1-1-2008

Proposed implementation of various imaging modes of OCT on FPGA

Vitaliy Mykhaylychenko
Ryerson University

Follow this and additional works at: <http://digitalcommons.ryerson.ca/dissertations>

Part of the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Mykhaylychenko, Vitaliy, "Proposed implementation of various imaging modes of OCT on FPGA" (2008). *Theses and dissertations*. Paper 353.

PROPOSED IMPLEMENTATION OF VARIOUS IMAGING MODES OF OCT ON FPGA

R
857.06
M95
2008

by
Vitaliy Mykhaylychenko
B. Eng., Ryerson University,
Toronto, 2005

A Project
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of
Master of Engineering
in the Program of
Electrical Engineering

Toronto, Ontario, Canada, 2008
©Vitaliy Mykhaylychenko, 2008

Declaration of Authorship

I hereby declare that I am the sole author of this project.

I authorize Ryerson University to lend this project to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this project by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Borrower's Page

Ryerson University requires all the signatures of all persons using or photocopying this thesis.

Please sign below, and give address and date.

[illegible]

ABSTRACT

Optical Coherence Tomography (OCT) is three-dimensional imaging technique, capable of producing high resolution cross-sectional images through homogeneous samples, such as biological tissue. Real-time acquisition rates are a significant advantage in OCT. However, when images are generated by first acquiring data and then postprocessed on PC to extract information, this advantage is cancelled out.

In this project a system which can provide estimation of pixel values for three imaging modes: Structural, Color-Flow Doppler and Velocity Variance is designed. The proposed system is standalone data processor using FPGA. The system was designed to process recorded data for verification purposes; however, it can be easily modified to process real-time data. Since the system has three modes of visualization it can be used for detecting *in vivo* blood flow. The proposed system was completed at MATLAB, Xilinx System Generator (XSG) and VHDL levels and the results show consistency between theoretical results, MATLAB, XSG and VHDL.

Acknowledgements

I would like to thank my supervisor, Dr. Adnan Kabbani, for providing his guidance, knowledge and support. His words of advice, his trust, and his patience and understanding helped me reach this goal of my life.

Dr. Lev Kirischian, the member of the review committee, needs to be thanked for his participation and invaluable feedback.

I would specially like to thank Dr. Victor Yang for explaining the principles of OCT to me.

I would like to acknowledge the Electrical and Computer Engineering department for excellent program of graduate studies, and for the opportunity of a teaching experience as a laboratory instructor.

I need to thank my parents, whose support and unconditional help were invaluable in completing this work.

Table of Contents

1. INTRODUCTION	1
1.1 Overview.....	1
1.2 Project Objectives	2
1.3 Original Contributions	3
1.4 Organization of the report.....	3
2. BACKGROUND	5
2.1 OCT background.....	5
2.1.1 The Doppler Effect	6
2.1.1.1 A Moving Source.....	7
2.1.1.2 A Moving Observer	8
2.1.1.3 A Moving Source and A Moving Observer.....	8
2.1.1.4 A Moving Medium	9
2.1.2 Doppler Signal	9
2.2 Imaging Modes	11
2.2.1 A-Mode Scan	12
2.2.2 B-Mode Scan	12
2.2.3 M-Mode Scan.....	13
2.2.4 Structural Imaging Mode	14
2.2.5 Color Flow Imaging.....	14
2.2.5.1 Color Doppler Flow Imaging.....	14
2.2.5.2 Velocity Variance Mode	15
2.3 Signal Processing Techniques.....	16
2.3.1 Hilbert Transformation	16
2.3.2 Autocorrelation	17
2.3.3 Cross correlation	18
2.3.4 In-phase and Quadrature modulator (I&Q).....	19
2.4 Technology Comparison.....	20
2.4.1 Digital Signal Processor.....	21
2.4.2 Field Programmable Gate Array with Digital Signal Processor.....	22

2.4.3	Field Programmable Gate Array	22
2.4.4	Application-Specific Integrated Circuit	23
2.4.5	Technology Selection	23
2.5	Field Programmable Gate Array background	24
2.6	M Language, Simulink and RTL	26
2.7	Summary	27
3.	PROPOSED DESIGN	28
3.1	Proposed Design Overview	28
3.1.1	SB Block	30
3.1.2	Structural Mode	33
3.1.2.1	Theoretical Description of Structural Mode	33
3.1.2.2	Proposed Structural Mode Design with XSG	37
3.1.3	Color-Doppler (C.D.) Mode	42
3.1.3.1	Theoretical Description of Color-Doppler Mode	42
3.1.3.2	Proposed C.D. Mode Design with XSG	43
3.1.4	Velocity Variance (V.V.) Mode	46
3.1.4.1	Theoretical Description of Velocity Variance Mode	46
3.1.4.2	Proposed V.V. Mode Design with XSG	46
3.2	Summary	48
4.	RESULTS	49
4.1	General flow	49
4.2	Simulation Results	50
4.2.1	Sampling Block test	50
4.2.2	Biasing block test	53
4.2.3	Structural block test	54
4.2.4	Kasai Block Test	56
4.2.5	VV Block Test	58
4.3	Resources Report	59
4.4	Timing Comparison	60
4.5	Summary	60
5.	CONCLUSIONS AND FUTURE WORK	62

5.1	Major Conclusions	62
5.2	Future Work	63
BIBLIOGRAPHY		64
A1.	Glossary	67
A2.	Selected subsystems used in the design	71
A3.	Selected Xilinx System Generator Blocks used in the design	73
A4.	Selected MCode functions	78
A5.	Sample Simulation Calculations	82

LIST OF FIGURES

Figure 2.1: Basic OCT system, based on a Michelson interferometer.....	6
Figure 2.2: The Doppler Effect: Moving Source [11].....	7
Figure 2.3: Coordinate system for blood flow (red) through the optical beam (blue) [12].....	10
Figure 2.4: A-Mode imaging [11].....	12
Figure 2.5: (a) A-mode display, (b) B-mode display of (a) [16].....	13
Figure 2.6: M-mode imaging [11].....	13
Figure 2.7: Schematic diagram of the quadrature interferometer for optical Hilbert transformation [21].....	1
6	
Figure 2.8: Schematic diagram of the endoscopic color Doppler OCT setup [23].....	18
Figure 2.9: Diagram of the Doppler processing unit using a digital cross-correlator [23].....	18
Figure 2.10: Schematics of the fast ODT system [24].....	19
Figure 3.1: System Overview.	28
Figure 3.2: Simulink with XSG Design.....	30
Figure 3.3: SB block.....	30
Figure 3.4: SB block (XSG level).....	33
Figure 3.5: Spatial averaging masks for improving the SNR in structural and flow images.....	34
Figure 3.6: Structural mode (XSG level).....	38
Figure 3.7: Structural Mode flowchart (part a).....	40
Figure 3.8: Structural Mode flowchart (part b).....	41
Figure 3.9: Color-Doppler mode (XSG level).....	44
Figure 3.10: Velocity Variance mode (XSG level).....	47
Figure 4.1: Sampling test block (XSG).....	51
Figure 4.2: Sampling test block schematic (ISE).....	51
Figure 4.3: First two Sampling block outputs.....	51
Figure 4.4: Last two Sampling block outputs.....	52
Figure 4.5: First three Biasing block outputs.....	53
Figure 4.6: First two Structural block outputs.....	56
Figure 4.7: First two Kasai block outputs.....	58
Figure 4.8: First VV block output.....	58
Figure A1.1: Beats formation: a) individual waves, b) combined wave [5].....	67
Figure A1.2: CLB [33].....	68
Figure A1.3: Slice [33].....	69

Figure A2.1: Sampling block (XSG level).....	71
Figure A2.2: RIRO block (XSG level).....	71
Figure A2.3: Str_Acc1 Accumulator block (XSG level).....	71
Figure A2.4: Str_DR block diagram (XSG level).....	72
Figure A2.5: Kasai_Add block (XSG level).....	72
Figure A3.1: Down Sample: first bit.....	75
Figure A3.2: Down Sample: last bit.....	75
Figure A3.3: Up Sample block (zero-padding enabled).....	77

LIST OF TABLES

Table 3.1: Input data (demo, recorded format).....	32
Table 3.2: Sample I and Q arrays (demo).....	32
Table 3.3: a) Input data (demo, partial), b) Rearranged data.....	35
Table 3.4: Structural mode pixel input elements.....	36
Table 3.5: Column zero-padding.....	36
Table 3.6: Row zero-padding.....	37
Table 3.7: Str_Selector truth table.....	41
Table 3.8: Color-Doppler mode pixel input values.....	43
Table 4.1: Sampling block behavioral simulation results comparison.....	52
Table 4.2: Biasing block behavioral simulation results comparison (I_{bias}).....	53
Table 4.3: Biasing block behavioral simulation results comparison (Q_{bias}).....	54
Table 4.4: $I^2/(M \times N)$ for element (1, 1) of the array.....	55
Table 4.5: Structural block behavioral simulation results comparison (S^2).....	56
Table 4.6: Num/($M \times (N-1)$) for pixel (1, 1).....	57
Table 4.7: Kasai block behavioral simulation results comparison (arctan).....	58
Table 4.8: VV block behavioral simulation results comparison (vv_out).....	59
Table 4.9: Device utilization summary (estimated values).....	59
Table 4.10: Results acquired from algorithms experimentation.....	60

List of Acronyms

A/D converter – Analog-to-Digital converter
ASIC – Application-Specific Integrated Circuit
c.c. – Clock Cycle
CORDIC – COordinate Rotation DIgital Computer
CT – Computer Tomography
DSP – Digital Signal Processor
DOCT – Doppler Optical Coherence Tomography
EOCT – Endoscopic Optical Coherence Tomography
FPGA – Field Programmable Gate Array
fps – frames per second
GSR – Global Set/Reset
GUI – Graphical User Interface
HDL – Hardware Description Language
LPF – Low Pass Filter
LSB – Least Significant Bit
LUT – Look Up Table
MGT – Multi-Gigabit Transceiver
MRI – Magnetic Resonance Imaging
MSB – Most Significant Bit
OCT – Optical Coherence Tomography
RTL – Register Transfer Level
SNR – Signal-to-noise ratio
SOP – Sum of Products
XSG – Xilinx System Generator

Chapter 1

INTRODUCTION

1.1 Overview

Medical Imaging plays an important role in the life of many people. Imaging techniques such as ultrasound, X-Ray, Magnetic Resonance Imaging (MRI) and computer tomography (CT) have been widely used for diagnostic purposes for many years. X-Ray and CT are used for visualizing skeletal structures; however, their ability to visualize soft tissue is poor. For visualizing soft tissues MRI is used. But, it is expensive and complex imaging modality. Another less expensive imaging modality used for imaging soft tissues is ultrasound. It does not achieve high penetration depths, though.

Optical Coherence Tomography (OCT) uses some of the same signal processing techniques as ultrasound and generates images with similar speckle pattern [1]. However, unlike ultrasound, OCT is able to achieve high penetration depths for its resolution characteristics. Advantages of OCT over aforementioned imaging modalities are reported in [2]: near video rate image acquisition, resolution improvement by at least an order of magnitude and the potentially small size and low cost of a simple fibre-optic OCT scanner.

One of OCT modalities – Doppler OCT (DOCT) is capable of monitoring *in vivo* blood flow beneath the skin [3], which is the interest of this project. Real-time acquisition rates are a significant advantage in OCT. Nevertheless, when images are generated by first acquiring data and then postprocessed to extract information, this advantage is cancelled out. Therefore, there is a need to develop a system capable of carrying out OCT calculations in real time.

Digital cameras with speeds up to 25 frames-per second (fps) and a minimal exposure time of 0.04 s are sufficient to image relatively slow-moving individual cells, such as rolling leukocytes (30-70 $\mu\text{m/s}$), in blood flow [4]. Sekizuka et al. [5] demonstrated that maximum cell velocity in rat mesenteric lymphatics was 2-3 mm/s using a high-speed videotape recorder (2000 fps). Dixon et al. [6] in a later experiment determined that maximum cell velocity in some mesenteric lymphatics may reach 7 mm/s and their oscillations correlate with phasic contractions. In [6] a relatively high-speed camera (500 fps) was used. Such variability of cell velocity possibly resulted from different experimental conditions. However, due to motion distortion, these speeds are not quite fast enough to image shape and subcellular structures of individual fast-moving cells [4]. For example, imaging in blood microvessels, with typical flow velocities of 5-10 mm/s, requires imaging speed ranges of 5,000-10,000 fps [7]. Given such number of fps, even for 200x200 pixels image require dedicated hardware to accelerate the computation of pixel values.

1.2 Project Objectives

Objective of this project was to research three most commonly used imaging modes of OCT: Structural, Color-Doppler Flow and Velocity Variance Imaging modes. The mathematical algorithms and suitable hardware for development of a prototype capable of evaluating pixel values in these three imaging modes needs to be chosen in order to accelerate signals processing comparing to PC.

This prototype must enable processing at high fps and give an idea of what kind of hardware will be required to carry out the calculations at higher fps.

1.3 Original Contributions

Main contributions of this work are as follows.

- Comparing and selecting the algorithms and hardware implementation platform for OCT system.
- Designing data inputting stage for the selected OCT system.
- Translating from mathematics to RTL level a design for estimating pixel values in Structural, Color-Doppler and Velocity Variance imaging modes simultaneously.
- Estimating the number of bits required to represent the output of each block to minimize resource consumption and to improve design performance on a Field Programmable Gate Array (FPGA).
- Implementing the design at VHDL level using ISE.
- Verifying the proposed design by running the simulations on MATLAB, Xilinx System Generator (XSG) and VHDL levels.
- Comparing floating point results obtained by MATLAB with fixed point results (VHDL).

1.4 Organization of the report

Chapter 2 provides the necessary background on OCT, Doppler Effect and its significance to OCT. Different scan and imaging modes are then introduced. Please note ultrasound and OCT are both used for explanation of scan and imaging modes since (OCT) uses some of the same signal processing techniques as ultrasound and generates images with similar speckle pattern. Various existing implementations of OCT systems hardware levels are compared and preferred technology is chosen.

In Chapter 3, the design is proposed at XSG level. The design of data sampling and biasing blocks is discussed in section 3.1.1. Data rearranging is presented. Axial and lateral averaging for improving signal-to-noise ratio (SNR) is introduced. Also the cases where the algorithm exceeds the range of the input data are discussed and possible solution is proposed. The theory and XSG top level design for Structural, Color Doppler Flow and Velocity Variance imaging modes is presented.

Chapter 4 shows the XSG blocks and VHDL implementation (fixed point) comparison with their counterparts produced by MATLAB (floating point).

The major contributions and proposed future work are discussed in Chapter 5.

Chapter 2

BACKGROUND

In this Chapter, the background theory pertaining to the design will be discussed. OCT basics are introduced. Various generic OCT imaging modes and imaging modes used in this project are also presented. Also, in this chapter, the hardware is chosen for the implementation.

2.1 OCT background

OCT is three-dimensional imaging technique, capable of producing high resolution cross-sectional images through homogeneous samples, such as biological tissue; first presented by Huang et al. [8]. It is an interferomic technique which relies on interference between split and later recombined broadband optical field. A typical OCT schematic is shown in Figure 2.1.

In OCT, the image is generated by the rotation of a single laser beam through a given angle [1]. A beam of light from a source is split up by a beam splitter into two parts: one part is sent to a reference arm and another to a sample arm (object of observation) [9]. Light reflected back from both reference and sample arms is recombined at a photodetector, creating the interference pattern. When the signal is output from photodetector(s), there are different techniques used to obtain the phase information for measuring the flow velocity (**I & Q creation block**) which are beyond of scope of this project and are discussed in Appendix A2. However, once the phase information has been extracted, it is used to estimate pixel values for different imaging modes.

Figure 2.1: Basic OCT system, based on a Michelson interferometer.

This project will discuss the post-processing of the output from the photodetector(s) (**Processing** block). Pixel value estimation using the phase information is the goal of this project. The phase information will be used to estimate pixel values in three different imaging modes. The output of this project can be used to display in real-time and record the processed data in different imaging modes (**Display & Recording** block).

One of major objectives of this project was measuring blood flow. For this objective, Doppler OCT (one of OCT modalities) was considered. It permits the quantitative imaging of fluid flow in highly scattering media, such as monitoring *in vivo* blood flow beneath the skin [3].

2.1.1 The Doppler Effect

The Doppler Effect is the shift in frequency and wavelength of waves that results from a source moving with respect to the medium, receiver moving with respect to the medium, source and receiver moving with respect to the medium, and a moving medium [10].

2.1.1.1 A Moving Source

If the source is stationary with respect to medium, all waves generated by the sources are symmetric on all sides of the source and the wavelength is $\lambda = v/f_0$, where v is the wave speed in medium and f_0 is the source frequency. If the source moves towards or away from stationary observer, during one period τ_0 it moves a distance $v_s \tau_0 = v_s/f_0$, where v_s is the speed of the source with respect to the medium (positive if moving away from the observer, negative if moving towards the observer). Thus the wavelength λ' is varied as

$$\lambda' = \lambda - \frac{v_s}{f_0} \quad (2.1)$$

An observer stationary relative to the medium detects waves with a frequency f given as

$$f = f_0 \left(\frac{v}{v + v_s} \right) = f_0 \quad (2.2)$$

The example of such system is shown in Figure 2.2.

Figure 2.2: The Doppler Effect: Moving Source [11].

Therefore, as shown in Figure 2.2, when the source is moving away from Listener (observer) A, wavelength measured by observer A is greater than the wavelength of the source and the observed frequency is decreased. When the source is moving towards Listener (observer) B, wavelength measured by observer B is less than the wavelength of the source and the observed frequency is increased.

2.1.1.2 A Moving Observer

When an observer moves with a speed v_{ob} relative to a stationary source, the frequency f heard by the observer is

$$f = f_0 \left(\frac{v + v_{ob}}{v} \right) = f_0 + \frac{v_{ob}}{v} f_0 = f_0 + f_D \quad (2.3)$$

where f_D is the Doppler shift and a positive value is substituted for v_{ob} and f_D when the observer moves toward the source and a negative value is substituted when the observer moves away from the source.

When an observer is moving away from a source the observed frequency is decreased. Contrary, when an observer is moving towards a source the observed frequency is increased. However, the speed of the source remains constant; therefore the wavelength observed does not change.

2.1.1.3 A Moving Source and A Moving Observer

In this case the combination of the previous two cases. Given that all motion is along the same line, the effect of the moving source is to change the wavelength but not the wave speed, and the effect of the moving observer and the effect of the moving observer is to change the wave speed but not the wavelength [10]. This leads to (2.4) and (2.5).

$$\lambda' = \frac{v - v_s}{f_0} \quad (2.4)$$

$$f = f_0 \left(\frac{v - v_r}{v - v_{ob}} \right) \quad (2.5)$$

When a source and an observer are approaching each other the observed frequency is increased. When a source and an observer are moving apart the observed frequency is decreased.

2.1.1.4 A Moving Medium

An example of this is waves generated in a stream. The source is stationary towards the bank the observer is standing on. The medium is moving with the speed v_m (positive if moving towards from the observer, negative if moving away the observer). Therefore the observed wavelength λ' changes according to (2.6) and the frequency remains unchanged.

$$\lambda' = \frac{v + v_m}{f_0} = \lambda \left(1 + \frac{v_m}{v}\right) \quad (2.6)$$

When a medium is moving away from an observer the observed wavelength is decreased and when a medium is moving towards an observer the observed wavelength is increased.

2.1.2 Doppler Signal

Instead of a change in audible frequency as is the case in sound, change of frequency in light produces a shift in color. If the light waves appear at the blue, or high-frequency end of the visible light spectrum, this means that the object is moving toward the observer (blue shift) meaning the distance between the source and observer is decreasing. If, on the other hand, the light waves appear at the red, or low-frequency end of the spectrum, the object is moving away (red shift) meaning the distance between the source and observer is increasing [10]. In this report, OCT and ultrasound are used interchangeably as they share signal processing techniques.

The Doppler shift of waves scattered from a moving target is developed in [12] as

$$f_D = \frac{2V_z}{c - V_z} f_0 \approx \frac{2V_z}{c} f_0 = \frac{2V_z}{\lambda_0} \quad (2.7)$$

where c is the speed of light, V_z is the target velocity in the direction of the wave propagation which travels at c ($c \gg V_z$) with a frequency of f_0 , and the wavelength is λ_0 .

In the case of arbitrary interrogation geometry represented by the Doppler angle θ shown in Figure 2.3, f_D is further derived in [12]

$$f_D = \frac{2n_t V \cos(\theta)}{c} f_0 = \frac{2n_t V \cos(\theta)}{\lambda n} \quad (2.8)$$

where V is the blood flow velocity ($= V_z / \cos\theta$), f_0 is the center frequency of the broadband light, and n_t is the refractive index of tissue.

Figure 2.3: Coordinate system for blood flow (red) through the optical beam (blue) [12].

Doppler-shifted signal is not at a single frequency due to a variety of reasons [13]:

1. Cells move at different velocities producing different shifts of the Doppler frequency.
2. Each cell remains within the beam for a short time; thus the signal from one cell is a pure frequency multiplied by some time-gate function, which yields a band of frequencies [14].
3. Acoustic energy traveling within the main beam, but at angles to the beam axis, plus energy in the side lobes, causes different Doppler frequency shifts due to an effective change in the angle, θ .

4. Also tumbling of cells and local velocities resulting from turbulence yield different Doppler frequency shifts. The resulting received signal is composed of large-amplitude signal at the carrier frequency plus the very low amplitude Doppler-shifted signal. Thus, the resulting spectrum has a band of frequencies from which the flow information must be extracted.

Doppler signals are obtained from all vessels in the path of the ultrasound beam (until the beam becomes attenuated due to depth). According to Deane [15], the size of the Doppler signal is directly dependent on four main factors:

1. *Blood velocity.* As the blood velocity increases, the Doppler frequency increases and the signal is greater.
2. *Ultrasound frequency.* The higher the operating frequency results in a greater Doppler frequency.
3. *Choice of frequency.* Lower frequencies have better penetration, and higher frequencies have better penetration; hence by choosing a compromise in frequency there can be a better sensitivity to flow or better penetration.
4. *Angle of insonation.* An angle of insonation, i.e. the angle at which the transducer is aligned to the vessel, will increase the Doppler signal if it is more aligned with the flow direction (the angle between the beam and the direction of flow becomes smaller).

2.2 Imaging Modes

In the 1950s physicians found an interest in using ultrasound energy as a valid medical tool for diagnostic purposes [16]. Ultrasound imaging is a technique where sound waves are used to see tissues and organs of the body. Several imaging modes were developed and will be discussed next as many processing techniques are similar to OCT.

2.2.1 A-Mode Scan

A-mode (A for Amplitude) imaging is used for studying tissue layer in 1 dimension (depth) in which depth is represented along 1 (horizontal) axis and an echo amplitude is displayed along a perpendicular (vertical) axis as shown in Figure 2.4.

Figure 2.4: A-Mode imaging [11].

As echoes return from tissue boundaries and scatterers, a digital signal proportional to the echo amplitude is produced as a function of time. One 'A-line' of data per pulse repetition period is the result [17]. As the speed of sound determines the depth (round-trip time), the tissue interfaces displayed are restricted to those which are along the path of the ultrasound beam and at a certain distance from the transducer. Amplitude mode is useful for studying tissue layers but cannot be used to determine structure or multiple dimensions.

2.2.2 B-Mode Scan

The electronic conversion of the A-mode information into brightness-modulated dots is termed B-mode image. The brightness of the dot is proportional to the echo signal amplitude and is used for M-mode imaging. It is also used for imaging (potential) cardiovascular diseases. Figure 2.5 shows an example of B-Mode image derived from A-Mode.

Figure 2.5: (a) A-mode display, (b) B-mode display of (a) [16].

Figure 2.5(a) depicts A-mode display of returned echoes as a function of time of flight. Figure 2.5(b) displays B-mode display of (a) in which the echo amplitude in each pixel is represented by gray level, with brighter pixels representing echoes with higher amplitudes. To construct B-mode image similar to A-Mode image shown in Figure 2.4, A-scans (sequentially spaced) need to be assembled in the transverse direction. Therefore, 2D image will be obtained with depth plotted against transverse location [1].

2.2.3 M-Mode Scan

M-mode (M for Motion) technique uses the B-mode information to display the echoes from a moving structure (e.g. any organ). The example of such display is shown in Figure 2.6.

Figure 2.6: M-mode imaging [11].

The echo data from a single ultrasound beam passing through a moving anatomy are acquired and displayed as a function of time, represented by reflector depth on the vertical axis (beam path direction) and time on the horizontal axis [17]. As can be seen from Figure 2.6 depth in tissue is varied across time, with combination of A-scans from the same transverse location over the time. The amplitude of the lines is displayed as a gray value, so that large reflections are white. M-mode can provide excellent temporal resolution of motion patterns, allowing the evaluation of the function of anatomical structures (e.g. valves in the heart) [18].

2.2.4 Structural Imaging Mode

Structural Imaging mode provides an idea about what the physical structure looks like: tissues, vessels, etc. The value of a given pixel in the image is calculated according to [19] as

$$S = I + jQ \quad (2.9)$$

where I is in-phase component and Q is quadrature component, which are the results of coherent demodulation that involves mixing the detected OCT signal with $\sin(2\pi f_c t)$ and $\cos(2\pi f_c t)$. The intensity of the backscattered light, which is displayed, is related to the reflectivity of the sample [1].

2.2.5 Color Flow Imaging

2.2.5.1 Color Doppler Flow Imaging

Color Doppler ultrasound displays Doppler shift information superimposed on a real-time, gray scale anatomic image (B-mode). Flow directions toward and away from the transducer result in Doppler shifts that are presented as different colors on the display. Conventionally, the color red is used to indicate flow toward the transducer (positive Doppler shift), and blue is assigned to indicate flow away from the transducer (negative Doppler shift).

The changes in magnitude of the velocity are visible by using different shades of the color to represent different velocities: typically, the lighter the color, the higher the velocity [18]. The magnitude of the Doppler shift is encoded by the “saturation” of the color. A totally unsaturated color is white since it emits a broad distribution of all frequencies of visible light. Hence, the more of the frequencies in the spectrum are of a single frequency, the higher saturation is. High saturation is interpreted by our retina as a purer color. The saturation encoding in color Doppler is usually arranged such that a greater shift (higher velocity either toward or away from the transducer) is encoded as a less saturated hue. An example of this given in [20] indicates that a small positive Doppler shift would be indicated by a dark red, whereas a larger positive Doppler shift would be indicated by a lighter, less saturated red.

2.2.5.2 Velocity Variance Mode

Velocity variance mode is a useful technique for obtaining Doppler image in vessels where blood flow is high. In such cases, a high axial scan frequency alone is insufficient to measure blood flow since aliasing occurs. The signals received become blurred as the sampling rate is too small in comparison to the high frequency Doppler signals generated by fast moving blood. Since the velocity scale is at a wider range for velocity variance mode, aliasing is reduced. The velocity measurement range is increased but the blood flow orientation is lost [19]. This mode is useful for detecting areas of significant turbulence, such as flow near obstructions and bifurcations in vessels since variance generally increases from the laminar to the turbulent regime. It also helps to distinguish true blood flow from bulk tissue motion (e.g. heart wall motion, which typically moves together and thus exhibits low velocity variance) [12].

2.3 Signal Processing Techniques

2.3.1 Hilbert Transformation

Ding et al. [21] suggested to combine circularly polarized reference and linearly polarized sample signals, which introduces a 90 degree phase shift in the interference signal. Based on polarization discrimination the real and imaginary parts of the interference field are produced in two output channels. This method allows generating analytical continuation (in-phase and quadrature components) optically and does not put any restrictions on phase modulation frequency. The example of such system is shown in Figure 2.7.

Figure 2.7: Schematic diagram of the quadrature interferometer for optical Hilbert transformation [21].

As shown in Figure 2.7, light emitted from a short coherence source is linearly polarized by a polarizer oriented at a 45° with respect to the horizontal axis. A non-polarizing beam splitter (BS) splits the input light into reference and sample arms. In the reference arm, linearly polarized light is changed to circularly polarized light by $1/8$ wave plate. In the sample arm, polarizer oriented similarly to that in the source port is there to ensure that backscattered light to the interferometer is linearly polarized. At BS, the linearly polarized light from the sample and

circularly polarized light from the reference are recombined and then divided into two orthogonal components by the polarizing beam splitter (PBS). Using the outputs of detectors D1 and D2, a phase-resolved analytic signal can be created with less data processing [21].

2.3.2 Autocorrelation

The autocorrelation is based on the fact that the centroid $\bar{\omega}$ of the power spectrum of the signal can be used as the estimator of the localized Doppler shift. According to [8], the interferogram signal can be written as an envelope and carrier

$$\tilde{g}(t) = g(t) \cos(\omega_g t) \quad (2.10)$$

where ω_g is the carrier angular frequency of the interferogram and t is time. The autocorrelation $\tilde{R}(\tau)$ is defined as the expected value of the product of the interferogram and a delayed version of the interferogram (with delay τ):

$$\tilde{R}(\tau) = \left\langle \tilde{g}(t) \tilde{g}(t + \tau) \right\rangle \quad (2.11)$$

It is shown in [22] that when the autocorrelation of the envelope is defined as $R(\tau) = \left\langle g(t) g(t + \tau) \right\rangle$, the autocorrelation of the interferogram can finally be written as

$$\tilde{R}(\tau) = \frac{1}{2} R(\tau) \cos(\omega_g \tau) \quad (2.12)$$

This proves that the Doppler shift information encoded in the carrier of the interferogram is present in the carrier of the autocorrelation of the interferogram, which is a function of delay rather than time.

2.3.3 Cross correlation

In order to measure Doppler shifts in the kHz range using cross correlation technique, it is necessary to stabilize the EOCT interferometer center frequency, eliminate parasitic phase noise, and to construct a digital cross correlation unit able to correlate signals of megahertz bandwidth by a fixed lag of up to a few ms. The advantage of using this technique over autocorrelation is correlating subsequent A-scans rather than a single A-scan with itself, which improves the velocity sensitivity on the order of 1000-fold as reported in [23]. The architecture of such system is similar to those used for autocorrelation techniques and is shown in Figure 2.8.

Figure 2.8: Schematic diagram of the endoscopic color Doppler OCT setup [23].

In Figure 2.8, in order to measure the shift in the centroid frequency, each A-scan is correlated with the previous one, delayed by 90° as shown in Figure 2.9.

Figure 2.9: Diagram of the Doppler processing unit using a digital cross-correlator [23].

The difference between cross correlation and auto correlation is in digital cross correlator, which able to correlate signals of several megahertz bandwidth with a fixed lag of

about a millisecond, which was implemented in [23] in analog hardware. Shown in Figure 2.9, the digitized signal gets split into two paths; one delayed by a whole A-scan, the other by a quarter of an interference fringe. Multiplying and low-pass filtering gives the cross-correlation at a fixed lag, and then the signal is converted back to analog.

2.3.4 In-phase and Quadrature modulator (I&Q)

The system using in-phase and quadrature (I&Q) demodulator with software implementation of the Kasai autocorrelation velocity estimator is shown in Figure 2.10.

Figure 2.10: Schematics of the fast ODT system [24].

Shown in Figure 2.10, LS, light source; PC is polarization controller; OC is optical circulator; PM is phase modulator; RSOD is rapid scanning optical delay; SD-1&2 are scan drivers; PMD is phase modulator driver; BPD is balanced photodetector; I&Q is in-phase & quadrature demodulator and COMP is computer. The complexity of Hilbert transformation is $2n\log(n)+n$, where n is the number of samples per A-line, which is usually large (4096) in order to digitize the carrier waveform. However, when using I&Q demodulator, the velocity can be estimated as

$$\bar{v} = \frac{\lambda_0}{2n \cos(\theta)} f_D = -\frac{\lambda_0}{2\pi n \cos(\theta) T} \times \arctan \left(\frac{\sum_{i=1}^{N-1} [Q(i)I(i-1) - I(i)Q(i-1)]}{\sum_{i=1}^{N-1} [I(i)I(i-1) + Q(i)Q(i-1)]} \right) \quad (2.13)$$

Using the above Equation instead of Hilbert transform reduces the computation time for 200x200 image by 87.5 times [24][24] .

2.4 Technology Comparison

In OCT, the scan rate is limited by the rate at which data can be transferred from the detector to a computer for processing. It is preferable to use 12- or even 14-bit detectors to achieve the maximum optical dynamic range of detection, essential for imaging through scattering tissue. Given an arbitrary bit depth b_d , each depth scan requires $b_d N$ bits of memory, where N is the number of pixels in the linear array. Using an assumption that a single lateral (horizontal) scan is generated from L_x axial (vertical) scans, the memory requirement is $b_d N L_x$ bits. If true video rate imaging requires 24 images per second; therefore, the required data transfer rate is $R = 24 b_d N L_x$ bits per second. To obtain video rate three-dimensional images, this number must be multiplied by the number of lateral scans orthogonal to the cross-section L_y . For example, using standard numbers of $b_d=12$, $N=512$, $L_x=512$, $L_y=512$, the data transfer rate of 4.8 GBytes per second is required. Transferring of such amount of data to personal computer is currently not feasible using standard computer hardware and data transfer methods [2].

Real-time acquisition rates are a significant advantage in medical imaging. However, when images are generated by first acquiring data and then postprocessed to extract information, this advantage is cancelled out. We will present different approaches taken by various groups to implement real-time OCT system.

To address these limitations and to complement existing OCT and Doppler OCT techniques, various solutions have been designed and implemented to enable a real-time OCT and Doppler OCT acquisition systems. The most common solutions to these limitations are:

- 1) Signal processing on combinations of Digital Signal Processor (DSP) with FPGA (FPGA only [26], DSP with FPGA [27], DSP only [29]).
- 2) Custom designed hardware [22], [28] and [21], where various equations are used to reduce amount of processing [22] or special interferometer is used so there is no need for stable HF modulator [21].

Imaging speed is fundamentally significant because of its relationship to detection sensitivity (minimum detectable reflectivity). As the A-line rate increases, the detection bandwidth should be increased proportionally, and therefore the sensitivity drops [9].

One straightforward solution is to increase the optical power, which would, in principle, improve the sensitivity; however, available sources and maximum permissible exposure levels of tissue represent significant practical limitations.

2.4.1 Digital Signal Processor

One of the implementations of OCT processing in real time can be done on DSP. The DSP is configured with a signal-processing evaluation module and an analog-to-digital (A/D) daughterboard, and it performs data acquisition, structural imaging, and flow processing sequentially [29]. Two velocity algorithms are used in this DSP: weighted-centroid and sliding-window filtering [30].

For structural image formation and flow calculation using the weighted-centroid algorithm described in [30], this DSP module processes each section in the circular buffer continuously.

2.4.2 Field Programmable Gate Array with Digital Signal Processor

It is also possible to implement signal processing stage for OCT and Doppler OCT techniques using a preprocessing FPGA and a programmable DSP to manage computationally intensive algorithms in real time. Such system was introduced in [27] and it supports both structural and velocity-based images.

The most computationally intense operation performed was the approximation of $\arctan(Im=Re)$ for each Hilbert filter output, and it was done through the look up table in the data cache which stored values of $\arctan(Im=Re)$ over the range of 0° to 45° and symmetry extended this range to 360° . The downside of this approach is that a table size that grows exponentially in size with increasing precision, because each additional bit doubles the storage requirement.

2.4.3 Field Programmable Gate Array

An FPGA implementation has certain advantages over a DSP due to the fact that the processing elements can operate in parallel as opposed to the DSP, which is primarily a sequential processor [26]. In [26] it is demonstrated that the use of an FPGA enables processing rates exceeding DSP-based solutions. In addition, this implementation has the important feature that calculation of the phase in addition to the amplitude of the interference fringe pattern only requires few additional resources. Such implementation ensures high sampling rates increasing the signal-to-noise ratio.

Since the analytic representation of the interference fringe signal is $\Gamma = I + jQ$, the envelope is obtained by taking the magnitude $\sqrt{I^2 + Q^2}$ and the phase by $\arctan(Q/I)$. This could be achieved using look-up tables [27]. However, this approach has the drawback of

requiring a table size that grows exponentially in size with increasing precision, as aforementioned. Larsen et al. [26] decided to use CORDIC algorithm summarized in [31]. According to [26], this algorithm does not require large storage capacity and maps both the Doppler and the envelope algorithm efficiently to the FPGA.

2.4.4 Application-Specific Integrated Circuit

Although signal processing on combinations of hardware DSP and FPGA may accelerate the processing, Rollins et al. [22] proves the short-time Fourier transform (STFT) method is not currently suitable for real-time blood flow imaging, and they introduced the system, where most of signal processing is performed on analog hardware.

Signal processing to extract the Doppler shift information is performed on line in analog hardware in real time. The signal is digitized by a variable scan frame grabber. Then it is color coded and displayed by computer and recorded to high quality video tape in real time. In [22] it is shown that the central velocity estimate can be obtained from the zero-lag element of the autocorrelation of the interferogram and its first derivative. It is significant because it indicates that in principle, the centroid may be measured by operating directly on the time domain signal, and thus may be implemented using on line signal processing.

2.4.5 Technology Selection

Based on the information discussed in this section, FPGA implementation has certain advantages over a DSP due to the fact that the processing elements operate in parallel as opposed to the DSP, which is primarily a sequential processor [26]. In [26] it is demonstrated that the use of an FPGA supports sampling rates exceeding DSP-based solutions. In [26] FPGA is reported to process data sampled at 120 MHz, while in [29] DSP is reported to process data at 8 MHz, both

in real time. In addition, implementation on FPGA has the important feature that calculation of the phase in addition to the amplitude of the interference fringe pattern only requires few additional resources. Such implementation ensures high sampling rates improving the signal-to-noise ratio [26] which is important for visibility of real signals, not noise speckles

All three proposed system implementations (DSP, DSP with FPGA, FPGA) used Hilbert transform for velocity estimation which requires $2n \log(n) + n$ complexity operations, where n is the number of samples per A-line, which is usually large (~ 4096) in order to digitize the carrier waveform. The improved system of the original Kasai autocorrelation velocity estimator (originally developed in [25]) was presented by Yang et al. [24] reduces the computation time for 200x200 image by 87.5 times [24]. Analog hardware was not available for this project and the comparison Rollins et al. showed in [22] did not include the scheme introduced in [24], as the research described in [24] was done later than [22].

Therefore FPGA hardware is selected to implement the proposed OCT algorithm which not only improves the processing speed required for high fps, but also it works as a rapid prototype platform to validate the implementation of design in ASIC. Virtex II Pro VP100 in the FF1704 package was used for this project.

2.5 Field Programmable Gate Array background

FPGA is a digital semiconductor device that contains programmable logic components called "logic blocks", and programmable interconnects, with logic gate density varying from tens of thousands to several millions. FPGAs use the Complementary Metal Oxide Semiconductor SRAM technology and are thus reset at power off [32].

The programmable cell is called Logic Element (LE) in Altera devices and Configurable Logic Block (CLB) in Xilinx devices. Xilinx XC2VP100 in the FF1704 package FPGA was available for the project; hence the rest of the discussion will pertain to Xilinx FPGAs.

The architecture incorporates both embedded RocketIO Multi-Gigabit Transceivers (MGTs) (20) that create high-speed serial links between devices and a hard processor core (IBM PowerPC 405 RISC – 2 blocks) within the FPGA fabric. RocketIO MGTs are parallel-to-serial and serial-to-parallel embedded transceiver cores that are also used for high-bandwidth interconnection between buses, backplanes, or other subsystems. Virtex-II Pro VP100 in the FF1704 package implements the following components:

- 11,024 Configurable Logic Blocks (CLBs) that provide functional elements for combinatorial and synchronous logic, including basic storage elements;
- 22,048 BUFTs (3-state buffers) associated with each CLB element drive dedicated segmentable horizontal routing resources;
- 308 Block SelectRAM+ memory modules provide large 18 Kb storage elements of True Dual-Port RAM;
- 308 Embedded multiplier blocks are 18-bit x 18-bit dedicated multipliers;
- 8 Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for clock distribution delay compensation, clock multiplication and division, and coarse- and fine-grained clock phase shifting.

The aforementioned components are all connected to each other through a fully buffered (SRAM controlled pass transistor) programmable Switching Matrix [33]. It is programmed and controlled via the configuration data (bit file); loaded into the configuration SRAM at power up.

This programmable logic can be used to implement any logic based on logic gates such as ANDs, ORs, and NOTs.

2.6 M Language, Simulink and RTL

M (MATLAB) language was used to create demo as a test comparison model for the Simulink design. The downside of M language is that necessary parameters are left out of the description and can't be unambiguously determined from the model. Most notable is the lack of fixed-point typing, sample rate timing, and parallelism [34].

Simulink is a supplement to MATLAB and is a platform for multi-domain modeling, simulating, and analyzing for dynamic systems using graphical user interface (GUI). Simulink utilizes comprehensive block libraries customizable for specialized applications unlike the script-based design using M Language. For this design we are interested in Xilinx Blockset and Xilinx Reference Blockset libraries which include pre-built Xilinx DSP logic cores. The XSG toolboxes extend the original capabilities of Simulink by integrating the cores from the Core Generator. Core Generator is another Xilinx software tool, which is a GUI-based tool that offers parameterized logic cores that have been optimized (for area & speed) to be implemented on Xilinx FPGAs. These cores range from simple counters, comparators, adders, and multipliers, to full system-level building blocks, such as BRAMs, arctangent and square root functions, etc. XSG therefore allows bit and cycle accurate system level simulation, and automatic code generation for XILINX FPGAs.

It is necessary to note that MATLAB and Simulink operate in either a 32 or 64 bit double precision format, while operations in actual hardware (i.e. an FPGA) must be executed in an integer representation of fixed-point, which needs to be specified unless full precision is used.

The Simulink system period represents the smallest sample period of the system and all other sample rates must be integer multiples of the system period. The FPGA clock period is also defined using XSG block. XSG automatically generates the HDL files, timing constraint files and test-benches for the target device, and project files for the target synthesis tools, which can be used to create the bitstreams and download into the FPGAs.

Once the design is complete and simulated from within MATLAB, the Xilinx specific blocks within the Simulink design can then be synthesized and placed and routed using Xilinx ISE. In this design XSG v9.2 was used and all blocks pertaining to the design are discussed in Appendix A3.

Register Transfer Level (RTL) is a way of describing of operation of synchronous digital circuit. The RTL abstraction is used in Hardware Description Languages (HDLs) such as VHDL and Verilog.

2.7 Summary

The estimation of the pixel values from the phase information of OCT signal is chosen for this project. Various imaging modes were introduced and will be discussed in more details in further chapters. Structural images, color Doppler flow and velocity variance images of B-Mode and M-Mode scans [1] will be developed. FPGA was chosen as preferred technology and improved Kasai autocorrelation velocity estimator algorithm from [24] was chosen for real-time on-line signal processing. Xilinx XC2VP100 FPGA was chosen for the design as it was readily available and provided sufficient resource to demonstrate the idea behind the design. Next Chapter is dedicated to each block of the proposed design.

Chapter 3

PROPOSED DESIGN

The proposed design overview is provided. The functionality of each top level block is discussed.

3.1 Proposed Design Overview

Since FPGA is chosen, XSG will be used to develop a system capable of evaluating pixel values in three different imaging modes: Structural, Color Doppler Flow and Velocity Variance as shown in Figure 3.1.

Figure 3.1: System Overview.

In Figure 3.1 the output signals (used in XSG design) of the suggested blocks shown in bold. Data sampling and biasing and their need are discussed in Section 3.1.1. Estimation of the pixels values are shown in Section 3.1.2 for Structural imaging mode, in Section 3.1.3 for Color-Doppler mode and in Section 3.1.4 for Velocity Variance mode. Appendix A3 provides the description and configuration of Xilinx blocks used in the design, Appendix A1 provides glossary of the terms used in this chapter and Appendix A4 provides some of MCode functions written for this project.

In this project, the M language model of each block shown in Figure 3.1 was created first in order to create very high level abstraction model and to use the results for verification of XSG result. High level abstraction system will be created using XSG blocks in order to create the FPGA solution of the design discussed in MATLAB section. Virtex2P xc2vp100-6ff1704 will be used as the target FPGA board, specified in XSG block. The Simulink clock cycle (c.c.) is set to '1' for demonstration purposes.

This system consists of the following subsystems: *Sampling and Biasing (SB)*, *Structural*, *Kasai*, and *VV* as shown in Figure 3.2. *SB Block* splits up the incoming signal into Channels *I* and *Q*. It also eliminates the bias which was introduced during the time the data was taken. *We_bias* signal is used as enable signal in *Structural* and *Kasai* subsystem. *Structural*, *Kasai*, and *VV* subsystems carry out the calculations for Structural, Color-Doppler and Velocity Variance modes, respectively.

The input(s) and output(s) of each block are displayed on the block. For example, the input of *SB* block is labeled *In1*, and the outputs are labeled *Ch I*, *Ch Q* and *we_bias*. *Y_Out*, *Y_Out*, *atan_Out*, *S2_Out* and *VV_Out* are top level output ports once the design is translated into hardware.

In this report, the block names that are italic and bold represent subsystems, and the block names that are just italic represent single block.

Each block of the design was translated to VHDL for verification of the results with MATLAB. The completed design was translated to VHDL too.

Figure 3.2: Simulink with XSG Design.

3.1.1 SB Block

The incoming data are sampled into Channels I and Q . After that it is biased as shown in Figure 3.3

Figure 3.3: SB block.

Initially the data are read in from A/D converter (**Real Time Data** block) or in case of MATLAB or XSG simulations from workspace (**Recorded Data** block). In demo case, it is assumed the output of A/D converter has 12 bit (signed) range and no decimal places. This number can be changed depending on the output of A/D converter used. A/D converter contains I & Q from demodulation block, which in turn has its input from photodetector(s).

In this design (for MATLAB and XSG simulations) the assumption is that the data are stored consecutively first and then read out. This means that 1st element of Channel Q is stored after 1st element of Channel I , 2nd element of Channel Q is stored after 2nd element of Channel I , etc. Channels I and Q are in-phase and quadrature components of OCT signal, respectively. They are obtained by demodulation of OCT signal. When doing real-time processing, these Channels will be already separated; therefore sampling will not be required.

The data used for demo simulations is shown in Table 3.1. Data are sampled as 1-D Channel (hence rows 1, 2, 3 and 4 are shown for illustration purposes only) with Channels I and Q encoded consecutively. The demo data has only 7 bits ($64=2^6+0$); however, during the design the number is assumed to be 12 bit two's complement signed integer, and all 12 bits are accounted for.

Therefore, we need to extract Channels I and Q from the input data (*Sampling* Block). Variable I_{raw} is Channel I extracted from raw data; variable Q_{raw} is Channel Q extracted from raw data. In our sample arrays (I and Q) we assume there are four rows and sixteen columns, as shown in Table 3.2. Please note that array index (in 1D) coincides with its value. This is done for demonstration purposes.

Table 3.1: Input data (demo, recorded format).

Row 1	Row 2	Row 3	Row 4
1	17	33	49
1	17	33	49
2	18	34	50
2	18	34	50
3	19	35	51
3	19	35	51
4	20	36	52
4	20	36	52
5	21	37	53
5	21	37	53
6	22	38	54
6	22	38	54
7	23	39	55
7	23	39	55
8	24	40	56
8	24	40	56
9	25	41	57
9	25	41	57
10	26	42	58
10	26	42	58
11	27	43	59
11	28	43	59
12	28	44	60
12	29	44	60
13	29	45	61
13	29	45	61
14	30	46	62
14	30	46	62
15	31	47	63
15	31	47	63
16	32	48	64
16	32	48	64

Table 3.2: Sample I and Q arrays (demo).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

The dynamic range of the input range is reduced by '2' (arbitrary value, can be modified depending on the input range – 1 to 64 in the demo case).. This number will also affect the values

of the pixels produced in resultant images, so special care needs to be taken in order for the data to be meaningful corresponding to color maps used image creation. There will be more discussion in the imaging modes sections. If the data are taken in Real Time, this dynamic range reduction needs to be done in the first input block for Real Time data processing – **Biasing** block.

I and Q Channels are biased (**Biasing** block), where biasing values represent the background noise level and gain differences between I and Q Channels. Therefore, these values will vary in different experiments; we are using arbitrary values of $I_bias=-1.5$, $Q_bias=-7.5$ and $IQFudge=0.875$ for demo purposes, where $IQFudge$ is the gain difference between Channels I and Q . In the later experiments with the proper balancing this step may not be needed.

On XSG level, SB block is realized as shown in Figure 3.4.

Figure 3.4: SB block (XSG level).

3.1.2 Structural Mode

In this section the theory used for Structural mode calculations is shown. Also the cases when the algorithm exceeds the range of the input data are discussed and possible solution is proposed. Axial and lateral averaging is introduced. After theoretical analysis of Structural Imaging mode, the proposed implementation at XSG levels is introduced.

3.1.2.1 Theoretical Description of Structural Mode

In order to reduce the number of computations required, axial and lateral averaging are introduced. N is the number of axial (depth) scans used in autocorrelation and M is the depth

window length (often called the gate length in ultrasound literature). Structural Mode, Color-Doppler Mode and Velocity Variance Mode calculations are performed within an $M \times N$ window.

Next, we need to define *Overlap*. *Overlap* allows for additional computation efficiency if the $M \times N$ windows of adjacent pixels overlap when the flow image is processed, because the calculations within the overlapping portions (αM and βN) can be shared between pixels, as shown in Figure 3.5.

Figure 3.5: Spatial averaging masks for improving the SNR in structural and flow images.

In Figure 3.5 shaded areas indicate regions with shared computation between pixels. A higher frame rate can be obtained by increasing β , at the expense of increased blurring in the lateral direction due to overlapping of sequential axial scans [19].

In the sample array we use $M=4$, $N=8$, $\beta=0.75$ as the values provided in [19], where M was chosen to be '4' as the A/D conversion rate is more than four times higher than the Low-Pass Filter (LPF) bandwidth (i.e., over-sampled). N and β values were set such that the resultant horizontal resolution is 500 pixels for all frame rates except 32 fps. At high frame rates (16 and 32 fps), high β and low N values were chosen to maintain the horizontal resolution. But at lower frame rates (2 – 8 fps), low β and high N values are used to reduce lateral blurring and improve velocity resolution [19]. As it can be seen Axial averaging was not performed on the sample array,

but Lateral averaging was. With overlap set at '0.75', the resulting step size according to (3.1) is '2'

$$stepsize = N - floor(Overlap * N) \quad (3.1)$$

where *floor* rounds the input elements to the nearest integers towards minus infinity. This means that for all imaging modes in every resultant row there will be 8 elements.

Data are scanned column-by-column (see *Imaging Modes*). Therefore, the data needs to be rearranged with the first four elements forming column 1, the next four elements forming column 2, etc. as shown in Table 3.3. The will need to be done in both *Structural* and *Kasai* blocks

Table 3.3: a) Input data (demo, partial), b) Rearranged data.

1	2	3	4	5	6	7	...
---	---	---	---	---	---	---	-----

a)

1	5
2	6
3	7
4	...

b)

First imaging mode that we will consider is Structural mode in which the value of a given pixel (S^2) is given mathematically in [19] as

$$\langle S^2 \rangle = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N [I_{m,n}^2 + Q_{m,n}^2] \quad (3.2)$$

Spatial averaging in the depth and lateral directions are performed to improve the SNR [19]. OCT technique has very high SNR ($\sim 10^4 - 10^5$) for specular reflection from transparent media. But, in scattering media the coherent nature of radiation and interferometric technique used in OCT leads to formation of speckles according to [35] & [37]. However, since the speckle

noise is random, it can be suppressed by spatial averaging [37]. While larger numbers of points provide a smoother image and higher effective SNR, the resultant image is blurred. The degree of blurring increases with the number of points. Hence, the spatial averaging represents a tradeoff between spatial resolution and SNR. We will rely on the results provided in [19] for the choice of parameters used for spatial averaging. Therefore, in Structural mode the value of the resultant pixel will be the mean value of $M \times N$ elements, as shown in Table 3.4 where m is the row index and n is the column index.

