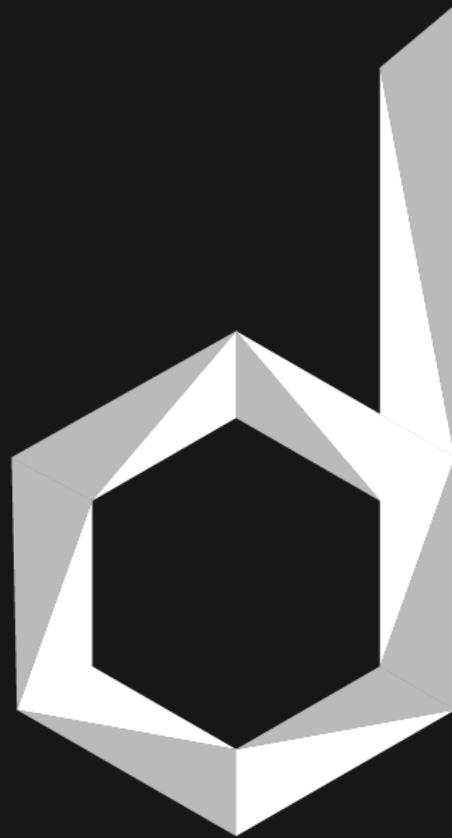


INGENIERÍA MECATRÓNICA



DI\_CERO

DIEGO CERVANTES RODRÍGUEZ

ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE VHDL Y VERILOG

XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

Introducción al Lenguaje VHDL

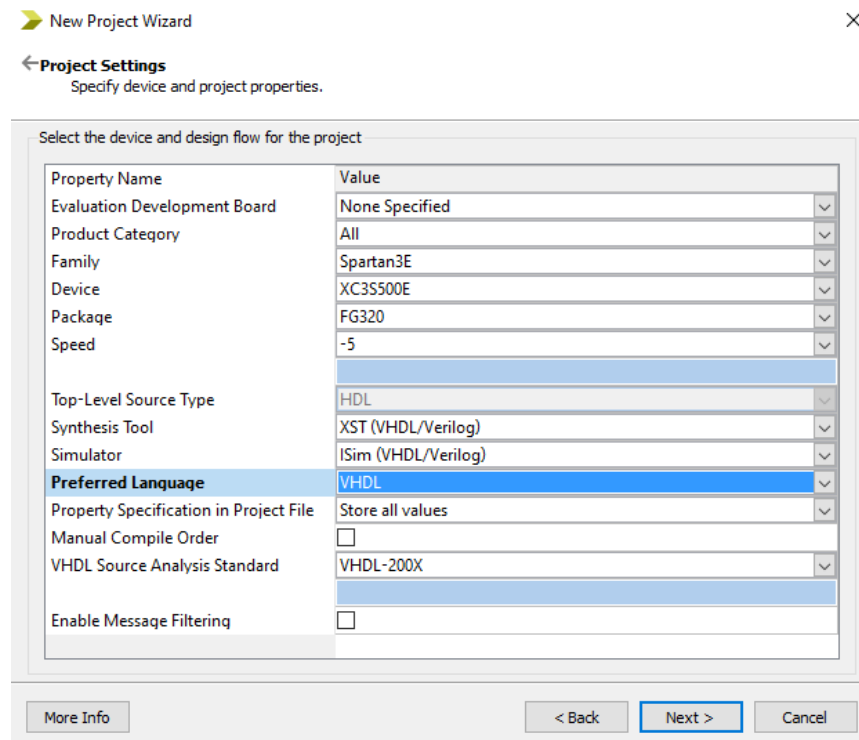
# Contenido

VHDL: Lenguaje de descripción de Hardware (HDL).....	2
Entidad: Declaración de Entradas/Salidas .....	4
Arquitectura: Acciones a ejecutar de las Entradas/Salidas .....	8
Código VHDL Completo para Circuitos Lógicos.....	9
Implementación: Checar sintaxis .....	13
Implementación: Archivo UCF .....	15
Implementación: Adept .....	22

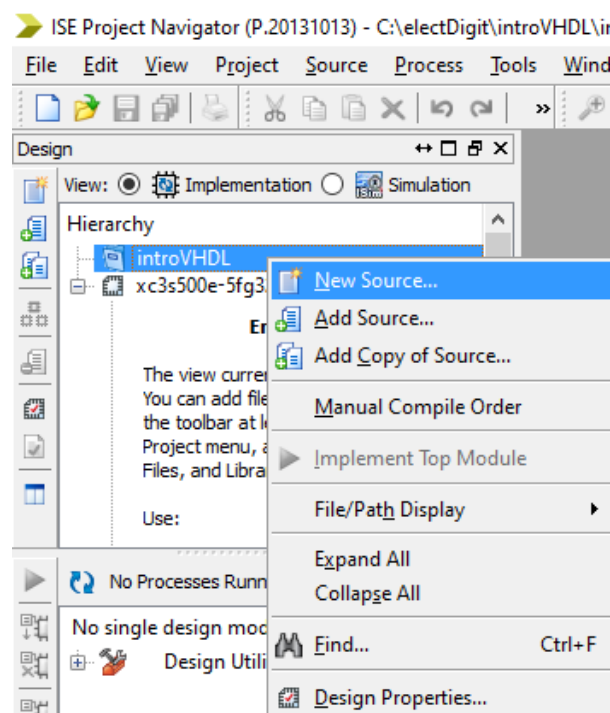


# VHDL: Lenguaje de descripción de Hardware (HDL)

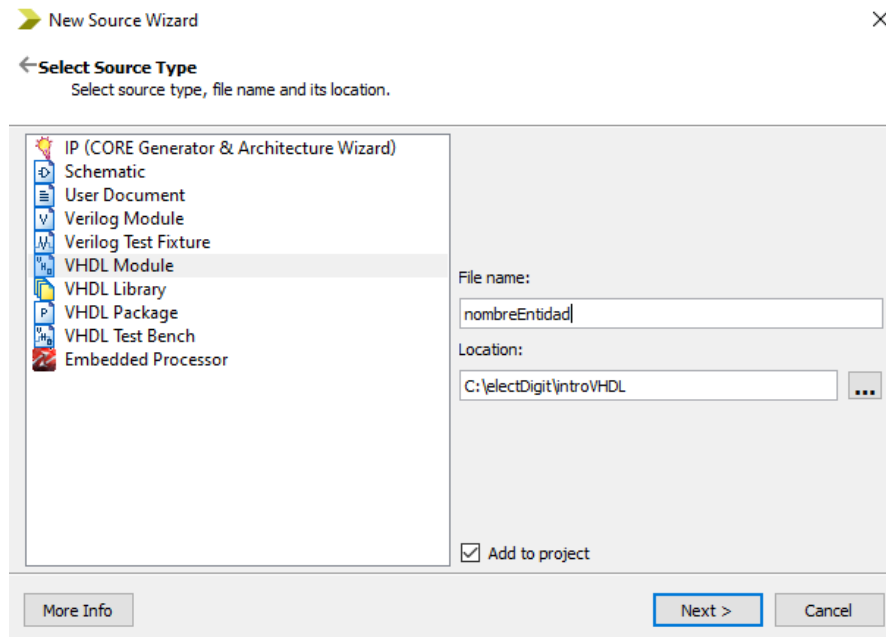
Cuando cree un proyecto nuevo debo indicar que el lenguaje que estoy usando es VHDL y es importante que las especificaciones sean las siguientes:



Después debo hacer es dar clic derecho a la carpeta de mi proyecto y seleccionar New Source.

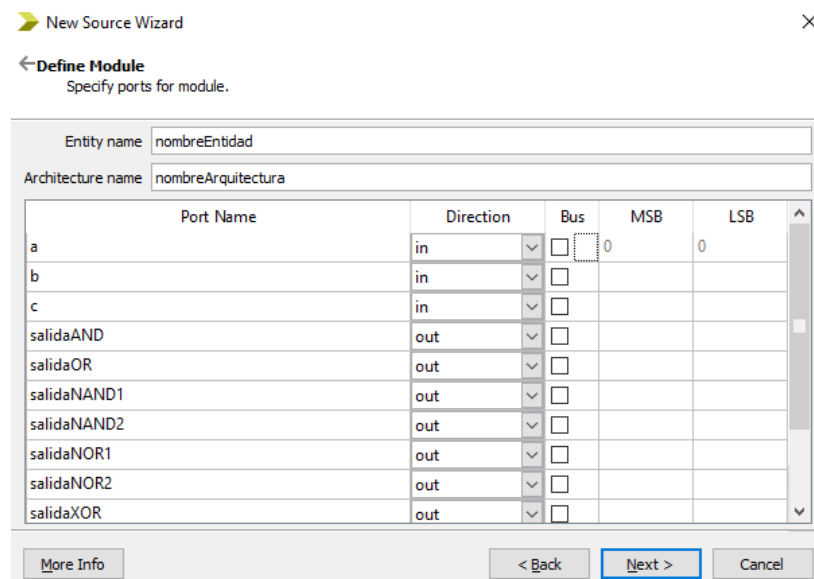


Dentro de la ventana de New Source debo elegir el módulo (así se le llama a todo el código escrito en un lenguaje descriptivo de hardware (HDL) ya sea VHDL o Verilog) del lenguaje de programación que haya elegido o sea **VHDL Module**, esto sirve para que pueda declarar mis entradas y salidas de una manera más amigable y que esta parte del código se escriba por sí sola, evitándome así errores de sintaxis (que escriba mal una palabra).



**En VHDL, el nombre del módulo será igual al nombre de la entidad.**

En la ventana consecuente puedo renombrar a la entidad (que es la parte del código donde se declaran las entradas/salidas) y debo nombrar a las variables que representarán mis entradas y/o salidas e indicar de que tipo es cada una, también debo nombrar a una parte del código llamada arquitectura que posteriormente les dirá a las entradas/salidas declaradas qué hacer.



## Entidad: Declaración de Entradas/Salidas

La función de cada módulo de código es describir lo que puede hacer un circuito integrado con un número de entradas y salidas. En VHDL estas entradas y salidas son declaradas en una parte del código llamada **entidad**, se usará la palabra reservada **in** para declarar las entradas y la palabra reservada **out** para declarar las salidas, todas usando la biblioteca **STD\_LOGIC**; y dentro de un elemento llamado **Port** (); donde se declaran los puertos (entradas y salidas).

Es importante mencionar que **en la última salida o entrada declarada no se debe poner un punto y coma al final ; pero en las demás sí**. Las entradas/salidas son siempre digitales por lo que pueden ser:

- **De un solo bit 0 o 1:** Ese bit funciona como una variable y puede recibir o entregar ceros lógicos, unos lógicos o ninguno de los dos (llamado Z o alta impedancia).

Lo que hace el 1 lógico es entregar en la salida un voltaje de 3.3V, lo que hace el 0 lógico es entregar en la salida un voltaje de 0V y lo que hace Z o alta impedancia es asignar un valor de voltaje que se encuentre entre el 0 lógico y el 1 lógico, por lo que en la salida no habrá nada.

--Los comentarios se declaran con dos guiones seguidos.

--La sintaxis en el código de la entidad para declarar las entradas y salidas en

--forma de bit es la siguiente:

**entity** nombreEntidad **is**

**Port** ( nombreVariableEntrada : **in** **STD\_LOGIC**;

        nombreVariableSalida1 : **out** **STD\_LOGIC**;

        nombreVariableSalida2 : **out** **STD\_LOGIC**

    );

**end** nombreEntidad;

- **Vectores 000000, 111111111111, 101, 01, etc.:** Lo que hace el vector es almacenar una variable completa que pueda recibir o entregar un número cualquiera de bits mayor a 1, cada bit individualmente puede recibir o entregar diferentes valores de ceros lógicos, unos lógicos o altas impedancias para crear diferentes números binarios completos, el vector declarado siempre entregará el mismo número de bits. Al bit de hasta la izquierda se le dice **bit más significativo porque es el que más valor tiene** y al de hasta la derecha se le llama **bit menos significativo porque es el que menos valor tiene**, en el sistema binario mientras más me haga a la izquierda del número binario más vale, no importando el número de bits que formen al vector **000000, 111111111111, 101 o 01**.

Para indicar el tamaño del vector y las coordenadas con las que voy a llamar a cada uno de sus bits se usan las sintaxis (**n downto 0**); o (**0 to n**), para indicar el tamaño del vector es indiferente usar una o la otra ya que en ambas se hace un conteo desde la coordenada n hasta la 0 y viceversa, incluyendo en mi conteo tanto a la coordenada n como a la 0, la diferencia entre usar una sintaxis o la otra es que con cada una deberé usar una coordenada diferente para llamar a cada bit, ya que con (**n downto 0**); el **bit más significativo** sería convocado con la coordenada **n** y el **bit menos**

**significativo** sería convocado con la coordenada **0** y con (0 to n); el **bit más significativo** sería convocado con la coordenada **0** y el **bit menos significativo** sería convocado con la coordenada **n**.

--La sintaxis en el código de la entidad para declarar las entradas y salidas en  
--forma de vector (número binario) es la siguiente:

**entity** nombreEntidad **is**

```
Port ( nombreVariableEntrada : in STD_LOGIC_VECTOR (2 downto 0);  
      nombreVariableSalida : out STD_LOGIC (0 to 1)  
      );
```

**end** nombreEntidad;

--Para crear vectores puedo usar la sintaxis (2 downto 0) o la sintaxis (0 to 2)

--ambas sirven para indicar cuantos bits tendrá mi vector y la coordenada con la

--que cada bit es llamado, desde el bit más significativo al menos significativo,

--leídos de izquierda a derecha, en los dos ejemplos del código anterior el vector

--tendrá 3 bits porque se cuenta de la siguiente manera:

--2,1,0 para el downto que formó un vector de 3 bits y donde la coordenada del bit

--más significativo es la de 2 y la del menos significativo es la de 0.

--0,1 para la sintaxis to que formó un vector de 2 bits y donde la coordenada del bit

--más significativo es la de 0 y la del menos significativo es la de 1.

Usualmente se usará una combinación de entradas en forma de bit y en forma de vector para programar la FPGA.

Para no tener que escribir el código anterior directamente, es que se usa la pantalla previamente mencionada donde podemos introducir directamente algunos aspectos como:


- Poner el **nombre de mis entradas** junto con la opción de **Direction in**.
- Indicar el **nombre de mis salidas** junto con la opción de **Direction out**.

Port Name	Direction
a	in
b	in
c	in
salidaAND	out
salidaOR	out
salidaNAND1	out
salidaNAND2	out
salidaNOR1	out
salidaNOR2	out
salidaXOR	out

- Elegir si están serán de solo un bit o serán vectores, esto lo hago dando **clic en el checkbox que dice Bus si la entrada o salida es un vector** (ya que Bus en los dispositivos digitales son varios cables que transmiten los ceros o unos de cada bit del número binario correspondiente a los dispositivos electrónicos)
  - Si mi entrada/salida es un vector debo indicar:
    - La coordenada de su **bit más significativo en donde dice MSB** (Most Significant Bit).
    - La coordenada de su **bit menos significativo donde dice LSB** (Least Significant Bit).
    - Ambas en conjunto indicarán el tamaño del vector y sus coordenadas.

Bus	MSB	LSB
<input checked="" type="checkbox"/>	2	0

Hasta arriba de la ventana debo indicar el nombre de la entidad y arquitectura (estos nombres sí aparecerán en el código de VHDL). Finalmente debo dar clic al botón de Next.

 New Source Wizard ×

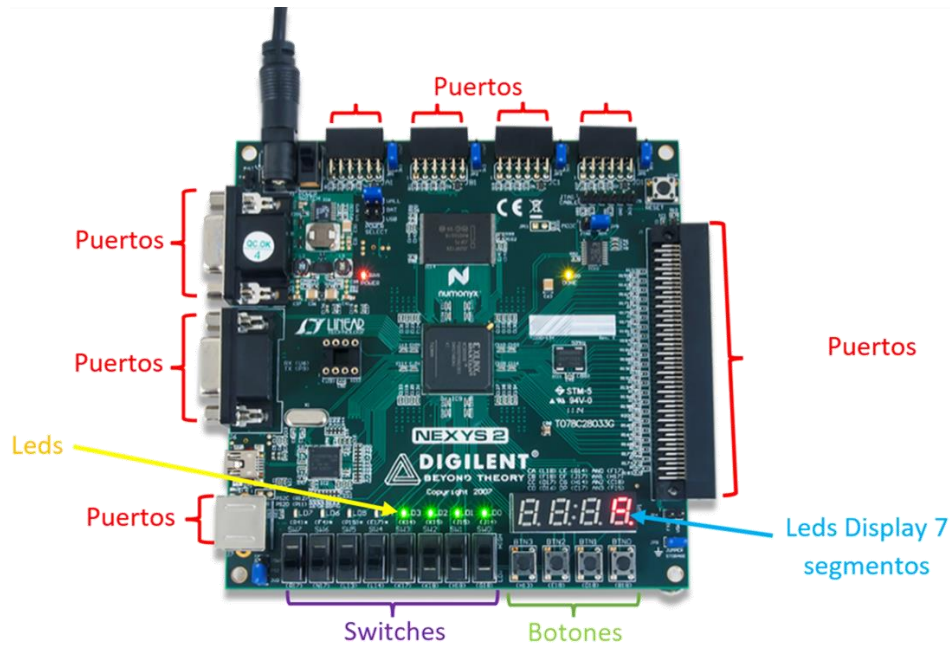
← **Define Module**  
Specify ports for module.

Entity name		nombreEntidad			
Architecture name		nombreArquitectura			
Port Name	Direction	Bus	MSB	LSB	
a	in	<input checked="" type="checkbox"/>	0	0	
b	in	<input type="checkbox"/>			
c	in	<input type="checkbox"/>			
salidaAND	out	<input type="checkbox"/>			
salidaOR	out	<input type="checkbox"/>			
salidaNAND1	out	<input type="checkbox"/>			
salidaNAND2	out	<input type="checkbox"/>			
salidaNOR1	out	<input type="checkbox"/>			
salidaNOR2	out	<input type="checkbox"/>			
salidaXOR	out	<input type="checkbox"/>			

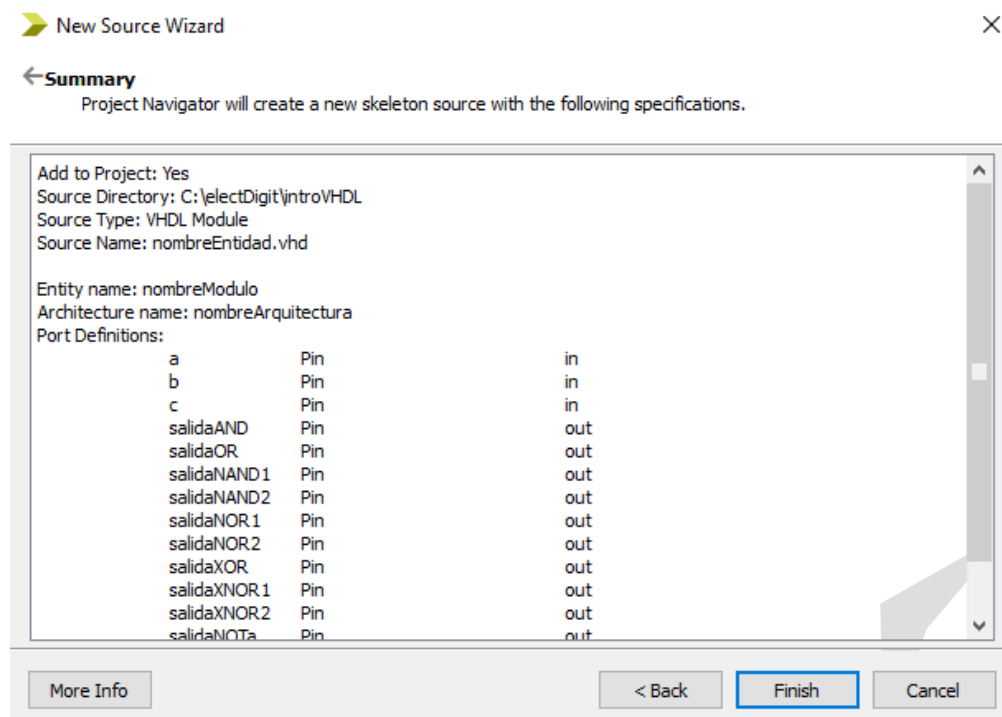
Las entradas y salidas pueden ser los siguientes elementos electrónicos pertenecientes a la tarjeta de desarrollo NEXYS 2:

- Puertos (agujeros en donde puedo conectar jumpers o cables a diferentes circuitos, PCBs o dispositivos externos).

- Leds.
- Leds de los displays de 7 segmentos.
- Switches.
- Botones (push buttons).



Después me saldrá una pantalla indicándome cuales son las entradas y salidas declaradas para que compruebe que no haya habido ningún error, ya que esté satisfecho le daré clic en Finish y me **creará mi mi archivo con código VHDL**.





Este es el código creado por el módulo ya con la entidad que contiene las entradas y salidas.

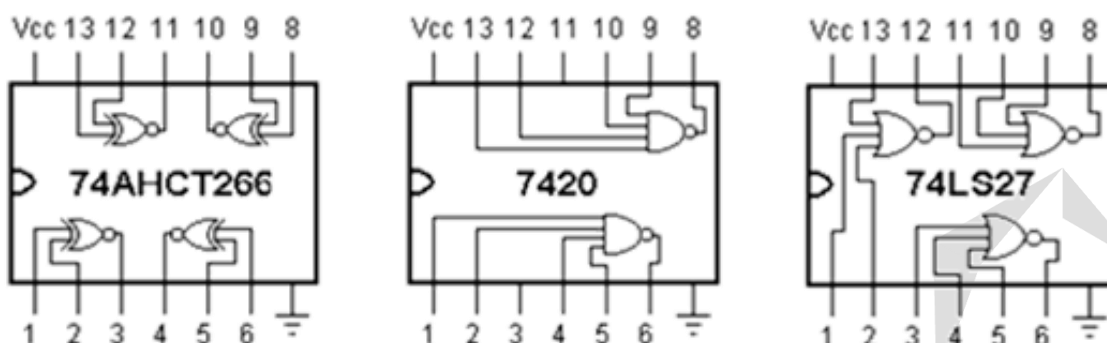
```
1  --Los comentarios se ponen con dos guiones seguidos
2  --El lenguaje VHDL no distingue entre mayúsculas y minúsculas
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5  --Las librerías IEEE y STD_LOGIC_1164 se declaran para poder usar el lenguaje de programación VHDL
6
7  --La ENTIDAD es donde declaro mis entradas y salidas, empieza con la palabra reservada is y termina con end, las
8  --entradas y salidas se ponen dentro de Port(); y usan las palabras reservadas in y out aparte de la librería STD_LOGIC;
9  entity nombreModulo is
10     Port ( a : in  STD_LOGIC;
11           b : in  STD_LOGIC;
12           c : in  STD_LOGIC;
13           salidaAND : out  STD_LOGIC;
14           salidaOR : out  STD_LOGIC;
15           salidaNAND1 : out  STD_LOGIC;
16           salidaNAND2 : out  STD_LOGIC;
17           salidaNOR1 : out  STD_LOGIC;
18           salidaNOR2 : out  STD_LOGIC;
19           salidaXOR : out  STD_LOGIC;
20           salidaXNOR1 : out  STD_LOGIC;
21           salidaXNOR2 : out  STD_LOGIC;
22           salidaNOTa : out  STD_LOGIC);
23 end nombreModulo;
24 --La ARQUITECTURA es donde declaro qué harán mis entradas y salidas, tiene el nombre del modulo, empieza con begin y
25 --termina con end y el nombre de la arquitectura
26 architecture nombreArquitectura of nombreModulo is
27 begin
28 |
29 end nombreArquitectura;
```

## Arquitectura: Acciones a ejecutar de las Entradas/Salidas

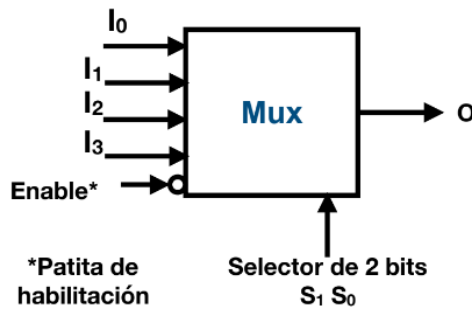
La función de cada módulo de código ya sea de VHDL o Verilog es describir las conexiones que hará la FPGA internamente para crear un diseño de circuito integrado personalizado que ejecutará una acción determinada con un número de entradas y salidas declaradas. En VHDL la parte del código donde se describe la acción que ejecutarán esas entradas y salidas es llamada **arquitectura**, en ella puedo ejecutar operaciones lógicas AND, OR, NOT, NAND, NOR, XOR y XNOR, por medio de condicionales o bucles hacer Comparadores (que comparan dos señales y dicen cuál es mayor, menor o igual a la otra), Multiplexores (que reciben varias entradas y dejan pasar solo una), Demultiplexores (que reciben una entrada y dejan pasar varias), sumadores, restadores, multiplicadores o divisores de números binarios, encender displays de 7 segmentos, usar pantallas LCD, motores a pasos, flip flops, etc.

Todo esto me sirve para poder hacer uso de la electrónica digital y al final obtener circuitos de la siguiente naturaleza:

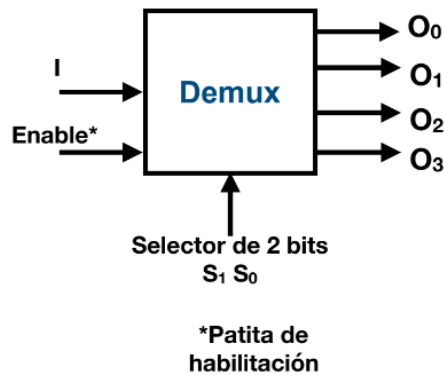
### Circuitos lógicos:



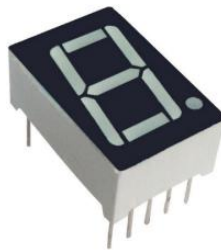
### Multiplexor o MUX:



### Demultiplexor o DEMUX:



Mostrar datos en displays de 7 segmentos, ya sea pertenecientes a la tarjeta o externos:



Se aprenderá a usar todas estas herramientas y dispositivos alrededor del curso, pero por lo mientras eso es todo lo que se debe saber para entender a grandes rasgos el lenguaje VHDL, para entender los conceptos se deberá trabajar con los códigos hechos, los apuntes escritos en papel y los documentos de Word.

En este primer ejemplo lo que haremos es usar todas las compuertas lógicas con 3 entradas de 1 bit.

## Código VHDL Completo

VHDL es un lenguaje que **no es case sensitive**, osea que **no distingue entre mayúsculas y minúsculas**, además se asigna un valor a cualquier variable con los signos menor que e igual <= no importando si la variable es una entrada o una salida.

Las operaciones lógicas en VHDL se hacen con la misma expresión que las describe.

--Declaración de librerías, estas son usadas simplemente para poder utilizar el  
--lenguaje VHDL

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

--Entidad: Parte del código VHDL donde se declaran las entradas/salidas

entity nombreEntidad is

```
    Port ( a : in STD_LOGIC;  
          b : in STD_LOGIC;  
          c : in STD_LOGIC;  
          salidaAND : out STD_LOGIC;  
          salidaOR : out STD_LOGIC;  
          salidaNAND1 : out STD_LOGIC;  
          salidaNAND2 : out STD_LOGIC;  
          salidaNOR1 : out STD_LOGIC;  
          salidaNOR2 : out STD_LOGIC;  
          salidaXOR : out STD_LOGIC;  
          salidaXNOR1 : out STD_LOGIC;  
          salidaXNOR2 : out STD_LOGIC;  
          salidaNOTa : out STD_LOGIC  
    );
```

end nombreEntidad;

--Arquitectura: Parte del código VHDL donde se indican las operaciones que hará el  
--programa con las entradas/salidas, se escribe primero poniendo la palabra reservada  
--architecture, seguida de su nombre, la palabra reservada of, el nombre de la entidad y  
--finaliza con la palabra reservada is. El contenido de la arquitectura está delimitado por  
--las palabras reservadas begin y end más el nombre de la arquitectura.  
--Las operaciones más sencillas que puedo hacer son las operaciones lógicas.

architecture nombreArquitectura of nombreEntidad is

begin

--Las operaciones lógicas en VHDL se hacen con la misma expresión que las  
--describe

salidaAND <= a and b and c;

--En VHDL se asigna un valor a una variable con los signos <= no importando si  
--la variable es una entrada o una salida.

salidaOR <= a or b or c;

salidaNAND1 <= a nand b;

salidaNAND2 <= not (a and b and c);

--El operador NAND con más de 2 variables se debe hacer negando la operación  
--and entre las 3 o más variables



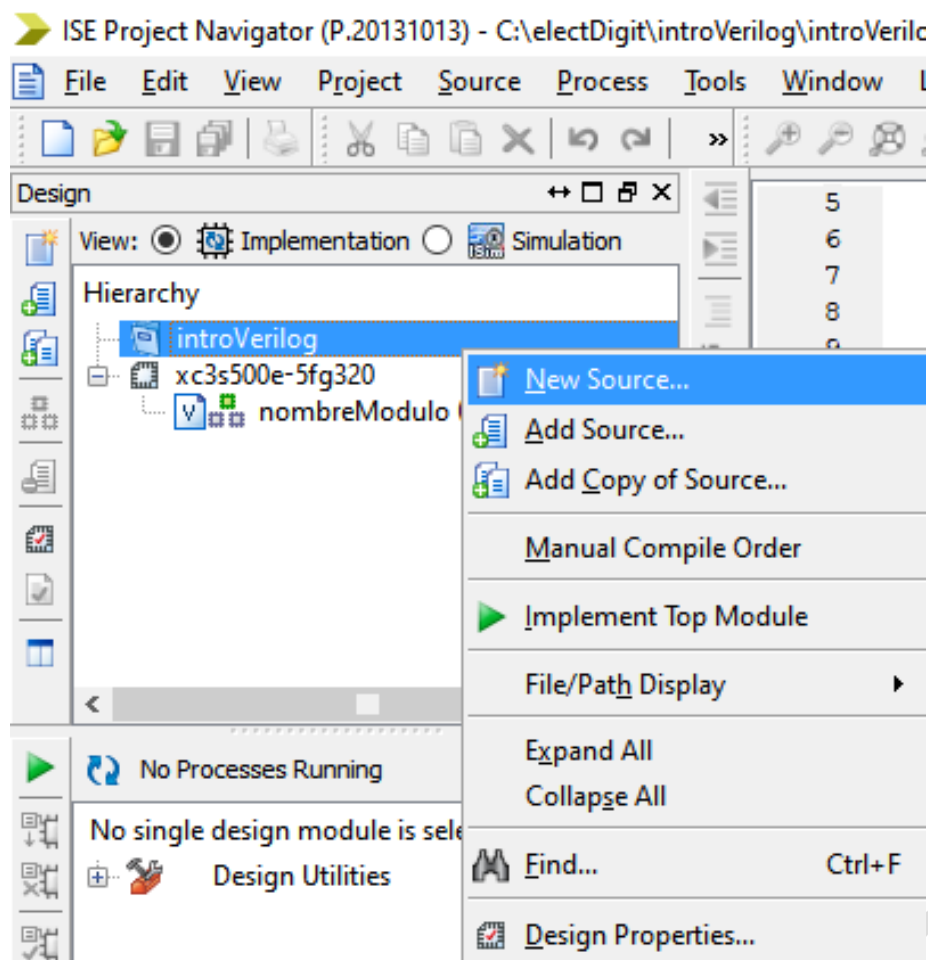
```

salidaNOR1  <= a nor b;
salidaNOR2  <= not (a or b or c);
--El operador NOR con más de 2 variables se debe hacer negando la operación or
--entre las 3 o más variables
salidaXOR   <= a xor b xor c;
salidaXNOR1 <= a xnor b;
salidaXNOR2 <= not (a xor b xor c);
--El operador XNOR con más de 2 variables se debe hacer negando la operación
--or entre las 3 o más variables
salidaNOTa  <= not a;
end nombreArquitectura;

```

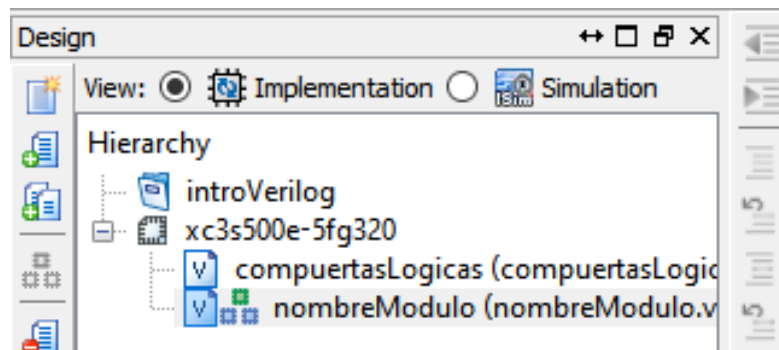
## Agregar Dos Módulos en un Mismo Proyecto

Para casos como estos donde quiero meter dos módulos en un mismo proyecto lo que debo hacer es dar clic derecho a la carpeta del proyecto y seleccionar New Source.

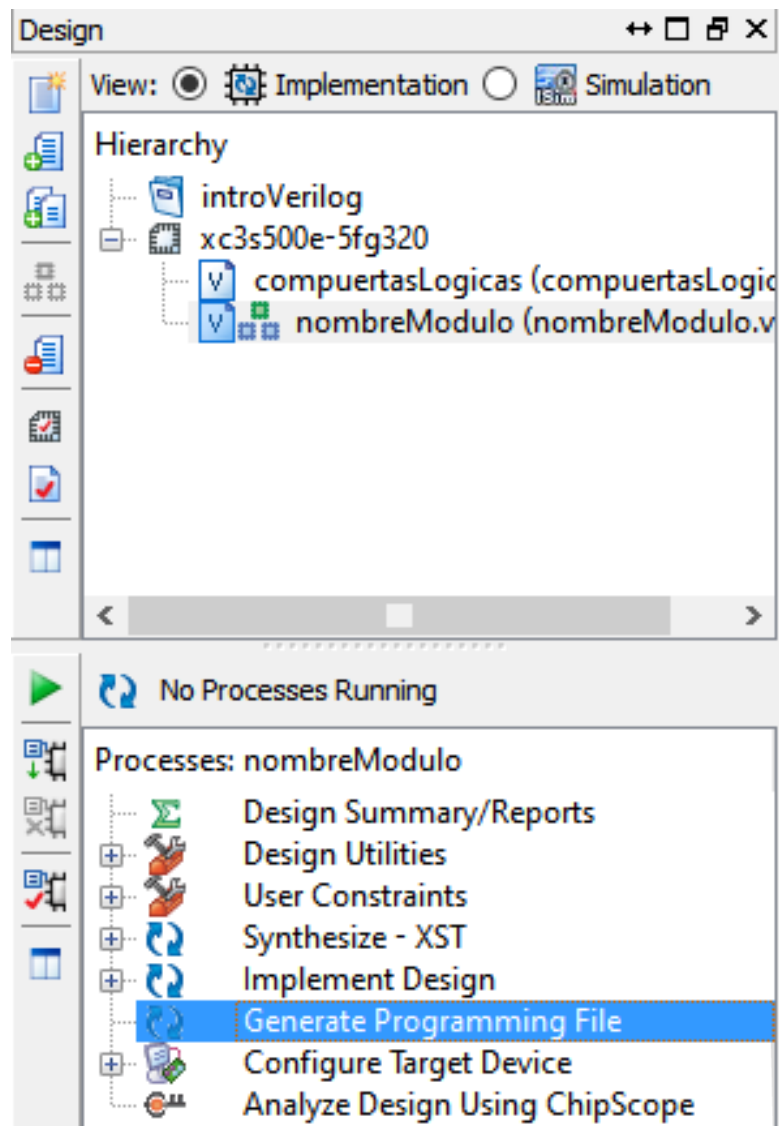


Al hacer esto tendré que repetir todo el proceso de crear un nuevo módulo y deberé nombrarlo diferente al previamente hecho.

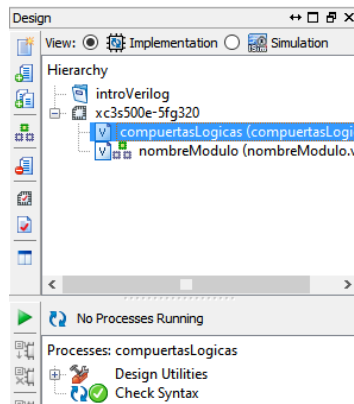
Si no hago que ambos proyectos funcionen de manera conjunta, uno de ellos deberá ser ignorado ya que solo puedo subir un solo módulo a la tarjeta NEXYS 2 y si este no manda a llamar a los demás, los módulos adicionales se quedarán sin utilizar.



El módulo que aparezca con los 2 cuadros azules y uno verde será el que podrá ser subido a la placa. Otro indicador de cuál es el módulo principal es que en este aparecerá la opción de Generate Programming File.



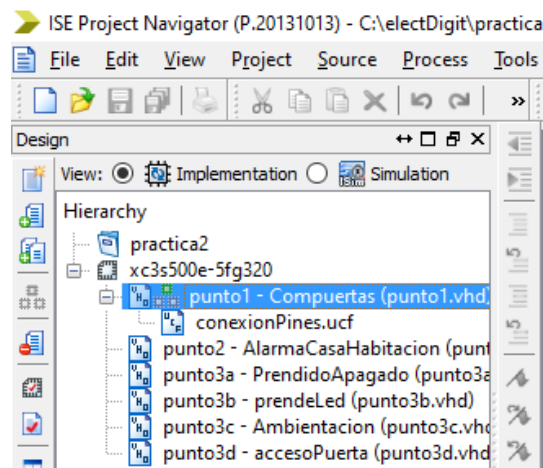
Y en el otro solo aparecerá la opción de Check Syntax (que sirve para ver si el código está bien escrito).



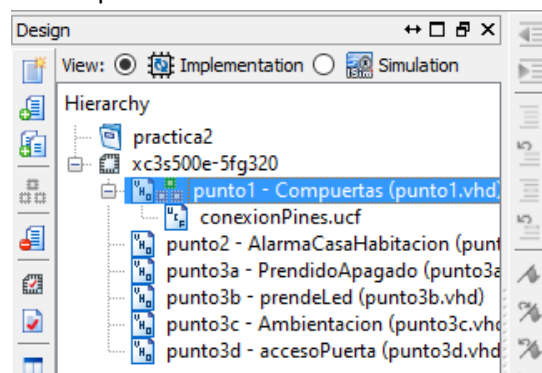
Es importante mencionar que un proyecto con dos módulos no checa correctamente la sintaxis de un código, si queremos asegurarnos de que el código está bien escrito, es mejor crear un proyecto vacío, copiar y pegar, el código que queremos comprobar y dar clic en Synthesize – XST.

## Implementación: Checar Sintaxis

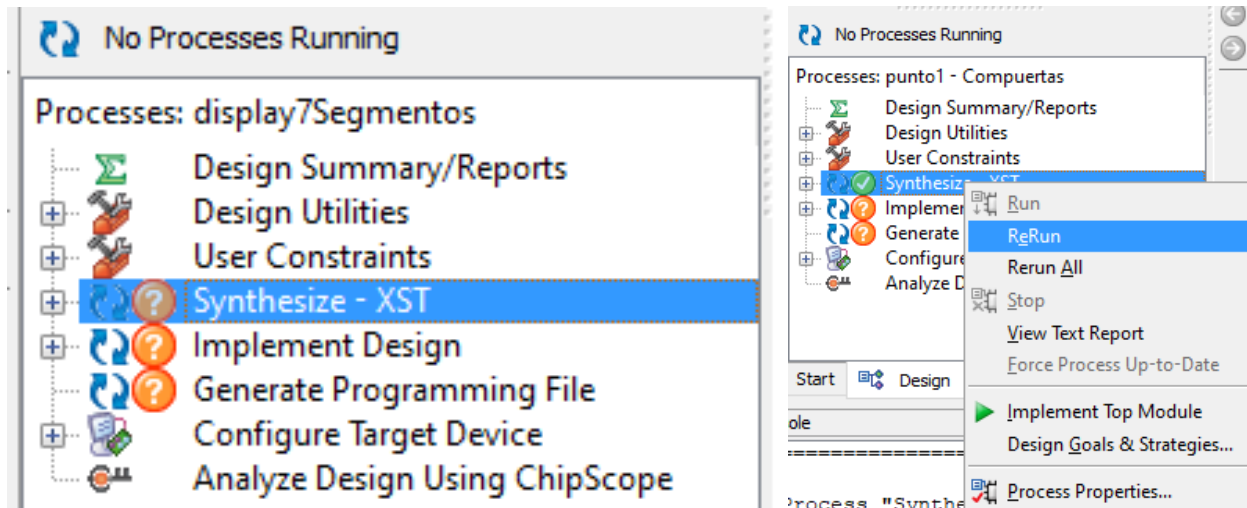
Ya que haya terminado de escribir mi código lo que debo hacer para pueda subir mi código a la tarjeta de desarrollo NEXYS 2 es dar clic en el radio button que dice **Implementation**.



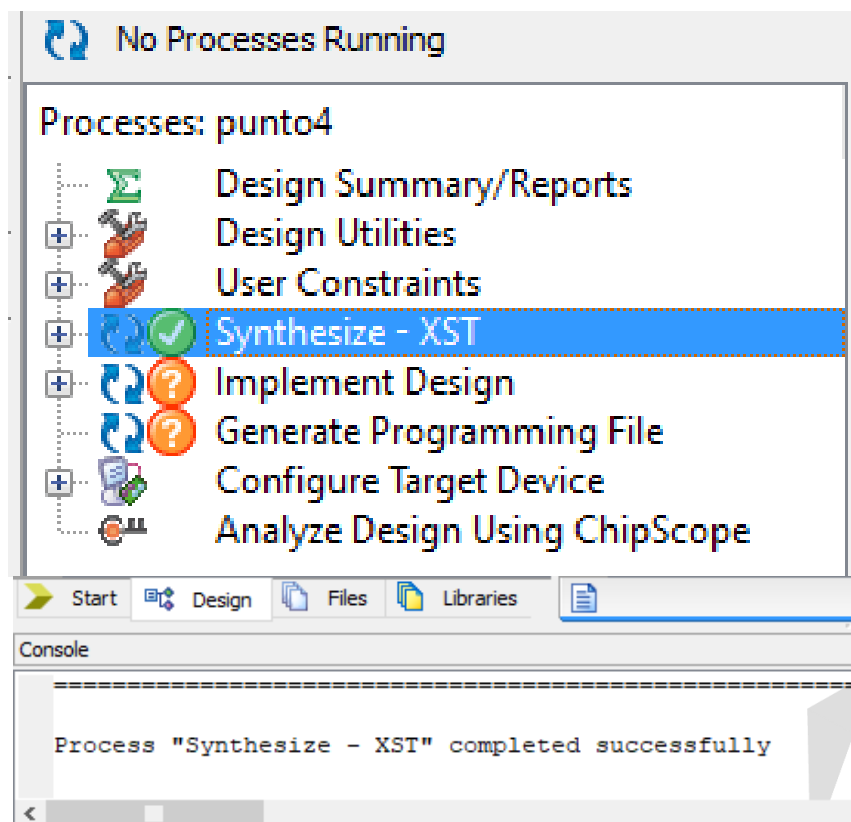
Luego debo seleccionar el módulo (archivo de código) que quiero subir a la placa, si tengo varios módulos solo puedo elegir uno para subir a la placa.



Después, el programa por medio de un proceso llamado **Synthesize - XST** checará si hay algún error en la sintaxis de mi código, esto lo ejecuto dando doble clic sobre el ícono que tiene el mismo nombre o dando clic derecho y seleccionando la opción de ReRun (si ya lo había corrido previamente), antes de ejecutarlo el símbolo aparecerá con un símbolo de pregunta.

