

Bevezető az ISE 14.7 rendszer használatához

(Szántó Péter, BME MIT 2024-09-02)

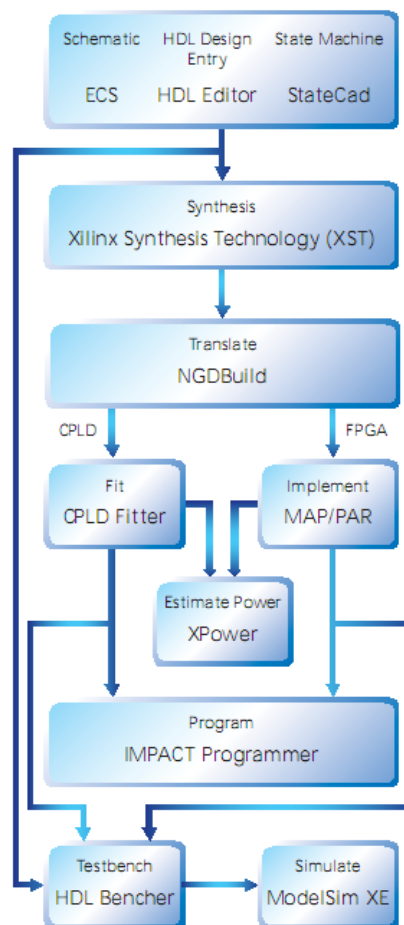
Ez az ismertető a ISE 14.7 fejlesztő rendszer alkalmazásába kívánja bevezetni az olvasót. Valóban csak bevezetőről van szó, mert a rendszert néhány nagyon egyszerű példán keresztül mutatjuk be, és sok funkciójáról nem is teszünk említést.

Először két nagyon egyszerű feladaton keresztül szerzünk némi gyakorlatot a fejlesztési folyamatban, majd egy BCD számlálót tervezünk meg, az egység működését hardver leíró nyelven specifikálva. A tervező rendszer szimulátorával ellenőrizzük (verifikáljuk) a terv helyességét. Ezt követően egy több funkcionális egységből felépülő, egyszerű készüléket tervezünk. A készülék működését Verilog nyelven, hierarchikusan írjuk le. Az elkészült leírást a tervező rendszerrel szintetizáljuk, és egy FPGA áramkörre képezzük le. Végül az elkészült konfigurációs fájlt letöltjük a mérőpanel FPGA áramkörébe, és kipróbáljuk a készülék működését.

1. Az ISE rendszer részei

A Xilinx cég, a programozható logikai eszközök (PLD és FPGA) egyik jelentős gyártója, kidolgozott egy számítógépes tervező rendszert ezeknek az eszközöknek a használatához. Ennek a rendszernek a neve: Xilinx ISE Logic Design Tools. A cég az ISE rendszer egyszerűbb, de funkcionálisan komplett változatát is összeállította, ami a WebPACK nevet kapta. A WebPACK rendszert a Xilinx cég (www.xilinx.com) díjtalanul bocsátja a felhasználók rendelkezésére, a felhasználónak csak regisztrálnia kell magát a szoftver letöltéséhez. A WebPACK rendszer természetesen csak a Xilinx cég IC-ivel való implementálást támogatja (de nem támogatja az összes család összes IC-jét, hanem tipikusan csak a kisebb komplexitásúakat). Az ingyenes szoftverrel a cég nyilván meg akarja könnyíteni az áramköreinek elterjedését.

Az ISE rendszer részeit jól szemlélteti az 1. ábra, amely a Programmable Logic Design Quick Start Handbook c. kiadványból [2] származik, mely elérhető a www.xilinx.com/univ/ weblapon keresztül.



1. ábra Tervezési folyamat a WebPACK rendszerrel

Az ISE tervező rendszer alrendszerinek, részeinek működését a Project Navigator szoftver, az ISE keretprogramja fogja össze.

A tervező az elképzeléseit, terveit háromféle formában viheti be a rendszerbe.

- Beviheti kapcsolási rajz (Schematic) formájában, a Xilinx ECS (Engineering Capture System), a kapcsolási rajz készítő és beviteli program segítségével. HASZNÁLATA NEM JAVASOLT.
- Beviheti hardver leíró nyelven. Ezt a bevitelt a HDL editor rész támogatja. A támogatott nyelvek: Verilog és VHDL. A rendszer sok mintaleírást is tartalmaz, úgynevezett sablonok (template) formájában.
- Illetve rendelkezésre állnak egyéb opciók, mint például a Xilinx által készített modulok (IP-k) használata vagy processzoros rendszer létrehozása.

A bevitel után következik a terv verifikálása, aminél azt ellenőrizzük, hogy a terv szerinti áramkör működése megfelel-e a feladat specifikációjának. A verifikálás 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 ún. tesztvektorok sorozatának ráadásával történik. A tesztvektorokat a tervező beírhatja a HDL leírásba, mint tesztelési környezete (testbench) – a régebbi ISE verziókban rendelkezésre álló grafikus felület már nem áll rendelkezésre.

Ha a tervet rendben találtuk, akkor következhet a szintézis, amit jelen esetben a Xilinx Synthesis Technology (XST) alrendszer végez, amely ugyancsak az ISE része (megj.: szintézisre léteznek más programok is). A szintézer a HDL leírásból előállít egy minimalizált és optimalizált huzalozási listát, amely az adott FPGA-ban rendelkezésre álló hardver erőforrásokat (az érthetőség kedvéért hazudjuk azt, hogy logikai kapuk és tárolók), és a köztük levő kapcsolatokat tartalmazza (megj: valójában a Xilinx FPGA-k LUT-okat és D FF-okat tartalmaznak, de ez most nem túl lényeges). 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, míg végül a Place&Route elhelyezi a primitíveket az eszközben, és kialakítja a köztük levő fizikai huzalozást.

A programozói fájl előállítását követő IC beprogramozását az ISE-ben az IMPACT alrendszer vezérli, de mivel a használt kártyáknak saját driver-e van, így a laboron nem ezt, hanem a saját driver-hez készült Logsys GUI-t fogjuk használni.

2. Az ISE rendszer használata az alaplaborban

2.1 A választott tervezési eljárás

Az ISE rendszer meglehetősen bonyolult, ami az általa nyújtott sok szolgáltatásból is következik. A rendszer alapos megismerésére az alaplaborban nincs elegendő idő, és természetesen idő hiányában kiváló FPGA tervezőt sem tudunk itt kiképezni – a cél a modern digitális áramkör tervezés alapjainak megismerése. Az érdeklődő hallgatóknak javasoljuk az „FPGA alapú rendszerek” MSc mellékspecializációt, ahol sokkal részletesebb és széleskörűbb ismeretekre tehetnek szert.

Az alaplabor számára kiválasztott tervezési eljárás a terv Verilog alapú leírásán és bevitelén alapul, tehát a HDL Design Entry alrendszert használja.

A terv bevitel után következik a funkcionális ellenőrzés szimulációval.

A helyes terv elkészülte után a fejlesztés további lépései már adottak a WebPACK rendszerben. (Ezek a lépések a terv kijelölése után a **Processes** ablakban is megtekinthetők, ill. indíthatók.)

A fejlesztéshez meg kell adni az ún. felhasználói megkötéseket (User Constrains). A mi esetünkben ez annak megadását jelenti, hogy az egyes jelek az FPGA melyik lábára kapcsolódnak (egyéb constraint-ek vonatkozhatnak még az időzítési paraméterekre, valamint az egyes részegységek elhelyezésére a chipen belül).

Ezután indítható a WebPACK szintézis (Synthesize) alrendszere, majd a szintetizált RTL leírás leképezése az adott FPGA struktúrára (Implement Design: Translate, Map, Place & Route).

Legvégül az FPGA-t konfiguráló állomány elkészítése következik (Generate Programming File).

Az előzőekben vázolt tervezési lépéseket a következőkben egy mintafeladaton fogjuk bemutatni.

A tervezés során betartjuk a digitális hálózatok korrekt tervezéseinek alapszabályait, amelyet a tantárgy házi feladatainak elkészítésekor is elvárunk.

2.2 A korrekt tervezés erre a területre vonatkozó fontosabb előírásai

1. Csak szinkron sorrendi hálózatokat tervezzünk!

Az alaplaborban ennek egy szigorúbb megkötésével élünk: csak szigorúan szinkron sorrendi hálózatokat szabad tervezni, a hálózatnak csak egyetlen órajele lehet. (Erre az alaplaborban azért van szükség, mert a hallgatóknak nincs megfelelő gyakorlata, és így kevesebb probléma lép fel a terv feldolgozásánál.)

2. A szinkron hálózat elsődleges (kívülről érkező) bemenő jeleit szinkronizálni kell, hogy elkerüljük a bemenő jel aszinkronitása okozta metastabil állapot felléptét a rendszerben.

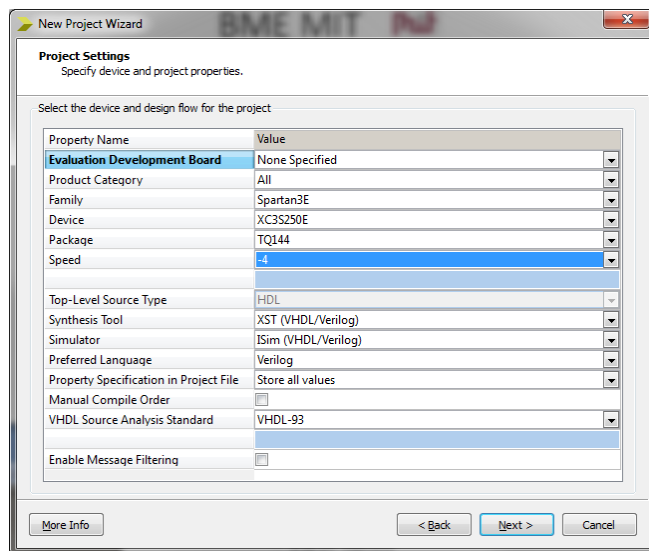
3. A tervnek ill. a készüléknek könnyen ellenőrizhetőnek kell lennie. Ezzel a témával egy külön szakterület, a tesztelhetőségre tervezés foglalkozik. Az egyszerűbb hálózatok esetén a legfontosabb előírás ezen a területen, hogy **minden, a belső állapotot meghatározó tároló elemnek, regiszternek legyen törlő (alaphelyzet beállító) jele.**

3. A project létrehozása

Az ISE program indításhoz kattintson az ISE Project Navigator ikonra, vagy a **Start** menüből indulva: **Xilinx Design Tools / 64-bit Project Navigator.**

A mintapélda projekt létrehozásához a **File** menüben válassza a **New Project** parancsot, és töltsé ki a párbeszédablakot az alábbi módon.

- A **Project Name** legyen *wpbev*. 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. Mivel az ISE sok fájlal dolgozik, a projectet mindig lokálisan tároljuk, ne az alaplabor szerverén!
- **Top-Level source type** mezőben a legördülő ablakból válassza a **HDL**-t.



A **Next** gombra kattintás után megjelenő **Device Properties** mezőben a **Value** oszlop legördülő listáiból (-- mely az értékekre kattintással jelenik meg --) válassza az alábbi értékeket:

Device Family: Spartan3E

Device: xc3s250E

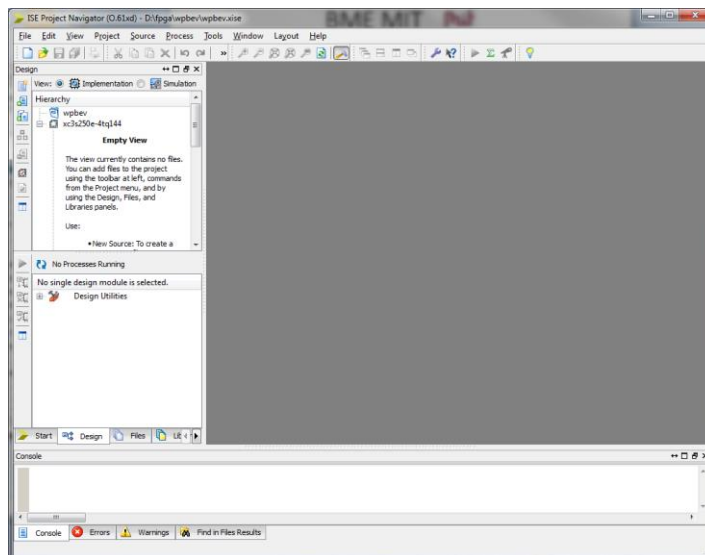
Package: tq144

Speed Grade: -4

Synthesis Tool: XST (VHDL/Verilog)

Simulator: ISim (VHDL/Verilog)

A **Next**, majd **Finish** gombra kattintva elkészül az üres project.



A Project Navigátor fő képernyőjén 4 ablak található:

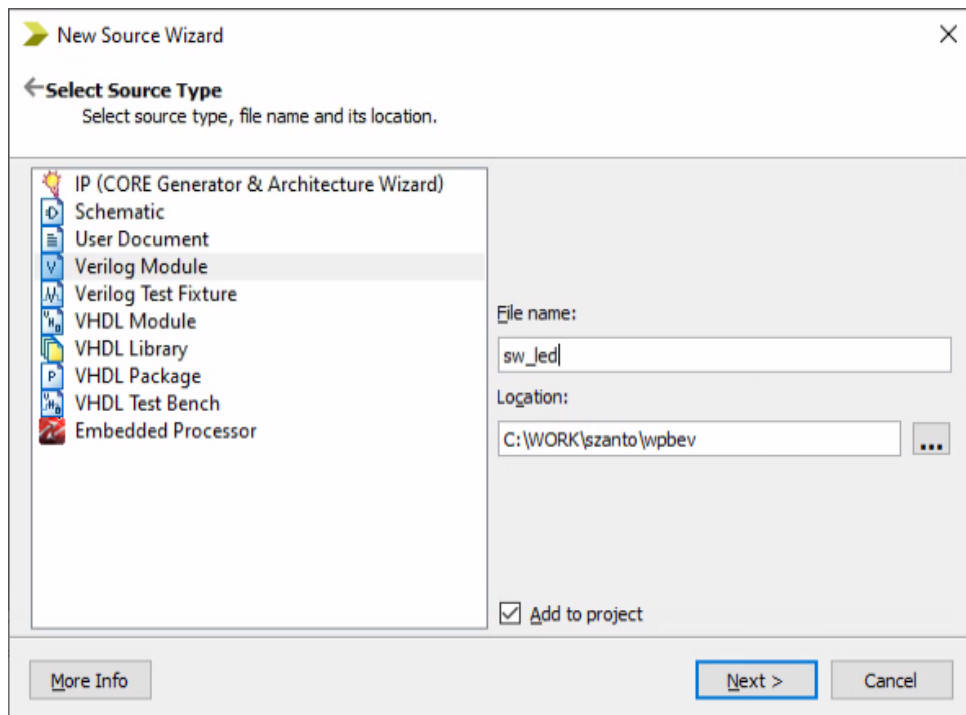
- Balra fent a **Hierarchy** (a projekt forrásai) ablak, amely megmutatja a projekthez tartozó tervezési fájlokat azok egymáshoz való kapcsolatával együtt. Itt választhatjuk ki azt is, hogy milyen típusú fájlokat szeretnénk látni: a **Implementation** opció csak az implementációra szánt modulok fájljait mutatja, míg a **Simulation** a szimulációhoz szükséges testbench-eket is.
- **Processes** ablak, mely azt mutatja meg, hogy a felette lévő **Hierarchy** ablakban kijelölt tervezési állományon milyen feldolgozások (processes) hajthatók végre.
- A fentiek mellett jobboldalt található az **Editor** ablak, melyben a különböző tervezési fájlokat megtekinthetjük és szerkeszthetjük.
- Legalul van a **Console** (üzenet) ablak, mely az éppen most futó ill. futott tervezési folyamat log-fájljait tartalmazza. Ennél az ablaknál négyféle nézet közül választhatunk a fülekkel. A **Console** nézetben a teljes üzeneteket látjuk. Nagyon hasznos a **Warnings** (figyelmeztetések) és az **Errors** (hibák) nézet, mely a hosszú log-fájlokból kigyűjti a figyelmeztetéseket ill. a hibaüzeneteket.

3. Kapcsolók állapotának megjelenítése a LED-eken

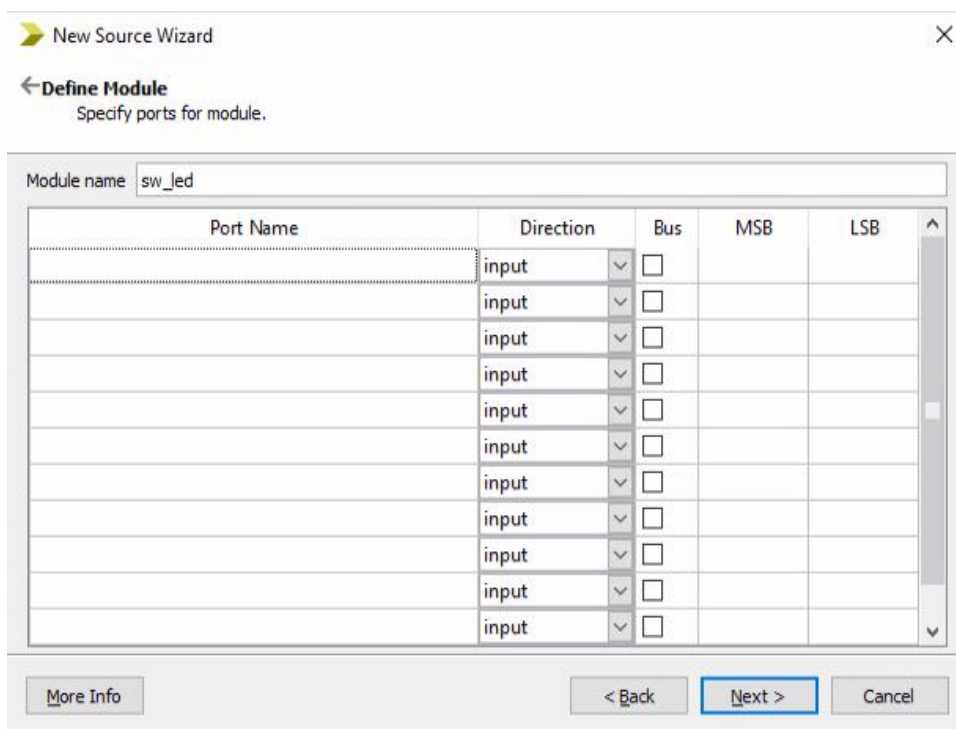
Legegyszerűbb példánkban a fejlesztői kártyán található 8 kapcsoló állapotát jelenítjük meg a 8 LED-en. Adott LED akkor fog világítani, ha a hozzátartozó kapcsoló fel van kapcsolva, azaz az FPGA felhasználásával lényegében létrehozunk egy jó drága vezetéket.

3.1 Verilog forráskód

Új modul létrehozásához a **Project** menüben kattintson a **New Source** parancsra, és válassza a **Verilog Module**-t forrásként. A létrehozandó modul neve legyen `sw_led`, amit a **File Name** ablakba kell beírni, és jelölje ki az **Add to Project** opciót. A **Next** gombra kattintás után a **Define Verilog Source** wizard felkínálja a portlista szerkesztését, amit most nem használunk, úgyhogy **Next**.



A port listát nem töltjük ki ebben a lépésben, majd közvetlenül a forráskóba írjuk bele.

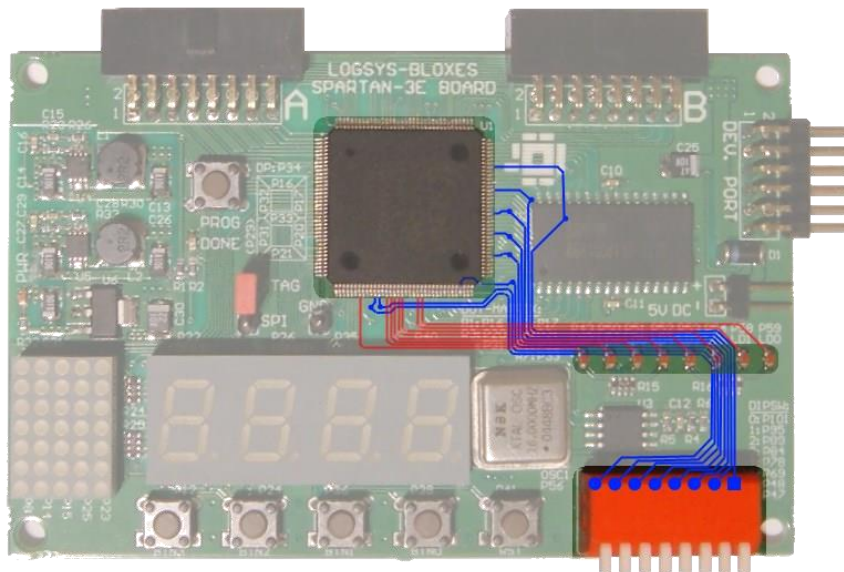


A modul portlistájában egy 8 bites bemenet (switch - sw) és egy 8 bites kimenet (LED - led) található. A modul törzse rem egyszerű, folytonos értékadással (assign) átadjuk a led kimenetnek az sw bemenet értékét.

```
module sw_led(
    input  [7:0] sw,
    output [7:0] led
);
    assign led = sw;
endmodule
```

3.2 Verilog port – FPGA láb hozzárendelés

A Verilog kódon túlmenően meg kell mondanunk a fejlesztői környezetnek, hogy a Verilog kódban megadott port-okat az FPGA mely lábára kell kötni. Ez a nyomtatott áramkörtől (NYÁK) függ, így a kártya dokumentációjában található meg, illetve a laboron használt kártya esetén rá van szitázva a NYÁK-ra. Szemléletesen:



A láb-hozzárendelések elvégzéséhez egy constraint fájlt adunk a projecthez. Válasszuk ki a (már ismerős) **Project / New Source** menüpontot, a felbukkanó ablakban pedig álljunk az **Implementation Constraint File**-ra, névnek pedig válasszuk a *pins*-t.

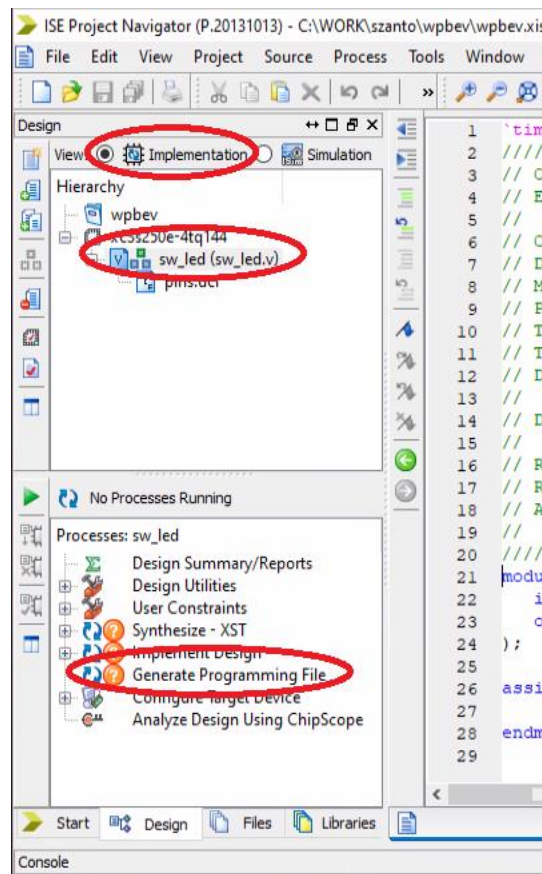
A megfelelő mennyiségű **Next/Finish** gomb megnyomása után a **Sources** ablakban meg is jelenik a *pins.ucf* fájl. Itt minden egyes port bithez (tehát a 8 bites jeleink esetében mind a nyolc bithez) meg kell adnunk a megfelelő lábat. Ha sikeresen leestük a panelről a használt lábak nevét, az alábbihoz egészen hasonló *ucf* fájlt kapunk. (Ugye? ☺)

```
NET "sw[0]" LOC="p101";
NET "sw[1]" LOC="p95";
NET "sw[2]" LOC="p89";
NET "sw[3]" LOC="p84";
NET "sw[4]" LOC="p78";
NET "sw[5]" LOC="p69";
NET "sw[6]" LOC="p48";
NET "sw[7]" LOC="p47";

NET "led[0]" LOC="p59";
NET "led[1]" LOC="p58";
NET "led[2]" LOC="p54";
NET "led[3]" LOC="p53";
NET "led[4]" LOC="p52";
NET "led[5]" LOC="p51";
NET "led[6]" LOC="p50";
NET "led[7]" LOC="p43";
```

3.3 Implementáció

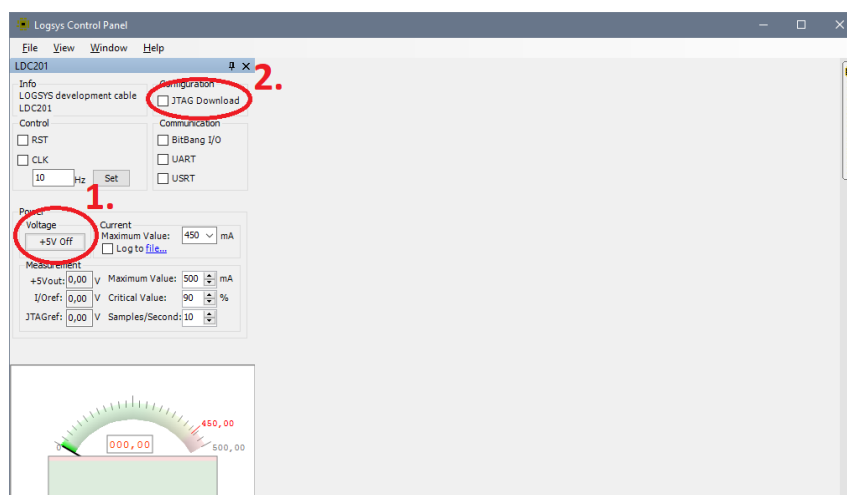
A realizáláshoz először szintetizálni kell a HDL leírást. A **Hierarchy** ablakban jelölje ki a Verilog modul-hierarchia legfelső szintű modulját (jelenleg csak egy modulunk van, úgyhogy azt)! Ezt követően a **Processes** ablakból indítsa el **Synthesize** eljárást. A szintézis végeztével meg kell valósítani a szintetizált terv leképezését az FPGA struktúrára (**Implement Design**), végül elő kell állítani az FPGA IC programozásához szükséges fájlt (**Generate Programming File**). Elegendő a **Generate Programming File** opcióra klikkelni, ha a megelőző implementációs lépések hiányoznak, akkor azokat automatikusan végrehajtja az ISE.



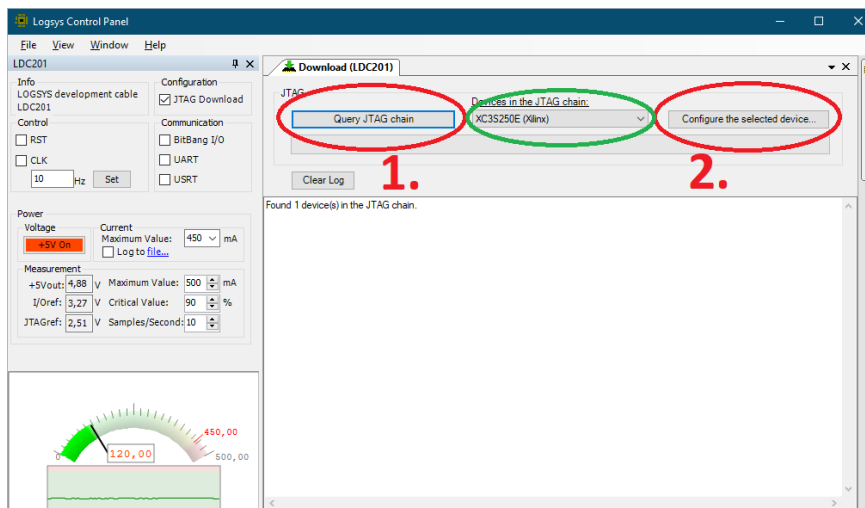
3.4 FPGA konfiguráció

A programozó fájl elkészülte után az FPGA programozása következik, beprogramozzuk az FPGA-ba a terv fordítása során generált hardver struktúrát. Ehhez a Logsys GUI alkalmazást használjuk.

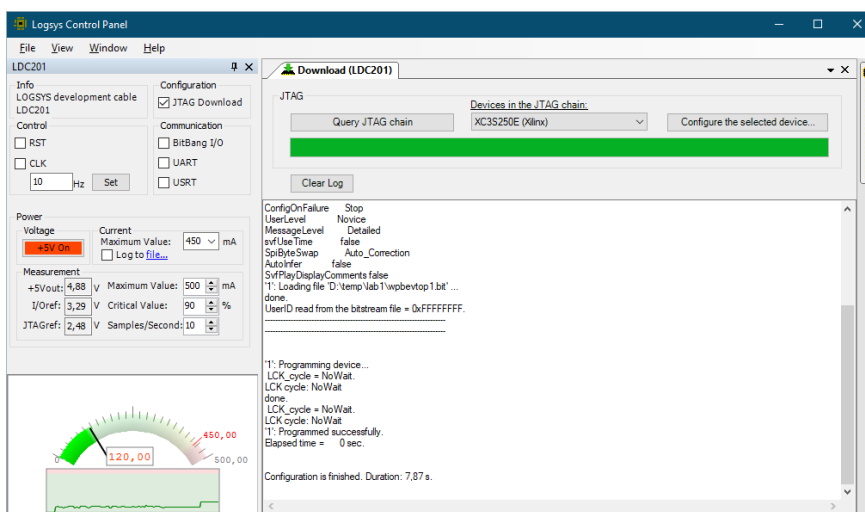
Amennyiben az FPGA kártya csatlakoztatva van a PC USB portjához, úgy indítás után az alábbi ablakban az első dolgunk a tápfeszültség bekapcsolása, majd a „JTAG Download” opció kiválasztása.



Ezután megjelenik a konfigurációs ablak, ahol előbb a „Query JTAG Chain” gombra kell klikkelni. Ekkor megtörténik az eszközök keresése, s a „Devices in the JTAG Chain” menüben megjelenik a megtalált XC3S250E típusú FPGA. A sikeres detektálást követően klikkeljünk a „Configure the selected device...” gombra, majd válasszuk ki az ISE által generált konfigurációs (.bit) fájlt.



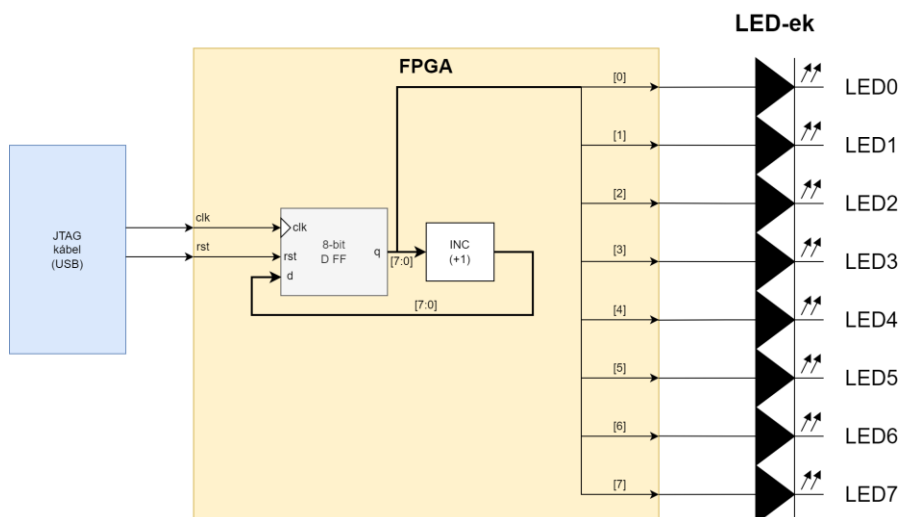
A sikeres konfigurációról a Log ablakban kapunk tájékoztatást.



A beprogramozás után ellenőrizze az eszköz működését, azaz nézze meg, hogy a kapcsolók állítgatására az egyes LED-ek ki/be kapcsolnak!

4. Bináris számláló megjelenítése a LED-eken

Második projektünkben már sorrendi hálózatot realizálunk. Ehhez először távolítsuk el a projektből az sw_led.v fájlt (jobb klikk a fájlra, majd „Remove”). A tervek blokkvázlata:



4.1 Verilog forráskód

Ezt követően az előző ponthoz hasonlóan a **Project** menüben a **New Source** parancsra kattintva, és kiválasztva a **Verilog Module**-t, hozzunk létre egy üres Verilog fájlt, aminek a neve legyen bin_cntr (binary counter).

Mivel 8 LED-ünk van, ezért egy 8 bites számlálót fogunk létrehozni. A 8 bites kimeneten túlmenően sorrendi hálózatról lévén szó szükségünk lesz egy órajel (clk) és egy alaphelyzetbe állító reset (rst) bemenetre. Port listánk tehát:

```
module bin_cntr(  
    input      clk,  
    input      rst,  
    output [7:0] led  
);
```

A sorrendi hálózat leírásához egy reg típusú változóra lesz szükségünk, ami órajel érzékeny always blokkban kap értéket. Funkciója egyszerű: reset hatására nullázzuk, egyébként pedig felfelé számolunk (a végérték – 255 – elérésekor a számláló túlsordul, s a következő érték 0 lesz).

```
reg [7:0] cntr;  
always @(posedge clk)  
    if (rst)  
        cntr <= 0;  
    else  
        cntr <= cntr + 1;  
assign led = cntr;
```

4.2 Port hozzárendelés

Az előző tervhez képest eltűnt a kapcsoló bemenet, valamint van két új portunk: az órajel és a reset. A használt fejlesztői kártyán utóbbiak származhatnak a JTAG fejlesztői kábeltől, vagy használhatjuk a kártyán levő reset és órajel forrásokat. Annak érdekében, hogy szemmel követni tudjuk a számláló működését, most alacsony frekvenciájú órajelre van szükségünk, amit a fejlesztői kábel tud szolgáltatni. Ennek megfelelően a port hozzárendelés:

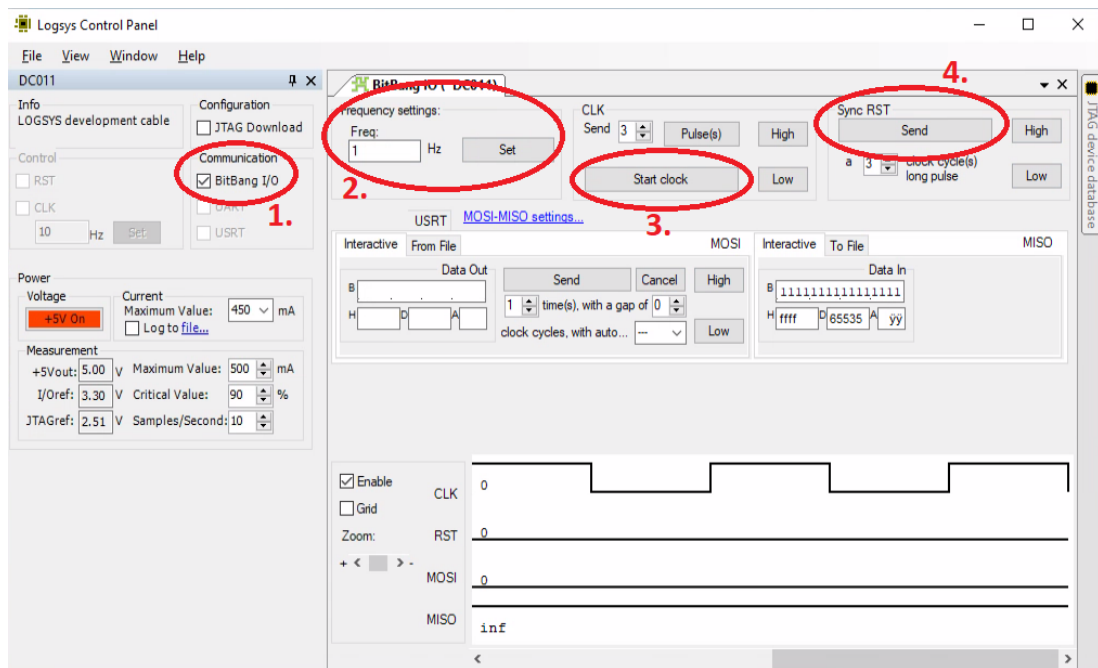
```
NET "clk" LOC="p129" | PULLDOWN;  
NET "rst" LOC="p119" | PULLDOWN;  
  
NET "led[0]" LOC="p59";  
NET "led[1]" LOC="p58";  
NET "led[2]" LOC="p54";  
NET "led[3]" LOC="p53";  
NET "led[4]" LOC="p52";  
NET "led[5]" LOC="p51";  
NET "led[6]" LOC="p50";  
NET "led[7]" LOC="p43";
```

4.3 Implementáció, programozás, tesztelés

Implementáljuk a tervet az ISE-ben a **Generate Programming File** opcióra kattikelve.

Ezután programozzuk fel az FPGA-t a most készített .bit fájlal.

A programozása után a Logsys GUI-ban válasszuk a „BitBang I/O” opciót, majd a megjelenő ablakban állítsunk be egy megfelelően alacsony frekvenciát (1..5 Hz), klikkeljünk a „Set” majd „Start clock” gombokra. Ezzel folyamatos órajelet generál a letöltőkábel. Amennyiben reset-elni szeretnénk a számlálót, klikkeljünk a Sync RST alatt található „Send” gombra.



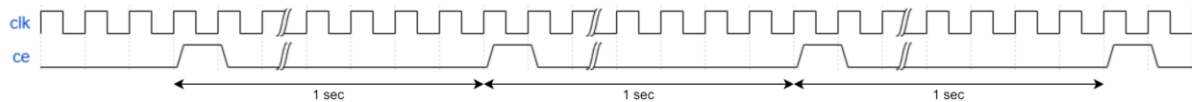
Ellenőrizzük a LED-eken a számláló megfelelő működését.

5. BCD számláló és engedélyező jel generátor

Harmadik feladatként egy egy-digites másodpercszámlálót valósítunk meg. A számláló az aktuális másodperc értékét BCD kódolással jeleníti meg a mérőpanel alsó 4 LED-jén, ebből következően ez a négy LED 0...9 értéket vehet fel. A számláló az SW0 kapcsoló állásától függően felfelé vagy lefelé számol.

A készüléket **szigorúan szinkron** sorrendi hálózattal fogjuk realizálni. Ez azt jelenti, hogy a készülék minden szekvenciális egységének ugyanaz az órajele, amit a blokkvázlaton rendszer-órajelnek (system clock) nevezünk. Az előző feladattal ellentétben ez most a kártyán található 16 MHz frekvenciájú oszcillátor lesz.

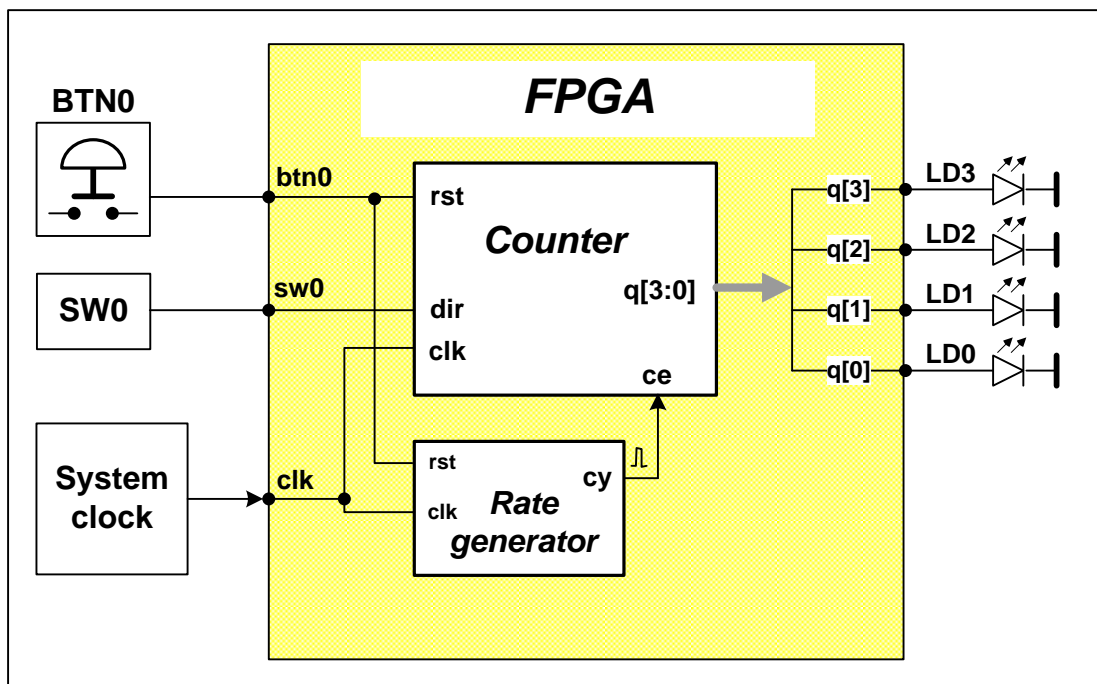
Azt, hogy a számláló csak másodpercenként lépjen, úgy valósítjuk meg, hogy másodpercenként egyszer 1 darab, egyetlen rendszer-órajel szélességű impulzust adunk a számláló *ce* (clock enable) engedélyező bemenetére. Ezt az engedélyező impulzust egy ütemgenerátorral (Rate generator) állítjuk elő.



Megj.: Elvileg választhatnánk azt a megoldást is, hogy az 16 MHz-es külső órajelből generálunk egy 1/másodperc frekvenciájú jelet, amit aztán ténylegesen órajelként alkalmazunk (azaz a flip-flop-ok órajel bemenetére kötjük). Ez a megoldás azonban itt most nem részletezett okok miatt FPGA-ban nem javasolt. A Laboratórium 1. tárgy során követelmény, hogy az összes flip-flop a külső órajelről működjön! A kevésbé gyakori eseményeket a most bemutatott módon, engedélyező jelekkel valósíthatjuk meg.

A hálózatot természetesen egy alaphelyzet-beállító jellel (*rst*) is ellátjuk, amit az egyik nyomógombhoz (BTN0) rendelünk.

A tervezés során először a készülék funkcionális egységeit (moduljait) fogjuk megtervezni, majd az egységeket a hierarchikus terv legfelső szintjén lévő Verilog modulban fogjuk összekapcsolni.



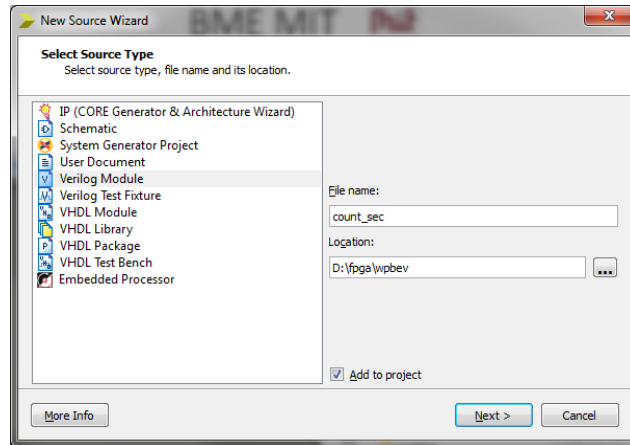
2. ábra A megtervezendő készülék funkcionális vázlata

Távolítsa el a projektből az előző pontban használt Verilog és UCF fájlokat: jobb klikk a fájlon, majd „Remove”!

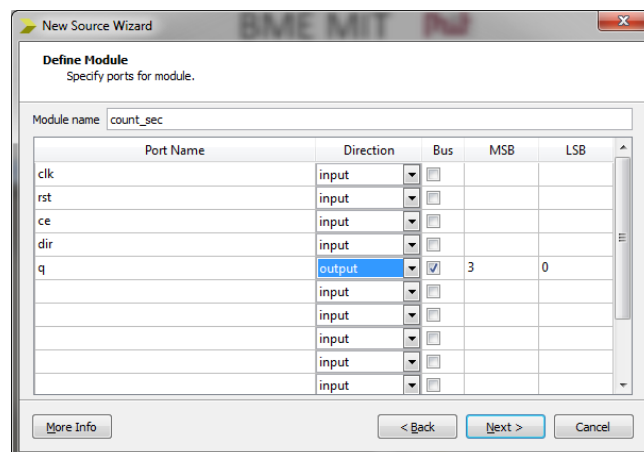
5.1. A számláló modul leírásának elkészítése

5.1.1 A számláló modul keretének létrehozása

Hozzon létre egy új Verilog modult: a **Project** menüben kattintson a **New Source** parancsra, és válassza a **Verilog Module**-t forrásként. A létrehozandó modul neve legyen *count_sec*, amit a **File Name** ablakba kell beírni, és jelölje ki az **Add to Project** opciót. A **Next** gombra kattintás után a **Define Verilog Source** wizard felkínálja a portlista szerkesztését, amit az eddigiekkel ellentétben most használunk is.



Vegyük elő a blokkvázlatot, és annak alapján kezdjük el a bemenetek majd a kimenetek beírását. A *q* jel busz-kimenet, ezért a **Direction** oszlop megfelelő sorára kattintva a legördülő listából válasszuk az *output* irányt, és az MSB ablakba pedig írunk 3-at a négy LED vezérléséhez.



Next majd **Finish** klikk után az editor ablakban megjelenik a modul kerete a portlistával és a jel-deklarációkkal:

```
module count_sec (
    input clk,
    input rst,
    input ce,
    input dir,
    output [3:0] q
);

endmodule
```

5.1.2 Modul működési leírásának létrehozása sablon segítségével

Az eddigiekben üres Verilog kódból indultunk ki, de érdemes megjegyezni, hogy az ISE fejlesztő rendszer sok funkcionális elem HDL nyelvű leírási vázát tartalmazza, amit itt sablonnak (template) neveznek. A számláló működési leírását (a modul törzsét) készítsük el ilyen sablon segítségével.

Az **Edit** menüben válassza a **Language Templates** parancsot, és a megjelenő ablakban válassza ki a **Verilog \ Synthesis Constructs \ Coding Examples \ Counters \ Binary \ Up/Down Counters** mappából a **w_CE_and_Sync_Active_High_Reset** (with Count Enable and ...) állományt, és az editor ablakban megjelenő leírás-sablont másolja át a HDL editorban megnyitott *count_sec.v* állományba, a modul fejléce alá.

A sablonban nem valódi, hanem funkcióra utaló általános jelnevek szerepelnek, mint pl. *<reg_name>*, *<up_down>*. Ezeket természetesen át kell írni a portlistában szereplő valódi jelneveknek megfelelően: *<reg_name>* helyet *q*, *<up_down>* helyet *dir* és így tovább. Ezután a modul-leírás az alábbiak szerinti lesz.

```
module count_sec(
    input clk,
    input rst,
    input ce,
    input dir,
    output [3:0] q);

    reg [3:0] cntr;
    always @(posedge clk)
    if (rst)
        cntr <= 0;
    else if (ce)
        if (dir)
            cntr <= cntr + 1;
        else
            cntr <= cntr - 1;

    assign q = cntr;

endmodule
```

Értelmezze az always utasítás sorait!

Megjegyzés: A **Synthesis Constructs** mappa tulajdonképpen egy példatár, érdemes körülnézni benne.

5.1.3 A sablon szerinti leírás módosítása

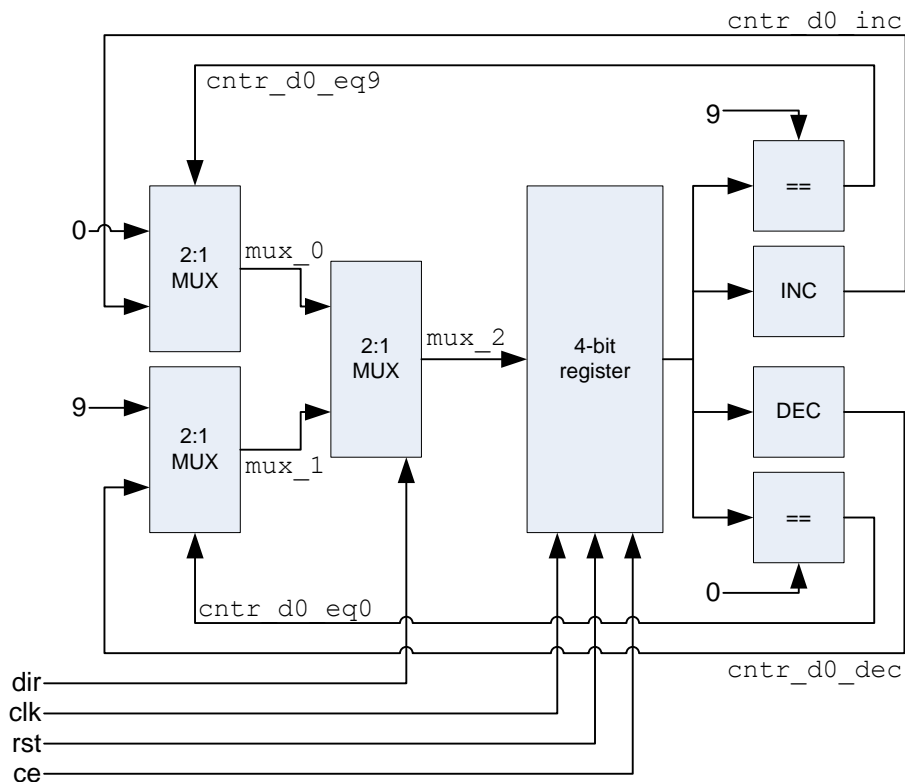
A sablon szerinti működés nem felel meg pontosan az elképzeléseinknek, ezért azt megfelelően módosítani fogjuk. Gondoljuk át, hogy mi a számunkra szükséges funkcionalitás, kezdjük az alacsonyabb helyiérték megvalósításával (és ebből kifolyólag a fenti kód *cntr* változóját a további funkcionális bővítésre való tekintettel nevezzük át (*cntr_d0*-ra).

A fel/le számlálást egy 4 bites regiszter (4 darab D flip-flop, mely rendelkezik reset és órajel engedélyező bemenetekkel is) illetve a regiszter kimenetét használó kombinációs logika valósítja meg; utóbbi állítja elő a regiszter kimenet egyel növelt és csökkentett értékét (INC illetve DEC blokk). A számláló aktuális értékét tároló regiszter tartalmát minden órajel ütemben frissítjük, amikor a külső engedélyező jel (*ce*) '1' értékű és a külső reset inaktív (*rst* = '0').

A regiszter bemenetére az aktuális vezérlőjelek alapján a következő értékek kerülhetnek:

- felfelé számlálás (*dir* == 1)
 - a számláló elérte a végértékét (9): 0
 - nem érte el a végértékét: aktuális érték + 1
- lefelé számlálás (*dir* == 0)
 - a számláló elérte a végértékét (0): 9
 - nem érte el a végértékét: aktuális érték - 1

A végértékek detektálásához két egyenlőség komparátort használhatunk, melyek a számláló regiszterének kimenetét hasonlítják össze 9-cel illetve 0-val. Ezen megfontolások alapján az alábbi blokkvázlat adódik:



Első Verilog leírásunkban ragaszkodjunk a blokkvázlatnak tökéletesen megfelelő struktúrához. A belső jelek használatához ezeket deklarálnunk kell: a sorrendi hálózatot megvalósító változó (regiszter) *reg* típusú, míg kombinációs logikát realizáló változók legyenek *wire* típusúak.

```
reg [3:0] cntr_d0;
wire [3:0] cntr_d0_inc, cntr_d0_dec;
wire [3:0] mux_0, mux_1, mux_2;
wire cntr_d0_eq0, cntr_d0_eq9;

always @(posedge clk)
if (rst)
    cntr_d0 <= 0;
else if (ce)
    cntr_d0 <= mux_2;

assign cntr_d0_inc = cntr_d0 + 1;
assign cntr_d0_dec = cntr_d0 - 1;
assign cntr_d0_eq0 = (cntr_d0 == 0);
assign cntr_d0_eq9 = (cntr_d0 == 9);

assign mux_0 = (cntr_d0_eq9) ? 0 : cntr_d0_inc;
assign mux_1 = (cntr_d0_eq0) ? 9 : cntr_d0_dec;
assign mux_2 = (dir) ? mux_0 : mux_1;
```

Az egyetlen *always* blokkunk érzékenységi listájában az órajel felfutó éle található, így a benne levő értékadások csak ekkor értékelődnek ki, azaz az itt írt változóból ténylegesen D FF-ok realizálódnak. A létrejövő D FF-ok tartalma azonban csak akkor módosul, ha a külső *ce* jel '1' értékű. A többi változónk *wire* típusú, így ezeknek csak az *assign* utasítással adhatunk értéket, implementáció után pedig kombinációs logikát eredményeznek.

A Verilog ugyanakkor viszonylag magas szintű nyelv, így lehetőség van a fenti leírással funkcionálisan megegyező, de könnyebben értelmezhető és tömörebb kód írására is:

```
reg [3:0] cntr_d0;
wire cntr_d0_eq0, cntr_d0_eq9;

always @(posedge clk)
if (rst)
    cntr_d0 <= 0;
else if (ce)
```

```

if (dir)                //DIR=1: count up
    if (cntr_d0==9)
        cntr_d0 <= 0; //overflow
    else
        cntr_d0 <= cntr_d0 + 1;
else                    //DIR=0: count down
    if (cntr_d0==0)
        cntr_d0 <= 9;
    else
        cntr_d0 <= cntr_d0 - 1;

```

5.1.4 A modul-leírás szintaktikai ellenőrzése

Miután gondosan átnéztük a forrás-állományt (pl. nem hiányzik-e helyenként utasítást lezáró pontosvessző), a fejlesztő rendszerrel is végeztethetünk szintaktikai ellenőrzést. Jelöljük ki az ellenőrzendő modult a **Project** (és amennyiben több modulunk van, állítsuk be ezt top modulnak: jobb klikk és **Set as Top Module**) ablakban, majd a **Processes** ablakban egy kettős kattintással indítsuk el a **Synthesize-XST / Check Syntax** alkalmazást. A vizsgálat eredménye az alsó üzenet-ablakban jelenik meg, ahol a megfelelő fülekkel külön megtekinthetjük a hibaüzeneteket (Errors) és figyelmeztetéseket (Warnings).

Egyébként az ISE alrendszerei minden további feldolgozást a kiinduló állomány ellenőrzésével kezdenek, így a következő szakaszban leírt szimuláció-indítással automatikusan egy ellenőrzés is végrehajtódik.

