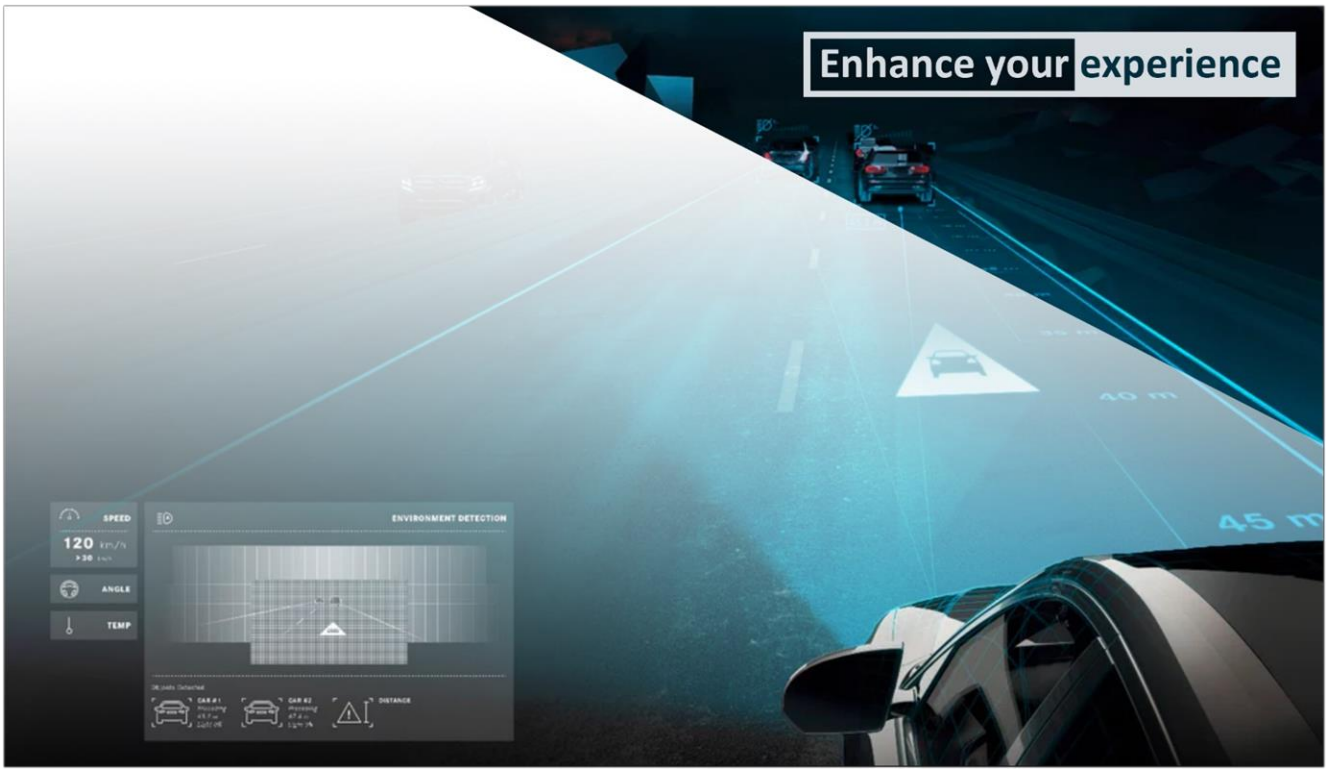
FPGA-based driving enhancement system

Sergiu Cuciurean
cuciurean.sergiu@yahoo.com

Submitted for the 2019 Digilent Design Contest Europe

5th of May 2019
Advisor: Albert Fazakas
Technical University of Cluj-Napoca
Applied Electronics Department
Cluj, Romania



CONTENTS

1. INTRODUCTION

1.1 ABSTRACT

This report presents the design of an embedded system that aims to improve the car driving experience while in dense fog conditions in order to have a safer journey.

1.2 OBJECTIVES

The main purpose of this project is to make a lane/car/sign detection in order to help the vehicle driver while going through very dense fog and reduced visibility.

In a scenario where a person is driving through fog, the eyesight is reduced by the fog's dense water droplets and also by the windshield refraction. In this case the driver should consider reducing the vehicle speed and turning on the headlights. This driving experience enhancement system can come as an additional safety measure that expands the road vision in order to avoid obstacles earlier.

The system should capture the driver's view, enhance the image and come back with a perspective compensation.

1.3 FEATURES-IN-BRIEF

- Video acquisition
- Edge detection
- Fog removal
- Video output
- Image projection

1.4 PROJECT SUMMARY

The system is composed of the following modules:

- Communication and configuration gateway
- Perspective acquisition module
- Data processing system
- Perspective compensation module

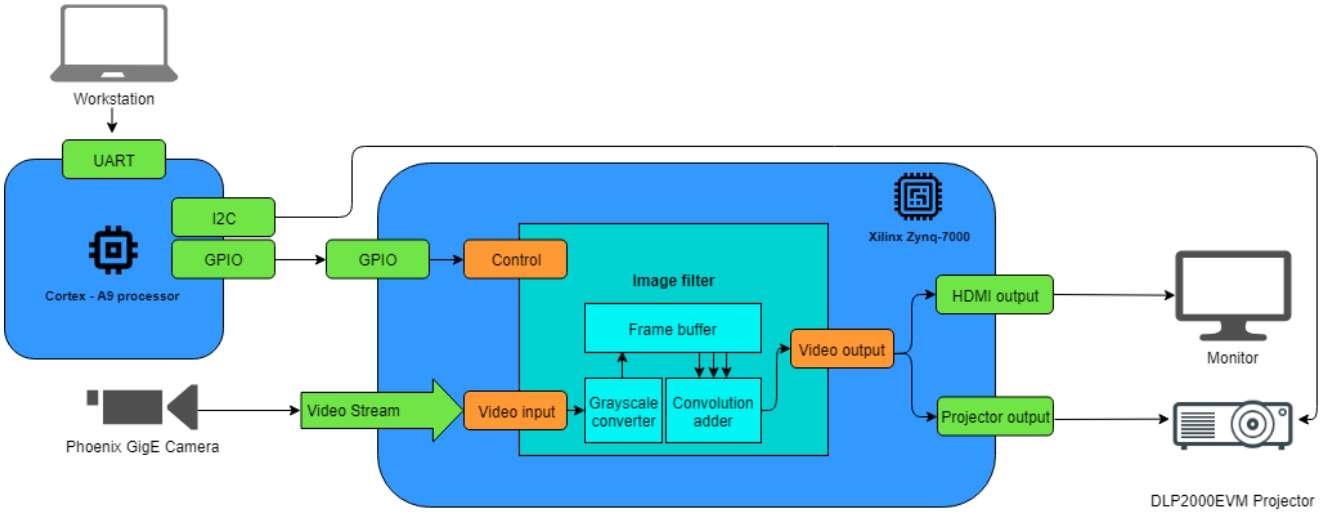


Fig . System architecture

1.6 DIGILENT PRODUCTS REQUIRED

- Zybo Z7-20

1.7 OTHER HARDWARE

- Thinklucid Phoenix GigE Camera
- PoE injector (if the network card doesn't support PoE)
- Ethernet CAT5 cable
- HDMI cable

1.8 TOOLS REQUIRED

- Vivado 2018.2
- Arena SDK

1.9 DESIGN STATUS

The polarized camera image acquisition should be done over Ethernet direclty to the Zybo board. Unfortunately, the SDK used with the polarized camera currently supports only 64-bit architecture and the ARM Cortex-A9 MPCore is a 32-bit processor core. As a replacement, the acquisition is done on a workstation and the frame is sent to the FPGA via HDMI.

Another dehazing filter used to remove the fog is under test. Right now the fog removal is done using only the camera's equalizer settings.

The pixel clock used by the projector has a frequency too high to be outputted on a regular GPIO pin. An alternative can be a HDMI projector.

The video loopback and eddge detection algorithm are working as expected.

2. BACKGROUND

2.1 WHY THIS PROJECT?

In some scenarios it can be hard to distinguish objects from their environments. Using polarized imaging, objects can be located by detecting the unique polarized angle of the reflected light from the object.

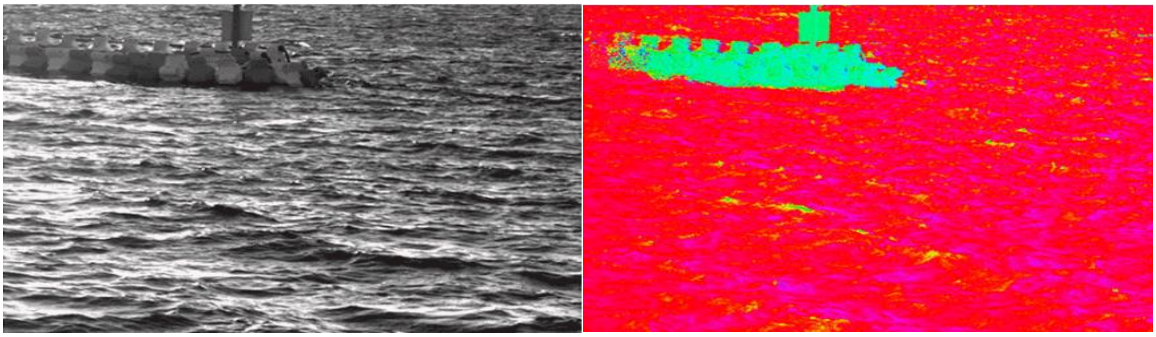


Fig . Object detection using polarized filter.

Researchers recorded a street scene viewed through a digital camera that records in grayscale (left), which is a little better than the cameras currently used in self-driving cars, and through two polarization cameras. Those two views look slightly different because one shows the amount of light polarization (middle) and the other shows the angle of polarization (right). In the middle panel, red hues indicate high levels of polarized light and blue hues reflect low levels. In the panel on the right, red hues indicate horizontally polarized light and blue hues show vertically polarized light. Images from both polarization cameras pick up a higher level of detail than the grayscale camera, bringing other cars, pedestrians and bicyclists more clearly into view.

Fig . Foggy street views

2.2 REFERENCE MATERIAL

2.2.1 IMAGE FILTERING

In image processing, image filtering involves applying a mathematical operation called convolution to each pixel that image. Similar to filtering images, filtering video sequences involves filtering each frame. Depending on the filter application, it may be spatial (if it depends only on the current frame) or space-time (if it depends on other frames).

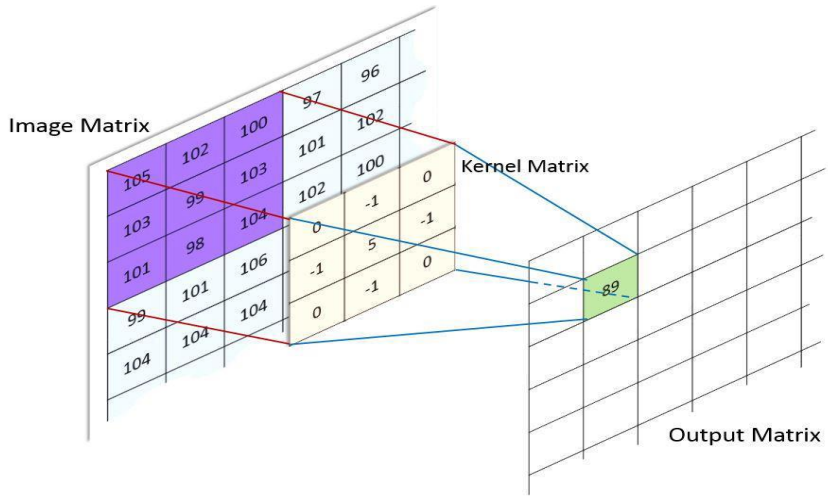


Fig . Convolution illustration

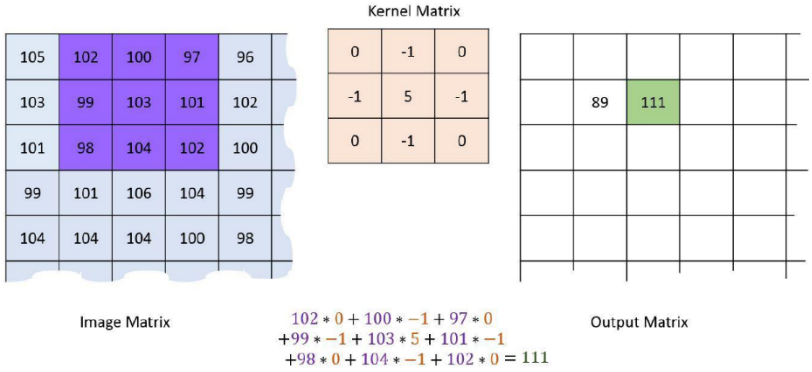


Fig . Convolution example

One type of spatial filtering is edge detection. This assumes that the value of each pixel in the output image is given by the formula:

$$\left(\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} \circ \begin{vmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ z_0 & z_1 & z_2 \end{vmatrix} \right) = (a * x_0) + (b * x_1) + (c * x_2) + (d * y_0) + (e * y_1) + (f * y_2) + (g * z_0) + (h * z_1) + (i * z_2)$$

Where :

- $\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$ is the convolution matrix

