Digitális Technika

Dr. Oniga István
Debreceni Egyetem, Informatikai Kar

9. Laboratóriumi gyakorlat

- Számlálók
  - Szimulációk:
    - 4 bites szinkron felfelé számláló aszinkron (vagy szinkron) törléssel
    - 4 bites szinkron lefelé számláló szinkron törléssel
    - 4 bites szinkron fel/le számláló
    - Decimális felfelé számláló, tölthető kezdő értekkel
    - N-bites (generikus) szinkron felfelé számláló aszinkron törléssel
  - Implementációk
    - Órajel osztás
    - 8 bites számláló LED-eken
    - 4 bites számláló 7 szegmenses kijelzőn

Lab9_1-Lab9_5: Szinkron számlálók

A következő feladat leírás a Lab9_x feladatokra vonatkozik (x = 1:5)

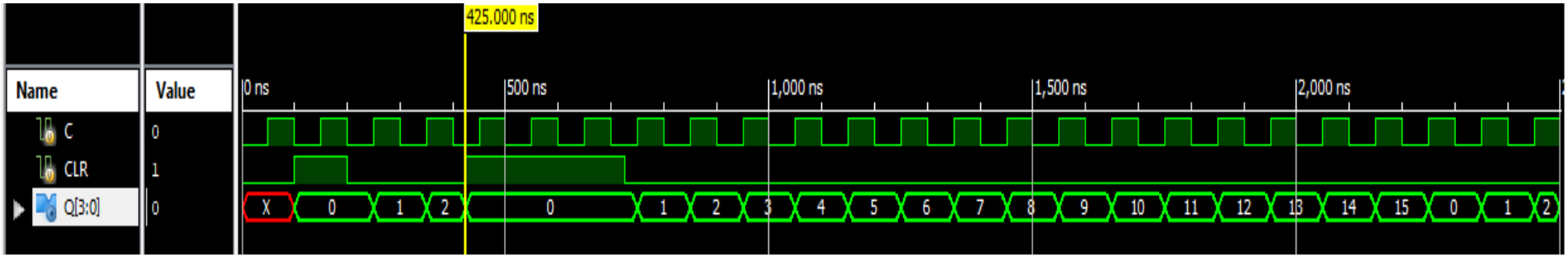
- Hozunk létre egy új projektet (Lab9_x)
- Adjunk hozzá egy új VERILOG forrásfájlt (Lab9_x.v).
- Specifikáljuk számláló működését.
- Adjunk hozzá egy Verilog test fixture fájlt.
- Gerjesztő jelek specifikálása
- Funkcionális kód ellenőrzése szimulációval.

Lab9_1a: 4 bites szinkron felfelé számláló aszinkron törléssel

```
//
// 4-bit up counter with an asynchronous clear.
//
module counter_1 (input C, CLR, output reg
[3:0] Q);

      always @(posedge C or posedge CLR)
      begin
            if (CLR)
                        Q <= 4'b0000;
            else
                        Q <= Q + 1'b1;
      end

endmodule
```

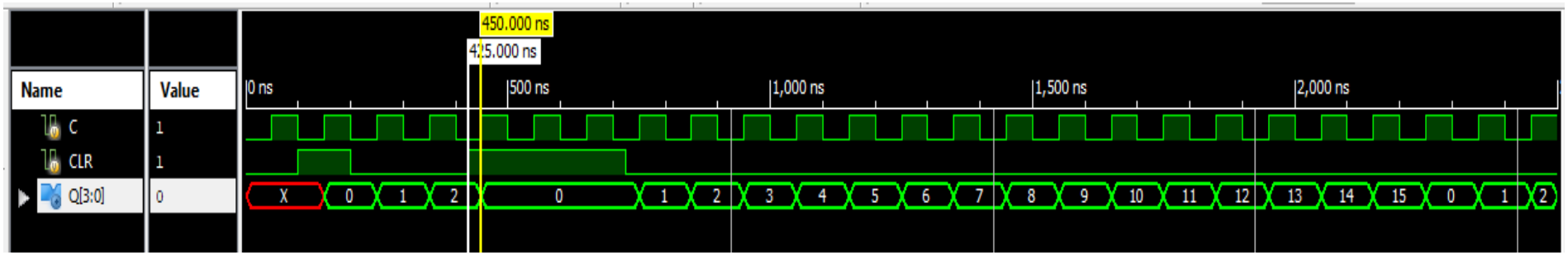
Gerjesztő jelek specifikálása

```
initial begin // Initialize Inputs
C = 0;
CLR = 0;
// Wait 100 ns for global reset to finish
#100;
// Add stimulus here
CLR = 1;
#100;
CLR = 0;
# 225
CLR = 1;
# 300
CLR = 0 ;
end
always #50
C <= ~ C;
```



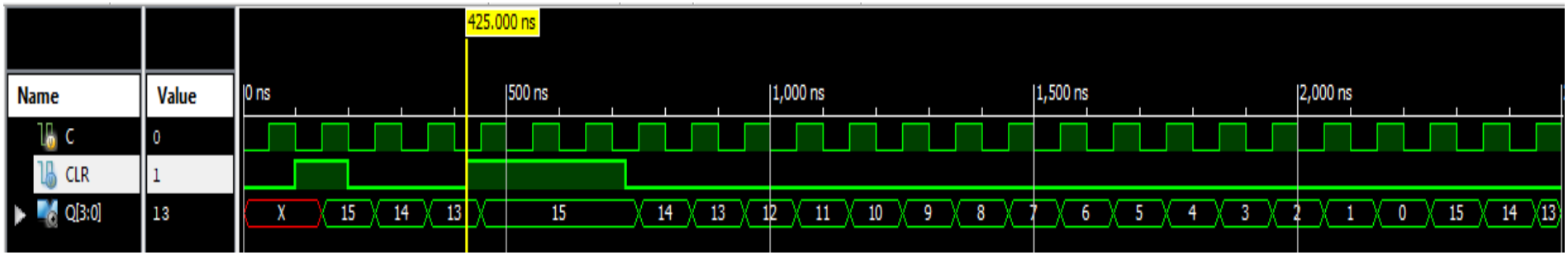
Lab9_1b: 4 bites szinkron felfelé számláló szinkron törléssel

```
//
// 4-bit up counter with an synchronous clear.
//
module counter_2 (input C, CLR, output reg [3:0] Q);

     always @(posedge C)
     begin
          if (CLR)
                    Q <= 4'b0000;
          else
                    Q <= Q + 1'b1;
     end

endmodule
```



Lab9_2: 4 bites szinkron lefelé számláló szinkron törléssel

```
module counter_3 (input C, CLR, output reg [3:0] Q);
     always @(posedge C)
     begin
          if (CLR)
                    Q <= 4'b1111;
          else
                    Q <= Q - 1'b1;
     end
endmodule
```

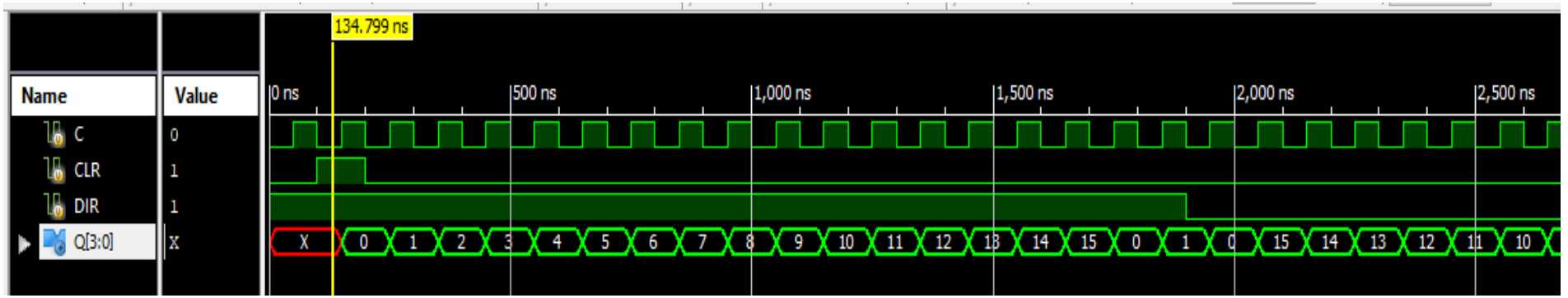


Lab9_3: 4 bites szinkron fel/le számláló

```
module counter_3 (input C, CLR, DIR, output
reg [3:0] Q);
      always @(posedge C)
      begin
            if (CLR)
                  Q <= 4'b0000;
            else if (DIR)
                  Q <= Q + 1'b1;
            else
                  Q <= Q - 1'b1;
      end
endmodule
```

Gerjesztő jelek specifikálása

```
initial begin
    // Initialize Inputs
    C = 0;
    CLR = 0;
    DIR = 1;

    // Wait 100 ns for global reset to finish
    #100;
    // Add stimulus here
    CLR = 1;
    #100;
    CLR = 0;
    #1700;
    DIR=0;

end

always #50
    C <= ~ C;
```

