Digitális technika HF1
Sorrendi hálózatok tervezése

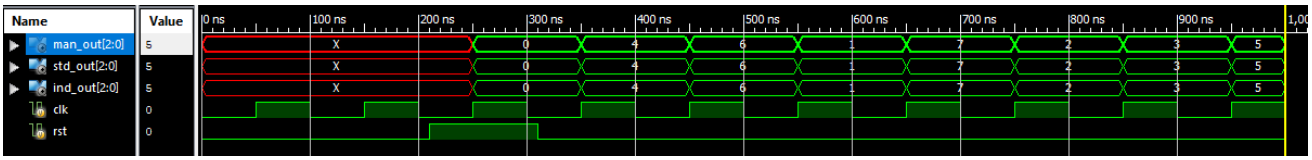
Név: Nádor Roland Viktor	**NEPTUN:** CYYRM5	**Email:** rolandnador@gmail.com
Tankör: I04	**Gyak_kurzus:** G17	**DIGIT_kód**: 04617235

A feladatokat önállóan, meg nem engedett segédeszközök használata, és mások közvetlen közreműködése nélkül oldottam meg:

aláírás

Összefoglaló a végeredményról:

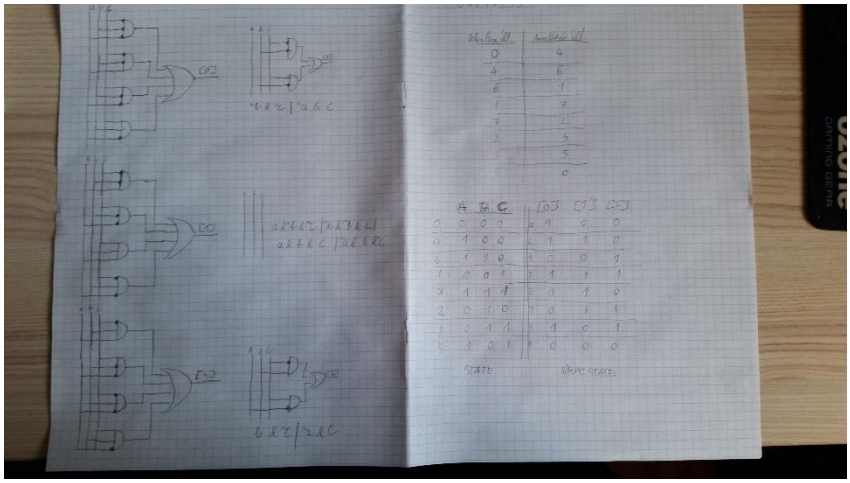
A DIGIT kódom által megadott sorrendben számoló 3 típusú véges állapotú vezérlő együttes szimulációs időidiagramja a következő:



Mint a képen is látszik, a feladatkiírásnak megfelelően mindhárom modul azonos kimenetet ad.
Mindhárom modul elkészítésénél adódtak problémák, főleg a verilog HDL szintaktikáját volt nehéz elsajátítani. A legtöbb fejtörést a MAN_FSM modul jelentette.

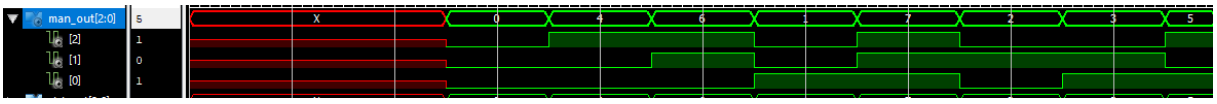
MAN_FSM modul:

A modul tervezésénél kihasználta, hogy a képen látható módon binárisan felírt számok minden oszlopában 4 darab 1es van, így a kimenet mindhárom bitjét 4 darab 3 bemenetes ÉS kapuval oldottam meg, amiket a képen látható, rajz alapú módszerrel egyszerűsítettem. Összevontam azon kapukat, melyekben két váltózó azonos módon, a harmadik pedig ponált és negált alakban is szerepel, mivel az a végeredmény szempontjából lényegtelen. Sajnos a második bit kapuit nem tudtam egyszerűsíteni.



```
module MAN_FSM(
    input clk,
    input rst,
    output [2:0] man_out
    );

reg [2:0] state;
reg [2:0] next_state;

wire a, b, c;

assign {a,b,c} = state;

always @(posedge clk)
begin
if(rst) state <= 3'b0;
else state <= next_state;
end

always@(*)
begin

next_state[2] = ~b & ~c | ~a & c;
next_state[1] = a & ~b & ~c | ~a & ~b & c | a & b & c | ~a & b & ~c;
next_state[0] = b & ~c | ~a & c;
end

assign man_out = state;

endmodule
```

Mivel ez volt az első modul, a legtöbb kód-beli elírást itt követtem el, de sikerült őket javítanom, és a későbbiekben már nem követtem el ennyi hibát.

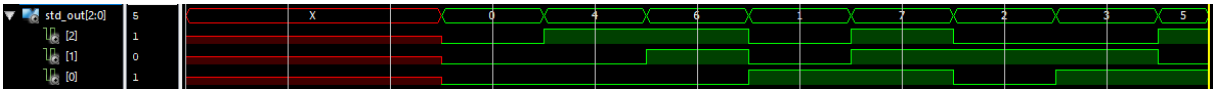


STD_FSM modul

Felvettem a 8 számot, a feladat kiírás szerint, paraméterként, majd az always blokkban case szerkezet használtaval megfelelő sorrendben egymás után kötöttem a számokat, mintegy 8 állapotú, körkörös számlálót létrehozva ezzel.

```
    input clk,
    input rst,
    output [2:0] std_out
    );


reg [2:0] state, next_state;

parameter START = 3'o0;
parameter A = 3'o4;
parameter B = 3'o6;
parameter C = 3'o1;
parameter D = 3'o7;
parameter E = 3'o2;
parameter F = 3'o3;
parameter G = 3'o5;


always@(posedge clk)
begin
if (rst) state <= 3'o0;
else state <= next_state;
end

//assign state = START;
always@(*)
begin
case(state)
START: next_state <= A;
A: next_state <= B;
B: next_state <= C;
C: next_state <= D;
D: next_state <= E;
E: next_state <= F;
F: next_state <= G;
G: next_state <= START;
default: next_state <= 3'o0;
endcase
end
```

Ez a feladat igényelte a legkevesebb gondolkodást. Ezt találtam a legkönnyebnek.



IND_FSM modul

Mint az előző mdulnál, itt is számláló alapú a működés, de itt külön egy változó (cnt) a számláló, és a digitkód számai vannak hozzárendelve az egyes értékeihez, mint kimenet.

```
module IND_FSM(
    input clk,
    input rst,
    output [2:0] ind_out
    );


reg [2:0] cnt, out;

always@(posedge clk)
begin
if(rst) cnt <= 3'o0;
else cnt <= cnt + 1;
end

always@(*)
case(cnt)
3'o0: out <= 3'o0;
3'o1: out <= 3'o4;
3'o2: out <= 3'o6;
3'o3: out <= 3'o1;
3'o4: out <= 3'o7;
3'o5: out <= 3'o2;
3'o6: out <= 3'o3;
3'o7: out <= 3'o5;
default: out <= 3'o0;
endcase

assign ind_out = out;

endmodule
```

Miután rájöttem hogyan kell számlálót készíteni a cnt segítségével, ezt a feladatot is könnyen megoldottam.