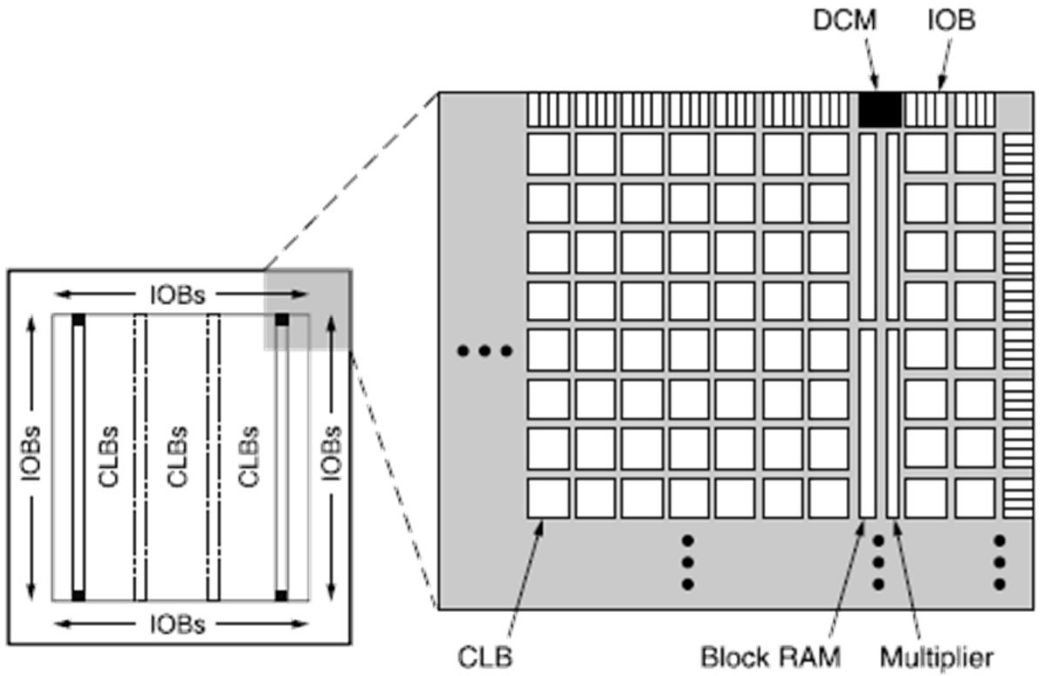
DIGITÁLIS TECHNIKA
12

Dr. Oniga István

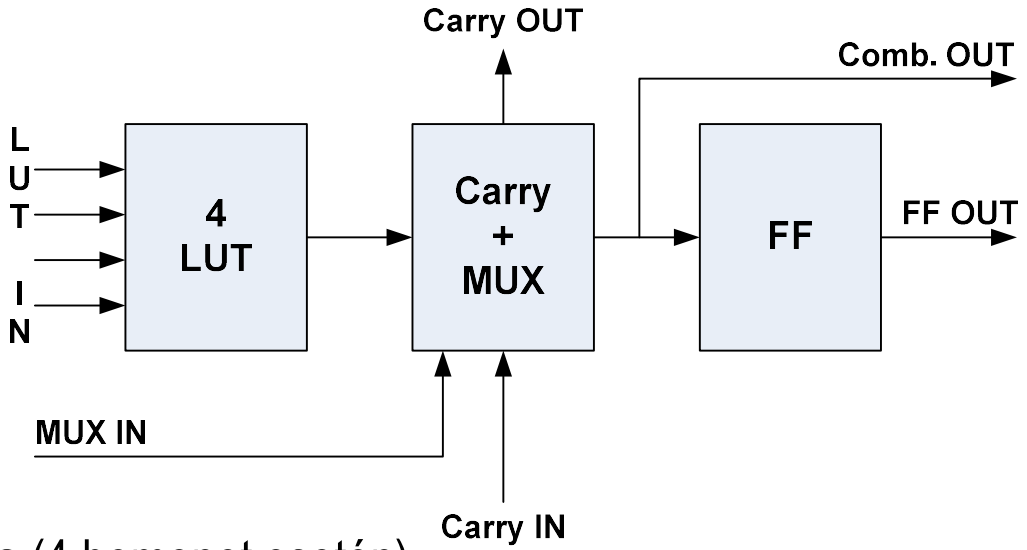
Xilinx FPGA-k

- Több család
  - Spartan: hatékony, optimalizált struktúra
  - Virtex: speciális funkciók, gyorsabb, komplexebb, gazdagabb funkcionalitás
- Felépítés:
  - CLB: logikai blokk
  - IOB: I/O blokk
  - BlokkRAM: belső memória
  - Szorzó
  - Huzalozás



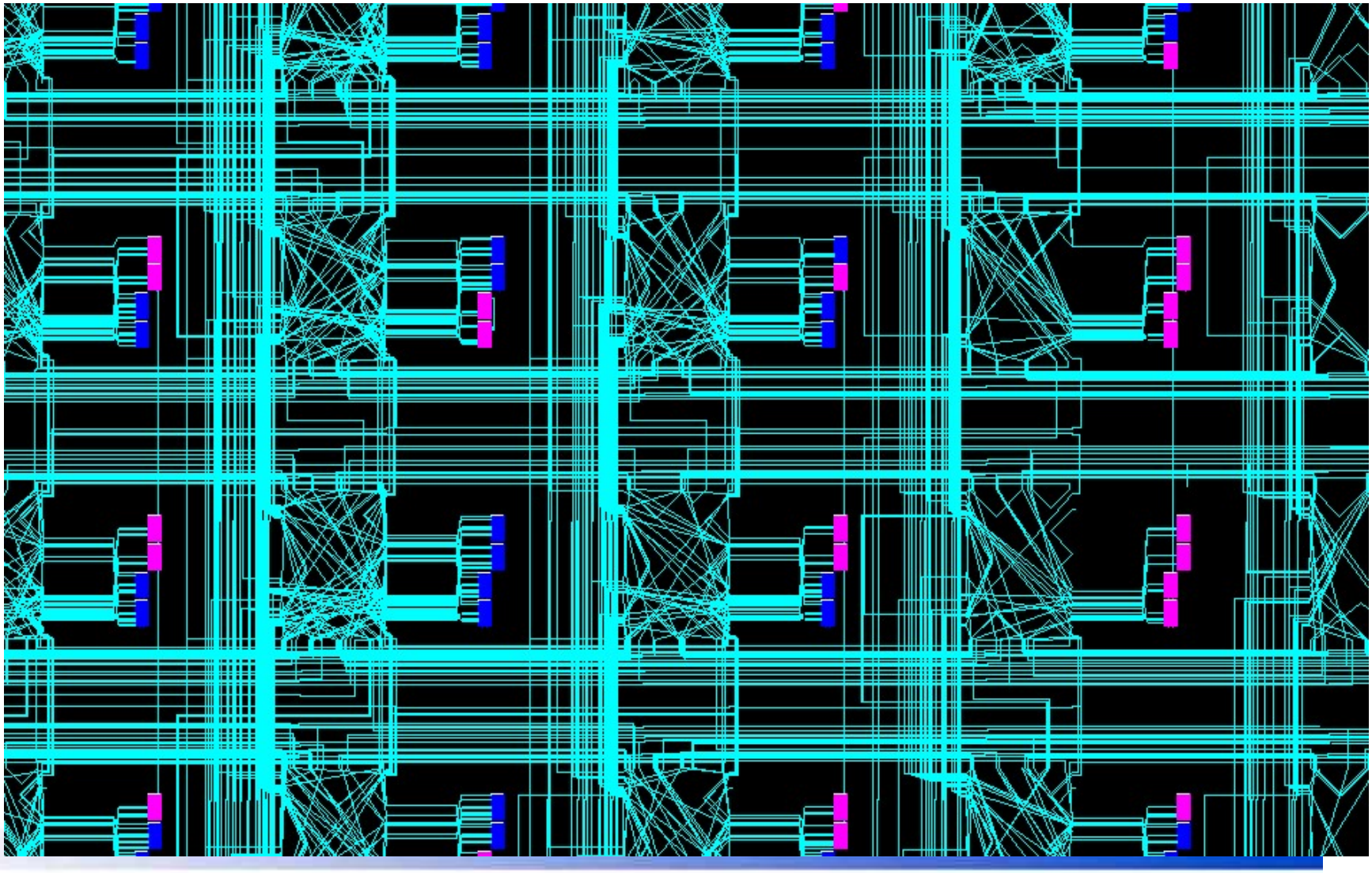
Xilinx FPGA: Alap logikai elem

- Logikai elem (Slice): 2 LUT + 2 FF + kiegészítő logika



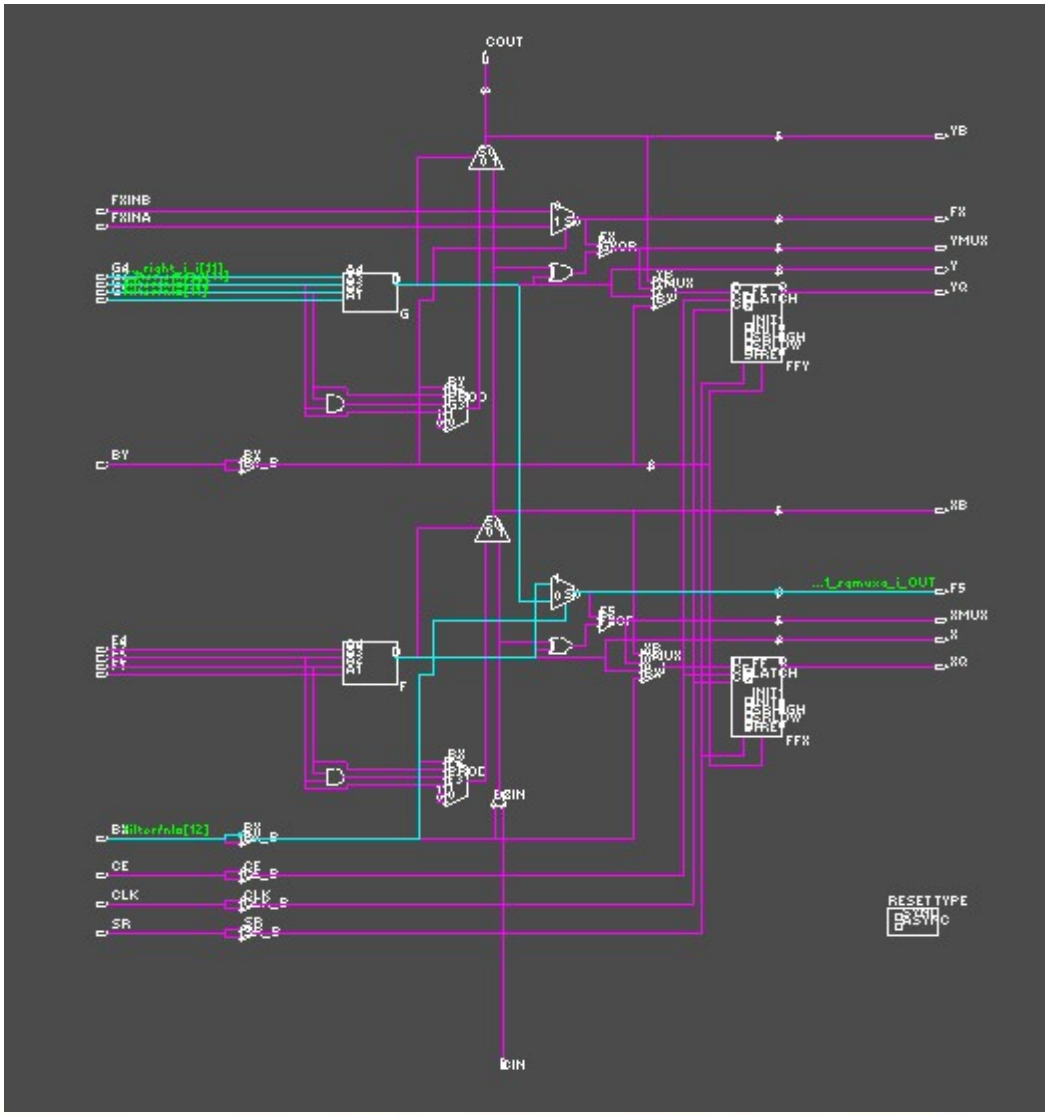
- LUT: Look-Up Table
  - 16x1 bites memória (4 bemenet esetén)
  - Cím: a logikai függvény bemeneti változói
  - Tartalom: igazságtábla
  - Bármilyen négy bemenetű, egy kimenetű logikai függvény megvalósítható

Xilinx FPGA-k



Xilinx FPGA-k: a logikai elem részletei

- **A CLB belső felépítése az FPGA Editor-ban nézve**

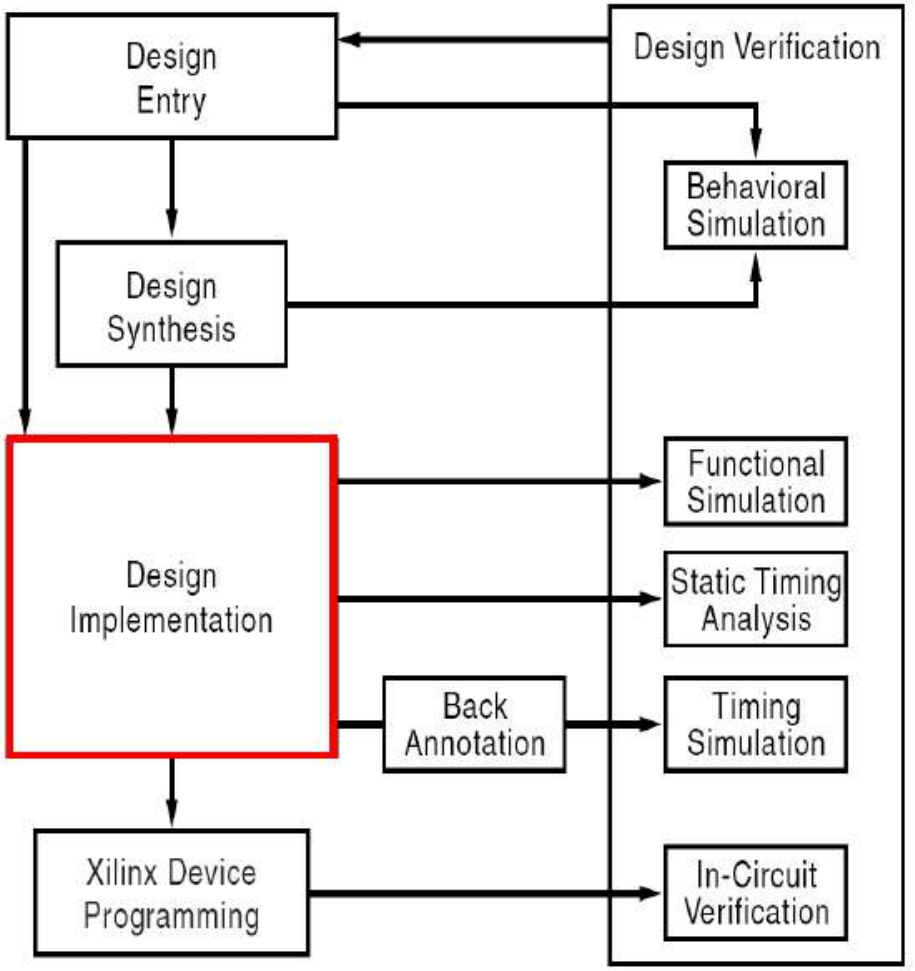


Xilinx FPGA: konfiguráció

- A konfigurációt (LUT tartalom, huzalozás, csatlakozások, egyéb paraméterek) SRAM tárolja
- Tápfeszültség kikapcsolásakor elveszíti a konfigurációt
- Bekapcsolás után konfiguráció szükséges
  - EEPROM-ból, automatikusan
  - Fejlesztői kábel segítségével ún. JTAG porton keresztül

Fejlesztő rendszerek

- Terv leírás: (Design Entry)
  - Xilinx Foundation ISE
  - Külső eszköz
    - Mentor Graphics: FPGA Advantage
    - Celoxica: DK Design Suite
- Szintézis terv: (Design Synthesis)
  - XST: Xilinx Synthesis Technology
  - Mentor: Leonardo Spectrum
  - Synplicity: Synplify Pro
  - Celoxica: DK Design Suite
- Szimuláció:
  - Mentor: Modelsim
  - Aldec: Active-HDL
  - Celoxica: DK Design Suite
- In Circuit verifikáció:
  - Xilinx: ChipScope



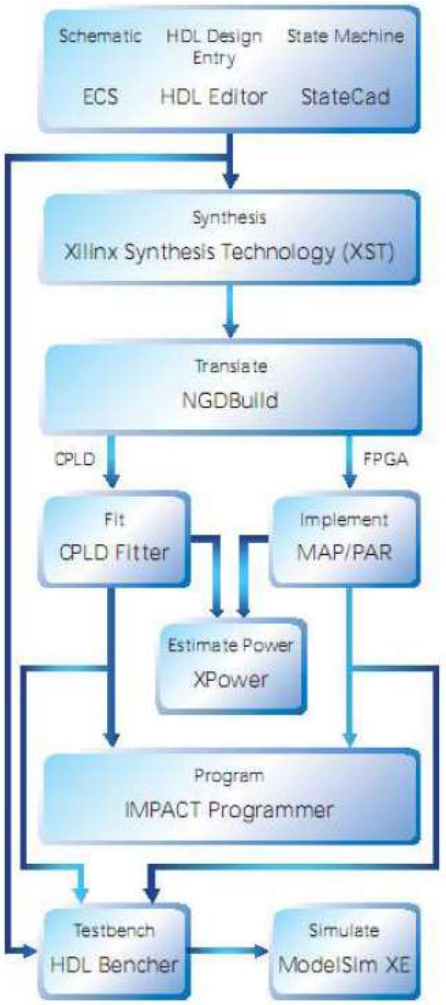
Az ISE rendszer részei

- Xilinx ISE - Integrated Software Environment – integrált szoftverkörnyezet
  - a Xilinx cég FPGA-ihoz és CPLD-ihez kifejlesztett szoftver.

- ***ISE WebPack -*** ISE rendszer egyszerűbb, de funkcionálisan komplett változata
  - csak a Xilinx cég IC-ivel való implementálást támogatja ,
  - nem támogatja az összes család összes IC-jét, hanem tipikusan csak a kisebb komplexitásúakat,
  - ingyenes szoftver, szabadon letölthető.

Fejlesztés folyamata

- Project Navigator szoftver, az ISE keretprogram
  - **Rendszertervezés** – (Terv leírás + tervezési megkötések – constraints)
  - RTL szimuláció - Tesztkörnyezet (Testbench)
  - **Szintézis**
  - **Implementáció**: TRANSLATE →MAP →PAR (place & route)
  - Statikus időzítési analizis: timing parameters meghatározása (max clock frequency, propagation delays etc.)
  - **Bitstream generálása és letöltése** (konfigurációs file - .bit)



1. ábra Tervezési folyamat a WebPACK rendszerrel

Terv leírás

- A tervező az elképzeléseit, terveit háromféle formában viheti be a rendszerbe.
  - **Kapcsolási rajz** (Schematic) formájában, a Xilinx ECS (Engineering Capture System), a kapcsolási rajz készítő és beviteli program segítségével.
  - **Hardver leíró nyelven**. Ezt a bevitelt a HDL editor rész támogatja. A támogatott nyelvek: ABELHDL, Verilog és VHDL. A rendszer sok mintaleírást is tartalmaz, úgynevezett sablonok (template) formájában.

Terv verifikálása

- Azt ellenőrizzük, hogy a terv szerinti áramkör működése megfelel-e a feladat specifikációjának.
- A verifikálás szimulációval történik.
- A WebPACK rendszer szimulátora a Xilinx ISE Simulator.
- A modellt működtetni, "gerjeszteni" kell, - a modell bemeneteire megfelelően változó jeleket kell adni. Ez az tesztvektorok sorozatának ráadásával történik.
- A tesztvektorokat a tervező beleírhatja a HDL leírásba, mint tesztelési környezet (testbench).

Szintézis

- Xilinx Synthesis Technology (XST) alrendszer végez, amely ugyancsak az ISE része (szintézisre léteznek más programok is).
- A szintézer a HDL leírásból előállít egy minimalizált és optimalizált huzalozási listát, amely az adott FPGA primitíveket (LUT, FF), és a köztük levő kapcsolatokat tartalmazza.

Implementáció

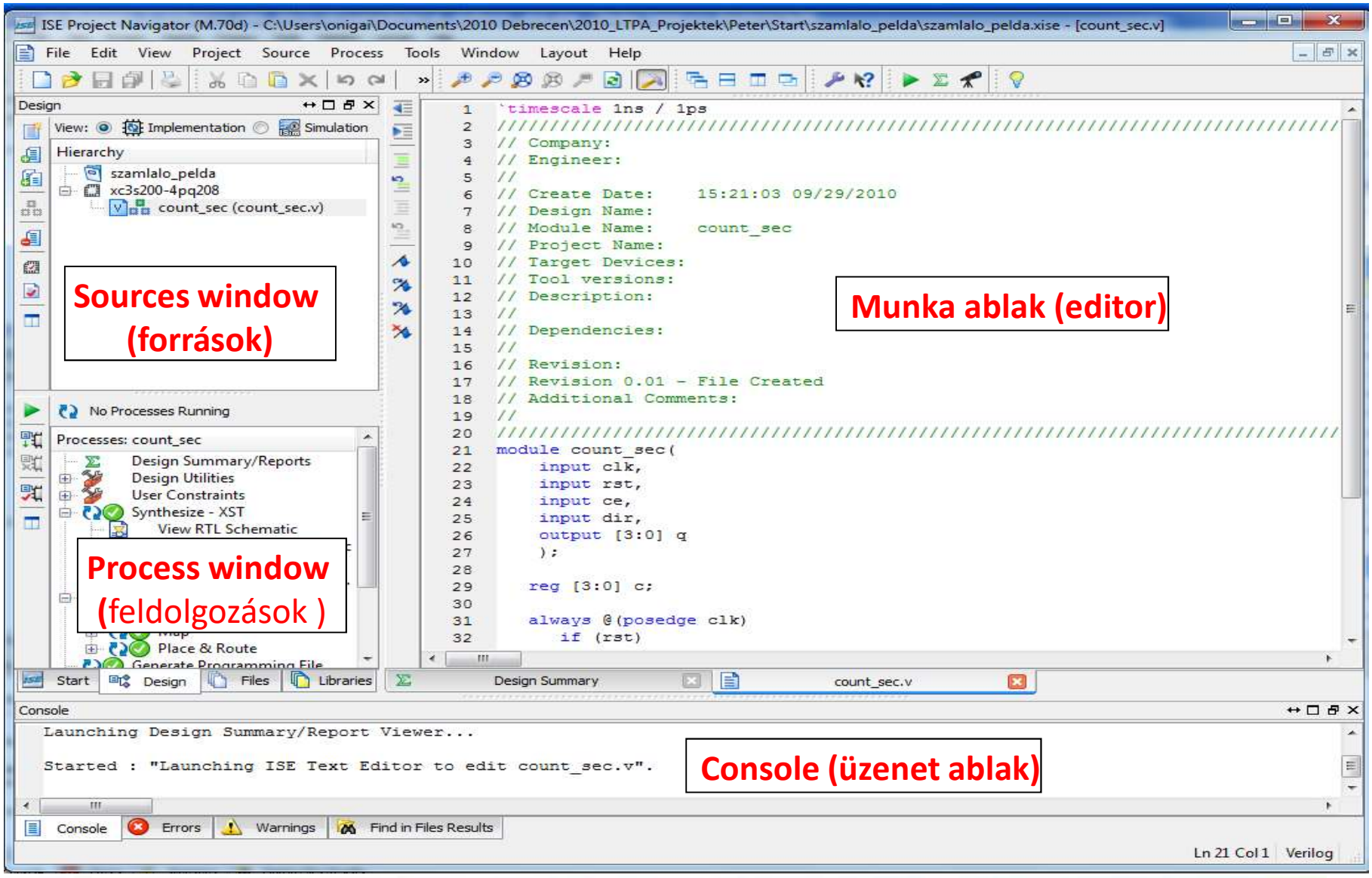
TRANSLATE → MAP →PAR (place & route)

- **TRANSLATE**: több, esetleg eltérő nyelven megírt tervezői file (HDL) összerendelése (merge) egyetlen netlist- fájlba (EDIF)
- **MAP** = technology mapping: leképzés az adott FPGA primitív-készletére (a kapukat CLB-é, ill. IOB-á „képezi le”)
- **PAR**: a végleges „fizikai” áramkört hozza létre amelyben a primitíveket fizikai helyekre rak és kialakítja a fizikai huzalozást (routing).

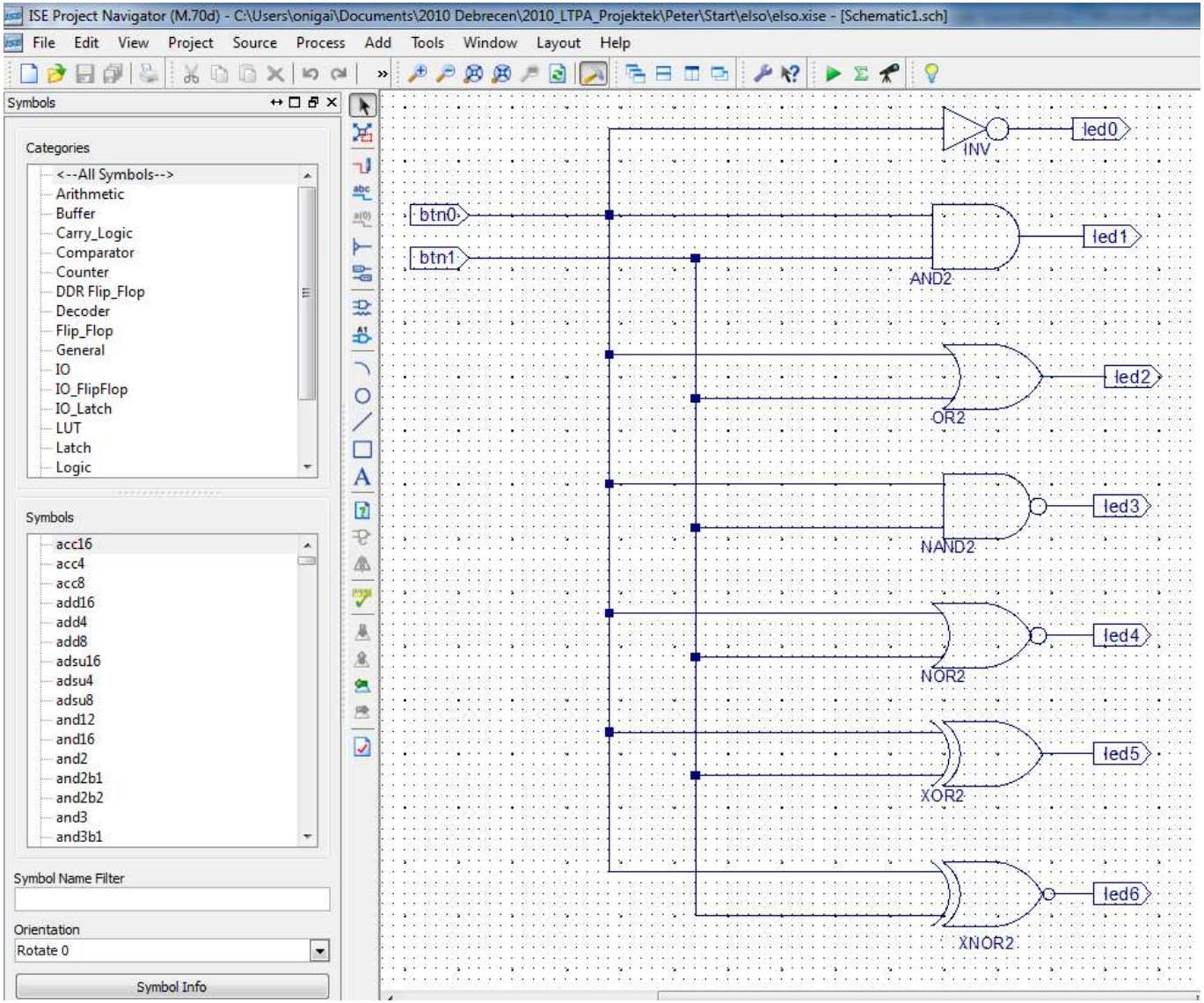
Konfiguráció

- **Bitstream -** konfigurációs file (.bit)
  - generálása
  - letöltése – soros interfészen keresztül (JTAG)
- IMPACT (vagy ADEPT) program végzi

Xilinx ISE bemutatása



Kapcsolási rajz alapú projekt

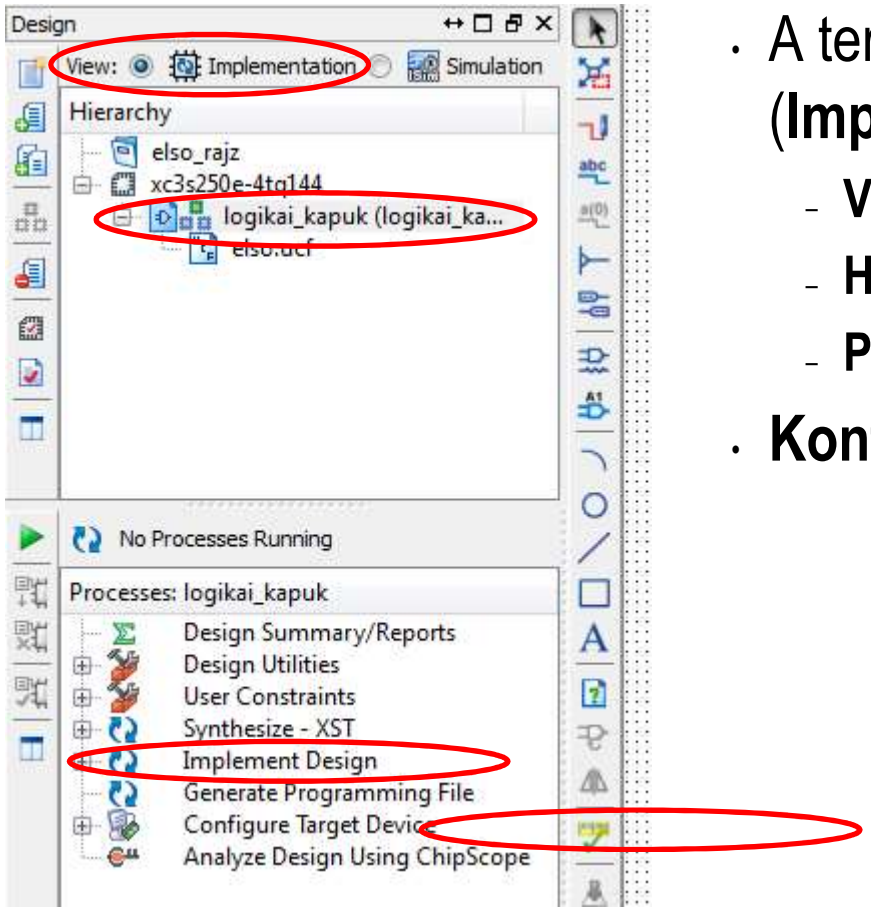


Constraints file

- A láb-hozzárendelések elvégzéséhez egy constraint fájlt adunk a projecthez.
- Válasszuk ki a **Project / New Source** menüpontot, a felbukkanó ablakban pedig álljunk az **Implementation Constraint File**-ra, névnek pedig válasszuk a *elso*-t.
- **Next/Finish** gomb megnyomása után a **Sources** ablakban meg is jelenik a *elso.ucf* fájl.
- Ha sikeresen lelestük a panelről a használt lábak nevét, az alábbihoz egészen hasonlatos *ucf* filet kapunk

```
NET "btn0"   LOC = „G12"   ;
NET "btn1"   LOC = „C11"   ;
NET "led0"   LOC = „M5"   ;
NET "led1"   LOC = „M11"   ;
NET "led2"   LOC = „P7"   ;
NET "led3"   LOC = „P6"   ;
NET "led4"   LOC = „N5"   ;
NET "led5"   LOC = „N4"   ;
NET "led6"   LOC = „P4"   ;
```

A terv implementációja



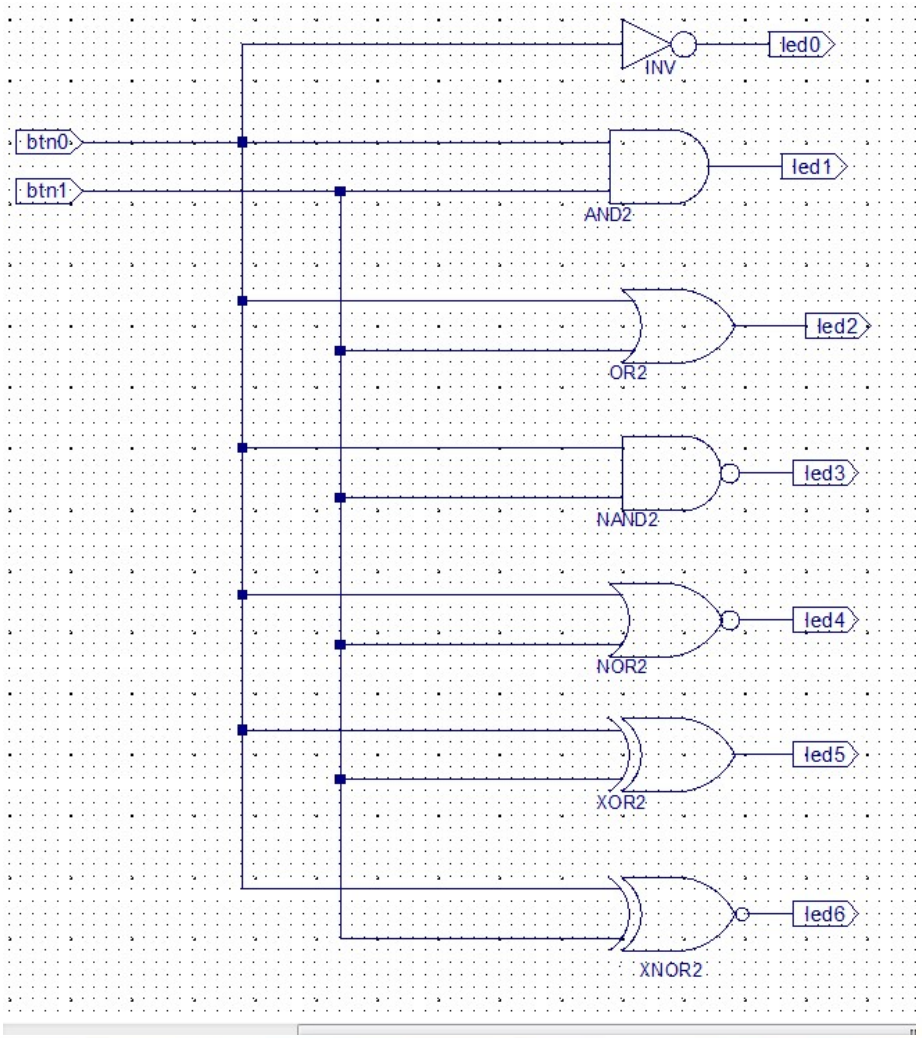
- A terv leképzése az FPGA struktúrára (**Implement Design**),
  - **View** →*implementation*
  - **Hierarchy** ablak → a top level fájl
  - **Processes** ablak → Implement Design
- **Konfigurációs bitminta létrehozása**

Az FPGA konfigurálása

- Tényleges realizálás az FPGA konfigurálásával (beprogramozásával), az előző műveletben létrehozott *.bit* konfigurációs fájlnak az FPGA-ba való letöltésével történik.

- Digilent Nexys 2 kártya:
- http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,789&Prod=NEXYS2 Digilent Adept Suite
- https://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,828&Prod=ADEPT2
- Nexys 2 reference manual
  http://www.digilentinc.com/Data/Products/NEXYS2/Nexys2_rm.pdf

Logikai kapuk implementációja HDL nyelven




```
module ElsoHDL(
    input btn0,
    input btn1,
    output led0, led1, led2, led3, led4, led5, led6
    );

assign led0 = ~ btn0;
assign led1 = btn0 & btn1;
assign led2 = btn0 | btn1;
assign led3 = ~ (btn0 & btn1);
assign led4 = ~ (btn0 | btn1);
assign led5 = btn0 ^ btn1;
assign led6 = btn0 ~^ btn1;

endmodule
```

Szabvány HDL nyelvek

- Szabványos HDL (hardware description language) nyelvek
  - Verilog
    - 1984: Gateway Design Automation Inc.
    - 1990: Cadence -> Open Verilog International
    - 1995: IEEE szabványosítás
    - 2001: Verilog 2001
    - *Verilog-2005* (IEEE Standard 1364-2005)
    - SystemVerilog (IEEE standard P1800-2005).
  - VHDL
    - 1983-85: IBM, Texas Instruments
    - 1987: IEEE szabvány
    - 1994: VHDL-1993

Egyéb HDL

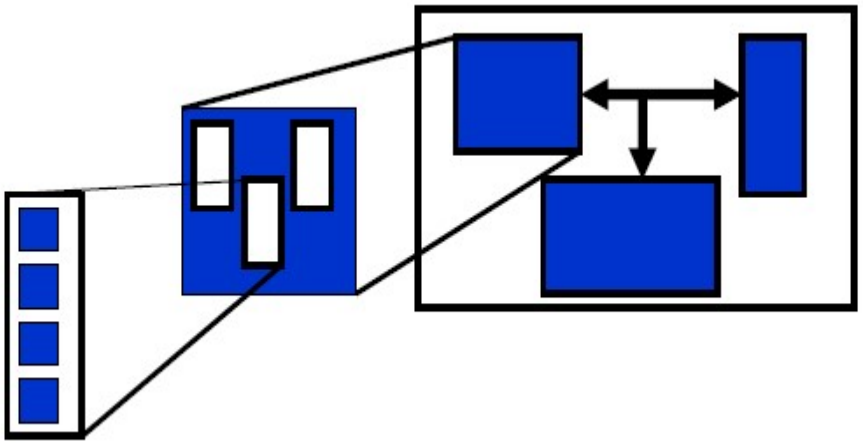
- HDL fejlesztés a szoftver fejlesztéshez viszonyítva továbbra is időigényes
- Sok fejlesztő rendelkezik C/C++ ismerettel, viszonylag kevés HDL ismerettel
- Magasszintű hardver leíró nyelvek
  - Celoxica Handel-C: C alapú, spec. kiegészítések
  - SystemC: szabványos, ma már (részben) szintetizálható, C++ alapú
  - Mentor Catapult-C: C++ kiegészítések nélkül
  - Impulse-C, Mitrion-C
- Gyorsabb szimuláció/verifikáció
- HW/SW együttes tervezés

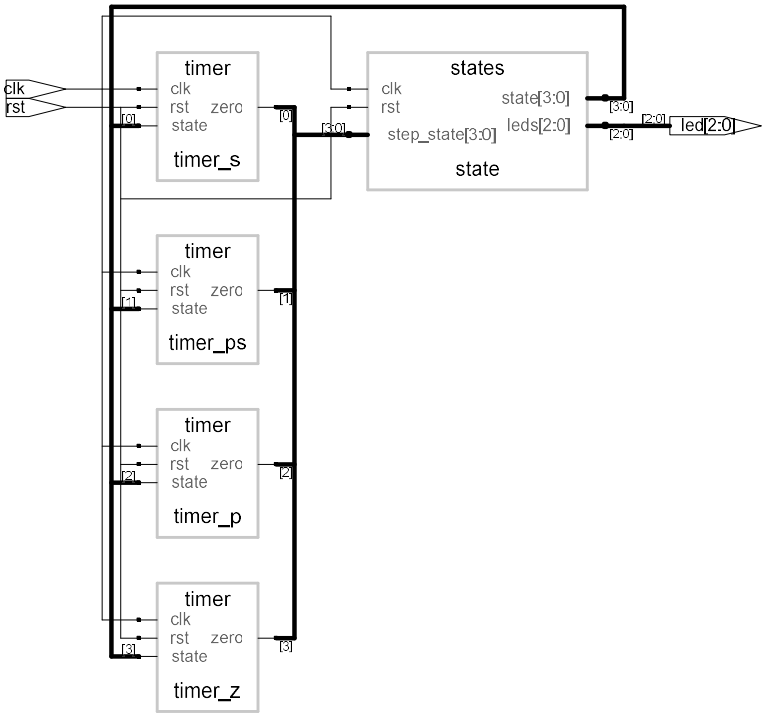
HDL nyelvek

- Alapvetően moduláris felépítésű tervezést tesz lehetővé
- HDL modul
  - Be-, kimenetek definiálása
  - Be-, kimenetek közötti logikai kapcsolatok és időzítések definiálása
- NEM szekvenciálisan végrehajtódó szoftver
  - Alapvetően **időben párhuzamos, konkurens** működést ír le

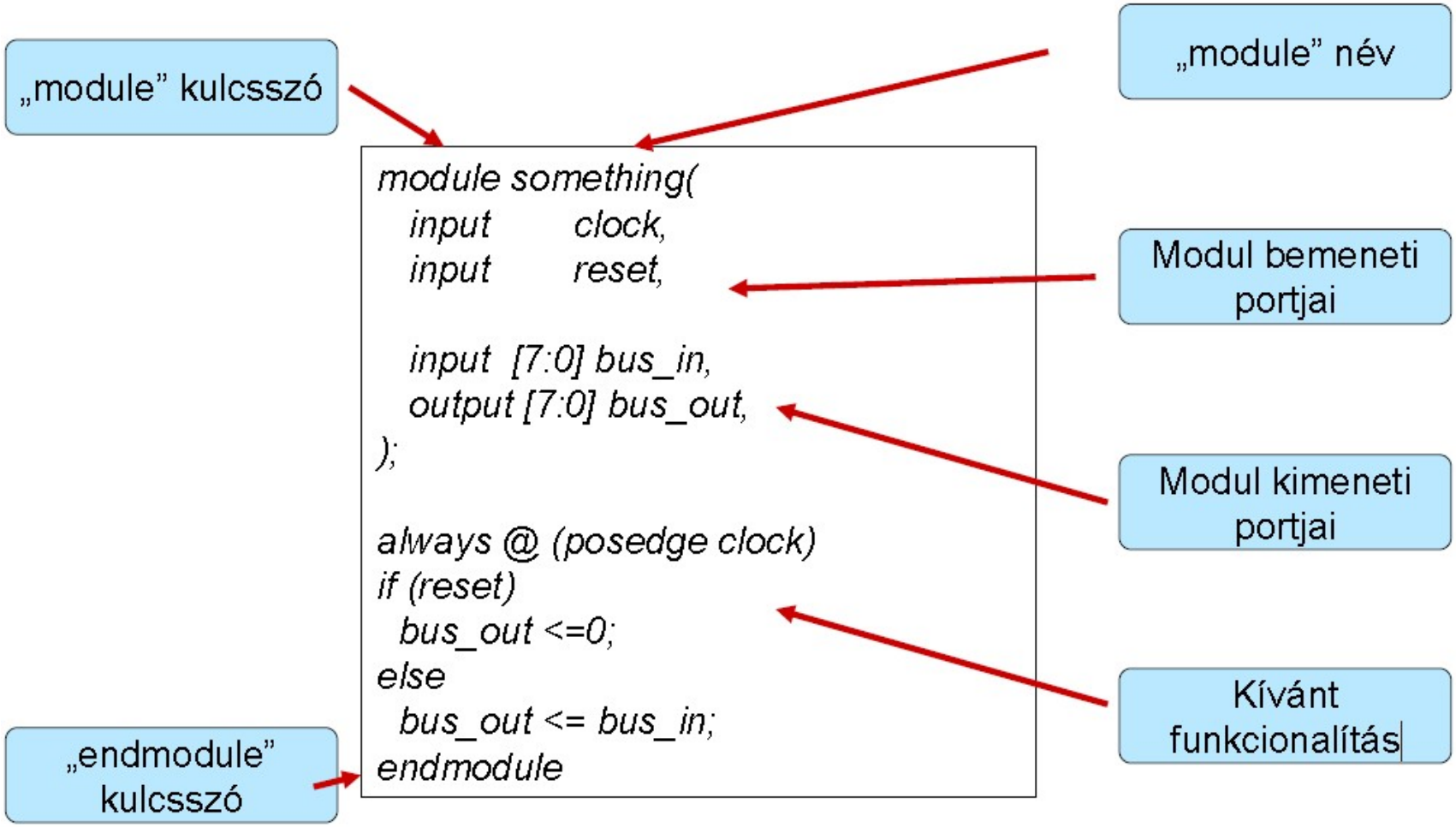
Modulok

- „Építőelem” komplex rendszerek létrehozására
- Hierarchikus leírás, feladat partícionálás
- Top-down tervezés
- Down-top tervezés



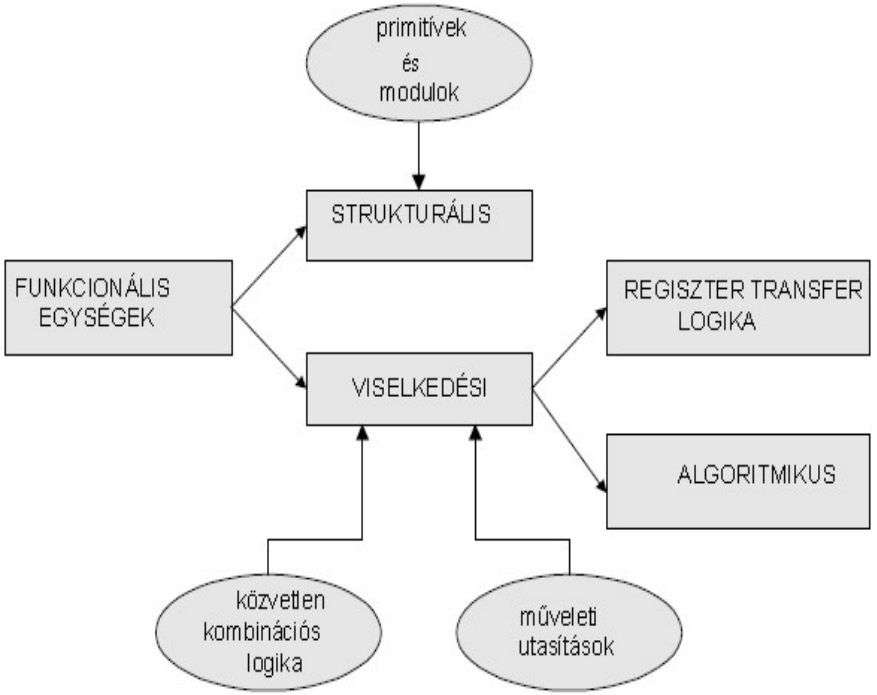


Verilog: module (2001)



Modul működés leírása

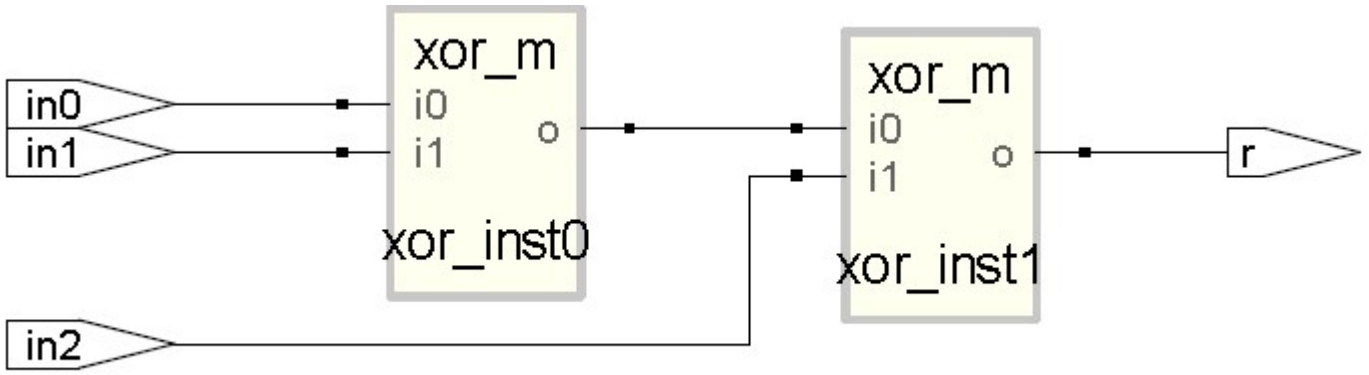
- **Három alapvető modell típus**
  - Strukturális modell
    - Explicit strukturális modell
    - Implicit strukturális modell
  - Procedurális modell
  - a fentiek keveréke is lehet



Strukturális leírás

- Hierarchia felépítése: modulok összekapcsolása

```
module top_level (input in0, in1, in2, output r);
wire xor0;
xor_m xor_inst0(.i0(in0), .i1(in1), .o(xor0));
xor_m xor_inst1(.i0(xor0), .i1(in2), .o(r));
endmodule
```

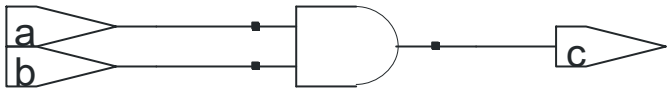


Adattípusok

- „wire” assign
  - Nevének megfelelően viselkedik (vezeték)
  - Folyamatos értékadás;
- „reg” always
  - Szintézis utáni eredmény nem mindig regiszter
    - Vezeték
    - Latch
    - Flip-flop

Assign

- „assign”-val csak „wire” típusú változónak lehet értéket adni
- Folyamatos értékadás
  - A bal oldali változó folyamatosan kiértékelődik
- Pl.
  - assign c = a & b;

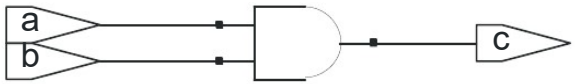


- Egy változó csak egy „assign” által kaphat értéket
- assign értékadások egymással **párhuzamosan** működnek (hardver)
- Kombinációs logika leírására alkalmas

Always

- **Kombinációs logika**

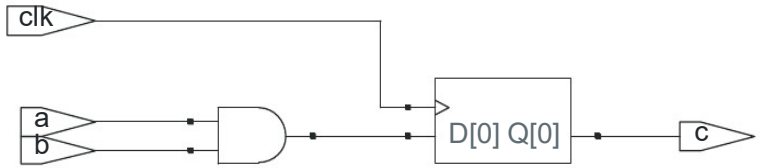
```
always @ (a, b)
   c <= a & b;
```




```
always @ (*)
   c <= a & b;
```

- **Flip-Flop**

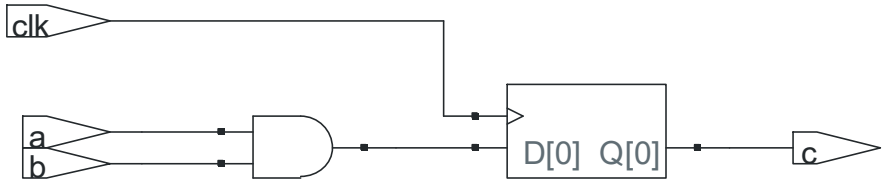
```
always @ (posedge clk)
        c <= a & b;
```



Always – Flip Flop

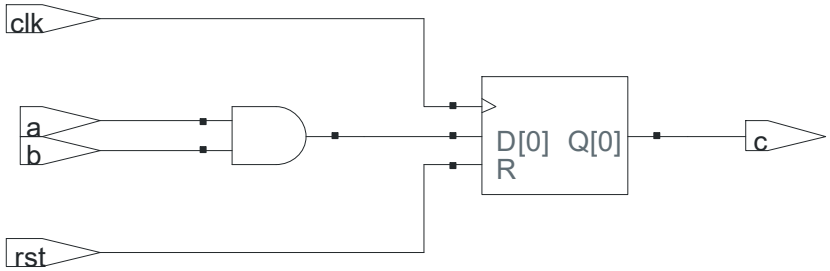
Flip Flop: élérzékeny tároló

```
always @ (posedge clk)
        c <= a & b;
```



Szinkron reset

```
always @ (posedge clk)
if (rst)
        c <= 1'b0;
else
        c <= a & b;
```

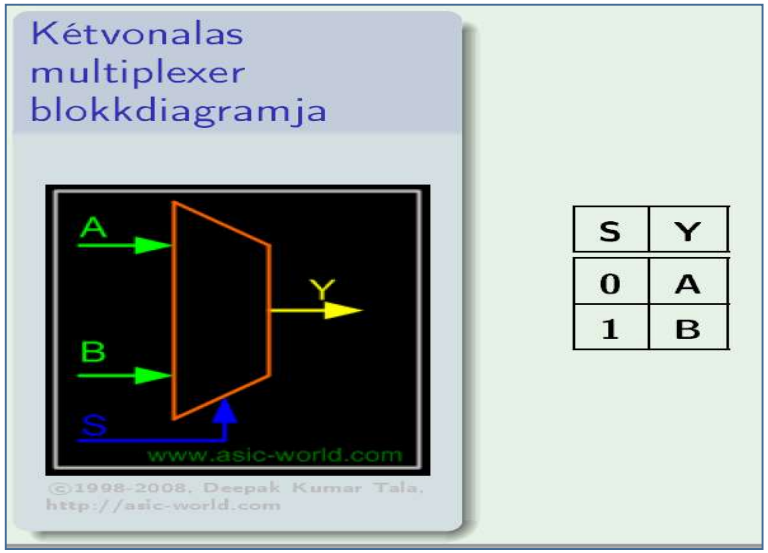


Aszinkron reset

```
always @ (posedge clk, posedge rst)
if (rst)
        c <= 1'b0;
else
        c <= a & b;
```

Példa – MUX

- 2:1 multiplexer



S	Y
0	A
1	B

```
module mux_21 (input A, B, S, output reg Y);

always @ (*)
        if (S==1'b1)        Y <= B;
        else                Y <= A;
endmodule
```

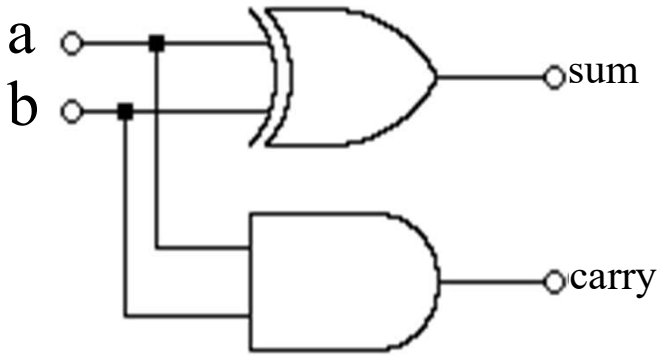
If

```
module mux_21 (input A, B, S, output reg Y);

always @ (*)
 case(S)
    1'b0:    Y <= A;
    1'b1:    Y <= B;
endmodule
```

Case

1-bites fél összeadó Verilog strukturális modellje

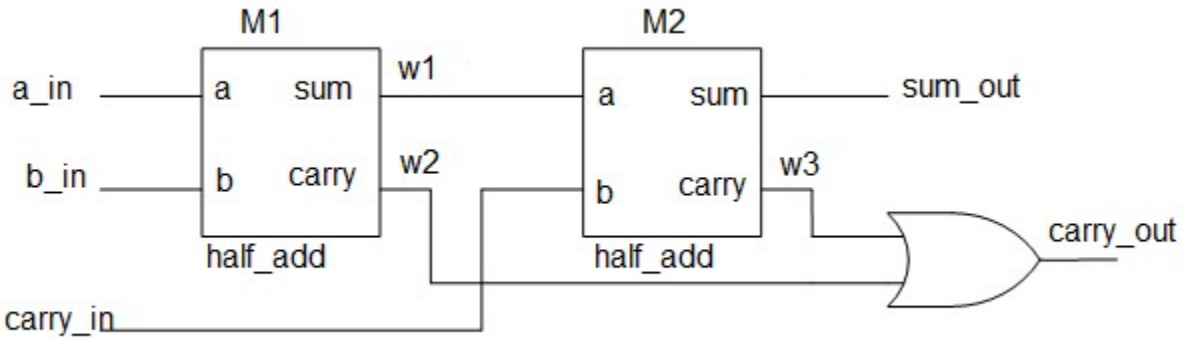



```
module half_add(output sum, carry, input a, b);

        xor (sum, a, b);          // exclusive OR
        and (carry, a ,b);        // and

endmodule
```

1-bites teljes összeadó Verilog strukturális modellje




```
module full_add(output sum_out, carry_out, input a_in, b_in, carry_in);

        wire w1, w2, w3;

        half_add M1 (.a(a_in), .sum(w1), .b(b_in), .carry(w2));
        half_add M2 (.sum(sum_out), .a(w1), .carry(w3), .b(carry_in));
        or (carry_out, w2, w3);

endmodule
```

Példa – 1 bites összeadó

```
module add1_full (input a, b, cin, output cout, s);
        xor3_m xor(.i0(a), .i1(b), .i2(cin), .o(s));
        wire a0, a1, a2;
        and2_m and0(.i0(a), .i1(b), .o(a0));
        and2_m and1(.i0(a), .i1(cin), .o(a1));
        and2_m and2(.i0(b), .i1(cin), .o(a2));
        or3_m or(.i0(a0), .i1(a1), .i2(a2) , .o(cout))
endmodule
```

```
module add1_full (input a, b, cin, output cout, s);
        assign s = a ^ b ^ cin;
        assign cout = (a & b) | (a & cin) | (b & cin);
endmodule
```

```
module add1_full (input a, b, cin, output cout, s);
        assign {cout, s} = a + b + cin;
endmodule
```

Példa – 4 bites összeadó

```
module add4 (input [3:0] a, b, output [4:0] s);
      wire [3:0] cout;
      add1_full add0(.a(a[0]), .b(b[0]), .cin(1'b0),    .cout(cout[0]), .s(s[0]));
      add1_full add1(.a(a[1]), .b(b[1]), .cin(cout[0]), .cout(cout[1]), .s(s[1]));
      add1_full add2(.a(a[2]), .b(b[2]), .cin(cout[1]), .cout(cout[2]), .s(s[2]));
      add1_full add3(.a(a[3]), .b(b[3]), .cin(cout[2]), .cout(s[4]),    .s(s[3]));
endmodule
```

```
module add4 (input [3:0] a, b, output [4:0] s);
       assign s = a + b;
endmodule
```