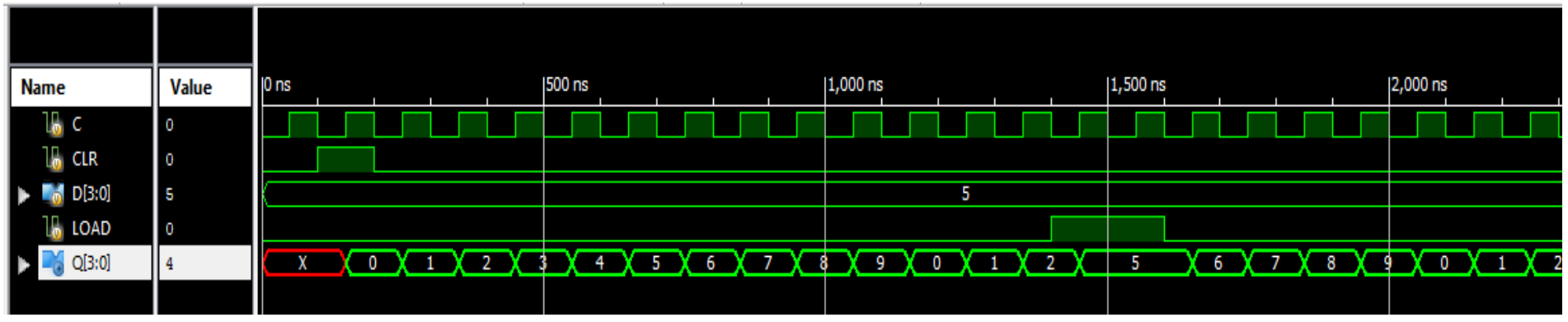


Lab9_4: Decimális felfelé számláló, tölthető kezdő értekkel

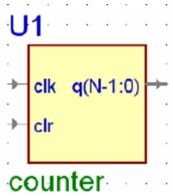
Gerjesztő jelek specifikálása

```
module counter_5 (input C, CLR, LOAD, input [3:0]
D, output reg [3:0] Q);

assign q9 = (Q== 4'd9); //assign q12 = (Q== 4'd12);

always @(posedge C)
          begin
          if (CLR | q9)
                    Q <= 4'b0000;
          else if (LOAD) // (LOAD==1)
                    Q <= D; //vagy konstanssal;
          else
                    Q <= Q + 1'b1;
          end
endmodule
```

```
initial begin
    // Initialize Inputs
    C = 0;
    CLR = 0;
    LOAD = 0;
    D = 4'b0101;;

    // Wait 100 ns for global reset
    #100;
    // Add stimulus here
    CLR = 1;
    #100;
    CLR = 0;
    #1200;
    LOAD = 1;
    # 200;
    LOAD = 0;

end

always #50
C <= ~ C;
```



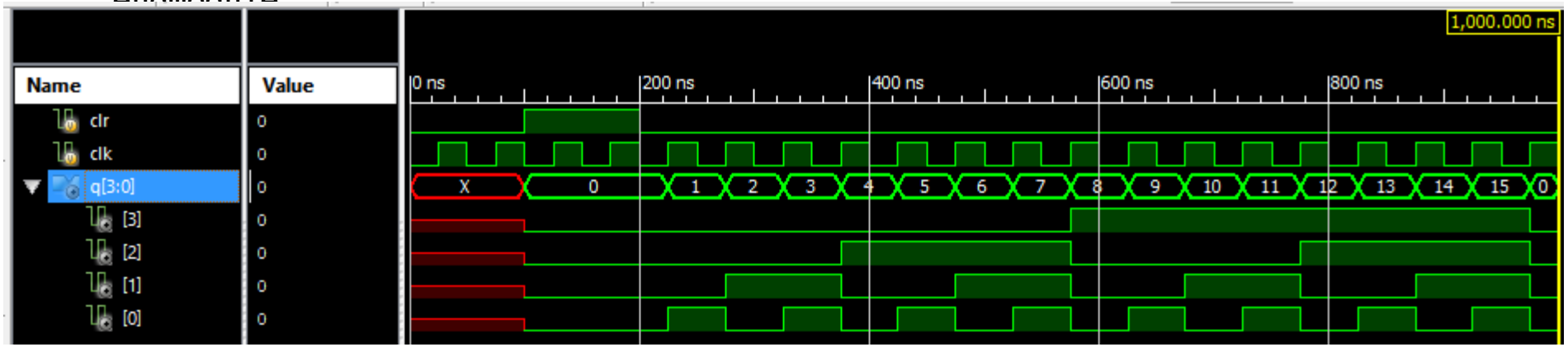
Lab9_5: N-bites (generikus) felfelé számláló




```
module counter
#(parameter N = 4)
   (input wire clr , clk ,
   output reg [N-1:0] q );
// N-bit counter

always @(posedge clk or posedge
clr)
   begin
      if(clr == 1)
         q <= 0;
      else
         q <= q + 1;
   end
endmodule
```

Gerjesztő jelek specifikálása

```
initial begin
   // Initialize Inputs
   clr = 0;
   clk = 0;

   // Wait 100 ns for global reset to finish
   #100 clr = 1;
   #100 clr = 0;

   // Add stimulus here

end
   always #25
   clk <= ~clk;
```