- $\begin{vmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ z_0 & z_1 & z_2 \end{vmatrix}$ is the current pixel ($y1$) and it's neighbors

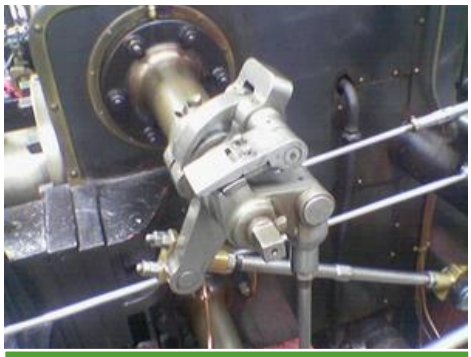
The sobel operator assumes the calculation of two gradients (one vertically and one horizontal) using two different convolution matrices. The matrix size is 3x3 pixels and are as follows:

$$G_V = \begin{vmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{vmatrix} \quad G_H = \begin{vmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix}$$

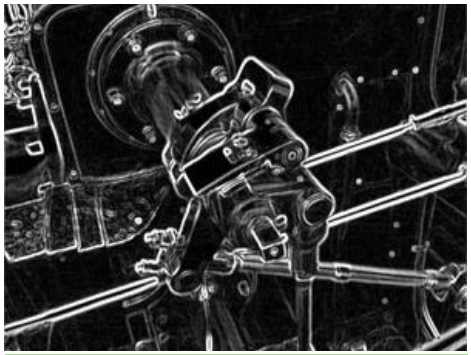
Using these matrices, the differences between neighbor pixel values are enhanced, resulting in the contour of the image content. For a higher contour detail, the following parameters can be calculated:

- Magnitude $M = \sqrt{G_V{}^2 + G_H{}^2}$
- Direction $D = \arctan\frac{G_V}{G_H}$

The mathematical operations are ilustrated in the following figure :



Original Image



Filtered Image

Fig . Filtering result

2.2.2 IMAGE POLARIZATION

When light hits a linear polarizer, such as the vertical and horizontal polarizers in the example below, the angled vibrations are filtered out with only vertical or horizontal vibrations passing through.

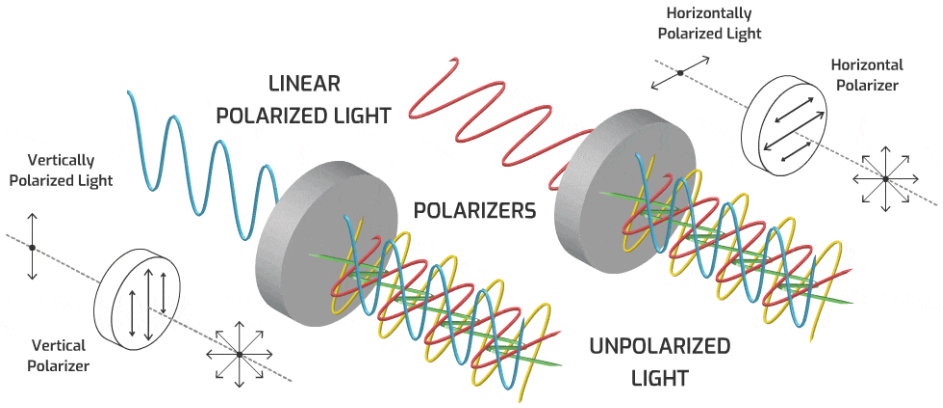


Fig. Example of Vertical and Horizontal Polarizers

Given that a water droplet can reflect the light, there is a chance that one out of the four polarized light waves can pierce the droplet. Therefore combining the four polarized images, the resulting one will be a slightly defogged one.

2.3 DESIGN

2.3.1 FEATURES AND SPECIFICATIONS

In the following figure are presented the four main components :

- Video in
- Video out
- Processor
- Image processing system

2.4 DESIGN OVERVIEW

The video in block takes the HDMI input and converts it in RGB value and sync signals. The RGB pixel gets passed to the image processing system and the output is and grayscale high contrast pixel. The video out block is used to take the outputted pixel with the sync signals and convert them back to HDMI.

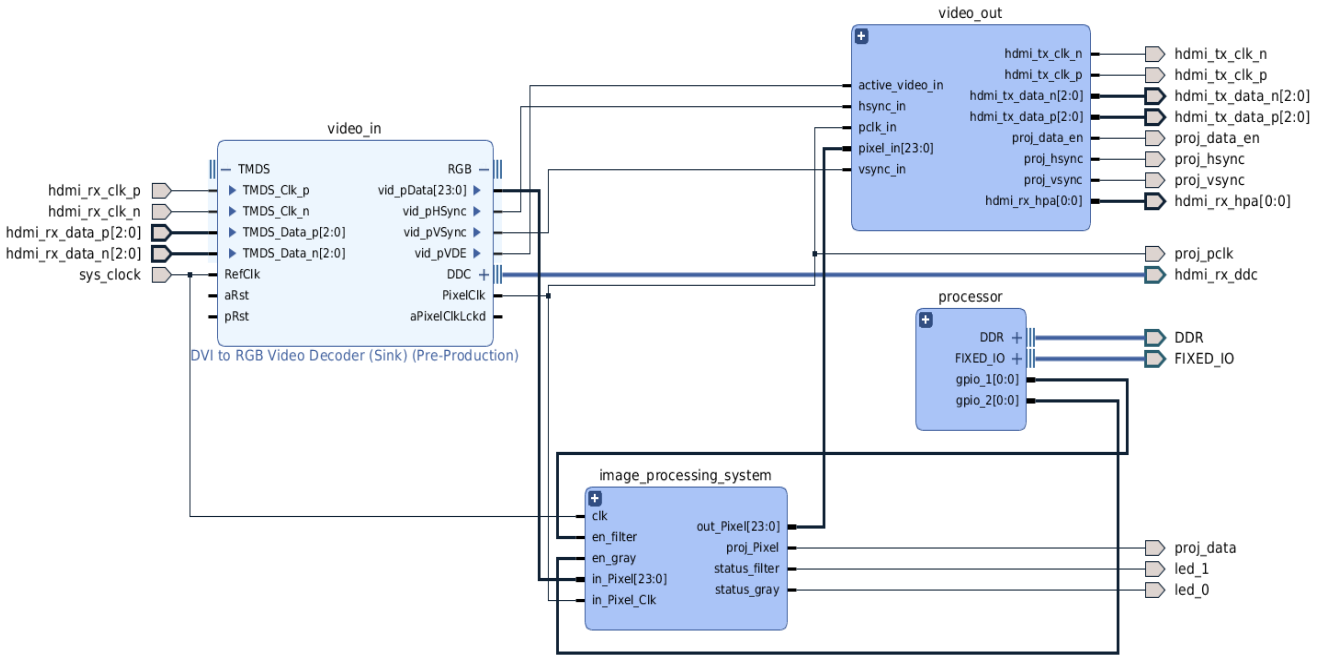


Fig. Block design

2.5 DETAILED DESIGN DESCRIPTION

The ZYNQ processing system is used to toggle the grayscale and edge filter over GPIO's and to configure the projector's behaviour over I2C.

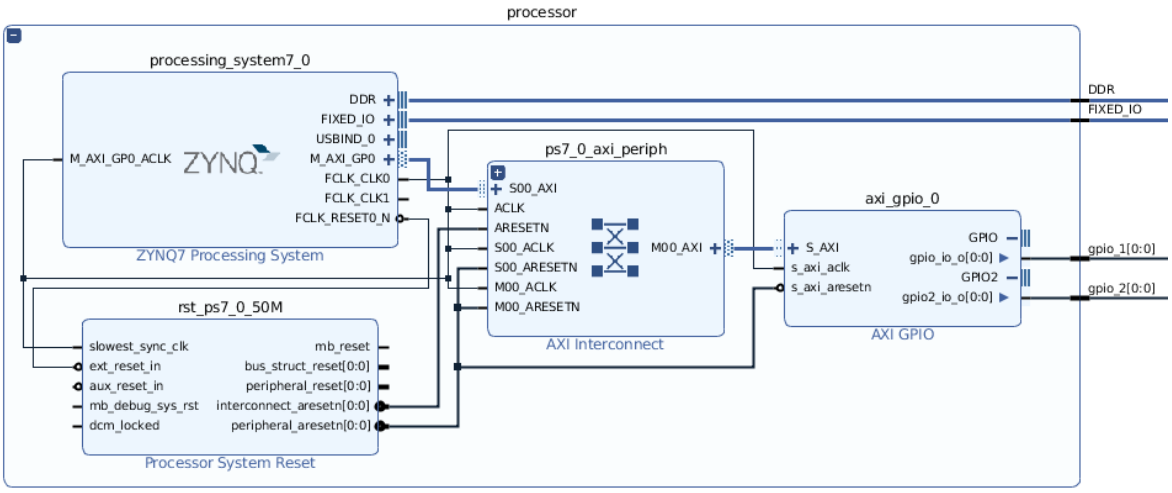


Fig. ZYNQ7 system

The image processing systems takes as input the RGB pixel, pixel clock, two GPIO inputs and one system clock. The two GPIOS are used to enable or disable the video filters (grayscale and edge). The architecture contains three FIFO buffers used to get the first three rows of pixels, a grayscale filter and edge detection filter. Nine pixels are passed from the frame buffers to the image filter as a 3x3 matrix later to be used in a convolution operation. The calculated RGB value is then further outputted alongside the projector pixel and two status flags.

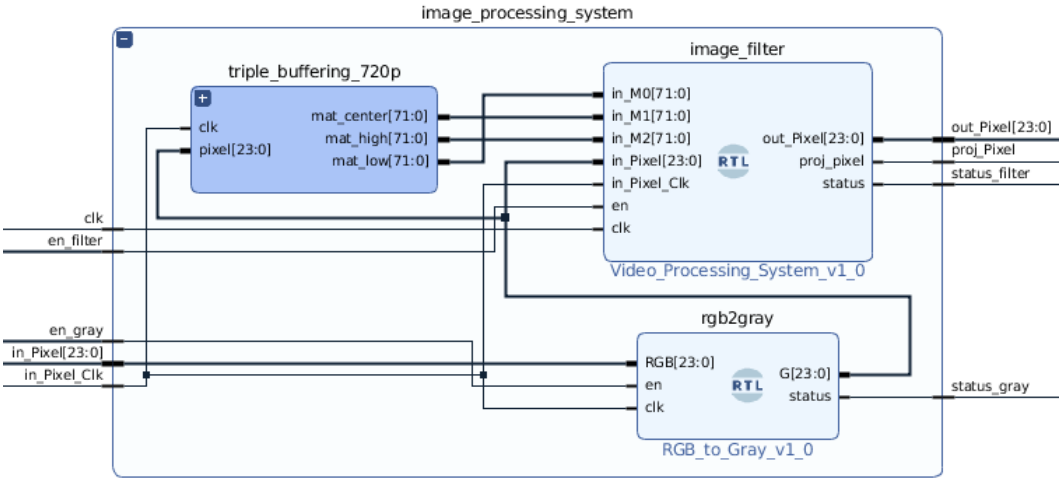


Fig. Image processing system

The input pixel and sync signals are distributed to a HDMI converter and a timing controller. The HDMI converter outputs the video to an external source while the timing controller downscales the video to the projector's resolution.

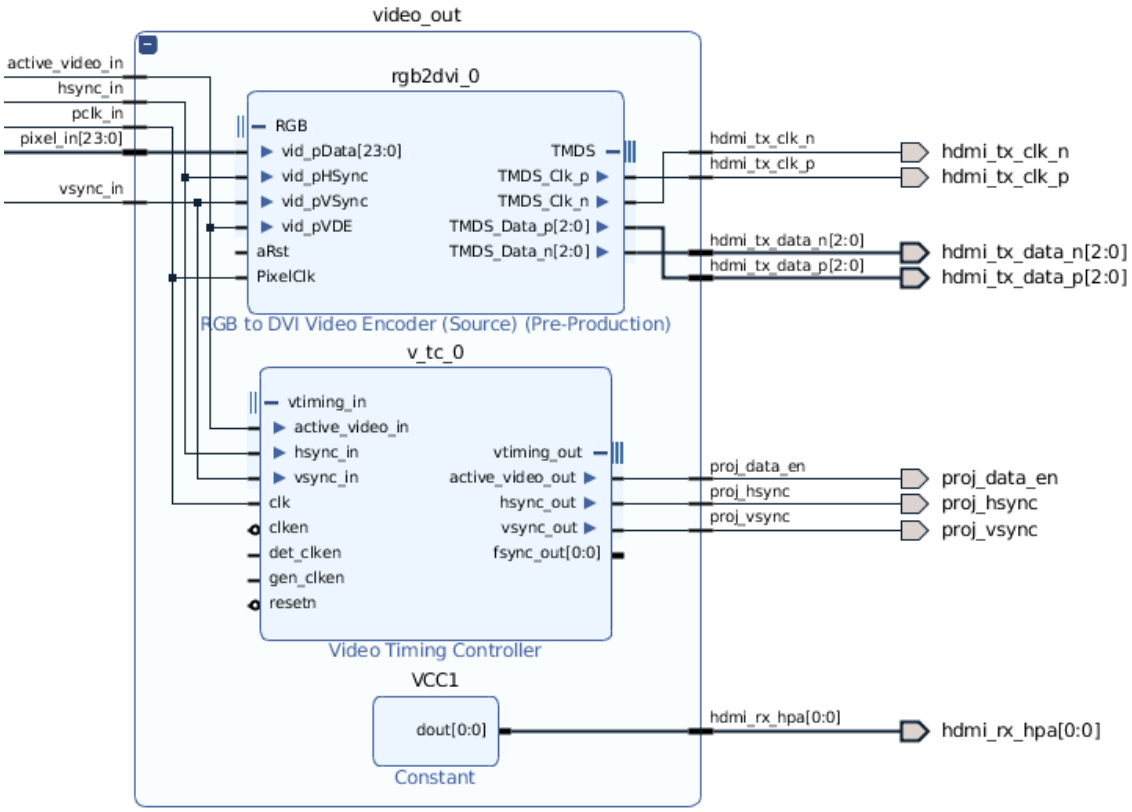


Fig. Video output

3. DISCUSSION

3.1 PROBLEMS ENCOUNTERED

There were some compatibility problems while using the example projects provided by Digilent.
Some HDMI monitors may not recognise some specific video timings and this was a big mislead for me.

3.2 ENGINEERING RESOURCES USED

This project was a work of several weeks with a significant amount of hours spent on preparing the demo environment.

3.3 MARKETABILITY

The concept of having a day time projector embedded in a car headlamp is already present in some high-end cars but with another purpose. As the automotive industry is constantly evolving and in search for innovation, a project of such kind may be, by suprise, accepted as a different approach. It could definitely be an alternative for the well-known heads-up displays just by the fact that the front-view image projections gives the driver a better perspective than a curved image on the windshield.

3.4 COMMUNITY FEEDBACK

The project was presented to persons with interests in the same topic and a positive feedback has been received.

4. REFERENCES

[1] http://ares.utcluj.ro/tpi_2018.html

[2] https://lauri.võsandi.com

[3] https://e2e.ti.com/

[4] TI DLP® LightCrafter™ Display 2000 EVM User's Guide

[5] DLPC2607 SoftwareProgrammer's Guide

[6] DLP2000 (.2nHD) DMD

[7] DLPC2607 Low-power DLP® Display Controller

[8] TI DLP® System Design: Brightness Requirements and Tradeoffs

[9] Introduction to graphics and LCD technologies NXP Product Line Microcontrollers

[10] EECS150 - Digital Design Lecture 14 - Project Description, Part 3

[11] UltraFast Design Methodology Guide for the Vivado Design Suite

[12] Vivado Design Suite Tutorial Using Constraints

[13] Zybo z7 schematic

5. APPENDIX A: RGB_TO_GRAY

```
`timescale
1ns / 1ps
            //////////////////////////////////////////////////////////////////////////////////
            // Company:
            // Engineer:
            //
            // Create Date: 04/07/2019 07:43:22 AM
            // Design Name:
            // Module Name: RGB_to_Gray
            // Project Name:
            // Target Devices:
            // Tool Versions:
            // Description: This module is used to convert a 24-bit RGB pixel into grayscale
            //
            // Dependencies:
            //
            // Revision:
            // Revision 0.01 - File Created
            // Additional Comments:
            //
            //////////////////////////////////////////////////////////////////////////////////


            module RGB_to_Gray(
                input [23:0] RGB,
                input en,
                input clk,
                output [23:0] G,
                output status
                );

                reg  [23:0] tempPixel;
                reg  [23:0] rgb2gray;
                reg  [23:0] grayPixel;
                wire [7:0]  BLUE;
                wire [7:0]  GREEN;
                wire [7:0]  RED;
```

```
    // Output the current status (enalbed/disabled)
    assign status   = en;
    // Assign a register to the output
    assign G        = rgb2gray;
    // Split the input in RGB values
    assign BLUE     = tempPixel[7:0];
    assign GREEN    = tempPixel[15:8];
    assign RED      = tempPixel[23:16];

always @ (posedge clk)
begin
    // If the module is enabled output grayscale pixel
    if(en) begin
        // Store the pixel value
        tempPixel = RGB;
        // Convert to grayscale
        grayPixel = ((RED * 76) + (GREEN * 151) +(BLUE * 28));
        // Place the gray value on all 3 components
        rgb2gray[23:16] = grayPixel[15:8];
        rgb2gray[15:8] = grayPixel[15:8];
        rgb2gray[7:0] = grayPixel[15:8];
    end else begin
        // If the module is disabled just pass the input pixel
        rgb2gray = RGB;
    end
end
endmodule
```

6. APPENDIX B: SHIFT_REGISTER

```
`timescale
1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/07/2019 08:24:11 AM
// Design Name:
// Module Name: Shift_register
// Project Name:
// Target Devices:
// Tool Versions:
// Description: This module is used store one line made out of 3 pixels used in the
//              convolution operation. After a pixel clock, the value gets shifted
//              with one pixel and the new pixel is added to the line.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Shift_register(
    input [23:0] pixel,
    input clk,
    input en,
    output [71:0] line
    );

    // Aux register
    reg [71:0] resultLine;
    // Assign the register to the output
    assign line  = resultLine;
```

```
    // Shift the value if enabled
    always @(posedge clk & en)
    begin
        // 9-byte shift register with added value on last 3 bytes
        resultLine = ((resultLine << 24) | pixel);
    end
    endmodule
```

7. APPENDIX C: VIDEO_PROCESSING_SYSTEM

```
`timescale
1ns / 1ps
    //////////////////////////////////////////////////////////////////////////////////
    // Company:
    // Engineer:
    //
    // Create Date: 04/07/2019 06:06:05 AM
    // Design Name:
    // Module Name: Video_Processing_System
    // Project Name:
    // Target Devices:
    // Tool Versions:
    // Description: This module is used to apply an image filter on the received frame.
    //              The image comes in as 3 frame buffers and an mathematical operation
    //              is applied on 3 pixels from each row (top, center, bottom). The
    //              resulted value is then outputed to the next component.
    //
    // Dependencies:
    //
    // Revision:
    // Revision 0.01 - File Created
    // Additional Comments:
    //
    //////////////////////////////////////////////////////////////////////////////////


    module Video_Processing_System(
```

```
    input [71:0]in_M0,
    input [71:0]in_M1,
    input [71:0]in_M2,
    input [23:0] in_Pixel,
    input in_Pixel_Clk,
    input en,
    input clk,
    output [23:0] out_Pixel,
    output proj_pixel,
    output status
    );

// Register used to output 1 bit pixel for the projector
reg  value;
// Pixel variable
reg  [23:0] resultPixel;
// Variable used for convolution operation
reg  [7:0] conv;
// 9 8-bit variables used as convolution pixel neighbours
wire [7:0] p0;
wire [7:0] p1;
wire [7:0] p2;
wire [7:0] p3;
wire [7:0] p4;
wire [7:0] p5;
wire [7:0] p6;
wire [7:0] p7;
wire [7:0] p8;
// Used for vertical and horizontal gradient
wire signed [10:0] gx,gy;
// Used to store the absolute value of the gradients
wire signed [10:0] abs_gx,abs_gy;
// Used in gratient operation
wire [10:0] sum;

// Output the current status (enalbed/disabled)
assign status        = en;
// Assign the output pixel
```

```
assign proj_pixel = value;
// Split the 3-byte matrix input in 1-byte pixels
assign p0 = in_M0[7:0];
assign p1 = in_M0[31:24];
assign p2 = in_M0[55:48];
assign p3 = in_M1[7:0];
assign p4 = in_M1[31:24];
assign p5 = in_M1[55:48];
assign p6 = in_M2[7:0];
assign p7 = in_M2[31:24];
assign p8 = in_M2[55:48];
// Calculate the gradients
assign gx=((p2-p0)+((p5-p3)<<1)+(p8-p6));
assign gy=((p0-p6)+((p1-p7)<<1)+(p2-p8));
// Extract the absolute values
assign abs_gx = (gx[10]? ~gx+1 : gx);
assign abs_gy = (gy[10]? ~gy+1 : gy);
// Add them up
assign sum = (abs_gx+abs_gy);
// Assign the output used for hdmi
assign out_Pixel = resultPixel;

always @ (posedge clk) begin
    // If the module is disabled just pass the pixel
    if(en == 0) begin
        resultPixel = in_Pixel;
    end else begin
        // If not truncate the convolution value
        conv = (|sum[10:8])?8'hff : sum[7:0];
        // Assign the value to the output pixel
        resultPixel[7:0] = conv;
        resultPixel[15:8] = conv;
        resultPixel[23:16] = conv;
        // Apply a treshold to the projector pixel output
        if(conv > 60) begin
            value = 1;
        end else begin
            value = 0;
```

```
            end
        end
    end

    endmodule
```