FPGA fejlesztés a gyakorlatban
oktatócsomag

Tartalomjegyzék

1. Általános ismertető....5
  1.1. A képzés célja....5
  1.2. Személyi feltételek....6
    1.2.1. A tanár személye....6
    1.2.2. A diák személye....6
  1.3. Tárgyi feltételek....7
  1.4. Útmutató az oktatócsomag használatához....7
2. Bevezetés....8
  2.1. A fejlesztőcsapat....8
    2.1.1. Előszó 1....8
    2.1.2. Előszó 2....8
    2.1.3. Előszó 3....9
  2.2. A kezdetek....10
  2.3. Némi magyarázat a címhez....11
  2.4. Az FPGA elhelyezése a műszaki életben....11
    2.4.1. ASIC és FPGA áramkörök....12
    2.4.2. FPGA felépítése....13
3. FPGA gyártók....15
4. Az FPGA belső lelki világa....15
  4.1. Spartan 3....15
    4.1.1. Bevezetés és rendelési információk....15
      4.1.1.1. Tulajdonságok....16
      4.1.1.2. Architektúrai áttekintés....17
      4.1.1.3. Konfiguráció....18
      4.1.1.4. I/O képességek....19
      4.1.1.5. Tokozás jelölései....19
      4.1.1.6. Rendelési információk....20
    4.1.2. Működés leírása....20
      4.1.2.1. IOB-k....20
      4.1.2.2. CLB áttekintés....26
      4.1.2.3. Blokk RAM áttekintés....27
      4.1.2.4. Dedikált szorzók....27
      4.1.2.5. Digitális Órajel Kezelő....27
      4.1.2.6. Huzalozás....30
    4.1.3. Határértékek....31
    4.1.4. Lábkiosztás....32
5. Fejlesztőpanelek....33
  5.1. Fejlesztőpanelek a ChipCad cég kínálatában 33
    5.1.1. BasysTM2 Spartan-3E FPGA panel....33
  5.2. NexysTM2 Spartan 3E FPGA fejlesztőpanel....34
  5.3. NexysTM3 Spartan 3E FPGA fejlesztőpanel....35
  5.4. Saját tervezésű fejlesztőpanelek....36
    5.4.1. Tápegységpanel felépítése....36

5.4.2. Adapterpanel felépítése....39
5.4.3. Perifériapanelek....40
5.4.3.1. 1. generációs próbapanel....41
5.4.3.2. 2. generációs próbapanel....42
5.4.3.3. 3. generációs próbapanel....43
6. Szoftveres fejlesztőkörnyezet....44
6.1. Xilinx ISE Design Suite....44
6.2. Xilinx Design Suite System Edition....46
6.2.1. Telepítési eljárás....46
6.2.2. A program felépítése és menürendszere....47
6.2.2.1. Menürendszer....48
6.2.2.1.1. File menü....49
6.2.2.1.2. Edit menü....50
6.2.2.1.3. View menü....68
6.2.2.1.4. Project menü....74
6.2.2.1.5. Source menü....75
6.2.2.1.6. Process menü....75
6.2.2.1.7. Tools menü....76
6.2.2.1.8. Window menü....78
6.2.2.1.9. Help menü....79
6.2.2.2. A leggyakrabban használt ikonok....80
6.2.3. Új projekt létrehozása....81
6.3. Impact bemutatása....81
6.4. A fejlesztés bemutatása egy komplex példán keresztül....82
7. SCH alapú fejlesztés....83
8. VHDL alapfogalmak....84
8.1. Röviden a VHDL nyelvről....84
8.1.1. Egy VHDL nyelven megírt alkatrész felépítése....85
8.2. Alapvető nyelvi elemek....86
8.2.1. Lefoglalt kulcsszavak....86
8.2.2. Karakterkészlet....87
8.2.3. Elválasztók....87
8.2.4. Megjegyzések....87
8.2.5. Konstansok....87
8.2.5.1. Numerikus konstansok....87
8.2.5.2. Felsorolás típusú konstansok....87
8.2.5.3. Sztring típusú konstansok....88
8.2.5.4. Bitsztring típusú konstansok....88
8.2.5.5. Null konstans....88
8.3. Adattípusok....88
8.3.1. Felsorolás típus....89
8.3.1.1. Felsorolás típus túltöltése....89
8.3.1.2. Felsorolás típus kódolása....90
8.3.2. Egész típus....90
8.3.3. Tömb típus....91
8.3.4. Record típus....92
8.3.5. Nem támogatott VHDL típusok....93
8.3.6. Előre definiált típusok....93

8.3.7. Típuskonverziók....93
8.3.8. Downto és To használata....94
8.4. Változók és jelek....95
8.5. Entitások....96
8.6. Architektúra leírása....97
8.7. Komponensek megadása....98
8.8. Operátorok....100
8.8.1. Logikai operátorok....100
8.8.2. Relációs operátorok....101
8.8.3. Összeadó operátorok....101
8.8.4. Előjel operátorok....101
8.8.5. Szorzó operátorok....101
8.8.6. Egyéb operátorok....101
8.8.7. Operandusok....101
8.8.7.1. Operandusok tulajdonságai....102
8.9. Utasítások....102
8.9.1. Kombinációs és sorrendi leírás....102
8.9.2. A process....102
8.10. Csomagok a VHDL nyelvben....102
8.10.1. Gyári könyvtárak használata....102
8.10.2. Saját package-ek....102
9. Programok a fejlesztőpanelekre....103
9.1. 1. generációs próbapanel....103
9.1.1. Lábkiosztás a próbapanelhez....104
9.1.2. LED-ek és kapcsolók....105
9.1.2.1. Egyszerű kombinációs hálózat....105
9.1.2.2. Szavazó áramkör....109
9.1.2.3. Adott frekvenciájú négyszögjel előállítása (villogás)....112
9.1.2.4. Pergésmentesítés....116
9.1.2.5. Futófény programok....117
9.1.2.6. PWM jel előállítása (fényerőszabályozás)....122
9.1.2.7. Knight Rider futófény....128
9.1.3. Hétszegmenses kijelző....135
9.1.3.1. Dekódolás és multiplexelés....135
9.1.3.2. BCD to Binary és Binary to BCD átalakítások....143
9.1.3.3. Fényerő-szabályozás a hétszegmenses kijelzőn....144
9.1.4. Komplex feladat az 1. generációs próbapanelre....144
9.2. 2. generációs próbapanel....144
9.2.1. Lábkiosztás a próbapanelhez....144
9.2.2. Mátrix tasztatúra lekezelése....145
9.2.3. Karakteres LCD kijelző meghajtása....147
9.2.4. A panel kiegészítése....154
9.2.4.1. Szervomotor vezérlés....154
9.2.4.2. Soros port lekezelése....154
9.2.5. Komplex feladat a 2. generációs próbapanelre....155
9.3. 3. generációs próbapanel....155
9.3.1. Lábkiosztás a próbapanelhez....155
9.3.2. VGA jel előállítása....155

9.3.3. PS2 port lekezelése....156
9.3.4. AD átalakítás és I2C protokoll....156
9.3.5. Komplex feladat a 3. generációs próbapanelre....156
10. Tematika....157
10.1. Egy javasolt szakköri óramenet....157
10.2. Tanmenet....158
10.2.1. Utószó 1....162
10.2.2. Utószó 2....162
10.2.3. Utószó 3....162

1. Általános ismertető

Jelen dokumentum egy oktatócsomag Xilinx típusú FPGA-k fejlesztésére irányuló szakkörök számára. A leírás középiskolában való oktatásra készült. A középiskolai szakköri oktatás az órarendszerivel szemben számos eltérést mutat. Mivel a szakkörökön való részvétel nem kötelező, feltételezhetjük, hogy komoly érdeklődéssel rendelkező diákok vesznek részt a foglalkozásokon. Mindemellett egyfajta kötetlenség is jellemző ezekre az órákra. Az oktatócsomag – nem véletlenül – nem tartalmazza a különböző fejlesztésre kerülő kompetenciákat, mivel hiszem, hogy az oktatásban nem egyes kompetenciákat kell fejleszteni, hanem a diák egész lényét, személyiségét kell helyes irányba terelnünk ismeretek, minták átadásával.

A dokumentum végén található egy szakköri tanmenet, amelytől azonban a helyi sajátosságokra való tekintettel el lehet, sőt érdemes is eltérni. Miután nem órarendi kötött foglalkozásokról van szó óravázlatok ismertetésétől el kellett tekintenünk. Ez sajnos többletfeladatként jelentkezik az oktatócsomagot használó pedagógus munkájában. Kompenzálásként megpróbálunk egy olyan tudásbázist háttérként felvonultatni, amelyre bátran lehet építkezni az oktatás során.

1.1. A képzés célja

A képzés célja olyan diákok megjelenése a műszaki területen, akik komplex fejlesztési feladatokat tudnak végrehajtani FPGA segítségével. Hiszem, hogy megfelelő középfokú képzés – nevezhetjük alapozásnak is – nélkül nem tudunk a társadalom számára hasznos szakembereket, technikusokat, mérnököket képezni. A középiskolában tudjuk megszerettetni a szakmát a gyerekekkel, itt keltődik fel bennük az érdeklődés egyes területek iránt. Itt alakul ki a szakma iránti alázat, a nyugodt, megfontolt, precíz munkavégzés.

A képzés nem titkolt másik célja szembeszállni az uralkodó nézettel, miszerint a mai fiatal generáció a „mások” generációja, akiket már új módszerekkel kell oktatni, más képességeket, készségeket, tudást kell elvárni tőlük. Bár az igaz, hogy minden generációnak vannak sajátosságaik, a társadalom elvárása csak nagyon hosszútávon változhat meg a fiatalokkal szemben. Nem fogadhatjuk el, hogy a gyerekek ma már nem olvasnak, nem tanulnak otthon, nem készülnek az órákra megfelelően, hogy „sikk” lett tanulás nélkül átmenni, hogy a kettes a cél. Ahelyett, hogy a kiváló képességű és sok munkát befektető tanulókat helyezzük előtérbe, őket állítjuk mintaként elnézzük, hogy társaik szemében sokszor a „problémás” diák az iránymutató, ő dominál. A tehetséggondozásra egyre kevesebb idő/pénz/energia jut. Pedig véleményem szerint a tehetséggondozás az egyetlen útja a tanulók többségének felzárkóztatásához. A fiatalokra sokkal nagyobb hatással vannak kortársaik, mint pedagógusaik. Amennyiben el tudjuk érni, hogy a megfelelő társaikra nézzenek fel, őket tekintsék mintaalapnak, azzal sokkal többet érünk el, mint hangzatos jelszavakkal és pótcselekvésekkel. A szakkörök kiváló lehetőséget biztosítanak az életre való nevelésre. A résztvevők számára be tudjuk mutatni, hogy milyen nehézségekkel, problémákkal találkoznak majd munkájuk során. A kiscsoportos foglalkozás lehetővé teszi a személyes kapcsolat kialakítását és elmélyítését. A diákok a gyakorlat központú oktatásban megtanulhatják, hogy tevékenységüknek következményei vannak, ezért a jogokkal szemben a kötelességek kerülnek előtérbe. Amennyiben a diákok a szakkörökön tanúsított magatartást, viselkedési modellt, munkamorált integrálják a hétköznapi órai tevékenységeikbe, kortársaik egy jól működő a társadalom számára is hasznos életstílust figyelhetnek meg, vehetnek mintának.

1.2. Személyi feltételek

1.2.1. A tanár személye

A képzés jól felkészült, rugalmas villamosmérnököt igényel a tanítás oldaláról. A szakköri munka mindemellett több energiát igényel, mint a hétköznapi tanítás. Miután érdeklődő és jól felkészült diákokkal kell dolgoznunk – az előnyei mellett – ennek hátrányaival is meg kell küzdenünk. Az alapok elsajátítása után célszerű egyéni, komplex feladatokat adni a diákoknak érdeklődésük alapján. Ez sok pluszmunkát igényel, mindemellett alkalmazkodni kell a helyi sajátosságokhoz is (iskolai felszereltség, anyagi körülmények, stb.). Egy pedagógus számára sohasem könnyű bevallani, ha valamihez nem ért, ha egy kérdésre nem tudja a választ. Szakköri foglalkozásokon gyakran találkozunk olyan problémákkal, amelyekre hirtelen nem tudjuk a megfelelő megoldást. Ilyen esetekben az elkerülés, a „nem mondom meg, nézz utána" effektus nagyon káros hatással lehet a diákok fejlődésére, és az oktatóval való kapcsolatukra. Ne féljünk elismerni, hogy az adott területen még nekünk is van mit tanulnunk! Próbáljuk a diákokkal együtt megkeresni az adott helyzethez legjobban illeszkedő megoldást! Ezáltal a diákok szemében a tanár sokkal megfoghatóbb, emberközelibb lesz, nő a diákok önbizalma, és a kialakult jó kapcsolatnak köszönhetően bátrabban fordulnak hozzánk problémáikkal. Szintén üdvös lehet az eltérő megoldásokra való befogadókészségünk növelése. Egy problémára számtalan megoldás adható, és bár a hatékonyságuk és kifinomultságuk (eleganciájuk) alapján lehet rangsorolni a műszaki életben mégis az a fontos, hogy az adott feladatot a készülék ellássa, működjön a vas. Célszerű a nem a mi gondolatmenetünkbe illeszkedő megoldásokat is díjazni. Elemzésük során sokat tanulhat mind a diák, mind a tanár.

1.2.2. A diák személye

Miután a képzés szakköri keretek között zajlik, ezért a résztvevő diákok odaadása nem lehet kétséges. Előzetes tudásuk alapján – tapasztalataim szerint – nagyon heterogén csoporttal is működtethető az oktatócsomag. Előnyt jelent, ha a résztvevők rendelkeznek digitális technikai és programozási ismeretekkel. Minimális előzetes ismeretként az elektronika és a számítástechnika területén történő jártasságot lehet megemlíteni. A legfontosabb tulajdonság azonban ami szükséges a tanulók részéről a kitartás. Ez talán a legnehezebb a mai oktatási helyzetben. A diákok többsége érdeklődő, szívesen foglalkozik új dolgokkal egészen az első kudarcélményig. Itt történik egy megtorpanás, ami az előző generációknak nem volt sajátja. Elkezdődik az „én ehhez hülye vagyok", az „úgysem fog sikerülni" és a kedvencem az „ez lehetetlen" mantra. Nagyon nehéz kizökkenteni ebből az érzelmi helyzetből tanulóinkat. A mai mainstream pedagógia a pozitív megerősítést szorgalmazza, és tekinti szinte csodaszernek. Ezzel mindössze az a probléma, hogy ha nincs mögötte tartalom, munka, akkor a pozitív megerősítés visszafelé sülhet el („mindegy mit csinálok, akkor is megdicsérnek"). Én nagyon fontosnak tartom, hogy a diákok felé ne tartsunk görbe tükröt. Komoly hátrányok származnak abból, ha nem a valós teljesítmény alapján ítéljük meg tanulóinkat, hanem megpróbáljuk még a széltől is óvni őket. Az életben ez a magatartás nagyon gyorsan vissza fog ütni. Éppen ezért fontos, hogy tisztázzuk, a kudarc az élet velejárója, nélküle nincs munkavégzés (gyakori mondás, hogy „csak az nem hibázik, aki nem dolgozik"). Meg kell értetnünk a gyerekekkel a kudarcok feldolgozását és a továbblépést. Közhely ugyan, de a hibáinkból tudunk a legtöbbet tanulni. Ebben a folyamatban nagyon sokat segíthet, ha a pedagógus őszintén beszél saját szakmai kudarcairól, olyan esetekről, amelyekben ő hibázott. Ezáltal a tanulók kézzelfogható, valós mintát láthatnak, hogy hogyan lesz egy gyakran hibázó, de érdeklődő és kitartó fiatalemberből kiváló szakember.

1.3. Tárgyi feltételek

Az FPGA fejlesztés oktatásához elengedhetetlen egy korszerűen felszerelt műhely, amelynek minimális tartozékai:

- Korszerű számítógéppark[1] (a szoftveres fejlesztőkörnyezet gépigénye igen magas, egyes projektek fordítása még erős processzor és nagy memória mellett is időigényes.)
- Alapműszerek a hardveres problémák felderítésére
  - Tápegység
  - Függvénygenerátor
  - Multiméter
  - Oszcilloszkóp
  - Fejlesztőpanelek

1.4. Útmutató az oktatócsomag használatához

Az oktatócsomag készítésekor kettős cél lebegett szemünk előtt. Egyrészt teljesíteni szerettük volna egy oktatócsomaggal szemben támasztott követelményeket – didaktikai célok – mindemellett szerettünk volna egy önállóan is jól használható ha nem is teljes, de széleskörű ismereteket biztosító leírást közreadni. A kettős célt igencsak nehéz volt a dokumentum készítésekor mindvégig szem előtt tartani. Néhány helyen a didaktikai célok háttérbe kényszerültek a rendszertervezési és szakmai célokkal szemben. Ezt a folyamatot kompenzálandó, amennyiben lehetőségem lesz rá szakdolgozatomban ellensúlyozni kívánom, amelyet az FPGA középiskolai oktatásának módszertanából kívánok majd megírni. Amennyiben erre sor kerül, akkor az oktatócsomagot igen komoly módszertani és didaktikai ismeretekkel, feladatokkal egészíthetjük ki.

1 Minimum: Dual Core, 2GB RAM, javasolt: I5/I7 4-8GB RAM.

2. Bevezetés

2.1. A fejlesztőcsapat

2.1.1. Előszó 1

A vezérlési rendszerekkel való ismerkedésemet – mint azt többen – a PIC mikrovezérlőkkel kezdtem. Eleinte csak assembly nyelven programoztam, de hamar megismerkedtem a C nyelvvel, és azóta szinte kizárólag azt használom. A váltáskor furcsa volt megszokni a magasszintű nyelv – assembly-hez képest – bonyolult szintaktikáját. Mikor megismertem a VHDL-t, úgy éreztem, ismét egy sokkal bonyolultabb nyelvi felépítést kell megszoknom. Minden programnyelv elsajátítása a szintaktika megismerésével kezdődik, így a VHDL igen könnyen el tudja ijeszteni a tanulni vágyókat már a kezdetekkor. Ráadásul a fejlesztőkörnyezet sokrétűsége is rátesz erre egy lapáttal. Azoknak ajánlom az FPGA-val való fejlesztés tanulását, akik érzik magukban a megfelelő kitartást.

Legnagyobb előnye – véleményem szerint – az FPGA-k használatának, a párhuzamos működés. A PIC mikrovezérlőknél hamar eljutottam ahhoz a problémához, hogy egyszerre több dolgot szeretnék csináltatni az eszközzel. Ilyenkor jöttek képbe a megszakítások, amelyek alkalmazása rendesen megbonyolította a programot, és sok probléma forrásává váltak. Mindemellett a megszakításokkal csak látszólag tudjuk azt a párhuzamost működést modellezni, ami egy FPGA esetén a gyakorlatban is rendelkezésünkre áll.

Erdélyi „Mindenmüxik” Zsolt

2.1.2. Előszó 2

Mindig is utáltam alkalmazkodni más rendszerekhez, szabványokhoz, mivel ha új dologgal foglalkoztam nem tudtam hogy egyes parancsok hogyan, milyen hatást fejtenek ki. Digitális vezérlés terén a PIC mikrovezérlővel kezdtem, mely lényegesen egyszerűbb mint az FPGA, de más által elkészített architektúrához és perifériákhoz kell alkalmazkodni. Az FPGA különb ettől. Itt a kezdetektől én írhatok meg mindent, a digitális vezérlés elemi szintjén én alkotom meg a perifériákat. Például PIC esetén van hardveres SPI (kommunikációs protokoll), végzi a dolgát ahogy a tervezők összerakták, de ha valami hiba van akkor adatlapokban, leírásokban kell keresnem információkat hogy mégis hogyan kell elküldenem a parancsot a perifériának. VHDL-ben én írom meg ezt a modult, és ha valami hiba felmerül, nem kell kutakodni annyit hisz én írtam, tudom hogy működik.

Röviden, tömören, az FPGA-val „korlátok nélkül” alkothatok a digitális technika keretein belül, és az jó – nagyon jó!

Pázmándi „Romboló” Péter

2.1.3. Előszó 3

Kinek is ajánlanám az FPGA fejlesztés oktatását? Mindenképpen olyan pedagógus kezdjen bele a munkába aki lelkileg frissnek érzi magát és képes a megújulásra, szereti az új dolgokat, mert a témakör a megszokott elektronika és programozás tanításával szemben új szemléletet kíván. Nehéz összeegyeztetni a hardverközpontú gondolkodást a hagyományos programozási ismereteinkkel. Az a tény, hogy nem egy hardverre írunk programot, hanem magát a hardvert alakítjuk ki, valamint a párhuzamos működés egy jól felkészült digitális technikában jártas személyt kíván. A fejlesztőkörnyezetek azonban támogatják a meglévő hardverelemekből felépített új absztrakciók létrehozását, ami nagyfokú kreativitást is feltételez. Mindemellett meg kell említenünk, ahogy a szakma minden szintjén itt is fontos a másik oldal, a diákok elkötelezettsége. Sok otthoni munkát felkészülést igényel az FPGA fejlesztés – amit a jelenlegi oktatáspolitika nem támogat (ahogy az eddigiek sem). A pedagógusok terheltsége előre láthatóan csak nőni fog a jövőben, úgyhogy érdemes mindenképpen elgondolkodni a munka megkezdése előtt, hogy végig tudjuk-e csinálni.

Szeretném azt gondolni, hogy jelen dokumentáció hiánypótlóként szerepel majd, hiszen a témáról magyar nyelvű szakirodalmat találni nem könnyű feladat.

Az oktatócsomag folyamatos fejlesztés alatt áll, mind a hardver-, mind a szoftverelemek bővítésre kerülnek. A fejlesztőkörnyezet leírását is bővíteni fogjuk. Oly sok funkcióval rendelkezik, amelyet egy dokumentumban nem tartottunk szerencsésnek szerepeltetni, ezért itt csak egy általános ismertetőre törekedtünk. A későbbiek folyamán tervezzük az egyes részegységek sokkal részletesebb bemutatását is. Ezek a munkák ugyan külön dokumentumban jelennek majd meg, mégis jelen leírás részének tekinthetőek. A teljes dokumentációt az iskolánk – Mechatronikai Szakközépiskola – által üzemeltetett Moodle keretrendszeren keresztül lehet elérni.

Varga László

2.2. A kezdetek

Egykori tanárom, jelenlegi barátom és kollégám Juhász Róbert iskolánkban már jó ideje tart sikeresen mikrovezérlő szakkört. A szakkörökön a diákok megismerkedhetnek a mikrovezérlő felépítésével programozásával, a leggyakrabban használt perifériák lekezelésével. Ezeken a szakkörökön és egyéb beszélgetéseinken került szóba az FPGA, mint a beágyazott rendszerek és irányítások egy újabb lehetősége. Még 2012 elején kérdezte meg egyik diákom – Pázmándi Péter – hogy nincs-e kedvem szakkör szintjén FPGA fejlesztést oktatni az iskolában. Nehezen tudott rávenni, hogy komolyabban foglalkozzak az üggyel. Az FPGA mint áramkör igen bonyolult, ennek következtében a rá alapuló fejlesztéshez igen széleskörű ismeretek, és nagyfokú kitartás szükséges. Egy ilyen irányú szakkör inkább egyetemi szinten lenne helyénvaló. A Kandón lehetőségem volt egy – FPGA fejlesztést is magába foglaló – Mikroszámítógép Laboratórium című tantárgy oktatásában való részvételre. Az itt tapasztaltak is meggyőztek arról, hogy az FPGA-val foglalkozni nagyon sok időt és energiát emészt fel, mindemellett a fejlesztőkörnyezetek is igen drágák. Azonban sokszor még mi tanárok is lebecsüljük egy diák kitartását és lelkesedését. Végül az egyetlen lehetőséget választottam, a szakkör elkezdését. Egy új témakör oktatása természetesen számos buktatót tartalmaz. Sok problémát kellett megoldanunk, amelyek között akadt hardveres és szoftveres gond is épp elég. Ebben a munkában rengeteg segítséget kaptam a szakkörön résztvevő diákoktól, amelyet itt szeretnék megköszönni. Az első év nehézségei után úgy érzem a tapasztalatokra egy jól működő és színvonalas szakkör építhető, amelyen a diákokat meg lehet ismertetni a legújabb – az iparban is használatos – technológiákkal.

A dokumentáció elkészültéhez lényegesen hozzájárult, hogy jelenleg az Óbudai Egyetem villamosmérnök-mérnöktanári képzésén veszek részt és a Trefort Ágoston Mérnökpedagógiai Központ keretében tartott tantárgyak teljesítéséhez szükséges ábrák, képek manipulálása, prezentációk készítése, oktatócsomag készítése, elektronikus tanulás elősegítése, stb. Úgy gondoltam, hogy készítsünk akkor egy jól használható teljes körű leírást, ami legújabb fejlesztéseinket tartalmazza.

Az oktatócsomagban megpróbáljuk bebizonyítani, hogy az FPGA fejlesztés – bár nem gyerekjáték – nem haladja meg a középfokú oktatás kereteit. Amennyiben hajlandóak vagyunk az elején egy kis energiabefektetésre és elvonatkoztatunk a hagyományos mikroprocesszoros rendszerektől, az alapperifériákat egyszerűen le tudjuk kezelni, a megalkotott architektúrák pedig igen sokrétűen használhatóak, egymásra építhetőek.

Olyan dokumentáció megalkotása volt a célunk, amelyet pedagógusi irányítás mellett a diákság széles köre tud használni.

A többes szám nem véletlen az utolsó három bekezdésben. Az oktatócsomag tartalmát a fejlesztőcsapat közösen készítette el – bár a projektvezetői és szerkesztési feladatokat magamra vállaltam.

A dokumentációk elkészítésekor próbáltuk az egységes kinézetet és szemléletet előtérbe helyezni. A megfogalmazásoknál az angol és magyar szakirodalom megnevezéseit felváltva használjuk (szükség szerint lábjegyzetben). Miután a munkában eltérő korú, tapasztalatú és habitusú emberek vettek részt, ez néhány helyen meglátszik a fogalmazásoknál. Hiszem, hogy ez munkánk értékét nemhogy csökkenti, inkább növeli.

2.3. Némi magyarázat a címhez

Az FPGA áramkörökben a mikroprocesszorral, vagy mikrovezérlővel ellentétben nincs kialakítva egy univerzális Turing gép, amivel minden feladat megoldható megfelelő program segítségével. Éppen ezért egy FPGA-ra nem lehet programot írni, hiszen nincs meg a vas, ami ezt a programot végrehajtsa. Bár egy szoftvert írunk alapvetően, ez mégis egy hardver kialakítását határozza meg. A fejlesztés során egy konfigurációs bitminta keletkezik, amelyet az FPGA-ba töltünk, így kialakítva a belső hardverelemek kapcsolatait. Aki FPGA-val foglalkozik tehát nem programot ír, hanem architektúrát fejleszt.

Az oktatócsomag a gyakorlatra helyezi a hangsúlyt. A legfontosabb egy adott feladat végrehajtása és a megfelelő működés biztosítása – némely esetben ez az elegancia és az elméleti ismeretek rovására is megy, azonban úgy gondoljuk, hogy a műszaki élet, a mérnöki munka fundamentuma egy projekt során a berendezés működőképességének biztosítása.

2.4. Az FPGA elhelyezése a műszaki életben

Az FPGA – Field Programmable Gate Array (helyben programozható logikai kaputömb) egy digitális technikai áramkör, amelyben alap építőelemekből tetszőleges architektúrát alakíthatunk ki[2].

Egy adott irányítástechnikai feladat végrehajtására ma számos eszközt felhasználhatunk. A hőskorban huzalozott architektúrákat alkalmaztak, amelyek lényege, hogy az adott feladatra megtervezünk egy hardveregységet (vezérlőegység), ami a lehető leghatékonyabban oldja meg az adott problémát. A hatékonyság mellett ez a rendszer rendkívül rugalmatlan volt. Amennyiben a feladat megváltozott csak újabb rendszertervezés és a huzalozás fizikai megváltoztatása segített. A mikroprogramozás növelte a rugalmasságot bár az univerzalitás bizonyos szempontból a hatékonyság rovására ment.

Ma a korszerű irányításokban mikroprocesszoros rendszereket, mikrovezérlőket, PLC-ket alkalmaznak. A PLC-k nagyon rugalmasak és ipari körülményekre vannak optimalizálva, azonban nagy ciklusidejük miatt nem alkalmasak gyors beavatkozásra, bonyolult műveletek végrehajtására. A mikroprocesszor nagy sebességű, de a kötött architektúra néhány hardveres szempontból egyszerű feladat végrehajtását igencsak megnehezíti. A mikrovezérlő kihasználva a mikroprocesszor nagy sebességét vezérlésre optimalizált RISC CPU-t tartalmaz és számos perifériát ami megkönnyíti a feladatok ellátását. Mindhárom rendszer hátránya a soros utasítás-végrehajtás, és az ezzel együtt járó behatárolt felhasználási terület. A hétköznapi ipari irányítástechnológiában ez nem jelent gondot. Egy csomagológép vezérléséhez nem kell nanoszekundumok alatt több műveletet is egymással párhuzamosan végrehajtani. A feladatot elláthatja egy PLC, vagy egy mikrovezérlő is. Hasonlóan egy gyártórendszer üzemeltetéséhez, vagy egy egyszerű épületautomatizálási probléma megoldásához.

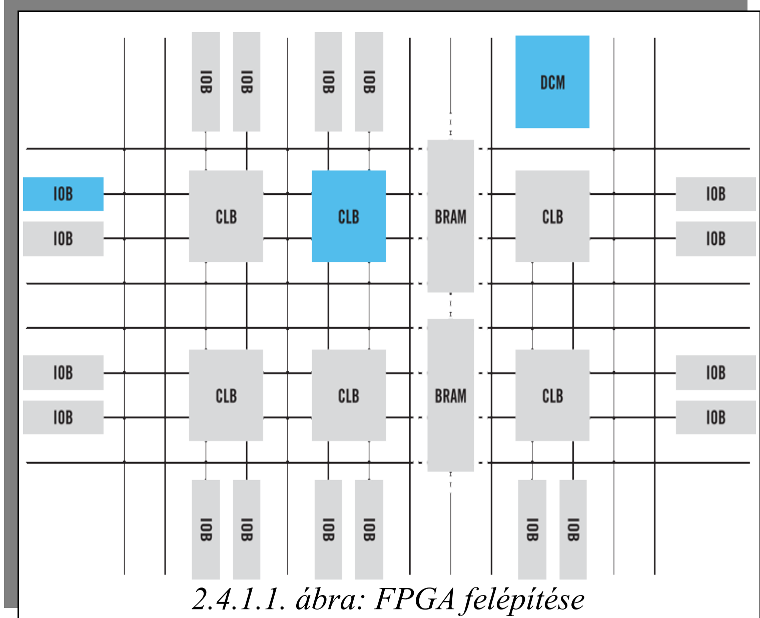
Egyre gyakoribb, hogy a megrendelő/felhasználó egyszerűbb eszközök esetén is elvárja az igen magas fokú intelligenciát a berendezéstől. Ezek többsége megoldható mikrovezérlőkkel, azonban vannak olyan esetek, ahol ez már kevésnek bizonyul (képfeldolgozás, VGA jelek előállítása). Ilyenkor kerülnek előtérbe az FPGA-k. Ma leginkább a nagyon gyors kommunikáció, a képfeldolgozás, és az összetett, hagyományos architektúrára nem illeszkedő, vagy párhuzamos utasítás-végrehajtást megkívánó feladatok esetén alkalmazunk FPGA-t. A kódtörésben pl. verhetetlen egy ilyen áramkör.

---

2 Természetesen az architektúra bonyolultságát korlátozza az adott típusra jellemző logikai cellák száma, a beépített memória mérete, a magasabb szintű hardverelemek típusa és száma.

2.4.1. ASIC és FPGA áramkörök

Ma egy bonyolult vezérlés, vagy szabályozás leghatékonyabban egy ASIC[3] (Application Specific Integrated Circuit – Alkalmazásra gyártott integrált áramkör), vagy FPGA áramkörrel oldható meg. Az alkalmazásra gyártott IC-k belső boundolását[4] úgy alakítják ki, hogy az adott feladatot a lehető legnagyobb hatékonysággal nagy megbízhatósággal, és kis fogyasztás mellett hajtsák végre. Integrált kivitelben meg lehet valósítani digitális, analóg és vegyes[5] áramköröket is. Az ASIC áramkörök nagy hátránya, hogy a fejlesztés nagyon speciális szoftver és hardverállományt igényel. Egy ASIC megalkotása nagyon nagy költségbefektetést jelent, ezért csak nagy szériaszám esetén éri meg az alkalmazása. Kisebb tételszám esetén kerülnek előtérbe az FPGA áramkörök. Ebben az esetben a hardver belső kialakítását a felhasználó[6] határozhatja meg. Komoly megkötés azonban, hogy csak digitális technikai áramköri elemeket alkalmazhatunk a hardverben.



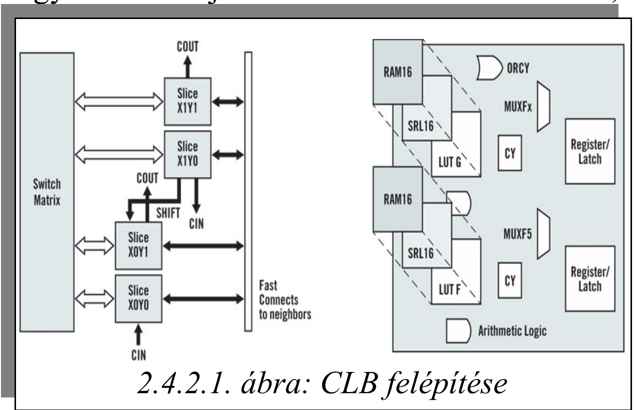
2.4.1.1. ábra: FPGA felépítése

Az FPGA-ban úgynevezett konfigurálható logikai blokkok[7] összeköttetését tudjuk megadni, amelyek IO[8] blokkokon keresztül kapcsolódnak a külvilághoz (l. 2.4.1.1. ábra[1]). Az ASIC áramkörök tehát hatékonyabbak, azonban ezek belső huzalozását a gyártó alakítja ki igen magas áron, míg az FPGA-t mi tudjuk felkonfigurálni alacsony költségek mellett az adott feladatra[9] – ezért azonban néhány megkötést el kell fogadnunk. Bár vannak OTP[10] tokozású FPGA-k kész projektekre, az uralkodó változatok SRAM alapúak, hogy a folyamatos fejlesztést megkönnyítsék. A 2.4.1.1. ábrán láthatjuk egy Xilinx FPGA felépítését. A már említett CLB és IOB mellett az FPGA tartalmaz BRAM[11] és DCM[12] egységeket is.

3 Magyar megfelelője a BOÁK – Berendezés Orientált ÁramKör.
4 huzalozás
5 mixed
6 Itt felhasználó alatt egy fejlesztőmérnököt értünk.
7 CLB – Configurable Logic Blocks.
8 Input/output – be/kimeneti
9 Egy FPGA-s fejlesztés során még a teljes hardverkialakítás után is tudunk módosítani a működésen, ha a feltételek, vagy a tesztek eredményei így kívánják – ez nagyon rugalmassá teszi a rendszert.
10 One-Time Programmable – Egyszer programozható
11 Block RAM – ezen memóriaegységek ún. többszörös hozzáférésű memóriablokkok.
12 Digital Clock Manager, ami felhasználható digitális frekvencia szintetizátorként, DLL-ként.

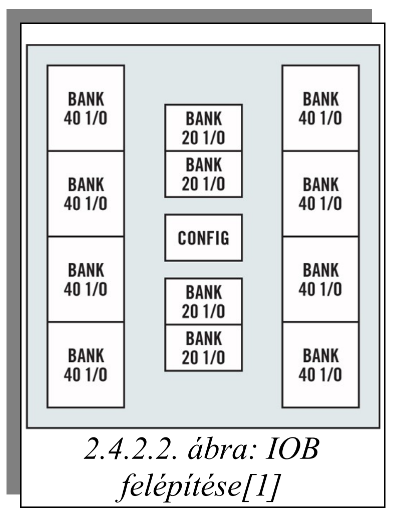
2.4.2. FPGA felépítése

Az FPGA alapvető építőelemei tehát a CLB-k, ezeknek a száma és pontos jellemzői típusfüggőek, azonban vannak általános tulajdonságok. Egy CLB általában 4-6 bemenetből áll, tartozik hozzá egy konfigurálható kapcsolómátrix – ami az összeköttetésekért felelős – egy tároló és néhány adatkiválasztó (2.4.2.1. ábra). A kapcsolómátrix segítségével tudjuk beállítani, hogy a CLB hogyan viselkedjen. Lehet kombinációs hálózat, shift regiszter, MUX/DEMUX, tároló, RAM, stb.



2.4.2.1. ábra: CLB felépítése

A 2.4.2.1. ábrán[1] is látható CLB-k megfelelő összeköttetését, az IOB-khez, órajelekhez való csatlakozását a tervező szoftver biztosítja, megkönnyítve ezzel a fejlesztést. A fejlesztő ezen CLB-k absztrakcióit látja csupán (pl. digitális kapukat, MUX-okat, flip-flop[13]-okat), vagy a hardver viselkedési leírását alkotja meg egy hardverleíró nyelv segítségével.



2.4.2.2. ábra: IOB felépítése[1]

Az FPGA áramkörökben lévő IOB-k segítségével számos szabványosított kommunikációt használhatunk a külvilággal való kapcsolattartásra (GTL, HSTTL, LVCMOS, stb.). A fejlesztést megkönnyíti, hogy az I/O lábak bankszelektáltak, ezáltal az egyes kimenetek driverei pl. különböző feszültségszintekre húzhatók fel. Az FPGA-k a hagyományos eszközökkel szemben tehát nagyszámú[14] és nagyon rugalmasan használható IO lábakkal rendelkeznek. Mindemellett a korszerű típusok tartalmaznak DCI áramkört, amivel a fel- és lehúzó ellenállások digitálisan állíthatók be az implementáció függvényében.

Minden FPGA rendelkezik beágyazott blokk RAM-mal[15], ez biztosíthatja, hogy az FPGA ne csak a vezérlést hajtsa végre, hanem az ehhez szükséges adatok tárolását is ellássa. Ezen memóriablokkok ún. dual portosak, vagyis kétirányú, egyidejű hozzáférést biztosítanak.

13 tároló
14 Spartan 3 esetén akár 633db
15 Spartan 3 esetén akár 1872kb

A Xilinx cég vezető FPGA családjainak tulajdonságai[1]						
Jellemző	Artix™-7	Kintex™-7	Virtex®-7	Virtex-6	Spartan®-6	Spartan®-3
Logikai cellák száma	215000	480000	2000000	760000	150000	74880
BRAM						1872k
DSP száma	740	1920	3600	2016	180	-
DSP teljesítmény (symmetric FIR[16])	930GMACS[17]	2,845GMACS	5,335GMACS	2,419GMACS	140GMACS	-
Adó-vevők száma	16	32	96	72	8	-
Adó-vevők sebessége	6,6Gb/s	12.5Gb/s	28.05Gb/s	11.18Gb/s	3.2Gb/s	-
Adó-vevők sávszélesség (full-duplex)	211Gb/s	800Gb/s	2,784Gb/s	536Gb/s	50Gb/s	-
Memória interfész (DDR3)	1,066Mb/s	1,866Mb/s	1,866Mb/s	1,066Mb/s	800Mb/s	333Mb/s[18]
PCI Expressz® interfész	x4 Gen2	x8 Gen2	x8 Gen3	x8 Gen2	x1 Gen1	-
AMS[19]/XADC	☺	☺	☺	☺	-	☺
AES[20]	☺	☺	☺	☺	☺	☺
I/O lábak száma	500	500	1200	1200	576	633
I/O lábak feszültsége	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.5V, 1.8V, 2.5V	1.2V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.5V, 1.8V, 2.5V, 3.3V
EasyPath Cost Reduction Solution	-	☺	☺	☺	-	-

16 Finite Impulse Response – Véges impulzusválasz
17 MAC: Multiply-Accumulate
18 DDR2
19 Analog Mixed Signal – analóg kevert jelek
20 AES: Advanced Encryption Standard

3. FPGA gyártók

A legfontosabb FPGA gyártók:

- Xilinx
- Altera
- Lattice
- Microsemi
- Quicklogic

Részesedésüket a piacon l. http://www.fpgadeveloper.com/2011/07/list-and-comparison-of-fpga-companies.html

Jegyzetünkben mi a legjelentősebb gyártó a Xilinx cég méltán népszerű Spartan-3 FPGA-jával foglalkozunk. A Xilinx gyártotta az első FPGA-t és azóta is megőrizte vezető pozícióját. A Spartan család nagyon jó ár/érték aránnyal rendelkezik. 2-3000 Ft-ért rendelkezésünkre áll egy sokoldalúan használható logikai kaputömb. A Virtex család már profi fejlesztésekre való, korszerű DSP rendszerekben alkalmazzák. Magas ára miatt speciális tulajdonságaira ebben a dokumentációban nem térünk ki.

4. Az FPGA belső lelki világa

Jelen fejezet ismerteti a Xilinx cég Spartan-3 FPGA-jának legfontosabb tulajdonságait. Az eredeti angol nyelvű leírás megtalálható az oktatócsomag mellett a Moodle rendszerben.

4.1. Spartan 3

4.1.1. Bevezetés és rendelési információk

A Spartan-3-as FPGA családot arra fejlesztették ki, hogy a nagy volumenű alkalmazások olcsón kivitelezhetőek legyenek. A nyolctagú családban 50.000-től 5.000.000-ig terjed a rendszerkapuk száma. (l. 17. oldal)

A Spartan-3 család az őt megelőző Spartan-IIE család sikerei alapján készült – megnövelték a belső RAM-ot, az I/O lábak számát és összességében a teljesítmény szintjét az órajel kezelő funkció javításával. Számos fejlesztés a Virtex-II technológiájából származik.

A kiemelkedően alacsony áruk miatt a Spartan-3 FPGA-k széles körben ideálisan alkalmazhatóak, pl. kijelzők, digitális TV eszközök, otthoni hálózatok fejlesztésénél.

Kiváló ár/érték arányuk miatt az oktatásban is ideális az alkalmazásuk. A VQ100-as tokozás még „házi” körülmények is viszonylag könnyen kezelhető.

4.1.1.1. Tulajdonságok

- Alacsony költségű, nagy teljesítményű logikai megoldások, fogyasztó orientált alkalmazásokhoz: akár 74880 logikai cella.
    - SelectIO interfész
        - akár 633 I/O láb
        - több mint 622Mb/s adatforgalom I/O lábanként
        - 26 I/O szabvány, köztük 8 szimmetrikus (LVDS, RSDS)
        - digitálisan vezérelhető impedancia
        - alkalmazható jelszintek 1.14V-tól 3.465V-ig
        - DDR, DDR2, SDRAM támogatás 333 Mb/s-ig
- Logikai készlet
    - Bőséges mennyiségű logikai cellák shift regiszter képességgel
    - Széles és gyors multiplexerek
    - Gyors, előreszámító átvitel egység (look-ahead carry logic)
    - Dedikált 18x18-as szorzók
    - IEEE 1149.1/1532 kompatibilis JTAG logika
- SelectRAM hierarchiájú memória
    - akár 1,872 Kbit blokk RAM
    - akár 520 Kbit megosztott RAM
- Digitális Órajel Kezelő
    - órajel elcsúszás kiküszöbölése
    - frekvencia szintézis
    - magas felbontású fázis csúsztatás
- Nyolc globális órajel vonal és bőséges huzalozási lehetőség
- MicroBlaze és PicoBlaze processzor, PCI, PCI Express PIPE végpont és egyéb IP magok
- Xilinx ISE és WebPACK szoftveres fejlesztőkörnyezetek teljes mértékű támogatása

Eszköz	Kapuk	Logikai cellák	CLB tömb			Különálló RAM	Blokk RAM	Dedikált szorzók	DCM	Maximálisan felhasználható I/O láb	Maximális szimmetrikus I/O párok
			sor	oszlop	összes						
XC3S50	50k	1 728	16	12	192	12kb	72k	4	2	124	56
XC3S200	200k	4 320	24	20	480	30kb	216k	12	4	173	76
XC3S400	400k	8 064	32	28	896	56kb	288kb	16	4	264	116
XC3S1000	1M	17 280	48	40	1 920	120kb	432kb	24	4	391	175
XC3S1500	1,5M	29 952	64	52	3 328	208kb	576kb	32	4	487	221
XC3S2000	2M	46 080	80	64	5 120	320kb	720kb	40	4	565	270
XC3S4000	4M	62 208	96	72	6 912	432kb	1 728kb	96	4	633	300
XC3S5000	5M	74 880	104	80	8 320	520kb	1 872kb	104	4	633	300

4.1.1.2. Architektúrai áttekintés

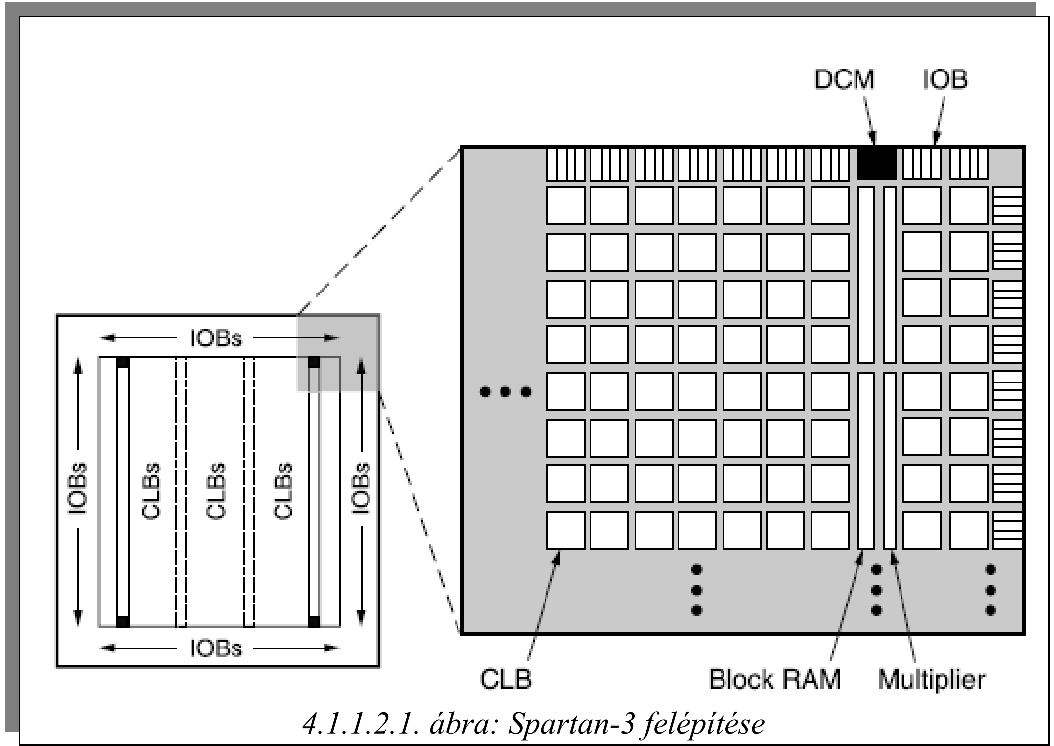
A Spartan-3 család architektúrája 5 alapvető programozható funkcionális elemből áll:

- Konfigurálható Logikai Blokkok (CLB – Configurable Logic Block): RAM-alapú Look-Up Táblákat (LUT) tartalmaznak, hogy logikai és tároló elemeket alkossanak, amiket flip-flop-oknak vagy latch-eknek lehet használni. A CLB-k széles körű logikai feladatra programozhatóak, például kódolás, adattárolás.
- Bemeneti/Kimeneti Blokkok (IOB – Input/Output Block): irányítják az adat áramlását az I/O lábak és az eszköz belső logikája között. Mindegyik IOB képes a 2 irányú adatforgalomra és a tristate[21] működésre. 26 különböző jelszabvány támogatott, köztük 8 szimmetrikus. A Digitálisan Vezérelhető Impedancia (DCI – Digitally Controlled Impedance) funkció chip-beli lezárásokat, illesztéseket, fel-lehúzásokat biztosít, egyszerűsítve ezzel a nyáktervet.
- Blokk RAM (BRAM): adattárolást biztosít 18kb-es dual-port-os blokkok formájában.
- Szorzók: két 18-bites számot fogadnak a bemeneteikre, és kiszámolják a szorzatukat.
- Digitális Órajel Kezelők (DCM – Digital Clock Manager): önkalibráló, teljesen digitális megoldást nyújtanak az órajel elosztására, késleltetésére, szorzására, osztására és a fázis csúsztatására.

Ezen elemek elrendezését a 4.1.1.2.1. ábra mutatja. Egy IOB-okból álló gyűrű veszi körül a CLB-k egy szabályos tömbjét. Az XC3S50 tömbjében egy oszlop BRAM van beágyazva. Az XC3S200-tól az XC3S2000-ig kettő, felette 4 oszlop BRAM-ot tartalmaznak az eszközök. Mindegyik oszlop számos 18kb-es RAM blokkot tartalmaz; mindegyik blokk kapcsolódik egy dedikált szorzóhoz. A DCM-k a BRAM oszlopok külsején vannak elhelyezve.

A Spartan-3 család huzalok és kapcsolók gazdag hálózatával rendelkezik, amelyek összekapcsolják mind az 5 funkcionális elemet, és biztosítják az adatáramlást. Minden funkcionális elemnek van egy kapcsoló mátrixa, amely engedélyezi a huzalozáshoz való csatlakozását.

21 háromállapotú



4.1.1.2.1. ábra: Spartan-3 felépítése

4.1.1.3. Konfiguráció

A Spartan-3 FPGA-ba a bitminta beírása úgy történik, hogy a konfigurációs adat az újraírható statikus CMOS latch-ekbe (CCL) töltődik, ami egységesen irányít minden funkcionális elemet és huzalozási eszközt. A FPGA bekapcsolása előtt a konfigurációs adat egy külső PROM-ban, vagy valamilyen más nemfelejtő köztes eszközben tárolódik, akár a panelen, vagy akár azon kívül. Bekapcsolás után ez az FPGA-ba töltődik az öt különböző mód valamelyikével[22]:

- Párhuzamos[23] Master
- Párhuzamos Slave
- Soros Master
- Soros Slave
- **Boundary Scan[24]** (JTAG)

22 Az M0, M1, M2 konfigurációt vezérlő lábak logikai szintjétől függően.

23 A Párhuzamos módok egy 8 bit széles SelectMAP portot használnak.

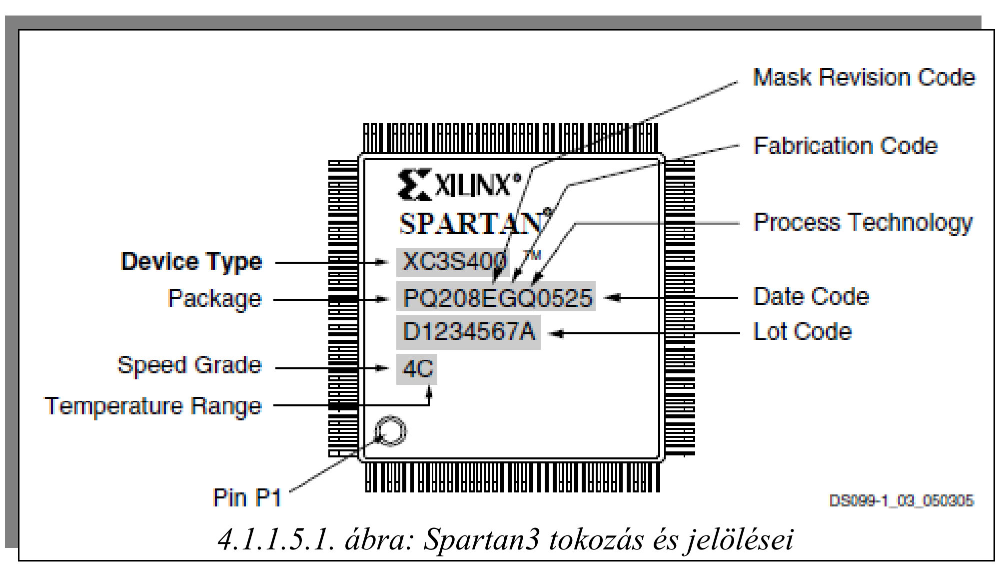
24 Peremfigyeléses technológia

4.1.1.4. I/O képességek

A Spartan-3 eszközök SelectIO tulajdonsága 18 asszimetrikus, és 8 szimmetrikus szabványt támogat. Valamennyi szabvány támogatja a DCI-t, amely integrált lezárásokat tartalmaz a nem kívánt jelvisszaverődések leválasztásának érdekében. A digitálisan vezérelt impedancia segítségével nagy sebességű kommunikációs szabványoknak is eleget tudunk tenni. Példaként említhetjük az RS-485, a koaxiális, vagy az optikai adatátvitelt.

4.1.1.5. Tokozás jelölései

A 4.1.1.5.1. ábra a **quad-flat** tokozás tetején lévő jelzéseket magyarázza.



4.1.1.5.1. ábra: Spartan3 tokozás és jelölései

Eszköz típus	Tokozás[25]		Sebesség fokozat	Hőmérséklet tartomány
XC3S50	VQ(G)100	100 lábú VQFP	-4: normál teljesítményű	C: (Commercial – Kereskedelmi) $T_j = 0 \div 85°C$
XC3S200	CP(G)132[1]	132 lábú CSP		
XC3S400	TQ(G)144	144 lábú TQFP		
XC3S1000	PQ(G)208	208 lábú PQFP		
XC3S1500	FT(G)256	256 lábú FTBGA		
XC3S2000	FG(G)320	320 lábú FBGA	-5[2]: nagy teljesítményű	I: (Industrial – Ipari) $T_j = -40 \div 100°C$
XC3S4000	FG(G)456	456 lábú FBGA		
XC3S5000	FG(G)676	676 lábú FBGA		
	FG(G)900	900 lábú FBGA		
	FG(G)1156[1]	1156 lábú FBGA		

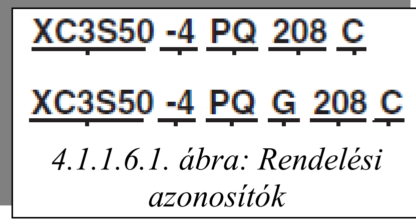
[1] A tokozások a közeljövőben megszűnnek, ezért új fejlesztésekben a használatuk nem javasolt.
[2] A -5 sebességű kizárólag kereskedelmi kivitelben kapható.

25 A tokozásokhoz segítséget a http://www.latticesemi.com/lit/docs/package/pkg.pdf?jsessionid=f0304e08611b41e14cb3c5928571e6e5f502 oldalon találhatunk.

Néhány eszköz lehet dual jelzésű sebesség és hőmérsékleti beosztás szempontjából. Ilyenkor a két érték (pl. ’5C’ és ’4I’) együtt szerepel a tokozáson (’5C/4I’). Ebben az esetben mindkettő szerint használható az eszköz. A single jelzésű eszközök kizárólag az adott sebesség osztályban, az adott hőmérsékleti skála körülményei között működnek.

Néhány specifikáció eltérő lehet a maszk ellenőrző kód függvényében. Az E jelzésű eszközök hibajegyzék mentesek (2006 óta minden kiadott eszköz E kóddal rendelkezik).

4.1.1.6. Rendelési információk



4.1.1.6.1. ábra: Rendelési azonosítók

Bármely Spartan-3 családba tartozó FPGA rendelhető ólom-mentes kiszerelésben. Ezeket a rendelési kódban található G betűvel különböztetik meg a normál kiszereléstől.

4.1.2. Működés leírása

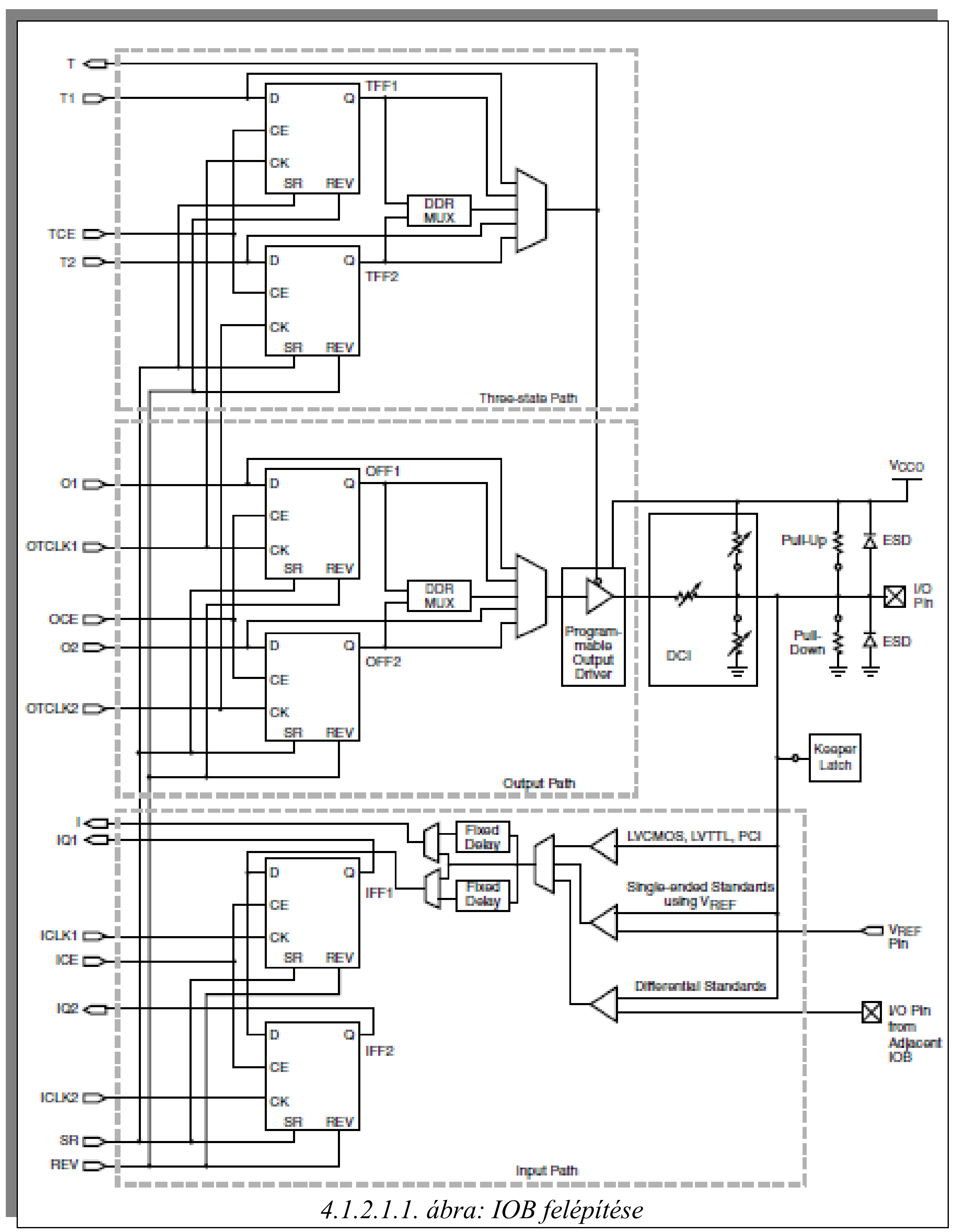
4.1.2.1. IOB-k

Áttekintés

Az Input/Output Blokk egy programozható, kétirányú felületet nyújt az I/O lábak és az FPGA belső logikája között.

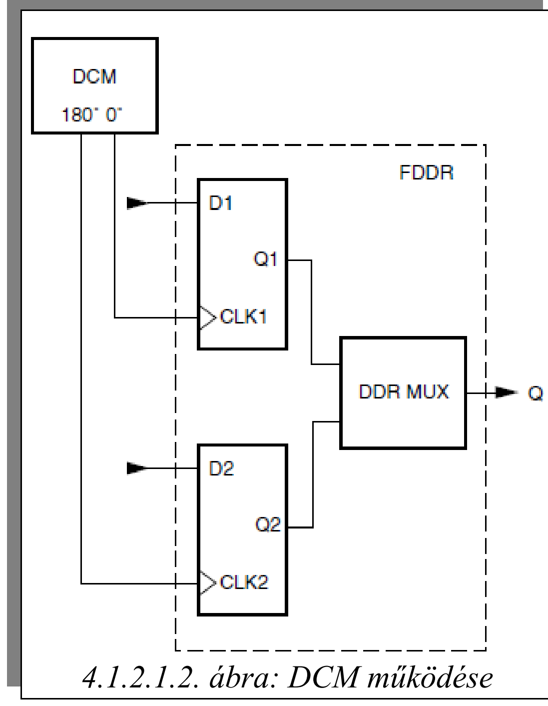
Az IOB egy egyszerűsített blokkvázlata az 4.1.2.1.1. ábrán látható. Három fő útvonal van az IOB-ben: bemeneti út, kimeneti út és a tristate út. Mindhárom útvonalnak van egy tárolóelem párja, amelyek használhatóak regiszterekként, vagy latch-ekként.
A három fő útvonal jellemzői:

- A bemeneti út adatot szállít egy I/O lábtól egy opcionálisan programozható késleltetésen keresztül közvetlenül az I vonalra. Vannak alternatív utak a tárolópáron keresztül az IQ1 és IQ2 vonalakra. Az I, IQ1 és IQ2 vonalak mind az FPGA belső logikájához vezetnek.
- A kimeneti út az O1 és O2 vonalakkal indul, az FPGA belső logikájából visz adatot egy multiplexeren és egy tristate driveren keresztül egy I/O lábhoz. Ezen kívül, a multiplexer lehetőséget biztosít egy tárolópár beillesztésére.
- A tristate út határozza meg, mikor van a kimeneti driver nagyimpedanciás állapotban. A T1 és T2 vonalak az FPGA belső logikájától viszik az adatot egy multiplexeren keresztül a kimeneti meghajtóhoz. Ezen kívül, a multiplexer lehetőséget biztosít egy tárolópár beillesztésére. Amikor a T1 vagy T2 vonal magas szinten van, a kimeneti driver nagyimpedanciás. A meghajtó aktív-nullás.
- Minden jelvonal, ami az IOB-be fut, rendelkezik egy invertáló funkcióval. Minden inverter, ami az útvonalhoz adódik, automatikusan az IOB-be épül.



4.1.2.1.1. ábra: IOB felépítése

Tárolóelem funkció



4.1.2.1.2. ábra: DCM működése

Három tárolóelem pár található minden IOB-ben, egy pár minden útvonalon. Ezek vagy élvezérelt D-flip-flop-oknak (FD), vagy szint-érzékeny latch-eknek (LD) konfigurálhatóak.

A kimeneti és a tristate útvonal tárolói összeköthetőek, így egy speciális multiplexerként működnek, amivel DDR (Double-Data-Rate – Dupla Adat Arány) átvitel valósítható meg. Ez úgy történik, hogy a működés nem csak az órajel fel-, vagy lefutó élére történik, hanem az órajel fel-, és lefutó élére is megtörténik a végrehajtás.

A kimeneti és a tri-state úton ehhez két – 50%-os kitöltésű – órajel szükséges, egyik az invertáltja a másiknak. Ezek a jelek felváltva triggerelik a regisztereket. (4.1.2.1.2. ábra) A DCM ezt úgy csinálja meg, hogy a bejövő jelet tükrözi, vagyis eltolja 180°-kal. A bemeneten a bejövő órajel működteti az egyik tárolóregisztert, az órajel invertáltja a másikat, így a bementi tárolóelem pár felváltva fogadja a biteket.

Symbol	Description	Test Conditions	Min	Typ	Max	Units
I_L (2)	Leakage current at User I/O, Dual-Purpose, and Dedicated pins	Driver is Hi-Z, V_{IN} = 0V or V_{CCO} max, sample-tested; $V_{CCO} \geq 3.0V$	-	-	±25	μA
		$V_{CCO} < 3.0V$	-	-	±10	μA
I_{RPU} (3)	Current through pull-up resistor at User I/O, Dual-Purpose, and Dedicated pins	V_{IN} = 0V, V_{CCO} = 3.3V	–0.84	-	–2.35	mA
		V_{IN} = 0V, V_{CCO} = 3.0V	–0.69	-	–1.99	mA
		V_{IN} = 0V, V_{CCO} = 2.5V	–0.47	-	–1.41	mA
		V_{IN} = 0V, V_{CCO} = 1.8V	–0.21	-	–0.69	mA
		V_{IN} = 0V, V_{CCO} = 1.5V	–0.13	-	–0.43	mA
		V_{IN} = 0V, V_{CCO} = 1.2V	–0.06	-	–0.22	mA
R_{PU} (3)	Equivalent resistance of pull-up resistor at User I/O, Dual-Purpose, and Dedicated pins, derived from I_{RPU}	V_{CCO} = 3.0V to 3.465V	1.27	-	4.11	kΩ
		V_{CCO} = 2.3V to 2.7V	1.15	-	3.25	kΩ
		V_{CCO} = 1.7V to 1.9V	2.45	-	9.10	kΩ
		V_{CCO} = 1.4V to 1.6V	3.25	-	12.10	kΩ
		V_{CCO} = 1.14 to 1.26V	5.15	-	21.00	kΩ
I_{RPD} (3)	Current through pull-down resistor at User I/O, Dual-Purpose, and Dedicated pins	V_{IN} = V_{CCO}	0.37	-	1.67	mA
R_{PD} (3)	Equivalent resistance of pull-down resistor at User I/O, Dual-Purpose, and Dedicated pins, driven from I_{RPD}	V_{IN} = V_{CCO} = 3.0V to 3.465V	1.75	-	9.35	kΩ
		V_{IN} = V_{CCO} = 2.3V to 2.7V	1.35	-	7.30	kΩ
		V_{IN} = V_{CCO} = 1.7V to 1.9V	1.00	-	5.15	kΩ
		V_{IN} = V_{CCO} = 1.4V to 1.6V	0.85	-	4.35	kΩ
		V_{IN} = V_{CCO} = 1.14 to 1.26V	0.68	-	3.465	kΩ
R_{DCI}	Value of external reference resistor to support DCI I/O standards		20	-	100	Ω
I_{REF}	V_{REF} current per pin	$V_{CCO} \geq 3.0V$	-	-	±25	μA
		$V_{CCO} < 3.0V$	-	-	±10	μA
C_{IN}	Input capacitance		3	-	10	pF

4.1.2.1.3. ábra: Sink és load áramok, impedanciák fel-, ill. lehúzó ellenállások használata esetén

Felhúzó és lehúzó ellenállások

A felhúzó és lehúzó ellenállások feladata a magas vagy alacsony alapszint biztosítása az I/O lábakon. A felhúzó ellenállások az I/O lábakat a V_{CCO}-ra csatolják. A lehúzó ellenállások az I/O lábakat a GND-re csatlakoztatják. Ezeket az ellenállásokat a kapcsolási rajzon lehet beilleszteni, vagy egy hardverleíró nyelv segítségével megadni. A Spartan-3 fel-, és lehúzó ellenállásai jelentősen erősebbek, mint a megelőző Xilinx FPGA családok ellenállásai (4.1.2.1.3. ábra).

Tartó áramkör

Minden I/O láb rendelkezik egy opcionális tartó áramkörrel, ami a legutolsó szintet tartja, miután már minden driver ki lett kapcsolva. A funkció hasznos, hogy ne lebegjen egy láb se, mikor minden driver nagyimpedanciás állapotban van. Ezt is a kapcsolási rajzon lehet beállítani. A húzóellenállások felülírják a tartóáramkört.

Slew rate

A slew-rate a kimeneti jel változásának sebessége. A Xlinx két lehetőséget biztosít számunkra, a FAST (gyors) és a SLOW (lassú) opciót. A minél nagyobb slew-rate (fast) nagyfrekvenciás működést tesz lehetővé. A slow opció csökkenti a buszon megjelenő tranzienseket. Ezek a lehetőségek csak az LVCMOS és LVTTL szabványok használatánál állnak rendelkezésre. Az LVCMOS és LVTTL szabványok hét különböző szintű driver áramerősséggel rendelkeznek. A megfelelő kiválasztásával szintén csökkenthető a buszon az átmeneti jelenségek okozta zavar.

Jel szabvány (IOSTANDARD)	Driver áramerőssége (mA)						
	2	4	6	8	12	16	24
LVTTL	✓	✓	✓	✓	✓	✓	✓
LVCMOS33	✓	✓	✓	✓	✓	✓	✓
LVCMOS25	✓	✓	✓	✓	✓	✓	✓
LVCMOS18	✓	✓	✓	✓	✓	✓	-
LVCMOS15	✓	✓	✓	✓	✓	-	-
LVCMOS12	✓	✓	✓	-	-	-	-

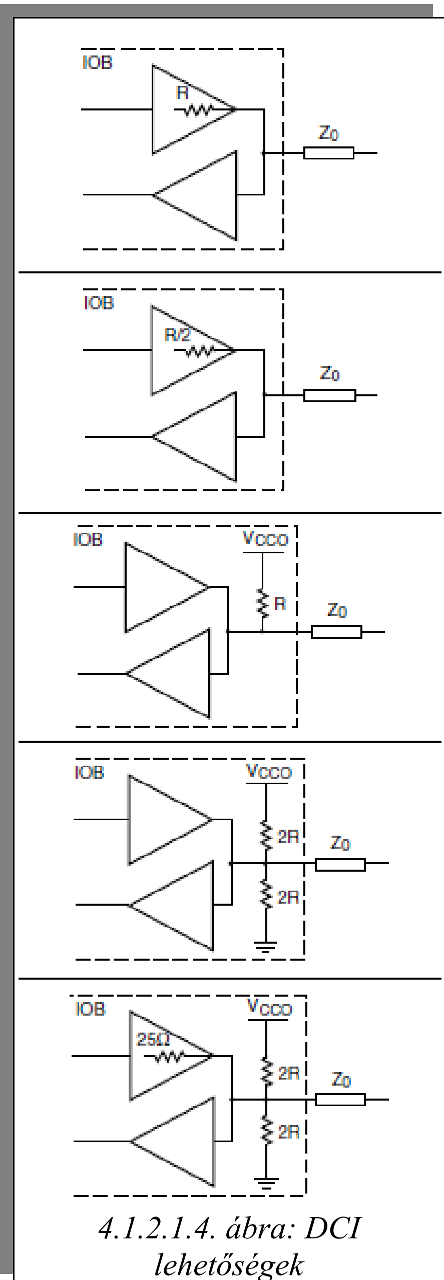
Boundary Scan

Minden Spartan-3 eszköz támogatja az IEEE 1149.1 szabvány szerinti Boundary Scan tesztelést. Peremfigyeléses műveletek közben (EXTEST, HIGHZ stb.) az I/O-k lehúzó ellenállásai aktívak. A peremfigyeléses technika lényege, hogy 4(5) vezetéken keresztül tudjuk a nagy lábszámú eszközöket, vagy akár összetett rendszereket is tesztleni, ill. felkonfigurálni. Részletesebben a peremfigyelésről az alábbi linken olvashatunk: http://hadmernok.hu/archivum/2007/4/2007_4_molnar.html

Digitálisan Vezérelhető Impedancia (DCI)

A DCI kétféle chipbe épített lezáró ellenállással rendelkezik. A párhuzamos lezárás egy integrált ellenállás hálózat. A soros lezárás a kimeneti meghajtók impedanciájának állításával képződik. A DCI mindkettőt használja, annak érdekében, hogy pontosan beállítsa az átviteli vonal karakterisztikus impedanciáját.

A DCI-t 5-féle módon lehet beállítani:



4.1.2.1.4. ábra: DCI lehetőségek

Beállított impedanciájú kimeneti meghajtó:

- LVDCI_15
- LVDCI_18
- LVDCI_25
- LVDCI_33
- HSLVDCI_15
- HSLVDCI_18
- HSLVDCI_25
- HSLVDCI_33

Beállított kimeneti driver fél impedanciával:

- LVDCI_DV2_15
- LVDCI_DV2_18
- LVDCI_DV2_25
- LVDCI_DV2_33

Egyetlen ellenállás:

- GTL_DCI
- GTLP_DCI
- HSTL_III_DCI
- HSTL_III_DCI_18

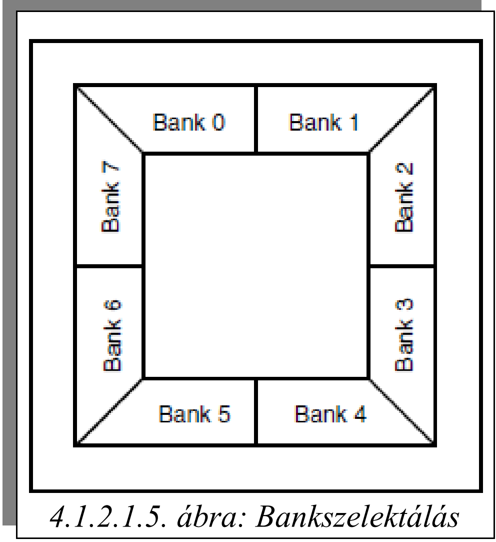
Ellenállás osztó:

- HSTL_I_DCI
- HSTL_I_DCI_18
- HSTL_II_DCI_18
- DIFF_HSTL_II_18_DCI
- DIFF_SSTL2_II_DCI
- LVDS_25_DCI
- LVDSEXT_25_DCI

Ellenállás osztó fix 25Ω-os impedanciájú kimeneti meghajtóval:

- SSTL18_I_DCI
- SSTL2_I_DCI
- SSTL2_II_DCI

IOB-k bankokba rendezése



4.1.2.1.5. ábra: Bankszelektálás

Az IOB-k nyolc bankba vannak rendezve, tehát az eszköz mind a négy oldalán kettő bank található (4.1.2.1.5. ábra). Minden tokozásban az összes bank különálló V_{REF} vonallal rendelkezik.

A TQ144 és a CP132 tokozásban az azonos oldalakon lévő bank-párok V_{CCO} vonalai közösítve vannak. A külön oldalak külön V_{CCO} vonalakat kapnak, tehát az eszköz négy különálló V_{CCO} vonallal rendelkezik. Más tokozásoknál minden banknak külön V_{CCO} vonala van. Ez azt jelenti, hogy minden Bank különböző feszültségszintekkel tud dolgozni.

Spartan-3 FPGA kompatibilitás

A Spartan-3 családon belüli FPGA-k tokozás szerint láb-kompatibilisek. Amikor a fejlesztés során a jelenleg használt FPGA már nem elégíti ki a szükségletet, egy nagyobb eszköz szolgálhat közvetlen cserére. A nagyobb eszközök esetleg több V_{CCO} és V_{REF} lábbal rendelkeznek a nagyobb I/O lábszám miatt. A nagyobb eszközökben több felhasználható I/O lábat lehet V_{REF} vonallá alakítani. Másrészt plusz V_{CCO} vonalak vannak lábakra kivezetve, amik nem voltak a kisebb eszköznél csatlakoztatva. Fontos, hogy a jövőbeli terveknél ez korrigálva legyen.

A Spartan-3 családból származó FPGA-k nem kompatibilisek a Xilinx korábbi FPGA családjaiból származó eszközökkel.

Bankokra vonatkozó szabályok

I/O lábak bankokhoz való rendelésénél fontos, hogy a következő V_{CCO} szabályok betartása:

- Minden V_{CCO} lábat csatlakoztatni kell.[1]
- Azonos bankhoz tartozó V_{CCO} lábnak azonos feszültségszinten kell lennie.[1]
- A V_{CCO} lábak feszültségszintje meg kell feleljen a hozzárendelt I/O lábak jel szabványának. A Xilinx fejlesztőszoftvere ellenőrzi ezt.
- Ha egy I/O szabványa se használja a V_{CCO} vonalat, minden V_{CCO} lábat 2,5V-ra kell kötni.

[1] Ezek a szabályok a V_{REF} vonalakra is vonatkoznak.

Bankok I/O szabványaira vonatkozó kivételek

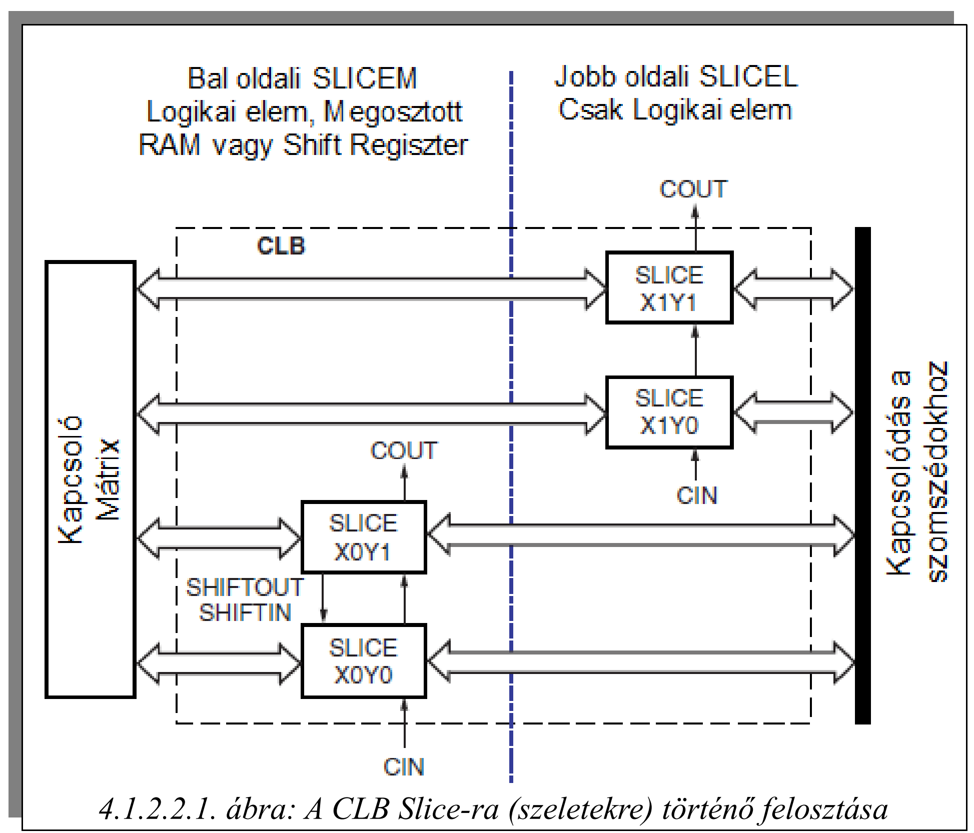
A VQ100, a CP132 és a TQ144 tokozásban az 5-ös bank nem támogatja a DCI szabványait. Emiatt ezen tokozásoknál az 5-ös banknak nincsen VRN és VRP kivezetése. Továbbá a VQ100 tokozásnál a 4-es és 5-ös bank nem támogatja azon jelszabványokat, amelyek a V_{REF} vonalat használják, ezáltal ezek a bankok nem is rendelkeznek V_{REF} kivezetéssel.

IOB-k tápja
Három különböző tápfeszültséget kapnak az IOB-k:

- Minden I/O bank külön V_{CCO} vonalat kap, ez táplálja a kimeneti meghajtókat is, kivéve a GTL és GTLP jelszabványoknál. A V_{CCO} vonal feszültségszintje meghatározza a kimenetek feszültségingadozását.
- A V_{CCINT} (2,5V) vonal a fő tápja az FPGA belső logikájának.
- Ezen kívül egy plusz V_{CCAUX} (1,2V) vonal optimalizálja a teljesítményét különböző funkcióknak, például az I/O váltóknak.

4.1.2.2. CLB áttekintés

A konfigurálható logikai blokkok (CLB – Configurable Logic Block) alkotják a fő elemeit a szinkron és kombinációs hálózatoknak. Minden CLB négy Slice-ból (szelet) áll (7. ábra). A Slice-ok párokba vannak rendezve. Minden páros olyan, mint egy oszlop egy független jeltovábbító lánccal.



4.1.2.2.1. ábra: A CLB Slice-ra (szeletekre) történő felosztása

Elemek a Slice-okon belül
Mind a négy Slice tartalmazza a következő elemeket: 2 logikai függvénygenerátor, 2 tároló elem, széles funkciójú multiplexerek, carry logika, és aritmetikai kapuk. Mind a jobb és bal oldali Slice-ok ezeket használják a logikai, aritmetikai és ROM funkciók elvégzésére. Ezeken kívül a baloldali páros két plusz funkcióval rendelkezik: adattárolás megosztott RAM segítségével, és adat léptetés 16-bites regiszterekkel.

A RAM alapú függvénygenerátor – Look-Up Tábla (LUT) néven is ismert – a fő eszköze a logikai funkciók megvalósításának. Továbbá a baloldali Slice párokban a LUT-ok konfigurálhatóak megosztott RAM-ként, vagy 16-bites shiftregiszterként.

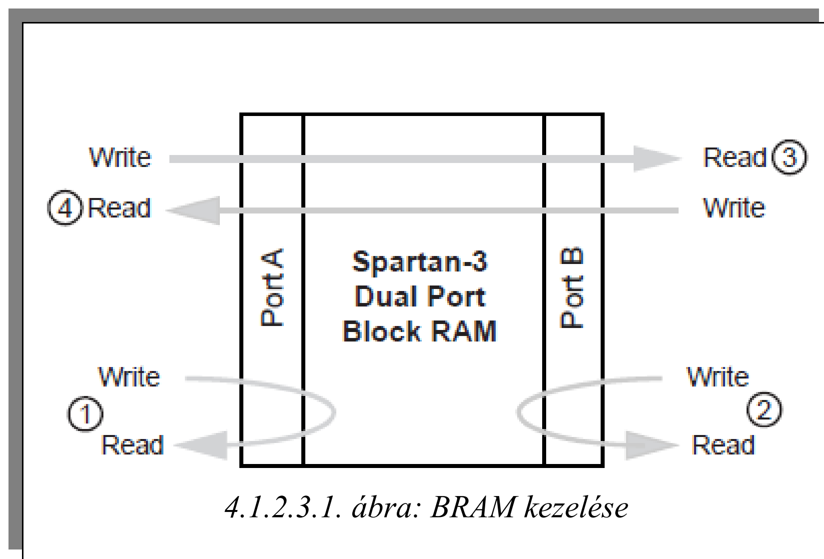
4.1.2.3. Blokk RAM áttekintés

Minden Spartan-3 eszköz biztosít Blokk RAM-okat, amik konfigurálható, szinkron, 18kbites blokkokba vannak rendezve. A BRAM viszonylag nagy méretű adat tárolására képes, sokkal hatékonyabban, mint a megosztott RAM-ok.

Az oldalviszonyok (szélesség vs. mélység) minden BRAM-nál beállíthatóak, ezáltal több blokk RAM-ra lehet egyet bontani, így több, azonos mélységű és/vagy magasságú RAM-ot kapunk.

Az XC3S50 tömbjében egy oszlop BRAM van beágyazva. Az XC3S200-tól az XC3S2000-ig kettő, felette 4 oszlop BRAM-ot tartalmaznak az eszközök.

A BRAM belső struktúrája



4.1.2.3.1. ábra: BRAM kezelése

A Blokk RAM dual-port struktúrájú. A két azonos adatport (Port A és Port B) külön elérhetőséget kap a közös BRAM-hoz, aminek a maximum kapacitása 18842 bit. Mindkét portnak megvan a saját, dedikált adat, vezérlő és órajel vonal készlete műveletek szinkron olvasásához és írásához. Négy fő adatút van (8. ábra), adat beírás és kiolvasás Port A-ból (1), adat beírás és kiolvasás Port B-ből (2), adatátvitel Port A-ból Port B-be (3), adatátvitel Port B-ből Port A-ba (4).

4.1.2.4. Dedikált szorzók

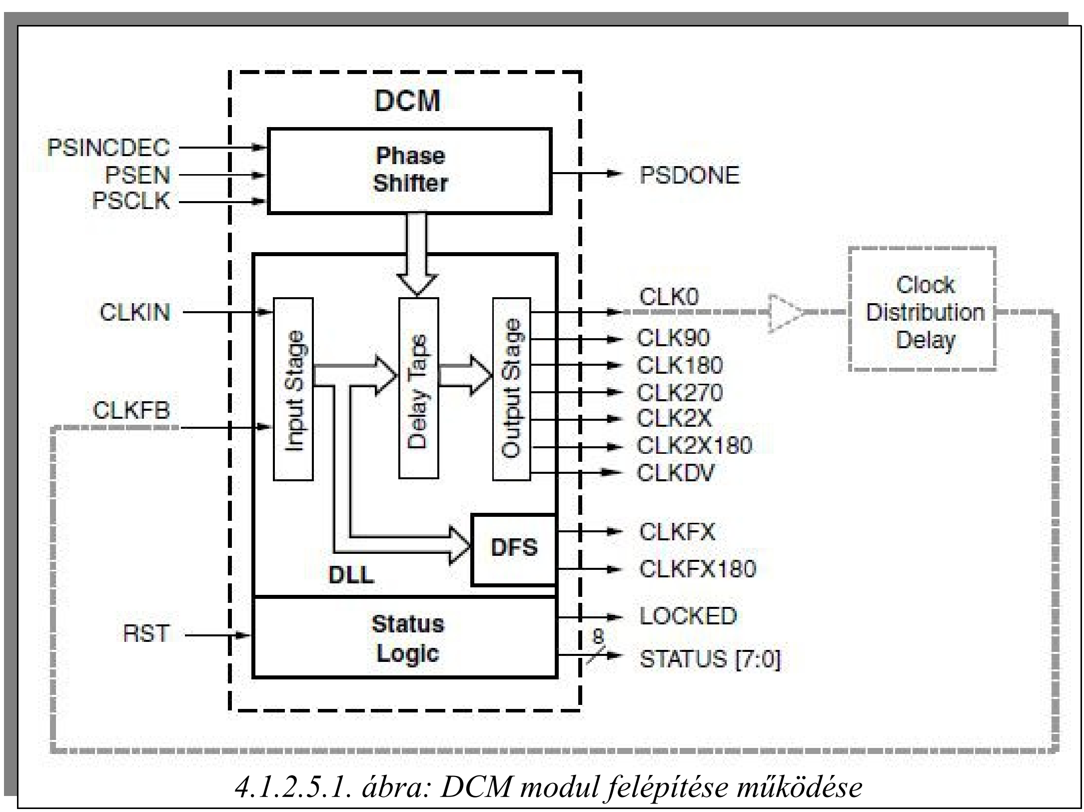
Minden Spartan-3 eszköz rendelkezik beágyazott szorzókkal, amelyek két 18-bites adatból csinálnak egy 36-bites szorzatot. A bemeneti buszok kettes komplemensű formában fogadják az adatokat (18-bites előjeles, vagy 17-bites előjel nélküli). Minden szorzó kapcsolódik egy BRAM-hoz. A két eszköz fizikai közelsége biztosítja a megbízható adatkezelést. A szorzókat a kapcsolási rajzon lehet beilleszteni.

4.1.2.5. Digitális Órajel Kezelő

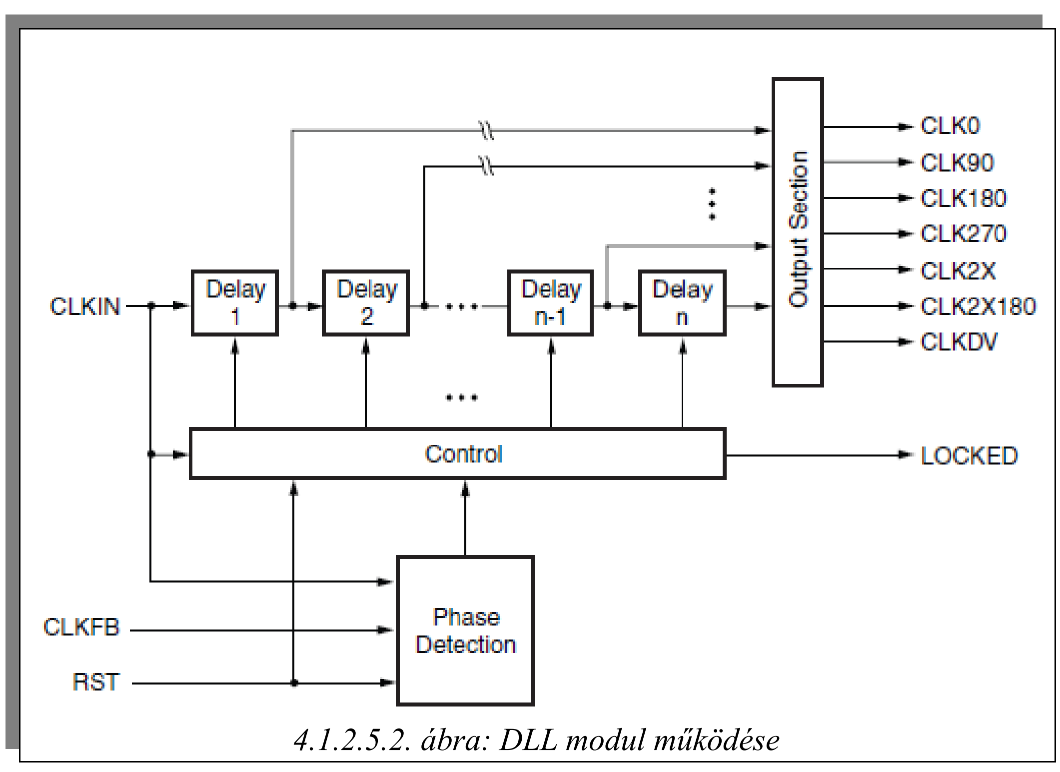
A Spartan-3 eszközök az órajel rugalmas és teljes irányítását teszik lehetővé a DCM (Digital Clock Manager) funkció segítségével. Ehhez a DCM (4.1.2.5.1. ábra) egy DLL (Delay-Locked Loop) (4.1.2.5.2. ábra) rendszert használ. Ez egy teljesen digitális irányító rendszer, ami visszacsatolást használ az órajel karakterisztikájának nagy pontosságú kezelésére a működési hőmérséklet és feszültség értékétől függetlenül. A Spartan-3 család minden tagja 4 DMC-et tartalmaz, kivéve az XC3S50-et, az csak 2-t. A DCM-eket a kapcsolási rajzon lehet beilleszteni.

A DCM három fő funkciója:

- Órajel meredekség növelése
- Frekvencia szintézis
- Fáziseltolás



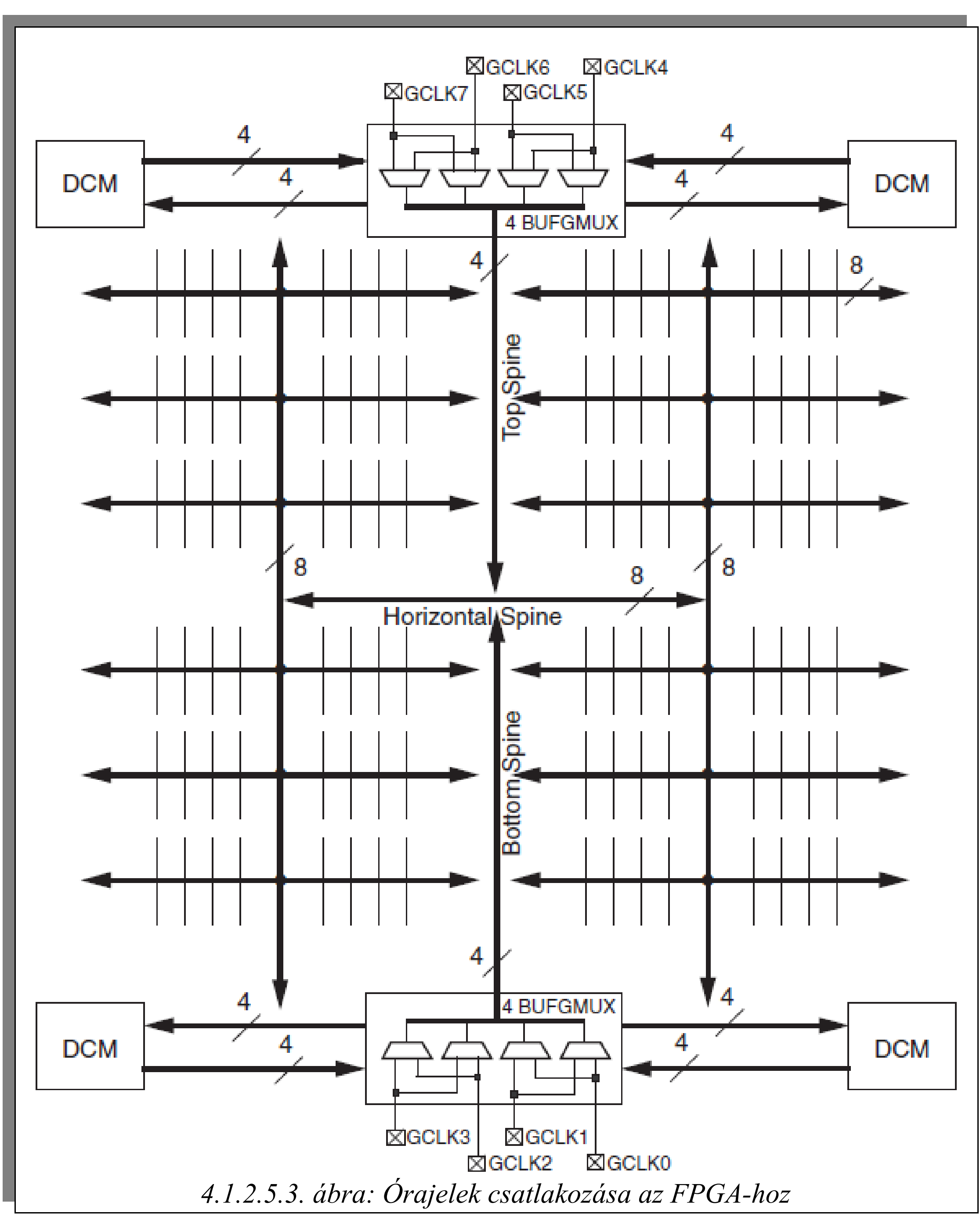
4.1.2.5.1. ábra: DCM modul felépítése működése



4.1.2.5.2. ábra: DLL modul működése

Globalis Órajel Hálózat

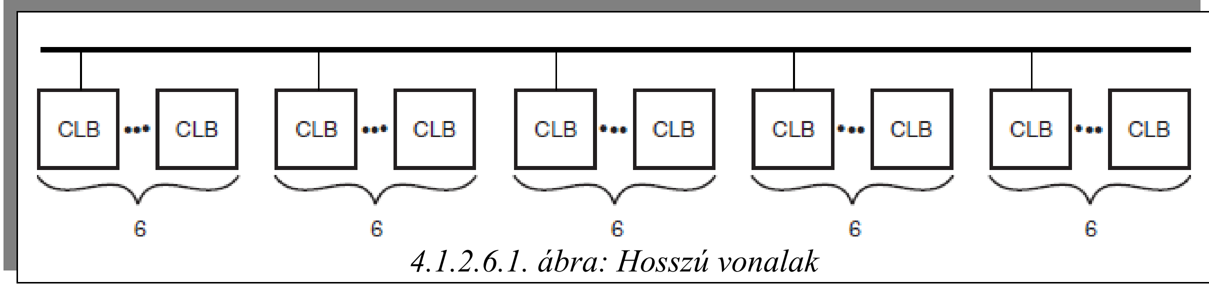
A Spartan-3 eszközök 8 globális órajel bemenettel rendelkeznek (GCLK0÷GCLK7). Ezek a bemenetek egy alacsony kapacitású hálózatra csatlakoznak, amely tökéletes nagyfrekvenciás jelek vezetésére. A GCLK0÷GCLK3 bemenetek a felső oldal közepén, a GCLK4÷GCLK7 bemenetek az alsó oldal alján vannak elhelyezve (4.1.2.5.3. ábra). Nyolc Globális Órajel Multiplexer (BUFGMUX) kapcsolja a GCLK bemeneteket a belső órajel hálózatra és a DCM-ekre (4 a felső, 4 az alsó oldalon). Azt hogy melyik GCLK vonal kerül a BUFGMUX kimenetére az S útválasztó vonal határozza meg.



4.1.2.5.3. ábra: Órajelek csatlakozása az FPGA-hoz

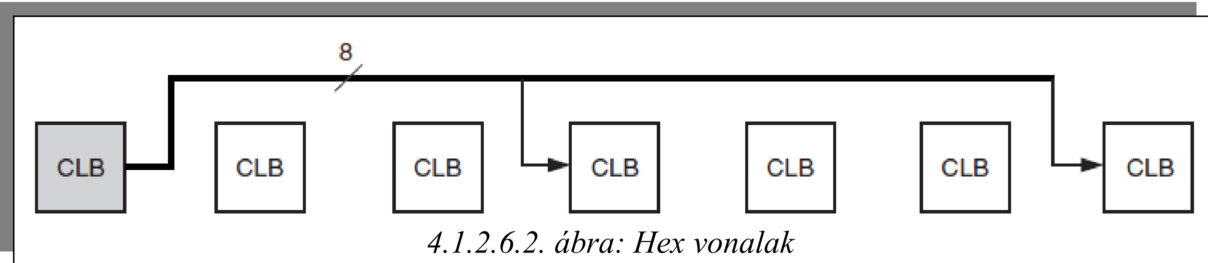
4.1.2.6. Huzalozás

A huzalozás segítségével haladnak az adatok a különböző funkcionális elemek között. Négyféle huzalozás létezik: hosszú vonalak, hex vonalak, dupla vonalak és direkt vonalak.



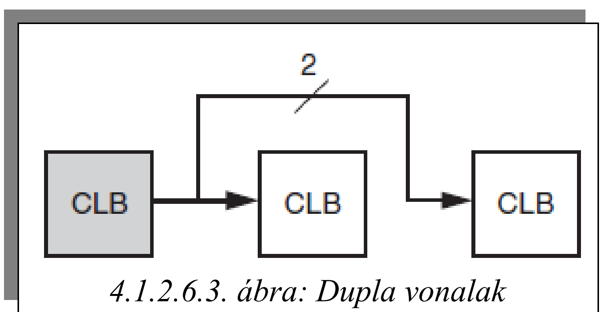
4.1.2.6.1. ábra: Hosszú vonalak

Minden 6 CLB-ből 1-et csatlakoztat. Alacsony kapacitásuk miatt tökéletesen alkalmasak nagyfrekvenciás jelek vezetésére. Ha már mind a 8 Globális Órajel Bemenet használatban van, de még szükség van órajelek használatára, a hosszú vonalak jó alternatívaként szolgának.



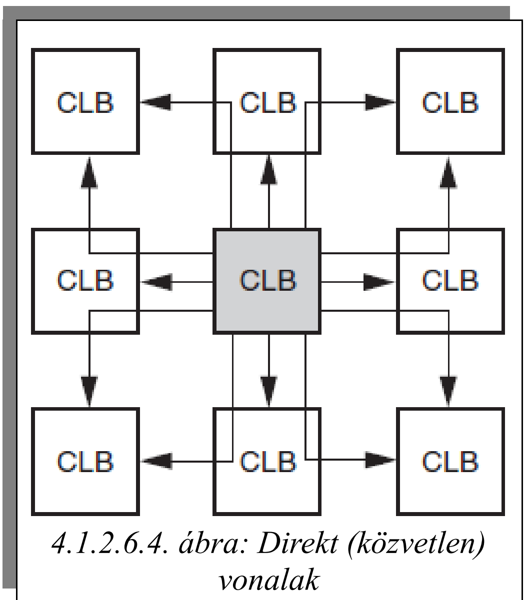
4.1.2.6.2. ábra: Hex vonalak

Minden 3 CLB-ből 1-et csatlakoztat. A kapacitása a Hosszú vonal és a Dupla vonal közé esik. Ezáltal a hosszú vonalnál kisebb frekvenciákon üzemel, azonban több kapcsolódási lehetőséget biztosít. A dupla vonalaknál jobb nagyfrekvenciás tulajdonságokkal rendelkezik, azonban nem biztosít olyan széleskörű útvonalat.



4.1.2.6.3. ábra: Dupla vonalak

Az előbbiekhez képest ez sokkal nagyobb rugalmasságot ad a kapcsolatokban, nagyfrekvenciás tulajdonságai ezzel szemben romlanak.



4.1.2.6.4. ábra: Direkt (közvetlen) vonalak

Minden CLB közvetlenül csatlakozik minden szomszédos CLB-hez. Általában egy forrás CLB-ből direkt vonallal csatlakozik a jel valamely előbbi vonalra, és onnan egy másik direkt vonalon keresztül kerül a jel a cél CLB-be.

4.1.3. Határértékek

Szimbólum	Leírás	Teszt körülmények	Min	Max	Mérték-egység
V_{CCINT}	Belső tápfeszültség[1]		-0,5	1,32	V
V_{CCAUX}	Segéd tápfeszültség[1]		-0,5	3,00	V
V_{CCO}	Kimeneti meghajtó tápfeszültsége[1]		-0,5	3,75	V
V_{REF}	Bemenő referencia feszültség[1]		-0,5	V_{CCO}+ 0,5	V
V_{IN}	Felhasználható I/O és két-célú lábak feszültsége[1]	Kereskedelmi	-0,95	4,4	V
		Ipari[26]	-0,85	4,3	
	Dedikált lábak feszültsége[1]	Minden hőmérséklet skála	-0,5	V_{CCAUX} +0,5	
I_{IK}	Bemenő áram I/O lábanként	$-0,5V < V_{IN} < (V_{CCO}+0,5V)$	-	±100	mA
V_{ESD}	Elektrosztatikus kisülési feszültség a lábakon[1]	Emberi test modell	-	±2000	V
		Feltöltött eszköz modell	-	±500	
		Gép modell	-	±200	
T_J	A kristály hőmérséklete		-	125	°C
T_{SOL}	Forrasztáskor a tok hőmérséklet		-	220	°C
T_{STG}	Tárolási hőmérséklet		-65	150	°C

[1] A GND-hez képest

26 Az ipari kivitelnek szélesebb tartományban kellene működnie, elképzelhető, hogy ezen a ponton hibás az adatlap.

4.1.4. Lábkiosztás

Láb típusa	Leírás	Lábak neve
I/O	Korlátlan, általános felhasználású I/O láb. A legtöbbje párosítható, így szimmetrikus I/O-kat lehet létrehozni.	IO, IO_Lxxy_#
DUAL	Két célú láb, amelyeket konfigurációs módokban használnak, de konfigurálás után felhasználható I/O-ként. Ha nincs használva konfigurálás közben, a láb I/O-ként működik. Minden tokozásban 12 db ilyen láb van. Az INIT_B láb a konfigurálás alatt rendelkezik egy belső felhúzó ellenállással a VCCO_4, vagy a VCCO_BOTTOM lábra.	IO_Lxxy_#/DIN/D0, IO_Lxxy_#/D1, IO_Lxxy_#/D2, IO_Lxxy_#/D3, IO_Lxxy_#/D4, IO_Lxxy_#/D5, IO_Lxxy_#/D6, IO_Lxxy_#/D7, IO_Lxxy_#/CS_B, IO_Lxxy_#/RDWR_B, IO_Lxxy_#/BUSY/DOUT, IO_Lxxy_#/INIT_B
CONFIG	Dedikált konfigurációs láb. Nem használható fel I/O lábnak. Minden tokozásban 7 db ilyen láb van. A tápfeszültséget a VCCAUX vonalról kapják, és rendelkeznek egy belső felhúzó ellenállással a VCCAUX-ra.	CCLK, DONE, M2, M1, M0, PROG_B, HSWAP_EN
JTAG	Dedikált JTAG láb. Nem használható fel I/O lábnak. Minden tokozásban 4 ilyen láb van. A tápfeszültséget a VCCAUX vonalról kapják, és rendelkeznek egy belső felhúzó ellenállással a VCCAUX-ra.	TDI, TMS, TCK, TDO
DCI	Két célú láb, ami felhasználható I/O-nak, vagy a kimeneti buffer impedanciájának kalibrálásához. Bankonként 2 ilyen láb van.	IO/VRN_#, IO_Lxxy_#/VRN_#, IO/VRP_#, IO_Lxxy_#/VRP_#
VREF	Két célú láb, ami felhasználható I/O-nak, vagy az adott bank VREF vonalához csatlakoztatható, így referencia feszültséget adva a használt jelszabványnak. Utóbbi esetben az összeset csatlakoztatni kell az adott bankban.	IO/VREF_#, IO_Lxxy_#/VREF_#
GND	Dedikált földpont láb. Száma a tokozástól függ. Mindet csatlakoztatni kell.	GND
VCCAUX	Dedikált segédfeszültség láb. Számuk a tokozástól függ. Mindet +2,5V-ra kell kötni.	VCCAUX
VCCINT	Dedikált láb, tápfeszültség a belső maglogikának. Számuk a tokozástól függ. Mindet +1,2V-ra kell kötni.	VCCINT
VCCO	Dedikált láb, tápfeszültség az I/O bankok kimeneti puffereinek. Adott I/O bank kimeneti bufferjének tápfeszültségét adja, és meghatározza a bemeneti küszöbfeszültséget néhány I/O szabványnál.	VCCO_# CP132 és TQ144 tokoásoknál: VCCO_LEFT, VCCO_TOP, VCCO_RIGHT, VCCO_BOTTOM
GCLK	Két célú láb, felhasználható I/O-nak, vagy bemenetnek egy speciális globális bemeneti bufferhez (órajel). Minden tokozásban 8 ilyen láb van.	IO_Lxxy_#/GCLK0, IO_Lxxy_#/GCLK1, IO_Lxxy_#/GCLK2, IO_Lxxy_#/GCLK3, IO_Lxxy_#/GCLK4, IO_Lxxy_#/GCLK5, IO_Lxxy_#/GCLK6, IO_Lxxy_#/GCLK7
N.C.	A tokozás ezen lába nincs csatlakoztatva.	N.C.

5. Fejlesztőpanelek

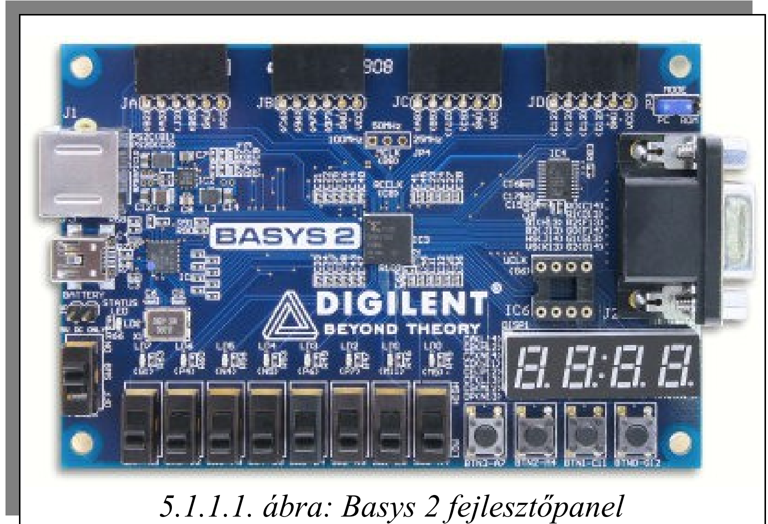
Mielőtt elkezdjük az FPGA-ra történő fejlesztést célszerű megnézni, hogy a piacon milyen eszközök segítik munkánkat. A leírásban a ChipCad cég – a Xilinx hazai disztribútora – ajánlatát, valamint a fejlesztőcsapat által elkészített próbapaneleket vesszük górcső alá.

5.1. Fejlesztőpanelek a ChipCad cég kínálatában

A ChipCad cég a Xilinx FPGA-ra történő fejlesztés elősegítése érdekében a Digilent fejlesztőeszközeit kínálja[27]. A panelek rövid leírását a gyártó honlapja alapján készítjük el. A képek is innen származnak.

5.1.1. Basys™2 Spartan-3E FPGA panel

A 5.1.1.1. ábrán látható Basys™2 fejlesztőpanel egy Xilinx Spartan 3E (100k, 200k) FPGA áramkört és alapperiferiákat tartalmaz.



5.1.1.1. ábra: Basys 2 fejlesztőpanel

A panel számos csatlakozót tartalmaz:

- USB port (2.0)
- VGA (8 bit)
- PS/2
- 4db 6 pólusú Pmod csatlakozó

A programozást JTAG interfészen keresztül tudjuk végrehajtani. A nyákon néhány egyszerű periféria is helyet kapott (kapcsoló, nyomógomb, LED, hétszegmenses kijelző).

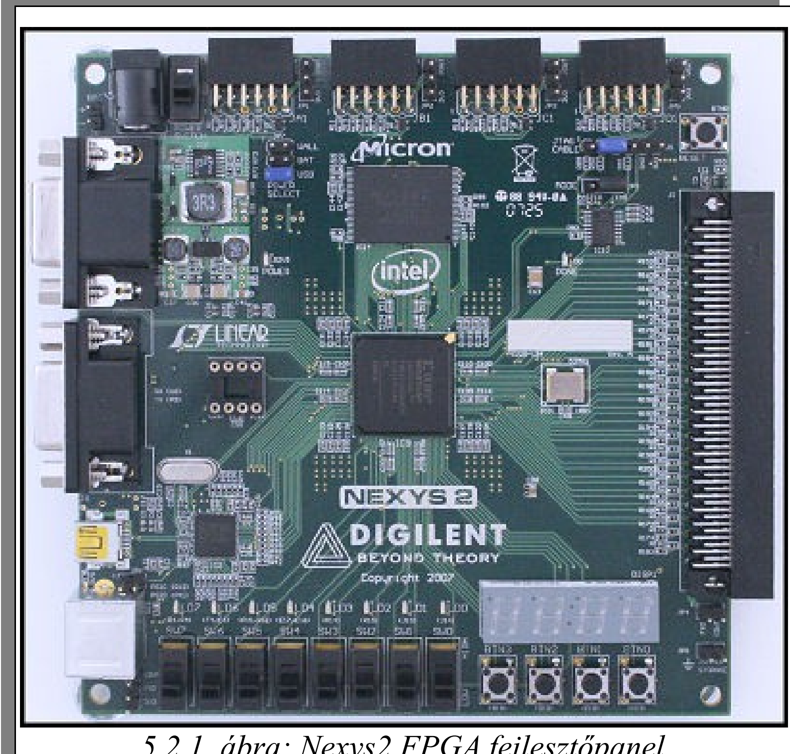
5.1.1.2. ábra: Perifériák elhelyezkedése a Basys2-őn

A fejlesztőpanel kiválóan alkalmas az FPGA-val való megismerkedéshez. A szabványos csatlakozókra kivezetett védelemmel ellátott I/O lábak számos bővítést tesznek lehetővé. Az USB, a VGA és a PS/2 portokkal már komolyabb fejlesztések is végrehajthatóak. A Digilent 3 különböző órajelet (25, 50, 100MHz) biztosít, amelyek a felhasználó által kiválaszthatók – mindemellett a panelhez egy foglalaton keresztül külső oszcillátor is csatlakoztatható. A panelen a konfigurációs bitminta tápellátás nélküli tárolását egy XCF02 Platform Flash teszi lehetővé. A részletes interfészkínálatot a 5.1.1.2. ábra szemlélteti.

27 A ChipCad forgalmaz Xilinx fejlesztőpaneleket is, azonban ezek ár/érték arányát rosszabbnak ítéltük, mint a Digilent termékeket.

5.2. Nexys™2 Spartan 3E FPGA fejlesztőpanel

A 5.2.1. ábrán látható áramkör egy Xilinx Spartan 3E (500k, 1200k) chipet és középkategóriás perifériákat tartalmaz.



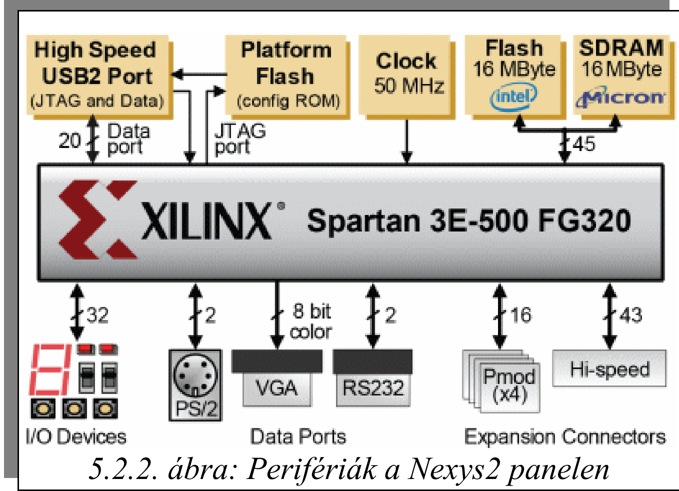
5.2.1. ábra: Nexys2 FPGA fejlesztőpanel

A nyák a következő csatlakozásokkal van ellátva:

- USB
- RS232
- VGA
- PS/2
- Hirose FX2
- 4db 6 pólusú Pmod csatlakozó

A programozás, az adatátvitel, a tápellátás USB csatlakozón keresztül történik.

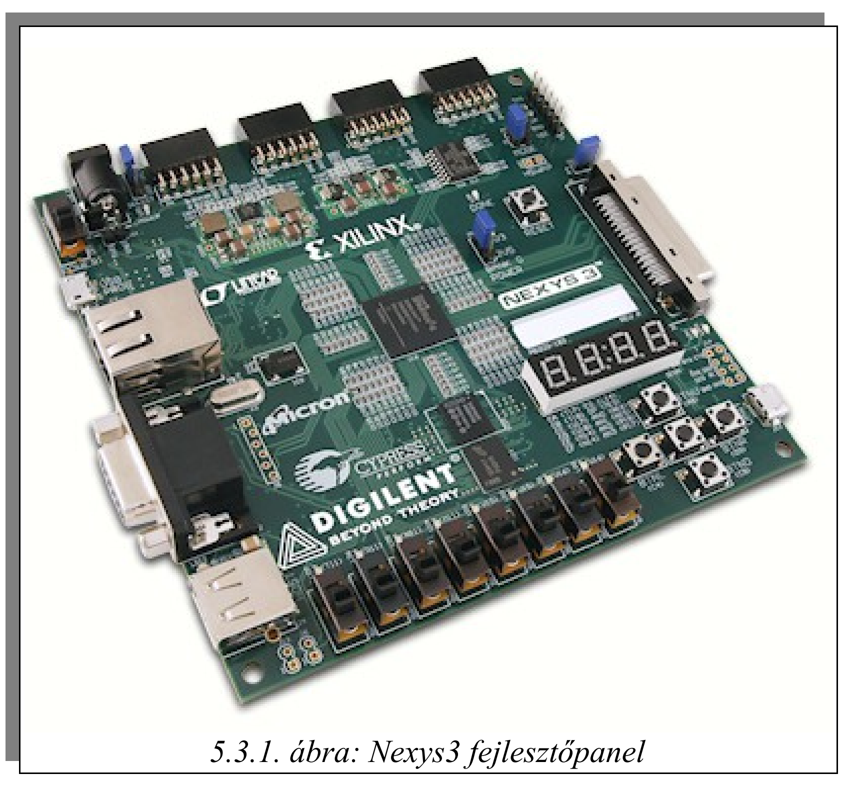
A nyákon található néhány egyszerűbb periféria is (Kapcsolók, nyomógombok, LED-ek, hétszegmenses kijelzők)



5.2.2. ábra: Perifériák a Nexys2 panelen

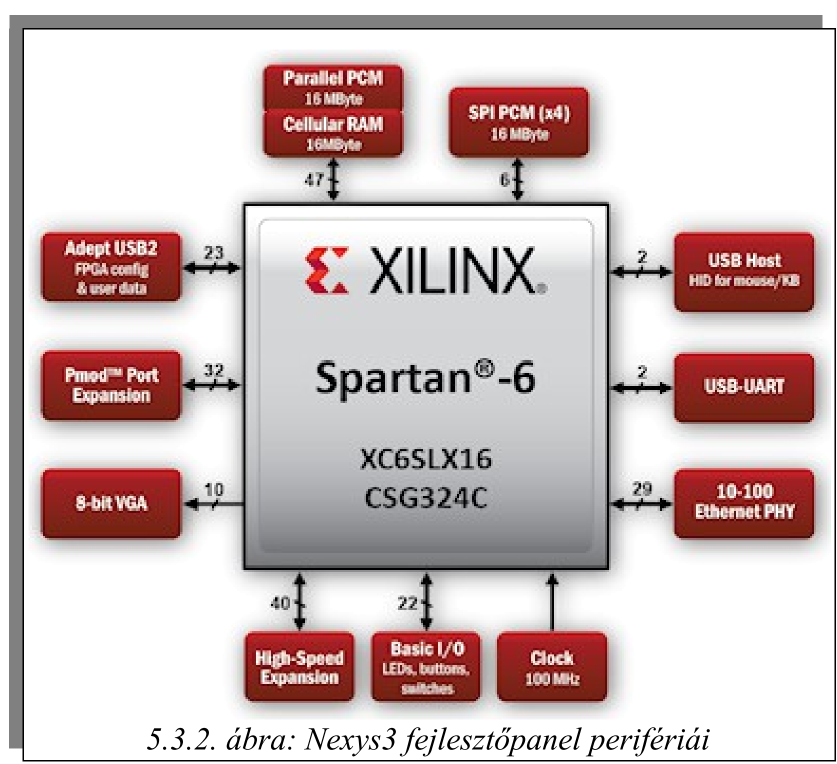
A panelen helyet kapott egy 50MHz-es oszcillátor, másodlagos forrásként egy foglalaton keresztül lehet órajelet csatlakoztatni. Egy 16MB flash memória mellett egy 16MB SDRAM segíti a képfeldolgozási és egyéb feladatok ellátását. 75 I/O láb lett kivezetve, ebből 43 nagy sebességű kis parazitakapacitású Hirose csatlakozón keresztül. A 8 bites VGA port mellett 2 db PS/2 csatlakozó teszi lehetővé számunkra a komplex feladatok tervezését, szimulálását és végrehajtását.

5.3. Nexys™3 Spartan 3E FPGA fejlesztőpanel



5.3.1. ábra: Nexys3 fejlesztőpanel

A 5.3.1. ábrán látható Nexys3 típusú fejlesztőpanel már profi felhasználásra készült. A Sparta3-at felváltotta a Spartan6 típusú FPGA. A már a Nexys2-nél is megismert csatlakozók, ill. perifériák kiegészítésre kerültek egy 10/100Mbites ethernetes csatlakozóval. A panelen összesen 48MB memória kapott helyet (párhuzamos, PSRAM, PCM). Az FPGA órajele 100MHz. Az I/O lábak a szabványos és könnyen kezelhető VHDC csatlakozón lettek kivezetve.



5.3.2. ábra: Nexys3 fejlesztőpanel perifériái

5.4. Saját tervezésű fejlesztőpanelek

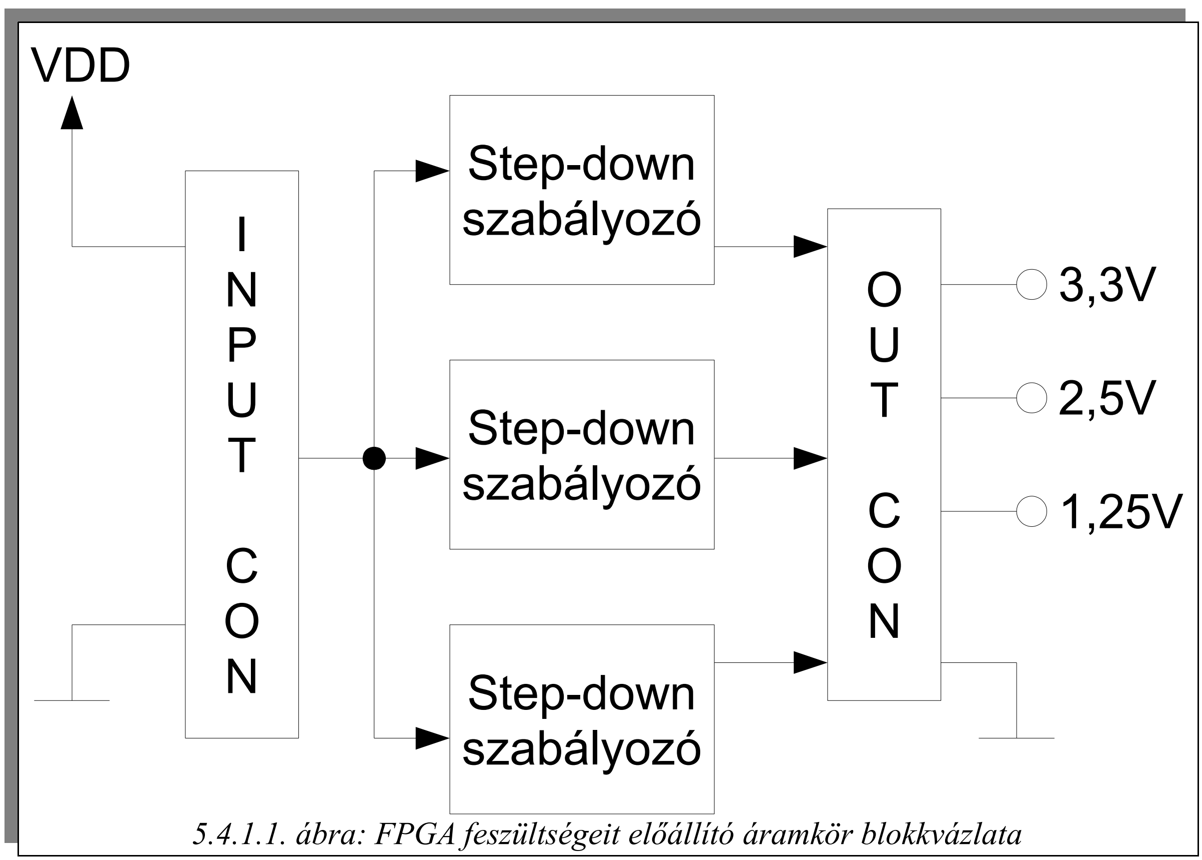
Minden fejlesztésünkhöz saját paneleket szoktunk készíteni. Ennek több oka is van. Egyrészt anyagi lehetőségeink korlátozottak, másrészt így tudjuk leginkább kielégíteni az egyes panelekkel szemben fejlesztéseink során felmerülő igényeinket. Kisebb módosítások ilyen esetben nem okozhatnak gondot, valamint meghibásodás esetén is egyszerű a javítás. Mindemellett nem szabad elmennünk azon tény mellet sem, hogy a beágyazott rendszerek fejlesztése nem csak programozási, hanem nagymértékben hardveres, tervezési feladat is. Amennyiben a későbbiekben szeretnénk egy konkrét feladatot megvalósítani akkor a saját fejlesztésű panelek megépítése, beüzemelése során tapasztaltak jelentősen lerövidítik a tervezési/gyártási időt.

A fejlesztőpaneljeink kapcsolási rajza és nyákterve, valamint az utánépítéshez szükséges alkatrészjegyzék és beültetési rajz elérthető a Moodle rendszerben. A legyártást azonban csak olyanoknak javasoljuk akik komoly tapasztalatokkal rendelkeznek az elektronikus eszközök gyártása és élesztése terén.

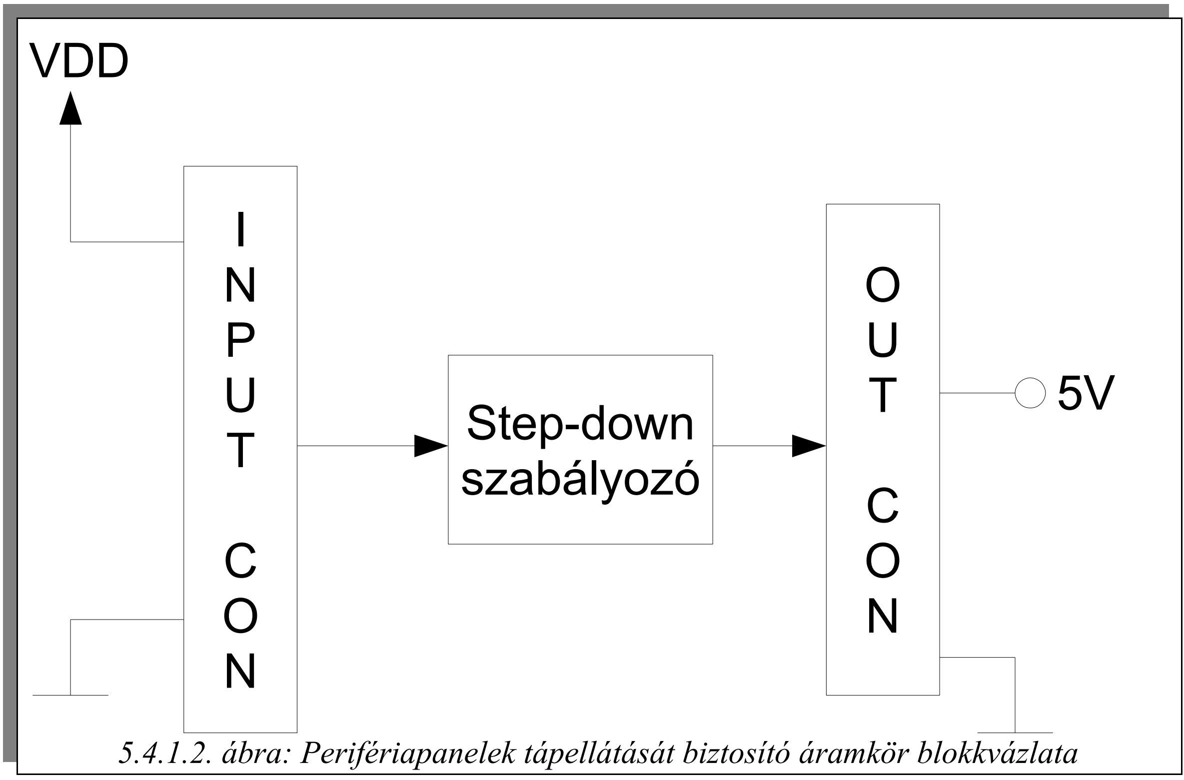
5.4.1. Tápegységpanel felépítése

Az FPGA működtetéséhez többféle tápfeszültség szükséges. A belső mag meghajtásához és az IO szabványok teljesítéséhez (a kimeneti driverek meghajtásához) szükségünk van 1,2V; 2,5V; és 3,3V feszültségszintekre. Mindemellett célszerű egy 5V-os tápfeszültséget is létrehozni a perifériáknak.

Az FPGA tápellátását 3 darab MC34063 kapcsolóüzemű IC biztosítja, a perifériákét pedig egy LM2576. Az alapvető működésük megegyezik, azonban a 2576 a TO263 tokozás miatt nagyobb áramerősséget képes biztosítani. Az FPGA tápellátását a 5.4.1.1. ábrán, míg a perifériákét a 5.4.1.2. ábrán láthatjuk.



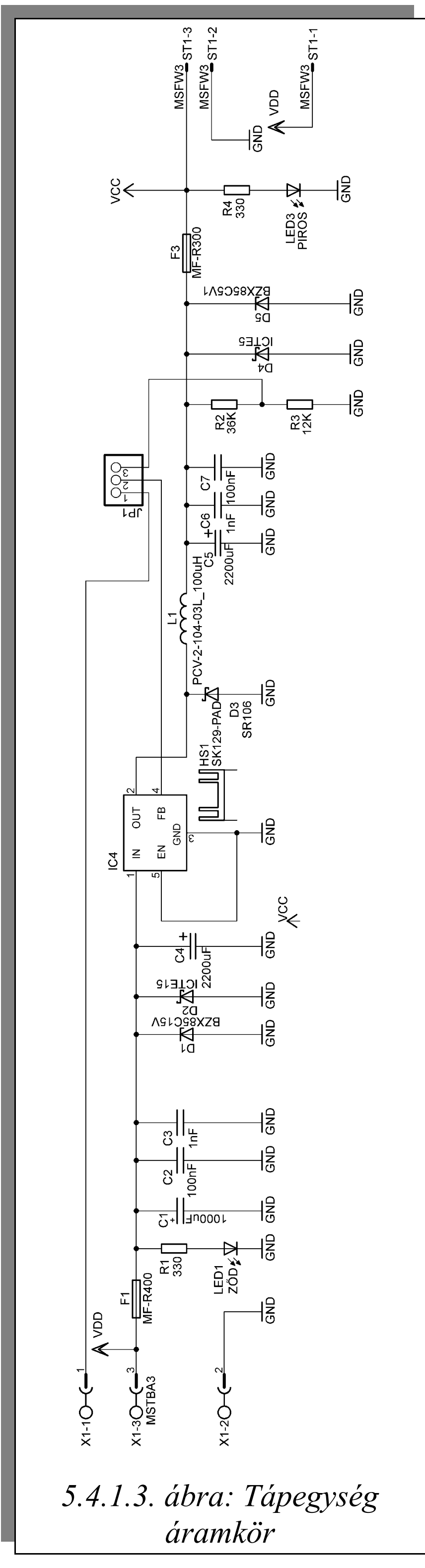
5.4.1.1. ábra: FPGA feszültségeit előállító áramkör blokkvázlata



5.4.1.2. ábra: Perifériapanelek tápellátását biztosító áramkör blokkvázlata

A VDD tápfeszültségnek miután step-down regulátorról van szó nagyobbnak kell lennie, mint a kimeneten előállított szabályozott feszültségnek[28]. Ez az FPGA tápellátása esetében minimum 5V, míg a perifériáknál minimum 7,5V. A maximális érték 12V.

28 A kimeneti feszültségértékhez hozzájön még az ún. drop feszültség is, ami esetünkben kb. 1,5V.



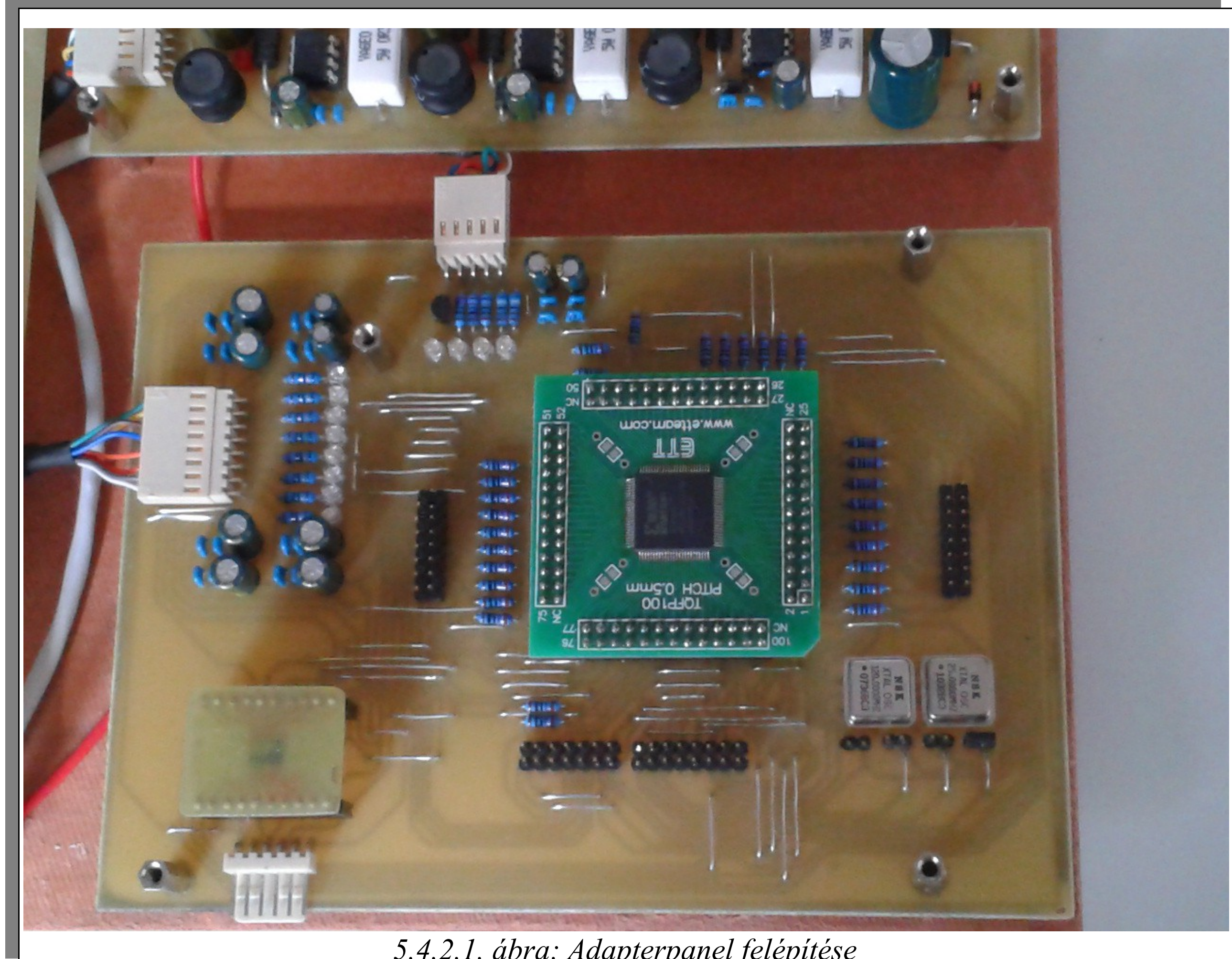
5.4.1.3. ábra: Tápegység áramkör

Az 5V-os Step-down üzemű kapcsolóüzemű feszültségstabilizátor kapcsolási rajzát a 5.4.1.3. ábrán láthatjuk[29]. Az áramkör nagyfrekvenciás szűréssel, túlfeszültség és túláramvédelemmel van ellátva. A bemeneten lévő szupresszor és Zener diódák biztosítják a túlfeszültség, míg a multifuse típusú regenerálódó biztosíték a túláramvédelmet. A kimeneten szintén hasonló védelem segíti a hibamentes működést. A kondenzátorok szűrik a kapcsolási frekvenciából (50kHz) eredő és a nagyfrekvenciás (MHz) zajokat egyaránt. Zöld LED jelzi a bemeneti csatlakozón lévő feszültség, míg piros a kimeneten előállított feszültség meglétét. Egy jumper segítségével lehetőségünk van a feedback (visszacsatolási) pont kívülről történő megadására is (ez nagy áramfelvételnél hasznos lehet). Az általunk használt konstrukcióban 12V-nál nagyobb feszültség elvileg nem kerülhetett az áramkör bemenetére, ezért 15V-nál határoltunk a bemeneten, a diódák kicserélésével ez akár 40V-ig, az IC határértékéig növelhető szükség esetén.

29 A többi feszültség előállítása is hasonló módon történik.

5.4.2. Adapterpanel felépítése

Az adapterpanel feladata, hogy az SMD tokozású 100 lábú FPGA kivezetéseit szabványos csatlakozókon biztosítsa. Az adapterpanel tartalmaz a későbbi használatra Platform Flash memóriát. Az eredeti fejlesztésen az M0; M1; M2 konfigurációs lábakat nem vezettük ki. Az oktatócsomag még ezt a verziót tartalmazza. Ez a későbbiekben módosításra került. A Moodle rendszerben amint lehet szerepeltetni fogjuk az új verziót[30]. Az adapterpanelen kapott helyet 4 órajelforrás is, ami a GCLK0÷3 lábakra van csatlakoztatva (32,768kHz; 25MHz; 100MHz; 120MHz).



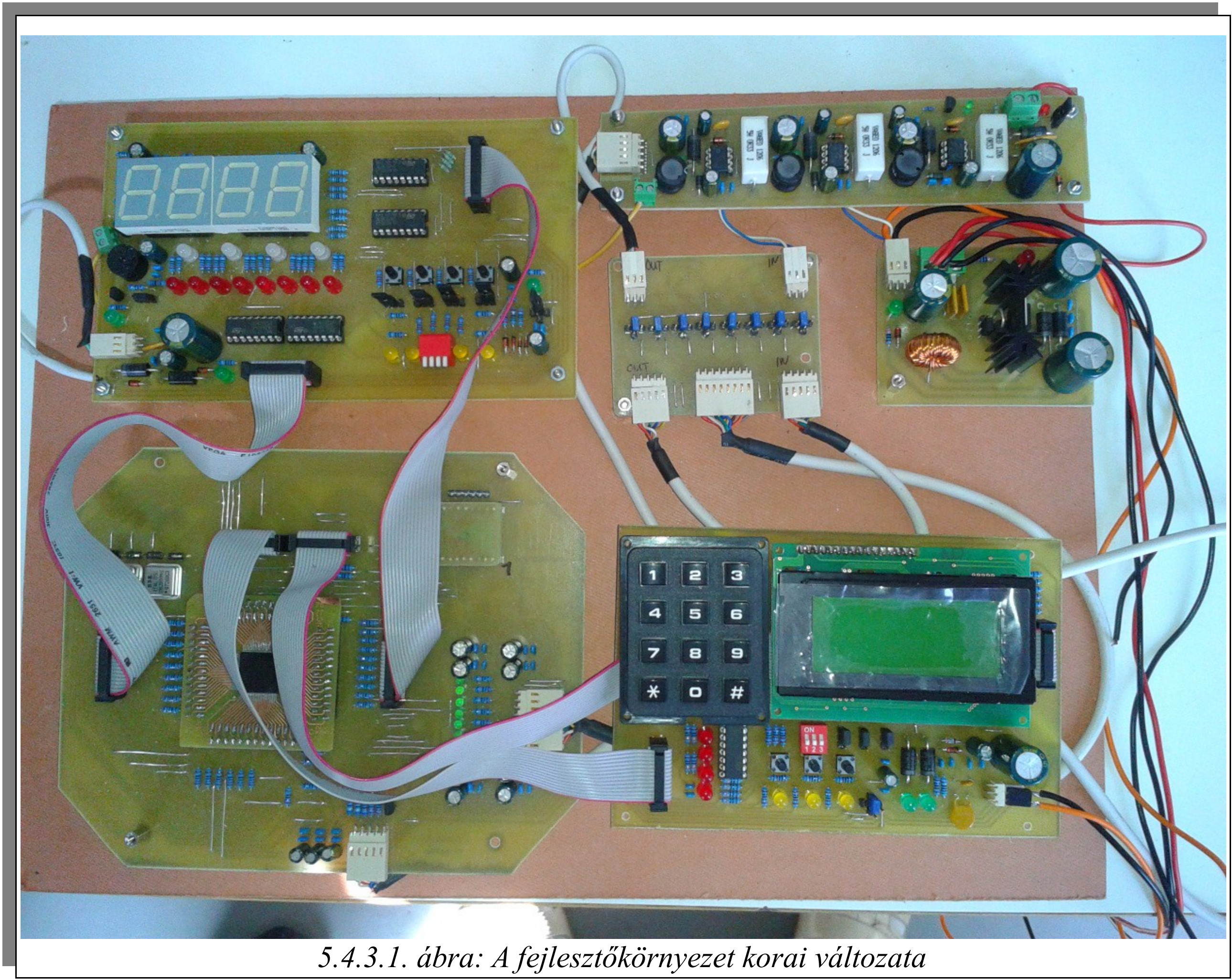
5.4.2.1. ábra: Adapterpanel felépítése

A 5.4.2.1. ábrán láthatjuk az adapterpanelt. Középen egy öcsi panelen helyezkedik el a 100 lábú VQFP tokozású Spartan3 FPGA, amelynek I/O lábait szabványos szalagkábel csatlakozókra vezettük ki. A jobb alsó sarokban található 2 THT tokozású oszcillátor, a másik két órajelforrás SMD kialakításban a panel alján kapott helyet (4 jumper segíti a megfelelő órajel kiválasztását). A Platform Flash a képen a bal alsó sarokban látható, miután csak felületszerelt kivitelben kapható, ezért ezt is egy öcsi panelen helyeztük el – a mellette lévő 6-os csatlakozó a JTAG interfész. A bal oldalon a V_{CCO} lábakhoz tartozó tápcsatlakozó található (minden bankhoz külön feszültségértéket lehet megadni). A fent lévő csatlakozón az FPGA egyéb feszültségeit adhatjuk meg (1,2V; 2,5V; 3,3V).

30 Jelen dokumentum írásakor az áttervezés a nyákterven már megtörtént. Az eredeti panelen átforrasztva ki is próbáltuk a módosítást, azonban új nyák még nem készült így pedig felelősséggel nem szerepeltethetjük a módosítást.

5.4.3. Perifériapanelek

A 5.4.3.1. ábrán láthatjuk az általunk használt fejlesztőkörnyezet korai változatát. Azóta több változtatás és egyszerűsítés történt.



5.4.3.1. ábra: A fejlesztőkörnyezet korai változata

A bal alsó sarokban található az adapterpanel. Felette az 1. generációs próbapanel. A jobb felső sarokban láthatjuk a tápegységpaneleket, míg alatta a 2. generációs próbapanelt. Középen egy csatlakozópanelt láthatunk, amin ki tudjuk választani, hogy az egyes bankok milyen feszültséget kapjanak (1,25V; 2,5V; 3,3V; vagy egy potenciométerrel beállított érték). A tápegységpanelhez sorkapcsokon keresztül tudunk csatlakozni. Minden más panelt szalagkábellel, ill. tápcsatlakozóval tudunk összekötni – a helytelen összekötés elkerülése végett.

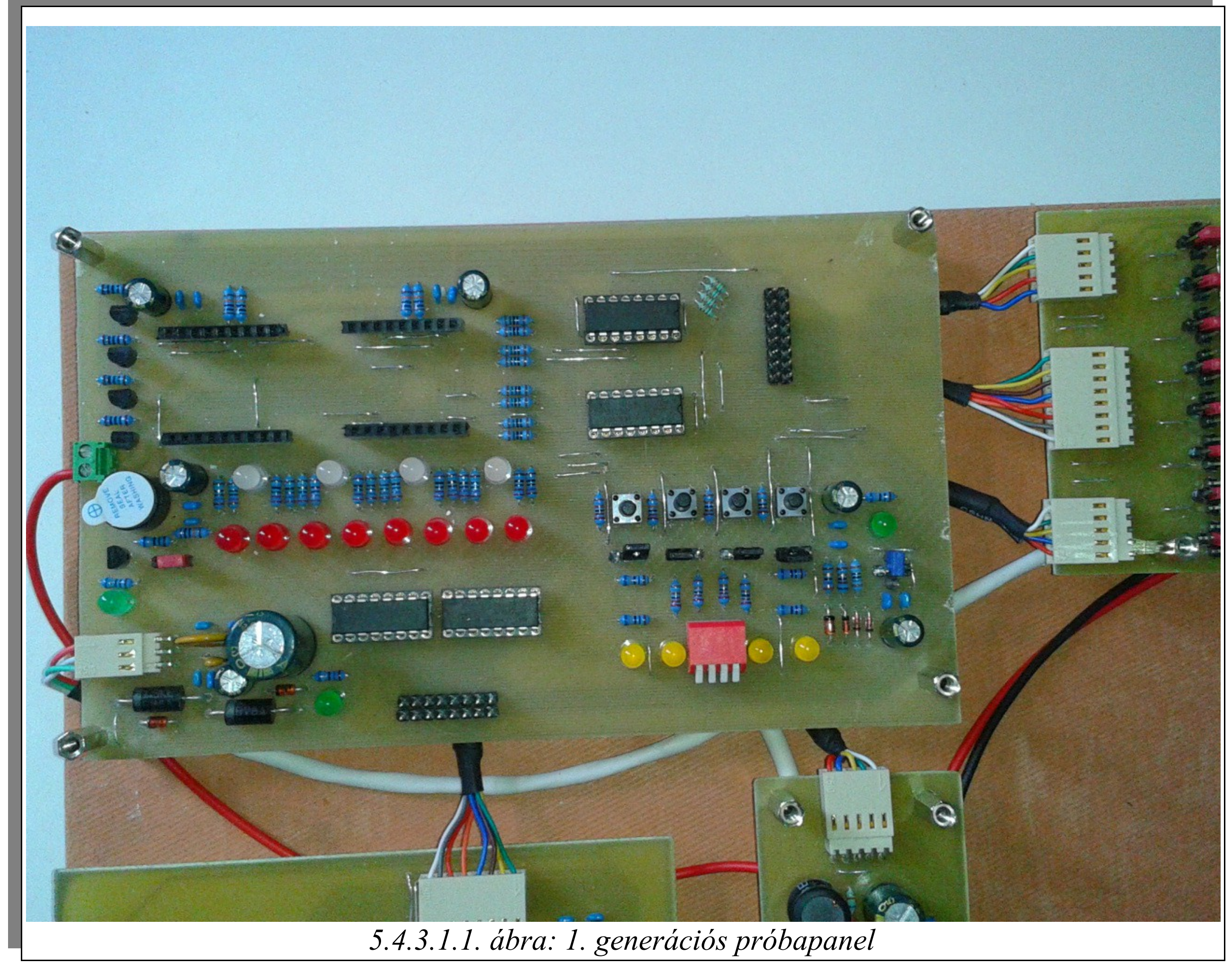
5.4.3.1. 1. generációs próbapanel

Az 1. generációs panel a legkönnyebben lekezelhető perifériákat tartalmazza, amik megkönnyítik a fejlesztés kezdeti lépéseinek elsajátítását. Panelünkön a következő elemek találhatóak meg:

- Nyomógomb
- DIP kapcsoló
- LED
- Kétszínű LED
- Zümmer
- 7-szegmenses kijelző

A kapcsolók/nyomógombok multiplexelve kerültek alkalmazásra – hogy a felhasználónak melyikre van szüksége, azt jumperek segítségével tudja kiválasztani. A zümmer egy LED-del van multiplexelve.

Az áramkör képét a 5.4.3.1.1. ábrán láthatjuk.



5.4.3.1.1. ábra: 1. generációs próbapanel

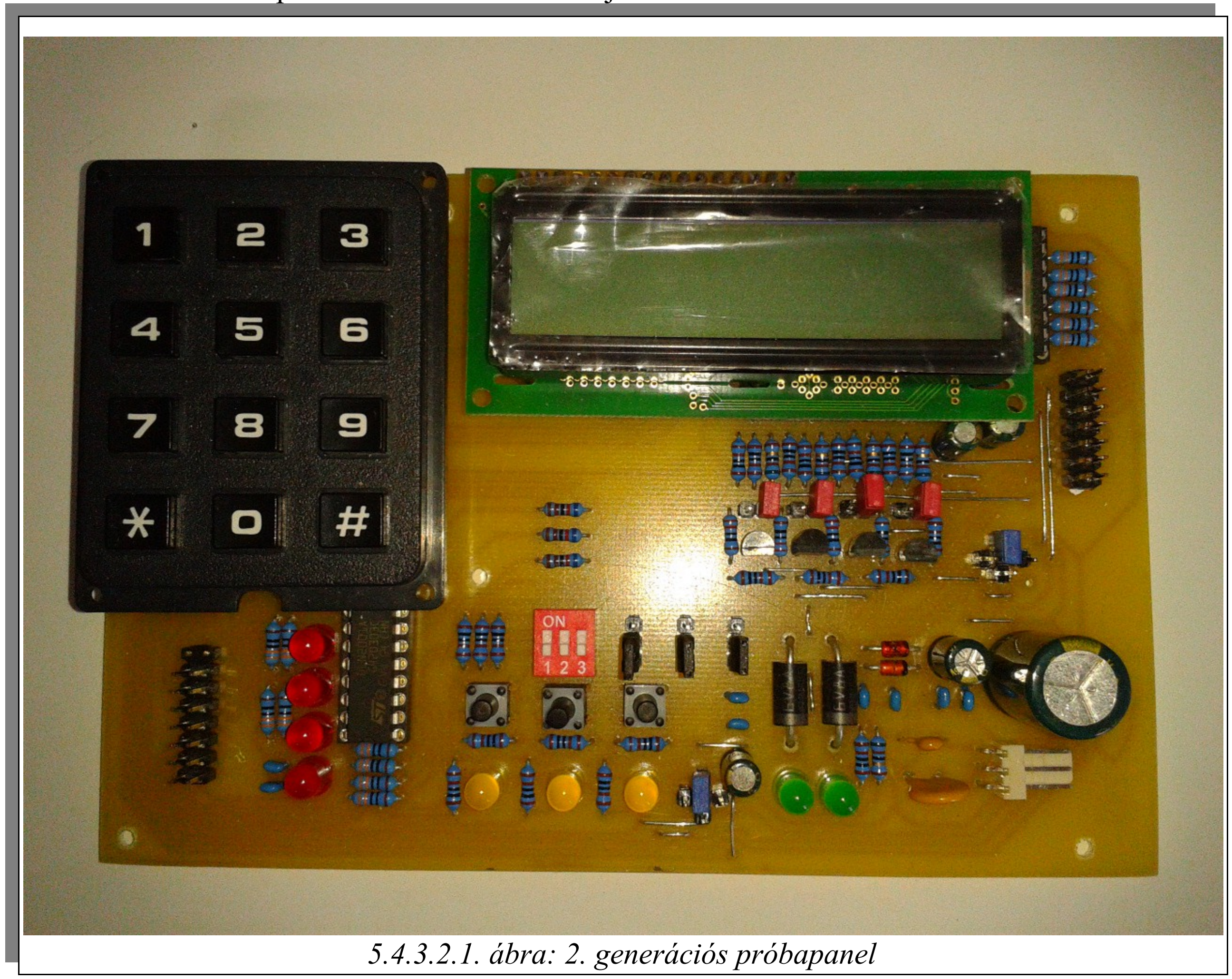
Az első generációs próbapanelen 2 db két hétszegmenses kijelzőt tartalmazó modul található a bal felső sarokban. Alatta helyezkednek el a kétszínű LED-ek, illetve a piros LED-ek. A LED-ek mellett bal oldalon láthatjuk a zümmert – a bal oldali LED és a zümmer között található a kiválasztó jumper. A panel bal alsó sarkában lévő tápcsatlakozón tudjuk megadni a működéshez szükséges feszültséget (5V). A jobb oldali kvadránsban helyezkednek el a nyomógombok és a DIP kapcsoló (4db) – kiválasztásuk a köztük lévő jumperek segítségével történhet. A beviteli periféria logikai értékét 4 db sárga LED jelzi. Az adapterpanelhez két szalagkábellel tudunk csatlakozni.

5.4.3.2. 2. generációs próbapanel

A 2. generációs panel a legegyszerűbb be- és kiviteli perifériákkal lett ellátva. A tervezésnél figyeltünk rá, hogy az 1. generációs panellel egyszerre lehessen alkalmazni. A használható perifériák:

- LED
- Nyomógomb
- DIP kapcsoló
- Mátrix tasztatúra
- Karakteres LCD kijelző

Az áramkör képét a 5.4.3.2.1. ábrán láthatjuk.



5.4.3.2.1. ábra: 2. generációs próbapanel

A 2. generációs próbapanelen helyet kapott egy mátrix tasztatúra és egy karakteres LCD kijelző – a szabványos csatlakozón 1, 2, ill. 4 soros kijelző is elhelyezhető. A panel beüzemelését 4 db LED, ill. 3 db nyomógomb/kapcsoló segíti. Az áramkör 5V-ról tápcsatlakozón keresztül üzemel, az adapterpanellel való összeköttetést szalagkábellel tudjuk biztosítani.

5.4.3.3. 3. generációs próbapanel

A 3. generációs panel felhasználja az FPGA összes I/O lábát a működéshez (nem számítva a 4 órajelbemenetet). A következő perifériák állnak rendelkezésünkre:

- LED
- Nyomógomb
- DIP kapcsoló
- Mátrix tasztatúra
- Karakteres LCD kijelző
- RGB LED
- A/D átalakító
- Potenciométer
- VGA csatlakozó
- RS-232
- 2db PS/2

6. Szoftveres fejlesztőkörnyezet

Amikor a szoftveres fejlesztőkörnyezet kiválasztása volt a feladat a fejlesztőcsapat a Xilinx saját fejlesztőkörnyezete, a Xilinx ISE mellett döntött, azonban emellett számos más programcsomag áll rendelkezésünkre, amennyiben az FPGA fejlesztés mellett döntünk (pl. Altium Designer).

6.1. Xilinx ISE Design Suite[31]

A Xilinx, hogy megkönnyítse az FPGA-ira történő fejlesztést biztosít egy ingyenes fejlesztőkörnyezetet a Xilinx ISE Webpack formájában, valamint a Xilinx ISE Design Suite bármely verziójához igényelhető regisztráció után egy 30 napos ingyenes licenc. Mi a System Edition-t és a Webpacket egyaránt kipróbáltuk. Leírásunk az SE verzió alapján készült el.

A Xilinx Design Suite programcsomag változatainak összehasonlítása[2]					
Szoftvercsomag / Programrészek	ISE WebPACK	Logic Edition	Embedded Edition	DSP Edition	System Edition
ISE Simulator (ISim)	☺	☺	☺	☺	☺
PlanAhead Design and Analysis Tool	☺	☺	☺	☺	☺
ChipScope Pro		☺	☺	☺	☺
ChipScope Pro Serial I/O Toolkit		☺	☺	☺	☺
Embedded Development Kit			☺		☺
Software Development Kit			☺		☺
System Generator for DSP				☺	☺

ISE WebPACK: Az ISE WebPACK az iparág egyetlen ingyenes, teljesértékű FPGA és CPLD fejlesztőkörnyezete. A fenti táblázatból is látható, hogy tartalmazza a HDL[32] tervező és analizáló, valamint szimulációs részt. Támogatja a JTAG programozást is.

Logic Edition: A Logic Edition főleg optimalizálásban és időzítési megoldásokban nyújt többet,mint a WebPACK, tartalmazza a ChipScope™[33] analizáló eszközt.

Embedded Edition: Tartalmaz mindent, amit a Logic Edition, valamint támogatja a PLB és AXI alapú beágyazott összetevőket (Xilinx Platform Studio (XPS), Software Development Kit (SDK), MicroBlaze Soft Processor, Embedded IP peripherals).

31 A Moodle rendszerben található egy a Xilinx cég által kiadott pdf, amelyben szoftvercsomagjainak összehasonlítása látható.
32 Hardware Description Language: hardverleíró nyelv
33 Részletesen l. http://www.xilinx.com/tools/cspro.htm

DSP Edition: Tartalmaz mindent, amit a Logic Edition, kiegészítve DSP tervezési megoldásokkal.

A Xilinx Design Suite programcsomag változatainak részletes összehasonlítása[3]					
Szoftvercsomag / Programrészek	ISE WebPACK	Logic Edition	Embedded Edition	DSP Edition	System Edition
System Generator for DSP				☺	☺
Platform Studio	Az eszköz zárolva a 3 legkisebb Zynq-hez	Az eszköz zárolva a 3 legkisebb Zynq-hez	☺	Az eszköz zárolva a 3 legkisebb Zynq-hez	☺
Software Development Kit		☺	☺	☺	☺
MicroBlaze Soft Processor			☺		☺
MicroBlaze Microcontroller System	☺				☺
Design Preservation	☺	☺	☺	☺	☺
Project Navigator	☺	☺	☺	☺	☺
CORE Generator	☺	☺	☺	☺	☺
PlanAhead	☺	☺	☺	☺	☺
ChipScope Pro and the ChipScope Pro Serial I/O Toolkit		☺	☺	☺	☺
Partial Reconfiguration[34]	☺	☺	☺	☺	☺
Power Optimization	☺	☺	☺	☺	☺
ISE Simulator (ISim)	Korlátozott	☺	☺	☺	☺
XST Synthesis	☺	☺	☺	☺	☺
Timing Driven Place & Route, SmartGuide, and SmartXplorer	☺	☺	☺	☺	☺

34 Megvásárolható lehetőség.

6.2. Xilinx Design Suite System Edition[35]

A keretprogram letölthető a Xilinx honlapjáról:

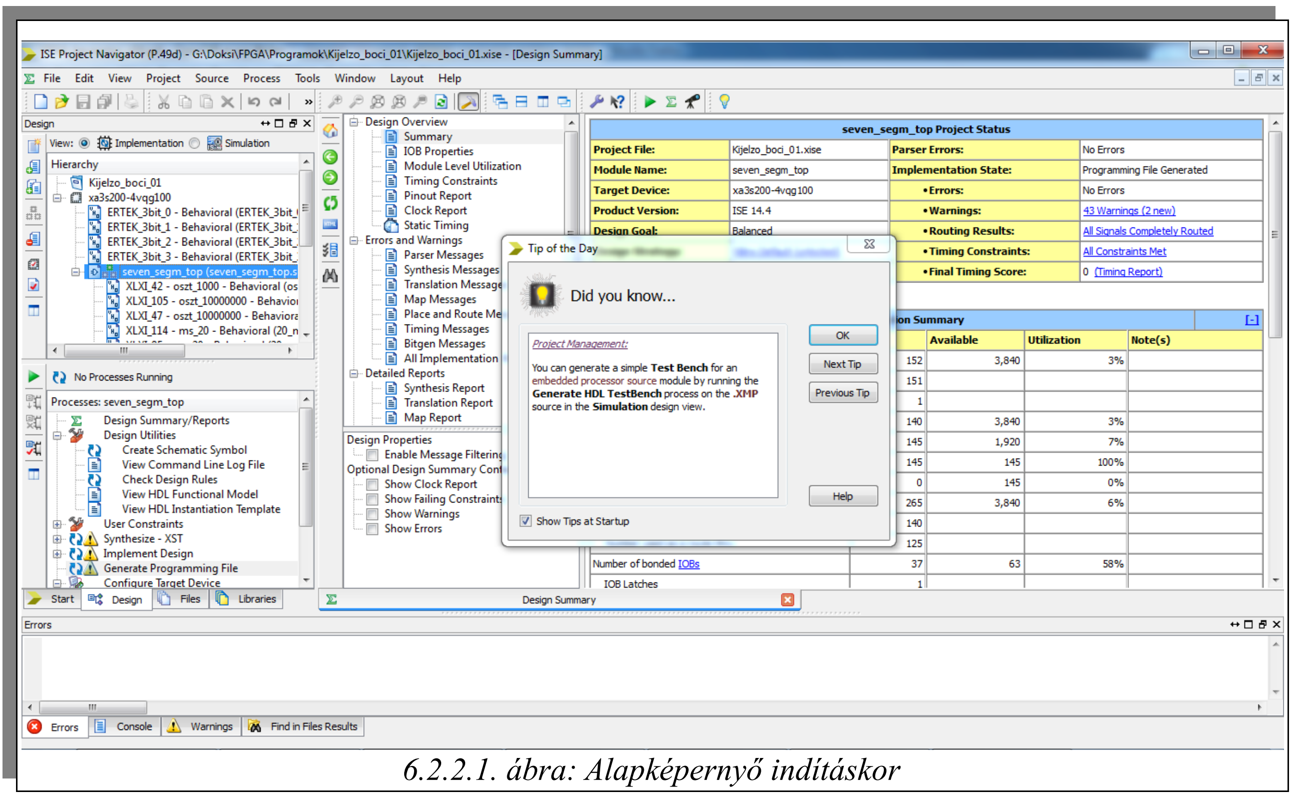
http://www.xilinx.com/products/design-tools/ise-design-suite/index.htm

6.2.1. Telepítési eljárás

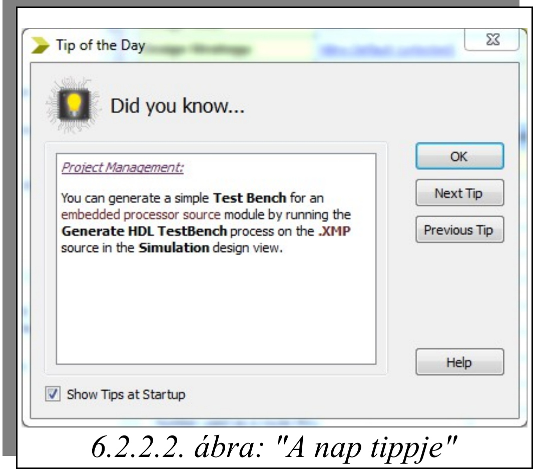
35 A leírásban a 14.4 verzió kerül ismertetésre.

6.2.2. A program felépítése és menürendszere

Indítás után a 6.2.2.1. ábrán látható képernyővel találkozunk. Alapértelmezetten a legutolsó projekt kerül megnyitásra.



6.2.2.1. ábra: Alapképernyő indításkor



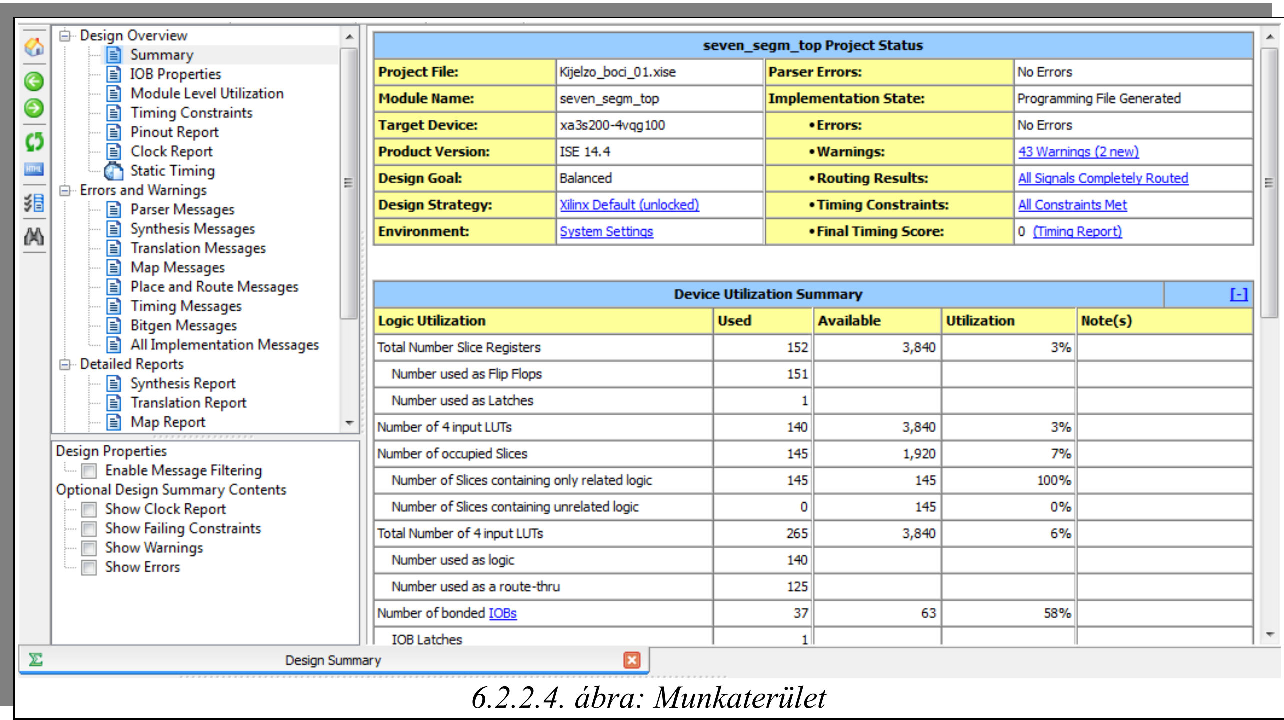
6.2.2.2. ábra: "A nap tippje"

A 6.2.2.2. ábrán láthatjuk a mai modern programok népszerű szolgáltatásaként jelentkező „Tudtad-e” ablakot. A *Show Tips at Startup* pontból kiszedve a pipát el tudjuk kerülni, hogy minden indításnál szembesüljünk az aznapi tippel. A *Next Tip* és a *Previous Tip* gombokkal tudunk lépkedni a különböző tanácsok között. A *Help* gomb megnyomására megnyílik a súgó ide tartozó része, ahol a fent leírtakat olvashatjuk angol nyelven. Az *OK* gombot megnyomva az ablak eltűnik, és kezdhetjük az érdemi munkánkat.

A 6.2.2.3. ábrán láthatjuk a képernyő felső kvadránsát elfoglaló szokásos menürendszert és eszköztárat.

6.2.2.3. ábra: Menürendszer és eszköztárak

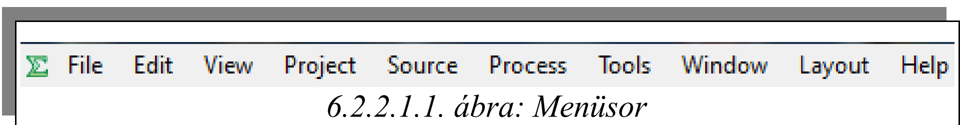
A 6.2.2.4. ábrán látható munkaterületen az éppen szerkesztés alatt lévő fájlt láthatjuk. Alapértelmezésben ez a tervezést összegző és a dokumentálást megkönnyítő ún. *Design Summary*.



6.2.2.4. ábra: Munkaterület

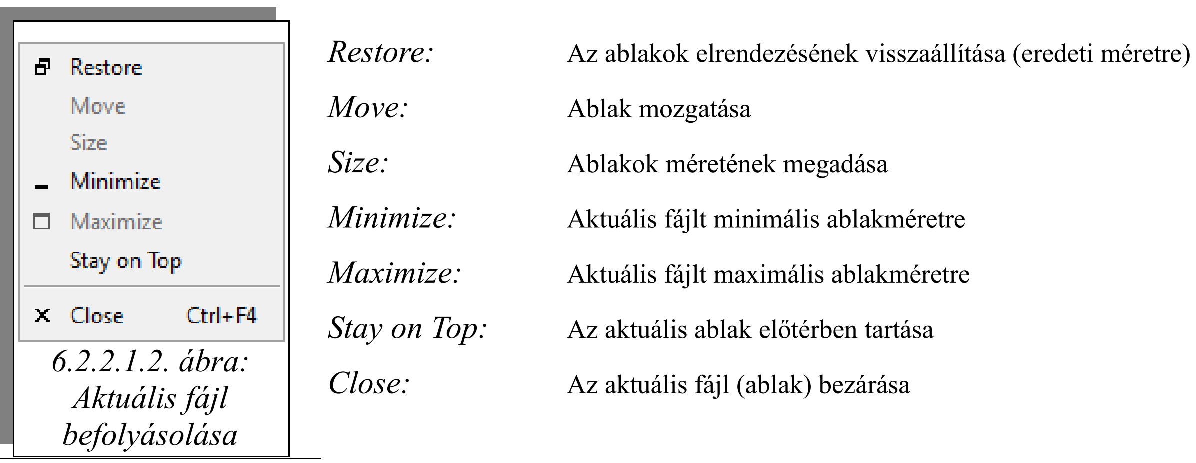
A képernyő többi részét a *View* menüpontban bekapcsolt panelek foglalják el, amikre majd a menü tárgyalásakor térünk ki részletesen (l. 6.2.2.1.3. fejezet).

6.2.2.1. Menürendszer



6.2.2.1.1. ábra: Menüsor

A 6.2.2.1.1. ábrán látható a *Xilinx ISE SE* menüsora. A [36] jel a munkaterületen lévő éppen aktív fájlal kapcsolatos 6.2.2.1.2. ábrán lévő manipulációkat tartalmazza.

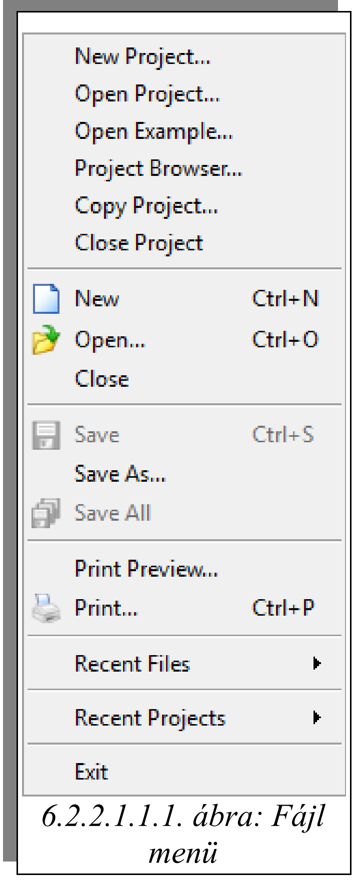


6.2.2.1.2. ábra: Aktuális fájl befolyásolása

Restore:	Az ablakok elrendezésének visszaállítása (eredeti méretre)
Move:	Ablak mozgatása
Size:	Ablakok méretének megadása
Minimize:	Aktuális fájlt minimális ablakméretre
Maximize:	Aktuális fájlt maximális ablakméretre
Stay on Top:	Az aktuális ablak előtérben tartása
Close:	Az aktuális fájl (ablak) bezárása

36 A jel utal a fájl típusára: míg a jelet a *design summary* fájl esetén láthatjuk, addig pl. egy *schematic* típusú fájl esetén a jelenik meg.

6.2.2.1.1. File menü



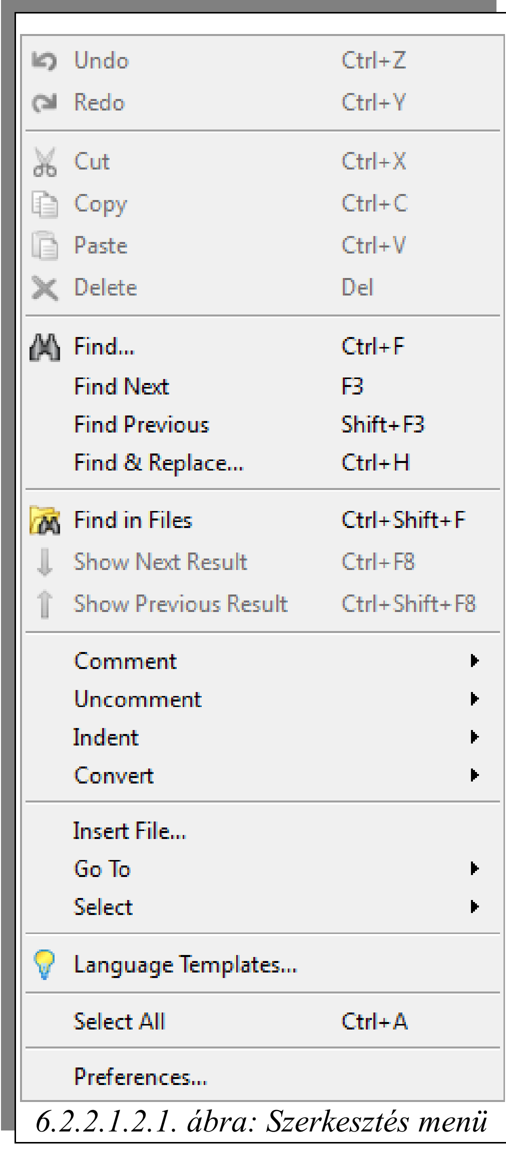
6.2.2.1.1.1. ábra: Fájl menü

New Project...:	Új projekt létrehozása
Open Project...:	Projekt megnyitása
Open Example...:	Gyári példa projektek megnyitása
Project Browser...:	Projekt intéző
Copy Project...:	Aktuális projekt másolása
Close Project:	Aktuális projekt bezárása
New:	Új fájl létrehozása
Open...:	Fájl megnyitása
Close:	Fájl bezárása
Save:	Fájl mentése
Save As...:	Fájl mentése más néven
Save All:	Az aktuális projekt összes fájljának mentése
Print Preview...:	Aktuális fájl nyomtatási képe
Print...:	Aktuális fájl nyomtatása
Recent Files:	Legutóbbi fájlok
Recent Projects:	Legutóbbi projektek
Exit:	Kilépés a programból

A fájl menüben a szokásos menüpontokat láthatjuk. Új projekt, fájl, létrehozása, létező megnyitása a 6.2.3. fejezetben kerül részletes ismertetésre.

Itt szeretnénk megemlíteni, hogy a Xilinx jól használható súgóval rendelkezik, ami bár angol nyelven érhető el, mégis nagyon informatív.

6.2.2.1.2. Edit menü[37]



6.2.2.1.2.1. ábra: Szerkesztés menü

Undo: Visszavonás

Redo: Ismétlés

Cut: Kivágás

Copy: Másolás

Paste: Beillesztés

Delete: Törlés

Find...: Keresés

Find Next: Következő keresése

Find Previous: Előző keresése

Find & Replace...: Keresés és csere

Find in Files: Keresés fájlban

Show Next Result: Mutasd a következő eredményt

Show Previous Result: Mutasd az előző eredményt

Comment: Megjegyzéssé alakítás[38]

Uncomment: Fordítandó kóddá alakítás[39]

Indent: Tabulálás[40]

Convert: Konvertálás

Insert File...: Fájl beillesztése

Go To: Ugorj

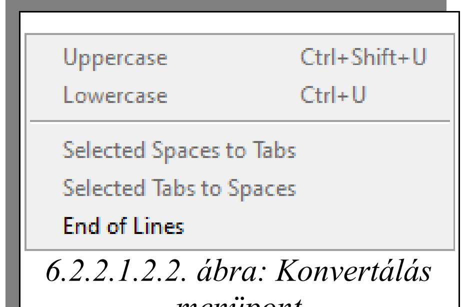
Select: Kijelölés

Language Templates...: Nyelvi sablonok

Select All: Minden kijelölése az aktuális fájlban

Preferences...: Tulajdonságok

37 Szöveges fájlok (pl. egy VHDL forrás) esetén.
38 Sor(ok), vagy kijelölés.
39 Sor(ok), vagy kijelölés.
40 Egy tabulátorral beljebb, vagy kijjebb.



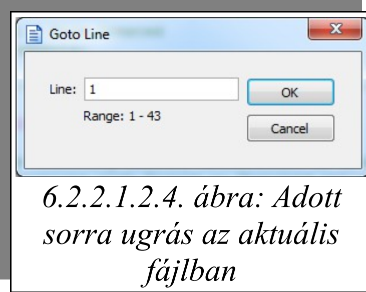
6.2.2.1.2.2. ábra: Konvertálás menüpont

Uppercase: Kijelölt karakterek nagybetűvé alakítása
Lowercase: Kijelölt karakterek nagybetűvé alakítása
Selected spaces to Tabs: Kijelölt szóközök tabulátorra alakítása
Selected Tabs to Spaces: Kijelölt tabulátorok szóközzé alakítása
End of Lines: Sorvége karakter (CR, CR+LF, EOL)

Next Word	Ctrl+Right
Previous Word	Ctrl+Left
Beginning of File	Ctrl+Home
End of File	Ctrl+End
Line...	Ctrl+G

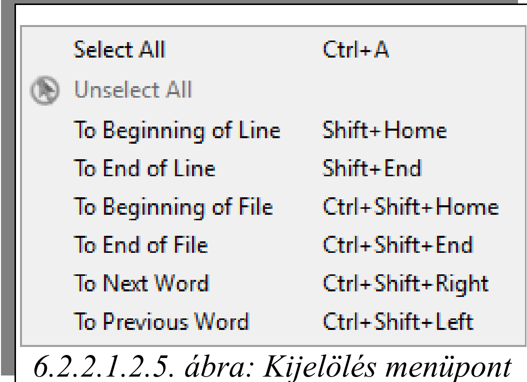
6.2.2.1.2.3. ábra: Ugrás menüpont

Next Word:	Ugrás a következő szóra
Previous Word:	Ugrás az előző szóra
Beginning of File:	Ugrás a fájl kezdetére
End of File:	Ugrás a fájl végére
Line...:	Adott sorra ugrás (l. 6.2.2.1.2.4. ábra)



6.2.2.1.2.4. ábra: Adott sorra ugrás az aktuális fájlban

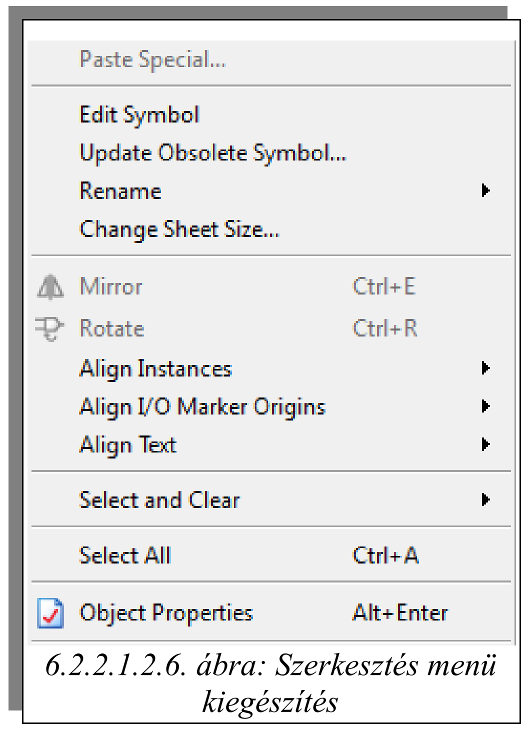
A *Line* mezőben adhatjuk meg, hogy az aktuális fájl melyik sorára szeretnénk ugrani. A *Range* mezőben láthatjuk a fájl terjedelmét sorokban megadva. Az *OK* gomb megnyomására megtörténik az ugrás, míg a *Cancel* gombbal visszaléphetünk.



6.2.2.1.2.5. ábra: Kijelölés menüpont

Select All: Minden kijelölése
Unselect All: Minden kijelölés megszüntetése
To Beginning of Line: Kijelölés a sor kezdetéig
To End of Line: Kijelölés a sor végéig
To Beginning of File: Kijelölés a fájl kezdetéig
To End of File: Kijelölés a fájl végéig
To Next Word: Kijelölés a következő szóig
To Previous Word: Kijelölés az előző szóig

Amennyiben nem szövegfájl az aktuális fájl, akkor a szerkesztésmenü pontjai is megváltoznak. Egy *schematic* típusú fájlban a karaktermanipulációs műveletek nem értelmezhetőek, ezzel szemben új pontokkal is találkozhatunk (l. 6.2.2.1.2.6. ábra), amik megkönnyítik a fejlesztést.



6.2.2.1.2.6. ábra: Szerkesztés menü kiegészítés

Paste Special: Speciális beillesztés (l. 6.2.2.1.2.7. ábra)

Edit Symbol: Kijelölt szimbólum szerkesztése

Update Obsolete Symbol...: Elavult szimbólumok frissítése (l. 6.2.2.1.2.8. ábra)

Rename: Átnevezés[41]

Change Sheet Size...: Lapformátumok (l. 6.2.2.1.2.10. ábra)

Mirror: Tükrözés

Rotate: Forgatás

Align Instances: Szimbólumok elhelyezése[42]

Align I/O Marker Origins: l. eggyel fentebb

Align Text: l. kettővel fentebb

Select and Clear: Objektumok kijelölés[43]

Select All: Minden kijelölése

Object Properties: Tulajdonságok (l. 6.2.2.1.2.11. ábra)

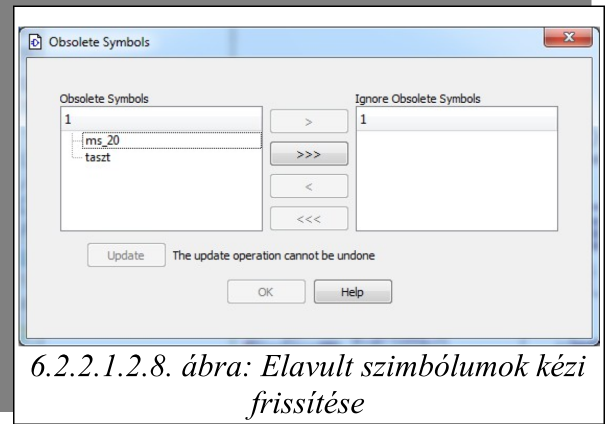
6.2.2.1.2.7. ábra: Speciális beillesztés

Schematic típusú fájlok esetén lehetőségünk van a 6.2.2.1.2.7. ábrán látható speciális beillesztésre. Lehetőségek:

- Minden net megőrzi az eredeti nevét
- Minden net új nevet kap
- A bemenetek megőrzik a nevüket
- A felhasználó által megadott nevek nem változnak

Azok a netek, amelyek ugyanazon néven szerepelnek egy kapcsolásban össze is vannak kötve, ezért nagyon fontos, hogy melyik lehetőség mellett döntünk.

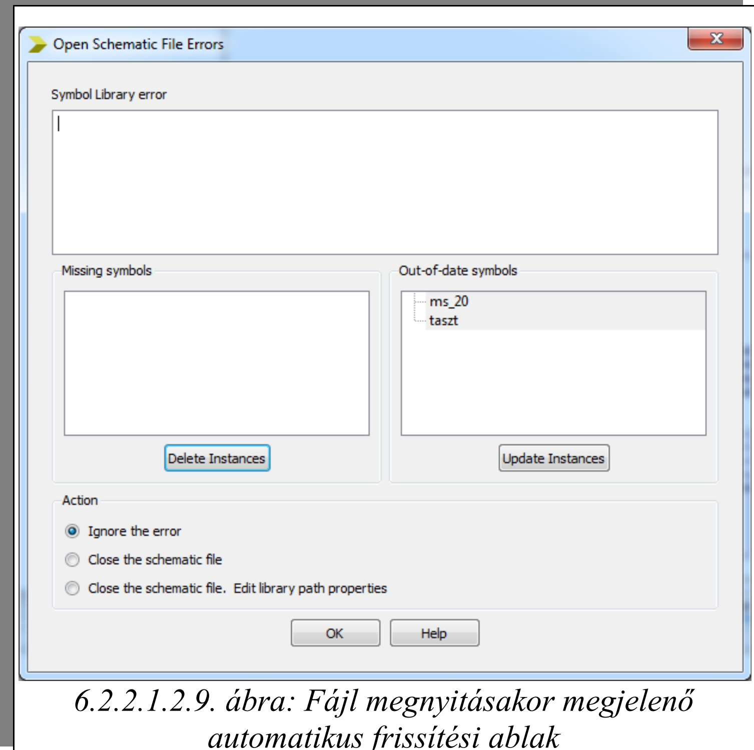
41 Kiválasztott busz, vezeték, objektum.
42 A kiválasztott szimbólumokat azonos függőleges, vagy vízszintes koordinátákra tudjuk helyezni. Nagyon hasznos lehet, ha adott szimbólumokat egy sorba, vagy oszlopba szeretnénk elhelyezni gyorsan. (Referenciaként megadható a bal, a jobb, a legfelső, vagy legalsó objektumok).
43 A *SHIFT*-et nyomva többszörös kijelölést végezhetünk, ha a csoportos kijelölésből el szeretnénk távolítani egy objektumot, akkor használjuk a *CTRL* billentyűt a kijelölésnél. A *Clear Highlighting* menüpont segítségével eltávolíthatjuk a *Tools* → *Query* menüponttal megjelölt (nem kijelölt!) alkatrészek háttérszínét. Részletesebben l. . fejezet.



6.2.2.1.2.8. ábra: Elavult szimbólumok kézi frissítése

A *View* menü *Update Obsolete Symbol* menüpontjának segítségével manuálisan tudjuk frissíteni módosított alkatrészeinket. Ezeket a módosításokat csak a szimbólum megváltozásakor kell megtennünk[44]. Amennyiben a szimbólum belső szerkezete, az általa leírt alkatrész működése alakult át, akkor erre nincs szükség.

Az *Obsolete Symbols* mezőben találjuk az elavult szimbólumokat. Az *Ignore Obsolete Symbols* mezőben pedig azon szimbólumainkat, amelyek ugyan megváltoztak, de ezeket a változásokat nem szeretnénk érvényre juttatni. A középen található nyilak segítségével

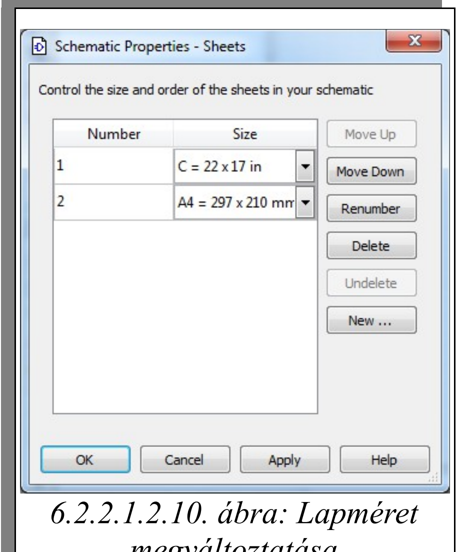


6.2.2.1.2.9. ábra: Fájl megnyitásakor megjelenő automatikus frissítési ablak

Amennyiben egy olyan fájlt nyitunk meg, amelyben szerepel egy módosított szimbólum, akkor a 6.2.2.1.2.9. ábrán látható ablak jelenik meg. A program megkérdezi, hogy a hibajelenséget hogyan szeretnénk kezelni.

Lehetőségünk van a módosított szimbólumok törlésére a fájlból (*Delete Instances*), vagy frissíthetjük őket az új verzióra (*Update Instances*). Az alsó részen választhatjuk a fájl bezárását, a fájl bezárását és a szimbólum útvonalának módosítását, vagy a hiba figyelmen kívül hagyását.

44 Pl. az IO lábak számának módosítása.

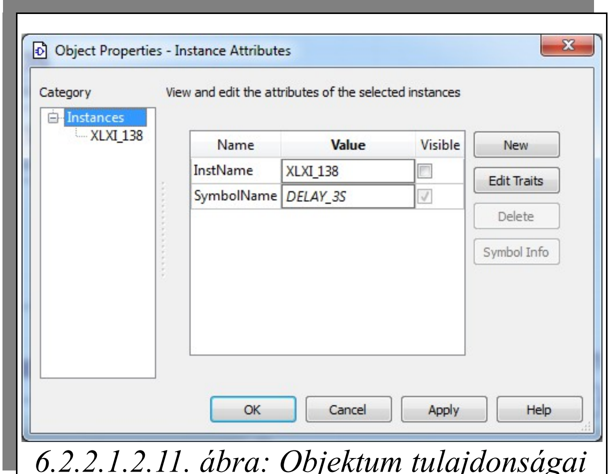


6.2.2.1.2.10. ábra: Lapméret megváltoztatása

A 6.2.2.1.2.10. ábrán láthatjuk a különböző lapok, lapméretek megadását. A *Move Up* és a *Move Down* gombokkal tudjuk megváltoztatni a lapok sorrendjét. A *Delete* gombbal törölhetjük a kiválasztott lapot, míg az *Undelete* gomb segítségével ezt vissza tudjuk vonni. A *New* gomb hatására egy új lapot kérhetünk. A *Size* legördülő sáv tartalmazza a gyakorlatban használt legtöbb szabványos lapformátumot.

A *Renumber* gomb hatására a program újraszámozza a lapokat 1-től kezdve sorrendben (ez a művelet nem vonható vissza).

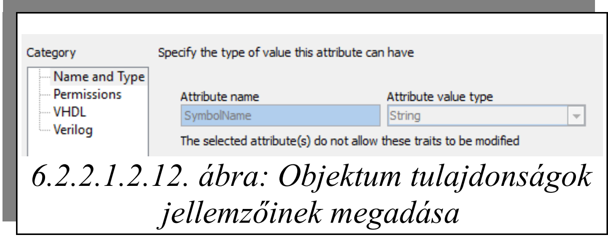
Amennyiben alkalmazni akarjuk a módosításokat kattintsunk az Apply gombra, majd az *OK*-ra. Amennyiben nem szeretnénk érvényre juttatni a változtatásokat, akkor kattintsunk a *Cancel* gombra. Ha szükséges a *Help* gombbal segítséget kérhetünk.



6.2.2.1.2.11. ábra: Objektum tulajdonságai

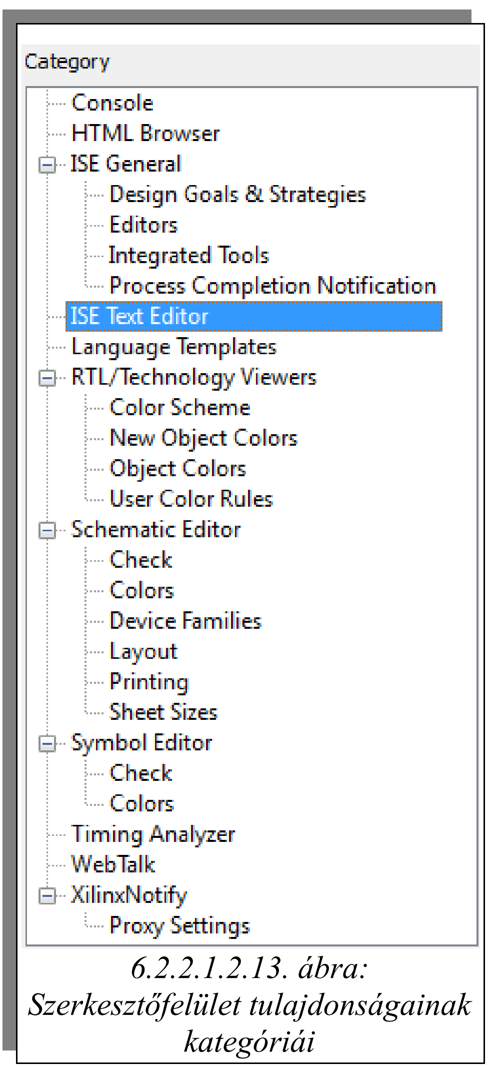
Az *Object Properties* menüpont segítségével beállíthatjuk objektumaink tulajdonságait. Megváltoztathatjuk egy szimbólum nevét (*SymbolName*), az adott példány nevét (*InstName*). A *Visible* opcióval a neveket láthatóvá tehetjük, vagy eltüntethetjük. Buszos csatlakozások esetén nevet adhatunk magának a busznak, ill. a buszhoz tartozó összes netet is elnevezhetjük. Eszköztől függően egyéb beállításokra is lehetőség van. A *Symbol Info* gyári szimbólumok esetén megnyitja a súgó adott szimbólumhoz tartozó részét. A *New* gombbal új jellemző is megadható.

Az *Edit Traits* gomb hatását a 6.2.2.1.2.12. ábrán láthatjuk.



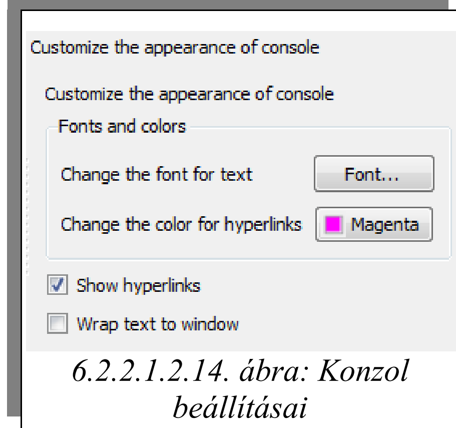
6.2.2.1.2.12. ábra: Objektum tulajdonságok jellemzőinek megadása

Gyári tulajdonságok esetén ez az opció csupán olvasható, nem szerkeszthető. A felhasználó által megadott tulajdonságok jellemzői szerkeszthetőek is (pl, név és típus, láthatóság, törölhetőség, stb.).



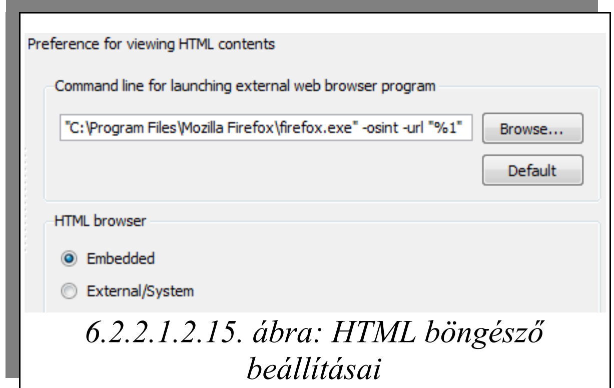
6.2.2.1.2.13. ábra: Szerkesztőfelület tulajdonságainak kategóriái

Consol: l. 6.2.2.1.2.14. ábra
HTML Browser: l. 6.2.2.1.2.15. ábra
ISE General: l. 6.2.2.1.2.16. ábra
Design Goals & Strategies: l. 6.2.2.1.2.17. ábra
Editors: l. 6.2.2.1.2.18. ábra
Integrated Tools: l. 6.2.2.1.2.19. ábra
Process Completion Notification: l. 6.2.2.1.2.20. ábra
ISE Text Editor: l. 6.2.2.1.2.21. ábra
Language Templates: l. 6.2.2.1.2.22. ábra
RTL/Technology Viewer: l. 6.2.2.1.2.23. ábra
Color Scheme: l. 6.2.2.1.2.24. ábra
New Object Colors: l. 6.2.2.1.2.25. ábra
Object Colors: l. 6.2.2.1.2.26. ábra
User Color Rules: l. 6.2.2.1.2.27. ábra
Schematic Editor: l. 6.2.2.1.2.28. ábra
Check: l. 6.2.2.1.2.29. ábra
Colors: l. 6.2.2.1.2.30. ábra
Devices Families: l. 6.2.2.1.2.31. ábra
Layout: l. 6.2.2.1.2.32. ábra
Printing: l. 6.2.2.1.2.33. ábra
Sheet Sizes: l. 6.2.2.1.2.34. ábra
Symbol Editor: l. 6.2.2.1.2.35. ábra
Check: l. 6.2.2.1.2.36. ábra
Colors: l. 6.2.2.1.2.37. ábra
Timing Analyzer: l. 6.2.2.1.2.38. ábra
WebTalk: l. 6.2.2.1.2.39. ábra
XilinxNotify: l. 6.2.2.1.2.40. ábra
Proxy Settings: l. 6.2.2.1.2.41. ábra



6.2.2.1.2.14. ábra: Konzol beállításai

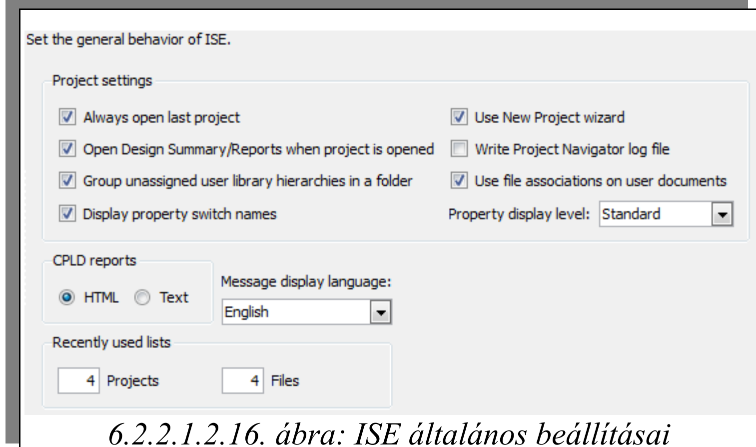
A 6.2.2.1.2.14. ábrán a konzol panel megjelenésének testreszabását láthatjuk. Beállíthatjuk a betűkészletet, a betűtípust, a betűméretet, (*Font*), valamint a hiperhivatkozások színét. A *Show hyperlinks* opciót kipipálva a konzolban a program mutatni fogja a hiperhivatkozásokat. A *Wrap text to window* opcióval a konzolban megjelenő szöveg tördelésre kerül a panel ablakának mérete szerint.



6.2.2.1.2.15. ábra: HTML böngésző beállításai

A *Command line for launching external web browser program* mezőben be tudjuk állítani, hogy mely program milyen parancsot hajtson végre, ha a Xilinx-ban hiperhivatkozásra kattintunk[45]. A HTML browser résznél ki tudjuk választani az alkalmazni kívánt böngészőt (beépített, vagy az oprendszer alapértelmezettje).

Általános beállítások



6.2.2.1.2.16. ábra: ISE általános beállításai

A 6.2.2.1.2.16. ábrán láthatjuk a Xilinx ISE általános beállításait. Fent a projekt beállításokat, míg lent a CPLD riportokkal kapcsolatos beállításokat módosíthatjuk. Legalul beállíthatjuk, hogy a legutóbb használt listák közül hány darab kerüljön mentésere.

Always open last project: Mindig nyissa meg a legutóbbi projektet.

Open Design Summary/Reports when project is opened: Tervezési összegző fájl automatikus megnyitása projekt megnyitása esetén.

Group unassigned user library hierarchies in a folder: A fel nem használt felhasználói modulok a Design panelen külön könyvtárban történő elhelyezése.

Display property switch names: A *Process → Process Properties* menüpontban a kapcsolók szerepeltetése (részletesebben l. 6.2.2.1.6. fejezet).

Use New Project wizard: Új projekt megnyitása esetén a varázsló alkalmazása.

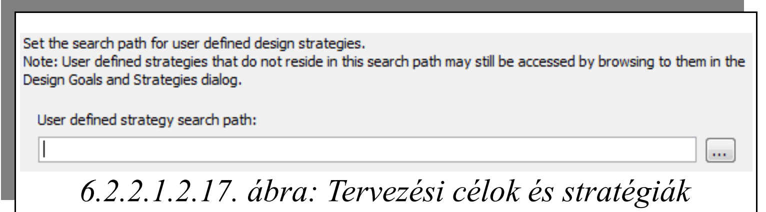
Write Project Navigator log file: Log fájl készítése a Projekt navigátor által. Amennyiben be van jelölve, akkor létrehozásra kerül egy log fájl, ami a konzol kimenetét tartalmazza.

45 A default érték, hogy az operációs rendszer alapértelmezett böngészője új lapon nyitja meg a linket.

Use file associations on user documents: A felhasználói szöveges dokumentumokhoz társított program (ha be van jelölve, akkor a windowsban alapértelmezett program pl. Notepad, egyébként az *Editors*[46] lapon megadott szövegszerkesztő).

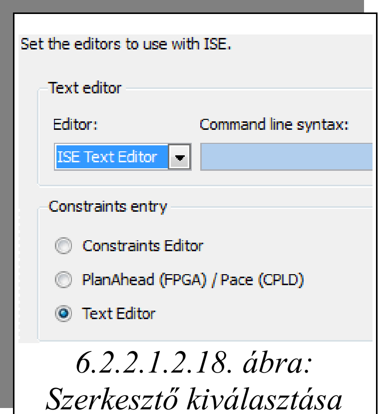
Property display level: A *Process* → *Process Properties* menüpontban való megjelenés beállítása. Lehetőségek: szokásos (standard), vagy haladó (Advanced). Részletekért l. 6.2.2.1.6. fejezet.

A *CPLD* jelentések formátuma lehet *HTML*, vagy szöveges (*Text*). Az üzenet nyelve alapértelmezetten angol, de alkalmazható a japán nyelv is.



6.2.2.1.2.17. ábra: Tervezési célok és stratégiák

A 6.2.2.1.2.17. ábrán adhatjuk meg a felhasználó által definiált tervezési célokat és stratégiákat. Részletekért l. 6.2.2.1.4. fejezet.



6.2.2.1.2.18. ábra: Szerkesztő kiválasztása

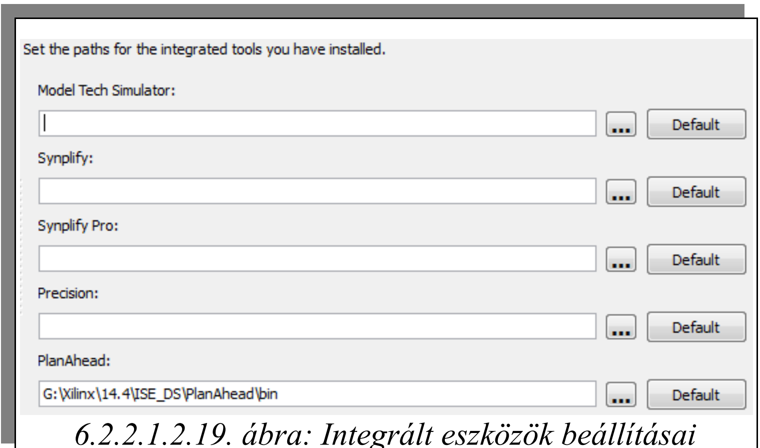
A 6.2.2.1.2.18. ábrán láthatjuk az ISE által használt szövegszerkesztő kiválasztását. Az *Editor* mezőben lehetőségünk van az *ISE Text Editor* (alapértelmezett), és saját (*Custom*) szövegszerkesztő kiválasztására is[47]. Utóbbi esetben a *Command line syntax* mezőben adjuk meg a megnyitáskor végrehajtandó parancsot a programnak. A parancssorban megadhatjuk a $1 (a fájl neve) és $2 (sorszám a fájlon belül) szimbólumokat. A Xilinx által megadott példa: *C:\windows\wordpad.exe $1*[48]

A *Constraints entry* ablakrészben válasszuk ki, hogy mivel történjen az UCF fájlok megnyitása. Az UCF fájlok kezelésére a 74. oldalon térünk ki.

46 l. 6.2.2.1.2.18. ábra.

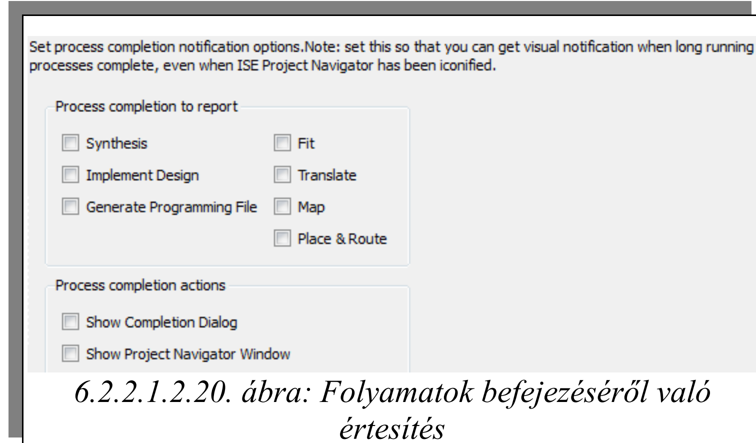
47 Windowsos operációs rendszer esetén lehetőség van az Ultraedit futtatására is, amely egy külön ablakban futó szerkesztő.

48 Amennyiben szóközöket tartalmaz az elérési út akkor kapcsos zárójelek közé kell tennünk. A gyártó által javasolt formátum: *{C: \ Program Files \ Windows NT \ Kellékek \ wordpad.exe} $ 1*



6.2.2.1.2.19. ábra: Integrált eszközök beállításai

A 6.2.2.1.2.19. ábrán láthatjuk az *ISE*-be integrált és telepített eszközök elérési útjainak megadását. A *Model Tech Simulator* mezőben megadhatjuk a modelsim elérési útját. A *Synplify* és a *Precision* mezőkben szintézis eszközök adhatóak meg.



6.2.2.1.2.20. ábra: Folyamatok befejezéséről való értesítés

Az FPGA-ra történő fejlesztés során meg kell barátkoznunk a hosszú folyamat-végrehajtási időkkel. A fejlesztőkörnyezet ezért lehetőséget biztosít számunkra, hogy egyes folyamatok befejezéséről értesítést küldjön számunkra a program. Ezen értesítések akkor is megjelennek (a Projekt Navigátor megnyílik), ha a programot minimalizáljuk és épp mással foglalkozunk.

ISE beépített szövegszerkesztő beállításai

Control the behavior of the ISE Text Editor

When you press the Tab key:
Insert spaces
Insert a tab character
Tab width: 3
Long line marker location: 80
Use black color scheme
Show line numbers
Show outline
Text Editor font
Change

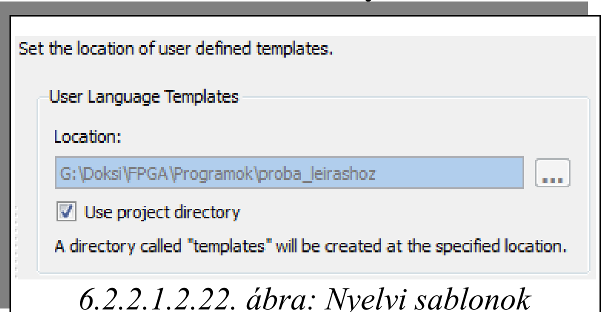
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

6.2.2.1.2.21. ábra: ISE szövegszerkesztő beállításai

A *When you press the Tab key* mezőben adjuk meg, hogy szóközt, vagy tabulátort szeretnénk beszúrni a Tab billentyű megnyomásakor. Beállíthatjuk a tabulátor szélességét is. A *Long line marker location* mezőben adjuk meg, hogy a *View → Long line marker* opció használatakor hova kerüljön a jelölő vonal (l. 6.2.2.1.3. fejezet). A *Use black color scheme* bejelölésekor a szövegszerkesztő átvált fekete háttérszínre és a hozzá illő előtérszínekre.

A *Show line numbers* opció használatakor az editor bal oldalon szerepelteti a sorok számát. A *Show outline* bepipálásának eredménye, hogy a bal oldalon a margón megjelennek kis vonalak, melyek segítségével az összetartozó egységek összecsukhatóak (elrejthetőek) lesznek (pl. összefüggő megjegyzések; az adott nyelv összetett utasításai). Használata megkönnyíti a dokumentálást és az átláthatóságot. Ez az opció elérhető a *View* menüből is. A *Text Editor font* mezőben láthatjuk a betűkészletet, a betűtípust, a betűméretet, amiket a *Change* gomb segítségével változtathatunk meg amennyiben ez szükséges.

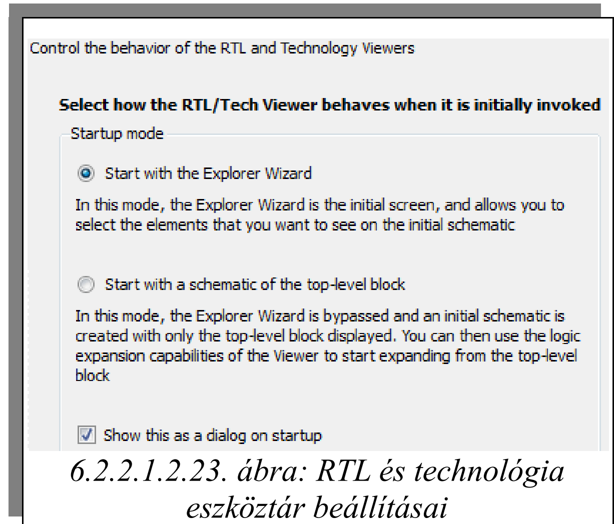
Nyelvi eszközök



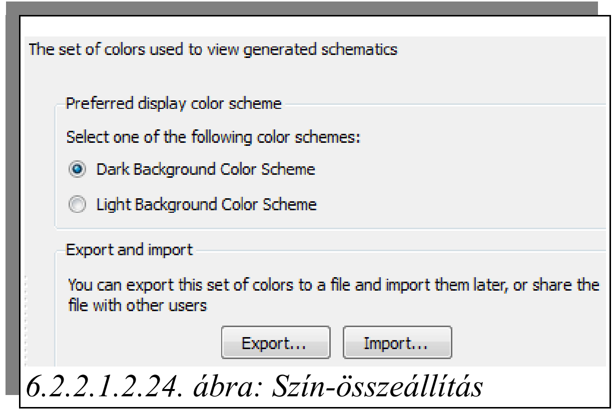
6.2.2.1.2.22. ábra: Nyelvi sablonok

A 6.2.2.1.2.22. ábrán a felhasználó nyelvi sablonjainak helyét tudjuk megadni. A Xilinx számos ilyen sablont nyújt a könnyebb fejlesztés érdekében. Ezeket az *Edit* → *Language Templates* menüben találjuk. Ilyen sablonokat mi is készíthetünk későbbi munkánk egyszerűsítésére.

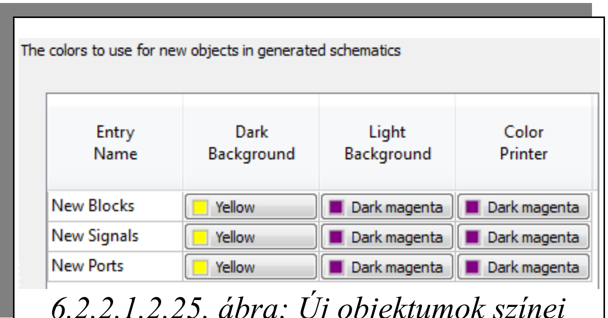
RTL és technológia eszköztár beállításai



6.2.2.1.2.23. ábra: RTL és technológia eszköztár beállításai



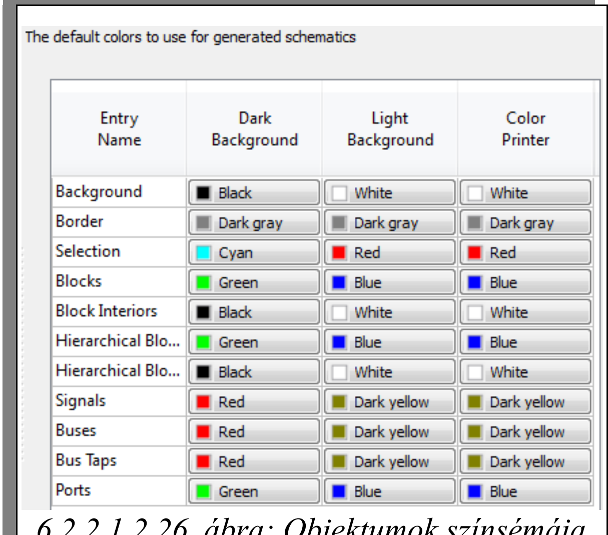
6.2.2.1.2.24. ábra: Szín-összeállítás



Entry Name	Dark Background	Light Background	Color Printer
New Blocks	Yellow	Dark magenta	Dark magenta
New Signals	Yellow	Dark magenta	Dark magenta
New Ports	Yellow	Dark magenta	Dark magenta

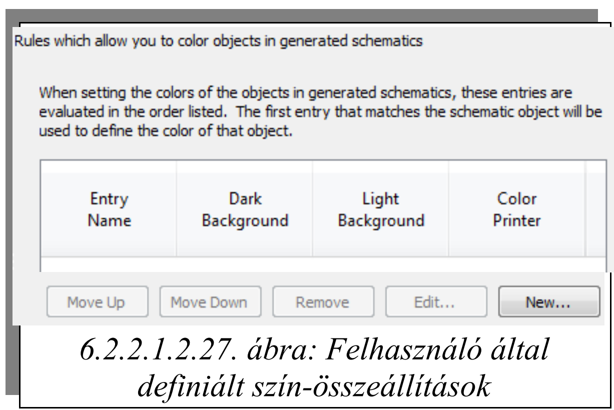
6.2.2.1.2.25. ábra: Új objektumok színei

Az új objektumok generálása esetén alkalmazott színsémákat a 6.2.2.1.2.25. ábrán láthatjuk.



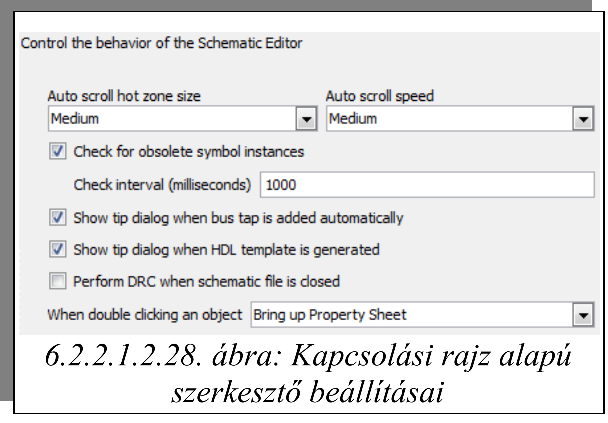
Entry Name	Dark Background	Light Background	Color Printer
Background	Black	White	White
Border	Dark gray	Dark gray	Dark gray
Selection	Cyan	Red	Red
Blocks	Green	Blue	Blue
Block Interiors	Black	White	White
Hierarchical Blo...	Green	Blue	Blue
Hierarchical Blo...	Black	White	White
Signals	Red	Dark yellow	Dark yellow
Buses	Red	Dark yellow	Dark yellow
Bus Taps	Red	Dark yellow	Dark yellow
Ports	Green	Blue	Blue

6.2.2.1.2.26. ábra: Objektumok színsémája



6.2.2.1.2.27. ábra: Felhasználó által definiált szín-összeállítások

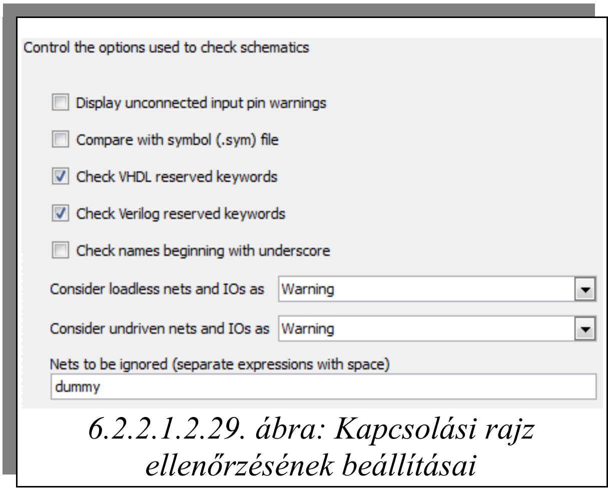
Kapcsolási rajz alapú szerkesztő beállításai



6.2.2.1.2.28. ábra: Kapcsolási rajz alapú szerkesztő beállításai

A 6.2.2.1.2.28. ábrán a *schematic editor* megjelenését és viselkedését tudjuk beállítani. Az *Auto scroll hot zone size* mezőben megadhatjuk, hogy az ablak szélétől milyen távolságra aktív az automatikus görgetés funkció. Az *Auto scroll speed* mezőben megadható az automatikus görgetés sebessége.

A *Check for obsolete symbol instances* opció bejelölésével beállíthatjuk hogy a program automatikusan ellenőrizze az elavult szimbólumokat, amennyiben ilyet talál megnyílik az ehhez tartozó párbeszédpanel (l. 6.2.2.1.2.8. ábra). A *Check interval* mezőbe írjuk be az ellenőrzés időközét. A *Show tip dialog when bus tap is added automatically* opció bejelölése esetén busz táp automatikus hozzáadásakor párbeszédablakot jelenít meg a program. A *Show tip dialog when HDL template is generated* opció bejelölése esetén HDL sablon generálásakor párbeszédablak jelenik meg. A *Perform DRC when schematic file is closed* opció a kapcsolási rajz bezárásakor automatikusan futtatja a DRC[49]-t. A *When double clicking an object* mezőben megadhatjuk, hogy egy objektumon való dupla kattintás milyen eredménnyel záruljon. Alapértelmezetten az *Edit → Object properties* menüpont nyílik meg (l. 6.2.2.1.2.11. ábra).



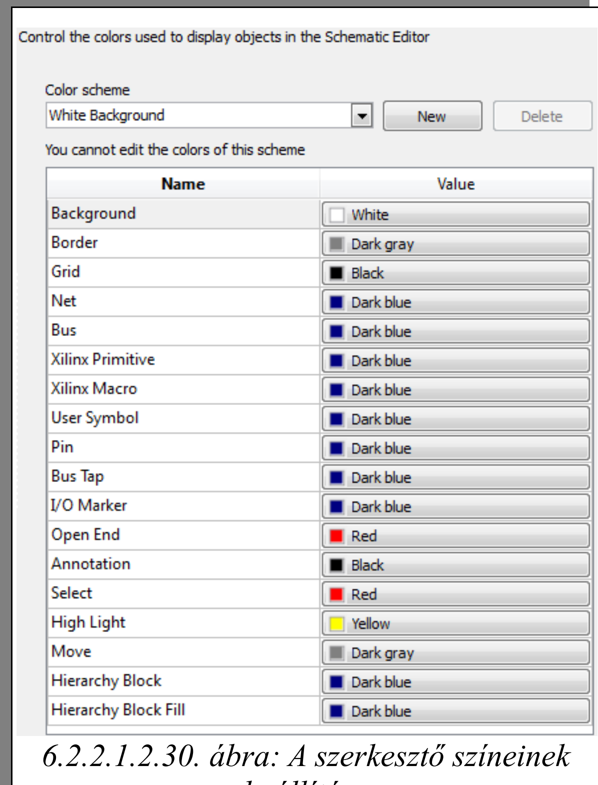
6.2.2.1.2.29. ábra: Kapcsolási rajz ellenőrzésének beállításai

A 6.2.2.1.2.29. ábrán láthatjuk a DRC beállításait. A *Display unconnected input pin warnings* opció veszélyként fogja mutatni azon lábakat, amik nem csatlakoznak sehova. A *Compare with symbol (.sym) file* opció összehasonlítja a *schematic* I/O markereinek nevét és polaritását az ugyanazon nevű szimbólummal. A *Check VHDL reserved keywords* ellenőrzi a VHDL nyelv kulcsszavainak használatát[50].

A *Check Verilog reserved keywords* a Verilog nyelv kulcsszavainak használatát. A *Check names beginning with underscore* ellenőrzi az aláhúzással kezdődő neveket. A *Consider loadless nets and Ios as és a Consider undriven nets and Ios as* mezőben kiválaszthatjuk, hogy

49 *Design Rules Check* – Tervezési szabályok ellenőrzése

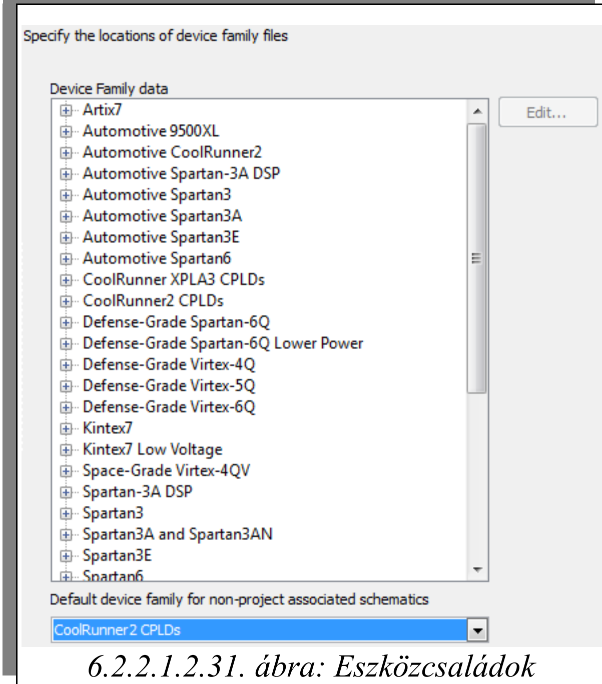
50 Minden nyelvnek vannak fenntartott kulcsszavaik, amelyeket nem alkalmazhatunk pl. változónévként. Az opció bejelölésekor ellenőrzi a *schematicban*, hogy a lábak, buszok, szimbólumok nem ütköznek-e a kulcsszavakkal.



6.2.2.1.2.30. ábra: A szerkesztő színeinek beállítása

A 6.2.2.1.2.30. ábrán láthatjuk a kapcsolási rajz szerkesztő színeinek beállítását. A *Color scheme* mezőben adjuk meg a háttérszínt, majd sorban a következő entitások színeit:

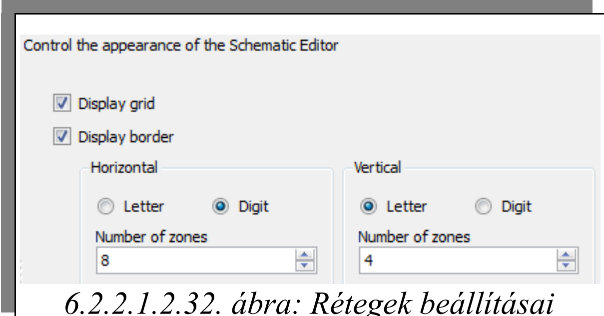
Background: háttér
Border:
Grid: rács
Net: vezeték
Bus: buszos csatlakozás
Xilinx Pimitive: xilinx egység
Xilinx Macro: xilinx makró
User Symbol: felhasználói szimbólum
Pin: lábak
Bus Tap: busz tápok
I/O Marker: be-, kimeneti portok
Open End: szabad végek
Annotation: megjegyzés
Select: kiválasztott objektumok
High Light: kiemelés
Move:
Hierarchy Block:
Hierarchy Block Fill:



6.2.2.1.2.31. ábra: Eszközcsaládok

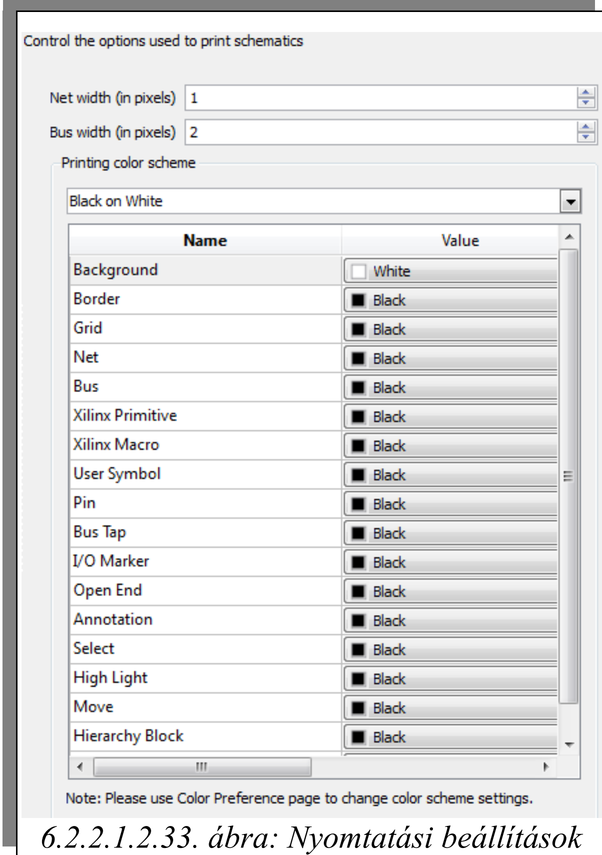
A 6.2.2.1.2.31. ábrán láthatjuk a Xilinx által kezel eszközcsaládok fájljainak helyét. Kiválaszthatjuk a makró, a szimbólum kategória fájlok (.cat) és a szimbólum könyvtár fájlok (.lib) elérési útját. Az *Edit* gombra kattintva az útvonal szerkeszthetővé válik.

Az alul látható *Default device family for non-project associated schematics* mezőben kiválaszthatjuk, hogy ha projekt nélkül nyitunk meg egy schematic fájlt, akkor milyen eszközcsaládot vegyen alapul a program.



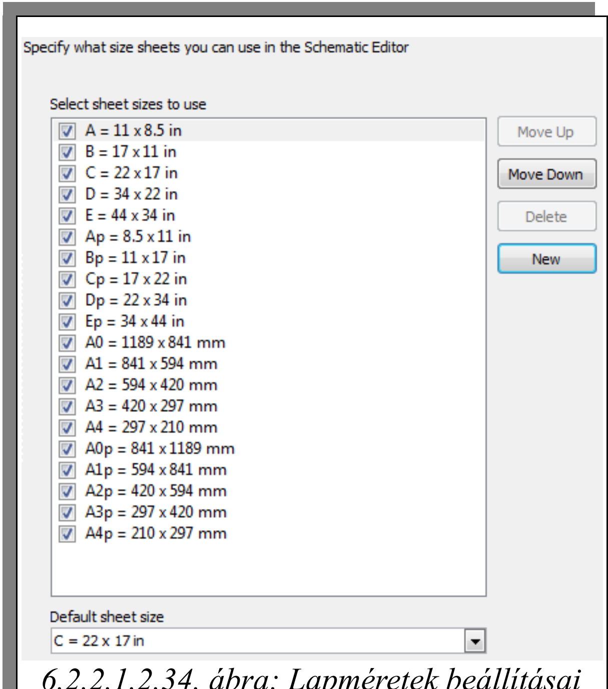
6.2.2.1.2.32. ábra: Rétegek beállításai

Szegély és rács beállítása. A *Display grid* opcióval a rács láthatóvá válik. A *Display border* bepipálására a lap szélén egy szegély jelenik meg az alul látható zónákra osztva (A *Letter* betűk, míg a *Digit* számjegyekkel címkézi fel a zónákat).



6.2.2.1.2.33. ábra: Nyomtatási beállítások

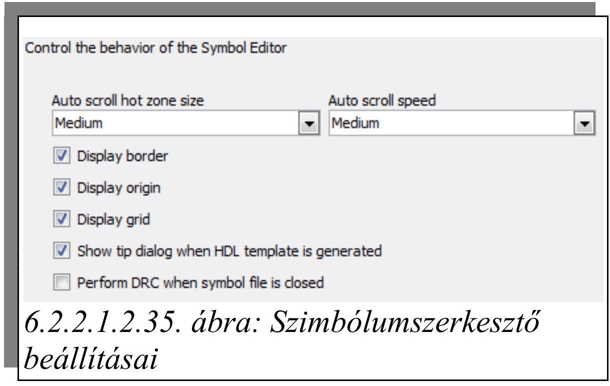
A 6.2.2.1.2.33. ábrán látható a nyomtatási képen, ill. nyomtatásban megjelenő színösszeállítás. Alapértelmezetten mindent fekete-fehérben nyomtat a program. A *Net widt (in pixels)* mezőben beállíthatjuk pixelben a vezetékek vastagságát nyomtatáskor. A *Bus widt (in pixels)* mezőben beállíthatjuk pixelben a buszok vastagságát nyomtatáskor. A *Printing color scheme* mezőben megadhatjuk a nyomtatási színsablont. Ami lehet fekete-fehér, vagy színes nyomtatás fehér háttérrel, ill. fekete háttérrel. Külön-külön az egyes színek itt nem állíthatóak. Amennyiben egyes entitások színeit szeretnénk megváltoztatni, akkor ezt a 6.2.2.1.2.30. ábrán látható ablakban tehetjük meg.



6.2.2.1.2.34. ábra: Lapméretek beállításai

A 6.2.2.1.2.34. ábrán beállíthatjuk, hogy milyen lapformátumokat szeretnénk használni a kapcsolási rajz alapú szerkesztőben. Az aktuális lapformátumot az *Edit → Change Sheet Size* menüpont segítségével (6.2.2.1.2.10. ábra) adhatjuk meg. Az meglévő formátumok között szerepelnek az amerikai és európai szabványos oldalak – álló (pl. *A0*) és fekvő (pl. *A0p*) tájolással egyaránt. A *New* gomb segítségével új lapformátumot tudunk hozzáadni, míg a *Delete* gombbal törölhetünk egy már meglévőt. A *Default sheet size* mezőben kiválaszthatjuk hogy milyen legyen az alapértelmezett lapformátum minden új lap esetén.

Szimbólumszerkesztő beállításai

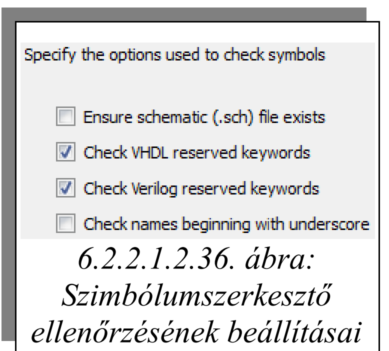


6.2.2.1.2.35. ábra: Szimbólumszerkesztő beállításai

A 6.2.2.1.2.35. ábrán láthatjuk a szimbólumszerkesztőhöz tartozó beállításokat. Az *Auto scroll hot zone size* mezőben megadhatjuk, hogy az ablak szélétől milyen távolságra aktív az automatikus görgetés funkció. Az *Auto scroll speed* mezőben megadható az automatikus görgetés sebessége.

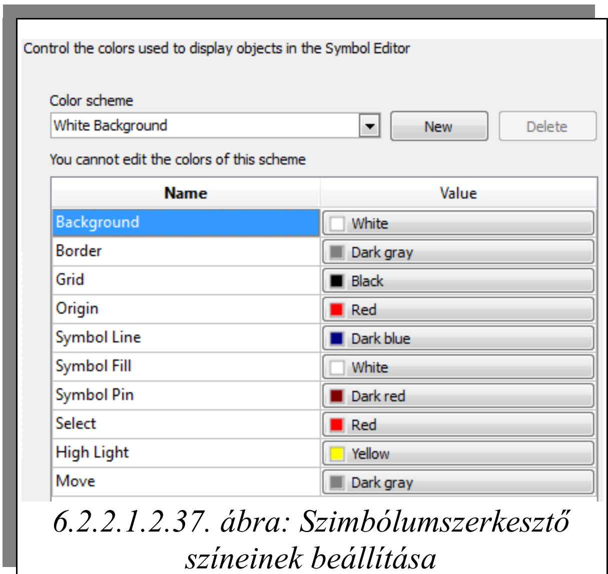
A *Display border* bejelölésével láthatóvá válik szimbólum határfelülete (az a tartomány, amire kattintva a schematic editorban a szimbólumra hivatkozhatunk) A *Display origin*, *Display grid* bejelölése megjeleníti a középpontot és a rácsot. A *Show tip dialog when HDL template is generated* bepipálásával HDL sablon generálásakor buboréktipp jelenik meg. A *Perform DRC when symbol file* is closed lehetővé teszi, hogy a szimbólumfájlok bezárásakor automatikusan DRC[51] fusson le.

51 Design Rules Check – Tervezési szabályok ellenőrzése



6.2.2.1.2.36. ábra: Szimbólumszerkesztő ellenőrzésének beállításai

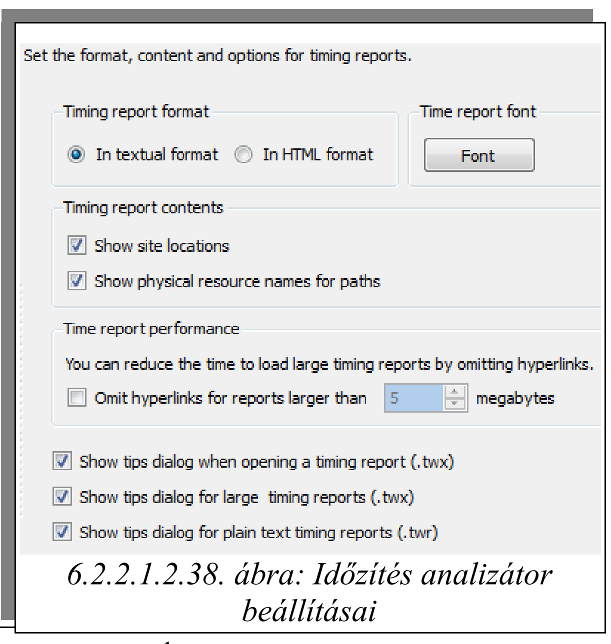
A 6.2.2.1.2.36. ábrán láthatjuk, hogy milyen opciókat lehet beállítani egy szimbólum ellenőrzésekor (DRC). Az *Ensure schematic (.sch) file exists* ellenőrzi, hogy a szimbólumhoz (.sym) tartozó schematic (.sch) fájl létezik-e. A *Check VHDL reserved keywords* és a *Check Verilog reserved keywords* opciók ellenőrzik a VHDL és a Verilog lefoglalt kulcsszavait (l. 8.2.1. fejezet). A *Check names beginning with underscore* bejelölése esetén a szimbólumunk ellenőrzésekor a program megvizsgálja, hogy a szimbólum neve nem kezdődik-e aláhúzással[52].



6.2.2.1.2.37. ábra: Szimbólumszerkesztő színeinek beállítása

Az 6.2.2.1.2.37. ábrán a szimbólumszerkesztő színpalettáját tudjuk beállítani. A *Color scheme* mezőben válasszuk ki a színsablont (egyéni palettát a *New* gomb segítségével tudunk létrehozni). Az alapértelmezett paletták esetén nincs módunk a színek megváltoztatására. Amennyiben nem vagyunk elégedettek a felkínált színösszeállításokkal hozzunk létre saját sémát.

Időzítés analizátor beállításai



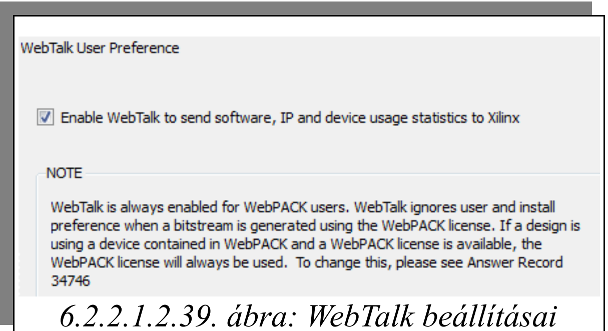
6.2.2.1.2.38. ábra: Időzítés analizátor beállításai

Az időzítés analizátor a háttérben fut a tervezéskor és a különböző útvonalakon haladó jelek időzítéseit, a kritikus időket, a ciklusidőket, a szinkron és aszinkron szekvenciális, valamint a kombinációs hálózatok útvonalait tervezi meg a sebesség és az időzítés optimalizálása érdekében. A *Timing report format* résznél kiválaszthatjuk a szöveges és a HTML formátumban való megjelenítést. A *Font* gombbal beállíthatjuk a betűtípust, betűkészletet s a betűméretet.

52 VHDL-ben ez nem megengedett.

A *Timing report contents* mezőben válasszuk ki, hogy az időzítés analizátor jelentése milyen elemeket tartalmazzon (pozíció, erőforrás neve, útvonala). A *Time report performance* beállításnál lerövidíthetjük a jelentést a hosszú hivatkozások elhagyásával. Az utolsó résznél a buboréktippeket állíthatjuk be.

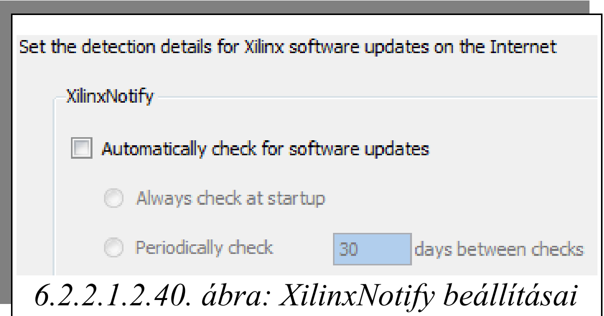
Webes beállítások



6.2.2.1.2.39. ábra: WebTalk beállításai

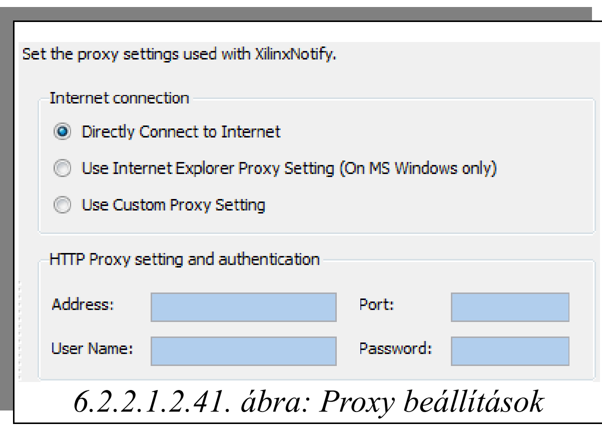
A WebTalk a Xilinx felhasználói élményjavító szolgáltatásához kapcsolódik. Sikeres konfigurációs bitminta generálásakor jelentést küld a Xilinxnak, ami különböző statisztikákat tartalmaz (konfigurációs adatok, lábak használata, Block RAM használat, parancssori információk, stb.).

A Xilinx nem gyűjt olyan adatokat, amelyekből a terv visszafejthető, a programozók szellemi tulajdonát tiszteletben tartják. Webpack licenc esetén ez a funkció nem tiltható le.



6.2.2.1.2.40. ábra: XilinxNotify beállításai

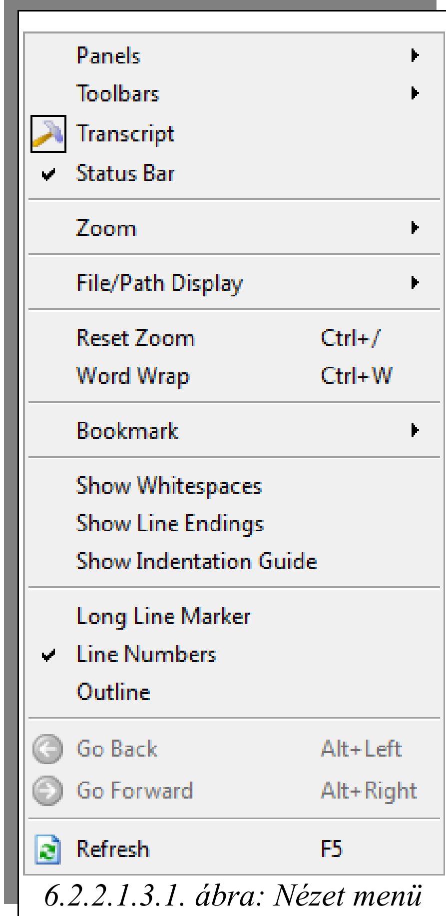
A Xilinx értesítési ablakát az 6.2.2.1.2.40. ábrán láthatjuk. Beállíthatjuk, hogy a program automatikusan keressen szoftverfrissítéseket az interneten. Amennyiben ezt az opciót választjuk, lehetőségünk van minden indítás esetén ellenőrzést kérni, ill. csak periodikusan adott napok elteltével.



6.2.2.1.2.41. ábra: Proxy beállítások

Az 6.2.2.1.2.41. ábrán láthatjuk, az internetelérés beállítását. Biztosíthatunk közvetlen elérést (*Directly Connect to Internet*), Windows operációs rendszer esetén használhatjuk az internet explorer beállításait (*Use Internet Exlorer Proxy Setting*), vagy saját egyéni proxy beállítást is megadhatunk (*Use Custom Proxy Setting*).

6.2.2.1.3. View menü



6.2.2.1.3.1. ábra: Nézet menü

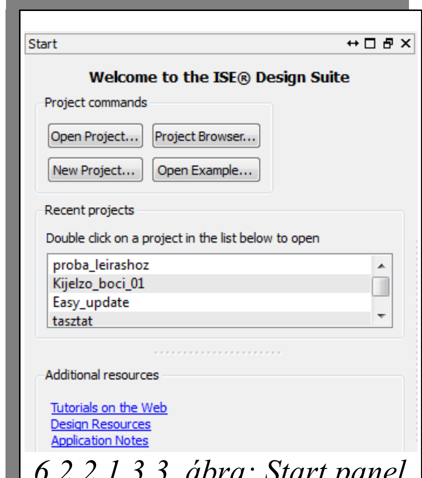
Panels:	Panelek (l. 6.2.2.1.3.2. ábra)[53]
Toolbars:	Eszköztárak (l. 6.2.2.1.3.11. ábra)
Transcript:	Átirat (l. 6.2.2.1.3.12. ábra)
Status Bar:	Állapotsor mutatása (l. 6.2.2.1.3.13. ábra)
Zoom:	Nagyítás (l. 6.2.2.1.3.14. ábra)
File/Path Display:	Fájlnév/út mutatása (l. 6.2.2.1.3.15. ábra)
Reset Zoom:	Nagyítás visszavonása
Word Wrap:	Szótördelés[54]
Bookmark:	Könyvjelzők[55]
Show Whitespaces:	Üres helyek mutatása[56]
Show Line Endings:	Sor végek mutatása
Show Indentation Guide:	Behúzások mutatása
Long Line Marker:	Egy függőleges vonal jelzi az *Edit* → *Preferences* menüben (6.2.2.1.2.21. ábra) megadott szélességet.
Line Numbers:	Sorok számának mutatása
Outline:	Vázlatjelzések[57]
Go Back:	Vissza az előző nézetre
Go Forward:	Következő nézet
Refresh:	Frissítés

53 Egyes panelek szövegfájlok esetén nem jeleníthetőek meg.
54 Amennyiben bekapcsoljuk ezt a funkciót, akkor a sorok végét a program automatikusan tördeli a nyomtathatóság kedvéért.
55 Könyvjelző lerakása (*Toggle Bookmark*), következő könyvjelzőre ugrás (*Next Bookmark*), előző könyvjelzőre ugrás (*Previous Bookmark*), minden könyvjelző törlése (*Delete All Bookmark*).
56 A szóközök ponttal, míg a tabulátorok nyíllal kerülnek jelölésre.
57 A szövegfájl bal oldalán kis vonásokkal kerülnek jelzésre az egyes utasítások/kommentek kezdetei.

Start
Options
Symbols
Design
Libraries
Find in Files Results
Files
Warnings
Errors
Timing Constraints
Tcl Console
Console

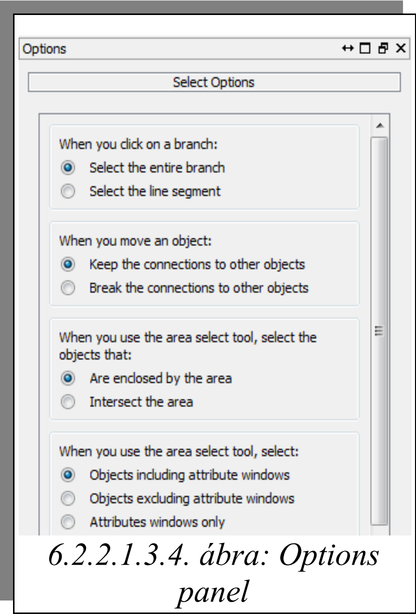
6.2.2.1.3.2. ábra: Panelek menüpont

Start:	Alapvető projektindítási műveletek
Options:	Beállítások
Symbols:	Schematic fájlok szimbólumai
Design:	Tervezési lépések, lehetőségek
Libraries:	Könyvtárak
Find in Files Results:	Fájlokban történő keresések eredményei
Files:	Fájlok
Warnings:	Veszélyek
Errors:	Hibák
Timing Costraints:	Időzítési beállítások
Tcl Console:	TCL konzol
Console:	Konzol



6.2.2.1.3.3. ábra: Start panel

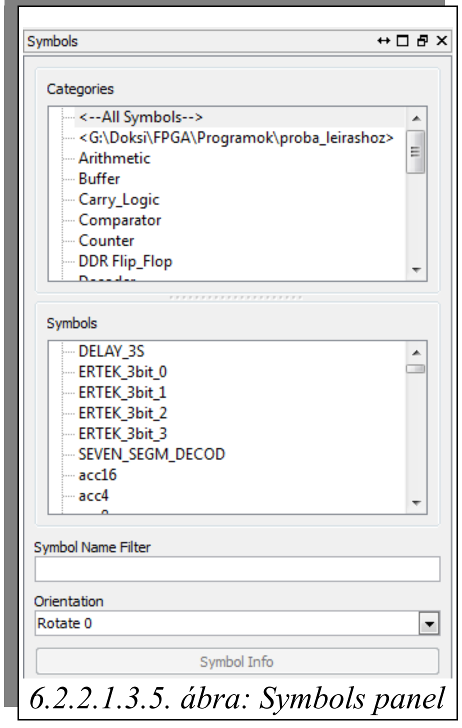
A *Start* panel nevéből adódóan a fejlesztés kezdetekor nyújt segítséget. Itt láthatjuk a legutóbb használt projekteket (*Recent projects*), ill. a projektek manipuláláshoz szükséges alapvető parancsokat (megnyitás, példák, új, intéző). Az ablak alsó részén webes támogatást találunk (mindhárom lehetőséghez szükséges internetes kapcsolat). A panelt – amennyiben nincs rá szükség – a jobb felső sarokban található *X*-el tudjuk bezárni, vagy a *View* → *Panels* menüpontban. Ez az út a további panelek esetén is járható.



6.2.2.1.3.4. ábra: Options panel

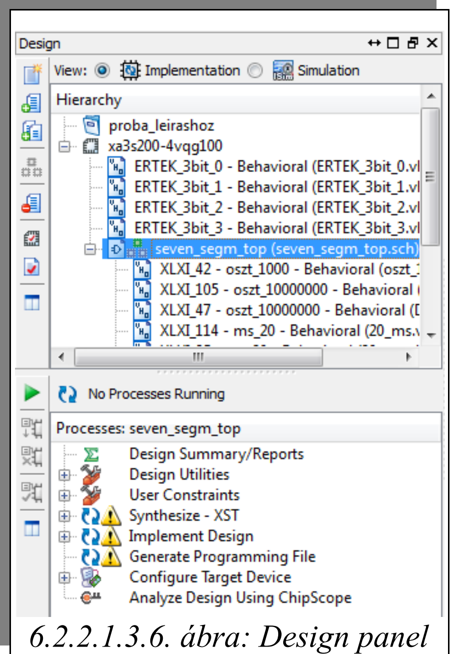
Az Options panelben az éppen aktuális művelet lehetőségei közül tudunk választani. Pl. a 6.2.2.1.3.4. ábrán látható esetben a kijelölés műveleteinek opcióit láthatjuk. Az első lehetőségben kiválaszthatjuk, hogy egy elágazásra kattintás esetén az egész netet, vagy csak a vezeték adott részét jelölje ki a program. Amennyiben területet jelölünk ki kiválaszthatjuk, hogy a megadott területtel érintkező, vagy az abban bennfoglalt objektumok kerüljenek kijelölésre. Számtalan beállítás lehet még a különböző eszközök esetén, ezekre részletesen majd a példák megoldásánál térünk ki. Amennyiben valamilyen műveletet szeretnénk elvégezni mindig érdemes megnézni az *Options* panelt, mert sok beállítása nagyban

megkönnyítheti a fejlesztést.



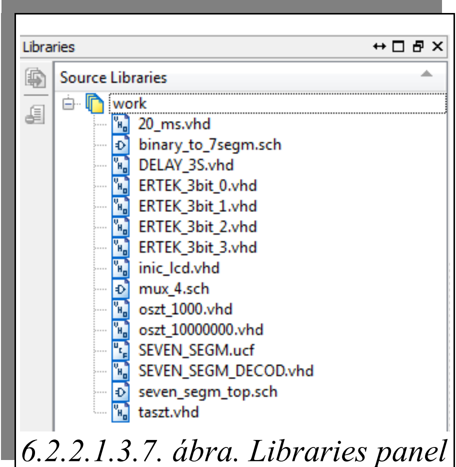
6.2.2.1.3.5. ábra: Symbols panel

A *Symbols* panelen találhatjuk a schematic típusú fájlok esetén alkalmazható különböző szimbólumokat. A *Categories* mezőben célszerű kiválasztani, hogy milyen típusú alkatrészre van szükségünk. Az így szűkített elemeket a *Symbols* mezőben láthatjuk. Az alkatrészek sokszínűsége függ az éppen fejlesztett típustól is (egy CPLD pl. sokkal kevesebb lehetőséggel rendelkezik, mint egy FPGA). Itt találhatjuk meg az általunk megalkotott szimbólumokat is. Amennyiben konkrét típus(oka)t név szerint szeretnénk keresni, használjuk a *Symbol Name Filter* mezőt. Az *Orientation* mezőben kiválaszthatjuk, hogy a lerakandó alkatrész milyen helyzetben kerüljön elhelyezésre a kapcsolási rajz alapú szerkesztőben. Forgatni s tükrözni is tudjuk az alkatrészt (ezt a későbbiek folyamán, a lerakás után is megtehetjük).



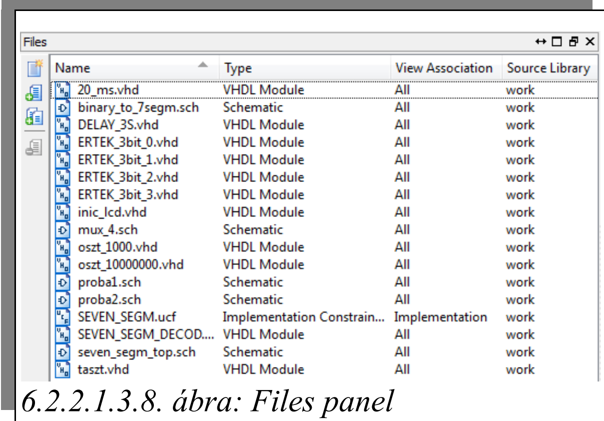
6.2.2.1.3.6. ábra: Design panel

A *Design* panelen láthatjuk a tervezéshez és a projekt fejlesztéséhez legszükségesebb parancsokat, lehetőségeket. Fontos, hogy a fent található nézetben kiválasszuk, hogy szimulációt (*Simulation*) szeretnénk, vagy élesben (*Implementation*) kipróbálni majd a fejlesztést. A *Hierarchy* mezőben láthatjuk a projektünkhöz tartozómodulokat a legfelső forrás kitüntetett helyen zöld színnel kerül megjelölésre. A nem használt, de csatolt források (*Ertek*) szintén külön vannak csoportosítva. Az alsó mezőben a különböző folyamatok figyelhetőek meg (l. 6.2.2.1.6.1. ábra). Mindkét mező bal oldalán ikonok segítik a könnyebb munkát. Mint minden Xilinx ikonnál itt is működik a *mouseover* funkció, vagyis ha az egeret az ikon fölé visszük, akkor láthatjuk hogy milyen akciót hajt végre.



6.2.2.1.3.7. *ábra. Libraries panel*

A *Libraries* panelen belül a *Source Libraries* mezőben láthatjuk a *work* (munka) könyvtárunkat, amelyen belül a projekthez tartozó összes forrás megjelenítésre kerül. A fájlok beazonosíthatóságuk megkönnyítése érdekében kiterjesztéssel együtt vannak szerepeltetve. Egy fájl megnyitását dupla kattintással tudjuk elérni. A bal felső sarokban látható ikonok segítségével hozzáadhatunk fájlokat a könyvtárunkhoz, ill. törölhetünk belőle. Jobb egérgombbal kattintva a fájlon számos opciót láthatunk még.

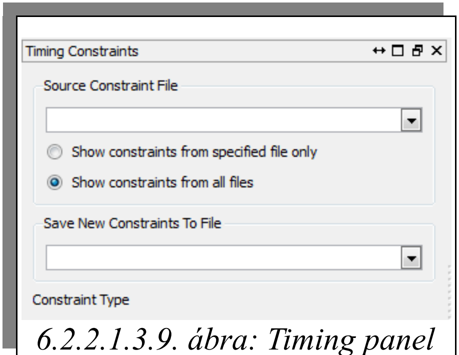


Name	Type	View Association	Source Library
20_ms.vhd	VHDL Module	All	work
binary_to_7segm.sch	Schematic	All	work
DELAY_3S.vhd	VHDL Module	All	work
ERTEK_3bit_0.vhd	VHDL Module	All	work
ERTEK_3bit_1.vhd	VHDL Module	All	work
ERTEK_3bit_2.vhd	VHDL Module	All	work
ERTEK_3bit_3.vhd	VHDL Module	All	work
inic_lcd.vhd	VHDL Module	All	work
mux_4.sch	Schematic	All	work
oszt_1000.vhd	VHDL Module	All	work
oszt_10000000.vhd	VHDL Module	All	work
proba1.sch	Schematic	All	work
proba2.sch	Schematic	All	work
SEVEN_SEGM.ucf	Implementation Constrain...	Implementation	work
SEVEN_SEGM_DECOD....	VHDL Module	All	work
seven_segm_top.sch	Schematic	All	work
taszt.vhd	VHDL Module	All	work

6.2.2.1.3.8. *ábra: Files panel*

A *Files* panelben láthatjuk a projektünkhöz tartozó forrásfájlokat. A különböző oszlopokra kattintva azon tulajdonságok alapján tudjuk az oszlopot abc sorrendbe rendezni. A *Name* oszlop a nevet, a *Type* a típust, a *View Association*[58] mutatja, hogy melyik tervezési modulhoz[59] tartozik a forrás, a *Source Library* a forrás könyvtárt tartalmazza.

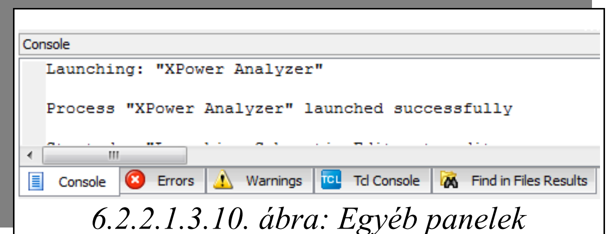
A fájlokon jobb egérgombbal kattintva számos lehetőség ugrik elő (új, hozzáadás, törlés, tulajdonságok, stb.).



6.2.2.1.3.9. *ábra: Timing panel*

A *Timing Constraints* panelben az UCF fájlunkhoz tartozó beállításokat tudjuk megadni. A *Source Constraints file* mezőben válasszuk ki a módosítani kívánt UCF fájlt. A *Save New Constraints To File* mezőben adjuk meg, hogy melyik fájlba mentsük a módosításokat. A *Constraints Editor* részletes leírásáért l. a 74. oldalt.

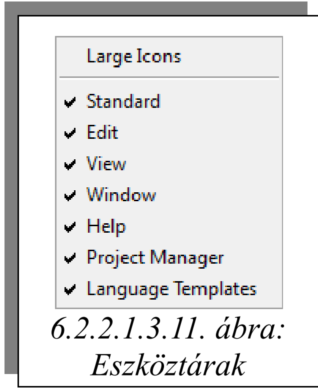
58 Jobb egérgombbal megváltoztatható.
59 Szimuláció, implementáció.



6.2.2.1.3.10. ábra: Egyéb panelek

A fejlesztőkörnyezet alsó felén találjuk az önálló csoportokba nem sorolható paneleket. A *Console* panel mutatja, hogy milyen parancsokat adtunk ki, és ezeknek milyen hatásaik voltak.

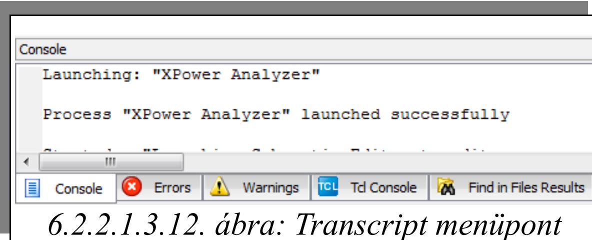
Az *Errors* panel a parancsok végrehajtása során keletkező hibaüzeneteket, míg a *Warnings* panel a veszélyeket tartalmazza. A *Tcl Console*-ban adhatók meg a különböző Tcl parancsok (projekt műveletek, folyamat manipulációk, keresési műveletek, időzítési feladatok). A *Find in Files Results* ablakban láthatjuk az *Edit → Find in Files* menüpont eredményeit.



6.2.2.1.3.11. ábra: Eszköztárak

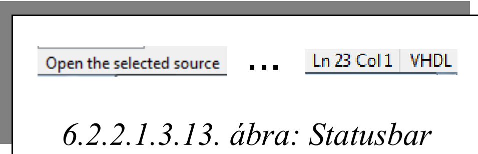
A *View → Toolbars* menüpontban kiválaszthatjuk, hogy mely eszköztárak látszódjanak a fejlesztőkörnyezetben. A *Large Icons* segítségével az ikonokat nagyméretűvé változtathatjuk. Az egyes ikonok jelentésére, hatásaikra és használatukra a 6.2.2.2. fejezetben térünk ki.

Az eszköztárak közül csak azokat érdemes szerepeltetni, amelyeket ténylegesen használunk és hasznosnak ítélünk, mert értékes helyet vesznek el a szerkesztőfelülettől.



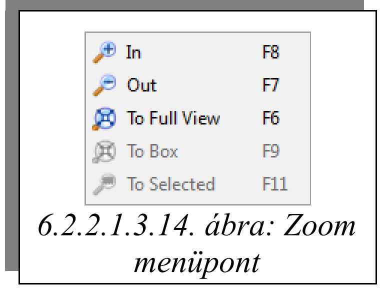
6.2.2.1.3.12. ábra: Transcript menüpont

A *View → Transcript* menüpont segítségével előhozható, ill. eltüntethető az egyéb panelek (*Console, Errors, Warnings, Tcl Console, Find in Files Results*) kategória.



6.2.2.1.3.13. ábra: Statusbar

A *View → Statusbar* menüpont segítségével hozhatjuk elő az állapotsort. Itt láthatjuk legutóbbi műveletünket, ill. a megnyitott fájlhoz tartozó aktuális tulajdonságokat (pl. pozíció).



6.2.2.1.3.14. ábra: Zoom menüpont

A *View → Zoom* segítségével közelíthetünk, ill. távolodhatunk az aktuális fájlban. Az *In* közelítést, az *Out* távolodást, a *To Full View* teljes nézetet tesz lehetővé. Amennyiben területet szeretnénk kijelölni használjuk a *To box* menüpontot, majd jelöljük ki a területet. A *To Selected* segítségével az éppen kijelölt egységre közelíthetünk rá.

6.2.2.1.3.15. ábra: Fájlnév/útvonal mutatása menüpont

Amikor a különböző panelekben a forrásainkat olvassuk, beállíthatjuk a *View → File/Path Display* menüpont segítségével, hogy a hozzájuk tartozó fájlok teljes, vagy relatív útvonallal legyenek megadva, ill. csak a nevük látszódjon.

A Show opció kikapcsolásával a teljes fájl eltüntethető a panelekből.

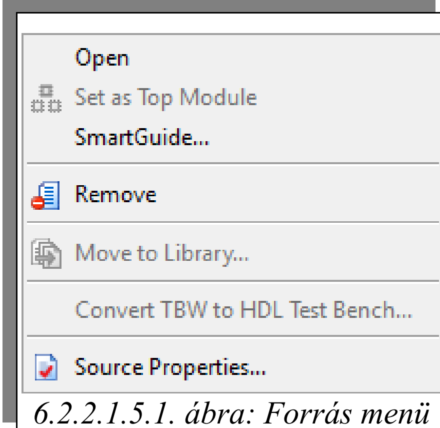
6.2.2.1.4. Project menü

6.2.2.1.4.1. ábra: Projekt menü

New Source...:	Új forrás létrehozása
Add Source...:	Forrás hozzáadása
Add Copy of Source...:	Forrás másolatának hozzáadása[60]
New VHDL Library...:	Új VHDL könyvtár létrehozása
Manual Compile Order:	
Import Custom Compile File List...:	
Disable Hierarchy Reparse:	
Cleanup Project Files...:	Projekt fájlok megtisztítása
Archive...:	Archiválás
Generate Tcl Script...:	
Design Goals & Strategies...:	
Design Summary/Reports:	Tervezés összegző fájl
Design Properties...:	Tervezési tulajdonságok

60 Egy távoli könyvtárból megadott fájlt tudunk a projekt könyvtárba másolni, és hozzáadni a projektünkhöz egy lépésben. (Vannak bizonyos források, amelyek megfelelő működésükhöz további fájlokat is igényelnek, ekkor ezek is a projekt könyvtárba kerülnek.)

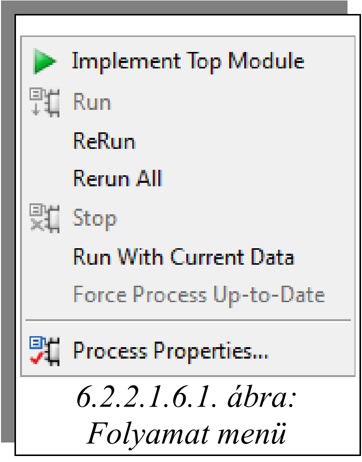
6.2.2.1.5. Source menü



6.2.2.1.5.1. ábra: Forrás menü

Open:	Forrás megnyitása
Set as Top Module:	Kijelölt forrás legfelső forrásként való megjelölése a hierarchiában
SmartGuide...:	SmartGuide[61] használata
Remove:	Forrás törlése a projektből
Move to Library...:	Fájl mozgatása a projekt könyvtárba
Convert TBW to HDL Test Bench...:	TBW teszt fájlok átalakítása HDL típusra[62]
Source Properties...:	Forrás tulajdonságai

6.2.2.1.6. Process menü



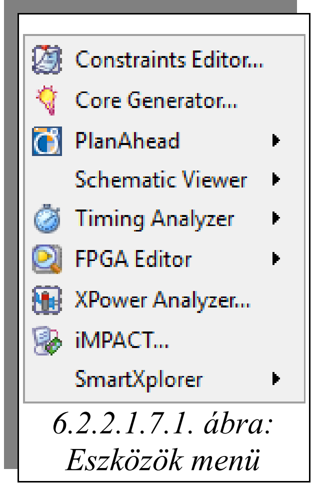
6.2.2.1.6.1. ábra: Folyamat menü

Implement Top Module:	legfelső szintű modul implementálása
Run:	folyamat futtatása
ReRun:	folyamat újra futtatása
Rerun All:	minden folyamat újrafuttatása
Stop:	futtatás megállítása
Run With Current Data:	futtatás a jelenlegi adatokkal
Force Process Up-to-Date:	folyamatfrissítés erőltetése
Process Properties...:	folyamatok tulajdonságai

61 Ez a technológia lehetővé teszi, hogy az előző implementáció eredményeit használjuk fel, lerövidítve ezzel a tervezési időt. Ilyen esetben csak a módosított alkatrészek cserélődnek le. Nagyobb projektekben történő kisebb változtatások esetén célszerű alkalmazni.

62 A Xilinx ISE már nem támogatja a TBW típusú tesztfájlokat, ezért ezeket át kell alakítanunk amennyiben használni szeretnénk őket.

6.2.2.1.7. Tools menü



6.2.2.1.7.1. ábra: Eszközök menü

Constraints Editor: Időzítés és be- kimeneti portok leírását segítő szerkesztő (l. . ábra)

Core Geerator...:

PlanAhead:

Schematic Viewer:

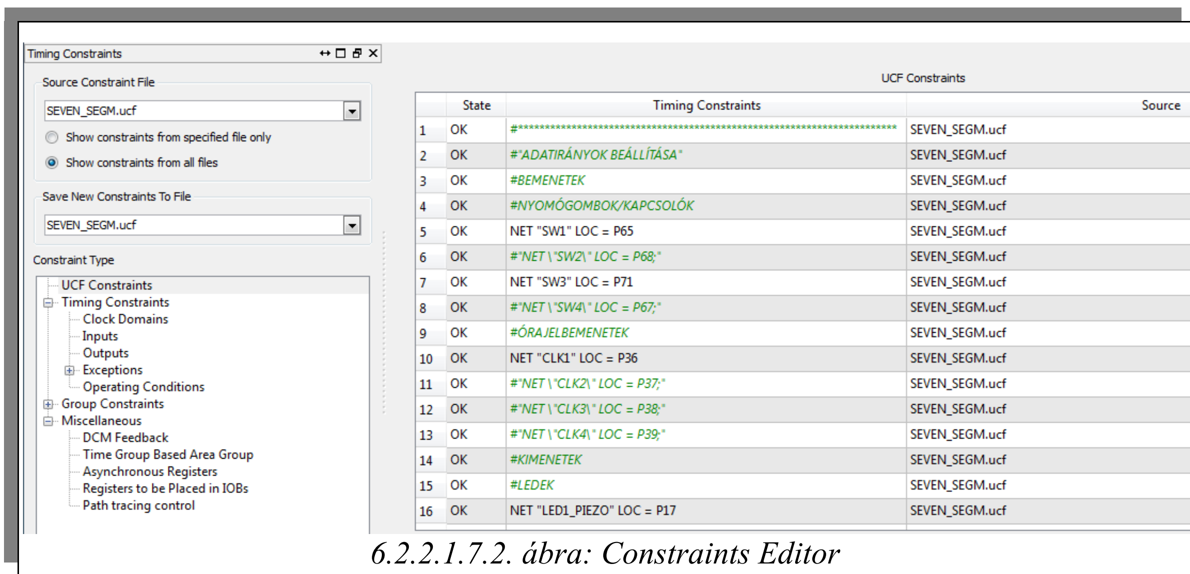
Timing Analyzer:

FPGA Editor:

XPower Analyzer...:

iMPACT...: Leírását l. 6.3. fejezet

SmartXplorer:



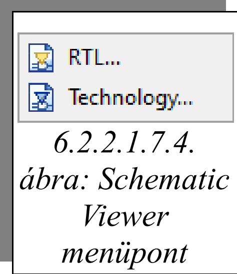
6.2.2.1.7.2. ábra: Constraints Editor

A 6.2.2.1.7.2. ábrán látható Constraints Editor elindításakor automatikusan megnyílik a projektünkhöz tartozó UCF[63] fájl. A fájl tartalmazza az egyes IO-Markerek elhelyezkedését, és a különböző időzítéseket[64].

63 User Constraints File
64 Beállíthatók különböző ofszetek, késleltetések pl. a glitch és a jitter jelenségek kiküszöbölésére.

I/O Pin Planning (PlanAhead) - Pre-Synthesis...
I/O Pin Planning (PlanAhead) - Post-Synthesis...
Floorplan Area/IO/Logic (PlanAhead)...
Analyze Timing / Floorplan Design (PlanAhead)...

6.2.2.1.7.3. ábra: PlanAhead menüpont



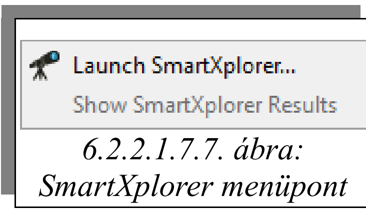
6.2.2.1.7.4. ábra: Schematic Viewer menüpont

Post-Map...
Post-Place & Route...
Post-Fit...

6.2.2.1.7.5. ábra: Timing Analyzer menüpont

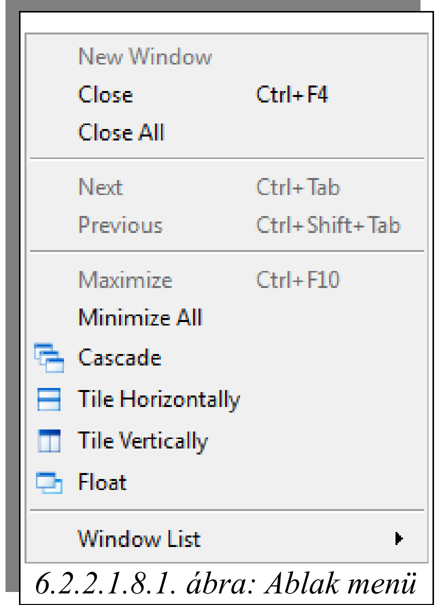
Post-Map...
Post-Place & Route...

6.2.2.1.7.6. ábra: FPGA Editor



6.2.2.1.7.7. ábra: SmartXplorer menüpont

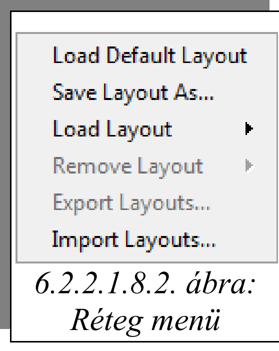
6.2.2.1.8. Window menü



6.2.2.1.8.1. ábra: Ablak menü

New Window:	Új ablak
Close:	Aktuális ablak bezárása
Close All:	Összes ablak bezárása
Next:	Következő ablakra ugrás
Previous:	Előző ablakra ugrás
Maximize:	Aktuális ablak maximalizálása
Minimize All:	Minden ablak minimalizálása
Cascade:	Kaszkád (egymáson történő) elrendezés
Tile Horizontally:	Vízszintes elrendezés
Tile Vertically:	Függőleges elrendezés
Float:	Lebegő elrendezés
Windows List:	Megnyitott fájlok ablakainak listája

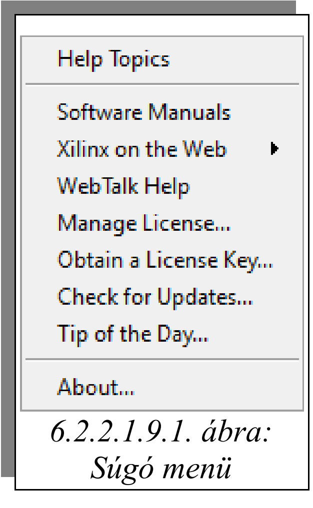
Layout[65] menü



6.2.2.1.8.2. ábra: Réteg menü

Load Default Layout:	Alapértelmezett réteg betöltése
Save Layout As...:	Réteg mentése más néven
Load Layout:	Réteg betöltése
Remove Layout:	Réteg törlése
Export Layout...:	Réteg exportálása
Import Layouts...:	Réteg importálása

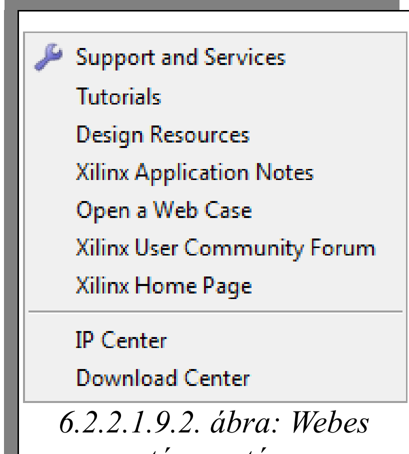
6.2.2.1.9. Help menü



6.2.2.1.9.1. ábra: Súgó menü

Help Topics:	Súgó témakörök
Software Manuals:	Különböző leírások a Xilinx honlapján
Xilinx on the Web:	Xilinx a weben (l. 6.2.2.1.9.2. ábra)
WebTalk Help:	Xilinx internetes jelentésküldő súgója
Manage License...:	Licenc menedzser megnyitása
Obtain a License Key...:	Licenc beállítása
Check for Update...:	Frissítés keresése
Tip of the Day...:	A nap tippje (l. 6.2.2.2. ábra)
About...:	Névjegy

65 A Xilinx rétegeknek nevezi a különböző ablakelrendezéseket, szerkesztőfelület kinézeteket, amiket elmenthetünk, majd később előhívhatunk. Részletesebben l. . fejezet.



6.2.2.1.9.2. ábra: Webes támogatás

Support and Services: Támogatás és szolgáltatások
Tutorials: Oktató példák
Design Resources: Tervezési erőforrások (oktatóvideók)
Xilinx Application Notes: Alkalmazási útmutatók
Open a Web Case: Internetes információk gyűjteménye
Xilinx User Community Forum: Netes forum megnyitása
Xilinx Home Page: Xilinx kezdőoldal
IP Center: IP[66] központ megnyitása
Download Center: Letöltési központ megnyitása

6.2.2.2. A leggyakrabban használt ikonok

6.2.3. Új projekt létrehozása

6.3. Impact bemutatása

6.4. A fejlesztés bemutatása egy komplex példán keresztül

A következő példa egy
Edit → Language templates
Edit → preferences

66 Intellectual Property (Szellemi tulajdon) ezek kulcsfontosságú építőelemei a Xilinx tervezőszoftverének. Részletekért l. fejezet.

7. SCH alapú fejlesztés

Egy digitális technikában jártas személy egy hardver leírását legegyszerűbben kapcsolási rajz alapú szerkesztővel tudja megvalósítani.

8. VHDL alapfogalmak

A műszaki élet fejlődése és az egyre bonyolultabb rendszerek tervezése iránti igény miatt szükségessé vált szabványos hardverleíró nyelvek kidolgozása. Az iparban használt nyelvek:

- VHDL
- Verilog
- System C

A Xilinx szoftvere támogatja mind a VHDL, mind a Verilog nyelven történő fejlesztés. Miután a VHDL nyelvhez több example, angol nyelvű szakirodalom s template található, valamint a fejlesztőcsapat úgy vélte, hogy szintaktikája „barátságosabb”, úgy döntöttünk, hogy ezt alkalmazzuk fejlesztéseinknél[67]. A nyelvi elemek használatát általában az angol nyelvű súgóban elérhető példákon keresztül mutatjuk be, kiegészítve magyar nyelvű megjegyzésekkel. Néhány esetben saját kódsorozatot hozunk példaként – ezek azonban önálló projektként nem állják meg a helyüket. Amennyiben az olvasó ilyen típusú mintapéldákat keres lapozzon a 9. fejezethez. Miután az említett fejezetben számos gyakorlati példát találhatunk, ezért itt csak a szintaktikai szabályokra igyekszünk kitérni.

Számos olyan kifejezés és megoldás van a VHDL nyelvben amiket a Xilinx nem támogat, mi a leírásban csak a támogatott nyelvi elemekre térünk ki részletesen – erre a későbbiekben néhány esetben külön fel is hívjuk a figyelmet.

8.1. Röviden a VHDL nyelvről

A VHDL mozaikszó az alábbi jelentéssel bír: VHSIC[68] Hardware Description Language, vagyis nagy sebességű integrált áramkörök hardver leíró nyelve. A VHDL-t az USA Védelmi Minisztériumának megbízásából fejlesztették ki ASIC áramkörök viselkedésének leírására, szimulációs környezetek megvalósítására. A nyelv több változtatáson esett át miközben az IEEE[69] bizottsága szabványosította. Az alapszabvány a IEEE Std 1076, amelynek az alábbi négy típusa van használatban:

- VHDL-87 (1076-1987)
- VHDL-93 (1076-1993)
- VHDL-02 (1076-2002)
- VHDL-08 (1076-2008)

A Xilinx fejlesztőkörnyezete jelenleg a 87-es és 93-as szabványt támogatja. Mi a továbbiakban a 93-assal dolgozunk majd. A szabványok között számos eltérés van, azonban lefelé kompatibilis (vagyis azok a nyelvi elemek, amiket a 87-es szabvány tartalmaz használhatóak a 93-asban is).

---

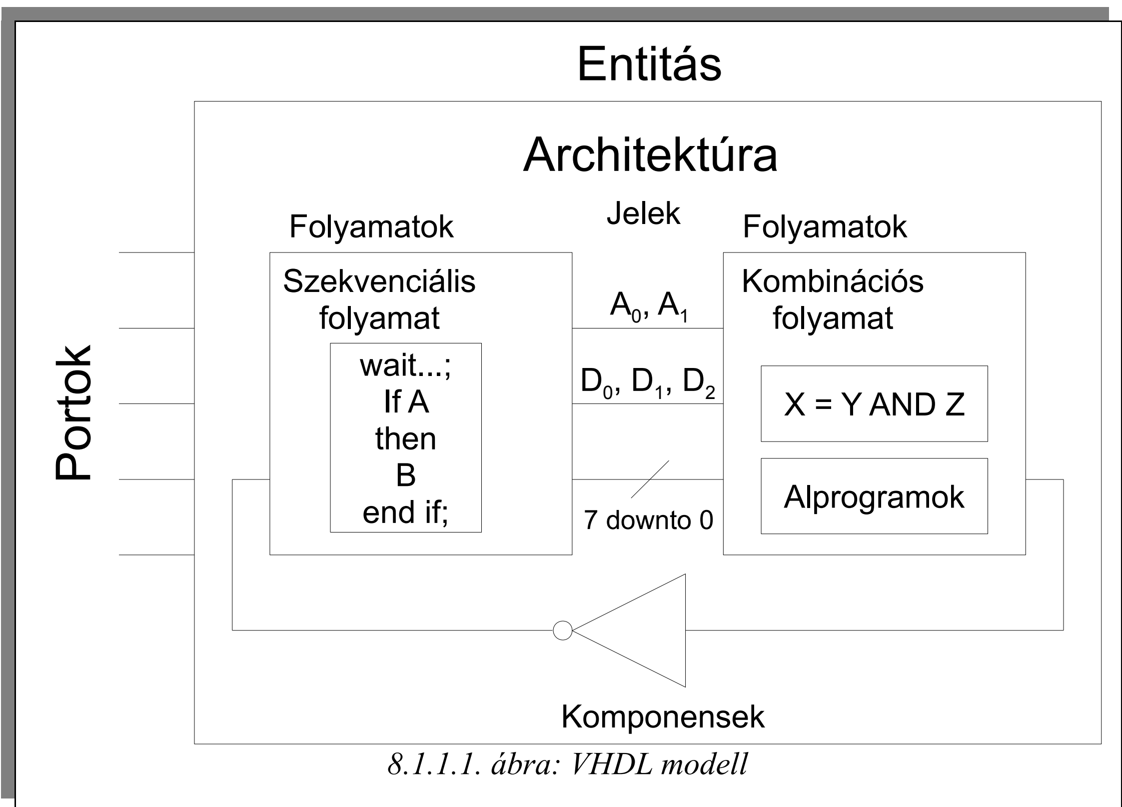
67 A VHDL nyelv leírásánál a Xilinx súgójában található angol nyelvű *VHDL Reference Guide* mellett felhasználtunk egy VHDL kézikönyvet: http://www.csee.umbc.edu/portal/help/VHDL/VHDL-Handbook.pdf

68 VHSIC: Very High Speed Integrated Circuits – nagysebességű integrált áramkörök.

69 Ejtsd: ájtriplö. IEEE: Institute of Electrical and Electronics Engineers.

8.1.1. Egy VHDL nyelven megírt alkatrész felépítése

A 8.1.1.1. ábrán láthatjuk, hogy hogyan néz ki egy VHDL-el kódolt entitás. Több jól elkülönülő építőelemet tudunk megfigyelni. Felbonthatjuk a leírást az architektúra és a külvilággal kapcsolatot tartó portok meghatározására. A portok megadhatóak különálló lábanként, vagy buszos csatlakozásként is. Az architektúra tartalmazhat komponenseket, ezek előre megírt VHDL alkotóelemek. Emellett lehetőségünk van sorrendi és kombinációs folyamatok leírására is. Az alkotóelemek között a jelek teremtenek kapcsolatot és így épül fel egy komplett alkatrész.



8.1.1.1. ábra: VHDL modell

Amennyiben VHDL-el szeretnénk egy feladatot megoldani, mindig a fenti ábrát tartsuk szemünk előtt! Alkatrészekben és ne programrészekben, rutinokban gondolkozzunk! Célszerű főleg a kezdeti fejlesztéseknél alapkomponensekre, blokkokra bontani a felépítendő hardvert, majd a részegységeket külön lekódolni és kipróbálni. Az egyes részek összekötésének biztosításához mi schematic alapú fájlt javaslunk, mert úgy gondoljuk, hogy „mérnök ember rajzból ért”.

8.2. Alapvető nyelvi elemek

Elsőként érdemes megismerkedni a VHDL nyelv alapvető szintaktikájával, nyelvi szabályaival.

8.2.1. Lefoglalt kulcsszavak

Minden programozási nyelvben vannak ún. lefoglalt kulcsszavak, amelyek a fordító számára speciális jelentéssel bírnak, így a programozó ezeket nem alkalmazhatja pl. változói elnevezésére, vagy címkék megadására. A VHDL nyelv az alábbi táblázatban szereplő kulcsszavakat tartalmazza.

abs	access	after	alias	all	and
architecture	array	assert	attribute	begin	block
body	buffer	bus	case	component	configuration
constant	disconnect	downto	else	elsif	end
entity	exit	file	for	function	generate
generic	**group**	guarded	if	**impure**	in
inertial	inout	is	label	library	linkage
literal	loop	map	mod	nand	new
next	nor	not	null	of	on
open	or	others	out	package	port
postponed	procedure	process	**pure**	range	record
register	**reject**	rem	**report**	return	**rol**
ror	select	severity	signal	**shared**	**sla**
sll	**sra**	**srl**	subtype	then	to
transport	type	**unaffected**	units	until	use
variable	wait	when	while	with	**xnor**
xo					

A félkövérrel szedett a VHDL-87-ben használható, de a VHDL-93-ban már lefoglalt kulcsszó.

8.2.2. Karakterkészlet

A VHDL-87 szabvány a 0÷127-ig terjedő ASCII karaktereket tartalmazza, míg a VHDL-93-ban a kibővített ASCII karakterek 0÷255-ig használhatóak. A karaktereket az alábbi csoportokba tudjuk sorolni:

- Kisbetűk (abcdefg)
- Nagybetűk (HIJKLMNOP)
- Számok (12345678)
- Különleges karakterek (?;.%)
- Szóköz
- Speciális formázó karakterek (CF+LR)

8.2.3. Elválasztók

A VHDL nyelvben a különböző elemek elválasztására a SPACE[70] karaktert használhatjuk.

8.2.4. Megjegyzések

A fordító megjegyzésnek tekinti az adott sorban két kötőjel (--) után lévő karaktereket. Egyes sorokat, vagy teljes kijelöléseket tudunk megjegyzéssé, vagy éppen fordítva futtatható kóddá alakítani az *Edit → Comment*, *Edit → Uncomment* menüpont segítségével (l. 6.2.2.1.2.1. ábra).

8.2.5. Konstansok

8.2.5.1. Numerikus konstansok

Numerikus konstansokat az alábbi formátumokban tudunk megadni:

- 4_000 (4000; az _ karaktert alkalmazhatjuk a jobb olvashatóság érdekében)
- 2E4 (20000; az E betű után lévő szám 10 adott hatványát jelenti)
- 16#FF# (255; a hashmark közé tett számok abban a formátumban értendőek, amit a # előtt megadtunk. Használható: 2; 8; 16)

8.2.5.2. Felsorolás típusú konstansok

Az enumerator típusú konstansok a következő formában használhatóak:

- *BIT* (bit): '0'; '1'
- *BOOLEAN* (logikai) true; false
- *CHARACTER* (karakter): 'A'; 'b'

Vagyis a felsorolás típus vagy egy karakter, vagy egy azonosító.

70 Alkalmazható helyette a Tabulátor is.

8.2.5.3. Sztring típusú konstansok

A sztring konstansok egydimenziós tömbök, amikben általában szöveges információt tudunk tárolni.

8.2.5.4. Bitsztring típusú konstansok

A VHDL-87 még csak a a *BIT_VECTOR* bitsztringet ismerte, míg a VHDL-93 már bevezette a *STD_LOGIC_VECTOR*[71] típust. A bitsztring típusok is egydimenziós tömbök, azonban ezeket nem karaktersorozat, hanem bitsorozat tárolására használjuk. Lehetséges formái:

- **B** – *Binary* (*0; 1*)
- **O** – *Octal* (*1÷7*): oktális nyolcas számrendszerű szám, három helyi értékű (pl. 4_o = *„101”*).
- **H** – *Hexadecimal (1÷9; A÷F*): hexadecimális, 16-os számrendszerű szám, négy helyi értékű (pl. A_h = *„1010”*).

  A jobb olvashatóság érdekében elválasztásra alkalmazható az _ karakter. Pl. 1001_1101

8.2.5.5. Null konstans

Mutatók (pointerek) esetén használatos, és azt jelzi, hogy a pointer üres, vagyis nem mutat sehova.

8.3. Adattípusok

A VHDL nagyon erős szintaktikai szabályokkal bír és erősen típusos nyelv. Pl. egy négy- és egy nyolcbites számot nem tudunk csak úgy összeadni, mint mondjuk egy C nyelvben. Műveleteket csak azonos adathosszon tudunk elvégezni – ezért nagyon fontos változóink hossza[72]. Amennyiben egy adott hosszúságú változót szeretnénk használni mindig meg kell mondanunk a pontos hosszát ha előre definiált típust alkalmazunk, vagy nekünk kell definiálni egy típust. A lenti részben saját típus definiálására láthatunk példát.

```
type SHORT is array(7 downto 0) of BIT;        -- SHORT: 8 bites adattípus
variable OP1: SHORT;                           -- OP1: 8 bites változó
.
.
.
OP1(0) := 1;                                   -- bitenkénti értékadás
.
.
.
OP1 := ”11110000”;                             -- teljes szám írása
.
.
.
OP1 := (1 => '1', 4 => '1', others => 0);      -- speciális értékadás
                                               --  1. és 4. bit egyes, a többi nulla
```

71 Szabványos logikai vektor

72 Különböző trükkökkel és konverziókkal természetesen megoldható különböző adathosszok kezelése is, erre a későbbiekben majd láthatunk példákat.

8.3.1. Felsorolás típus

Felsorolás[73] adattípust nekünk kell definiálnunk felsorolás konstansokból (azonosító, vagy egy karakter). Alább láthatjuk ennek egy lehetséges módját.

```
type COLOR is (RED, GREEN, BLUE);          -- COLOR: 3 (konstans) elemű felsorolás típus
type LOGIC_HZ is ('0','1','Z');            -- LOGIC_HZ 3 (konstans) elemű felsorolás típus
variable SZIN1: COLOR;                     -- SZIN1: LOGIC_HZ típusú változó
signal JEL1:  LOGIC_HZ;                    -- JEL1: LOGIC_HZ típusú jel
.
.                                          -- egyéb deklarálások
.
SZIN1: RED;                                -- SZIN1 értéke: RED
JEL1 <= '0' ;                              -- JEL1 értéke: 0
```

8.3.1.1. Felsorolás típus túltöltése

73 Enumeration

8.3.1.2. Felsorolás típus kódolása

A felsorolás típus értékei alapértelmezésben sorrendben 0-tól kezdődnek. Az alábbiakban a kódolásra láthatunk példát.

```
type COLOR is (RED, GREEN, BLUE);         -- RED: 00; GREEN: 01; BLUE: 10
```

A Xilinx fejlesztőkörnyezete lehetőséget nyújt az előre definiált kódolás felülírására[74]. Az alábbiakban erre láthatunk egy példát.

```
attribute ENUM_ENCODING: STRING;    -- tulajdonság definiálása75
type COLOR is (RED, GREEN, BLUE);     -- RED: 00; GREEN: 01; BLUE: 10 (alapértelmezett)
attribute ENUM_ENCODING of COLOR: type is "10 01 00";    -- RED: 00;
                                                          -- GREEN: 01;
                                                          -- BLUE: 10
```

Az ENUM_ENCODING lehetséges értékei:

- 0: '0' értékű bit
- 1: '1' értékű bit
- D: Don't-care (lényegtelen)
- U: Unknow (ismeretlen)
- Z: High Impedance (nagyimpedanciás állapot)

8.3.2. Egész típus

Az egész típus minimális értéke $-2^{31}-1$, míg maximális értéke $2^{31}-1$. Csoportosításra elválasztásra használható az _ karakter. (Pl. -2_147_483_647; 2_147_483_647). A típus definíciója alább látható:

type *type_name* is range *integer range*;

A type is range; kötelezően alkalmazandó szintaxis. A *type_name* a programozó által megadott típusnév, míg az *integer range* az egész típus tartománya (a minimum és a maximum közé a to szót kell beszúrnunk).

```
type kitoltes is range 0 to 100;              -- a kitoltes egy 0-tól százig terjedő egész típus lesz
```

74 Más VHDL alapú fordítók esetén általában csak a szabványos alapértelmezett kódolásra van lehetőségünk.
75 Az ENUM_ENCODING tulajdonság csak a Xilinx fejlesztőkörnyezete esetén értelmezhető.

8.3.3. Tömb típus

type *type_name* is array (*array_range*) of *predefined_type*;

A type is array () of; kötelezően alkalmazandó nyelvi elem. A *type_name* mezőbe írhatjuk az új típus nevét. Az *array_range* a tömb elemeinek számát határozza meg. A *predefined_type* egy előre definiált típus, a tömb egyes eleminek típusát határozza meg. A tömb adott elemére a mögötte lévő zárójelbe tett értékkel tudunk hivatkozni (ez egy változó, vagy jel is lehet).

```
type SHORT_ARRAY is array (7 downto 0) of BIT;                -- 1 dimenziós tömb
type SHORT2D_ARRAY is array (7 downto 0) of SHORT_ARRAY ;  -- 2 dimenziós tömb
type STRING_100 is array (0 to 100) of CHARACTER;             -- sztring
signal szoveg1: STRING_100;                                   -- szoveg1 egy 100 elemű sztring
.
.
.
szoveg1(0) <= 'A';                                            -- tömb adott elemének értékadás
```

A következő szintaxis olyan tömbtípusoknál alkalmazható, ahol nem szeretnénk korlátozni az elemszámot:

type *array_type_name* is array (*range_type_name* range <>) of *element_type_name;*

Az *array_type_name* a tömb típusának neve. A *range_type_name* a tömb elemeinek tartománytípusa. Az *element_type_name* a tömb elemeinek típusa.

```
type INTEGER_ARRAY is array (NATURAL range <>) of INTEGER;    -- integer tömbtípus
.
.
.
variable tomb1: INTEGER_ARRAY(100 to 0);                      -- 100 elemű egész tömb
```

Tömb típus használatára l. 9.1.2.7. és fejezet.

8.3.4. Record típus

A record típus összetartozó alaptípusokból felépülő összetett adattípus. Használata hasonlít a tömbökéhez, azonban az elemei különböző típusok is lehetnek. Az alábbi példa egy RGB LED színösszeállításához használ fel egy record típusú signalt.

```
type RGB_8bit is record                                    -- record típus definíció
RED: NATURAL range 0 to 100;
GREEN: NATURAL range 0 to 100;
BLUE: NATURAL range 0 to 100;
end record RGB_8bit
.
.
.
signal SZIN RGB_8bit;                           -- 3 elemből álló rekord
SZIN.RED <= 16#F0#;                             --  értékadás a rekord 1 elemének
.
.
.
SZIN <= (RED => 50, GREEN => 20, BLUE => 100);  --  értékadás az összes elemnek
                                                -- a nevek használata nem kötelező
                                                -- pl. SZIN <= (50,20,100);
.
.
.
SZIN <= (”11110000”,1);                         --  record adott sorszámú elemének
                                                -- feltöltése értékkel
```

A fenti példában nem használjuk ki a record típus előnyeit, ezért – bár a programrészlet működőképes – erre a feladatra célszerűbb tömböt használni. Az alábbiakban olyan esetet mutatunk be ahol ez már nem lenne lehetséges.

```
type DATUM is record
EV is positive range 2000 to 2100;          -- évszám
HONAP is string(5 to 10);                   --  hónap megadása karakterekkel
NAP is positive range 1 to 31;              -- nap
HET_NAPJA  is string(4 to 9);               -- a hét aktuális napja karakterekkel
end record DATUM;
.
.
.
DATUM <= (2013, aprilis, 13, szombat);      -- record feltöltése
```

Record típusok használatát l. . fejezet.

8.3.5. Nem támogatott VHDL típusok

Vannak olyan szabványos VHDL adattípusok, amelyeket a Xilinx fejlesztőkörnyezete nem támogat. Ebben a fejezetben ezeket a típusokat soroljuk fel:

- Fizikai (*Physical*) típus
- Lebegőpontos (*Floating-Point*) típus
- Hozzáférés (*Access*) típus
- Fájl (*File*) típus

8.3.6. Előre definiált típusok

Az előre definiált típusok rövid összefoglalóját az alábbi táblázat tartalmazza.

Típus	Lehetséges értékei	Alkalmazható operátorok
INTEGER	-2147483647 ÷ 2147483647	ABS ** - * / MOD REM + - (sign) + - = /= < <= > >=
BIT	'0'; '1'	NOT = /= < <= > >= AND NAND OR NOR XOR XNOR*
BOOLEAN	TRUE; FALSE	NOT = /= < <= > >= AND NAND OR NOR XOR XNOR*
BIT_VECTOR	Korlátlan bittömb	NOT & SLL* SRL* SLA* SRA* ROL* ROR* = /= < <= > >= AND NAND OR NOR XOR XNOR*
CHARACTER	128 karakter (VHDL-87) [ISO 646-1983] 256 karakter (VHDL-93) [ISO 8859-1 : 1987(E)]	
STRING		
Altípusok[76]		
NATURAL		l. integer
POSITIVE		l. integer

76 Subtype

Az előzőekben látott adattípus megadások ismerete után a következő ún. STANDARD csomag már könnyen elemezhető.

```
package STANDARD is
type BOOLEAN is (FALSE, TRUE);
type BIT is (’0’, ’1’);
type CHARACTER is (
NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
BS, HT, LF, VT, FF, CR, SO, SI,
DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
CAN, EM, SUB, ESC, FSP, GSP, RSP, USP,
’ ’, ’!’, ’"’, ’#’, ’$’, ’%’, ’&’, ’’’,
’(’, ’)’, ’*’, ’+’, ’,’, ’-’, ’.’, ’/’,
’0’, ’1’, ’2’, ’3’, ’4’, ’5’, ’6’, ’7’,
’8’, ’9’, ’:’, ’;’, ’<’, ’=’, ’>’, ’?’,
’@’, ’A’, ’B’, ’C’, ’D’, ’E’, ’F’, ’G’,
’H’, ’I’, ’J’, ’K’, ’L’, ’M’, ’N’, ’O’,
’P’, ’Q’, ’R’, ’S’, ’T’, ’U’, ’V’, ’W’,
’X’, ’Y’, ’Z’, ’[’, ’\’, ’]’, ’^’, ’_’,
’‘’, ’a’, ’b’, ’c’, ’d’, ’e’, ’f’, ’g’,
’h’, ’i’, ’j’, ’k’, ’l’, ’m’, ’n’, ’o’,
’p’, ’q’, ’r’, ’s’, ’t’, ’u’, ’v’, ’w’,
’x’, ’y’, ’z’, ’{’, ’|’, ’}’, ’~’, DEL);
type INTEGER is range -2147483647 to 2147483647;
subtype NATURAL is INTEGER range 0 to 2147483647;
subtype POSITIVE is INTEGER range 1 to 2147483647;
type STRING is array (POSITIVE range <>)
of CHARACTER;
type BIT_VECTOR is array (NATURAL range <>)
of BIT;
end STANDARD;
```

A fent látható típusok alapértelmezetten meg vannak adva. Azok használata minden további nélkül elfogadott egy VHDL kódban.

8.3.7. Downto és To használata

Deklarációknál – típusok, altípusok, jelek és változók megadásakor, tömbbe, ill. rekordba rendezésekor – használatos kulcsszavak a *downto,* s a *to*. Egyáltalán nem mindegy, mikor melyiket alkalmazzuk. Alapvetően elmondható, hogy amennyiben bitenkénti hozzáférést szeretnénk alkalmazni akkor a *downto* használata, míg minden más esetben a *to* használata javasolt. Az alábbi kódrészletben a downto használatára láthatunk példát:

```
signal BYTE1: unsigned(7 downto 0):="10000000"; -- balról-jobbra olvasva MSB...LSB
```

Láthatjuk, hogy egy binárisan megadott szám (bitsztring) esetén nagyon fontos, hogy a *downto* kulcsszót alkalmazzuk, különben az MSB és az LSB általánosan elfogadott helyiértéke felcserélődik. Egy integer tömb megadásánál[77] ezzel szemben a *to* kulcsszó használata az ajánlott, hiszen balról-jobbra olvasva a tömb elemeit a 0. elemtől haladunk a legmagasabb számú elemig.

77 És minden más esetben is.

8.3.8. Típuskonverziók

A VHDL nyelvben egyes operátorok, utasítások, függvények csak adott típusokat támogatnak, a művelet csak meghatározott típus esetén hajtható végre. A fejlesztéshez szükségünk van a különböző típusok és adathosszok közötti átjárásra. Ezt teszi lehetővé a típuskonverziók használata. Az alábbi kódban láthatunk néhány típuskonverziót.

Egyéb konverziós függvények használata

Az IEEE.NUMERIC.STD és az IEEE.ARITH

8.3.9. Értékadás

Változóinknak és jeleinknek, be- és kimeneteinknek a folyamatok során értékeket kell tudnunk adni. Ezt a VHDL nyelv segítségével igencsak könnyen el tudjuk végezni. Értéket adhatunk egy-egy bitnek, a teljes bináris számnak, vagy a bináris szám egy szeletének. A következő kódrészlet értékadásra nyújt példákat:

```
S1 <= ”1000_0111_1111_1111_1000_0011_1011_1001”; -- értékadás 32 bites jelnek binárisan
S1 <= X”FA13D25A”; -- értékadás 32 bites jelnek hexadecimálisan
S1 <= (1 => '1', 2 => '1', others => '0'); -- 1. és 2. bit egyes, a többi nulla
S1(15 downto 0) <= S2; -- S1 alsó 16 bitje S2 16 bites jel
S1(31 downto 16) <= S3; -- S1 felső 16 bitje S3 16 bites jel
```

8.4. Változók és jelek

A változók és jelek használatára nagyon kell ügyelnünk a VHDL nyelv alkalmazásakor. Egy jelet könnyen el tudunk képzelni, elég ha egy adott áramkör egyik netjére gondolunk. A jelnek tehát van fizikai megfelelője, míg a változó egy absztrakció, amelyet a programozói kreativitás tesz szükségessé. A jel egy folyamat során egyszer kap értéket, míg a változó többször is értéket kaphat. Fontos ügyelnünk rá, hogy mind a változó, mind a jel esetében csak egy folyamat lehet a meghajtó. Vagyis nem tehetjük meg, hogy két külön folyamat is ugyanazt a jelet, vagy változót írja, hiszen ezen folyamatok egymással párhuzamosan futnak le, és emiatt megtörténhetne, hogy egy „vezetékre” egyszerre szeretnénk logikai nullát és egyest is kötni. Természetesen az olvasással ilyen gondunk nincsen. A jeleket és változókat máshol kell deklarálnunk és ezáltal a hatókörük is eltérő lesz. A jel a teljes architektúrára kiterjed, míg a változó csak az adott blokkon *(process, for loop*, stb.) belül érvényes[78].

A hatókör miatt a jeleket a *signal* kulcsszó használatával az architektúra leírás *begin* kulcsszava előtt kell megadnunk. Itt nem szerepelhet változó deklarálás, mert a változókat csak az adott blokkokon belül adhatjuk meg, amelyekben használni kívánjuk. A jeleknek és változóknak a *:=* segítségével adhatunk kezdőértéket. Értékadásra a jeleknél a <= míg a változóknál a *:=* karakterek használatosak. Jelek megadása:

signal *signal_name: type* := *value*;

A *signal* kötelező nyelvi elem után adhatjuk meg a jel nevét, ügyeljünk rá, hogy ne használjunk különleges karaktert, ill. lefoglalt kulcsszót. A *type* helyére írjuk be a jelünk típusát, pl. *STD_LOGIC*, *unsigned(7 downto 0)*, stb. A *:=* opcionális, amennyiben alkalmazzuk, utána meg kell adnunk a jel kezdőértékét.

variable *variable_name: type* := *value*;

A *variable* szintaktikai elem után adhatjuk meg változónk nevét. A *type* helyére írjuk be a változó típusát, pl. *integer*, *unsigned(7 downto 0)*, stb. A *:=* opcionális, amennyiben alkalmazzuk, utána meg kell adnunk a változó kezdőértékét.

A következőkben rengeteg példát láthatunk jelek és változók megadására és használatára. További gyakorlati példákért l. 9. fejezet.

Miután a jelnek van fizikai tartalma, és így könnyebben megfogható, ezért döntően ezek használatát javasoljuk a fejlesztések során. Megjegyezzük azonban, hogy pl. függvények és ciklusok esetén a változók használata – a korlátozott hatókör miatt – elengedhetetlen.

78 Kivételt képez ez alól a megosztott változó, amelyet ugyanott kell megadnunk, ahol a jeleket és hatókörük kiterjed a teljes architektúrára, azonban használatuk megegyezik a többi változóéval.

8.5. Entitások

Az entitások a be- és kimeneti portok meghatározását szolgálják. Az entitások megadása a következőképpen történhet:

entity *entity_name* is generic (*generic_declarations*);
port (*port_declarations*);
end *entity_name*;

Az *entity_name* helyére az entitásunk nevét kell írnunk (ügyeljünk rá, hogy ne használjuk a lefoglalt kulcsszavakat). A *generic_declarations* mezőbe kerül az adott példányra jellemző érték, amennyiben van ilyen. A *port_declarations* helyre kell a be- és kimeneti portokat megadni – több port esetén a ; karakter szolgál elválasztásra. Az alábbi kódrészletek néhány entitás megadását tartalmazzák.

```
entity SEVEN_SEGM_DECOD is                              -- Hétszegmenses dekóder megadása
   Port ( BCD : in  STD_LOGIC_VECTOR(3 downto 0);       -- 4 bites BCD bemenet
        SEGM : OUT STD_LOGIC_VECTOR(7 downto 0));       -- 8 bites kimenet
end SEVEN_SEGM_DECOD;
```

```
entity oszt_1000 is                      -- frekvenciaosztó megadása
   Port ( CLK : in  STD_LOGIC;           -- CLK: clock bemenet
        DIV_1000 : out  bit);            -- DIV_1000: a bemeneti jel osztott értéke
end oszt_1000;
```

```
entity LOGIC_1 is                        -- egyszerű kombinációs hálózat megadása
Port ( A : in  STD_LOGIC;
B : in  STD_LOGIC;
AND_AB : out  STD_LOGIC;
OR_AB : out  STD_LOGIC;
INV_A : out  STD_LOGIC;
INV_B : out  STD_LOGIC;
XOR_AB : out  STD_LOGIC);
end LOGIC_1;
```

```
entity rotate_right is                   -- bitforgató alkatrész megadása
Port ( CLK : in  std_logic;
EN : in  std_logic;
Q : out  unsigned (7 downto 0));
end rotate_right;
```

Az alább látható kódsorozat egy két bemenetű N-bites komparátor megadását mutatja be:

```
entity COMP is                           -- COMP néven egy
generic(N: INTEGER := 8);                -- alapértelmezetten 8 bites
port(X, Y: in BIT_VECTOR(0 to N-1);      -- két bemenetű komparátor
EQUAL: out BOOLEAN);                     -- egy logikai kimenettel van megadva
end COMP;
```

8.6. Architektúra leírása

Az architektúra leírásában helyezkedik el az eszköz (alkatrész) viselkedésének leírása. Itt adhatjuk meg, hogy entitásunk valójában milyen – akár kombinációs, akár sorrendi – feladatokat hajtson végre.

architecture ***architecture_name*** **of** ***entity_name*** **is**
block_declarative_item
begin
concurrent_statement
end ***architecture_name*****;**

Az *architecture_name* mezőbe az architektúra neve (pl. leiras; behavioral, viselkedes), míg az *entity_name* mezőbe annak az entitásnak a neve kerül, aminek az architektúráját meg szeretnénk adni. A block_declarative_item helyre kerülhet a típus, altípus, konstans, jel deklaráció. A concurrent_statement mezőbe kerülnek az egyidejű utasítások[79].

A következő kódrészlet egy futófény program alapját képező bitforgatást végrehajtó alkatrész leírását tartalmazza:

```
entity rotate_right_2 is                        -- bitforgatás
  Port ( CLK : in  std_logic;                   -- CLK: clock bemenete, a forgatás ütemezése
    EN : in  std_logic;                         -- EN: forgatás engedélyezése
    RL : in  std_logic;                         -- RL: irány meghatározása
    Q : out  unsigned (7 downto 0));            -- Q: 8 bites kimenet
end rotate_right_2;                             -- párhuzamos beírásra nincs lehetőség

architecture Behavioral of rotate_right_2 is    -- architektúra viselkedésének leírása
  signal OUT_TEMP: unsigned(7 downto 0):="10000000";
                                                -- kezdőérték a kimeneti bufferben: 10000000
begin
  process                                       -- sorrendi folyamat
  begin
    wait until CLK'event and CLK = '0';         -- várakozás órajel lefutó élére
    if EN = '1' then                            -- engedélyezésre
      if RL = '1' then                          -- RL függvényében
      OUT_TEMP <= OUT_TEMP ror 1;               -- jobbra forgat egyet
      else
      OUT_TEMP <= OUT_TEMP rol 1;               -- balra forgat egyet
      end if;
    end if;
  end process;                                  -- sorrendi folyamat vége
Q <= OUT_TEMP;                                  -- kimeneti buffer kimenetre töltése
                                                -- egyidejű folyamat
end Behavioral;                                 -- architektúra leírás vége
```

79 Az ide beírt parancsok egyszerre, egymással párhuzamosan kerülnek végrehajtásra.

8.7. Komponensek megadása

Ahhoz, hogy az egyes VHDL entitásokat a későbbiekben fel tudjuk használni komponensként kell rájuk hivatkoznunk. Ez valójában megegyezik egy kapcsolási rajz alapú szerkesztés esetén az egyes schematic szimbólumok összekötésével.

component ***identifier***
generic(*generic_declarations*);
port(*port_declarations*);
end component ;

Az *identifier* mezőbe írjuk be komponensünk nevét, amivel majd hivatkozni tudunk rá. A *generic_declarations* helyre az entitásnál megismert lehetőségeink vannak (8.5. fejezet). A portleírásnak meg kell egyeznie az adott komponens entitásleírásában foglaltakkal. Ezen neveket nem használhatjuk abban a VHDL kódban amelyben a komponenst alkalmazzuk.

```
entity Delay is
   Port ( CLK_25MHz : in  STD_LOGIC;
        IDO : in  STD_LOGIC_VECTOR (7 downto 0);
        START_US : in  STD_LOGIC;
          START_MS : in  STD_LOGIC;
        DONE : buffer  STD_LOGIC:='0');
end Delay;
architecture Behavioral of Delay is
          signal ido_ms: STD_LOGIC;         -- 1ms-os belső jel
          signal ido_us: STD_LOGIC;                   -- 1us-os belső jel
          signal szam: unsigned (7 downto 0) :=(OTHERS => '0');
          signal CLK_int: STD_LOGIC;
component kHz1 is                   -- 1 kHz-et előállító komponens
   Port ( CLK : in  STD_LOGIC;
        kHz : buffer  STD_LOGIC);
end component;
component MHz1 is                   -- 1 MHz-et előállító komponens
   Port ( CLK : in  STD_LOGIC;
        MHz : buffer  STD_LOGIC);
end component;
begin
          ms_1: kHz1 port map(CLK_25MHz, ido_ms); -- 1ms előállítása
          us_1: MHz1 port map(CLK_25MHz, ido_us); -- 1us előállítása
          process(START_US, START_MS, CLK_int)
          begin
          if START_US = '1' then    -- ha us bement aktív (prioritásos)
                    CLK_int <= ido_us;         -- akkor mikroszekundumos az órajel
          elsif START_MS = '1' then            -- különben ha ms bement aktív
                    CLK_int <= ido_ms;         -- akkor milliszekundumos az órajel
          end if;
          if falling_edge(CLK_int) then                    -- lefutó élre
                              if DONE = '1' then DONE <= '0'; end if;
                              szam <= szam+1;                    -- szám növelése
                              if szam = unsigned(IDO) then       -- ha letelt az idő
                                        DONE <= '1';                    -- a kimenet = 1
                                        szam <= (OTHERS => '0');
                              end if;
          end if;
          end process;
end Behavioral;
```

Az előző oldalon látható kódrészlet két órajelosztó komponens használatát mutatja be. A részletes projektleírásért l. 9.2.3. fejezet. A *kHz1* és a *MHz1* entitások leírását egy másik VHDL kódban kell szerepeltetnünk, ami szintén a projektünk része. Ügyeljünk, hogy az entitás és a komponens portleírása megegyezzen. A legegyszerűbb, ha az entitás leírását kimásoljuk és beillesztjük abba a kódba, ahol használni akarjuk, majd az *entity* kulcsszót lecseréljük a *component*re, valamint a leírás befejezését is javítjuk *end component*re. A komponensek deklarálását a jelekhez hasonlóan az architektúra leírás *begin* kulcsszava előtt kell szerepeltetnünk. Használatuk egyszerű. Minden komponensnek adjunk egy saját nevet, ami után egy *:* elválasztásával kerül az adott alkatrész eredeti neve. A *port map* kulcsszó után megadhatjuk, hogy mit szeretnénk az adott komponens be- és kimeneteire kötni. Itt szerepelhet konstans, jel, vagy változó is. Ügyeljünk az adathosszokra, és hogy a sorrend megegyezzen azzal, ami az entitás leírásában szerepelt! A komponensek használatbavételére az egyidejű részben kerüljön sor!

A fenti kód tehát a következőt jelenti komponens használat terén. Lesz egy ms_1 nevű alkatrészünk, ami a kHz1 leírás alapján fog működni és a CLK bemenetére a CLK_25MHz bemenet kerül, míg a kHz kimenetét az ido_ms jelre csatlakoztatjuk. Az us_1 nevű alkatrészünk a MHz1 leírás alapján fog működni, a CLK bemenetére a CLK_25MHz bemenet kerül, míg a MHz kimenetét az ido_us jelre csatlakoztatjuk.

A 9. fejezetben számos példát találunk komponensek használatára.

8.8. Operátorok[80]

Az operátorokat az alábbi csoportokba tudjuk sorolni[81]:

Csoportok	Operátorok	A művelet-végrehajtás során alkalmazott prioritás
Logikai	and or nand nor xor	↓
Relációs	= /= < <= > >=	
Összeadó	+ - &	
Előjel	+ -	
Szorzó	* / mod rem	
Egyéb	not ** abs	

8.8.1. Logikai operátorok

Logikai operátorok (*AND, OR, NAND, NOR, XOR, NOT*) *BIT, BOOLEAN* és olyan egydimenziós tömbök esetén alkalmazhatóak, amelyek azonos elemszámmal és *BIT* alapelemekkel rendelkeznek. A *NOT* prioritással rendelkezik a többi logikai operátorral szemben. Azonos operátorok többszöri használata esetén a sorrend lényegtelen. Különböző operátorok esetén zárójelekkel történhet a csoportosítás. Az alábbi kódsorozat a logikai operátorok használatára mutat példát.

```
signal A, B, C: BIT_VECTOR(7 downto 0);          -- 8 bites jelek
signal D, E, F, G: BIT_VECTOR(3 downto 0);       -- 4 bites jelek
signal H, I, J, K: BIT;                          -- 1 bites BIT típusú jelek
signal L, M, N, O, P: BOOLEAN;                   -- 1 bites logikai jelek
.
.
.
A <= B and C;                  -- egy logikai operátor használata tömb esetén
D <= E or F or G;              -- több azonos típusú operátor használata tömb esetén
H <= (I nand J) nand K;        -- több azonos típusú operátor használata
                               -- zárójel használata nem szükséges
L <= (M xor N) and (O xor P);  -- több különböző típusú operátor használata
                               -- zárójel használata szükséges
```

80 A VHDL nyelv delimiters (határolók) néven is hivatkozik bizonyos operátorokra.
81 Más csoportosítás is lehetséges (pl. logikai, relációs, aritmetikai). Az itt alkalmazott csoportok esetén azonban világos a műveletvégzés sorrendje.

8.8.2. Relációs operátorok

Relációs operátorokat két azonos típusú operandus összehasonlítására alkalmazhatunk. Az eredmény egy logikai érték (*TRUE; FALSE*) lesz. Relációs operátorok használatát láthatjuk az alábbi kódsorozatban:

```
signal A, B: BIT_VECTOR(3 downto 0);        -- 4 bites jel
signal C, D: BIT_VECTOR(1 downto 0);        -- 2 bites jel
signal E, F, G, H, I, J: BOOLEAN;           -- logikai jel
.
.
.
G <= (A = B);        -- amennyiben A egyenlő B, akkor G értéke igaz, különben hamis lesz
H <= (C < D);        -- amennyiben C kisebb mint D, akkor H értéke igaz, különben hamis lesz
I <= (C >= D);       -- ha C nagyobb, vagy egyenlő D-nél akkor I értéke igaz
                     -- különben hamis lesz
J <= (E > F);        -- amennyiben E nagyobb mint F, akkor J értéke igaz, különben hamis lesz
```

Fontos, hogy az operátor két oldalán elhelyezett érték, jel, változó azonos típusú s adathosszú legyen!

8.8.3. Összeadó operátorok

8.8.4. Előjel operátorok

8.8.5. Szorzó operátorok

8.8.6. Egyéb operátorok

8.8.7. Operandusok

8.8.7.1. Operandusok tulajdonságai

A VHDL nyelvben nem csak az egyes operandusokra tudunk hivatkozni, hanem azok néhány speciális tulajdonságára is. Ezek a tulajdonságok[82] a következők:

- left
- right
- high
- low
- lenght
- range
- reverse_range
- event
- stable

Az event és a stable – amelyek egy bitre vonatkoznak – kívül a fenti felsorolás tömbök esetén értelmezhető.

A left és right kulcsszóval a tömb balszélső és jobbszélső elemére tudunk hivatkozni. Egy STD_LOGIC_VECTOR esetén pl. el tudjuk dönteni a segítségükkel, hogy a legfelső (MSB), illetve legalsó bit (LSB) milyen értékű, és ettől tehetjük függővé a működést. Vagy egészen addig futtathatunk egy ciklust, míg a tömbünk utolsó elemét fel nem töltötte.

A high és low nyújtotta lehetőségek megegyeznek a left s right módosítóéval. A high a legmagasabb, míg a low a legalacsonyabb elemszámra hivatkozik. Egy bináris szám esetén talán jobban olvasható a high és low.

A lenght segítségével el tudjuk dönteni egy adott tömb méretét, ami nagyon hasznos lehet a vele végzett műveletek során. A range és a reverse_range segítségével tudjuk meghatározni egy tömb terjedelmét. A to és a downto miatt szükséges lehet a visszafelé megadott terjedelem is.

Az event és a stable tulajdonság az if-es szerkezetben, illetve a wait utasítás esetén lehet hasznos számunkra. Az event segítségével élt tudunk figyelni egy adott jelen, míg a stable egy új jelet hoz létre, ami csak akkor változik, mikor az eredetileg figyelt jel stabil állapotba kerül.

8.9. Utasítások

82 Itt csak a Xilinx által támogatott tulajdonságokat jelenítjük meg.

8.9.1. Kombinációs és sorrendi leírás

8.9.2. A process

A folyamat minden sorrendi leírás alapja. A folyamaton belül az egyes utasítások kifejezések kiértékelése egymás után és nem egymással párhuzamosan történik. Amennyiben sorrendi utasításokat szeretnénk végrehajtani mindenképpen szükségünk van a processre.

process(*sensitivity list*)
begin
sorrendi leírás
end process;

A process() a begin és az end process; kötelező nyelvi elem. A zárójelben lévő kifejezés az ún. érzékenységi lista, ami tartalmaz minden olyan jelet, változót, be-, vagy kimenetet ami hatással van a folyamatra, vagyis aminek megváltozása aktiválja, lefuttatja az adott processt[83].

8.10. Csomagok a VHDL nyelvben

8.10.1. Gyári könyvtárak használata

Ebben a fejezetben áttekintjük a legfontosabb Xilinx által is támogatott IEEE szabványos könyvtárakat és használatukat.

8.10.2. Saját package-ek

Gyakran szükségünk van saját könyvtárak használatára is a fejlesztés megkönnyítése érdekében. Ezekben a könyvtárakban általában függvényeket, alkatrészeket, ritkább esetben saját konstansokat, típusokat szerepeltetünk.

Egy elektronikai fejlesztés végrehajtásához általában szükségünk van valamilyen kódolásra-dekódolásra. Ehhez számos függvényt, alkatrészt készíthetünk, amelyeket egy csomagban szerepeltetve megkíméljük magunkat attól, hogy később újra kénytelenek legyünk megírni ezeket az alkalmazásokat. Mindemellett gondoljunk arra, hogy milyen sokféle frekvenciájú órajel szükséges ahhoz, hogy egy feladatot végrehajtsunk. Megtehetjük, hogy létrehozunk egy komponenst, ami a megadott órajelből előállít számunkra egy másik leosztott értéket. Az alkatrészt saját csomagunkban szerepeltetve bármely projektünkben egyszerűen tudjuk implementálni.

83 A Xilinx fordítója elfogadja a process megadását érzékenységi lista nélkül is.

9. Programok a fejlesztőpanelekre

Az FPGA fejlesztést érdemes modulokban elképzelni, minden egyes modulnak megvan a maga feladata, s együtt egy nagy komplexumot képeznek. Azaz egy adott feladatot érdemes lebontani kisebb egységekre, mert ha egy „dobozban” próbáljuk megcsinálni az egészet, könnyen összekuszálódhat, és egy hibás funkció a többi részre is hatással lehet. Ellenben ha egy modult egyszer megírtunk, és működik, megfelelő bemeneti adatok esetén más programelem nem befolyásolhatja. Vegyünk példának egy fényerő-szabályozást. A probléma megoldása alap esetben 2 részből áll, egy PWM modulból, és egy vezérlőből. A vezérlő megmondja hogy milyen kitöltési tényező legyen, a PWM egység végrehajtja. Ha a fényerő-szabályozásnál valami hiba merül fel, abban az esetben le tudjuk szűkíteni a kört a vezérlőre, hiszen ha egyszer már megírtuk a PWM modult és működött, most miért lenne rossz?

A programunk TOP rétegét javasolt SCH-nak (kapcsolási rajz alapúnak) választani, ugyanis a VHDL-el szemben áttekinthetőbb és jobban látjuk, hogy mi a feladata az adott projektnek.

A VHDL kódokban, ahol ezt a feladat megkívánta a változóknak és jeleknek adtunk kezdőértéket. Felhívjuk a figyelmet, hogy az sohasem hiba, ha ezt megtesszük olyan jelek esetén is ahol erre nincs szükség. Amennyiben nem vagyunk biztosak, hogy az adott problémát hogyan befolyásolná, hogy az adott jel random értékről indul, inkább adjunk neki kezdőértéket.

A projektek teljes egészét jelen dokumentumban a terjedelem miatt nem tudjuk szerepeltetni. Maguk a projektek elérhetők iskolánk Moodle rendszerében – javasoljuk, hogy a kódok és rajzok áttanulmányozása közben nyissuk meg a hozzájuk tartozó projekteket is. A leírásban csak a legfontosabb rajzokat, kódokat elemezzük és látjuk el megjegyzésekkel. Ahogy haladunk az egyre bonyolultabb feladatok felé, a magyarázatok annál inkább feltételezik az olvasó VHDL alapismereteit. Elképzelhető, hogy a könnyebb érthetőség kedvéért néhány projektnél érdemes visszalapozni és átolvasni még egyszer az előző feladatoknál leírt fogásokat. A dokumentációban az ismétlések elkerülése végett gyakran hivatkozunk előzőleg már megadott információkra.

A projektek elkészítése során didaktikai célokat is figyelembe vettünk, ezért nem mindig a lehető legegyszerűbb megoldások születtek. Fontosabbnak tartottuk, hogy az olvasót megismertessük a fejlesztés során alkalmazható legfontosabb és leggyakoribb fogásokkal. Ahogy haladunk az egyre bonyolultabb példák felé, úgy próbáljuk adagolni az egyre mélyebb ismereteket. Aki a teljes fejezet áttanulmányozása után újraolvassa az egyes projekteket számos helyen találhat optimalizálási lehetőséget. Ill. új ismereteit felhasználva általánosabban használható, elegánsabb kódot tud majd írni az adott problémára. Az nem várható el, hogy egy ilyen dokumentáció segítségével valakiből profi FPGA fejlesztő váljon. A célunk ehelyett inkább az, hogy a fiatal fejlesztők betekintést nyerjenek az FPGA világába, az egyszerű feladatok végrehajtása során kedvet kapjanak az áramkörrel való behatóbb ismerkedésre.

A konfigurációs bitmintákat természetesen többszörös teszteknek vetettük alá, mielőtt közreadnánk őket. Felhívjuk azonban a figyelmet, hogy minden program az általunk tervezett és használt próbapanelekhez illeszkedik, ezért a más rendszerekkel történő együttműködés biztosításához valószínűleg módosítások elvégzése szükséges.

9.1. 1. generációs próbapanel

Az első generációs próbapanelre történő feladatok megoldásával sajátítjuk el a fejlesztőkörnyezet használatát, illetve az alapvető programozási fogásokat. A példák éppen ezért a lehető legegyszerűbbek. Az első feladatoknál megpróbálunk többféle megközelítést is alkalmazni, később a problémák bonyolultsága miatt már csak egy lehetséges megoldást ismertetünk, ami nem jelenti azt, hogy nem lehetne találni egy másik – talán elegánsabb – megoldást is.

9.1.1. Lábkiosztás a próbapanelhez

Lábszám	Funkció	Megjegyzés
65	SW1; Nyomógomb/kapcsoló	Aktív nullás bemenet
68	SW2; Nyomógomb/kapcsoló	Aktív nullás bemenet
71	SW3; Nyomógomb/kapcsoló	Aktív nullás bemenet
67	SW4; Nyomógomb/kapcsoló	Aktív nullás bemenet
36	CLK1; órajelbemenet	25MHz
37	CLK2; órajelbemenet	
38	CLK3; órajelbemenet	
39	CLK4; órajelbemenet	
17	LED1/Piezo (zümmer)	Piros LED
4	LED2	Piros LED
5	LED3	Piros LED
8	LED4	Piros LED
9	LED5	Piros LED
11	LED6	Piros LED
14	LED7	Piros LED
15	LED8	Piros LED
40	BILED1P; kétszínű LED, piros	
35	BILED1Z; kétszínű LED, zöld	
34	BILED2P; kétszínű LED, piros	
32	BILED2Z; kétszínű LED, zöld	
30	BILED3P; kétszínű LED, piros	
28	BILED3Z; kétszínű LED, zöld	
27	BILED4P; kétszínű LED, piros	
21	BILED4Z; kétszínű LED, zöld	
43	SEGMA; a szegmens	Hétszegmenses kijelző
44	SEGMB; b szegmens	Hétszegmenses kijelző
47	SEGMC; c szegmens	Hétszegmenses kijelző
55	SEGMD; d szegmens	Hétszegmenses kijelző
59	SEGME; e szegmens	Hétszegmenses kijelző
60	SEGMF; f szegmens	Hétszegmenses kijelző
61	SEGMG; g szegmens	Hétszegmenses kijelző
62	SEGMDP; tizedes pont	Hétszegmenses kijelző
85	MUX1; 1. kijelző közös anódja	Aktív nullás kimenet
92	MUX2; 2. kijelző közös anódja	Aktív nullás kimenet
72	MUX3; 3. kijelző közös anódja	Aktív nullás kimenet
81	MUX4; 4. kijelző közös anódja	Aktív nullás kimenet

9.1.2. LED-ek és kapcsolók

9.1.2.1. Egyszerű kombinációs hálózat

Első feladatként egy egyszerű kombinációs hálózatot valósítunk meg, amely modellezi az alapvető logikai függvényeket két bemenetre. A programban felhasználunk még egy bemenetet, amivel ki tudjuk választani, hogy a logikai kapuk bemenetei aktív nullásak[84], vagy aktív egyesek. Ez a funkció az adott lábak invertálását jelenti, amit kizáró-vagy kapcsolattal tudunk elérni.

Összerendelési táblázat:

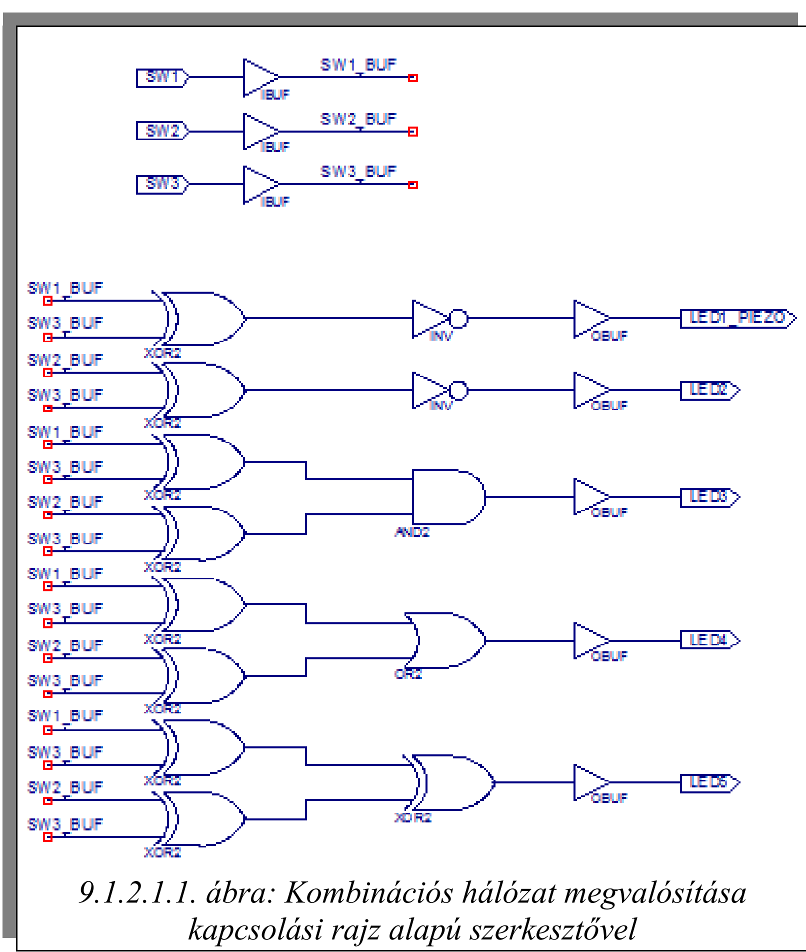
SW1	SW2	SW3	LED1	LED2	LED3	LED4	LED5
1. bemenet	2. bemenet	bemenetek invertálása	1. bemenet negáltja	2. bemenet negáltja	ÉS kapcsolat	VAGY kapcsolat	KIZÁRÓ-VAGY kapcsolat

Az áramkör igazságtáblája alább látható:

SW3	SW1	SW2	LED1	LED2	LED3	LED4	LED5
0	**0**	**0**	1	1	0	0	0
0	**0**	**1**	1	0	0	1	1
0	**1**	**0**	0	1	0	1	1
0	**1**	**1**	0	0	1	1	0
1	**0**	**0**	0	0	1	1	1
1	**0**	**1**	0	1	1	0	0
1	**1**	**0**	1	0	1	0	0
1	**1**	**1**	1	1	0	0	1

84 Az általunk fejlesztett próbapanelen a kapcsolók és nyomógombok aktív nullásak.

A 9.1.2.1.1. ábrán láthatjuk a feladat schematicban megvalósított egy lehetséges megoldását.



9.1.2.1.1. ábra: Kombinációs hálózat megvalósítása kapcsolási rajz alapú szerkesztővel

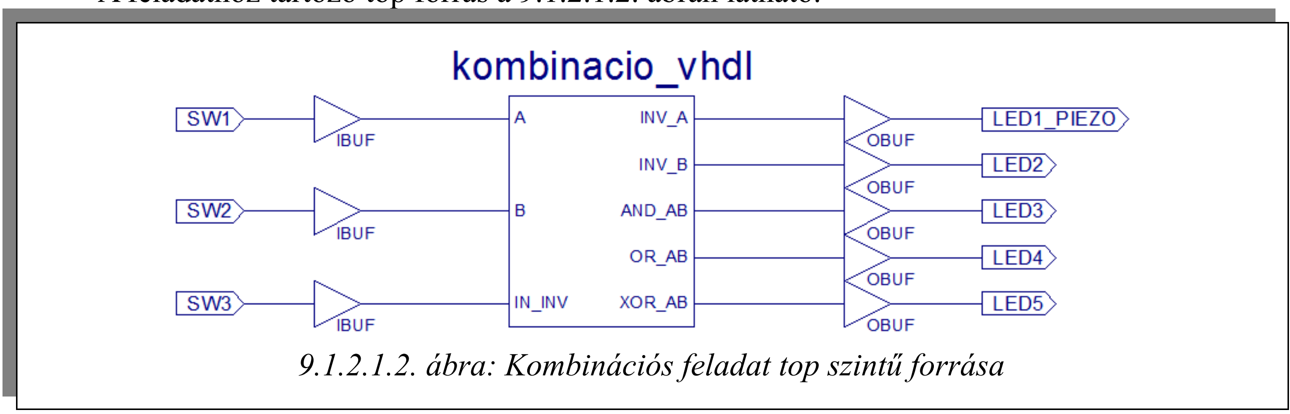
A bemeneteket és kimeneteket IBUF és OBUF elemekkel kell leválasztanunk. Az egyszerűbb huzalozás érdekében a neteket célszerű elneveznünk és az összeköttetést így megvalósítanunk.

Természetesen a feladat végrehajtható egy VHDL kód segítségével is. Az alábbi kódrészlet ezt a célt szolgálja:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity kombinacio_vhdl is
   Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          IN_INV : in  STD_LOGIC;
          INV_A : out  STD_LOGIC;
          INV_B : out  STD_LOGIC;
          AND_AB : out  STD_LOGIC;
          OR_AB : out  STD_LOGIC;
          XOR_AB : out  STD_LOGIC);
end kombinacio_vhdl;
architecture Behavioral of kombinacio_vhdl is
begin
        process(IN_INV, A, B)
        begin
                if IN_INV = '0' then
                        INV_A <= not A;
                        INV_B  <= not B;
                        AND_AB <= A and B;
                        OR_AB <= A or B;
                        XOR_AB <= A xor B;
                else
                        INV_A <= A;
                        INV_B  <= B;
                        AND_AB <= (not A) and (not B);
                        OR_AB <= (not A) or (not B);
                        XOR_AB <= (not A) xor (not B);
                end if;
        end process;
end Behavioral;
```

Maga az entitás megadása grafikus felületen történik a modul elkészítésekor[85]. Az architektúra leírásában egy processt alkalmaztunk, mert szükségünk volt az if funkcióra, ami csak sorrendi folyamat esetén értelmezhető. Az IN_INV bemenet függvényében vagy az if, vagy az else ág fog végrehajtódni. Az érzékenységi listába a 3 bemenet kerül. Az else ágban alkalmazhattuk volna a De-Morgan azonosságot és akkor egy invertálást meg tudnánk spórolni az ÉS és a VAGY felcserélésével[86].

A feladathoz tartozó top forrás a 9.1.2.1.2. ábrán látható.



9.1.2.1.2. ábra: Kombinációs feladat top szintű forrása

85 Természetesen ezt később a karakteres részben tudjuk módosítani//bővíteni.

86 $\overline{A\cdot B}=\overline{A}+\overline{B}$; $\overline{A+B}=\overline{A}\cdot\overline{B}$

A következő kódrészlet egy másik – teljesen egyenértékű – megoldás[87].

```
architecture Behavioral of kombinacio_vhdl is
	signal A_TEMP: STD_LOGIC;
	signal B_TEMP: STD_LOGIC;
begin
	process(IN_INV, A,B)	-- a process negálja A és B bemenetet
	begin				-- ha IN_INV = 1
	if	IN_INV = '0' then
		A_TEMP <= A;
		B_TEMP <= B;
	else
		A_TEMP <= not A;
		B_TEMP <= not B;
	end if;
	end process;
			INV_A <= not A_TEMP;		-- egyidejű leírás
			INV_B  <= not B_TEMP;
			AND_AB <= A_TEMP and B_TEMP;
			OR_AB <= A_TEMP or B_TEMP;
			XOR_AB <= A_TEMP xor B_TEMP;
end Behavioral;
```

A fenti kódrészlet tartalmaz sorrendi és egyidejű utasításokat is. A process jelen esetben két jelet állít elő, amik az IN_INV bemenet függvényében vagy az A és B bemenet, vagy annak negáltjai. Az egyidejű folyamat ezeket a jeleket használja fel a függvények bemeneteihez. A működés megegyezik az előző leírással.

Alkalmazhattuk volna a schematicra illeszkedő leírást is, amennyiben az invertálást XOR kapcsolattal oldjuk meg. Ebben az esetben nem lett volna szükség az if elágaztató utasításra és így a processre se. Az így leírt működés azonban nehezebben értelmezhető/olvasható kódot eredményez.

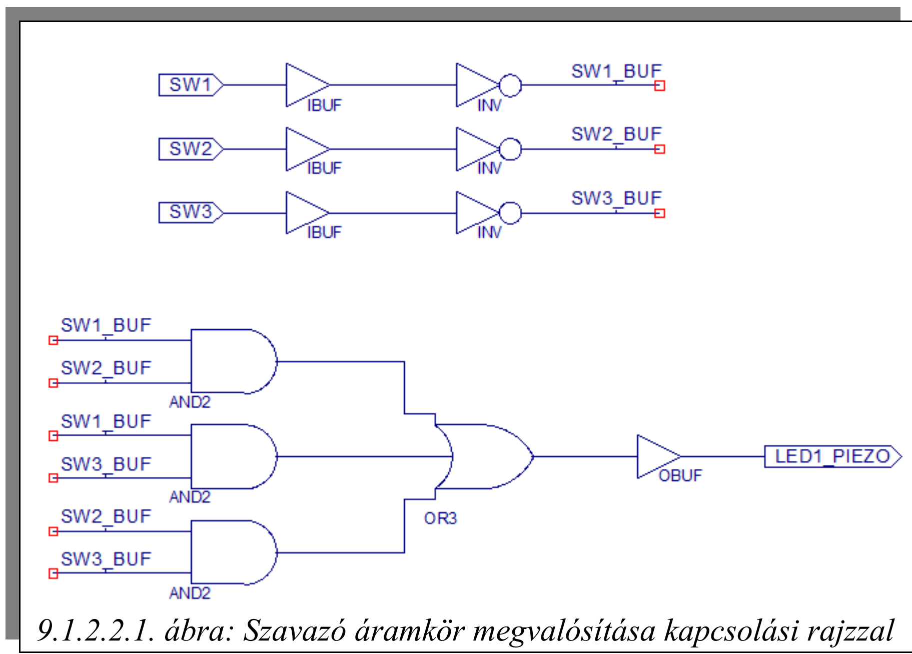
Miután STD_LOGIC be- és kimeneteket alkalmaztunk a VHDL kódban az IBUF és OBUF elemektől eltekinthetünk az automatikus bufferelés miatt.

87 Mivel csak az architektúra leírás változott, ezért csak ezt a részt szerepeltetjük. A továbbiakban is hasonlóan járunk el.

9.1.2.2. Szavazó áramkör

Az áramkör lényege, hogy a kimenetén akkor ad igaz logikai értéket, ha a bemenetein érkező jelek közül a többség logikai igaz érték. A stabil működéshez legalább 3 páratlan számú bemenet szükséges[88]. Az egyszerűség kedvéért itt 3 bemenetre valósítjuk meg az áramkört.

A feladat megoldható egy schematic-ban megrajzolt áramkör (9.1.2.2.1. ábra), vagy egy VHDL kód segítségével.



9.1.2.2.1. ábra: Szavazó áramkör megvalósítása kapcsolási rajzzal

A művelethez elegendő pár logikai kaput felhasználnunk. Először is a bemenetet invertálnunk kell mert a periféria panelen lévő kapcsoló aktív '0'-ás[89]. Amennyiben a feladatot meg akarjuk fogalmazni a digitális technika nyelvén, akkor a következő függvényt kapjuk:

$$F^3 = AB + AC + BC + ABC$$

A kombinációs hálózat igazságtáblája:

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

88 Egy, vagy két bemenet esetén nincs értelme többségről beszélni. Páros számú bemenet esetén elképzelhető, hogy a bemeneti kombinációból ugyanannyi az igaz, mint a hamis, s ilyenkor az áramkör bizonytalanná válhat.

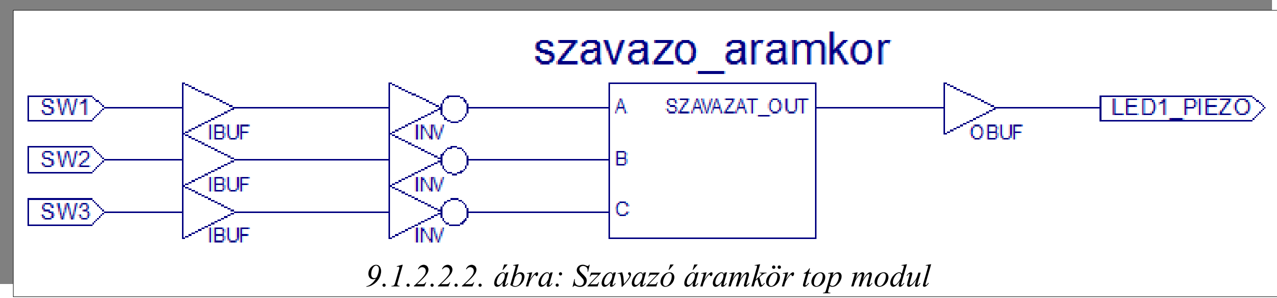
89 A kapcsoló bekapcsolt állapotban logikai ”0”, míg kikapcsolt állapotban logikai ”1” értéket ad.

A 9.1.2.2.1. ábrán látható kapcsolásban 3 db kétbemenetű ÉS kaput használtunk fel, amelyekre páronként kötöttük a bemeneteket. Az ABC kombináció nem tartalmaz új információt.

A következő kódsorozat a lehető legegyszerűbben valósítja meg az áramkört:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity szavazo_aramkor is
   Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          C : in  STD_LOGIC;
          SZAVAZAT_OUT : out  STD_LOGIC);
end szavazo_aramkor;
architecture Behavioral of szavazo_aramkor is
begin
          SZAVAZAT_OUT <= (A and B) or (A and C) or (B and C);
end Behavioral;
```

A VHDL kódban valójában a schematicban megadott logikai funkciókat realizáltuk. A kódhoz tartozó top forrás a 9.1.2.2.2. ábrán látható. Az invertálást itt valósítottuk meg. Ennek praktikus okai vannak. Amennyiben olyan módosító részt kell a konfigurációs bitmintába integrálnunk, ami hardverspecifikus célszerű ezt minél magasabb szinten realizálni, így az alsóbb szinteken található modulok működése nem függ a hardvertől.



9.1.2.2.2. ábra: Szavazó áramkör top modul

A lenti kódrészlet ugyanezt a feladatot hajtja végre, más megközelítéssel.

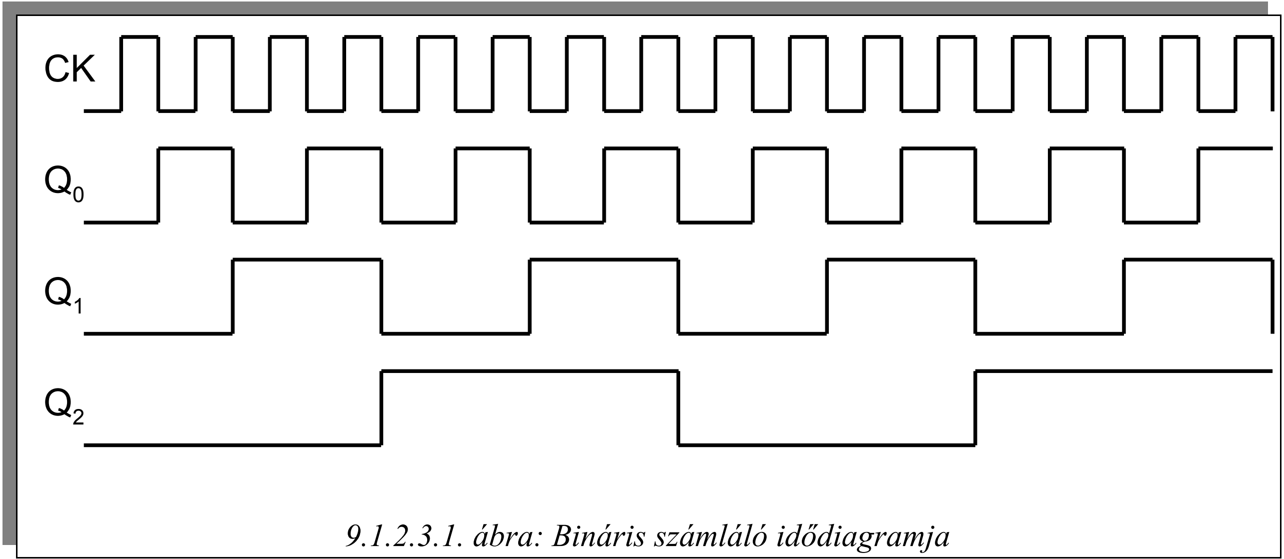
```
architecture Behavioral of szavazo_aramkor is
          signal ABC: STD_LOGIC_VECTOR(2 downto 0);
begin
          process(ABC)
          begin
                    case ABC is
                              when "000" => SZAVAZAT_OUT <= '0';
                              when "001" => SZAVAZAT_OUT <= '0';
                              when "010" => SZAVAZAT_OUT <= '0';
                              when "011" => SZAVAZAT_OUT <= '1';
                              when "100" => SZAVAZAT_OUT <= '0';
                              when "101" => SZAVAZAT_OUT <= '1';
                              when "110" => SZAVAZAT_OUT <= '1';
                              when "111" => SZAVAZAT_OUT <= '1';
                              when others =>
                    end case;
          end process;
          ABC <= A&B&C;
end Behavioral;
```

A feladat végrehajtásához felhasználtuk a *CASE* utasítást – ennek alkalmazásához szükségünk van egy *processre*. Miután a top modulban a bemeneteket nem célszerű buszos kialakításban kezelni, a *case* utasításhoz azonban a vektoros ábrázolásra van szükségünk, ezért meg kell oldani az egyes bemenetek összefűzését. Ehhez az *&* összefűzés operátort alkalmaztuk. Ne tévesszen meg senkit, hogy az összefűzés a *process* után szerepel, hiszen mind a folyamat, mind az összefűzés az egyidejű részben van, ezért ezek párhuzamosan hajtódnak végre. Az *ABC* 3 bites jel tehát a 3 bementünk összefűzése. Jelen esetben – miután a bitek sorrendje nem, csak az egyesek száma számít – mindegy, hogy a *downto*, vagy a *to* kulcsszót alkalmazzuk a *signal* megadásánál, ill. az összefűzés során is tetszőleges az *A*, *B*, *C* bemenetek sorrendje. A kódhoz ugyanaz a schematic top modul tartozik, mint az előzőhöz.

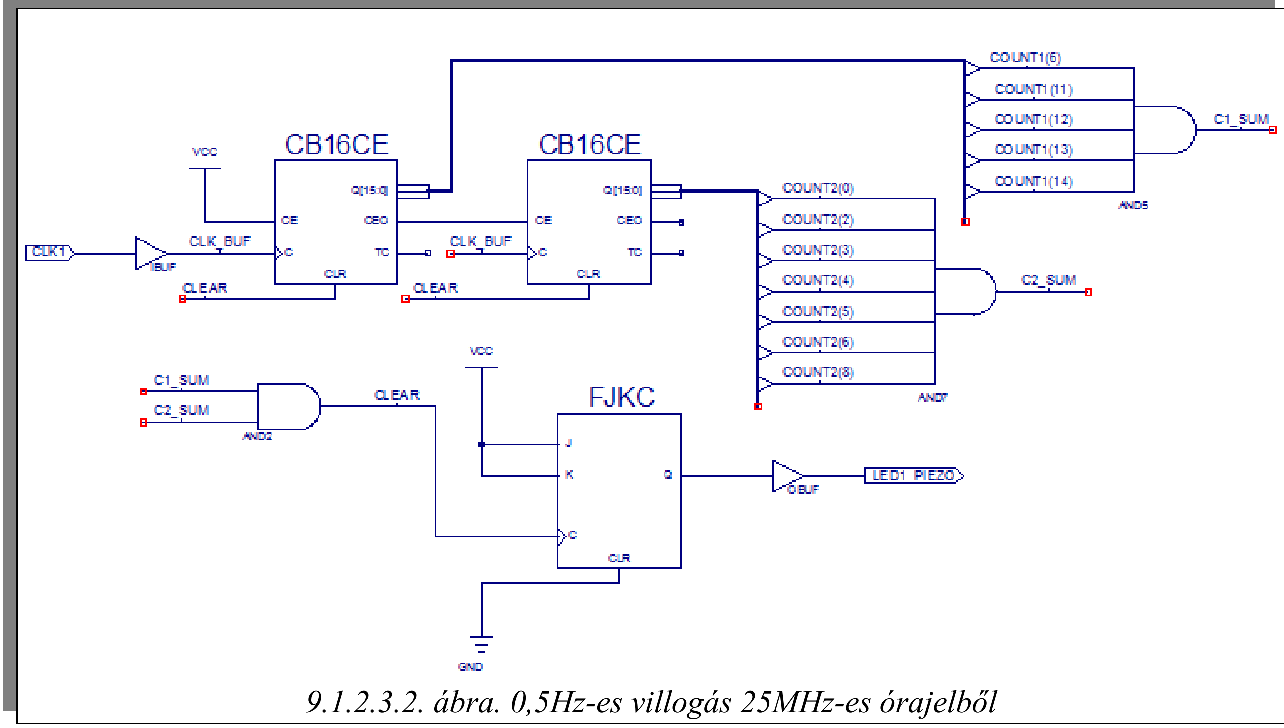
Az első VHDL kód amit bemutattunk kétség kívül egy optimalizált megoldás a logikai cellák minimalizálása érdekében. Vigyáznunk kell azonban, hiszen a kódból a Xilinx fejlesztőkörnyezete generál majd bitmintát, és az általunk optimálisabbnak vélt kód, lehet hogy adott esetben több *CLB*-t igényel. Ilyen egyszerű projekteknél ez a lehetőség még nem áll fenn. Mikor érdemes mégis a második verziót alkalmazni? Nos, ha az igazságtáblában változás történik, a második VHDL kódban ezt könnyű realizálni, míg az elsőben nehézkes. Mindemellett a második kód sokkal beszédesebb (látjuk az igazságtáblát), azonban terebélyesebb is. A kérdésre még visszatérünk a hétszegmenses kijelző dekóderének elkészítésekor (l. 9.1.3.1. fejezet).

9.1.2.3. Adott frekvenciájú négyszögjel előállítása (villogás)

A frekvencia leosztása lényegében egy számlálóval történik. A számláló bemenetére kötünk egy adott frekvenciájú négyszögjelet és ő a kimenetein ennek 2 hatványainak megfelelő leosztott értéket állít elő (idődiagram l. 9.1.2.3.1. ábra). Így előállítható a frekvencia fele, negyede, nyolcada, stb. Amennyiben a kimenetek együttes feltételét szabjuk meg, akkor tetszőleges egész szám előállítható. Pl. a Q_0 minden lefutó élre invertálódik, a Q_1 pedig minden második lefutó élre. Ha a harmadik lefutó élt szeretnénk feltételül szabni, akkor az a pillanat kell, amikor a Q_0 és a Q_1 is egyszerre van magas szinten. A Xilinx fejlesztőkörnyezetében 16 bites a legnagyobb bitszélességgel rendelkező számláló, ez sokszor kevésnek bizonyulhat. Ilyen esetekben a számlálókat egymás után fűzve nagyobb bitszám is elérhető.



9.1.2.3.1. ábra: Bináris számláló idődiagramja



9.1.2.3.2. ábra. 0,5Hz-es villogás 25MHz-es órajelből

Az adott frekvencia előállításához a következő lépéseket kell tennünk. A CLK1-es clock bemenetet használva az órajelünk 25MHz. Ahhoz, hogy 1Hz-et tudjunk előállítani 25.000.000-val kell osztanunk az eredeti jelet[90]. A szám bináris értéke: „1011111010111100001000000”. Amikor a számlálónk eléri ezt az értéket, akkor kell invertálni a kimenetünket és elölről kezdeni a számlálást.

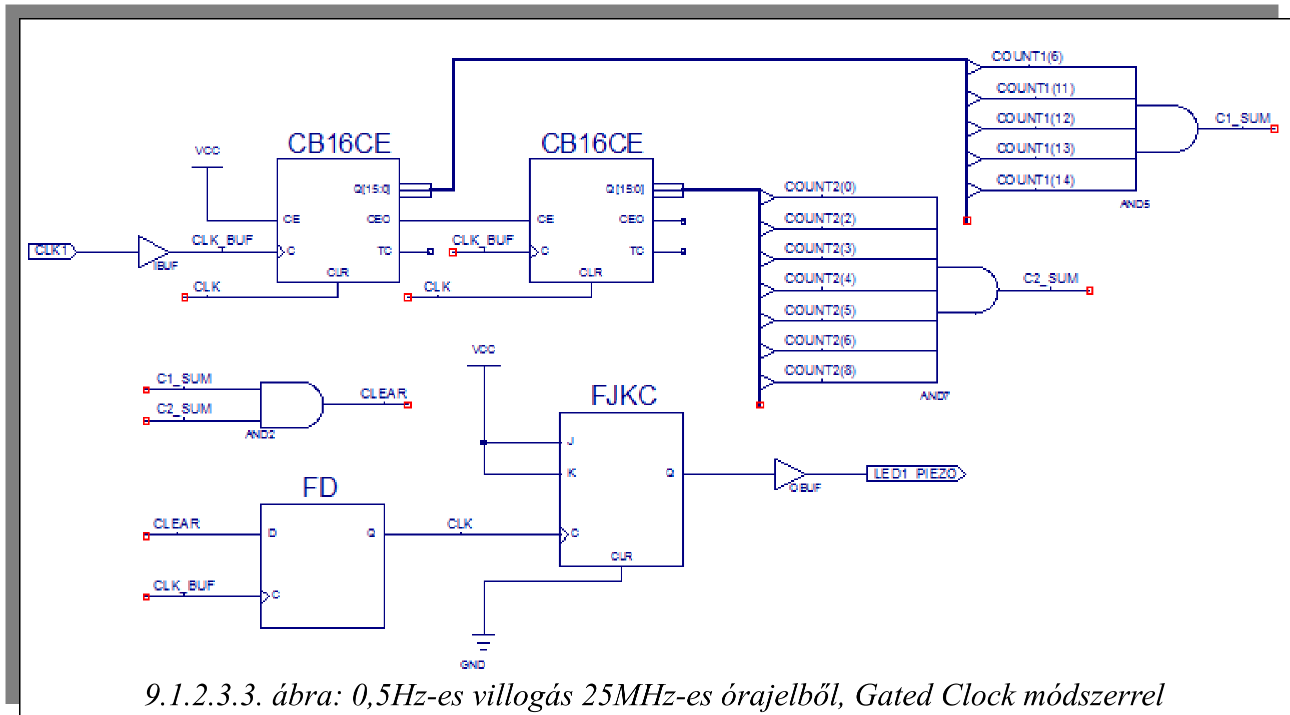
Több problémába is ütközünk a folyamat során. Először is nincs ilyen nagy bitszámú számláló a schematic részben, másodszor nem egy bitet kell invertálnunk, hanem egy esemény hatására kell a kimenetet invertálni, ami utána az esemény következő bekövetkeztéig megőrzi tartalmát. Az első problémát két 16 bites számláló összefűzésével tudjuk megoldani. A számlálók bővítését a szinkron számlálóknál ismert módon végezzük el. A 9.1.2.3.2. ábrán is látható kialakítás szerint a számlálók órajele közös, és a felfűzést a *CEO* kimenet és *CE* bemenet segítségével valósítjuk meg. Az invertálást és tárolást egy JK flip-flop segítségével célszerű elvégezni. A J és K bemenetekre egyet kötve, a kimenet minden órajel hatására az eredeti érték negáltja lesz. A teljes áramkör rajzát a 9.1.2.3.2. ábrán láthatjuk. A számláló megfelelő kimeneteinek[91] együttes fennállását *ÉS* kapukkal figyeljük. Ügyeljünk rá, hogy a 16. bit a második számláló 0. bitje és így tovább. A számlálókat az esemény bekövetkeztekor törölnünk kell – erre szolgál a *CLEAR* jel.

Figyeljük meg, hogy bár 1Hz-es négyszögjelet szerettünk volna előállítani a programunk mégis 0,5Hz-et hozott létre! A fenti programot működtetve a LED 1 másodpercig be-, majd egy másodpercig kikapcsolt állapotban van és a folyamat kezdődik elölről. A problémát az okozza, hogy a számláló adott kimenetinek együttes fennállásakor az áramkör kimenetét invertáljuk, azonban egy perióduson belül két invertálás szükséges. Ezért ha egy adott frekvenciához tartozó számot kiszámoltunk azt mindig 2-vel kell osztanunk, így kapva meg a fenti elrendezéssel a helyes frekvenciát.

A módszerrel a frekvenciát bármilyen egész számmal tudjuk osztani.

A schematic fordítása során több warning üzenetet is kapunk. Ezek egy része arra vonatkozik, hogy nem használtuk fel a számláló minden lábát, ami esetünkben nem okoz problémát. Egy másik hibaüzenet jelzi, hogy ún. *Gated Clock* áll fenn a JK tároló órajel bemenetén. Ez egy módszer az energiafelhasználás csökkentése érdekében. Lényege, hogy az állandó fogyasztást jelentő kombinációs logika helyett egy szinkron tárolóról vezéreljük az órajel bemenetet. A tároló órajelét szolgáltathatja a rendszer órajele mivel a JK tároló kimenete csak ezzel a jellel szinkronban változhat. Az így javított változat a 9.1.2.3.3. ábrán látható.

90 Olvassuk el a félkövérrel szedett részt!
91 Ahol a bináris számunkban 1 szerepel.



9.1.2.3.3. ábra: 0,5Hz-es villogás 25MHz-es órajelből, Gated Clock módszerrel

Frekvenciaosztást természetesen VHDL kód segítségével is meg tudunk valósítani. Az alábbi kód és a 9.1.2.3.4. ábrán látható top schematic már 1Hz-es villogást eredményez a LED-en.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
entity villog is
   Port ( CLK_25MHz : in  STD_LOGIC;    -- 25Mhz-es órajel
          OUT_1Hz : out  STD_LOGIC);    -- 1Hz-es kimenet
end villog;
architecture Behavioral of villog is
        signal szam: unsigned (23 downto 0):= (others => '0'); --24 bites jel az osztáshoz
        signal OUT_1Hz_temp: STD_LOGIC:= '0'; -- segédjel a kimenethez
begin
        process(CLK_25MHz)
        begin
                if CLK_25MHz'event and CLK_25MHz='0' then       -- ha az órajelen lefutó él érkezik
                szam <= szam +1;                                        -- a számot növeljük eggyel
                        if szam = 12500000 then                 -- ha elértük a 12500000 értéket
                                OUT_1Hz_temp <= not OUT_1Hz_temp;       -- akkor a segédjelet
                                                                        -- invertáljuk
                                szam <= (others => '0');                -- és a számot töröljük
                        end if;
                end if;
        end process;
        OUT_1Hz <= OUT_1Hz_temp;                        -- a kimenet megegyezik a segédjellel
end Behavioralz
```

Az entitásnak egy 25MHz-es órajelbemenete és egy 1Hz-es kimenete van. Két jel segíti az architektúra leírását. Az egyik a kimenetet puffereli, míg a másik segítségével számoljuk az órajelimpulzusokat. Az *OUT_1Hz_temp* jelre azért van szükségünk, mert a kimenet nem olvasható, így az invertálást nem tudnánk megvalósítani. Egyéb esetekben sem célszerű közvetlenül a kimenetet írni, vele műveleteket végezni. A szám kezdőértékének megadásánál és törlésénél használtuk az *others* kulcsszót az egyszerűbb írásmód kedvéért.

A számot – miután műveleteket kell végeznünk vele[92] – nem célszerű *std_logic_vector*-ként megadni. **Ahhoz, hogy használhassuk az unsigned típust meg kell adnunk az *IEEE.numeric_std* könyvtárat[93].** Az órajel élváltását az *event* kulcsszó segítségével tudjuk detektálni – a kódban lefutó élt figyelünk.

9.1.2.3.4. ábra: 1Hz-es villogás top modulja

Az alábbi kódrészlet ugyanezt a funkciót valósítja meg, azonban felfedezhetünk néhány eltérést.

```
architecture Behavioral of villog is
	signal szam: unsigned (23 downto 0):= (others => '0'); -- 24 bites jel az osztáshoz
begin
	process(CLK_25MHz)
	begin
		if falling_edge(CLK_25MHz) then -- ha az órajelen lefutó él érkezik
		szam <= szam +1;				-- a számot növeljük eggyel
			if szam = 12500000 then		-- ha elértük a 12500000 értéket
				OUT_1Hz <= not OUT_1Hz;	-- akkor a segédjelet invertáljuk
				szam <= (others => '0');	-- és a számot töröljük
			end if;
		end if;
	end process;
end Behavioral;
```

A kimenetet bufferként adtuk meg, ezáltal elértük, hogy tudjuk olvasni is. Így feleslegessé vált az egyik jel. Megjegyezzük azonban, hogy a kimenetet szimbolizáló segédjelekre gyakran szükség van a programozás során a műveletvégzés megkönnyítése érdekében. A lefutó élt a standard könyvtárban megadott függvény segítségével teszteljük. A függvény nem csak az élváltást és a szintet figyeli, hanem megvizsgálja az előző állapotot is. Így el tudjuk kerülni a hamis éldetektálásokat, amik fel/lehúzó ellenállások használatából, nagyimpedanciás váltásokból erednek[94]. **Ezért javasoljuk, hogy mindig ezen függvényeket alkalmazzuk a hibamentes működés biztosításához az if-es szerkezettel szemben.**

Mindkét kódban az 1Hz-hez 12.500.000-al kellett osztanunk a 25MHz-es órajelet, mivel egy perióduson belül két invertálás szükséges a már a schematic alapú programnál leírtak szerint.

92 A legtöbb művelet az unsigned és signed típusok esetén értelmezhető. Részletekért l. . oldal.

93 Megadhatnánk az *IEEE.STD_LOGIC_UNSIGNE*D és/vagy az *IEEE.STD_LOGIC_ARITH* package-et is, amelyek szintén tartalmazzák az unsigned típust. Mindemellett nem muszáj a teljes (all) könyvtárat beincludolnunk, elég csak az általunk használni kívánt részt (pl. egy konverziós függvény használatához a use *IEEE.STD_LOGIC_ARITH.CONV_STD_LOGIC_VECTOR* megadása is elfogadható). Vigyázzunk azonban a gyári include fájlokkal, mert sajnos itt elég furcsa és nehezen kiküszöbölhető hibákba ütközhetünk. l. 116. oldal.

94 Részletekért l. http://vhdlguru.blogspot.hu/2010/04/difference-between-risingedgeclk-and.html

9.1.2.4. Pergésmentesítés

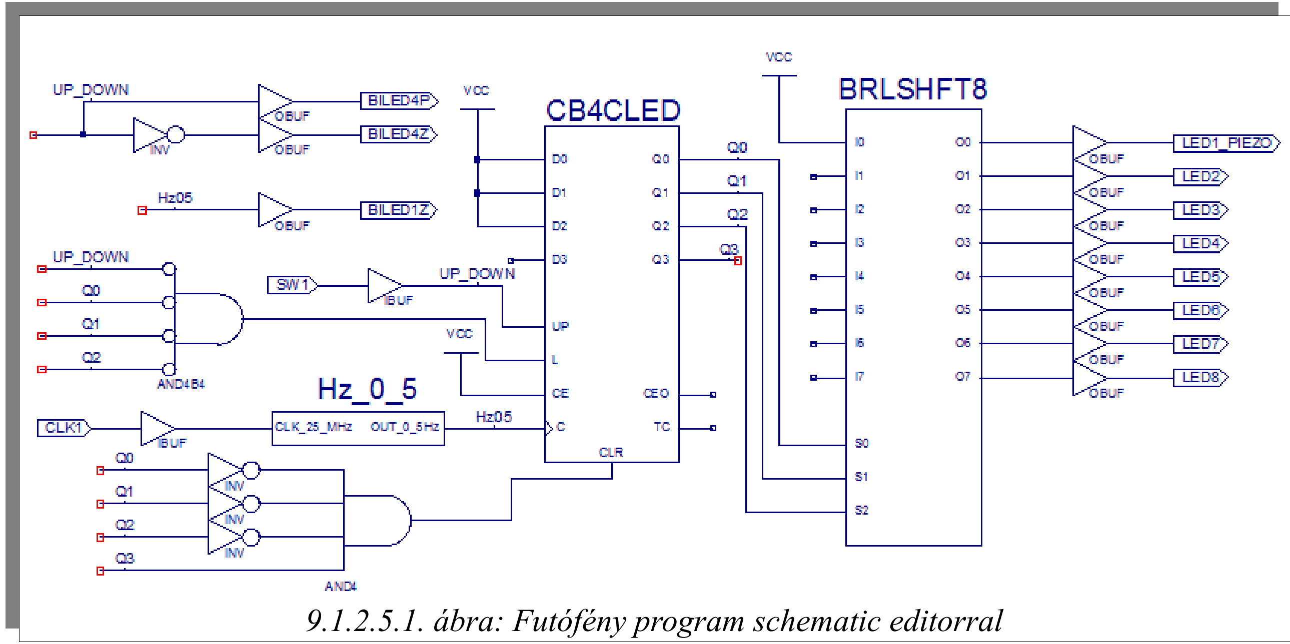
9.1.2.5. Futófény programok

Futófény létrehozásához többféle áramköri elem is rendelkezésünkre áll a schematic editorban. Első lépésként szükségünk van egy időzítő modulra, ami a léptetés frekvenciáját szolgáltatja. Ehhez felhasználjuk a villogó program esetén megalkotott schematic forrás. Az sch kiterjesztésű állományt másoljuk a jelenlegi projektünk könyvtárába és adjuk hozzá a projektünkhöz. Miután most nem a villogás a funkciója, hanem időzítés, célszerű ennek megfelelően átnevezni. Az *I/O markereket* is nevezzük át. Az *ibuf* és *obuf* elemeket cseréljük bufferekre (buf)[95]. Majd csináljunk belőle saját alkatrészt[96]. A fény futtatásához a 9.1.2.5.1. ábrán látható elrendezésben egy ún. *barrel shiftert* alkalmaztunk. Az alkatrész működésének leírását a következő táblázat tartalmazza:

Bemenetek											**Kimenetek**							
S2	**S1**	**S0**	**I0**	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**O0**	**O1**	**O2**	**O3**	**O4**	**O5**	**O6**	**O7**
0	0	0	a	b	c	d	e	f	g	h	a	b	c	d	e	f	g	h
0	0	1	a	b	c	d	e	f	g	h	b	c	d	e	f	g	h	a
0	1	0	a	b	c	d	e	f	g	h	c	d	e	f	g	h	a	b
0	1	1	a	b	c	d	e	f	g	h	d	e	f	g	h	a	b	c
1	0	0	a	b	c	d	e	f	g	h	e	f	g	h	a	b	c	d
1	0	1	a	b	c	d	e	f	g	h	f	g	h	a	b	c	d	e
1	1	0	a	b	c	d	e	f	g	h	g	h	a	b	c	d	e	f
1	1	1	a	b	c	d	e	f	g	h	h	a	b	c	d	e	f	g

Látható, hogy ha S0÷S2 lábakon lévő számot nullától egyesével növeljük, akkor pont a számunkra szükséges egyik funkciót (balra léptetés) érjük el, amennyiben a számot 7-től csökkentjük, akkor a másik funkciót (jobbra léptetés) kapjuk. Az *I* bemenetekre kell kötnünk a léptetni kívánt kombinációt.

95 I/O illesztést mindig csak a legfelső szintű forrásban helyezzünk el az átláthatóság és a bővíthetőség érdekében.
96 Részletekért l. . fejezet.



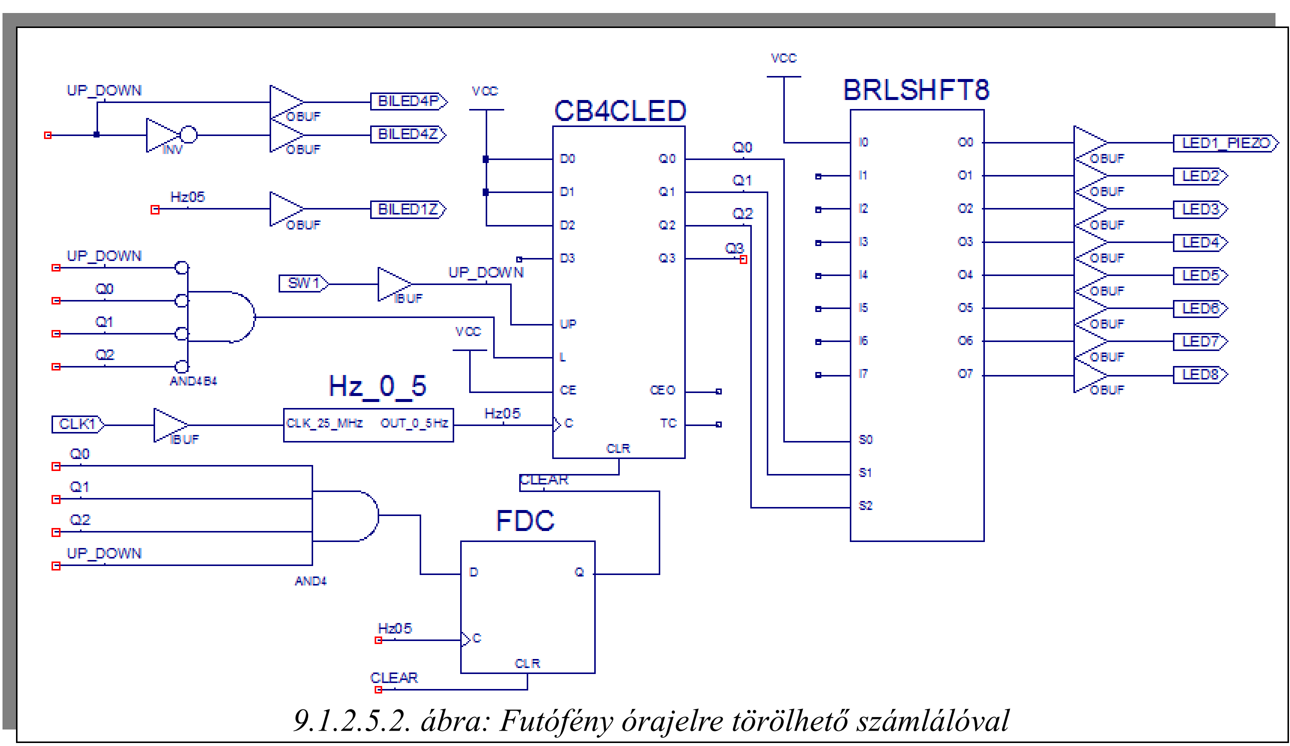
9.1.2.5.1. ábra: Futófény program schematic editorral

Az S0÷S2 bemeneteket célszerűen egy bináris számlálóval hajtjuk meg. A számláló órajelét a fél Hz-es villogó programunkból készített alkatrész szolgáltatja. Amennyiben felfelé számolunk a „1000” kombináció helyett a „0000”-nak kell megjelennie, vagyis a számlálást elölről kell kezdenünk – hiszen csak 3 bites számlálóra van szükségünk a gyári négy bites helyett. Ezt a funkciót elérhetjük, ha a kombináció bekövetkeztekor töröljük a számlálót. A bal alsó sarokban látható inverterek és egy *ÉS* kapu végzi el a kombináció figyelését. Lefelé számláláskor az „1111” kombináció helyett kell a „0111”-nek megjelennie. Ezt a kombinációt kötöttük a *D0÷D3* bemenetekre. Vigyázzunk azonban, hogy a *LOAD* bemenet a *CLEAR* bemenettel szemben szinkron, vagyis csak a következő órajelre fogja beolvasni az értékét a számláló. Ezért az eggyel előző kombinációt („0000”) vezetjük a *LOAD* lábra. Ez a kombináció érvényes a másik irányban történő számlálás esetén, ezért csak akkor vesszük figyelembe, ha lefelé számlálás van (ezért van a kombinációs logika bemenetére kötve az *UP/DOWN* láb is)[97]. Az órajelet egy LED-re, a számláló *UP/DOWN* bemenetét egy kétszínű LED-re vezettük ki. A nem használt bemeneteket a Xilinx alapértelmezetten nullára csatlakoztatja (ezt a warning üzenetek között közli is velünk).

A 9.1.2.5.2. ábrán látható áramkör ugyanezt a funkciót látja el, azonban itt a *CLEAR* lábat is szinkron bemenetként működtetjük[98]. Gondoljuk végig, hogy a 9.1.2.5.1. ábrán látható verzióban a számláló kimenetén egy pillanatra érvénytelen érték („1000” a „0000” helyett) jelenik meg, amíg a számláló nem törlődik. Mivel ez aszinkron módon történik a hatás majdnem azonnali. Amennyiben olyan áramkört üzemeltetünk a számlálónkról aminél ez a rövid ideig tartó helytelen kombináció sem megengedhető, akkor szükséges a 9.1.2.5.2. ábrán látható kialakításhoz fordulnunk. Most ez nem okozott problémát, hiszen a számláló kimenetének helytelen értékét (Q_3) nem használtuk fel.

97 A törlésnél olyan kombinációt alkalmaztunk, ami nem szerepel a másik irányban történő számlálás szekvenciájában, ezért ott nem kellett az *UP/DOWN* lábat figyelnünk. A Q_3 kimenetet is rávezethettük volna a *LOAD* bemenet kombinációs hálózatára, azonban vegyük észre, hogy ettől függetlennek kell lennie a működésnek. A Q_3 kimenet lefelé számláláskor semmilyen esetben se lehet aktív!

98 Mivel csak a következő órajelre történik a beolvasás, ezért az „1000” kombináció helyett a „0111” kombinációt kapuzzuk ki. A Q_3 kimenet feleslegessé vált, hiszen most se a fel, se a lefelé számlálásnál nem engedjük meg, hogy a Q_3 aktív legyen. Az *UP/DOWN* viszont kell a CLEAR előállításához is, mert a „0111” érvényes kimeneti érték a lefelé számlálásnál is. Fontos, hogy ne hagyjuk, a következő órajelig törlésben a számlálót, hiszen akkor elvesztegetünk egy *clockot*, ezért olyan *flip-flopot* alkalmaztunk, ami törölhető. Így elérhetjük, hogy a *CLEAR* csak egy nagyon rövid ideig áll fenn, valójában csak egy impulzus keletkezik.



9.1.2.5.2. ábra: Futófény órajelre törölhető számlálóval

A következő VHDL kód és a hozzá tartozó top modul (9.1.2.5.3. ábra) egy olyan futófény programot hajt végre, amin beállítható két kapcsoló segítségével, hogy jobbra/balra, vagy oda-vissza fusson a fény, vagy megálljon a kétszínű LED-eken. Egy másik kapcsoló segítségével beállítható, hogy a szín ami fut piros, vagy zöld legyen.

Szükségünk van az *IEEE.NUMERIC_STD* csomagra, mert ez tartalmazza a forgató utasítást. Ehelyett használhatnánk az *IEEE.STD_LOGIC_ARITH*[99] *package*-t és a benne lévő léptető utasítást. Vigyázzunk azonban, mert a két könyvtár együttes használata kerülendő, ugyanis komoly típuskonverziós gondokat okozhat. A csomagok közötti átfedések okozzák a problémát. Az *unsigned* és *signed* típus mellett jó néhány függvény is definiálva van mindkét könyvtárban, ráadásul eltérő módon. A különböző VHDL fórumokon – még a hivatalosokon is – eléggé megoszlanak a vélemények erről a problémáról. A többség az *IEEE.NUMERIC_STD* csomag használatát javasolja. Komoly hiányossága azonban a könyvtárnak, hogy nem tartalmazza a másik *package* széleskörű konverziós függvényeit. A javaslatunk, hogy ha lehet használjuk mindig az *IEEE.NUMERIC_STD* csomagot. Mindenképpen kerüljük a két könyvtár együttes használatát. Ha szükségünk van mindkét *include* fájlból függvényekre, akkor próbáljuk meg ezeket egyenként használatba venni a teljes *package* helyett. Megjegyezzük, hogy a gyári csomagok használata nem szükséges a VHDL alapú fejlesztéshez, csupán megkönnyíti azt. A most ismertetett probléma is mutatja, hogy sajnos sokszor még a szabványosított csomagokkal is akadhatnak problémák.

A következőekben csak az új operátorok/függvények esetén hívjuk fel a figyelmet a megfelelő csomagok használatára, de **egy kód elemzésekor soha ne felejtsük el megnézni, hogy milyen package-k szükségesek a helyes futtatáshoz! Fontos, hogy csak olyan csomagokat használó kódot alkalmazzunk, amelyeket a Xilinx támogat**[100]**!**

99 Gyakran a *IEEE.STD_LOGIC_UNSIGNED* csomaggal együtt használatos.
100Részletekért l . oldal.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity Futofeny is
    Port ( SW1 : in  STD_LOGIC;
           SW2 : in  STD_LOGIC;
           SW3 : in  STD_LOGIC;
           LED : out  STD_LOGIC_VECTOR (7 downto 0);
           CLK : in  STD_LOGIC);
end Futofeny;
architecture Behavioral of Futofeny is
        signal SW12: STD_LOGIC_VECTOR(1 downto 0);
        signal LED_position: unsigned (3 downto 0):="1000";
        signal oda_vissza: bit :='0';
begin
        process(SW12, CLK)      -- LED pozicionálás
        begin
                if falling_edge(CLK) then               -- CLK lefutó élére indul a folyamat
                        case SW12 is
                --                      when "00" =>    ebben az esetben a futófény áll (when others)
                                when "01" =>                    -- futófény balra
                                        LED_position <= LED_position rol 1;
                                                                -- a LED-eken egy forgatás balra
                                when "10" =>                    -- futófény jobbra
                                        LED_position <= LED_position ror 1;
                                                                -- a LED-eken egy forgatás jobbra
                                when "11" =>                    -- futófény oda-vissza
                                        if oda_vissza = '1' then
                                                if LED_position = "1000" then
                                                                -- ha már átfordulna
                                                        LED_position <= LED_position ror 1;
                                                                -- visszaforgatjuk
                                                        oda_vissza <= '0';
                                                                -- és másik irányt kérünk
                                                else            -- különben
                                                        LED_position <= LED_position rol 1;
                                                                -- a LED-eken egy forgatás balra
                                                end if;
                                        else
                                                if LED_position = "0001" then
                                                                -- ha már átfordulna
                                                        LED_position <= LED_position rol 1;
                                                                -- visszaforgatjuk
                                                        oda_vissza <= '1';
                                                                -- és másik irányt kérünk
                                                else            -- különben
                                                        LED_position <= LED_position ror 1;
                                                                -- a LED-eken egy forgatás jobbra
                                                end if;
                                        end if;
                                when others =>          -- semmi
                        end case;
                end if;
        end process;
```

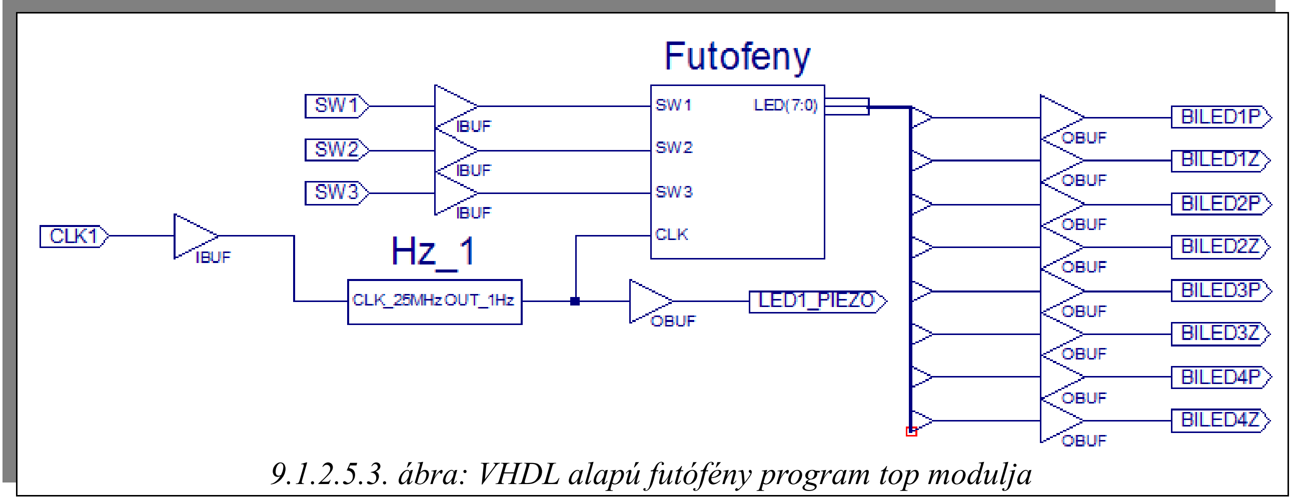
```
        process(SW3, LED_position)                    -- LED színének és helyének meghatározása
        begin
                if SW3 = '1' then                             -- zöld szín
                        case LED_position is
                                when "1000" => LED <= "10000000";        -- 1. zöld led
                                when "0100" => LED <= "00100000";        -- 2. zöld led
                                when "0010" => LED <= "00001000";        -- 3. zöld led
                                when "0001" => LED <= "00000010";        -- 4. zöld led
                                when others =>                -- más kombináció nem lehet
                        end case;
                else                                                  -- piros szín
                        case LED_position is
                                when "1000" => LED <= "01000000";        -- 1. piros led
                                when "0100" => LED <= "00010000";        -- 2. piros led
                                when "0010" => LED <= "00000100";        -- 3. piros led
                                when "0001" => LED <= "00000001";        -- 4. piros led
                                when others =>                -- más kombináció nem lehet
                        end case;
                end if;
        end process;

        SW12 <= SW1 & SW2;                                    -- kapcsolók összefűzése
end Behavioral;
```

A modulnak 4 bemenete (3 kapcsoló, egy órajel) és egy 8 bites buszos kimenete (LED-ek) van. Az utolsó előtti sorban láthatjuk az SW12 jel előállítását. Az SW1, SW2 bemeneteket egy kétbites buszos jelre fűztük fel a könnyebb kezelhetőség végett. A VHDL modul 2 folyamatot tartalmaz, amelyek egymással és a kapcsolók felfűzésével párhuzamosan futnak.

Az első *process* végzi el a kapcsolók beállításának megfelelően az órajelre adott jel alapján a léptetést. Egy négy bites jelet állít elő, amelyik megmondja, hogy melyik LED-nek kell az adott pillanatban világítania.

A második folyamat adja meg az első *process* LED_position jele és az SW3 bemenet alapján a 8 bites LED kimenet értékét. A két folyamatot két külön alkatrészben is megalkothattuk volna és összeköthettük volna őket egy felsőbb modulban – a későbbiekben erre is látunk majd példát. Az egyes VHDL modulokat addig érdemes bonyolítani, amíg egyrészt átláthatóak maradnak, másrészt csak szorosan összetartozó folyamatokat szerepeltetünk egy kódban.

Természetesen az itt megadott megoldáson kívül még számos, akár teljes mértékben eltérő megoldás is adható az adott problémára.

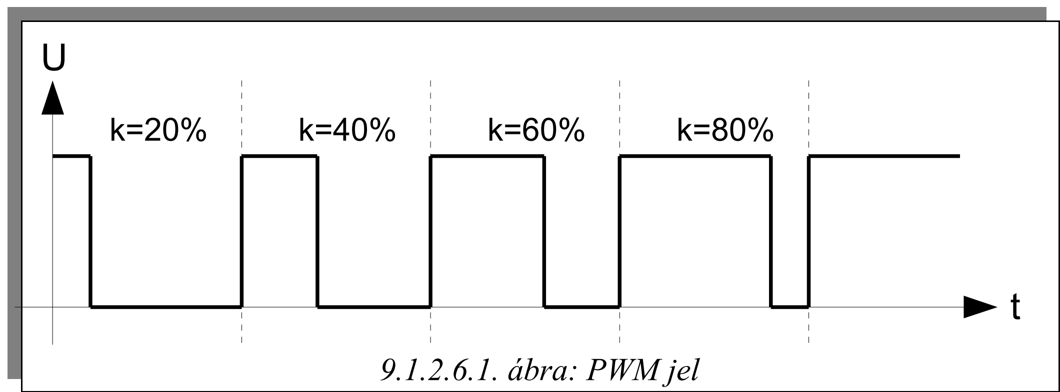


9.1.2.5.3. ábra: VHDL alapú futófény program top modulja

A Hz_1 jelzésű alkatrész a villogó programban megismert VHDL modul (112. oldal).

9.1.2.6. PWM jel előállítása (fényerőszabályozás)

Egy PWM[101] jelet mutat a 9.1.2.6.1. ábra. A PWM jel előállításához ismernünk kell a frekvenciáját és a kitöltés finomságának mértékét. Ahány részre szeretnénk felosztani a négyszögjelet, annyiszor kell megszoroznunk az alapfrekvenciánkat az órajelünk kiszámításához. Ha pl. egy 1kHz-es PWM jelre van szükségünk amit 0-100%-os kitöltéssel szeretnénk üzemeltetni 1%-os pontossággal, akkor ehhez egy 100kHz-es órajelre lesz szükségünk.

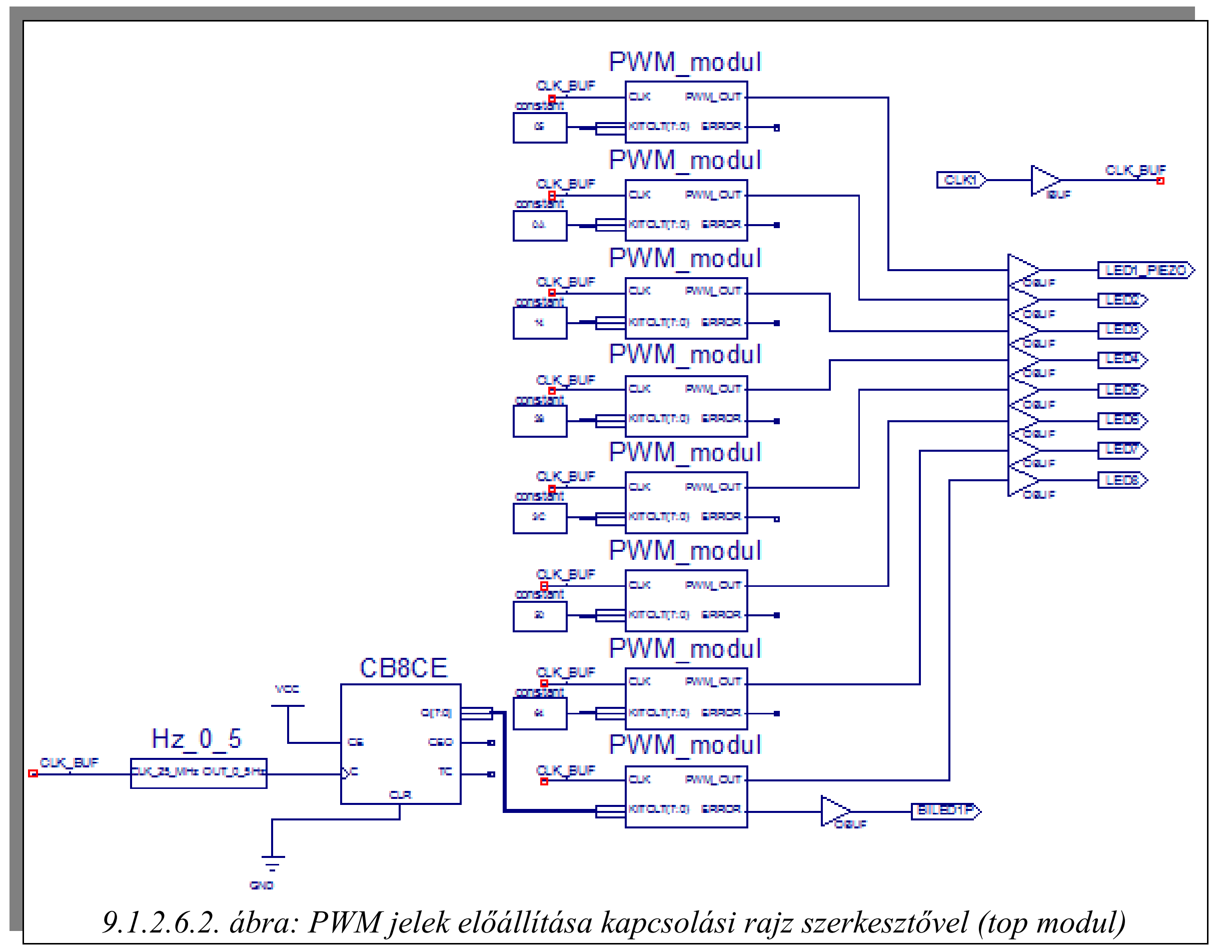


9.1.2.6.1. ábra: PWM jel

Gyakran szükségünk van egy feladatban a digitális technikában alkalmazott két diszkrét feszültségértéken (0V-5V; 0-3,3V; stb.) kívül egyéb feszültségértékek előállítására, illetve ezen feszültségek analóg módon történő változtatására (fényerő szabályozás, DC motor vezérlés). Erre találták ki a D/A átalakítót. A legtöbb esetben azonban igyekszünk külső hardverelem nélkül megoldani a problémát, és ehhez nyújt segítséget a PWM jel. Tételezzük fel, hogy a 9.1.2.6.1. ábrán látható jelnek a magas szintje 5V. Ekkor az első periódust átlagolva kapunk, a második periódus esetén feszültség, a harmadikban mérhető. Az átlagolást el tudjuk végezni egy integráló tag (RC szűrő) segítségével, ill. vannak egyes alkatrészek, amelyek tehetetlenségüknél fogva átlagoló tulajdonsággal rendelkeznek (motor). De nem szükséges pl. egy LED esetében sem átlagolni, elvégzi ezt a feladatot a szemünk is. Egy PWM jel esetében nagyon fontos a helyes frekvencia megválasztása. Fényerő szabályozás esetén minimum 25 frame/sec szükséges az állóképhez. Ez azonban nagyon bántja a szemet, bár látni nem látjuk, érzékeljük a villogást és hamar megfájdul a fejünk. Célszerű ezért növelni a frekvenciát minimum 100Hz-re (gondoljunk csak a mai TV-kre). Nem szabad a frekvenciát minden határon túl növelni hiszen egyrészt minél nagyobb frekvencián dolgozunk annál nagyobb bizonytalanságot viszünk a rendszerbe, másrészt a hardverelemeknek is van egy minimális be- és kikapcsolási idejük.

A 9.1.2.6.2. ábrán egy 100Hz alapfrekvenciájú PWM modulokat tartalmazó *schematic* alapú leírást figyelhetünk meg. A PWM jelet 1% pontossággal tudjuk beállítani 0÷100%-ig.

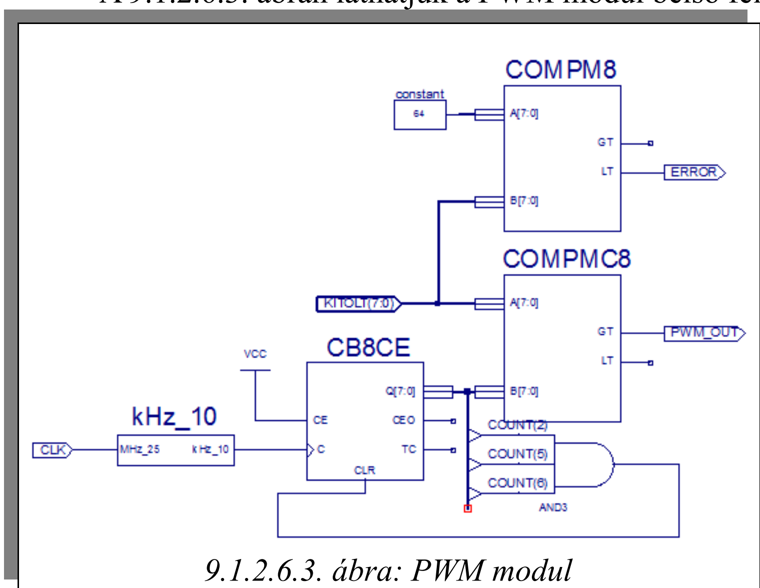
101A szakmában a PWM rövidítés a *Pulse Width Modulation* – impulzusszélesség modulációt takarja. Sajnos sokszor szokás helytelenül PDM-nek nevezni (*Pulse Duration Modulation* – impulzushossz moduláció). A PDM rövidítés azonban az elektronikában inkább a *Pulse Density Modulation* – impulzussűrűség modulációt takarja, amelynél a nagyon kis, azonos szélességű impulzusok segítségével alkotjuk meg az analóg jelet.



9.1.2.6.2. ábra: PWM jelek előállítása kapcsolási rajz szerkesztővel (top modul)

A kapcsolás tartalmaz 8 különálló PWM modult, amik ki vannak vezetve LED-ekre. A PWM modulon kívül megfigyelhetjük a bal alsó sarokban a fél Hz-es órajel előállító áramkört, ami egy számláló *clockját* biztosítja. A számláló egy lassan változó 0÷255 értéket állít elő a kimenetén, amit egy PWM modul használ fel kitöltési tényezőnek. Így tudjuk elérni a változó fényerőt.

A 9.1.2.6.3. ábrán láthatjuk a PWM modul belső felépítését.

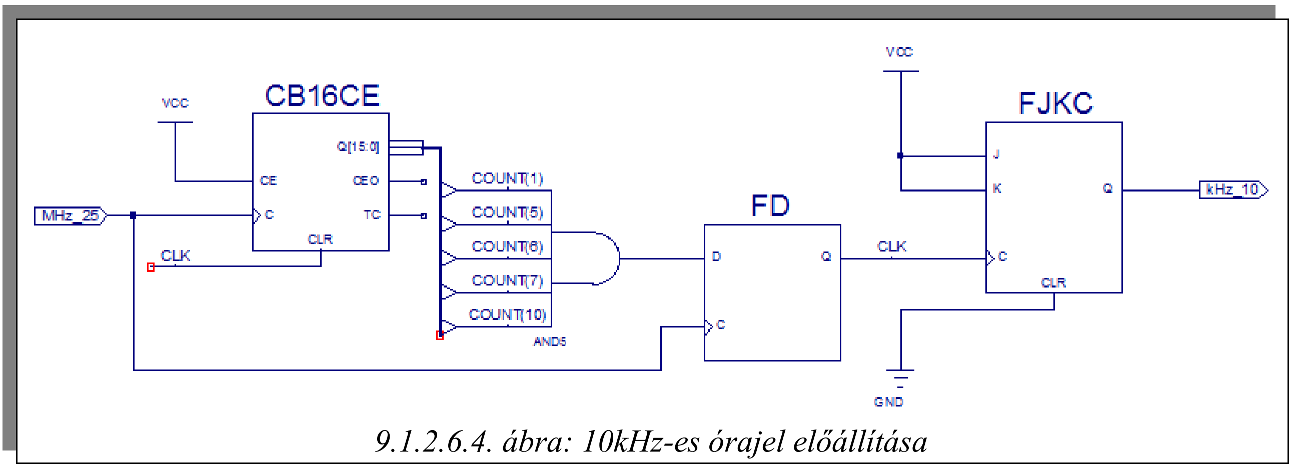


9.1.2.6.3. ábra: PWM modul

A modul tartalmaz egy 10kHz-es órajelet előállító alkatrészt. Egy számlálót üzemeltetünk erről a frekvenciáról. A számláló kimenetét egy komparátor segítségével összehasonlítjuk a *KITOLT* 8 bites bemenettel. A számlálót 100-ig[102] számoltatjuk, utána töröljük és a számlálás nulláról indul újra. Ameddig a *KITOLT* értéke nagyobb a számláló aktuális értékénél (*GT*), addig a kimenet értéke logikai „1”, különben „0”. Egy másik komparátor gondoskodik a hibajel előállításáról ha a kitöltési tényezőnek 100%-nál nagyobb értéket szeretnénk megadni.

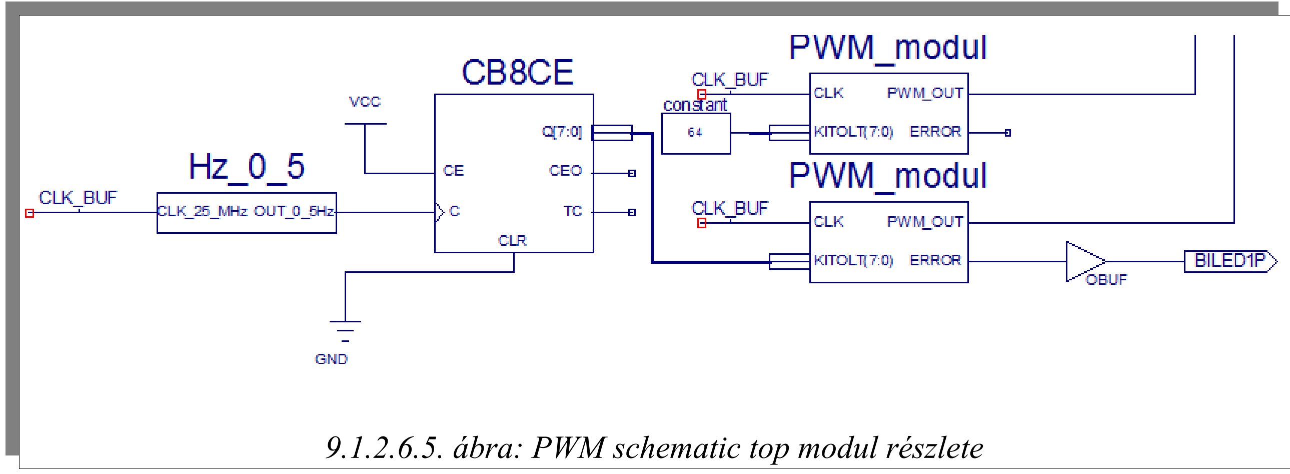
[102] 100%, minden egyes lépés 1%-ot jelent. Így a *KITOLT* bemeneten 0÷100 állíthatunk be értéket.

A *kHz_10*-es alkatrész a villogó programnál megismert modulból lett megalkotva. A 9.1.2.6.4. ábrán látható az alkatrész belső felépítése. Miután a 10kHz-es jel előállításához a 25MHz-es jelet 2500-al kell osztanunk, a szám amit kikapuzunk ennek a fele, mert egy perióduson belül kétszer invertálunk. Az 1250 bináris kódja a „10011100010”, ennek biztosításához most elegendő számunkra egy 16 bites számláló is, nem kell bővítenünk.



9.1.2.6.4. ábra: 10kHz-es órajel előállítása

A 10kHz-es órajel előállító áramkörnél most csomópont segítségével alakítottuk ki a két órajelbemenet összeköttetését. Ez a megoldás teljesen egyenértékű az azonos netnevek használatával.



9.1.2.6.5. ábra: PWM schematic top modul részlete

A 9.1.2.6.5. ábrán figyelhetjük meg a PWM modulok alkalmazásának lehetőségeit A felső modul kitöltési tényezőjét egy konstans elemmel állítottuk be 100%-ra. Az alsó modul kitöltési tényezőjét egy számláló változtatja folyamatosan 0 és 255 között. Amikor átlépi a 100-at az *ERROR* jel által meghajtott kétszínű LED piros fénnyel világít. Amikor a számláló átfordul a LED kialszik. A 8 LED-en sorban a következő kitöltési tényezőket figyelhetjük meg: 5%, 10%, 20%, 40%, 60%, 80%, 100%, 0÷100%. Megfigyelhetjük, hogy a fényerőérzet mennyire nem lineáris skála mentén valósul meg.

A következő VHDL kód és a 9.1.2.6.6. ábrán látható top modul ugyanezt a feladatot hivatott végrehajtani.

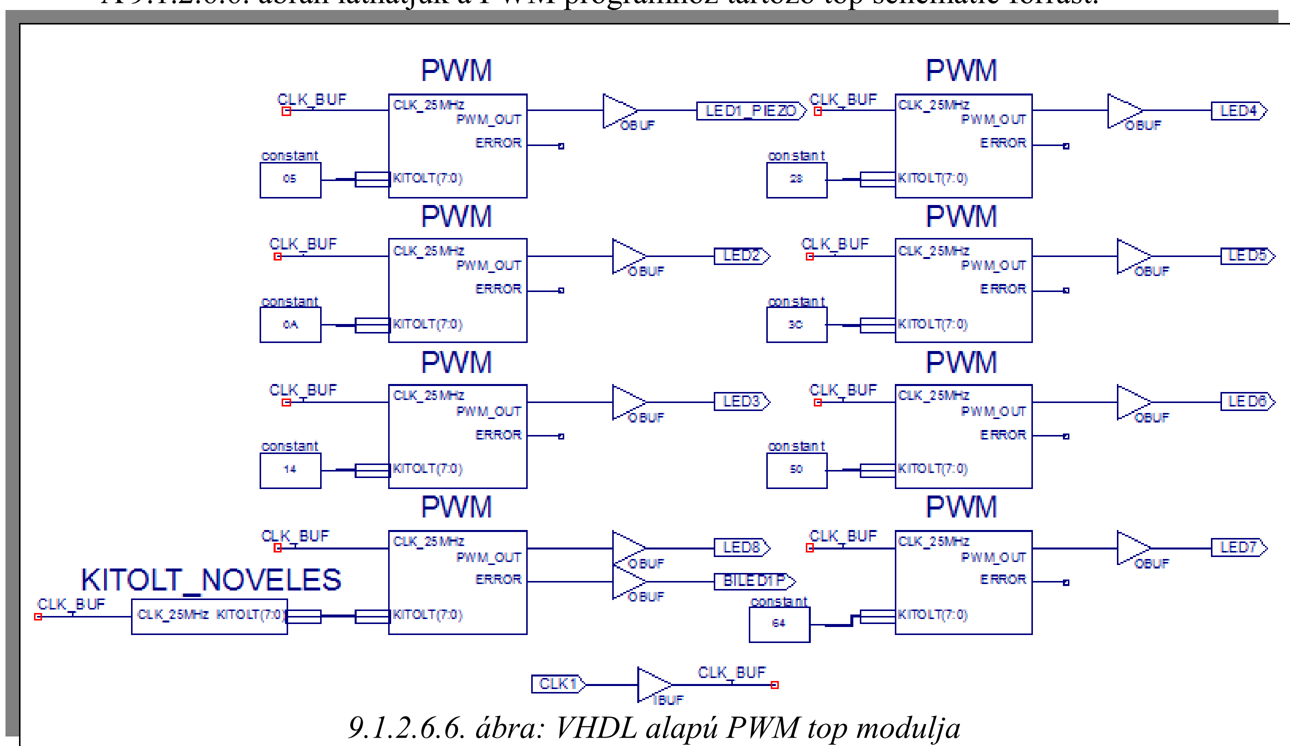
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity PWM is
   Port ( CLK_25MHz : in  STD_LOGIC;
         KITOLT : in  STD_LOGIC_VECTOR(7 downto 0);
         PWM_OUT : out  STD_LOGIC;
         ERROR : out  STD_LOGIC);
end PWM;
architecture Behavioral of PWM is
         signal    signal_10kHz: STD_LOGIC;
         signal    szam1: unsigned (10 downto 0):=(others => '0');
         signal    szam2: unsigned (7 downto 0):=(others => '0');
         signal    kitolt_temp: STD_LOGIC_VECTOR(7 downto 0);
begin
process(CLK_25MHz)                -- 10kHz előállítása
         begin
                  if falling_edge(CLK_25MHz) then -- ha a 25MHz-e órajelen lefutó él érkezik
                  szam1 <= szam1 +1;                      -- a számot növeljük eggyel
                           if szam1 = 1250 then           -- ha elértük a 1250 értéket
                                    signal_10kHz <= not signal_10kHz;        -- akkor a segédjelet invertáljuk
                                    szam1 <= (others => '0');           -- és a számot töröljük
                           end if;
                  end if;
         end process;
process(signal_10kHz)     -- PWM rész
         begin
                  if falling_edge(signal_10kHz) then -- ha a 10kHz-es órajelen lefutó él érkezik
                  szam2 <= szam2 +1;                      -- a számot növeljük eggyel
                           if szam2 > 100 then            -- ha elértük a 100 értéket
                                    szam2 <= (others => '0');  -- a számot töröljük (100%)
                           end if;
                           if szam2 > unsigned(KITOLT) then           -- ha a szám nagyobb
                           PWM_OUT <= '0';                            -- a kitoltési tényezőnél, akkor PWM: 0
                           else
                           PWM_OUT <= '1';                            -- különben a PWM kimenet: 1
                           end if;
                  end if;
                  if unsigned(KITOLT) > 100 then    -- ha a kitöltési tényező nagyobb 100%-nál
                  ERROR <= '1';                           -- akkor hibajelzés
                  else
                  ERROR <= '0';                           -- különben nincs hibajelzés
                  end if;
         end process;
end Behavioral;
```

A kódban két folyamatot figyelhetünk meg. Az első process hozza létre a PWM-hez szükséges 10kHz-es jelet. A működése megegyezik a villogó programnál ismertetettel (112. oldal). A második process a 10KHz-es frekvenciát felhasználva növel egy értéket (szam2), amennyiben ez az érték eléri a 100-at (100%) akkor törli és a növelés elölről kezdődik. Ezt a számot összehasonlítjuk a KITOLT 8 bites bemenetre érkező kitöltési tényezővel. Amennyiben a szám2 a nagyobb érték, akkor a kimenet '0', ellenkező esetben '1' értéket vesz fel. Egy külön feltételértékelés végzi annak ellenőrzését kitöltési tényező nagyobb-e 100-nál, ha igen akkor az ERROR kimenet '1', ha nem akkor '0' értéket vesz fel. Miután a KITOLT bemenetet STD_LOGIC_VECTOR-ként adtuk meg szükségünk van konverzió használatára a műveletvégzéshez. Ezt a következőképpen tudjuk elvégezni: unsigned(SLV). Természetesen a konverzió oda-vissza alkalmazható a különböző típusok között (pl. std_logic_vector(unsigned)).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity KITOLT_NOVELES is
   Port ( CLK_25MHz : in  STD_LOGIC;
          KITOLT : out  STD_LOGIC_VECTOR(7 downto 0)
                                    );
end KITOLT_NOVELES;
architecture Behavioral of KITOLT_NOVELES is
        signal    szam1: unsigned (24 downto 0):=(others => '0');
        signal    szam2: unsigned (7 downto 0):=(others => '0');
begin
process(CLK_25MHz)                  -- 1Hz-el való növelgetés
        begin
                if falling_edge(CLK_25MHz) then -- ha a 25MHz-es órajelen lefutó él érkezik
                szam1 <= szam1 +1;                      -- a szám1-et növeljük eggyel
                        if szam1 = 25000000 then -- ha elértük a 25000000 értéket
                                szam2 <= szam2+1;           -- akkor a szám2-őt növeljük
                                szam1 <= (others => '0');  -- és a szám1-et töröljük
                        end if;
                end if;
        end process;
        KITOLT <= std_logic_vector(szam2);
end Behavioral;
```

A kitöltési tényező folyamatos másodpercenkénti növeléséhez szükségünk van a fenti kódrészletre. A leírás hasonlít az 1Hz-es frekvenciaosztásra, azonban itt nem invertálnunk kell, hanem egy számértéket (szam2) növelni ha letelt az 1 másodperc. Természetesen most miután nem két invertálás szerepel egy perióduson belül, hanem egy szám eggyel történő növelése nem kell a kiszámolt számértéket (25000000) kettővel osztanunk. Ahhoz, hogy a KITOLT kimenetet STD_LOGIC_VECTOR-ként adhassuk meg használnunk kell egy segédjelet, és el kell végeznünk a köztük lévő konverziót. Ezt figyelhetjük meg az egyidejű résznél.

A 9.1.2.6.6. ábrán láthatjuk a PWM programhoz tartozó top schematic forrást.

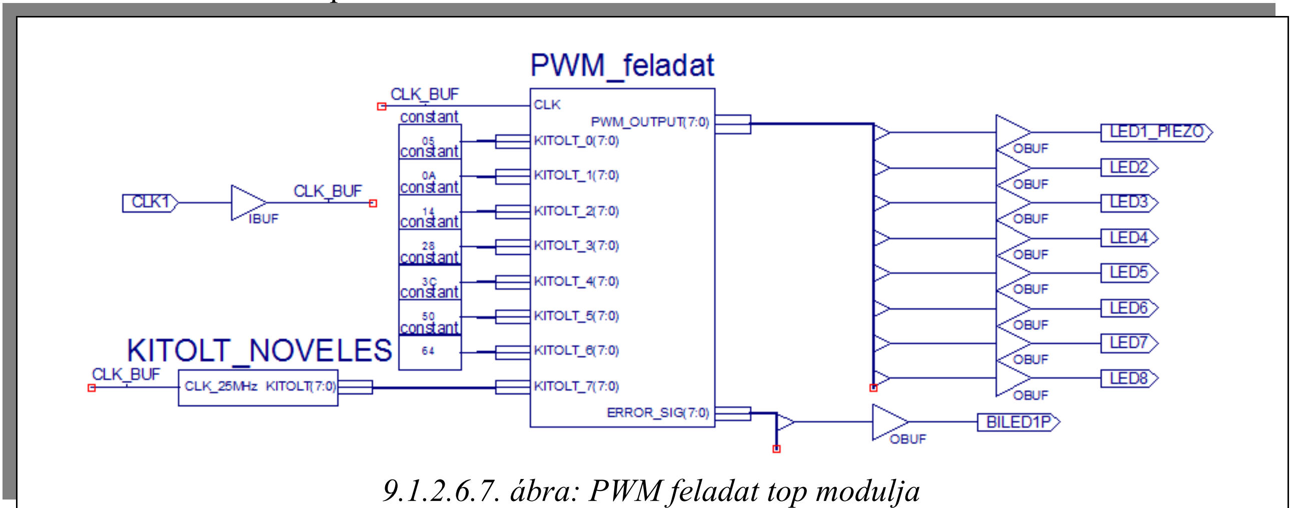


9.1.2.6.6. ábra: VHDL alapú PWM top modulja

Ha megvizsgáljuk a 9.1.2.6.6. ábrát, akkor azt vesszük észre, hogy kissé zsúfolt az elrendezés. A top modulon célszerű csak egy-két komponenst szerepeltetni valamint a be- és kimeneteket. Az alábbi kód ebben nyújt segítséget.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity PWM_feladat is
   Port ( CLK : in  STD_LOGIC;
          KITOLT_0 : in  STD_LOGIC_VECTOR(7 downto 0);
          KITOLT_1 : in  STD_LOGIC_VECTOR(7 downto 0);
          KITOLT_2 : in  STD_LOGIC_VECTOR(7 downto 0);
          KITOLT_3 : in  STD_LOGIC_VECTOR(7 downto 0);
          KITOLT_4 : in  STD_LOGIC_VECTOR(7 downto 0);
          KITOLT_5 : in  STD_LOGIC_VECTOR(7 downto 0);
          KITOLT_6 : in  STD_LOGIC_VECTOR(7 downto 0);
          KITOLT_7 : in  STD_LOGIC_VECTOR(7 downto 0);
          PWM_OUTPUT : out  STD_LOGIC_VECTOR (7 downto 0);
          ERROR_SIG : out  STD_LOGIC_VECTOR (7 downto 0));
end PWM_feladat;
architecture Behavioral of PWM_feladat is
component PWM
   Port ( CLK_25MHz : in  STD_LOGIC;
          KITOLT : in  STD_LOGIC_VECTOR(7 downto 0);
          PWM_OUT : out  STD_LOGIC;
          ERROR : out  STD_LOGIC);
end component;
            begin
PWM1: PWM port map (CLK, KITOLT_0, PWM_OUTPUT(0), ERROR_SIG(0));
PWM2: PWM port map (CLK, KITOLT_1, PWM_OUTPUT(1), ERROR_SIG(1));
PWM3: PWM port map (CLK, KITOLT_2, PWM_OUTPUT(2), ERROR_SIG(2));
PWM4: PWM port map (CLK, KITOLT_3, PWM_OUTPUT(3), ERROR_SIG(3));
PWM5: PWM port map (CLK, KITOLT_4, PWM_OUTPUT(4), ERROR_SIG(4));
PWM6: PWM port map (CLK, KITOLT_5, PWM_OUTPUT(5), ERROR_SIG(5));
PWM7: PWM port map (CLK, KITOLT_6, PWM_OUTPUT(6), ERROR_SIG(6));
PWM8: PWM port map (CLK, KITOLT_7, PWM_OUTPUT(7), ERROR_SIG(7));

            end Behavioral;
```

A fenti kód felhasználja a PWM modulunkat komponensként (részletekért l. 8.7. fejezet). Ügyeljünk rá, hogy a komponens és az őt leíró entitás portleírása megegyezzen! A kódhoz a 9.1.2.6.7. ábrán látható top modul tartozik.



9.1.2.6.7. ábra: PWM feladat top modulja

Természetesen ezt a top modult is helyettesíthettük volna egy VHDL kóddal, amiben komponensként használjuk a most megismert PWM_feladat és a már előzőekben ismertetett KITOLT_NOVELES entitásokat, azonban úgy gondoljuk a kapcsolási rajz alapú leírás a legfelső forrásban. A komponensek használata nagyon megkönnyíti a későbbi fejlesztéseket. Amikor egy feladatot próbálunk a későbbiekben végrehajtani, alkalmazzunk minél általánosabb leírást nyújtó entitásokat, hogy fel tudjuk használni őket a későbbiekben is.

9.1.2.7. Knight Rider futófény

A Knight Rider futófény programban egyesítenünk kell a futófény programnál és a fényerő szabályozásnál megtanult ismereteinket. A következő linken láthatjuk, hogy milyen hatást szeretnénk elérni a LED-eken: http://www.youtube.com/watch?v=p5w7sbx26jQ

Gyakori hiba, hogy Knight Rider futófényként ismertetnek olyan áramköröket, amelyeken azt láthatjuk, hogy mindig csak egy LED világít, aminek a fényereje pozíciófüggő. Fontos azonban tisztáznunk, hogy bár a feladat megoldása hasonló fogásokat kíván a programozótól, ha megnézzük a videót, akkor láthatjuk, hogy a *Knight Industries Two Thousand* (KITT) első szkennerének fénysorában minden LED világít különböző fényerővel és a maximális fényerő megfelelő pozicionálásával érhető el a futófény hatás. Az egyes LED-ek kitöltési tényezőit megválasztani próbálgatás útján célszerű. Láthattuk, hogy nem lineáris karakterisztika mentén történik a megvilágítás[103]. Az egyszerűség kedvéért itt most csak a VHDL alapú megoldást ismertetjük, fontosnak tartjuk azonban megjegyezni, hogy a feladat végrehajtható kapcsolási rajz szerkesztő segítségével is.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity Knight_Rider is
   Port ( CLK_IN : in  STD_LOGIC;
          LED_OUT: out STD_LOGIC_VECTOR(7 downto 0));
end Knight_Rider;
architecture Behavioral of Knight_Rider is
          signal LED_helyzet: STD_LOGIC_VECTOR (7 downto 0);
          signal KITOLT_0: STD_LOGIC_VECTOR (7 downto 0);
          signal KITOLT_1: STD_LOGIC_VECTOR (7 downto 0);
          signal KITOLT_2: STD_LOGIC_VECTOR (7 downto 0);
          signal KITOLT_3: STD_LOGIC_VECTOR (7 downto 0);
          signal KITOLT_4: STD_LOGIC_VECTOR (7 downto 0);
          signal KITOLT_5: STD_LOGIC_VECTOR (7 downto 0);
          signal KITOLT_6: STD_LOGIC_VECTOR (7 downto 0);
          signal KITOLT_7: STD_LOGIC_VECTOR (7 downto 0);
component PWM
   Port ( CLK_25MHz : in  STD_LOGIC;
        KITOLT : in  STD_LOGIC_VECTOR(7 downto 0);
        PWM_OUT : out  STD_LOGIC;
        ERROR : out  STD_LOGIC);
end component;
component Futofeny is
   Port ( CLK : in  STD_LOGIC;
           LED : out  STD_LOGIC_VECTOR (7 downto 0));
end component;
```

103Mindemellett érdemes megjegyezni, hogy a fényérzetet jelentősen befolyásolja a választott LED típusa és az előtétellenállás is.

```
        begin
POSITION: Futofeny port map (CLK_IN, LED_helyzet);
PWM1: PWM port map (CLK_IN, KITOLT_0, LED_OUT(0));
PWM2: PWM port map (CLK_IN, KITOLT_1, LED_OUT(1));
PWM3: PWM port map (CLK_IN, KITOLT_2, LED_OUT(2));
PWM4: PWM port map (CLK_IN, KITOLT_3, LED_OUT(3));
PWM5: PWM port map (CLK_IN, KITOLT_4, LED_OUT(4));
PWM6: PWM port map (CLK_IN, KITOLT_5, LED_OUT(5));
PWM7: PWM port map (CLK_IN, KITOLT_6, LED_OUT(6));
PWM8: PWM port map (CLK_IN, KITOLT_7, LED_OUT(7));
        process(LED_helyzet)
        begin
                case LED_helyzet is
                                when "10000000" =>
                                        KITOLT_0 <= X"64" ;     -- 100%
                                        KITOLT_1 <= X"28" ;     -- 40%
                                        KITOLT_2 <= X"28" ;     -- 80%
                                        KITOLT_3 <= X"14" ;     -- 20%
                                        KITOLT_4 <= X"0A" ;     -- 10%
                                        KITOLT_5 <= X"05" ;     -- 5%
                                        KITOLT_6 <= X"02" ;     -- 2%
                                        KITOLT_7 <= X"01" ;     -- 1%
                                when "01000000" =>
                                        KITOLT_0 <= X"28" ;     -- 40%
                                        KITOLT_1 <= X"64" ;     -- 100%
                                        KITOLT_2 <= X"28" ;     -- 40%
                                        KITOLT_3 <= X"14" ;     -- 20%
                                        KITOLT_4 <= X"0A" ;     -- 10%
                                        KITOLT_5 <= X"05" ;     -- 5%
                                        KITOLT_6 <= X"02" ;     -- 2%
                                        KITOLT_7 <= X"01" ;     -- 1%
                                when "00100000" =>
                                        KITOLT_0 <= X"14" ;     -- 20%
                                        KITOLT_1 <= X"28" ;     -- 40%
                                        KITOLT_2 <= X"64" ;     -- 100%
                                        KITOLT_3 <= X"28" ;     -- 40%
                                        KITOLT_4 <= X"14" ;     -- 20%
                                        KITOLT_5 <= X"0A" ;     -- 10%
                                        KITOLT_6 <= X"05" ;     -- 5%
                                        KITOLT_7 <= X"02" ;     -- 2%
                                when "00010000" =>
                                        KITOLT_0 <= X"0A" ;     -- 10%
                                        KITOLT_1 <= X"14" ;     -- 20%
                                        KITOLT_2 <= X"28" ;     -- 40%
                                        KITOLT_3 <= X"64" ;     -- 100%
                                        KITOLT_4 <= X"28" ;     -- 40%
                                        KITOLT_5 <= X"14" ;     -- 20%
                                        KITOLT_6 <= X"0A" ;     -- 10%
                                        KITOLT_7 <= X"05" ;     -- 5%
                                when "00001000" =>
                                        KITOLT_0 <= X"05" ;     -- 5%
                                        KITOLT_1 <= X"0A" ;     -- 10%
                                        KITOLT_2 <= X"14" ;     -- 20%
                                        KITOLT_3 <= X"28" ;     -- 40%
                                        KITOLT_4 <= X"64" ;     -- 100%
                                        KITOLT_5 <= X"28" ;     -- 40%
                                        KITOLT_6 <= X"14" ;     -- 20%
```

```
                                        KITOLT_7 <= X"0A" ;     -- 10%
                              when "00000100" =>
                                        KITOLT_0 <= X"02" ;     -- 2%
                                        KITOLT_1 <= X"05" ;     -- 5%
                                        KITOLT_2 <= X"0A" ;     -- 10%
                                        KITOLT_3 <= X"14" ;     -- 20%
                                        KITOLT_4 <= X"28" ;     -- 40%
                                        KITOLT_5 <= X"64" ;     -- 100%
                                        KITOLT_6 <= X"28" ;     -- 40%
                                        KITOLT_7 <= X"14" ;     -- 20%
                              when "00000010" =>
                                        KITOLT_0 <= X"01" ;     -- 1%
                                        KITOLT_1 <= X"02" ;     -- 2%
                                        KITOLT_2 <= X"05" ;     -- 5%
                                        KITOLT_3 <= X"0A" ;     -- 10%
                                        KITOLT_4 <= X"14" ;     -- 20%
                                        KITOLT_5 <= X"28" ;     -- 40%
                                        KITOLT_6 <= X"64" ;     -- 100%
                                        KITOLT_7 <= X"28" ;     -- 40%
                              when "00000001" =>
                                        KITOLT_0 <= X"01" ;     -- 1%
                                        KITOLT_1 <= X"02" ;     -- 2%
                                        KITOLT_2 <= X"05" ;     -- 5%
                                        KITOLT_3 <= X"0A" ;     -- 10%
                                        KITOLT_4 <= X"14" ;     -- 20%
                                        KITOLT_5 <= X"28" ;     -- 40%
                                        KITOLT_6 <= X"28" ;     -- 40%
                                        KITOLT_7 <= X"64" ;     -- 100%
                              when others => -- semmi
                    end case;
          end process;
end Behavioral;
```

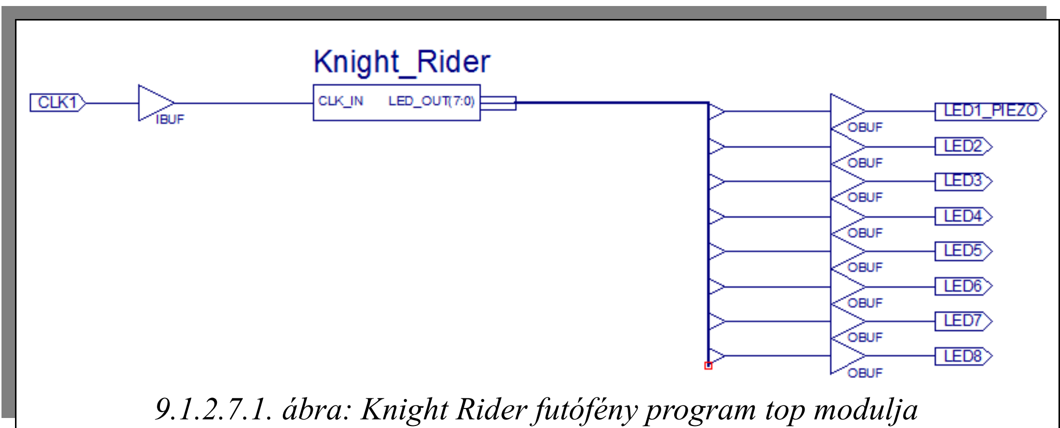
A fenti kódrészlet két komponenst alkalmaz. Az egyik a már megismert PWM modul, a másik a futófény programból lett átalakítva és alább látható. A kódban található még egy folyamat, amely a futófény komponens által megadott LED pozíció alapján beállítja a megfelelő fényerőt egy case utasítás segítségével.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity Futofeny is
   Port ( CLK : in  STD_LOGIC;
            LED : out  STD_LOGIC_VECTOR (7 downto 0));
end Futofeny;
architecture Behavioral of Futofeny is
          signal LED_position: unsigned (7 downto 0):="10000000";
          signal oda_vissza: bit :='0';
          signal CLK_int : STD_LOGIC;
component idozito is
   Port ( CLK_25MHz : in  STD_LOGIC;    -- 25Mhz-es órajel
        OUT_10Hz : buffer  STD_LOGIC);  -- 10Hz-es kimenet
end component;
begin
IDOALAP: idozito port map(CLK,CLK_int);
          -- a futófény időalapjának leképzése a 25MHz-es órajelből
```

```
	process(CLK_int)		-- LED pozicionálás
	begin
		if falling_edge(CLK_int) then			-- CLK lefutó élére indul a folyamat
			if oda_vissza = '1' then
				if LED_position = "10000000" then		-- ha már átfordulna
					LED_position <= LED_position ror 1;		-- visszaforgatjuk
						oda_vissza <= '0';			-- és másik irányt kérünk
				else						-- különben
					LED_position <= LED_position rol 1;
								-- a LED-eken egy forgatás balra
				end if;
			else
				if LED_position = "00000001" then		-- ha már átfordulna
					LED_position <= LED_position rol 1;		-- visszaforgatjuk
						oda_vissza <= '1';			-- és másik irányt kérünk
				else						-- különben
					LED_position <= LED_position ror 1;
								-- a LED-eken egy forgatás jobbra
				end if;
			end if;
		end if;
	end process;
	LED <= std_logic_vector(LED_position);
end Behavioral;
```

Az eredeti futófény programból csak az oda-vissza ágat tartottuk meg. Az időalapot most a VHDL kódon belüli idozito komponens biztosítja. Ez egy 10Hz-es jelet szolgáltat számunkra.

A kódokhoz tartozó top forrás a 9.1.2.7.1. ábrán látható.



9.1.2.7.1. ábra: Knight Rider futófény program top modulja

Mind a top modul, mind az idozíto, a PWM és a futofeny komponens eléggé tömör, azonban a Knight Rider entitás case utasításában található kódrészlet nem túl elegáns. Az alábbi leírás helyettesíti a Knight Rider és a futofeny entitásokat, az idozito és a PWM modul továbbra is komponensként van használva – itt a tömbkezelés talán kicsit szebb kódot eredményez.

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

entity Knight_Rider is
    Port ( CLK_IN : in  STD_LOGIC;
           LED_OUT: out STD_LOGIC_VECTOR(7 downto 0));
end Knight_Rider;

architecture Behavioral of Knight_Rider is
        type tomb is array (0 to 14) of STD_LOGIC_VECTOR (7 downto 0);
        signal KITOLT: tomb:=
                        (
                            X"01" , -- 1%           A kezdő irány
                            X"02" , -- 2%                   |
                            X"03" , -- 3%                   |
                            X"05" , -- 5%                   |
                            X"0A" , -- 10%                  |
                            X"14" , -- 20%                 \  /
                            X"28" , -- 40%                  \/
                            X"64" , -- 100%     közép, ez a bal szélső LED
                            X"28" , -- 40%      2. LED
                            X"14" , -- 20%      3. LED
                            X"0A" , -- 10%      4. LED
                            X"05" , -- 5%       5. LED
                            X"03" , -- 3%       6. LED
                            X"02" , -- 2%       7. LED
                            X"01") ; -- 1%;     8. LED
        signal CLK_int: STD_LOGIC;
        signal oda_vissza: STD_LOGIC:='0';
        signal kitolt_temp: STD_LOGIC_VECTOR(7 downto 0);

component PWM
    Port ( CLK_25MHz : in  STD_LOGIC;
           KITOLT : in  STD_LOGIC_VECTOR(7 downto 0);
           PWM_OUT : out  STD_LOGIC;
           ERROR : out  STD_LOGIC);
end component;

component idozito is
    Port ( CLK_25MHz : in  STD_LOGIC;            -- 25Mhz-es órajel
           OUT_10Hz : buffer  STD_LOGIC);        -- 10Hz-es kimenet
end component;

        begin
IDOALAP: idozito port map(CLK_IN,CLK_int);
                        -- a futófény időalapjának leképzése a 25MHz-es órajelből
PWM1: PWM port map (CLK_IN, KITOLT(7), LED_OUT(0));
PWM2: PWM port map (CLK_IN, KITOLT(8), LED_OUT(1));
PWM3: PWM port map (CLK_IN, KITOLT(9), LED_OUT(2));
PWM4: PWM port map (CLK_IN, KITOLT(10), LED_OUT(3));
PWM5: PWM port map (CLK_IN, KITOLT(11), LED_OUT(4));
PWM6: PWM port map (CLK_IN, KITOLT(12), LED_OUT(5));
PWM7: PWM port map (CLK_IN, KITOLT(13), LED_OUT(6));
PWM8: PWM port map (CLK_IN, KITOLT(14), LED_OUT(7));
```

```
    process(CLK_int)                                        -- LED pozicionálás
    begin
        if falling_edge(CLK_int) then                   -- CLK lefutó élére indul a folyamat
            if oda_vissza = '1' then
                if KITOLT(7) = X"64"    then        -- mielőtt átfordulna, "10000000"
                    oda_vissza <= '0';                          -- irányt váltunk
                else                                            -- különben
                    for I in 0 to 14 loop
                        -- for ciklus a tömbök elemeinek balra forgatásához
                        if I = 0 then
                            kitolt_temp <= KITOLT(I);
                                -- első lépésben segédjelbe az első elem
                        end if;
                            if I = 14 then      -- utolsó lépésben
                                KITOLT(I) <= kitolt_temp;
                                -- segédjel az utolsó elem helyére
                            else        -- minden más lépésben
                                KITOLT(I) <= KITOLT(I+1);
                        -- adott elemszámba az eggyel nagyobb elemszám
                            end if;
                    end loop;
                end if;
            else
                if KITOLT(14) = X"64"   then        -- mielőtt átfordulna, "00000001"
                    oda_vissza <= '1';                  -- irányt váltunk
                else                                    -- különben
                    for I in 14 downto 0 loop
                        -- for ciklus a tömbök elemeinek balra forgatásához
                        if I = 14 then
                            kitolt_temp <= KITOLT(I);
                        -- első lépésben segédjelbe az első elem
                        end if;
                            if I = 0 then       -- utolsó lépésben
                                KITOLT(I) <= kitolt_temp;
                        -- segédjel az utolsó elem helyére
                            else                -- minden más lépésben
                                KITOLT(I) <= KITOLT(I-1);
                        -- adott elemszámba az eggyel kisebb elemszám
                            end if;
                    end loop;
                end if;
            end if;
        end if;
    end process;
    end Behavioral;
```

A fenti kódban deklaráltunk egy 15 elemű tömböt, amiben a LED-ekre kirakott PWM jel kitöltési tényezőit tároljuk. A process ezeket a tömbelemeket cserélgeti az idozito modul által megadott órajel szerint. Nem elegendő egy 8 elemű tömb, ugyanis annak oda-vissza forgatásakor a legkisebb fényerő kerülne a legnagyobb mellé. A folyamatban for ciklus segítségével történik a forgatás (részletekért l. fejezet). Az első for ciklus balra forgat a tömbelemeken, míg a második jobbra. Értelemszerűen az elsőnél 0-ról növeljük a ciklusváltozót, ami meghatározza az elemszámot, míg a másodiknál 14-ről csökkentjük. A folyamat első lefutásakor a második for ciklus kezd el futni. A második for ciklus lefutásának eredményét követhetjük nyomon a következő oldalon lévő táblázatban (a ciklus független az órajeltől). Az első sor a táblázatban a kiindulási érték. A félkövérrel szedett értékek változnak az adott ciklusban.

T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	k_t[104]	ov[105]
1	2	3	5	10	20	40	64	40	20	10	5	3	2	1	0	0
1	2	3	5	10	20	40	64	40	20	10	5	3	2	**2**	**1**	0
1	2	3	5	10	20	40	64	40	20	10	5	3	**3**	2	1	0
1	2	3	5	10	20	40	64	40	20	10	5	**5**	3	2	1	0
1	2	3	5	10	20	40	64	40	20	10	**10**	5	3	2	1	0
1	2	3	5	10	20	40	64	40	20	**20**	10	5	3	2	1	0
1	2	3	5	10	20	40	64	40	**40**	20	10	5	3	2	1	0
1	2	3	5	10	20	40	64	**64**	40	20	10	5	3	2	1	0
1	2	3	5	10	20	40	**40**	64	40	20	10	5	3	2	1	0
1	2	3	5	10	20	**20**	40	64	40	20	10	5	3	2	1	0
1	2	3	5	10	**10**	20	40	64	40	20	10	5	3	2	1	0
1	2	3	5	**5**	10	20	40	64	40	20	10	5	3	2	1	0
1	2	3	**3**	5	10	20	40	64	40	20	10	5	3	2	1	0
1	2	**2**	3	5	10	20	40	64	40	20	10	5	3	2	1	0
1	**1**	2	3	5	10	20	40	64	40	20	10	5	3	2	1	0

Egy teljes oda-vissza forgatást a tömb elemein a következő táblázat tartalmazza (az értékek hexadecimális számokban vannak megadva). A sorok közötti váltáshoz órajel szükséges[106].

T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	k_t	o_v
1	2	3	5	10	20	40	**64**	**40**	**20**	**10**	**5**	**3**	**2**	**1**	0	0
1	1	2	3	5	10	20	**40**	**64**	**40**	**20**	**10**	**5**	**3**	**2**	1	0
2	1	1	2	3	5	10	**20**	**40**	**64**	**40**	**20**	**10**	**5**	**3**	2	0
3	2	1	1	2	3	5	**10**	**20**	**40**	**64**	**40**	**20**	**10**	**5**	3	0
5	3	2	1	1	2	3	**5**	**10**	**20**	**40**	**64**	**40**	**20**	**10**	5	0
10	5	3	2	1	1	2	**3**	**5**	**10**	**20**	**40**	**64**	**40**	**20**	10	0
20	10	5	3	2	1	1	**2**	**3**	**5**	**10**	**20**	**40**	**64**	**40**	20	0
40	20	10	5	3	2	1	**1**	**2**	**3**	**5**	**10**	**20**	**40**	**64**	40	1
20	10	5	3	2	1	1	**2**	**3**	**5**	**10**	**20**	**40**	**64**	**40**	20	1
10	5	3	2	1	1	2	**3**	**5**	**10**	**20**	**40**	**64**	**40**	**20**	10	1
5	3	2	1	1	2	3	**5**	**10**	**20**	**40**	**64**	**40**	**20**	**10**	5	1
3	2	1	1	2	3	5	**10**	**20**	**40**	**64**	**40**	**20**	**10**	**5**	3	1
2	1	1	2	3	5	10	**20**	**40**	**64**	**40**	**20**	**10**	**5**	**3**	2	1
1	1	2	3	5	10	20	**40**	**64**	**40**	**20**	**10**	**5**	**3**	**2**	1	1
1	2	3	5	10	20	40	**64**	**40**	**20**	**10**	**5**	**3**	**2**	**1**	1	0

104kitolt_temp
105oda_vissza
106Az első sor a kezdőérték. A félkövérrel szedett részt láthatjuk a LED-eken.

9.1.3. Hétszegmenses kijelző

A hétszegmenses kijelző lekezelése az utolsó projekt, aminél megadjuk a schematic és a VHDL alapú megoldást is. A következő feladatoknál inkább a VHDL kódot részesítjük előnyben, mindamellett, hogy megtartjuk top forrásként a kapcsolási rajzot.

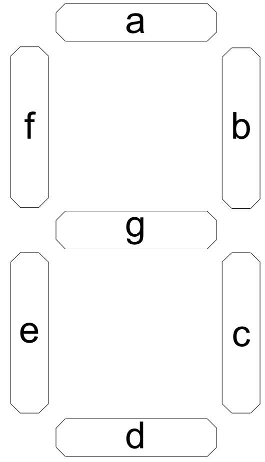
9.1.3.1. Dekódolás és multiplexelés

A hétszegmenses kijelző lekezeléséhez két modulra lesz szükségünk. Szükségünk van egy dekóderre, ami a BCD számból megmondja, hogy a hétszegmenses kijelző mely szegmenseinek kell világítania az adott szám esetén. Mindemellett meg kell oldanunk a kijelzők multiplexelését, amelyek közös adatbuszra vannak felfűzve és a minden kijelzőnél külön vezérelt közös anód segítségével választhatóak ki.

Kezdjük a kapcsolási rajz alapján történő megoldással! Alapfeladatként az 1234 számokat szeretnénk látni a kijelzőkön.

A BCD-hétvonalas dekódoló egy tipikus kombinációs logikai áramkör. Feladata a BCD-kódban érkező jelek továbbítása 7-szegmenses kijelző által értelmezhető jelekre.[107]

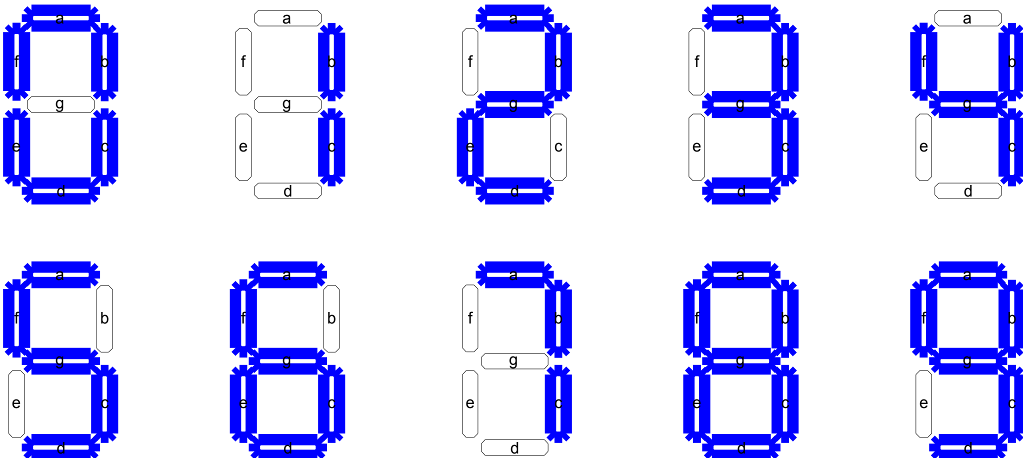
Hétszegmenses kijelző:



Bemenetek (BCD)				Kimenetek (hét-szegmens meghajtás)						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	x	x	x	x	x	x	x
1	0	1	1	x	x	x	x	x	x	x
1	1	0	0	x	x	x	x	x	x	x
1	1	0	1	x	x	x	x	x	x	x
1	1	1	0	x	x	x	x	x	x	x
1	1	1	1	x	x	x	x	x	x	x

107Az itt látható hétszegmenses dekóder logikai kapukkal történő megvalósítását Varga László digitális technika jegyzetéből idézzük.

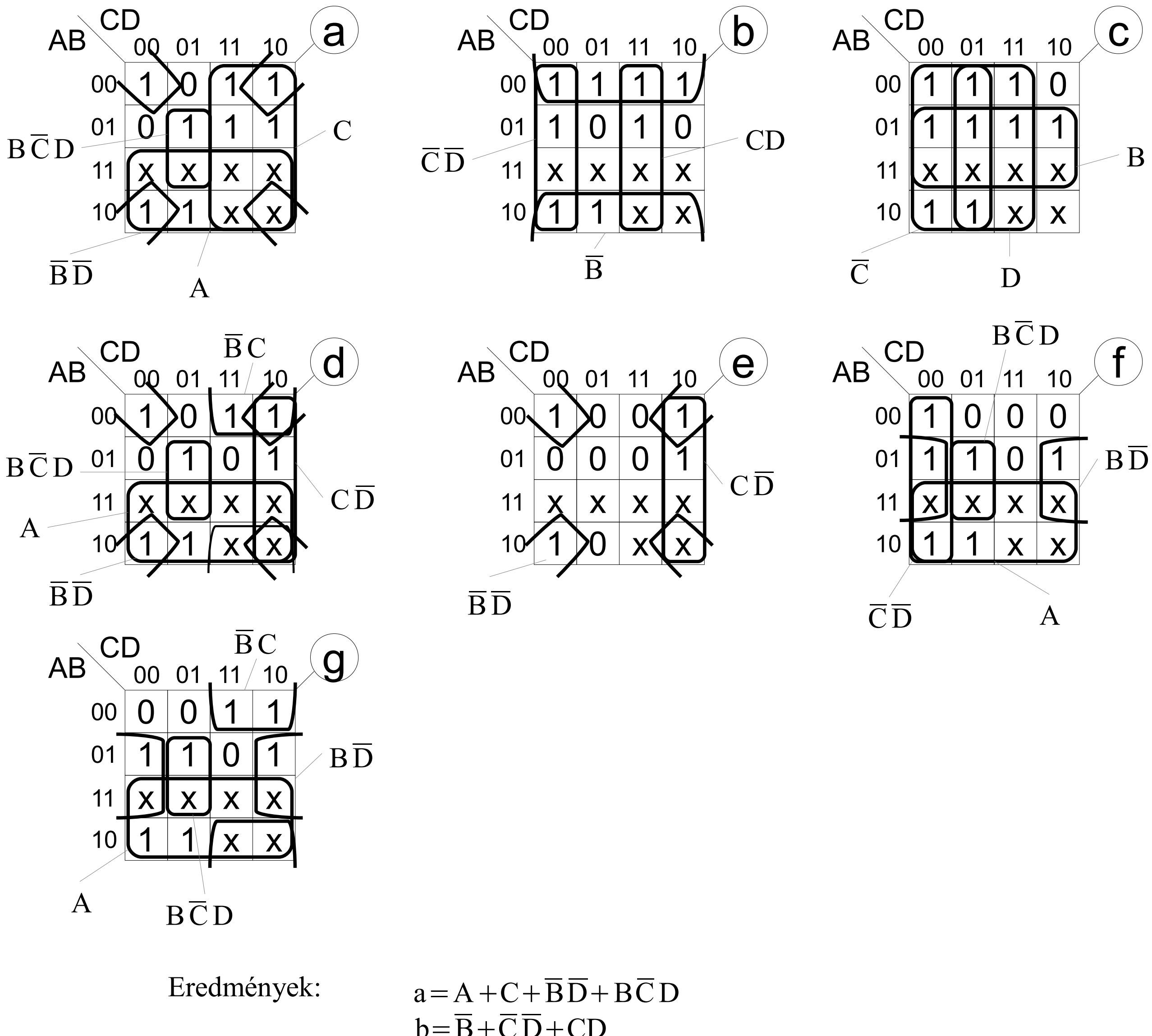
Szám-megjelenítés a kijelzőn:



Látható, hogy pl. a szavazóáramkörrel szemben most több kimenetű hálózatot kell tervezni, egyszerűsíteni. A minimálást Karnaugh tábla segítségével fogjuk elvégezni. Alapesetben egy hálózat akkor lesz a lehető legegyszerűbb, ha a táblában a lehető legnagyobb csoportosításokat végezzük el. Ez egy kimenet esetén igaz is, de most a minimálási eljárást rendszer szinten kell elemeznünk. Előfordulhat, hogy nem a lehető legegyszerűbb alakot választjuk egy adott kimenet esetén – abban az esetben ha egy másik kimenetnél a bonyolultabb kombinációra mindenképpen kell egy kapuhálózat, ezt felhasználhatjuk más kimeneteknél is.

A táblázatban x-el jelölt elemek ún. Don't Care (nem meghatározott) kombinációk. A könnyebb egyszerűsítés végett a határozatlan kimeneti állapotokat a nekünk megfelelő logikai értékre választhatjuk. Ez az eljárás régebben volt gyakorlat, mikor a kapuk ill. bemenetek számát mindenáron csökkenteni igyekeztünk. A kulturált megoldás, ha az x-el jelölt értékek helyére olyan logikai változókat írunk, melyek hatására a kijelző sötét marad. Helyes működés esetén ezek a kombinációk (10; 11; 12; 13; 14; 15) természetesen nem jelenhetnek meg a bemeneten. Most a schematic alapú tervezés során felhasználjuk a határozatlan termeket, míg a VHDL kódban teljes dekódolást végzünk majd.

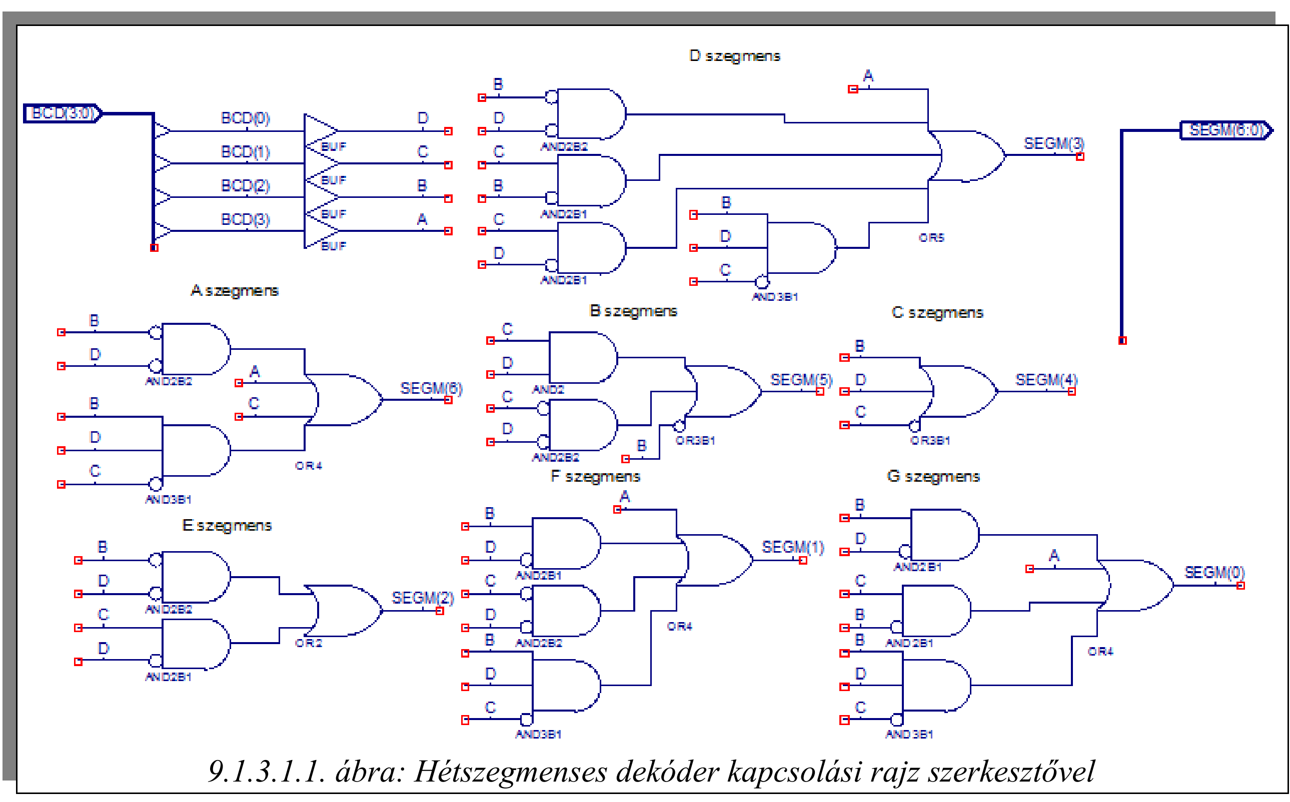
Közös katódos kijelzőre határozatlan termek felhasználásával:



Eredmények:

$$a = A + C + \overline{B}\,\overline{D} + B\overline{C}D$$
$$b = \overline{B} + \overline{C}\,\overline{D} + CD$$
$$c = B + \overline{C} + D$$
$$d = A + \overline{B}\,\overline{D} + \overline{B}C + C\overline{D} + B\overline{C}D$$
$$e = \overline{B}\,\overline{D} + C\overline{D}$$
$$f = A + B\overline{D} + \overline{C}\,\overline{D} + B\overline{C}D$$
$$g = A + B\overline{D} + \overline{B}C + B\overline{C}D$$

Az így kialakított dekódoló a 9.1.3.1.1. ábrán látható. Az egyes szegmensek esetén többször láthatunk azonos logikai kapcsolatokat, amelyeket újra fel lehetne használni, csökkentve ezzel a logikai kapuk számát. A könnyebb áttekinthetőség érdekében ezeket itt nem alkalmazzuk.

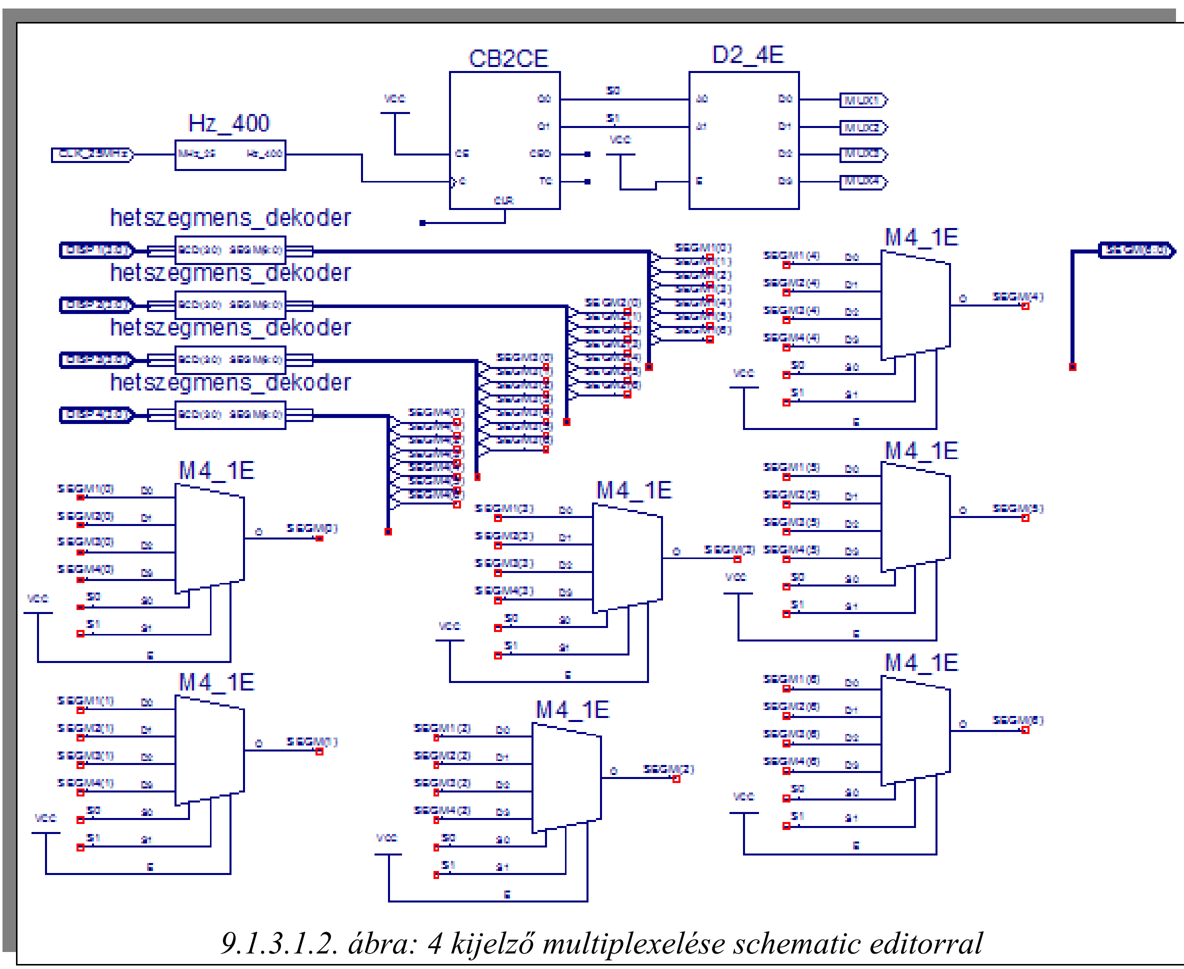


9.1.3.1.1. ábra: Hétszegmenses dekóder kapcsolási rajz szerkesztővel

A dekódoló egy négybites BCD bemenettel és egy 7[108] bites szegmens kimenettel van ellátva.

A multiplexer áramkör belső felépítését a 9.1.3.1.2. ábrán láthatjuk. Miután a fejlesztőkörnyezet rendelkezésünkre bocsájt MUX/DEMUX áramköri elemeket könnyű dolgunk van a tervezés során. A multiplexálás frekvenciáját – miután 4 kijelzőnk van – célszerű 400 Hz-re választani. Ezt a már többször használt órajelosztó áramkörünkkel tudjuk végrehajtani. Az áramkör 4 – már az előbb ismertetett – hétszegmenses dekódert tartalmaz, amelyek ki vannak vezetve a modul bemenetére. Miután 4 bemenet közül kell kiválasztani az éppen a kijelzőre kerülőt, négy bemenetű multiplexerre van szükségünk. A 7 szegmens kiválasztásához összesen 7 multiplexer kell. A multiplexerek kiválasztó bemeneteit (S0, S1) egy számláló segítségével hajtjuk meg, amit az előzőleg már előállított 400Hz-es órajelről üzemeltetünk. A számláló kimeneteit egy 2-ből 4-et dekóderre kötöttük aminek kimenetei fogják megadni, hogy melyik kijelzőt kell éppen bekapcsolnunk (MUX1, MUX2, MUX3, MUX4).

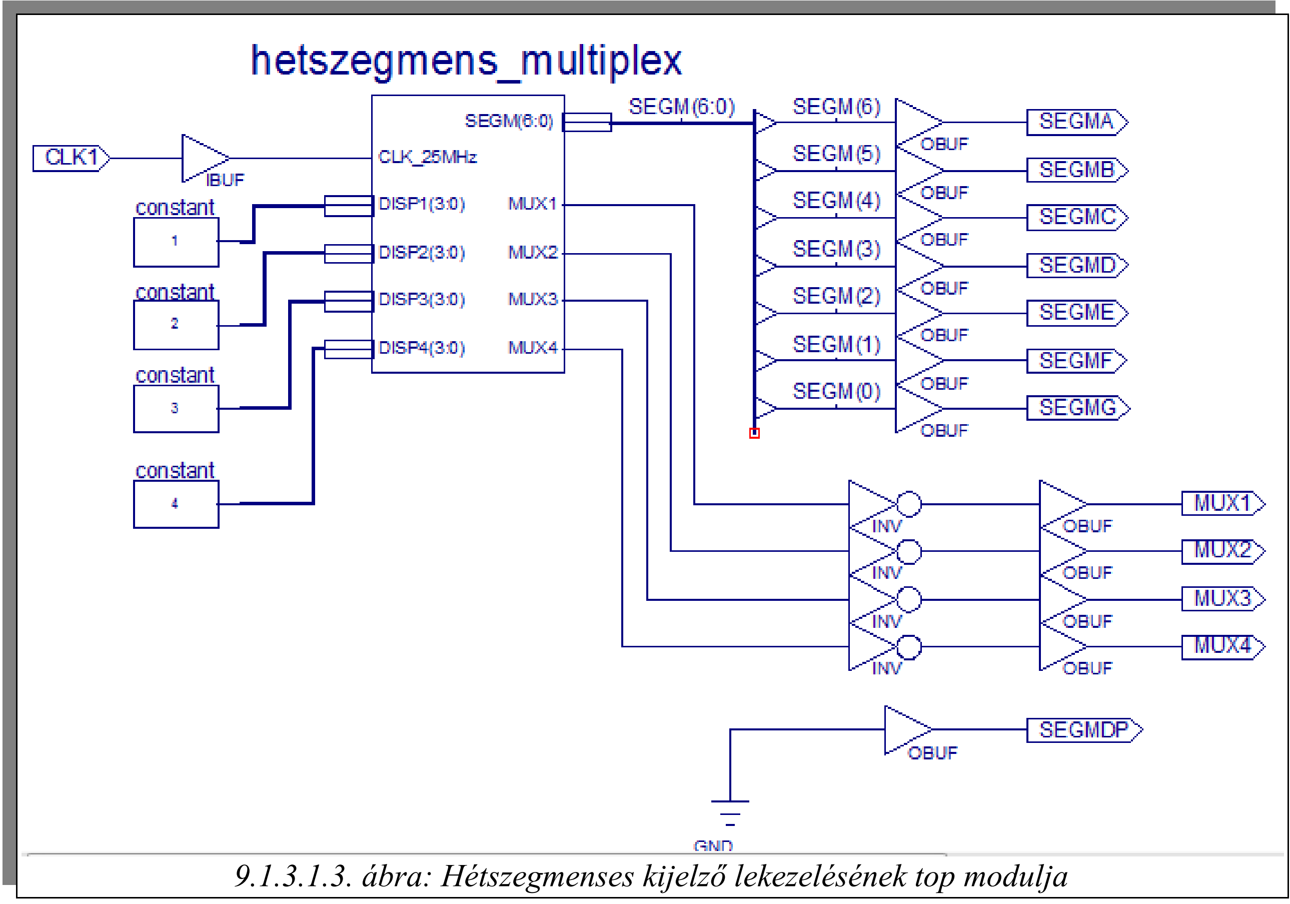
108A nyolcadik bit a DP, vagyis a tizedespont lenne, azonban ennél a projektnél ezt nem használjuk.



9.1.3.1.2. ábra: 4 kijelző multiplexelése schematic editorral

A SEGM kimenet MSB-je az a szegmens, míg az LSB a g szegmens. Erre a top modulban megadott kivezetéseknél ügyelnünk kell.

A top modult a 9.1.3.1.3. ábrán figyelhetjük meg. A MUX kivezetéseket a hardverkialakítás miatt kell invertálnunk (aktív nullás kimenet).



9.1.3.1.3. ábra: Hétszegmenses kijelző lekezelésének top modulja

A fenti feladatot a következőkben VHDL alapon oldjuk meg. A lentebb látható kód hajtja végre a dekódolást. A problémát a már többször alkalmazott case utasítással célszerű kiküszöbölni.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity dekoder is
   Port ( BCD : in  STD_LOGIC_VECTOR (3 downto 0);
          SEGM : out  STD_LOGIC_VECTOR (7 downto 0));
end dekoder;
architecture Behavioral of dekoder is
begin
            process(BCD)
            begin
                        case BCD is
                                    WHEN X"0" => SEGM <= "00111111";      --0
                                    WHEN X"1" => SEGM <= "00000110";      --1
                                    WHEN X"2" => SEGM <= "01011011";      --2
                                    WHEN X"3" => SEGM <= "01001111";      --3
                                    WHEN X"4" => SEGM <= "01100110";      --4
                                    WHEN X"5" => SEGM <= "01101101";      --5
                                    WHEN X"6" => SEGM <= "01111101";      --6
                                    WHEN X"7" => SEGM <= "00000111";      --7
                                    WHEN X"8" => SEGM <= "01111111";      --8
                                    WHEN X"9" => SEGM <= "01101111";      --9
                                    WHEN OTHERS =>        SEGM <= "00000000";     --
            --                      WHEN X'A' => SEGM <='00000000';
            --                      WHEN X'B' => SEGM <='00000000';
            --                      WHEN X'C' => SEGM <='00000000';
            --                      WHEN X'D' => SEGM <='00000000';
            --                      WHEN X'E' => SEGM <='00000000';
            --                      WHEN X'F' => SEGM <='00000000';
                        end case;
            end process;
end Behavioral;
```

Az alábbi kód hajtja végre a dekóder és egy 400Hz-es órajel modul segítségével a multiplexelést.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity hetszegmens_mux is
   Port ( BCD1 : in  STD_LOGIC_VECTOR (3 downto 0);
          BCD2 : in  STD_LOGIC_VECTOR (3 downto 0);
          BCD3 : in  STD_LOGIC_VECTOR (3 downto 0);
          BCD4 : in  STD_LOGIC_VECTOR (3 downto 0);
          CLK : in  STD_LOGIC;
          SEGMENS : out  STD_LOGIC_VECTOR (7 downto 0);
                                    MUX1 : out STD_LOGIC;
                                    MUX2 : out STD_LOGIC;
                                    MUX3 : out STD_LOGIC;
                                    MUX4 : out STD_LOGIC);
end hetszegmens_mux;
```

```
architecture Behavioral of hetszegmens_mux is
            signal CLK_int: STD_LOGIC;
            signal SEGM1: STD_LOGIC_VECTOR(7 downto 0);
            signal SEGM2: STD_LOGIC_VECTOR(7 downto 0);
            signal SEGM3: STD_LOGIC_VECTOR(7 downto 0);
            signal SEGM4: STD_LOGIC_VECTOR(7 downto 0);
            signal szam: unsigned(1 downto 0);
component dekoder is
   Port ( BCD : in  STD_LOGIC_VECTOR (3 downto 0);
         SEGM : out  STD_LOGIC_VECTOR (7 downto 0));
end component;
component Hz_400 is
   Port ( CLK_25MHz : in  STD_LOGIC;    -- 25Mhz-es órajel
         OUT_400Hz : buffer  STD_LOGIC);             -- 400Hz-es kimenet
end component;
begin
idoalap: Hz_400 port map(CLK, CLK_int);
DISP1: dekoder port map(BCD1, SEGM1);
DISP2: dekoder port map(BCD2, SEGM2);
DISP3: dekoder port map(BCD3, SEGM3);
DISP4: dekoder port map(BCD4, SEGM4);
            process(CLK_int)
            begin
                        if falling_edge(CLK_int) then
                        szam <= szam + 1;
                        --if szam = 4 then szam <= "00"; end if;
                        case szam is
                                    WHEN "00" => SEGMENS <= SEGM1; MUX4 <= '0'; MUX3 <= '0';
                                                                    MUX2 <= '0'; MUX1 <= '1';          --DISP1
                                    WHEN "01" => SEGMENS <= SEGM2; MUX4 <= '0'; MUX3 <= '0';
                                                                    MUX2 <= '1'; MUX1 <= '0';          --DISP2
                                    WHEN "10" => SEGMENS <= SEGM3; MUX4 <= '0'; MUX3 <= '1';
                                                                    MUX2 <= '0'; MUX1 <= '0';          --DISP3
                                    WHEN "11" => SEGMENS <= SEGM4; MUX4 <= '1'; MUX3 <= '0';
                                                                              MUX2 <= '0'; MUX1 <= '0';          --DISP4
                                    WHEN OTHERS =>
                        end case;
                        end if;
            end process;
end Behavioral;
```

A kiválasztást egy órajelre lefutó folyamatban elhelyezett case utasítással végezzük el. Fontos, hogy amikor az egyes váltásokat elvégezzük, a MUX kimeneteket is állítanunk kell.

9.1.3.2. BCD to Binary és Binary to BCD átalakítások

Nem csak a hétszegmenses kijelző lekezeléséhez, hanem az általános adatfeldogozáshoz is gyakran szükséges a különböző számábrázolási módok, kódolási eljárások közötti átalakítás. Egy számot egy kijelzőre – legyen az hétszegmenses, LCD, vagy egy TFT monitor – számjegyenként tudunk kitenni, így azonban nem célszerű tárolni. Egyrészt nagy a memória igénye, másrészt a műveletek végzése se egyszerű BCD számokkal. Ezért szükségesek az átalakítások oda-vissza a BCD és a bináris számok között.

Az alábbi kódrészlet a bemeneti négy kapcsoló által meghatározott érték négyzetéhez ad hozzá 114-et és azt teszi a kimenetére[109]. Felhasználva a már megírt hétszegmenses kijelző lekezelését tesztelhetjük a programot.

9.1.3.3. Fényerő-szabályozás a hétszegmenses kijelzőn

9.1.4. Komplex feladat az 1. generációs próbapanelre

9.2. 2. generációs próbapanel

9.2.1. Lábkiosztás a próbapanelhez

109Gyakori, hogy egy érzékelő lekezeléséhez kell egy képlet a karakterisztikájának linearizálásához. Pl. egy hőmérséklet érzékelésnél elengedhetetlen a hozzáadás, kivonás művelet használata, de gondolhatunk bonyolultabb esetekre is pl. egy gyorsulásmérő, vagy szögelfordulás mérő esetén nem ritka a szögfüggvények használata sem.

9.2.2. Mátrix tasztatúra lekezelése

A mátrix tasztatúra működése a nevéből adódik. A nyomógombok csatlakozása sor és oszlop alapján van elrendezve. Az általunk készített próbapanelen oszlopmeghajtás van. Miután egy ULN driver van az FPGA és a billentyűzet oszlopa közé kötve, ezért a meghajtás aktív egyes, míg az olvasás aktív nullás üzemmódban történik. A következő VHDL kódot használhatjuk a mátrix tasztatúra lekezeléséhez[110]:

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

entity Matrix_tasztatura_1 is
   Port ( CLK : in  STD_LOGIC;
         ROW : in  STD_LOGIC_VECTOR (3 downto 0);
           COL : buffer STD_LOGIC_VECTOR (2 downto 0);
           GOMB: out STD_LOGIC_VECTOR (3 downto 0):="0000";
           INT: out STD_LOGIC:='0'
        );
end Matrix_tasztatura_1;

architecture Behavioral of Matrix_tasztatura_1 is
        signal CLK_int: STD_LOGIC;
        signal COL_temp: unsigned(2 downto 0):="100";     -- 1. oszlop aktív

        component Hz_50 is
           Port ( CLK_25MHz : in  STD_LOGIC;    -- 25Mhz-es órajel
                  OUT_50Hz : buffer  STD_LOGIC);  -- 50Hz-es kimenet
        end component;

begin
idozito: Hz_50 port map(CLK, CLK_int);               --50Hz-es időzítő a forgatáshoz
        process(CLK_int, ROW, COL_temp)
                begin
                        if falling_edge(CLK_int) then
                                COL_temp <= COL_temp ror 1;    -- forgatás az oszlopokon
                                if COL_temp = "100" then                 -- 1. oszlop ellenőrzése
                                        case ROW is
                                                when "1110" => INT <= '1'; GOMB <= X"1";          -- 1
                                                when "1101" => INT <= '1'; GOMB <= X"4";          -- 4
                                                when "1011" => INT <= '1'; GOMB <= X"7";          -- 7
                                                when "0111" => INT <= '0';                                -- *
                                                when others => INT <= '0';
                                        end case;
                                end if;
                                if COL_temp = "010" then                 -- 2. oszlop ellenőrzése
                                        case ROW is
                                                when "1110" => INT <= '1'; GOMB <= X"2";          -- 2
                                                when "1101" => INT <= '1'; GOMB <= X"5";          -- 5
                                                when "1011" => INT <= '1'; GOMB <= X"8";          -- 8
                                                when "0111" => INT <= '1'; GOMB <= X"0";          -- 0
                                                when others => INT <= '0';
                                        end case;
                                end if;
```

110A kód teszteléséhez a kimenetet csatlakoztathatjuk a hétszegmenses kijelző program bemenetére.

```
                        if COL_temp = "001" then                    -- 3. oszlop ellenőrzése
                              case ROW is
                                    when "1110" => INT <= '1'; GOMB <= X"3";          -- 3
                                    when "1101" => INT <= '1'; GOMB <= X"6";          -- 6
                                    when "1011" => INT <= '1'; GOMB <= X"9";          -- 9
                                    when "0111" => INT <= '0';                              -- #
                                    when others => INT <= '0';
                              end case;
                        end if;
                  end if;
            end process;
            COL <= std_logic_vector(COL_temp);        -- oszlop a kimenetre
end Behavioral;
```

A modul az oszlop meghajtást az „100” kombináció forgatásával éri el. Miután std_logic_vector típust nem tudunk forgatni, ezért egy segédjelet alkalmazunk, amit majd az egyidejű részben fogunk a kimenetre tenni egy konverzió segítségével. A forgatást 50Hz-es órajelre végezzük. Minden forgatás során megvizsgáljuk a sorokat és amennyiben valahol nullát mérünk az oszlopnak és sornak megfelelő értéket töltjük a GOMB 4 bites kimenetre. Erről a case utasítások gondoskodnak. A modul tartalmaz még egy INT lábat, amin a felfutó él jelzi a gomb lenyomását, míg a lefutó a gomb elengedését. A „*” és a „#” gombok lenyomását, valamint több gomb egyszerre történő megnyomását a kód nem kezeli.

9.2.3. Karakteres LCD kijelző meghajtása

A továbbiakban a HD44780 típusú kijelzők adatlapja alapján járunk el.

A karakteres LCD kijelző lekezeléséhez szükségünk van olyan áramkörre,, amely adott késleltetést tesz számunkra lehetővé. A kijelzőnek kell egy kis idő, amíg „feléled” bekapcsolás után. Ez az adatlap szerint 15ms. A továbbiakban az inicializálásnál ettől eltérő időket is tudnunk kell várni (l. táblázat). Természetesen megtehetjük, hogy minden lépésben ugyanannyit várakozunk (a legnagyobb időt) azonban korrektebb, ha az adatlap által megadott időkkel dolgozunk[111]. A kijelző felélesztéséhez szükség van egy adott szekvenciára, az ún. inicializálásra. Ez a folyamat különböző kombinációk elküldéséből áll. Miután mi 4 biten kezeljük a kijelzőt, célszerű az adatlap ide vonatkozó részét tanulmányozni:

HD44780 inicializálása 4 bites interfésszel			
RS	*RW*	*DB*	*Megjegyzés*
			15ms várakozás a táp felfutásáig (2,7V esetén 40ms)
0	0	0011	Function set (Busy még nincs) 4,1ms várakozás
0	0	0011	Function set (Busy még nincs) 100µs várakozás
0	0	0011	Function set (Busy még nincs)
0	0	0010	Busy már él[112]
0	0	0010	Function set
0	0	N F * *	
0	0	0000	Display off
0	0	1000	
0	0	0000	Dipslay clear
0	0	0001	
0	0	0000	Entry mode set
0	0	01 I/D S	

*: Dont' Care
N: „0” → egysoros „1” → kétsoros
F:: „0” → 5X8 pont; „1” → 5X10 pont
I/D: „0” → Decremenet ; „1” → Increment (kurzor automatikus léptetése)
S: A kiírást kijelzőléptetés kíséri

Szükségünk van egy olyan alkatrészre, ami a különböző időközöket biztosítja számunkra. Ehhez tanulmányozzuk a következő kódot:

111 Az ismertetett kód az adatlapon megadottal szemben valamivel többet várakozik az egyes szekvenciákban a biztonság kedvéért, hogy ha lehet kijelzőtől és hardvertől függetlenül biztosítsa a működőképességet.

112 Amennyiben nem a Busy flaget teszteljük, akkor a kijelző törlése után legalább 1,52ms, míg minden más parancs kiadása és adat küldése után 37µs várakozás szükséges.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity Delay is
   Port ( CLK_25MHz : in  STD_LOGIC;
        IDO : in  STD_LOGIC_VECTOR (7 downto 0);
        START_US : in  STD_LOGIC;
          START_MS : in  STD_LOGIC;
        DONE : buffer  STD_LOGIC:='0');
end Delay;
-- várakozás entitás, az ido bemenetenek legalább 2-őnek kell lennie!!!
architecture Behavioral of Delay is
          signal ido_ms: STD_LOGIC;         -- 1ms-os belső jel
          signal ido_us: STD_LOGIC;                   -- 1us-os belső jel
          signal szam: unsigned (7 downto 0) :=(OTHERS => '0');
          signal CLK_int: STD_LOGIC;
component kHz1 is                    -- 1 kHz-et előállító komponens
   Port ( CLK : in  STD_LOGIC;
        kHz : buffer  STD_LOGIC);
end component;
component MHz1 is                    -- 1 MHz-et előállító komponens
   Port ( CLK : in  STD_LOGIC;
        MHz : buffer  STD_LOGIC);
end component;
begin
          ms_1: kHz1 port map(CLK_25MHz, ido_ms); -- 1ms előállítása
          us_1: MHz1 port map(CLK_25MHz, ido_us); -- 1us előállítása
          process(START_US, START_MS, CLK_int)
          begin
          if START_US = '1' then   -- ha us bement aktív (prioritásos)
                    CLK_int <= ido_us;         -- akkor mikroszekundumos az órajel
          elsif START_MS = '1' then            -- különben ha ms bement aktív
                    CLK_int <= ido_ms;         -- akkor milliszekundumos az órajel
          end if;
          if falling_edge(CLK_int) then                   -- lefutó élre
                              if DONE = '1' then DONE <= '0'; end if;
                              szam <= szam+1;                    -- szám növelése
                              if szam = unsigned(IDO) then       -- ha letelt az idő
                                        DONE <= '1';                       -- a kimenet = 1
                                        szam <= (OTHERS => '0');
                              end if;
          end if;
          end process;
end Behavioral;
```

Az entitásunk tartalmaz két komponenst, az egyik 1kHz-es, míg a másik 1MHz-es órajelet állít elő. Abban az esetben, ha a START_MS bemenet aktív, akkor 1kHz-es, ha a START_US bemenet aktív, akkor 1MHz-es órajellel dolgozunk[113]. Amennyiben az órajelből elérjük az IDO bemenet által meghatározott értéket, a DONE kimenetet '1'-be állítjuk. Miután a kiszolgálni kívánt áramkörben éldetektálás történik, ezért csak egy impulzust kell adnunk ezen a kimeneten. Amennyiben jön egy következő órajel a DONE lábat nullázzuk. A lefutó élek az IDO bemenet alapján követik egymást. Ezáltal létrehoztunk egy változtatható értékű várakozó áramkört.

113Amennyiben mindkét bemeneten logikai '1'-et érzékel az áramkör, akkor a mikroszekundumos bemenetet kezeli prioritással.

Az alábbi kód teszi lehetővé az 1MHz-es jel létrehozását a 25MHz-es órajelünkből.

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

entity MHz1 is
   Port ( CLK : in  STD_LOGIC;
          MHz : buffer  STD_LOGIC);
end MHz1;

architecture Behavioral of MHz1 is
	signal szam: unsigned (3 downto 0):= (others => '0');
	signal paros: bit :='0';
begin
	process(CLK)
	begin
	if falling_edge(CLK) then				-- 25MHz-es órajelre
		szam <= szam+1;					-- szám növelése
		if szam = 12 and paros = '0' then	-- ha 12 akkor
			MHz <= not MHz;					-- kimenet invertálása
			szam <= (others => '0');	-- valójában 12,5 kellene
			paros <= '1';					-- ezért a következő körben
		end if;
		if szam = 13 and paros = '1' then	-- ha 13 akkor
			MHz <= not MHz;					-- kimenet invertálása
			szam <= (OTHERS => '0');
			paros <= '0';					-- következőre 12-öt nézünk
		end if;
	end if;
	end process;
end Behavioral;
```

A 12,5-el való osztást úgy tudjuk elérni, hogy elsőre 12-vel osztunk, majd utána 13-al. Így hosszabb távon kiátlagolva kijön a 12,5.

A kijelzőt meghajtó áramkörnek képesnek kell lennie – bekapcsolás késleltetés után – végrehajtani az inicializáló szekvenciát, majd átadnia a vezérlést a felhasználónak és a bemenetére érkező parancsokat/adatokat továbbítani a kijelzőnek.

Az alábbi VHDL leírás ezt teszi lehetővé:

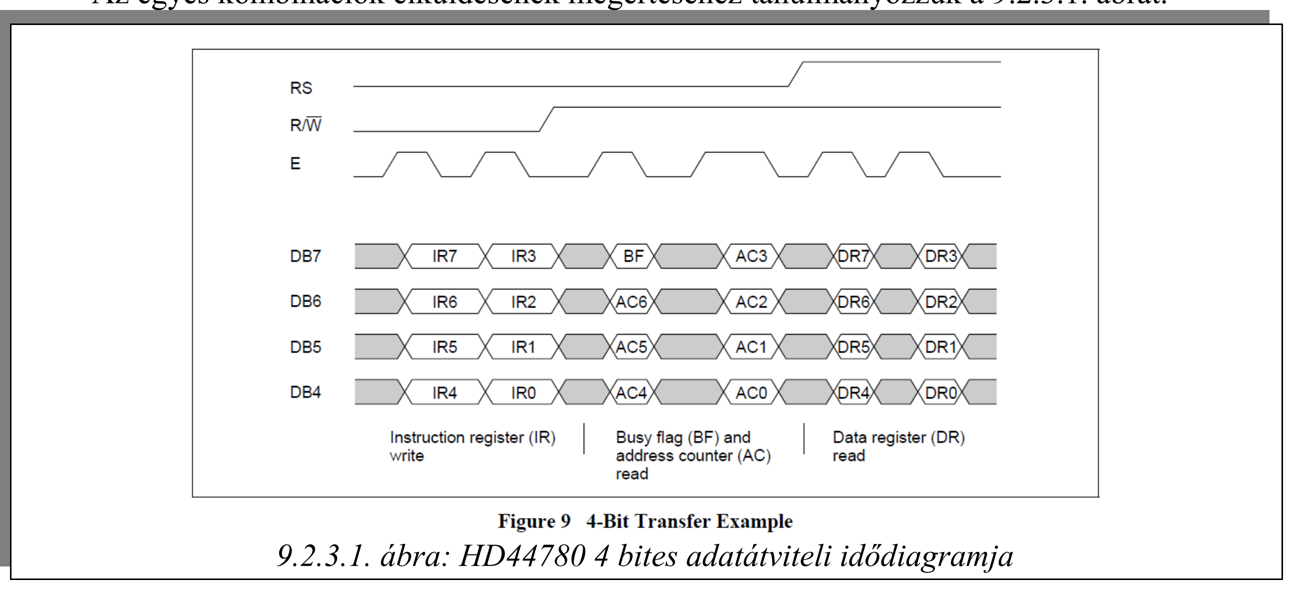
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity LCD_driver is
   Port ( DATA : in  STD_LOGIC_VECTOR (7 downto 0);
          RS_IN : in  STD_LOGIC;
          CLK : in  STD_LOGIC;
          VALID : in  STD_LOGIC;
          DB : out  STD_LOGIC_VECTOR (3 downto 0);
          RW_OUT : out  STD_LOGIC;
          RS_OUT : out  STD_LOGIC;
          E_OUT : out  STD_LOGIC);
end LCD_driver;
```

```
architecture Behavioral of LCD_driver is
            signal ido_int: STD_LOGIC_VECTOR(7 downto 0):= X"32";             -- várakozási idő 50ms
            signal start_us_int: STD_LOGIC:= '0';         -- várakozás mikrosecben
            signal start_ms_int: STD_LOGIC:= '1';         -- várakozás millisecben
            signal done_int: STD_LOGIC:= '0';                         -- várakozás letelt
            signal szam: unsigned(7 downto 0):= X"00";                          -- lépések száma
            signal char_done: STD_LOGIC:= '0';            -- karakter kirakva
            signal data_valid: STD_LOGIC:= '0';           -- adat érvényes

component Delay is                                                   -- változtatható paraméterű késleltetés
   Port ( CLK_25MHz : in  STD_LOGIC;
         IDO : in  STD_LOGIC_VECTOR (7 downto 0);
         START_US : in  STD_LOGIC;
                                 START_MS : in  STD_LOGIC;
         DONE : buffer  STD_LOGIC);
end component;

begin
            idozito: Delay port map(CLK,ido_int,start_us_int,start_ms_int,done_int);

process(done_int)
begin
            if rising_edge(done_int)  then -- ha lejárt a késleltetés
                                    case szam is
            when X"00" => DB <= "0011"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='1';
            ido_int <= X"04"; start_ms_int <= '0'; start_us_int <= '1';
                                    -- ENABLE láb 4 us-ig magas szinten
            when X"01" => DB <= "0011"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='0';
            ido_int <= X"0A"; start_ms_int <= '1'; start_us_int <= '0';
                                    -- parancs után 10ms várakozás
                                    -- első 4 bit elküldve
            when X"02" => DB <= "0011"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='1';
            ido_int <= X"04"; start_ms_int <= '0'; start_us_int <= '1';
                                    -- ENABLE láb 4 us-ig magas szinten
            when X"03" => DB <= "0011"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='0';
            ido_int <= X"96";
                                    -- parancs után 150us várakozás
                                    -- második 4 bit elküldve, innentől kezdve normál várakozás (min 37 us)
            when X"04" => DB <= "0011"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='1';
            ido_int <= X"04";
                                    -- ENABLE láb 4 us-ig magas szinten
            when X"05" => DB <= "0011"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='0';
            ido_int <= X"04";
                                    -- ENABLE láb 4 us-ig alacsony szinten
            when X"06" => DB <= "0010"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='1';
            ido_int <= X"04";
                                    -- ENABLE láb 4 us-ig magas szinten
            when X"07" => DB <= "0010"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='0';
            ido_int <= X"32";
                                    -- parancs után 50us várakozás
```

```
--************************* FUNCTION SET ******************************
when X"08" => DB <= "0010"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='1';
ido_int <= X"04";
                    -- ENABLE láb 4 us-ig magas szinten
when X"09" => DB <= "0010"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='0';
ido_int <= X"04";
                    -- ENABLE láb 4 us-ig alacsony szinten
when X"0A" => DB <= "1000"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='1';
ido_int <= X"04";
                    -- ENABLE láb 4 us-ig magas szinten
when X"0B" => DB <= "1000"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='0';
ido_int <= X"32";
                    -- parancs után 5ms várakozás

--***************** DISPLAY, CURSOR ON, BLINK ON ********************
when X"0C" => DB <= "0000"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='1';
ido_int <= X"04";
                    -- ENABLE láb 4 us-ig magas szinten
when X"0D" => DB <= "0000"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='0';
ido_int <= X"04";
                    -- ENABLE láb 4 us-ig alacsony szinten
when X"0E" => DB <= "1111"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='1';
ido_int <= X"04";
                    -- ENABLE láb 4 us-ig magas szinten
when X"0F" => DB <= "1111"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='0';
ido_int <= X"32";
                    -- parancs után 50us várakozás

--************************* DISPLAY_CLEAR ***************************
when X"10" => DB <= "0000"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='1';
ido_int <= X"04";
                    -- ENABLE láb 4 us-ig magas szinten
when X"11" => DB <= "0000"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='0';
ido_int <= X"04";
                    -- ENABLE láb 4 us-ig alacsony szinten
when X"12" => DB <= "0001"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='1';
ido_int <= X"04";
                    -- ENABLE láb 4 us-ig magas szinten
when X"13" => DB <= "0001"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='0';
ido_int <= X"04";  start_ms_int <= '1'; start_us_int <= '0';
                    -- parancs után 4ms várakozás

--************************* ENTRY MOD SET ***************************
when X"14" => DB <= "0000"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='1';
ido_int <= X"04"; start_ms_int <= '0'; start_us_int <= '1';
                    -- ENABLE láb 4 us-ig magas szinten
when X"15" => DB <= "0000"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='0';
ido_int <= X"04";
                    -- ENABLE láb 4 us-ig alacsony szinten
when X"16" => DB <= "0110"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='1';
ido_int <= X"04";
                    -- ENABLE láb 4 us-ig magas szinten
when X"17" => DB <= "0110"; RS_OUT <='0'; RW_OUT <='0'; E_OUT <='0';
ido_int <= X"32";
                    -- parancs után 50us várakozás, készen áll a karakter kirakásra
```

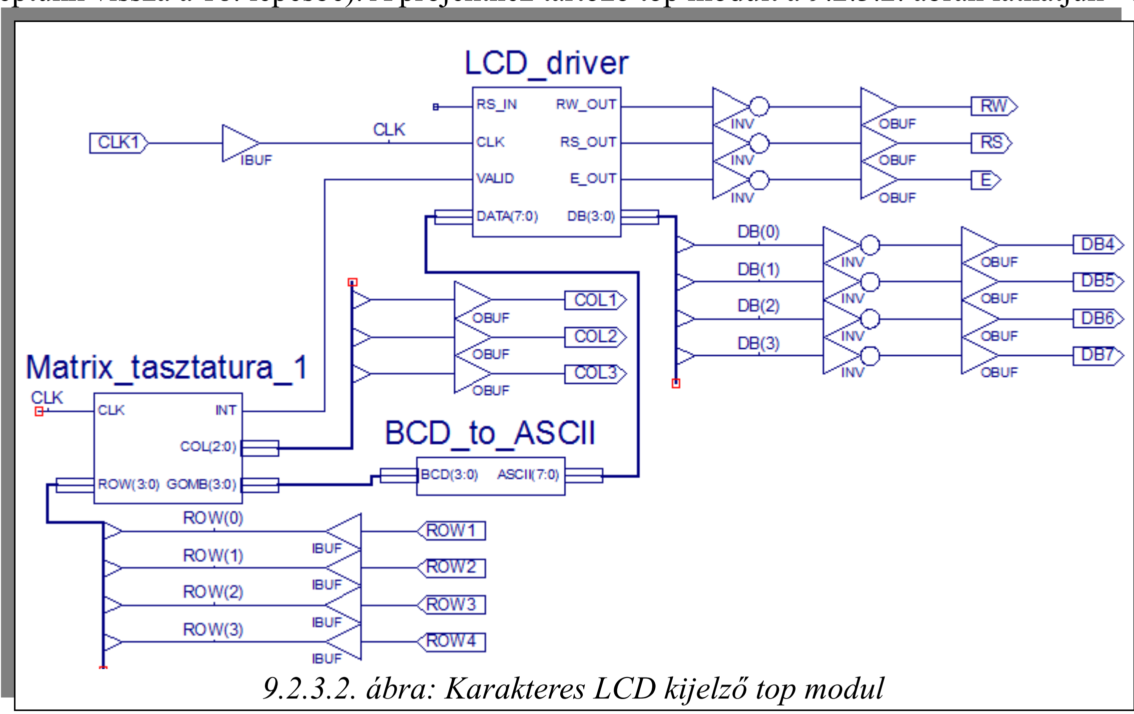
```
--************************* DATA kirakása ********************************
        when X"18" => ido_int <= X"04";
                        -- karakterre várakozás
        when X"19" => DB <= DATA(7 downto 4); RS_OUT <='1'; RW_OUT <='0'; E_OUT <='1';
        ido_int <= X"04"; start_ms_int <= '0'; start_us_int <= '1';
                        -- ENABLE láb 4 us-ig magas szinten
        when X"1A" => DB <= DATA(7 downto 4); RS_OUT <='1'; RW_OUT <='0'; E_OUT <='0';
        ido_int <= X"04";
                        -- ENABLE láb 4 us-ig alacsony szinten
        when X"1B" => DB <= DATA(3 downto 0); RS_OUT <='1'; RW_OUT <='0'; E_OUT <='1';
        ido_int <= X"04";
                        -- ENABLE láb 4 us-ig magas szinten
        when X"1C" => DB <= DATA(3 downto 0); RS_OUT <='1'; RW_OUT <='0'; E_OUT <='0';
        ido_int <= X"32";
                        -- adat után 50us várakozás
                        when others =>
                        end case;
                if data_valid = '1' then        -- ha egy karaktert ki kell raknunk
                        szam <= X"19";  -- azt a 19-as lépésnél kell kezdeni
                        elsif szam = X"1C" then
                        szam <= X"18";  -- várakozás új karakterre
                        elsif szam /= X"18"        then        -- különben ha nem karakterre várakozunk
                        szam <= szam +1;           -- lépésszám növelése
                end if;
        end if;
end process;
process(VALID, szam)
begin
        if rising_edge(VALID) then              -- karaktert ki kell rakni
                if szam = X"18" then            -- inicializálás megtörtént
                        data_valid <= '1';
                end if;
        end if;
        if szam = X"19" then            -- karaktert
                data_valid <= '0';
        end if;
end process;
end Behavioral;
```

Az egyes kombinációk elküldésének megértéséhez tanulmányozzuk a 9.2.3.1. ábrát.



9.2.3.1. ábra: HD44780 4 bites adatátviteli idődiagramja

A fenti entitás komponensként használja az előbbiekben már megbeszélt várakozást. Kezdőértékként 50ms-ot adunk meg – ez a bekapcsolási késleltetés. A továbbiakban követjük az adatlapban található szekvenciát. Az időket kissé megnöveltük. Az Enable lábnak legalább 1µs-ig magas szinten kell lennie a parancsok kiadásakor[114]. Az első process 17. lépéséig tart az inicializálási rész, eddig folyamatosan növeljük a szam jel értékét a várakozásnak megfelelően. Amennyiben elérjük a 18-as számértéket a növelés megáll. Egészen addig nem folytatódik, míg a data_valid jel egybe nem áll a másik processnek köszönhetően. Ha ez megtörténik, végrehajtódik a további négy lépés, majd újra a 18. következik, a következő data_valid jelig. A 2. folyamat a VALID bemeneten történő felfutó él hatására '1'-be állítja a data_valid jelet amennyiben a 18. lépésben van a másik folyamat (ez jelzi, hogy kész a karakter kirakására). A 19. lépés kinullázza a data_valid jelet és egészen addig nem is lehet újra aktiválni, míg ki nem raktuk a karaktert (vagyis nem léptünk vissza a 18. lépésbe). A projekthez tartozó top modult a 9.2.3.2. ábrán láthatjuk[115].



9.2.3.2. ábra: Karakteres LCD kijelző top modul

A könnyebb vizsgálat érdekében felhasználtuk az előző projektben készített tasztatúra modult. Az invertálásokra a hardverkialakítás miatt van szükség.

A lenti kód a BCD_to_ASCII entitás leírását tartalmazza, amely BCD számból generál ASCII karaktereket – az alkatrész a teszteléshez szükséges.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity BCD_to_ASCII is
   Port ( BCD : in  STD_LOGIC_VECTOR (3 downto 0);
        ASCII : out  STD_LOGIC_VECTOR (7 downto 0));
end BCD_to_ASCII;
architecture Behavioral of BCD_to_ASCII is
begin
          ASCII <= std_logic_vector(unsigned(BCD) + X"30");
end Behavioral;
```

114Miután egy tranzisztoros szinteltolón keresztül hajtjuk meg a lábat a biztonság kedvéért 4µs-ot várunk.

115Jelenleg az RS_IN bemenetet a VHDL kód nem hasznosítja. Amennyiben szeretnénk parancsokat is megadni az LCD-nek, akkor az LCD_driver CASE utasításában lévő utolsó lépésekben (19÷1C) az RS_OUT kimenetet az RS_IN bemenetről kell vezérelni.

9.2.4. A panel kiegészítése

A második generációs panel használható az 1. nélkül is, így számos I/O láb felszabadul számunkra. Ezekre csatlakoztathatunk különböző perifériákat.

9.2.4.1. Szervomotor vezérlés

A szervomotor vezérlése viszonylag egyszerű feladat. Elő kell állítanunk egy 50Hz-es PWM jelet. A 20ms-os periódusból a magas szintnek 1 és 2ms között kell lennie. Az 1ms jelenti a bal, a 2ms a jobb végállást, míg a 1,5 ms a középállást. A skála lineárisan van felosztva 0-90º, vagy 0-180º között – ez függ a szervomotortól.

A következő kódrészlet előállítja a szükséges PWM jelet, a bemenetén 0 és 180 közötti értékben tudjuk megadni a kívánt szöget. Amennyiben ennél nagyobb számot kötünk a bemenetre, az áramkör nem veszi azt figyelembe.

A VHDL leíráshoz tartozó top modult a . ábrán láthatjuk.

A következő feladathoz felhasználunk két nyomógombot és két kapcsolót. A két nyomógombbal tudjuk a szervomotort az óramutató járásával megegyezően, ill. ellentétesen forgatni, míg a két kapcsoló a minimális forgatás szögértékét (a forgatás finomságát) határozza meg.

9.2.4.2. Soros port lekezelése

A panelhez csatlakoztathatunk egy sorosport-USB átalakítót (részletekért l . fejezet), ezáltal szabványos soros kommunikációs perifériát alakíthatunk ki az FPGA-val, nullmodemes

kapcsolathoz. A kommunikáció RS-232C protokoll alapján működik. A kommunikáció fontosabb adatai:

- Aszinkron, 8 bites adatátvitel
- Nincs paritás
- Baud rate: 9600

9.2.5. Komplex feladat a 2. generációs próbapanelre

9.3. 3. generációs próbapanel

9.3.1. Lábkiosztás a próbapanelhez

9.3.2. VGA jel előállítása

9.3.3. PS2 port lekezelése

9.3.4. AD átalakítás és I2C protokoll

9.3.5. Komplex feladat a 3. generációs próbapanelre

10. Tematika

Miután az oktatócsomag használatát szakköri keretek között tudjuk csak megvalósítani, ezért az alábbiakban olvasható tematika és tanmenet csupán ajánlásnak tekinthető.

A különböző képességű és képzettségű diákok eltérő haladási sebességét célszerű figyelembe venni a szakköri foglalkozások megtartásakor. Érdemes – az érdeklődési körnek megfelelő – önálló feladatokkal, otthoni fejlesztésekkel színesebbé tenni az órákat.

10.1. Egy javasolt szakköri óramenet

- A szakkör elején érdemes tisztázni az előző alkalommal végzett munkával kapcsolatban felmerülő kérdéseket
    - Kérdezzük meg a diákokat, hogy elsajátították-e a kívánt ismereteket (lehetőség szerint ellenőrizzük ezeket)
    - A problémás pontokra térjünk ki ismételten fokozottan bevonva a szakkör résztvevőit a megoldás kialakításába
- Ismertessük az újdonságokat
    - Próbáljuk elhelyezni az új információt a műszaki életben
    - Hozzunk minél több ipari példát
    - Hivatkozzunk a későbbi szakköri foglalkozásokon való alkalmazhatóságra és fontosságra
- Készüljünk plusz feladatokkal, amelyek komolyabb erőfeszítéseket igényelnek, illetve olyan problémákkal, amelyek az adott órán elsajátítandó ismereteket más megközelítésből használják fel. A plusz feladatokkal ki tudjuk küszöbölni az egyéni sajátosságokból adódó differenciálási nehézségeket. Amennyiben több azonos ismereteket követelő probléma áll rendelkezésünkre, ezek megoldásával elősegíthetjük azon diákok fejlődését, akiknek egy-egy új információ alkalmazásához több gyakorlásra van szükségük.
- A diákokat az új ismeret átadása után az önálló tevékenységre ösztönözzük. Próbáljuk rávenni őket saját elképzeléseik végrehajtására. Támogassuk őket abban, hogy dolgozzanak ki az iparban is elképzelhető problémákra megoldásokat, részmegoldásokat. Minden ötletet díjazzunk, lehetőség szerint együtt beszéljük meg őket, és dolgozzuk ki a részleteket. Amennyiben tudjuk építsük be az új információt tanmenetünkbe.
- A szakkör végén ne felejtsük el a tanulságok levonását. Mindig próbáljuk meg elgondolkodtató otthoni feladatokkal fenntartani a folyamatos érdeklődést.

10.2. Tanmenet

MECHATRONIKAI SZAKKÖZÉPISKOLA ÉS GIMNÁZIUM

TANMENET

Az FPGA szakkör tanításához

34 hét – heti 2 óra – évi 68 óra

Összeállította: Varga László

2013-2014.

…..

aláírás

Tematika

FPGA SZAKKÖR
34 HÉT, HETI 2 ÓRA, ÉVI 68 ÓRA

1. **foglalkozás**

- Ismerkedés a tanulókkal
  - A tematika rövid ismertetése
- Munkavédelmi és tűzvédelmi oktatás, általános műhelyrend ismertetése[116]
- Érintésvédelmi alapismeretek
  - A villamos áram élettani hatása
    - Izmokra gyakorolt bénulás, vegyi- és hőhatás
    - Az áramütés hatásának függése (áramút, frekvencia, áramerősség, időtartam, emberi tényező)
  - Áramhatárértékek ismertetése (érzetküszöb, elengedési áram, veszélyes érték, halálos dózis)
  - Törpe-, kis- és nagyfeszültség fogalma
  - Kisfeszültségű berendezések alapfogalmainak tisztázása (föld, földelés, fázis, nulla, védővezető, átütési feszültség)
  - Passzív és aktív érintésvédelem fajtái és hatékonyságuk (elkerítés, védőelválasztás biztonsági transzformátor alkalmazásával, szigetelés-kettős szigetelés, burkolás; feszültség- és áramvédő kapcsolás, nullázás, védőföldelés)
  - Teendők áramütés esetére, a sérült ellátása
  - Villámvédelem

2. **foglalkozás**

- A műhely eszköz- és műszerparkjának bemutatása
- Műszerkezelési gyakorlat[117]

3. **foglalkozás**

- Automatikai alapfogalmak
  - Vezérlés
  - Szabályozás
  - Stabilitás

4. **foglalkozás**

- Mikrovezérlők, mikroprocesszorok

5. **foglalkozás**

- Korszerű fejlesztéstechnika
  - ASIC
  - PIC
  - FPGA

116A munkavédelmi oktatás kötelező gyakorlati foglalkozások esetén, azonban ez ismétlő oktatás esetén jelentősen lerövidülhet. Mindenképpen töltessünk ki a tanulókkal az oktatásról jegyzőkönyvet, és a felmerülő esetleges kérdéseket tisztázzuk velük.

117Az előzetes képzéstől függően elhagyható. Multimétert már az első feladatokhoz is célszerű alkalmazni.

6. foglalkozás
- Frontális előadás az FPGA-ról
  - Használjuk a Moodle keretrendszerben található prezentációt

7. foglalkozás
- Az előadáson felmerült kérdések tisztázása

8. foglalkozás
- A fejlesztőpanelek ismertetése
  - Perifériák bemutatása
  - Interfészek, csatlakozások, tápellátás

9. foglalkozás
- FPGA fejlesztőkörnyezet bemutatása (pl. Xilinx ISE Webpack)

10. foglalkozás
- Szintézis kapcsolási rajz alapú szerkesztővel

11. foglalkozás
- UCF fájl generálása, lábkiosztás beállítása

12. foglalkozás
- Implementáció, boundary scan
- Dokumentációs feladatok fejlesztés közben

13. foglalkozás
- Kombinációs hálózatok I.

14. foglalkozás
- Kombinációs hálózatok II.

15. foglalkozás
- Önálló feladat megoldása

16. foglalkozás
- Szekvenciális hálózatok

17. foglalkozás
- Egyszerű szekvenciális feladatok megoldása

18. foglalkozás
- PWM jel előállítása, pergésmentesítés

19. foglalkozás
- HDL nyelvek

20. foglalkozás
- VHDL, Verilog szintaktikai alapok I.

21. foglalkozás
- VHDL, Verilog szintaktikai alapok II.

22. foglalkozás
- Hétszegmenses kijelző lekezelése
  - Dekóder implementálása
  - Multiplexelés

23. foglalkozás
- Komplex feladat végrehajtása

24. foglalkozás
- Mátrix tasztatúra lekezelése

25. foglalkozás
- LCD kijelző meghajtása I.
  - HD44780 alapok

26. foglalkozás
- LCD kijelző meghajtása II.
  - Sztringek kijelzése
  - Mátrix tasztatúráról beolvasott számértékek kijelzése

27. foglalkozás
- Szervo motor meghajtása, soros port

28. foglalkozás
- VGA jelek előállítása I.

29. foglalkozás
- VGA jelek előállítása II.

30. foglalkozás
- A PONG nevű program elemzése és vizsgálata

31. foglalkozás
- Egyszemélyes PONG megvalósításának lehetősége FPGA-val

32. foglalkozás
- Egyszemélyes PONG megvalósításának lehetősége FPGA-val

10.2.1. Utószó 1

Erdélyi „Mindenmüxik” Zsolt

10.2.2. Utószó 2

Pázmándi „Romboló” Péter

10.2.3. Utószó 3

Mottó: *Ha távolabbra láttam másoknál,*
azt azért tehettem,
mert óriások vállán álltam.

Az oktatócsomagot elkészülte után átolvasva – mint minden eddigi munkámnál – itt is sok hiányosságot vélek felfedezni. Úgy gondolom egy ilyen jellegű leírást nem lehet befejezni, csak abbahagyni. A leírás készítésénél sajnos nagyon szoros határidőt kellett tartanunk – úgy érzem ez kicsit meg is látszik a munkán. Mindemellett sajnos a fejlesztőcsapatot több külső körülmény is nehéz helyzetbe hozta (különböző vizsgák, versenyek, érettségi, stb.). Objektíven nézve azonban remélem sikerült egy használható dokumentumot elkészítenünk. A további kiegészítéseket, bővítéseket minél előbb megpróbáljuk feltölteni iskolánk Moodle rendszerébe.

Itt szeretném megragadni az alkalmat, hogy köszönetet mondjak néhány embernek, akik hozzájárultak ahhoz, hogy FPGA-val kezdjek foglalkozni és hogy ez a dokumentáció elkészüljön. Mindenekelőtt a fejlesztőcsapat két másik tagjának (**Erdélyi Zsolt**nak és **Pázmándi Péter**nek) szeretném megköszönni, hogy lehetővé tették az oktatócsomag megszületését. Köszönet illeti **Juhász Róbert** barátomat és kollégámat, aki nagyban hozzájárult fejlesztéseinkhez és tartotta bennem a lelket akkor is, amikor már inkább feladtam volna az egész projektet. Természetesen nem hagyhatom ki a felsorolásból a Kandó Kálmán Villamosmérnöki Főiskolai Kar Műszer-Automatika Tanszékének néhány oktatóját (**Kohut József, Molnár Ferenc, Molnár Zsolt, Zsom Gyula**), akiktől nagyon sokat tanultam mind az elektronika, mind a digitális technika, mind a beágyazott rendszerek, mind az FPGA területéről. Köszönet illeti még **Szabó Zoltán**t aki a Trefort Ágoston Mérnökpedagógia Központ tanáraként segítséget nyújtott az oktatócsomag, valamint a hozzá tartozó módszertani ismereteket tartalmazó szakdolgozat létrejöttéhez.

Varga László

Irodalomjegyzék

1: , http://www.xilinx.com/fpga/index.htm, 2013,
2: , http://www.origin.xilinx.com/products/design-tools/ise-design-suite/index.htm, 2012,
3: , http://www.xilinx.com/products/design-tools/ise-design-suite/system-edition.htm, 2013,

Ábrajegyzék

2.1. ábra: FPGA felépítése....11
2.2. ábra: CLB felépítése....12
2.3. ábra: IOB felépítése[1]....12
4.1. ábra: Spartan-3 felépítése....17
4.2. ábra: Spartan3 tokozás és jelölései....18
4.3. ábra: Rendelési azonosítók....19
4.4. ábra: IOB felépítése....20
4.5. ábra: DCM működése....21
4.6. ábra: Sink és load áramok, impedanciák fel-, ill. lehúzó ellenállások használata esetén....21
4.7. ábra: DCI lehetőségek....23
4.8. ábra: Bankszelektálás....24
4.9. ábra: A CLB Slice-ra (szeletekre) történő felosztása....25
4.10. ábra: BRAM kezelése....26
4.11. ábra: DCM modul felépítése működése....27
4.12. ábra: DLL modul működése....27
4.13. ábra: Órajelek csatlakozása az FPGA-hoz....28
4.14. ábra: Hosszú vonalak....29
4.15. ábra: Hex vonalak....29
4.16. ábra: Dupla vonalak....29
4.17. ábra: Direkt (közvetlen) vonalak....30
5.1. ábra: Basys 2 fejlesztőpanel....32
5.2. ábra: Perifériák elhelyezkedése a Basys2-őn....32
5.3. ábra: Nexys2 FPGA fejlesztőpanel....33
5.4. ábra: Perifériák a Nexys2 panelen....33
5.5. ábra: Nexys3 fejlesztőpanel....34
5.6. ábra: Nexys3 fejlesztőpanel perifériái....34
5.7. ábra: FPGA feszültségeit előállító áramkör blokkvázlata....35
5.8. ábra: Perifériapanelek tápellátását biztosító áramkör blokkvázlata....36
5.9. ábra: Tápegység áramkör....37
5.10. ábra: Adapterpanel felépítése....38
5.11. ábra: A fejlesztőkörnyezet korai változata....39
5.12. ábra: 1. generációs próbapanel....40
5.13. ábra: 2. generációs próbapanel....41
6.1. ábra: Alapképernyő indításkor....46
6.2. ábra: "A nap tippje"....46
6.3. ábra: Menürendszer és eszköztárak....46
6.4. ábra: Munkaterület....47
6.5. ábra: Menüsor....47
6.6. ábra: Aktuális fájl befolyásolása....47
6.7. ábra: Fájl menü....48
6.8. ábra: Szerkesztés menü....49
6.9. ábra: Konvertálás menüpont....49
6.10. ábra: Ugrás menüpont....50
6.11. ábra: Adott sorra ugrás az aktuális fájlban....50
6.12. ábra: Kijelölés menüpont....50
6.13. ábra: Szerkesztés menü kiegészítés....51

6.14. ábra: Speciális beillesztés......51
6.15. ábra: Elavult szimbólumok kézi frissítése......52
6.16. ábra: Fájl megnyitásakor megjelenő automatikus frissítési ablak......52
6.17. ábra: Lapméret megváltoztatása......53
6.18. ábra: Objektum tulajdonságai......53
6.19. ábra: Objektum tulajdonságok jellemzőinek megadása......53
6.20. ábra: Szerkesztőfelület tulajdonságainak kategóriái......54
6.21. ábra: Konzol beállításai......54
6.22. ábra: HTML böngésző beállításai......55
6.23. ábra: ISE általános beállításai......55
6.24. ábra: Tervezési célok és stratégiák......56
6.25. ábra: Szerkesztő kiválasztása......56
6.26. ábra: Integrált eszközök beállításai......56
6.27. ábra: Folyamatok befejezéséről való értesítés......57
6.28. ábra: ISE szövegszerkesztő beállításai......57
6.29. ábra: Nyelvi sablonok......58
6.30. ábra: RTL és technológia eszköztár beállításai......58
6.31. ábra: Szín-összeállítás......58
6.32. ábra: Új objektumok színei......58
6.33. ábra: Objektumok színsémája......59
6.34. ábra: Felhasználó által definiált szín-összeállítások......59
6.35. ábra: Kapcsolási rajz alapú szerkesztő beállításai......59
6.36. ábra: Kapcsolási rajz ellenőrzésének beállításai......60
6.37. ábra: A szerkesztő színeinek beállítása......61
6.38. ábra: Eszközcsaládok......61
6.39. ábra: Rétegek beállításai......62
6.40. ábra: Nyomtatási beállítások......62
6.41. ábra: Lapméretek beállításai......63
6.42. ábra: Szimbólumszerkesztő beállításai......63
6.43. ábra: Szimbólumszerkesztő ellenőrzésének beállításai......64
6.44. ábra: Szimbólumszerkesztő színeinek beállítása......64
6.45. ábra: Időzítés analizátor beállításai......64
6.46. ábra: WebTalk beállításai......65
6.47. ábra: XilinxNotify beállításai......65
6.48. ábra: Proxy beállítások......65
6.49. ábra: Nézet menü......66
6.50. ábra: Panelek menüpont......67
6.51. ábra: Start panel......67
6.52. ábra: Options panel......67
6.53. ábra: Symbols panel......68
6.54. ábra: Design panel......68
6.55. ábra. Libraries panel......69
6.56. ábra: Files panel......69
6.57. ábra: Timing panel......69
6.58. ábra: Egyéb panelek......70
6.59. ábra: Eszköztárak......70
6.60. ábra: Transcript menüpont......70
6.61. ábra: Statusbar......70

6.62. ábra: Zoom menüpont..70
6.63. ábra: Fájlnév/útvonal mutatása menüpont...71
6.64. ábra: Projekt menü..72
6.65. ábra: Forrás menü...73
6.66. ábra: Folyamat menü...73
6.67. ábra: Eszközök menü...74
6.68. ábra: Constraints Editor...74
6.69. ábra: PlanAhead menüpont..75
6.70. ábra: Schematic Viewer menüpont..75
6.71. ábra: Timing Analyzer menüpont..75
6.72. ábra: FPGA Editor...75
6.73. ábra: SmartXplorer menüpont...76
6.74. ábra: Ablak menü..76
6.75. ábra: Réteg menü...77
6.76. ábra: Súgó menü..77
6.77. ábra: Webes támogatás..78
8.1. ábra: VHDL modell..84

Betűrendes tárgymutató