



Digitális Rendszerek és Számítógép Architektúrák

6. előadás: Programozható logikai eszközök:
CPLD, FPGA. HLS: magas szintű szintézis

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

Jegyzetek, segédanyagok:

- Könyvfejezetek:

- <http://www.virt.uni-pannon.hu>

- Oktatás → Tantárgyak → Digitális Rendszerek és Számítógép Architektúrák (Nappali)

- FPGA-s rész fóliákon

- Fóliák, óravázlatok .ppt (.pdf)

- Feltöltésük folyamatosan

Eml: Vezérlő egységek fajtái:

- I. Huzalozott (klasszikus) módszerek (pl. korábban RISC architektúrák):
 - ☐ Mealy-modell,
 - ☐ Moore-modell.
- II. Mikro-programozott módszerek (reguláris vezérlési szerkezettel – pl. CISC architektúrák):
 - ☐ Horizontális mikrokódos vezérlő,
 - ☐ Vertikális mikrokódos vezérlő.
- **III. Programozható logikai eszközök (PLD):**
 - ☐ 1.) Maszk-programozható/makrocellás típusok: PLA, PAL, PROM, GAL, CPLD,
 - ☐ 2.) Tetszőlegesen újra-konfigurálható (=szoftveresen) típus: FPGA **(csak fóliákon!)**

PLD/ FPGA: tárgyalt ismeretekörök

1. Mik azok a.) Programozható Logikai Eszközök és az b.) FPGA-k?
 - Összeköttetések programozhatósága
2. Tervezési módszerek (Design methods)
3. Tervezés folyamata (Design-flow)
4. Magas-szintű szintézis (HLS – High-Level Synthesis)
5. Fejlesztő környezetek, hardver leíró nyelvek (HDL - Hardware Description Languages)

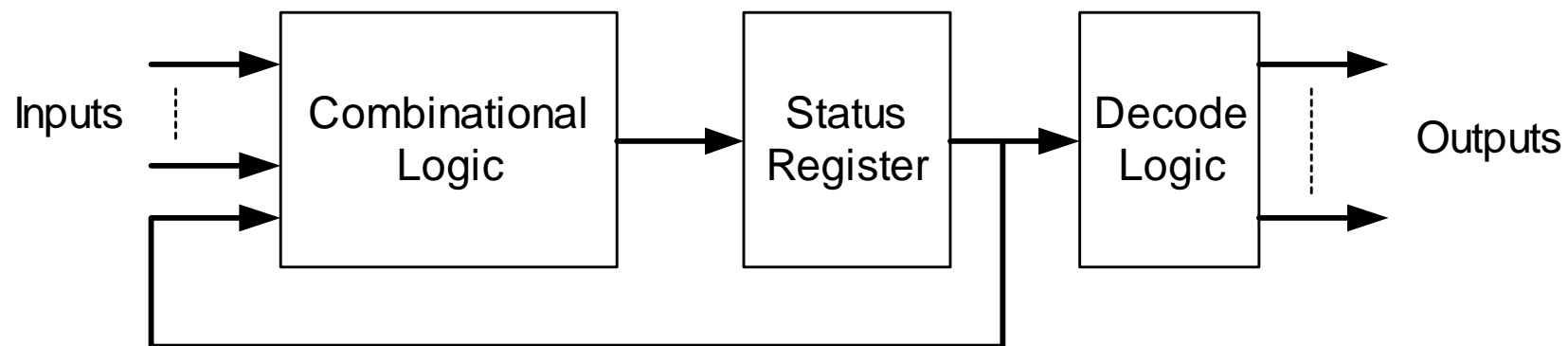


Ismertetés

1.a) Programozható logikai eszközök (PLD)

Állapotgép FSM tervezés tulajdonságai

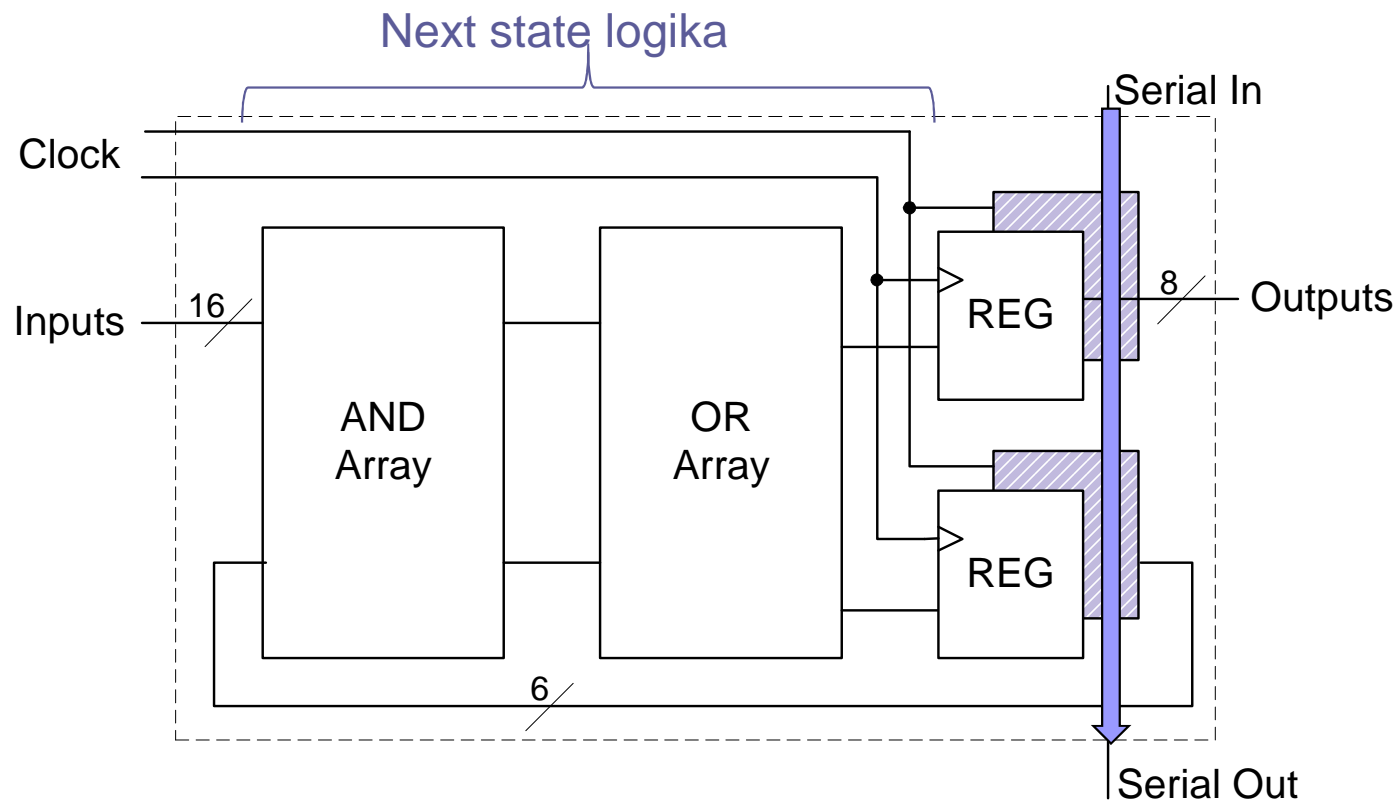
- Két kombinációs logikai hálózatról és egy regiszterből áll
- Tervezés során az állapot-átmeneteket vesszük figyelembe, **DE**
- Hibavalószínűség nagy,
- Szimulációs eszközök (CAD Tools) hiánya,
- Hibák lehetségesek a prototípus fejlesztése során is,
- Könnyen konfigurálható / flexibilis eszközök szükségesek
→ mindezek használunk programozható alkatrészeket



Field Programmable Logic Sequencer (FPLS) – Logikai sorrendvezérlő

- Egy vezérlő egység Next-State logikai blokkjának megvalósítása a tervező, vagy gyártó feladata. Általában változó logikát használnak a függvények megvalósításánál.
- A felhasználó által programozható logikai sorrendvezérlő (**Field Programmable Logic Sequencer**) programozható alkatrészekből építhető fel, amelyek a következőkben részletesen ismertetésre kerülnek.

Field Programmable Logic Sequencer (FPLS) – Logikai sorrendvezérlő

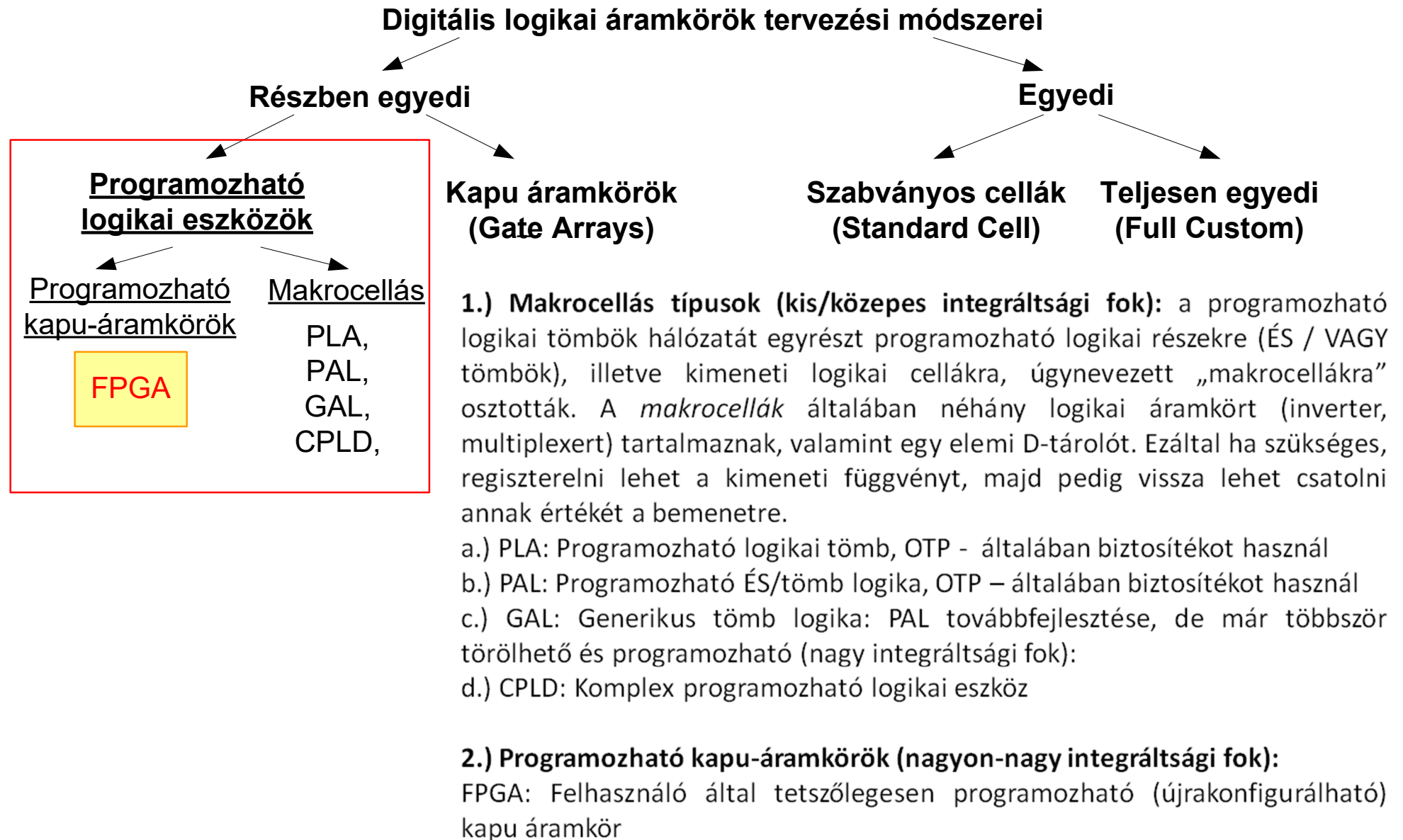


Működése:

- Normál (D-FF),
- Debug (Shift Reg.)

A logikai sorrendvezérlőnek 16 külső bemeneti vonala van, 1 RESET és 1 kimenet-engedélyező vonala, ill. 8 kimeneti vonala. A regiszterek egy-egy állapotot tárolnak, amelyek az órajel hatására a kimenetre íródnak, vagy a 6 belső, *visszacsatoló* vonalon keresztül visszacsatolódnak. A Next-State ill. a kimeneti szintek meghatározásánál programozható AND/OR tömböket használnak.

Tervezési módszerek



PLD – Programozható logikai eszközök

- A Programozható logikai áramköröket (PLD: Programmable Logic Devices) általánosan a kombinációs logikai hálózatok és sorrendi hálózatok tervezésére használhatjuk. Napjainkban ezek *VLSI* (Very-Large Scaling Integrated IC) típusú alkatrészek:
 - Több millió logikai kaput, vagy tranzisztort jelenthet
- Azonban míg a hagyományos kombinációs logikai hálózatok dedikált összeköttetésekkel, illetve kötött funkcióval (kimeneti függvény) rendelkeznek, addig a programozható logikai eszközökben pontosan ezek változtathatók, az alábbi lehetséges módokon:
 - A felhasználó által *egyszer programozható* / konfigurálható logikai eszközök (**OTP**: One Time Programmable), amelynél a gyártás során nem definiált funkció egyszer még megváltoztatható (ilyenek pl. a korai PAL, PLA eszközök)
 - *Többször, akár tetszőleges módon* programozható logikai eszközök = *újrakonfigurálható* (ilyenek pl. a korábbi GAL, vagy a mai modern CPLD, **FPGA** eszközök)

PLD-k két fő típusa:

■ 1.) **Makrocellás** PLD-k: (Programmable Logic Devices):

- ☐ PLA
- ☐ PAL
- ☐ GAL
- ☐ **CPLD**

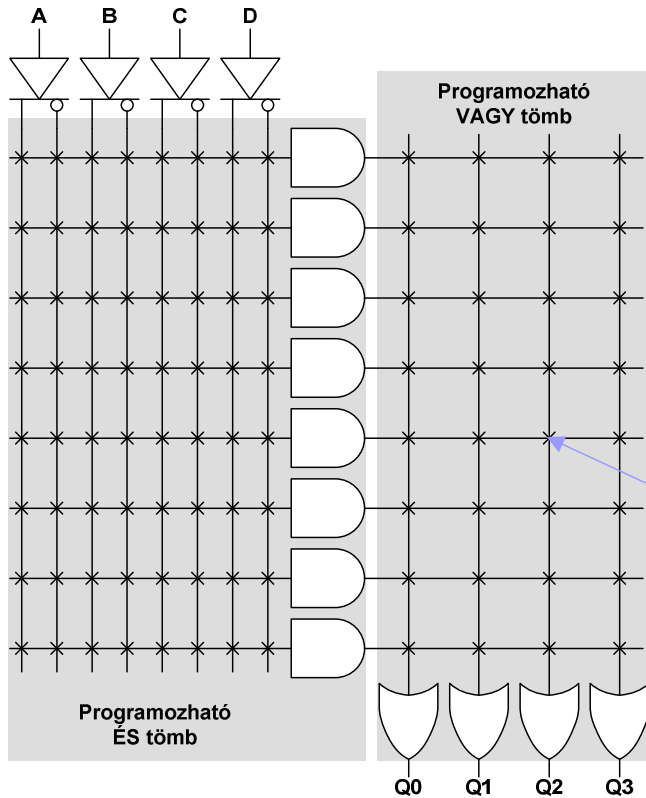
■ 2.) **FPGA** (Field Programmable Gate Array): Programozható Gate Array áramkörök

- ☐ **XILINX** (Spartan, Virtex, Kintex, Artix) ~ 53 % !
- ☐ **Altera/Intel-FPGA** (Stratix, Arria, Cyclone), ~ 36 %
- ☐ MicroSemi-Actel (pl. úrkutatásban),
- ☐ Lattice sorozatok,
- ☐ QuickLogic,
- ☐ További kisebb gyártók termékei.

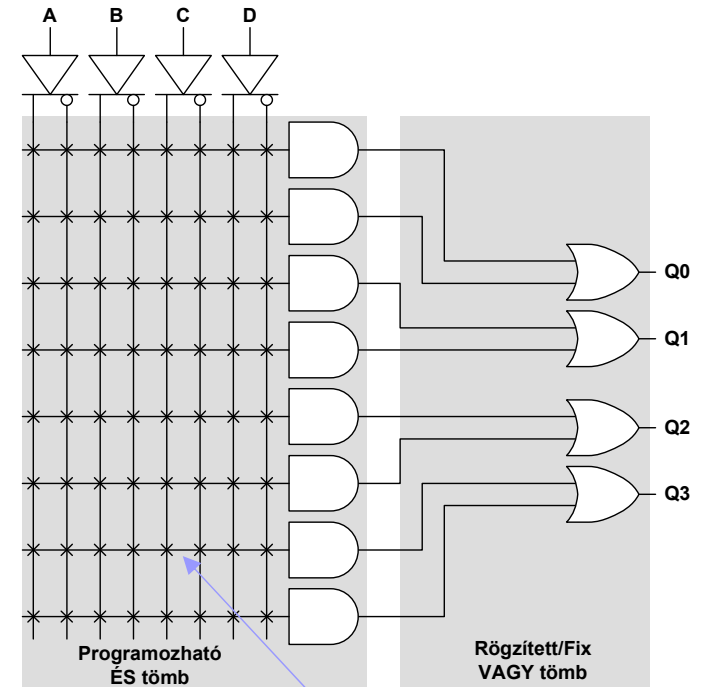


Makrocellás PLD-k

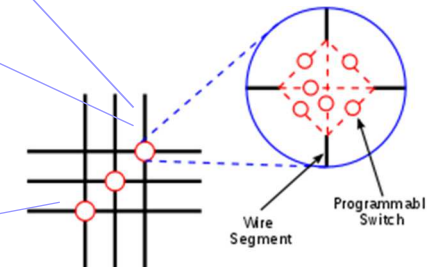
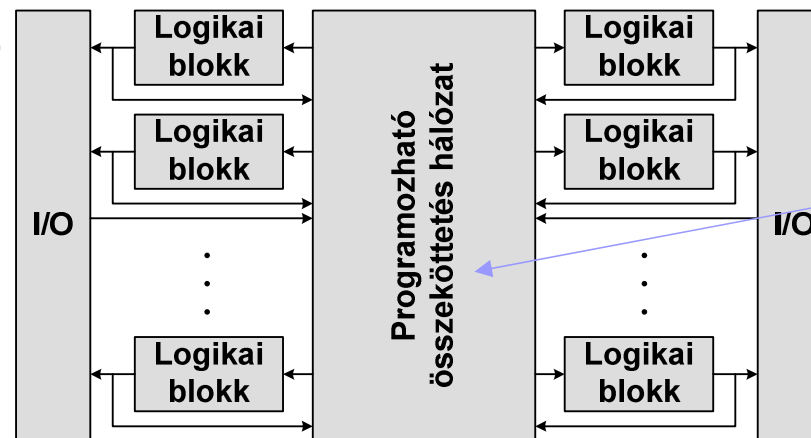
PLA



PAL

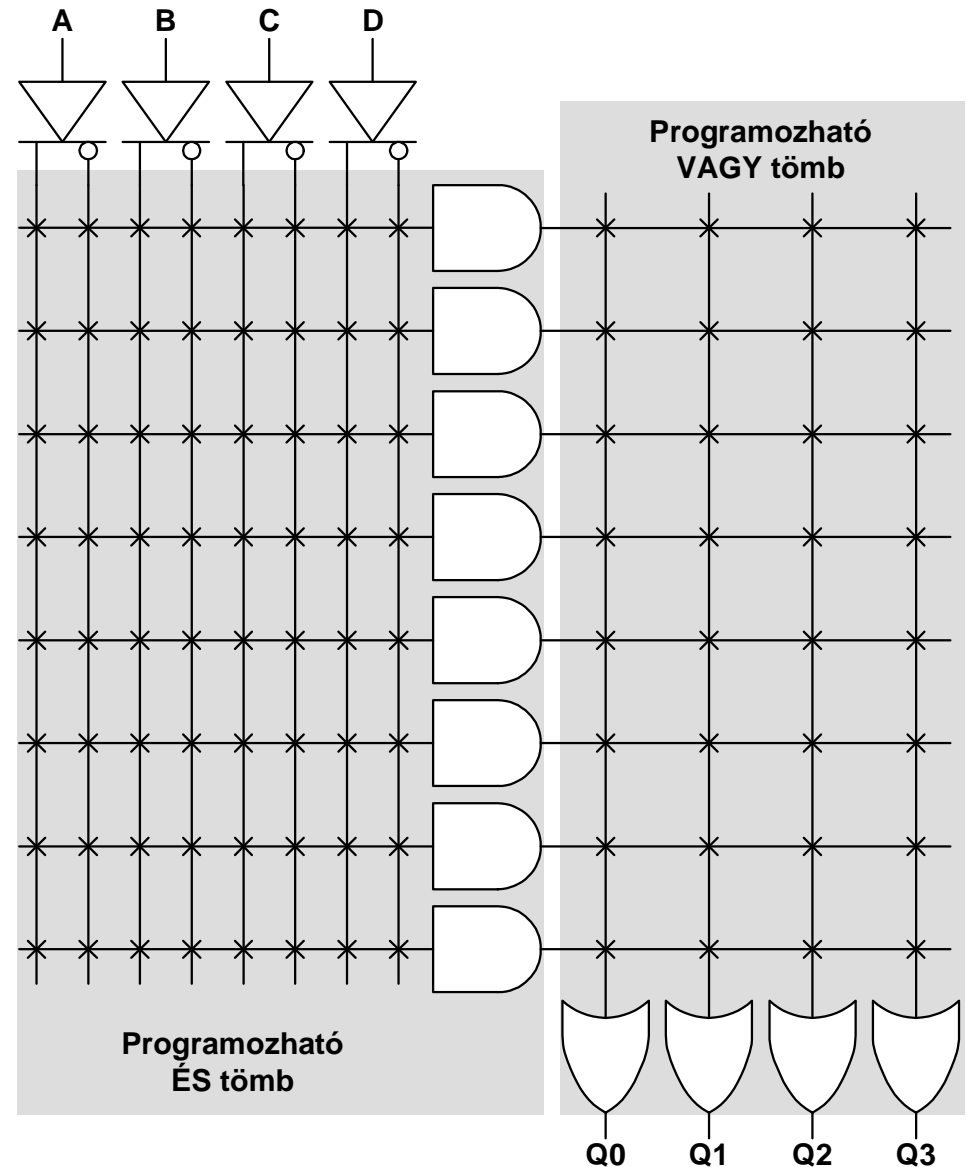


CPLD



Programmable Logic Array (PLA)

- Mindkét része (AND, OR) programozható
- Bármely kombinációja az AND / OR-nak előállítható
- Mintermek OR kapcsolata (DNF)
- Programozható kapcsolók a horizontális/ vertikális vonalak metszésében
- Q_n Kimeneteken D tárolók! (visszacsat. a bemenetekre)



PLA

- 1970-ben, a TI (Texas Instruments) által kifejlesztett eszköz *mindkét részhálózata (ÉS, illetve VAGY tömb) programozható* összeköttetéseket tartalmazott, amelyek segítségével tetszőleges mintermek tetszőleges VAGY kapcsolata előállítható (DNF alakot), ezáltal bármilyen kombinációs logikai hálózat realizálható volt (természetesen adott bemenet, ill. kimenet szám mellett).
- A programozható ÉS / VAGY tömbökben úgynevezett „*programozható kapcsolók*” vannak elhelyezve a horizontális/ vertikális vonalak metszéspontjában.
- Amennyiben a Q_n kimenet(ek)re tárolókat kötünk (pl. egyszerű D tárolót), majd pedig visszacsatoljuk a programozható logikai hálózat bemenete(i)re akár egy *sorrendi hálózati viselkedést* is meghatározhatunk.

Programozásuk (Fuse) biztosítékok segítségével

- Az összeköttetés mátrix metszéspontjaiban akár kis biztosítékok (fuse) helyezkednek el. Gyárilag logikai '1'-est definiál, tehát vezetőképes. Ha valamilyen spec. programozó eszközzel, a küszöbnél nagyobb feszültséget kapcsolunk rá, átégethető, tehát szigetelővé (nem-vezető) válik, és logikai '0'-át fog reprezentálni.
- A biztosíték átégetése, csak egyszer lehetséges, utána már csak a programozott állapotot fogja tárolni (**OTP** – One time programmable IC).

Példa: PLA tervezése

- Realizálja a következő függvényeket:

$$X = C + \bar{A} \cdot B + A \cdot \bar{B}$$

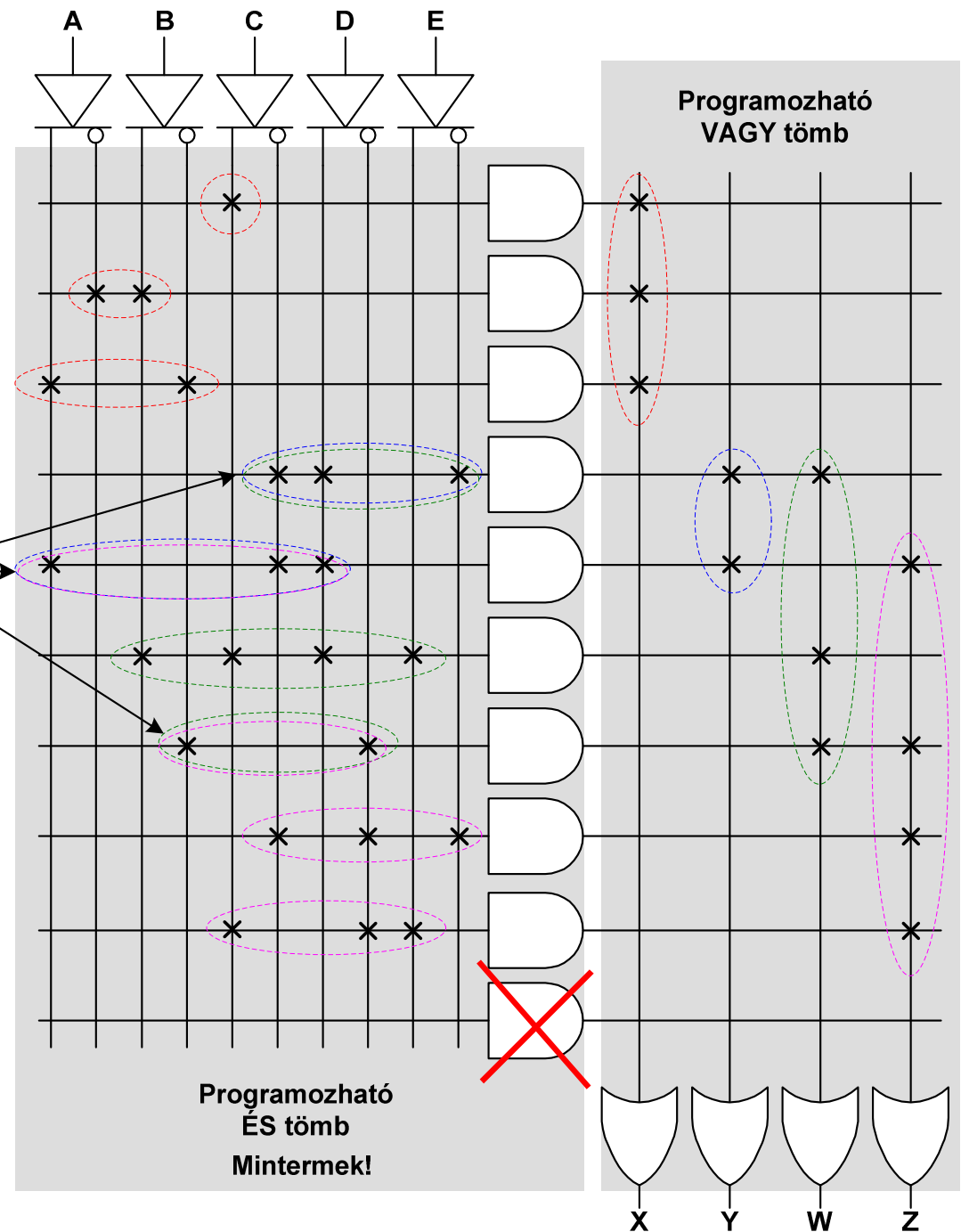
$$Y = \bar{C} \cdot D \cdot \bar{E} + A \cdot \bar{C} \cdot D$$

$$W = \bar{C} \cdot D \cdot \bar{E} + B \cdot C \cdot D \cdot E + \bar{B} \cdot \bar{D}$$

$$Z = A \cdot \bar{C} \cdot D + \bar{B} \cdot \bar{D} + \bar{C} \cdot \bar{D} \cdot \bar{E} + C \cdot \bar{D} \cdot E$$

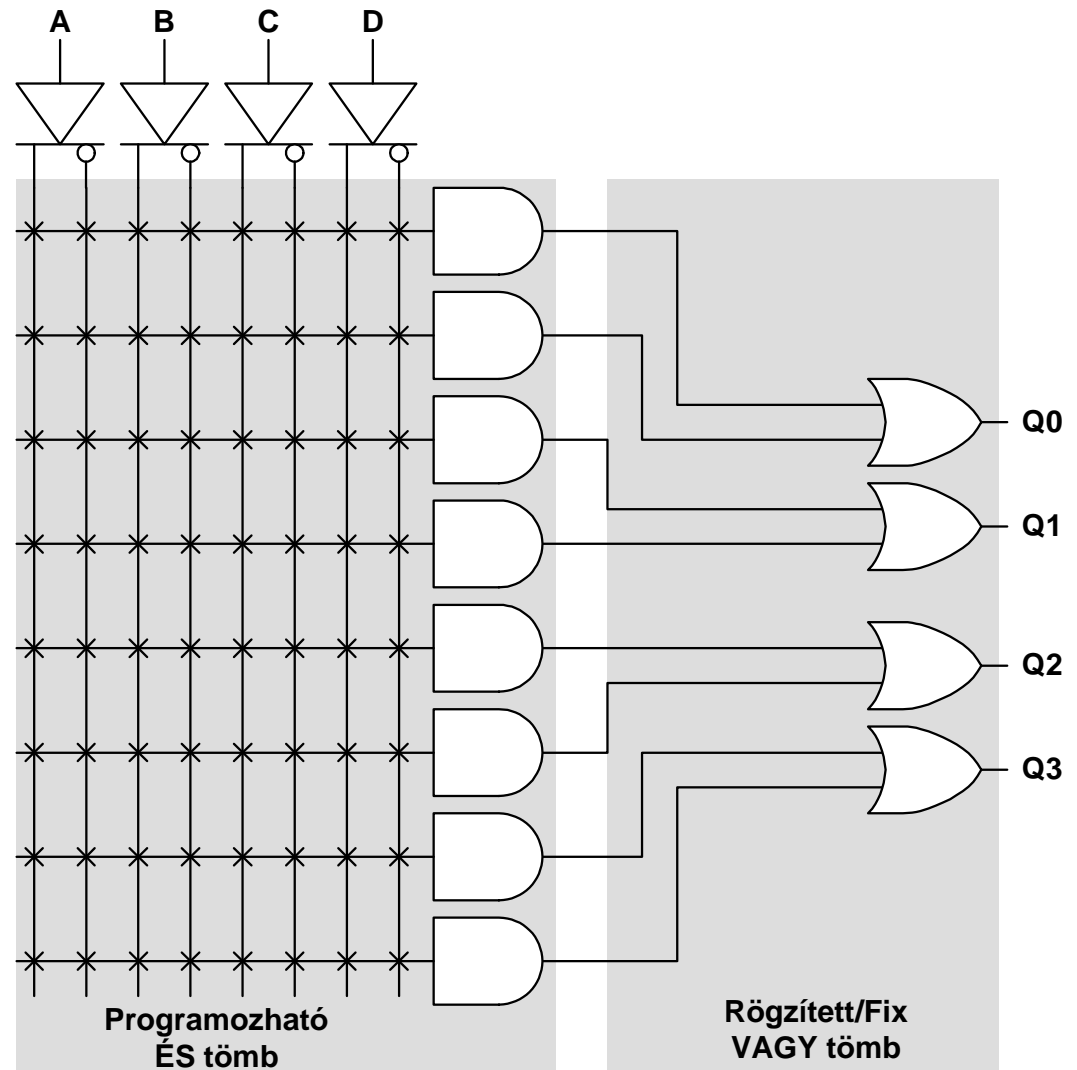
- Tehát 5 bemenete (A,B,C,D,E) és
- 4 kimenete (X,Y,W,Z) van.
- Rajzoljuk fel a kapcsolást is!

Közös mintermek



Programmable AND Logic (PAL)

- Egy programozható rész - AND / míg az OR fix
- Véges kombinációja áll elő az AND / OR kapcsolatoknak
- Metszéspontokban kevesebb kapcsoló szükséges
- Gyorsabb, mint a PLA
- Q_n kimeneteken D tárolók (visszacsatolódhatnak a bemenetekre)

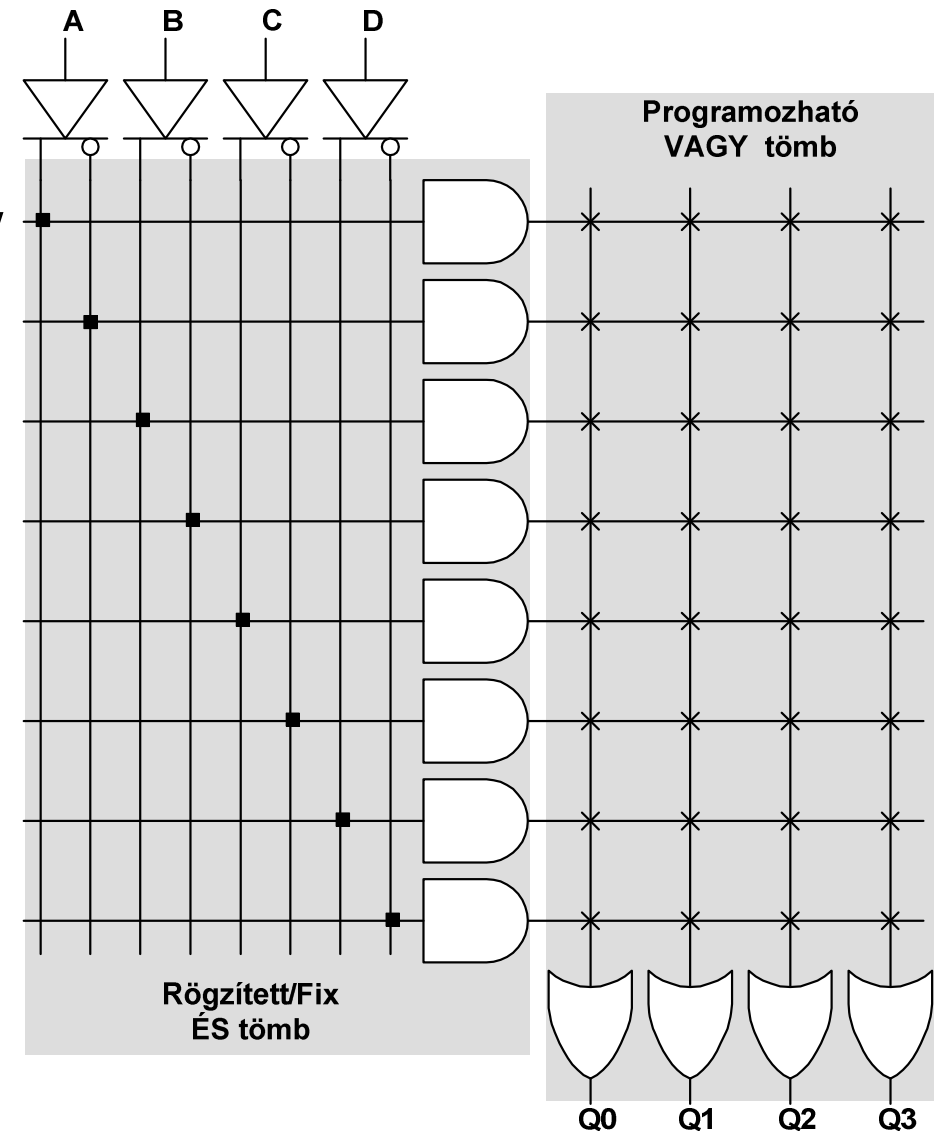


PAL

- Elsőként, 1978-ban az MMI (Monolithic Memories Inc.) jelent meg ilyen programozható eszközökkel, majd pedig későbbi jogutódja a Lattice Semiconductor, illetve az AMD a 80'-as évek végén.
- A PAL hálózatban *VAGY tömb fix/rögzített* a *programozható részt az ÉS tömb* jelenti, míg az. Így a tetszőleges mintermeknek csak egy véges kombinációja (VAGY) állítható elő: a lehetséges kimeneti függvények variálhatóságából veszítünk, cserébe viszont a VAGY részek dedikált útvonalainak jelterjedési sebessége nagyobb, míg az eszköz mérete kisebb és ezáltal olcsóbb is lesz.
- Ezáltal a metszéspontokban kevesebb kapcsoló szükséges („gyorsabb”, mint a PLA). Hasonlóan a PLA-khoz, amennyiben a Q_n kimenet(ek)re tárolókat kötünk (pl. egyszerű D tárolót), majd pedig visszacsatoljuk a programozható PAL logikai hálózat bemenete(i)re akár sorrendi hálózati viselkedést is könnyen valósíthatunk.

Programmable Read Only Memory (PROM)

- Egy programozható rész - OR / míg az AND fix (valamilyen elrendezés szerint)
- Véges kombinációja áll elő az AND / OR kapcsolatoknak
- Metszéspontokban kevesebb kapcsoló szükséges
- Gyorsabb, mint a PLA
- Q_n kimeneteken D tárolók! (visszacsatolódhatnak a bemenetekre)



x: programozható
■: fix/rögzített

GAL (Generic Array Logic): Általános tömb logika

- 1985-ben a Lattice Semiconductor fejlesztette ki elsőként,
 - amely a *PAL*-nak egy továbbfejlesztett változatát képviseli.
 - Ugyanolyan belső struktúrával rendelkezik, mint egy *PAL* áramkör,
 - azonban többször programozható: tehát törölhető és újraprogramozható eszköz.
 - EEPROM technológiát (lásd. lebegő-gate) alkalmaz. Később a National Semiconductor, és AMD is megjelent saját *GAL* sorozataival a piacon.

CPLD

CPLD (Complex Programmable Logic Devices): Komplex-programozható Logikai eszközök

- Valójában *átmenetet* képeznek a kis/közepes integráltsági fokú makrocellás PLD-k GAL/PAL áramkörei, illetve a nagy integráltsági fokú FPGA kapu-áramkörök között.
- A GAL/PAL áramköröktől architektúráisan annyiban különbözik, hogy ki lett bővítve: nem egy-, hanem több logikai cellamátrixot tartalmaz, amelyek konfigurálható blokkok reguláris struktúrájában vannak elrendezve. A mai modern FPGA áramköröktől viszont az különbözteti meg felépítésben, hogy *nem tartalmaz dedikált* erőforrásokat (pl. szorzók, memória blokkok).
- a legnagyobb gyártók, amelyek jelenleg is aktív szereplői a CPLD-k piacának a következők: Xilinx, Altera, Lattice Semiconductor, Atmel stb.

Complex Programmable Logic Device (CPLD)

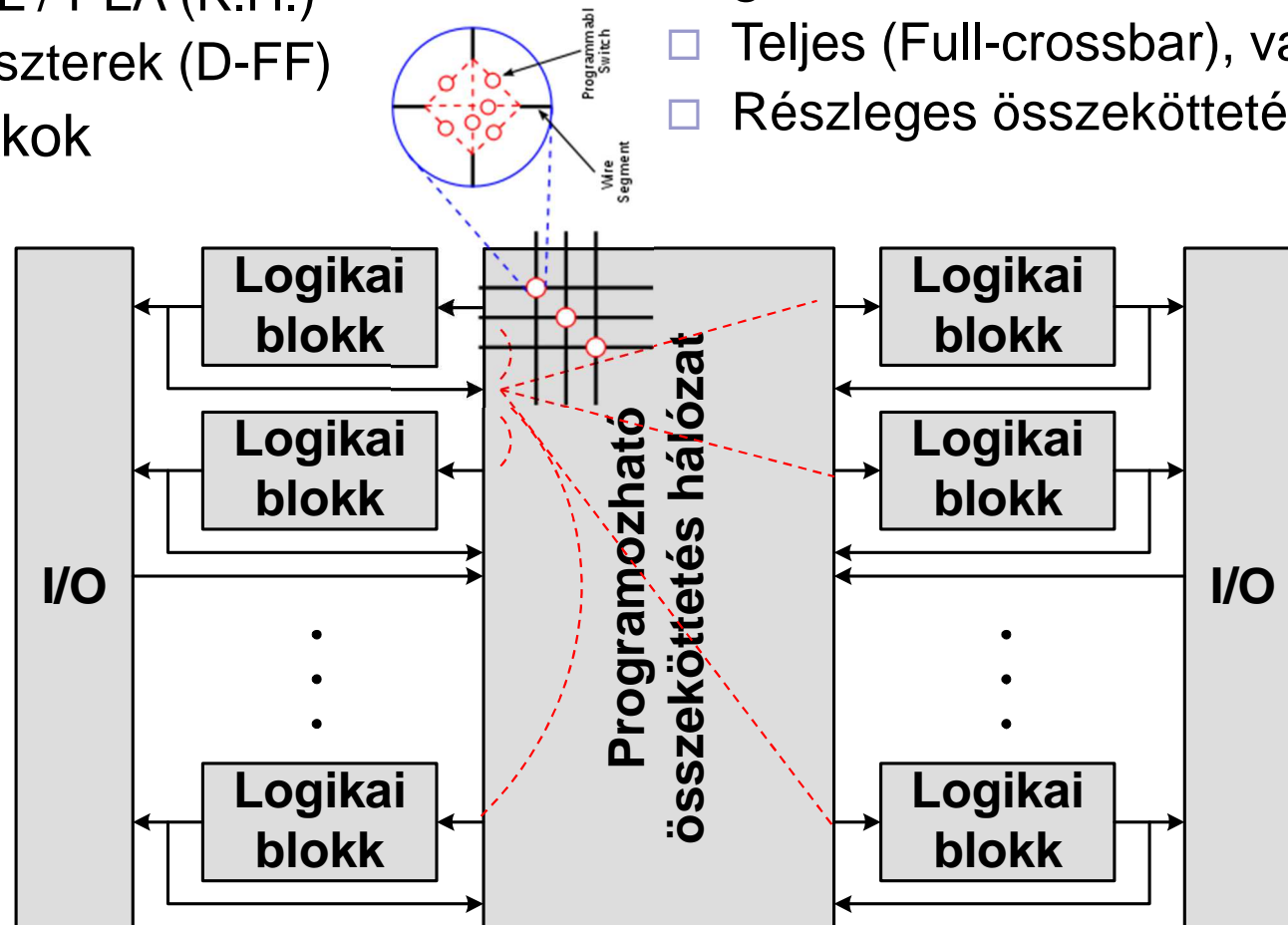
- 1 Logikai Blokkon belül:

- ~ PAL / PLA (K.H.)
- Regiszterek (D-FF)

- I/O Blokkok

- Programozható összeköttetések (PI: Programmable Interconnection)

- Teljes (Full-crossbar), vagy
- Részleges összeköttetés hálózat



CPLD (folyt)

A CPLD-kben található Logikai Blokk-ok (~**makrocellák**):

- egyrészt *logikai kapuk* tömbjeit tartalmazzák (hasonlóan a PAL/GAL áramkörök felépítéséhez – DNF alak),
- másrészt *regisztereket* (D-tárolókból) tartalmaznak a logikai tömbök által előállított kimenetek átmeneti tárolásához, valamint
- *multiplexereket*, mellyel a programozható összeköttetés hálózatra, vagy I/O blokkok celláihoz lehet továbbítani a belső Logikai Blokkok által előállított kimeneti értékeket. Ezáltal nemcsak logikai kombinációs hálózatokat, hanem akár sorrendi hálózatokat is egyszerűen megvalósíthatunk CPLD-k segítségével

- A CPLD-kben található *Programozható összeköttetés hálózat*
 - teljes összeköttetést (mindenki-mindekivel), vagy
 - részleges összeköttetést (valamilyen struktúra szerint, pl. bemenetet – kimenettel, főként régi CPLD típusok esetén) biztosít az egyes blokkok között.
- Kikapcsoláskor a CPLD konfigurációs memóriája megtartja értékét (non-volatile típus), ezért nem kell egy külső pl. ROM memóriát használni az inicializációs minták tárolásához, bekapcsoláskor ezek automatikusan betöltésre kerülnek. A CPLD-et közkedvelten alkalmazzák különböző interfészek jeleinek összekapcsolásához (*glue-logic*), amennyiben a jeleken átalakításra is szükség van, továbbá áraik az FPGA-k árainál jóval kedvezőbbek.



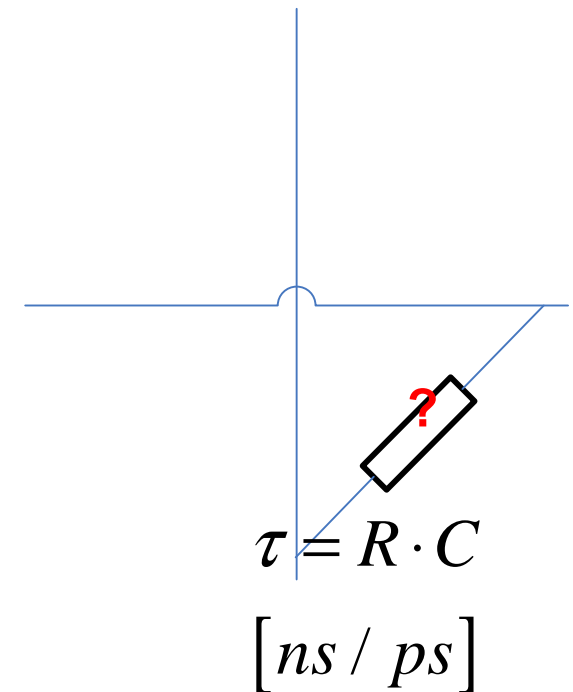
Hogyan programozhatók a
VLSI alkatrészek?

Programozási technikák
összeköttetésekre

Programozási technikák

Mi van a programozható összeköttetések csomópontjaiban, illetve milyen módszerrel programozhatóak?

- a.) SRAM
- b.) MUX
- c.) Antifuse
- d.) Floating Gate
 - e.) EPROM/EEPROM/Flash



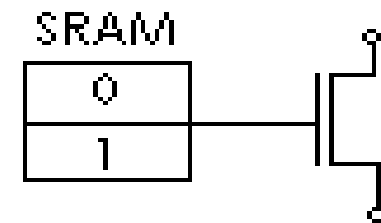
Programozás = konfigurálás

Konfigurálás (PLD/FPGA esetén) – mielőtt az eszközt használni szeretnénk egy speciális (manapság általában JTAG szabványú) programozó segítségével „fel kell programozni”: le kell tölteni a konfigurációs állományt (bitfájl, vagy object fájl). A programozás a legtöbb PLD esetében a belső programozható összeköttetésének fizikai típusától függően azok beállításával történik. A programozható összeköttetésekben a következő lehetséges alkatrészek találhatók:

- **Biztosíték (Fuse):** átégetésük után nem visszafordítható a programozási folyamat (OTP). Korábban a PAL eszközök népszerű kapcsoló elemeként használták.
- **Antifuse technológia:** (OTP), az antifuse-os kristályszerkezetű kapcsoló elem 'átolvasztása' után egy nagyon stabilan működő összeköttetést kapunk, amely sajnos szintén nem visszafordítható folyamatot jelent. A technológia drága, az előállításához szükséges maszk-rétegek nagy száma miatt, nagyon jó zavarvédetség elérése érdekében használják (pl. úrkutatás).
- **SRAM cella + tranzisztor:** tetszőlegesen programozható (FPGA-k esetén legelterjedtebb kapcsolás-technológia), az SRAM-ban tárolt inicializáló értéktől függően vezéri a tranzisztor gate-elektrodáját
- **SRAM cella + multiplexer:** tetszőlegesen programozható az SRAM cellában tárolt értéktől függően (kiválasztó jel) vezérelhető a multiplexer
- **Lebegő kapus tranzisztor (Floating Gate) technológia:** elektromosan tetszőlegesen programozható, a mai EEPROM/Flash technológia alapja. **26**

a.) SRAM cellás

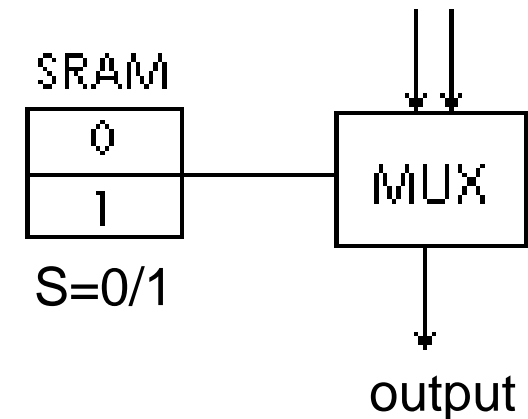
- Tulajdonságai: (pl Xilinx, Altera)
 - végtelen sokszor újraprogramozható (statikus RAM)
 - táp kikapcsolása után az SRAM elveszti tartalmát
 - bekapcsoláskor (inicializáláskor) a programot be kell tölteni, fel kell programozni
 - az SRAM cellára egy áteresztő tranzisztor van csatolva. A tranzisztor vagy kinyit (vezet), vagy lezár. Az SRAM értéke, ami egy bitet tárol ('0' vagy '1') letölthető. Összeköttetések, vagy MUX-ok állását is eltárolja.
 - 1 bit tárolása az SRAM-ban (min. 6 tranzisztorból áll)
 - sok tranzisztor (standard CMOS), nagy méret, nagy disszipáció
 - nem kell frissíteni az SRAM-ot
 - nagy 0.5-2 k Ω átmeneti ellenállás
 - nagy 10-20 femtoF parazita kapacitás



b.) MUX - multiplexeres

- Tulajdonságai:

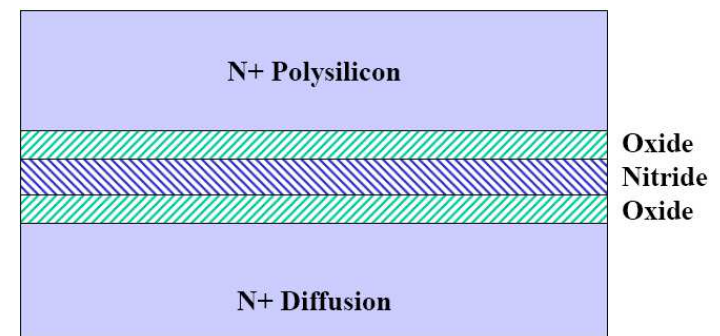
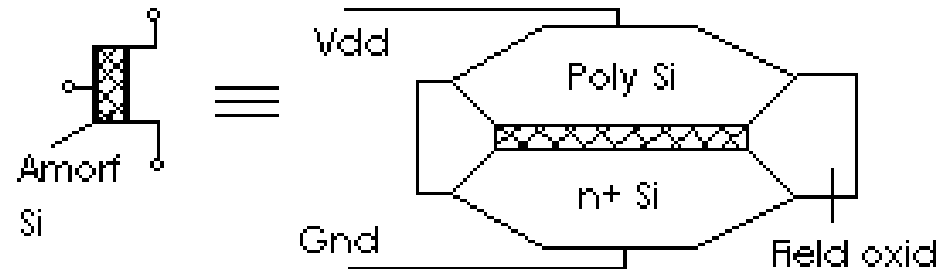
- az SRAM-ban tárolt '0' vagy '1' értéket használunk a Multiplexer bemeneti vonalának kiválasztásához. (Működése hasonló az SRAM celláéhoz.) /Bemenetek közül választ a szelektáló SRAM-beli érték segítségével és a kimenettel köti össze./



c.) Antifuse

A tranzisztor Gate-jét amorf kristályos Si alkotja, amelyet relatíve nagy feszültség (kb 20-30V) hatására átkristályosítunk (átolvasztás), így vezetővé válik véglegesen. Pl. Texas Instruments, Actel, QuickLogic alkalmazza ezt a technológiát. Tulajdonságai:

- A dielektrikum „átégetése” irreverzibilis folyamat, nem lehet újraprogramozni
- csak egyetlen egyszer programozható (OTP)
- kis méreten megvalósítható, kis disszipáció
- kis átmeneti ellenállás 300 Ω
- kis parazita kapacitás 1.1-1.3 femtoF
- előállításához sok maszkréteg szükséges, drága technológiát igényel
- Típusai
 - ☐ ONO (Oxid-Nitrid-Oxid)
 - ☐ Amorf Si

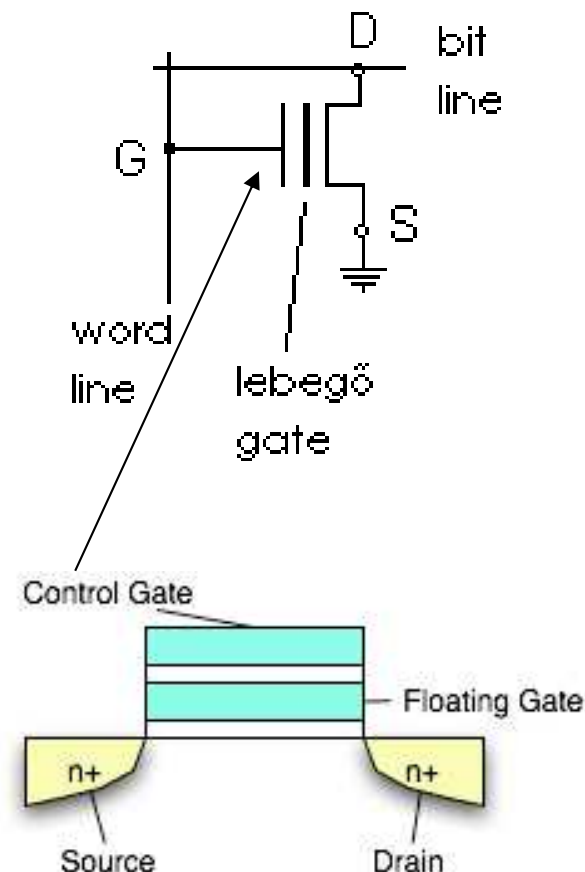


d.) Floating gate

Két-gates tranzisztor**, melynek középső gate-je a lebegő gate, mely tárolja az információt. A másik külső gate fix (control, vagy érzékelő gate), biztosítja az írást/olvasást. Programozható összeköttetéseknel, csomópontokban is használatos.

■ Tulajdonságai:

- Programozása/írás: külső control gate segítségével töltéseket viszünk fel a belső ún. lebegő gate-re, ~(ki)nyit a tranzisztor
- többször törölhető
- kikapcsoláskor is megőrzi tartalmát (non-volatile, akár 99 évig), töltések nem sülnek ki
- nagy 2-4 k Ω átmeneti ellenállás
- nagy 10-20 femtoF parazita kapacitás
- PL. **Intel**, Actel, Lattice



**Megj: 2012. Intel tri-gate 3D tranzisztora (22 nm)

e.) EPROM / EEPROM / Flash

■ Tulajdonságai:

- ☐ „**Floating-gate**” technológiát alkalmazza!
- ☐ 10000x szer programozható
- ☐ Megőrzi tartalmát (Non-volatile)
- ☐ UV-fénnyel törölhető (EPROM)
- ☐ Elektromosan törölhető (EEPROM / Flash)
- ☐ Nagy felület
- ☐ Nagy átmeneti ellenállás, nagy parazita kapacitás
- ☐ További CMOS gyártási lépések (sok maszk réteg) szükségesek – drága
- ☐ Pl: Altera (3000, 5000 - első sorozatai)



Ismertetés

1.b) FPGA (Field Programmable Gate Array) architektúrák

Irodalom:

- FPGA gyártók oldalai:

- <http://www.xilinx.com>

- <http://www.altera.com>

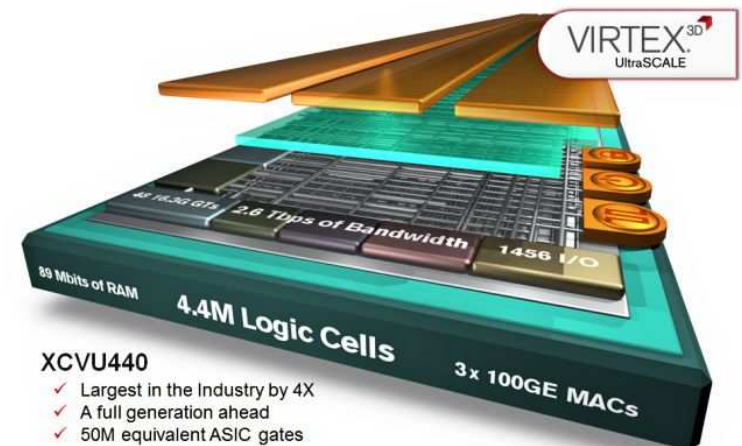
- <http://www.actel.com>

- FPGA és Programmable Logic Journal:

- <http://www.fpgajournal.com>

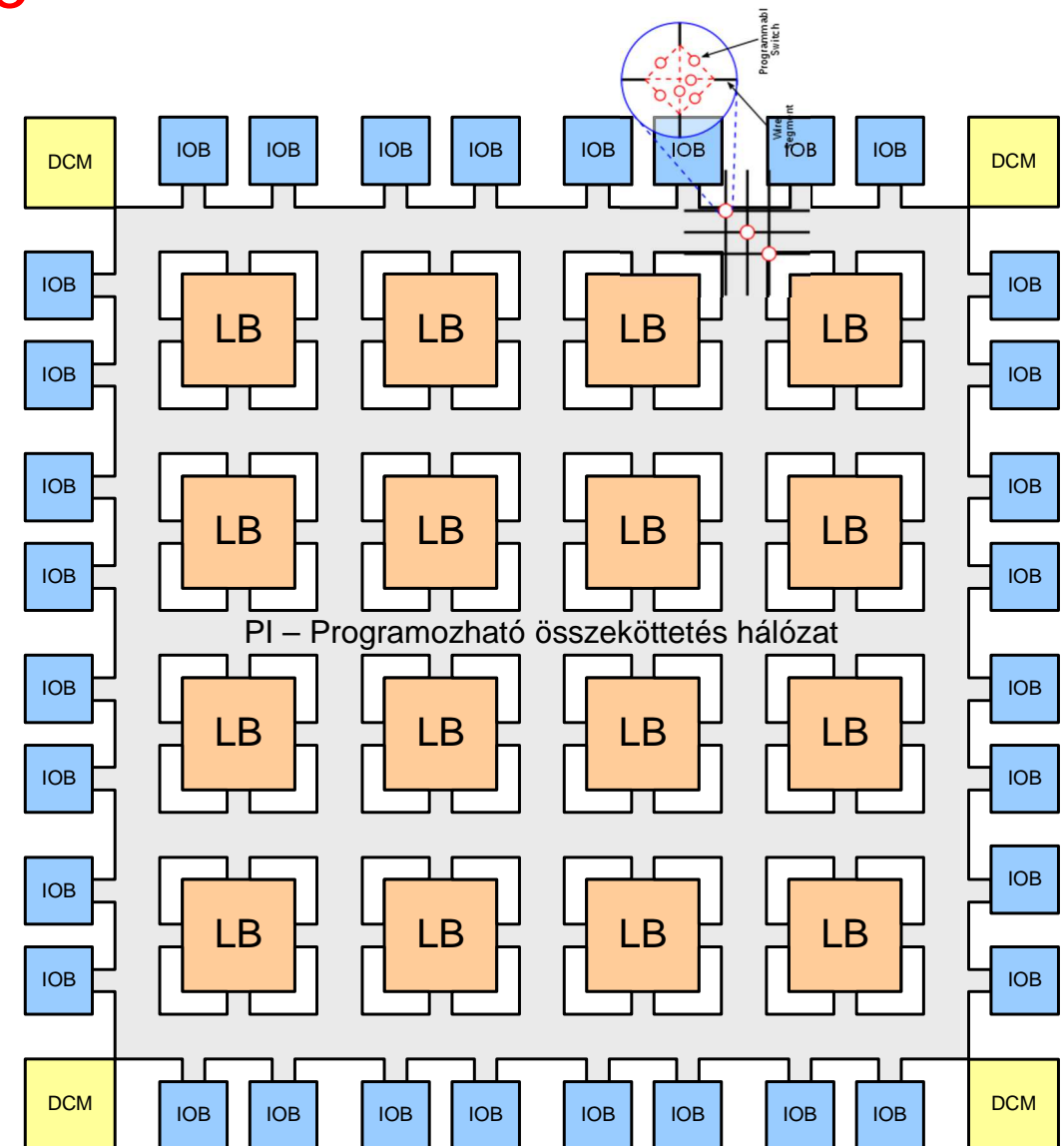
FPGA

- **Field Programmable Gate Array** = „Felhasználó által tetszőlegesen/többször” programozható kapuáramkörök
 - architektúráisan tükrözik mind a PAL, ill. CPLD felépítését, komplexitásban pedig a CPLD-ket is felülmúlják. Nagy, illetve nagyon-nagy integráltsági fokkal rendelkeznek: ~10.000 - ~10.000.000 !! *ekvivalens logikai kapu* is tartalmazhat gyártótól, és sorozattól függően.
 - *Ekvivalens tranzisztorszám*
 - Xilinx Virtex-7 2000T FPGA esetén már meghaladta a ~6.5 milliárdot (2012 – 28nm), amely ~2 millió logikai cellát jelentett.
 - a kapható legnagyobb Xilinx Virtex-Ultrascale+ XCVU440 (2015 – 20nm->16nm) FPGA:
~20 milliárd tr. - ~4.4 millió logikai cella!
(~ 50 millió logikai kapu ekv.)



Field Programmable Gate Array (FPGA) általános felépítése

- **LB/CLB:** Konfigurálható Logikai Blokkok, amelyekben LUT-ok (Look-up-table) segítségével realizálhatók például tetszőleges, több bemenetű (ált. 4 vagy 6), egy-kimenetű logikai függvények. Ezek a kimeneti értékek szükség esetén egy-egy D flip-flopban tárolhatók el; továbbá multiplexerek, egyszerű logikai kapukat, és összeköttetések is tartalmazznak.
- **IOB:** I/O Blokkok, amelyek a belső programozható logika és a külvilág között teremtenek kapcsolatot. Programozható I/O blokkok kb. 30 ipari szabványt támogatnak (pl. LVDS, LVCMOS, LVTTL, SSTL stb.).
- **PI:** az FPGA belső komponensei között a programozható összeköttetés hálózat teremt kapcsolatot (lokális, globális és regionális útvonalak segítségével, melyeket konfigurálható kapcsolók állítanak be)
- **DCM:** Digitális órajel menedzselő áramkör, amely képes a külső bejövő órajelből tetszőleges fázisú és frekvenciájú belső órajel(ek) előállítására



FPGA – további dedikált erőforrások

Általános erőforrások mellett a további **dedikált erőforrások** a következők (amelyek száma és felépítése az FPGA típusától és komplexitásától függően akár nagy-mértékben is változhat):

- **BRAM:** egy/két-portos Blokk-RAM memóriák, melyek összessége nagy mennyiségű (~x100Kbyte – akár ~x10Mbyte) adat/utasítás tárolását teszik lehetővé
- **MULT** / vagy **DSP** Blokkok: beágyazott szorzó áramköröket jelentenek, amelyek segítségével hagyományos szorzási műveletet, vagy a DSP blokk esetén akár bonyolultabb DSP MAC (szorzás-akkumulálás), valamint aritmetikai (kivonás) és logikai műveleteket is végrehajthatunk, nagy sebességgel
- **Beágyazott processzor(ok):**
 - Tetszés szerint konfigurálható / [beágyazható](#) ún. *soft-processzor* mag(ok)
 - Példa: Xilinx PicoBlaze, **Xilinx MicroBlaze**, Altera Nios II stb.
 - Fixen [beágyazott](#), ún. *hard-processzor* mag(ok)
 - Példa: IBM PowerPC 405/450 (Xilinx Virtex 2 Pro, Virtex-4 FXT, Virtex-5 FXT), ARM Cortex A9 (Xilinx Zync, illetve Altera Cyclone V, Stratix V, Arria V) stb.

Xilinx FPGA családok *

■ Nagy teljesítmény

- Virtex (1998)
 - 50K-1M kapu, 0.22μm
- Virtex-E/EM (1999)
 - 50K-4M kapu, 0.18μm
- Virtex-II (1999)
 - 40K-8M kapu, 0.15μm
- Virtex-II Pro/X (2002)
 - 50K-10M kapu, 0.13μm
- Virtex-4 (2004) [LX, FX, SX]
 - 50K-10M kapu, 90nm
- Virtex-5 (2006) [LX, LXT, SXT]
 - 65nm
- Virtex-5 FXT, TXT (2008)
 - 65nm
- Virtex-6 LXT, SXT (2009)
 - 40nm

- Virtex-7 (2011)
 - 28nm

- Kintex-7 (2011)
 - 28nm

■ Alacsony költség

- Spartan-II (2000)
 - 15K-200K kapu, 0.22μm
- Spartan-II E (2001)
 - 50K-600K kapu, 0.18μm
- Spartan-3 (2003)
 - 50K-5M kapu, 90nm
- **Spartan-3E (2005)**
 - **100K-1.6M kapu, 90nm**
- Spartan-3AN (2006)
 - 50K-1.4M kapu, 90nm
- Spartan-3A - DSP (2006)
 - 1.8M-3.4M kapu, 90nm
- Spartan-6 LX, LXT (2009)
 - 45nm

...

- Artix-7 (2011)
 - 28nm

- **Zynq-7000 SoC (2012)**
 - **28nm**


FPGA létjogosultsága?

A mai modern FPGA-k a

- nagyfokú flexibilitásukkal,
- nagy számítási teljesítményükkel,
- és (ASIC-el szemben) gyors prototípus-fejlesztési,
- ezáltal olcsó kihozatali (piacra kerülési) költségükkel

igen jó alternatívát teremtenek a mikrovezérlős (uC), illetve DSP-alapú implementációk kiváltására (pl. jelfeldolgozás, hálózati titkosítás, beágyazott rendszerek, stb. alkalmazásai területén).

Fejlődésüket jól tükrözi a mikroprocesszorok és az FPGA áramköri technológia fejlődési üteme között fennálló nagyfokú hasonlóság a méretcsökkenésnek (scaling-down) - *Gordon Moore-törvénynek* megfelelően.



Pannon Egyetem - VIRT

Tanszéken lévő fejlesztő kártyák

- **Digilent ZYBO (Xilinx Zynq)**
- Digilent ZED (Xilinx Zynq)
- **Digilent Nexys-2 (Xilinx Spartan-3E)**
- Digilent Atlys (Xilinx Spartan-6)
- Xilinx ML506 (Virtex-5)
- Xilinx VirtexII-Pro (VirtexII-Pro)
- Celoxica RC203/RC200 (Xilinx Virtex-II)

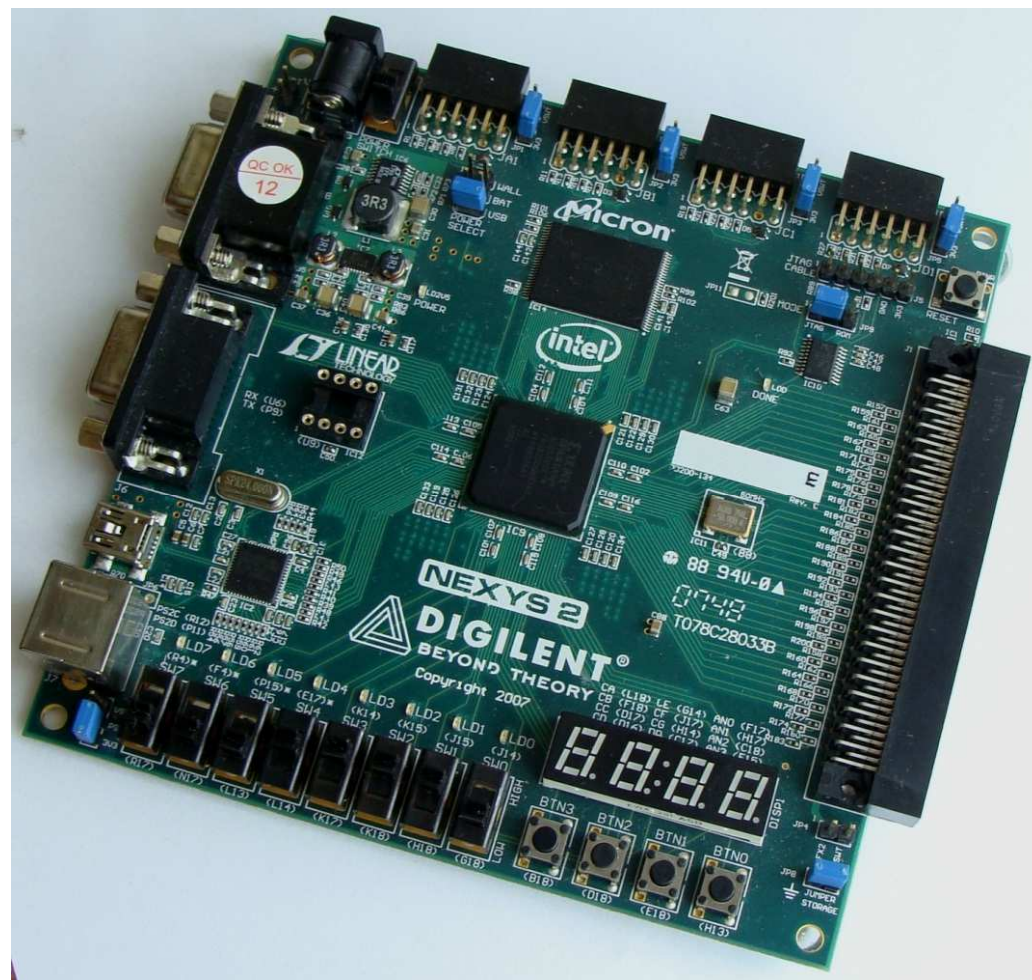
SW:

- Xilinx ISE Design Suite 14.7
- Xilinx Vivado 2015.2

Digilent Nexys-2 fejlesztő kártya

Nexys™2 Xilinx Spartan-3E FPGA fejlesztő kártya

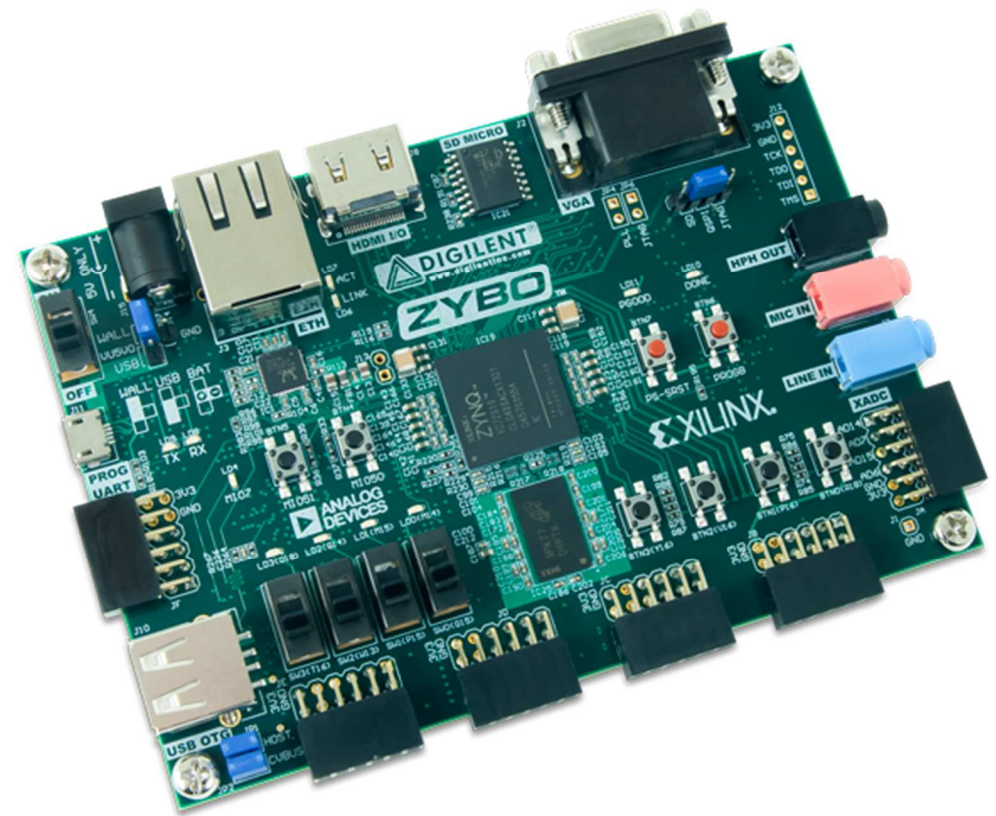
- Xilinx Spartan-3E FPGA, 500K / 1200K ekvivalens kapuval
- USB2 port (táp, konfiguráció, adat-transzfer egyben)
- Xilinx ISE/Webpack/EDK
- 16MB Micron PSDRAM
- 16MB Intel StrataFlash Flash
- Xilinx Platform Flash ROM
- 50MHz osszcillátor
- 75 FPGA I/O's (1 nagy-sebességű Hirose FX2 konnektor és 4 db 2x6 PMOD konnektor)
- GPIO: 8 LED, 4-jegyű 7-szegmenses kijelző, 4 nyomógomb, 8 kapcsoló
- VGA, PS/2, Soros port



Digilent ZYBO fejlesztő kártya

ZYBO™ Zynq FPGA fejlesztő kártya

- *Xilinx Zynq-7000 (Z-7010)*
 - 650 MHz dual ARM Cortex-A9 magok (PS)
 - 8-csatornás DMA vezérlő (PS)
 - 1G ethernet, I2C, SPI, USB-OTG vezérlő (PS)
 - Artix-7 FPGA logika (PL)
 - 28Kbyte logikai cella, 240 Kbyte BRAM, 80 DSP szorzó(PL)
 - 12-bites, 1MSPS XADC (PL)
- 512 Mbyte DDR3 x32-bit (adatbusz), 1050Mbps sávszélességgel
- Tri-mode 10/100/1000 Ethernet PHY
- HDMI port: Dual role (source/sink)
- VGA port: 16-bites
- uSD kártya: OS tartalom tárolása
- OTG USB 2.0 (host és device)
- Audio codec
- 128Mbit x Serial Flash/QSPI (konfiguráció tárolási célokra)
- JTAG-USB programozhatóság, UART-USB vezérlő
- GPIO: 5 LED, 6 nyomógomb, 4 kapcsoló
- 4+1 PMOD csatlakozó (A/D átalakítóhoz)



„FPGA” témájú tárgyak a Pannon Egyetemen – VIRT tanszék (2015)

- **(VEMIVI4144B) FPGA-alapú beágyazott rendszerek**

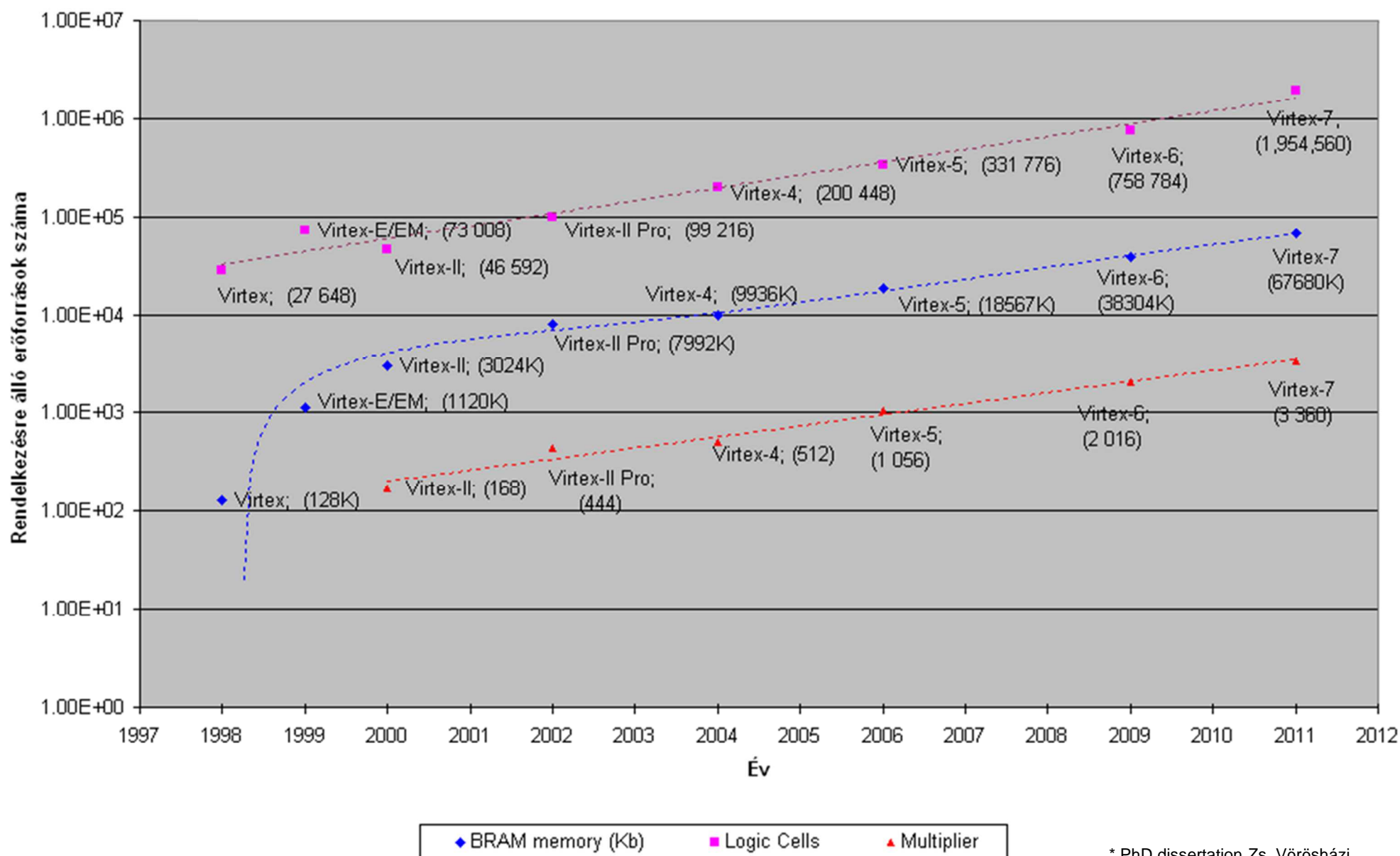
- <http://virt.uni-pannon.hu/index.php/oktatas/tantargyak/1030-fpga-alapu-beagyazott-rendszerek>

- **(VEMIVIB544T) Tervezési módszerek programozható logikai alkatrészekkel (VHDL)**

- <http://virt.uni-pannon.hu/index.php/oktatas/tantargyak/770-tervezesi-modszerek-programozhato-digit-vlsi-alkatreszekkel>

Nagy-teljesítményű Xilinx Virtex FPGA család erőforrásainak alakulása (1998-2012 időszakban)

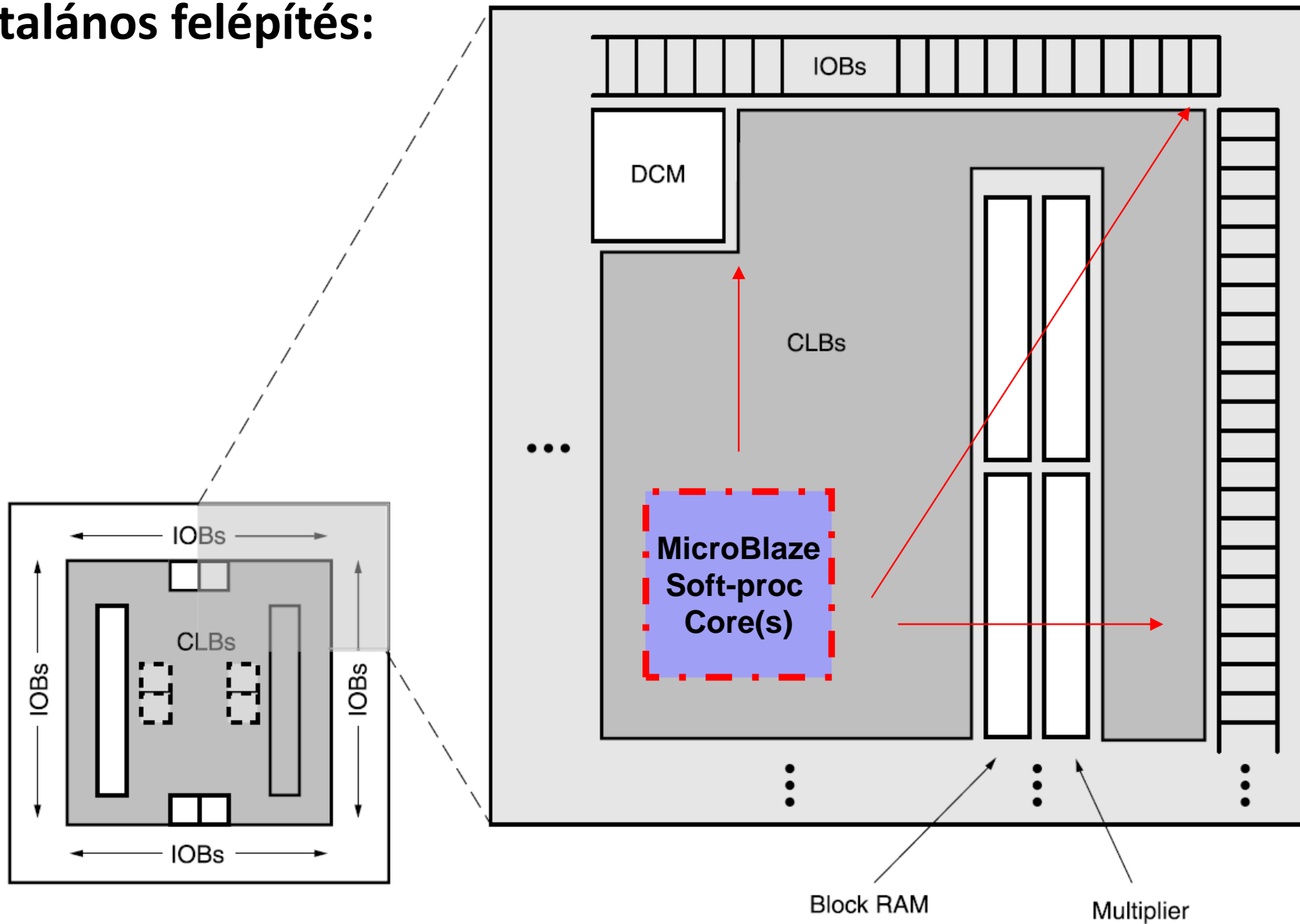
*Tranzisztorok fizikai méretcsökkenésének („Scaling-down”) hatása a fejlődésre





Pl. Spartan-3E architektúra

Általános felépítés:

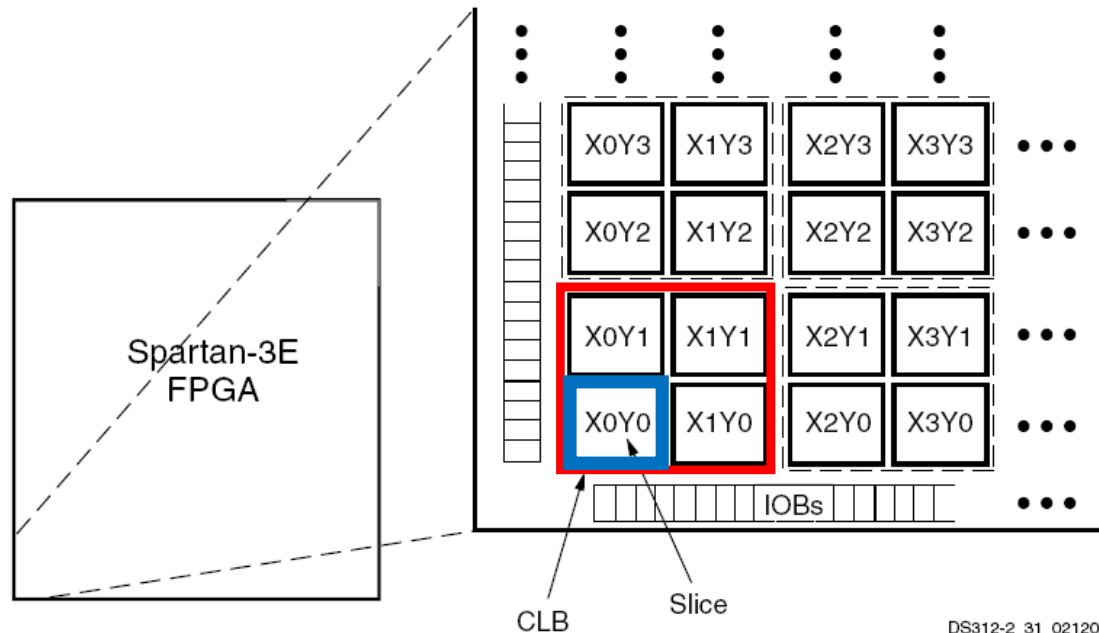


Spartan-3E FPGA építőelemei



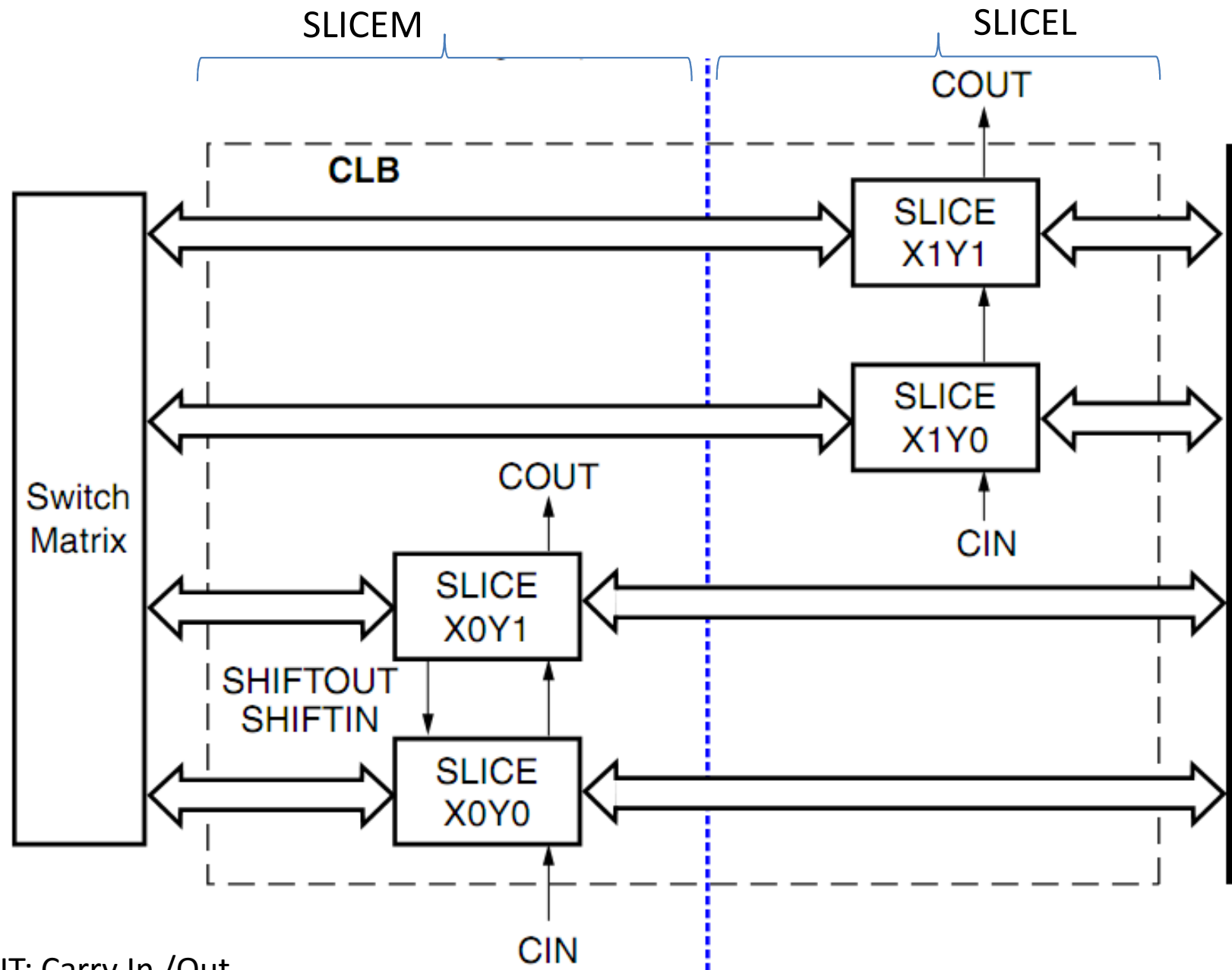
- CLB: Konfigurálható Logikai blokkok
 - Slice, mint *logika*: LUT, D-FF, MUX, Carry Logika
 - Slice, mint *memória*: SRL-16x1, RAM-16x1
- IOB: I/O blokkok
- MULT: 18x18 bites előjeles (2's) szorzó(k)
- BRAM: konfigurálható 18 Kbites (~2 Kbyte + paritás) egy/két-portos memóri(ák)
- DCM: Digitális órajel menedzser blokkok
- Beágyazható processzor(ok):
 - Csak szoft-processzor mag(ok) alakíthatóak ki
 - PI: Xilinx MicroBlaze, PicoBlaze, (esetleg külső független IP – szellemi terméke integrálható)

1 CLB = 4
Slice !



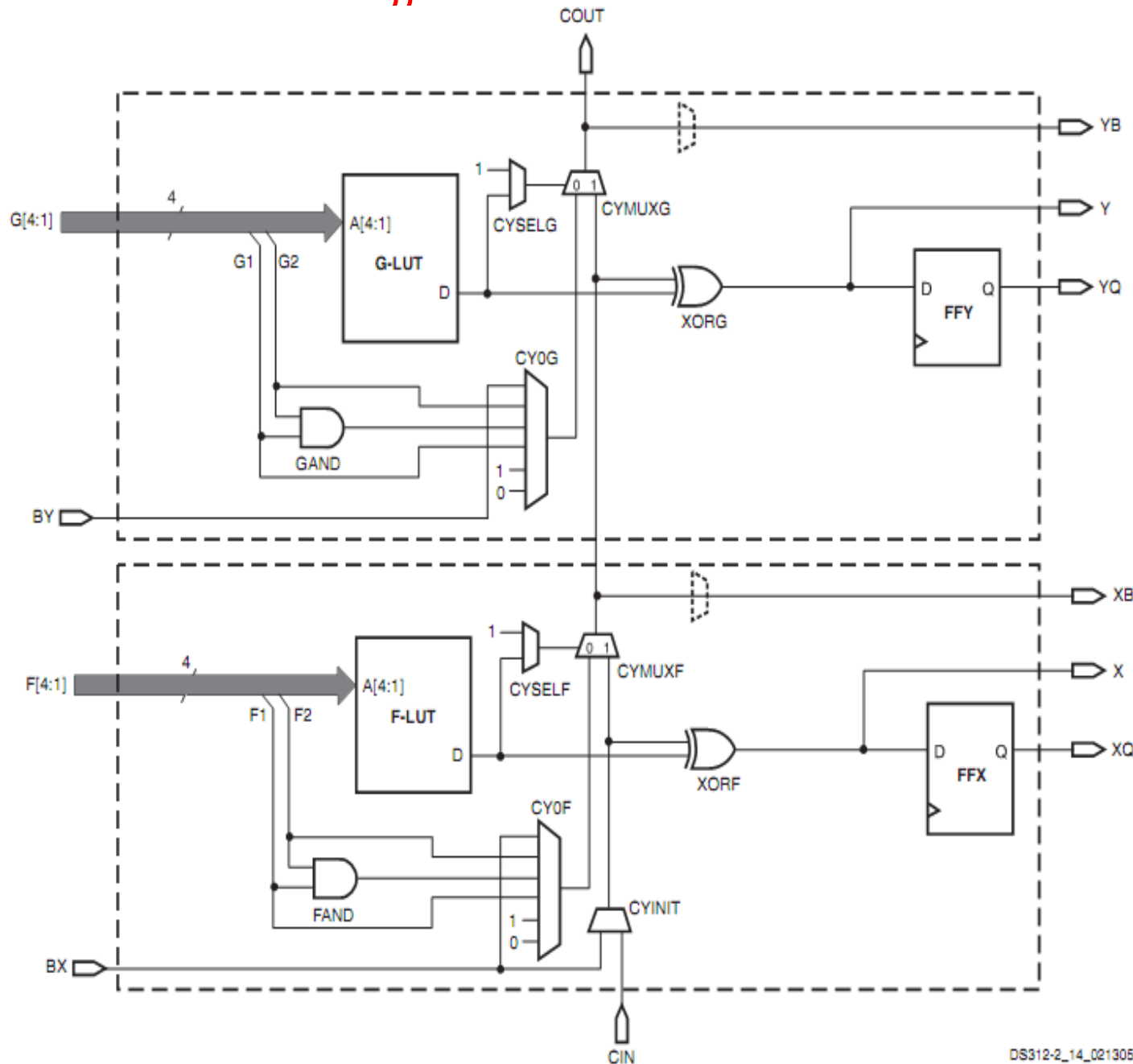
Device	CLB Rows	CLB Columns	CLB Total ⁽¹⁾	Slices	LUTs / Flip-Flops	Equivalent Logic Cells	RAM16 / SRL16	Distributed RAM Bits
XC3S100E	22	16	240	960	1,920	2,160	960	15,360
XC3S250E	34	26	612	2,448	4,896	5,508	2,448	39,168
XC3S500E	46	34	1,164	4,656	9,312	10,476	4,656	74,496
XC3S1200E	60	46	2,168	8,672	17,344	19,512	8,672	138,752
XC3S1600E	76	58	3,688	14,752	29,504	33,192	14,752	236,032

CLB-k összekapcsolása



Összeköttetés a szomszédos CLB-k felé

Slice – „szelet”



Look-Up Table konfigurálható:

- 4-bemenetű LUT-ként (F,G)
- 16x1-bit szinkron RAM-ként
- 16-bit shift regiszterként

Tároló elemek

- D-típusú flip-flop-ok, vagy latch-ek

További Logikai áramkörök

- F5MUX multiplexer
 - bármely 5-bemenetű függvény
 - 4:1 multiplexer
- FiMUX multiplexer
 - bármely 6-bemenetű függvény
 - 8:1 multiplexer

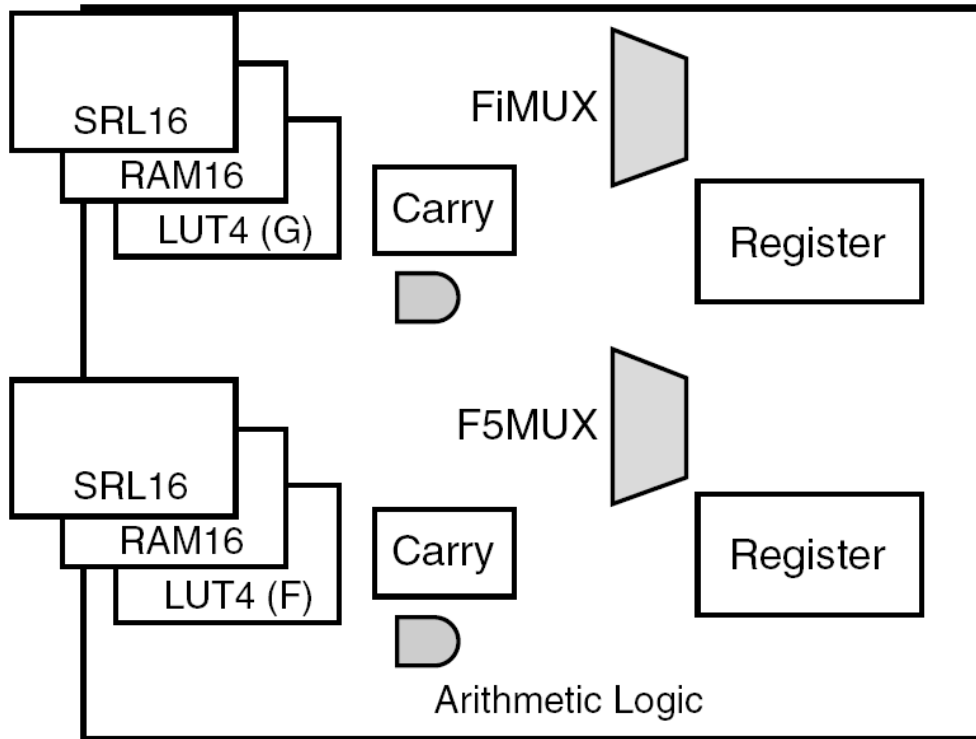
Aritmetikai Logikai Egység

- Dedikált carry logika (CYSEL_, CYMUX_, CY0_)
- Dedikált AND kapuk (GAND, FAND)

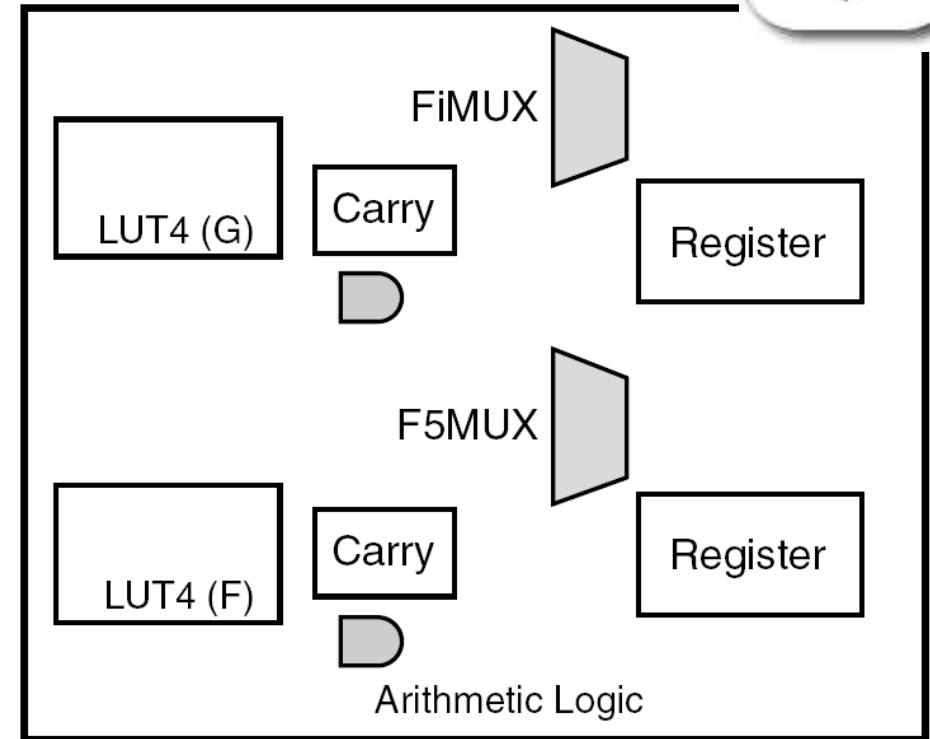
Milyen erőforrásként konfigurálhatók a Slice-ok?



1 CLB = 4 slice (Spartan 3E esetén) = 2-2 SLICEM-SLICEL pár



SLICEM



SLICEL

DS312-2_13_020905

Slice”M”: M, mint memória elem lehet:

- A.) *LUT-4*: Logika 4-bemenetű
- B.) *RAM16*: Distributed (elosztott) RAM – regiszterekből 16x1-bit,
- C.) *SRL16*: Shift Regiszter 16x1-bit

Slice”L”: L, mint logika lehet:

- -.) Csak *LUT-4*!
- D.) Dedikált *MUX*
- E.) *Carry logika*

A.) Look-Up Table

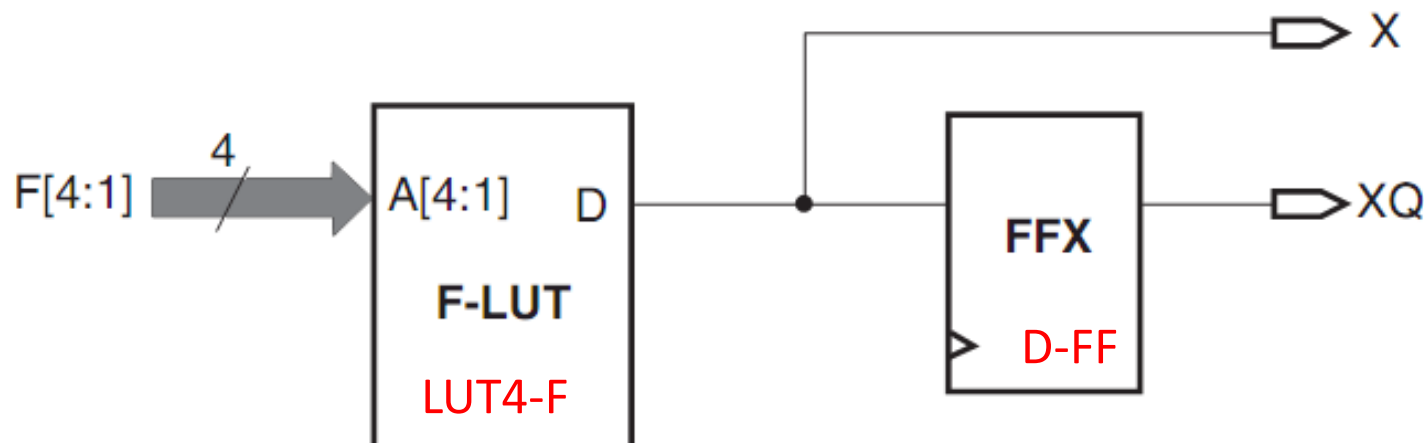


LUT: memória alapú

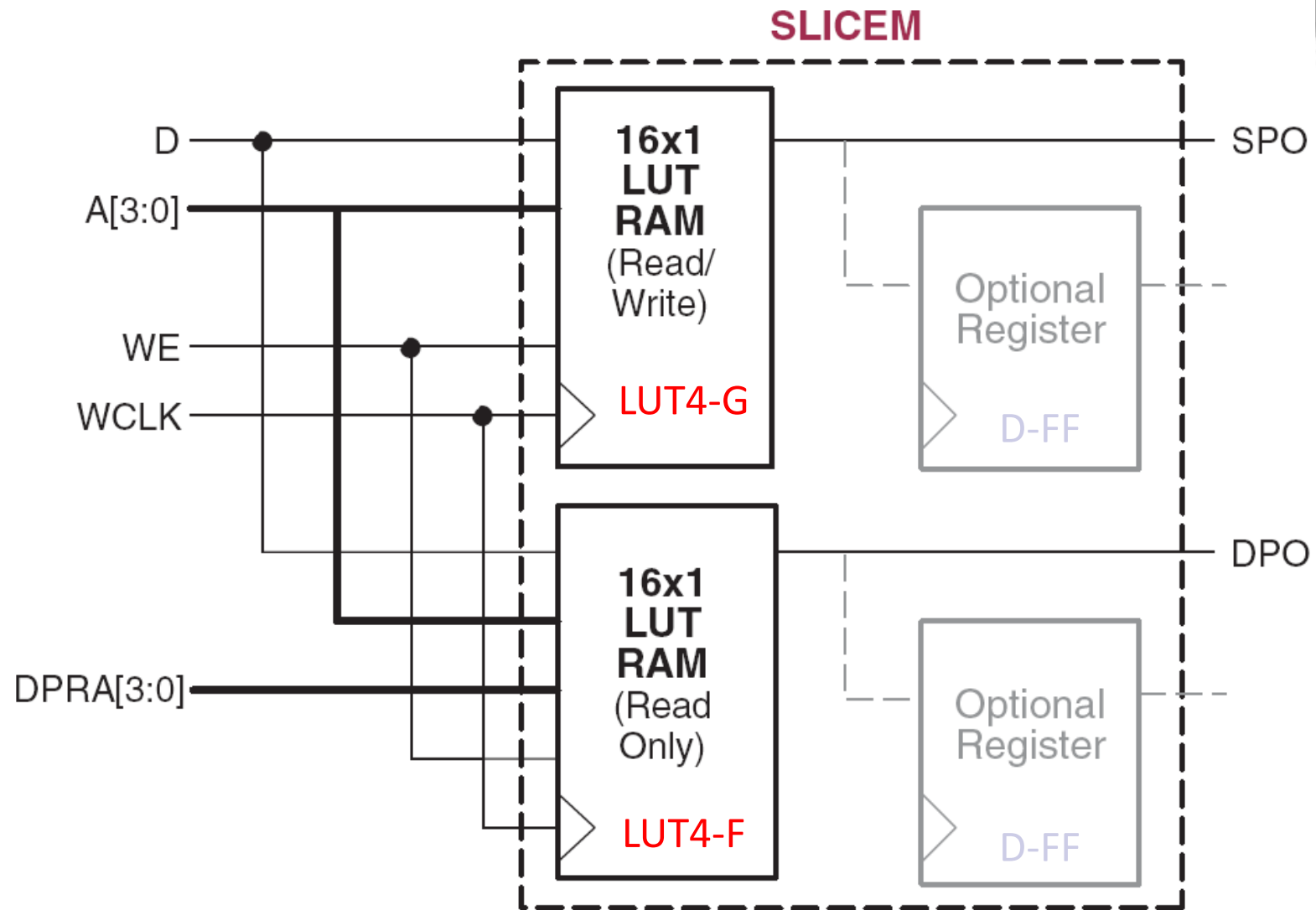
-egyrészt logikai függvény generátor
(*4-bemenetű 1-kimenetű fgv.)

- *Megj. Nagyobb FPGA-k esetén 6-bemenetű LUT-6 (pl Virtex-6, -7, Spartan-6 sorozatok)

-másrészt elosztott memória v. shift regiszter (csak SLICEM párokban)

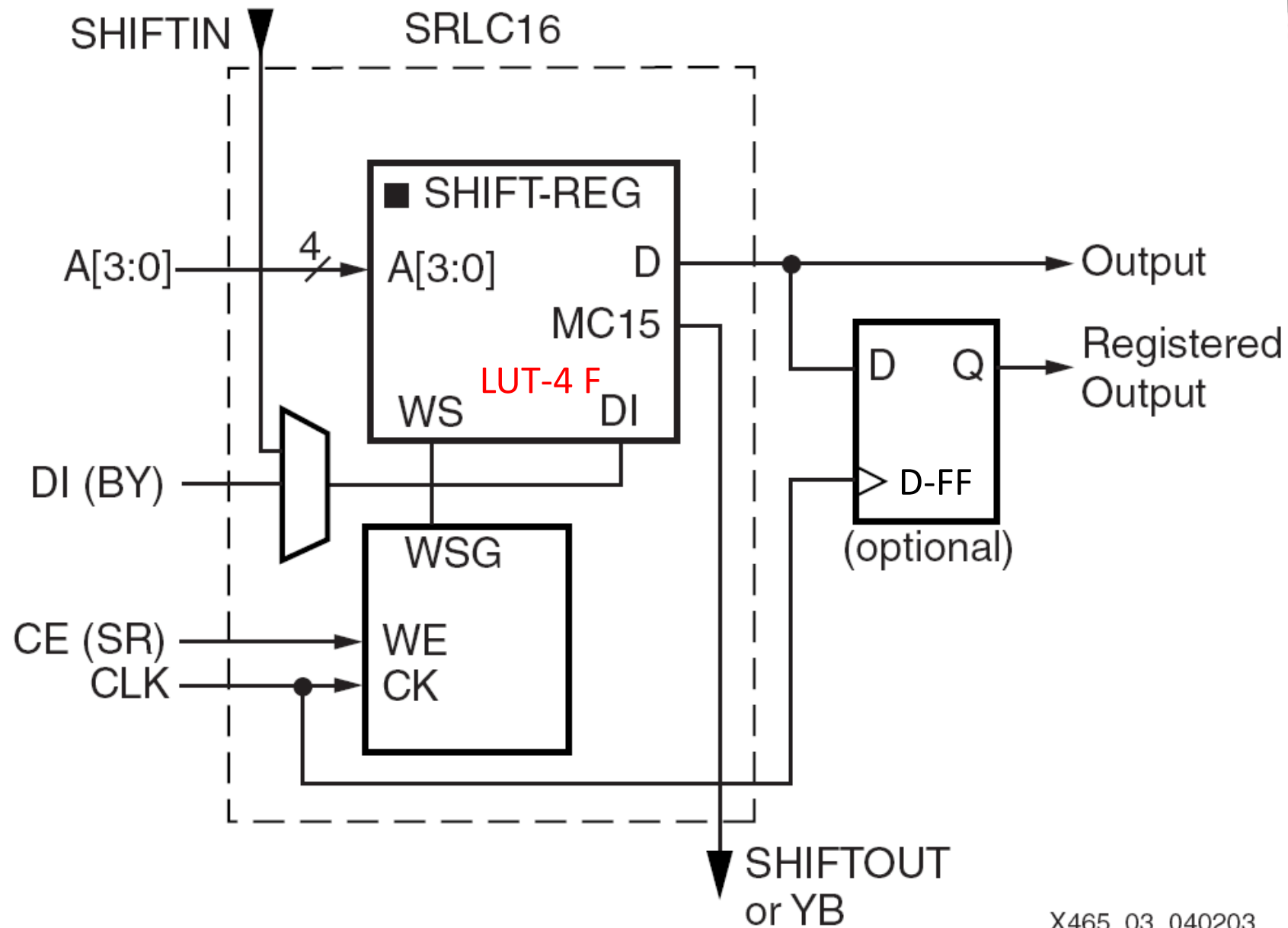


B.) RAM16X1D két-portos

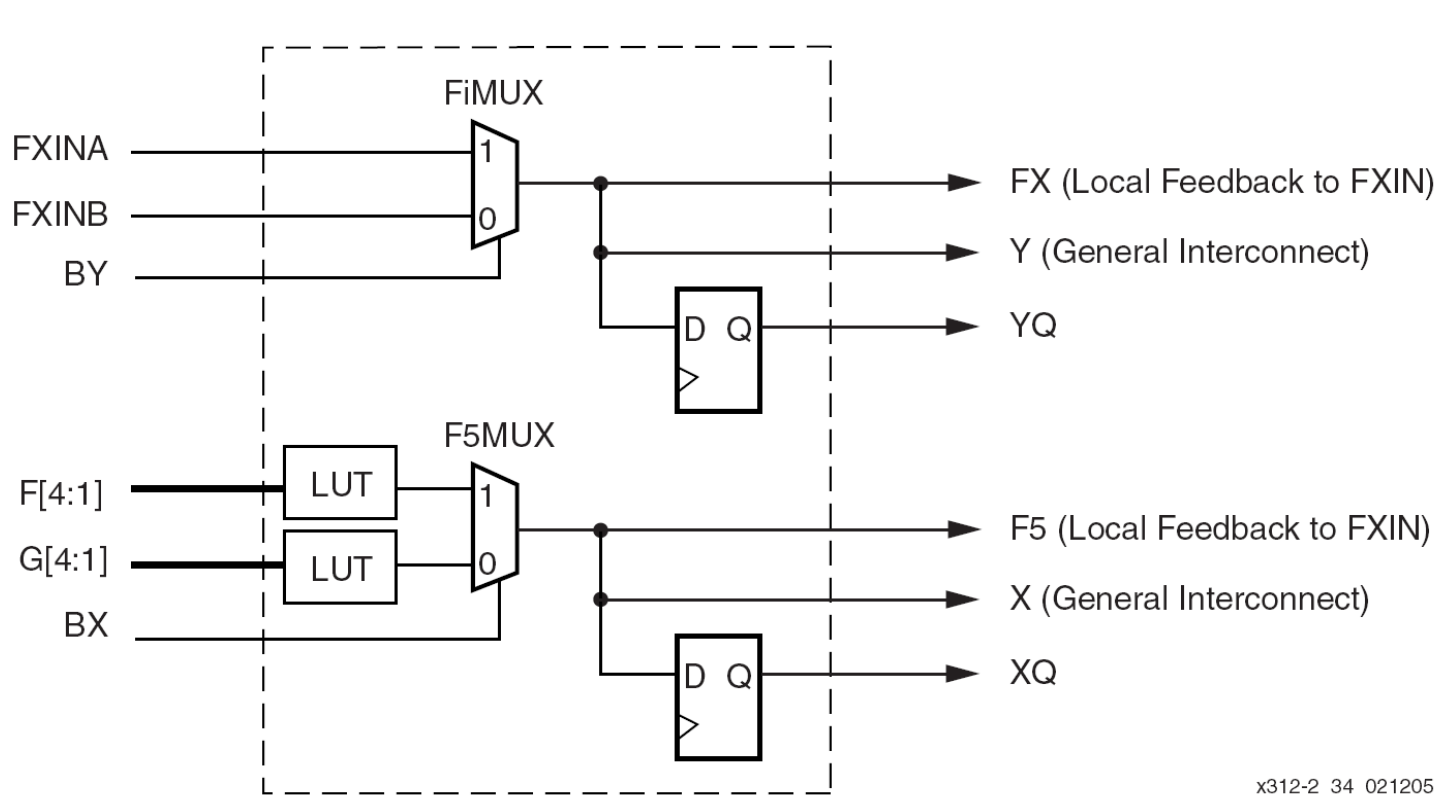


DS312-2_41_021305

C.) SRL-16x1 shift regiszter



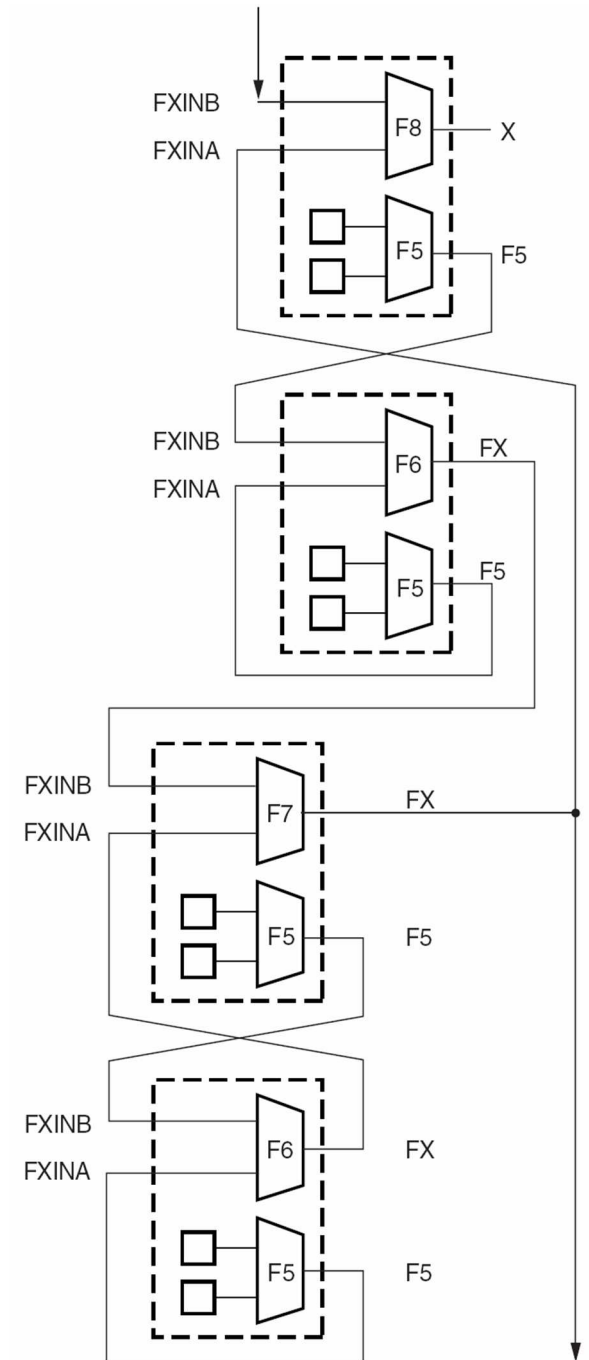
D.) Dedikált MUX-ok



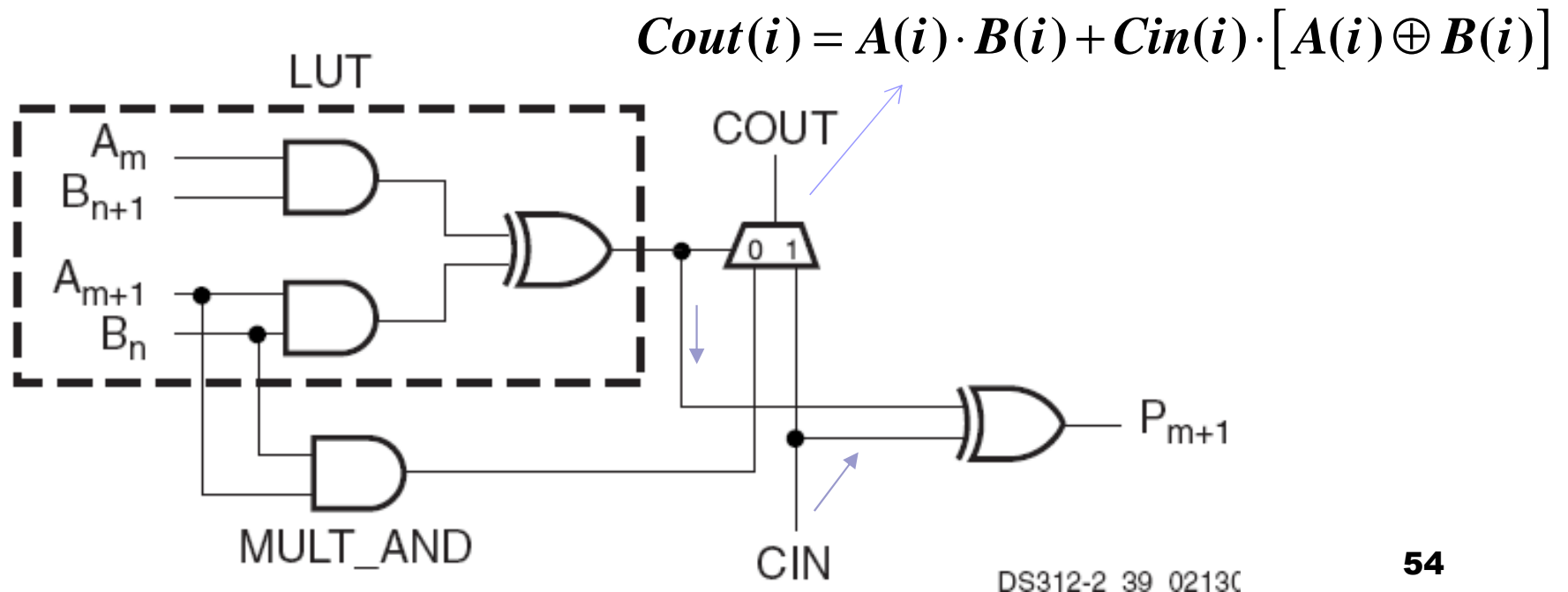
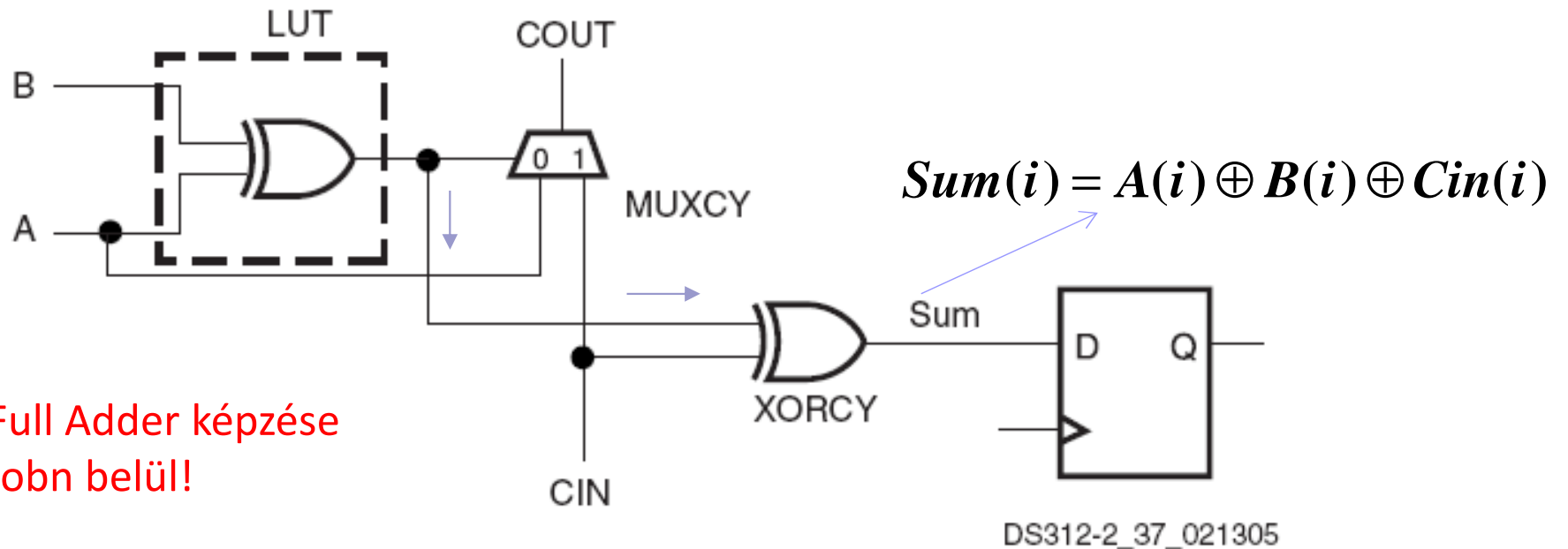
x312-2_34_021205

Table 11: Mux Capabilities

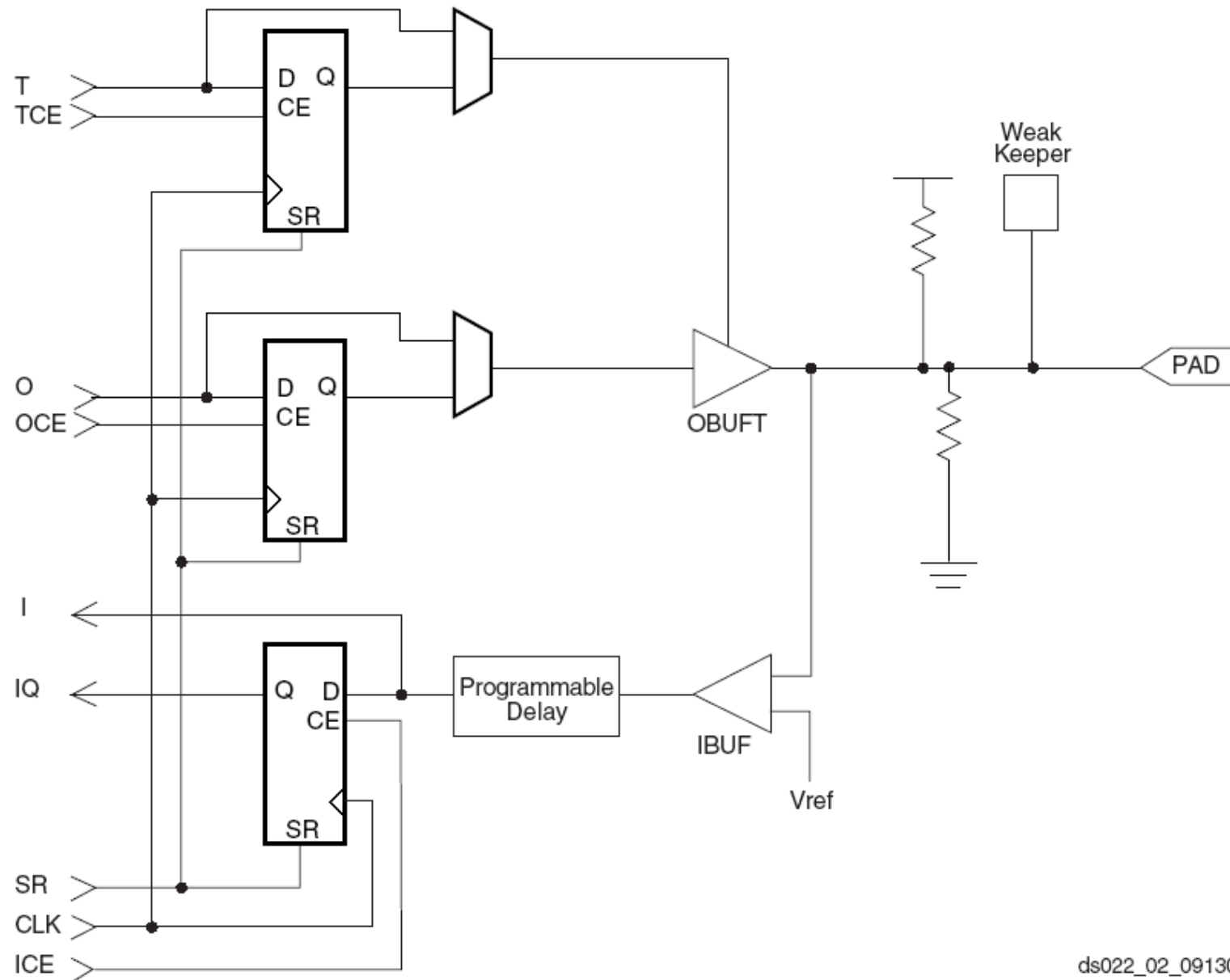
Mux	Usage	Input Source	Total Number of Inputs per Function		
			For Any Function	For Mux	For Limited Functions
F5MUX	F5MUX	LUTs	5	6 (4:1 mux)	9
FiMUX	F6MUX	F5MUX	6	11 (8:1 mux)	19
	F7MUX	F6MUX	7	20 (16:1 mux)	39
	F8MUX	F7MUX	8	37 (32:1 mux)	79



Dedikált Carry logika



IOB – Programozható I/O Blokkok



IOB – programozható I/O blokkok



■ Támogatott IO szabványok

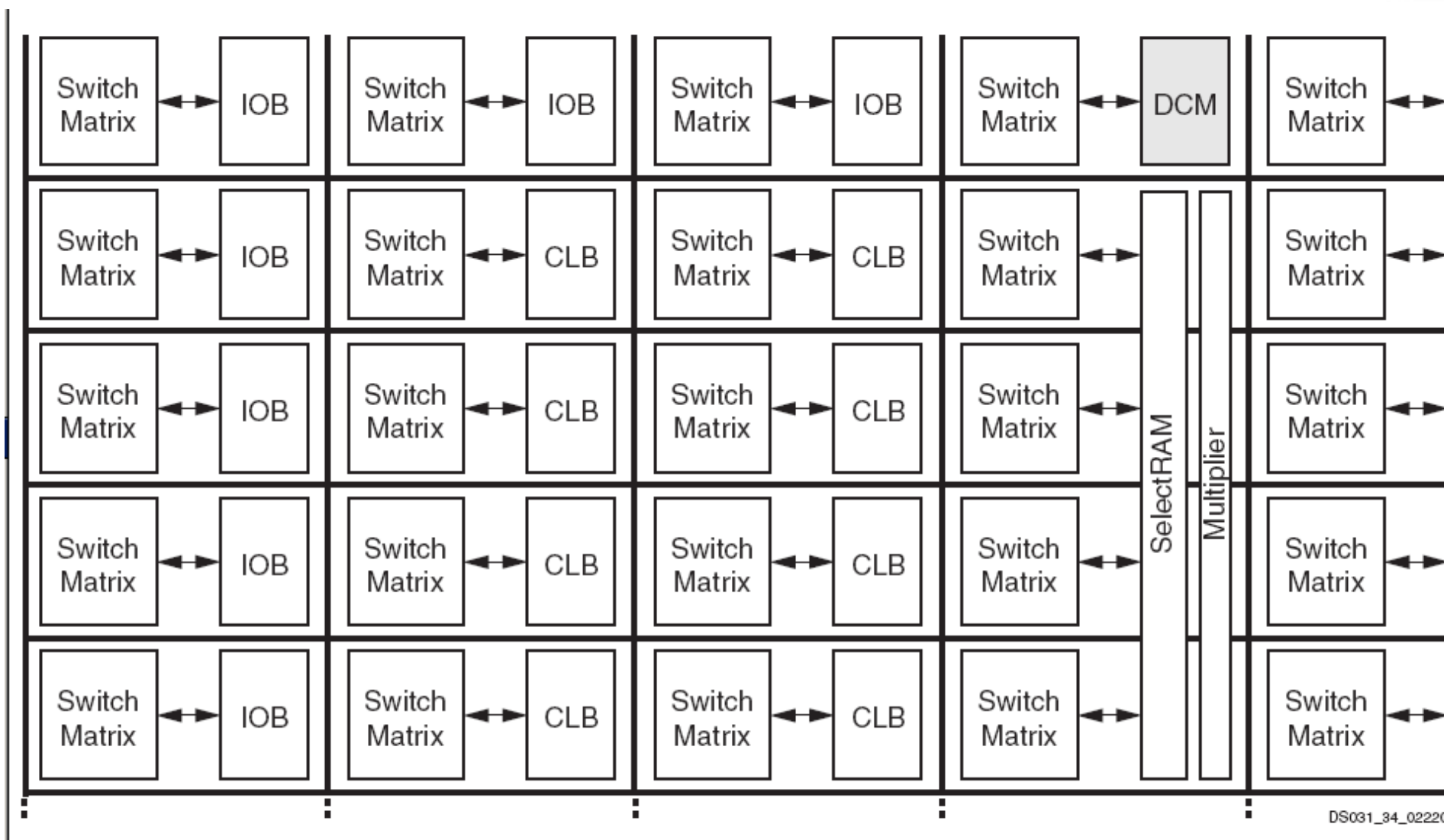
□ „Single-ended” jelek:

- 3.3V low-voltage TTL (LVTTTL),
- Low-voltage CMOS (LVCMOS) 3.3V, 2.5V, 1.8V, 1.5V, 1.2V
- 3V PCI @ 33 MHz / 66 MHz
- HSTL I - III @ 1.8V (memória)
- SSTL I @ 1.8V, 2.5V (memória)

□ Differenciális jelek:

- LVDS
- Bus LVDS
- mini-LVDS
- Differential HSTL (1.8V, Types I and III)
- Differential SSTL (2.5V, 1.8V, Type I)

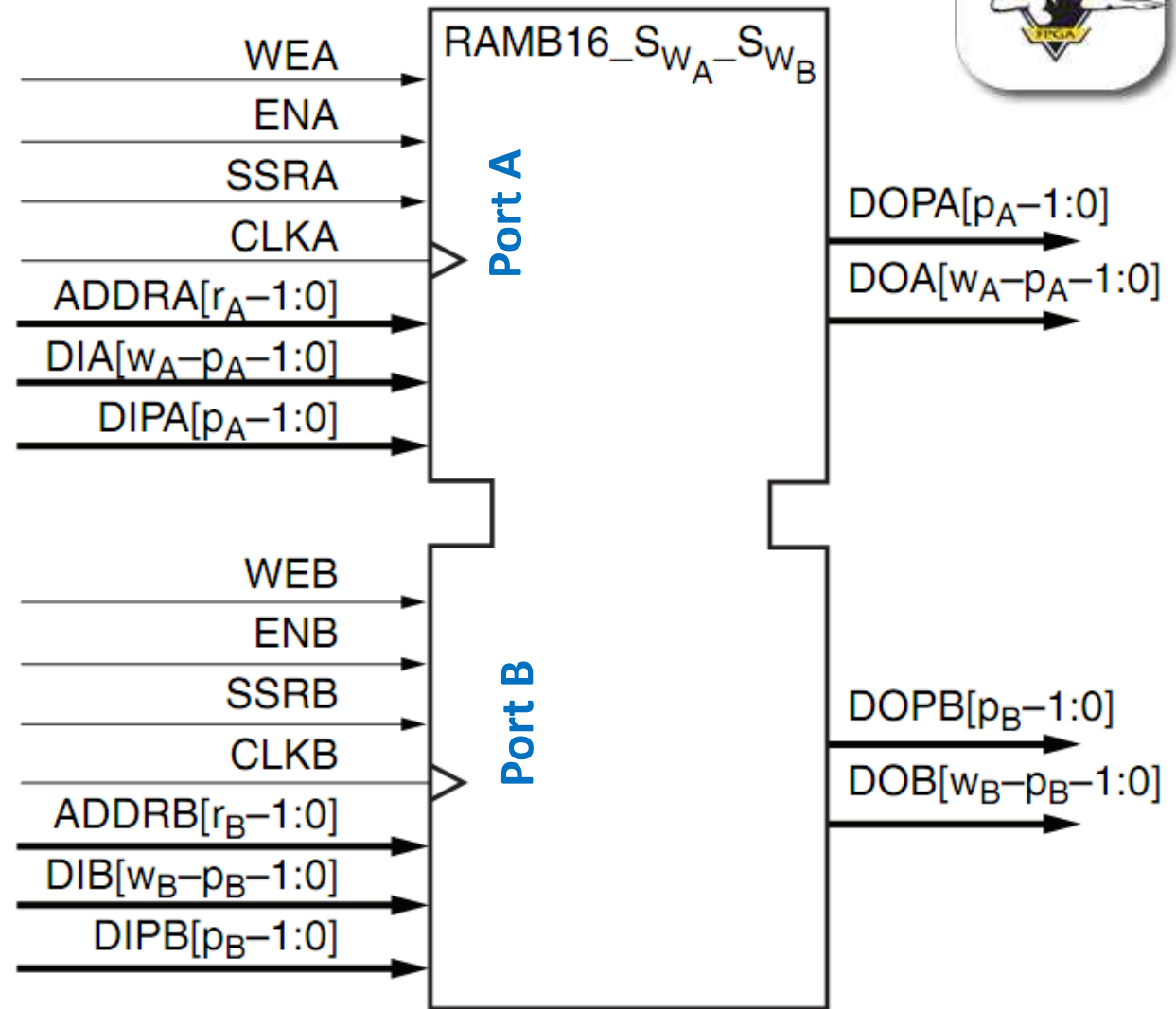
Összeköttetési erőforrások



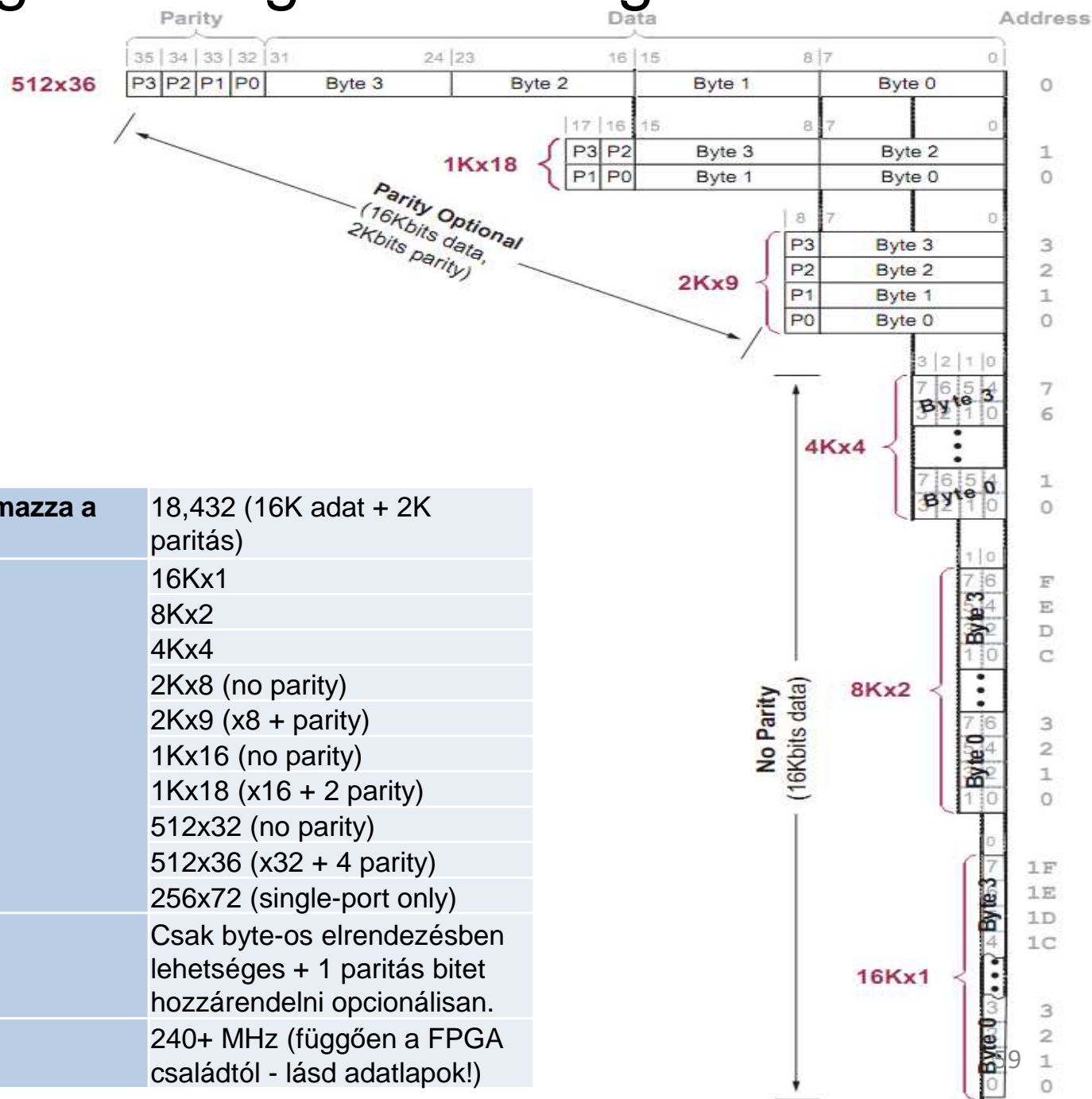
DS031_34_022205

Blokk-RAM

- „SelectRAM” tulajdonság (konfigurálható)
 - FIFO, RAM, ROM-ként is
- Dedikált BRAM lehet:
 - Egy-portos
 - Két-portos (A,B)
 - Minkét portját (A, B) külön címezhetjük (RW)
 - Független adatbusz szélesség definiálható



BRAM tetszőleges konfigurálhatósága

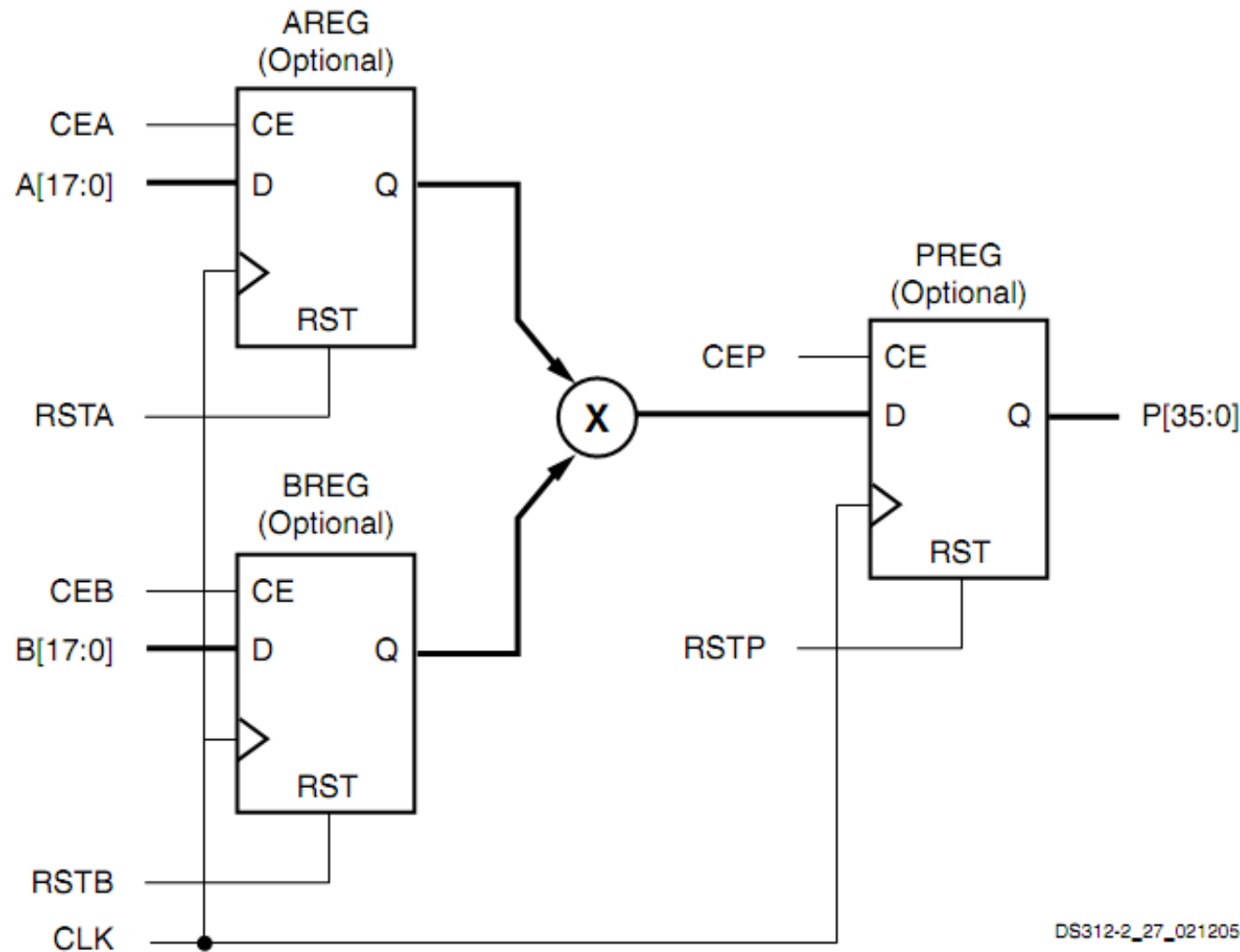


Teljes RAM kapacitás (bit), mely tartalmazza a paritásokat is	18,432 (16K adat + 2K paritás)
Memória szervezési módszerek:	16Kx1
	8Kx2
	4Kx4
	2Kx8 (no parity)
	2Kx9 (x8 + parity)
	1Kx16 (no parity)
	1Kx18 (x16 + 2 parity)
	512x32 (no parity)
	512x36 (x32 + 4 parity)
	256x72 (single-port only)
Paritás:	Csak byte-os elrendezésben lehetséges + 1 paritás bitet hozzárendelni opcionálisan.
Sebesség:	240+ MHz (függően a FPGA családtól - lásd adatlapok!)

Multiplier - Szorzó



- $P = A * B$ dedikált szorzó áramkör
 - 18x18 bites, 2-komplementens szorzást támogat
 - Opcionális szorzó-, szorzandó-, szorzat-regiszterek



DCM – Digital Clock Manager

Digitális órajel menedzser áramkör(ök)



- DCM-ek száma: 4 –
XC3S500E, 8 –
XC3S1200E

DLL: Delayed Locked Loop

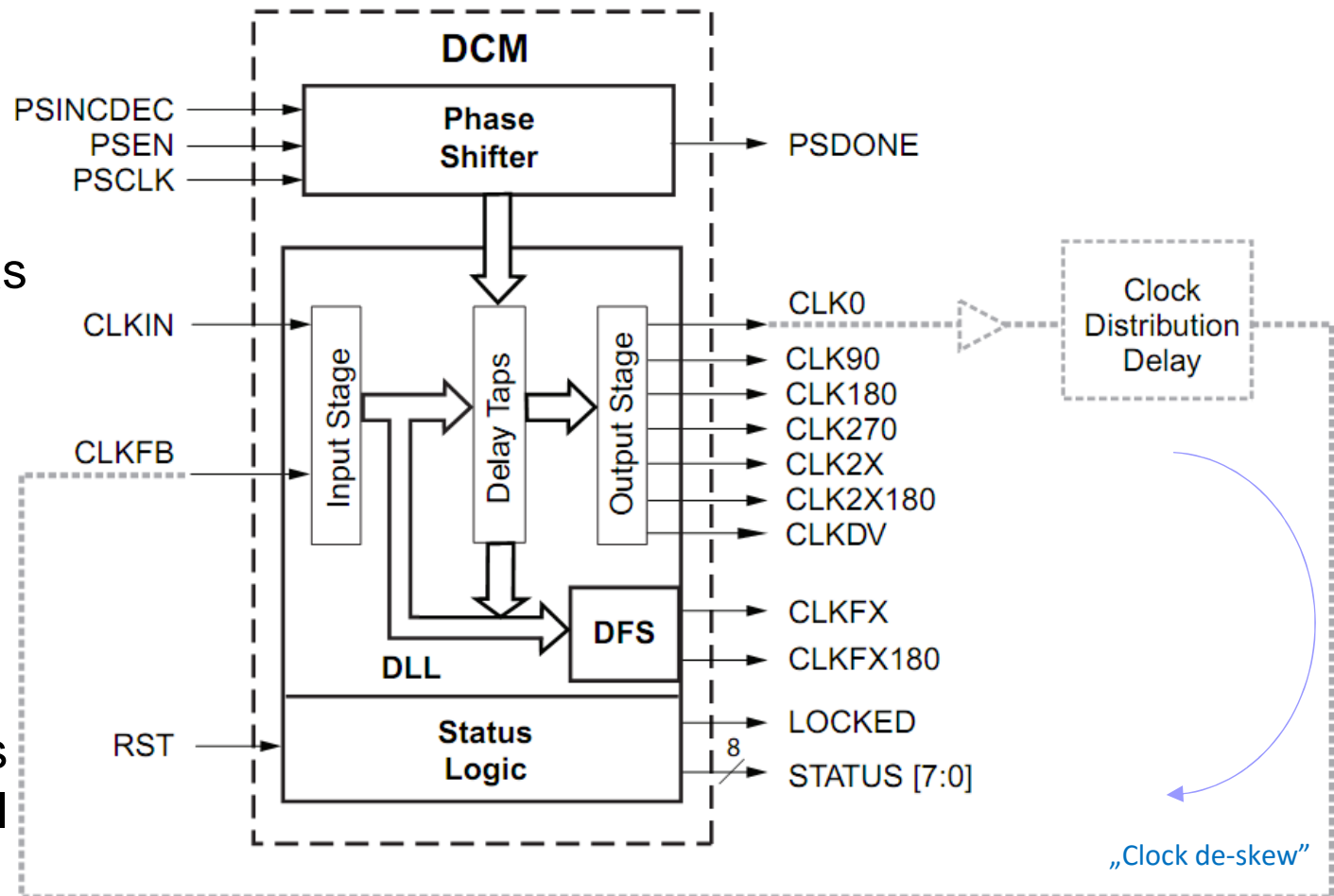
- Negyedelt fázis tolás: 0° , 90° , 180° , 270°
- Órajel szorzás (M)/ osztás (D) 1.5, 2, 2.5, 3, 4, 5, ...
16

- Tetszőleges órajel generálható

- 5 MHz – 333 MHz

DFS: Digitális Frekvencia szintézis

- Órajel duplázás / felezés
- Bemeneti, kimeneti órajel pufferelés
- Locked: DCM kimenetei a CLKIN-el fázisban vannak



PCB

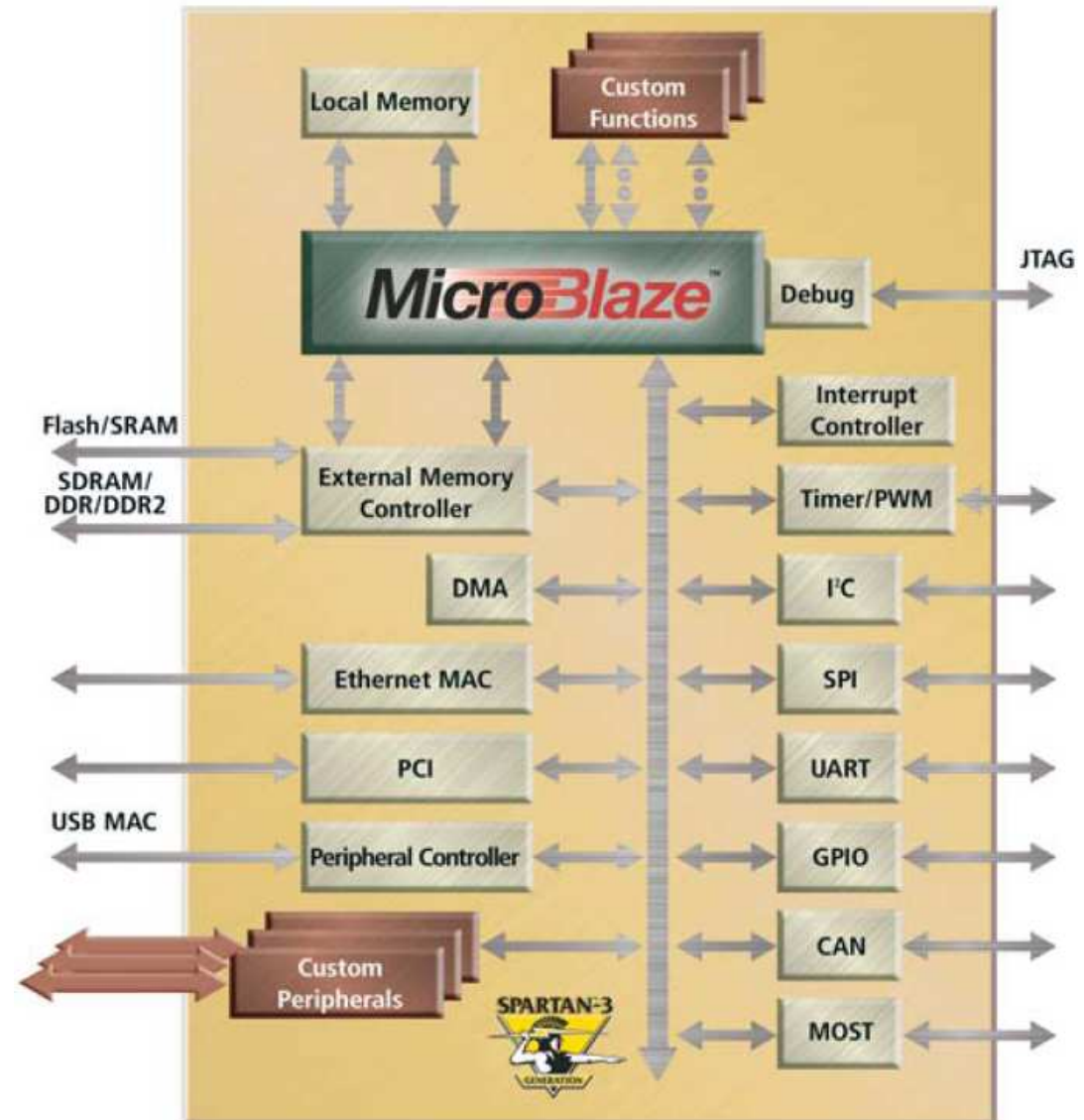
Beágyazott processzorok

- „Beágyazható” (lágy) **soft-processzor** magok:
 - Xilinx *PicoBlaze*: 8-bites (VHDL, Verilog HDL forrás)
 - Xilinx *MicroBlaze*: 32-bites (XPS – EDK/SDK támogatás!)*
 - PLB, OPB (régi), AXI buszrendszerekhez is csatlakoztatható
 - 3rd Party: nem-Xilinx gyártók processzorai (HDL)
- „Beágyazott” (kemény) **hard-processzor** magok:
 - Korábban IBM *PowerPC* 405/450 processzor (dedikált): 32-bites (EDK/SDK támogatás), PLB buszrendszerhez integrálható
 - de kizárólag Virtex II Pro, Virtex-4 FX, Virtex-5/6 FXT FPGA-kon!
 - *Jelenleg: ARM Cortex-A9* processzor (dedikált): kétmagos, AMBA-AXI buszrendszerhez integrálható
 - Xilinx Zynq APSoC: FPGA logikához integrált ARM magok

MicroBlaze magra épülő beágyazott rendszer

„Beágyazható” processzor

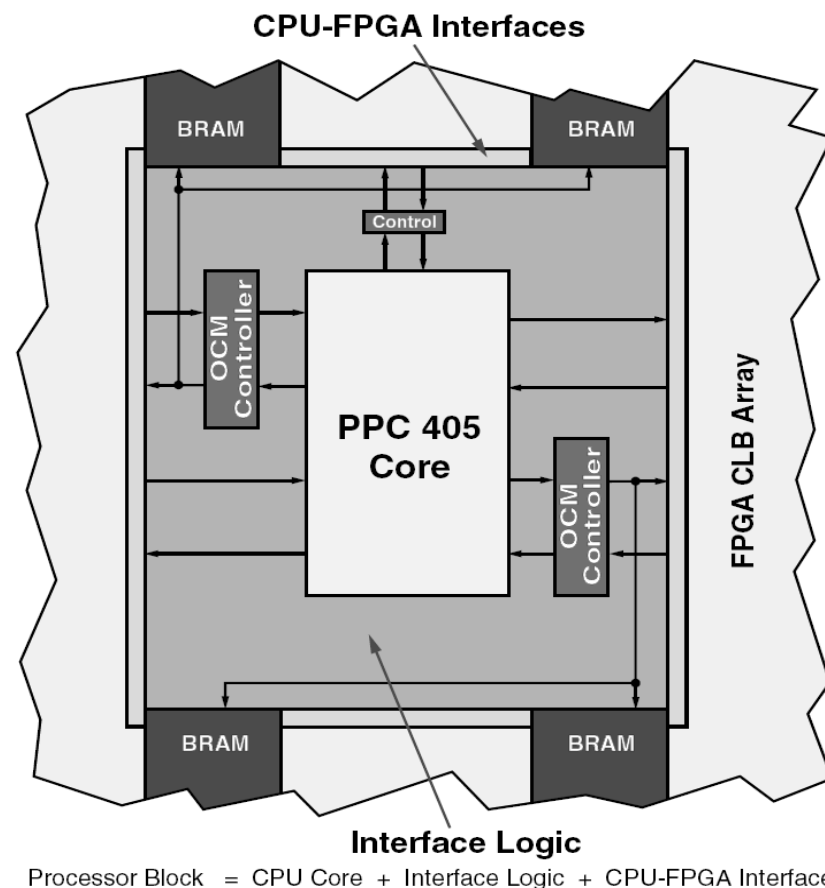
- RISC utasítás készlet architektúra
- 32-bites soft-processor core!
- 133+ MHz órajel (PLB busz)
- Harvard blokk-architektúra
- Kis fogyasztás: ~ mW/MHz
- 3/5 lépcsős adatvonal pipeline
- 32 darab 32-bites Általános célú regiszter
- utasítás cache / adat cache
- Időzítési lehetőségek (timer)
- Sokféle periféria, kommunikációs interfész csatlakoztatható (IP core-ok)
- **Minden** Xilinx FPGA-n implementálható, melynek elegendő erőforrása van és a fejlesztő szoftver támogatja!



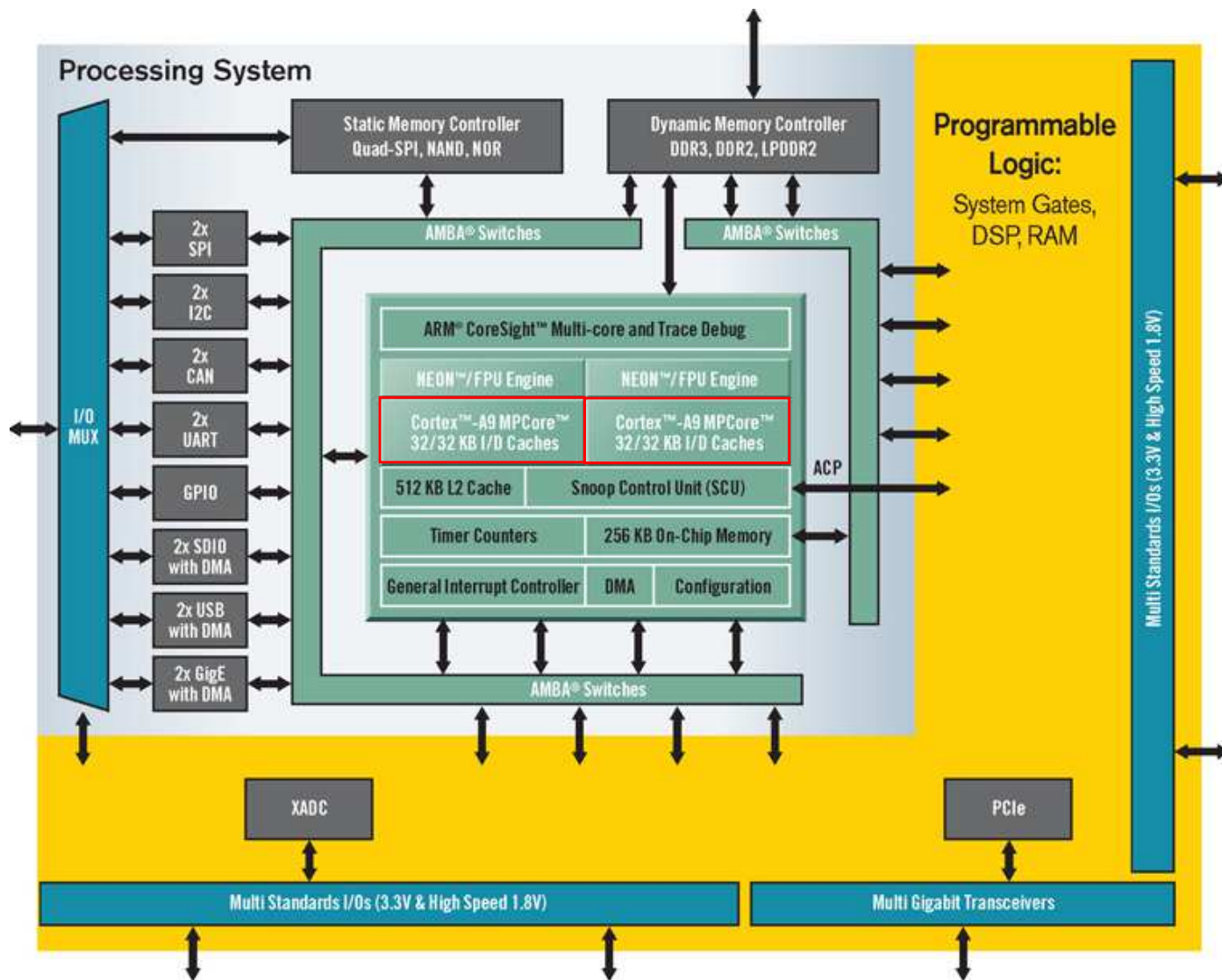
PowerPC - Hard-processzor mag

IBM PowerPC 405/450 blokk jellemzői:

- RISC utasítás készlet architektúra
- 32-bites Hard-processor core!
- Beágyazott 400+ MHz
- Harvard blokk-architektúra (ma ~650 MHz Virtex6 FXT)
- Kis fogyasztás: 0.9 mW/MHz
- 5 lépcsős adatvonal pipeline
- Hardveres szorzó/osztó egység
- 32 darab 32-bites Általános célú regiszter
- 16 KB 2-utas csoport asszociatív utasítás cache
- 16 KB 2-utas csoport asszociatív adat cache
- Memória Menedzsment egység (MMU)
- TLB: változtatható lap-méret (1 KB –től 16 MB-ig)
- Dedikált On-Chip Memória (OCM) intf.
- Időzítési lehetőségek (timer)
- Sokféle periféria, kommunikációs interfész csatlakoztatható (IP core-ok)
- Csak bizonyos FPGA-kon integrálva:
 - Virtex-II Pro, Virtex-4 FX, Virtex-5/6 FXT



Xilinx Zynq APSoC – ARM hard-processzor



- **Dupla ARM Cortex™-A9 MPCore (800MHz)**
- Neon: 32-/ 64-bit FPU
- 32kB utasítás & 32kB adat L1 Cache
- Közös 512kB L2 Cache
- 256kB on-chip memória
- Integrált DDR3, DDR2 and LPDDR2 DDR vezérlő
- Integrált 2x QSPI, NAND Flash and NOR Flash memória vezérlő
- Perifériák: 2x USB2.0 (OTG), 2x GbE, 2x CAN2.0B 2x SD/SDIO, 2x UART, 2x SPI, 2x I2C, 4x 32b GPIO, PCI Express® Gen2 x8
- Két 12-bit 1Msps ADC
- **28nm Programozható FPGA Logika:**
- 28k - 350k Logikai cella (~ 430k to 5.2M ekvivalens kapu)
- 240KB - 2180KB Block RAM
- 80 - 900 18x25 DSP szorzó (58 - 1080 GMACS -DSP teljesítmény)

■ **FPGA → APSoC** = All Programmable System on a Chip

■ (Kibővíthető Számítási Platform)



Szemponatok, absztrakciós szintek, modellezés

2. TERVEZÉSI MÓDSZEREK

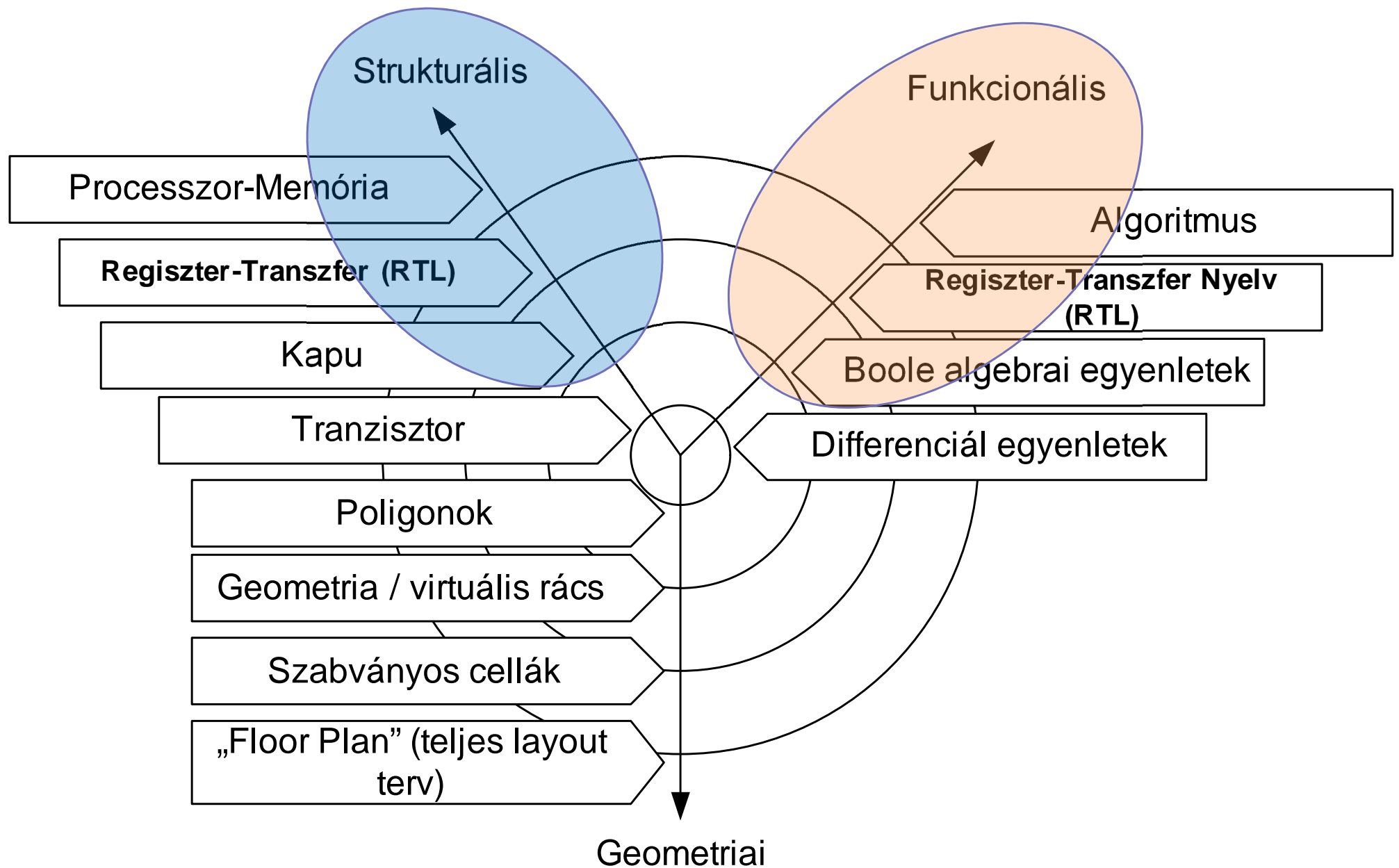
Tartományok – absztrakciós szintek

- 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-diagramm* szerint 3 lehetséges tartományt különböztethetünk meg:

- ☐ *Funkcionális/viselkedési,*
- ☐ *Strukturális és*
- ☐ *Geometriai.*

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

Gajski-Kuhn „Y-diagramm”



I. Funkcionális tartomány

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

■ A funkcionális tartomány a 'leg-absztraktabb' megközelítést jelenti, mivel a teljes rendszer viselkedése megadható az **algoritmikus szinten**.

■ A következő szint a **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 a hálózat feszültségében, és áramában történő *változásokat* hivatott leírni. ⁶⁹

II. Strukturális tartomány

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

■ A strukturális tartomány legfelső szintjén kell a fő komponenseket és összeköttetéseit definiálni, amelyet **processzor-memória kapcsolatnak** (**processor-memory switch**) is hívják.

■ A második szint itt is a **regiszter-átviteli szint**, 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 alappapuk, és összeköttetések segítségével építhető fel.

■ A negyedik, legalacsonyabb szinten a **tranzisztorok**, mint az *áramköri rajzolatok (layout)* elemi egységeit kell tekinteni (azokból építkezni).

III. 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: Electronic Design Automation

- Manapság a számítógéppel segített 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 generálódnak (**EDA**).

PLD/ FPGA: tárgyalat ismeretkörök

1. Mik azok a.) Programozható Logikai Eszközök és az b.) FPGA-k?
 - Összeköttetések programozhatósága
2. Tervezési módszerek (Design methods)
3. Tervezés folyamata (Design-flow)
4. Magas-szintű szintézis (HLS – High-Level Synthesis)
5. Fejlesztő környezetek, hardver leíró nyelvek (HDL - Hardware Description Languages)

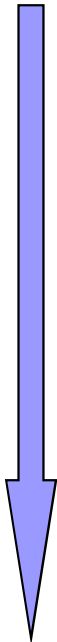


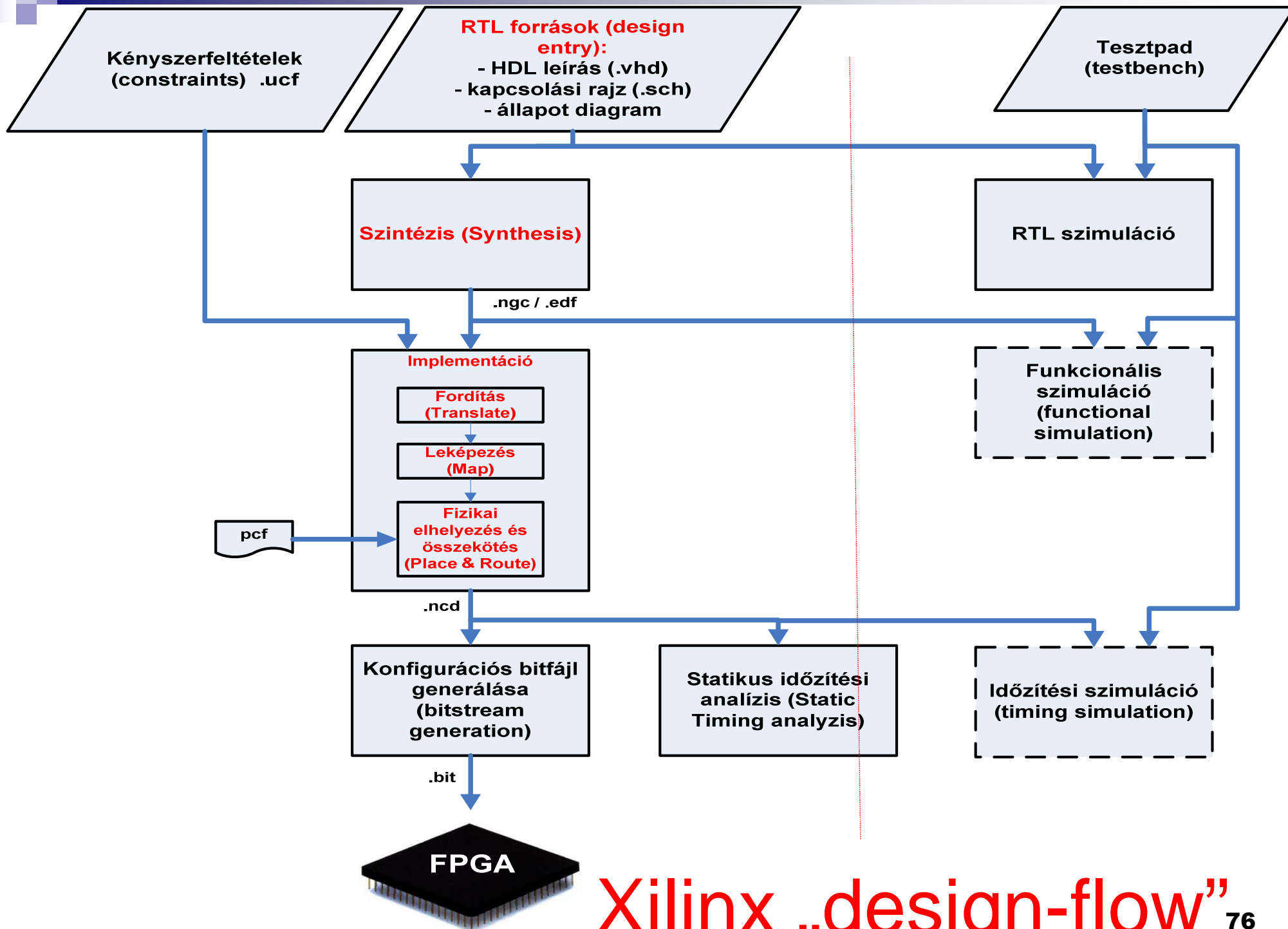
„DESIGN FLOW,, (XILINX FPGA-k esetében)

3. TERVEZÉS FOLYAMATA

Xilinx „design-flow”

- Az FPGA alapú rendszertervezés folyamatát a Xilinx ISE fejlesztő környezetben történő egymást követő **lépéseken** keresztül demonstráljuk. A fejlesztés egyszerűsített folyamatát a következő oldalon lévő ábra szemlélteti. Lépései:

- 
- ☐ Design Entry (pl. HDL forrás)
 - ☐ 1.) Szimuláció
 - ☐ 2.) Szintézis és
 - ☐ 3.) Implementáció
 - ☐ 4.) Időzítési analízis
 - ☐ 5.) Bitfolyam (konfiguráció) létrehozása



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 - *ucf*) 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.) Szintézis és implementáció!:

- **Szintézis:** „logikai szintézis” során a HDL leírás 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 ISE üzenet ablakában megjelenített információk alapján láthatjuk - és csak utána tudjuk a további implementációs lépéseket végrehajtani.

A tervezés fontosabb lépései (III.):

Implementációs fázisok (részletezve):

- **TRANSLATE:** több, esetleg eltérő hardver leíró nyelven megírt tervezői file összerendelése (merge) egyetlen *netlist*-fájlba (*EDIF*). A netlista fájl tartalmazza a komponensek és összeköttetések szabványos szöveges leírását.
- **MAP:** az elkészült „*logikai*” tervnek egy adott technológia szerinti „*leképezése*” (*technology mapping*), amelyet az előző lépésben kapott EDIF netlistából a kapukból 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 időzíítési analízis:** időzíítési 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).



HLS – HIGH LEVEL SYNTHESIS

4. MAGAS-SZINTŰ SZINTÉZIS

HLS elméleti háttere

■ ***Turing-Church tézis:*** HW – SW ekvivalencia.

- a μ -rekurzív függvények (utasítások),
 - környezet független nyelvtan (algoritmikus 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 célja

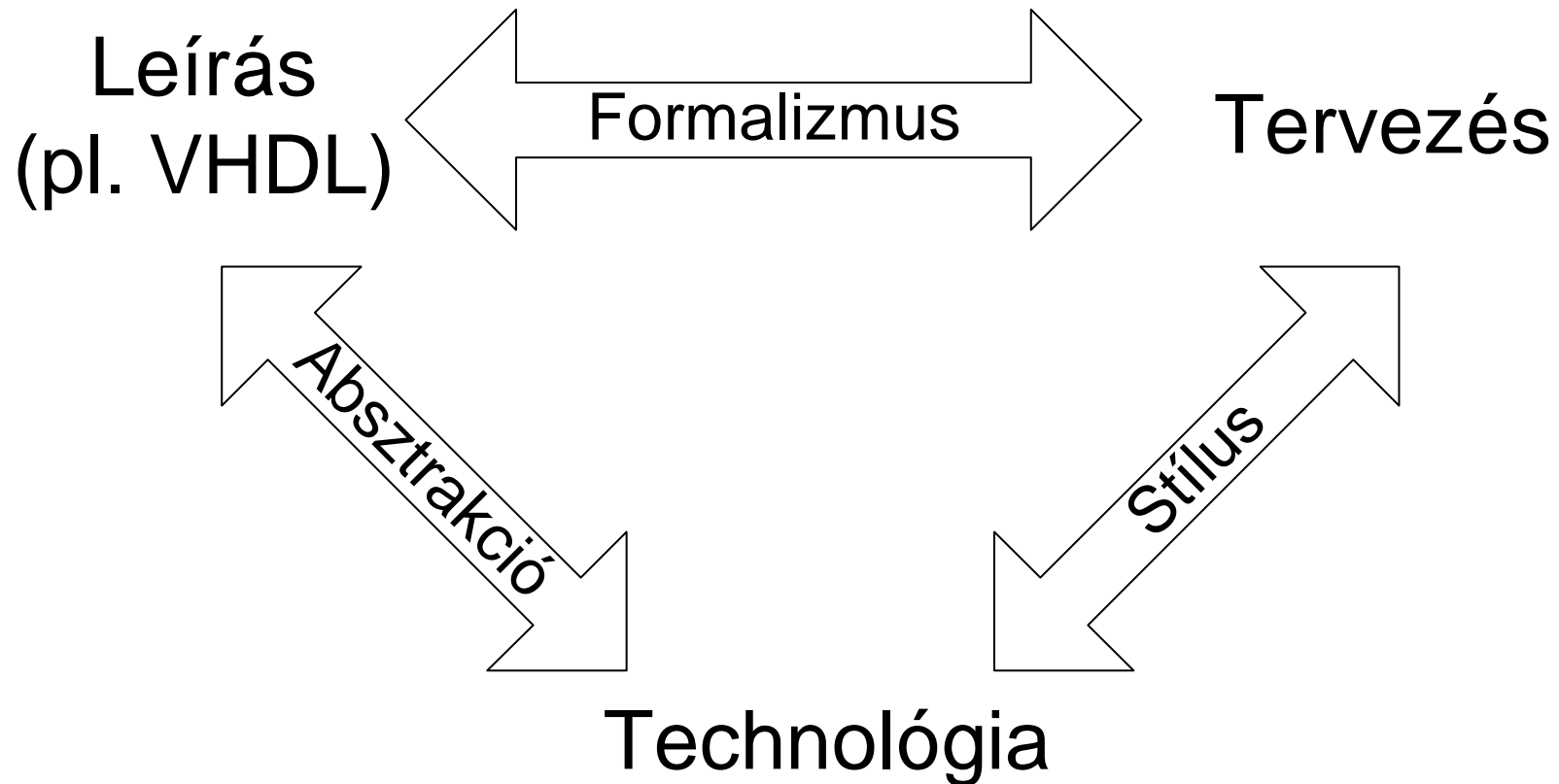
- tervezés *automatizálásának* vizsgálata kombináció és, szinkron digitális hálózatok esetén, méghozzá a logikai (Boole-algebrai) szint feletti hierarchia szinten definiált gyakorlati környezetben
 - Rendszer megvalósítása gyorsan, akár komplex rendszerek szintézisére is (elfogadható időben): „rapid prototyping, time-to-market” fogalmak
 - Bemenete magas szintű hardver leíró nyelv (hagyományos HDL, pl. VHDL, vagy Verilog), amelynek segítségével a rendelkezésre álló adatbázisok (DB – tervezői alap-építőelem könyvtárak) alapján digitális áramköröket építhetünk pl. újrakonfigurálható PLD, FPGA eszközökön.
-
- „**Magas szintű szintézis**” – fő lépései a fókuszbán:
 - Tervezés-Ütemezés (SCHEDULING)
 - Erőforrás foglalás / feladat felosztás - (ALLOCATION)
 - Kötött algoritmusok (BINDING) - megkötések

HLS során használt algoritmusok

- i.) In –core modell szintézis: a bemeneti magas-szintű nyelvből **gráf-reprezentációt** állít elő, amely a vezérlést és az adatokat külön-külön kezeli. (Lásd CFG, DFG).
- ii.) 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.
- iii.) 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.
- iv.) 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 az a megfelelő FU-t az adott utasítás elvégzéséhez.
- v.) Kezdeti (internal based) **ALLOKÁCIÓ**: a teljes kezdő adatút (data-path) lévő blokkok generálása (fizikai lefoglalása) – **Place&Route**, majd optimalizálása
- vi.) Késleltetés és méret becslése: logikai minimalizálás (vezérlőjelek számának csökkentése, ha lehetséges), ill. kimenetek generálása

Stílus – Absztrakció – Formalizmus

■ Erős korreláció



1. Stílus

- Komplex feladat \Rightarrow egyszerűbb, kezelhető részfeladatokra bontása

- ☐ Szisztematikus
- ☐ Érthető módszerek kellenek

Pl: programozási stílusok (Top-down, Strukturált modellek, stb.)

- Jó stílus kialakításának szabályai:

- ☐ „*Top-down*”/ „*Bottom-up*” módszerek szerinti tervezés
- ☐ Csak kiforrott, biztos technikákat szabad alkalmazni
- ☐ Fontos a dokumentálás (specifikáció)!

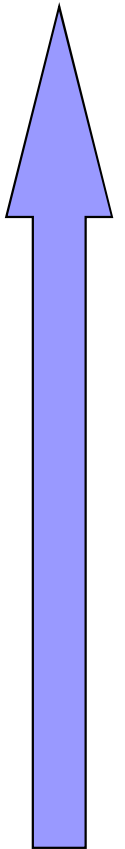
Stílushoz soroljuk a tervezés menetét (design-flow)

2. Absztrakció

- Digitális tervezés „elvi-fogalmi” szintje
 - Kezdeti absztrakció a tervezés során meghatározó, kritikus pont!
 - i. koncepcionális modell (elvi elgondolás) vs.
 - ii. 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: tranzisztor szint (legalsó szint)
 - erősen gyártás-technológia függő – (pl. CMOS-VLSI, deep submicron technology)



3. Formalizmus

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

- (felsőbb-, rendszer-szint) (alsóbb-, áramköri szint)
Absztrakció \Leftrightarrow Boole algebra

(konkretizálás)

Nyelvi eszközök



5. HARDVER LEÍRÓ NYELVEK ÉS SW FEJLESZTŐ ESZKÖZÖK

„FPGA-k lehetséges programozási nyelvei”:

■ I.) Hagyományos HDL nyelvek:

- ☐ a.) *VHDL*,
- ☐ Verilog

■ II.) C-alapú nyelvek ($C \rightarrow$ FPGA szintézis):

- ☐ a.) Xilinx Vivado HLS (*C/C++*, *System C*, *OpenCV*)
- ☐ b.) Altera SDK (OpenCL)
- ☐ c.) Impulse-C,
- ☐ d.) Catapult-C, (régen Mentor Graphics, ma Calypto)
- ☐ Handel-C, System-C, Mitrion-C, stb. (még ~10 további)

■ III) Modell alapú nyelvek:

- ☐ a.) Matlab Simulink,
- ☐ b.) NI LabView (FPGA Module)

Hagyományos HDL nyelvek célja I.

- Hardver modellezés
- A nyelv elemkészlet jelentős része
 - csak a hardver funkciók modellezésére ill. PC-n történő szimulációra használható, ill.
- A nyelv elemkészlet bővebb tartománya
 - Szintézisre és implementációra is használható (FPGA-n futtatható), azonban
 - Szintetizálható részalmaz szintézis eszköz (synthesizer, pl XST) függő
- Kapusintű 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
 - Be-, kimenetek, illetve kétirányú port-ok definiálása
 - Be-, kimenetek közötti logikai kapcsolatok és időzítések definiálása
- NEM szoftver!!
- Alapvetően **időben párhuzamos, konkurens** működést ír le (gondoljunk az FPGA nagyszámú, párhuzamosan elérhető blokkjainak felépítésére)
 - Ha külön definiáljuk (process, always), akkor fogja csak szekvenciálisan végrehajtani az utasításokat!

FPGA-k, 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 terjedésének egyik fő korlátja.



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

- + Gyorsabb prototípus fejlesztés, szimuláció/verifikáció
- + HW(FW) / SW együttes tervezés, szimuláció és verifikáció

Tradicionális HDL nyelvek

Szabványos HDL (Hardware Description Language)

VHDL

- 1983-85: IBM, Texas Instruments
- 1987: IEEE szabvány (IEEE 1076-1987)
- 1994: VHDL-1993

Verilog

- 1984: Gateway Design Automation Inc.
- 1995: IEEE szabvány (IEEE 1364)
- 2001: Verilog-2001

I. a.) VHDL hardver leíró nyelv

- **VHDL = VHSIC HDL** (Very High Speed Integrated Circuit Hardware Description Language) nevű nyelv a DARPA program keretében a 80-as évek közepén alakult ki, melyet azóta nemzetközi szabványnak is elfogadtak (IEEE-1076 1987ben).
 - Nagyon-nagy sebességű Integrált Áramkörök - Hardver Leíró Nyelve,
 - A VHDL szintaxisa az ADA programnyelven alapul: erősen típusos, jól strukturált, objektum-orientált magas-szintű „programnyelv”,
 - elsődleges célja volt: a magasabb absztrakciós szinteken az ember számára is áttekinthetővé tenni egy hardver dokumentációját,
 - biztosítja az egyes rétegnek megfelelően egy szabványos „nyelven” a hardver leírását,
 - az alsó szinteken lehetővé teszi a tervek átadását a gyártás CAD/CAM rendszereinek,
 - 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 (viselkedése) és szerkezete
 - *egyed* (entity): hw egységek közötti kommunikációs interfész (protokoll)
 - *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
```

Altera vs. Xilinx fejlesztő környezetek

■ Altera

- Quartus II: integrált fejlesztő környezet (HW/FW fejlesztés) *HDL*-ből (OpenCL is – 2013)
- QSys: modularizált, hierarchikus beágyazott rendszer fejlesztés
- EDS: beágyazott SW fejlesztés (Eclipse)
- Modelsim: szimulációs környezet

■ Xilinx

- ISE Design Suite: integrált fejlesztő környezet (HW/FW) *HDL*-ből (C/C++ Vivado HLS -2014.1)
- EDK: beágyazott rendszer fejlesztő környezet
- SDK: beágyazott SW fejlesztő körny. (Eclipse)
- ModelSim, vagy ISim: szimulációs környezetek

CFG: Control Flow Graph

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

DFG: Data Flow Graph

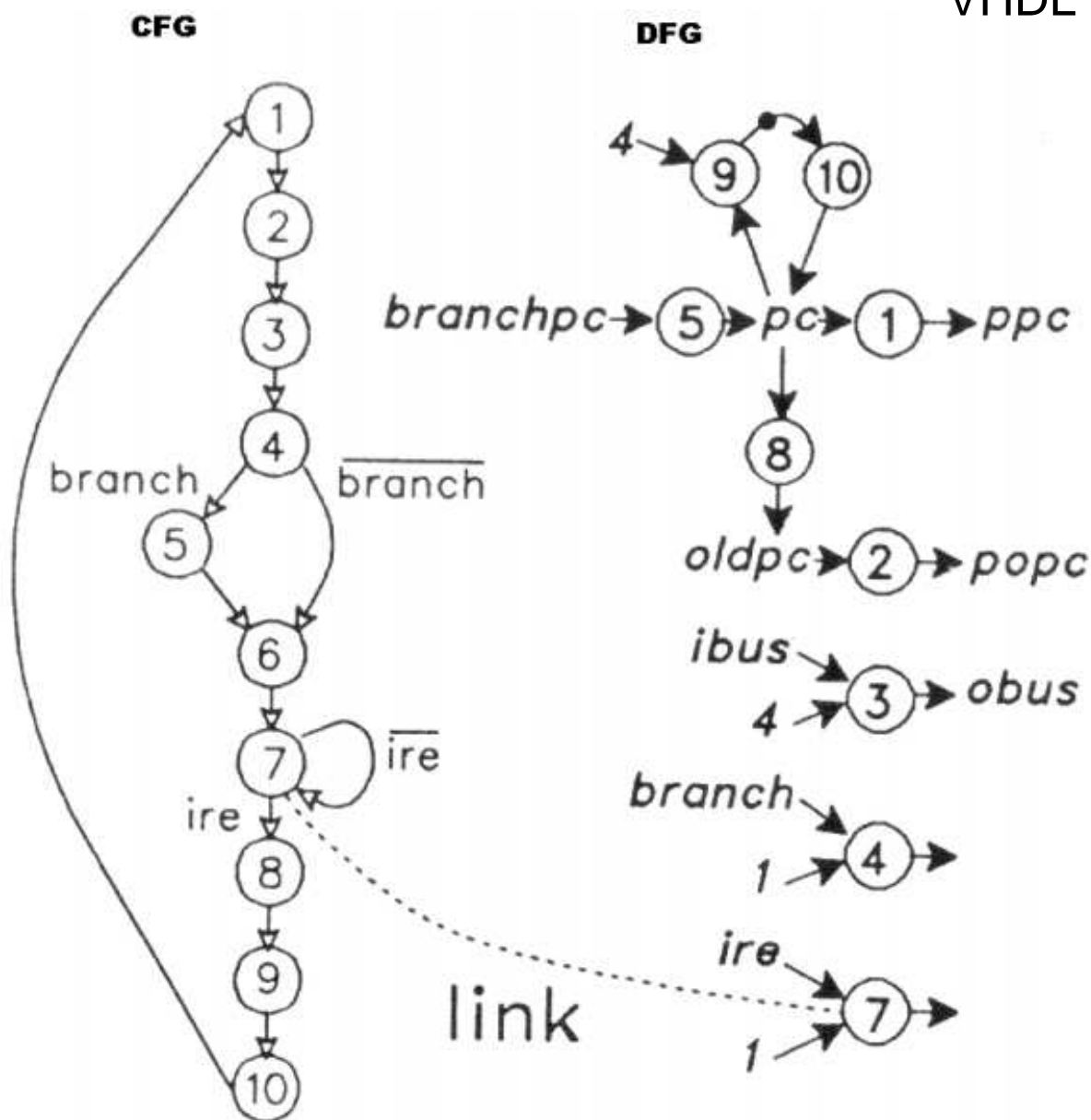
- Irányított gráf, $DFG=(N \cup V,D)$ Adat-folyam gráf
- Csúcsok halmaza:
 - **N: utasítások**
 - **V: változók**
- Élek halmaza (D):
 - Adatkapcsolatok
- Fontos: nem feltétlenül összefüggő gráf, azaz a DFG-nél az egyes műveletek lehetnek ábrázolva úgy is, hogy nincs közöttük kapcsolódási pont (nem húzódik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

Példa

Csak az
hozzárendelési utasítások
megszámozása (begin-end részen)

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;

 for i in 0 to 3 loop

vsum(i) := (a(i) xor b(i)) xor carry;

carry := (a(i) and b(i)) or (carry and (a(i) or b(i)));

 end loop;

sum <= vsum;

cout <= carry;

 end process p1;

end behavioral;

Csak az
Utasítások
megszámozása!



--1

--2

--3

--4

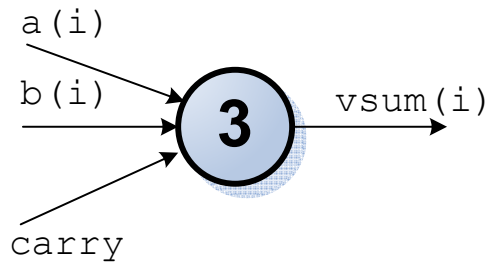
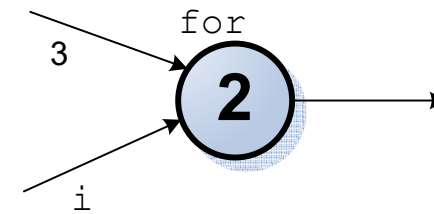
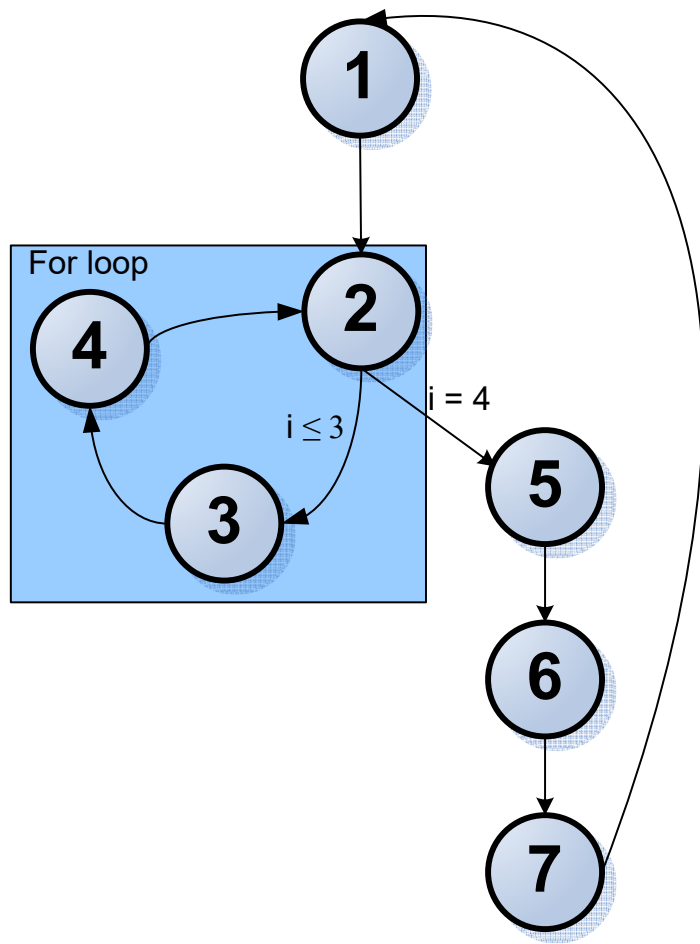
--5

--6

--7

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

VHDL CFG



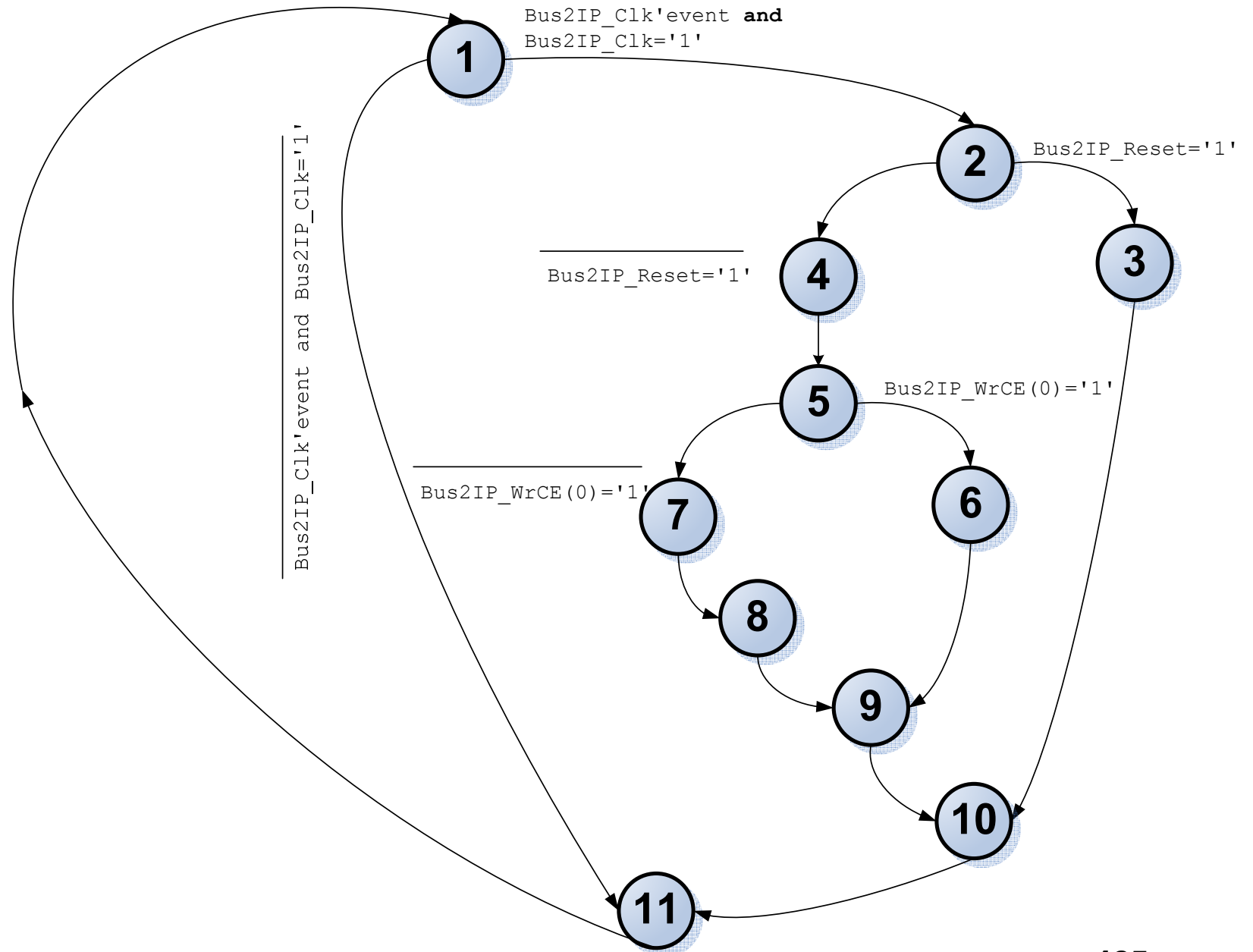
Példa 3): VHDL -> CFG

```
-- 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
      if Bus2IP_Reset='1' then
        LED_i<="0000";
      else
        if Bus2IP_WrCE(0)='1' then
          LED_i<=Bus2IP_Data(0 to 3);
        else
          LED_i<=LED_i;
        end if;
      end if;
    end if;
  end process LED_PROC;
end behavioral;
```

Csak az
Utasítások
megszámozása!

--1
--2
--3
--4
--5
--6
--7
--8
--9
--10
--11

VHDL CFG



Altera vs. Xilinx fejlesztő környezetek

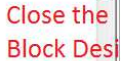
■ Altera

- Quartus II: integrált fejlesztő környezet (HW/FW fejlesztés) *HDL*-ből
 - Qsys: modularizált, hierarchikus beágyazott rendszer fejlesztés
 - EDS: beágyazott SW fejlesztés (Eclipse)
 - NiosII EDS: soft core, EDS-SOC: ARM hard-core
- Altera SDK OpenCL (2013)
- Modelsim: professzionális HDL szimulációs környezet

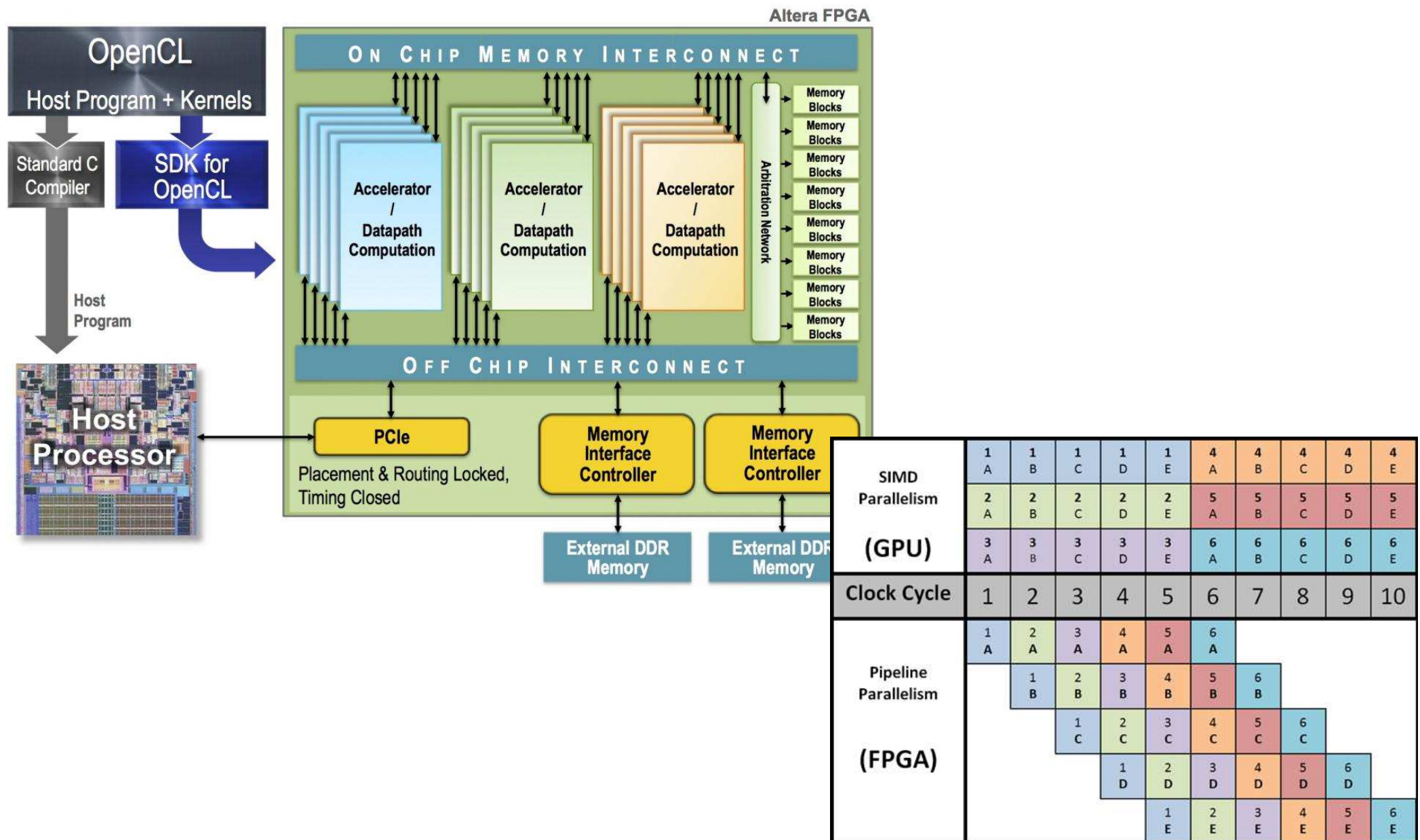
■ Xilinx

- Vivado: HDL alapú környezet, Vivado HLS: C/C++
- ISE Design Suite: integrált fejlesztő környezet (HW/FW) *HDL*-ből (C/C++ Vivado HLS (2013))
 - EDK: beágyazott rendszer fejlesztő környezet
 - SDK: beágyazott SW fejlesztő körny. (Eclipse)
- ModelSim, vagy ISim: szimulációs környezetek

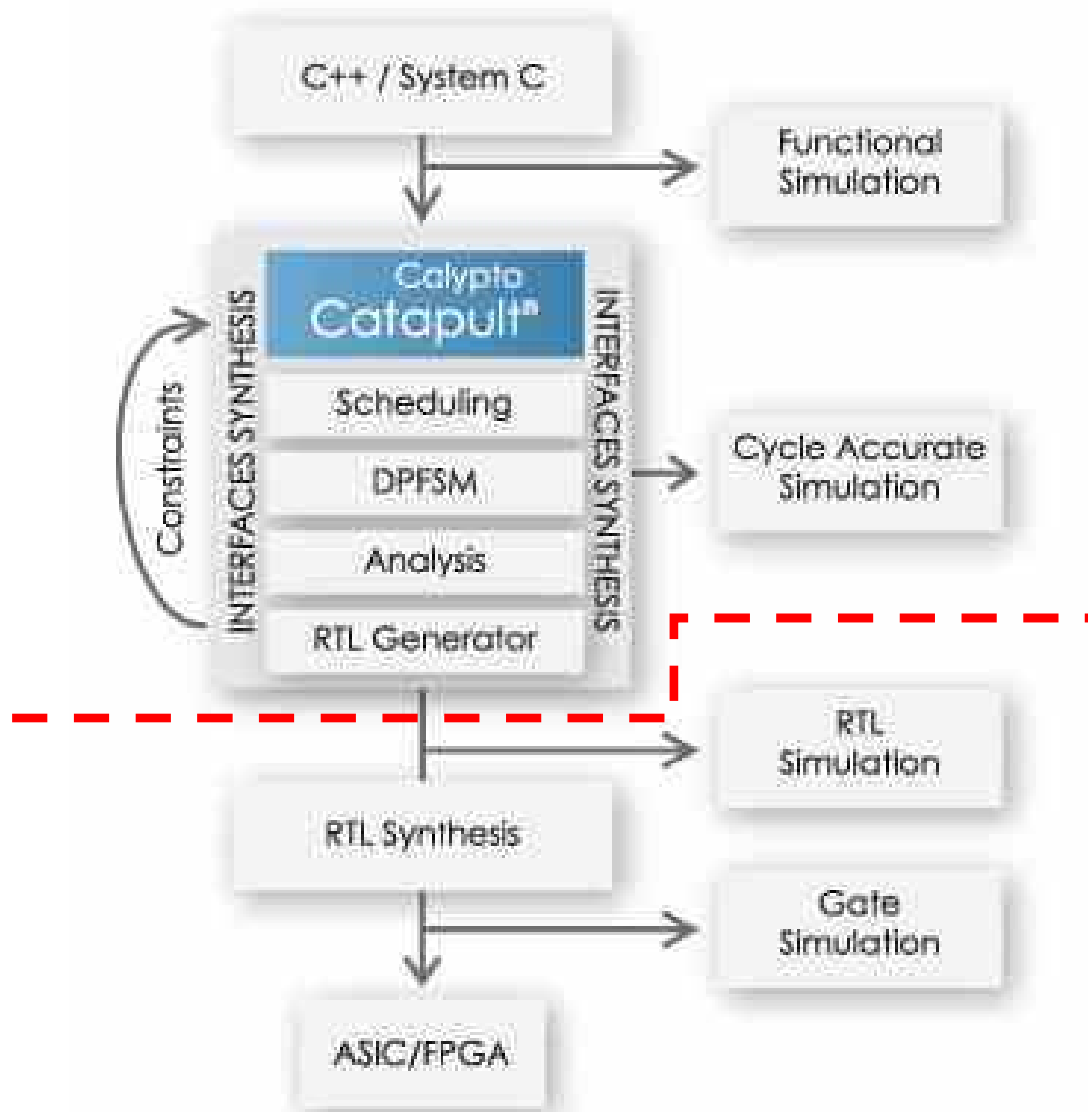
A decorative graphic consisting of a 3x3 grid of squares. The top-left square is dark blue, the top-middle square is light blue, and the top-right square is white. The middle-left square is light blue, the middle-middle square is white, and the middle-right square is light blue. The bottom-left square is white, the bottom-middle square is light blue, and the bottom-right square is white.



I. b. Altera SDK for OpenCL



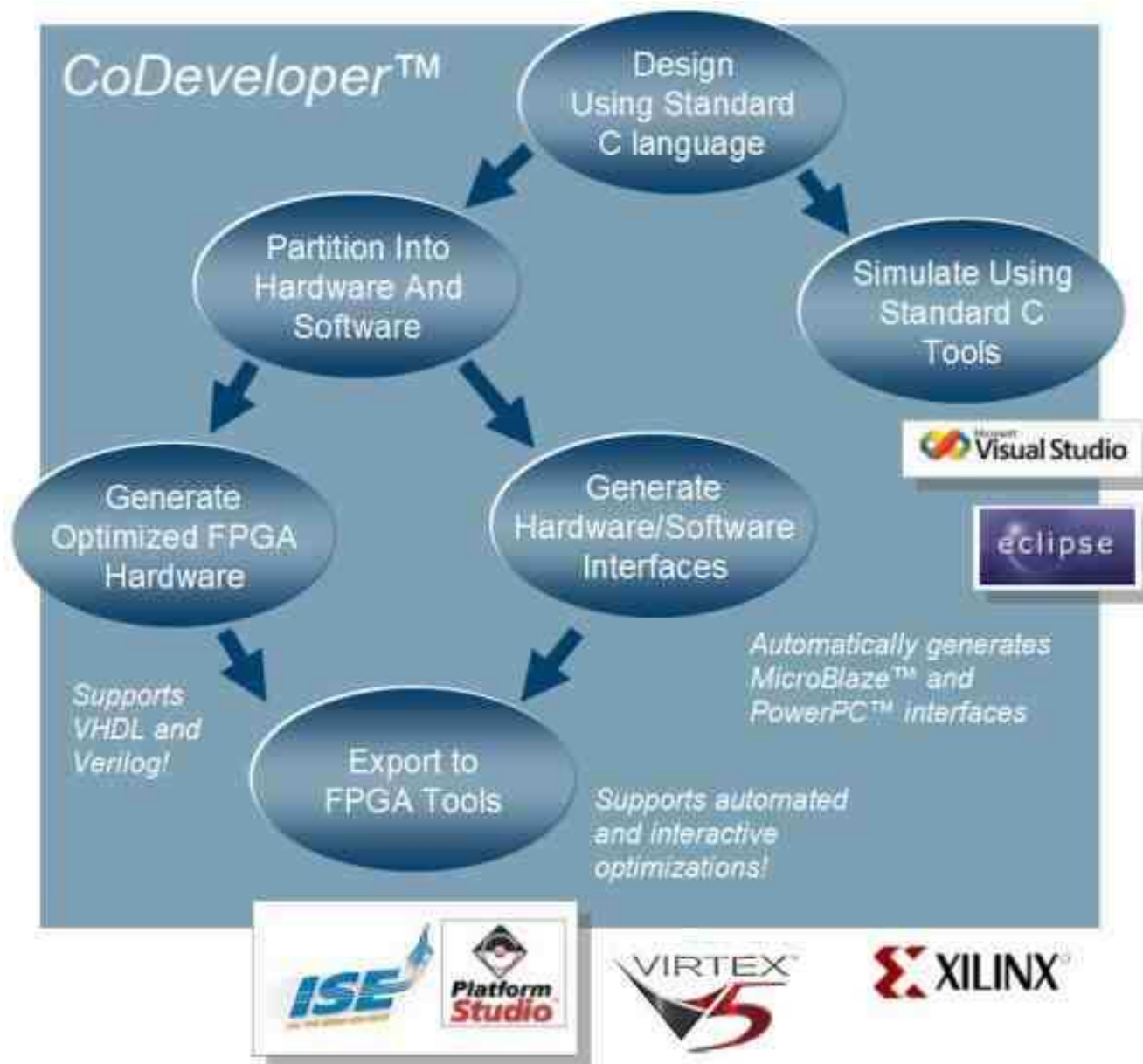
II. a. Calypto Catapult-C



Tervezés folyamata
Xilinx FPGA-kra:

- Bemenet: C++, vagy System C
- Kimenet: RTL szintű HDL leírás

II. b.) ImpulseC



Tervezés folyamata
Xilinx FPGA-kra:

- Bemenet: ImpulseC
- Kimenet: RTL szintű leírás (HDL)

II. c.) Handel-C

■ Par: Parallelismus!

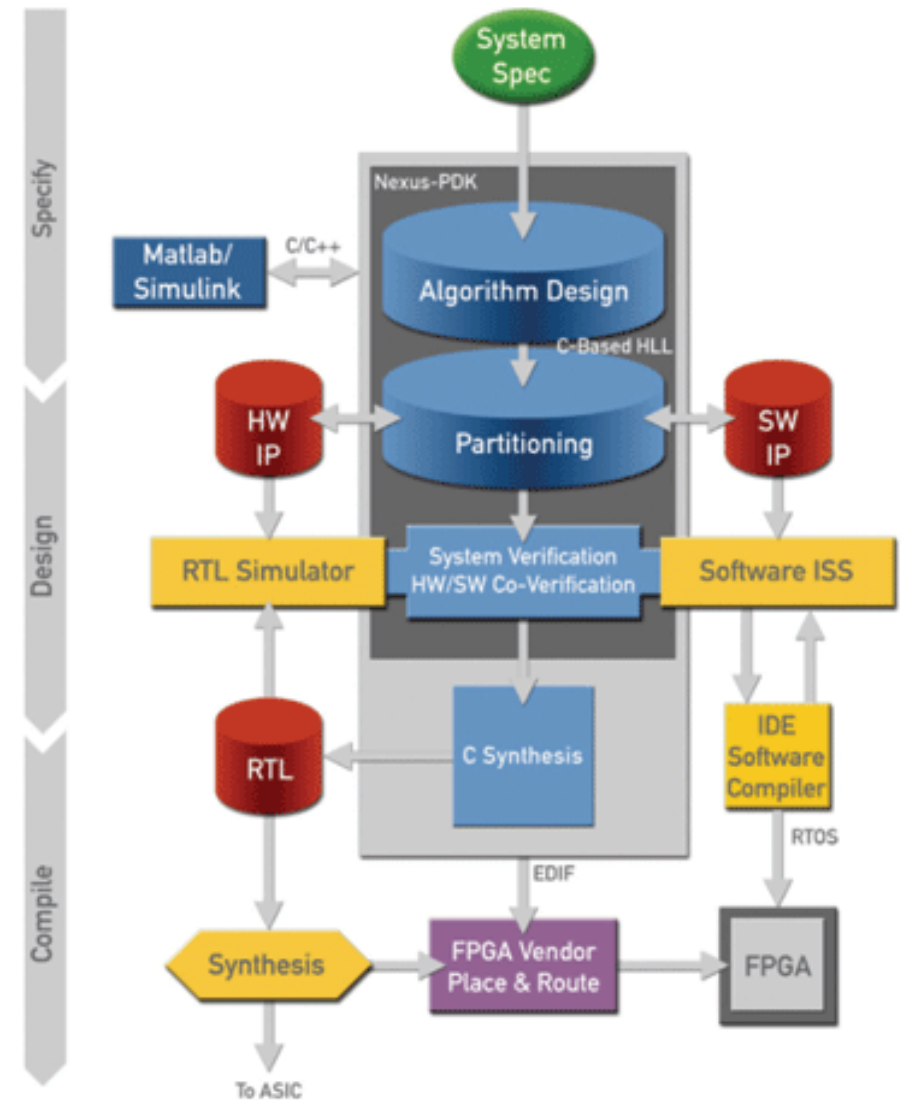
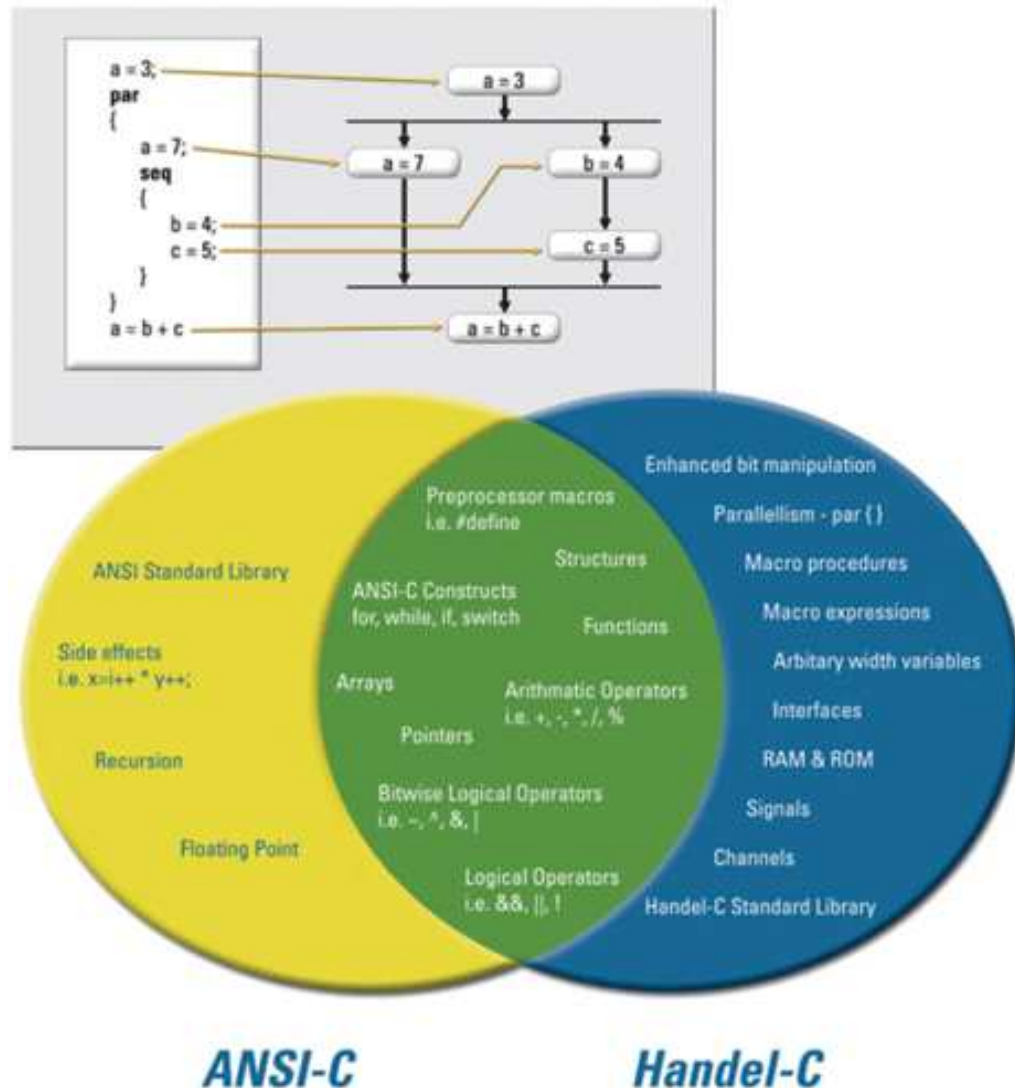
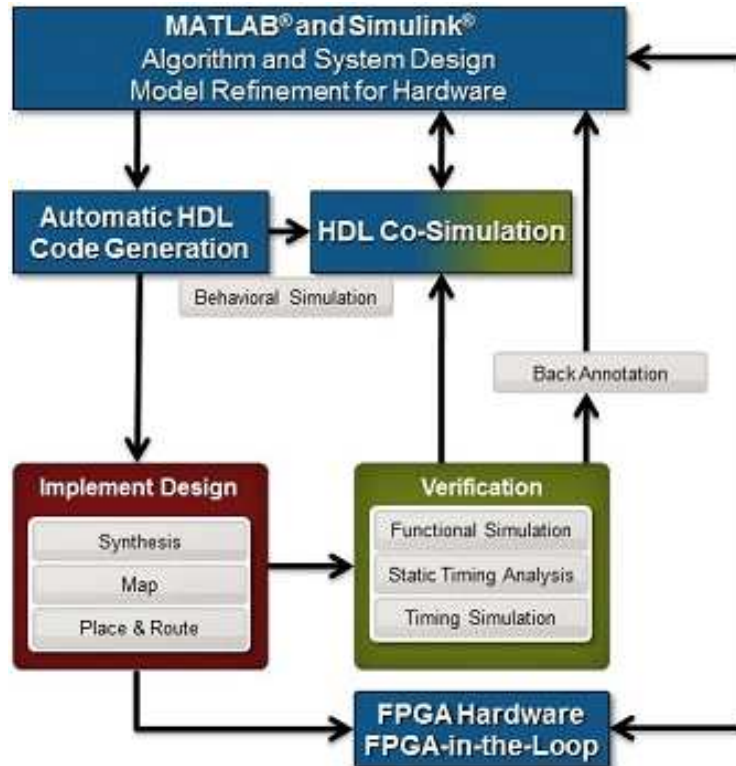
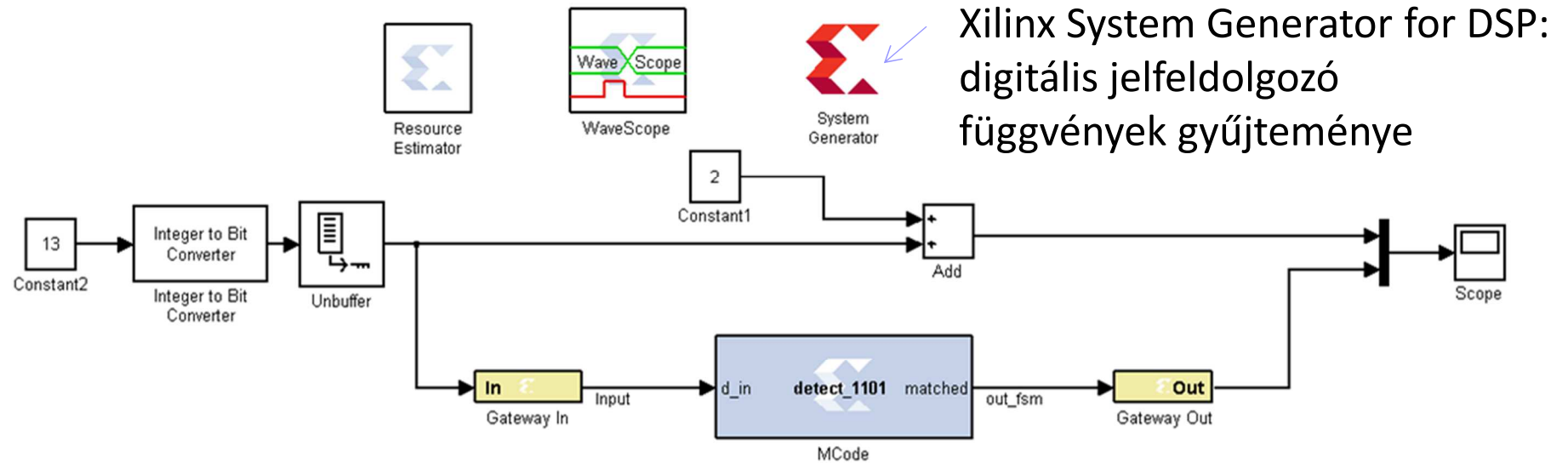


Figure 4 Design flows from C to FPGA combine various system models to produce a verified representation of the embedded algorithm that can then be directly synthesized to FPGA. Celoxica calls this process Software-Compiled System Design.

III. a.) Matlab - Simulink



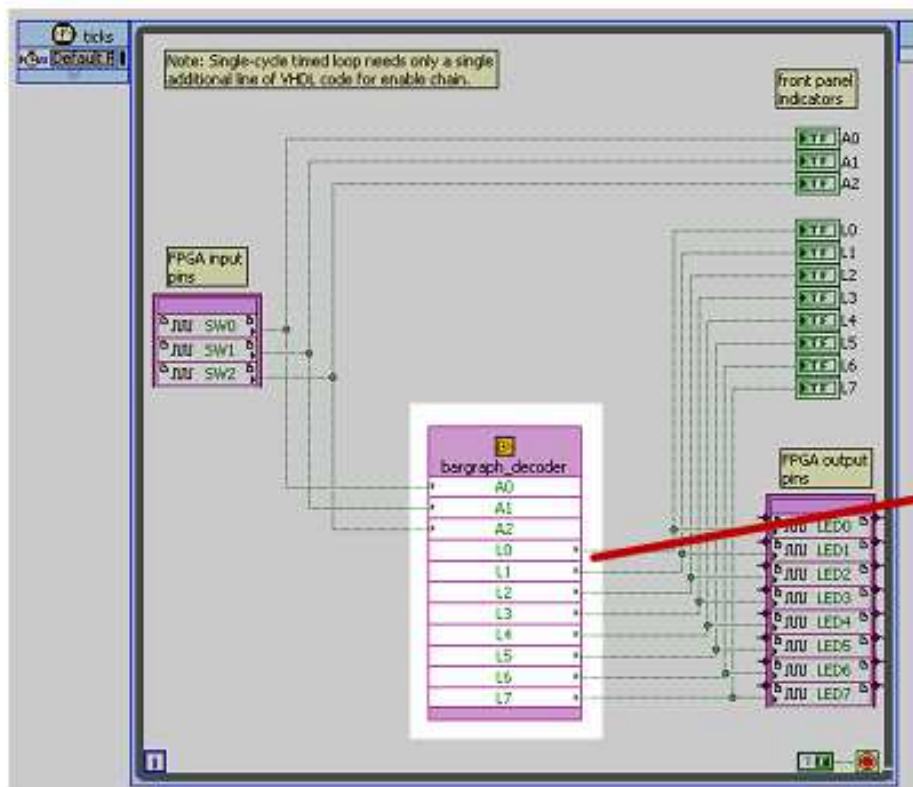
Mcode: Matlab-Code (.m nyelven írható egyedi modul, amely beépíthető akár a Simulink blokkok közé

III. b.) LabView – FPGA modul



VHDL node

3->8 dekóder



HDL Interface Node Properties

Category: Parameters, Code, VI Debugging, External Files, Execution Control, Simulation

Code

Libraries: library ieee; use ieee.std_logic_1164.all;

entity bargraph_decoder is

generic(
ClockFrequency : Integer := 50000000;
InSingleCycle : boolean := true
);
end bargraph_decoder;

architecture implementation of bargraph_decoder is

begin

```
PROCESS(A0, A1, A2)
BEGIN
L0 <= "1"; -- Use vector-style constant to match LabVIEW defn
L1 <= A2 OR A1 OR A0;
L2 <= A2 OR A1;
L3 <= A2 OR (A1 AND A0);
L4 <= A2;
L5 <= (A2 AND A1) OR (A2 AND A0);
L6 <= A2 AND A1;
```

end implementation;

Check syntax

OK Cancel Help

További ajánlott irodalom

- Fontosabb FPGA gyártók oldalai:
 - <http://www.xilinx.com>
 - <http://www.altera.com>
 - <http://www.microsemi.com>
- FPGA és Programmable Logic Journal:
 - <http://www.fpgajournal.com>
- Xilinx FPGA Silicon Devices:
 - <http://www.xilinx.com/products/devices.htm>
- Fodor Attila, Dr. Vörösházi Zsolt: Beágyazott rendszerek és programozható logikai alkatrészek (TÁMOP 4.1.2) Egyetemi jegyzet (2011)
 - http://www.tankonyvtar.hu/hu/tartalom/tamop425/0008_fodorvoroshazi/Fodor_Voroshazi_Beagy_0903.pdf
- **Témához kapcsolódó tárgyak a Pannon Egyetemen:**
 - **Tervezési módszerek programozható logikai alkatrészekkel (VHDL)**
 - <http://virt.uni-pannon.hu/index.php/oktatas/tantargyak/770-tervezesi-modszerek-programozhato-logikai-alkatreszekkel-vhdl>
 - **FPGA-alapú beágyazott rendszerek**
 - <http://virt.uni-pannon.hu/index.php/oktatas/tantargyak/1030-fpga-alapu-beagyazott-rendszerek>