UNIVERSITY OF THESSALY

IMPLEMENTATION OF A FACE DETECTION ALGORITHM ON A RECONFIGURABLE PLATFORM

Author:

Aikaterini Servou

Supervisors:

Dr. Nikolaos Bellas,
Associate Professor

Dr. Christos D. Antonopoulos,
Assistant Professor

A Thesis submitted for the degree of

Diploma of Science in

Computer and Communication Engineering

University of Thessaly
Department of Electrical and Computer Engineering
Volos, Greece

July 2015

IMPLEMENTATION OF A FACE DETECTION ALGORITHM ON A RECONFIGURABLE PLATFORM

ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΟΥ ΑΝΑΓΝΩΡΙΣΗΣ ΠΡΟΣΩΠΟΥ ΣΕ ΕΠΑΝΑΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΗ ΠΛΑΤΦΟΡΜΑ

By

Aikaterini Servou

A THESIS

Submitted in partial fulfillment of the requirements for the degree of

DIPLOMA OF SCIENCE

In Computer and Communication Engineering

UNIVERSITY OF THESSALY

2015

Declaration of Authorship

We, Aikaterini Servou and Christos Vakerlis, hereby certify that this thesis titled, ‘Implementation of a face detection algorithm on a reconfigurable platform’ and the work presented in it has been composed by us and is based on our own work, unless stated otherwise.

The research was carried out wholly or mainly while in candidature for the graduate degree of Diploma of Science in Computer and Communication Engineering at the University of Thessaly, Department of Electrical and Computer Engineering, Greece.

Wherever we have consulted or quoted from the work of others, it is always attributed and the source is given. The main sources of help are referenced in the Bibliography section of this thesis.

Dedicated to our families

Acknowledgements

With the fulfillment of this project, we would like to thank both of our supervisors Dr. Nikolaos Bellas and Dr. Christos Antonopoulos for the great collaboration, and especially our professor Dr. Nikolaos Bellas for his continuous guidance and help throughout the semester, which played a vital role in tackling many unpredictable difficulties in the process of fulfilling our Thesis.

Moreover, we would like to thank all of our friends for their company and psychological support in this magnificent journey of knowledge that has now come to its end.

At this point we would like to express our gratitude to all our professors who put their faith in us and urged us to do better.

In conclusion, we would like to thank our families for the love and support they provided us through our entire lives, for the sacrifices they made on our behalf and for believing in us.

Abstract

Field Programmable Gate Array (FPGA) technology has gained vital importance mainly because of its parallel processing hardware which makes it ideal for image and video processing. In this paper, a step by step approach to apply linear spatial and temporal filters on real time video frame sent by Omnivision OV7670 camera using Zynq Evaluation and Development board based on Xilinx XC7Z020 has been discussed. Face detection application was chosen to explain above procedure. This procedure is applicable to most of the complex image processing algorithms which needs to be implemented using FPGA.

Face detection is one of the challenging problems in Computer Applications and it is widely used for many purposes, especially in digital photography, video surveillance, biometrics and video coding. The applicability of face detection in energy conservation is not as obvious as in other applications. It is, however, a very time consuming task, hence, implementing it in hardware is of primary importance when a real time face detection system is needed.

The purpose of this Thesis was to implement a real-time system on a ZedBoard FPGA to detect a human face. The face detection system mainly consists of a camera and its controller that provides a live streaming of RGB color pixels, a module that converts RGB to YUV color pixels and detects if a pixel is a skin one or not and filter modules that erode and dilate the regions with skin pixels in order to deplete the noise and the non-skin objects. A temporal filter was also implemented in order to reduce flickering as even small changes in lighting could make the result displayed on the VGA screen less stable and finally an attempt to track the faces from the video streaming.

There are also some filters on the skin regions that algorithm involved color-based skin segmentation and image filtering. The face location was determined by calculating the centroid of the detected region. A software version of the algorithm was independently implemented and tested on still pictures in MATLAB. Although the transition from MATLAB to Verilog was not as smooth as expected, experimental results proved the accuracy and effectiveness of the real-time system, even under varying conditions of lights, facial poses and skin colors. All calculation of the hardware implementation was done in real time with minimal computational effort, thus suitable for power-limited applications.

Περίληψη

Η τεχνολογία συστοιχιών επιτόπιων προγραμματιζόμενων πυλών (FPGA) έχει εξελιχθεί σε ζωτικής σημασίας κυρίως λόγω της παράλληλης επεξεργασίας του υλικού, το οποίο το καθιστά ιδανικό για επεξεργασία εικόνας και βίντεο. Σε αυτή τη διπλωματική διατριβή επιχειρείται μία βήμα προς βήμα προσέγγιση της εφαρμογής γραμμικών χωρικών και χρονικών φίλτρων σε πραγματικού χρόνου καρέ, τα οποία αποστέλλονται από την κάμερα Omnivision OV7670 χρησιμοποιώντας την πλατφόρμα ZedBoard Xilinx XC7Z020. Η εφαρμογή ανίχνευσης προσώπου επιλέχθηκε για να επεξηγήσουμε την παραπάνω διαδικασία. Αυτή κυρίως η εφαρμογή χρησιμοποιείται ως βάση για τους περισσότερο σύνθετους αλγορίθμους επεξεργασίας εικόνας με τη χρήση κάποιας FPGA.

Η ανίχνευση προσώπου έγκειται στις μεγαλύτερες προκλήσεις υπολογιστικών εφαρμογών και επιπλέον χρησιμοποιείται για ποικίλους σκοπούς, κυρίως σε ψηφιακή φωτογραφία, παρακολούθηση βίντεο, βιομετρικά στοιχεία και κωδικοποίηση βίντεο. Η εφαρμοσιμότητα της ανίχνευσης προσώπου για την εξοικονόμηση ενέργειας δεν είναι τόσο προφανής όσο σε άλλες εφαρμογές. Είναι, ωστόσο, ένα χρονοβόρο έργο και ως εκ τούτου η υλοποίησή του σε υλικό είναι πρωταρχικής σημασίας σε περιπτώσεις που απαιτείται η χρήση ενός συστήματος ανίχνευσης προσώπου πραγματικού χρόνου.

Σκοπός αυτής της διπλωματικής διατριβής είναι η υλοποίηση ενός συστήματος πραγματικού χρόνου σε μία πλακέτα ZedBoard FPGA για την ανίχνευση ανθρώπινων προσώπων. Το σύστημα ανίχνευσης προσώπου αποτελείται κυρίως από την κάμερα και τον ελεγκτή που παρέχει μία ζωντανή ροή από RGB pixels χρώματος, μία μονάδα που μετατρέπει το RGB σε YUV pixels χρώματος και ανιχνεύει εάν το pixel αντιστοιχεί σε δέρμα ή όχι, καθώς και φίλτρα διάβρωσης και διαστολής των πριοχών που αντιστοιχούν σε δέρμα, με σκοπό τη μείωση του θορύβου και των αντικειμένων που δεν είναι δέρμα. Επιπρόσθετα, υλοποιήθηκε ένα χρονικό φίλτρο, έτσι ώστε να μειωθεί το «τρεμόπαιγμα», καθώς ακόμη και μικρές αλλαγές στο φωτισμό ήταν δυνατόν να συμβάλλουν σε ένα ασταθές αποτέλεσμα στην οθόνη Video Graphics Array (VGA) και εν τέλει επιχειρήθηκε μία προσπάθεια παρακολούθησης του ανιχνευθέντος προσώπου.

Υλοποιήθηκαν, επίσης, κάποια φίλτρα στις περιοχές δέρματος, τα οποία εμπεριέχονται στον αλγόριθμο κατακερματισμού του δέρματος με βάση το χρώμα και στο φιλτράρισμα της εικόνας. Μία έκδοση λογισμικού του αλγορίθμου υλοποιήθηκε ανεξάρτητα

και ελέχθηκε σε μεμονωμένες εικόνες στο MATLAB. Παρόλο που η μετάβαση από το MATLAB σε γλώσσα περιγραφής υλικού (Verilog HDL) δεν ήταν ομαλή, τα αποτελέσματα των πειραμάτων απέδειξαν την εγκυρότητα και την αποτελεσματικότητα του συστήματος πραγματικού χρόνου, ακόμη και υπό μεταβαλλόμενες συνθήκες φωτισμού, θέσεων προσώπου και χρωμάτων δέρματος. Όλοι οι υπολογισμοί της υλοποίησης σε υλικό επιτεύχθηκε σε πραγματικό χρόνο με ελάχιστο υπολογιστικό κόστος, γεγονός που το καθιστά κατάλληλο για περιορισμένης κατανάλωσης ενέργειας εφαρμογές.

Contents

List of Figures

List of Tables

Abbreviations

ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
AWB	**A**uto **W**hite **B**alance
BRAM	**B**lock **R**AM
CLB	**C**onfigurable **L**ogic **B**lock
DSP	**D**igital **S**ignal **P**rocessing/**P**rocessor
FP	**F**alse **P**ositive
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
HDL	**H**ardware **D**escription **L**anguage
ISE	**I**ntegrated **S**ynthesis **E**nvironment
MATLAB	**M**atrix **L**aboratory
RAM	**R**andom **A**ccess **M**emory
RGB	**R**ed **G**reen **B**lue
RTL	**R**egister **T**ransfer **L**evel
SCCB	**S**erial **C**amera **C**ontrol **B**us
SIOC	**S**erial **I**nput **O**utput **C**lock
SIOD	**S**erial **I**nput **O**utput **D**ata
VGA	**V**ideo **G**raphics **A**rray

Chapter 1

Introduction

1.1 Describing the motives

Now a days image processing has become very powerful tool in the field of medical imaging, digital photography, video surveillance etc. Face detection is a computer technology that identifies human faces in digital images. It detects human faces which might then be used for recognizing a particular face [5]. Face detection can be regarded as a specific case of object-class detection. In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class. Examples include upper torsos, pedestrians, and cars. Image processing usually requires very large number of operations and high speed data transfer, therefore parallel processing or multiprocessing hardware is essential. Because of this, FPGA is one of the best alternatives for image processing as it performs the operations in parallel fashion.

There are various algorithms used in the detection process right from the skin color detection to the estimation model. In recent years, skin color detection has become a hot topic between researchers, and great progress has been made in this field. Therefore, there are variety of applications using skin color detection like detecting and tracking human faces and gestures, filtering web image contents and retrieving people in databases and Internet, even diagnosing diseases. In our hardware-based approach to the face detection algorithm skin detection is of primary importance, as it sets the appropriate thresholds to locate the face portion of a human. This allows easy face localization of potential facial regions without any consideration of its texture and geometrical properties.

Many different types of color models are used for skin detection. Each one differs from the others in terms of the manner of transformation (linear or non-linear), the robustness to adapt to light changing, and shadow noises. There are many skin color spaces like RGB, HSV, YCbCr, YIQ, YUV, etc. that are used for skin color segmentation. The RGB color model represents the colors that are in the red, green, and blue planes and does not separate

the luminance from the chrominance components, which makes it a poor choice for color analysis and color based recognition. In this thesis, we used a proposed threshold, which is based on the YUV and RGB color spaces. Our approach is able to benefit from the characteristics of each color model for enhancing the accuracy of skin detection [3].

The next step of the face detection system involves the use of morphological operations to refine the skin regions extracted from the segmentation step. To do so, we have used spatial filters to reproduce in the first place the erosion operation in MATLAB, which basically shrinks the object in order to remove noise and other non-skin pixels, and subsequently the dilate operation, which can group together fragmented sub-regions by applying simple dilation on the large regions. A temporal filter applied on the final result displayed in the VGA screen, allowed flickering to be reduced significantly as even small changes in lighting could make the frame to be depicted less stable.

Finally, centroid computation and tracking was implemented to locate the face region with a maximum of seven (7) faces detected under ideal circumstances, where other skin regions are not falsely detected as human faces (false positives). Due to the limitations of the hardware used, it was infeasible to calculate the centroid and track each face region separately. Our final design is a fully hardware implementation of the face detection algorithm that corresponds to its software implementation and thus it is considered a success.

1.2 Thesis Structure

This thesis is divided in three main Chapters, each one of those includes smaller sections and possibly subsections.

Chapter 2 provides background information useful to understanding the development and the hardware used in this project. At first, it describes the architecture and operation of FPGAs in general and then focuses on the technical characteristics of ZedBoard Xilinx XC7Z020, the FPGA used for development. It also offers information concerning the camera module, OV7670 by OmniVision, used for real time video capture. Finally, the functionality of a general VGA controller is described in order to explain how the output is displayed on a monitor.

Chapter 3 begins with a brief introduction to the hierarchy of the design and then follows with an exhaustive analysis divided into sections for each of its parts. Parts of the design are considered algorithms implemented, in which case the algorithm is explained first and then the approach of its design, or functions necessary to the whole operation of the project. More specifically, an initial approach in the YUV algorithm for color thresholding and skin segmentation, and its implementation are thoroughly analyzed, and then a detailed description on the hardware implementation of the spatial and temporal filters along with the design and realization of the tracking are also examined.

Chapter 4 summarizes the results generated, the problems faced and the given solutions along with the project summary and the resources used in the FPGA. Finally, it provides some future improvements and potential work that could possibly lead to a design corresponding to its software implementation.

Chapter 2

Background

2.1 Field Programmable Gate Array – FPGA [6]

FPGAs are programmable semiconductor devices that are based around a matrix of Configurable Logic Blocks (CLBs) connected through programmable interconnects. As opposed to Application Specific Integrated Circuits (ASICs), where the device is custom built for the particular design, FPGAs can be programmed to the desired application or functionality requirements. Although One-Time Programmable (OTP) FPGAs are available, the dominant type is SRAM-based which can be reprogrammed as the design evolves. Due to this programmability, FPGAs are ideal for a large variety of markets such as ASIC prototyping, Aerospace and Defense, Automotive, Communications, High Performance Computing, Industrial, Medical and Video and Image Processing.

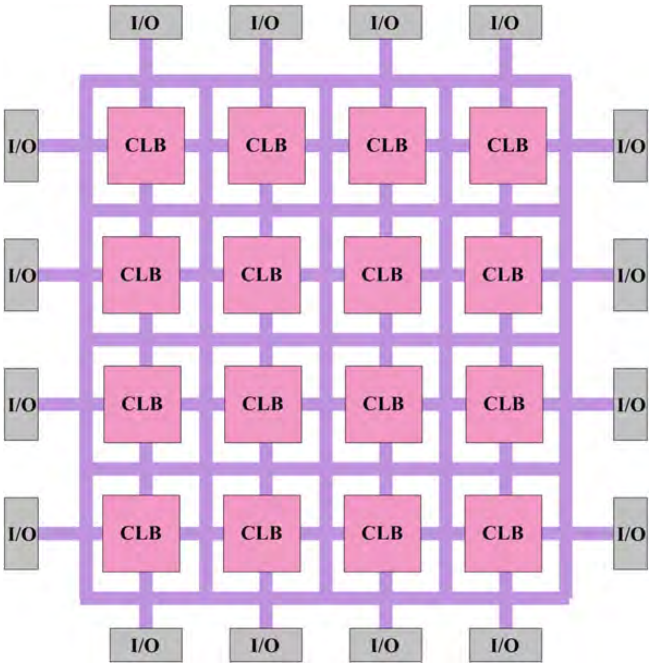

Figure 2.1: Overview of FPGA architecture

2.1.1 Architecture of a FPGA

FPGAs have evolved far beyond the basic capabilities present in their predecessors, and incorporate hard (ASIC type) blocks of commonly used functionality such as RAM, clock management, and DSP. The following are the basic components in a FPGA [6]:

- Configurable Logic Blocks (CLBs) - The CLB is the basic logic unit in a FPGA. Exact numbers and features vary from device to device, but every CLB consists of a configurable switch matrix with 4 or 6 inputs, some selection circuitry (MUX, etc), and flip-flops. The switch matrix is highly flexible and can be configured to handle combinatorial logic, shift registers or RAM.

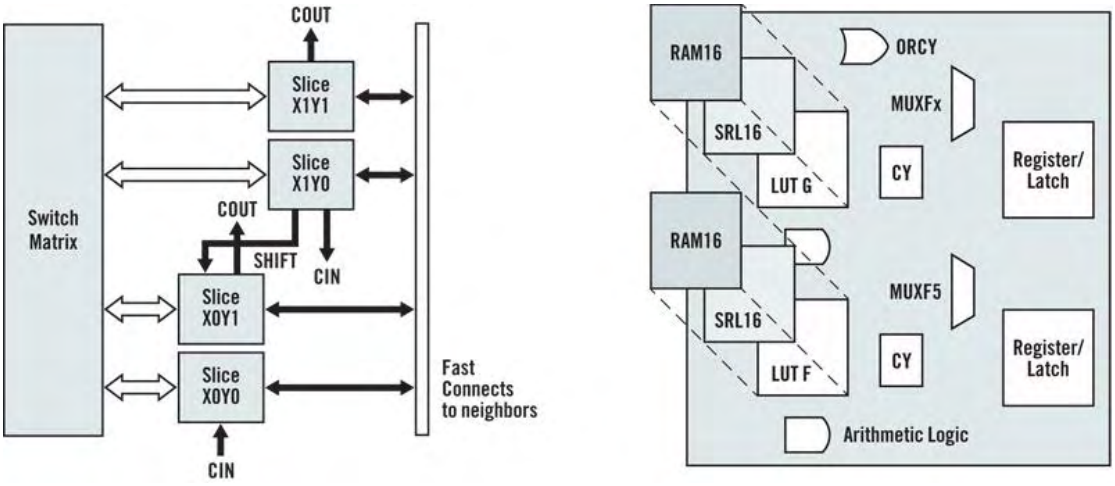

Figure 2.2: Basic Configurable Logic Block Structure [6]

- Interconnect - While the CLB provides the logic capability, flexible interconnect routing routes the signals between CLBs and to and from I/Os. Routing comes in several flavors, from that designed to interconnect between CLBs to fast horizontal and vertical long lines spanning the device to global low-skew routing for Clocking and other global signals. The design software makes the interconnect routing task hidden to the user unless specified otherwise, thus significantly reducing design complexity.
- SelectIO (IOBs) - Today's FPGAs provide support for dozens of I/O standards thus providing the ideal interface bridge in our system. I/O in FPGAs is grouped in banks with each bank independently able to support different I/O standards.

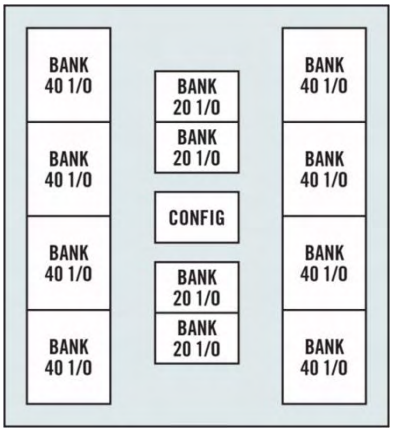

Figure 2.3: Basic SelectIO (IOBs) Structure [6]

- Memory - Embedded Block RAM memory is available in most FPGAs, which allows for on-chip memory in your design.
- Complete Clock Management - Digital clock management provides users the ability to manage the original clock generated from an oscillator on the FPGA and create new clocks, with lower or higher frequency. The most advanced FPGAs offer both digital clock management and phase-looped locking that provide precision clock synthesis combined with jitter reduction and filtering.

2.1.2 ZedBoard™ [14]

The project was developed on the ZedBoard™, which uses Xilinx ZynqR-7000 All Programmable SoC 7z020-CLG484. The device is equipped with an ARM® Processor of approximately 900 MHz and with a variety of Hardware Programmable Logic, allowing designers to add peripherals according to the desirable application.

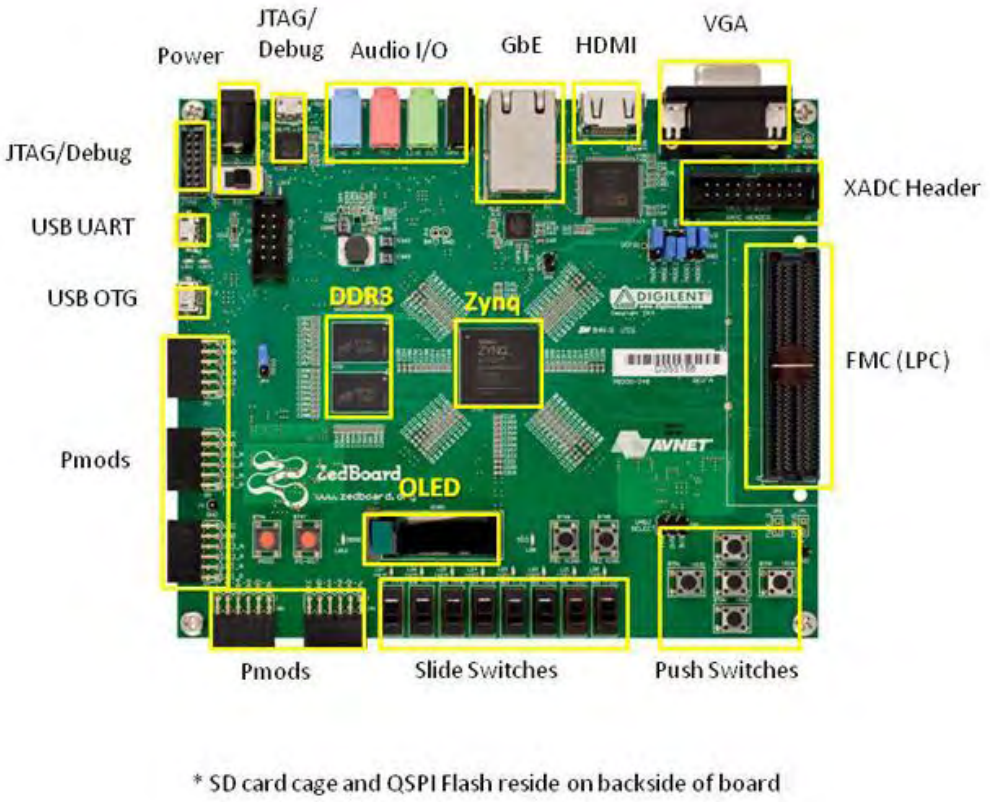

Figure 2.4: Overview of the ZedBoard [14]

The Programmable Logic (PL) section is ideal for implementing high-speed logic, arithmetic and data flow subsystems, while the Processing System (PS) supports software routines and/or operating systems, meaning that the overall functionality of any designed system can be appropriately partitioned between hardware and software. Links between the PL and PS are made using industry standard Advanced eXtensible Interface (AXI) connections.

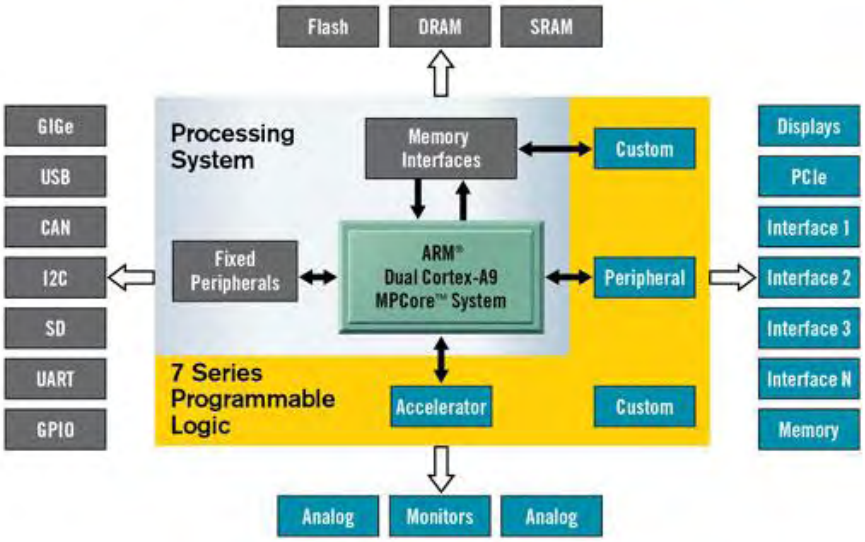

Figure 2.5: System architecture's block diagram for Zynq-7000 AP SoC [12]

The device's technical features are provided in the table below:

Device Name	Z-7020
Xilinx 7 Series Programmable Logic Equivalent	Artix-7 FPGA
Programmable Logic Cells (Approximate ASIC Gates(4))	85K Logic Cells (1.3M)
Look-Up Tables (LUTs)	53,200
Flip-Flops	106,400
Extensible Block RAM (# 36 Kb Blocks)	560 KB (140)
Programmable DSP Slices (18x25 MACCs)	220
Peak DSP Performance (Symmetric FIR)	276 GMACs

Table 2.1: Programmable logic of ZedBoard™

2.2 OmniVision OV7670 CMOS VGA (640x480) CAMERACHIP™ Sensor [7]

The OV7670 CAMERACHIP™ image sensor is a low voltage CMOS device that provides the full functionality of a single-chip VGA camera and image processor in a small footprint package. The OV7670 camera provides an image sensor with 18 (9x2) pins package, which operates at maximum of 30 fps. It has highest resolution of 640 x 480 (VGA) which is equivalent to 0.3 megapixels. This resolution is used in our design. It has built-in digital

signal processor which processes the image before sending to the Zedboard. The pre-processing of image is done by digital signal processor via I2C control bus by setting different values for device control registers which are contained by OV7670 camera module. The coding was done in such a way that outputs of camera module were inputs to the Zedboard and outputs of Zedboard were inputs to the camera. An RGB565 output format of OV7670 was chosen by setting appropriate registers. The key specifications of the camera module are listed in the table below:

Active Array Size		640x480
Power Supply	Digital Core	1.8VDC ±10%
	Analog	2.45V to 3.0V
	I/O	1.7V to 3.0V
Power Requirements	Active	60 mW typical (15fps VGA YUV format)
	Standby	<20 uA
Temperature Range	Operation	-30°C to 70°C
	Stable Image	0°C to 70°C
Output Formats (8-bit)		YUV/YCbCr 4:2:2 RGB565/555/444 GRB 4:2:2 Raw RGB Data
Lens Size		1/6″
Chief Ray Angle		25°
Maximum Image Transfer Rate		30 fps for VGA
Sensitivity		1.3 V/(Lux·sec)
S/N Ratio		46 dB
Dynamic Range		52 dB
Scan Mode		Progressive
Electronic Exposure		Up to 510:1 (for selected fps)
Pixel Size		3.6 um x 3.6 um
Dark Current		12 mV/s at 60°C
Well Capacity		17 K e
Image Area		2.36 mm x 1.76 mm
Package Dimensions		3785 um x 4235 um

Table 2.2: Key Specifications of OV7670 CAMERACHIP™

The pin diagram and description for each pin are shown below:

Pin	Type	Description
3V3, GND	Power	Power Supply, Ground
SIOC	Input	SCCB serial interface clock input
SIOD	Input	SCCB serial interface data I/O
XCLK	Input	Camera clock input
PWDN	Input	Power down mode selection (Active High)
RESET	Input	Resets all registers to default value (Active Low)
D[7:0]	Output	RGB video component output (parallel data)
HREF	Output	Horizontal synchronization
VSYNC	Output	Vertical synchronization
PCLK	Output	Pixel clock output

Table 2.3: OV7670 Pins

The functionality of the camera's module is shown in the following figure:

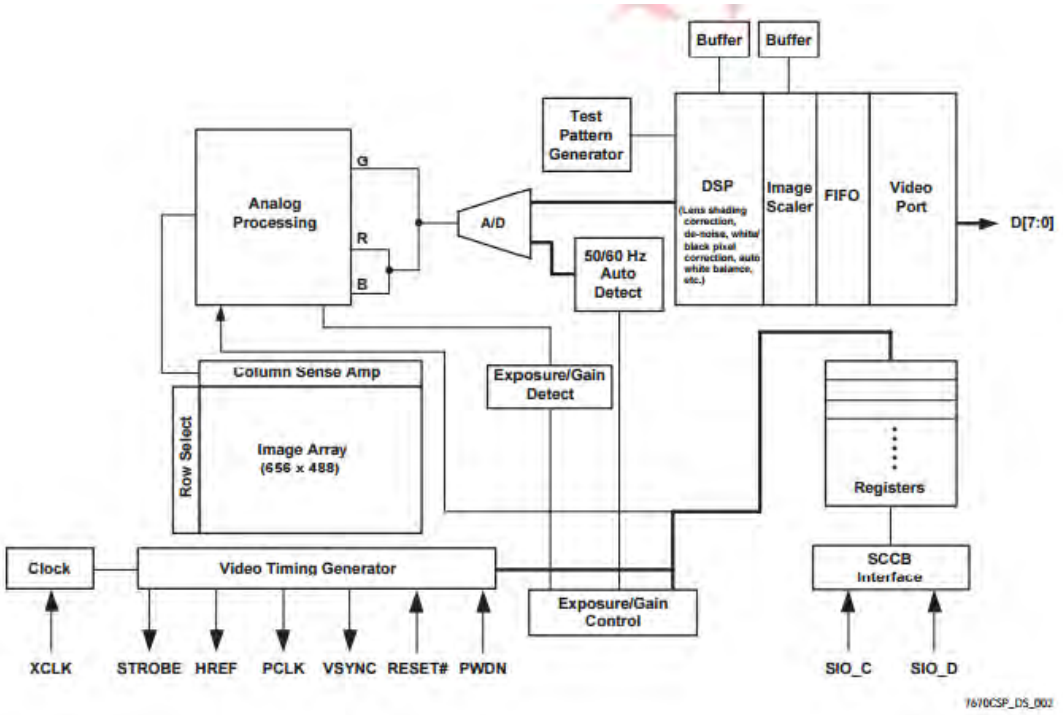

Figure 2.6: Functional Block Diagram of OV7670 CAMERACHIP™ [1]

2.2.1 I2C Bus Protocol

OV7670 is a versatile camera module due to its inner Digital Signal Processor (DSP), which can pre-process image before it is sent. This DSP can be accessed via an I2C bus protocol interface. I2C_sender module is responsible to send the configurations of OV7670_registers module to the camera. As a result, it creates a 32-bit vector and sends it serially using Serial Input Output Clock (SIOC) and Serial Input Output Data (SIOD) signals.

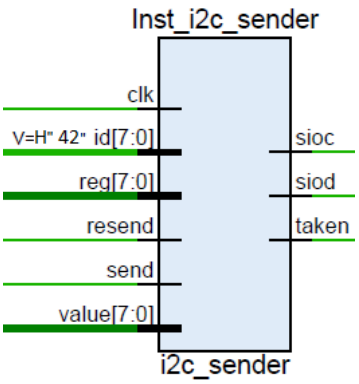

Figure 2.7: I2C Sender Module

The vector sent to the camera has the following format: {3'b100, id, 1'b0, reg, 1'b0, value, 3'b001}. The three first bits "100" correspond to the starting sequence, the last two "01" correspond to the stopping sequence and the '0' after 'id' , 'reg' and 'value' indicates that the Slave is not asked to return acknowledgment for their successful transmission. In our design we have only one slave module for the I2C bus protocol, i.e. the camera.

The 'id' indicates the code for the Slave's address and if we want to write to or read from it. We only need to write to camera's registers and the according code for the OV7670 camera is 0x42(hex). Signal 'taken' becoming high (i.e. '1') means that the 32-bit vector has been sent to the camera and the OV7670_registers module needs to give the next configuration to be sent. As long as signal 'send' is high that means that OV7670_registers module has at least one valid data to send to the camera. An 8-bit counter is used in order to create SIOC frequency dividing the original clk frequency (50MHz) by 256 (2^8).

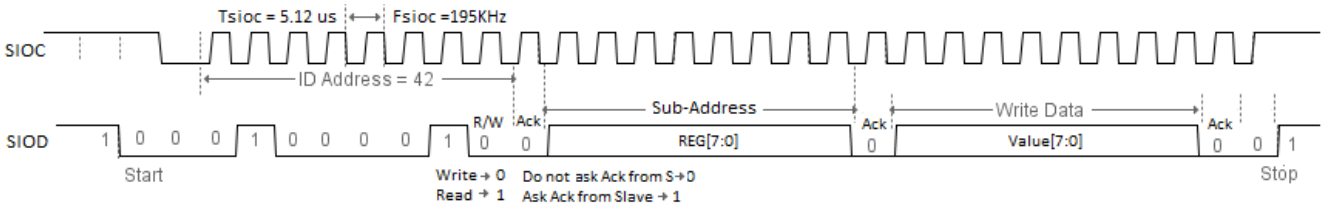

Figure 2.8: I2C Bus Protocol Communication Waveform

2.2.2 OV7670 CAMERACHIP™ Device Control Registers

Device Control registers contained in the OV7670 enable different configurations (e.g. saturation, brightness, contrast etc.) and operating modes. In our design we have used the 7 out of 8 switches (SW 1-7) and the left and right buttons of the ZedBoard to model such configurations and change their default values.

The ov7670_registers module is responsible for initializing the camera with the desired configurations. We use a counter initialized to zero and for each counter's value there is a different output (command: {reg, value}) via a multiplexer. These outputs are the configurations of the camera showing the value that will be written to the camera's corresponding register. When 'advance' becomes high, the counter is incremented by 1 giving the next output. There are many stored configurations that can affect the camera's output and

combinations according to the Switches 1-7 and buttons left and right. The counter stops increasing when the output takes the value ‘ffff’, which is the last one. If ‘resend’ becomes high, the counter is initialized back to ‘0’.

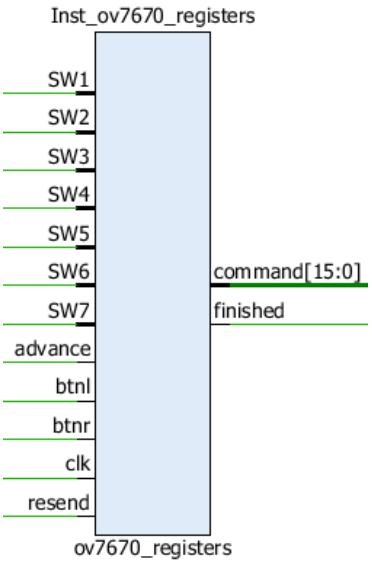

Figure 2.9: ov7670_registers Module Block Diagram

As mentioned above, we have managed to make the values of some basic camera configurations parameterized using the switches and buttons of the FPGA board. The significance of each of them will be presented in the section below [8]:

- Saturation: High color saturation would make the picture look more vivid, but the side effect is the bigger noise and not accurate skin color.

- Auto White Balance (AWB): Simple white balance assumes “gray world”, which means the average color of world is gray. It is true for most environments. The advantage of simple AWB is that it does not depend on lens. A general setting for simple white balance could be applied for all modules with different lens. However, the disadvantage of simple AWB is that the color is not accurate in conditions where “gray world” is not true. For example, if there is in the background a huge red, blue or green etc., the color of the foreground would not be accurate. If the camera targets single color such as red, blue, green, the simple white balance will make the single color gray.

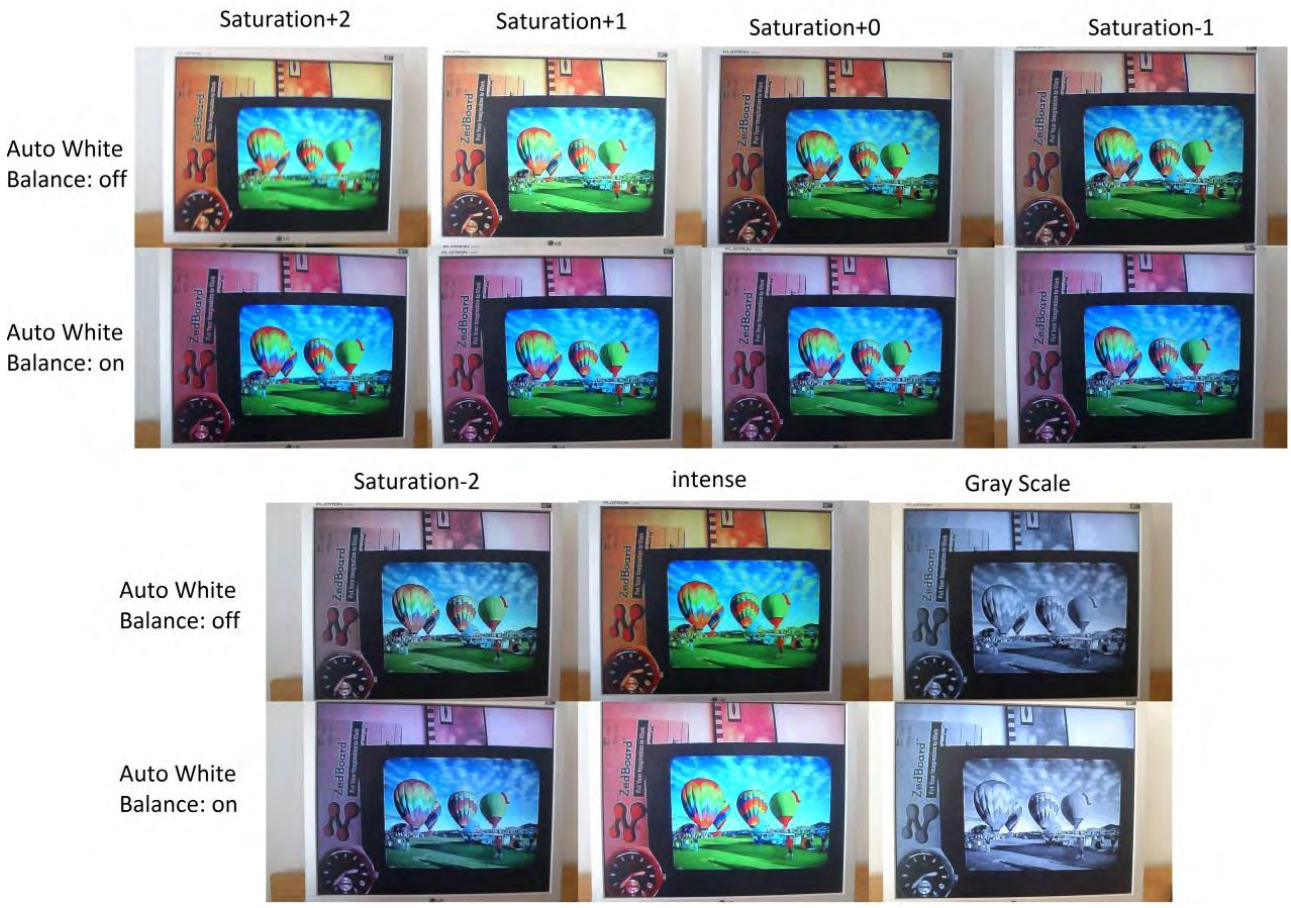

Figure 2.10: Different Options for Saturation and Auto White Balance (AWB)

- Brightness: Higher brightness will make the picture brighter. The side effect, however, is the picture looks foggy.

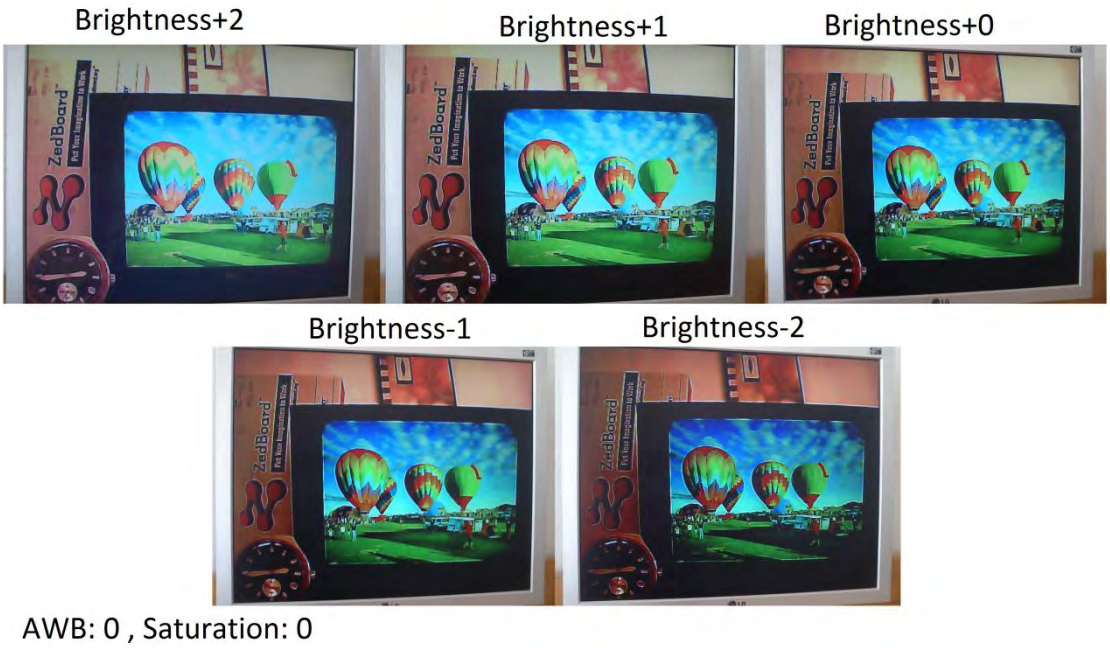

Figure 2.11: Different Options for Brightness

- Contrast: Higher contrast will make the picture sharp, but the side effect is losing dynamic range.

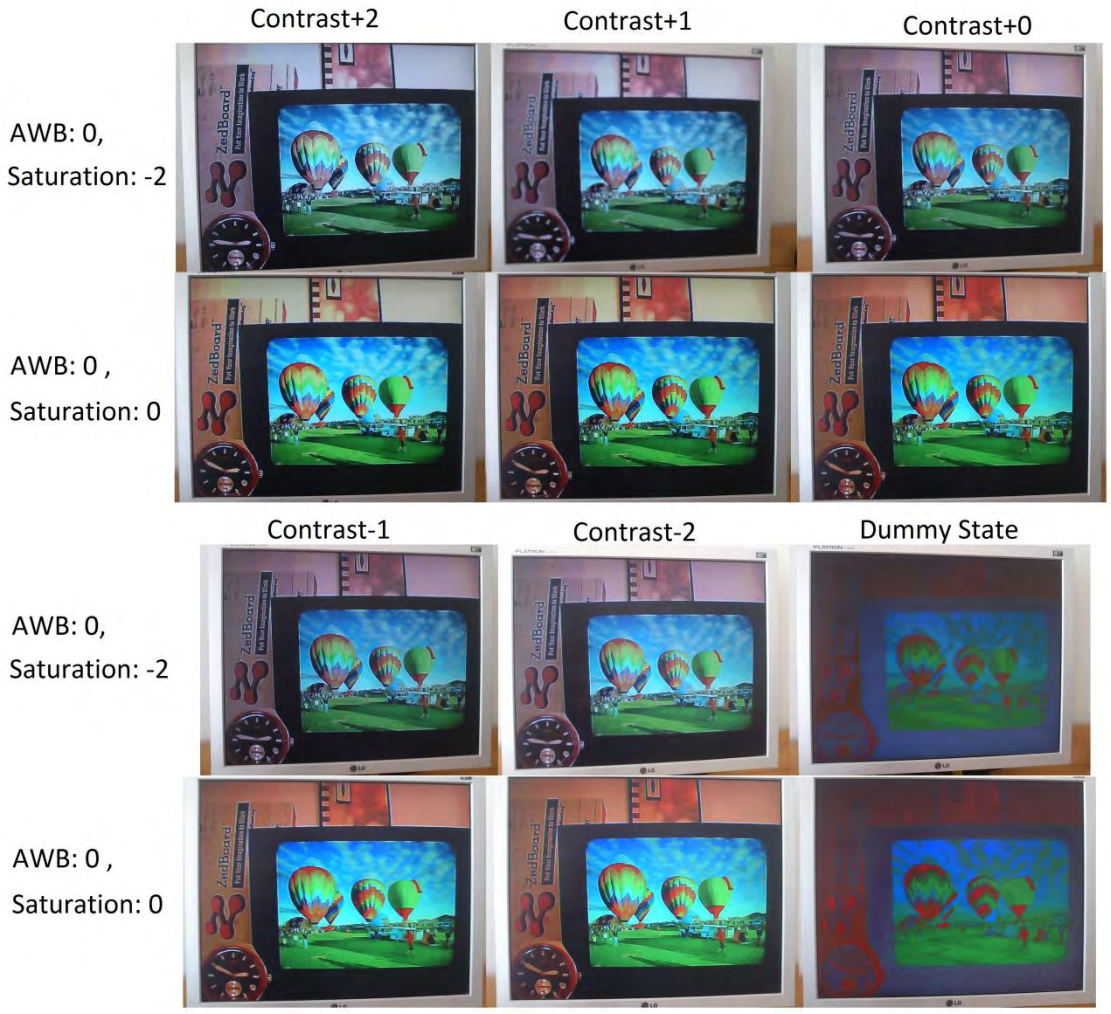

Figure 2.12: Different Options for Saturation and Contrast

- Gamma (Red, Green, Blue): Gamma is the relationship between the brightness of a pixel as it appears on the screen, and the numerical value of that pixel. Gamma encoding of images is used to optimize the usage of bits when encoding an image or bandwidth used to transport an image, by taking advantage of the non-linear manner in which humans perceive light and color.

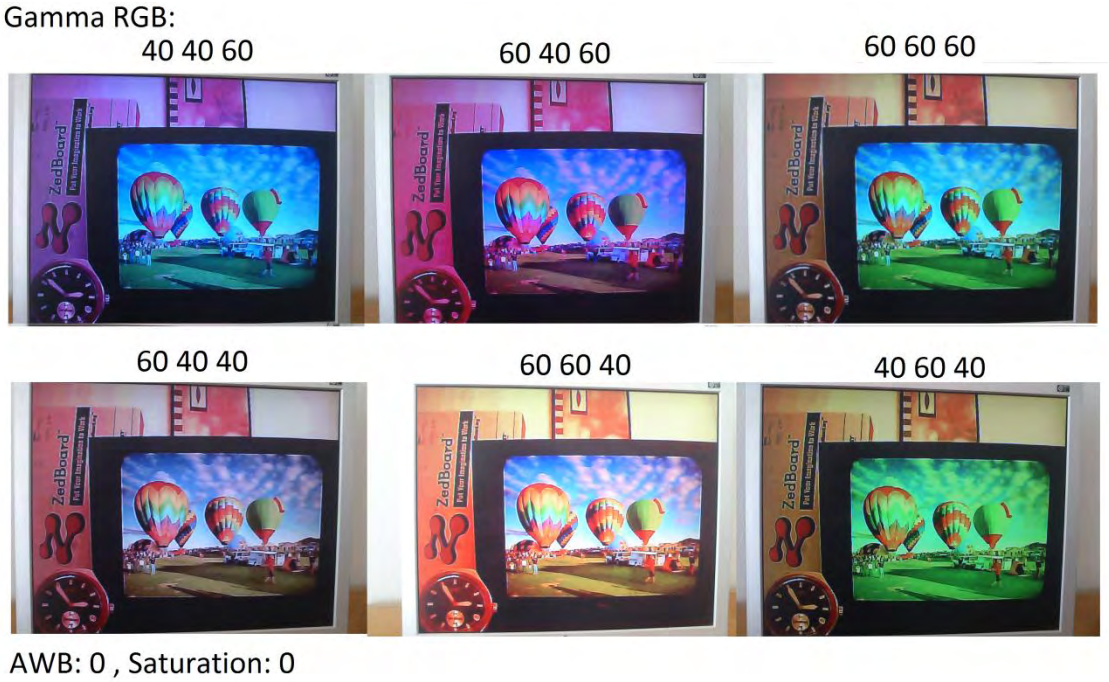

Figure 2.13: Different Options for Gamma (RGB)

- Gain Control (Red, Green, Blue): Increasing the gain control of a single color would increase the component of this specific color against the other colors for each color pixel.

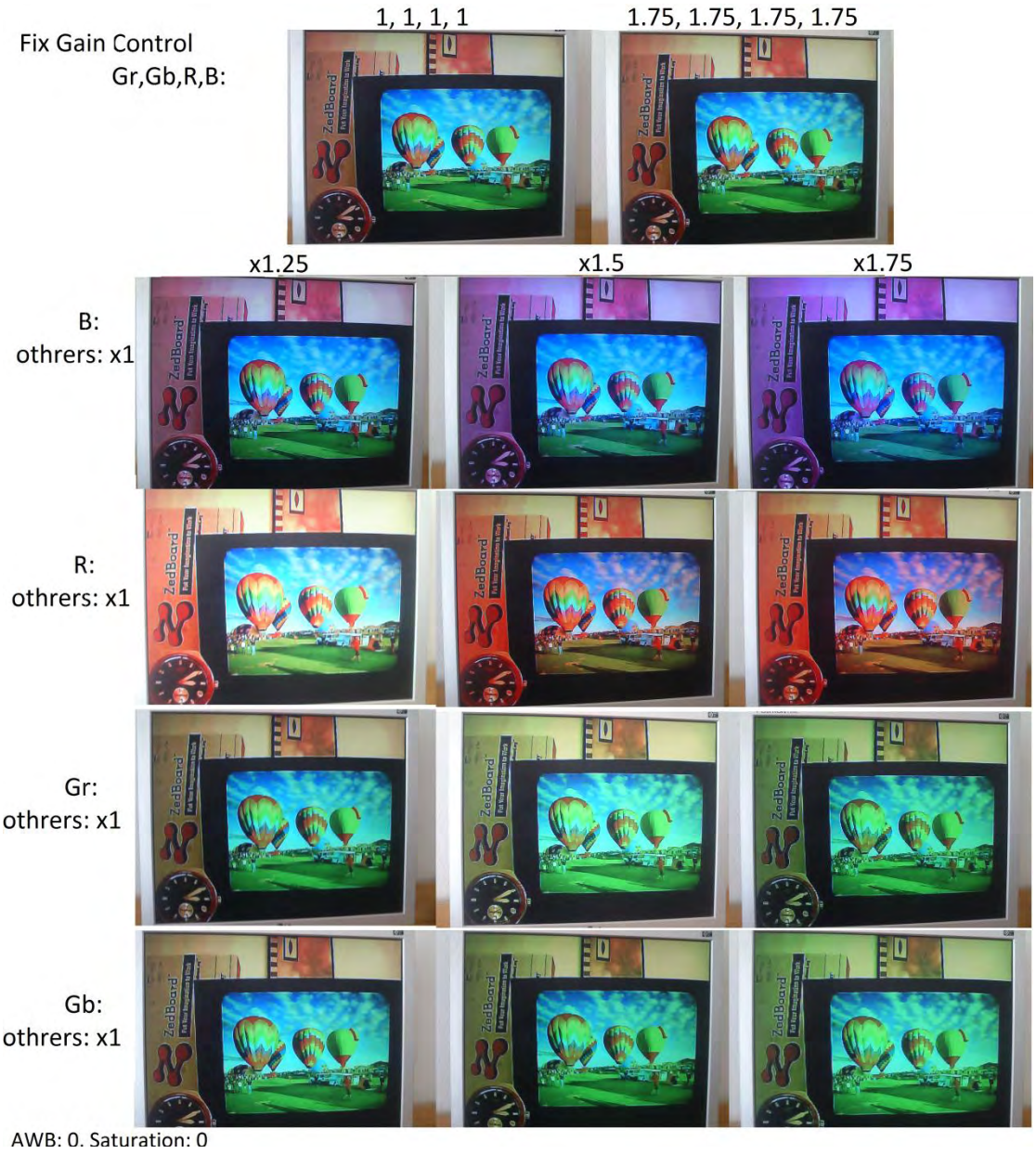

Figure 2.14: Different Options for Fix Gain Control (Red, Green, Blue)

Figure 2.15: Different Options for Fix Gain Control (Red, Green, Blue)

If the user would like to change any of the parameterized configurations, he should then also press the reset button (center button) on the ZedBoard to reinitialize the camera with the changed settings. In case he would like to return to the initial configurations, he should press the left and right button simultaneously on the FPGA. The default values came from different attempts and some of these registers' values are cited in the given references ([2], [8] and [16]).

These parameterized configurations and the corresponding switches and buttons used are shown in the Table 2.4 that follows:

Operations	Switches 7	6	5	4	3	2	1	0	Buttons	Modes - Ranges	Initial mode
Camera											
Saturation		0	0	0	0	0	0		Left, Right	+2, +1, +0, -1, -2, i, GS, GS	-2
Brightness							1		Left, Right	+2, +1, +0, -1, -2, 0, 0, 0	+0
Auto White Balance	1									On, Off	
Gamma Blue		0				1			Left(-1), Right(+8)	00,...,ff (hex)	40(hex)
Gamma Red		0			1				Left(-1), Right(+8)	00,...,ff (hex)	40(hex)
Gamma Green		0		1					Left(-1), Right(+8)	00,...,ff (hex)	40(hex)
Contrast		0	1						Left, Right	+2, +1, 0, -1, -2, DS, DS, DS	+1
Fix Gain Control Green(Blue)		1	1						Left, Right	x1, x1.25, x1.5, x1.75	x1.25
Fix Gain Control Green(Red)		1		1					Left, Right	x1, x1.25, x1.5, x1.75	x1.25
Fix Gain Control Red		1			1				Left, Right	x1, x1.25, x1.5, x1.75	x1.25
Fix Gain Control Blue		1				1			Left, Right	x1, x1.25, x1.5, x1.75	x1.25
Face Detection											
Binary VGA output								1		On, Off	
Face Tracking				1						On, Off	
Skin threshold - RGB condition			1							On, Off	
Skin threshold - YUV condition			0							On, Off	
Filter mode (applied on binary skin image)					0				Down, Up	1.Without filtering 2.Spatial erosion 3.Spatial dilation 4.Temporal	1
Erosion filter (threshold)					1	0	1		Down(-1), Up(+1)	0,...,127 (dec)	60(dec)
					1	1	1		Down(-10), Up(+10)		
Dilation filter (threshold)					1	0	0		Down(-1), Up(+1)	0,...,127 (dec)	30(dec)
					1	1	0		Down(-10), Up(+10)		

Empty Switch Cells: Do not Care
GS : Gray Scale
DS: Dummy State
i: intense

Table 2.4: Parameterized Camera Operations and Configurations

The design and interface of the camera controller is shown in the following diagram:

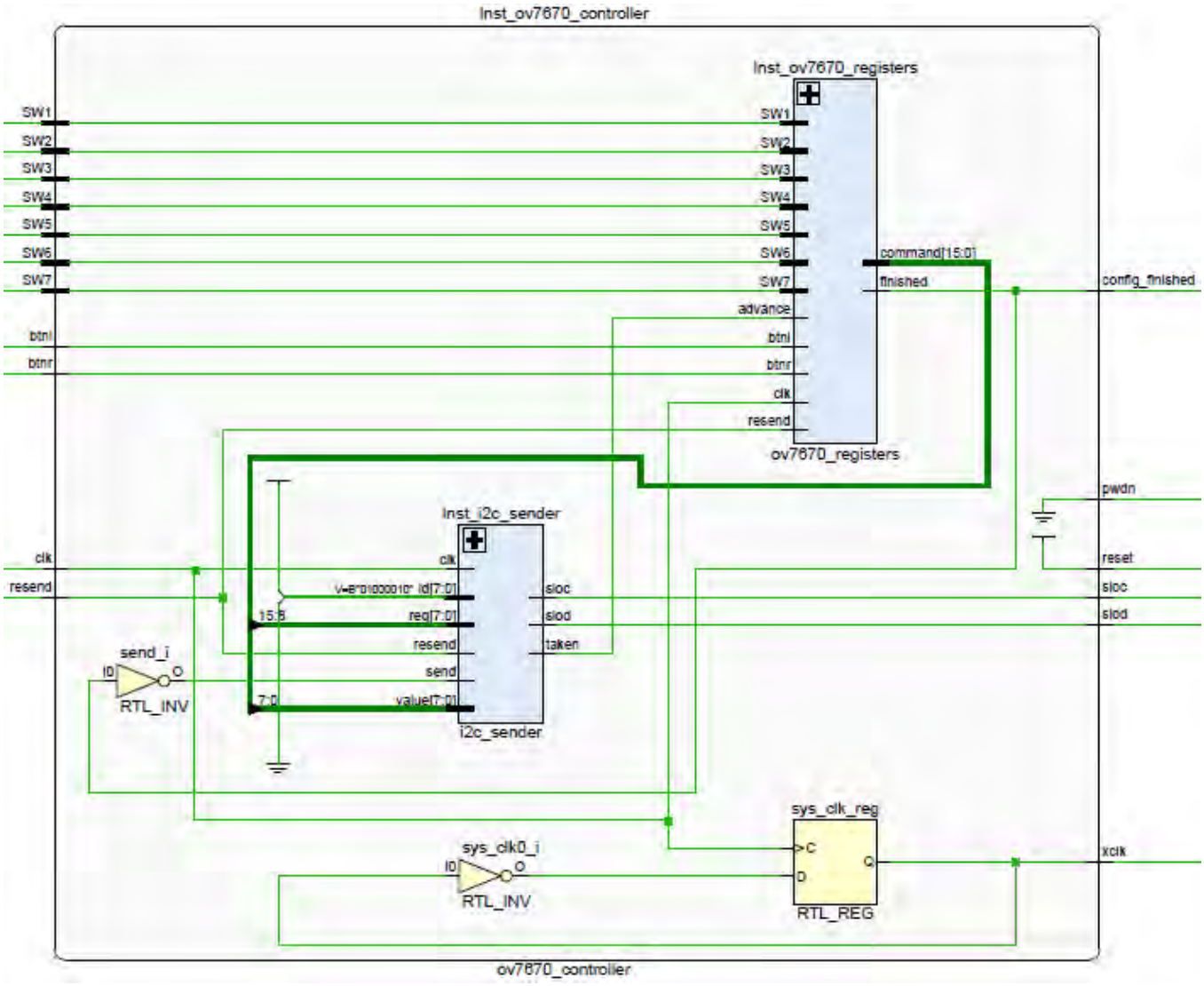

Figure 2.16: Block Diagram of Camera Controller

2.3 VGA Protocol

VGA stands for "Video Graphics Array". VGA has become a well-known standard interface in many embedded systems such as video surveillance systems, ATM machines, or video players. It provides a simple method to connect a system with a monitor for showing information/images, or for users to interact with the system. The VGA standard was originally developed by IBM in 1987 and allowed for a display resolution of 640x480 pixels. Since then, many revisions of the standard have been introduced. The most common is Super VGA

(SVGA), which allows for resolutions greater than 640x480, such as 800x600 or 1024x768 [21].

In order to display an image on screen, VGA monitor controller has to read every pixel data of the image while driving the color signals and synchronization signals of the VGA interface. All pixels are scanned in raster order at a frequency called pixel frequency. To ensure the visual quality, whole image will be re-drawn at a rate determined by refresh rate. The pixel frequency certainly depends on the display resolution and the refresh rate, better resolution or higher fresh rate will require higher pixel frequency.

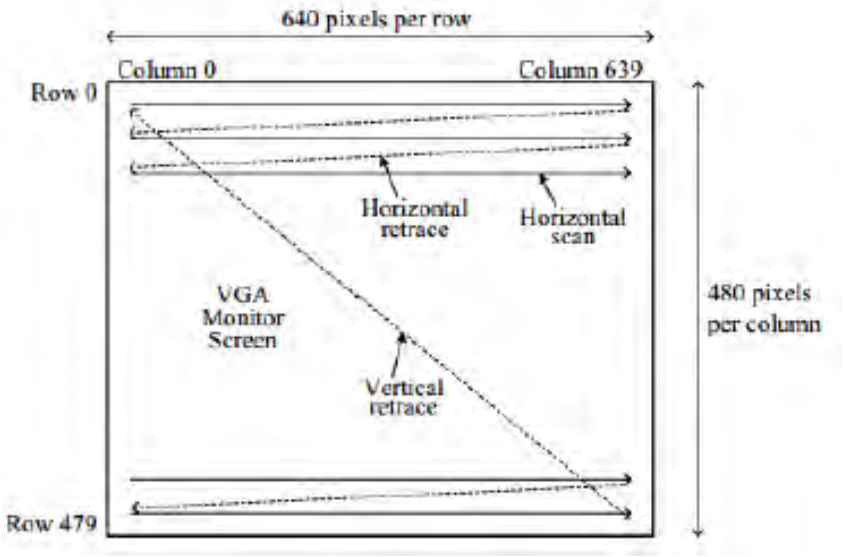

Figure 2.17: Horizontal and Vertical Inversion Process

The video displayed is a stream of still frames that the eye perceives as a moving image. Each frame is an array of pixels set horizontally and vertically that are drawn in order of lines from top to bottom and in each line from left to right. The interface provides the monitor with horizontal and vertical sync signals, color magnitudes, and ground references. The hsync and vsync are digital signals that synchronize the signal timings with the monitor. Both follow the same wavelength pattern but with different timings. These wavelengths can be divided into two main regions. The reset is the active region where color is transmitted and the actual display takes place. The second is the blanking region, where color should not be transmitted. In about the middle of this blanking interval, a pulse of the synch signal takes place that defines three inner regions. The region before the pulse is called front porch, followed by the pulse region and then the back porch. While the pattern is the same for both signals, the hsync wavelength applies to a single line, but the vsync wavelength applies to a

whole frame. So during the active region of the vsync signal all lines must be drawn, meaning the hsynch signal has to repeat its pattern multiple times.

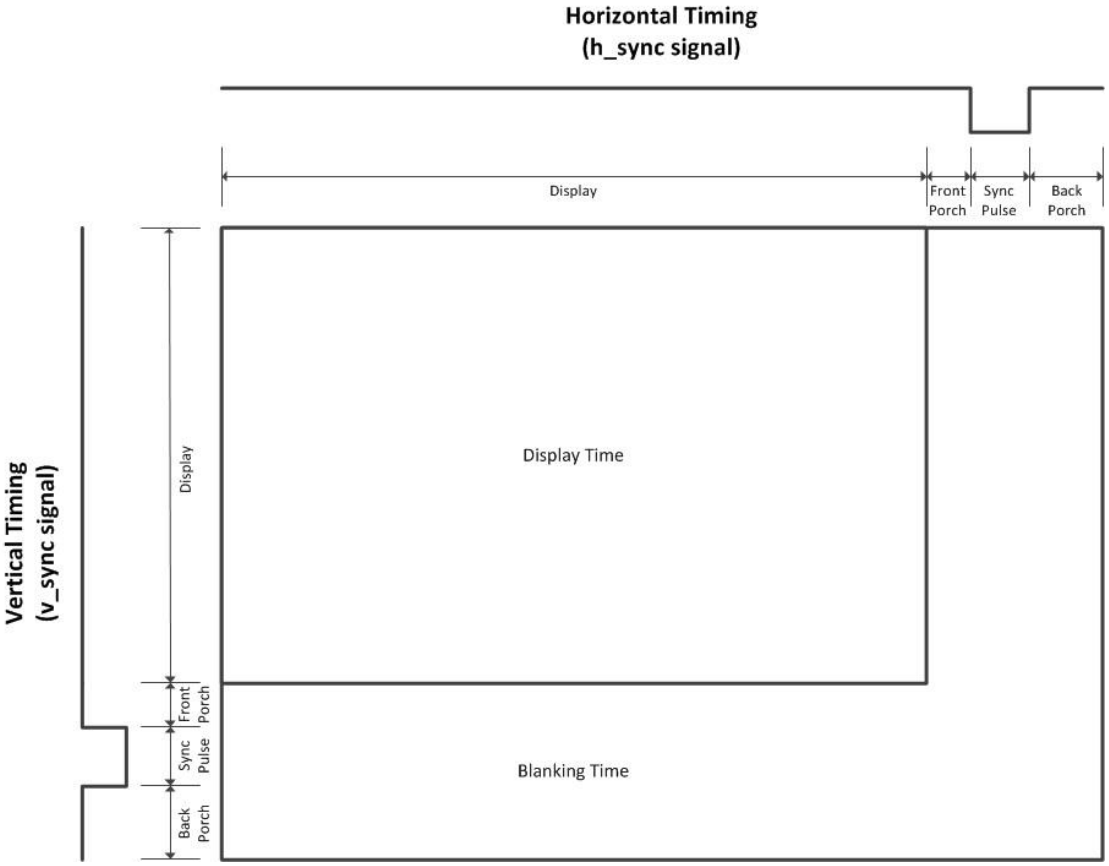

Figure 2.18: VGA Synchronization through h_sync and v_sync Signals [13]

The system clock is the source clock for the side of bus interface and is equivalent to a 50MHz clock frequency, while the pixel clock is used for the side of VGA interface and is equivalent to a 25 MHz clock frequency. The pixel clock frequency is required according to the display standard for a resolution of 640x480 pixels at 60 Hz Refresh Rate. It is driven by an on-chip clock generator using Digital Clock Manager and PLL blocks of the ZedBoard chips [9]. The VGA standard timings are shown in the tables below [10]:

General Timing

Screen refresh rate	59.52 Hz
Vertical refresh	31.25 kHz
Pixel freq.	25 MHz

Table 2.5: General Timing [10]

Horizontal Timing (Line) - Polarity of horizontal sync pulse is negative.

Scanline part	Pixels	Time [us]
Visible area	640	25.6
Front porch	16	0.64
Sync pulse	96	3.84
Back porch	48	1.92
Whole line	800	32

Table 2.6: Horizontal Timing for a single line [10]

Vertical Timing (Frame) - Polarity of vertical sync pulse is negative.

Frame part	Lines	Time [ms]
Visible area	480	15.36
Front porch	10	0.32
Sync pulse	2	0.064
Back porch	33	1.056
Whole line	525	16.8

Table 2.7: Vertical Timing for a whole frame [10]

In order to calculate the pixel's clock frequency needed, we have to multiply the number of clock cycles a frame needs by the frame rate. The calculations are shown below:

$$PixelClockFreq = FrameClockCycles * FrameRate$$

$$= LinesPerFrame * LineClockCycles * FrameRate$$

$$= 525 * 800 * 60$$

$$= 25.2\ MHz$$

Each pixel's color is a combination of red, green and blue, the size of which depends on the output device. The output pixel's color size from the Zynq 7z020 FPGA board to the VGA cable is 12-bit wide (4 bits for each color), resulting in 4096 possible colors, imposed by ZedBoard's characteristics. The color magnitudes are 0V-0.7V analog signals sent over to the RGB wires (Red, Green and Blue). To produce those magnitudes a digital representation of arbitrary bit size for each of red, green and blue passes through a Digital to Analog Converter (DAC), along with all the other signals properly set from the VGA controller. The connector consists of 15 pins. Six pins are used for the colors (RGB), and their respective ground signals two for the hsync and vsync signals, two for grounds and the remaining five are not used.

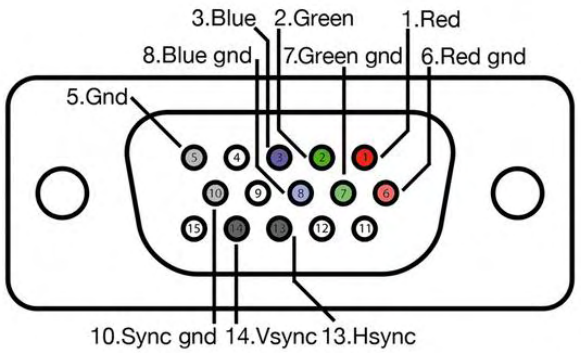

Figure 2.19: VGA Connector Pins

Chapter 3

Design and Implementation

In this chapter we will focus on the design and implementation of our work; the implementation of the face detection and tracking algorithm in a FPGA device purely in hardware using both VHDL and Verilog Hardware Description Languages. In the first section we will introduce an abstract view of the design, the modules and the interconnections of the digital circuit. Following, we will elaborate each module and describe the algorithms implemented in hardware. Finally, we will present an algorithm for tracking the detected faces on real-time video streaming.

The concept of the design is based on the software implementation of the face detection algorithm using the OpenCV functions erode() and dilate() in C++. A general approach to the algorithm is compromised by some specific steps. More explicitly, a skin detection algorithm is used as a first step for color segmentation, as this approach has been proved to be an effective method to detect face regions due to its low computational requirements and ease of implementation. The original image is converted to a different color space, i.e. from RGB to YUV. Then spatial filtering is applied to reduce false positives and create solid areas of skin-based pixels. A temporal filter is also applied to reduce the flickering of the image in the monitor due to lighting. Finally, only regions with large area are considered face regions and for each of them a centroid is computed to show its location.

A general design stages hardware implementation is illustrated in Figure 3.1:

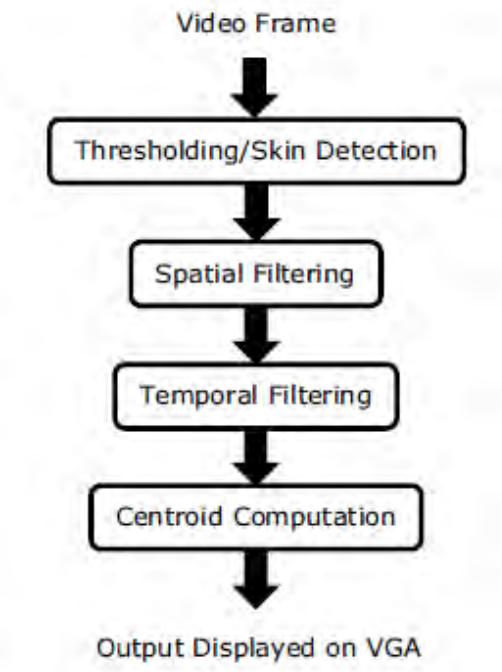

Figure 3.1: Face Detection Algorithm Design Flow [4]

3.1 High Level Design

Face detection and tracking algorithm in real-time video streaming is fully implemented in hardware, and thus no processor is needed for the operations of the YUV skin thresholding, the spatial filters (i.e. erode and dilate), the temporal filter for flickering reduction and finally the tracking of the detected faces. The modules used are the VGA controller, a debounce module for each of the five buttons (up, down, left, right and center) on the FPGA board, the controller for the camera module OV7670, which consists of the I2C sender bus protocol and the registers for the configurations of the camera along with the capture module for the pixel's color data transmission, and also the spatial and temporal filters used along with a face tracking implementation.

A high level illustration of the design is shown in the block diagram below:

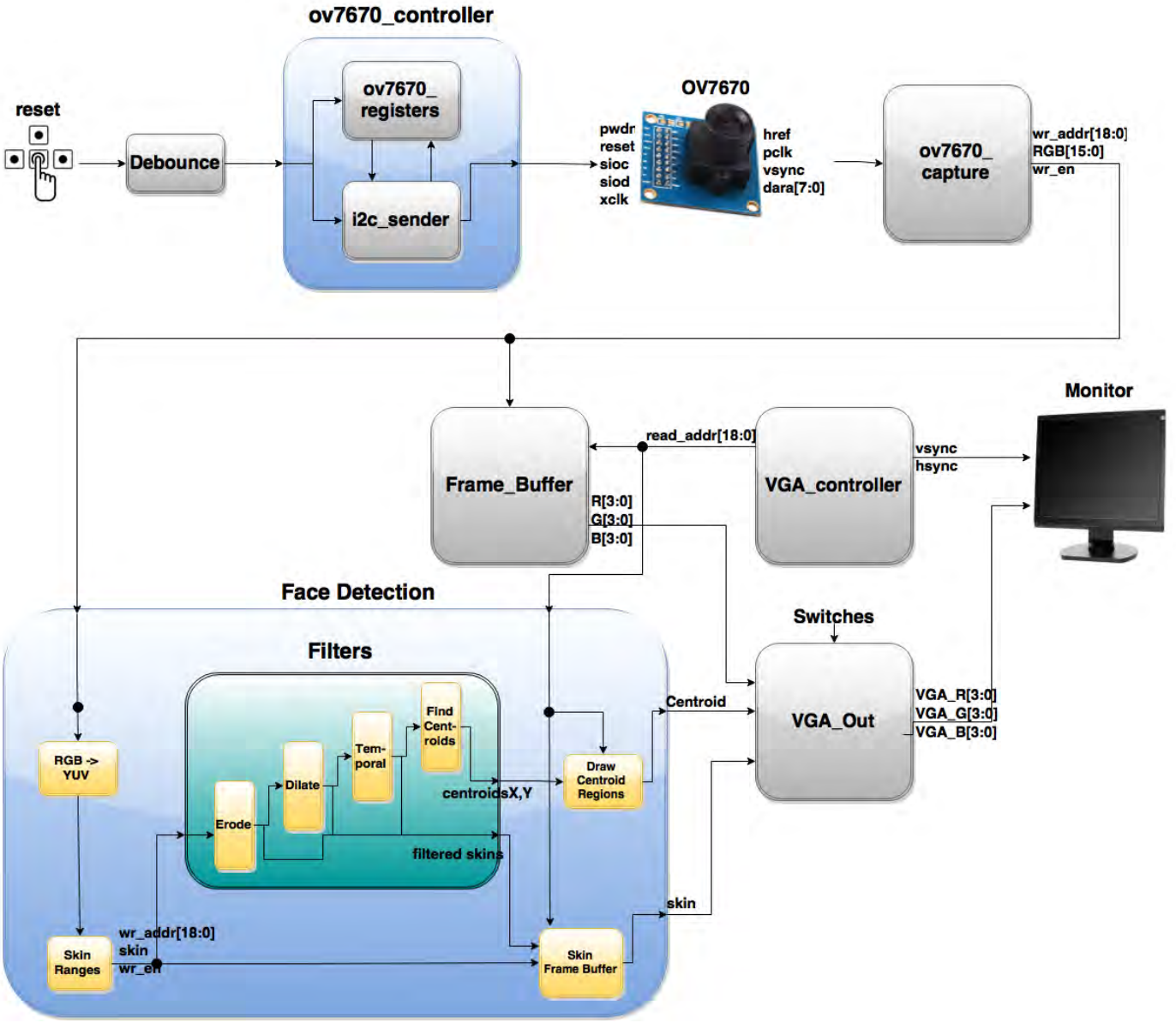

Figure 3.2: High Level Block Diagram of the Digital Circuit

3.1.1 Camera Interface

The image processing is achieved with the use of a low cost camera module (OV7670) as mentioned before. The camera is connected to the JA and JB ports of ZedBoard using jumper wires and PMOD connectors. The VGA output port of ZedBoard is connected to monitor through DB-15 connector. Image processing setup via ZedBoard and OV7670 is shown in the following picture:

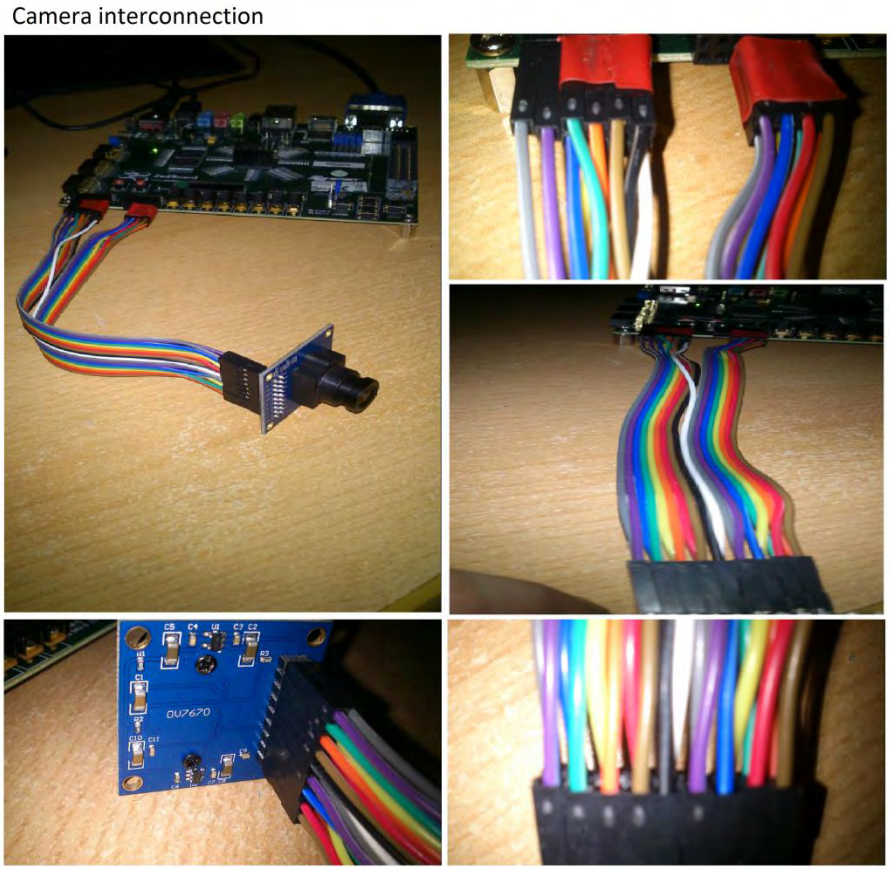

Figure 3.3: ZedBoard and Camera Interconnection

We have already analyzed the camera controller on Chapter 2. The camera implementation and interface with the ZedBoard is based on the work of reference [2]. Now we will discuss on the ov7670_capture module and data manipulation. An abstract representation of the camera design is shown below [1]:

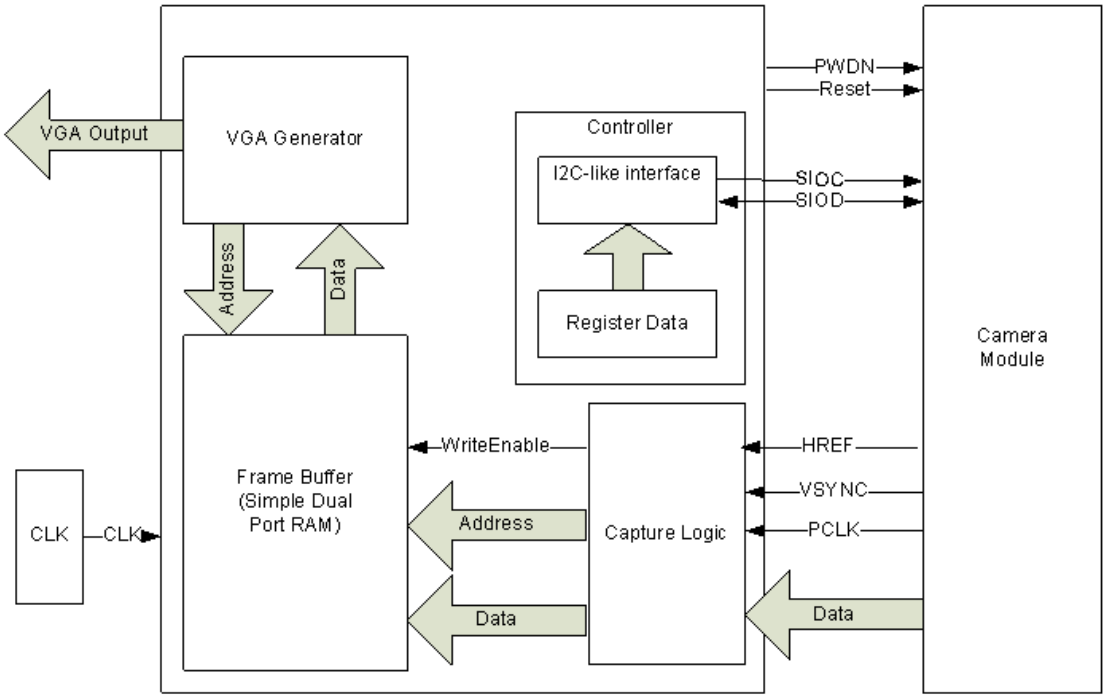

Figure 3.4: High Level Camera Design [1]

The ov7670_capture module has four inputs: pclk (all write data operations to the Block RAMs are held through this clock), href, vsync (synchronization signals) and Data [7:0] (camera 8 bits parallel data output), and outputs the signal 'write enable', and two vectors; addr [18:0] (pixel write address for the frame buffer) and dout [15:0] (RGB565 color pixel). A data transmission is accomplished only if vsync signal is low; then href starts a pixel transfer that takes 2 cycles. More specifically, we acquire the first byte (8 bits parallel data) of pixel data output during the first cycle, and on the next cycle we acquire the second byte of data (a total of 16 bits RGB565 color pixel). The code in Verilog HDL that implements the described logic for the capture module is given below:

```
always@(posedge pclk) begin
        If (vsync) begin
            addr <= 19'b0;
            we    <= 1'b0;
            wr_hold <= 2'b0;
         end
         else begin
            Dout[7:0]   <= d;
```

```
            Dout[15:0] <= Dout[7:0];
            wr_hold     <= {wr_hold[0], href && ~wr_hold[0]};
            if (wr_hold[1]) begin
                we <= 1;
                addr <= addr + 1;
            end
            else
                we <= 0;
        end
end
```

The waveforms in Figure 3.5 indicate the operation of href signal and the data transmission as described above:

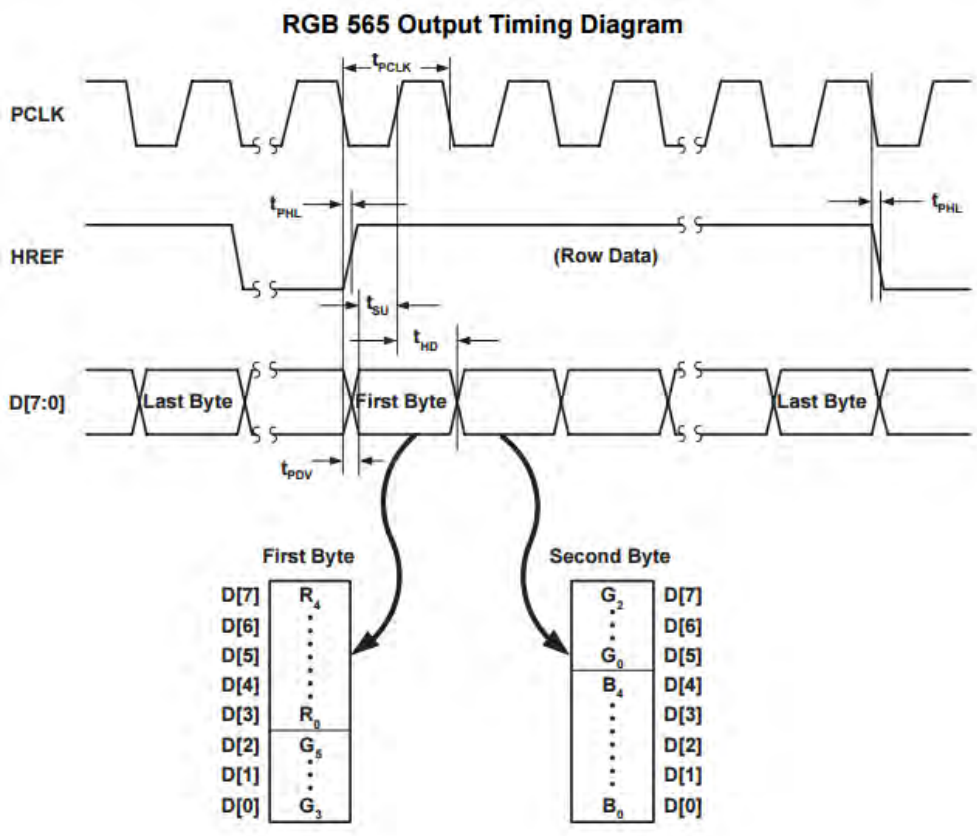

Figure 3.5: Data Transmission of a single pixel RGB565 color data

The 640 x 480 pixels frame is stored in 3 Block RAMs, one BRAM for each color (Red, Green and Blue) where in each address in the frame buffer is stored a 4-bit data for each color pixel (RGB444) as shown in the Figure 3.6:

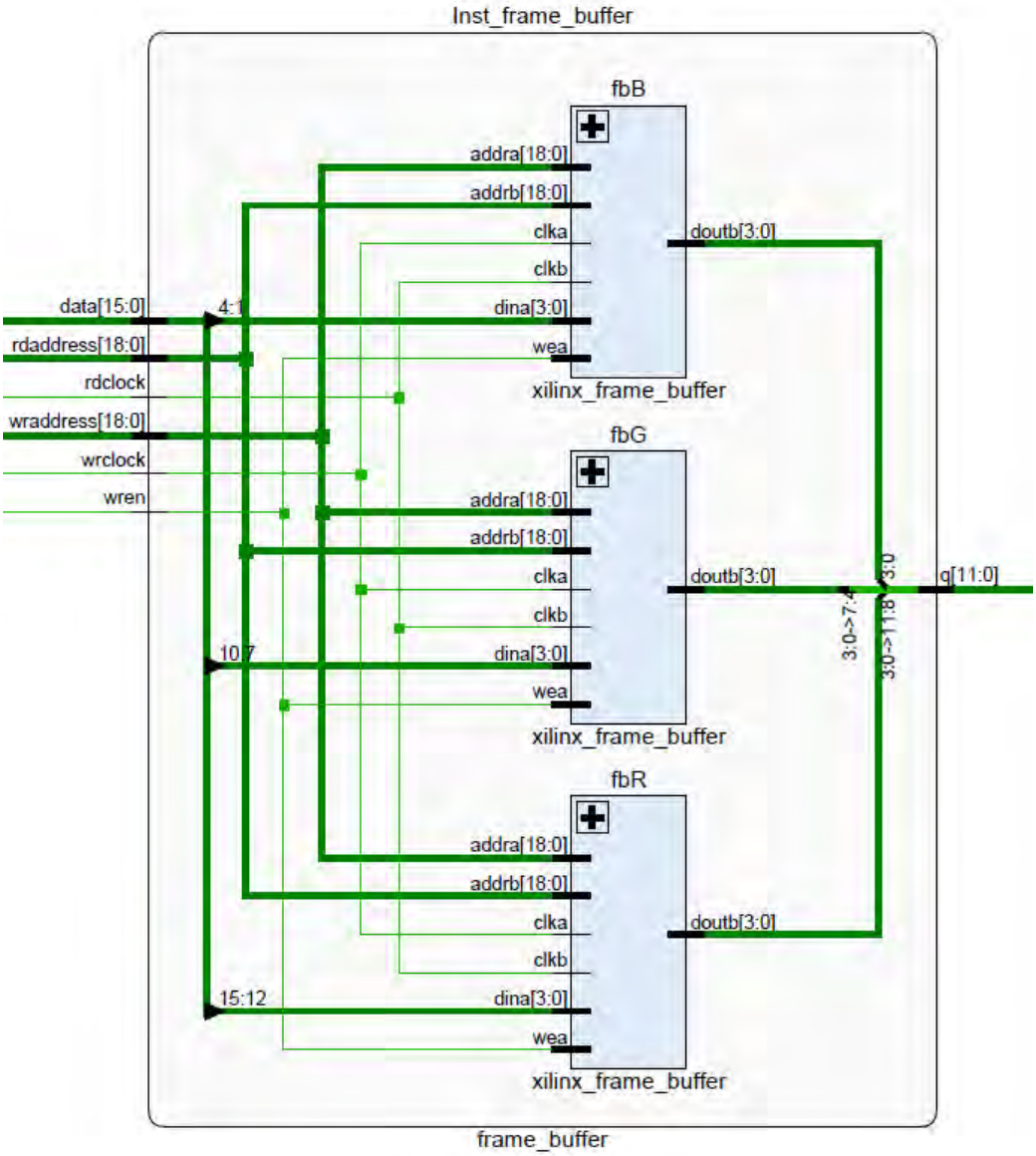

Figure 3.6: RGB444 Color Pixel Stored in Frame Buffer

3.2 Skin Detection

Many different types of color models are used for skin detection. Each one differs from the others in terms of the manner of transformation (linear or non-linear), the robustness to adapt to light changing, and shadow noises. Converting the skin pixel information to the

modified YUV color space would be more advantageous since human skin tones tend to fall within a certain range of chrominance values (i.e. U-V component), regardless of the skin type. You can see these ranges and different skin tone samples in the Figure 3.7:

45 34 30	#2D221E
60 46 40	#3C2E28
75 57 50	#4B3932
90 69 60	#5A453C
105 80 70	#695046
120 92 80	#785C50
135 103 90	#87675A
150 114 100	#967264
165 126 110	#A57E6E
180 138 120	#B48A78
195 149 130	#C39582
210 161 140	#D2A18C
225 172 150	#E1AC96
240 184 160	#F0B8A0
255 195 170	#FFC3AA
255 206 180	#FFCEB4
255 218 190	#FFDABE
255 229 200	#FFE5C8

Figure 3.7: Different Skin Tone Samples

The YUV color space was chosen due to the fast transformation of the RGB model. The Y channel represents the luminance of the color, while the U and V channels represent the chrominance. Separating the luma from the chromatic reduces the effect of light changing and shadow noises. This method starts by converting the RGB color space to the YUV color space using the following equations [3]:

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} +0.299 & +0.587 & +0.114 \\ -0.147 & -0.289 & +0.436 \\ +0.615 & -0.515 & -0.100 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

After skin pixels are converted to the modified YUV space, the skin pixels are segmented based on the following thresholds for both YUV and RGB color spaces:

$$\begin{cases} 80 < U < 130 \\ 136 < V < 200 \\ V > U \\ R > 80 \;\&\; G > 30 \;\&\; B > 15 \\ |R - G| > 15 \end{cases}$$

$$\begin{cases} R > 95 \,\&\, G > 40 \,\&\, B > 20 \\ \max(R, G, B) - \min(R, G, B) > 15 \\ |R - G| > 15 \,\&\, R > G \,\&\, R > B \end{cases}$$

The following flowchart shows the process of skin color segmentation:

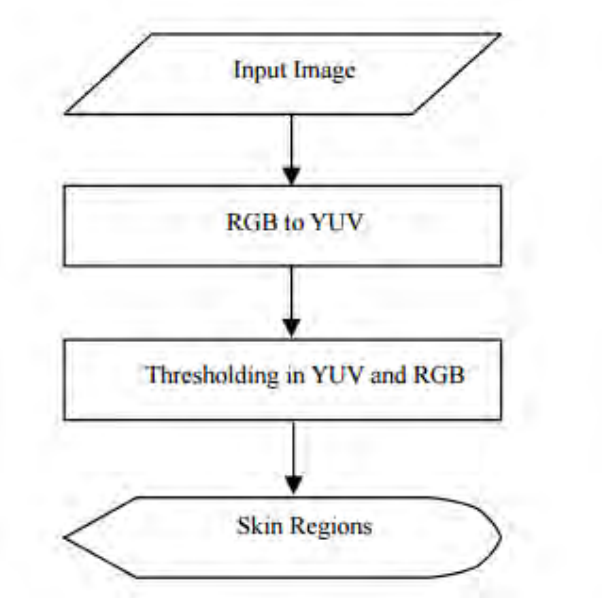

Figure 3.8: Skin Segmentation Process [3]

The RGB color signals deriving from the camera capture are consisted initially by 16 bits (5 bits for Red, 6 bits for Green, 5 bits for Blue). Because in the upper flowchart the calculations need 24 bits RGB color signals (8 bits for Red, 8 bits for Green, 8 bits for Blue), we have to shift left the initial RGB signals in order to have the required sizes. The outcome of this approach (having less resolution than needed) results in having multiples of 4 for Green and 8 for Red and Blue, which did not harm the efficiency of our system.

This proposed method is proved to detect skin regions with low false positive and false negative rates and at a high detection rate. In order to implement operations with floating point numbers, at first we had to multiply each of the floating point numbers with 256 (this selected number gave us a good estimation on the desired accuracy), which actually means that the floating point number is shifted left 8 times and then proceed with the rest of the calculations. The result of these operations was again divided with 256 (right shift 8 times) in order to estimate the appropriate YUV ranges. Ranges from the RGB color space are also applied to achieve the final result with raw skin and non-skin pixels.

Here are examples of how the operations with floating numbers are implemented for the U and V component:

*U = -0.147*R -0.289*G +0.436*B + 128 = (-0.147*256*R -0.289*256*G +0.436*256*B) / 256 + 128 =>*

*U = ((-0.147<<8)*R + (-0.289<<8)*G + (0.436<<8)*B) >> 8 + 128 =>*

*U ≈ (-38*R - 74*G + 112*B) >> 8 + 128*

*V = +0.615*R -0.515*G -0.100*B + 128 = (157.44*R - 131.84*G - 25.6*B) / 256 + 128 =>*

*V ≈ (158*R - 132*G - 26*B) >> 8 + 128*

38R = {R,5'b0} + {R,2'b0} + {R,1'b0}

74G = {G,6'b0} + {G,3'b0} + {G,1'b0}

112B = {B,6'b0} + {B,5'b0} + {B,4'b0}

158R = {R,7'b0} + {R,4'b0} + {R,3'b0} + {R,2'b0} + {R,1'b0}

132G = {G,7'b0} + {G,2'b0}

26B = {B,4'b0} + {B,3'b0} + {B,1'b0}

As shown from the equations above, no multiplication operations were applied. All calculations resulted from fast operations, i.e. shifts and additions.

The outcome of these calculations for the predefined ranges is a binary image with raw segmentation result. Now every pixel is assigned a single bit value (1 if it is considered a skin pixel and 0 for non-skin pixel). Every pixel of the image after thresholding is driven to the filters for processing and it is stored to block RAM ip Core namely skin_frame_buffer, in order the frame to be displayed on the monitor as a black and white image.

In the following diagram an illustration of the skin detection implementation is shown:

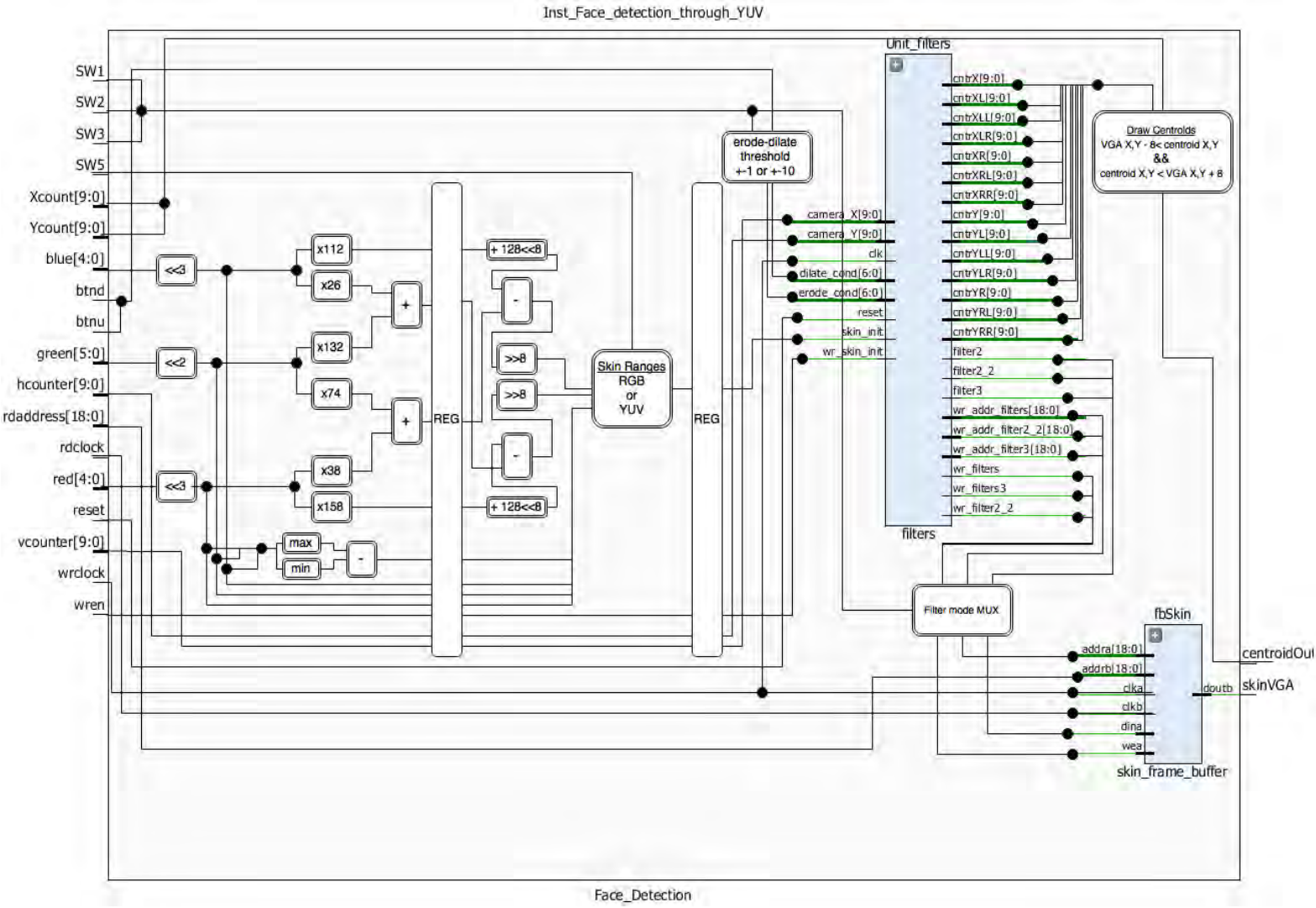

Figure 3.9: Block Diagram of Skin Detection Implementation

3.3 Spatial Filters

A spatial filtering on image is a neighborhood operation which changes the value of any pixel by a predefined function of the values of pixels in a neighborhood of that pixel. Spatial filtering can be used to perform some of the significant image operations such as edge enhancement, image sharpening and noise reduction. These operations can be linear or nonlinear. If the operation performed on the pixels are linear then the corresponding filter is called a linear filter otherwise it is a nonlinear filter. In our design we have implemented only linear spatial filters.

After thresholding, the resulting data needs to be stored in case we want it displayed to the monitor; hence it can be stored in 'skin_frame_buffer' according to the switches of the FPGA. Here output is written to the block RAM according to write address ("wraddress"), i.e. valid pixel's address that originates from camera and then the stored data is read according to read address ("rdaddress"), i.e. address for displaying the pixels on VGA monitor. Both the addresses are 19 bit vectors to address 640x480 = 307200 number of pixels.

3.3.1 Erode Filter

This step is similar to the erosion operation used in the software algorithm, which basically shrinks the object [19]. According to this method, for every pixel p, its neighboring pixels in a 9x9 neighborhood are checked. If more than 74% approximately, i.e. 60 of its neighbors are skin pixels, then p is declared a skin pixel. Otherwise p is declared a non-skin pixel. This threshold can be modified using the buttons of the FPGA board as shown in Table 2.4. This stage allows most background noise to be removed because noise is usually scattered randomly through space.

To examine the neighbors around a pixel, their values needed to be stored. Therefore, ten shift registers were created to buffer the values of ten consecutive rows in each frame. As seen in Figure 3.10, each register is 640-bit long to hold the binary values of 640 pixels in a row. Each bit in data_reg1 is updated according to the X coordinate. For instance, when the X coordinate is 2, data_reg1[2] is updated according to the result of thresholding from the previous stage. Thus, data_reg1 is updated every clock cycle with the values of a new line. When all the bits of data_reg1 are updated, its entire value is shifted to data_reg2, while the bits of data_reg2 are shifted to the next register in a sequential manner. Thus, other registers (from data_reg2 to data_reg10) are only updated when the X coordinate was 0. Values of data_reg2 to data_reg10 are used to examine a pixel's neighborhood [4].

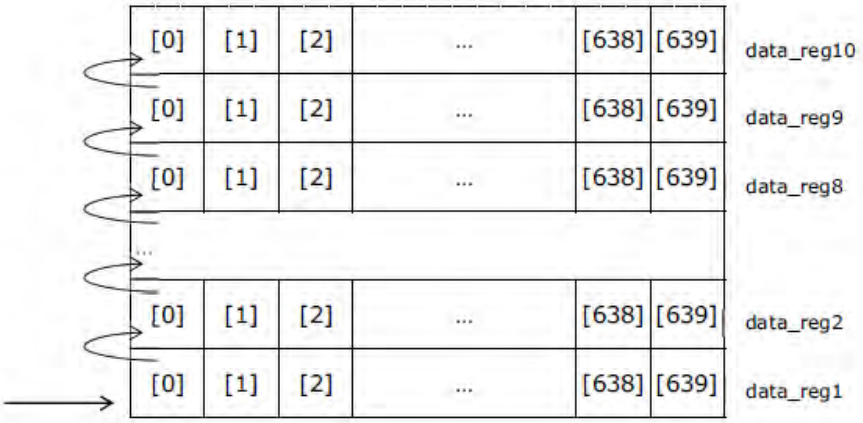

Figure 3.10: Ten Shift Registers for Ten Consecutive Rows [4]

There was a trade-off between the number of shift registers being used (i.e. the size of the neighborhood) and the performance of the spatial filter. A larger neighborhood required more registers to be used but, at the same time, allowed more noise to be removed. The resulted image after erosion is stored in simple dual port block RAM 'skin_frame_buffer', the same used for storing data after skin detection through YUV algorithm.

3.3.2 Dilate Filter

This step is similar to the dilation operation used in the software algorithm, which basically dilates the object by filling small gaps created during the previous steps [19]. Considering that most of the noise is now extracted from the image after dilation filtering and in saving resources from the ZedBoard, we used the same technique as mentioned before with the ten shift registers, where each register is now 320-bit long to hold the binary values of all the odd columns by applying a bitwise 'or' operation between the value of the odd column's pixel and the value of the previous pixel.

According to this method, for every pixel p, its neighboring pixels in a 9x5 (5 out of 9) neighborhood are checked. If more than 33.3% approximately, i.e. 15 of its neighbors are skin pixels, then p is declared a skin pixel. Otherwise p is declared a non-skin pixel. This threshold can also be modified using the buttons of the FPGA board as shown in Table 2.4. This stage allows some of the missing pixels of the detected face regions to be filled in.

The process remains the same as with the erode filter with the ten shift registers and their operation based on the neighboring pixels of every pixel. The resulted image after dilation is also stored in simple dual port BRAM 'skin_frame_buffer', the same used for storing data after skin detection through YUV algorithm and erosion filtering.

3.4 Temporal Filter

Even small changes in lighting could cause flickering and made the result displayed on the VGA screen less stable. Applying temporal filtering allowed flickering to be reduced significantly. In order to implement the functionality of such filter, we store in a simple dual port BRAM 'history_frame_buffer' the binary value of each pixel from three (3) consecutive frames as it results after dilation filtering for all the 320x480 pixels. Therefore, a 3-bit vector used as 3 history bits is stored in each pixel address of the 320x480 pixels of the 'history_frame_buffer', which corresponds to the value of the pixel in the three previous frames.

In order a pixel to be considered non-skin all the values of the 3 history bits and the current value of it should be '0'; else the pixel is considered a skin-pixel. This assumption relies on the fact that during the two spatial filters most of the noise is now reduced and these regions of pixels are truly skin pixels. Consequently, the temporal filter smoothed the output and, thus, reduced flicker significantly, although it creates a sense of slow motion in the displayed image.

In the Figure 3.11 a block diagram for the implementation of the pipeline for the filters, both spatial and temporal, is shown:

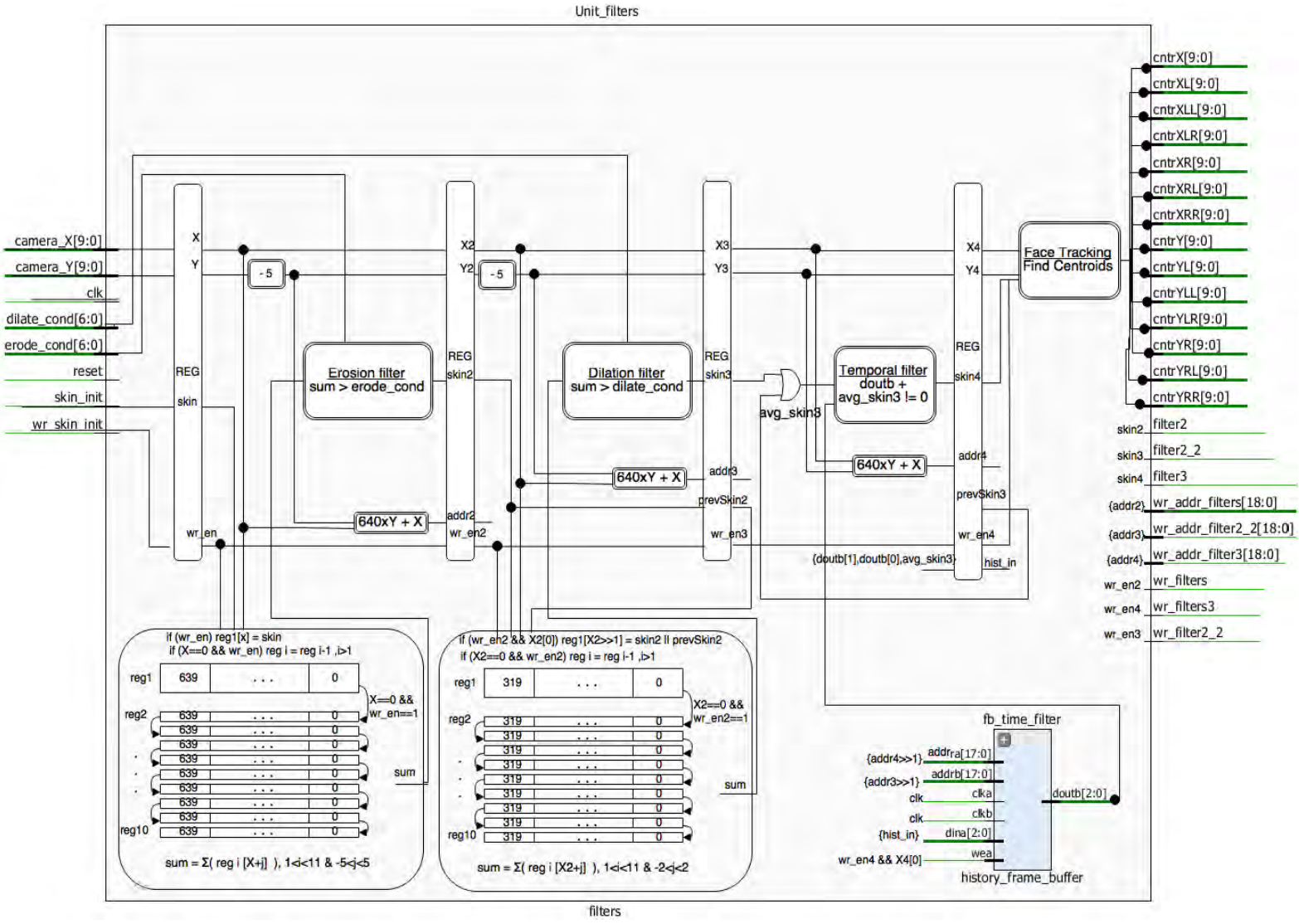

Figure 3.11: Pipeline of Spatial & Temporal Filters

3.5 Tracking of the Detected Faces

Finally, centroid is computed to locate the face region. Because it was infeasible to calculate the centroid for each face region separately, in our design the maximum number of faces that can be detected is seven (7). In order to achieve that, we created seven detectors. The detectors can be divided in 3 categories according to their priority. The Center detector

has the greatest priority among the others and the average X and Y coordinates of the detected face arise almost from the whole frame (purple ranges in Figure 3.12). The detectors with the next most significant priority are L and R and arise from the yellow ranges as shown in Figure 3.12. The L detector can detect a face on the left side of the face that is detected from the Center detector (Avg_X). Accordingly, the R detector detects a face on the right side of the face that is detected from the Center detector (Avg_X). The detectors with the lowest significance LL, LR, RL and RR can detect faces from the regions that are separated from the 3 detectors L, Center and R (LL left from L, LR right from L, RL left from R and RR right from R) and arise from the red ranges as shown in Figure 3.12.

After the initial step for finding the average X and Y coordinates for all the detectors, as shown in Figure 3.12, a counter named 'threshold' is created for each detector showing how many skin pixels are near (dashed ranges as shown in Figure 3.12) to each average X and Y. For the three most significant detectors (L, R and Center) a new average Y is calculated from its previous value up to the top of the image giving importance to the upper pixels, due to the assumption that the head is higher than other skin regions, e.g. arms and hands. Finally, the X and Y centroids' coordinates for each detector will be activated if the analogous threshold is big enough and at the same time if there are detectors with greater priority, they have to be de-activated or be far away from this specific detector. If a detector is de-activated its X and Y centroids' coordinates are set to zero.

Finally, in "Face_Detection" module for each detector's coordinates X and Y, if their value is greater than zero that subsequently implies that a face is detected by this detector. For this reason, when X and Y coordinates coming from VGA controller are placed in a neighbor 15x15 with center the centroid of any detector then a signal "centoidOut" is triggered in order this neighbor to be drawn red. This denotes the presence and tracking operation of a detected face.

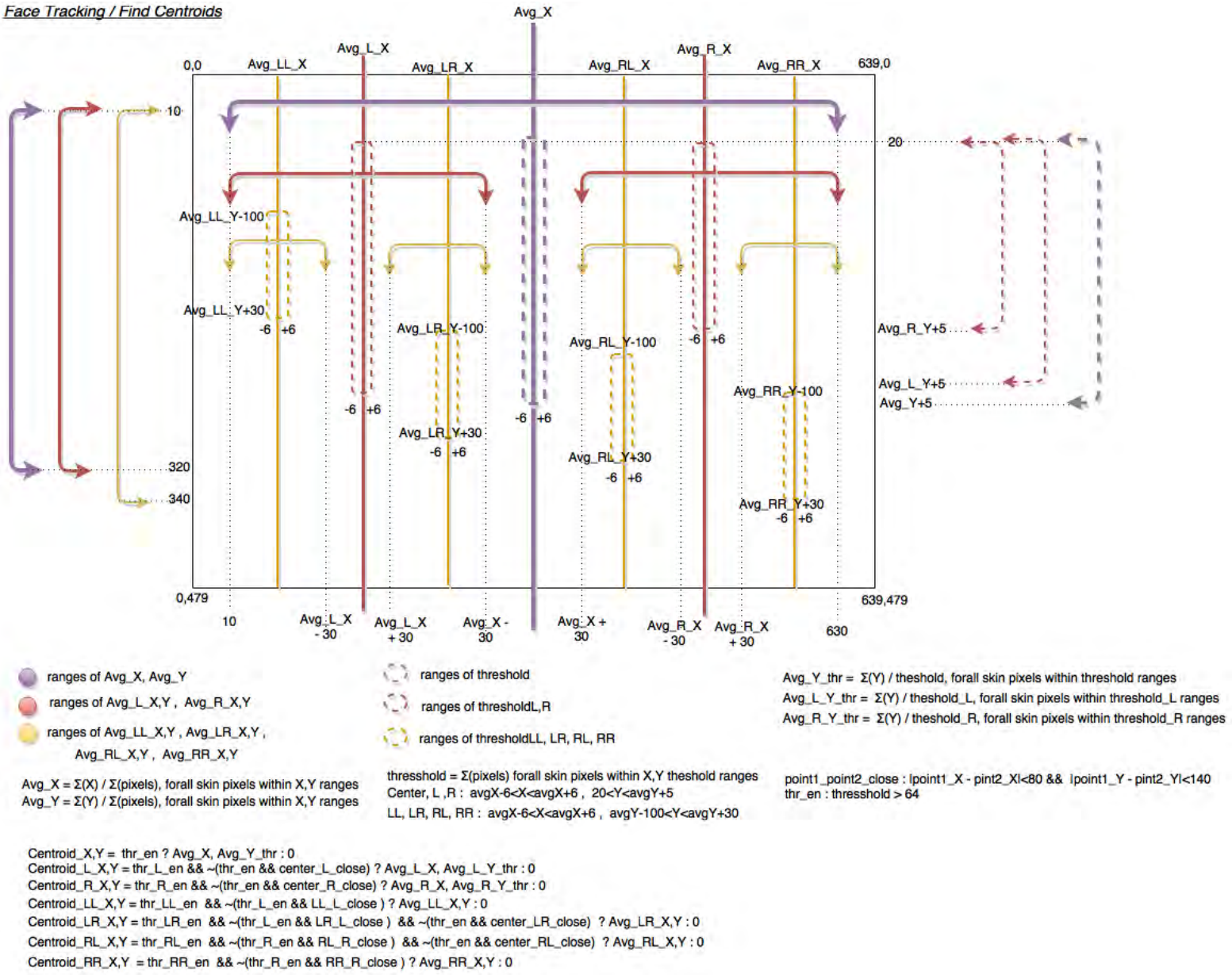

Figure 3.12: Computing Centroids for Face Regions

Chapter 4

Conclusion

In recent years, face detection has attracted much attention and its research has rapidly expanded by not only engineers but also neuroscientists, since it has many potential applications in computer vision communication and automatic access control system. However, face detection is not straightforward because it has lots of variations of image appearance, such as pose variation (front, non-front), occlusion, image orientation, illuminating condition and facial expression [20].

In our design we tried to create a functional system fully implemented in hardware, which is capable to detect and track faces in real time. Although the tracking capability of the algorithm is limited up to 4 faces and in ideal circumstances can reach a limit of 7 faces, it can still detect the presence of their faces. Experiments also showed that different light settings did not significantly alter the final results. Furthermore, the system was able to ignore background noise very well—mostly came from light reflection. When there were objects that had color similar to skin color, both spatial and temporal filtering helped erode these detected regions, therefore reducing the number of false positives.

4.1 Results

In the following images we present the functionality and the results of our system in real time video streaming:

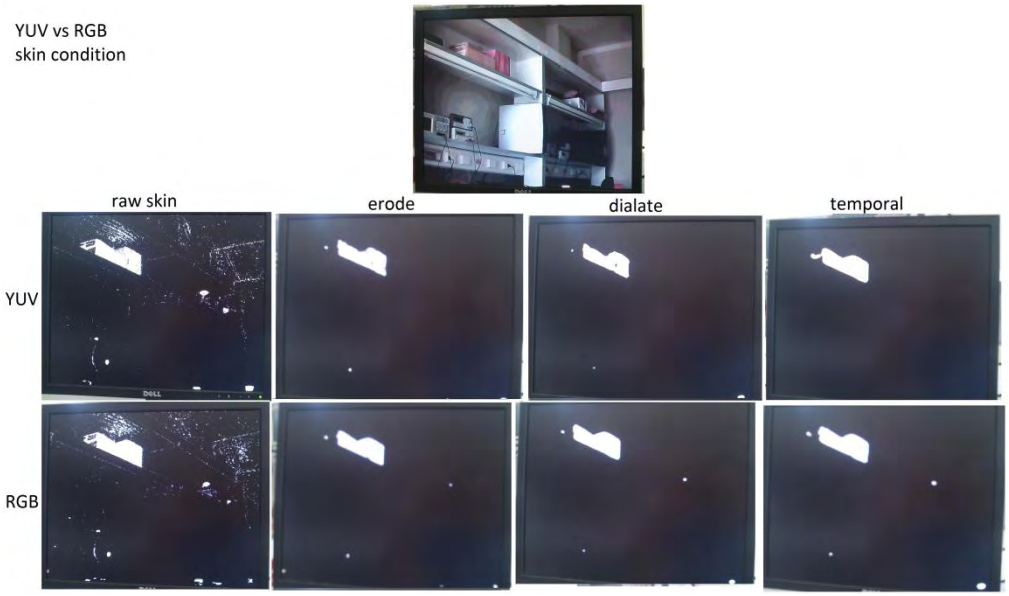

Figure 4.1: YUV vs RGB skin ranges

In Figure 4.1 we show the different stages of the algorithm for both YUV and RGB skin thresholding from skin detection to the erode, dilate and finally temporal filter, where no face is present. An object, however, with color similar to the skin is still detected.

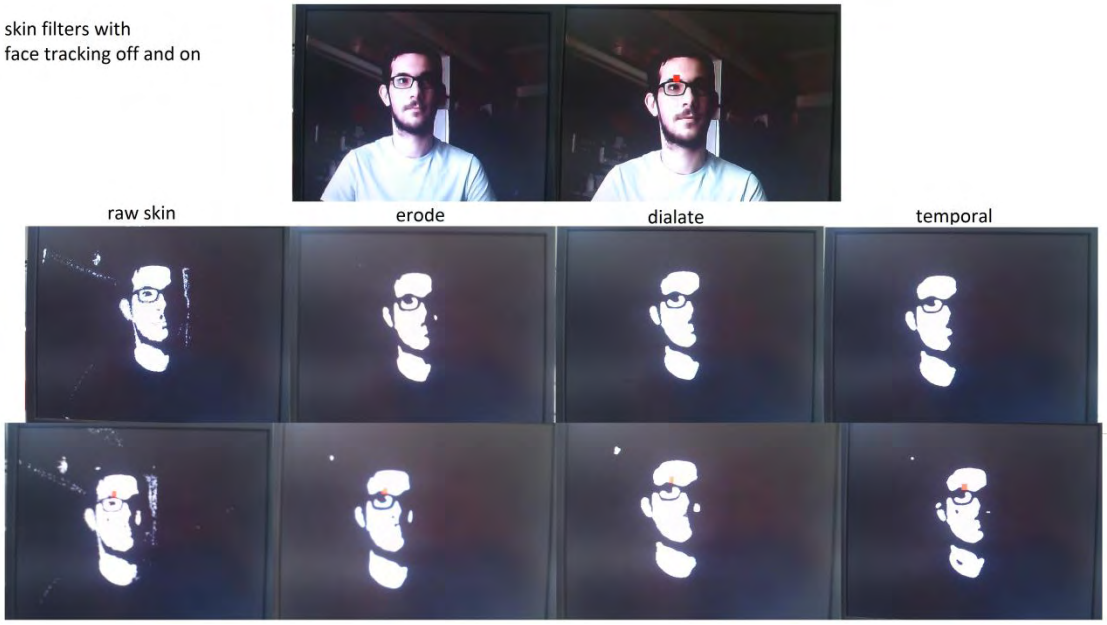

Figure 4.2: One Face Present

In Figure 4.2 we show the different stages of the face detection system for both YUV and RGB thresholding with the presence of one person.

Figure 4.3: Face Detection and Tracking with FP Faces

In Figure 4.3 different cases are shown where two faces are correctly and incorrectly detected. In the case of correct detection there are indeed two faces. In the other case a FP (False Positive) detection and tracking occurs in the presence of a box with color similar to human's skin.

Figure 4.4: Face Detection and Tracking for 3-5 Faces

In Figure 4.4 it is shown the functionality of the algorithm for 3, 4 and 5 present faces, which are all properly detected and tracked.

4.2 Project Report

The project was described on both VHDL and Verilog HDL and after its synthesis to a netlist file was mapped to the FPGA resources. The final device utilization is available in Table 4.1 below:

Resource	Utilization	Available	Utilization %
FF	11817	106400	11.11
LUT	31636	53200	59.47
I/O	46	200	23.00
BRAM	136.5	140	97.50
BUFG	4	32	12.50
MMCM	1	4	25.00

Table 4.1: The resources report for the final version of the project

4.3 Problems & Solutions

One of the first and most crucial problems we encountered was the quality of the camera output. More specifically, the image was blurry and the colors were distorted, and as a result we were not able to distinguish the displayed objects. The following image shows the displayed frame on the monitor:

Figure 4.5: Initial Image Displayed on Monitor

In order to face this problem we had to change and experiment on the camera configurations and thus we created reconfigurable configurations giving the user the possibility to change the default characteristics and operations according the imposed needs.

Another issue was the limited resources of the FPGA board and especially the BRAMs, which resulted in storing the half frame for the temporal filter. Moreover, in saving resources we could only create seven detectors for the centroid computations as an optimal, yet functional solution.

4.4 Future Work

In this project, the goal of implementing a hardware system to detect and track human faces in real time was achieved. A software implementation of the algorithm was examined in order to study, transform and finally implement the functionality of the OpenCV functions erode() and dilate(). Although the transition from software to hardware required some modification to the original algorithm, the initial goal was still accomplished. The face detection algorithm was derived from a skin detection method. Face tracking was achieved by computing the centroid of each detected region, although it only worked in the presence of at most 4 to 5 people. Different types of filter were applied to avoid flickering and stabilize the output displayed on the VGA screen. The system was proved to work in real time with no lagging and under varying conditions of facial expressions, skin tones, and lighting.

However, a future improvement of the design could result in tracking accurately more than 4 faces with the implementation of a more efficient tracking algorithm. That would probably require the displayed frame be segmented in more sections.

Another improvement would be the application of a morphological filter in order to extract large regions of skin pixels that are falsely detected as faces. That would reduce the ratio of false positives and will result in a more effective algorithm. Yet such implementation would require more resources currently unavailable in the used ZedBoard edition.

Bibliography

[1] **OV7670 Camera Design**
http://hamsterworks.co.nz/mediawiki/index.php/OV7670_camera

[2] **OV7670 Camera Module attached to ZedBoard using PMOD connectors**
http://hamsterworks.co.nz/mediawiki/index.php/Zedboard_OV7670

[3] **Skin Segmentation Using YUV and RGB Color Spaces**
http://jips-k.org/dlibrary/JIPS_v10_no2_paper9.pdf

[4] **Real-Time Face Detection and Tracking**
http://people.ece.cornell.edu/land/courses/eceprojectsland/STUDENTPROJ/2012to2013/tnn7/tnn7_report_201212141110.pdf

[5] **Real Time Implementation of Spatial Filtering on FPGA**
http://www.academia.edu/10478288/REAL_TIME_IMPLEMENTATION_OF_SPATIAL_FILTERING_ON_FPGA

[6] **Field Programmable Gate Array (FPGA)**
http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm

[7] **OV7670 CMOS VGA (640x480) CAMERACHIP™ Sensor Datasheet by OmniVision**
http://www.cutedigi.com/pub/sensor/Imaging/OV7670-Datasheet.pdf

[8] **OV7670 Software Application Note – Camera Configurations**
http://wenku.baidu.com/view/aab1f11cc281e53a5802ffe4.html

[9] **FPGA Implementation of VGA Controller**
http://www.researchgate.net/publication/233743657_FPGA_IMPLEMENTATION_OF_VGA_CONTROLLER

[10] **VGA Signal 640 x 480 @ 60 Hz Industry standard timing**
http://tinyvga.com/vga-timing/640x480@60Hz

[11] **Hacking the OV7670 camera module (SCCB cheat sheet inside)**
http://embeddedprogrammer.blogspot.jp/2012/07/hacking-ov7670-camera-module-sccb-cheat.html

[12] **Field-programmable gate array**
https://en.wikipedia.org/wiki/Field-programmable_gate_array

[13] **VGA Controller (VHDL)**
https://eewiki.net/pages/viewpage.action?pageId=15925278

[14] **ZedBoard**
http://zedboard.org/product/zedboard

[15] **Face Detection using Skin Color Model and Distance between Eyes**
http://www.warse.org/pdfs/ijccn01132012.pdf

[16] **Linux Drivers – OV7670 Camera Configurations**
http://lxr.free-electrons.com/source/drivers/media/i2c/ov7670.c#L167

[17] **OmniVision Serial Camera Control Bus (SCCB) Functional Specification**
http://www.ovt.com/download_document.php?type=document&DID=63

[18] **Piping OV7670 video to VGA output on ZYBO**
https://github.com/laurivosandi/hdl/blob/master/zynq/zybo-ov7670-to-vga.rst

[19] **Morphological Transformations**
http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

[20] **Face detection**
https://web.stanford.edu/class/ee368/Project_03/Project/reports/ee368group02.pdf

[21] **Video Graphics Array – Wikipedia**
https://en.wikipedia.org/wiki/Video_Graphics_Array