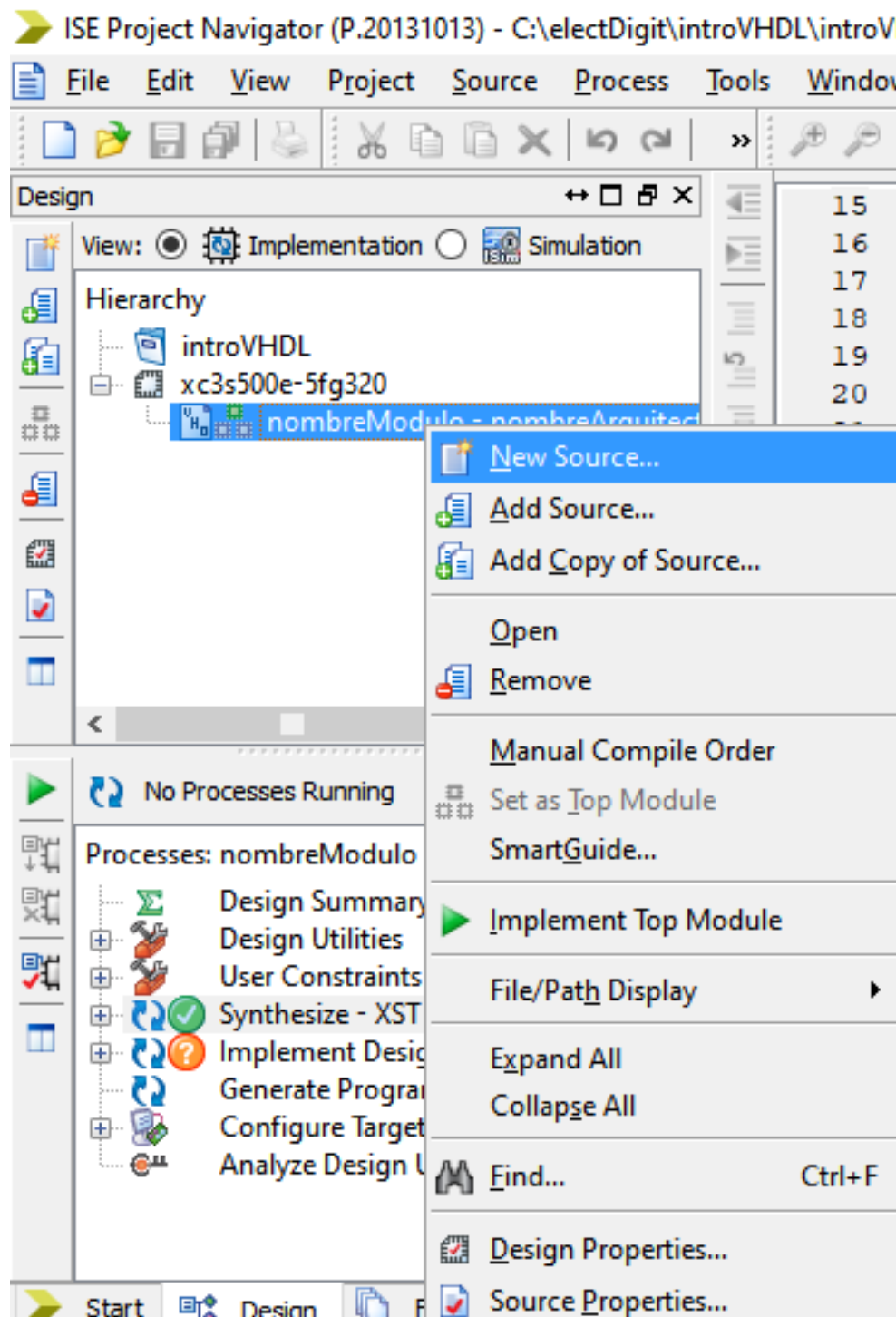


Si hay un error en el código, hasta abajo en la consola se me indicará en qué línea se encuentra. Si no hay ningún error, el ícono de Synthesize se convertirá en una palomita verde y en consola dirá Process "Synthesize - XST" completed successfully.



## Implementación: Archivo UCF

Ya que haya comprobado que no hay error en mi código, debo dar clic derecho en el módulo a subir y seleccionar la opción de New Source.

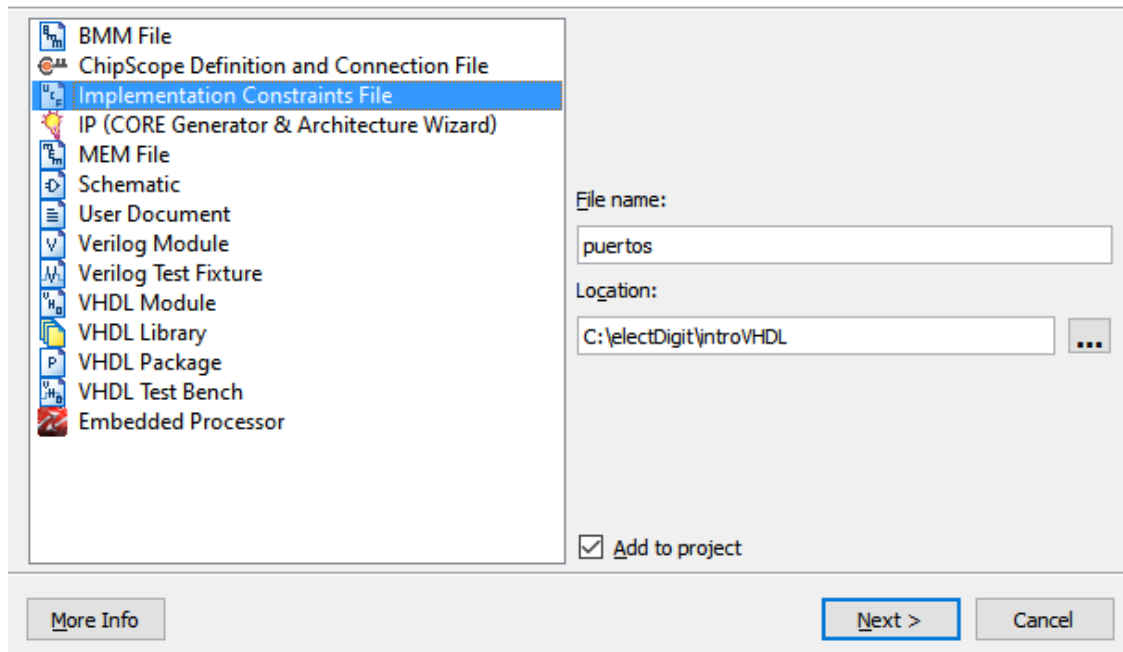


En la ventana consecuente debo seleccionar la opción de **Implementation Constraints File**, este archivo **tendrá extensión UCF** y me servirá para indicar los puertos físicos de mi tarjeta de desarrollo a los que quiero que asignen las salidas y entradas de mi código.



## ← Select Source Type

Select source type, file name and its location.



The dialog box shows a list of source types on the left. 'Implementation Constraints File' is selected and highlighted in blue. On the right, the 'File name' field contains 'puertos' and the 'Location' field contains 'C:\electDigit\introVHDL'. There is a browse button (three dots) next to the location field. At the bottom right, the 'Add to project' checkbox is checked. At the bottom of the dialog are three buttons: 'More Info', 'Next >', and 'Cancel'.

Source types list:

- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File**
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

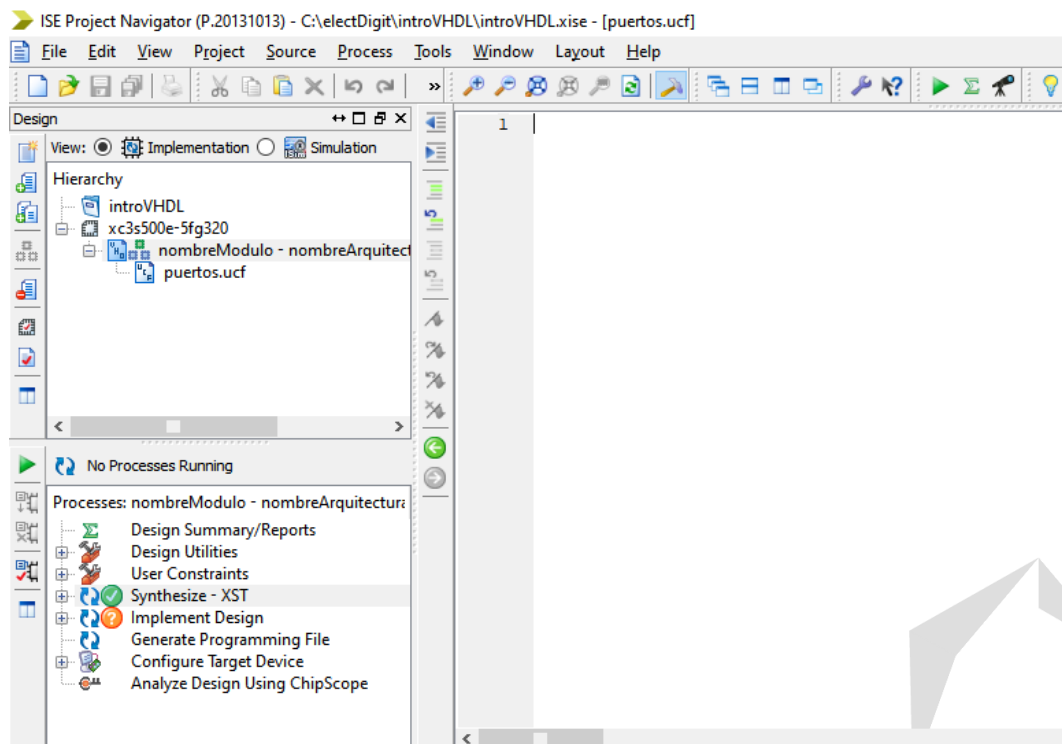
File name:

Location:  ...

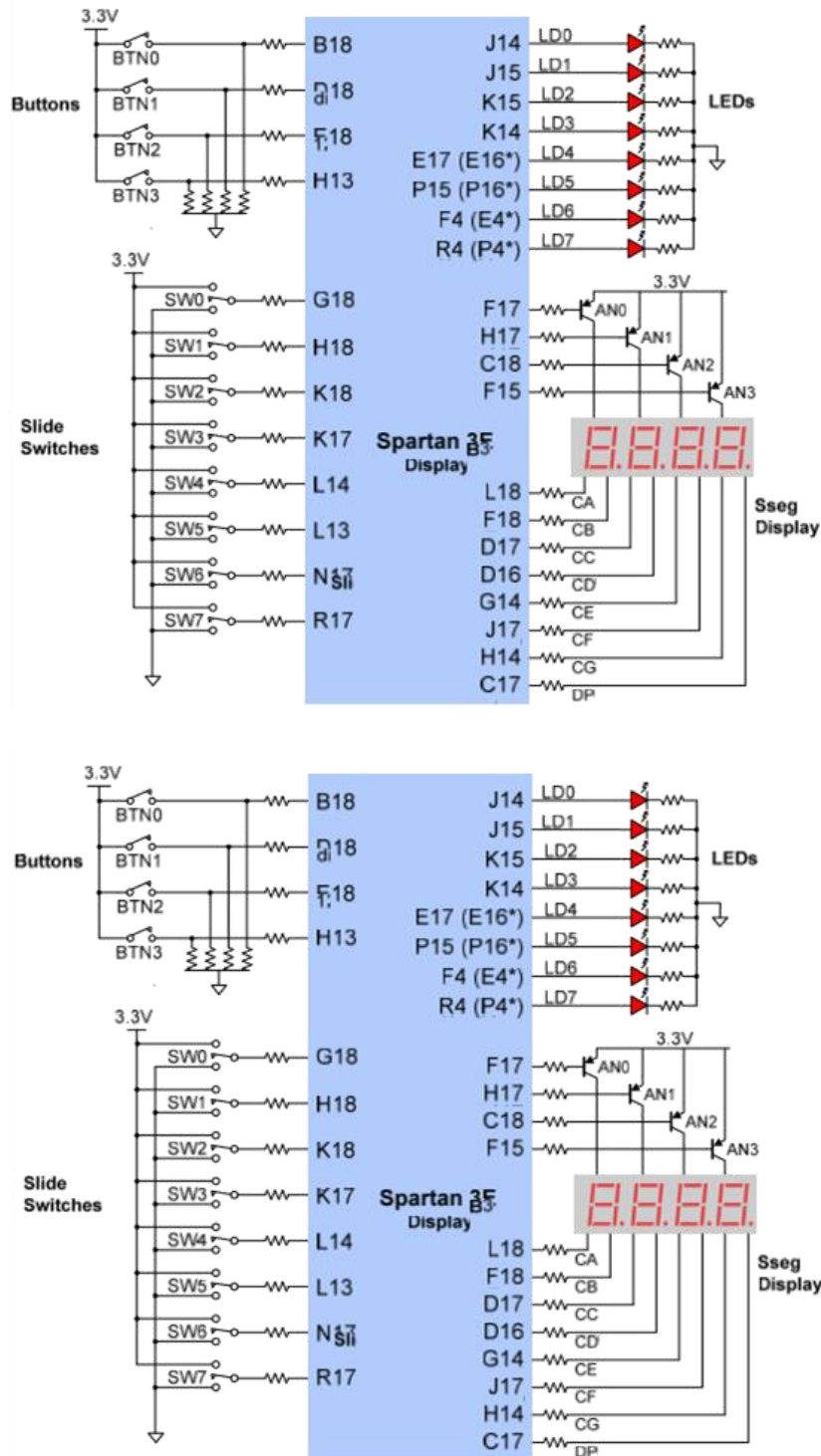
☒ Add to project

Buttons: More Info, Next >, Cancel

Este archivo UCF se vinculará al módulo principal de mi código ya que **por proyecto solo puede existir un con extensión UCF**, inicialmente aparecerá vacía la ventana, en ella debo escribir un código que indique exactamente qué entrada o salida quiero asignar a cada parte de tarjeta NEXYS 2 ya sea un puerto, un led, un switch, etc.



En la siguiente figura (extraída del manual de la tarjeta NEXYS 2) se muestran los nombres y códigos de los diferentes elementos electrónicos, a algunos elementos se debe enviar un solo bit y a otros se debe enviar un vector con un número determinado de bits:



#### • Switches en la NEXYS 2:

##### Entradas de 1 bit:

- SW0 (nombre) - **G18 (código).**
- SW1 (nombre) - **H18 (código).**
- SW2 (nombre) - **K18 (código).**
- SW3 (nombre) - **K17 (código).**
- SW4 (nombre) - **L14 (código).**
- SW5 (nombre) - **L13 (código).**
- SW6 (nombre) - **N17 (código).**
- SW7 (nombre) - **R17 (código).**

#### • Botones en la NEXYS 2:

##### Entradas de 1 bit:

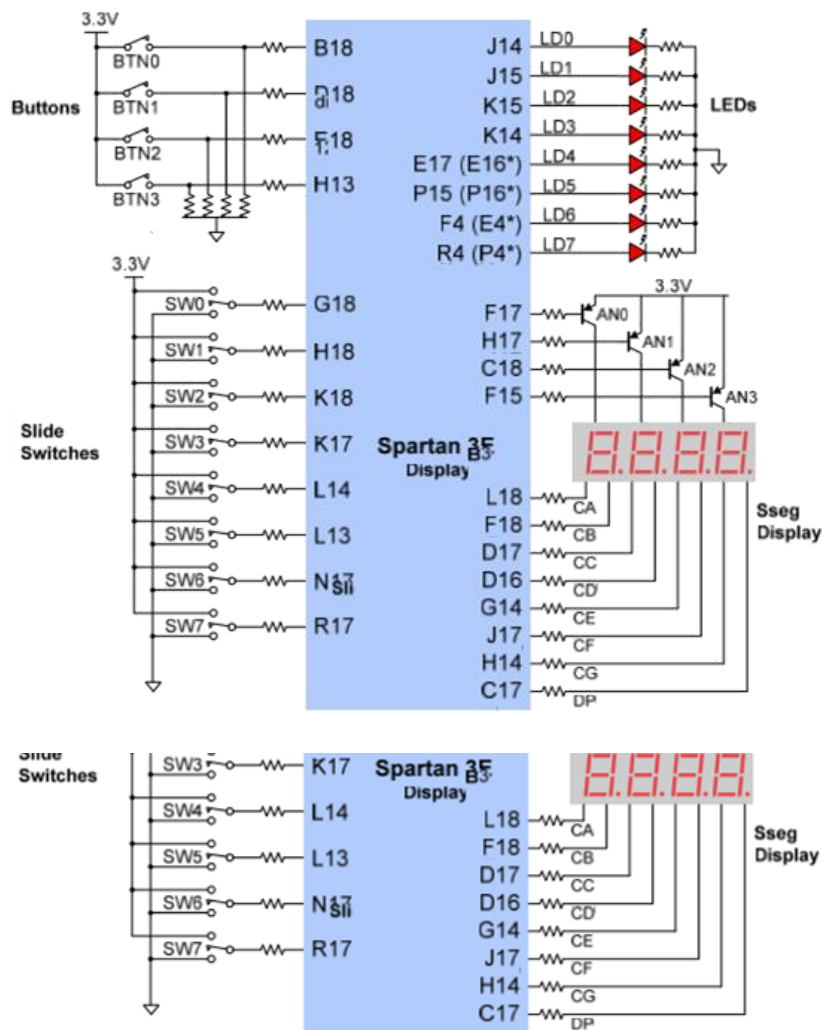
- BTN0 (nombre) - **B18 (código).**
- BTN1 (nombre) - **D18 (código).**
- BTN2 (nombre) - **F18 (código).**
- BTN3 (nombre) - **H13 (código).**

#### • Leds en la NEXYS 2:

##### Salidas de 1 bit:

- LD0 (nombre) - **J14 (código).**
- LD1 (nombre) - **J15 (código).**
- LD2 (nombre) - **K15 (código).**
- LD3 (nombre) - **K14 (código).**
- LD4 (nombre) - **E17 (código).**
- LD5 (nombre) - **P15 (código).**
- LD6 (nombre) - **F4 (código).**
- LD7 (nombre) - **R4 (código).**

Las salidas para los ánodos comunes de los displays de 7 segmentos son de 1 bit, el 0 lógico prende cada uno de los displays y el 1 lógico los apaga.



- **Ánodos comunes de los 4 displays de 7 segmentos para activar cada uno** en la NEXYS 2:

- **Salidas de 1 bit:**

- AN0 (nombre) - **F17 (código).**
- AN1 (nombre) - **H17 (código).**
- AN2 (nombre) - **C18(código).**
- AN3 (nombre) - **F15 (código).**

Las salidas CA, CB, ..., CG y DP son de 1 bit y prenden cada led A, B, C, D, E, F, G o los Puntos de todos los displays de la tarjeta, los 4 displays de la NEXYS 2 muestran siempre la misma figura o número.

- **Leds de los displays de 7 segmentos** en la NEXYS 2:

- **Salidas de 1 bit:**

- CA (nombre) – **L18 (código).**
- CB (nombre) - **J15 (código).**
- CC (nombre) - **K15 (código).**
- CD (nombre) - **K14 (código).**
- CE (nombre) - **E17 (código).**
- CF (nombre) - **P15 (código).**
- CG (nombre) - **F4 (código).**
- DP (nombre) - **R4 (código).**

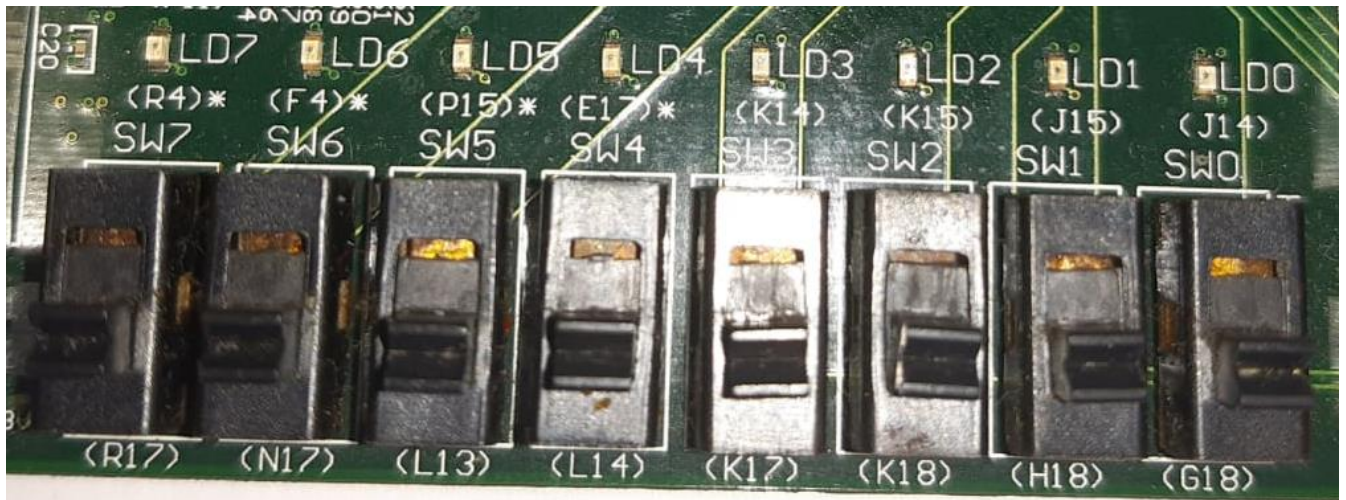
Después veremos más a fondo la forma de conectar los display de 7 segmentos.

Los nombres y códigos de los **switches**, **botones**, **leds**, **ánodos comunes** y **leds de los displays de 7 segmentos** están impresos sobre la tarjeta de desarrollo por lo que no es necesario siempre checkarlo en el manual o en este documento, basta con ver el nombre (que no está entre paréntesis) y el código (que sí va entre paréntesis) de cada uno.

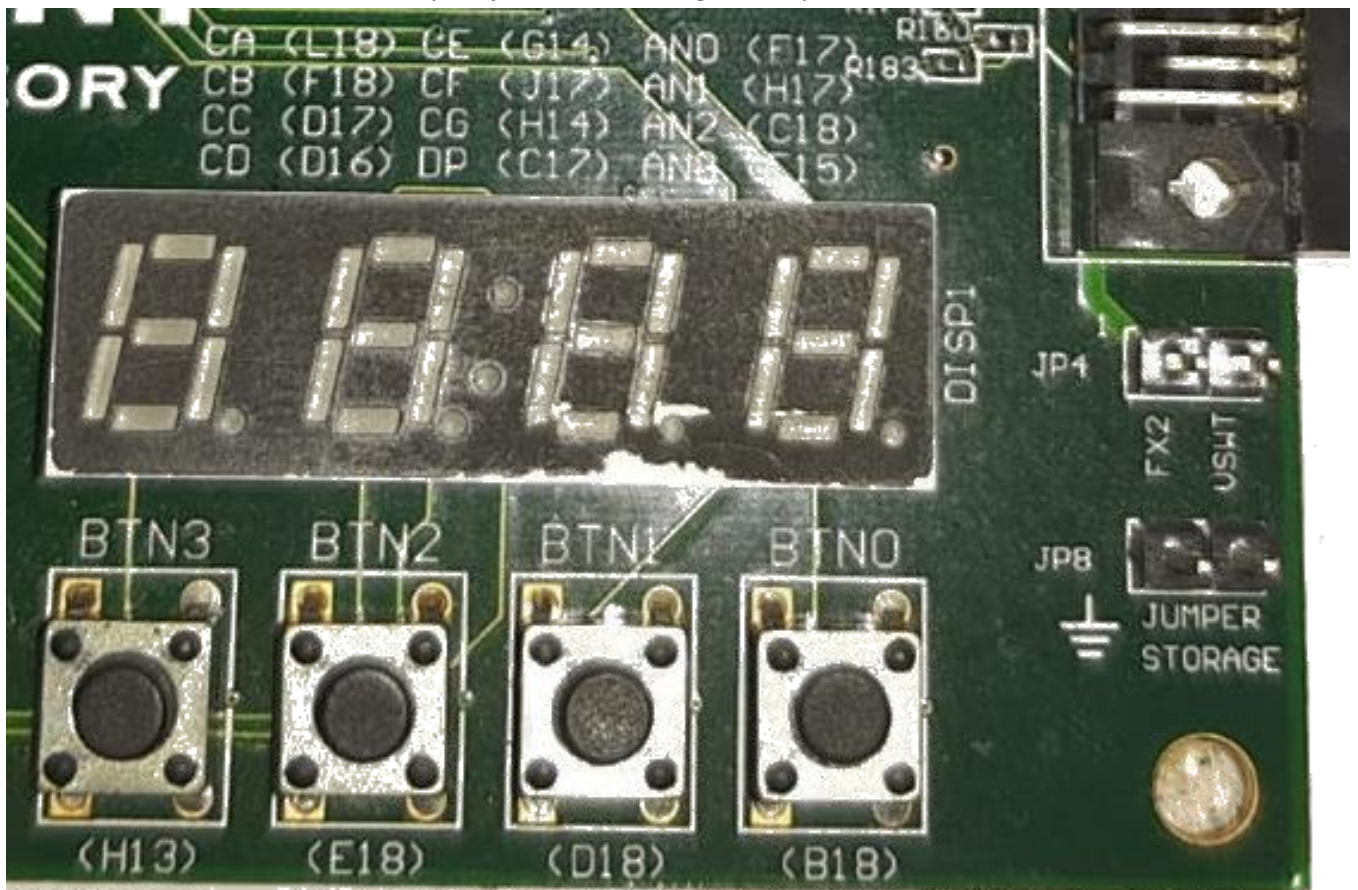




**Leds y switches:** Arriba del elemento se muestra el nombre y abajo el código entre paréntesis.



**Botones, ánodos comunes y leds de los displays de 7 segmentos:** Arriba del elemento se encuentra el nombre y abajo o alado el código entre paréntesis.



En el siguiente diagrama se muestran los nombres y códigos de todos los agujeros pertenecientes a cada conjunto de puertos en la tarjeta de desarrollo NEXYS 2, en total hay 4 conjuntos de puertos y estos sirven

para que por medio de cables o jumpers podamos conectar dispositivos electrónicos externos como motores a pasos, sensores, displays de 7 segmentos externos a la placa, etc.

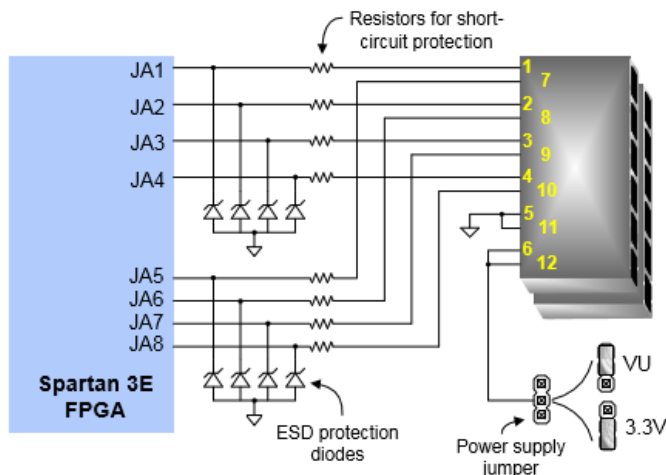
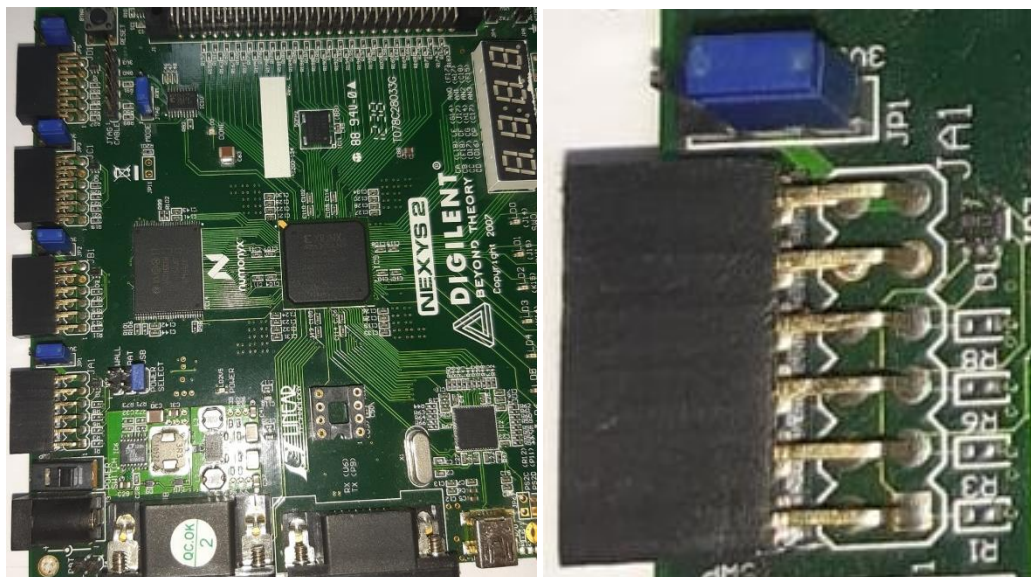


Figure 23: Nexys2 Pmod connector circuits

Table 3: Nexys2 Pmod Connector Pin Assignments							
Pmod JA		Pmod JB		Pmod JC		Pmod JD	
JA1: L15	JA7: K13	JB1: M13	JB7: P17	JC1: G15	JC7: H15	JD1: J13	JD7: K14 <sup>1</sup>
JA2: K12	JA8: L16	JB2: R18	JB8: R16	JC2: J16	JC8: F14	JD2: M18	JD8: K15 <sup>2</sup>
JA3: L17	JA9: M14	JB3: R15	JB9: T18	JC3: G13	JC9: G16	JD3: N18	JD9: J15 <sup>3</sup>
JA4: M15	JA10: M16	JB4: T17	JB10: U18	JC4: H16	JC10: J12	JD4: P18	JD10: J14 <sup>4</sup>

Dentro de cada uno de los 4 conjuntos de puertos encontramos una columna de dos pines que se conectan a tierra y otra columna de dos pines que proporciona un voltaje de alimentación de 3.3V. No aparece el nombre y código específico de cada agujero impresos en la tarjeta, pero podemos ver impreso el nombre del 1er pin de cada uno de los 4 conjuntos, ya sea **JA1**, **JB1**, **JC1** o **JD1** para saber cuál conjunto es cual y ubicar las columnas de tierra y alimentación, ya que ambas se encuentran en el puerto de la otra esquina a la que tiene impreso el nombre.





La sintaxis del código que debo insertar es la siguiente y cada puerto de la tarjeta tiene un código en específico:

- La asignación de las variables que representan entradas y salidas que son de un solo bit se hacen de la siguiente manera:

```
net "nombreEntrada" loc = "codigoDelPuerto";
//Variable de entrada conectada a alguno de los puertos
net "nombreSalida" loc = "codigoDelPuerto";
//Variable de salida conectada a alguno de los puertos
```

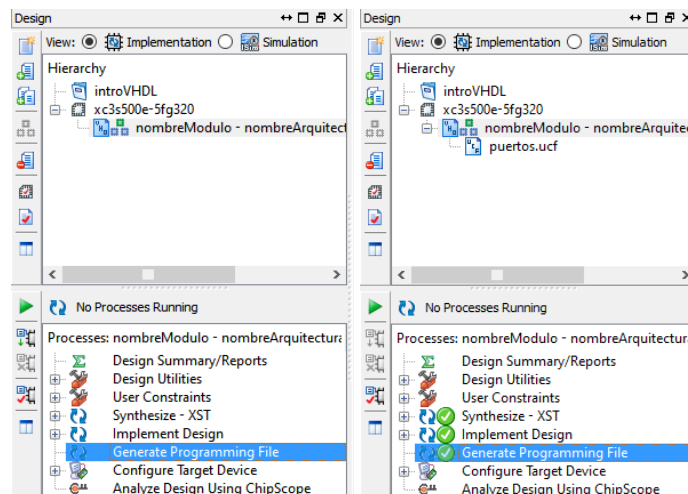
- La asignación de las entradas y salidas que sean vectores de varios bits se hacen de la siguiente manera, poniendo el nombre de la variable y entre corchetes la coordenada del bit que quiero asignar a cada salida si esta solo recibe un bit:

```
net "nombreEntrada[coordenadaDelBit]" loc = "codigoDelPuerto";
//Variable de entrada conectada a alguno de los puertos
net "nombreSalida[coordenadaDelBit]" loc = "codigoDelPuerto";
//Variable de salida conectada a alguno de los puertos
```

Este es el código ucf creado para asignar las entradas y salidas.

```
1 //Entradas y salidas asignadas a cada elemento electrónico de la tarjeta NEXYS2
2 net "a" loc = "R17"; //Entrada a asignada al switch SW7
3 net "b" loc = "N17"; //Entrada a asignada al switch SW6
4 net "c" loc = "L13"; //Entrada a asignada al switch SW5
5
6 net "salidaAND" loc = "R4"; //Salida salidaAND asignada al led LD7
7 net "salidaOR" loc = "F4"; //Salida salidaOR asignada al led LD6
8 net "salidaNAND2" loc = "F15"; //Salida salidaNAND2 asignada al led LD5
9 net "salidaNOR2" loc = "E17"; //Salida salidaNOR2 asignada al led LD4
10 net "salidaXOR" loc = "K14"; //Salida salidaXOR asignada al led LD3
11 net "salidaXNOR2" loc = "K15"; //Salida salidaXNOR2 asignada al led LD2
12 net "salidaNOTa" loc = "J15"; //Salida salidaNOTa asignada al led LD1
13
14 //Las salidas faltantes podrian ser dirigidas a los puertos para que enciendan leds externos
```

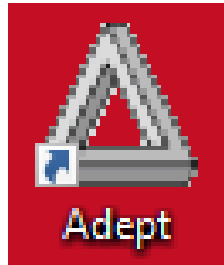
Ahora lo que debo hacer es dar clic en donde dice **Generate Programming File** dando doble clic sobre él para que se cree el archivo con extensión bit, este archivo es el que puede ser subido a la tarjeta por medio de un programa externo llamado Adept.



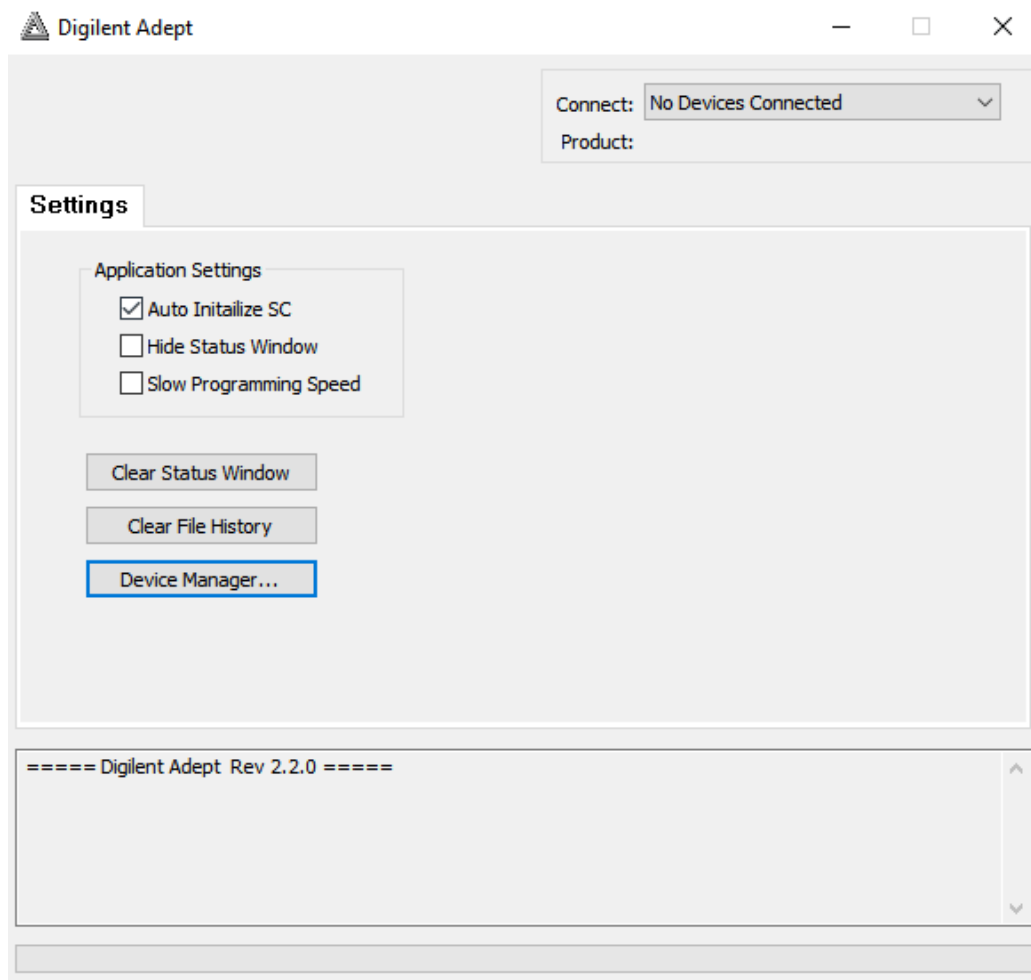
Si no hay ningún error en el código escrito en VHDL ni en el archivo ucf, los tres símbolos se mostrarán con una palomita verde.

## Implementación: Adept

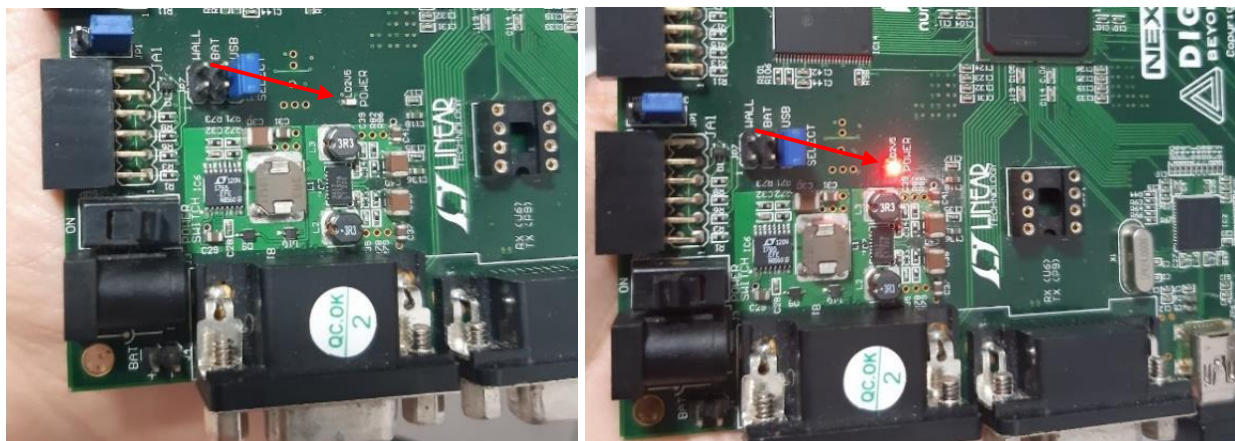
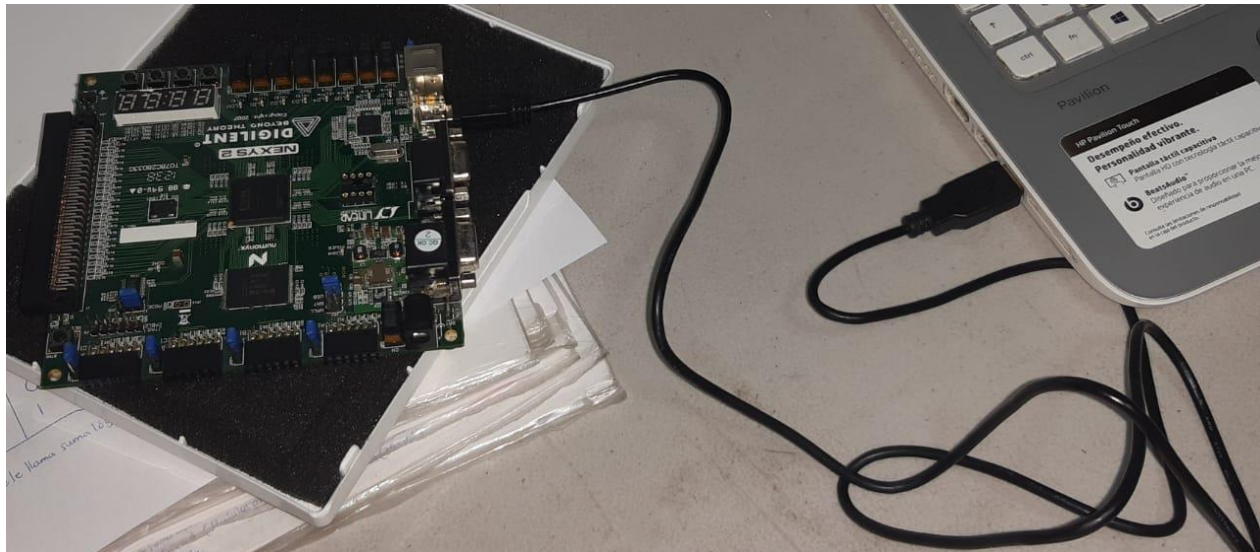
Finalmente, para que mis códigos escritos ya sea en VHDL o Verilog puedan ser subidos a mi tarjeta de desarrollo debo descargar un programa adicional llamado Adept.



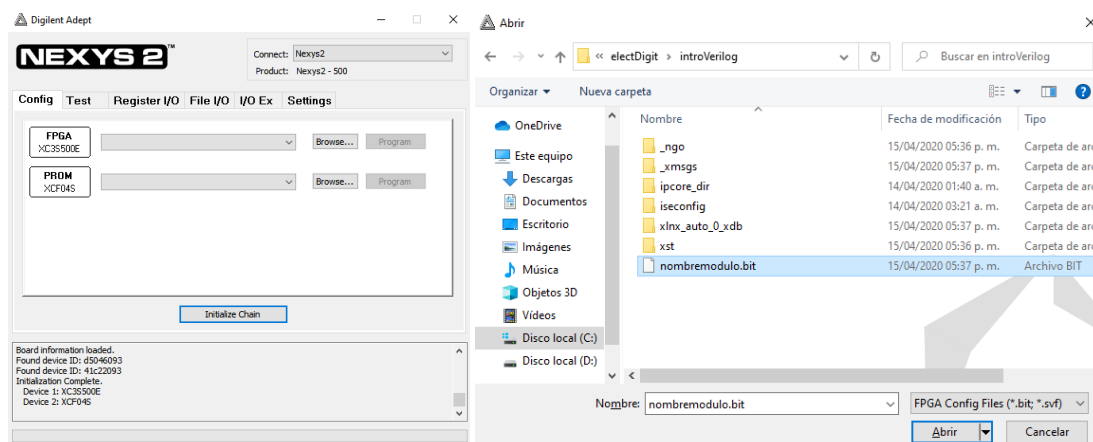
Este programa lo que hace es ver si hay alguna tarjeta de desarrollo conectada al ordenador para poder asignar a sus puertos las entradas y salidas de mi código y ejecutar las acciones programadas.



Para que el programa de Adept reconozca la tarjeta, ésta debe estar conectada vía USB a la computadora y estar encendida.

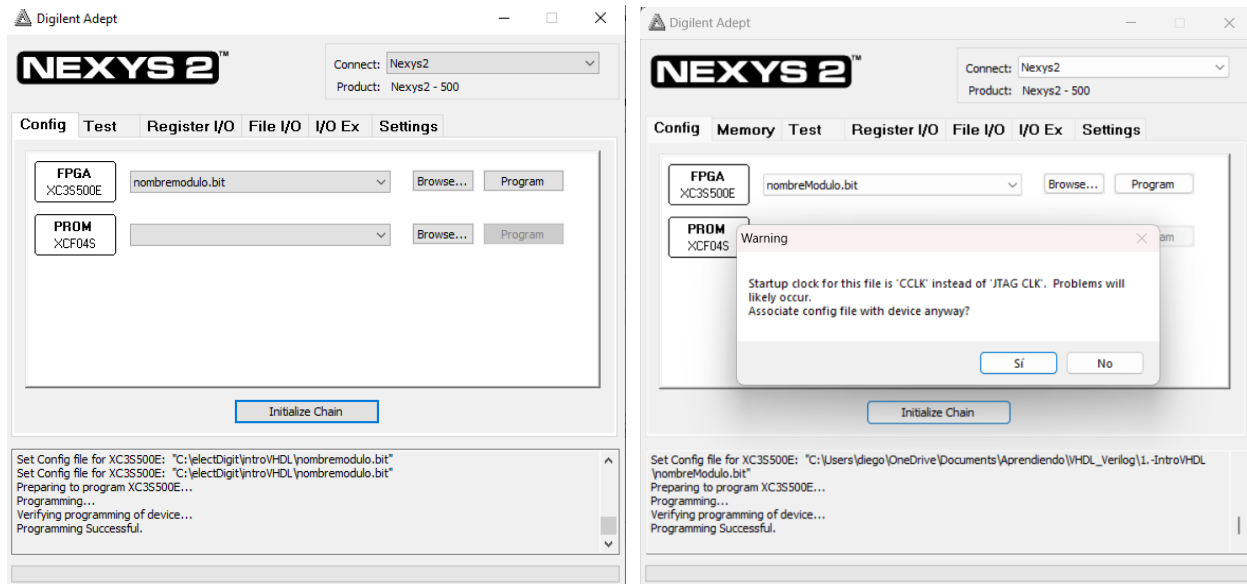


Ya que Adept haya reconocido la tarjeta que estoy usando, solo deberé tomar el archivo bit previamente obtenido del programa ISE y subirlo en donde está la opción de FPGA dando clic en el botón de Browse para buscarlo en la carpeta de mi proyecto.





Ya que seleccione este archivo bit me saldrán varias ventanas donde debo dar clic que sí y finalmente debo dar clic en el botón de Program para que se suba el código a la tarjeta de desarrollo.



Cada que haga un cambio al código y quiera ejecutarlo deberé repetir este proceso después de haber realizado el cambio. La tarjeta de desarrollo al final se conecta de la siguiente manera:



En Windows 11, para poder programar la FPGA, la Virtual Box debe estar cerrada o mínimo se debe haber conectado la FPGA con Adept antes de abrirla.