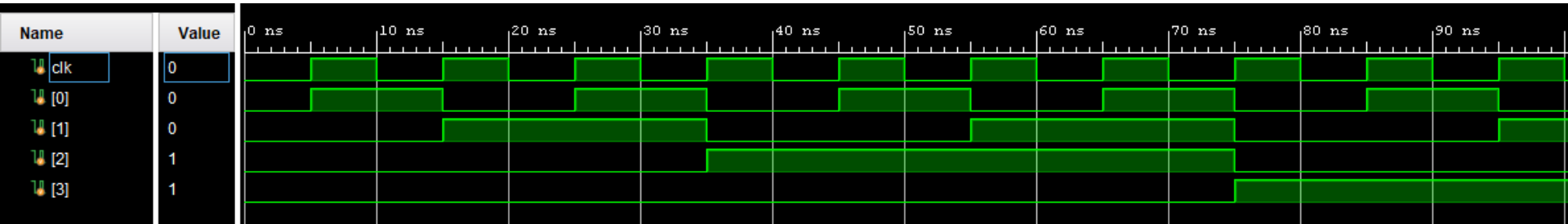


Lab9_6: Órajel osztó

- Nexys-4 board-nak egy 100 MHz órajele van (clk100mhz)
- A későbbi projektek során 3 féle különböző órajelre lesz szükség
- Ezeket az alap órajelek osztásával érjük el
  - mindegyik FF 2-vel oszt: q[0]= clk/2; q[1]= q[0]/2 …
  - Egy 27 bites számlálót használunk az órajel osztásra

```
module clkdiv ( input clk, clr, output clk1, clk2, clk3 );
reg [26:0] q;
// 27-bit counter
always @(posedge clk or posedge clr)
     begin
          if(clr == 1)
               q <= 0;
          else
               q <= q + 1;
          end
assign clk1 = q[26]; // ~0.75 Hz
assign clk2 = q[25]; // ~1.5
assign clk3 = q[24]; // ~3 Hz
endmodule
```

Q(i)	Frecquency (Hz)	Period (ms)
0	50000000.00	0.00002
1	25000000.00	0.00004
2	12500000.00	0.00008
3	6250000.00	0.00016
4	3125000.00	0.00032
5	1562500.00	0.00064
6	781250.00	0.00128
7	390625.00	0.00256
8	195312.50	0.00512
9	97656.25	0.01024
10	48828.13	0.02048
11	24414.06	0.04096
12	12207.03	0.08192
13	6103.52	0.16384
14	3051.76	0.32768
15	1525.88	0.65536
16	762.94	1.31072
17	381.47	2.62144
18	190.73	5.24288
19	95.37	10.48576
20	47.68	20.97152
21	23.84	41.94304
22	11.92	83.88608
23	5.96	167.77216
24	2.98	335.54432
25	1.49	671.08864
26	0.745	1342.17728



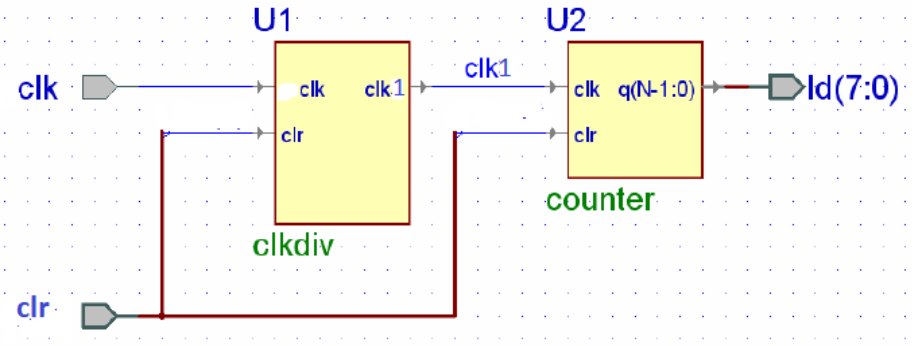
Lab9_6b: Órajel osztó implementáció

- Hozzuk létre a projekthez tartozó ucf fájlt
  - Inputok
    - clk -> clk100mhz, clr -> btnc,
  - Outputok
    - clk1 -> led<0>, clk2 -> led<1>, clk3 -> led<2>.

```
## Clock signal
NET "clk"   LOC = "E3"  | IOSTANDARD = "LVCMOS33"; #Bank = 35, Pin name =
NET "clk" TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100 MHz HIGH 50%;

## Buttons
NET "clr"           LOC=N17 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_14

## LEDs
NET "clk1"         LOC=H17 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A24_15
NET "clk2"         LOC=K15 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_RS1_15
NET "clk3"         LOC=J13 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A25_15
```

- Konfigurációs fájl generálása, letöltése és a működés tesztelése a kártyán

Lab9_7: 8 bites számláló LED-eken




```
module count8_top (input clk, input  clr,
output [7:0] led) ;

wire clk1;

clkdiv
U1(.clk1(clk1),.clr(clr),.clk(clk));

counter #( .N(8)) U2
(.clk(clk1), .clr(clr), .q(led[7:0]));
endmodule
```

