Software-Defined Smart Camera

B.E. (CIS) PROJECT REPORT

by

Iqra Haider Malik

Department of Computer and Information Systems Engineering

NED University of Engineering & Technology,
Karachi-75270

Software-Defined Smart Camera

B.E. (CIS) PROJECT REPORT

Project Group:

IQRA HAIDER MALIK	CS-16011
EHAB REHMAN	CS-16008
MARIA SYED	CS-16004

BATCH: 2016-17

Project Advisors:

Dr. Majida Kazmi	(Internal Advisor)
Dr. Arshad Aziz	(External Advisor)

October 2020

Department of Computer and Information Systems Engineering

NED University of Engineering & Technology,
Karachi-75270

ABSTRACT

Image processing applications need to analyze and process a large number of images and video streams in real-time. This real-time constraint can be handled if the live camera feed is preprocessed right at the edge device i.e. an FPGA. This will shift some of the load from the servers where these applications usually exist and also allow the preprocessing to be adaptable to the environment where the video stream is being captured. The FPGA coupled with an image sensor is used to develop a camera that is adaptive and can preprocess the incoming live camera feed using high-level synthesis. Thus, this approach proposes to define a Software-Defined Smart Camera that utilizes the software-hardware codesign functionality of the Zynq-7000 FPGA.

CONTENTS

CHAPTER 1

Introduction

All the computer vision algorithms that perform image analysis and processing require high-quality images [1], but the conventional cameras available nowadays have fixed hardware i.e. their internal architecture is not programmable, so they cannot be exploited to their full extent.

When faced with real-time applications that need to process the data as fast as possible, the images need to be preprocessed as well before they can be used. Using an embedded approach at the point of video capture to perform the preprocessing will cut down a significant amount of overwork. [2][3]. For this purpose, existing image enhancement algorithms can be converted into a synthesizable design on the FPGA board. [4]

Moreover, these cameras are also not aware of their environment and are not adaptive. As a result, acquired images need to be processed before further use.

This leads to the need for a software-defined smart camera, which can enhance images using images preprocessing algorithms along with the ability to set itself according to the environment at the time of the video capture.

1.1 Objectives

To develop an efficient software-defined smart camera that can preprocess the input live video stream before and after receiving input from the image sensor.

- Studying and determining exact parameters which can be manipulated to produce images of high quality.

- Use of gathered data to profile the environment.
- Integrating models of the environment for an efficient solution.
- Exploiting the internal architecture of the image sensor.
- Using HLS to make image pre-processing IP cores
- Implementing a solution that will integrate the cores and manipulate the image sensor to get a Software-Defined Smart Camera

CHAPTER 2

Literature Review

Since the last century, a multitude of cameras have been in use. Imaging solutions have become a basic requirement in today's modern world and its technology. There are thousands of applications for cameras in different fields and the solutions are as unique as surveillance (to fight crimes and record material evidences); military (to predict the battlefield high in the sky used in fighter jets, bomb detection and many more); medical (3D-Medical imaging used in MRI, endoscopes, tooth-scanning, inspecting eye retina, skin tissues etc.); industries (fruit sorting, pharmaceuticals, automotive mining); traffic control systems (to enforce the law and avoid accidents). Hence, it is a ubiquitous technology. All of these applications depend on the image quality of the camera. Higher the image quality better the processing. Less noise and data loss, greater accuracy. Hence, image quality is an essential element for all these applications.

The conventional approach is to use a traditional camera (fixed hardware therefore fixed functionality). These cameras are not flexible and are unaware of their environments and hence cannot provide a constant and standard image quality in all scenarios. If the environment varies and unexpected noise is generated, since they have fixed hardware, they fail to maintain the standard image quality. The more intelligent camera provides an expensive solution that only operates under certain circumstances and can provide images that are visually acceptable to humans, but they need more processing to generate clean and clear images if they are to be used in computer vision applications.

A majority of this image enhancement and quality amplification task is being carried out on the server-side where these images are used as data to these computer vision applications. As we know that more authentic data produces more accurate output. These servers use different computational extensive as well as cost expensive algorithms to intensify the image quality so that they are capable of further processing (i.e. in face detection, if the image quality is compromised the system might not recognize the person correctly). If we consider a network of cameras, each stored in a different environment having different signal-to-noise ratio, the computational complexity of server-side increases as it has to respond to hundreds of requests at a time, considering every type of environment and noise. Another problem that usually arises in this conventional approach is that, if the image captured has as an in-depth data loss, these algorithms which are used to amplify the quality might add more noise to the image (i.e. if the image captured is extreme dark these algorithms has no data that can be recovered in order to increase the image quality). A simpler, faster, better and efficient solution is Edge Computing.

2.1 Edge Computing

Edge computing provides a way to process the images directly on the camera [2], taking advantage of the sensors to become environmentally conscious and harnessing the power of end devices to take over the load of preprocessing images [3][5]. Our strategy is to make our camera smart enough so that it can sense its surroundings and noise, adjusts its internal hardware, accordingly, select the best parameters for the situation even before when the image is captured. So, the quality of the captured image is constant (i.e. to avoid situations with extreme dark image). Since it is a practical approach not ideal hence some of the images might be affected but there would not be in-depth data loss hence data recovery will be possible.

2.2 High-Level Synthesis for IP Generation

To further preprocess the live video stream, the image enhancement can be done after the video stream is obtained from the image sensor and before any other image processing can be implemented [6], utilizing the Vivado High-Level Synthesis (HLS) tool to generate IPs that will preprocess the incoming video feed before the camera's output stream is generated. Using the HLS tool, a hardware/software codesign approach can be used that will enable the ability to add image processing IP cores by designing algorithms in languages such as C and C++ and then integrating them into the camera's hardware design [4]. The FPGA's reconfigurable capability [7] will allow the IPs to be added or removed as needed, and this property can be used to add more functionality to the design later on as well [8]. The HLS will also significantly cut down the design and testing time relative to making the IPs directly using HDL languages (Verilog or VHDL) [9]. This method harnesses the power of Vivado's software to convert the functional specifications in high-level languages directly into the RTL design.

CHAPTER 3

Hardware and Development Tools

3.1 Hardware

3.1.1 FPGA Board

For development purposes, the Zybo Z7-10 development board is being used. It contains the Zynq processor which has a dual-core ARM Cortex-A9 processor tightly integrated with the Xilinx 7-series Field Programmable Gate Array (FPGA) logic.

The peripherals, present on the board, that are relevant to the project include the MIPI CSI-2 compatible Pcam connector, HDMI output and Pmod ports. [10]

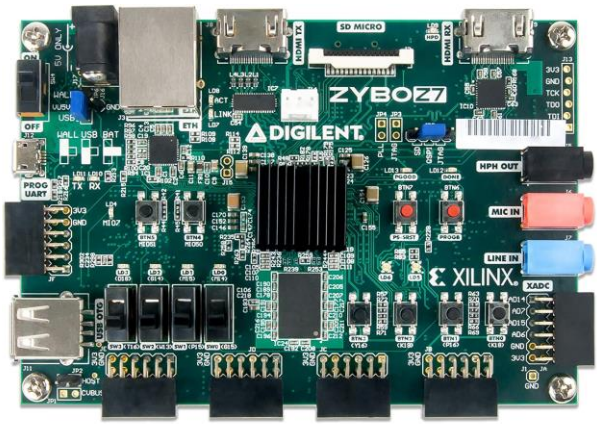

Figure 3.1 Zybo Z7: Zynq-7000 ARM/FPGA SoC Development Board

3.1.2 Image Sensor

Our goal was to identify a programmable image sensor that will provide a way to ‘preprocess’ the camera input before the capture of images. For this reason, the imaging module, used, is the PCAM 5C, that is based around the Omnivision OV5649 image sensor.

Figure 3.2 PCAM 5C containing the Omnivision OV5649 image sensor

3.2 Development Tools

3.2.1 Xilinx Vivado Design Studio

Vivado Design Studio allows the functionality to synthesis and implement hardware designs on FPGA using hardware-definition languages (HDL). The graphical block design extends this IDEs ability to design hardware by adding the ability to integrate IPs directly in the top-level block design and connect them to other components.

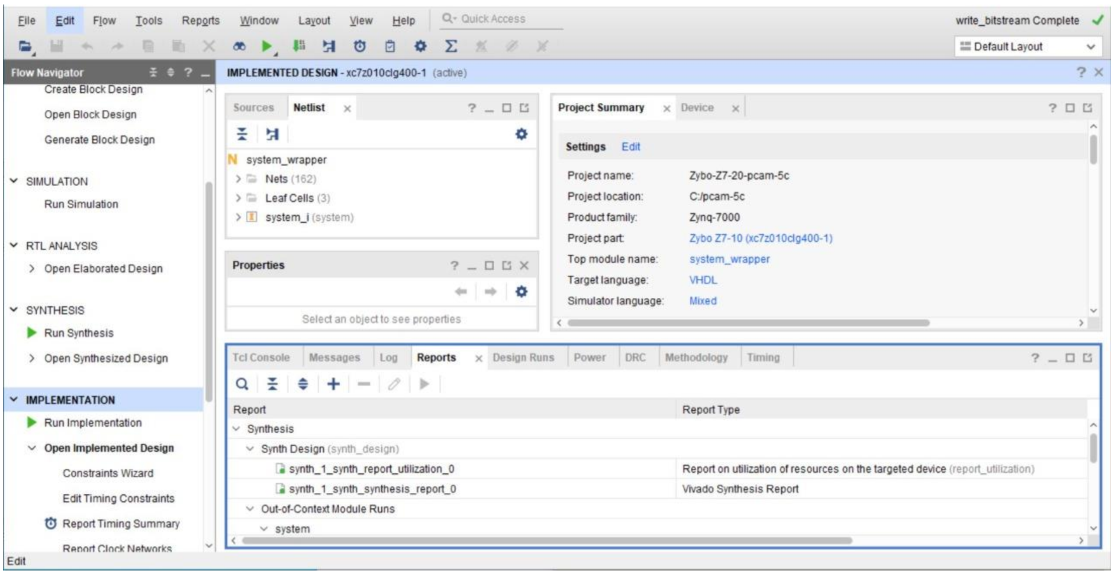

Figure 3.3 Vivado Design Studio

3.2.2 Xilinx Software Development Kit (SDK)

The Xilinx Software Development Kit (SDK) is an Integrated Development Environment (IDE). It is used for the development of embedded software applications on the FPGA board. Utilizing the SDK, programs designed in C or C++ can run on top of the HDL designs developed through the Vivado Design Studio.

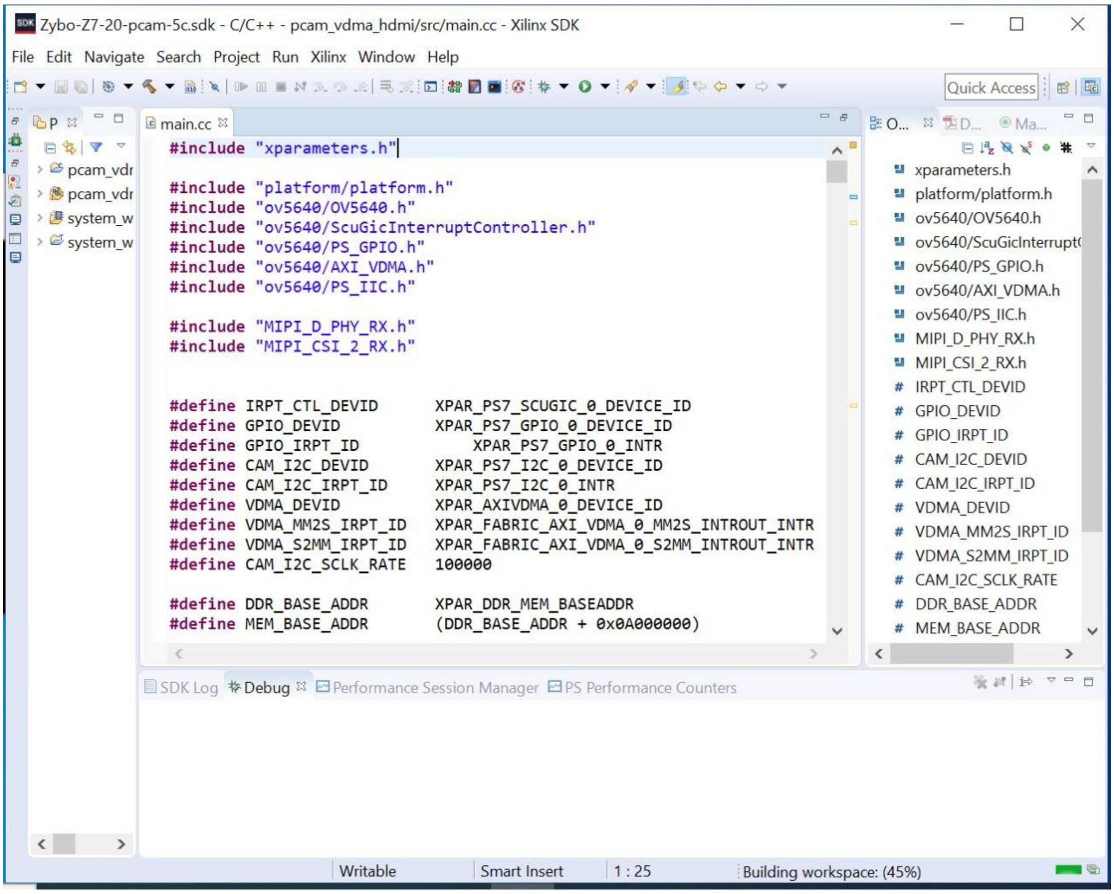

Figure 3.4 Xilinix SDK

3.2.3 Xilinx Vivado High-Level Synthesis (HLS)

The Xilinx Vivado High-Level Synthesis (HLS) tool allows the generation of IPs using software languages such as C, C++ and system C. This hardware-software co-design approach accelerates the time needed to make new IPs and test them [11] [12]. It also includes a number of useful libraries for a wide range of applications.

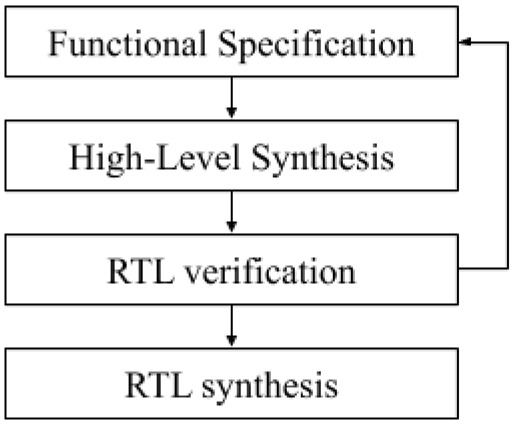

Figure 3.5 HLS Design Flow [13]

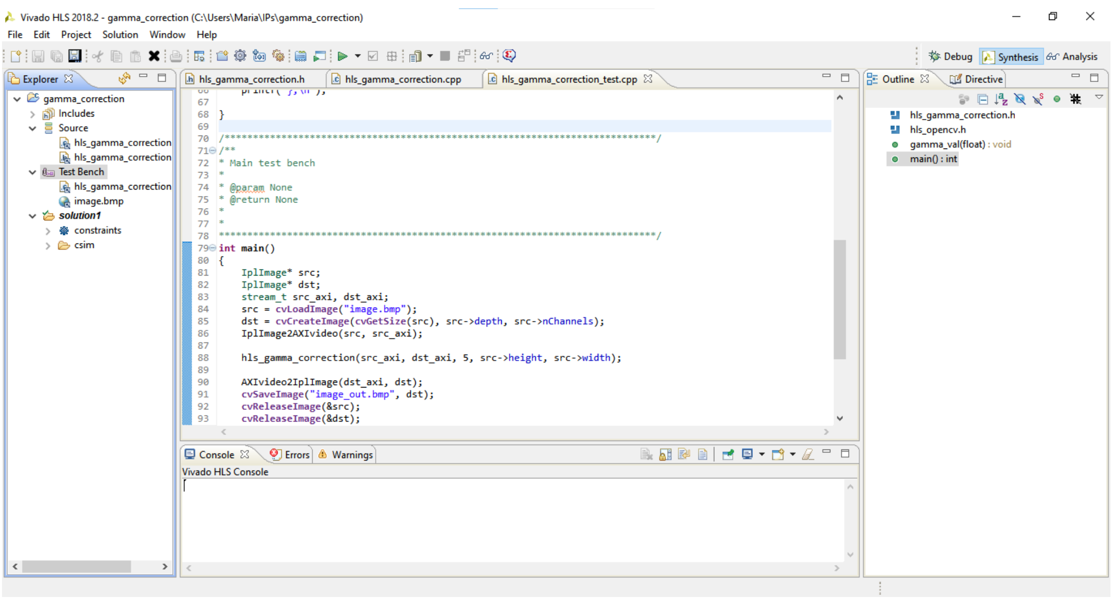

Figure 3.6 Vivado HLS IDE

3.2.4 Teraterm

Teraterm is a terminal emulator program. It was used to communicate with the FPGA board using a serial connection from a computer.

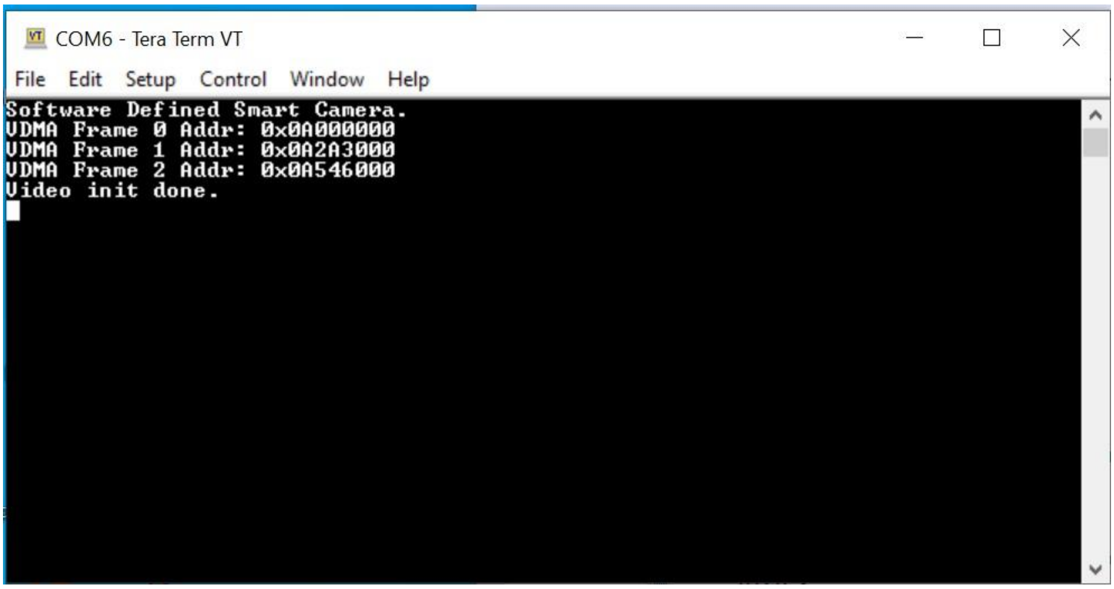

Figure 3.7 Teraterm Console

CHAPTER 4

Methodology

4.1 Environmental Profiling

Before using the FPGA board and a connected camera, conventional cameras were used for the purpose of environmental profiling. For this objective, various environments were chosen. With one particular location, the camera was placed in a fixed position and was made to capture the images at regular intervals. These images were then analyzed in MATLAB and their certain parameters were calculated. These image parameters comprised of brightness, hue, saturation, sharpness, and luminance. Studying these parameters and their change that result in images of varying degree of quality helped determine the most relevant ones which can be later manipulated in preprocessing of the video stream captured by the image sensor used later.

Table 1 Environmental Profiling using a traditional Camera in a particular environment

Image #	Time	Mean Brightness	Median Brightness	Mean Saturation	Median Saturation	Mean Hue	Median Hue	Sharpness	Mean Luminance	Median Luminance
1	6:32:00	0.550520803	0.584313725	0.142395522	0.070588235	0.298135255	0.273809524	5.949145965	136.5677421	145
2	6:33:00	0.567795377	0.603921569	0.157405229	0.077348066	0.296065907	0.272222222	6.092881948	140.4469348	149
3	6:34:00	0.580999818	0.619607843	0.145564895	0.069651741	0.299879782	0.277777778	6.612048545	143.9801842	153
4	6:35:00	0.595612374	0.635294118	0.145356	0.068571429	0.299151288	0.277777778	6.991325781	147.555352	158
5	6:36:00	0.592914617	0.631372549	0.143014786	0.067669173	0.300115478	0.277777778	6.995509533	146.9746856	157
6	6:37:00	0.603293894	0.643137255	0.135822801	0.068627451	0.31015093	0.291666667	7.136078657	149.5317785	159
7	6:38:00	0.59942015	0.639215686	0.136748751	0.065789474	0.303042577	0.277777778	7.633507411	148.7260745	158
8	6:39:00	0.601251116	0.639215686	0.136862795	0.066176471	0.299531191	0.270833333	7.673207469	149.1829456	159
9	6:40:00	0.601856553	0.639215686	0.136285214	0.065989848	0.296831473	0.270833333	7.64822469	149.3608073	159
10	6:41:00	0.60081587	0.639215686	0.135705359	0.066666667	0.295252254	0.269230769	7.655987238	149.0937587	159
568	15:59:00	0.53083274	0.564705882	0.175689592	0.12804878	0.348272606	0.321428571	8.165185715	128.7013513	135
569	16:00:00	0.531632546	0.564705882	0.170819297	0.124324324	0.349545399	0.333333333	7.77066837	129.0550019	136
570	16:01:00	0.537557534	0.57254902	0.170660854	0.123076923	0.345973425	0.3125	8.338961885	130.5605199	138
571	16:02:00	0.538943964	0.57254902	0.170912778	0.123188406	0.345447643	0.3125	8.279745209	130.8921528	138
572	16:03:00	0.537760061	0.57254902	0.1718341	0.125	0.346854277	0.316666667	8.396601902	130.515653	138
573	16:04:00	0.530813684	0.560784314	0.176561741	0.125827815	0.343021836	0.305555556	7.642751882	128.8661748	134
574	16:05:00	0.532722302	0.564705882	0.174071373	0.12568306	0.347890765	0.321428571	8.044829986	129.1828665	136
575	16:06:00	0.529400766	0.564705882	0.174009919	0.126760563	0.346495061	0.3125	8.188513509	128.3808111	135
576	16:07:00	0.527202796	0.560784314	0.171836395	0.126373626	0.347144191	0.314814815	7.968723546	127.937907	135

In the above parameters' table, each of the parameter's value changes with changing environmental conditions. Some are more affected than others. For example, here, sharpness is the most varying.

4.2 Image Sensor Integration

The PCAM module was connected to the board via a 15-pin flat-flexible cable (FFC). The board controls the image sensor with a MIPI CSI-2 controller in hardware. [14] To configure it in software, a set of open-source Vivado IP cores were used. However, the complete integration of the imaging module with our FPGA board proved to be a laborious task. The example projects provided by the manufacturer, Digilent, were not available for the selected board. But there was one present for the integration of Zybo Z7-20 and PCAM. After carefully studying the difference between this board and Z7-10, it was found that they were different with respect to memory that was available on the board. With this in consideration, the debug module of the IP block MPI_CSI_RX was safely removed, as its elimination did not restrict the functionality of the design, and an HDL design was formed that integrated Zybo Z7-10 with PCAM.

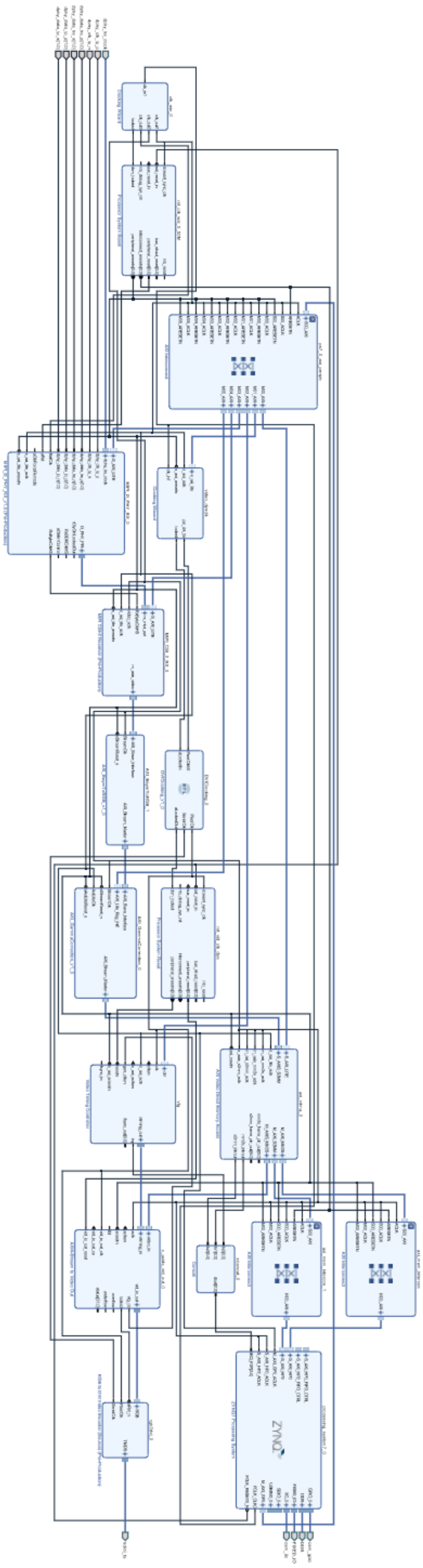

Figure 4.1 Block Design of the Image Sensor Integrated Hardware

4.3. Manipulation of Internal Architecture of Image Sensor

With the use of the Xilinx SDK, an embedded application was developed to run on the FPGA for configuration of the image sensor. Using C++, the sensor's internal architecture was exploited on the fly. In order to communicate with the FPGA board and the connected image sensor, Teraterm was used to set up a serial connection between the board and the computer. This functionality granted the ability to program how the camera captured the live stream.

The program that was used can be found in Appendix A.

4.3.1 Results

Experimenting with exploiting the internal architecture of the image sensor in different ways on different environments, resulting in the following observations,

Figure 1 Observation of Images Sensor's Internal Values and their Effect

REGISTER ADDRESS	OBSERVATIONS IN DIFFERENT ENVIRONMENTS	REGISTER FUNCTION	DEFAULT (DATASHEET VALUE)	INITIAL VALUE OF REGISTER	CHANGED VALUE	BINARY VALUE	EFFECT
0x3820	Environment 1	vertical flip	0x40	0x46	0x00	0000 0000	vertical flip
					0x01	0000 0001	BLACK BAR on top black bar on middle blinking
					0x02	0000 0010	red green (REVERSE FILTER)
					0x03	0000 0011	2+ black bar
					0x04	0000 0100	red green (REVERSE FILTER) + FLIP
					0x05	0000 0101	tera term freeze (illegal value)
					0x06	0000 0110	no change
					0x07	0000 0111	system freeze (illegal value)
					0x08	0000 1000	FLIP VERTICAL
					0x09	0000 1001	system freeze (illegal value)

					0x0a	0000 1010	red green filter
					0x0b	0000 1011	freeze+ black bar + middle black bar blink
					0x0c	0000 1100	red green filter + flip
					0x0d	0000 1101	system freeze (illegal value)
					0x0e	0000 1110	no change
					0x0f	0000 1111	system freeze (illegal value)
					ff	1111 1111	no change
0x4001 register enabled							
0x4002	Environment 1	black level calibration	0x45	0x45	0x45	0100 0101	no change
					0x00	0000 0000	slight change
					0x85	1000 0101	BLACK COLOR SEEMS TO BE BLACK ONLY
0x4003			0x08	0x09	0x48	0100 1000	no change
0x4005			0x18	0x19	0x00	0000 0000	no change
0x4009			0x10	0x11	0x10	0001 0000	less white
					0x20	0010 0000	whiter
					0x30	0011 0000	whiter
					0x11	0001 0001	no change
					0xe3	1101 0011	clearer
					0xff	1111 1111	clearer than e3
0x5581	Environment 2	special digital effect (SDE)			ff	1111 1111	greenish effect (changes remain by changing one or many register)
0x5582					ff	1111 1111	
0x5583					ff	1111 1111	
0x5584					ff	1111 1111	
0x5585					ff	1111 1111	
0x5586					ff	1111 1111	
0x5587					ff	1111 1111	
0x5588					ff	1111 1111	
0x5589					ff	1111 1111	
0x558a					ff	1111 1111	
0x558b					ff	1111 1111	
0x558c					ff	1111 1111	
0x558f					ff	1111 1111	
0x3503 is enabled (first enable then change value)							
0x350a	Environment 3	AGC (average luminance)	0x00	0x00	0xff	1111 1111	brighter than original
0x350b			0xf8	0xf8	0x00	0000 0000	normal brightness
0x3500			0x00	0x00	0x35	0011 0101	brighter than original

0x3501			0x6e	0x6e	0x22	0010 0010	less bright closer to dark
0x3502			0xa0	0xa0	77	0111 0111	no change
					ff	1111 1111	no change
0x350b			0xf8	0xf8	0	0000 0000	to darker side
0x3502			0xa0	0xa0	0xee	1101 1101	no change
0x3503 is enabled (first enable then change value)							
0x350a and 0x350b more effect as extreme values							
0x350a			0x00	0x00	0x00	0000 0000	not much difference as close to original range
0x350b			0xf8	0xf8	0xff	1111 1111	
0x3501			0x6e	0x6e	0x00	0000 0000	no change
0x350a			0x00	0x00	0xff	1111 1111	close to original
0x350b			0xf8	0xf8	0x00	0000 0000	
0x3501	Environment 3	AGC (average Luminance)	0x6e	0x6e	0x88	1000 1000	
0x3503 is enabled							
0x350a			0x00	0x00	0x68	0110 1000	darker
0x350b			0xf8	0xf8	0x78	0111 1000	
0x3500			0x00	0x00	0xff	1111 1111	no change
0x3501			0x6e	0x6e	0x22	0010 0010	no change
0x3502			0xa0	0xa0	0xee	1110 1110	no change
without changing gain, if we are changing exposure, no change as 0x380e and 0x380f is 0x04 and 0x60 so range is from 4 to 60 0x3500,0x3501,0x3502 should have values less than this range							
0x3503 is enabled							
0x3500			0x00	0x00	0x33	0011 0011	no change
0x350a	Environment 3	AGC (average luminance)	0x00	0x00	0xdd	1101 1101	brighter image
0x350b			0xf8	0xf8	0xf8		
changing values of 0x380e and 0x380f to increase range							
0x380e, new value:0x 0f , 0x380f, new value:0xAA							
0x3501	Environment 3	AGC (luminance)	0x6e	0x6e	0x11	0001 0001	no change
0x3503 enabled							
0x3a1b			0x78	0x78	0x12	0001 0010	range set
0x3a1e			0x68	0x68	0xff	1111 1111	
0x350a			0x00	0x00	0xff	1111 1111	white color is brighter
0x350c			0x00	0x00	0xff	1111 1111	object duplicated and greenish-yellow effect
0x350d	Environment 3	AGC (Average Luminance)	0x00	0x00	0x11	0001 0001	frame move upward, with same greenish-yellow effect
0x350d			0x00	0x00	0xff	1111 1111	image closer to normal but brighter and clearer
0x501d: manual mode on							
0x5001 enabled							
0x5581	Environment 3	SDE	0x80	0x88	0xff	1111 1111	greenish color

0x5582			0x00	0x00	0xff	1111 1111	

Table 2 Observation of Images Sensor's Internal Values and their Effect

Register Address	Register Value (Default)	Register Value (Changed)	Register Value (Changed) In Binary	Image Obtained	Effect Observed
0x4009	0x11	0x11	0001 0001		no change
0x4009	0x11	0xe3	1110 0011		more clear
0x4009	0x11	0xff	1111 1111		more clear than e3

4.4 Integration of Image Processing IPs

4.4.1 Image Processing Cores using Vivado HLS

IPs or Intellectual Property cores or blocks in the FPGA environment are reusable packaged hardware modules that can be added to other hardware designs to extend their functionality. They can be either hard core IPs or soft core IPs. Hard core IPs are designed by a foundry and cannot be modified or reused in other projects not intended for their use at all. On the other hand, soft IP cores are synthesizable RTL and made in and distributed in Verilog or VHDL languages. These can be then modified later as well and adapted to suit other applications. This is the type chosen for the image processing IPs to be used.

The RTL of the IPs can be designed using Verilog or VHDL, which are hardware-definition languages. But for this application, the Vivado High-Level Synthesis (HLS) tool has been utilized, which allows the algorithmic generation of IPs using C.

Before any application can work to extract information from an image, it needs to be preprocessed to remove unwanted distortions and to enhance certain features. For this software-defined smart camera, three algorithms have been used that will perform image enhancement in order to preprocess the images before further use by other applications. These algorithms are gamma correction, contrast stretching and saturation enhancement.

4.4.1.1 High-Level Synthesis

Each IP that was to be designed was first programmed in C++. Then it was tested using the simulation ability of the HLS tool to verify its working. Then it was synthesized and thus become ready to be imported to projects made with the Vivado Design Studio.

4.4.1.1.1 Testing the HLS Tool

Before implementing the required image processing program, a number of simpler code for certain image processing algorithms in C++ were simulated and synthesized to generate their IP blocks. The code was tested on the following image.

Figure 4.2 Original Image

4.4.1.1.1.1 Black and White Conversion

black_and_white.h

```
#include "hls_video.h"
typedef ap_axiu<24,1,1,1> interface_t;
typedef ap_uint<3> interface_3_bits;
typedef hls::stream<interface_t> stream_t;

void color_to_bw(stream_t &stream_in, stream_t &stream_out);

#define MAX_WIDTH      1280
#define MAX_HEIGHT     720

#define INPUT_IMAGE "fox.bmp"
#define OUTPUT_IMAGE "fox_output.bmp"

typedef hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC3> rgb_img_t;
typedef hls::Scalar<3, unsigned char> rgb_pix_t;
```

black_and_white.cpp

```
#include "black_and_white.h"


void color_to_bw(stream_t &stream_in, stream_t &stream_out){

        int const rows = MAX_HEIGHT;
        int const cols = MAX_WIDTH;
```

```
        rgb_img_t img0(rows, cols);
        rgb_img_t img1(rows, cols);

        hls::AXIvideo2Mat(stream_in, img0);
        hls::CvtColor<HLS_RGB2GRAY>(img0, img1);
        hls::Mat2AXIvideo(img1, stream_out);
}
```

black_and_white_test.cpp

```
#include "black_and_white.h"
#include "hls_opencv.h"

int main(){
        int const rows = MAX_HEIGHT;
        int const cols = MAX_WIDTH;

        cv::Mat src = cv::imread(INPUT_IMAGE);
        cv::Mat dst = src;
        stream_t stream_in, stream_out;
        cvMat2AXIvideo(src, stream_in);
        color_to_bw(stream_in, stream_out);
        AXIvideo2cvMat(stream_out, dst);
        cv::imwrite(OUTPUT_IMAGE, dst);

        return 0;
}
```

Simulation Result,

Figure 4.3 Black and White Conversion

4.4.1.1.1.2 Color Inversion

invert.h

```
#include "hls_video.h"

typedef ap_axiu<24,1,1,1> interface_t;
typedef ap_uint<3> interface_3_bits;
typedef hls::stream<interface_t> stream_t;

void invert(stream_t &stream_in, stream_t &stream_out);

#define MAX_WIDTH      1280
#define MAX_HEIGHT   720

#define INPUT_IMAGE "fox.bmp"
#define OUTPUT_IMAGE "fox_output.bmp"

typedef hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC3> rgb_img_t;
typedef hls::Scalar<3, unsigned char> rgb_pix_t;
```

invert.cpp

```
#include "invert.h"


void invert(stream_t &stream_in, stream_t &stream_out)
{

        int const rows = MAX_HEIGHT;
        int const cols = MAX_WIDTH;

        rgb_img_t img0(rows, cols);
        rgb_img_t img1(rows, cols);

        rgb_pix_t pix(250,250,250);

        hls::AXIvideo2Mat(stream_in, img0);
        hls::SubRS(img0, pix, img1);
        hls::Mat2AXIvideo(img1, stream_out);
}
```

invert_test.cpp

```
#include "invert.h"
#include "hls_opencv.h"

int main()
{
        int const rows = MAX_HEIGHT;
        int const cols = MAX_WIDTH;

        cv::Mat src = cv::imread(INPUT_IMAGE);
```

```
        cv::Mat dst = src;

        stream_t stream_in, stream_out;

        cvMat2AXIvideo(src, stream_in);
        invert(stream_in, stream_out);
        AXIvideo2cvMat(stream_out, dst);

        cv::imwrite(OUTPUT_IMAGE, dst);

        return 0;
}
```

Simulation result,

Figure 4.4 Color Inversion

4.4.1.1.1.3 Histogram Equalization

hist_qualization.h

```
#include "hls_video.h"
#include <ap_fixed.h>

#define MAX_WIDTH  2000
#define MAX_HEIGHT 2000

#define INPUT_IMAGE "fox.jpg"
#define OUTPUT_IMAGE "fox_output.jpg"
```

```
typedef hls::stream<ap_axiu<32,1,1,1> >     AXI_STREAM;
typedef hls::Mat<MAX_HEIGHT,   MAX_WIDTH,   HLS_8UC3> RGB_IMAGE;
typedef hls::Mat<MAX_HEIGHT,   MAX_WIDTH,   HLS_8UC1> GRAY_IMAGE;


void getHistEq(AXI_STREAM& INPUT_STREAM, AXI_STREAM& OUTPUT_STREAM,
int rows, int cols);
```

hist_equalization.cpp

```
#include "hist_equalization.h"
void getHistEq(AXI_STREAM& INPUT_STREAM, AXI_STREAM& OUTPUT_STREAM,
int rows,int cols) {

#pragma HLS INTERFACE axis port=INPUT_STREAM
#pragma HLS INTERFACE axis port=OUTPUT_STREAM

  RGB_IMAGE img_main(rows, cols);

  GRAY_IMAGE img_gray(rows, cols);

  GRAY_IMAGE img_grayCopy1(rows, cols);
  GRAY_IMAGE img_grayCopy2(rows, cols);

  GRAY_IMAGE img_grayhisteq1(rows, cols);
  GRAY_IMAGE img_grayhisteq2(rows, cols);

  RGB_IMAGE img_fin(rows, cols);


  hls::AXIvideo2Mat(INPUT_STREAM, img_main);
  hls::CvtColor<HLS_BGR2GRAY>(img_main, img_gray);
  hls::Duplicate(img_gray, img_grayCopy1, img_grayCopy2);

  hls::EqualizeHist(img_grayCopy1,img_grayhisteq1);
  hls::EqualizeHist(img_grayCopy2,img_grayhisteq2);
  hls::CvtColor<HLS_GRAY2RGB>(img_grayhisteq2, img_fin);
  hls::Mat2AXIvideo(img_fin, OUTPUT_STREAM);

}
```

hist_equalization.cpp

```
#include <hls_opencv.h>
#include "hist_equalization.h"
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
  IplImage* src;
  IplImage* dst;
  AXI_STREAM src_axi, dst_axi;
  src = cvLoadImage(INPUT_IMAGE);
  dst = cvCreateImage(cvGetSize(src), src->depth, src->nChannels);
  IplImage2AXIvideo(src, src_axi);
```

```
  getHistEq(src_axi, dst_axi, src->height, src->width);
  AXIvideo2IplImage(dst_axi, dst);
  cvSaveImage(OUTPUT_IMAGE, dst);
  cvReleaseImage(&src);
  cvReleaseImage(&dst);
}
```

Simulation Result,

Figure 4.5 Histogram Equalization

4.4.1.1.1.3 Edge Detection

edge_detect.h

```
#include "hls_video.h"

typedef ap_axiu<24,1,1,1> interface_t;
typedef hls::stream<interface_t> stream_t;

#define MAX_HEIGHT 720
#define MAX_WIDTH 1280

#define INPUT_IMAGE "fox.bmp"
#define OUTPUT_IMAGE "fox_output.bmp"

typedef hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC3> rgb_img_t;

void edge_detect(stream_t& stream_in, stream_t& stream_out);
```

edge_detect.cpp

```
#include "edge_detect.h"

void edge_detect(stream_t& stream_in, stream_t& stream_out)
{
                rgb_img_t img0(MAX_HEIGHT, MAX_WIDTH);
        #pragma HLS STREAM variable=img0 depth=1 dim=1
                rgb_img_t img1(MAX_HEIGHT, MAX_WIDTH);
        #pragma HLS STREAM variable=img1 depth=1 dim=1
                rgb_img_t img2(MAX_HEIGHT, MAX_WIDTH);
        #pragma HLS STREAM variable=img2 depth=1 dim=1
                rgb_img_t img3(MAX_HEIGHT, MAX_WIDTH);
        #pragma HLS STREAM variable=img3 depth=1 dim=1

        hls::AXIvideo2Mat(stream_in, img0);
        hls::CvtColor<HLS_RGB2GRAY>(img0, img1);
        hls::Sobel<1,0,3>(img1, img2);
        hls::CvtColor<HLS_GRAY2RGB>(img2, img3);
        hls::Mat2AXIvideo(img3, stream_out);
}
```

edge_detect_test.cpp

```
#include "edge_detect.h"
#include "hls_opencv.h"

int main()
{
        cv::Mat src = cv::imread(INPUT_IMAGE);
        cv::Mat dst = src;

        stream_t stream_in, stream_out;
        cvMat2AXIvideo(src, stream_in);
        edge_detect(stream_in, stream_out);
        AXIvideo2cvMat(stream_out, dst);

        cv::imwrite(OUTPUT_IMAGE, dst);

        return 0;
}
```

Figure 4.6 Edge Detection using Sobel filter

Next, the IPs were integrated one by one in the previous block design to test them out. The input came from the AXI Video Direct Memory Access, then the AXI video stream was processed and then forwarded to AXI-Stream to Video Out IP, that converts the AXI video to RGB output.

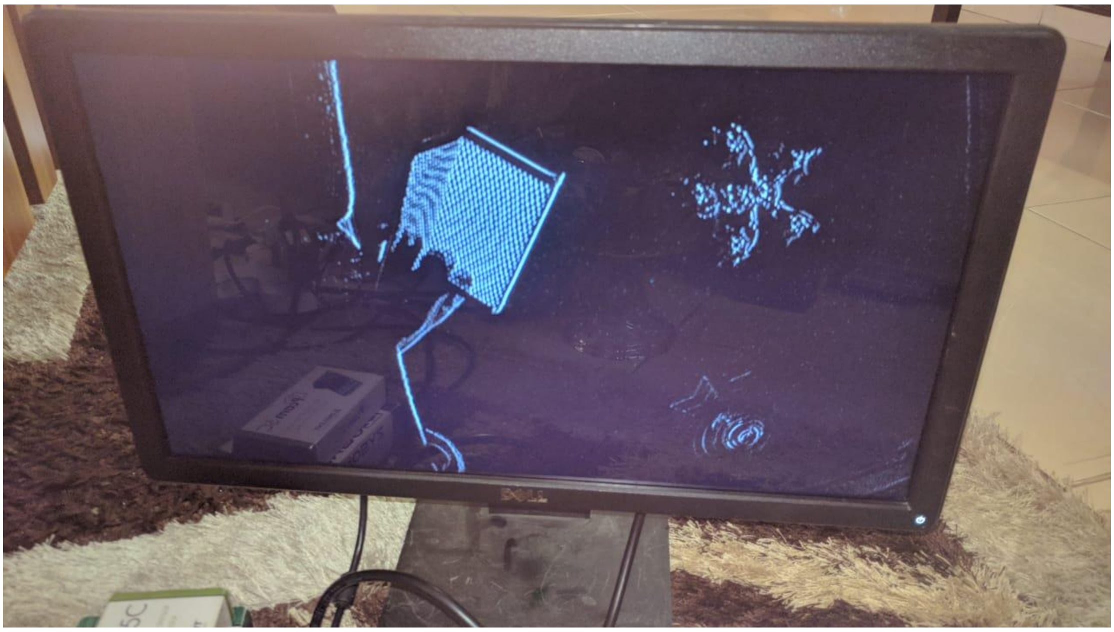

Figure 4.7 Testing the custom IP on FPGA board (Edge Detection)

4.4.1.1.2 Contrast Stretching

The following files were used in order to generate the contrast stretching IP,

hls_contrast_strech.h

```
#define HLS_CONTRAST_STRETCH_H_

#include "hls_video.h"

#define MAX_WIDTH      1920
#define MAX_HEIGHT  1080

#define INPUT_IMAGE "fox.jpg"
#define OUTPUT_IMAGE "fox_output.jpg"

typedef ap_axiu<24,1,1,1> interface_t;
typedef ap_uint<3> interface_3_bits;
typedef hls::stream<interface_t> stream_t;
typedef unsigned short u_int16_t;

typedef hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC3> rgb_img_t;
typedef hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC1> single_img_t;
typedef hls::Scalar<3, unsigned char> rgb_pix_t;


void hls_contrast_stretch(stream_t &stream_in, stream_t &stream_out,
u_int16_t height, u_int16_t width, unsigned char min, unsigned char
max);

#endif
```

hls_contrast_strech.cpp

```
//-- Purpose:
//-- This core has been written in order to perform a contrast
stretch
//-- on a input image stream (RGB 24 bits, 8 bits/color) and output
it to a stream
//-- (RGB 24 bits, 8 bits/color). The stretching is done by remapping
the pixels of
//-- histogram so that the min threshold is set to 0 and the max
threshold is set to
//-- 255. The remapping is done by applying the formula
//-- dst.cont = ((src.cont-min)*255)/(max-min).
//--
//----------------------------------------------------------------------
------------

#include "hls_contrast_strech.h"

/*
 * Applies the remapping of the contrast on a input matrix HLS_8UC3
based on
```

```
 * the threshold min and max. Remapping is done in real time based on
a simple
 * remapping formula
 *
 * @param src is the input matrix on 3 channels with each component
on 8 bits
 * @param dst is the output matrix on 3 channels with each component
on 8 bits
 * @param min the low threshold which will be set to 0
 * @param max the high threshold which will be set to 255
 *
 * @return None
 *
*/

void contrast_stretch(rgb_img_t &src, rgb_img_t &dst, unsigned char
min, unsigned char max)
{
#pragma HLS INLINE

        HLS_SIZE_T rows = dst.rows;
        HLS_SIZE_T cols = dst.cols;

        hls::Scalar<HLS_MAT_CN(HLS_8UC3), HLS_TNAME(HLS_8UC3)> s;
        hls::Scalar<HLS_MAT_CN(HLS_8UC3), HLS_TNAME(HLS_8UC3)> d;

        //loop trough the matrix
        loop_height: for (HLS_SIZE_T i = 0; i < rows; i++) {
           loop_width: for (HLS_SIZE_T j = 0; j < cols; j++) {
        #pragma HLS loop_flatten off
        #pragma HLS pipeline II=1
                    //dumping the input stream pixel in to a single
                    //24 bit variable (8 bits with 3 values)
                         src >> s;
                         if(s.val[0] > max)
                         {
                                 //set everything grater then max to
255
                                 d.val[0] = 255;
                         }
                         else
                         {
                                 if(s.val[0] < min)
                                 {
                                         //set everything less then min
to 0
                                         d.val[0] = 0;
                                 }
                                 else
                                 {
                                         //remapping the rest of the
values
                                         d.val[0] = (unsigned
char)(((s.val[0]-min)*255)/(max-min));
                                 }
                         }
                         //assuming that the format is yCbCr
                         //leave Cb and Cr unchanged
                         d.val[1] = s.val[1];
                         d.val[2] = s.val[2];
                         //dump the pixel in to the destination matrix
```

```
                        dst << d;
                }
            }

}

/**********************************************************************
********/
/**
 * This is the main function of the core, this function will be
synthesized
 * and packaged as an IP core. Its main purpose is to define the IP
ports
 * and busses while converting the RGB image in to HLS and calling
the contrast_strech
 * function.
 *
 * @param stream_in is the input Axi-Stream for video format
 * @param stream_out is the output Axi-Stream for video format
 * @param height parameter defines the height of the input image.
This parameter can
 *               be edited using the Axi-Light interface.
 * @param width parameter defines the width of the input image. This
parameter can
 *               be edited using the Axi-Light interface.
 * @param min sets the low threshold for the contrast stretch in
absolute terms. This
 *               parameter can be edited using the Axi-Light interface
 * @param max sets the high threshold for the contrast stretch in
absolute terms. This
 *               parameter can be edited using the Axi-Light interface
 *
 * @return None
 *

**********************************************************************
********/

void hls_contrast_stretch(stream_t &stream_in, stream_t &stream_out,
u_int16_t height, u_int16_t width, unsigned char min, unsigned char
max)
{
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS DATAFLOW
#pragma HLS INTERFACE s_axilite port=max
#pragma HLS INTERFACE s_axilite port=width
#pragma HLS INTERFACE s_axilite port=height
#pragma HLS INTERFACE s_axilite port=min
#pragma HLS INTERFACE axis register both port=stream_in
#pragma HLS INTERFACE axis register both port=stream_out

        u_int16_t rows = height;
        u_int16_t cols = width;


        rgb_img_t img0(rows, cols);
        rgb_img_t img1(rows, cols);
        rgb_img_t img2(rows, cols);
        rgb_img_t img3(rows, cols);

        float scale, offset;
```

```
        //transforms stream to matrix
        hls::AXIvideo2Mat(stream_in, img0);
        //convert RGB to YCbCr
        hls::CvtColor<HLS_RGB2YCrCb>(img0,img1);
        //perform contrast stretch
        contrast_stretch(img1, img2, min, max);
        //convert YCbCr to RGB
        hls::CvtColor<HLS_YCrCb2RGB>(img2,img3);
        //transform resulting matrix to output stream
        hls::Mat2AXIvideo(img3, stream_out);

}
```

Before being synthesized, the testbench file first needs to run to ensure that the code is correct.

`hls_contrast_strech_test.cpp`

```
#include "hls_contrast_strech.h"
#include "hls_opencv.h"

int main()
{
        IplImage* src;
        IplImage* dst;
        stream_t src_axi, dst_axi;
src = cvLoadImage(INPUT_IMAGE);
dst = cvCreateImage(cvGetSize(src), src->depth, src->nChannels);
IplImage2AXIvideo(src, src_axi);

hls_contrast_stretch(src_axi, dst_axi, src->height, src->width, 15,
230);

AXIvideo2IplImage(dst_axi, dst);
cvSaveImage(OUTPUT_IMAGE, dst);
cvReleaseImage(&src);
cvReleaseImage(&dst);

}
```

Simulation Result,

Figure 4.8 Contrast Stretching

4.4.1.1.3 Saturation Enhance

The following files were used in order to generate the saturation enhance IP,

hls_saturation_enhanche.h

```
#ifndef HLS_SATURATION_ENHANCHE_H_
#define HLS_SATURATION_ENHANCHE_H_

#include "hls_video.h"

#define MAX_WIDTH      1920
#define MAX_HEIGHT   1080

#define INPUT_IMAGE "fox.jpg"
#define OUTPUT_IMAGE "fox_output.jpg"

typedef ap_axiu<24,1,1,1> interface_t;
typedef ap_uint<3> interface_3_bits;
typedef hls::stream<interface_t> stream_t;

typedef hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC3> rgb_img_t;
typedef hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC1> single_img_t;
typedef hls::Scalar<3, unsigned char> rgb_pix_t;

void hls_saturation_enhance(stream_t &stream_in, stream_t
&stream_out, u_int16_t height, u_int16_t width, unsigned char sat);

#endif
```

hls_saturation_enhanche.cpp

```
//-- Purpose:
//-- This core has been written in order to perform a saturation
enhancement
//-- on a input image stream (RGB 24 bits, 8 bits/color) and output
it to a stream
//-- (RGB 24 bits, 8 bits/color). The enhancement is done by applying
the formula
//-- dst.sat = src.sat + src.sat*fact and a non-linear adaptation for
over saturation.
//-- In order to reduce the resource consumption of the IP the actual
transformation
//-- has been implemented using lookup tables with predefined
coefficients.
//-- If new LUTs have to be implemented please refer to the
hls_saturation_enhanche_test.cpp
//-- test bench which contains the elaborated formula of the
saturation enhancement
//-- algorithm
//--
//--------------------------------------------------------------------
------------


#include "hls_saturation_enhanche.h"


/************************** Constant Definitions
****************************/
const unsigned char lut_s_n0_2[256] =
{0,1,2,2,3,4,5,6,6,7,8,9,10,10,11,12,13,14,14,15,16,17,18,18,19,20,21
,22,22,23,24,25,26,26,27,28,29,30,30,31,32,33,34,34,35,36,37,38,38,39
,40,41,42,42,43,44,45,46,46,47,48,49,50,50,51,52,53,54,54,55,56,57,58
,58,59,60,61,62,62,63,64,65,66,66,67,68,69,70,70,71,72,73,74,74,75,76
,77,78,78,79,80,81,82,82,83,84,85,86,86,87,88,89,90,90,91,92,93,94,94
,95,96,97,98,98,99,100,101,102,102,103,104,105,106,106,107,108,109,11
0,110,111,112,113,114,114,115,116,117,118,118,119,120,121,122,122,123
,124,125,126,126,127,128,129,130,130,131,132,133,134,134,135,136,137,
138,138,139,140,141,142,142,143,144,145,146,146,147,148,149,150,150,1
51,152,153,154,154,155,156,157,158,158,159,160,161,162,162,163,164,16
5,166,166,167,168,169,170,170,171,172,173,174,174,175,176,177,178,178
,179,180,181,182,182,183,184,185,186,186,187,188,189,190,190,191,192,
193,194,194,195,196,197,198,198,199,200,201,202,202,203,204};
const unsigned char lut_s_0_2[256]  =
{0,1,2,4,5,6,7,8,10,11,12,13,14,15,17,18,19,20,21,23,24,25,26,27,28,3
0,31,32,33,34,35,36,38,39,40,41,42,43,44,46,47,48,49,50,51,52,54,55,5
6,57,58,59,60,61,63,64,65,66,67,68,69,70,71,72,74,75,76,77,78,79,80,8
1,82,83,85,86,87,88,89,90,91,92,93,94,95,96,97,98,100,101,102,103,104
,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,
123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,1
40,141,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,15
7,158,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173
,174,175,176,177,178,179,179,180,181,182,183,184,185,186,187,188,189,
190,191,192,192,193,194,195,196,197,198,199,200,201,201,202,203,204,2
05,206,207,208,209,210,210,211,212,213,214,215,216,217,217,218,219,22
0,221,222,223,223,224,225,226,227,228,229,229,230,231,232,233,234,235
```

```
,235,236,237,238,239,240,240,241,242,243,244,244,245,246,247,248,249,
249,250,251,252,253,253,254,255};
const unsigned char lut_s_0_4[256]  =
{0,1,3,4,6,7,8,10,11,12,14,15,17,18,19,21,22,23,25,26,27,29,30,31,33,
34,35,37,38,39,41,42,43,44,46,47,48,50,51,52,53,55,56,57,59,60,61,62,
64,65,66,67,69,70,71,72,73,75,76,77,78,80,81,82,83,84,86,87,88,89,90,
91,93,94,95,96,97,98,100,101,102,103,104,105,107,108,109,110,111,112,
113,114,116,117,118,119,120,121,122,123,124,125,126,128,129,130,131,1
32,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,14
9,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166
,167,168,169,170,171,172,173,174,175,176,177,177,178,179,180,181,182,
183,184,185,186,187,187,188,189,190,191,192,193,194,194,195,196,197,1
98,199,199,200,201,202,203,204,204,205,206,207,208,209,209,210,211,21
2,213,213,214,215,216,216,217,218,219,220,220,221,222,223,223,224,225
,226,226,227,228,228,229,230,231,231,232,233,233,234,235,236,236,237,
238,238,239,240,240,241,242,242,243,244,244,245,246,246,247,248,248,2
49,249,250,251,251,252,253,253,254,254,255};
const unsigned char lut_s_0_6[256]  =
{0,2,3,5,6,8,10,11,13,14,16,17,19,20,22,23,25,27,28,30,31,33,34,36,37
,39,40,41,43,44,46,47,49,50,52,53,55,56,57,59,60,62,63,64,66,67,69,70
,71,73,74,75,77,78,80,81,82,84,85,86,88,89,90,91,93,94,95,97,98,99,10
0,102,103,104,106,107,108,109,110,112,113,114,115,117,118,119,120,121
,123,124,125,126,127,128,130,131,132,133,134,135,136,138,139,140,141,
142,143,144,145,146,148,149,150,151,152,153,154,155,156,157,158,159,1
60,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,17
7,178,179,180,181,182,183,183,184,185,186,187,188,189,190,191,191,192
,193,194,195,196,197,197,198,199,200,201,202,202,203,204,205,206,206,
207,208,209,209,210,211,212,213,213,214,215,215,216,217,218,218,219,2
20,220,221,222,223,223,224,225,225,226,227,227,228,228,229,230,230,23
1,232,232,233,233,234,235,235,236,236,237,238,238,239,239,240,240,241
,241,242,242,243,244,244,245,245,246,246,247,247,248,248,248,249,249,
250,250,251,251,252,252,253,253,253,254,254,255,255};
const unsigned char lut_s_0_8[256]  =
{0,2,4,5,7,9,11,12,14,16,18,19,21,23,25,26,28,30,31,33,35,36,38,40,41
,43,45,46,48,50,51,53,54,56,58,59,61,62,64,65,67,69,70,72,73,75,76,78
,79,81,82,84,85,87,88,90,91,92,94,95,97,98,100,101,102,104,105,107,10
8,109,111,112,113,115,116,117,119,120,121,123,124,125,127,128,129,130
,132,133,134,135,137,138,139,140,141,143,144,145,146,147,149,150,151,
152,153,154,156,157,158,159,160,161,162,163,164,166,167,168,169,170,1
71,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,18
8,189,190,191,191,192,193,194,195,196,197,198,199,199,200,201,202,203
,204,204,205,206,207,208,208,209,210,211,212,212,213,214,215,215,216,
217,218,218,219,220,220,221,222,222,223,224,224,225,226,226,227,228,2
28,229,229,230,231,231,232,232,233,233,234,235,235,236,236,237,237,23
8,238,239,239,240,240,241,241,242,242,242,243,243,244,244,245,245,245
,246,246,247,247,247,248,248,248,249,249,249,250,250,250,251,251,251,
252,252,252,252,253,253,253,253,254,254,254,254,255,255,255};
const unsigned char lut_s_1_0[256]  =
{0,2,4,6,8,10,12,14,16,18,20,22,23,25,27,29,31,33,35,37,38,40,42,44,4
6,48,49,51,53,55,56,58,60,62,63,65,67,69,70,72,74,75,77,79,80,82,84,8
5,87,89,90,92,93,95,97,98,100,101,103,104,106,107,109,110,112,113,115
,116,118,119,121,122,124,125,127,128,129,131,132,134,135,136,138,139,
140,142,143,144,146,147,148,150,151,152,153,155,156,157,158,160,161,1
62,163,164,166,167,168,169,170,171,173,174,175,176,177,178,179,180,18
1,182,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199
,199,200,201,202,203,204,205,206,207,208,208,209,210,211,212,213,213,
214,215,216,217,217,218,219,220,220,221,222,223,223,224,225,225,226,2
27,227,228,229,229,230,231,231,232,232,233,234,234,235,235,236,236,23
7,237,238,238,239,239,240,240,241,241,242,242,243,243,244,244,244,245
,245,246,246,246,247,247,247,248,248,248,249,249,249,250,250,250,250,
```

```
251,251,251,251,252,252,252,252,253,253,253,253,253,253,254,254,254,2
54,254,254,254,254,255,255,255,255,255,255,255,255,255,255,255,255};
const unsigned char lut_s_1_2[256]  =
{0,2,4,7,9,11,13,15,17,19,22,24,26,28,30,32,34,36,38,40,42,44,46,48,5
0,52,54,56,58,60,62,64,66,67,69,71,73,75,77,79,80,82,84,86,88,89,91,9
3,95,97,98,100,102,103,105,107,108,110,112,113,115,117,118,120,122,12
3,125,126,128,129,131,132,134,136,137,139,140,141,143,144,146,147,149
,150,152,153,154,156,157,159,160,161,163,164,165,167,168,169,170,172,
173,174,175,177,178,179,180,182,183,184,185,186,187,189,190,191,192,1
93,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,209,21
0,211,212,213,214,215,216,217,218,218,219,220,221,222,223,223,224,225
,226,226,227,228,229,229,230,231,232,232,233,234,234,235,236,236,237,
237,238,239,239,240,240,241,241,242,242,243,244,244,245,245,245,246,2
46,247,247,248,248,249,249,249,250,250,250,251,251,251,252,252,252,25
3,253,253,254,254,254,254,254,255,255,255,255,255,255,255,255,255,255
,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,2
55};

/********************************************************************
********/
/**
 * Applies the saturation enhanced coefficients to the input matrix
HLS_8UC3,
 * form the desired LUT. the selection is done using sat variable as
a simple
 * switch between LUTs
 *
 * @param src is the input matrix on 3 channels with each color on 8
bits
 * @param dst is the output matrix on 3 channels with each color on 8
bits
 * @param sat is a 8 bit parameter which selects between saturation
factors
 *              which should be applied.
 *              sat == 0 =>  dst.sat = src.sat
 *              sat == 1 =>  dst.sat = src.sat - 0.2*src.sat
 *              sat == 2 =>  dst.sat = src.sat + 0.2*src.sat
 *              sat == 3 =>  dst.sat = src.sat + 0.4*src.sat
 *              sat == 4 =>  dst.sat = src.sat + 0.6*src.sat
 *              sat == 5 =>  dst.sat = src.sat + 0.8*src.sat
 *              sat == 6 =>  dst.sat = src.sat + 1.0*src.sat
 *              sat == 7 =>  dst.sat = src.sat + 1.2*src.sat
 *
 * @return None
 *
 * @note The pragmas are part of the HLS optimization flow and should
be
 *              researched before using them in other functions.
Please refer to
 *              UG902 of your current Vivado version
 *
*********************************************************************
********/
void sat_e(rgb_img_t &src, rgb_img_t &dst, unsigned char sat)
{
#pragma HLS INLINE

        HLS_SIZE_T rows = dst.rows;
        HLS_SIZE_T cols = dst.cols;
```

```
        hls::Scalar<HLS_MAT_CN(HLS_8UC3), HLS_TNAME(HLS_8UC3)> s;
        hls::Scalar<HLS_MAT_CN(HLS_8UC3), HLS_TNAME(HLS_8UC3)> d;

        //loop trough the matrix
        loop_height: for (HLS_SIZE_T i = 0; i < rows; i++) {
           loop_width: for (HLS_SIZE_T j = 0; j < cols; j++) {
        #pragma HLS loop_flatten off
        #pragma HLS pipeline II=1
                        //dumping the input stream pixel in to a single
                        //24 bit variable (8 bits with 3 values)
                             src >> s;
                             //assuming that the format is HLS(hue
lightness saturation)
                             //leave hue and lightness unchanged
                             d.val[0] = s.val[0];
                             d.val[1] = s.val[1];
                             //based on the sat value implement the desired
saturation
                             //coefficient
                             switch(sat)
                             {
                                     case 0:
                                             d.val[2] = s.val[2];
                                             break;
                                     case 1:
                                             d.val[2] =
lut_s_n0_2[s.val[2]];
                                             break;
                                     case 2:
                                             d.val[2] = lut_s_0_2[s.val[2]];
                                             break;
                                     case 3:
                                             d.val[2] = lut_s_0_4[s.val[2]];
                                             break;
                                     case 4:
                                             d.val[2] = lut_s_0_6[s.val[2]];
                                             break;
                                     case 5:
                                             d.val[2] = lut_s_0_8[s.val[2]];
                                             break;
                                     case 6:
                                             d.val[2] = lut_s_1_0[s.val[2]];
                                             break;
                                     case 7:
                                             d.val[2] = lut_s_1_2[s.val[2]];
                                             break;
                                     default:
                                             d.val[2] = s.val[2];
                                             break;
                             }
                             //dump the pixel in to the destination matrix
                             dst << d;
                     }
             }

}

/**********************************************************************
********/
/**
```

```
 * This is the main function of the core, this function will be
synthesized
 * and packaged as an IP core. Its main purpose is to define the IP
ports
 * and busses while converting the RGB image in to HLS and calling
the sat_e
 * function.
 *
 * @param stream_in is the input Axi-Stream for video format
 * @param stream_out is the output Axi-Stream for video format
 * @param height parameter defines the height of the input image.
This parameter can
 *               be edited using the Axi-Light interface.
 * @param width parameter defines the width of the input image. This
parameter can
 *               be edited using the Axi-Light interface.
 * @param sat slects which saturation factor will be applied to the
image based on the
 *               predefined LUTs which are available. This parameter
can be edited using the
 *               Axi-Light interface.
 *
 * @return None
*********************************************************************
********/

void hls_saturation_enhance(stream_t &stream_in, stream_t
&stream_out, u_int16_t height, u_int16_t width, unsigned char sat)
{
#pragma HLS INTERFACE ap_ctrl_none port=return //disables the ctrl
interface
#pragma HLS DATAFLOW
#pragma HLS INTERFACE s_axilite port=sat
#pragma HLS INTERFACE s_axilite port=width
#pragma HLS INTERFACE s_axilite port=height
#pragma HLS INTERFACE axis register both port=stream_out
#pragma HLS INTERFACE axis register both port=stream_in

        u_int16_t rows = height;
        u_int16_t cols = width;

        rgb_img_t img0(rows, cols);
        rgb_img_t img1(rows, cols);
        rgb_img_t img2(rows, cols);
        rgb_img_t img3(rows, cols);

        //transforms stream to matrix
        hls::AXIvideo2Mat(stream_in, img0);
        //converts rgb to hls
        hls::CvtColor<HLS_RGB2HLS>(img0,img1);
        //apply saturation enhancement
        sat_e(img1, img2, sat);
        //convert hls to rgb
        hls::CvtColor<HLS_HLS2RGB>(img2,img3);
        //transform resulting matrix to output stream
        hls::Mat2AXIvideo(img3, stream_out);

}
```

Before being synthesized, the testbench file first needs to run to ensure that the code is correct.

hls_saturation_enhanche_test.cpp

```
#include "hls_saturation_enhanche.h"
#include "hls_opencv.h"

/*********************************************************************
********/
/**
 * Creates the lookup table for the chosen saturation coefficient.
The chosen
 * Implementation for enhancing saturation is dst.sat = src.sat +
s*src.sat;
 * in order to avoid over saturation and false colors a couple of
safeguards
 * have been implemented. If the the saturation value is greater then
0
 * a gray factor is calculated to that the unsaturated colors don't
get
 * saturated gray_factor = src.sat/255, also the upper limit must be
defined so that
 * the max value of the saturation can be remaped to 255 by using
 * var_interval = 255-src.sat. If the factor is less then 0 then no
safe guards
 * have been applied.
 * The function will list the results to the console
 *
 * @param s is a floating point variable which represents the
saturation factor
 *              which will be applied
 *
 * @return None
 *
 * @note
 *
**********************************************************************
********/
void sat_lut_creator(float s)
{
        unsigned char lut[256];

        printf("%f = {",s);
        //calculate the sat value for each possible input value
[0;255]
        for (int i = 0; i < 256; i++)
        {
                if(s >= 0)
                {
                        //var_interval = 255-src.sat
                        //gray_factor = src.sat/255,
                        //dst.sat = src.sat + gray_factor * src.sat *
var_interval
                        lut[i] = cv::saturate_cast<uchar>(i +
(s*(i/255.0)*(255.0-i)) );
                }
```

```
                else
                {
                        //dst.sat = src.sat + s*src.sat
                        lut[i] = cv::saturate_cast<uchar>(i+(s*i));
                }
                printf("%d,", lut[i]);
        }
        printf("};\n");

}

/**********************************************************************
********/
/**
* Main test bench
*
* @param None
* @return None
*
*
**********************************************************************
********/

int main()
{
        IplImage* src;
        IplImage* dst;
        stream_t src_axi, dst_axi;
src = cvLoadImage(INPUT_IMAGE);
dst = cvCreateImage(cvGetSize(src), src->depth, src->nChannels);
IplImage2AXIvideo(src, src_axi);

hls_contrast_stretch(src_axi, dst_axi, src->height, src->width, 15,
230);
        hls_saturation_enhance(src_axi, dst_axi, src->height, src-
>width, 5);

AXIvideo2IplImage(dst_axi, dst);
cvSaveImage(OUTPUT_IMAGE, dst);
cvReleaseImage(&src);
cvReleaseImage(&dst);

}
```

Simulation Result,

Figure 4.9 Saturation Enhance

4.4.1.1.4 Gamma Correction

hls_gamma_correction.h

```
#ifndef HLS_GAMMA_CORRECTION_H_
#define HLS_GAMMA_CORRECTION_H_

#include "hls_video.h"

#define MAX_WIDTH      1920
#define MAX_HEIGHT  1080

#define INPUT_IMAGE "fox.jpg"
#define OUTPUT_IMAGE "fox_out.jpg"

typedef ap_axiu<24,1,1,1> interface_t;
typedef ap_uint<3> interface_3_bits;
typedef hls::stream<interface_t> stream_t;

typedef hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC3> rgb_img_t;
typedef hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC1> single_img_t;
typedef hls::Scalar<3, unsigned char> rgb_pix_t;

void hls_gamma_correction(stream_t &stream_in, stream_t &stream_out,
unsigned char gamma, u_int16_t height, u_int16_t width);

#endif
```

hls_gamma_correction.cpp

```
//-- Purpose:
```

```
//-- This core has been written in order to perform a gamma
correction
//-- on a input image stream (RGB 24 bits, 8 bits/color) and output
it to a stream
//-- (RGB 24 bits, 8 bits/color). The correction is done by applying
the formula
//-- dst = (pow((src / 255.0), (1/g)) * 255.0) on each RGB component
independently.
//-- In order to reduce the resource consumption of the IP the actual
transformation
//-- has been implemented using lookup tables with predefined
coefficients.
//-- If new LUTs have to be implemented please refer to the
hls_gamma_correction_test.cpp
//-- test bench which contains the elaborated formula of the gamma
correction
//-- algorithm
//--
//---------------------------------------------------------------------
------------

#include "hls_gamma_correction.h"

/*************************** Constant Definitions
****************************/

unsigned char lut0_4[256] =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,2,
2,2,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5,5,6,6,6,6,7,7,7,7,8,8,8,9,9,9
,10,10,10,11,11,12,12,12,13,13,14,14,15,15,15,16,16,17,17,18,18,19,19
,20,20,21,22,22,23,23,24,25,25,26,26,27,28,28,29,30,30,31,32,33,33,34
,35,36,36,37,38,39,40,40,41,42,43,44,45,46,46,47,48,49,50,51,52,53,54
,55,56,57,58,59,60,61,62,63,64,65,67,68,69,70,71,72,73,75,76,77,78,80
,81,82,83,85,86,87,89,90,91,93,94,95,97,98,99,101,102,104,105,107,108
,110,111,113,114,116,117,119,121,122,124,125,127,129,130,132,134,135,
137,139,141,142,144,146,148,150,151,153,155,157,159,161,163,165,166,1
68,170,172,174,176,178,180,182,184,186,189,191,193,195,197,199,201,20
4,206,208,210,212,215,217,219,221,224,226,228,231,233,235,238,240,243
,245,248,250,253,255};
unsigned char lut0_2[256] =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,3,
3,3,3,3,3,3,4,4,4,4,4,5,5,5,5,5,6,6,6,6,7,7,7,8,8,8,8,9,9,9,10,10,11,
11,11,12,12,13,13,14,14,15,15,16,16,17,17,18,19,19,20,20,21,22,23,23,
24,25,26,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,45,46,
47,49,50,51,53,54,56,57,59,60,62,63,65,67,68,70,72,74,76,78,80,82,84,
86,88,90,92,94,97,99,101,104,106,109,111,114,116,119,122,125,128,130,
133,136,139,143,146,149,152,156,159,162,166,170,173,177,181,184,188,1
92,196,200,205,209,213,217,222,226,231,236,240,245,250,255};
unsigned char lut1_2[256] =
{0,3,4,6,8,10,11,13,14,16,17,19,20,21,23,24,25,27,28,29,31,32,33,34,3
6,37,38,39,40,42,43,44,45,46,48,49,50,51,52,53,54,56,57,58,59,60,61,6
2,63,65,66,67,68,69,70,71,72,73,74,75,76,77,78,80,81,82,83,84,85,86,8
7,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107
,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,
125,126,127,128,128,129,130,131,132,133,134,135,136,137,138,139,140,1
41,142,143,144,145,145,146,147,148,149,150,151,152,153,154,155,156,15
7,157,158,159,160,161,162,163,164,165,166,167,168,168,169,170,171,172
,173,174,175,176,177,177,178,179,180,181,182,183,184,185,185,186,187,
188,189,190,191,192,193,193,194,195,196,197,198,199,200,200,201,202,2
```

```
03,204,205,206,207,207,208,209,210,211,212,213,213,214,215,216,217,21
8,219,219,220,221,222,223,224,225,225,226,227,228,229,230,231,231,232
,233,234,235,236,237,237,238,239,240,241,242,242,243,244,245,246,247,
247,248,249,250,251,252,252,253,254,255};
unsigned char lut1_4[256] =
{0,5,8,11,13,15,18,20,22,23,25,27,29,30,32,34,35,37,38,40,41,43,44,46
,47,49,50,51,53,54,55,57,58,59,60,62,63,64,65,67,68,69,70,72,73,74,75
,76,77,78,80,81,82,83,84,85,86,87,89,90,91,92,93,94,95,96,97,98,99,10
0,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117
,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,133,
134,135,136,137,138,139,140,141,142,143,143,144,145,146,147,148,149,1
50,151,151,152,153,154,155,156,157,158,158,159,160,161,162,163,164,16
4,165,166,167,168,169,170,170,171,172,173,174,175,175,176,177,178,179
,180,180,181,182,183,184,184,185,186,187,188,188,189,190,191,192,192,
193,194,195,196,196,197,198,199,200,200,201,202,203,204,204,205,206,2
07,207,208,209,210,211,211,212,213,214,214,215,216,217,217,218,219,22
0,220,221,222,223,223,224,225,226,226,227,228,229,229,230,231,232,232
,233,234,235,235,236,237,238,238,239,240,241,241,242,243,243,244,245,
246,246,247,248,249,249,250,251,251,252,253,254,254,255};
unsigned char lut1_6[256] =
{0,8,12,16,19,22,24,27,29,32,34,36,38,40,42,43,45,47,49,50,52,54,55,5
7,58,60,61,63,64,66,67,68,70,71,72,74,75,76,78,79,80,81,83,84,85,86,8
7,89,90,91,92,93,94,96,97,98,99,100,101,102,103,104,105,106,107,109,1
10,111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,125,12
6,127,128,129,130,131,132,133,134,135,136,137,138,138,139,140,141,142
,143,144,145,146,146,147,148,149,150,151,152,152,153,154,155,156,157,
158,158,159,160,161,162,162,163,164,165,166,167,167,168,169,170,171,1
71,172,173,174,175,175,176,177,178,178,179,180,181,181,182,183,184,18
5,185,186,187,188,188,189,190,191,191,192,193,194,194,195,196,196,197
,198,199,199,200,201,202,202,203,204,204,205,206,207,207,208,209,209,
210,211,211,212,213,214,214,215,216,216,217,218,218,219,220,220,221,2
22,222,223,224,225,225,226,227,227,228,229,229,230,231,231,232,233,23
3,234,235,235,236,236,237,238,238,239,240,240,241,242,242,243,244,244
,245,246,246,247,247,248,249,249,250,251,251,252,252,253,254,254,255}
;
unsigned char lut1_8[256] =
{0,12,17,22,25,29,32,35,37,40,42,44,47,49,51,53,55,57,58,60,62,64,65,
67,69,70,72,73,75,76,78,79,80,82,83,85,86,87,89,90,91,92,94,95,96,97,
98,100,101,102,103,104,105,107,108,109,110,111,112,113,114,115,116,11
7,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134
,135,136,137,138,139,139,140,141,142,143,144,145,146,146,147,148,149,
150,151,152,152,153,154,155,156,157,157,158,159,160,161,161,162,163,1
64,165,165,166,167,168,169,169,170,171,172,172,173,174,175,175,176,17
7,178,178,179,180,181,181,182,183,183,184,185,186,186,187,188,188,189
,190,191,191,192,193,193,194,195,195,196,197,198,198,199,200,200,201,
202,202,203,204,204,205,206,206,207,208,208,209,209,210,211,211,212,2
13,213,214,215,215,216,217,217,218,218,219,220,220,221,222,222,223,22
3,224,225,225,226,226,227,228,228,229,230,230,231,231,232,233,233,234
,234,235,236,236,237,237,238,238,239,240,240,241,241,242,243,243,244,
244,245,245,246,247,247,248,248,249,249,250,251,251,252,252,253,253,2
54,254,255};
unsigned char lut2_0[256] =
{0,16,23,28,32,36,39,42,45,48,50,53,55,58,60,62,64,66,68,70,71,73,75,
77,78,80,81,83,84,86,87,89,90,92,93,94,96,97,98,100,101,102,103,105,1
06,107,108,109,111,112,113,114,115,116,117,118,119,121,122,123,124,12
5,126,127,128,129,130,131,132,133,134,135,135,136,137,138,139,140,141
,142,143,144,145,145,146,147,148,149,150,151,151,152,153,154,155,156,
156,157,158,159,160,160,161,162,163,164,164,165,166,167,167,168,169,1
70,170,171,172,173,173,174,175,176,176,177,178,179,179,180,181,181,18
2,183,183,184,185,186,186,187,188,188,189,190,190,191,192,192,193,194
,194,195,196,196,197,198,198,199,199,200,201,201,202,203,203,204,204,
```

```
205,206,206,207,208,208,209,209,210,211,211,212,212,213,214,214,215,2
15,216,217,217,218,218,219,220,220,221,221,222,222,223,224,224,225,22
5,226,226,227,228,228,229,229,230,230,231,231,232,233,233,234,234,235
,235,236,236,237,237,238,238,239,240,240,241,241,242,242,243,243,244,
244,245,245,246,246,247,247,248,248,249,249,250,250,251,251,252,252,2
53,253,254,254,255};
unsigned char lut2_2[256] =
{0,21,28,34,39,43,46,50,53,56,59,61,64,66,68,70,72,74,76,78,80,82,84,
85,87,89,90,92,93,95,96,98,99,101,102,103,105,106,107,109,110,111,112
,114,115,116,117,118,119,120,122,123,124,125,126,127,128,129,130,131,
132,133,134,135,136,137,138,139,140,141,142,143,144,144,145,146,147,1
48,149,150,151,151,152,153,154,155,156,156,157,158,159,160,160,161,16
2,163,164,164,165,166,167,167,168,169,170,170,171,172,173,173,174,175
,175,176,177,178,178,179,180,180,181,182,182,183,184,184,185,186,186,
187,188,188,189,190,190,191,192,192,193,194,194,195,195,196,197,197,1
98,199,199,200,200,201,202,202,203,203,204,205,205,206,206,207,207,20
8,209,209,210,210,211,212,212,213,213,214,214,215,215,216,217,217,218
,218,219,219,220,220,221,221,222,223,223,224,224,225,225,226,226,227,
227,228,228,229,229,230,230,231,231,232,232,233,233,234,234,235,235,2
36,236,237,237,238,238,239,239,240,240,241,241,242,242,243,243,244,24
4,245,245,246,246,247,247,248,248,249,249,249,250,250,251,251,252,252
,253,253,254,254,255,255};

/*********************************************************************
********/
/**
 * Applies the gamma correction coefficients to the input matrix
HLS_8UC3,
 * form the desired LUT. the selection is done using sat variable as
a simple
 * switch between LUTs
 *
 * @param src is the input matrix on 3 channels with each color on 8
bits
 * @param dst is the output matrix on 3 channels with each color on 8
bits
 * @param g is a 8 bit parameter which selects between gamma factors
 *              which should be applied.
 *              sat == 0 =>  dst.sat = src.sat
 *              sat == 1 =>  dst = (pow((src / 255.0), (1/0.4)) *
255.0)
 *              sat == 2 =>  dst = (pow((src / 255.0), (1/0.2)) *
255.0)
 *              sat == 3 =>  dst = (pow((src / 255.0), (1/1.2)) *
255.0)
 *              sat == 4 =>  dst = (pow((src / 255.0), (1/1.4)) *
255.0)
 *              sat == 5 =>  dst = (pow((src / 255.0), (1/1.6)) *
255.0)
 *              sat == 6 =>  dst = (pow((src / 255.0), (1/1.8)) *
255.0)
 *              sat == 7 =>  dst = (pow((src / 255.0), (1/2.0)) *
255.0)
 *              sat == 8 =>  dst = (pow((src / 255.0), (1/2.2)) *
255.0)
 *
 * @return None
 *
 * @note The pragmas are part of the HLS optimization flow and should
be
```

```
 *              researched before using them in other functions.
Please refer to
 *              UG902 of your current Vivado version
 *
**********************************************************************
********/
void gamma_c(rgb_img_t &src, rgb_img_t &dst, unsigned char g)
{
#pragma HLS INLINE

        HLS_SIZE_T rows = dst.rows;
        HLS_SIZE_T cols = dst.cols;

        hls::Scalar<HLS_MAT_CN(HLS_8UC3), HLS_TNAME(HLS_8UC3)> s;
        hls::Scalar<HLS_MAT_CN(HLS_8UC3), HLS_TNAME(HLS_8UC3)> d;

        //loop trough the matrix
        loop_height: for (HLS_SIZE_T i = 0; i < rows; i++) {
           loop_width: for (HLS_SIZE_T j = 0; j < cols; j++) {
        #pragma HLS loop_flatten off
        #pragma HLS pipeline II=1
                          //dumping the input stream pixel in to a
single
                          //24 bit variable (8 bits with 3 values)
                          src >> s;
                          //loop trough the channels and apply the
gamma coefficients
                          loop_channels: for (HLS_CHANNEL_T k = 0; k
< HLS_MAT_CN(HLS_8UC3); k++) {
                               switch(g)
                               {
                                       case 0:
                                               d.val[k] = s.val[k];
                                               break;
                                       case 1:
                                               d.val[k] =
lut0_4[s.val[k]];
                                               break;
                                       case 2:
                                               d.val[k] =
lut0_2[s.val[k]];
                                               break;
                                       case 3:
                                               d.val[k] =
lut1_2[s.val[k]];
                                               break;
                                       case 4:
                                               d.val[k] =
lut1_4[s.val[k]];
                                               break;
                                       case 5:
                                               d.val[k] =
lut1_6[s.val[k]];
                                               break;
                                       case 6:
                                               d.val[k] =
lut1_8[s.val[k]];
                                               break;
                                       case 7:
                                               d.val[k] =
lut2_0[s.val[k]];
```

```
                                                break;
                                        case 8:
                                                d.val[k] =
lut2_2[s.val[k]];
                                                break;
                                        default:
                                                d.val[k] = s.val[k];
                                                break;
                                }
                        }
                        //dump the pixel in to the destination
matrix
                        dst << d;
                }
        }

}

/*********************************************************************
********/
/**
 * This is the main function of the core, this function will be
synthesized
 * and packaged as an IP core. Its main purpose is to define the IP
ports
 * and busses while applying the gamma correction factor on the image
 *
 * @param stream_in is the input Axi-Stream for video format
 * @param stream_out is the output Axi-Stream for video format
 * @param gamma selects which gamma factor will be applied to the
image based on the
 *              predefined LUTs which are available. This parameter
can be edited using the
 *              Axi-Light interface.
 * @param height parameter defines the height of the input image.
This parameter can
 *              be edited using the Axi-Light interface.
 * @param width parameter defines the width of the input image. This
parameter can
 *              be edited using the Axi-Light interface.
 *
 * @return None
 *
 * @note The pragmas are part of the HLS optimization flow and should
be
 *              researched before using them in other functions.
Please refer to
 *              UG902 of your current Vivado version.
 *
**********************************************************************
********/
void hls_gamma_correction(stream_t &stream_in, stream_t &stream_out,
unsigned char gamma, u_int16_t height, u_int16_t width)
{
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS DATAFLOW
#pragma HLS INTERFACE s_axilite port=width
#pragma HLS INTERFACE s_axilite port=height
#pragma HLS INTERFACE s_axilite port=gamma
#pragma HLS INTERFACE axis register both port=stream_in
#pragma HLS INTERFACE axis register both port=stream_out
```

```
        u_int16_t rows = height;
        u_int16_t cols = width;

        rgb_img_t img0(rows, cols);
        rgb_img_t img3(rows, cols);

        //transforms stream to matrix
        hls::AXIvideo2Mat(stream_in, img0);
        //apply gamma correction
        gamma_c(img0, img3, gamma);
        //transform resulting matrix to output stream
        hls::Mat2AXIvideo(img3, stream_out);

}
```

hls_gamma_correction_test.cpp

```
*********************************************************************
*******/
/**
 * Creates the lookup table for the chosen gamma coefficient. The
function
 * applied for gamma correction is dst = (pow((src / 255.0), (1/g)) *
255.0)
 * The function will list the results to the console
 *
 * @param g is a floating point variable which represents the gamma
factor
 *               which will be applied
 *
 * @return None
 *
 * @note
 *
*********************************************************************
********/
void gamma_val(float g)
{
        unsigned char lut[256];

        printf("%f = {",g);
        for (int i = 0; i < 256; i++)
        {
                lut[i] = cv::saturate_cast<uchar>(pow((float)(i /
255.0), (1/g)) * 255.0f);
                printf("%d,", lut[i]);
        }
        printf("};\n");

}

/********************************************************************
********/
/**
* Main test bench
*
* @param None
* @return None
```

```
*
*
**********************************************************************
********/
int main()
{

        IplImage* src;
        IplImage* dst;
        stream_t src_axi, dst_axi;
src = cvLoadImage(INPUT_IMAGE);
dst = cvCreateImage(cvGetSize(src), src->depth, src->nChannels);
IplImage2AXIvideo(src, src_axi);

        hls_gamma_correction(src_axi, dst_axi, 5, src->height, src-
>width);


        AXIvideo2IplImage(dst_axi, dst);
cvSaveImage(OUTPUT_IMAGE, dst);
cvReleaseImage(&src);
cvReleaseImage(&dst);

        return 0;
}
```

Simulation Result,

Figure 4.10 Gamma Correction

4.4.2 Integration of Image Processing IP cores

The IPs thus generated were then used in the existing project to enhance the live input stream from the image sensor to produce a high-quality camera output.

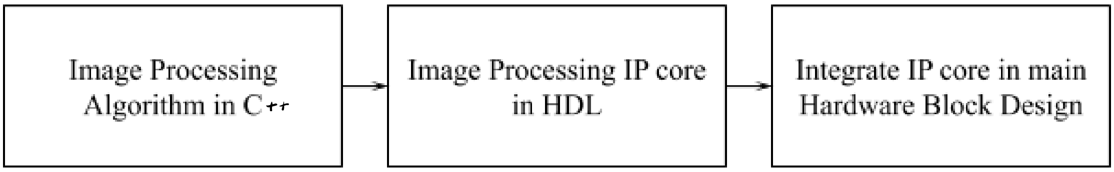

Figure 4.11 Design flow to generate image processing IPs

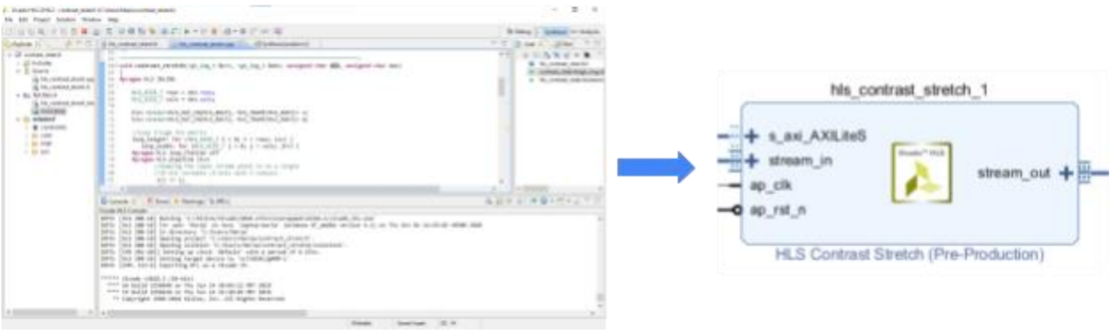

Figure 4.12 Conversion of C++ code to RTL

Initially, we wanted to adapt the camera's output according to the environment but were unable to access the required sensors due to the COVID-19 pandemic.

So, to mimic the different outputs that can be generated switches are used. Each switch implements a different image processing algorithm on the live input feed.

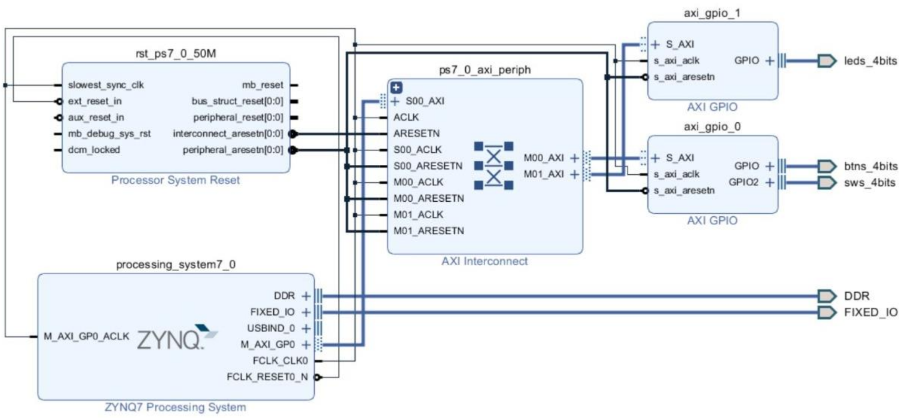

Figure 4.13 Switch Test Block Design

We have used two switch IP cores in our design block. The first switch is being used as a 1x4 mu and the second switch as a 4x1 demux.

The input to the switch 1 IP core is the live stream of our image sensor and the outputs are the IPs that we have generated through Vivado HLS. Output1 is directly forwarding the live stream. The output2 is connected to the gamma correction IP, so, in case of low light one, can turn on switch1 on the FPGA board to have a better-quality bright image. The output3 is connected to the contrast stretching IP so that in case of a situation where there is low contrast one can turn on switch 2 to have a better-contrasted image and the ouput3 is connected to the saturation enhancement IP so one can turn on switch 3 to improve the saturation of the live feed.

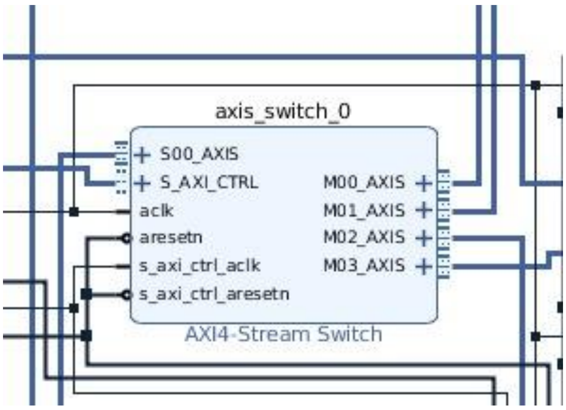

Figure 4.14 Switch 0 IP Core

All these IPs are then in turn connected to switch 2 IP core, which then produces a single output which is forwarded as the video output.

We have tried to use these switches as enablers here. Previously our proposed approach was to use sensors to enable the required IP according to the environment but considering the COVID-19 pandemic situation, we had to change our strategy and now we are achieving this same functionality using switches.

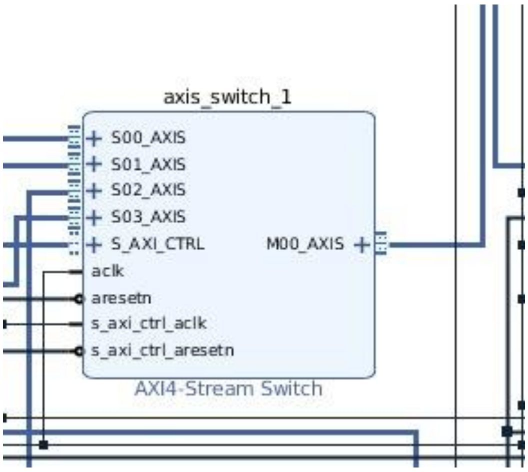

Figure 4.15 Switch 1 IP Core

Code for controlling switches,

AXI_SWITCH.h

```
#ifndef SRC_AXIS_SWITCH_AXI_SWITCH_H_
#define SRC_AXIS_SWITCH_AXI_SWITCH_H_

#include <stdexcept>

#include "xaxis_switch.h"

#define STRINGIZE(x) STRINGIZE2(x)
#define STRINGIZE2(x) #x
#define LINE_STRING STRINGIZE(__LINE__)

class AXI_SWITCH {
public:
        AXI_SWITCH(uint16_t dev_id) :
                drv_inst_()
        {
            XAxis_Switch_Config *Config;
            XStatus Status;

            Config = XAxisScr_LookupConfig(dev_id);
                if (NULL == Config) {
                        throw std::runtime_error(__FILE__ ":"
LINE_STRING);
                }

                Status = XAxisScr_CfgInitialize(&drv_inst_, Config,
```

```
                                                Config-
>BaseAddress);
                if (Status != XST_SUCCESS) {
                        throw std::runtime_error(__FILE__ ":"
LINE_STRING);
                }
        }

        void changeSlaveChannel(uint8_t slave_ch)
        {
                XAxisScr_RegUpdateDisable(&drv_inst_);
                XAxisScr_MiPortDisableAll(&drv_inst_);
                XAxisScr_MiPortEnable(&drv_inst_, 0, slave_ch);
                XAxisScr_RegUpdateEnable(&drv_inst_);
        }

        void changeMasterChannel(uint8_t master_ch)
        {
                XAxisScr_RegUpdateDisable(&drv_inst_);
                XAxisScr_MiPortDisableAll(&drv_inst_);
                XAxisScr_MiPortEnable(&drv_inst_, master_ch, 0);
                XAxisScr_RegUpdateEnable(&drv_inst_);
        }

private:
        XAxis_Switch drv_inst_;
};

#endif
```

SWITCHGPIO.h

```
#ifndef SRC_AXIS_SWITCH_SW_GPIO_H_
#define SRC_AXIS_SWITCH_SW_GPIO_H_

#include "xgpio.h"
#include <stdexcept>

#define STRINGIZE(x) STRINGIZE2(x)
#define STRINGIZE2(x) #x
#define LINE_STRING STRINGIZE(__LINE__)

class SWITCH_GPIO {
public:
        SWITCH_GPIO(uint16_t dev_id) :
                drv_inst_()
        {
                XStatus Status;
                Status = XGpio_Initialize(&drv_inst_, dev_id);
                if (Status != XST_SUCCESS) {
                        throw std::runtime_error(__FILE__ ":"
LINE_STRING);
                }
                XGpio_SetDataDirection(&drv_inst_, 1, 1);
        }
        uint32_t readSwitchGpio()
        {
                return XGpio_DiscreteRead(&drv_inst_, 1);
```

```
	}
private:
	XGpio drv_inst_;
};

#endif
```

SWITCH_CTL.h

```
#ifndef SRC_AXIS_SWITCH_SWITCH_CTL_H_
#define SRC_AXIS_SWITCH_SWITCH_CTL_H_

#include "AXI_SWITCH.h"
#include "SW_GPIO.h"

class SWITCH_CTL {
public:
	SWITCH_CTL(AXI_SWITCH& src_axi_sw, AXI_SWITCH& dst_axi_sw,
SWITCH_GPIO& gpio, uint16_t max_nr_ch, uint16_t default_pos):
		src_axi_sw_(src_axi_sw), dst_axi_sw_(dst_axi_sw),
gpio_(gpio), nr_of_ch_(max_nr_ch-1), current_pos_(default_pos)
	{
		setChannel();
	}
	void setChannel()
	{
		uint16_t sw;

		sw = gpio_.readSwitchGpio();
		if(sw > nr_of_ch_)
		{
			sw = 0;
		}

		src_axi_sw_.changeMasterChannel(sw);
		dst_axi_sw_.changeSlaveChannel(sw);
	}
	void updateChannel()
	{
		uint16_t sw;

		sw = gpio_.readSwitchGpio();
		if(sw > nr_of_ch_)
		{
			sw = 0;
		}

		if(sw != current_pos_)
		{
			current_pos_ = sw;
			setChannel();
		}
	}
private:
	AXI_SWITCH& src_axi_sw_;
```

```
        AXI_SWITCH& dst_axi_sw_;
        SWITCH_GPIO& gpio_;
        uint16_t nr_of_ch_;
        uint16_t current_pos_;
};

#endif
```

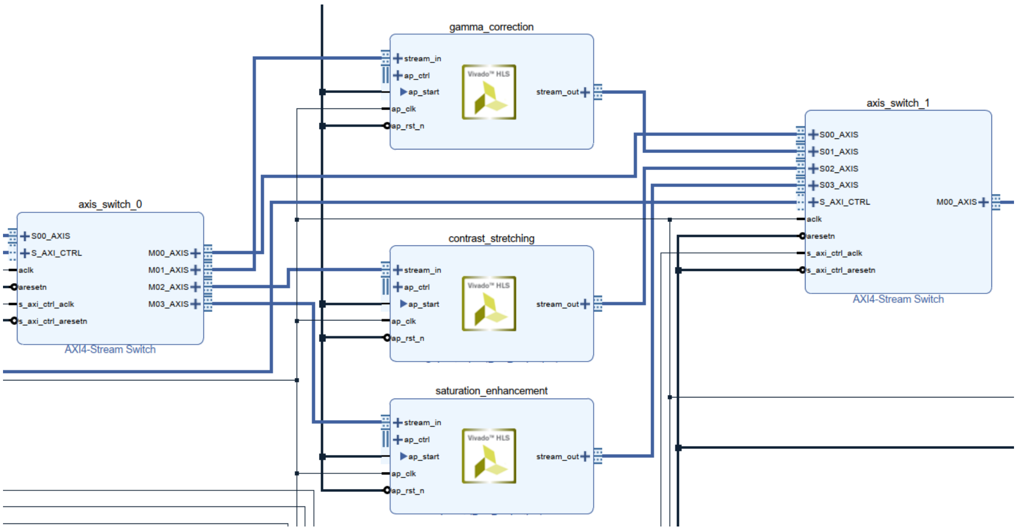

Figure 4.16 Image Processing IPs connected to the switches

4.5. Final Block Design

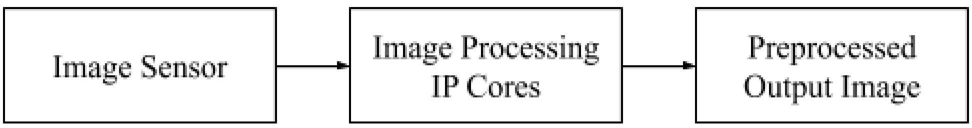

Figure 4.17 Design flow for the implemented solution for the Software-Defined Smart Camera

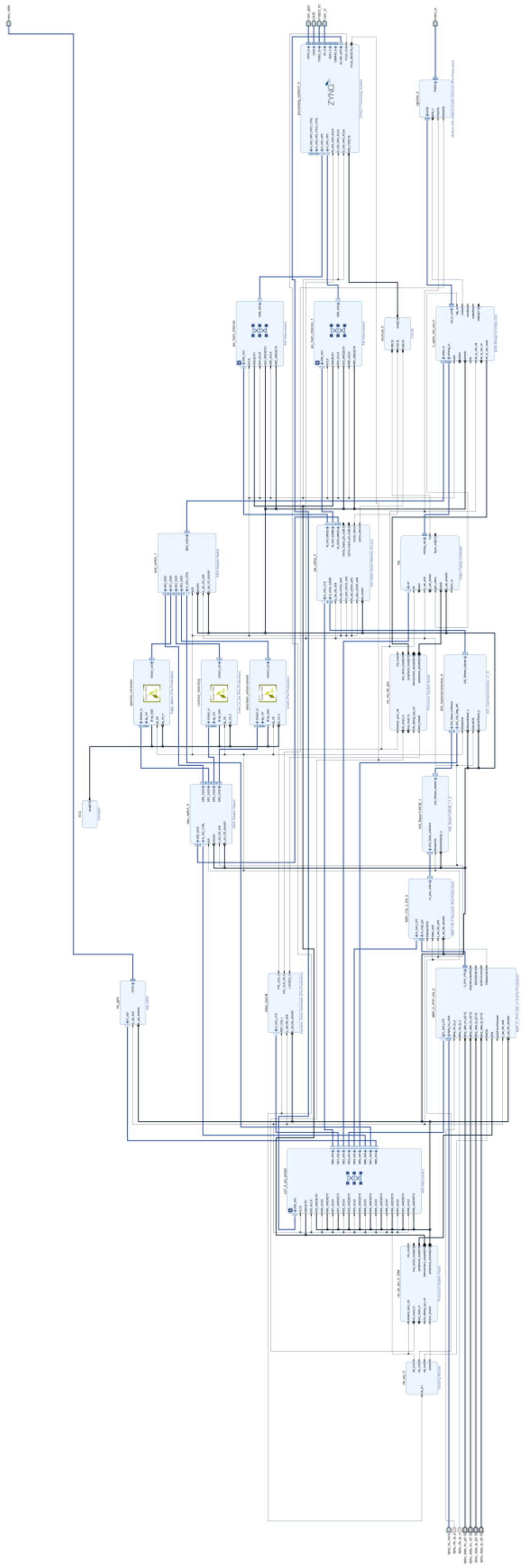

Figure 4.18 Final Block Design of Software-Defined Smart Camera

CHAPTER 5

Outcome

A remotely deployable software-defined smart camera solution that can adjust to different conditions, producing high-quality output video stream with minimal processing time, thus lessening the load on server-side applications that otherwise would need to preprocess the camera input.

CHAPTER 6

Conclusion

The implemented solution provides a way to realize a design on the FPGA board to preprocess the live video stream being captured in real-time. The proposed design for this can be extended with further image processing blocks and environment adaptable capturing features for further applications as well.

Appendix A: Program to Control the Image Sensor

```
#include "xparameters.h"

#include "platform/platform.h"
#include "ov5640/OV5640.h"
#include "ov5640/ScuGicInterruptController.h"
#include "ov5640/PS_GPIO.h"
#include "ov5640/AXI_VDMA.h"
#include "ov5640/PS_IIC.h"
#include "axis_switch/SWITCH_CTL.h"

#include "MIPI_D_PHY_RX.h"
#include "MIPI_CSI_2_RX.h"

#define IRPT_CTL_DEVID       XPAR_PS7_SCUGIC_0_DEVICE_ID
#define GPIO_DEVID           XPAR_PS7_GPIO_0_DEVICE_ID
#define GPIO_IRPT_ID             XPAR_PS7_GPIO_0_INTR
#define CAM_I2C_DEVID        XPAR_PS7_I2C_0_DEVICE_ID
#define CAM_I2C_IRPT_ID      XPAR_PS7_I2C_0_INTR
#define VDMA_DEVID           XPAR_AXIVDMA_0_DEVICE_ID
#define VDMA_MM2S_IRPT_ID    XPAR_FABRIC_AXI_VDMA_0_MM2S_INTROUT_INTR
#define VDMA_S2MM_IRPT_ID    XPAR_FABRIC_AXI_VDMA_0_S2MM_INTROUT_INTR
#define CAM_I2C_SCLK_RATE    100000
#define SRC_AXIS_SW_DEVID    XPAR_AXIS_SWITCH_0_DEVICE_ID
#define DST_AXIS_SW_DEVID    XPAR_AXIS_SWITCH_1_DEVICE_ID
#define GPIO_SW_DEVID        XPAR_AXI_SW_DEVICE_ID

#define DDR_BASE_ADDR        XPAR_DDR_MEM_BASEADDR
#define MEM_BASE_ADDR        (DDR_BASE_ADDR + 0x0A000000)

#define GAMMA_BASE_ADDR      XPAR_AXI_GAMMACORRECTION_0_BASEADDR

using namespace digilent;

void pipeline_mode_change(AXI_VDMA<ScuGicInterruptController>& vdma_d
river, OV5640& cam, VideoOutput& vid, Resolution res, OV5640_cfg::mod
e_t mode)
{
    //Bring up input pipeline back-to-front
    {
        vdma_driver.resetWrite();
        MIPI_CSI_2_RX_mWriteReg(XPAR_MIPI_CSI_2_RX_0_S_AXI_LITE_BASEA
DDR, CR_OFFSET, (CR_RESET_MASK & ~CR_ENABLE_MASK));
        MIPI_D_PHY_RX_mWriteReg(XPAR_MIPI_D_PHY_RX_0_S_AXI_LITE_BASEA
DDR, CR_OFFSET, (CR_RESET_MASK & ~CR_ENABLE_MASK));
        cam.reset();
    }

    {
        vdma_driver.configureWrite(timing[static_cast<int>(res)].h_ac
tive, timing[static_cast<int>(res)].v_active);
        Xil_Out32(GAMMA_BASE_ADDR, 3); // Set Gamma correction factor
 to 1/1.8
        //TODO CSI-2, D-PHY config here
        cam.init();
    }

    {
        vdma_driver.enableWrite();
```

```
        MIPI_CSI_2_RX_mWriteReg(XPAR_MIPI_CSI_2_RX_0_S_AXI_LITE_BASEA
DDR, CR_OFFSET, CR_ENABLE_MASK);
        MIPI_D_PHY_RX_mWriteReg(XPAR_MIPI_D_PHY_RX_0_S_AXI_LITE_BASEA
DDR, CR_OFFSET, CR_ENABLE_MASK);
        cam.set_mode(mode);
        cam.set_awb(OV5640_cfg::awb_t::AWB_ADVANCED);
    }

    //Bring up output pipeline back-to-front
    {
        vid.reset();
        vdma_driver.resetRead();
    }

    {
        vid.configure(res);
        vdma_driver.configureRead(timing[static_cast<int>(res)].h_act
ive, timing[static_cast<int>(res)].v_active);
    }

    {
        vid.enable();
        vdma_driver.enableRead();
    }
}

int main()
{
    init_platform();

    ScuGicInterruptController irpt_ctl(IRPT_CTL_DEVID);
    PS_GPIO<ScuGicInterruptController> gpio_driver(GPIO_DEVID, irpt_c
tl, GPIO_IRPT_ID);
    PS_IIC<ScuGicInterruptController> iic_driver(CAM_I2C_DEVID, irpt_
ctl, CAM_I2C_IRPT_ID, 100000);
    SWITCH_GPIO sw_gpio(GPIO_SW_DEVID);
    AXI_SWITCH src_switch(SRC_AXIS_SW_DEVID);
    AXI_SWITCH dst_switch(DST_AXIS_SW_DEVID);

    SWITCH_CTL axis_switch_ctl(src_switch, dst_switch, sw_gpio, 3, 0)
;

    OV5640 cam(iic_driver, gpio_driver);
    AXI_VDMA<ScuGicInterruptController> vdma_driver(VDMA_DEVID, MEM_B
ASE_ADDR, irpt_ctl,
            VDMA_MM2S_IRPT_ID,
            VDMA_S2MM_IRPT_ID);
    VideoOutput vid(XPAR_VTC_0_DEVICE_ID, XPAR_VIDEO_DYNCLK_DEVICE_ID
);

    pipeline_mode_change(vdma_driver, cam, vid, Resolution::R1280_720
_60_PP, OV5640_cfg::mode_t::MODE_720P_1280_720_60fps);

    xil_printf("Video init done.\r\n");

    // Liquid lens control
    uint8_t read_char0 = 0;
    uint8_t read_char1 = 0;
    uint8_t read_char2 = 0;
    uint8_t read_char4 = 0;
    uint8_t read_char5 = 0;
```

```
    uint16_t reg_addr;
    uint8_t reg_value;
    uint32_t color_values;

    while (1) {

        axis_switch_ctl.updateChannel();

        xil_printf("\r\n\r\n\r\nPcam 5C MAIN OPTIONS\r\n");
        xil_printf("\r\nPlease press the key corresponding to the des
ired option:");
        xil_printf("\r\n  a. Change Resolution");
        xil_printf("\r\n  b. Change Liquid Lens Focus");
        xil_printf("\r\n  d. Change Image Format (Raw or RGB)");
        xil_printf("\r\n  e. Write a Register Inside the Image Sensor
");
        xil_printf("\r\n  f. Read a Register Inside the Image Sensor"
);
        xil_printf("\r\n  g. Change Gamma Correction Factor Value");
        xil_printf("\r\n  h. Change AWB Settings\r\n\r\n");

        read_char0 = getchar();
        getchar();
        xil_printf("Read: %d\r\n", read_char0);

        switch(read_char0) {

        case 'a':
            xil_printf("\r\n  Please press the key corresponding to t
he desired resolution:");
            xil_printf("\r\n    1. 1280 x 720, 60fps");
            xil_printf("\r\n    2. 1920 x 1080, 15fps");
            xil_printf("\r\n    3. 1920 x 1080, 30fps");
            read_char1 = getchar();
            getchar();
            xil_printf("\r\nRead: %d", read_char1);
            switch(read_char1) {
            case '1':
                pipeline_mode_change(vdma_driver, cam, vid, Resolutio
n::R1280_720_60_PP, OV5640_cfg::mode_t::MODE_720P_1280_720_60fps);
                xil_printf("Resolution change done.\r\n");
                break;
            case '2':
                pipeline_mode_change(vdma_driver, cam, vid, Resolutio
n::R1920_1080_60_PP, OV5640_cfg::mode_t::MODE_1080P_1920_1080_15fps);
                xil_printf("Resolution change done.\r\n");
                break;
            case '3':
                pipeline_mode_change(vdma_driver, cam, vid, Resolutio
n::R1920_1080_60_PP, OV5640_cfg::mode_t::MODE_1080P_1920_1080_30fps);
                xil_printf("Resolution change done.\r\n");
                break;
            default:
                xil_printf("\r\n  Selection is outside the available
options! Please retry...");
            }
            break;

        case 'b':
            xil_printf("\r\n\r\nPlease enter value of liquid lens reg
ister, in hex, with small letters: 0x");
```

```
            //A, B, C,..., F need to be entered with small letters
            while (read_char1 < 48) {
                read_char1 = getchar();
            }
            while (read_char2 < 48) {
                read_char2 = getchar();
            }
            getchar();
            // If character is a digit, convert from ASCII code to a 
digit between 0 and 9
            if (read_char1 <= 57) {
                read_char1 -= 48;
            }
            // If character is a letter, convert ASCII code to a numb
er between 10 and 15
            else {
                read_char1 -= 87;
            }
            // If character is a digit, convert from ASCII code to a 
digit between 0 and 9
            if (read_char2 <= 57) {
                read_char2 -= 48;
            }
            // If character is a letter, convert ASCII code to a numb
er between 10 and 15
            else {
                read_char2 -= 87;
            }
            cam.writeRegLiquid((uint8_t) (16*read_char1 + read_char2)
);
            xil_printf("\r\nWrote to liquid lens controller: %x", (ui
nt8_t) (16*read_char1 + read_char2));
            break;

        case 'd':
            xil_printf("\r\n  Please press the key corresponding to t
he desired setting:");
            xil_printf("\r\n    1. Select image format to be RGB, out
put still Raw");
            xil_printf("\r\n    2. Select image format & output to bo
th be Raw");
            read_char1 = getchar();
            getchar();
            xil_printf("\r\nRead: %d", read_char1);
            switch(read_char1) {
            case '1':
                cam.set_isp_format(OV5640_cfg::isp_format_t::ISP_RGB)
;
                xil_printf("Settings change done.\r\n");
                break;
            case '2':
                cam.set_isp_format(OV5640_cfg::isp_format_t::ISP_RAW)
;
                xil_printf("Settings change done.\r\n");
                break;
            default:
                xil_printf("\r\n  Selection is outside the available 
options! Please retry...");
            }
            break;
```

```
        case 'e':
            xil_printf("\r\nPlease enter address of image sensor regi
ster, in hex, with small letters: \r\n");
            //A, B, C,..., F need to be entered with small letters
            while (read_char1 < 48) {
                read_char1 = getchar();
            }
            while (read_char2 < 48) {
                read_char2 = getchar();
            }
            while (read_char4 < 48) {
                read_char4 = getchar();
            }
            while (read_char5 < 48) {
                read_char5 = getchar();
            }
            getchar();
            // If character is a digit, convert from ASCII code to a
digit between 0 and 9
            if (read_char1 <= 57) {
                read_char1 -= 48;
            }
            // If character is a letter, convert ASCII code to a numb
er between 10 and 15
            else {
                read_char1 -= 87;
            }
            // If character is a digit, convert from ASCII code to a
digit between 0 and 9
            if (read_char2 <= 57) {
                read_char2 -= 48;
            }
            // If character is a letter, convert ASCII code to a numb
er between 10 and 15
            else {
                read_char2 -= 87;
            }
            // If character is a digit, convert from ASCII code to a
digit between 0 and 9
            if (read_char4 <= 57) {
                read_char4 -= 48;
            }
            // If character is a letter, convert ASCII code to a numb
er between 10 and 15
            else {
                read_char4 -= 87;
            }
            // If character is a digit, convert from ASCII code to a
digit between 0 and 9
            if (read_char5 <= 57) {
                read_char5 -= 48;
            }
            // If character is a letter, convert ASCII code to a numb
er between 10 and 15
            else {
                read_char5 -= 87;
            }
            reg_addr = 16*(16*(16*read_char1 + read_char2)+read_char4
)+read_char5;
            xil_printf("Desired Register Address: %x\r\n", reg_addr);
```

```
            read_char1 = 0;
            read_char2 = 0;
            xil_printf("\r\nPlease enter value of image sensor regist
er, in hex, with small letters: \r\n");
            //A, B, C,..., F need to be entered with small letters
            while (read_char1 < 48) {
                read_char1 = getchar();
            }
            while (read_char2 < 48) {
                read_char2 = getchar();
            }
            getchar();
            // If character is a digit, convert from ASCII code to a
digit between 0 and 9
            if (read_char1 <= 57) {
                read_char1 -= 48;
            }
            // If character is a letter, convert ASCII code to a numb
er between 10 and 15
            else {
                read_char1 -= 87;
            }
            // If character is a digit, convert from ASCII code to a
digit between 0 and 9
            if (read_char2 <= 57) {
                read_char2 -= 48;
            }
            // If character is a letter, convert ASCII code to a numb
er between 10 and 15
            else {
                read_char2 -= 87;
            }
            reg_value = 16*read_char1 + read_char2;
            xil_printf("Desired Register Value: %x\r\n", reg_value);
            cam.writeReg(reg_addr, reg_value);
            xil_printf("Register write done.\r\n");

            break;

        case 'f':
            xil_printf("Please enter address of image sensor register
, in hex, with small letters: \r\n");
            //A, B, C,..., F need to be entered with small letters
            while (read_char1 < 48) {
                read_char1 = getchar();
            }
            while (read_char2 < 48) {
                read_char2 = getchar();
            }
            while (read_char4 < 48) {
                read_char4 = getchar();
            }
            while (read_char5 < 48) {
                read_char5 = getchar();
            }
            getchar();
            // If character is a digit, convert from ASCII code to a
digit between 0 and 9
            if (read_char1 <= 57) {
                read_char1 -= 48;
            }
```

```
            // If character is a letter, convert ASCII code to a numb
er between 10 and 15
            else {
                read_char1 -= 87;
            }
            // If character is a digit, convert from ASCII code to a
digit between 0 and 9
            if (read_char2 <= 57) {
                read_char2 -= 48;
            }
            // If character is a letter, convert ASCII code to a numb
er between 10 and 15
            else {
                read_char2 -= 87;
            }
            // If character is a digit, convert from ASCII code to a
digit between 0 and 9
            if (read_char4 <= 57) {
                read_char4 -= 48;
            }
            // If character is a letter, convert ASCII code to a numb
er between 10 and 15
            else {
                read_char4 -= 87;
            }
            // If character is a digit, convert from ASCII code to a
digit between 0 and 9
            if (read_char5 <= 57) {
                read_char5 -= 48;
            }
            // If character is a letter, convert ASCII code to a numb
er between 10 and 15
            else {
                read_char5 -= 87;
            }
            reg_addr = 16*(16*(16*read_char1 + read_char2)+read_char4
)+read_char5;
            xil_printf("Desired Register Address: %x\r\n", reg_addr);

            cam.readReg(reg_addr, reg_value);
            xil_printf("Value of Desired Register: %x\r\n", reg_value
);

            break;

        case 'g':
            xil_printf("  Please press the key corresponding to the d
esired Gamma factor:\r\n");
            xil_printf("    1. Gamma Factor = 1\r\n");
            xil_printf("    2. Gamma Factor = 1/1.2\r\n");
            xil_printf("    3. Gamma Factor = 1/1.5\r\n");
            xil_printf("    4. Gamma Factor = 1/1.8\r\n");
            xil_printf("    5. Gamma Factor = 1/2.2\r\n");
            read_char1 = getchar();
            getchar();
            xil_printf("Read: %d\r\n", read_char1);
            // Convert from ASCII to numeric
            read_char1 = read_char1 - 48;
            if ((read_char1 > 0) && (read_char1 < 6)) {
                Xil_Out32(GAMMA_BASE_ADDR, read_char1-1);
                xil_printf("Gamma value changed to 1.\r\n");
```

```
            }
            else {
                xil_printf("  Selection is outside the available opti
ons! Please retry...\r\n");
            }
            break;

        case 'h':
            xil_printf("  Please press the key corresponding to the d
esired AWB change:\r\n");
            xil_printf("    1. Enable Advanced AWB\r\n");
            xil_printf("    2. Enable Simple AWB\r\n");
            xil_printf("    3. Disable AWB\r\n");
            read_char1 = getchar();
            getchar();
            xil_printf("Read: %d\r\n", read_char1);
            switch(read_char1) {
            case '1':
                cam.set_awb(OV5640_cfg::awb_t::AWB_ADVANCED);
                xil_printf("Enabled Advanced AWB\r\n");
                break;
            case '2':
                cam.set_awb(OV5640_cfg::awb_t::AWB_SIMPLE);
                xil_printf("Enabled Simple AWB\r\n");
                break;
            case '3':
                cam.set_awb(OV5640_cfg::awb_t::AWB_DISABLED);
                xil_printf("Disabled AWB\r\n");
                break;
            default:
                xil_printf("  Selection is outside the available opti
ons! Please retry...\r\n");
            }
            break;

        default:
            xil_printf("  Selection is outside the available options!
 Please retry...\r\n");
        }

        read_char1 = 0;
        read_char2 = 0;
        read_char4 = 0;
        read_char5 = 0;
    }

    cleanup_platform();

    return 0;
}
```

REFERENCES

[1] Bansal, M., Kumar, M. & Kumar, M. 2D Object Recognition Techniques: State-of-the-Art Work. *Arch Computat Methods Eng* (2020). https://doi.org/10.1007/s11831-020-09409-1

[2] Ahmed, E., Ahmed, A., Yaqoob, I., Shuja, J., Gani, A., Imran, M. and Shoaib, M. (2017). Bringing Computation Closer toward the User Network: Is Edge Computing the Solution?. *IEEE Communications Magazine*, 55(11), pp.138-144.

[3] Z. Song, J. Cheng, A. Chauhan and E. Tilevich, "Pushing Participatory Sensing Further to the Edge," *2019 IEEE International Conference on Edge Computing (EDGE),* Milan, Italy, 2019, pp. 24-26.

[4] M. S. AZZAZ, A. MAALI, R. KAIBOU, I. KAKOUCHE, M. SAAD and H. HAMIL, "FPGA HW/SW Codesign Approach for Real-time Image Processing Using HLS," *020 1st International Conference on Communications, Control Systems and Signal Processing (CCSSP),* EL OUED, Algeria, 2020, pp. 169-174, doi: 10.1109/CCSSP49278.2020.9151686.

[5] H. Kavalionak, C. Gennaro, G. Amato, C. Vairo, C. Perciante, C. Meghini and F. Falchi, ”Distributed video surveillance using smart cameras,” *Journal of Grid Computing*, 17(1), 2019

[6] D. Tsiktsiris, D. Ziouzios and M. Dasygenis, "HLS Accelerated Noise Reduction Approach Using Image Stacking on Xilinx PYNQ," 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece, 2019, pp. 1-4, doi: 10.1109/MOCAST.2019.8741574.

[7] P. Benoit, L. Dalmasso, G. Patrigeon, T. Gil, F. Bruguier and L. Torres, "Edge-Computing Perspectives with Reconfigurable Hardware," 2019 *14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, York, United Kingdom, 2019, pp. 51-58, doi: 10.1109/ReCoSoC48741.2019.9034961.

[8] M. Yildirim and A. Çinar, "Simultaneously Realization of Image Enhancement Techniques on Real-Time Fpga," *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*, Malatya, Turkey, 2019, pp. 1-6.

[9] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers and Z. Zhang, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473-491, April 2011, doi: 10.1109/TCAD.2011.2110592.

[10] Zybo Z7 Board Reference Manual, Digilent, 2018.

[11] M. Samarawickrama, R. Rodrigo and A. Pasqual, "HLS approach in designing FPGA-based custom coprocessor for image preprocessing," *2010 Fifth International Conference on Information and Automation for Sustainability*, Colombo, 2010, pp. 167-171, doi: 10.1109/ICIAFS.2010.5715654.

[12] S. Lahti, P. Sjövall, J. Vanne and T. D. Hämäläinen, "Are We There Yet? A Study on the State of High-Level Synthesis," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 898-911, May 2019, doi: 10.1109/TCAD.2018.2834439.

[13] O'Loughlin, Declan & Coffey, A. & Callaly, F. & Lyons, D. & Morgan, F.. (2014). *Xilinx Vivado High Level Synthesis*: Case studies. 352-356.

[14] Pcam 5C Reference Manual, Digilent, 2018