5.2. A modul ellenőrzése szimulációval

5.2.1 Az ellenőrzési környezet létrehozása

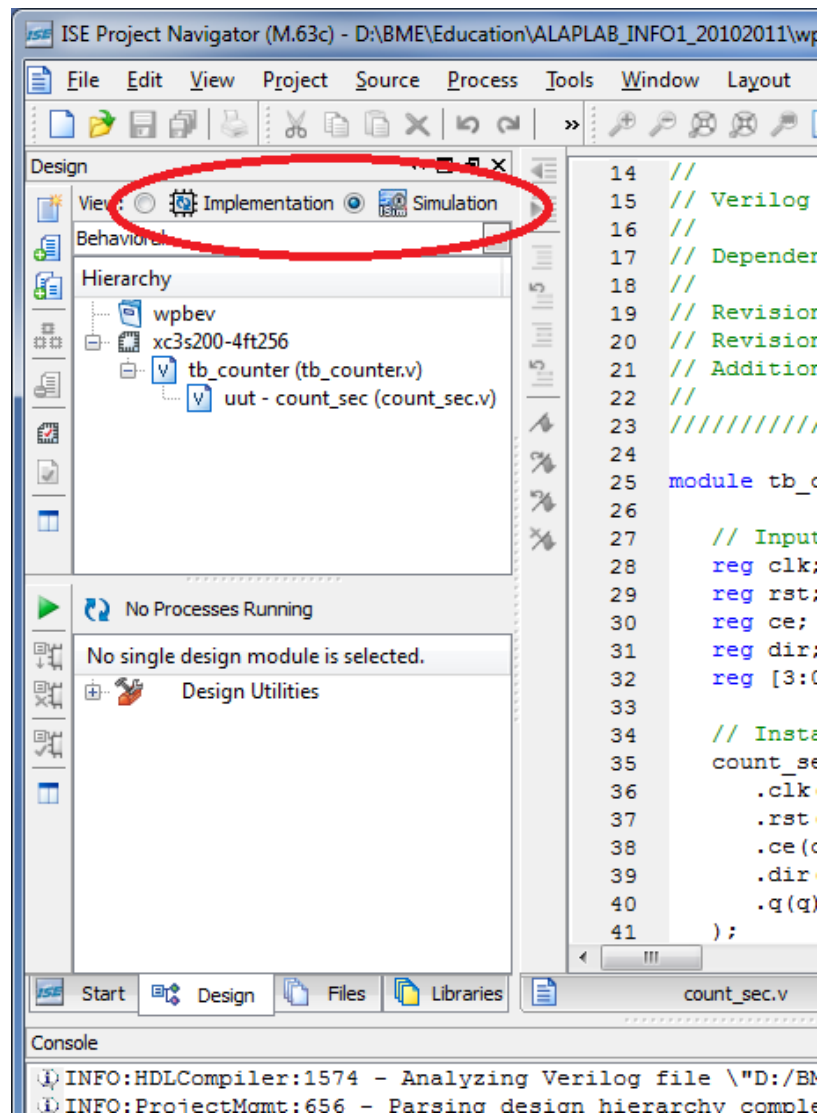
Az ellenőrzési környezet (test fixture, tesztelési környezet) létrehozása az egységet működtető jelformák (gerjesztő jelek) megadásával történik. A tesztelési környezet elnevezéssel kapcsolatban megjegyezzük, hogy itt a tesztelés helyett verifikálásról kellene beszélni, mert azt ellenőrizzük, hogy a tervezett berendezés a specifikációnak megfelelően fog-e működni.

Ahhoz, hogy az elkészített, implementációra szánt tervet ellenőrizni tudjuk, létre kell hoznunk egy környezetet, amely a tesztelendő rendszer bemeneteit megfelelően gerjeszti (test fixture). Régebbi ISE verziókban lehetőség volt a gerjesztőjelek grafikus felületen történő megadására, az újabb verziókból azonban kikerült ez a funkció, így az egyetlen lehetőség a HDL nyelvű leírás.

Első lépésként adjunk hozzá a project-hez egy új forrást: **Project / New Source** ablakban válassza a **Verilog Test Fixture** opciót. A fájl neve legyen **tb_counter**!

A következő ablakban válasszuk ki azt a modult, amelyhez a testbench-et generáljuk – jelen esetben egyetlen modulunk van – count_sec –, így értelemszerűen ez az egyetlen választási lehetőség.

Next, Finish kattintásokkal hagyjuk jóvá a file generálását, így visszajutunk az ISE főablakába. A bal felső **View** résznél **Simulation**-t választva a **Hierarchy** ablakban megjelenik a testbench file is, aminek a tesztelendő modul egy almodulja (a hierarchia a modul neve előtti + jellel bontható ki).



5.2.2 Gerjesztőjelek létrehozása

Az automatikusan generált Verilog Test Fixture file az alábbiakat tartalmazza:

- a tesztelendő modul példányosítása
- a bemeneti jelek esetén *reg* típusú változók deklarálása
- a kimeneti jelek esetén *wire* típusú változók deklarálása
- az összes bemeneti változó 0-ba állítása

A létrehozott Verilog kód (a fentiekben elmondottak alapján ennek értelmezését az olvasóra bízunk):

```
`timescale 1ns / 1ps
module tb_conter;

// Inputs
reg clk;
reg rst;
reg ce;
reg dir;

// Outputs
wire [3:0] q;

// Instantiate the Unit Under Test (UUT)
count_sec uut (
```

```

        .clk(clk),
        .rst(rst),
        .ce(ce),
        .dir(dir),
        .q(q)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        rst = 0;
        ce = 0;
        dir = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

    end

endmodule

```

Mint minden sorrendi hálózathoz, a tesztelendő számlálóhoz is szükséges egy órajel előállítása. A szimuláció 0. pillanatában a *clk* változó értéke 0 (lásd: fenti *initial* blokk). Az órajel nem más, mint egy 50%-os kitöltési tényezőjű négyszögjel; ez generálható pl. úgy, hogy adott időegységenként invertáljuk az aktuális értéket. Egy lehetséges Verilog kód (megj.: az *always* blokk a fenti *initial* blokkokon kívül van!):

```

always #5
    clk <= ~clk;

```

Mivel a *timescale* direktívában a beállított időegység ns, így a generált négyszögjel alacsony és magas állapota is 5 ns hosszúságú, tehát a periódusideje $2 \cdot 5 = 10$ ns (100 MHz frekvencia). Ennek viselkedési szimulációnál nincs jelentősége, az FPGA belső időzítéseit figyelembevevő implementáció utáni szimuláció esetén azonban már fontos a megfelelő frekvenciájú órajel használata.

A *rst*, *ce* és *dir* vezérlőjeleket generáljuk az alábbi módon:

- a *rst* legyen '1' értékű a szimuláció 7 – 27 ns alatt
- a *ce* legyen '1' értékű 107 ns után
- a *dir* váltson '1'-be 1007 ns után

Ezt az alábbi Verilog kóddal generálhatjuk

```

initial
begin
    #7 rst <= 1;
    #20 rst <= 0;
end

initial
    #107 ce <= 1;

initial
    #1007 dir <= 1;

```

Megjegyzések:

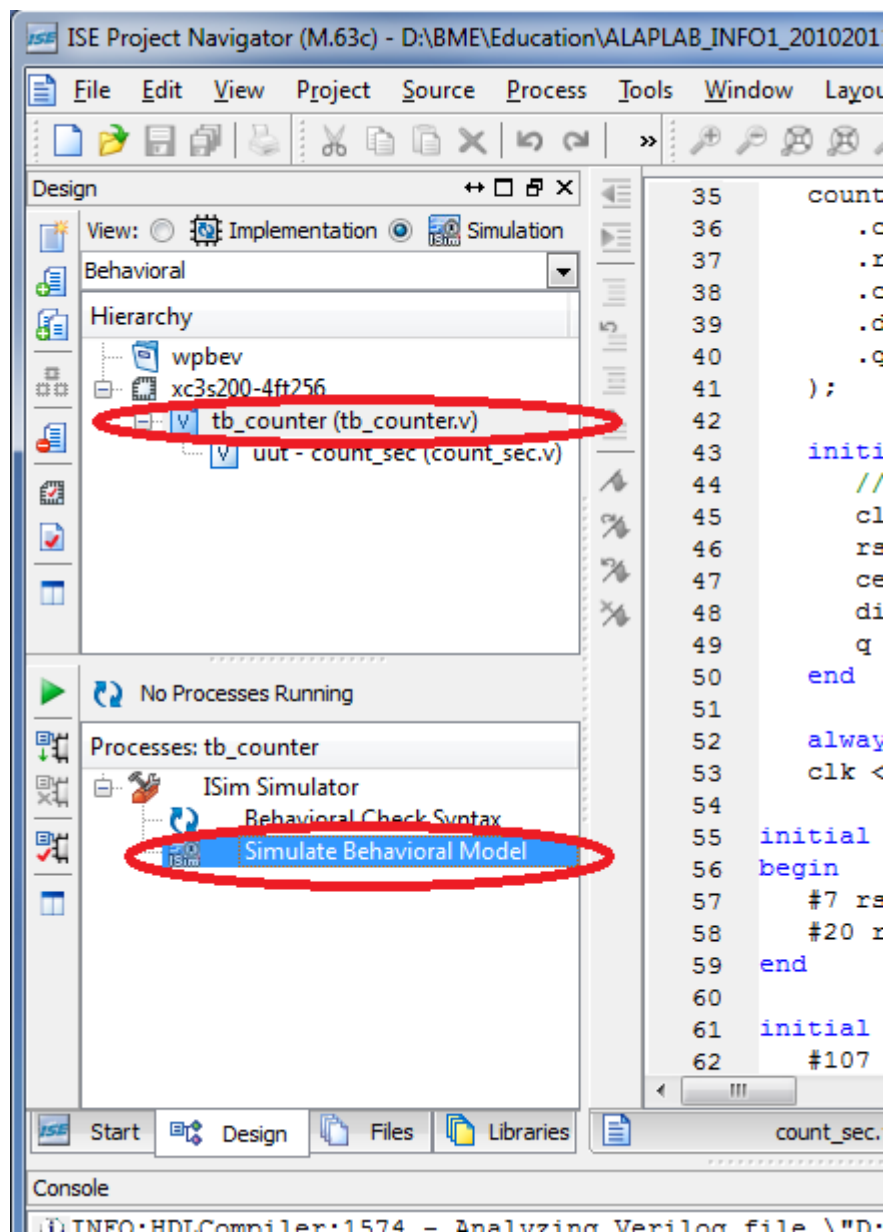
- egy *initial* blokkon belül a késleltetések összeadódnak
- az *initial* blokkok futása párhuzamosan történik, mindegyik a 0. időpillanatban indul

5.2.3 Funkcionális szimuláció az ISE Simulator segítségével

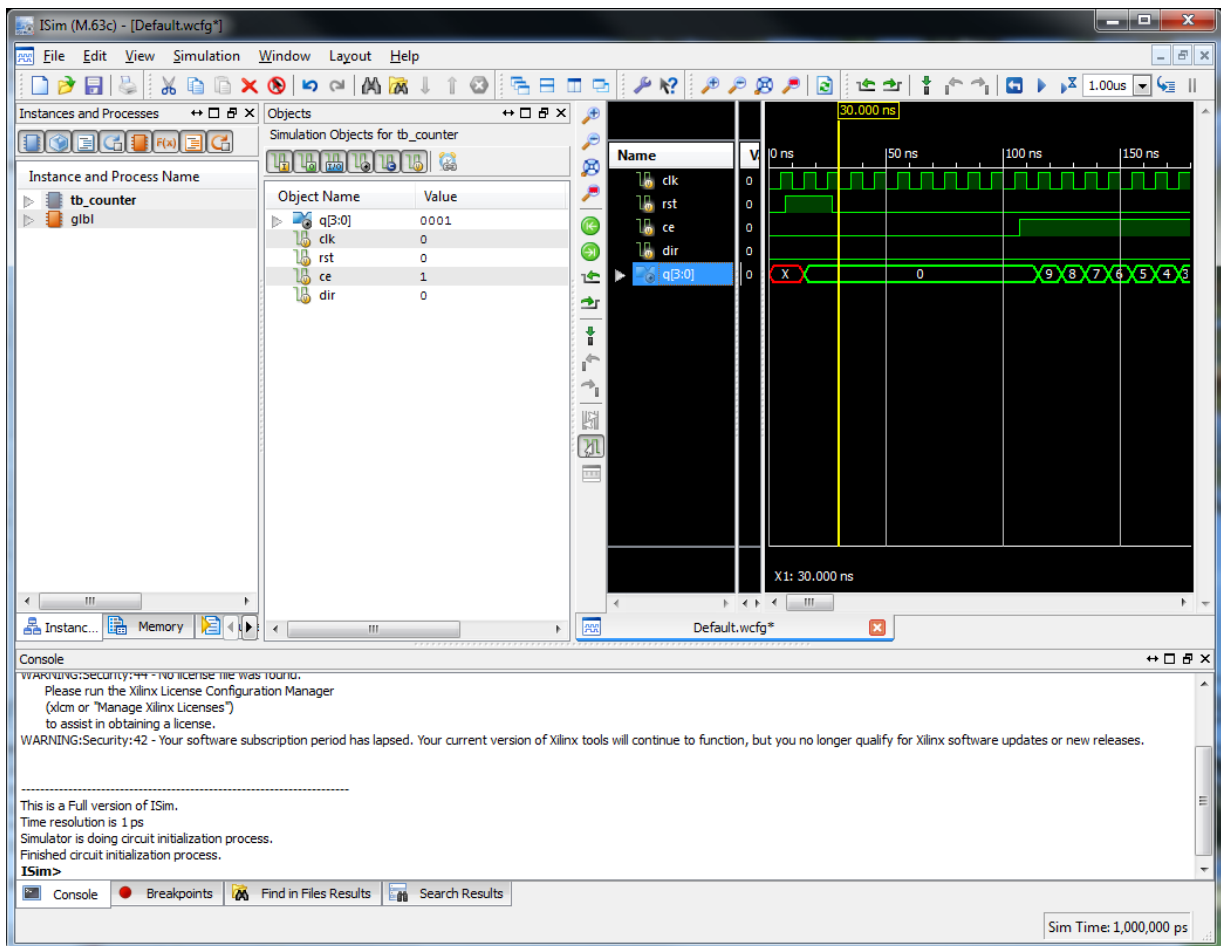
Ennél a tervezési lépésnél végzett szimulációval tulajdonképpen csak a funkcionalitást ellenőrizzük. Az ISE Simulator az elemkésleltetéseket is figyelembe tudja venni, de ennél a szintnél a terv még nincs leképezve egy valódi IC típusra, ezért nem tudunk valódi késleltetési értékekkel számolni.

A **Project Navigator** program **View** opciói közül válassza ki a **Simulation**-t, majd a **Hierarchy** ablakban jelölje ki a testbench fájlt (*tb_counter*). A **Processes** ablakban indítsa el az **ISim Simulator** / **Simulate Behavioral Model** programot.

FIGYLEM: Szimulációhoz a **Hierarchy** ablakban mindig a testbench file-t kell kiválasztani!! A szimulátor akkor is elindítható, ha az implementációra szánt modult választjuk ki, de ebben az esetben a bemeneti jeleket semmi nem fogja meghajtani, azaz a modul nem fog működni (ebben az esetben a szimuláció hullámforma ablakában az összes bemeneti jel nagyimpedanciás (kék), míg a kimeneti jelek nem definiáltak (piros)).



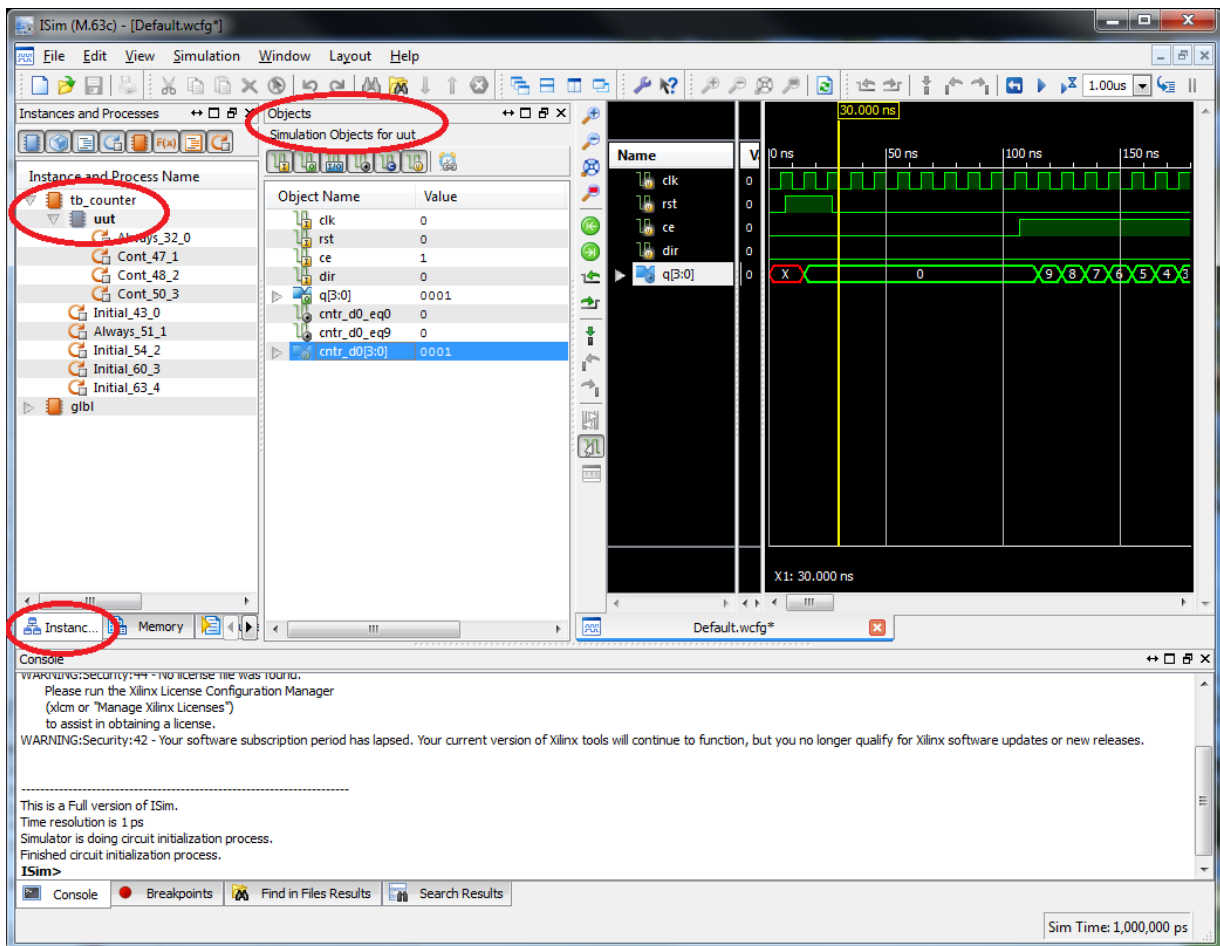
A szimuláció eredménye egy új ablakban fog megjelenni (5. ábra). Érdemes megfigyelni, hogy a *rst* jel aktiválását megelőzően a tároló elemek (regiszterek) értékét a szimulátor – helyesen - határozatlannak (unknown) tekinti, amit X értékkel jelez (piros szín).



5. ábra A szimuláció eredménye a Xilinx ISE Simulator hullámforma ablakában

Az egyes jelnevekre kattintva lehetőség van a kijelzés formátumát megváltoztatni. A q[3:0] kimenet esetén ezt hexadecimálisra állítva máris könnyebben ellenőrizhető a helyes működés.

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 bal oldali ablakban az **Instances and Processes** fülre kattikelve megjelenik a szimulációs hierarchia, amelyben megtalálható a példányosított *count_sec* modul (UUT – unit under test – néven). Kiválasztva ezt a modult, majd az **Objects** fülre kattikelve látjuk a modul összes jelét, melyek közül a vizsgálni kívánt jel egyszerű drag-and-drop módszerrel adható hozzá a **Wave** ablakhoz (6. 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).



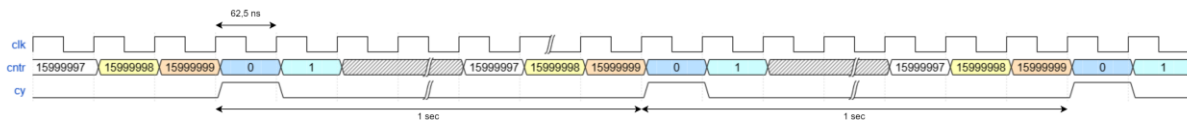
6. ábra Szimuláció belső jelekkel

5.3. A készülék további funkcionális egységeinek megtervezése

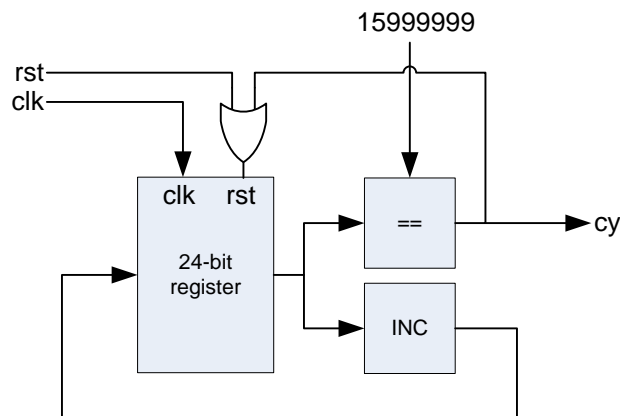
5.3.1 Az ütemgenerátor modul

A modul neve legyen röviden "rategen". A **Project / New Source** paranccsal egy Verilog Module típusú fájlt generálunk, melynek neve rategen, majd a portlista táblázatot üresen hagyva **Next** és **Finish** gombot nyomunk.

Az ütemgenerátor lényegében egy számláló, mely az FPGA mérőpanel 16 MHz-es órajelét 16 millióval osztja le (azaz 0-tól 15999999-ig számol), és számlálási periodusonként egyszer generál egy egy órajel hosszúságú pulzust (cy). Ekkora osztáshoz egy legalább 24 bites bináris számlálóra van szükség. Mivel nagyon kreatívak vagyunk, a számláló neve megint legyen cntr, amit regiszterként kell deklarálni (**reg** [23:0] cntr;). Időzítési digramon:



A megvalósítandó egység blokkvázlata tehát:



Az ezt megvalósító Verilog kód:

```
module rategen(
    input clk, rst,
    output cy
);
    //Generate 1 clock wide pulse on output CY
    reg [23:0] cntr;

    always @(posedge clk)
    begin
        if (rst | cy)
            cntr <= 0;
        else
            cntr <= cntr + 1;
    end

    assign cy = (cntr == 15999999);
    //assign cy = (cntr == 4);

endmodule
```

A modul-leírásban jelenleg "kikommentezve" egy másik osztásarány is fel van tüntetve. Ezt az osztásarányt a terv szimulációval történő ellenőrzésénél használhatjuk. Az "üzemi" osztásaránynál a modul kimenetén csak minden 16 milliomodik szimulációs lépésnél lenne változás, ami igen hosszú szimulációs futási időkhöz vezet.

5.3.2 A legfelső szintű modul és felhasználói megkötések létrehozása

A legfelső szintű modul írja le a funkcionális egységek kapcsolódásait és az egész készülék kapcsolatát a külvilággal. A legfelső modulban általában sok ki- és bemenet és jó néhány funkcionális modul szerepel. Az ebből következően nagyszámú jelnevet körültekintően kell megadni.

A **Project / New Source / Verilog module** paranccsal hozzunk létre egy *wpbevtop1* modul-keretet.

A top modul leírásának elkészítéséhez vegyünk elő ismét a készülék funkcionális tömbvázlatát (2. ábra), amelynek alapján a tervezést elkezdtük. A top modul lényegében ezt a tömbvázlatot írja le.

