

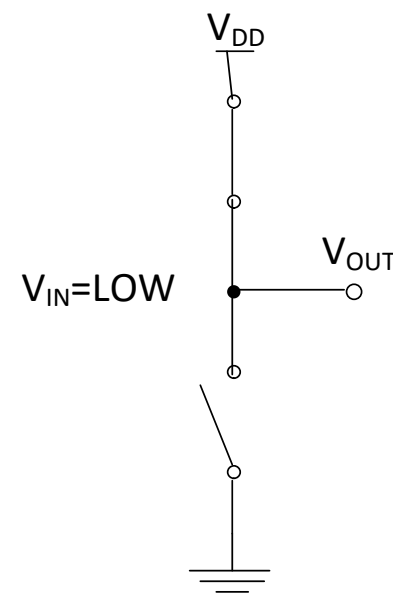
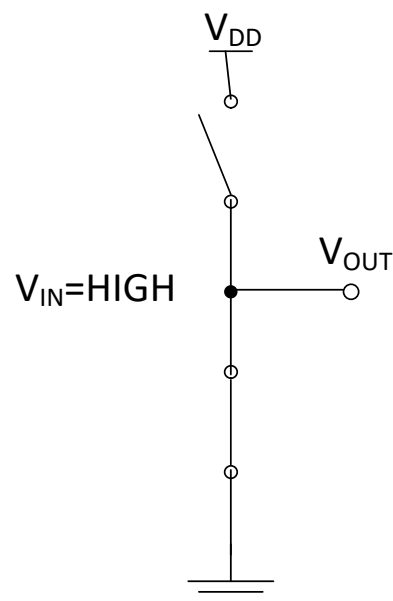
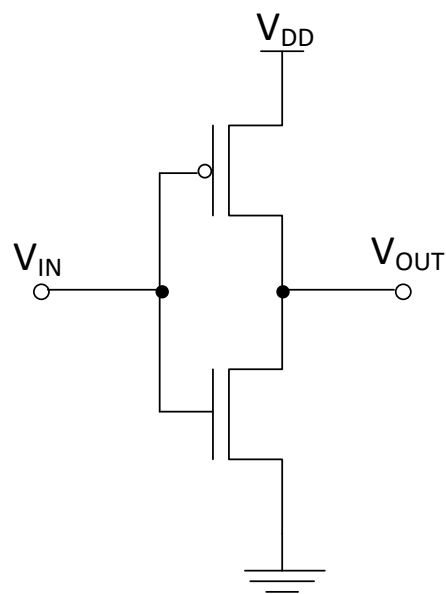


Budapesti Műszaki és Gazdaságtudományi Egyetem
Elektronikus Eszközök Tanszéke

CMOS áramkörök késleltetése és fogyasztása

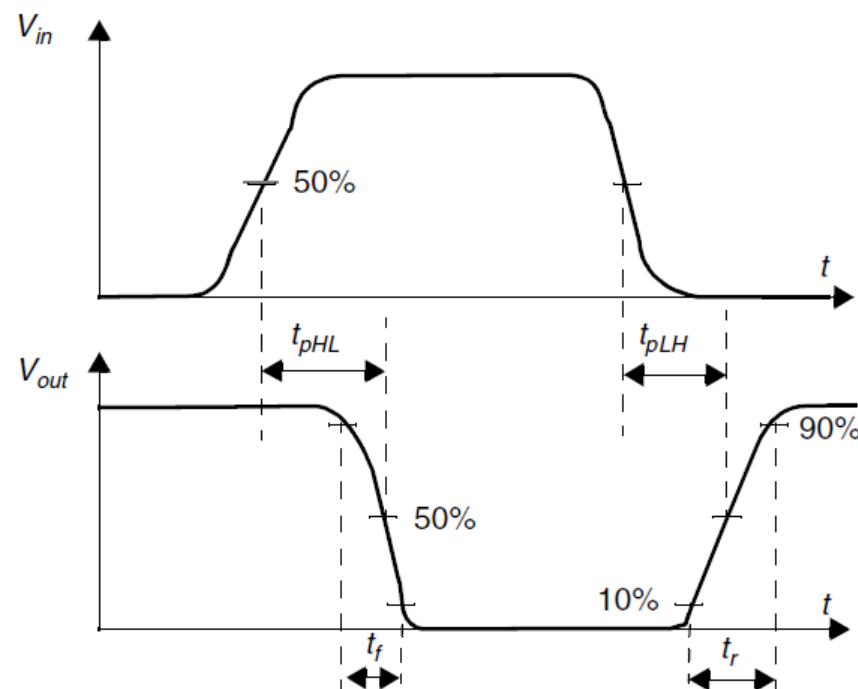
A CMOS inverter

- Egy n és egy p csatornás MOS tranzisztorból áll.
 - Állandósult állapotban a két tranzisztor közül csak az egyik vezet, a másik mindig lezár.
- Azaz, mint egy olyan kapcsoló, ami a kimenetre a bemeneti jel szintjétől függően vagy a tápfeszültséget, vagy a földet kapcsolja.



Késleltetés

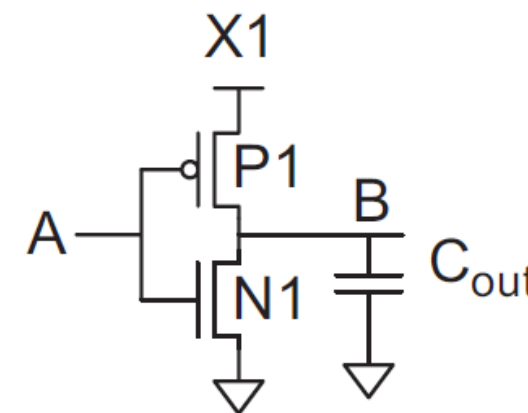
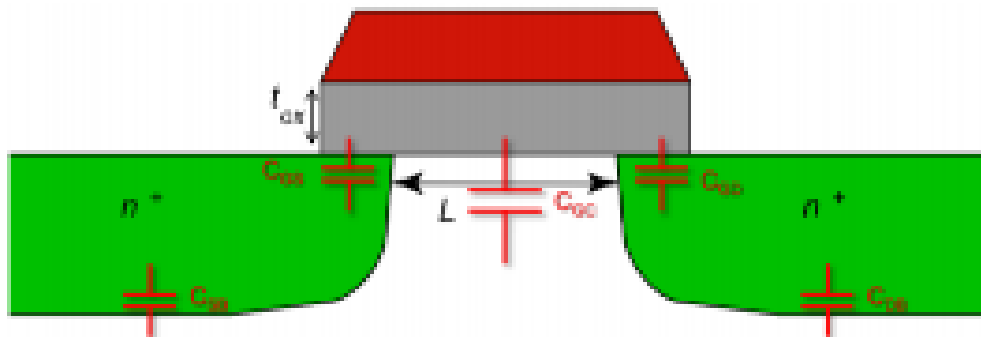
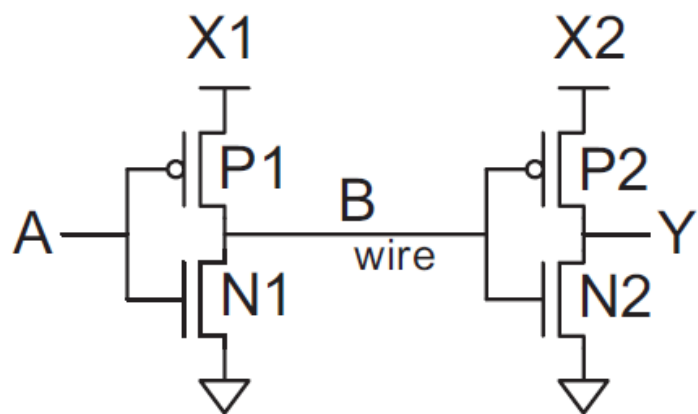
- Ha az idő függvényében vizsgáljuk az átkapcsolást, akkor látjuk, hogy a logikai kapu nem végtelen sebességű
 - Ennek oka a kapu nem tökéletes alkatrészekből épül fel, van egy belső késleltetés – (intrinsic delay)
 - A késleltetést az 50%-on mérjük, a fel és lefutást a pedig a 10%-90% között.



CMOS inverter (kapu) terhelése

■ A terhelés kapacitív

- A tranzisztorok belső kapacitásai (intrinsic kapacitás)
- A következő kapu bemenetének kapacitása
 - Ez látható a keresztmetszeti ábrán: a MOS tranzisztor gate-je mint egy síkkondenzátor
- **Az összekötő vezeték kapacitása – egy modern technológiában ez határozza meg a késleltetést leginkább.**



A késleltetés

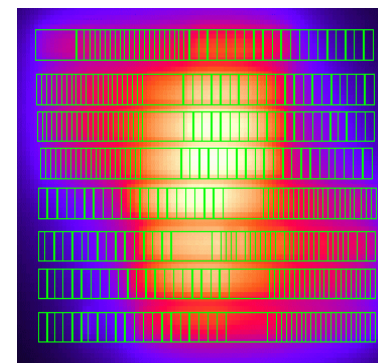
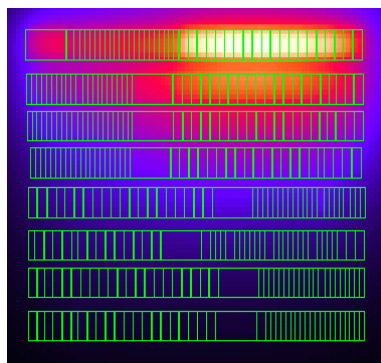
- Láttuk, hogy a késleltetést tulajdonképpen egy kapacitás töltése-kisütése határozza meg.
- Minél nagyobb a kapacitás, annál nagyobb a késleltetés (kb. arányosan)
- Ha a kapacitást nagyobb árammal töltjük, csökken a késleltetés
- **A tápfeszültség növelésével a késleltetés csökken**, mivel nagyobb árammal töltjük a kapacitásokat.
- **A hőmérséklet csökkentésével csökken a késleltetés**
- $t_{pd} \sim \frac{CV}{I}$
 - (a pontosság kedvéért: $Q = CV_{DD}$ de $I \sim V_{DD}^2$)
 - Pontosán így működik (működött) a „tuningolás”
 - A magfeszültség növelésével a mikroprocesszort a névlegesnél magasabb órajelfrekvencián lehet járatni.

Teljesítmény és energia

- **Vigyázat! Könnyű összekeverni, mert mindkettő szinonimájaként használjuk a „fogyasztás” szót (helytelenül)!**
- **Teljesítmény = egységnyi idő felvett energia. (Power)**
 - Mértékegysége a Watt (J/s)
 - Két dologra vagyunk kíváncsiak:
 - átlagos teljesítmény: $P_{av} = \frac{V_{DD}}{T} \int_0^T I(t) dt$
 - csúcsteljesítmény: $P_{peak} = V_{DD} I_{peak} = V_{DD} \max(I(t))$
 - ahol V_{DD} a kapu tápfeszültsége, I pedig az árama.
- **Két részre bontható**
 - Statikus fogyasztás – folyamatosan jelen van a kapu bekapcsolásától
 - Dinamikus fogyasztás – a kapcsolási események okozta fogyasztás. Függ a frekvenciától és a kapcsolás valószínűségétől.
- **Energia**
 - $E = \int P(t) dt$
 - Mértékegysége a Joule (kWh)

Teljesítmény

- A statikus fogyasztás alacsony, oka a szivárgási áram.
 - Modern áramkörökben már nem hanyagolható el...
- A dinamikus fogyasztás minden kapcsolási eseménynél fellép
 - Arányos az eseménysűrűséggel, amit
 - az órajelfrekvencia és az áramkör aktivitása határoz meg.



A dinamikus fogyasztás

- Két komponense van
 - Átkapcsolás: a bemeneti jel felfutó szakaszában mindkét tranzisztor egyszerre nyitott.
 - Töltéspumpálás
 - A dinamikus működés során a kimeneti kapacitást a jelváltáskor először tápfeszültségre töltjük.
 - Majd amikor logikai 0-ra vált a kimenet, kisütjük.
 - Azaz **szemléletesen** töltést pumpálunk a tápfeszültségből a föld irányába!
- **A fogyasztás legnagyobb részét a töltéspumpálás adja.**

A töltéspumpálás

- Tételezzük fel, hogy feltöltöttük a kimenetet!

- Ekkor a kapacitásban tárolt energia:

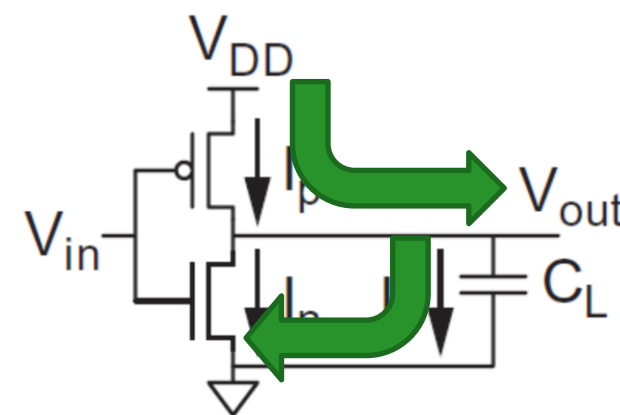
$$E_C = \frac{1}{2} C_L V_{DD}^2$$

- A tápfeszültség által szolgáltatott energia:

$$E = \int_0^{\infty} I(t) V_{DD} dt = \int_0^{\infty} C \frac{dV}{dt} V_{DD} dt = C V_{DD} \int_0^{V_{DD}} dV = C V_{DD}^2$$

Az energia fele a kapacitásba került, a másik fele „elveszett”, eldisszipálta a pMOS tranzisztor.

- Kisütéskor az nMOS tranzisztor fogja a töltést eltávolítani. Ilyenkor azonban a tápfeszültségből nem vesz fel energiát.



