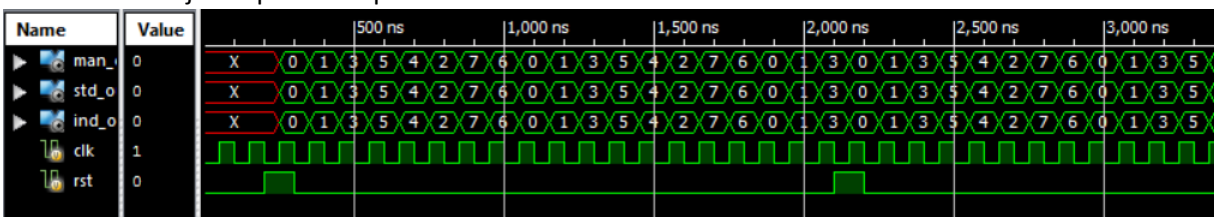
Digitális technika 1. házi feladat
Sorrendi hálózatok tervezése

Adatok
Név: Nagy Marcell
Neptun kód: G4SDSA
E-mail cím: –
Tankör: I03
Gyakorlat kurzus: G07
Digit kód: 1354276

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

Összefoglalás a végeredményről
Mindhárom elkészített modul kimenete megegyezik a digit kódom 0-val kiegészített kódjával. A modulokat működés közben is kipróbáltam az FPGA kártyán, az eredmény itt is megfelelő volt. Az ISim szimulációjának pillanatképe:



Vélemény
Eddigi tudásom alapján az indirekt SFM-et tartom a legjobb megoldásnak, ezt a legkönnyebb továbbbővíteni, 8-nál akár sokszor több állapotú állapotgép készítésére használni, ahol az állapotok utasítását nem befolyásolja az állapot sorszáma. A hagyományos módot túlbonyolítottnak találom, ezt nem alkalmaznám a gyakorlatban. A hagyományosnál sokáig tartana egyesével megadni a következő állapotot, az általánosnál pedig túl sok paramétert kellene felsorolni.
Megoldás közben az egyedüli problémát a szimuláció jelentette, vélhetően elrontottam valamit a teszt fájlban. Bár konkrétan még most se tudom, mi volt a hiba, később sikerült megoldani.

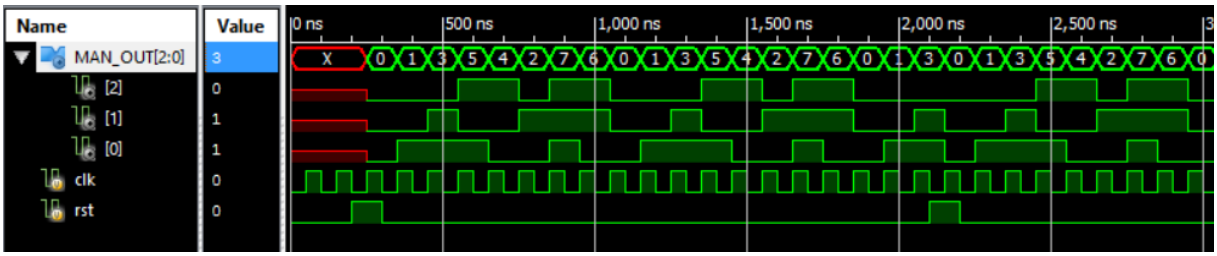
1. feladat: hagyományos tervezési mód

Ennél a módnál csak az aktuális állapotot kell eltárolni, a többi állapot értéke ebből lesz kiszámítva. Bizonyos esetekben ez a módszer kevesebb műveletet vehet igénybe. Ez hasznos, ha minél kevesebb adat eltárolása, és a minél gyorsabb működés a cél, de nehezen módosítható, a kód eléggé átláthatatlan, könnyen lehet hibázni az elkészítés közben.

A modul forráskódja:

```
module MAN_SFM(
    input clk,
    input rst,
    output [2:0] MAN_OUT
);

    reg [2:0] state, next_state;
    wire a, b, c;

    assign MAN_OUT = state;
    assign {a,b,c} = state;

    always @ (posedge clk) begin
        if (rst)
            state <= 3'b0;
        else
            state <= next_state;
    end
    always @ (*) begin
        next_state[2] <= ~a&b | a&c;
        next_state[1] <= ~a&~b&c | a&~b&~c | ~a&b&~c | a&b&c;
        next_state[0] <= ~a;
    end
endmodule
```

A modul tesztelésének eredménye:



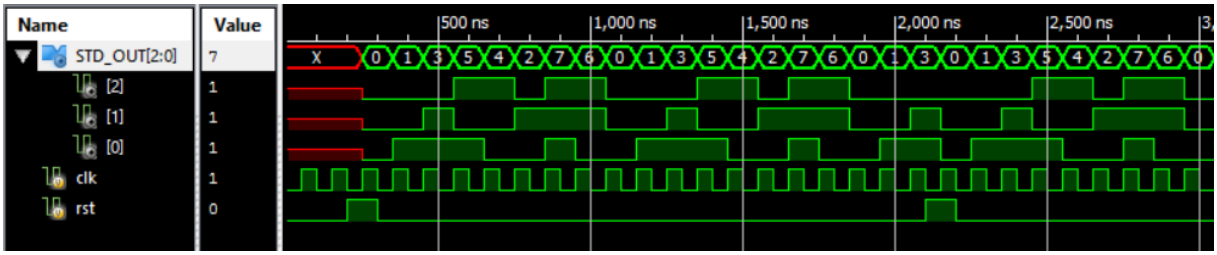
2. feladat: általános tervezési mód

Ezzel a tervezési móddal könnyen látható a készítő számára, melyik állapot után melyik másik állapot fog következni. Az állapotokat egy helyen kell felsorolni. Sok állapot esetében sok helyet foglalhat a felsorolás, és a következő állapot kiértékelése.

A modul forráskódja:

```
module STD_SFM(
    input clk,
    input rst,
    output [2:0] STD_OUT
);

    reg [2:0] state, next_state;

    assign STD_OUT = state;

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

    always @ (posedge clk) begin
        if (rst)
            state <= START;
        else
            state <= next_state;
    end
    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;
            default: next_state <= START;
        endcase
    end
endmodule
```

A modul tesztelésének eredménye:



3. feladat: indirekt tervezési mód

Az egyes állapotokat ha sorszámmal látjuk el, akár több száz állapottal is dolgozhatunk anélkül, hogy mindegyiket külön meg kellene adnunk. Azonban a számláló csak egy meghatározott szabály szerint tud változni, ha speciális sorszámot akarunk az állapotoknak, azt nehéz lenne megoldani, vagy rontaná a teljesítményt.

A modul forráskódja:

```
module IND_SFM(
    input clk,
    input rst,
    output [2:0] IND_OUT
);

    reg [2:0] cnt, out;

    assign IND_OUT = out;

    always @ (posedge clk) begin
        if (rst)
            cnt <= 3'b0;
        else
            cnt <= cnt + 3'b1;
    end
    always @ (*) begin
        case (cnt)
            3'o0: out <= 3'o0;
            3'o1: out <= 3'o1;
            3'o2: out <= 3'o3;
            3'o3: out <= 3'o5;
            3'o4: out <= 3'o4;
            3'o5: out <= 3'o2;
            3'o6: out <= 3'o7;
            3'o7: out <= 3'o6;
            default: out <= 3'o0;
        endcase
    end
endmodule
```

A modul tesztelésének eredménye: