado es la simulación.

Otro de los usos de este lenguaje es la síntesis automática de circuitos. En el

proceso de síntesis, se parte de una especificación de entrada con un determinado nivel de abstracción, y se llega a una implementación más detallada, menos abstracta. Por tanto, la síntesis es una tarea vertical entre niveles de abstracción, del nivel más alto en la jerarquía de diseño se va hacia el más bajo nivel de la jerarquía.

El VHDL es un lenguaje que fue diseñado inicialmente para ser usado en el modelado de sistemas digitales. Es por esta razón que su utilización en síntesis no es inmediata, aunque lo cierto es que la sofisticación de las actuales herramientas de síntesis es tal que permiten implementar diseños especificados en un alto nivel de abstracción.

La síntesis a partir de VHDL constituye hoy en día una de las principales aplicaciones del lenguaje con una gran demanda de uso. Las herramientas de síntesis basadas en el lenguaje permiten en la actualidad ganancias importantes en la productividad de diseño.

Algunas ventajas del uso de VHDL para la descripción hardware son:

- VHDL permite diseñar, modelar, y comprobar un sistema desde un alto nivel de abstracción bajando hasta el nivel de definición estructural de puertas.

- Circuitos descritos utilizando VHDL, siguiendo unas guías para síntesis, pueden ser utilizados por herramientas de síntesis para crear implementaciones de diseños a nivel de puertas.

- Al estar basado en un estándar (IEEE Std 1076-1987) los ingenieros de toda la industria de diseño pueden usar este lenguaje para minimizar errores de comunicación y problemas de compatibilidad.

- VHDL permite diseño Top-Down, esto es, permite describir (modelado) el comportamiento de los bloques de alto nivel, analizándolos (simulación), y refinar la funcionalidad de alto nivel requerida antes de llegar a niveles más bajos de abstracción de la implementación del diseño.

- Modularidad: VHDL permite dividir o descomponer un diseño hardware y su descripción VHDL en unidades más pequeñas.

3.1.1. VHDL describe estructura y comportamiento

Existen dos formas de describir un circuito. Por un lado se puede describir un circuito indicando los diferentes componentes que lo forman y su interconexión, de esta manera tenemos especificado un circuito y sabemos como funciona; esta es la forma habitual en que se han venido describiendo circuitos y las herramientas utilizadas para ello han sido las de captura de esquemas y las descripciones netlist.

La segunda forma consiste en describir un circuito indicando lo que hace o cómo funciona, es decir, describiendo su comportamiento. Naturalmente esta forma de describir un circuito es mucho mejor para un diseñador puesto que lo que realmente lo que interesa es el funcionamiento del circuito más que sus componentes. Por otro lado, al encontrarse lejos de lo que un circuito es realmente puede plantear algunos problemas a la hora de realizar un circuito a partir de la descripción de su comportamiento.

El VHDL va a ser interesante puesto que va a permitir los dos tipos de descripciones:

- Estructura: VHDL puede ser usado como un lenguaje de Netlist normal y corriente donde se especifican por un lado los componentes del sistema y por otro sus interconexiones.

- Comportamiento: VHDL también se puede utilizar para la descripción comportamental o funcional de un circuito. Esto es lo que lo distingue de un lenguaje de Netlist. Sin necesidad de conocer la estructura interna de un circuito es posible describirlo explicando su funcionalidad. Esto es especialmente útil en simulación ya que permite simular un sistema sin conocer su estructura interna, pero este tipo de descripción se está volviendo cada día más importante porque las actuales herramientas de síntesis permiten la creación automática de circuitos a partir de una descripción de su funcionamiento.

3.2. Ejemplo básico de descripción VHDL

Ejemplo 3.1 Describir en VHDL un circuito que multiplexe dos líneas (a y b) de un bit, a una sola línea (salida) también de un bit; la señal selec sirve para indicar que a la salida se tiene la línea a (selec='0') o b (selec='1').

En la figura 3.1 se muestra el circuito implementado con puertas lógicas que realiza la función de multiplexación.

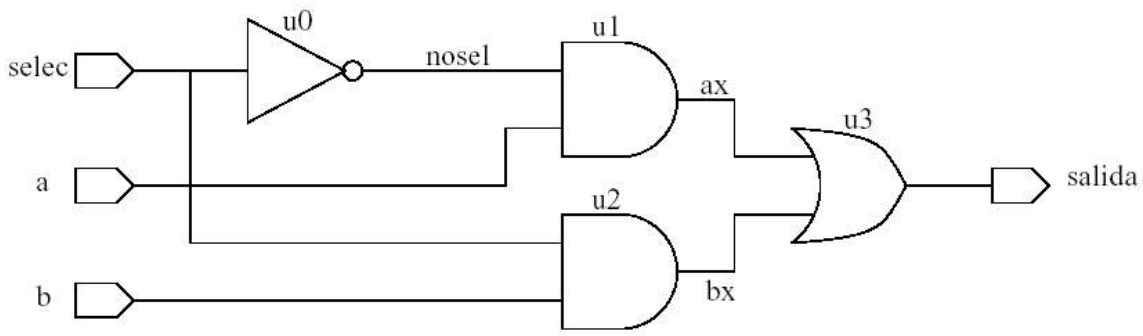

Figura 3.1: Esquema del ejemplo básico en VHDL

Lo que se va a realizar a continuación es la descripción comportamental del circuito de la figura 3.1, y luego se realizará la descripción estructural para ver las diferencias. Más adelante se verá que hay dos tipos de descripción comportamental, pero de momento, el presente ejemplo únicamente pretende introducir el lenguaje VHDL y su estructura.

La sintaxis del VHDL no es sensible a mayúsculas o minúsculas por lo que se puede escribir como se prefiera. A lo largo de las explicaciones se intentará poner siempre las palabras claves del lenguaje en mayúsculas para distinguirlas de las variables y otros elementos.

En primer lugar, sea el tipo de descripción que sea, hay que definir el símbolo o entidad del circuito. En efecto, lo primero es definir las entradas y salidas del circuito, es decir, la caja negra que lo define. Se le llama entidad porque en la sintaxis de VHDL esta parte se declara con la palabra clave ENTITY. Esta definición de entidad, que suele ser la primera parte de toda descripción VHDL, se expone a continuación:

```
-- Los comentarios empiezan por dos guiones
ENTITY mux IS
PORT (  a: IN bit;
                b: IN bit;
                selec: IN bit;
```

```
        salida: OUT bit);
END mux;
```

Esta porción del lenguaje indica que la entidad mux (que es el nombre que se le ha dado al circuito) tiene tres entradas de tipo bit, y una salida también del tipo bit. Los tipos de las entradas y salidas se verán más adelante. El tipo bit simplemente indica una línea que puede tomar los valores '0' y '1'.

La entidad de un circuito es única, sin embargo, se mostró que un mismo símbolo, en este caso entidad, podía tener varias vistas o en el caso de VHDL arquitecturas. Cada bloque de arquitectura, que es donde se describe el circuito, puede ser una representación diferente del mismo circuito. Por ejemplo, puede haber una descripción estructural y otra comportamental, ambas son descripciones diferentes, pero ambas descripciones corresponden al mismo circuito, símbolo, o entidad. Veamos entonces la descripción comportamental:

```
ARCHITECTURE comportamental OF mux IS
BEGIN
        PROCESS(a,b,selec)
        BEGIN
                IF (selec='0') THEN
                        salida<=a;
                ELSE
                        salida<=b;
                END IF;
        END PROCESS;
END comportamental;
```

Un bloque PROCESS se considera como una especie de subrutina cuyas instrucciones se ejecutan secuencialmente cada vez que algunas de las señales de la lista sensible cambia. Esta lista sensible es una lista de señales que se suele poner junto a la palabra clave PROCESS, y en el caso del ejemplo es (a,b,selec).

Esta descripción comportamental es muy sencilla de entender ya que sigue una estructura parecida a los lenguajes de programación convencionales. Lo que se está indicando es simplemente que si la señal selec es cero, entonces la salida es la entrada a, y si selec es uno, entonces la salida es la entrada b. Esta forma tan sencilla de describir el circuito permite a ciertas herramientas sintetizar un circuito a partir de una descripción comportamental como esta. La diferencia con un Netlist es directa: en una descripción comportamental no se están indicando ni los componentes ni sus interconexiones, sino simplemente lo que hace, es decir, su comportamiento o funcionamiento.

La descripción anterior era puramente comportamental, de manera que con una secuencia sencilla de instrucciones podíamos definir el circuito. Naturalmente, a veces resulta más interesante describir el circuito de forma que esté más cercano a una posible realización física del mismo. En ese sentido VHDL posee una forma de describir circuitos que además permite la paralelización de instrucciones y que se encuentra más cercana a una descripción estructural del mismo. A continuación se muestran dos ejemplos de una descripción concurrente o también llamada de transferencia entre registros:

```
ARCHITECTURE transferencia OF mux IS
SIGNAL nosel,ax,bx: bit;
BEGIN
        nosel<=NOT selec;
        ax<=a AND nosel;
        bx<=b AND selec;
        salida<=ax OR bx;
END transferencia;
```

```
ARCHITECTURE transferencia OF mux IS
BEGIN
        salida<=a WHEN selec='0' ELSE b;
END transferencia;
```

En la descripción de la izquierda hay varias instrucciones todas ellas concurrentes, es decir, se ejecutan cada vez que cambia alguna de las señales que intervienen en la asignación. Este primer caso es casi una descripción estructural ya que de alguna manera se están definiendo las señales (cables) y los componentes que la definen, aunque no es comportamental ya que en realidad se trata de asignaciones a señales y no una descripción de componentes y conexiones. El segundo caso (derecha) es también una descripción de transferencia aunque basta una única instrucción de asignación para definir el circuito.

Aunque no es la característica más interesante del VHDL, también permite ser usado como Netlist o lenguaje de descripción de estructura. En este caso, esta estructura también estaría indicada dentro de un bloque de arquitectura, aunque la sintaxis interna es completamente diferente:

```
ARCHITECTURE estructura OF mux IS
        COMPONENT and2
                PORT(e1,e2: IN bit; y: OUT bit);
        END COMPONENT;
        COMPONENT or2
                PORT(e1,e2: IN bit; y: OUT bit);
        END COMPONENT;
        COMPONENT inv
                PORT(e: IN bit; y: OUT bit);
        END COMPONENT;
```

```
        SIGNAL ax,bx,nosel: bit;

BEGIN
        u0: inv PORT MAP(e=>selec,y=>nosel);
        u1: and2 PORT MAP(e1=>a,e2=>nosel,y=>ax);
        u2: and2 PORT MAP(e1=>b,e2=>sel,y=>bx);
        u3: or2 PORT MAP(e1=>ax,e2=>bx,y=>salida);
END estructura;
```

Se observa fácilmente que esta descripción es más larga y encima menos clara que las anteriores. Dentro de la arquitectura se definen en primer lugar los componentes que se van a utilizar. Esto se realiza mediante la palabra clave COMPONENT, donde se indican además las entradas y salidas mediante la cláusula PORT. Estos componentes deben tener una entidad y arquitectura propia indicando su comportamiento. Normalmente estas entidades se suelen poner en una librería separada.

Al igual que ocurre en cualquier netlist, las señales o conexiones deben tener un nombre. En el esquema se le han puesto nombres a las líneas de conexión internas al circuito. Estas líneas hay que declararlas como SIGNAL en el cuerpo de la arquitectura y delante del BEGIN. Una vez declarados los componentes y las señales que intervienen se procede a conectarlos entre si. Para ello la sintaxis es muy simple. Lo primero es identificar cada componente, es lo que comúnmente se conoce como instanciación, es decir, asignarle a cada componente concreto un símbolo. En este ejemplo se le ha llamado u a cada componente y se le ha añadido un número para distinguirlos, en principio el nombre puede ser cualquier cosa y la única condición es que no haya dos nombres iguales. A continuación del nombre viene el tipo de componente que es, en nuestro caso puede ser una and2, una or2, o una puerta inversora inv. Después se realizan las conexiones poniendo cada señal en su lugar correspondiente con las palabras PORT MAP. Así, los dos primeros argumentos en el caso de la puerta and2 son las entradas, y el último es la salida. De esta forma tan simple se va creando el netlist o definición de la estructura.

3.3. Herramientas utilizadas en VHDL.

Existen muchas herramientas que trabajan con este lenguaje, como por ejemplo: el MaxPlus y el Quartus de Altera, Veribest, etc.

El entorno de programación es también gráfico como vimos en el caso del Orcad, facilitando la unión de bloques en proyectos grandes. En la figura 3.2 se muestra el entorno de programación de la herramienta Quartus II de Altera.

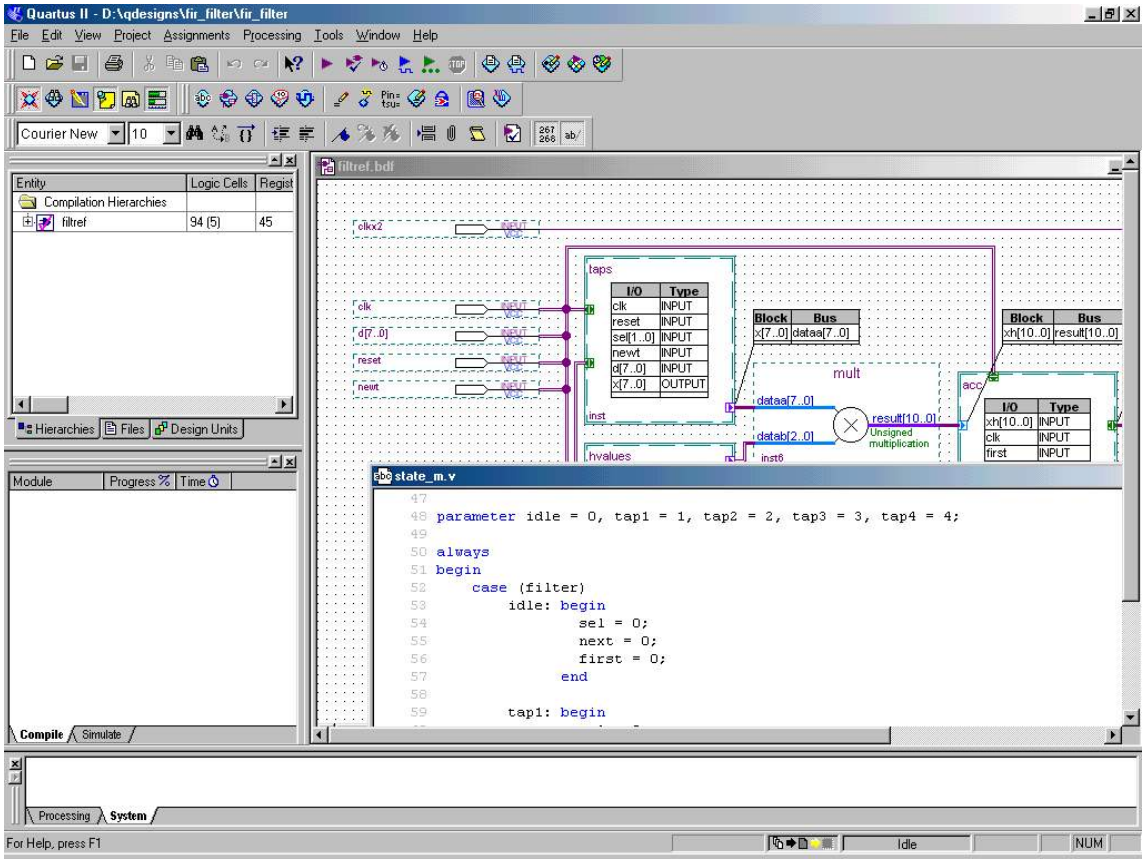

Figura 3.2: Entorno del Quartus II.

La ventaja de trabajar con Altera es la posibilidad de utilizar alguno de los kits de desarrollo fabricados por la misma empresa, lo que posibilita el desarrollo en hardware de las simulaciones realizadas en la computadora.

Una aplicación muy interesante es la implementación de filtros digitales, utilizando por ejemplo un kit FLEX 10K de Altera.

Luego del desarrollo de una interfaz electrónica adecuada, se puede conectar la salida acondicionada al kit para tomar muestras de la señal, y así poder trabajar con una señal digital, lo cual es mucho más sencillo que trabajar en el campo analógico.

Existen muchas aplicaciones para los filtros digitales. Se puede medir la rugosidad de superficies utilizando sensores de ultrasonido, cuyas señales moduladas son difíciles de tratar en forma analógica, pero digitalmente se el trabajo se vuelve más sencillo, gracias al rendimiento de las computadoras modernas, que permiten realizar grandes cómputos en poco tiempo, relativamente hablando.

También se utilizan para medición con exactitud de niveles de líquidos en recipientes difíciles de observar, medición de distancias, sensores de posición, ecografía computarizada, etc.

4. Bibliografía

[1] Fernando Pardo Carpio. Lenguaje para descripción y modelado de circuitos. Universidad de Valencia. Valencia, España. 1997.
[2] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993. IEEE, June 6, 1994.
[3] Olga L. Mazuela. Curso de diseño basado en VHDL. Departamento de Ing. Eléctrica de la Universidad de Los Andes. Bogotá, Colombia, Enero 24, 1994.
[4] Miguel Angel Freire Rubio. Introducción al lenguaje VHDL. Universidad Politécnica de Madrid, Departamento de Sistemas Electrónicos y de Control.