Transcripciones de las presentaciones de clases de teoría 2015

IMPORTANTE: Estas notas de clases sirven como complemento de los apuntes ya editados por esta cátedra y no deben ser considerados como el material didáctico final a estudiar. Se aprovecha en las mismas refrescar ciertos conceptos vertidos en los mismos, complementarlos y actualizarlos.

TEMA 12b: Lenguaje de Descripción de Hardware VHDL:

Filminas 1 a 4:

VHDL es un lenguaje de texto que permite modelizar y sintetizar estructuras.

En términos generales, permite describir el comportamiento de sistemas que trabajen con las reglas de juego, en principio, de lógica booleana, es decir, en esencia, VHDL está pensado para poder describir el comportamiento de circuitos electrónicos (y dispositivos mas complejos aún) DIGITALES, donde se permiten que las variables actuantes tengan básicamente dos valores posibles: 1 y 0 (ó verdadero y falso) y eventualmente un estado de alta impedancia "Z" y un estado X ó don´t care (para la caracterización de entradas). Más allá de esto, se permite en casos donde NO se sintetice nada, otros valores adicionales como "1 fuerte", "1 débil", "0 fuerte", "0 débil", etc. que se emplean solamente para el caso de simulaciones de circuitos que NO serán luego construídos, es decir, virtuales.

Las maneras de describir un sistema digital son varias, lo que le dá al lenguaje una mayor flexibilidad a la hora de diseñar como de simular el comportamiento de un circuito o sistema.

Se puede describir el comportamiento de algo basándonos en la descripción de su estructura.

Por ejemplo, si definimos algo formado por una compuerta AND de 2 entradas cuya salida tiene conectada la entrada de un INVERSOR, estamos describiendo una NAND de 2 entradas. Esta manera de hacerlo coincide con la forma habitual de armar un circuito en base a componentes básicos e interconectándolos según sea necesario (ejemplo en ORCAD, SPICE, etc.).

Otra manera de hacerlo es describiendo que hace el circuito. Por ejemplo podemos decir para el caso anterior que el circuito trabaja tal que cuando sus 2 entradas están en 1, la salida se pone en 0 y para cualquier otra combinación, la salida será siempre 1.

Volvemos a describir una NAND de dos entradas pero de una forma mas ABSTRACTA, con un lenguaje mas parecido al "HUMANO".

Esta manera de descripción, plantea una interesante opción de describir cosas.

Cuando mas nos acerquemos a un lenguaje parecido al nuestro, mejor debería ser la precisión con la cual quede ese circuito o sistema descripto, dando flexibilidad a la hora de querer describir algo muy complejo, por ejemplo un microprocesador.

Existen inclusive herramientas de diseño que permiten diseñar circuitos en base a información temporal de cómo la evolución de las entradas modifican el comportamiento de las salidas.

Pero en cualquier caso, el precio que hay que pagar por tanta “inteligencia” es el desarrollo de herramientas de descripción, análisis y síntesis muy complejas.

De por sí, un compilador gratuito de una EPLD o FPGA es muy complejo, ya que si queremos implementar por ejemplo un simple sumador invocando un símbolo vía esquemático (aquí sólo entramos a una librería y elegimos un sumador), el software debe decidir en base a los datos ingresados como debe interconectar internamente las macroceldas ó los LE (elementos lógicos) para generar ese circuito e interconectarlo a los pines especificados por el usuario.

Por ahora, aunque dispongamos de un excelente compilador nuestro cerebro es el “manager” de todo.

NOTA IMPORTANTE:

Gran parte del éxito de un buen diseño está en describir correctamente lo que queremos.

En sistemas (circuitos) complejos es una buena regla plantear como estrategia, el dibujar en una simple hoja de papel los diagramas en bloque de subsistemas (subcircuitos) previamente desglosados por funciones perfectamente establecidas y desde allí, con ese primer bosquejo, ir armando el sistema circuito por circuito, usando un HDL.

Por ejemplo si necesitamos diseñar un frecuencímetro, podemos descomponer el diseño en bloques como: base de tiempo programable, contador BCD de N dígitos, controlador de display de 7 segmentos, controlador para comunicación serie de datos a PC, etc.

Cada uno puede independientemente ser descripto y simulado en archivos separados en VHDL y posteriormente se interconectan en un archivo final, compilado, simulado y sintetizado en una EPLD ó FPGA.

Filmina 5:

Las ventajas de diseñar con un lenguaje textual potente como el VHDL son varias.

Las mas importantes son:

--Diversas maneras de describir lo mismo (mayor flexibilidad en el diseño).

--Permite describir un circuito o sistema ya sea que se sintetice posteriormente o no.

--Permite además simularlos antes de programar el chip en diferentes etapas de diseño (considerando o no las especificaciones temporales) .

El concepto de simulación en VHDL es muy poderoso ya que se puede describir en forma textual las señales de excitación a un circuito ó dispositivo bajo prueba (DUT) y obtener los resultados de las salidas con la posibilidad de agregar alarmas que puedan indicar condiciones

no deseadas de comportamiento que serían difíciles o imposibles de detectar en los simuladores convencionales que se basan en el análisis de un diagrama temporal.

--Al estar estandarizado por la IEEE permite reusar los diseños con otros modelos de chips incluso de diferentes fabricantes lo que permite la portabilidad y universalidad de los mismos.

NOTA: Esto no incluye el usar algún módulo de software especifico de un fabricante que pueda ser invocado desde VHDL. En esos casos lo mas posible sea que otro fabricante o modelo de chip no lo pueda soportar (ejemplo de ello es invocar vía VHDL a un módulo multiplicador en la Cyclone IV y luego pretender usar ese mismo diseño en otro dispositivo que no contenga multiplicadores embebidos).

Aquí se habla de la utilización del set completo de sentencias del lenguaje VHDL.

--La posibilidad de modelizar el concepto de tiempo agrega una importante característica.

Un ejemplo de esto es el diseño de un microprocesador para síntesis en una FPGA.

Todo micro trabaja siempre con memorias físicas llamadas memoria de datos y de programa.

A la hora de simular al mismo en VHDL se puede hacerlo junto con una memoria “virtual”, es decir, una memoria descripta en VHDL pero que jamás será sintetizada en una FPGA.

Dentro de la descripción de su funcionamiento se utilizan las especificaciones temporales de las hojas de datos del fabricante de la memoria real que luego se interconectará físicamente a la FPGA donde estará alojado el microprocesador.

Filminas 6 y 7:

La forma de diseño basado en VHDL, implica que luego de la etapa de “planteo mental” que haga el diseñador es la siguiente:

Como primer paso hay que escribir el código. Esto puede quedar residente en un archivo o en varios, donde cada uno puede contener diferentes partes del proyecto.

Por ejemplo, para el caso del frecuencímetro, podemos describir el comportamiento de la unidad base de tiempos en un archivo y en otro el bloque contador, etc.

Luego podrá, de la manera adecuada enlazarse dichos archivos, para formar un único sistema a modo de rompecabezas pero en forma escrita y no gráfica.

Teniendo descripto el circuito podemos realizar una primera simulación “cualitativa” ó “funcional” sin considerar en principio el tema temporal, es decir, considerando que el circuito es ideal.

Esta primera aproximación es mas fácil y rápida de simular y puede evitarnos problemas futuros ya que generalmente si no funciona así, tampoco lo haría cuando se consideren los retardos propios de los componentes utilizados.

En esta etapa todavía no se sabe que lógica programable ni bloques internos se usarán en el diseño.

El compilador armará con algún algoritmo un circuito que responda a lo especificado.

La simulación podrá ser basada en un diagrama temporal ó visualizado en forma de texto (si uno ha escrito el test apropiado en VHDL) y mostrará en principio los resultados de las salidas basado en las excitaciones que oportnamente se han aplicado a las entradas.

Si se considera que todo ha salido bien, se prosigue con el diseño. Sino, se deberá encontrar el o los errores hasta que el circuito funcione como debería hacerlo.

Puede avanzarse y eventualmente, realizar una mejor aproximación, considerando el tema temporal, conociendo de antemano el tipo de lógica a emplear (definiendo marca y modelo de FPGA ó EPLD).

Esto nos acerca aún mas a la realidad.

Hasta este punto se puede trabajar con herramientas de software provista por terceros, es decir, por otros que no sean los fabricantes del chip a utilizar.

Por último, en la etapa final, en Place and Route, con las herramientas del fabricante, definimos la marca, tipo de lógica, familia, modelo específico conociendo el, grado de velocidad, se ensamblará el circuito y se podrá simular con bastante exactitud la respuesta del mismo, no sólo cualitativa (que se hizo anteriormente), sino temporal.

Si todo está bien, entonces podemos pasar a la etapa de programación del chip, el cual ya estará generalmente en el circuito impreso.

La prueba final la tendremos que hacer ya en “hardware”, eventualmente simulando señales físicas si es posible, o con todo el sistema interconectado.

El mercado ofrece varias variantes para cada una de las etapas de diseño.

Hay empresas dedicadas al diseño y simulación que pueden ser mas flexibles y poderosas en las primeras etapas que las proporcionadas por los fabricantes de chips.

NOTA: Hay que tener presente que los fabricantes de chips como Xilinx, Altera, Actel, QuicLogic, etc. proveen software de diseño con versiones gratuitadivs, éstas no pueden competir en cuanto a versatilidad y “poder de diseño” con aquellas “pagas”.

El precio que se paga al ser gratis es disponer de un compilador “medianamente inteligente” y con diversas restricciones como por ejemplo el tipo de chip que puede diseñarse ó que permite hacerlo pero no programarlo.

Filmina 8:

VHDL es un lenguaje textual que permite la descripción, modelado y simulación de circuitos y sistemas lógicos digitales.

La idea alrededor del diseño es la de permitir describir cosas en forma ordenada y estructurada y en forma jerárquica.

Podemos describir un frecuencímetro en principio como una caja negra con todo incluído en ella donde sabemos cuales son las entradas y salidas.

Luego, siendo mas específicos, podemos empezar a armar esa caja negra con bloques cada uno con una función determinada: por ejemplo un bloque será el contador de N dígitos BCD, otro será la base de tiempo y así siguiendo.

Esta forma de armado tipo top-down (de arriba para abajo) permite ir componiendo el proyecto de una manera mas sencilla, desmenuzando al sistema ó circuito en partes cada vez mas pequeñas (el bloque de base de tiempo, a su vez puede estar formado por otros bloques mas pequeños).

La ventaja que tiene este tipo de forma de describir es que nos obliga a ser las veces de un primer compilador, ordenando las cosas y describiendo la estructura general de forma mas clara, en resúmen, saber que es lo que estamos construyendo, tanto en la idea general de lo que debe hacer como de qué está hecho esa caja negra.

Filminas 9 a 11:

Las librerías al igual que las que se pueden encontrar en lenguajes como “C” sirven para agrupar y ordenar piezas de código que eventualmente son utilizadas de manera frecuente, permitiendo reusar Funciones, Procesos, etc. y compartirlos en otros diseños como así también definir procedimientos en el tratamiento de operaciones matemáticas, lógicas, manejo de entrada y salida de texto, etc..

Esto ayuda enormemente al diseño ya de de esta manera es mucho menos lo que hay que escribir.

Los paquetes mas comunes son Librerías de la IEEE, standard y work.

En IEEE se define, entre otras cosas, las características del comportamiento eléctrico de los pines de entrada y salida de un proyecto y la definición en el manejo de datos de entrada y salida en operaciones matemáticas.

Por ejemplo: Ieee.std_logic_1164.all define las características del comportamiento eléctrico de los pines de entrada-salida de un proyecto.

Ieee.std_logic_arith.all especifica los tipos de datos aceptados para diversas operaciones matemáticas , de comparación y conversión de datos.

Las librerías “std” y “work” son las que rigen el comportamiento del sistema en el manejo de texto ya sea para visualización, impresión y manejo de archivos de datos para lectura-escritura.

En particular, estas librerías vienen “embebidas” en el programa y por lo tanto no deben ser invocadas en el proyecto que se inicia.

Filminas 12 y 13:

Se muestra a modo de ejemplo parte del contenido de una de las librerías (ieee_std_arith). En ella se pueden ver como se definen las funciones de suma y resta, comparaciones y conversiones de diferentes tipos de datos.

Filmina 14:

Todo proyecto debe contener una entidad (“entity”) la cual debe tener un nombre que la designe e identificar los puertos de entrada y salida.

Se la puede considerar como una caja negra que contiene el diseño.

Hay dos tipos de Entity:

- ✓ La que generalmente se utiliza, que es la que contiene el circuito ó sistema a modelar donde se deben especificar los puertos de entradas y/o salidas.
- ✓ La que se utiliza para contener el procedimiento de test a realizar sobre algún o algunos de los diseños descriptos en VHDL.

Debe notarse que en este nivel alto de jerarquía en la descripción de un diseño, no se hace mención a que hay adentro o que hace lo que hay en el interior. Sólo se describe su conectividad con el mundo exterior, es decir, la denominación y tipo de puertos que tiene dicha entidad.

En el ejemplo tenemos “algo” denominado “mux” que tiene 6 entradas y una sola salida.

Podríamos inferir por el nombre de la entidad y de los puertos, que se trata de un multiplexor, pero sin mas información sobre que hace o con que está construido, sólo podemos conjeturar.

La siguiente especificación necesaria para ello es la denominada “arquitectura” (architecture).

Filmina 15:

Básicamente existen según su función, cuatro tipos de puertos: entrada (IN), salida (OUT), entrada-salida (IN-OUT) y tipo buffer.

Para el caso particular de IN-OUT se puede sintetizar siempre y cuando exista asociado una estructura bidireccional en el hardware formada generalmente por compuertas con capacidad de tri-state.

IMPORTANTE. En las EPLD y FPGA no hay estructuras internas que funcionen en estado de alta impedancia. Sólo se encuentran estas estructuras en los bloques de IN/OUT para manejar las señales de entrada y salida del chip.

Si en el diseño se requiere implementar algo parecido a “tri-state” se deberá sustituirlo con selectores de señales para que una sóla señal esté “activa” por vez.

Filmina 16:

En esta parte del código VHDL, es donde se describe la funcionalidad del modelo del sistema ó componente a desarrollar.

No hay una única manera de hacerlo. Es más, dependiendo de lo que se pretenda modelizar, hay opciones mas convenientes que otras.

IMPORTANTE: En el ejemplo, se describe la función de un MUX 4:1 pero de una manera muy particular ya que se introducen EXPLÍCITAMENTE retardos en la respuesta de la salida Z.

Aquí se plantea una manera de describir el funcionamiento de una manera “virtual” y sirve como se verá mas adelante para especificar aquello que NO será luego sintetizado. Sólo queda en las simulaciones.

En el diseño de sistemas y componentes en FPGA, por ejemplo, NO SE ESPECIFICAN RETARDOS ya que éstos serán producto de la síntesis final al cargar el diseño realizado sobre un dado chip.

La velocidad de repuesta, del conjunto y componentes individuales creados dentro de la FPGA quedarán definidos por la elección del tipo de dispositivo, marca, modelo y las interconexiones internas que realizó el compilador para la configuración final del hardware del circuito.

Filminas 17, 18 y 19:

VHDL dispone de la capacidad de definir los llamados TIPOS y SUBTIPOS de “cosas”.

Con ello pueden definirse distintos formatos de números, generando así los ENTEROS, PUNTO FLOTANTE, etc.

Una posible clasificación es la siguiente:

Escalares: Enteros, punto flotante, magnitudes físicas (la más importante es tiempo), por enumeración, etc..

Compuestos: Matrices y vectores.

De acceso: punteros.

Archivos: Para manejo de entrada y salida de datos en un sistema de archivos.

Por otro lado los SUBTIPOS son una clase particular dentro de un TIPO dado.

Ejemplo: Podemos definir dentro del TIPO byte (formado por un número binario de 8 bits), un SUBTIPO nible (formado por un número binario de 4 bits).

Una de las formas mas utilizadas para especificar un dato formado por un grupo de entradas o salidas, es la de agrupadas dentro de un formato tipo array:

TYPE byte is array (NATURAL range 7 downto 0) of BIT;

TYPE byte is array (NATURAL range 0 upto 7) of BIT;

Ambas maneras definen lo mismo pero se enumeran al revés.

El TIPO “BIT” ya está contemplado dentro de las librerías de VHDL y no es necesario de definirla cada vez que se quiera utilizar.

“Constante” como su nombre lo indica, permite definir un con un valor fijo a un dato (sea numérico o no).

Como se verá mas adelante, por ejemplo, se puede describir un sumador de n bits siendo ese “n” definido por una constante al principio de la descripción.

Si se hace adecuadamente, bastará con cambiar el valor de esa constante para cambiar la longitud de bits del sumador.

Variables son un tipo de dato similar a lo que ocurre en un programa de software.

Son dinámicas y locales dentro de un proceso. Generalmente son “virtuales” y no forman parte de un hardware, sino que sirven en el lenguaje para ser usadas en sentencias que permitan especificar mejor al sistema o componente a crear.

Se usan dentro de procesos y en ellos se ejecutan secuencialmente según el órden de aparición en el mismo.

Mas adelante se verá como crear un sumador riple-carry de “n” bits en base a un sumador completo de 1 bit, empleando un lazo FOR de “n” iteraciones empleando variables.

Las Señales sirven para interconectar bloques funcionales entre sí, por ejemplo en un diseño, pueden unir un contador con un MUX para armar un generador de funciones programable.

Son en general globales. Si bien en la especificación de algo una señal sirve de vínculo entre componentes, el compilador al final del análisis puede convertir esas señales en cables o nó. Dependerá de la estructura interna del dispositivo.

Filmina 20:

Operadores: Análogo a la definición en los lenguajes de diseño de software.

Permiten realizar operaciones matemáticas, lógicas, etc.. entre datos de diferente tipo, inclusive.

Filmina 21:

Hay en principio dos tipos de descripciones en la manera de escribir el código:

Sentencias concurrentes.

Sentencias secuenciales.

En el primer caso (concurrente), no importa el órden en que se escriban las ecuaciones: el compilador cuando lee, considerará que todos los eventos se ejecutarán en paralelo.

Si escribo en el siguiente órden:

D <= F OR A; (D es la salida de la function que hace la OR entre las entradas A y F)

A <=B AND C; (A es la salida de la function que hace la AND entre las entradas B y C)

El compilador considerará primero la AND entre B y C y su salida que es A, la conectará a algo para que haga la OR con la entrada F, siendo la salida D.

NOTA: Dependiendo del tipo de dispositivo que se emplee para hacer el diseño, el compilador puede usar una estrategia que difiera de las ecuaciones planteadas, es decir, puede al final, resolverlo con uno o dos MUX´s por ejemplo si se trata de una FPGA.

Lo importante aquí es como el compilador interpretará las sentencias que escribamos en nuestro código VHDL.

El segundo caso, que es el de las sentencias secuenciales, éstas se usan dentro de los denominados “Procesos”.

Aquí si que importa el órden de cómo escribamos las sentencias.

Por ejemplo, como se verá mas adelante, para describir un Flip-Flop sincrónico, el órden en donde se escriba la acción de la entrada de RESET (antes o después de especificar el reloj), cambia totalmente la acción del RESET: en un caso será “asincónico” y en el otro “sincrónico” con el reloj.

Filmina 22

En VHDL no hay una única forma de especificar el diseño de un sistema.

Una clasificación desde el punto de vista del modo en que se hace es la siguiente:

- ✓ Esctructural.
- ✓ Por comportamiento.

“Estructural” es como cuando uno diseña con lógica standard: Planteamos el circuito como un conjunto de componentes interconectados entre sí (por ejemplo chips de compuertas, flip-flops, contadores, etc..) a fin de funcionar de la manera especificada.

Por “Comportamiento” es cuando no tenemos que decir que voy a usar, sino como debe responder el sistema ante determinadas excitaciones.

En general en un diseño de mediana a gran complejidad, es muy común que se empleen ambos tipos de modo de diseño.

El segundo tipo, tiene la particularidad de ser de mayor nivel de abstracción, el cual permite en forma mas sintética y menos laboriosa para el diseñador, especificar el componente o sistema deseado. El compilador es el que tiene la tarea mas ardua de ver como interpretar la entrada de información de diseño del usuario y convertirla luego en hardware.

Sin embargo, el primer tipo, es mas determinístico, ya que es el diseñador el que tuvo que hacer un trabajo de “inteligencia” (en vez de usar un compilador) para definir bloques y luego interconectarlos entre sí. Se tiene un mejor control de lo que se hace, pero insume mucho mas tiempo y si el diseño es complejo, puede resultar en equivocaciones difíciles de hallar,

Filminas 23 y 24:

El diseño estructural es el que conlleva el menor nivel de abstracción en cuanto al nivel de inteligencia de diseño por parte del compilador.

Este es un nivel básico de representación donde el diseñador humano conoce los componentes que maneja y sabe como interconectarlos.

Como ejemplo supongamos que con lógica estándar hay que diseñar con compuertas una función OR-Exclusiva de 2 entradas. Para ello necesito dos inversores, dos compuertas ANDde 2 entradas y una OR de 2 entradas. Con el diagrama eléctrico en mente, realizo las interconexiones necesarias.

En VHDL hacer esto, se puede lograr con el denominado diseño “estructural”.

En el ejemplo, se describe un MUX 4:1 diseñado con compuertas básicas.

NOTA: Aquí sólo se describe como se interconectan dichas compuertas. En otra parte del diseño hay que definir que hace o cómo están diseñadas cada compuertas.

Lo que se pretende destacar aquí, es solamente el modo en que se describe la interrelación entre compomentes que definen al MUX.

Como se verá en breve, hay otras maneras de describir un MUX mucho más simples y con un lenguaje más compacto y potente.

Esto no quiere decir que el tipo de descripción por “estructuras” sea primitivo sino que generalmente se utiliza para la interconexión de bloques de relativa media a alta complejidad, en los niveles mas altos de diseño.

