

SISTEMES DIGITALS (Curs 2017-2018)

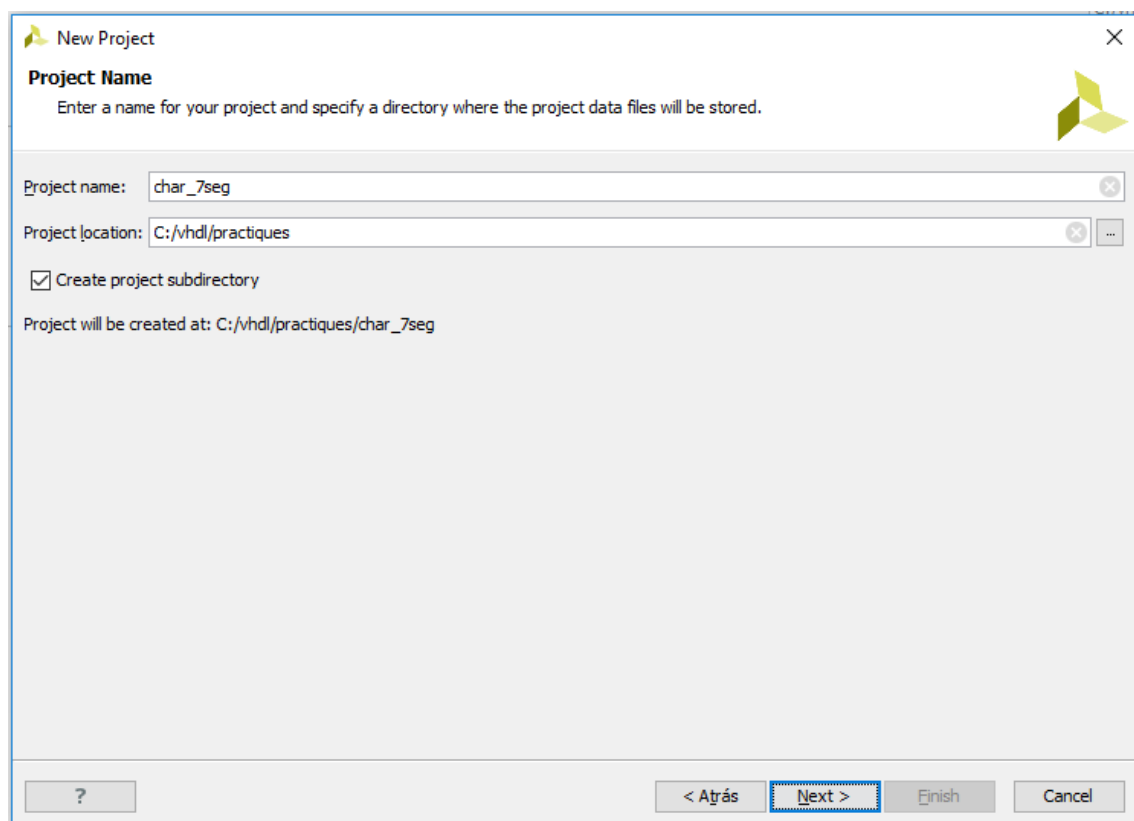
PRÀCTICA 1: Introducció a l'entorn VIVADO.

L'objectiu d'aquesta pràctica és familiaritzar-se amb l'entorn Vivado de Xilinx i les seves eines d'edició, simulació, síntesi i implementació, com també amb l'ús de la placa de desenvolupament Basys 3 de Digilent i els seus perifèrics.

La pràctica consistirà en descriure el hardware necessari per a visualitzar en un dels 4 displays de 7 segments de la Basys 3, un simple valor hexadecimal (0..F), que introduïrem mitjançant 4 interruptors corresponents als 4 bits del valor.

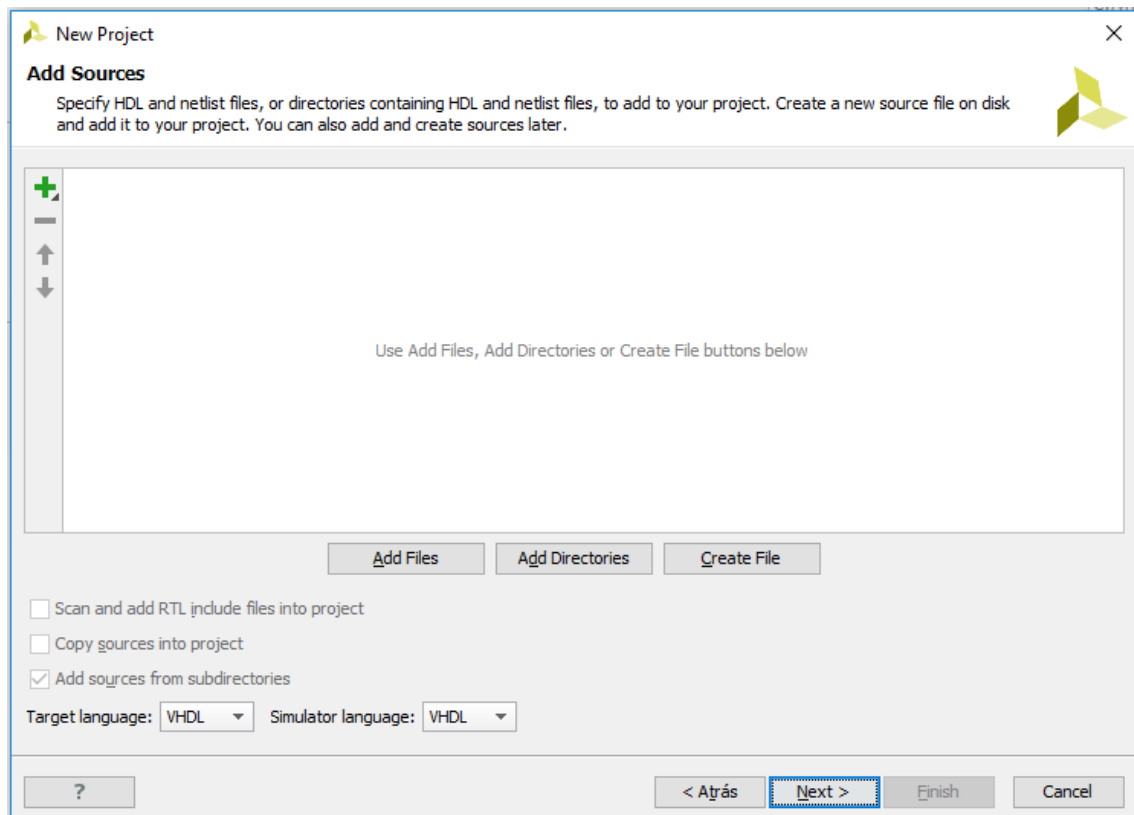
1. Creació d'un nou projecte:

Obriu l'entorn Vivado, i a la pantalla inicial, al Quick Start, cliqueu a "Create Project", Next, i us apareixerà:

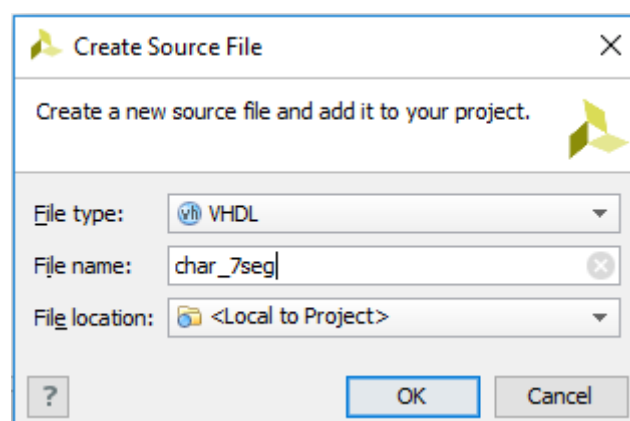


Introduïu un nom i una ruta, i Next. A la següent finestra, seleccioneu "RTL project", i si no heu seleccionat la casella "Do not specify sources at this time", quan feu Next us

apareixerà la finestra per a crear un nou arxiu VHDL (corresponent a un disseny, una entitat).

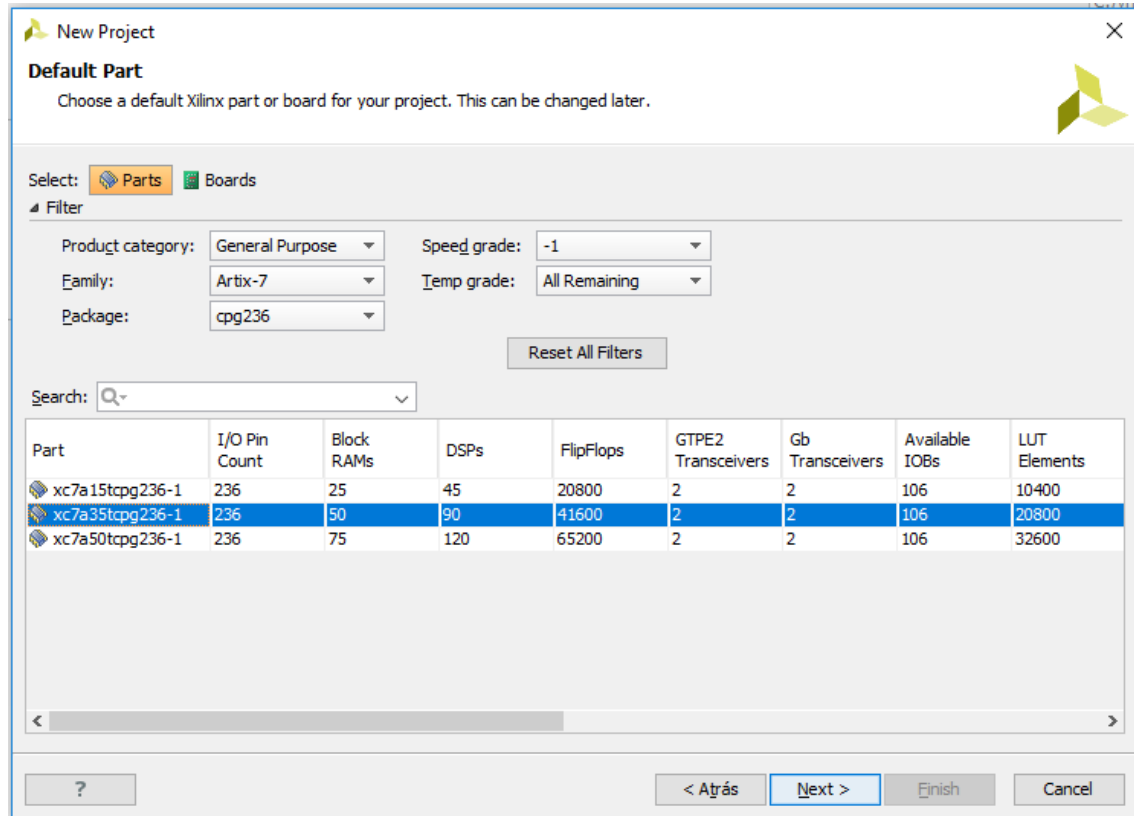


Click a Create File i crearem el nou VHDL:



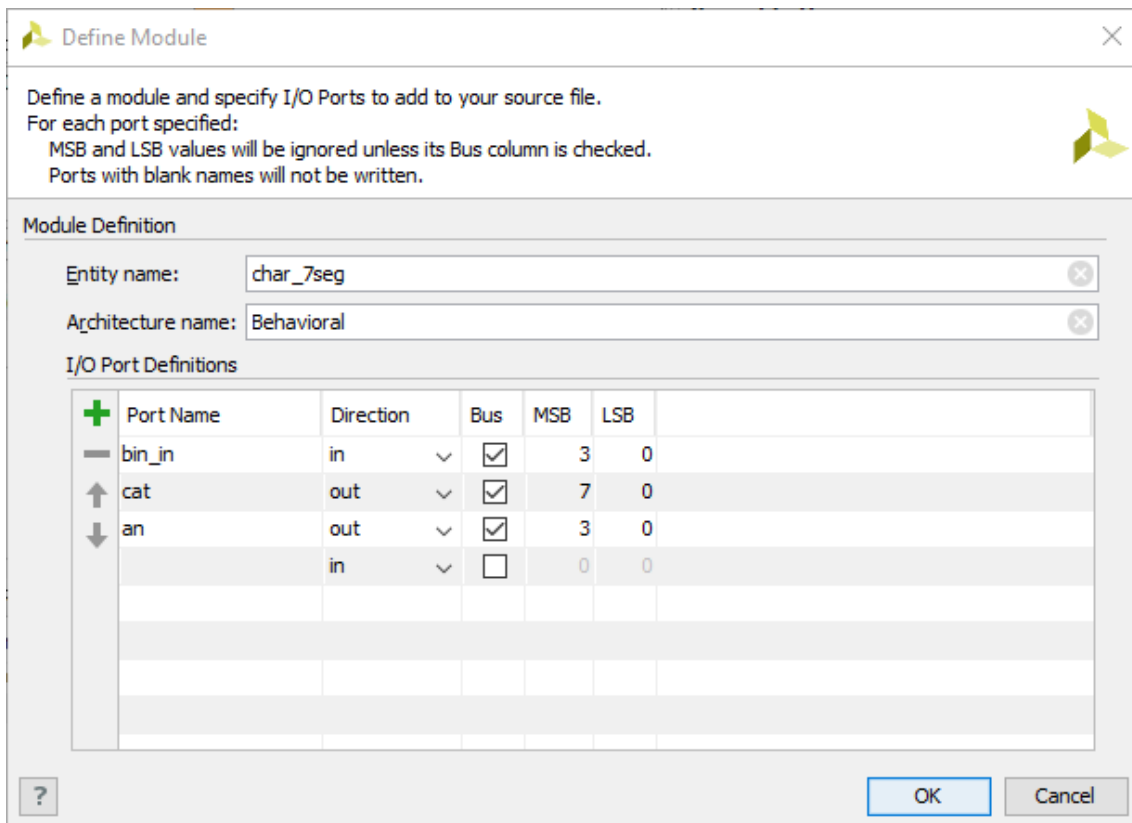
Ja que el nostre projecte no tindrà una estructura jeràrquica i només contindrà una sola entitat, li podem posar el mateix nom que al projecte.

Tot seguit, l'arxiu que hem creat apareixerà a la llista. Podem continuar... Fent next, ens saltarem les finestres corresponents a incloure al projecte IP's existents i restriccions, i finalment arribarem a la selecció del dispositiu.



La FPGA de la Basys 3 és la seleccionada a la imatge. Emprant els filtres corresponents s'hi pot arribar fàcilment. Fent Next ens apareixerà el resum del projecte, i amb Finish procedirà a la creació del projecte.

Quan apareix l'entorn del projecte, s'obrirà una finestra corresponent a la definició del mòdul (entitat, arxiu VHDL, o com vulgueu dir-li), que ens farà d'assistent per a crear l'estructura bàsica del nostre codi. Aquí simplement hem d'introduir els ports d'entrada/sortida que utilitzarem. En el nostre cas, i al ser un disseny estrictament combinatori (no seqüencial) basat en un simple codificador d'hexadecimal a 7 segments, no requereix senyal de reset ni de rellotge, simplement una entrada de 4 bits corresponent als interruptors, una sortida de 8 bits pels càtodes dels segments, i una altra de 4 bits pels seus ànodes.



Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Entity name:

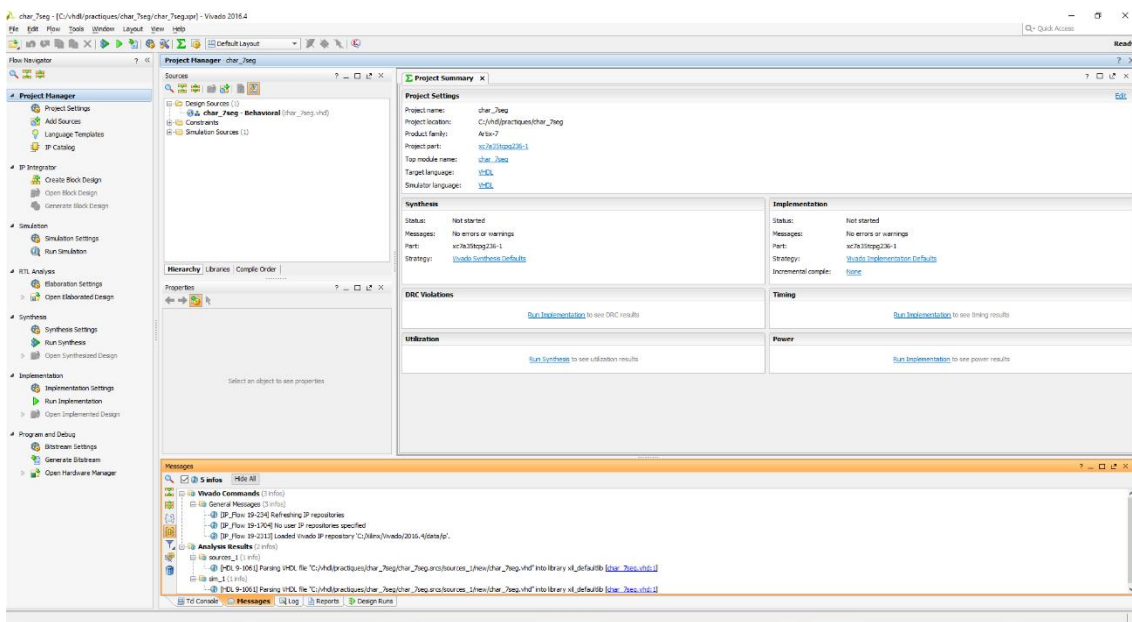
Architecture name:

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
bin_in	in	<input checked="" type="checkbox"/>	3	0
cat	out	<input checked="" type="checkbox"/>	7	0
an	out	<input checked="" type="checkbox"/>	3	0
	in	<input type="checkbox"/>	0	0

OK Cancel

De cada port, s’ha d’especificar el mode (in/out/inout), i si és de tipus escalar (un sol bit) o de tipus bus, com és el nostre cas. Pels busos també s’ha d’especificar el bit més significatiu i el menys significatiu. Clicant a OK ens apareixerà l’entorn de treball amb l’arbre del nostre projecte i el resum d’aquest en la pantalla principal:



Obrint l'arxiu vhdل que conté el nostre projecte (doble click) podrem veure que ens ha creat l'esquelet bàsic d'aquest: Llibreries, declaració de l'entitat i arquitectura :

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity char_7seg is
    Port ( bin_in : in STD_LOGIC_VECTOR (7 downto 0);
          cat : out STD_LOGIC_VECTOR (7 downto 0);
          an : out STD_LOGIC_VECTOR (3 downto 0));
end char_7seg;

architecture Behavioral of char_7seg is

begin

end Behavioral;
```

Per tant, “només” haurem d'escriure l'arquitectura que descriurà la funció del nostre disseny.

2. Descripció de l'arquitectura:

Entre el begin i el end de la nostra arquitectura introduïrem una sentència concurrent with-select que assignarà un valor als càtodes dels leds dels segments en funció del valor hexadecimal que introduïm a través dels interruptors de la placa.

Per tal d'activar només el primer dígit (el de més a la dreta del display) assignarem un '0' al seu ànode, i un '1' als dels altres dígits.

La descripció completa serà la següent:

```

entity char_7seg is
    Port ( bin_in : in STD_LOGIC_VECTOR (3 downto 0); --Interruptors
          cat : out STD_LOGIC_VECTOR (7 downto 0); --Càtodes corresponents als segments
          an : out STD_LOGIC_VECTOR (3 downto 0)); --Ànodes
end char_7seg;

architecture Behavioral of char_7seg is
begin

    with bin_in select
        cat <= "00000011" when "0000", --0
              "10011111" when "0001", --1
              "00100101" when "0010", --2
              "00001101" when "0011", --3
              "10011001" when "0100", --4
              "01001001" when "0101", --5
              "01000001" when "0110", --6
              "00011111" when "0111", --7
              "00000001" when "1000", --8
              "00011001" when "1001", --9
              "00000101" when "1010", --a
              "11000001" when "1011", --b
              "11100101" when "1100", --c
              "10000101" when "1101", --d
              "01100001" when "1110", --E
              "01110001" when "1111", --F
              "11111111" when others;

    an <= "1110";

end Behavioral;

```

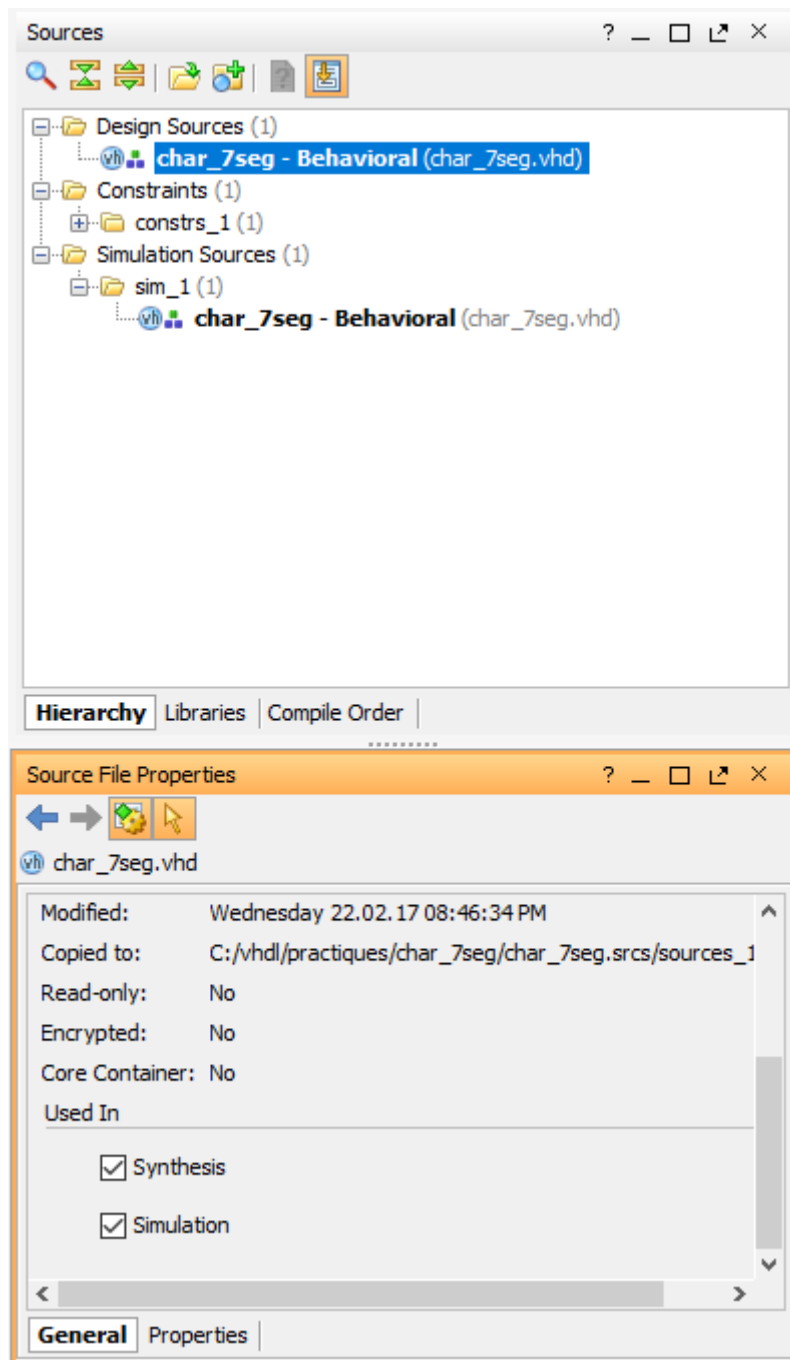
Si no hi ha cap error sintàctic (fragment de codi subratllat de color vermell), ja podem procedir al procés de simulació.

3. Simulació

Per a poder simular el funcionament de la nostra descripció tenim dues opcions:

- Simular directament el nostre mòdul i forçar el canvi de valor de les entrades “online” durant la simulació.
- Crear un test-bench que canviï de forma predeterminada el valor de les entrades cada cert temps.

Tant si optem per un mètode o l’altre, primer ens hem d’assegurar que el mòdul vhdI està configurat per ser utilitzat tant en síntesi com en simulació. Per a això només cal que despleguem la carpeta Simulation Sources dins de l’arbre del projecte i comprovem que hi és.



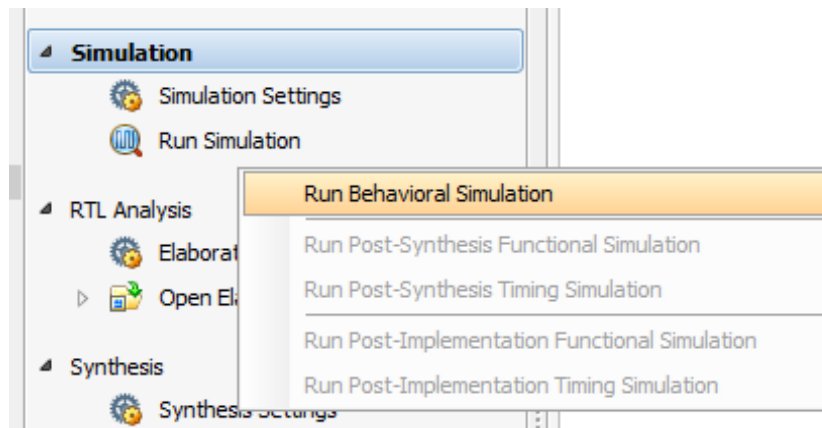
Per a modificar això, seleccionant l'arxiu a l'arbre del projecte, ens apareixeran totes les propietats de l'arxiu a la finestra "Source File Properties", on a l'apartat de "Used In" podem seleccionar on volem utilitzar-lo.

3.1. Simulació directa:

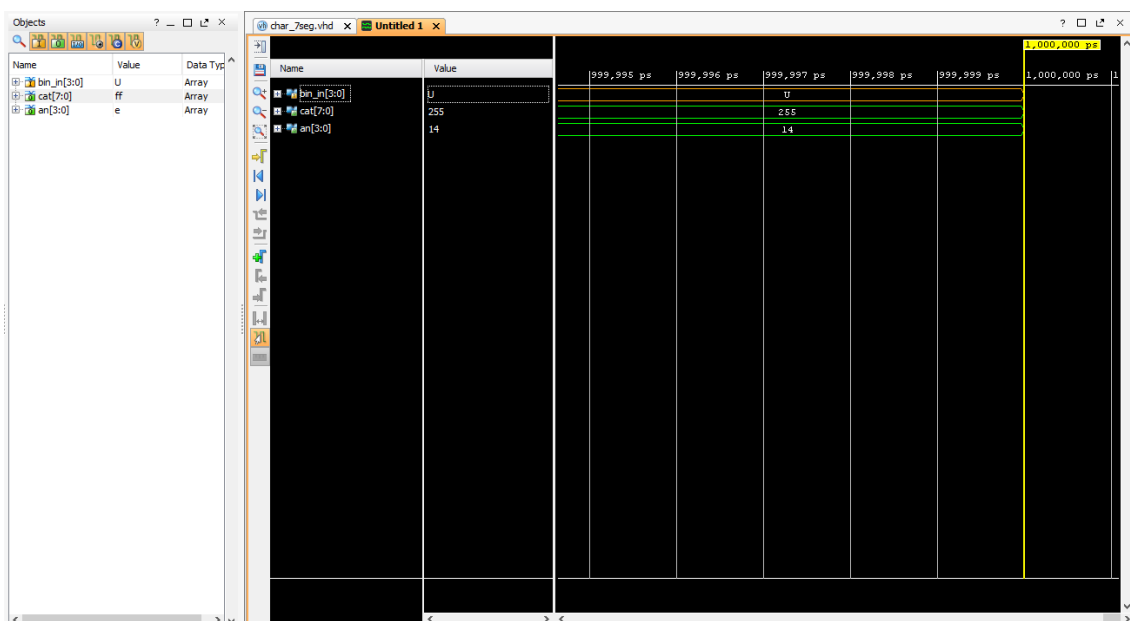
De moment, optarem per la primera opció, que de fet seria la més adequada per una descripció tan simple com aquesta, en la que no hi ha cap procés síncron ni res que depengui del temps ni de cap senyal de rellotge. A més ens estalviem la feina

d'escriure un test-bench, ja que el Vivado no inclou cap assistent per a la generació automàtica d'aquest.

A la part esquerra de la pantalla, al Flow Navigator, anirem a l'apartat de simulació, cliquem a "Run Simulation" i seleccionem "Run Behavioral Simulation".



Immediatament després s'obrirà l'entorn de simulació i es desplegarà el cronograma del primer microsegon de simulació (període que podem modificar a Simulation Settings (dins del Flow Navigator)).

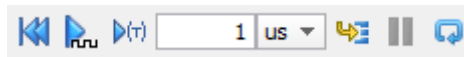


Al no haver-hi cap test-bench que assigni un valor inicial a l'entrada, aquesta mostrarà un valor 'U' (indeterminat) des del principi de la simulació fins que no forcem cap canvi. Els ànodes tindran el valor fix 14 ("1110") que li assignem de forma concurrent, i

els càtodes 255 ("11111111"), corresponent a la opció per defecte de la sentència with-select.

Pel que fa a l'entorn de simulació, farem una descripció de les eines bàsiques més utilitzades:

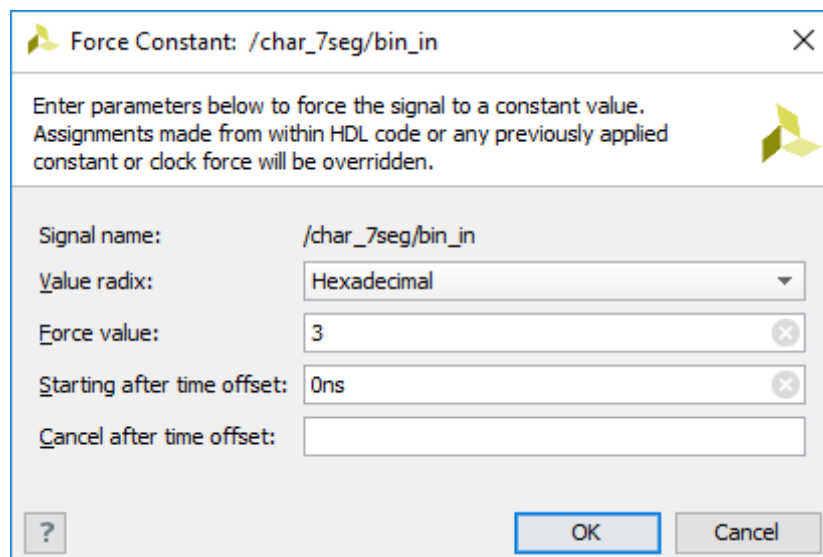
A la barra d'accions superior:



D'esquerra a dreta:

- **Restart:** Reinicia la simulació al instant 0.
- **Run All:** Inicia la simulació de forma indefinida fins que l'aturem voluntàriament.
- **Run for 'x':** Inicia la simulació per un temps 'x', establert en la finestra adjacent.
- **Step:** Fa un sol pas de simulació, sense que corri el temps.
- **Break:** Atura la simulació iniciada per Run All o Run For abans de que s'acabi el període establert.
- **Relaunch:** Torna a carregar el mòdul a simular. S'utilitza després de realitzar algun canvi en la descripció.

Per verificar el funcionament del nostre mòdul, el que podem fer és forçar l'entrada bin_in a un valor concret. Per a fer això, farem botó dret sobre el nom del senyal i seleccionarem "Force Constant":



Introduïrem el valor a Force Value i OK. Seguidament, podem fer un step per actualitzar el valor del senyal, i un Run For per un instant curt de temps (1n per exemple) per a poder comprovar que les sortides canvien al valor corresponent.



Fent “Zoom Out” (a la barra d’eines vertical), podem ajustar el cronograma a les nostres necessitats.

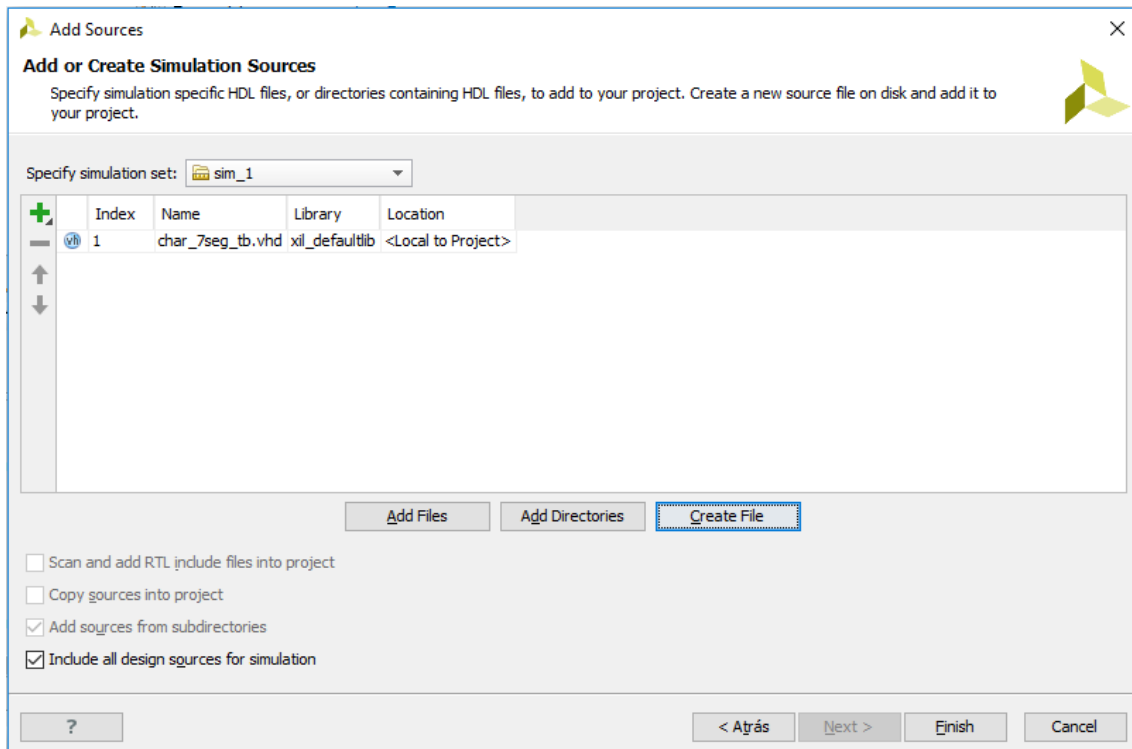
Si volem canviar el format de visualització dels valors, fent botó dret sobre el nom del senyal, seleccionem “Radix” i triem el format amb el qual poguem interpretar millor el resultat; en aquest cas potser el binari seria el més indicat.

A partir d’aquí, podem forçar l’entrada a tants valors com vulguem comprovar repetint el procés anterior.

3.2. Simulació mitjançant test-bench

Un test-bench és un mòdul d’un nivell jeràrquic superior que instancia el mòdul que nosaltres volem simular. Aquest mòdul pot tenir, o no, ports; en el nostre cas només utilitzarem senyals interns connectats als ports del mòdul a simular, i on aplicarem els estímuls necessaris per a comprovar el comportament d’aquest.

Per crear un test-bench, farem doble click a la finestra de “Sources” (arbre del projecte) i “Add Sources”. Seleccionarem “Add or Create Simulation Sources”. Cliquem a Create i se’ns obrirà la finestra per a crear un nou arxiu (un VHDL en el nostre cas). Li posarem el nom char_7seg_tb i seguirem.



Tot seguit s'obrirà la finestra de "Define Module", on no hem de modificar res, ja que el nostre test-bench no disposarà de ports.

Ja creat l'arxiu, ens apareixerà a la carpeta de "Simulation Sources" juntament amb el mòdul char_7seg. Ara el que hem de fer és escriure el codi del test-bench, que consisteix en una arquitectura estructural en que s'instancia el mòdul i es canvia el valor de l'entrada de forma cíclica mitjançant un loop dins d'un procés.

El codi seria el següent:

```
entity char_7seg_tb is
-- Port ( );
end char_7seg_tb;

architecture Behavioral of char_7seg_tb is

signal bin_in : std_logic_vector(3 downto 0);
signal cat : std_logic_vector(7 downto 0);
signal an : std_logic_vector(3 downto 0);

component char_7seg
port ( bin_in : in std_logic_vector(3 downto 0);
      cat : out std_logic_vector(7 downto 0);
      an : out std_logic_vector(3 downto 0));
end component;
```

```

begin

uut: char_7seg
    port map ( bin_in => bin_in,
               cat => cat,
               an => an);

test_entrades: process
begin
    for i in 0 to 15 loop
        bin_in <= std_logic_vector(to_unsigned(i,4));
        wait for 10 ns;
    end loop;
end process;

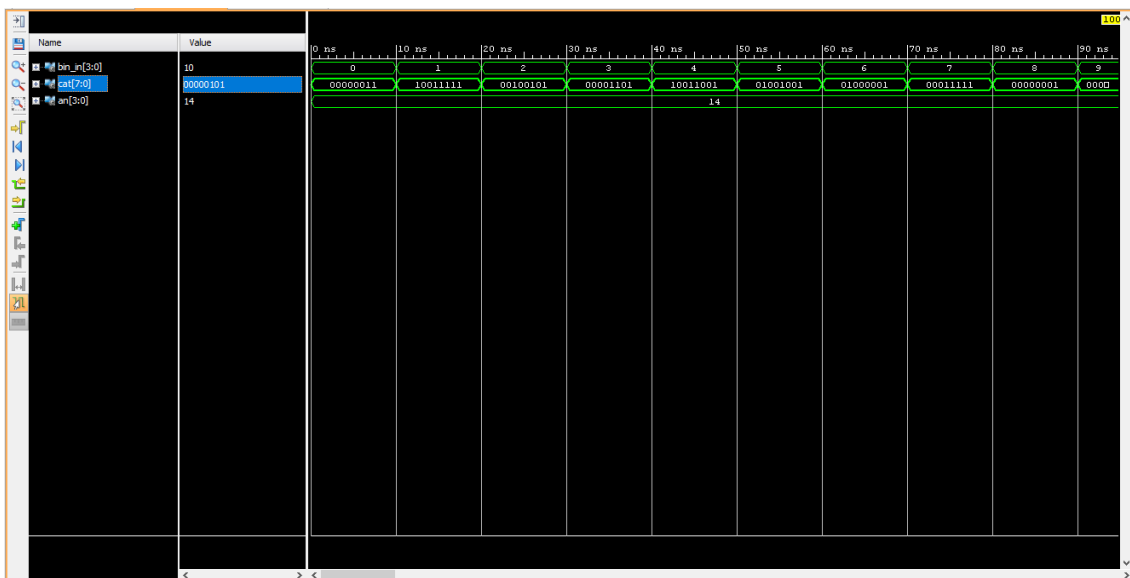
end Behavioral;

```

És important que afegiu la llibreria numeric (*use IEEE.NUMERIC_STD.ALL;*) que apareix comentada a l'inici del codi, ja que és la que conté la descripció de la funció de conversió *to_unsigned* que utilitzem en el loop.

Quan acabem d'escriure el codi, el guardem, i si no hi ha errors sintàctics ens apareixerà a l'arbre del projecte al top level de la jerarquia dels arxius de simulació, incloent dins seu el mòdul a simular (*char_7seg*).

Procedirem a la simulació mitjançant "Run Behavioral Simulation" i podrem observar el següent cronograma, on podem comprovar que per a cada valor de l'entrada, el port dels càtodes treu el valor corresponent, i el dels ànodes queda invariant.



Si hi ha errors sintàctics en el test-bench, al ser un mòdul no utilitzat per a la síntesi, no se'ns mostraran subratllats de color vermell. Al gravar l'arxiu, ens quedarà inclòs dins d'un conjunt anomenat "Syntax Error Files" dins de l'arbre del projecte. Per a poder-los identificar, els trobarem detallats dins de la finestra inferior de l'entorn, seleccionant la pestanya "Messages", a l'apartat "Analysis Results".

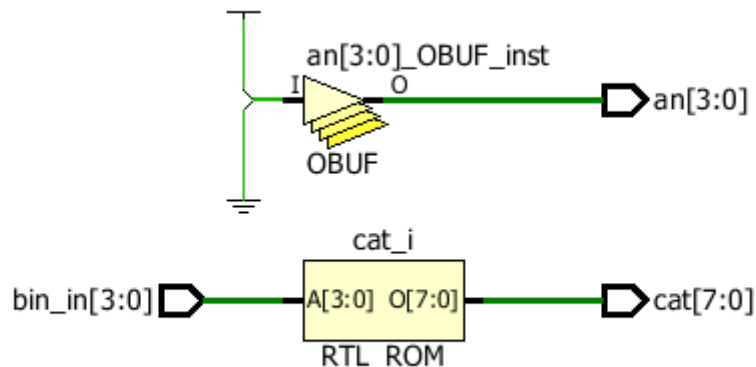
Si la sintaxi és correcta però té errors funcionals que impossibilitin la simulació, el procés s'aturarà amb un missatge d'error. Podrem trobar les causes d'aquest a la l'històric de la consola TCL (finestra inferior, pestanya Tcl Console).

Més informació sobre el simulador al document *Logic Simulation* (a Atenea).

4. RTL Analysis

Aquest entorn ens mostrarà el disseny a nivell RTL (Register Transfer Level), és a dir, a nivell funcional abans de ser sintetitzat.

Clicant a "Open Elaborated Design" ens obrirà l'entorn corresponent. Seleccionant "Schematic" podrem veure el disseny funcional com a esquema de blocs. Podríem dir que aquest apartat ens mostra el que el Vivado ha interpretat del nostre codi.



Dins del RTL Analysis hi ha un seguit de funcionalitats (Report Methodology, DRC and Noise) que permeten fer diferents anàlisis del nostre disseny, tenint en compte unes certes regles (Rules) que nosaltres hem fixat anteriorment. Al nivell d'aquest curs no entrarem en detalls sobre aquestes funcions. Si voleu aprofundir més, podeu consultar el document *Design Analysis and Closure Techniques*.

Quan s'obre el disseny elaborat, a la finestra inferior s'hi inclou una pestanya anomenada I/O ports, en la qual nosaltres podrem assignar els ports del nostre mòdul a pins físics de la FPGA, i on també podrem configurar algunes característiques elèctriques d'aquests. Si no hi surt, anirem al menú "Layout" i farem click a "I/O Planning".

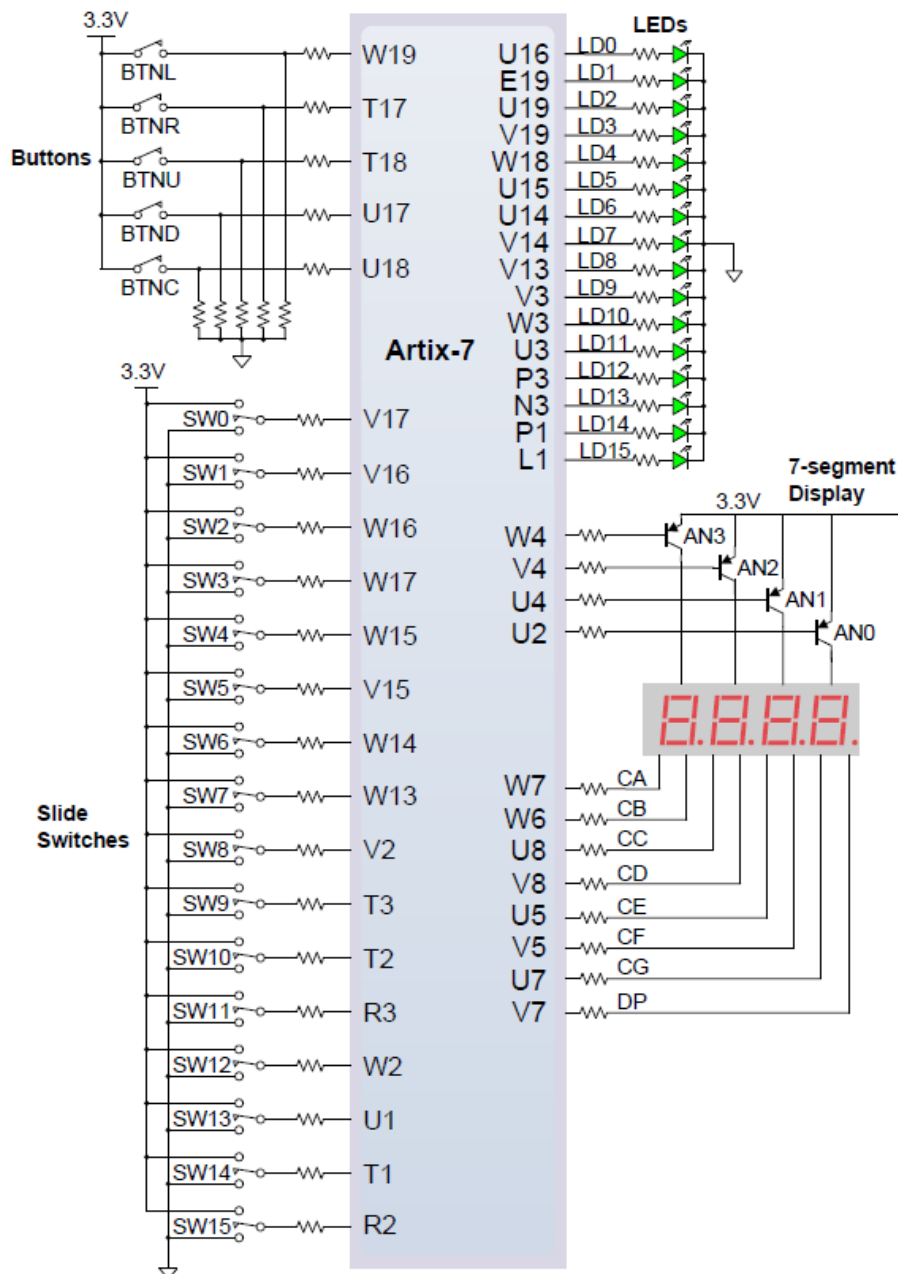
Aquestes correspondències entre ports, pins i el seu nivell elèctric ha de constar en un arxiu de “Constraints” d’extensió XDC. Per a crear aquest arxiu, farem botó dret a la finestra de “Sources” i crearem un nou arxiu de constraints que automàticament quedarà ubicat en l’apartat de Constraints de l’arbre del projecte. Li podeu posar de nom, per exemple, char_7seg_const. Un cop creat editarem els ports com s’indica a la següent figura:

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std
an (4)	OUT			✓		34 LVCMOS33*
an[3]	OUT		W4	✓		34 LVCMOS33*
an[2]	OUT		V4	✓		34 LVCMOS33*
an[1]	OUT		U4	✓		34 LVCMOS33*
an[0]	OUT		U2	✓		34 LVCMOS33*
bin_in (4)	IN			✓		14 LVCMOS33*
bin_in[3]	IN		W17	✓		14 LVCMOS33*
bin_in[2]	IN		W16	✓		14 LVCMOS33*
bin_in[1]	IN		V16	✓		14 LVCMOS33*
bin_in[0]	IN		V17	✓		14 LVCMOS33*
cat (8)	OUT			✓		34 LVCMOS33*
cat[7]	OUT		W7	✓		34 LVCMOS33*
cat[6]	OUT		W6	✓		34 LVCMOS33*
cat[5]	OUT		U8	✓		34 LVCMOS33*
cat[4]	OUT		V8	✓		34 LVCMOS33*
cat[3]	OUT		U5	✓		34 LVCMOS33*
cat[2]	OUT		V5	✓		34 LVCMOS33*
cat[1]	OUT		U7	✓		34 LVCMOS33*
cat[0]	OUT		V7	✓		34 LVCMOS33*
Scalar ports (0)						

Aquí ens limitarem a omplir els camps “Package Pin” (pin físic de la FPGA), i el I/O Std (tecnologia i nivell de voltatge)

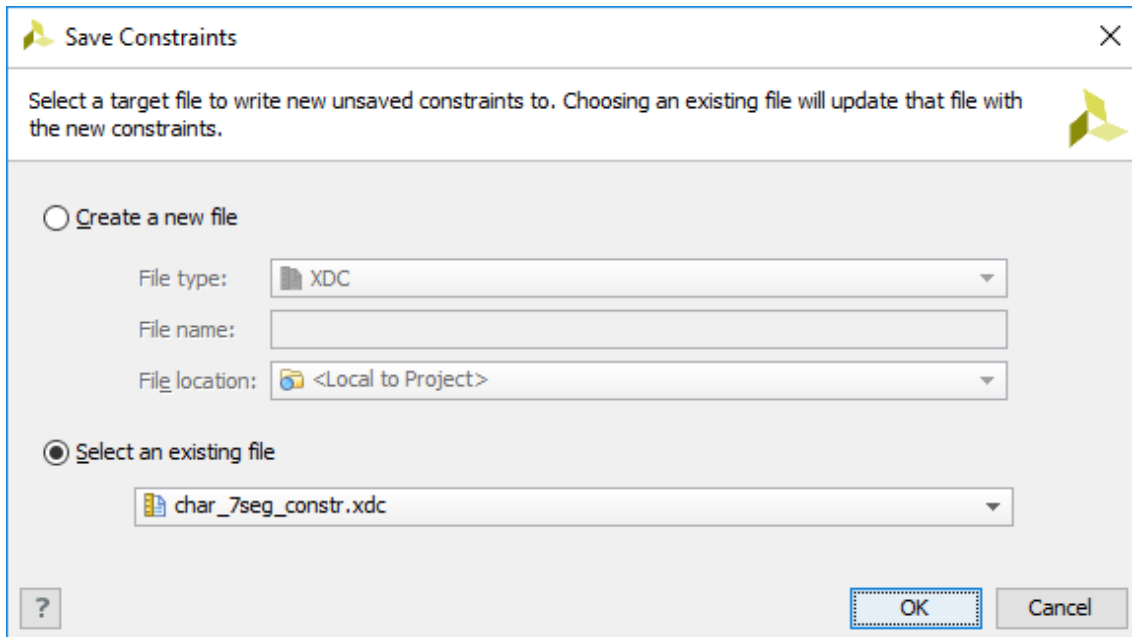
Pel que fa al I/O Std, seran tots LVCMOS33 (Low Voltage CMOS 3.3V).

Per poder localitzar la connexió entre els pins de la FPGA i els perifèrics de la placa Basys 3, ens basarem en la informació que ens dona el document *Basys 3 FPGA Board Reference Manual*, que trobareu penjat a Atenea. En el nostre cas:



...on utilitzarem els SW[0..3] (interruptors), AN[0..3] (ànodes) i C[A..G] i DP (càtodes dels segments i DOT).

Un cop completada l'assignació de pins, guardarem amb Ctrl+S (o la icona corresponent) i ens apareixerà una finestra on podrem triar si crear un arxiu xdc nou o bé utilitzar l'existent. Òbviament optarem per la segona opció.



The dialog box is titled "Save Constraints" and contains instructions: "Select a target file to write new unsaved constraints to. Choosing an existing file will update that file with the new constraints." It has two main sections: "Create a new file" (unselected) and "Select an existing file" (selected). The "Create a new file" section includes fields for "File type" (set to "XDC"), "File name" (empty), and "File location" (set to "<Local to Project>"). The "Select an existing file" section has a dropdown menu showing "char_7seg_constr.xdc". At the bottom are buttons for "?", "OK", and "Cancel".

Ja guardat, l'obrirem fent doble click sobre el mateix a la finestra de Sources i veurem el següent codi:

```

1 set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
2 set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
3 set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
4 set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
5 set_property IOSTANDARD LVCMOS33 [get_ports {bin_in[3]}]
6 set_property IOSTANDARD LVCMOS33 [get_ports {bin_in[2]}]
7 set_property IOSTANDARD LVCMOS33 [get_ports {bin_in[1]}]
8 set_property IOSTANDARD LVCMOS33 [get_ports {bin_in[0]}]
9 set_property IOSTANDARD LVCMOS33 [get_ports {cat[7]}]
10 set_property IOSTANDARD LVCMOS33 [get_ports {cat[6]}]
11 set_property IOSTANDARD LVCMOS33 [get_ports {cat[5]}]
12 set_property IOSTANDARD LVCMOS33 [get_ports {cat[4]}]
13 set_property IOSTANDARD LVCMOS33 [get_ports {cat[3]}]
14 set_property IOSTANDARD LVCMOS33 [get_ports {cat[2]}]
15 set_property IOSTANDARD LVCMOS33 [get_ports {cat[1]}]
16 set_property IOSTANDARD LVCMOS33 [get_ports {cat[0]}]
17 set_property PACKAGE_PIN U2 [get_ports {an[0]}]
18 set_property PACKAGE_PIN U4 [get_ports {an[1]}]
19 set_property PACKAGE_PIN V4 [get_ports {an[2]}]
20 set_property PACKAGE_PIN W4 [get_ports {an[3]}]
21 set_property PACKAGE_PIN V17 [get_ports {bin_in[0]}]
22 set_property PACKAGE_PIN V16 [get_ports {bin_in[1]}]
23 set_property PACKAGE_PIN W16 [get_ports {bin_in[2]}]
24 set_property PACKAGE_PIN W17 [get_ports {bin_in[3]}]
25 set_property PACKAGE_PIN V7 [get_ports {cat[0]}]
26 set_property PACKAGE_PIN U7 [get_ports {cat[1]}]
27 set_property PACKAGE_PIN V5 [get_ports {cat[2]}]
28 set_property PACKAGE_PIN U5 [get_ports {cat[3]}]
29 set_property PACKAGE_PIN V8 [get_ports {cat[4]}]
30 set_property PACKAGE_PIN U8 [get_ports {cat[5]}]
31 set_property PACKAGE_PIN W7 [get_ports {cat[7]}]
32 set_property PACKAGE_PIN W6 [get_ports {cat[6]}]
33

```


De forma alternativa a tot aquest procés, Digilent disposa al seu repositori de GitHub d'un arxiu XDC complet amb les assignacions i nivells de tots els pins de la FPGA que s'utilitzen en la tarja Basys 3. L'arxiu s'anomena **Basys3_Master.xdc** i el podreu trobar penjat a Atenea. En aquest arxiu, per a cada port utilitzat, hi ha una línia que defineix el PACKAGE_PIN (pin de la FPGA) i una altra IOSTANDARD (nivell de voltatge), les dues comentades mitjançant el caràcter '#'. Si optem per aquest arxiu, haurem d'activar les línies de codi corresponents a tots els ports que utilitzem, eliminant el '#' del principi de cada línia. A més, haurem de tenir en compte que el nom dels ports del nostre mòdul haurà de coincidir amb el nom utilitzat en aquest arxiu.

Els avantatges d'aquesta opció són que no haurem de consultar la documentació de la placa per a fer l'assignació de pins, i que serà més difícil cometre errors durant aquest procés.

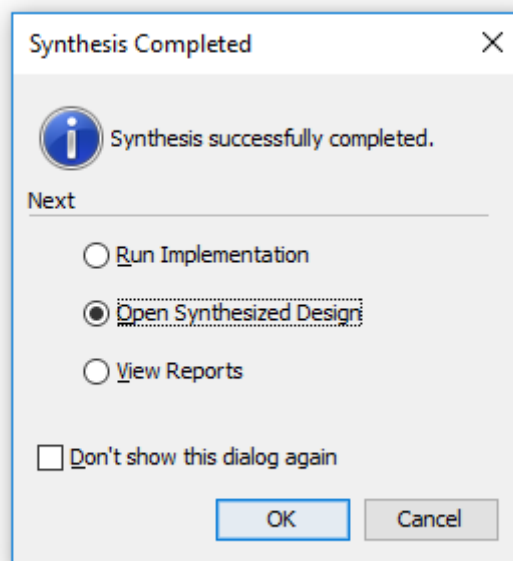
```
81 ##7 segment display
82 set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
83     set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
84 set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
85     set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
86 set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
87     set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
88 #set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
89     #set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
90 #set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
91     #set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
92 #set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
93     #set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
94 #set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
95     #set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
96
97 #set_property PACKAGE_PIN V7 [get_ports dp]
98     #set_property IOSTANDARD LVCMOS33 [get_ports dp]
99
100 #set_property PACKAGE_PIN U2 [get_ports {an[0]}]
101     #set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
102 #set_property PACKAGE_PIN U4 [get_ports {an[1]}]
103     #set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
104 #set_property PACKAGE_PIN V4 [get_ports {an[2]}]
105     #set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
106 #set_property PACKAGE_PIN W4 [get_ports {an[3]}]
107     #set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
108
109
110 ##Buttons
111 #set_property PACKAGE_PIN U18 [get_ports btnC]
112     #set_property IOSTANDARD LVCMOS33 [get_ports btnC]
```

5. Síntesi

La síntesi és un procés que consisteix en transformar un disseny RTL (Register Transfer Level) a una representació a nivell de portes lògiques (Gate-Level). Interpreta el nostre disseny funcional i el sintetitza utilitzant els recursos que conté la FPGA per tal que aquesta es comporti com nosaltres desitgem.

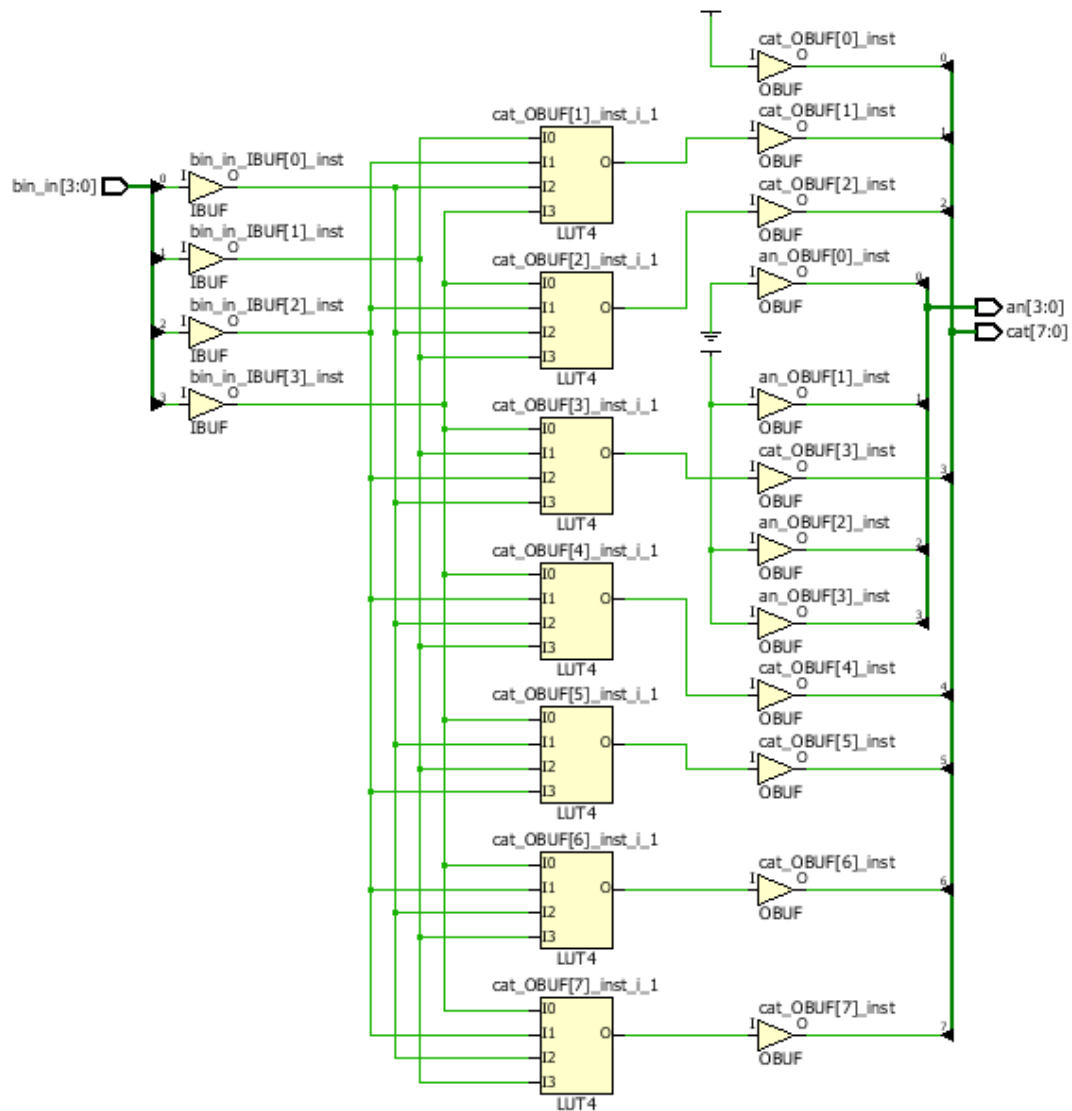
Tant en la síntesi com en la implementació, la durada del procés depèn de la velocitat del processador de la nostra màquina i del nombre de threads (tasques en paral·lel) que aquest pot realitzar. Per tant, depenent de la complexitat del nostre disseny i de la capacitat de càlcul del nostre ordinador, això pot durar des d'uns pocs segons fins a varis minuts. D'aquí la importància de la simulació. Si enlloc de simular una descripció volem provar-la a la tarja després de cada canvi que li fem, podem perdre molt temps en aquests processos, a part de ignorar errors de funcionament difícils de veure en el funcionament final. Per molt simple que sigui el nostre mòdul, val la pena assegurar un bon resultat en simulació abans de continuar amb els següents processos.

A l'apartat de "Synthesis" farem click a "Run Synthesis" i s'iniciarà el procés. Quan acabi, s'obrirà una finestra on podrem elegir la següent acció a realitzar:



Seleccionarem "Open Synthesized Design".

Entre totes les opcions (Reports i edició de Constraints) disponibles en l'entorn del disseny sintetitzat, ens fixarem en la última (Schematic), on podem veure la diferència entre l'esquema a nivell RTL vist anteriorment i a nivell gate.



En aquest esquema ja apareixen els recursos de hardware utilitzats en la FPGA, bàsicament buffers i look up tables.

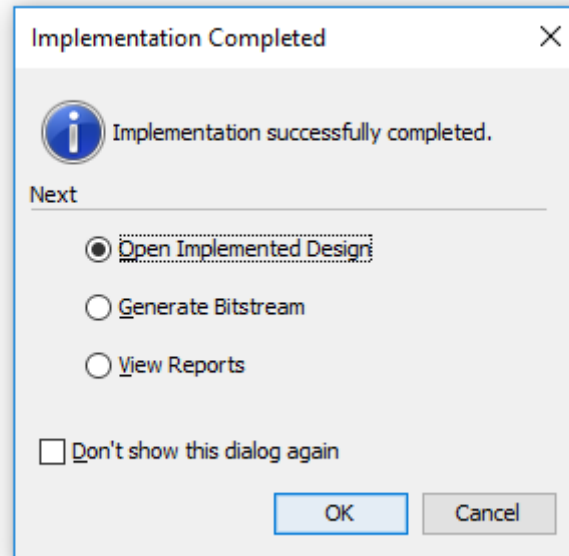
A la finestra inferior, seleccionant la pestanya de Messages, podem veure tots els Warnings i Errors que s'hagin generat durant la síntesi.

Per a més informació sobre el procés de síntesi de Vivado, teniu el document *Synthesis* a Atenea.

6. Implementació

El procés d'implementació consisteix en ubicar i "rutejar" (place and route) el netlist del disseny ja sintetitzat a nivell de portes, dins la nostra FPGA, tenint en compte totes les regles i restriccions que haguem fixat.

A l'apartat de "Implementation" farem click a "Run Implementation" i s'iniciarà el procés. Quan acabi, s'obrirà una finestra on podrem elegir la següent acció a realitzar:



Seleccionem "Open Implemented Design".

Veurem que a l'apartat d'implementació apareixen exactament les mateixes eines que a l'apartat de Síntesi, tret de l'esquemàtic.

De la mateixa manera que en el disseny sintetitzat, revisarem la finestra de Messages per si hi ha algun Warning o Error greu.

Per a més informació sobre el procés d'implementació, podeu consultar el document *Implementation*, que podeu trobar a Atenea.

7. Program and Debug

Aquest és l'apartat que ens permetrà generar l'arxiu de configuració, connectar-nos via USB amb la nostra tarja, i finalment configurar la FPGA.

Per a fer això tenim dues opcions:

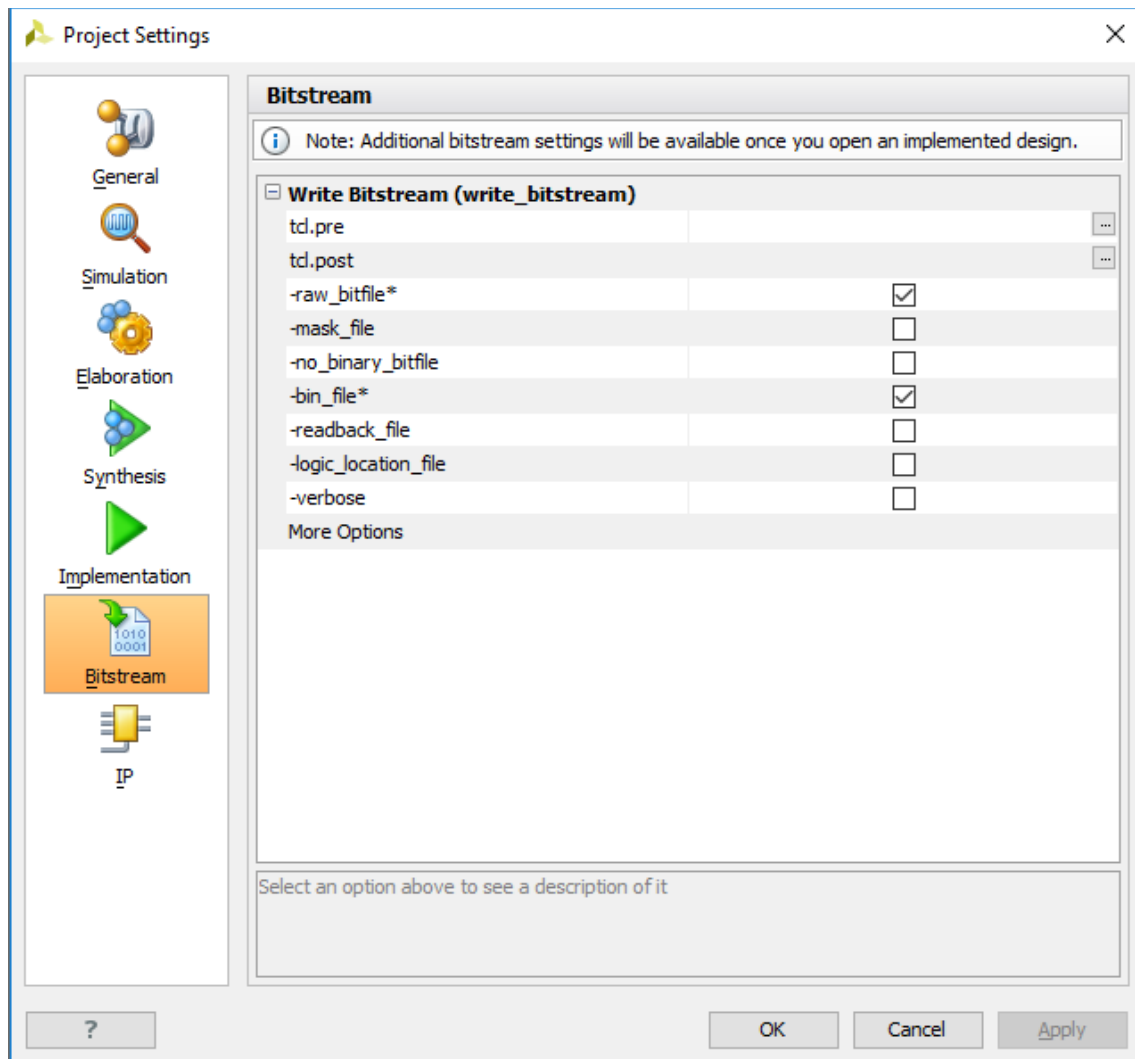
1. Configurant directament la FPGA a través del JTAG amb un arxiu .bit. En aquest cas, quan desconnectem l'alimentació de la tarja, la FPGA quedarà desconfigurada.
2. Gravant un arxiu .bin a la serial prom de la Basys 3. Així, cada vegada que alimentem la tarja, la FPGA es configurarà automàticament llegint aquesta memòria.

Abans de començar és recomanable fer alguns ajustos en la configuració del dispositiu. Dins del menú Tools, anem a “Edit Device Properties” per a modificar els següents paràmetres:

- **General:** Enable Bitstream Compression (TRUE).
- **Configuration:** Configuration Rate (Mhz) (33).
- **Configuration Modes:** Master SPI x4

OK i Ctrl+S.

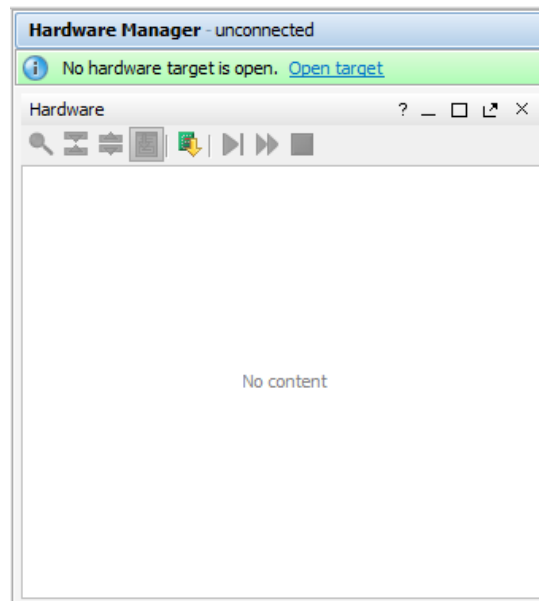
A l'apartat de “Program and Debug” obrirem el “Bitstream Settings” i seleccionarem les dues opcions següents:



El bitfile és l'arxiu que utilitzarem per configurar la FPGA, i el bin_file és el que utilitzarem per gravar la serial-prom de la basys 3 quan optem per la segona opció.

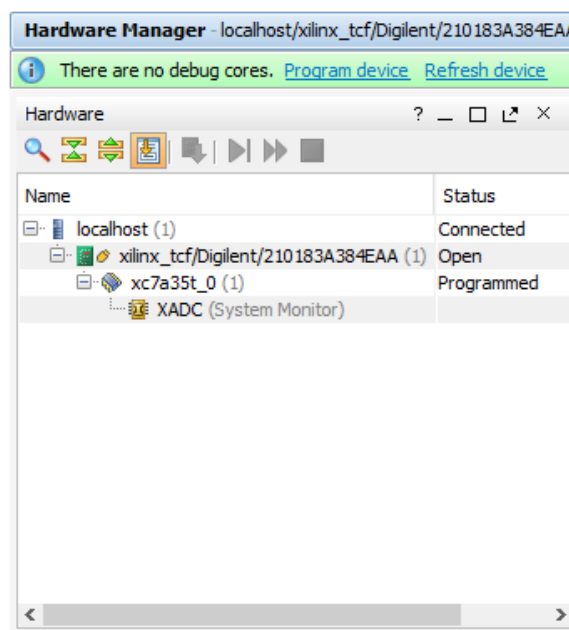
7.1. Configuració amb arxiu .bit (directe a la FPGA):

En el mateix apartat de “Program and Debug”, executarem l’acció “Generate Bitstream”. Quan acabi el procés s’obrirà la finestra per triar la següent acció a realitzar, triarem “Open Hardware Manager”.

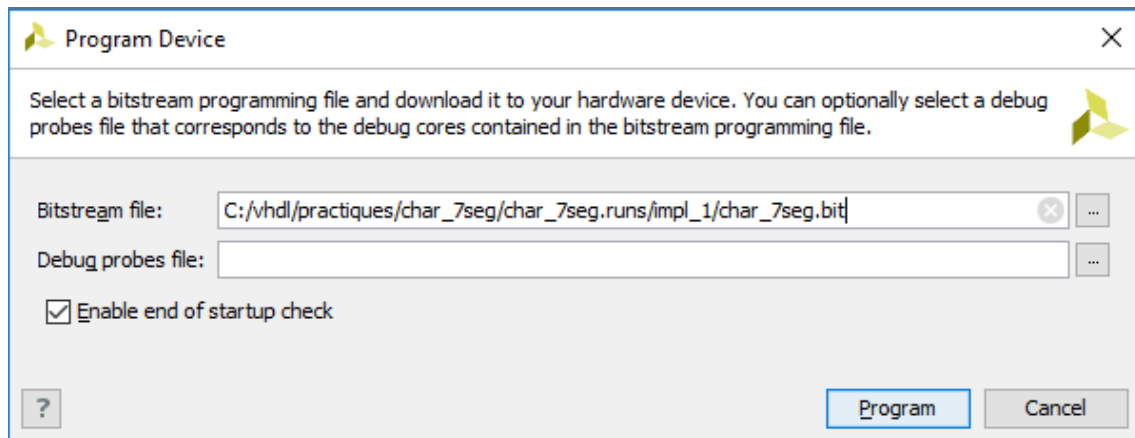


Abans de continuar, assegurem-nos que tenim la Basys 3 connectada a un port USB del PC, amb l'interruptor d'alimentació a ON i el jumper JP1 a la posició central (JTAG).

A la barra de color verd, farem click sobre “Open target” i seguidament a “Auto-Connect”.



Ara ens apareixerà el dispositiu connectat. Només cal fer click a “Program Device” (a la barra verda, també), i sobre el nom del dispositiu que ens apareixerà tot seguit.



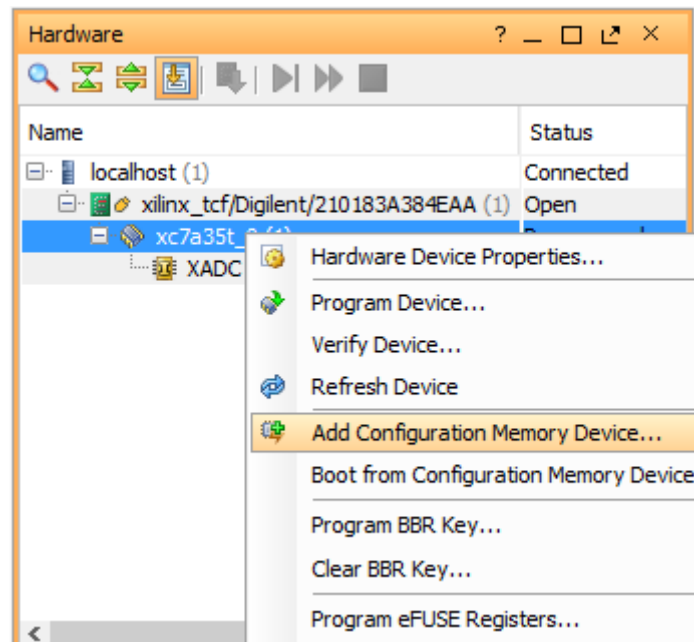
En aquesta pantalla podem seleccionar l'arxiu .bit que volem gravar a la FPGA, per defecte ens surt el .bit que hem generat en la operació anterior. Click a “Program” i ja ho tenim. La FPGA ja està configurada i ja podem comprovar el seu funcionament.

7.2. Configuració amb arxiu .bin (gravació de la EEPROM sèrie):

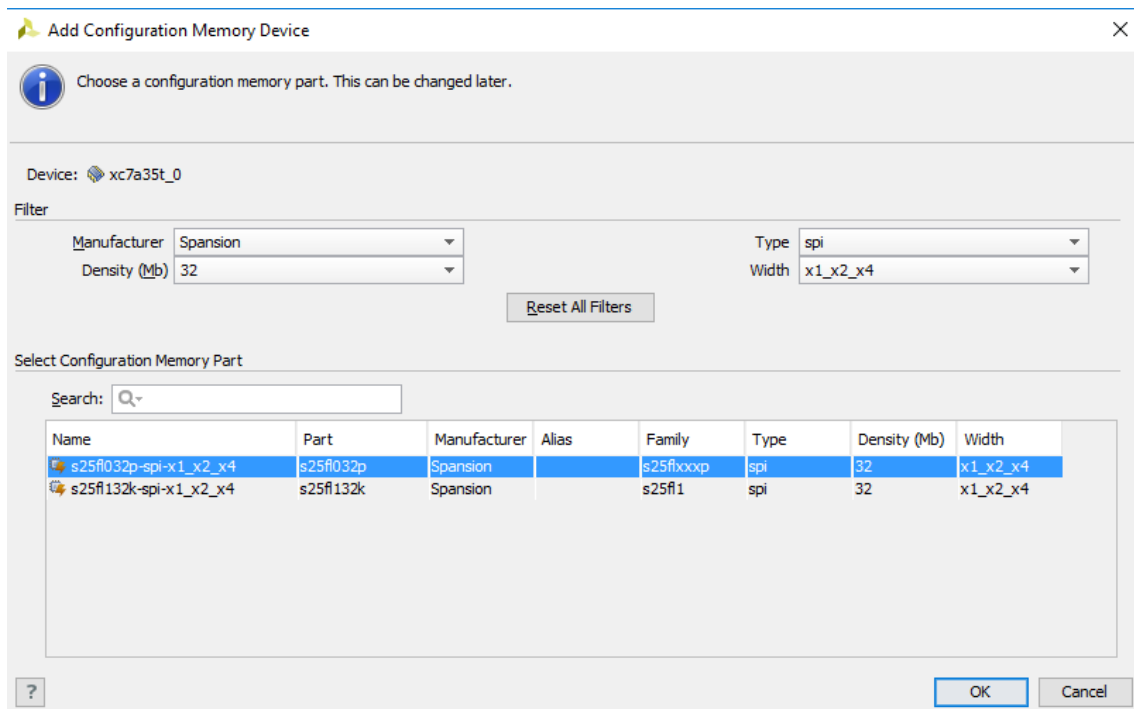
Aquesta opció té l'avantatge que si hem d'endollar i desendollar la tarja, no l'haurem de configurar cada vegada.

Primer de tot, parem la tarja amb el switch d'alimentació, posem el jumper JP1 en la posició superior (QSPI) i tornem a alimentar.

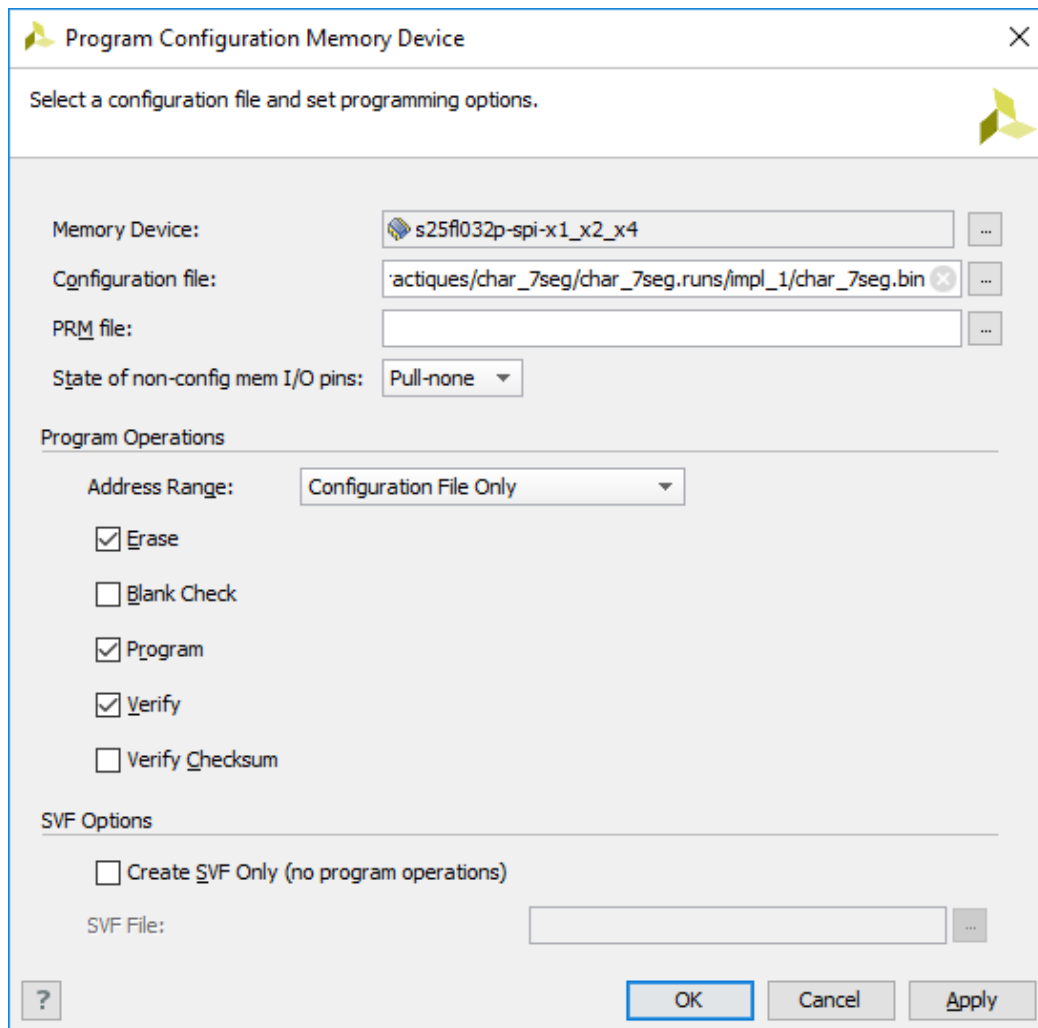
A la finestra de “Hardware”, fem botó dret sobre el nom del dispositiu, i triem “Add Configuration Memory Device”.



I seleccionar el model de memòria S25f032p a la llista. Podeu utilitzar els filtres.



Un cop seleccionada la referència, OK. A la següent finestra també OK, i us apareixerà la següent finestra:



On seleccionarem l'arxiu .bin que hem generat anteriorment. La resta d'opcions les podem deixar tal i com estan. Fem OK i ens gravarà la EEPROM.

Un cop gravada, podem comprovar que cada cop que alimentem la tarja, la FPGA es configurarà automàticament.