LOGSYS

DIGITÁLIS VIDEO INTERFÉSZ MEGVALÓSÍTÁSA A LOGSYS KINTEX-7 FPGA KÁRTYÁVAL

2017. június 16.

Verzió 1.0

http://logsys.mit.bme.hu

Tartalomjegyzék

1 Bevezetés

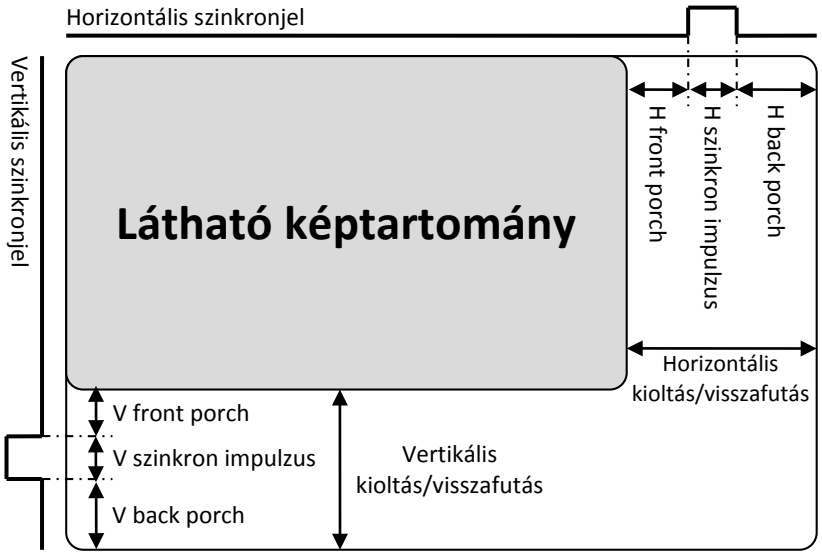
A digitális video interfészek (DVI, HDMI, DisplayPort) számos előnnyel rendelkeznek az analóg megvalósításhoz képest. Legfőbb előnyük, hogy ezeken keresztül lehetőség van a teljesen torzításmentes átvitelre. Az analóg VGA interfész esetén a digitális jelfeldolgozáshoz szükség van A/D és D/A átalakításra, illetve nagy felbontás esetén a kábel hosszától és minőségétől függően láthatóan csökkenhet a kép minősége.

A Xilinx 7-es sorozatú FPGA eszközei már tartalmazzák mindazon hardveres erőforrásokat, melyekkel könnyen megvalósítható a DVI és a HDMI interfész külső kiegészítő áramkörök használata nélkül. A dokumentum célja egy DVI/HDMI adó egység elkészítésének bemutatása a LOGSYS Kintex-7 FPGA kártyára. A leírtak megértéséhez digitális technika és Verilog HDL ismeretek szükségesek. Az egyes fejezetek tartalmazzák az ott leírtakhoz tartozó Verilog forráskódot.

2 A VGA kép felépítése

A DVI és a HDMI interfész ugyanazt a képi információt és vezérlő jeleket továbbítja a monitor felé, mint az analóg VGA interfész, csak digitálisan kódolva. Ezért először megnézzük a VGA kép felépítését és annak előállítását.

A VGA kép két fő részre osztható: a látható tartományra és a kioltási/visszafutási (nem látható) tartományra, melyet a 2-1. ábra szemléltet. A látható képtartományban minden pixel órajel ütemre ki kell adni a piros, a zöld és a kék színjelekre az aktuális pixel színét meghatározó értéket. A látható képtartományon kívül a színjelek értéke 0 kell, hogy legyen (kioltás). A horizontális, illetve a vertikális kioltási tartományok további három részre oszthatók fel: a szinkron impulzus előtti ***front porch*** szakaszra, a ***szinkronjel aktív szakaszára*** (szinkron impulzus), valamint a következő sor/kép kezdetéig tartó ***back porch*** szakaszra. A horizontális és a vertikális szinkronjel az adott felbontástól függően lehet aktív alacsony, illetve aktív magas. A vertikális visszafutás alatt is ki kell adni a horizontális szinkron impulzusokat.



2-1. ábra: A VGA kép felépítése.

Az adó egység kipróbálásához a monitoron egy 1440 x 900 @ 60 Hz szabványos felbontású képet fogunk megjeleníteni. Ehhez a felbontáshoz 106,47 MHz pixel órajel frekvencia tartozik, az egyéb időzítési paramétereket a 2-1. táblázat tartalmazza. A horizontális szinkron pulzus aktív alacsony, a vertikális szinkron pulzus pedig aktív magas szintű.

2-1. táblázat: Az 1440 x 900 @ 60 Hz VGA felbontás időzítési paraméterei.

	Látható rész	Front porch	Szinkron pulzus	Back porch	Összesen
Horizontális időzítés (pixel)	1440	80	152 (-)	232	1904
Vertikális időzítés (sor)	900	1	3 (+)	28	932

Az aktuális pixel pozíciót, valamint az időzítést meghatározó szinkron és kioltó jeleket a ***vga_timing*** modul állítja elő a leírtaknak megfelelően.

```
//*************************************************************************
//* 1440 x 900 @ 60 Hz VGA időzítés generátor.                            *
//*************************************************************************
module vga_timing(
   //Órajel és reset.
   input  wire        clk,            //Pixel órajel bemenet.
   input  wire        rst,            //Reset bemenet.

   //Az aktuális pixel pozíció.
   output reg  [10:0] h_cnt = 11'd0,  //X-koordináta.
   output reg  [9:0]  v_cnt = 10'd0,  //Y-koordináta.

   //Szinkron és kioltó jelek.
   output reg         h_sync = 1'b1,  //Horizontális szinkron pulzus.
   output reg         v_sync = 1'b0,  //Vertikális szinkron pulzus.
   output wire        blank           //Kioltó jel.
);
```

A 2-1. táblázatban szereplő adatokat, valamint a szinkron és a kioltó jelek aktivitásának kezdetét és végét lokális paraméterekként definiáljuk a modul elején. Az utóbbiak előállítását flip-flop-okkal oldjuk meg, így ezek beállítása és törlése egy órajel periódussal korábban kell, hogy történjen.

```
//*************************************************************************
//* Időzítési paraméterek.                                                *
//*************************************************************************
localparam H_VISIBLE     = 11'd1440;
localparam H_FRONT_PORCH = 11'd80;
localparam H_SYNC_PULSE  = 11'd152;
localparam H_BACK_PORCH  = 11'd232;

localparam V_VISIBLE     = 10'd900;
localparam V_FRONT_PORCH = 10'd1;
localparam V_SYNC_PULSE  = 10'd3;
localparam V_BACK_PORCH  = 10'd28;

localparam H_BLANK_BEGIN = H_VISIBLE     - 1;
localparam H_SYNC_BEGIN  = H_BLANK_BEGIN + H_FRONT_PORCH;
localparam H_SYNC_END    = H_SYNC_BEGIN  + H_SYNC_PULSE;
localparam H_BLANK_END   = H_SYNC_END    + H_BACK_PORCH;

localparam V_BLANK_BEGIN = V_VISIBLE     - 1;
localparam V_SYNC_BEGIN  = V_BLANK_BEGIN + V_FRONT_PORCH;
localparam V_SYNC_END    = V_SYNC_BEGIN  + V_SYNC_PULSE;
localparam V_BLANK_END   = V_SYNC_END    + V_BACK_PORCH;
```

A pixel pozíciót két, a sor hosszának, valamint a sorok számának megfelelő állapottal rendelkező számlálóval állíthatjuk elő. Az X-koordinátát a 11 bites horizontális (***h_cnt***), az Y-koordinátát pedig a 10 bites vertikális (***v_cnt***) számláló adja meg. Az utóbbi működését értelemszerűen csak a sorok végén kell engedélyezni.

```
//*************************************************************************
//* A horizontális és vertikális számlálók.                               *
//*************************************************************************
always @(posedge clk)
begin
   if (rst || (h_cnt == H_BLANK_END))
      h_cnt <= 12'd0;
   else
      h_cnt <= h_cnt + 12'd1;
end

always @(posedge clk)
begin
   if (rst)
      v_cnt <= 11'd0;
   else
      if (h_cnt == H_BLANK_END)
         if (v_cnt == V_BLANK_END)
            v_cnt <= 11'd0;
         else
            v_cnt <= v_cnt + 11'd1;
end
```

A horizontális és vertikális szinkron (***h_sync***, ***v_sync***), illetve kioltó (***h_blank***, ***v_blank***) jeleket egy-egy flip-flop kimenete adja, melyek a horizontális és vertikális számlálók megfelelő állapotában kerülnek beállításra és törlésre. A nem látható képtartományt jelző összevont ***blank*** kioltó jel a ***h_blank*** és a ***v_blank*** jelek VAGY kapcsolataként áll elő.

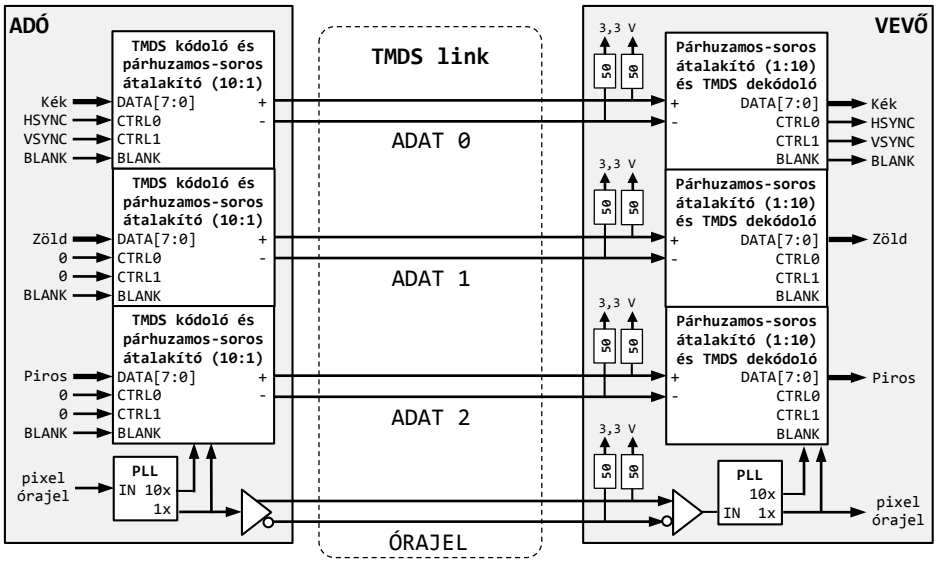
```
//*************************************************************************
//* A szinkron pulzusok generálása.                                       *
//*************************************************************************
always @(posedge clk)
begin
   if (rst || (h_cnt == H_SYNC_END))
      h_sync <= 1'b1;
   else
      if (h_cnt == H_SYNC_BEGIN)
         h_sync <= 1'b0;
end

always @(posedge clk)
begin
   if (rst)
      v_sync <= 1'b0;
   else
      if (h_cnt == H_BLANK_END)
         if (v_cnt == V_SYNC_BEGIN)
            v_sync <= 1'b1;
         else
            if (v_cnt == V_SYNC_END)
               v_sync <= 1'b0;
end


//*************************************************************************
//* A kioltó jel előállítása.                                             *
//*************************************************************************
reg h_blank = 1'b0;
reg v_blank = 1'b0;

always @(posedge clk)
begin
   if (rst || (h_cnt == H_BLANK_END))
      h_blank <= 1'b0;
   else
      if (h_cnt == H_BLANK_BEGIN)
         h_blank <= 1'b1;
end
```

```
always @(posedge clk)
begin
   if (rst)
      v_blank <= 1'b0;
   else
      if (h_cnt == H_BLANK_END)
         if (v_cnt == V_BLANK_BEGIN)
            v_blank <= 1'b1;
         else
            if (v_cnt == V_BLANK_END)
               v_blank <= 1'b0;
end

assign blank = h_blank | v_blank;

endmodule
```

3 A TMDS adó megvalósítása

3.1 Fizikai interfész

A DVI és a HDMI adó és vevő egység vázlatos felépítése, valamint ezek összeköttetése a 3-1. ábrán látható. Mindkét esetben az adatátvitel az úgynevezett TMDS (Transition-Minimized Differential Signaling) linken keresztül történik.



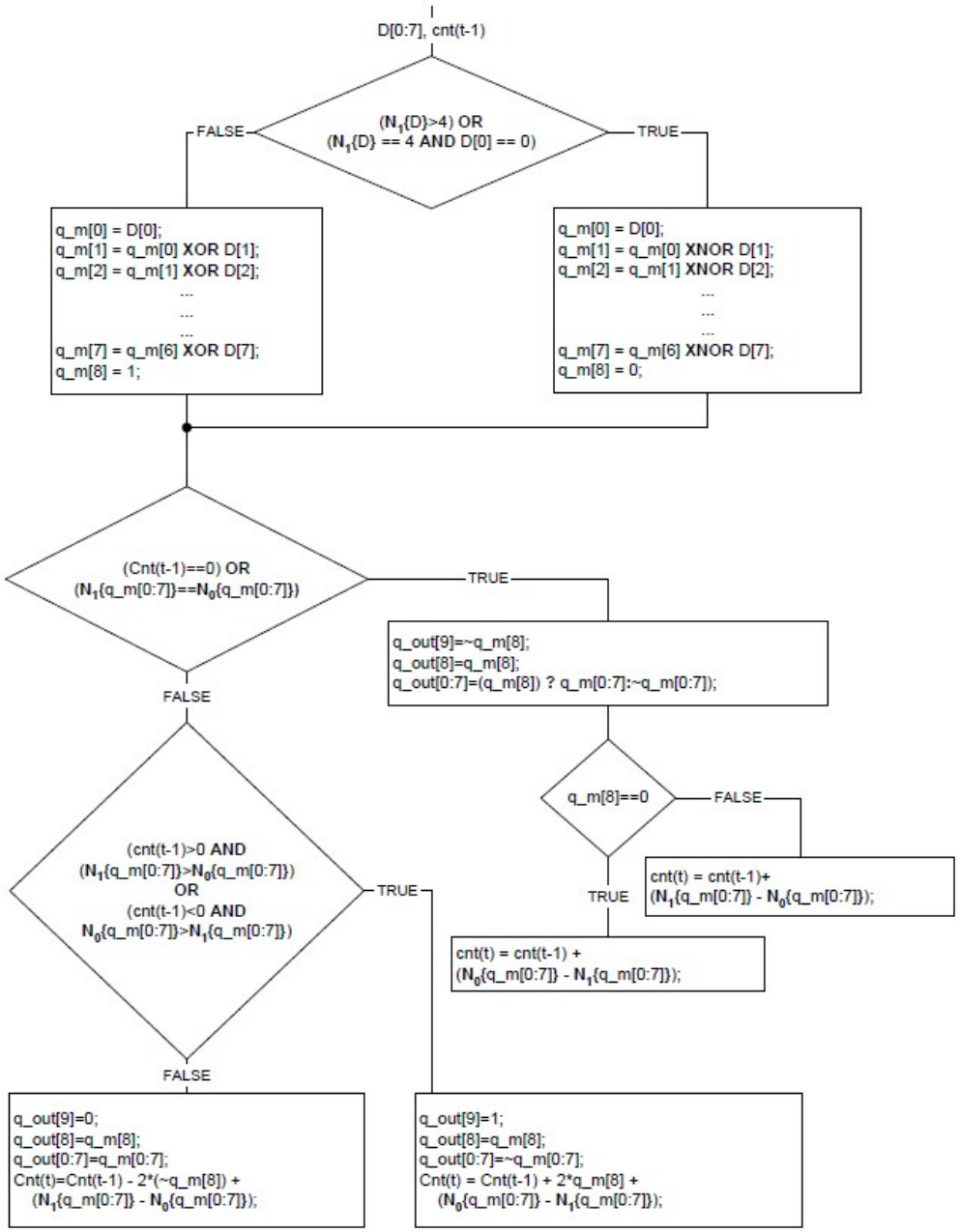
3-1. ábra: A DVI/HDMI adó és vevő egység blokkvázlata.

A TMDS link négy csatornából áll, melyek mindegyike egy-egy differenciális pár. A csatornák közül három az adatátvitelre szolgál, a negyedik pedig a pixel órajel átvitelét biztosítja. A TMDS link áram módú logikát használ: a differenciális párok vevőnél lévő felhúzó ellenállásait az adóban lévő áramgenerátorok különböző árammal terhelik, a vevő az ebből adódó feszültségkülönbséget érzékeli. Az adatcsatornákon 10 bites karakterek átvitele történik sorosan, tehát ezek működési frekvenciája a pixel órajel 10-szerese. Az átmenő pixel órajel és az adatok közötti fáziskülönbség nincs előírva, a vevőben történik a megfelelő mintavételező órajel előállítása, valamint az egyes adatcsatornák esetleges eltérő késleltetésének a kiegyenlítése. Kép információ esetén a kék színkomponens, valamint a horizontális és a vertikális szinkronjel a 0. adatcsatornán, a zöld színkomponens az 1. adatcsatornán, a piros színkomponens pedig a 2. adatcsatornán kerül átvitelre. A HDMI interfész a kép adatok mellett képes még hang adatok továbbítására is, ezzel azonban nem foglalkozunk, csak a DVI interfész által biztosított lehetőségek kerülnek megvalósításra.

3.2 TMDS kódoló

A TMDS kódoló a látható képtartomány esetén a 8 bites pixel adatokat a 3-2. ábrán látható algoritmus szerint alakítja át 10 bites karakterekké oly módon, hogy a kimenő adatban az átmenetek száma minimális legyen a DC egyensúly megtartása mellett. Az ábrán látható jelölések értelmezése a következő:

- ***D***: a kódoló bemenetére kerülő 8 bites pixel adat
- ***cnt***: a kimenő adatfolyamban az 1 és a 0 értékű bitek számának különbségét tároló regiszter, amely a DC egyensúly biztosításához szükséges
- ***q_m***: a kódolás első lépése után keletkező 9 bites érték
- ***q_out***: a kódoló által előállított 10 bites kimeneti adat
- ***N_1{X}***: az X argumentum 1 értékű bitjeinek száma
- ***N_0{X}***: az X argumentum 0 értékű bitjeinek száma



3-2. ábra: TMDS kódolási algoritmus a pixel adatok esetén.

A nem látható képtartomány esetén a ***CTRL0*** és ***CTRL1*** bemenetekre kerülő szinkronjelek a 3-1. táblázat szerint kerülnek kódolásra. Ezek a 10 bites értékek olyanok, hogy sohasem állnak elő a színkomponensek kódolása során.

3-1. táblázat: A vezérlőjelek kódolása a nem látható képtartományban.