Por ejemplo, en el diseño de un microprocesador puede haber descripciones de los componentes internos del mismo (una ALU, Bloque de registros y la Unidad Central de Procesos, etc.) hechas con algún tipo de descripción y en el nivel más alto del desarrollo, las interconexiones entre ellos se hace con el tipo “estructural”.

Filminas 25, 26 y 27:

El otro tipo de forma diferente de describir un sistema o circuito es el denominado “por comportamiento” ó “behavior”.

Algunos autores dividen a este en dos tipos: “algorítmico” y “por flujo de datos”.

En el algorítmico, se utilizan para describir cosas, funciones similares a los programas para generación de software de alto nivel (C, pascal, etc.) tales como IF, CASE, Lazos FOR, etc..

Esto es muy poderoso ya que se puede diseñar sistemas muy complejos con un nivel de abstracción elevado.

Hasta que nivel se puede llegar, dependerá de la habilidad del diseñador y fundamentalmente de la complejidad del “compilador” y esto dependerá de si es una versión “paga” o “gratuita” (se supone que la primera debería dar mayores satisfacciones).

En esta sección,se dan dos ejemplos de cómo describir un MUX 2:1 mediante sentencias CASE e IF.

Dado que se usan dentro de una función “PROCESS” lo que se escriba adentro de ella el compilador lo ejecutará secuencialmente (siguiendo el órden de lo escrito) para generar el hardware solicitado (si es que se quiere implementarlo) o describirlo si es sólo para su simulación.

NOTA: Se verá más adelante en Flip-flops, la importancia del órden de las sentencias.

En el ejemplo con “CASE”, esta función evalúa a “sel” que es la entrada de selección de datos y dependerá de su valor (sólo 2 en este caso) para que la salida del MUX denominada “output” adquiera alguno de los valores de las entradas “input0” ó “input1”.

NOTA: En el caso en que se use esta función y queden alguna combinación de valores para evaluar sin ser usados, es de buena práctica de diseño, poner al final la cláusura “when others ……” a fin de que no se generen problemas en la síntesis como por ejemplo inferir (crear) elementos de memoria innecesarios. Se verá esto mas adelante.

El segundo ejemplo, usa una sentencia IF. Similarmente a lenguajes computacionales, existe la posibilidad de emplear cláusuras ELSE y anidar varias sentencias IF.

Se debe tener especial cuidado con esto ya que las evaluaciones que hará el compilador al analizar luego lo escrito, es secuencial como se dijo y ejecutará las acciones para la síntesis en base al órden en que se escribieron las sentencias.

En este caso sencillo, IF evalúa a “sel” y dependiendo de su valor la salida tomará el valor de “input0” ó “input1”.

NOTA: IF tiene como ventaja que puede evaluar mas de un dato a al vez, cosa que “CASE” no lo puede hacer. Pero CASE es mas determinístico en su construcción y en varios casos conviene emplearlo en lugar a IF, por las razones antes explicadas (inferir lógica innecesaria y/o incurrir en errores por anidamientos o mal uso de las cláusuras IF-ELSE).

Respecto de la función PROCESS, como se puede observar en ambos ejemplos, hay declaraciones entre paréntesis. Esto es lo que se denomina “lista sensible”, es decir, escribir los datos que PROCESS usará para activarse (en este caso las entradas de datos).

Aquí se presenta una sutileza del lenguaje de VHDL. Uno podría pensar que “sel” debería estar en la lista de sensibilidad, pero lo que aquí se evalúa es si hay algún cambio en las entradas de datos. Si la hay, dependerá de “sel” en como la salida cambie.

Filmina 28:

Otra manera de describir algo en el modo “BEHAVIOR” o por comportamiento es en la modalidad denominada “por flujo de datos” donde no se requiere un PROCESS.

En el ejemplo se describe simplemente una compuerta de 2 AND entradas con la función primitiva “and”.

Filminas 29 y 30:

Aquí se describe el funcionamiento de un buffer óctuple no inversor con salida “tri-state”. Para ello de utiliza la sentencia WHEN – ELSE.

Notar la sencillez de la descripción en cuanto a la cantidad de líneas de código empleadas.

La entrada “habilita” es evaluada en la sentencia para saber que valor tendrá “salida” que es en realidad un vector formado por 8 líneas, es decir, hay 8 buffers no inversores con 8 entradas-salidas independientes.

Dado que se ha empleado la librería “ieee.std_logic_1164” para definir las entradas y salidas, se contempla el tercer estado lógico (alta impedancia) denominado como “Z”.

Se puede ver en el simulador como las salidas entran en este estado al ser “habilita” adopta el valor “1”.

Filmina 31:

Aquí se emplea de nuevo WHEN – ELSE para describir un MUX 2:1 de una manera mucho mas compacta que con “CASE” ó “IF –ELSE”, sin necesidad de PROCESS y líneas extras de código.

Es importante aclarar que no hay una única manera de describir algo. En algunos casos ciertas sentencias son mas convenientes que otras.

Filmina 32:

Otra manera de describir lo mismo, empleando una sentencia “WITH-SELECT”.

Un compilador que lea lo descripto antes con “WITH-SELECT”, “CASE”, “IF-ELSE” ó “WHEN-ELSE” generará el mismo hardware.

Filmina 33:

Asignación concurrente:

Como se comentó, y todo lo que se escriba en este caso, el compilador lo analizará como que todo se realiza en el mismo tiempo, es decir, en paralelo. No importa el órden de cómo se escriban las cosas (por ejemplo en la lista de las entradas de la sentencia WITH-SELECT” que definen el valor de “salida”).

En este ejemplo se describe el comportamiento de un decodificador BCD a 7 segmentos empleando la sentencia “WITH-SELECT”.

Evaluando “bcdin” que puede adoptar 10 valores diferentes, las salidas 7 salidas adoptarán los valores lógicos según se indica en binario.

Notar que bcdin puede adoptar 16 valores en total (4 bits), pero sólo 10 son requeridos.

Para evitar que los otros 6 queden indefinidos, si se presentan hará que las salidas adopten el valor “0”. Esto, a través de la cláusura “WHEN OTHERS”.

Filmina 34:

Otro ejemplo de “Asignación concurrente”:

Esta vez se trata de la descripción de un decodificador “2 a 4”.

Filmina 35:

Asignación secuencial:

Aquí se dá un ejemplo de asignación secuencial. Al haber un PROCESS (como en los ejemplos anteriormente vistos) sí importa el órden en como se escriben las sentencia.

En este caso tenemos la descripción de un sumador ripple-carry de 4 bits, basado en describir un sumador completo de 1 bit y repetirlo 4 veces empleando una sentencia “FOR-LOOP”.

Como se puede apreciar esto tiene un nivel de abstracción elevado ya que la forma de describirlo es básicamente empleando algoritmos computacionales para sintetizar una cadena de bloques idénticos que se repitan en hardware además de estar convenientemente interconectados para formar el dispositivo propuesto.

Empezamos declarando a la entidad “sumador”.

A fin de darle flexibilidad al diseño, insertamos una sentencia “generic” para poder modificar a voluntad el número de bits del sumador, solamente cambiando el dato “n_bits”, sin tener que modificar nada mas en el programa.

En “ARCHITECTURE” llamamos a la descripción “sumador_rc” simplemente para recordar que su descripción responde a un sumador ripple-carry.

En lo que sigue, dado que se quiere describir un sumador cuya estructura se basa en un bloque que debe ser replicado “n_bits2 veces e interconectando cada uno de ellos en forma sucesiva, se debe emplear un PROCESS para contener tal descripción.

Siendo “a” y “b” las entradas paralelo del sumador y “c_in” el eventual carry de entrada al mismo, dichas entradas deberían formar parte de la lista sensible de PROCESS.

La idea es usar en este caso, una sentencia “FOR – LOOP” para generar un lazo “n_bits” veces.

Dado que en cada pasada se deben generar valores para el bit de salida del sumador y del carry a la siguiente etapa, se emplean dos variables donde la información del bit de suma para cada iteración estará contenido en el arreglo de “n_bits” denominado “vsuma”.

Como el carry de cada etapa no es necesario de tenerlo (salvo para el último bit mas significativo), se define una variable unidimensional “carry”.

En cada pasada por el lazo FOR, se genera según las ecuaciones vistas anteriormente, los valores de vsuma(i) y el carry (i+1) (aquí denominado siempre como “carry”).

Para cada iteración del lazo FOR, se procesarán los datos que correspondan a la posición de bit del caso, es decir, a(i), b(i) y carry.

Al terminar el lazo FOR, sólo restatransferir la información de la variable “vsuma” que es un arreglo de “n_bits” a la salida “suma”.

Lo mismo con la información de “carry” a la salida “c_out”.

Filmina 36:

En esta filmina se muestra otro ejemplo para describir lo mismo, pero utilizando la función “FOR - GENERATE”.

Dado que es una sentencia concurrente, aquí no se emplea ningún PROCESS.

De esta forma se hace una descripción más compacta que la anterior ya que se eliminaron las variables y los valores de los bits correspondientes del arreglo de salida “sum” son definidos en cada iteración.

A partir de aquí se describirán diversos tipos de estructuras ya estudiadas: desde compuertas hasta una unidad aritmético-lógica.

Filmina 37:

Aquí se describe el comportamiento de una compuerta AND de 2 entradas usando la función primitiva “and” dentro de un PROCESS.

Filmina 38:

Se describe un “latch” que para algunos autores es un almacenador de información “asincrónico” y para otros “sincrónico”.

En este caso es asincrónico ya que no hay involucrada alguna señal de reloj.

La salida “q” sólo adquiere valores de la entrada “d” cuando la entrada de control “enable” esté en “1”. Para “enable” = 0, la salida “q” mantiene su último valor actualizado.

Filmina 39:

Aquí describimos por primera vez a un Flip-Flop tipo “D” empleando dentro de un PROCESS la sentencia WAIT UNTIL.

Lo que se evalúa aquí es la señal “reloj” donde la salida “q” del FF sólo adoptará el valor de la entrada “d”, cuando se haya determinado que reloj sea “1”.

Esta forma de describir al FF es activado por NIVEL y no por flanco.

Para hacerlo por flanco, se debe detectar que hubo cambio de nivel del reloj y además comprobar que quedó en el nivel correcto:

WAIT UNTIL (reloj`EVENT and reloj=´1´) analiza cuando llega el flanco ascendente de reloj.

WAIT UNTIL (reloj`EVENT and reloj=´0´) analiza cuando llega el flanco descendente de reloj.

Filmina 40:

Aquí se describe un FF tipo “D” con entrada de reset “asincrónico” y señal de “enable”.

Como se debe recordar, el órden de las sentencias en un PROCESS es importante. Como aquí la acción de “reset” está antes que la evaluación del estado del reloj, convierte la entrada de “reset” de este FF tipo “D” en ASINCRÓNICO.

NOTA: Aquí se empleó la sentencia “rising_edge (CLK)” que permite ALTERA para detectar un flanco ascendente de reloj. Puede esto no ser compatible para otros fabricantes como XILINX.

Lo usual, como sentencia standard en VHDL, es usar (CLK´EVENT AND CLK=´1´) para detectar un flanco ascendente.

Este FF tiene además una entrada de habilitación de reloj “enable”, la cual permite inhibir el comportamiento del mismo si está en “0”.

Filmina 41:

Este caso es similar al anterior, salvo que reset se evalúa después de evaluar a la señal de reloj, por lo cual convierte a este Flip-Flop en uno con entrada de reset SINCRÓNICA.

Filminas 42 y 43

Este es un caso generalizado. Aquí se describe un registro sincrónico de 8 bits con entrada de reset sincrónico, que actúa sobre todos los FF´s.

Se puede observar en la simulación cómo actúa el registro ante diversos estímulos: Por ejemplo, el primer pulso positivo de reset, no borra las salidas ya que temporalmente lo hace cuando no hay flanco de subida del reloj. Por lo tanto el registro no considera la órden de borrar como válida. Sí ocurre, en el segundo pulso positivo de reset, dado que en ese intervalo, existe un flanco ascendente de reloj.

Filminas 44 y 45:

Este es un ejemplo de cómo se puede describir el funcionamiento de un contador. En este caso se trata de un contador binario progresivo de 32 bits con reset asincrónico y comando para carga paralela en modo sincrónico.

Aquí se puede apreciar la potencialidad de VHDL, dado que con una simple función de suma, es posible ordenar al compilador que implemente la función de incremento automático del número que presenta el contador a su salida, cada vez que se registra un flanco activo en la entrada de reloj.

De forma relativamente sencilla, se puede agregar a este contador una entrada adicional que sirva de comando para seleccionar si el mismo cuenta en forma progresiva o regresiva.

Esto puede ser descripto empleando otra función IF después de la declaración de LOAD tal que dependiendo del nivel de esa entrada de comando, la operación que se realice sea cnt < cnt +1 ó cnt <= cnt -1.

Filmina 46:

Aquí se describe un registro de desplazamiento (RD), paralelo-serie con carga sincrónica.

Lo interesante de este ejemplo es que utilizando una descripción algorítmica es posible simplificar su diseño.

En este caso se emplea el carácter “&” ó de “concatenación” donde la clave está en la línea:

ELSE reg <= reg(6 DOWNTO 0) & ‘0’;

Esto se traduce en que cuando haya un flanco válido de reloj y la entrada load sea “0”la información contenida en las salidas del RD, se desplazarán un lugar (hacia la derecha, según el esquema) y el bit LSB (Q0) se carga con “0”, es decir. Dicho de otra manera: El valor de Q0, Q1, …Q6 antes del flanco, al venir éste, tendrán nuevos contenidos que serán: Q0 = “0”, Q1 = Q0(anterior), Q2 = Q1(anterior), etc, etc. , es decir, se desplazó la información un lugar hacia el lado de los bits mas significativos.

Filminas 47 – 48 y 49:

Otro RD, es descripto aquí. Esta vez es un RD serie-serie, el cual tiene como una de sus aplicaciones, servir como línea de retardo digital de 64 bits.

Simplemente se define en un Process una señal que es un arreglo de 64 elementos, donde cada vez que se recibe un flanco de reloj activo, se desplaza en un bit la información desde la posición LSB a MSB.

Filminas 50 y 51:

Aquí tenemos una máquina de estado de Moore que describe a un monoestable, disparado por flanco ascendente de la señal de entrada “input”.

Al detectarse ese flanco, la salida vá a “1” durante un ciclo de reloj y vuelve a bajar en el siguiente ciclo.

Con la ayuda de una sentencia “CASE” y sentencias “IF” se pueden contemplar los diferentes casos, en los 3 estados definidos.

Para ello se define “TYPE” como tipo “STATE_TYPE” formado por 3 elementos:s0, s1 y s2.

La variable asociada es “state” que adoptará alguno de esos estados, dependiendo en que estado se encuentre y que valor tenga la entrada “input” al venir un flanco activo de reloj.

NOTA: Hay ambientes de desarrollo que permiten realizar una representación gráfica para definir la máquina de estados. En nuestro caso, sólo cabe la opción de escribir la tabla de estados.

Filminas 52 y 53:

Este ejemplo describe un barrel shifter aritmético de desplazamiento hacia derecha que puede ser usado como un multiplicador arbitrario.

Esta formado por 3 grupos de MUX´s 2:1.

La primera línea comandada por sel(0) modifica los 8 bits de entrada.

Si sel(0)=”0” la Fila 1 no altera el resultado. De lo contrario en Fila 1, se tendrá el valor de la entrada multiplicado x2.

La entrada sel(1) hace lo mismo, donde si sel(0) =”0”, la entrada será la misma o multiplicada x4, según el valor de sel(1).

Idem con sel(2), donde se puede multiplicar x 16 si sel(0) = sel(1) = “0”.

Y con las otras opciones restantes, se pueden lograr otros valores según los estados de sel(2), sel(1) y sel(0).

Lo interesante de la descripción que se hace aquí es el uso de la sentencia FOR .. LOOP, donde se puede definir la salida de cada FILA empleando una ecuación.

Filminas 54 y 55:

En este ejemplo se describe un comparador numérico con signo entre dos números de 8 bits cada uno. La descripción de “a” y”b”, definen en este caso que las operaciones se harán “con signo”.

En la filmina55, se observa una simulación, donde aparecen “glitches” debido a las diferencias de retardo involucrado en la lógica que decide el estado final de cada salida.

Esto se puede subsanar con el agregado de una etapa basada en registros que sólo actualicen la información cuando haya un flanco de reloj válido, evitando así las variaciones temporales de las salidas cuando se modifiquen las entradas.

Filminas 56, 57, 58 y 59:

Se describe una unidad aritmético-lógica de 8 bits.

El diseño consta de 3 partes:

1 – Descripción de las operaciones lógicas.

2 – Descripción de las operaciones aritméticas.

3 – Selección del tipo de operación a realizar.

En este ejemplo, las operaciones lógicas y aritméticas se realizan simultáneamente.

Las 3 entradas LSB del array Sel(3..0), son las que definen que función se aplicará.

La entrada Sel(3) es la que selecciona cual salida (la del bloque aritmético ó la del bloque lógico), se conectará con la salida de la ALU.

NOTA: Una de las operaciones lógicas que se implementan es la de “swapping”.

El contenido del operador de entrada “a” se divide en dos “nibbles” (a(7...4) y a(3…0)) y se permutan los nibbles dando como resultado a(3…0) como más significativo y a(7…4) como menos significativo.

Esta función es útil para la implementación de varios algoritmos.

Filminas 60 y 61: Ejemplo de diseño de un circuito anti-rebote.

Filmina 60:

En este caso se muestra el esquemático del circuito. El circuito trabaja tal que la salida "result" sólo puede cambiar si las salidas de FF1 y FF2 se mantienen idénticas durante todo el conteo de "counter" desde "0……0" hasta que se pone el MSB en "1". Eses es el momento en que se habilita FF3 y permite copiar el dato de salida de FF2 en "result".

Si los datos de FF1 y FF2 difieren, se borra sincrónicamente el contador y debe empezar otro nuevo ciclo.

De esta manera, se prevee que si la entrada "button"cambia de nivel lógico pero con oscilaciones hasta quedar en el nuevo nivel lógico definitivo, "result" no se modificará si al menos dichas variaciones no superan el tiempo en el que el contador cuenta desde 0 hasta "1000…….000".

Esto equivale a un tiempo de 1/50 MHz x 2^{19} segundos ó 10, 48 ms.

Filmina 61:

Aquí aparece una posible descripción en VHDL del circuito presentado.

counter_set es la señal similar a la salida de la OR-EXCL.

El proceso evalúa si hay una subida de clk.

Si sucede, actualiza las señales "flipflops(0)" y "flipflops(1)".

Esto infiere dos flipflops tipo "D" y están en en cascada [flipflops(1) <= flipflops(0)].

Se infiere además un contador con la señal "counter_out".

En su funcionamiento, se evalúa primero la señal "counter_set". Si vale "1", se borra sincrónicamente al contador [counter_out <= (others => ´0´)].

Si vale "0", éste contará sólo cuando la salida MSB que es "counter_out(counter_size)" sea "0".

Caso contrario (counter_out(counter_size) = ´1´), habilitará al FF3 y éste copiará la salida de FF2.

Filminas 62 y 63:

Como se comentó, existen ciertas descripciones que no sirven para la síntesis de circuitos digitales, pero sirven para propósitos de test.

Un ejemplo de esto es el de describir una memoria paralelo sincrónica para ser simulada en VHDL junto con un microprocesador que sí se quiere implementar.

La descripción de la misma generalmente implica la especificación temporal de retardos cuando ésta es accedida por ejemplo para leer ó escribir.

Se dispone en VHDL una serie de sentencias que ayudan a describir diferentes comportamientos de un bloque lógico desde el punto de vista temporal.

Dos de ellas son “AFTER” y “TRANSPORT AFTER”.

La diferencia entre ambas, es que la primera tiene un umbral mínimo de aceptación de ancho de pulso de la señal de entrada y la segunda, no.

En el ejemplo de la filmina 60, la entrada “a” genera un pulso de 10 ns y la salida “b”del bloque no reaccionará ya que el parámetro de AFTER es de 20ns.

Caso contrario ocurre en “TRANSPORT AFTER” donde “b” copia a “a” con una demora de 20ns.

Filminas desde 64:

Existen en VHDL básicamente dos formas de poder realizar una comprobación del diseño lógico.

1 - Usando un simulador temporal.

2 - Usando un procedimiento escrito en VHDL, denominado TEST BENCH.

El primero es muy simple y para diseños elementales (poca lógica y número de entradas y salidas) puede resultar bastante útil.

La decisión de si pasa o no el diseño, dependerá del buen ojo del diseñador y su habilidad para tener en cuenta todos los casos posibles de excitaciones de las entradas para saber como evolucionan las salidas.

La segunda, es decir, desarrollar un procedimiento escrito en VHDL, puede ser tedioso y "pesado" pero tiene grandes ventajas frente al caso anterior:

Permite generar fácilmente las excitaciones necesarias de las entradas hacia el circuito a "testear" ordenándolas en forma temporal.

Puede incluso disponer de archivos de entrada de datos y generar archivos de salida con los resultados.

Permite realizar un "debugging" mucho más fino del comportamiento de las diferentes señales involucradas, con la inclusión de sentencias que reaccionen de una determinada manera si se encuentra alguna condición previamente "cargada".

Ej. La detección de glitches que puede ser que con el método anterior se le"escapen" al observador en el diagrama de tiempos.

En el ejemplo que se dá aquí, algo muy básico, se plantea un "híbrido", es decir, se testea una compuerta AND2 donde la estimulación de las entradas proviene de un procedimiento escrito en VHDL y el análisis de las salidas se realiza en un simulador digital que muestra los resultados en un diagrama temporal.

Si por ejemplo quisiéramos un control de la salida de la compuerta AND para detectar alguna condición ó condiciones específicas se puede utilizar la sentencia ASSERT.

Por ejemplo si esperamos que la salida de la AND denominada "salida" en un determinado momento sea "1" y no lo sea, podemos incluir en el código lo siguiente:

```
ASSERT salida="1"

REPORT "Error: La salida dió "0" cuando se esperaba un "1" "

SEVERITY NOTE;
```