A töltéspumpálás inverter esetén

- Ha t idő alatt f_{sw} frekvenciával kapcsolgatunk, akkor a kapacitást $t f_{sw}$ alkalommal töltjük fel és sűtjük ki.
- Így a szükséges teljesítmény:

$$P = \frac{E}{t} = f C V_{DD}^2$$

Általános kapura:

- Ha a kapu kimenete p valószínűséggel változik
- Így a töltéspumpáláshoz szükséges teljesítmény:

$$P = p f C V_{DD}^2$$

Ez általánosságban is igaz. Azaz egy CMOS áramkör fogyasztása egyenesen arányos az órajelfrekvenciával és négyzetesen arányos a tápfeszültséggel!

$$P \sim f V_{DD}^2$$

A kimenet megváltozási valószínűsége

- Ha ismerjük a kapu bemenetének megváltozási valószínűségét, akkor az igazságtábla ismeretében kiszámíthatjuk a kimenet megváltozási valószínűségét
- Pl. kétbemenetű NOR kapu mindkét bemenete p valószínűséggel változik meg. Mekkora valószínűséggel változik meg a kimenet?
 - Megoldás: felsoroljuk az állapotokat és megnézzük, hogy az adott állapotból indulva mekkora a megváltozás valószínűsége:

A	B	Y	Kimenet változásának valószínűsége
0	0	1	Legalább az egyik megváltozik == nem marad mindkettő ugyanaz $1-(1-p)^2$
0	1	0	A marad, B 0-ra változik $(1-p)p$
1	0	0	B marad, A 0-ra változik $p(1-p)$
1	1	0	A és B is egyaránt 0-ra vált p^2

- $\frac{1}{4}$ súllyal összeadva: $P(\text{megváltozik})=p-p^2/2$

Energia (J, kWh, Ah, akkumulátoridő, Ft 😊)

- Energiahatékony működés nem csak azt jelenti, hogy kicsi a teljesítmény
- Pl. egy processzor esetén azt is kell vizsgálni, mennyi ideig tart egy adott taszk. Ha f frekvenciájú az órajel és a taszk N órajel alatt fut le, akkor

$$t_{task} = N/f$$

$$E = P \cdot t = P \cdot \frac{N}{f} = const \cdot f V_{DD}^2 \cdot \frac{N}{f} = const \cdot N V_{DD}^2$$

Azaz a felhasznált energia az **órajelek számával és tápfeszültség négyzetével** arányos. **Önmagában a frekvencia csökkentése energiahatékony működés szempontjából egy feldolgozás jellegű taszk esetén nem sokat ér.** A teljesítmény csökken, de felhasznált energia nem változik, mert a taszk arányosan tovább tart.
















A teljesítmény-késleltetés szorzat (PDP)

- A késleltetés és a teljesítmény **EGYSZERRE** jellemzik a digitális kaput.
 - A kapu sebessége ugyanis attól függ, milyen gyorsan lehet megváltoztatni a kapacitások energiáját.
 - Kis késleltetést csak nagyobb teljesítményfelvétellel lehet elérni.
- A késleltetés és a teljesítmény szorzata egy energia dimenziójú mennyiség
- Ez az ún. **Power-Delay product**
 - Szemléletesen azt mutatja meg, hogy egy bit feldolgozása mennyi energiát igényel.
 - (pontosabban egy kapcsolási esemény mennyi energiát igényel)
 - Ez a technológia **mérőszáma**

Dynamic Voltage Frequency Scaling/Enhanced SpeedStep stb.

- Az OS az igényeknek megfelelően változtatja a mikroprocesszor órajelfrekvenciáját és tápfeszültségét.
- Miért is?
 - Nagyobb órajelhez nagyobb tápfeszültség szükséges!
 - Csökkentett órajel esetén kevesebbrel is beéri.
 - A felvett teljesítmény viszont NÉGYZETESEN változik a tápfeszültséggel.
- Csak azt a magot kapcsoljuk nagyobb órajelfrekvenciára, amire valóban szükség van.
- big.LITTLE (DynamIQ)
 - Különböző teljesítményű magok.
 - (architektúra és/vagy technológia)
 - Mindig a tasknak megfelelő processzort választja ki a rendszer.

Energiatakarékossági módok, példa (i7)

	Active state			
	C0	C1	C3	C6
Core clock		off	off	off
PLL			off	off
Core caches			flushed	flushed
Shared cache				
Wakeup time*	active			
Core Idle power*				~ 0

- C1 – a mag órajelét kikapcsolja (halt)
 - A mag órajelét előállító egység (PLL) aktív. Utasítást a processzor nem hajt végre, de az első megszakítás felébreszti és rögtön végrehajtásra kész.
- C3 – a PLL-t is kikapcsolja, a mag cache-t kiüríti és lekapcsolja
 - A feléledés hosszabb időt vesz igénybe, az órajelet lokálisan előállító PLL-nek stabilizálódnia kell.
- C6 – a mag állapotát elmenti az ún. LLC (Last level cache)-be
 - A tápfeszültséget lekapcsolja. Így a mag nem fogyaszt.



Budapesti Műszaki és Gazdaságtudományi Egyetem
Elektronikus Eszközök Tanszéke

Digitális rendszertervezés

A digitális rendszerek bonyolultsága exponenciálisan növekszik

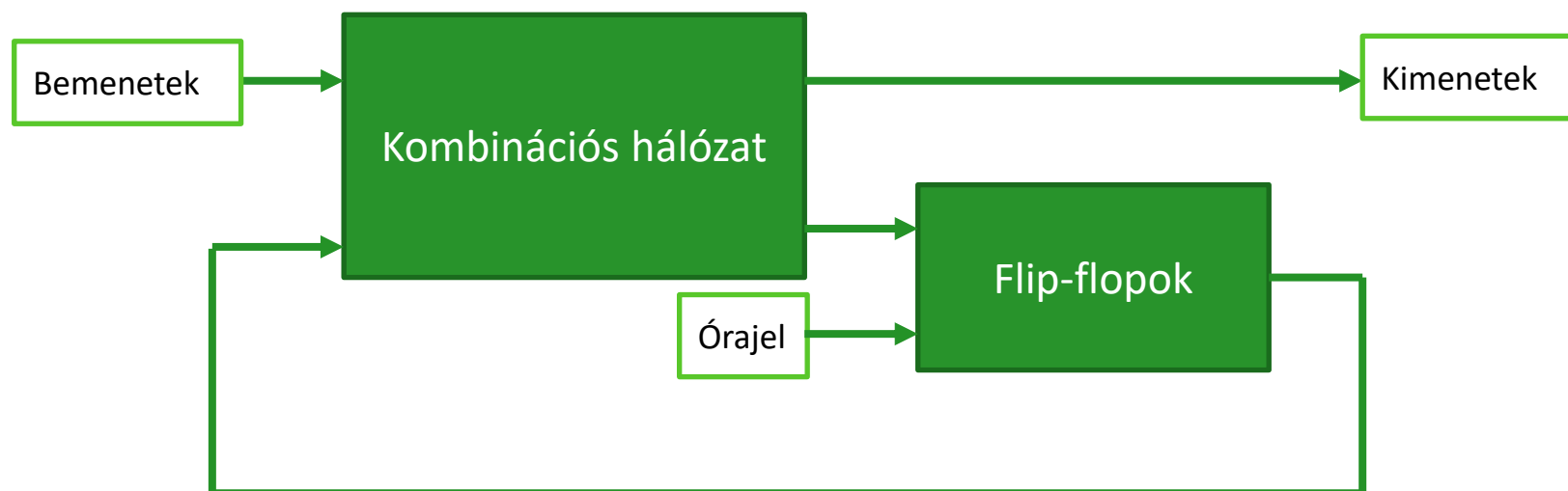
- Egyrészt a technológia lehetőségek, másrészt az igények...

A tervezési metódusoknak ezzel kell(ene) lépést tartani

- Ez csak úgy lehetséges, hogy egyrészt a tervezés egyre **magasabb absztrakciós szinten** történik, jóval a tranzisztorok, kapuk szintje felett
- Hasonlóan ahhoz, ahogy a magas szintű programnyelveket, keretrendszereket használjuk az informatikában
- A fizikai szintre történő leképezés **automatikusan**, de ember által felügyelt határok és **kényszerek** (design constrains) történik, 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?

Szinkron szekvenciális logika

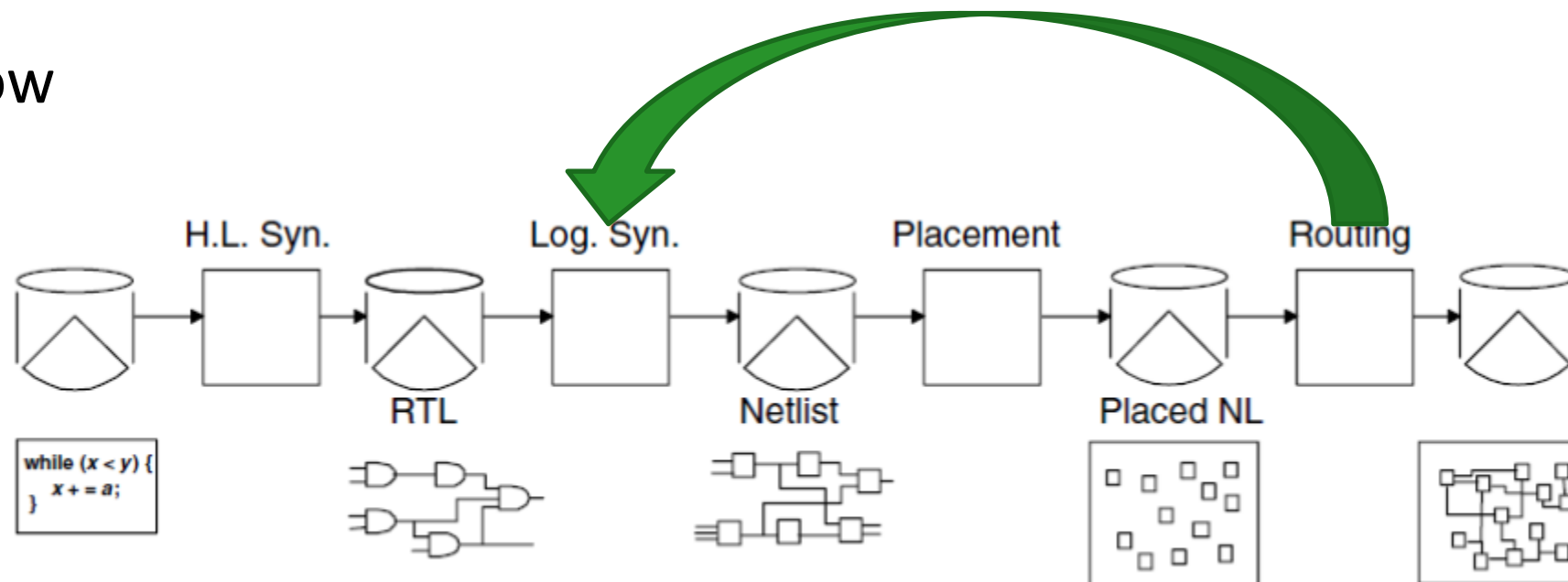
- Alapvetően ezt tervezzük.
 - Alapvető tároló elem a flip-flop, amelynek a kimenete az órajel aktív élére változik meg.
 - A változás az előző órajelben érvényes állapottól függ.



A tervezés folyamata (design flow)

- Szoftver modell elkészítése
 - Magas szinten definiáljuk a működést
- Hardver leírása ún. hardver leíró nyelven, az ún. magasszintű (high level) szintézis
 - Ez a szinkron logikát írja le, azaz hogyan keletkezik a kimenet a bemenetekből
 - Ember által vagy automatikusan
- Verifikáció (a hardver leíró modell tesztelése)
 - A tesztelés szót a kész áramkör tesztelésére használjuk. Itt azt ellenőrizzük, hogy a HDL modell működése megfelelő-e
- Logikai szintézis és szimuláció (automatikus)
 - Logikai kapuk szintjén a működés
- Fizikai szintézis (automatikus)
 - Place & route, azaz a kapuk elhelyezése és összekötése
 - Fizikai szimuláció (időzítések, energia stb.)

A design flow



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

Mintapélda: KITT

- Szoftver modell (algoritmus)

```

3 #####
4 class KITT:
5     def reset(self):
6         self.direction= False
7         self.state='_ '*7 + '*'
8     def clk(self):
9         if self.direction:
10            self.state= '_' + self.state[0:7]
11            if self.state[7]== '*':
12                self.direction= False
13        else:
14            self.state= self.state[1:8]+'_'
15            if self.state[0] == '*':
16                self.direction= True
17    def __init__(self):
18        self.reset()
19    def __str__(self):
20        return self.state
21 #####

```



A hardver leíró nyelv

- A digitális tervezés elméleti lépései (állapottábla, logikai kifejezések optimalizálása stb.) ismertek.
- Ez automatizálható és magasabb szinten definiálható
 - Megmondjuk, hogy a bemenetek és az aktuális állapot függvényében mi lesz a kimenet és mi lesz a következő állapot.
- Ez az ún. RTL – register transfer level – regiszter transzfer szint
- Erre szolgálnak a hardver leíró nyelvek (Hardware description language – HDL)
 - **VHDL** (*VHSIC Hardware Description Language*, IEEE STD 1987)
 - **SystemVerilog** (*VERification and LOGic*, IEEE STD 2005)

A koncepció

- Definiálni kell a rendszer be és kimeneteit
- Kombinációs hálózat modellezésénél egyszerűen leírjuk, hogy mi történik
 - Azaz hogyan kapjuk meg a kimeneteket a bemenetek függvényében!
 - Használjuk a szokásos logikai és aritmetikai kifejezéseket.
- Szinkron hálózat modellezése
 - Megadunk egy érzékenységi listát és leírjuk, hogy az érzékenységi listában szereplő változások esetén pontosan mi történik.
 - Szintén használhatjuk a szokásos logikai és aritmetikai kifejezéseket
 - Programozáshoz hasonló szerkezeteket (if-else, case stb.)
 - Az „utasítások” „párhuzamosan” hajtódnak végre! A kódban sorrend mindegy! Ez hardver.

KITT 😊

- HDL mintapélda



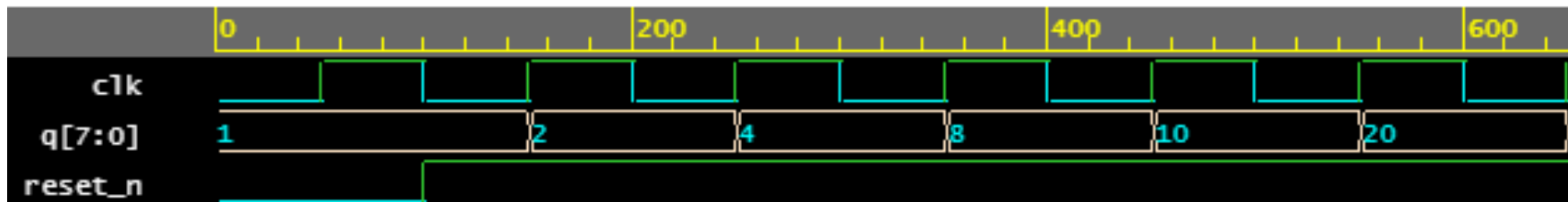
```

1  //////////////////////////////////////
2  module kitt(
3      input          clk,
4      input          reset_n,
5      output reg[7:0] cout
6  );
7
8      reg            direction;
9
10     always @(posedge clk or negedge reset_n)
11     begin: KITT
12         if (reset_n == 0)
13         begin
14             cout <= 8'b00000001;
15             direction <= 0;
16         end
17         else
18             if (direction == 0)
19             begin
20                 cout <= { cout[6:0], 1'b0};
21                 if (cout == 8'b01000000) // !!!
22                     direction <= 1;
23             end
24             else
25             begin
26                 cout <= {1'b0, cout[7:1]};
27                 if (cout == 8'b00000010)
28                     direction <= 0;
29             end
30     end
31 endmodule
32  //////////////////////////////////////

```


Logikai verifikáció

- Az RTL kód és a specifikáció összevetése
- Testbench készítése
 - Gerjesztések ráadása és a válasz megfigyelése
- Nem egyszerű. Ugyanazok az elvek, megoldások, problémák, mint a szoftver tesztelés esetén.
 - Kód lefedettség
 - Általában nagy az állapottér, nem lehet mindent kipróbálni
 - Alapvető funkcionális teszt, majd (irányított) véletlen tesztelés
 - A mintapéldára:



A logikai szintézis

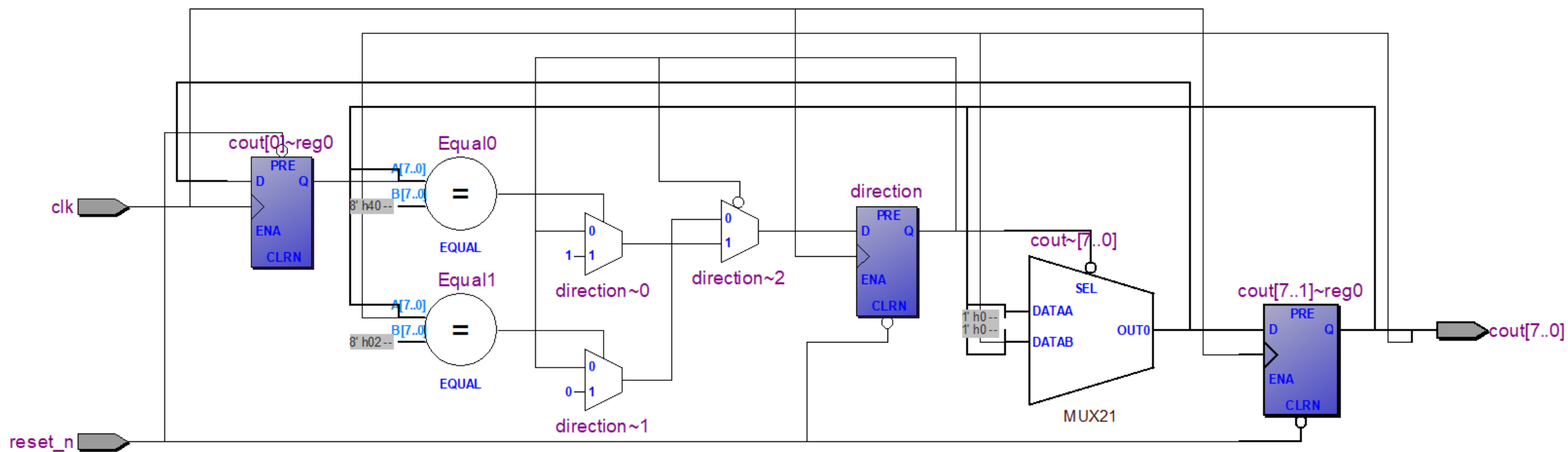
- **Bemenet**
 - RTL kód
 - A cellakönyvtár (a rendelkezésre álló kapuk) leírása (hdl nyelven)
 - A cellakönyvtár időzítési és teljesítményadatai
- **Kimenet**
 - Strukturális HDL, ami már csak a cellakönyvtárbeli elemeket tartalmazza.
- **Kényszerek:**
 - **Időzítés, terület, teljesítmény**
 - (alapesetben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

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, komparátor, multiplexer stb.
 - (Itt fontos, hogy a szintézer „ízlése szerint” kódoljuk az RTL-t.)
 - Ezeket megfelelő template-tel helyettesíti
 - Optimalizálja a logikai kifejezéseket
4. Leképezés makrocellára
 - Logikai kifejezések esetén a makrocella könyvtárból válogat megfelelő cellákat, amivel a kifejezést meg lehet valósítani.

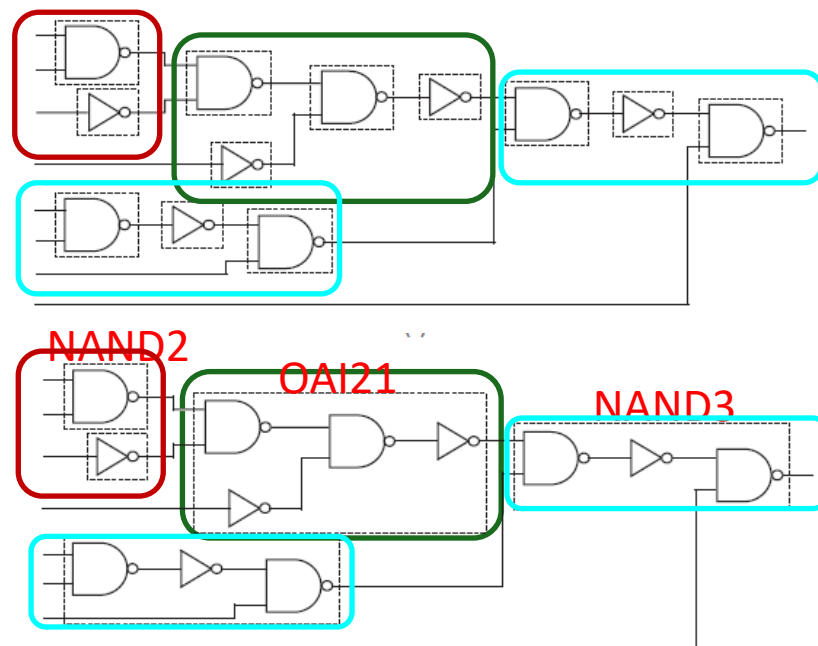
Mintapélda: generikus szintézis

- Felismer alapvető szerkezeteket.
 - Regiszterek, multiplexerek, komparátor stb.



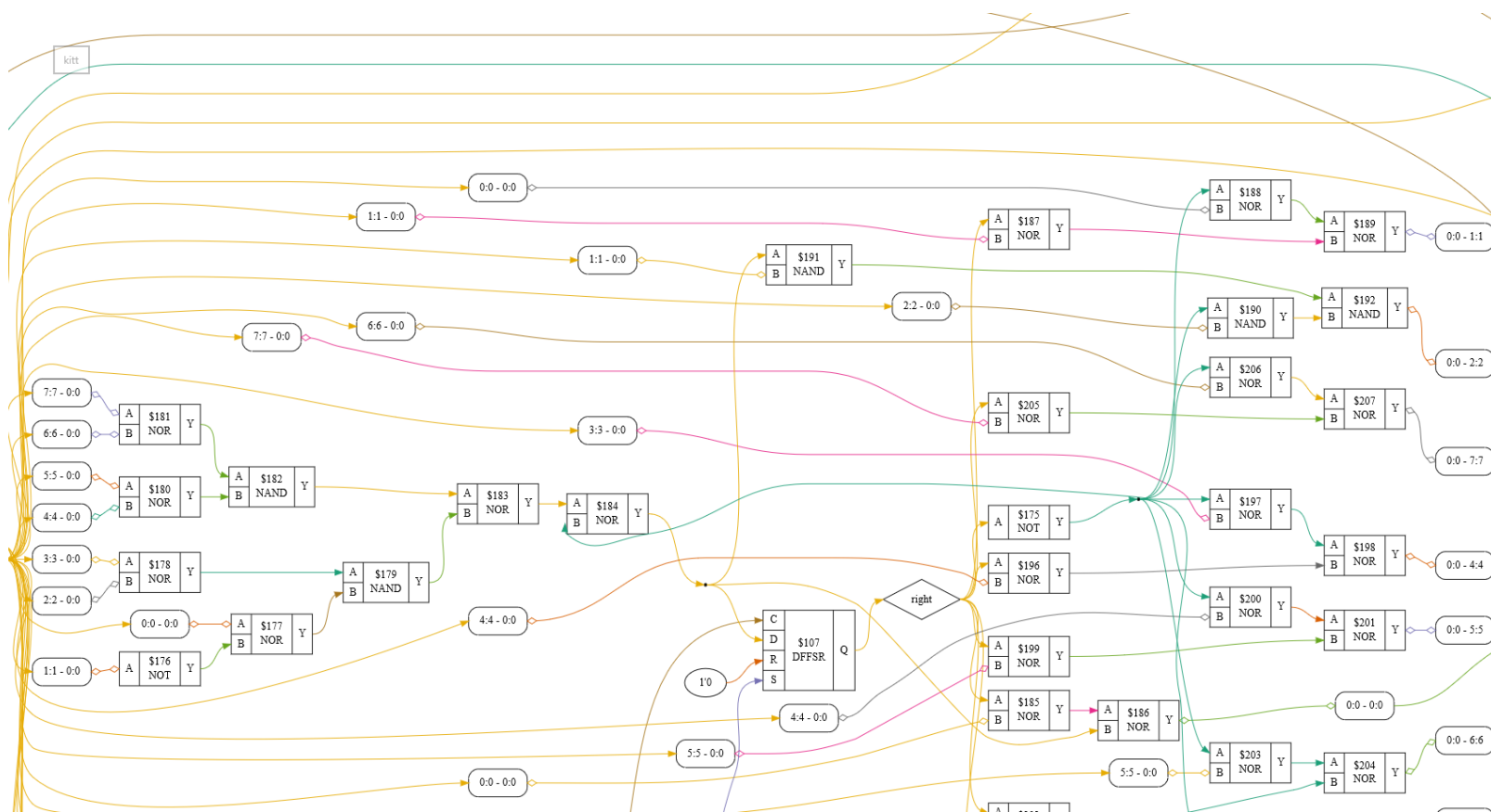
Leképezés makrocellára – vázlatos elv

- 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.



Mintapélda, logikai szintézis (részlet)

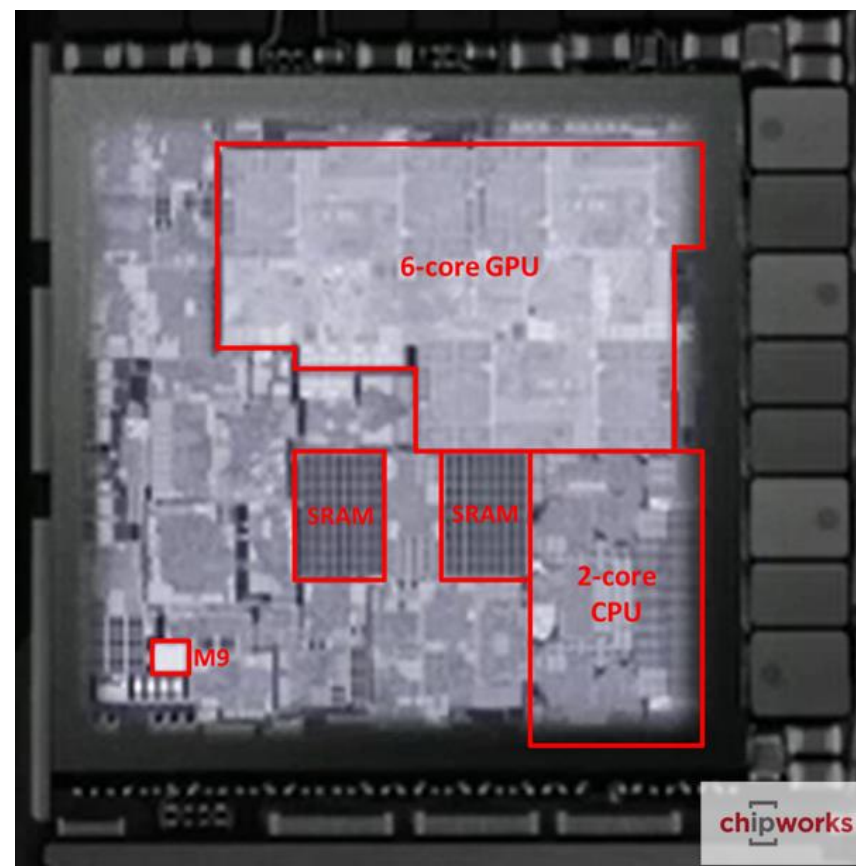
- Az ábra ember számára értelmezhetetlen már egy ilyen egyszerű terv esetén is. Ez már technológiafüggő leképezés...



Kapu	Felhasznált darab
INV	10
NAND	23
NOR	9
BUF	9
DFF	9

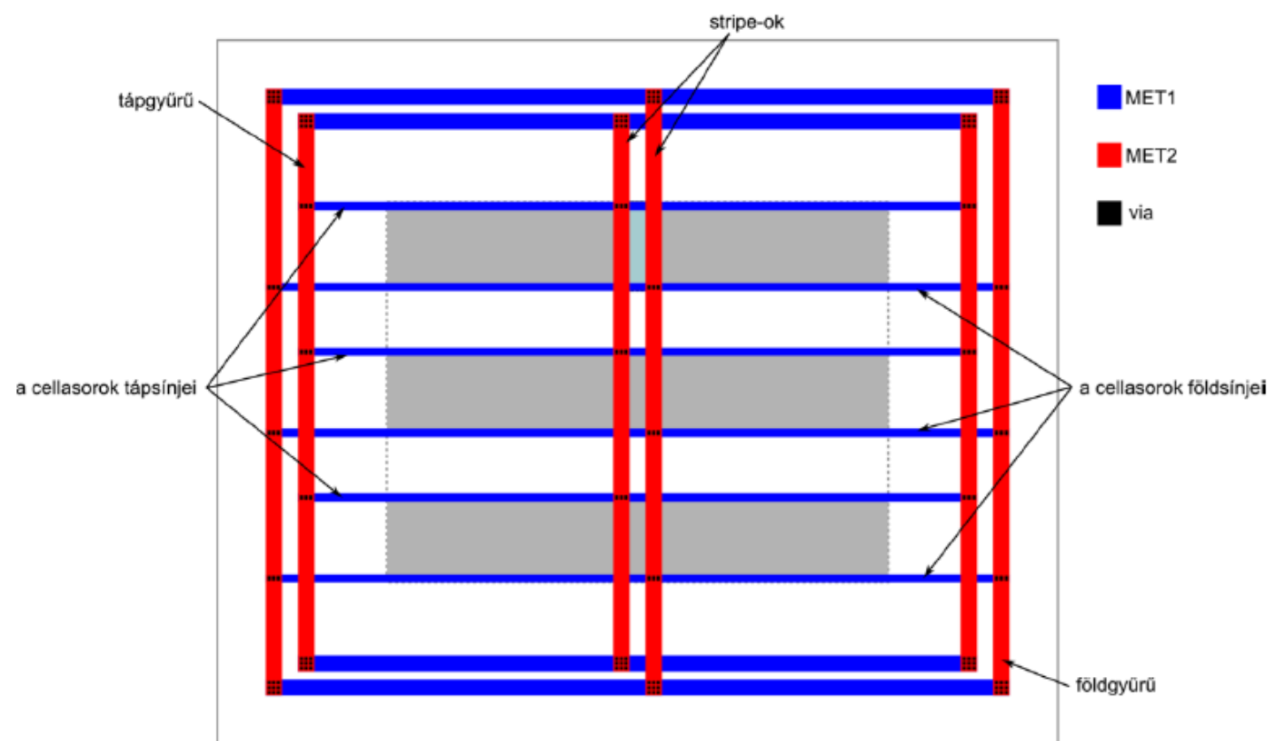
1. Floorplan

- 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)



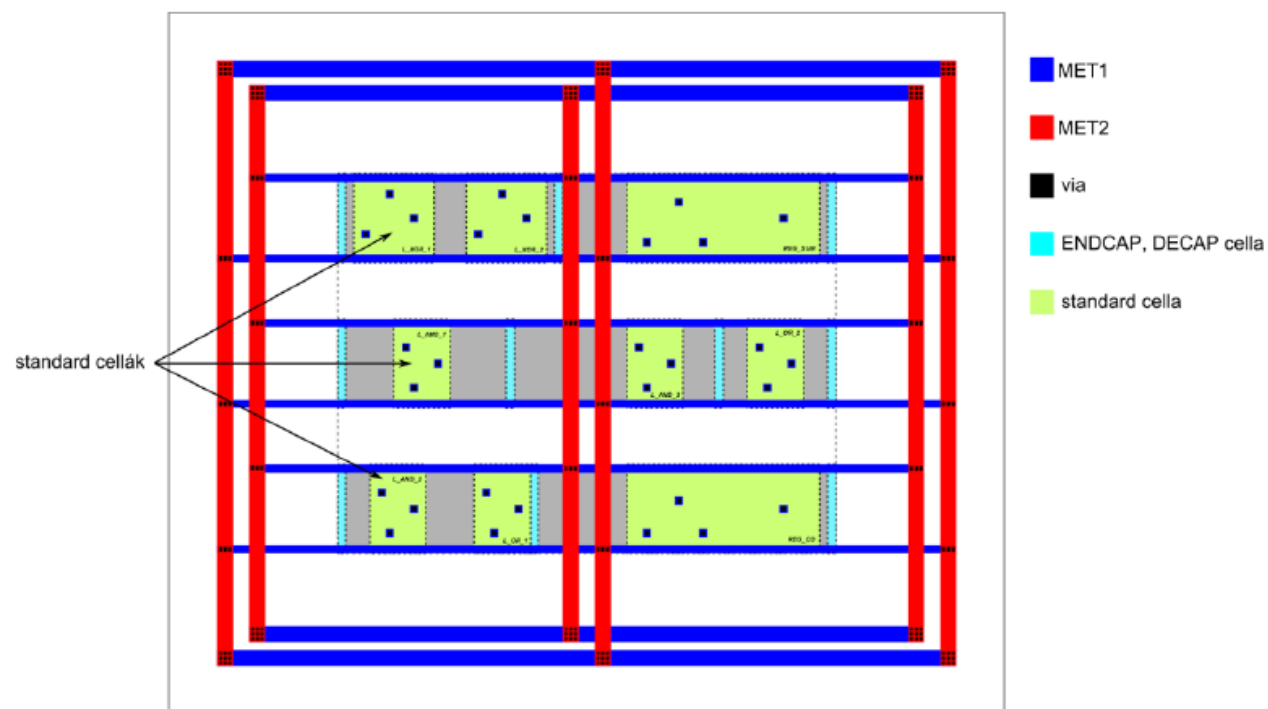
2. Tápellátás (power plan)

- 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%



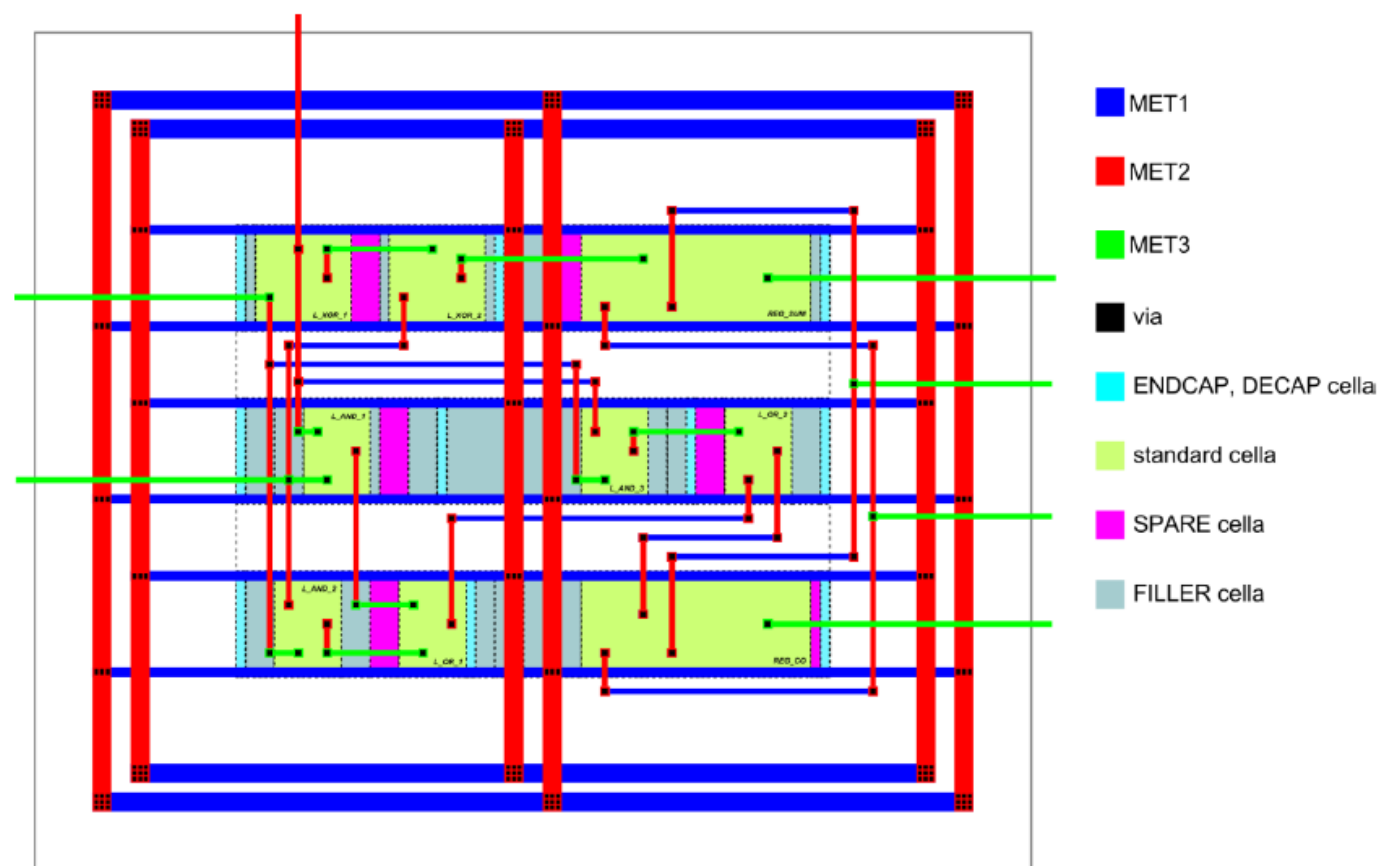
3. A cellák elhelyezése (place)

- A logikai funkciót megvalósító standard cellák elhelyezése

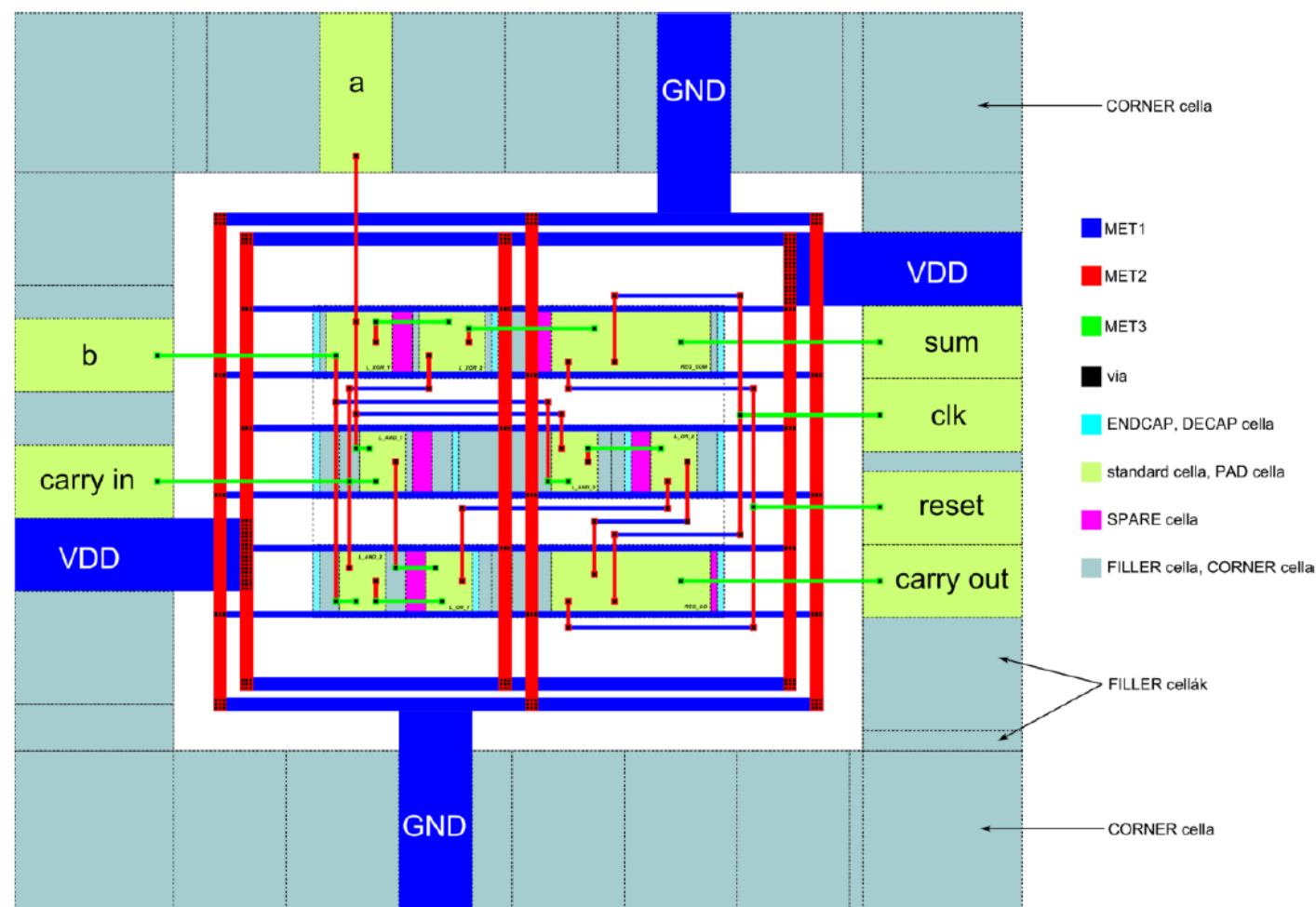


4. Huzalozás (route)

- 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 kell ellenőrizni a tervet.
 - (és újratekinteni, ha az időzítés nem megfelelő...)

Források, ajánlott irodalom, érdekességek

1. [Intel Turbo Boost](#)
2. [Intel Extreme Tuning Utility](#)
3. [Intel Performance Monitor](#)
4. [EDA playground](#)
5. [MOS Technology 6502 processzor visszafejtve és rétegszintű! szimulációja](#)