Véges állapotú gépek (FSM) tervezése

F1. Tervezzünk egy soros mintafelismerőt, ami a bemenetére **ciklikusan,** sorosan érkező 4 bites számok közül felismeri azokat, amelyek 3-mal vagy 5-tel oszthatók. A fenti tulajdonságnak megfelelő bit sorozatokat a 4. ütemben, a 4. bit beérkezésével egy időben jelzi, a kimenet 1-be állításával. Egyébként a kimenet a többi bitciklusban legyen 0.

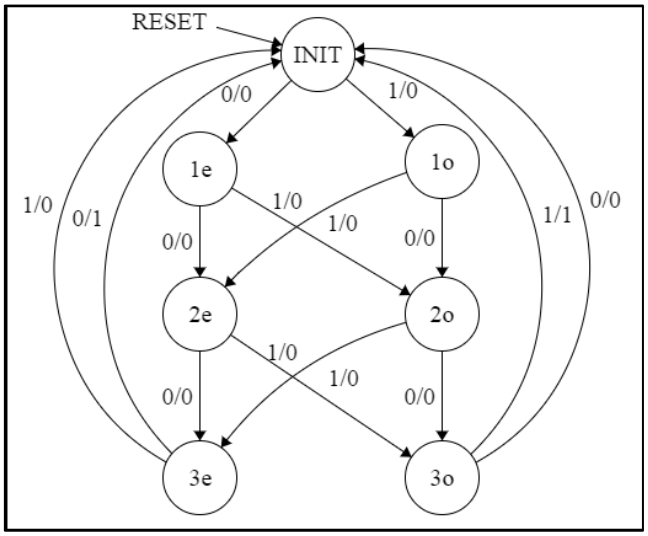
F1.a Gondoljuk végig, mi a megadott feltételnek megfelelő számok halmaza és milyen tulajdonság alapján tervezhető sorosan érkező bitek alapján a logika? A soros működésnél melyik sorrendben várjuk a biteket?

A feltételeket kielégítő számok: 0, 3, 5, 6, 9, 10, 12, 15. A lehetséges értékhalmaz fele, azaz 8 érték. A bitmintákat elemezve észrevehető, hogy a jó értékekre jellemző az adatbitek páros paritása, tehát 4 bitre a párhuzamos megoldás egy XNOR függvény lenne. A soros mintafelismerő egy lehetséges verziója is e tulajdonság alapján származtatható. A 4 bites beérkező minták páros paritását kell vizsgálnunk.

A beérkező bitsorrend ebben az esetben érdekes módon, ill. természetesen közömbös. Egyrészt a nyolc bitminta mindegyikének szimmetrikus párja is szerepel a jó értékek között, ezért nyilván a bitsorrend megcserélhető. Másrészt a páros/páratlan paritás tulajdonság csak a 0 és/vagy az 1 bitek számosságától függ, és független a számolás sorrendjétől. De feltételezhetjük a továbbiakban, hogy pl. MSB-vel (legnagyobb helyiértékű bit) kezdve érkeznek a bitek, ha ez segít a feladat megoldásában.

F1.b Tervezze meg a mintafelismerő egység állapotdiagramját és előzetes állapottábláját az FSM-ek általános felépítését választva!

A páros paritást felismerő állapotdiagram a következő:



RESET után, vagy minden 4. ciklusban a rendszer az INIT állapotba kerül. A következő 4 bit első beérkező bitje alapján a páros (e, even) vagy páratlan (o, odd) oldalra lép. A 2. és 3. bitek beérkezésével a lehetséges állapotátmenetek lényegében a páros-páratlan tulajdonság megtartását vagy váltását jelentik. A 3. bit beérkezése után a 4. bit hatására visszalép az INIT állapotba, egyúttal a kimeneten kiadja a találatot jelző 1 vagy 0 kimeneti értéket. **Ennek megfelelően ez egy Mealy FSM. Mivel a**

bemeneti bitek megszakítás nélkül, sorban érkeznek egymás után, itt értelemszerűen a Mealy modell szerinti működés a megfelelő, ami a 4. bittel egyidőben jelzi a feltétel esetleges teljesülését.

Állapotminimalizálás sajnos nem lehetséges, mert meg kell különböztetni a 4 bit beérkezési fázisát, és minden bitnél szükséges a páros-páratlan megkülönböztetés is. Az állapot átmeneti tábla a következő:

ÁLL	INPUT	
	0	1
INIT	1e/0	1o/0
1e	2e/0	2o/0
1o	2o/0	2e/0
2e	3e/0	3o/0
2o	3o/0	3e/0
3e	INIT/1	INIT/0
3o	INIT/0	INIT/1

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

Az állapotkódoláshoz 3 bites állapotregiszter szükséges, az állapotkódok a beérkező bitek sorszámával és értékével logikusan is kijelölhetők. Az első két bit a sorszám, 0,1,2,3, az utolsó bit a beérkezett bit értéke. Ez a fajta kódolás előnyös az állapotátmeneti függvény egyszerűsíthetőségéhez is.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Soros mintafelismerő 4 bites ciklikus mintákra a 3 vagy 5 oszthatóságra
//////////////////////////////////////////////////////////////////////////////////
module MOD3_5_ser(
    input clk,
    input rst,
    input inp,
    output reg out
    );

//////////////////////////////////////////////////////////////////////////////////
// Állapot specifikáció
//////////////////////////////////////////////////////////////////////////////////

parameter INIT = 3'b000;
parameter _1o  = 3'b011;
parameter _1e  = 3'b010;
parameter _2o  = 3'b101;
parameter _2e  = 3'b100;
parameter _3o  = 3'b111;
parameter _3e  = 3'b110;
```

```
////////////////////////////////////////////////////////////////////////////////
// Az állapotregiszter működése
////////////////////////////////////////////////////////////////////////////////

reg [2:0] state, next_state;

always @ (posedge clk)
   if (rst) state <= INIT;
   else     state <= next_state;
```

```
////////////////////////////////////////////////////////////////////////////////
// Az állapot átmeneti logika
////////////////////////////////////////////////////////////////////////////////

always @ (*)
   case (state)
   INIT: if (inp) next_state <= _1o;
         else     next_state <= _1e;
   _1o : if (inp) next_state <= _2e;
         else     next_state <= _2o;
   _1e : if (inp) next_state <= _2o;
         else     next_state <= _2e;
   _2o : if (inp) next_state <= _3e;
         else     next_state <= _3o;
   _2e : if (inp) next_state <= _3o;
         else     next_state <= _3e;
   _3o :          next_state <= INIT;
   _3e :          next_state <= INIT;
      default:    next_state <= INIT;
   endcase
```

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


always @ (*)
   case (state)
   _3o : if (inp) out <= 1'b1;
         else     out <= 1'b0;
   _3e : if (inp) out <= 1'b0;
         else     out <= 1'b1;
      default:    out <= 1'b0;
   endcase
```