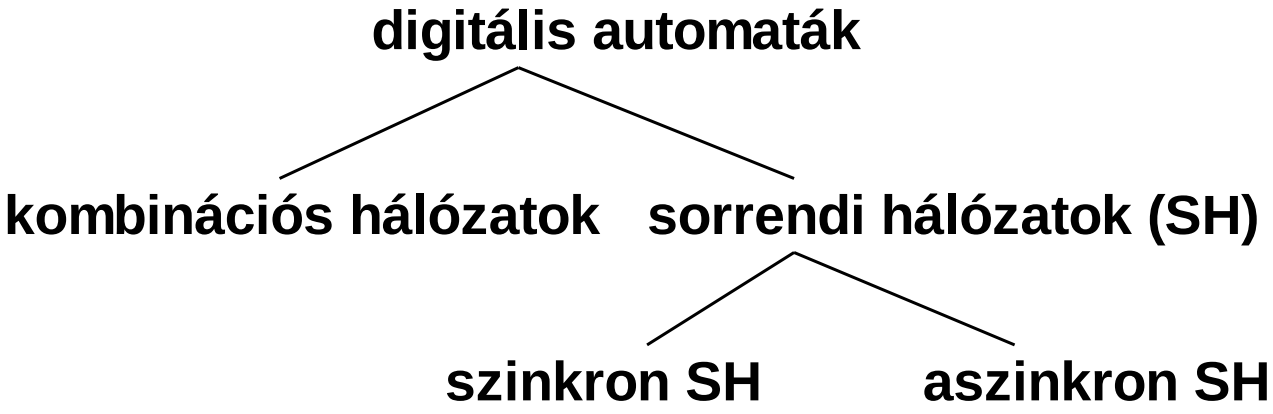
Kiegészítő segédlet szinkron sorrendi hálózatok tervezéséhez

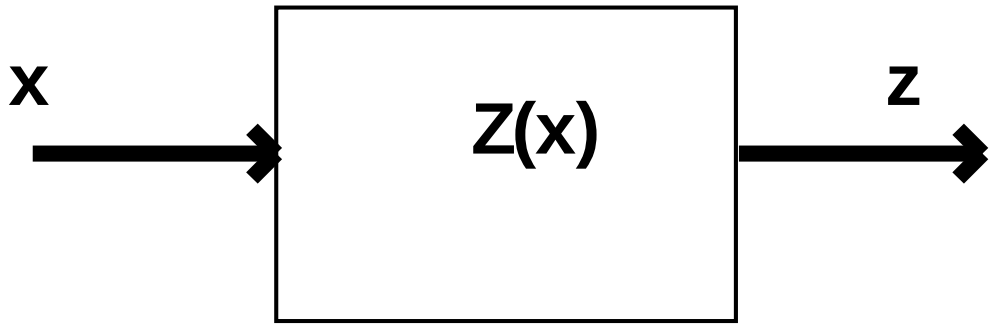
Benesóczky Zoltán 2019



Kombinációs automata

Logikai függvényt valósít meg. A kimenete csak az aktuális bemeneti kombinációtól függ.

$$z=Z(x)$$



Megvalósítása kombinációs hálózattal történik.

Sorrendi automata

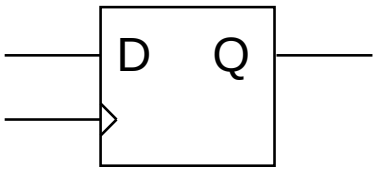
A kimenet az ***előző bemenetektől*** is függ.

- ***Emlékező*** tulajdonsága van, melyet az ***állapotok***kal (state, s) reprezentálunk. Az állapota megváltozhat, ha új bemenetet érzékel.
- A ***következő állapot*** (next state, next_s) az aktuális állapot és az aktuális bemenet függvénye.

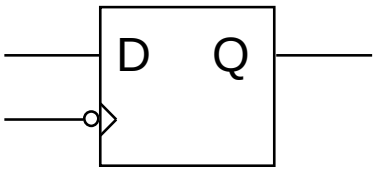
$$next\_s=S(s,x),$$

- **A szinkron sorrendi hálózat (SSH)** ***aktuális állapotát (az állapot kódját) memória tulajdonságú alkatrész, flip-flopokból álló állapot regiszter tárolja, órajellel (clk) ütemezett időpontokban, az órajel aktív élénél.*** (Ez vagy felfutó él (0-1 átmenet), vagy lefutó él (1-0 átmenet) lehet. A példákban felfutó élet használunk.)

Legegyszerűbb állapottároló a ***D flip-flop***.



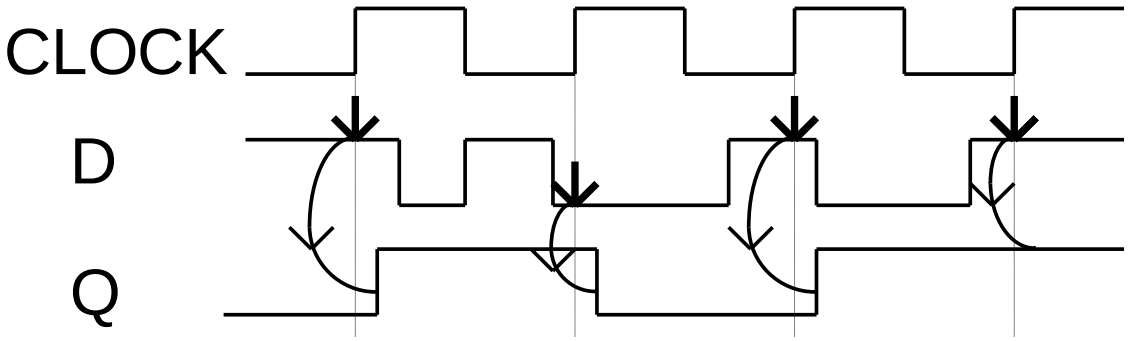
felfutó él érzékeny
D flip-flop



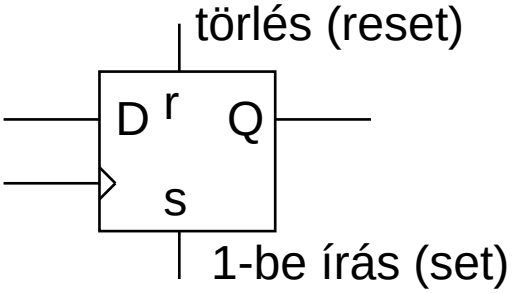
lefutó él érzékeny
D flip-flop

Az órajel aktív élére tárolja a D bemeneten levő értéket. Q = D

A működést szemléltető idődiagram felfutó él érzékeny flip-flop esetén:

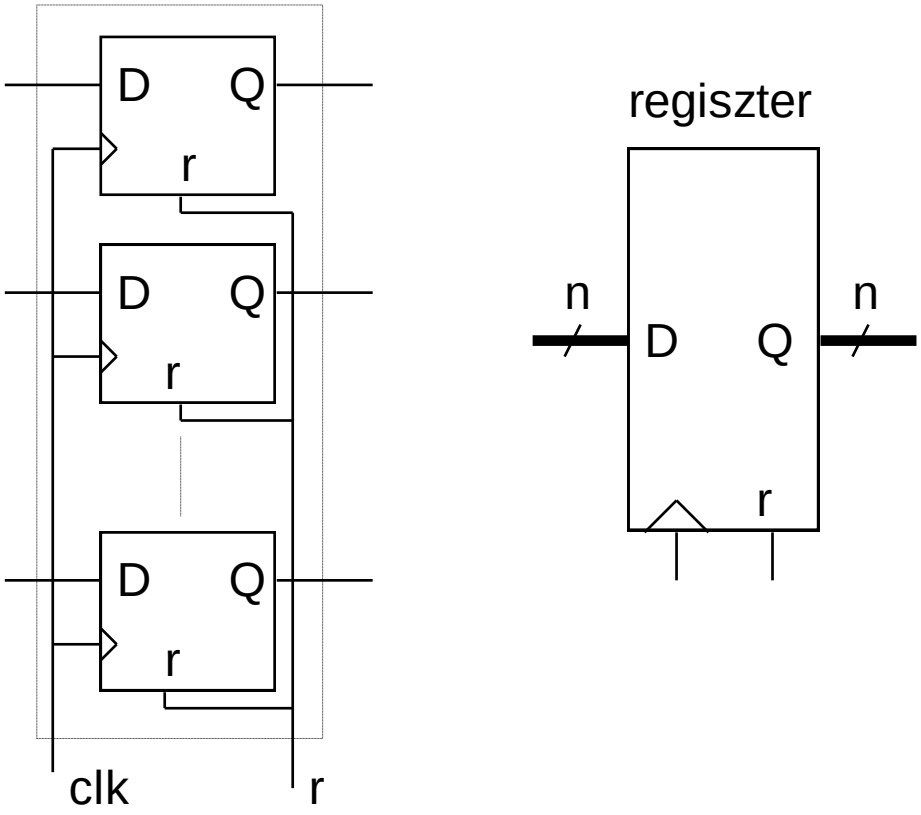


Reset és preset bemenete is lehet:

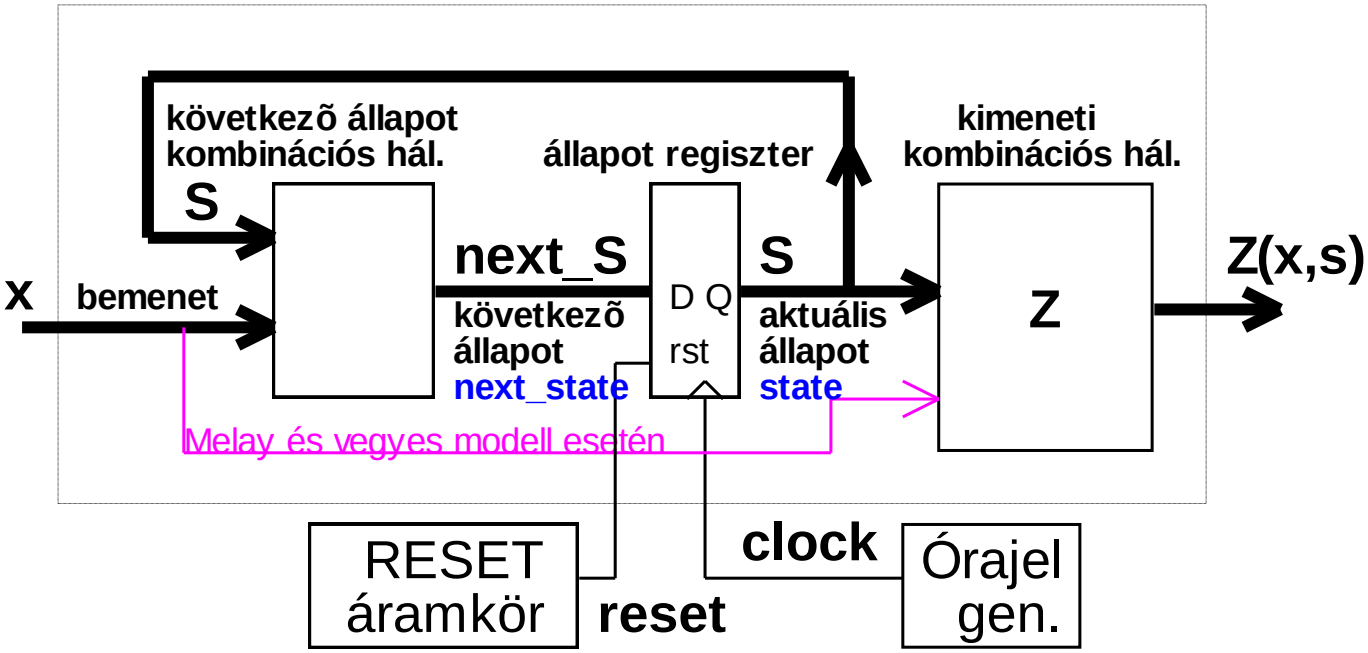


Aszinkron set és reset esetén kimenet az órajeltől függetlenül 1 lesz ***s***, 0 lesz ***r*** hatására. Szinkron esetben csak ha ezen kívül megjön az órajel aktív éle is. Utóbbit használjuk.

A szinkron sorrendi hálózat állapotregiszterét közös órajelű D flip-flopok alkotják.



A szinkron sorrendi hálózat felépítése

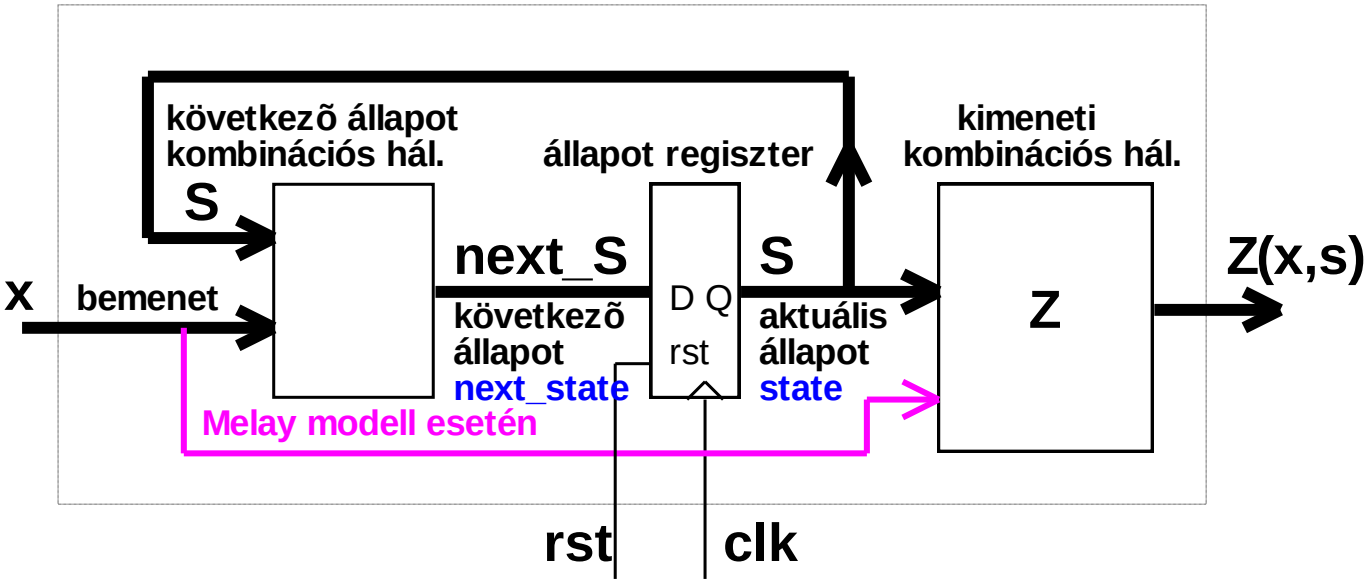


Az ***óragenerátor*** állítja elő az órajelet.

A ***RESET áramkör*** bekapcsoláskor vagy a nyomógomb megnyomásakor megfelelő hosszúságú impulzust ad ki (reset, röviden rst), melyet a sorrendi hálózat kezdő állapotának beállítására használunk. (A továbbiakban nem rajzoljuk le külön.)

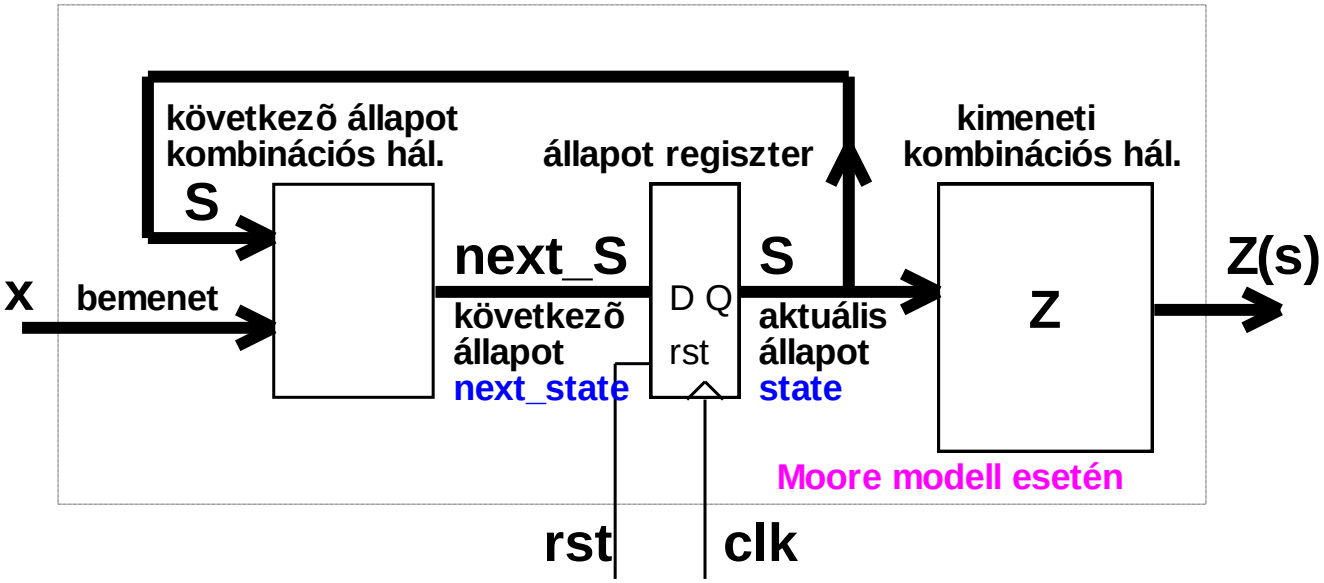
- A szinkron sorrendi hálózat ***kimenete*** az ***aktuális állapot és az aktuális bemenet függvénye*** ún. ***Mealy modell*** szerinti működésnél.

$$z=Z(s,x)$$



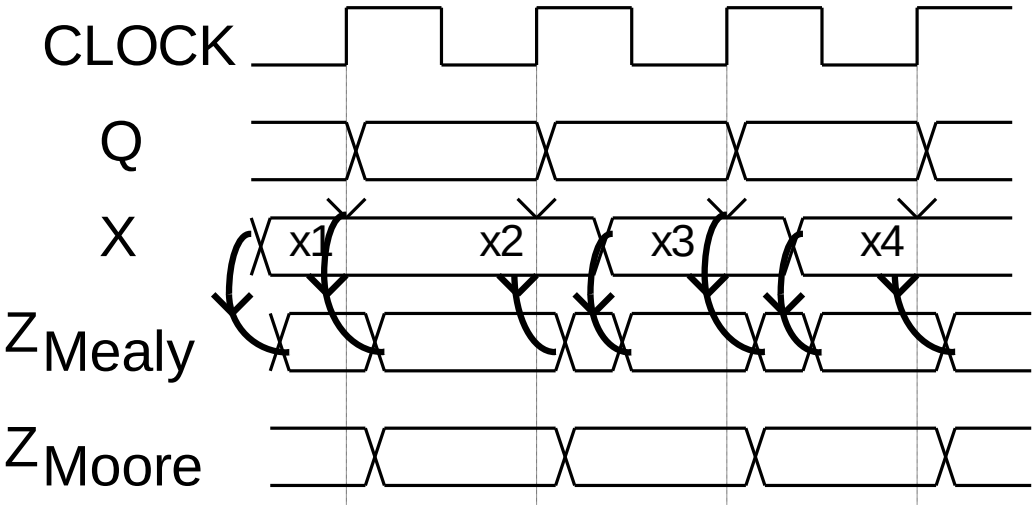
A kimenet ***csak az aktuális állapottól függ*** ún. ***Moore modell*** szerinti működés esetén.

$$z=Z(s)$$



Vegyes modell ***esetén egyes kimenetek Mealy*** $z_i=Z_i(s,x)$***, mások Moore jellegűek*** $z_j=Z_j(s)$

A működés szemléltetése idődiagrammon

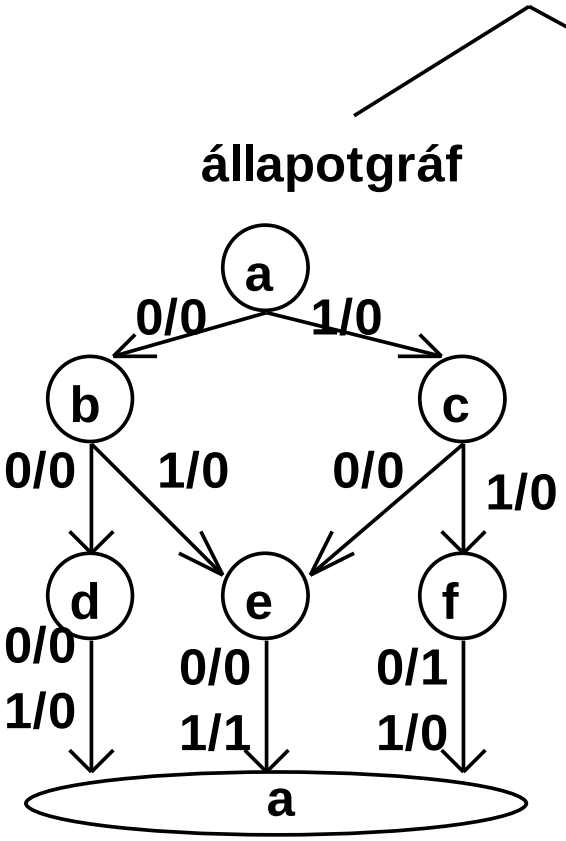


A ***Melay*** típusú ***kimenet megváltozhat, ha a bemenet vagy az állapot megváltozik***.

A ***Moore*** típusú ***kimenet csak állapot változáskor változhat meg***.

Az állapot (állapot kód, állapotregiszterben tárolt érték) ***csak az órajel aktív élének hatására változik*** meg.

A sorrendi hálózat leírási módjai



- irányított gráf
- állapot -> gráfpont
- állapot név:ABC betűi
- állapot átmenet ->él
- élre írva: bemeneti kombináció/kimeneti kombináció

- táblázatos forma
- aktuális állapot ->sor
- bemeneti kombinációk -> oszlopok
- rubrika: következő állapot /aktuális kimenet

SSH tervezése szisztematikusan

Példa:

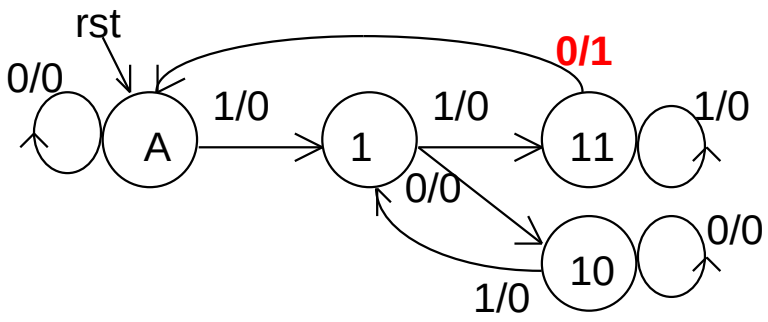
Egy sorrendi hálózat egyetlen bemenetére sorosan érkeznek a bitek, az órajellel szinkronban. A megtervezendő automatának fel kell ismerni, ha a bitfolyamban a legutolsó 3 bit 110 és a kimeneten az utolsó bittel egyidőben Z=1-el kell jeleznie. A kimenet egyébként 0.

Pl:

x: 0101110110101011110…

y: 0000001001000000**1**…

1. Szöveg alapján *előzetes állapotgráf* megrajzolása.



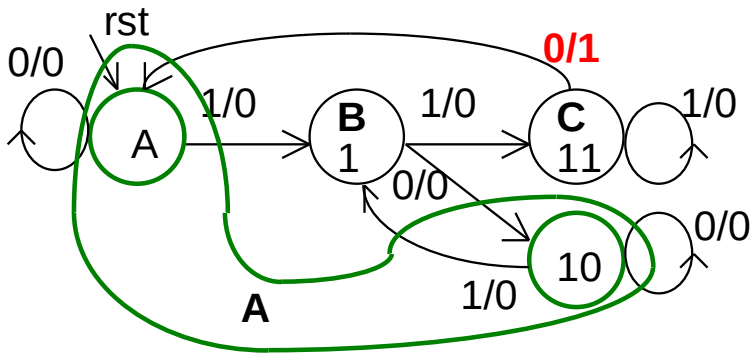
Az állapotgráf rajzolása során egy olyan kezdőállapotból indulunk ki, ami azt az információt tárolja, hogy a bekapcsolás (reset) óta nem jött adat. Ezt az állapotot általában A-val jelöljük.

Utána a már meglévő állapotból kiindulva a feladat leírása alapján minden bemeneti kombinációra megvizsgáljuk, hogy az adott állapotban milyen kimenet kell adni és, hogy fel kell-e venni új állapotot, vagy egy már meglévő tárolja azt az információt, amire az adott bemeneti kombináció után emlékeznie kell a hálózatnak.

Az állapotokba az első felíráskor azt az információt írjuk be, hogy mit jegyez meg a bemeneti sorozatból (ha ez egyszerűen megtehető). Előbb-utóbb eljutunk oda, hogy nem kell új állapotot felvennünk és minden állapotban minden bemeneti kombiációra meghatároztuk a kimenetet és a következő állapotot. Ekkor elkészült az ***előzetes állapotgráf***.

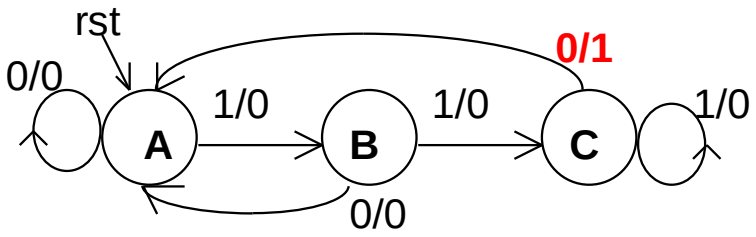
Az előzetes állapotgráf redundáns állapotokat tartalmazhat. (Olyan állapotokat, amelyek a feladat szempontjából ugyanazt az információt jegyzik meg.) Ezeket meg kell keresni és egyetlen állapottal helyettesíteni. Tehát állapotminimalizálásra lehet szükség.

2. Állapotminimalizálás



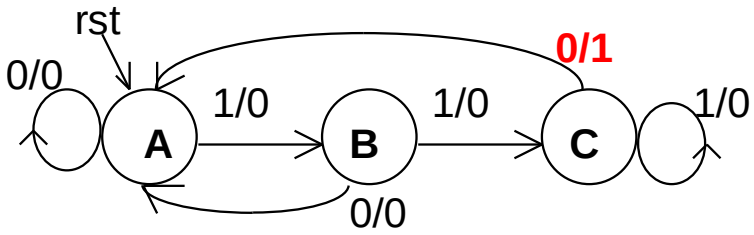
Az előbbi ***A*** és (***10***) állapotok egyetlen állapottal helyettesíthetők mivel tetszőleges de ugyanazon bemeneti kombinációra a kimenetük megegyezik és a következő állapotuk ugyanaz vagy önmaguk (a megegyezőknek feltételezett állapotok halmazában van). ***Teljesen specifikált hálózat***nál (TSH, amelyben minden állapotátmenet és kimenet specifikált) ezt úgy nevezik, hogy ***ekvivalensek*** (A == (01)). Minden egymással ekvivalens állapotot egyetlen állapottal helyettesítünk.
Gyaorlatilag az egymással ekvivalens állapotok közül 1-et meghagyunk a többit töröljük a gráfból vagy állapottáblából és az összes megmaradt állapotot átnevezzük.

A minimalizált állapotgráf:



Az állapotok száma is befolyásolja a magvalósítandó hálózat bonyolultságát. Kevesebb állapothoz többnyire egyszerűbb hálózat tartozik. (Azonban nem biztos, hogy a minimális állapotszámú lesz a legegyszerűbb.)
Az ***állapotminimalizálásra szisztematikus módszerek léteznek***. Ezek kiindulópontja az előzetes állapottábla. (A szisztematikus módszereket nem tanuljuk.)

A további eljárások kiindulópontja a ***minimalizált állapottábla***.
A ***minimalizált állapottáblát*** az állapotgráf alapján töltöttük ki:



	x=0	x=1
A	A/0	B/0
B	A/0	C/0
C	A/1	C/0

3. Állapotkódolás

Az állapotokhoz kódokat kell rendelni. A megvalósított hálózatban ezek képviselik az állapotokat. Az állapotkódolásra szintén vannak ***szisztematikus módszerek***, mivel ***az állapotkódolás is befolyásolja a megvalósítandó hálózat bonyolultságát***.

(A szisztematikus módszereket nam tanuljuk.) Itt 'ad hoc' kódoltunk. Az állapotot az állapotregiszter aktuális értéke (s1s0) reprezentálja.

s2s1s0	**x=0** **next_s1, next_s0/z**	**x=1** **next_s1, next_s0/z**
A 00	A **00**/**0**	B **01**/**0**
B 01	A **00**/**0**	C **11**/**0**
C 11	A **00**/**1**	C **11**/**0**
10	?	?

A kódolt állapottábla megadja, hogy adott kódú állapotból, adott bemeneti kombináció hatására mi lesz a következő állapot kódja (**next_s1, next_s0** egyes bitjei) és a kimenet (**z** egyes bitjei, ha több kimenetű). Tehát ***a kódolt állapottábla a következő állapot bitjeinek (next s_i) és a kimenet(ek)nek (z_j) az igazságtábláit tartalmazza!***

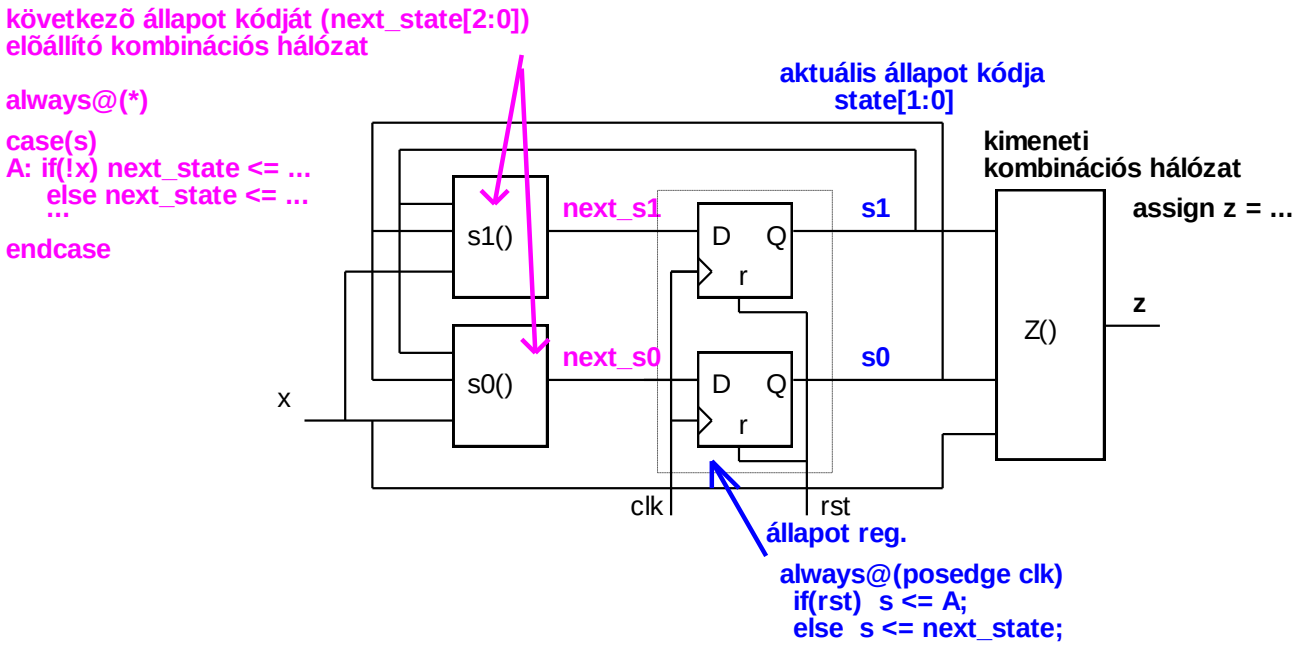
Az előző kódolt állapottáblában még nem tüntettük fel azon állapotkódokat, amelyeket nem használunk. ***A nem használt állapotkódokhoz is ki kell tölteni a kódolt állapottáblát***.

s2s1s0	**x=0** **next_s1, next_s0/z**	**x=1** **next_s1, next_s0/z**
A 00	A **00**/**0**	B **01**/**0**
B 01	A **00**/**0**	C **11**/**0**
C 11	A **00**/**1**	C **11**/**0**
10	A **00**/**0**	A **00**/**0**

- Ha ***megbízható működés***re törekszünk, akkor célszerű előírnunk, hogy ***a normál működésnél nem használt állapotkódokból is a használtakba vezessen a hálózat***. (Normál működésnél nem használt állapotkódba kerülhet a hálózat külső zavar hatására. Közömbös kitöltésnél szerencsétlen esetben itt ragadhat az automata, ha a tervezés után a következő állapotok kódja szintén nem használt állapotkódba vezet!)

4. Következő állapot kódját előállító logika és a kimenet függvényeinek meghatározása

A megtervezendő automata struktúrája:



Megtervezendők a következő állapot logika

next_s1(s1,s0,x)

next_s0(s1,s0,x)

és a kimenet **Z(s1 ,s0 ,x)** függvényei.

Mivel igazságtáblájukat egyszerre tartalmazza a kódolt állapottábla, ez alapján lehet az egyes függvények igazságtábláit kitölteni majd valamely módszerrel egyszerűsíteni.

s2s1s0	**x=0** **next_s1, next_s0/z**	**x=1** **next_s1, next_s0/z**
A 00	A **00**/**0**	B **01**/**0**
B 01	A **00**/**0**	C **11**/**0**
C 11	A **00**/**1**	C **11**/**0**
10	A **00**/**0**	A **00**/**0**

Az igazságtáblák (Logic Friday-ban megadva):

Term	s1	s0	x	=>	next_s1	next_s0	z
0	0	0	0		0	0	0
1	0	0	1		0	1	0
2	0	1	0		0	0	0
3	0	1	1		1	1	0
4	1	0	0		0	0	0
5	1	0	1		0	0	0
6	1	1	0		0	0	1
7	1	1	1		1	1	0

A Logic Friday-val minimalizált függvények:
(Abban a formában, ahogy az eredményt adja. Az x' az x negáltját jelöli)

```
next_s1 = s0 x;
next_s0 = s0 x + s1' x;
z = s1 s0 x';
```

Verilogban megadva az állapotgép leírását:

```
reg [1:0] s;
wire [1:0] next_s;          //assign értékadáshoz
//reg [1:0] next_s;         //always blokkos értékadűshoz
//Állapotkódolás
parameter A = 2'b00, B = 2'b01, C = 2'b11;

//Állapotregiszter
always @ (posedge clk)
  begin
    if (rst)    state <= A;
    else        state <= next_s;
  end
```

A következő állapot logikát többféleképpen is megadhatjuk.

a. megadhatjuk a minimalizált függvények alapján Boole algebrai alakban assign-al.

```
//next state logika
wire a,b,; // ez a hozzárendelés rövidebb leírást
//eredményez és reg típusú next_s esetén is működik
assign {a,b} = s;
assign next_s[1] = b&x;
assign next_s[0] = b&x | ~a& x;
```

b. Azonban sokkal ***célszerűbb*** az állapottábla alapján ***az állpotátmeneteket megadni és a minimalizálást a tervezőrendszerre bízni***.

c.

```
//next state logika
    always @(*)
    case(state)
    A: if(~x) next_s = A;
          else  next_s = B;
    B: if(~x) next_s = A;
          else  next_s = C;
    C: if(~x) next_s = A;
          else  next_s = C;
    default:   next_s = A;
    endcase
```

A kimeneti függvény

A kimeneti függvény megadására is több lehetőségünk van. Megadhatjuk a Logic Friday által minimalizált függvényt SOP Boole algebrai alakban:

```
assign z =  a&b&~x;
```

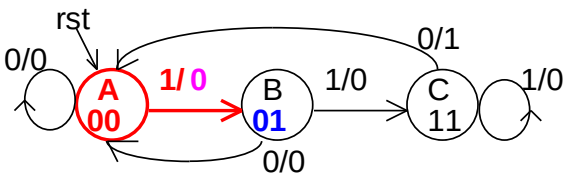
Megadhatjuk az állapotgráf/állapottábla alapján az állapotkódok felhasználásával:

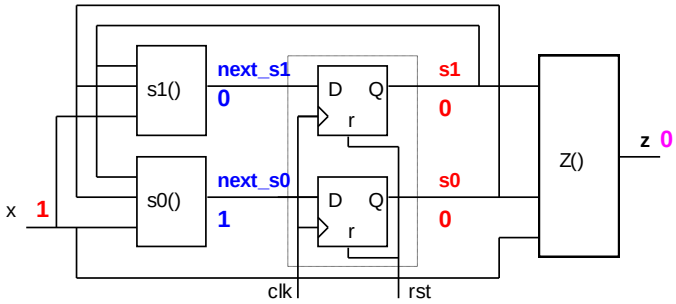
```
assign z = (state == C)&~x;
```

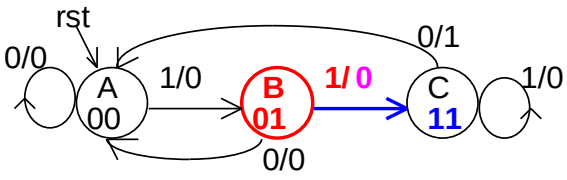
A fenti 2 megoldás összehasonlításából látható, hogy a&b = (s == C), ami az állapotkódolásból következik. (Az AND kapcsolat konstanssal történő komparálásra is használható.)

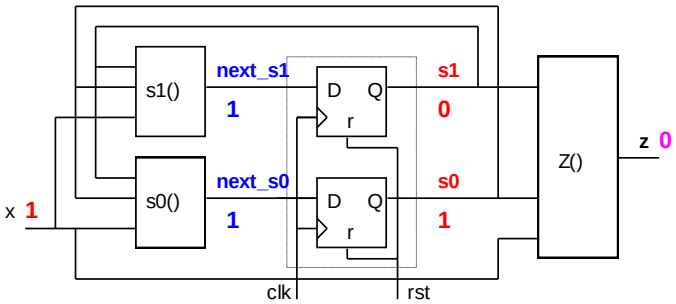
A következő oldalon a szinkron hálózat jobb megértése érdekénben bemutatjuk a működést egy rövid bemeneti sorozat esetére az állapotgráfon és a blokkvázlaton.

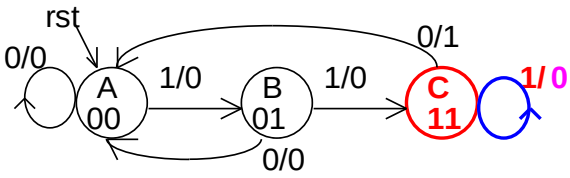
Mit csinál az automata RESET után az X=1110 bemeneti sorozat hatására?

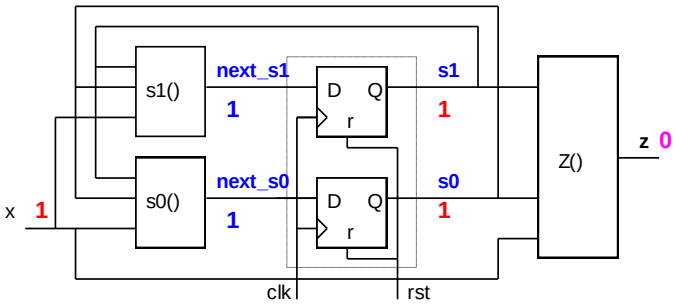


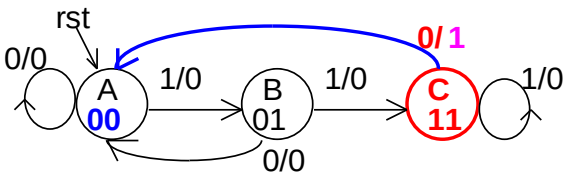


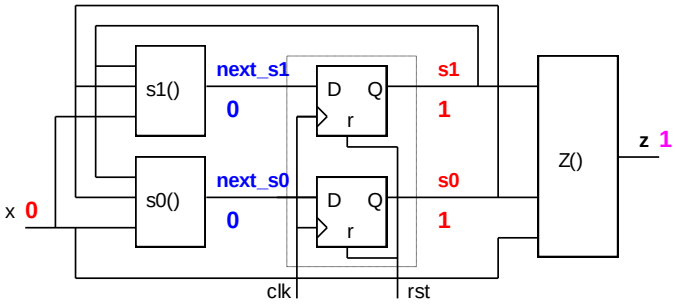


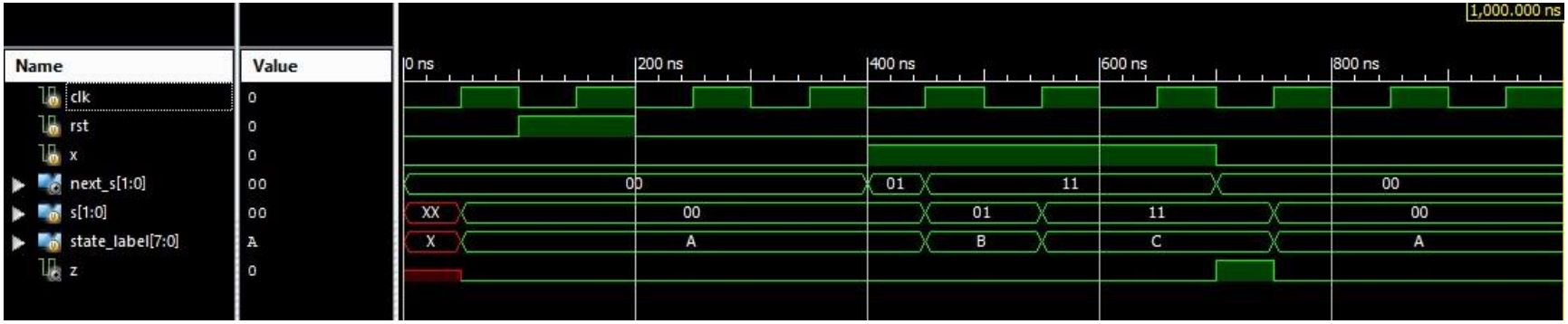











Name
Value
clk
rst
x
next_s[1:0]
s[1:0]
state_label[7:0]
z
0
0
0
00
00
A
0
0 ns
200 ns
400 ns
600 ns
800 ns
1,000.000 ns
00
01
11
00
XX
00
01
11
00
X
A
B
C
A