INGENIERÍA MECATRÓNICA

DI_CERO

DIEGO CERVANTES RODRÍGUEZ

ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE VHDL Y VERILOG

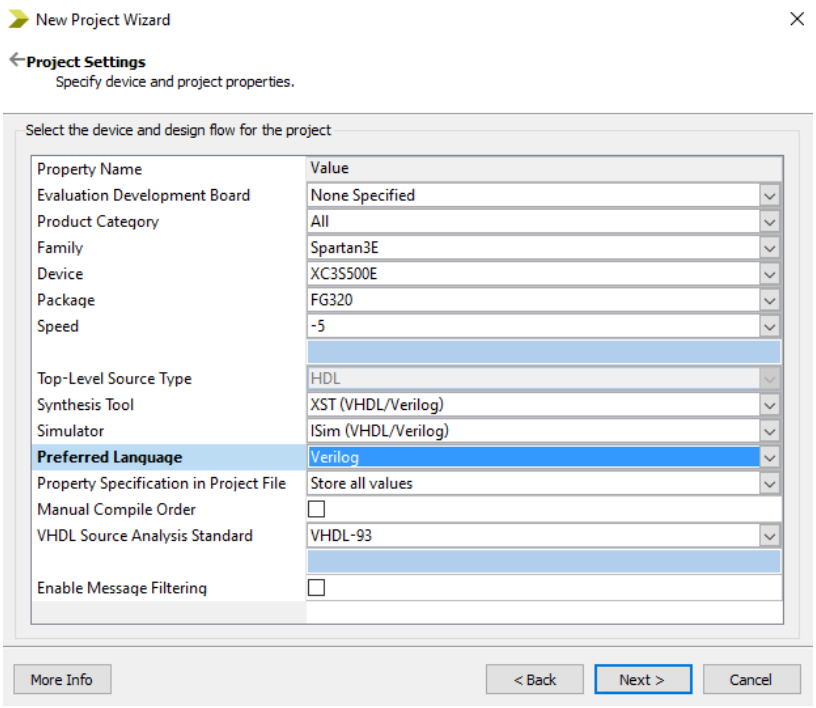
XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

Introducción al Lenguaje Verilog

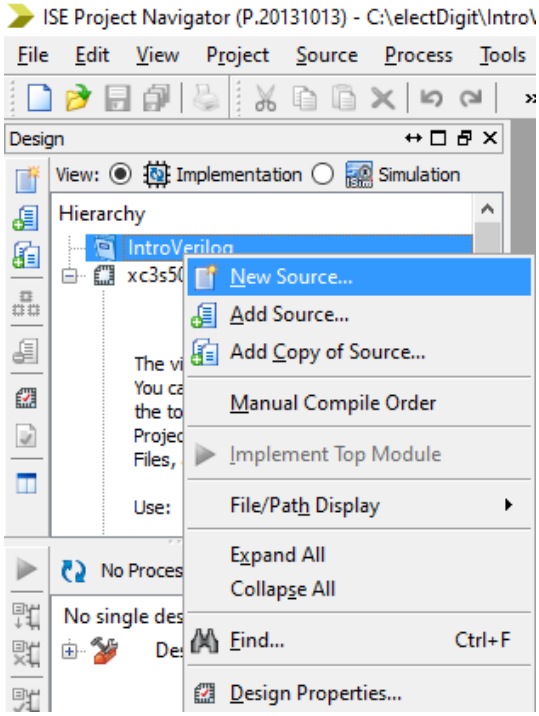
Contenido

Verilog: Lenguaje de descripción de Hardware (HDL)

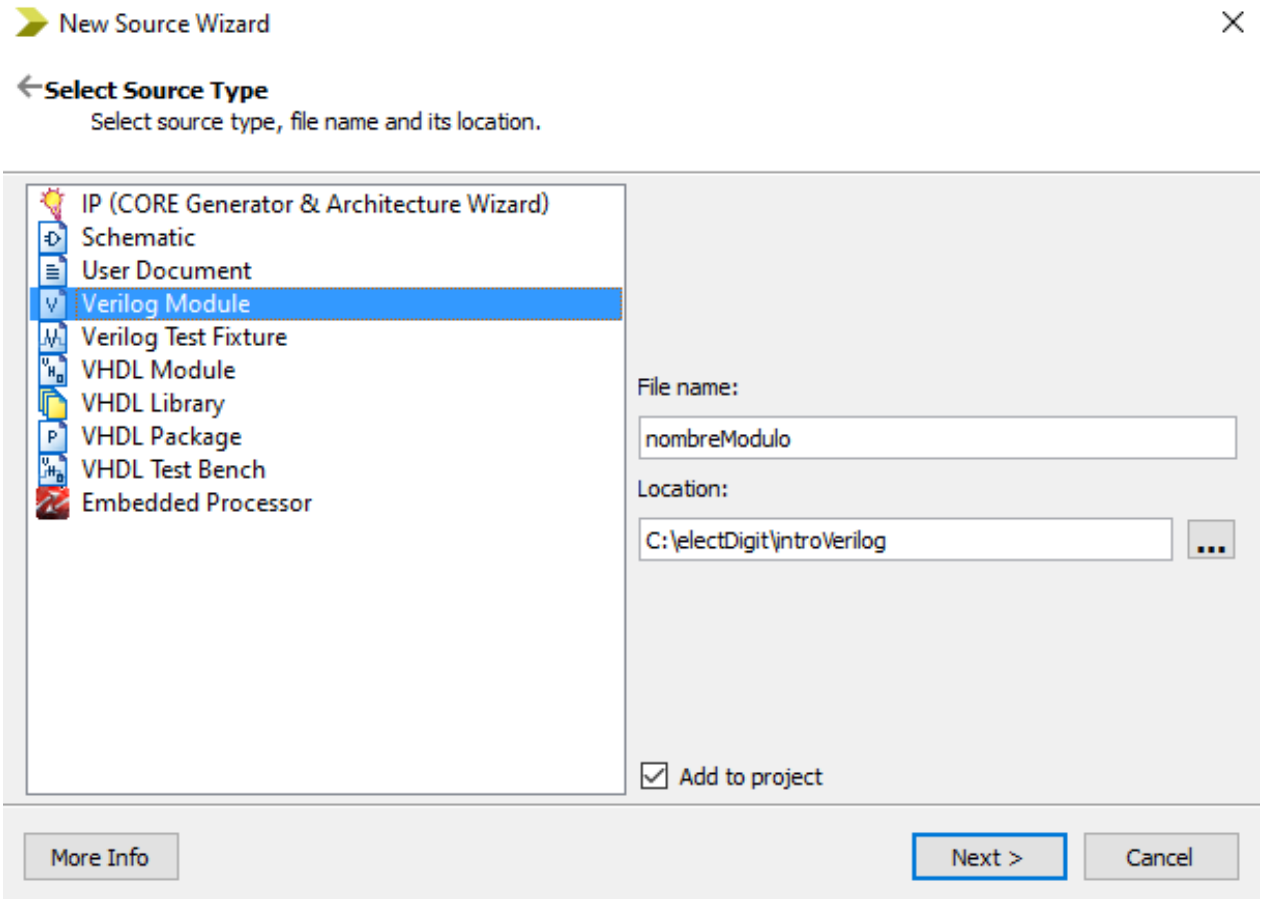
Cuando cree un proyecto nuevo debo indicar que el lenguaje que estoy usando es Verilog. **Lo que ponga en donde dice "VDHL Source Analysis Standard" no importa porque estoy programando en Verilog,** pero las demás especificaciones deben ser las siguientes:



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



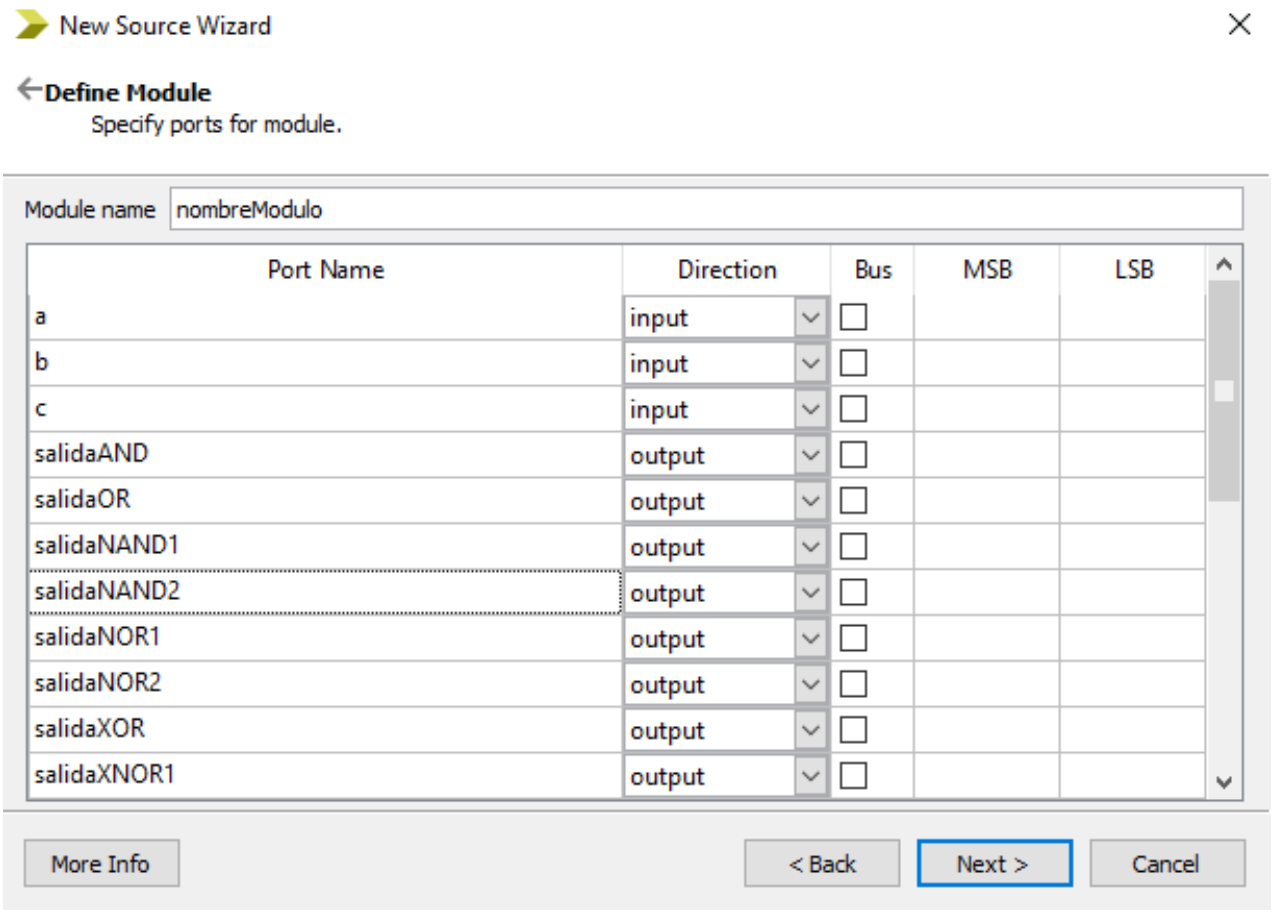
Dentro de la ventana de New Source debo elegir el módulo (así se le llama a todo el código escrito en un lenguaje descriptivo de hardware (HDL) ya sea VHDL o Verilog) del lenguaje de programación que haya elegido, osea **Verilog Module**, esto sirve para que pueda declarar mis entradas y salidas de una 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 Verilog el nombre del módulo aparecerá en el código, porque todo el código se encuentra dentro de un elemento llamado module cuya sintaxis es la siguiente:

```
module nombreModulo (
    //Los comentarios se declaran con dos barras diagonales seguidas.
    //Dentro de estos paréntesis van declaradas las entradas/salidas
    );
//Fuera de los paréntesis van las instrucciones para lo que harán las
//entradas/salidas
endmodule
```

En la ventana consecuente puedo renombrar al módulo y debo declarar el nombre de las variables que representarán mis entradas y/o salidas e indicar que es cada una.



Módulo: 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 Verilog estas entradas y salidas son declaradas dentro de los paréntesis que van después del nombre del módulo, se usará la palabra reservada input **para declarar las entradas** y la palabra reservada output **para declarar las salidas**, es importante mencionar que las entradas/salidas se separan entre sí por una coma, **en la última salida o entrada declarada no se debe poner una 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.

```
//La sintaxis en el código de la entidad para declarar las entradas y salidas en
//forma de bit es la siguiente:
module nombreModulo (
```

```
    //Dentro del paréntesis declaro mis entradas/salidas
    input   entrada,
    output salida1,
    output salida2
    );
//Fuera del paréntesis declaro las acciones que harán mis entradas/salidas
endmodule
```

- **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 usa las sintaxis [n:m], el tamaño del vector se calcula contando desde la coordenada n hasta la m, incluyendo en mi conteo tanto a la coordenada n como a la m. El **bit más significativo** es convocado con la coordenada de la izquierda, osea **n** y el **bit menos significativo** es convocado con la coordenada de la derecha, osea **m**, puedo indicar el intervalo de coordenadas que quiera.

```
//La sintaxis en el código de la entidad para declarar las entradas y salidas en
//forma de vector (número binario) es la siguiente:
module nombreModulo (
    //Dentro del paréntesis declaro mis entradas/salidas
    input   [2:0] entrada,
    output [0:2] salida1,
    output [0:3] salida2
    );
//Fuera del paréntesis declaro las acciones que harán mis entradas/salidas
endmodule
//el bit más significativo y el menos significativo son 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 la entrada 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.
//0,1,2 para la salida1 que formó un vector de 3 bits también y donde la
//coordenada del bit más significativo es la de 0 y la del menos significativo es la
//de 2.
//0,1,2,3 para la salida2 que formó un vector de 4 bits y donde la coordenada del
//bit más significativo es la de 0 y la del menos significativo es la de 3.
```

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 sirve la pantalla previamente mencionada donde podemos introducir directamente algunos aspectos como:

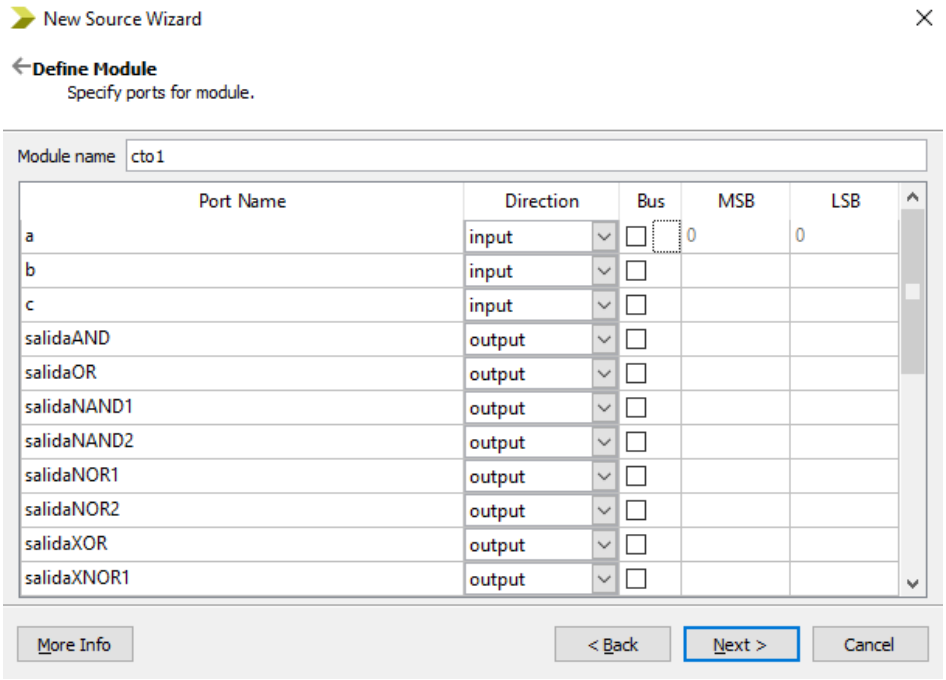
- Poner el **nombre de mis entradas** junto con la opción de **Direction input**.
- Indicar el **nombre de mis salidas** junto con la opción de **Direction output**.

Port Name	Direction
a	input
b	input
c	input
salidaAND	output
salidaOR	output
salidaNAND1	output
salidaNAND2	output
salidaNOR1	output
salidaNOR2	output
salidaXOR	output
salidaXNOR1	output

- 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 Significative Bit)
        - La coordenada de su **bit menos significativo donde dice LSB** (Least Significative Bit)
        - Ambas indicarán el tamaño del vector y sus coordenadas.

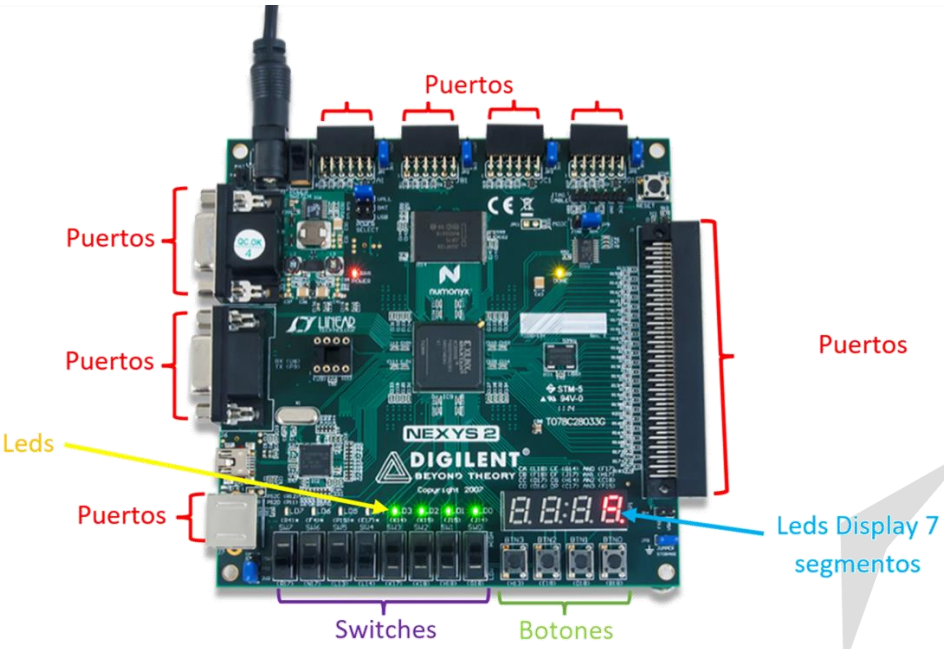
Bus	MSB	LSB
☑	2	0

Hasta arriba de la ventana aparecerá el nombre de mi módulo (este es el nombre que tendrá todo mi código escrito en Verilog). Finalmente debo dar clic al botón de Next.

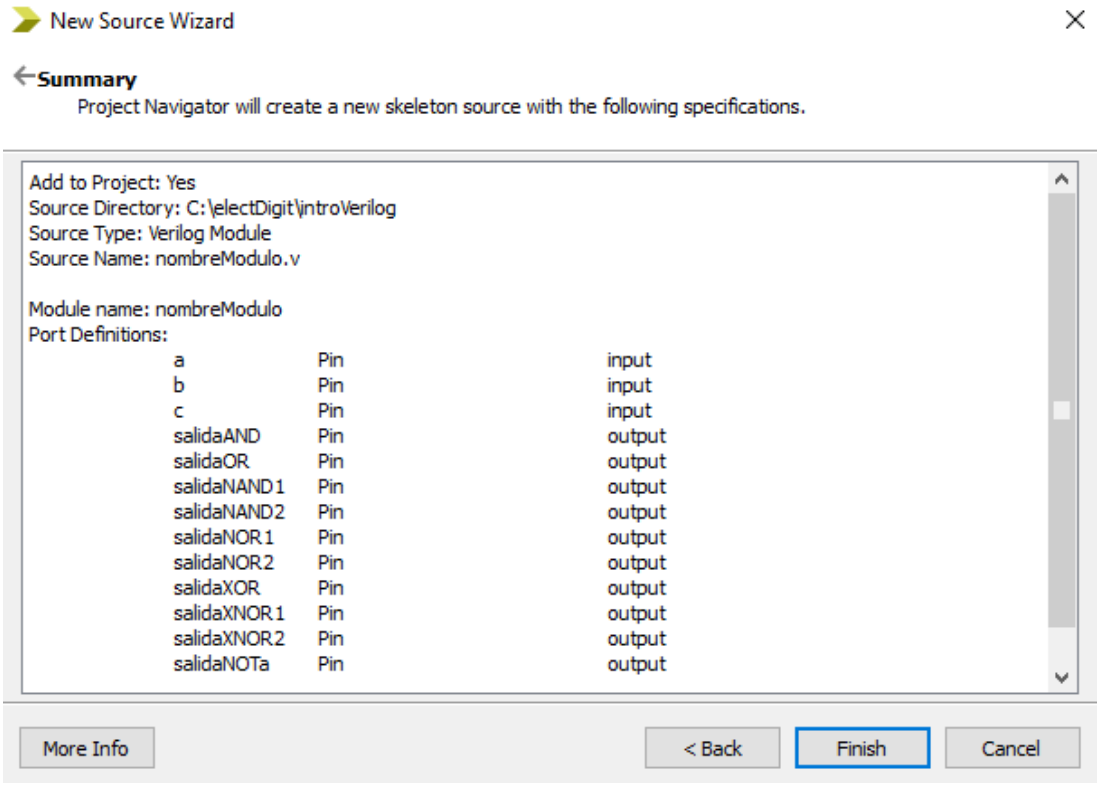


Las entradas y salidas pueden ser los siguientes elementos electrónicos pertenecientes a la tarjeta de desarrollo NEXYS 2:

- 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 Verilog**.



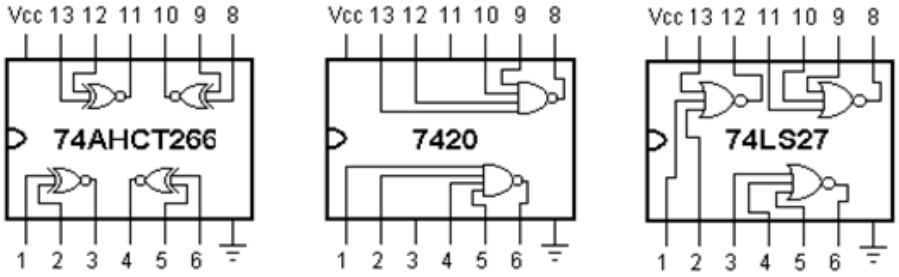
Este es el código creado por el módulo declarado contiene las entradas y salidas.

```
1  //Los comentarios se ponen con dos barras diagonales seguidas
2  //El lenguaje Verilog si distingue entre mayúsculas y minúsculas
3  module nombreModulo(
4      //Dentro de estos paréntesis declaro mis entradas/salidas
5      input a,
6      //Las entradas/salidas se separan entre ellas por una coma
7      input b,
8      input c,
9      output salidaAND,
10     output salidaOR,
11     output salidaNAND1,
12     output salidaNAND2,
13     output salidaNOR1,
14     output salidaNOR2,
15     output salidaXOR,
16     output salidaXNOR1,
17     output salidaXNOR2,
18     output salidaNOTa
19     //La última entrada o salida no lleva una coma
20     );
21
22 //Fuera de los paréntesis declaro las acciones que harán mis entradas/salidas
23
24 endmodule
```

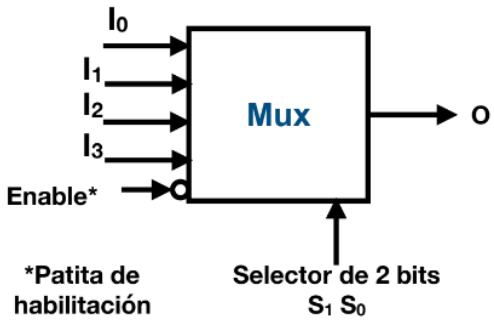
La función de cada módulo de código ya sea de VHDL o Verilog es describir las conexiones que hará la FPGA para crear un diseño de circuito integrado personalizado para ejecutar una acción determinada con un número de entradas y salidas declaradas. En Verilog a todo el código se le llama **módulo**, en los paréntesis del módulo se declaran las entradas/salidas pero fuera de los paréntesis puedo ejecutar operaciones lógicas AND, OR, NOT, NAND, NOR, XOR y XNOR, o 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:

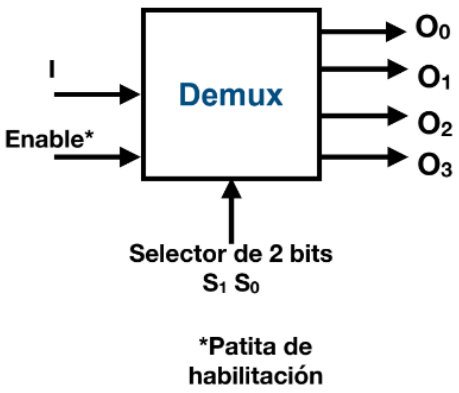
Circuitos lógicos:



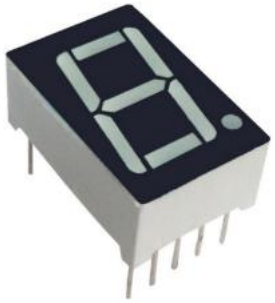
Multiplexor o MUX:



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 Verilog Completo

Verilog es un lenguaje **case sensitive**, osea que **distingue entre mayúsculas y minúsculas**, hay que tener cuidado con este aspecto, además se asigna un valor a cualquier variable con el signo igual **=** no importando si la variable es una entrada o una salida.

Circuitos Lógicos en Verilog

Las operaciones lógicas en Verilog se pueden hacer de dos maneras:

1) **Por medio de símbolos:** En esta forma las entradas se deben declarar como wire y cuando asigne valores a mis variables debo poner la palabra reservada assign.

```
//1) Operaciones lógicas por medio de símbolos
//Módulo: Dentro de éste va todo el código escrito en Verilog
module nombreModulo (
    //Las entradas y salidas: van declaradas dentro del paréntesis
     input  wire a,
     input  wire b,
     output wire salidaAND,
     output wire salidaOR,
     output wire salidaNOTa,
     output wire salidaNAND,
     output wire salidaNOR,
     output wire salidaXOR,
     output wire salidaXNOR
     );
```

```
//Fuera del paréntesis del módulo se indica lo que hará el programa con las
//entradas/salidas declaradas, las operaciones más sencillas son las lógicas.
//Recuerdo que en Verilog para asignar un valor debo usar el signo igual =
assign salidaAND    = a & b & c;
//La operación AND se hace con el símbolo & presionando "alt" + "6"
assign salidaOR     = a | b | c;
//La operación OR se hace con el símbolo | presionando la tecla a la izquierda del
//número 1 en el teclado
assign salidaNOTa   = ~ (a);
//La operación NOT se hace con el símbolo ~ presionando "alt gr" + "+"
assign salidaNAND = ~ (a & b & c);
//La operación NAND se hace negando la operación AND con el símbolo ~
//presionando "alt gr" + "+"
assign salidaNOR    = ~ (a | b | c);
//La operación NOR se hace negando la operación OR con el símbolo ~
//presionando "alt gr" + "+"
assign salidaXOR    = ~ (a ^ b ^ c);
//La operación XOR se hace con el símbolo ^ presionando "alt gr" + "{" + "space bar"
assign salidaXNOR = ~ (a | b | c);
//La operación NOR se hace negando la operación OR con el símbolo ~
//presionando "alt gr" + "+"
endmodule
```

2) **Por medio de funciones:** Las entradas y salidas se declaran de forma normal solo usando input y output, además las funciones usadas tienen los mismos nombres que cada operación lógica.

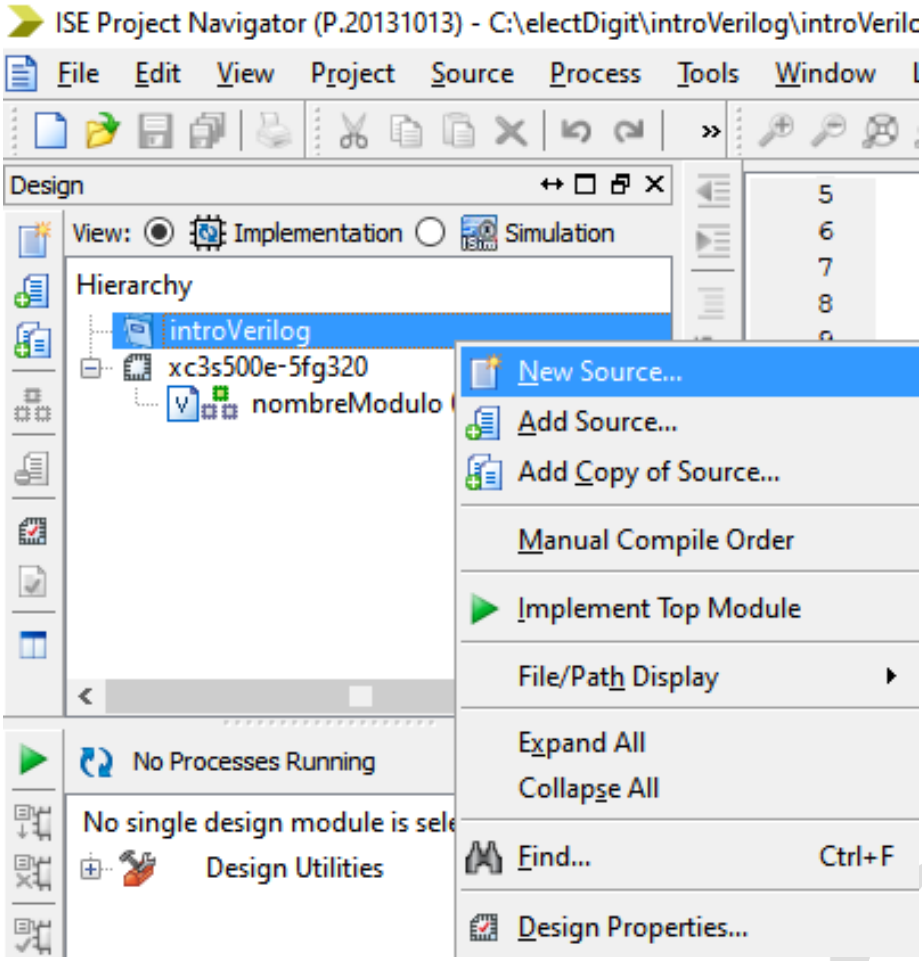
```
//2) Operaciones lógicas por medio de funciones
//Módulo: Dentro de éste va todo el código escrito en Verilog
module nombreModulo (
    //Las entradas y salidas: van declaradas dentro del paréntesis
    input    a,
    input    b,
    output   salidaAND,
    output   salidaOR,
    output   salidaNOTa,
    output   salidaNAND,
    output   salidaNOR,
    output   salidaXOR,
    output   salidaXNOR
    );
//Fuera del paréntesis del módulo se indica lo que hará el programa con las
//entradas/salidas declaradas, las operaciones más sencillas son las lógicas.
//En Verilog para asignar un valor debo usar el signo igual =
```

```
//Cuando hago operaciones lógicas de esta manera debo poner como palabra reservada
//la operación que estoy haciendo seguido de unos paréntesis en el 1er elemento que
//ponga se guardará el resultado de mi operación lógica y los demás la ejecutarán
and (salidaAND, a, b, c);
or (salidaOR, a, b, c);
not (salidaNOTa, a);
nand (salidaNAND, a, b, c);
nor (salidaNOR, a, b, c);
xor (salidaXOR, a, b, c);
xnor (salidaXNOR, a, b, c);

endmodule
```

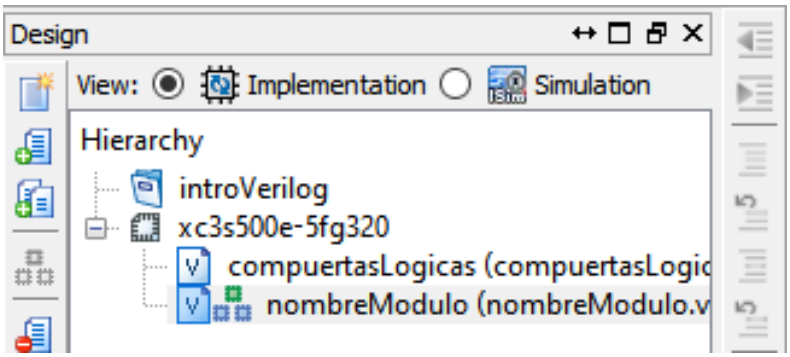
Agregar Dos Módulos en un Mismo Proyecto

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.

