```
*********************************************
Alapok:
*********************************************

// rategen for 6.25 Mhz
//*******************

module rategen(
                input clk,rst,
                output en
                );

reg [2:0] Q;
always @ (posedge clk)
        if(rst|en)
                Q <= 0;
        else
                Q <= Q + 1;
assign en= Q[2];
endmodule


//*****************************************************
// mux
//*****************************************************
module mux_31(
        input in0,in1,in2,in3,
        input [1:0] sel,
        output reg r
        );

always @ (*)
case(sel)
        2'b00: r <= in0;
        2'b01: r <= in1;
        2'b10: r <= in2;
        default: r <= in3;
endcase
endmodule


//*****************************************************
// shift reg
//*****************************************************
module shift_reg(
                input clk,ce,ld,dir,sdin,
                input [15:0] din,
                output [15:0] dout
```

```
                    );

reg [15:0] shr;

always @ (posedge clk)
        if(ce==1)
                if(ld==1)
                        shr <= din;
                else if(dir==1)
                        shr <= {sdin,shr[15:1]};
                else
                        shr <= {shr[14:0],sdin};
assign dout= shr;
endmodule


//*************************************************
Éldetektálás
//*************************************************

reg D_prev;
always @ (posedge clk) D_prev<=D;
wire d_rise,d_fall,d_both;

assign d_rise =~D_prev&D;
assign d_fall =D_prev&~D;
assign d_both =D_prev^D;
```

################
0-9 ig számláló#
################

rategen.v:

```
module rategen(
   input rst,
   input clk,
   output cy
   );

reg [23:0] Q;

always @(posedge clk)
begin
  if (rst | cy)
     Q <= 0;
  else
     Q <= Q + 1;
end
assign cy = (Q == 6000000);
//assign cy = (Q == 4);


endmodule
```

---

countsec.v:

```
module countsec(
   input clk,
   input rst,
   input ce,
   input dir,
   output [3:0] q,
   output [2:0] l
   );

reg [3:0] cntr;
reg [2:0] cntr2;
always @(posedge clk)
if (rst)
begin
  cntr <= 0;
  cntr2 <= 0;
end
else if (ce)
```

```
begin
  if (dir)
  begin
    cntr <= cntr + 1;

  if (cntr == 9)
  begin
    cntr <= 0;
    cntr2 <= cntr2 + 1;
    if (cntr2 == 5)
      cntr2 <= 0;
  end
  end
  else
  begin
    cntr <= cntr - 1;

  if (cntr == 0)
  begin
    cntr <= 9;
    cntr2 <= cntr2 - 1;
    if (cntr2 == 0)
      cntr2 <= 5;
  end
  end
 end

assign q = cntr;
assign l = cntr2;

endmodule
```

---

caunter_pins.ucf:

```
NET "clk" LOC="p56";
NET "btn0" LOC="p38";
NET "sw0" LOC="p101";
NET "q[3]" LOC="p53";
NET "q[2]" LOC="p54";
NET "q[1]" LOC="p58";
NET "q[0]" LOC="p59";

NET "l[2]" LOC="p43";
NET "l[1]" LOC="p50";
NET "l[0]" LOC="p51";
```

---

wbbevtop1.v:

```
module wpbevtop1(
  input clk,btn0, sw0,
  output [3:0] q,
  output [2:0] l
  );
reg rst, dir;
always @(posedge clk)
//Synchronize inputs
begin
  rst <= btn0;
  dir <= sw0;
end

wire ce;
rategen rategenerator(
  .clk(clk),
  .rst(rst),
  .cy(ce)
);

countsec counter(
  .clk(clk),
  .rst(rst),
  .ce(ce),
  .dir(dir),
  .q(q),
  .l(l)
);
endmodule
```

