FPGA fejlesztés a Xilinx ISE Webpack-ben

Írta: Varga László

Szerkesztette: Molnár Zsolt

BMF KVK MAI 2007. október 25.

Tartalomjegyzék

1. Bevezetés

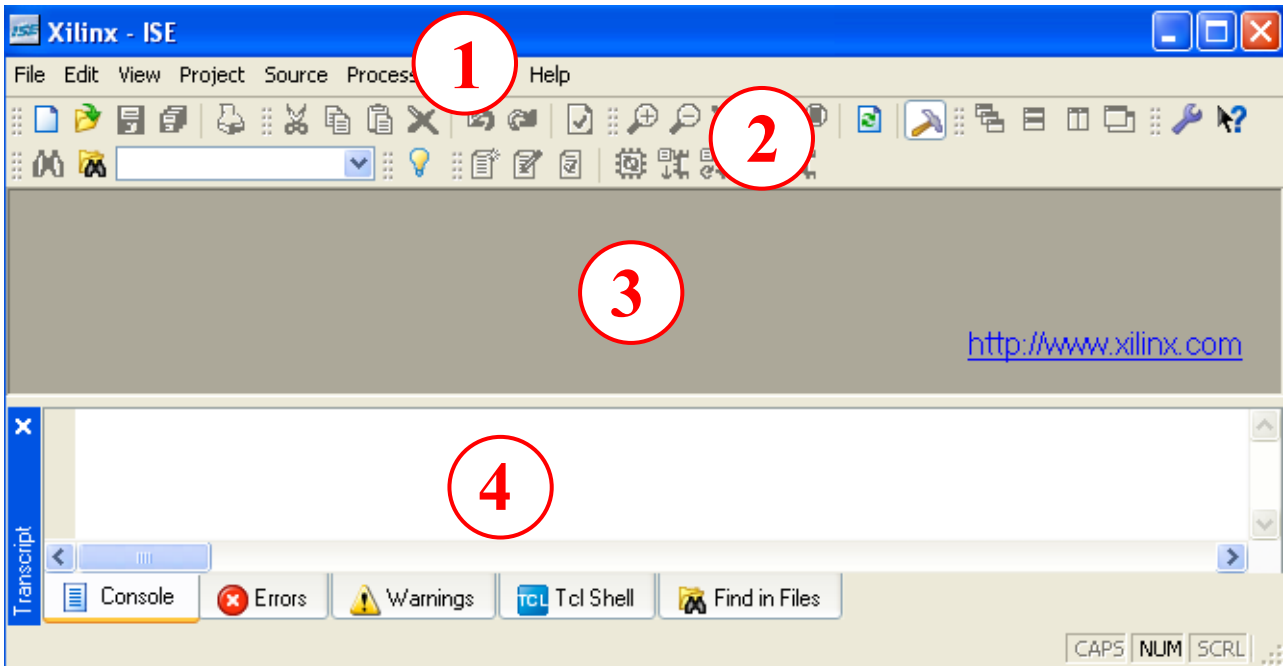
A **Xilinx ISE WebPack** (Integrated Software Environment – integrált szoftverkörnyezet) a Xilinx cég **FPGA**-ihoz és **CPLD**-ihez kifejlesztett ingyenes szoftver, amely az Internetről, a cég weboldaláról szabadon letölthető. A fejlesztőrendszer tartalmazza mindazon elemeket, amelyek **kapcsolási rajz** alapú, **állapotgép** alapú, illetve **hardverleíró nyelv** alapú fejlesztéshez szükségesek. Szimulációra korlátozottan alkalmas a fejlesztőrendszer, de ingyen letölthető a **ModelSim XE**, amely a Xilinx eszközökre szűkített képességű nagy tudású **szimulációs eszköz**.

A segédlet megírásakor a Xilinx ISE WebPack legfrissebb elérhető verziója a 9.1-es. (A fejlesztőrendszer állandó bővítés és fejlesztés alatt van.) A mérési segédletben ennek tulajdonságait vesszük alapul.

A mérési segédlet **bevezetést** kíván nyújtani – az FPGA-k tervezésébe, célja **a tervezési lehetőségek minél szélesebb körben** való ismertetése. Elmélyülésre – a korlátozott laboridő miatt – saját munkával, illetve a konzultációs időpontokban van lehetőség. A mérés sikeres elvégzését nagymértékben segíti a mérést előkészítő előadás anyagának elsajátítása.

A mérés során a SPARTAN-3 típust tartalmazó **demonstrációs panelt** (SPARTAN-3 Starter Board) fogjuk használni. A panelen alkalmazott SPARTAN-3 FPGA egy 256 lábú BGA tokban kapott helyet, kapuszáma 200000, de a kereskedelemben többféle kapuszámmal és tokozásban kapható. A panelen helyet kapott számos kiegészítő egység: 8 db. LED, egy 4 digites 7 szegmenses kijelző, 8 db. kapcsoló, 4 db. nyomógomb, PS2, RS232, és VGA portok, 2db. 256kx16bit SRAM, konfigurációs FLASH, és egy 50MHz-es órajel-forrás. Az FPGA összes I/O lába a panel peremén elhelyezett csatlakozókra van kivezetve. A konfigurációs bitminta **letöltésére** a laboratóriumban egy **JTAG letöltő kábel** szolgál.

A fejlesztőkörnyezet elindítása a *Start* menüből (*Xilinx ISE 9.1*), vagy közvetlen link segítségével történhet. A program elindítása után a következő **felülettel** találkozunk:



1. ábra

A Xilinx ISE Webpack kezelőfelülete

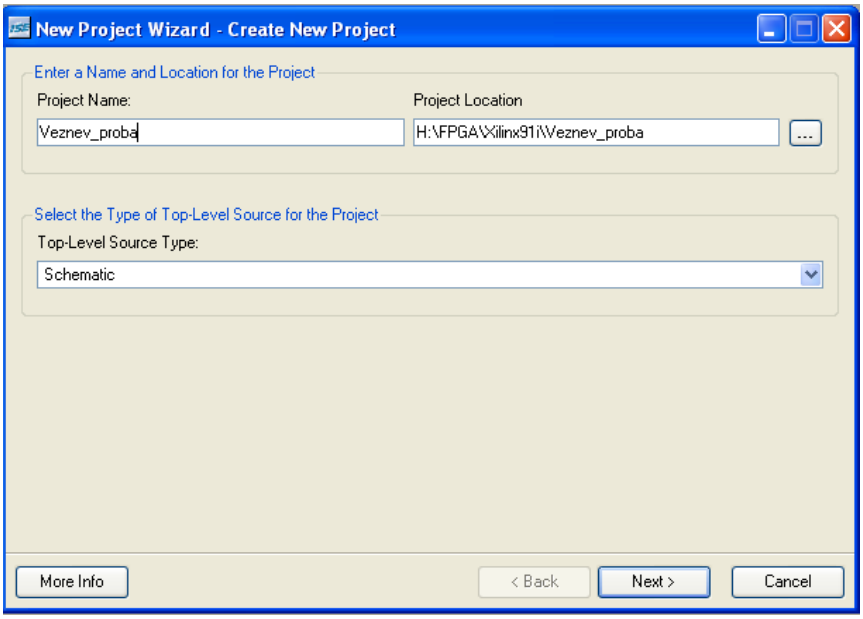
A felhasználói felület felosztása:

1. Menüsor
2. Fontosabb parancsikonok
3. Munkafelület
4. Eszköztárak (ki/be kapcsolhatóak, helyük változtatható)

2. Mintafeladat megoldása – kapcsolási rajz alapú tervezés

A Xilinx ISE WebPack 9.1 segítségével hozzunk létre egy egyszerű példaprogramot a gyakorlópanelen található négydigites hétszegmenses kijelző és a LED-ek tesztelésére! A feladat első részét kapcsolási rajz alapon, a második részét állapotdiagramos tervezéssel mutatjuk be.

Első lépésként készítsünk el egy **új projekt fájlt** (*File→New Project*). Amennyiben rendelkezünk már projekt fájllal nyissuk meg az *Open project* menüponttal! A fejlesztőkörnyezet **minden projektnek külön könyvtárat** hoz létre, és a „Xilinx91i" mappába menti. Projektünk neve legyen *„vezetéknevünk_proba"* (javaslat), **legmagasabb szintű** forrásként **kapcsolási rajz** alapú (schematic) típust adjunk meg!



2. ábra

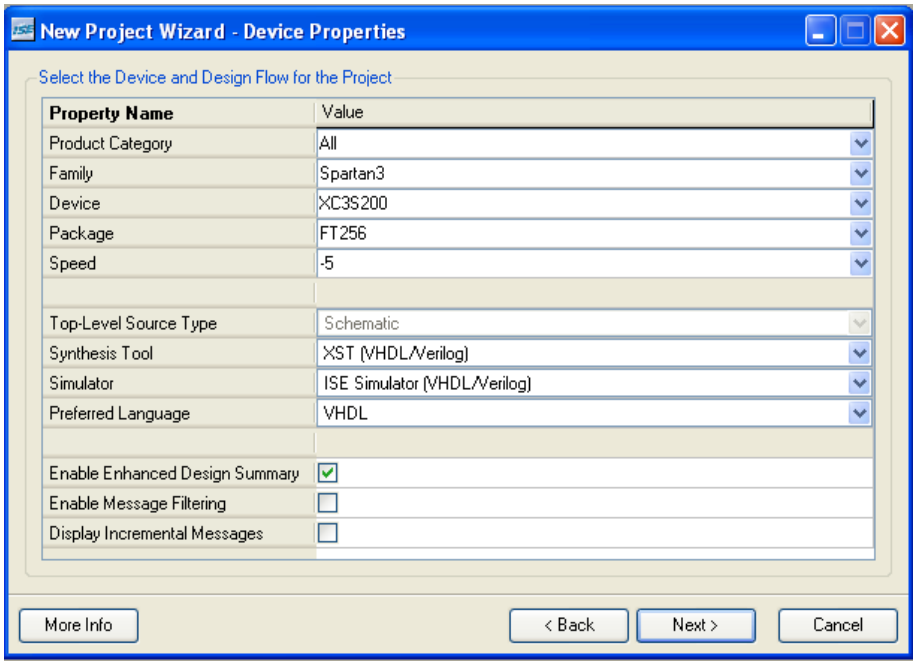
Új projekt létrehozása

Az általunk elkészíteni kívánt program modulokból épül fel, minden modulnak **háromféle forrása** lehet:

- **Kapcsolási rajz** (Schematic)
- **Hardverleíró nyelv** (VHDL, Verilog)
- **Állapotdiagram** (StateCad)

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

A következő ablakban annak a félvezető eszköznek a tulajdonságait állíthatjuk be, amelyre fejlesztünk:

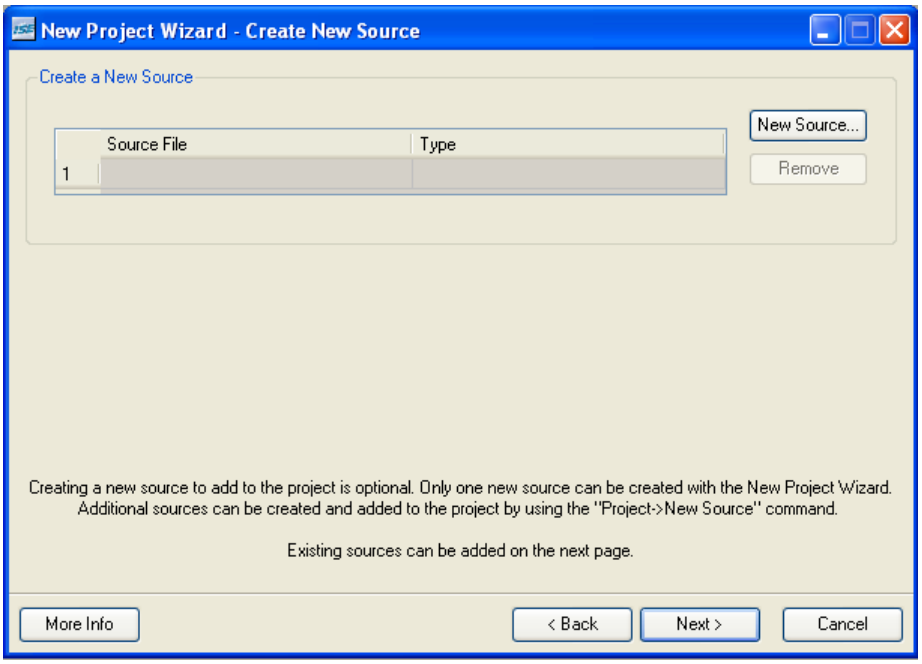


3. ábra

FPGA tulajdonságainak beállítása

Forrást a *Next* gombra kattintva **hozhatunk létre**. Ezt megtehetjük most, vagy akár a későbbiek folyamán a *Project→New Source…* menüponttal! (Egyenlőre ne hozzunk létre forrást!)

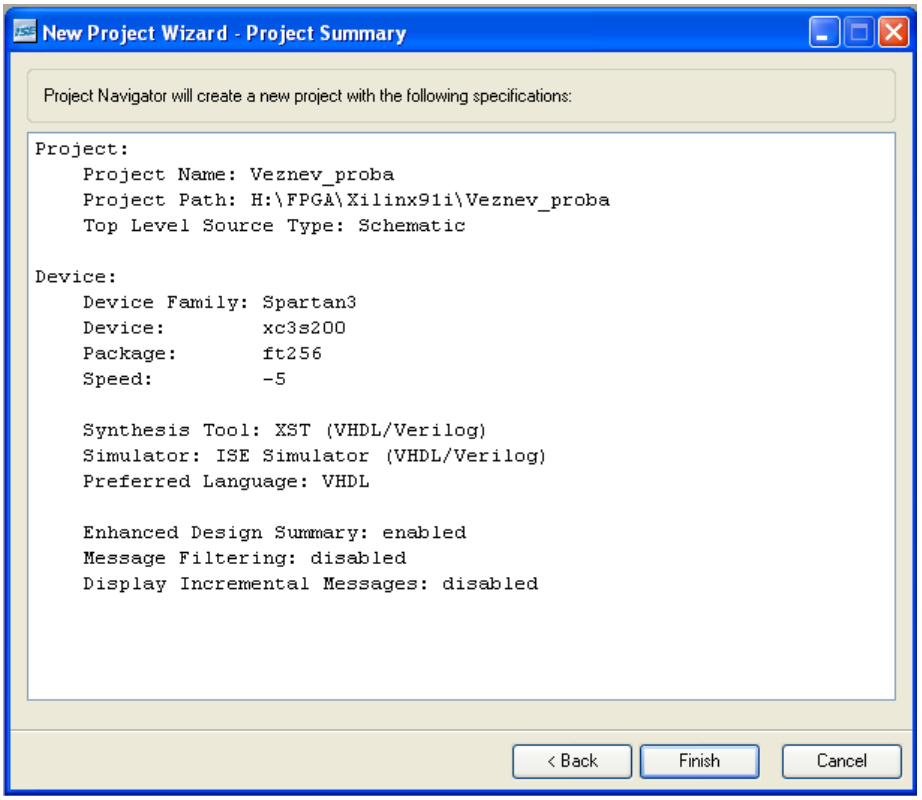
A következő ablakban **meglévő forrást adhatunk hozzá** a projekt fájlunkhoz. E lépés végrehajtására is van lehetőségünk a későbbiek folyamán a *Project→Add Source…* menüponttal.



4. ábra

Új forrás létrehozása

Miután ismét a *Next* gombra kattintottunk, a projektünk **összefoglalója** jelenik meg:

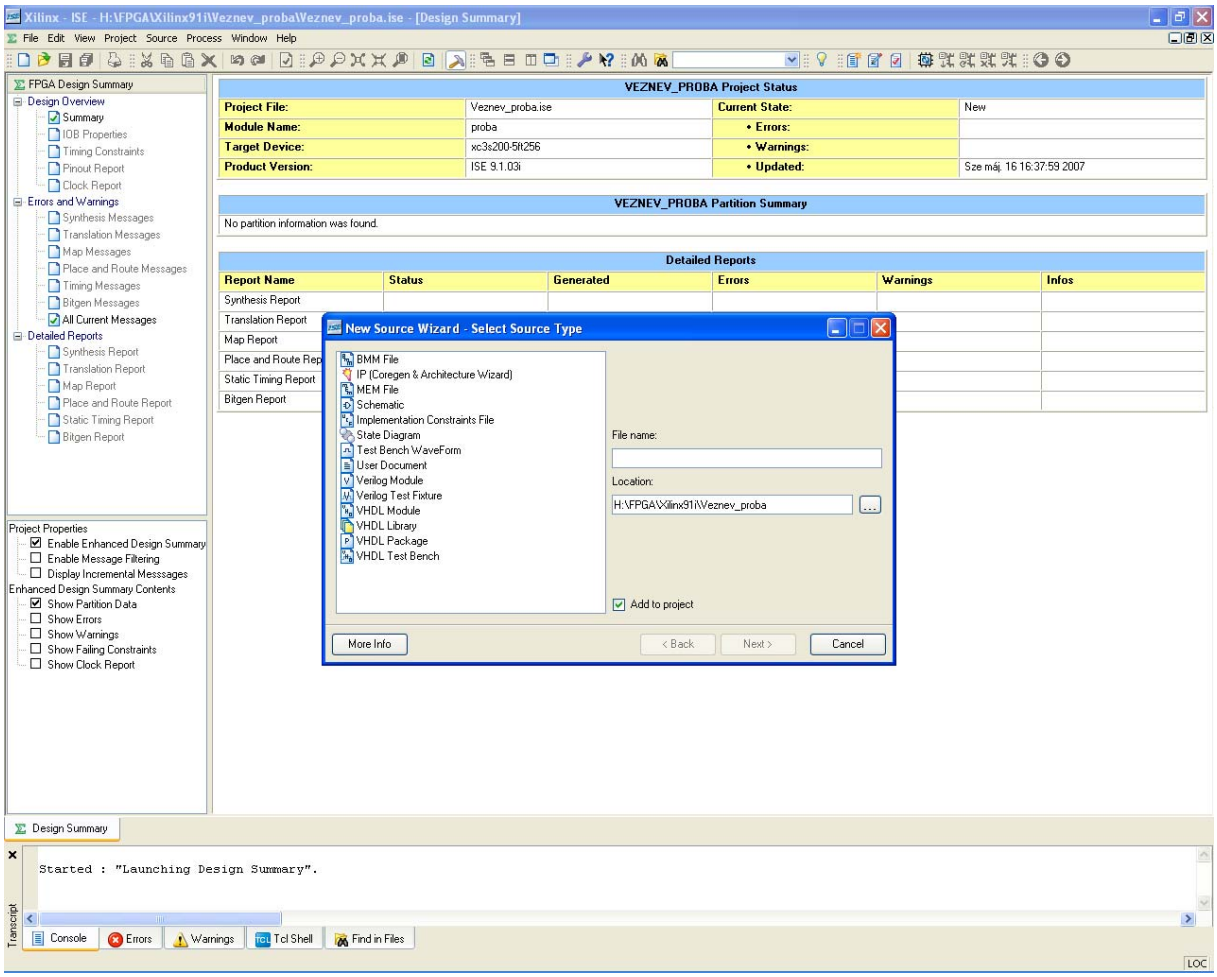


5. ábra

Projekt tulajdonságainak összegzése

Ellenőrizzük, a projekt, valamint az eszköz **beállításait**! Amennyiben mindent rendben találunk, kattintsunk a *Finish* gombra!

Hozzunk létre a projectünkhöz **forrás fájlt** (*Project→New Source...*)! Legyen a **forrásunk típusa schematic**, a neve tetszőleges[1]! Amennyiben nem történt meg automatikusan, adjuk hozzá a forrás fájlt a projektünkhöz (*Project→Add Source...*)



6. ábra

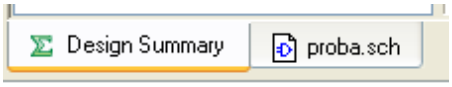
Új forrás létrehozása

A *View* menüpontban **állítsuk aktívra** a források (*Sources*), és a folyamatok (*Processes*) ablakát! Ennek hatására ezek megjelennek baloldalon.[2]

[1] Mivel később ez lesz a legfelső szintű modul, célszerű a nevében szerepeltetni a „top” megjelölést, pl. „top_pelda” (szerkesztői megjegyzés)

[2] Az előzetes megjelenítési beállításoktól függően, nem biztos, hogy szükség van az ablakok aktiválására, illetve nem biztos, hogy a fent említett helyen jelennek meg. (szerkesztői megjegyzés)

Váltsunk kattintással a forrásunk (itt: proba.sch) fülére:



7. ábra

Váltás a forrásra

Kezdjünk hozzá a programozáshoz! **Első programunk** a következő **funkciókat** fogja megvalósítani:

- **Bekapcsolja** az egymás után a hétszegmenses kijelzők egyes **szegmenseit**
- **Bekapcsolja** egymás után a gyakorlópanelen lévő **LED**-eket
- Mindezt egy **kapcsolóval leállíthatjuk** menet közben, ilyenkor őrzi az előző állapotokat

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

Az órajel leosztó áramkört **kapcsolási rajz alapú szerkesztővel definiáljuk**:

1. **Hozzunk létre** egy új forrást: *Project→New Source…* (típusa: schematic, neve (példánkban: „idozito.sch”))[3]!

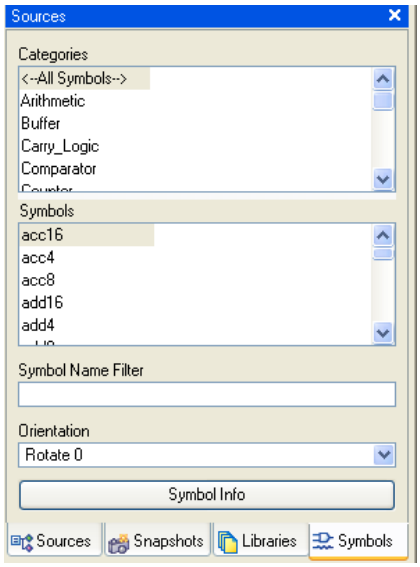
2. Amennyiben nem történt meg automatikusan, **adjuk hozzá** ezt a forrást a projektünkhöz *Project→Add Source*! A további lépések során **ügyeljünk** arra, hogy **a most létrehozott forrás** füle **legyen aktív**!

3. **Válasszuk ki** a szükséges **alkatrészeket** a baloldalon található Sources ablak Symbols fülének segítségével.

   Az alkatrészek megkeresése történhet **kategóriák** alapján, de használhatjuk a

[3] Javasoljuk, hogy a könnyű tájékozódás miatt az elnevezésben szerepeljen a „oszto” v. „osztó” szó. Ügyeljünk, hogy ne a mostani, hanem az előzőekben létrehozott modul maradjon a TOP modul. Beállítás: jobb kattintás a forrás felett, majd *Set As Top Module (szerkesztői megjegyzés)*

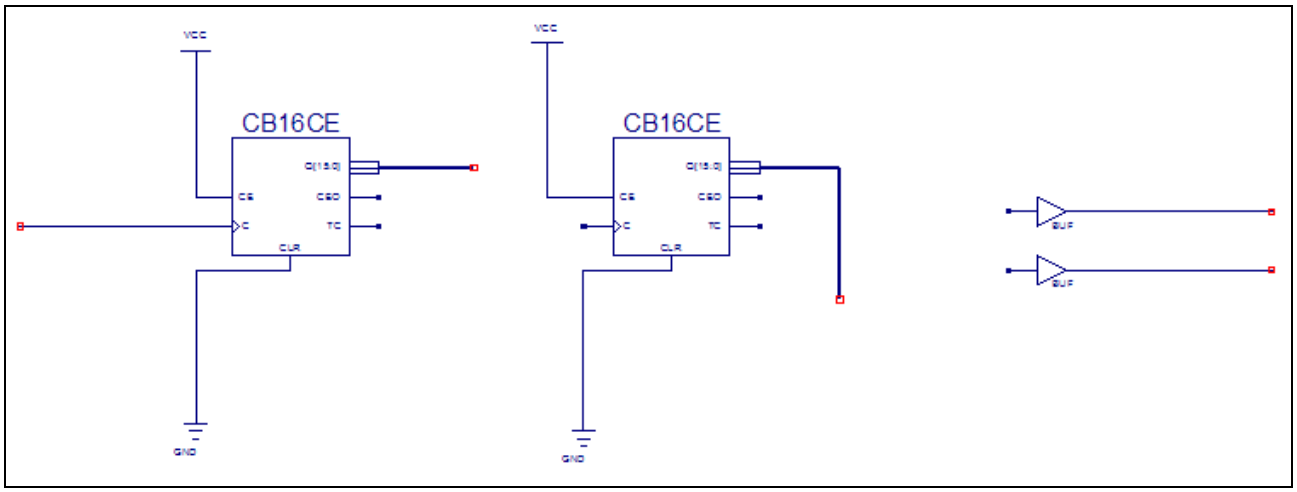
névkeresőt is (*Symbol Name Filter*). Az alkatrészek **forgatását** az *Orientation* mezőben állíthatjuk be. (A panel órajele 50MHz - két 16 bites osztóval megvalósíthatjuk mind a pergésmentesítéshez, mind az időzítéshez szükséges belső órajelet. (A teljes osztási lehetőség kihasználásával a legalsó bit $f = \frac{50MHz}{2^{32}} = 11{,}6mHz$ frekvenciával billeg.)



8. ábra

Az alkatrészek kiválasztása

4. **Kezdjük el szerkeszteni** a következő hálózatot (alkatrész elhelyezés, huzalozás)!



9. ábra

Az alkatrészek elhelyezése a szerkesztő felületen

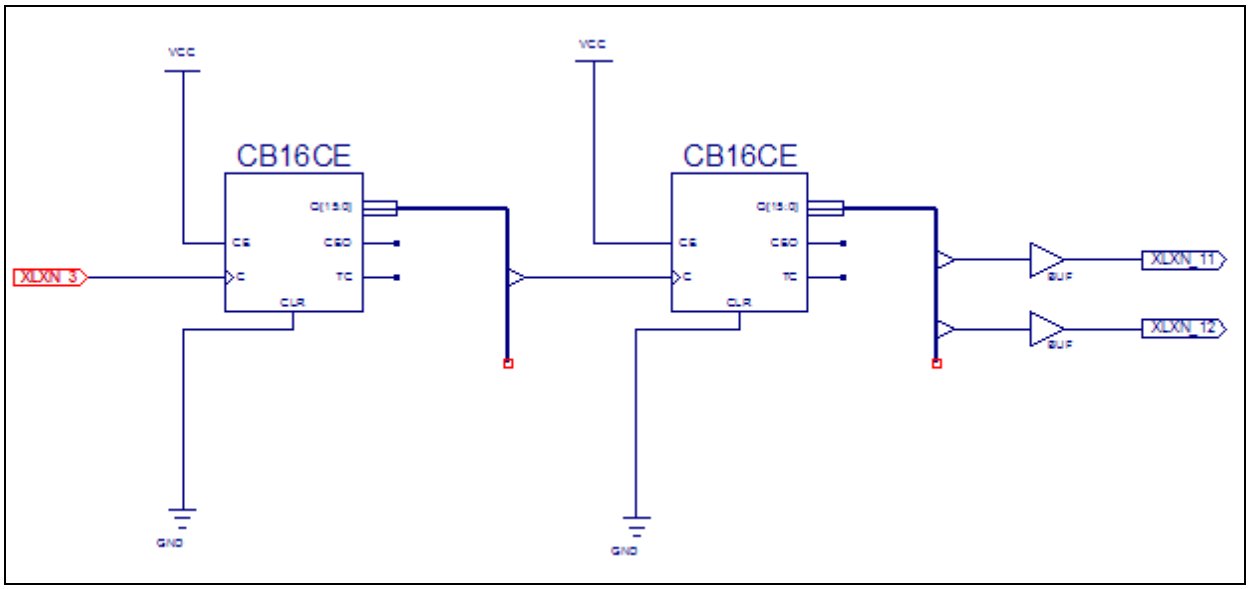
Alkatrészek:

- CB16CE: Engedélyezhető 16 bites számláló (2db, frekvenciaosztáshoz)
- BUF: Nem invertáló puffer
- VCC: Pozitív tápfeszültség (magas szint (H))
- GND: Földpont (alacsony szint (L))

A hozzávezetéseket, illetve az **alkatrész összeköttetéseket** az *Add Wire* ikon ( )segítségével készíthetjük el. Jól látható, hogy a számláló kimenetéhez csatlakoztatott vezeték vastagabb a többinél, ez jelzi annak **busz** jellegét.

5. **Végezzük el a hálózat teljes bekötését** (be- és kimenetek (I/O Marker) hozzáadása, csatlakozás a buszra)!

   Tegyünk az első számláló kimenetére egy, a második számláló kimenetére két buszleágazást (*Add Bus Tap*)[4]! Helyezzük el az I/O Markereket!



10. ábra

Buszcsatlakozók és I/O markerek elhelyezése

A felhasznált parancsok illetve parancsikonok:

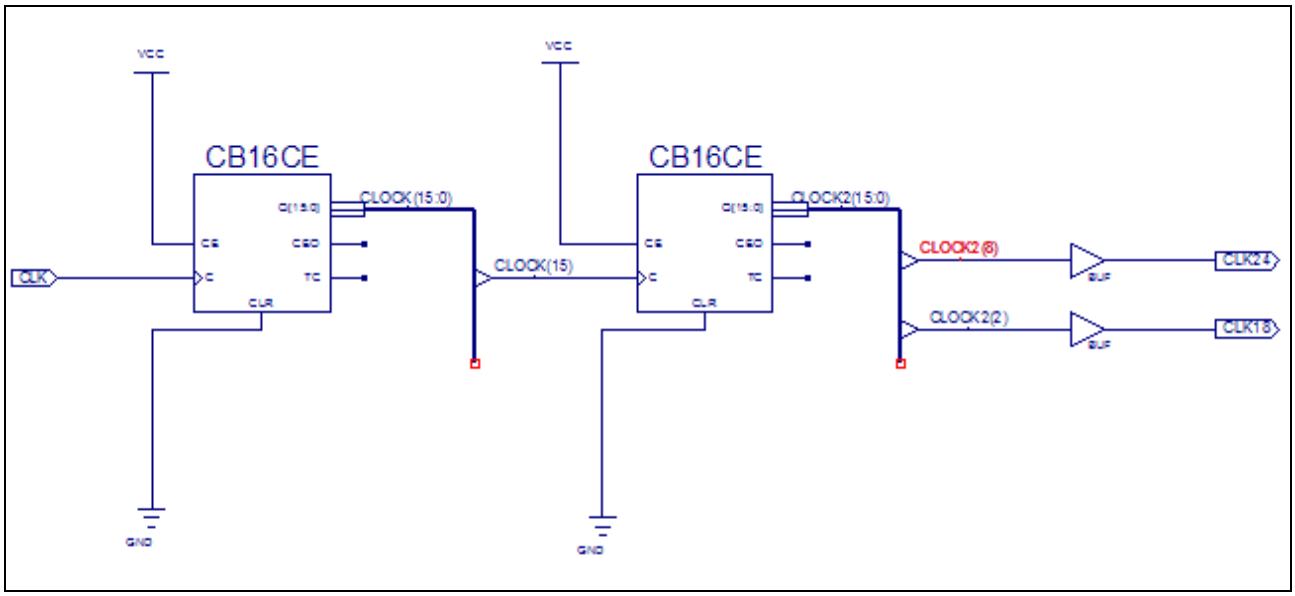
- Select ( ): Kiválasztás/mozgatás.

[4] Az *Options* ablakban beállítható, hogy huzalozáskor a buszra való bekötés automatikusan helyezzen el egy buszcsatlakozót (Tap) is. (szerkesztői megjegyzés)

- 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 beállítási lehetőségeket, illetve műveleteket.)

6. **Nevezzük el** a különböző busz-elnevezéseket és az 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. Az esztétikus kivitelezésre törekedve ügyeljünk, hogy a vezeték elég hosszú legyen a felirat elhelyezéséhez!).[5]
   - **I/O marker elnevezése**: dupla kattintás az I/O markeren, a felugró ablakban *Name* mező átírása.



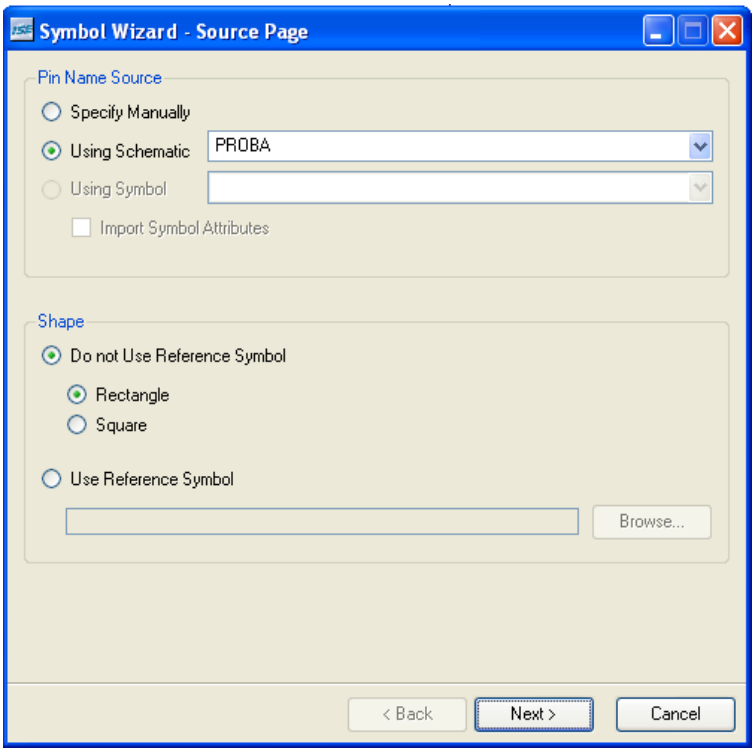
11. ábra
Az I/O markerek, vezetékek, buszok elnevezése

A buszok, illetve leágazásaik elnevezése utal arra, hogy adott buszon belül melyik

[5] Az Options ablakban beállítható automatikus sorszámozás, amely hasznos, ha több, egy buszhoz tartozó, egymás után következő címkét kell elhelyeznünk. (szerkesztői megjegyzés)

vonalra (bitre) csatlakozunk (Pl. a fenti ábrán CLOCK(15:0) egy 16 bites busz, amelynek legfelső helyértékű bitjére csatlakozunk (CLOCK(15)). A markerek elnevezése tetszőleges[6] (a fenti ábrán: CLK: clock-órajel, CLK24: az órajel frekvenciájának 24-ed része, CLK18: az órajel frekvenciának 18-ad része). A CLOCK18 jelét a későbbiekben használhatja a kapcsolók, vagy nyomógombok pergésmentesítésére.

7. A fejlesztőkörnyezet lehetőséget nyújt **saját alkatrész definiálására** az átláthatóság megkönnyítése érdekében – használjuk ki ezt a lehetőséget! A *Tools* menüben válasszuk ki a *Symbol Wizardot*!



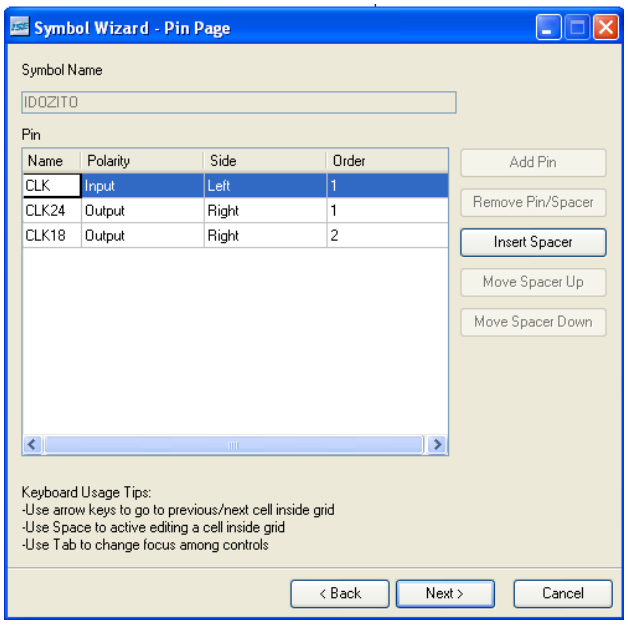
12. ábra
Saját alkatrész definiálása

Fontos, hogy a *Using Schematic*, illetve a *Rectangle* opció legyen bejelölve. A felhasznált kapcsolási rajz (vagy egyéb modul) nevének helyes beállítására ügyeljünk! (Azt a modult kell beállítani, amelyikből szimbólumot szeretnénk készíteni!)

[6] A markerek elnevezésével célszerű utalni a jelek funkciójára. (szerkesztői megjegyzés)

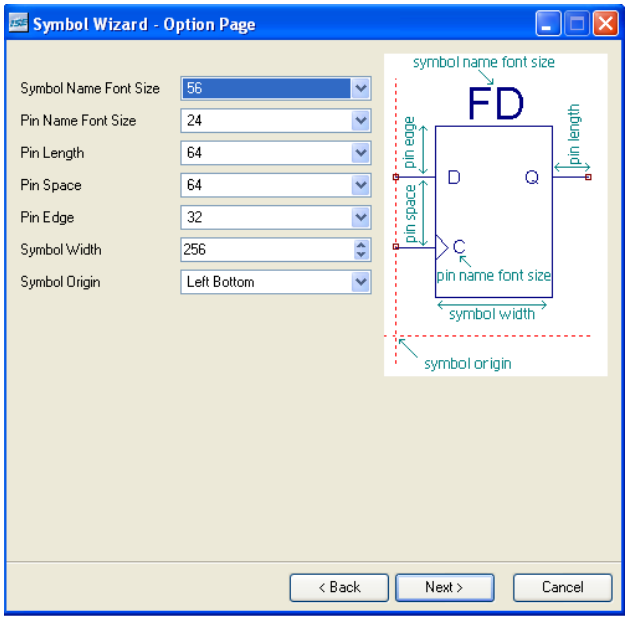
A *Next*-re kattintva jelentkező következő ablakban **az alkatrész be és kimenetei** vannak megjelenítve, ezen nem szükséges változtatnunk. Megadhatjuk azt, hogy a szimbólum jobb, vagy baloldalán legyen-e az adott kivezetés (*Side*), illetve a megjelenítés sorrendjét is beállíthatjuk (*Order*).

Ismét továbblépve (*Next*) az alkatrészünk **geometriai tulajdonságait** módosíthatjuk (átméretezhetjük). Az utolsó ablakban megtekinthetjük alkatrészünk **előnézetét**.



13. ábra

Saját alkatrész kivezetéseinek beállításai

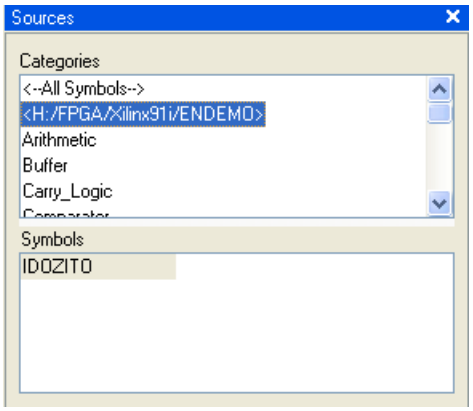


14. ábra

Saját alkatrész méreteinek beállítása

8. **Váltsunk** a **legfelső szintű** modulunk fülére (itt: proba.sch)!

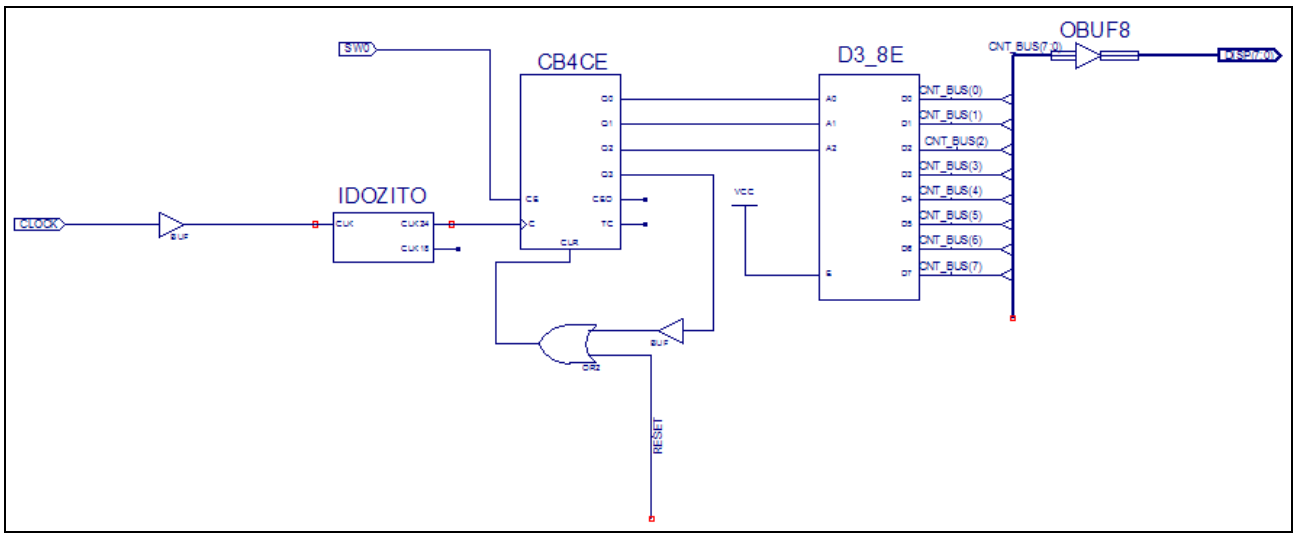
Amennyiben mindent jól csináltunk, saját definiálású alkatrészünknek látszania kell a szimbólumok között:



15. ábra

Saját alkatrész kiválasztása

9. A legfelső szintű modulban **készítsük el a következő áramkört**![7]



16. ábra

A végleges áramkör a szegmensek tesztelésére[8]

[7] Ügyeljünk arra, hogy azt a modult, amelyben a saját alkatrészt definiáltuk, ne írjuk felül, vagy ne töröljük! Ebben az esetben – bár a rajz alapján létrehozott alkatrész lehelyezhető, de – a fordítás során hibajelzést kapunk, hiszen nem tudja a fordító „visszafejteni” a szimbólumot. (szerkesztői megjegyzés)

[8] A CLOCK és SW0 bemenetekre IBUF, a DISP(7:0) kimenetekre pedig OBUF (8 bites OBUF8) elemeket kell elhelyezni, ez a szabályos megoldás. Ettől függetlenül működhet így is az elrendezés (az automatikus buffer-elhelyezés miatt), de a be- és kimenetek nincsenek optimálisan bufferelve. Az IBUF és OBUF elemekkel számos beállítás mellett pl. a logikai családhoz, illetve a logikai szintekhez való illesztés valósítható meg. Számos célra állnak rendelkezésre buffer elemek, ezekről többek között a súgóból tájékozódhatunk. A RESET címke egy megfelelő bufferen keresztül a RESET bemenethez csatlakoztatható. (szerkesztői megjegyzés)

A felhasznált elemek:

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

10. **Adjunk** a projektünkhöz egy **Implementation Constraints** fájlt (*Project→New Source*), amelyben a **lábkiosztást** fogjuk definiálni![9]

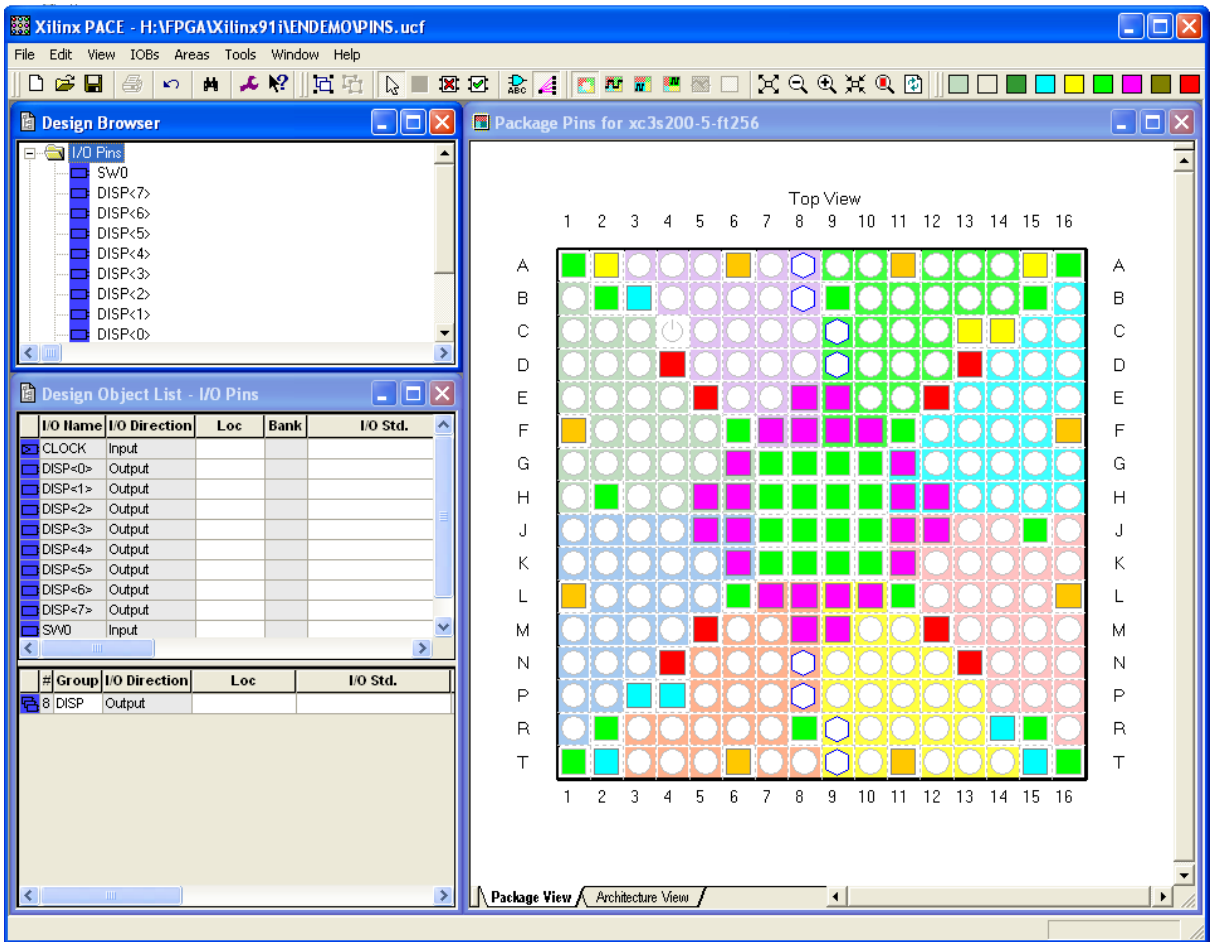
    Kétféle módon tudjuk a lábakat az egyes I/O markerekhez rendelni: **karakteres**, illetve **grafikus** módon.

    Lábkiosztás grafikusan:

    - **Tegyük aktívvá** az *Imlementation Constraits* fájlunkat!
    - **Kattintsunk duplán** a *Create Area Constraints* pontra![10]

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

[9] Ügyeljünk, hogy a fájl létrehozásakor a legfelső szintű modulhoz rendeljük azt. (szerkesztői megjegyzés)
[10] A Create Area Constraints pont helyett az Assign Package Pins menüpont választása helyesebb, mert míg az előbbi a belső elrendezés kialakítását is befolyásolja, utóbbi a lábkiosztás meghatározására való. (A két felhasználói felület azonos.) (szerkesztői megjegyzés)



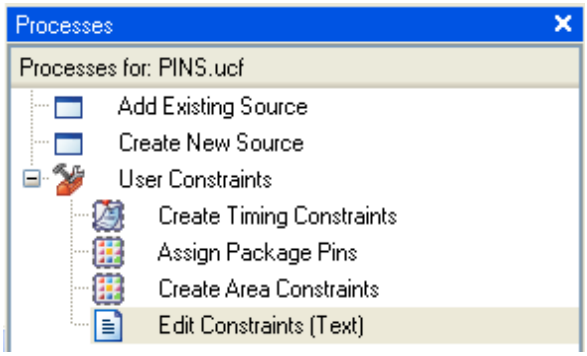
17. ábra

Lábkiosztás grafikus felületen

A különböző **jelölések értelmezéséhez** bekapcsolhatjuk a *View→Toolbars→Legend* menüpontot. Ezt a funkciót a *CTRL+L* billentyűkombinációval is aktiválhatjuk.

A Design Browser ablakból **„húzd és vidd" módszerrel** vigyük át a jobb oldalon található ablak megfelelő helyére minden I/O markert! Miután az összes markert elhelyeztük, **mentsük el** a fájlt, majd **zárjuk be**!

A lábkiosztásra lehetőséget nyújt egy **szövegszerkesztő** is, melyet az *Edit Constraints*-re kettőt kattintva hívhatunk elő.



18. ábra

Lábkiosztás szöveges módszerrel

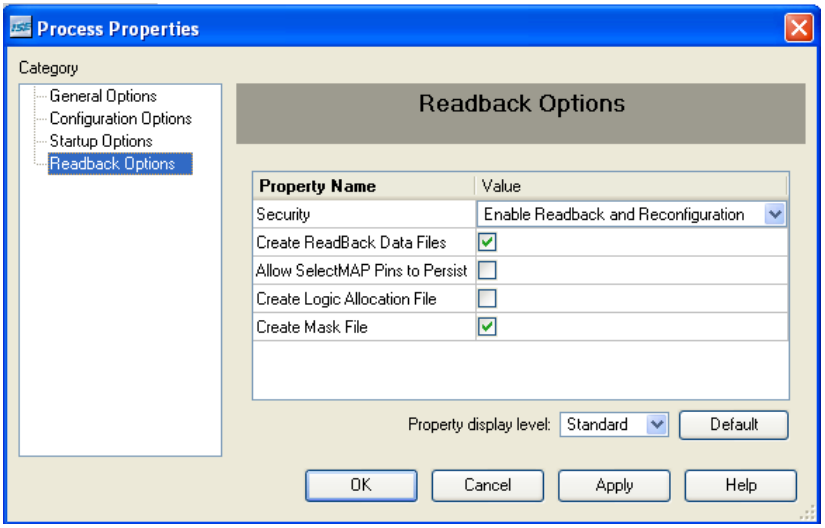
```
1  #PACE: Start of Constraints generated by PACE
2
3  #PACE: Start of PACE I/O Pin Assignments
4  NET "CLOCK"  LOC = "T9"  ;
5  NET "DISP<0>"  LOC = "E14"  ;
6  NET "DISP<1>"  LOC = "G13"  ;
7  NET "DISP<2>"  LOC = "N15"  ;
8  NET "DISP<3>"  LOC = "P15"  ;
9  NET "DISP<4>"  LOC = "R16"  ;
10 NET "DISP<5>"  LOC = "F13"  ;
11 NET "DISP<6>"  LOC = "N16"  ;
12 NET "DISP<7>"  LOC = "P16"  ;
13 NET "SWO"  LOC = "F12"  ;
14
15 #PACE: Start of PACE Area Constraints
16
17 #PACE: Start of PACE Prohibit Constraints
18
19 #PACE: End of Constraints generated by PACE
20
```

19. ábra

A lábkiosztás szöveges formája

A #-tel kezdődő sorokat a fordító **nem veszi figyelembe**, így lehetőségünk van univerzális lábkiosztást írni, melyben csak az adott programban használt perifériákat használjuk, a többit #-tel megjelöljük.

11. **Hozzuk létre a konfigurációs bitmintát**! Kattintsunk jobb egérgombbal a *Generate Programming File*-ra, majd a *Properties*-re:

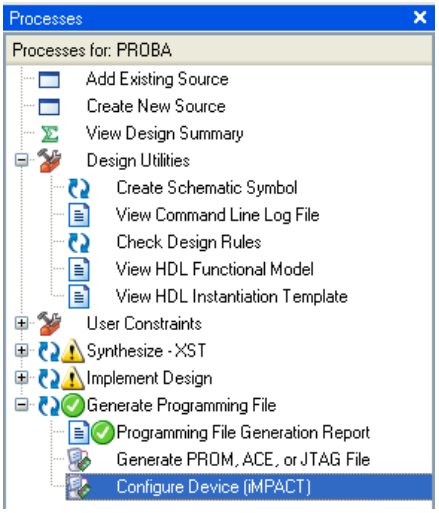


20. ábra

A programozó fájl elkészítésének beállításai

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!

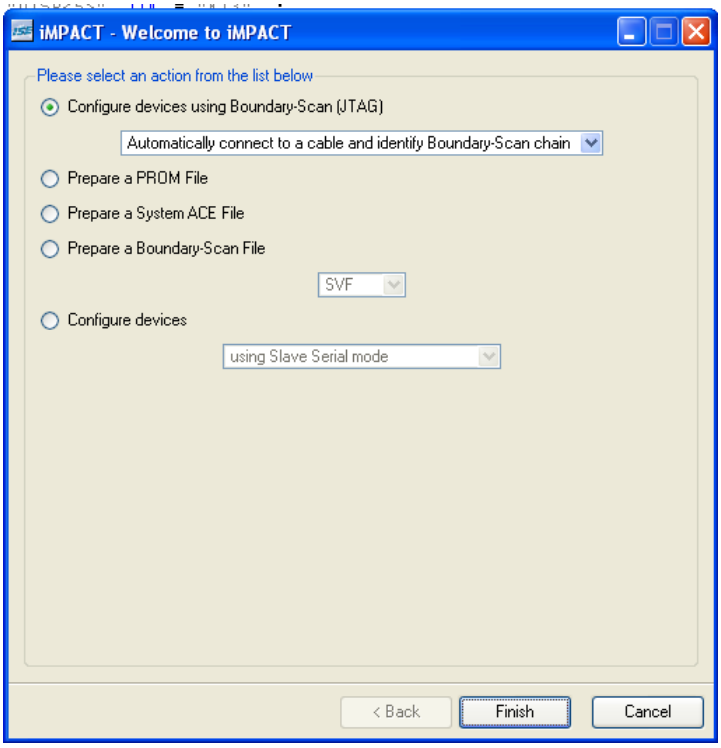


21. ábra

Az iMPACT elindítása

A *Configure devices using Boundary-Scan (JTAG)* (Eszközök konfigurálása

peremfigyelés segítségével) **legyen kijelölve**, majd kattintsunk a **befejezésre** (*Finish*)!



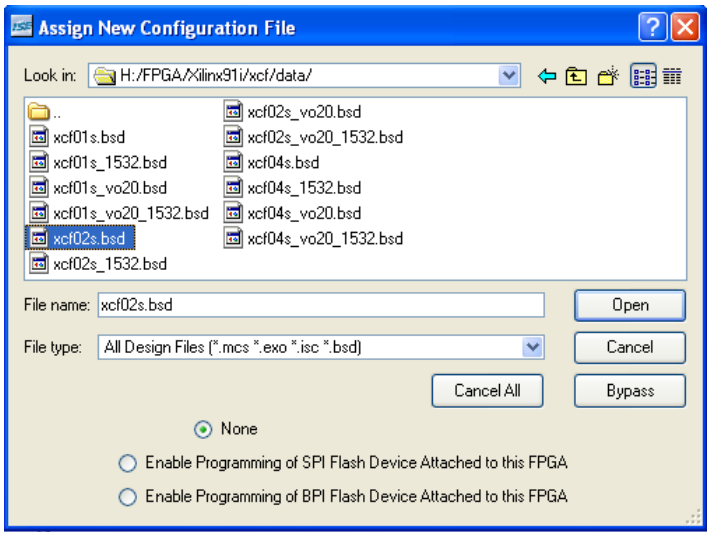
22. ábra

Csatlakoztatás a peremfigyelés módszerével

Ezután **automatikusan** megnyílik a **következő ablak.** (Ha mégsem, akkor jobb egérgombbal kattintsunk az IC-n, majd válasszuk az *Assign New Configuration File*-t!) **Válasszuk ki** a betöltendő fájlunkat (*.bit* kiterjesztéssel), majd nyissuk meg (*Open*)!

A következő ablakban egy, az FPGA-hoz tartozó külső konfiguráló **FLASH memóriához** (IC9, XCF02S) kell **rendelni az azt leíró gyári fájl**t (xcf02s.bsd).[11]

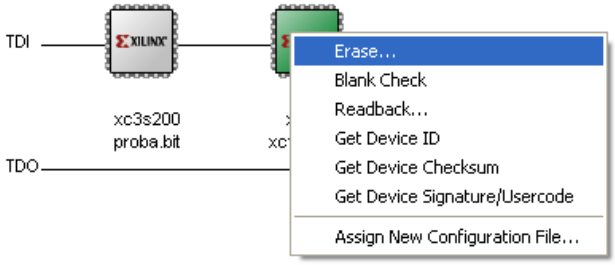
[11] Bár példánkban nem a külső FLASH memóriába történik a konfigurációs bitminta betöltése, de a letöltés ellenőrzéséhez szükség van erre a lépésre, és a következő XCF02S-sel kapcsolatos lépésekre is. Másik (javasolt) megoldás a betöltési ablakban (ld. 23. ábra) a Bypass választása, ilyenkor a nem használt JTAG elem lényegében nem kerül a peremfigyeléses láncba. Így nem kell hozzá fájlt kiválasztani, törölni sem. (szerkesztői megjegyzés)



23. ábra

Az XCF02S memória gyári leíró fájljának kiválasztása

A jobb oldali „IC”-n jobb egérgombbal kattintva **töröljük** az XCF02S FLASH memóriát (*Erase*)!



24. ábra

A konfigurációs FLASH memória törlése

A bal oldali „IC”-n kattintva **töltsük be** a létrehozott konfigurációs bitmintánkat az FPGA-ba (*Program*)! (Ha változtatunk valamelyik modulon, a bitminta generálást mindig meg kell ismételni!)

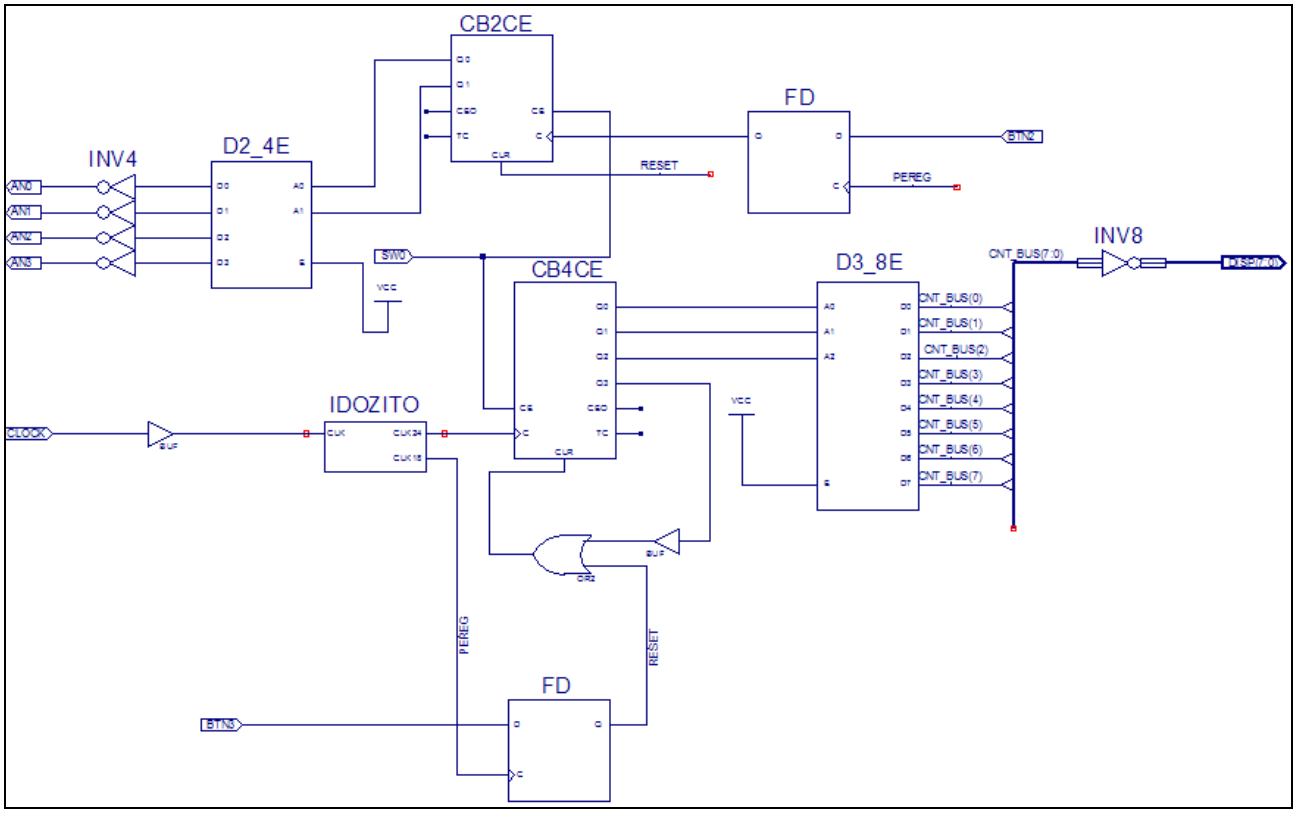
25. ábra

Program letöltése az FPGA-ba

3. Feladatok a kapcsolási rajz alapú tervezéshez

1. **Figyelje meg** a működést, javítsa a **hibát**!
2. **Alakítsa át** az elrendezést úgy, hogy a BTN3 nyomógomb hatására a kapcsolás kerüljön alaphelyzetbe (a BTN3 nyomógombot pergésmentesítse)!
3. Alaphelyzetben minden kijelző be van kapcsolva. Bővítse a kapcsolást, hogy egyszerre mindig csak egy kijelző legyen bekapcsolva. A kiválasztást a BTN2 nyomógombbal lehessen megtenni!

Segítség a feladat megoldásához:



26. ábra

Az első feladat egy lehetséges megoldása[12]

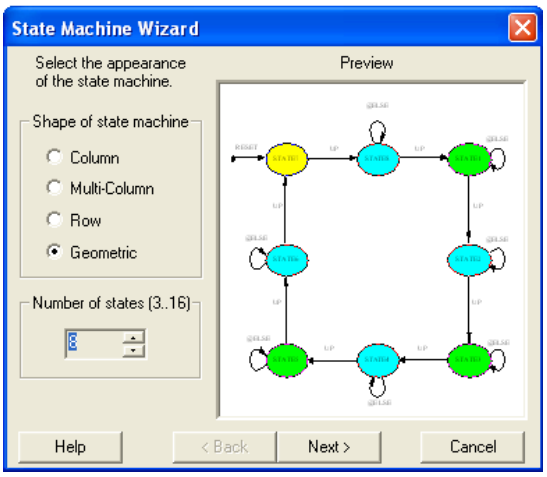
[12] A kimeneteken kimeneti bufferek (OBUF), a bemeneteken pedig bemeneti bufferek (IBUF) elhelyezése szükséges! (szerkesztői megjegyzés)

4. Mintafeladat kiegészítése állapotdiagram modellel

A 2. fejezetben definiált feladatból a LED-ek egymás után történő bekapcsolását **állapotdiagramos tervezéssel**, a StateCad program segítségével oldjuk meg. (A részfeladat külön funkcióként való megvalósításához nyithat új projektet, de itt – a feladatkiírás szerint – az előző tervben implementáljuk a megoldást.) Az állapotdiagramos tervezésnél **állapotokat definiálunk**, valamint megadjuk az egyes állapotok közötti **átmenet feltételét**.

1. **Hozzon létre** egy *State Diagram* típusú új forrást (*Project→New Source*)! A fejlesztőkörnyezet **automatikusan megnyitja** az állapotdiagram-szerkesztőt.

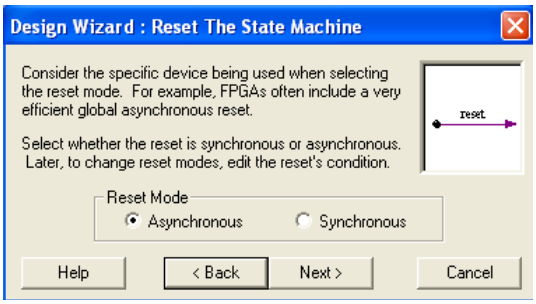
2. A *Draw State Machines* ikonra kattintva **hozza létre** az alábbi geometriai elrendezést!



27. ábra

Az állapotgép geometriai elrendezése

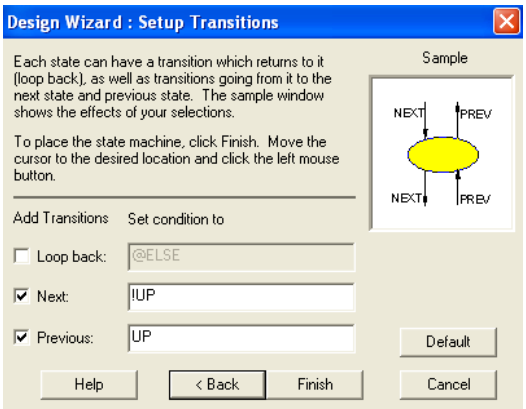
A *Next*-re kattintva az alaphelyzetbe állítás szinkronitását állíthatjuk be. Válasszuk az **aszinkron** módot!



28. ábra

A Reset feltétel szinkronitásának megadása

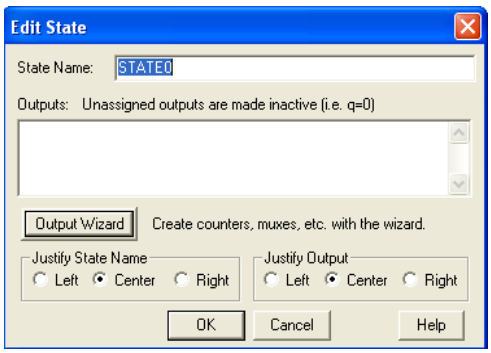
A *Setup Transitions* ablakban **jelöljük be** a *Next*-et és a *Previous*-t, és válasszuk az *!UP*, illetve *UP* funkciót!



29. ábra

Az állapotgép futásának beállítása

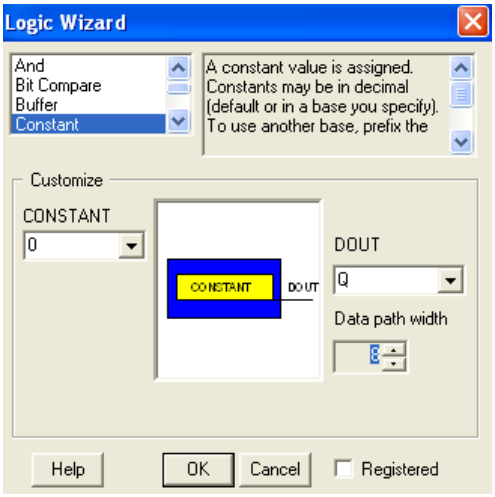
3. Az egyes állapotokon kettőt kattintva **hozzuk elő** az *Output Wizard*-ot!



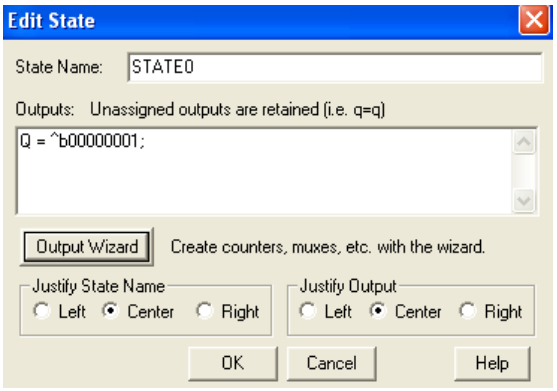
30. ábra

Az egyes állapotok beállítása

Az **állapotokat konstansként** adjuk meg. Az **elnevezés tetszőleges**, de természetesen nem szerepelhet két különböző állapotnál ugyanaz a név. Az **adatszélesség 8 bit**, mivel 8 kimenetet szeretnénk vezérelni. **Írjuk be** binárisan az általunk kívánt értéket minden állapothoz!



31. ábra
A kimenet típusának és szélességének megadása

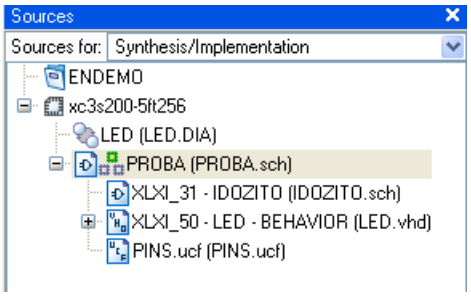


32. ábra
Az állapot kimeneti értékének megadása

4. Ezután kattintsunk az *Optimize* ikonra (itt több ablakon keresztül beállíthatjuk, hogy a fordító milyen szempontok szerint **optimalizálja** az állapotgépet)! **Céleszközként** *FPGA*-t, **célforrásként** *VHDL*-t válasszunk!

5. Kattintsunk a *Generate HDL* ikonra, majd **sikeres lefordítás** után **adjuk hozzá** a projectünkhöz a **.dia*, és a **.vhd* fájlokat a projekt *Add Copy of Source* menüpont segítségével! (Amennyiben nem vagyunk biztosak az állapotgép helyes működésében, **szimulálhatjuk** a *StateBench* segítségével.)

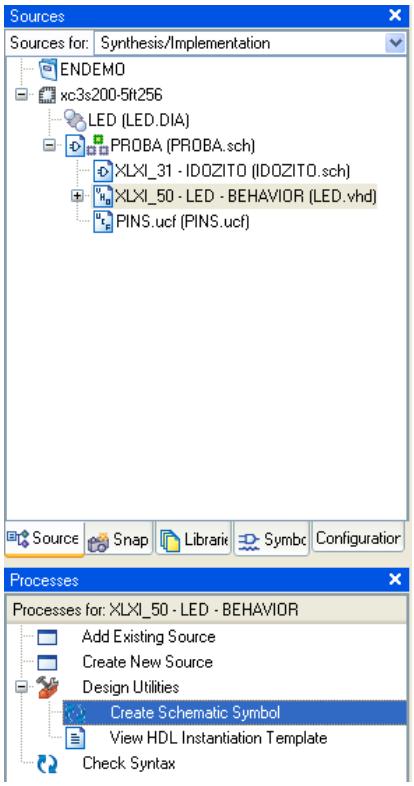
6. Ha minden lépést helyesen csináltunk végig, **megjelenik** a két fájl a forrásaink között.



33. ábra

StateCad fájl, és annak VHDL forrása a projektben

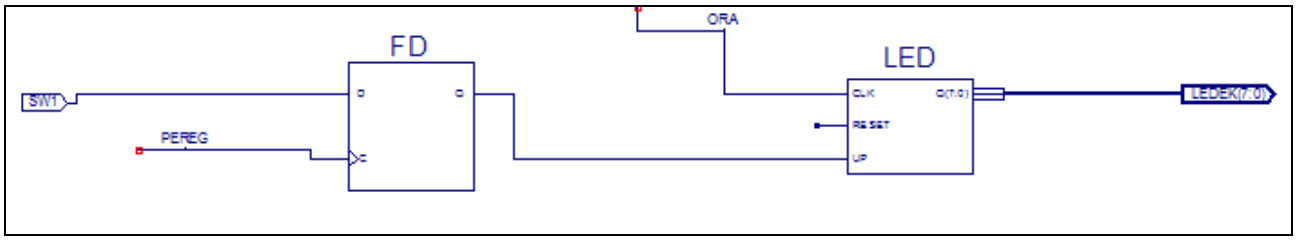
7. **Hozzunk létre saját alkatrészt** (saját kapcsolásrajzi szimbólumot) a **.vhd* fájlunkhoz!



34. ábra

Szimbólum rendelése a VHDL forráshoz

8. **Valósítsuk meg** a következő áramkörrészletet! (PEREG és ORA az előzőekben létrehozott órajel osztó CLK18 és CLK24 kimenetei)



35. ábra

A kiegészítő áramkör rajza[13]

[13] A be- és kimeneteken IBUF és OBUF elemek elhelyezése szükséges. (szerkesztői megjegyzés)

5. Feladatok az állapotdiagram alapú tervezéshez

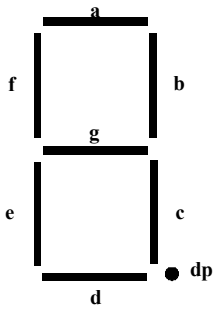
1. **Próbálja ki** SW1 kapcsoló hatását!
2. **Rajzolja át** a kapcsolást, illetve az állapotdiagramot úgy, hogy a fény oda-vissza fusson, és az irányt az SW2 kapcsolóval lehessen beállítani!
3. **Hozzon létre** egy olyan logikát, amely oda-vissza futtat egy fénypontot a LED soron, de azok a LED-ek nem világítanak, amelyek hozzárendelt SW kapcsolója aktív! (pl.: SW3 és SW4 aktív, akkor LD3 és LD4 ne világítson.)
4. Az előző **két programot egyesítse** úgy, hogy a BTN2 nyomógombbal lehessen közöttük!
5. Valósítson meg **Knight-Rider futófényt** a LED-eken!*

6. VHDL modul fejlesztése

Az eddig megszerzett ismereteinket bővítsük **VHDL programozással**! Írjunk egy egyszerű VHDL programot, mely a bemenetére érkező 4 bites bináris számból előállítja annak 7-szegmenses kódját!

7 szegmenses kódtáblázat közös anódos kijelzőre:

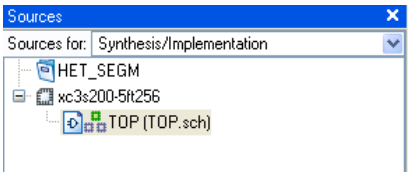
Hexadecimális jegy	*Bináris kód*	*7 szegmenses kód* Dp g f e d c b a
0	0000	1 1 0 0 0 0 0 0
1	0001	1 1 1 1 1 0 0 1
2	0010	1 0 1 0 0 1 0 0
3	0011	1 0 1 1 0 0 0 0
4	0100	1 0 0 1 1 0 0 1
5	0101	1 0 0 1 0 0 1 0
6	0110	1 0 0 0 0 0 1 0
7	0111	1 1 1 1 1 0 0 0
8	1000	1 0 0 0 0 0 0 0
9	1001	1 0 0 1 0 0 0 0
A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



36. ábra

A 7 szegmenses kijelző szegmenseinek elnevezése

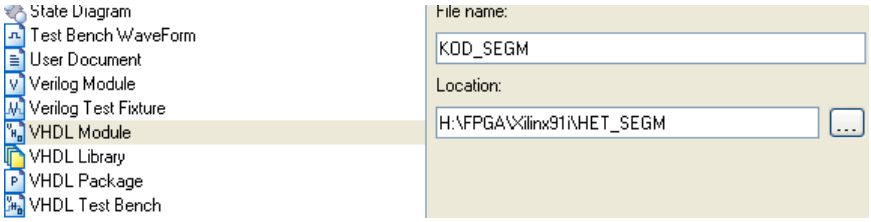
1. **Hozzunk létre** egy **új projektet**, az **elsődleges forrás** legyen ismét **schematic**!



37. ábra

A projekt és a legfelső szintű forrás létrehozása

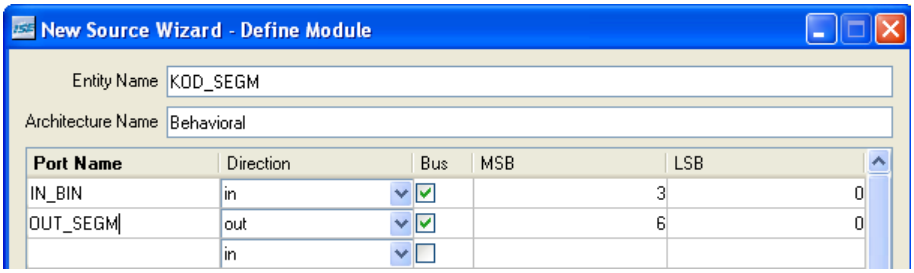
2. **Új forrásként** adjunk a projektünkhöz egy VHDL típusú fájlt!



38. ábra

A VHDL modul létrehozása

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



39. ábra

A VHDL modul inicializálása

Felhasznált be- és kimenetek:

- Bemenet (IN_BIN): 4 bites bináris kód
- Kimenet (OUT_SEGM): 7 szegmenses kód (7 bites, mivel példánkban a tizedespontot nem használjuk)

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

```
----------------------------------------------------------------------------------
-- 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;
```

40. ábra

A fejlesztőkörnyezet által generált VHDL kódváz

4. **Írjuk be** a 7 szegmenses dekódolót megvalósító **kódrészletet a programvázba**! (A VHDL kódot a „*begin*” és az „*end Behavioral;*” közé írjuk be. Amennyiben „--” (két mínusz jel) kerül a kódba, a fordító nem veszi figyelembe a sor végéig álló szöveget (megjegyzések, jegyzetek lehetősége).[14])

[14] VHDL példákat és leírást az előadáson, illetve a kiadott segédletekben találhat. (szerkesztői megjegyzés)

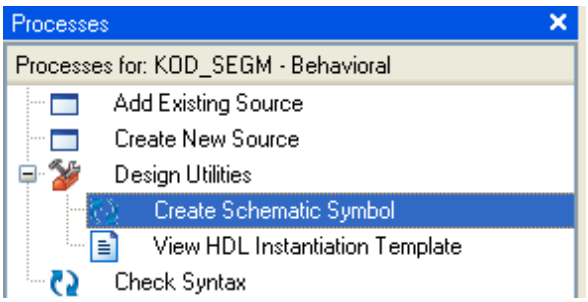
```
begin

   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;
```

41. ábra

Az általunk írt VHDL kód

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

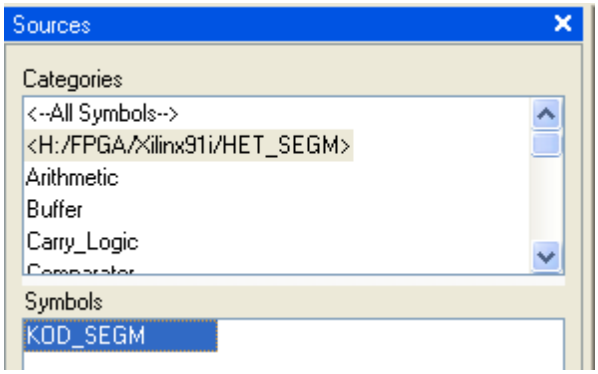
5. **Készítsünk kapcsolási rajz szimbólumot** VHDL forrásunkból!



42. ábra

Kapcsolási rajz szimbólum létrehozása VHDL forrásból

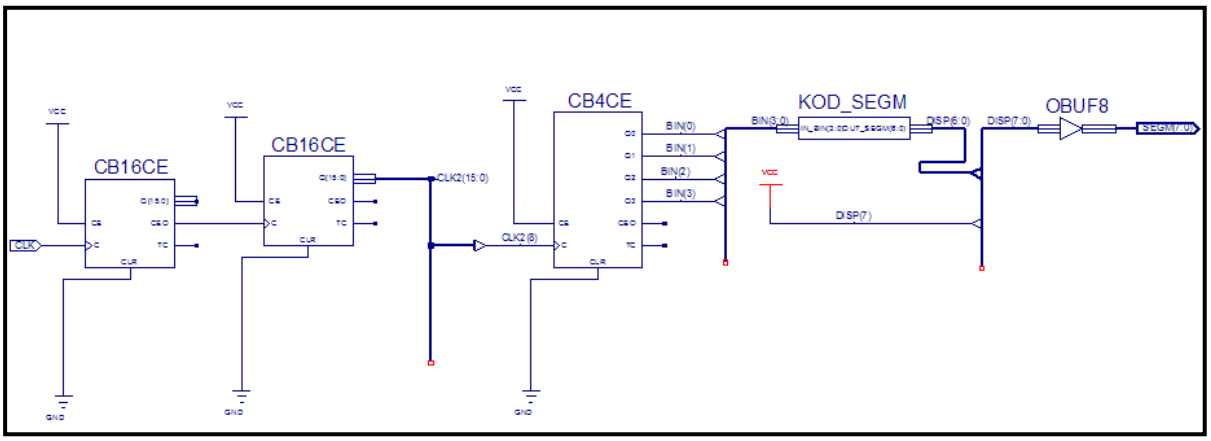
Ellenőrizzük, hogy alkatrészünk **megjelent-e a szimbólumok között**!



43. ábra

Elkészített szimbólumunk a projekt szimbólumai között

6. **Hozzuk létre** a feladatnak megfelelő logikai elrendezést a TOP modulban! A program egy **lehetséges megvalósítása** az alábbi ábrán látható.



44. ábra

A feladat egy lehetséges megoldása[15]

Az első két blokkot helyettesíthetjük az első feladat során megalkotott órajel osztó áramkörrel. Az eredeti feladatkiírás teljesítéséhez rögtön az első projektben is implementálhatjuk VHDL modulunkat a gyorsabb fejlesztés érdekében. (Így nincs szükség a kész kapcsolás átmásolására.)

[15] A CLK bemeneten OBUF elem elhelyezése szükséges. (szerkesztői megjegyzés)

7. Feladatok a VHDL alapú tervezéshez

1. **Módosítsa a logikát,** hogy egy kapcsolóval (SW0) lehessen kiválasztani, hogy **felfelé, vagy lefelé számlálás** történjen!
2. **Módosítsa a logikát**, hogy egy másik kapcsolóval (SW1) át lehessen állítani a számlálást **decimálisra**!
3. **Készítsen** 2 vagy **több digitre számlálót**!
4. **Valósítsa meg** saját ötletét, vagy a mérésvezető által kiadott feladatot! (Nyomógomb lenyomások számlálása, aritmetikai egység, stb.)

1. sz. melléklet – Az általunk használt gyakorlópanel lábkiosztása

Periféria	*Pozíció*	*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	0. digit anódja
AN1	G14	1. digit anódja
AN2	F14	2. digit anódja
AN3	E13	3. digit anódja
LED0	K12	
LED1	P14	
LED2	L12	
LED3	N14	
LED4	P13	
LED5	N12	
LED6	P12	
LED7	P11	
CLOCK	T9	Órajel

2. sz. melléklet – Néhány egyszerű VHDL példa[16]

1. NAND kapcsolat kétféle formában

```
library ieee;
use ieee.std_logic_1164.all;

----------------------------------------

entity NAND_gate is
port( A: in std_logic;
      B: in std_logic;
      Y: out std_logic);
end NAND_gate;

----------------------------------------

architecture NAND_leiras of NAND_gate is
begin

      process(A, B)
      begin
            if (A='1' and B='1') then
            Y <= '0';
      else
            Y <= '1';
      end if;
      end process;

end NAND_leiras;
```

```
library ieee;
use ieee.std_logic_1164.all;

----------------------------------------

entity NAND_gate is
port( A: in std_logic;
      B: in std_logic;
      Y: out std_logic);
end NAND_gate;

----------------------------------------

architecture NAND_leiras of NAND_gate is
begin

      Y <= A nand B;

end NAND_leiras;
```

[16] E melléklet csak érintőleges betekintést ad a VHDL programozásba. Az Interneten számos példagyűjtemény és részletes leírás fellelhető. (szerkesztői megjegyzés)

2. 2-ről 4-re dekódoló kétféle formában

```
library ieee;
use ieee.std_logic_1164.all;

-----------------------------------------------

entity DECODER is
port( I:    in std_logic_vector(1 downto 0);
      O:    out std_logic_vector(3 downto 0));
end DECODER;

-----------------------------------------------

architecture leiras of DECODER is
begin

      process (I)
      begin

      case I is
            when "00" => O <= "0001";
            when "01" => O <= "0010";
            when "10" => O <= "0100";
            when "11" => O <= "1000";
            when others => O <= "XXXX";
      end case;

      end process;

end leiras;
```

```
library ieee;
use ieee.std_logic_1164.all;

-----------------------------------------------

entity DECODER is
port( I:    in std_logic_vector(1 downto 0);
      O:    out std_logic_vector(3 downto 0));
end DECODER;

-----------------------------------------------

architecture leiras of DECODER is
begin

      O <=  "0001" when I = "00" else
            "0010" when I = "01" else
            "0100" when I = "10" else
            "1000" when I = "11" else
            "XXXX";

end leiras;
```

3. D flip-flop

```
library ieee;
use ieee.std_logic_1164.all;

---------------------------------------------

entity dff is
port( D:    in std_logic;
      CK:   in std_logic;
      Q:    out std_logic);
end dff;

---------------------------------------------

architecture leiras of dff is
begin

      process(D, CK)
      begin

      if (CK='1' and CK'event) then
            Q <= D;
      end if;

      end process;

end leiras;
```

4. 4 bites shift-regiszter

```
library ieee;
use ieee.std_logic_1164.all;

----------------------------------------------------

entity shift_reg is
port( I:    in std_logic;
      CK:   in std_logic;
      Q:    out std_logic_vector(3 downto 0));
end shift_reg;

----------------------------------------------------

architecture shift_leiras of shift_reg is

      signal S: std_logic_vector(3 downto 0):="1111";

begin

      process(I, CK, S)
      begin

      if CK'event and CK='1' then
            S <= I & S(3 downto 1);
      end if;

      end process;

      Q <= S;

end shift_leiras;
```

5. 8 bites számláló

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

---------------------------------------------------

entity szamlalo is

port( CK:   in std_logic;
      CL:   in std_logic;
      Q:    out std_logic_vector(7 downto 0));
end counter;

---------------------------------------------------

architecture szlo_leiras of szamlalo is

      signal Pre_Q: std_logic_vector(7 downto 0);

begin

      process(CK, CL)
      begin
      if CL = '1' then
            Pre_Q <= Pre_Q - Pre_Q;
      elsif (CK='1' and CK'event) then
            Pre_Q <= Pre_Q + 1;
      end if;
      end process;

      Q <= Pre_Q;

end szlo_leiras;
```