
PACKAGES Y LIBRARIES DE VHDL

Tutorial para su creación y uso con Vivado

Nicolás Ruiz Requejo (Autor)

María Ángeles Cifredo Chacón (Supervisora)



3 DE ENERO DE 2019

ÁREA DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES
UNIVERSIDAD DE CÁDIZ

Contenido

Contenido	2
Tabla de figuras.....	3
Package de VHDL.....	4
Creación del package: combinational_components	5
Hora de utilizar el package: cláusula <i>use</i> y biblioteca <i>work</i>	7
Nombre seleccionado (selected name).....	9
Consideraciones sobre .all	10
Señales globales	11
Package body	11
Bibliotecas	13
Ver las bibliotecas de un proyecto en Vivado®.....	13
Asignar una biblioteca	14
Cambiar la biblioteca de un fichero de código en nuestro proyecto	14
Crear una biblioteca reutilizable.....	16
Referencias	22

Tabla de figuras

Figura 1. Forma general package declaration	4
Figura 2. Forma general package body	4
Figura 3. Creación módulo combinational_components.....	5
Figura 4. Código declaración de package	5
Figura 5. Copiando fuentes.....	6
Figura 6. combinational_components.vhd.....	7
Figura 7. ALU_1bit.vhd	8
Figura 8. Sintaxis general de la sentencia use	8
Figura 9. sentencia use con work en ALU_1bit.vhd	9
Figura 10. Instanciación de AND_GATE con su selected name.....	9
Figura 11. pkg_constants1.vhd	10
Figura 12. pkg_constants2.vhd	10
Figura 13. Colisión de nombres	10
Figura 14. Colisión de nombres solucionada	11
Figura 15. . Códgo del package body para pkg_constants1.....	12
Figura 16. pkg_constants1 sin la declaración de num_selected.....	12
Figura 17. Error al eliminar la declaración de la constante	12
Figura 18. Propiedad library de un fichero	13
Figura 19. Ficheros en cada biblioteca	14
Figura 20. Crear biblioteca lib_mycomp	15
Figura 21. lib_mycomp está "unreferenced".....	15
Figura 22. Cambiando bibliotecas a lib_mycomp en las propiedades	16
Figura 23. Crear directorio para las fuentes de la biblioteca lib_mycomp.....	17
Figura 24. Eliminar las fuentes de los pasos anteriores	17
Figura 25. Seleccionar directorio de la biblioteca	18
Figura 26. Añadir directorio de las fuentes.....	19
Figura 27. lib_mycomp importada satisfactoriamente	20
Figura 28. Código de ALU_1bit.vhd incluyendo la biblioteca lib_mycomp	21

Package de VHDL

En VHDL un **package** proporciona un depósito donde podemos incluir declaraciones de distintos elementos como por ejemplo: componentes, procedimientos, tipos, otros packages, constantes, señales y variables. De manera que las declaraciones del **package** puedan reutilizarse en diseños diferentes o diferentes partes de un diseño puedan compartir las mismas declaraciones.

Los **package** se componen de dos partes: **package declaration** (declaración de paquete) y **package body** (cuerpo del paquete). El **package declaration** siempre ha de estar presente, define la vista externa del **package** (su interfaz). La implementación de los elementos declarados en un **package declaration** se realiza en su **package body**, por tanto el **package body** será necesario solo cuando declaremos elementos en el **package** que necesiten una implementación, este es el caso de los procedimientos y funciones como por ejemplo la función de conversión `to_integer(unsigned)` del **package** `ieee.numeric_std`.

Veamos la sintaxis resumida del **package declaration**:

```
package package_name is
    [component declarations]
    [signal declarations]
    [constant declarations]
    [variable declarations]
    [procedure declarations]
    [function declarations]
end package;
```

Figura 1. Forma general package declaration

Y la sintaxis del **package body**:

```
package body package_name is
    [function implementation]
    [procedure implementation]
end package body;
```

Figura 2. Forma general package body

Vamos a comenzar con un ejemplo, en el **proyecto 10** teníamos un módulo **ALU_1bit** con un diseño estructural, en la región declarativa de la arquitectura declarábamos los componentes de los módulos que necesitábamos y en el cuerpo de la arquitectura instanciábamos los componentes. Para prescindir de la declaración de los componentes en la arquitectura de **ALU_1bit** vamos a crear un paquete.

Creación del package: combinational_components

Abrimos el proyecto 10 de ALU_1bit y añadimos un nuevo archivo fuente, dejamos el nombre de la entidad como **combinational_components** y no incluiremos ninguna definición de puertos:

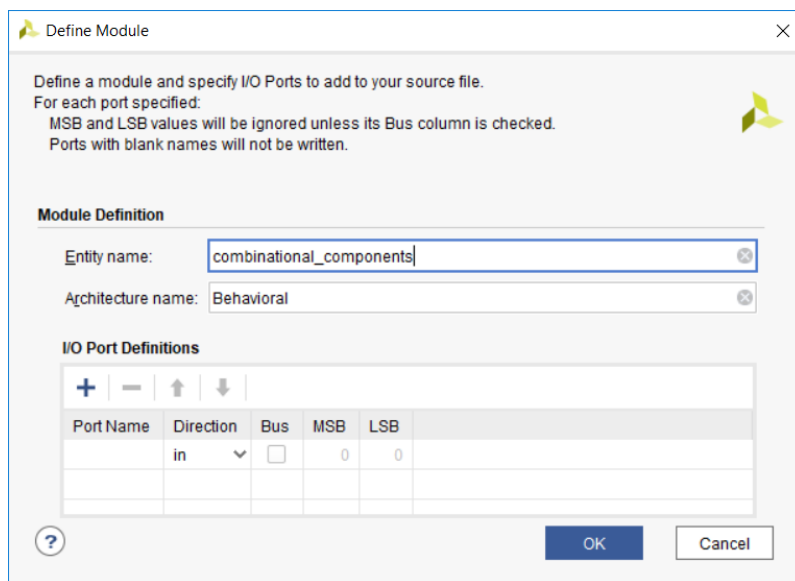


Figura 3. Creación módulo combinational_components

Eliminamos el contenido del fichero **combinational_components.vhd** la (declaración de entity, architecture) y ponemos la declaración del package:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

package combinational_components is

end package;
```

Figura 4. Código declaración de package

Para el diseño estructural de **ALU_1bit** necesitamos componentes de las entidades: **AND_GATE** (Project_01), **XOR_GATE** (Project_03), **FULL_ADDER** (Project_06) y **MUX4** (Project_07).

Los módulos de cada entidad tienen que añadirse al proyecto para poder usarlos, los añadimos de la forma mostrada en la *Figura 5*.

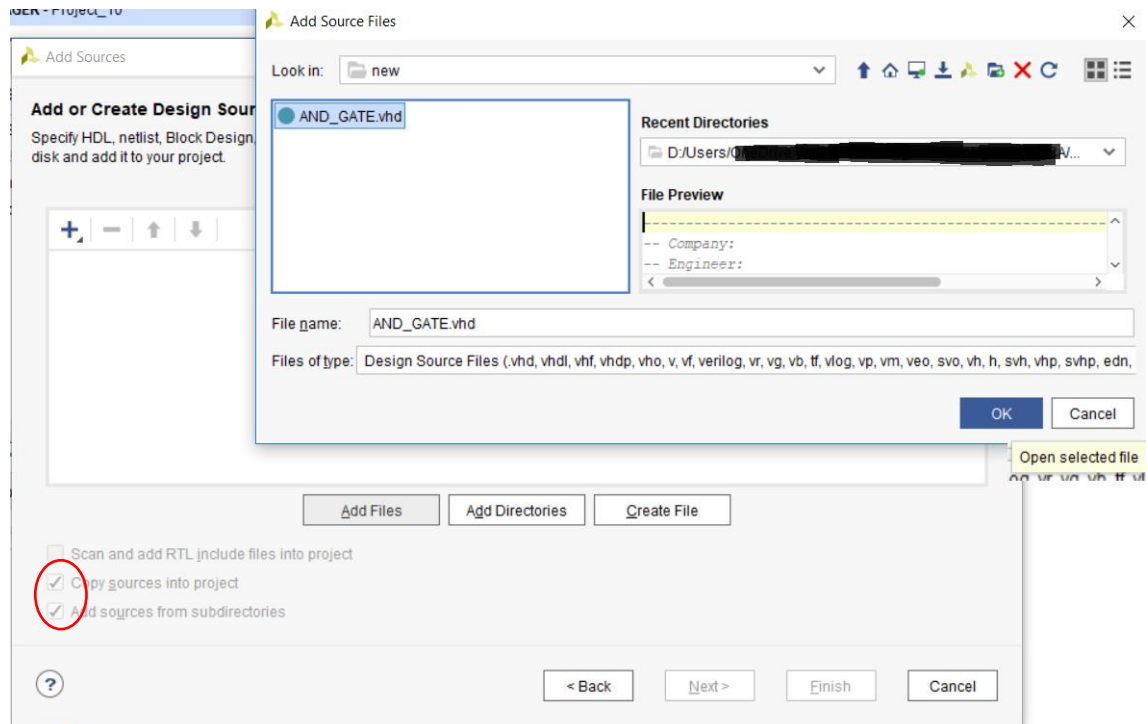


Figura 5. Copiando fuentes

Ahora ponemos en la declaración del package ***combinational_components.vhd*** las declaraciones de componente para cada entidad:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

package combinational_components is

    component AND_GATE
        Port ( A_i : in STD_LOGIC;
              B_i : in STD_LOGIC;
              Z_o : out STD_LOGIC );
    end component;

    component XOR_GATE
        generic (WIDTH : INTEGER := 2);
        Port ( A_i : in STD_LOGIC_VECTOR(WIDTH-1 downto 0);
```

```

        B_i : in STD_LOGIC_VECTOR(WIDTH-1 downto 0);
        Z_i : out STD_LOGIC_VECTOR(WIDTH-1 downto 0));
end component;

component FULL_ADDER
    Port ( A_i : in STD_LOGIC;
           B_i : in STD_LOGIC;
           CARRY_i : in STD_LOGIC;
           SUM_o : out STD_LOGIC;
           CARRY_o : out STD_LOGIC);
end component;

component MUX4
    Generic (width: integer:=2);
    Port ( A_i : in STD_LOGIC_VECTOR (width-1 downto 0);
           B_i : in STD_LOGIC_VECTOR (width-1 downto 0);
           C_i : in STD_LOGIC_VECTOR (width-1 downto 0);
           D_i : in STD_LOGIC_VECTOR (width-1 downto 0);
           SEL_i : in STD_LOGIC_VECTOR (1 downto 0);
           Z_o : out STD_LOGIC_VECTOR (width-1 downto 0));
end component;

end package;

```

Figura 6. *combinational_components.vhd*

Hora de utilizar el package: cláusula *use* y biblioteca *work*

En este punto si echamos un vistazo a ***ALU_1bit.vhd*** (al cual le hemos quitado el código de las declaraciones de componentes en la parte declarativa de la arquitectura) el IDE nos marca las instanciaciones de componentes como erróneas ya que estos no están declarados, para hacer uso de nuestro package necesitamos introducir la sentencia ***use***.

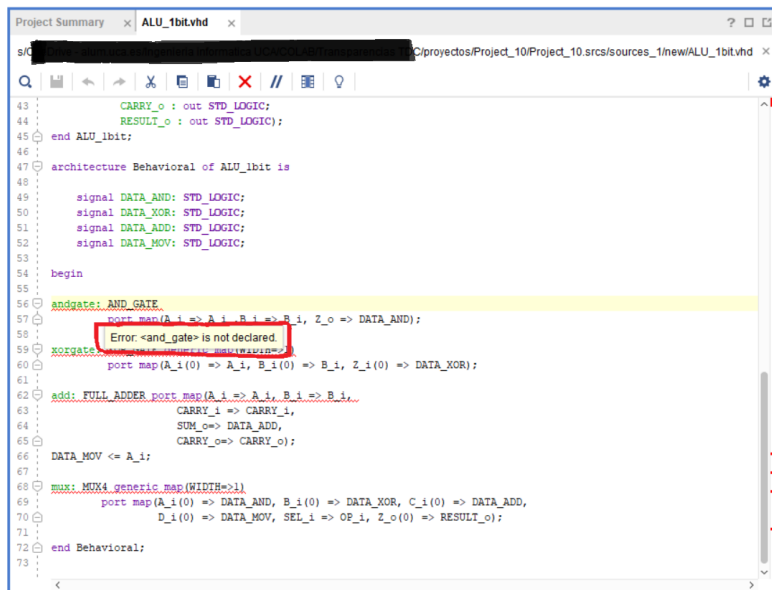


Figura 7. ALU_1bit.vhd

Por defecto los diseños analizados quedan automáticamente disponibles en la biblioteca **work**; **work** no es realmente una biblioteca, denota una referencia a la biblioteca de trabajo actual que por defecto en Vivado® es **xil_defaultlib**. Mas adelante se profundizará en las bibliotecas (libraries). Ahora basta con saber que para utilizar nuestro paquete **combinational_components** necesitamos poner la sentencia **use**; como por defecto todas las fuentes del proyecto están en la librería **work** y lo que está en **work** está automáticamente disponible podemos prescindir de la sentencia “**library work;**” que sería redundante. La sintaxis general de **use** es:

```
use library_name.package_name.item_name;
```

ó

```
use library_name.package_name.all;
```

Figura 8. Sintaxis general de la sentencia use

Con **all** conseguimos que todos los elementos del paquete estén visibles en el fichero de código.

Ponemos la sentencia **use** para nuestro paquete en **ALU_1bit.vhd** y podemos observar que desaparecen los errores en las instanciaciones de los componentes:

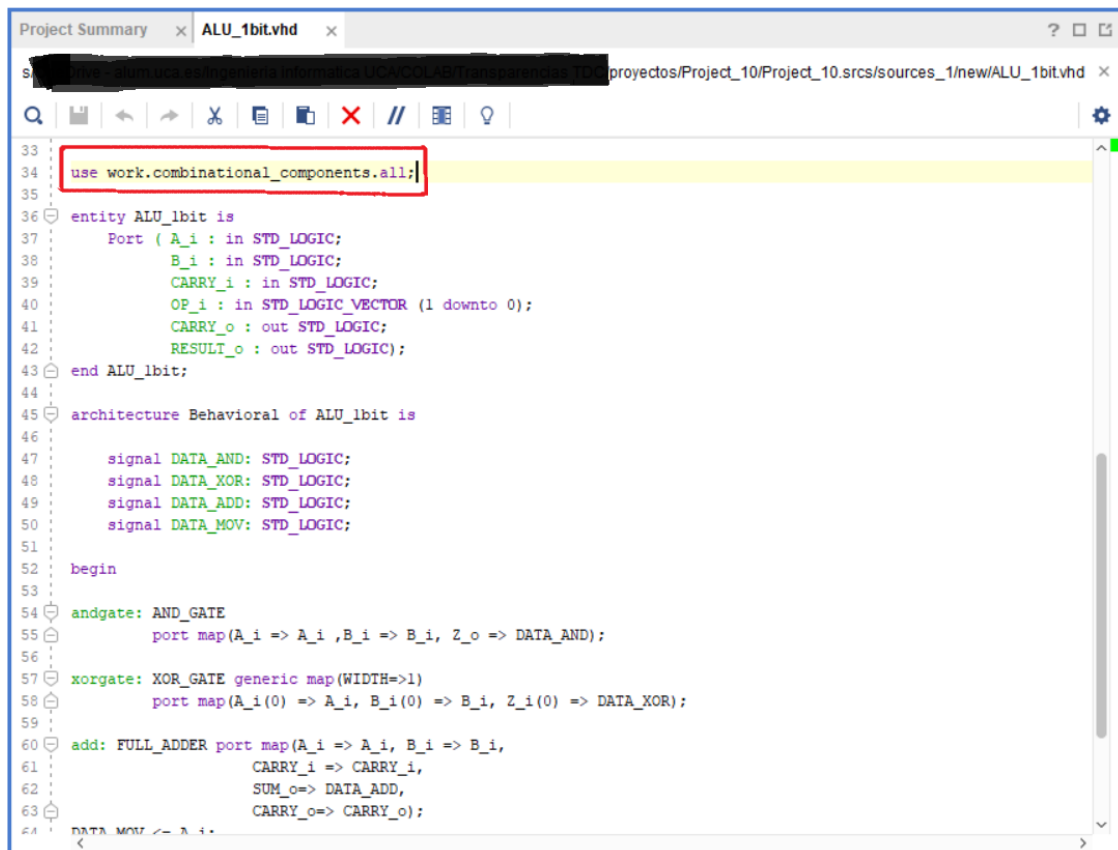


Figura 9. sentencia use con work en ALU_1bit.vhd

Nombre seleccionado (selected name)

Otra forma de referirnos a un elemento de un package sin usar la sentencia **use** es con su “selected name” que consiste en escribir el nombre completo del elemento en la forma: *library_name.package_name.item_name*

En cada lugar en el código que reference a ese elemento. Esta forma es más verbosa y usarla para todo complica el mantenimiento del código. Ejemplo:

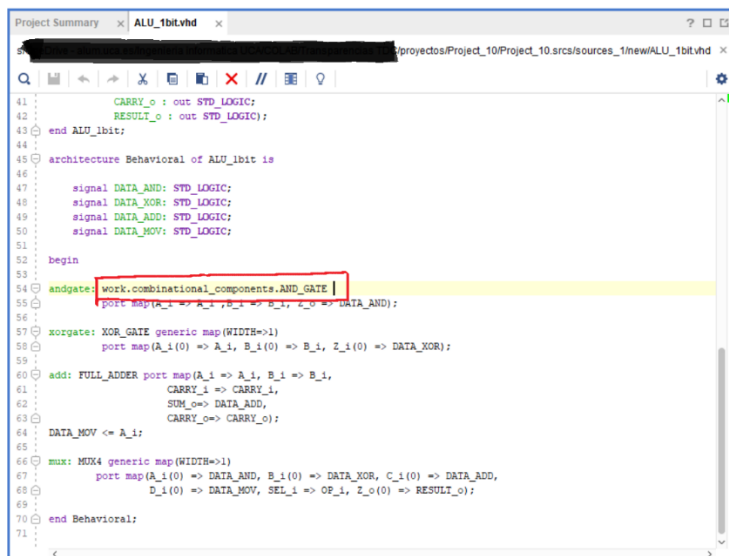


Figura 10. Instanciación de AND_GATE con su selected name

Consideraciones sobre *.all*

Cuando usamos packages de distintas bibliotecas en un diseño y estos contienen elementos con el mismo nombre, estos colisionan (por ejemplo dos constantes que se llamen igual) y VHDL lo arregla haciendo que los elementos que colisionan no sean directamente visibles (la sentencia **use** no surte efecto para ellos). En estos casos **si** debemos usar su “**selected name**” para poder referirnos a ellos en el diseño donde los necesitamos.

Ejemplo: creamos dos packages, **pkg_constants1** y **pkg_constants2**

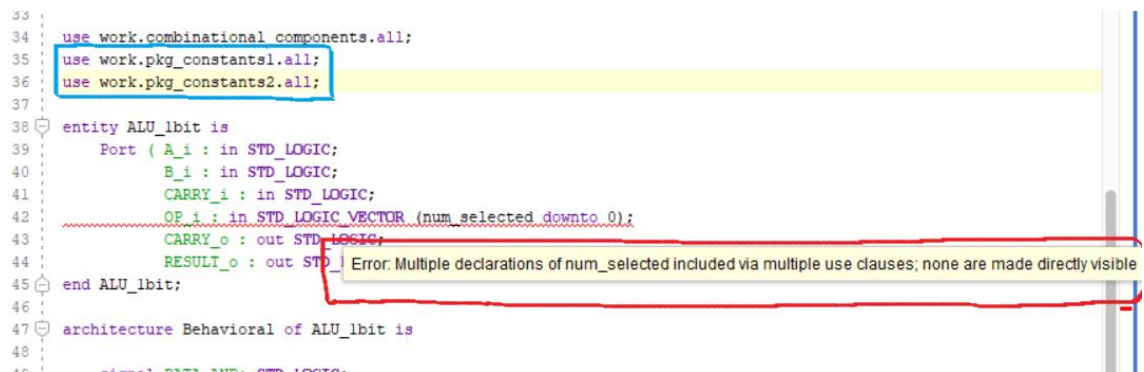
```
package pkg_constants1 is
    constant num_selected : integer := 2;
end package;
```

Figura 11. *pkg_constants1.vhd*

```
package pkg_constants2 is
    constant num_selected : integer := 8;
end package;
```

Figura 12. *pkg_constants2.vhd*

Vamos a usar ambos packages en **ALU_1bit**, pondremos la constante *num_selected* como índice superior del puerto *OP_i* que selecciona la operación de la ALU. La constante que queremos usar es la de **pkg_constants1**, vamos a ver como si no ponemos el nombre seleccionado se produce la colisión de nombres y vhdl no hará visible ninguna de las constantes:



```
33
34 use work.combinational_components.all;
35 use work.pkg_constants1.all;
36 use work.pkg_constants2.all;
37
38 entity ALU_1bit is
39     Port ( A_i : in STD_LOGIC;
40           B_i : in STD_LOGIC;
41           CARRY_i : in STD_LOGIC;
42           OP_i : in STD_LOGIC_VECTOR (num_selected downto 0);
43           CARRY_o : out STD_LOGIC;
44           RESULT_o : out STD_LOGIC);
45 end ALU_1bit;
46
47 architecture Behavioral of ALU_1bit is
48
49     signal DATA_AND_CSTD_LOGIC;
```

Figura 13. Colisión de nombres

Entonces escribimos el nombre seleccionado:

`work.pkg_constants1.num_selected`

Y el analizador ya puede saber la constante que queremos referenciar:

```
34 use work.combinational_components.all;
35 use work.pkg_constants1.all;
36 use work.pkg_constants2.all;
37
38 entity ALU_lbit is
39     Port ( A_i : in STD_LOGIC;
40           B_i : in STD_LOGIC;
41           CARRY_i : in STD_LOGIC;
42           OP_i : in STD_LOGIC_VECTOR (work.pkg_constants1.num_selected-1 downto 0);
43           CARRY_o : out STD_LOGIC;
44           RESULT_o : out STD_LOGIC);
45 end ALU_lbit;
46
```

Figura 14. Colisión de nombres solucionada

Señales globales

En un paquete podemos declarar señales globales con **signal**, estas señales no están restringidas en su uso a un solo cuerpo de arquitectura, pueden ser usadas en cualquier diseño donde se use el paquete en el que está declarada. Se debe usar con cautela pues compartir una señal para varias entidades introduce una dependencia en esos módulos y un cambio en alguno de los módulos podría afectar al comportamiento del resto, se suele referir a esto como que parte de la interfaz está implícita.

Package body

El principal uso del cuerpo de un paquete (*package body*) es la implementación de procedimientos y funciones de VHDL cuyas cabeceras han sido declaradas en su correspondiente declaración de paquete (*package declaration*). En los casos en los que solo necesitamos declarar elementos como componentes, tipos, señales o constantes basta con utilizar una declaración de paquete. En el caso de las constantes podemos optar por declararlas en la declaración de paquete y asignarle su valor en un cuerpo de paquete. La ventaja que proporciona el cuerpo de paquete es ocultar la implementación de la interfaz externa.

Ejemplos:

Vamos a crear un **package body** para el paquete **pkg_constants1** y ponemos el valor de la constante **num_selected** en el cuerpo del paquete:

```
package pkg_constants1 is
    constant num_selected : integer;
end package;
```

```

package body pkg_constants1 is

    constant num_selected : integer := 2;

end package body;

```

Figura 15. Código del package body para pkg_constants1

La declaración de la constante tiene que seguir apareciendo en la *declaración de paquete*; veamos que ocurre si la quitamos y dejamos la constante solo en el *cuerpo del paquete*, así:

```

package pkg_constants1 is

    --constant num_selected : integer;

end package;

package body pkg_constants1 is

    constant num_selected : integer := 2;

end package body;

```

Figura 16. pkg_constants1 sin la declaración de num_selected

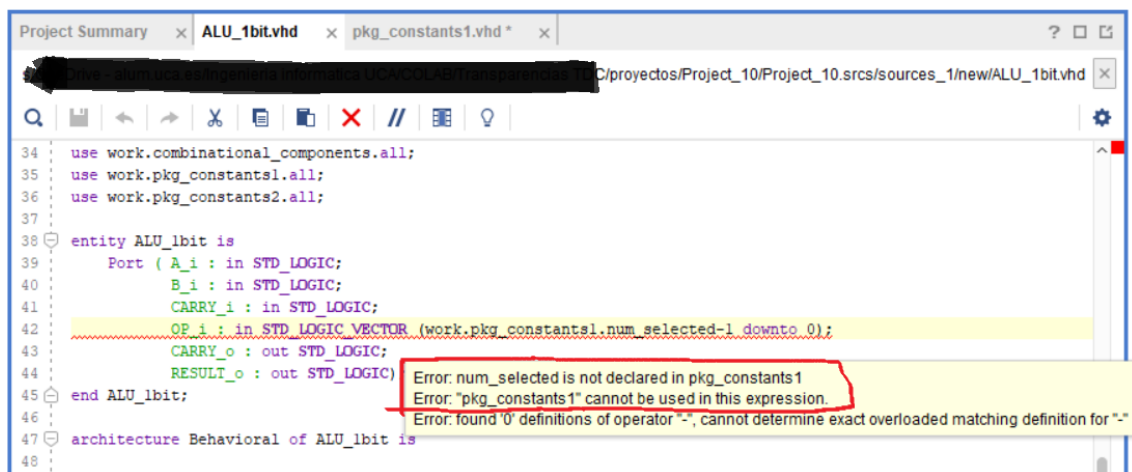


Figura 17. Error al eliminar la declaración de la constante

Como podemos ver en la Figura 17 el IDE nos informa de que la constante no ha sido declarada, esto se produce porque el analizador de VHDL siempre analiza primero la declaración de paquete para determinar que elementos forman el paquete y considera el cuerpo de un paquete como una unidad separada que

tiene una dependencia con la declaración de paquete que se analizará cuando se resuelvan las referencias a los ítem del paquete. Resumiendo:

*Todo lo que vaya a implementarse/definirse en un cuerpo de paquete (**package declaration**) tiene que estar declarado en su declaración de paquete (**package body**)*

Bibliotecas

En Vivado® cada archivo de código reside en una biblioteca. VHDL especifica dos tipos de biblioteca:

- Biblioteca de trabajo (*working library*): referencia a la biblioteca en la que la herramienta de diseño utilizada pone los módulos analizados y es la que hemos utilizado hasta ahora mediante **work**. En Vivado® la biblioteca por defecto es **xil_defaultlib**.
- Biblioteca de recursos (*resource library*): son las bibliotecas definidas por el usuario, la forma en la que se crean o configuran depende de la herramienta de diseño que utilicemos.

Ver las bibliotecas de un proyecto en Vivado®

Cuando abrimos un proyecto, en la ventana **Sources** si hacemos clic en un fichero de código aparecerá en la ventana **Source File Properties** las propiedades del fichero; la propiedad **Library** indica la biblioteca donde Vivado® coloca el módulo. Podemos verlo en la *Figura 18*.

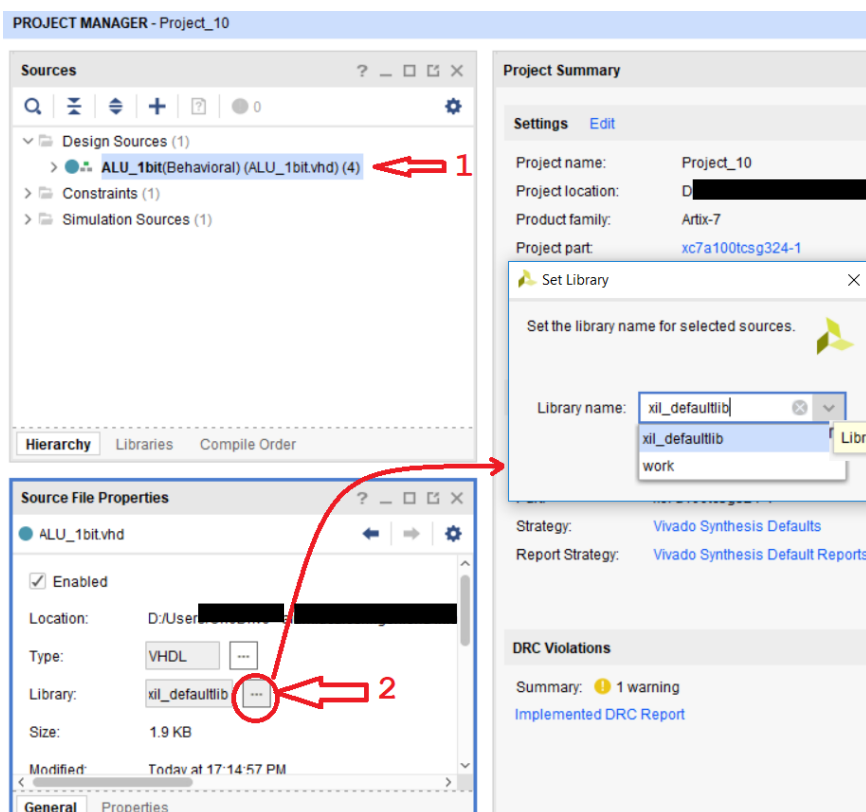


Figura 18. Propiedad library de un fichero

En la ventana **Sources** tenemos tres vistas diferentes la primera **Hierarchy** nos muestra la jerarquía de nuestro diseño y en la segunda que es **Libraries** nos muestra la organización de los ficheros de código por bibliotecas, podemos ver en la *Figura 19* como ahora mismo todos los ficheros están en la biblioteca por defecto, el orden en el que aparecen de arriba hacia abajo es el orden en el que se van a analizar los módulos tanto en simulación como en síntesis.

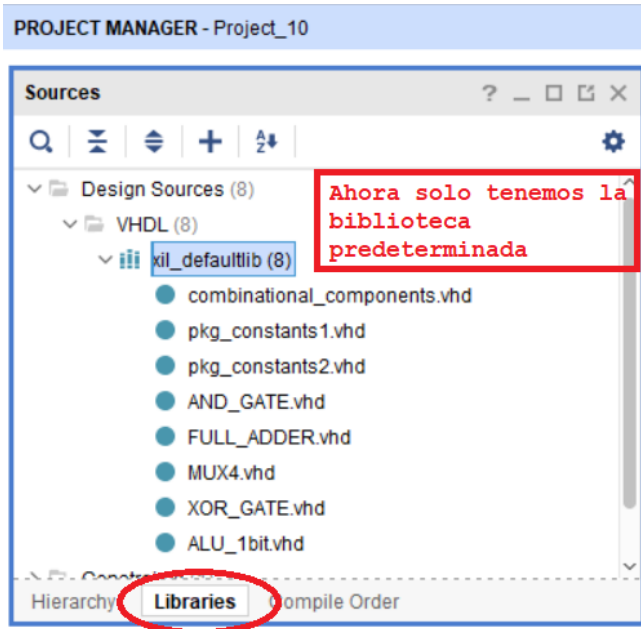


Figura 19. Ficheros en cada biblioteca

Asignar una biblioteca

Tenemos dos formas de asignar la biblioteca donde queremos que resida un fichero fuente:

1. En las propiedades del fichero en la ventana **Source File Properties**
2. Cuando creamos o añadimos un fichero desde la ventana **Add Sources**

El primer método lo aplicamos para cambiar la biblioteca de un fichero que ya está añadido al proyecto. El segundo para los ficheros que vayamos a crear dentro del proyecto o ficheros que estén en otra ubicación.

Cambiar la biblioteca de un fichero de código en nuestro proyecto

Para cambiar la biblioteca de un fichero que a está en nuestro proyecto seguimos los pasos de la *Figura 18* y una vez en la ventana **Set Library** seleccionamos la biblioteca del desplegable o **escribimos el nombre de la biblioteca para crearla**. En la *Figura 20* creamos una biblioteca que se llama *lib_mycomp* y ponemos el fichero *combinational_components.vhd* en ella.

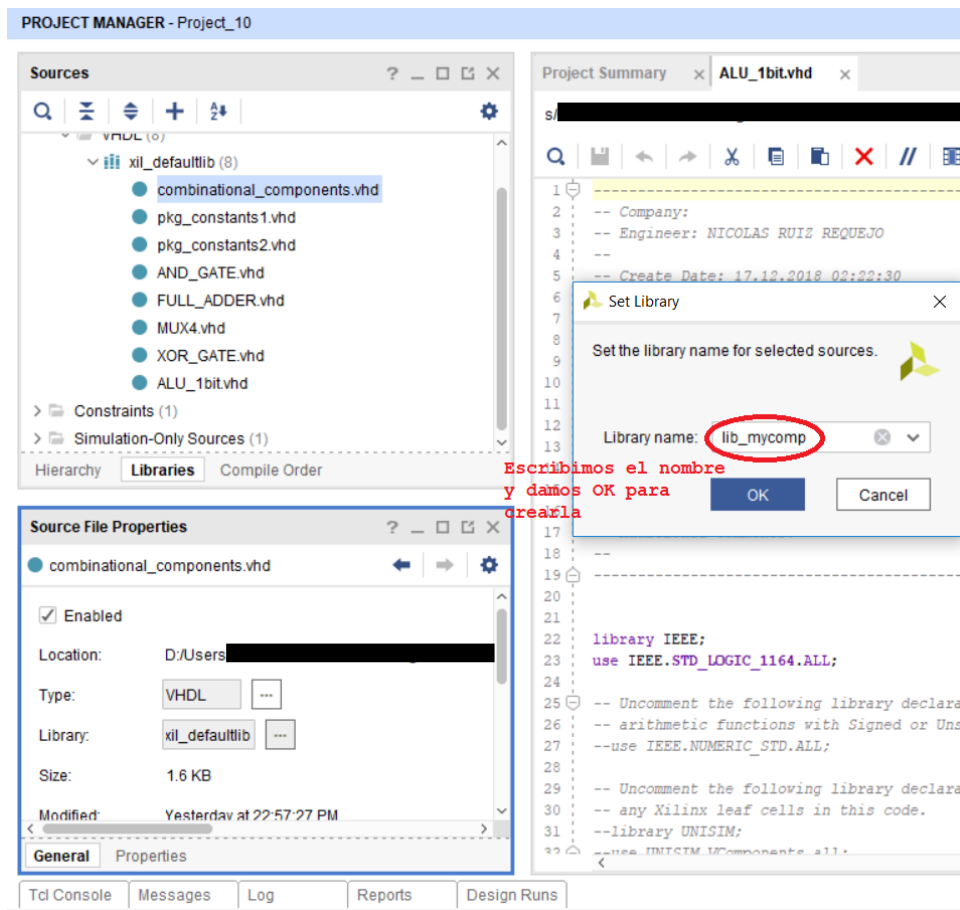


Figura 20. Crear biblioteca lib_mycomp

Ahora en **Sources** aparece lib_mycomp, pero aparece como “unreferenced” como podemos ver en la Figura 21. Esto es porque ningún otro módulo del proyecto está referenciando a combinational_components.vhd.

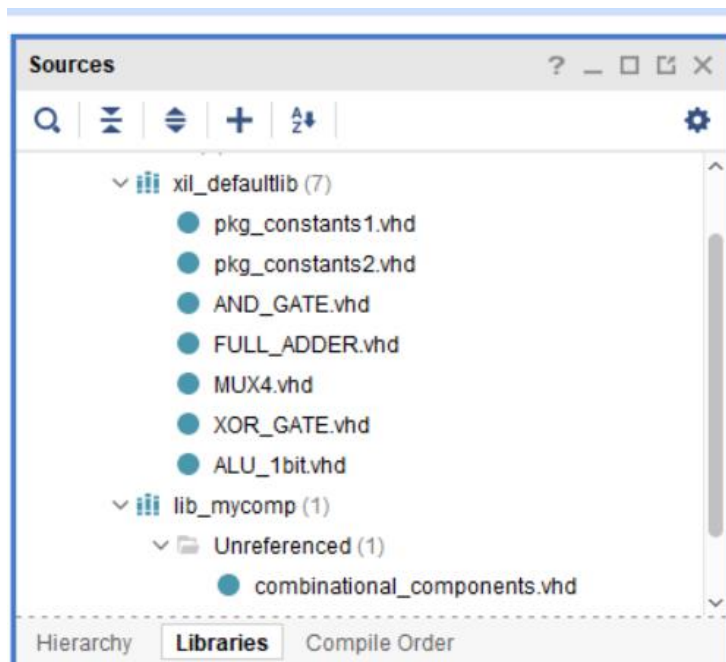


Figura 21. lib_mycomp está “unreferenced”

Para referenciarlo tenemos que incluir la biblioteca en ALU_1bit.vhd con la sentencia:

```
library lib_mycomp;
```

Seguida de una sentencia **use**:

```
use lib_mycomp.combinational_components.all;
```

Añadiendo la sentencia **library** ya podremos usar los elementos que están colocados en la biblioteca correspondiente, ya sea con cláusulas **use** o usando el *selected name* del elemento.

Vamos a cambiar a la biblioteca *lib_mycomp* el resto de paquetes y módulos que tienen dependencia con el paquete *combinational_components*, se puede ver en la Figura 22.

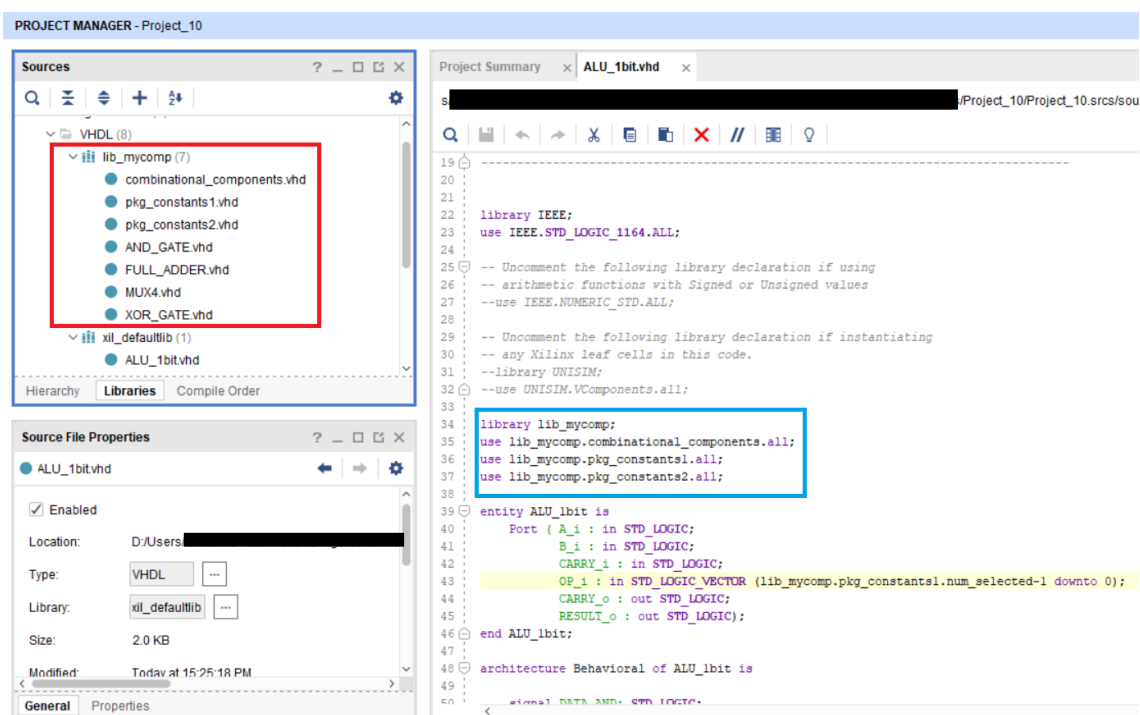


Figura 22. Cambiando bibliotecas a *lib_mycomp* en las propiedades

Crear una biblioteca reutilizable

En el ejemplo anterior los ficheros de cada componente se habían copiado dentro del proyecto, ahora vamos a crear una biblioteca que sea reutilizable de manera que no necesitemos copiar todos los ficheros a un proyecto donde queramos usar la biblioteca.

Para este ejemplo necesitamos primero:

1. Copiar las fuentes que se incluyen en la biblioteca *lib_mycomp* a un directorio en el sistema de archivos de la computadora como en la Figura 23:

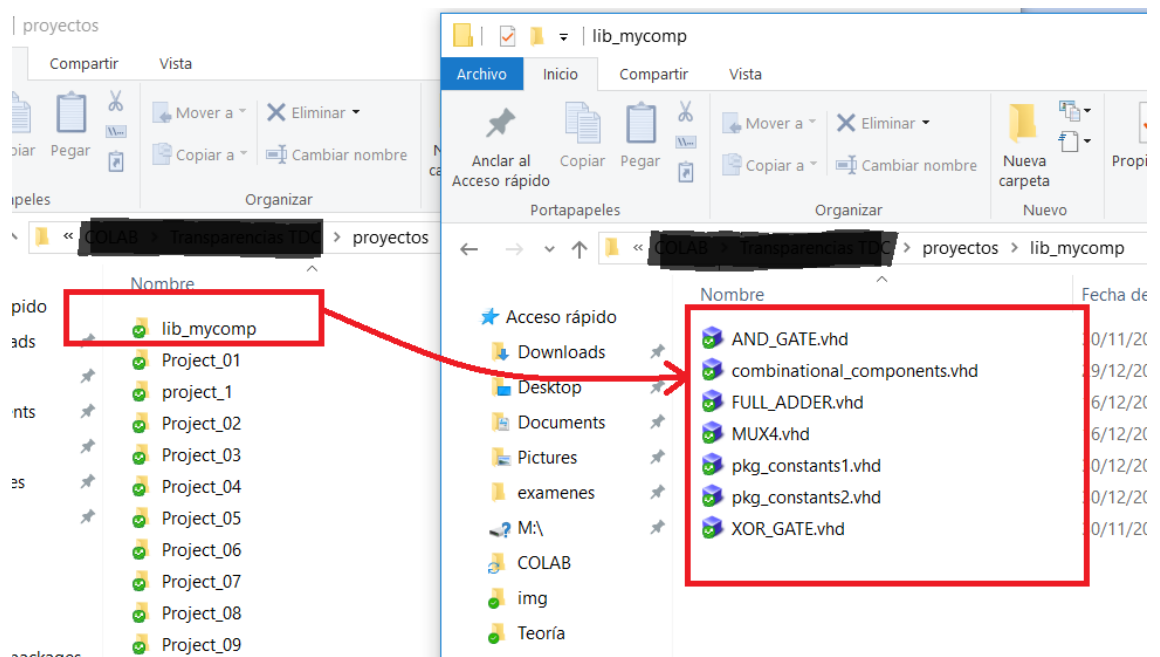


Figura 23. Crear directorio para las fuentes de la biblioteca lib_mycomp

2. Eliminar las fuentes en el proyecto y dejar únicamente **ALU_1bit.vhd**:

Seleccionando las fuentes en la ventana **Sources > libraries** que teníamos ya en el proyecto 10 de los pasos anteriores (click en el primer archivo + pulsar tecla mayus y click en el último) y haciendo click en el botón de suprimir (aspa roja) y aceptando. Se puede ver en la Figura 24.

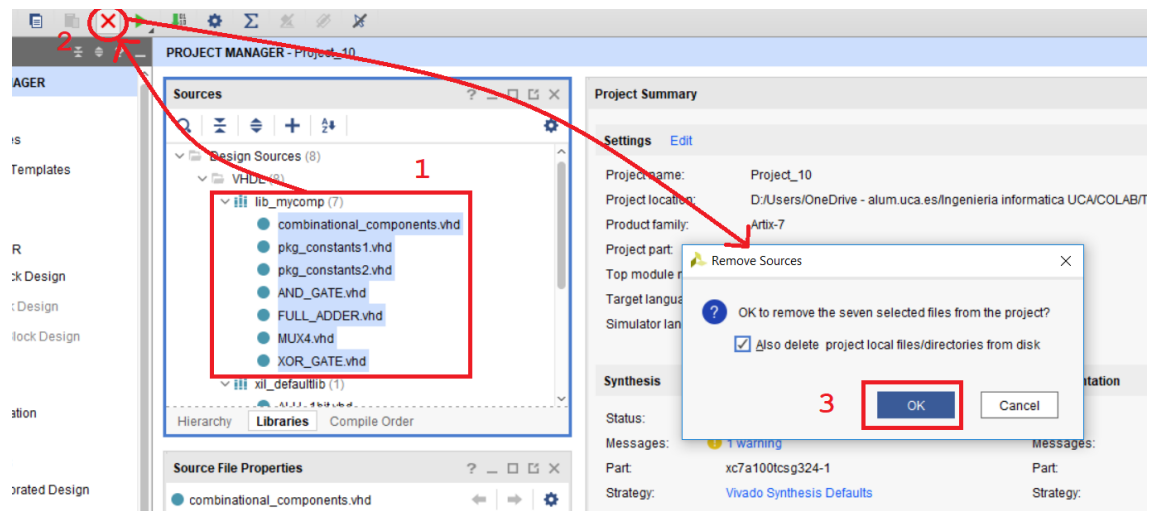


Figura 24. Eliminar las fuentes de los pasos anteriores

Ahora para importar la biblioteca damos en la ventana **Sources** al botón de **Add Sources**, en la ventana en **Add or create design sources** y en siguiente. Y en la ventana que aparece pulsamos en **Add Directories** y buscamos el directorio con las fuentes de nuestra biblioteca **lib_mycomp** y lo seleccionamos. Ver Figura 25.

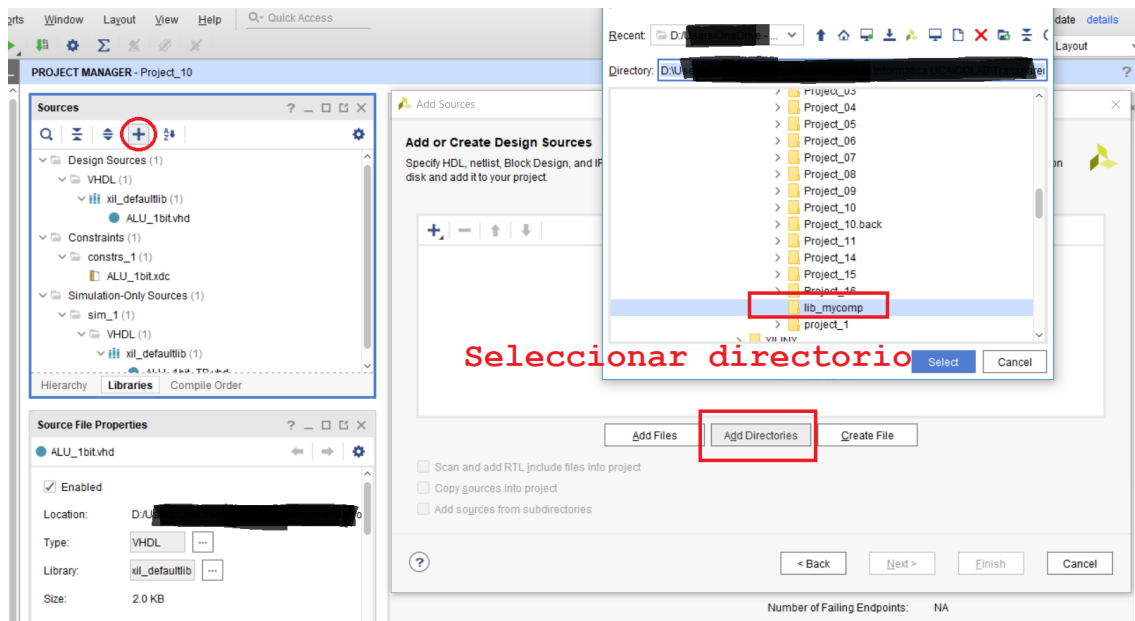


Figura 25. Seleccionar directorio de la biblioteca

Ahora ponemos el nombre de la biblioteca que estamos importando en la casilla **Library**, deseleccionamos la opción **Copy sources into project** y dejamos marcada **Add sources from subdirectories**. Ver Figura 26.