---

test a top module hoz:

```
module wpbevtop1(
  input clk,btn0, sw0,
  output [3:0] q,
  output [2:0] l
  );
reg rst, dir;
always @(posedge clk)
//Synchronize inputs
begin
  rst <= btn0;
```

```
  dir <= sw0;
end

wire ce;
rategen rategenerator(
  .clk(clk),
  .rst(rst),
  .cy(ce)
);

countsec counter(
  .clk(clk),
  .rst(rst),
  .ce(ce),
  .dir(dir),
  .q(q),
  .l(l)
);
endmodule
```

```
//*************************************************
// BCD szamlalo 99ig - 2bites - fel le
//*************************************************

module count_nn(
input clk,
input rst,
input dir,
input ce,
output [7:0] q
);

reg [3:0] cntr;
reg [3:0] cntr_h;

always @(posedge clk)
	if (rst)
	begin
		cntr <= 0;
		cntr_h <= 0;
	end
	else if (ce)
		if (dir)
			if (cntr == 9)
				if ( cntr_h == 9)
					begin
						cntr <= 0;
						cntr_h <= 0;
					end
				else
					begin
						cntr <= 0;
						cntr_h <= cntr_h + 1;
					end
			else
				cntr <= cntr + 1;
		else
		if (cntr == 0)
			if(cntr_h == 0)
				begin
					cntr <= 9;
					cntr_h <= 9;
				end
			else
				begin
					cntr <= 9;
					cntr_h <= cntr_h - 1;
				end
```

```
                else
                cntr <= cntr - 1;

assign q = {cntr_h,cntr};
endmodule
```

```
//************************************************
/*
A hálózat egy 8 bites Johnson számláló (shift regiszter, melynek soros kimenete invertálva van
visszavezetve a soros bemenetre). A számláló a BTN2 gombbal vihető alapállapotba, az alapállapot
8'b0000_0000. A hálózat órajele az FPGA panel 16 MHz-es órajele. A számláló léptetését egy
másodpercenként érkező, 1 órajel szélességű engedélyező jel végzi. Ezt a jelet is a hálózat állítja elő.
A
számláló kimeneteinek állapotát a 8 LED jelzi. A számlaz SW0 kapcsoló 1 állapotában számol, 0
állapotában „pause” módban van.
*/
//************************************************

module main(
input clk,
input rst,
input sw0,
output[7:0] out
   );

reg[7:0] shr;
wire ce;
reg[23:0] cntr;

always@(posedge clk)
begin
        if(rst|ce)
                cntr<=0;
        else if(sw0)
                cntr<=cntr+1;

end
assign ce = (cntr==15999999 & sw0);
//teszthez assign ce = (cntr==4 & sw0);

always@(posedge clk)
begin
        if(rst)
                shr<=0;
        else if(ce)
                shr<={~shr[0],shr[7:1]};
end

assign out=shr;
endmodule
```

```
//************************************************
/*
A hálózat egy 8 bites, bináris számlálót valósít meg, amely az SW1 kapcsoló 0 állásában a páros, 1
állásában a páratlan számokon lépdel végig. A generált számokat a LED-eken kell megjeleníteni, a
számolást
pedig az 16 MHz-es órajelből generált, másodperc frekvenciájú engedélyező jel ütemezi. Az SW0
kapcsoló 1
állása a rendszert alapállapotba viszi (az SW1 kapcsoló állásától függően 0 vagy 1).
*/
//************************************************

module szamlalo(input clk, input rst, input let, output  [7:0] out)

//let: párosat vagy páratlant akarunk számolni
reg [7:0] ki = 8'b00000000;
wire [1:0] el =0;
reg [23:0] cntr=24'b000000000000000000000000;
always @(posedge clk)
      begin
      if (rst|en)
            cntr<=0;
      else
            cntr<=cntr+1;
      end

      assign en = (cntr==15999999);

assign el ={ki[0],let}; //azt nézi hogy mi az utolsó bit páros-e/páratlan és mit szeretnén számolni



always @(posedge clk)
      begin
      if (rst)
            ki <= 8'b00000000;
      else if(en)
            begin
             case (el)
                  00: ki<=ki+2;   //páros bit és párosat szeretnénk számolni
                  01: ki<=ki+1;   //páros bit és páratlant szeretnénkszámolni
                  10: ki<=ki+1;   //páratlan bit és párosat szeretnénkszámolni
                  11: ki<=ki+2;   //páratlan bit és páratlant szeretnénkszámolni
            endcase;
            end
      end

assign out =ki;

endmodule
```

```
//************************************************
/*
A hálózat egy futófényt (knight rider) valósít meg. A BTN0 gomb hatására (reset jel) csak a jobb
oldali LD0 LED világít, majd a gomb elengedésekor a fény másodperces ütemezéssel balra shiftelődik.
A
ledsor két végére érve a futási irány megfordul. A hálózat órajele az FPGA panel 16 MHz-es órajele. A
shiftelést egy másodpercenként érkező, 1 órajel szélességű engedélyező jel ütemezi. Ezt a jelet is a
hálózat
állítja elő. A generálandó minta tehát:
0000_0001 -> 0000_0010 -> 0000_0100 -> 0000_1000 -> 0001_0000 -> 0010_0000 -> 0100_0000 ->
1000_0000 -> 0100_0000 -> 0010_0000 -> 0001_0000 -> 0000_1000 -> 0000_0100 -> 0000_0010 ->
0000_0001 -> 0000_0010 -> ……..
Ügyeljen rá, hogy minden állapot ugyannyi ideig tartson (egy másodperc)

*/
//************************************************
module knightrider(input clk, input rst, output [7:0] out);

reg[7:0] shreg;
wire ce;
reg[23:0] cntr;
reg dir;

always@(posedge clk)
begind
        if(rst|ce)
                cntr<=0;
        else
                cntr<=cntr+1;
end

//assign ce=(cntr==15999999);
assign ce=(cntr==1);

always@(posedge clk)
begin
        if (rst)
                shreg<=8'b00000001;
        else
                begin
                case(shreg)
                        8'b00000001 : dir<=0                                    //
balra shiftel
                        8'b10000000 : dir<=1                                    //
jobbra shiftel
                endcase
        if (ce)
```

```
                begin
                if(dir)
                //jobbra
                shreg<={shreg[0],shreg[7:1]};
        else
                //balra
                shreg<={shreg[6:0],shreg[7]};
              end
         end
end
assign [7:0] out=[7:0] shreg;

endmodule
```

```
//************************************************
/*
A hálózat adott ütemben villogó 8 darab LED-t valósít meg, amelynek az ütemezését a SW1 és
SW0 kapcsoló állása határozza meg. A LED-ek {SW1, SW0}==0 esetben 0,25 mp-enként, {SW1,
SW0}==1
esetben 0,5 mp-enként, {SW1, SW0}==2 esetben 1 mp-enként, {SW1, SW0}==3 esetben pedig 2 mp-
enként
váltanak állapotot (mind a 8 LED egyszerre). A hálózat órajele az FPGA panel 16 MHz-es órajele.
*/
//************************************************
module pelda_4(
      input clk,
      input rst,
      input [1:0] sw,
      output [7:0] out
   );

reg [25:0] x;

always @ (*)
case (sw)
2'b00: x <= 3999999;
2'b01: x <= 7999999;
2'b10: x <= 15999999;
2'b11: x <= 31999999;
endcase

wire en;
reg [25:0] q1;

always @ (posedge clk)
          if(rst|en)
                    q1 <= 0;

          else
                    q1 <= q1 + 1;
assign en = (q1 == x );

reg [7:0] q;
always @ (posedge clk)
if(rst)
      q<=0;
else if(en)
      q<=~q;

assign out=q;
endmodule
```

```
//************************************************
/*
A hálózat bekapcsolás után egy 0 karaktert jelenít meg a jobb oldali hétszegmenses kijelzőn, majd
ezt követően másodperces időzítéssel a 0 karakter jobbról-balra halad a hétszegmenses kijelzőkön. A
bal
oldali kijelzőt elérve a karakter ismét jobbról lép be. A hálózat órajele az FPGA panel 16 MHz-es
órajele.
*/
//************************************************
module futonulla(input rst, input clk, output[3:0] an, output[6:0] SEG);

reg[23:0] sec;
wire sec_en;
always@(posedge clk)
begin
	if(rst | sec_en) sec<=0;
	else
		sec<=sec+1;
end
//assign sec_en=(sec==15999999); //1 másodpercenként váltja az anódot;
assign sec_en = (sec==19); //simhez
reg[3:0] an_reg;

always@(posedge clk) //1 másodpercenként shifteli a kijelzőt jobbrol balra
begin
	if(rst) an_reg<=4'b1110;
	else if(sec_en)
		an_reg<={an_reg[2:0],an_reg[3]};
end
assign an = an_reg;

wire khz_en;
reg[13:0] khz;
always@(posedge clk) //1kHz-es villogás a kijelzőnek  ELVILEG NEM KELL NEKI VILLOGNIA JÓ HA
KONSTANS MEGY
begin
	if(rst | khz_en) khz<=0;
	else khz<=khz+1;
end
//assign khz_en = (khz==15999);
assign khz_en = (khz==1); //szimulációhoz

reg[6:0] nul;
always@(posedge clk) //villoganulla 1kHz gyorsasággal
begin
	if(rst | !khz_en) nul<=7'b1111111;
	else if(khz_en) nul<=7'b1000000;
end
```

```
assign SEG=nul;
endmodule
```

```
//*************************************************
/*
A hálózat egy, az SW0 kapcsolóval változtatható számlálási irányú 0-9 értéktartományú
másodpercszámláló értékét mutatja meg a jobb oldali hétszegmenses kijelzőn, az LD0 LED egy
paritásbitet
jelenít meg, az LD1 LED pedig a szám páros jellegét azonosítja. A hálózat órajele az FPGA panel 16
MHz-es
órajele. A számlálást egy másodpercenként érkező, 1 órajel szélességű engedélyező jel ütemezi. Ezt a
jelet is
a hálózat állítja elő.
*/
//*************************************************
module pelda6(
   input clk,
   input rst,
   input SW0,
   output LD0,
   output LD1,
   output [3:0] AN,
   output [7:0] SEG
   );

      reg [23:0] cntr;
      wire oe;

      always @(posedge clk)
      begin
           if(rst | oe)
                cntr <= 0;
           else
                cntr <= cntr + 1;
      end

      //assign oe = (cntr == 15999999);
      assign oe = (cntr == 1);

      reg [3:0] cntr2;
      wire cntr2_eq_0; //lefel számláláshoz, ha a cntr2 = 0
      wire cntr2_eq_9;      //felfele számlálshoz, ha a cntr2 = 9

      always @(posedge clk)
      begin
           if(rst)
                cntr2 <= 0;
           else if(oe)
           begin
                if(SW0)
                begin
```

```
                if(cntr2_eq_9)
                    cntr2 <= 0;
                else
                    cntr2 <= cntr2 + 1;
            end
            else
            begin
                if(cntr2_eq_0)
                    cntr2 <= 9;
                else
                    cntr2 <= cntr2 - 1;
            end
        end
    end

    assign cntr2_eq_0 = (cntr2 == 0);
    assign cntr2_eq_9 = (cntr2 == 9);

    assign LD0 = ^(cntr2);      //paritás
    assign LD1 = ~^(cntr2);     //páros jelleg (~paritás)

    assign AN = 4'b1110;        //mivel csak a jobb oldali digitet használjuk

    reg [7:0] SEG_DEC;

    //7szegmens dekóder
    always @(cntr2)
    begin
        case (cntr2)
            4'h0:   SEG_DEC <= 8'b00000011;
            4'h1:   SEG_DEC <= 8'b10011111;
            4'h2:   SEG_DEC <= 8'b00100101;
            4'h3:   SEG_DEC <= 8'b00001101;
            4'h4:   SEG_DEC <= 8'b10011001;
            4'h5:   SEG_DEC <= 8'b01001001;
            4'h6:   SEG_DEC <= 8'b01000001;
            4'h7:   SEG_DEC <= 8'b00011111;
            4'h8:   SEG_DEC <= 8'b00000001;
            4'h9:   SEG_DEC <= 8'b00001001;
            default: SEG_DEC <= 8'b11111111;
        endcase
    end

    assign SEG = SEG_DEC;

endmodule
```

```
//************************************************
/*
Készítsen 16 bites bináris számlálót, melynek értékeit jelenítse meg a 4 darab hétszegmenses
kijelzőn. A BTN0 (reset jel) gomb hatására a számláló a 0 értéket veszi fel, egyébként
másodpercenként
számol, az aktuális értékek megjelennek a hétszegmenses kijelzőn. A hálózat órajele az FPGA panel
16 MHz-es órajele. A számlálást egy fél másodpercenként érkező, 1 órajel szélességű engedélyező jel
ütemezi. Ezt a jelet is a hálózat állítja elő.
*/
//************************************************

module pelda8(
  input clk,
  input rst,
  output [3:0] AN,
  output [7:0] SEG
  );

    reg [23:0] cntr;
    wire oe;

    always @(posedge clk)
    begin
        if(rst | oe)
            cntr <= 0;
        else
            cntr <= cntr + 1;
    end

    //assign oe = (cntr == 15999999);
    assign oe = (cntr == 4);

    //1.digit
    reg [3:0] cntr1;
    wire eq1_9;

    always @(posedge clk)
    begin
        if(rst)
            cntr1 <= 0;
        else if(oe)
            if(eq1_9)
                cntr1 <= 0;
            else
                cntr1 <= cntr1 + 1;
    end

    assign eq1_9 = (cntr1 == 9);
```

```
//2.digit
reg [3:0] cntr2;
wire eq2_9;

always @(posedge clk)
begin
    if(rst)
        cntr2 <= 0;
    else if(oe & eq1_9)
        if(eq2_9)
            cntr2 <= 0;
        else
            cntr2 <= cntr2 + 1;
end

assign eq2_9 = (cntr2 == 9);

//3.digit
reg [3:0] cntr3;
wire eq3_9;

always @(posedge clk)
begin
    if(rst)
        cntr3 <= 0;
    else if(oe & eq1_9 & eq2_9)
        if(eq3_9)
            cntr3 <= 0;
        else
            cntr3 <= cntr3 + 1;
end

assign eq3_9 = (cntr3 == 9);

//4.digit
reg [3:0] cntr4;
wire eq4_9;

always @(posedge clk)
begin
    if(rst)
        cntr4 <= 0;
    else if(oe & eq1_9 & eq2_9 & eq3_9)
        if(eq4_9)
            cntr4 <= 0;
        else
            cntr4 <= cntr4 + 1;
```

```
end

assign eq4_9 = (cntr4 == 9);


//számláló a kHz nagyságrendű engedélyező jel az anód(AN) jelhez
reg [3:0] cntr5;
wire en;

always @(posedge clk)
begin
    if(rst | en)
        cntr5 <= 0;
    else
        cntr5 <= cntr5 + 1;
end

//assign en = (cntr5 == 7999);
assign en = (cntr5 == 0);

//anod jel a megfelelő digit kiválasztásához
reg [3:0] anod;

always @(posedge clk)
begin
    if(rst)
        anod <= 4'b1110;
    else if(en)
        anod <= {anod[2:0],anod[3]};
end

assign AN = anod;

//2 bites számláló a multiplexerhez
reg [1:0] cntr6;

always @(posedge clk)
begin
    if(rst)
        cntr6 <= 0;
    else if(en)
        cntr6 <= cntr6 + 1;
end

//multiplexer ami a megfelelő regiszter értéket töli be a kiválasztott digitbe
reg [3:0] dmux;

always @(*)
```

```
    begin
        case(cntr6)
        2'b00: dmux <= cntr1;
        2'b01: dmux <= cntr2;
        2'b10: dmux <= cntr3;
        2'b11: dmux <= cntr4;
        endcase
    end

    //7szegmens dekóder
    reg [7:0] SEG_DEC;

    always @(dmux)
        case (dmux)
            4'h0:   SEG_DEC <= 8'b00000011;
            4'h1:   SEG_DEC <= 8'b10011111;
            4'h2:   SEG_DEC <= 8'b00100101;
            4'h3:   SEG_DEC <= 8'b00001101;
            4'h4:   SEG_DEC <= 8'b10011001;
            4'h5:   SEG_DEC <= 8'b01001001;
            4'h6:   SEG_DEC <= 8'b01000001;
            4'h7:   SEG_DEC <= 8'b00011111;
            4'h8:   SEG_DEC <= 8'b00000001;
            4'h9:   SEG_DEC <= 8'b00001001;
            default: SEG_DEC <= 8'b11111111;
        endcase

    assign SEG = SEG_DEC;

endmodule
```