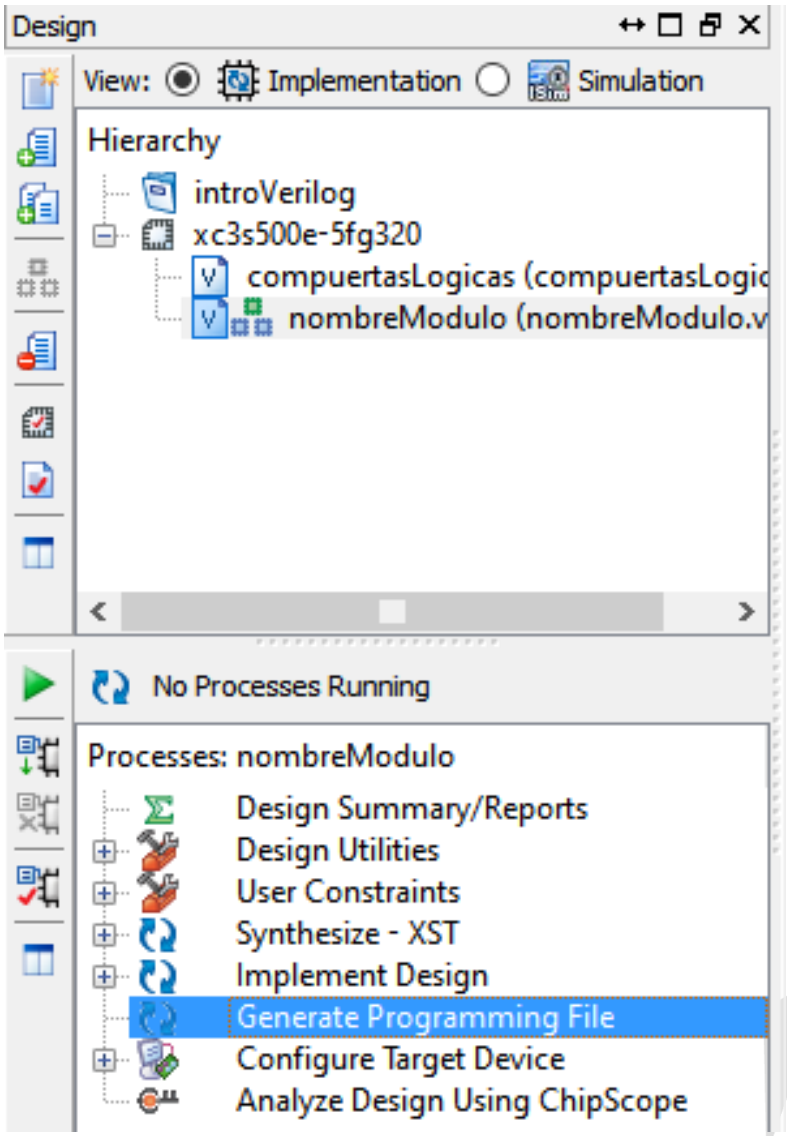


Al hacer esto tendré que repetir todo el proceso de crear un nuevo módulo y deberé nombrarlo diferente al previamente hecho.

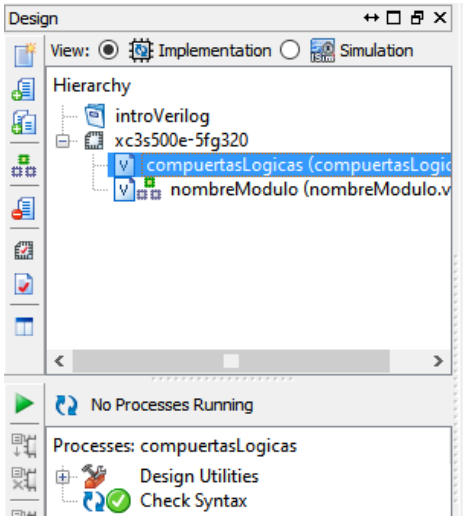
Si no hago que ambos proyectos funcionen de manera conjunta, uno de ellos deberá ser ignorado ya que solo puedo subir un solo módulo a la tarjeta NEXYS 2 y si este no manda a llamar a los demás, los módulos adicionales se quedarán sin utilizar.



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.



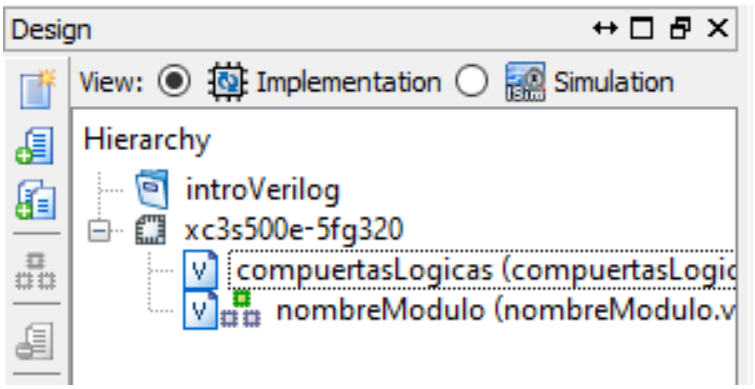
Y en el otro solo aparecerá la opción de Check Syntax (que sirve para ver si el código está bien escrito).



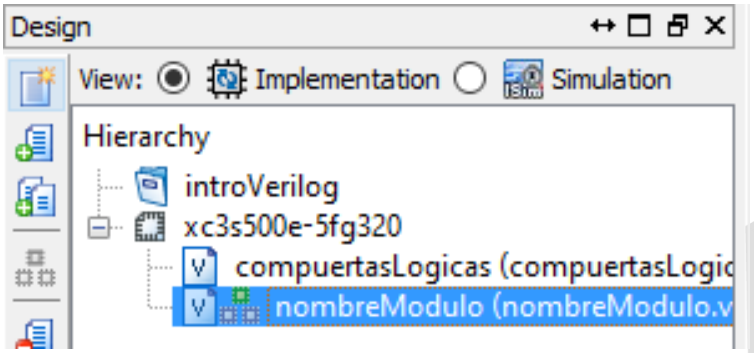
Es importante mencionar que un proyecto con dos módulos no checa correctamente la sintaxis de un código, si queremos asegurarnos de que el código está bien escrito, es mejor crear un proyecto vacío, copiar y pegar, el código que queremos comprobar y dar clic en Synthesize – XST.

Implementación: Checar Sintaxis

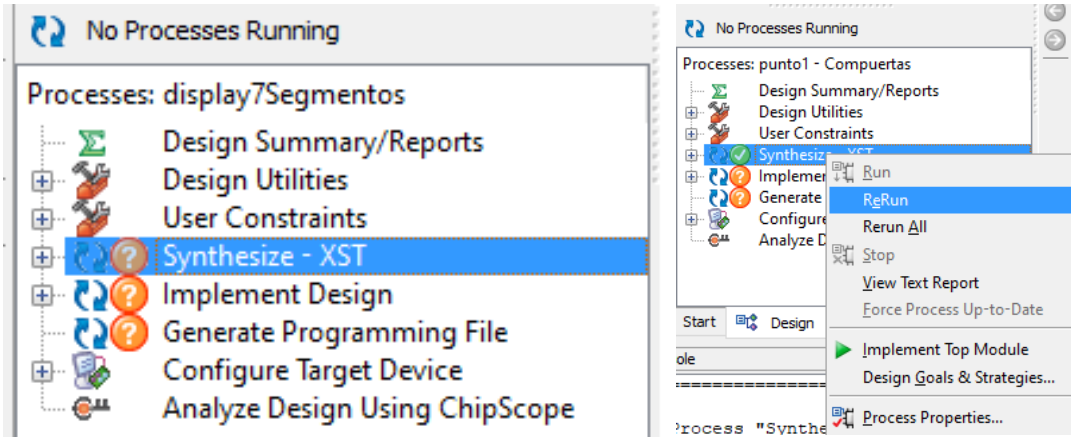
Ya que haya terminado de escribir mi código lo que debo hacer para pueda subir mi código a la tarjeta de desarrollo NEXYS 2 es dar clic en el radio button que dice **Implementation**.



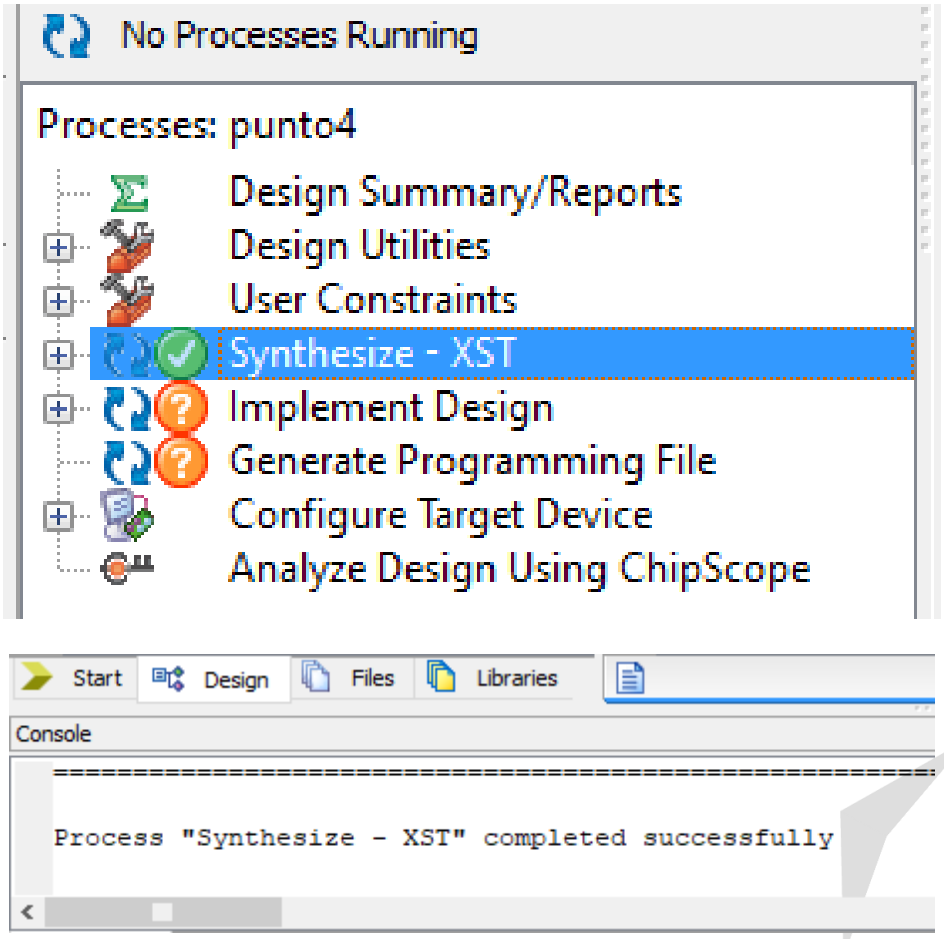
Luego debo seleccionar el módulo (archivo de código) que quiero subir a la placa (debe ser el que tiene los cuadrados), si tengo varios módulos solo puedo elegir uno para subir a la placa.



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.

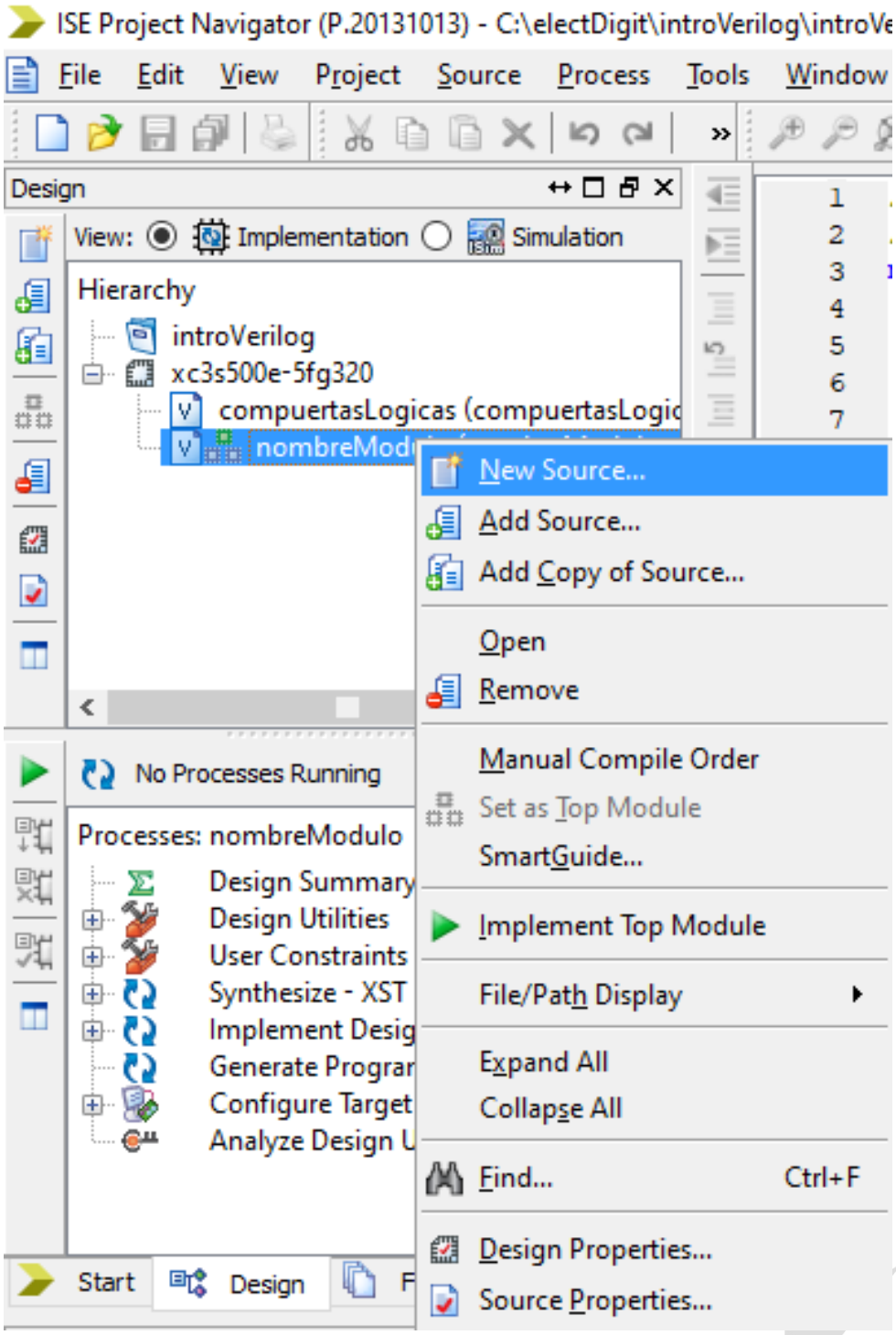


Si hay un error en el código, hasta abajo en la consola se me indicará en qué línea se encuentra.
Si no hay ningún error, el ícono de Synthesize se convertirá en una palomita verde y en consola dirá Process "Synthesize - XST" completed successfully.

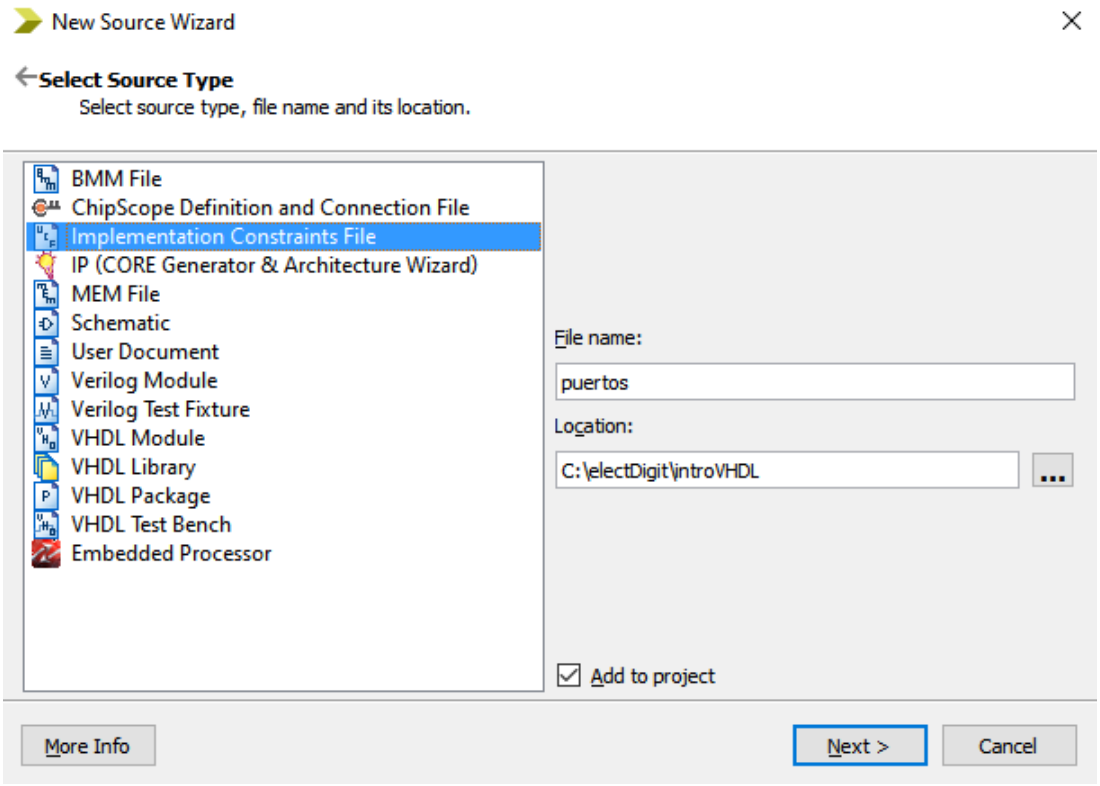


Implementación: Archivo UCF

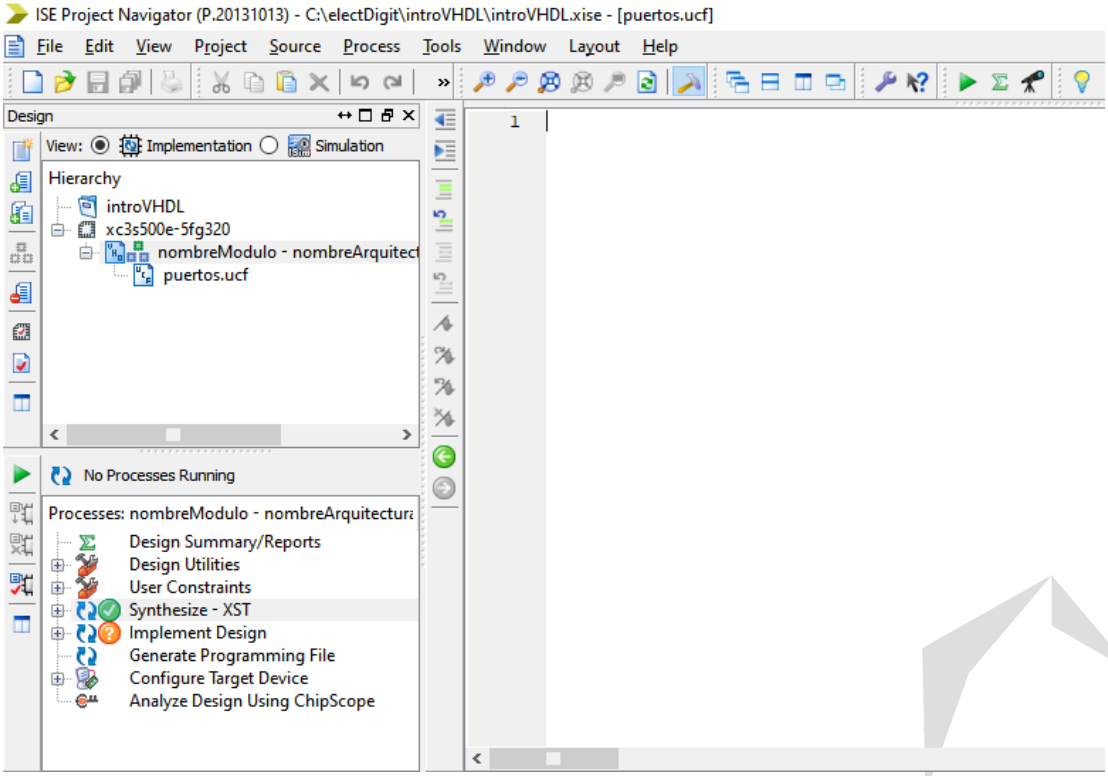
Ya que haya comprobado que no hay error en mi código, debo dar clic derecho en el módulo a subir y seleccionar la opción de New Source.



