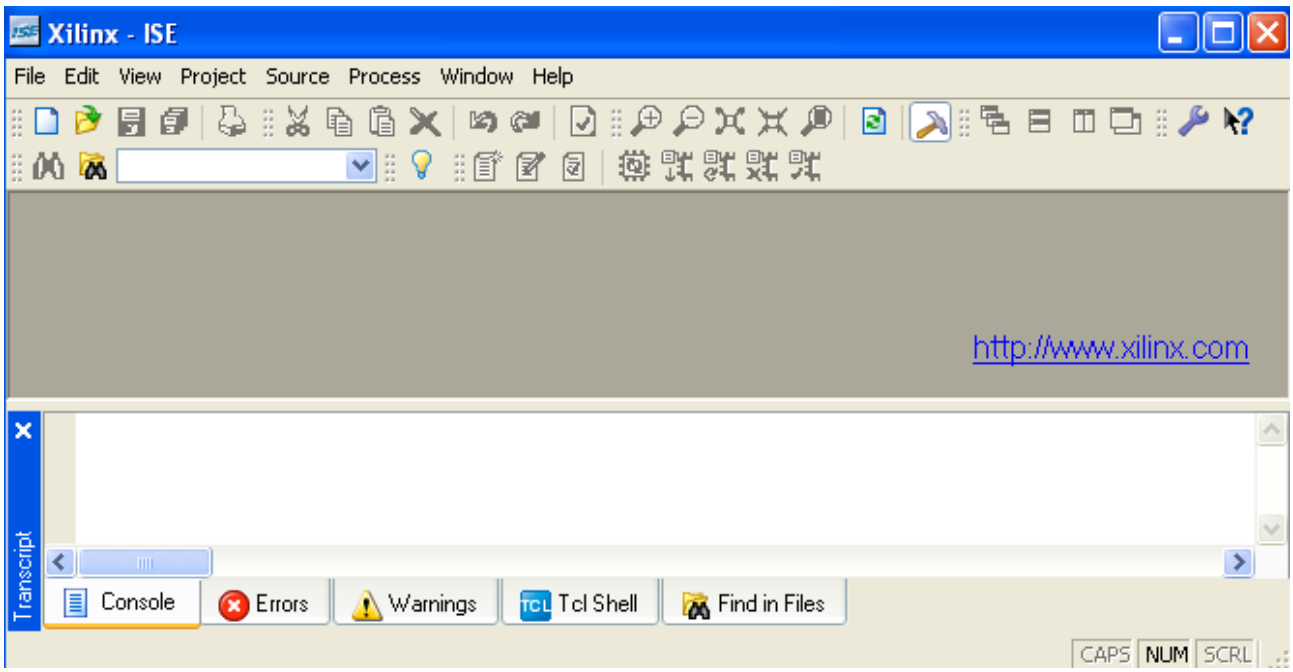
Segédlet a Xilinx Spartan-3 FPGA méréshez

A Xilinx ISE WebPack 9.1 IDE (Integrated Development Envirement – Integrált Fejlesztő Környezet) segítségével hozzunk létre egy egyszerű demo programot a gyakorlópanelen található ledek, illetve a kijelző tesztelésére!

Indítsuk el a Xilinx ISE WebPack 9.1 programot:Start menüből, vagy közvetlen link  segítségével.

A program elindítása után a következő felülettel találkozunk:



1.1. ábra

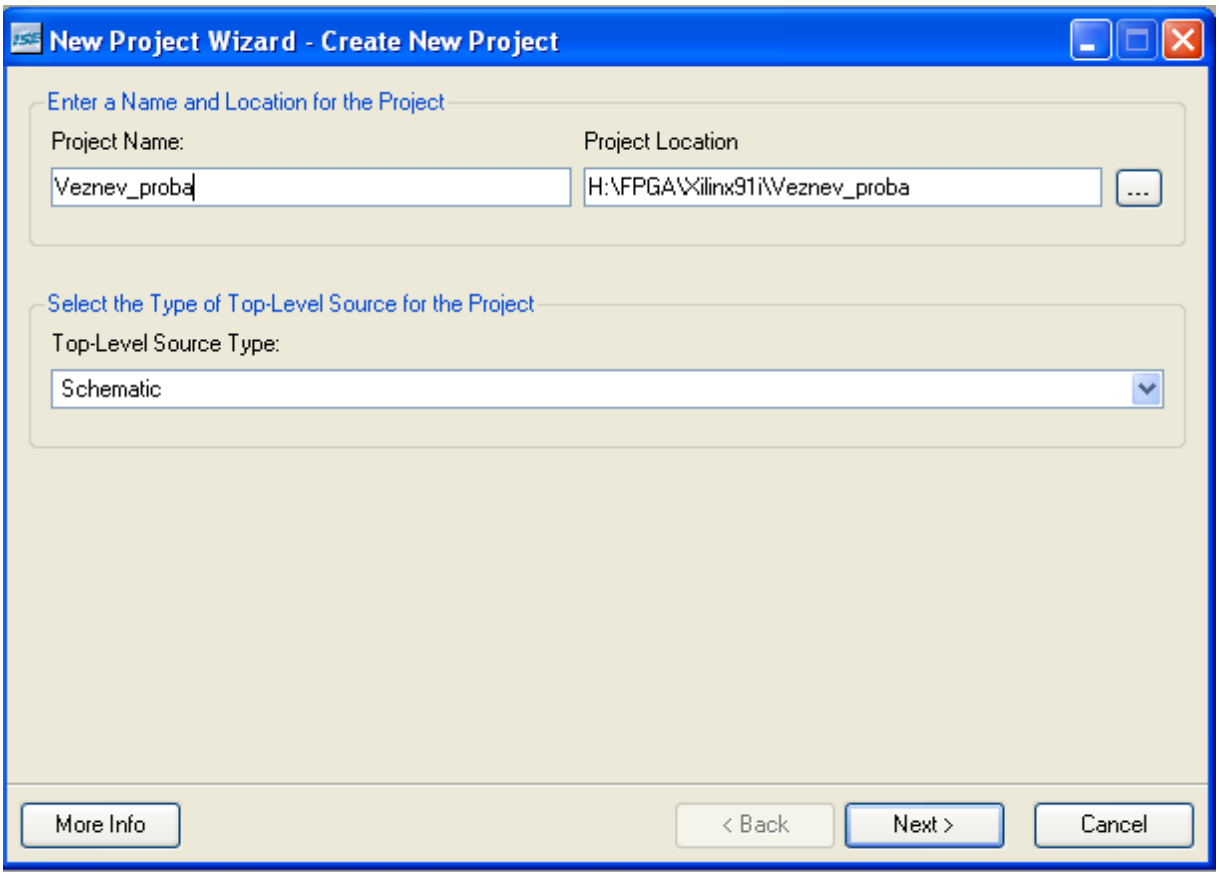
Xilinx Ise Webpack kezelőfelülete

Első lépésként készítsünk el egy új projekt file-t (**File --> New Project**):

Amennyiben rendelkezünk már projekt fájllal nyissuk meg az **Open project** menüponttal!

A fejlesztőkörnyezet minden projektnek külön könyvtárat hoz létre, és a xxx\Xilinx91i mappába menti.

Projektünk neve legyen, javaslat: a vezetéknevünk_proba file, legmagasabb szintű forrásként **schematic** (kapcsolási rajz alapú) típust adjunk meg!



1.2.1. ábra

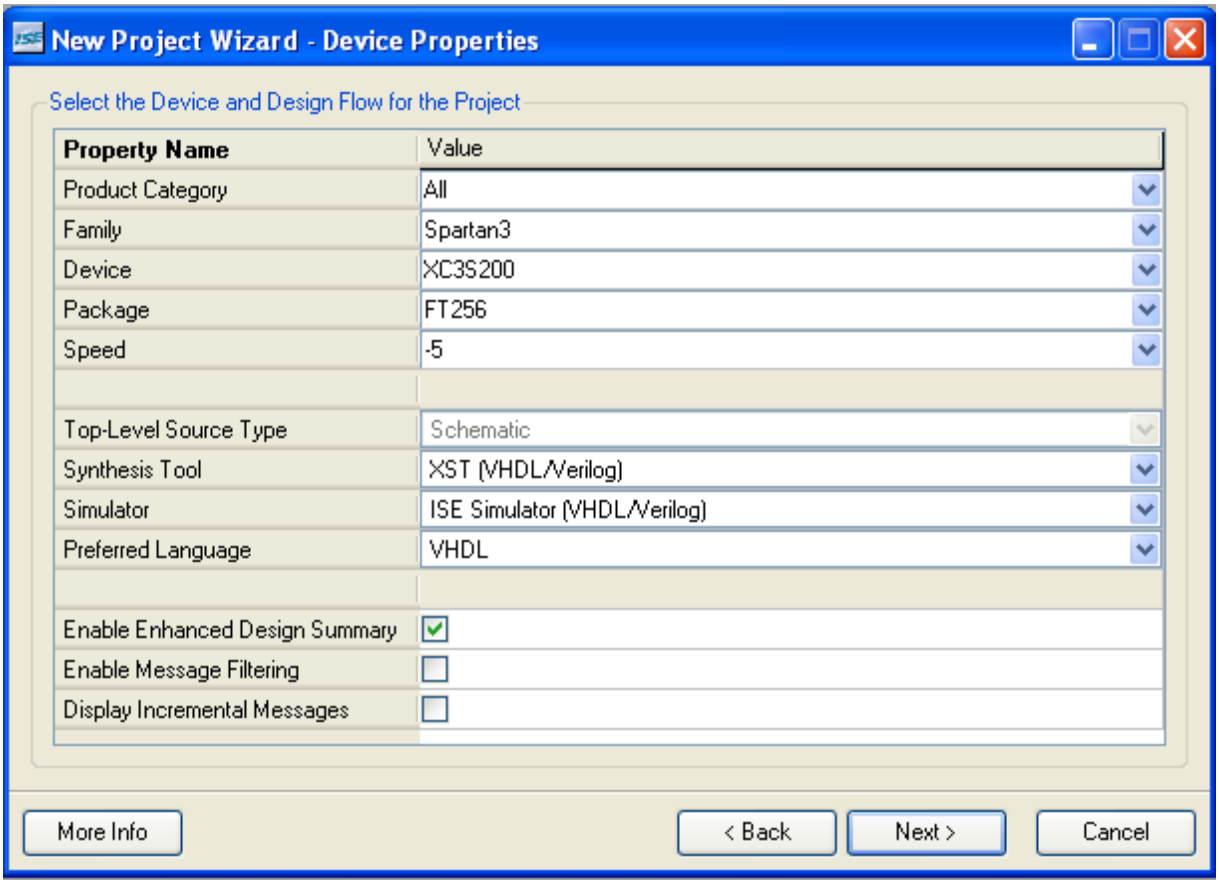
Project elsődleges forrástípusának kiválasztása

Az általunk elkészíteni kívánt program modulokból épül fel, minden modulnak 3 különféle forrása lehet:

- Schematic: kapcsolási rajz
- VHDL: hardverleíró nyelv
- StateCad: állapotdiagram

A legfelső ún. TOP forrásban fogjuk összeállítani modulunkat – ezt a legegyszerűbben kapcsolási rajz (**schematic**) típus esetén tudjuk megtenni!

A következő ablakban a fejlesztéssel kapcsolatos eszköz fontosabb tulajdonságait állíthatjuk be:



1.2.2. ábra

FPGA tulajdonságainak beállítása

Termék kategória (**Product Category**) **All** (minden)

Eszközcsalád (**Family**): **Spartan-3**

Eszköztípus (**Device**): **XC3S200**

Tokozás (**Package**): **FT256**

Sebesség (**Speed**): **-5**

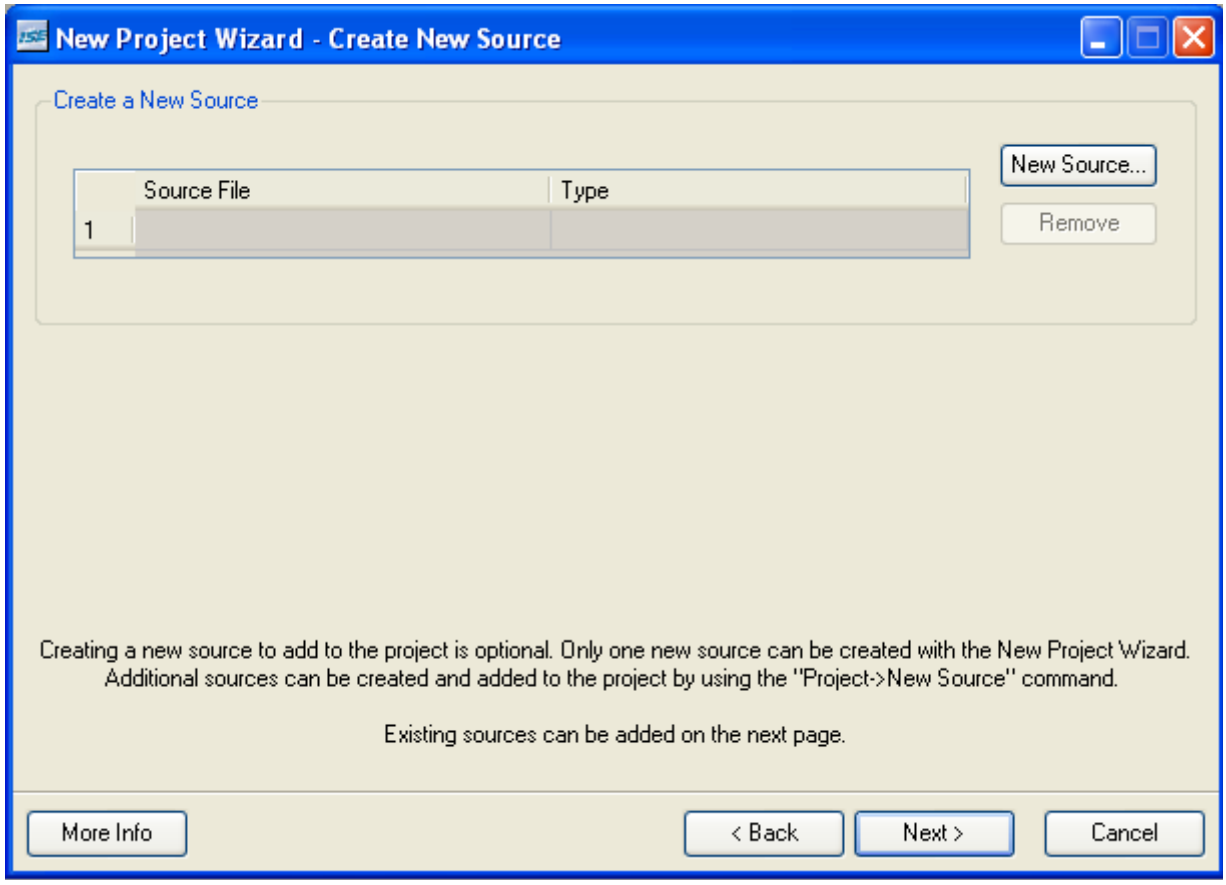
Elsődleges forrás (**Top Level Source Type**): **Schematic** (nem módosítható)

Szintézis eszköze (**Synthesis Tool**): **XST (VHDL/Verilog)**

Szimulátor (**Simulator**): **ISE Simulator (VHDL/Verilog)**

Előnyben részesített nyelv (**Preferred Language**): **VHDL**

Forrást a Next gombra kattintva hozhatunk létre:



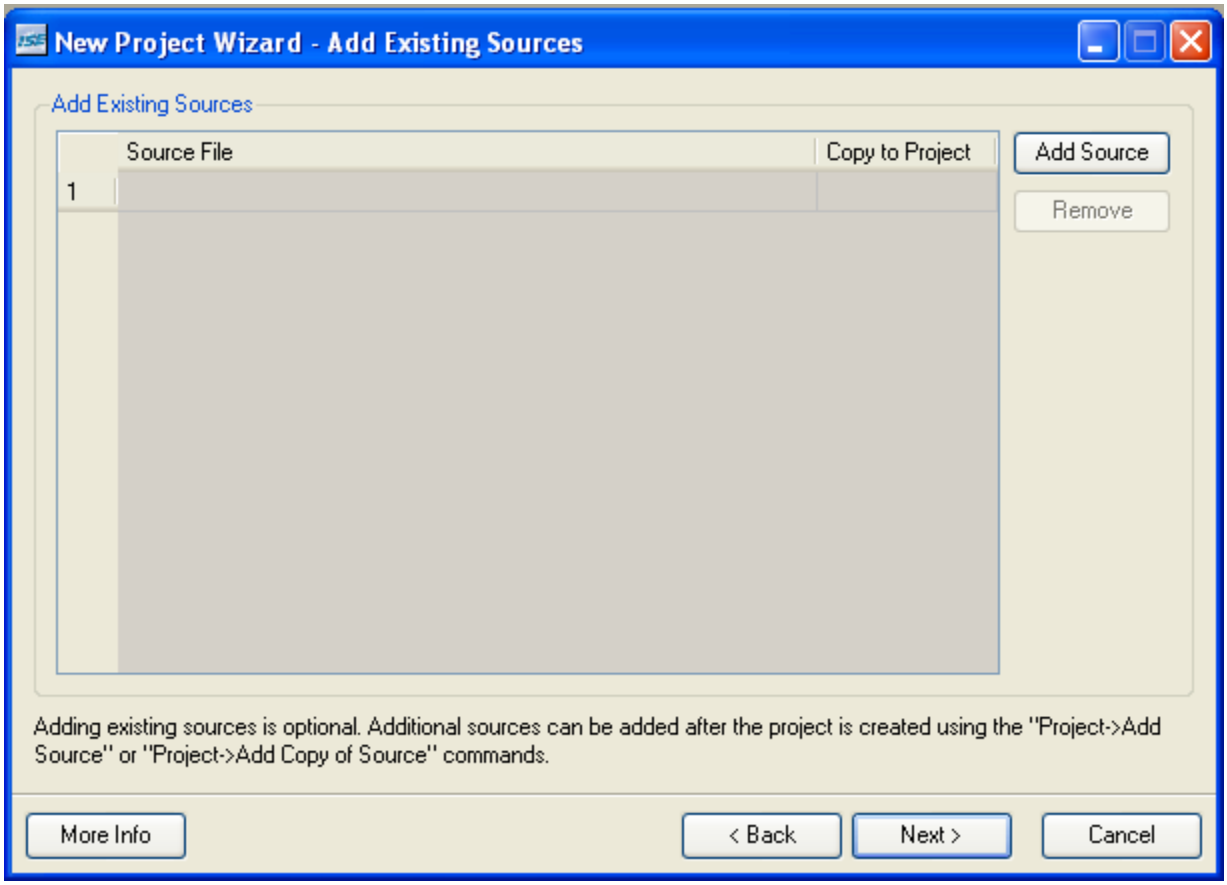
1.2.3. ábra

Új forrás létrehozása

Ezt megtehetjük most, vagy akár a későbbiek folyamán a **Project --> New Source** menüponttal!

Egyenlőre ne hozzunk létre forrást!

A következő ablakban forrást adhatunk hozzá a projekt fájlunkhoz, amennyiben az előbb létrehoztunk egyet természetesen azt is hozzáadhatjuk!

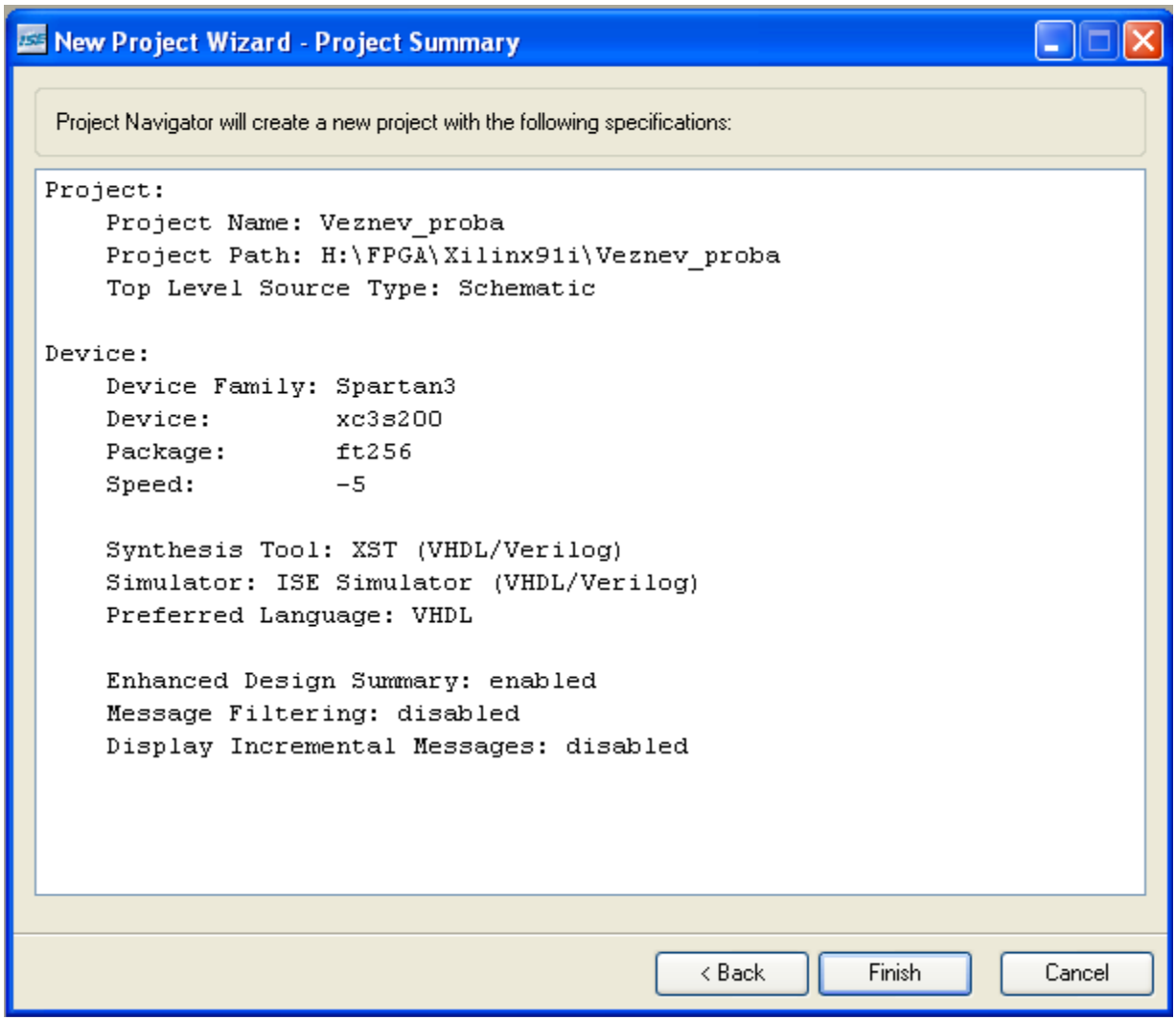


1.2.4. ábra

Forrás hozzáadása a projekthez

E lépés végrehajtására is van lehetőségünk a későbbiek folyamán a **Project -> Add Source** menüponttal!

Miután ismét a Next gombra kattintottunk a projektünk specifikációjának összefoglalója jelenik meg:



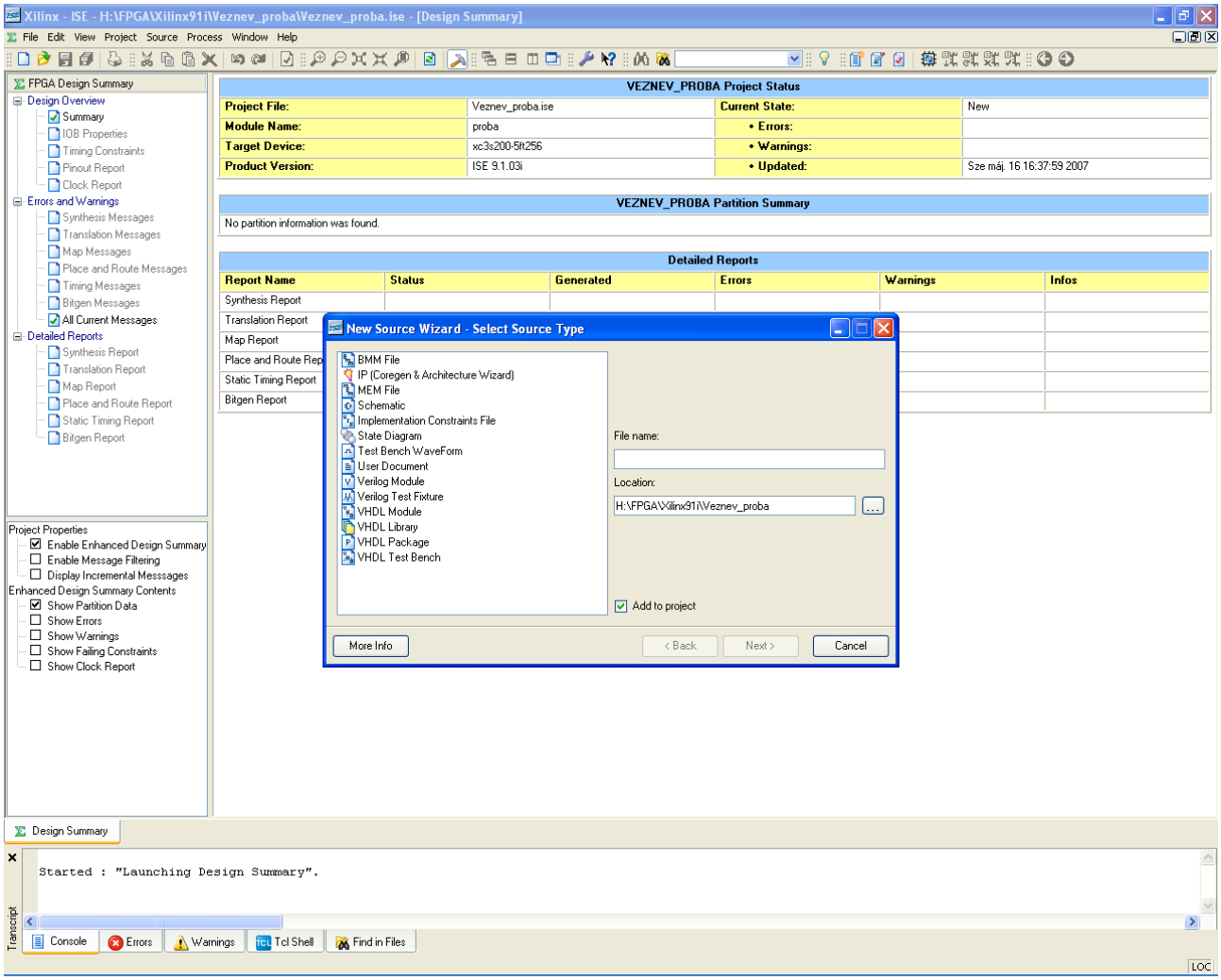
1.2.5. ábra

Projekt tulajdonságainak összegzése

Ellenőrizzük, hogy az elsődleges forrás schematic, valamint az eszköz tulajdonságait!

Amennyiben mindent rendben találunk, kattintsunk a Finish gombra!

Hozzunk létre a projectünkhöz forrás fájlt (**Project -> New source...**):



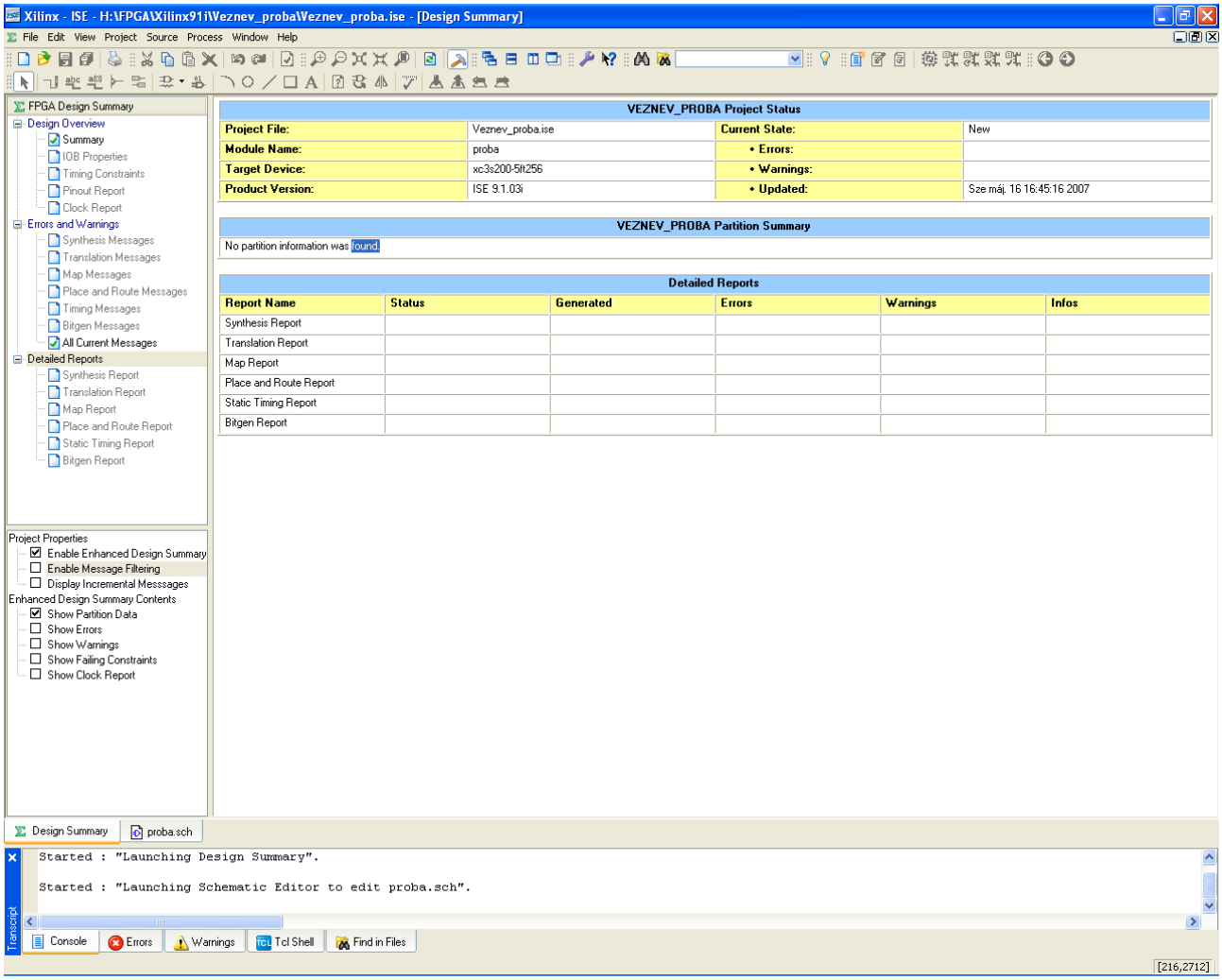
1.3 ábra

Új forrás létrehozása

Legyen a forrásunk típusa schematic, a neve tetszőleges!

Adjuk hozzá a forrást a projecthez (**Project -> Add source...**):

Siker esetén a következő felülettel találkozunk:

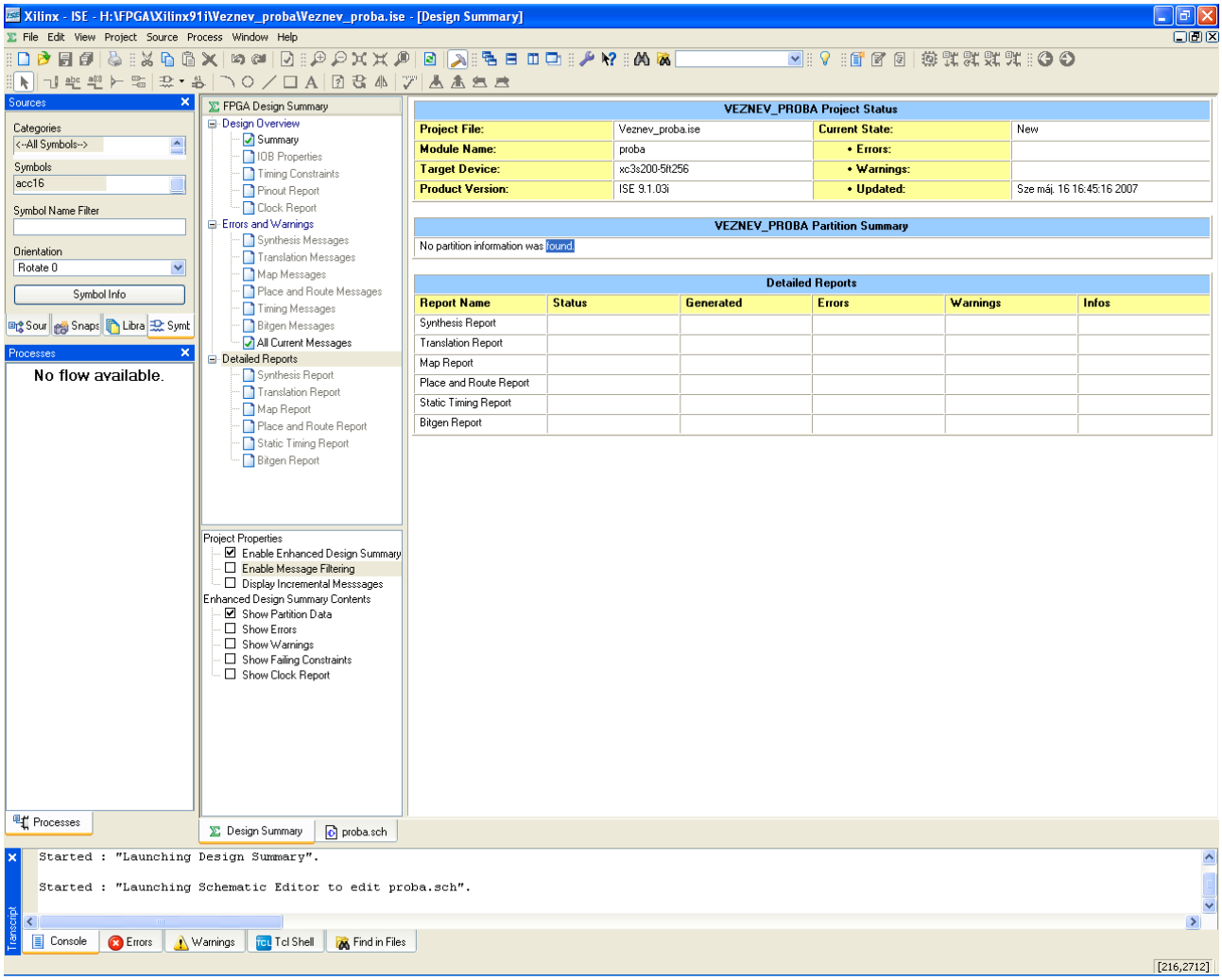


1.4. ábra

Projekt forrás hozzáadása után

A **View** menüpontban állítsuk aktívra a forrás (**sources)**, és a folyamatok (**processes)** menüpontokat!

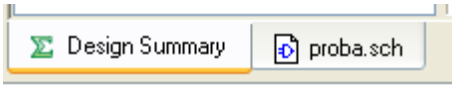
Ennek hatására felületváltás történik (megjelenik bal oldalon a források és a folyamatok ablak):



1.5. ábra

Sources és Processes ablak láthatóvá tétele

Kattintsunk a proba.sch fülre:



1.5.1. ábra

Váltás a forrásra

Kezdjünk hozzá a programozáshoz!

Első programunk a következő funkciókat fogja megvalósítani:

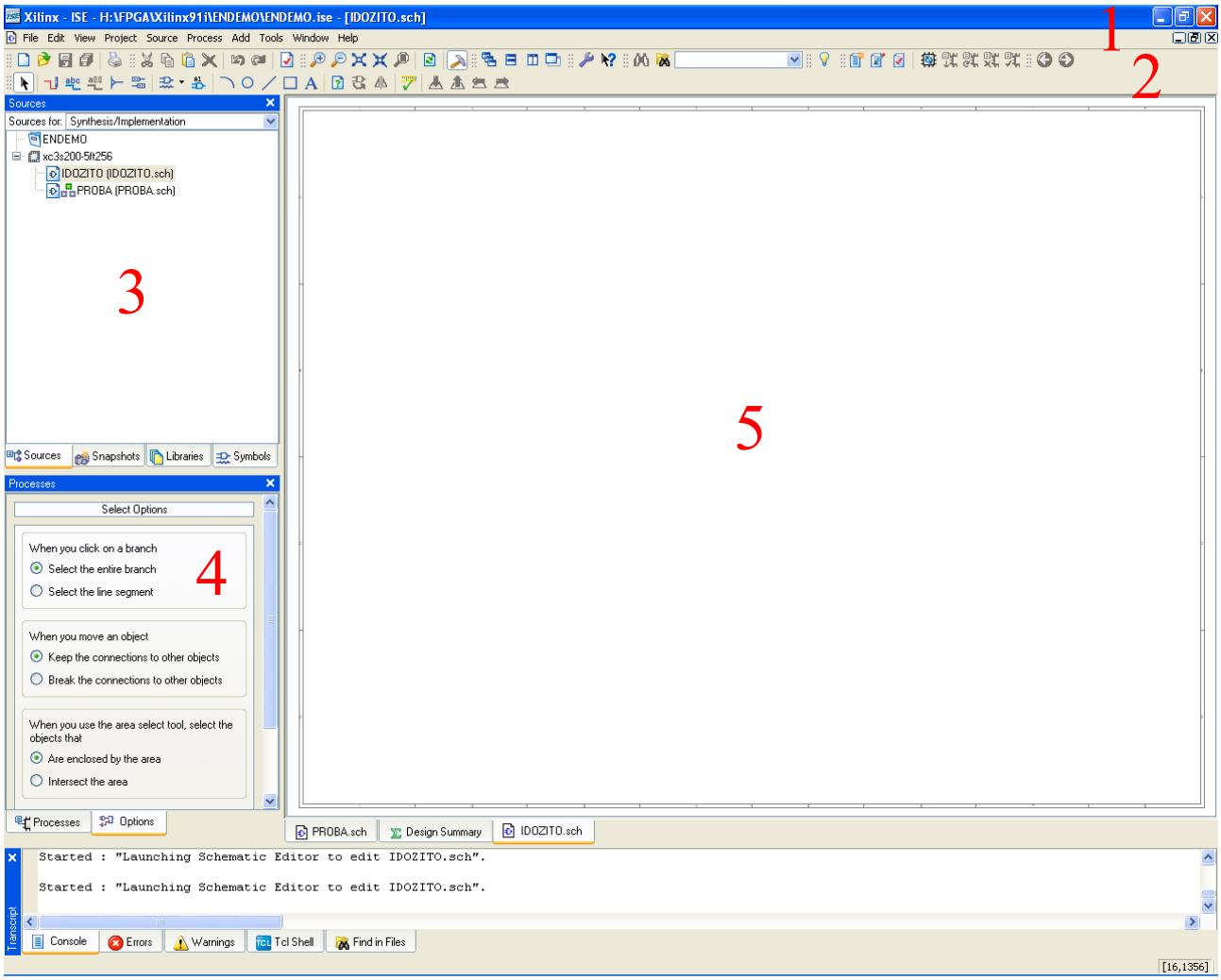
- egy kapcsoló hatására bekapcsolja egymás után a gyakorlópanelen lévő LED-eket
- valamint az egyes szegmenseket a kijelzőkön
- (amennyiben a kapcsolót lekapcsoljuk működés közben, őrzi az előző állapotok

Ehhez szükségünk van egy időzítőáramkörre, amely az egyes LED-ek ill. szegmensek bekapcsolásának időközét állítja be, a láthatóság érdekében (amennyiben szimulációt kívánunk csak végezni eltekinthetünk az órajel-osztó áramkör megépítésétől)!

Az időzítőáramkört kapcsolási rajz alapú szerkesztővel definiáljuk:

1. Hozzunk létre egy új forrást **Project -> New Source** (schematic)!
2. Adjuk hozzá ezt a forrást a projektünkhöz **Projekt -> Add Source**!

A kényelmes fejlesztéshez a következő ablakelrendezés ajánlott:



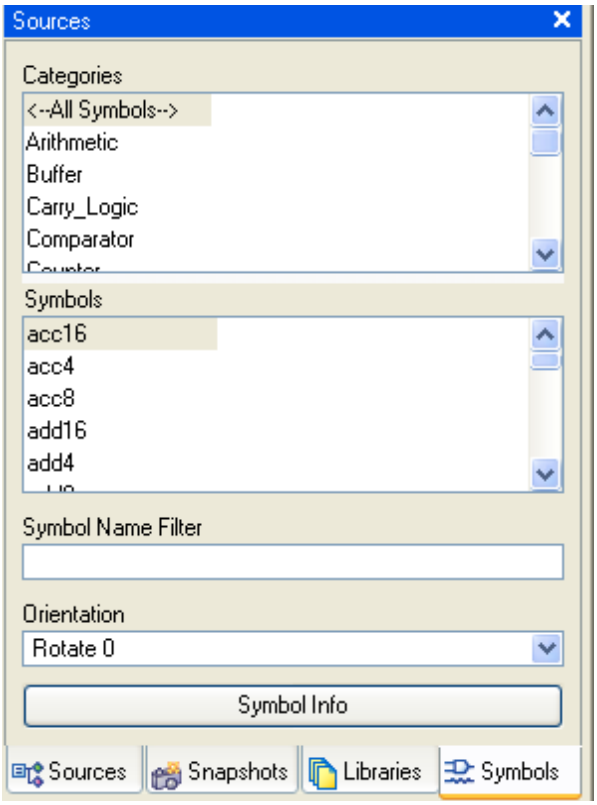
1.6. ábra

Időzítő forrás hozzáadás a projekthez

1: Menüsor
2: Fontosabb ikonok
2: Források (**Source**) ablak
4: Folyamatok (**Process**) ablak
5: Munkatér

Ügyeljünk rá, hogy az IDOZITO.sch fül legyen aktív!

Válasszuk ki a szükséges alkatrészeket a bal oldalon található Sources ablak Symbols fülének segítségével:



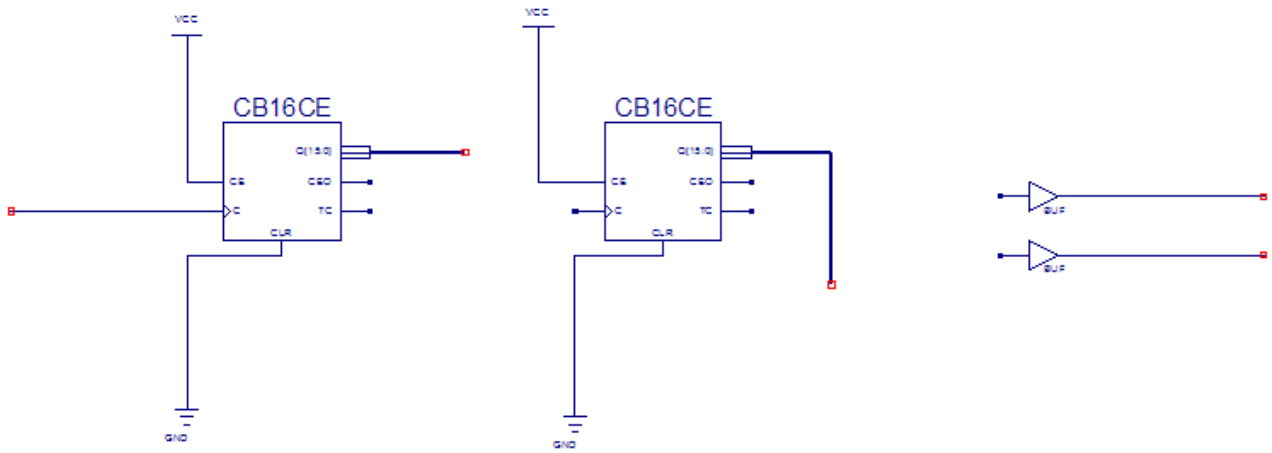
1.7. ábra

Alkatrészek kiválasztása

Az alkatrész megkeresése történhet kategóriák alapján, de használhatjuk a névkeresőt is (Symbol Name Filter).

Órajelünk 50MHz - két 16 bites osztóval megvalósíthatjuk mind a pergésmentesítéshez, mind az időzítéshez szükséges belső órajelet.

Hozzuk létre a következő hálózatot:



1.8.1 ábra

Alkatrészek elhelyezése a szerkesztő felületen

Alkatrészek:

- CB16CE: Engedélyezhető 16 bites számláló (2db) /itt: frekvenciaosztó/
- BUF: Neminvertáló puffer
- VCC: Pozítiv tápfeszültség (+5V; magas szint (H); 1)
- GND: Földpont (0V; alacsony szint; 0)

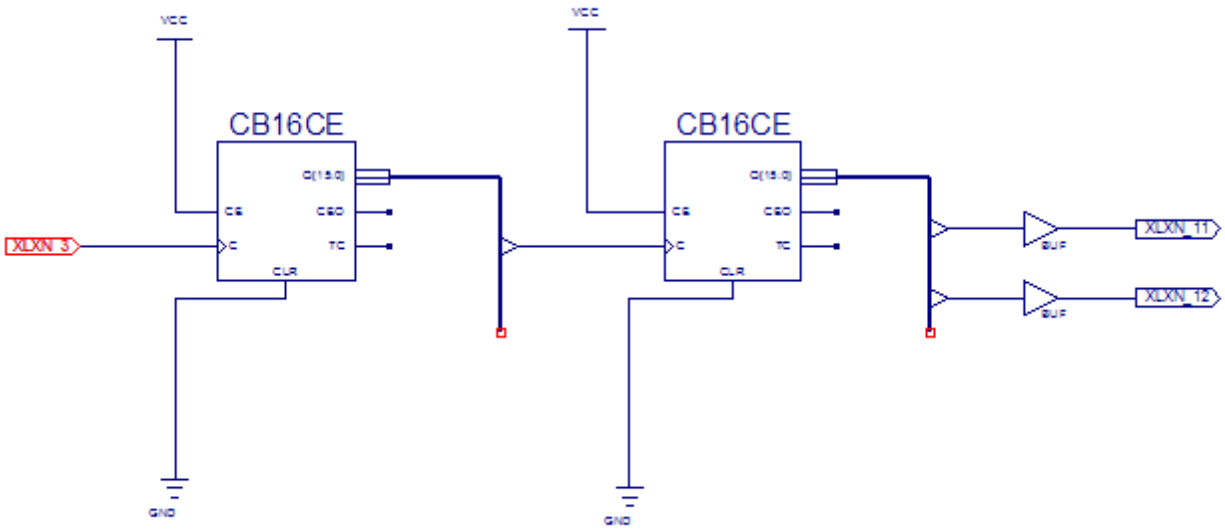
A hozzávezetéseket, illetve az alkatrész összeköttetéseket az Add Wire ikon segítségével készíthetjük el:

Select ( ):	Kiválasztás/mozgatás.
Add Wire ( ):	Vezeték hozzáadása.
Add Net Name ( ):	Vezetékezés elnevezése.
Rename Selected Bus ( ):	A kijelölt busz átnevezése.
Add Bus Tap ( ):	Buszos vonalra csatlakozás hozzáadása.
Add I/O Marker ( ):	Be-, kimeneti pontok hozzáadása.
Add Symbol ( ):	Szimbólum (alkatrész, könyvtári, vagy saját) hozzáadása.

Bármely művelethez a bal alsó ablakban (**Process/Options**) találunk opciókat, illetve műveleteket.

Jól látható, hogy a számláló kimenetéhez csatlakoztatott vezeték vastagabb a többinél, ez jelzi annak buszos voltát. Tegyünk az első számláló kimenetére egy, ill. a második számláló kimenetére két buszos csatlakozót (**Add Bus Tap**):

Végezzük el a hálózat teljes bekötését (be- kimenet hozzáadása: Add I/O Marker):

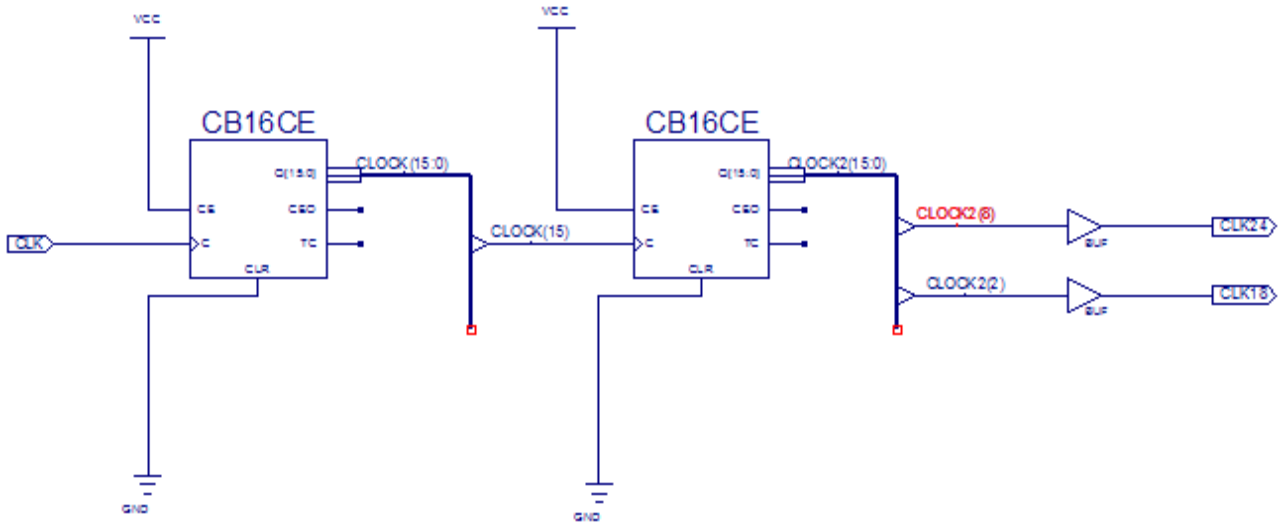


1.8.2. ábra

I/O markerek elhelyezése

Nevezzük el a különböző buszokat, ill. I/O markereket:

- Vezeték, busz elnevezése: Add Net Name (a bal alsó ablakban az Options fülnél a Name sorba írjuk a nevet, majd kattintsunk az elnevezni kívánt vezetékre, ügyeljünk, hogy ez a vezeték elég hosszú legyen a felirat elhelyezéséhez!).
- I/O marker elnevezése: dupla kattintás a markeren!



1.8.3. ábra

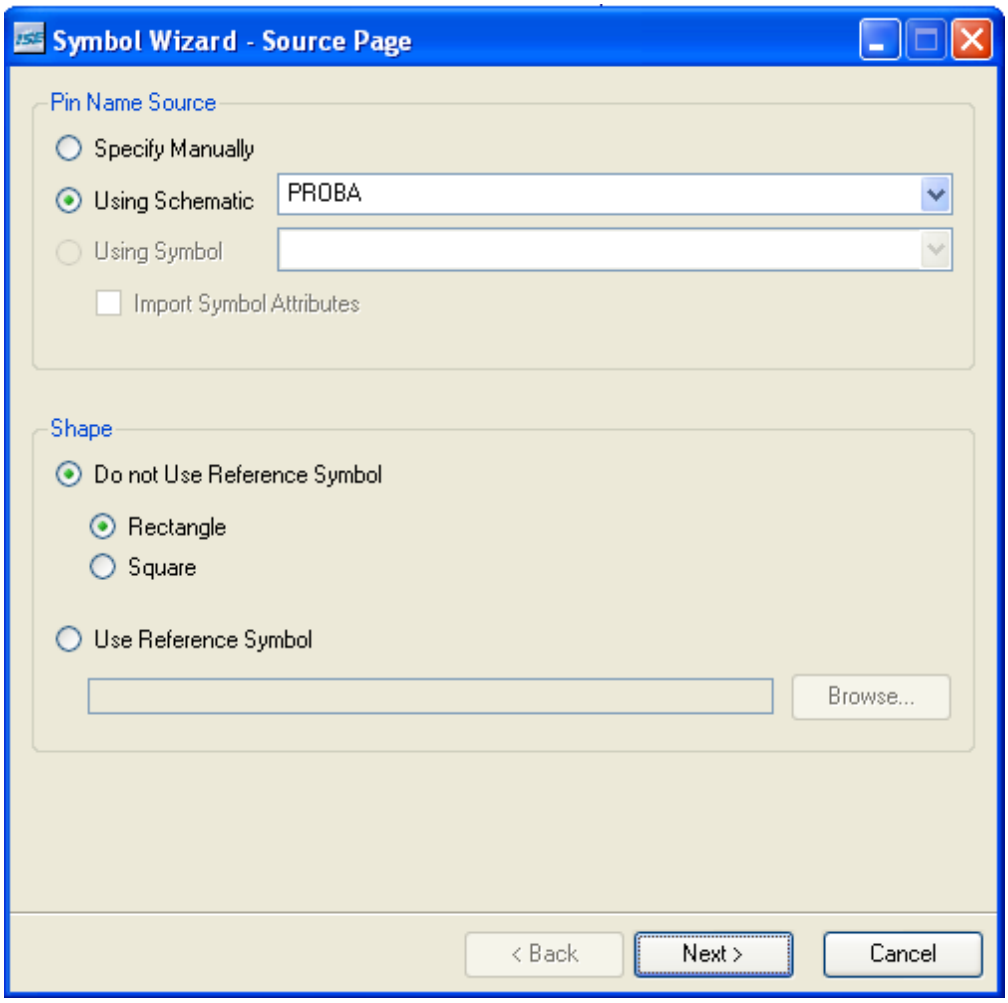
I/O markerek, vezetékek, buszok elnevezése

A buszok, ill. leágazásaik elnevezése utal arra, hogy melyik vonalról (bitről) van szó, míg a markerek elnevezése tetszőleges (CLK: clock-órajel; CLK24: az órajel frekvenciájának 24-ed része; CLK18: az órajel frekvenciának 18-ed része).

A CLOCK18 jelét a későbbiekben használhatja a kapcsolók, vagy nyomógombok prellmentesítésére (prellezés: a mechanikus kapcsolók nyomógombok nem adnak rögtön stabil jelet, két állapot között „peregnek”, s bár ez emberi mértékkel gyorsnak tekinthetjük (max n*ms), figyelembe véve a digitális rendszerek órajelét (n*MHz) ez komoly gondot okozhat)!

A fejleszőkörnyezet lehetőséget nyújt saját alkatrész definiálására az átláthatóság megkönnyítése érdekében – használjuk ki ezt a lehetőséget!

– A Tools menüben válasszuk ki a Symbol Wizardot!

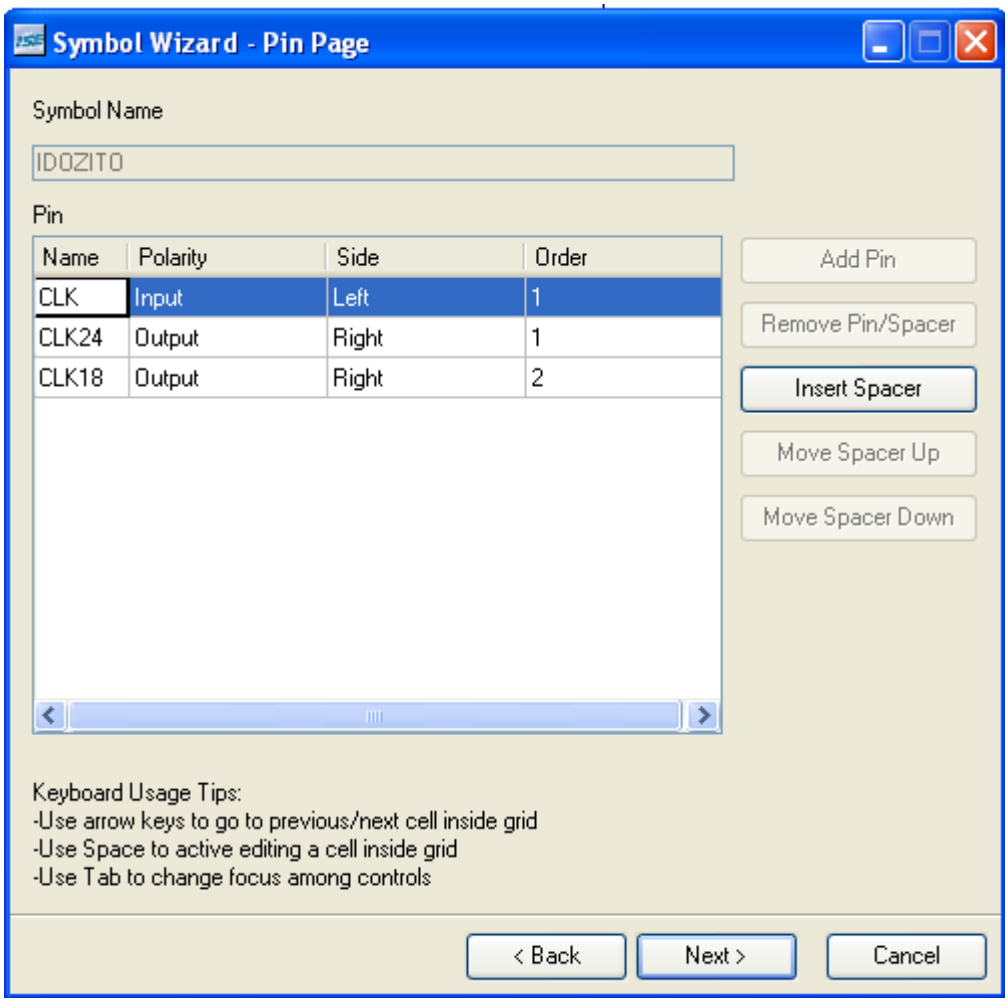


1.9.1. ábra

Saját alkatrész definiálása

Fontos, hogy a Using Schematic, illetve a Rectangle legyen bejelölve (névként bármit választhatunk)!

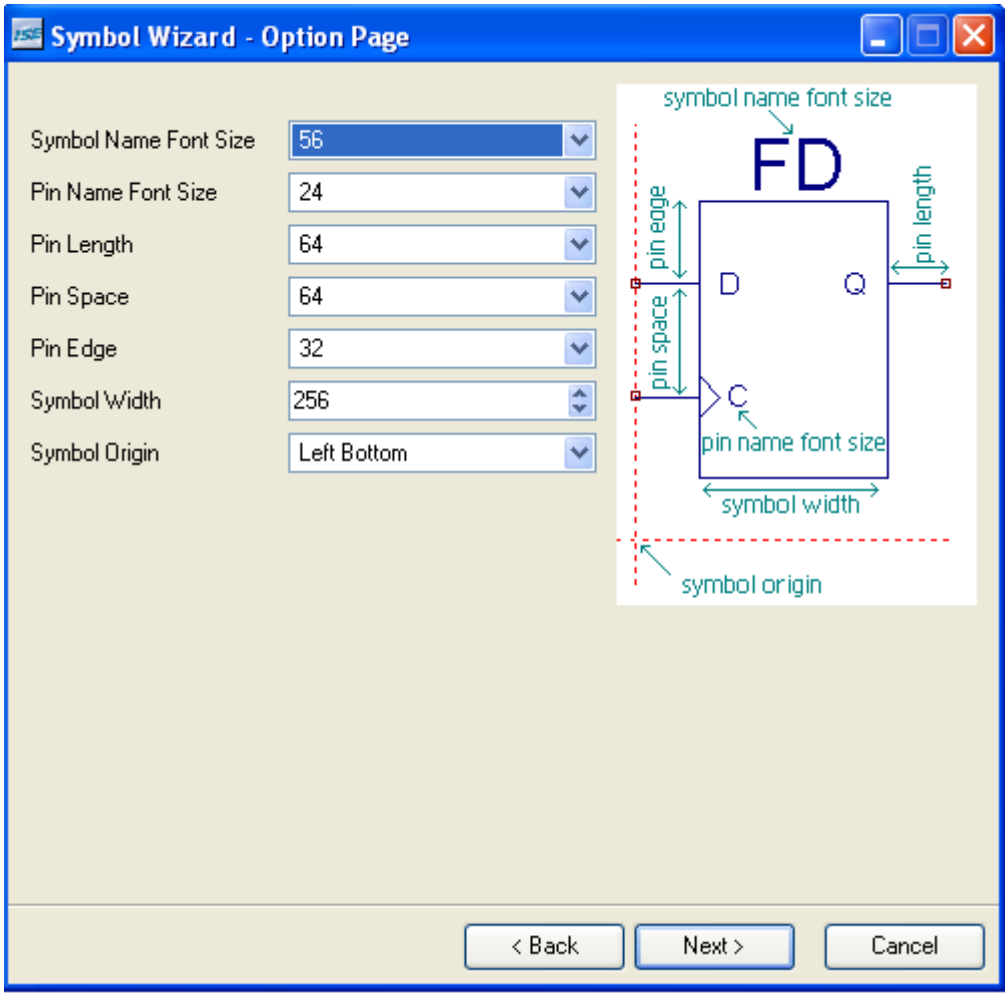
A következő ablakban az alkatrész be és kimenetei lesznek megjelenítve, ezen ne változtassunk:



1.9.2. ábra

Saját alkatrész I/O kivezetései

A Nextre kattintva alkatrészünk geometriai tulajdonságait módosíthatjuk (méretezhetjük):



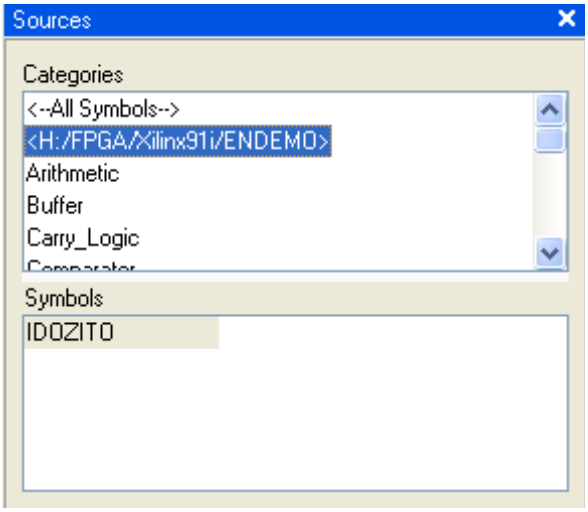
1.9.3. ábra

Geometriai tulajdonságok

Az utolsó ablakban megtekinthetjük alkatrészünk előnézetét.

Váltsunk a proba.sch fülre!

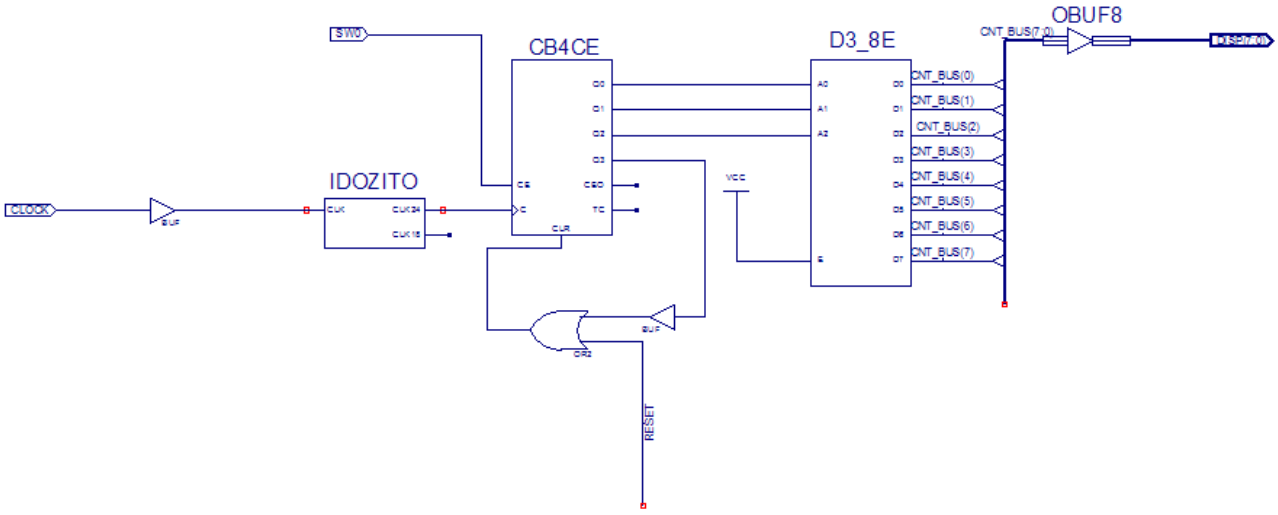
Amennyiben mindent jól csináltunk alkatrészünknek látszania kell a szimbólumok között:



1.9.4. ábra

Saját alkatrész kiválasztása

Készítsük el a következő áramkört:



1.10. ábra

Áramkörkészítés

Elemek:

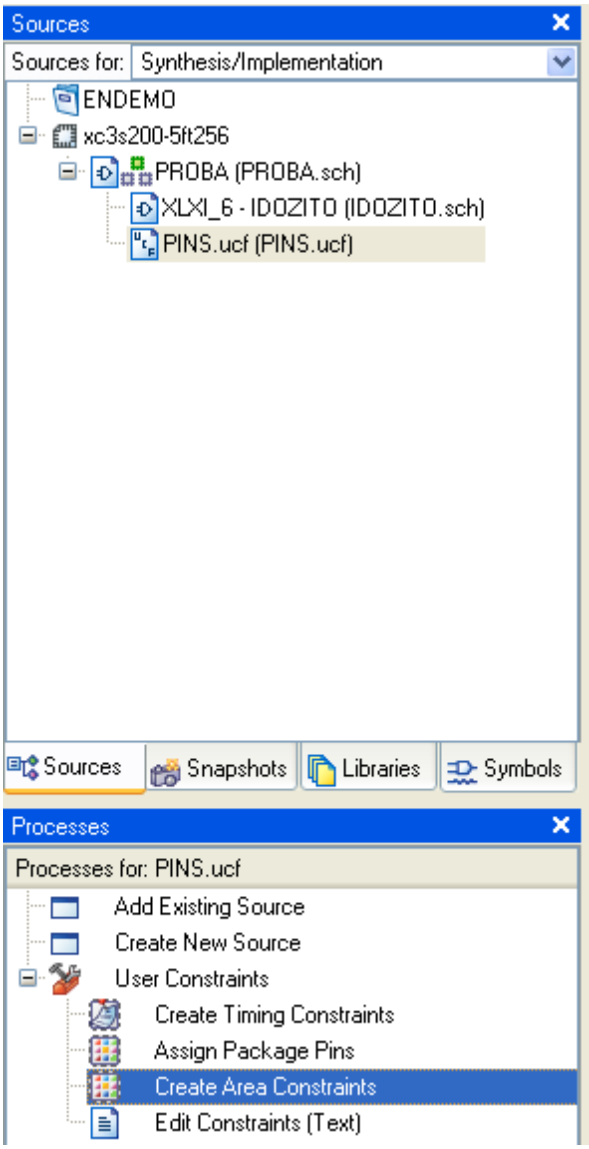
- I/O markerek:
    - CLOCK (órajel)
    - DISP(7:0) (7szegmens)
    - SW0 (kapcsoló)
- IDOZITO (saját alkatrész)
- BUF; OBUF8 (bufferek)
- CB4CE (számláló)
- D3_8E (dekóder)
- OR2 (kétbemenetű VAGY kapu)
- VCC (pozitív tápfeszültség; High szint; 1)

Adjunk a projektünkhöz egy Implementation Constraints Filet (Project->New Source)! Ebben a fileban fogjuk a lábkiosztást definiálni!

Kétféle módon tudjuk a lábakat az egyes I/O markerekhez rendelni karakteres, ill. grafikus módon.

Lábkiosztás grafikusan:

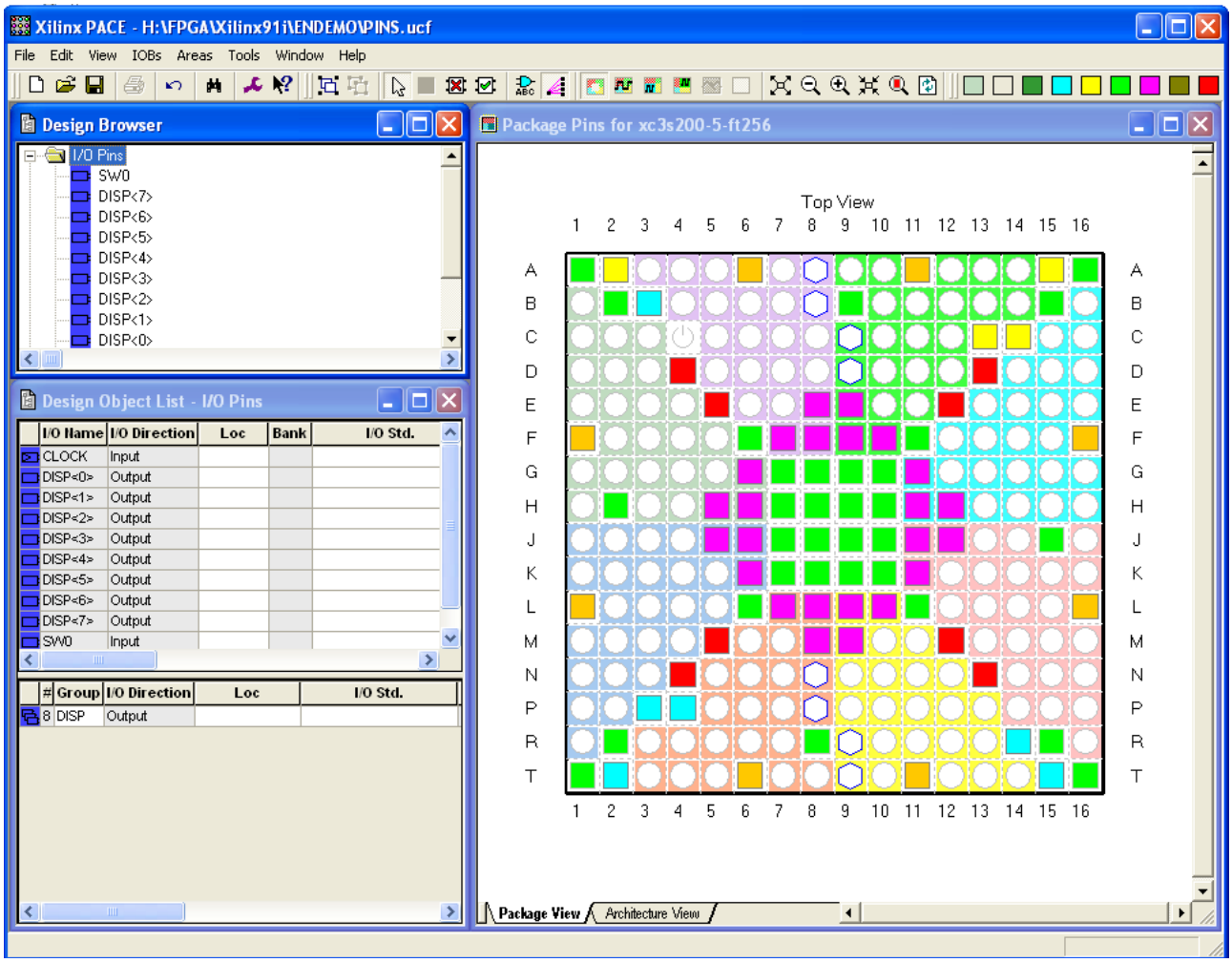
- Tegyük aktívvá az Imlementation Constraits Fileunkat!
- Kattintsunk duplán a Creat Area Constraints pontra!



1.11.1 ábra

Lábkiosztás

A grafikus felület megjelenése előtt a fordító elvégzi a felépített áramkör szintaktikai ellenőrzését, ezek után a következő ablak nyílik meg:

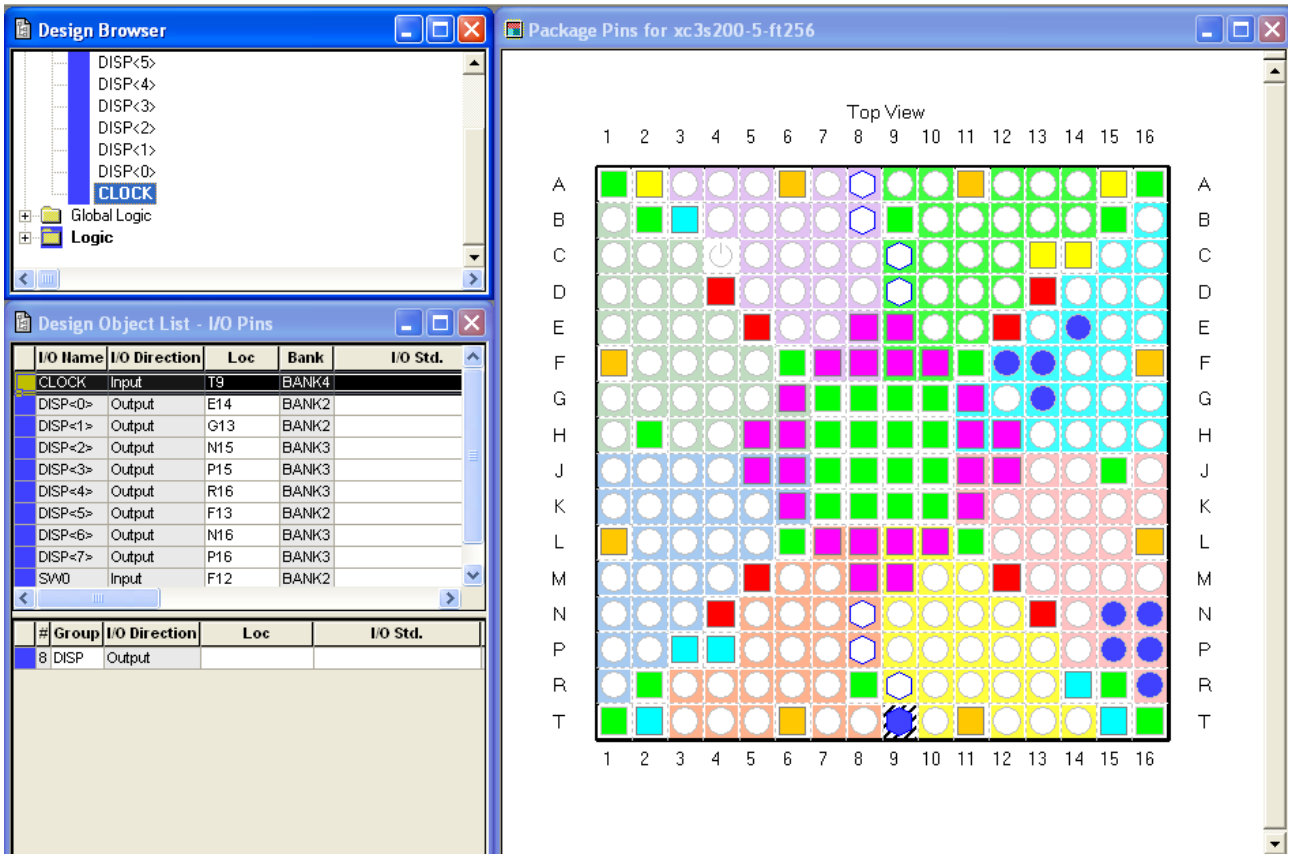


1.11.2. ábra

Lábkiosztás grafikus felületen

Az különböző jelölések értelmezéséhez bekapcsolhatjuk a **View -> Toolbars -> Legend** menüpontot. Ezt a funkciót a **CTRL+L** billentyű kombinációval is aktiválhatjuk.

A Design Browser ablakból „húzd és vidd” módszerrel vigyük át a jobb oldalon található ablak megfelelő helyére minden markert!

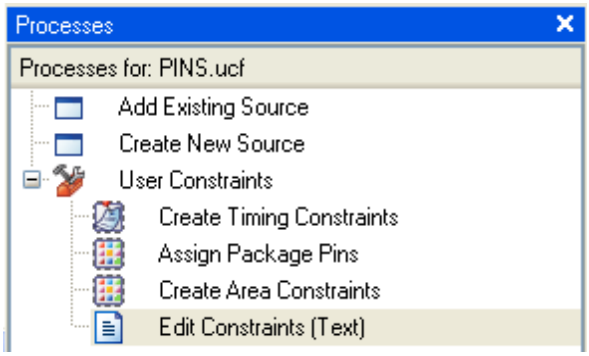


1.11.3. ábra

Lábak elhelyezése az IC felülnézeti képén

Miután az összes markert elhelyeztük, mentsük el a fájlt, majd zárjuk be!

A lábkiosztásra lehetőséget nyújt egy text editor is, melyet az **edit constraintsre** kettőt kattintva hívhatunk elő:



1.11.4. ábra

Lábkiosztás szöveges módszerrel

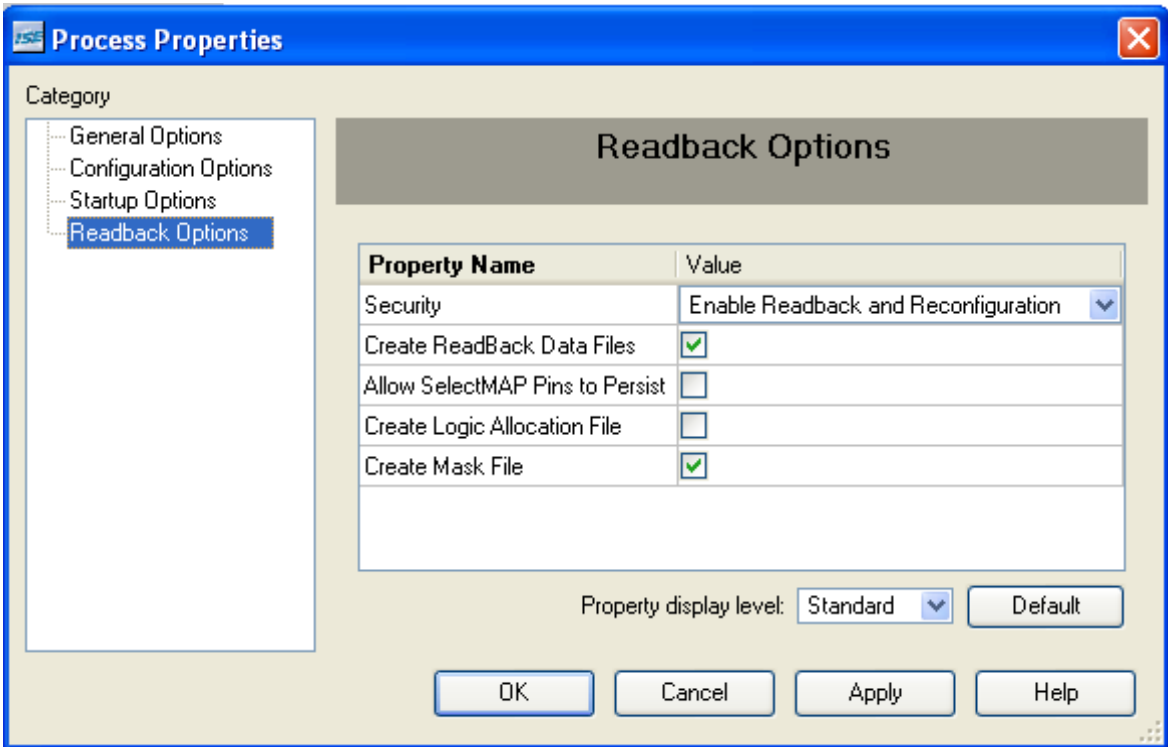
```
#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
NET "CLOCK"  LOC = "T9"  ;
NET "DISP<0>"  LOC = "E14"  ;
NET "DISP<1>"  LOC = "G13"  ;
NET "DISP<2>"  LOC = "N15"  ;
NET "DISP<3>"  LOC = "P15"  ;
NET "DISP<4>"  LOC = "R16"  ;
NET "DISP<5>"  LOC = "F13"  ;
NET "DISP<6>"  LOC = "N16"  ;
NET "DISP<7>"  LOC = "P16"  ;
NET "SW0"  LOC = "F12"  ;

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE
```

1.11.5. ábra

A lábkiosztás szöveges formája

A #-al kezdődő sorokat a fordító nem veszi figyelembe, így lehetőségünk van univerzális lábkiosztást írni, melyben csak az adott programban használt perifériákat használjuk, a többit kikommentezzük!

Kattintsunk jobb egérgombbal a Generate Programming Filera, majd Properties:

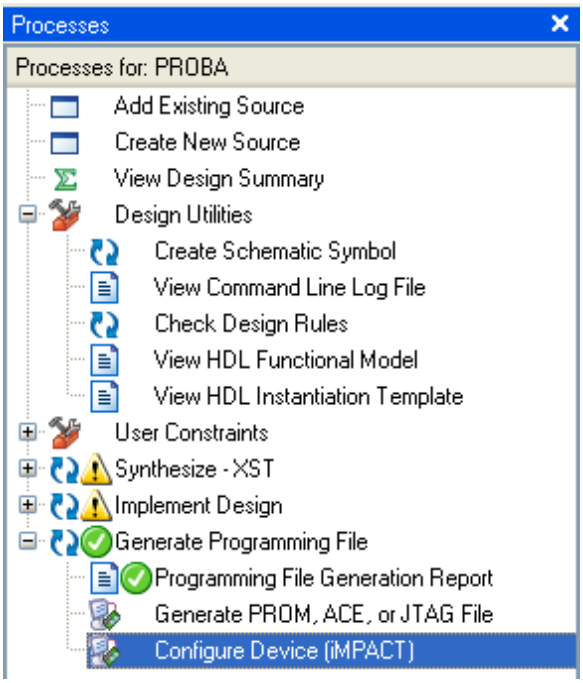


1.12. ábra

Maszk készítésének engedélyezése

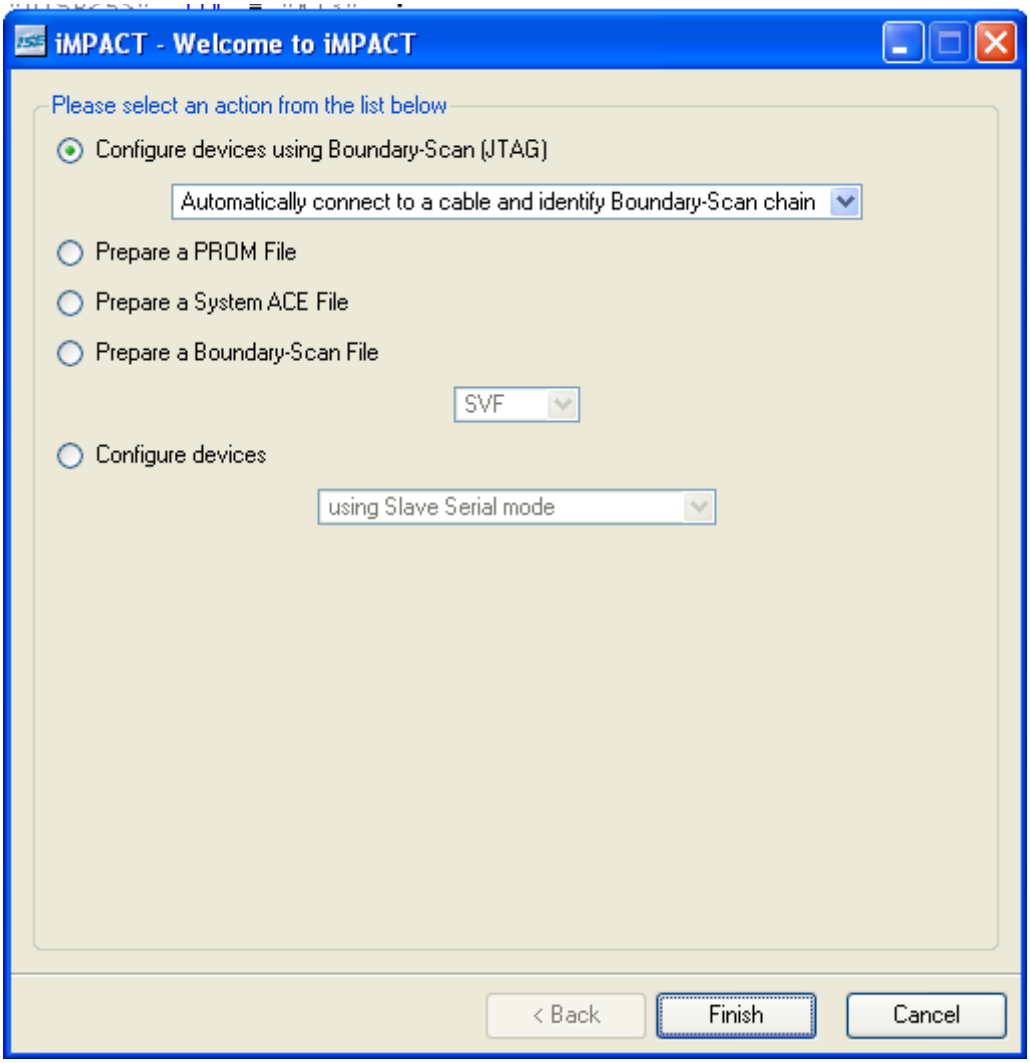
A ReadBack Optionsban jelöljük be a Create Readback Data Files, valamint a Create Mask File opciót, majd nyomjuk meg az OK gombot (amennyiben ezt a lépést kihagyjuk, nem fog működni a Verify funkció)!

Kattintsunk duplán a Generate Programming File-ra, majd azon belül a Configure Device (iMPACT)-re!



1.13.1 ábra

PC csatlakoztatása a panelhez

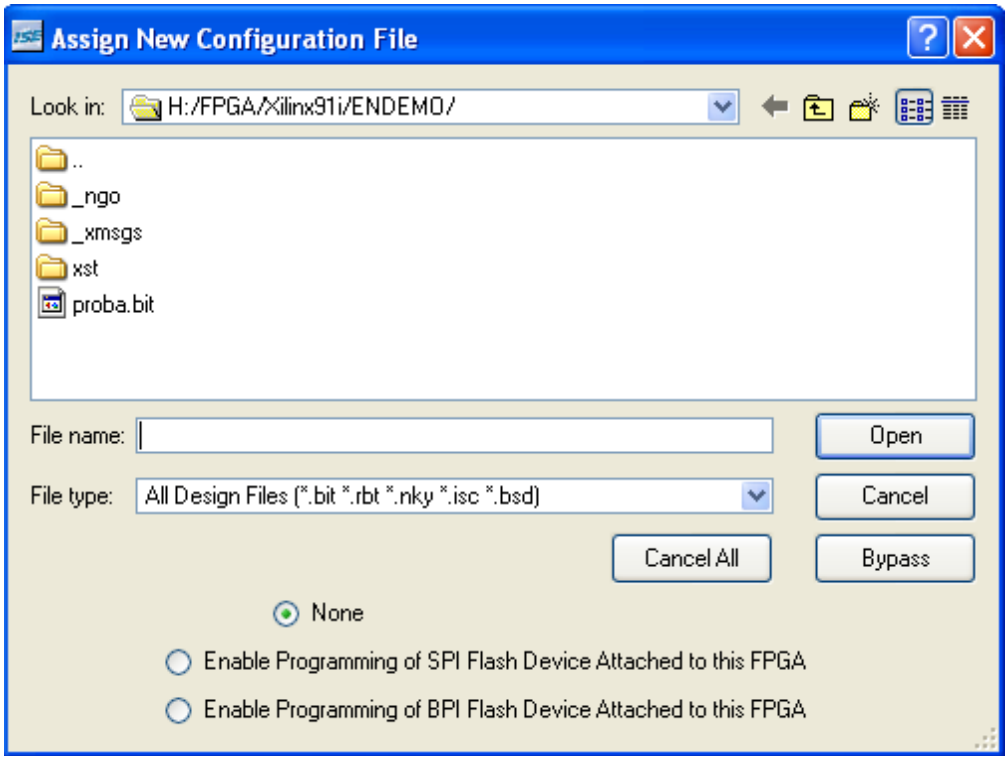


1.13.2. ábra

Csatlakoztatás Boundary Scan módszerrel

A Configure devices Boundary-Scan (JTAG) legyen kijelölve, majd Finish!

Automatikusan megnyílik a következő ablak (ha mégse akkor jobb egérgombbal kattintsunk az IC-n, majd vállasszuk az Assign New Configuration File-t):

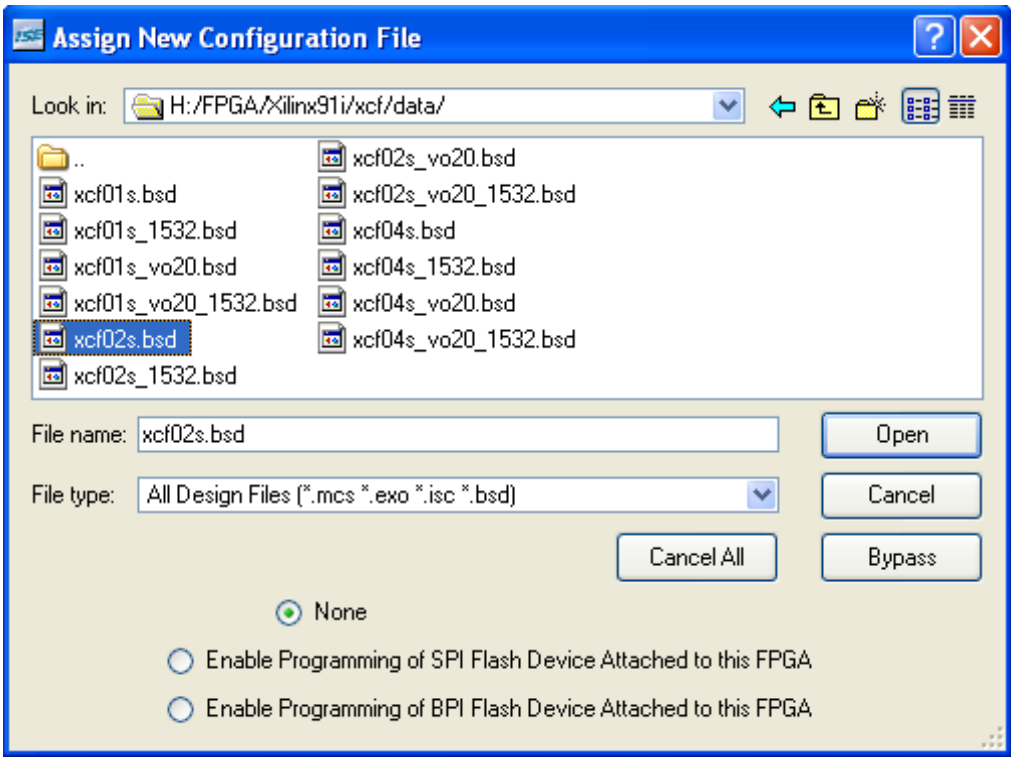


1.14. ábra

Program kiválasztása a beégetéshez

Válasszuk ki a fileunkat, majd Open!

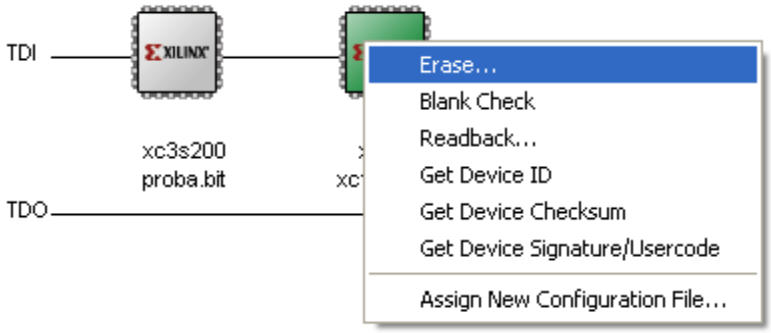
A következő ablakban egy az FPGA-hoz tartozó gyári filet kell megnyitni (xcf02s.bsd):



1.15. ábra

FPGA gyári leíró file kiválasztása

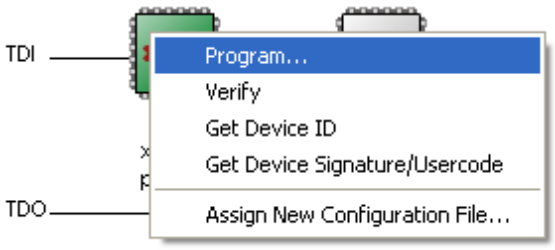
A jobb oldali IC-n jobb egérgombbal kattintva töröljük az FPGA flash memóriáját (Erase).



1.16.1. ábra

FPGA flash memóriájának törlése

A bal oldali IC-n kattintva égessük be programunkat az FPGA-ba!



1.16.2. ábra

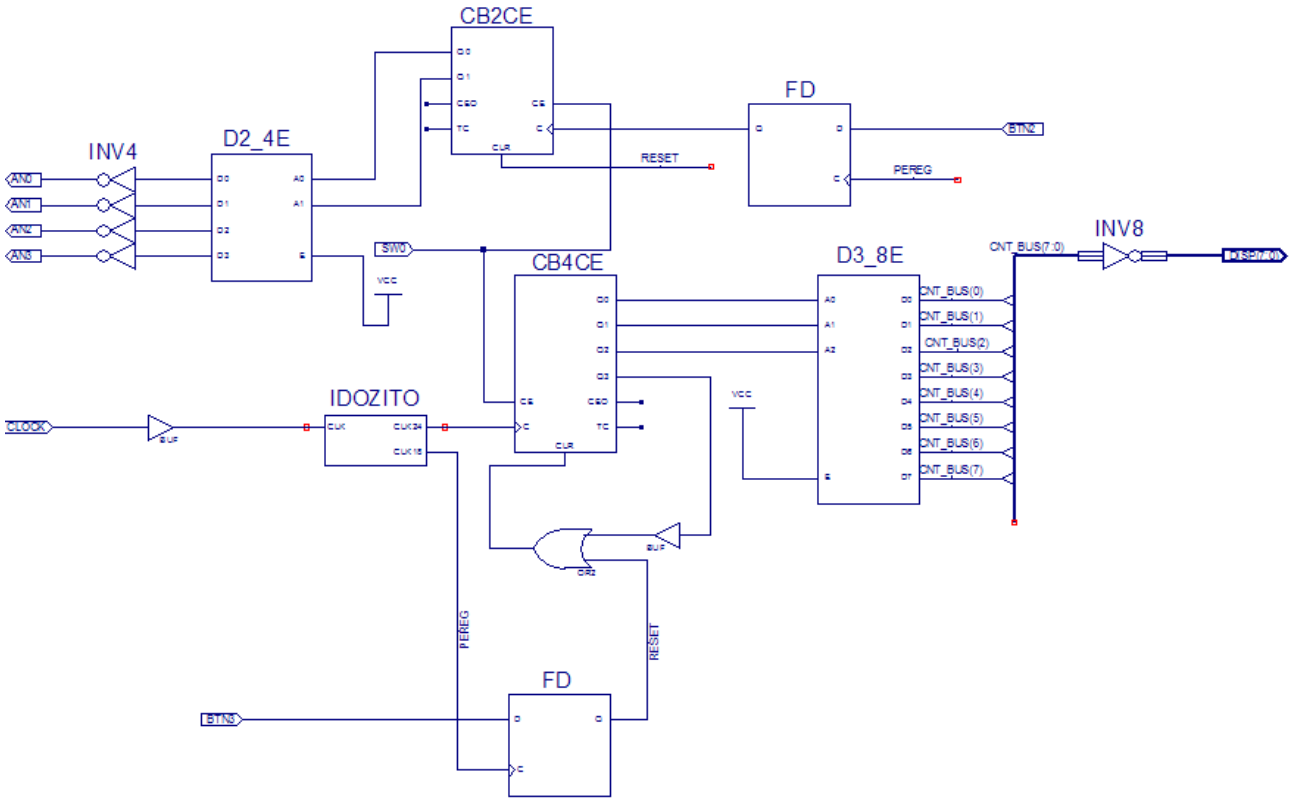
Program beégetése az FPGA-ba

Amennyiben változtat a programon mindig mentse el, majd újra generálja a programozó filet!

1. Feladat:

- Figyelje meg a programot, javítsa a hibát!
- Írja át a programot úgy, hogy a BTN3 nyomógomb hatására a program reszeteljen (a nyomógombot pergésmentesítse; a lábkiosztást is változtatni kell)!
- Alaphelyzetben minden kijelző be van kapcsolva bővítse a programot, hogy egyszerre mindig csak egy kijelző legyen bekapcsolva. A kiválasztást a BTN2 nyomógombbal lehessen megtenni (a lábkiosztást is változtani kell AN0-3)!

  Megoldás:

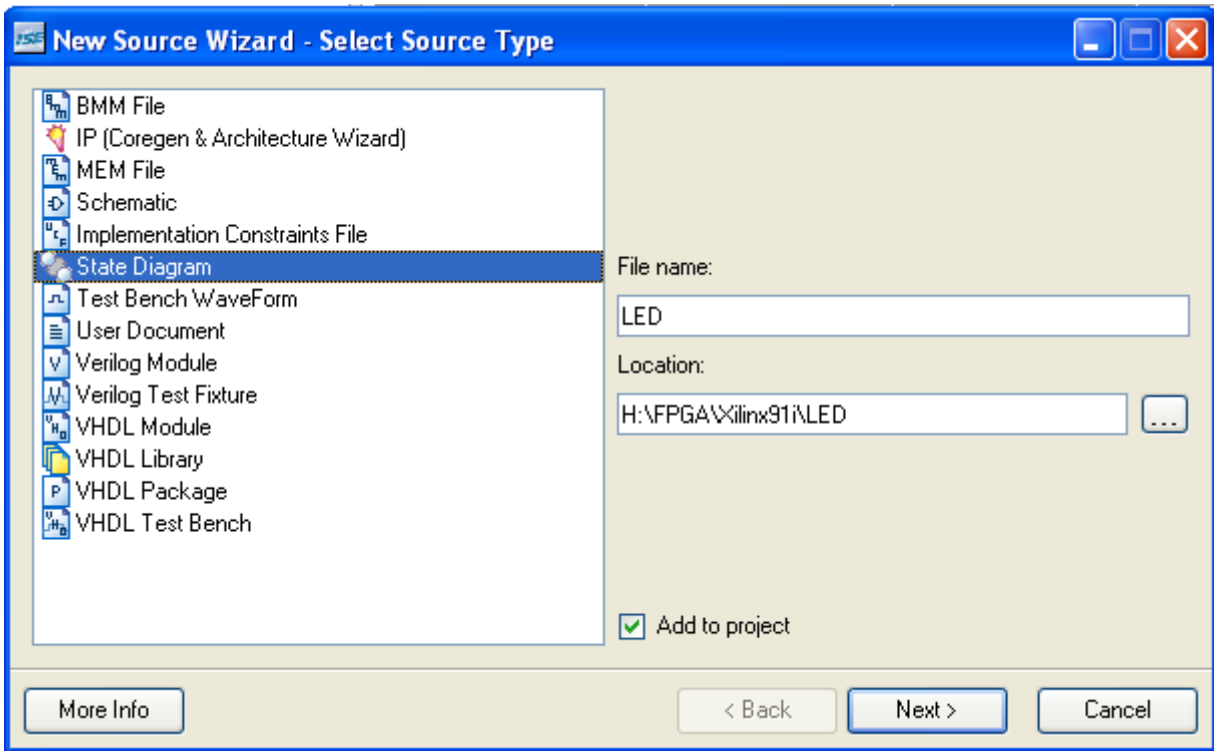


1.17. ábra

Az első feladat egy lehetséges megoldása

Kapcsolja be egyenként a LED-eket is StateCad segítségével (a feladathoz használhat új projektet, de akár az előző programba is implementálhatja a megoldást)!

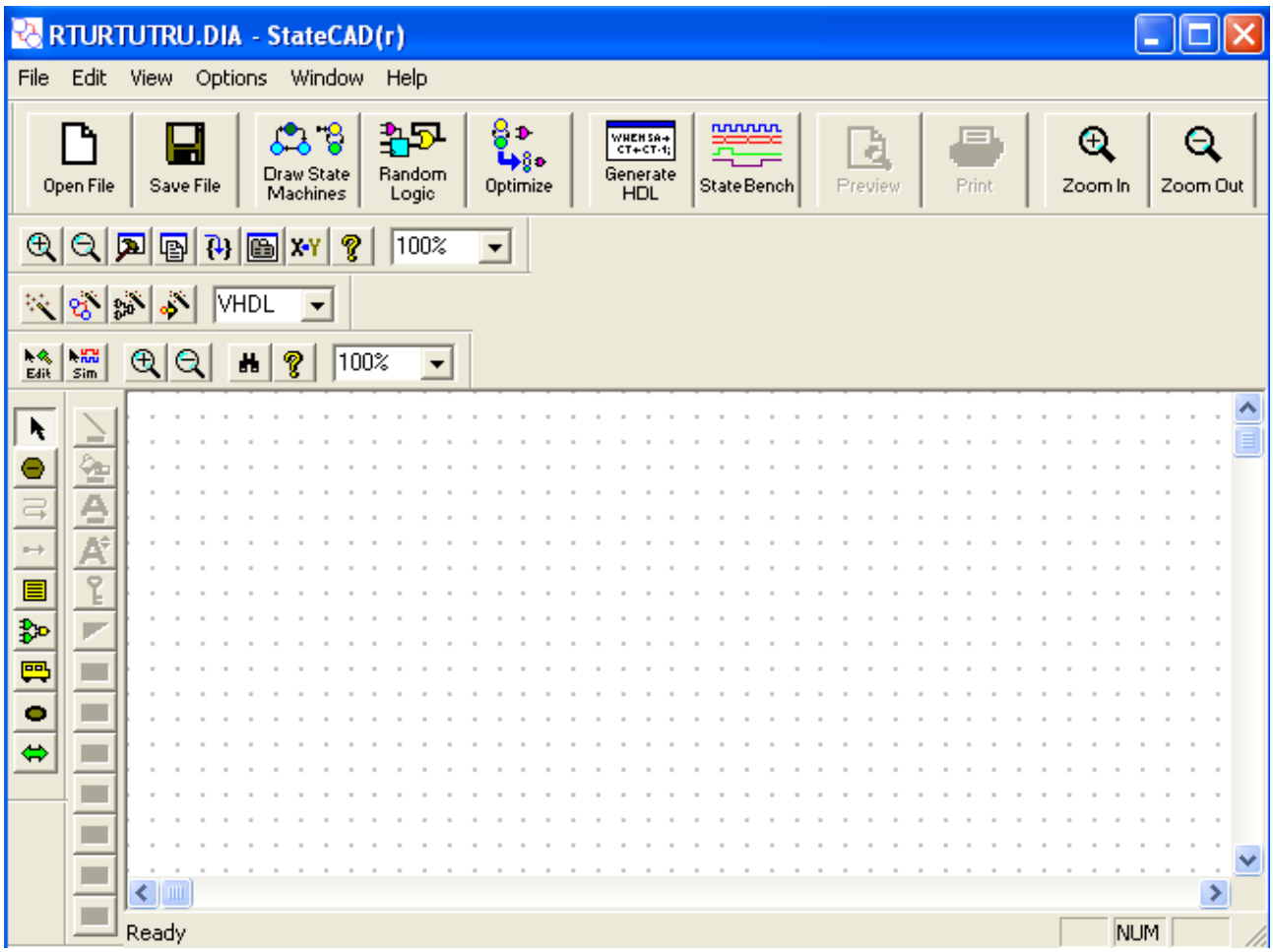
Hozzon létre egy **State Diagram** típusú új forrást!



2.1. ábra

Állapotdiagram típusú forrás létrehozása

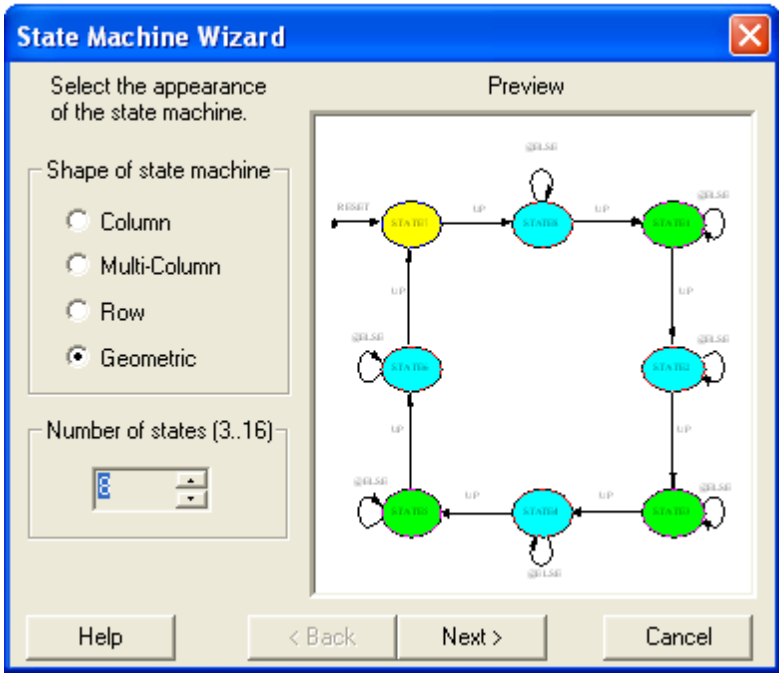
A fejlesztőkörnyezet automatikusan megnyitja az állapotdiagram-szerkesztőt:



2.2. ábra

StateCad szerkesztő kezelőfelülete

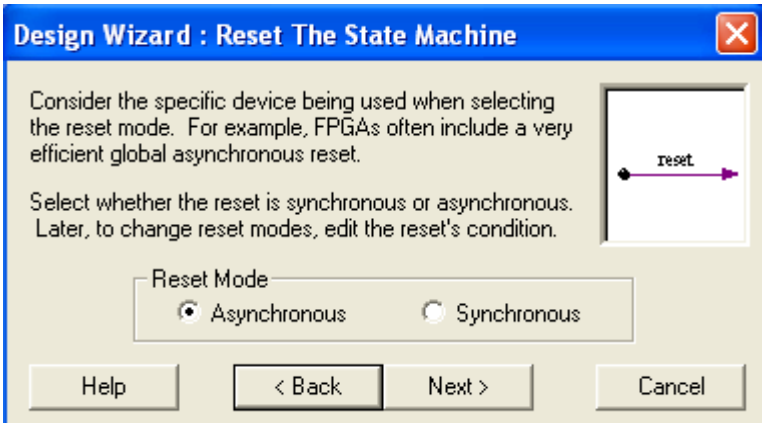
A **Draw State Machines** ikonra kattintva hozza létre az alábbi geometriai elrendezést:



2.3.1. ábra

Állapotgép geometriai elrendezése

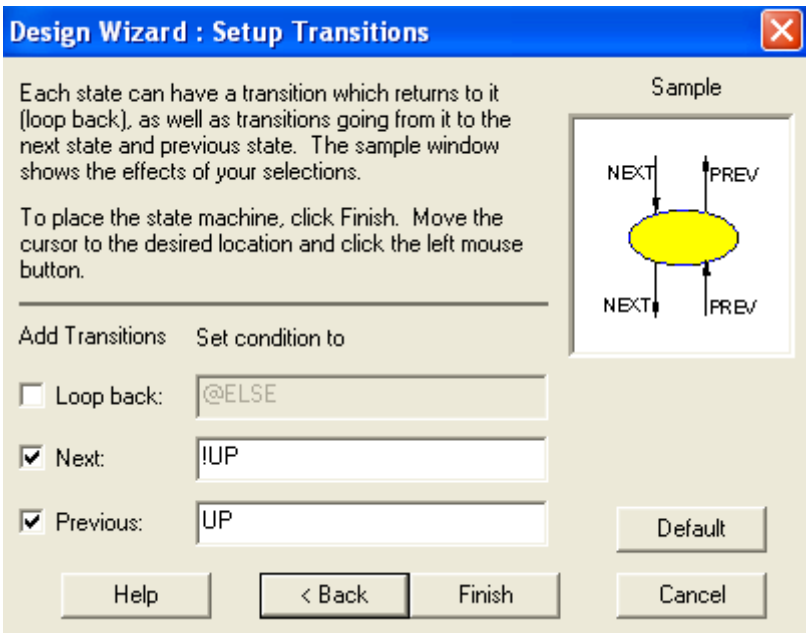
A **Next**re kattintva reszetelési módként válasszunk aszinkront!



2.3.2. ábra

Állapotgép reszetfeltétele

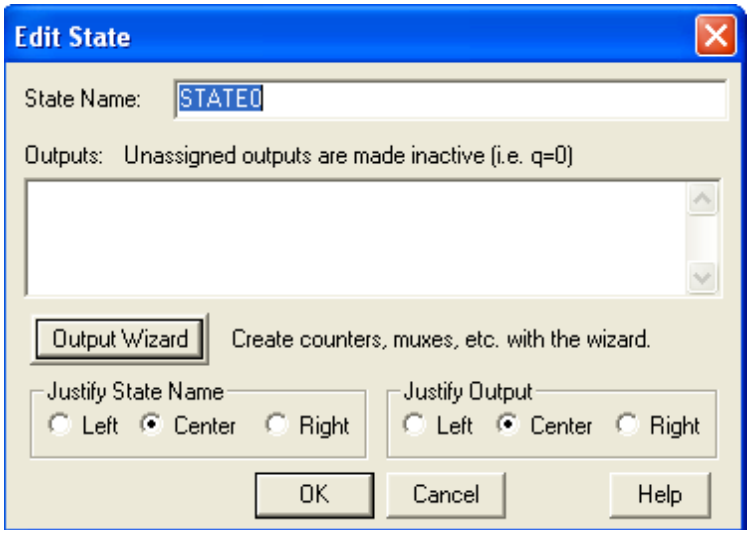
A **Setup Transitions** ablakban jelöljük be a **Next**-et és a **Previous**t, és válasszuk az **!UP**, ill **UP** funkciót!



2.3.3. ábra

Állapotgép futásának beállítása

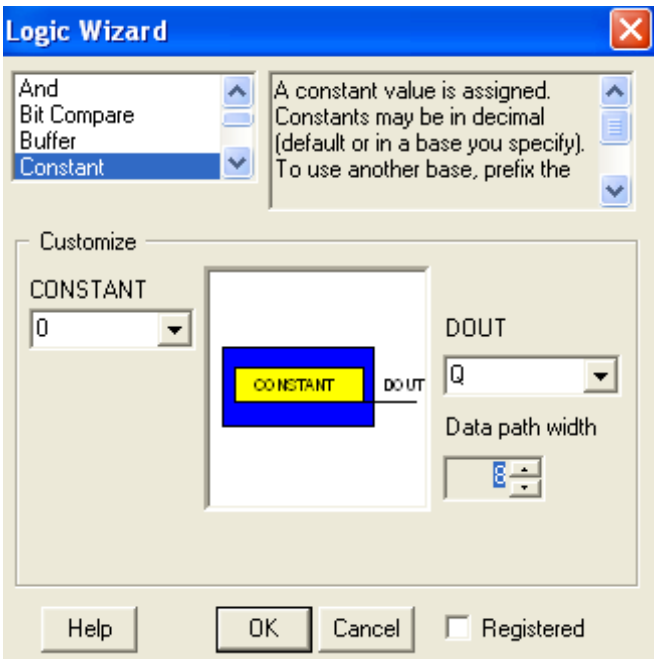
Az egyes állapototkon kettőt kattintva hozzuk elő az **Output Wizardot**:



2.4.1 ábra

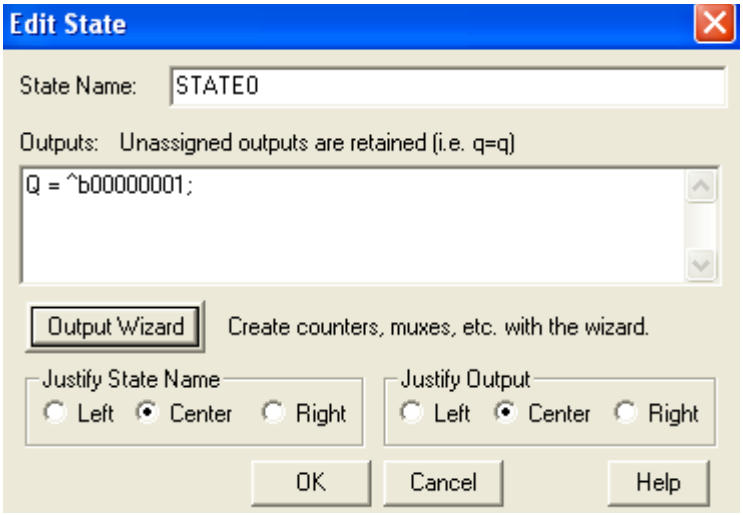
Egyes állapotokhoz kimenet rendelése

Az állapototkat konstansként adjuk meg az elnevezés tetszőleges, de nem szerepelhet két különböző állapotnál ugyanaz a név. Az adatszélesség 8.



2.4.2. ábra

Kimenet típusának és szélességének megadása



2.4.3. ábra

Kimeneti állapot értékének megadása

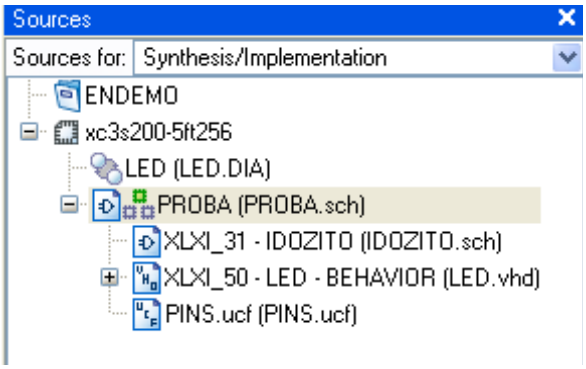
Binárisan írjuk be az általunk kívánt értékeket minden állapothoz!

Ezután kattintsunk az **Optimize** ikonra (itt állíthatjuk be, hogy a fordító milyen szempontok szerint optimalizálja az állapotgépet)! Céleszközként **CPLD**-t, valamint célforrásként **VHDL**-t válasszunk!

Kattintsunk a **Generate HDL** ikonra, majd sikeres lefordulás után adjuk hozzá a projectünkhöz a *.dia, és a *.vhd fileokat a projekt **Add Copy of Source** menüpont segítségével (amennyiben nem vagyunk biztosak az állapotgép helyes működésébe, szimuláljuk le a **StateBench**

segítségével)!

Ha minden lépést helyesen csináltunk végig, megjelenik a két file a forrásaink között:

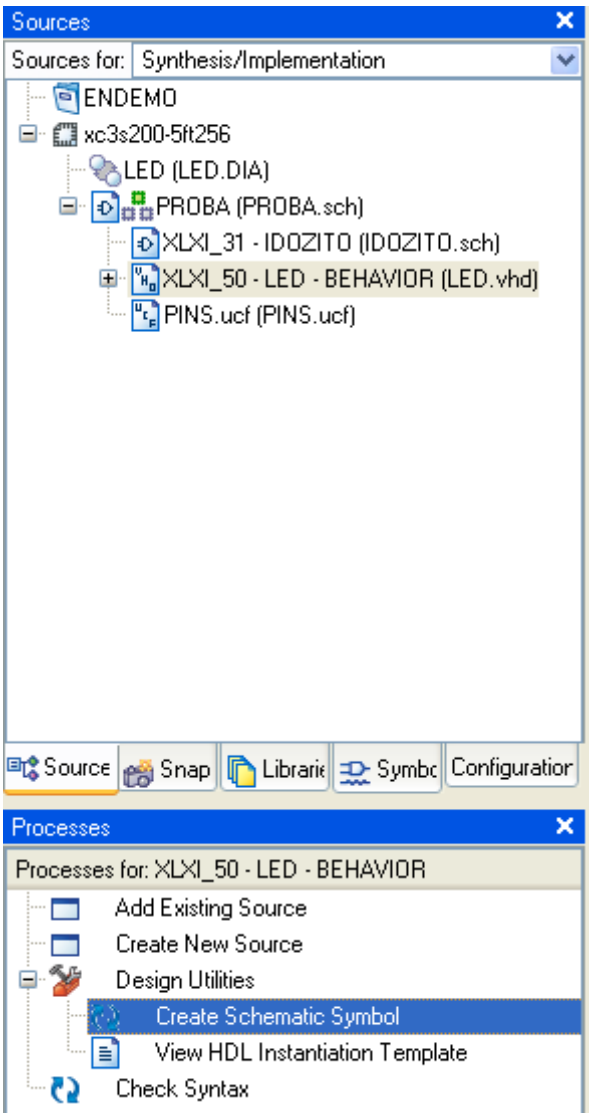


2.5. ábra

StateCad file, és annak

VHDL forrása a projektben

Rendeljünk saját alkatrészt a *.vhd fileunkhoz:

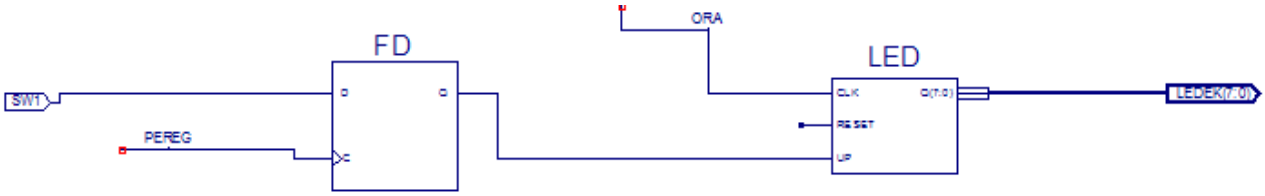


2.6. ábra

Szimbólum rendelése

VHDL forráshoz

Valósítsa meg a következő áramkörrészletet:



2.7. ábra

A program áramköri rajza

Pereg; ill ora: CLK18;CLK24 az időzítőáramkör kimenetei!

Próbálja ki SW1 kapcsoló hatását!

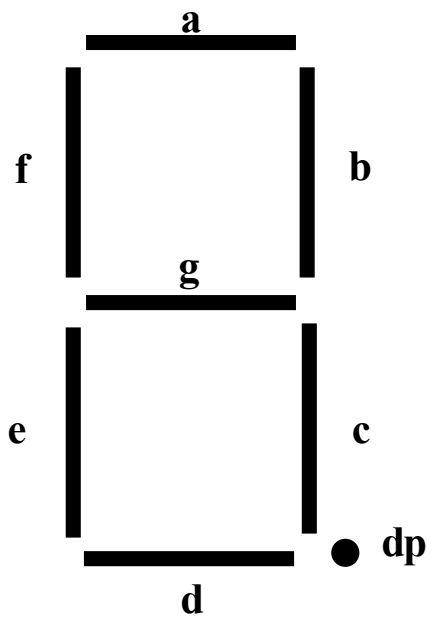
2. Feladat:

- Írja át a programot, hogy a fény oda-vissza fusson, és az irányt az SW2 kapcsolóval lehessen beállítani!
- Fusson oda-vissza a fény, de azok a LED-ek ne világítsanak, melyekhez rendelt SW kapcsoló aktív (pl: SW3 és SW4 aktív, akkor LD3 és LD4 nem világít)
- Az előző két programot egyesítse, úgy, hogy a BTN2 nyomógombbal lehessen váltani közöttük!
- Valósítson meg Knight-Rider futófényt a LED-eken! *

Készítsünk a 7-szegmens kijelzőkön egyszerű számláló programot!

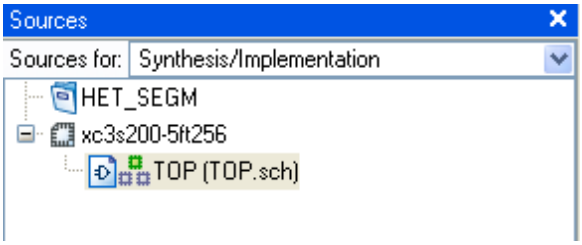
Az előző ismereteinket bővítsük egy kis VHDL programozással! Írjunk egy algoritmust, mely a bemenetére érkező 4 bites bináris számból előállítja annak 7-szegmenses kódját!

7-szegmens kódtáblázat közös anódos kijelzőre:



Bináris kód	*Számjegy*	*Vezérlőkód*
		Dp g f e d c b a
0000	0	1 1 0 0 0 0 0 0
0001	1	1 1 1 1 1 0 0 1
0010	2	1 0 1 0 0 1 0 0
0011	3	1 0 1 1 0 0 0 0
0100	4	1 0 0 1 1 0 0 1
0101	5	1 0 0 1 0 0 1 0
0110	6	1 0 0 0 0 0 1 0
0111	7	1 1 1 1 1 0 0 0
1000	8	1 0 0 0 0 0 0 0
1001	9	1 0 0 1 0 0 0 0
1010	A	1 0 0 0 1 0 0 0
1011	b	1 0 0 0 0 0 1 1
1100	C	1 1 0 0 0 1 1 0
1101	d	1 0 1 0 0 0 0 1
1110	E	1 0 0 0 0 1 1 0
1111	F	1 0 0 0 1 1 1 0

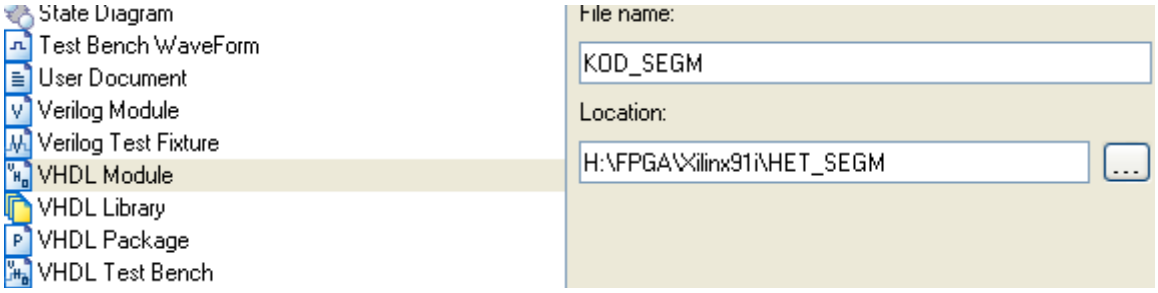
Hozzunk létre egy új projektet, az elsődleges forrás legyen ismét schematic!



3.1. ábra

Projekt és top forrás létrehozása

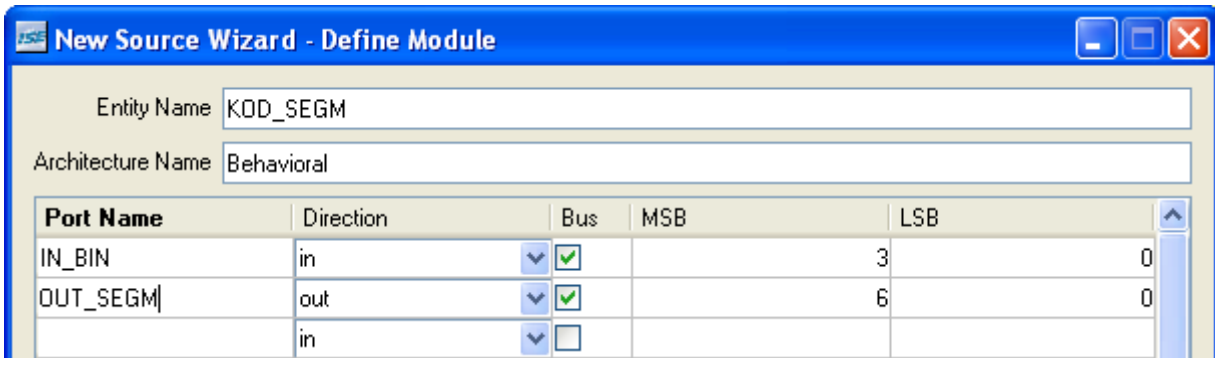
Új forrásként adjunk a projektünkhöz egy VHDL típusú filet:



3.2.1 ábra

VHDL modul létrehozása

Adjuk meg a VHDL forrás be- és kimeneteit, valamint azok szélességét:

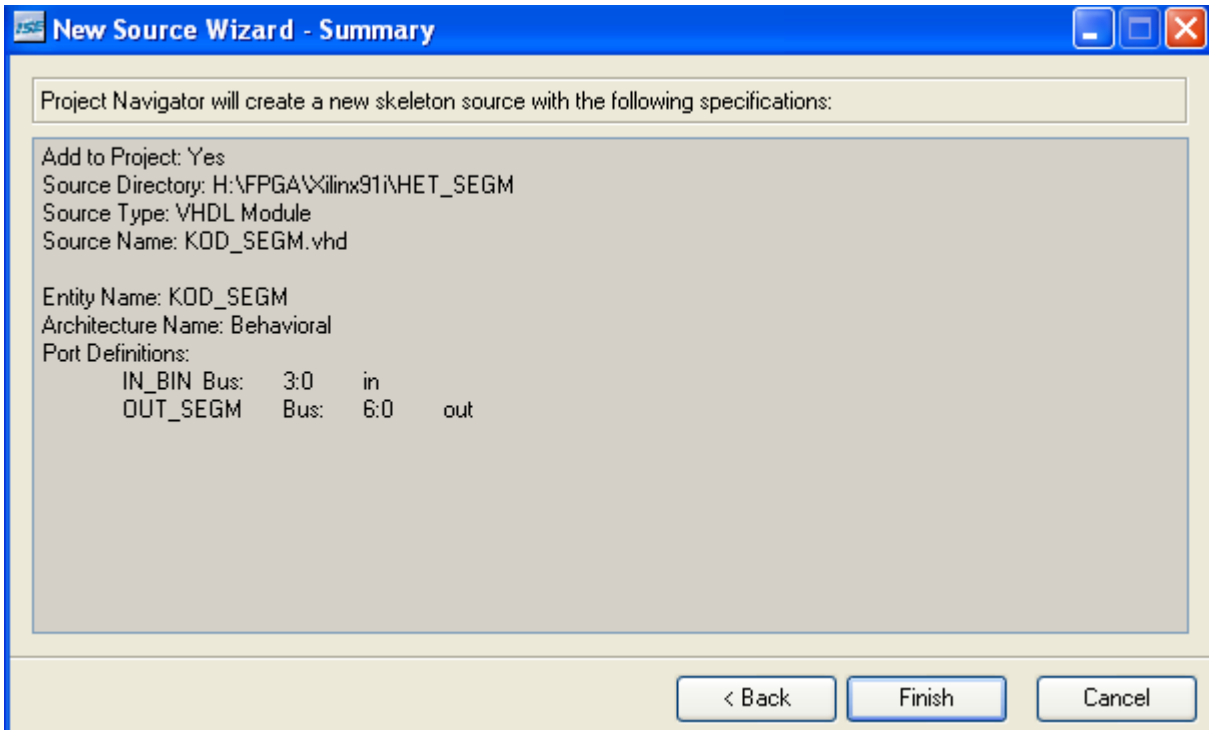


3.2.2. ábra

VHDL modul inicializálása

Bemenet: bináris kód (4 bites).
Kimenet: 7-szegmens vezérlőkód (7 bites, jelenleg a tizedespontot nem használjuk)

A következő ablakban egy összegzés látható a beállításainkról, ez alapján a program elkészíti a VHDL modul inicializálását, nekünk már csak a funkcionális programot kell megírnunk:



3.2.3. ábra

VHDL modul összegzése

```
----------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    09:04:29 05/29/2007
-- Design Name:
-- Module Name:    KOD_SEGM - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity KOD_SEGM is
    Port ( IN_BIN : in  STD_LOGIC_VECTOR (3 downto 0);
           OUT_SEGM : out  STD_LOGIC_VECTOR (6 downto 0));
end KOD_SEGM;

architecture Behavioral of KOD_SEGM is

begin


end Behavioral;
```

3.3.1. ábra

A fejlesztőkörnyezet által generált VHDL kód

A VHDL kódot a begin és az end Behavioral közé írjuk be, amennyiben „--” karakterek után a fordító nem veszi figyelembe a szöveget (kommentnek)!

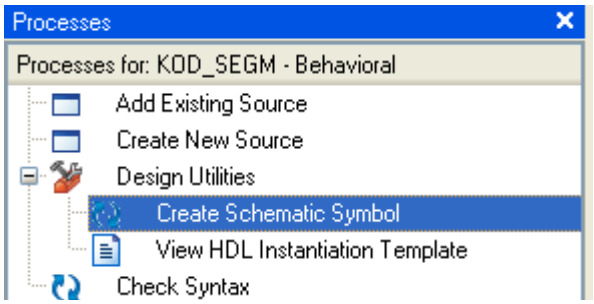
```
begin

    with IN_BIN SELect

    OUT_SEGM<=  "1111001" when "0001", -- 1
                "0100100" when "0010", -- 2
                "0110000" when "0011", -- 3
                "0011001" when "0100", -- 4
                "0010010" when "0101", -- 5
                "0000010" when "0110", -- 6
                "1111000" when "0111", -- 7
                "0000000" when "1000", -- 8
                "0010000" when "1001", -- 9
                "0001000" when "1010", -- A
                "0000011" when "1011", -- b
                "1000110" when "1100", -- C
                "0100001" when "1101", -- d
                "0000110" when "1110", -- E
                "0001110" when "1111", -- F
                "1000000" when others; -- 0



end Behavioral;
```

3.3.2. ábra

Az általunk írt VHDL kód

A fenti kódsorozat egy switch-case szerkezet (jobb oldalt a gerjesztés, bal oldalt az erre adott válasz látható).

Készítsünk sematikus szimbólumot VHDL forrásunkból:

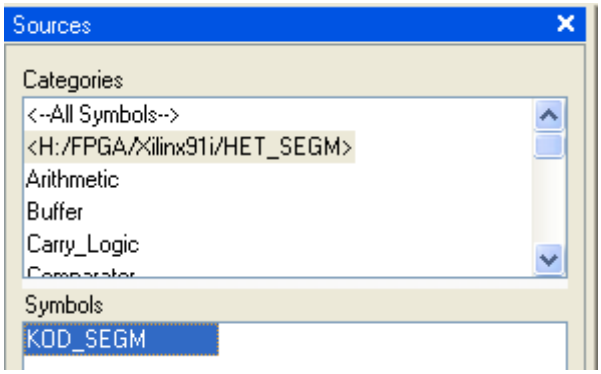


3.4.1. ábra

Sematikus model létrehozása

VHDL forrásból

Ellenőrizzük, hogy alkatrészünk megjelent a szimbólumok között:

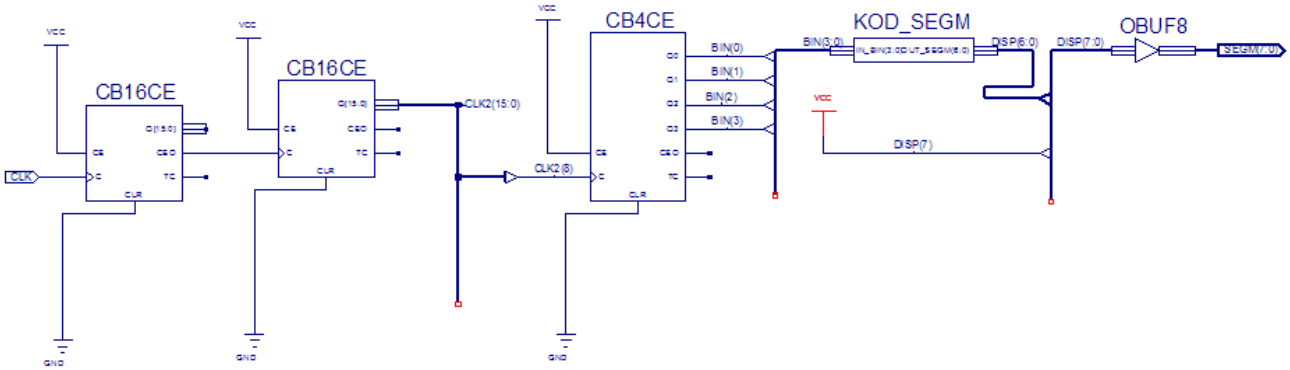


3.4.2. ábra

VHDL forrásból elkészített alkatrész

a project szimbólumai között

A program egy lehetséges megvalósítása:



3.5. ábra

A feladat megoldása

Az első két blokkot helyettesíthetjük az első feladat során megalkotott órajelosztó áramkörrel, valamint akár abba a projectbe is implementálhatjuk VHDL modulunkat a gyorsabb fejlesztés érdekében.

Feladatok:

- Oldja meg, hogy egy kapcsolóval (SW0) lehessen kiválasztani, hogy felfelé, vagy lefelé számlálás történjen!
- Egy másik kapcsolóval (SW1) át lehessen állítani a számlálást decimálisra!
- SW2 aktív esetén a számlálás „helyiérték-helyes” legyen, vagyis készítsen valódi számlálót a hétszegmens kijelzőre (a tesztelés rövidítése érdekében növelje meg az órajelet).
- Valósítsa meg saját ötletét (nyomógombok számlálása, aritmetikai egység, stb.)

Melléklet

Az általunk használt gyakorlópanel lábkiosztása a következő:

Periféria	*Lábszám*	*Megjegyzés*
SW0	F12	Kapcsoló
SW1	G12	Kapcsoló
SW2	H14	Kapcsoló
SW3	H13	Kapcsoló
SW4	J14	Kapcsoló
SW5	J13	Kapcsoló
SW6	K14	Kapcsoló
SW7	K13	Kapcsoló
BTN0	M13	Nyomógomb
BTN1	M14	Nyomógomb
BTN2	L13	Nyomógomb
BTN3	L14	Reset nyomógomb
DISP_LED0	E14	a
DISP_LED1	G13	b
DISP_LED2	N15	c
DISP_LED3	P15	d
DISP_LED4	R16	e
DISP_LED5	F13	f
DISP_LED6	N16	g
DISP_LED7	P16	dp
AN0	D14	DISP0 ENABLE
AN1	G14	DISP1 ENABLE
AN2	F14	DISP2 ENABLE
AN3	E13	DISP3 ENABLE
LED0	K12	
LED1	P14	
LED2	L12	
LED3	N14	
LED4	P13	
LED5	N12	
LED6	P12	
LED7	P11	
CLOCK	T9	Órajel