FPGA alapú áramkörök fejlesztése a Xilinx ISE Webpack használatával

Tartalom

1 Bevezetés

A könyv bevezetést kíván nyújtani a digitális áramkörök tervezésébe, célja a tervezési lehetőségek minél szélesebb körben való ismertetése. Manapság több módja is van az áramkörtervezésnek. A következőkben az FPGA alapú digitális áramkörtervezéshez szükséges ismeretekkel és eszközökkel ismerkedhetünk meg. Ismertetésre kerül a XILINX cég SPARTAN 3 FPGA családja és a hozzá kiadott fejlesztői környezet. Valamint egy Basys 2 nevű FPGA fejlesztői kártya, aminek segítségével a megszerzett tudás kipróbálására is lehetőség nyílik. A tananyag végén egy összetettebb projekt keretén belül, négybites CPU tervezésére és szimulációjára kerül sor.

2 A mérés során használt eszközök

A következő részben az FPGA alapú tervezés alapjainak elsajátításához szükséges eszközök és szoftverek kerülnek ismertetésre. Természetesen a felsoroltak csak egy lehetőség, sok gyártó sokféle FPGA családot forgalmaz a különböző feladatok megoldására, valamint a tervezőszoftverekből is igen sokféle létezik.

2.1 Spartan 3 FPGA család

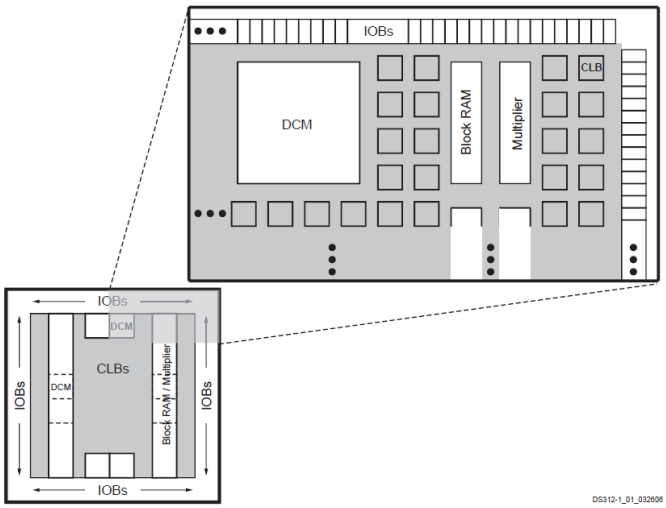
A Xilinx cég a 90 nm-es technológiával készült Spartan-3 FPGA családot olcsó, közepes bonyolultságú digitális áramkörök megvalósítására tervezte. A felhasználó a családon belül rengeteg különböző méretű, tulajdonságú és tokozású eszköz közül választhat igényeinek és az adott alkalmazásnak megfelelően.

A Spartan-3 család öt alapvető programozható funkcionális elemből épül fel, melyek FPGA-n belüli elhelyezkedését az 1. ábra szemlélteti:

- **Konfigurálható logikai blokkok (CLB-k):** Logikai függvények és tároló funkció megvalósítására alkalmas elem.
- **Input/Output blokkok (IOB-k):** A be- és kimenetek (a külvilág) valamint a belső logika elemek közötti adatáramlást valósítják meg. Lehetővé teszik a kétirányú és háromállapotú (tri-state) interfészek valamint különböző szabványú és feszültségszintű digitális jelek illesztését.
- **Blokk RAM:** Adattárolásra alkalmas memória elem.
- **Szorzó blokk (Multiplier):** Két 18-bites bináris szám gyors összeszorzására alkalmas egység.
- **Digitális órajel menedzser blokk (DCM):** Az órajelek kezelésére szolgáló programozható egység. Szolgáltatásai: késleltetés, frekvencia szorzás-osztás, fázistolás.

2.1.1 Kapcsolómátrix

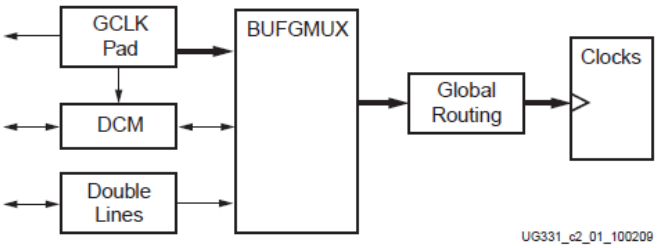
Az FPGA-n belül egy sűrű vezetékhálózat biztosítja az egyes elemek közötti kapcsolatot. A funkcionális blokkok programozható összeköttetéseken (kapcsoló mátrix) keresztül kapcsolódnak a vezetékhálózathoz. Az FPGA programja (konfigurációja) tulajdonképpen a funkcionális blokkok vezérlőjeleit valamint a kapcsolómátrixok állapotát határozza meg, vagyis, hogy a belső vezetékhálózaton keresztül mely egységek kerüljenek egymással összeköttetésbe. A programot az FPGA-n belül statikus konfigurációs memória tárolja, amelyet a tápegység bekapcsolása után a kívánt logikai funkciónak megfelelő adattal fel kell tölteni. A feltöltés történhet külső PROM-ból, külső mikroprocesszoron keresztül, vagy JTAG interfészen keresztül. A konfigurációs adat előállítását, a konfigurációs memóriák feltöltését, valamint a JTAG interfészen történő programozást a gyártó saját Xilinx ISE programcsomagja támogatja.



1. ábra - A Spartan-3 család belső felépítése

2.1.2 Órajel

Az órajelek FPGA-n belüli elosztásáért speciális belső vezetékhálózat felelős, amely a flip-flopok és egyéb órajeles működésű egységek órajel bemeneteihez kapcsolódik. Az órajel hálózathoz speciális bemeneti blokk (GCLK) és meghajtó-multiplexer (BUFGMUX) tartozik. A hálózatra kerülő órajelet a multiplexer választja ki a GCLK bementről vagy a DCM valamelyik kimenetéről (2. ábra).

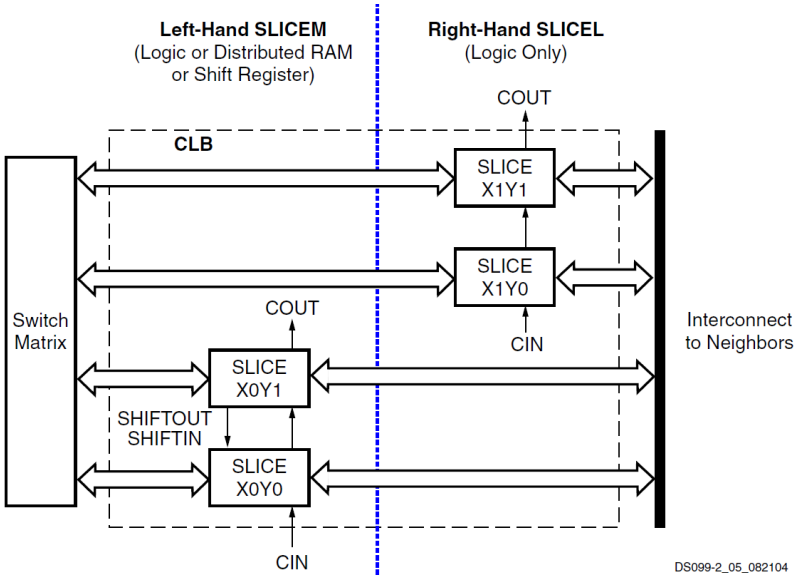


2. ábra - Az órajel hálózat részei

Az FPGA-n belül megengedett, hogy egy flip-flop egy általános felhasználású bemenetről vagy belső vezetékről kapja az órajelét, a nagysebességű szinkron működés biztosítása érdekében azonban ez nem javasolt.

2.1.3 Konfigurálható logikai blokk

A konfigurálható logikai blokkok (CLB-k) az elsődleges építő elemei az FPGA-ban felépített kombinációs vagy szinkron sorrendi hálózatoknak. A CLB-k egymással összeköttetésben lévő szeletekből (SLICE) épülnek fel a 3. ábra szerint.



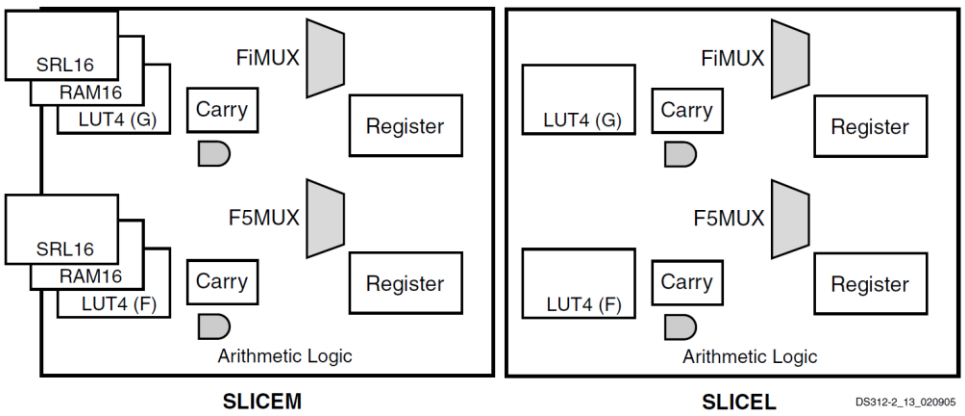
3. ábra - A szeletek elhelyezkedése a CLB-ben

A szeletek, típustól függően logikai és memória funkciót (SLICEM) vagy csak logikai funkciót (SLICEL) láthatnak el (4. ábra). Mindkét szelet tartalmazza az alábbi összetevőket:

- Két 4-bemenetű LUT
- Két tároló elem
- Két multiplexer
- Carry és aritmetikai logika

A SLICEM szeletek további összetevői:

- Két 16x1 bit RAM blokk
- két 16-bites shift-regiszter



4. ábra - A szeletek összetevői

2.1.4 Lookup table (LUT)

A számítógép (sőt számológép) előtti időkben a bonyolultabb függvényeket egy könyv alakban kinyomtatott táblázat segítségével számították ki, ahol a függvény értékei bizonyos felbontásban és pontossággal voltak megadva, és a táblázatban megadott értékek között lineáris interpolációval számoltak (logaritmustáblák, trigonometrikus függvények, stb.). A megfelelő értékek kikeresése sokkal gyorsabb volt, mint a függvény valamilyen képlettel történő közelítő kiszámítása. Természetesen, valakiknek előzőleg ki kellett számolni a táblázat értékeit, sokszor sok-sok éves munkával.

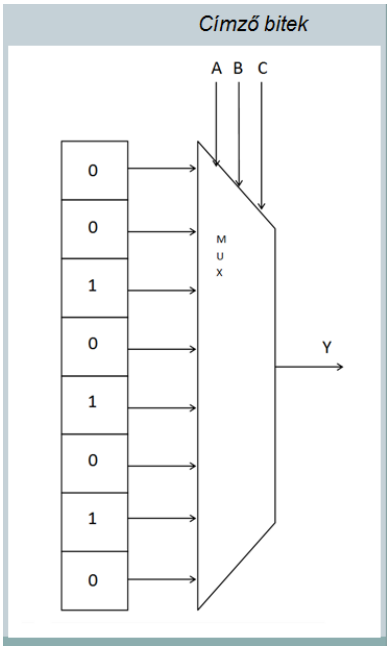
A számítástechnikában a használata azon alapul, hogy egy előre feltöltött tárolóból rendszerint sokkal gyorsabban lehet az adatokat kiolvasni, mint azokat valamilyen algoritmussal kiszámolni. A fixpontos és lebegőpontos aritmetikákban is használnak LUT-ot a kezdeti pontosságú érték előállítására, a végső pontosságot már számítással érik el. Az FPGA CLB blokkjában a LUT az adott logikai függvény értékeit tartalmazza, amit a fordítóprogram a logikai rajz alapján állít elő és tárol el benne. 4 bemeneti változó esetén az összes lehetséges logikai függvény száma 64K.

A következő példa az alábbi háromváltozós logikai függvény Look-Up-Table lel történő megvalósítását láthatjuk.

$Y(A,B,C) = \overline{A}\,B\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,B\,\overline{C}$

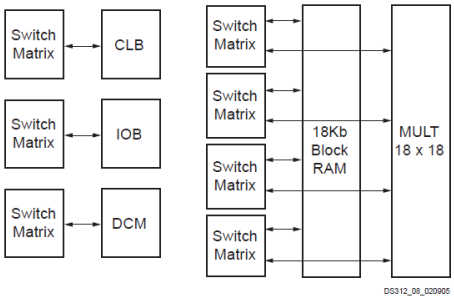
A függvény igazságtáklája:

	A	B	C	Y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

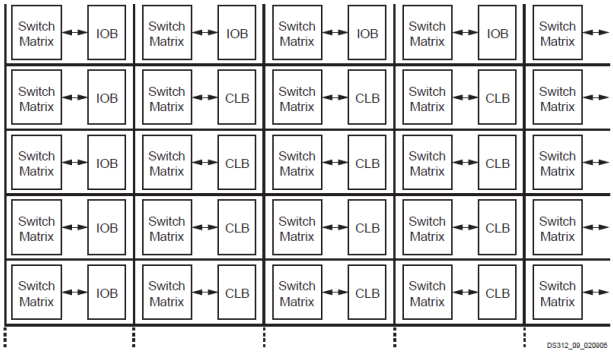


2.1.5 Erőforrások

Az FPGA erőforrásai (CLB, IOB, DCM, RAM és a szorzó) közötti kapcsolatot a kapcsoló mátrixok biztosítják (5. ábra). A kapcsoló mátrixok a belső vezetékhálózatra kapcsolódnak, ami horizontálisan és vertikálisan az egész FPGA-t lefedi, így – bizonyos megkötésekkel – bármely elem bármelyik másikkal összeköttetésbe hozható (5. ábra, 6. ábra).



5. ábra - Kapcsoló mátrix összeköttetések



6. ábra - A részegységek FPGA-n belüli tömbszerkezete

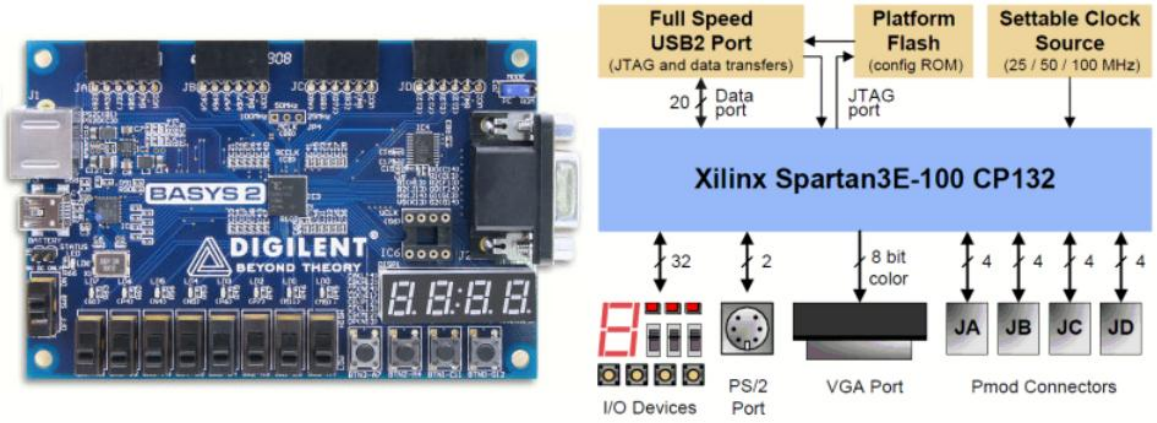
A Spartan-3 családról további információ a gyártó honlapján érhető el:

http://www.xilinx.com/support/documentation/spartan-3.htm

2.2 Digilent Basys™2 Spartan-3E FPGA Board

A mérés során a SPARTAN-3E típust tartalmazó demonstrációs panelt (Digilent Basys™2 Spartan-3E FPGA Board) fogjuk használni (7. ábra). A fejlesztőkártya paraméterei a következőek:

- Xilinx Spartan-3E FPGA, 100000 kapu
- 18-bites szorzók, 72Kbits dual-port block RAM
- USB 2 port az FPGA konfigurálásához (ingyenes Adept 2.0 szoftverrel)
- XCF02 Platform Flash ROM az FPGA konfiguráció tárolásához
- Programozható frekvenciájú oszcillátor (25, 50, and 100 MHz), egy további foglalat kvarc-oszcillátorhoz
- Stabilizált tápegységek (1.2V, 2.5V, and 3.3V)
- 8 LED, 4 db. hétszegmenses kijelző, 4 nyomógomb, 8 kapcsoló, PS/2 port, VGA port
- Felhasználói I/O csatlakozók

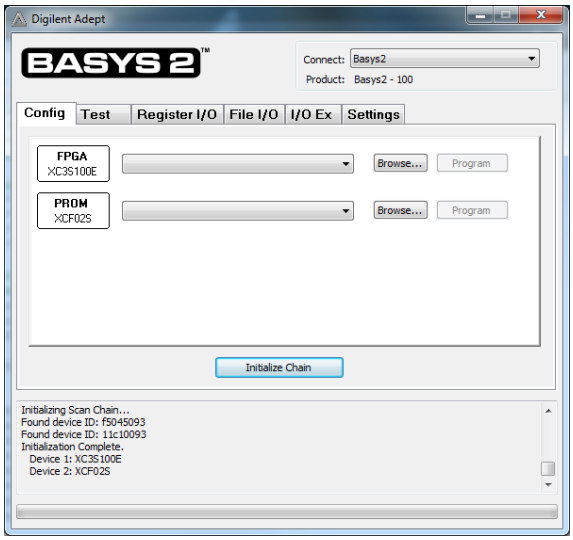


7. ábra - Digilent Basys™2 Spartan-3E FPGA panel

A panelt a hozzá tartozó USB kábel segítségével csatlakoztatjuk a PC-hez. Bár lehetőség van külső tápforrás bekötésére, a tápellátás és a programozás is az USB kábelen keresztül történik. A panel be- és kikapcsolása az SW8 jelű kapcsolóval végezhető.

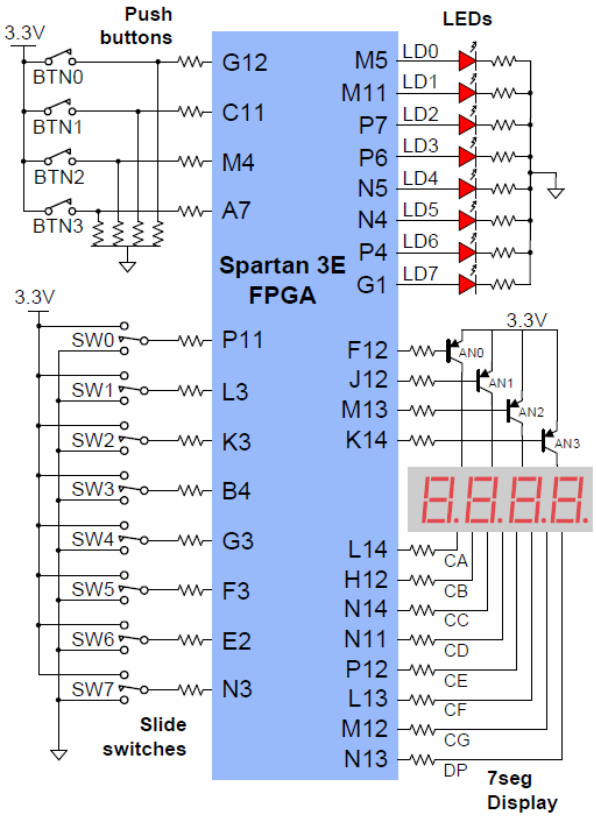
A programozásához szükséges (.bit kiterjesztésű) konfigurációs állományt – amelyet a későbbiekben ismertetett Xilinx ISE fejlesztő rendszerrel állítunk elő – a panel gyártója által ingyenesen hozzáférhető Adept segédprogram segítségével tölthetjük le közvetlenül az FPGA-ba, vagy a hozzá kapcsolódó külső konfigurációs PROM-ba (Platform Flash). A program kezelői felületét a 8. ábra szemlélteti. A közvetlenül az FPGA-ba töltött konfiguráció a tápfeszültség kikapcsolása után törlődik, a következő bekapcsolást követően az FPGA a Platform Flash-ben tárolt állomány szerint konfigurálódik fel automatikusan.

A panelen elhelyezkedő kapcsolók, nyomógombok és hétszegmenses kijelzők valamint az FPGA kapcsolatát a 9. ábra szemlélteti. A négy nyomógomb bemenet alapállapotban logikai „0” (L) szinten van. Logikai „1” szint (H) a gomb megnyomásával adható a bemenetekre. A nyolc kapcsoló konstans „L” vagy „H” szintet ad a hozzájuk tartozó bemenetekre. A nyolc LED a hozzájuk tartozó kimenetek logikai „1” szintje esetén világítanak.



8. ábra - Az Adept program konfigurációs felülete

A négy hétszegmenses kijelző szegmensei és a tizedes pont multiplexált módban vezérelhetők. A CA, CB, CC, CD, CE, CF, CG és DP jelekkel a szegmenseket és a tizedes pontot kapcsolhatjuk ki/be. Az AN0, AN1, AN2 és AN3 jelekkel választhatjuk ki, hogy a négy kijelző közül melyikre vonatkoznak a szegmenseket vezérlő jelek. A vezérlő és kiválasztó jelek megfelelően gyors egymás utáni kapcsolgatásával négydigites számokat jelezhetünk ki.



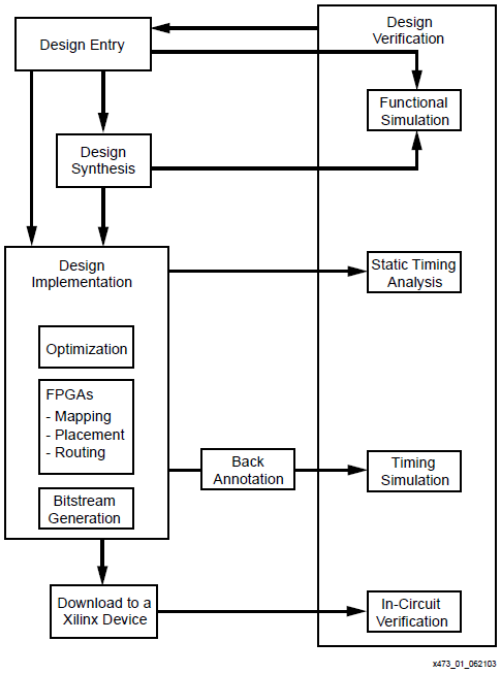
9. ábra - Kapcsolók, nyomógombok és hétszegmenses kijelzők

3 Xilinx ISE fejlesztő rendszer

A Xilinx ISE WebPack (Integrated Software Environment – integrált szoftverkörnyezet) a Xilinx cég FPGA-ihoz és CPLD-ihez kifejlesztett ingyenes szoftvere, amely az internetről, a cég weboldaláról szabadon letölthető. A fejlesztőrendszer tartalmazza mindazon elemeket, amelyek kapcsolási rajz alapú és hardverleíró nyelv alapú fejlesztéshez szükségesek.

A mérési segédlet bevezetést kíván nyújtani az áramkörök tervezésébe, célja a tervezési lehetőségek minél szélesebb körben való ismertetése. Elmélyülésre – a korlátozott laboridő miatt – saját munkával, illetve a konzultációk alatt van lehetőség. A mérések sikeres elvégzését nagymértékben segíti a mérést előkészítő előadás anyagának elsajátítása.

Az ISE tervező rendszer alrendszereinek, részeinek működését a Project Navigator szoftver, az ISE keretprogramja fogja össze. A tervezés menetét a 10. ábra szemlélteti.



10. ábra - A tervezés menete

A tervező az elképzeléseit, terveit:

- beviheti kapcsolási rajz (Schematic) formájában, a kapcsolási rajz készítő és beviteli program segítségével.
- beviheti hardver leíró nyelven. Ezt a HDL editor rész támogatja. A támogatott nyelvek: Verilog és VHDL. A rendszer sok mintaleírást is tartalmaz, úgynevezett sablonok (template) formájában.

A bevitel utáni az ellenőrzéssel megbizonyosodunk arról, hogy a terv szerinti áramkör működése megfelel-e a feladat specifikációjának. Az ellenőrzés általában szimulációval történik. A WebPACK rendszer szimulátora a Xilinx ISE Simulator.

A szimulációs vizsgálathoz a modellt működtetni, "gerjeszteni" kell, azaz a modell bemeneteire megfelelően változó jeleket kell adni. Ez az úgynevezett tesztvektorok sorozatának ráadásával történik. A tesztvektorokat a tervező HDL leírással adhatja meg (testbench). A régebbi ISE verziókban rendelkezésre álló grafikus felület az újabb verziókban már nem érhető el.

Ha a tervet rendben találtuk, akkor következhet a szintézis, amit jelen esetben a Xilinx Synthesis Technology (XST) alrendszer végez, amely ugyancsak az ISE része. A szintézer a HDL leírásból előállít egy minimalizált és optimalizált huzalozási listát, amely az adott FPGA primitíveket (LUT, FF…) és a köztük levő kapcsolatokat tartalmazza. Ezt követik a Translate, a Map és a Place&Route fázisok (összefoglalva: implementáció). A Translate a huzalozási listákból és a felhasználói megkötésekből (constraint-ek) generál egy fájlt, a Map leképezi ezt az adott FPGA primitív-készletére, végül a Place&Route elhelyezi a primitíveket az eszközben és kialakítja a köztük levő fizikai huzalozást.

A programozói fájl előállítását követő programozást normál esetben ("égetés") az IMPACT alrendszer vezérli. A mérések során használt demonstrációs panelhez a panel gyártója saját programozói felületet biztosít.

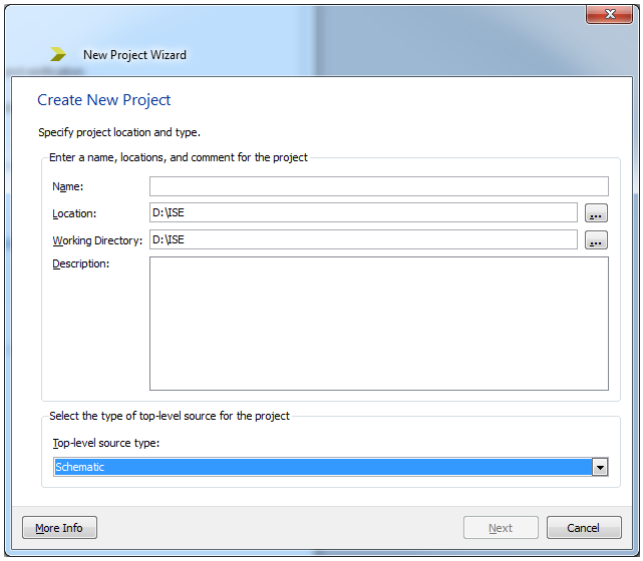
3.1 Első mintafeladat elkészítése – kapcsolási rajz alapú tervezés

A Xilinx ISE WebPack segítségével hozzunk létre egy egyszerű példaprogramot a gyakorlópanelen található LED-ek, kapcsolók és nyomógombok tesztelésére! A program ikonjára kattintva, vagy a Start menüből indítsuk el a Xilinx ISE Design Suite→ISE Design Tools→32-bit Project Navigator-t.

A mintapélda projekt létrehozásához a File menüben válasszuk a **New Project** parancsot, és töltsük ki a párbeszédablakot az alábbi módon (11. ábra).

Az ISE fejlesztő környezetben kerülni kell a fájlnevekben és a könyvtárszerkezetben a különleges karaktereket (ékezet, szóköz, vessző, pont, stb). A fájlok, könyvtárak lenevezésekor csak az angol ABC betüit használjuk!

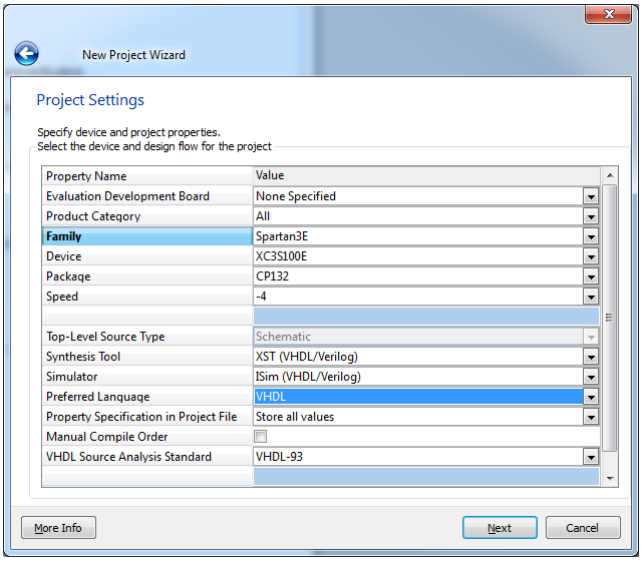
- A **Project Name** legyen „proba”. A rendszer automatikusan létrehoz egy ilyen nevű mappát a **Project Location** mezőben megadott elérési útnak megfelelően. Az ISE a projekthez tartozó állományokat ebbe a mappába fogja menteni.
- **Top-Level Module Type** mezőben a legördülő ablakból válasszuk a **Schematic**-ot.



11. ábra - Új projekt létrehozása

- A **Next** gombra kattintás után megjelenik a **New Project Wizard**, a **Project Settings** táblába írhatunk be paramétereket, válasszuk a próbapanelhez tartozó alábbi értékeket:
  - Family: Spartan3E

- Device: XC3S100E
- Package: CP132
- Speed : -4
- Synthesis Tool: XST (VHDL/Verilog)
- Simulator: ISim (VHDL/Verilog)

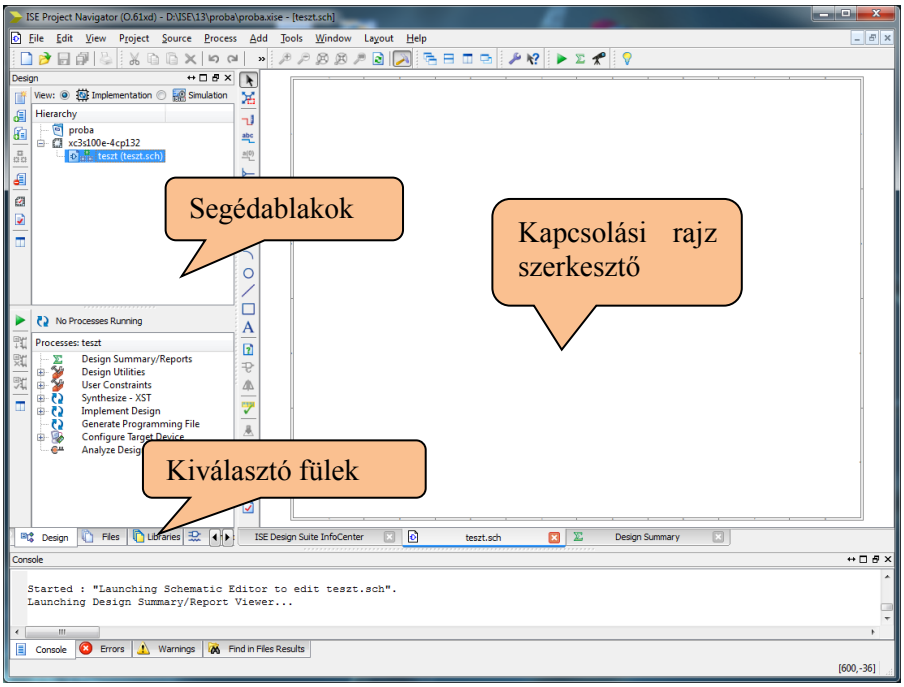


12. ábra - Új projekt létrehozása

A többi paramétert most nem kell változtatnunk. A **Next** gombra kattintva megjelennek a készülő projekt adatai. A **Finish** gombra kattintva elkészül a még üres próba projekt.

Új modul létrehozásához a **Project** menüben kattintsunk a **New Source** parancsra, és válasszuk a **Schematic** típusú forrást. A létrehozandó modul neve legyen „teszt”, amit a **File name** ablakba kell beírni, és jelöljük ki az **Add to Project** opciót. A **Next** majd **Finish** gombra kattintás után egy üres kapcsolási rajz szerkesztő jelenik meg, amiben elkészíthetjük a próbaalkalmazást.

A **View** menüpont **Panel** alpontjában kijelölhetjük, hogy a szerkesztő ablakban mely segédablakok jelenjenek meg. Egyelőre hagyjuk meg az alapértelmezés szerinti beállításokat. A jobb oldalon a rajzlap, a bal oldalon a **Select Options** jelenik meg. Itt a rajzolással kapcsolatos néhány paramétert állíthatunk be. Általában a képernyő az alábbi részekre van osztva (12. ábra13. ábra). Alul a Consol látható, ahol a parancsok szöveges formában jelennek meg.



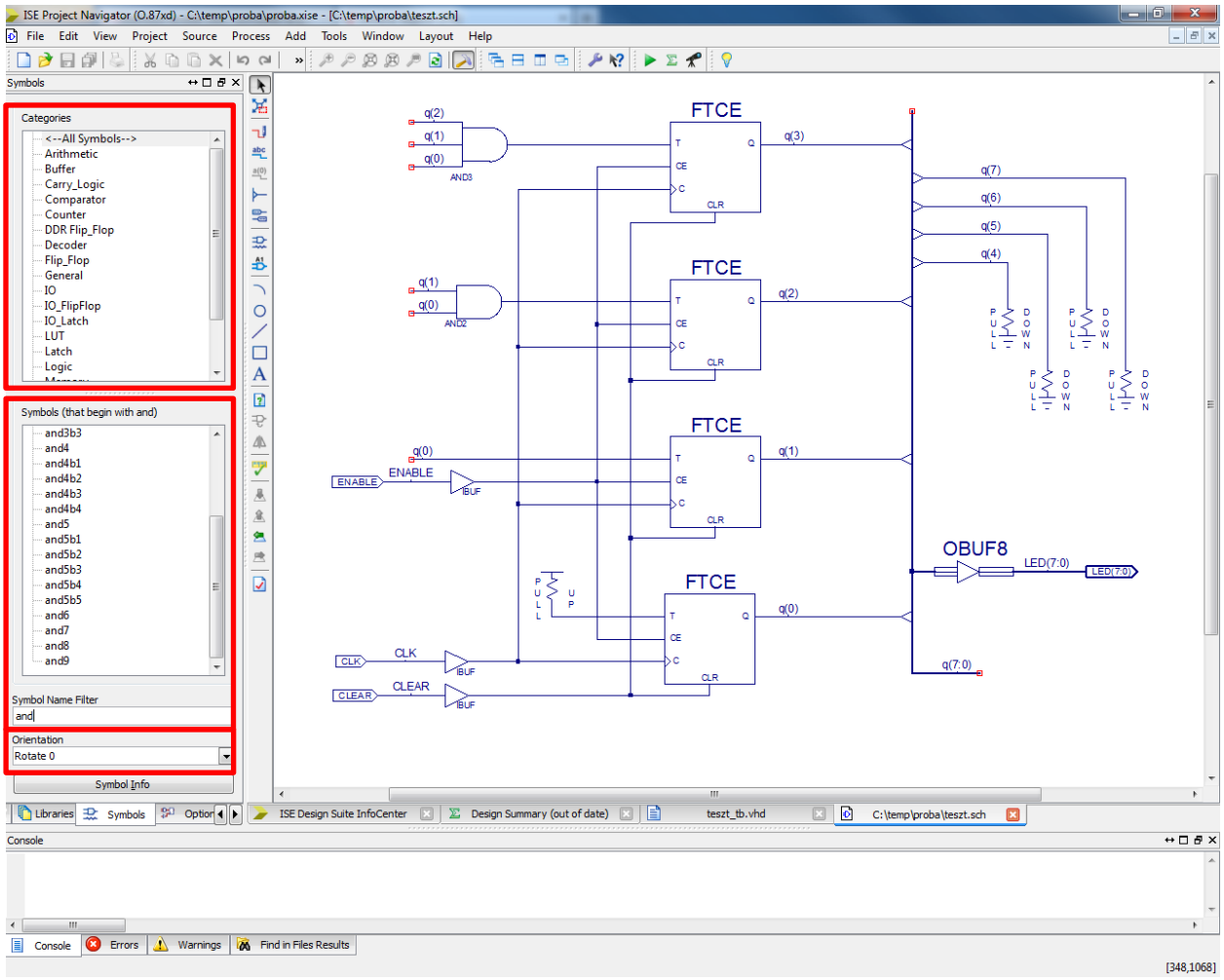
13. ábra - A Project Navigator részei

3.1.1 Kapcsolási rajz készítése

A tesztalkalmazásban egy T-tárolókból felépített 4-bites bináris számláló kapcsolást valósítunk meg. A számláló kimeneti bitjei a panelen található LED-eket hajtják meg, így ellenőrizhető a kapcsolás működése. A számlálót a Basys 2 FPGA fejlesztő kártyán található **SW0** kapcsolóval engedélyezzük, a **BTN0** nyomógombbal pedig nullázhatjuk. A segédablakok közül válasszuk a **Symbols** fület, ennek segítségével helyezhetjük el a kapcsolási rajzban használt alkatrészeket (14. ábra), felül a kategória, alatta a hozzátartozó egyik elem kiválasztásával:

- FTCE: Engedélyezhető T-flip-flop (4 db, a négybites számlálóhoz)
- AND3, AND2: három, ill. két bemenetű 'ÉS' kapuk (a bemeneti kombinációs hálózathoz)
- PULLUP: Pozitív tápfeszültség (logikai magas jelszint (H))
- PULLDOWN: Földpont (logikai alacsony jelszint (L))
- CONSTANT: Jelnek vagy busznak konstans értékadás.
- IBUF: Egybites bemeneti buffer (több bites változatai: IBUF4, IBUF8, IBUF16)
- OBUF: Egybites kimeneti buffer (több bites változatai: OBUF4, OBUF8, OBUF16)

Az alkatrészek megkeresése történhet kategóriák (**Categories**) alapján, de használhatjuk a névkeresőt is (**Symbol Name Filter**). Az alkatrészek forgatását az **Orientation** mezőben állíthatjuk be.



14. ábra - A Symbols segédablak és a használt alkatrészek

Az alkatrészek huzalozásakor a következő parancsokat, ill. parancsikonokat fogjuk használni:

- 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.

Mindegyik művelethez a bal alsó ablakban (Process/Options) találunk beállítási lehetőségeket, illetve műveleteket. A kényelmesebb szerkesztés érdekében megnövelhetjük a lap méretét is. Ezt a rajz területen jobb kattintásra felugró **Object Properties** parancs kiválasztásával tehetjük meg. A megjelenő ablakban a **Sheets** sort kiválasztva módosíthatjuk a lap méretét (**Size**).

Vezeték vagy busz elnevezéséhez válasszuk az **Add Net Name** parancsot majd a bal alsó ablakban az **Options** fülnél a **Name** sorba írjuk a nevet. Ezután kattintsunk az elnevezni kívánt vezetékre. A vezetékek neveire a kapcsolási rajz átláthatósága érdekében is hivatkozhatunk, ugyanis ha két vagy több vezetéket ugyan úgy nevezünk el, az olyan, mintha fizikailag összehuzaloztuk volna őket (lásd az és kapuk bemeneteit és a tárolók kimeneteit). A tárolók **q(0)**, **q(1)**, **q(2)** és **q(3)** kimenetei a speciális **q(7:0)** un. busz vezetékre csatlakoznak (buszvezetékeket a kapcsolási rajzokban az összetartozó

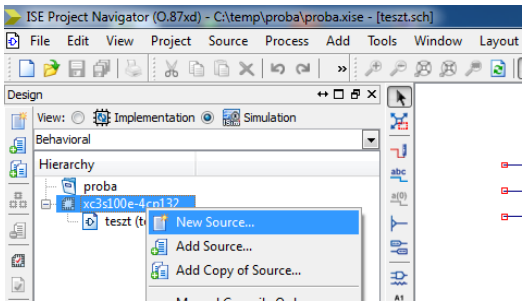
vezetékek ábrázolása érdekében használunk). A buszok, illetve leágazásaik elnevezése utal arra, hogy adott buszon belül melyik vonalra (bitre) csatlakozunk. A 14. ábra kapcsolásán például a **q(7:0)** egy 8 bites busz, melyre az adott helyértékű bitvezeték (**q(0)**…**q(7)**) buszcsatlakozáson (**Bus Tap**) keresztül kapcsolódik. A kapcsolás be- és kimeneti jelei **I/O buffer**-eken keresztül kapcsolódnak a külvilágot reprezentáló **I/O Marker**-ekhez. A CLK, ENABLE és CLEAR bemenetekre IBUF, a LED(7:0) kimenetekre pedig OBUF (itt egy 8 bites OBUF8) elemeket helyezünk el, ez a szabályos megoldás. Az automatikus buffer-elhelyezés miatt a kapcsolás ezek nélkül is implementálható, 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. A rajz ellenőrzése az alsó **Processes** területen kibontva a **Design Utilities**-t, a **Check Design Rules**-ra kattintva fut le, ami ha jó, egy zöld pipa jelenik meg mellette. Ugyanezt tehetjük meg a **Tools > Check Schematic** paranccsal.

3.2 A kapcsolás ellenőrzése szimulációval

Teszteljük le a kapcsolás működését szimulációval! Ehhez létre kell hoznunk egy ellenőrzési környezetet (testbench, tesztelési környezet) melyben az egységet működtető gerjesztő jeleket és jelformákat definiáljuk. Régebbi ISE verziókban lehetőség volt a gerjesztő-jelek grafikus felületen történő megadására, az újabb verziókból azonban kikerült ez a funkció, így az egyetlen lehetőség a HDL nyelvű leírás. A gyakorlatokon nem cél a HDL nyelvek ismertetése, ezért a tesztkörnyezetet leíró forrást nem kell elkészíteni, a kész forrásokat csak a projekthez kell adni.

3.2.1 Szimulációs fájl készítése

Az elkészült projekthez szimulációs fájlt a „Design/Simulation” ablakban, az „Add New Source” opcióval lehet hozzáadni (15. ábra). Fontos hogy minden port nevet adjunk meg a kapcsolási rajzon a szimulációs fájl elkészítése előtt, mert a program az aktuális port neveket használja fel a sablon elkészítésére.



15. ábra - Szimulációs fájl hozzáadása a projekthez

Ezután a „VHDL Test Bench” opciót válasszuk és adjuk meg a tesztfájl nevét (teszt_tb). A „Next” gomb lenyomása után a program megkérdezi melyik forrásfájlhoz készül a tesztfájl. Itt válasszuk a fő kapcsolási rajzot (példánkban: teszt.sch), majd válasszuk a „Next” gombot, végül kattintsunk a „Finish” gombra. Az ISE által generált sablon fájl tartalmazza a ki- és bemeneti portokat és a VHDL fájl keretét. Ha mindent jól csináltunk a következő VHDL fájlt kapjuk eredményül:

```
-- Vhdl test bench created from schematic C:\temp\proba\teszt.sch - Wed May
23 09:01:04 2012
--
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY teszt_teszt_sch_tb IS
END teszt_teszt_sch_tb;
ARCHITECTURE behavioral OF teszt_teszt_sch_tb IS

   COMPONENT teszt
   PORT( ENABLE :   IN  STD_LOGIC;
         CLK    :   IN  STD_LOGIC;
         CLEAR  :   IN  STD_LOGIC;
         LED    :   OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
   END COMPONENT;

   SIGNAL ENABLE:   STD_LOGIC;
   SIGNAL CLK    :   STD_LOGIC;
   SIGNAL CLEAR  :   STD_LOGIC;
   SIGNAL LED    :   STD_LOGIC_VECTOR (7 DOWNTO 0);

BEGIN

   UUT: teszt PORT MAP(
        ENABLE => ENABLE,
        CLK => CLK,
        CLEAR => CLEAR,
        LED => LED
   );

-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
      WAIT; -- will wait forever
   END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

A vizsgálójelek előállításához az alapértelmezett tesztfájlt módosítani kell. Az első kiegészítéssel adjuk hozzá a szimulációhoz a számlálót működtető 1MHz-es órajelet (CLK). A kódba a *constant clk_period : time := 1 us;* sor beszúrásával definiálásra kerül az órajel periódusideje és a *clk_process :process begin .... end* process-el az órajel előállításra kerül. Amennyiben nem CLK az órajel port neve, módosítani kell az általunk használt névre. A fenti kiegészítéseket elvégezve, a szimuláció során a CLK bemeneti porton egy 1 us periódusidejű órajel jelenik meg.

```
-- Vhdl test bench created from schematic C:\temp\proba\teszt.sch - Wed
May 23 09:01:04 2012
--
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY teszt_teszt_sch_tb IS
END teszt_teszt_sch_tb;
ARCHITECTURE behavioral OF teszt_teszt_sch_tb IS

   COMPONENT teszt
   PORT( ENABLE :   IN  STD_LOGIC;
         CLK    :   IN  STD_LOGIC;
         CLEAR  :   IN  STD_LOGIC;
         LED    :   OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
   END COMPONENT;

   SIGNAL ENABLE    :   STD_LOGIC;
   SIGNAL CLK   :   STD_LOGIC;
   SIGNAL CLEAR :   STD_LOGIC;
   SIGNAL LED   :   STD_LOGIC_VECTOR (7 DOWNTO 0);

   constant clk_period : time := 1 us;  --Órajel periódidő

BEGIN

   UUT: teszt PORT MAP(
        ENABLE => ENABLE,
        CLK => CLK,
        CLEAR => CLEAR,
        LED => LED
   );

   clk_process :process  -- Órajel előállítása
   begin
        CLK <= '0';
        wait for clk_period/2;
        CLK <= '1';
        wait for clk_period/2;
   end process;

-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
        --ENABLE jelnek logikai '0' és
        --CLEAR jelnek logikai '1' érték adása 2.3us időre
        ENABLE <= '0';
        CLEAR <= '1';
        wait for 2.3 us;

        --ENABLE='1' és CLEAR='0' 10.5us időre
        ENABLE <= '1';
        CLEAR <= '0';
        wait for 10.5 us;

        CLEAR <= '1';
        wait for 3 us;
        CLEAR <= '0';
        wait for 10.6 us;
        ENABLE <= '0';
        wait for 4.3 us;
        ENABLE <= '1';
      WAIT;     -- stop
   END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

Órajel periódusidejének definiálása

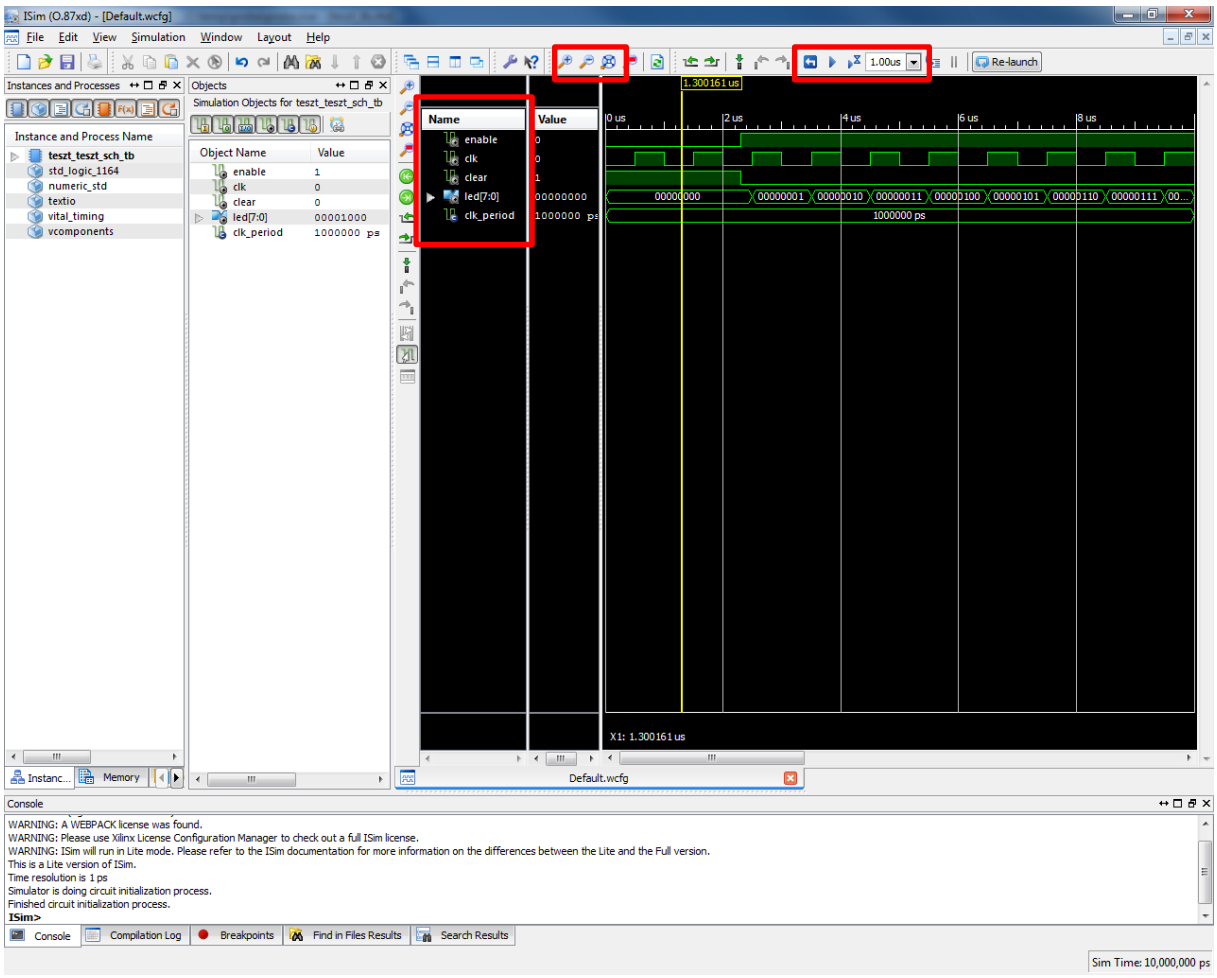
Órajel előállítása, a definiált clk_period periódusidővel.

Szimulációs jelek

A jeleknek értéket a „<=” értékadó utasítással lehet adni (példa: ENABLE<=’1’). Buszvezetékeknél értékadás binárisan történik (példa egy négybites busz értékadására: INB<=”1011”). A jelek beállítása után a **wait for** paranccsal adhatjuk meg mennyi ideig maradjanak meg a beállított értékek.

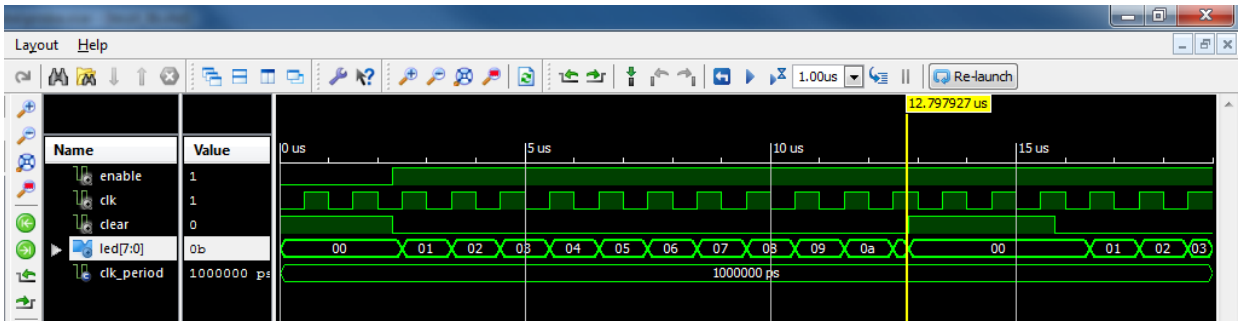
Lehetőség van kész VHDL testbench-et hozzáadni a projekthez. A *„valami_tb.vhd”* fájlt ami tartalmazza a szimulációs környezetet, célszerű a projekt könyvtárába másolni, majd a **Project** menü **Add Source** parancsával adhatjuk hozzá a projekthez. A megjelenő **Adding Source Files** ablakban az Association alatt a Simulation kell, hogy legyen. A szimuláció elindítása a korábban ismertetett módon történik.

A szimuláció elindításához a Design segédablakban jelöljük be a Simulation rádiógombot, majd jelöljük ki a megjelenő *„teszt_tb.vhd”* fájlt. Kibontva a **Isim Simulator**t az alsó segédablakban, megjelenik a **Simulate Behavioral Model** parancs, amivel elindíthatjuk az ISE szimulátorát. A szimuláció lépéseit alul, a Console területen láthatjuk. A szimuláció során a CLK órajel, az ENABLE és a CLEAR vezérlő jelek a VHDL forrásban megadottak szerint változnak. A szimuláció eredménye egy új ablakban fog megjelenni (16. ábra, a képen látható eredmény a **Zoom to Full View** ikonra kattintás után látható). A **Zoom In, Zoom Out, Zoom to Full View** ikonokkal (16. ábra, piros téglalappal jelölt rész) tudjuk a megjelenített időtartományt módosítani. A Restart ikon újraindítja a szimulátort, törli a megjelenített időidiagramot, **Run for the time specified on toolbar** ikon lenyomásakor a szimuláció a beállított időegységnyit lép előre.



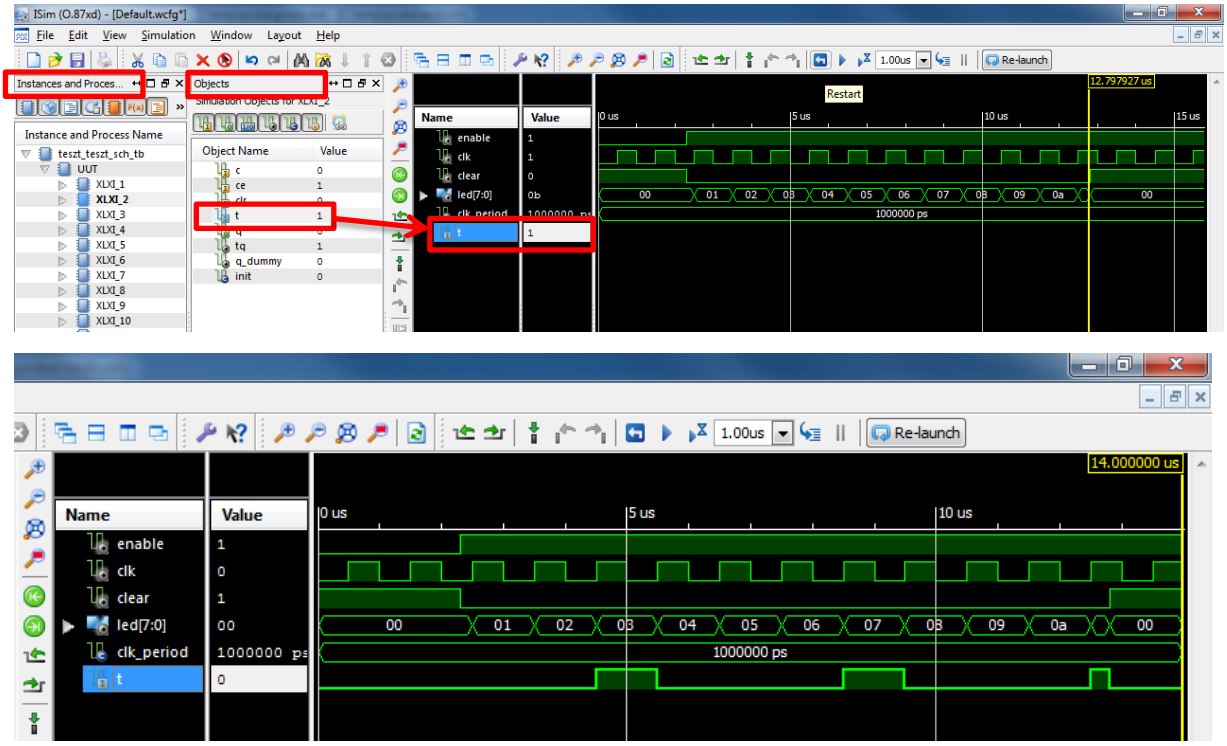
16. ábra - A szimuláció eredménye a Xilinx ISE Simulator hullámforma ablakában.

