Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Logikai tervezés házi feladat
Videó feldolgozás: Sobel éldetektálás

Hantos Márton
(I6FRZS)

Fucskó Norbert
(A50PVY)

2022

Tartalomjegyzék

1. Házi feladat specifikációja

1.1. Házi feladat kiírás

Az egység 8 bites RGB értékeket tartalmazó videón végez éldetektálást Sobel algoritmus segítségével. A be- és kimenet a Logsys Kintex-7 kártya HDMI interfésze, ennek megvalósítása nem része a házi feladatnak, kész modulként rendelkezésre áll.

1.2. Magasszintű leírása a feladatnak

A feladat HDMI interfészen keresztül videót feldolgozni, azon éldetektálást elvégezni. Ennek elvégzéséhez egy PC HDMI portját csatlakoztattuk a Logsys Kintex-7 kártya HDMI bemenetére, kimenete pedig egy monitornak adta a feldolgozott jelet.

Az éldetektálás Sobel algoritmus segítségével valósult meg, melyben egy vízszintes és egy függőleges Sobel operátorral konvolváltuk a bemeneti videót. Mindkét esetben egy 3x3-as méretű kernelt csúsztatunk végig a képeken, és lépésenként előáll egy szám a konvolúció eredményeként. A konvolúció definíciója:

$$(k * I)(x, y) = \sum_{u=-n}^{n} \sum_{v=-n}^{n} k(u, v) * I(x - u, y - v)$$

Ahol *I(x, y)* az *y.* képsor *x.* pixele, *k* pedig a kernel. A két kernel közül $\boldsymbol{G_x}$ a függőleges élek detektálására alkalmas szűrő, míg $\boldsymbol{G_y}$ a vízszintes élek detektálására használatos.

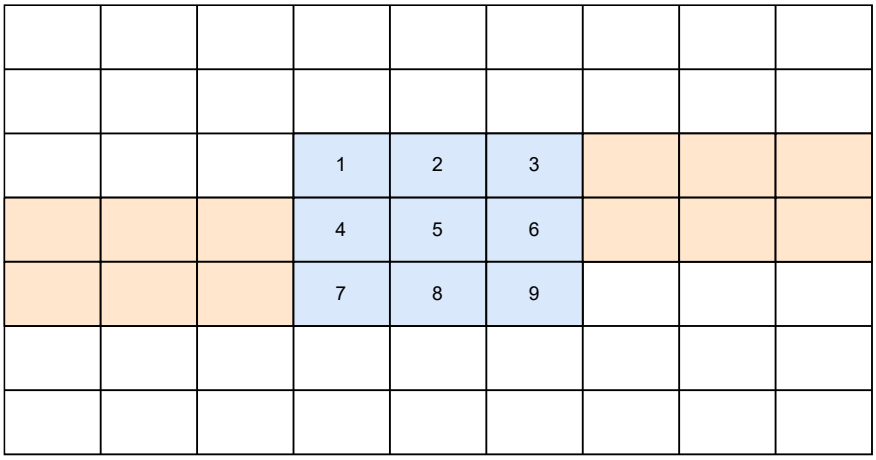
$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

1.3. Megvalósítás FPGA-n

Funkcionálisan 4 nagyobb részre osztható a probléma megoldása: egyrészt az FPGA-ban a HDMI kapcsolatot megvalósító bemeneti-, illetve kimeneti interfészek, az algoritmus megvalósításához pedig a sorok tárolásához szükséges sorbuffer, végül pedig az éldetektálást végző kernel.

A HDMI átalakító modulból a hagyományos videójelek érkeznek, ezek a színenként 8 bites pixel értékek, egy horizontális- és vertikális szinkron jel (hsync, vsync), illetve az aktuálisan érkező pixel láthatóságát mutató jel (de), a pixelek egymás után folyamként érkeznek. A Sobel algoritmus a képen egy 3x3 méretű kernelt húz végig, tehát egyszerre 9 pixelt kell biztosítani az algoritmust megvalósító modulnak.



1-1. Sorok tárolása

A fenti ábrán látható, hogy ahhoz, hogy 9 pixelt biztosítsunk az éldetektálást végző modulnak, szükséges eltárolni a képből 2 egész sort, illetve még 3 pixelt. Ezt a feladatot a sorbuffer látja el, tehát az a feladat, hogy egy olyan modult tervezzünk, ami egy sor késleltetéssel biztosítja a feldolgozandó pixel mellett a körülötte levő pixeleket is egyidejűleg.

Végül pedig magát az éldetektálást megvalósító modul szükséges, ez a modul a csatornánként beérkező pixeleken elvégzi az éldetektálást. A modul kimenetén 255 vagy 0 jelenik meg annak a függvényében, hogy van-e él, vagy sem, tehát az élek teljesen fehér pixelek lesznek, ahol nincsenek élek azok pedig feketék.

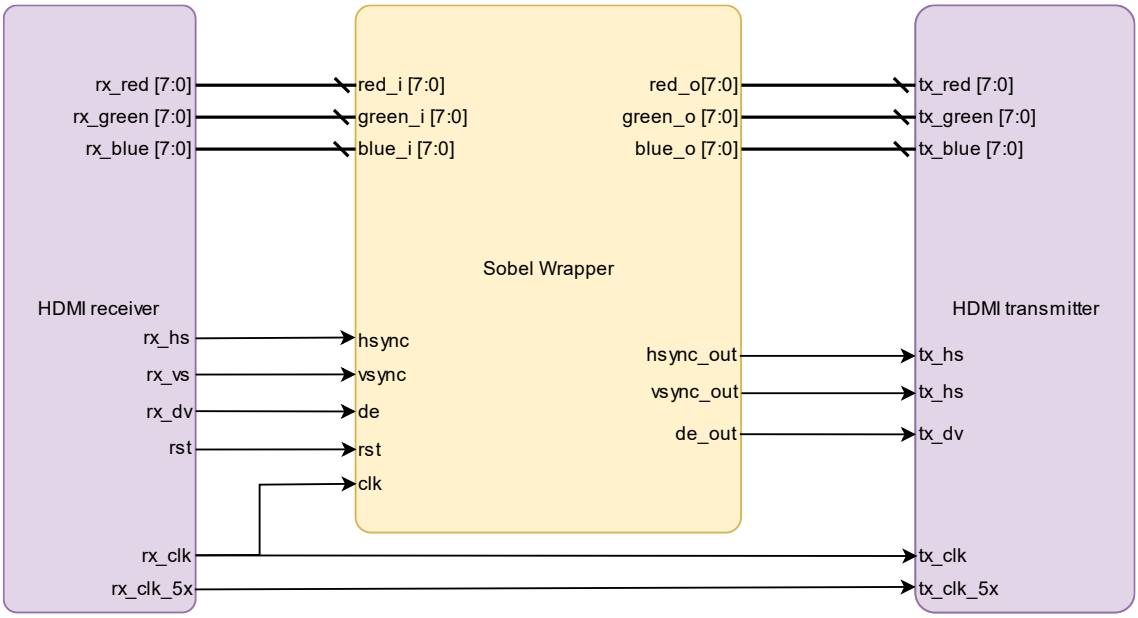
2. Implementáció

Az implementáció során a specifikációban említett 2 általunk megvalósított modulnak létrehoztunk egy burkoló modult, így az egész egy egységként példányosítható a top modulban.

A modulok egymás között a HDMI vevő által biztosított hagyományos videó jel formátumot használják a pixelek átadására, így a sorbuffer bemenetén is ilyen adatot vár. Az általunk megvalósított Sobel feldolgozó egységnek elég a feldolgozandó pixelhez tartozó szinkronizációs jeleket átadni, az őt körülölelő 8 további pixelnek csak a 8 bites értéke szükséges.

2.1. Top level

A feladat megoldása során az elkészült modult Sobel Wrapper néven illesztettük be a HDMI receiver és HDMI transmitter modulok közé. A HDMI receiver és transmitter közvetlenül a kártyán található HDMI csatlakozók lábaira vannak kötve.



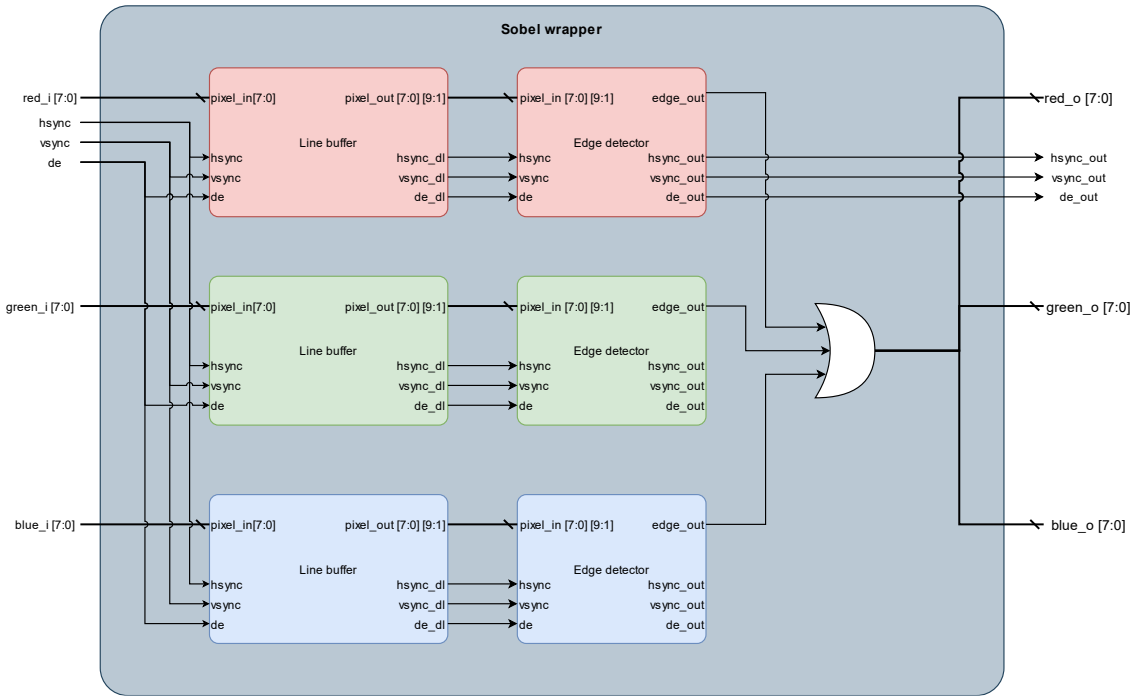
2-1. Top level blokkvázlata

2.2. Sobel Wrapper

A Sobel Wrapper modul feladata, hogy a bemeneti három színcsatornához példányosítsa háromszor a line buffer és az edge detector modulokat. Ezeken külön-külön végrehajtódnak a lentebb kifejtett műveletek, majd előáll mindhárom csatornára egy *edge_out* érték, ami azt jelzi, hogy az adott pixel környezetében lett-e él detektálva vagy sem.

A modul kimenetei a *hsync_out, a vsync_out,* és *de_out,* melyek a HDMI vezérlőjelek megfelelően elkésleltetve, illetve csatornánként 8 bit, tehát összesen 24 bit RGB pixelérték. Mivel az edge detector csatornánként 1 bitet ad ki, ezért a wrapper felelős azért, hogy a három érték VAGY kapcsolatát képezze, illetve a kimeneti *blue_o, green_o,* és *red_o*

mindegyik bitjére ezt az eredményt állítsa be. Ennek az az eredménye, hogy ha bármelyik csatornán él kerül detektálásra, akkor mind a három csatornára a kimeneti bitek értéke egyesével 1 lesz, tehát egy vastagabb vonalat fogunk érzékelni, ellenkező esetben pedig természetesen 0 lesz az összes kimeneti bit értéke. Ez csupán személyes preferencia, akár csatornánként adhatnánk különböző kimeneti jelet, viszont ez minimálisan változtatna a végső kép esztétikáján.

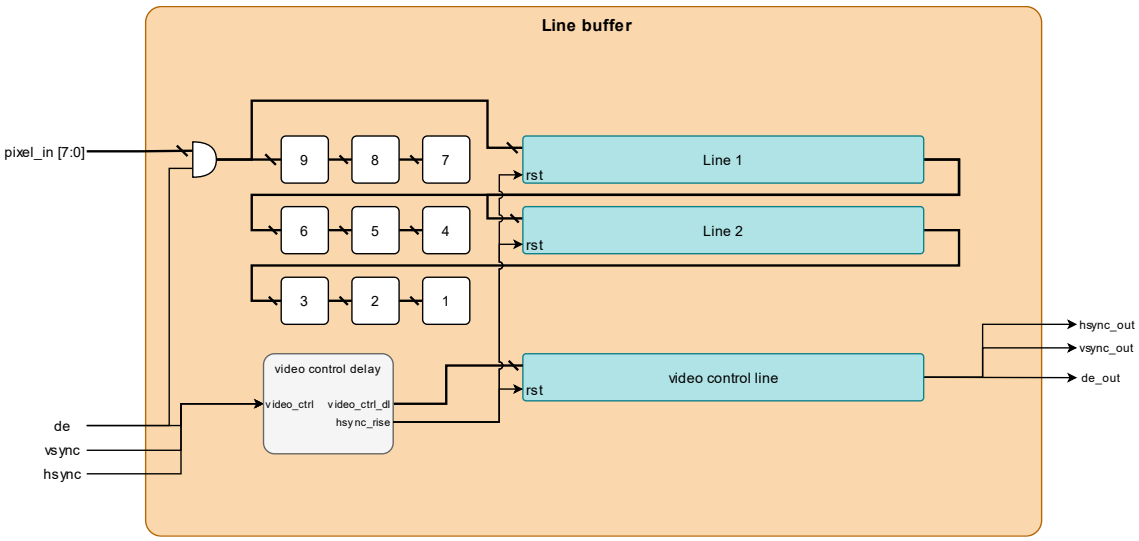


2-2. Sobel Wrapper modul blokkvázlata

A line buffer és az edge detector megvalósításában benne van a *hsync*, *vsync*, *de* késleltetése, de ez minden csatornára megegyező eltolás lesz, így bár létrehoztunk 3 széles *hsync_dl*, *vsync_dl*, *de_dl* regisztereket, de csupán az egyik (piros csatorna) jeleit vezettük a kimenetre.

2.3. Line buffer

A rendszerben a pixelsorok eltárolását, illetve egyszerre a 9 feldolgozandó pixel biztosítását a line buffer végzi. A rendszer blokkvázlata az alábbi ábrán látható:



2-3. Line buffer blokkvázlata

A modul bemenetére a hagyományos videó jeleket kapja. Ahhoz, hogy a modulban az érkező kép egymás alatti pixeleit tudja kiadni, késleltetni kell a beérkező jelet egy sorral, ezzel egyidejűleg pedig CLB flip-flopokba tárolja a modul a pixeleket. A késleltetési feladatot a line delay modul látja el.

A line buffer feladata továbbá, hogy a kép széleinél megvalósítsa a paddinget, erre azt a megoldást használtuk, hogy a bemeneti pixelt akkor kapja meg a line delay modul, ha az adott pixel a kép látható tartományában van, egyébként 0 érték kerül bele.

Mivel a videó felbontása a feldolgozás során ismeretlen, ezért a hsync jel felfutó élét használtuk a line delay modulok reseteléséhez, ez a belső megvalósítás miatt szükséges (cirkuláris buffer, részletesebben a modul leírásánál).

Ahhoz, hogy az aktuálisan feldolgozott pixellel (tehát ebben az esetben az 5-ös indexű pixellel) szinkronban érkezzenek a hozzá tartozó video control jelek, ezért azokat is el kell késleltetni, hogy a kép megfelelően jelenjen meg a monitoron.

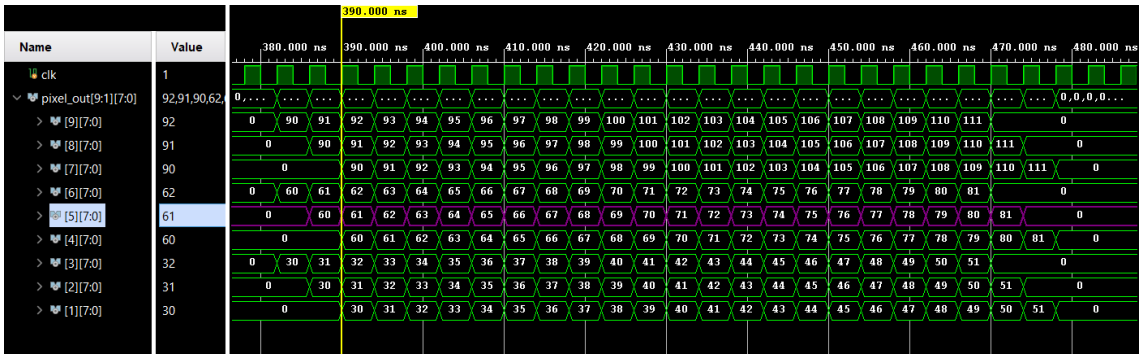
A Line buffer modul példányosításánál megadható a maximális sorméret (beleszámolva a blanking pixeleket is). Mi az alkalmazásunk során ezt az értéket 2200-ra választottuk meg, hagyományos blanking paraméterekkel így a rendszer képes helyesen feldolgozni egy 1600x900-as képet. A paraméter értéke ebben a dizájnban a Sobel Wrapper példányosítása során adható meg.

A modul működése szimuláción

A modul működését szimulációval ellenőriztük. A bemeneti jeleket a szimuláció során generáltuk úgy, hogy két számlálót léptettünk. A szimulációban egy kép szélessége 30 pixel, a pixelek értékei az adott képen belül az aktuális pozíciójuk (tehát a *h_cntr + 30 * v_cntr*), ebből az első 22 pixel található a látható tartományban. A *hsync* jel a soron belül a 24. pixelnél 1 értékű, egyébként 0, ezzel szimulálva a *hsync* impulzusszerű működését.

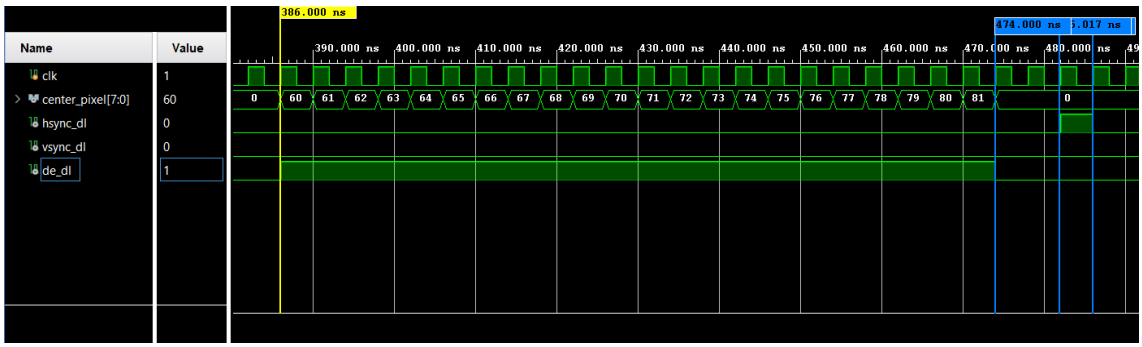
Az első hullámformán azt ellenőrizzük, hogy a kimenő pixelek a 9 elemű pixel mátrixban egymáshoz képest pontosan jó időzítéssel vannak elkésleltetve, vagyis a 9 kimeneti pixel

valóban a specifikációban szemléltetett módon helyezkedik el egymáshoz képest. Ez azt jelenti a szimulációs beállításnak megfelelően, hogy az 1, 2, 3-as indexű pixelek egymást követő értékek, a 4, 5, 6 ezekkel képest 30-cal, a 7, 8, 9 pixelek pedig ezekhez képest 60-nal vannak elkésleltetve. A hullámformán más színnel jelöltük a középső (5-ös indexű) pixelt.



2-4. Line buffer kimeneti pixelek szimulációja

Az ábráról leolvasható, hogy a pixelek egymáshoz képesti eltolása jól működik. A következő lépés a bemeneti generált adatok ismeretében a kimeneti videó vezérlő jelek késleltetésének az ellenőrzése. A következő ábra ezt szemlélteti:



2-5. Line buffer videó control jel szimulációja

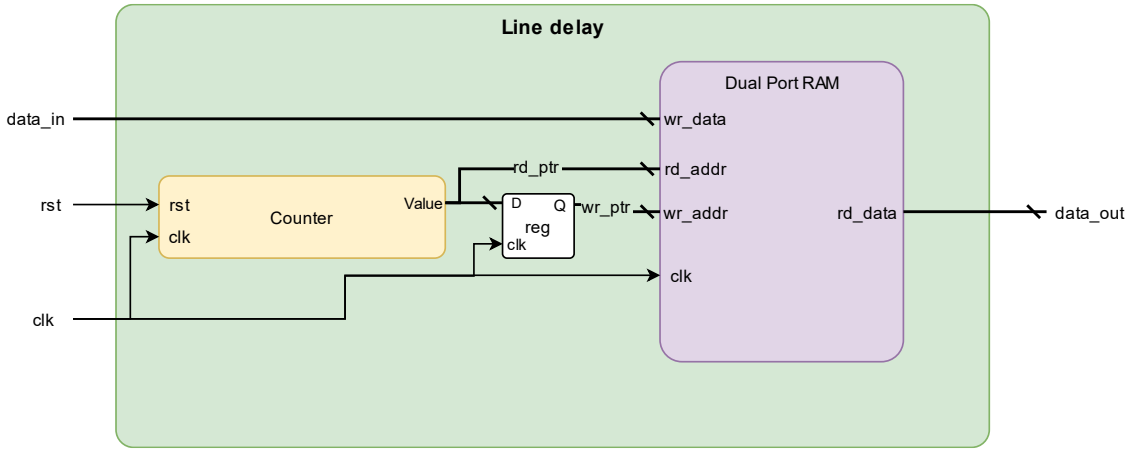
Az ábráról leolvasható, hogy a *de* és *hsync* késleltetett kimeneti videó vezérlő jelek szinkronban vannak a bemeneti generált jellel, így ez szimulációs környezetben igazolja a modul működését.

2.3.1. Line delay

A line buffer modulban az adatok késleltetését egy line delay nevű modul segítségével valósítottuk meg. Egy line delay modul pontosan egy sorral késlelteti el az adatokat. Mivel a videójelekben nagyságrendileg több, mint ezer darab esetünkben 8 bites pixel értéket kell eltárolni, ezért az FPGA BRAM erőforrásait használtuk a késleltetés megvalósítására. Lényegében shift regiszter-szerű működést szeretnénk elérni, erre a cirkuláris buffer nyújtott megoldást.

A modulként felhasznált dual port memóriák címzésével valósítható meg a cirkuláris működés. A címzést két pointer adja, ezek az írási (*wr_ptr*) és olvasási (*rd_ptr*) pointerek. Cirkuláris buffer működése során a *rd_ptr* pont egy egész sornyival a *wr_ptr* előtti érték.

Viszont a mi esetünkben, ha már fel van töltve a buffer, tulajdonképpen a *rd_ptr*-t követi közvetlen a *wr_ptr*, ezért így valósítottuk meg a késleltetés címzését.

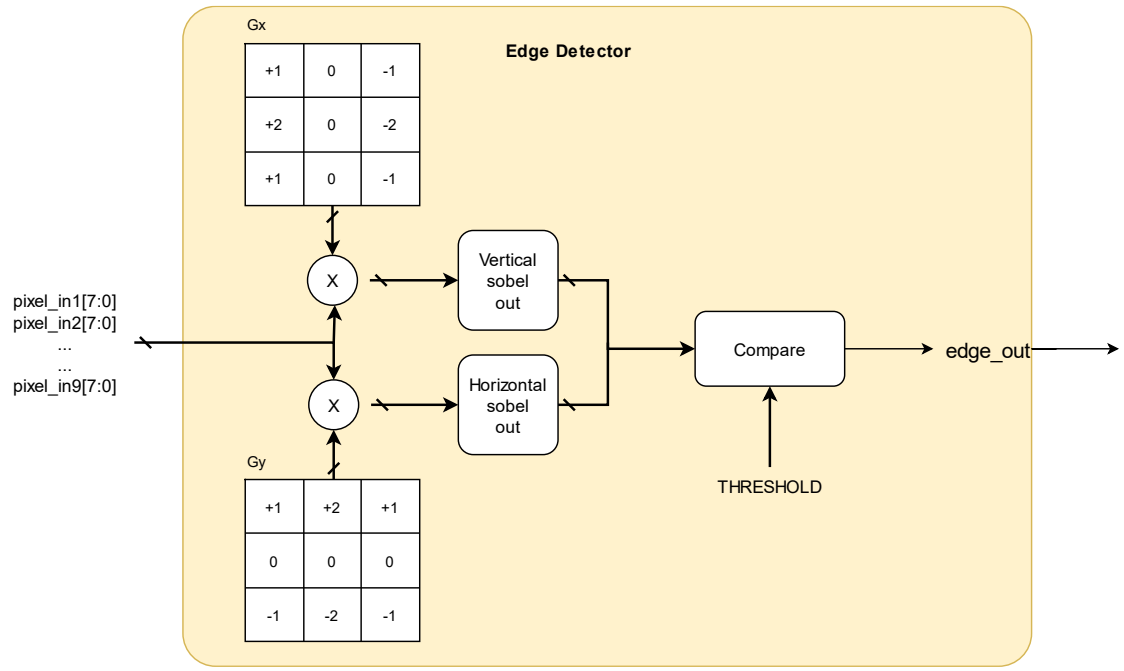


2-6. Line delay modul blokkvázlata

A fenti ábrán látható a modul blokkvázlata, a *rd_ptr* értéke egy számláló kimenete, a *wr_ptr* értéke pedig az előbb említett logika alapján a *rd_ptr* értéke egy órajellel elkésleltetve.

2.4. Edge detector

Az Edge detector modul végzi el az éldetektálás műveletét, tehát összekonvolválja a 3x3-as kernelt a 9 bemeneti pixelértékkel. Az alábbi ábrán látható a blokkvázlata az edge detector funkcionalitásának.



2-7. Edge detector működésének blokkvázlata.

A konvolválás eredményeképp két értéket kapunk, ami azt adja meg, hogy egy adott pixel környezetében van-e függőleges, illetve vízszintes él. Ebből egy értéket akarunk előállítani, melyet egy küszöbértékhez hasonlítva megkaphatjuk, hogy van-e él a pixel környezetében vagy sem. Ez többféle módon is lehetséges:

- Az első lehetőség a két érték négyzetösszegének a gyökét venni. Két szám négyzetét előállítani és összeadni őket nem túl hardverintenzív, hiszen MAC műveletekkel megoldható a probléma, viszont a gyökképzés feleslegesen növelné a hardverigényt, illetve a komplexitást.
- A második opció, hogy a két érték abszolút értékeinek a maximumát vesszük és ezt hasonlítjuk a küszöbértékhez. Ez logikailag ekvivalens azzal, ha mindkét értéket összehasonlítjuk a küszöbértékkel és a küszöbérték -1-szeresével.

A két lehetőség esztétikailag nagyon hasonló eredményt adna, ezért tekintettel a hardverigényekre az utóbbi megoldást választottuk az implementációban.

További észrevétel, hogy a Sobel szűrő során felhasznált szorzótényezők mindegyike vagy 0, vagy az abszolút értéke 2-nek valamelyik hatványa. Ez azt jelenti, hogy a szorzás során nincs szükségünk DSP blokkokra, mivel egyszerűen shifteléssel vagy az adott bit 0-ba állításával elvégezhető ez a művelet. A szintézis után a reportot megtekintve felfedezhető, hogy ez optimalizálásra is került, mivel egy DSP blokkot sem használ az FPGA.

Megemlítendő még, hogy a *vertical_sobel_out* és a *horizontal_sobel_out* miért lettek 11 bit szélesek. Ezt azért választottuk meg így, mert a legnagyobb elképzelhető abszolútértékük 4*255 = 1020. Ez abból következik, hogy egy sor/oszlop együtthatói +1, +2, +1 vagy ennek -1-szeresei, amit összegezve és megszorozva a legnagyobb lehetséges pixel értékkel 1020-at kapunk. Ha csak szimplán pozitív értékekkel számolnánk, akkor erre 10 bit elég lenne, viszont mivel ez egy signed regiszter, ezért 11 bit nagyságúnak kell lennie.

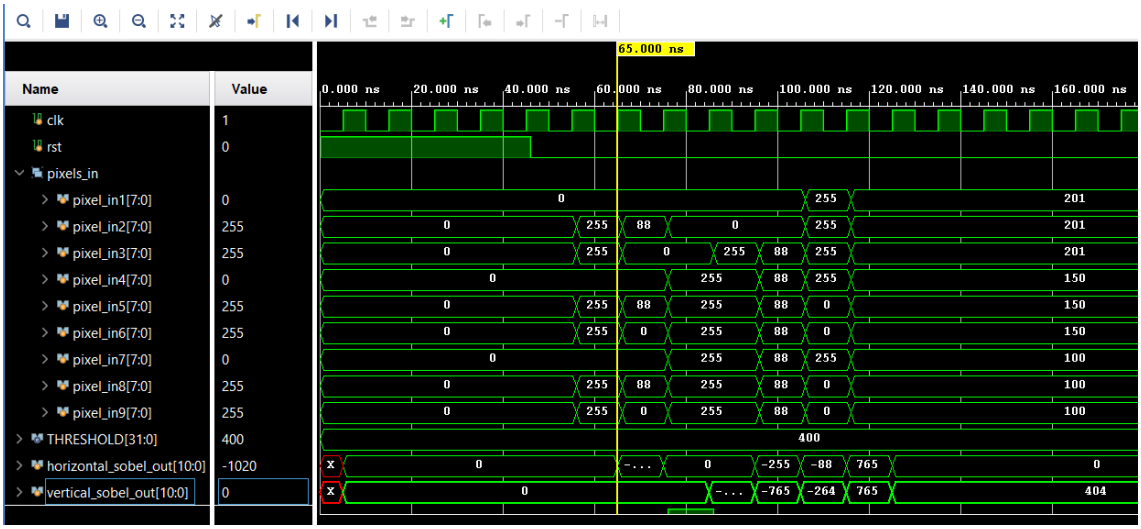
A modul belsejében kezeljük a *hsync, vsync, de* jelek késleltetését is, melyet egy 2 bit szélességű shift regiszterrel valósítottunk meg, mivel a kimenet előállítása is 2 órajel késleltetéssel történik meg a pixelek mintavételezéséhez képest.

A modul működése szimuláción

A modul tesztelése során megvizsgáltuk a pixel értékek és a *horizontal_sobel_out* és *vertical_sobel_out* helyes működését. A kombinációk az alábbi ábrán láthatóak. A következő esetek kerültek tesztelésre:

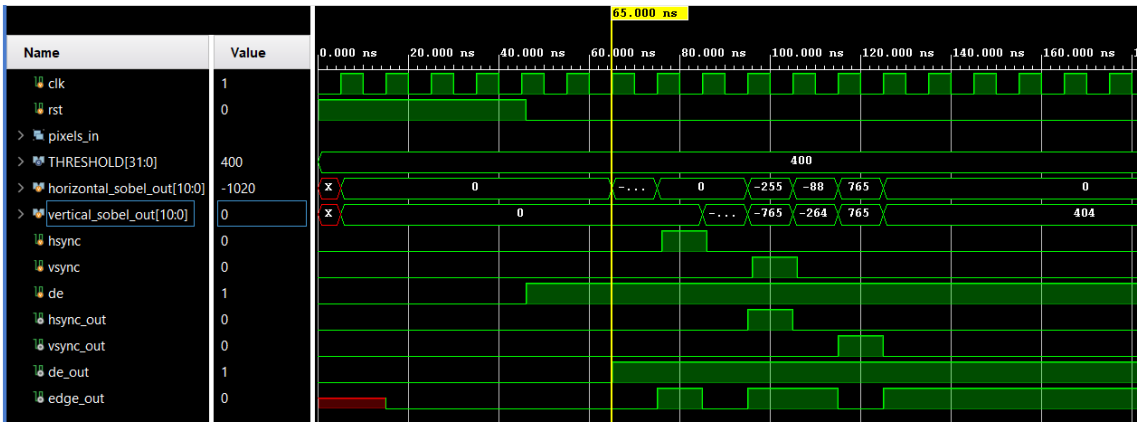
1. Minden pixel 0, ekkor mindkét kimenet helyesen 0.
2. Horizontális él, ahol a kép egyik és másik oldala között maximális az eltérés (0-255), ekkor a *horizontal_sobel_out* helyesen -1020.
3. A kép közepén függőlegesen 0-tól eltérő azonos értékek, a többi pozícióban pedig 0. Ekkor nem lesz él detektálva, és mindkét Sobel művelet kimenete 0.
4. Vertikális él, ahol a kép egyik és másik oldala között maximális az eltérés. Ebben az esetben is -1020 lesz a megfelelő sobel_out kimenet értéke.
5. Horizontális és vertikális él is található a képen, ekkor látható, hogy mindkét sobel_out kimeneti érték 0-tól eltérő.

6. Előző eset, de kisebb eltérésekkel, ekkor látható, hogy az *edge_out* nem vált 1-be.
7. Diagonális él, ebben az esetben is sikeres az éldetektálás.
8. Teszt eset, ahol minimálisan nagyobb az egyik sobel_out, mint a küszöbérték (itt a *THRESHOLD* 400).



2-8. Edge detector modul szimulációja, a -… értékek helyén -1020 szerepel.

Az alábbi ábrán pedig látható, hogy a *hsync, vsync, de* jelek is megfelelően lettek késleltetve, hiszen a modul bemenete és kimenete között 2 órajel késleltetés van, ez teljesül ezekre a jelekre is. Továbbá megfigyelhető, hogy az edge_out jel is helyesen vált a 2 sobel_out jel és a *THRESHOLD* függvényében.



2-9. Edge detector szimulációja, feltüntetve a *hsync, vsync, de* jeleket.

3. Dizájn tesztelése

A Vivado 2021.2-es szoftverrel a projektből generáltunk egy bitstreamet, azt pedig a tanszéki Logsys Kintex-7 kártyára töltöttük fel.

A szintézis után megvizsgáltuk a Report fájlt is, mely tartalmazza, hogy milyen erőforrásokat használtunk fel. Itt látható, hogy az erőforrásigény viszonylag alacsony, még a három csatorna párhuzamos feldolgozása ellenére is, illetve az előzetes várakozásnak megfelelően a felhasznált DSP blokkok száma 0. Ezek az alábbi képeken láthatóak:

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	671	0	0	41000	1.64
LUT as Logic	647	0	0	41000	1.58
LUT as Memory	24	0	0	13400	0.18
LUT as Distributed RAM	24	0			
LUT as Shift Register	0	0			
Slice Registers	792	0	0	82000	0.97
Register as Flip Flop	792	0	0	82000	0.97
Register as Latch	0	0	0	82000	0.00
F7 Muxes	8	0	0	20500	0.04
F8 Muxes	0	0	0	10250	0.00

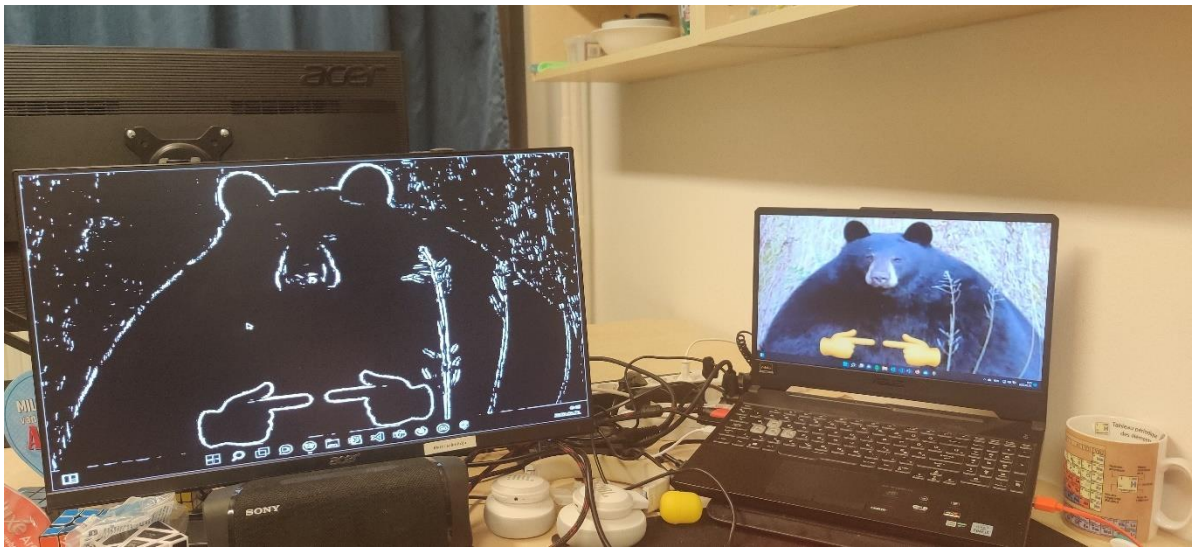
2. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	7.5	0	0	135	5.56
RAMB36/FIFO*	6	0	0	135	4.44
RAMB36E1 only	6				
RAMB18	3	0	0	270	1.11
RAMB18E1 only	3				

3. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	0	0	0	240	0.00

A dizájn működés közben:



A kimeneti képen látható, hogy az éleket jól detektálja a rendszer. A 0 padding miatt a kép körül egy fehér keret fedezhető föl.

Függelék

hdmi_top:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////
// Company:
// Engineer:
//
// Create Date: 04/03/2019 04:56:58 PM
// Design Name:
// Module Name: hdmi_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////
```

```
module hdmi_top(
    input  wire       clk100M,
    input  wire       rstbt,
    output wire [7:0] led_r,
    input  wire [7:0] sw,
    inout  wire [3:0] bt,

    input  wire       hdmi_rx_d0_p,
    input  wire       hdmi_rx_d0_n,
    input  wire       hdmi_rx_d1_p,
    input  wire       hdmi_rx_d1_n,
    input  wire       hdmi_rx_d2_p,
    input  wire       hdmi_rx_d2_n,
    input  wire       hdmi_rx_clk_p,
    input  wire       hdmi_rx_clk_n,
    input  wire       hdmi_rx_cec,
    output wire       hdmi_rx_hpd,
    input  wire       hdmi_rx_scl,
    inout  wire       hdmi_rx_sda,

    output wire       hdmi_tx_d0_p,
    output wire       hdmi_tx_d0_n,
    output wire       hdmi_tx_d1_p,
    output wire       hdmi_tx_d1_n,
    output wire       hdmi_tx_d2_p,
```

```
    output wire         hdmi_tx_d2_n,
    output wire         hdmi_tx_clk_p,
    output wire         hdmi_tx_clk_n,
    input  wire         hdmi_tx_cec,
    input  wire         hdmi_tx_hpdn,
    input  wire         hdmi_tx_scl,
    input  wire         hdmi_tx_sda
);
```

```
//*************************************************************************
**
//* Generating the 200 MHz reference clock for the IDELAYCTRL.
*
//*************************************************************************
**
wire clk200M;
wire pll_clkfb;
wire pll_locked;
```

```
PLLE2_BASE #(
    .BANDWIDTH("OPTIMIZED"),          // OPTIMIZED, HIGH, LOW
    .CLKFBOUT_MULT(10),               // Multiply value for all CLKOUT, (2-64)
    .CLKFBOUT_PHASE(0.0),             // Phase offset in degrees of CLKFB, (-
360.000-360.000).
    .CLKIN1_PERIOD(1000.0 / 100.0),   // Input clock period in ns to ps reso-
lution (i.e. 33.333 is 30 MHz).
    .CLKOUT0_DIVIDE(5),               // CLKOUT0_DIVIDE - CLKOUT5_DIVIDE: Divide
amount for each CLKOUT (1-128)
    .CLKOUT1_DIVIDE(1),
    .CLKOUT2_DIVIDE(1),
    .CLKOUT3_DIVIDE(1),
    .CLKOUT4_DIVIDE(1),
    .CLKOUT5_DIVIDE(1),
    .CLKOUT0_DUTY_CYCLE(0.5),         // CLKOUT0_DUTY_CYCLE -
CLKOUT5_DUTY_CYCLE: Duty cycle for each CLKOUT (0.001-0.999).
    .CLKOUT1_DUTY_CYCLE(0.5),
    .CLKOUT2_DUTY_CYCLE(0.5),
    .CLKOUT3_DUTY_CYCLE(0.5),
    .CLKOUT4_DUTY_CYCLE(0.5),
    .CLKOUT5_DUTY_CYCLE(0.5),
    .CLKOUT0_PHASE(0.0),              // CLKOUT0_PHASE - CLKOUT5_PHASE: Phase
offset for each CLKOUT (-360.000-360.000).
    .CLKOUT1_PHASE(0.0),
    .CLKOUT2_PHASE(0.0),
    .CLKOUT3_PHASE(0.0),
    .CLKOUT4_PHASE(0.0),
    .CLKOUT5_PHASE(0.0),
    .DIVCLK_DIVIDE(1),                // Master division value, (1-56)
    .REF_JITTER1(0.0),                // Reference input jitter in UI, (0.000-
0.999).
    .STARTUP_WAIT("FALSE")            // Delay DONE until PLL Locks,
("TRUE"/"FALSE")
) clk_generator1 (
    .CLKOUT0(clk200M),                // 1-bit output: CLKOUT0
```

```
    .CLKOUT1(),                     // 1-bit output: CLKOUT1
    .CLKOUT2(),                     // 1-bit output: CLKOUT2
    .CLKOUT3(),                     // 1-bit output: CLKOUT3
    .CLKOUT4(),                     // 1-bit output: CLKOUT4
    .CLKOUT5(),                     // 1-bit output: CLKOUT5
    .CLKFBOUT(pll_clkfb),           // 1-bit output: Feedback clock
    .LOCKED(pll_locked),            // 1-bit output: LOCK
    .CLKIN1(clk100M),               // 1-bit input: Input clock
    .PWRDWN(1'b0),                  // 1-bit input: Power-down
    .RST(rstbt),                    // 1-bit input: Reset
    .CLKFBIN(pll_clkfb)             // 1-bit input: Feedback clock
);
```

```
wire rst;
assign rst = ~pll_locked;
```

```
wire clk_200M;
BUFG BUFG_200M (
    .O(clk_200M),
    .I(clk200M)
);
```

```
wire rx_clk, rx_clk_5x;
wire [7:0] rx_red, rx_green, rx_blue;
wire rx_dv, rx_hs, rx_vs;
wire [5:0] rx_status;
hdmi_rx hdmi_rx_0(
    .clk_200M(clk_200M),
    .rst(rst),
    .hdmi_rx_cec(hdmi_rx_cec),
    .hdmi_rx_hpd(hdmi_rx_hpd),
    .hdmi_rx_scl(hdmi_rx_scl),
    .hdmi_rx_sda(hdmi_rx_sda),
    .hdmi_rx_clk_p(hdmi_rx_clk_p),
    .hdmi_rx_clk_n(hdmi_rx_clk_n),
    .hdmi_rx_d0_p(hdmi_rx_d0_p),
    .hdmi_rx_d0_n(hdmi_rx_d0_n),
    .hdmi_rx_d1_p(hdmi_rx_d1_p),
    .hdmi_rx_d1_n(hdmi_rx_d1_n),
    .hdmi_rx_d2_p(hdmi_rx_d2_p),
    .hdmi_rx_d2_n(hdmi_rx_d2_n),
    .rx_clk(rx_clk),
    .rx_clk_5x(rx_clk_5x),
    .rx_red(rx_red),
    .rx_green(rx_green),
    .rx_blue(rx_blue),
    .rx_dv(rx_dv),
    .rx_hs(rx_hs),
    .rx_vs(rx_vs),
    .rx_status(rx_status)
);
```

```
// Loopback
// Replace with image processing block
wire [7:0] tx_red, tx_green, tx_blue;
wire tx_dv, tx_hs, tx_vs;
// Instantiate wrapper
sobel_wrapper#(
    .MAX_LINE_WIDTH(2200)
)sobel_wrapper_u(
    .clk(rx_clk),
    .rst(rst),
    .red_i(rx_red),
    .green_i(rx_green),
    .blue_i(rx_blue),
    .de(rx_dv),
    .hsync(rx_hs),
    .vsync(rx_vs),
    .red_o(tx_red),
    .green_o(tx_green),
    .blue_o(tx_blue),
    .de_out(tx_dv),
    .hsync_out(tx_hs),
    .vsync_out(tx_vs)
);
```

```
hdmi_tx hdmi_tx_0(
    .tx_clk(rx_clk),
    .tx_clk_5x(rx_clk_5x),
    .rst(rst),
    .tx_red(tx_red),
    .tx_green(tx_green),
    .tx_blue(tx_blue),
    .tx_dv(tx_dv),
    .tx_hs(tx_hs),
    .tx_vs(tx_vs),
    .hdmi_tx_clk_p(hdmi_tx_clk_p),
    .hdmi_tx_clk_n(hdmi_tx_clk_n),
    .hdmi_tx_d0_p(hdmi_tx_d0_p),
    .hdmi_tx_d0_n(hdmi_tx_d0_n),
    .hdmi_tx_d1_p(hdmi_tx_d1_p),
    .hdmi_tx_d1_n(hdmi_tx_d1_n),
    .hdmi_tx_d2_p(hdmi_tx_d2_p),
    .hdmi_tx_d2_n(hdmi_tx_d2_n)
);
```

```
assign led_r = {pll_locked, 1'b0, rx_status};
```

```
endmodule
```

sobel_wrapper.v:

```
// this is a wrapper for sobel edge detection
// it uses dp_ram, edge_detector, line_buffer, and line_delay modules
// the wrapper receives 24 bit color image and produces 24 bit edge
image
// inside the wrapper the modules are created 3 times, for r, g, b
channels
module sobel_wrapper#(
    MAX_LINE_WIDTH = 2100
)(
    input         clk,
    input         rst,

    input [7:0]   red_i,
    input [7:0]   green_i,
    input [7:0]   blue_i,
    input         hsync,
    input         vsync,
    input         de,

    output [7:0] red_o,
    output [7:0] green_o,
    output [7:0] blue_o,
    output       hsync_out,
    output       vsync_out,
    output       de_out
);

wire [2:0] hsync_dl, vsync_dl, de_dl;
wire [7:0] pixel_out_r [9:1];
wire [7:0] pixel_out_g [9:1];
wire [7:0] pixel_out_b [9:1];

line_buffer#(
    .WIDTH(MAX_LINE_WIDTH)
) line_buffer_r(
    .clk         ( clk ),
    .rst         ( rst ),
    .pixel_in    ( red_i ),
    .hsync       ( hsync ),
    .vsync       ( vsync ),
    .de          ( de ),
    .pixel_out1  ( pixel_out_r[1] ),
    .pixel_out2  ( pixel_out_r[2] ),
    .pixel_out3  ( pixel_out_r[3] ),
    .pixel_out4  ( pixel_out_r[4] ),
    .pixel_out5  ( pixel_out_r[5]),
    .pixel_out6  ( pixel_out_r[6] ),
    .pixel_out7  ( pixel_out_r[7] ),
    .pixel_out8  ( pixel_out_r[8] ),
```

```
    .pixel_out9 ( pixel_out_r[9] ),
    .hsync_dl   ( hsync_dl[0] ),
    .vsync_dl   ( vsync_dl[0] ),
    .de_dl      ( de_dl[0] )

);

line_buffer#(
    .WIDTH(MAX_LINE_WIDTH)
) line_buffer_g(
    .clk(clk),
    .rst(rst),

    .pixel_in(green_i),
    .hsync(hsync),
    .vsync(vsync),
    .de(de),

    .pixel_out1(pixel_out_g[1]),
    .pixel_out2(pixel_out_g[2]),
    .pixel_out3(pixel_out_g[3]),
    .pixel_out4(pixel_out_g[4]),
    .pixel_out5(pixel_out_g[5]),
    .pixel_out6(pixel_out_g[6]),
    .pixel_out7(pixel_out_g[7]),
    .pixel_out8(pixel_out_g[8]),
    .pixel_out9(pixel_out_g[9]),

    .hsync_dl(hsync_dl[1]),
    .vsync_dl(vsync_dl[1]),
    .de_dl(de_dl[1])

);

line_buffer#(
    .WIDTH(MAX_LINE_WIDTH)
) line_buffer_b(
    .clk(clk),
    .rst(rst),

    .pixel_in(blue_i),
    .hsync(hsync),
    .vsync(vsync),
    .de(de),

    .pixel_out1(pixel_out_b[1]),
    .pixel_out2(pixel_out_b[2]),
    .pixel_out3(pixel_out_b[3]),
    .pixel_out4(pixel_out_b[4]),
```

```
    .pixel_out5(pixel_out_b[5]),
    .pixel_out6(pixel_out_b[6]),
    .pixel_out7(pixel_out_b[7]),
    .pixel_out8(pixel_out_b[8]),
    .pixel_out9(pixel_out_b[9]),

    .hsync_dl(hsync_dl[2]),
    .vsync_dl(vsync_dl[2]),
    .de_dl(de_dl[2])

);

wire edge_out_r, edge_out_g, edge_out_b;

edge_detector#(
    .THRESHOLD(100)
)edge_detector_r(
    .clk(clk),
    .rst(rst),
    .pixel_in1(pixel_out_r[1]),
    .pixel_in2(pixel_out_r[2]),
    .pixel_in3(pixel_out_r[3]),
    .pixel_in4(pixel_out_r[4]),
    .pixel_in5(pixel_out_r[5]),
    .pixel_in6(pixel_out_r[6]),
    .pixel_in7(pixel_out_r[7]),
    .pixel_in8(pixel_out_r[8]),
    .pixel_in9(pixel_out_r[9]),
    .hsync(hsync_dl[0]),
    .vsync(vsync_dl[0]),
    .de(de_dl[0]),

    .hsync_out(hsync_out),
    .vsync_out(vsync_out),
    .de_out(de_out),
    .edge_out(edge_out_r)
);

edge_detector#(
    .THRESHOLD(100)
)edge_detector_g(
    .clk(clk),
    .rst(rst),
    .pixel_in1(pixel_out_g[1]),
    .pixel_in2(pixel_out_g[2]),
    .pixel_in3(pixel_out_g[3]),
    .pixel_in4(pixel_out_g[4]),
    .pixel_in5(pixel_out_g[5]),
    .pixel_in6(pixel_out_g[6]),
```

```
    .pixel_in7(pixel_out_g[7]),
    .pixel_in8(pixel_out_g[8]),
    .pixel_in9(pixel_out_g[9]),
    .hsync(hsync_dl[1]),
    .vsync(vsync_dl[1]),
    .de(de_dl[1]),

    .hsync_out(),
    .vsync_out(),
    .de_out(),
    .edge_out(edge_out_g)
);

edge_detector#(
    .THRESHOLD(100)
)edge_detector_b(
    .clk(clk),
    .rst(rst),
    .pixel_in1(pixel_out_b[1]),
    .pixel_in2(pixel_out_b[2]),
    .pixel_in3(pixel_out_b[3]),
    .pixel_in4(pixel_out_b[4]),
    .pixel_in5(pixel_out_b[5]),
    .pixel_in6(pixel_out_b[6]),
    .pixel_in7(pixel_out_b[7]),
    .pixel_in8(pixel_out_b[8]),
    .pixel_in9(pixel_out_b[9]),
    .hsync(hsync_dl[2]),
    .vsync(vsync_dl[2]),
    .de(de_dl[2]),

    .hsync_out(),
    .vsync_out(),
    .de_out(),
    .edge_out(edge_out_b)
);

assign {blue_o, green_o, red_o} = {24{edge_out_b | edge_out_g |
edge_out_r}};

endmodule
```

line_buffer.v:

```
module line_buffer#(
    WIDTH = 2100
)(
    input         clk,
    input         rst,
```

```
    input  [7:0] pixel_in,
    input        hsync,
    input        vsync,
    input        de,

    output       hsync_dl,
    output       vsync_dl,
    output       de_dl,

    output [7:0] pixel_out1,
    output [7:0] pixel_out2,
    output [7:0] pixel_out3,
    output [7:0] pixel_out4,
    output [7:0] pixel_out5,
    output [7:0] pixel_out6,
    output [7:0] pixel_out7,
    output [7:0] pixel_out8,
    output [7:0] pixel_out9
);

// Delay video control signals to detect hsync rising edge
reg [1:0] hsync_dl1;
reg [1:0] vsync_dl1;
reg [1:0] de_dl1;

always @ (posedge clk) begin
    hsync_dl1 <= {hsync_dl1[1:0], hsync};
    vsync_dl1 <= {vsync_dl1[1:0], vsync};
    de_dl1    <= {de_dl1[1:0], de};
end

// Hsync rising edge is the reset signal for the line delay modules
wire hsync_rise;
assign hsync_rise = (~hsync_dl1[0] && hsync);

// Output pixel matrix
(* ram_style = "registers" *) reg [7:0] pixel_matrix [9:1];

wire [7:0] line1;
wire [7:0] line2;

integer i;
always @ (posedge clk) begin
    if (rst) begin
        for (i = 1; i < 9; i = i + 1)
            pixel_matrix[i] <= 0;
    end
    else begin
        pixel_matrix[9] <= de ? pixel_in : 8'h00;
```

```
        pixel_matrix[8] <= pixel_matrix[9];
        pixel_matrix[7] <= pixel_matrix[8];

        pixel_matrix[6] <= line1;
        pixel_matrix[5] <= pixel_matrix[6];
        pixel_matrix[4] <= pixel_matrix[5];

        pixel_matrix[3] <= line2;
        pixel_matrix[2] <= pixel_matrix[3];
        pixel_matrix[1] <= pixel_matrix[2];
    end
end

wire [7:0] line1_in;
assign line1_in = de ? pixel_in : 8'h00;

line_delay#(
    .LINE_WIDTH (WIDTH),
    .DATA_WIDTH (8)
)linedl_1(
    .clk                ( clk        ),
    .rst                ( hsync_rise ),
    .data_in            ( line1_in   ),
    .data_valid         ( 1'b1       ),
    .data_out           ( line1      )
);

line_delay#(
    .LINE_WIDTH (WIDTH),
    .DATA_WIDTH (8)
)linedl_2(
    .clk                ( clk        ),
    .rst                ( hsync_rise ),
    .data_in            ( line1      ),
    .data_valid         ( 1'b1       ),
    .data_out           ( line2      )
);

// Delay video control signals to align with output pixel
wire [2:0] video_control;
wire [2:0] video_control_dl;

assign video_control = {de_dl1[1], vsync_dl1[1], hsync_dl1[1]};

line_delay#(
    .LINE_WIDTH (WIDTH + 3),
    .DATA_WIDTH (3)
)videoctrl_dl(
    .clk                ( clk              ),
```

```
    .rst                ( hsync_rise       ),
    .data_in            ( video_control    ),
    .data_valid         ( 1'b1             ),
    .data_out           ( video_control_dl )
);

// Output assignment
assign pixel_out1 = pixel_matrix[1];
assign pixel_out2 = pixel_matrix[2];
assign pixel_out3 = pixel_matrix[3];
assign pixel_out4 = pixel_matrix[4];
assign pixel_out5 = pixel_matrix[5];
assign pixel_out6 = pixel_matrix[6];
assign pixel_out7 = pixel_matrix[7];
assign pixel_out8 = pixel_matrix[8];
assign pixel_out9 = pixel_matrix[9];

assign hsync_dl = video_control_dl[0];
assign vsync_dl = video_control_dl[1];
assign de_dl    = video_control_dl[2];

endmodule
```

line_delay.v:

```
module line_delay #(
    LINE_WIDTH = 1920,
    DATA_WIDTH = 8
)(
    input                       clk,
    input                       rst,

    input  [DATA_WIDTH - 1:0] data_in,
    input                       data_valid,

    output [DATA_WIDTH - 1:0] data_out
);

reg [$clog2(LINE_WIDTH) - 1:0] wr_ptr;
reg [$clog2(LINE_WIDTH) - 1:0] rd_ptr;

// write address construction
always @(posedge clk ) begin
    if (data_valid)
        wr_ptr <= rd_ptr;
end

// read address construction
always @(posedge clk) begin
    if (rst)
```

```
        rd_ptr <= 0;
    else if (data_valid)
        // if (rd_ptr == LINE_WIDTH - 1)
        //     rd_ptr <= 0;
        // else
            rd_ptr <= rd_ptr + 1'b1;
end

dp_ram #(
    .ADDR_WIDTH($clog2(LINE_WIDTH)),
    .DATA_WIDTH(DATA_WIDTH)
) u_dp_ram(
    .clk      ( clk        ),
    .we       ( data_valid ),
    .wr_addr  ( wr_ptr     ),
    .wr_data  ( data_in    ),
    .rd_addr  ( rd_ptr     ),
    .rd_data  ( data_out   )
);

endmodule
```

dp_ram.v:

```
module dp_ram#(
    ADDR_WIDTH = 10,
    DATA_WIDTH = 32
)(
    input                           clk,

    input                           we,
    input      [ADDR_WIDTH - 1 : 0] wr_addr,
    input      [DATA_WIDTH - 1 : 0] wr_data,
    input      [ADDR_WIDTH - 1 : 0] rd_addr,
    output reg [DATA_WIDTH - 1 : 0] rd_data
);

(* ram_style = "block" *)reg [DATA_WIDTH - 1 : 0] ram [2**ADDR_WIDTH -
1 : 0];
integer i;

initial begin
    for (i = 0; i < 2**ADDR_WIDTH; i = i + 1)
        ram[i] <= 0;
end

always @ (posedge clk) begin
    if (we)
        ram[wr_addr] <= wr_data;
    rd_data <= ram[rd_addr];
```

```
end

endmodule
```

edge_detector.v:

```
// edge_detector receives 9 input pixels from line_buffer and returns
if there is an edge or not
module edge_detector#(
    THRESHOLD = 100  // subject to change
)(
    input clk,
    input rst,
    input [7:0] pixel_in1,
    input [7:0] pixel_in2,
    input [7:0] pixel_in3,
    input [7:0] pixel_in4,
    input [7:0] pixel_in5,
    input [7:0] pixel_in6,
    input [7:0] pixel_in7,
    input [7:0] pixel_in8,
    input [7:0] pixel_in9,
    input hsync,
    input vsync,
    input de,
    output hsync_out,
    output vsync_out,
    output de_out,
    output reg edge_out
);

// vertical and horizontal sobel filter
reg signed [10:0] vertical_sobel_out, horizontal_sobel_out;
always @(posedge clk) begin
    if (rst) begin
        vertical_sobel_out <= 0;
        horizontal_sobel_out <= 0;
    end else begin
        vertical_sobel_out <= pixel_in1 + 2*pixel_in2 + pixel_in3 -
pixel_in7 - 2*pixel_in8 - pixel_in9;
        horizontal_sobel_out <= pixel_in1 + 2*pixel_in4 + pixel_in7 -
pixel_in3 - 2*pixel_in6 - pixel_in9;
    end
end

// if the absolute value of the vertical and horizontal sobel filter
is greater than the threshold, then there is an edge
always @(posedge clk) begin
```

```
    edge_out <= (vertical_sobel_out > THRESHOLD) || (horizontal_so-
bel_out > THRESHOLD) || (vertical_sobel_out < -1 * THRESHOLD) || (ho-
rizontal_sobel_out < -1 * THRESHOLD);
end

reg [1:0] hsync_dl = 0;
reg [1:0] vsync_dl = 0;
reg [1:0] de_dl = 0;

always @(posedge clk) begin
    if (rst) begin
        hsync_dl <= 0;
        vsync_dl <= 0;
        de_dl <= 0;
    end else begin
        hsync_dl[1:0] <= {hsync_dl[0], hsync};
        vsync_dl[1:0] <= {vsync_dl[0], vsync};
        de_dl[1:0] <= {de_dl[0], de};
    end
end

assign hsync_out = hsync_dl[1];
assign vsync_out = vsync_dl[1];
assign de_out = de_dl[1];

endmodule
```