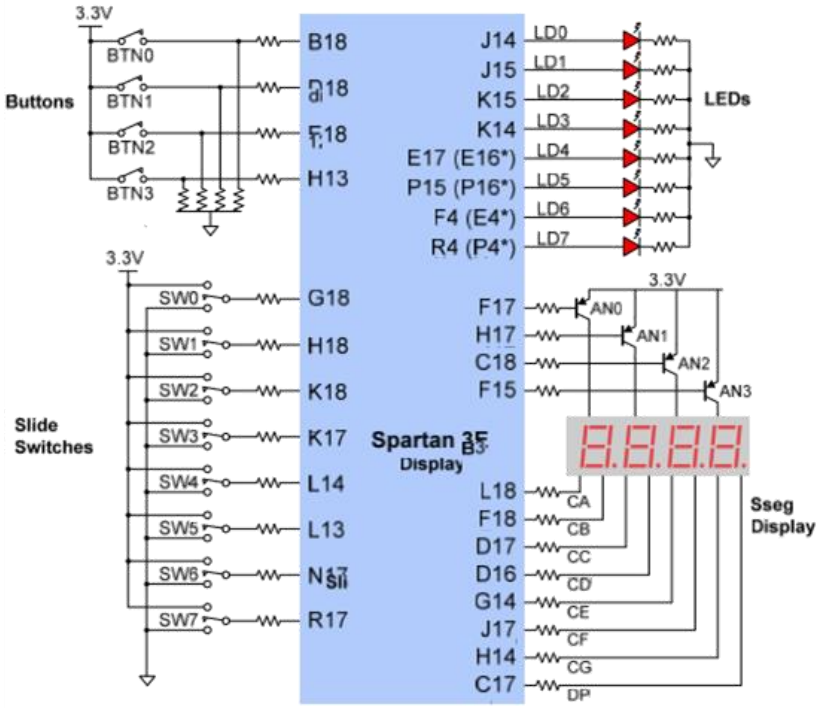
En la ventana consecuente debo seleccionar la opción de **Implementation Constraints File**, este archivo **tendrá extensión UCF** y me servirá para indicar los puertos físicos de mi tarjeta de desarrollo a los que quiero que asignen las salidas y entradas de mi código.

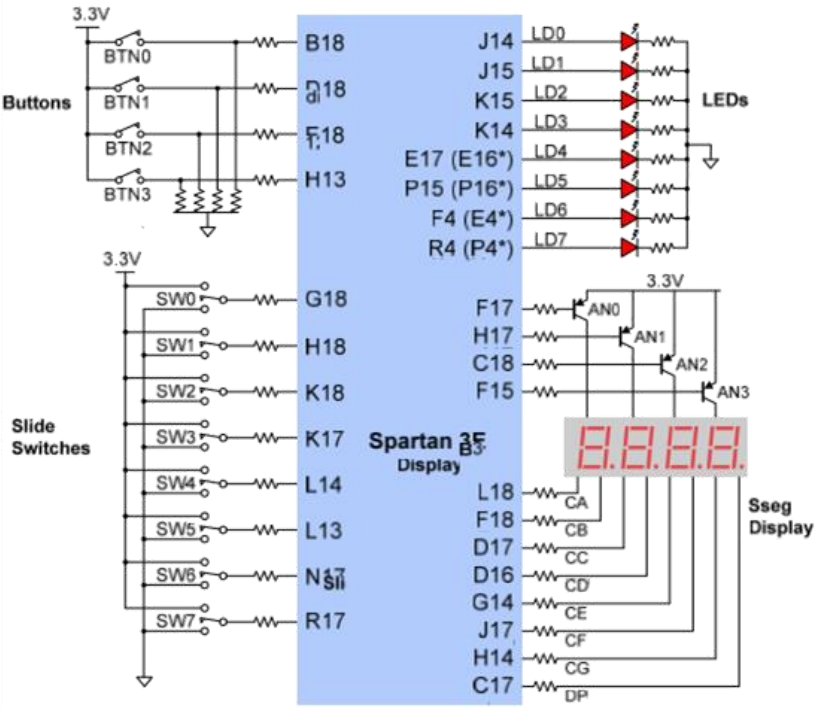


Este archivo UCF se vinculará al módulo principal de mi código ya que **por proyecto solo puede existir un con extensión UCF**, inicialmente aparecerá vacía la ventana, en ella debo escribir un código que indique exactamente qué entrada o salida quiero asignar a cada parte de tarjeta NEXYS 2 ya sea un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

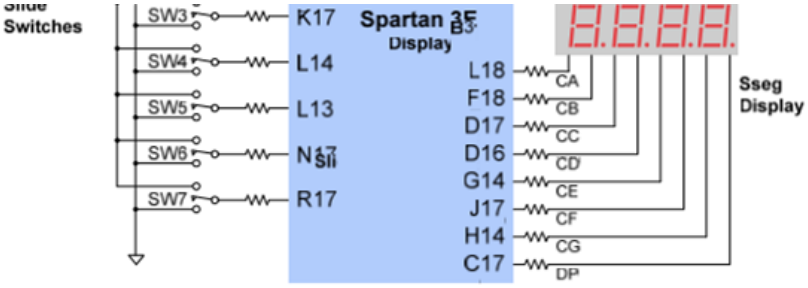
BTN0 (nombre) - **B18 (código).**
BTN1 (nombre) - **D18 (código).**
BTN2 (nombre) - **F18 (código).**
BTN3 (nombre) - **H13 (código).**

• **Leds** en la NEXYS 2:

Salidas de 1 bit:

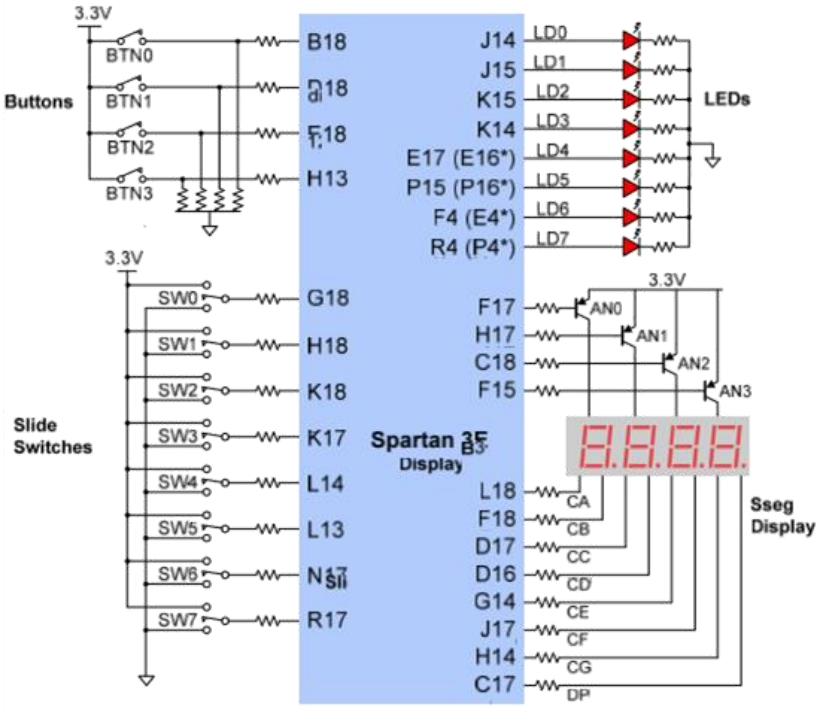
LD0 (nombre) - **J14 (código).**
LD1 (nombre) - **J15 (código).**
LD2 (nombre) - **K15 (código).**
LD3 (nombre) - **K14 (código).**
LD4 (nombre) - **E17 (código).**
LD5 (nombre) - **P15 (código).**
LD6 (nombre) - **F4 (código).**
LD7 (nombre) - **R4 (código).**

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).**



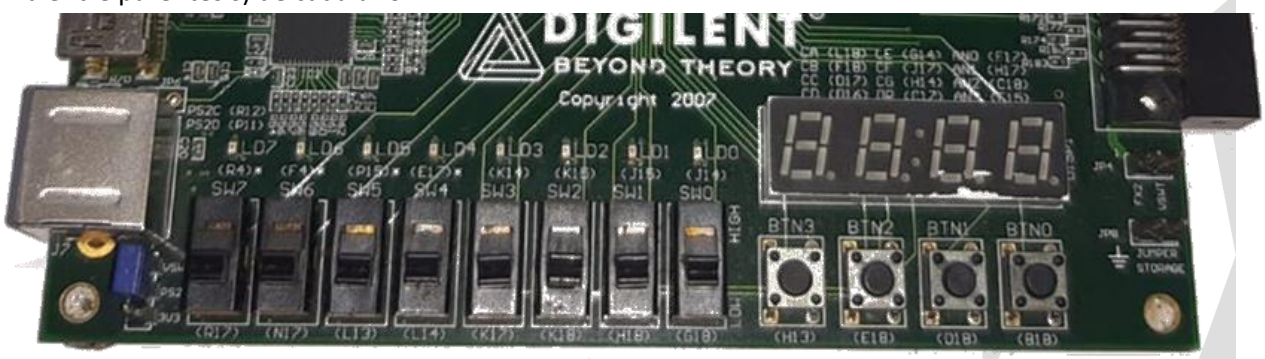
Las salidas CA, CB, …, CG y DP son de 1 bit y prenden cada led A, B, C, D, E, F, G o los Puntos de todos los displays de la tarjeta, los 4 displays de la NEXYS 2 muestran siempre la misma figura o número.

- **Leds de los displays de 7 segmentos** en la NEXYS 2:
  - **Salidas de 1 bit:**

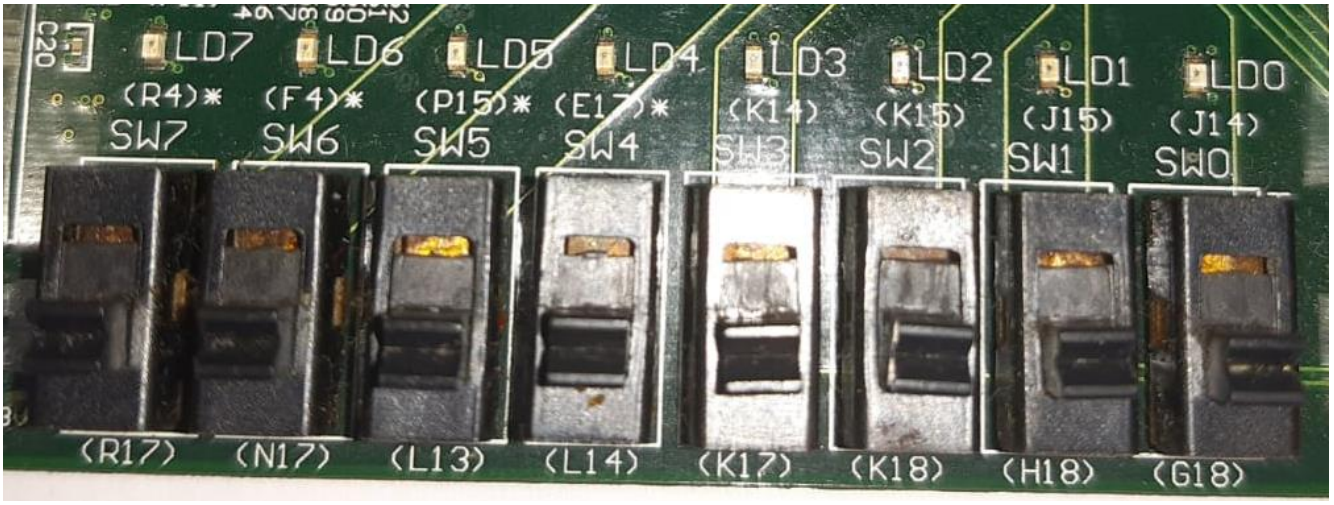
CA (nombre) – **L18 (código).**
CB (nombre) - **J15 (código).**
CC (nombre) - **K15 (código).**
CD (nombre) - **K14 (código).**
CE (nombre) - **E17 (código).**
CF (nombre) - **P15 (código).**
CG (nombre) - **F4 (código).**
DP (nombre) - **R4 (código).**