Az áttekinthetőség és a könnyebb megértés érdekében ezt a vázlatot egy kissé átrajzoltuk (7. ábra), a modul-leírás szempontjait figyelembe véve.

A top modul be- és kimeneteit tulajdonképpen tetszés szerint nevezhetjük el, de az elnevezésnek összhangban kell lennie a felhasználói "pin"-megkötésekben (az FPGA egyes lábaira milyen jelek kapcsolódjanak) szereplő elnevezésekkel. A laborban használt fejlesztői kártya összes I/O lábát definiálja az ehhez tartozó UCF file, ami a dokumentáció 15. oldalán található. Jelen esetben azonban ne ezt használjuk, hanem készítsünk saját ucf filet! Minden elem (kvarc, gombok, kapcsolók) mellett a kártyán megtaláljuk, az az FPGA mely lábára csatlakozik.

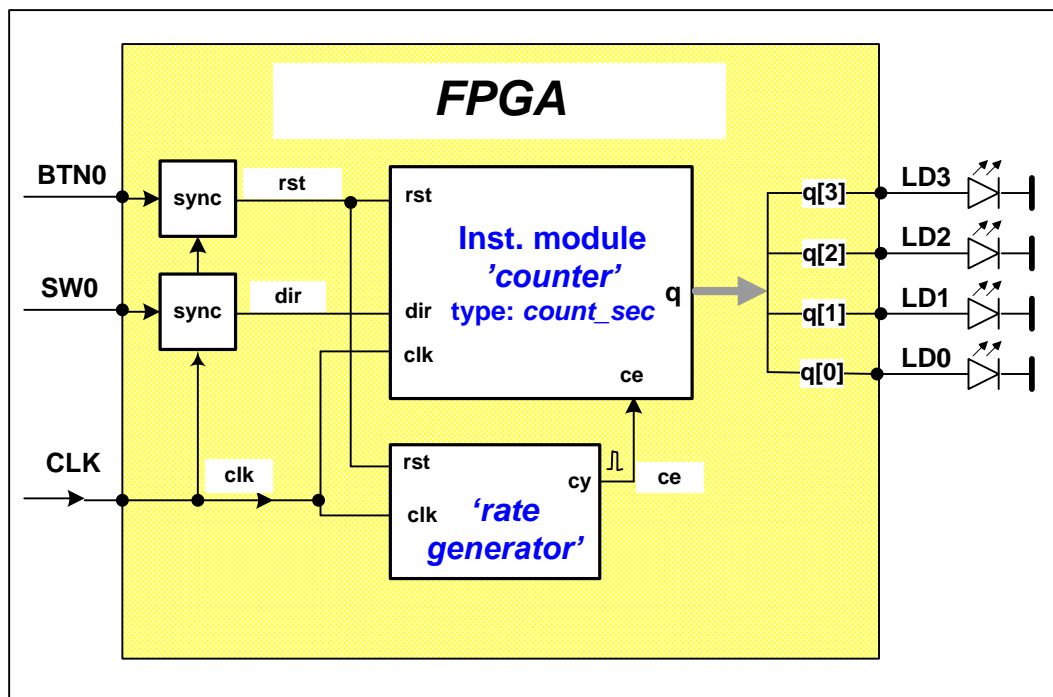
Vegyünk elő a tömbvázlatot, és sorban írjuk át a jelneveket az input/output listába.

```
module wpbevtop1(
    input clk, btn0, sw0,
    output [3:0] q
```

```
);
```

Mint a bevezetőben már említettük, korrekt tervezésnél a szinkron hálózat elsődleges (kívülről érkező) bemenő jeleit szinkronizálni kell, hogy elkerüljük a bemenő jel aszinkronitása okozta metastabil állapot felléptét a rendszerben. Ebben a feladatban a szinkronizáló hálózatot nem érdemes külön modulként leírni, mert az áttekintést az nem javítja. Egyszerű, egyfokozatú szinkronizáló hálózatot választunk, melyet egy, a top modulban elhelyezett *always* utasítással írunk le.

```
reg rst, dir;
always @(posedge clk)
//Synchronize inputs
begin
    rst <= btn0;
    dir <= sw0;
end
```



7. ábra A top modul vázlata

Ezután következzen a modulok példányosítása (behelyezése). A példányosításnál meg kell adni a top modulban aktuális jelneveket is (azaz a top modulban érvényes jeleket „rá kell kötni” a példányosított modul portjaira). Ha egy modul csak egyszer fordul elő, (ill. többszöri példányosítás esetén az egyik példánynál) az aktuális jelnév lehet a modul-definícióban szereplő jelnév is. Ezt a mintapéldánkban többször is alkalmazzuk. Egy jelnév tulajdonképpen egy összekötő vezetéket (wire) is jelent. **Figyeljünk arra, hogy a Verilog a nem deklarált jeleket 1 bites wire típusúnak tekinti (ez a helyzet akkor is, ha elgépeljük a változó nevét).** Mivel a jel így automatikusan deklaráródik, számos esetben hibaüzenetet (Error) nem kapunk, csak a működés nem megfelelő. A Warning-ok között viszont felfedezhető a hiba. Ha két modul valamely portjainak aktuális jelneve azonos, akkor az a két kivezetés össze van kötve. Ez vonatkozik a top modul egyéb elemeire is.

Ha két pontot össze akarunk kötni, melyekhez eltérő aktuális jelnév tartozik, akkor az összekötést az *assign* utasítással tudjuk megvalósítani.

A példányosítást kezdjük az 1 másodperces engedélyező jelünket generáló rategen modullal. Induljunk ki a modul deklarációjából:

```
module rategen(
    input    clk, rst,
    output   cy
);
```

Először is töröljük ki a module kulcszót, majd a modul típusa után adjuk meg a példány nevét. Végül pedig csatlakoztassuk a példány portjaihoz a top modul megfelelő jeleit.


```
wire ce;
rategen rategenerator(
    .clk(clk),
    .rst(rst),
    .cy(ce)
);
```

Lehetőség van egy alternatív port-jel hozzárendelésre is, amikor a modul deklarációnak megfelelő sorrendben felsoroljuk a portokhoz kapcsolódó jeleket. Mivel ez sorrend alapú hozzárendelés (azaz könnyebb elrontani), így használata **nem** javasolt. Az engedélyező jel generátorunkra ez a következőképp nézne ki.

```
rategen rategenerator(clk, rst, ce);
```

Folytassuk a másodpercszámlálónk beillesztésével.

```
count_sec counter(
    .clk(clk),
    .rst(rst),
    .ce(ce),
    .dir(dir),
    .q(q)
);
```

A bemeneti szinkronizációval és az almodulok beillesztésével tehát a teljes top level modulunk a következőképp néz ki.

```
module wpbevtop1(
    input clk, btn0, sw0,
    output [3:0] q
);

reg rst, dir;
always @(posedge clk)
//Synchronize inputs
begin
    rst <= btn0;
    dir <= sw0;
end

wire ce;
rategen rategenerator(
    .clk(clk),
    .rst(rst),
    .cy(ce)
);

count_sec counter(
    .clk(clk),
    .rst(rst),
    .ce(ce),
    .dir(dir),
    .q(q)
);

endmodule
```

A láb-hozzárendelések elvégzéséhez egy constraint fájlt adunk a projecthez. Válasszuk ki a **Project / New Source** menüpontot, a felbukkanó ablakban pedig álljunk az **Implementation Constraint File**-ra, névnek pedig válasszuk a *counter_pins*-t.

A megfelelő mennyiségű **Next/Finish** gomb megnyomása után a **Sources** ablakban meg is jelenik a *counter_pins.ucf* fájl, adjuk meg a lábhozzárendelést az alábbiaknak megfelelően (illetve a kártyáról leolvasva). Vegyük észre, hogy bár az órajel (clk) port neve ugyanaz, mint az előző feladatban, most más lábra van kötve, hiszen eddig a fejlesztői kábel szolgáltatta ezen bemenetet, most pedig a NYÁK-on levő oszcillátor a rendszerórajel forrása.

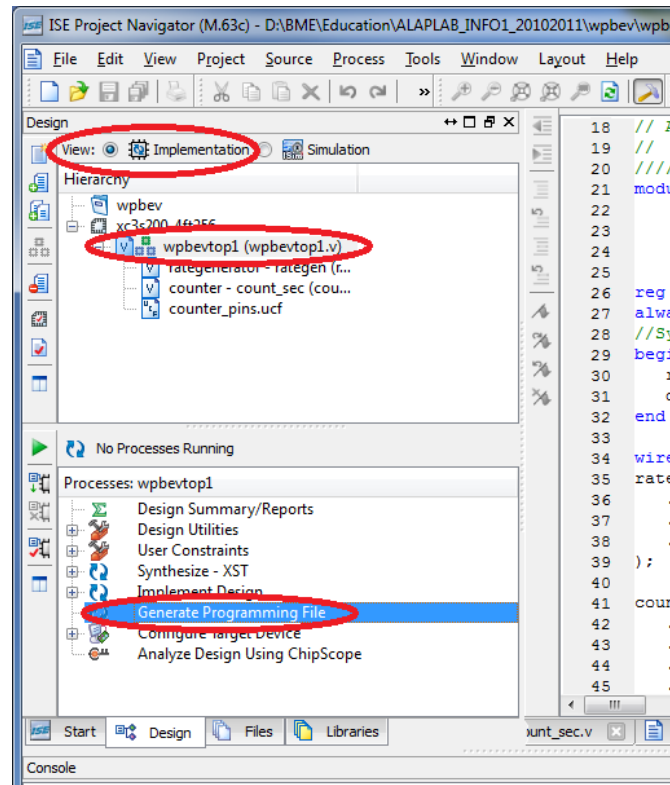
```

NET "clk" LOC="p56";
NET "btn0" LOC="p38";
NET "sw0" LOC="p101";
NET "q[3]" LOC="p53";
NET "q[2]" LOC="p54";
NET "q[1]" LOC="p58";
NET "q[0]" LOC="p59";

```

5.4. Implementáció, FPGA konfiguráció

Fordítsuk le a tervet és generáljunk FPGA konfigurációs fájlt a „**Generate Programming File**” opcióval. Ügyeljen rá, hogy a Hierarchy ablakban a legfelső szintű modul legyen kiválasztva.



A konfigurációs fájl elkészülte után konfigurálja az FPGA-t LOGSYS GUI-ban, és ellenőrizze a működést!

6. Megjegyzések a tervezési folyamathoz

Előfordulhat, hogy a Verilog leírás a szimulátor számára megfelelő, tehát szintaktikailag helyes, de a szintetizáló alrendszer nem fogadja el. Ennek oka lehet, hogy valamit nem talál egyértelműnek, vagy egy részt a leírás adott módjából nem képes szintetizálni. Az adott hibajelzések általában egyértelműek, és útmutatást adnak a leírás módosításához.

A **Synthesize-XST** alrendszerben a **View RTL Schematic** paranccsal meg lehet tekinteni a Verilog leírásból létrehozott RTL szintű vázlatot. Legelőször a legfelső szint jelenik meg, és ezután a hierarchia alsóbb szintjeinek RTL szintű kapcsolási vázlatait is megtekinthetjük. A **View Technology Schematic** az FPGA primitívekre (LUT, FF) leképzett tervet mutatja meg. Érdekes lehet megtekinteni a fejlesztő rendszer által generált jelentéseket (report) is.

