EL ENTORNO DE DISEÑO WARP2: UNA APROXIMACIÓN AL LENGUAJE VHDL.

A. Rosado, M. Bataller, J. Guerrero, J.V. Francés, J.R. Magdalena.
GPDS. Dpto. Electrónica e Informática. Universidad de Valencia.
C/ Dr. Moliner, 50. 46100 Burjassot. VALENCIA.
Tno.: (96)3864300 Ext. 3397. Fax: (96)3864568
e-mail: alfredo.rosado@uv.es

RESUMEN: Con el programa Warp 2 se pretende introducir un compilador VHDL que permite un acercamiento de los alumnos a dicho lenguaje con la adicional ventaja de permitir la simulación del diseño y la obtención del fichero necesario para programar el dispositivo físico, todo ello dentro de un mismo entorno y con la simplicidad añadida de ejecutarse bajo Windows. Para conseguir un dominio en la programación de VHDL se proponen algunas prácticas que contemplen los diferentes aspectos sintácticos del lenguaje y permitan un dominio del entorno Warp 2.

1.- INTRODUCCIÓN.

El programa Warp 2 ha sido desarrollado por Cypress Semiconductors para la programación de sus dispositivos a través del lenguaje VHDL. Dado que Warp 2 se va a emplear con fines docentes, el principal objetivo que se persigue es la introducción al alumno en la descripción de sistemas digitales empleando VHDL, para ello se explican los recursos de que dispone dicho lenguaje así como la sintaxis y posibilidad de jerarquización del diseño a través de los llamados paquetes (packages).
En esta versión del programa sólo se permite la programación de dispositivos de tipo PLD y CPLD, las versiones Warp 2+ y Warp 3 permiten la síntesis y grabación de FPGA de la familia pASIC 380 de Cypress, aunque Warp 2 resulta suficiente para los objetivos marcados como son la obtención de un compilador VHDL a bajo coste y de fácil manejo.
Estas prácticas se enmarcan dentro de la asignatura de "Diseño de Circuitos y Sistemas Electrónicos" perteneciente al 2° cuatrimestre de 4° curso de Ingeniería Electrónica impartido en la Universidad de Valencia, donde la enseñanza de VHDL se comparte con el diseño de lógica programable y el empleo de otro tipo de herramientas para la descripción de sistemas lógicos. Dado que se trata de un tiempo relativamente corto para la realización de un alto número de prácticas, lo que se pretende con las sesiones de laboratorio es mostrar al alumno los recursos del lenguaje a través de prácticas "modelo".

2.- DESCRIPCIÓN DEL SISTEMA.

En primer lugar, las descripciones en VHDL se realizan a través de un editor de texto ASCII como puede ser el bloc de notas de Windows. Una vez creada correctamente la especificación del diseño, se ejecutan los pasos de síntesis a través del programa GALAXY [1] y [2]

encargado de la compilación, verificación de la sintaxis, inclusión de elementos propios de librería para englobar todo el diseño en un sólo fichero compilado donde Warp 2 detecta automáticamente el fichero de mayor jerarquía, indicando en este paso si únicamente se desea compilar, o compilar y sintetizar el diseño, por lo que si se pretende sintetizar, se debe especificar el tipo de dispositivo en el que se va a implementar, de esta forma se compila el diseño y se generan los ficheros de programación.
Una vez generado el fichero de programación compilado de tipo JEDEC (formato generado automáticamente cuando se trata de un PLD o CPLD), a través de NOVA [1] y [2] se procede a la simulación para comprobar el correcto funcionamiento del diseño realizado. Los pasos que se deben seguir en el diseño de sistemas a través de Warp 2 vienen descritos en la figura 1. En la pantalla de NOVA se pueden visualizar todas las señales que se deseen a través del menú Views. Con el resto de menús, asignamos valores a las entradas del sistema y podemos analizar la salida una vez ejecutada la simulación para cada una de las entradas que se han especificado. Este simulador permite añadir vectores de test al fichero JEDEC para poder comprobar el sistema una vez programado así como la posibilidad de guardar los vectores para posteriores simulaciones.

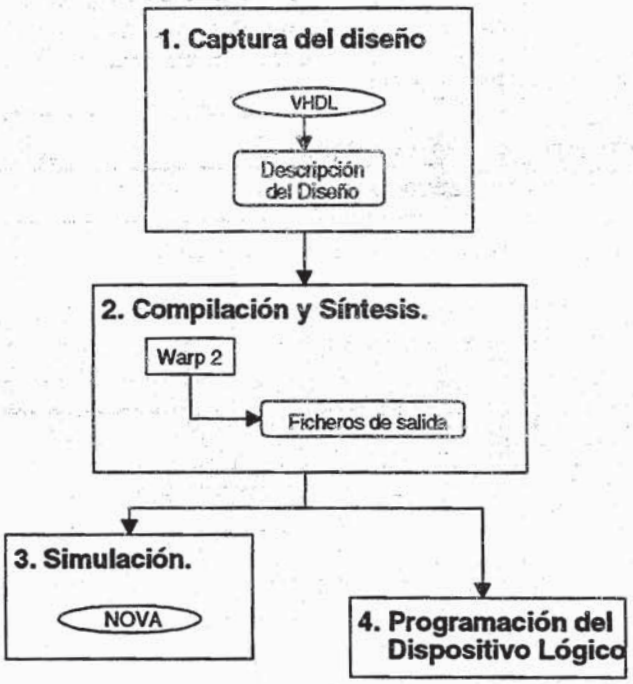

Figura 1.- Flujo de diseño en el entorno Warp 2.

3.- EJEMPLOS ILUSTRATIVOS.

Los diseños propuestos pretenden hacer un uso progresivo de los recursos de que dispone el lenguaje como son las declaraciones de entidades, arquitecturas, paquetes, procesos, procedimientos, etc.
Supuesto un conocimiento teórico de la gramática del lenguaje (obtenido mediante el complemento de clases teóricas), en un primera sesión se propone diseñar varios elementos básicos como un contador up-down, un sumador completo de 8 bits y un autómata de estados finitos para el control de una alarma de coche, así el alumno comienza a adquirir soltura en el

manejo de las opciones básicas de Warp2 y en la especificación de las estructuras básicas de VHDL.
En una segunda sesión se trata de diseñar dos memorias de pila [3], una de tipo LIFO y otra de tipo FIFO asíncronas, en donde se incluyen los conceptos de procesos, funciones, procedimientos y elementos de entrada de tipo genérico. La figura 2 muestra la entidad que define la estructura LIFO.

```
Entity Pila_LIFO is
      generic( num_bits       :INTEGER;    -- Número de bits por palabra
               num_palabras   :INTEGER);   -- Número de palabras
      port(    PAC            :in BIT;     -- Puesta a cero global
               LC             :in BIT;     -- Instrucción de lectura
               EC             :in BIT;     -- Instrucción de escritura
               DE             :in BIT_VECTOR (1 to num_bits); -- dato in
               DS             :out BIT_VECTOR (2 to num_bits); -- dato sal
               LLENO          :out BIT;    -- Señal de pila llena
               VACIO          :out BIT;    -- Señal de pila vacía
               ERROR          :out BIT);   -- Error de lectura o escritura
end Pila_LIFO
```

Figura 2.- Definición de la pila LIFO.

El comportamiento de la pila se define mediante un proceso (process) que incluye un procedimiento encargado de apilar y desapilar tantos elementos como se especifiquen, y otra parte que permite el control, chequeo y análisis de todas las señales involucradas y se encarga de llamar al procedimiento en el momento apropiado. En la declaración de este proceso se incluyen varios tipos que modelizan la estructura y definen el tamaño de la pila, la longitud de palabra y la dirección (apilar o desapilar), así como variables para el control del proceso. Básicamente, esta arquitectura es sensible a las señales PAC (puesta a cero), LC (lectura) y EC (escritura) y el proceso se encarga de, dependiendo de estas señales, llamar al procedimiento apilar-desapilar con las variables adecuadas, activándose el proceso en el momento en el que algunas de las señales de entrada se activan. Una vez completado el diseño, se propone como trabajo personal la realización de una pila FIFO consistente en una variación sobre la pila LIFO previamente descrita, así, el alumno debe haber entendido la implementación LIFO para ser capaz de construir la pila FIFO.
Como práctica final se propone el diseño de una red neuronal [4] donde es necesaria la creación de una arquitectura básica que luego será instanciada por una descripción de un nivel superior, de esta forma, con la definición de elementos de librería se completa la descripción de los recursos básicos del lenguaje.
Para completar la visión general sobre el lenguaje, esta última práctica incluye la generación de un módulo VHDL para el test de la red, dado que anteriormente las simulaciones se habían basado en NOVA, sin necesidad de hacer uso de esta posibilidad que incluye VHDL.
La red neuronal a implementar es del tipo back-propagation. La neurona que se ha empleado es la descrita por McCulloh y Pitts que toma todas las entradas del sistema y genera una única salida obtenida mediante la suma de cada entrada multiplicada por un coeficiente o "peso", generando una salida a nivel alto/bajo si el resultado es mayor/menor que un determinado umbral. La figura 3 muestra la descripción de una neurona en forma de paquete y su funcionamiento, donde **estímulos** son las entradas y **pesos** son las constantes que se aplican a las entradas, cuya longitud se especifica con una entrada adicional de tipo genérico que se incluye en la entidad.
Para la creación del elemento básico se deben declarar las entidades, que especifican el nombre, dirección (entrada o salida) y tipo de datos de cada una de las señales externas del elemento, la arquitectura, que describe el comportamiento de dicho elemento, y finalmente el

paquete (package), que proporcionará información al compilador para que el elemento pueda ser empleado en descripciones de nivel superior.

```
Package Neurona is
      type vector_estimulos is array      (natural range<>) of real;
      type vector_pesos     is array      (natural range<>) of real;

      function calculo_suma (
               estimulos : vector_estimulos;
               pesos     : vector_pesos)
      return real;
end Neurona

Package body Neurona is
      function calculo_suma(
               estimulos : vector_estimulos;
               pesos     : vector_pesos)
      return real;
      is
      variable suma:real:=0.0;
begin
      for I in estimulos'range loop
        suma:=suma+estimulos(I) * pesos(I);
      end loop;
      return suma;
end calculo_suma;

end Neurona;
```

Figura 3.- Descripción del funcionamiento de la neurona básica.

Una vez finalizada la descripción del elemento básico a emplear, éste debe ser compilado para comprobar si es sintácticamente correcto. Una vez comprobada la sintaxis, se realiza la descripción del sistema completo donde ahora sólo son necesarias las especificaciones de entidad y arquitectura, en este caso no se declara el paquete ya que estamos en el nivel superior.
Finalmente, este módulo se comprueba a través del módulo de test que interconecta varias neuronas capaces de realizar diferentes funciones dependiendo del número de neuronas interconectadas, del umbral definido para cada neurona y del valor de las entradas. En concreto, se han organizdo las neuronas de tres formas diferentes para funcionar de forma sencilla como puertas del tipo OR, AND, y XOR. El módulo de test incluye una declaración de las neuronas y señales de entrada necesarias así como la especificación de los pesos y el número de entradas de que consta cada uno de los módulos.

4.- CONCLUSIONES.

Con este sistema de desarrollo, el alumno consigue una simulación y compilación de las descripciones VHDL donde para su simulación y comprobación se requieren paquetes software destinados a tal efecto, en este caso se ha propuesto Warp 2 debido a su facilidad de manejo en un entorno Windows y su bajo coste.
Cypress distribuye el software a un precio asequible para incentivar el empleo de PLD y CPLD fabricadas por ellos. Su bajo coste (15.000 pesetas aproximadamente) hace que este programa resulte interesante para fines docentes ya que además del compilador VHDL, principal atractivo del paquete, se genera código para programar el dispositivo lógico que se especifique. Una limitación que posee el programa es la de emplear únicamente dispositivos de tipo PLD y CPLD, pues si el diseño excede la capacidad del dispositivo disponible, el fichero

JEDEC necesario para la simulación no es generado. De cualquier forma, resulta un entorno apropiado para la verificación de diseños VHDL de nivel medio.

5.- BIBLIOGRAFÍA.

[1] Cypress Semiconductor: "WARP. VHDL Synthesis Reference". Cypress Semiconductor. USA. 1995.
[2] Cypress Semiconductor: "WARP2. VHDL Compiler for PLD's and CPLD's". Cypress Semiconductor. USA. 1995.
[3] Airiau R., Bergé J.-M., Olive V., Rouillard J.: "VHDL. Du langage a la Modélisation". Presses Polytechniques et Universitaires Romandes. Lausanne. Suisse. 1990.
[4] Lipsett R., Schaffer C.F., Ussery C.: "VHDL: Hardware description and design". Kuwler Academic Publishers. USA. 1991.