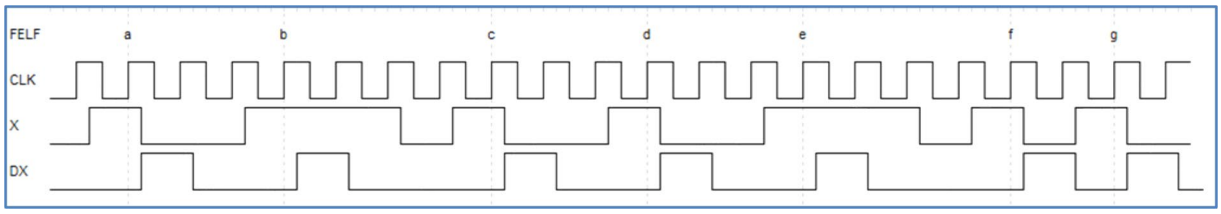
Véges állapotú gépek (FSM) tervezése

F1. A digitális tervezésben gyakran szükséges a logikai jelek változását érzékelni és jelezni. A változásdetektorok készülhetnek csak egy típusú változás (0→1, vagy 1→0) jelzésére, de lehetséges tetszőleges típusú jelváltás egyetlen áramkörön belüli észlelése és jelzése is. A feladatban egy felfutó él detektort tervezünk. Az X bemenőjel az órajellel szinkron működik (azaz teljesül az előkészítési és tartási idő feltétel), és tetszőlegesen változhat, azaz lehet rajta 1, 2, 3, vagy több órajelnyi, bármilyen hosszú magas vagy alacsony pulzus is. Tervezze meg azt a szinkron működésű sorrendi hálózatot (véges állapotú automatát, FSM-et), amely biztosítja, hogy az X bemenőjel 0→1 váltására a DX kimenet pontosan 1 órajel széles pulzussal reagál.

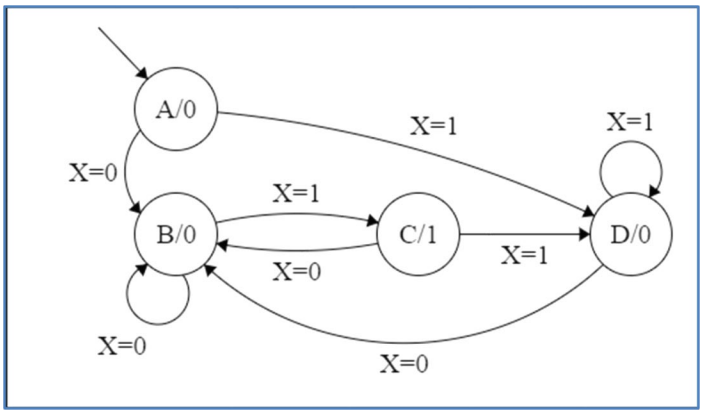
F1.a Tervezze meg az áramkört hagyományos módon, kézi módszerrel előállítva a szükséges függvényeket, állapotkódolást! Rajzolja fel az áramkört!

Az előírt működést az alábbi idődiagram szemlélteti, az X jel 0→1 átmeneteit az a,b,c,d,e,f,g órajel felfutó éleknél érzékelve, azaz az előző X=0 mintavételi értékhez képest az új mintavétel X=1 lesz:



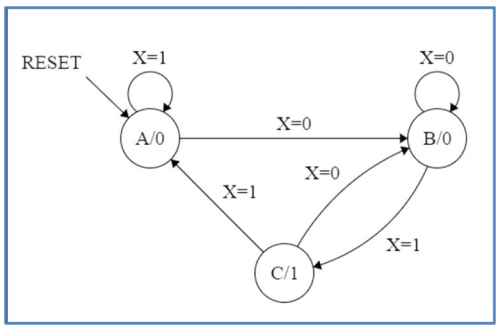
A feladat megoldásánál az előzetes állapotdiagramból érdemes kiindulni. A tervezett FSM Moore modell szerinti lesz, azaz a kimenet csak az állapotregiszter értékétől függ.

A feladat megoldása során gondoskodnunk kell arról, hogy valóban csak az érvényes 0→1 átmeneteket jelezzük, tehát a bekapcsolás utáni vagy a RESET alatti folyamatos X=1 esetben ne adjunk ki felfutó élet jelző kimeneti pulzust. Ennek figyelembe vételével a felfutó él detektor állapot diagramja a következő:



Bekapcsolás vagy RESET hatására az A kezdeti állapotba kerül. Ha a bemenetei jel értéke 0, akkor átlép a B állapotba (a kimenet ez alatt 0 marad) és várja a 0→1 átmenetet. Ha azonban a bekapcsolás vagy a RESET ideje alatt a bemenet folyamatosan „1” volt, akkor az A állapotból a D állapotba lép, mert ahhoz, hogy érvényes 0→1 átmenetet detektálhasson, legalább egy „0” állapotnak is kell lennie. Így az állapot diagramunk 4 állapotot tartalmaz. Vizsgáljuk meg, esetleg lehetséges-e egyes állapotok összevonása: Két állapotra (ezek az A és D) láthatóan a kimenetek értéke azonosan 0, továbbá a következő állapotok is azonosak, a bemenet megfelelő értékeire. X=0-ra B és X=1-re D. Ezért ez a két

állapot összevonható. Az új összevont állapotot A-val jelölve az egyszerűsített állapot diagram a következő (A RESET jel hatása természetesen öröklődött):



Az előzetes állapot átmeneti tábla:

Aktuális állapotok és / kimenetek	Következő állapot, ha az X bemenet = 0	Következő állapot, ha az X bemenet = 1
A / 0	B	A
B / 0	B	C
C / 1	B	A

Három állapot szükséges, a fenti kimeneti értékekkel. Láthatóan Moore modell szerint működik, hiszen csak az állapot határozza meg a kimenet értékét, az a bemenet aktuális értékétől független.

A 3 állapot 2 bites állapotregisztert igényel. Az állapotkódolás legyen A=00, B=01, C=10. A feladatmegoldás lényegi része eddig tartott. Ha jól értelmeztük a feladatot és helyesen rajzoltuk fel az állapotdiagramot, továbbá az előzetes állapottáblát, akkor a további lépések, bár fontosak, de már csak a részletek kidolgozását jelentik, tartalmilag nem adnak újat a munkához. Ezután a kódolt állapot átmeneti tábla:

Állapot neve	Bemenet	state		next_state		Kimenet
	X	[1]	[0]	[1]	[0]	DX
A	0	0	0	0	1	0
A	1	0	0	0	0	0
B	0	0	1	0	1	0
B	1	0	1	1	0	0
C	0	1	0	0	1	1
C	1	1	0	0	0	1

A kódolt állapot átmeneti táblából az állapot átmeneti függvény és a kimeneti függvény kiolvasható és felrajzolható. Például az utolsó oszlop alapján a kimenet a C állapotban mindig 1, természetesen X-től

függetlenül (Moore modell!). Ez egy 2 mintermet tartalmazó szorzatkifejezés, melyben az X bemenet értéke közömbös, tehát egyszerűsíthető:

DX = state[1] * /state[0] * /X + state[1] * /state[0] * X = state[1] * /state[0]

MEGJEGYZÉS: Felmerülhetne a DX = state[1] egyszerűsítés is, azonban biztonsági okokból meghagyjuk a C állapot teljesen specifikált kódját. Az 11 állapotkód ugyanis nem használt, és ha esetleg véletlenül, valamely külső hatásra abba az állapotba lépne a vezérlő, akkor a DX = state[1] egyúttal aktív kimenetet is generálna, ami hibás jelzést jelentene.

A következő állapot logika függvénye a next_state[0] bitre nagyon egyszerű lesz. Ha X=0, akkor a következő állapot mindig B egyébként pedig A vagy C. Láthatóan a next_state[0] az X invertáltja.

next_state[0] = /X

A következő állapot logika függvénye a next_state[1] bitre sem bonyolult. Egyetlen esetben 1, amikor B állapotba lépve az X továbbra is 1 értékű.

next_state[1] = X */state[1] * state[0])

F1.b Tervezze meg az áramkört Verilog HDL specifikáció alapján, a nyelvi elemek legkedvezőbb alkalmazásával!

Túl sok optimalizációs lehetőség nem adódik. Ha már manuálisan elkészítettük a részletes logikai függvényeket, talán az a legegyszerűbb, ha ezeket írjuk be a kódba is. Definiáljuk az állapotváltozót, state[1:0], az őt vezérlő következő állapot logika kimeneti változóját, next_state[1:0] és felírjuk a függvényeket. Az állapotváltozó egy törölhető, minden órajelben frissített 2 bites regiszter.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Felfutó él detektor
// Az X bemenőjel 0->1 váltása után kiad egy 1 órajelpulzus széles jelzést
//////////////////////////////////////////////////////////////////////////////////
module rising_edge_detector(
    input clk,
    input rst,
    input x,
    output dx
    );

//////////////////////////////////////////////////////////////////////////////////
// Belső jelek deklarálása
//////////////////////////////////////////////////////////////////////////////////
reg  [1:0] state;
wire [1:0] next_state;

//////////////////////////////////////////////////////////////////////////////////
// A kétbites állapotregiszter
//////////////////////////////////////////////////////////////////////////////////

always @ (posedge clk)
   if (rst)     state <= 2'b00;
   else         state <= next_state;

//////////////////////////////////////////////////////////////////////////////////
// Az állapotátmeneti logika
//////////////////////////////////////////////////////////////////////////////////

assign next_state[0] = ~x ;
assign next_state[1] = x & ~state[1] & state[0];

//////////////////////////////////////////////////////////////////////////////////
// A kimeneti logika
//////////////////////////////////////////////////////////////////////////////////

assign dx = state[1] &  ~state[0];

endmodule
```

A következő állapot logika leírása lehetséges lenne egy **case** szerkezettel, de ebben az esetben nem segít sokat

```
reg [1:0] next_state;

always @(*)
   case (state)
      2'b00: if (x) next_state <= 2'b01;
                else next_state <= 2'b00;
      2'b01: if (x) next_state <= 2'b10;
                else next_state <= 2'b01;
      2'b10: if (x) next_state <= 2'b00;
                else next_state <= 2'b01;
      default       next_state <= 2'b00;
```

A felfutó él detektor működésének ellenőrzése szimulációval: