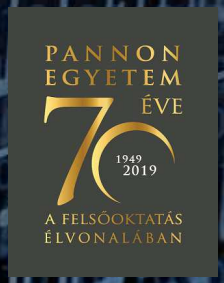
Pannon Egyetem, MIK-VIRT, Veszprém





Dr. Vörösházi Zsolt
voroshazi.zsolt@virt.uni-pannon.hu

Tervezési módszerek programozható logikai eszközökkel

2. Tervezési módszerek. Tervezés folyamata.
HLS: magas szintű szintézis. Programozási módok. Xilinx Vivado használata.



Frissítve: 2020. 09. 15.

Tárgyalt ismeretkörök
2. előadás

I.) Tervezési módszerek

II.) Tervezés folyamata (Design-flow)

III.) HLS: High Level Synthesis - Magas-szintű szintézis (CFG, DFG)

IV.) FPGA-k integrált fejlesztő környezetei

V.) Xilinx Vivado Design Suite (2020.1)



VI.) Digilent Adept Suite (csak konfigurálás, vagy kártya diagnosztika)

Szempontok, absztrakciós szintek, modellezés

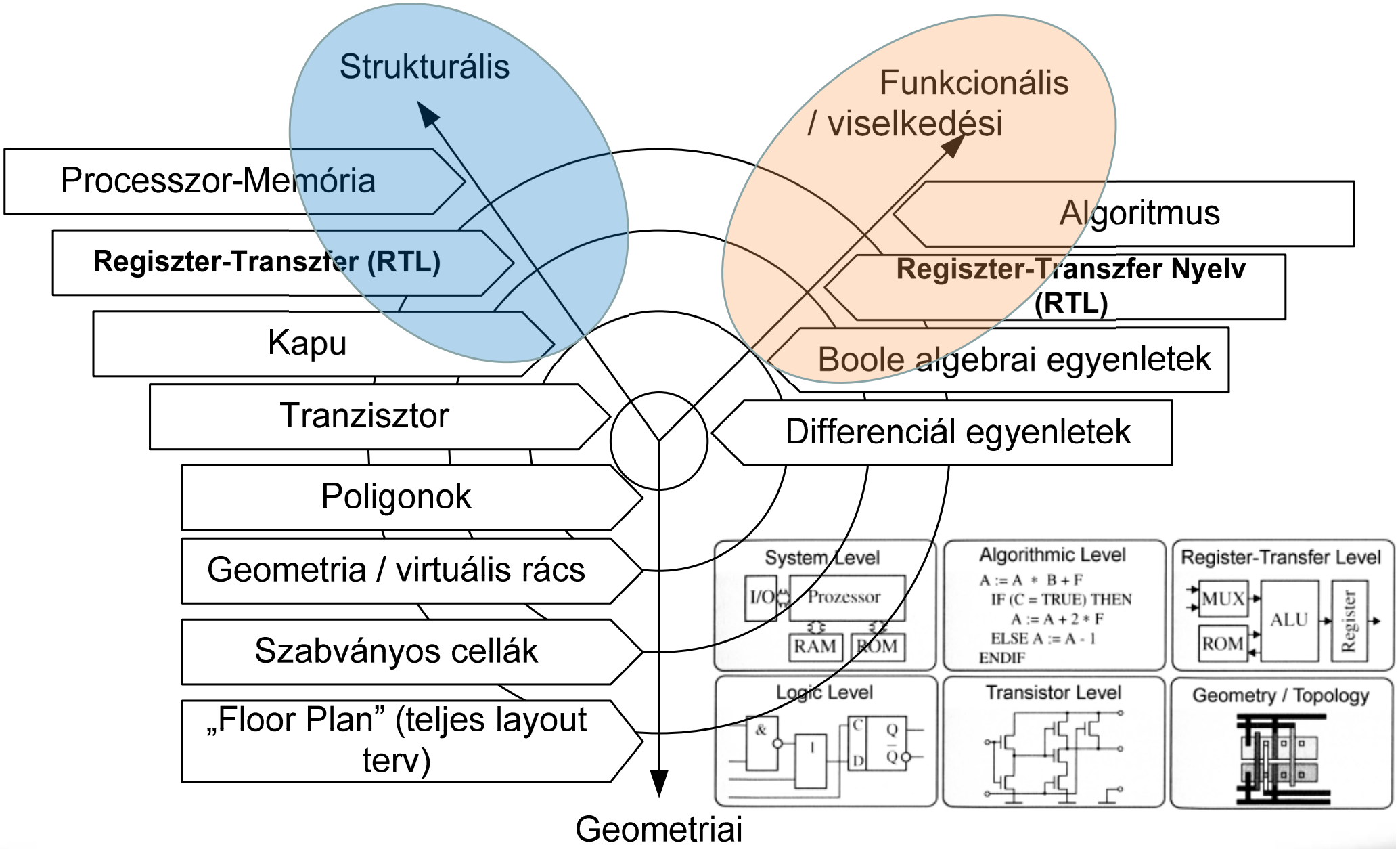
I. TERVEZÉSI MÓDSZEREK

Tartományok – absztrakciós szintek

- Különböző *tervezési szempontok* (metodika) szerint dolgozhatunk, azaz a rendszer modellezését különféle **tartományokon** és azokon belül eltérő **absztrakciós szinteken végezhetjük**. Az ismert *Gajski és Kuhn féle Y-diagram* szerint 3 lehetséges tartományt különböztethetünk meg*:*
  - *Funkcionális/viselkedési,*
  - *Strukturális és*
  - *Geometriai*.

  (későbbiekben: HDL leírások során, általánosan a viselkedési és a strukturális tartomány szerinti tervezést alkalmazzuk)

Gajski-Kuhn - „Y-diagram”



Funkcionális tartomány

A rendszer **viselkedésének** leírását adjuk meg, de nem foglalkozunk annak belső részleteivel (felépítésével).

- A funkcionális tartomány a ’leg-absztraktabb’ megközelítést jelenti, ahol a teljes rendszer viselkedése megadható az **algoritmikus szinten**.
- A következő szint az RT, azaz **regiszter-átviteli szint** (**Register-Transfer**), vagyis a *regiszter-memória-processzor* elemek közötti transzformációk megadása. Az adatot egy regiszter, vagy egy memória adott cellájának értéke, míg a transzformációkat az aritmetikai és logikai *operátorok* jellemezhetik. Az adat-, és vezérlési információ áramlása definiálható akár a regiszter-transzfer szintű nyelvi leírásokkal (RT Language) is, vagy hatékonyan szemléltethetők grafikus módon az **adat-, és vezérlési-folyam gráfok segítségével (DFG, CFG).**
- A harmadik szint a hagyományos logikai szint, ahol egy-egy funkció megadható a **Boole-algebrai** kifejezések, igazság táblázatok segítségével.
- Végül az utolsó szinten az áramkör működését definiáló **differenciál-egyenleteket** kell definiálni, amely pl. a hálózat feszültségében, áramában bekövetkező *változásokat* hivatott leírni.

Strukturális tartomány

A **strukturális tartományban** viszont pontosan a rendszer *belső elemeinek felépítését* és közöttük lévő *összeköttetéseket* vizsgáljuk.

- A strukturális tartomány legfelső szintjén kell a fő komponenseket és összeköttetéseit definiálni, amelyet **processzor-memória kapcsolatnak (processor-memory** switch) is hívnak.
- A második szint itt is a **regiszter-átviteli szint (RTL)**, amely egyrészt az adatútból (*data path*), másrészt vezérlési szakaszokból (control path, sequence), szekvenciákból áll.
- A harmadik szint a **logikai szint**, amelyben a rendszer struktúrája az alapkapuk és összeköttetéseik segítségével építhető fel.
- A negyedik, legalacsonyabb szinten a **tranzisztorok**, mint az *áramköri rajzolatok (layout)* elemi egységeit kell tekinteni (azokból építkezni).

Geometriai tartomány

Végül a **geometriai tartomány** azt mutatja, ahogyan a rendszert elhelyezzük, *leképezzük* (=*MAPPING*) a rendelkezésre álló fizikai erőforrások felhasználásával (*felület*).

- A legfelső hierarchia szinten, a *szilícium felületen elhelyezett* vagy ún. „kiterített” VLSI áramkört kell tekinteni **(floor-plan**): FPGA esetén tehát magukat a *logikai cellákat* és *makrocellákat*, valamint a közöttük lévő *összeköttetéseket* (lásd FPGA felépítés)
- A következő szintet a **szabványos alapcellák (Standard Cell)** *könyvtárai* képezhetik, amelyeket, mint *technológiai adatbázist* használhatunk fel a regiszterek, memóriák, vagy akár aritmetikai-logikai egységek megvalósításához.
- A harmadik szinten az egyedi tervezésű integrált áramkörök (ASIC) **geometriája** egy virtuális rácson adható meg.
- Végül az utolsó, legalacsonyabb szinten a **poligonokat** kell megrajzolni, amelyek csoportjai az áramkör egyes maszk-rétegeinek feleltethetők meg.

EDA: **E**lectronic **D**esign **A**utomation

- Manapság a számítógéppel segített (CAD) elektronikus tervezői, fejlesztői és szimulációs eszközök széles skálája áll a rendelkezésünkre.
- Segítségükkel egy-egy tartományon belül nem kell minden szintet külön-külön pontosan definiálni (sokszor nem is lehet), elegendő a **tervezést a legfelsőbb absztrakciós szinteken** elvégezni (*design entry*) → amelyből az alacsonyabb szintek automatikusan generálhatók (**EDA**).

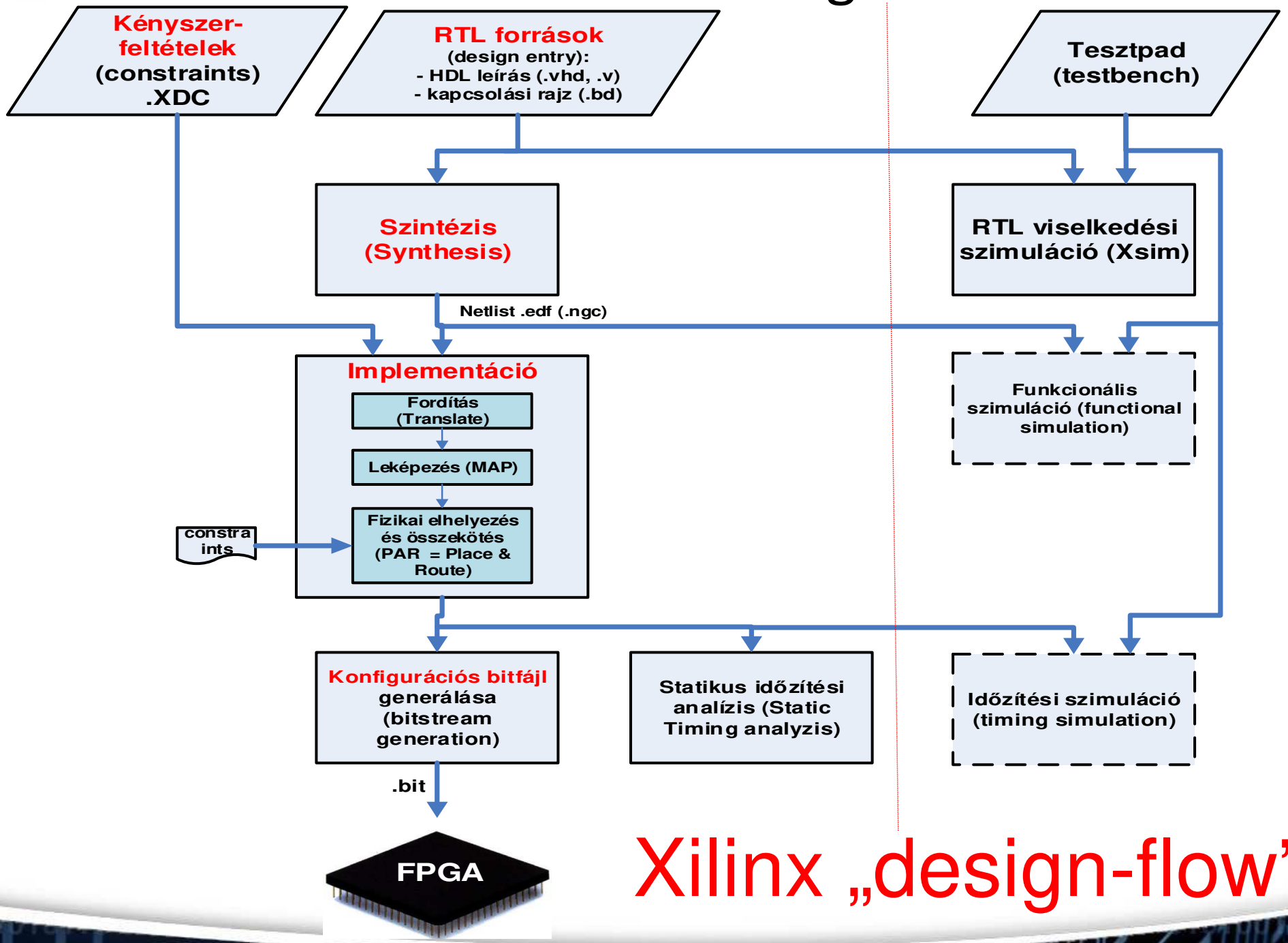
„DESIGN FLOW„ (XILINX FPGA-k esetében)

II. TERVEZÉS FOLYAMATA

Xilinx „design-flow”

- Az FPGA alapú rendszertervezés folyamatát a Xilinx **Vivado**/ISE fejlesztő környezetben történő egymást követő **lépéseken** keresztül demonstráljuk. A tervezés egyszerűsített folyamatát a következő oldalon lévő ábra szemlélteti. Lépései:
  - 1.) Design Entry – Tervezés „belépési” szintje (pl. HDL forrás)
  - 2.) Szimuláció
  - 3.) a.) Szintézis és b.) Implementáció
  - 4.) Időzítési analízis
  - 5.) Bitfolyam (konfiguráció) létrehozása

Xilinx Vivado - Design Flow



Xilinx „design-flow”

A tervezés fontosabb lépései (I.):

- **1.) Moduláris, vagy komponens alapú (hierarchikus) rendszertervezés**
  - Lentről-felfelé (bottom-up), vagy fentről-lefelé (top-down) irányú tervezési metodika alkalmazása
  - HDL leírások, kapcsolási rajzok, vagy állapot diagramok megadása a tervezés kezdeti fázisában (=design entry), illetve
  - felhasználói-tervezési megkötések, kényszerfeltételek (user constraints – ***XDC***) rögzítése (lásd később a szintézis és implementáció során),
- **2.) Szimuláció:** párhuzamosan a tervezés egyes szintjein, illetve a legfelsőbb hierarchia szinten **HDL tesztkörnyezet,** tesztágy vagy **tesztpad (testbench)** összeállítása = **szimulációs modell**
  - RTL / viselkedési szimuláció tesztvektorokkal, amely még PC-n történik

A tervezés fontosabb lépései (II.):

- **3.) a.) Szintézis és b.) implementáció!:**
  - **Szintézis:** „logikai szintézis” során a HDL leírás még csak *általános kapu-szintű* komponensekké transzformálódik (pl. logikai kapuk, FFs)
  - **Implementáció** 3 fő lépésből áll: ***TRANSLATE (Compiler) → MAP (Fit) → PAR (Placer & Router / Assembler)*** lépéseinek sorozata. Ezeket a kifejezéseket a Xilinx FPGA rendszer-fejlesztési folyamatán kívül is hasonlóan nevezik. Ha az implementáció bármely adott lépésében hiba történt, akkor az implementáció további lépései már végre sem hajtódnak. Az adott lépésben meglévő hibát először ki kell javítani – ezt az Xilinx Vivado üzenet ablakában megjelenített információk alapján láthatjuk - és csak utána tudjuk a további implementációs lépéseket végrehajtani.

A tervezés fontosabb lépései (III.):

Implementációs fázisok (3./b. részletezve):

- **TRANSLATE**: több, esetleg eltérő hardver leíró nyelven megírt tervezői file összerendelése (merge) egyetlen *netlist*-fájlba (*EDIF*). A netlista fájl tartalmazza a komponensek és összeköttetéseik szabványos szöveges leírását.
- **MAP**: a szintézis során elkészült „*logikai*” tervnek egy adott technológia szerinti „*leképezése*” (*technology mapping*), amelyet az előző lépésben kapott EDIF netlistából a kapukból pl. CLB-ket, ill. IOB-kat képez le.
- **PAR**: végleges „*fizikai*” áramköri tervet hoz létre periféria kényszerfeltételeitől függően (.PCF – peripheral constraint file), amely fázisban a komponenseket az **fizikailag** az FPGA-n meglévő és azonosítható cellákon **helyezi el** (pl. XnYm). Az elhelyezett fizikai komponenseket végül a ‚*routing*’ eljárás *összehuzalozza*, az egyes tervezési megkötéseket, kényszer-feltételeket figyelembe véve (.PCF). Kimenetén egy .NCD fájl jön létre.

A tervezés fontosabb lépései (IV.):

- **4.) Statikus időzítési analízis**: időzítési paraméterek (timing parameters) meghatározása (max. órajel frekvencia, kapuk megszólalási idejének, vezetékek jelterjedési késleltetések hatásának vizsgálata stb.)
- **5.) Bit-folyam (bit-stream),** mint *konfigurációs* fájl generálása (.BIT) és letöltése FPGA-ra (CLB-k, programozható összeköttetések beállítása, konfigurálása minden egyes inicializációs fázis után szükséges, köszönhetően annak, hogy a Xilinx FPGA-kat alapvetően az SRAM alapú technológia jellemzi).

Általános ismertetés

III. HLS: HIGH-LEVEL SYNTHESIS, MAGAS-SZINTŰ SZINTÉZIS ALAPJAI

Elméleti háttér

- ***Turing-Church tézis***: HW – SW ekvivalencia.
  - a μ-rekurzív függvények (utasítások),
  - környezet független nyelvtan (algoritmus modell, pl. CFG), ill.
  - TM – Turing Machine (HW)

  egymásba ekvivalens módon transzformálhatóak.

- **Tervezés:** a valós rendszerek felépítéséhez különböző modellek szükségesek. Egy valós modellt implementálhatunk a számítógéppel segített (CAD tool) tervező rendszerrel: a struktúráját felépítjük, ill. viselkedését (behaviour) is tudjuk szimulálni.
- **Folyamatok:**
  - Tervezés (CAD): információ-feldolgozó folyamat: Δ
  - Gyártás (CAM): π
  - Tesztelés v. Verifikáció (CAE) : τ

