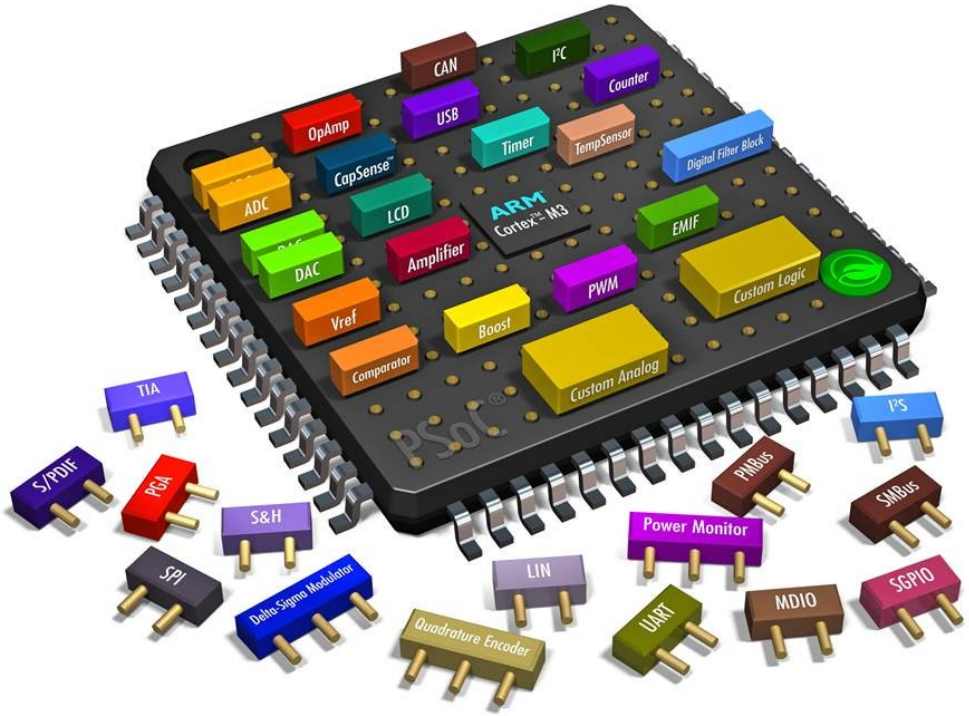
Újrakonfigurálható eszközök



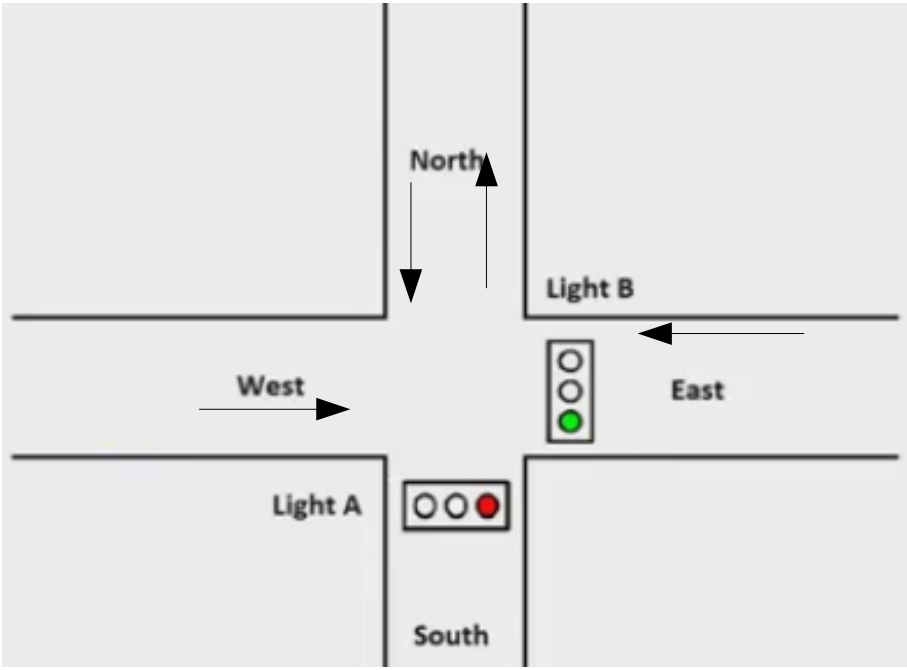
6. Véges állapotgépek: közlekedési lámpa vezérlése

Felhasznált irodalom és segédanyagok

- Icarus Verilog Simulator: http://iverilog.icarus.com/
- GtkWave wave viever: http://gtkwave.sourceforge.net/
- Verilog online tutorial: vol.verilog.com/VOL/main.htm
- Verilog tutorial for beginners: www.referencedesigner.com/tutorials/verilog/verilog_01.php
- University of Washington Computer Science & Engineering, CSE370 Lecture 23: Factoring FSMsDesign Examples
- ASIC world – Verilog tutorial: asic-world.com/verilog/veritut.html
- Végh János: Bevezetés a **Verilog** hardverleíró nyelvbe
- Végh János: Segédeszközök az **Altera DE2** tanulói készlethez
- Végh János: Bevezetés a **Quartus II V13** fejlesztő rendszerbe

Trafficlight1: Közlekedési lámpa vezérlése

- Elsőként egy nagyon egyszerű esettel foglalkozunk: egyszerű útkereszteződés, ahol nincs kanyarodási lehetőség.
- A lámpák vezérléséhez szükségünk lesz egy olyan állapotgépre, amely végiglépked az összes állapoton, s mindegyik állapotban adott ideig tartózkodik.

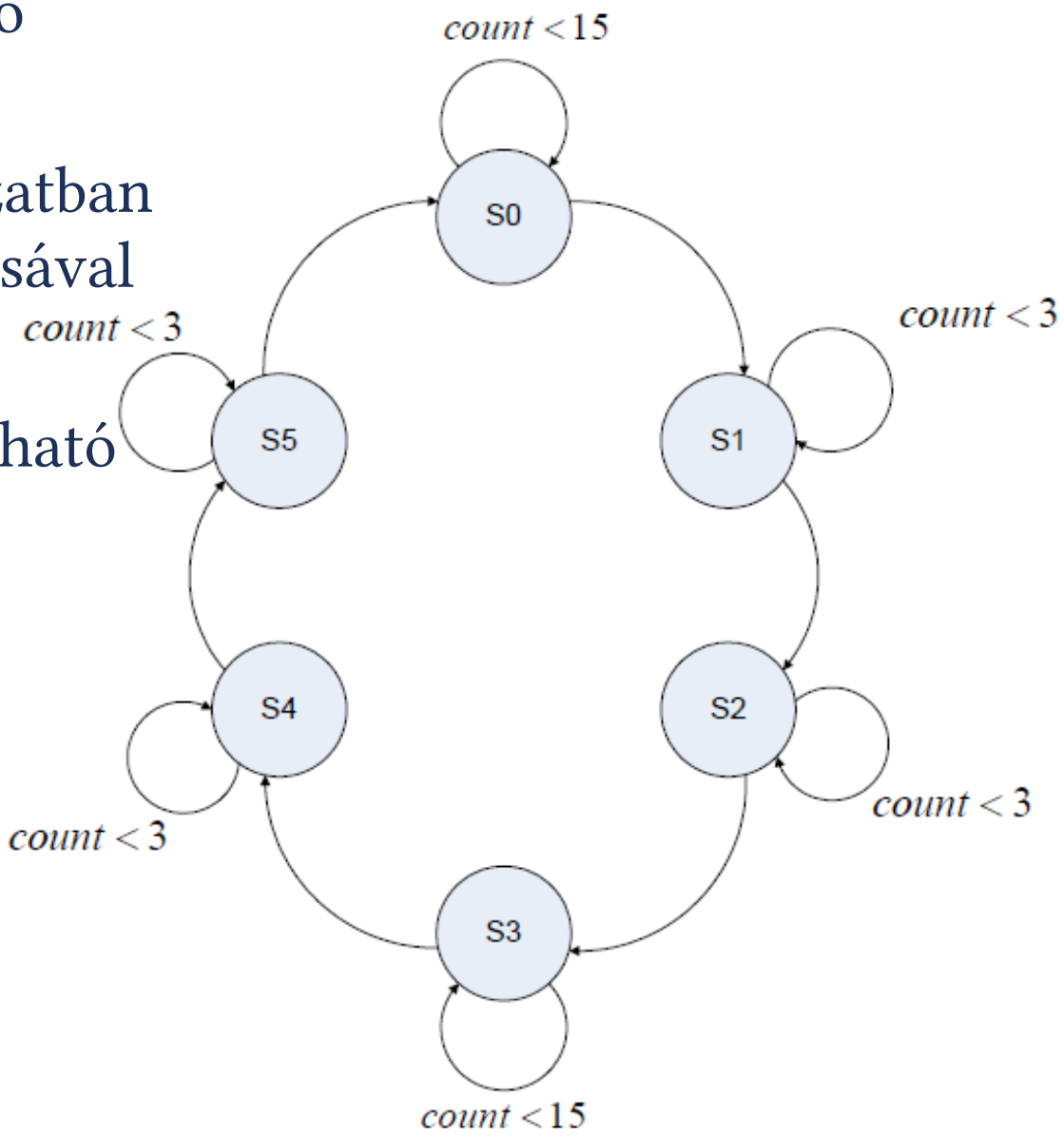


State	North - South	East - West	Delay (sec.)
0	Green	Red	5
1	Yellow	Red	1
2	Red	Red	1
3	Red	Green	5
4	Red	Yellow	1
5	Red	Red	1

Forrás: Richard E. Haskell and Darrin M. Hanna: Digital Design Using Digilent FPGA Boards - Verilog/Active-HDL Edition, 3rd Edition

Állapotdiagram

- Az időzítést egy újraindítható számlálóval oldjuk meg
- 3 Hz-es órajel esetén a táblázatban szereplő értékek háromszorosával kell számolni
- Az állapotdiagramról leolvasható a továbblépés feltétele

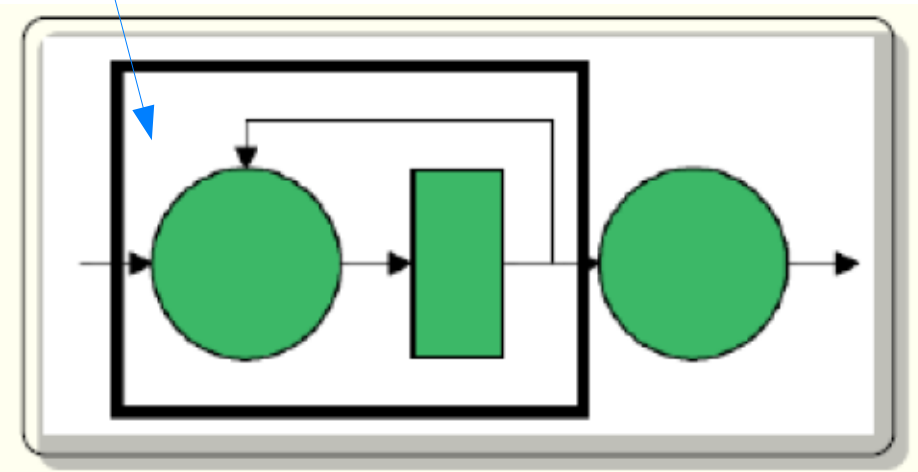


Az állapotgép forráskódja (traffic.v)

```
module traffic (input wire clk, input wire clr, output reg [5:0] lights);
reg [2:0] state;
reg [3:0] count;

parameter S0 = 3'b000, S1 =3'b001, S2 = 3'b010, // states
          S3 = 3'b011, S4 = 3'b100, S5 = 3'b101;

parameter SEC5 = 4'b1111, SEC1 = 4'b0011;       // delays

always @(posedge clk or posedge clr)
  begin
    if (clr == 1) begin
        state <= S0;
        count <= 0;
    end else
    case (state)
        S0: if (count < SEC5) begin
                state <= S0; count <= count + 1;
            end
            else begin
                state <= S1; count <= 0;
            end
        S1: if (count < SEC1) begin
                state <= S1; count <= count + 1;
            end
            else begin
                state <= S2;
                count <= 0;
            end
```

Ez az ***always*** blokk
az állapotvektort és az
állapotátmeneteket írja le.

Lásd: **cpld05** előadás, 8. dia



Az állapotgép forráskódja (traffic.v)

```
S1:
    if (count < SEC1) begin
        state <= S1; count <= count + 1;
    end
    else begin
        state <= S2; count <= 0;
    end
S2:
    If (count < SEC1) begin
        state <= S2; count <= count + 1;
    end
    else begin
        state <= S3; count <= 0;
    end

S3:
    if (count < SEC5) begin
        state <= S3; count <= count + 1;
    end
    else begin
        state <= S4; count <= 0;
    end
S4:
    if  (count < SEC1) begin
        state <= S4; count <= count + 1;
    end
    else begin
        state <= S5; count <= 0;
    end
```

Az állapotgép forráskódja (traffic.v)

```
        S5:
            if (count < SEC1) begin
                state <= S5;
                count <= count + 1;
            end
            else begin
                state <= S0;
                count <= 0;
            end
        default
          state <= S0;
    endcase
  end

always  @(*) begin
    case (state)
        S0: lights = 6'b100001;
        S1: lights = 6'b100010;
        S2: lights = 6'b100100;
        S3: lights = 6'b001100;
        S4: lights = 6'b010100;
        S5: lights = 6'b100100;
        default lights = 6'b100001;
    endcase
end

endmodule
```

A kimeneti állapotok dekódolása

Feladat: módosítsuk az állapotgép forráskódját úgy, hogy mindkét útvonalon a zöld jelzés előtt a hazai szabályoknak megfelelő piros/sárga előkészítő jelzés is legyen!

Tipp: ehhez elegendő csupán a kimeneti dekódolást módosítani.

A „próbapad” állomány (traffic_tb.v)

```
`include "traffic.v"                // Becsatoljuk az állapotgép forráskódját

module test();
reg clk,clr;
wire [5:0] lights;

traffic fsm(clk, clr, lights); // Példányosítjuk az állapotgépet

initial begin
    $dumpfile("test.vcd");      // Naplózó állomány GtkWave számára
    $dumpvars(0,test);
    clr = 1'b0;
    clk = 1'b0;
    $monitor("TIME = %d, reset = %b, lights = %6b",$time,clr,lights);

    #1 clr = 1;                 // RESET jel induláskor
    #2 clr = 0;
    #500 $finish;
end

always
    #1 clk = ~clk;              // 0.5 Hz-es órajel generálása

endmodule
```

Iverilog szimuláció (trafficlights1)

- A szimulációt az alábbi parancsokkal futtathatjuk:

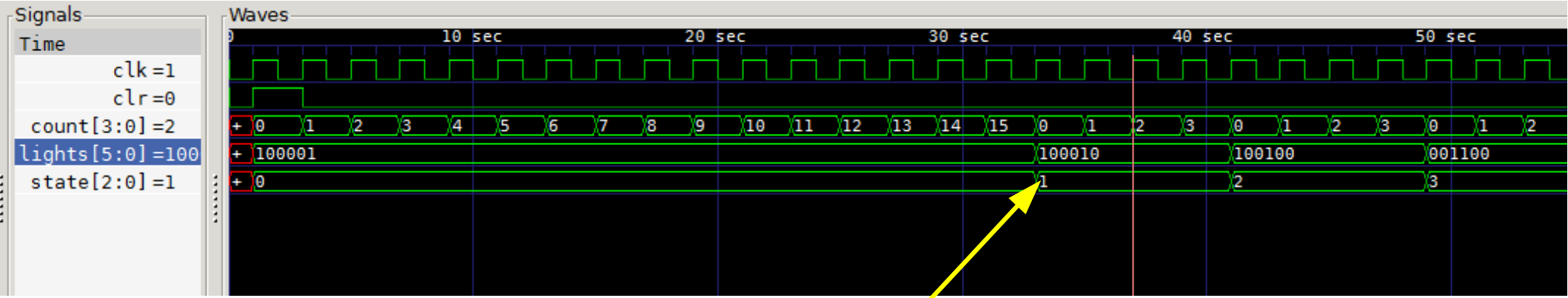
```
> PATH=%PATH%;C:\iverilog\bin;C:\iverilog\gtkwave\bin
> iverilog -o test traffic_tb.v
> vvp test
```

Automatikusan becsatolja a **traffic.v** állományt is!

```
VCD info: dumpfile test.vcd opened for output.
TIME =                      0, reset = 0, lights = xxxxxx
TIME =                      1, reset = 1, lights = 100001
TIME =                      3, reset = 0, lights = 100001
TIME =                     33, reset = 0, lights = 100010
TIME =                     41, reset = 0, lights = 100100
TIME =                     49, reset = 0, lights = 001100
TIME =                     81, reset = 0, lights = 010100
TIME =                     89, reset = 0, lights = 100100
TIME =                     97, reset = 0, lights = 100001
TIME =                    129, reset = 0, lights = 100010
TIME =                    137, reset = 0, lights = 100100
TIME =                    145, reset = 0, lights = 001100
TIME =                    177, reset = 0, lights = 010100
TIME =                    185, reset = 0, lights = 100100
TIME =                    193, reset = 0, lights = 100001
TIME =                    225, reset = 0, lights = 100010
…
```

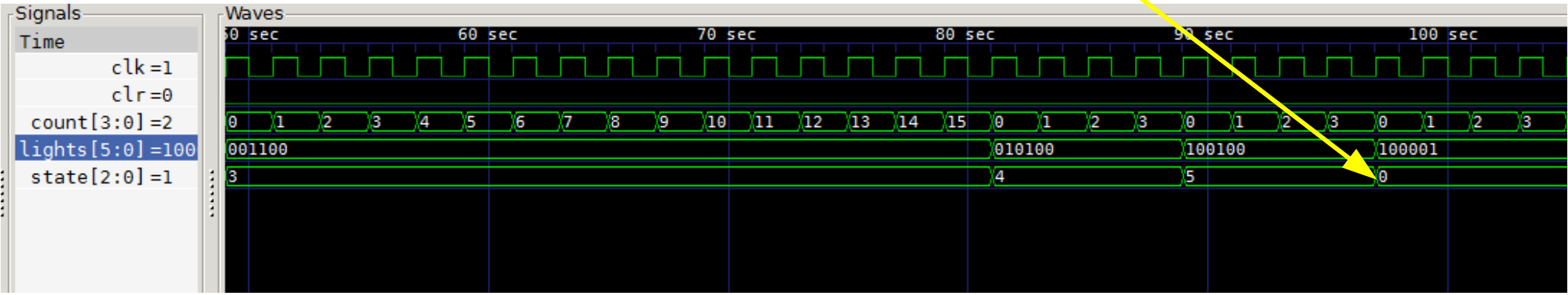
```
> gtkwave test.vcd
```

A szimuláció eredménye



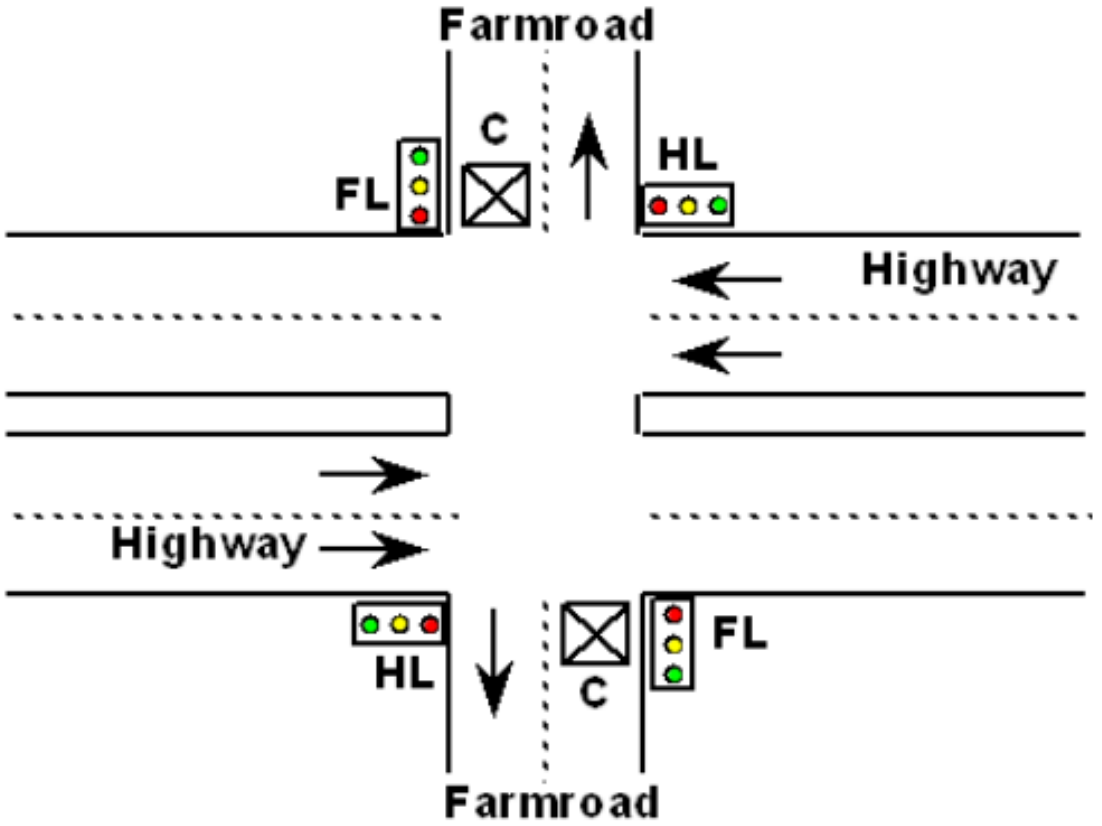
Az első váltás

Egy körülfordulási ciklus



Állapotgépek összekapcsolása

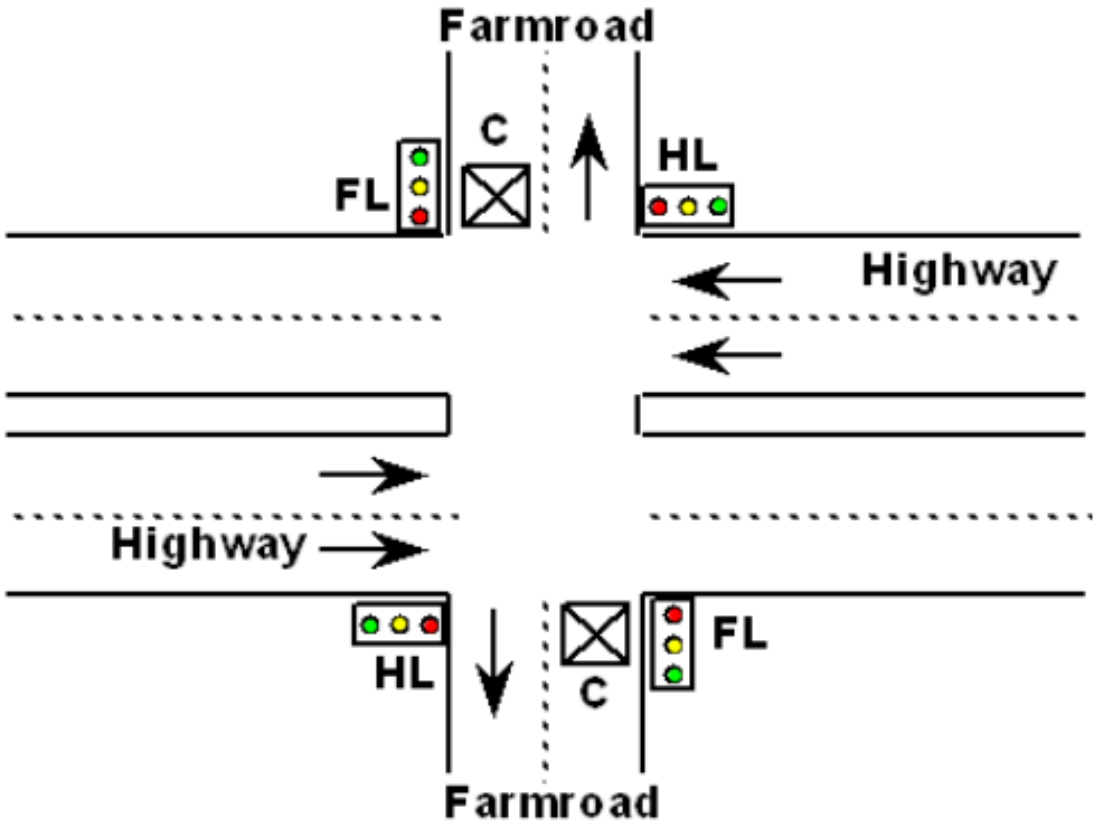
- A következő mintapéldát az University of Washington Computer Science & Engineering, CSE370 kurzusának **23. előadásából** vettük
- Ebben egy olyan útkereszteződéshez kell lámpavezérlést terveznünk, amelyben az alárendelt út csak akkor kap zöld jelzést, ha az úttestve épített járműdetektorok forgalmat észlelnek.