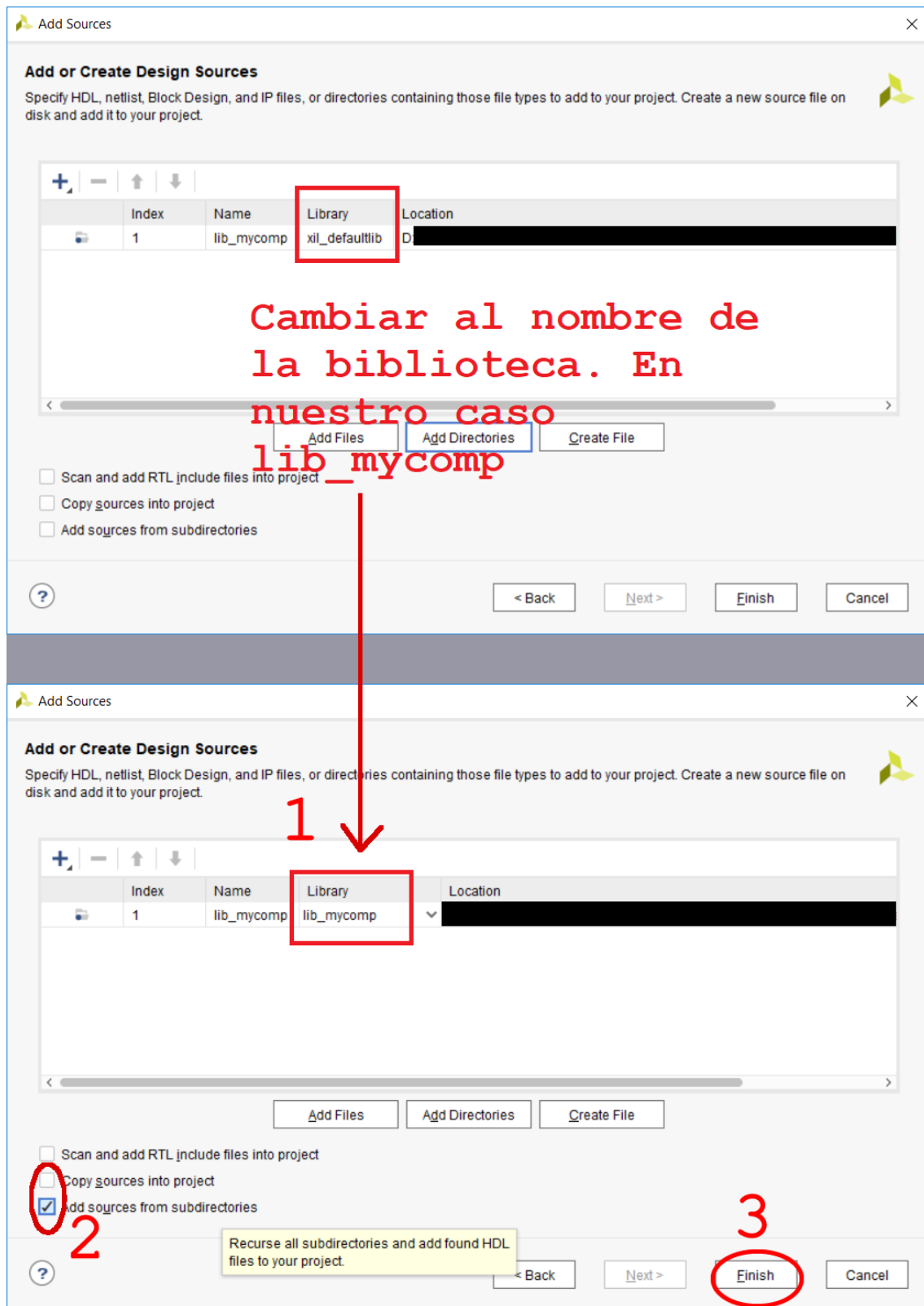


Figura 26. Añadir directorio de las fuentes

Podremos ver en la ventana **Sources** como aparecen todas las fuentes de la biblioteca **lib_mycomp** en nuestro proyecto. Ver Figura 27.

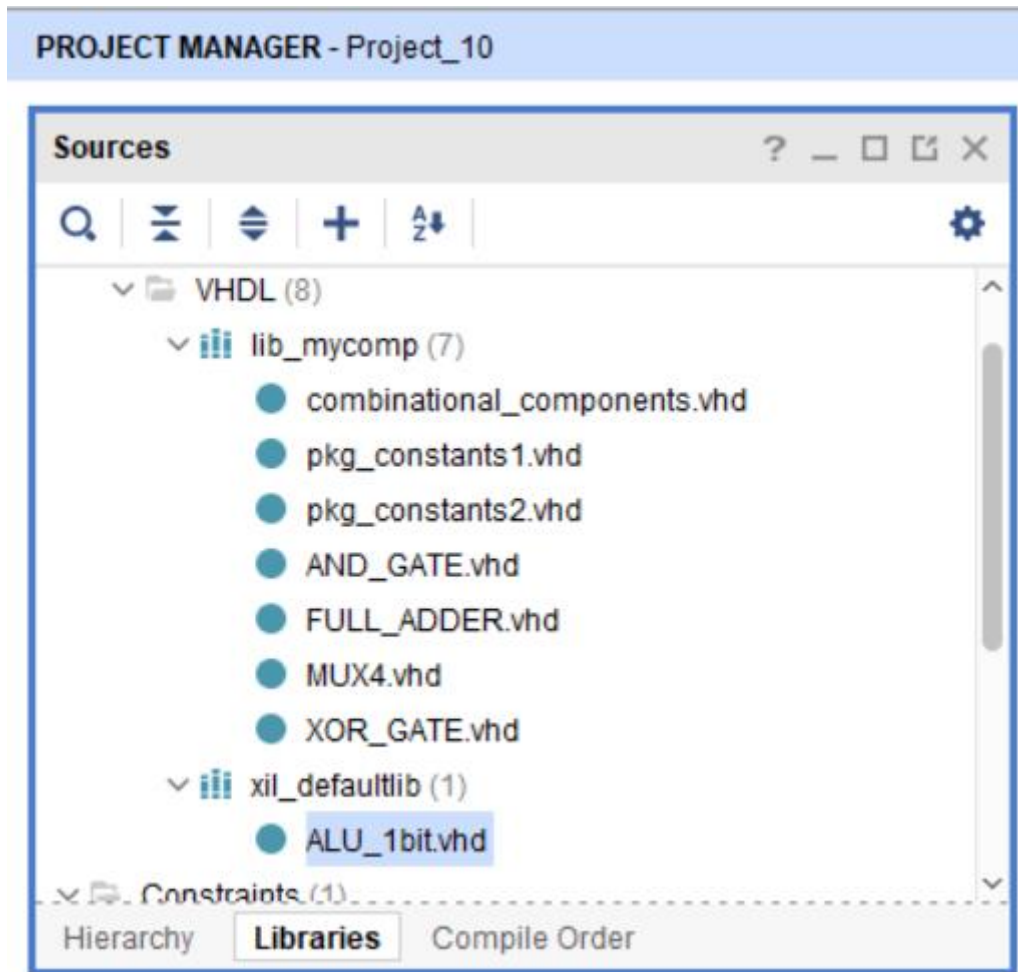


Figura 27. *lib_mycomp* importada satisfactoriamente

El código de **ALU_1bit.vhd** haciendo uso de la biblioteca es el mismo que el de la Figura 22 y se lista al completo en la Figura 28.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library lib_mycomp;
use lib_mycomp.combinational_components.all;
use lib_mycomp.pkg_constants1.all;
use lib_mycomp.pkg_constants2.all;

entity ALU_1bit is
    Port ( A_i : in STD_LOGIC;
          B_i : in STD_LOGIC;
          CARRY_i : in STD_LOGIC;
          OP_i : in STD_LOGIC_VECTOR
            (lib_mycomp.pkg_constants1.num_selected-1 downto 0);
          CARRY_o : out STD_LOGIC;
          RESULT_o : out STD_LOGIC);
end ALU_1bit;

architecture Behavioral of ALU_1bit is

    signal DATA_AND: STD_LOGIC;
    signal DATA_XOR: STD_LOGIC;
    signal DATA_ADD: STD_LOGIC;
    signal DATA_MOV: STD_LOGIC;

begin

    andgate: AND_GATE
        port map(A_i => A_i ,B_i => B_i, Z_o => DATA_AND);

    xorgate: XOR_GATE generic map(WIDTH=>1)
        port map(A_i(0) => A_i, B_i(0) => B_i, Z_i(0) => DATA_XOR);

    add: FULL_ADDER port map(A_i => A_i, B_i => B_i,
        CARRY_i => CARRY_i,
        SUM_o=> DATA_ADD,
        CARRY_o=> CARRY_o);

    DATA_MOV <= A_i;

    mux: MUX4 generic map(WIDTH=>1)
        port map(A_i(0) => DATA_AND, B_i(0) => DATA_XOR, C_i(0) => DATA_ADD,
            D_i(0) => DATA_MOV, SEL_i => OP_i, Z_o(0) => RESULT_o);

end Behavioral;

```

Figura 28. Código de ALU_1bit.vhd incluyendo la biblioteca lib_mycomp

Referencias

- Ashenden, P. J. (2008). *The Designer's Guide to VHDL*. Elsevier Science & Technology.
- IEEE. (26 de Septiembre de 2008). IEEE Std 1076-2008. IEEE Standard VHDL Language.
- StackExchange. (s.f.). *how-do-i-build-and-use-my-own-vhdl-library*. Obtenido de <https://electronics.stackexchange.com/questions/162019>
- Xilinx. (2 de 6 de 2014). *AR# 52575 Vivado - How do I add HDL source files to a specific library in Vivado?* Obtenido de <https://www.xilinx.com/support/answers/52575.html>
- Xilinx. (6 de Junio de 2018). UG895. Vivado Design Suite - System-Level Design Entry.