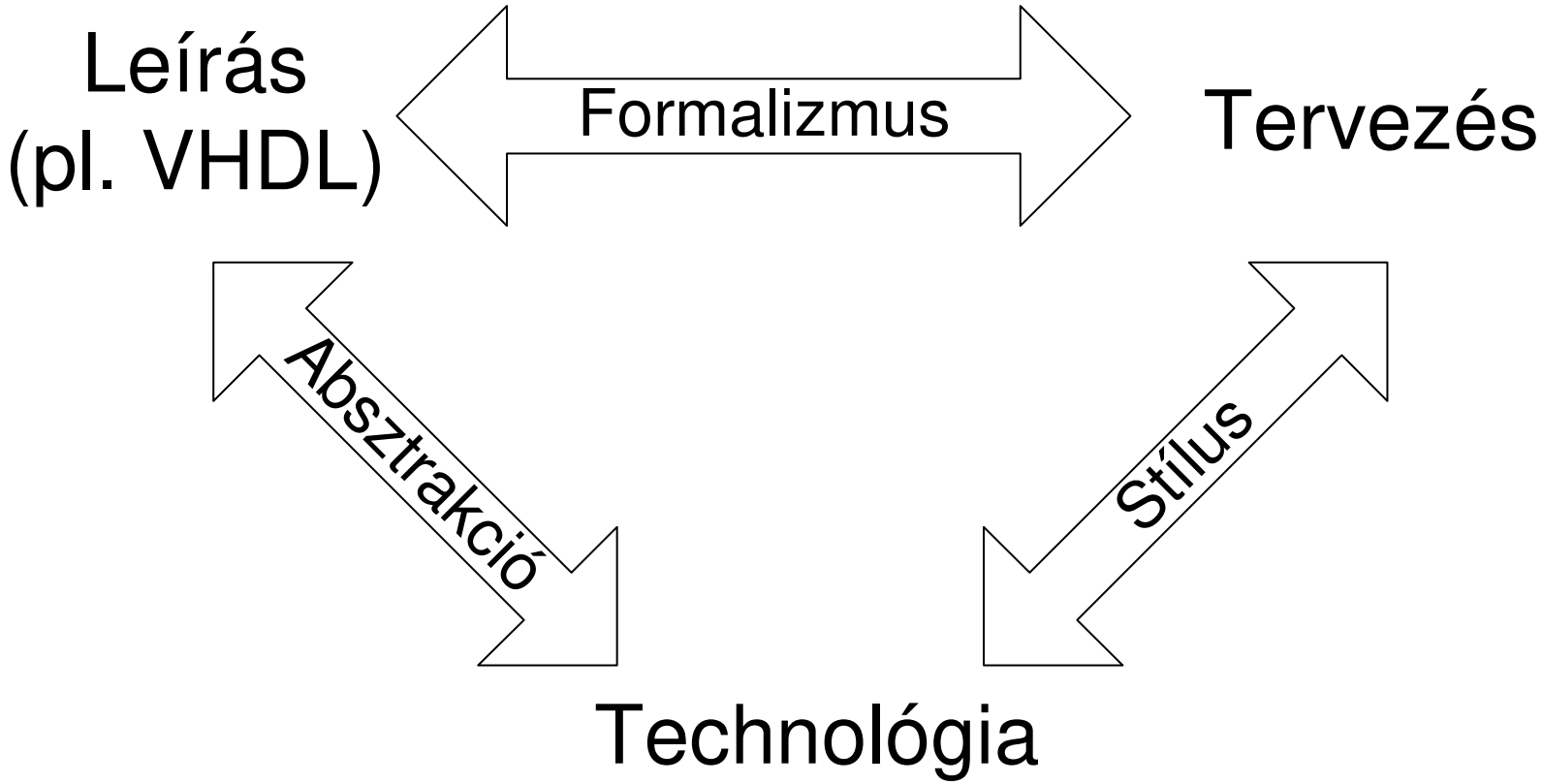
HLS: High Level Synthesis

- **HLS célja:**
  - *tervezés automatizálásának (EDA)* vizsgálata kombinációs- és, sorrendi digitális hálózatok (KH, SH) esetén, méghozzá a logikai (Boole-algebrai) szint feletti hierarchia szinten definiált gyakorlati környezetben,
  - Rendszer megvalósítása *gyorsan*, akár komplex rendszerek *szintézisére* is (elfogadható időben): „rapid prototyping, time-to-market” fogalmak,
  - *Bemenete* magas szintű hardver leíró nyelv (pl. konvencionális VHDL, vagy Verilog), → amelynek segítségével a rendelkezésre álló adatbázisok (DB – tervezői alap építőelem könyvtárak) alapján elektromos áramköröket tervezhetünk pl. újrakonfigurálható CPLD, FPGA áramkörökön.

- „Magas szintű szintézis” – fő lépései a fókuszban:
  - Tervezés-Ütemezés (SCHEDULING)
  - Erőforrás foglalás / feladat felosztás - (ALLOCATION)
  - Kötött algoritmusok (BINDING algorithms) – megkötések (constraints)

Erős korreláció:

Stílus – Absztrakció – Formalizmus



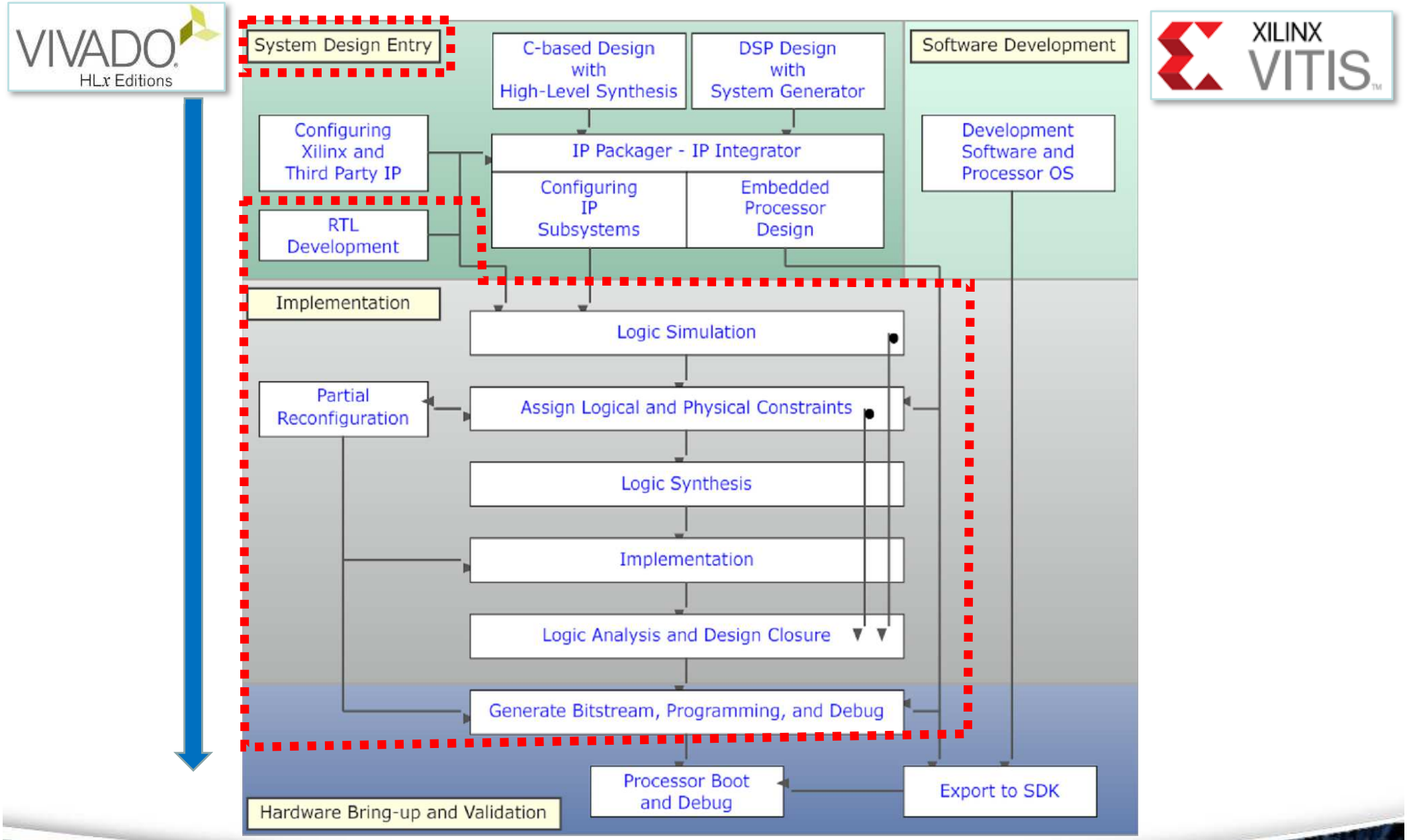
1. Stílus

- Komplex feladat $\Rightarrow$ egyszerűbb, kezelhető részfeladatokra bontása
  - Szisztematikus
  - Érthető módszerek kellenek

  Pl: programozási stílusok (Top-down, strukturált)

- Jó stílus kialakításának szabályai:
  - „Top-down”/Bottom-up módszer szerinti tervezés
  - Csak kiforrott, biztos technikákat szabad alkalmazni
  - Fontos a dokumentálás!
  - Stílushoz soroljuk a tervezés menetét (design-flow)

Xilinx Vivado Design Flow

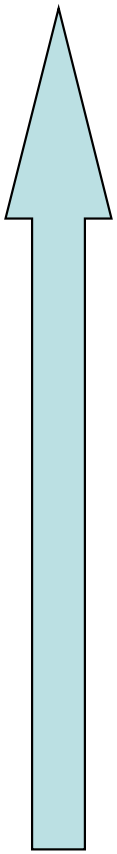


2. Absztrakció

- Digitális tervezés „elvi-fogalmi” szintje
  - Kezdeti absztrakció a tervezés során meghatározó, kritikus pont!
    - 1. koncepcionális modell (elvi elgondolás)
    - 2. megvalósítható, realizálható modell (HW)
  - Magas-szintű absztrakció $\Rightarrow$ elvi modell szintenkénti finomítása és felépítése

Absztrakciós szintek

- Rendszer szint (legfelsőbb szint)
- Algoritmikus szint
- Funkcionális szint (pl. multiplexer, dekóder, ALU stb.) – ***RTL szint*** (VHDL, Verilog stb.)
- Logikai szint (kapuk – Boole algebra)
- Fizikai áramkör szint (tranzisztor szint) –
  - erősen gyártás-technológia függő – pl. deep submicron technology)



3. Formalizmus

- A rendszer viselkedésének **leírására** szolgál (HDL)
  - Szisztematikus szabályok és eljárások
  - Minden absztrakciós szinten fontos a használatuk
    - Pl: alapvető formalizmus a Boole-algebra (bináris logika elmélete) – de csak alsóbb szinteken használható

- (felsőbb-, rendszer-szint) (alsóbb-, áramköri szint)

  Absztrakció ⇔ Boole algebra

  (konkretizálás)

VHDL-ről röviden

- **VHDL = VHSIC HDL** (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit **H**ardware **D**escription **L**anguage) nevű nyelv a 80-as évek közepén alakult ki, melyet azóta nemzetközi szabványnak is elfogadtak (*IEEE-1076* 1987 és ANSI 1989).
  - biztosítja az egyes rétegnek megfelelően egy szabványos „nyelven” a hardver leírását,
  - a magasabb szinteken az ember számára is áttekinthetővé teszi a hardver dokumentációját,
  - az alsó szinteken lehetővé teszi a tervek átadását a gyártás CAD/CAM rendszerei felé,
  - biztosítja a hardver szimulációs ellenőrzését (verifikációt).

VHDL

- A VHDL nyelvű hardver leírás leginkább egy emberi nyelvhez közel álló, magas szintű programnyelvhez hasonlítható. A VHDL hardver leíró nyelv három alapegysége
  - architektúra (architecture): HW egység funkciója és szerkezete
  - interfész (entity): HW egységek közötti kommunikáció (protokoll) jellemzői
  - konfiguráció (configuration): architektúrák és interfészek egymáshoz rendelése

```
ENTITY or_gate IS
        PORT (a, b: IN bit; y: OUT bit)
END or_gate
ARCHITECTURE viselkedes OF or_gate IS
BEGIN
        y <= a OR b
END viselkedes
CONFIGURATION or2_konfig OF or_gate IS
        FOR viselkedes
        END FOR
END or2_konfig
```

HLS során használt algoritmusok

1.) In –core modell szintézis: a bemeneti magas-szintű nyelvből **gráf-reprezentációt** állíthat elő, amely a vezérlést és az adatokat külön-külön kezeli. (Lásd CFG vs. DFG).

2.) Adatfolyam analízis. Ez a módszer határozza meg a **változók értékének élettartamát**, amelyeket sokszor használunk a magas-szintű fordítónál, továbbá meghatározza rendeltetésszerű viselkedést (változók kiértékelése). Allokáció és ütemezés (scheduling) során is használjuk.

3.) A lehető leggyorsabb SCHEDULING / ütemezés (ASAP): a gyors végrehajtás eléréséhez **minimalizálja** a lehetséges útvonalak számát (azaz minimalizálja az utasításokban szereplő ciklusok számát). Optimalizálja az utak hosszát, amely az egyes vezérlési lépések számát jelenti. Az FSM (véges állapotú gép) készítésekor a CFG-t körmentessé teszi a visszacsatolások elhagyásával. Lehetnek feltételes elágazások. Az utak száma a csomópontok számának növelésével exponenciálisan nő.

