Budapesti Műszaki és Gazdaságtudományi Egyetem
Elektronikus Eszközök Tanszéke

IT eszközök technológiája

4. előadás

Digitális rendszertervezés

Digitális rendszertervezés

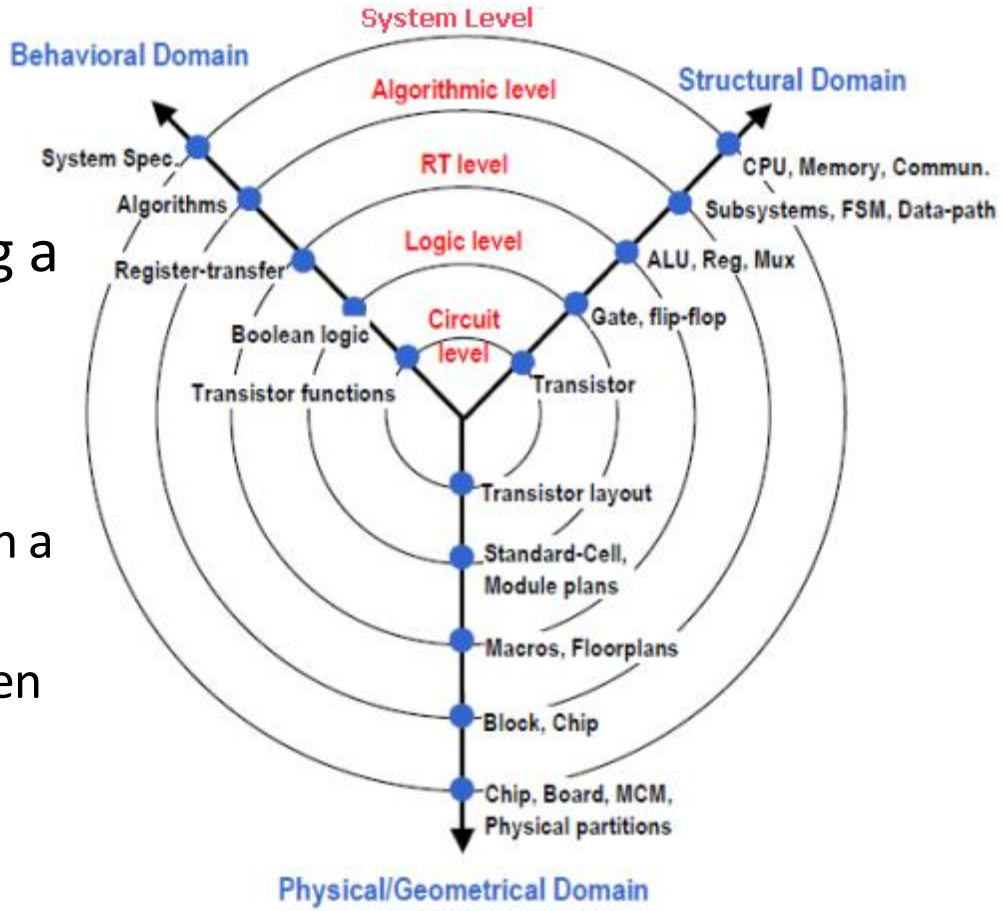
- A digitális tervezés absztrakciós szintjei
- A digitális tervezési útvonal (design flow)
- A digitális design flow eszközei

Absztrakciós szintek

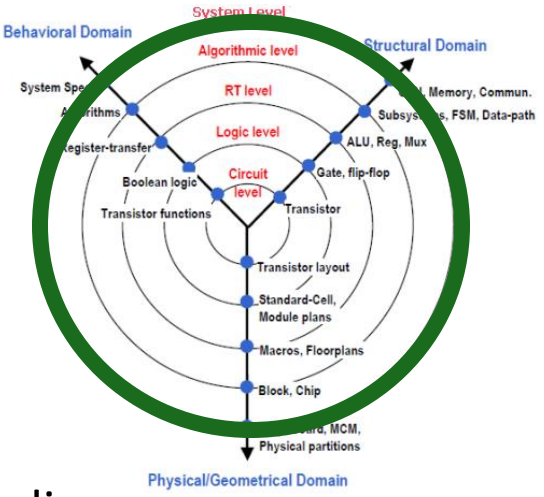
- A digitális rendszerek bonyolultsága exponenciálisan növekszik
  - Egyrészt a technológia lehetőségek, másrészt az igények…
- A tervezési metódusoknak ezzel kell(ene) lépést tartani
  - Ez csak úgy lehetséges, hogy egyrészt a tervezés egyre magasabb absztrakciós szinten történik.
  - Másrészt a fizikai szintre történő leképezés automatikusan, de ember által felügyelt határok és kényszerek (design constrains) történik.
  - A tervezés több, egymást követő lépésből áll, amelyek során az emberi tényező szerepe egyre csökken.
  - Az automatikus eszközök használata kikerülhetetlen, még az elméletileg elérhető hatékonyság rovására is.
    - vö: ki kódol assemblyben 20XX-ban? Ki fog banki rendszert technológiai szempontból hatékony nyelven, pl. C++-ban megvalósítani?

A Gajski-Kuhn Y diagramm

- Az absztrakciós szinteket a koncentrikus körök
- A vizsgálat szempontjait pedig a nyilak reprezentálják.
  - Pl. a rendszer specifikáció struktúrában CPU, memória, kommunikáció, míg fizikai szinten a chip, kártya stb.
  - A fizikai szinttel csak érintőlegesen foglalkozunk a továbbiakban.

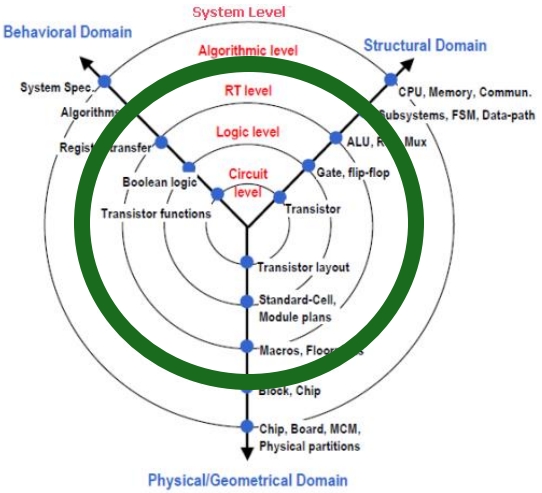


A rendszerszint



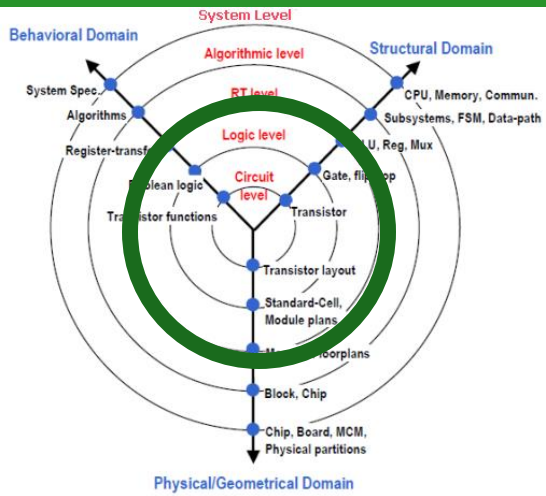
- A fő cél a terv részekre bontása (partícionálás)
  - Az egyes részek feladatainak rögzítése
  - A közöttük és külvilággal történő kommunikáció kipróbálása, kiválasztása, modellezése.
  - A tervező nem foglalkozik a megvalósítás részleteivel – teljesen hardverfüggetlen leírást készít – amelyből akár több különböző megvalósítás is keletkezhet.
  - Lehetőség van szoftver és hardver együttes tervezésére.
- Eszközök
  - Általános célú magas szintű programnyelvek illetve célszoftverek

Az algoritmus szintje

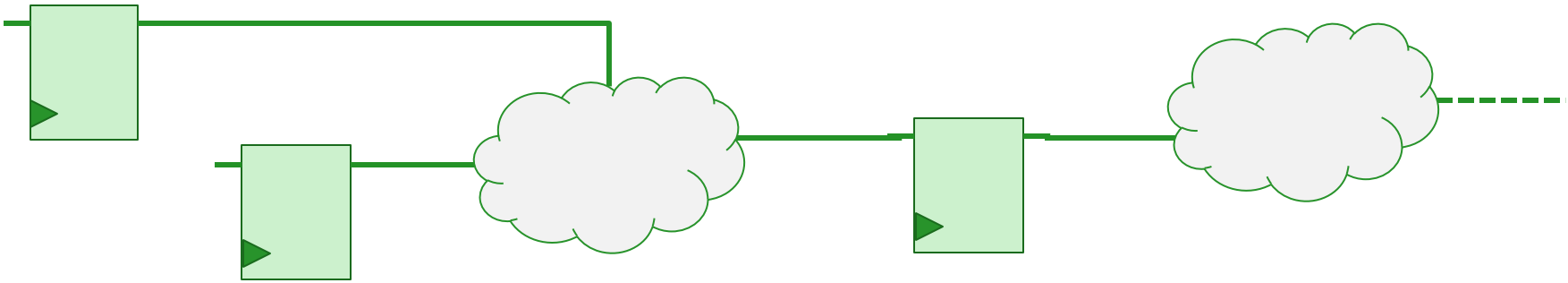


- Az egyes alrendszerek által megvalósított funkció viselkedési szintű modellezése
- A viselkedési modellek segítségével elvégezhető szimulációja
- A szimuláció erőforrásigényes, így alapvetően a modellezés elsődleges eszközei a C, C++
- Léteznek olyan osztálykönyvtárak, amelyek lehetőséget adnak az algoritmus „bit” és időzítéspontos modellezésére is.
- Ilyen pl. a SystemC
  - Ingyenesen elérhető http://accellera.org/downloads/standards/systemc

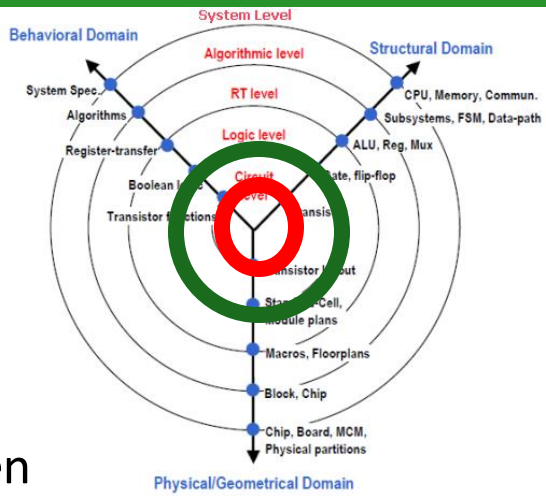
A regiszter-transzfer szint (RTL)



- „Az az elvonatkoztatási szint, ahol a **regisztereket** és a közöttük végbemenő **adatátvitelt** definiáljuk, beleértve az adatátvitelt **összeköttetéseket** és az átvitel **időzítését**”
- Magyarul: Mi történjen az áramkörben az **órajel két aktív éle között**
- (feltételezzük, hogy szinkron hálózatokat tervezünk...)



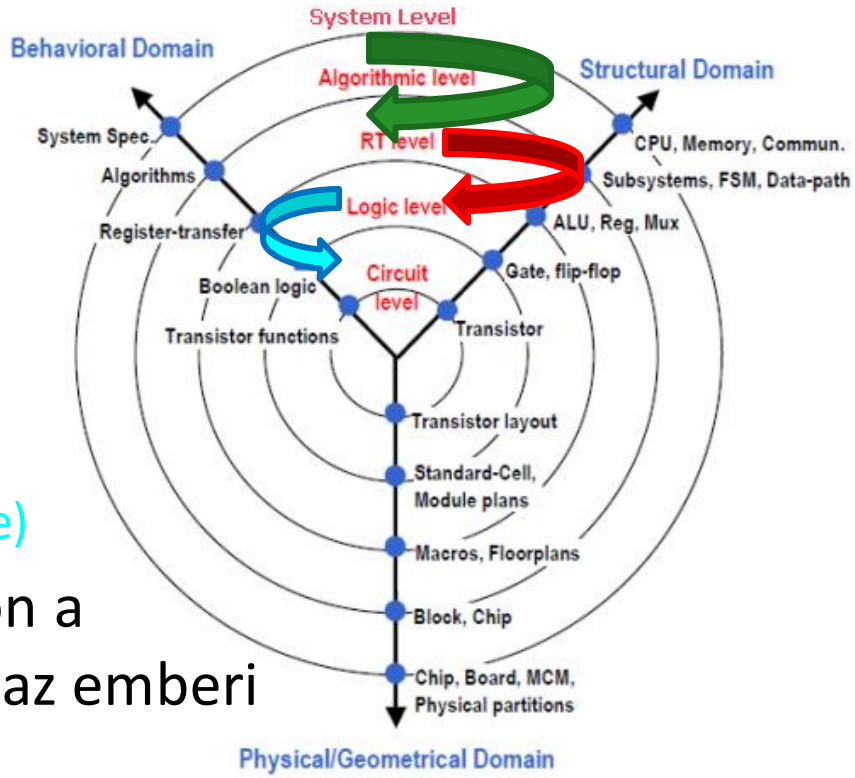
A logikai szint és az áramköri szint



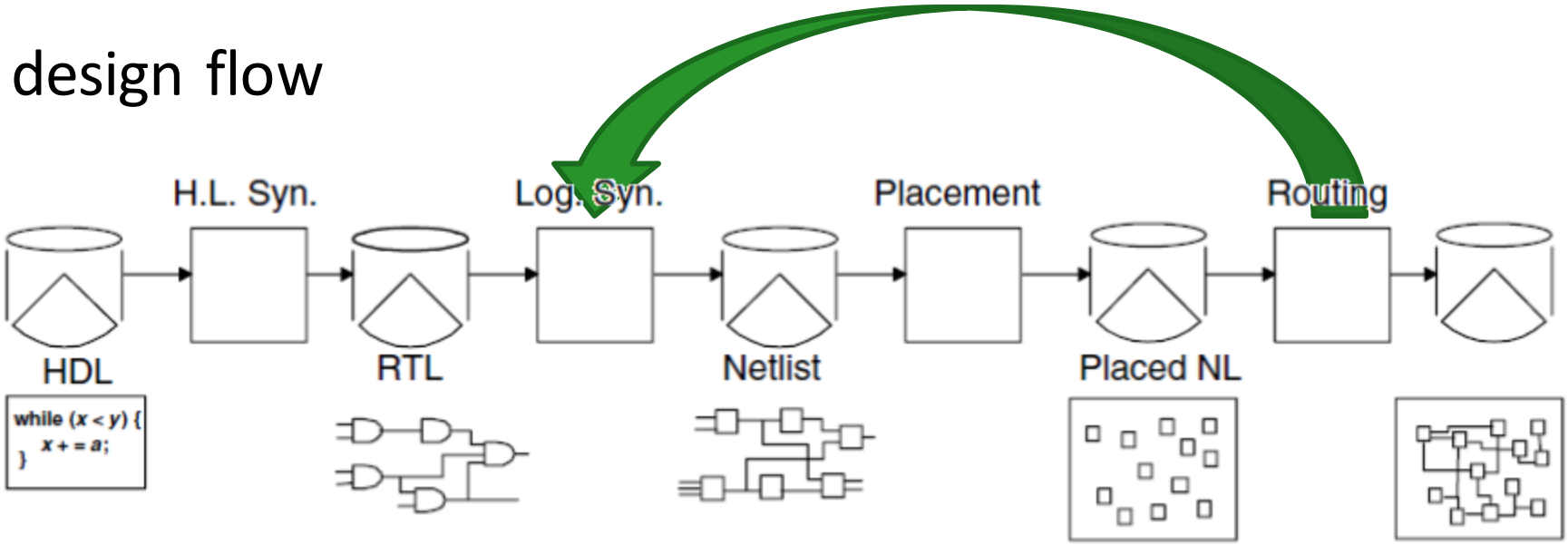
- Logikai szint
  - Logikai kapuk és összeköttetéseik
  - Azaz hálózatlista (netlist)
  - A logikai szint eléréséig a tervezés megvalósításfüggetlen
  - (többé kevésbé…)
- Áramköri szint
  - Kapcsolási rajz
  - Fizikai terv
    - (hogyan épülnek fel rétegből a tranzisztorok és hogyan vannak összekötve)
  - Ez az ún. physical design.

A szintézis

- A magasabb absztrakciós szintről az alacsonyabb absztrakciós szintre szintézissel kerülünk.
  - Ez történhet gépi vagy emberi úton
  - Magas szintű szintézis (HLS)
  - Logikai szintézis
  - Elhelyezés és huzalozás (place and route)
- Ahogy előre haladunk a diagrammon a mélyebb rétegek felé, úgy váltja fel az emberi tevékenységet az automatikus (de általában ember által felügyelt) szintézis.



A design flow



- A tervezés azonban (sajnos) nem ilyen egyszerű, hanem iteratív
  - A CMOS logikában ugyanis a késleltetés pontosan csak az összekötő vezetékhosszak ismeretekor számítható.
  - Azaz az elhelyezés és huzalozás után van pontos késleltetés.
  - Ez visszahathat minimum a logikai szintézisig, de néha feljebb is.

Magas szintű szintézis

- Algoritmikus szintről RTL szintre
- Leggyakrabban még emberi közreműködéssel készült, de a magas szintű szintézer programok egyre jobb minőséget képviselnek.
  - Cadence / Stratus
  - Mentor Graphics / Catapult C
  - Synopsis / Synphony C compiler
- Időzítésmentes C kódból generálnak RTL szintű leírást
- 5-10x tervezői produktivitást ígérnek

Magas szintű szintézis

- Akár kézzel, akár számítógépes programmal történik a RTL kód előállítása, a következő problémákat kell megoldani:
- **Vezérlés** jellegű funkció esetén
  - **Állapotgépek** konstruálása
  - Állapotgépek és **kisegítő áramkörök együttműködésének** megszervezése
- **Adatfeldolgozás** jellegű funkció esetén: mikroarchitektúra választás/szintézis
  - Erőforrás-allokáció
  - Ütemezés
  - Erőforrás-vezérlési állapot összerendelés *(binding, resource sharing)*

Az automatikus HLS előnyei

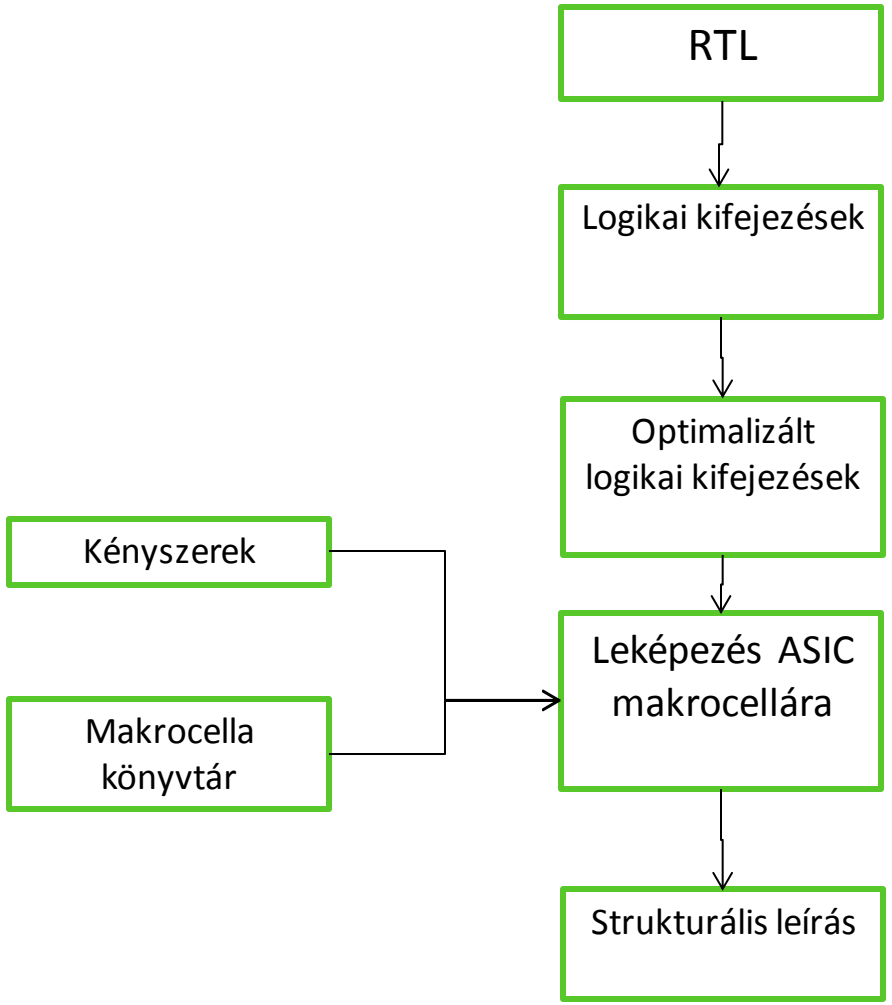
- Egy 1M ekvivalens kapuval rendelkező terv
  - Ez nem tekinthető igazán nagynak…
- (ekvivalens kapuszám: az elhelyezett cellák területe/a NAND2 területe)
  - Kb. 300k sor RTL szinten
  - 30-40k sor algoritmikus szinten
- Az újrafelhasználás könnyebb
  - Az RTL kód mikroarchitektúra és interface szinten kötött
  - Egy technológiai váltás esetén már nem lesz optimális
- A kényszereket magasabb szinten alkalmazzuk
- Egy algoritmikus leírásból készülhet több, különböző mikroarchitektúra, amiből a technológiai lehetőségek figyelembe vételével lehet az optimálisat kiválasztani.

A logikai szintézis

- Bemenet
  - RTL kód
  - A cellakönyvtár (a rendelkezésre álló kapuk) leírása (hdl nyelven)
  - A cellakönyvtár időzítési és teljesítményadatai
- Kimenet
  - Strukturális HDL, ami már csak a cellakönyvtárbeli elemeket tartalmazza. (valójában ez már netlistnek tekinthető)
- Kényszerek:
  - Időzítés, terület, teljesítmény
  - (alapesetben becsült)
  - Együtt tud működni a fizikai elhelyezést végző layout szintézis eszközzel, így jobb minőségű kimenetet szolgáltat.
    - (jobb lesz mindhárom kényszer becslése)

A logikai szintézis

- RTL leírás → strukturális leírás (kapuszintű)
- Kényszerek:
  - Időzítés
  - Teljesítmény
  - Terület
- ASIC makrocella
  - Ami az adott technológiában megtalálható
    - IC: kapuk
    - FPGA: LE-k

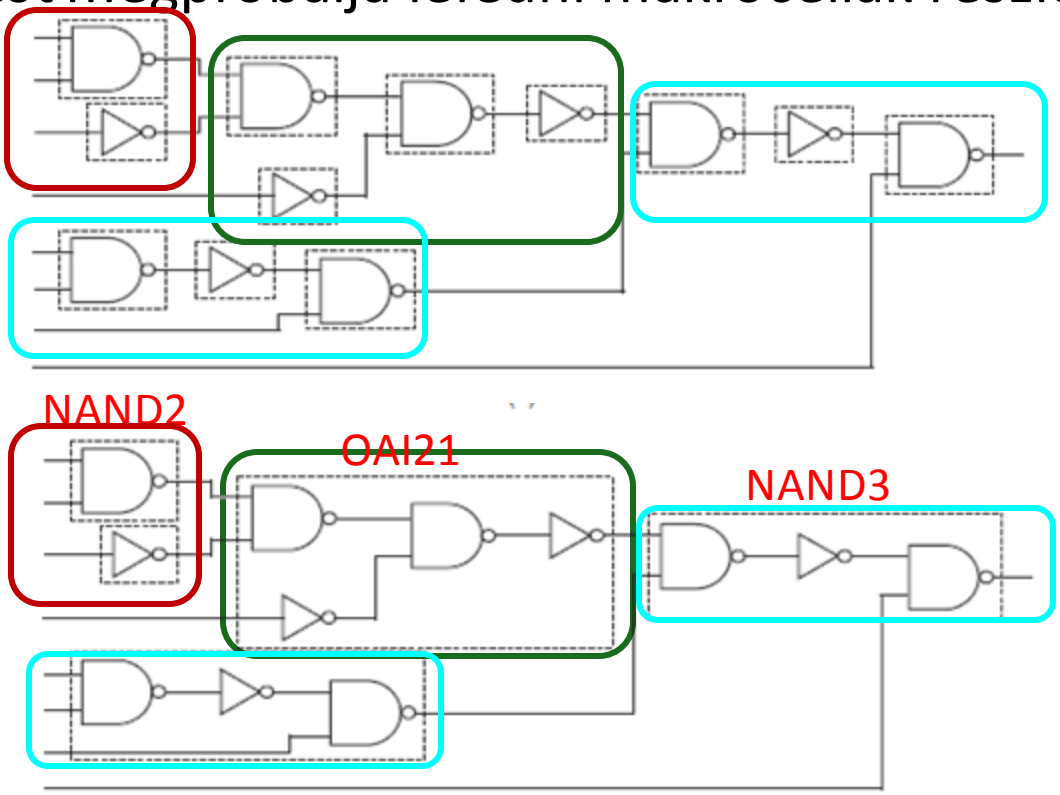


Lépések

1. HDL beolvasása, optimalizálás (pl. dead code removal)
2. A hierarchia kifejtése, ha szükséges
   - Ez az ún. flattening – jobb minőségű megoldás kapható globális optimalizációval – nagyságrendekkel erőforrásigényesebb, mintha modulonként készülne.
   - „spend CPU cycles rather than human cycles”
3. Logikai kifejezések optimalizálása (generic)
   - Felismer szerkezeteket, pl. összeadó, számláló stb.
   - (Itt fontos, hogy a szintézer „ízlése szerint” kódoljuk az RTL-t.)
     - Ezeket megfelelő template-tel helyettesíti
     - Automatikusan, vagy pragmával lehet megadni az architektúrát
       Pl. összeadó esetén CLA, CSA stb.
   - Optimalizálja a logikai kifejezéseket
     - Hasonlóan, mint anno Digitben, de nem biztos, hogy kétszintű a megvalósítás.
4. Leképezés ASIC makrocellára
   - Template-ek esetében ez viszonylag egyszerű
   - Logikai kifejezések esetén a makrocella könyvtárból válogat megfelelő cellákat, amivel a kifejezést meg lehet valósítani.

Leképezés ASIC makrocellára – vázlatos elv

- A logikai kifejezésből egy fát épít, ami kétbemenetű NAND kapukat és invertereket tartalmaz.
  - Ugyanezt elvégzi az összes makrocellára.
- A logikai kifejezést megpróbálja lefedni makrocellák részleteivel.



Hardverleíró nyelvek

HDL-ek

- VHDL (***V**HSIC **H**ardware **D**escription **L**anguage*, IEEE STD 1987)
  - VHSIC: ***V**ery **H**igh **S**peed **I**ntegrated **C**ircuit*
  - Amerikai Védelmi Minisztérium (DoD) megrendelésére
  - Eredeti célja: beszállított ASIC-ek funkciójának formális dokumentálása
- SystemVerilog (***VERI**fication and **LOG**ic*, IEEE STD 2005)
  - Gateway Design Automation → Cadence Design Systems
  - Eredeti célja: logikai szimuláció
  - Az eredeti nyelv (Verilog) 2005-ben továbbfejlesztve → SystemVerilog

Terminológia

- A HDL-ek **NEM programozási nyelvek!**
  - **Hasonló** nyelvi szerkezetek
  - Részben vagy teljesen **eltérő jelentés**

- Az elterjedten használt szakkifejezések félrevezetők, pontatlanok
  - HDL program → **HDL modell**
  - HDL kód lefordítása → **HDL modell elemzése** *(elaboration)* vagy **HDL-modell szintézise**, attól függően, hogy épp melyik folyamatot nevezzük helytelenül fordításnak…
  - HDL program futtatása → **HDL szimuláció**
  - FPGA programozása → **FPGA konfigurálása**
  - stb…

A VHDL története

- Az Amerikai Védelmi Minisztérium megrendelésére fejlesztették ki
- Célja a DoD számára leszállított ASIC-ek funkcionális specifikációjának egyértelmű dokumentálására használták
- Később logikai szimulációra, azután pedig automatizált áramkörszintézis bemenetéül is alkalmazták
- 1987-ben IEEE szabvány (IEEE 1076-1987)
  - Kiegészítés: IEEE STD 1164: 9 állapotú logikai jel (??? 01UXZWLH- ☺)
- További fejlesztések: 1993, 2000, 2002
- A legfrissebb szabvány: VHDL 2008 (IEEE 1076-2008)
  - Újrafelhasználható elemek modellezése
  - Funkcionális verifikáció

A VHDL

- Az ADA programozási nyelv az alapja.
- Erősen típusos, szimbólumok helyett inkább kulcsszavakat használ
  - Így ember számára jobban olvasható.
  - Kisbetű nagybetű érzéketlen.
- Nagy projektek létrehozására is alkalmas
  - Támogatja a hierarchikus tervezést
  - (könyvtár (library), csomag (package))

VHDL példa

```
library ieee;
use ieee.std_logic_1164.all;

entity d_flip_flop is
  port (clk:   in  std_logic;
        rst_n: in  std_logic;
        d:     in  std_logic;
        q:     out std_logic);
end entity d_flip_flop;

architecture rtl of d_flip_flop is
begin
  L_MAIN: process (clk, rst_n)
  begin
    if ( rst_n = '0' ) then
      q <= '0';
    elsif ( rising_edge(clk) ) then
      q <= d;
    end if;
  end process;
end architecture rtl;
```

Könyvtár- és csomagdeklarációk

Egyed deklaráció

- generikus paraméterek
- portlista

Architektúra

- modul funkcionális leírása
- almodulok példányosítása
- Több architektúra is lehetséges

A SystemVerilog története

- A Verilog nyelvet a Gateway Design Automation cég fejlesztette ki.
- A GDA-t a Cadence Design Systems felvásárolta, a Verilog nyílt szabvánnyá vált.
- Az első IEEE szabvány 1995-ben készült el.
- A szabványt többször frissítették, a 2005-ös szabványfrissítés (SystemVerilog) jelentős változtatásokat tartalmazott, elsősorban a verifikációt támogató nyelvi elemeket fejlesztve.
- A SystemVerilog 2009-es változata már tartalmazza a teljes Verilogot.

SystemVerilog példa

```
module d_flip_flop (input  logic clk,
                    input  logic rst_n,
                    input  logic d,
                    output logic q);

  always_ff @ (posedge clk, negedge rst_n)
  begin
    if ( !rst_n ) q <= 0;
    else q <= d;
  end

endmodule
```

Modul deklaráció

- generikus paraméterek
- portlista

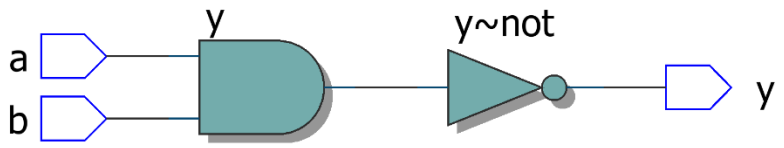
Modultörzs

- modul funkcionális leírása
- almodulok példányosítása

NAND kapu

```
library ieee;
use ieee.std_logic_1164.all;

entity nand_gate is
  port (a: in  std_logic;
        b: in  std_logic;
        y: out std_logic);
end entity nand_gate;

architecture rtl of nand_gate is
begin

  y <= a nand b;

end architecture rtl;
```

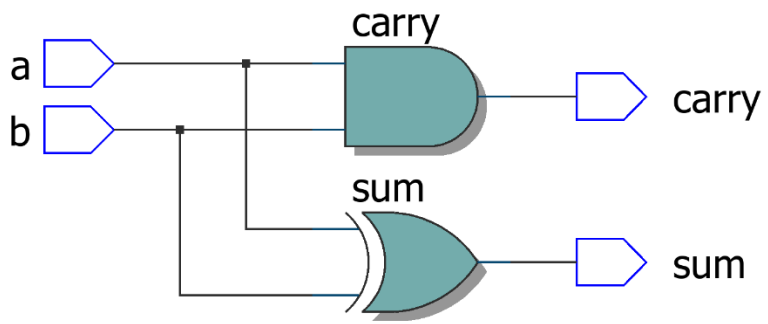
```
module nand_gate (input  logic a,
                  input  logic b,
                  output logic y);

  assign y = ~(a & b);

endmodule
```



Félösszeadó (RTL szinten)

```
library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
  port (a:     in  std_logic;
        b:     in  std_logic;
        sum:   out std_logic;
        carry: out std_logic);
end entity half_adder;

architecture rtl of half_adder is
begin

  sum <= a xor b;
  carry <= a and b;

end architecture rtl;
```

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

  assign sum = a ^ b;
  assign carry = a & b;

endmodule
```



A koncepció

- Láthattuk, hogy mindkét elterjedt HDL hasonló koncepciók alapján működik. (legalábbis a szintetizálható részhalmaz)
- Az utasítások „párhuzamosan” hajtódnak végre! A sorrend mindegy!
- Definiálni kell az entitás/modul be és kimeneteit
- Kombinációs hálózat modellezésénél egyszerűen leírjuk, hogy mi történik
  - Azaz hogyan kapjuk meg a kimeneteket a bemenetek függvényében!
- Szinkron hálózat modellezése
  - Megadunk egy érzékenységi listát és leírjuk, hogy az érzékenységi listában szereplő változások esetén pontosan mi történik.

A SystemC

Történet

Alapfogalmak

Mintapéldák

A System C

- C++ osztálykönyvtár digitális rendszertervezésre
  - Alkalmas a „klasszikus” hardverleírásra is.
    - Azaz bit és késleltetés szinten pontosan leírható a hardver működése
    - (jelentős szintaktikai többlettel, még a VHDL-hez képest is…)
  - Alkalmas magas szintű tervezésre
- ESL: Electronic System Level design
  - A terv könnyebben átlátható
  - A hardver/szoftver együtt tervezhető
  - A hardver architektúrák és a köztük lévő interfészek könnyen modellezhetők
- C++ fejlesztői környezetben reprezentálunk hardvert
  - Azaz: párhuzamosság, időzítés, késleltetés, port, bit pontos adattípusok
    - Ezért kell a C++ osztálykönyvtár (valójában a modellezést végző kód nagyrészt C)
  - Alkalmas lefordított (compiled code) szimulációra (gyors!)
  - A magas szintű szintézer eszköz bemenete lehet.

A System C története

- Nagyon sok helyen használnak C++ -t hardver modellezésre és szimulációra
  - (nem véletlenül: eredetileg Stroustrup is…)
  - **An Efficient Implementation of Reactivity for Modeling Hardware in the Scenic Design Environment (1997)**
  - Érdemes lett szabványosítani
    - Synopsys kezdeményezésére első verzió 1999-ben.
  - Open SystemC Initiative
    - Mindhárom nagy EDA (Electronic Design Automation) cég (Cadence, Mentor Graphics, Synopsys) részt vesz benne.
    - IEEE 1666–2011 szabvány
    - Nyílt forráskódú
    - http://www.accellera.org/downloads/standards/systemc

Alapfogalmak

- Modul
  - Egy adott funkcionalitás megvalósítására szolgál. Más modulokat és process-eket foglalhat magába. (VHDL entity vagy Verilog module)
  - Egy (megfelelően származtatott) C++ osztály
- Process
  - A funkció leírására szolgál. A modult megvalósító osztály metódusa. (VHDL process vagy Verilog always szerkezet megfelelője)
- Port
  - A logikai jel csatlakozó pontja
    - Iránya van! (eltérően egy C változótól…)

Mintapélda, NAND kapu

```
#include <systemc.h>

SC_MODULE (nand2)  // ronda makró az öröklésre az sc_module-ból
{
        sc_in<bool>  a, b; // bemeneti portok
        sc_out<bool> y;    // kimeneti port

        void nand2_method() // a működés leírása
        {
                y.write( ! ( a.read() && b.read() )); // y= ! (a && b)
        }


        SC_CTOR (nand2) // constructor for nand2
        {
                SC_METHOD (nand2_method); // ez a függvény írja le a működést
                sensitive << a << b; // érzékenységi lista
        }
};
```

Mintapélda, D-FF

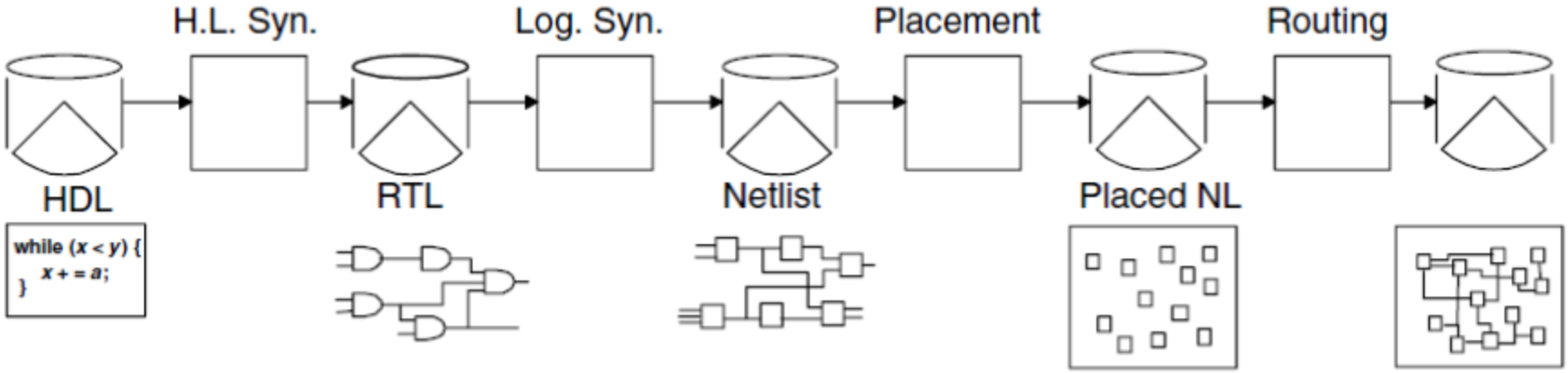
```
#include <systemc.h>

SC_MODULE (dff)
{
    sc_in<bool>  clk;
    sc_in<bool>  din;
    sc_out<bool> dout;

    void dff_method()
    {
         dout= din; // dout.write( din.read())
    }

    SC_CTOR (dff)
    {
        SC_METHOD (dff_method);
        sensitive_pos << clk;
    }
};
```

Szimulációs kernel

- A SystemC tartalmaz egy beépített eseményvezérelt szimulációs kernelt.
- A main függvényben írhatjuk meg a gerjesztéseket
  - Bár elegánsabb HDL módjára testbench-et készíteni és a main-ben „csak” összekötni.
- A lefordított fájl maga a szimulátor!
  - Ez nagy szimulációs sebességet eredményez.
- A teljes C/C++ eszközkészlet és a rendszer könyvtárai rendelkezésre állnak a szimuláció kimenetének vizsgálatára
  - Ez modern HDL szimulátorokban is megoldható, bár nem egyszerű. (VPI/VHPI)
  - Már az std::cout lehetőségei is sokkal bővebbek mint a hullámforma kimenet.
  - Itt található egy komplett karakteres videokártya megvalósítás, SDL alapon
    - http://www.eet.bme.hu/~czirkos/icterv/syscvideo-0.6.zip

Verifikáció és szimuláció

A design flow



- Magasabb absztrakciós szintről alacsonyabb absztrakciós szintre kézi vagy gépi szintézissel kerülünk.
- Valamilyen módon meg kell bizonyosodni arról, hogy az alacsonyabb szintű reprezentáció megfelel a magasabb szintűnek.
  - Ember által végzett szintézis esetén ez létfontosságú!
- Ennek folyamata a verifikáció
- Eszközei pedig a szimuláció, ill. a különböző ellenőrző programok.

Időzítés és teljesítmény a különböző absztrakciós szinteken

Szint	Időzítés és teljesítmény
Rendszerszint	Csak a teljes rendszerre ismert követelmény
Algoritmus	Tervező által becsült
RTL	Program által becsült vagy visszavezetett
Logikai szint	Kapu: pontos, huzalozás becsült vagy visszavezetett
Áramköri szint	Pontos (nagyon jól szimulált)

- Ahogy haladunk lefelé a hierarchiában, egyre pontosabb képünk van a késleltetésről és a teljesítményről
- A huzalozás által okozott késleltetés csak a fizikai tervezéskor határozható meg pontosan.
  - A tervezési folyamat iteratív!
  - A kiszámított késleltetés és teljesítményadatok visszavezethetők magasabb szintre – ez az ún. back annotation.

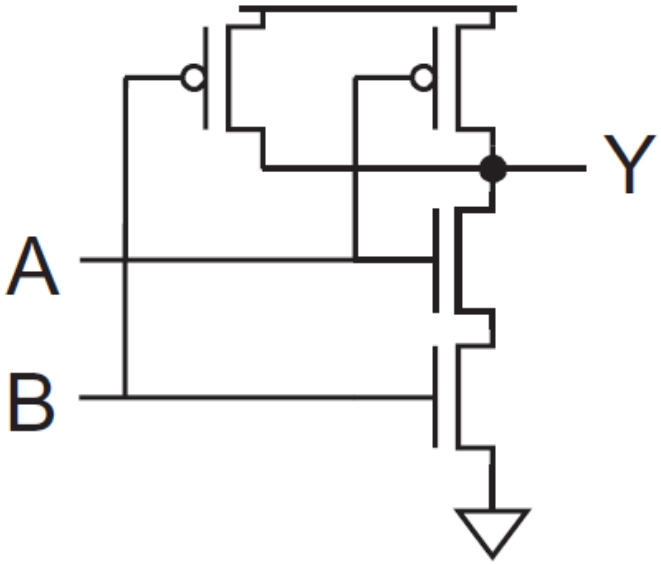
A szimuláció

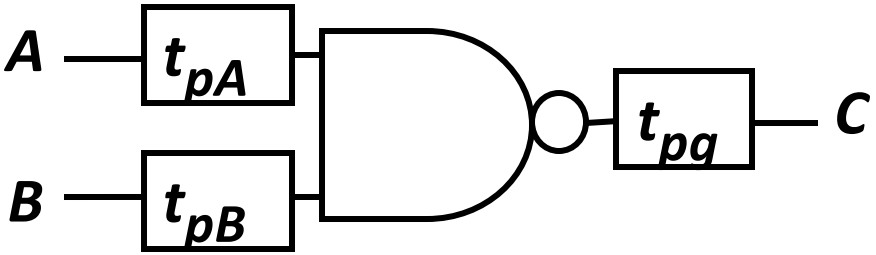
- A verifikáció eszköze
  - Szimulátor segítségével jóval könnyebb a hibák felderítése
- Minden szinten használt a feltételezések és a működés ellenőrzésére.
- Az egyes absztrakciós szinteken lefelé haladva egyre bonyolultabb és erőforrásigényesebb a szimuláció:
  - Rendszerszintű szimuláció
  - Logikai szimuláció
  - Áramköri szimuláció
  - Technológiai szimuláció
  - Stb.
- Az alacsonyabb szintű szimulációk feladata többek között, hogy segítségükkel gyorsan szimulálható modelleket állítsunk elő a magasabb szintű szimulátor számára.

A logikai szimuláció

- A logikai szimuláció a valóság **közelítése**
  - A fizikai folyamatok nagy része erősen **leegyszerűsítve** jelenik meg a szimulációban.
  - Logikai értékek (0, 1, X, Z) a folytonos feszültségértékek helyett
    - **Esemény** absztrakt fogalma: a logikai érték megváltozik.
    - Csak az **események hatását vizsgáljuk**, az események között az áramkör viselkedése nem releváns
    - A szimuláció csak a leegyszerűsítések után is értelmezhető hibák felderítésére képes (logikai, nem áramköri hibák)!
    - Védekezés: a logikai szimulációban a határozatlan (X) és inicializálatlan (U) értékek „önfenntartók”

Egy CMOS NAND kapu modellje





- t_{pA}, t_{pB} az A ill. B bemenet késleltetése (összeköttetések miatt)
- t_{pg}: a kapu késleltetése, amit többféleképpen kezelhetünk.

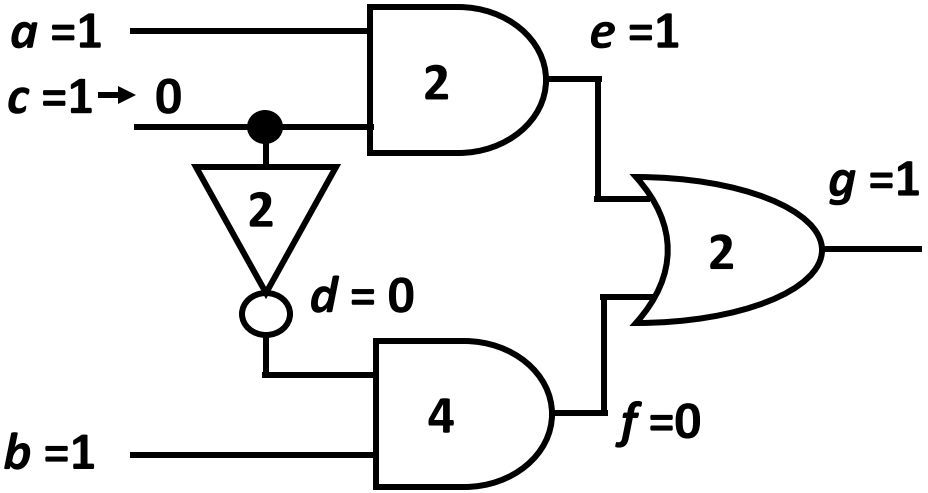
A logikai szimuláció – zero delay

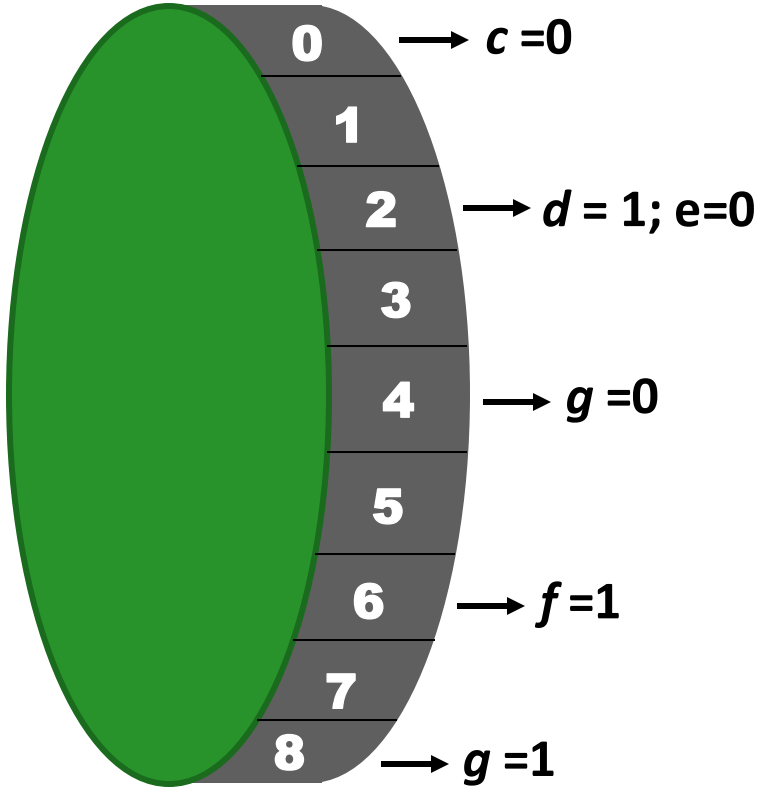
- Késleltetésmentes (zero delay)
  - Algoritmus ill. RT szinten.
  - A kombinációs hálózatok kimenetén az érték a bemenetek megváltozásakor rögtön megjelenik.
  - Szinkron hálózatokban az új értékek az órajel aktív éle után előállnak
- Gyakorlatilag a Boole egyenletek statikus megoldása
  - Gyors, de a késleltetésekről semmilyen információt nem ad.
  - Az alapvető logikai működés ellenőrizhető vele.

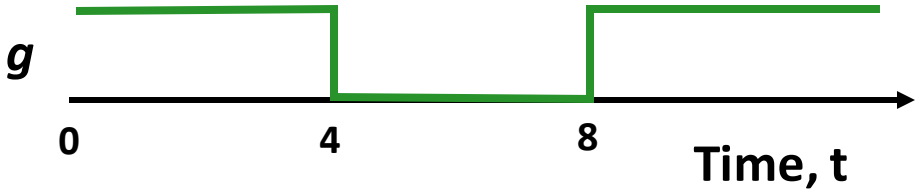
Az eseményvezérelt logikai szimuláció

- Eseményvezérelt (event driven) szimuláció
  - A hálózatnak csak azt a részeit kell szimulálni, ahol változás történik.
  - Ez általában kisebb része, mint 10%.
- Az esemény valamilyen változás a kapu bemenetén.
- Ezt az eseményt ki kell értékelni.
  - Ha a kapu kimenete megváltozik, ez újabb eseményt jelent, amit be kell iktatni az esemény sorba (priority queue) , a megfelelő késleltetéssel.
  - Ha minden, adott időpillanathoz tartalmazó eseményt feldolgoztunk, akkor előállt a megoldás és továbblépünk.
  - Ezt a megvalósítást hívják időkeréknek (time wheel)

Példa

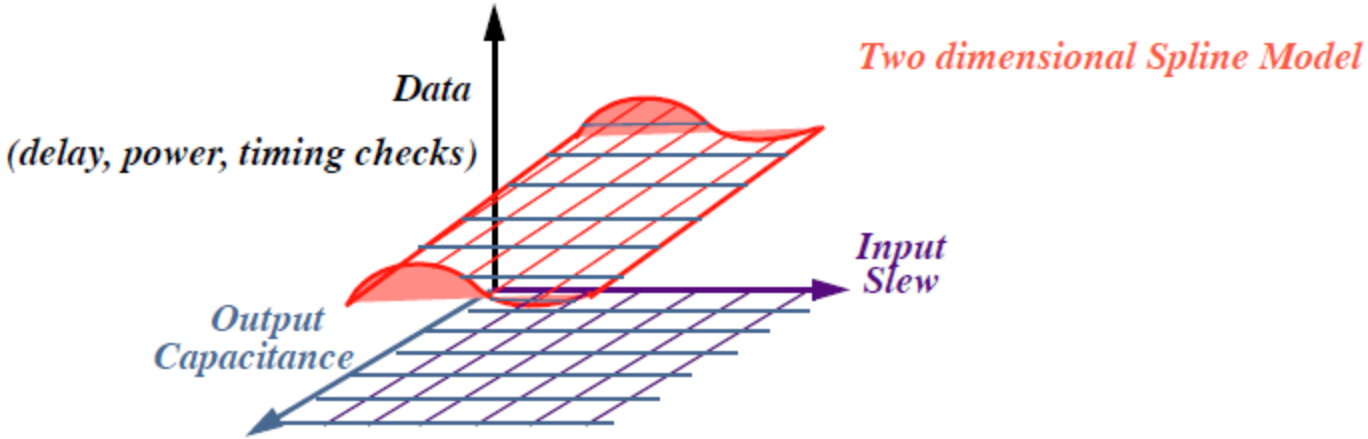






A kapuk késleltetés és teljesítmény adatai

- A standard cellák méret és késleltetés adatait adatbázisokban tárolják.
- Az időzítés függ a bemeneti jel fel vagy lefutásától és a kimenet terhelésétől (a kapcsolódó kapuk számától és az összeköttetések hosszától)
  - A cella áramköri szimulációjával állítják elő
  - Egy kétdimenziós táblázat, amelyből általában interpolációval állapítják meg az aktuális késleltetést. (nem túlzottan nagy, kb. 5x5)
  - $t_{PD} = f(t_t, C)$
- A fizikai elhelyezés ismeretében számítható ki pontosan.



A fizikai tervezés

Fizikai tervezés

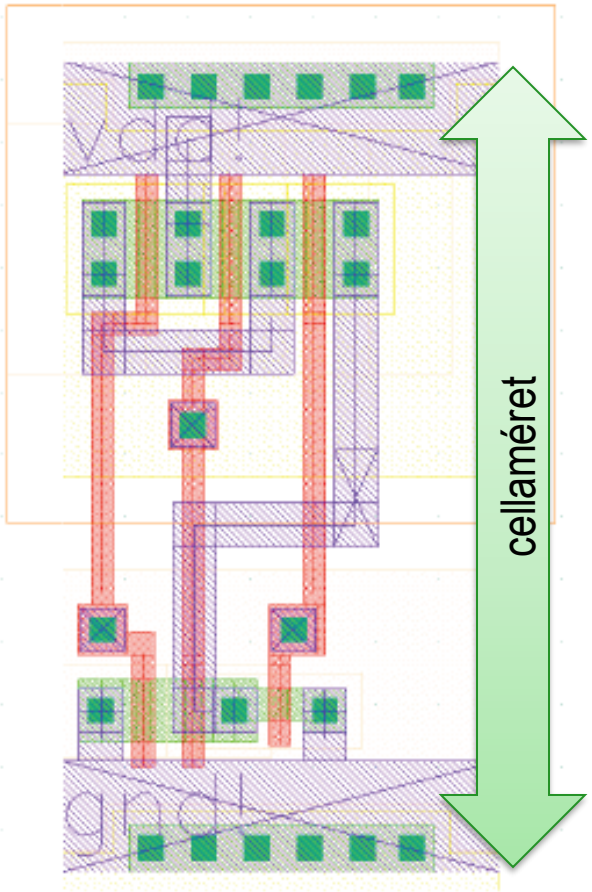
- A tervezés eddigi lépései nagyjából technológiafüggetlenek voltak
  - (valójában már az RTL esetén gondolkodni kell a megvalósítási technikán és ahhoz tartozó optimális mikroarchitektúrát választani… - ez akár az algoritmus szintjére is felkerülhet)
    - Pl. FPGA-ban általában van DSP blokk, így egy szorzást tartalmazó művelet kevésbé kínos, mint IC esetében.
- A kész terv megvalósítási lehetőségei:
  - Teljesen kézzel, tranzisztor szinten, „full custom” módon.
    - Ez csak kritikus blokkok esetében képzelhető el
  - Előre tervezett cellakönyvtár segítségével:
    - Ez az ún. standard cellás tervezés
  - Programozható logikai eszköz segítségével
    - FPGA, CPLD stb.
    - Ezekről majd később…

A cellakönyvtár

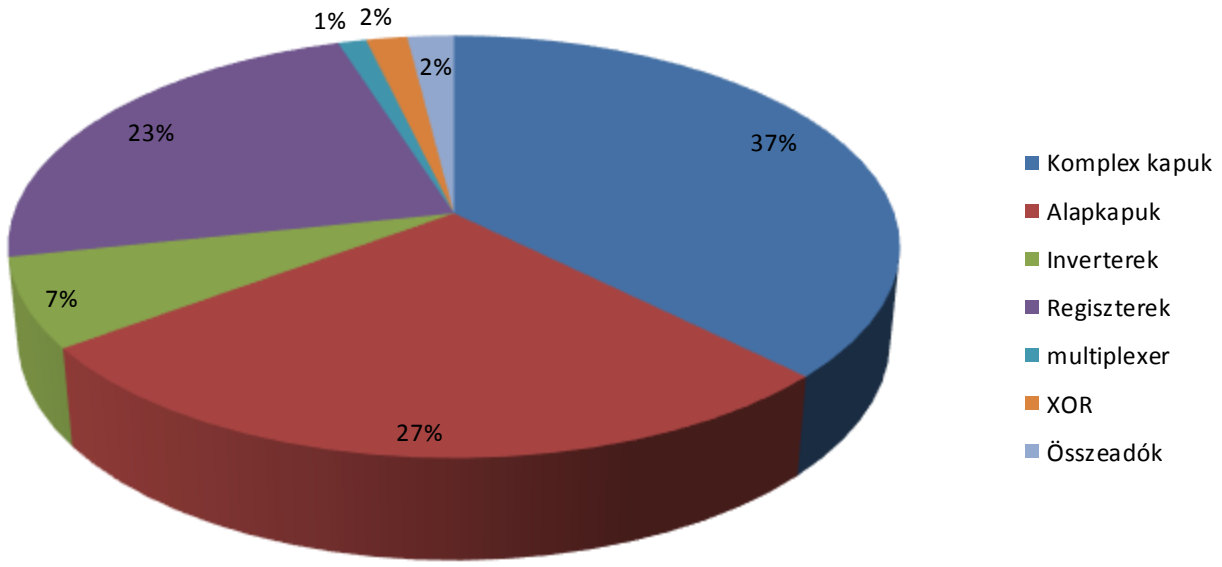
- Alapvető logikai áramkörök gyűjteménye
  - Alapkapuk, komplex kapu, flip-flop-ok stb.
  - Különböző meghajtóképességgel
    - (kimeneti árammal – ugyanazon terhelés mellett a nagyobb meghajtóképességű gyorsabb lesz.)
    - Ebből következően a fizikai méret és fogyasztás is más lesz
- A standard cella magassága rögzített, szélessége változhat.
  - Így a cellák azonos magasságú sorokban, az ún. cellasorokban helyezhetők el.
  - A cella tetején a táp, a cella alján a föld sín található.
  - A cella belsejében általában az első fémrétegen történik a huzalozás, így a magasabb szintű fémezés a cellán keresztülhúzható.

Példa: egy komplex kapu cellája

- A különböző színek a tranzisztorokat és a huzalozást leíró geometriai alakzatokat jelképezik.
  - Ezek alapján készül el a litográfiai maszk a gyártáshoz
- Egyszerre ábrázolva jutunk az ún. layout-hoz.
  - Az összes, gyártáshoz szükséges mintázat.
- Pl. $Y = \overline{AB + C}$ komplex kapu terve
  - Kék: 1.fémréteg
  - Piros poliszilícium, a tranzisztor gate anyaga
  - Zöld : az ún. aktív zóna, ide kerül adalékolás (S,D)
  - Kékes zöld: kontaktus a rétegek között



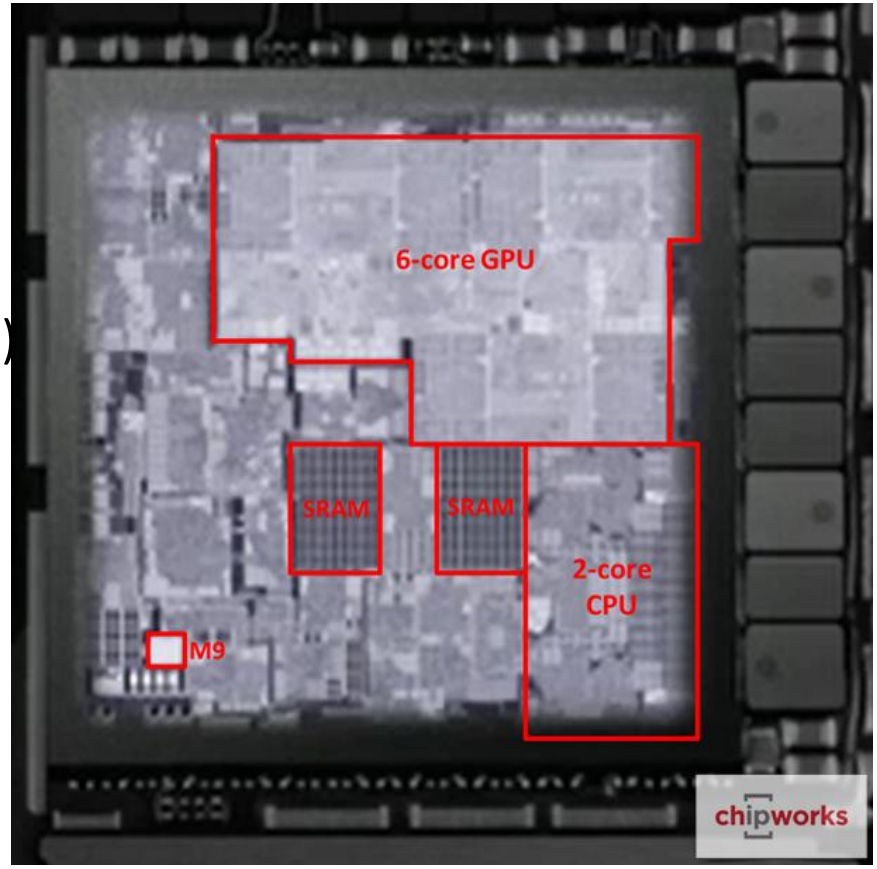
Példa cellakönyvtár használatára



- 6502 processzor mag (OpenCores.org)
- AMS 350nm, logikai szintézis: RTL compiler, alapbeállításokkal.
- 1679 cella, kb. 0,14mm^2, 12088 db. tranzisztor

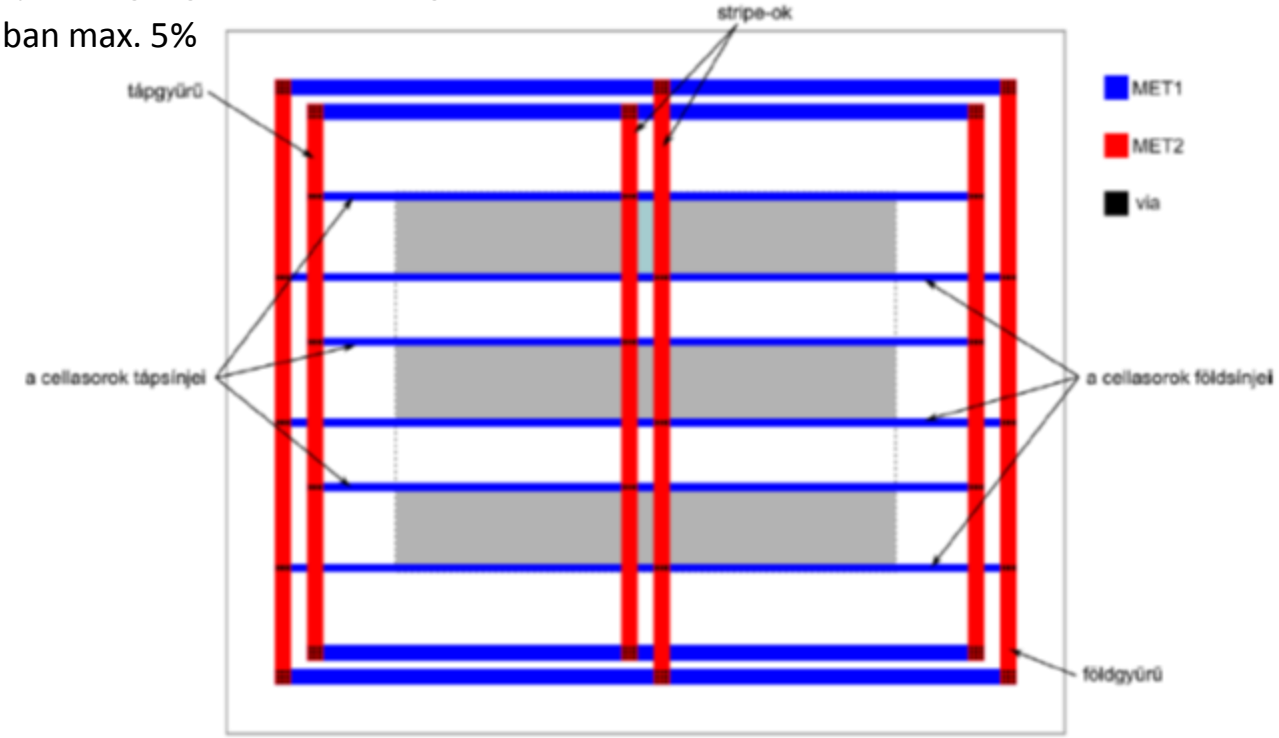
1. Floorplan

- Az áramkört alkotó blokkok, a be és kimenet elhelyezése a chip felszínén
  - Core: az áramköri mag
  - Pad: a kivezetések
    - A kivezetések körbeölelik az áramkör magját, innen az elnevezés: pad-ring.
  - (Apple A9 – jól követhető a floorplan)



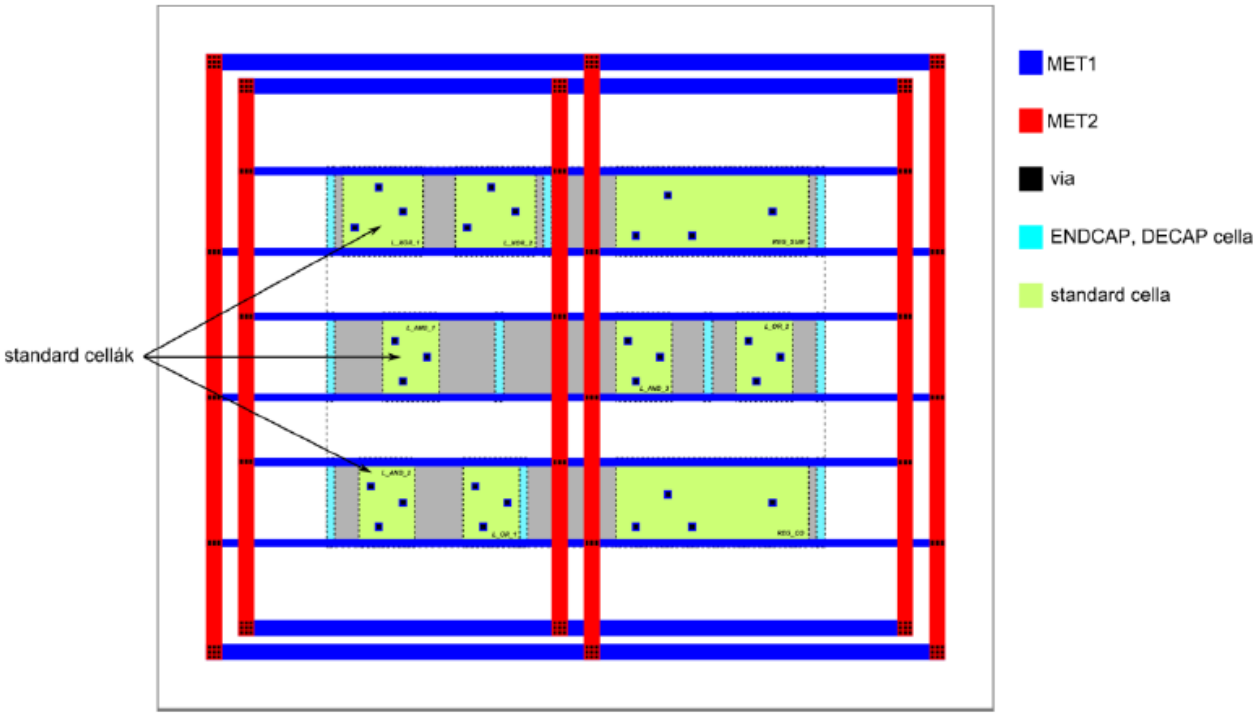
2. Tápellátás (power plan)

- Meg kell határozni a statikus és a dinamikus áramfelvételt.
  - CMOS esetben a dinamikus áramfelvétel adja a fogyasztás nagy részét.
  - Ehhez az egyes kapuk aktivitásának ismerete szükséges, amit logikai szimulációkkal kell előállítani.
- Az átlagos és a maximális fogyasztás ismeretében megtervezhető a tápellátó hálózat.
  - IR drop: a megengedett feszültségesés a cellán, maximális áram esetén
  - Általában max. 5%



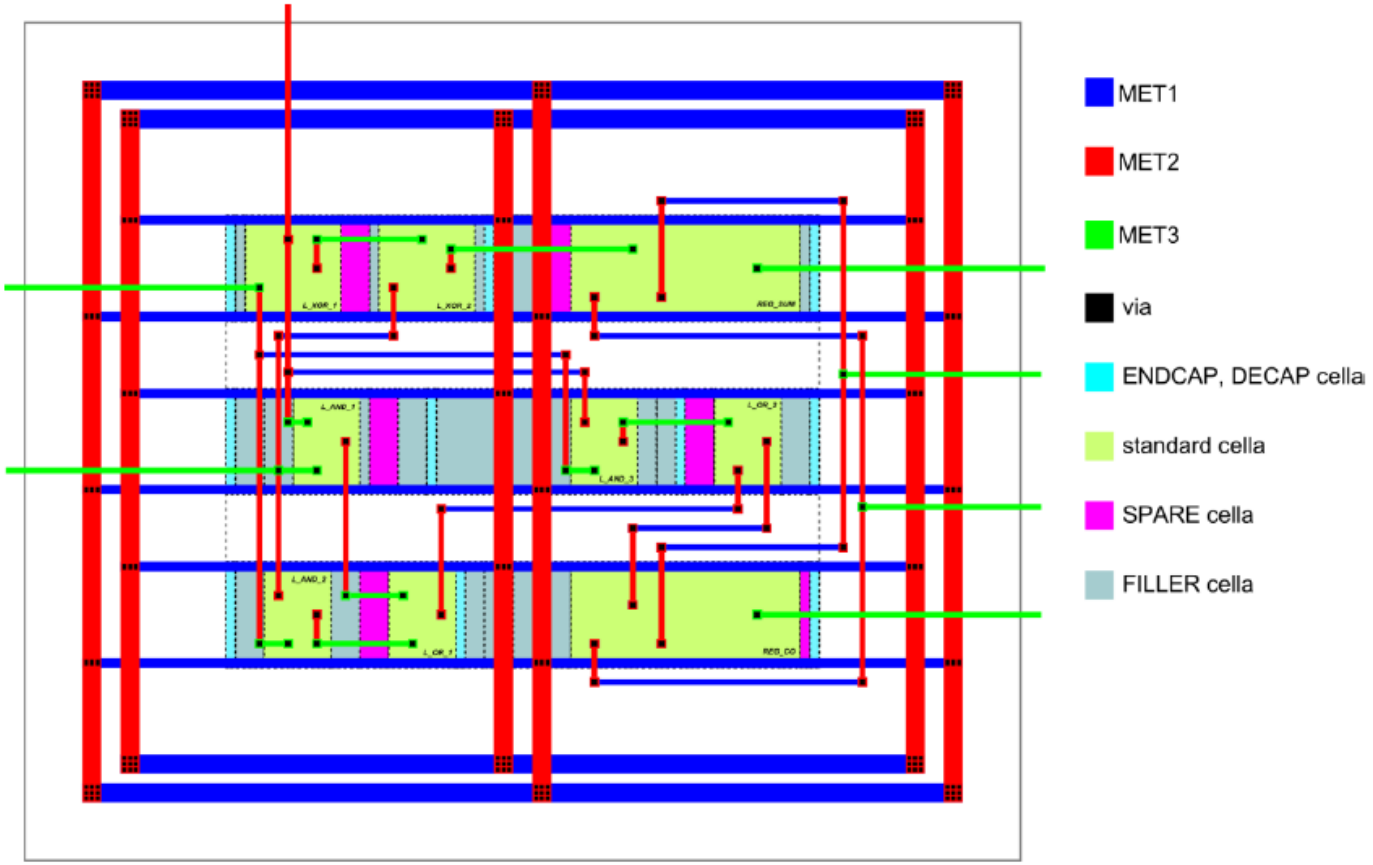
3. A cellák elhelyezése (place)

- A logikai funkciót megvalósító standard cellák elhelyezése

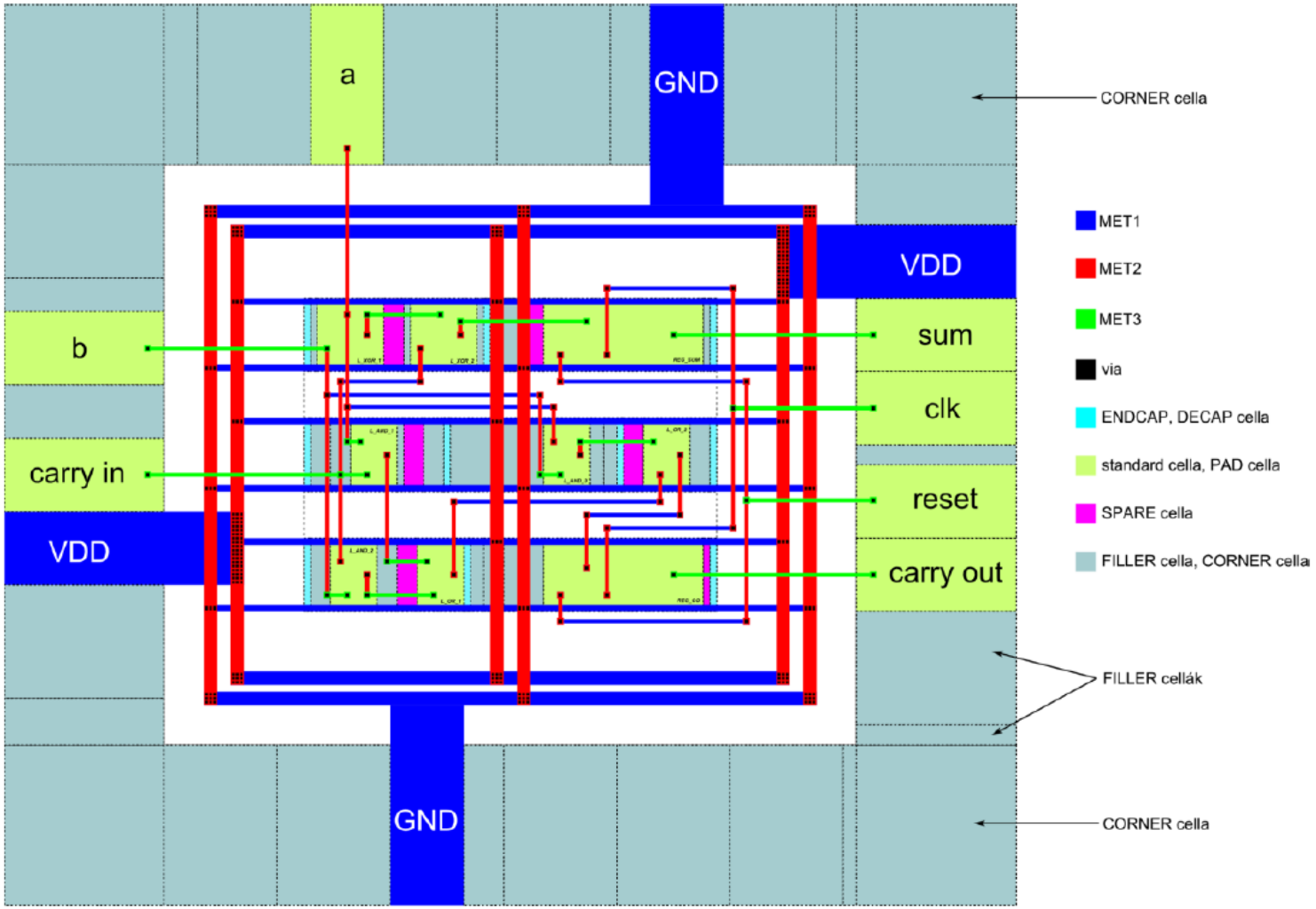


4. Huzalozás (route)

- A cellák közötti huzalozás elkészítése



5. A pad-ring elkészítése



A post-layout szimuláció

- A fizikai tervezés befejezésével az összes kapu kimeneti terhelése pontosan ismert
  - A fizikai tervből a geometriai méretek visszafejthetőek, ezek ismeretében pedig a kapuk terhelése kiszámítható.
- Így pontos késleltetési adatok állnak rendelkezésre
  - Ekkor hajtják végre a post-layout szimulációt.
  - Az időzítési adatok visszavezethetők a logikai vagy az RT szintű leírásba.
  - Így újra lehet ellenőrizni a tervet.
  - (és újrakezdeni, ha az időzítés nem megfelelő…)

A félvezető IP

A félvezető IP (semiconductor IP)

- IP core vagy IP block: egy újrafelhasználható egység, amely kereskedelmi vagy egyéb úton a végfelhasználóhoz (itt: a digitális rendszer tervezője) jut.
- Mintha alkatrészt venne és összeszerelné a chip felszínén
  - De ez az alkatrész csak virtuálisan létezik.
    - A helyzet hasonló, mint egy szoftverkönyvtár esetében.
  - A blokk felhasználásáért licenszdíjat fizet.
- A mai bonyolultságok mellett gyakorlatilag nem lehet mindent „házon belül” kifejleszteni.
  - A jól bevált blokkokat érdemesebb megvenni.
  - Sok cég sikeres üzleti modellje, pl. ARM

Soft IP Core

- Szintetizálható RTL leírás (Verilog vagy VHDL nyelven)
  - A forrás titkosítható, a szintézist végző program tudja kikódolni a licenszkulcs ismeretében.
- Vagy generikus netlista (ld. logikai szintézis lépései)
  - Az RTL leírásból készített általános, csak absztrakt kapukat és feldolgozó egységeket tartalmazó hálózatlista
- Mindkét esetben tetszőleges technológiára szintetizálható
  - Azaz hordozható különböző gyártók között
- Az adott technológiára történő fizikai tervezést és az optimalizálást azonban a tervezők végzik, ami nem mindig egyszerű feladat.
  - Sem az időzítés, sem az elfoglalt terület nem ismert előre és az IP gyártója sem tud garantálni emiatt semmit.

Hard IP Core

- A fizikai tervezés végeredménye, azaz a layout.
- Egy adott félvezetőgyártó technológiájához kötődik
  - A méret, a késleltetés és a fogyasztás garantált és ismert.
  - Analóg áramkörök esetén mindig, digitális áramkörök esetén pedig gyakran kézzel tervezett és optimalizált részeket is tartalmaz.
  - Emiatt minden paraméterében általában jobb minőségű, mint a soft IP
  - Viszont félvezetőgyártóhoz kötött
    - Más gyártó választása esetén újat kell licenszelni, nem hordozható.