Közlekedési lámpa feltételes állapotváltással

- Egy főútvonalat (highway) egy alárendelt út (farmroad) keresztez.
- Detektorok érzékelik az alsóbbrendű úton várakozó járműveket.
- Ha nincs várakozó jármű, a főútvonal folyamatosan zöld jelzést kap
- Ha van várakozó jármű, a főút zöld jelzése sárgára, majd pirosra vált, hogy az alsóbbrendű úton várakozó járművek áthaladhassanak.
- Az alsóbbrendű úton a jelzés csak addig lesz zöld, amíg járművet érzékelünk, de akkor is csak egy előre meghatározott ideig. Ezután a lámpa zöldről sárgára majd pirosra vált, a főútvonalon pedig zöldre.
- Ha az alsóbbrendű úton van várakozó jármű, a főúton akkor sem lehet rövidebb a zöld jelzés, mint egy előre megadott időtartam.
- Legyen a sárga jelzések ideje 5 órajel ciklus!
- Legyen a zöld jelzés *legalább* 20 ciklus a főúton és *legfeljebb* ugyanennyi az alsóbbrendű úton!

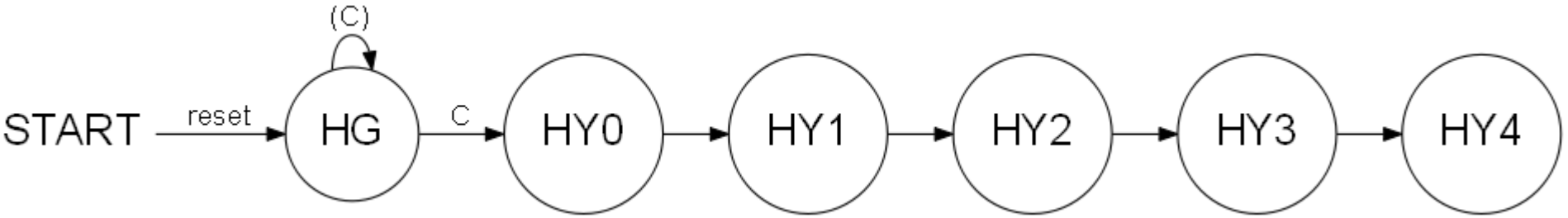
Közlekedési lámpa vezérlése



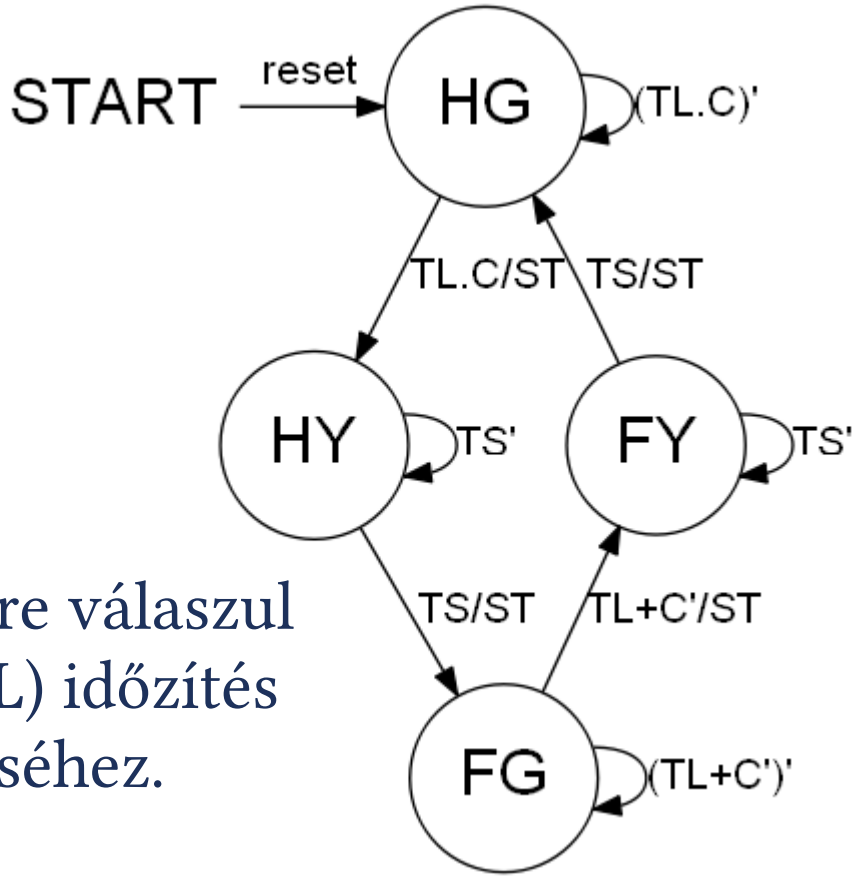
- Kimenetek: **HL** [2:0] – a főút lámpái, **FL** [2:0] – az alárendelt út lámpái
- Bemenetek: **C** – a járműdetektor jele, **RESET** – indító jel, **clk_50M** – rendszer órajel ⟶ **clk** – 1 Hz-es időalap

Részfeladatokra bontás

- A triviális megoldás túl bonyolult állapotgéphez vezetne:



- Célszerűbb a lámpák vezérlését és az időzítést különválasztani!
- A lámpák lehetséges állapotai:
  HG – zöld jelzés a főúton
  HY – sárga jelzés a főúton
  FG – zöld jelzés a mellékúton
  FY – sárga jelzés a mellékúton



- Az időzítő állapotgép az **ST** indítójelre válaszul jelzi a rövid (**TS**), illetve a hosszú (**TL**) időzítés leteltét a sárga, ill. zöld jelzés időzítéséhez.

Állapotvezérlő táblázat

- A HL, TL lámpa állapotok kódolása: Green = 001, Yellow= 010, Red = 100
- A kimenő jelek és az állapotgép állapotai között kölcsönösen egyértelmű kapcsolat van, ezért nincs szükség rá, hogy ezeket külön-külön tároljuk!

state = {HL, FL}

```
parameter highwaygreen   = 6'b001100;
parameter highwayyellow  = 6'b010100;
parameter farmroadgreen  = 6'b100001;
parameter farmroadyellow = 6'b100010;
```

Bemenetek			Jelenlegi állapot	Következő állapot	Kimenetek		
C	TL	TS			ST	HL	FL
0	–	–	HG	HG	0	Green	Red
–	0	–	HG	HG	0	Green	Red
1	1	–	HG	HY	1	Green	Red
–	–	0	HY	HY	0	Yellow	Red
–	–	1	HY	FG	1	Yellow	Red
1	0	–	FG	FG	0	Red	Green
0	–	–	FG	FY	1	Red	Green
–	1	–	FG	FY	1	Red	Green
–	–	0	FY	FY	0	Red	Yellow
–	–	1	FY	HG	1	Red	Yellow

Verilog kód (traffic.v)

- Be- és kimenetek deklarálása, állapotok specifikálása, állapotbitek és a kimenetek összekapcsolása

```
module FSM(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, clk);
output HR;
output HY;
output HG;
output FR;
output FY;
output FG;
output ST;
input TS;
input TL;
input C;
input reset;
input clk;
reg [6:1] state;
reg ST;
```

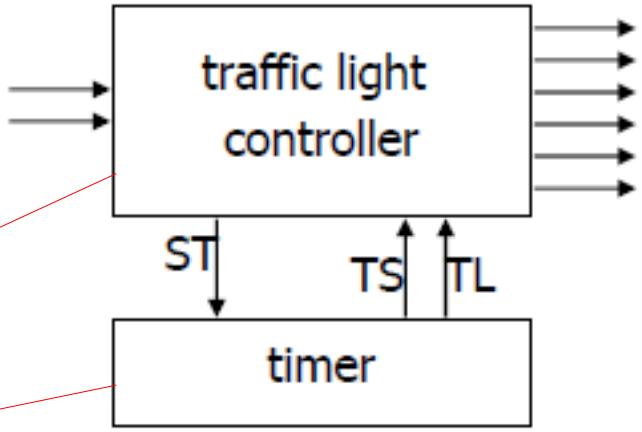
```
parameter highwaygreen = 6'b001100;
parameter highwayyellow = 6'b010100;
parameter farmroadgreen = 6'b100001;
parameter farmroadyellow = 6'b100010;
assign HR = state[6];
assign HY = state[5];
assign HG = state[4];
assign FR = state[3];
assign FY = state[2];
assign FG = state[1];
```

Verilog kód (traffic.v)

```
initial begin state = highwaygreen; ST = 0; end
always @(posedge clk)
  begin
    if (reset)
      begin state = highwaygreen; ST = 1; end
    else begin
    ST = 0;
    case (state)
      highwaygreen:
        if (TL & C) begin state = highwayyellow; ST = 1; end
      highwayyellow:
        if (TS) begin state = farmroadgreen; ST = 1; end
      farmroadgreen:
        if (TL | !C) begin state = farmroadyellow; ST = 1; end
      farmroadyellow:
        if (TS) begin state = highwaygreen; ST = 1; end
    endcase
  end
end
endmodule
```

Verilog kód (traffic.v)

```
module Timer(TS, TL, ST, clk);
  output TS;
  output TL;
  input ST;
  input clk;
  integer value;
  assign TS = (value > 4);          // 5 ciklus reset után
  assign TL = (value > 19);         // 20 ciklus reset után
  always @(posedge ST) value = 0;   // aszinkron reset
  always @(posedge clk) value = value + 1;
Endmodule
```




```
module main(HR,HY,HG,FR,FY,FG,reset,C,clk);
  output HR,HY,HG,FR,FY,FG;
  input reset, C, clk;
  Timer part1(TS, TL, ST, clk);
  FSM   part2(HR,HY,HG,FR,FY,FG,ST,TS,TL,C,reset,clk);
endmodule
```

A próbapad kódja (traffic_tb.v)

- A próbapad kódjába beépítettük az előző oldali main modul kódját

```
`include "traffic.v"
module test();
wire HR, HY, HG, FR, FY, FG, ST;
reg reset, C, clock;
reg [2:0] HL;
reg [2:0] FL;

Timer part1(TS, TL, ST, clock);
FSM part2(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, clock);

initial begin
    $dumpfile("test.vcd");
    $dumpvars(0,test);
    reset = 1'b0; C = 1'b0; clock = 1'b0;
    $monitor("TIME = %d, reset = %b, C = %b, ST = %b,
              HL = %3b, FL = %3b",$time,reset,C,ST,HL,FL);
    #1 reset = 1;
    #5 reset = 0;
    #150 C = 1;
    #250 C = 0;
    #260 C = 1;
    #140 C = 0;
    #200 $finish;
end

always  #1 clock = ~clock;

always @ (HR,HY,HG,FR,FY,FG) begin
    HL = {HR,HY,HG};
    FL = {FR,FY,FG};
end
endmodule
```

Iverilog szimuláció (trafficlights2)

- A szimulációt az alábbi parancsokkal futtathatjuk:

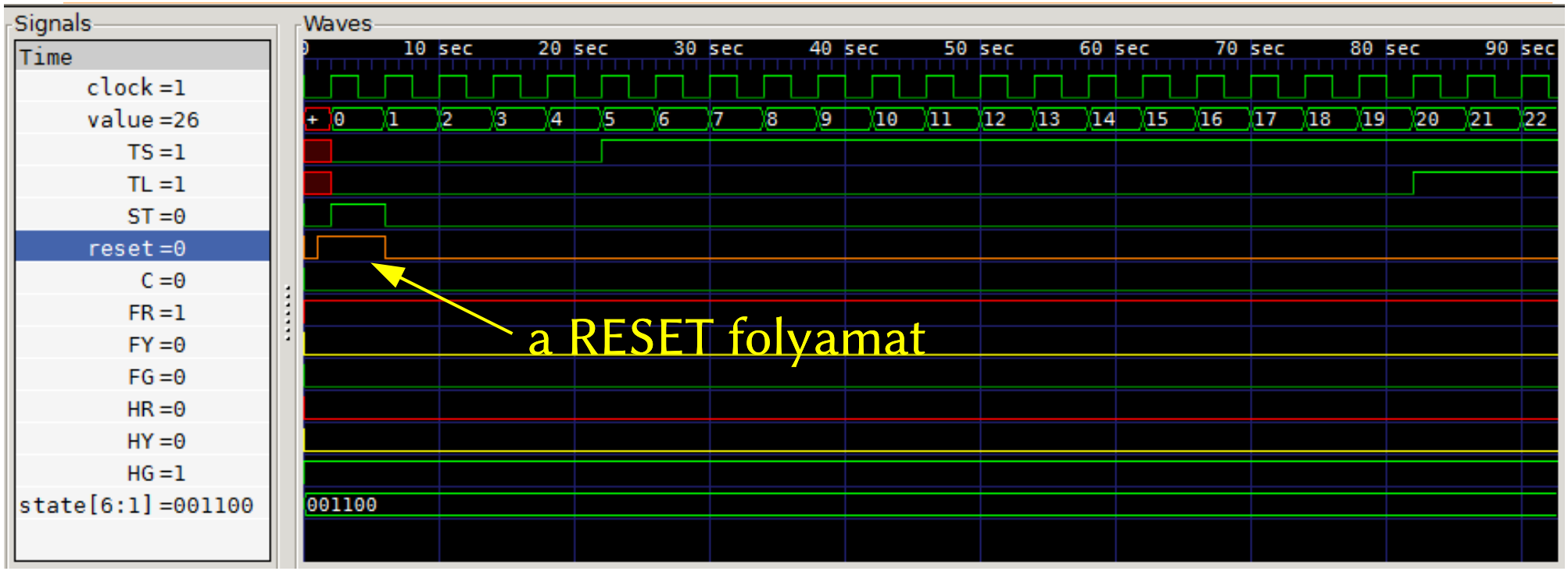
```
> PATH = %PATH%; C:\iverilog\bin; C:\iverilog\gtkwave\bin
> iverilog -o test traffic_tb.v
> vvp test
```

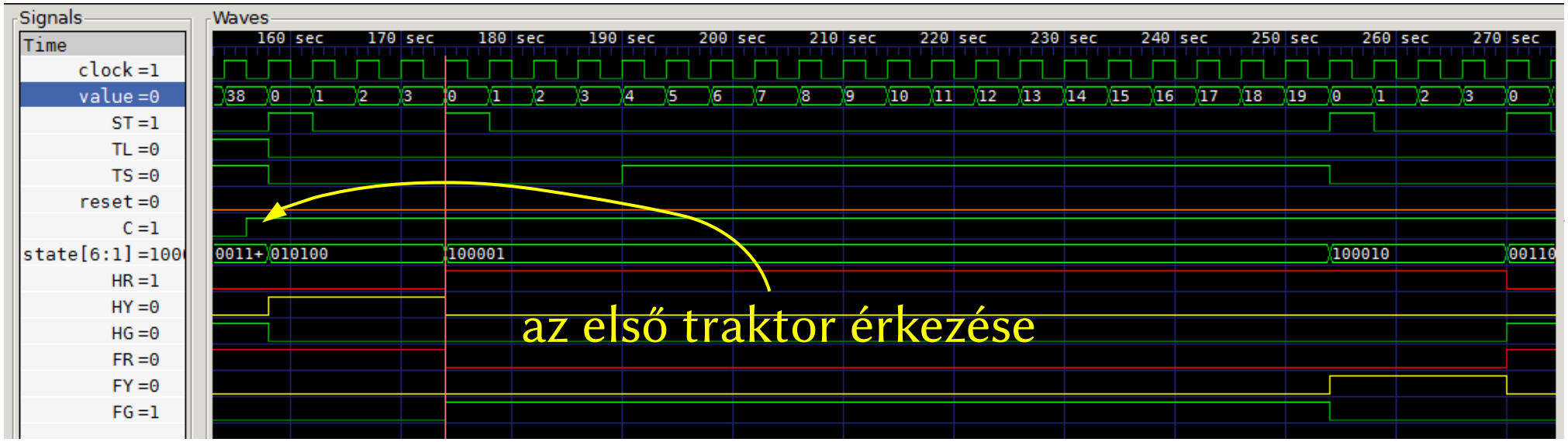
Automatikusan becsatolja a **traffic.v** állományt is!

```
VCD info: dumpfile test.vcd opened for output.
TIME =                     0, reset = 0, C = 0, ST = 0, HL = 001, FL = 100
TIME =                     1, reset = 1, C = 0, ST = 0, HL = 001, FL = 100
TIME =                     2, reset = 1, C = 0, ST = 1, HL = 001, FL = 100
TIME =                     6, reset = 0, C = 0, ST = 0, HL = 001, FL = 100
TIME =                   156, reset = 0, C = 1, ST = 0, HL = 001, FL = 100
TIME =                   158, reset = 0, C = 1, ST = 1, HL = 010, FL = 100
TIME =                   162, reset = 0, C = 1, ST = 0, HL = 010, FL = 100
TIME =                   182, reset = 0, C = 1, ST = 1, HL = 100, FL = 001
TIME =                   186, reset = 0, C = 1, ST = 0, HL = 100, FL = 001
TIME =                   262, reset = 0, C = 1, ST = 1, HL = 100, FL = 010
TIME =                   266, reset = 0, C = 1, ST = 0, HL = 100, FL = 010
TIME =                   286, reset = 0, C = 1, ST = 1, HL = 001, FL = 100
TIME =                   290, reset = 0, C = 1, ST = 0, HL = 001, FL = 100
TIME =                   366, reset = 0, C = 1, ST = 1, HL = 010, FL = 100
TIME =                   370, reset = 0, C = 1, ST = 0, HL = 010, FL = 100
TIME =                   390, reset = 0, C = 1, ST = 1, HL = 100, FL = 001
…
```

```
> gtkwave test.vcd
```

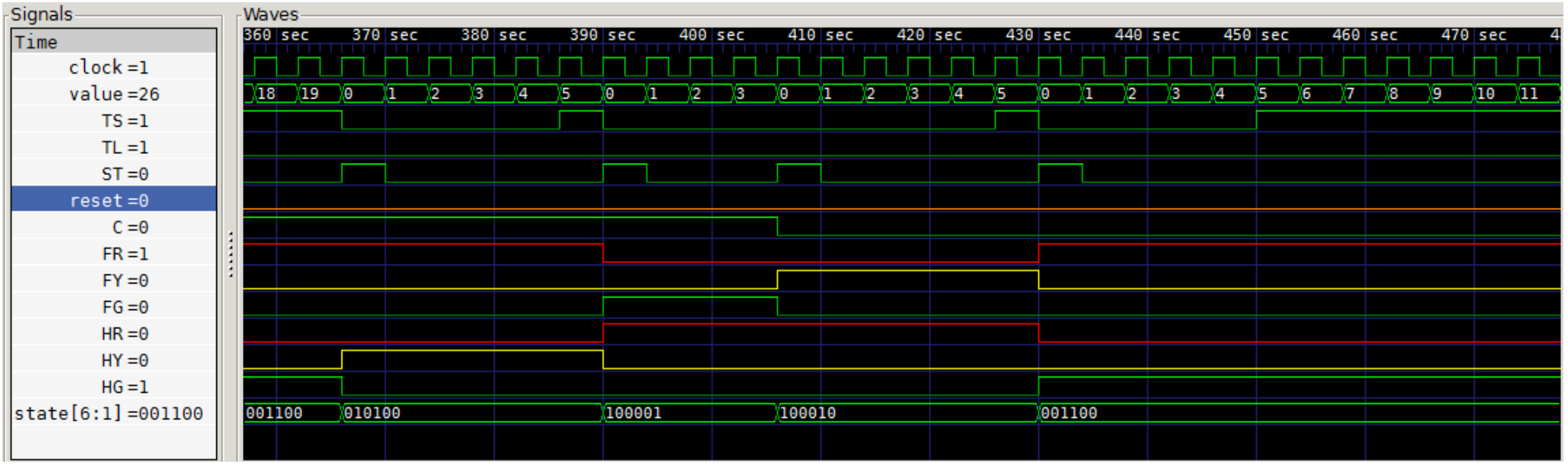
A szimuláció eredménye (trafficlights2)





A szimuláció eredménye (trafficlights2)

- Az utolsó jármű hamarabb áthalad, mint a farmút zöld jelzésének maximális időtartama, ezért az **FG** állapotból hamarabb kilépünk a **C** jel megszűnése miatt
- Megfigyelhető, hogy az ***always*** ciklusok versengése miatt a sárga jelzések néhol 4, máshol 6 órajel ciklusig tartanak. Ez az átgondolatlan tervezés következménye.



Trafficligh2 a C-M240 kártyán

- Az előző mintapéldát Quartus projektként is bemutatjuk
- A projektben egy Verilog HDL állományt hozunk létre **traffic.v** néven (az előzőekben bemutatott kódot meg kell változtatni)
- Ki- és bemenetek kivezetésekhez rendelése az alábbi táblázat szerint. K1 adja a reset, K4 pedig a C járműdetektor jelet.

JEL	Irány	Kivezetés	Eszköz
clk_50M	input	PIN_12	50MHz oszcillátor
nReset	input	PIN_29	K1 nyomógomb
nC	input	PIN_26	K4 nyomógomb
HR	output	PIN_58	D37 LED
HY	output	PIN_57	D36 LED
HG	output	PIN_56	D35 LED
FR	output	PIN_53	D32 LED
FY	output	PIN_52	D31 LED
FG	output	PIN_51	D30 LED

traffic.v 1 / 4

```
module FSM(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, clock);
output HR, HY, HG, FR, FY, FG, ST;
input TS, TL, C, reset, clock;

reg [6:1] state;
reg ST;

parameter highwaygreen = 6'b001100;
parameter highwayyellow = 6'b010100;
parameter farmroadgreen = 6'b100001;
parameter farmroadyellow = 6'b100010;

assign HR = !state[6];    // Negation required due to the
assign HY = !state[5];    // negative I/O logic of the C-M240 card
assign HG = !state[4];
assign FR = !state[3];
assign FY = !state[2];
assign FG = !state[1];

initial begin
    state = highwaygreen;
    ST = 0;
end
```

Változás: negáljuk a kimeneteket

traffic.v 2 / 4

```
always @(posedge clock) begin
    if (reset) begin
        state = highwaygreen;
        ST = 1;
    end
    else begin
        ST = 0;
        case (state)
            highwaygreen:
                if (TL & C) begin
                    state = highwayyellow;
                    ST = 1;
                end
            highwayyellow:
                if (TS) begin
                    state = farmroadgreen;
                    ST = 1;
                end
            farmroadgreen:
                if (TL | !C) begin
                    state = farmroadyellow;
                    ST = 1;
                end
            farmroadyellow:
                if (TS) begin
                    state = highwaygreen;
                    ST = 1;
                end
        endcase
    end
end
endmodule
```

Ebben a részben nincs változás!

traffic.v 3 / 4

```
module Timer(TS, TL, ST, clock);
output TS;
output TL;
input ST;
input clock;
reg [7:0] value;

assign TS = (value > 4);        //  5 cycles after reset
assign TL = (value > 19);       // 20 cycles after reset

always @(posedge clock, posedge ST)
  if(ST==1) value = 0;
  else value = value + 1;
endmodule
```

Változások:

- value reg típusú legyen
- ST esemény kezelése ne legyen külön blokkban

traffic.v 4 / 4

```
module main(HR, HY, HG, FR, FY, FG, nReset, nC, clk_50M,C,reset);
  output HR, HY, HG, FR, FY, FG, C, reset;
  input nReset, nC, clk_50M;
  reg clock;
  reg [24:0] count = 0;
  assign C = !nC;                 // Negation required due to the
  assign reset = !nReset;         // negative I/O logic of the C-M240 card

  Timer part1(TS, TL, ST, clock);
    FSM part2(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, clock);

  //--- Clock signal generation --------------------
  always @ (posedge clk_50M)
    begin
      if (count == 25000000)
        begin                     // our clock is of 50,000,000 Hz
                                  // We now use the count counter 25,000,000 cycles
          clock <= ~clock;        // so that we get a signal with 1 s period (1 Hz).
          count <= 0;             // clear the counter.
        end
      else
        count <= count + 1;       // counter from the increase.
    end

endmodule
```

Változások:

- Bemenetek negálása (reset, C)
- Frekvenciaosztó (1 : 25 000 000)