Table 3.4: Structural mode pixel input elements.

m, n	$m, n+1$	$m, n+2$	$m, n+3$	$m, n+4$	$m, n+5$	$m, n+6$	$m, n+7$
$m+1, n$	$m+1, n+1$	$m+1, n+2$	$m+1, n+3$	$m+1, n+4$	$m+1, n+5$	$m+1, n+6$	$m+1, n+7$
$m+2, n$	$m+2, n+1$	$m+2, n+2$	$m+2, n+3$	$m+2, n+4$	$m+2, n+5$	$m+2, n+6$	$m+2, n+7$
$m+3, n$	$m+3, n+2$	$m+3, n+3$	$m+3, n+3$	$m+3, n+4$	$m+3, n+5$	$m+3, n+6$	$m+3, n+7$

When the end of the row is reached or there are no more rows in the input array, zero-padding is used (in MATLAB no value is added, in XSG '0' is added). The example of such case is the calculation of element $(1, 15)$ – resultant pixel $(1, 8)$ as shown in Table 3.5.

Table 3.5: Column zero-padding.

1,15	1,16	-	-	-	-	-	-
2,15	2,16	-	-	-	-	-	-
3,15	3,16	-	-	-	-	-	-
4,15	4,16	-	-	-	-	-	-

In this case in Structural mode we can only use numbers from columns 15 and 16, as column 17 does not exist. Another example is the calculation for $m, n=2, 1$ as shown in Table 3.6. In this example, we can only use numbers from rows 2, 3 and 4, as row 5 does not exist.

Table 3.6: Row zero-padding.

2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7	2, 8
3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7	3, 8
4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7	4, 8
0	0	0	0	0	0	0	0

After the summation the value of each resultant pixel is divided by $M \times N$ to find the mean of $M \times N$ window. The natural logarithm of the result is then taken to further reduce the dynamic range [19]. Dynamic range is reduced even more by setting the values greater/lesser the thresholds to maximum and minimum, respectively. We used the maximum of 3.6 and the minimum of 0.3. These values are estimated by trial-and-error method.

3.1.2.2 Proposed Structural Mode Design with XSG

The subsystem used calculations of the values of pixels in the structural image is shown in Figure 3.6. This subsystem consists the following blocks: *Str_WC*, *Str_RC_EN*, *Str_C*, *Str_Acc1*, *Str_Acc2*, *Str_32* and *Str_DR*. *Str_WC* provides the counter values used as indexes to write in the data in *Str_I_RAM* and *Str_Q_RAM* blocks for the Structural mode calculations. *Str_RC_EN* enables the counter (*Str_C*) to read out the data for the Structural mode calculations.

a) Part a.

b) Part b.

Figure 3.6: Structural mode (XSG level).

Channels I (input Str_I , output I_{bias_out} of **Biasing** block) and Q (input Str_Q , output Q_{bias_out} of **Biasing** block) are squared prior to being written into RAM blocks via Str_{Ip2} and Str_{Qp2} multiplication blocks, respectively. It is done in order to minimize the number of multiplications after the data are read out. For example, $I_{4,1}$ is squared once prior to being written

into *Str_I_RAM* block vs. being squared four times if squared when it is read out from *Str_I_RAM* block, as it is used for calculating the output values of $I_{1,1}$, $I_{2,1}$, $I_{3,1}$ and $I_{4,1}$.

When the data are read out from *Str_I_RAM* and *Str_Q_RAM* blocks it is multiplied by 0.03125 ($1/(M \times N) = 1/(4 \times 8)$) – fractional part of the number in binary is ‘00001’. The reason this multiplication can be done when the data are read out is that the multiplication will be performed only when it is required. The reason it (using multipliers only when required) is not done in the current design is because the physical implementation needs to be done to assess possible delays introduced in the other parts of the circuit where these multipliers can be reused.

The number of bits is then reduced to 32 with binary point at 13 bits since maximum integer number we will have with the given input is maximum $2^{23} \times (0.03125) < 2^{18}$, 1 bit is reserved as sign bit and the binary point is set at 13 bits because with the result of squaring I and Q channels can have no more than 8 fractional bits and 0.03125 has 5 fractional bits; therefore the result of multiplication can have no more than 13 fractional bits.

Str_Acc1 and *Str_Acc2* are used to accumulate the values of 32 element addition ($M \times N = 4 \times 8 = 32$). *Str_32* extracts every 32nd element of the accumulation. The outputs of *Str_32* are then added and (3.2) is solved, hence the output of *Str_Add* is $\langle S^2 \rangle$.

The logarithm of the result is then taken to further reduce the dynamic range (in *Str_DR* block). Dynamic range can be reduced even more by setting the values greater/lesser than thresholds to maximum and minimum, respectively. In demo case, the maximum of 3.6 and minimum of 0.3 were used. These values are estimated by trial-and-error method and are shown here for demonstration purposes only.

A more detailed description of Figure 3.6 is shown in Figures 3.7 and 3.8.

Figure 3.7: Structural Mode flowchart (part a).

Figure 3.7 shows the flowchart of the design presented in Figure 3.6a). Once the first non-zero input has been detected in *Biasing* block, *Str_we_In* becomes '1'. This enables read in counter contained in *Str_WC* block. This read in counter took into consideration the need to rearrange the data according to Table 3.3. While the read in counter is enabled, its output *Str_WC_Out* is used for addressing the squared data (via *Str_Ip2* and *Str_Qp2* multiplier blocks) that is read in Single Port RAM blocks (*Str_I_RAM* and *Str_Q_RAM*). *Str_WC_Mux_Sel_w* is '1' when the data are read in. It (*Str_WC_Mux_Sel_w*) is input *a* of *Str_selector* block and when input *a* is '1', the output is '0' – hence, input *d0* of *Str_Mux* block is selected as its (*Str_Mux*) output. When all elements have been read in, *Str_WC_Mux_Sel_w* becomes '0', hence making input *a* of *Str_selector* block '0'. In addition, when all elements have been read in the read out counter *Str_C* is enabled by setting output *Str_RC_EN_C_en* of *Str_RC_EN* block to '1'. Output *Str_RC_EN_C_en* of *Str_RC_EN* block is used as input *b* of *Str_selector* block, which when

equal to '1' selects $d1$ of Str_Mux block. While the data are being read out, it is multiplied by '1/32' – to find the mean according to (3.2). When all elements are read out, Str_C_rst of Str_C becomes '1' which resets read in counter via Str_RC_EN block. The truth table of $Str_Selector$ is shown in Table 3.7. Please note that with current design it is impossible to have case $x=1, y=1$.

Table 3.7: $Str_Selector$ truth table.

x	y	z
0	0	2
0	1	1
1	0	0
1	1	2

The flowchart of the design presented in Figure 3.6b) is shown in Figure 3.8.

Figure 3.8: Structural Mode flowchart (part b).

When the data are being read out from Str_I_RAM and Str_Q_RAM blocks and multiplied by '1/32', it is being accumulated using Str_Acc1 and Str_Acc2 blocks. 32 consecutive elements are added according to (3.2). Once all data are read out, every 32nd element (the sum of 32 elements) is extracted from RAM blocks used to store data in Str_32 block. I^2 and Q^2 are then added and dynamic range is reduced using Str_DR block. Taking logarithm of Structural mode signal intensity is suggested by [19]. Dynamic range can be reduced even more

(suggested by [19]) by setting the values greater/lesser than thresholds to maximum (3.6) and minimum (0.3), respectively. These are demo values and one has to make sure that $\langle S^2 \rangle$ fits the color map used for displaying Structural Mode correctly.

The output S^2 can be used to display Structural Mode image and is also used for estimation of the pixel values in Velocity Variance Mode.

3.1.3 Color-Doppler (C.D.) Mode

The theory used for Color-Doppler Imaging mode calculations is discussed in this section. Reading out the data from two different (consecutive) addresses of RAM memory using two indexes (counters) simultaneously is introduced. The proposed implementation at XSG component level is presented.

3.1.3.1 Theoretical Description of Color-Doppler Mode

In Color-Doppler mode the Doppler frequency shift f_D in a given pixel is calculated as

$$f_D = \frac{f_a}{2\pi} \arctan \left\{ \frac{\frac{1}{M(N-1)} \sum_{m=1}^M \sum_{n=1}^{N-1} (I_{m,n+1} Q_{m,n} - Q_{m,n+1} I_{m,n})}{\frac{1}{M(N-1)} \sum_{m=1}^M \sum_{n=1}^{N-1} (Q_{m,n+1} Q_{m,n} + I_{m,n+1} I_{m,n})} \right\} \quad (3.3)$$

$$= \frac{f_a}{2\pi} \arctan \left\{ \frac{\langle Y \rangle}{\langle X \rangle} \right\}$$

where f_a is the axial scan frequency. In [24] the mean velocity $\langle v \rangle$ is further derived as

$$\langle v \rangle = \frac{\lambda_0 f_D}{2n_t \cos(\theta)} \quad (3.4)$$

where λ_0 is the source center wavelength, n_t is the tissue index of refraction, and θ is the angle between the direction of movement and the optical axis, i.e., the Doppler angle. Therefore, in Colour-Doppler mode the value of the resultant pixel will be the mean value of $M \times (N-1)$ elements as shown in Table 3.8.

Table 3.8: Color-Doppler mode pixel input values.

$(m,n+1)^*$ (m,n)	$(m,n+2)^*$ $(m,n+1)$	$(m,n+3)^*$ $(m,n+2)$	$(m,n+4)^*$ $(m,n+3)$	$(m,n+5)^*$ $(m,n+4)$	$(m,n+6)^*$ $(m,n+5)$	$(m,n+7)^*$ $(m,n+6)$
$(m+1,n+1)^*$ $(m+1,n)$	$(m+1,n+2)^*$ $(m+1,n+1)$	$(m+1,n+3)^*$ $(m+1,n+2)$	$(m+1,n+4)^*$ $(m+1,n+3)$	$(m+1,n+5)^*$ $(m+1,n+4)$	$(m+1,n+6)^*$ $(m+1,n+5)$	$(m+1,n+7)^*$ $(m+1,n+6)$
$(m+2,n+1)^*$ $(m+2,n)$	$(m+2,n+2)^*$ $(m+2,n+1)$	$(m+2,n+3)^*$ $(m+2,n+2)$	$(m+2,n+4)^*$ $(m+2,n+3)$	$(m+2,n+5)^*$ $(m+2,n+4)$	$(m+2,n+6)^*$ $(m+2,n+5)$	$(m+2,n+7)^*$ $(m+2,n+6)$
$(m+3,n+1)^*$ $(m+3,n)$	$(m+3,n+2)^*$ $(m+3,n+1)$	$(m+3,n+3)^*$ $(m+3,n+2)$	$(m+3,n+4)^*$ $(m+3,n+3)$	$(m+3,n+5)^*$ $(m+3,n+4)$	$(m+3,n+6)^*$ $(m+3,n+5)$	$(m+3,n+7)^*$ $(m+3,n+6)$

M is the row index and n is the column index. Zero-padding is used in the same cases as in Structural mode. After the summation arctangent of each pixel is taken according to (3.3).

3.1.3.2 Proposed C.D. Mode Design with XSG

The subsystem used calculations of the values of pixels in the Color-Doppler mode image is shown in Figure 3.9. This subsystem consists the following blocks: **Kasai_WC**, **Kasai_RC_EN**, **Kasai_C**, **Kasai_Add**, **Kasai_Acc1**, **Kasai_Acc2**, **Kasai_28** and **Kasai_atan**. **Kasai_WC** provides the counter values used as indexes to write in the data in **Kasai_I_RAM_1**, **Kasai_I_RAM_2**, **Kasai_Q_RAM_1** and **Kasai_Q_RAM_2** blocks to write in the data for the Color-Doppler mode calculations. It is similar to **Str_WC** except the counter in **Kasai_WC** starts 3 c.c. earlier than in **Str_WC**, since the inputs to RAM do not need to be squared (multiplication es 3 c.c. by default).

a) Part A.

b) Part B.

Figure 3.9: Color-Doppler mode (XSG level).

Kasai_RC_EN enables the counter (**Kasai_C**) to read out the data for the Color-Doppler mode calculations. It is similar to **Str_RC_EN**, except delay blocks that are used to start the read out counter are started 3 c.c. earlier than those in **Str_RC_EN** and those that are used to stop the read out counter are delayed by 896 and 897 c.c., respectively, since there 896 elements in the needed array: $4(\text{rows}) * 8(\text{\# of elements in a row}) * 28(\text{\# of elements used for calculating the value of a single pixel})$. Output **Kasai_RC_EN_C_en1** is later used as write enable signal for **Kasai_Acc1** and **Kasai_Acc2** blocks.

Kasai_C counter provides the indexes to read out the data from RAM blocks for the Color-Doppler mode calculations. It is similar to **Str_C**, however, it provides two counters. One counter provides the values of the following elements need to be evaluated in each row (of 16 elements): 1, 3, 5, 7, 9, 11, 13 and 15. The second counter provides the values for evaluation of the following elements in each row (of 16 elements): 2, 4, 6, 8, 10, 12, 14 and 16 according to (3.3).

When the data are read out from **Kasai_I_RAM_1**, **Kasai_I_RAM_2**, **Kasai_Q_RAM_1** and **Kasai_Q_RAM_2** blocks it is sent to **Kasai_Add** block to evaluate (3.5) and (3.6).

$$(I_{m,n+1}Q_{m,n} - Q_{m,n+1}I_{m,n}) \quad (3.5)$$

$$(Q_{m,n+1}Q_{m,n} + I_{m,n+1}I_{m,n}) \quad (3.6)$$

Division by 28 (mean on 28 elements) is also performed in **Kasai_Add** block. The data are then sent to **Kasai_Acc1** and **Kasai_Acc2** accumulator blocks to find the sum of 28 elements and to **Kasai_28** block to extract every 28th element. The arctangent of $\langle Y \rangle$ and $\langle X \rangle$ is calculated using **Kasai_atan** block.

3.1.4 Velocity Variance (V.V.) Mode

The equation used for estimating pixel values in Velocity-Variance Mode is introduced in this section. It uses the results established in two previous imaging modes. The proposed implementation XSG component level is presented.

3.1.4.1 Theoretical Description of Velocity Variance Mode

Third imaging mode that we will consider is Velocity Variance mode in which the value of a given pixel is calculated according to (3.7).

$$\frac{\sigma_v^2}{f_a^2} = \left(1 - \frac{\sqrt{\langle X \rangle^2 + \langle Y \rangle^2}}{\langle S^2 \rangle} \right) \quad (3.7)$$

In (3.7), $\langle X \rangle$ and $\langle Y \rangle$ are the outputs of Color-Doppler mode (*Kasai*) block and $\langle S^2 \rangle$ is the output of Structural mode (*Structural*) block. When implementing the system in real time, it is necessary that the elements of $\sqrt{\langle X \rangle^2 + \langle Y \rangle^2}$ and $\langle S^2 \rangle$ arrive at the time, i.e. 1st element of numerator must be produced at the same time as 1st element of $\langle S^2 \rangle$ is produced.

3.1.4.2 Proposed V.V. Mode Design with XSG

To evaluate the value of a given pixel in Velocity Variance mode, the values of $\sqrt{X^2 + Y^2}$ and $\langle S^2 \rangle$ must appear at the same time for a given pixel as shown in Figure 3.10. According to (3.7), $\langle Y \rangle$ and $\langle X \rangle$ are squared via *VV_Mult1* and *VV_Mult2*, respectively, and added via *VV_Add1*. *VV_Mult1* and *VV_Mult2* can have no more than 52 integer digits since *VV_Y_In* ($\langle Y \rangle$) and *VV_X_In* ($\langle X \rangle$) have no more than 26 input integer bits, e.g. $2^x \times 2^x = 2^{2x}$. 1 extra bit is used as sign bit. Please note that as a result of multiplication there are 48 fractional bits, but the outputs of *VV_Mult1* and *VV_Mult2* multiplier blocks only have 24 fractional bits and the rest are rounded to the nearest bit.

Figure 3.10: Velocity Variance mode (XSG level).

The output of *VV_Add1* addition block can have no more than 77 bits since only one extra bit can be introduced, e.g. $2^x + 2^x = 2^{x+1}$.

The result of addition is sent to *VV_SQRT* (*CORDIC SQRT*) block, with the parameters equal to the output of *VV_Add1* block: 32 processing elements, input data with of 77 bits, with binary point set at 24 bits. Please note if the input data width and its binary point is different from that specified in *VV_SQRT* (*CORDIC SQRT*) block it could lead to unpredictable results.

$\langle S^2 \rangle$ and $\sqrt{\langle Y^2 \rangle + \langle X^2 \rangle}$ are then adjusted to number of bits – 81, with binary point at 51 bits and sent to *VV_DIVIDER* (*CORDIC DIVIDER*) block. The input parameters of *VV_DIVIDER* (*CORDIC DIVIDER*) block are set as 32 processing elements, input data with of 81 bits, with binary point set at 51 bits. The result of *VV_DIVIDER* is subtracted from constant ‘1’ block (*VV_1*) via *VV_Sub* subtraction block. Please note if the inputs x and y of *VV_DIVIDER* have different data with and binary point it could lead to unpredictable results.

3.2 Summary

The mathematical calculations behind the design were introduced. The design was completed at XSG level and translated to VHDL. The following blocks were created: **SB** for sampling and biasing of the data; **Structural**, **Kasai** and **VV** for evaluating the values of pixels in Structural, Color Doppler and Velocity Variance Imaging modes. In the next Chapter we will compare the results of floating point (MATLAB) and fixed point (VHDL).

Chapter 4

RESULTS

In this Chapter, behavioral simulation of the blocks Chapters will be performed. The results will be compared with floating point results obtained by MATLAB. Before proceeding, the way to create testbench will be presented in section 4.1.

4.1 General flow

In order to transfer the design for Simulink to Xilinx ISE, System Generator tool was used. FPGA clock period is specified to be 40 ns, part number is Virtex2P xc2vp100-6ff1704, compilation option is netlist and Simulink system period is 1 (used in previous Chapters for XSG simulations). In each demo **memory** and **counter** blocks will be used. **Memory** block provides data pre-loaded in ROM block of FPGA, and **counter** block is used to read it out. 128 elements of input demo data shown in (Table 3.1) were used for demonstration purposes. Throughout the simulations 40ns c.c. length was used (25 MHz clock rate) with offset of 100ns and Global Set/Reset (GSR) support of 100ns (Xilinx recommendation). GSR emulates Power-On-Reset of FPGA hardware.

The input data from Table 3.1 is converted to the binary format with 12 bits used to represent each number with binary point set at '0' (no decimal places in the input). It is then stored in *m2_coe.coe* file, where keyword *memory_initialization_radix* is used for memory initialization values to specify the radix used and *memory_initialization_vector* is used for distributed memory. **Memory** block is generated using *Distributed Memory Generator v3.3* core. The *Distributed Memory Generator* core is used to create memory structures using the distributed ROM as a very large look-up table. Depth is the number of words (128 in demo case)

and width is the width of each word (12 in demo case). Values from *m2_coe.coe* file are loaded as initial values of the memory elements.

After ISE project is generated, it is open in Xilinx ISE; schematic symbol is created for the generated block and **counter** and **memory** blocks. Once top level schematic is created, top level test bench is created.

Initial clock timing for simulation is set as: *Clock High Time* (20ns) and *Clock Low Time* (20ns) comprise 1 c.c. *High for Initial* sets the period of the GSR in design. Xilinx recommends always use the default of 100 ns. *Initial length* can be varied depending on how many c.c. the simulation needs to be run for.

When test bench is generated, *pn_behavioral.do* is also generated, which can be used as they custom *do* file during simulations. In *pn_behavioral.do* file *vcom* command compiles VHDL source code into a specified working library (or to the work library by default), *vsim* command is used to invoke the VSIM simulator and initial run time can be specified.

4.2 Simulation Results

The entire process of creating the simulation is shown for **Sampling** block with the similar process repeated for all other tested blocks.

4.2.1 Sampling Block test

The XSG block used for testing is shown in Figure 4.1, with the details of **Sampling** Block shown in Figure A2.1.

Figure 4.1: Sampling test block (XSG).

The resultant Sampling block is shown in Figure 4.2, where the output nets are *Isampled* and *Qsampled* with data width of 12 bits (0 to 11).

Figure 4.2: Sampling test block schematic (ISE).

Figure 4.3 shows the *clk* signal, output of data from memory block (*gateway_in_net*), and *I_sampled* and *Q_sampled* outputs – *i_sampled_net* and *q_sampled_net*. As can be seen from Figure 4.3 *I_sampled* and *Q_sampled* outputs are generated three c.c. later than the first output of **memory** block is generated. The first output of **memory** block is generated at 200 ns (100ns delay from High Initial and 100 ns from offset).

Figure 4.3: First two Sampling block outputs.

