Budapesti Műszaki és Gazdaságtudományi Egyetem
Elektronikus Eszközök Tanszéke

IT eszközök technológiája

3. előadás

Logikai kapuk megvalósítása

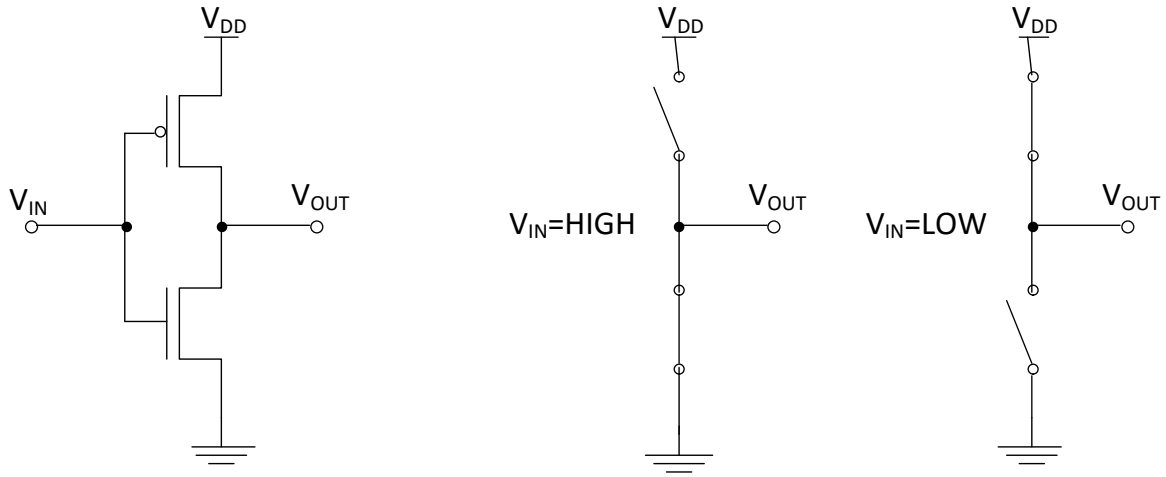
A transzfer kapu

Tárolók (Regiszterek)

Nagysebességű CMOS logika

Aritmetika

Emlékeztető: A CMOS inverter

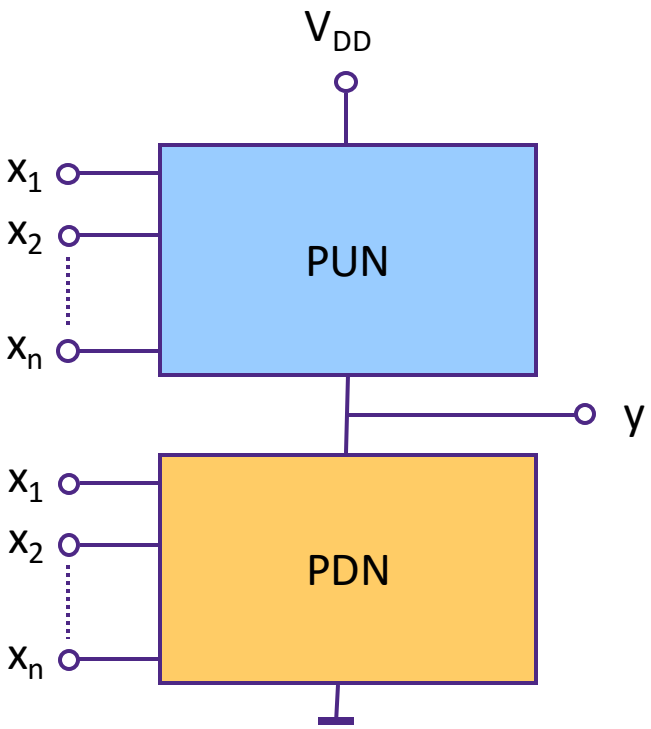


- Egy n és egy p csatornás MOS tranzisztorból áll.
  - Állandósult állapotban a két tranzisztor közül csak az egyik vezet, a másik mindig lezár.
- Azaz, mint egy olyan kapcsoló, ami a kimenetre a bemeneti jel szintjétől függően vagy a tápfeszültséget, vagy a földet kapcsolja.
  - Ha tranzisztor helyett egy hálózatot használunk, ami a bemenetektől függően vagy tápfeszültséget, vagy földet kapcsol -> elkészítettük a logikai kapukat.

CMOS kapuk

- A kapuk esetében egy p csatornás tranzisztorokból álló „pull up” (PUN) ill. n csatornás tranzisztorokból álló „pull down” hálózat (PDN) alkotja a kaput, mindkét hálózat annyi tranzisztorból áll, ahány bemenete van a függvénynek.
  - Azoknál a bemeneti kombinációknál, ahol a függvény értéke 0, a PDN rövidzár a kimenet és a föld között, míg a PUN szakadás a kimenet és a táp között.
  - Ha a függvény értéke 1, akkor a PDN szakadás, a PUN rövidzár.
  - duális alkatrészek: NMOS helyett PMOS
  - De Morgan azonosságok alapján ez könnyen belátható
  - ÉS jellegű kapcsolatot két tranzisztor(vagy hálózatrész) soros, VAGY jellegű kapcsolatot két tranzisztor (vagy hálózatrész) párhuzamos kapcsolása ad.

Összefoglalva

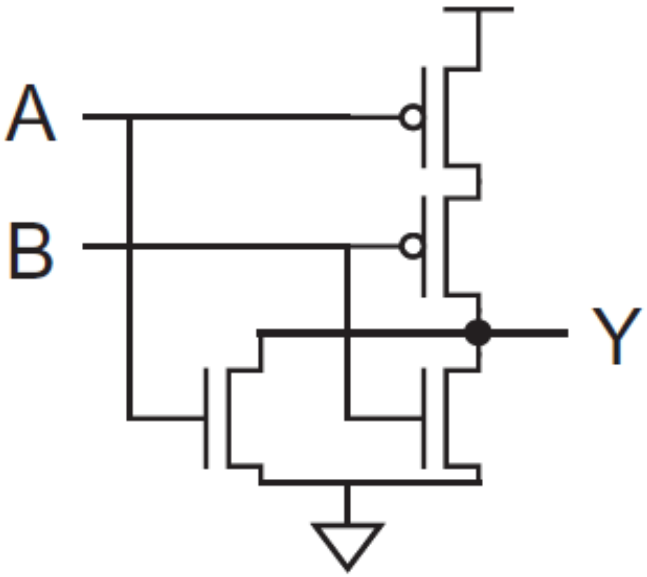


$$y = f(\mathbf{X})$$

- pull up network
  - p-csatornás tranzisztorok
  - rövidzár, ha f(**X**)=1
  - szakadás, ha f(**X**)=0
- pull down network
  - n-csatornás tranzisztorok
  - rövidzár, ha f(**X**)=0
  - szakadás, ha f(**X**)=1

CMOS NOR kapu

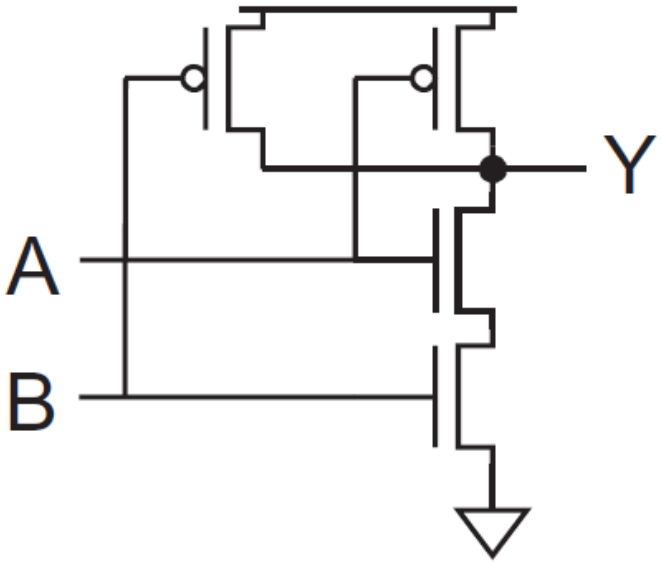
- A pull down network két párhuzamosan kötött tranzisztorból áll
- Ha A vagy B bemenet igaz, valamelyik nMOS tranzisztor vezet, a hozzátartozó pMOS viszont lezár, így a kimenet 0 lesz.



$$Y = \overline{A + B} = \bar{A} \cdot \bar{B}$$

CMOS NAND kapu

- A pull-down network most két, sorba kötött nMOS tranzisztorból áll.