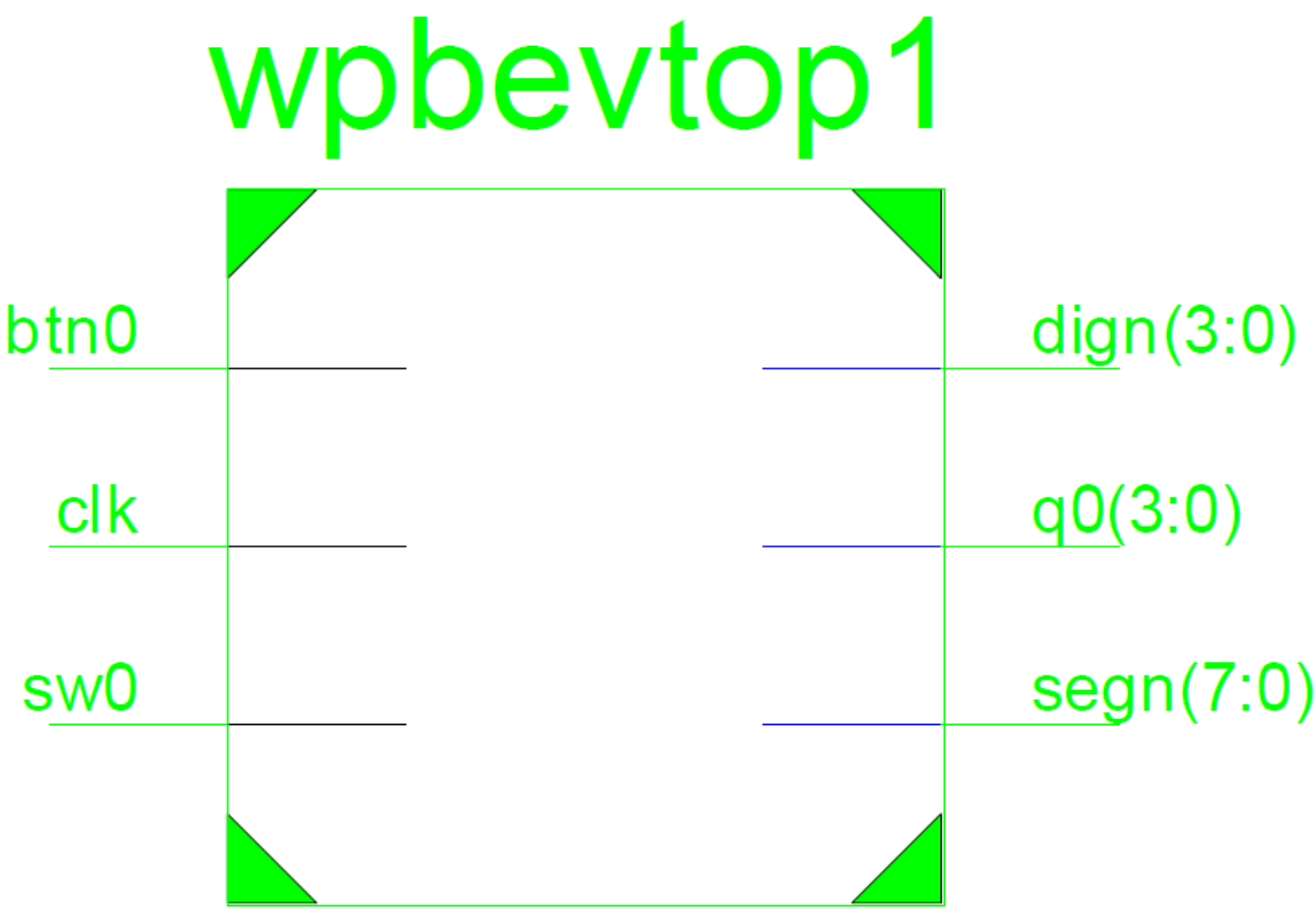
- Hozunk létre egy új projektet (Lab9_7)
- Adjunk hozzá az előző feladatokhoz elkészült clkdiv és counter modulokat.
- Adjunk hozzá egy új VERILOG forrásfájlt (Lab9_7.v) ez lesz a top modul ami össze köti az előző két modult a fenti ábra szerint.
- Hozzuk létre a projekthez tartozó ucf fájlt
  - Inputok: clk -> clk100mhz, clr -> btnc,
  - Outputok: led -> led.
- Konfigurációs fájl generálása, letöltése és a működés tesztelése a kártyán

Lab9_8: 4 bites számláló 7 szegmenses kijelzőn

- Hozunk létre egy új projektet (Lab9_8)
- Adjunk hozzá az előző feladatokhoz elkészült clkdiv és counter modulokat.
- Adjuk hozzá a hex7seg modult amit az előző laborokhoz használtunk.
- Adjunk hozzá egy új VERILOG forrásfájlt (Lab9_8.v) ez lesz a top modul ami össze köti az előző három modult.
- Hozzuk létre a projekthez tartozó ucf fájlt
  - Inputok: clk, clr,
  - Outputok: an, a_to_g, dp, led.
- Konfigurációs fájl generálása, letöltése és a működés tesztelése a kártyán

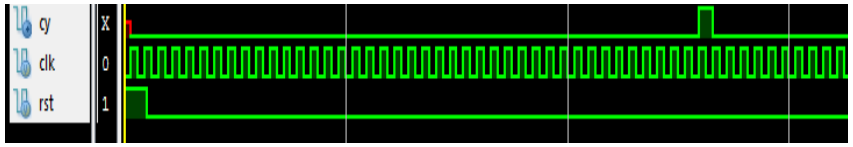
```
module count4_top (input clk, input  clr, output [7:0] led,
          output [6:0] a_to_g, output [7:0] an, output dp )
;

wire clk1;

clkdiv U1(.clr(clr),.clk(clk), .clk1(clk1));
counter #( .N(8)) U2 (.clk(clk1), .clr(clr), .q(led[7:0]));
hex7seg U3 (.x(led[3:0]), .a_to_g(a_to_g));

assign an = 8'b11111110;
assign dp = 1; // dp off

endmodule
```

Másodperc számláló



Rategen modul

```
21  module rategen(
22  input clk,rst,
23  output cy
24  );
25  //Generate 1 clock wide pulse on output CY
26  reg [29:0 ] Q;
27  always @(posedge clk)
28  begin
29  if (rst | cy)
30  Q <= 0;
31  else
32  Q <= Q + 1;
33  end
34  assign cy = (Q == 99999999 );
35  //assign cy = (Q == 4);
36  endmodule
```



Másodperc számláló

count_sec.v

```
1   module count_sec(
2       input clk,
3       input rst,
4       input ce,
5       input dir,
6       output [3:0] q,
7       output eq9
8       );
9
10  reg [3:0] cntr_d0;
11  wire cntr_d0_eq0, cntr_d0_eq9;
12
13      always @(posedge clk)
14          if (rst)
15              cntr_d0 <= 0;
16          else if (ce)
17              if (dir) //DIR=1: count up
18                  if (cntr_d0_eq9)
19                      cntr_d0 <= 0; //overflow
20                  else
21                      cntr_d0 <= cntr_d0 + 1;
22              else //DIR=0: count down
23                  if (cntr_d0_eq0)
24                      cntr_d0 <= 9;
25                  else
26                      cntr_d0 <= cntr_d0 - 1;
27
28      assign cntr_d0_eq0 = (cntr_d0 == 0);
29      assign cntr_d0_eq9 = (cntr_d0 == 9);
30
31      assign q = cntr_d0;
32      assign eq9 = cntr_d0_eq9;
33
34  endmodule
35
```

hex2seg modul

```
1
2  module hex2seg(hex,seg);
3      input [3:0] hex;
4      output [6:0] seg;
5
6  reg [6:0] seg;
7   always @(hex)
8   begin
9        case (hex)
10           4'b0001 : seg = 7'b1111001;   // 1
11           4'b0010 : seg = 7'b0100100;   // 2
12           4'b0011 : seg = 7'b0110000;   // 3
13           4'b0100 : seg = 7'b0011001;   // 4
14           4'b0101 : seg = 7'b0010010;   // 5
15           4'b0110 : seg = 7'b0000010;   // 6
16           4'b0111 : seg = 7'b1111000;   // 7
17           4'b1000 : seg = 7'b0000000;   // 8
18           4'b1001 : seg = 7'b0010000;   // 9
19           4'b1010 : seg = 7'b0001000;   // A
20           4'b1011 : seg = 7'b0000011;   // b
21           4'b1100 : seg = 7'b1000110;   // C
22           4'b1101 : seg = 7'b0100001;   // d
23           4'b1110 : seg = 7'b0000110;   // E
24           4'b1111 : seg = 7'b0001110;   // F
25           default : seg = 7'b1000000;   // 0
26       endcase
27 end
28 endmodule
```

Top modul

```
1   module wpbevtop1(
2       input clk,btn0, sw0,
3   output [3:0] q, output [6:0] segn, output [3:0] dign
4   );
5   reg rst, dir;
6   //wire [3:0] q1;
7   always @(posedge clk)
8   //Synchronize inputs
9      begin
10        rst <= btn0;
11        dir <= sw0;
12     end
13     wire ce;
14  rategen rategenerator(
15        .clk(clk),
16        .rst(rst),
17        .cy(ce)
18        );
19  count_sec counter(
20        .clk(clk),
21        .rst(rst),
22        .ce(ce),
23        .dir(dir),
24        .q(q)
25        );
26  hex2seg segdecoder (
27        .hex(q),
28        .seg(segn)
29        );
30     assign dign = 4'b0000;
31
32  endmodule
```

Négy digites másodpercszámláló

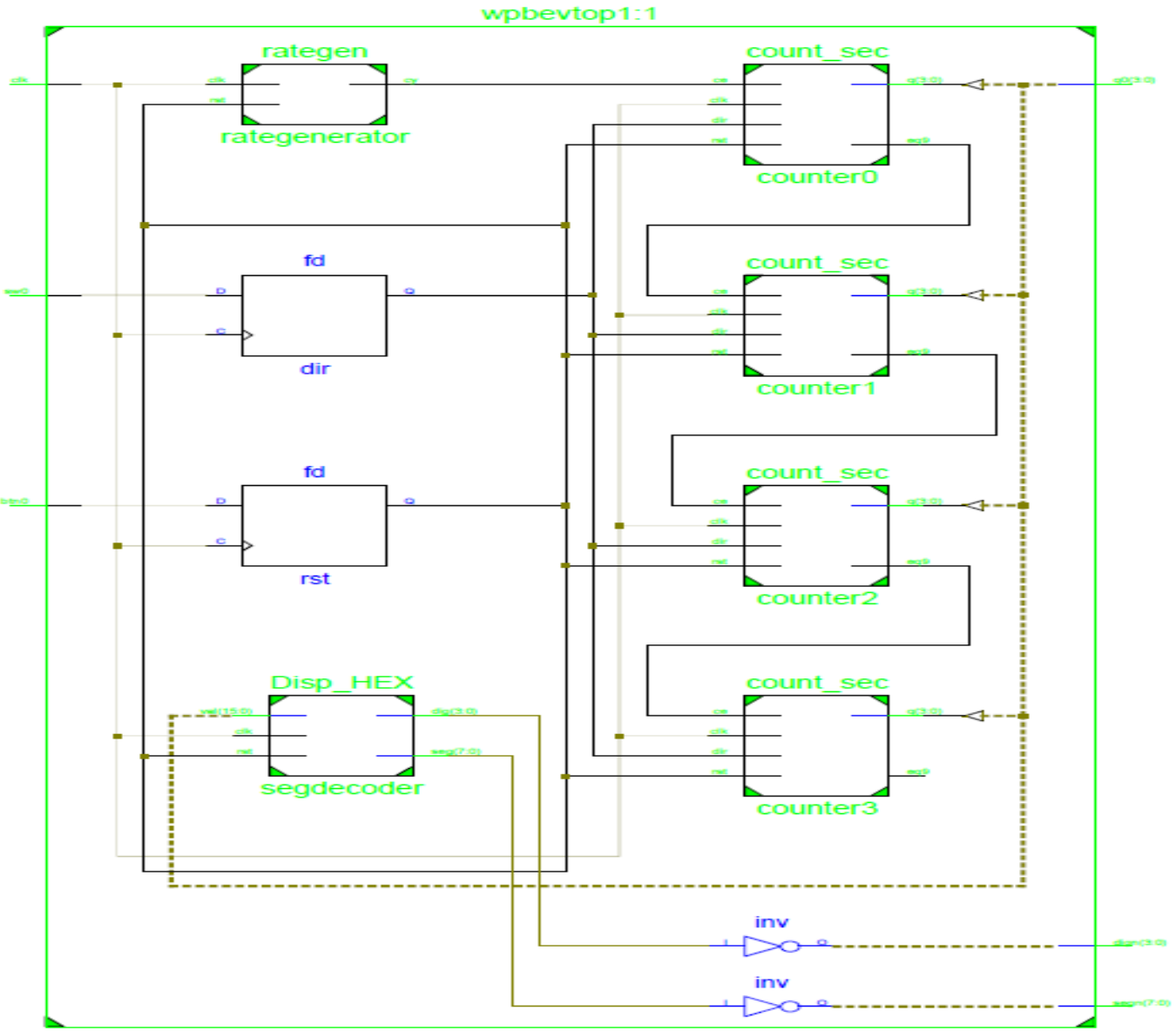
```
// Time multiplexed 4 digit display unit, timing based on 16 MHz system clock
// Input is 16 bit value, output is 4 digit hexadecimal
// Operation: From the high speed clock we create slower dig[3:0] select signals
// From the full 16 bits val[15:0] value four bit hexa digits are selected,
// and converted to 7 segment display code one at a time.
// These are then connected time multiplexed to the seg segment output pins with
// synchron control of the dig digit select signals
// In the module all signals are active high. Physical outputs may need inversion.
// Example of the operation. val = 5678
//       >< val[ 3: 0] >< val[ 7: 4] >< val[11: 8] >< val[15:12] >< val[ 3: 0] ><
//digit          8            7            6            5            8
//convert | digit -> seg| digit -> seg| digit -> seg| digit -> seg| digit -> seg
//seg    ><   1111111  ><   0000111  ><   1111101  ><   1101101  ><    1111111 ><
//dig3____--------------______________________________------------____
//dig2________________--------------______________________________
//dig1____________________________--------------______________________
//dig0____________________________________------------______________

module Disp_HEX(
    input clk,
    input rst,
    input [15:0] val,
    output [7:0] seg,
    output [3:0] dig
    );

// Generate switching signals for time multiplexing
// Timing is not critical, any frequency between 100 Hz and 1000Hz is acceptable
// The 16 bit counter will turn around in 16.000.000/2^16=244,14 Hz, so it is OK
// The counter will be in free run, the two MSB give us the digit timing

reg [15:0] cnt;

always @(posedge clk)                // Allow the counter to free run
   if (rst) cnt <= 16'b0;            // RESET is not important, but simulation is easi
   else     cnt <= cnt+1;

wire [1:0] mpx;                      // Two bit time multplex signal for the 4 phases

assign mpx = cnt[15:14];             // We use the two MSBs of the counter for this pu
```

```
assign dig = 4'b0001<<mpx;           // This is a 2 bit decoder, to generate the selections
                                     // dig[3:0] = 0001  0010  0100  1000  0001  0010  etc.

reg [3:0] digit;                     // Four bit variable for the hexa digits

//wire [15:0] val;
//assign val = 16'b0011001000010000;

always @(*)                          // Select actual bit fields for conversion
   case (mpx)
      2'b00 : digit <= val[ 3: 0];  // dig[0]
      2'b01 : digit <= val[ 7: 4];  // dig[1]
      2'b10 : digit <= val[11: 8];  // dig[2]
      2'b11 : digit <= val[15:12];  // dig[3]
   endcase

reg [7:0] disp;                        // disp is 8 bits to disable decimal points

always @(*)
   case (digit)          //DP_6543210                    --0--
      4'b0000 : disp <= 8'b0_0111111;  // 0             |     |
      4'b0001 : disp <= 8'b0_0000110;  // 1            5     1
      4'b0010 : disp <= 8'b0_1011011;  // 2            |     |
      4'b0011 : disp <= 8'b0_1001111;  // 3             --6--
      4'b0100 : disp <= 8'b0_1100110;  // 4            |     |
      4'b0101 : disp <= 8'b0_1101101;  // 5            4     2
      4'b0110 : disp <= 8'b0_1111101;  // 6            |     |
      4'b0111 : disp <= 8'b0_0000111;  // 7             --3--  DP
      4'b1000 : disp <= 8'b0_1111111;  // 8
      4'b1001 : disp <= 8'b0_1101111;  // 9
      4'b1010 : disp <= 8'b0_1110111;  // A
      4'b1011 : disp <= 8'b0_1111100;  // b
      4'b1100 : disp <= 8'b0_0111001;  // C
      4'b1101 : disp <= 8'b0_1011110;  // d
      4'b1110 : disp <= 8'b0_1111001;  // E
      4'b1111 : disp <= 8'b0_1110001;  // F
   endcase
  assign seg = disp;
endmodule
```

Egy irányú másodperc számláló

```
1  module count_sec(
2      input clk, rst, input ce,
3      output [3:0] q,
4      output eq9
5      );
6
7  reg [3:0] cntr_d0;
8  wire  cntr_d0_eq9;
9
10    always @(posedge clk)
11       if (rst)
12          cntr_d0 <= 0;
13       else if (ce)
14
15             if (cntr_d0_eq9)
16                cntr_d0 <= 0; //overflow
17             else
18                cntr_d0 <= cntr_d0 + 1;
19
20    assign cntr_d0_eq9 = (cntr_d0 == 9);
21
22    assign q = cntr_d0;
23    assign eq9 = cntr_d0_eq9;
24
25 endmodule
```

Top Modul


wpbevtop1:1
rategen
rategenerator
count_sec
counter0
fd
dir
count_sec
counter1
fd
rst
count_sec
counter2
Disp_HEX
segdecoder
count_sec
counter3
inv
inv

Top Modul

```
module wpbevtop1( input clk, btn0, sw0,
output [3:0] q0, output [7:0] segn, output [7:0] dign
);
reg rst;
wire [15:0] hex;
wire [7:0] seg;
wire ce4, ce3, ce2, ce1;
wire [3:0] q3, q2, q1, dig;
always @(posedge clk)
//Synchronize inputs
                begin
                                rst <= (btn0);
                wire ce;
                rategen rategenerator(.clk(clk), .rst(rst),.cy(ce));
                count_sec counter0(.clk(clk), .rst(rst), .ce(ce),  .q(q0), .eq9(ce1));
                count_sec counter1(.clk(clk), .rst(rst), .ce(ce&ce1),  .q(q1), .eq9(ce2));
                count_sec counter2(.clk(clk), .rst(rst), .ce(ce&ce1&ce2),  .q(q2), .eq9(ce3));
                count_sec counter3(.clk(clk), .rst(rst), .ce(ce&ce1&ce2&ce3),  .q(q3), .eq9(ce4));
                Disp_HEX segdecoder ( .clk(clk), .rst(rst), .val(hex), .seg(seg), .dig(dig));

                assign hex = {q3,q2,q1,q0};
                assign segn = ~seg;             // Board signals are active low
                assign dign = {4'b1111, ~dig};          //

endmodule
```