CTRL1	CTRL0	10 bites kód
0	0	1101010100
0	1	0010101011
1	0	0101010100
1	1	1010101011

A TMDS kódolót a ***tmds_encoder*** modul valósítja meg a fent leírtak szerint. A kódolás első lépése egyben is leírható, hiszen a XOR és XNOR műveletek egymás negáltjai, a feltételes negálás pedig az első döntési feltétel XOR-olásával megoldható. A forráskódban csak az 1 értékű bitek számát határozzuk meg, amely minden esetben 8 bites adatra történik, így a 0 értékű bitek száma esetén az $N_0\{X\} = 8 - N_1\{X\}$ helyettesítés használható.

```
//*****************************************************************************
//* TMDS kódoló.                                                              *
//*****************************************************************************
module tmds_encoder(
   //Órajel és reset.
   input  wire       clk,        //Pixel órajel bemenet.
   input  wire       rst,        //Aszinkron reset bemenet.

   //Bemenő adat.
   input  wire [7:0] data_in,    //A kódolandó pixel adat.
   input  wire       data_en,    //A látható képtartomány jelzése.
   input  wire       ctrl0_in,   //Vezérlőjelek.
   input  wire       ctrl1_in,

   //Kimenő adat.
   output reg  [9:0] tmds_out
);

//*****************************************************************************
//* Az "1" értékű bitek számának meghatározása a bejövő pixel adatokban.      *
//* A pipeline fokozatok száma: 1                                             *
//*****************************************************************************
reg [7:0] data_in_reg;
reg [3:0] din_num_1s;

always @(posedge clk)
begin
   data_in_reg <= data_in;
   din_num_1s  <= ((data_in[0] + data_in[1]) + (data_in[2] + data_in[3])) +
                  ((data_in[4] + data_in[5]) + (data_in[6] + data_in[7]));
end


//*****************************************************************************
//* A TMDS kódolás első lépése: 8 bitről 9 bitre történő átalakítás.          *
//* A pipeline fokozatok száma: 1                                             *
//*****************************************************************************
wire [8:0] stage1;
reg  [8:0] stage1_out;

//Az első döntési feltétel:
//- az "1" bitek száma nagyobb 4-nél vagy
//- az "1" bitek száma 4 és a bejövő adat LSb-je 0.
wire decision1 = (din_num_1s > 4'd4) | ((din_num_1s == 4'd4) & ~data_in_reg[0]);

assign stage1[0] = data_in_reg[0];
assign stage1[1] = (stage1[0] ^ data_in_reg[1]) ^ decision1;
assign stage1[2] = (stage1[1] ^ data_in_reg[2]) ^ decision1;
assign stage1[3] = (stage1[2] ^ data_in_reg[3]) ^ decision1;
assign stage1[4] = (stage1[3] ^ data_in_reg[4]) ^ decision1;
assign stage1[5] = (stage1[4] ^ data_in_reg[5]) ^ decision1;
```

```
assign stage1[6] = (stage1[5] ^ data_in_reg[6]) ^ decision1;
assign stage1[7] = (stage1[6] ^ data_in_reg[7]) ^ decision1;
assign stage1[8] = ~decision1;

always @(posedge clk)
begin
   stage1_out <= stage1;
end


//************************************************************************
//* Az "1" értékű bitek számának meghatározása az első lépés kimenetében.     *
//* A pipeline fokozatok száma: 1                                             *
//************************************************************************
reg  [8:0] stage2_in;
reg  [3:0] s1_num_1s;

always @(posedge clk)
begin
   stage2_in <= stage1_out;
   s1_num_1s <= ((stage1_out[0] + stage1_out[1]) + (stage1_out[2] + stage1_out[3])) +
                ((stage1_out[4] + stage1_out[5]) + (stage1_out[6] + stage1_out[7]));
end


//************************************************************************
//* Pipeline regiszterek az engedélyező és a vezérlő jelek számára.           *
//************************************************************************
reg [2:0] data_en_reg;
reg [5:0] ctrl_reg;

always @(posedge clk)
begin
   if (rst)
      data_en_reg <= 3'd0;
   else
      data_en_reg <= {data_en_reg[1:0], data_en};
end

always @(posedge clk)
begin
   if (rst)
      ctrl_reg <= 6'd0;
   else
      ctrl_reg <= {ctrl_reg[3:0], ctrl1_in, ctrl0_in};
end


//************************************************************************
//* A TMDS kódolás második lépése: 9 bitről 10 bitre történő átalakítás.      *
//************************************************************************
localparam CTRL_TOKEN_0 = 10'b1101010100;
localparam CTRL_TOKEN_1 = 10'b0010101011;
localparam CTRL_TOKEN_2 = 10'b0101010100;
localparam CTRL_TOKEN_3 = 10'b1010101011;

//A kimeneti "0" és "1" bitek számának különbsége (MSb az előjel bit).
reg [4:0] cnt;

//A második döntési feltétel:
//- az eddig kiadott "0" és "1" bitek száma azonos vagy
//- az első lépés kimenetének alsó 8 bitjén a "0" és az "1" bitek száma azonos.
wire decision2 = (cnt == 5'd0) | (s1_num_1s == 4'd4);

//A harmadik döntési feltétel:
//- eddig több "1" bit került elküldésre, mint "0" és az első lépés kimenetében
//  az "1" értékű bitek száma a nagyobb vagy
//- eddig több "0" bit került elküldésre, mint "1" és az első lépés kimenetében
//  a "0" értékű bitek száma a nagyobb
wire decision3 = (~cnt[4] & (s1_num_1s > 4'd4)) | (cnt[4] & (s1_num_1s < 4'd4));

always @(posedge clk or posedge rst)
begin
   if (rst || (data_en_reg[2] == 0))
      cnt <= 5'd0;
   else
```

```
      if (decision2)
         if (stage2_in[8])
            //cnt = cnt + (#1s - #0s)
            cnt <= cnt + ({s1_num_1s, 1'b0} - 5'd8);
         else
            //cnt = cnt + (#0s - #1s)
            cnt <= cnt + (5'd8 - {s1_num_1s, 1'b0});
      else
         if (decision3)
            //cnt = cnt + 2*stage2_in[8] + (#0s - #1s)
            cnt <= (cnt + {stage2_in[8], 1'b0})  + (5'd8 - {s1_num_1s, 1'b0});
         else
            //cnt = cnt - 2*(~stage2_in[8]) + (#1s - #0s)
            cnt <= (cnt - {~stage2_in[8], 1'b0}) + ({s1_num_1s, 1'b0} - 5'd8);
end

always @(posedge clk or posedge rst)
begin
   if (rst)
      tmds_out <= 10'd0;
   else
      if (data_en_reg[2])
         if (decision2)
            tmds_out <= {~stage2_in[8], stage2_in[8], stage2_in[7:0] ^ {8{~stage2_in[8]}}};
         else
            if (decision3)
               tmds_out <= {1'b1, stage2_in[8], ~stage2_in[7:0]};
            else
               tmds_out <= {1'b0, stage2_in[8],  stage2_in[7:0]};
      else
         case (ctrl_reg[5:4])
            2'b00: tmds_out <= CTRL_TOKEN_0;
            2'b01: tmds_out <= CTRL_TOKEN_1;
            2'b10: tmds_out <= CTRL_TOKEN_2;
            2'b11: tmds_out <= CTRL_TOKEN_3;
         endcase
end

endmodule
```

3.3 Párhuzamos-soros átalakító

A TMDS kódoló 10 bites kimenetének sorosítása az FPGA áramkörben található OSERDESE2 egységek felhasználásával történik. Egy ilyen egység legfeljebb 8 bites adatot tud sorosítani, így a 10 bites TMDS kód sorosításához két OSERDESE2 kaszkádosítása szükséges. Ennél az adatszélességnél csak a DDR üzemmód használható, tehát a soros kiléptető órajel (***clk_5x***) frekvenciája a pixel órajel (***clk***) frekvenciájának ötszöröse kell, hogy legyen. Az OSEDRESE2 működéséről és használatáról részletesen a 7 Series FPGAs SelectIO Resources User Guide (UG471)[1] dokumentumban olvashatunk. A párhuzamos-soros átalakítót az ***oserdes_10to1*** modul írja le. A két OSERDESE2 példányosítása után példányosítunk egy differenciális kimeneti buffert (OBUFDS) az adott TMDS csatornához.

```
//*********************************************************************************
//* 10:1 párhuzamos-soros átalakító differenciális kimenettel.                    *
//*********************************************************************************
module oserdes_10to1(
   //Órajel és reset.
   input  wire       clk,              //1x órajel bemenet.
   input  wire       clk_5x,           //5x órajel bemenet (DDR mód).
   input  wire       rst,              //Aszinkron reset jel.

   //10 bites adat bemenet.
   input  wire [9:0] data_in,

   //Differenciális soros adat kimenet.
   output wire       dout_p,
   output wire       dout_n
);
```

[1] https://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf

```
//*****************************************************************************
//* Master OSERDES.                                                           *
//*****************************************************************************
wire data_to_iob, master_shiftin1, master_shiftin2;

OSERDESE2 #(
   .DATA_RATE_OQ("DDR"),
   .DATA_RATE_TQ("DDR"),
   .DATA_WIDTH(10),
   .INIT_OQ(1'b0),
   .INIT_TQ(1'b0),
   .SERDES_MODE("MASTER"),
   .SRVAL_OQ(1'b0),
   .SRVAL_TQ(1'b0),
   .TBYTE_CTL("FALSE"),
   .TBYTE_SRC("FALSE"),
   .TRISTATE_WIDTH(1)
) master_oserdes (
   .OFB(),
   .OQ(data_to_iob),
   .SHIFTOUT1(),
   .SHIFTOUT2(),
   .TBYTEOUT(),
   .TFB(),
   .TQ(),
   .CLK(clk_5x),
   .CLKDIV(clk),
   .D1(data_in[0]),
   .D2(data_in[1]),
   .D3(data_in[2]),
   .D4(data_in[3]),
   .D5(data_in[4]),
   .D6(data_in[5]),
   .D7(data_in[6]),
   .D8(data_in[7]),
   .OCE(1'b1),
   .RST(rst),
   .SHIFTIN1(master_shiftin1),
   .SHIFTIN2(master_shiftin2),
   .T1(1'b0),
   .T2(1'b0),
   .T3(1'b0),
   .T4(1'b0),
   .TBYTEIN(1'b0),
   .TCE(1'b0)
);


//*****************************************************************************
//* Slave OSERDES.                                                            *
//*****************************************************************************
OSERDESE2 #(
   .DATA_RATE_OQ("DDR"),
   .DATA_RATE_TQ("DDR"),
   .DATA_WIDTH(10),
   .INIT_OQ(1'b0),
   .INIT_TQ(1'b0),
   .SERDES_MODE("SLAVE"),
   .SRVAL_OQ(1'b0),
   .SRVAL_TQ(1'b0),
   .TBYTE_CTL("FALSE"),
   .TBYTE_SRC("FALSE"),
   .TRISTATE_WIDTH(1)
) slave_oserdes (
   .OFB(),
   .OQ(),
   .SHIFTOUT1(master_shiftin1),
   .SHIFTOUT2(master_shiftin2),
   .TBYTEOUT(),
   .TFB(),
   .TQ(),
   .CLK(clk_5x),
   .CLKDIV(clk),
   .D1(1'b0),
   .D2(1'b0),
   .D3(data_in[8]),
```

```
   .D4(data_in[9]),
   .D5(1'b0),
   .D6(1'b0),
   .D7(1'b0),
   .D8(1'b0),
   .OCE(1'b1),
   .RST(rst),
   .SHIFTIN1(1'b0),
   .SHIFTIN2(1'b0),
   .T1(1'b0),
   .T2(1'b0),
   .T3(1'b0),
   .T4(1'b0),
   .TBYTEIN(1'b0),
   .TCE(1'b0)
);


//*********************************************************************************
//* Differenciális kimeneti buffer.                                               *
//*********************************************************************************
OBUFDS #(
   .IOSTANDARD("TMDS_33"),
   .SLEW("FAST")
) output_buffer (
   .I(data_to_iob),
   .O(dout_p),
   .OB(dout_n)
);

endmodule
```

3.4 TMDS adó modul

A TMDS adóhoz tartozó alegységeket a ***tmds_transmitter*** modul tartalmazza. A 3-1. ábrának megfelelően az adatcsatornákhoz három-három példány szükséges a TMDS kódolóból és a párhuzamos-soros átalakítóból. A pixel órajelet az I/O lábakra közvetlenül nem, hanem csak kimeneti DDR regiszter alkalmazásával lehet kivezetni, így az órajel csatorna megvalósításához egy ODDR és egy OBUFDS példány szükséges.

```
//*********************************************************************************
//* TMDS adó.                                                                     *
//*********************************************************************************
module tmds_transmitter(
   //Órajel és reset.
   input  wire       clk,                //Pixel órajel bemenet.
   input  wire       clk_5x,             //5x pixel órajel bemenet.
   input  wire       rst,                //Reset jel.

   //Bemeneti video adatok.
   input  wire [7:0] red_in,             //Piros színkomponens.
   input  wire [7:0] green_in,           //Zöld színkomponens.
   input  wire [7:0] blue_in,            //Kék színkomponens.
   input  wire       blank_in,           //A nem látható képtartomány jelzése.
   input  wire       hsync_in,           //Horizontális szinkronjel.
   input  wire       vsync_in,           //Vertikális szinkronjel.

   //Kimenő TMDS jelek.
   output wire       tmds_data0_out_p,   //Adat 0.
   output wire       tmds_data0_out_n,
   output wire       tmds_data1_out_p,   //Adat 1.
   output wire       tmds_data1_out_n,
   output wire       tmds_data2_out_p,   //Adat 2.
   output wire       tmds_data2_out_n,
   output wire       tmds_clock_out_p,   //Pixel órajel.
   output wire       tmds_clock_out_n
);

//*********************************************************************************
//* A TMDS kódolók példányosítása.                                                *
//*********************************************************************************
```

```
wire [9:0] tmds_red, tmds_green, tmds_blue;

tmds_encoder encoder_r(
   //Órajel és reset.
   .clk(clk),                        //Pixel órajel bemenet.
   .rst(rst),                        //Aszinkron reset bemenet.

   //Bemenő adat.
   .data_in(red_in),                 //A kódolandó pixel adat.
   .data_en(~blank_in),              //A látható képtartomány jelzése.
   .ctrl0_in(1'b0),                  //Vezérlőjelek.
   .ctrl1_in(1'b0),

   //Kimenő adat.
   .tmds_out(tmds_red)
);

tmds_encoder encoder_g(
   //Órajel és reset.
   .clk(clk),                        //Pixel órajel bemenet.
   .rst(rst),                        //Aszinkron reset bemenet.

   //Bemenő adat.
   .data_in(green_in),               //A kódolandó pixel adat.
   .data_en(~blank_in),              //A látható képtartomány jelzése.
   .ctrl0_in(1'b0),                  //Vezérlőjelek.
   .ctrl1_in(1'b0),

   //Kimenő adat
   .tmds_out(tmds_green)
);

tmds_encoder encoder_b(
   //Órajel és reset.
   .clk(clk),                        //Pixel órajel bemenet.
   .rst(rst),                        //Aszinkron reset bemenet.

   //Bemenő adat.
   .data_in(blue_in),                //A kódolandó pixel adat.
   .data_en(~blank_in),              //A látható képtartomány jelzése.
   .ctrl0_in(hsync_in),              //Vezérlőjelek.
   .ctrl1_in(vsync_in),

   //Kimenő adat
   .tmds_out(tmds_blue)
);

//*********************************************************************************
//* A párhuzamos-soros átalakítok példányosítása.                                 *
//*********************************************************************************
oserdes_10to1 oserdes0(
   //Órajel és reset.
   .clk(clk),                        //1x órajel bemenet.
   .clk_5x(clk_5x),                  //5x órajel bemenet (DDR mód).
   .rst(rst),                        //Aszinkron reset jel.

   //10 bites adat bemenet.
   .data_in(tmds_blue),

   //Differenciális soros adat kimenet.
   .dout_p(tmds_data0_out_p),
   .dout_n(tmds_data0_out_n)
);

oserdes_10to1 oserdes1(
   //Órajel és reset.
   .clk(clk),                        //1x órajel bemenet.
   .clk_5x(clk_5x),                  //5x órajel bemenet (DDR mód).
   .rst(rst),                        //Aszinkron reset jel.

   //10 bites adat bemenet.
   .data_in(tmds_green),

   //Sifferenciális soros adat kimenet.
   .dout_p(tmds_data1_out_p),
   .dout_n(tmds_data1_out_n)
```

```
);

oserdes_10to1 oserdes2(
   //Órajel és reset.
   .clk(clk),                       //1x órajel bemenet.
   .clk_5x(clk_5x),                 //5x órajel bemenet (DDR mód).
   .rst(rst),                       //Asynchronous reset signal.

   //10 bites adat bemenet.
   .data_in(tmds_red),

   //Differenciális soros adat kimenet.
   .dout_p(tmds_data2_out_p),
   .dout_n(tmds_data2_out_n)
);


//*********************************************************************************
//* TMDS pixel órajel csatorna.                                                   *
//*********************************************************************************
wire clk_out;

ODDR #(
   .DDR_CLK_EDGE("OPPOSITE_EDGE"),  // "OPPOSITE_EDGE" vagy "SAME_EDGE".
   .INIT(1'b0),                     // A Q kimenet kezdeti értéke.
   .SRTYPE("ASYNC")                 // "SYNC" vagy "ASYNC" beállítás/törlés.
) ODDR_clk (
   .Q(clk_out),                     // 1 bites DDR kimenet.
   .C(clk),                         // 1 bites órajel bemenet.
   .CE(1'b1),                       // 1 bites órajel engedélyező bemenet.
   .D1(1'b1),                       // 1 bites adat bemenet (felfutó él).
   .D2(1'b0),                       // 1 bites adat bemenet (lefutó él).
   .R(rst),                         // 1 bites törlő bemenet.
   .S(1'b0)                         // 1 bites 1-be állító bemenet.
);

OBUFDS #(
   .IOSTANDARD("TMDS_33"),
   .SLEW("FAST")
) OBUFDS_clk (
   .I(clk_out),
   .O(tmds_clock_out_p),
   .OB(tmds_clock_out_n)
);

endmodule
```

4 A TMDS adó kipróbálása

A TMDS adó kipróbálásához szükséges kódrészt a ***hdmi_tx*** modul tartalmazza. Ennek bemenetei az FPGA kártyán lévő 100 MHz-es oszcillátor által generált órajel, valamint az RST nyomógomb. A HDMI kimeneti csatlakozóhoz tartozó jelek közül csak a TMDS csatornákat használjuk. A nem használt jelek (CEC, hot-plug érzékelés, EDID EEPROM I^2C interfész) mindegyike bemenetként van definiálva, hogy ne lehessen meghajtani ezeket.

```
//*********************************************************************************
//* HDMI adó teszt alkalmazás.                                                    *
//*********************************************************************************
module hdmi_tx(
   //Órajel és reset.
   input  wire       clk100M,             //100 MHz órajel bemenet.
   input  wire       rstbt,               //Reset bemenet (RST gomb).

   //A HDMI kimeneti csatlakozó jelei.
   output wire       hdmi_tx_d0_p,        //TMDS adat 0.
   output wire       hdmi_tx_d0_n,
   output wire       hdmi_tx_d1_p,        //TMDS adat 1.
   output wire       hdmi_tx_d1_n,
   output wire       hdmi_tx_d2_p,        //TMDS adat 2.
   output wire       hdmi_tx_d2_n,
   output wire       hdmi_tx_clk_p,       //TMDS pixel órajel.
```

```
   output wire       hdmi_tx_clk_n,
   input  wire       hdmi_tx_cec,         //HDMI CEC (nem használt).
   input  wire       hdmi_tx_hpdn,        //HDMI hot-plug érzékelés (nem használt).
   input  wire       hdmi_tx_scl,         //EDID EEPROM I2C interfész (nem használt).
   input  wire       hdmi_tx_sda
);
```

A 100 MHz-es órajelből egy MMCM felhasználásával állítjuk elő az 1140 x 900 @ 60 Hz felbontáshoz szükséges 106,47 MHz-es pixel órajelet és az ötször nagyobb frekvenciájú, 532,35 MHz-es soros kiléptető órajelet. Az MMCM használatáról bővebben a 7 Series FPGAs Clocking Resources User Guide (UG472)[2] dokumentumban olvashatunk, a működési határadatokat a Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics (DS182)[3] tartalmazza. Ezek figyelembevételével a kívánt frekvenciák előállításához az alábbi konfigurációt kell alkalmazni:

- A globális szorzás értéke (***CLKFBOUT_MULT_F***): 63,875
- A globális osztás értéke (***DIVCLK_DIVIDE)***: 6
- Az osztás értéke a CLKOUT0 kimeneten (***CLKOUT0_DIVIDE_F***): 2,0 → $f_{CLKOUT0}$ = 532,29 MHz
- Az osztás értéke a CLKOUT1 kimeneten (***CLKOUT1_DIVIDE***): 10 → $f_{CLKOUT1}$ = 106,46 MHz

Az MMCM modul CLKOUT0 és CLKOUT1 kimenetét egy-egy BUFG-n keresztül kapcsoljuk az FPGA globális órajel hálózatára. A rendszert mindaddig reset állapotban tartjuk, amíg az MMCM órajel kimenetek nem stabilak.

```
//*********************************************************************************
//* MMCM példányosítása a szükséges órajelek előállításához:                      *
//* - clk   : 106,47 MHz pixel órajel                                             *
//* - clk_5x: 532,35 MHz órajel a soros adatok kiléptetéséhez                     *
//*********************************************************************************
wire mmcm_clkout0, mmcm_clkout1, mmcm_clkfb, mmcm_locked;

MMCME2_ADV #(
   .BANDWIDTH           ("OPTIMIZED"),
   .CLKOUT4_CASCADE     ("FALSE"),
   .COMPENSATION        ("ZHOLD"),
   .STARTUP_WAIT        ("FALSE"),
   .DIVCLK_DIVIDE       (6),              //A globális osztás értéke.
   .CLKFBOUT_MULT_F     (63.875),         //A globális szorzás értéke.
   .CLKFBOUT_PHASE      (0.000),
   .CLKFBOUT_USE_FINE_PS("FALSE"),
   .CLKOUT0_DIVIDE_F    (2.0),            //A CLKOUT0 kimenetre beállított osztás.
   .CLKOUT0_PHASE       (0.000),
   .CLKOUT0_DUTY_CYCLE  (0.500),
   .CLKOUT0_USE_FINE_PS ("FALSE"),
   .CLKOUT1_DIVIDE      (10),             //A CLKOUT1 kimenetre beállított osztás.
   .CLKOUT1_PHASE       (0.000),
   .CLKOUT1_DUTY_CYCLE  (0.500),
   .CLKOUT1_USE_FINE_PS ("FALSE"),
   .CLKIN1_PERIOD       (10.000)          //Bemeneti órajel periódusidő [ns].
) mmcm_adv_inst (
   //Kimenő órajelek.
   .CLKFBOUT            (mmcm_clkfb),
   .CLKOUT0             (mmcm_clkout0),
   .CLKOUT1             (mmcm_clkout1),
   //Bemenő órajelek.
   .CLKFBIN             (mmcm_clkfb),
   .CLKIN1              (clk100M),
   .CLKIN2              (1'b0),
   //Az órajel bemenet kiválasztó jele (a CLKIN1 bemenetet használjuk).
   .CLKINSEL            (1'b1),
   //A dinamikos átkonfiguráláshoz tartozó jelek.
   .DADDR               (7'h0),
   .DCLK                (1'b0),
   .DEN                 (1'b0),
   .DI                  (16'h0),
   .DO                  (),
```

[2] https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf
[3] https://www.xilinx.com/support/documentation/data_sheets/ds182_Kintex_7_Data_Sheet.pdf

```
   .DRDY               (),
   .DWE                (1'b0),
   //A dinamikos fázis léptetéshez tartozó jelek.
   .PSCLK              (1'b0),
   .PSEN               (1'b0),
   .PSINCDEC           (1'b0),
   .PSDONE             (),
   //Egyéb vezérlő és státusz jelek.
   .LOCKED             (mmcm_locked),
   .PWRDWN             (1'b0),
   .RST                (rstbt)
);

//Órajel bufferek.
wire clk, clk_5x;

BUFG BUFG_fastclk(.I(mmcm_clkout0), .O(clk_5x));
BUFG BUFG_slowclk(.I(mmcm_clkout1), .O(clk));

//Reset jel.
wire rst = ~mmcm_locked;
```

A következő kódrészlet tartalmazza a VGA időzítés generátor példányosítását és a megjelenítendő minta előállítását. A monitoron színes négyzetekből álló mintát jelenítünk meg, amely egyszerűen előállítható a megfelelő számláló bitek XOR kapcsolatával.

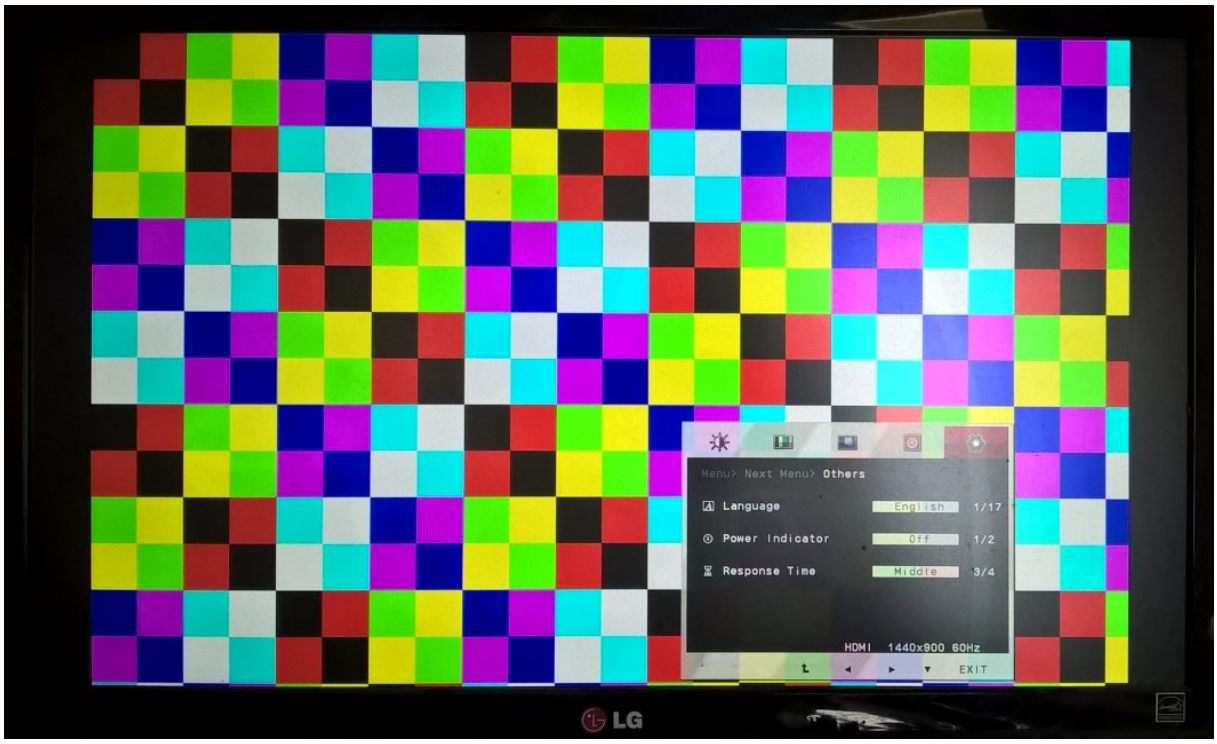
```
//*****************************************************************************
//* 1440 x 900 @ 60 Hz VGA időzítés generátor.                                *
//*****************************************************************************
wire [10:0] h_cnt;
wire [9:0]  v_cnt;
wire        h_sync, v_sync, blank;

vga_timing vga_timing(
   //Órajel és reset.
   .clk(clk),                       //Pixel órajel bemenet.
   .rst(rst),                       //Reset bemenet.

   //Az aktuális pixel pozíció.
   .h_cnt(h_cnt),                   //X-koordináta.
   .v_cnt(v_cnt),                   //Y-koordináta.

   //Szinkron és kioltó jelek.
   .h_sync(h_sync),                 //Horizontális szinkron pulzus.
   .v_sync(v_sync),                 //Vertikális szinkron pulzus.
   .blank(blank)                    //Kioltó jel.
);

//A megjelenítendő minta előállítása.
wire [7:0] red   = {8{h_cnt[6] ^ v_cnt[6]}};
wire [7:0] green = {8{h_cnt[7] ^ v_cnt[7]}};
wire [7:0] blue  = {8{h_cnt[8] ^ v_cnt[8]}};
```

A modul végén példányosítjuk meg a TMDS adót.

```
//*****************************************************************************
//* TMDS adó.                                                                 *
//*****************************************************************************
tmds_transmitter tmds_transmitter(
   //Órajel és reset.
   .clk(clk),                       //Pixel órajel bemenet.
   .clk_5x(clk_5x),                 //5x pixel órajel bemenet.
   .rst(rst),                       //Reset jel.

   //Bemeneti video adatok.
   .red_in(red),                    //Piros színkomponens.
   .green_in(green),                //Zöld színkomponens.
   .blue_in(blue),                  //Kék színkomponens.
   .blank_in(blank),                //A nem látható képtartomány jelzése.
   .hsync_in(h_sync),               //Horizontális szinkronjel.
   .vsync_in(v_sync),               //Vertikális szinkronjel.
   //Kimenő TMDS jelek.
```

```
    .tmds_data0_out_p(hdmi_tx_d0_p),    //Adat 0.
    .tmds_data0_out_n(hdmi_tx_d0_n),
    .tmds_data1_out_p(hdmi_tx_d1_p),    //Adat 1.
    .tmds_data1_out_n(hdmi_tx_d1_n),
    .tmds_data2_out_p(hdmi_tx_d2_p),    //Adat 2.
    .tmds_data2_out_n(hdmi_tx_d2_n),
    .tmds_clock_out_p(hdmi_tx_clk_p),   //Pixel órajel.
    .tmds_clock_out_n(hdmi_tx_clk_n)
);

endmodule
```

Hozzunk létre egy projektet a Xilinx Vivado fejlesztői környezetben és a forrásfájlok hozzáadása után gépeljük be vagy másoljuk be azok tartalmát. Majd futtassuk le a szintézist és adjuk meg a top-level modul portjaihoz tartozó I/O lábakat (ez megtalálható a LOGSYS Kintex-7 FPGA kártya felhasználói útmutatójában), valamint állítsuk be a ***clk100M*** portra a 100 MHz-es órajel frekvenciát. A megkötések megadása után generálhatjuk az FPGA konfigurációs fájlt. Az FPGA felkonfigurálása és a monitor csatlakoztatása után a 4-1. ábrán látható színes négyzetekből álló minta jelenik meg.



4-1. ábra: A teszt rendszer által generált kép.