4.) Modul hozzárendelés (MAP): A modul hozzárendelési probléma akkor lép fel, amikor a műveleteket többfajta funkcionális egységgel (FU: Functional Unit) kell megvalósítani, mivel ez a hozzárendelés választja ki a megfelelő FU-t az adott utasítás elvégzéséhez.

5.) Kezdeti (internal based) ALLOKÁCIÓ: a teljes kezdő adatút (data-path) lévő blokkok generálása (lefoglalása), majd optimalizálása - Place and Route

6.) Késleltetés és méret becslése: logikai minimalizálás (vezérlőjelek számának csökkentése, ha lehetséges), ill. kimenetek generálása

SYNTHESIS „IN-CORE” MODEL

- Magas-szintű nyelvből gráf reprezentáció
  - a.) Vezérlés folyam gráf (CFG)
  - b.) Adatfolyam gráf (DFG)
- A két gráf-reprezentáció között az egyes állapotokban lehetnek csatolások, *linkek* is.

a.) CFG: Control Flow Graph

- Irányított gráf, CFG=(N,P)
- Csúcsok halmaza (N):
  - hozzárendelés,
  - összeadás,
  - logikai műveletek, stb.
- Élek halmaza (P):
  - Precedencia relációk,
  - egymást követő utasítások
- Összefüggő gráf (nem aciklikus!)

a.) CFG (folyt.)

- Sorozat, szekvencia: egy él (*n1,n2*)$\in$ P azt jelenti, hogy *n2* következik *n1* végrehajtása után
- Feltételes végrehajtás (branch, if): egy művelet után egy másik művelet, ha a feltétel teljesül.
  - A feltétel Boolean kifejezés, melynek értéke ‘1’, akkor végrehajtódik, különben ‘0’.
- Iteráció (loop): Ciklusok, amelyek a folyamat iteratív viselkedését jelzik

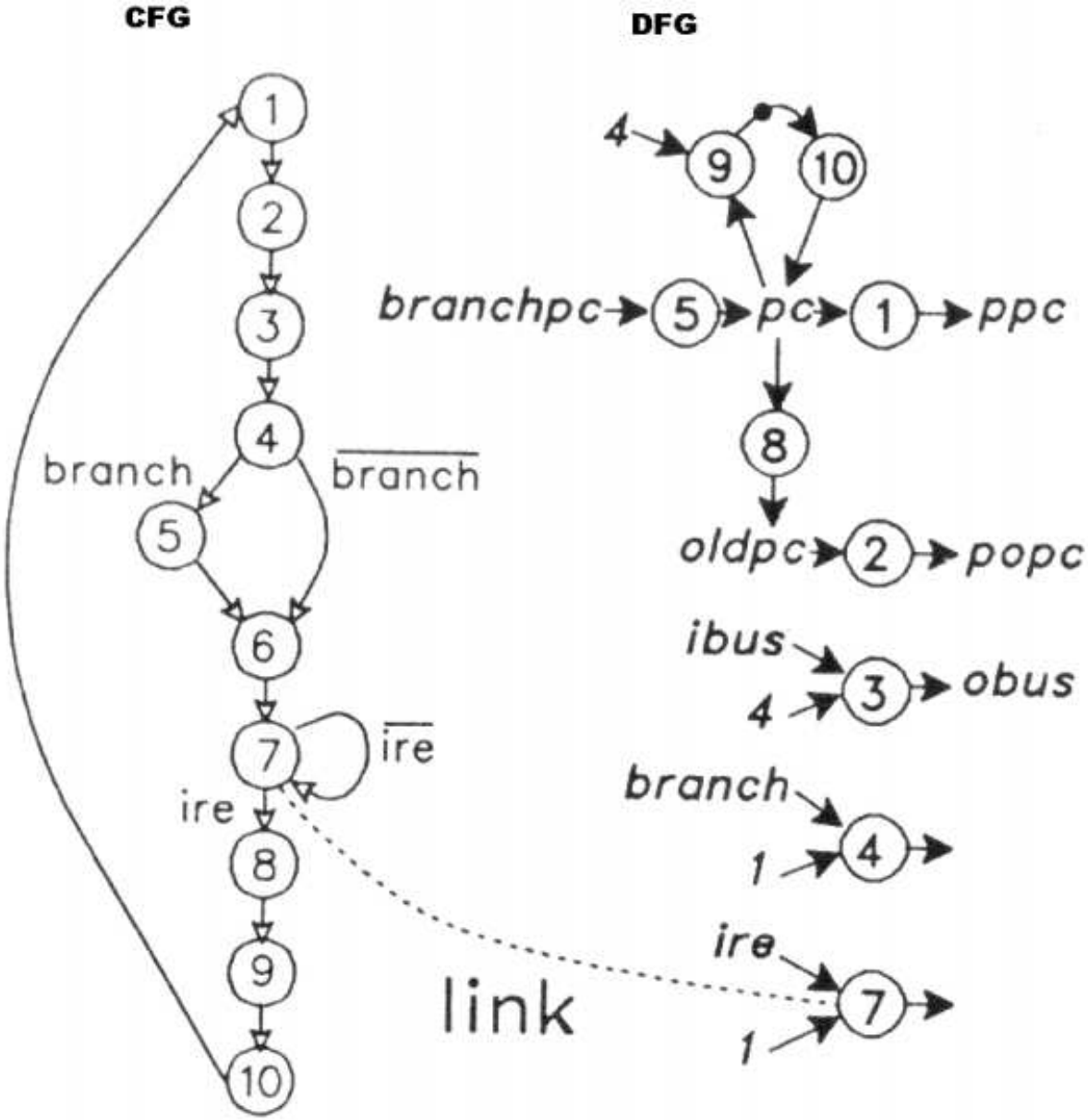
b.) DFG: Data Flow Graph

- Irányított gráf, DFG=($N \cup V, D$)
- Csúcsok halmaza:
  - N: utasítások
  - V: változók
- Élek halmaza (D):
  - Adatkapcsolatok
- Fontos: nem feltétlenül összefüggő gráf, azaz a DFG-nél az egyes műveletek lehetnek ábrázolva úgy is, hogy nincs közöttük kapcsolódási pont (nem húzódik él), tehát részekre lehet bontani.

Példa #1: CFG vs. DFG

Csak a hozzárendelési utasítások megszámozása (begin-end részen) szükséges.

Process-en belüli részek sorrendben (egymás után hajtódnak végre)



VHDL leírás:

```
Entity prefetch is
      Port ( branchpc, ibus in bit32;
             branch, ire: in bit;
             ppc, popc, obus: out bit32);
end prefetch;

architecture behavior of prefetch is
begin
      process
      variable pc, oldpc: bit32:=0;
        begin
             ppc <= pc;                --1
             popc <= oldpc;            --2
             obus <= ibus + 4;         --3
             if (branch = '1') then    --4
                   pc:= branchpc;      --5
             end if;                   --6
           wait until (ire='1');       --7
           oldpc:=pc;                  --8
           pc:=pc+4;                   --9, 10
        end process;
end behavior;
```

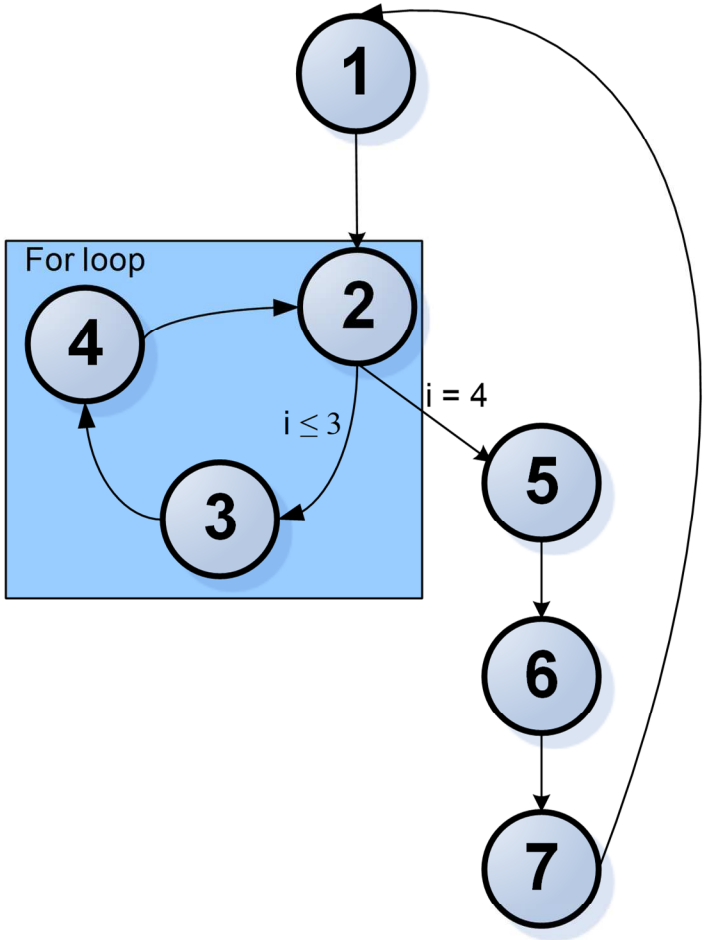
Példa #2: VHDL → CFG, DFG

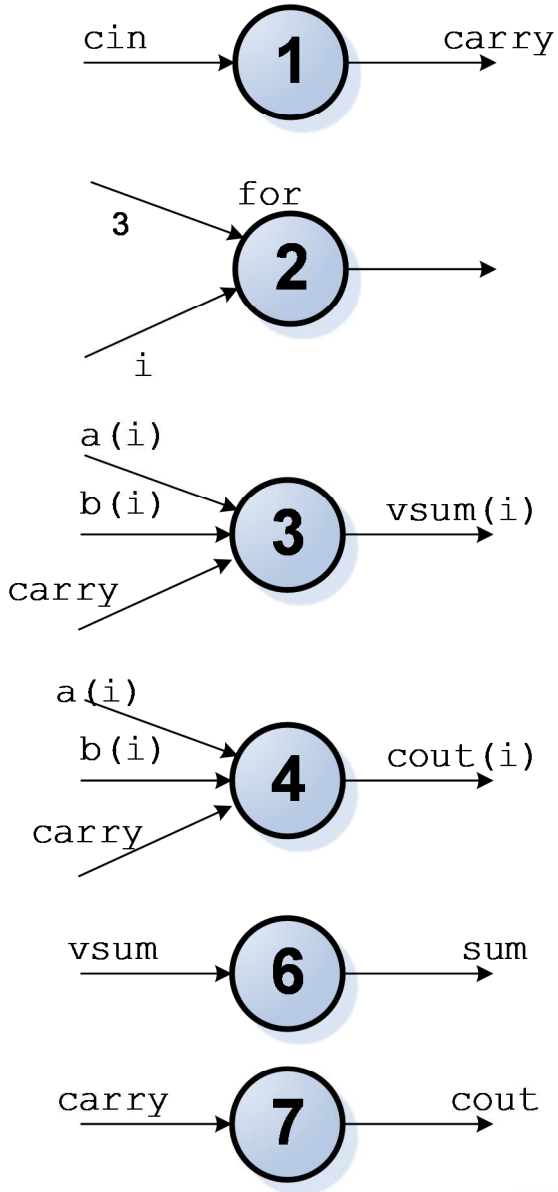
```
architecture behavioral of MODULE is
begin
        p1: process(a, b, cin)
        variable vsum : std_logic_vector(3 downto 0);
        variable carry : std_logic;
        begin
            carry := cin;                                                    --1
            for i in 0 to 3 loop                                             --2
                vsum(i) := (a(i) xor b(i)) xor carry;                        --3
                carry := (a(i) and b(i)) or (carry and (a(i) or b(i)));      --4
            end loop;                                                        --5
            sum <= vsum;                                                     --6
            cout <= carry;                                                   --7
        end process p1;
end behavioral;
```

Csak az Utasítások megszámozása!

