Logikai tervezés házi feladat dokumentáció

Feladat: Éldetektálás

Készítette:

Benedek Ádám(CIB4N0)

K.Tóth Lilla Magdolna (QINJHB)

Konzulens:

Szántó Péter

2016. május 19.

Specifikáció

A megvalósítandó feladat egy képfeldolgozás. Az egység szürkeárnyalatos képeken éldetektálást végez. Az adatformátum 8 bites.

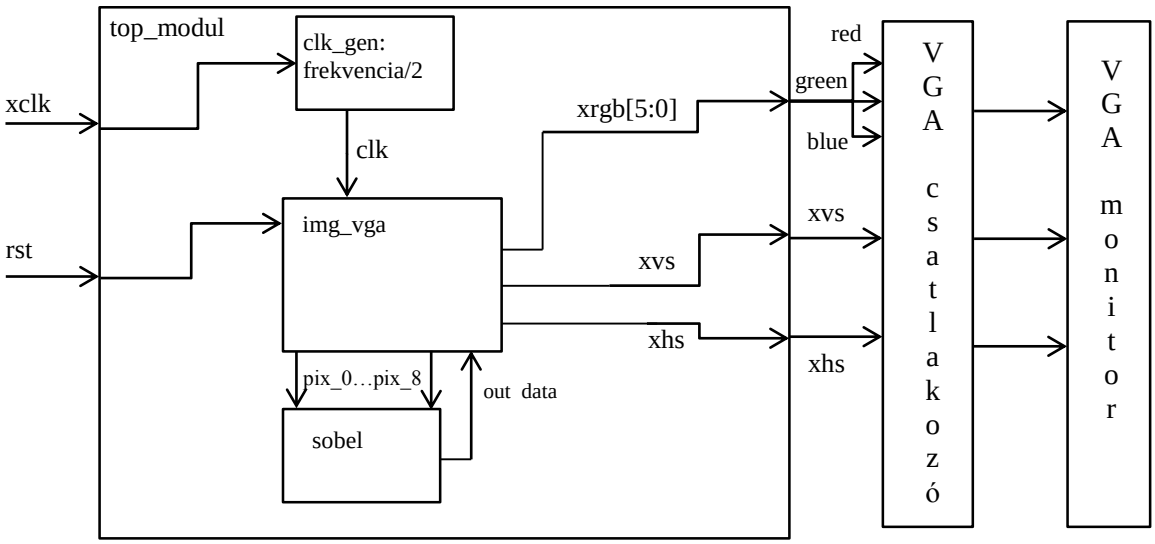
A házi feladat elkészítése során első lépésben az éldetektáló algoritmusokkal ismerkedtünk meg. Ehhez főként internetes forrásokat használtunk. Ezután átgondoltuk és kidolgoztuk a feladat részletes specifikációját.

Az elképzelés az volt, hogy real time valósítjuk meg, vagyis a bemeneti kép HDMI-n érkezne és egy HDMI➔ DVI átalakítóm keresztül megjelenne monitoron az éldetektált kép. A félév során erre a megvalósításra nem jutott idő, de a későbbiekben szeretnénk ezt is megoldani.

A feladatnak egy egyszerűbb verziója került kidolgozásra. A feldolgozandó képet statikusan egy Blokk RAM-ba kell betölteni, majd azon elvégezni az éldetektálást. Végül VGA monitoron megjeleníteni az eredményt. Ehhez szükséges a megfelelő bemeneti kép előállítása, melyet MATLAB® segítségével készítünk el. A képet szürkeárnyalatosra kell konvertálni (8 bites adatformátum), majd azt txt fájlként a projekthez adni.

A feladat megvalósításához egy Logsys Spartan-6 FPGA kártya és kiegészítő VGA modul állt rendelkezésre. Illetve egy VGA csatlakozóval rendelkező monitor is szükséges volt a teszteléshez. A feladatot Verilog nyelven készítettük el.

Modul szintű blokkvázlat



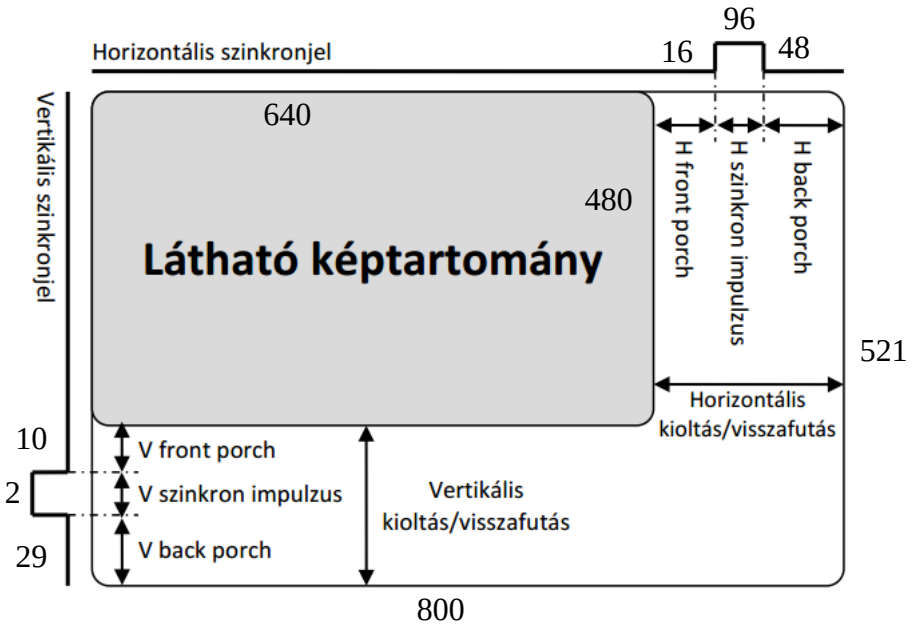
A fenti blokkvázlaton látható, hogy az elkészült feladat három modulra bontható: clk_gen.v, img_vga.v és a sobel.v. Ezen kívül a top.v és pins.ucf fájlok kerültek megvalósításra.

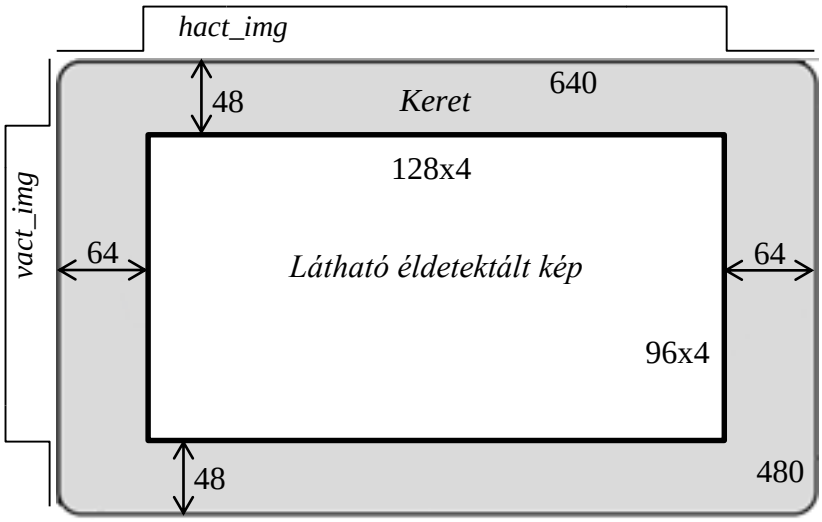
Modulok részletes leírása

A *top.v* top_modul fogja össze a projektet, itt van példányosítva az órajel felezést megvalósító és a kijelzést, éldetektálást végző modul.

Az 50MHz-es órajelet a VGA használata miatt felezni kell, mivel az 25MHz-es frekvenciával működik. Ennek megvalósítása DCM használatával történik. A bejövő órajelet bufferelјük az IBUFG constrain segítségével. Az órajel felezéséhez teljesen elegendő egy DCM-t bekonfigurálni a 4 közül. Paraméterként meg kellet adnunk a bemeneti 50Mhz-es óra periódusidejét (20.0ns) és be kellett állítani a CLKFX_DIVIDE és CLKFX_MULTIPLY segítségével az órajel felezést. CLKFX-ről vettük le a felezett órajelet, amit egy BUFG bufferen keresztül juttatunk el a többi modulhoz.

A következő modul az *img_vga.v*. Ebben van megvalósítva a kép inicializálása a Blokk RAM-ba, illetve kijelzésért felelős jelek előállítása. Ezen kívül itt történik a sobel algoritmus meghívása is. Az *im_vga.v* modulban először a VGA-hoz szükséges jelek előállítása történik. Mivel a memóriába csak korlátozott méretű adat tölthető be, ezért mi a 4:3-os képarány megtartása mellett a 128x96 pixeles felbontást választottuk. A megjelenítési módot, illetve az ehhez szükséges jeleket az alábbi ábrákon foglaltuk össze, ahol a számok pixelekben értendők:





Az ábrák alapján látható, hogy az alapból 128x96-os képet négyszeres nagyításban és középre rendezve ábrázoljuk.

A modul elején egy *initial* blokkban történik a kép beolvasása. (Választhatunk három kép közül fordítás előtt.) A modulban található egy számláló, mely a 800x521 pixeles felbontásban számolja az aktuális pozíciót. Itt egy vertikális (*vcntr*) és horizontális (*hcntr*) számlálóról van szó. Ezeknek az értéke alapján generáltuk le az összes vezérlőjelet:

- *hsync* és *vsync*: szinkronjelek
- *hact_reg* és *vact_reg*: a látható képtartomány jelzésére vízszintes és függőleges irányban
- *hact_img* és *vact_img*: a középre pozícionált éldetektált kép megjelenítéséhez
- *act* és *act_img*: jelezzük, hogy a látható képtartományt, illetve azon belül az érvényes kép helyét rajzoljuk e ki az adott számláló értékekkel

Miután a vezérlőjelek előálltak, kiválasztjuk a *pix_cntr* változó segítségével a kép megfelelő egyetlen pontját. Ezután erre a pontra, pontosabban ezt körülvevő 8 pontra elvégezzük a konvolúciót a sobel maszkokkal (3x3 blokk), mely egy küszöbértéktől függően egyetlen bittel tér vissza: 1, ha él és 0, ha nem detektált élt. Fontos, hogy a kép széleinél ez az algoritmus „bajban lenne”. Ennek kiküszöbölésére ilyenkor 0-val hívjuk meg, vagyis biztosan fekete lesz majd a keret. Ezután már csak a kimeneti változók előállítására kerül sor. Az *rgb* tartalmazza az adott képpont színét. Itt lehetőség van a képet körülvevő keret színének módosítására.

A Sobel algoritmust megvalósító modulnak az aktuális pixelt körülvevő 8 képpontot kell beadni a következőképpen, ahol a p4 az aktuális pixel:

p0	p1	p2
p3	p4	p5
p6	p7	p8

Az éldetektáláshoz az alábbi két mátrixot kell használni, mellyel a horizontális és vertikális deriváltját számoljuk ki a képnek. Ezeket Sobel maszknak is szokták nevezni:

-1	0	+1
-2	0	+2
-1	0	+1

G_x

+1	+2	+1
0	0	0
-1	-2	-1

G_y

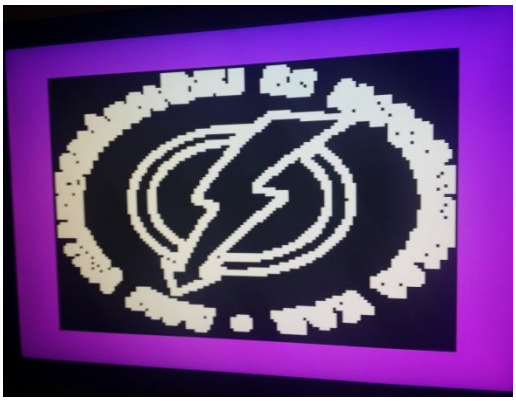
A 3x3 pixelekből álló mátrixot először a G_x-szel, majd G_y-nal kell elemenként összeszorozni. Ezután ezek abszolút értékét kell összegezni. Majd definiálható egy küszöb érték, melyet mi 128-nak vettünk, mely felett élt detektált az algoritmus és amely alatt nem tekintjük élnek az adott pontot.

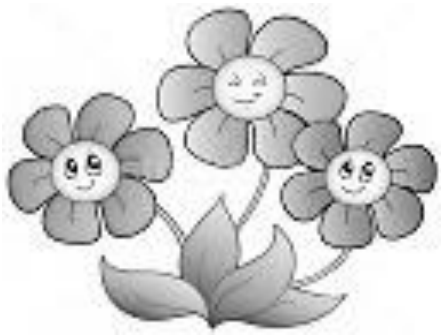
A Sobel tehát a p4-es pixelről állapítja meg, hogy él vagy sem a körülötte lévő pixelek alapján.

Eredmény

A feladat megvalósítása során szimulációt csak a kezdeti fázisban alkalmaztunk, amikor még nem volt meg a VGA kezelő modul. Ekkor szimuláció során beolvastuk a képet, és egy fájlba írattuk ki az eredményt. Ezután MATLAB-ba beolvasva ábrázoltuk a kapott képet, és megvizsgáltuk, hogy működik-e az algoritmus. Ehhez összehasonlításként egy scriptet is írtunk, mely elvégezte a Sobel éldetektáló algoritmust a képen. Miután ez sikerült a VGA modul tesztelése a VGA kiegészítő kártya és monitor segítségével történt.

Íme néhány kép, melyekkel teszteltük a működést:





HDL kódok

top.v:

```
module top_level(
	input xclk,
	input rst,
	output [5:0] xrgb,
	output  xvs,
	output  xhs
	);

//órajel felezés: 25MHz a VGA miatt
	wire clk;
	clk_gen clk_gen_0(
		.clk_in(xclk),
		.clk_out(clk)
	);

//kép beolvasás, a modulon belül sobel algoritmus, majd a képernyõre rajzoltatás
	img_vga img_vga_0(
		.clk(clk),
		.rst(~rst),
		.rgb(xrgb),
		.vsync(xvs),
		.hsync(xhs)
```

```
    );
endmodule//////////////////////////////////////////////////////////////////////////////
////////
```

clk_gen.v:

```
module clk_gen(
  input  clk_in,
  output clk_out
);

wire clk_in_bufg;
IBUFG ibufg_in (                        //Bejövõ órajel bufferelése
  .O(clk_in_bufg),
  .I(clk_in)
);

DCM_CLKGEN #(
  .CLKFXDV_DIVIDE(2),                   // Ezt az osztót nem használjuk
  .CLKFX_DIVIDE(64),                    // Ezzel állítjuk be a 64 es osztást
  .CLKFX_MD_MAX(0.0),
  .CLKFX_MULTIPLY(32),                  // 32-vel szorozva kijön az órajel felezés
  .CLKIN_PERIOD(20.0),                  // 20ns - 50MHz
  .SPREAD_SPECTRUM("NONE"),
  .STARTUP_WAIT("FALSE")
)
DCM_CLKGEN_0 (
  .CLKFX(clkfx),                        // Innen vesszük le a 32/64-el osztott órajelet
  .CLKFX180(),
  .CLKFXDV(),                           // Innen vehetnénk le CLKFX CLKFXDV_DIVIDE-al ossztottját
  .LOCKED(),
  .PROGDONE(),
  .STATUS(),
  .CLKIN(clk_in_bufg),                  // 1-bit input: Input clock
  .FREEZEDCM(1'b0),
  .PROGCLK(1'b0),
  .PROGDATA(1'b0),
  .PROGEN(1'b0),
  .RST(1'b0)
);

BUFG bufg_dcm0_clkfx (                  //Kimenõ órajel bufferelése
  .O(clkfx_bufg),
  .I(clkfx)
);

assign clk_out = clkfx_bufg;
endmodule
//////////////////////////////////////////////////////////////////////////////////
```

img_vga.v:

```
module img_vga(
        input clk,
        input rst,
        output [5:0] rgb,
        output vsync,
        output hsync
   );
```

```
        parameter WIDTH = 128;
        parameter HEIGHT = 96;

//kép beolvasása blockram-ba
        (* ram_style = "block" *)
        reg [7:0] img [WIDTH*HEIGHT-1:0];
        initial $readmemh("virag_128_96.txt", img);
        //initial $readmemh("vik_128_96.txt", img);
        //initial $readmemh("kocka_128_96.txt", img);

//horizontális és vertikális pixel számlálók
        reg [9:0] hcntr;
        reg [9:0] vcntr;
        always @(posedge clk)
                if(rst)begin
                        hcntr <= 10'b0;
                        vcntr <= 10'b0;
                end
                else if(hcntr == 799)begin
                        if(vcntr == 520)
                                vcntr <= 0;
                        else
                                vcntr <= vcntr + 1;
                        hcntr <= 0;
                end
                else
                        hcntr <= hcntr + 1;

//hsync jel
        reg hsync_reg;
        always @(posedge clk)
                if(hcntr == 655 | rst)
                        hsync_reg <= 0;
                else if(hcntr == 751)
                        hsync_reg <= 1;

//vsync jel
        reg vsync_reg;
        always @(posedge clk)
                if((vcntr == 489 & hcntr == 799) | rst)
                        vsync_reg <= 0;
                else if(vcntr == 491 & hcntr == 799)
                        vsync_reg <= 1;

//aktuális tartomány horizontálisan
        reg hact_reg;
        always @(posedge clk)
                if(hcntr == 799)
                        hact_reg <= 1;
                else if(hcntr == 639)
                        hact_reg <= 0;

//aktuális tartomány vertikálisan
        reg vact_reg;
        always @(posedge clk)
                if(vcntr == 520 & hcntr == 799)
                        vact_reg <= 1;
                else if(vcntr == 479 & hcntr == 799)
                        vact_reg <= 0;
```

```
//aktuális tartomány horizontálisan     kép kirajzolásához
	reg hact_img;
	always @(posedge clk)
		if(hcntr == 63)
			hact_img <= 1;
		else if(hcntr == 575)
			hact_img <= 0;

//aktuális tartomány vertikálisan kép kirajzolásához
	reg vact_img;
	always @(posedge clk)
		if(vcntr == 47 & hcntr == 799)
			vact_img <= 1;
		else if(vcntr == 431 & hcntr == 799)
			vact_img <= 0;

//aktuális tartomány: látható képtartomány és kép (körülötte keret)
	reg act;
	reg act_img;
	always @(posedge clk)
		if(rst) begin
			act <= 0;
			act_img <= 0;
			end
		else begin
			act <= (hact_reg & vact_reg);
			act_img <= (hact_img & vact_img);
			end

//pixelek címzése
	wire [9:0] vcntr2;
	wire [9:0] hcntr2;
	assign vcntr2 = vcntr - 48;
	assign hcntr2 = hcntr - 64;

	reg  [14:0] pix_cntr;
	always @ (posedge clk)
		if(rst)
			pix_cntr <= 0;
		else
			pix_cntr <= vcntr2[9:2]*WIDTH + hcntr2[9:2];

//3x3 blokkok a sobel algoritmus számára
	reg [7:0] pix_0_reg,pix_1_reg,pix_2_reg,pix_3_reg,pix_5_reg,pix_6_reg,pix_7_reg,pix_8_reg;

	always @ (posedge clk)begin
	    if((pix_cntr <= (WIDTH-1)) |        // a kép szélének kezelése
	      (pix_cntr%WIDTH == 0) |         // elsõ sor, elsõ oszlop, utolsó oszlop, utolsó sor esetén fekete a pixel
	      ((pix_cntr-(WIDTH-1))%WIDTH  ==  0) |
	      (pix_cntr> (WIDTH*HEIGHT-WIDTH)) |
	      ~act_img)

			begin
			pix_0_reg <= 0;                 //így fekete lesz a kép széle
			pix_1_reg <= 0;                 //1 esetén pedig fehér
			pix_2_reg <= 0;
			pix_3_reg <= 0;
			pix_5_reg <= 0;
			pix_6_reg <= 0;
			pix_7_reg <= 0;
```

```
                pix_8_reg <= 0;
                end
            else begin
                pix_0_reg <= img[pix_cntr-WIDTH-1];
                pix_1_reg <= img[pix_cntr-WIDTH];
                pix_2_reg <= img[pix_cntr-WIDTH+1];
                pix_3_reg <= img[pix_cntr-1];
                pix_5_reg <= img[pix_cntr+1];
                pix_6_reg <= img[pix_cntr+WIDTH-1];
                pix_7_reg <= img[pix_cntr+WIDTH];
                pix_8_reg <= img[pix_cntr+WIDTH+1];
             end
        end

//sobel algoritmus
        wire out_data;
        sobel sobel_0(
            .clk(clk),
            .rst(rst),
            .pix_0(pix_0_reg),
            .pix_1(pix_1_reg),
            .pix_2(pix_2_reg),
            .pix_3(pix_3_reg),
            .pix_5(pix_5_reg),
            .pix_6(pix_6_reg),
            .pix_7(pix_7_reg),
            .pix_8(pix_8_reg),
            .out_data(out_data)
        );

//színek elõállítása
        reg [5:0]rgb_reg;
        always @(posedge clk)
            if(act_img)
                rgb_reg <= {6{out_data}};
            else if(act)
                rgb_reg <= 6'b101011;                   //itt lehet a keretnek színt adni
            else
                rgb_reg <= 6'b0;

//kimenetek
        assign rgb = rgb_reg;
        assign hsync = hsync_reg;
        assign vsync = vsync_reg;

endmodule//////////////////////////////////////////////////////////////////////////////
////////
```

sobel.v:

```
module sobel(
            input clk,
            input rst,
            input [7:0] pix_0,pix_1,pix_2,pix_3,pix_5,pix_6,pix_7,pix_8,
            output out_data
            );

wire signed [10:0] gx, gy;                  //11 bit: gx es gy max ertekei: 255*4 + 1
wire signed [10:0] abs_gx, abs_gy;          //absz.ertek
wire [10:0] sum;                            //kimenet: max 255*8 bit lehet
```

```
assign gx =((pix_2-pix_0) + ((pix_5-pix_3)<<1) + (pix_8-pix_6));        //sobel maszk (horizontális)
assign gy =((pix_0-pix_6) + ((pix_1-pix_7)<<1) + (pix_2-pix_8));        //sobel maszk (vertikális)

assign abs_gx = (gx[10] ? ~gx+1 : gx);          //ha negativ: absz erteket veszem
assign abs_gy = (gy[10] ? ~gy+1 : gy);

assign sum = abs_gx + abs_gy;                   //x es y irany osszeadasa

assign out_data = |sum[10:7];                   //ha nagyobb 128-nál a kimenet 1 (él); 0 (nem él)

endmodule
//////////////////////////////////////////////////////////////////////////////////
```

pins.ucf:

```
NET "xclk" LOC="P55";
NET "rst" LOC="P67" | PULLUP;

NET "xrgb<5>" LOC="P137" | IOSTANDARD=LVCMOS33;
NET "xrgb<4>" LOC="P138" | IOSTANDARD=LVCMOS33;
NET "xrgb<3>" LOC="P139" | IOSTANDARD=LVCMOS33;
NET "xrgb<2>" LOC="P140" | IOSTANDARD=LVCMOS33;
NET "xrgb<1>" LOC="P141" | IOSTANDARD=LVCMOS33;
NET "xrgb<0>" LOC="P142" | IOSTANDARD=LVCMOS33;
NET "xvs"     LOC="P134" | IOSTANDARD=LVCMOS33;
NET "xhs"     LOC="P133" | IOSTANDARD=LVCMOS33;

NET "xclk" TNM_NET = xclk;
TIMESPEC TS_xclk = PERIOD "xclk" 50 MHz HIGH 50%;
```

Felhasznált források

http://logsys.mit.bme.hu/sites/default/files/page/2009/09/LOGSYS_VGA_PS2_modul.pdf

http://www.sciencepublication.org/ijast/documents/proceeding/46.pdf