Az egyes jelnevekre a jobb gombbal kattintva lehetőség van a kijelzés formátumát megváltoztatni. A led[7:0] kimenet esetén ezt (jobb gomb, **Radix / Hexadecimal**) hexadecimálisra állítva máris könnyebben ellenőrizhető a helyes működés.



17. ábra - Kijelzés formátuma

Sokszor nagyon hasznos a modulok belső jeleit vizsgálni, ennek elvégzéséhez nem szükséges ezeket a jeleket kivezetni a modulból, a szimulátorban egyszerűen hozzáadhatjuk őket a **Wave** ablakhoz. A szimuláció a **View>Panels> Instance and Process Name** bal oldali ablakában megjelenik a szimulációs hierarchia, amelyben megtalálható (**UUT** – unit under test – néven) a tesztelés alatt álló kapcsolási rajz modul. A **View>Panels>Objects** ablakban látjuk a modul összes jelét, melyek közül a vizsgálni kívánt jel egyszerű drag-and-drop módszerrel adható hozzá a **Wave** ablakhoz (18. ábra). Ezután már csak a szimuláció újraindítására van szükség ahhoz, hogy a jelek értékeit figyelemmel tudjuk követni (Kék, enter-szerű gomb a felső menüben, majd a homokórás play-szerű gomb, amelyet nyomkodva, a mellette beállított értéknek megfelelő időt haladunk a szimulációban).



18. ábra - További jelek hozzáadása

Lehetőség van a szimulációs beállítások mentésére, ezt a **File>Save** paranccsal tehetjük meg. Az elmentett „***.wcfg**” fájlt a szimulátorban megnyitva, visszaállítható a korábbi konfiguráció. A

szimulátor bezárásakor, ha nincs elmentve a szimulátor konfigurációja, a program megkérdezi, hogy szeretnénk e menteni a beállításokat.

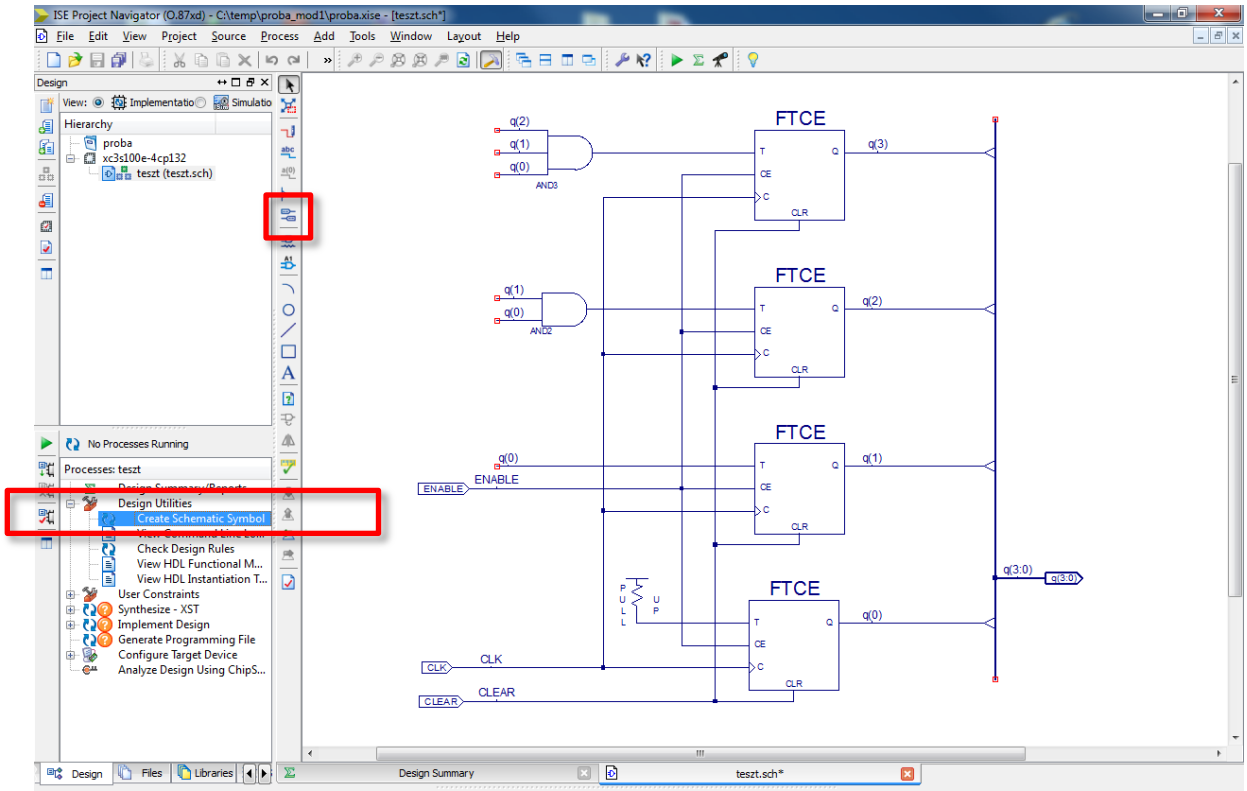
3.3 Makró hozzáadása meglévő projekthez

Minden az ISE kapcsolási rajz szerkesztőjében használt makróhoz két fájl tartozik. Az egyik egy szimbólumfájl (*.sym), ami leírja a makró kapcsolási rajzon való megjelenését. A másik a működést leíró fájl, ami lehet kapcsolási rajz (*.sch), vagy HDL (*.vhd) alapú.

Ha makrót akarunk a rajzból készíteni későbbi felhasználásra, akkor a portok be- és kimenetekre történő hozzáadása után, a **Design** ablak alsó **Processes** részén kibontva a **Design Utilities** menüpontot rá kell kattintanunk a **Create Schematic Symbol**-ra. Az eddigi rajz neve alatt egy új elem jelenik meg, amit egy másik rajzra le is tehetünk. Ha rögtön makrót akarunk készíteni, akkor más az eljárás. A **Project Navigator**-ban, kiválasztva a **Project > New Source** menüpontot, megjelenik a **New Source Wizard** ablak. A forrás típusaként kiválasztjuk a rajzot (**Schematic**), megadjuk a fájl nevét és helyét majd a **Next** és **Finish** megnyomása után hozzáadódik a projekthez és megnyílik az adott nevű üres rajz. Ezután i/o portokat kell a rajzhoz hozzáadnunk a **Tools > Create I/O Markers** segítségével. A megjelenő dialógus dobozba be kell írni sorban, vesszővel elválasztva, az összes bemeneti, kimeneti és esetleg kétirányú port nevét. Az üres rajzon megjelennek a portok, ezután hozzárajzoljuk a makró többi részét. Ha kész, el kell mentenünk.

3.3.1 Példa kapcsolási rajz alapú makró készítésére

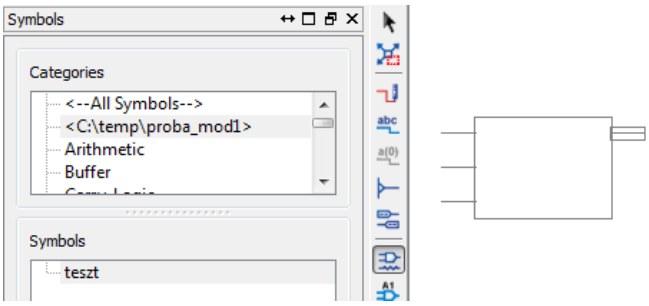
A következő példa során a korábbi négybites számlálót felhasználva készítjük el a makrónkat, így később lehetőség nyílik rá, hogy a számlálót önálló szimbólumként kezeljük. A 19. ábra szerint módosítsuk az előző feladatban szereplő számlálót. A korábbi kapcsolási rajzhoz képest csak a kimeneti busz szélessége változott, valamint a ki és bemeneti portokon csak IO markerek találhatóak. Az IO markereket a „ADD" I/O Marker" nevű gombbal lehet hozzáadni. IBUF-ra és OBUF-ra csak olyan portok esetén van szükség ami a külvilággal tart kapcsolatot.



19. ábra – Kapcsolási rajz módosítása

Az I/O markerekhez kapcsolódó vezeték neve lesz az IO marker neve és ez lesz a makró ki- és bemeneti portjának a nev is. A kapcsolási rajzot kijelölve, megjelenik a „Create Schematic Symbol” opció (19. ábra). Ezt a processt futtatva elkészül a makró (létrejön egy sym kiterjesztésű fájl a kapcsolási rajz mellett).

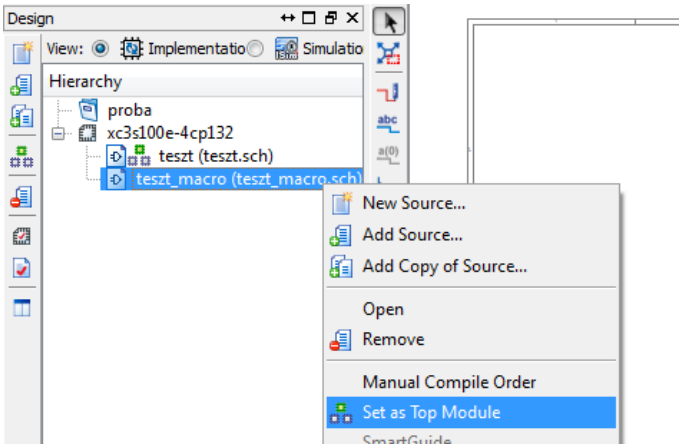
A fejlesztő környezet parancssorában a makró sikeres elkészítését a következő üzenet jelzi: *„Process "Create Schematic Symbol" completed successfully”*. A szimbólumok között ezután már megtalálható a makró szimbólum (20. ábra).



20. ábra - Elkészült makró kiválasztása

A makró tartalmát a „View” menü „Push into Symbol” gombjával lehet megtekinteni. Előtte szükséges kijelölni a makrót, a kapcsolási rajzon!

A korábbi teszt.sch fájl most szimbólum fájlként létezik, így szükséges egy új kapcsolási rajz fájl hozzáadása a projekthez, aminek adjuk a teszt_macro.sch nevet (21. ábra). Az új kapcsolási rajz file jelenleg nem „Top Modul”, ezért módosítsuk a teszt_macro.sch fájl beállításait top modul szintűre (21. ábra).

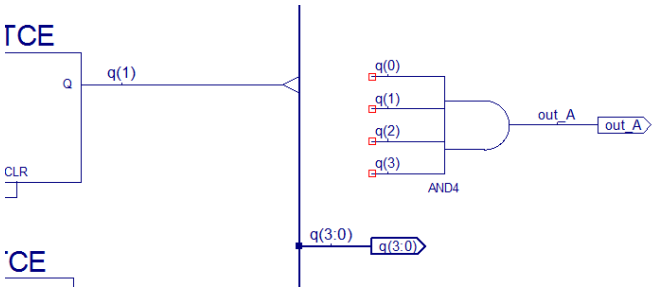


21. ábra - Top modul

3.3.2 Kapcsolási rajz alapú saját makró módosítása

Ha a változtatás nem érinti a makró ki- és bemeneti portjait, akkor a „Push into Symbol” paranccsal megnyitva lehetőség van a makró módosítására. A változásokat minden esetben el kell menteni.

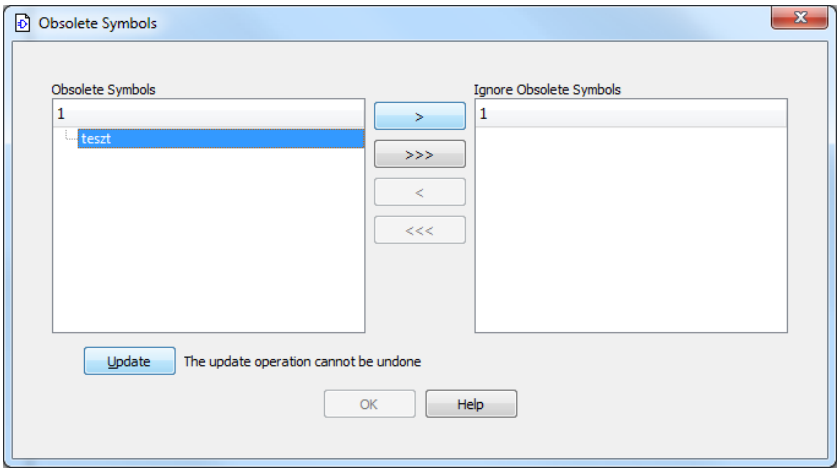
Amennyiben a változás a makró megjelenésére is kihat, például új portot kell hozzáadni, törölni, vagy port nevet kell változtatni, akkor a következőképpen kell eljárni. Első lépésként meg kell nyitni a makrót tartalmazó sch fájlt. Ezt a „Push into Symbol” paranccsal, vagy a „Design” ablakban a kapcsolási rajzra kétszer kattintva lehet megtenni. Végezzük el a változtatásokat a makró fájlon. A példában a régi rajzot egészítsük ki egy új kimenettel, ami a számláló 15-ös értékét jelzi (22. ábra).



22. ábra - Kimenet

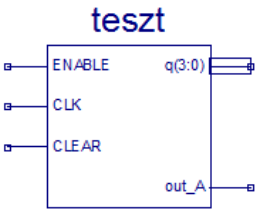
A módosítások mentése után a teszt.sch kapcsolási rjz fált kijelölve, a „Create Schemetic Symbol" parancs-ra jobb gombbal kattintva a „Process Properties…" opciót kiválasztva engedélyezni kell a létező szimbólumfájl felülírását, úgy hogy az „Owerwrite Existing Symbol" jelölő négyzetet bepipáljuk.

A „Run" parancs futtatásával módosul a makró megjelenését leíró szimbólum fájl. Ha valamelyik kapcsolási rajzban használunk egy ilyen makrót, amit módosítottunk, majd a makró megjelenésére kiható változtatatást hajtunk végre, a kapcsolási rajz szerkesztőben „Obsolate Symbols" ablak jelenik meg. Az „Update" paranccsal frissíthetjük a kijelölt objektumra vonatkozó módosításokat.



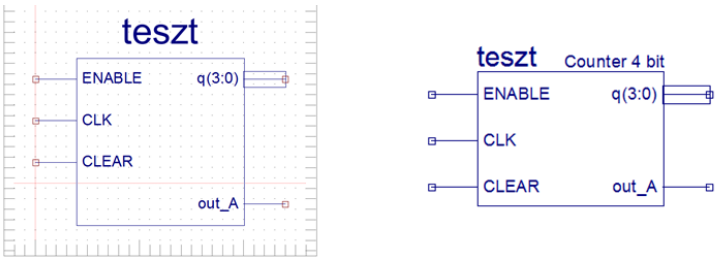
23. ábra - Kapcsolási rajz alapú makró módosítása

Ha a makró újrafordítása után nem kínálja fel a program a makró szimbólum frissítését, és a kapcsolási rajzon nem is változik meg a makró megjelenése, akkor újra kell indítani az ISE fejlesztőkörnyezetet (előtte mindent mentsünk el!). Az újraindulás után már a módosított szimbólumnak kell megjelennie (24. ábra).



24. ábra - Módosított szimbólumfájl

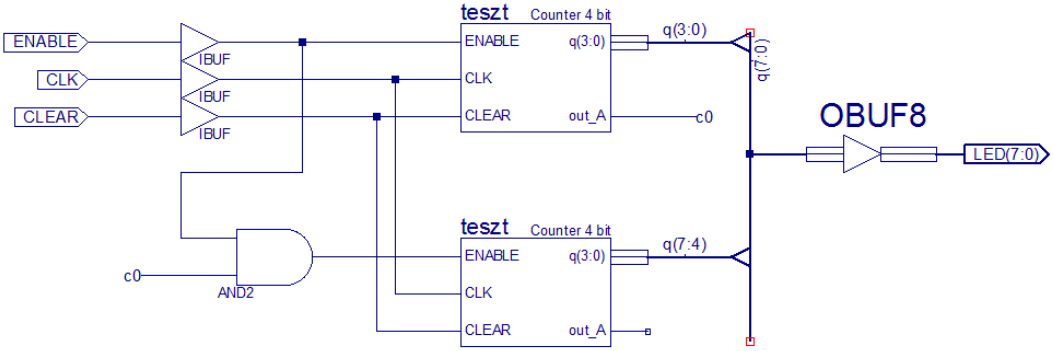
A szimbólumfájl külső megjelenésén is lehetőség van módosítani. A kapcsolási rajzon elhelyezett teszt nevű makró fájlra jobb gombbal kattintva válasszuk a **Symbol->Edit Symbol** opciót (25. ábra).



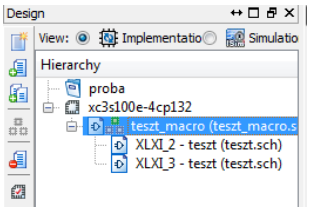
25. ábra - Makró szerkesztés

3.4 Kapcsolási rajz elkészítése a makró használatával

Készítsünk egy új kapcsolási rajzot „teszt_macro.sch” néven. A 26. ábra szerint készítsük el a kapcsolási rajzot. Felhasználva a négybites számláló makrót, készíthetünk 8 bites számlálót. Figyeljünk rá, hogy a „teszt_macro.sch” rajz legyen a topmodul (27. ábra).



26. ábra - 8 bites számláló kapcsolási rajza

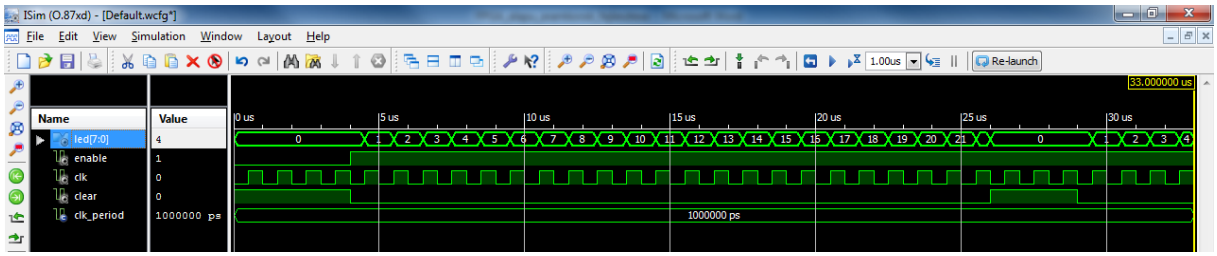


27. ábra - 8 bites számláló fájlok

Az áramkör szimulációját hasonló képen végezzük el, mint a 4 bites számláló esetében. A tesztfájl elkészítése után a következő VHDL kódot használjuk fel

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY teszt_macro_teszt_macro_sch_tb IS
END teszt_macro_teszt_macro_sch_tb;
ARCHITECTURE behavioral OF teszt_macro_teszt_macro _sch_tb IS

   COMPONENT teszt_macro
   PORT( ENABLE :   IN  STD_LOGIC;
         CLK    :   IN  STD_LOGIC;
         CLEAR  :   IN  STD_LOGIC;
         LED    :   OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
   END COMPONENT;

   SIGNAL ENABLE    :   STD_LOGIC;
   SIGNAL CLK   :   STD_LOGIC;
   SIGNAL CLEAR :   STD_LOGIC;
   SIGNAL LED   :   STD_LOGIC_VECTOR (7 DOWNTO 0);

   constant clk_period : time := 1 us;  --Órajel periódidő

BEGIN
   UUT: teszt_macro PORT MAP(
        ENABLE => ENABLE,
        CLK => CLK,
        CLEAR => CLEAR,
        LED => LED
   );
   clk_process :process  -- Órajel előállítása
   begin
        CLK <= '0';
        wait for clk_period/2;
        CLK <= '1';
        wait for clk_period/2;
   end process;

-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
         ENABLE <= '0';
         CLEAR <= '1';
         wait for 4 us;
         ENABLE <= '1';
         CLEAR <= '0';
         wait for 22 us;
         CLEAR <= '1';
         wait for 3 us;
         CLEAR <= '0';
         wait for 10.6 us;
         ENABLE <= '0';
         wait for 4.3 us;
         ENABLE <= '1';
      WAIT;     -- stop
   END PROCESS;
-- *** End Test Bench - User Defined Section ***
END;
```

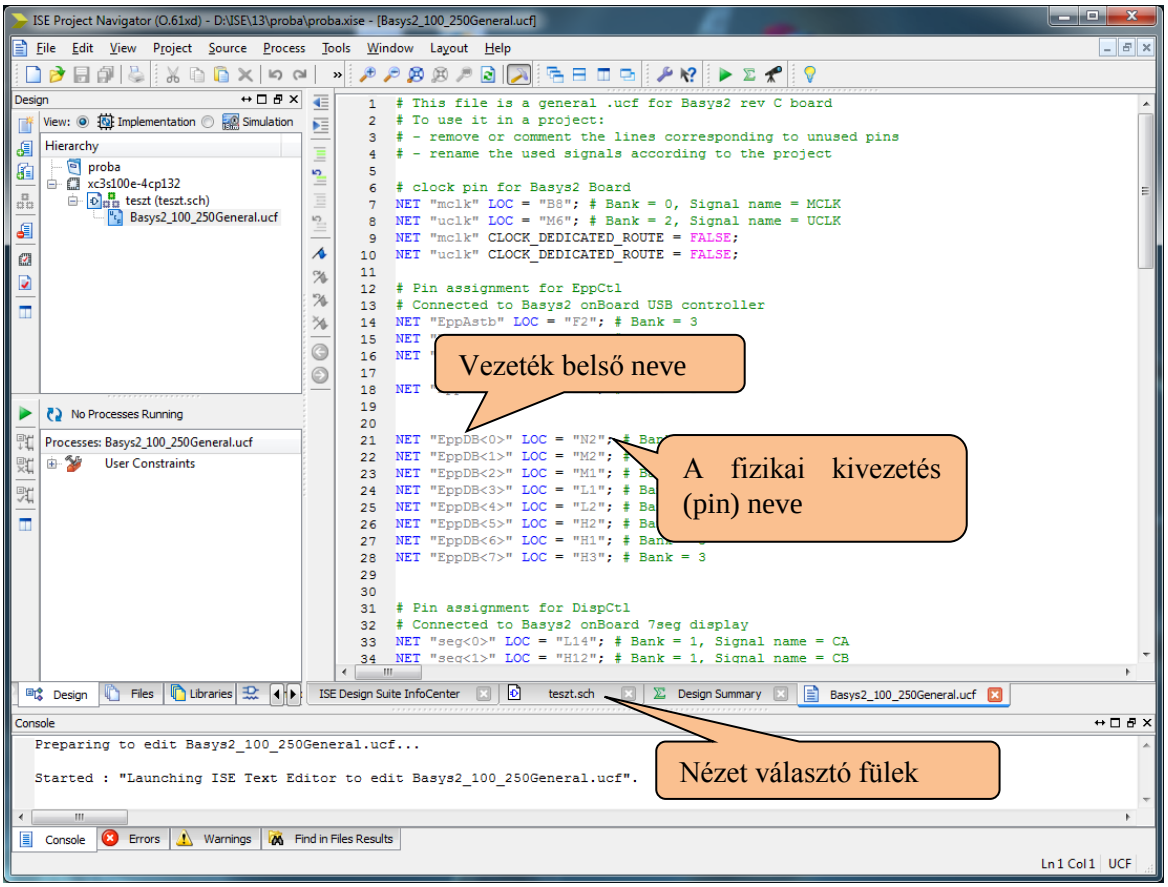
A szimuláció lefuttatása után a 28. ábra szerinti eredményt kell kapnunk.



28. ábra - 8 bites számláló szimulációs eredmény

3.5 A kapcsolás hardveres ellenőrzése

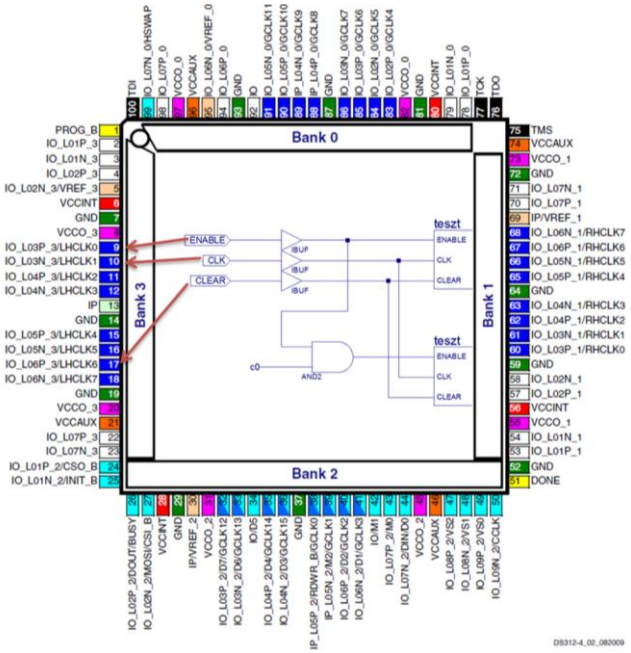
Jelöljük be az **Implementation** rádiógombot a **Design** panelen. A **Project** menü **Add Source** parancsával adjuk hozzá a projekthez a laborok során használt Basys™2 fejlesztőpanel alapértelmezett lábkiosztását és a felhasználói megkötéseket (constraint-ek) tartalmazó „*Basys2_100_250General.ucf*” fájlt. A **Megnyitás** és **OK** gombok megnyomása után a projektben megjelenik a kiválasztott fájl, amit kettős kattintással szerkesztésre meg is nyithatunk (29. ábra). A fájl egyéb – itt nem részletezett – felhasználói megkötések mellett a kapcsolási rajzban használt vezetékek neveit (NET) és a hozzájuk tartozó FPGA IC kivezetések (lábak) összerendeléseit definiálja.



29. ábra - A „Basys2_100_250General.ucf” fájl

A „*Basys2_100_250General.ucf*” fájl az FPGA összes panelen bekötött kivezetéséhez alapértelmezett belső vezetékneveket rendel. Az alapértelmezett elnevezések segítenek a panelen való tájékozódásnál,

azonban nincsenek összhangban a kapcsolási rajzban általunk használt I/O Marker nevekkel, emellett legtöbbjüket nem is használjuk.



30. ábra -

A helyes működéshez, módosítsuk az alapértelmezett ucf fájlt az alábbiak szerint.

```
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;

# Pin assignment for LEDs
NET "Led<7>"   LOC = "G1" ; #Bank = 3, Signal name = LD7
NET "Led<6>"   LOC = "P4" ; #Bank = 2, Signal name = LD6
NET "Led<5>"   LOC = "N4" ; #Bank = 2, Signal name = LD5
NET "Led<4>"   LOC = "N5" ; #Bank = 2, Signal name = LD4
NET "Led<3>"   LOC = "P6" ; #Bank = 2, Signal name = LD3
NET "Led<2>"   LOC = "P7" ; #Bank = 3, Signal name = LD2
NET "Led<1>"   LOC = "M11"; #Bank = 2, Signal name = LD1
NET "Led<0>"   LOC = "M5" ; #Bank = 2, Signal name = LD0

# Pin assignment for SWs
NET "enable"   LOC = "P11"; #Bank = 2, Signal name = SW0

NET "clk"      LOC = "C11"; #Bank = 2, Signal name = BTN1
NET "clear"    LOC = "G12"; #Bank = 0, Signal name = BTN0
```

Ezzel a kapcsolási rajzunk Led(7:0) kimeneteit a panelen található LED-ekhez, az ENABLE bemenetet az SW0 kapcsolóhoz a CLK és CLEAR bemeneteket pedig a BTN1 és BTN0 nyomógombokhoz rendeltük (a buszvezetékek elnevezésében szereplő zárójelek az *.ucf*-fájlban <> karakterekkel helyettesítendők). Az összes többi nem használt vezetékek definíciót tartalmazó sorokat a # karakterrel alakítsuk megjegyzéssé (comment), vagy töröljük. Mentsük el a változtatásokat.

A kapcsolás implementálásához és az FPGA konfigurációs állományának előállításához a **Design** segédablakban jelöljük ki a *„teszt.sch”* kapcsolási rajzot. Ennek hatására a **Processes** ablakban az **Implement Design** kibontásakor megjelenik a **Generate Programming File** parancs, amelynek feladata, hogy létrehozza az FPGA-ba letölthető *„teszt.bit”* állományt. Ez általában elég sokáig tart, a művelet lépéseit a **Console** területen követhetjük. Először a parancs futtatásának eredménye

hibaüzenet, amit az **Errors** ablakban tekinthetünk meg. Mivel a kapcsolásban a tárolók órajele nem dedikált órajel vezetéken keresztül kerül az FPGA-ba, a program figyelmeztet. Mivel esetünkben ez a nem optimális vezetékezés elfogadható, az ucf-fájlban el kell helyeznünk a

```
NET "CLK" CLOCK_DEDICATED_ROUTE = FALSE;
```

kiegészítést. Az újabb **Generate Programming File** parancs futtatásakor a fenti hibajelzés már csak a **Warnings** ablakban jelenik meg. Az elkészült konfigurációs állományt (*„teszt.bit”*) a 3. fejezetben ismertetett Adept program segítségével letölthetjük az FPGA-ba.

3.6 A konfigurációs fájl letöltése az FPGA-ba

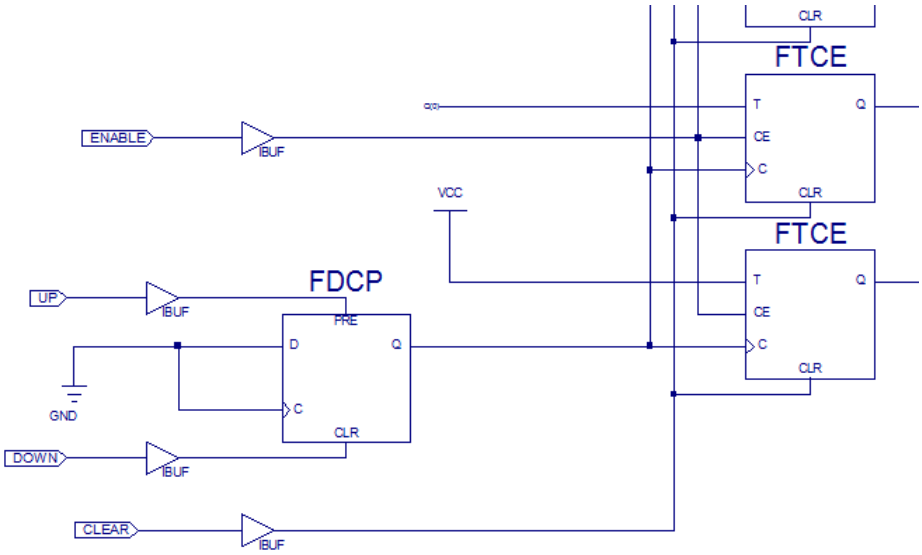
Először telepíteni kell az Adept kezelő programot (jelenleg digilent.adept.system_v2.8.1.exe), ezt még azelőtt kell megtenni, mielőtt a tesztkártyát az PC-hez csatlakoztatnánk. Csatlakoztatáskor a Windows észreveszi az új hardvert, engedélyezni kell a driver telepítését. Az Adept indulásakor észreveszi, hogy melyik eszköz van csatlakoztatva. A **Config** fül kiválasztásával **Browse**-ra kattintva kikeressük az adott bit kiterjesztésű fájlunkat és bevisszük az FPGA melletti edit box-ba. A **Program** gombra kattintva megtörténik a letöltés, és a kártyán azt látjuk, ami sikerült.

3.7 Az órajel bemenet prellmentesítése

A prellezés (a mechanikusan működő villamos érintkezők egymáson való pattogása) miatt az órajelet szolgáltató BTN1 nyomógomb nem mindig ad rögtön stabil jelet. Rövid ideig egyszeri lenyomás hatására is két állapot között „pereg”, ezzel többszöri 0 → 1 átmenetet adhat a tárolók órajel bemenetére. Ennek a jelenségnek a kiküszöbölésére módosítsuk a kapcsolást a 19. ábra szerint, az ucf-fájlt pedig az alábbiak szerint.

```
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;

NET "up"      LOC = "A7";  #Bank = 1, Signal name = BTN3
NET "down"    LOC = "M4";  #Bank = 0, Signal name = BTN2
#NET "clk"    LOC = "C11"; #Bank = 2, Signal name = BTN1
NET "clear"   OC = "G12";  #Bank = 0, Signal name = BTN0
```

Az aszinkron törlés és beállítás (CLR, PRE) bemenetekkel rendelkező FDCP flip-flop az UP bemenetre csatlakozó BTN3 nyomógomb megnyomásával (preset) logikai 1 szintre vált, a DOWN bemenetre csatlakozó BTN2 nyomógomb megnyomásával (clear) pedig logikai 0 szintre vált. Egy teljes órajel periódust (felfutó él - lefutó él) a BTN3 és BTN2 nyomógombok egymás utáni lenyomásával hozhatunk létre.



31. ábra - A „teszt.sch” kapcsolási rajz módosítása