# UMHDL
# Manual de instalación

Universidad Miguel Hernández [1]

Julio de 2015

# Índice general

# Capítulo 1

# Descripción

UMHDL es un entorno de desarrollo de sistemas digitales mediante lenguajes de descripción de hardware o *Hardware Description Language* (HDL), específicamente diseñado para su uso docente. Es una aplicación de código abierto creada en la Universidad Miguel Hernandez (UMH).

La motivación de desarrollo de UMHDL fue el de disponer de una aplicacion con interface gráfico o *front-end* que permita el aprendizaje del lenguaje VHDL mediante simulación, que no tenga restricciones de licencias y que requieriese pocos recursos. Este interface permite la escritura del código con diversas ayudas (resalte de sintaxis, ...), invocar al compilador de VHDL y al visualizador de ondas GTKwave.

Actualmente, la aplicación permite utilizar varios **compiladores VHDL**, todos ellos de código abierto: GHDL, FreeHDL, y nvc. Aunque sólo se requiere uno de ellos, el tener la posibilidad de seleccionar otro distinto tiene la ventaja de que se podrá utilizar diferentes estándares o versiones del lenguaje, según nuestras necesidades, o generar diferentes mensajes ante un mismo error, lo cual puede ser de ayuda para identificar las causas.

El usuario también puede elegir el **idioma** de la aplicación (Español, Inglés, Ucraniano, Arabe, Francés, Chino, ...).

UMHDL ha sido desarrollado en el lenguaje de programacion Java, por lo que es **multi-plataforma**; se ejecuta en Windows (32/64-bit), Linux (32/64-bit) y Mac OS X.

Para mas información:

- http://sourceforge.net/projects/umhdl/

En las siguientes secciones se detalla la instalación de la aplicación UMHDL en las distintas plataformas. Más adelante se explica la instalación de cada uno de los compiladores VHDL soportados, caso de ser optar por su instalación.

# Capítulo 2

# Instalación de UMHDL

En este capítulo se describe cómo instalar la aplicación UMHDL en sí; para la instalación de otras herramientas externas, como los compiladores VHDL o GTKwave, consultar el resto de capítulos. En el caso de querer utilizar dichas herramientas externas desde línea de órdenes, el instalador de UMHDL crea un subdirectorio `scripts` con archivos por lotes (.bat) y *scripts* (.sh) para todas las plataformas, y para cada uno de los compiladores VHDL.

## 2.1. UMHDL :: Windows

Para **instalar** UMHDL en Windows simplemente hay que descargar y ejecutar el instalador (p.e. `umhdl-setup-2.0.exe`) y seguir las instrucciones. Se requiere instalar también un compilador de VHDL y el visualizador de ondas GTKWave, pero el instalador para Windows ya incluye ambas cosas: incluye al compilador GHDL y a GTKWave, los cuales se instalarán juntos en un subdirectorio `'Ghdl_gtkwave'` bajo la ruta de instalación de UMHDL. Bajo ese subdirectorio, a su vez, habrá otro subdirectorio `'bin'`, que contiene los ejecutables de ambos, el cual deberá incluirse en la variable de entorno `PATH`. Por ejemplo, si no se cambia el directorio por defecto de instalación de UMHDL, la ruta completa a incluir en la variable `PATH` será la siguiente: `C:\Archivosdeprograma\Umhdl\Ghdl_gtkwave\bin`

Otra opción para no tener que modificar las variables de entorno sería indicar dicha ubicación en las opciones de configuración de UMHDL (menú Opciones > Configurar).

La aplicación intenta primero invocar las órdenes externas sin especificar la ruta, suponiendo que se han especificado en la variable de entorno `PATH`, y en caso de no encontrarlas, entonces lo volverá a intentar con las rutas

especificadas en las opciones de configuración. De este modo se tienen dos posibilidades para configurar las herramientas externas.

Para **desinstalar** la aplicación basta con ejecutar el programa `Uninstall` que se encuentra en la ruta de instalación de la aplicación, o bien, acceder desde su icono en el menú Inicio > Programas > UMHDL > Uninstall. En caso de reinstalar una nueva versión no es necesario desinstalar manualmente, ya que el instalador lo detectara y desinstalará automáticamente.

NOTA: es posible que en algún caso sea necesario instalar GHDL manualmente en caso de obtener cierto error de compilación (detectado en Windows 7), lo cual sería necesario sólo una vez; ver la siguiente sección.

### 2.1.1. Posibles problemas

A continuación se enumera una lista de posibles problemas que pueden ocurrir durante el uso de la aplicación, sobre todo durante su puesta en marcha, y su solución.

#### No se encuentra el compilador VHDL

Si la aplicación no encuentra el compilador GHDL (o cualquiera de los compiladores VHDL soportados) o la aplicación externa GTKWave, hay dos opciones para solucionarlo, comentadas anteriormente:

1. Modificar la variable de entorno `PATH` para que incluya el directorio donde reside el compilador (ver sección 2.1.2). Con esta opción, la ruta especificada puede contener espacios en blanco.

2. Revisar las opciones de configuración de UMHDL (Opciones > Configurar). Primero seleccionaremos el compilador, y en su solapa correspondiente indicaremos en el campo `'bin'` la ruta donde reside el ejecutable en cuestión. Como la ruta escrita se incluirá a la hora de invocar a estos programas externos desde línea de ordenes, en este caso no debe contener espacios en blanco, y hay que utilizar el formato corto de MS-DOS (8 de nombre + 3 de extensión), aunque en este caso sería sólo 8 caracteres del nombre. Es decir, en vez de escribir: `C:\Archivosdeprograma\Umhdl\Ghdl_gtkwave\bin`

   escribiremos la ruta que UMHDL sugiere por defecto en formato corto: `C:\Archiv~1\Umhdl\Ghdl_gtkwave\bin`

   En la ventana de configuración, siempre que se abandona un campo de una ruta se chequea si se encuentra el ejecutable correspondiente (de

GHDL, GtkWave, ...); caso de no encontrarse avisará con un mensaje y la ruta especificada aparecerá de color rojo.

En Windows 8 existen dos directorios `'C:\ProgramFiles'`, uno para aplicaciones de 32 y otro para aplicaciones de 64 bits:

- Aplicaciones de 64-bit: `'C:\ProgramFiles'`, en nombre corto `'C:\Progra~1'`.
- Aplicaciones de 32-bit: `'C:\ProgramFiles(x86)'`, en nombre corto `'C:\Progra2'`.

Puesto que GHDL, GTKWave, ...son programas de 32-bit, se instalarán en este ultimo, y el directorio será: `C:\Progra~2\Umhdl\Ghdl_gtkwave\bin`

Si se hubiera instalado GHDL desde su propio instalador en lugar de utilizar el que integra UMHDL (p.e. `ghdl-installer-0.29.1.exe`), la ruta a especificar será la elegida en esta ocasión, por ejemplo: `C:\Progra~2\Ghdl\bin`

**Error:** `file std_logic_1164.v93 has changed and must be reanalysed`

En ocasiones, al compilar un modulo VHDL en Windows con el compilador GHDL se obtiene el siguiente mensaje:

```
module.vhd:xx:xx: file std_logic_1164.v93 has changed and
                  must be reanalysed
```

Ocurre, por ejemplo, al utilizar por primera vez el tipo `unsigned` del paquete `ieee.numeric_std.all`.

La solución consiste en reconstruir las librerías del compilador GHDL. La aplicación UMHDL proporciona una opción para solucionarlo junto con las opciones de configuración del compilador GHDL: existe un campo donde podemos elegir la ruta de las librerías de GHDL (subdirectorio 'lib') y botón de 'emergencia'. Por ejemplo, si se ha instalado UMHDL en la ruta por defecto la ruta a elegir puede ser: `C:\Progra~1\Umhdl\Ghdk_gtkwave\lib`

Al pulsar el botón de 'emergencia' se recompilarán las librerías (lo cual dura unos segundos), y volveremos a intentar la compilación de nuestro modulo.

**Error:** `package 'numeric_std' is obsoleted by package 'std_logic_1164'`

Al compilar un modulo VHDL en Windows con GHDL puede obtenerse el siguiente mensaje:

```
module.vhd:xx:xx: package "numeric_std" is obsoleted
                  by package "std_logic_1164"
module.vhd:xx:xx: package "numeric_std" is obsolete
Finished with 2 errors! (1883 ms.)
```

Este problema ocurre cuando se utiliza el paquete `'numeric_std'` (para realizar conversiones, etc.). Probaremos primero a quitar la linea que declara el uso de este paquete, pues es posible que no lo estemos utilizando. En caso de necesitarlo, la solución sera reinstalar de forma separada el compilador GHDL. Como se ha indicado anteriormente, el instalador de Windows ya incluye el compilador GHDL, pero si ocurre este problema tendremos que reinstalarlo manualmente con el instalador: `ghdl-installer-0.29.1.exe`. Puesto que se instalará en otra ruta diferente, después tendremos que incluir la nueva ruta en la variable de entorno `PATH`, o modificar las opciones de configuración de la aplicación, tal y como se ha explicado anteriormente.

#### Abortar una simulación

Si se ejecuta una simulación es posible que no termine y se quede un proceso ejecutándose indefinidamente. Puesto que los resultados de la simulación se escriben en un archivo .vcd, es posible que éste crezca sin control, pudiendo agotar por completo el espacio en el disco duro si no se cancela.

UMHDL tiene previsto un tiempo máximo de ejecución de 30 segundos, tras lo cual intenta cancelar automáticamente el proceso, y también, una opción para que el usuario lo aborte de forma manual (botón de 'stop'). Sin embargo, en determinados casos no lo consigue, por lo que tendremos que utilizar el administrador de tareas del sistema para ver los procesos que hay en ejecución, y cancelar aquellos relacionados con UMHDL o con cualquiera de los módulos que se hayan simulado. Después borraremos el archivo .vcd generado en el mismo directorio de cada proyecto que se haya simulado.

Una situación muy común en que ocurre esta circunstancia es cuando el programador no escribe la sentencia `'wait; -- wait forever'` al final del proceso de estímulos de los *test-bench*, prestando especial atención también al proceso de generación de señal de reloj en el caso de sistemas secuenciales, ya que se requiere un proceso separado.

### 2.1.2. Variables de entorno

Para crear o modificar el valor de una variable del sistema de Windows debemos llegar a la ventana del Sistema, que depende de la versión concreta de nuestro sistema operativo. Hasta Windows XP, la forma de llegar era

desde el Panel de Control > Sistema > solapa Avanzado > botón [Variables de entorno].

Una vez hemos llegado a esa ventana, la forma de proceder es idéntica, y tenemos dos posibilidades:

- Variables del *Usuario*: tendremos que volver a reiniciar el programa.
- Variables del *Sistema*: tendremos que reiniciar el ordenador (o la sesión).

En el caso particular de la variable `PATH`, ésta contiene una lista de directorios separados por punto y coma `';'`.

## 2.2. UMHDL :: Linux

En Linux es necesario instalar tanto UMHDL como el compilador de VHDL y GTKwave; no importa el orden que se siga. En este apartado se explica la instalación de UMHDL; para instalar las otras herramientas externas ver los apartados correspondientes.

Para **instalar** UMHDL en Linux seguiremos los siguientes pasos:

1. Descargar y copiar el instalador (`umhdl-setup-xx.yy.bin`) a una ruta que no contenga espacios en blanco.
2. Asegurarse de que el instalador tiene permisos de ejecución:

```
$ chmod +x umhdl-setup-xx.yy.bin
```

3. Ejecutar el instalador. Cuando pida la ruta de instalación, aceptar la que ofrece (`/usr/local/umhdl`) pulsando [Intro], o escribir cualquier otra, pero no utilizar rutas con espacios en blanco o acentos; en caso de instalarlo en un directorio fuera de nuestro `HOME` habrá que ejecutarlo como superusuario o mediante "sudo":

```
$ sudo ./umhdl-setup-xx.yy.bin
Introduzca la ruta de destino[/usr/local/umhdl]: _
...
UMHDL xx.yy se ha instalado correctamente en el sistema.
```

4. Crear un acceso directo en el escritorio. En Ubuntu o Linux Mint sería:
   *a*) Crear un lanzador (click-dcho. sobre el escritorio >Crear lanzador)

*b*) Una vez creado, click-dcho. >Properties >Arrastar el icono (*.gif, *.ico) a la solapa "Basico"(parte superior izquierda)

A continuación procederemos a instalar el compilador VHDL y GTKWave, según se describe más adelante.

Para **desinstalar** el programa podemos hacer:

1. Ir al directorio donde se instaló el programa:

```
$ cd /usr/local/umhdl
```

2. Ejecutar el desinstalador como superusuario:

```
$ sudo Uninstall
Esta seguro de que quiere borrar enteramente UMHDL xx.yy del sistema
   (sí/no)?: sí (con acento)
Todos los componentes de UMHDL xx.yy han sido borrados correctamente
   del sistema.
```

### 2.2.1. Posibles problemas

A continuación se enumera una lista de posibles problemas que pueden ocurrir durante el uso de la aplicación en Linux, y su solución.

#### Linux 64-bit no permite la ejecución de aplicaciones 32-bit

Si nuestro sistema operativo Linux es de 64-bit y no se abre la aplicación, ejecutarlo desde la consola (no desde el icono) para ver si muestra algún error. Puesto que tanto UMHDL como las herramientas externas son aplicaciones de 32-bit, es muy posible que se obtenga el siguiente error, indicando que tendremos que instalar las librerías para dar soporte a la ejecución de aplicaciones de 32-bit (*multiarch*):

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:
             exception occurred in JNI_OnLoad
```

En este caso será necesario instalar el paquete `ia32-libs` y cualquier otro del que dependa (pueden ser bastantes):

```
$ sudo apt-get install ia32-libs
```

### 2.2.2. Variables de entorno

Tras la instalación de cualquier herramienta externa (GHDL, GTKWave, ...) debemos asegurarnos de que las rutas de los ejecutables estén incluidas en la variable de entorno `PATH`. Por defecto, en Linux las rutas son:

- ghdl: `/usr/local/bin`
- gtkwave: `/usr/bin`

Otra opción más sencilla (para eso está) es configurar correctamente las rutas de todas las utilidades externas en las opciones de configuración de la aplicación. En cualquier caso, podemos encontrar la ubicación de los ejecutables con las órdenes `'which'` o `'whereis'`:

```
$ which ghdl
/usr/local/bin/ghdl
$ which gtkwave
/usr/bin/gtkwave
```

Normalmente, las rutas anteriores ya están incluidas en la variable `PATH`. No obstante, si se necesita modificar alguna variable de entorno del sistema se procedería del siguiente modo, dependiendo del tipo de *shell*:

- *Shell* de tipo: bash, ksh, zsh o sh: se trata de modificar el fichero `.bash_profile` o `.profile`, escribiendo lo siguiente:

```
export PATH=$PATH:.:/usr/local/bin:/usr/bin
```

- *Shell* de tipo csh or tcsh: el fichero a modificar es `.login`, y escribiremos:

```
setenv PATH PATH=$PATH:.:/usr/local/bin:/usr/bin
```

## 2.3. UMHDL :: Mac OS X

En MacOS X es necesario instalar tanto UMHDL como el compilador GHDL y GTKwave.

Para instalar UMHDL simplemente hay que descomprimir el archivo `'umhdl-mac-app-xx.xx.zip'` en algún lugar, como por ejemplo, la carpeta /Aplicaciones. Contiene únicamente un archivo .app que podemos copiar a la carpeta /Aplicaciones, poner en el Dock, o crear un alias para tener un acceso directo en el escritorio. Este archivo .app ya debería tener un **icono**

asociado pero, debido a que el compilador e instalador ExcelsiorJet aun no está totalmente terminado, hay tareas que deben hacerse de forma manual, como la de asociar el icono a la aplicación; en ese caso de que no lo tenga, se le puede asignar posteriormente en sus propiedades (click-dcho. y elegir Obtener Información, y finalmente arrastrar el icono en formato .icns a la parte superior.

Respecto a la instalación de las herramientas externas necesarias como GHDL y GTKWave, comentar que hay una serie de requisitos previos que debemos comprobar, básicamente para poder instalarlas mediante el gestor de paquetes MacPorts. De ello trata el siguiente apartado, que en caso de ya tenerlo instalado podemos ignorar y saltar directamente a los apartados correspondientes a las herramientas concretas a instalar.

### 2.3.1. Requisitos previos: X11, Xcode y MacPorts

Puesto que para instalar las herramientas externas se recomienda algún gestor de paquetes como MacPorts o Fink, y éstos, a su vez, requieren tener instalados ciertos requisitos, comentamos su instalación. Todo lo que aquí se explica es necesario para la instalación de cualquier programa utilizando MacPorts, por lo que es muy probable que ya se tenga instalado; en ese caso, se puede saltar al siguiente paso, ejecutando, si fuese necesario, una actualización de los paquetes (los dos últimos pasos).

1. Instalar un **servidor X11**: Xcode y GTKWave requieren que se tenga instalado un servidor X11. En versiones antiguas de Mac OS X (10.8 Mountain Lion, 10.7 Lion, 10.6 Snow Leopard, 10.5 Leopard) ya venía instalado en el sistema el servidor X11, o podía instalarse desde el propio disco de instalación (Optional Installs > X11); en versiones actuales (Mavericks 10.9, Yosemite 10.10, ...) ya no se incluye X11 de serie, por lo que hay que instalarlo manualmente. Existe un servidor X11 llamado XQuartz, que se puede descargar para OS X 10.6 o posteriores desde: `http://xquartz.macosforge.org`. Una vez instalado es necesario reiniciar para que se active como servidor X11 por defecto.

2. Instalar **Xcode**. Dependiendo de la versión de OS X y de Xcode, puede ser necesario también instalar de forma explícita las *Developer Command Line Tools*. Dependiendo de ello, se pueden seguir las instrucciones del enlace https://developer.apple.com/technologies/tools/, o también, http://guide.macports.org/#installing.xcode.

3. Instalar **MacPorts**: Seguir las instrucciones que se muestran en la página oficial https://www.macports.org/install.php, en donde se ofrecen

diversas alternativas (instalador .pkg, código fuente, ...); conviene instalar desde el paquete instalador .pkg, ya que se encarga de establecer las variables de entorno apropiadas y se actualiza automáticamente.

4. Actualizar MacPorts (si hace mas de dos semanas desde la última vez):

```
$ sudo port selfupdate --nosync
$ sudo port selfupdate
```

5. Actualizar programas previamente instalados:

```
$ sudo port outdated
$ sudo port selfupdate
$ sudo port upgrade outdated
```

Una vez instalado MacPorts ya será posible instalar las herramientas externas de una forma más sencilla. Cualquier programa instalado con MacPorts se instalará en la ruta `/opt/local/bin`, y los archivos se descargan previamente a al directorio `/opt/local/var/macports/distfiles` (aunque esto no es necesario conocerlo).

### 2.3.2. Variables de entorno

Tras la instalación de cualquier herramienta externa (GHDL, GTKWave, ...) debemos asegurarnos de que las rutas de los ejecutables estén incluidas en la variable de entorno `PATH`. Por defecto, las rutas son:

- ghdl: `/usr/local/bin`
- gtkwave: `/opt/local/bin`

Para añadir alguna de estas rutas a la variable de entorno `PATH` debemos utilizar el fichero `'/etc/launchd.conf'`, ya que si se incluye en el archivo `'.bash_profile'`, podría invocarse a `'ghdl'` o `'gtkwave'` desde el Terminal, pero no funcionará desde una aplicación gráfica, como sería el caso de UMHDL:

```
$ sudo vi /etc/launchd.conf
setenv JAVA_HOME /Library/Java/Home
setenv PATH /usr/local/bin
```

Como en este archivo no se puede hacer referencia a otras variables de entorno (p.e. `$PATH`), la solución es ejecutar `'launchctl'`:

```
$ launchctl setenv PATH /usr/local/bin:/opt/local/bin
```

Para automatizar esto puede añadirse la siguiente línea al final del archivo `'.bash_profile'`:

```
$ vi $HOME/.bash_profile
...
grep -E "^setenv" /etc/launchd.conf | xargs -t -L 1 launchctl
```

Otra opción más sencilla (para eso está) es configurar correctamente las rutas de todas las utilidades externas en las opciones de configuración de la aplicación. En cualquier caso, podemos encontrar la ubicación de los ejecutables con:

```
$ which ghdl
/usr/local/bin/ghdl
$ which gtkwave
/opt/local/bin/gtkwave
```

# Capítulo 3

# GHDL

GHDL es un compilador/simulador del lenguaje VHDL que se ejecuta desde línea de órdenes. Es el compilador VHDL más recomendado actualmente por su sencillez de instalación, tanto en Windows, como Linux, y Mac OS X.

Algunos **enlaces**:

- http://gna.org/projects/ghdl/
- http://ghdl.free.fr/
- http://sourceforge.net/projects/ghdl-updates/
- http://eng-osx.sourceforge.net/GHDL.html

## 3.1. Windows

El instalador de UMHDL ya incluye tanto el compilador GHDL como a GTKWave, instalándose en un subdirectorio llamado `'Ghdl_gtkwave'`, cuyo subdirectorio `'bin'` es el que debemos añadir a la variable de entorno `PATH` o establecer en las opciones de UMHDL.

Por tanto, al instalar UMHDL en Windows ya es necesario instalar nada más. No obstante, en algún caso puede ser necesario volver a instalar GHDL de forma manual. Para ello, puede descargarse un instalador: ghdl-installer-0.29.1.exe. Simplemente hay que ejecutarlo y seguir instrucciones.

## 3.2. Linux

Tenemos dos opciones para instalar el compilador GHDL:

1. Descargar e instalar mediante algún gestor de paquetes, recomendado porque se comprueban si existen dependencias de otros paquetes. Basta con ejecutar la siguiente orden desde la consola, que instala GHDL en la ruta `/usr/local/bin`:

```
$ sudo apt-get install zlib1g-dev lib32z1-dev ghdl
```

   Es necesario instalar la librería `'lib32z1-dev'` para evitar el siguiente error al elaborar un ejecutable con `'ghdl -e'` (ver abajo):

   `/usr/bin/ld: cannot find -lz`

2. Descargar los binarios y desempaquetar directamente el archivo:

```
$ wget http://ghdl.free.fr/site/uploads/Main/ghdl-i686-linux-latest.
    tar
$ sudo tar xvf ghdl-i686-linux-latest.tar
  (esto genera ghdl-0.29-i686-pc-linux.tar.bz2)
$ cd ghdl-0.29-i686-pc-linux
$ sudo tar -C / -jxvf ghdl-0.29-i686-pc-linux.tar.bz2
  (esto copia archivos en /usr/local/bin y /usr/local/lib)
```

En cualquier caso, asegurarse de que la ruta del compilador (p.e. `/usr/local/bin`) esta en el `PATH`.

### 3.2.1. Posibles problemas

La siguiente son una serie de problemas encontrados al ejecutar GHDL desde línea de órdenes, los cuales no se incluyen en la sección 2.2.1 porque la propia aplicación UMHDL ya lo tiene en cuenta, detectando si la aplicación se ejecuta en un sistema operativo de 32 o de 64-bit para añadir las opciones pertinentes. No obstante, se listan aquí para el caso de querer utilizar el compilador GHDL desde línea de órdenes de forma manual.

Además de estos problemas, también podrían ocurrir los enumerados para Windows en la sección 2.1.1.

### 3.2.2. Error en Linux 64-bit: `push/pop`

Si se obtienen los siguientes errores al compilar, el problema es que GHDL es una aplicación de 32-bit:

```
/home/myworkspace/e~and_gate.s: Assembler messages:
/home/myworkspace/e~and_gate.s:40: Error: invalid instruction suffix for `
    push'
```

```
/home/myworkspace/e~and_gate.s:42: Error: invalid instruction suffix for `
    pop'
```

La solución es añadir ciertas opciones extra al invocar al compilador GHDL:

```
$ ghdl -a -Wa,--32 ...
$ ghdl -e -Wa,--32 -Wl,-m32 ...
```

Como ya se ha comentado, esto ya lo tiene en cuenta la aplicación UMHDL, por lo que no debería ocurrir.

### 3.2.3. Error en Linux 64-bit: `incompatible libgcc.a`

Podemos obtener los siguientes errores al compilar:

```
/usr/bin/ld: skipping incompatible
        /usr/lib/gcc/x86_64-linux-gnu/4.6/libgcc.a
        when searching for -lgcc
```

En este caso es necesario instalar el paquete gcc-4.5-multilib y otros de los que dependa (12):

```
$ sudo apt-get install gcc-4.5-multilib
```

### 3.2.4. Error: `cannot find -lz`

Podemos obtener los siguientes errores al elaborar con `'ghdl -e'`:

```
/usr/bin/ld: cannot find -lz
```

Como se ha comentado anteriormente, en este caso hay que instalar algunos paquetes: zlib1g-dev (zlib1g-dbg), lib32z1-dev (+lib32z1):

```
$ sudo apt-get install lib32z1-dev
```

## 3.3. Mac OS X

Existen paquetes de GHDL ya compilados, los cuales dependen de la versión de nuestro sistema operativo:

- Para OS X 10.9 (Mavericks) o superior, descargar el archivo ghdl-0.31-mcode-darwin13.mpkg.zip.

- Para OS X 10.8 o anteriores, descargar el archivo ghdl_mcode_r142.dmg.

Simplemente hay que ejecutarlo y seguir instrucciones. El programa se instalará en el directorio `/usr/local/bin`

# Capítulo 4

# FreeHDL

FreeHDL es un compilador/simulador de VHDL, pensando principalmente para el sistema operativo Linux. Existen paquetes que se supone debería poder ejecutarse también para Windows y Mac OS X, aunque de momento no ha sido posible su instalación. No obstante, en las siguientes secciones se indican los pasos llevados a cabo; si el lector consigue su instalación en estos sistemas operativos se agradecería me escribiese, ¡gracias!

Algunos enlaces:

- http://freehdl.seul.org/
- https://sourceforge.net/projects/qucs/files/freehdl/
- http://www.dettus.net/tutorials/freehdl/

## 4.1. FreeHDL :: Windows (TO-DO)

Aunque no se conseguido con éxito, a continuación se indican los pasos llevados a cabo para la instalación de FreeHDL en Windows.

### 4.1.1. Requisitos previos: Perl, Cygwin/MinGW

FreeHDL requiere tener instalado Cygwin o MinGW; ambos son un conjunto de herramientas de desarrollo de GNU para Windows (gcc, etc.). Además, es necesario instalar también un intérprete de lenguaje Perl para poder invocar al `script gvhdl`.

### Intérprete de Perl

Comenzando por lo sencillo, vamos a instalar un intérprete de Perl. Existen varias opciones. Por elegir uno, instalamos ActivePERL, que simplemente tenemos que descargar y ejecutar un instalador.

### Cygwin/MinGW

En la página Web de Qucs, que integra a FreeHDL y que parece que lo han portado a Windows, se dan varias posibilidades para su instalación, entre ellas, mediante Cygwin o mediante MinGW. En este punto aconsejo el uso de Cygwin por diversos motivos:

1. Puesto que el compilador nvc requiere Cygwin, parece redundante instalar ambos.

2. Por el hecho de tener instalado MinGW tuve problemas para hacer funcionar el compilador nvc; de hecho hasta que no lo desinstalé no fue posible ejecutarlo correctamente.

3. Los paquetes

Para la instalación de **Cygwin** (recomendado) puedes seguir los pasos explicados en el apartado 5.1.1.

Si optas por **MinGW**, puedes descargarlo desde los siguientes enlaces:

- http://www.mingw.org
- http://sourceforge.net/projects/mingw
- http://sourceforge.net/projects/mingw-w64

En la misma página de Qucs existe un instalador que dicen que tiene lo mínimo necesario para que funcione FreeHDL. Pues no, al intentar ejecutar FreeHDL tras su instalación (ver más adelante) le faltan dos archivos DLL: `libgcc_s_dw2-1.dll` y `libwinpthread-1.dll`, los cuales que deben copiarse al directorio: `C:\FreeHDL\bin` (suponiendo que el directorio de instalación de FreeHDL sea ese). Para conseguir estos 2 archivos (cosa que lleva su rato averiguar), puedes descargar el paquete MinGW completo desde SourceForge, concretamente el instalador mingw-w64-install.exe, concretamente desde Files / Toolchains targetting Win32 / Personal Builds / mingw-builds / installer; no descargues esos archivos desde cualquier sitio.

En cualquier caso, elegir una ruta sin espacios en blanco (p.e. `C:\cygwin` ó `C:\MinGW`).

### 4.1.2. Instalación de FreeHDL

Una vez instalado el intérprete de Perl y Cygwin (o MinGW), procedemos a instalar FreeHDL simplemente ejecutando el instalador `freehdl-0.0.8-setup.exe`, `freehdl-0.0.7-setup.exe`, ..., que podremos descargar desde la página de Qucs, o directamente desde el siguiente enlace: http://sourceforge.net/projects/qucs/files/freehdl/.

Como ruta de instalación elegir una sin espacios en blanco, como `C:\FreeHDL` (no la ruta sugerida `C:\ProgramFiles\FreeHDL`). El subdirectorio `C:\FreeHDL\bin` deberá estar incluido en la variable PATH. Esa ruta contiene varios *script* en Perl, por lo que hemos tenido que instalar un intérprete de Perl anteriormente.

Automáticamente se crea una variable de entorno llamada `FREEHDL`, con el valor `C:\FreeHDL`. Puesto que este valor se utiliza el *script* `vghdl`, y hace referencia a las rutas de FreeHDL en formato Linux, modificaremos su valor a `C:/FreeHDL`.

Seguramente obtendremos los tres **errores** siguientes:

```
C:\FreeHDL\bin>perl gvhdl
gvhdl: FreeHDL root path is 'C:\FreeHDL'.
gvhdl:
gvhdl: ================================
gvhdl: Compiling simulator main file '.cc'...
gvhdl: ================================
gvhdl: wine g++.exe  -I C:\FreeHDL/include -c .cc
c++: 'wine' is not recognized as an internal or external command,
c++: operable program or batch file.
gvhdl: Compilation failed!
Died at gvhdl line 256.

...

linker: 'libtool' is not recognized as an internal or external command,
linker: operable program or batch file.
gvhdl: Linking failed!
Died at C:\FreeHDL\bin\gvhdl line 276.

...

linker: libtool: link: unable to infer tagged configuration
linker: libtool:   error: specify a tag with '--tag'
gvhdl: Linking failed!
```

En ese caso tendremos que modificar el *script* `C:\FreeHDL\bin\gvhdl` cambiar las siguientes líneas:

```
my $cc = "wine g++.exe";
my $libtool = "glibtool";
my $libtool_options = "--mode=link";
```

por:

```
my $cc = "g++.exe";
my $libtool = "bash.exe libtool";
my $libtool_options = "--mode=link --tag=CXX";
```

Además, si estamos trabajando con Cygwin, ejecutaremos de nuevo el ejecutable `setup-xxx.exe` para comprobar si hemos instalado `'g++'` y `'libtool'` de la categoría `'Devel'`. También debemos comprobar si la ruta `C:\cygwin\bin` se incluye en la variable de entorno `PATH`.

Intentando compilar un primer ejemplo, tras los cambios anteriores se siguen obteniendo unos errores al enlazar:

```
linker: C:/FreeHDL/lib/libfreehdl-kernel.a(libfreehdl_kernel_la-handle.o):
(...): undefined reference to `_assert'
...
linker: C:/FreeHDL/lib/libfreehdl-std.a(vhdl_types.o):vhdl_types.cc:
(.text+0x29b2): undefined reference to `_imp___stricmp'
linker: collect2: error: ld returned 1 exit status
```

Las funciones `assert()` y `imp___stricmp()` no las encuentra. No sé en qué paquete exactamente se incluyen, pero ejecuto el instalador de Cygwin `setup-xxx.exe` y selecciono para instalar todos los paquetes de la categoría `'Devel'` por completo. Tras varias horas, ... idem.

De momento nos quedamos sin haber probado FreeHDL en Windows. Si lo consigues, escríbeme un correo. Gracias!

## 4.2. FreeHDL :: Linux

La instalación de FreeHDL en Linux es muy sencilla, requiriendo únicamente la instalación del paquete 'freehdl' con algún gestor de paquetes (Synaptic, apt-get, yum, ...).

Para utilizar FreeHDL se invoca al ejecutable `freehdl-v2cc`, pero existe un script escrito en Perl que facilita la tarea de la compilación llamado `/usr/bin/gvhdl`. Dicho script generará un ejecutable con el mismo nombre que el correspondiente al testbench, que será el que tendremos que ejecutar directamente para ejecutar la simulación.

Como veremos en el siguiente ejemplo de compilación y simulación de un proyecto, será necesario hacer dos ligeras modificaciones en los siguientes *scripts* o archivos tal y como se muestra a continuación:

- `/usr/bin/gvhdl`: añadiendo la opción `'-tag=CXX'` a la variable `'libtool_options'`.
- `/usr/bin/freehdl-config`: estableciendo correctamente la ruta de la variable `'libdir'`.

NOTA: comentar que FreeHDL parece que requiere que la extensión de los archivos sea .vhdl en lugar de .vhd (ver la sección de posibles problemas 4.2.3).

En esta sección tuve la colaboración del alumno Daniel Valero Carreras.

### 4.2.1. Ejemplo

Vamos a suponer que tenemos un proyecto llamado `'buffer_amplifier'` con los archivos `buffer_amplifier.vhdl` y `buffer_amplifier_tb.vhdl`. Como se ha comentado, se compilaría utilizando el script `gvhdl`, y se generará un ejecutable con el nombre `buffer_amplifier`.

```
$ gvhdl buffer_amplifier.vhdl
gvhdl: FreeHDL root path is '/usr'.
gvhdl: executing '/usr/bin/freehdl-v2cc -m buffer_amplifier._main_.cc -L /
    usr/share/freehdl/lib   -o buffer_amplifier.cc buffer_amplifier.vhdl'
gvhdl:
gvhdl: ================================
gvhdl: Compiling 'buffer_amplifier.cc'...
gvhdl: ================================
gvhdl: x86_64-linux-gnu-g++  -I /usr/include -c buffer_amplifier.cc
gvhdl:
gvhdl: ================================
gvhdl: Compiling simulator main file 'buffer_amplifier._main_.cc'...
gvhdl: ================================
gvhdl: x86_64-linux-gnu-g++  -I /usr/include -c buffer_amplifier._main_.cc
gvhdl: Linking simulator 'buffer_amplifier'...
gvhdl: libtool --mode=link x86_64-linux-gnu-g++  buffer_amplifier._main_.o
    buffer_amplifier.o -lm /usr/lib/libfreehdl-kernel.la /usr/lib/libfreehdl
    -std.la -o buffer_amplifier
linker: libtool: link: unable to infer tagged configuration
linker: libtool: link: specify a tag with `--tag'
gvhdl: Linking failed!
Died at /usr/bin/gvhdl line 276.
```

Como vemos, la compilación aborta, y según se informa en este enlace, es necesario modificar el script `/usr/bin/gvhdl`, cambiando la línea:

```
    my $libtool_options = "--mode=link";
```

por:

```
    my $libtool_options = "--mode=link --tag=CXX";
```

Por otro lado, en el enlace anterior también se comenta que puede ser necesario modificar el archivo `/usr/bin/freehdl-config`. Hay una línea que ejecuta para obtener la ruta de las librerías y establecer la variable `'libdir'`, cuyo valor es `/usr/lib` y debería ser `/usr/lib64`. Pero en mi caso (Linux Mint 64) la ruta que se obtiene en esa línea es justamente `/usr/lib`, como se

comprueba ejecutando dicha línea desde el terminal, por lo que, en mi caso particular no fue necesario realizar dicho cambio:

```
libdir=‘pkg-config --variable libdir freehdl‘
$ pkg-config --variable libdir freehdl
usr/lib
```

Probamos de nuevo:

```
$ gvhdl buffer_amplifier.vhdl
gvhdl: FreeHDL root path is ’/usr’.
gvhdl: executing ’/usr/bin/freehdl-v2cc -m buffer_amplifier._main_.cc -L /
    usr/share/freehdl/lib   -o buffer_amplifier.cc buffer_amplifier.vhdl’
gvhdl:
gvhdl: ================================
gvhdl: Compiling ’buffer_amplifier.cc’...
gvhdl: ================================
gvhdl: x86_64-linux-gnu-g++  -I /usr/include -c buffer_amplifier.cc
gvhdl:
gvhdl: ================================
gvhdl: Compiling simulator main file ’buffer_amplifier._main_.cc’...
gvhdl: ================================
gvhdl: x86_64-linux-gnu-g++  -I /usr/include -c buffer_amplifier._main_.cc
gvhdl: Linking simulator ’buffer_amplifier’...
gvhdl: libtool --mode=link --tag=CXX x86_64-linux-gnu-g++  buffer_amplifier.
    _main_.o  buffer_amplifier.o -lm /usr/lib/libfreehdl-kernel.la /usr/lib/
    libfreehdl-std.la -o buffer_amplifier
linker: libtool: link: x86_64-linux-gnu-g++ buffer_amplifier._main_.o
    buffer_amplifier.o -o buffer_amplifier  -lm /usr/lib/libfreehdl-kernel.
    so /usr/lib/libfreehdl-std.so
linker: buffer_amplifier.o: In function ‘L4work_E16buffer_amplifier::
    L4work_E16buffer_amplifier(name_stack&, map_list*, void*)’:
linker: buffer_amplifier.cc:(.text+0x99): undefined reference to ‘
    L4ieee_Q14std_logic_1164_I9std_logic_INFO’
linker: buffer_amplifier.cc:(.text+0xe4): undefined reference to ‘
    L4ieee_Q14std_logic_1164_I9std_logic_INFO’
linker: buffer_amplifier.o: In function ‘L4work_E16buffer_amplifier_init()’:
linker: buffer_amplifier.cc:(.text+0x19f): undefined reference to ‘
    L4ieee_Q11numeric_std_init()’
linker: buffer_amplifier.cc:(.text+0x1a4): undefined reference to ‘
    L4ieee_Q14std_logic_1164_init()’
linker: collect2: ld returned 1 exit status
gvhdl: Linking failed!
Died at /usr/bin/gvhdl line 277.
```

Ahora vuelve a abortar por otro motivo, y es que para utilizar el tipo `'std_logic'` hay que especificar que se utilizan las librerías `'ieee'`, simplemente añadiendo la opción `'-libieee'` a la hora de invocar a `gvhdl`:

```
$ gvhdl --libieee buffer_amplifier.vhdl
gvhdl: FreeHDL root path is ’/usr’.
gvhdl: executing ’/usr/bin/freehdl-v2cc -m buffer_amplifier._main_.cc -L /
    usr/share/freehdl/lib   -o buffer_amplifier.cc buffer_amplifier.vhdl’
gvhdl:
gvhdl: ================================
gvhdl: Compiling ’buffer_amplifier.cc’...
gvhdl: ================================
```

```
gvhdl: x86_64-linux-gnu-g++  -I /usr/include -c buffer_amplifier.cc
gvhdl:
gvhdl: ================================
gvhdl: Compiling simulator main file 'buffer_amplifier._main_.cc'...
gvhdl: ================================
gvhdl: x86_64-linux-gnu-g++  -I /usr/include -c buffer_amplifier._main_.cc
gvhdl: Linking simulator 'buffer_amplifier'...
gvhdl: libtool --mode=link --tag=CXX x86_64-linux-gnu-g++  buffer_amplifier.
    _main_.o  buffer_amplifier.o -lm /usr/lib/libfreehdl-kernel.la /usr/lib/
    libfreehdl-std.la /usr/lib/freehdl/libieee.la -o buffer_amplifier
linker: libtool: link: x86_64-linux-gnu-g++ buffer_amplifier._main_.o
    buffer_amplifier.o -o buffer_amplifier  -lm /usr/lib/libfreehdl-kernel.
    so /usr/lib/libfreehdl-std.so /usr/lib/freehdl/libieee.so -Wl,-rpath -Wl
    ,/usr/lib/freehdl -Wl,-rpath -Wl,/usr/lib/freehdl
gvhdl: ================================
gvhdl: Simulator 'buffer_amplifier' created.
gvhdl: ================================

$ gvhdl --libieee buffer_amplifier_tb.vhdl
gvhdl: FreeHDL root path is '/usr'.
gvhdl: executing '/usr/bin/freehdl-v2cc -m buffer_amplifier_tb._main_.cc -L
    /usr/share/freehdl/lib   -o buffer_amplifier_tb.cc buffer_amplifier_tb.
    vhdl'
gvhdl:
gvhdl: ================================
gvhdl: Compiling 'buffer_amplifier_tb.cc'...
gvhdl: ================================
gvhdl: x86_64-linux-gnu-g++  -I /usr/include -c buffer_amplifier_tb.cc
gvhdl:
gvhdl: ================================
gvhdl: Compiling simulator main file 'buffer_amplifier_tb._main_.cc'...
gvhdl: ================================
gvhdl: x86_64-linux-gnu-g++  -I /usr/include -c buffer_amplifier_tb._main_.
    cc
gvhdl: Linking simulator 'buffer_amplifier_tb'...
gvhdl: libtool --mode=link --tag=CXX x86_64-linux-gnu-g++
    buffer_amplifier_tb._main_.o  buffer_amplifier_tb.o -lm /usr/lib/
    libfreehdl-kernel.la /usr/lib/libfreehdl-std.la /usr/lib/freehdl/libieee
    .la -o buffer_amplifier_tb
linker: libtool: link: x86_64-linux-gnu-g++ buffer_amplifier_tb._main_.o
    buffer_amplifier_tb.o -o buffer_amplifier_tb  -lm /usr/lib/libfreehdl-
    kernel.so /usr/lib/libfreehdl-std.so /usr/lib/freehdl/libieee.so -Wl,-
    rpath -Wl,/usr/lib/freehdl -Wl,-rpath -Wl,/usr/lib/freehdl
gvhdl: ================================
gvhdl: Simulator 'buffer_amplifier_tb' created.
gvhdl: ================================
```

Vemos que ahora sí hemos podido compilar correctamente tanto el módulo principal como el correspondiente al *testbench*. En realidad, como tenemos varios módulos (dos) no es necesario enlazarlos todos, tan sólo el que se ejecutará, que es el *testbench*. Por ello, compilaremos el módulo principal con la opción `-c` (para que no genere ningún ejecutable), y el módulo del *testbench* sí que se enlazará a los anteriores, siendo éste el único ejecutable a generar:

```
$ gvhdl -c --libieee buffer_amplifier_tb.vhdl
gvhdl: FreeHDL root path is '/usr'.
```

```
gvhdl: executing '/usr/bin/freehdl-v2cc -L /usr/share/freehdl/lib   -o
    buffer_amplifier_tb.cc buffer_amplifier_tb.vhdl'
gvhdl:
gvhdl: ================================
gvhdl: Compiling 'buffer_amplifier_tb.cc'...
gvhdl: ================================
gvhdl: x86_64-linux-gnu-g++  -c -I /usr/include -c buffer_amplifier_tb.cc

$ gvhdl --libieee buffer_amplifier_tb.vhdl buffer_amplifier.o
gvhdl: FreeHDL root path is '/usr'.
gvhdl: executing '/usr/bin/freehdl-v2cc -m buffer_amplifier_tb._main_.cc -L
    /usr/share/freehdl/lib   -o buffer_amplifier_tb.cc buffer_amplifier_tb.
    vhdl'
gvhdl:
gvhdl: ================================
gvhdl: Compiling 'buffer_amplifier_tb.cc'...
gvhdl: ================================
gvhdl: x86_64-linux-gnu-g++  -I /usr/include -c buffer_amplifier_tb.cc
gvhdl:
gvhdl: ================================
gvhdl: Compiling simulator main file 'buffer_amplifier_tb._main_.cc'...
gvhdl: ================================
gvhdl: x86_64-linux-gnu-g++  -I /usr/include -c buffer_amplifier_tb._main_.
    cc
gvhdl: Linking simulator 'buffer_amplifier_tb'...
gvhdl: libtool --mode=link --tag=CXX x86_64-linux-gnu-g++
    buffer_amplifier_tb._main_.o  buffer_amplifier_tb.o buffer_amplifier.o -
    lm /usr/lib/libfreehdl-kernel.la /usr/lib/libfreehdl-std.la /usr/lib/
    freehdl/libieee.la -o buffer_amplifier_tb
linker: libtool: link: x86_64-linux-gnu-g++ buffer_amplifier_tb._main_.o
    buffer_amplifier_tb.o buffer_amplifier.o -o buffer_amplifier_tb  -lm /
    usr/lib/libfreehdl-kernel.so /usr/lib/libfreehdl-std.so /usr/lib/freehdl
    /libieee.so -Wl,-rpath -Wl,/usr/lib/freehdl -Wl,-rpath -Wl,/usr/lib/
    freehdl
gvhdl: ================================
gvhdl: Simulator 'buffer_amplifier_tb' created.
gvhdl: ================================
```

Sin la opción `-c` se generan los archivos: `buffer_amplifier.cc` y `buffer_amplifier.o`, `buffer_amplifier._main_.cc`, `buffer_amplifier._main_.o`, y el ejecutable `buffer_amplifier`, que no se utilizará para nada. Por otro lado, especificando la opción -c se generan únicamente los dos primeros archivos: `buffer_amplifier.cc` y `buffer_amplifier.o` (ningún ejecutable ni archivos intermedios).

Finalmente, para realizar la simulación se ejecuta el ejecutable generado, con una serie de opciones que conviene consultar en el manual ejecutando `'man gvhdl'` o en este enlace. Si todo ha ido bien, se habrá generado el archivo `buffer_amplifier_tb.vcd`, que podremos visualizar mediante GTKwave de la forma habitual:

```
$ buffer_amplifier_tb -cmd "dc -f buffer_amplifier_tb.vcd; d; run 1000 ns; q
    ;"
$ gtkwave buffer_amplifier_tb.vcd
```

En la ventana de GTKWave marcamos arriba üutz nos aparecen las señales abajo. Las marcamos y pulsamos [Append] o [Insert] para que aparezcan en el diagrama de ondas. Finalmente ajustaremos el zoom.

### 4.2.2. Posibles problemas

La siguiente son una serie de problemas encontrados al ejecutar FreeHDL desde línea de órdenes.

### 4.2.3. Error: `No default binding for component xxx found!`

Si obtenemos este error puede ser porque los archivos fuente de VHDL utilizan la extensión .vhd en vez de .vhdl, que parece que FreeHDL necesita. Es decir, los archivos a utilizar con FreeHDL deberían ser `buffer_amplifier.vhdl` y `buffer_amplifier_tb.vhdl`, en lugar de archivos `buffer_amplifier.vhd` y `buffer_amplifier_tb.vhd` (¡a quién se le ocurre utilizar la extensión .vhd!).

```
$ buffer_amplifier_tb -cmd "dc -f buffer_amplifier_tb.vcd; d; run 1000 ns; q
    ;"
Sorry, only default component binding is currently supported. No default
    binding for component buffer_amplifier found!
```

### 4.2.4. Error: `undefined reference to 'main'`

Si genera el error `'undefined reference to 'main''` es porque el nombre del archivo VHDL no existe (p.e. se ha escrito una extensión diferente: .vhdl en vez de .vhd). Suponiendo que existen los archivos `buffer_amplifier.vhd` y `buffer_amplifier_tb.vhd` se invoca a `gvhdl` utilizando la extensión .vhdl:

```
$ gvhdl --libieee -l work buffer_amplifier_tb.vhdl buffer_amplifier.o
gvhdl: FreeHDL root path is '/usr'.
gvhdl: executing '/usr/bin/freehdl-v2cc -m buffer_amplifier_tb._main_.cc -L
    /usr/share/freehdl/lib   -o buffer_amplifier_tb.cc buffer_amplifier_tb.
    vhdl'
gvhdl:
gvhdl: ================================
gvhdl: Compiling 'buffer_amplifier_tb.cc'...
gvhdl: ================================
gvhdl: x86_64-linux-gnu-g++  -I /usr/include -c buffer_amplifier_tb.cc
gvhdl:
gvhdl: ================================
gvhdl: Compiling simulator main file 'buffer_amplifier_tb._main_.cc'...
gvhdl: ================================
gvhdl: x86_64-linux-gnu-g++  -I /usr/include -c buffer_amplifier_tb._main_.
    cc
```

```
gvhdl: Linking simulator 'buffer_amplifier_tb'...
gvhdl: libtool --mode=link --tag=CXX x86_64-linux-gnu-g++
    buffer_amplifier_tb._main_.o  buffer_amplifier_tb.o buffer_amplifier.o -
    lm /usr/lib/libfreehdl-kernel.la /usr/lib/libfreehdl-std.la /usr/lib/
    freehdl/libieee.la -o buffer_amplifier_tb
linker: libtool: link: x86_64-linux-gnu-g++ buffer_amplifier_tb._main_.o
    buffer_amplifier_tb.o buffer_amplifier.o -o buffer_amplifier_tb  -lm /
    usr/lib/libfreehdl-kernel.so /usr/lib/libfreehdl-std.so /usr/lib/freehdl
    /libieee.so -Wl,-rpath -Wl,/usr/lib/freehdl -Wl,-rpath -Wl,/usr/lib/
    freehdl
linker: /usr/lib/gcc/x86_64-linux-gnu/4.6/../../../x86_64-linux-gnu/crt1.o:
    In function `_start':
linker: (.text+0x20): undefined reference to `main'
linker: collect2: ld returned 1 exit status
gvhdl: Linking failed!
Died at /usr/bin/gvhdl line 277.
```

### 4.2.5. Error al compilar yyy: `work.xxx is undeclared`

Si no se especifica librería de trabajo los archivos se generarán en el directorio del proyecto, y no se podrán reutilizar otros componentes. Para eso hay que especificar dónde reside la librería `'work'` (no fue fácil, la solución en chino en este enlace).

```
$ gvhdl --libieee -l work buffer_amplifier_tb.vhdl buffer_amplifier.o
gvhdl: FreeHDL root path is '/usr'.
gvhdl: executing '/usr/bin/freehdl-v2cc -m buffer_amplifier_tb._main_.cc -L
    /usr/share/freehdl/lib -l work  -o buffer_amplifier_tb.cc
    buffer_amplifier_tb.vhd'
buffer_amplifier_tb.vhd: in buffer_amplifier_tb(Testbench):
buffer_amplifier_tb.vhd:31: WORK.buffer_amplifier is undeclared
v2cc: buffer_amplifier_tb.vhd: 1 errors
gvhdl: Compilation failed!
Died at /usr/bin/gvhdl line 212.
```

La solución consiste en especificar como nombre de librería de trabajo la del subdirectorio dentro del *workspace*, es decir, el proyecto con la opción `'-l'`, y el directorio padre (..) como ruta base del *workspace* con la opción `'-L'`.

Por tanto, un fragmento de lo que podría ser un *script* para la compilación de un proyecto típico con FreeHDL podría ser el siguiente; aunque todo ello ya lo tiene en cuenta la aplicación UMHDL. Como se comentó al comienzo de este manual, el instalador de UMHDL crea un subdirectorio `scripts` con archivos por lotes y *scripts* para todas las plataformas, y para cada compilador VHDL:

```
$ MODULE=buffer_amplifier
$ MODULETB=buffer_amplifier_tb

$ gvhdl -L .. -l $MODULE --libieee $MODULE.vhdl -c
```

```
$ gvhdl -L .. -l $MODULE --libieee $MODULETB.vhdl $MODULE.o --relaxed-
    component-visibility
$ ./$MODULETB -cmd "dc -f $MODULE.vcd -t 1 ns;d;run 1000 ns;q;"
$ gtkwave $MODULE.vcd
```

## 4.3. FreeHDL :: Mac OS X (TO-DO)

Existe un paquete de MacPorts (port FreeHDL) denominado `'freehdl'`. Por tanto, antes de instalar FreeHDL se requiere tener instalado MacPorts (ver sección 2.3.1).

Instalo FreeHDL mediante mac ports, seleccionando el paquete "freehdl". Antes tendremos que actualizar macports y los paquetes previamente instalados:

```
$ port selfupdate --nosync
$ port selfupdate
$ port upgrade outdated

$ port install freehdl
```

MacPorts descargará el archivo al directorio `/opt/local/var/macports/distfiles/freehdl`, y lo instalará en `/opt/local/bin/`, lo cual puede comprobarse mediante lo siguiente:

```
$ port contents freehdl
  /opt/local/bin/freehdl-config
  /opt/local/bin/freehdl-gennodes
  /opt/local/bin/freehdl-v2cc
  /opt/local/bin/gvhdl
  ...
  /opt/local/lib/freehdl/libieee.a
  /opt/local/lib/freehdl/libieee.la
  ...
$ freehdl-config --version
0.0.7
```

Actualmente se instala la versión 0.0.7, como puede comprobarse de las órdenes anteriores, o comprobando el archivo descargado: `/opt/local/var/macports/distfiles/freehdl/freehdl-0.0.7.tar.gz`.

---

### 4.3.1. Ejemplo

Vamos a compilar como primer ejemplo un proyecto denominado `'buffer_amplifier'` con los archivos `buffer_amplifier.vhdl` y `buffer_amplifier_tb.vhdl`. La forma de proceder sería similar al caso del sistema Linux (sección 4.2.1);

por mostrar otras posibilidades, a continuación lo hacemos sin utilizar el script `gvhdl`, haciendo uso directamente del traductor de VHDL a lenguaje C (`freehdl-v2cc`) y el compilador del lenguage C++, pero sería más sencillo como se muestra en Linux.

```
$ freehdl-v2cc -m buffer_amplifier._main_.cc -o buffer_amplifier.cc
    buffer_amplifier.vhd
$ g++ -I/opt/local/include -c buffer_amplifier.cc
$ g++ -I/opt/local/include -c buffer_amplifier._main_.cc
$ glibtool --mode=link --tag=CXX g++ buffer_amplifier._main_.o
    buffer_amplifier.o \
  -lm /opt/local/lib/libfreehdl-kernel.la /opt/local/lib/libfreehdl-std.la -
      o buffer_amplifier
```

La primera orden genera los archivos fuente en C `buffer_amplifier._main_.cc` y `buffer_amplifier.cc` mediante el traductor `freehdl-v2cc`. Ambos son compilados posteriormente mediante el compilador de C++, generando los archivos binarios .o de mismo nombre. Como vemos, es necesario especificar correctamente la ruta para los `include`, de lo contrario no se encontrarán y obtendremos el error `'xxx.h file not found'`. Finalmente se enlaza todo con la orden `glibtool`.

Como se ha comentado antes, todo lo anterior se hace de forma más cómoda utilizando el script `/opt/local/bin/gvhdl`, aunque hay que modificar ligeramente porque se obtienen los dos siguientes errores al compilar o al enlazar:

```
$ gvhdl buffer_amplifier.vhd
gvhdl: FreeHDL root path is '/opt/local'.
gvhdl: executing '/opt/local/bin/freehdl-v2cc -m buffer_amplifier._main_.cc
    -L /opt/local/share/freehdl/lib   -o buffer_amplifier.cc
    buffer_amplifier.vhd'
gvhdl:
gvhdl: ================================
gvhdl: Compiling 'buffer_amplifier.cc'...
gvhdl: ================================
gvhdl: /usr/bin/g++-4.2  -I /opt/local/include -c buffer_amplifier.cc
gvhdl: Compilation failed!
Died at /opt/local/bin/gvhdl line 234.

...

glibtool: link: unable to infer tagged configuration
glibtool:   error: specify a tag with '--tag'
```

Modificamos el *script* `/opt/local/bin/gvhdl`, cambiando la variable que especifica el compilador; en mi caso cambié dos líneas, para solventar los dos errores anteriores:

```
   my $cc = "/usr/bin/g++-4.2";
   my $libtool_options = "--mode=link";
```

por:

```
my $cc = "/usr/bin/g++";
my $libtool_options = "--mode=link --tag=CXX";
```

Para compilar utilizando el tipo `'std_logic'` es necesario especificar la opción `'-libieee'` al *script* `gvhdl`, de lo contrario se obtiene el siguiente error:

```
$ gvhdl buffer_amplifier.vhd buffer_amplifier_tb.vhd
linker: Undefined symbols for architecture x86_64:
linker:   _L4ieee_Q14std_logic_1164_I9std_logic_INFO", referenced from:
linker:   "L4ieee_Q11numeric_std_init()", referenced from:
linker:   "L4ieee_Q14std_logic_1164_init()", referenced from:
linker:   "name_stack::set(...)
linker:   "name_stack::push(...)

$ gvhdl --libieee buffer_amplifier.vhd buffer_amplifier_tb.vhd
linker: Undefined symbols for architecture x86_64:
linker:   "name_stack::set(...)
linker:   "name_stack::push(...)
```

Vemos que ahora se generan otros errores al enlazar, informando de que no encuentra las funciones `set()` y `push()` de la clase `name_stack`. Estas funciones están en el archivo `name_stack.cc` de la librería `'kernel'`, que ya se incluye, por lo que ... no se entiende.

Visto lo cual, pruebo a desinstalar FreeHDL 0.0.7 e intento instalar la versión 0.0.8 existente en la página de Qucs.

```
$ port uninstall freehdl
$ port clean freehdl
--->  Cleaning freehdl

$ tar -xvzf freehdl-0.0.8.tar.gz -C /opt/local
$ cd /opt/local/freehdl-0.0.8
$ ./configure
$ make
$ make check      <-- Opcional
$ make install
$ make clean
```

Debido a otros problemas, decido desinstalar FreeHDL 0.0.8 y volver a instalar la versión 0.0.7 instalada previamente mediante MacPorts: ahora ya no me deja instalarlo:

```
$ port install freehdl
--->  Computing dependencies for freehdl
...
--->  Building freehdl
Error: org.macports.build for port freehdl returned: command execution
    failed
Please see the log file for port freehdl for details:
```

```
    /opt/local/var/macports/logs/_opt_local_var_macports_sources_rsync.
        macports.org_release_ports_science_freehdl/freehdl/main.log
To report a bug, follow the instructions in the guide:
    http://guide.macports.org/#project.tickets
Error: Processing of port freehdl failed
```

Uff! Parece que varios ya se han quejado del asunto.

De momento nos quedamos sin haber probado FreeHDL en Mac OS X. Si lo consigues, escríbeme un correo. Gracias!

# Capítulo 5

# nvc

Se trata de otro compilador y simulador VHDL, muy similar a GHDL, algo más complicado de instalar, sobre todo en Windows.

Algunos enlaces:

- https://github.com/nickg/nvc

## 5.1. nvc :: Windows

El compilador nvc se puede ejecutar en Windows a través de Cygwin (ver a continuación), por lo que es un requisito a instalar previamente.

### 5.1.1. Requisitos previos

#### Cygwin

Cygwin es un conjunto de herramientas abiertas de desarrollo que proporcionan una funcionalidad similar a las de una distribución Linux (compilador gcc, ...).

Para instalar Cygwin hay que descargar un **instalador** `setup-xxx.exe`, el cual se ejecutará tanto para instalar Cygwin como para instalar nuevos paquetes o actualizar los ya instalados. Dependiendo de nuestra versión de Windows, el enlace para descargar dicho instalador son los siguientes:

- Para Windows 32-bit: setup-x86.exe
- Para Windows 64-bit: setup-x86_64.exe

Los enlaces anteriores están accesibles desde las siguientes páginas:

- https://www.cygwin.com

- http://cygwin.com/install.html

Para ejecutar el instalador `setup-xxx.exe`, en el caso de Windows 64-bit especificaremos además la opción '–no-admin'; en el caso de 32-bit no es necesario especificar nada:

```
C:\> setup-x86.exe
C:\> setup-x86_64.exe --no-admin
```

El instalador es un ejecutable de poco tamaño, el cual se descargará desde el origen que elijamos los paquetes que hayamos seleccionado, que para el caso que nos ocupa de instalar el compilador nvc se deberá incluir los siguientes:

- En Devel y Libs: gcc, g++, clang
- En Devel: make, automake, autoconf, flex, pkg-config
- En Interpreters y Libs: llvm, libllvm-devel
- En Tcl: tcl, Tcl-tk, Tcl-tk-devel
- En Devel y Libs: libreadline-devel, libffi-devel, libncurses-devel
- En Devel: git-completion, git-gui y gitk.
- En net: openssh

Una vez seleccionados los paquetes aceptaremos ([Next]) para que se descarguen e instalen dichos paquetes. Finalmente añadir la ruta `C:\cygwin\bin` a la variable `PATH` del sistema.

### GitHub

Una vez instalado Cygwin, abriremos la consola para instalar nvc, para lo cual nos hará falta tener GitHub instalado (orden `'git'`); si hemos seleccionado los paquetes mencionados anteriormente ya lo deberemos tener. Para comprobarlo ejecutaremos lo siguiente desde la consola de Cygwin:

```
$ git
-bash: git: command not found
```

Si obtenemos un mensaje como el anterior es que no lo tenemos instalado. Ejecutaremos de nuevo el instalador `setup-xxx.exe` y nos aseguraremos de seleccionar los paquetes relacionados con 'git' y 'openssh' (ver arriba).

### 5.1.2. Instalación de nvc

Una vez instalado Cygwin y git, nos movemos al lugar donde vayamos a instalar el compilador. Puesto que el compilador se invocará desde Windows, utilizaremos un directorio corto y sin espacios en blanco; en lo siguiente supononemos que se va a instalar el compilador en la ruta `C:\nvc` (desde Windows), que se corresponde con la ruta `/cygdrive/c/nvc` desde Cygwin. Por tanto, comenzamos con movernos a la ruta `/cygdrive/c`, y desde ahí seguiremos los siguientes pasos para instalar el compilador nvc utilizando 'git' para descargar los fuentes de nvc desde su repositiorio oficial:

```
$ cd /cygdrive/c
$ git clone https://github.com/nickg/nvc [:/cygdrive/c/nvc]
```

Con lo anterior se crea un subdirectorio `/cygdrive/c/nvc` desde Cygwin (`C:\nvc` desde Windows) con el código fuente de nvc descargado. A continuación, nos movemos a dicho directorio y procedemos a compilar el programa. Las órdenes serían las siguientes; más abajo se vuelven a repetir pero incluyendo la salida de las mismas:

```
$ cd nvc
$ ./autogen.sh
$ ./tools/fetch-ieee.sh
$ mkdir build && cd build
$ ../configure
$ make
$ make install
```

La salida de estas órdenes se muestra a continuación (abreviada por claridad):

```
$ pwd
/home/Pablo

$ cd /cygdrive/c

$ git clone https://github.com/nickg/nvc [:/cygdrive/c/nvc]
Cloning into 'nvc'...
remote: Counting objects: 19547, done.
remote: Total 19547 (delta 0), reused 0 (delta 0), pack-reused 19547
Receiving objects: 100% (19547/19547), 8.83 MiB | 262.00 KiB/s, done.
Resolving deltas: 100% (13554/13554), done.
Checking connectivity... done.

$ cd nvc
$ ./autogen.sh
configure.ac:23: installing './compile'
configure.ac:21: installing './config.guess'
configure.ac:21: installing './config.sub'
configure.ac:15: installing './install-sh'
configure.ac:15: installing './missing'
Makefile.am: installing './depcomp'
```

```
configure.ac: installing './ylwrap'

$ ./tools/fetch-ieee.sh
This program will download IEEE library sources from
  http://standards.ieee.org/downloads/1076/1076.2-1996
  http://svn.gna.org/svn/ghdl/trunk/libraries/vital2000
and modify them by uncommenting the definition of operator XNOR.

Continue? (yes/no) yes
math_complex-body.vhdl
math_complex.vhdl
math_real-body.vhdl
math_real.vhdl
numeric_bit-body.vhdl
numeric_bit.vhdl
numeric_std-body.vhdl
numeric_std.vhdl
std_logic_1164-body.vhdl
./tools/fetch-ieee.shstd_logic_1164.vhdl
timing_p.vhdl
timing_b.vhdl
prmtvs_p.vhdl
prmtvs_b.vhdl
memory_p.vhdl
memory_b.vhdl

$ mkdir build && cd build
$ ../configure
...
checking for LLVM (engine bitreader bitwriter)... no
checking for LLVM shared library... yes
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
configure: WARNING: libcheck not found - unit tests will not run
...

$ make
  CC       thirdparty/fstapi.o
  CC       thirdparty/lz4.o
  AR       lib/libfst.a
  CC       thirdparty/lxt_write.o
  AR       lib/liblxt.a
  CC       thirdparty/fastlz.o
  AR       lib/libfastlz.a
  CC       src/lib_libcgen_a-cgen.o
  AR       lib/libcgen.a
  CC       src/lib.o
  CC       src/util.o
...
  NATIVE   lib/ieee/IEEE.STD_LOGIC_TEXTIO
  NVC      lib/synopsys/SYNOPSYS.ATTRIBUTES
  NVC      lib/ieee/IEEE.STD_LOGIC_MISC
  NATIVE   lib/ieee/IEEE.STD_LOGIC_MISC

$ make install
 /usr/bin/mkdir -p '/usr/local/bin'
  /usr/bin/install -c bin/nvc.exe '/usr/local/bin'
 /usr/bin/mkdir -p '/usr/local/share/nvc/ieee'
 /usr/bin/install -c ...
...
```

Si todo lo anterior ha ido correctamente, el programa se instala la ruta `/usr/local/bin` y las librerías en `/usr/local/share/nvc`, cosa que podemos comprobar con las siguientes órdenes:

```
$ which nvc
/usr/local/bin/nvc

$ nvc --help
...
Library search paths:
  /home/Pablo/.nvc/lib
  /usr/local/share/nvc
```

### 5.1.3. Ejemplo

En el punto actual ya será posible ejecutar nvc desde la consola de Cygwin. Como la ruta `/usr/local/bin` de Cygwin se corresponde con `C:\cygwin\bin` desde Windows, para invocar a nvc desde Windows (como hace la aplicación UMHDL), nos aseguraremos de que la ruta `C:\cygwin\bin` está en la variable `PATH` de Windows. En esa ruta deben estar los programas: bash.exe, run.exe, etc. Si comprobamos esto, UMHDL ya debería funcionar correctamente con el compilador nvc.

No obstante, a continuación se explica cómo se puede ejecutar nvc desde la línea de órdenes de MS-DOS, sin llegar a abrir la consola de Cygwin. Esto es lo que hace UMHDL, por lo que puedes saltarte el resto de la sección si vas a utilizar UMHDL.

Ejecutando bash.exe se inicia la consola de Cygwin, pero si especificamos lo siguiente podemos ejecutar una orden de Cygwin sin que ni siquiera aparezca la consola.

```
C:> C:\cygwin\bin\bash.exe --login -c "nvc ..."
```

Como ejemplo práctico, supongamos que tenemos un proyecto de VHDL en el directorio `C:\UMHDL\and2`, con los archivos `and2.vhd` y `and2_tb.vhd`:

```
C:\> set MODULE=and2
C:\> set MODULETB=and2_tb
C:\> cd C:\UMHDL\%MODULE%
C:\> set CYGWIN_BASH=C:\cygwin\bin\bash.exe --login -c
```

Por defecto, el directorio de trabajo cuando se invoca una orden con `bash.exe` es el `HOME` dentro de Cygwin, no la ruta actual en Windows. Podemos comprobarlo ejecutando la orden `'pwd'` para conocer la ruta actual:

```
C:\> %CYGWIN_BASH% "pwd"
```

```
/home/Pablo
```

Por eso hay cambiar al directorio concreto del proyecto antes de invocar al compilador nvc:

```
C:\> set CMD_CD=cd /cygdrive/c/UMHDL/%MODULE%
C:\> %CYGWIN_BASH% "%CMD_CD% && nvc --work=work:../work -a %MODULE%.vhd %
    MODULETB%.vhd"
C:\> %CYGWIN_BASH% "%CMD_CD% && nvc --work=work:../work -e %MODULETB%"
C:\> %CYGWIN_BASH% "%CMD_CD% && nvc --work=work:../work -r --stop-time=500ns
     --format=vcd --wave=%MODULE%.vcd %MODULETB%"
```

Tanto el directorio al que cambiar, como el directorio donde reside la librería de trabajo `'work'` se deben especificar según Cygwin (formato de Linux), no de Windows.

En las órdenes anteriores se han utilizado variables de entorno pensando en la escritura de un archivo por lotes .bat que facilite la tarea, el cual podría recibir como único argumento el nombre del módulo a compilar/simular.

Como vemos, las órdenes necesarias para compilar y ejecutar un proyecto mediante nvc son muy similares a las del compilador GHDL, exceptuando la historia de tener que ejecutarlas vía Cygwin y el cambio previo de directorio. En el caso de Linux o Mac se muestra este mismo ejemplo de forma más clara.

## 5.2. nvc :: Linux

La instalación de nvc en Linux es relativamente sencilla. Primero es necesario instalar una serie de paquetes, y finalmente se instalará nvc de forma similar a las otras plataformas.

### 5.2.1. Requisitos previos

Mediante el gestor de paquetes según nuestra distribución (apt-get, Synaptic, yum, ...) instalaremos una serie de paquetes. En el caso de Linux Mint (Ubuntu), la sentencia sería:

```
$ sudo apt-get install build-essential automake autoconf autoconf-archive \
    flex libreadline-dev tcl-dev check llvm-dev pkg-config zlib1g-dev curl
```

Si se obtiene el error 'Dependencias incumplidas' ejecutar la siguiente orden y volver a intentar la orden anterior:

```
$ sudo apt-get -f install
```

Por otro lado, parece que requiere también llvm y clang, de versiones posteriores a 3.0. Desde el gestor de paquetes Synaptic marcar para instalar: llvm-3.4 y clang-3.4 (ver el siguiente apartado).

### 5.2.2. Instalación de nvc

Una vez instalados los paquetes del apartado anterior podemos proceder a la instalación de nvc. Nos movemos a un directorio de trabajo temporal, en el que se creará un directorio `nvc` que contendrá el código fuente descargado desde GitHub, por ejemplo, nuestro directorio raíz de usuario (`HOME`): `/home/usuario/`, y ejecutaremos las siguientes órdenes; más abajo se vuelven a repetir pero incluyendo la salida de las mismas:

```
$ cd
$ git clone https://github.com/nickg/nvc
$ cd nvc
$ ./autogen.sh
$ ./tools/fetch-ieee.sh
$ mkdir build && cd build
$ ../configure
$ make
$ make install
```

La salida de estas órdenes se muestra a continuación (abreviada por claridad):

```
$ cd
$ git clone https://github.com/nickg/nvc
Cloning into 'nvc'...
remote: Counting objects: 19547, done.
remote: Total 19547 (delta 0), reused 0 (delta 0), pack-reused 19547
Receiving objects: 100% (19547/19547), 8.83 MiB | 262.00 KiB/s, done.
Resolving deltas: 100% (13554/13554), done.
Checking connectivity... done.

$ cd nvc
$ ./autogen.sh
configure.ac:25: installing `./compile'
configure.ac:21: installing `./config.guess'
configure.ac:21: installing `./config.sub'
configure.ac:15: installing `./install-sh'
configure.ac:15: installing `./missing'
Makefile.am: installing `./depcomp'
configure.ac: installing `./ylwrap'

$ ./tools/fetch-ieee.sh
This program will download IEEE library sources from
  http://standards.ieee.org/downloads/1076/1076.2-1996
  http://svn.gna.org/svn/ghdl/trunk/libraries/vital2000
and modify them by uncommenting the definition of operator XNOR.

Continue? (yes/no) yes
math_complex-body.vhdl
math_complex.vhdl
```

```
math_real-body.vhdl
math_real.vhdl
numeric_bit-body.vhdl
numeric_bit.vhdl
numeric_std-body.vhdl
numeric_std.vhdl
std_logic_1164-body.vhdl
./tools/fetch-ieee.shstd_logic_1164.vhdl
timing_p.vhdl
timing_b.vhdl
prmtvs_p.vhdl
prmtvs_b.vhdl
memory_p.vhdl
memory_b.vhdl
```

A continuación procedemos a compilar el proyecto:

```
$ mkdir build && cd build
$ ../configure
...
checking for deflate in -lz... yes
configure: error: LLVM version 3.0 or later required
```

Si no se encuentran las librerías llvm hay que asegurarse de que el ejecutable `llvm-config` está accesible en el `PATH`, o se ha especificado el parámetro `-with-llvm=<absolute_path_to_llvm_bin_directory>` y, según el mensaje, que sean de una versión 3.0 o posterior:

```
$ which llvm-config
/usr/bin/llvm-config
$ llvm-config --version
2.9
```

Podemos comprobar que sí que están en el `PATH`, pero el problema es que se ha instalado una versión anterior a la 3.0. Aunque se ejecute con la opción `-with-llvm` seguimos teniendo el problema.

```
$ ../configure --with-llvm=/usr/bin
```

Para solucionar el problema anterior, desde Synaptic marcar para instalar los paquetes: llvm-3.4 y clang-3.4, y comprobamos la versión instalada cuando se instalen. Además, añadiremos la ruta a la variable de entorno `PATH` tal como se muestra:

```
$ llvm-config --version
3.4

$ clang --version
Ubuntu clang version 3.4-1ubuntu3~precise2 (tags/RELEASE_34/final) (based on
     LLVM 3.4)
Target: x86_64-pc-linux-gnu
Thread model: posix
```

```
$ export PATH=$PATH:/usr/lib/llvm-3.4/bin
```

```
$ ../configure
...
checking for LLVM (engine bitreader bitwriter)... no
checking for LLVM shared library... yes
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
...

$ make
$ make install
```

Si todo lo anterior ha ido correctamente, el programa se instala la ruta `/usr/local/bin` y las librerías en `/usr/local/share/nvc`, lo cual podemos comprobar con las siguientes órdenes:

```
$ which nvc
/usr/local/bin/nvc

$ nvc --help
...
Library search paths:
  /home/pgarrido/.nvc/lib
  /usr/local/share/nvc
```

### 5.2.3. Ejemplo

Como ejemplo de prueba del uso de nvc podemos compilar y simular un modelo sencillo como el `and2.vhd`:

```
$ nvc --work=work:../work -a and2.vhd and2_tb.vhd
$ nvc --work=work:../work -e and2_tb
$ nvc --work=work:../work -r --stop-time=50ns --format=vcd --wave=and2.vcd
    and2_tb
```

## 5.3. nvc :: Mac OS X

### 5.3.1. Requisitos previos

Para instalar nvc en Mac OS X se ejecutan las mismas instrucciones que para Windows y Linux. Sin embargo, es necesario instalar previamente una serie de paquetes y librerías, que es lo que cambia entre los distintos sistemas. En este caso se requiere: Xcode, autotools, LLVM, git.

### Xcode

Para la instalación de Xcode consultar el apartado 2.3.1.

### autotools (Autoconf, Automake, Libtool)

Hasta la versión Mac OS X 10.7 (Lion), habiendo instalado Xcode se tenía también las utilidades AutoTools, es decir, Autoconf, Automake, y Libtool, que se utilizan para generar makefiles y librerías dinámicas. En caso de necesitar instalarlas de forma manual, lo más sencillo es utilizar un gestor de paquetes como MacPorts (o Fink, Brew, ...) del siguiente modo:

```
$ port install autoconf
$ port install automake
$ port install libtool
```

Otra posibilidad es obtener las últimas versiones directamente con las siguientes órdenes:

```
$ curl -OL http://ftpmirror.gnu.org/autoconf/autoconf-2.69.tar.gz
$ tar -xzf autoconf-2.69.tar.gz
$ cd autoconf-2.69
$ ./configure && make && sudo make install

$ curl -OL http://ftpmirror.gnu.org/automake/automake-1.14.tar.gz
$ tar -xzf automake-1.14.tar.gz
$ cd automake-1.14
$ ./configure && make && sudo make install

$ curl -OL http://ftpmirror.gnu.org/libtool/libtool-2.4.2.tar.gz
$ tar -xzf libtool-2.4.2.tar.gz
$ cd libtool-2.4.2
$ ./configure && make && sudo make install
```

### LLVM (Low-Level Virtual Machine)

La *Low-Level Virtual Machine* (LLVM) es un conjunto de librerías y herramientas que facilitan la tarea de construir compiladores, optimizadores, generadores de código, y otras utilidades de desarrollo necesarios para el compilador nvc.

Descargar el binario desde http://llvm.org/releases/download.html y desempaquetarlo (por ejemplo al directorio /usr/local):

```
$ su
$ tar -xvf clang+llvm-3.6.1-x86_64-apple-darwin.tar.xz -C /usr/local
```

**GitHub**

Para instalar GitHub, lo más sencillo es utilizar instalar el paquete `'git-core'` mediante un gestor de paquetes como MacPorts. Como es habitual, conviene primero actualizar los paquetes antes de instalar el paquete requerido, como superusuario o mediante `sudo`:

```
$ sudo port selfupdate
$ sudo port search git-core
$ sudo port variants git-core
$ sudo port install git-core
```

Finalmente comprobaremos que se ha instalado ejecutando la orden `'git'`:

```
>git --help
usage: git [--version] [--help] [-c name=value] [--exec-path[=]] ...
```

### 5.3.2. Instalación de nvc

Una vez instalados los paquetes del apartado anterior podemos proceder a la instalación de nvc. Para ello, nos movemos a un directorio de trabajo temporal, en el que se creará un directorio `nvc` que contendrá el código fuente descargado desde GitHub, por ejemplo, nuestro directorio raíz de usuario (`HOME`): `/Users/usuario/`, y ejecutaremos las siguientes órdenes; más abajo se vuelven a repetir pero incluyendo la salida de las mismas:

```
$ cd
$ git clone https://github.com/nickg/nvc
$ cd nvc
$ ./autogen.sh
$ ./tools/fetch-ieee.sh
$ mkdir build && cd build
$ ../configure
$ make
$ make install
```

La salida de estas órdenes se muestra a continuación (abreviada por claridad):

```
$ cd
$ git clone https://github.com/nickg/nvc
Cloning into 'nvc'...
remote: Counting objects: 19547, done.
remote: Total 19547 (delta 0), reused 0 (delta 0), pack-reused 19547
Receiving objects: 100% (19547/19547), 8.83 MiB | 262.00 KiB/s, done.
Resolving deltas: 100% (13554/13554), done.
Checking connectivity... done.
```

Una vez descargados los fuentes procedemos a compilar el proyecto:

```
$ cd nvc
$ ./autogen.sh

$ ./tools/fetch-ieee.sh
This program will download IEEE library sources from
  http://standards.ieee.org/downloads/1076/1076.2-1996
  http://svn.gna.org/svn/ghdl/trunk/libraries/vital2000
and modify them by uncommenting the definition of operator XNOR.

Continue? (yes/no) yes
math_complex-body.vhdl
math_complex.vhdl
math_real-body.vhdl
math_real.vhdl
numeric_bit-body.vhdl
numeric_bit.vhdl
numeric_std-body.vhdl
numeric_std.vhdl
std_logic_1164-body.vhdl
./tools/fetch-ieee.shstd_logic_1164.vhdl
timing_p.vhdl
timing_b.vhdl
prmtvs_p.vhdl
prmtvs_b.vhdl
memory_p.vhdl
memory_b.vhdl

$ mkdir build && cd build
$ ../configure
...
checking for LLVM (engine bitreader bitwriter)... no
checking for LLVM shared library... no
configure: error: We could not detect the llvm libraries make sure that llvm
    -config is on your path or specified by --with-llvm=<
    absolute_path_to_llvm_bin_directory>

$ ../configure CXX=clang++ CC=clang --with-llvm=/usr/local/clang+llvm-3.6.1-
    x86_64-apple-darwin/bin
<Idem>

$ PATH=$PATH:/usr/local/clang+llvm-3.6.1-x86_64-apple-darwin/bin
$ ../configure CXX=clang++ CC=clang
...
checking for LLVM (engine bitreader bitwriter)... yes
checking for LLVM shared library... no
checking for pkg-config... /opt/local/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
configure: WARNING: libcheck not found - unit tests will not run
...

$ make
$ make install
```

Si todo lo anterior ha ido correctamente, el programa se instala la ruta `/usr/local/bin` y las librerías en `/usr/local/share/nvc`, lo cual podemos comprobar con las siguientes órdenes:

```
$ which nvc
```

```
/usr/local/bin/nvc

$ nvc --help
...
Library search paths:
  /Users/pgarrido/.nvc/lib
  /usr/local/share/nvc
```

### 5.3.3. Ejemplo

Como ejemplo de prueba del uso de nvc podemos compilar y simular un modelo sencillo como el `and2.vhd`, tal y como se muestra a continuación:

```
$ nvc --work=work:../work -a and2.vhd and2_tb.vhd
$ nvc --work=work:../work -e and2_tb
$ nvc --work=work:../work -r --stop-time=50ns --format=vcd --wave=and2.vcd
    and2_tb
```

# Capítulo 6

# GTKWave

GTKWave es un visualizador gráfico de ondas multiplataforma, soportando diversos formatos de archivo como .vcd, .lxt, .lxt2, .vzt, .fst, .ghw, entre otros. Utilizando la aplicación UMHDL se genera únicamente archivos .vcd, que son los menos eficientes, en cuanto a espacio en disco se refiere, y más restringidos, en cuanto a modelar los distintos tipos del lenguaje VHDL. No obstante, es el formato más sencillo y más soportado por los distintos compiladores de VHDL utilizados.

Algunos enlaces:

- http://gtkwave.sourceforge.net/
- http://sourceforge.net/projects/gtkwave/
- http://www.dspia.com/gtkwave.html

## 6.1. GTKWave :: Windows

Como ya se ha comentado previamente, el instalador de UMHDL para Windows incluye tanto el compilador GHDL como a GTKWave, instalándose en un subdirectorio llamado `'Ghdl_gtkwave'`, cuyo subdirectorio `'bin'` es el que debemos añadir a la variable de entorno `PATH` o establecer en las opciones de UMHDL.

No obstante, si se desea instalarlo manualmente, por ejemplo, para instalar alguna versión más reciente, existe un paquete ya compilado para Windows que se puede descargar desde el enlace `http://www.dspia.com/gtkwave.html`. En concreto, podemos seguir los siguientes pasos:

1. Descargar el conjunto de DLLs all_libs.tar.gz. Al descomprimirlo aparece un archivo `all_libs.tar.tar`, que descomprimiremos tal cual, sin modificar el nombre.

2. Descargar el ejecutable de GTKWave gtkwave.exe.gz. Si se utiliza Internet Explorer puede ocurrir que se descomprima automáticamente, pero no se cambie la extensión `gtkwave.exe.gz` a `gtkwave.exe`, cosa que habrá que hacer manualmente.

3. Finalmente, el ejecutable gtkwave.exe lo copiaremos al directorio de las librerías DLLs, el cual debe incluirse en la variable de entorno `PATH`.

## 6.2. GTKWave :: Linux

Al igual que para GHDL, tenemos dos opciones para instalar GTKwave:

1. Descargar e instalar mediante algún **gestor de paquetes**, recomendado porque se comprueban si existen dependencias de otros paquetes: tk, .... Basta con ejecutar la siguiente orden desde la consola:

```
$ sudo apt-get install gtkwave
```

2. Desde el **código fuente**:

```
$ wget http://gtkwave.sourceforge.net/gtkwave-3.3.58.tar.gz
$ sudo tar xvfz gtkwave-3.3.58.tar.gz -C /usr/local
  (genera el directorio /usr/local/gtkwave-3.3.58)
$ sudo ./configure
$ sudo make
$ sudo make install
```

Con los pasos anteriores el programa se instalará en el directorio: `/usr/local/bin`. Debemos asegurarnos de que dicha ruta se encuentra en la variable de entorno `PATH`, o bien, se ha especificado en la sección de GTKWave en las opciones de configuración de la aplicación (sección 2.3.2).

## 6.3. GTKWave :: Mac OS X

Una vez instalado y actualizado MacPorts (ver apartado 2.3.1), ya podemos proceder a la instalación de GTKWave simplemente instalando el paquete `'gtkwave'`:

```
$ port search  gtkwave
$ port install gtkwave
```

El programa se instalará en el directorio `/opt/local/bin`, el cual añadiremos a la variable de entorno `PATH` (sección 2.3.2), o bien, especificaremos en la sección de GTKWave en las opciones de configuración de la aplicación UMHDL.

Existen otras alternativas para instalar GTKWave en Mac OS X, enumerándolas a continuación simplemente a modo de referencia:

1. Puede descargarse un instalador desde: `http://eng-osx.sourceforge.net/GTKwave.html`

2. Puede utilizarse WINE para utilizar el ejecutable de windows: `http://dwellangle.wordpress.com/2010/04/30/installing-ghdl-on-a-mac/`