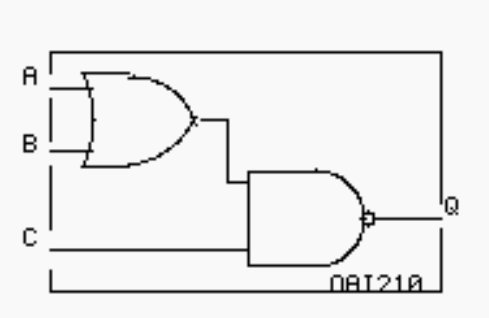
$$Y = \overline{AB} = \bar{A} + \bar{B}$$

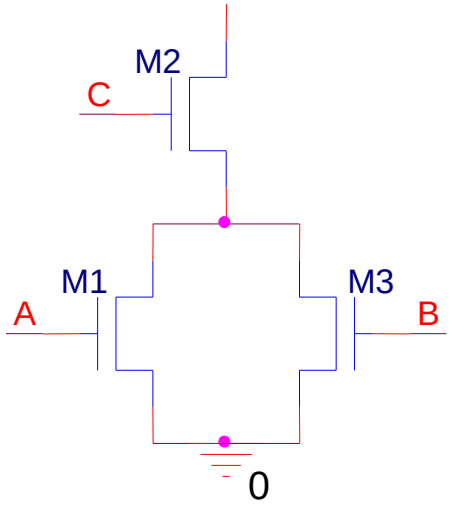
Komplex kapuk

- Tranzisztor szinten tudunk bonyolultabb (nem alapvető) logikai függvényeket megvalósítani.
- Általában maximum 4 bemenettel, az és ill. vagy függvényeket kombináljuk.
- Például:
  - OAI21
    - $Y = \overline{(A + B)C}$
  - AOI22
    - $Y = \overline{AB + CD}$
- Általában n bemenet és/vagy kombinációja *2n* tranzisztor segítségével megvalósítható.
  - Ha feltételezzük, hogy a bemenet ponált és negált változata is rendelkezésre áll
- Mivel egy kapu, a késleltetés a többszintű realizációhoz képest kedvezőbb.

Példa

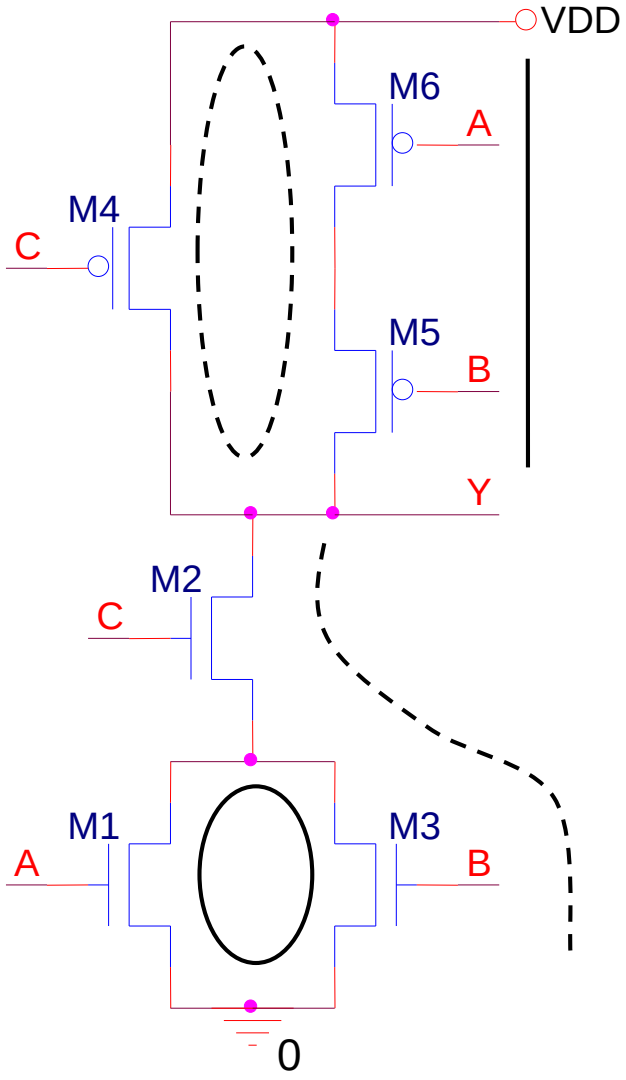
- Tervezzük meg az $Y = \overline{(A + B)C}$ függvényt megvalósító komplex kaput!
  - Első lépés a PDN (pull-down network) megtervezése, ezt n csatornás tranzisztorokkal kell megvalósítani.
    A negált logikai függvény minden 0 értékéhez a kimenet és föld között áramutat kell biztosítani, a függvényben szereplő összegnek párhuzamosan, a szorzatoknak sorba kapcsolt hálózatrészek felelnek meg.
  - Itt tartunk:





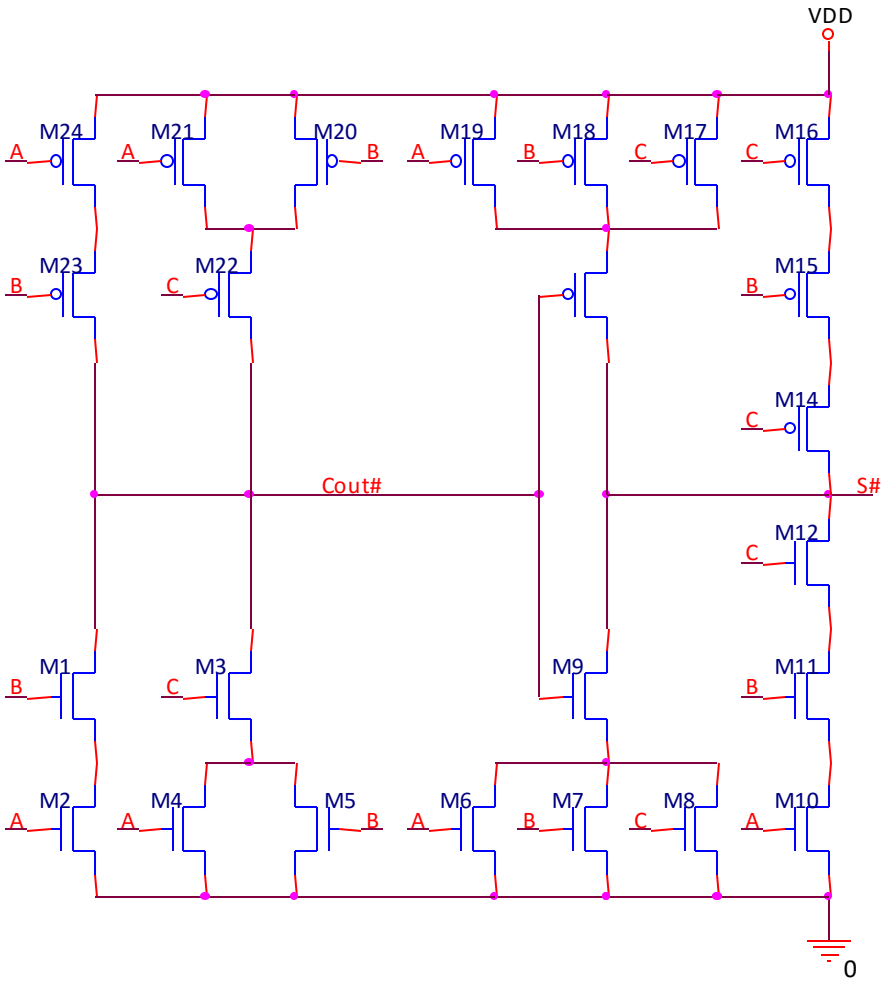
Példa, folytatás

- A második lépés PUN (pull-up network) megtervezése, ezt a p vezetéses tranzisztorokkal kell megvalósítani.
  - A pull-up network a negált függvény minden 1 értékéhez a tápfeszültség és a föld között áramutat kell, hogy biztosítson.
- $Y = \overline{C(A+B)} = \bar{C} + \overline{A+B} = \bar{C} + \bar{A}\,\bar{B}$



Teljes összeadó (full adder)

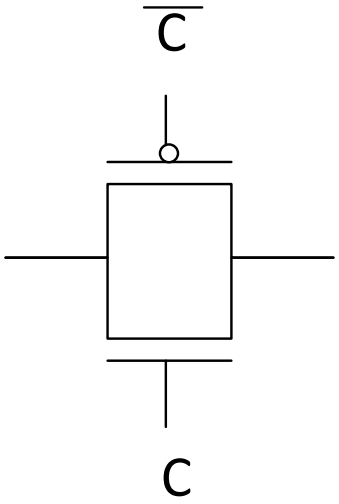
- Minden aritmetikai áramkör alapja
  - Összeadó, számláló, komparátor stb.
  - A teljes összeadónál – ha több bites számokat adunk össze – a kritikus út a carry lesz.
- Ezt kihasználva a carry-t felhasználjuk az összeg képzésekor:
  - $C_{OUT} = AB + C(A + B)$
  - $S = ABC + (A + B + C)\overline{C_{OUT}}$
  - 24 tranzisztor + a 4 a két inverter.
- A kapcsolás teljesen szimmetrikus
- Először a carry készül el, majd utána az összeg.



A CMOS transzfer kapu

CMOS transzfer kapu

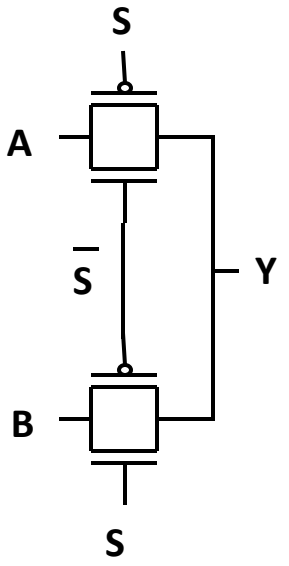
- A jelfolyam útjába helyezett kapcsoló.
- CMOS kivitelben egy n és egy p típusú tranzisztort kapcsolnak össze, a vezérlő jelek egymás inverzei.
- Transzfer kapuk alkalmazásával tovább egyszerűsíthetők az áramkörök.



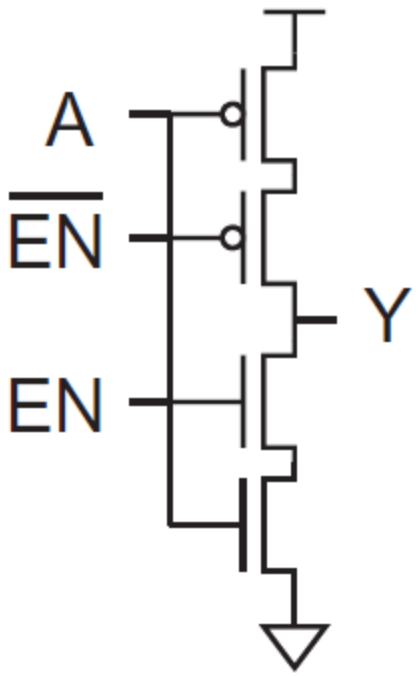
C=0 -> szakadás
C=1 -> rövidzár

Transzfer kapuk alkalmazása

- Bizonyos funkciók transzfer kapuk alkalmazásával jóval egyszerűbben, kevesebb tranzisztorral valósíthatóak meg.
- Tipikusan a kiválasztó jellegű funkciók ilyenek.
  - Sok függvény ide vezet.
  - Programozható logikák egy részében a kiválasztás így történik. (ld. FPGA)
- Tekintsünk például egy kétbemenetű multiplexert!
  - $Y = A\bar{S} + BS$
- Komplex kapuval 8 tranzisztor szükséges, a transzfer kapus pedig csak 4 tranzisztorból áll!



Órajel vezérelt CMOS (Clocked CMOS, C²MOS)



- Háromállapotú (tri-state kapu)
- EN=0 – a kimenet lebeg
- EN=1 – a kimenet a bemenet inverze.

CMOS tárolók

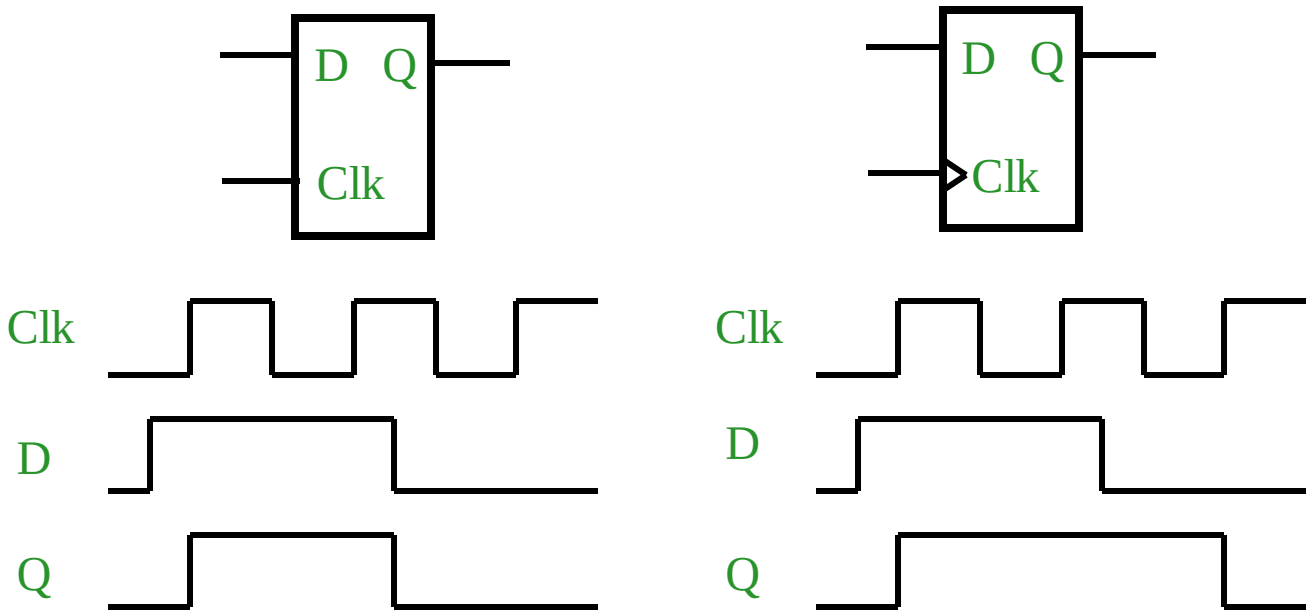
Latch

Flip-flop

regiszter

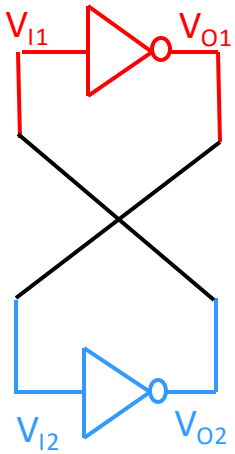
Latch és flip-flop

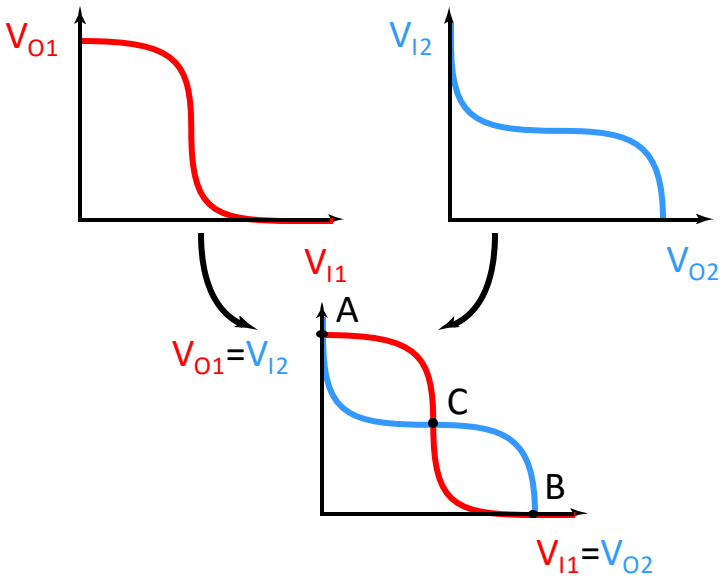
- Sok helyen ekvivalens a két fogalom. Ebben az anyagban így használjuk:
  - latch: az engedélyezett latch átlátszó, a bemeneti változás (késleltetés után a kimenetre jut), tehát SZINTVEZÉRELT
  - flip-flop: a beírás az órajel fel vagy lefutó élére történik, tehát ÉLVEZÉRELT



A tárolás alapelve

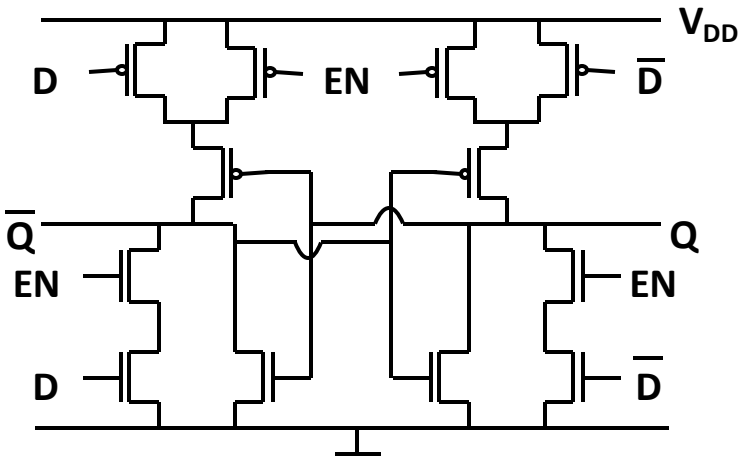
- Két állapotú (bistabil) áramkörök alapja: két, gyűrűbe kapcsolt inverter
  - Ennek az elrendezésnek két stabil állapota van (A, B)
    - (ezen kívül egy metastabil, (C) ekkor minden feszültség a komparálási feszültség, de az inverter karakterisztika gondoskodik róla, hogy ebben az állapotban ne maradjon meg – ezt csak szimulátorban lehet előállítani.)
- Ahhoz, hogy tárolóként lehessen használni, írhatóvá kell tenni.
  - Az invertálási funkciót megtartva az inverterek helyett negált kimenetű logikai kapukat használunk, amelyeknek a további bemenetei lehetővé teszik a beállítást.





Tárolók felépítése

- Természetesen van lehetőség kapukból felépíteni a tárolókat, de ez tranzisztorszám szempontjából nem lesz kedvező.
  - Pl. SR latch engedélyező bemenettel felépíthető négy kétbemenetű NAND kapuból
- Kihasználjuk a CMOS áramkörökben könnyű komplex kaput készíteni (12 tranzisztor)
  - Két AOI (and-or-invert, $Y = \overline{AB + C}$ kapuból pl. lehet D-latchet készíteni.

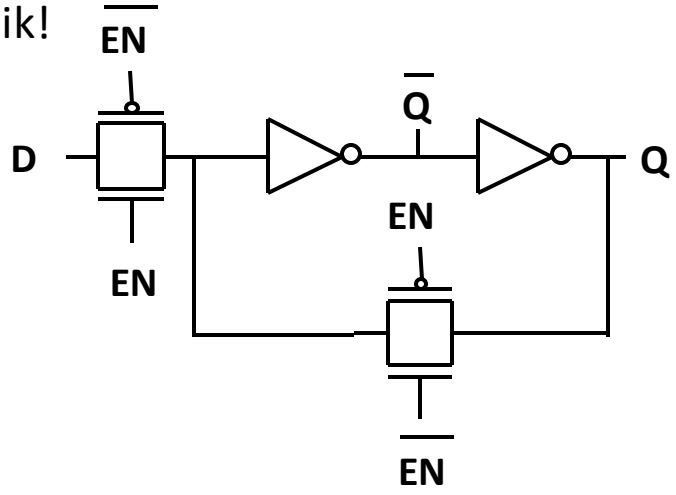


$$Q = \overline{EN \cdot \bar{D} + \bar{Q}}$$

$$\bar{Q} = \overline{EN \cdot D + Q}$$

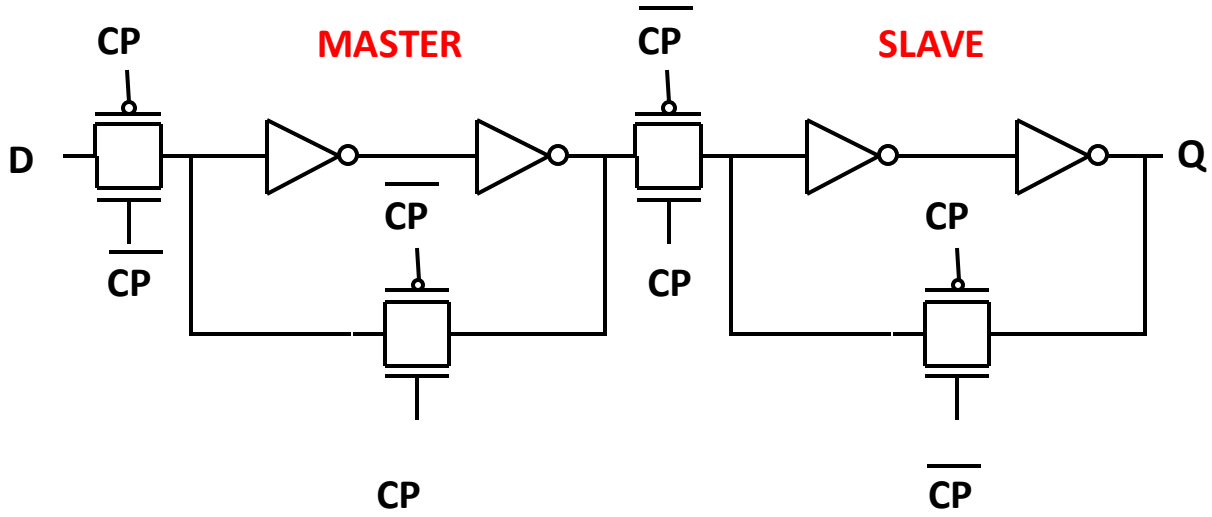
D-latch transzfer kapuval

- A beírás és a visszacsatoló ág egy-egy transzfer kapuval van ellenfázisban vezérelve.
  - EN=1 alatt transzparens működés, Q = D, mivel a beíró ág transzfer kapuja vezet, a visszacsatoló ágban elhelyezett transzfer gate viszont zárt.
  - EN=0 alatt a kimenet visszaíródik a bemenetre, a transzfer kapuk most ellentétesen vezetnek: a visszacsatoló ág „él”, a beíró ág elzárt.
  - Összesen nyolc tranzisztorral megvalósítható és nincs szükség a D negáltjára.
  - Figyeljük meg, hogy a bemenetről „átengendő” és a visszacsatoló transzfer kapu ellenfázisban működik!



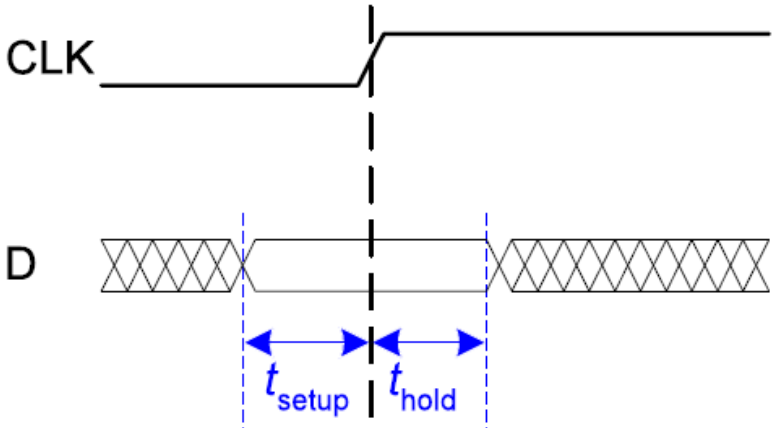
D-flipflop

- A master-slave flip-flop két sorbakötött, ellenütemű órajellel vezérelt latch
  - CP alacsony szintjén az első tároló átlátszó.
  - CP felfutó élére a master nemátlátszó lesz és a tartalom a slave-be íródik



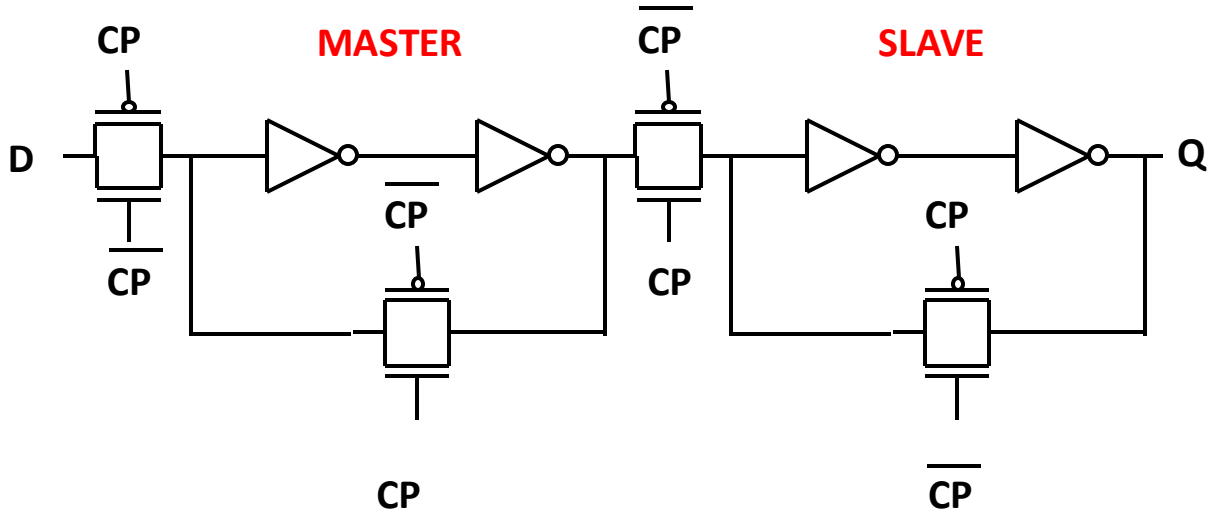
Flip-flop-ok időzítése

- Setup time: az órajel aktív éle előtt a mintavételezett adatnak stabilnak kell már lennie.
- Hold time: az órajel aktív éle után ennyi ideig nem szabad megváltoznia.



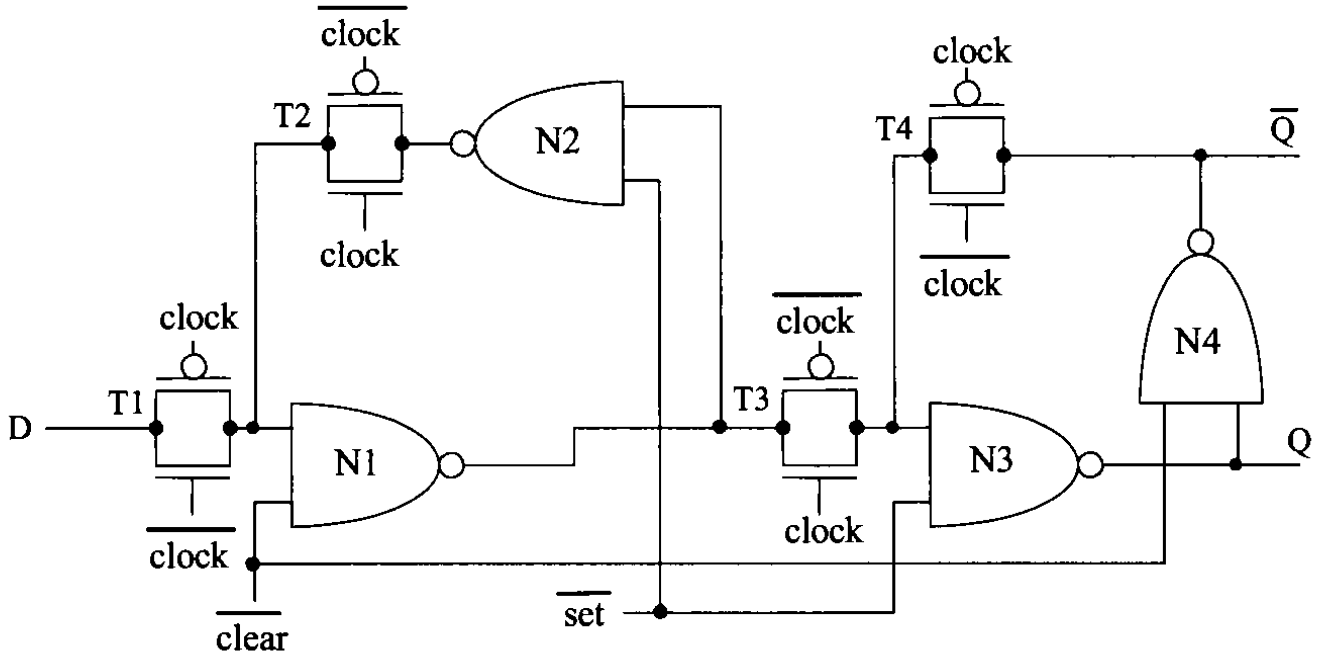
D-flipflop időzítései

- $t_{SETUP} > t_{PGTG} + 2t_{PGINV}$
  - Egy transzfer kapun és két inverteren kell keresztülhaladnia a bemenetnek, mire a mintavétel megtörténik.
  - Azaz az órajel felfutó éle előtt már stabilnak kell lennie.
- $t_{HOLD} > t_{PGTG}$
  - Amíg a MASTER transzfer gate-je elzáródik, addig nem szabad, hogy a bemenet megváltozzon



D flip-flop aszinkron clear és set-tel.

- A gyűrűbe kapcsolt inverter helyett (az invertáló funkciót megtartva) kétbemenetű kapukat használunk.
- Például:



Budapesti Műszaki és Gazdaságtudományi Egyetem
Elektronikus Eszközök Tanszéke

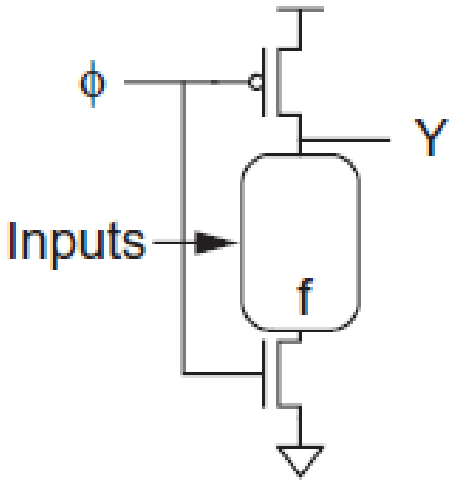
Nagysebességű CMOS logika

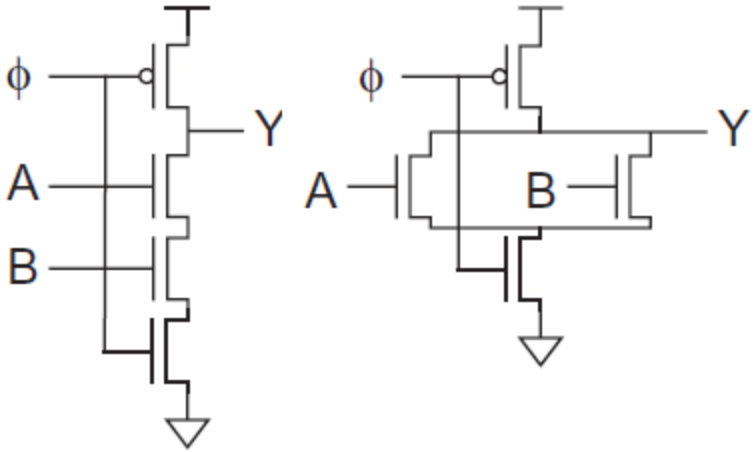
Áramköri lehetőségek a késleltetés csökkentésére

Nagysebességű CMOS logika

- A statikus CMOS logika késleltetése
  - $t_{pd} \sim \frac{CV}{I}$
- Technológia gyorsítási lehetőségek:
  - A kapacitások csökkentése
  - Az áram növelése (úgy, hogy a méret csökken!)
- Áramköri lehetőségek:
  - A logikai szint távolság (swing) csökkentése és differenciális logika
    - Két feszültségszint különbségének előjele adja a logikai értéket.
    - SCL – ún. source-coupled logic. Csak megemlítjük.
  - Kihasználjuk a mindenütt jelenlévő *szórt kapacitást* logikai szint *ideiglenes tárolására*
    - Ezáltal területet nyerünk és kapacitást csökkentünk, tehát gyorsabb lesz
    - CMOS domino logika, dinamikus tárolók

CMOS dominó logika

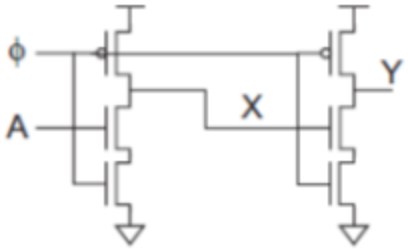


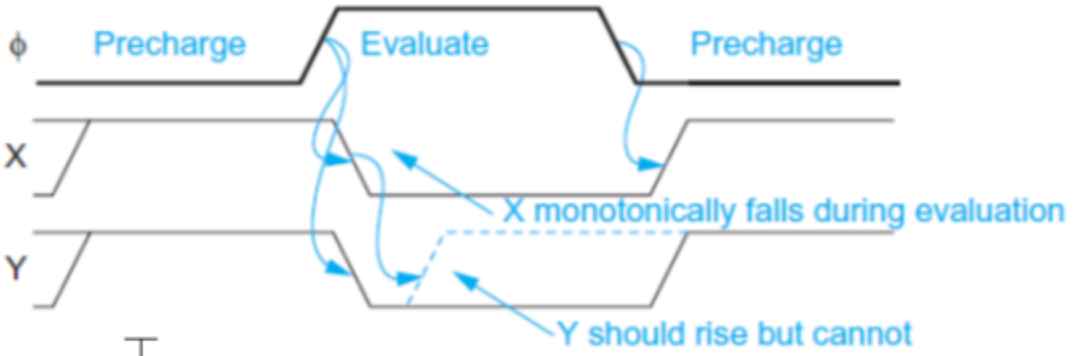


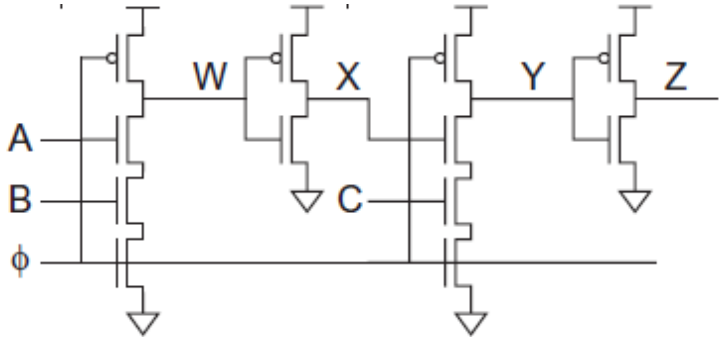
- A kimenetet terhelő szórt kapacitást előtöltjük. (precharge)
- Φ=0: előtöltés tápfeszültségre.
  - Ilyenkor pMOS nyitott, nMOS zárt, a pMOS tranzisztor előtölti a kimenetet logikai 1-re
- Φ=1: kiértékelés
  - A pull-down network a bemenetek állapotától függően vagy eltávolítja a szórt kapacitás töltését, vagy meghagyja
  - Előnyök:
    - N bemenetű függvényhez N+2 tranzisztor szükséges
    - Azaz gyorsabb, hiszen az előző fokozatot kisebb kapacitás terheli.

CMOS dominó logika

- A kapuk összekapcsolásakor vigyázni kell!
  - A kiértékelési fázis elején, ha függvény értéke 0 kellene, hogy legyen, egy hamis logikai 1 van a kimeneten egy rövid ideig.
  - Ez a következő kapu tranzisztorát kinyithatja, így a kapacitás töltése sérül.
  - Csak egy inverteren keresztül köthetjük a következő kaputt. (a hamis logikai 0 gondot nem okoz.

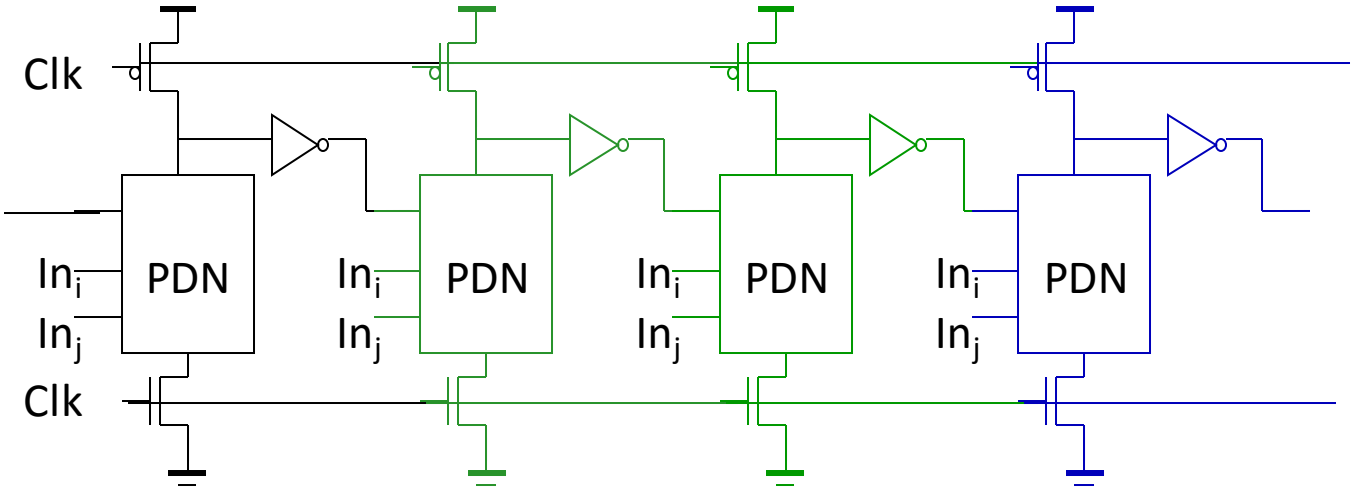






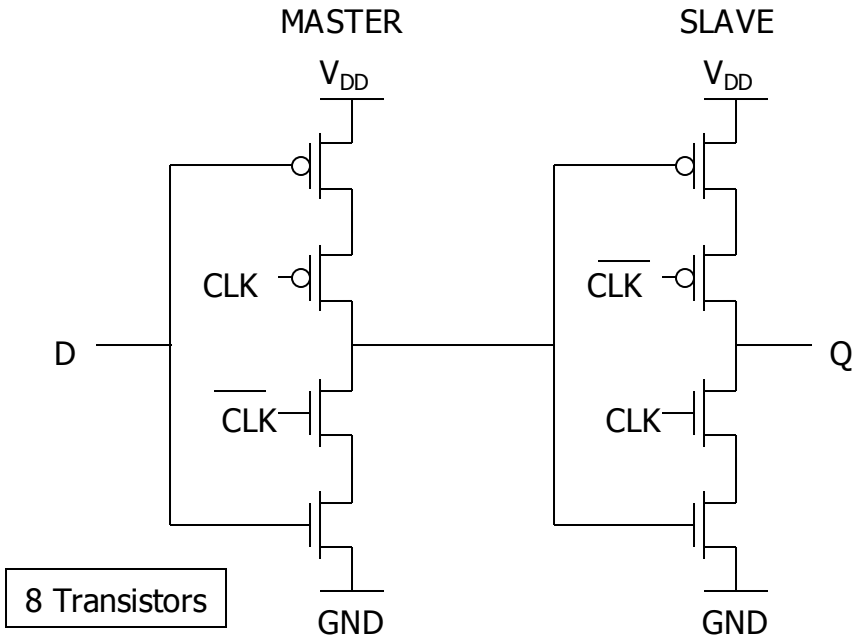
CMOS dominó logika

- Kb. 1,5× gyorsabb, mint a statikus CMOS



Dinamikus D flip-flop

- Két órajel vezérelt invertert kapcsolunk össze, amelyeket ellentétesen vezérlünk.
- Az információ a master és a slave közötti szórt kapacitásban tárolódik

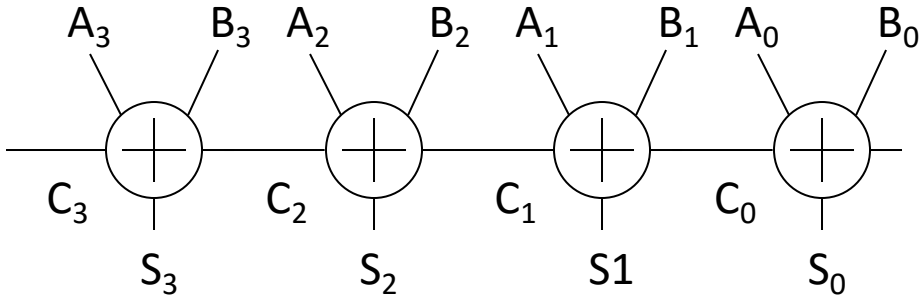


Az adatút elemei

- Összeadók
  - Ripple carry adder
  - Összeadó architektúrák
    - Carry Skip
    - Carry Look-Ahead
    - Carry Select
- Funnel shifter
- Kombinációs szorzó

Ripple-carry adder

- Több bites számok összeadására gyakorlatilag sorbakapcsolunk teljes összeadókat.
- A művelethez szükséges idő: $n \cdot t_{pdcarry}$
- A kritikus út az átvitel terjedése
  - CMOS váltott logikával készül, így az inverterek megspórolhatók.
  - (páros ponált, páratlan negált logika, vagy fordítva, ez mindegy…)
  - Könnyen huzalozható, de egy bizonyos bitszám felett túl lassú lesz



Összeadás gyorsítása

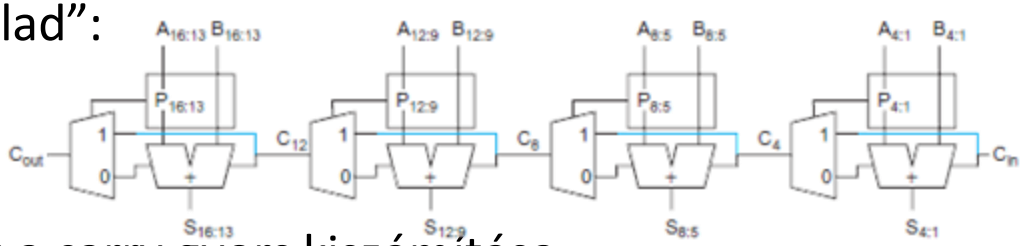
- A kritikus út az átvitel terjedési ideje – ezt kell gyorsítani!
- Definiáljuk a következő jeleket
  - *generate* – a fokozat mindenféleképpen átvitelt generál
  - *propage* – a fokozaton az előző fokozat átvitele áthalad
  - egy bitre például:
    - $G = AB$ illetve $P = A \oplus B$
  - ezeket a jeleket egyszerre nagyobb bitszámra rekurzívan számíthatjuk
    - $G_{i:0} = G_{i:k} + P_{i:k}G_{k-1:0}$ illetve $P_{i:0} = P_{i:k}P_{k-1:0}$
    - Pl. 8 bitre, 4 bites fokozatokat tekintve:

$$G_{7:0} = G_{7:4} + P_{7:4}G_{3:0} \text{ illetve } P_{7:0} = P_{7:4}P_{3:0}$$

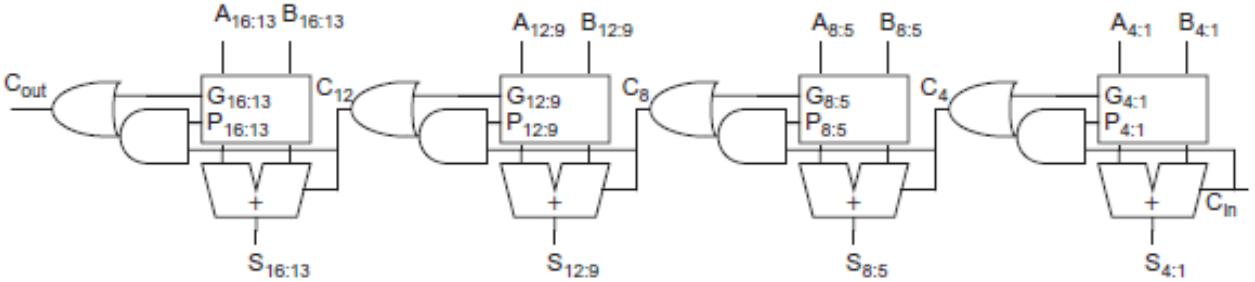
    - Azaz egy fokozat akkor generál átvitelt, ha vagy önmaga generált, vagy az előző fokozat generált és az áthaladt.

Carry-Skip, Carry Look Ahead, Carry Select

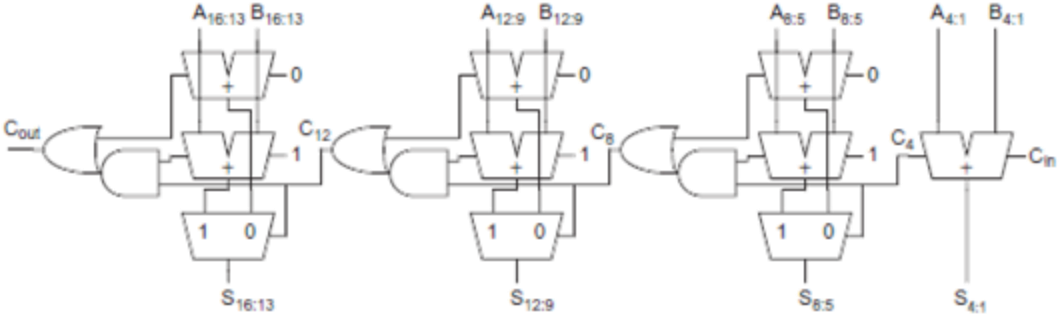
- CSA: fokozatonként történik a propagate kiszámítása és egy multiplexer választja ki, így a carry gyorsan „áthalad”:



- CLA: fokozatonként történik a carry gyors kiszámítása:



- CSEL: fokozatonként kiszámolja az összeget átvitel és átvitel nélkül, amiből az előző fokozat átvitele választja ki a végeredményt egy multiplexer segítségével

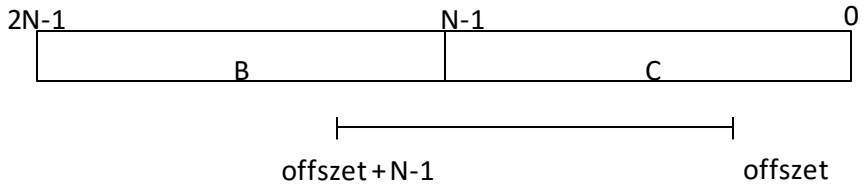


Összeadás

- A bonyolultabb összeadó struktúrák egyre nagyobb területet és teljesítményt igényelnek.
- Nincs optimális architektúra, mert minden technológia függő
  - Pl. hány bites fokozatokat érdemes készíteni, hogyan lehet összekötni stb.
  - Más lesz az optimum különböző technológiákon.
  - Az igazán nagysebességű architektúrák kézzel készülnek.
- A logikai szintézis feladata, hogy olyan megfelelő architektúrát találjon, ami még épp teljesíti az időzítési követelményeket.
  - ez általában kézzel felülírható

Eltolás (Shift)

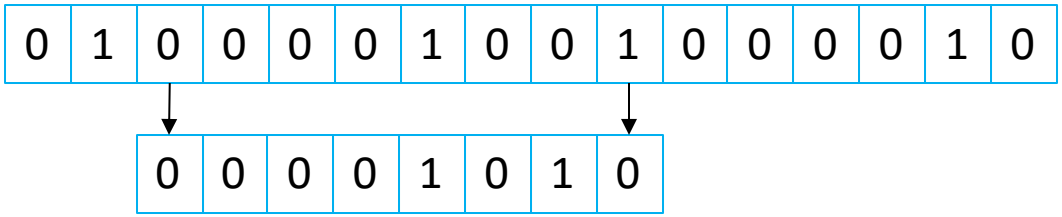
- Konstanssal történő eltolás
  - Ez a legegyszerűbb, hiszen ez huzalozási kérdés.
- Funnel-shifter
  - Az N bites bemenetet 2N bitesre egészítik ki, művelettől függően, majd ebből választanak ki N bitet egy megadott offszettel.
  - Ezzel minden szokásos művelet elvégezhető



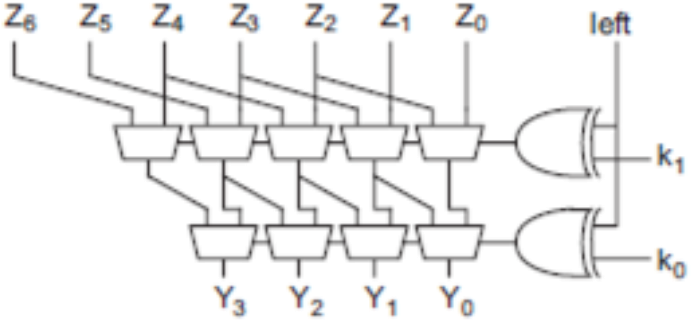
Eltolási művelet	B	C	Offszet
unsigned >> k	0..0	$A_{N-1}..A_0$	k
signed >> k	$A_{N-1}..A_{N-1}$	$A_{N-1}..A_0$	k
<<	$A_{N-1}..A_0$	0	N-k
rotate right	$A_{N-1}..A_0$	$A_{N-1}..A_0$	k
rotate left	$A_{N-1}..A_0$	$A_{N-1}..A_0$	N-k

Az eltolás - példa

- Pl. 8 biten forgassuk el balra kettővel a 0x42-t



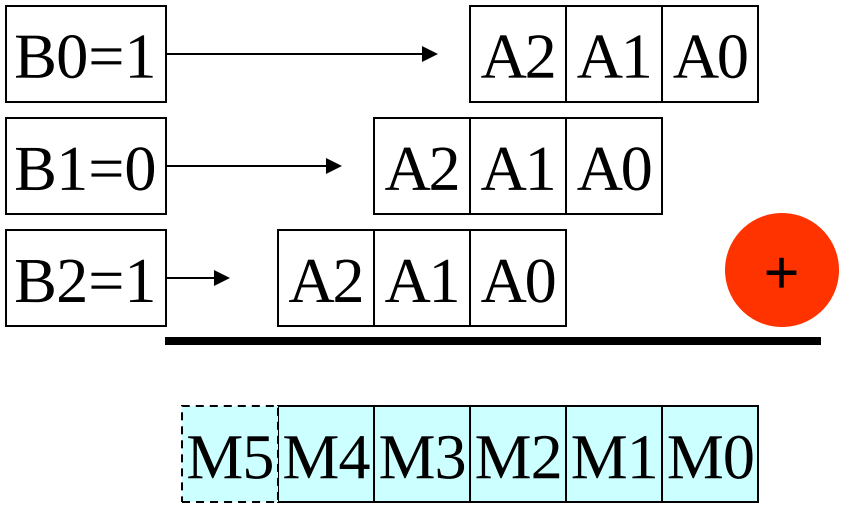
- Multiplexerek sorozatával oldható meg. Pl. 4 bitre:



- Az első sor multiplexer 2-vel végzi a durva kiválasztást, majd a 2. sor pedig végzi el a finom kiválasztást.
  - Az N-k kivonás (valójában inverz) itt be van „építve”, a xor kapu invertál
  - Ha a 0-val történő eltolást tiltjuk, Z7 már nem is kell.
  - N bithez log2 N multiplexer sor kellene, de lehet 4-1 vagy 8-1 multiplexereket is alkalmazni.
    - Az, hogy melyiket éri meg, az technológiafüggő… (huzalozás…)

Szorzók

- Kombinációs szorzó
- Az összeszorzandó számok A(n-1:0) és B(n-1:0)
- Ha B adott bitje 1, bekerül az összegbe, egyébként nem.



Kombinációs szorzó áramköri elrendezése

- Részösszegek előállítása + összeadása a helyiérték figyelembe vételével. (figyeljük meg az átvitel trükkös bekötéseit!)