Megj. Ez egy 4-bites RCA (4 Full Adder sorba kapcsolását 0..3-ig) valósítja meg!

Megoldás #2: VHDL → CFG, DFG



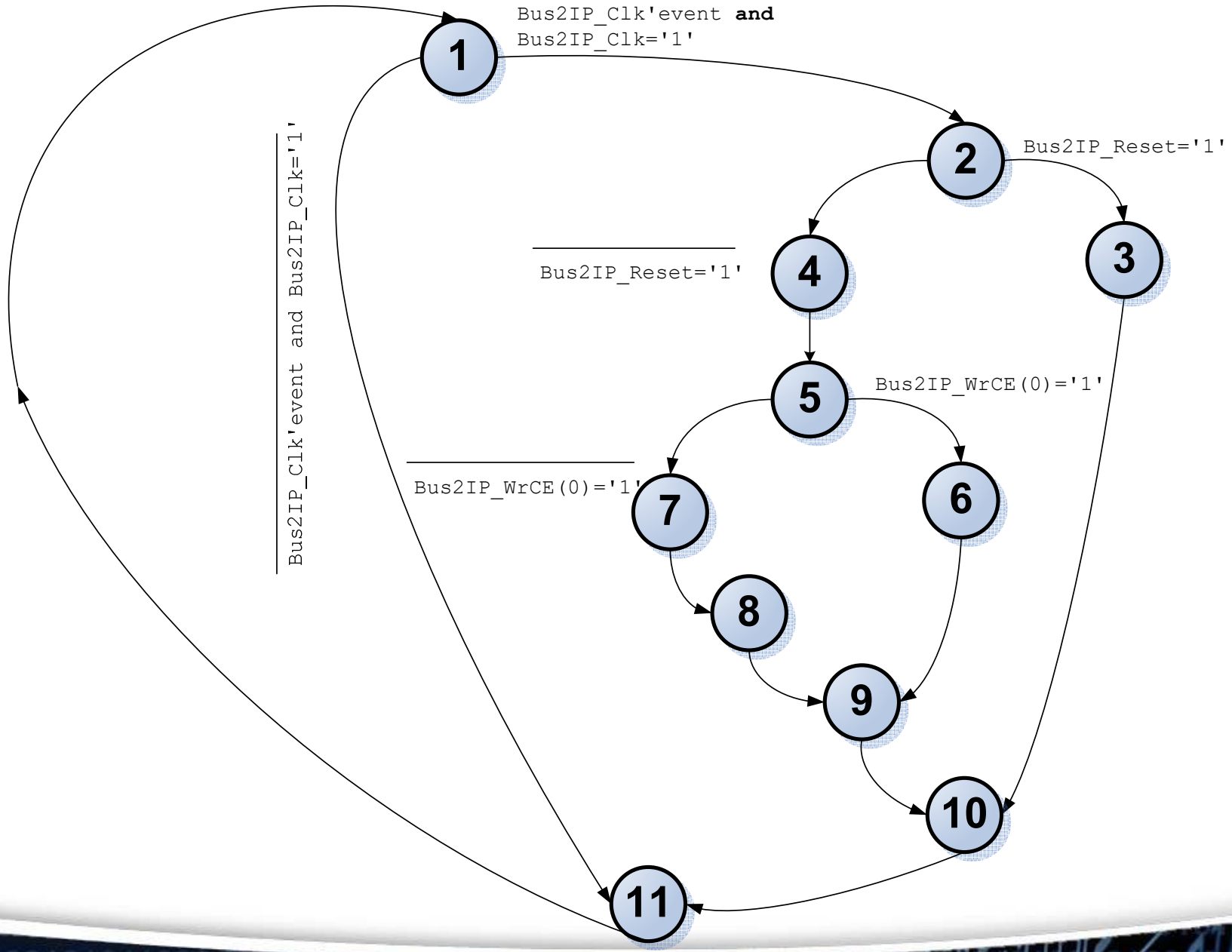


Példa #3: VHDL → CFG

Csak az Utasítások megszámozása!

```
 -- behavioral implementation of the LED_proc
architecture behavioral of LED is
begin
   LED_PROC: process (Bus2IP_Clk) is
   begin
     if Bus2IP_Clk'event and Bus2IP_Clk='1' then          --1
      if Bus2IP_Reset='1' then                            --2
         LED_i<="0000";                                   --3
      else                                                --4
         if Bus2IP_WrCE(0)='1' then                       --5
            LED_i<=Bus2IP_Data(0 to 3);                   --6
         else                                             --7
            LED_i<=LED_i;                                 --8
         end if;                                          --9
      end if;                                             --10
     end if;                                              --11
   end process LED_PROC;
end behavioral;
```

Megoldás #3: VHDL → CFG





Áttekintés

IV. FPGA-K FEJLESZTŐ KÖRNYEZETEI

FPGA-k lehetséges „programozási nyelvei” / fejlesztő környezetei:

- **I.) Hagyományos HDL nyelvek (HDL → FPGA szintézis):**
  - VHDL,
  - Verilog HDL (System Verilog)
- **II.) C-alapú nyelvek (C → FPGA szintézis):**
  - a.) Xilinx Vivado HLS/HLX, Xilinx VITIS 2020.1 (unified software environment)
  - b.) Intel FPGA SDK OpenCL, Intel HLS (20.2)
  - Catapult-C, (régen Mentor Graphics, ma Calypto)
  - Impulse-C, System-C, Mitrion-C, stb.
- **III) Modell alapú nyelvek (modell → FPGA szintézis):**
  - a.) Matlab Simulink,
  - b.) NI LabView (FPGA Modul)

Hagyományos HDL nyelvek célja I.

- Hardver modellezés (korábban dokumentáció),
- A nyelv elemkészlet jelentős része,
  - csak a hardver funkciók modellezésére ill. PC-n történő szimulációra használható, ill.
- A nyelvi elemkészlet bővebb tartománya
  - Szintézisre és implementációra is használható (FPGA-n futtatható), azonban
  - Szintetizálható részhalmaz szintézis eszköz függő (pl: **XST = Xilinx Synthesis Tool**), „primitívek” alkalmazása révén
- Kapuszintű modulokból építkező, kapcsolási rajzon alapuló tervezési módszerek leváltása,
- **RTL** (Register-Transfer Level) szintű leírás: **VHDL, Verilog**
- Automatikus hardver szintézis a leírásból,
- Tervezői hatékonyság növelése.

HDL nyelvek II.

- Alapvetően **moduláris** és egyben **hierarchikus** felépítésű tervezést tesz lehetővé
- HDL modul = entitás / modul
  - Be-, kimenetek, illetve kétirányú port-ok definiálása
  - Be-, kimenetek közötti logikai kapcsolatok és időzítések definiálása
- NEM szoftver!!
  - Nem utasítások sorozatára hanem HW erőforrásokra képződik le
- Alapvetően **időben párhuzamos, konkurens** működést ír le (gondoljunk az FPGA nagyszámú, párhuzamosan elérhető blokkjainak felépítésére)
  - Ha külön definiáljuk egy eljárásban (process, always), lévő utasításokat fogja csak szekvenciálisan végrehajtani!

HDL nyelvek gyengeségei

– HDL nyelveken történő fejlesztés a hagyományos szoftver fejlesztéshez viszonyítva továbbra is *időigényes*,

– Sok SW fejlesztő rendelkezik C/C++ ismerettel, azonban kevesen HDL tapasztalattal, amely a használatának egyik fő korlátja lehet.



Éppen ezért születtek meg napjainkra a *magas-szintű C szintaktikát követő nyelvek, valamint a Modell-alapú nyelvek/környezetek:*

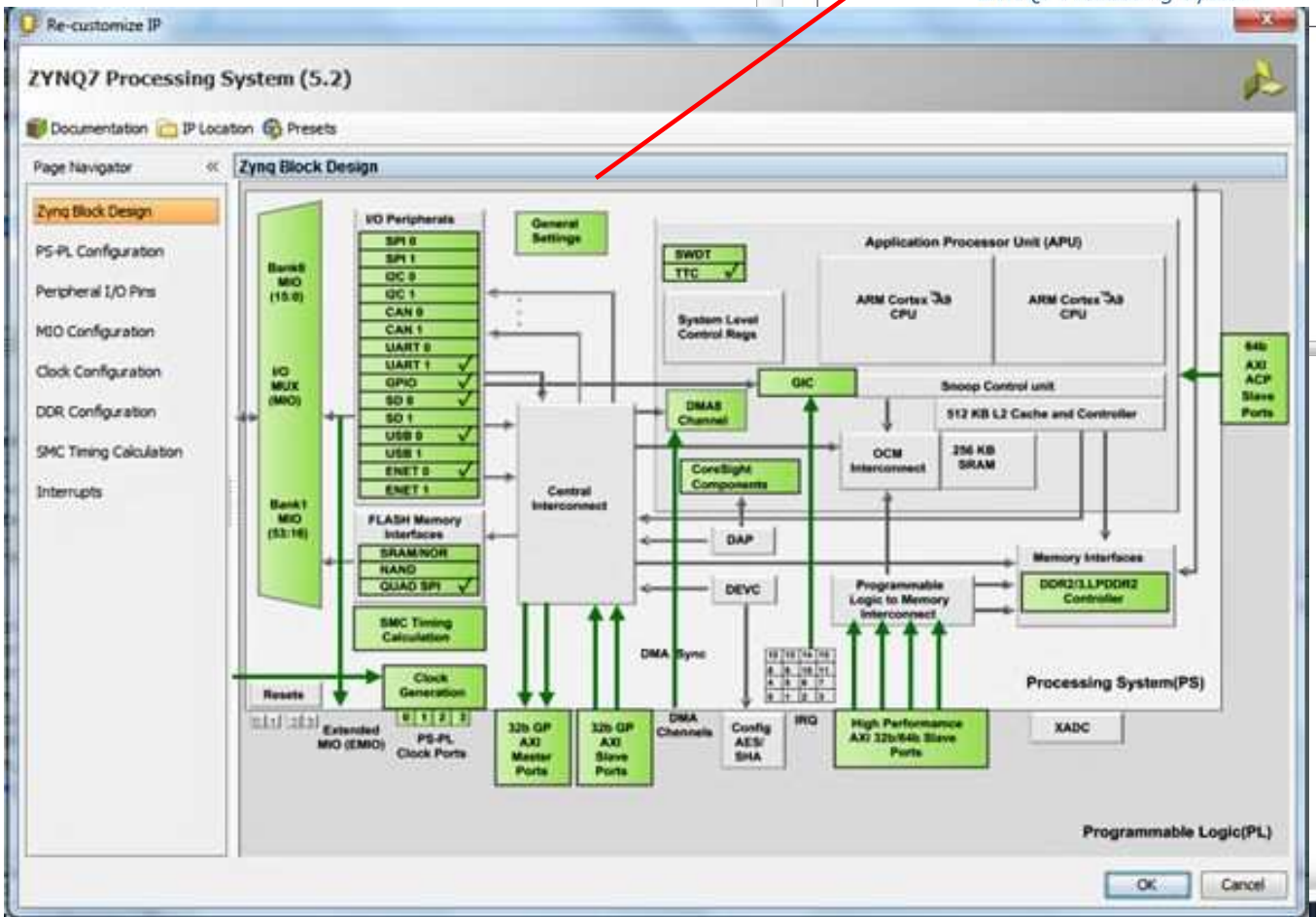
+ Gyorsabb prototípus fejlesztés, szimuláció/verifikáció

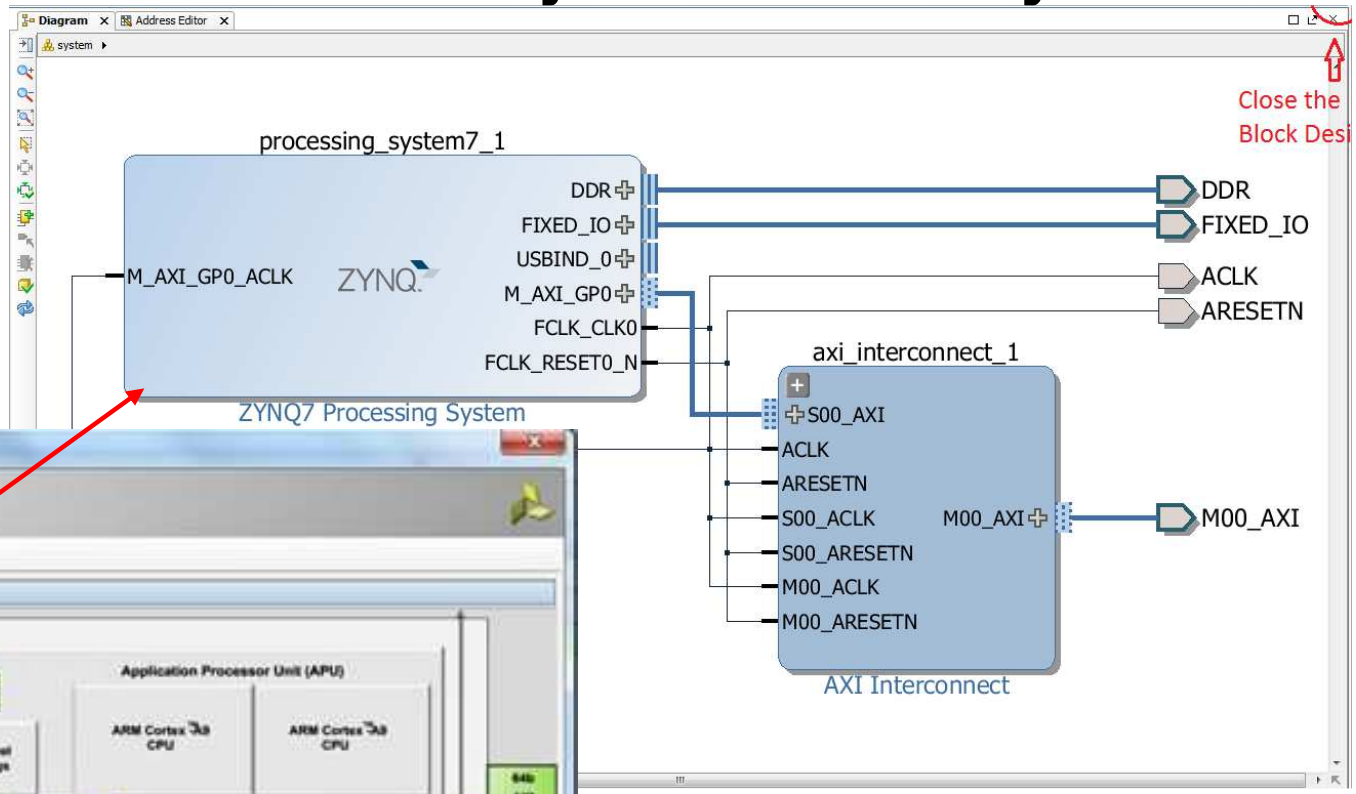
+ HW(FW) / SW együttes tervezés, szimuláció és verifikáció

Intel FPGA vs. Xilinx fejlesztő környezetek

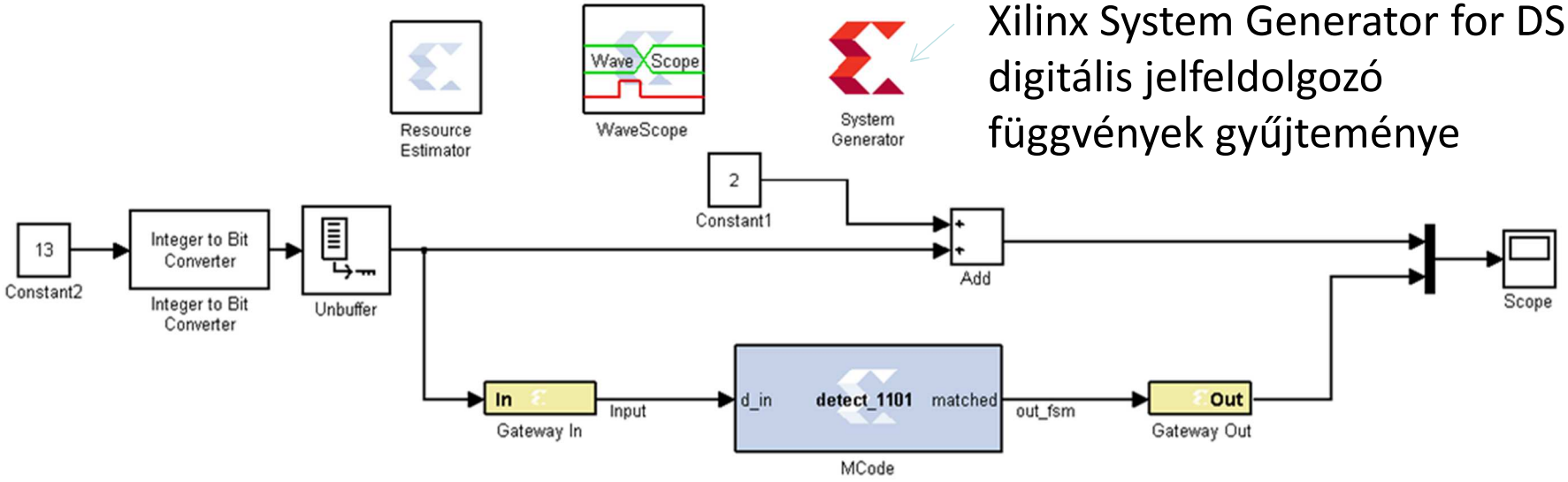
- **Intel FPGA (korábban Altera)**
  - Quartus II: integrált fejlesztő környezet (HW/FW fejlesztés hagyományos *HDL*-alapon)
    - Qsys: modularizált, hierarchikus beágyazott rendszer fejlesztés
    - EDS: beágyazott SW fejlesztés (Eclipse)
    - NiosII EDS: soft-core, illetve EDS-SOC DS5: ARM hard-core proc. támogatás
  - ModelSim: professzionális HDL szimulációs környezet
  - Altera SDK OpenCL (2013-): C/C++ alapú
  - Altera HLS: (2017-)
- **Xilinx**
  - **Vivado Design Suite: hagyományos HDL alapú környezet (2012-)**
  - *ISE Design Suite: integrált fejlesztő környezet (HW/FW)*
    - EDK: beágyazott rendszer fejlesztő környezet
    - SDK: beágyazott SW fejlesztő körny. (Eclipse)
  - *ModelSim, vagy ISim: szimulációs környezetek*
  - Vivado HLS/HLx: C/C++ alapú rendszerfejlesztés
  - **VITIS**: egységes szoftverfejlesztő környezet (2019-)

I. a. Xilinx Vivado / HLS fejlesztő környezet

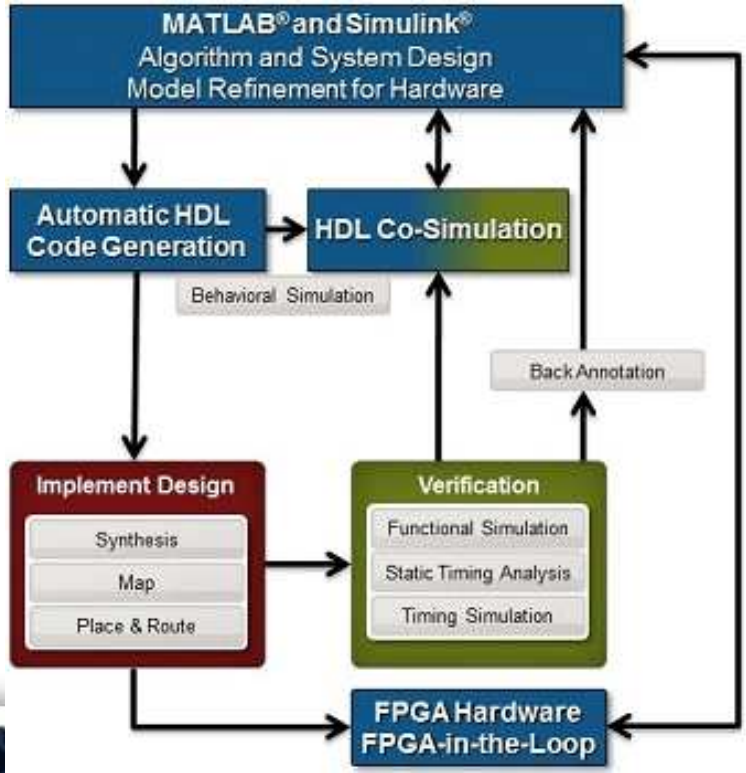




III. a.) Matlab - Simulink



Xilinx System Generator for DSP: digitális jelfeldolgozó függvények gyűjteménye



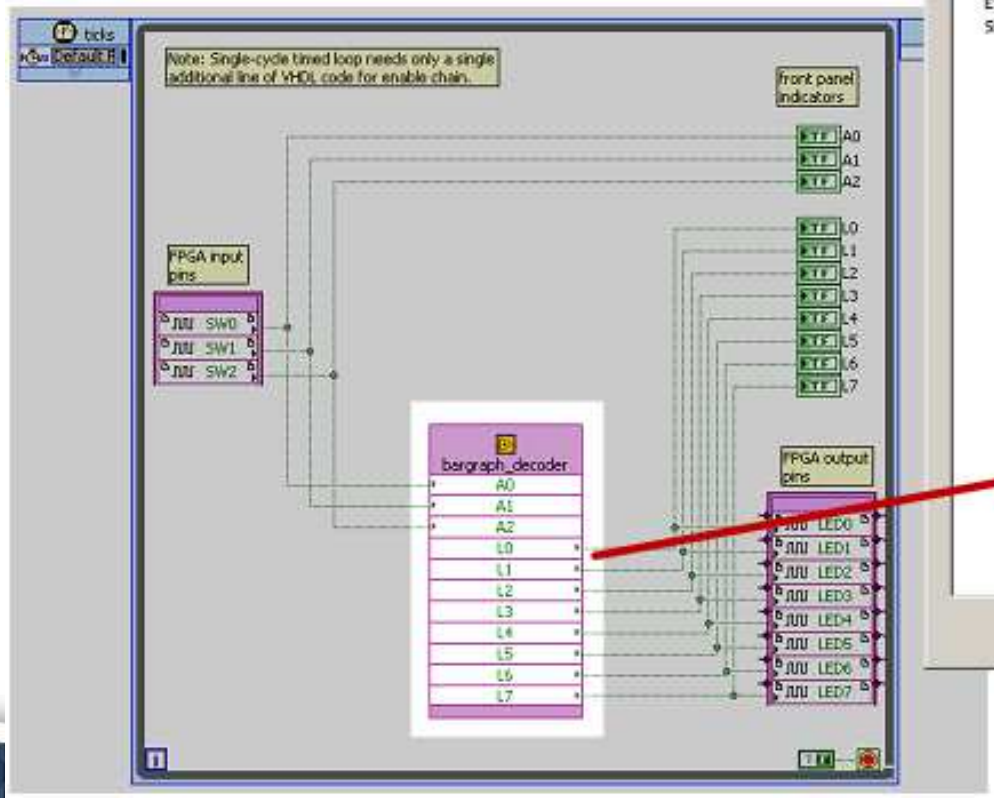
Mcode: Matlab-Code (.m nyelven írható egyedi modul, amely beépíthető akár a Simulink blokkok közé

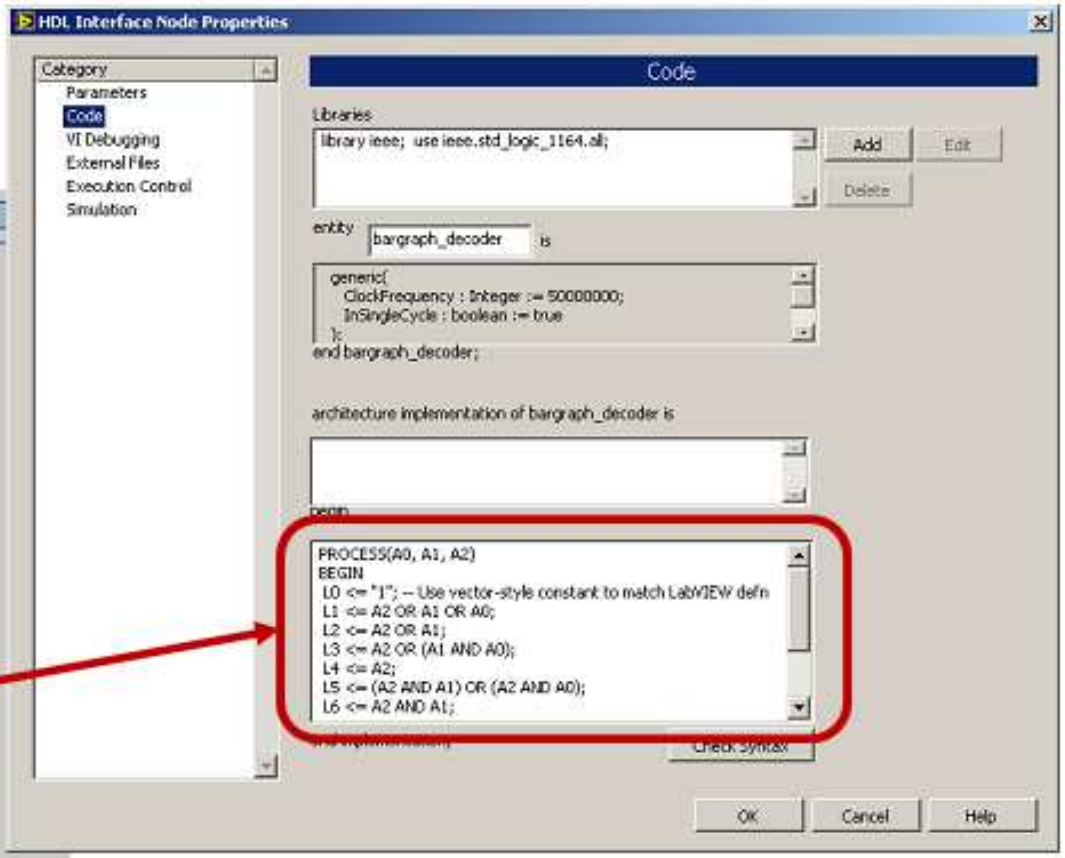
III. b.) LabView – FPGA modul



VHDL node

3->8 dekóder







Rövid általános ismertetés

V. XILINX VIVADO DESIGN SUITE (2020.1)

Xilinx Vivado Design Suite 2020.1

Laborokon a Xilinx Vivado Design Suite 2020.1 integrált keretrendszert használjuk!

- **Vivado*: integrált fejlesztő környezet, amelyből más program modulok indíthatók, mint pl:**
  - **IP Integrator – IP magok paraméterezése**
  - **Source Editor: forráskód szerkesztő**
  - **…**
- **Vivado Simulator*: integrált szimulátor**
- Vivado keretrendszer (~EDK) = FPGA-alapú sbeágyazott rendszer / firmware fejlesztő környezete
- Timing Analyzer – Időzítési analizátor
- FPGA Editor (chip layout, floorplan)
- **Hardware manager* – bitstream (FPGA konfiguráció) letöltő program**
- ChipScope: logikai analizátor

* Következő előadásokon részletesen tárgyaljuk és labor gyakorlatokon használjuk.

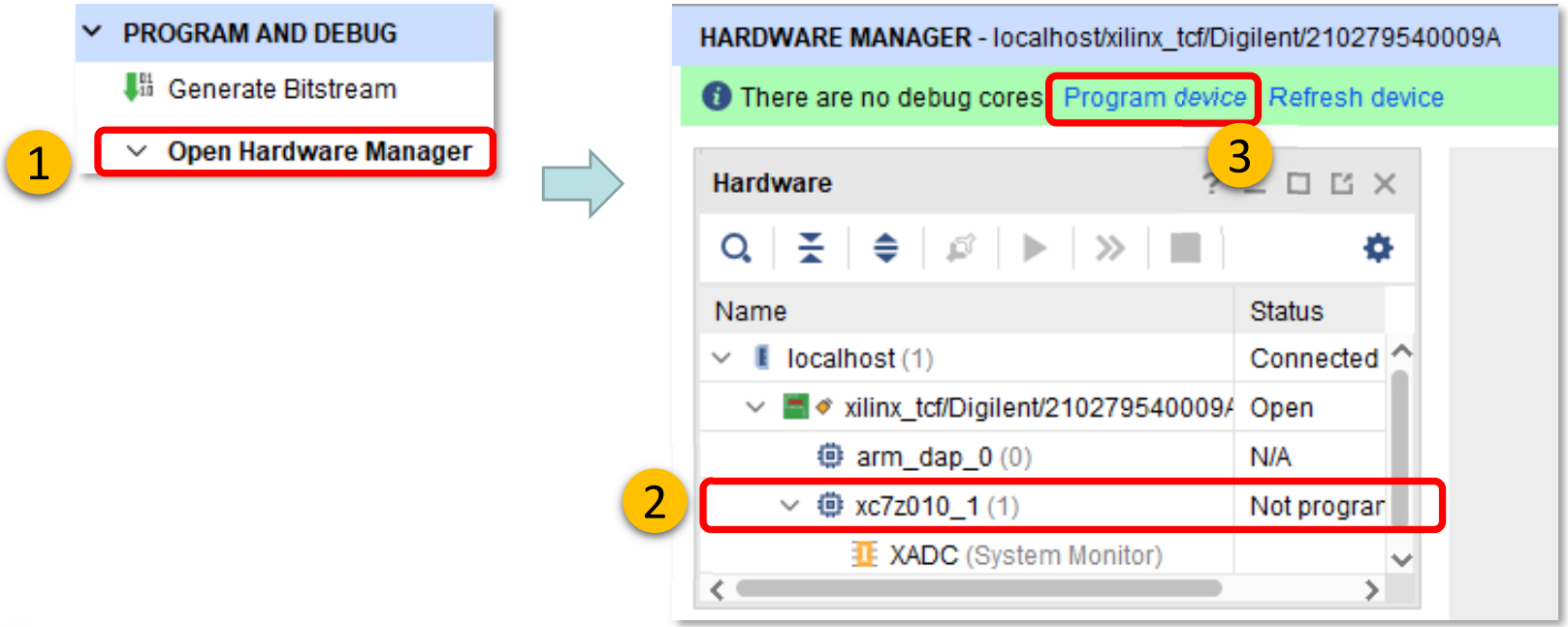
FPGA konfigurálhatóságának lehetőségei

- Az FPGA konfigurálásra három lehetőség is adódik:
  - **1.) Xilinx Vivado 2020.1 környezetbe beépülő Hardware Manager használata.**
    - **Mi ezt használjuk!**
  - 2.) Digilent Adept Suite használata,

1.) Xilinx VIVADO – Hardware Manager

Programok → Xilinx Design Tools → Vivado 2020.1

- Flow Navigator → Hardware Manager (Auto Connect)
  a JTAG lánc két elemből áll: (gyártónként változhat!)
  - ARM_DAP: Debug Access Port
  - FPGA (pontosabban APSoC): **xc7z010** = Xilinx Zynq 7010

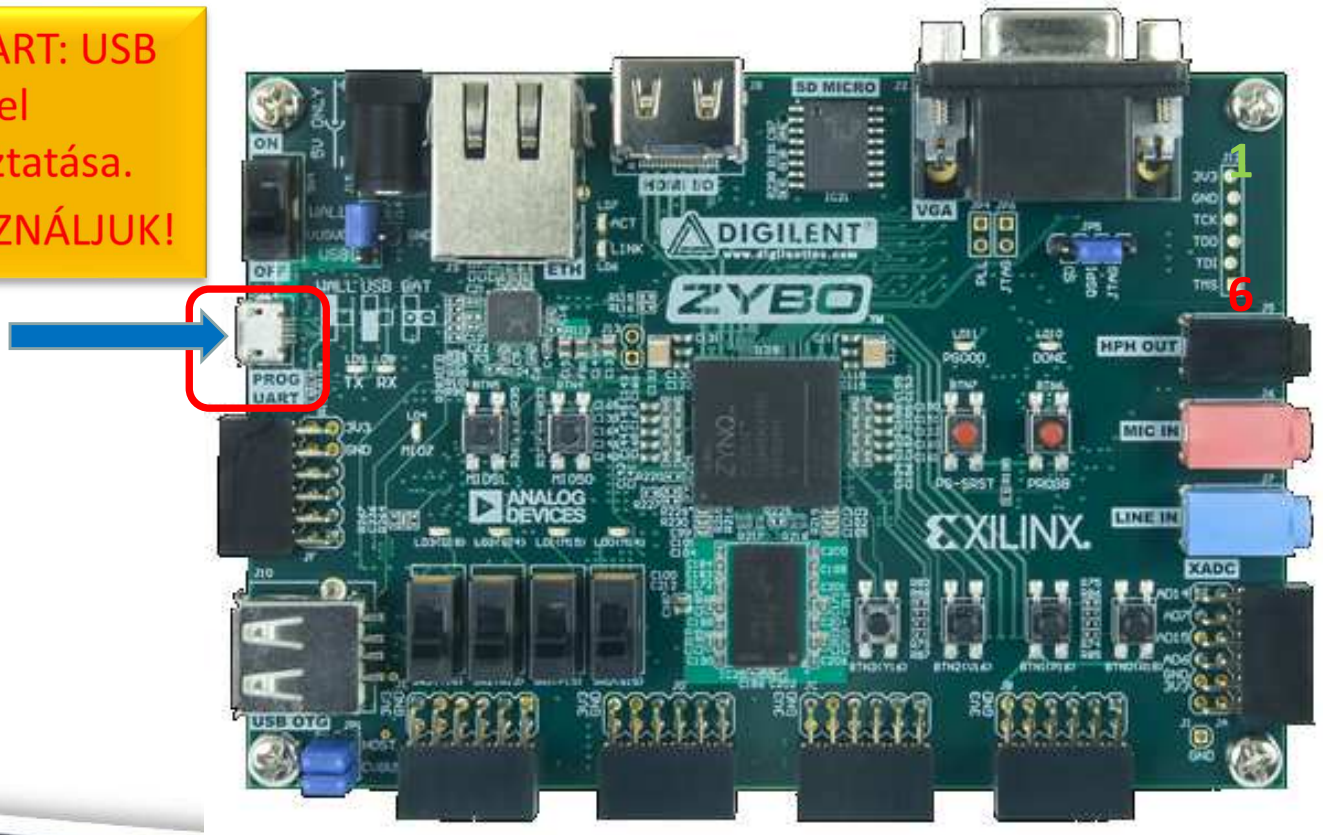


JTAG kapcsolat

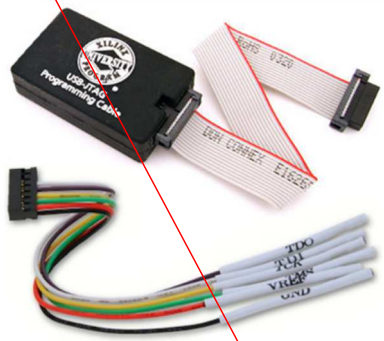
- FTDI USB-UART chip driverének telepítése (ha szükséges)
- USB mini kábel csatlakoztatása a PROG-UART jelű JTAG USB portra (JP5 jumper „JTAG” állásban van)
- a Digilent ZyBo kártya bekapcsolása
  - (Megj: a Xilinx Platform-USB-programozó is használható lenne)

PROG UART: USB mini kábel csatlakoztatása.

EZT HASZNÁLJUK!



Xilinx Platform USB programozó.
NEM HASZNÁLJUK!



ZyBo – Xilinx USB programming cable

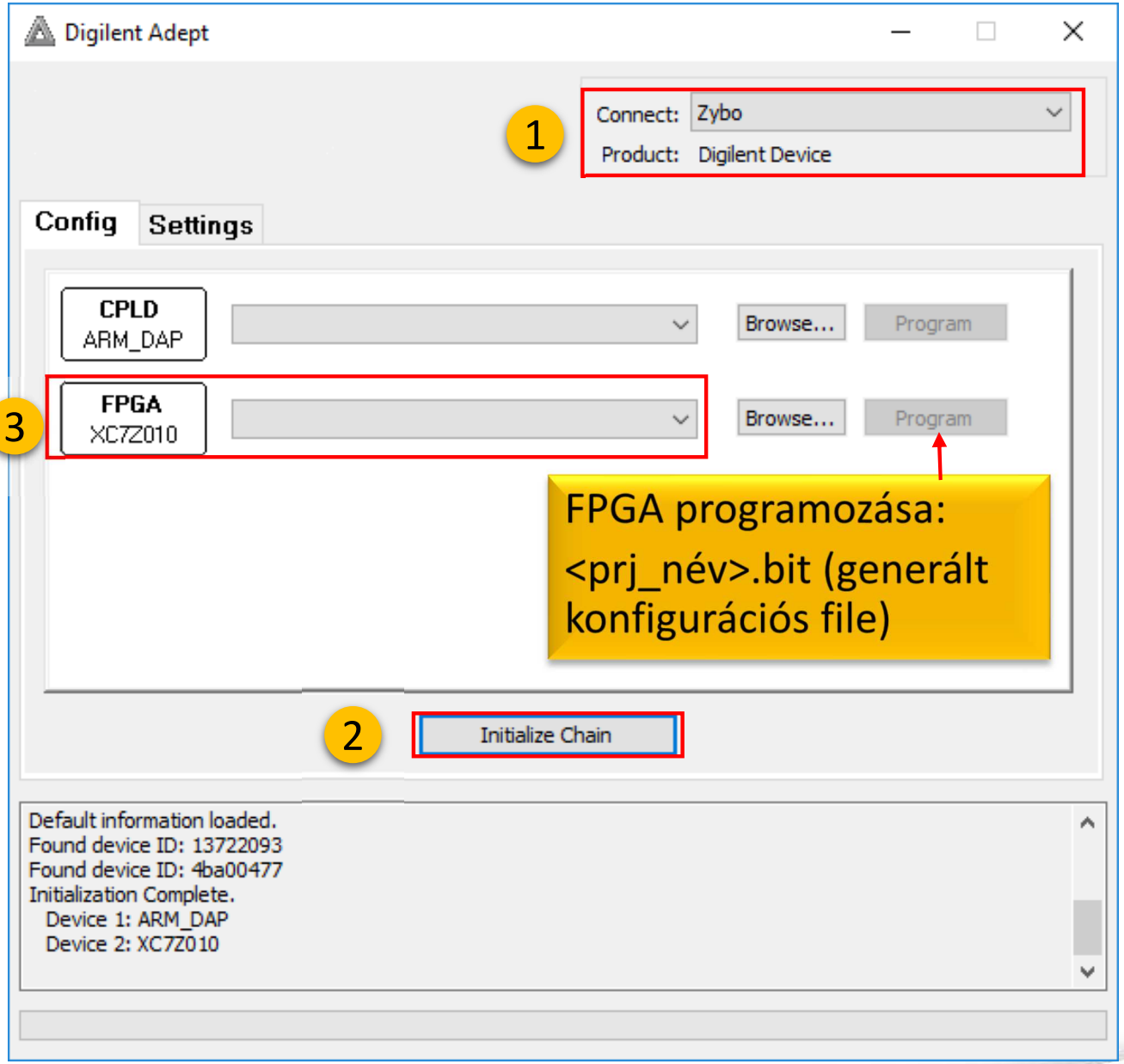
VCC3V3 – piros VREF (6)
GND – fekete GND (5)
TCK-JTAG – sárga TCK (4)
TDO-FX2 – lila – TDO (3)
TDI-JTAG – fehér TDI (2)
TMS-JTAG – zöld TMS (1)

2.) Digilent: Adept Suite

- Adept Suite alkalmazás egy PC-n futó interfész a programozható Digilent FPGA kártyákhoz (pl. ZYBO, stb.)
- Biztosítja az „on-board FPGA eszközök” konfigurálhatóságát, adatok fel-/letöltését (USB-n keresztül), I/O perifériák funkcióinak bővítését, valamint bizonyos kártyák esetén a beépített öntesztek futtatását (kártya verifikáció) is.
- **További funkciók:**
  - JTAG lánc, v. BSCAN (Boundary Scan) lánc keresése, inicializálása (konfiguráció letöltése előtt szükséges lépés),
  - Xilinx CPLD-k, FPGA-k, és PROM-ok programozása a megfelelő konfigurációs formátumban,
    - Xilinx FPGA (Virtex, Spartan családok) és CPLD: *.bit*
    - XCFS (Xilinx Platform Flash) PROM: *.bit* vagy *.mcs*
  - Adatátvitel az FPGA és PC között (specifikus regiszterek írása, olvasása), adatfolyam (stream) írása-olvasása (nem használjuk).

Digilent Adept szoftver használata

- Programok → Digilent → Adept Suite
- ①FPGA kártya bekapcsolása után megbizonyosodni, hogy az Adept Suite felismerte
- ②JTAG (IEEE 1149.1) lánc inicializálása (ARM DAP + FPGA)
- ③FPGA konfigurációs bitfile (projekt_neve.bit) megkeresése és felprogramozása



Felhasznált irodalom

- Digilent - Adept Suite (2.20.2):
  - https://reference.digilentinc.com/reference/software/adept/start?redirect=1id=digilent_adept_2
- Programok → Digilent → Adept → Adept Application User's Manual.pdf
- Digilent Plug-in:
  - http://store.digilentinc.com/digilent-plugin-for-xilinx-tools-download-only/