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

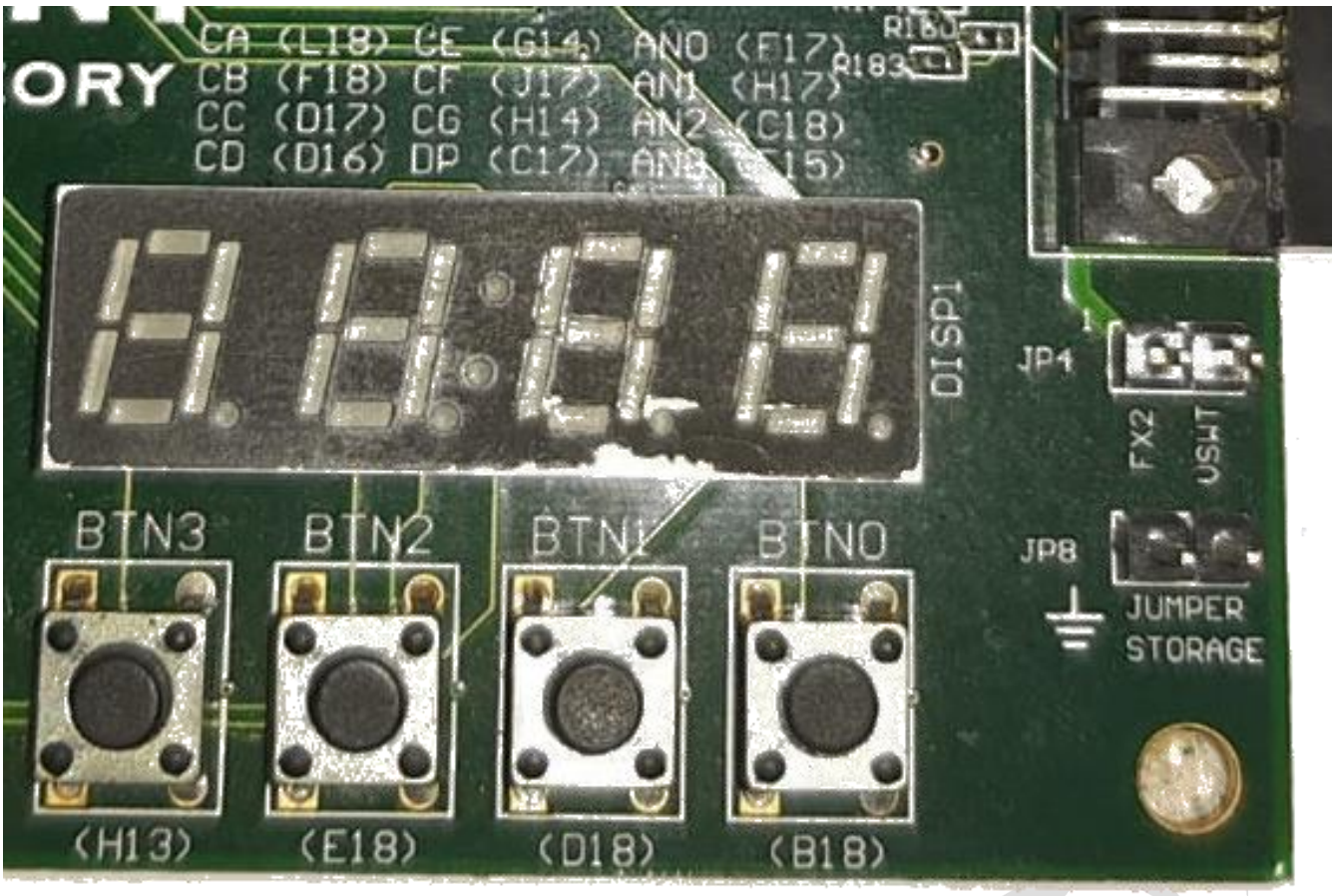
Los nombres y códigos de los **switches**, **botones**, **leds**, **ánodos comunes** y **leds de los displays de 7 segmentos** están impresos sobre la tarjeta de desarrollo por lo que no es necesario siempre checarlo en el manual o en este documento, basta con ver el nombre (que no está entre paréntesis)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

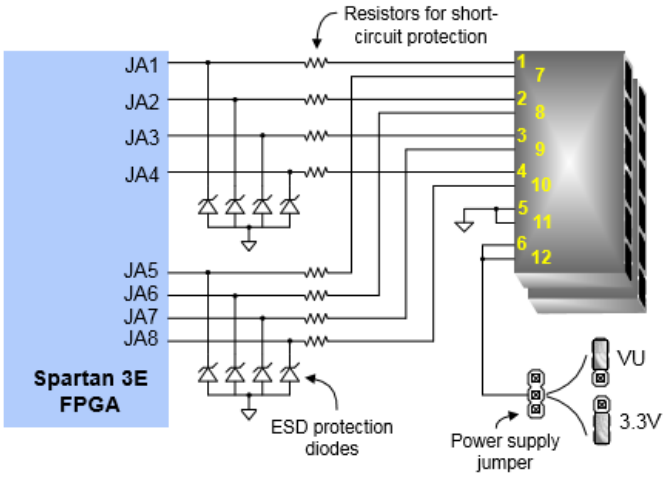
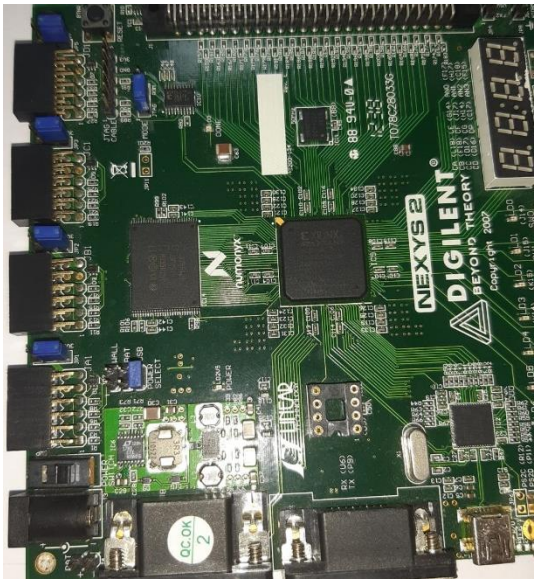


Figure 23: Nexys2 Pmod connector circuits

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[1]
JA2: K12	JA8: L16	JB2: R18	JB8: R16	JC2: J16	JC8: F14	JD2: M18	JD8: K15[2]
JA3: L17	JA9: M14	JB3: R15	JB9: T18	JC3: G13	JC9: G16	JD3: N18	JD9: J15[3]
JA4: M15	JA10: M16	JB4: T17	JB10: U18	JC4: H16	JC10: J12	JD4:; P18	JD10: J14[4]

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 especifico 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
```

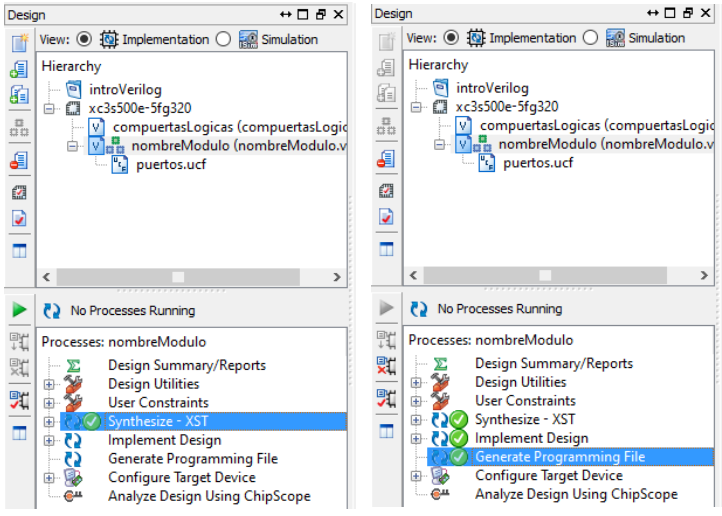
- La asignación de las entradas y salidas que sean vectores de varios bits se hacen de la siguiente manera, poniendo el nombre de la variable y entre corchetes la coordenada del bit que quiero asignar a cada salida si esta solo recibe un bit:

```
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 "salidaNOTa" loc = "P15";//Salida salidaNOTa asignada al led LD5
9  net "salidaNAND" loc = "E17";//Salida salidaNAND asignada al led LD4
10 net "salidaNOR" loc = "K14";//Salida salidaNOR asignada al led LD3
11 net "salidaXOR" loc = "K15";//Salida salidaXOR asignada al led LD2
12 net "salidaXNOR" loc = "J15";//Salida salidaXNOR asignada al led LD1
13
14 //Las salidas faltantes podrían ser dirigidas a los puertos para que enciendan leds externos
```

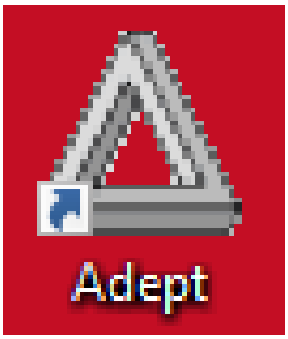
Ahora lo que debo hacer es dar clic en donde dice **Generate Programming File** dando doble clic sobre él para que se cree el archivo con extensión bit, este archivo es el que puede ser subido a la tarjeta por medio de un programa externo llamado Adept.



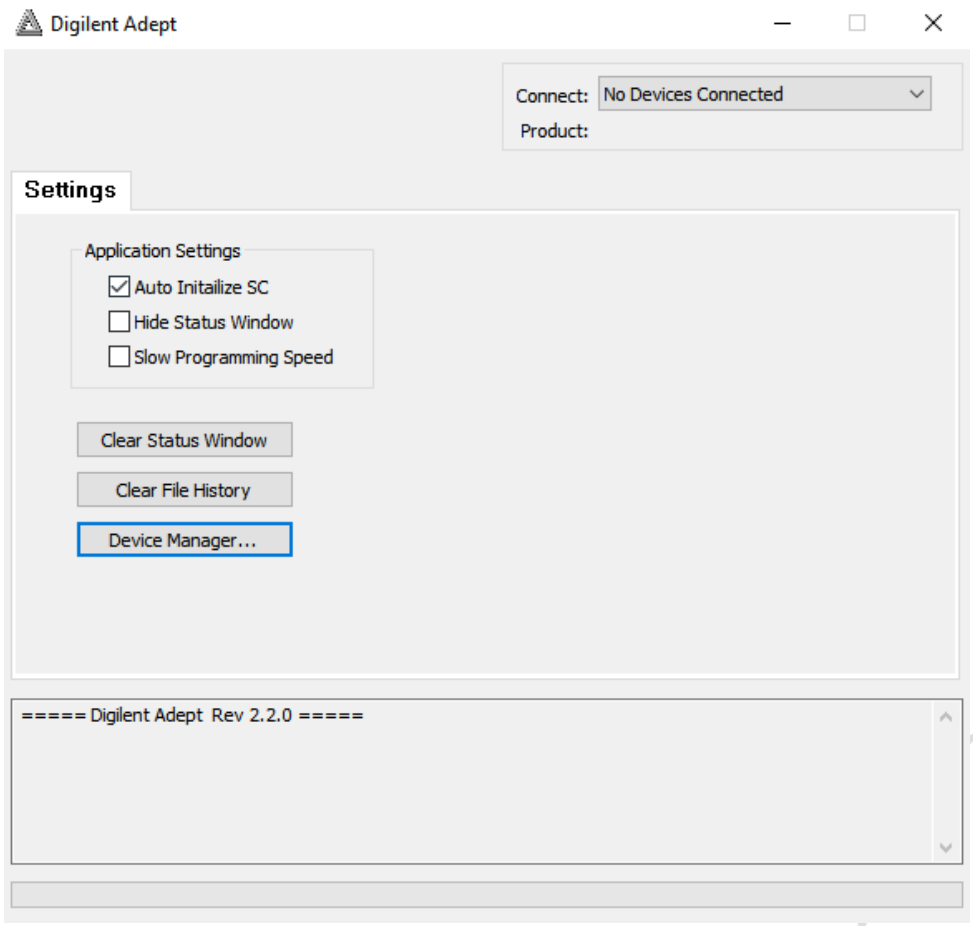
Si no hay ningún error en el código escrito en VHDL ni en el archivo ucf, los tres símbolos se mostrarán con una palomita verde.

Implementación: Adept

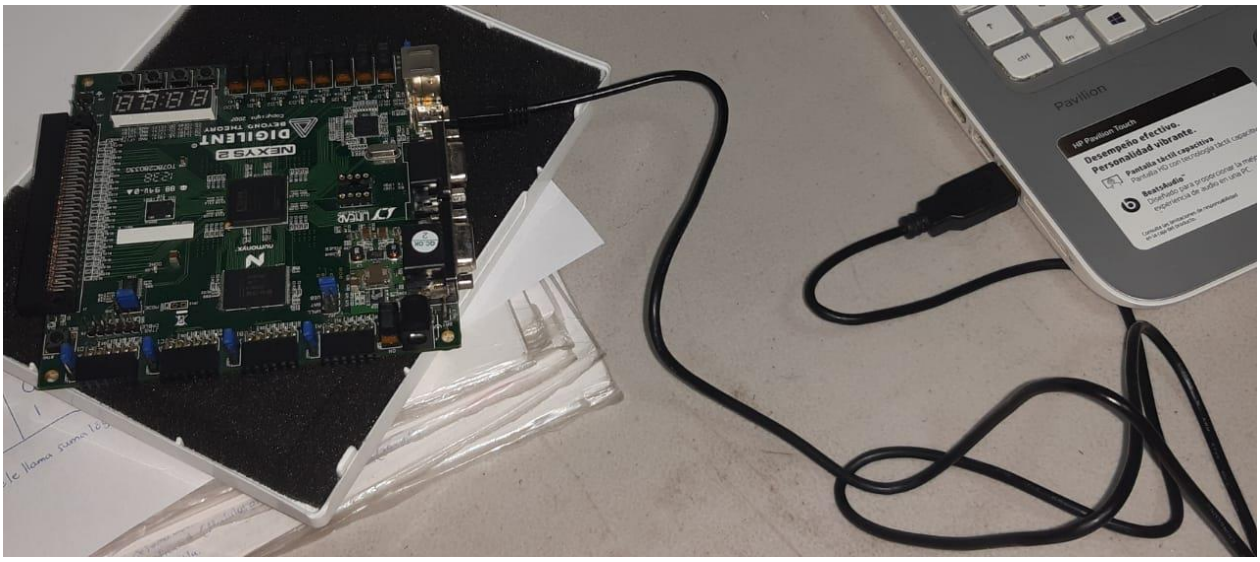
Finalmente, para que mis códigos escritos ya sea en VHDL o Verilog puedan ser subidos a mi tarjeta de desarrollo debo descargar un programa adicional llamado Adept.

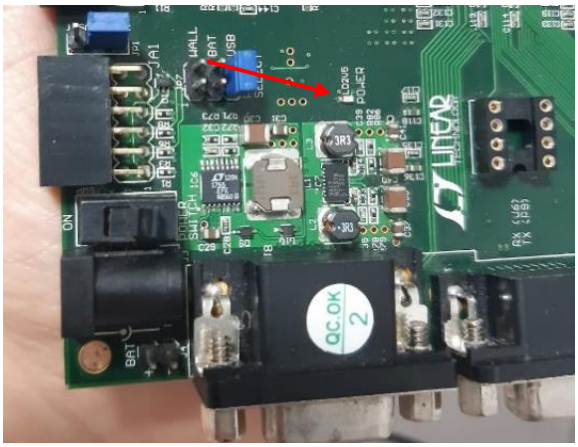


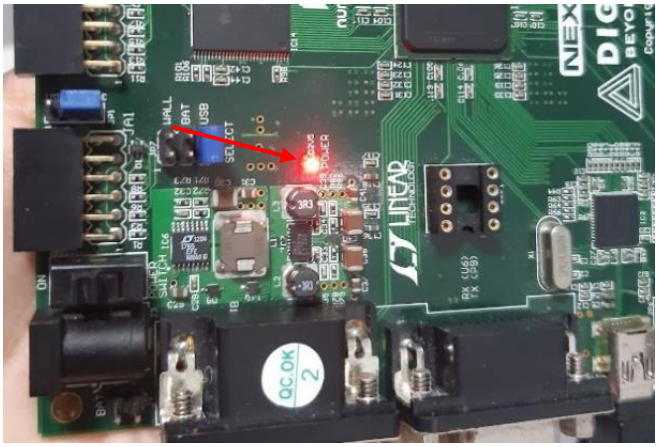
Este programa lo que hace es ver si hay alguna tarjeta de desarrollo conectada al ordenador para poder asignar a sus puertos las entradas y salidas de mi código y ejecutar las acciones programadas.



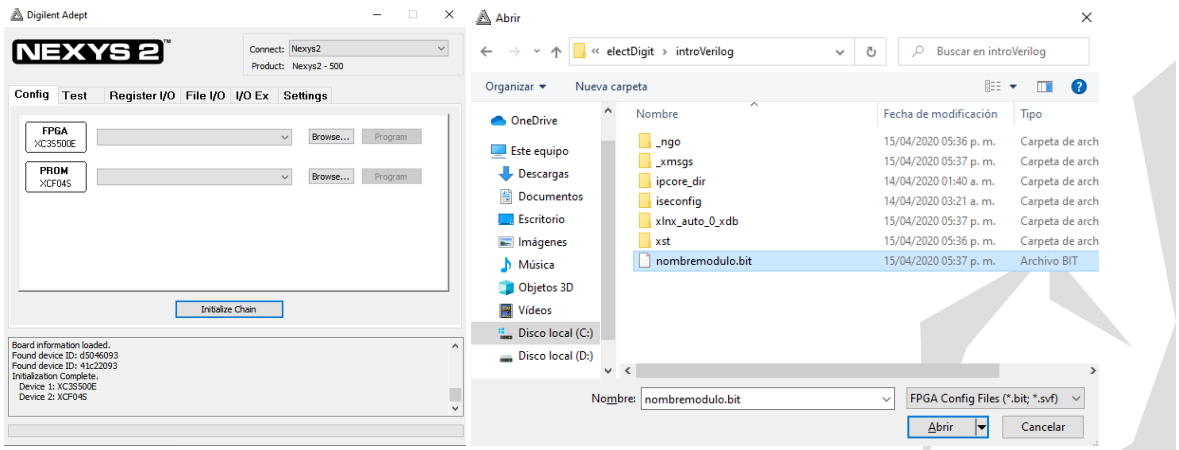
Para que el programa de Adept reconozca la tarjeta, ésta debe estar conectada vía USB a la computadora y estar encendida.



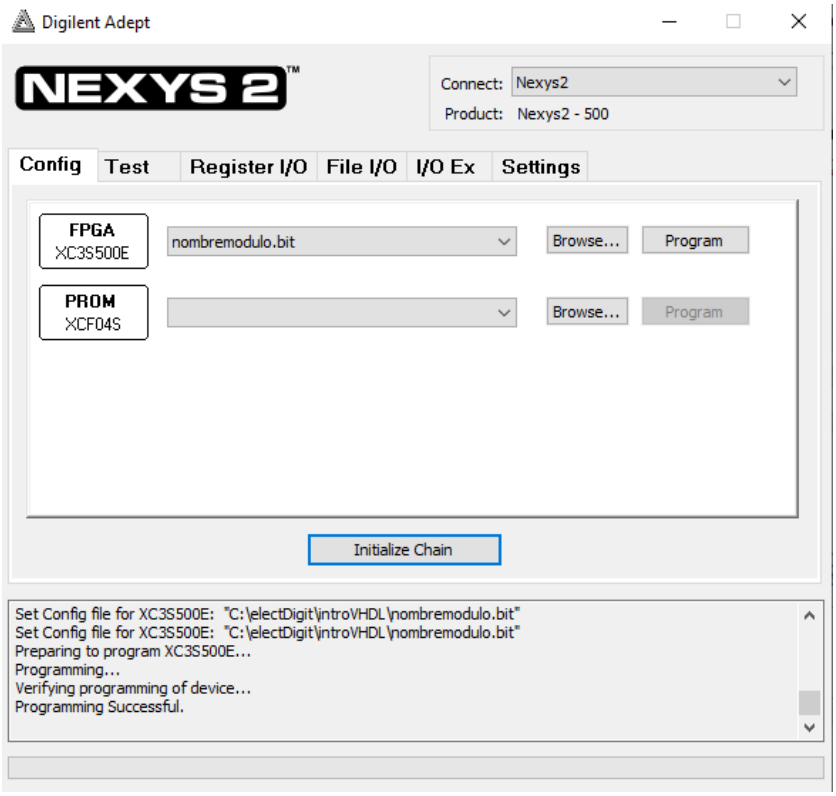




Ya que Adept haya reconocido la tarjeta que estoy usando, solo deberé tomar el archivo bit previamente obtenido del programa ISE y subirlo en donde está la opción de FPGA dando clic en el botón de Browse para buscarlo en la carpeta de mi proyecto.



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: