Real-time video processing

Lab Section 6: Aizhuldyz Nadirkhanova (aizhuldyz.nadirkhanova@nu.edu.kz) **, Anuar Karzhaubayev** (anuar.karzhaubayev@nu.edu.kz**), Aidana Kalimbekova** (aidana.kalimbekova@nu.edu.kz)

1. Introduction

The aim of the project was to design the real-time video processing on an FPGA and efficiently process the pixel data and quickly show the output. Real-time image processing is difficult to achieve on a serial processor. This is due to several factors such as the large data set represented by the image, and the complex operations which may need to be performed on the image. At real-time video rates of 25 frames per second a single operation performed on every pixel of a 768 by 576 color image (NSTC frame) equates to 33 million operations per second. This does not take into account the overhead of storing and retrieving pixel values. Many image processing applications require that several operations be performed on each pixel in the image resulting in an even large number of operations per second.

2. Materials and Methods

a. Altera Cyclone® V SE 5CSEMA5F31C6N device (DE1_SoC board)
b. USB-Blaster II onboard for programming; JTAG Mode
c. NTSC Camera
d. Power battery 9V
e. RCA cable, VGA cable

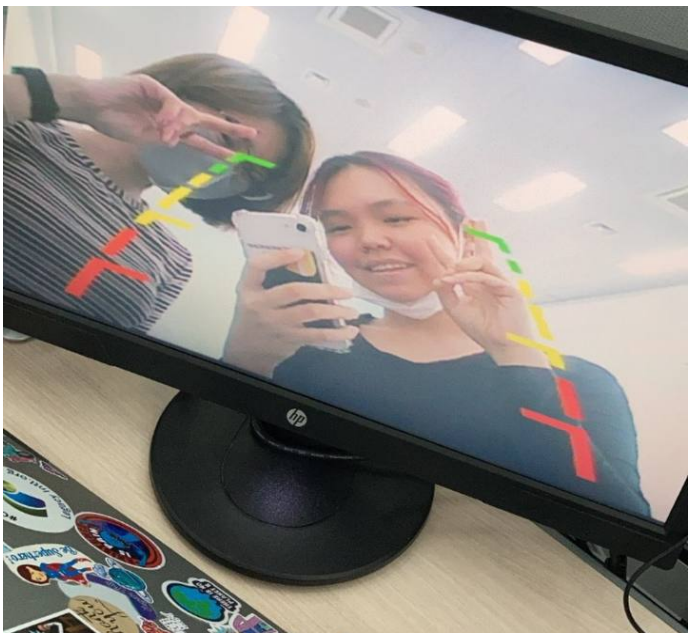
3. Results

We wanted a gesture control which will switch the music when the camera will see magenta color in the certain active areas of the display.

However, it ended up being an advanced Computer Vision project requiring HPS (High Performance System) which will compile C++/Matlab code for processing. Although, FPGA Cyclone V have HPS built in it, we still did not find any manual for implementation (how compile back-end in Quartus II, while sending data to HPS, saving to SDRAM memory and display through VGA). For this reason we ended up working with RGB colors and edge detection on the real time picture transmitted by camera.

At first we thought we were going to use just any camera(ex. from laptop). However, we realized that it needs an NTSC camera, which as we google turns out to be an auto camera for line detection.

We found a supplier who sells these budget oriented cameras, but our problem was also to get rid of colorful lines on display. So we opened the chip and accidentally broke our first camera. Thus, we had to buy a new one and luckily got another version where the parking wires were highlighted for us.



Picture 1. First try with NTSC camera transmitting image to VGA display

After we found the appropriate camera to transmit the data, we faced a problem that not many resources were available on the Internet. So we found some useful ones: DE1-Soc-User Manual(page 71-72) from TerasIC Altera University and Cornell University Archive for ECE5760 from Professor Bruce R Land. When we first found information regarding the NTSC camera it was dedicated either to Cyclone II or Cyclone IV(not V, which we needed). Thus, we modified our version to transmit the real-time information.

Picture 2. Final Result of Grayscale VGA display

Main module (DE1_SoC_TV):

In the Main Module such modules as I2C Multiplexer, VGA, HPS, SW, TD, I/O are declared at first. Then wires for VGA Controller, ITU-R 656, YUV 4:2:2 to YUV 4:4:4, field select, and NTSC are declared. All inout ports are turned into tri-state and TV Decoder is enabled to work with VGA data requests. For VGA we wire mRed (the data we receive from TV decoder to FPGA), mux_Red(the

data we processed and output) and similarly for Green and Blue colors. We initialize the buffer3 which give us 3X3 grid for each pixels we calculate the the intensity and assign them to R, G, and B canals for getting Sobel filter., Also, here we decide if there is an edge, which we see the edge, we output the black or sobel filtered values for RGB values.

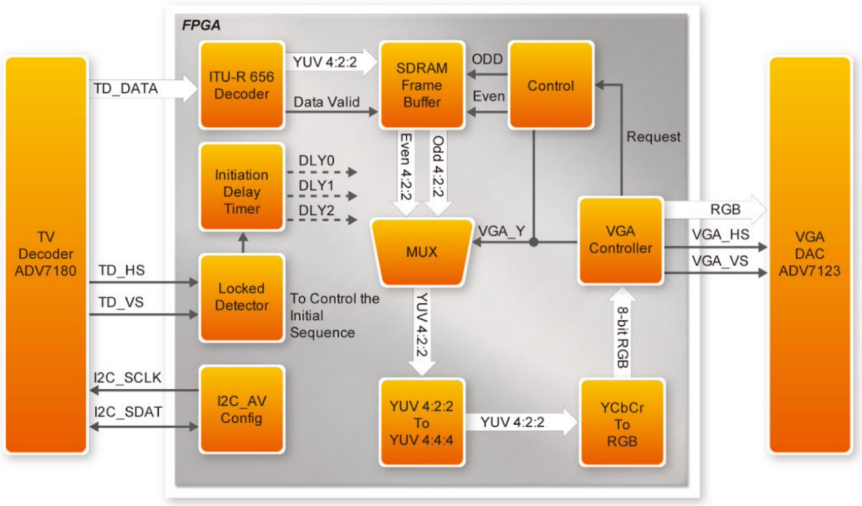


Figure 5-9 Block diagram of the TV box demonstration

TV box demonstration:

The video input is produced through a special NTSC camera, which delivers 525 scan lines, 30 frames per second with 60-Hz grid frequency. Using such a camera made it easier in transmitting the already needed format(NTSC standard video format) in real time. It was connected via RCA output, where it is delivered to TV Decoder(ADV7180).

The I2C_AV_Config block uses the I2C protocol and configures the TV decoder register values. From TV Decoder, the ITU-R 656 Decoder block gets the YcrCb 4:2:2(YUV 4:2:2) video signals, which were sent firstly to its data stream. Since the TV decoder is interlaced, we need to deinterlace it by SDRAM Frame Buffer and a field selection multiplexer(MUX), which is controlled by VGA_Ctrl. This VGA Controller computes odd/even selection signals and data requests to them. Then, the FPGA converts YcrCb 4:2:2(YUV 4:2:2) into YcrCb 4:4:4(YUV 4:4:4) video data format.

In the final step, the YcrCb_to_RGB block converts the YcrCb data into RGB data output. The VGA Controller block generates standard VGA synchronous signals VGA_HS and VGA_VS to enable the display on a VGA monitor.

Programming of FPGA:

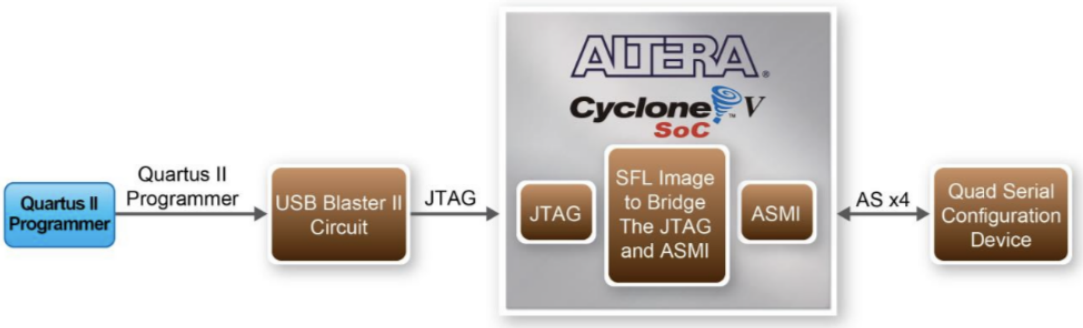


Figure 3-9 Programming a quad serial configuration device with SFL solution

Vga Controller:

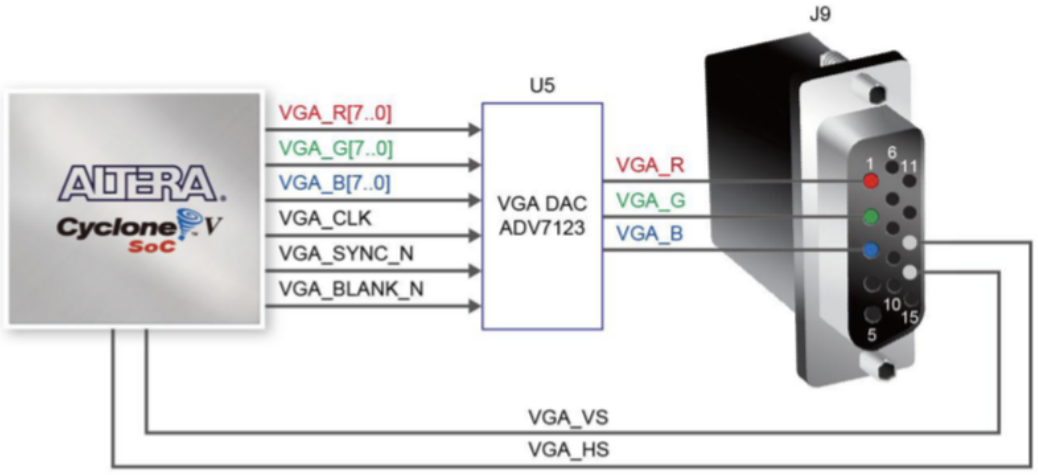


Figure 3-22 Connections between the FPGA and VGA

The board has a 15-pin connector populated for VGA output. Synchronized VGA signals are generated from the board and converted into analog video signals by video DAC so that it represents the RGB colors. In figure 3-22, connections between FPGA and VGA are depicted. We have horizontal synchronization (hsync) input of the monitor, which is rows and vertical, which is columns. When the hsync pulse occurs, RGB color signals will be off and the next row will be displayed as it is shown in the figure 3-23 VGA horizontal timing specification.

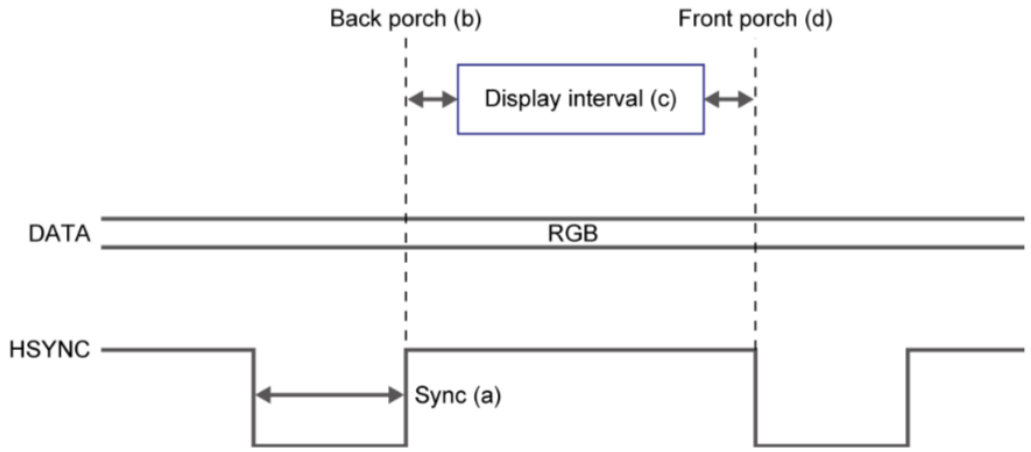


Figure 3-23 VGA horizontal timing specification

Vsync pulse occurs at the end of each frame. So each time vertical sync ends, it means the time required for refreshing the entire screen. In figure 3-16 you can see required pins for synchronization of VGA, which we used in VGA_ctrl.v.

Table 3-16 Pin Assignment of VGA

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
VGA_R[0]	PIN_A13	VGA Red[0]	3.3V
VGA_R[1]	PIN_C13	VGA Red[1]	3.3V
VGA_R[2]	PIN_E13	VGA Red[2]	3.3V
VGA_R[3]	PIN_B12	VGA Red[3]	3.3V
VGA_R[4]	PIN_C12	VGA Red[4]	3.3V
VGA_R[5]	PIN_D12	VGA Red[5]	3.3V
VGA_R[6]	PIN_E12	VGA Red[6]	3.3V
VGA_R[7]	PIN_F13	VGA Red[7]	3.3V
VGA_G[0]	PIN_J9	VGA Green[0]	3.3V
VGA_G[1]	PIN_J10	VGA Green[1]	3.3V
VGA_G[2]	PIN_H12	VGA Green[2]	3.3V
VGA_G[3]	PIN_G10	VGA Green[3]	3.3V
VGA_G[4]	PIN_G11	VGA Green[4]	3.3V
VGA_G[5]	PIN_G12	VGA Green[5]	3.3V
VGA_G[6]	PIN_F11	VGA Green[6]	3.3V
VGA_G[7]	PIN_E11	VGA Green[7]	3.3V
VGA_B[0]	PIN_B13	VGA Blue[0]	3.3V
VGA_B[1]	PIN_G13	VGA Blue[1]	3.3V
VGA_B[2]	PIN_H13	VGA Blue[2]	3.3V
VGA_B[3]	PIN_F14	VGA Blue[3]	3.3V
VGA_B[4]	PIN_H14	VGA Blue[4]	3.3V
VGA_B[5]	PIN_F15	VGA Blue[5]	3.3V
VGA_B[6]	PIN_G15	VGA Blue[6]	3.3V
VGA_B[7]	PIN_J14	VGA Blue[7]	3.3V
VGA_CLK	PIN_A11	VGA Clock	3.3V
VGA_BLANK_N	PIN_F10	VGA BLANK	3.3V
VGA_HS	PIN_B11	VGA H_SYNC	3.3V
VGA_VS	PIN_D11	VGA V_SYNC	3.3V
VGA_SYNC_N	PIN_C10	VGA SYNC	3.3V

Vga_sync generates synchronization of signals. A pixel rate (27 MHz clock) means 27 million pixels are processed in one cycle. When there is no video signal, but the hsync and vsync occurs, no frame will be displayed. A VGA controller and TV system modules should be synchronized and thus timings should be accurate in order for data to be displayed and also correct data to be written into memory for processing. Therefore, we receive the camera input data in YCrCb format and send it to

the SDRAM frame buffer, after which the VGA controller requests data and selects a signal it got from the buffer.

I2C Multiplexer:

The I2C Multiplexer of the FPGA gives access to the I2C bus for HPS. The pin assignments of the I2C bus are listed in the table below. Also, in the figure 3-21 the control mechanism for the I2C Multiplexer.

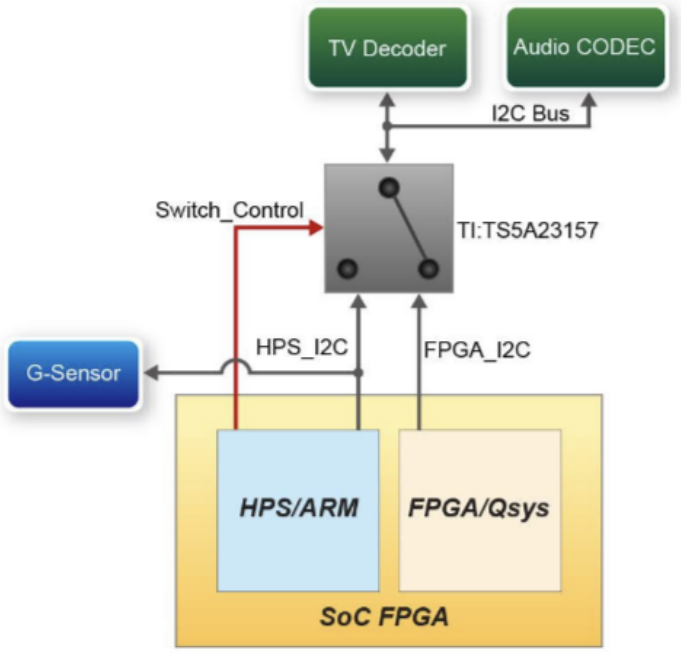


Figure 3-21 Control mechanism for the I2C multiplexer

Table 3-13 Pin Assignment of I2C Bus

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
FPGA_I2C_SCLK	PIN_J12	FPGA I2C Clock	3.3V
FPGA_I2C_SDAT	PIN_K12	FPGA I2C Data	3.3V
HPS_I2C1_SCLK	PIN_E23	I2C Clock of the first HPS I2C concontroller	3.3V
HPS_I2C1_SDAT	PIN_C24	I2C Data of the first HPS I2C concontroller	3.3V
HPS_I2C2_SCLK	PIN_H23	I2C Clock of the second HPS I2C concontroller	3.3V
HPS_I2C2_SDAT	PIN_A25	I2C Data of the second HPS I2C concontroller	3.3V

TV Decoder(RCA):

The DE1-SoC board is equipped with an Analog Device ADV7180 TV decoder chip. The ADV7180 is an integrated video decoder which automatically detects and converts a standard analog baseband television signals (NTSC, PAL, and SECAM) into 4:2:2 component video data, which is compatible with the 8-bit ITU-R BT.656 interface standard. The ADV7180 is compatible with a wide range of video devices, including DVD players, tape-based sources, broadcast sources, and security/surveillance cameras.

The registers in the TV decoder can be accessed and set through serial I2C bus by the Cyclone

V SoC FPGA or HPS. Note that the I2C address W/R of the TV decoder (U4) is 0x40/0x41. The pin assignment of the TV decoder is listed in Table 3-17. More information about the ADV7180 is available on the manufacturer's website, or in the directory \DE1_SOC_datasheets\Video Decoder of DE1-SoC System CD.

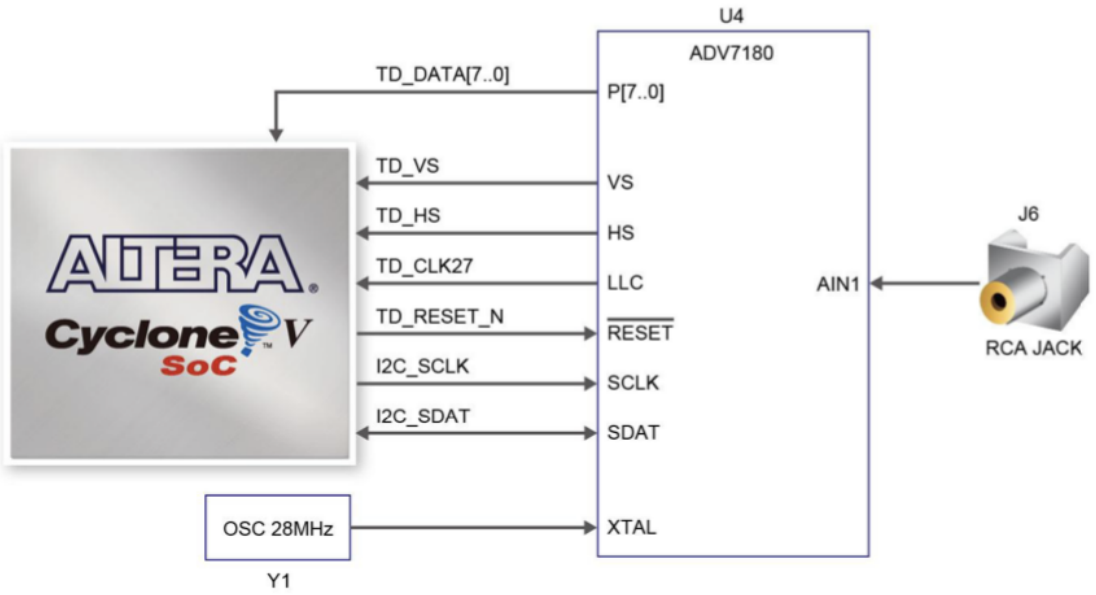


Figure 3-24 Connections between the FPGA and TV Decoder

Table 3-17 Pin Assignment of TV Decoder

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
TD_DATA [0]	PIN_D2	TV Decoder Data[0]	3.3V
TD_DATA [1]	PIN_B1	TV Decoder Data[1]	3.3V
TD_DATA [2]	PIN_E2	TV Decoder Data[2]	3.3V
TD_DATA [3]	PIN_B2	TV Decoder Data[3]	3.3V
TD_DATA [4]	PIN_D1	TV Decoder Data[4]	3.3V
TD_DATA [5]	PIN_E1	TV Decoder Data[5]	3.3V
TD_DATA [6]	PIN_C2	TV Decoder Data[6]	3.3V
TD_DATA [7]	PIN_B3	TV Decoder Data[7]	3.3V
TD_HS	PIN_A5	TV Decoder H_SYNC	3.3V
TD_VS	PIN_A3	TV Decoder V_SYNC	3.3V
TD_CLK27	PIN_H15	TV Decoder Clock Input.	3.3V
TD_RESET_N	PIN_F6	TV Decoder Reset	3.3V
I2C_SCLK	PIN_J12 or PIN_E23	I2C Clock	3.3V
I2C_SDAT	PIN_K12 or PIN_C24	I2C Data	3.3V

3 line buffer line:

The information we got from Terasic Altera University for video streaming. But for processing pixels, we cannot request for 27 million pixels at one cycle, so it will be better to receive pixels to process in buffers. We can get 3-line buffers, which are large registers with pixel information, which is one line in the VGA screen. A line consists of 640 pixels and each pixel has 30 bits (10 bits for Red, 10 for Blue, 10 for Green). On clock edge, shiftin input will be stored on the first pixel of line 1[0], while the rest of pixels will be shifted up to one, and we output line 2[638] as shiftout. Then this pixel

data will be used to output to the screen. Moreover, for edge detection filters, we need neighbors of each pixel which will be grouped. The center of the grid will be line 2[638], so calculations for the filter will be done to this pixel so when we output the result, this pixel will be outputted. If there is an edge, the pixel will be turned into black. These registers then will be memory blocks which is done by a compiler.

Intensity Calculation:

Implementation of weighted averaging between color channels of the pixel is intensity calculator module. For our intensity calculation, we use 25% red, 50% green, and 25% blue. We have chosen these numbers because the human eye is most sensitive to green. The coefficients of 25% and 50% were used in order to use shift and add operations in order to reach optimization, instead of costly multiplication.

Grayscale:

```
wire    [29:0] Gray;

assign Gray = {Intensity3,Intensity3,Intensity3};

assign mux_Red   = (is_edge) ? (10'd0) : Gray[29:20];
assign mux_Green = (is_edge) ? (10'd0) : Gray[19:10];
assign mux_Blue  = (is_edge) ? (10'd0) : Gray[9:0];
```

We assigned Gray wire with 30 bits, and sent to its 30 bits, 3 degrees of intensity that was found previously in the edge detection module. After that, we send its values by 10 bits each into mux_Red, mux_Green, mux_Blue, with which we get our picture in gray colors.

Edge detection through Sober Filter:

Edge Detection is computed by finding the direction of the largest increase from light to dark and the rate of change in that direction. For this project, the edge detection algorithms, the two Sobel filters are used: EdgedetectH.v and EdgedetectV.v. They are used to generate the filter for both Horizontal(right) and Vertical edges, which are used for the entire pixel image demonstrated on the VGA display. The Sobel Filters works in a way where the algorithm focuses more on the centered pixels, so that noise is reduced and edge response is sharper. Shortly, it works by calculating the gradient of image intensity for each pixel, where the middle ones are weighted heavier than outer ones(in 3x3 picture).

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

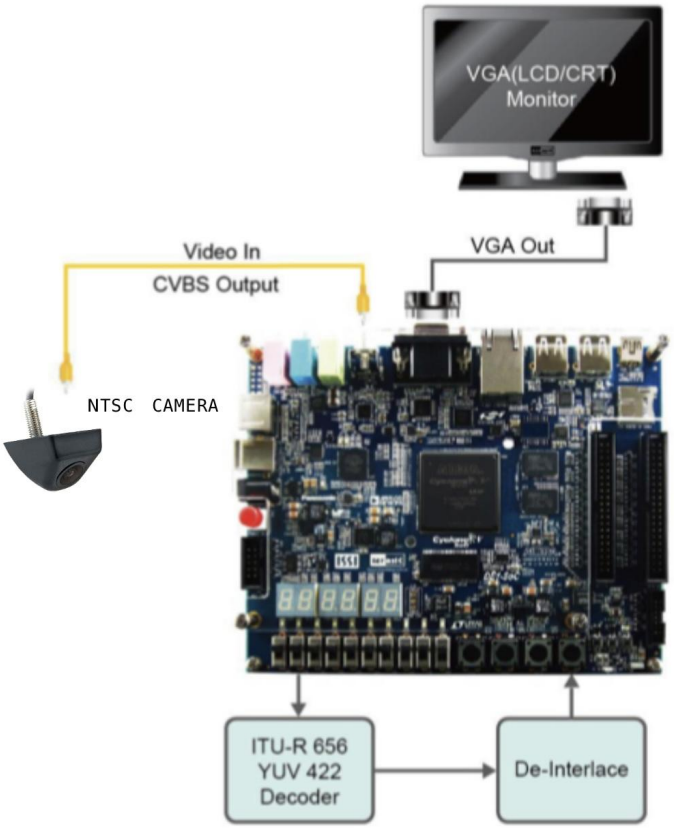
Sobel filter for vertical edge detection **Sobel filter for horizontal edge detection**

Picture 3. Sobel convolution kernels

The 9 pixel(3x3) input grid is obtained from an intensity calculator, which is later used for all the grid of the VGA display. The sum in both Verilog files are checked for the negativity, since one of the sums may appear negative. In order to avoid this, the OR loop is used so that the left-right(sum2) and right-left(sum1) are always printed only on the positive one. Then this sum is checked whether it is against the threshold, so we can avoid complications with sign on comparisons.

The threshold is a variable which can be registered via the push button and switches. Originally this variable was at 16 bits, however, to reduce timing issues it was shortened down to 10 bits(0:9). In the end, it outputs 1 for pixel edge and 0 for pixel not an edge.

4. Software



Picture 4. System scheme of project

Code:

Here is the link for the project code:
https://drive.google.com/file/d/1RQjGKfcI5HmN0zN5U_MbZZFhItNkCS8l/view?usp=sharing

5. Discussion and Conclusions

We think that the project on an FPGA is the best way to process the video data. It can help in medical image processing to faster determine diseases (for example, edge detection) or for robots to only focus on the edges of the objects for vision.

Also, now we know how to use it and in the future develop our initial idea with gesture control and do research on FPGA. Now, we know how to get the data for processing, we now detect a green color, and use it for controlling the switch of the music by gesturing with hand the sign “play next”.

6. Student Contributions

- Aidana Kalimbekova (VGA controller, 3 buffer line, assigning bits to registers)
- Aizhuldyz Nadirkhanova (Software installation, edge detection algorithms, DE1-SoC main module)
- Anuar Karzhaubayev (Intensity calculation, camera wires connection to FPGA)

7. References

1. People.ece.cornell.edu. 2022. *ECE 5760*. [online] Available at: <https://people.ece.cornell.edu/land/courses/ece5760/>.
2. People.ece.cornell.edu. 2022. [online] Available at: <https://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/DE1-SoC_User_manualv.1.2.2_revE.pdf>*ECE 5760 Final Projects*. 2022. [Video] https://www.youtube.com/playlist?list=PL2E0D05BEC0140F13.
3. DE1-SoC. User manual. 2019. Terasic Technologies. http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=836&PartNo=4#contents