# Simulación de Diseños VHDL con Software Libre: La Herramienta GHDL

González-Gómez J.

Escuela Politécnica Superior, Universidad Autónoma de Madrid, España,
{Juan.Gonzalez, Eduardo.Boemo}@ii.uam.es
http://www.eps.uam.es

**Resumen** En este tutorial se explica el funcionamiento de la herramienta *GHDL*, un compilador libre de VHDL basado en el *GCC*. Permite generar directamente programas ejecutables a partir de fuentes en *VHDL*. Al tratarse de *software* libre, se puede utilizar y distribuir sin ninguna restricción. Primero se presentan dos ejemplos muy sencillos, que se compilan con GHDL y se visualizan sus resultados con GTKWAVE. A continuación se muestran cómo trabajar con proyectos más complejos, constituidos por múltiples ficheros y entidades. El compilador genera el fichero *Makefile*, que permite automatizar la compilación mediante el uso de la herramienta *Make*. Por último, se desarrolla un ejemplo de cómo invocar funciones escritas en C, desde código *VHDL*.

## 1 Introducción

Existen aplicaciones comerciales muy potentes para trabajar con *VHDL*, como *ModelSim*[1], de la compañía ModelTech, o *Active-HDL*[2], de Aldec. Las herramientas libres disponibles no son tan completas y no se pueden emplear todavía en diseños profesionales. Sin embargo, están lo suficientemente maduras para su utilización en docencia, donde los diseños no son excesivamente complejos y las simulaciones son cortas. Además, el software libre no tiene restricciones de uso y de distribución, pudiéndose instalar en cualquier número de ordenadores sin tener que pagar licencias.

En este tutorial explicamos cómo simular diseños VHDL utilizando la herramienta GHDL[3], un *front-end* del compilador *GCC*[4] para el lenguaje VHDL, disponible bajo licencia GPL. Para la edición del código fuente se puede emplear cualquier editor de textos ASCII, como por ejemplo *EMACs*[5], que resalta la sintaxis y dispone de un asistente que auto-completa las palabras clave y muestra plantillas de las diferentes construcciones sintácticas del VHDL. Con las aplicaciones *GTKWAVE*[6] o *IVI*[7] se pueden visualizar los resultados de las simulaciones y exportar los diagramas de ondas a formato *postcript*, para incluirlos en las documentaciones.

Primero veremos dos ejemplos sencillos, el clásico “hola mundo” y un inversor. Los compilaremos y mostraremos los resultados de la simulación. Luego nos centraremos en la herramienta GHDL, describiendo todas su opciones y dando un ejemplo de cómo realizar llamadas a funciones en C, desde el código VHDL.

Todas los ejemplos de este tutorial se han probado en una máquina *GNU/Linux* con la distribución *Debian/Sarge*.

# 2 Primeros ejemplos

## 2.1 Programa “hola mundo”

Empezaremos con el programa *hola mundo*, que saca un mensaje por la consola. Se encuentra en el fichero *hola_mundo.vhdl*:

```
use std.textio.all;

entity hola_mundo is
end hola_mundo;

architecture beh of hola_mundo is
begin
  process
    variable l : line;
  begin
    l := new string’("¡¡Hola mundo!!");
    writeline (output, l);
    wait;
  end process;
end beh;
```

Una vez editado, los pasos para simularlo son los siguientes:

```
$ ghdl -a hola_mundo.vhdl
```

Realiza el análisis sintáctico y crea el fichero objeto y la librería de trabajo. Los ficheros que aparecen son:

```
$ ls
hola_mundo.o hola_mundo.vhdl work-obj93.cf
```

Al realizar la elaboración, se genera el ejecutable *hola_mundo*. Hay que especificar el nombre de la entidad superior, que en este caso coincide con el nombre que le hemos dado al fichero:

```
$ ghdl -e hola_mundo
$ ls
e~hola_mundo.o hola_mundo hola_mundo.o hola_mundo.vhdl
work-obj93.cf
```

Finalmente ejecutamos el programa:

```
$ ./hola_mundo
¡¡Hola mundo!!
$
```

## 2.2 Un inversor

El siguiente ejemplo es un inversor y su banco de pruebas. En el fichero *inversor.vhdl* está la entidad *inversor*:

```
library ieee;
use ieee.std_logic_1164.all;

entity inversor is
  port (entrada : in std_logic;
        salida : out std_logic);
end inversor;

architecture beh of inversor is
begin
  salida <= not entrada;
end beh;
```

y el banco de pruebas en el fichero *tb_inversor.vhdl*:

```
library ieee;
use ieee.std_logic_1164.all;

entity tb_inv is
end tb_inv;

architecture beh of tb_inv is
component inversor
  port (entrada : in std_logic;
        salida : out std_logic);
end component;

signal entrada : std_logic:='0';
signal salida : std_logic;

begin
INV: inversor port map (
                   entrada =>entrada,
                   salida =>salida);
-- Generar una senal cuadrada
-- por la entrada del inversor
  process
  begin -- process
    wait for 100 ns;
    entrada<= not entrada;
  end process;
end beh;
```

Los pasos para realizar el análisis y la elaboración son:

```
$ ghdl -a inversor.vhdl tb_inversor.vhdl
$ ghdl -e tb_inv
```

Se genera el fichero *tb_inv* y al ejecutarlo comienza la simulación. Hay que especificar su duración y el archivo en el que se volcarán los resultados (fichero *tb_inv.vcd*):

```
$ ./tb_inv --stop-time=1000ns --vcd=tb_inv.vcd
./tb_inv:info: simulation stopped by --stop-time
```

Para visualizar los resultados usamos el programa *GTKWAVE*:

```
$ gtkwave tb_inv.vcd
```

Este programa nos permite seleccionar qué señales queremos visualizar, establecer el sistema de representación, buscar patrones, etc. Tenemos la opción de volcar los resultados a un fichero *postcript* para incluirlas en las documentaciones (figura1).

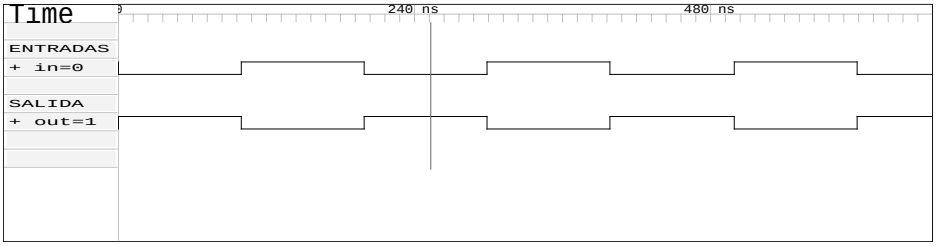


**Figura 1.** Resultado de la simulación del inversor

# 3 Compilador/Simulador GHDL

## 3.1 Características

La herramienta *GHDL* es un compilador/simulador de *VHDL* basado en el *GCC* de *GNU*. No utiliza lenguajes intermedios, sino que traduce directamente a código máquina. Al compilar el banco de pruebas (*testbench*) se genera el ejecutable.

Soporta los estándares *VHDL87* (IEEE 1076-1987), *VHDL93* (IEEE 1076-1993), *VHDL00* (IEEE 1076-2000) e incluye algunas de las revisiones realizadas en el 2002. Se puede seleccionar el estándar mediante el parámetro *–std* en la invocación.

Todavía es una herramienta que se encuentra en estado beta, sin embargo puede compilar correctamente los paquetes *std_logic_1164* y *VITAL*. Por ejemplo, se ha probado simulando un modelo de procesador *DLX*[9] y *LEON1*[8].

Puede realizar anotaciones en ficheros *SDF* (característica experimental, no totalmente soportada). Otra característica importante es la llamada a funciones o procedimientos externos, no definidos en VHDL.

### 3.2 Modos de funcionamiento

- Análisis: *ghdl -a ficheros.vhdl*. Realiza el análisis sintáctico y genera los correspondientes ficheros objetos.
- Elaboración: *ghdl -e entidad*. Realiza la elaboración y genera el fichero ejecutable.
- Directorio: *ghdl -d*. Muestra todas las unidades de diseño que se encuentran en la librería de trabajo: entidades, arquitecturas y configuraciones.
- Importación: *ghdl -i ficheros.vhdl*. Incluye las unidades en la librería de trabajo. No se generan ficheros ejecutables.
- Modo Make: *ghdl -m entidad*. Analiza automáticamente las entidades obsoletas y elabora el diseño.
- Generación de Makefile: *ghdl –gen-makefile entidad*. Construye una fichero *Makefile* con todas las dependencias necesarias para la construcción de la entidad especificada. Luego se genera el ejecutable utilizando la herramienta *make*. Esta opción es muy útil, ya que permite realizar la compilación desde diferentes entornos de trabajo integrados (IDE's) como *Anjuta* o *Emacs*.
- Modo de búsqueda: *ghdl -f ficheros.vhdl*. Indica las unidades de diseño presentes en los ficheros especificados. Junto a las entidades superiores se imprimen dos asteriscos.
- *Clean mode*: *ghdl –clean*. Elimina los ficheros .o y los ejecutables, pero la librería se mantiene.
- *Remove mode*: *ghdl –remove*. Similar al *clean mode* pero eliminando además la librería de trabajo.

### 3.3 Un segundo ejemplo

Con los ficheros del primer ejemplo, utilizaremos los diferentes modos del GHDL:

```
$ ls
hola_mundo.vhdl inversor.vhdl tb_inversor.vhdl
```

Sin necesidad de abrir ningún fichero, podemos conocer qué entidades y arquitecturas están definidas:

```
$ ghdl -f inversor.vhdl
entity inversor
architecture beh of inversor
```

Importamos todas las unidades de diseño en la librería. No tenemos que conocer el diseño:

```
$ ghdl -i *.vhdl
```

Visualizamos todas las unidades que hay en la librería:

```
$ ghdl -d
entity hola_mundo
architecture beh of hola_mundo
entity inversor
architecture beh of inversor
```

```
entity tb_inv
architecture beh of tb_inv
```

En nuestro diseño, sabemos que la entidad superior (*top level*) es *tb_inv*. No obstante, si el diseño no lo hemos hecho nosotros, podemos intentar descubrir cuál es la entidad superior utilizando el modo de búsqueda:

```
$ ghdl -f *.vhdl
entity hola_mundo **
architecture beh of hola_mundo
entity inversor
architecture beh of inversor
entity tb_inv **
architecture beh of tb_inv
```

Las entidades superiores son las que se indican con doble asterisco.

En vez de analizar "a mano" (usando la opción -a), es mejor utilizar el modo *make*, que hace todo lo necesario para elaborar la entidad indicada:

```
$ ghdl -m tb_inv
analyze tb_inversor.vhdl
analyze inversor.vhdl
elaborate tb_inv
```

Se analizan las dependencias, se hace el análisis y finalmente la elaboración. Sin embargo, es mucho más cómodo generar un fichero *Makefile* y trabajar directamente con *make*:

```
$ ghdl --gen-makefile tb_inv >Makefile
```

Ahora, podemos modificar cualquier de los ficheros *.vhdl* con un editor de texto, y al hacer *make*, automáticamente se analizarán sólo los ficheros actualizados y se realizará la elaboración de la nueva entidad:

```
$ make
ghdl -a tb_inversor.vhdl
ghdl -a inversor.vhdl
ghdl -e tb_inv
```

El comando *make* se puede invocar directamente desde entornos de trabajo como *Emacs* o *Anjuta*.

### 3.4 Parámetros de interés

Algunos parámetros de interés son los siguientes:

- *–work=NOMBRE*. Especificar la librería de trabajo.
- *–workdir=PATH*. Especificar la ruta donde se encuentra la librería de trabajo. Por defecto se toma el directorio desde el que se ejecuta *ghdl*.
- *–std=XX*. Indicar el estándar *VHDL* a emplear: 87,93,93c, 00, 02. El 93c es igual que el 93 pero permite las construcciones del 87.

- *–ieee=LIBRERIA*. Establecer la librería a utilizar. Los valores que puede tomar son
  - *standard*: Usar la librería estándar del IEEE (por defecto)
  - *synopsys*: Usar la versión de Synopsys (no recomendado)
  - *mentor*: Usar la versión de Mentor (no recomendado)
  - *none*: No usar ninguna librería del IEEE predefinida

### 3.5 Parámetros de simulación

El ejecutable generado, admite una serie de parámetros. Los más interesantes son:

- *–assert-level=LEVEL*: Indicar el nivel para el que se debe abortar la simulación
- *–stop-time=TIME*. Indicar el tiempo de simulación. Ej. –stop-time=100ns
- *–disp-tree*. Mostrar un árbol de dependencias de las entidades instanciadas
- *–vcd=FICHERO*. Volcar los datos de la simulación a un fichero VCD, para que se puedan ver con un simulador.

### 3.6 Librerías

En diferentes herramientas comerciales, como Mentor o Synopsys, se han añadido a la librería IEEE paquetes que no forma parte del estándar. Un ejemplo son *std_logic_arith* y *std_logic_unsigned*. Es posible utilizarlos con *GHDL*, pero en ese caso hay que especificar que no se está empleando la librería estándar del IEEE, mediante el parámetro *–ieee*. Para evitar problemas de compatibilidad, se recomienda emplear sólo las librerías estándar, como la *numeric_std*.

### 3.7 Invocación desde otros lenguajes

Una característica muy interesante del *GHDL* es la posibilidad de invocar funciones o procedimientos definidos en otros lenguajes, como por ejemplo C o ADA. También se puede llamar a una simulación desde un programa externo.

Esto permite que los bancos de pruebas puedan utilizar funciones no disponibles en VHDL, como por ejemplo funciones matemáticas (senos, cosenos...) o funciones para la generación de números aleatorios.

El proceso para realizarlo lo mostraremos con un ejemplo. En el fichero *holac.c* está definida la función *holac()*, que imprime el mensaje “hola desde C”.

```
#include <stdio.h>
int holac(void)
{
  printf ("Hola desde C...\n");
  return 1;
}
```

El programa *VHDL* es el *hola mundo* modificado. Además de sacar el mensaje, llamará a la función *holac()*. Creamos el paquete *test*, que contiene la definición de la función *holac*. Mediante el atributo *foreign* se indica que es externa. Todo ello está en el fichero *hola_mundo.vhdl*:

```
package test is
  function holac return integer;
  attribute foreign of holac : function is
    "VHPIDIRECT holac";
end test;

package body test is
  function holac return integer is
  begin
    assert false severity failure;
  end holac;
end test;

use std.textio.all;
use work.test.holac;

entity hola_mundo is
end hola_mundo;

architecture beh of hola_mundo is
begin
  process
    variable l : line;
    variable v : integer;
  begin
    l := new string'("Hola desde VHDL...");
    writeline (output, l);
    --Invocar a la funcion de C
    v:=holac;
    wait;
  end process;
end beh;
```

Para la compilación utilizamos el siguiente *Makefile*:

```
GHDL=ghdl
CC=gcc
all: hola_mundo

hola_mundo: test.o holac.o
	$(GHDL) -e -Wl,holac.o hola_mundo

test.o: hola_mundo.vhdl
	$(GHDL) -a $<

clean:
	rm -f hola_mundo *.o *.cf *~
```

A continuación compilamos con *make* y ejecutamos el programa:

```
$ make
ghdl -a hola_mundo.vhdl
gcc -c -o holac.o holac.c
ghdl -e -Wl,holac.o hola_mundo
$ ./hola_mundo
Hola desde VHDL...
Hola desde C...
$
```

## 4 Conclusiones

Hemos presentado una serie de herramientas libres que permiten la simulación de diseños VHDL. De todas ellas, la más importante es el compilador *GHDL*, basado en el *GCC*, que permite crear ficheros ejecutables a partir de las fuentes en *VHDL*. Además, se pueden invocar funciones externas, definidas en otros lenguajes, como C. Esto permite utilizar librerías y funciones que no están implementadas en el estándar de *VHDL*, como por ejemplo la función *random* para obtener números aleatorios.

Las herramientas están lo suficientemente maduras como para utilizarlas en la docencia de VHDL. Al tratarse de software libre, se pueden distribuir sin ninguna restricción e instalarlos en el número de ordenadores que se quiera. Las fuentes están también disponibles, lo que garantiza su portabilidad y su longevidad.

## Agradecimientos

Este trabajo está financiado parcialmente por el Proyecto TIC2001-2688-C03-03 del Ministerio de Ciencia y Tecnología de España, y en parte por el Proyecto 07T/0052/2003-3 de la Consejería de Educación de la Comunidad de Madrid.

## Referencias

1. Herramienta de simulación y depuración *Modelsim*. [En línea]. http://www.model.com/
2. Herramienta *Active-HDL*. [En línea]. http://www.aldec.com/ActiveHDL/
3. Compilador/simulador *GHDL*. [En línea]. http://ghdl.free.fr/
4. Compilador *GCC*: *Gnu C Compiler*. [En línea]. http://gcc.gnu.org/
5. Editor de textos *GNU EMACs*. [En línea]. http://www.gnu.org/software/emacs/emacs.html
6. Visualizador de señales *GTKWAVE*. [En línea]. http://www.cs.man.ac.uk/apt/tools/gtkwave/index.html
7. Herramienta de simulación *IVI*. [En línea]. http://ivi.sourceforge.net/
8. Procesador *Sparc LEON1*. http://www.estec.esa.nl/wsmwww/leon/leon.html
9. Procesador *DLX*. [En línea]. http://www.csee.umbc.edu/courses/undergraduate/411/spring96/dlx.html