An Image Based Colorimetric Technique for Portable Blood Gas Analysis

By,

Jaideep Chandran

Submitted in Fulfillment of the Requirement for

The Degree of Doctor of Philosophy

School of Engineering and Science

Faculty of Health, Engineering, and Science

Victoria University

PO Box 14428

Melbourne City MC

Victoria, Australia, 8001

Table of Contents

Declaration of Originality

"I, Jaideep Chandran, declare that the PhD thesis entitled "An Image based Colorimetric Techniques for Portable Blood Gas Analysis" is no more than 45,000 words in length, exclusive of tables, figures, appendices, references and footnotes. This thesis contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma. Except where otherwise indicated, this thesis is my own work".

Signature: *Jaideep Chandran* Date: 16/04/2012

Acknowledgement

I would like to begin by thanking God for everything He has provided me in this life and continuously showering his love and blessing on me.

I thank my parents, Mr Chandran Srinivasan and Mrs Rajalakshmi Chandran, my brother Mr Gurucharan Chandran and his wife Rupal and my nieces Riya,Rati and Reeti, my uncle Vijay Raghvan Srinivasan for their unconditional love and support to me.

I would like to express my deepest regards and appreciation for my primary supervisor, Professor Aleksandar Stojcevksi for providing me with the opportunity to do my PhD at Victoria University and agreeing to guide me through the research degree. He has provided me with guidance, advice, motivation and support in the matters of research, work and also life. Our relationship has grown from a student and mentor to a wonderful friendship and would like to continue to be so.

I would like to thank my associate supervisors Associate Professor Aladin Zayegh and Dr.Thinh Nguyen for supporting me and providing their valuable time and advice. Their skills and experience as researchers have helped me solve many problems in this research. I would like to express my deepest regards for Dr Manoj Durairaj for his expert guidance in helping me choose the field of application for thesis.

To my colleagues Dr. Ronny Veljanovski, Dr. David Fitrio, Dr. Anand Mohan, Dr. Eder Kikianty, Nikhil Joglekar, Hayrettin Arisoy, Mark Mojic and all members of the coffee club I thank all of you. You have made my life in Victoria University very colourful and gave me a sense of belonging. David, Anand and Eder special thanks for all the advice and help provided during the project. A special mention to Nikhil for being my friend and guide since the time we started on this journey from our days in the Masters of Microelectronics Engineering in 2002.

My gratitude also goes to Victoria University staff members, Professor Stephen Collins, Professor Mike Faulkner, Shukonya Benka, members from the chemistry lab in building G level 6 and more for their support and help. Special thanks to Abdul Hadbah, System Administrator, without his support there would be no EDA tools to work on.

I would also like to thank my friends and flatmates Bhupesh, Chetan, Amit, Kalyani, Aditi, Anagha and Krupa, for their support and friendship. The endless parties and get together helped balance the work and life and de-stress during crucial occasions.

A special mention for Robin Kalia, Omprakash Ola, Roshan Kushwaha and Tahir Mehmood for their help and support. Thank you.

Last, but not least, I would like to thank my wife Lavanya and son Ishan for their unceasing love, support and faith in me. I would like to thank them for their patience during this period of my PhD and motivating me during difficult times.

List of Figures

List of Tables

List of Acronyms

ABG	Arterial Blood Gas
AlGaN	Aluminium Gallium Nitride
CAT	Computed Axial Tomography
CCD	Charge Coupled Device
CMOS	Complimentary Metal Oxide Semiconductor
CO_2	Carbon Dioxide
GC	Gas Chromatography
Hb	Haemoglobin
HCO_3^-	Bicarbonate Ion
H_2CO_3	Carbonic Acid
IC	Integrated Circuit
ISFET	Ion Sensitive Field Effect Transistor
MEMS	Micro Electro Mechanical Systems
MRI	Magnetic Resonance Imaging
MS	Mass Spectrometry
pO_2	Partial Pressure of Oxygen
pCO_2	Partial Pressure of Carbon dioxide
REFET	Reference Field Effect Transistor
RGB	Red-Green-Blue
VHDL	Very High Speed Integrated Circuit Hardware Description Language

List of Publications

1 N. Joglekar, J. Chandran, R. Veljanovski, A. Stojcevski, A. Zayegh, and M. Dorairaj, "A Review of Blood Gas Sensors and a Proposed Portable Solution," in *Integrated Circuits, 2007.* ISIC '07. International Symposium on, 2007, pp. 244-247.

2 Jaideep Chandran, Robin Kalia, Alex Stojcevski, Thinh Nguyen and Aladin Zayegh, "A Novel Current to Voltage Converter for a Portable Blood Gas Analyser", in MS'08 JORDAN, *Proceedings of The International Conference on Modelling and Simulation*, Petra, Jordan, 18-20 November, 2008

3 Jaideep Chandran, Omprakash Ola, Alex Stojcevski, AladinZayegh, Thinh Nguyen, "Implementation of a colorimetric algorithm for an image based pH analyser", in *Proceedings of the 2009 IEEE Regional Symposium on Micro and Nano Electronics (RSM09),* Kota Bahru, Malaysia, 2009, 319-322.

4 Jaideep Chandran, Alex Stojcevski, AladinZayegh, Thinh Nguyen, "Mathematical model for a colour based pH analsyer", in *Proceedings MS09 India – International Conference on Modelling and Simulation*, Trivandrum,India, 2009, 117-120.

5 J. Chandran, A. Stojcevski, A. Zayegh, and N. Thinh, "Implementation of a colorimetric algorithm for portable blood gas analysis," in Microelectronics (ICM), 2010 International Conference on, pp. 411-414.

6 Jaideep Chandran, Alex Stojcevski, AladinZayegh, Thinh Nguyen "A Colorimetric algorithm for portable blood gas Analysis", Submitted AMSE Journal Feb 2010, Accepted to be published.

Abstract

The field of electronics has been assisting medical practitioners with the diagnosis of medical conditions and also aiding in therapy. The joint fields of biology, medicine and electronics have given rise to the field of biomedical electronics. Biomedical electronics has helped develop sensors to measure biological parameters both invasively and no invasively. These sensors provide the doctors with critical and accurate information on the patient's heath which allows them to do a proper diagnosis and prevent a serious health condition and determine the most suitable treatment.

Many of the instruments require the patient to be present at a clinic for data recording. This constrains the data collected by the instrument and the patient to the clinic. The patient cannot be monitored continuously and in normal settings, where subtle changes in the behavior of their body can be recorded, which will give the doctor a better understanding of their condition. The focus of research is now moving towards wearable sensors and portable equipment. This is due to change in the approach of health care which has moved from being "hospital centred" towards being "citizen centred" and more effort is being put towards prevention and early risk detection.

Blood gas analysis is an important part of a doctor's diagnosis; it primarily consists of the partial pressure of oxygen, partial pressure of carbon dioxide and the pH. It provides the medical practitioners with insight into the patient's respiratory health, the metabolic activity in the body and the health of the renal system.

Blood gases have been measured predominantly either by electrochemical or optical systems. The electrochemical systems, though accurate, are big and bulky and require mains power to operate.Optical systems are smaller but do the measuring intravenously. Thus the mains power operation and the need to measure intravenously using the current solutions for blood gas analysis hinder portability. This thesis discusses the design and development of a portable blood gas analyser with clear emphasis on size, power consumption and cost.

A novel approach using an image based colorimetric technique where an algorithm detects the change of colour of indicator to measure the value of pH and pCO_2 is proposed. The principle of this approach is as follows: the change in colour of the indicator is imaged and

the colour of the indicator is quantified; the colour of the indicator is plotted against the change in pH. The relation between the colour of the indicator and the pH is then mapped mathematically. The experiments conducted to map the relation of the colour of the indicator to pH value show the colour red reacts the sharpest to the change in pH and can be described by a fifth order polynomial equation.

The emphasis of the thesis is based on the development of the colorimetric technique to measure the pH and pCO_2 and the design and implementation of the colorimetric algorithm in hardware. The colorimetric algorithm is implemented using floating point arithmetic architectures for high accuracy with an emphasis on low power consumption and area.

The results from the implementation of the colorimetric algorithm show the deviation from the observed results for the mathematical equation for pH is ±1% and for the pCO_2 equation the deviation is ±10%. The ASIC synthesis results for a 0.35µ CMOS process show the design meeting the timing constraints of the design and a power consumption of 495mW at an operating speed of 50 MHz. The design is a simple construction and the entire processing unit can be fitted on to a single chip with only the camera as an off chip component. The final results indicate and substantiate the suitability of the colorimetric technique for portable blood gas analysis.

Chapter One: Introduction

1.1 Background of this research

The field of electronics has made a major impact on human life. The advent of the transistor allowed engineers to build a range of products like the radio and television in more compact ways. Further progress in reducing the size of the transistor and getting more than one transistor on a silicon die gave birth to the Integrated Circuit(IC). The IC revolution brought in the development of the digital gates the basic building blocks of the modern processor. The computing power of the processor has allowed designers to build new products in the fields of communication, entertainment and home appliances. The field of medicine has also benefited from improvements in electronics with the advent of medical diagnostic equipment to aid the medical practitioners. They have provided medical practitioners with a holistic view and in depth analysis of a patient's health condition to make informed decisions with regards to treatment required. The parameters include body temperature, blood pressure and heart rate. Advanced imaging techniques like Computed Axial Tomography (CAT) scan and Magnetic Resonance Imaging (MRI) have helped them understand the damage caused to the tissues and various body part due to injuries and stress. Sensors have also been used invasively and noninvasively to access such parameters. The collaboration between engineers and medical practitioners coming together to build and develop new solutions for diagnosis and health care has given rise to the field of Biomedical Electronics [1].

Diagnostic instruments have been traditionally confined to the hospitals or large clinics thus requiring the patient to be present in the clinic or the hospital in order for measurements to be taken. This constrains the data collected to only the clinical environment. Medical doctors are looking to monitor the patient's health continuously in their day to day environmentfor a better understanding of their health condition and how it is affecting them. It also allows them to monitor if the prescribed treatment is working for the better or worse [2, 3].

There has been a rise in the aging population and the number of chronic diseases. This has resulted in a rise of people afflicted by these diseases. The demand for health services and

the service providers is also increasing daily. Reliable medical services in rural areas add further pressure on the health care professionals and the diagnostic equipment. These societal changes have changed the approach to medicine. It has changed from a hospital-centred approach to a patient-centred approach. The focus of medical practitioners has changed from treatment of after detection of chronic health conditions to prevention and early risk detection of these conditions and patient well being. Wearable sensors are able to provide the medical professionals with the data required [3, 4].

Advances in the field of electronics, optics, micro-electro-mechanical systems (MEMS) and telecommunications have aided in the development of wearable sensors. The sensors can measure the physical parameters of the body outside the hospital without constraining the patient to a bed and in some cases not requiring them to visit the hospital. The sensors are smart devices which sense the vital parameters [5, 6]. The sensors can also communicate with each other and the data can be transmitted to a central unit. The central unit collects the data and can communicate it to the medical professionals. This allows the medical doctors access to information on the individuals over an extended period of time and in different scenarios. It also allows forsensing and recording of sudden acute events which can trigger a series of events, sometimes even leading to fatality. The doctor is well informed by the data and thus is aided in making an informed decision [7].

Arterial blood gases are one of the important parameters that the medical professionals read. Arterial blood gase parameters consist of the partial pressure of oxygen, the partial pressure of carbon dioxide and the pH of the blood. The blood gas analysis provides the medical doctor with an insight to the health of the respiratory system and also the health of the kidneys and renal functions. Present day blood gas analysers are based on either electrochemical or optical principles to measure the blood gases. The electrochemical analysers are big and bulky and work on mains power supply. Optical analysers are smaller in size but still run on mains power supply and operate invasively to measure the blood gases. These features of the optical and electrochemical analysers limit their portability [8].

1.2 Motivation

The need for affordable and quality health care is one of the primary and most basic rights for all human beings. The complexity and cost of the diagnostic medical equipment have

affected the disbursement of these services. Many of the services are located in the cities and can be accessed only by people who can afford them.

The advancement in electronics has helped designers and engineers to develop new products which are smaller in size, easier to operate and low in power consumption. These advancements have helped in the development of sensor and diagnostic equipment which are smaller and cheaper. The design of wearable sensors and portable diagnostic equipment allows the medical doctor to collect data from the patient at their home or work and also to be taken along to rural areas during visits by the doctors [1, 7].

This research looks at an image based methodology to measure the blood gases using a colorimetric algorithm for portable blood gas analysis. The research involves circuit designs and techniques to reduce the size and power consumption and aid portability of the device.

1.3 Research Aims and Objectives

The objective of this research is to develop and implement a Colorimetric Algorithm for portable blood gas analysis. The specific aims for the research are given as follows:

- To investigate Blood Gas Analysers and to obtain blood gas analyser specifications
- To establish a relationship between the pH and colour
- To mathematically model the relationship between the pH and colour
- To implement the mathematical relation in CMOS
- To implement the CMOS Components
- To integrate and analyse the complete system

1.4 Research Design Methodologies and Techniques

The proposed specific research aims and design methodologies to achieve the aforementioned aims are as follows;

- **Investigation of Blood Gas Analysers and obtaining of specifications**

  A thorough investigation/review on the different blood gas analysersis carried out based on current analysers, and related material. Also a review in the physiology and

chemistry of blood gases is carried out in order to understand the phenomena and the techniques used to measure these gases. The review also looks at a suitable indicator for colour change. This is required as the aim of the research is to produce a colour algorithm to be used in a portable blood gas analyser. The design parameters for the analyserwere obtained from this review.

- **Establishing the relation between colour and pH**

In this research, the measurement of blood pH and pCO_2 is based on the principle of colour change. To establish the relationship between the changes in colour of the indicator pH, samples of known values were prepared and then mixed with the chosen indicator. The tests were conducted for indicators with various concentration and lighting conditions. The samples were imaged and the colour of the indicator was analysed. The colour of the indicator was measured in the primary colours Red Green and Blue (RGB). The tests provide the data to map out the relationship between the pH and the colour of the indicator.

- **Mathematical modelling of the relation between pH and colour**

The data collected from the test provided the trends for the RGB with the change in pH. The trends from the data were modelled using regression analysis to provide a mathematical relationship. The mathematical model is then tested to check for precision of the predictions.

- **Implementation of the mathematical relationship in CMOS**

Different architectures and algorithms for implementing mathematical functions were studied. The most suitable architecture in terms of accuracy and power was chosen and implemented. The architecture was also fully tested for accuracy. The mathematical relationship describing the partial pressure of carbon dioxide in terms of the pH was also implemented. The implementation was fully tested for functionality. The mathematical relationships are implemented in Very High Speed Integrated Circuit (VHSIC) Hardware Descriptive Language (VHDL).

- **Implementation of a Colour Extraction Algorithm**

A memory block holds the image of the sample and an algorithm to extract the values colour from the image is implemented. The mean value of the colour is then used to

calculate the blood gas values. The implementation in CMOS is done using VHDL. The algorithm is simulated to verify functionality.

The entire system including the mathematical relations, the memory block and the colour extraction algorithm were integrated and synthesized using the 0.35µ CMOS process using Cadence tools. The design was also implemented on the Xilinx Spartan3 Field Programmable Gate Array (FPGA) using the Xilinx ISE tool.

1.5 Originality of thesis

This research will be significant as a new technique for blood gas analysis will be developed. The colorimetric technique by its nature reduces the size of the device; the power consumed and reduces the cost of the device. The colorimetric technique contributes to new knowledge in following perspective;

- Power: The development of a colorimetric technique based on imaging reduces the implementation of the analyser to a single chip using CMOS technology. The architectures and the circuits used are low power components. The implementation has no moving parts or probes thus further reducing the power consumption.
- Weight: The design uses few external components and has no moving parts. The architecture is implementable on a single chip and reduces the size of the instrument. The weight of the instrument will be reduced significantly and aid the portability of the instrument.
- Functionality: The architecture and design use few components and the interface for the user is simple. The algorithm is designed for low power consumption and reduction in weight and size, which aids in the portability of the device. The ease of operation and portability enhance the functionality and use of the instrument.
- Cost: With implementation in CMOS technology, it is a low cost solution. This is because CMOS currently offers the most affordable implementation option in terms of operation and design.

1.6 Thesis Organisation

This thesis has been arranged in seven chapters. Following this introduction chapter, Chapter 2 presents a review of blood gas analysis. A description of the physiology of blood gases and the blood gas values are presented. The current state of the art techniques used to measure the blood gases and their shortcomings are also presented. Chapter 3 presents the research proposal and details the benefits of the techniques. The chemistry of pH and the different indicators to measure pH are presented in detail. The imaging technique used is also discussed. Chapter 4 describes the system development and the data acquisition. The chapter discusses the procedure to acquire images and the processing of the images to acquire the colour. It also discusses the analysis of the data acquired from the experiments. The curve fitting analysis and regression analysis applied to model the relationship between the colour of the indicator and pH is also presented. The hardware implementation strategies of the mathematical model developed are discussed in Chapter 5. The review of the architectures used to implement the mathematical model is presented. In addition the chapter also describes the implementation of the other component besides the mathematical model. Chapter 6 presents the design and implementation of the complete system and all results and the results for the ASIC and FPGA synthesis are presented. Chapter 7 concludes the research and presentsthe possibilities for future work following this dissertation.

1.7 Conclusion

This thesis is a study into integrated circuit design, optical design techniques targeting Bio-medical devices. The study is an attempt to address the challenges of portability and power consumption for blood gas analysis.

Chapter Two: Blood Gas Analysis

2.1 Biomedical Electronics

Electronics designers and medical practitioners have been working together to develop instruments to aid in diagnosis of medical conditions and in therapy. This has given rise to the field of biomedical electronics. The field of biomedical electronics has given new solutions for diagnosis and health care [1]. The use of sensors to measure the vital parameters of the body is not new; the electrocardiogram has been aiding the medical practitioners to give them a view of the workings of the human heart. It is used to note any abnormalities in its functioning. Similarly the electroencephalogram, electrooculogram and electromyogram allow the medical practitioner to have insight into the brain activity, health of the eyes and the muscular action respectively. The data collected from these instruments have helped doctors make more informed decisions about the individual's health. The miniaturisation of some instruments made possible by advances in electronics has made the instruments easier to operate [2, 3]. In many cases the patients can use the instruments themselves for self diagnosis. The glucose meter has helped many diabetic patients to monitor their blood sugar levels and keep them in control. Blood pressure and heart rate monitors using electronic sensors and digital readouts and their ease of operations has been used by the sports fraternity to monitor peak performance of the athletes. The development of these sensors and devices has allowed doctors to collect data outside the clinical environment [4, 7].

2.2 Prevention before Cure

The uses of new diagnostic equipment and easy to use instruments have helped the doctors but the majority of the medical procedures require the individuals to come to the hospital /clinic to get a check up done. This restricts the data collected to the clinical environment. The data collected in this environment may not reflect the true nature of the individuals' health as some patients have been shown to exhibit White Coat Syndrome where by the patient's exhibit symptoms as a result of the clinical environment e.g. a raise in the blood pressure; this hinders the decisions for diagnosis of the patient's health

condition [1, 6]. The condition of the patient being present in the clinic for diagnostic tests to be conducted requires infrastructure and adds to the costs of the tests. The cost of infrastructure also limits the location of clinics and hospital to only populated areas and around cities and towns; the rural areas are thus bereft of facilities to aid the medical professional.

The rise in fatalities due to chronic diseases and combined with the rise in the ageing population has changed the approach of medical professional. They are changing from a hospital centred approach to a more patient centred approach. Medical professionals are looking to gather data in the individual's day to day environment. The data collected gives a better idea of how the health condition affects them with changes in environment from home to work and also gives them a picture of how the prescribed treatment is working to change the health condition. The focus in the medical world has changed to prevention before cure. Doctors would like to monitor patients with chronic health conditions, with family histories of chronic diseases and the ageing population and keep track of their health and predict a serious health condition and take preventive action to avert the onset of medical problems. This has given rise to a demand in wearable sensors which are non obtrusive, small, accurate, portable and allow wireless communication of the data.

2.3 Home Care

The need to monitor the vital signs of the body has made engineers strive to design better diagnostic equipment. The change in approach of the medical professionals from a hospital centred approach to a patient centred approach has resulted in increased development of wearable sensors. Recent advances in the field of the electronics, optics and communication have allowed the realisation of wearable sensors for medical diagnostic purposes. Such sensors can measure the vital signs of the body and are non obtrusive. The advances in MEMS technology has aided in developing miniature pressure sensors, gyroscopes and gait analysis sensors [9]. These advances have aided in development of accurate blood pressure sensors, electrocardiograms which can be miniaturised and placed on the body. Gait analysis sensors can monitor movement of the individuals allowing detection of falls. The sensors consume very low power thereby reducing the frequency of which batteries need to be recharged. The sensors can be

embedded in clothing or placed as patches on the body. They can feature wireless communication; they can communicate among themselves and also communicate the data to a central server eliminating wires and the need for the devices to be physically connected to a computer for data to be downloaded. The data collected from the central server can be transmitted via the internet or mobile telephone to the medical practitioner [10]. Figure 2.1 shows an example of the wearable sensors and the communication interface. The medical practitioner can monitor the patient continuously and track the course of the treatment prescribed outside clinical environments.

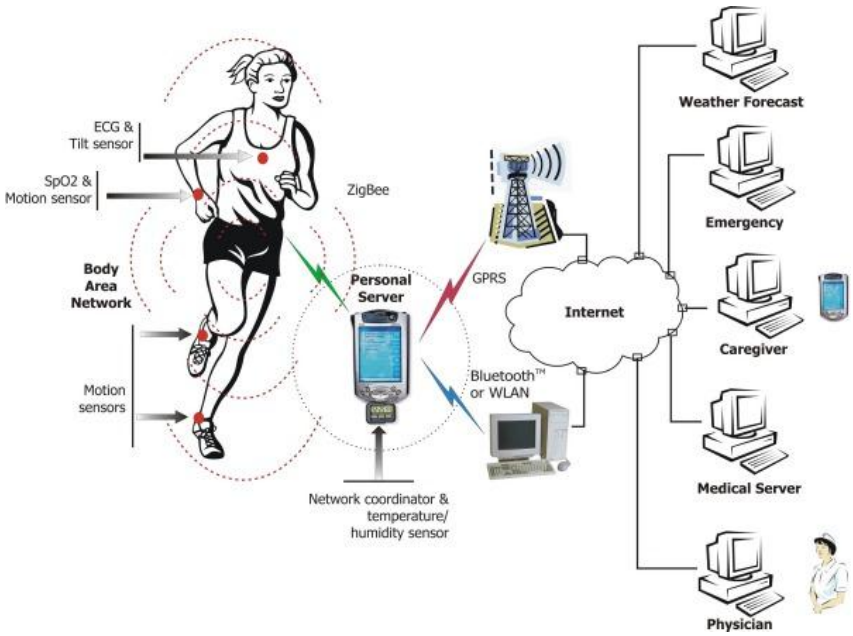


Figure 2.1: Wearable sensors for health monitoring [11].

2.4 Blood Gas Analysis

Blood in a human body performs many vital functions. It takes the oxygen from the lungs to the tissues and to the brain and supplies nutrients from the digestive tract to the different parts of the body. It removes carbon dioxide from the tissues to the lungs to be expelled and removes wastes products like urea and lactic acid through the kidney. It provides immunological functions using the white blood cells and antibodies, repairs the body with coagulation, regulates the temperature of the body and transports hormones and signals tissue damage [12, 13].

Monitoring the properties of blood gives the medical practitioner a wealth of information about the health of the patient. Blood sugar levels are measured for diabetes, cholesterol

levels can be determined from blood tests. The doctor accesses a variety of parameters to assess an individual's health; blood gases are one the parameter accessed by the doctors. The blood gases consist of the partial pressure of oxygen, pO_2, partial pressure of carbon dioxide, pCO_2 and blood pH [13].

2.4.1 Cardiovascular System

Blood is circulated around the body through blood vessels by the pumping action of the heart. In humans, blood is pumped from the strong left ventricle of the heart through arteries to peripheral tissues and returns to the right atrium of the heart through veins. It then enters the right ventricle and is pumped through the pulmonary artery to the lungs and returns to the left atrium through the pulmonary veins. The blood in pulmonary artery exchanges the carbon dioxide in the lungs with oxygen, thus deoxygenated blood enters the lungs and oxygenated blood comes out. Blood then enters the left ventricle to be circulated again. Arterial blood carries oxygen from inhaled air to all of the cells of the body, and venous blood carries carbon dioxide, a waste product of metabolism by thecells, to the lungs to be exhaled. Figure 2.2 shows the cardiovascular system in a human being [13, 14].

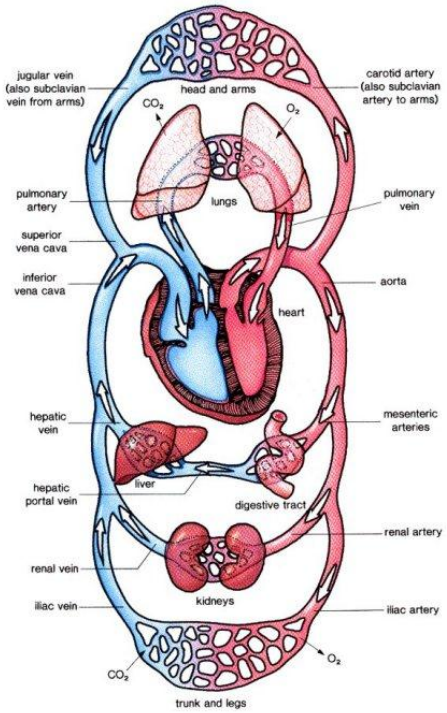


Figure2.2: Human cardiovascular System [15].

2.4.2 Gas Exchange

The air inhaled during respiration gets saturated with water vapour as it enters the alveolar sac. In the alveolus, it also mixes with carbon dioxide. The blood from the pulmonary artery enters the lungs and the exchange of the gases happens by diffusion at the alveolar membrane. The direction of the diffusion is determined by the partial pressure of the gas. The blood entering the lungs is depleted of oxygen and has a lower partial pressure for oxygen thus the oxygen diffuses from the alveolus to the blood. The opposite occurs with carbon dioxide which diffuses towards the alveolus and mixes with the air. The diffused oxygen is taken up by the red blood cells. As blood circulates through the body, an opposite change occurs in the capillaries where oxygen diffuses from the blood into the cells. Carbon dioxide diffuses from the cells into the blood [14, 16].

2.4.3 Blood gases

2.4.3.1 Partial Pressure of Oxygen PO2

Oxygen is transported to the tissues through the blood stream in two ways, bound to haemoglobin within the red blood cells and dissolved in the plasma. Oxygen is poorly soluble in water so only 1.5% of the oxygen transported is carried in dissolved form. The remainder of oxygen 98.5% that is not dissolved in blood combines with haemoglobin, a protein-iron compound found in red blood cells. Haemoglobin contains four iron atoms and each iron atom can combine with one oxygen O_2 molecule, thus one haemoglobin molecule can combine with up to four O_2 molecules. The oxygen content measurement in an Arterial Blood Gas (ABG) analysis indicates how much oxygen is combined with the haemoglobin. A related value is the oxygen saturation, which compares the amount of oxygen actually combined with haemoglobin to the total amount of oxygen that the haemoglobin is capable of combining with. How much oxygen dissolves and how much combines with haemoglobin depends on the partial pressure of the oxygen (the pressure that the gas exerts on the walls of the arteries). Therefore, testing the partial pressure of oxygen is a measure of how much oxygen the lungs are delivering to the blood. Tissue

oxygenation happens when the blood with a higher partial pressure of oxygen reaches the cells which are at a lower pressure of oxygen; the oxygen dissolved in plasma and combined with haemoglobin diffuses to the cells and carbon dioxide diffuses to the blood. A low partial pressure of oxygen in the artery would thus result in inadequate tissue oxygenation [16].

2.4.3.2 Partial Pressure of Carbon Dioxide

Carbon dioxide is formed in the tissues and cells as a result of aerobic metabolism and is expelled from the body via lungs through blood. Carbon Dioxide CO_2 is transported in blood from the tissues to the lungs in three forms: as a gas dissolved in plasma, bound to haemoglobin and as the bicarbonate ion in plasma. Carbon dioxide dissolves more easily than oxygen thus at the tissue level as CO_2 diffuses from cells to blood it dissolves in the plasma. However, only 7-10% of carbon dioxide remains in the dissolved state, as soon as the carbon dioxide molecules enter the blood they react with the red blood cells and carbon dioxide gets prepared for transport [16].

Carbon dioxide reacts with the haemoglobin molecules to form carbaminohaemoglobin. The carbon dioxide molecule disassociates from the haemoglobin molecule in the lungs and is then expelled out. The association and disassociation of carbon dioxide is dependent on the partial pressure of carbon dioxide. In the lungs the partial pressure of carbon dioxide is higher in the blood than the air in the alveoli hence the gas diffuses to the alveoli to be expelled. In the tissues the situation is reversed and the gas diffuses from the higher pressure in the blood to the lower pressure of the blood. 20% of carbon dioxide is transported as carbaminohaemoglobin.

A large portion of the carbon dioxide 70% entering the blood from the tissues gets converted to bicarbonate ions (HCO_3^-) and is transported in the plasma. When carbon dioxide diffuses into the red blood cells it combines with water vapour to form carbonic acid H_2CO_3. Carbonic acid is unstable and quickly dissociates into hydrogen ions and bicarbonate ions. The reaction is shown in equation 2.1.

$$CO_2 + H_2O \rightleftharpoons H_2CO_3 \rightleftharpoons H^+ + HCO_3^- \qquad 2.1$$

The hydrogen ions released during the reaction bind readily to the haemoglobin molecules this triggers the Bohr Effect and releases the oxygen to the tissues, thus the oxygen release

is enhanced by the carbon dioxide diffusion. The negative bicarbonate ions formed bind with the hydrogen ions in the lungs to form carbonic acid again which then breaks as carbon dioxide and water. This carbon dioxide diffuses from blood to alveoli along with the carbon dioxide released from the haemoglobin molecule and the molecules dissolved in plasma. The partial pressure of carbon dioxide gives you an indication if the lungs are effectively expelling the carbon dioxide from the blood. It also maintains the acid-base balance of the body using the bicarbonate ions. Falls in the partial pressure of carbon dioxide result in an imbalance of the bicarbonate ions causing in an imbalance of the pH and thus disrupting the cellular functions of the body [13, 16, 17].

2.4.3.3 Blood pH

The pH in the human body is critically balanced and is maintained in a narrow range. The pH of blood is typically between 7.35 and 7.45. The pH values maintain the functioning of the enzymes, proteins, and nerve and muscle activity. Abrupt changes in pH disrupts these functions and the metabolic activities become unstable. Human life is not possible outside the pH range of 6.8 to 7.8. The pH of the body is regulated by the metabolic activity and the respiratory activity. The metabolic activity in the body results in metabolic acidosis which raises the acidity of blood. The by-products from metabolism urea and lactic acid are removed from the body by the kidneys thus maintaining the pH balance. Any disruption in the functioning of the kidneys results in big changes in the pH value of blood. The respiration process changes the pH value with the formation of the bicarbonate ions and carbonic acid during the carbon dioxide diffusion process. The change in pH is regulated by the haemoglobin molecules in blood which act as buffering agents. The lungs and kidneys both participate in maintaining the pH balance. The lungs control the carbonic acid level and the kidneys regulate the metabolic acids. If either organs are not functioning properly, an acid-base imbalance can result. Determination of bicarbonate and pH levels, aids in diagnosing the cause of abnormal blood gas values [13, 16, 17]. Table 2.1 presents the normal values of blood gases for a healthy individual.

Table2.1: Normal values for blood gas analysis in a healthy patient[13]

	Quantities Measured in Blood Gas Analysis	**Normal Values for a Healthy Patient**
1.	Partial pressure of oxygen (pO_2)	75-100 mm Hg
2.	Partial pressure of carbon dioxide (pCO_2)	35-45mm Hg
3.	Oxygen content (O_2CT)	75-100 mm Hg
4.	Oxygen saturation (SaO_2)	94-100%
5.	Bicarbonate (HCO_3)	22-26 mEq/litre
6.	Acidity (pH)	7.35-7.45

The measurement of blood gas aids the medical practitioner to diagnose the health of the respiratory system and the kidneys. The partial pressure of oxygen and carbon dioxide provides a ready picture of tissue oxygenation and how effectively the lungs are functioning. The blood pH value gives an insight of the functioning of the kidney and also determines the course of action in critically ill patients if the pH values tend toward the acidic values [16]. Blood gases have been measured either by electrochemical methods or optical methods.

2.4.4 Electrochemical Methods

Electrochemistry is a branch of chemistry that has been used to study the chemical reactions in a solution with the use of electric potentials and currents. There are two types of reactions; a) when a voltage or current has to be applied to drive a chemical reaction known as electrolysis, and b) when the chemical reaction drives the voltage or current is known as an electrochemical reaction. The principle of the electrochemical reaction has been used to develop electrodes to measure the blood gases [18-20]. The development of the Clark's Electrode aided the development of the electrode to measure the partial pressure of oxygen. It consists of a platinum cathode and a silver / silver chloride reference anode shown in figure 2.3. The platinum cathode has an affinity with oxygen. The electrode provides a path for reduction as shown in equation 2.2

$$O_2 + 4e^- \longrightarrow 2O^- \qquad 2.2$$

The current produced by the cathode is proportional to the partial pressure of oxygen. The electrodes are covered by a semi permeable, constant diffusion membrane which allows the oxygen molecules to pass through. A potassium chloride electrolyte is also used with the membrane; this improves the response, longevity and stability of the electrode. These stop the electrode from getting coated in blood and degrade the sensitivity of the electrode [16, 21].

The pH electrode uses a glass electrode instead of a platinum electrode. The electrode construction has a glass membrane acting as the sensor for pH. The body houses the silver chloride anode and a reference electrode and is covered by the glass membrane at the tip. The electrodes are covered by an electrolyte and the glass membrane separates the electrolyte from the blood sample. A current is produced when the glass electrode comes in contact with the blood sample due to the dissociation of the hydrogen ions from the glass membrane [21, 22].

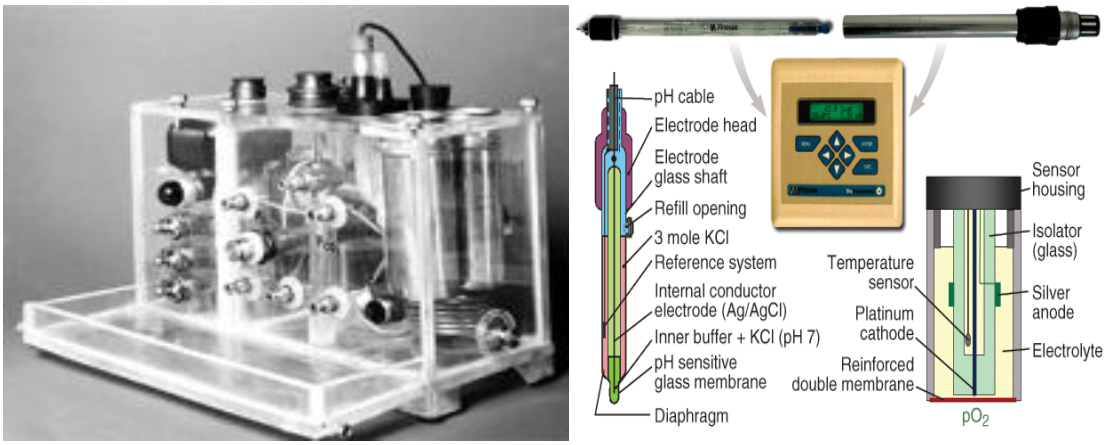


Figure 2.3: Blood Gas Analyser based on the electrochemical principle and a standard pH probe.[18, 23]

The pCO_2 electrode uses the pH glass electrode to measure the partial pressure of carbon dioxide. The glass electrode is immersed in a bicarbonate buffer solution and is covered by a silicon membrane to separate the buffer from the blood sample. The membrane allows carbon dioxide and water vapour to diffuse through to the bicarbonate buffer. The carbon dioxide and water vapour combine to form carbonic acid, this changes the pH of the buffer. The change in the pH of the bicarbonate buffer is proportional to the partial pressure of carbon dioxide [16, 21, 22].

2.4.5 Transcutaneous Methods

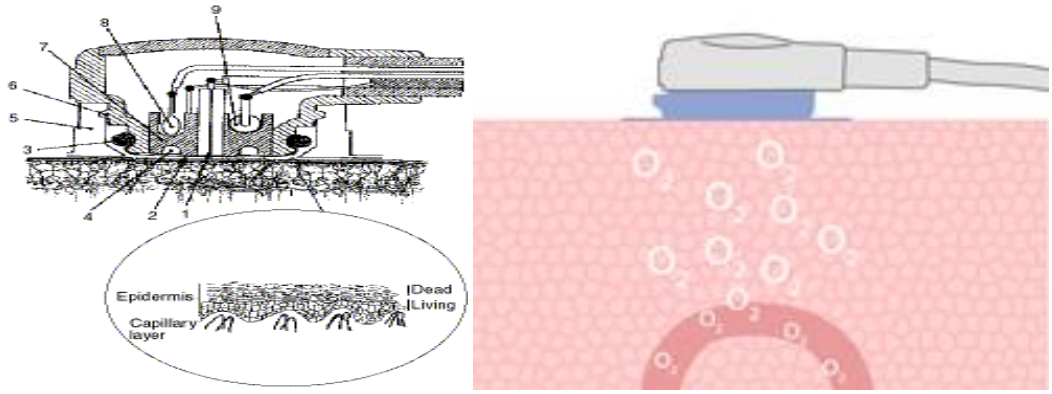


Figure 2.4: Transcutaneous Method of measuring Blood Gases

Transcutaneous measurement is a measurement of gas perfusion of the skin. The skin is the largest tissue of the human body and respires oxygen. The partial pressure of gas measured is called the transcutaneous partial pressure [24]. The technique involves heating the skin and measuring the perfusion of the blood gases from the skin. The skin is heated for 5-15 minutes to reach equilibrium between the oxygen transported by the blood and the oxygen extracted by the skin and the electrode. The measurement of perfusion can be done either by optical or by using the electrochemical techniques [25, 26]. Transcutaneous monitoring is non-invasive and relatively simple to use. In neonates and small infants, this monitoring technique may provide very useful clinical information. Transcutaneous gas monitoring provides a means of trending the values of pO_2 and pCO_2 in most patients with relatively normal cardiovascular function. Typically the skin of the patient is heated to 44 °C for measurement of perfusion, this can cause burns at the site of measuring hence the probe has to moved constantly. Also the measurement is more suitable for neonates and infants as the skin gives better perfusion than adults [20, 24, 25, 27, 28].

2.4.6 Optical Methods

The optical methods for measuring blood gases are based on the absorption, reflection, scattering and fluorescence techniques[21, 22]. The principles of fluorescence and absorption have been widely used in developing sensors for blood gases. Fluorescence is a phenomenon seen with certain materials when they are excited by light. The excitation causes the atoms and molecules in the material to a higher energy state. The energy

absorbed to rise to this higher state is then released as a photon to return the atoms and molecules back to the ground state. Blood gases affect the fluorescence emissions of the material and the change in the fluorescence is used to measure the blood gases [29]. The intensity of the fluorescence is measured; the change in the intensity of the fluorescence is proportional to the blood gas values [30, 31]. The absorption principle has been used to develop sensors for pH and carbon dioxide [32]. A typical optical blood gas system is shown in figure 2.5; the system consists of the optical blood gas sensors, circuits to measure the light intensity in terms of voltage. The intensity values are digitised by the analog to digital controller and are passed on to the processor where the blood gas values are calculated.

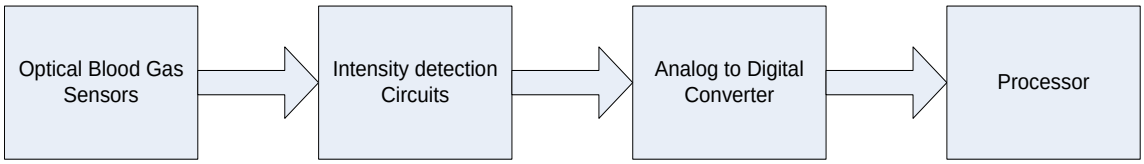


Figure 2.5: Block Diagram of a typical optics based blood gas analysers

2.4.6.1 Optical pO_2 sensor

The principle used in optical sensors to measure the partial pressure of oxygen is the fluorescent quenching properties of oxygen [31]. The device includes an optical waveguide and an oxygen sensing medium deposited on the waveguide. The sensing medium fluoresces in response to light from a light source such that the intensity of fluorescence is dependent on the partial pressure of oxygen in the environment [29-31]. The sensing medium includes an oxygen sensitive fluorescent dye in a matrix consisting of a plasticized polymer. A zero state can be measured by filling the chamber with oxygen concentration mixture. The measured intensity is compared with the incident light source and is used to calculate the concentration of the sample. The oxygen tension is calculated using the Stern-Volmer quenching formula and the pO_2 value is calculated empirically [21, 22, 29, 30, 33]

$$I_0 / I = 1 + K(pO_2) \qquad 2.3$$

Where I (pO_2) is the fluorescence intensity in presence of oxygen, I_0 is fluorescence intensity in absence of oxygen and K is the overall quenching constant.

2.4.6.2 Optical pH Sensor

The pH sensor is based on a dye molecule whose optical properties change with the change in pH from 6.8 to 7.8. The pH sensitive dye is bonded to a cellulose matrix and is attached to the tip of the optical fibre. The dye molecule has distinct optical properties when it is in its acid form and when it is in its base form [33, 34].

The fluorescence based pH sensor works with dyes which fluoresces at different excitation wavelength when in the acid form and the base form. The dye hydroxypyrenetrisulphuric acid has maximum excitation at 460 nm in the base form and a maximum excitation of 410 nm in the acidic form and the emission in both forms occurs at 520 nm. The ratio of the fluorescence emission in the base form and in the acid form is used to calculate the pH. The pH is calculated using a variation of the Henderson Hasselbalch equation shown in equation 2.4 [31].

$$\frac{\mathrm{I_0}}{\mathrm{I}} = 1 + \mathrm{K(H^+)} \qquad 2.4$$

Where I is relative fluorescent intensity, H^+ is the hydrogen ion concentration, I_0 is the reference intensity and K is the dissociation constant.

Absorption based pH sensor measure the absorption of the light by the dye as the pH varies. The dye is irradiated with a light of known intensity and the intensity is measured after it passes through the dye [35, 36]. The incident light is absorbed by the dye and the amount of absorbance varies as the pH changes. The dye phenol red when used in its base form displays more absorbance as the pH changes from 6.8 to 7.8 [32, 33, 37]. This change in the absorbance can be determined using the Beer-Lamberts law as shown in equation 2.5.

$$\log\left(\frac{\mathrm{I_0}}{\mathrm{I}}\right) = [\mathrm{A^-}]\mathrm{l}\varepsilon \qquad 2.5$$

Where I_0 is the incident intensity of light, I is the intensity of light after is passes through the dye, the logarithm of this ratio is the absorbance. $[A^-]$ is the concentration of the dye phenol red in its base form, l is the dye pathlength and ε is the absorption coefficient of the dye phenol red in its base form. The relation between the pH and the base form of the dye can be explained using the Henderson Hasselbalch equation

$$\mathrm{pH} = \mathrm{pK_a} - \log\frac{[\mathrm{HA}]}{[\mathrm{A^-}]} \quad 2.6$$

Where pKa is the negative log of the acid dissociation constant and [HA] is the concentration of the dye in the acid form. When equations 2.5 and 2.6 are combined and the defining the total concentration of the dye as [dye] = [HA] + [A^-] the equation relating the pH to measured intensity is derived [33, 35, 36, 38].

$$\mathrm{pH} = \mathrm{pK_a} - \left(\log\frac{[\mathrm{dye}]\mathrm{l}\varepsilon}{\log\left(\frac{\mathrm{I_0}}{\mathrm{I}}\right)} - 1\right) \quad 2.7$$

2.4.6.3 Optical $pC0_2$ sensor

The optical pCO_2 sensor is not different from the electrochemical sensor; it also employs the optical pH sensor to measure the pCO_2. The pH sensor is covered with a gas permeable membrane; inside the membrane the pH sensor is in contact with a bicarbonate solution[33, 37, 39, 40]. The pH of the bicarbonate solution is known. The gas permeable membrane allows carbon dioxide and water vapour to pass through which combines to form carbonic acid. This carbonic acid changes the pH of the bicarbonate solution[33, 40, 41]. The change in the pH value of the bicarbonate solution is proportional to the partial pressure of carbon dioxide. The relation between the pH and pCO_2 is described as

$$\mathrm{pH} = \log \mathrm{N} + \mathrm{pK_1} - \log \mathrm{K_s pCO_2} \quad 2.8$$

Where N is the concentration of the bicarbonate buffer, pK1 is the negative log of the acid dissociation constant for H_2CO_3 times the hydration constant for CO_2, and K_s is the solubility coefficient for CO_2. The optical sensor for pH can be based on the fluorescence or absorption methods.

2.4.6.4 PulseOximetry

Pulse oximetry is a measurement of oxygen saturation SaO_2 in blood. The pulse oximeter is a simple and non-invasive technique, where a probe is attached to the earlobe or the index finger. The probe then irradiates the body part with red and near red wavelengths of light. The wavelengths are measured on the other side after they have passed through the measuring site [21, 22, 42]. The ratio of the red and the infrared light intensity is calculated and the oxygen saturation is calculated using the following formula

$$SaO_2 = \frac{A_1 \lambda 1/\lambda 2 - A_2}{(A_1 - B_1)\lambda 1/\lambda 2 + (B_2 - A_2)} \quad 2.9$$

Where A_1, A_2→ are functions of the specificabsorptivities of haemoglobinHb

B_1, B_2→ are functions of the specific absorptivities of HbO_2

λ_1, λ_2→ wavelength of red and infrared light respectively

Pulse oximeters measure the oxygen saturation of blood, which gives an indication of the amount of oxygen dissolved in blood. Oxygen saturation is related with the partial pressure of oxygen in blood and under controlled conditions it can be used to calculate PO2. Figure 2.6 shows the oxygen dissociation curve, oxygen saturation has a sigmoid and nonlinear relationship with PO2, and hence 100% oxygen saturation doesn't mean 100% tissue perfusion. Thus pulse oximeters cannot be used to measure blood gas accurately[42].

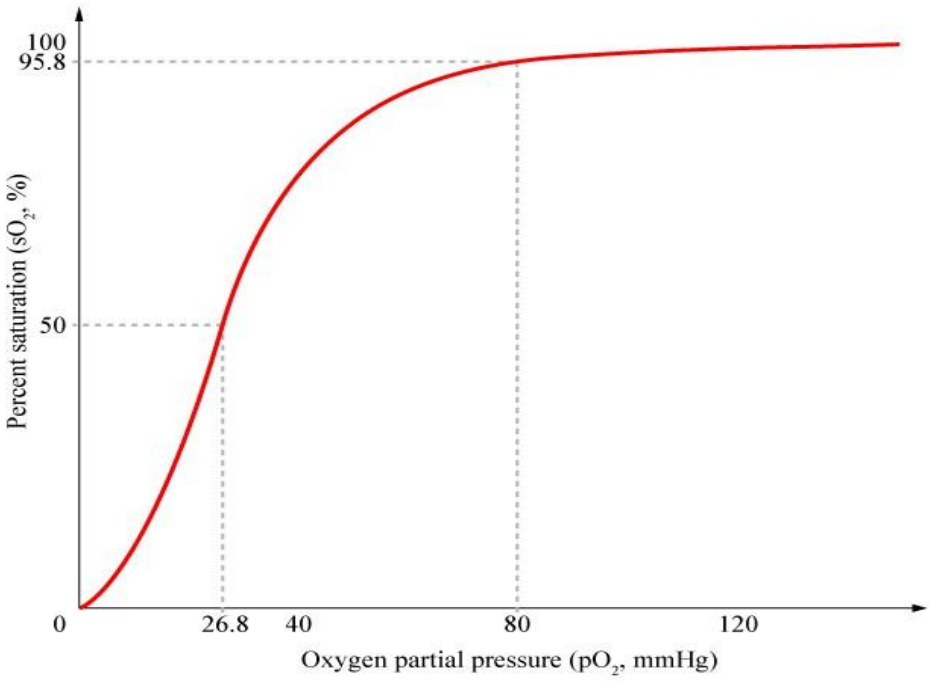


Figure 2.6: Oxygen Dissociation Curve [42]

2.4.7 Mass Spectrometry

Mass spectrometry is an analytical technique that measures the molecular mass of individual compounds and atoms by converting them into charged ions. Mass spectrometry can also deduce the structure of a molecule. It is used to identify unknown compounds and get the chemical properties of molecules. The mass spectrometer is the instrument which measures these characteristics and has four stages. The first stage is the ionization stage where the sample is ionized by knocking off one or more electrons to form a positive ion. The second stage is the acceleration stage; here the ions are accelerated so they all have the same kinetic energy. In the next stage the ions are passed through a magnetic field. The magnetic field deflects the ions and the amount deflection of the ions is a measure of the molecular mass. The final stage is the detector where the ions are detected. A diagram of a mass spectrometer is shown in figure 2.7.[43]

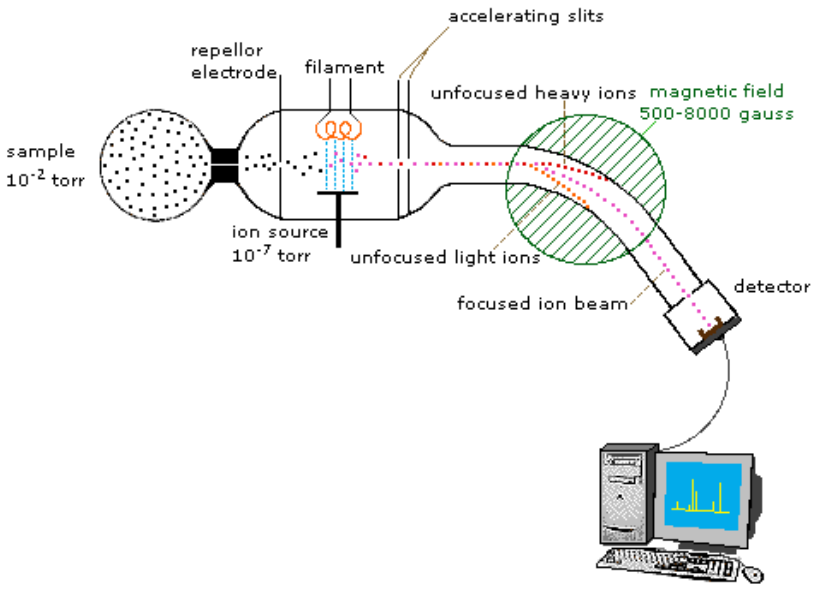


Figure 2.7: Diagram of a mass spectrometer[43]

Mass spectrometers are used either as stand alone or along with the transcutaneous measurement technique to detect the blood gases. In the transcutaneous technique the gases respire from the skin from the localized heating and the gases are passed to the mass spectrometer which ionises the gas samples and detects the partial pressure of

oxygen and carbon dioxide. In the stand alone mode the blood sample is vaporized and passed through the mass spectrometer to detect the blood gas values. The mass spectrometer can be used in -ivo for continuous measurement of the blood gases [24].

2.4.8 Capnometry

Capnometry is the measurement of carbon dioxide in a volume of gas. It is used to measure the partial pressure of carbon dioxide in the breath expired out of the lungs. The partial pressure of the expired carbon dioxide is an indirect measure of the partial pressure of carbon dioxide in arterial blood. In a healthy individual the difference between the partial pressure of the exhaled carbon dioxide and the arterial carbon dioxide is very small. Capnometers play a vital role in monitoring the carbon dioxide volume for patients during anaesthesia and in intensive care and helps avoid hypoxia in the patients [44].

A capnometer works on the principle of absorbtion of light. An infrared light is passed through the gas sample and the intensity of the transmitted light is measured. The amount of light absorbed is dependent on the concentration of carbon dioxide in the sample [44, 45].

2.4.9 Solid State Sensors

Solid state sensors use the properties of the semiconductor materials and the metal oxide layers in the device as the sensing medium [46]. The ion sensitive field effect transistor (ISFET) is an example of a semiconductor device used for detection of ions in a chemical compound. The ISFET is sensitive to the change in concentration of ions and accordingly changes the current passing through the transistor. The ISFET is used in place of a glass electrode to detect the pH value of a sample as it is sensitive to the H^+ ions in a solution. The chemical compound in this case acts as the gate terminal controlling the flow of current through the transistor [47].

Recent improvements have identified other semiconductors materials suitable to be used as ion sensors. Gallium Nitride (GaN) based structures are part of wide band gap semiconductors sensitive for use as chemical sensors. They are used in high electron mobility transistors which have high electron mobility, large sheet carrier concentration and a fast response. The large sheet carrier concentration is due to the piezoelectric

polarisation caused by an Aluminium Gallium Nitride(AlGaN) layer and this piezoelectric effect forms a part of the sensing mechanism. The ions in the sample interact with the surface charge thereby creating a change in the surface potential which influences the sheet concentration leading to a drop in current at the AlGaN/GaN interface [46, 48, 49].

2.5 Shortcomings

Figure 2.8 shows a present day blood gas analyser, the analyser works on the principle of electrochemical analysis where the blood sample is drawn from the patient using a syringe and the syringe is attached to the analyser. The analyser draws the blood to a heated sampling chamber where there are miniature electrodes which measure pO_2, pCO_2 and pH values of the blood sample. The unit needs to be properly calibrated to give accurate readings [50]. Based on the three readings the unit calculates the values for HCO_3^- and oxygen saturation. The unit is accurate but is big and bulky and is not easily portable; also the unit needs to run on mains supply.

Figure 2.8: AVL Compact 2 Blood Gas Analyser [50].

The optical sensors for blood gas analysis are smaller in size than the electrochemical sensors and aid continuous measure of blood gases but they measure the blood in vivo thus restricting the mobility of the patient and confining them to the hospital bed. The

associated electronics for the optical sensors draw a lot of power thus limiting the portability of the optical sensors [37].

Transcutaneous measurements are non invasive but they are not suitable for adult patients [22]. Pulse oximeters provide a non invasive method to measure the oxygen saturation which can be used to predict the values of partial pressure of oxygen. However the relation between the partial pressure of oxygen to the oxygen saturation is a sigmoid, hence can't predict the partial pressure in all situations. Also the values of pH and pCO_2 cannot be known through oximetry [42].

The capnometer is a portable instrument is a portable instrument but only measure the partial pressure of carbon dioxide from the exhaled air. The mass spectrometer can vary in size from a small box to the size of a small room and also requires mains power for operation.

The sensors for pH based on the ISFET require a reference electrode to measure the pH of a solution. The conventional type of the reference electrode puts the same limitations on the ISFET sensor like that of the conventional glass electrode. The conventional reference electrode also puts a limitation on the miniaturization of the sensor and the bulky and fragile nature of the reference electrode affects the portability of the system. The reference field effect transistors (REFET) alleviate the problems from the conventional reference electrodes but they require frequent recalibration as they are not always in thermodynamic equilibrium with the test solution.

2.6 Conclusion

Wearable sensors and home care are breaking new ground in the medical world. They allow the doctors access to data from the day to day environment of the patients; which allows them to make well informed decisions for diagnosis. The data allows the doctors to predict a serious health condition and take remedial actions to avert it which is aligned with the new philosophy of Prevention of serious medical condition before care. Blood gases are one of the vital signs doctors assess to diagnose the health of the respiratory system and the functioning of the kidneys. This chapter attempts to give the reader an overview to what blood gases are, their physiology and current techniques for measure them. The review focuses on the electrochemical and optical methods to measure the

blood gases. The methods were evaluated for portability with focus on size and power consumption. The electrochemical and optical techniques utilise the main supply to power the unit. The current electrochemical analysers are big and bulky and the optical analysers are used in vivo. These factors rule out the techniques for a portable design. The following chapter discusses an image based colorimetric technique for portable blood gas analysis. The chapter also details the pH and indicator chemistry.

Chapter Three: Design Proposal

3.1 Introduction

The preceding chapter discussed the properties of blood gases and the different techniques to measure the blood gases. The electrochemical and optical measuring techniques are the predominant techniques to measure blood gases. These techniques use both invasive and non invasive methods to measure the blood gases[21, 22].

The non-invasive methods of measuring the blood gases mainly the partial pressure of oxygen are based on the measurement of the amount of oxygen saturation. The relation between the partial pressure of oxygen and the saturation of oxygen in blood is a sigmoid relation. Thus the partial pressure of oxygen can be predicted accurately from the oxygen saturation only in a controlled situation[22].

The invasive measurements with electrochemical methods required the withdrawal of a small quantity of blood from the patient's body and aspirating it in the analyser to analyse the blood gas contents. The optical methods use an optical fibre which is embedded in the artery of the patient and the blood gases are measured using the principles of fluorescence or absorption [21, 22, 32, 33].

Both the techniques mentioned above require the patient to be present at the clinic and in most cases confined to a bed in the hospital. The instruments are big and bulky and run on AC power supply which is a hindrance to portability.

Solid state sensors are a cheap and a portable alternative for the sensors to be used to measure the blood gases. They are an effective replacement for the electrodes in the electrochemical process; however they still suffer from the same problems such as drift associated with the electrochemical electrodes and need constant recalibration.

The aim of this project is to find an alternative solution to measure blood gases with a low power consumption, area and size suitable for a portable system.

3.2 Design Proposal

This research looks at a colour based methodology to measure the blood gases which is portable, cost effective and easy to use. The change in colour of an indicator to the change in pH of a solution is well known. This thesis explores this change in colour of the indicator to assess the blood gases. The change in the colour of a pH Indicator is used as a principle to measure the pH in the design[17, 51]. A diagram of the proposed Colorimetry based blood gas analyser shown in figure 3.1 below.

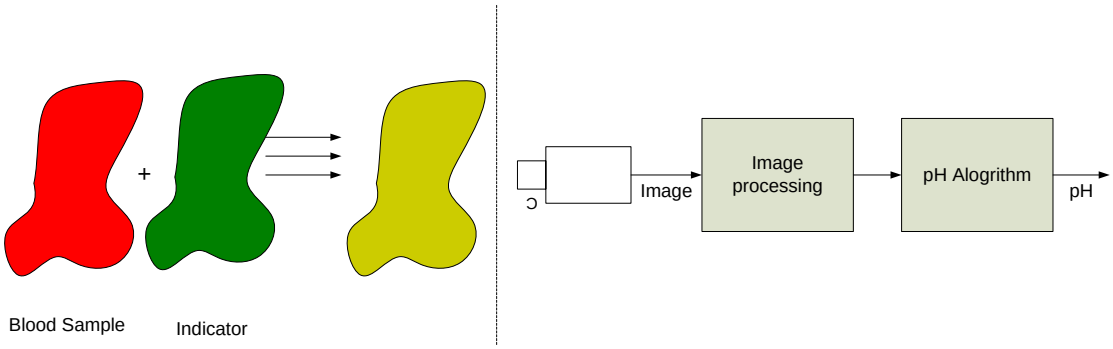


Figure 3.1: The Colorimetric Design Proposal

The sample is mixed with the indicator and the change in the colour of the indicator is imaged. The image is then processed to quantify the colour of the indicator. A mathematical relation which defines the relation between the colour of the indicator and the corresponding pH level is analysed after receiving the colour values from the image and processes this information to give the pH value of the sample. The derivation of the mathematical relation and its implementation is one of the central themes of this thesis.

3.2.1 Research Methodology

The research consists of a literature review on pH chemistry and indicator chemistry which is important to understand the chemical processes and reactions of the indicator to provide the link between the colour and pH and to select the proper indicator for the product. The literature review also provides insight on the colorimetric techniques and colour imaging used to quantify the colour value.

The next stage is the proof of concept for the relation between the colour of the indicator to the pH value of the sample; this is achieved using an experimental setup to image the

change in colour of the indicator and quantifying the colour change in terms of the red,green and blue colour components.

The literature review on the pH chemistry, the indicator chemistry, colour imaging and colorimetry are provided in the following sections which is then followed by the experimental setup for proof of concept.

3.3 pH Chemistry

As it is well known chemistry is the study of materials and its properties and the changes in its properties when they come in contact with other materials. Being one of the principal sciences, has many applications in the fields of physics, biology, medicine, engineering and other subjects. Chemistry involves studying the materials as chemical elements in their atomic and molecular form. One of the properties which is studied and measured is the acidity and basicity and the chemistry of acids and bases. Typically acids are described as having a sour taste and change the colour of litmus to red, while bases have a bitter taste, feel slippery and change the colour of litmus to blue [17].

The first definition of an acid was given by the Arrhenius theory[51]. The Arrhenius theory defines an acid as a substance which disassociates to produce a hydrogen ion and a base as a substance which disassociates to produce a hydroxide ion. Equation 3.1 shows the dissociation of hydrochloric acid when dissolved in water and the release of the hydrogen ion. Equation 3.2 shows the dissociation of sodium hydroxide in water and the release of the hydroxide ion[17].

$$\mathrm{HCl} \xrightarrow{\mathrm{H_2O}} \mathrm{H^+ + Cl^-} \qquad 3.1$$

$$\mathrm{NaOH} \xrightarrow{\mathrm{H_2O}} \mathrm{Na^+ + OH^-} \qquad 3.2$$

Therefore the Arrhenius theory attributes the acidity of a solution to the activity of the hydrogen ions. The Arrhenius theory can only be applied to aqueous solutions of the acids and bases and only to certain situations [17, 51].

To accommodate all forms a broader definition was proposed by the Bronsted Theory[51]. The Bronsted theory states an acid is a proton donor and a base is a proton acceptor. The Bronsted theory explains the nature of materials like baking soda $NaHCO_3$ which act like a

base even if they do not contain the hydroxide OH^- ion. In a reaction with hydrochloric acid baking soda accepts a proton H^+ thus it acts like a base which is shown in equation 3.3[51].

$$HCl + NaHCO_3 \rightarrow H_2CO_3 + NaCl \quad 3.3$$

The Bronsted theory does not require acids and bases to be in aqueous form. The Bronsted theory relates the strength of both an acid and base to the concentration of the hydrogen ions present. As the strength of the acid increases the concentration of the hydrogen ions increases and as the strength of a base increases the concentration of the hydrogen ion decreases. Thus the concentration of the hydrogen ions is used as a measure of the acidity or basicity. The concentration of the hydrogen ions in an aqueous solution is a very small number and is inconvenient to work with; therefore a practical approach was proposed in 1909 called pH[51]. The pH of a solution is defined as the negative logarithm of the hydrogen ion concentration shown in equation 3.4[17, 51].

$$pH = -\log_{10} H^+ \quad 3.4$$

The pH values have scale which goes from 0 to 14 .The concentration of hydrogen ions in acids is greater than 10^{-7} and the pH values of acids is less than 7. Bases have the concentration of hydrogen ions less than 10^{-7} and have pH values greater than 7. The neutral solutions have a hydrogen ion concentration of 10^{-7} and have a pH value of 7. Water is a neutral solution and has a pH of 7. Water has the ability to act as an acid or as a base. Water functions as a base in reactions with acid like HCl and as an acid in reactions with bases like NH3. The pH of water also changes with the pH of the material it dissolves as water is a solvent for many materials. When mixed with an acid the pH of water falls below 7 and vice versa when mixed with a base it is greater than seven. Table 3.1 shows the ph scale and some of the common household items against it[51, 52].

3.4 Indicator Chemistry

The change in colour of dyes by the action of acids and bases is one of the oldest reactions in chemistry. The first scale for pH was bases on a table of indicator dyes. Colorimetric methods were prevalent before the development of the potentiometric glass electrodes and are still used because of their low cost and faster detection time. pH indicators are chemical compounds, organic acids, which help detect the hydronium ions and give a

visual measurement of the pH. The change in colour occurs as a result of the rearrangement of the molecules when hydrogen ions are released or taken up [17, 52].

Table 3.1:Table showing the ph values of some everyday items[51].

Solution	pH value
Gastric Juice	1
Lemon Juice	2.3
Vinegar	3.1
Beer	4.3
Sour Milk	4.4
Plum Juice	5-6
Fresh Milk	6.5
Saliva	6.7
Pure Water	7.0
Blood	7.36
Sea Water	8.3
Soap-Suds	8.4
Lime Water	12.3
Sodium Hydroxide	14

pH indicators are weak acids or bases themselves. An Indicator can be termed as HIn in the acidic form and In_- in the base form and Kin is the dissociation constant for the indicator[17, 52].

$$HIn + H_2O \rightleftharpoons In_- + H_3O^+ \quad 3.5$$

$$KIn = \frac{[In_-][H_3O^+]}{[HIn]} \quad 3.6$$

$$\frac{KIn}{[H_3O^+]} = \frac{[In_-]}{[HIn]} \quad 3.7$$

The colour of the indicator at any pH is determined by the ratio of $[In_-]$: [HIn]. As pH increases, $[H_3O^+]$ decreases, $[In_-]$: [HIn] increases, so the indicator has increasingly more of the colour of the base form and less of the acid form [53]. Litmus is one of the indicators which have been used in chemistry to determine the acidity of a solution. Litmus changes to a deep red colour when the solution is acidic and changes to blue if the solution is basic[52, 53].

The range of an indicator is the range of pH over which the reaction between the indicator and the chemical compound causes the indicator to change colour. The change in colour is not a sharp change but a change over the range. The range of the indicator is dependent

on the dissociation constant *Kin* of the indicator. The dissociation constant is always referred as pKin which the negative log of *Kin*. Thus in a chemical reaction when the acid and its ion are equal, the pH of the solution indicates the pKin of the indicator. The end points for the indicator are entirely dependent on this value as shown in table3.2 below[52, 53].

Table 3.2:Common Indicators their pKind values and the pH range.[53]

Indicator	**pKind**	**pH range**
litmus	6.5	5 – 8
Methyl Orange	3.7	3.2 -4.4
phenolphthalein	9.3	8.2 -10.0

The pH value of blood for a healthy human being lies between the ranges of 7.35 to 7.45[13, 16]. The extreme value for it is 6.9 at its lowest and to 7.8 at its highest and human life is not possible if it exceeds this range[16].A suitable indicator for this range is bromothymol blue which has a range from 6.0 to 7.6 as shown in table 3.3. The indicator exhibits a colour change from yellow to blue over the range which suitable to this applicationas it is in contrast with the colour of blood[52]. To determine the relation between the changes in colour of the indicator to change in pH, a test bed was setup to image the samples.

Table 3.3: List of the indicators [52]

Name	Acid Colour	pH Range of Colour Change	Base Colour
Methyl violet	Yellow	0.0 - 1.6	Blue
Thymol blue	Red	1.2 - 2.8	Yellow
Methyl orange	Red	3.2 - 4.4	Yellow
Bromocresol green	Yellow	3.8 - 5.4	Blue
Methyl red	Red	4.8 - 6.0	Yellow
Litmus	Red	5.0 - 8.0	Blue
Bromothymol blue	Yellow	6.0 - 7.6	Blue
Thymol blue	Yellow	8.0 - 9.6	Blue
Phenolphthalein	Colourless	8.2 - 10.0	Pink
Thymolphthalein	Colourless	9.4 - 10.6	Blue
Alizarin yellow R	Yellow	10.1 - 12.0	Red

3.5 Colour Imaging

Images form a very important part in representing all the visual data taken in by us in the day to day life. Pictures were the earliest form of representation followed by photographs and now digital images have become a common place. A digital image can be generated by a digital camera, video recorder or an image scanner.

The digital camera works on the same principle of the film based camera. The image is projected on to a solid state sensing array which acts as an image sensor instead of a film. The solid state array records the image and creates a raw image. The image sensors convert the light into electrical charges. The image sensor employs two techniques to

achieve this; the Charged Coupled Device (CCD) and the Complementary Metal Oxide Semiconductor (CMOS) technology and both convert light into electrons[54].

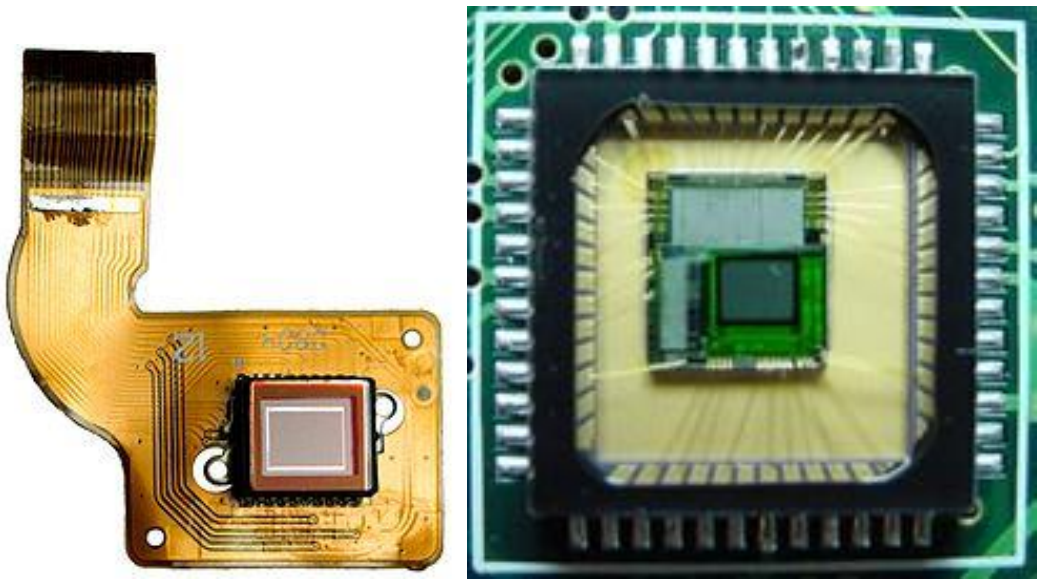


Figure 3.2: Image sensors with a Charge Coupled Device on the left and a Complementary Metal Semiconductor Image Sensor on the right[55].

A CCD sensor uses a capacitor array to accumulate an electric charge proportional to the light intensity falling on a location and after it has been exposed to the image the control circuit causes each capacitor to transfer its content to its neighbour mimicking the operation of a shift register. The last capacitor in the array moves the content to a charge amplifier which converts the charge to a voltage. The voltage is passed to an analog to digital convertor which converts the value to digital form for each pixel[54, 56].

The CMOS image sensor also known as an active-pixel sensor contains an integrated circuit consisting of an array of pixel sensor, each pixel sensor contains a photo detector and an active amplifier. The photo detector converts the light intensity at each pixel to a current; which is then converted to digital form using an analog to digital converter similar to the CCD sensor[54, 56].

The CCD sensors can create high quality and low noise images, the CMOS sensors in contrast are more susceptible to noise, however has a smaller footprint. The CMOS senor has lesser light sensitivity as many of the photons hit the transistors in the CMOS instead of the photo detector. The CMOS sensors consume very little power when compared to CCD sensors which consume a lot of power [54].

Both the CCD and CMOS sensors can only produce a black and white image of the original image projected on to them as the sensors record only the intensity of the light projected on them and not their colour. To get a full colour image filtering is used to measure the light in its primary colours. The camera records the image in all three colours and then combines it to form the full picture. This can be achieved by using three separate sensors each with a filter for one of the colours and use a beam splitter to split the incoming light to the three different sensors. This lets the camera record the three colours at each location but the system becomes big and bulky. To reduce the size of the camera a colour filter array is used which is permanently fixed over the sensors and record only one of the primary colours at a pixel location. The sensor area is broken up in to a variety of red, blue and green pixels. The true colour at a pixel can be guessed accurately by interpolating the information from the adjoining pixels. The Bayer filter pattern is the most common pattern of the filter used in colour digital cameras. The pattern alternates a row of red and green filters with a row of blue and green filters. The pattern has the same number of green pixel as the red and blue combined; this is because the human eye is not equally sensitive to all the three colours. In order to create an image that the eye will recognize as a true colour image more information is needed from the green pixels[54]. The Bayer pattern is shown in figure 3.3.

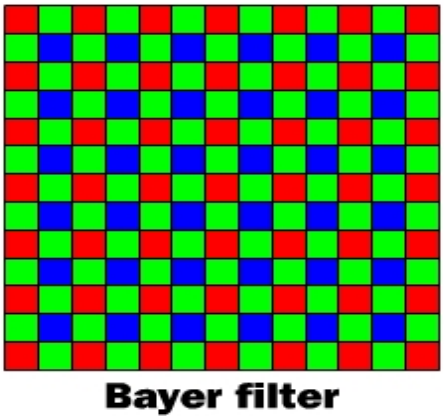


Figure 3.3: The Bayer filter[54]

The colour image created after the light passes through the Bayer filter and sensor is a raw image and is a mosaic of red, blue and green pixels of varying intensity. The camera uses an algorithm to create the true colour image from the raw mosaic called a Demosaicing Algorithm. The demosaicing algorithm determines the true colour of a single pixel by averaging the values from the closest surrounding pixels[54].

3.6 Colorimetry

Colours are an important aspect of life and are necessary information needed for the brain to process the visual data received. Colours describe the property of an object and also describe the characteristics of the light being reflected from an object. Colours also define the brains interpretation of the specific manner in which the light is perceived by the eye. Colorimetry is the technique to measure and describe the human colour perception[57].

The Human eye is composed of photoreceptors which are called as cones and rods which act as the sensors to produce an image in the brain. The rods and cones act in different ways, the rods are monochromatic and are used at low light conditions. The cones are the receptors which are responsible for the colour image and work in bright light conditions. There are three types of cones and each one is sensitive to a different wavelength of light spectrum. Figure 3.4 shows the sensitivities of the cones to the different wavelengths. The three wavelengths fall in the red, green and blue regions of the spectrum and are referred as the long, medium and short wavelength regions. The light entering in the eye is reduced to the three colour components and the three cones generate signals based on the energy of the spectra falling on them and the values are known as the tristimulus values[57, 58].

The tristimulus values of a colour are the amounts of the three primary colours needed to reproduce the original colour using a three component additive colour model. The tristimulus colorimetric model based on the RGB colour model measures the colour in the three basic colours red, blue and green and it is called as the RGB colour space. Zero intensity for each of the colour components results in the darkest colour black which means there is no light and the full intensity of all the colour components gives the colour white.[59] When one of the components has the highest intensity then the colour is closer to that primary colour for example if the intensity of the component green is the highest then the colour will be greenish. A secondary colour is formed when two out of the three components have a stronger intensity than the third; high intensity in green and blue result in a colour cyan, magenta is the result when red and blue have high intensity and yellow is a result of red and green having higher intensity than blue. Figure 3.5 shows the

primary colour components red, green and blue and the secondary colours from their varying intensity[57, 58, 60].

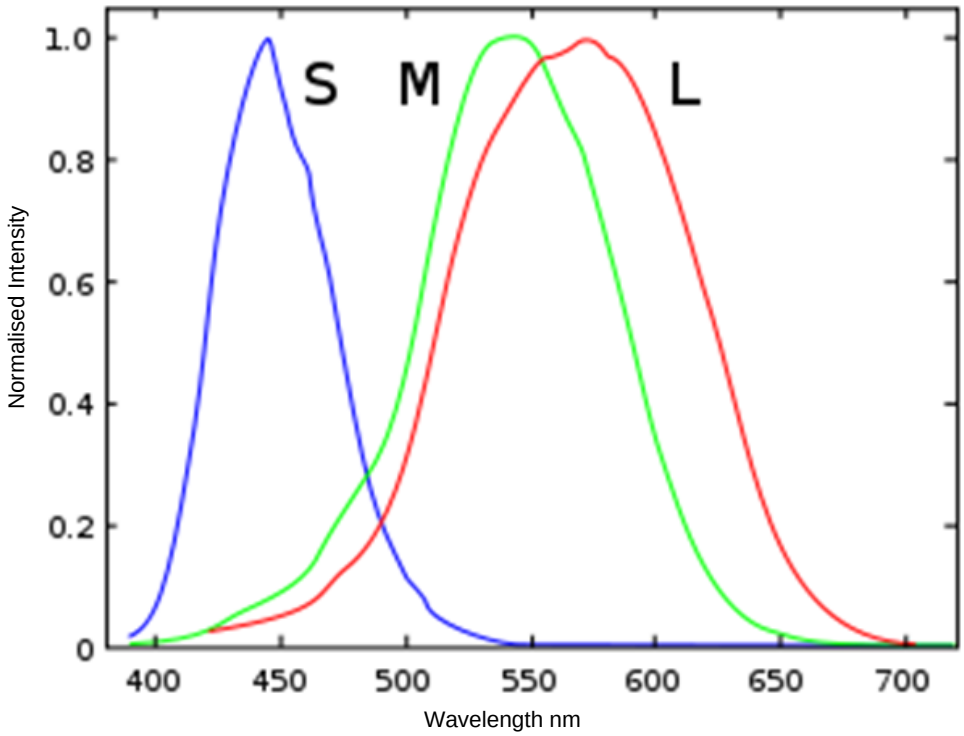


Figure 3.4: The sensitivities of the cones in the human eye to the different wavelengths of light[61]

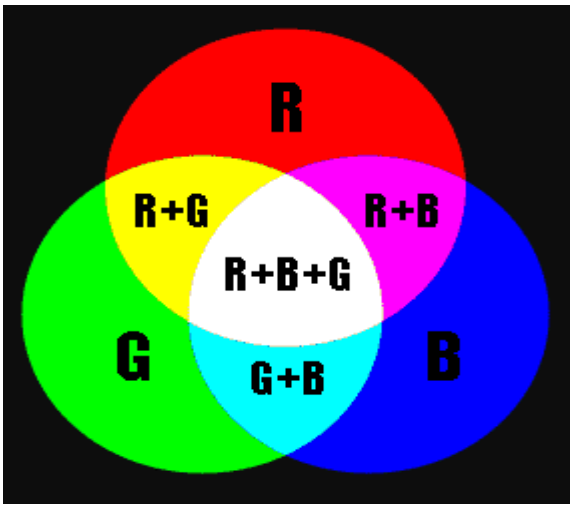


Figure 3.5: The RGB colour space[62]

In a digital image file the data for the colour is stored using the primary coloursand the values of the primary colours are used to recreate the colour as the original. Each pixel has eight bits to store the intensity values of the primary colours in a range of 0 to 255 and when the colour is reproduced on the screen it's an addition of the three primary colours.

Thus the digital image can be read to find the colour of the image in terms of the RGB values[60].

3.7 Experiment Setup

The change in colour of an indicator to change in pH is explained in the previous sections. The measurement of colour in terms of the primary colours from a digital image has also been discussed. To determine the pH from the colour of the indicator a mathematical relation is needed. A test bed was setup to explore the relation between the colour of the indicator and the pH.

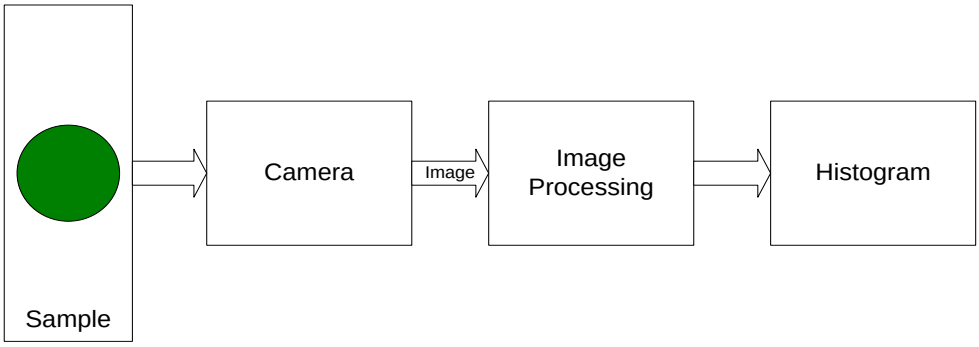


Figure 3.6: Block Diagram of the experimental setup

A block diagram of the test setup is shown in figure 3.6. It consists of the pH sample mixed with the indicator bromothymol blue, the camera to record the image and the image processing software to record the histogram of the colour from the samples.

The pH samples were prepared in the range of 6.1 to 7.9 in units of 0.1 and the indicator bromothymol blue of concentration 0.1 Molar (M). The indicator Bromothymol Blue was mixed with pH samples of known values. The samples were backlit with a white light source. The samples were then imaged using a 1.3 MP CCD camera. The camera used for the experiment is a µEye UI-2240SE-C-HQ from IDS Imaging Development Systems. The CCD camera was used as it provides more sensitivity to colour than the CMOS camera. The resolution of the camera does not influence the accuracy of the analyser due to the fact that the camera used is of very high resolution.

The camera was controlled using a Lab-VIEW Virtual Instrument (VI).Laboratory Virtual Instrumentation Engineering Workbench (Lab-VIEW) is a graphical programming

environment used to develop sophisticated measurement, test and control systems. It uses graphical icons and wires and the flow of logic resembles a flowchart. The program through the use of drivers and interface cards allows the user to interface with different instruments and input devices. It provides a seamless approach to integrate the camera with the software and control its operations.

The block diagram of the Lab-VIEW VI is shown in figure 3.7. The camera is initialized by the initialization block which sets up the camera and the drivers to operate the camera. The next block is the Get Image block which activates the camera to capture the image. The image captured by the camera is in a raw format, the next block is an image acquisition block which acquires the image from the camera and converts the image to one of the standard formats in this case a bitmap format. The final block after this is the exit block which handles the exit procedure and delinks the camera and brings it back to the initialisation phase. These blocks act like a set of instructions and are programmed to take images of the sample over a period of 70 minutes with an image taken every 5 minutes. The loop also saves the images automatically to a specified location in the memory.

The stored images were analysed to quantify the change in colour of the indicator. To measure the colour of the indicator a colour histogram of each image was recorded. The colour histogram was taken in a rectangular cross section of 200 by 200 pixels in the centre of the image; this was done as the centre of the sample displayed the most homogeneous mix with the indicator than at the edges. The histogram of the images provides the information of the colour in the primary Red-Green-Blue (RGB) colours. The RGB values of the images were plotted against the pH values.

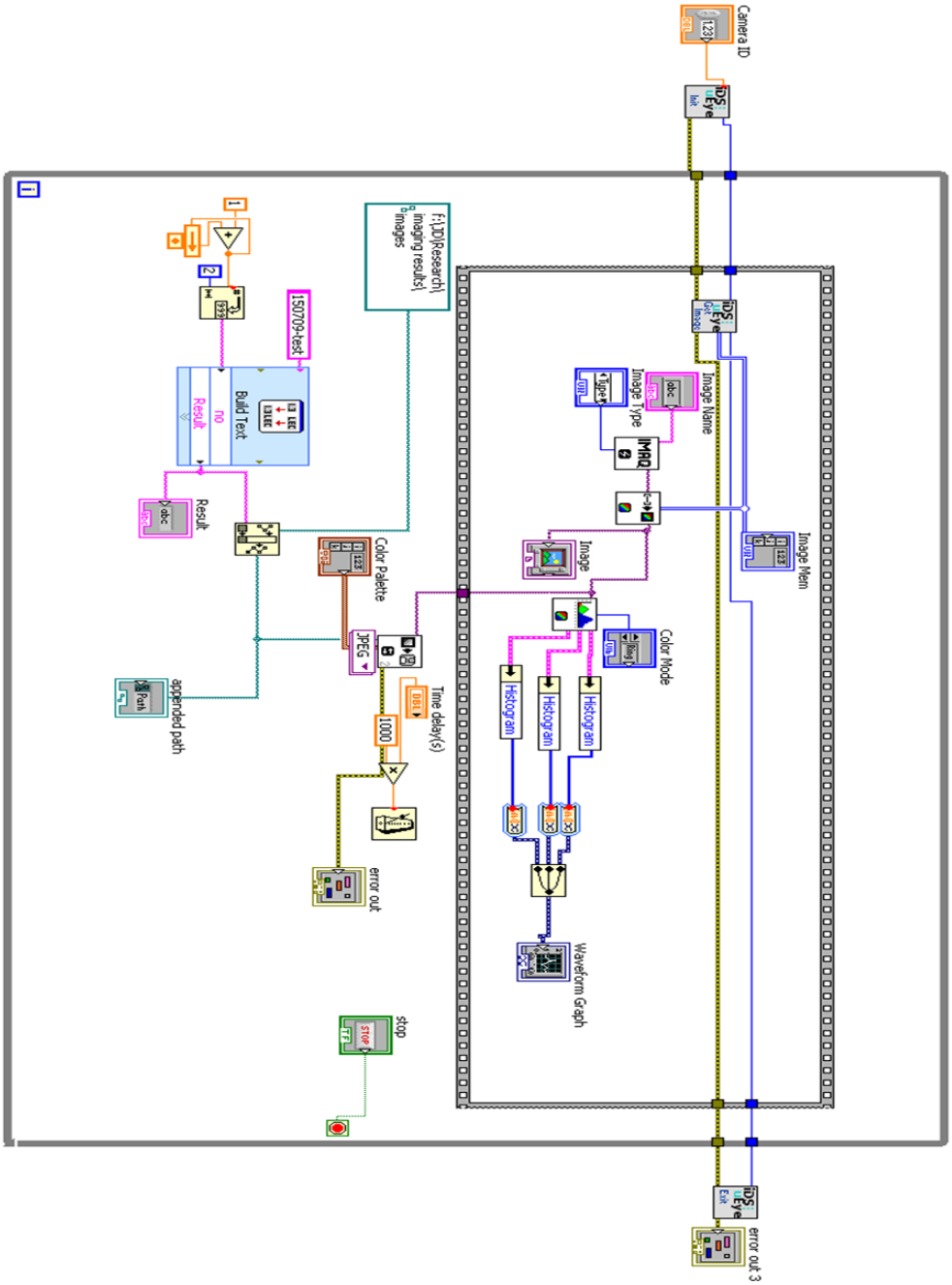
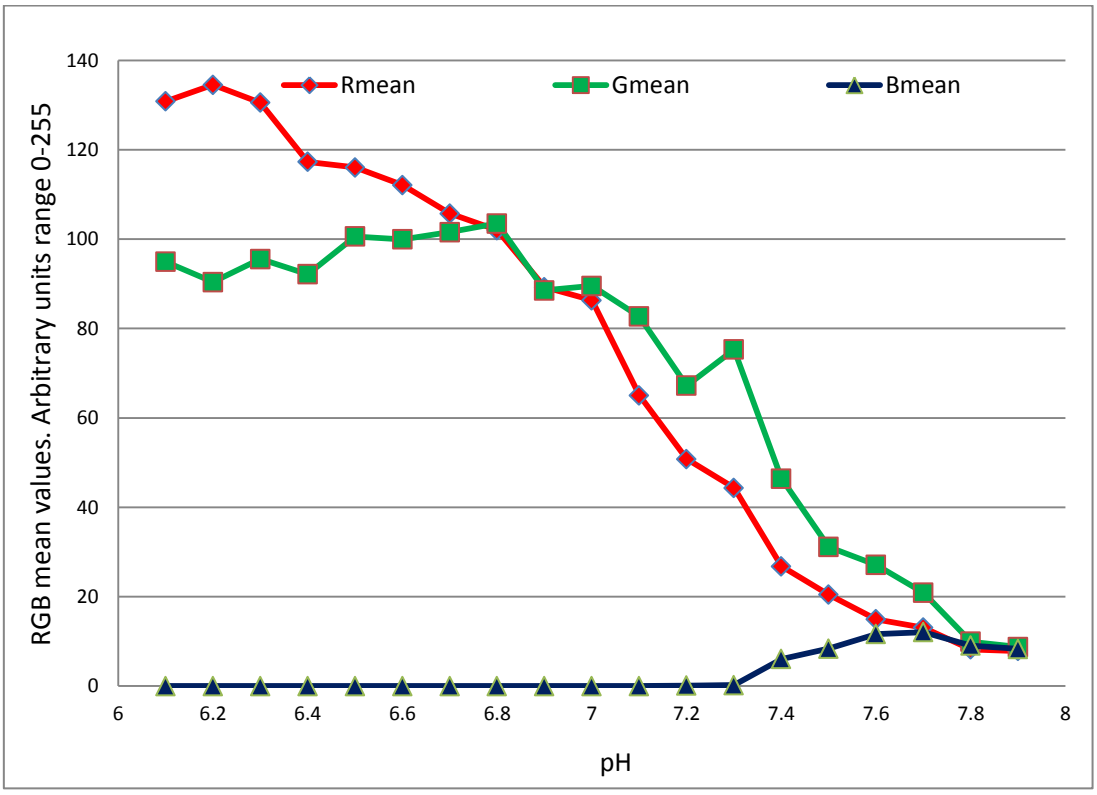


Figure 3.7: Block Diagram of the Lab view VI controlling the camera.

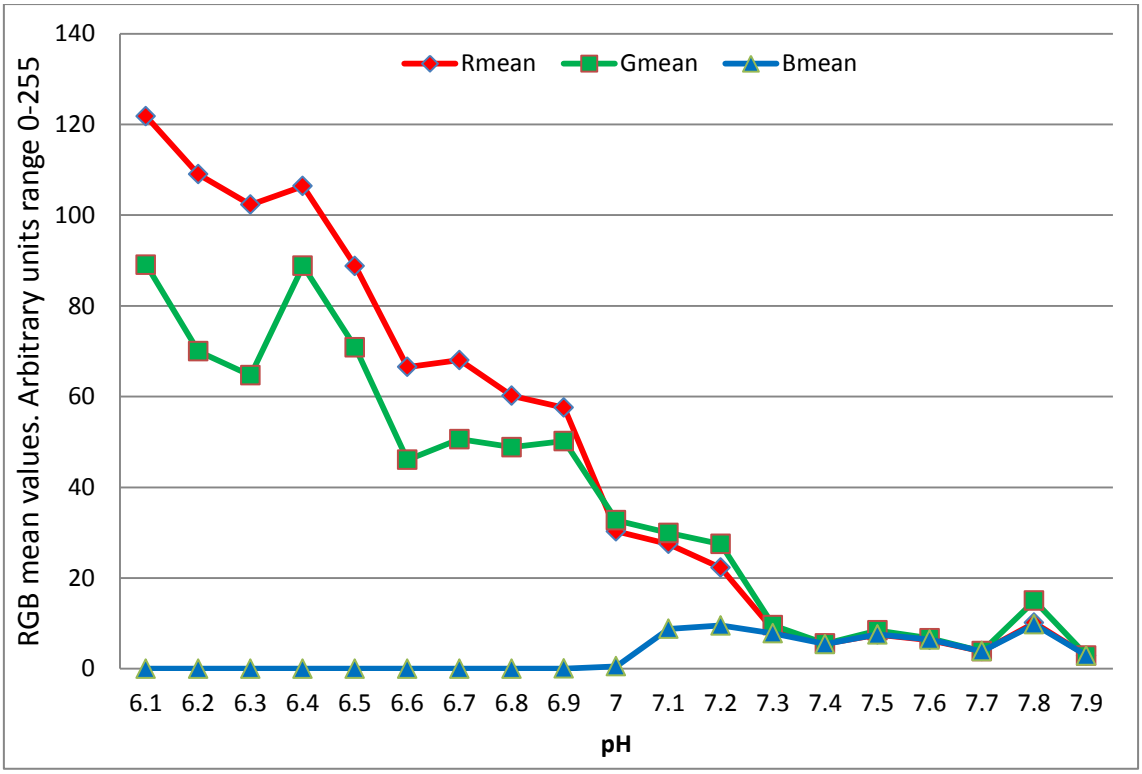
Figure 3.8 shows the RGB values plotted against varying pH for experiments carried out on three different days. Please refer to the appendix to see more plots. The trends in the graph are consistent with the change in colour of the indicator from yellow to blue. The colour yellow in terms of the RGB spectrum is represented as R=G=255 and B=0 and the colour blue is represented as R=G=0 and B=255. The red and green colours are higher at the ph values below 7.0 and start to taper down as the pH increases; the colour blue comes into effect after a pH value of 7.2.

The tests were conducted in certain conditions to check what effect it has on the trends of the colours. The samples were imaged in the presence and absence of ambient light and it was found the results were much better in the absence of ambient of light. The images for rest of the tests were taken in an enclosed environment so as to reduce the interference of changes to the ambient light leaving only the back light to affect the colour in the images.

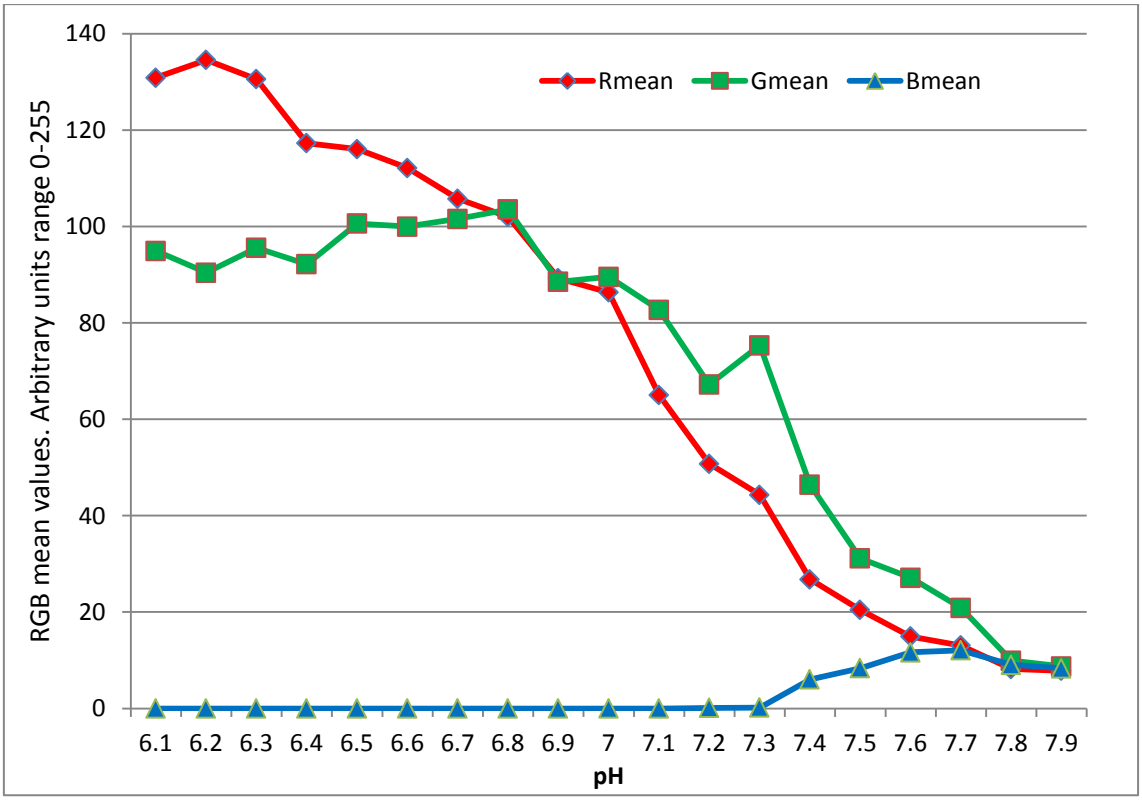
The samples were also imaged from different heights to check for the effect on the colour values and to determine a suitable height to place the camera. It was found the height of the camera did not influence the trends too much. The samples were imaged over the period of an hour to check if the drying of the sample has any effect over the colour values and if any changes are observed in the trends of the colours. The same trends in the colour values were observed even in the dried out samples.



(a)



(b)



(c)

Figure 3.8a, b & c: RGB colour plot with respect to the varying pH

The trends for the colours were repeated across the data collected from all the tests. The trends show a distinguishable change in colour across the range of the pH. The colour change is sharp across the pH values from 6.8 to 7.6 which is the range of the pH values for blood. This trend in the change in colour is mapped to a mathematical relation to determine the pH value from the colour of the indicator.

3.8 Conclusion

The design proposal looks at a colour based approach to measure the pH and employs the change in colour of an indicator as its principle. pH the measure of the acidity or basicity of a solution can be determined by electrochemical methods like glass electrodes and also by using indicator chemistry. Indicators are weak acids or bases which provide a visual measurementof pH and have been used before the potentiometric techniques to measure the pH. Bromothymol blue is chosen as a suitable indicator for the design for its range and colour change of yellow to blue. The design looks to quantify the change in colour of an indicator to the change in pH. This chapter presents the concept of Colorimetry the technique used to measure and quantify colour. Colour is the perception of the light spectrum entering the human eyes and can be quantified in the three primary colours red, green and blue. The chapter also describes the digital imaging process and how the information is stored in a digital image. The test bed establishes the relation between the colour change of the indicator to the change in pH and from the RGB plots it shows there is sharp response in the plot in the pH range 6.8 to 7.6 which is desirable. The relation between the colour and pH is mapped to a mathematical relation in the next chapter. The next chapter discusses the derivation of the mathematical relation and the associated algorithm.

Chapter Four: Mathematical Modeling

4.1 Introduction

The previous chapter presents the design proposal for portable blood gas analysis using colorimetry. The design quantifies the change in colour of the indicator in terms of the primary colours of red, green and blue. The colours values are processed using a mathematical equation defining the relation between the colour of the indicator and the pH. The mathematical equation determines the pH. The trends of the colour of the indicator and the change in pH are shown in the previous chapter. This chapter discusses the relation between the colour of the indicator and pH and also present derivation of the mathematical equation defining it. The chapter also discusses about curve fitting and regression analysis used to derive the mathematical equation.

4.2 Relation between the colour and pH

The indicator bromothymol blue has a colour change from yellow to blue for the pH range of 6.2 to 7.6 and this change in colour of the indicator has been reflected in the tests conducted to map the change. The tests show the colour changes from a greenish yellow to a blue for the pH samples and are shown in figure 4.1. This colour change is consistent with all the data collected in the tests.

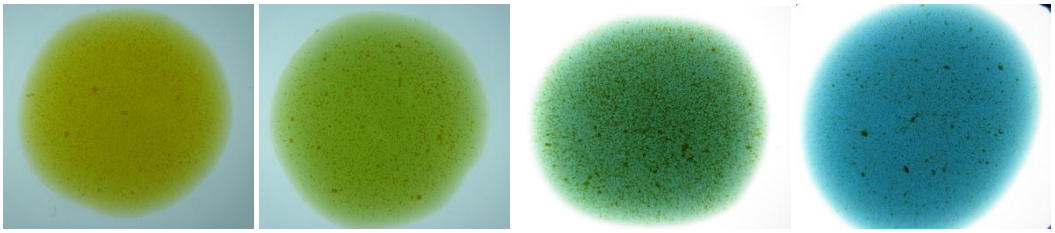
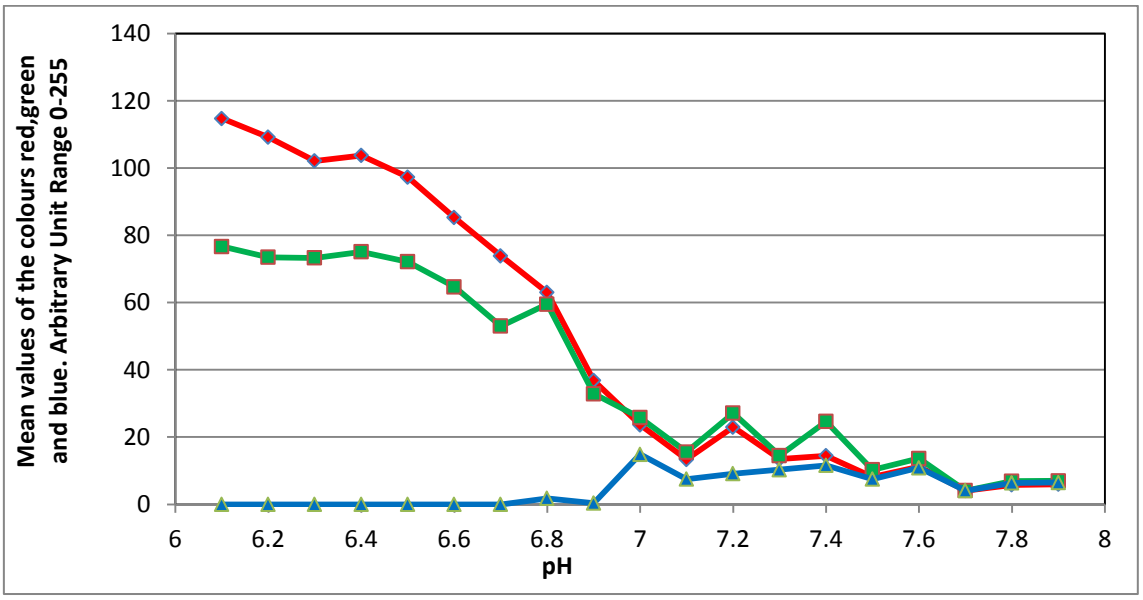


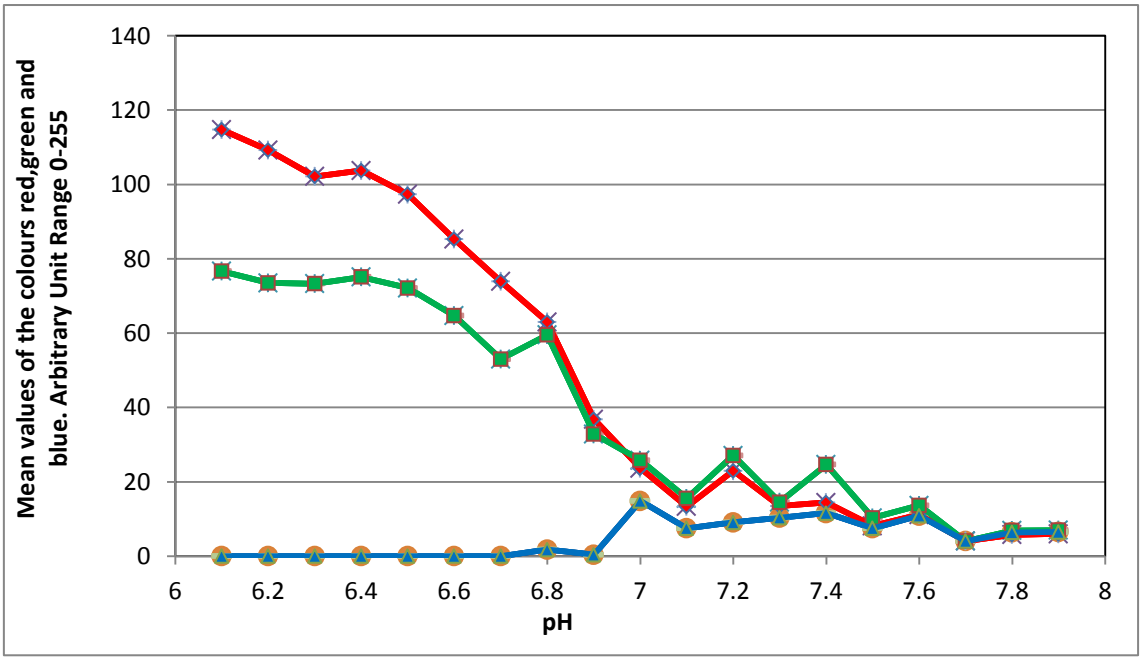
Figure 4.1: The change in colour from greenish yellow to blue as the pH changes from 6.1 to 7.9.

The colour of the indicator was measured in terms of the RGB colours and was plotted against the pH value. The plot shows the trends which reflect the change in colour of the indicator with respect to the change in colour of the indicator and similar trends have been observed over all the data collected. The pH range of human blood is from 6.8 to 7.8 and the pH value ofa healthy human being is 7.35 to 7.45, the response of the indicator and the trends in the RGB colour values with change in pH in this range are of interest.

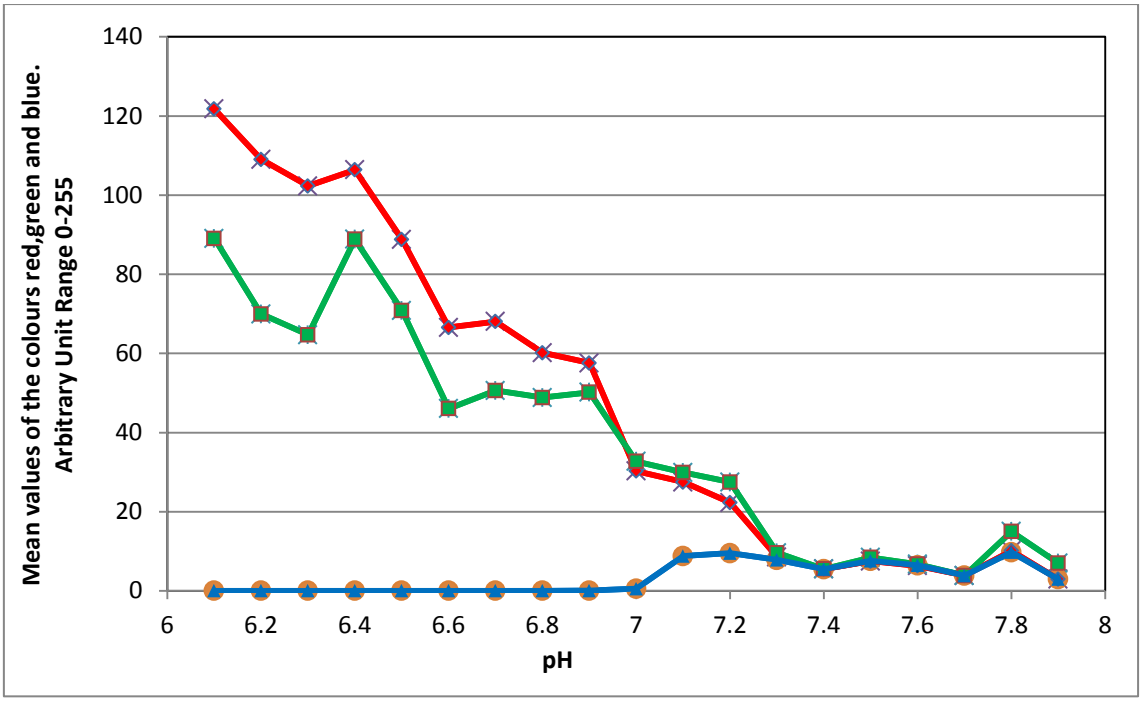
The data collected was tested to see the effects of the variation of the concentration of the bromothymol blue indicator. Bromothymol blue is a very strong colour for the indicator and a higher concentration of the indicator solution had a tendency to mask the change in colour of the indicator with the change in pH. The change in the concentration of the indicator varied the graphs slightly; but overall the trends of the colour change remained the same.



(a)



(b)



(c)

Figure 4.2a, b&c: The trends of the mean values of the colours red, green and blue across the pH range 6.1 to 7.9.

The observations from the trends show the colours red and green have a sharper response to the change in pH than the colour blue. The trends of the colour blue shows an inconsistent pattern in some cases showing a more pronounced change in value to the change in pH insome cases than on the others and is shown in the figure below. Due to the inconsistent and varying nature of the colour blue in the trends it was excluded from the data to search for an algorithm to determine a relation for the colour and pH.

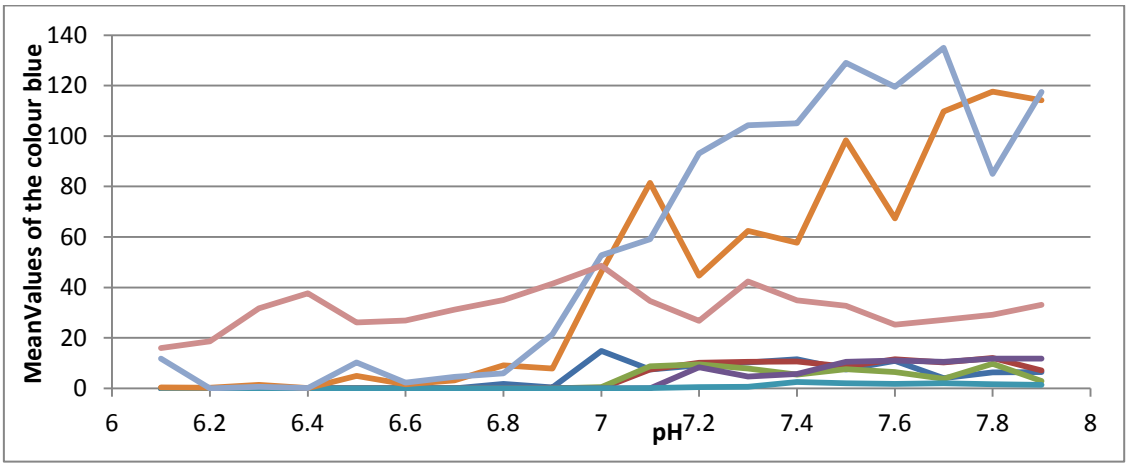


Figure 4.3: The trends for the colour blue with respect to the change in pH and the inconsistent nature of the trends across the ph range

The trends in the colour red and green were observed to establish a relation between the colour and the pH. From the data collected and the trends it was observed the colour green has a better response than the colour blue but in the pH range from 6.0 to 6.9 the response is almost flat, it shows a small drop from 6.9 to 7.2 and then starts to flatten again. The colour green has a shallow response in the human blood pH range.

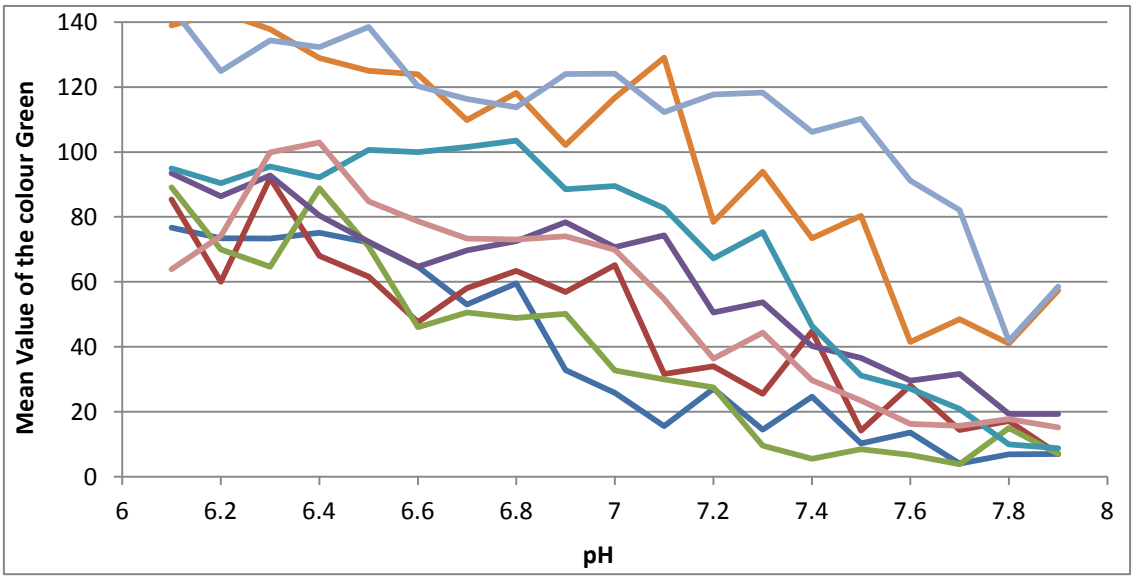


Figure 4.4: The trends for the colour green from the different samples show a flat response across the pH range 6.1 to 7.9; especially in the human blood pH range (6.8 to 7.8).

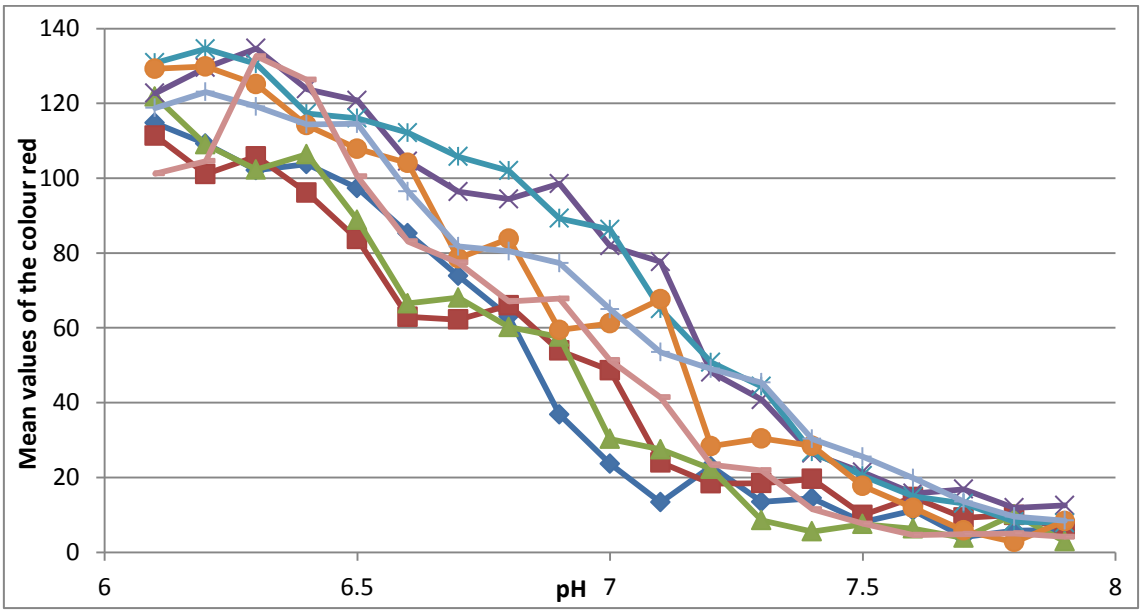


Figure 4.5: Trends of the colour red from the different sample sets across the pH range 6.1 to 7.9 showing the sharpest response to change in pH.

In comparison to the colour green, the response from the colour red is the sharpest and is consistent across the data collected and is shown in the figure 4.x which shows the change

in the colour red to the change in pH. The colour red has a sharp response across the whole range of the test samples from 6.1 to 7.9 and especially in the range of interest from 6.8 to 7.8 the trends show a near linear response. The figure shows the responses for the colour red to pH over the range from 6.1 to 7.9 and figure shows the response of the colour red to the pH in human blood pH range. The response of the colour in the blood pH range is sharp and is close to a linear response; given the characteristics of the trends and the response of the colour red it was selected to model a relation between the colour red and the pH.

The trends in the colour red were mapped to a mathematical equation using curve fitting and regression techniques. The mathematical equation forms the basis of the colorimetric algorithm to measure pH based on colour. The next sections explain about curve fitting and regression analysis and also discuss the derivation of the mathematical equation.

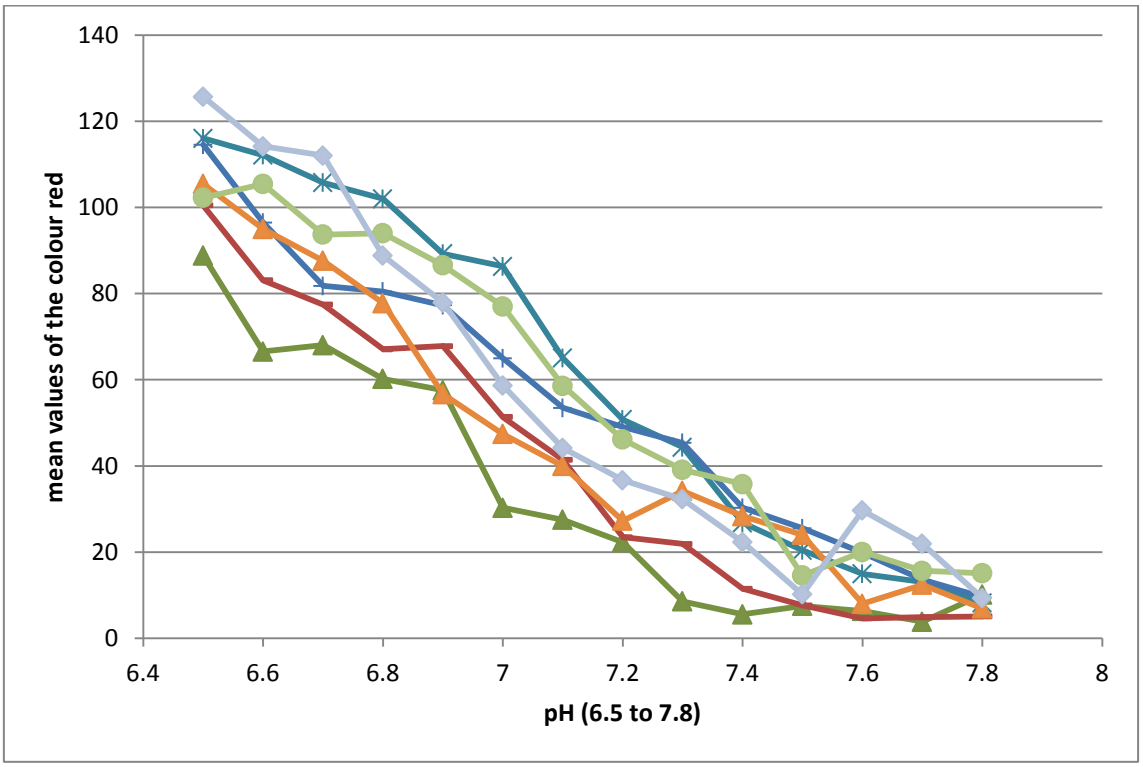


Figure 4.6: Trends of the colour red from the different sample sets show a consistent pattern and sharp nearly linear response sets in the human blood pH range.

4.3 Mathematical Relation

4.3.1 Curve Fitting and Regression Analysis

A mathematical function defines the relation between two variables and represents empirical data. It can also be used to compute the value of the dependent variable from the value of the independent variables. In most cases the underlying relation between the variables is known and it is observed in the data sets collected and this relation is represented in graphical forms as a curve. But there are cases in which the relationship is not evident directly from the data set and a mathematical equation representing the relation is required[63].

Curve fitting is the process of building a curve or a line governed by a mathematical equation which will best fit the data set. The goal of curve fitting is to find the parameter values which closely match the data set. The models used to fit to the data are dependent on adjustable parameters[64, 65].

Regression analysis is a process of analysing the data set and modelling the relation between the variables in the data set. Regression analysis allows us to predict the value of the dependent variable for a given independent variable when the other independent variables are held constant. The evaluation of the value of the dependent variable is a function of the independent variables, regression parameters and a random error term and is known as the regression function [66].

The regression function is the model describing the relation between the variables. The regression function determined only by a set of parameters is known as parametric regressions. There are many methods to determine the parametric relation between the dependent variable and the independent variables and different methods and models assume different regression functions and error terms. Linear regression and non linear regression are two models to determine the regression functions[67].

4.3.2 Linear Regression

Linear regression models the relation as a linear function of the regression parameters and random error. The models which define the relation are known as linear models. One of the simplest forms of linear regression is known as a simple linear regression. The simple regression model tries to fit a straight line to the data set and can be expressed as shown in equation 4.1 where y is the dependent variable, a0 is the y intercept, a1 is the slope of the regression line, x is the independent variable and e is the random error.

$$y = a_0 + a_1 x + e \tag{4.1}$$

The other model is known as a multiple linear regression it also models the regression function as a linear function but there are more than one independent variables in the model and is expressed in the equation 4.2 where y is the dependent variable, a_0, a_1, a_2,…a_n are the regression coefficients, x_1, x_2,…, x_n are the independent variables and e is the random error[66].

$$y = a_0 + a_1 x_1 \dots\dots + a_n x_n + e \tag{4.2}$$

Simple and multiple linear regressions both investigate the linear relation between the dependent variable and the independent variables while the simple regression focuses on the relation between one dependent variable to one independent variable, multiple regression focuses on the relation between one dependent variable and many independent variables.

4.3.3 Non Linear Regression

Non linear regression is another type of model applied to data, which assumes the relation between the dependent variable and the independent variables is a non linear function of the regression parameters. The non linear regression function in one of its forms is shown in equation 4.3

$$y = \frac{b}{1+e^{ax}} + e \tag{4.3}$$

Where y is the dependent variable, a & b are the model parameters, x is the independent variable and e is the random error. Non linear regression models are more complex than a

linear model in terms of model parameter estimation, model selection and variable selections. It requires a lot of iterations and computing to determine the regression parameters[64, 66].

4.3.4 Polynomial Regression

The relation between x and y doesn't always turn out to be a straight line and also a nonlinear relation is not better at describing the data; in cases like this polynomial regression provides us with a model which helps describe the behaviour of the data and its departures from linearity. Polynomial regression is a special case of linear regression; it applies a nonlinear model to the data but is still considered linear in statistics. This is because the regression function is linear in the unknown parameters estimated from the data. Polynomial regression applies an nth order polynomial to define the relation between the independent variable x and the dependent variable y[64, 66]. The general form of a polynomial regression model is shown in equation 4.4

$$y = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n + e \quad n < m\text{-}1 \qquad 4.4$$

where y is the dependent variable, $a_0, a_1, a_2, \ldots a_n$ are the regression coefficients, $x, x^2, \ldots x^n$ are the higher orders of the independent variable x and e is the random error and n is the order of the polynomial and m is the number of data points. The variable term x explains the continuous relation and the higher powers x^2, x^3, etc. explain the curves in the relation between x and y.

Polynomial models are best suited when interpolating the value within the range of the independent variables. The order of the polynomial is dependent on the number of data points but polynomials of the order above 6 are rarely used as the fit is not good and it doesn't provide any significant change in the error term.

4.3.5 Least Squares Method

To describe the relation between the dependent variable and the independent variables the regression parameters in the regression function need to be determined. The method of least squares is a method which is used to determine the regression parameters. The

least squares method can be applied to both linear and nonlinear regression models[64, 68].

The method of least squares is an important application in curve fitting and data fitting. The method provides the best fit line for a given set of data. It achieves this by minimizing the sum of squared residuals where a residual if the difference between the observed value and the fitted value provided by a model.

A typical simple linear regression model can be described as shown in equation 4.1, the parameters a_0 and a_1 determine the fit of the model and accuracy with which the model can predict the dependent variable from the independent variable. The equation can have several solutions for the values of a0 and a1 and the method least squares helps in finding the values of a0 and a1 which provides the best fit where the error involved is the sum of the squares of the difference between the value of the dependent variable y from the model and the values in the data set is minimum[69]. The equation to minimise the least squares error and the error function is shown below in equation 4.5

$$E(a_0, a_1) = \sum_{i=1}^{n} [y_i - (a_1 x_i + a_0)]^2 \quad 4.5$$

Where a0, a1 are the regression parameters and n represents the number of data points.

The goal is to minimise the total error for the line of best fit for the given data and for a minimum to occur the partial derivatives are taken and set to zero [63, 66, 69].

$$0 = \frac{\partial}{\partial a_0} \sum_{i=1}^{n} [y_i - (a_1 x_i + a_0)]^2 = 2 \sum_{i=1}^{n} (y_i - a_1 x_i - a_0)(-1) \quad 4.6$$

and

$$0 = \frac{\partial}{\partial a_1} \sum_{i=1}^{n} [y_i - (a_1 x_i + a_0)]^2 = 2 \sum_{i=1}^{n} (y_i - a_1 x_i - a_0)(-x_i) \quad 4.7$$

The above equations can be simplified to

$$a_0 . n + a_1 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \quad 4.8$$

and

$$a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i \quad 4.9$$

When the above equations are solved for a0 and a1 we get the following equations for them

$$a_0 = \frac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i y_i \sum_{i=1}^{n} x_i}{m\left(\sum_{i=1}^{n} x_i^2\right) - \left(\sum_{i=1}^{n} x_i\right)^2} \quad 4.10$$

$$a_1 = \frac{\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\left(\sum_{i=1}^{n} x_i^2\right) - \left(\sum_{i=1}^{n} x_i\right)^2} \quad 4.11$$

The least squares method can also be applied for polynomial regression and the general form the polynomial model is shown in equation 4.12. In the case of the polynomial model the constants a_0, a_1, a_2, a_3,…,a_n are chosen to minimise the least squares error.

$$y = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \quad 4.12$$

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \quad 4.13$$

Where n <m-1, n is the order of the polynomial and m is the number of data points.

The error function for minimising the least squares is given as

$$E(a_0, a_1, \dots, a_n) = \sum_{i=1}^{m} [y_i - P_n(x)]^2 \quad 4.14$$

$$= \sum_{i=1}^{m} y_i^2 - 2\sum_{i=1}^{m} P_n(x_i) y_i + \sum_{i=1}^{m} \left(P_n(x_i)\right)^2 \quad 4.15$$

$$= \sum_{i=1}^{m} y_i^2 - 2\sum_{i=1}^{m} \left(\sum_{j=0}^{n} a_j x_i^j\right) y_i + \sum_{i=1}^{m} \left(\sum_{j=0}^{n} a_j x_i^j\right)^2 \quad 4.16$$

$$= \sum_{i=1}^{m} y_i^2 - 2\sum_{j=0}^{n} a_j \left(\sum_{i=1}^{m} y_i x_i^j\right) + \sum_{j=0}^{n} \sum_{k=0}^{n} a_j a_k \left(\sum_{i=1}^{m} x_i^{j+k}\right) \quad 4.17$$

Similar to the previous case of simple linear regression for the error function to be minimised it is necessary that the partial derivatives $\frac{\partial E}{\partial a_j} = 0$, for each j=0,1,2,…,n.

$$0 = \frac{\partial E}{\partial a_j} = -2\sum_{j=0}^{n} a_j \left(\sum_{i=1}^{m} y_i x_i^j\right) + 2\sum_{k=0}^{n} a_k \left(\sum_{i=1}^{m} x_i^{j+k}\right) \quad 4.18$$

This gives n+1 normal in the n+1 unknown aj,

$$\sum_{k=0}^{n} a_k \left(\sum_{i=1}^{m} a_j x_i^{j+k}\right) = \sum_{i=1}^{m} y_i x_i^j \quad 4.19$$

for each j=0,1,2,…,n. The equation can be expanded as shown below, where m is the number of data points, $a_0,a_1,a_2,\ldots,a_n$ are the constants in the polynomial, n is the order of the polynomial , x is the independent variable and y is the dependent variable.

$$a_0\sum_{i=1}^{m}x_i^0+a_1\sum_{i=1}^{m}x_i^1+a_2\sum_{i=1}^{m}x_i^2+\cdots+a_n\sum_{i=1}^{m}x_i^n=\sum_{i=1}^{m}y_ix_i^0$$

$$a_0\sum_{i=1}^{m}x_i^1+a_1\sum_{i=1}^{m}x_i^2+a_2\sum_{i=1}^{m}x_i^3+\cdots+a_n\sum_{i=1}^{m}x_i^{n+1}=\sum_{i=1}^{m}y_ix_i^1$$

to

$$a_0\sum_{i=1}^{m}x_i^n+a_1\sum_{i=1}^{m}x_i^{n+1}+a_2\sum_{i=1}^{m}x_i^{n+2}+\cdots+a_n\sum_{i=1}^{m}x_i^{2n}=\sum_{i=1}^{m}y_ix_i^n$$

The above equations can be solved in the same way as the equations for the simple regression model and will generate the values of the constants which give the least error and best fit for the given data.

4.3.6 Mathematical Relation

The method of least squares allows the user to apply the regression models to the data and determine the regression coefficients for the model chosen for analysis. To determine the mathematical equation defining the relation between the colour red and pH the method of least squares was applied to the data collected.

The mean values of the colour red collected from the histogram over the pH range 6.1 to 7.9 were used to derive the mathematical relation. The mean values for the colour red were normalised to a scale of 0 to 1. The normalised values of the colour red are shown below in table 4.1.

Table 4.1: The normalised values of the colour red for the different samples used regression and the average of the normalised values

pH	R30-6	R 01-07	R03-07	R05-07	R07-07	R27-12	R29-12	R20-04	R16-02	R18-02	R19-02	AVG
6.1	1.000	1.000	1.000	0.902	0.971	0.995	0.963	0.756	1.000	0.915	1.000	0.955
6.2	0.950	0.902	0.893	0.958	1.000	1.000	1.000	0.782	0.914	1.000	0.790	0.926
6.3	0.886	0.947	0.836	1.000	0.969	0.963	0.966	1.000	0.706	0.714	0.966	0.905
6.4	0.901	0.855	0.871	0.912	0.864	0.877	0.925	0.951	0.506	0.663	0.587	0.810
6.5	0.843	0.737	0.723	0.887	0.854	0.827	0.926	0.750	0.552	0.676	0.763	0.776
6.6	0.734	0.540	0.535	0.754	0.823	0.797	0.768	0.615	0.609	0.530	0.532	0.658
6.7	0.631	0.533	0.548	0.688	0.773	0.597	0.640	0.571	0.526	0.670	0.619	0.618
6.8	0.532	0.569	0.482	0.672	0.743	0.638	0.629	0.490	0.359	0.499	0.381	0.545
6.9	0.296	0.454	0.460	0.706	0.643	0.446	0.601	0.496	0.195	0.343	0.414	0.459
7	0.177	0.404	0.230	0.570	0.619	0.460	0.493	0.368	0.193	0.203	0.396	0.374
7.1	0.084	0.170	0.207	0.536	0.452	0.511	0.393	0.290	0.099	0.067	0.222	0.275
7.2	0.171	0.116	0.163	0.295	0.339	0.202	0.354	0.150	0.242	0.088	0.257	0.216
7.3	0.085	0.117	0.048	0.235	0.288	0.218	0.322	0.138	0.050	0.062	0.160	0.157
7.4	0.094	0.128	0.022	0.119	0.150	0.202	0.190	0.057	0.029	0.035	0.062	0.099
7.5	0.037	0.036	0.039	0.078	0.100	0.118	0.149	0.027	0.019	0.007	0.087	0.063
7.6	0.064	0.081	0.029	0.031	0.056	0.072	0.099	0.004	0.005	0.010	0.040	0.045
7.7	0.000	0.029	0.008	0.041	0.041	0.024	0.044	0.006	0.005	0.000	0.023	0.020
7.8	0.016	0.036	0.061	0.000	0.003	0.000	0.009	0.007	0.000	0.001	0.000	0.012
7.9	0.017	0.000	0.000	0.005	0.000	0.044	0.000	0.000	0.004	0.001	0.010	0.007

The plot of the normalised values of the colour red from different samples against the ph values is shown in figure 4.7 and as it has been observed previously in the plots of the colour red against pH they all follow a similar trend. The normalised values of the colour red were averaged and have been shown in the last column of the table 4.7 and have also been plotted in the figure 4.7. The trendline in the figure 4.7 shows the average values. The method of least squares was applied to all the curves in the plot to determine the curve which best fit the data. As observed from the plot the relation between the colour and the pH is not completely linear and as explained in section 4.3.4, a non-linear relation will not provide a better explanation and hence polynomial regression was chosen to define the relation.

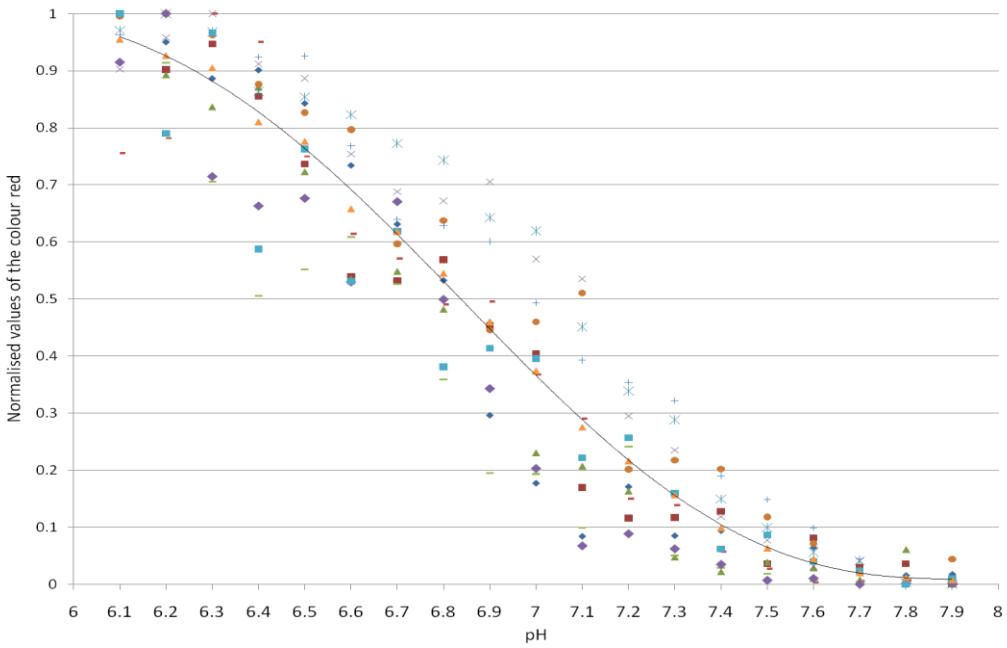


Figure 4.7: A plot of the normalised values of the colour red against the pH. The plot also shows the average values on the colour red.

Polynomial regression using the method of least squares was applied to all the curves in the plot to find the curve which will best fit the data. To define the equation in terms of the colour red a regression of pH on the colour red was done[64]. The results of the regression analysis presented different equations for the data, but the equation which best fit the data was from the curve for the average values. The curve with average values was used to define the equation in terms of the colour red and a regression of pH on the colour red was performed. As a part of the polynomial regression, polynomials from the order of two and higher were applied to the curve. Table 4.2 shows the comparison of the predicted values of the pH for different orders of the polynomial. Polynomials of orders higher than six weren't applied as they generally don't provide any improvement in the error function. It can be observed from the table the fifth order polynomial predicts the pH closest to the actual value and going to the sixth order doesn't increase the accuracy of the predictions. The fifth order polynomial model was used define the relation and the polynomial equation is shown in equation 4.21.

$$y = -29.554x^5 + 76.472x^4 - 73.748x^3 + 32.37x^2 - 7.5441x + 7.8908 \qquad 4.21$$

Where y represents the value of pH and x represents the mean value of the colour red.

Table 4.2: The table shows the comparison of the predicted pH values for the different orders of the polynomial equation determined by polynomial regression.

pH	*Predicted pH - first order*	*Predicted pH - second order*	*Predicted pH - third order*	*Predicted pH - fourth order*	*Predicted pH - fifth order*	*Predicted pH- sixth order*
6.1	6.141	6.214	6.113	6.154	6.108	6.114
6.2	6.187	6.244	6.192	6.198	6.208	6.204
6.3	6.221	6.267	6.246	6.234	6.268	6.262
6.4	6.372	6.377	6.447	6.403	6.439	6.438
6.5	6.427	6.418	6.505	6.464	6.481	6.484
6.6	6.615	6.573	6.663	6.659	6.622	6.626
6.7	6.679	6.629	6.705	6.716	6.675	6.677
6.8	6.796	6.735	6.776	6.807	6.781	6.778
6.9	6.932	6.868	6.857	6.897	6.908	6.902
7	7.069	7.008	6.949	6.976	7.019	7.017
7.1	7.226	7.180	7.085	7.081	7.121	7.125
7.2	7.320	7.289	7.190	7.167	7.181	7.186
7.3	7.415	7.402	7.319	7.284	7.263	7.265
7.4	7.508	7.515	7.471	7.439	7.397	7.394
7.5	7.565	7.587	7.580	7.561	7.526	7.521
7.6	7.594	7.625	7.641	7.634	7.612	7.608
7.7	7.633	7.676	7.728	7.742	7.751	7.751
7.8	7.646	7.692	7.758	7.781	7.804	7.806
7.9	7.654	7.702	7.776	7.804	7.836	7.840

The fifth order polynomial equation was chosen as it provides better resolution and avoids errors due to rounding up in determining the pH. The higher orders did not present any improvements in the resolution or rounding errors.

4.4 Results

The model was applied to all data sets to determine the accuracy of pH prediction. The table 4.3 shows the predicted values of the pH for all the different data sets. The table shows the predicted values of the ph in the pH range of human blood. The results show the model predicts the value the pH with a maximum deviation of 5% from the expected results.

Table 4.3: The table shows a comparison of the predicted pH values for the different sample data sets.

pH	R30-06	R 01-07	R03-07	R05-07	R07-07	R27-12	R29-12	R20-04	R16-02	R18-02	R19-02	R22-07	R23-07	R30-07	R01-09	R02-09	R16-12	R20-12	R23-12	AVG	95% Confidence Interval
6.8	6.898	6.763	6.867	6.768	6.639	6.753	6.855	6.879	6.937	6.790	7.016	6.810	6.744	6.594	6.634	6.934	6.813	6.555	6.847	6.801	0.054
6.9	7.145	6.927	6.897	6.726	6.775	6.982	6.885	6.871	7.175	7.010	6.979	7.012	6.581	6.612	6.729	6.758	6.837	6.673	6.937	6.881	0.073
7	7.308	6.991	7.184	6.887	6.804	6.967	6.992	7.027	7.178	7.196	7.000	7.092	6.938	6.833	6.843	7.051	7.263	6.936	7.080	7.028	0.064
7.1	7.513	7.325	7.220	6.924	6.993	6.912	7.085	7.115	7.385	7.514	7.206	7.160	6.751	6.907	7.029	6.830	7.197	7.062	7.197	7.099	0.094
7.2	7.318	7.462	7.297	7.163	7.106	7.261	7.122	7.328	7.104	7.445	7.159	7.313	7.115	6.869	7.148	7.186	7.168	7.187	7.273	7.194	0.060
7.3	7.510	7.459	7.629	7.235	7.160	7.237	7.155	7.354	7.555	7.532	7.308	7.222	7.269	7.264	7.230	7.138	7.344	7.084	7.327	7.289	0.068
7.4	7.487	7.428	7.740	7.439	7.367	7.260	7.327	7.584	7.654	7.641	7.559	7.297	7.224	7.074	7.275	7.358	7.469	7.223	7.486	7.382	0.076
7.5	7.664	7.775	7.665	7.543	7.482	7.421	7.403	7.702	7.703	7.774	7.481	7.365	7.331	7.364	7.810	7.596	7.514	7.452	7.784	7.548	0.070
7.6	7.572	7.579	7.710	7.689	7.609	7.545	7.518	7.810	7.778	7.757	7.637	7.772	7.295	7.617	7.615	7.613	7.359	7.581	7.363	7.582	0.063
7.7	7.810	7.810	7.810	7.657	7.659	7.708	7.681	7.801	7.779	7.810	7.706	7.628	7.724	7.676	7.767	7.810	7.575	7.635	7.494	7.707	0.040
7.8	7.745	7.776	7.577	7.810	7.810	7.810	7.810	7.794	7.810	7.803	7.810	7.810	7.810	7.810	7.789	7.758	7.810	7.810	7.810	7.786	0.024

Figure 4.8 shows the plot of the predicted values of the pH value of 7.3 across the data sets. The plot shows the predicted values have a 5% deviation from the expected value and also shows the 95% confidence intervals.

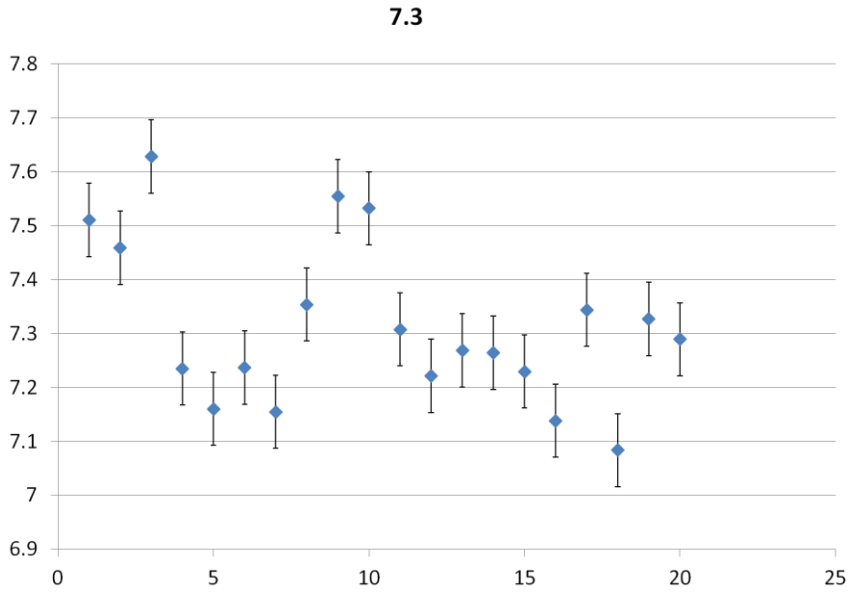


Figure 4.8: A plot of the predicted values of pH for 7.3 across all the data sets with the 95% confidence intervals

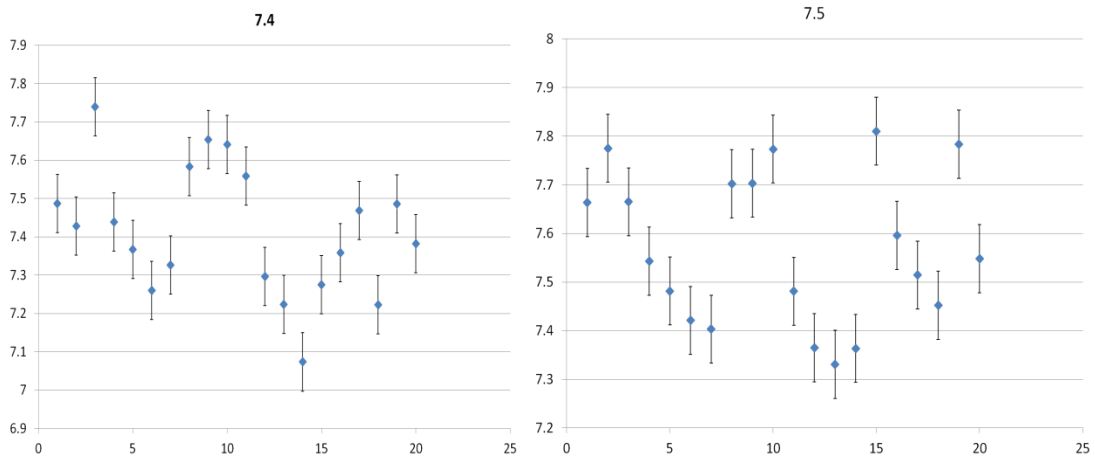


Figure 4.9: A plot of the predicted values of pH for 7.4 and 7.5 across all the data sets with the 95% confidence intervals.

The plots for the predicted value of 7.4 and 7.5 shown in figure 4.9 also show the same deviation of 5% from the expected value and also shows the 95% confidence intervals.

4.5 Conclusion

The trends from the plot for colour values against the pH show the red and green colours with the most consistent pattern. The red colour shows the sharpest response to the change in pH and is consistent across all the data sets. The trends of the colour red against the pH value have been used to model a mathematical relation between the colour and pH. Curve fitting and regression analysis provide the tools to analyse the trends in a given data set and model the trends and relation between the variables in terms of a mathematical equation. Linear regression and non-linear regression are the common models applied to the data to determine a mathematical model. Polynomial regression is a modified version of linear regression which is applied to the data where the linear cannot be fit to the data and the non linear model cannot explain the data fully. The polynomial regression model was applied to the data and the regression model coefficients were determined using the method of least squares. A fifth order polynomial equation provided the best fit to the data and also provides the necessary resolution in the pH values. The mathematical model predicts the pH with a maximum deviation of 5% from the expected values. The relation between the colour and pH mapped as a fifth order polynomial is implemented in hardware in the next chapter. The next chapter discusses design and implementation of the different hardware blocks for the mathematical equation.

Chapter Five: Image based Colorimetric pH and pCO_2 analyser

5.1 Introduction

In chapters three and four the indicator chemistry was presented and explained the change in colour of an indicator due to the change in pH. The chapter also presented the method of Colorimetry which quantifies the value of colour in the three primary colours of red, green and blue. The experiments conducted to map change in colour of the indicator to the change in pH showed a definite and repeating trend in which the colour red showed the sharpest response. The trends in the colour red were analysed using the curve fitting techniques to map the relation between the colour and pH as a mathematical equation. The mathematical equation forms the basis of the image based colorimetric pH and pCO_2analyser.In this chapter, the implementation of the colorimetric algorithm is presented. A block diagram of the image based colorimetric ph and pCO_2 is shown in figure 5.1.

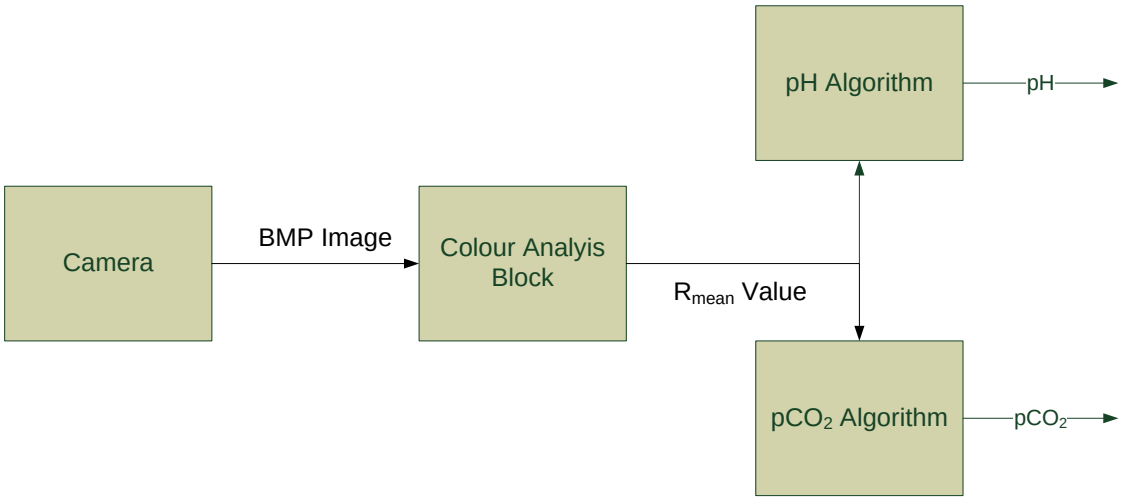


Figure 5.1: Block diagram of the image based colorimetric pH and pCO_2analyser.

The analyser consists of four blocks the camera, the colour extraction block and finally the ph and pCO_2 algorithms. The camera is the front end of the device which images the sample mixed with the indicator and captures the change in colour of the indicator as a

bitmap(bmp) image. The bmp image is processed in thecolour analysis block to generate the mean value of the colour red. The pH and pCO_2 algorithms process the mean value of the colour red to calculate the value of pH and pCO_2.The following sections explains the implementation of the colour extraction block and the ph and pCO_2 algorithms.

5.2 Colour Analysis Block

The colour analysis block is the first step in the colorimetric analyser. The block processes the bmp image received from the camera and extracts the mean value of the colour red from the bmp image. The mean value of the colour red is passed on to the pH and pCO_2 algorithm blocks to calculate the pH and pCO_2 values.

5.2.1 Bitmap (bmp) Image Format

The bitmap is one of the standard file formats to store a digital image.Bitmaps is a rectangular collection of pixels and each pixel holds the colour value. Figure 5.2 shows a representation of bitmap image with each pixel of 24 bits in 3 continuous bytes holding the value of the colour[70]. Each colour consisting of One byte (8 bits) for Red, 1 byte (8 bits) for Green and 1 byte (8 bits) for Blue, where the windows bmp file are characterized by the number of pixels and the colour depth per pixel[70, 71].

The bitmap file stores an image in four different formats; 1 bit, 8 bits, 16 bits and 24bits. These different formats represent the size of the data registers holding the information of the colour in each of the pixels in the image. In the 24 bit bitmap, it uses 24 bits of data to document each single pixel in the bitmap and the structure is divided into 8 bit pieces for each pixel. The first 8-bits hold the blue colour value, the next 8 bits hold the green colour value, and the final 8 bits hold the red colour value. Each colour value can represent 256 different states 0 - 255 and the three combined can represent a total of 16777216 colours[71].

The 1 bit format has one bit to represent the colour information of each pixel and is used generally for black and white images with a 1 representing the colour white and a 0 representing the colour black. Similarly the 8 bits and 16 bits formats use one byte and two bytes respectively to represent the information of the colour in each pixel of the image.

The 8bit format has only one byte for all the colour information; as a result of this the depth of colour information will be limited. To overcome this a colour table is included after the header information and before the pixel data in BMP File.The table for a 256-colour bitmap can hold up to 256 colours. There is a standard Windows palette of 256-colours, but bitmaps can also have custom palettes. The byte assigned for each pixel doesn't hold the value of the colour instead holds an index value of the colour table, the index value determines the colour based on the values stored for the colours red, green and blue.

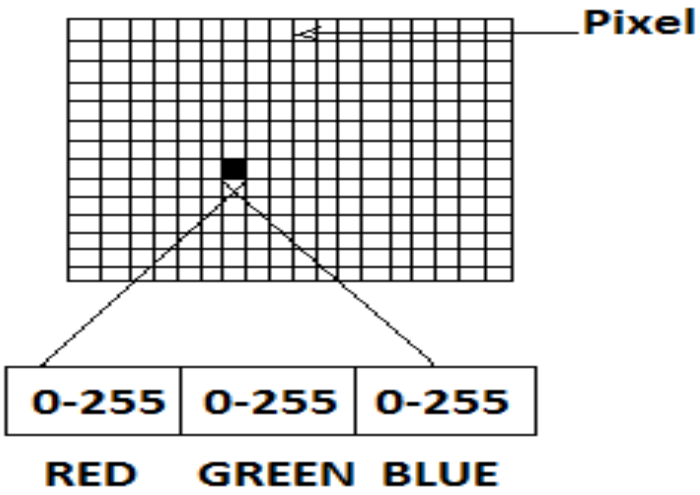


Figure 5.2:Colour Pixel[70]

For a bitmap with large dimensions, this palette indexing method of documenting colours creates a huge savings in file size. In general, 256-colour bitmaps have much smaller file sizes than a 24-bit bitmap with the same dimensions.

The format in which a bitmap is read is as follows;

1. File Header file is read
2. Image Header Information and
3. Bitmap pixel Data.

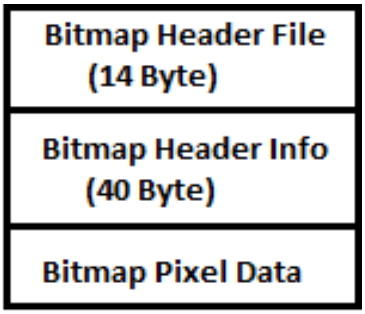


Figure 5.3: BMP File Structure

5.2.1.1 BMP ImageFile Header

This block is the starting of the block of bytes where the file is identified. A description of the contents of a BMP file header is shown in Table 5.1. In an application, the block file is checked to see whether the file is a BMP and whether the file is damaged. The first two bytes of the BMP file format are the character 'B' then the character 'M' in 1-byte ASCII encoding and are usually stored first. The Colour Map sizes are normally 2, 16 or 256 entries, but can be smaller if the image does not need a full set of colours[70].

Table 5.1: BMP File Header

OFFSET#	SIZE	PURPOSE
0000h	2 bytes	The header field used to identify the BMP & DIB file is 0x42 0x4D in Hexadecimal, same as BM in ASCII. The following entries are possible: BM-Windows 3.1x,95,NT,…etc; and it is not mandatory unless file size is greater or equal to SIGNATURE BA-OS/2 struct Bitmap Array CI-OS/2 struct Colour Icon &CP-OS/2 const Colour Pointer IC-OS/2 struct Icon *PT-OS/2 Pointer
0002h	4 bytes	The size of the BMP file in byte
0006h	2 bytes	Reserved; actual value depends on the application that creates the image
0008h	2 bytes	Reserved; actual vale depends on the application that creates the image
000Ah	4 bytes	The offset, i.e. starting address, of the byte where the bitmap image data (Pixel Array) can be found.

5.2.1.2 BMP Image Header Information

Information about the image in detail is in the BMP Header Information, where the image is displayed and viewed on the screen. The block also matches the header used internally by Windows and OS/2 and has several different variants. All of them contain a 32 bit field, specifying their size, so that an application can easily determine which header is used in the image[70].

Table 5.2: BMP Image Information Header

Size	Header Name	OS Support	Features Added (incremented)
12	BITMAPCOREHEADER OS21XBITMAPHEADER	OS/2 and also all Windows versions since Windows 3.0	
64	BITMAPCOREHEADER2 OS22XBITMAPHEADER	OS/2	Adds Half toning. Adds RLE and Huffman 1D compression.
40	BITMAPINFOHEADER	all Windows versions since Windows 3.0	Removes RLE-24 and Huffman 1D compression. Adds 16bpp and 32bpp pixel
52	BITMAPV2INFOHEADER	Undocumented	Removes optional RGB bit masks. Adds mandatory RGB bit masks.
56	BITMAPV3INFOHEADER	Undocumented	Adds mandatory Alpha channel bit mask.
108	BITMAPV4HEADER	all Windows versions since Windows 95/NT4	Adds Colour Space type and Gamma correction
124	BITMAPV5HEADER	Windows 98/2000 and newer	Adds ICC Colour Profiles

5.2.1.3 Pixel Data

The bitmap pixel data is stored upside down from its actual image as depicted in figure 5.4 showing the complete structure of the bmp image file. Therefore when reading or writing data from a bitmap, it is important to start at the bottom left corner and from left to right, and bottom to top in pixel unit. Therefore the RGB is also written in reverse order, starting with blue, green and then red[70].

A 24 bit pixel is formatted as the following:

Red (8 bits) = 11011011, Green (8 bits) = 11110110, and Blue (8 bits) = 01111110.

8 bits is able to hold 256 possible values ranging from 0-255. An 8-bit bitmap requires only 8 bits of data storage space to document each pixel. This shorthand is accomplished by including a palette in the file, which is placed after the header information, and before the pixel data. A palette is also called a Colour Table. Each pixel in the bitmap that is to be displayed in colour number 1 will have data equal to" 1" in binary numbers.

The structures of a 256 colour (8-bit) bitmap are as follows:

- Bitmap header;
- Indexed palette information;
- Pixel data requiring 8-bits for each pixel.

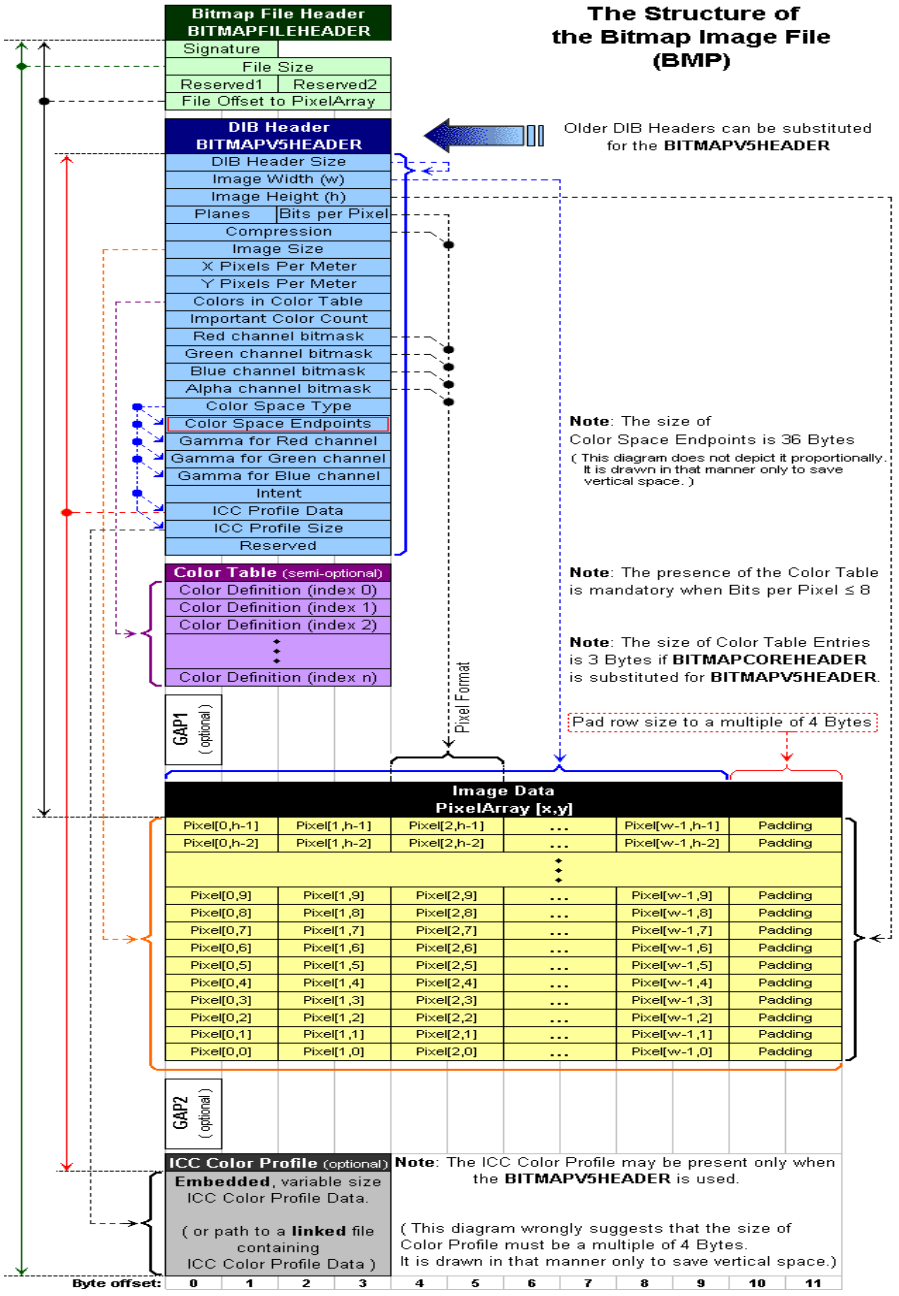
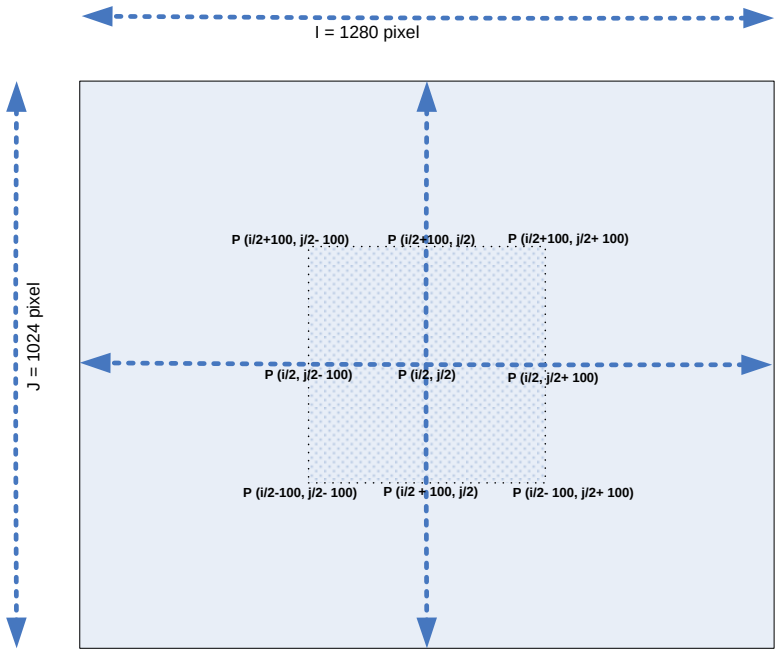


Figure 5.4: Structure of BMP file[70]

5.2.2 Implementation of the colouranalysis block

The colour analysis block is an algorithm to determine the value of the colour red from the image. The algorithm extracts the RGB information of the colour from the image and determines the mean value of the colour Red from a window of 200x 200 pixels from the centreof the 1280 x 1024 pixel image as shown in figure 5.5. The colour information is retrieved from the centre of image as it displays the most homogenous mix compared to the indicator at the edges of the image.



$$R_{(Red)} = 1/N \sum_{m=(i/2-100)}^{m=(i/2+100)} \sum_{n=(j/2-100)}^{n=(j/2+100)} P(i, j)$$

Figure 5.5:Reading of the pixel data from the centre of the image.

The BMP image colour analysis block is composed of the following blocks:

- Register Interface Block.
- Colour Table.
- BMP image Colour Extraction.
- System control and Initialization.
- Address Mapping and Memory Interface Block. As shown in figure 5.6.

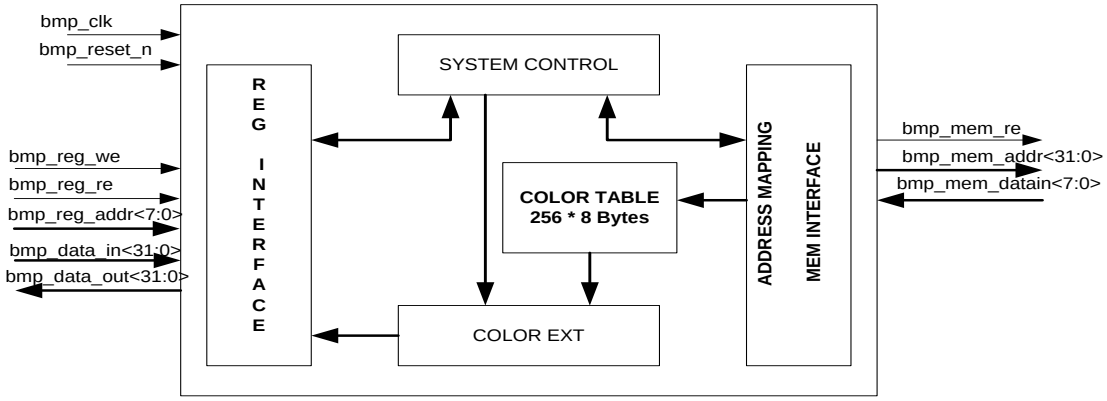


Figure 5.6:Block diagram of the Colour Analysis Block

5.2.2.1 External I/O Interfaces

The BMP Image colour analysis controller block interfaces to the other blocks on-chip using the following signals listed in Table 5.3.

Table 5.3:BMP Controller IO Interface

Name	Dir	Width	Rst	Description
bmp_clk	in	1	0	Controller clock
bmp_reset_n	in	1	0	Controller reset
RegisterInterface				
bmp_reg_re	in	1	0	Controller register read enable
bmp_reg_we	in	1	0	Controller register write enable
bmp_reg_addr	in	8	0	Controller register address
bmp_data_in	in	32	0	Controller register Data in
bmp_data_out	out	32	0	Controller register Data out
External Memory Interface				
bmp_mem_re	out	1	0	Controller memory read enable
bmp_mem_addr	out	32	0	Controller memory address
bmp_mem_data_in	in	8	0	Controller memory Data in
bmp_mem_data_out	out	8	0	Controller memory Data out

5.2.2.2 Register Interface Block

The Register Interface block contains the register which controls the functionality of the controller as shown in figure 5.7.

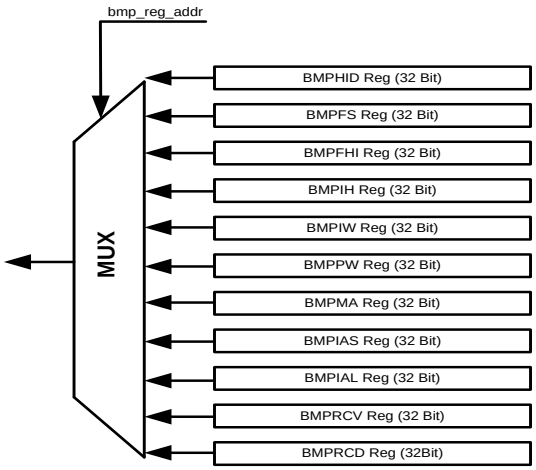


Figure 5.7: Register Interface Block Diagram

The multiplexer is controlled by the register address signal which controls the dataflow for the different registers.

Table 5.4: Register Interface Block IO Interface

Name	Dir	Width	Rst	Description
bmp_clk	in	1	0	Controller clock
bmp_reset_n	in	1	0	Controller reset
bmp_reg_re	in	1	0	Controller register read enable
bmp_reg_we	in	1	0	Controller register write enable
bmp_reg_addr	in	8	0	Controller register address
bmp_data_in	in	32	0	Controller register Data in
bmp_ca_dn	in	1	0	BMP Image Analysis complete
bmp_hid_reg	in	16	0	BMP Header File ID
bmp_fs_reg	in	32	0	BMP Image file size
bmp_fhi_reg	in	8	0	BMP Header Information Size
bmp_ih_reg	in	32	0	BMP Image height in Pixel
bmp_iw_reg	in	32	0	BMP Image width in Pixel
bmp_pw_reg	in	8	0	BMP Image Pixel Width 8Bit
bmp_red_mv	in	32	0	32 Bit Floating point sum value of pixels
bmp_red_mf	in	32	0	32 Bit Floating point divider value of pixels
bmp_data_out	out	32	0	Controller register Data out
bmp_cstart	out	1	0	Start image Analysis

BMP Header ID

BMPHID REG – 00H

Bit		Name	POR	Description
31-16 15 – 8 7- 0	r r r		0 0 0	Reserved This block of bytes is at the start of the file and is used to identify the file. A controller reads this block first to ensure that the file is actually a BMP file and that it is not damaged. The first two bytes of the BMP file format are the character 'B' then the character 'M' in 1-byte ASCII encoding.

BMP File Size

BMPFS REG – 04H

Bit		Name	POR	Description
31-0	r		0	Size of the BMP Files in Bytes

BMP File Header Info

BMPFHI REG – 08H

Bit		Name	POR	Description
31-8 7 - 0	r r		0 0	Reserved Returns the size of BMP Info header(Bytes)

BMP Image Height

BMPIH REG – 0CH

Bit		Name	POR	Description
31	r		0	Sign Bit
30-0	r		0	This register returns the Bitmap Image Height in Pixel

BMP Image Width

BMPIW REG – 10H

Bit		Name	POR	Description
31	r		0	Sign Bit
30-0	r		0	This register returns the Bitmap Image Width in Pixel

BMP Pixel Widths

BMPPW REG – 14H

Bit		Name	POR	Description
31- 16	r		0	Reserved
15 - 0	r		0	The number of bits per pixel, which is the colour depth of the image. Typical values are 1, 4, 8, 16, 24 and 32. For Controller its value should be 8.

BMP Image Mem Address

BMPMA REG – 18H

Bit		Name	POR	Description
31- 0	r/w		0	Start address of BMP file in Memory

BMP Image Analysis

BMPIAS REG – 1CH

Bit		Name	POR	Description
31- 1	r		0	Reserved
0	r/w	sanylsis	0	Start image analysis process

BMP Interrupt Register

BMPIAL REG – 20H

Bit		Name	POR	Description
31- 1	r		0	Reserved
0	r/c	ANAD	0	BMP image Analysis process complete

BMP Red Colour Value

BMPRCV REG – 24H

Bit		Name	POR	Description
31	r		0	Sign Bit
30-24	r		0	Mantissa
23 - 0	r		0	Exponent

BMP Red Colour Divider

BMPRCD REG – 28H

Bit		Name	POR	Description
31	r		0	Sign Bit
30-24	r		0	Mantissa
23 - 0	r		0	Exponent

5.2.2.3 Colour Table

The Colour Table is a block of bytes (a table) listing the colours used by the BMP image. Each pixel in an indexed colour image is described by a number of bits (1, 4, or 8) which is an index of a single colour described by this colour table. The purpose of the colour palette in indexed colour bitmaps is to inform the application about the actual colour that each of these index values corresponds to. The purpose of the Colour Table in non-indexed (non-palletized) bitmaps is to list the colours used by the bitmap for the purposes of optimization on devices with limited colour display capability and to facilitate future conversion to different pixel formats and paletization[70, 71].

The colour table occurs in the BMP image file directly after the BMP file header, the DIB header. The colours in the Colour Table are usually specified in the 4-byte per entry 8.8.8.0.8 format (in RGBAX notation) In order to save area and time we only read Red colour value and save in colour table RAM.

Table 5.5:Colour Table Block IO Interface

Name	Dir	Width	Rst	Description
bmp_clk	in	1	0	Controller clock
bmp_ram_we	in	1	0	Write enable signal to write data in Colour Table RAM
bmp_ram_addr	in	8	0	Colour Table RAM Address
bmp_ram_datain	in	8	0	True Colour Value
bmp_ram_dataout	out	8	0	True Colour Value output from Colour Table

5.2.2.4 Colour Table Block Timing Diagram

In Initialization Phase system control block reads the Red colour value from the BMP file and writes thedata in the Colour Table RAM and the timing diagram if the initialization phase is shown in figure 5.8.

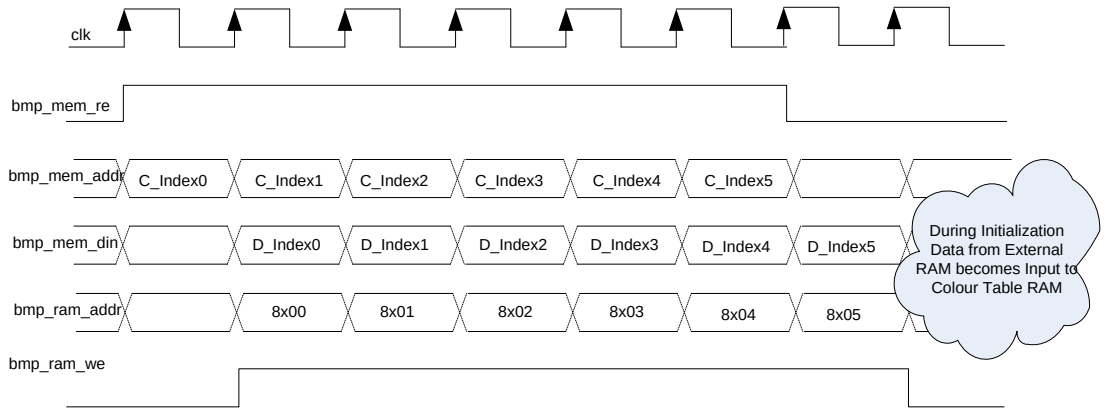


Figure 5.8:Colour Table Timing Diagram in Initialization

In Normal mode system control block reads the Pixel Value from the BMP file and pixel valuebecomes the address of the colour table in the RAM and the timing diagram of the normal phase is shown in figure 5.9.

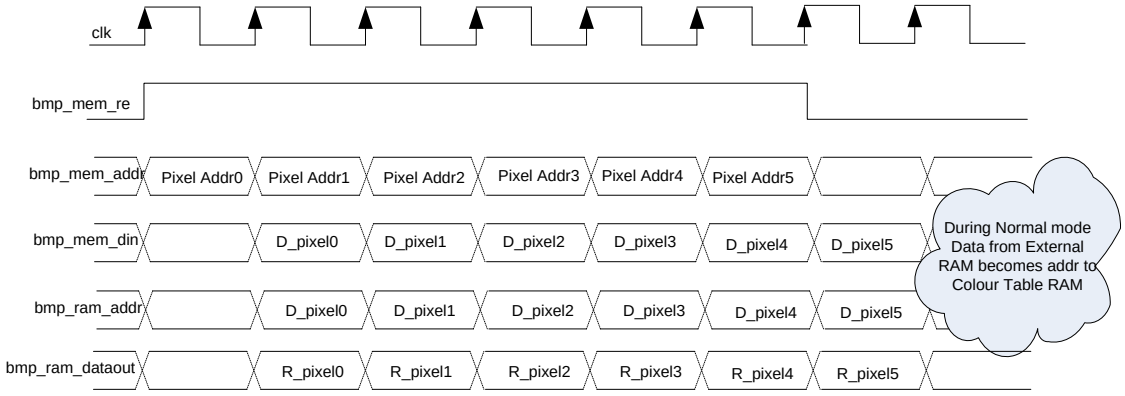


Figure 5.9:Colour Table Timing Diagram in Normal Mode

5.2.2.5 BMP Image Colour Extraction

The BMP image colour extraction block reads the actual red colour value from colour table block for an 8 bit bitmap or the red colour value from the pixel data directly for a 24 bit bitmap. The red colour values extracted from the pixels are added together and repeated for all (200 * 200) 40,000 pixels from centre of the image to generate the mean value of the colour red. Table 5.4 shows the interface for the extraction block. The block diagram for the colour extraction block is shown in figure 5.6.

Table 5.6:Colour Extraction Block IO interface

Name	Dir	Width	Rst	Description
bmp_clk	in	1	0	Controller clock
bmp_reset_n	in	1	0	Controller reset
bmp_con_flt	in	1	0	Start converting the Integer Value to Floating Point Value
bmp_strt_val	in	1	0	Start adding the Red colour Value
bmp_ca_dn	in	1	0	BMP colour analysis of image complete
bmp_ca_strt	in	1	0	Start BMP colour Analysis
bmp_ram_dataout	in	1	0	Red Colour Value from Colour Table
bmp_cflt_dn	out	1	0	Conversion to Floating point format complete
bmp_red_mv	out	32	0	Sum of 40,000 pixel red colour value in 32 Bit single Precision floating point format
bmp_red_mf	out	32	0	Dividing Factor of Red Colour Mean Value i.e. 40,000 in 32 Bit single Precision floating point format

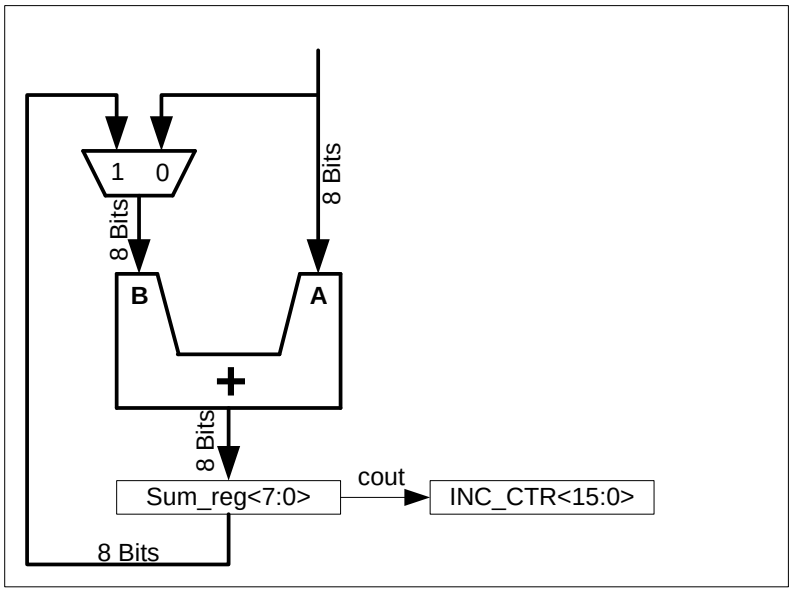


Figure 5.10: Block Diagram of Colour Extraction Block

A flow chart of the entire operation of the colour extraction block is shown in figure 5.32. The sum of all 40,000 pixels is added together and converted into IEEE 754 single precision 32Bit Format. To convert the sum to floating point the number of leading zeros in the final sum register is counted. The final sum register is a 24 bit register and the number of leading zeroes is counted from the MSB of the register which determine the exponent of the floating point number and the rest of the bits after the first bit with the value 1 form the mantissa.

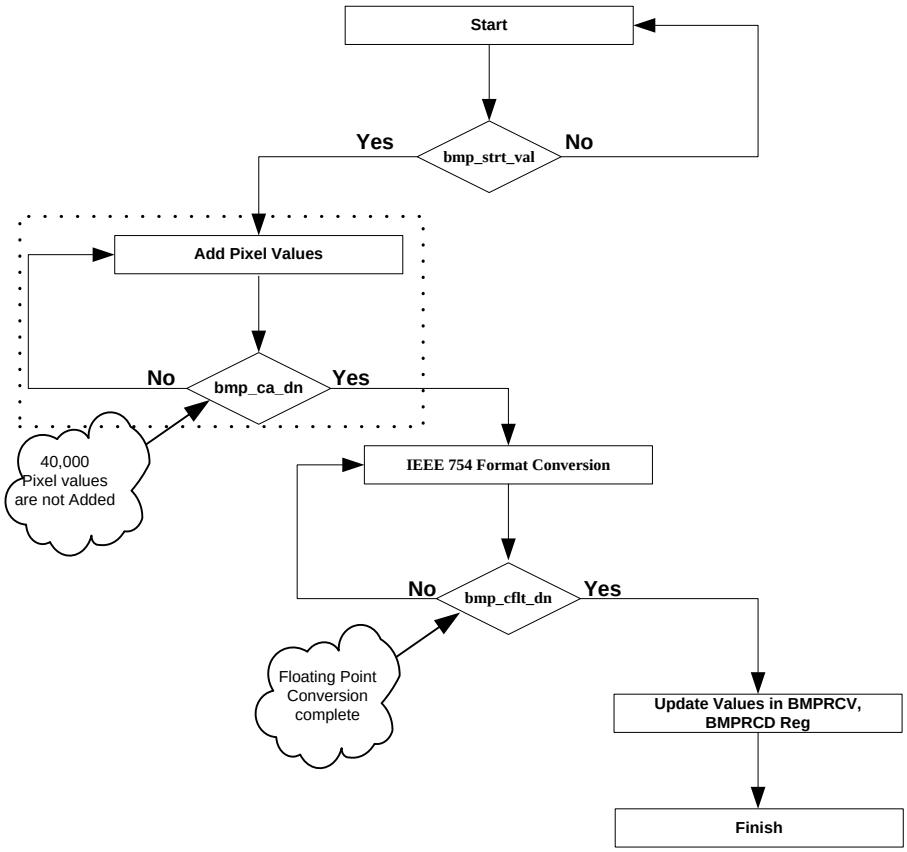


Figure 5.11:Colour Extraction Block Data Flow Diagram

5.2.2.6 System Control & Initialization Block

The System Control Block (FSM Based) controlsthe operation of the BMP image analysis controller and its sequence of operation is given below

- Read BMP image file parameters and store in register for controllability and observability
    - BMP Image Header
    - BMP File Size
    - BMP image width
    - BMP image Height
    - BMP image Pixel Size
    - BMP image Pixel data address
- Generate BMP image Pixel Address to read the Index value of each Pixel.
- Generate Control Signals for Colour Extraction Block, Colour Table Block, and Register Interface Block and address Mapping and Memory interface block.

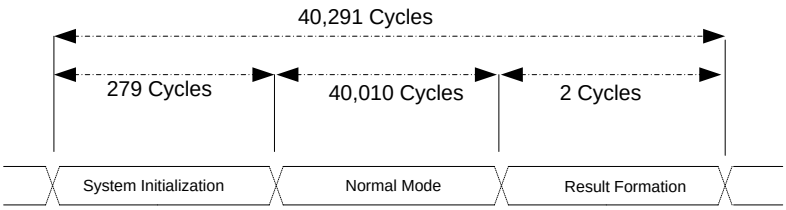


Figure 5.12: System Control Block operations/Performance

Name	Dir	Width	Rst	Description
bmp_clk	in	1	0	Controller clock
bmp_reset_n	in	1	0	Controller reset
bmp_cstart	in	1	0	Start image Analysis
bmp_mem_data_in	in	8	0	Controller memory Data in
bmp_con_flt	out	1	0	Start converting the Integer Value to Floating Point Value
bmp_strt_val	out	1	0	Start adding the Red colour Value
bmp_ca_dn	out	1	0	BMP colour analysis of image complete
bmp_mem_re	out	1	0	Controller memory read enable
bmp_mem_addr	out	32	0	Controller memory address
bmp_ca_dn	out	1	0	BMP Image Analysis complete
bmp_hid_reg	out	16	0	BMP Header File ID
bmp_fs_reg	out	32	0	BMP Image file size
bmp_fhi_reg	out	8	0	BMP Header Information Size
bmp_ih_reg	out	32	0	BMP Image height in Pixel
bmp_iw_reg	out	32	0	BMP Image width in Pixel
bmp_pw_reg	out	8	0	BMP Image Pixel Width 8Bit

Figure 5.13: System Control Block IO Interface

5.2.2.6.1 System Initialization

System control Block in system initialization phase reads the BMP File Information and save in registers and also read Colour Table values of Red colour from BMP image file and save True colour values in Colour Table Block. The flowchart of the system initialization and the initialization of the colour table is shown in figure 5.14 and 5.15.

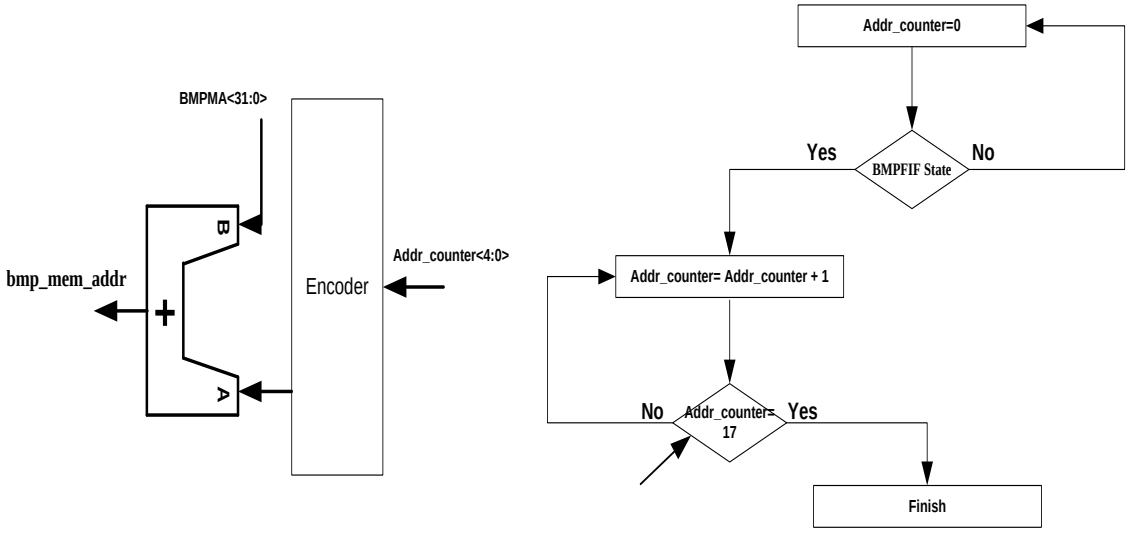


Figure 5.14: System Initialization Phase Data Flow & Logic Diagram

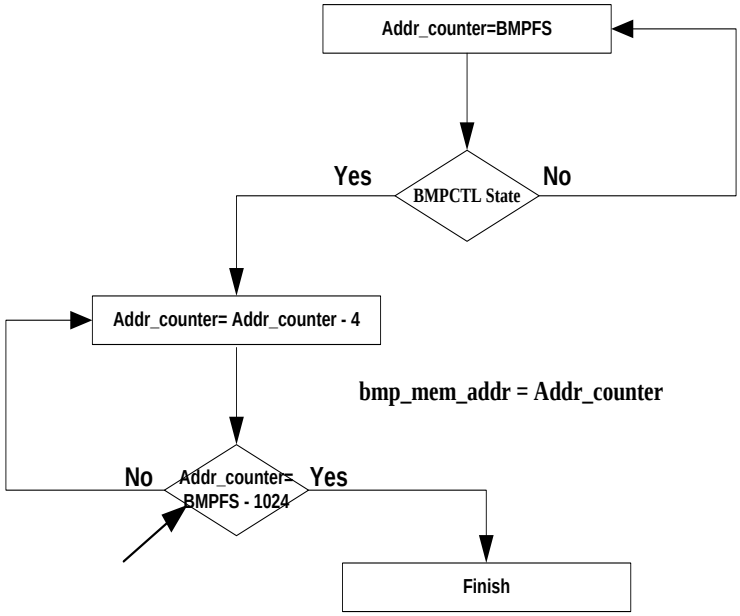


Figure 5.15: System Initialization Colour Table Data Flow

5.2.2.6.2 Normal Mode

In Normal System control block reads the pixel data from BMP image file and convert pixel index value to the true colour red value. In the normal mode the system control also generate the control signal of the other blocks.

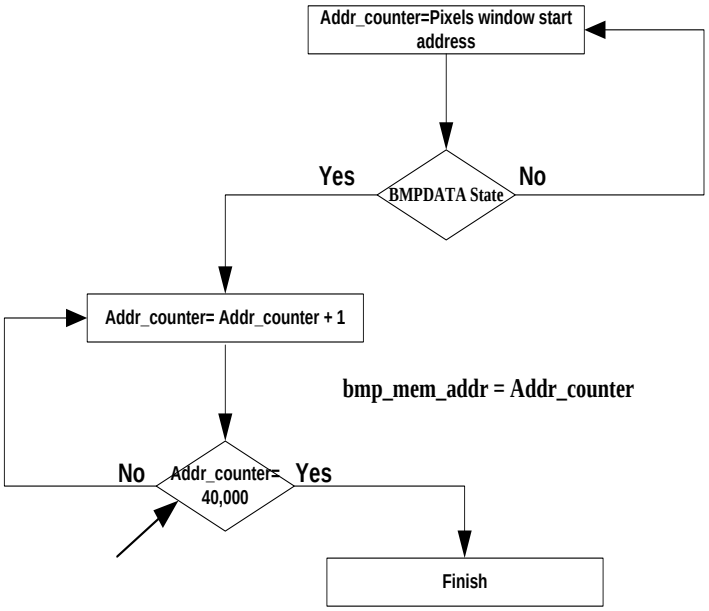


Figure 5.16: System Initialization Normal Mode Data Flow

5.2.2.6.3 Result Formation Mode

In Result Formation Mode; System control block generates the control signals to convert values to the 32 Bit Single precision Floating Point format and load values in BMPRCV, BMPRCD registers.

5.2.2.6.4 System Control & Initialization Block SM

The flowchart of the system control and intialisation state machine is shown in figure 5.17. The first state is the bmp_cstart state where the controller is intialised with the address of the image in the memory. In the next state the header information is read giving the controller the information of the size of the image and the address of the colour table. The colour table is read in the next state and is used to determine the value of the red colourcomponent from the 40,000 pixels selected from the centre of the image. In the final state is the conversion of the mean value of the colour red to the floating point format and the bmp_cflt_dn signal is asserted when the process of conversion is completed.

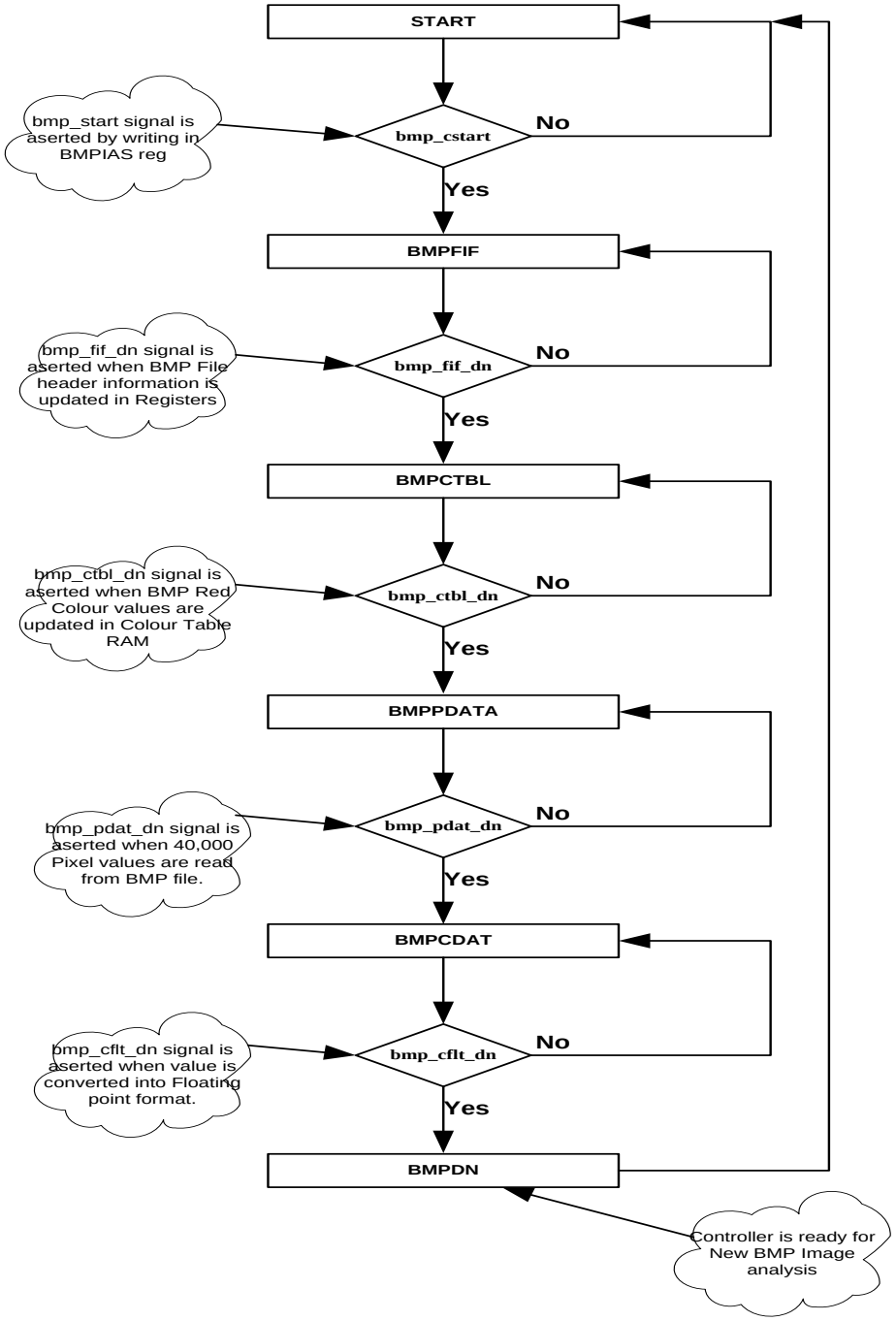


Figure 5.17: System Control Block State Machine Data Flow Diagram

5.2.2.7 Address Mapping and Memory Interface Block

The Address Mapping and Memory Interface Block generates the addresses to read the pixel value (Index) from the Bit map image file and pass this index value to Colour table block which will convert the Pixel index value to the true Red Colour value of the Pixel

which is then used in colour extraction block to calculate the mean value of The Red Colour.

Pixel address calculation expression is given below

NR = Number of Rows of Pixel in BMP Image = 1280

NC = Number of Cols of Pixel in BMP Image = 1024

BMPMA = Address of BMP Image in external memory

BMPFS = Start Address of BMP Pixel Data in BMP Image File

PL = Pixel Location (Column) in BMP image

Valid Pixel address is given by the following Relation

VP_ADD = (NR / 2 + I) * NC + PL + BMPMA + BMPFS

(NC / 2 - 100) ≤ PL ≥ (NC / 2 + 100)

-100 < I > 100

5.2.2.8 Colour Analysis Block Data Flow/PIO

The data flow diagram of the complete colour analysis block is shown in figure 5.18. The first step is the initialization of the address location in the external memory where the bmp image is stored. The next step the information of the bmp file is read to determine where the pixel data starts and the information about the colour table or palette. The colour table is read and stored in the internal colour table registers. The next step the pixel data is read for a window of 200 x 200 pixels from the centre of the image and the index values are corresponded with the colour table to read the true value of the colour red. The colour values are added to generate the sum of 40,000 pixels and converted to floating point. The values are updated in the registers passing the data to the algorithm blocks for calculation of pH and pCO_2.

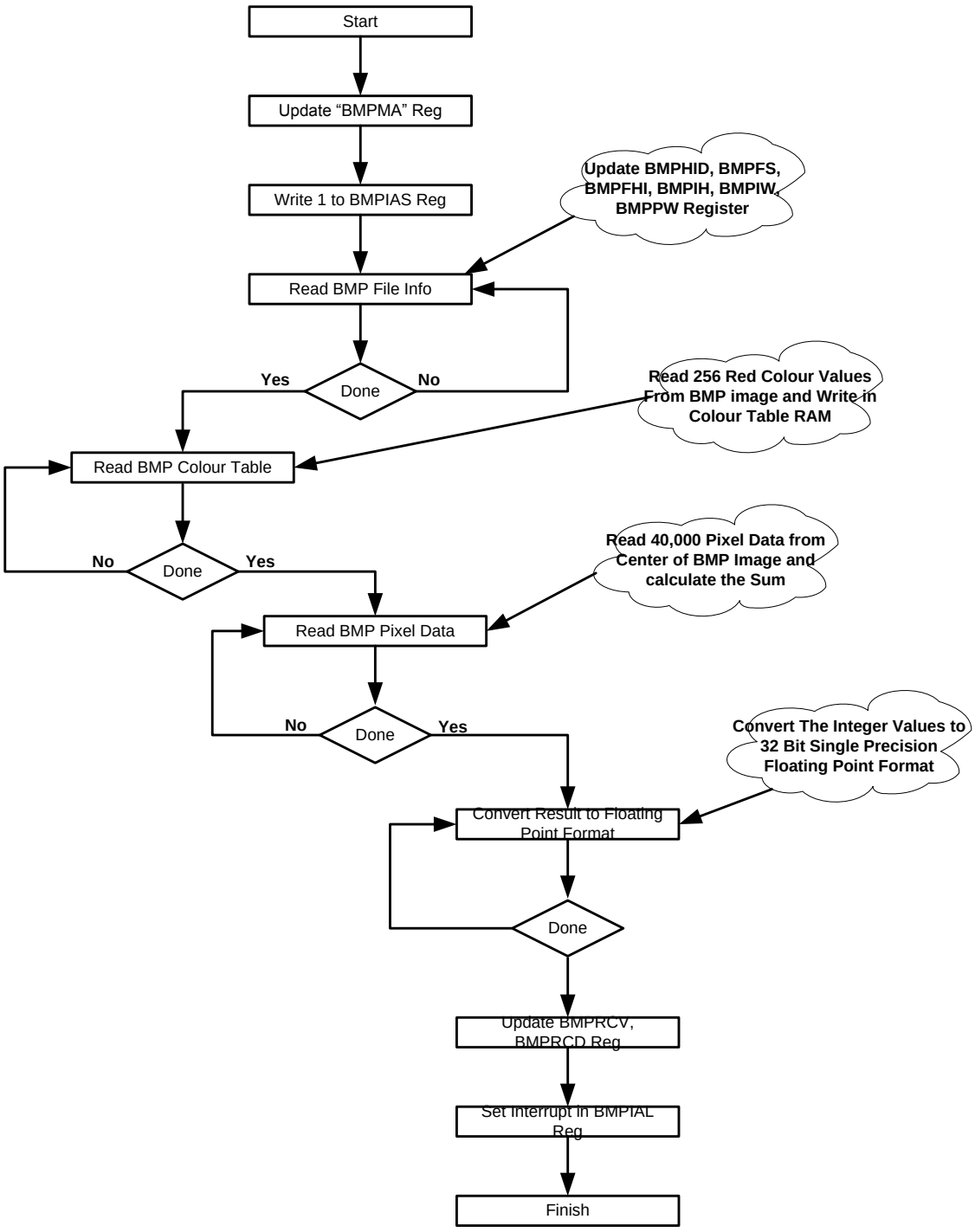


Figure 5.18: BMP Image Controller Data Flow/PIO

5.3 pH and pCO_2 algorithm blocks

In the preceding chapter the regression analysis applied on the trends of the colour red showed that a fifth order polynomial equation fitted the data the best and explained the relation between the pH and the colour red. This fifth order polynomial equation forms the basis of the design of the colorimetric algorithm calculating the pH from the colour of the indicator. The pH algorithm block is an implementation of this fifth order polynomial equation.

5.3.1 pCO_2Algorithm Block

The fluorescence and absorption based optical sensors use the pH sensor to measure the partial pressure of carbon dioxide pCO_2. A similar approach is used to measure the partial pressure of carbon dioxide in the image based colorimetric technique presented in this project. A diagram of the pCO2 measurement technique is shown in figure 5.19.

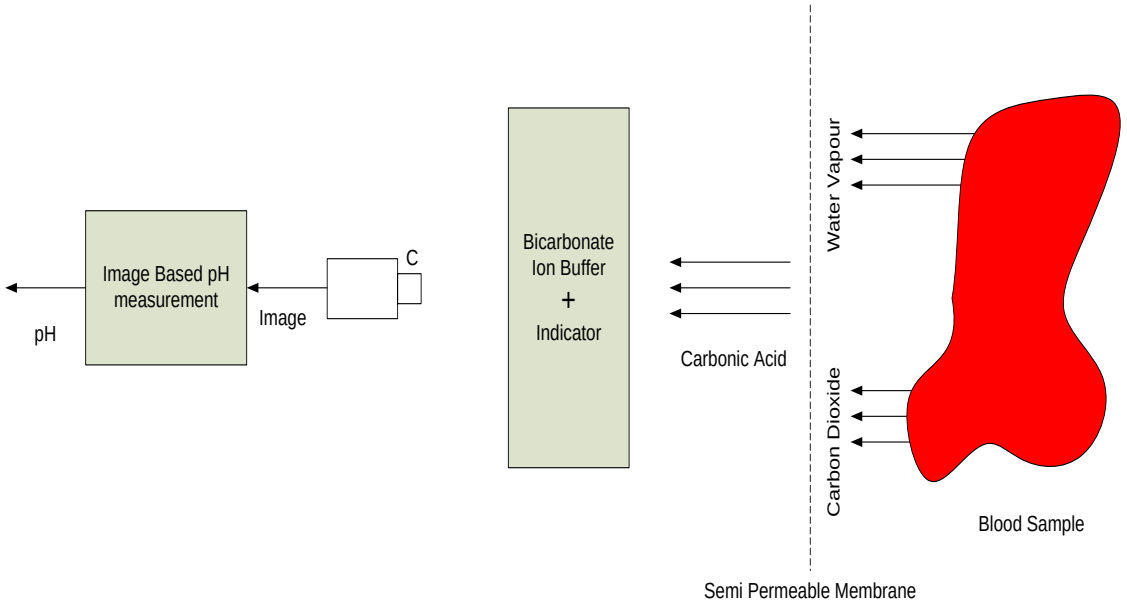


Figure 5.19: The PCO_2measurement technique using the image based pH measurement technique

The measurement technique uses a bicarbonate ion buffer of a known pH value suspended in a gas permeable membrane. The gas permeable membrane only allows carbon dioxide and water vapour to pass through [33].

Carbonic acid is formed as a result of the combination between water vapour and carbon dioxide combine and is shown in equations 5.4 to 5.6.

$$CO_2 + H_2O \rightarrow H_2CO_3 \quad 5.4$$

$$H_2CO_3 \rightarrow H^+ + HCO_3^- \quad 5.5$$

$$HCO_3^- \rightarrow H^+ + CO_3^- \quad 5.6$$

The carbonic acid alters the pH of the bicarbonate buffer and this change in pH of the bicarbonate buffer is dependent on the partial pressure of carbon dioxide. The partial pressure of carbon dioxide can be related to the measured pH through equilibrium constants and is shown in equation 5.7 [13, 21, 22, 33].

$$pH = \log N + pK_1 - \log K_s pCO_2 \quad 5.7$$

where N = concentration of the bicarbonate ion in the sensor

pK_1= negative log of the acid dissociation constant of H_2CO_3 timesthe hydration constant of CO_2 = 6.37

K_s = solubility coefficient of CO_2 = 0.57

When equation 5.7 is solved for $pCO_2$the result is shown in equation 5.8 and is implemented to form the pCO_2 algorithm block in the colorimetric analyser[33].

$$\log pCO_2 = A - pH \quad 5.8$$

where A = log N + pK_1 –log K_s

5.4 Fractional number Representation

The equation for the colorimetric algorithm for pH and the exponential equation for pCO_2 have fractional weighted coefficients. It is very important that the fractional coefficients are represented accurately as changes in the values of the coefficients will lead to errors in the calculated value. The representation of fractional data in binary requires an accurate representation of both the integer and fractional part of the number. Typically fractional numbers have been represented by fixed point or floating point representations.

5.4.1 Fixed Point Representation

The binary representation of data is a collection of m binary bits and has 2m possible combinations or states. The binary representation has no inherent meaning and it can be adapted to represent any form of data that is conceivable. The meaning of the binary representation is entirely dependent on the way it is interpreted and in general it has been interpreted as a positive integer. An example of an 8 bit binary number and its interpretation as a positive integer is shown in figure 5.20[72, 73].

1011→ 1*2^3 +0*2^2+1*2^1+1*2^0→8 + 0+ 2 + 1 = 11							
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	1	0	1	1

Figure 5.20: Binary representation and its interpretation

Thus, an m-bit binary word will correspond to an integer with a value between 0 and 2m-1 and this representation is called as an unsigned integer as it represents only positive values. To represent signed integers the two's complement representation is used and an example of an 8 bit two's complement representation is shown in the figure 5.21. The Most Significant Bit (MSB) is used to denote the sign of the number; 0 for positive and 1 for negative[72-74].

-127→ 01111111→10000000 + 1→10000001							
S	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	0	0	0	0	1

Figure 5.21: Two's Complement representation for signed integers.

Fractional numbers have two components the integer part before the decimal point and the fractional part after the decimal point. The fixed point representation of fractional numbers has a fixed number of bits for representing the integer part and the fractional part, thereby a fixed binary point and hence is called as fixed point. In figure 5.22 the fixed point representation of the fractional number 0.75 is shown; the MSB is reserved for the sign bit, the bit after the MSB is reserved for the integer and the rest of the bits after that represent the fractional part of the number. The range of the fractional number which can be covered depends on the number bits of the binary word and the number of bits allocated to the integer part. The accuracy of the fractional part is dependent on the number of bits allocated to the fractional part. The advantage of the fixed point

representation is that it can be easily implemented and the arithmetic operations used for integers can be used for fixed point fractional numbers[72, 73].

0.75→ 00.110000							
S	2^0 .	1/2^1	1/2^2	1/2^3	1/2^4	1/2^5	1/2^6
0	0 .	1	1	0	0	0	0

Figure 5.22: Fixed point representation of a fractional number.

The disadvantage with fixed point representation is that it has a limited range and it is susceptible to computational inaccuracies. The floating point representation has a broader range from 3.4 x 10^{38} to -3.4 x10^{-38}[75, 76].

5.4.2 Floating Point Representation

The floating point representation represents a fractional similar to the scientific notation with the only difference being the numbers are binary instead of a base 10 as shown below in equation 5.9; s represents the sign bit, f is the fractional part and e is the exponent.

$$x = (-1)^{s}1.fx2^{e} \quad 5.9$$

The IEEE 754 -2008 is the standard for the floating point representation and the single precision format is shown in figure 5.23. The single precision format is 32 bits long and can be broken into three parts a) the sign bit, b) the exponent 8 bits and c) the mantissa 23 bits. The sign bit is 0 if the number is positive and 1 if it is negative similar to the two's complement representation [72, 74-77].

Sign	Exponent		Mantissa	
32	31	23	22	0

Figure 5.23: IEEE 754-2008 single precision floating point representation.

The exponent has a range from -126 to 127 but is not represented in the typical two's complement representation. The value of the exponent is calculated by adding the bias value 127 to the actual exponent, thus allowing the representation of both positive and negative exponents without the two's complement format. The mantissa represents the fractional number. It is composed of an implicit leading bit and the fraction bits. The implicit leading bit has a value of one, thus the complete mantissa would be 1.f where f is

the fraction bits. The values stored in the mantissa are a normalised binary representation of the decimal number. An example of the floating point representation is shown in figure 5.24[72, 74-77].

0.425 → 00111110110110011001100110011010		
Sign	Exponent	Mantissa
0	01111101 = 125 -127 = -2	10110011001100110011010 =1.7

Figure 5.24: Example of a fractional number in the floating point representation

The floating point representation has a broader range and is more precise in representing fractional numbers. Also they are less susceptible to the inaccuracies created by rounding off during computation. The floating point representation allows accurate representation of the fractional weighted coefficients of the polynomial equation.

5.5 Implementation of the Colorimetric Equations

The colorimetric algorithms are mathematical equations defining the relation between the change in colour of the indicator and the pH value of the sample. The blocks operate using floating point numbers and the arithmetic blocks need to have an architecture that can handle floating point numbers and operations. The two key arithmetic operations performedare the floating point multiplication and the floating point addition.

The colorimetric algorithm is aimed at providing a portable solution for blood gas analysis; keeping this in mind it is important that the power consumption of the device is low so as to run on a battery power source. The architectures of the designs for the floating point multiplier and adder were based on this primary thought of low power consumption.

The arithmetic operations on a floating point number are different from the arithmetic operations on a fixed point number or an integer representation in binary. The arithmetic operation has to take into consideration both the fractional part and the exponential part of the floating point number and the mantissa and exponent need to be adjusted to the normalised form in which the number is stored.

5.5.1 Floating Point Addition

The adding operation of floating point numbers is one of the most complex operations in floating point arithmetic. The addition of two floating point numbers requires the exponents of both the numbers to be the same. The example in figure 5.7 shows the addition of two numbers x and y represented in the floating point format. The number y is shifted to the right by one place so the exponent of x and y are the same. Now both the number are added to generate the result z, but the result is not in the normalised form in which the result will be stored in the floating point format and hence the number is shifted to the right again by one bit and the exponent is increased by one and this generates the number shown below in the proper floating point format as shown in figure 5.25[72, 74-76].

$x = 1.5 = 01.10000000000000000000000 \times 2^{0}$ $y = 0.5 = 01.00000000000000000000000 \times 2^{-1}$

$$01.10000000000000000000000 \times 2^{0}$$
$$+\ 01.00000000000000000000000 \times 2^{-1}$$

$$01.10000000000000000000000 \times 2^{0}$$
$$+\ 00.10000000000000000000000 \times 2^{0} \text{ shift y right 1 bit}$$
$$z=2= 010.00000000000000000000000 \times 2^{0} \text{ result after addition}$$

$$z=2=01.\ 00000000000000000000000 \times 2^{1} \text{ shift right to normalise result}$$

Figure 5.25: Example of a floating point addition procedure

The steps mentioned above in the example form the basis of the standard algorithm for floating point addition. The steps of the standard algorithm are as follows

a) Compare the exponents of both the input numbers and determine which of them has a greater value. The exponent of the result is assigned the greater exponent value.
b) The difference between the two exponents is calculated, this is determine how many places the mantissa of the input number with the lower exponent value has to be shifted.
c) Shift the mantissa of the input number with the lower exponent value to the right for the bit positions equal to the difference value to align both the mantissas
d) Add the two mantissas and the result is the mantissa of the final result.

e) The mantissa of the result is checked for special cases like if the mantissa of the result is 0 then the exponent value of the result is set to -128 and goes to step h.
f) If the mantissa of the result overflows the mantissas is shifted right by one bit position and increases the exponent by one and go to step h.
g) If the addition result has leading zeros in the mantissa as shown in the example below, z=00.00000011100000000001110, then the mantissa is shifted to left by the number of zero bits leading the first bit with the value 1. The exponent is also decreased by the same number of leading zero bits.
h) If the exponent of the result underflows then the result is set to 0 i.e. the exponent of the result is set to -128 and the mantissa is set to 0. If the exponent overflows then the result is set to the most positive value if the mantissa is positive or is set to the most negative number if the mantissa is negative. No adjustment is done to the value of the result if the exponent doesn't overflow or underflow.

5.5.1.1 Leading One Predictor Algorithm

The standard algorithm is the most common implementation of the floating point adder but it has a higher degree of latency, however it has a considerably smaller size in terms of area. There are other algorithms which have focussed on reducing the latency of the adder and increasing the speed of the adder.

One of them is the Leading one Predictor (LOP) algorithm, the standard algorithm is based on finding the leading one in the result after addition to normalise the result and this is termed as the Leading One Detector (LOD). The LOD contributes to the latency as you have to wait for the addition result to detect the leading zeros in order to shift them to normalise the result. The LOP algorithm predicts the number of leading zeros in the result and this operation goes in parallel thereby reducing the latency and increases the speed of the algorithm. A block diagram showing the difference between a LOD and LOP adder is depicted in the figure 5.26[75, 78-80].

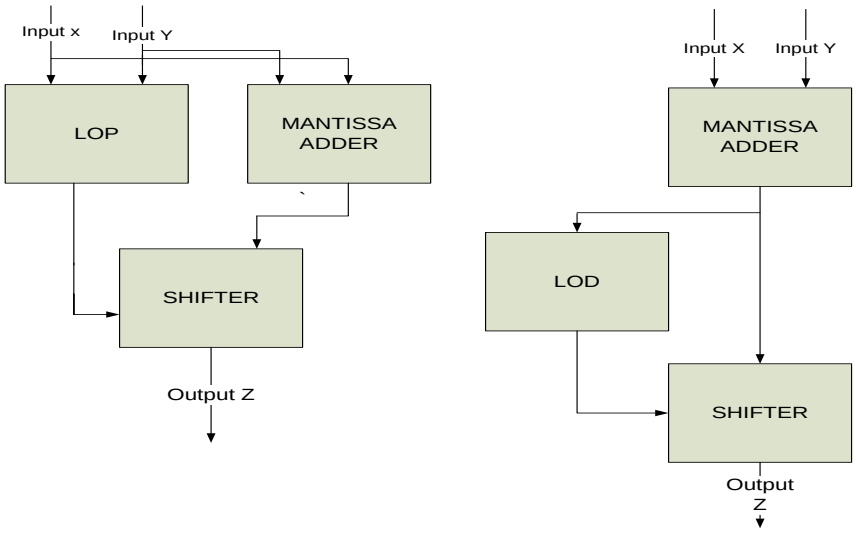


Figure 5.26: Block diagram of a LOD and LOP adder

As can be seen in the block diagram the LOD adder starts detecting the leading zeroes only after the result from the adder is available; on the other hand the LOP adder uses an algorithm to predict the number of leading zeroes from the inputs and does it in parallel along with the addition. The block diagram of the LOP algorithm is shown in figure 5.27. The LOP algorithm consists of a pre-encoding stage, a leading one detector (LOD) and an error compensation block [78, 79, 81].

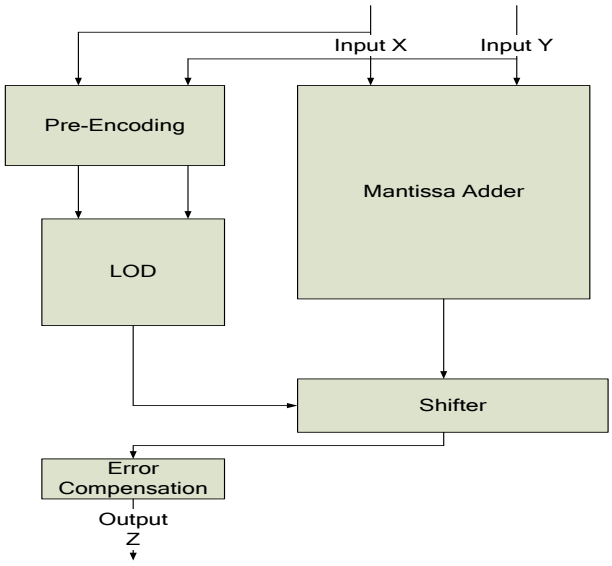


Figure 5.27: Block diagram of a LOP algorithm

The pre-encoding block analyses the inputs and produces a string of zeros and ones and the leading one has the same position in the string as it would have from the result of the addition. This string is used by the LOD block to encode the leading position to determine

the number bits the shifter has to shift the result from the addition. This encoding of the leading one can have an error of one bit and as a result of this the error needs to be corrected and hence the result of the addition after the first shifting is passed through the error compensation block to perform a one bit left shift operation[78-80, 82].

5.5.1.2 Far and Close Data path Algorithm

The far and close data path algorithm is another algorithm which looks to reduce the latency in the floating point addition and also increase the speed of operation of the design. In most cases of the floating point addition the difference between the exponents of the input numbers is either 0 or 1. The leading number of zeros only needed to be counted when the effective operation is a subtraction operation and the exponent difference is 0 or 1; the rest of the cases there is no need to count the leading number of zeros. This principle is used in the far and close data path architecture for the adder with one path with the leading one detector called the close path and one path without the leading one detector called the far path. This approach for the addition allows reducing the latency in floating point addition and also allows improvement in the speed of the adder[82].

5.5.1.3 Floating Point Adder Design

The above mentioned algorithms provide an improvement in the latency of the addition and also a marked improvement in the speed of the operation but they require extra components and thereby increase the area of the integrated circuit chip. The standard algorithm is more efficient in terms of the area required for the implementation. The latency in the standard algorithm can be improved by using pipelining. The nature of the application for this thesis doesn't require high speed floating operations and the standard algorithm works out to be the best solution for low speed applications in terms of complexity, high accuracy, area and power consumption[30, 78, 80]. Chang's adaption of the standard algorithm was chosen for implementing the floating point adder. A flow chart of the design for the floating point adder based on the standard algorithm is shown in figure 5.28[83].

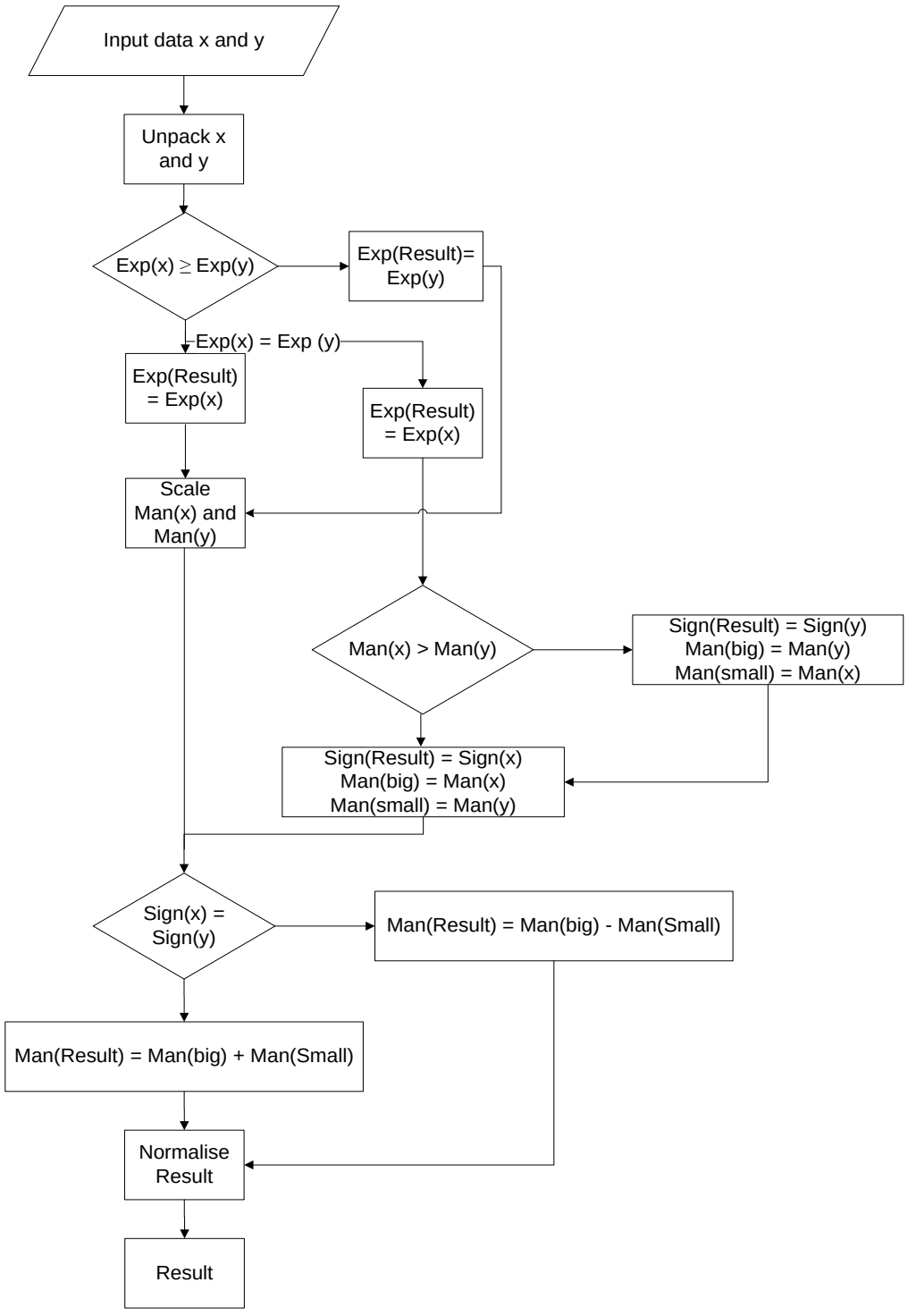


Figure 5.28: Flow chart for the floating point adder design

The first step to the addition of two floating point numbers is to unpack the two numbers. The unpacking procedure separates the sign bit, the exponent and the mantissa from the format in which the number is represented. The mantissa is normalised when stored in the IEEE 754 format and only 23 bits without the one leading the decimal point and during the

unpacking the mantissa is stored in a 24bit format with the leading one included. The unpacking step is shown in figure 5.29[72, 75].

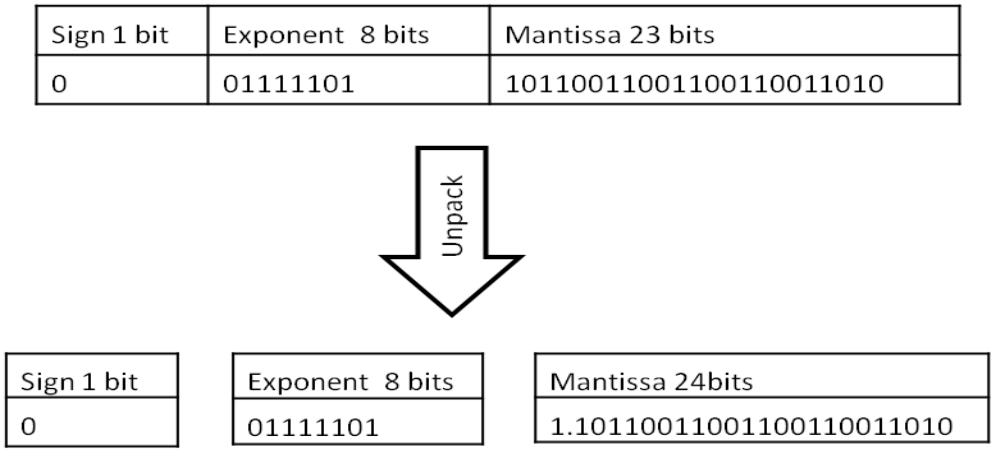


Figure 5.29: Unpacking of the IEEE 754 floating point number

The next step after the unpacking of the numbers is the comparison of the exponent values to determine which exponent has the higher value and a comparator unit does the job and a block diagram of the unit is shown in figure 5.30. The exponent of the two input numbers are compared and if exponent of x is greater than or equal to the exponent of y the exponent of the result is given the value of the exponent of x. If exponent of x is lesser than the exponent of y then the exponent of the result is given the value of the exponent of y.

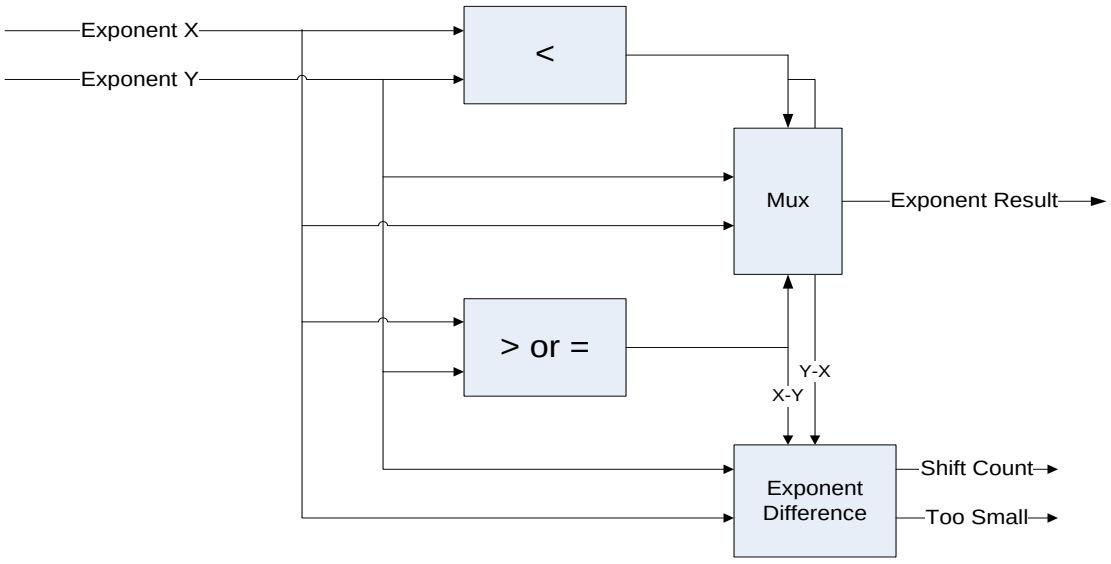


Figure 5.30: The exponent comparison block

The difference between the exponents is calculated to determine the number of bits the mantissa of the number with the lower exponent needs to be shifted. It is also determined if the difference between the two exponents is too large; in our case it is 23 bit as we are using the single precision format of the IEEE 754 format; then the number with the smaller exponent becomes insignificant as all the bits in the mantissa will be shifted out.

The next step in the addition is the scaling of the mantissa; this involves shifting the mantissa of the number with the smaller exponent to the right for the same number of bits as the difference in the exponents. The mantissa of the number with the bigger exponent is stored as the bigger mantissa and the other mantissa as the smaller mantissa. If the difference in the exponents is large then the mantissa of the smaller exponent number is set to zero.

If the exponents of both the numbers are same then the mantissas of both the numbers are compared to determine the bigger mantissa and the smaller mantissa but no shifting of the smaller mantissa is done as the exponents are equal. A block diagram of this step is shown in figure 5.31.

Once the big and small mantissas are determined and the small mantissa is shifted according to the shift count provided by the exponent difference the mantissas are now ready to be added. The addition process starts with the comparison of the sign bits of the two numbers and the determination of the sign of the result. The sign of the result is always the sign of the number with the bigger exponent or in the case of the exponents being equal the sign of the number with the bigger mantissa is given to the sign of the result.

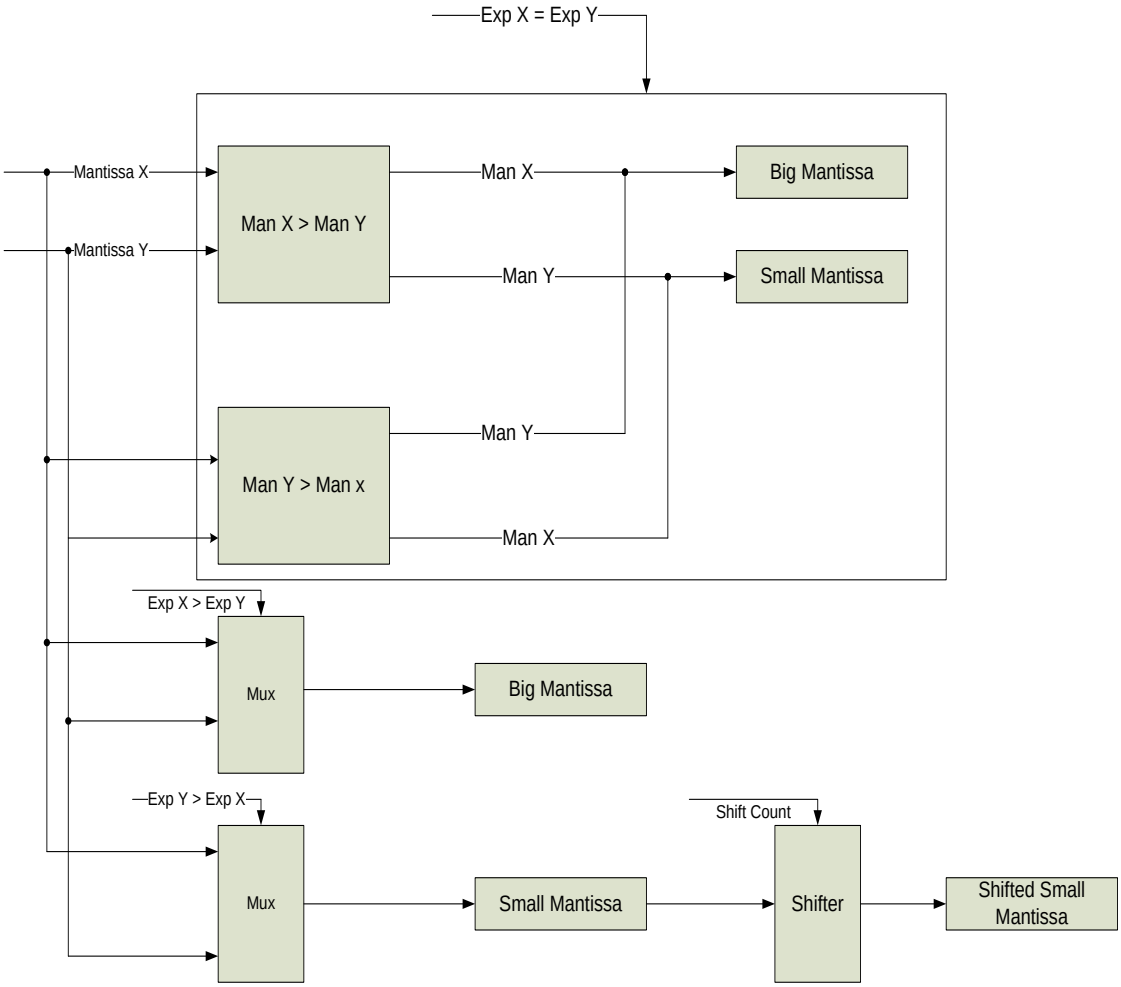


Figure 5.31: Scaling of the mantissa and the determination of the big and small mantissa

The sign bits of the input numbers determine the operation of addition or subtraction. If the sign bits of the two numbers are equal then the two mantissas are proceeded to be added and if the sign bits don't match then the smaller mantissa is subtracted from the bigger mantissa. The actual addition of the two mantissas is in itself similar to the addition of two fixed point numbers. As explained earlier the application in this thesis is not of a very high speed nature the simple adder architecture was used to implement the mantissa adder. During the addition of the mantissas an extra bit is added in front of the most significant bit of the mantissa. This bit is added to account for the overflow due to carry during an addition operation. A block diagram of this unit is shown in figure 5.32.

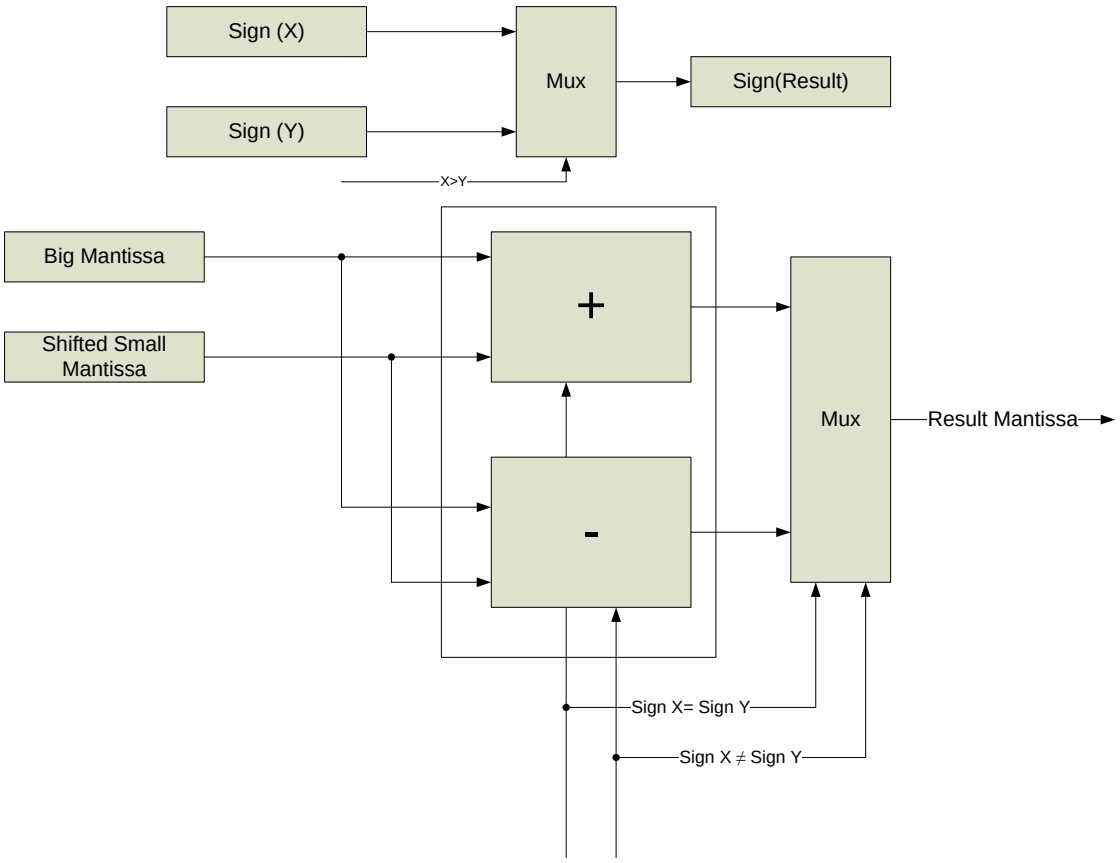


Figure 5.32: Mantissa addition and the determination of the sign for the result.

The final step after the addition result is determined is the normalisation of the result and packing the result in the IEEE 754 format. As explained earlier the result generated from the addition operation is 25 bits long as an extra bit is included for the carry overflow. The result in the IEEE format needs to be adjusted back to the 23 bit format with the implied 1 bit in the front; this also results in the exponent of the result being adjusted.

The adjustment of the result starts with the leading one detector. The mantissa from the addition is passed through leading one detector. A block diagram of the normalising block is shown in figure 5.33.

The leading one detector starts from the leftmost bit of the mantissa and checks if it's a one, if so the mantissa is shifted to the right by one bit and the exponent of the result is incremented by one. If the bit is not a one the leading one detector moves on to the next bit and increments a counter by one and checks to see if its value is a one; this action is continued till the leading one detector finds the first bit with a value of one. The counter value is the number of bits the mantissa needs to be shifted left to get the mantissa in the normalised format. The number bits that have been shifted left are subtracted from the

exponent value of the result to adjust it for the shifting. Once the normalising is performed the first two bits of the mantissa are excluded and the next 23 bits are passed on along with the adjusted value of the exponent and the sign bit to be packed together to represent the result from the addition of the two floating point numbers.

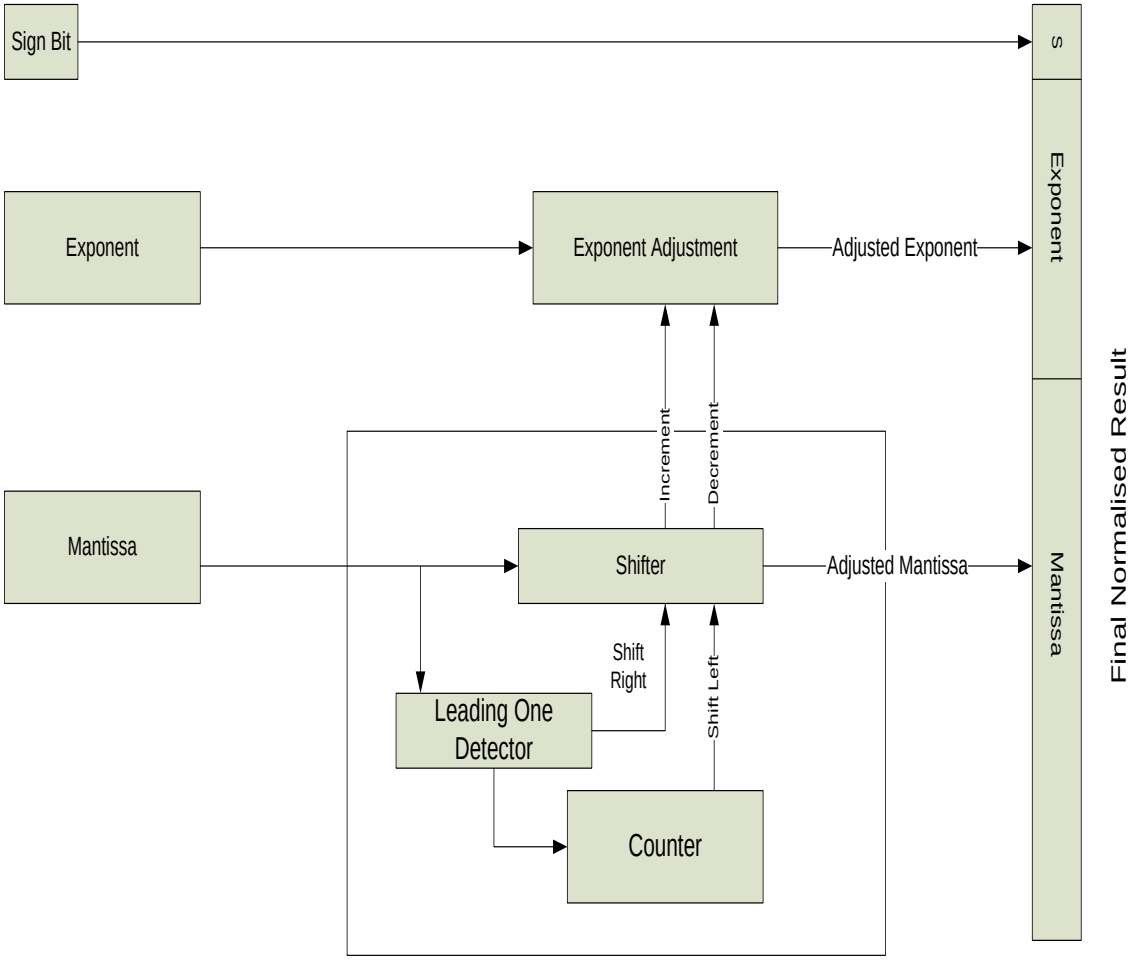


Figure 5.33: Result Normalisation block

The floating point adder was implemented in Very High Speed Integrated Circuit Hardware Descriptive Language (VHDL) and simulated inModelsim. The design has a latency of seven clock cycles. The simulation results for the floating point adder are shown in figure 5.34. A schematic of the implementation is shown in figure 5.35.

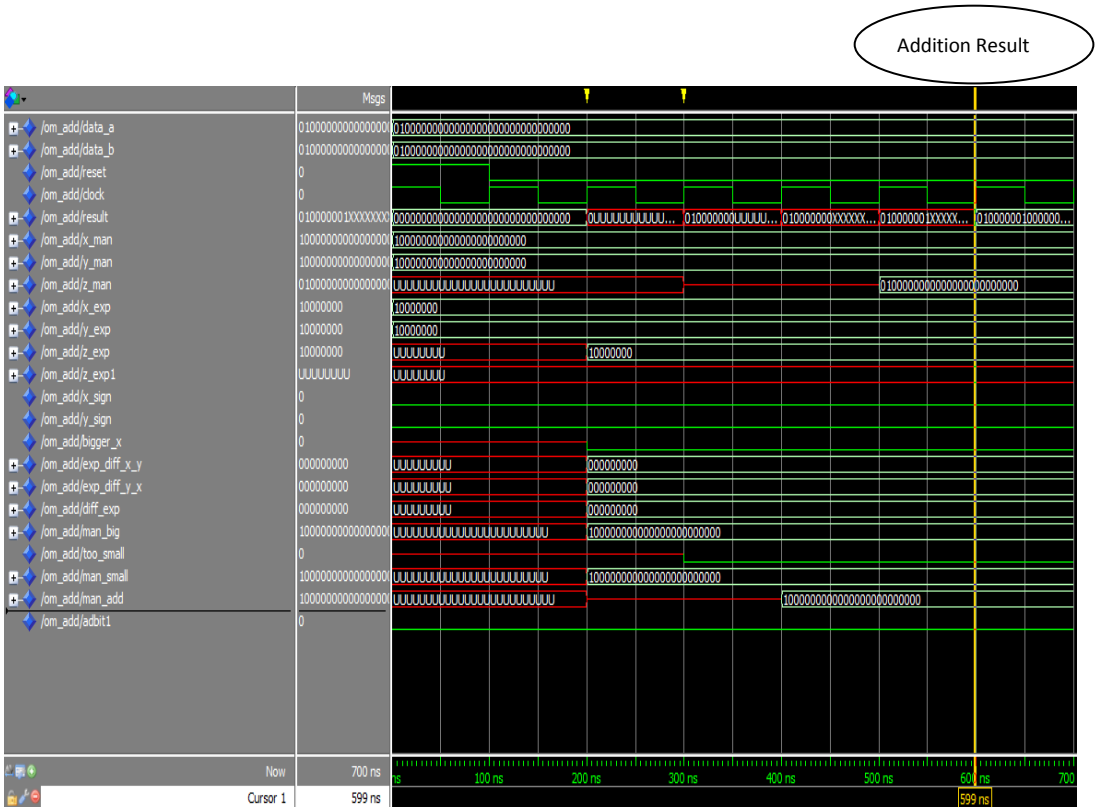


Figure 5.34: Simulation results for the floating point adder.

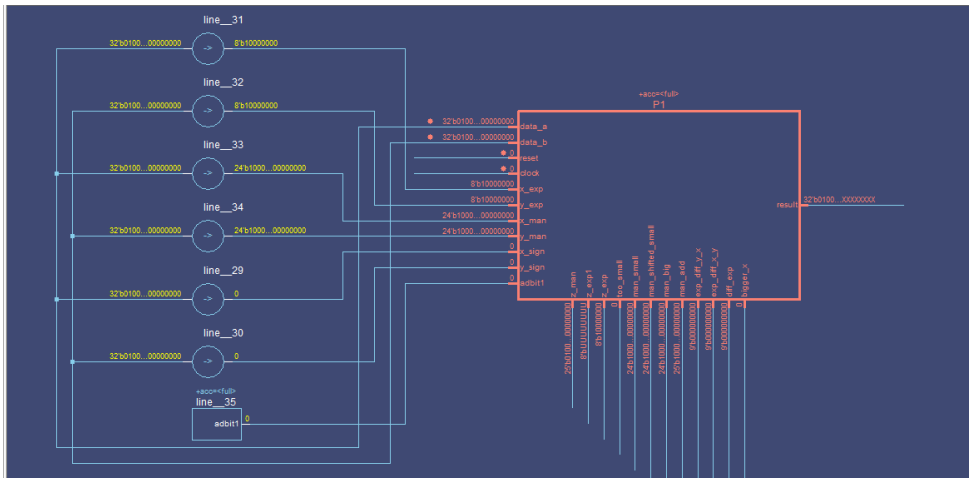


Figure 5.35: Dataflow Diagram for the floating point adder.

5.5.2 Floating Point Multiplication

The floating point multiplication operation is a simpler operation when compared with the addition operation. The value of a floating point number *x* can be represented as

x(man) x $2^{x(exp)}$ and the multiplication of two floating point numbers can be represented as follow *x(man) *y(man)* x $2^{(x(exp)+y(exp))}$.[72, 74-76].

The steps in the standard algorithm for floating point multiplication are as follows

a) Multiply the input mantissas to generate the mantissa for the result.
b) Add the exponents of the inputs to generate the exponent for the result.
c) To determine the sign of the result perform an exclusive or operation of the sign bits of the input numbers.
d) Normalise the mantissa and shift right by 1 or 2 bits if required and accordingly increment the exponent by 1 or 2. If the result mantissa is zero then the exponent value of the result is set to -128.
e) If the exponent of the result underflows then the result is set to 0 i.e. the exponent of the result is set to -128 and the mantissa is set to 0. If the exponent overflows then the result is set to the most positive value if the mantissa is positive or is set to the most negative number if the mantissa is negative. No adjustment is done to the value of the result if the exponent doesn't overflow or underflow.

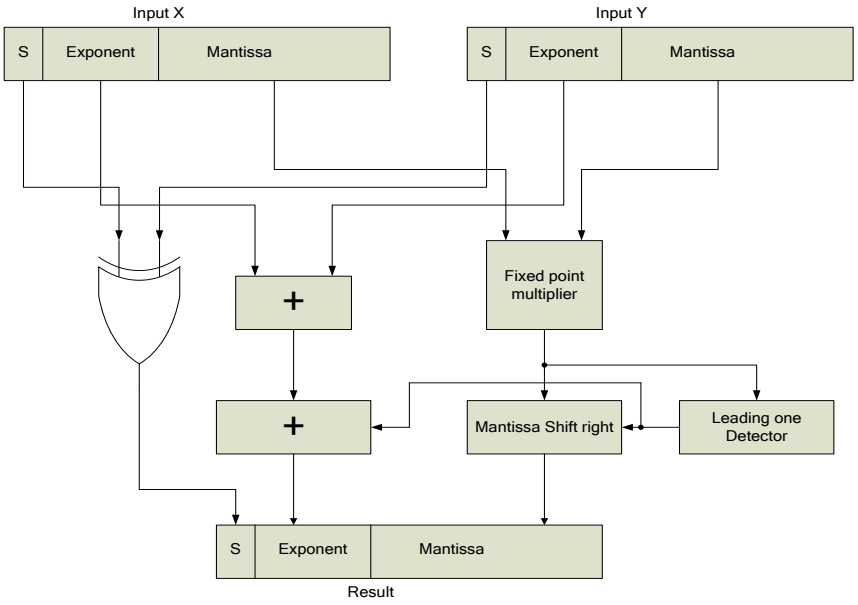


Figure 5.36: Block diagram of the floating point multiplier.

The block diagram of the floating point multiplier is shown in figure 5.36 and it is a simple architecture when compared with the floating point adder. The architecture can be divided into three parts. The first part consists of the exclusive or function of the sign bits from the inputs which determines the sign of the product thus the multiplication of two

positive numbers and two negative number will generate a positive number as the result and the multiplication of a positive number and negative number will generate a negative number as the result. The second part determines the exponent of the product by adding the exponents of the inputs and subtracting the bias in this case 127 for the single precision format from the addition result [72, 74, 76].

The third part is the mantissa of the product which is calculated by multiplying the mantissas of the two inputs. The mantissa multiplier is a fixed point multiplier and in case of the single precision floating point multiplier it is a 24 x 24 bit multiplier. The multiplier architecture determines the performance in terms of the speed of the operations and the power consumed. The commonly used algorithms for the multiplier are the Bit array multiplier algorithm, Carry Save multiplier, Wallace Tree multiplier and the Baugh Wooley multiplier[84, 85].

5.5.2.1 Bit Array Multiplier

The array multiplier originates from the multiplication parallelogram and a block diagram of a 4x4 bit array multiplier is shown in figure 5.37.

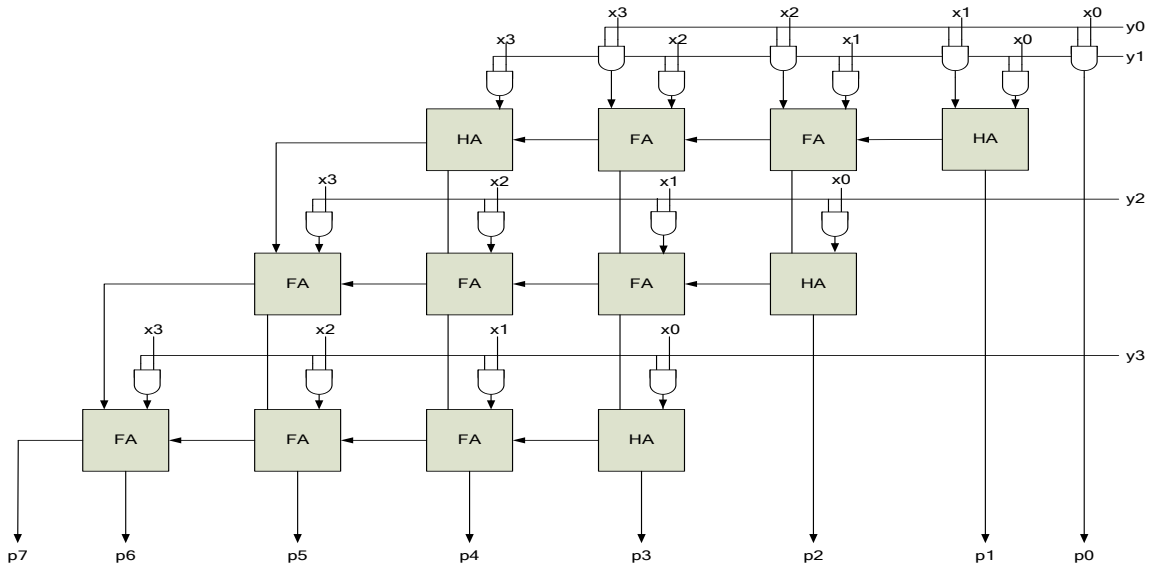


Figure 5.37: Block diagram of the 4x4 bit array multiplier.[86]

The partial products are generated by multiplying all the bits of the multiplicand with one bit of the multiplier. The partial products are shifted left according to the order of the bits from the multiplier and then added. The number of partial product terms depends on the

number of multiplier bits as the partial products for all the multiplier bits need to be generated. The AND gates in the design generate the partial products and for a X x Y bit multiplier X * Y number of AND gates are required. The design also requires (Y-1) X-bit adders to perform the addition of the partial products. The shifting of the partial products to align them correctly is done by routing and no extra logic is needed[84-89].

The bit array multiplier is very area intensive as it requires a large area for all the adders required to perform the partial product addition. The critical path in the multiplier is from the least significant bit (LSB) to the most significant bit of the product and remains the same whichever path is taken in the multiplier. The delay for the multiplier can be given as

$$\Delta T = T_{and} + T_{sum} + [(Y-1) + (X-2)]T_{carry}$$

Where $T_{and}$is the delay contributed by the AND gates used to generated the partial product. T_{sum} is the delay between the input carry and the sum bit of the full adder, *Y* is the width of the multiplicand, *X* is the width of the multiplier and the T_{carry} is the propagation delay between the input and output carry[86, 90].

5.5.2.2 Carry save Multiplier

The carry save multiplier has a structure similar to the array multiplier except for the way the carry generated by the adders is used in the circuit. The basis of the algorithm is that the multiplication result doesn't change if the output carry bits are passed diagonally downwards instead of only right as in the case of the bit array multiplier. The carry bits are not added immediately rather they are saved and added in the next addition stage hence it's called as a carry save multiplier. In the final stage the carries and the sums are merged together using a fast- carry propagate adder like a carry look ahead adder. The block diagram of the multiplier is shown in figure 5.38[84, 86, 90-92].

The use of the additional adder in the final stage increases the area slightly but the delay in the circuit is reduced. The circuit can be easily pipelined thereby increasing the throughput of the circuit. It has only one critical path as opposed to multiple critical paths in the array multiplier.

The delay for this multiplier can be given as

$$\Delta T = T_{and} + T_{final} + (X-1)T_{carry}$$

Where $T_{and}$is the delay contributed by the AND gates used to generated the partial product. $T_{final}$is the delay from the final stage adder, X is number of partial product stages, and the T_{carry} is the propagation delay between the input and output carry.[86]

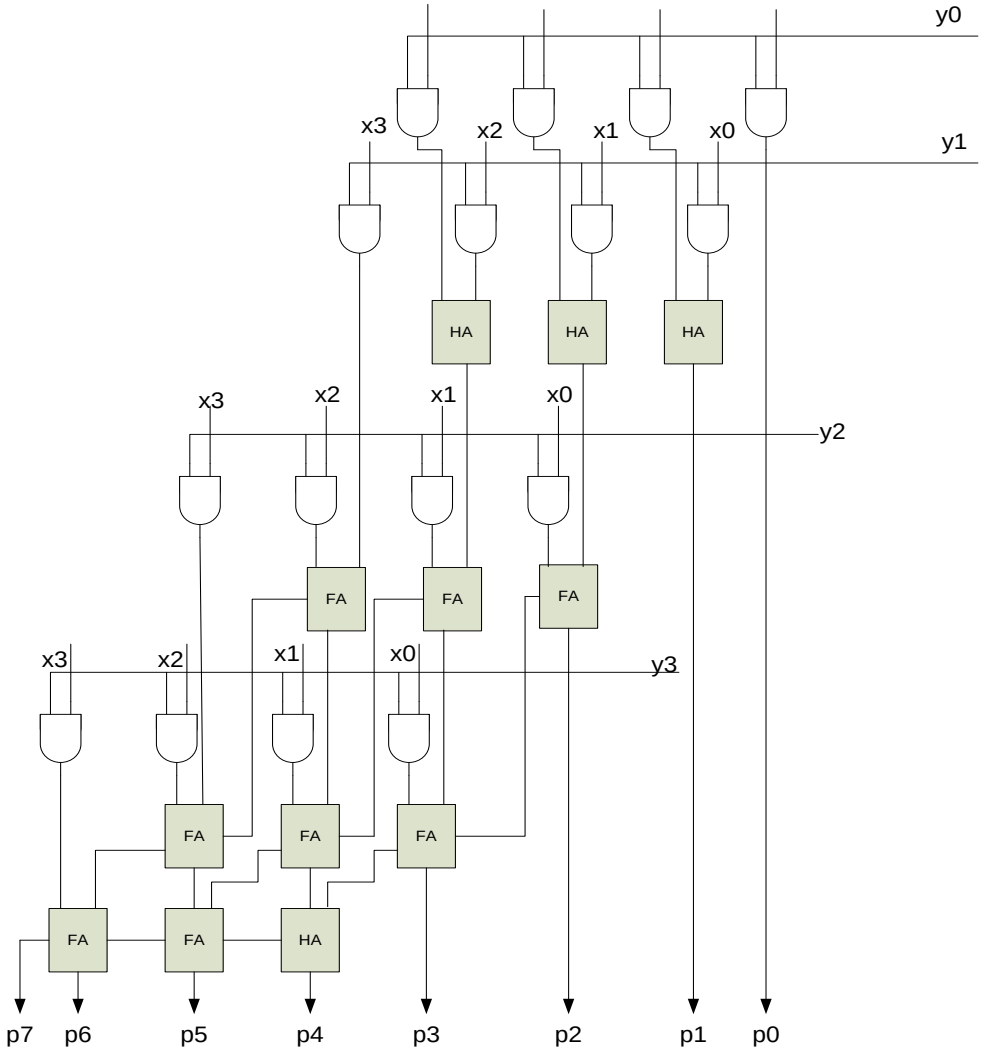


Figure 5.38: Block diagram of the 4x4 carry save array multiplier.[86]

5.5.2.3 Wallace Tree Multiplier

The Wallace tree implementation is designed for minimum propagation delay. The implementation doesn't add up the partial products completely like the array multiplier;

rather it allocates weights to the partial products and sums up all the bits of the same weight in the same tree. The steps to process the multiplication operation are

1) Formation of the bit products
2) Reduction of the bit product matrix into a two row matrix by means of a carry save adder
3) The remaining two rows are summed using a fast carry propagate adder to produce the product.

A 4x4 Wallace tree multiplier structure is shown in figure 5.39[86].

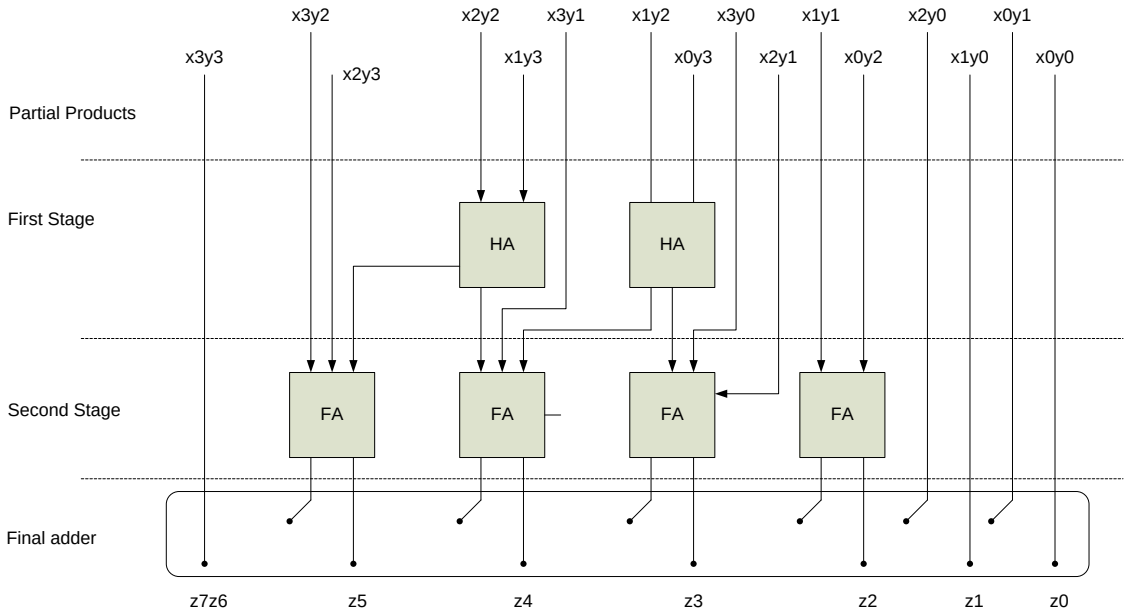


Figure 5.39: Block diagram of the 4x4 Wallace Tree multiplier.

The multiplier usually uses full adder as 3 equally weighted bits can be combined to generate two bits: the carry which has a weight of n+1 and the sum which has weight of n. Thus each layer of the tree reduces the number of vectors by a factor of 3:2. The multiplier is considerably faster than the array multiplier as its height is logarithmic in word size as opposed to linear in the array multiplier[84, 85, 88, 90-94].

The Wallace tree achieves high speeds in the multiplication function at the cost of the area which is increased considerably because of the number of adders required and complexity in the circuit connection and layout.

5.5.2.4 Baugh Wooley Multiplier

The Baugh Wooley Multiplier is primarily used for 2's complement multiplication. The partial products are adjusted to maximise the regularity of the multiplication array. The partial products with negative signs are moved to the last steps and also the design performs addition of the negation partial products rather than subtraction. A gate level diagram of a 4 bit Baugh Wooley multiplier is shown in figure 5.40[85, 86, 90].

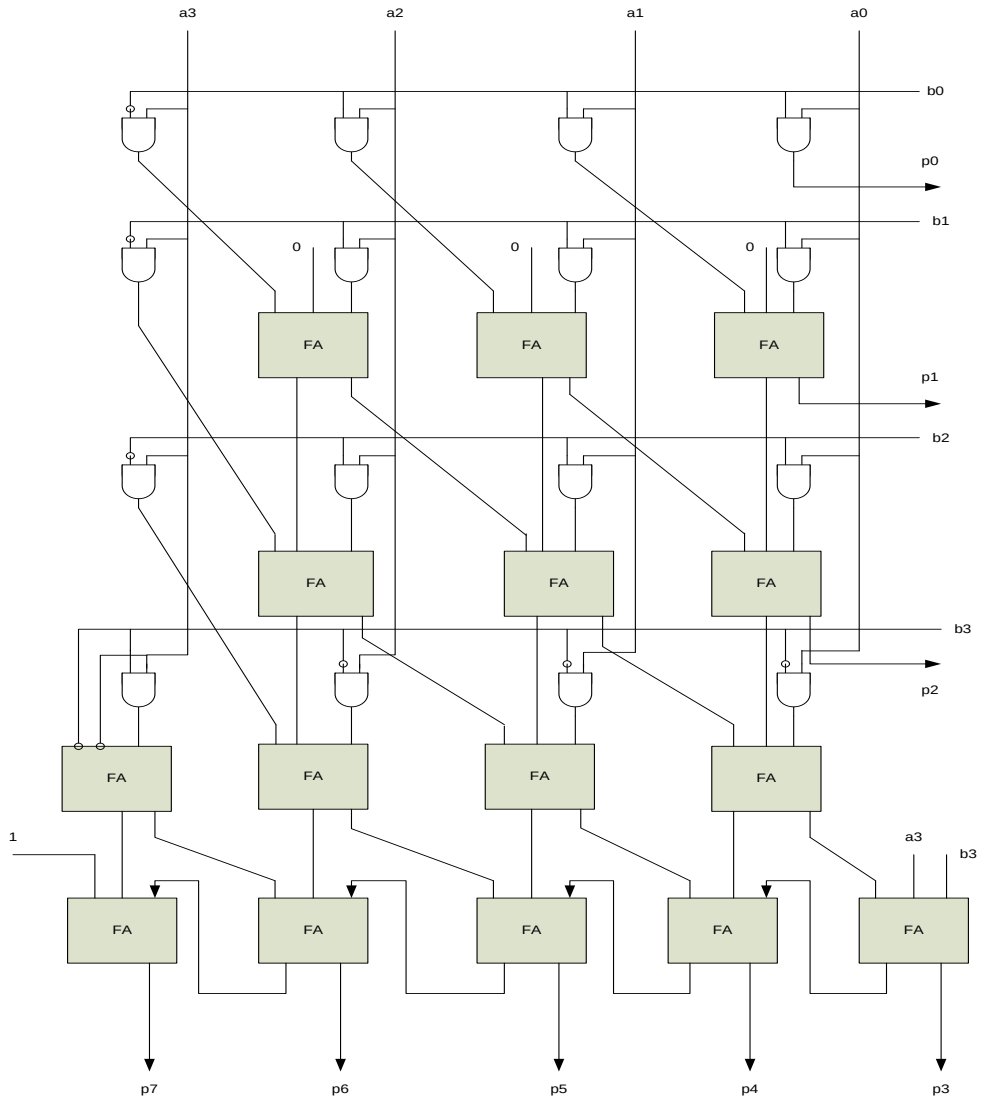


Figure 5.40: A Gate level representation of a 4x4 Baugh Wooley multiplier[95]

The equation of the Baugh Wooley algorithm for a MxM multiplier is given by equation 5.10.

$$X\ x\ Y = -2^{2M-1} + (\overline{x_{M-1}} + \overline{y_{M-1}} + x_{M-1}y_{M-1}) * 2^{2M-2} + \sum_0^{M-2}\sum_0^{M-2} x_i y_j 2^{i+j} + (x_{M-1} + y_{M-1}) * 2^{M-1} + \sum_0^{M-2} y_{M-1} * \overline{x_i} * 2^{i+M-1} + \sum_0^{M-2} x_{M-1} * \overline{y_i} * 2^{i+N-1} \quad 5.10$$

Where X and Y are M-bit operands, so their product is a 2M bits number. As a result the most significant bit is 2M-1, and the first term -2^{2M-1} is accounted for by adding a 1 in the most significant cell of the multiplier[84].

The partial products generated by AND gates and inverters are used at the inputs of specific AND gates to generate the complements needed for the algorithm. The partial products are then added together similar to the array multiplier.

The Baugh Wooley multiplier algorithm can be used to multiply both unsigned and signed numbers. The multiplier has the same delay equation as that of the array multiplier[88, 95].

The four architectures mentioned above were compared in terms of power consumption, speed and area. The array multiplier has the maximum delay and requires the highest in terms of the area. The multiplier architecture has multiple critical paths thus requiring changes to the whole architecture to affect the transmission delay. The delay in the array multiplier increases linearly with the bit width of the multiplier and a new row of adders and AND gates is required for the partial products. They also have high power consumption[85, 86].

The Wallace tree and other tree structured multipliers are designed to reduce the propagation delay in the multiplier circuit. They do so by the reducing the multiple critical paths in the array multiplier to a single critical path. They reduce the delay by reducing the number of partial products and employ a logarithmic depth reduction. The logarithmic reduction also reduces the number of adders used and reduces the power consumption of the circuit. The drawback with this implementation is the irregularity in the layout and the complicated interconnects. In comparison the array multiplier has a more regular layout and has simpler interconnects[88, 91, 92].

The Wallace tree multipliers are very suitable for high speed low power applications as they offer low propagation delay and low power consumption. The array multipliers are more suitable for low speed applications where the delay will not affect the result[85, 86, 89, 90, 94].

As the nature of the application in this thesis does not require very high speeds of operation the array multiplier was chosen for the implementation of the fixed point multiplier in the floating point multiplier. The carry save array multiplier was chosen as it can be pipelined to reduce the delay and it has only one critical path when compared with a standard array multiplier.

5.5.2.5 Floating point multiplier design

The design and implementation for the floating point multiplier can be divided in four parts the first part is the sign determination, the second part is the calculation of the exponent for the product, the third part is the fixed point multiplier and the final part is the normalisation of the product. The first step to the floating point multiplication is the unpacking of the input floating point numbers into the sign bits, exponents and the mantissas with the implied bit as explained in section 5.5.1.3 and figure 5.29.

5.5.2.5.1 Product Sign Determination

The sign determination of the product is an exclusive or operation. The sign bits of the multiplier and the multiplicand are passed to an exclusive or gate to generate the sign of the product. Thus when two positive numbers or two negative numbers are multiplied the sign of the product will be positive and when a positive number and a negative number is multiplied the result is negative[87, 96, 97].

5.5.2.5.2 Product Exponent Calculation

The exponent of the product is the sum of the exponents of the input numbers. A block diagram of the exponent calculation block is shown in figure 5.41.

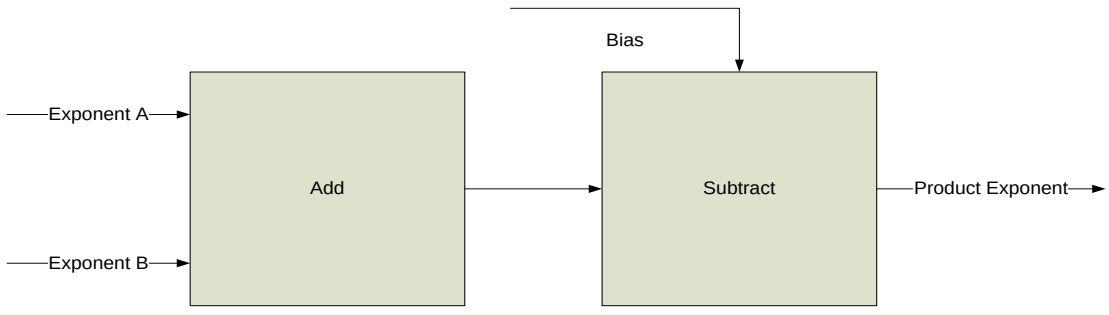


Figure 5.41: Block Diagram of the product exponent calculation

The exponents from the multiplier and the multiplicand are inputs to the addition block and are appended with an extra bit with a value of '0' to their MSB's; the extra bits are appended to take care of the overflow from the addition of the exponent values. The adder circuit is a simple 9 bit unsigned adder.

The exponents in the IEE 754 format are adjusted by bias of 127 for the single precision format, this allows the representation of negative exponent values using the unsigned format of representation and not requiring them to represent in the traditional two's complement format. In the calculation of the product exponent the bias value needs to be subtracted from the sum of the input exponent values to adjust the exponent value to the IEEE 754 format. The subtraction operation was achieved by adding the two's complement value of 127 to the sum of the input exponents[72, 93, 98, 99].

5.5.2.5.3 Fixed Point Multiplier

The third part of the implementation of the floating point multiplier is the fixed point multiplier. The carry save array multiplier architecture was chosen for the implementation of the fixed point multiplier. The carry save multiplier uses the carry save adder instead of the normal ripple carry adder used in the bit array multiplier. The carry is sent forward to the next row instead of being passed to the left in the bit array multiplier. The block diagram of the carry save multiplier is shown in figure 5.38.

5.5.2.5.4Normalisation of the Product

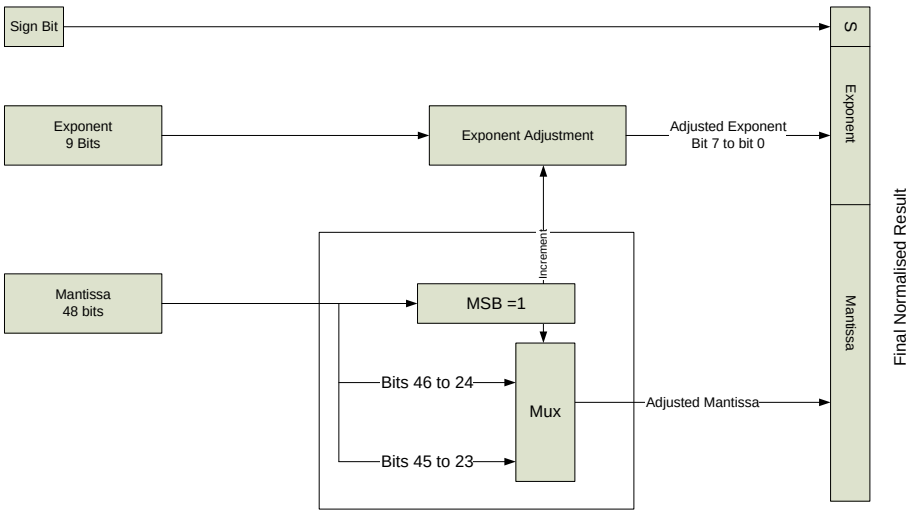


Figure 5.42: Block Diagram of the product normalization.

The mantissa of the product generated from the fixed point multiplier needs to be adjusted and normalised before packing with the exponent of the product and the sign of the product; a block diagram is shown in figure 5.42. The MSB of the mantissa of the product generated from the fixed point multiplier is checked if it is one and if so it is taken as the implied bit and the bits from numbers 46 to 24 are stored as the final product mantissa and the exponent is incremented by one. If the MSB is not one the bit next to the MSB is taken as the implied bit and the bits from number 45 to 24 are stored as the final product mantissa and the exponent value remains untouched. The final value of the mantissa, exponent and the sign are packed together to give the final product value[87, 96-98, 100, 101].The floating point multiplier was implemented in VHDL and simulated using Modelsim. The multiplier has a latency of 4 clock cycles. The simulation results are shown in figure 5.43 and a schematic of the implementation is shown in figure 5.44.

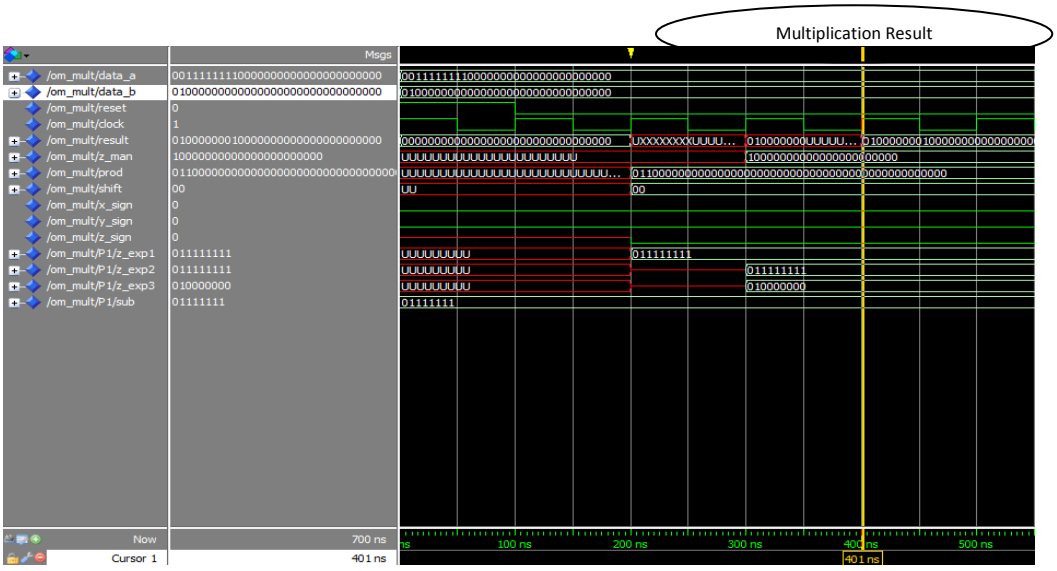


Figure 5.43: Simulation Result for the floating point multiplier.

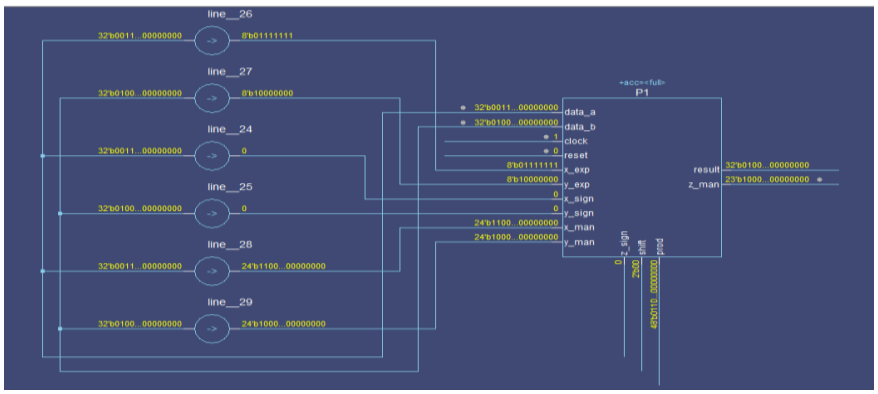


Figure 5.44: Data flow diagram for the floating point multiplier.

5.5.3 Implementation of the pH equation

The design for the pH measurement is based on the colorimetric algorithm calculating the pH from the colour of the indicator and a block diagram of its implementation is shown in figure 5.45. The implementation of the block is the implementation of the fifth order polynomial equation. The R_{mean} value is the mean value of the colour red calculated from a histogram of the image received from the colour analysis block. The higher powers of the R_{mean} value are calculated and multiplied with the coefficients of the polynomial equation and finally added to derive the pH value of the sample.

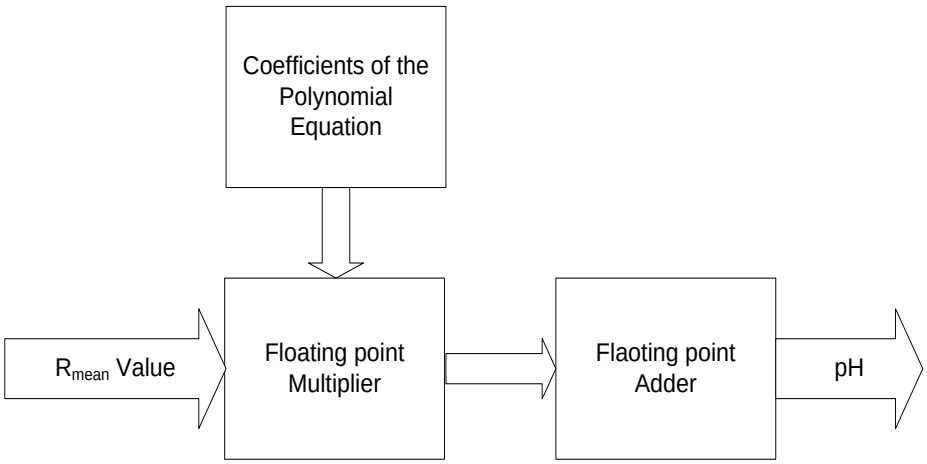


Figure 5.45: Block Diagram of the pH measurement based on the colorimetric algorithm

The pH equation is implemented in VHDL using Modelsim. Simulation results for the design are shown in figure 5.46. The pH equation is simulated for a clock with a period of 20ns. The first clock cycle is the reset state where all the registers are cleared, in the next clock edge the data from the colour analysis block is read and passed through the equation to calculate the pH. The pH calculation requires 32 clock cycles to calculate the pH value.

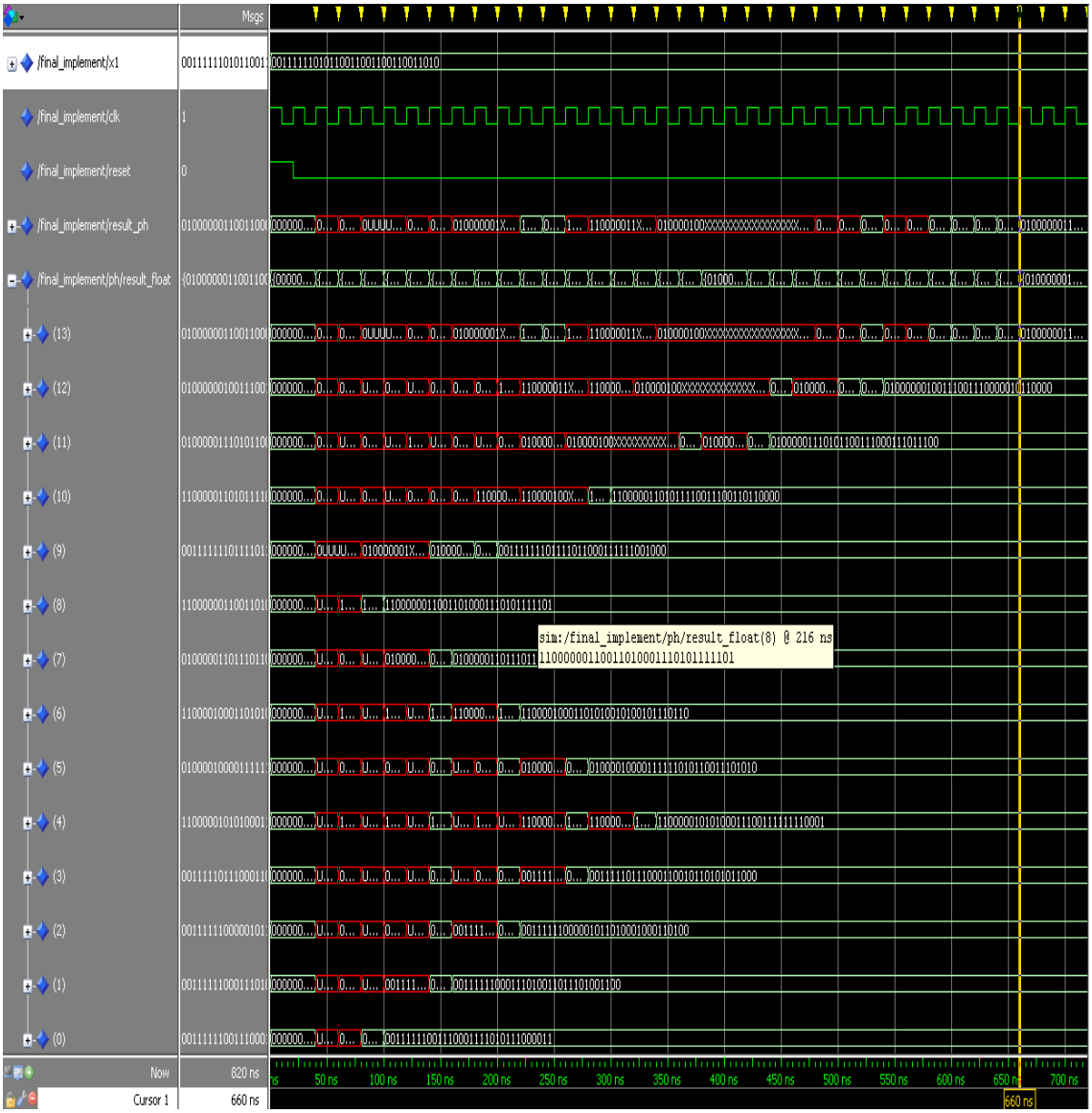


Figure 5.46: Simulation Result for the pH equation.

5.5.4 Implementation of the pCO_2equation

The equation for the partial pressure of carbon dioxide is a logarithmic expression and requires the implementation of a logarithm function more specifically the anti-logarithm function to the base 10 for floating point operations.The function can be implemented by using a look up table but the implementation requires a lot of memory. The accuracy of the operation depends on the size of the look up table[102-104]. A different approach was looked at with improved accuracy without the need for big memory. The logarithmic expression can be expressed as an exponential expression which is shown in equations 5.11 and 5.12.

$$\log_e pCO_2 \log_{10} e = A - pH \quad 5.11$$

$$pCO_2 = e^{\left(\frac{A-pH}{\log_{10} e}\right)} \quad 5.12$$

To implement the exponential function it was expressed using the Taylor Series and representation is shown in equation 5.13[105, 106].

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \quad 5.13$$

Equation 5.14 shows the relation for pCO_2 expressed as a Taylor series.

$$pCO_2 = e^{\frac{A-pH}{\log_{10} e}} = \sum_{n=0}^{\infty} \frac{\left(\frac{A-pH}{\log_{10} e}\right)^n}{n!} \quad 5.14$$

where A = log N + pK_1 –log K_s

The terms pK1 and Ks are constantsin the equation and are stored in the memory. The concentration of the bicarbonate buffer determines the range of the change in pH when the buffer comes in contact with the carbonic acid formed from the carbon dioxide and water vapour in the blood. Previous experiments conducted on this technique have shown that a buffer concentration of 0.035Molar gives a pH change between the range of 6.8 and 7.8[33].This is the range of the blood pH allowing us to use the same colorimetric algorithm to measure the pH without needing it to be recalibrated to measure the change in pH of the bicarbonate buffer.

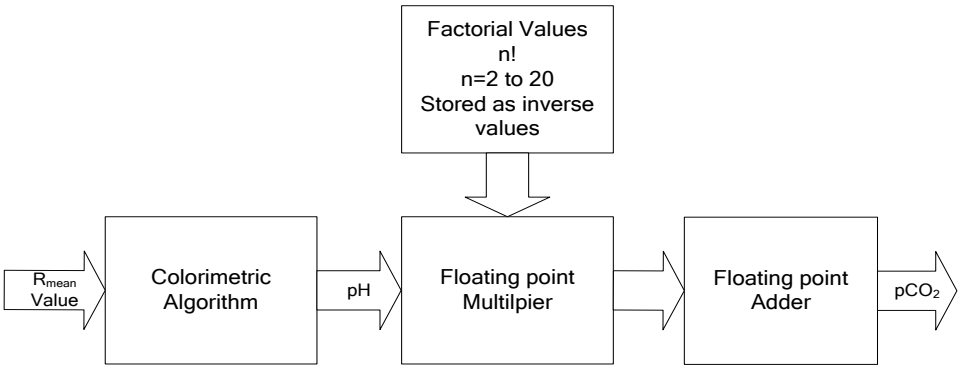


Figure 5.47: Block diagram of the pCO_2equation implementation

Figure 5.47 shows a block diagram of the implementation of the equation for pCO_2.The image of the indicator combined with the bicarbonate buffer is analysed and the mean value of the colour red is passed to the colorimetric algorithm to calculate the pH of the bicarbonate buffer. The pH value obtained from the algorithm along with the constants is

substituted in the Taylor series equation. The Taylor series was restricted to the first 20 terms as they gave the necessary resolution needed of two decimal places in the result.

Taylor series implementation uses the floating point multiplier and adder design used in the pH measurement block.There is a division operation in the implementation which was eliminated by storing the factorial term values inversed in the memory as they are constant. The inverse values of the factorial terms were multiplied to the power terms thus achieving the division function without needing to implement a floating point division algorithm. The powers of pH from n= 0 to 20 were calculated using the floating point multiplier. The inverse factorial values were multiplied with the powers of pH and then added using the floating point adder. The final result from the adder is the partial pressure of carbon dioxide.Figure 5.48 shows the simulation results for the implementation of the equation for pCO_2 in VHDL. The implementation requires 128 clock cycles to calculate the pCO_2 value including the 32 clock cycles to generate the pH value needed for the pCO_2 equation.

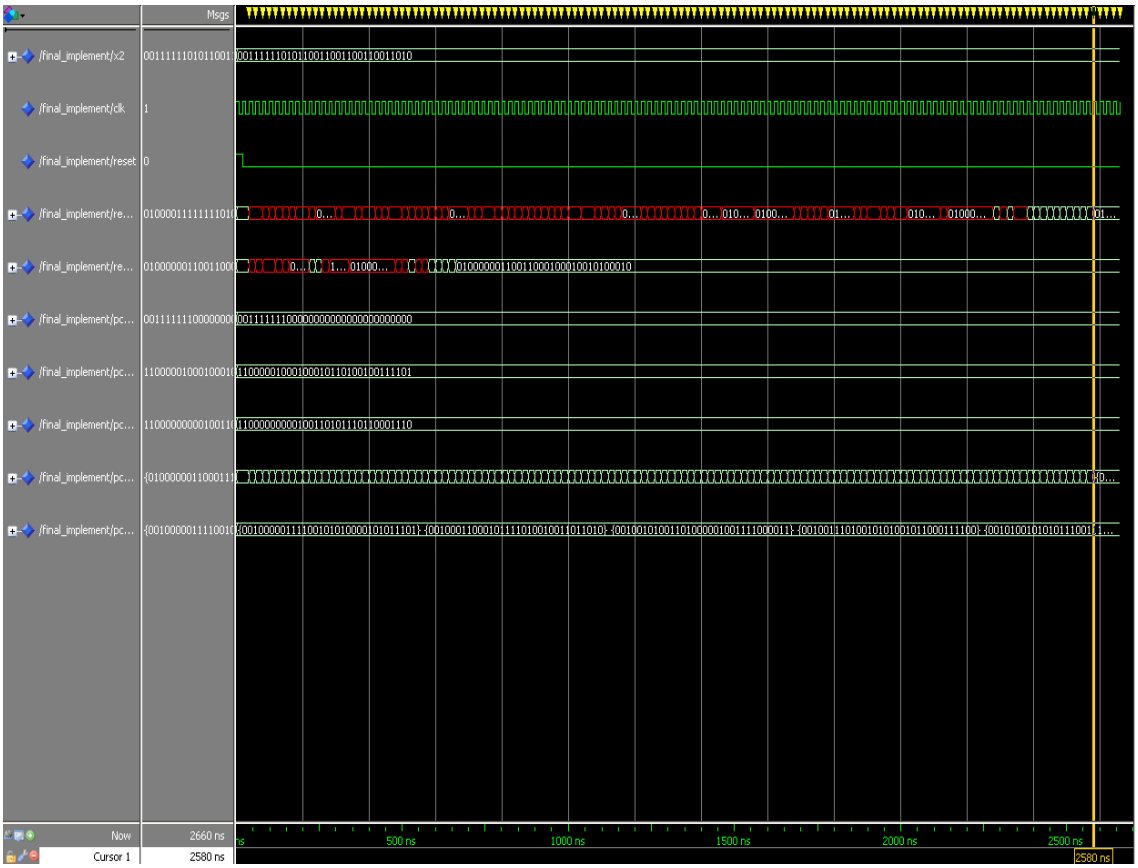


Figure 5.48: Simulation Result for the pCO_2 equation.

5.6 Conclusion

This chapter presents the design and implementation of the Image based Colorimetric pH and pCO_2analyser. The analyser has three distinct parts: the camera, the colour analysis block and the colorimetric algorithm blocks.

The colour analysis block implemented can adapt to read an 8 bit, 16 bit and 24 bit format bitmap image. The block analyses the image and reads the value of the red colour components for each pixel in a 200 x200 window from the centre of the image. The colour data from the 40,000 pixels is averaged to find the mean value of the colour red. The mean value is converted to a floating point value by using a leading zero counter and a shift register.

The colorimetric algorithms for pH and pCO_2 are mathematical equations with fractional coefficients requiring the need to use floating point architecture to implement the equations in hardware. The equations have two primary operations: addition and multiplication required to implement the mathematical equations.

The leading one predictoralgorithm and the far and close data path algorithmfor floating point addition were compared with the standard algorithm in terms of speed of operation and power consumption. The LOP and far and close algorithm provided an advantage over the standard algorithm in terms of speed of operation but required extra components to enhance the speed thus increasing the area. The standard algorithm was chosen as speed of operation is not the primary criteria rather low power consumption and lower area is. The floating adder based on the standard algorithm is implemented and has a latency of 7 clock cycles.

The floating point multiplier is implemented based on the standard algorithm with a carry save multiplier to multiply the mantissa. The carry save multiplier provides an easy to layout architecture with good performance in low speed operations. The floating point multiplier implementation has a latency of 4 clock cycles.The pH and pCO_2 equations are implemented using the floating point multipliers and the implementation have a latency of 32 and 128 clock cycles respectively. The next chapter discusses the implementation of the complete system on a 0.35 µCMOS process and Xilinx Spartan3 FPGA.

Chapter Six: Implementation Results and Analysis

6.1 Introduction

The design for the individual blocks in the Image based Colorimetric Analyser was completed in chapter 5. The next step is the integration of all the blocks to form the Image based Colorimetric Analyser. The design of the integrated block and its implementation are presented in this chapter.

This chapter is divided in two parts. Section 6.2 presents the design of the integrated block and the synthesis of the block in a Xilinx Spartan 3 FPGA and in 0.35 µ CMOS. The section also presents the results from the performance analysis of the design and implementation. Section 6.3 evaluates the performance of the Colorimetric analyser against the current analysers.

6.2 Implementation results of the Image based Colorimetric analyser

Figure 6.1a and b shows the block diagram of the complete Image based colorimetric analyser and the implementation of the image processing and colorimetric analysis blocks.

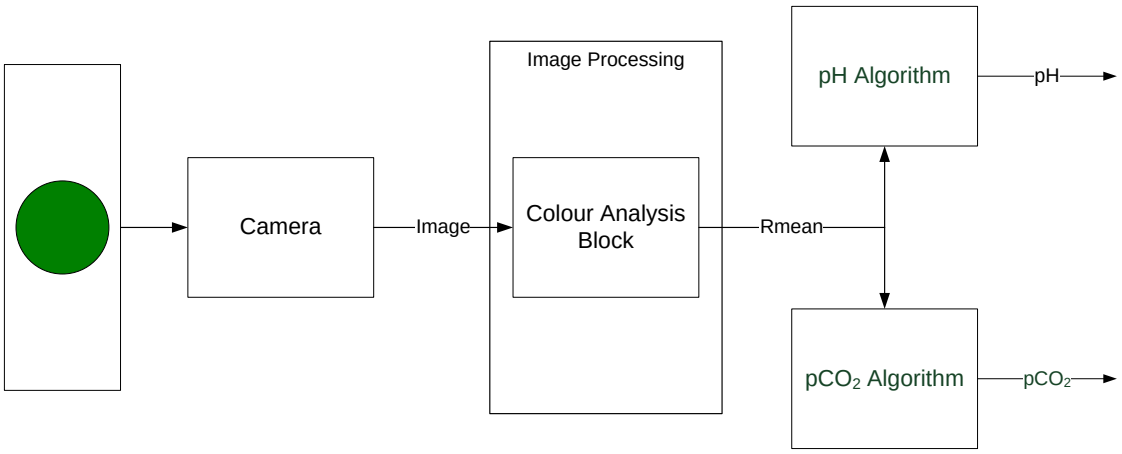


Figure 6.1a: Block diagram of the complete Colorimetric Analyser

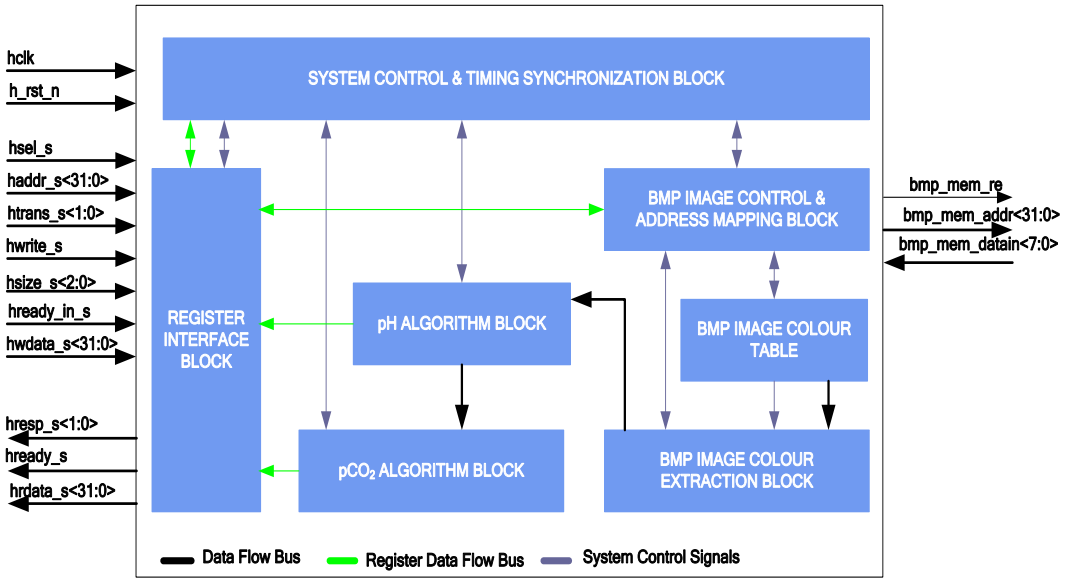


Figure 6.1b: Block diagram of the implementation of the complete Colorimetric Analyser

The register interface block provides the interface for the external I/O signals. The bmp image control and the address mapping block interfaces with the memory where the images are stored. The System Control and Timing Synchronisation block controls the operation of the analyser. Figure 6.2 shows the flow chart for the controller.

The controller is finite state machine based; the first step initialises the unit and the address location of the RAM holding the images are initialised in the Image Control and Address Mapping block. In the next step the colour analysis unit reads the image for the pH reaction and extracts the mean value of the colour red from the image. The mean value of the colour red is fed to the pH algorithm and the pH value of the sample is calculated and stored. The controller goes back to the colour analysis unit to read the images for the pCO_2 reaction and generate the mean value of the colour red. The mean value is passed to the pH unit to calculate the pH of the sample and then is passed on to the pCO_2 algorithm block to calculate the value of pCO_2. The controller returns back to the start state.

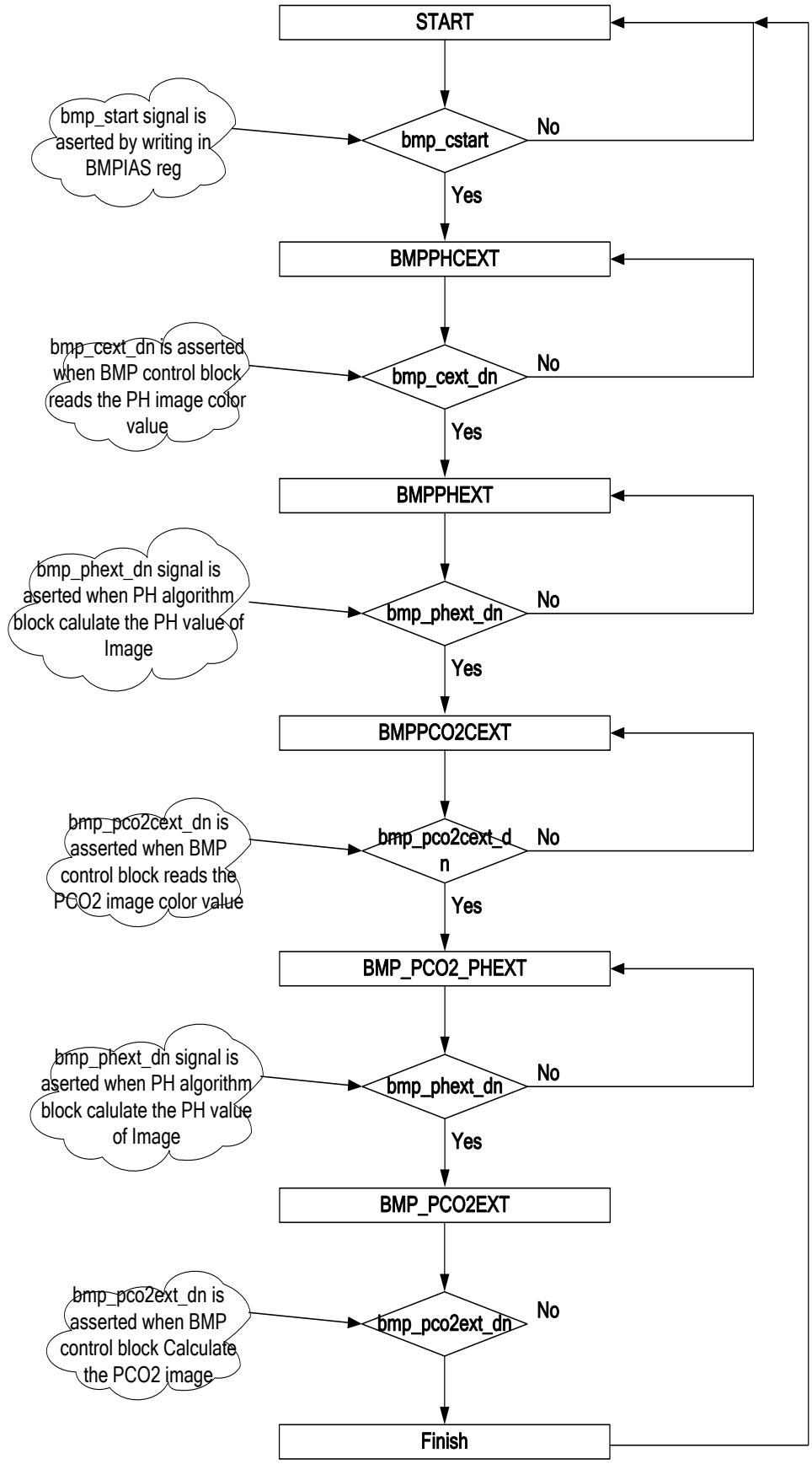


Figure 6.2: Flowchart for the system control and timing synchronization block

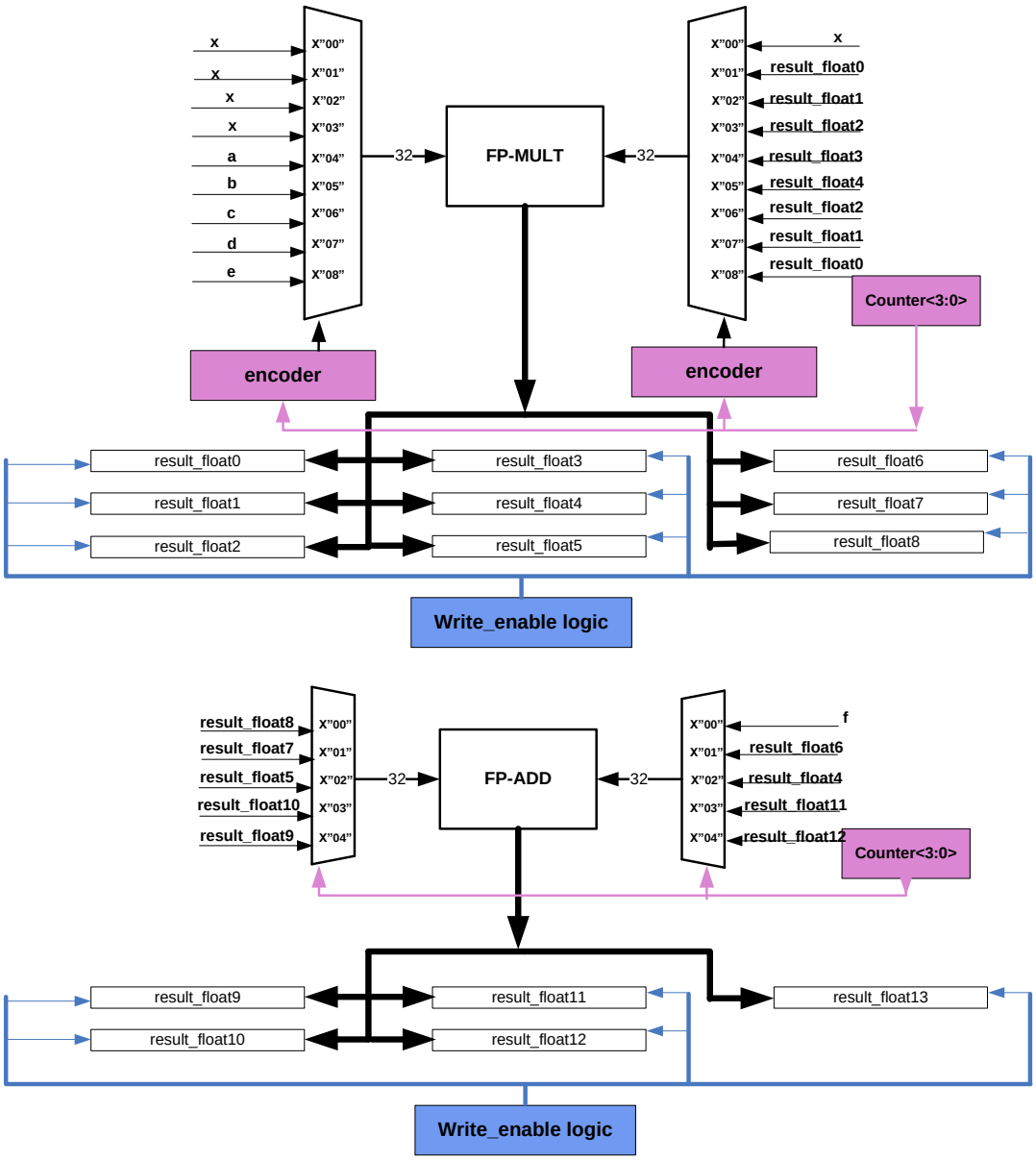


Figure 6.3: Implementation of the pH algorithm block

Figure 6.3, 6.4a and 6.4b shows the implementation of the pH and pCO_2 algorithm blocks. The implementation uses two floating point multipliers and two floating point adders to implement the mathematical operations. The multiplier in case of the pH equation receives inputs from two 9 to 1 multiplexers and the select lines of the multiplexers are controlled by a 4 bit counter. Similarly the floating point adder receives the inputs from two 5 to 1 multiplexers also controlled by a 4 bit counter.

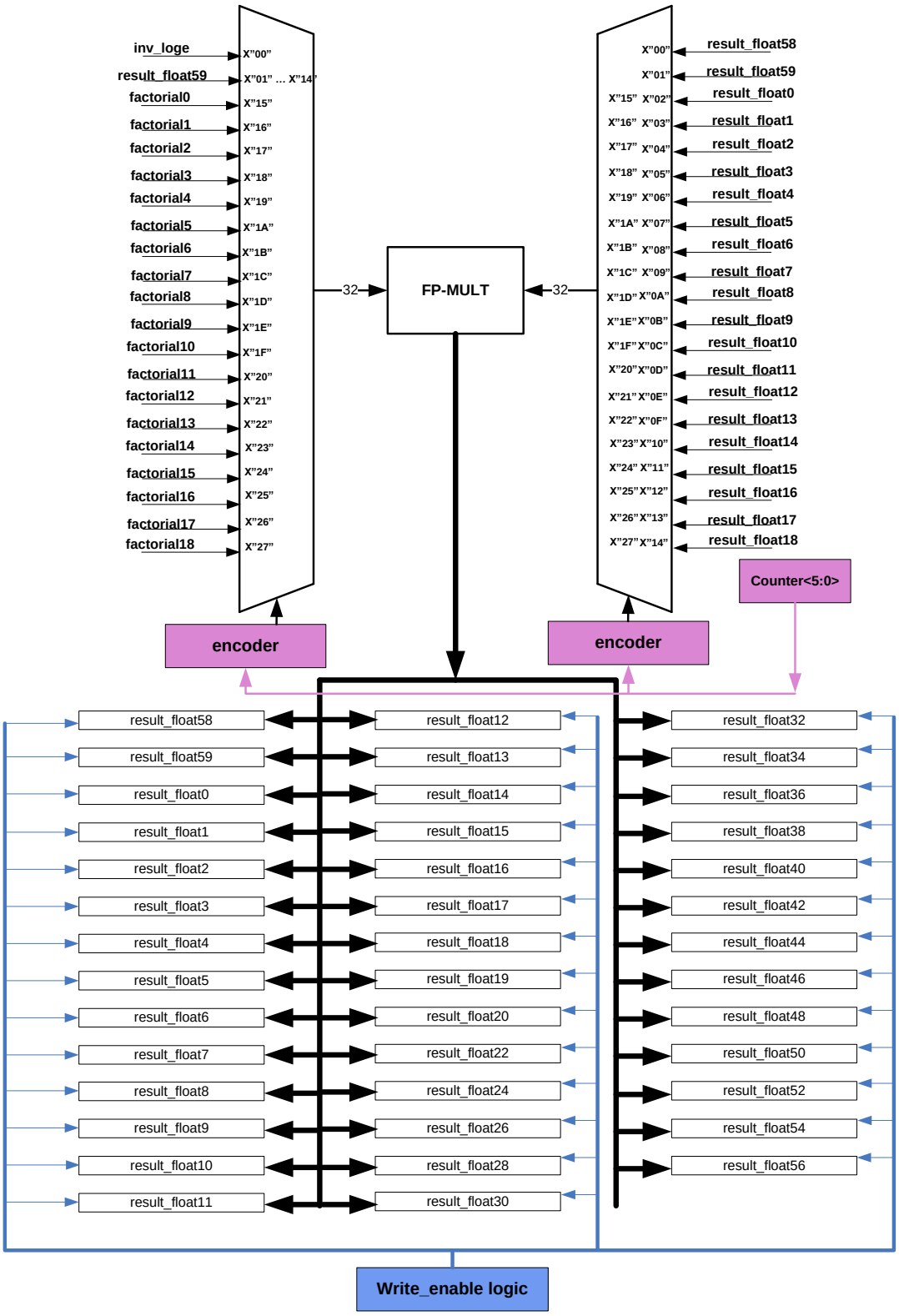


Figure 6.4a: Implementation of the pCO_2algorithm block

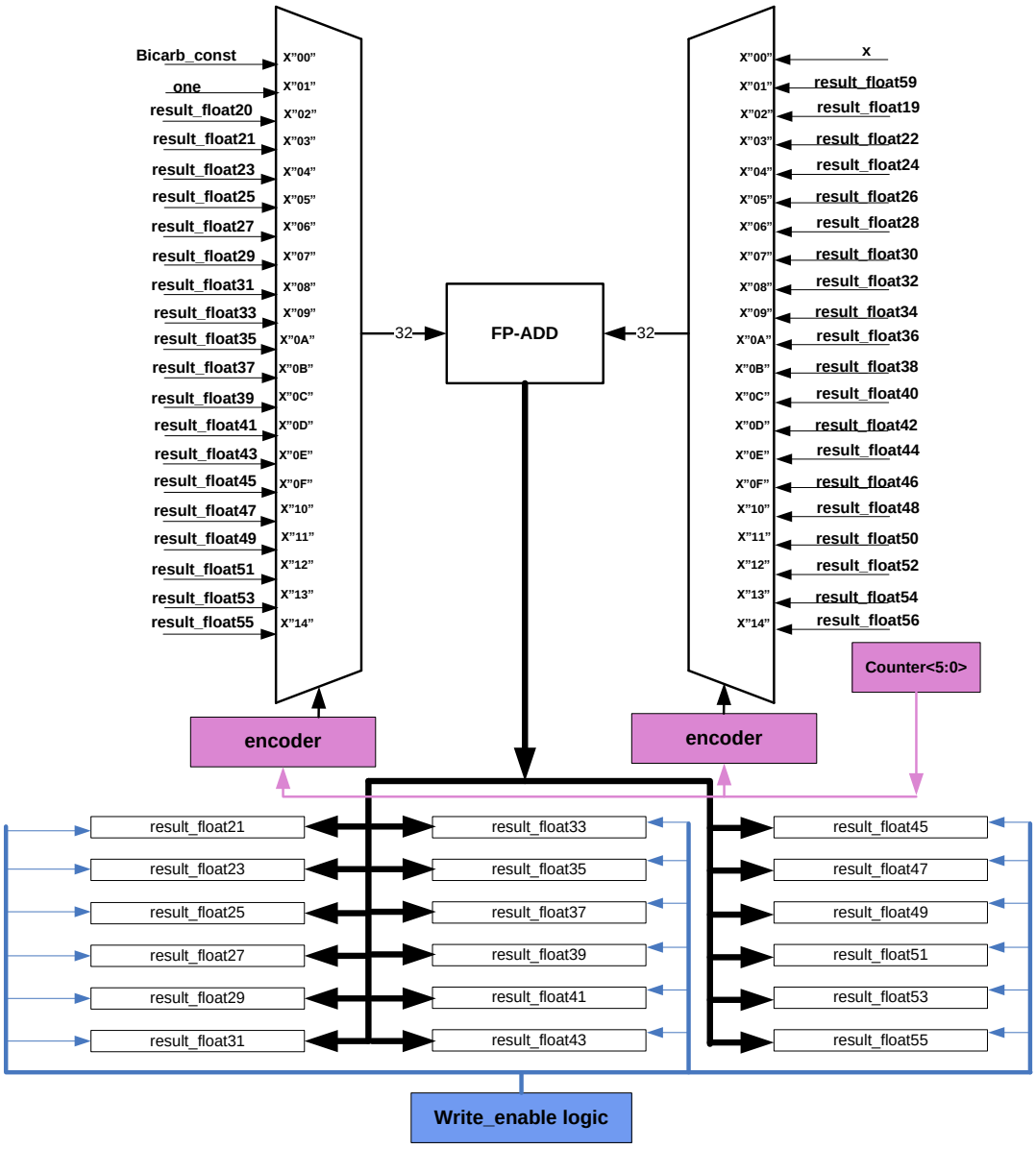


Figure 6.4b: Implementation of the pCO_2algorithm block

The intermediate results from the addition and multiplication are stored in a bank of registers result_float. The multiplexers in the implementation of the pCO_2 equation are controlled by a 5 bit counter. This implementation of the pH and pCO_2 saves a lot of area as there are multiple floating point multiplication and addition operations required for the implementation of the pH and pCO_2 algorithms.

The design of the image based colorimetric analyser is implemented in VHDL using ModelSim.The complete VHDL code for the design can be seen in Appendix B. Table 6.1 presents the simulation results for the mathematical equation to calculate pH. The table

also compares the pH values calculated from the equation and the implementation of the equation in VHDL. The implementation results show a deviation of ±1% from the expected values of pH which is shown in figure 6.5.

Table 6.1: Comparison of the calculated pH values

Normalised Value of the colour red	Expected value	pH value calculated from the polynomial equation	pH value calculated from the deign implemented in VHDL
0.954699	6.1	6.108441482	6.1113944
0.926276	6.2	6.208209348	6.211081
0.904845	6.3	6.268356701	6.2711606
0.81004	6.4	6.439119068	6.4416285
0.776017	6.5	6.481164205	6.4835677
0.657841	6.6	6.621812343	6.623851
0.617768	6.7	6.675589329	6.6775055
0.544941	6.8	6.78146707	6.783156
0.459298	6.9	6.907964435	6.9093895
0.373921	7	7.019098417	7.0202575
0.275337	7.1	7.120976843	7.1218295
0.21612	7.2	7.180752219	7.1814213
0.156666	7.3	7.263092613	7.2635784
0.098817	7.4	7.39725181	7.397559
0.063207	7.5	7.525849923	7.5260463
0.044708	7.6	7.611932163	7.612068
0.020155	7.7	7.751304344	7.7513695
0.012102	7.8	7.804114664	7.8041506
0.007435	7.9	7.836469593	7.836492

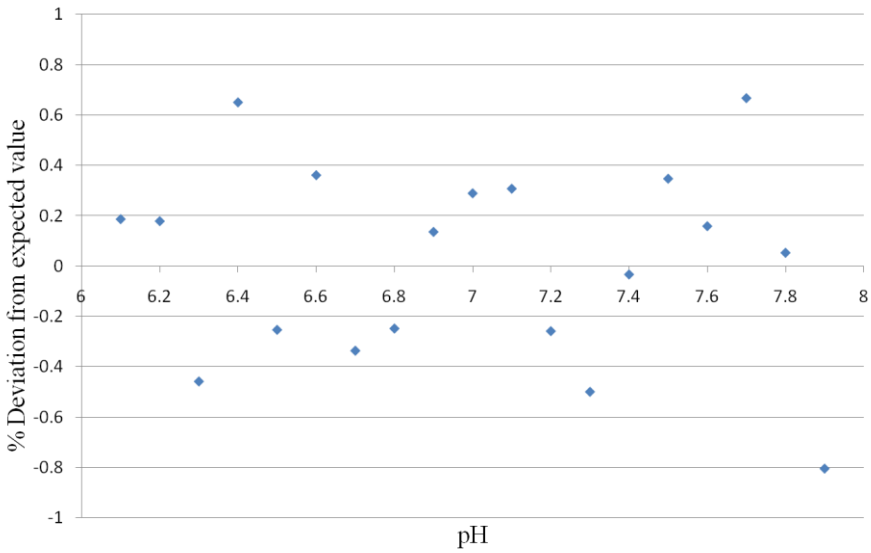


Figure 6.5: % deviation of the calculated pH value from the expected values

Table 6.2 presents the simulation results for the implementation of the equation to calculate PaCO2. The table also compares the PaCO2 values calculated from the equation

and the implementation of the equation in VHDL. The implementation results show a deviation of ±10% from the expected values of PaCO2 shown in figure 6.6.

Table 6.2: Comparison of the calculated pCO2 values

pH Value of buffer	Expected PaCO2	Calculated PaCO2	Calculated PaCO2 from the VHDL implementation	Deviation %
6.8	194.1725968	202.6379709	201.85382	3.955874
6.9	154.2368546	151.4341215	150.93945	-2.13788
7	122.5147509	117.2438324	116.93255	-4.55635
7.1	97.31697534	92.72819504	92.54724	-4.90124
7.2	77.3016606	80.80468884	80.68098	4.3716
7.3	61.40292288	66.84922095	66.77502	8.748927
7.4	48.77410018	49.08371771	49.049362	0.564361
7.5	38.74266463	36.50392985	36.487736	-5.82027
7.6	30.77440808	29.94039815	29.931229	-2.73987
7.7	24.44499369	21.72129426	21.718166	-11.155
7.8	19.41735858	19.23426058	19.232777	-0.9506

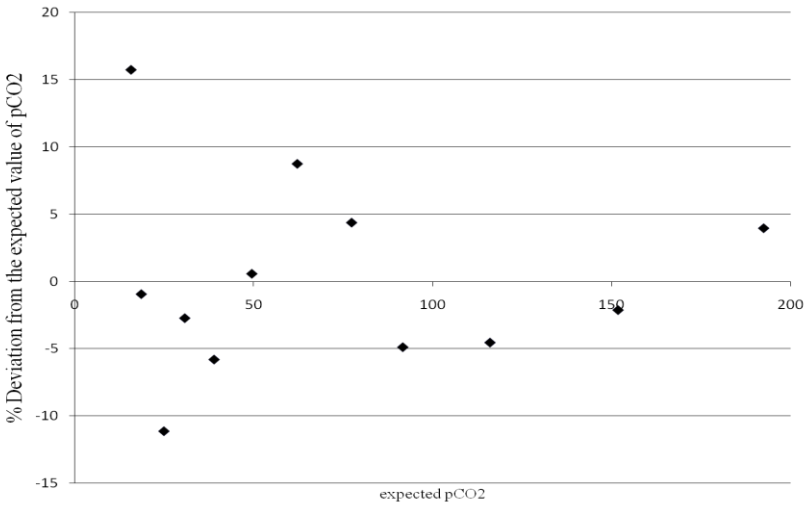


Figure 6.8: % deviation of the calculated pCO_2 value from the expected values

6.2.1 Synthesis Results

The design was synthesised for both FPGA based hardware and an ASIC based hardware. The synthesis results for the FPGA Implementation are shown in Table 6.3. The design met the timing constraints and has a maximum operating frequency of 70.671 MHz.

Table 6.3: The Implementation and Analysis results for the Xilinx FPGA

Top Level Entity Name	**cph_analyzer**
Family	Spartan 3
Device	3s1500lfg320-4
Number of Slices	3614 out of 13312
Number of Slice Flip Flops	3872 out of 26624
Number of 4 input LUTs	4534 out of 26624
Number of IOs	158
Number of bonded IOBs	129 out of 221
Number of MULT18X18s	8 out of 32
Number of GCLKs	1 out of 8
Required Clock Frequency	50 MHz
Maximum Clock Frequency	70.671 MHz

The design was synthesized using the RTL compiler for an implementation in 0.35µm technology. The implementation results are shown in table 6.4 and a schematic of the implementation is shown in figure6.9. The design meets the constraints set for timing and has a positive slack. The design used 58316 cells and has a total power consumption of 495.11 mW.

Table 6.4: Power and Timing Analysis for the implementation in 0.35µ CMOS

Parameter	Performance Value
Supply Voltage	3.3v
Required Clock Frequency	50 MHz – 20ns
Maximum Clock Frequency	50.005Mhz – 19.998 ns
Average Power Consumption	495.11 mW
Dynamic Power	495.102 mW
Leakage Power	5.64 µW
Cells	58316

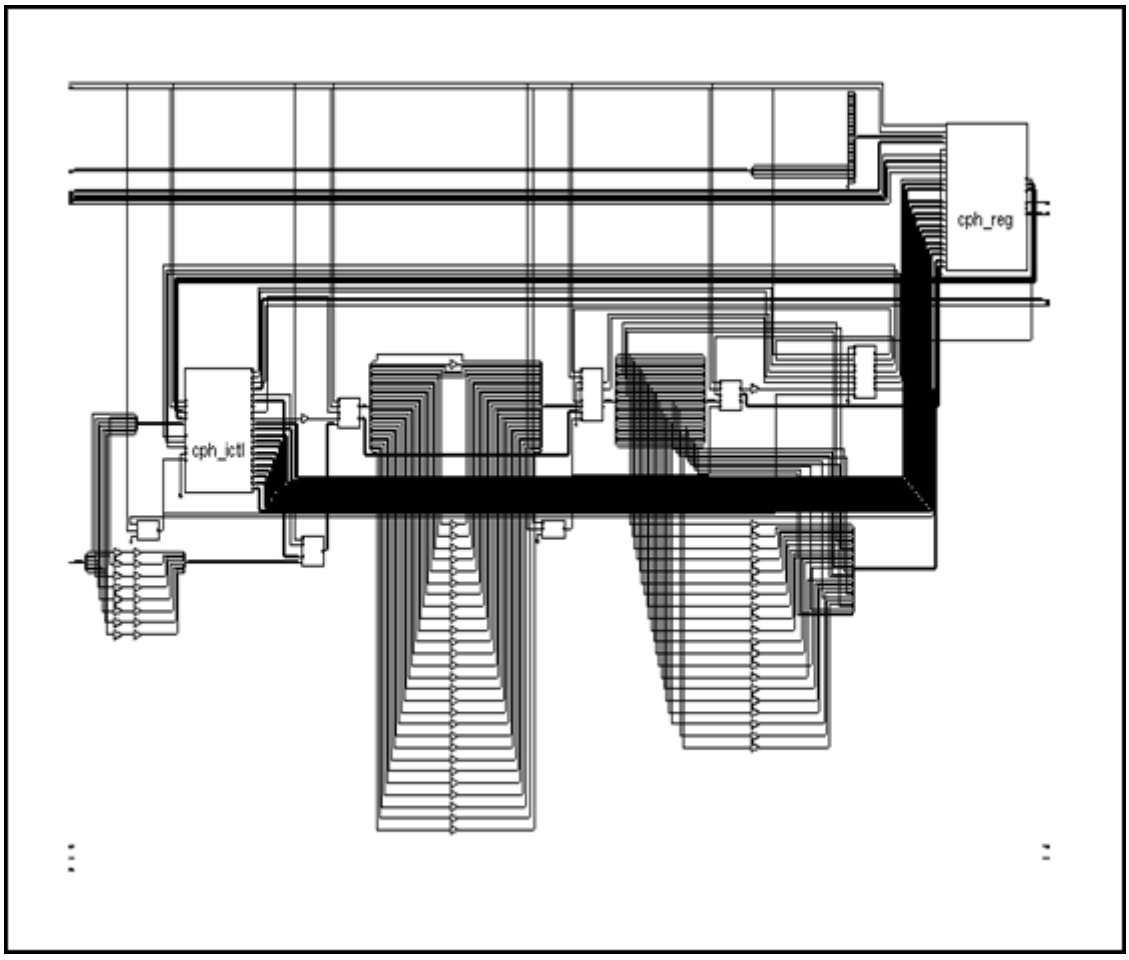


Figure 6.9: RTL Schematic of the implementation

The layout of the design was performed on the Cadence SoC-Encounter tool using the 0.35µ CMOS technology library from Austria Microsystems. Figure 6.10 illustrates the layout of the Image based colorimetric analyser.

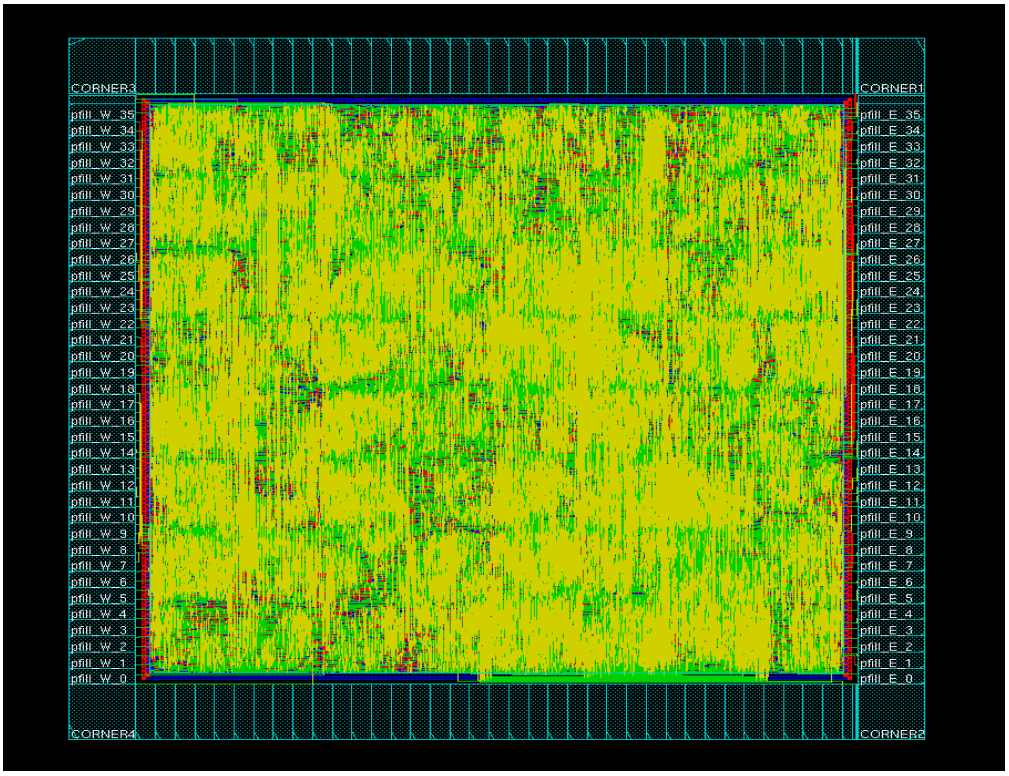


Figure 6.10: Layout of the Image based Colorimetric Analyser

6.3 Performance Evaluation

In this section the performance of the Colorimetric Analyser is evaluated against the current analysers available in the market. Table 6.5 presents the performance characteristics based on their size, power consumption, accuracy and cost and those of the colorimetric analyser in comparison to them. The above mentioned characteristics were chosen as the performance criteria as the Colorimetric analyser is aimed as a portable solution for blood gas analysis.

6.3.1 Size:

The colorimetric analyser as a system has no moving parts and the processing of the analysis is fit on a single chip therefore significantly reducing the size of the analyser. The only parts external to the chip are the camera and the indicator chemistry apparatus. The colorimetric analyser matches the i-STAT analyser and the ISFET/HEMT based solid State sensors in terms of size and will be a handheld device.

6.3.2 Power Consumption:

The total power consumption of the chip is 495.11m w at an operating speed of 50 MHz. The latest advances in the technology of cameras used in mobile phones and digital camera has greatly reduced the power consumption of the camera and its associated electronics. Thus the power consumption of the colorimetric analyser can be limited to be operable on a battery powered power supply.

Table 6.5: Performance characteristics of current analysers in the market[45, 50, 107-109]

	Roche ABL 520	OPTI Medical Systems OPTI CCA-TS	Abott Point of Care - i-STAT	ISFET/ HEMT based pH Measurement	Mass Spectrometer	Capnometers	Image Based Colorimetric Analyser **(This Work)**
Technique	Electrochemical	Optical	Electrochemical	Semiconductor chemistry	Spectroscopy	Gas Chromatography	Colorimetry
Size	Bench top	Compact Bench top	Handheld	Handheld	Big	Handheld	Handheld
Power	Mains Supply	Mains Supply and supported by a battery for extended Operation	Battery Operated	Battery Operated	Mains	Battery/Mains	Battery
Accuracy	Very Good	Good	Good	moderate	Very Good	Good	Good

6.3.3 Accuracy:

The Colorimetry based analyser can calculate the pH value with a deviation of ±1 % and the partial pressure of carbon dioxide with a deviation of ±10%. The results are not comparable with the accuracy of the other analysers but are comparable with the analyser based on the semiconductor sensors. The accuracy of the product can be improved using an indicator with a tighter range.

6.3.4 Cost:

The cost of the analyser is greatly reduced as the construction of the analyser is simple and the camera and the indicator chemistry material can come from off the shelf products. The use of CMOS technology to develop the processor for the analyser also helps in the reduction of the cost of the product. The use of simple construction, CMOS for the processor and off the shelf components makes this a cheap and cost effective solution in comparison to the other analysers listed on the table. The semiconductor based solid state sensors are the only other solution which can provide the same cost effectiveness but they suffer from performance related issues discussed in chapter 2. The total cost of the system including the off chip components and the camera will not exceed more than $US 500.

The image based colorimetric blood gas analysis system matches the performance criteria set in terms of accuracy and outperforms the current solutions in terms of the size, power consumption and cost of the system when put in perspective for a portable blood gas system.

6.4 Conclusion

The focus of this chapter was to present the design and implementation of the integrated Images based Colorimetric algorithm for portable blood gas analysis and analyse the performance of the device.

The design of the complete block is first presented with the implementation strategies for the pH and pCO_2 algorithm blocks. The design of the algorithm blocks utilizes a combination of multiplexers and counters to control the multiplexers to execute the multiple addition and multiplication operations using only two floating point multipliers and two floating point adders; thus reducing the area required for the implementation of the algorithm blocks. The results from the implementation show the deviation from the observed results for the mathematical equation for pH is ±1% and for the PaCO2 equation the deviation is ±10%.

The design is synthesized for hardware implementation both on a FPGA and an ASIC. The synthesis results for the FPGA show that the design meets the timing constraints set for the design. The ASIC synthesis results for a 0.35µ CMOS process show the design meeting the timing constraints of the design and a power consumption of 495mW at an operating speed of 50 MHz.

The performance of the design was evaluated for portability based on the criteria of size, power consumption, cost and accuracy. The design with the use of a simple construction with no moving parts and off shelf components for the camera and indicator chemistry proves to be very cost effective. The use of CMOS technology for both the processor and the camera reduces the size of the product. The use of CMOS technology and lesser parts also reduces the power consumption of the product. The accuracy of the device is comparable to some of the current techniques available; thus proving the suitability of the system as a solution for portable blood gas analysis.

Chapter 7: Conclusion and Future Work

This thesis presented the design and implementation of an Image based Colorimetric technique for portable blood gas analysis. The design is based on a floating point architecture comprising of two floating point adders, two floating point multipliers and eight multiplexers. The design detects the colour of the indicator and calculates the pH value from the colour value. The pH value is utilized to calculate the partial pressure of carbon dioxide.

Previous work involved with blood gas analysis has shown two major techniques electrochemical analysis and optical commonly used to detect the blood gases. The analysers based on these techniques have predominately been high powered machines with a bulky size and high cost which is not suitable for portability. The design presented in this thesis was an attempt to implement a colorimetric algorithm for blood gas analysis. The target of this work was to implement a technique based on colour analysis performed on an image capturing the change in colour of an indicator to determine the pH and partial pressure of carbon dioxide providing a cheaper, smaller and a low power solution for a portable system. The design presented in this thesis addresses the requirements of power, area, cost and size for the application of this technique for blood gas analysis. The work demonstrates the effectiveness of the colorimetric technique in meeting these requirements.

Section 7.1 presents some of the key research outcomes from this work and Section 7.2 concludes this thesis and presents the future direction and scope of work for the research presented in this thesis.

7.1 Summary of research Contribution

This thesis presents the design and implementation of an Image based Colorimetric technique for portable blood gas analysis. The portability of the technique was based on the use of the colorimetric approach to reduce the size of the analyser and its power

consumption. The colorimetric technique is based on measuring change in colour of an indicator in reaction to a change in pH. The research explored the indicator chemistry and mapped the relation between the change in colour of the indicator and pH to a mathematical equation. To evaluate the colour of the indicator the reaction is imaged and the image is analysed to provide a histogram which quantifies the colour of the indicator in the three primary colours Red Green and Blue. The experiments conducted to map the relation of the colour of the indicator to pH value show the colour red reacts the sharpest to the change in pH. The trends in the colour red with respect to the change in pH were chosen to map the mathematical relation. The mathematical equation was derived using curve fitting and regression analysis techniques and the regression analysis yielded a fifth order polynomial equation defining the relation between the colour red and pH. The determination of the partial pressure of carbon dioxide (pCO_2) is based on the pH measurement technique which results in an exponential equation.

The implementation of the colorimetric technique is an implementation of the fifth order polynomial equation and the exponential equation. The implementation is based on floating point architecture as the coefficients in the polynomial equation are fractional and require accurate representation to provide an accurate answer. The pH equation was implemented using one floating point adder, one floating point multiplier and four multiplexers controlled by a 5 bit counter. The pCO_2 equation; an exponential equation is implemented as a Taylor Series and implemented using one floating point adder, one floating point multiplier and four 32 to 1 multiplexer controlled by a 6 bit counter. The design also includes a Colour Analysis Block which reads the image file from the memory and processes the image file to generate a mean value of the colour red from the selected pixels. A top level controller based on a finite state machine controls the operations of the Colour analysis block and the colorimetric equation for pH and pCO_2.

Some of the key contributions from this research work are listed below;

- Investigation into Blood Gases and the current techniques to measure the blood gases focussing on the requirements for a portable solution for measuring blood gases. A review of the current measurement techniques for blood gas analysis, with an in-depth study of the suitability to build a portable solution.
- The pH chemistry and indicator chemistry were investigated as a part of this work and the relation between the colour of the indicator and the pH value was

explored. The quantification of colour in terms of the three primary colour components red, green, and blue using colorimetry was also explored.

- Experiments conducted to establish a relation between the colour of the indicator and pH showed a definite and repeatable trend. The trends established the colour component red as the colour with the sharpest response to the change in pH.
- Mathematical modelling of the trends of the colour component red with respect to the change in pH using curve fitting and regression analysis. The relation was modelled using a polynomial regression model to derive a fifth order polynomial equation.
- Design and implementation of the fifth order polynomial equation to calculate the pH value using floating point architecture. The polynomial equation with fractional constants was implemented in floating point for accuracy in calculations. Design and implementation of the exponential equation calculating the partial pressure of carbon dioxide. The exponential equation was implemented as a Taylor Series.
- Design and implementation of a colour analysis block. The block reads a bitmap format image from the memory location and determines the mean value of the colour component red from a selected group of pixels. It achieves this by reading the specific pixel locations in the memory and adds the values to produce the mean value.
- Test and analysis of the designs show that the pH results are calculated with a ±1% deviation from the expected value and the pCO_2 results are calculated with a ±10% deviation from the expected results. The implementation results of the design using FPGA and ASIC devices show that the design has a power consumption of 495.15mW at a speed of 50MHz.
- The design matches the requirements set in terms of power, area and size as the design is a simple construction and the processing unit can be fit in single chip. The off-chip components the camera and the indicator can be used off the shelf. The size of the complete system is comparable to the size of a handheld blood pressure meter; the system has low power consumption and can run on batteries. The cost of the complete system including the off chip components and the camera should not exceed more than USD 500.

7.2 Future Research Directions

The work presented in this thesis has discussed the use of colour as an agent to identify the characteristics of the blood sample. The method of Colorimetry to quantify the colour has been successfully implemented to measure the pH and partial pressure of carbon dioxide. The technique has not been applied to the measurement of the partial pressure of oxygen as a suitable indicator which provides the same colour change as the bromothymol blue indicator for pH was not available. It is hoped that research into an indicator which reacts to oxygen and produces a change in colour will allow the use of the colorimetric algorithm to detect the partial pressure of oxygen.

The colorimetric algorithm can be applied to detect the blood gas parameters other than the ph and pCO_2. The measurement of the bicarbonate ion status in blood can be calculated from the pH and pCO_2 using the Henderson Hasselbalch equation. The colorimetric technique can be used with applications other than the biomedical ones targeted in this project. It can be applied in the food industry to map the colour of the product to the quality of the product. The technique can be modified for any application where colour is used to identify the state of a product.

The research presented in this work is a practical attempt to deal with the shortcomings present in the current techniques to build a portable system. The author hopes that the contributions towards this research are measurable and appropriate.

Bibliography

[1] P. Bonato, "Wearable sensors/systems and their impact on biomedical engineering," *Engineering in Medicine and Biology Magazine, IEEE,* vol. 22, pp. 18-20, 2003.

[2] P. Bonato, "Wearable Sensors and Systems," *Engineering in Medicine and Biology Magazine, IEEE,* vol. 29, pp. 25-36.

[3] A. Dittmar and A. Lymberis, "Smart clothes and associated wearable devices for biomedical ambulatory monitoring," in *Solid-State Sensors, Actuators and Microsystems, 2005. Digest of Technical Papers. TRANSDUCERS '05. The 13th International Conference on*, 2005, pp. 221-227 Vol. 1.

[4] A. Lymberis, "Progress in R&D on wearable and implantable biomedical sensors for better healthcare and medicine," in *Microtechnology in Medicine and Biology, 2005. 3rd IEEE/EMBS Special Topic Conference on*, 2005, pp. 296-298.

[5] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks,* vol. 38, pp. 393-422, 2002.

[6] K. Hung, Y. T. Zhang, and B. Tai, "Wearable medical devices for tele-home healthcare," in *Engineering in Medicine and Biology Society, 2004. IEMBS '04. 26th Annual International Conference of the IEEE*, 2004, pp. 5384-5387.

[7] B. Paolo, "Wearable Technology in the Rehabilitation Hospital of the Future," in *Life Science Systems and Applications Workshop, 2006. IEEE/NLM*, 2006, pp. 1-2.

[8] N. Joglekar, J. Chandran, R. Veljanovski, A. Stojcevski, A. Zayegh, and M. Dorairaj, "A Review of Blood Gas Sensors and a Proposed Portable Solution," in *Integrated Circuits, 2007. ISIC '07. International Symposium on*, 2007, pp. 244-247.

[9] S. Patel, C. Mancinelli, P. Bonato, J. Healey, and M. Moy, "Using Wearable Sensors to Monitor Physical Activities of Patients with COPD: A Comparison of Classifier Performance," in *Wearable and Implantable Body Sensor Networks, 2009. BSN 2009. Sixth International Workshop on*, 2009, pp. 234-239.

[10] D. Fitrio, "Power Management Schemes for Ultra Low Power Biomedical Devices," in *School of Electrical Engineering, Faculty of Health,Engineering and Sciences*. vol. Doctor of Philosophy: Victoria University, 2008.

[11] "Body Area Networks." vol. 2010: http://www.wica.intec.ugent.be/research/wireless/body-area-networks.

[12] "The Human Heart: Blood." vol. 2010: http://www.fi.edu/learn/heart/blood/blood.html, 2011.

[13] C. Turkington, *Gale Encyclopedia of Medicine*: The Gale Group, December 2002.

[14] K. Rogers, *The Respiratory System*, 1 ed.: Britannica Educational Publishing, Jan 2010.

[15] "Ciculatory System," http://www.daviddarling.info/encyclopedia/C/circulatory_system.html, 2010.

[16] A. Hasan, "Handbook of Blood Gas/Acid-Base Interpretation," Springer London Ltd, 2009.

[17] H. W. Davenport, *The ABC of acid-base chemistry; the elements of physiological blood gas chemistry for medical students and physicians*, 5th ed. Chicago: University of Chicago Press, 1969.

[18] J. W. Severinghaus, "Blood gas calculator," *J Appl Physiol,* vol. 21, pp. 1108-1116, May 1, 1966 1966.

[19] JOHN W. SEVERINGHAUS, P. ASTRUP, and JOHN F. MURRAY, "Blood Gas Analysis and Critical Care Medicine," *Am. J. Respir. Crit. Care Med.,* vol. 157, pp. S114-122, April 1, 1998 1998.

[20] J. W. Severinghaus, "A combined transcutaneous PO2-PCO2 electrode with electrochemical HCO3- stabilization," *J Appl Physiol,* vol. 51, pp. 1027-1032, October 1, 1981 1981.

[21] R. S. Khandpur, *Handbook of Analytical Instruments*: Tata McGraw Hill Handbooks, 1989.

[22] J. G. Webster, *The measurment, instrumentation, and sensors handbook*. Boca raton,Fla: CRC Press published in cooperation with IEEE Press, 1999.

[23] J. W. Severinghaus, "First electrodes for blood PO2 and PCO2 determination," *Journal of Applied Physiology,* vol. 97, pp. 1599-1600, November 1, 2004 2004.

[24] Y. Mendelson and R. A. Peura, "Noninvasive Transcutaneous Monitoring of Arterial Blood Gases," *Biomedical Engineering, IEEE Transactions on,* vol. BME-31, pp. 792-800, 1984.

[25] G. Stege, F. J. J. van den Elshout, Y. F. Heijdra, M. J. T. van de Ven, P. N. R. Dekhuijzen, and P. J. E. Vos, "Accuracy of Transcutaneous Carbon Dioxide Tension Measurements during Cardiopulmonary Exercise Testing," *Respiration,* vol. 78, pp. 147-153, 2009.

[26] J. M. Steinacker and W. Spittelmeister, "Dependence of transcutaneous O2 partial pressure on cutaneous blood flow," *J Appl Physiol,* vol. 64, pp. 21-25, January 1, 1988 1988.

[27] V. Bernet-Buettiker, M. J. Ugarte, B. Frey, M. I. Hug, O. Baenziger, and M. Weiss, "Evaluation of a New Combined Transcutaneous Measurement of PCO₂/Pulse Oximetry Oxygen Saturation Ear Sensor in Newborn Patients," *Pediatrics,* vol. 115, pp. e64-e68, 2005.

[28] S. Iguchi, K. Mitsubayashi, T. Uehara, and M. Ogawa, "A wearable oxygen sensor for transcutaneous blood gas monitoring at the conjunctiva," *Sensors and Actuators B: Chemical,* vol. 108, pp. 733-737, 2005.

[29] E. Toba, "Fiber optic fluorosensor for oxygen measurement," in *Instrumentation and Measurement Technology Conference, 1999. IMTC/99. Proceedings of the 16th IEEE*, 1999, pp. 1426-1430 vol.3.

[30] N. Joglekar, R. Veljanovski, and A. Zayegh, "Implementation of ratiometric algorithm to compute partial pressure of Oxygen for a blood oxygen analyser," in *Integrated Circuits, ISIC '09. Proceedings of the 2009 12th International Symposium on*, 2009, pp. 204-207.

[31] F. J. F. Martin, J. C. C. Rodriguez, J. C. A. Anton, J. C. V. Perez, I. Sanchez-Barragan, J. M. Costa-Fernandez, and A. Sanz-Medel, "Design of a low-cost optical instrument for pH fluorescence measurements," *Instrumentation and Measurement, IEEE Transactions on,* vol. 55, pp. 1215-1221, 2006.

[32] R. Wolthuis, D. McCrae, E. Saaski, J. Hartl, and G. Mitchell, "Development of a medical fiber-optic pH sensor based on optical absorption," *Biomedical Engineering, IEEE Transactions on,* vol. 39, pp. 531-537, 1992.

[33] B. R. Soller, "Design of intravascular fiber optic blood gas sensors," *Engineering in Medicine and Biology Magazine, IEEE,* vol. 13, pp. 327-335, 1994.

[34] K. J. Proctor and G. P. Seifert, "Colorimetric blood-gas monitoring sensors," in *Fiber Optic Sensors in Medical Diagnostics*, Los Angeles, CA, USA, 1993, pp. 70-77.

[35] B. J. C. Deboux, E. Lewis, P. J. Scully, and R. Edwards, "A novel technique for optical fiber pH sensing based on methylene blue adsorption," *Lightwave Technology, Journal of,* vol. 13, pp. 1407-1414, 1995.

[36] F. Baldini, "Critical review of pH sensing with optical fibers," in *Chemical, Biochemical, and Environmental Fiber Sensors X*, Boston, MA, USA, 1999, pp. 2-9.

[37] K. J. Proctor and G. P. Seifert, "Colorimetric blood-gas monitoring sensors," 1993, pp. 70-77.

[38] H. Aizawa, K. Ohkubo, T. Katsumata, and S. Komuro, "Multi-channel optical pH sensor using organic dye," in *Control, Automation and Systems, 2007. ICCAS '07. International Conference on*, 2007, pp. 2365-2368.

[39] J. L. Gehrich, D. W. Lubbers, N. Opitz, D. R. Hansmann, W. W. Miller, J. K. Tusa, and M. Yafuso, "Optical Fluorescence and Its Application to an Intravascular Blood Gas Monitoring System," *Biomedical Engineering, IEEE Transactions on,* vol. BME-33, pp. 117-132, 1986.

[40] S. A. Grant and R. S. Glass, "A sol-gel based fiber optic sensor for local blood pH measurements," *Sensors and Actuators B: Chemical,* vol. 45, pp. 35-42, 1997.

[41] M. Cajlakovic, A. Bizzarri, M. Suppan, C. Konrad, M. Tscherner, V. Ribitsch, E. Beran, and I. Knez, "Simultaneously monitoring of tissue O<inf>2</inf> and CO<inf>2</inf> partial pressures by means of miniaturized implanted fiber optical sensors," in *Sensors, 2009 IEEE*, 2009, pp. 31-36.

[42] S. M. R. V.Nagendranath, "Arterial Blood Gas Monitoring," *Indian Journal of Anaesthesia,* pp. 289-297, 2002.

[43] "Mass Spectrometry." vol. 2011: http://www2.chemistry.msu.edu/faculty/reusch/VirtTxtJml/Spectrpy/MassSpec/masspec1.htm.

[44] "Capnometry/Capnography." vol. 2011: www.ebme.co.uk.

[45] "Capnometer and Capnometry." vol. 2011: http://www.breathing.com/articles/capnometry.htm.

[46] M. S. Z. Abidin, M. E. Sharifabad, A. M. Hashim, S. F. A. Rahman, A. R. A. Rahman, R. Qindeel, N. A. Omar, A. A. Aziz, M. R. Hashim, and M. H. M. Mohamed, "Gateless-FET undoped AlGaN/GaN HEMT structure for liquid-phase sensor," in *Semiconductor Electronics (ICSE), 2010 IEEE International Conference on*, pp. 309-312.

[47] S. Swaminathan, S. M. Krishnan, K. Lim Wee, Z. Ahamed, and G. Chiang, "Microsensor characterization in an integrated blood gas measurement system," in *Circuits and Systems, 2002. APCCAS '02. 2002 Asia-Pacific Conference on*, 2002, pp. 15-20 vol.1.

[48] H. Hsin-Shun, L. Chao-Wei, and C. Hsien-Chin, "High sensitivity pH sensor using Al<inf>X</inf>Ga<inf>1-X</inf>N/GaN HEMT heterostructure design," in *Electron Devices and Solid-State Circuits, 2008. EDSSC 2008. IEEE International Conference on*, 2008, pp. 1-4.

[49] M. Stutzmann, G. Steinhoff, M. Eickhoff, O. Ambacher, C. E. Nebel, J. Schalwig, R. Neuberger, and G. Müller, "GaN-based heterostructures for sensor applications," *Diamond and Related Materials,* vol. 11, pp. 886-891, 2002/6//.

[50] "Roche ABL 520 Blood Gas Analyzer ". vol. 2011: http://www.gmi-inc.com/ABL-520-Blood-Gas-Analyzer.html, 2011.

[51] A. Carpi, "Acids and Bases: An Introduction." vol. 2011: http://www.visionlearning.com/library/module_viewer.php?mid=58, 2003.

[52] "Indicators." vol. 2011: http://www.science.uwaterloo.ca/~cchieh/cact/c123/indicator.html.

[53] H. Galster, *pH measurement: fundamentals, methods, applications, instrumentation*: Weinheim,Federal Republic of Germany; New York, USA, 1991.

[54] H. J. Trussell, *Fundamentals of digital imaging*. Cambridge; New York: Cambridge University, 2008.

[55] T. V. W. Karim Nice, Gerald Gurevich, "How digital cameras work?," http://electronics.howstuffworks.com/cameras-photography/digital/digital-camera2.htm.

[56] R. Bourne, *Fundamentals of digital imaging in Medicine*. London: Springer, 2010.

[57] C. o. C. Optical Society of America, *The science of colour*. Washington: Optical Society of America, 1963.

[58] M. D. Fairchild, *Color Appearance Models*: John Wiley & Sons, Ltd., 2005.
[59] R. W. G. Hunt, *Measuring colour*. Kingston-upon-Thames: Fountain, 1998.
[60] A. R. R. Noboru Ohta, *Colorimetry: fundamentals and applications*. Hoboken, NJ: Wiley, 2005.
[61] "Cone Cell." vol. 2011: http://en.wikipedia.org/wiki/Cone_cells.
[62] G. Hernandez, "Hue." vol. 2011: http://www.georgehernandez.com/h/xzMisc/Color/Hue.asp, 2007.
[63] L. H. Kahane, *Regression Basics*, 2nd ed. Los Angeles: Sage Publications, 2008.
[64] P. F. Hultquist, *Numerical methods for engineers and computer scientists*: Menlo Park,Calif. : Benajamin/Cimmungs Pub. Co., 1988.
[65] G. B. P.Armitage, J.N.S. Matthews, *Statistical methods in medical research*, 4th ed ed.: Malden, MA : Blackwell Science, 2001.
[66] X. Yan, *Linear Regression Analysis*: World Scientific, 2009.
[67] J. D. F. Richard L. Burden, *Numerical Analysis*: Belmont,CA: Thomson Brooks/Cole, 2005.
[68] M. J. Crawley, *The R Book*: John Wiley & Sons, Ltd., 2007.
[69] D. S. Paulson, *Handbook of Regression and Modelling*: CRC, 2007.
[70] P. Bourke, "A Beginners Guide to Bitmaps." vol. 2011: http://paulbourke.net/dataformats/bitmaps/, 1993.
[71] J. Sanchez and M. P. Canton, *P C Graphics Handbook*: CRC Press, 2003.
[72] K. C. Chang, *Digital systems design with VHDL and synthesis : an integrated approach*. Los Alamitos,Calif: IEEE Computer Society, 1999.
[73] I. Grout, "Digital Systems Design with FPGAs and CPLDs," Newnes, 2008.
[74] C. H. Roth, *Digital systems design using VHDL*. Boston: PWS Pub.Co., 1998.
[75] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008,* pp. 1-58, 2008.
[76] D. E. Knuth, *The art of computer programming*, 3rd ed. vol. 2. Reading, Mass: Addison-wesley, 1997-1998.
[77] D. E. Knuth, *The art of computer programming*, 3rd ed. vol. 1. Reading, Mass: Addison-Wesley, 1997 -1998.
[78] M. Ali and K. Seok-Bum, "A Study on the Floating-Point Adder in FPGAS," in *Electrical and Computer Engineering, 2006. CCECE '06. Canadian Conference on*, 2006, pp. 86-89.
[79] J. D. Bruguera and T. Lang, "Leading-one prediction with concurrent position correction," *Computers, IEEE Transactions on,* vol. 48, pp. 1083-1097, 1999.
[80] A. Malik, C. Dongdong, C. Younhee, L. Moon, and K. Seok-Bum, "Design tradeoff analysis of floating-point adders in FPGAs," *Electrical and Computer Engineering, Canadian Journal of,* vol. 33, pp. 169-175, 2008.
[81] M. M. Ozbilen and M. Gok, "A multi-precision floating-point adder," in *Research in Microelectronics and Electronics, 2008. PRIME 2008. Ph.D.*, 2008, pp. 117-120.
[82] J. D. Bruguera and T. Lang, "Floating-point fused multiply-add: reduced latency for floating-point addition," in *Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on*, 2005, pp. 42-51.
[83] J. Chandran, A. Stojcevski, A. Zayegh, and N. Thinh, "Implementation of a colorimetric algorithm for portable blood gas analysis," in *Microelectronics (ICM), 2010 International Conference on*, pp. 411-414.
[84] I. S. Abu-Khater, A. Bellaouar, and M. I. Elmasry, "Circuit techniques for CMOS low-power high-performance multipliers," *Solid-State Circuits, IEEE Journal of,* vol. 31, pp. 1535-1546, 1996.
[85] P. V. Rao, C. Prasanna Raj P, and S. Ravi, "VLSI Design and Analysis of Multipliers for Low Power," in *Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP '09. Fifth International Conference on*, 2009, pp. 1354-1357.

[86] E. J. D. Kudithipudi "Implementation of Low Power Digital Multipliers Using 10 transistor Adder Blocks," *Journal of Low Power Electronics,* vol. 1, pp. 286-296, 2005.

[87] S. V. Siddamal, R. M. Banakar, and B. C. Jinaga, "Design of High-Speed Floating Point Multiplier," in *Electronic Design, Test and Applications, 2008. DELTA 2008. 4th IEEE International Symposium on*, 2008, pp. 285-289.

[88] D. D. Sumit Vaidya, "DELAY-POWER PERFORMANCE COMPARISON OF MULTIPLIERS IN VLSI CIRCUIT DESIGN," *International Journal of Computer Networks & Communications (IJCNC),* vol. 2, pp. 47-56, 2010.

[89] H. Thapliyal, H. R. Arabnia, and A. P. Vinod, "Combined Integer and Floating Point Multiplication Architecture(CIFM) for FPGAs and Its Reversible Logic Implementation," in *Circuits and Systems, 2006. MWSCAS '06. 49th IEEE International Midwest Symposium on*, 2006, pp. 438-442.

[90] D. Kudithipudi, P. Nair, and E. John, "On estimation and optimization of leakage power in CMOS multipliers," in *Circuits and Systems, 2007. MWSCAS 2007. 50th Midwest Symposium on*, 2007, pp. 859-862.

[91] M. A. Al-Qutayri, H. R. Barada, and A. Al-Kindi, "Comparison of Multipliers Architectures through Emulation and Handle-C FPGA Implementation," in *Computer Systems and Applications, 2006. IEEE International Conference on.*, 2006, pp. 240-247.

[92] P. C. H. Meier, R. A. Rutenbar, and L. R. Carley, "Exploring multiplier architecture and layout for low power," in *Custom Integrated Circuits Conference, 1996., Proceedings of the IEEE 1996*, 1996, pp. 513-516.

[93] B. Hickmann, A. Krioukov, M. Schulte, and M. Erle, "A parallel IEEE P754 decimal floating-point multiplier," in *Computer Design, 2007. ICCD 2007. 25th International Conference on*, 2007, pp. 296-303.

[94] C. N. M. P. Thangaraj, "Transmission Gate based High Performance Low Power Multiplier," *Journal of Applied Sciences,* vol. 10, pp. 3051-3059, 2010.

[95] M. Sjalander and P. Larsson-Edefors, "High-speed and low-power multipliers using the Baugh-Wooley algorithm and HPM reduction tree," in *Electronics, Circuits and Systems, 2008. ICECS 2008. 15th IEEE International Conference on*, 2008, pp. 33-36.

[96] N. T. Quach, N. Takagi, and M. J. Flynn, "Systematic IEEE rounding method for high-speed floating-point multipliers," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 12, pp. 511-521, 2004.

[97] K. E. Wires, M. J. Schulte, and J. E. Stine, "Combined IEEE compliant and truncated floating point multipliers for reduced power dissipation," in *Computer Design, 2001. ICCD 2001. Proceedings. 2001 International Conference on*, 2001, pp. 497-500.

[98] P. Karlstrom, A. Ehliar, and D. Liu, "High Performance, Low Latency FPGA based Floating Point Adder and Multiplier Units in a Virtex 4," in *Norchip Conference, 2006. 24th*, 2006, pp. 31-34.

[99] K. Shiann-Rong, W. Jiun-Ping, and H. Hua-Yi, "Variable-Latency Floating-Point Multipliers for Low-Power Applications," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 18, pp. 1493-1497.

[100] D. Tan, C. E. Lemonds, and M. J. Schulte, "Low-Power Multiple-Precision Iterative Floating-Point Multiplier with SIMD Support," *Computers, IEEE Transactions on,* vol. 58, pp. 175-187, 2009.

[101] K. E. Wires, M. J. Schulte, and J. E. Stine, "Variable-correction truncated floating point multipliers," in *Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference on*, 2000, pp. 1344-1348 vol.2.

[102] K. H. Abed and R. E. Siferd, "CMOS VLSI implementation of a low-power logarithmic converter," *Computers, IEEE Transactions on,* vol. 52, pp. 1421-1433, 2003.

[103] K. H. Abed and R. E. Siferd, "VLSI implementation of a low-power antilogarithmic converter," *Computers, IEEE Transactions on,* vol. 52, pp. 1221-1228, 2003.

[104] S. Paul, N. Jayakumar, and S. P. Khatri, "A Fast Hardware Approach for Approximate, Efficient Logarithm and Antilogarithm Computations," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 17, pp. 269-277, 2009.

[105] E. Kreyszig, *Advanced Engineerign Mathematics*, 9th ed. Hoboken,NJ: John Wiley, 2006.

[106] E. W. Weisstein, "Taylor Series." vol. 2011, F. M.-.-A. W. W. Resource, Ed.: http://mathworld.wolfram.com/TaylorSeries.html.

[107] "Product and Services." vol. 2011: http://www.abbottpointofcare.com/Products-and-Services.aspx.

[108] "OPTI Medical Systems OPTI® CCA-TS Blood Gas and Electrolyte Analyzer." vol. 2011: http://www.optimedical.com/products/opti/opti_cca_touch_over.htm.

[109] "Agilent HP 5972 Mass Spectrometer (HP5972)." vol. 2011: http://www.gmi-inc.com/Agilent-HP-5972-Mass-Spectrometer.html.

Appendix A: Trends for the colours against the pH

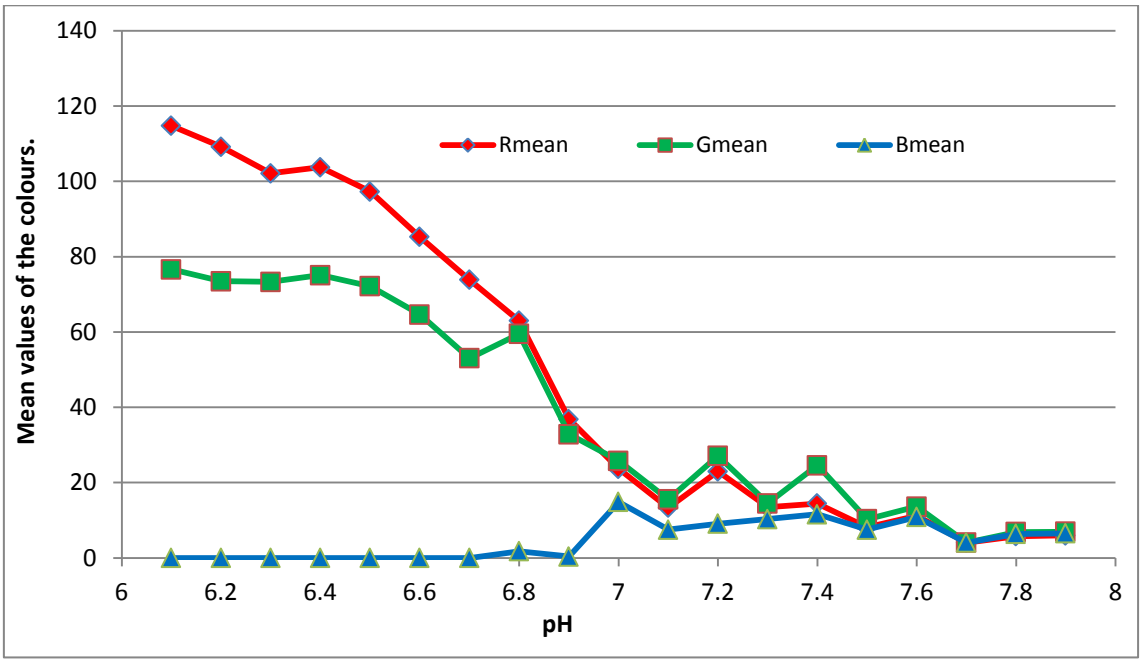


Figure: Plot of the RGB coloursvs pH 01/07/08

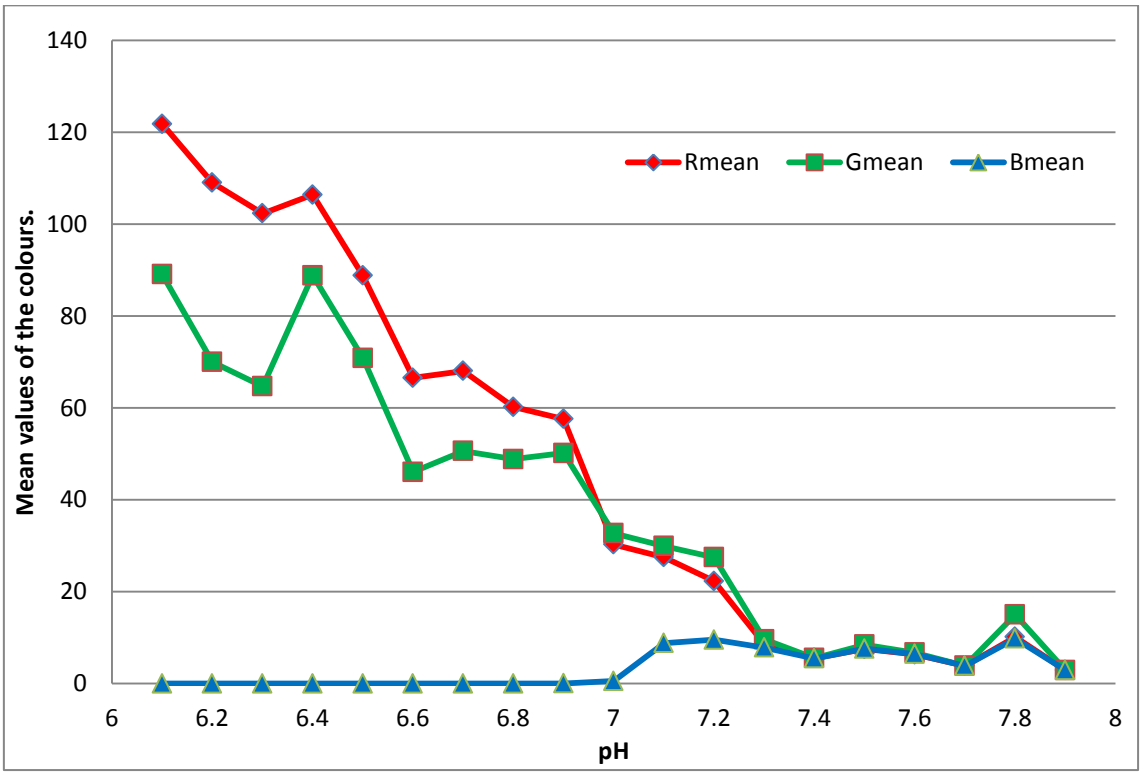


Figure: Plot of the RGB coloursvs pH 03/07/08

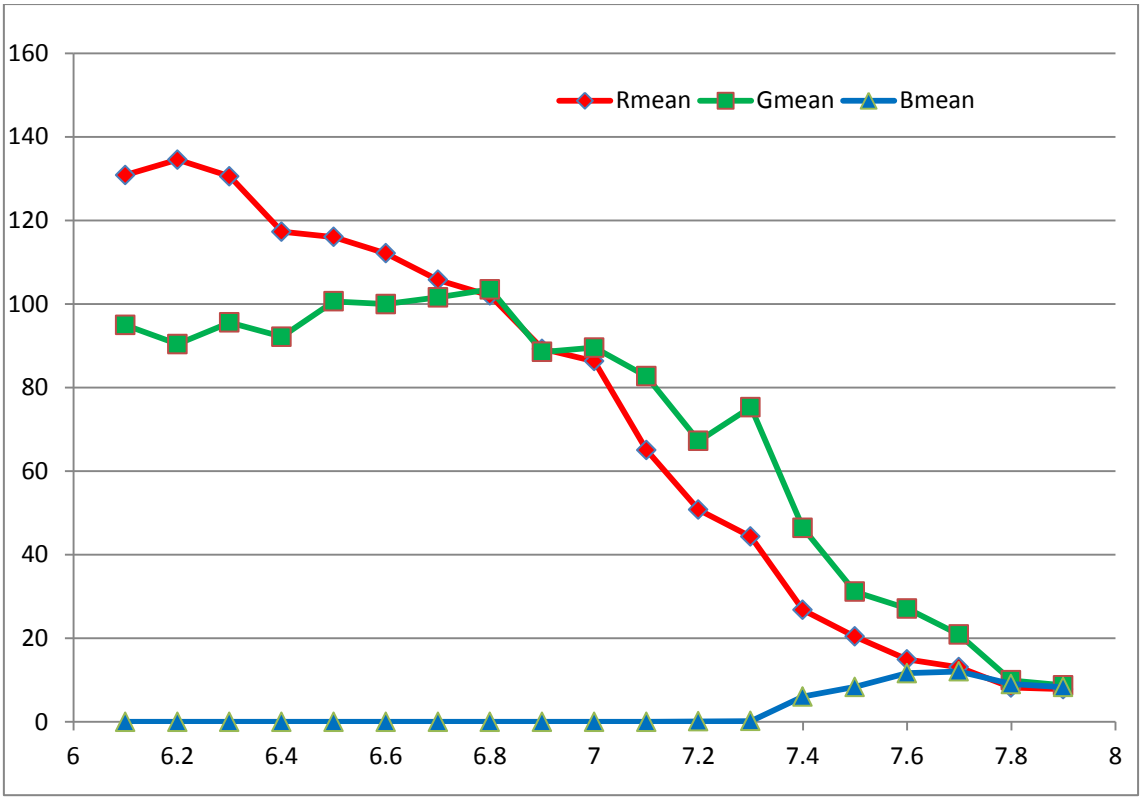


Figure: Plot of the RGB coloursvs pH 07/07/08

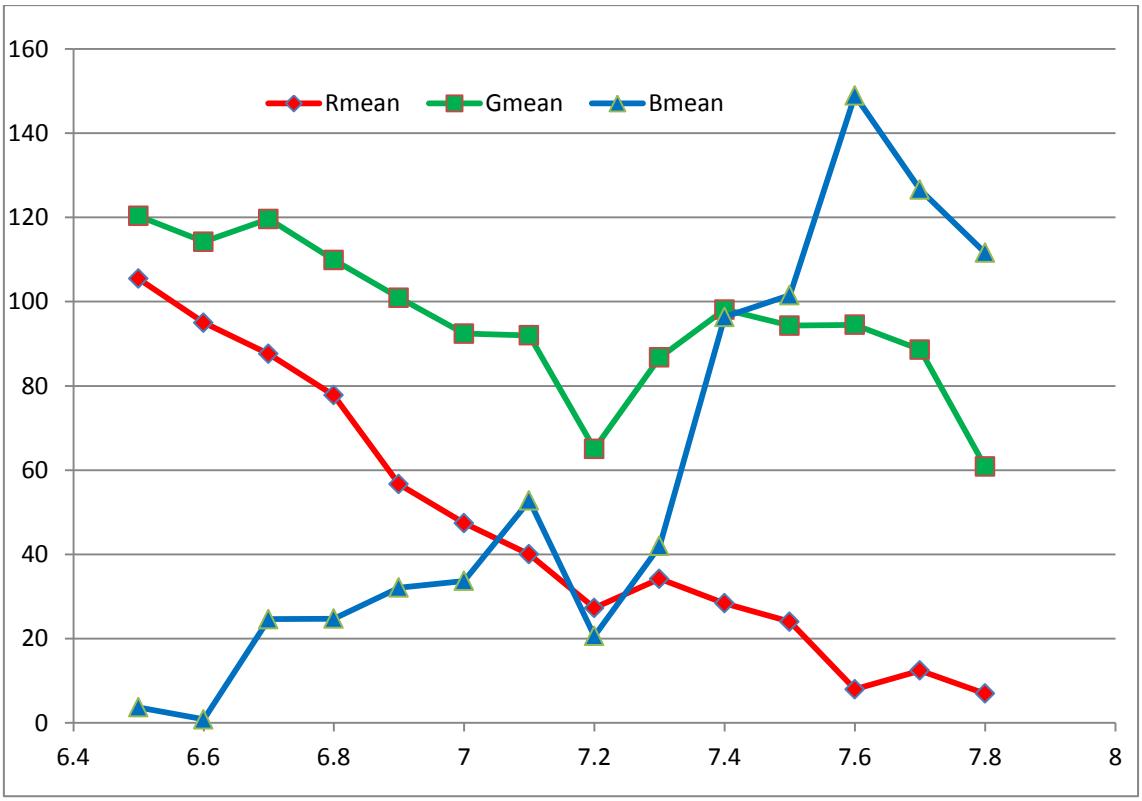


Figure: Plot of the RGB coloursvs pH 22/07/08

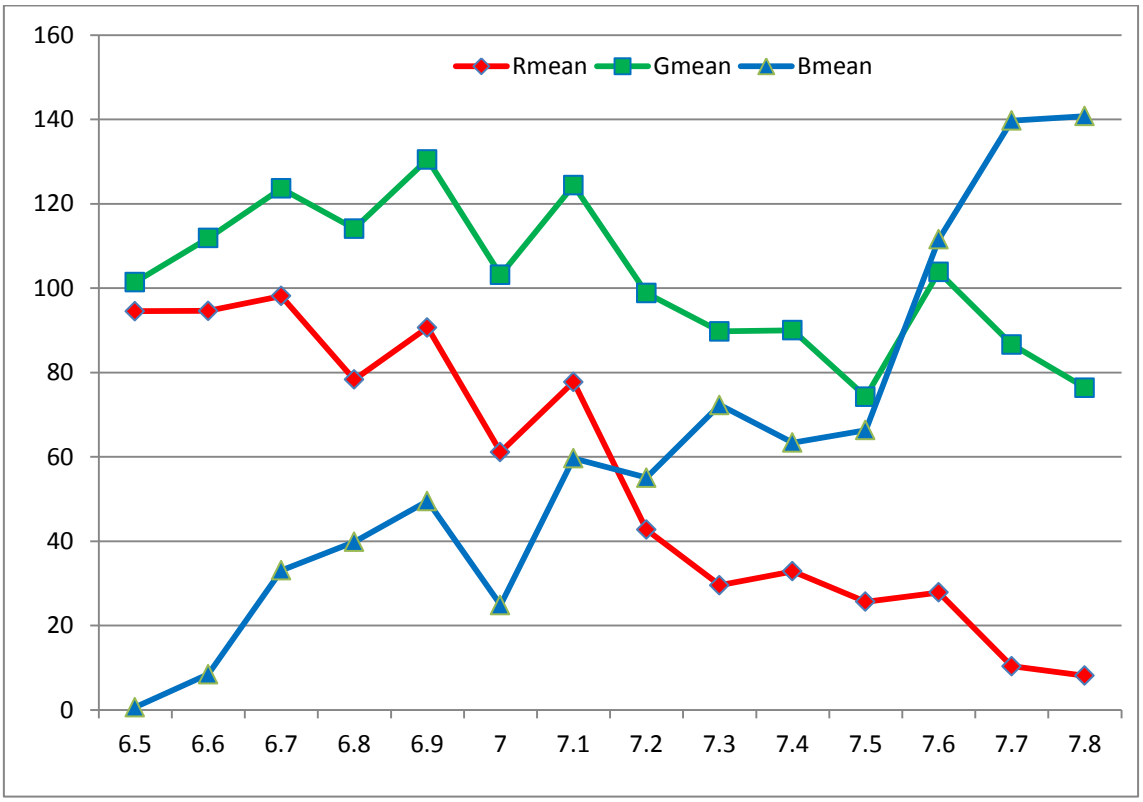


Figure: Plot of the RGB coloursvs pH 05/07/08

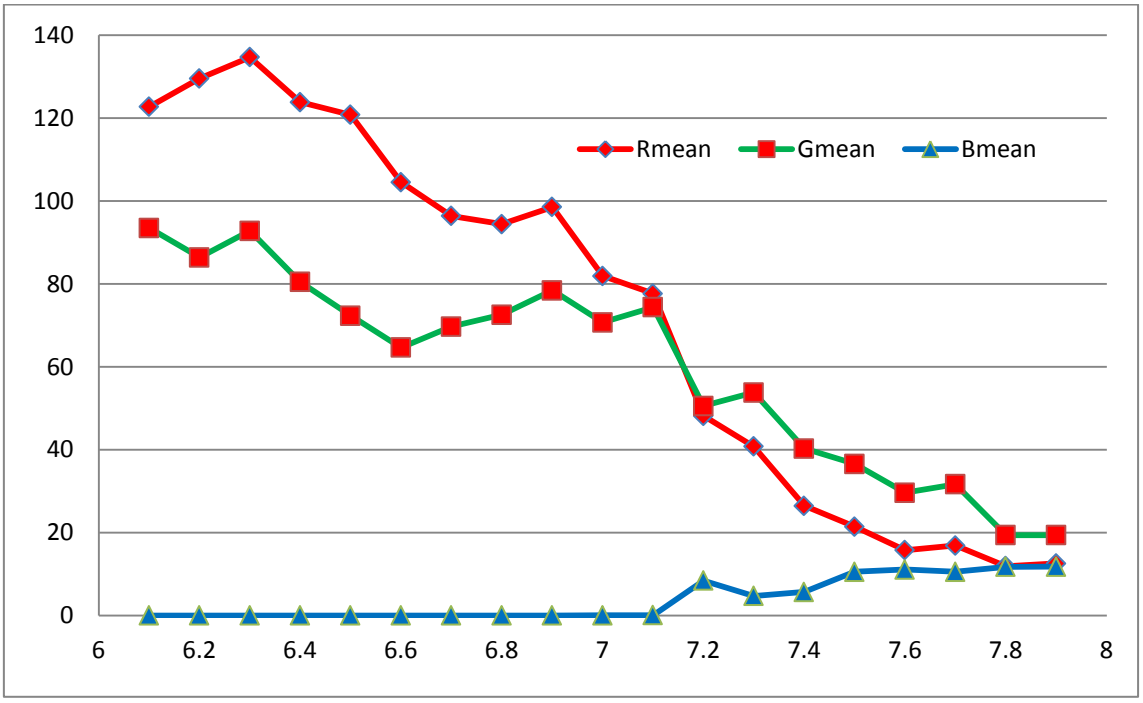


Figure: Plot of the RGB coloursvs pH 05/07/08

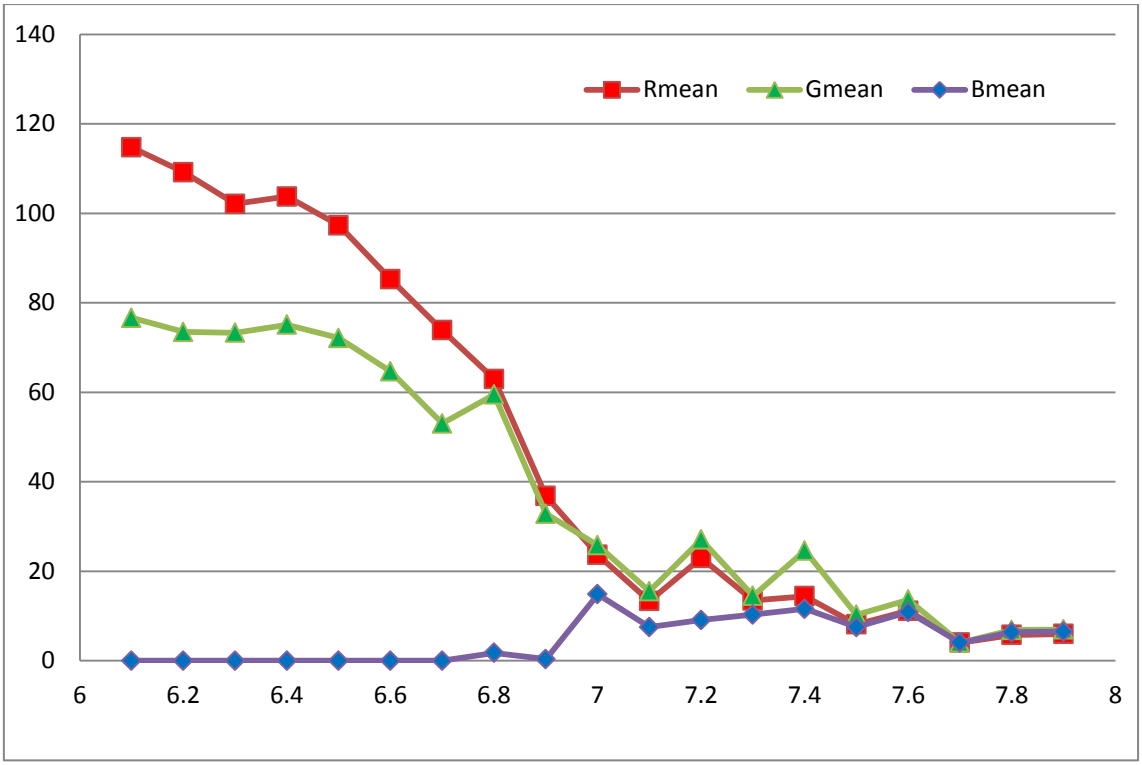


Figure: Plot of the RGB coloursvs pH 30/06/08

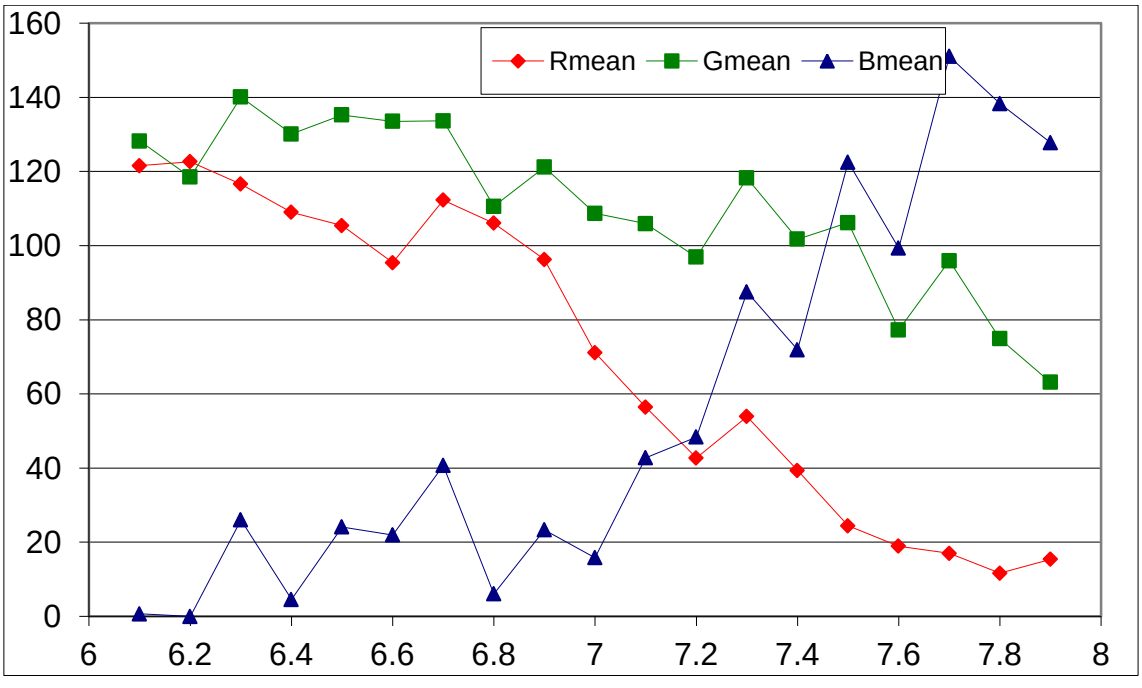


Figure: Plot of the RGB coloursvs pH 23/11/08

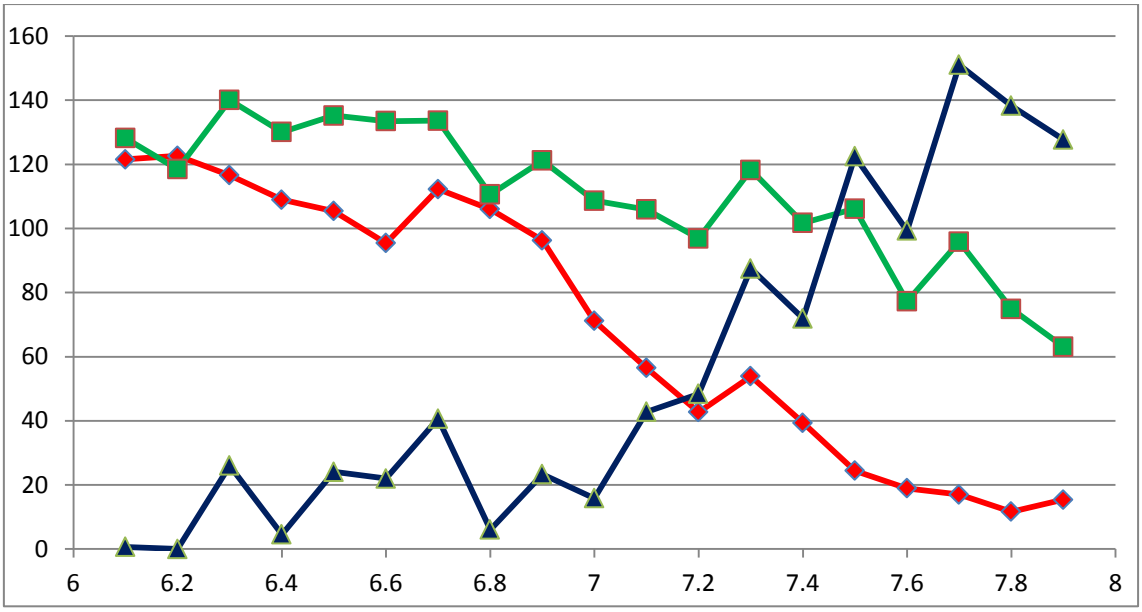


Figure: Plot of the RGB coloursvs pH 20/12/08

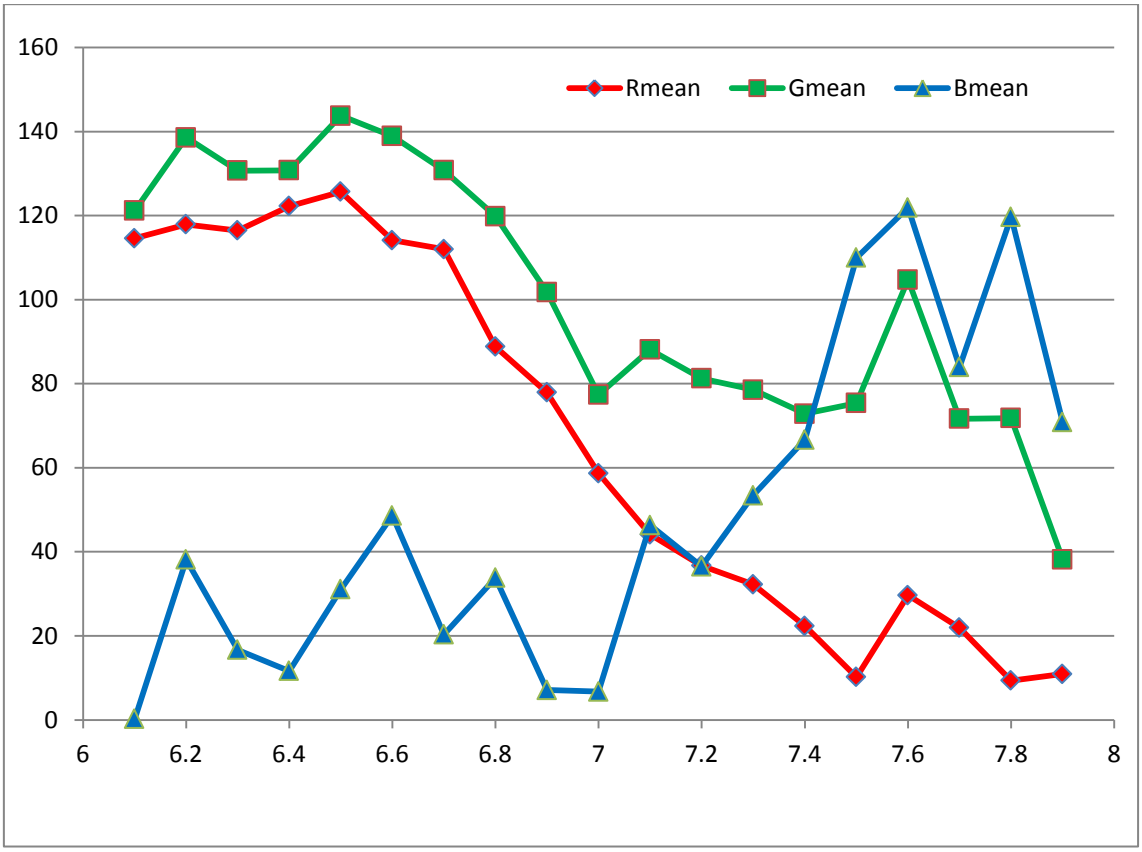


Figure: Plot of the RGB coloursvs pH 23/12/08

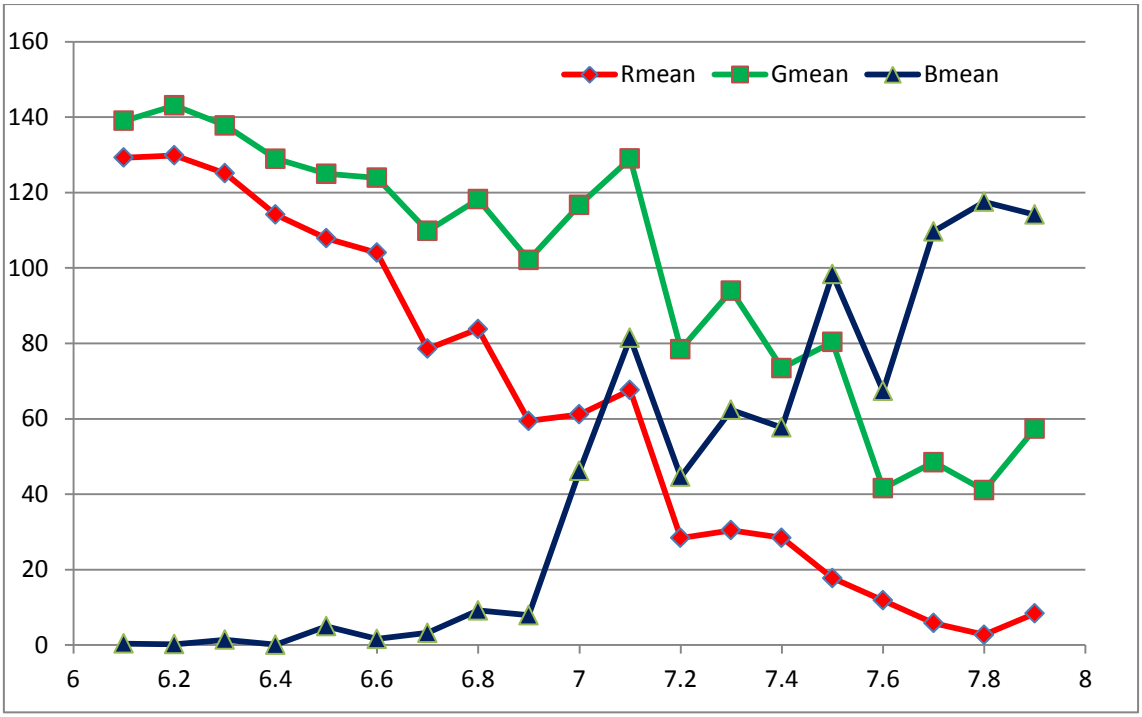


Figure: Plot of the RGB coloursvs pH 27/12/08

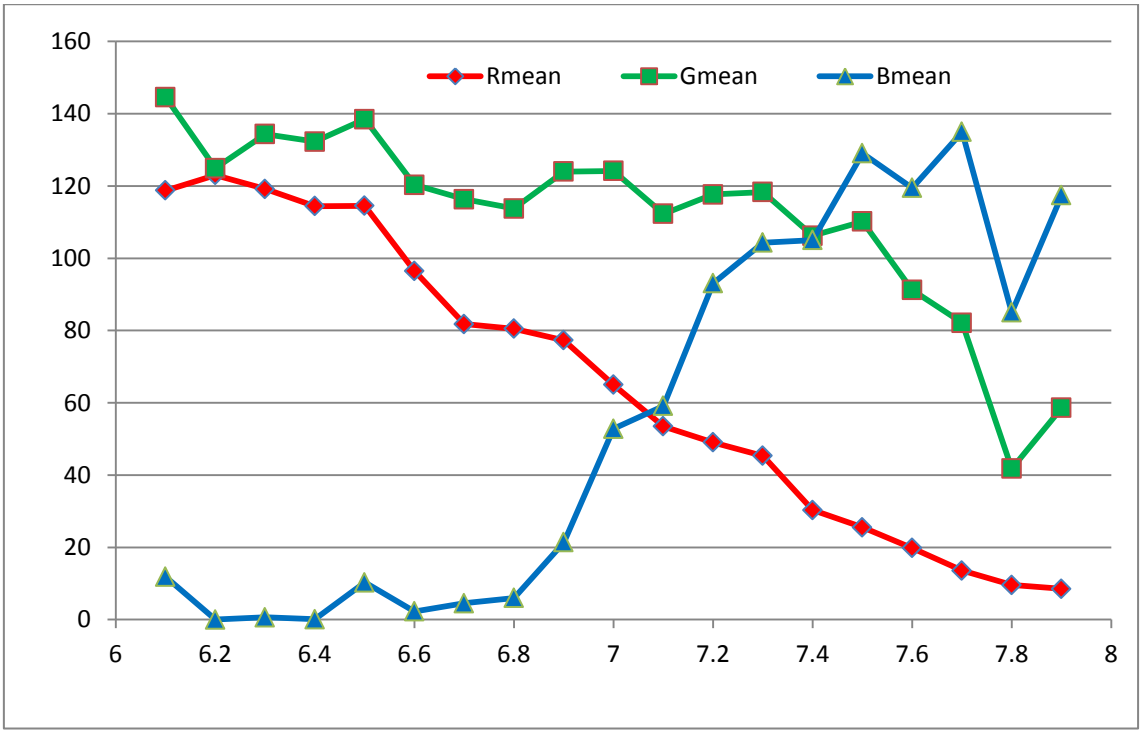


Figure: Plot of the RGB coloursvs pH 29/12/08

Appendix B: VHDL Code for the image based colorimetric analyser

```
--
*********************************************************************
*******
--* File Name  : cph_analyzer.vhd
*
--* Designer   : Jaideep Chandran
*
--* Date       : 10/03/2011
*
--* Description: Colour Base PH Analyzer of Blood.
*
--
*********************************************************************
*******
--
*********************************************************************
*******
--* Include Libraries
*
--
*********************************************************************
*******
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--
*********************************************************************
*******
--* Entity Decleration, I/O ports
*
--
*********************************************************************
*******
entity cph_analyzer is
    Port (
           hclk            : in  STD_LOGIC;
           hreset_n        : in  STD_LOGIC;
           haddr_s         : in  STD_LOGIC_VECTOR (31 downto 0);
           hwdata_s        : in  STD_LOGIC_VECTOR (31 downto 0);
           htrans_s        : in  STD_LOGIC_VECTOR (1 downto 0);
           hsel_s          : in  STD_LOGIC;
           hsize_s         : in  STD_LOGIC_VECTOR (2 downto 0);
           hready_in       : in  STD_LOGIC;
           hwrite_s        : in  STD_LOGIC;
           bmp_mem_data_in : in  STD_LOGIC_VECTOR (7 downto 0);
           hrdata_s        : out STD_LOGIC_VECTOR (31 downto 0);
           hready_out      : out STD_LOGIC;
           hresp_s         : out STD_LOGIC_VECTOR (1 downto 0);
           bmp_mem_re      : out STD_LOGIC;
           bmp_mem_addr    : out STD_LOGIC_VECTOR (31 downto 0);
           bmp_mem_data_out: out STD_LOGIC_VECTOR (7 downto 0));
end cph_analyzer;

architecture Behavioral of cph_analyzer is
```

```
--
************************************************************************
*******
--* Signal Decleration
*
--
************************************************************************
*******
   signal  bmp_msaddr_ph   : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_msaddr_co2  : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_icext_strt  : STD_LOGIC;
   signal  bmp_iph_ana     : STD_LOGIC;
   signal  bmp_ipc02_ana   : STD_LOGIC;
   signal  bmp_phcext_dn   : STD_LOGIC;
   signal  bmp_pco2cext_dn : STD_LOGIC;
   signal  bmp_ram_we      : STD_LOGIC;
   signal  bmp_ram_addr    : STD_LOGIC_VECTOR (7 downto 0);
   signal  bmp_ram_datain  : STD_LOGIC_VECTOR (7 downto 0);
   signal  bmp_strt_val    : STD_LOGIC;
   signal  bmp_sr_sav      : STD_LOGIC;
   signal  bmp_hid_reg_ph  : STD_LOGIC_VECTOR (15 downto 0);
   signal  bmp_fs_reg_ph   : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_fhi_reg_ph  : STD_LOGIC_VECTOR (7 downto 0);
   signal  bmp_ih_reg_ph   : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_iw_reg_ph   : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_pw_reg_ph   : STD_LOGIC_VECTOR (7 downto 0);
   signal  bmp_hid_reg_co2 : STD_LOGIC_VECTOR (15 downto 0);
   signal  bmp_fs_reg_co2  : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_fhi_reg_co2 : STD_LOGIC_VECTOR (7 downto 0);
   signal  bmp_ih_reg_co2  : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_iw_reg_co2  : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_pw_reg_co2  : STD_LOGIC_VECTOR (7 downto 0);
   signal  bmp_red_mv      : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_red_mf      : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_iph_sext    : STD_LOGIC;
   signal  bmp_phext_dn    : STD_LOGIC;
   signal  cph_ph_val      : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_ipco2_sext  : STD_LOGIC;
   signal  bmp_iph_val     : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_pco2ext_dn  : STD_LOGIC;
   signal  cph_co2_val     : STD_LOGIC_VECTOR (31 downto 0);
   signal  bmp_ca_dn       : STD_LOGIC;
   signal  bmp_cstart      : STD_LOGIC;
   signal  bmp_pco2phext_dn: STD_LOGIC;
      signal  bmp_ram_dataout : STD_LOGIC_VECTOR (7 downto 0);

--
************************************************************************
*******
--* Component Decleration
*
--
************************************************************************
*******
  component cph_analyzer_csync
    Port (
           hclk             : in  STD_LOGIC;
           hreset_n         : in  STD_LOGIC;
           bmp_cstart       : in  STD_LOGIC;
           bmp_phcext_dn    : in  STD_LOGIC;
           bmp_phext_dn     : in  STD_LOGIC;
```

```
        bmp_pco2cext_dn  : in  STD_LOGIC;
        bmp_pco2phext_dn : in  STD_LOGIC;
        bmp_pco2ext_dn   : in  STD_LOGIC;
        bmp_iph_sext     : out STD_LOGIC;
        bmp_ipco2_sext   : out STD_LOGIC;
        bmp_icext_strt   : out STD_LOGIC;
        bmp_iph_ana      : out STD_LOGIC;
        bmp_ipc02_ana    : out STD_LOGIC;
        bmp_ca_dn        : out STD_LOGIC);

end component;

component cph_analyzer_reg
  Port (
        hclk            : in  STD_LOGIC;
        hreset_n        : in  STD_LOGIC;
        haddr_s         : in  STD_LOGIC_VECTOR (31 downto 0);
        hwdata_s        : in  STD_LOGIC_VECTOR (31 downto 0);
        htrans_s        : in  STD_LOGIC_VECTOR (1 downto 0);
        hsel_s          : in  STD_LOGIC;
        hsize_s         : in  STD_LOGIC_VECTOR (2 downto 0);
        hready_in       : in  STD_LOGIC;
        hwrite_s        : in  STD_LOGIC;
        bmp_ca_dn       : in  STD_LOGIC;
        bmp_hid_reg_ph  : in  STD_LOGIC_VECTOR (15 downto 0);
        bmp_fs_reg_ph   : in  STD_LOGIC_VECTOR (31 downto 0);
        bmp_fhi_reg_ph  : in  STD_LOGIC_VECTOR (7 downto 0);
        bmp_ih_reg_ph   : in  STD_LOGIC_VECTOR (31 downto 0);
        bmp_iw_reg_ph   : in  STD_LOGIC_VECTOR (31 downto 0);
        bmp_pw_reg_ph   : in  STD_LOGIC_VECTOR (7 downto 0);
        bmp_hid_reg_co2 : in  STD_LOGIC_VECTOR (15 downto 0);
        bmp_fs_reg_co2  : in  STD_LOGIC_VECTOR (31 downto 0);
        bmp_fhi_reg_co2 : in  STD_LOGIC_VECTOR (7 downto 0);
        bmp_ih_reg_co2  : in  STD_LOGIC_VECTOR (31 downto 0);
        bmp_iw_reg_co2  : in  STD_LOGIC_VECTOR (31 downto 0);
        bmp_pw_reg_co2  : in  STD_LOGIC_VECTOR (7 downto 0);
        cph_ph_val      : in  STD_LOGIC_VECTOR (31 downto 0);
        cph_co2_val     : in  STD_LOGIC_VECTOR (31 downto 0);
        bmp_cstart      : out STD_LOGIC;
        bmp_msaddr_ph   : out STD_LOGIC_VECTOR (31 downto 0);
        bmp_msaddr_co2  : out STD_LOGIC_VECTOR (31 downto 0);
        hready_out      : out STD_LOGIC;
        hresp_s         : out STD_LOGIC_VECTOR (1 downto 0);
        hrdata_s        : out STD_LOGIC_VECTOR (31 downto 0));

end component;
component cph_analyzer_phcal
  Port (
        hclk             : in  STD_LOGIC;
        hreset_n         : in  STD_LOGIC;
        bmp_iph_ana      : in STD_LOGIC;
        bmp_ipc02_ana    : in STD_LOGIC;
        bmp_iph_sext     : in  STD_LOGIC;
        bmp_red_mv       : in  STD_LOGIC_VECTOR (31 downto 0);
        bmp_red_mf       : in  STD_LOGIC_VECTOR (31 downto 0);
        bmp_phext_dn     : out STD_LOGIC;
        bmp_pco2phext_dn : out STD_LOGIC;
        cph_ph_val       : out STD_LOGIC_VECTOR (31 downto 0);
        bmp_iph_val      : out STD_LOGIC_VECTOR (31 downto 0));
end component;
```

```
component cph_analyzer_co2cal
  Port (
         hclk             : in  STD_LOGIC;
         hreset_n         : in  STD_LOGIC;
         bmp_ipco2_sext   : in  STD_LOGIC;
         bmp_iph_val      : in  STD_LOGIC_VECTOR (31 downto 0);
         bmp_pco2ext_dn   : out STD_LOGIC;
         cph_co2_val      : out STD_LOGIC_VECTOR (31 downto 0));
end component;

component cph_analyzer_ctbl
  Port (
            hclk            : in  STD_LOGIC;
         bmp_ram_we      : in  STD_LOGIC;
         bmp_ram_addr    : in  STD_LOGIC_VECTOR (7 downto 0);
         bmp_ram_datain  : in  STD_LOGIC_VECTOR (7 downto 0);
         bmp_ram_dataout : out STD_LOGIC_VECTOR (7 downto 0));
end component;

component cph_analyzer_cext
  Port (
            hclk            : in  STD_LOGIC;
         hreset_n        : in  STD_LOGIC;
         bmp_strt_val    : in  STD_LOGIC;
         bmp_sr_sav      : in  STD_LOGIC;
         bmp_ram_dataout : in  STD_LOGIC_VECTOR (7 downto 0);
         bmp_red_mv      : out STD_LOGIC_VECTOR (31 downto 0);
         bmp_red_mf      : out STD_LOGIC_VECTOR (31 downto 0));
end component;

component cph_analyzer_ictl
  Port (
         hclk             : in STD_LOGIC;
         hreset_n         : in STD_LOGIC;
         bmp_msaddr_ph    : in STD_LOGIC_VECTOR (31 downto 0);
         bmp_msaddr_co2   : in STD_LOGIC_VECTOR (31 downto 0);
         bmp_mem_data_in  : in STD_LOGIC_VECTOR (7 downto 0);
         bmp_icext_strt   : in STD_LOGIC;
         bmp_iph_ana      : in STD_LOGIC;
         bmp_ipc02_ana    : in STD_LOGIC;
         bmp_phcext_dn    : out  STD_LOGIC;
         bmp_pco2cext_dn  : out  STD_LOGIC;
         bmp_mem_re       : out STD_LOGIC;
         bmp_mem_addr     : out STD_LOGIC_VECTOR (31 downto 0);
         bmp_ram_we       : out STD_LOGIC;
         bmp_ram_addr     : out STD_LOGIC_VECTOR (7 downto 0);
         bmp_ram_datain   : out STD_LOGIC_VECTOR (7 downto 0);
         bmp_strt_val     : out STD_LOGIC;
         bmp_sr_sav       : out STD_LOGIC;
         bmp_hid_reg_ph   : out STD_LOGIC_VECTOR (15 downto 0);
         bmp_fs_reg_ph    : out STD_LOGIC_VECTOR (31 downto 0);
         bmp_fhi_reg_ph   : out STD_LOGIC_VECTOR (7 downto 0);
         bmp_ih_reg_ph    : out STD_LOGIC_VECTOR (31 downto 0);
         bmp_iw_reg_ph    : out STD_LOGIC_VECTOR (31 downto 0);
         bmp_pw_reg_ph    : out STD_LOGIC_VECTOR (7 downto 0);
         bmp_hid_reg_co2  : out STD_LOGIC_VECTOR (15 downto 0);
         bmp_fs_reg_co2   : out STD_LOGIC_VECTOR (31 downto 0);
         bmp_fhi_reg_co2  : out STD_LOGIC_VECTOR (7 downto 0);
         bmp_ih_reg_co2   : out STD_LOGIC_VECTOR (31 downto 0);
         bmp_iw_reg_co2   : out STD_LOGIC_VECTOR (31 downto 0);
         bmp_pw_reg_co2   : out STD_LOGIC_VECTOR (7 downto 0));
```

```
  end component;

begin
    bmp_mem_data_out <= (others => '0');
--
*************************************************************************
*******
--* Module Instantiation
*
--
*************************************************************************
*******
  cph_csync:
    cph_analyzer_csync
    Port map (
               hclk             => hclk,
               hreset_n         => hreset_n,
               bmp_cstart       => bmp_cstart,
               bmp_phcext_dn    => bmp_phcext_dn,
               bmp_phext_dn     => bmp_phext_dn,
               bmp_pco2cext_dn  => bmp_pco2cext_dn,
               bmp_pco2phext_dn => bmp_pco2phext_dn,
               bmp_pco2ext_dn   => bmp_pco2ext_dn,
               bmp_iph_sext     => bmp_iph_sext,
               bmp_ipco2_sext   => bmp_ipco2_sext,
               bmp_icext_strt   => bmp_icext_strt,
               bmp_iph_ana      => bmp_iph_ana,
               bmp_ipc02_ana    => bmp_ipc02_ana,
               bmp_ca_dn        => bmp_ca_dn
              );

  cph_reg:
    cph_analyzer_reg
    Port map (
               hclk             => hclk,
               hreset_n         => hreset_n,
               haddr_s          => haddr_s,
               hwdata_s         => hwdata_s,
               htrans_s         => htrans_s,
               hsel_s           => hsel_s,
               hsize_s          => hsize_s,
               hready_in        => hready_in,
               hwrite_s         => hwrite_s,
               bmp_ca_dn        => bmp_ca_dn,
               bmp_hid_reg_ph   => bmp_hid_reg_ph,
               bmp_fs_reg_ph    => bmp_fs_reg_ph,
               bmp_fhi_reg_ph   => bmp_fhi_reg_ph,
               bmp_ih_reg_ph    => bmp_ih_reg_ph,
               bmp_iw_reg_ph    => bmp_iw_reg_ph,
               bmp_pw_reg_ph    => bmp_pw_reg_ph,
               bmp_hid_reg_co2  => bmp_hid_reg_co2,
               bmp_fs_reg_co2   => bmp_fs_reg_co2,
               bmp_fhi_reg_co2  => bmp_fhi_reg_co2,
               bmp_ih_reg_co2   => bmp_ih_reg_co2,
               bmp_iw_reg_co2   => bmp_iw_reg_co2,
               bmp_pw_reg_co2   => bmp_pw_reg_co2,
               cph_ph_val       => cph_ph_val,
               cph_co2_val      => cph_co2_val,
               bmp_cstart       => bmp_cstart,
               bmp_msaddr_ph    => bmp_msaddr_ph,
               bmp_msaddr_co2   => bmp_msaddr_co2,
```

```
            hready_out       => hready_out,
            hresp_s          => hresp_s,
            hrdata_s         => hrdata_s
           );

cph_phcal:
  cph_analyzer_phcal
  Port map (
            hclk             => hclk,
            hreset_n         => hreset_n,
            bmp_iph_ana      => bmp_iph_ana,
            bmp_ipc02_ana    => bmp_ipc02_ana,

            bmp_iph_sext     => bmp_iph_sext,
            bmp_red_mv       => bmp_red_mv,
            bmp_red_mf       => bmp_red_mf,
            bmp_phext_dn     => bmp_phext_dn,
            bmp_pco2phext_dn => bmp_pco2phext_dn,

            cph_ph_val       => cph_ph_val,
            bmp_iph_val      => bmp_iph_val
           );

cph_co2cal:
  cph_analyzer_co2cal
  Port map (
            hclk             => hclk,
            hreset_n         => hreset_n,
            bmp_ipco2_sext   => bmp_ipco2_sext,
            bmp_iph_val      => bmp_iph_val,
            bmp_pco2ext_dn   => bmp_pco2ext_dn,
            cph_co2_val      => cph_co2_val
           );

cph_ctbl:
  cph_analyzer_ctbl
  Port map (
            hclk             => hclk,
            bmp_ram_we       => bmp_ram_we,
            bmp_ram_addr     => bmp_ram_addr,
            bmp_ram_datain   => bmp_ram_datain,
            bmp_ram_dataout  => bmp_ram_dataout
           );

cph_cext:
cph_analyzer_cext
  Port map (
            hclk             => hclk,
            hreset_n         => hreset_n,
            bmp_strt_val     => bmp_strt_val,
            bmp_sr_sav       => bmp_sr_sav,
            bmp_ram_dataout  => bmp_ram_dataout,
            bmp_red_mv       => bmp_red_mv,
            bmp_red_mf       => bmp_red_mf
           );

cph_ictl:
  cph_analyzer_ictl
  Port map (
            hclk             => hclk,
            hreset_n         => hreset_n,
```

```
              bmp_msaddr_ph    => bmp_msaddr_ph,
              bmp_msaddr_co2   => bmp_msaddr_co2,
              bmp_mem_data_in  => bmp_mem_data_in,
              bmp_icext_strt   => bmp_icext_strt,
              bmp_iph_ana      => bmp_iph_ana,
              bmp_ipc02_ana    => bmp_ipc02_ana,
              bmp_phcext_dn    => bmp_phcext_dn,
              bmp_pco2cext_dn  => bmp_pco2cext_dn,
              bmp_mem_re       => bmp_mem_re,
              bmp_mem_addr     => bmp_mem_addr,
              bmp_ram_we       => bmp_ram_we,
              bmp_ram_addr     => bmp_ram_addr,
              bmp_ram_datain   => bmp_ram_datain,
              bmp_strt_val     => bmp_strt_val,
              bmp_sr_sav       => bmp_sr_sav,
              bmp_hid_reg_ph   => bmp_hid_reg_ph,
              bmp_fs_reg_ph    => bmp_fs_reg_ph,
              bmp_fhi_reg_ph   => bmp_fhi_reg_ph,
              bmp_ih_reg_ph    => bmp_ih_reg_ph,
              bmp_iw_reg_ph    => bmp_iw_reg_ph,
              bmp_pw_reg_ph    => bmp_pw_reg_ph,
              bmp_hid_reg_co2  => bmp_hid_reg_co2,
              bmp_fs_reg_co2   => bmp_fs_reg_co2,
              bmp_fhi_reg_co2  => bmp_fhi_reg_co2,
              bmp_ih_reg_co2   => bmp_ih_reg_co2,
              bmp_iw_reg_co2   => bmp_iw_reg_co2,
              bmp_pw_reg_co2   => bmp_pw_reg_co2
             );
end Behavioral;


--
**********************************************************************
*******
--* File Name  : cph_analyzer_add.vhd
*
--* Designer   : Jaideep Chandran
*
--* Date       : 10/03/2011
*
--* Description: Floating Point Multiplier.
*
--
**********************************************************************
*******
--
**********************************************************************
*******
--* Include Libraries
*
--
**********************************************************************
*******
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--
**********************************************************************
*******
```

```
--* Entity Decleration, I/O ports
*
--
**************************************************************************
*******
entity cph_analyzer_add is
  port (op_a,op_b : in std_logic_vector(31 downto 0);
        hreset_n,hclk:in std_logic;
              result: out std_logic_vector(31 downto 0));
end cph_analyzer_add;

--
**************************************************************************
*******
--* Main Block Logic
*
--
**************************************************************************
*******
architecture Behavioral of cph_analyzer_add is

signal x_man,y_man: std_logic_vector(23 downto 0);
signal z_man : unsigned(24 downto 0);
--signal z_man1 : std_logic_vector(23 downto 0);
signal x_exp,y_exp,z_exp,z_exp1 : std_logic_vector(7 downto 0);
signal x_sign,y_sign,z_sign : std_logic;
--signal compare_x_y : std_logic_vector(1 downto 0);
signal bigger_x :std_logic;
signal exp_diff_x_y, exp_diff_y_x : std_logic_vector(8 downto 0);
signal diff_exp : std_logic_vector(8 downto 0);
signal man_big : std_logic_vector (23 downto 0);
signal too_small : std_logic;
signal man_small : std_logic_vector(23 downto 0);
signal man_add : std_logic_vector(24 downto 0);
signal man_shifted_small : UNSIGNED(23 downto 0);
signal shift_check_positive,adbit1  : std_logic;
--signal adbit2 : std_logic_vector(1 downto 0);

begin
x_sign <= op_a(31);
y_sign <= op_b(31);
x_exp <= op_a(30 downto 23);
y_exp <= op_b(30 downto 23);
x_man <= '1' & op_a(22 downto 0);
y_man <= '1' & op_b(22 downto 0);
adbit1 <= '0';


-- diff_exp : it contains the counter for shifting
--man_exp : it contains the greater mantissa
-- compare_x_y : it is used to compare x and y for bigger number
-- bigger_x: it returns boolean expression whether x is bigger than y
or not
P1:process(op_a,op_b,hreset_n,hclk)
--variable diff_exp: signed(8 downto 0);
--variable man_big,man_less : unsigned(x_man'range);
--variable result_exp :SIGNED (8 downto 0);
--variable result_man :UNSIGNED (22 downto 0);
--variable flag : BOOLEAN;
variable shift_check_negative: std_logic_vector(4 downto 0);
--variable i: integer := 23;
```

```
constant one: unsigned(1 downto 0):= "01";
begin
if hreset_n = '0' then
  result <= (others => '0');
--  test<=(others => '0');
elsif (hclk'event and hclk= '1') then

exp_diff_x_y <= ('0' & x_exp) - ('0' & y_exp);
exp_diff_y_x <= ('0' & y_exp) - ('0' & x_exp);

--check which exponent is bigger
--compare_x_y <= x_exp(7) & y_exp(7);
--if compare_x_y = "00" then
--  bigger_x <= not(exp_diff_x_y(8));
--elsif compare_x_y = "01" then
--  bigger_x <= '0';
--elsif compare_x_y = "10" then
--  bigger_x <= '1';
--else
--  bigger_x <= not (exp_diff_x_y(8));
--end if;


-- find out whether the shifting counter is not too high i.e. more
than 23
--too_small <= diff_exp(8) or diff_exp(7) or diff_exp(6) or
diff_exp(5);
-- man_small: it contains the smaller mantissa with its right shifted
version with counter diff_Exp;
if x_exp = y_exp then
  if x_man > y_man then
  z_exp <= x_exp;
  diff_exp <= (others => '0');
  man_big <= x_man;
  man_small <= y_man;
  bigger_x <= '1';
  else
  z_exp <=  y_exp;
  diff_exp <= (others => '0');
  man_big <= y_man;
  man_small <= x_man;
  bigger_x <='0';
  end if;
elsif x_exp > y_exp then
  z_exp <= x_exp;
  diff_exp <= exp_diff_x_y;
  man_big <= x_man;
  man_small <= y_man;
  bigger_x <= '1';
else
  z_exp <=  y_exp;
  diff_exp <= exp_diff_y_x;
  man_big <= y_man;
  man_small <= x_man;
  bigger_x <='0';
end if;


-- find out whether the shifting counter is not too high i.e. more
than 23
too_small <= diff_exp(8) or diff_exp(7) or diff_exp(6) or diff_exp(5);
```

```
if(too_small= '1')  then
    man_shifted_small <= (others => '0');
else
    man_shifted_small <= shr(unsigned(man_small),unsigned(diff_exp));
end if;


-- now add the two mantissa smaller one and the bigger one and also
check the sign bit whether both are same or not.
if (x_sign = y_sign) then
  result(31) <= x_sign;
  man_add <= (adbit1 & man_big) + (adbit1 &
std_logic_vector(man_shifted_small));
  if ( man_add(24) = '1' ) then
    z_man <= shr(unsigned(man_add),one);
    result(30 downto 23) <= z_exp + "01";
  else
    z_man <= unsigned(man_add);
    result(30 downto 23) <= z_exp;
  end if;
  --result(22 downto 0) <= std_logic_vector(z_man(22 downto 0));
else
  if (bigger_x = '1') then
    result(31) <= x_sign;
  else
    result(31) <= y_sign;
  end if;
  man_add <= (adbit1 & man_big) - (adbit1 &
std_logic_vector(man_shifted_small));

-- check for the msb's position i.e. first '1' from left;

--  L1:while i >= 0  loop
--  if man_add(i) = '0' then
--     shift_check_negative := shift_check_negative + "01";
--      i := i-1;
--  else
--  exit L1; --when man_add(i)='1';
--  end if;
--  end loop L1;

--P2: process(man_add)
--begin
shift_check_negative := (others => '0');
loop1: for i in 23 downto 0 loop
 --wait on man_add;
 if (man_add(i) = '0') then
    shift_check_negative := shift_check_negative + 1;
 else
     exit loop1;
 end if;
 end loop loop1;
 --end process P2;
-- test <= shift_check_negative;
 z_man   <= shl(unsigned(man_add),unsigned(shift_check_negative));
  z_exp1  <= z_exp - shift_check_negative;
  result(30 downto 23)<= z_exp1;
 end if;
  result(22 downto 0)<= std_logic_vector(z_man(22 downto 0));
end if;
```

```
--wait on op_a,op_b,clk,hreset_n;
end process P1;

end Behavioral;

--
*************************************************************************
*******
--* File Name  : cph_analyzer_cext.vhd
*
--* Designer   : Jaideep Chandran
*
--* Date       : 10/03/2011
*
--* Description: Block calculate BMP image Red Color value, and
returns 32-Bit*
--*              single precision floating point format
*
--
*************************************************************************
*******

--
*************************************************************************
*******
--* Include Libraries
*
--
*************************************************************************
*******
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--
*************************************************************************
*******
--* Entity Decleration, I/O ports
*
--
*************************************************************************
*******
entity cph_analyzer_cext is
    Port ( hclk            : in  STD_LOGIC;
           hreset_n        : in  STD_LOGIC;
           bmp_strt_val    : in  STD_LOGIC;
           bmp_sr_sav      : in  STD_LOGIC;
           bmp_ram_dataout : in  STD_LOGIC_VECTOR (7 downto 0);
           bmp_red_mv      : out STD_LOGIC_VECTOR (31 downto 0);
           bmp_red_mf      : out STD_LOGIC_VECTOR (31 downto 0));
end cph_analyzer_cext;

--
*************************************************************************
*******
--* Main Block Logic
*
--
*************************************************************************
*******
```

```
architecture Behavioral of cph_analyzer_cext is

--
*************************************************************************
*******
--* Signal Decleration
*
--
*************************************************************************
*******
  signal cvr_sum   : STD_LOGIC_VECTOR (23 downto 0);
  begin
--
*************************************************************************
*******
--*
*
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     cvr_sum <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if (bmp_strt_val = '1') then
      cvr_sum <= cvr_sum + (X"0000" & bmp_ram_dataout);
     end if;
    end if;
   end if;
  end process;

--
*************************************************************************
*******
--* bmp_red_mf : BMP image mean factor value
*
--* As we have to calculate the Red color Value of 200 * 200 Pixel
from the   *
--* center of the image. so we divide the sum of 200 * 200 pixel by
40,000    *
--* bmp_red_mf contains the [1/40,000 = 0.00025 base 10] binary value
in 32Bit*
--* single precision floating point format
*
--* 32Bit Single Precision Floating Point Format:
*
--* ==========================================
*
--* 31          24,23
0  *
--* <||-------------||-------------------------------------------------
-----|> *
--* s  Mantissa                          Exponent
*
--* Bit[31]    = s = Sign Bit
*
--* Bit[30:23] = Mantissa
*
--* Bit[23:0]  = Exponent
*
```

```
--* 1/40,000 =
0000_0000_0000_0001_1011_1011_1110_0111_0110_1100_1000_1011(2) *
--* After Normalize
*
--* 1/40,000 = 0.000025
*
--*        =
0.0000_0000_0000_0001_1011_1011_1110_0111_0110_1100_1000_1011(2) *
--*          1.1011_1011_1110_0111_0110_1100_1000_1011 2 ^ -16
*
--* Mantissa = 1011_1011_1110_0111_0110_110
*
--* Exponent = -16 + 127 = 121 = 0111_1001
*
--* sign bit = 0
*
--* 32 Bit Floating Point Final Value
*
--* 0011_1100_1101_1101_1111_0011_1011_0110
*
--
*********************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_red_mf <= (others => '0');
    else if (hclk 'event and hclk = '1') then
     if (bmp_sr_sav = '1') then
      bmp_red_mf <= X"3cddf3b6";
    end if;
    end if;
   end if;
  end process;

--
*********************************************************************
*******
--* Convert Red colour Value to Floating Point Numbers
*
--
*********************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_red_mv <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if (bmp_sr_sav = '1' and cvr_sum(23) = '1') then
      bmp_red_mv <= "0" & X"96" & cvr_sum (22 downto 0);
     else if (bmp_sr_sav = '1' and cvr_sum(22) = '1') then
      bmp_red_mv <= "0" & X"95" & cvr_sum (21 downto 0) & "0";
     else if (bmp_sr_sav = '1' and cvr_sum(21) = '1') then
      bmp_red_mv <= "0" & X"94" & cvr_sum (20 downto 0) & "00";
     else if (bmp_sr_sav = '1' and cvr_sum(20) = '1') then
      bmp_red_mv <= "0" & X"93" & cvr_sum (19 downto 0) & "000";
     else if (bmp_sr_sav = '1' and cvr_sum(19) = '1') then
      bmp_red_mv <= "0" & X"92" & cvr_sum (18 downto 0) & X"0";
     else if (bmp_sr_sav = '1' and cvr_sum(18) = '1') then
      bmp_red_mv <= "0" & X"91" & cvr_sum (17 downto 0) & "00000";
     else if (bmp_sr_sav = '1' and cvr_sum(17) = '1') then
```

```
      bmp_red_mv <= "0" & X"90" & cvr_sum (16 downto 0) & "000000";
     else if (bmp_sr_sav = '1' and cvr_sum(16) = '1') then
      bmp_red_mv <= "0" & X"8F" & cvr_sum (15 downto 0) & "0000000";
     else if (bmp_sr_sav = '1' and cvr_sum(15) = '1') then
      bmp_red_mv <= "0" & X"8E" & cvr_sum (14 downto 0) & "00000000";
     else if (bmp_sr_sav = '1' and cvr_sum(14) = '1') then
      bmp_red_mv <= "0" & X"8D" & cvr_sum (13 downto 0) & "000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(13) = '1') then
      bmp_red_mv <= "0" & X"8C" & cvr_sum (12 downto 0) &
"0000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(12) = '1') then
      bmp_red_mv <= "0" & X"8B" & cvr_sum (11 downto 0) &
"00000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(11) = '1') then
      bmp_red_mv <= "0" & X"8A" & cvr_sum (10 downto 0) &
"000000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(10) = '1') then
      bmp_red_mv <= "0" & X"89" & cvr_sum (9 downto 0) &
"0000000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(9) = '1') then
      bmp_red_mv <= "0" & X"88" & cvr_sum (8 downto 0) &
"00000000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(8) = '1') then
      bmp_red_mv <= "0" & X"87" & cvr_sum (7 downto 0) &
"000000000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(7) = '1') then
      bmp_red_mv <= "0" & X"86" & cvr_sum (6 downto 0) &
"0000000000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(6) = '1') then
      bmp_red_mv <= "0" & X"85" & cvr_sum (5 downto 0) &
"00000000000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(5) = '1') then
      bmp_red_mv <= "0" & X"84" & cvr_sum (4 downto 0) &
"000000000000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(4) = '1') then
      bmp_red_mv <= "0" & X"83" & cvr_sum (3 downto 0) &
"0000000000000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(3) = '1') then
      bmp_red_mv <= "0" & X"82" & cvr_sum (2 downto 0) &
"00000000000000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(2) = '1') then
      bmp_red_mv <= "0" & X"81" & cvr_sum (1 downto 0) &
"000000000000000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(1) = '1') then
      bmp_red_mv <= "0" & X"80" & cvr_sum (0) &
"0000000000000000000000";
     else if (bmp_sr_sav = '1' and cvr_sum(0) = '1') then
      bmp_red_mv <= "0" & X"7F" & "00000000000000000000000";
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
```

```
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
     end if;
   end if;
   end if;
  end process;

end Behavioral;


--
**********************************************************************
*******
--* File Name  : cph_analyzer_co2cal.vhd
*
--* Designer   : Jaideep Chandran
*
--* Date       : 10/03/2011
*
--* Description:
*
--
**********************************************************************
*******
--
**********************************************************************
*******
--* Include Libraries
*
--
**********************************************************************
*******
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--
**********************************************************************
*******
--* Entity Decleration, I/O ports
*
--
**********************************************************************
*******
entity cph_analyzer_co2cal is
    Port (
           hclk             : in  STD_LOGIC;
           hreset_n         : in  STD_LOGIC;
           bmp_ipco2_sext   : in  STD_LOGIC;
           bmp_iph_val      : in  STD_LOGIC_VECTOR (31 downto 0);
           bmp_pco2ext_dn   : out STD_LOGIC;
           cph_co2_val      : out STD_LOGIC_VECTOR (31 downto 0));
```

```
end cph_analyzer_co2cal;

--
*************************************************************************
*******
--* Main Block Logic
*
--
*************************************************************************
*******
architecture Behavioral of cph_analyzer_co2cal is

  signal co2_ip_mreg0   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg1   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg2   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg3   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg4   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg5   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg6   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg7   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg8   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg9   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg10  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg11  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg12  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg13  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg14  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg15  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg16  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg17  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg18  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg20  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg22  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg24  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg26  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg28  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg30  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg32  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg34  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg36  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg38  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg40  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg42  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg44  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg46  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg48  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg50  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg52  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg54  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg56  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_mreg59  : STD_LOGIC_VECTOR (31 downto 0);

  signal co2_ip_areg19  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg21 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg23 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg25 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg27 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg29 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg31 : STD_LOGIC_VECTOR (31 downto 0);
```

```
  signal co2_ip_areg33 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg35 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg37 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg39 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg41 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg43 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg45 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg47 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg49 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg51 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg53 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg55 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg58 : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_ip_areg57 : STD_LOGIC_VECTOR (31 downto 0);

  signal co2_inv_loge    : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_bicarb_con  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_one_const   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial0  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial1  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial2  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial3  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial4  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial5  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial6  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial7  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial8  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial9  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial10  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial11  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial12  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial13  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial14  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial15  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial16  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial17  : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_factorial18  : STD_LOGIC_VECTOR (31 downto 0);


  signal co2_ip_mcnt   : STD_LOGIC_VECTOR (5 downto 0);
  signal co2_ip_acnt   : STD_LOGIC_VECTOR (4 downto 0);
  signal co2_mresult   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_aresult   : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_muxa_data : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_muxb_data : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_muxc_data : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_muxd_data : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_muxa_sel  : STD_LOGIC_VECTOR (4 downto 0);
  signal co2_muxb_sel  : STD_LOGIC_VECTOR (4 downto 0);
  signal co2_ipreg_we  : STD_LOGIC_VECTOR (39 downto 0);
  signal co2_iareg_we : STD_LOGIC_VECTOR (20 downto 0);

--
************************************************************************
*******
--* Pipeline Register
*
--
************************************************************************
*******
```

```
  signal co2_muxa_data_p : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_muxb_data_p : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_muxc_data_p : STD_LOGIC_VECTOR (31 downto 0);
  signal co2_muxd_data_p : STD_LOGIC_VECTOR (31 downto 0);

--
*****************************************************************
*******
--* Component Decleration
*
--
*****************************************************************
*******
  component cph_analyzer_add is
    port (
          hclk     : in STD_LOGIC;
          hreset_n : in STD_LOGIC;
          op_a     : in STD_LOGIC_VECTOR (31 downto 0);
          op_b     : in STD_LOGIC_VECTOR (31 downto 0);
          result   : out std_logic_vector(31 downto 0));
  end component;

  component cph_analyzer_mul is
    port (
          hclk     : in STD_LOGIC;
          hreset_n : in STD_LOGIC;
          op_a     : in STD_LOGIC_VECTOR (31 downto 0);
          op_b     : in STD_LOGIC_VECTOR (31 downto 0);
          result   : out std_logic_vector(31 downto 0));
  end component;

begin
--
*****************************************************************
*******
--* Constant values used in equations calculation
*
--
*****************************************************************
*******
  co2_one_const   <= "00111111100000000000000000000000";
  co2_bicarb_con  <= "11000001000100010110100100111101";-- bicarbonate
constants logN+pka-logKs
  co2_inv_loge    <= "11000000000100110101110110001110";-- 1/log e
  co2_factorial0  <= "00111111000000000000000000000000" ;--2!
  co2_factorial1  <= "00111110001010101010101010101011" ;--3!
  co2_factorial2  <= "00111101001010101010101010101011" ;--4!
  co2_factorial3  <= "00111100000010001000100010001001" ;--5!
  co2_factorial4  <= "00111010101101100000101101100001" ;--6!
  co2_factorial5  <= "00111001010100000000110100000001" ;--7!
  co2_factorial6  <= "00110111111010000000110100000001" ;--8!
  co2_factorial7  <= "00110110001110001110111100011101" ;--9!
  co2_factorial8  <= "00110100100100111111001001111110" ;--10!
  co2_factorial9  <= "00110010111010111001001000101011" ;--11!
  co2_factorial10 <= "00110001000011110111011011000111" ;--12!
  co2_factorial11 <= "00101111001100001001001000110001" ;--13!
  co2_factorial12 <= "00101101010010011100101110100101" ;--14!
  co2_factorial13 <= "00101011010101110011111110011111" ;--15!
  co2_factorial14 <= "00101001010101110011111110011111" ;--16!
  co2_factorial15 <= "00100111010010101001011000111100" ;--17!
  co2_factorial16 <= "00100101001101000001001111000011" ;--18!
```

```
  co2_factorial17 <= "00100011000101111010010011011010" ;--19!
  co2_factorial18 <= "00100000111100101010000101011101" ;--20!
  cph_co2_val     <= co2_ip_areg57;
--
*************************************************************************
*******
--* Logic to Calculate the Intermediate product of multiplication
*
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     co2_ip_mcnt <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if ( bmp_ipco2_sext = '1' and co2_ip_mcnt < "100111") then
        co2_ip_mcnt <= co2_ip_mcnt + 1;
      else if (bmp_ipco2_sext = '1' and co2_ip_mcnt = "100111") then
        co2_ip_mcnt <= co2_ip_mcnt;
      else
        co2_ip_mcnt <= (others => '0');
      end if;
      end if;
    end if;
    end if;
  end process;

--
*************************************************************************
*******
--* Multiplexer
*
--
*************************************************************************
*******
  process(co2_ip_mcnt)
  begin
     case (co2_ip_mcnt) is
       when "000000"  => co2_muxa_sel <= "00000";
       when "000001"  => co2_muxa_sel <= "00001";
       when "000010"  => co2_muxa_sel <= "00010";
       when "000011"  => co2_muxa_sel <= "00011";
       when "000100"  => co2_muxa_sel <= "00100";
       when "000101"  => co2_muxa_sel <= "00101";
       when "000110"  => co2_muxa_sel <= "00110";
       when "000111"  => co2_muxa_sel <= "00111";
       when "001000"  => co2_muxa_sel <= "01000";
       when "001001"  => co2_muxa_sel <= "01001";
       when "001010"  => co2_muxa_sel <= "01010";
       when "001011"  => co2_muxa_sel <= "01011";
       when "001100"  => co2_muxa_sel <= "01100";
       when "001101"  => co2_muxa_sel <= "01101";
       when "001110"  => co2_muxa_sel <= "01110";
       when "001111"  => co2_muxa_sel <= "01111";
       when "010000"  => co2_muxa_sel <= "10000";
       when "010001"  => co2_muxa_sel <= "10001";
       when "010010"  => co2_muxa_sel <= "10010";
       when "010011"  => co2_muxa_sel <= "10011";
       when "010100"  => co2_muxa_sel <= "10100";
       when "010101"  => co2_muxa_sel <= "00010";
```

```
      when "010110"  => co2_muxa_sel <= "00011";
      when "010111"  => co2_muxa_sel <= "00100";
      when "011000"  => co2_muxa_sel <= "00101";
      when "011001"  => co2_muxa_sel <= "00110";
      when "011010"  => co2_muxa_sel <= "00111";
      when "011011"  => co2_muxa_sel <= "01000";
      when "011100"  => co2_muxa_sel <= "01001";
      when "011101"  => co2_muxa_sel <= "01010";
      when "011110"  => co2_muxa_sel <= "01011";
      when "011111"  => co2_muxa_sel <= "01100";
      when "100000"  => co2_muxa_sel <= "01101";
      when "100001"  => co2_muxa_sel <= "01110";
      when "100010"  => co2_muxa_sel <= "01111";
      when "100011"  => co2_muxa_sel <= "10000";
      when "100100"  => co2_muxa_sel <= "10001";
      when "100101"  => co2_muxa_sel <= "10010";
      when "100110"  => co2_muxa_sel <= "10011";
      when "100111"  => co2_muxa_sel <= "10100";
      when others    => co2_muxa_sel <= (others => '0');
   end case;
end process;

process(co2_ip_mcnt)
begin
   case (co2_ip_mcnt) is
     when "000000"  => co2_muxb_sel <= "00000";
     when "000001"  => co2_muxb_sel <= "00001";
     when "000010"  => co2_muxb_sel <= "00001";
     when "000011"  => co2_muxb_sel <= "00001";
     when "000100"  => co2_muxb_sel <= "00001";
     when "000101"  => co2_muxb_sel <= "00001";
     when "000110"  => co2_muxb_sel <= "00001";
     when "000111"  => co2_muxb_sel <= "00001";
     when "001000"  => co2_muxb_sel <= "00001";
     when "001001"  => co2_muxb_sel <= "00001";
     when "001010"  => co2_muxb_sel <= "00001";
     when "001011"  => co2_muxb_sel <= "00001";
     when "001100"  => co2_muxb_sel <= "00001";
     when "001101"  => co2_muxb_sel <= "00001";
     when "001110"  => co2_muxb_sel <= "00001";
     when "001111"  => co2_muxb_sel <= "00001";
     when "010000"  => co2_muxb_sel <= "00001";
     when "010001"  => co2_muxb_sel <= "00001";
     when "010010"  => co2_muxb_sel <= "00001";
     when "010011"  => co2_muxb_sel <= "00001";
     when "010100"  => co2_muxb_sel <= "00001";
     when "010101"  => co2_muxb_sel <= "00010";
     when "010110"  => co2_muxb_sel <= "00011";
     when "010111"  => co2_muxb_sel <= "00100";
     when "011000"  => co2_muxb_sel <= "00101";
     when "011001"  => co2_muxb_sel <= "00110";
     when "011010"  => co2_muxb_sel <= "00111";
     when "011011"  => co2_muxb_sel <= "01000";
     when "011100"  => co2_muxb_sel <= "01001";
     when "011101"  => co2_muxb_sel <= "01010";
     when "011110"  => co2_muxb_sel <= "01011";
     when "011111"  => co2_muxb_sel <= "01100";
     when "100000"  => co2_muxb_sel <= "01101";
     when "100001"  => co2_muxb_sel <= "01110";
     when "100010"  => co2_muxb_sel <= "01111";
     when "100011"  => co2_muxb_sel <= "10000";
```

```
       when "100100"  => co2_muxb_sel <= "10001";
       when "100101"  => co2_muxb_sel <= "10010";
       when "100110"  => co2_muxb_sel <= "10011";
       when "100111"  => co2_muxb_sel <= "10100";
       when others    => co2_muxb_sel <= (others => '0');
     end case;
  end process;

--
*************************************************************************
*******
--* Multiplexer for Multiplier OperandA
*
--
*************************************************************************
*******
  process(co2_ip_mreg0 , co2_ip_mreg1 , co2_ip_mreg2 , co2_ip_mreg3,
          co2_ip_mreg4 , co2_ip_mreg5 , co2_ip_mreg6 , co2_ip_mreg7,
          co2_ip_mreg8 , co2_ip_areg9 , co2_ip_mreg10, co2_ip_mreg11,
          co2_ip_mreg12, co2_ip_mreg13, co2_ip_mreg14, co2_ip_mreg15,
          co2_ip_mreg16, co2_ip_mreg17, co2_ip_mreg18, co2_ip_mreg59,
          co2_ip_areg58, co2_muxa_sel)
  begin
     case (co2_muxa_sel) is
       when "00000"  => co2_muxa_data <= co2_ip_areg58;
       when "00001"  => co2_muxa_data <= co2_ip_mreg59;
       when "00010"  => co2_muxa_data <= co2_ip_mreg0;
       when "00011"  => co2_muxa_data <= co2_ip_mreg1;
       when "00100"  => co2_muxa_data <= co2_ip_mreg2;
       when "00101"  => co2_muxa_data <= co2_ip_mreg3;
       when "00110"  => co2_muxa_data <= co2_ip_mreg4;
       when "00111"  => co2_muxa_data <= co2_ip_mreg5;
       when "01000"  => co2_muxa_data <= co2_ip_mreg6;
       when "01001"  => co2_muxa_data <= co2_ip_mreg7;
       when "01010"  => co2_muxa_data <= co2_ip_mreg8;
       when "01011"  => co2_muxa_data <= co2_ip_areg9;
       when "01100"  => co2_muxa_data <= co2_ip_mreg10;
       when "01101"  => co2_muxa_data <= co2_ip_mreg11;
       when "01110"  => co2_muxa_data <= co2_ip_mreg12;
       when "01111"  => co2_muxa_data <= co2_ip_mreg13;
       when "10000"  => co2_muxa_data <= co2_ip_mreg14;
       when "10001"  => co2_muxa_data <= co2_ip_mreg15;
       when "10010"  => co2_muxa_data <= co2_ip_mreg16;
       when "10011"  => co2_muxa_data <= co2_ip_mreg17;
       when "10100"  => co2_muxa_data <= co2_ip_mreg18;
       when others   => co2_muxa_data <= co2_ip_areg58;
     end case;
  end process;

--
*************************************************************************
*******
--* Multiplexer for Multiplier OperandB
*
--
*************************************************************************
*******
  process(co2_factorial0 , co2_factorial1 , co2_factorial2 ,
co2_factorial3,
          co2_factorial4 , co2_factorial5 , co2_factorial6 ,
co2_factorial7,
```

```
          co2_factorial8 , co2_factorial9 , co2_factorial10,
co2_factorial11,
          co2_factorial12, co2_factorial13, co2_factorial14,
co2_factorial15,
          co2_factorial16, co2_factorial17, co2_factorial18,
co2_ip_mreg59,
          co2_inv_loge   , co2_muxb_sel)
  begin
     case (co2_muxb_sel) is
       when "00000"  => co2_muxb_data <= co2_inv_loge;
       when "00001"  => co2_muxb_data <= co2_ip_mreg59;
       when "00010"  => co2_muxb_data <= co2_factorial0;
       when "00011"  => co2_muxb_data <= co2_factorial1;
       when "00100"  => co2_muxb_data <= co2_factorial2;
       when "00101"  => co2_muxb_data <= co2_factorial3;
       when "00110"  => co2_muxb_data <= co2_factorial4;
       when "00111"  => co2_muxb_data <= co2_factorial5;
       when "01000"  => co2_muxb_data <= co2_factorial6;
       when "01001"  => co2_muxb_data <= co2_factorial7;
       when "01010"  => co2_muxb_data <= co2_factorial8;
       when "01011"  => co2_muxb_data <= co2_factorial9;
       when "01100"  => co2_muxb_data <= co2_factorial10;
       when "01101"  => co2_muxb_data <= co2_factorial11;
       when "01110"  => co2_muxb_data <= co2_factorial12;
       when "01111"  => co2_muxb_data <= co2_factorial13;
       when "10000"  => co2_muxb_data <= co2_factorial14;
       when "10001"  => co2_muxb_data <= co2_factorial15;
       when "10010"  => co2_muxb_data <= co2_factorial16;
       when "10011"  => co2_muxb_data <= co2_factorial17;
       when "10100"  => co2_muxb_data <= co2_factorial18;
       when others   => co2_muxb_data <= co2_ip_mreg59;
     end case;
  end process;

--
*************************************************************************
*******
--* Logic to generate intermediate(multiplication) reg write enable
signals   *
--
*************************************************************************
*******
  process(co2_ip_mcnt)
  begin
     case (co2_ip_mcnt) is
       when "000000"  => co2_ipreg_we <= X"0000000001";
       when "000001"  => co2_ipreg_we <= X"0000000002";
       when "000010"  => co2_ipreg_we <= X"0000000004";
       when "000011"  => co2_ipreg_we <= X"0000000008";
       when "000100"  => co2_ipreg_we <= X"0000000010";
       when "000101"  => co2_ipreg_we <= X"0000000020";
       when "000110"  => co2_ipreg_we <= X"0000000040";
       when "000111"  => co2_ipreg_we <= X"0000000080";
       when "001000"  => co2_ipreg_we <= X"0000000100";
       when "001001"  => co2_ipreg_we <= X"0000000200";
       when "001010"  => co2_ipreg_we <= X"0000000400";
       when "001011"  => co2_ipreg_we <= X"0000000800";
       when "001100"  => co2_ipreg_we <= X"0000001000";
       when "001101"  => co2_ipreg_we <= X"0000002000";
       when "001110"  => co2_ipreg_we <= X"0000004000";
       when "001111"  => co2_ipreg_we <= X"0000008000";
```

```
       when "010000"  => co2_ipreg_we <= X"0000010000";
       when "010001"  => co2_ipreg_we <= X"0000020000";
       when "010010"  => co2_ipreg_we <= X"0000040000";
       when "010011"  => co2_ipreg_we <= X"0000080000";
       when "010100"  => co2_ipreg_we <= X"0000100000";
       when "010101"  => co2_ipreg_we <= X"0000200000";
       when "010110"  => co2_ipreg_we <= X"0000400000";
       when "010111"  => co2_ipreg_we <= X"0000800000";
       when "011000"  => co2_ipreg_we <= X"0001000000";
       when "011001"  => co2_ipreg_we <= X"0002000000";
       when "011010"  => co2_ipreg_we <= X"0004000000";
       when "011011"  => co2_ipreg_we <= X"0008000000";
       when "011100"  => co2_ipreg_we <= X"0010000000";
       when "011101"  => co2_ipreg_we <= X"0020000000";
       when "011110"  => co2_ipreg_we <= X"0040000000";
       when "011111"  => co2_ipreg_we <= X"0080000000";
       when "100000"  => co2_ipreg_we <= X"0100000000";
       when "100001"  => co2_ipreg_we <= X"0200000000";
       when "100010"  => co2_ipreg_we <= X"0400000000";
       when "100011"  => co2_ipreg_we <= X"0800000000";
       when "100100"  => co2_ipreg_we <= X"1000000000";
       when "100101"  => co2_ipreg_we <= X"2000000000";
       when "100110"  => co2_ipreg_we <= X"4000000000";
       when "100111"  => co2_ipreg_we <= X"8000000000";
       when others    => co2_ipreg_we <= (others => '0');
     end case;
  end process;

--
*************************************************************************
*******
--* Logic to Calculate the Intermediate product of Addition
*
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     co2_ip_acnt <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if (bmp_ipco2_sext = '1' and co2_ip_mcnt = "100111") then
        co2_ip_acnt <= co2_ip_acnt + 1;
      end if;
    end if;
    end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_pco2ext_dn <= '0';
    else if (hclk 'event and hclk = '1') then
      if ( co2_ip_acnt = "1001") then
        bmp_pco2ext_dn <= '1';
      end if;
    end if;
    end if;
  end process;
```

```
--
*************************************************************************
*******
--* Multiplexer for Adder OperandA
*
--
*************************************************************************
*******
  process(bmp_iph_val  , co2_ip_acnt  , co2_ip_mreg59, co2_ip_areg19,
          co2_ip_mreg22, co2_ip_mreg24, co2_ip_mreg26, co2_ip_mreg28,
          co2_ip_mreg30, co2_ip_mreg32, co2_ip_mreg34, co2_ip_mreg36,
          co2_ip_mreg38, co2_ip_mreg40, co2_ip_mreg42, co2_ip_mreg44,
          co2_ip_mreg46, co2_ip_mreg48, co2_ip_mreg50, co2_ip_mreg52,
          co2_ip_mreg54, co2_ip_mreg56)
  begin
     case (co2_ip_acnt) is
       when "00000"  => co2_muxc_data <= bmp_iph_val;
       when "00001"  => co2_muxc_data <= co2_ip_mreg59;
       when "00010"  => co2_muxc_data <= co2_ip_areg19;
       when "00011"  => co2_muxc_data <= co2_ip_mreg22;
       when "00100"  => co2_muxc_data <= co2_ip_mreg24;
       when "00101"  => co2_muxc_data <= co2_ip_mreg26;
       when "00110"  => co2_muxc_data <= co2_ip_mreg28;
       when "00111"  => co2_muxc_data <= co2_ip_mreg30;
       when "01000"  => co2_muxc_data <= co2_ip_mreg32;
       when "01001"  => co2_muxc_data <= co2_ip_mreg34;
       when "01010"  => co2_muxc_data <= co2_ip_mreg36;
       when "01011"  => co2_muxc_data <= co2_ip_mreg38;
       when "01100"  => co2_muxc_data <= co2_ip_mreg40;
       when "01101"  => co2_muxc_data <= co2_ip_mreg42;
       when "01110"  => co2_muxc_data <= co2_ip_mreg44;
       when "01111"  => co2_muxc_data <= co2_ip_mreg46;
       when "10000"  => co2_muxc_data <= co2_ip_mreg48;
       when "10001"  => co2_muxc_data <= co2_ip_mreg50;
       when "10010"  => co2_muxc_data <= co2_ip_mreg52;
       when "10011"  => co2_muxc_data <= co2_ip_mreg54;
       when "10100"  => co2_muxc_data <= co2_ip_mreg56;
       when others   => co2_muxc_data <= bmp_iph_val;
     end case;
  end process;

--
*************************************************************************
*******
--* Multiplexer for Adder  OperandB
*
--
*************************************************************************
*******
  process(co2_one_const, co2_ip_acnt  , co2_bicarb_con, co2_ip_mreg20,
          co2_ip_areg21, co2_ip_areg31, co2_ip_areg41 , co2_ip_areg49,
          co2_ip_areg23, co2_ip_areg33, co2_ip_areg43 , co2_ip_areg51,
          co2_ip_areg25, co2_ip_areg35, co2_ip_areg45 , co2_ip_areg53,
          co2_ip_areg27, co2_ip_areg37, co2_ip_areg47 , co2_ip_areg55,
          co2_ip_areg29, co2_ip_areg39)
  begin
     case (co2_ip_acnt) is
       when "00000"  => co2_muxd_data <= co2_one_const;
       when "00001"  => co2_muxd_data <= co2_bicarb_con;
       when "00010"  => co2_muxd_data <= co2_ip_mreg20;
       when "00011"  => co2_muxd_data <= co2_ip_areg21;
```

```
      when "00100"  => co2_muxd_data <= co2_ip_areg23;
      when "00101"  => co2_muxd_data <= co2_ip_areg25;
      when "00110"  => co2_muxd_data <= co2_ip_areg27;
      when "00111"  => co2_muxd_data <= co2_ip_areg29;
      when "01000"  => co2_muxd_data <= co2_ip_areg31;
      when "01001"  => co2_muxd_data <= co2_ip_areg33;
      when "01010"  => co2_muxd_data <= co2_ip_areg35;
      when "01011"  => co2_muxd_data <= co2_ip_areg37;
      when "01100"  => co2_muxd_data <= co2_ip_areg39;
      when "01101"  => co2_muxd_data <= co2_ip_areg41;
      when "01110"  => co2_muxd_data <= co2_ip_areg43;
      when "01111"  => co2_muxd_data <= co2_ip_areg45;
      when "10000"  => co2_muxd_data <= co2_ip_areg47;
      when "10001"  => co2_muxd_data <= co2_ip_areg49;
      when "10010"  => co2_muxd_data <= co2_ip_areg51;
      when "10011"  => co2_muxd_data <= co2_ip_areg53;
      when "10100"  => co2_muxd_data <= co2_ip_areg55;
      when others   => co2_muxd_data <= co2_one_const;
    end case;
  end process;



--
************************************************************************
*******
--* Logic to generate intermediate(addition) reg write enable signals
*
--
************************************************************************
*******
  process(co2_ip_acnt)
  begin
    case (co2_ip_acnt) is
      when "00000"  => co2_iareg_we <= "0" & X"00001";
      when "00001"  => co2_iareg_we <= "0" & X"00002";
      when "00010"  => co2_iareg_we <= "0" & X"00004";
      when "00011"  => co2_iareg_we <= "0" & X"00008";
      when "00100"  => co2_iareg_we <= "0" & X"00010";
      when "00101"  => co2_iareg_we <= "0" & X"00020";
      when "00110"  => co2_iareg_we <= "0" & X"00040";
      when "00111"  => co2_iareg_we <= "0" & X"00080";
      when "01000"  => co2_iareg_we <= "0" & X"00100";
      when "01001"  => co2_iareg_we <= "0" & X"00200";
      when "01010"  => co2_iareg_we <= "0" & X"00400";
      when "01011"  => co2_iareg_we <= "0" & X"00800";
      when "01100"  => co2_iareg_we <= "0" & X"01000";
      when "01101"  => co2_iareg_we <= "0" & X"02000";
      when "01110"  => co2_iareg_we <= "0" & X"04000";
      when "01111"  => co2_iareg_we <= "0" & X"08000";
      when "10000"  => co2_iareg_we <= "0" & X"10000";
      when "10001"  => co2_iareg_we <= "0" & X"20000";
      when "10010"  => co2_iareg_we <= "0" & X"40000";
      when "10011"  => co2_iareg_we <= "0" & X"80000";
      when "10100"  => co2_iareg_we <= "1" & X"00000";
      when others   => co2_iareg_we <= (others => '0');
    end case;
  end process;
```

```
--
*************************************************************************
*******
--* Store intermediate results in reg
*
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     co2_ip_mreg59 <= (others => '0');
     co2_ip_mreg0  <= (others => '0');
     co2_ip_mreg1  <= (others => '0');
     co2_ip_mreg2  <= (others => '0');
     co2_ip_mreg3  <= (others => '0');
     co2_ip_mreg4  <= (others => '0');
     co2_ip_mreg5  <= (others => '0');
     co2_ip_mreg6  <= (others => '0');
     co2_ip_mreg7  <= (others => '0');
     co2_ip_mreg8  <= (others => '0');
     co2_ip_mreg10 <= (others => '0');
     co2_ip_mreg11 <= (others => '0');
     co2_ip_mreg12 <= (others => '0');
     co2_ip_mreg13 <= (others => '0');
     co2_ip_mreg14 <= (others => '0');
     co2_ip_mreg15 <= (others => '0');
     co2_ip_mreg16 <= (others => '0');
     co2_ip_mreg17 <= (others => '0');
     co2_ip_mreg18 <= (others => '0');
     co2_ip_mreg20 <= (others => '0');
     co2_ip_mreg22 <= (others => '0');
     co2_ip_mreg24 <= (others => '0');
     co2_ip_mreg26 <= (others => '0');
     co2_ip_mreg28 <= (others => '0');
     co2_ip_mreg30 <= (others => '0');
     co2_ip_mreg32 <= (others => '0');
     co2_ip_mreg34 <= (others => '0');
     co2_ip_mreg36 <= (others => '0');
     co2_ip_mreg38 <= (others => '0');
     co2_ip_mreg40 <= (others => '0');
     co2_ip_mreg42 <= (others => '0');
     co2_ip_mreg44 <= (others => '0');
     co2_ip_mreg46 <= (others => '0');
     co2_ip_mreg48 <= (others => '0');
     co2_ip_mreg50 <= (others => '0');
     co2_ip_mreg52 <= (others => '0');
     co2_ip_mreg54 <= (others => '0');
     co2_ip_mreg56 <= (others => '0');

    else if (hclk 'event and hclk = '1') then
      if ( co2_ipreg_we(0) = '1') then
       co2_ip_mreg59 <= co2_mresult;
      end if;
      if ( co2_ipreg_we(1) = '1') then
       co2_ip_mreg0  <= co2_mresult;
      end if;
      if ( co2_ipreg_we(2) = '1') then
       co2_ip_mreg1  <= co2_mresult;
      end if;
      if ( co2_ipreg_we(3) = '1') then
```

```
        co2_ip_mreg2  <= co2_mresult;
       end if;
       if ( co2_ipreg_we(4) = '1') then
        co2_ip_mreg3  <= co2_mresult;
       end if;
       if ( co2_ipreg_we(5) = '1') then
        co2_ip_mreg4  <= co2_mresult;
       end if;
       if ( co2_ipreg_we(6) = '1') then
        co2_ip_mreg5  <= co2_mresult;
       end if;
       if ( co2_ipreg_we(7) = '1') then
        co2_ip_mreg6  <= co2_mresult;
       end if;
       if ( co2_ipreg_we(8) = '1') then
        co2_ip_mreg7  <= co2_mresult;
       end if;
       if ( co2_ipreg_we(9) = '1') then
        co2_ip_mreg8  <= co2_mresult;
       end if;
       if ( co2_ipreg_we(10) = '1') then
        co2_ip_areg9  <= co2_mresult;
       end if;
       if ( co2_ipreg_we(11) = '1') then
        co2_ip_mreg10 <= co2_mresult;
       end if;
       if ( co2_ipreg_we(12) = '1') then
        co2_ip_mreg11 <= co2_mresult;
       end if;
       if ( co2_ipreg_we(13) = '1') then
        co2_ip_mreg12 <= co2_mresult;
       end if;
       if ( co2_ipreg_we(14) = '1') then
        co2_ip_mreg13 <= co2_mresult;
       end if;
       if ( co2_ipreg_we(15) = '1') then
        co2_ip_mreg14 <= co2_mresult;
       end if;
       if ( co2_ipreg_we(16) = '1') then
        co2_ip_mreg15 <= co2_mresult;
       end if;
       if ( co2_ipreg_we(17) = '1') then
        co2_ip_mreg16 <= co2_mresult;
       end if;
       if ( co2_ipreg_we(18) = '1') then
        co2_ip_mreg17 <= co2_mresult;
       end if;
       if ( co2_ipreg_we(19) = '1') then
        co2_ip_mreg18 <= co2_mresult;
       end if;
--       if ( co2_ipreg_we(20) = '1') then
--        co2_ip_areg19 <= co2_mresult;
--       end if;
       if ( co2_ipreg_we(21) = '1') then
        co2_ip_mreg20 <= co2_mresult;
       end if;
       if ( co2_ipreg_we(22) = '1') then
        co2_ip_mreg22 <= co2_mresult;
       end if;
       if ( co2_ipreg_we(23) = '1') then
        co2_ip_mreg24 <= co2_mresult;
```

```
    end if;
    if ( co2_ipreg_we(24) = '1') then
     co2_ip_mreg26 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(25) = '1') then
     co2_ip_mreg28 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(26) = '1') then
     co2_ip_mreg30 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(27) = '1') then
     co2_ip_mreg32 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(28) = '1') then
     co2_ip_mreg34 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(29) = '1') then
     co2_ip_mreg36 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(30) = '1') then
     co2_ip_mreg38 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(31) = '1') then
     co2_ip_mreg40 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(32) = '1') then
     co2_ip_mreg42 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(33) = '1') then
     co2_ip_mreg44 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(34) = '1') then
     co2_ip_mreg46 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(35) = '1') then
     co2_ip_mreg48 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(36) = '1') then
     co2_ip_mreg50 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(37) = '1') then
     co2_ip_mreg52 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(38) = '1') then
     co2_ip_mreg54 <= co2_mresult;
    end if;
    if ( co2_ipreg_we(39) = '1') then
     co2_ip_mreg56 <= co2_mresult;
    end if;
  end if;
  end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   co2_ip_areg58 <= (others => '0');
   co2_ip_areg19  <= (others => '0');
   co2_ip_areg21  <= (others => '0');
   co2_ip_areg23  <= (others => '0');
   co2_ip_areg25  <= (others => '0');
```

```
 co2_ip_areg27  <= (others => '0');
 co2_ip_areg29  <= (others => '0');
 co2_ip_areg31 <= (others => '0');
 co2_ip_areg33 <= (others => '0');
 co2_ip_areg35 <= (others => '0');
 co2_ip_areg37 <= (others => '0');
 co2_ip_areg39 <= (others => '0');
 co2_ip_areg41 <= (others => '0');
 co2_ip_areg43 <= (others => '0');
 co2_ip_areg45 <= (others => '0');
 co2_ip_areg47 <= (others => '0');
 co2_ip_areg49 <= (others => '0');
 co2_ip_areg51 <= (others => '0');
 co2_ip_areg53 <= (others => '0');
 co2_ip_areg55 <= (others => '0');
 co2_ip_areg57 <= (others => '0');

else if (hclk 'event and hclk = '1') then
  if ( co2_iareg_we(0) = '1') then
   co2_ip_areg58 <= co2_aresult;
  end if;
  if ( co2_iareg_we(1) = '1') then
   co2_ip_areg19  <= co2_aresult;
  end if;
  if ( co2_iareg_we(2) = '1') then
   co2_ip_areg21  <= co2_aresult;
  end if;
  if ( co2_iareg_we(3) = '1') then
   co2_ip_areg23  <= co2_aresult;
  end if;
  if ( co2_iareg_we(4) = '1') then
   co2_ip_areg25  <= co2_aresult;
  end if;
  if ( co2_iareg_we(5) = '1') then
   co2_ip_areg27  <= co2_aresult;
  end if;
  if ( co2_iareg_we(6) = '1') then
   co2_ip_areg29  <= co2_aresult;
  end if;
  if ( co2_iareg_we(7) = '1') then
   co2_ip_areg31  <= co2_aresult;
  end if;
  if ( co2_iareg_we(8) = '1') then
   co2_ip_areg33  <= co2_aresult;
  end if;
  if ( co2_iareg_we(9) = '1') then
   co2_ip_areg35  <= co2_aresult;
  end if;
  if ( co2_iareg_we(10) = '1') then
   co2_ip_areg37  <= co2_aresult;
  end if;
  if ( co2_iareg_we(11) = '1') then
   co2_ip_areg39 <= co2_aresult;
  end if;
  if ( co2_iareg_we(12) = '1') then
   co2_ip_areg41 <= co2_aresult;
  end if;
  if ( co2_iareg_we(13) = '1') then
   co2_ip_areg43 <= co2_aresult;
  end if;
  if ( co2_iareg_we(14) = '1') then
```

```
       co2_ip_areg45 <= co2_aresult;
      end if;
      if ( co2_iareg_we(15) = '1') then
       co2_ip_areg47 <= co2_aresult;
      end if;
      if ( co2_iareg_we(16) = '1') then
       co2_ip_areg49 <= co2_aresult;
      end if;
      if ( co2_iareg_we(17) = '1') then
       co2_ip_areg51 <= co2_aresult;
      end if;
      if ( co2_iareg_we(18) = '1') then
       co2_ip_areg53 <= co2_aresult;
      end if;
      if ( co2_iareg_we(19) = '1') then
       co2_ip_areg55 <= co2_aresult;
      end if;
      if ( co2_iareg_we(20) = '1') then
       co2_ip_areg57 <= co2_aresult;
      end if;
    end if;
    end if;
  end process;

--
*************************************************************************
*******
--* Pipeline Mux Logic
*
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     co2_muxa_data_p <= (others => '0');
     co2_muxb_data_p <= (others => '0');
     co2_muxc_data_p <= (others => '0');
     co2_muxd_data_p <= (others => '0');
    else if (hclk 'event and hclk = '1') then
     co2_muxa_data_p <= co2_muxa_data;
     co2_muxb_data_p <= co2_muxb_data;
     co2_muxc_data_p <= co2_muxc_data;
     co2_muxd_data_p <= co2_muxd_data;
    end if;
    end if;
  end process;
--
*************************************************************************
*******
--* PH Extraction Block Multiplier Instantiation
*
--
*************************************************************************
*******
  co2_mul:
    cph_analyzer_mul
      port map(
               hclk     => hclk,
               hreset_n => hreset_n,
               op_a     => co2_muxa_data_p,
```

```
               op_b     => co2_muxb_data_p,
               result   => co2_mresult
              );

  co2_add:
    cph_analyzer_add
      port map(
               hclk     => hclk,
               hreset_n => hreset_n,
               op_a     => co2_muxc_data_p,
               op_b     => co2_muxd_data_p,
               result   => co2_aresult
              );


end Behavioral;

--
**********************************************************************
*******
--* File Name  : cph_analyzer_csync.vhd
*
--* Designer   : Jaideep Chandran
*
--* Date       : 10/03/2011
*
--* Description:
*
--
**********************************************************************
*******
--
**********************************************************************
*******
--* Include Libraries
*
--
**********************************************************************
*******
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--
**********************************************************************
*******
--* Entity Decleration, I/O ports
*
--
**********************************************************************
*******
entity cph_analyzer_csync is
    Port (
           hclk             : in  STD_LOGIC;
           hreset_n         : in  STD_LOGIC;
           bmp_cstart       : in  STD_LOGIC;
           bmp_phcext_dn    : in  STD_LOGIC;
           bmp_phext_dn     : in  STD_LOGIC;
           bmp_pco2cext_dn  : in  STD_LOGIC;
           bmp_pco2phext_dn : in  STD_LOGIC;
```

```
            bmp_pco2ext_dn   : in  STD_LOGIC;
            bmp_iph_sext     : out STD_LOGIC;
            bmp_ipco2_sext   : out STD_LOGIC;
            bmp_icext_strt   : out STD_LOGIC;
            bmp_iph_ana      : out STD_LOGIC;
            bmp_ipc02_ana    : out STD_LOGIC;
            bmp_ca_dn        : out STD_LOGIC);

end cph_analyzer_csync;

--
*************************************************************************
*******
--* Main Block Logic
*
--
*************************************************************************
*******
architecture Behavioral of cph_analyzer_csync is
  Type state is (BMP_IA_START   , BMP_IPH_CEXT     , BMP_IPH_EXT,
BMP_IPCO2_CEXT,
                 BMP_IPCO2_PHEXT, BMP_IPCO2_PCO2EXT, BMP_IA_DN);
  signal p_state, n_state: state;
begin
--
*************************************************************************
*******
--* State Machine Sequential Part
*
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     p_state <= BMP_IA_START;
    else if (hclk 'event and hclk = '1') then
    p_state <= n_state;
   end if;
    end if;
  end process;

--
*************************************************************************
*******
--* State Machine Combinational Part
*
--
*************************************************************************
*******
  process (bmp_cstart       , bmp_phcext_dn , bmp_phext_dn,
bmp_pco2cext_dn ,
           bmp_pco2phext_dn , bmp_pco2ext_dn, p_state)
  begin
    case p_state is
     when BMP_IA_START    =>
      if (bmp_cstart = '1') then
        n_state <= BMP_IPH_CEXT;
      else
        n_state <= BMP_IA_START;
      end if;
```

```
    when BMP_IPH_CEXT   =>
      if (bmp_phcext_dn = '1') then
        n_state <= BMP_IPH_EXT;
      else
        n_state <= BMP_IPH_CEXT;
      end if;
    when BMP_IPH_EXT  =>
      if (bmp_phext_dn = '1') then
        n_state <= BMP_IPCO2_CEXT;
      else
        n_state <= BMP_IPH_EXT;
      end if;
    when BMP_IPCO2_CEXT =>
      if (bmp_pco2cext_dn = '1') then
        n_state <= BMP_IPCO2_PHEXT;
      else
        n_state <= BMP_IPCO2_CEXT;
      end if;
    when BMP_IPCO2_PHEXT  =>
      if (bmp_pco2phext_dn = '1') then
        n_state <= BMP_IPCO2_PCO2EXT;
      else
        n_state <= BMP_IPCO2_PHEXT;
      end if;
    when BMP_IPCO2_PCO2EXT    =>
      if (bmp_pco2ext_dn = '1') then
        n_state <= BMP_IA_DN;
      else
        n_state <= BMP_IPCO2_PCO2EXT;
      end if;
    when BMP_IA_DN    =>
        n_state <= BMP_IA_START;
    when others   => n_state <= BMP_IA_START;
   end case;
  end process;

--
*************************************************************************
*******
--* Output assignment for synchronization
*
--
*************************************************************************
*******
  process (p_state)
  begin
    case p_state is
     when BMP_IA_START      =>
       bmp_iph_sext   <= '0';
       bmp_ipco2_sext <= '0';
       bmp_icext_strt <= '0';
       bmp_ca_dn      <= '0';
       bmp_iph_ana    <= '0';
       bmp_ipc02_ana  <= '0';
     when BMP_IPH_CEXT      =>
       bmp_iph_sext   <= '0';
       bmp_ipco2_sext <= '0';
       bmp_icext_strt <= '1';
       bmp_ca_dn      <= '0';
       bmp_iph_ana    <= '1';
       bmp_ipc02_ana  <= '0';
```

```
      when BMP_IPH_EXT       =>
        bmp_iph_sext   <= '1';
        bmp_ipco2_sext <= '0';
        bmp_icext_strt <= '0';
        bmp_ca_dn      <= '0';
        bmp_iph_ana    <= '1';
        bmp_ipc02_ana  <= '0';
      when BMP_IPCO2_CEXT    =>
        bmp_iph_sext   <= '0';
        bmp_ipco2_sext <= '0';
        bmp_icext_strt <= '1';
        bmp_ca_dn      <= '0';
        bmp_iph_ana    <= '0';
        bmp_ipc02_ana  <= '1';
      when BMP_IPCO2_PHEXT   =>
        bmp_iph_sext   <= '1';
        bmp_ipco2_sext <= '0';
        bmp_icext_strt <= '0';
        bmp_ca_dn      <= '0';
        bmp_iph_ana    <= '0';
        bmp_ipc02_ana  <= '1';
      when BMP_IPCO2_PCO2EXT =>
        bmp_iph_sext   <= '0';
        bmp_ipco2_sext <= '1';
        bmp_icext_strt <= '0';
        bmp_ca_dn      <= '0';
        bmp_iph_ana    <= '0';
        bmp_ipc02_ana  <= '1';
      when BMP_IA_DN         =>
        bmp_iph_sext   <= '0';
        bmp_ipco2_sext <= '0';
        bmp_icext_strt <= '0';
        bmp_ca_dn      <= '1';
        bmp_iph_ana    <= '0';
        bmp_ipc02_ana  <= '0';
      when others            =>
        bmp_iph_sext   <= '0';
        bmp_ipco2_sext <= '0';
        bmp_icext_strt <= '0';
        bmp_ca_dn      <= '0';
        bmp_iph_ana    <= '0';
        bmp_ipc02_ana  <= '0';
    end case;
  end process;

end Behavioral;

--
*********************************************************************
*******
--* File Name  : cph_analyzer_ctbl.vhd
*
--* Designer   : Jaideep Chandran
*
--* Date       : 10/03/2011
*
--* Description: BMP image Red Color Table.
*
--
*********************************************************************
*******
```

```
--
**********************************************************************
*******
--* Include Libraries
*
--
**********************************************************************
*******
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--
**********************************************************************
*******
--* Entity Decleration, I/O ports
*
--
**********************************************************************
*******
entity cph_analyzer_ctbl is
    Port ( hclk            : in  STD_LOGIC;
           bmp_ram_we      : in  STD_LOGIC;
           bmp_ram_addr    : in  STD_LOGIC_VECTOR (7 downto 0);
           bmp_ram_datain  : in  STD_LOGIC_VECTOR (7 downto 0);
           bmp_ram_dataout : out STD_LOGIC_VECTOR (7 downto 0));
end cph_analyzer_ctbl;

--
**********************************************************************
*******
--* Main Block Logic
*
--
**********************************************************************
*******
architecture Behavioral of cph_analyzer_ctbl is

--
**********************************************************************
*******
--* Signal Decleration
*
--
**********************************************************************
*******
  type mem_ram is array (0 to 255) of STD_LOGIC_VECTOR (7 downto 0);
  signal memory : mem_ram ;

--
**********************************************************************
*******
--* Process to write data in memory
*
--
**********************************************************************
*******
begin
  process (hclk , bmp_ram_we)
```

```
  begin
    if (hclk 'event and hclk = '1')
   then
      if (bmp_ram_we = '1')
    then
      memory (conv_integer (bmp_ram_addr)) <= bmp_ram_datain;
    end if;
    end if;
  end process;

--
*********************************************************************
*******
--* Read data from memory
*
--
*********************************************************************
*******
  bmp_ram_dataout <= memory (conv_integer (bmp_ram_addr));

end Behavioral;


--
*********************************************************************
*******
--* File Name  : cph_analyzer_ictl.vhd
*
--* Designer   : Jaideep Chandran
*
--* Date       : 10/03/2011
*
--* Description: BMP image color Analysis contol logic.
*
--
*********************************************************************
*******
--
*********************************************************************
*******
--* Include Libraries
*
--
*********************************************************************
*******
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--
*********************************************************************
*******
--* Entity Decleration, I/O ports
*
--
*********************************************************************
*******
entity cph_analyzer_ictl is
    Port ( hclk            : in STD_LOGIC;
           hreset_n        : in STD_LOGIC;
```

```
            bmp_msaddr_ph   : in STD_LOGIC_VECTOR (31 downto 0);
            bmp_msaddr_co2  : in STD_LOGIC_VECTOR (31 downto 0);

            bmp_mem_data_in : in STD_LOGIC_VECTOR (7 downto 0);
            bmp_icext_strt  : in STD_LOGIC;
            bmp_iph_ana     : in STD_LOGIC;
            bmp_ipc02_ana   : in STD_LOGIC;
            bmp_phcext_dn   : out  STD_LOGIC;
            bmp_pco2cext_dn : out  STD_LOGIC;
            bmp_mem_re      : out STD_LOGIC;
            bmp_mem_addr    : out STD_LOGIC_VECTOR (31 downto 0);
            bmp_ram_we      : out STD_LOGIC;
            bmp_ram_addr    : out STD_LOGIC_VECTOR (7 downto 0);
            bmp_ram_datain  : out STD_LOGIC_VECTOR (7 downto 0);
            bmp_strt_val    : out STD_LOGIC;
            bmp_sr_sav      : out STD_LOGIC;
            bmp_hid_reg_ph  : out STD_LOGIC_VECTOR (15 downto 0);
            bmp_fs_reg_ph   : out STD_LOGIC_VECTOR (31 downto 0);
            bmp_fhi_reg_ph  : out STD_LOGIC_VECTOR (7 downto 0);
            bmp_ih_reg_ph   : out STD_LOGIC_VECTOR (31 downto 0);
            bmp_iw_reg_ph   : out STD_LOGIC_VECTOR (31 downto 0);
            bmp_pw_reg_ph   : out STD_LOGIC_VECTOR (7 downto 0);
            bmp_hid_reg_co2 : out STD_LOGIC_VECTOR (15 downto 0);
            bmp_fs_reg_co2  : out STD_LOGIC_VECTOR (31 downto 0);
            bmp_fhi_reg_co2 : out STD_LOGIC_VECTOR (7 downto 0);
            bmp_ih_reg_co2  : out STD_LOGIC_VECTOR (31 downto 0);
            bmp_iw_reg_co2  : out STD_LOGIC_VECTOR (31 downto 0);
            bmp_pw_reg_co2  : out STD_LOGIC_VECTOR (7 downto 0));
end cph_analyzer_ictl;

--
*********************************************************************
*******
--* Main Block Logic
*
--
*********************************************************************
*******
architecture Behavioral of cph_analyzer_ictl is
  Type state is (START, BMPFIF, BMPCTBL, BMPPDATA, BMPCDAT, BMPDN);
  signal p_state, n_state: state;
  signal bmp_fif_dn     : STD_LOGIC;
  signal bmp_ctbl_dn    : STD_LOGIC;
  signal bmp_pdat_dn    : STD_LOGIC;
  signal bmp_ram_we_p   : STD_LOGIC;
  signal sys_init_cnt   :STD_LOGIC_VECTOR (4  downto 0);
  signal sys_init_addr  :STD_LOGIC_VECTOR (31 downto 0);
  signal sys_ictbl_addr :STD_LOGIC_VECTOR (31 downto 0);
  signal sys_ictbl_cnt  :STD_LOGIC_VECTOR (7  downto 0);
  signal bmp_pix_addr   :STD_LOGIC_VECTOR (31 downto 0);
  signal bmp_reg_we     :STD_LOGIC_VECTOR (19 downto 0);
  signal bmp_pcol_loc   :STD_LOGIC_VECTOR (9 downto 0);
  signal bmp_prow_loc   :STD_LOGIC_VECTOR (10 downto 0);
  signal bmp_mstrt_addr :STD_LOGIC_VECTOR (31 downto 0);

begin
--
*********************************************************************
*******
--* State Machine Sequential Part
*
```

```
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     p_state <= START;
    else if (hclk 'event and hclk = '1') then
    p_state <= n_state;
   end if;
    end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_mstrt_addr <= (others => '0');
    else if (hclk 'event and hclk = '1') then
         if (bmp_ipc02_ana = '1') then
        bmp_mstrt_addr <= bmp_msaddr_co2;
            else
              bmp_mstrt_addr <= bmp_msaddr_ph;
            end if;
    end if;
    end if;
  end process;

--
*************************************************************************
*******
--* State Machine Combinational Part
*
--
*************************************************************************
*******
  process (bmp_fif_dn , bmp_ctbl_dn, bmp_pdat_dn, p_state,
bmp_icext_strt)
  begin
    case p_state is
     when START    =>
      if (bmp_icext_strt = '1') then
        n_state <= BMPFIF;
      else
        n_state <= START;
      end if;
    when BMPFIF   =>
      if (bmp_fif_dn = '1') then
        n_state <= BMPCTBL;
      else
        n_state <= BMPFIF;
      end if;
    when BMPCTBL  =>
      if (bmp_ctbl_dn = '1') then
        n_state <= BMPPDATA;
      else
        n_state <= BMPCTBL;
      end if;
    when BMPPDATA =>
      if (bmp_pdat_dn = '1') then
        n_state <= BMPCDAT;
      else
```

```
        n_state <= BMPPDATA;
      end if;
    when BMPCDAT  =>
          n_state <= BMPDN;
    when BMPDN    =>
        n_state <= START;
    when others   => n_state <= START;
   end case;
  end process;

--
*************************************************************************
*******
--*                       External Memory Address
*
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_mem_addr <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if ( p_state = BMPFIF and bmp_fif_dn = '0') then
        bmp_mem_addr <= bmp_mstrt_addr + sys_init_addr;
      else if ( (p_state = BMPCTBL and bmp_ctbl_dn = '0') or
                ( p_state = BMPFIF and bmp_fif_dn = '1')) then
        bmp_mem_addr <= bmp_mstrt_addr + sys_ictbl_addr;
      else if (p_state = BMPPDATA or
        (p_state = BMPCTBL and bmp_ctbl_dn = '1')) then
      bmp_mem_addr <= bmp_mstrt_addr + bmp_pix_addr + (x"00" & "000" &
bmp_prow_loc & bmp_pcol_loc);
    end if;
    end if;
    end if;
   end if;
    end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_mem_re <= '0';
    else if (hclk 'event and hclk = '1') then
      if ( p_state = BMPFIF or p_state = BMPCTBL or p_state =
BMPPDATA) then
      bmp_mem_re <= '1';
    else
      bmp_mem_re <= '0';
    end if;
   end if;
    end if;
  end process;

--
*************************************************************************
*******
--*                       System Initialization Phase
*
--*                       ===========================
*
```

```
--
************************************************************************
*******
--* Read Values of These Register
*
--* BMPHID --> bmp_hid_reg
*
--* BMPFS  --> bmp_fs_reg
*
--* BMPFHI --> bmp_fhi_reg
*
--* BMPIH  --> bmp_ih_reg
*
--* BMPIW  --> bmp_iw_reg
*
--* BMPPW  --> bmp_pw_reg
*
--
************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
      bmp_phcext_dn   <= '0';
      bmp_pco2cext_dn <= '0';
    else if (hclk 'event and hclk = '1') then
      if ( p_state = BMPCDAT and bmp_iph_ana = '1') then
        bmp_phcext_dn<= '1';
            else
              bmp_phcext_dn <= '0';
            end if;
      if ( p_state = BMPCDAT and bmp_ipc02_ana = '1') then
        bmp_pco2cext_dn <= '1';
            else
              bmp_pco2cext_dn <= '0';
            end if;

   end if;
    end if;
  end process;

  process (p_state)
  begin
    case (p_state) is
     when BMPCDAT  => bmp_sr_sav      <= '1';
                      bmp_strt_val    <= '0';
     when BMPPDATA => bmp_sr_sav   <= '0';
                      bmp_strt_val <= '1';
     when BMPDN    => bmp_sr_sav   <= '0';
                      bmp_strt_val <= '0';

      when others  => bmp_sr_sav   <= '0';
                      bmp_strt_val <= '0';
    end case;
  end process;

  process (sys_init_cnt)
  begin
    case (sys_init_cnt) is
     when "10100" => bmp_fif_dn <= '1';
      when others  => bmp_fif_dn <= '0';
```

```
  end case;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   sys_init_cnt <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if ( p_state = BMPFIF) then
  sys_init_cnt <= sys_init_cnt + 1;
  end if;
 end if;
  end if;
end process;

process(sys_init_cnt)
begin
  case (sys_init_cnt) is
    when "00000"  => sys_init_addr <= x"00000000"; --BMPHID
    when "00001"  => sys_init_addr <= x"00000001"; --
  when "00010"  => sys_init_addr <= x"00000002"; --BMPFS
    when "00011"  => sys_init_addr <= x"00000003"; --
    when "00100"  => sys_init_addr <= x"00000004"; --
    when "00101"  => sys_init_addr <= x"00000005"; --
  when "00110"  => sys_init_addr <= x"0000000A"; -- bmp_pix_addr
    when "00111"  => sys_init_addr <= x"0000000B"; --
    when "01000"  => sys_init_addr <= x"0000000C"; --
    when "01001"  => sys_init_addr <= x"0000000D"; --
    when "01010"  => sys_init_addr <= x"0000000E";  -- BMPFHI
    when "01011"  => sys_init_addr <= x"00000012";  -- BMPIH
    when "01100"  => sys_init_addr <= x"00000013";
    when "01101"  => sys_init_addr <= x"00000014";
    when "01110"  => sys_init_addr <= x"00000015";
    when "01111"  => sys_init_addr <= x"00000016";  -- BMPIW
  when "10000"  => sys_init_addr <= x"00000017";
    when "10001"  => sys_init_addr <= x"00000018";
  when "10010"  => sys_init_addr <= x"00000019";
  when "10011"  => sys_init_addr <= x"0000001A";    -- BMPPW
    when others   => sys_init_addr <= x"00000000";
  end case;
end process;

process(hclk , hreset_n)
begin
  if (hreset_n = '0') then
   bmp_reg_we <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    case (sys_init_cnt) is
      when "00000" => bmp_reg_we <= x"00001"; --BMPHID
      when "00001" => bmp_reg_we <= x"00002"; --
      when "00010" => bmp_reg_we <= x"00004"; --BMPFS
      when "00011" => bmp_reg_we <= x"00008"; --
      when "00100" => bmp_reg_we <= x"00010"; --
      when "00101" => bmp_reg_we <= x"00020"; --
      when "00110" => bmp_reg_we <= x"00040"; -- bmp_pix_addr
      when "00111" => bmp_reg_we <= x"00080";
      when "01000" => bmp_reg_we <= x"00100";
      when "01001" => bmp_reg_we <= x"00200";
      when "01010" => bmp_reg_we <= x"00400"; -- BMPFHI
      when "01011" => bmp_reg_we <= x"00800"; -- BMPIH
      when "01100" => bmp_reg_we <= x"01000";
```

```
      when "01101" => bmp_reg_we <= x"02000";
      when "01110" => bmp_reg_we <= x"04000";
      when "01111" => bmp_reg_we <= x"08000"; -- BMPIW
    when "10000" => bmp_reg_we <= x"10000";
      when "10001" => bmp_reg_we <= x"20000";
    when "10010" => bmp_reg_we <= x"40000";
    when "10011" => bmp_reg_we <= x"80000"; -- BMPPW
      when others  => bmp_reg_we <= x"00000";
    end case;
end if;
 end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   bmp_hid_reg_ph <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if (bmp_reg_we(0) = '1' and bmp_iph_ana = '1') then
    bmp_hid_reg_ph (7 downto 0) <= bmp_mem_data_in;
  end if;
    if (bmp_reg_we(1) = '1' and bmp_iph_ana = '1') then
    bmp_hid_reg_ph (15 downto 8) <= bmp_mem_data_in;
   end if;
  end if;
 end if;
end process;


process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   bmp_fs_reg_ph <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if (bmp_reg_we(2) = '1' and bmp_iph_ana = '1') then
    bmp_fs_reg_ph (7 downto 0) <= bmp_mem_data_in;
  end if;
    if (bmp_reg_we(3) = '1' and bmp_iph_ana = '1') then
    bmp_fs_reg_ph (15 downto 8) <= bmp_mem_data_in;
   end if;
    if (bmp_reg_we(4) = '1' and bmp_iph_ana = '1') then
    bmp_fs_reg_ph (23 downto 16) <= bmp_mem_data_in;
  end if;
    if (bmp_reg_we(5) = '1' and bmp_iph_ana = '1') then
    bmp_fs_reg_ph (31 downto 24) <= bmp_mem_data_in;
   end if;
  end if;
 end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   bmp_pix_addr <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if (bmp_reg_we(6) = '1') then
    bmp_pix_addr(7 downto 0) <= bmp_mem_data_in;
  end if;
    if (bmp_reg_we(7) = '1') then
    bmp_pix_addr(15 downto 8) <= bmp_mem_data_in;
   end if;
```

```
    if (bmp_reg_we(8) = '1') then
    bmp_pix_addr(23 downto 16) <= bmp_mem_data_in;
  end if;
    if (bmp_reg_we(9) = '1') then
    bmp_pix_addr(31 downto 24) <= bmp_mem_data_in;
   end if;
  end if;
 end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   bmp_fhi_reg_ph <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if (bmp_reg_we(10) = '1' and bmp_iph_ana = '1') then
    bmp_fhi_reg_ph (7 downto 0) <= bmp_mem_data_in;
  end if;
  end if;
 end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   bmp_ih_reg_ph <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if (bmp_reg_we(11) = '1' and bmp_iph_ana = '1') then
    bmp_ih_reg_ph (7 downto 0) <= bmp_mem_data_in;
  end if;
    if (bmp_reg_we(12) = '1' and bmp_iph_ana = '1') then
    bmp_ih_reg_ph (15 downto 8) <= bmp_mem_data_in;
   end if;
    if (bmp_reg_we(13) = '1' and bmp_iph_ana = '1') then
    bmp_ih_reg_ph (23 downto 16) <= bmp_mem_data_in;
  end if;
    if (bmp_reg_we(14) = '1' and bmp_iph_ana = '1') then
    bmp_ih_reg_ph (31 downto 24) <= bmp_mem_data_in;
   end if;
  end if;
 end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   bmp_iw_reg_ph <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if (bmp_reg_we(15) = '1' and bmp_iph_ana = '1') then
    bmp_iw_reg_ph (7 downto 0) <= bmp_mem_data_in;
  end if;
    if (bmp_reg_we(16) = '1' and bmp_iph_ana = '1') then
    bmp_iw_reg_ph (15 downto 8) <= bmp_mem_data_in;
   end if;
    if (bmp_reg_we(17) = '1' and bmp_iph_ana = '1') then
    bmp_iw_reg_ph (23 downto 16) <= bmp_mem_data_in;
  end if;
    if (bmp_reg_we(18) = '1' and bmp_iph_ana = '1') then
    bmp_iw_reg_ph (31 downto 24) <= bmp_mem_data_in;
   end if;
  end if;
```

```
 end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   bmp_pw_reg_ph <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if (bmp_reg_we(19) = '1' and bmp_iph_ana = '1') then
    bmp_pw_reg_ph (7 downto 0) <= bmp_mem_data_in;
  end if;
  end if;
 end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   bmp_hid_reg_co2 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if (bmp_reg_we(0) = '1' and bmp_ipc02_ana = '1') then
    bmp_hid_reg_co2 (7 downto 0) <= bmp_mem_data_in;
  end if;
    if (bmp_reg_we(1) = '1' and bmp_ipc02_ana = '1') then
    bmp_hid_reg_co2 (15 downto 8) <= bmp_mem_data_in;
   end if;
  end if;
 end if;
end process;


process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   bmp_fs_reg_co2 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if (bmp_reg_we(2) = '1' and bmp_ipc02_ana = '1') then
    bmp_fs_reg_co2 (7 downto 0) <= bmp_mem_data_in;
  end if;
    if (bmp_reg_we(3) = '1' and bmp_ipc02_ana = '1') then
    bmp_fs_reg_co2 (15 downto 8) <= bmp_mem_data_in;
   end if;
    if (bmp_reg_we(4) = '1' and bmp_ipc02_ana = '1') then
    bmp_fs_reg_co2 (23 downto 16) <= bmp_mem_data_in;
  end if;
    if (bmp_reg_we(5) = '1' and bmp_ipc02_ana = '1') then
    bmp_fs_reg_co2 (31 downto 24) <= bmp_mem_data_in;
   end if;
  end if;
 end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   bmp_fhi_reg_co2 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if (bmp_reg_we(10) = '1' and bmp_ipc02_ana = '1') then
    bmp_fhi_reg_co2 (7 downto 0) <= bmp_mem_data_in;
  end if;
  end if;
```

```
   end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_ih_reg_co2 <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if (bmp_reg_we(11) = '1' and bmp_ipc02_ana = '1') then
      bmp_ih_reg_co2 (7 downto 0) <= bmp_mem_data_in;
    end if;
      if (bmp_reg_we(12) = '1' and bmp_ipc02_ana = '1') then
      bmp_ih_reg_co2 (15 downto 8) <= bmp_mem_data_in;
     end if;
      if (bmp_reg_we(13) = '1' and bmp_ipc02_ana = '1') then
      bmp_ih_reg_co2 (23 downto 16) <= bmp_mem_data_in;
    end if;
      if (bmp_reg_we(14) = '1' and bmp_ipc02_ana = '1') then
      bmp_ih_reg_co2 (31 downto 24) <= bmp_mem_data_in;
     end if;
    end if;
   end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_iw_reg_co2 <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if (bmp_reg_we(15) = '1' and bmp_ipc02_ana = '1') then
      bmp_iw_reg_co2 (7 downto 0) <= bmp_mem_data_in;
    end if;
      if (bmp_reg_we(16) = '1' and bmp_ipc02_ana = '1') then
      bmp_iw_reg_co2 (15 downto 8) <= bmp_mem_data_in;
     end if;
      if (bmp_reg_we(17) = '1' and bmp_ipc02_ana = '1') then
      bmp_iw_reg_co2 (23 downto 16) <= bmp_mem_data_in;
    end if;
      if (bmp_reg_we(18) = '1' and bmp_ipc02_ana = '1') then
      bmp_iw_reg_co2 (31 downto 24) <= bmp_mem_data_in;
     end if;
    end if;
   end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_pw_reg_co2 <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if (bmp_reg_we(19) = '1' and bmp_ipc02_ana = '1') then
      bmp_pw_reg_co2 (7 downto 0) <= bmp_mem_data_in;
    end if;
    end if;
   end if;
  end process;

--
*************************************************************************
*******
```

```
--*                       System Initialization Phase
*
--*                       ===========================
*
--
************************************************************************
*******
--* Read Colour Table Values From BMP File and Save in Colour Table
RAM       *
--
************************************************************************
*******
  bmp_ram_we      <= bmp_ram_we_p;
  bmp_ram_datain  <= bmp_mem_data_in;


  process (sys_ictbl_cnt , bmp_mem_data_in, p_state)
  begin
    case p_state is
      when BMPCTBL  =>
        bmp_ram_addr    <= sys_ictbl_cnt;
      when BMPPDATA =>
        bmp_ram_addr    <= bmp_mem_data_in;
      when BMPCDAT  =>
        bmp_ram_addr    <= bmp_mem_data_in;
      when others   =>
        bmp_ram_addr    <= (others => '0');
    end case;
  end process;

  process (sys_ictbl_cnt)
  begin
    case (sys_ictbl_cnt) is
     when "00000000" => bmp_ctbl_dn <= '1';
      when others     => bmp_ctbl_dn <= '0';
    end case;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_ram_we_p <= '0';
    else if (hclk 'event and hclk = '1') then
      if ( p_state = BMPFIF and bmp_fif_dn = '1') then
      bmp_ram_we_p <= '1';
    end if;
      if ( p_state = BMPCTBL and bmp_ctbl_dn = '1') then
      bmp_ram_we_p <= '0';
    end if;
   end if;
    end if;
  end process;


  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     sys_ictbl_cnt <= (others => '1');
    else if (hclk 'event and hclk = '1') then
      if ( p_state = BMPFIF and bmp_fif_dn = '1') then
    sys_ictbl_cnt <= (others => '1');
```

```
    end if;
      if ( p_state = BMPCTBL) then
    sys_ictbl_cnt <= sys_ictbl_cnt - 1;
    end if;
   end if;
    end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     sys_ictbl_addr <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if (p_state = BMPFIF) then
      sys_ictbl_addr <= bmp_pix_addr - 4;
    end if;
      if (p_state = BMPCTBL or (p_state = BMPFIF and bmp_fif_dn =
'1')) then
      sys_ictbl_addr <= sys_ictbl_addr - 4;
    end if;
   end if;
    end if;
  end process;

--
*************************************************************************
*******
--*                       Normal Mode Operation
*
--*                       ======================
*
--
*************************************************************************
*******
--* Read Pixel Data Value from BMP File and calculate the sum of all
pixel    *
--
*************************************************************************
*******
  process (bmp_pcol_loc, bmp_prow_loc)
  begin
    if (bmp_prow_loc = "01011100100" and bmp_pcol_loc = "1001100100")
then
     bmp_pdat_dn <= '1';
   else
     bmp_pdat_dn <= '0';
    end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_pcol_loc <= "0110011100"; --// (412)d = (19C)H
    else if (hclk 'event and hclk = '1') then
        if ((p_state = BMPPDATA and bmp_pcol_loc < "1001100100") or
            (p_state = BMPCTBL and bmp_ctbl_dn = '1')          )then
          bmp_pcol_loc <= bmp_pcol_loc + 1;
        else if (p_state = BMPPDATA and bmp_pcol_loc = "1001100100")
then
          bmp_pcol_loc <= "0110011100";
        else if (p_state = BMPDN) then
```

```
          bmp_pcol_loc <= "0110011100";
        end if;
        end if;
        end if;
   end if;
   end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_prow_loc <= "01000011100"; --// (540)d = (21C)H
    else if (hclk 'event and hclk = '1') then
      if (p_state = BMPPDATA and bmp_pcol_loc = "1001100100") then
      bmp_prow_loc <= bmp_prow_loc + 1;
    end if;
      if (p_state = BMPDN) then
      bmp_prow_loc <= "01000011100";
    end if;
   end if;
    end if;
  end process;

end Behavioral;


--
**********************************************************************
*******
--* File Name  : cph_analyzer_mul.vhd
*
--* Designer   : Jaideep Chandran
*
--* Date       : 10/03/2011
*
--* Description: Floating Point Multiplier.
*
--
**********************************************************************
*******
--
**********************************************************************
*******
--* Include Libraries
*
--
**********************************************************************
*******
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--
**********************************************************************
*******
--* Entity Decleration, I/O ports
*
--
**********************************************************************
*******
```

```
entity cph_analyzer_mul is
  port (
        hclk     : in std_logic;
              hreset_n : in std_logic;
              op_a     : in std_logic_vector(31 downto 0);
              op_b     : in std_logic_vector(31 downto 0);
              result   : out std_logic_vector(31 downto 0));
end cph_analyzer_mul;

--
************************************************************************
*******
--* Main Block Logic
*
--
************************************************************************
*******
architecture Behavioral of cph_analyzer_mul is

signal x_man,y_man: std_logic_vector(23 downto 0);
signal z_man : std_logic_vector(22 downto 0);
--signal z_man1 : std_logic_vector(23 downto 0);
signal prod : std_logic_vector(47 downto 0);
signal shift : std_logic_vector(1 downto 0);
signal x_exp,y_exp,z_exp : std_logic_vector(7 downto 0);
signal x0,y0,x_sign,y_sign,z_sign : std_logic;
--constant f0 : std_logic_vector(31 downto 0) : = "1
begin
x_sign <= op_a(31);
y_sign <= op_b(31);
x_exp <=  op_a(30 downto 23);
y_exp <=  op_b(30 downto 23);
x_man <= '1' & op_a(22 downto 0);
y_man <= '1' & op_b(22 downto 0);



P1:process(op_a,op_b,hclk,hreset_n,z_man)
variable z_exp1,z_exp2,z_exp3 : std_logic_vector(8 downto 0);
--variable exp_ovf,exp_undf: std_logic;
constant sub : std_logic_vector(7 downto 0) := "01111111";

begin
if hreset_n = '0' then
result <= (others => '0');
elsif (hclk'event and hclk ='1') then
z_exp1 := unsigned('0' & x_exp)+ unsigned('0' & y_exp);
z_exp2 := unsigned(z_exp1) + unsigned("000000" & shift);
z_exp3 := unsigned(z_exp2) - unsigned(sub);
z_sign <= x_sign xor y_sign;
prod <= (x_man) * (y_man);
if prod(47) = '1' then
  shift <= "01" ;
else
  shift <= "00" ;
end if ;
if shift = "01" then
z_man <= prod(46 downto 24);
else
z_man <= prod(45 downto 23);
end if;
```

```
--z_man <= z_man1(22 downto 0);
result <= z_sign & z_exp3(7 downto 0) & z_man;
end if;
end process;

end Behavioral;


--
**********************************************************************
*******
--* File Name  : cph_analyzer_phcal.vhd
*
--* Designer   : Jaideep Chandran
*
--* Date       : 10/03/2011
*
--* Description:
*
--
**********************************************************************
*******
--
**********************************************************************
*******
--* Include Libraries
*
--
**********************************************************************
*******
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--
**********************************************************************
*******
--* Entity Decleration, I/O ports
*
--
**********************************************************************
*******
entity cph_analyzer_phcal is
    Port (
           hclk             : in  STD_LOGIC;
           hreset_n         : in  STD_LOGIC;
           bmp_iph_ana      : in STD_LOGIC;
           bmp_ipc02_ana    : in STD_LOGIC;
           bmp_iph_sext     : in  STD_LOGIC;
           bmp_red_mv       : in  STD_LOGIC_VECTOR (31 downto 0);
           bmp_red_mf       : in  STD_LOGIC_VECTOR (31 downto 0);
           bmp_phext_dn     : out STD_LOGIC;
                    bmp_pco2phext_dn : out STD_LOGIC;
           cph_ph_val       : out STD_LOGIC_VECTOR (31 downto 0);
           bmp_iph_val      : out STD_LOGIC_VECTOR (31 downto 0));


end cph_analyzer_phcal;
```

```
--
************************************************************************
*******
--* Main Block Logic
*
--
************************************************************************
*******
architecture Behavioral of cph_analyzer_phcal is

  signal ph_cred_val  : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_mreg0  : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_mreg1  : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_mreg2  : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_mreg3  : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_mreg4  : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_mreg5  : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_mreg6  : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_mreg7  : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_mreg8  : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_areg9  : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_areg10 : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_areg11 : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_areg12 : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_areg13 : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_const_a   : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_const_b   : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_const_c   : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_const_d   : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_const_e   : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_const_f   : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_ip_mcnt   : STD_LOGIC_VECTOR (3 downto 0);
  signal ph_ip_acnt   : STD_LOGIC_VECTOR (2 downto 0);
  signal ph_mresult   : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_aresult   : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_muxa_data : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_muxb_data : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_muxc_data : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_muxd_data : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_muxa_sel  : STD_LOGIC_VECTOR (2 downto 0);
  signal ph_muxb_sel  : STD_LOGIC_VECTOR (2 downto 0);
  signal ph_ipreg_we  : STD_LOGIC_VECTOR (9 downto 0);
  signal ph_ipareg_we : STD_LOGIC_VECTOR (4 downto 0);

--
************************************************************************
*******
--* Pipeline Register
*
--
************************************************************************
*******

  signal ph_muxa_data_p : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_muxb_data_p : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_muxc_data_p : STD_LOGIC_VECTOR (31 downto 0);
  signal ph_muxd_data_p : STD_LOGIC_VECTOR (31 downto 0);

--
************************************************************************
*******
```

```
--* Component Decleration
*
--
*************************************************************************
*******
  component cph_analyzer_add is
    port (
          hclk     : in STD_LOGIC;
          hreset_n : in STD_LOGIC;
          op_a     : in STD_LOGIC_VECTOR (31 downto 0);
          op_b     : in STD_LOGIC_VECTOR (31 downto 0);
          result   : out std_logic_vector(31 downto 0));
  end component;

  component cph_analyzer_mul is
    port (
          hclk     : in STD_LOGIC;
          hreset_n : in STD_LOGIC;
          op_a     : in STD_LOGIC_VECTOR (31 downto 0);
          op_b     : in STD_LOGIC_VECTOR (31 downto 0);
          result   : out std_logic_vector(31 downto 0));
  end component;

begin
--
*************************************************************************
*******
--* Constant values used in equations calculation
*
--
*************************************************************************
*******
  ph_const_a  <= "11000001111011000110111010011000"; -- -29.554
  ph_const_b  <= "01000010100110001111000110101010"; --  76.472
  ph_const_c  <= "11000010100100110111111011111010"; -- -73.48
  ph_const_d  <= "01000010000000001011110101110001"; --  32.37
  ph_const_e  <= "11000000111100010100111111011111"; -- -07.541
  ph_const_f  <= "01000000111111001000000101101111"; --  07.8908
  cph_ph_val  <= ph_ip_areg13;
  bmp_iph_val <= ph_ip_areg13;

--
*************************************************************************
*******
--* Logic to Calculate the Intermediate product of multiplication
*
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     ph_ip_mcnt <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if ( bmp_iph_sext = '1' and ph_ip_mcnt < "1001") then
        ph_ip_mcnt <= ph_ip_mcnt + 1;
      else if ( bmp_iph_sext = '1' and ph_ip_mcnt = "1001") then
        ph_ip_mcnt <= ph_ip_mcnt;
      else
        ph_ip_mcnt <= (others => '0');
      end if;
```

```
      end if;
    end if;
    end if;
  end process;

--
*************************************************************************
*******
--* Multiplexer
*
--
*************************************************************************
*******
  process(ph_ip_mcnt)
  begin
     case (ph_ip_mcnt) is
       when "0000"  => ph_muxa_sel <= "000";
       when "0001"  => ph_muxa_sel <= "001";
       when "0010"  => ph_muxa_sel <= "010";
       when "0011"  => ph_muxa_sel <= "011";
       when "0100"  => ph_muxa_sel <= "100";
       when "0101"  => ph_muxa_sel <= "101";
       when "0110"  => ph_muxa_sel <= "110";
       when "0111"  => ph_muxa_sel <= "100";
       when "1000"  => ph_muxa_sel <= "011";
       when "1001"  => ph_muxa_sel <= "010";
       when others  => ph_muxa_sel <= (others => '0');
     end case;
  end process;

  process(ph_ip_mcnt)
  begin
     case (ph_ip_mcnt) is
       when "0000"  => ph_muxb_sel <= "000";
       when "0001"  => ph_muxb_sel <= "001";
       when "0010"  => ph_muxb_sel <= "001";
       when "0011"  => ph_muxb_sel <= "001";
       when "0100"  => ph_muxb_sel <= "001";
       when "0101"  => ph_muxb_sel <= "010";
       when "0110"  => ph_muxb_sel <= "011";
       when "0111"  => ph_muxb_sel <= "100";
       when "1000"  => ph_muxb_sel <= "101";
       when "1001"  => ph_muxb_sel <= "110";
       when others  => ph_muxb_sel <= (others => '0');
     end case;
  end process;

--
*************************************************************************
*******
--* Multiplexer for Multiplier OperandA
*
--
*************************************************************************
*******
  process(bmp_red_mv   , ph_cred_val , ph_ip_mreg0 , ph_ip_mreg1 ,
          ph_ip_mreg2  , ph_ip_mreg3 , ph_ip_mreg4 , ph_muxa_sel)
  begin
     case (ph_muxa_sel) is
       when "000"  => ph_muxa_data <= bmp_red_mv;
       when "001"  => ph_muxa_data <= ph_cred_val;
```

```
       when "010"  => ph_muxa_data <= ph_ip_mreg0;
       when "011"  => ph_muxa_data <= ph_ip_mreg1;
       when "100"  => ph_muxa_data <= ph_ip_mreg2;
       when "101"  => ph_muxa_data <= ph_ip_mreg3;
       when "110"  => ph_muxa_data <= ph_ip_mreg4;
       when others => ph_muxa_data <= ph_cred_val;
     end case;
  end process;

--
*************************************************************************
*******
--* Multiplexer for Multiplier OperandB
*
--
*************************************************************************
*******
  process(bmp_red_mf  , ph_cred_val , ph_const_a , ph_const_b ,
          ph_const_c  , ph_const_d  , ph_const_e , ph_muxb_sel)
  begin
     case (ph_muxb_sel) is
       when "000"  => ph_muxb_data <= bmp_red_mf;
       when "001"  => ph_muxb_data <= ph_cred_val;
       when "010"  => ph_muxb_data <= ph_const_a;
       when "011"  => ph_muxb_data <= ph_const_b;
       when "100"  => ph_muxb_data <= ph_const_c;
       when "101"  => ph_muxb_data <= ph_const_d;
       when "110"  => ph_muxb_data <= ph_const_e;
       when others => ph_muxb_data <= ph_cred_val;
     end case;
  end process;

--
*************************************************************************
*******
--* Logic to generate intermediate(multiplication) reg write enable
signals   *
--
*************************************************************************
*******
  process(ph_ip_mcnt)
  begin
     case (ph_ip_mcnt) is
       when "0000"  => ph_ipreg_we <= "0000000001";
       when "0001"  => ph_ipreg_we <= "0000000010";
       when "0010"  => ph_ipreg_we <= "0000000100";
       when "0011"  => ph_ipreg_we <= "0000001000";
       when "0100"  => ph_ipreg_we <= "0000010000";
       when "0101"  => ph_ipreg_we <= "0000100000";
       when "0110"  => ph_ipreg_we <= "0001000000";
       when "0111"  => ph_ipreg_we <= "0010000000";
       when "1000"  => ph_ipreg_we <= "0100000000";
       when "1001"  => ph_ipreg_we <= "1000000000";
       when others  => ph_ipreg_we <= (others => '0');
     end case;
  end process;

--
*************************************************************************
*******
```

```
--* Logic to Calculate the Intermediate product of Addition
*
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     ph_ip_acnt <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if ( bmp_iph_sext = '1' and ph_ip_mcnt = "1001") then
        ph_ip_acnt <= ph_ip_acnt + 1;
      end if;
    end if;
    end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_phext_dn     <= '0';
        bmp_pco2phext_dn <= '0';
    else if (hclk 'event and hclk = '1') then
      if (ph_ip_acnt = "1001" and bmp_iph_ana = '1') then
        bmp_phext_dn <= '1';
            else
        bmp_phext_dn <= '0';
      end if;
      if (ph_ip_acnt = "1001" and bmp_ipc02_ana = '1') then
        bmp_pco2phext_dn <= '1';
            else
        bmp_pco2phext_dn <= '0';
      end if;
    end if;
    end if;
  end process;

--
*************************************************************************
*******
--* Multiplexer for Adder OperandA
*
--
*************************************************************************
*******
  process(ph_const_f   , ph_ip_mreg6  , ph_ip_mreg4 ,
          ph_ip_areg11 , ph_ip_areg12 , ph_ip_acnt)
  begin
     case (ph_ip_acnt) is
       when "000"  => ph_muxc_data <= ph_const_f;
       when "001"  => ph_muxc_data <= ph_ip_mreg6;
       when "010"  => ph_muxc_data <= ph_ip_mreg4;
       when "011"  => ph_muxc_data <= ph_ip_areg11;
       when "100"  => ph_muxc_data <= ph_ip_areg12;
       when others => ph_muxc_data <= ph_const_f;
     end case;
  end process;

--
*************************************************************************
*******
```

```
--* Multiplexer for Adder  OperandB
*
--
*************************************************************************
*******
  process(ph_ip_areg10 , ph_ip_mreg8 , ph_ip_mreg7 , ph_ip_mreg5 ,
          ph_ip_areg9  , ph_ip_acnt)
  begin
     case (ph_ip_acnt) is
       when "000"  => ph_muxd_data <= ph_ip_mreg8;
       when "001"  => ph_muxd_data <= ph_ip_mreg7;
       when "010"  => ph_muxd_data <= ph_ip_mreg5;
       when "011"  => ph_muxd_data <= ph_ip_areg10;
       when "100"  => ph_muxd_data <= ph_ip_areg9;
       when others => ph_muxd_data <= ph_ip_mreg8;
     end case;
  end process;


--
*************************************************************************
*******
--* Logic to generate intermediate(addition) reg write enable signals
*
--
*************************************************************************
*******
  process(ph_ip_acnt)
  begin
     case (ph_ip_acnt) is
       when "000"   => ph_ipareg_we <= "00001";
       when "001"   => ph_ipareg_we <= "00010";
       when "010"   => ph_ipareg_we <= "00100";
       when "011"   => ph_ipareg_we <= "01000";
       when "100"   => ph_ipareg_we <= "10000";
       when others  => ph_ipareg_we <= (others => '0');
     end case;
  end process;

--
*************************************************************************
*******
--* Store intermediate results in reg
*
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     ph_cred_val <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if ( ph_ipreg_we(0) = '1') then
        ph_cred_val <= ph_mresult;
      end if;
    end if;
    end if;
  end process;

  process (hclk , hreset_n)
  begin
```

```
  if (hreset_n = '0') then
   ph_ip_mreg0 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if ( ph_ipreg_we(1) = '1') then
      ph_ip_mreg0 <= ph_mresult;
    end if;
  end if;
  end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   ph_ip_mreg1 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if ( ph_ipreg_we(2) = '1') then
      ph_ip_mreg1 <= ph_mresult;
    end if;
  end if;
  end if;
end process;

 process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   ph_ip_mreg2 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if ( ph_ipreg_we(3) = '1') then
      ph_ip_mreg2 <= ph_mresult;
    end if;
  end if;
  end if;
end process;

 process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   ph_ip_mreg3 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if ( ph_ipreg_we(4) = '1') then
      ph_ip_mreg3 <= ph_mresult;
    end if;
  end if;
  end if;
end process;

 process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   ph_ip_mreg4 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if ( ph_ipreg_we(5) = '1') then
      ph_ip_mreg4 <= ph_mresult;
    end if;
  end if;
  end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
```

```
   ph_ip_mreg5 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if ( ph_ipreg_we(6) = '1') then
      ph_ip_mreg5 <= ph_mresult;
    end if;
  end if;
  end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   ph_ip_mreg6 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if ( ph_ipreg_we(7) = '1') then
      ph_ip_mreg6 <= ph_mresult;
    end if;
  end if;
  end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   ph_ip_mreg7 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if ( ph_ipreg_we(8) = '1') then
      ph_ip_mreg7 <= ph_mresult;
    end if;
  end if;
  end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   ph_ip_mreg8 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if ( ph_ipreg_we(9) = '1') then
      ph_ip_mreg8 <= ph_mresult;
    end if;
  end if;
  end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   ph_ip_areg9 <= (others => '0');
  else if (hclk 'event and hclk = '1') then
    if ( ph_ipareg_we(0) = '1') then
      ph_ip_areg9 <= ph_aresult;
    end if;
  end if;
  end if;
end process;

process (hclk , hreset_n)
begin
  if (hreset_n = '0') then
   ph_ip_areg10 <= (others => '0');
```

```
    else if (hclk 'event and hclk = '1') then
      if ( ph_ipareg_we(1) = '1') then
        ph_ip_areg10 <= ph_aresult;
      end if;
    end if;
    end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     ph_ip_areg11 <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if ( ph_ipareg_we(2) = '1') then
        ph_ip_areg11 <= ph_aresult;
      end if;
    end if;
    end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     ph_ip_areg12 <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if ( ph_ipareg_we(3) = '1') then
        ph_ip_areg12 <= ph_aresult;
      end if;
    end if;
    end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     ph_ip_areg13 <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if ( ph_ipareg_we(4) = '1') then
        ph_ip_areg13 <= ph_aresult;
      end if;
    end if;
    end if;
  end process;
--
*****************************************************************************
*******
--* Pipeline Mux Logic
*
--
*****************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     ph_muxa_data_p <= (others => '0');
     ph_muxb_data_p <= (others => '0');
     ph_muxc_data_p <= (others => '0');
     ph_muxd_data_p <= (others => '0');
    else if (hclk 'event and hclk = '1') then
     ph_muxa_data_p <= ph_muxa_data;
     ph_muxb_data_p <= ph_muxb_data;
```

```
     ph_muxc_data_p <= ph_muxc_data;
     ph_muxd_data_p <= ph_muxd_data;
    end if;
    end if;
  end process;
--
*********************************************************************
*******
--* PH Extraction Block Multiplier Instantiation
*
--
*********************************************************************
*******
  ph_mul:
    cph_analyzer_mul
      port map(
               hclk     => hclk,
               hreset_n => hreset_n,
               op_a     => ph_muxa_data_p,
               op_b     => ph_muxb_data_p,
               result   => ph_mresult
              );

  ph_add:
    cph_analyzer_add
      port map(
               hclk     => hclk,
               hreset_n => hreset_n,
               op_a     => ph_muxc_data_p,
               op_b     => ph_muxd_data_p,
               result   => ph_aresult
              );


end Behavioral;


--
*********************************************************************
*******
--* File Name  : cph_analyzer_reg.vhd
*
--* Designer   : Jaideep Chandran
*
--* Date       : 10/03/2011
*
--* Description: BMP image color Analysis register Interface unit.
*
--
*********************************************************************
*******
--
*********************************************************************
*******
--* Include Libraries
*
--
*********************************************************************
*******
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--
************************************************************************
*******
--* Entity Decleration, I/O ports
*
--
************************************************************************
*******
entity cph_analyzer_reg is
    Port (
           hclk            : in  STD_LOGIC;
           hreset_n        : in  STD_LOGIC;
           haddr_s         : in  STD_LOGIC_VECTOR (31 downto 0);
           hwdata_s        : in  STD_LOGIC_VECTOR (31 downto 0);
           htrans_s        : in  STD_LOGIC_VECTOR (1 downto 0);
           hsel_s          : in  STD_LOGIC;
           hsize_s         : in  STD_LOGIC_VECTOR (2 downto 0);
           hready_in       : in  STD_LOGIC;
           hwrite_s        : in  STD_LOGIC;
           bmp_ca_dn       : in  STD_LOGIC;
           bmp_hid_reg_ph  : in  STD_LOGIC_VECTOR (15 downto 0);
           bmp_fs_reg_ph   : in  STD_LOGIC_VECTOR (31 downto 0);
           bmp_fhi_reg_ph  : in  STD_LOGIC_VECTOR (7 downto 0);
           bmp_ih_reg_ph   : in  STD_LOGIC_VECTOR (31 downto 0);
           bmp_iw_reg_ph   : in  STD_LOGIC_VECTOR (31 downto 0);
           bmp_pw_reg_ph   : in  STD_LOGIC_VECTOR (7 downto 0);
           bmp_hid_reg_co2 : in  STD_LOGIC_VECTOR (15 downto 0);
           bmp_fs_reg_co2  : in  STD_LOGIC_VECTOR (31 downto 0);
           bmp_fhi_reg_co2 : in  STD_LOGIC_VECTOR (7 downto 0);
           bmp_ih_reg_co2  : in  STD_LOGIC_VECTOR (31 downto 0);
           bmp_iw_reg_co2  : in  STD_LOGIC_VECTOR (31 downto 0);
           bmp_pw_reg_co2  : in  STD_LOGIC_VECTOR (7 downto 0);
           cph_ph_val      : in  STD_LOGIC_VECTOR (31 downto 0);
           cph_co2_val     : in  STD_LOGIC_VECTOR (31 downto 0);
           bmp_cstart      : out STD_LOGIC;
           bmp_msaddr_ph   : out STD_LOGIC_VECTOR (31 downto 0);
           bmp_msaddr_co2  : out STD_LOGIC_VECTOR (31 downto 0);
           hready_out      : out STD_LOGIC;
           hresp_s         : out STD_LOGIC_VECTOR (1 downto 0);
           hrdata_s        : out STD_LOGIC_VECTOR (31 downto 0));

end cph_analyzer_reg;

--
************************************************************************
*******
--* Main Block Logic
*
--
************************************************************************
*******
architecture Behavioral of cph_analyzer_reg is

--
************************************************************************
*******
--* Register Decleration
*
```

```
--
*************************************************************************
*******
  signal bmp_ma_reg_ph  : STD_LOGIC_VECTOR (31 downto 0);
  signal bmp_ma_reg_co2 : STD_LOGIC_VECTOR (31 downto 0);
  signal bmp_c_reg      : STD_LOGIC;
  signal bmp_ir_reg     : STD_LOGIC;
  signal hready_p1      : STD_LOGIC;
  signal hready_p2      : STD_LOGIC;
  signal bmp_reg_addr   : STD_LOGIC_VECTOR (4 downto 0);
  signal bmp_rd_data    : STD_LOGIC_VECTOR (31 downto 0);
  signal reg_dcd        : BIT_VECTOR (3 downto 0);

begin
  bmp_reg_addr   <= haddr_s(6 downto 2);
  bmp_msaddr_ph  <= bmp_ma_reg_ph;
  bmp_msaddr_co2 <= bmp_ma_reg_co2;
  hresp_s        <= "00";
  hready_out     <=  hready_p2;
--
*************************************************************************
*******
--* Address Decoder
*
--
*************************************************************************
*******
  process(bmp_reg_addr)
  begin
     case (bmp_reg_addr) is
       when "00110"  => reg_dcd <= "0001";
       when "01101"  => reg_dcd <= "0010";
       when "01110"  => reg_dcd <= "0100";
       when "01111"  => reg_dcd <= "1000";
       when others   => reg_dcd <= (others => '0');
     end case;
  end process;

--
*************************************************************************
*******
--* Generating signal for hready_s signal
*
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     hready_p1 <= '0';
    else if (hclk 'event and hclk = '1') then
      if (hready_in = '1') then
        hready_p1 <= '1';
      end if;
    end if;
   end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
```

```
     hready_p2 <= '0';
    else if (hclk 'event and hclk = '1') then
      if (hready_in = '1' and hready_p1 = '0') then
        hready_p2 <= '1';
      end if;
    end if;
   end if;
  end process;

--
*************************************************************************
*******
--* Generating Write enable signal from controller side
*
--
*************************************************************************
*******
  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_ma_reg_ph <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if (hwrite_s = '0' and hsel_s = '1' and reg_dcd(0) = '1') then
      bmp_ma_reg_ph <= hwdata_s;
     end if;
    end if;
   end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_ma_reg_co2 <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if (hwrite_s = '0' and hsel_s = '1' and reg_dcd(1) = '1') then
      bmp_ma_reg_co2 <= hwdata_s;
     end if;
    end if;
   end if;
  end process;

  bmp_cstart <= bmp_c_reg;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     bmp_c_reg <= '0';
    else if (hclk 'event and hclk = '1') then
      if (hwrite_s = '0' and hsel_s = '1'  and reg_dcd(2) = '1') then
      bmp_c_reg <= hwdata_s(0);
    else if (bmp_ca_dn = '1') then
      bmp_c_reg <= '0';
      end if;
     end if;
    end if;
   end if;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
```

```
     bmp_ir_reg <= '0';
    else if (hclk 'event and hclk = '1') then
      if (hwrite_s = '0' and hsel_s = '1' and reg_dcd(3) = '1' and
hwdata_s(0) = '1') then
      bmp_ir_reg <= '0';
    else if (bmp_ca_dn = '1') then
      bmp_ir_reg <= '1';
      end if;
     end if;
    end if;
   end if;
  end process;

--
*****************************************************************
*******
--* Multiplexer
*
--
*****************************************************************
*******
  process(bmp_reg_addr   , bmp_hid_reg_ph , bmp_fs_reg_ph  ,
bmp_fhi_reg_ph ,
          bmp_ih_reg_ph  , bmp_iw_reg_ph  , bmp_pw_reg_ph  ,
bmp_ma_reg_ph  ,
          bmp_hid_reg_co2, bmp_fs_reg_co2 , bmp_fhi_reg_co2,
bmp_ih_reg_co2 ,
          bmp_iw_reg_co2 , bmp_pw_reg_co2 , bmp_ma_reg_co2 , bmp_c_reg
,
          bmp_ir_reg     , cph_ph_val     , cph_co2_val)
  begin
     case (bmp_reg_addr) is
       when "00000"  => bmp_rd_data <= x"0000" & bmp_hid_reg_ph;
       when "00001"  => bmp_rd_data <= bmp_fs_reg_ph;
       when "00010"  => bmp_rd_data <= x"000000" & bmp_fhi_reg_ph;
       when "00011"  => bmp_rd_data <= bmp_ih_reg_ph;
       when "00100"  => bmp_rd_data <= bmp_iw_reg_ph;
       when "00101"  => bmp_rd_data <= x"000000" & bmp_pw_reg_ph;
       when "00110"  => bmp_rd_data <= bmp_ma_reg_ph;
       when "00111"  => bmp_rd_data <= x"0000" & bmp_hid_reg_co2;
       when "01000"  => bmp_rd_data <= bmp_fs_reg_co2;
       when "01001"  => bmp_rd_data <= x"000000" & bmp_fhi_reg_co2;
       when "01010"  => bmp_rd_data <= bmp_ih_reg_co2;
       when "01011"  => bmp_rd_data <= bmp_iw_reg_co2;
       when "01100"  => bmp_rd_data <= x"000000" & bmp_pw_reg_co2;
       when "01101"  => bmp_rd_data <= bmp_ma_reg_co2;
       when "01110"  => bmp_rd_data <= x"0000000" & "000" & bmp_c_reg;
       when "01111"  => bmp_rd_data <= x"0000000" & "000" &
bmp_ir_reg;
       when "10000"  => bmp_rd_data <= cph_ph_val;
       when "10001"  => bmp_rd_data <= cph_co2_val;
       when others => bmp_rd_data <= (others => '0');
     end case;
  end process;

  process (hclk , hreset_n)
  begin
    if (hreset_n = '0') then
     hrdata_s <= (others => '0');
    else if (hclk 'event and hclk = '1') then
      if (hsel_s = '1' and hwrite_s = '1' and hsize_s = "010") then
```

```
      hrdata_s <= bmp_rd_data;
     end if;
    end if;
   end if;
  end process;

end Behavioral;
```