In Figure 4.3 data is shown in binary format. $I_sampled$ and $Q_sampled$ outputs consist of 16 bits, with LSB being decimal place. For example, the first $I_sampled$ output is ‘0.5’ with base 10 or ‘0000000000000001’ in binary notation. Channel I is delayed has 1ns uncertainty compared to Channel Q when non-zero element is output, as shown in Figure 4.3. Also uncertainty appears on Channel I not Channel Q as shown in Figure 4.3, it actually appears on both Channels. The reason the uncertainty is not seen at the output of Channel Q is because this uncertainty is removed by flip-flops in *Sample_Delay1* block. This uncertainty does not affect design functionality.

The last two *Sampling* block outputs are shown in Figure 4.4. The last element of $I_Sampled$ and $Q_Sampled$ is ‘0000000001000000’ in binary notation with 16 bits used to represent each element and the binary point set at 1 or $0(2^{10}) + 0(2^9) + 0(2^8) + 0(2^7) + 0(2^6) + 1(2^5) + 0(2^4) + 0(2^3) + 0(2^2) + 0(2^1) + 0(2^0) + 0(2^{-1}) = 32$ in decimal notation.

Figure 4.4: Last two Sampling block outputs.

Let's compare fixed point (VHDL) simulation results with the first row of results obtained using floating point (MATLAB). The detailed calculations are provided in Appendix A5. As can be seen, the results are identical.

Table 4.1: Sampling block behavioral simulation results comparison.

Floating point	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5	7.0	7.5	8.0
Fixed point	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5	7.0	7.5	8.0

4.2.2 Biasing block test

The first output of *Biasing* block is generated at 320 ns, and the last output is generated at 5440 ns, hence the difference is 5120 ns or 128 c.c. of 40 ns. The reason it is 128 c.c. is because as the result of upsampling in *Sampling* block, every other c.c. has the value of '0' (or '-1.5' as the result of bias removal). *We_bias* signal is generated 1 c.c. after the first output is generated (same as in XSG design) and there is 1ns delay due to the delay in combinational logic.

Figure 4.5: First three Biasing block outputs.

The elements of the first row of *I_bias* (Table 4.2) and *Q_bias* (Table 4.3) outputs are compared with the results obtained using floating point. As can be observed from the above calculations, the results achieved from behavioral simulation for *I_bias* and *Q_bias* are identical to those obtained using floating point from MATLAB. The detailed calculations are presented in Appendix A5. Please note *I_bias* and *Q_bias* outputs have 16 bits representing each element with binary point set at 4.

Table 4.2: Biasing block behavioral simulation results comparison (*I_bias*).

Element #	1	2	3	4	5	6	7	8
Floating point	-1	-0.5	0.0	0.5	1.0	1.5	2.0	2.5
Fixed point	-1	-0.5	0.0	0.5	1.0	1.5	2.0	2.5
Element #	9	10	11	12	13	14	15	16
Floating point	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5
Fixed point	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5

Table 4.3: Biasing block behavioral simulation results comparison (Q_bias).

Element #	1	2	3	4	5	6	7	8
Floating point	-6.1250	-5.6875	-5.2500	-4.8125	-4.3750	-3.9375	-3.5000	-3.0625
Fixed point	-6.1250	-5.6875	-5.2500	-4.8125	-4.3750	-3.9375	-3.5000	-3.0625
Element #	9	10	11	12	13	14	15	16
Floating point	-2.6250	-2.1875	-1.7500	-1.3125	-0.8750	-0.4375	0.0000	0.4375
Fixed point	-2.6250	-2.1875	-1.7500	-1.3125	-0.8750	-0.4375	0.0000	0.4375

4.2.3 Structural block test

The comparison between the results obtained from 32 element addition for pixel (1, 1) of floating point vs. *Str_Acc1* Block (shown in Figure A2.3) is presented in Table 4.4. As it can be seen from Table 4.4 the results are the same, meaning that using fixed point did not introduce any errors thus far.

Table 4.4: $I^2/(M \times N)$ for element (1, 1) of the array.

Element #	$I^2/(M \times N)$ Floating point	$I^2/(M \times N)$ Fixed point	$I^2/(M \times N)$ addition Floating point	$I^2/(M \times N)$ addition Fixed point
1, 1	0.0312500	0.0312500	0.0312500	0.0312500
1, 2	0.0312500	0.0312500	0.0625000	0.0625000
1, 3	0.2812500	0.2812500	0.3437500	0.3437500
1, 4	0.7812500	0.7812500	1.1250000	1.1250000
1, 5	1.5312500	1.5312500	2.6562500	2.6562500
1, 6	2.5312500	2.5312500	5.1875000	5.1875000
1, 7	3.7812500	3.7812500	8.9687500	8.9687500
1, 8	5.2812500	5.2812500	14.2500000	14.2500000
2, 1	0.0078125	0.0078125	14.2578125	14.2578125
2, 2	0.0703125	0.0703125	14.3281250	14.3281250
2, 3	0.3828125	0.3828125	14.7109375	14.7109375
2, 4	0.9453125	0.9453125	15.6562500	15.6562500
2, 5	1.7578125	1.7578125	17.4140625	17.4140625
2, 6	2.8203125	2.8203125	20.2343750	20.2343750
2, 7	4.1328125	4.1328125	24.3671875	24.3671875
2, 8	5.6953125	5.6953125	30.0625000	30.0625000
3, 1	0.0000000	0.0000000	30.0625000	30.0625000
3, 2	0.1250000	0.1250000	30.1875000	30.1875000
3, 3	0.5000000	0.5000000	30.6875000	30.6875000
3, 4	1.1250000	1.1250000	31.8125000	31.8125000
3, 5	2.0000000	2.0000000	33.8125000	33.8125000
3, 6	3.1250000	3.1250000	36.9375000	36.9375000
3, 7	4.5000000	4.5000000	41.4375000	41.4375000
3, 8	6.1250000	6.1250000	47.5625000	47.5625000
4, 1	0.0078125	0.0078125	47.5703125	47.5703125
4, 2	0.1953125	0.1953125	47.7656250	47.7656250
4, 3	0.6328125	0.6328125	48.3984375	48.3984375
4, 4	1.3203125	1.3203125	49.7187500	49.7187500
4, 5	2.2578125	2.2578125	51.9765625	51.9765625
4, 6	3.4453125	3.4453125	55.4218750	55.4218750
4, 7	4.8828125	4.8828125	60.3046875	60.3046875
4, 8	6.5703125	6.5703125	66.8750000	66.8750000

The final results of *Structural* Block are shown in Figure 4.6 and Table 4.5. Figure 4.6 shows the *clk* signal, and *S2_out* output – *s2_out_net*.

Table 4.6: Num/(Mx(N-1)) for pixel (1, 1).

Element	Num/(Mx(N-1)) Floating point	Num /(Mx(N-1)) Fixed point	Num /(Mx(N-1)) addition Floating point	Num /(Mx(N-1)) addition Fixed point
1, 1	-0.3750000000000000	-0.375068664550781	-0.3750000000000000	-0.375068664550781
1, 2	-0.3750000000000000	-0.375068664550781	-0.7500000000000000	-0.750137329101562
1, 3	-0.3750000000000000	-0.375068664550781	-1.1250000000000000	-1.125205993652344
1, 4	-0.3750000000000000	-0.375068664550781	-1.5000000000000000	-1.500274658203125
1, 5	-0.3750000000000000	-0.375068664550781	-1.8750000000000000	-1.875343322753906
1, 6	-0.3750000000000000	-0.375068664550781	-2.2500000000000000	-2.250411987304688
1, 7	-0.3750000000000000	-0.375068664550781	-2.6250000000000000	-2.625480651855469
2, 1	-0.3750000000000000	-0.375068664550781	-3.0000000000000000	-3.000549316406250
2, 2	-0.3750000000000000	-0.375068664550781	-3.3750000000000000	-3.375617980957031
2, 3	-0.3750000000000000	-0.375068664550781	-3.7500000000000000	-3.750686645507812
2, 4	-0.3750000000000000	-0.375068664550781	-4.1250000000000000	-4.125755310058594
2, 5	-0.3750000000000000	-0.375068664550781	-4.5000000000000000	-4.500823974609375
2, 6	-0.3750000000000000	-0.375068664550781	-4.8750000000000000	-4.875892639160156
2, 7	-0.3750000000000000	-0.375068664550781	-5.2500000000000000	-5.250961303710938
3, 1	-0.3750000000000000	-0.375068664550781	-5.6250000000000000	-5.626029968261719
3, 2	-0.3750000000000000	-0.375068664550781	-6.0000000000000000	-6.001098632812500
3, 3	-0.3750000000000000	-0.375068664550781	-6.3750000000000000	-6.376167297363281
3, 4	-0.3750000000000000	-0.375068664550781	-6.7500000000000000	-6.751235961914062
3, 5	-0.3750000000000000	-0.375068664550781	-7.1250000000000000	-7.126304626464844
3, 6	-0.3750000000000000	-0.375068664550781	-7.5000000000000000	-7.501373291015625
3, 7	-0.3750000000000000	-0.375068664550781	-7.8750000000000000	-7.876441955566406
4, 1	-0.3750000000000000	-0.375068664550781	-8.2500000000000000	-8.251510620117188
4, 2	-0.3750000000000000	-0.375068664550781	-8.6250000000000000	-8.626579284667969
4, 3	-0.3750000000000000	-0.375068664550781	-9.0000000000000000	-9.001647949218750
4, 4	-0.3750000000000000	-0.375068664550781	-9.3750000000000000	-9.376716613769531
4, 5	-0.3750000000000000	-0.375068664550781	-9.7500000000000000	-9.751785278320312
4, 6	-0.3750000000000000	-0.375068664550781	-10.1250000000000000	-10.126853942871094
4, 7	-0.3750000000000000	-0.375068664550781	-10.5000000000000000	-10.501922607421875

The difference between the final result of floating point and fixed point simulations is 0.0183%. The difference exists due to the fact that MATLAB uses higher precision during the calculation process as $1/28 = .03571428571428571428\dots$ not 0.03571428571428571 used in *Kasai_Add* block shown in Figure A2.5.

Figure 4.7 shows the *clk* signal, and *arctan* output – *atan_out_net*.

Figure 4.7: First two Kasai block outputs.

The elements of the first row of *atan_out_net* outputs are compared with the outputs shown in floating point calculations. Please note *atan_out_net* outputs have 50 bits representing each element with binary point set at 24.

Table 4.7: Kasai block behavioral simulation results comparison (arctan).

Element #	(1, 1)	(1, 2)	(1, 3)	(1, 4)
Floating point	-0.1427992942	-0.0655829232	-0.0346156809	-0.0208692089
Fixed point	-0.1427992582	-0.0655828714	-0.0346148014	-0.0208691359
Element #	(1, 5)	(1, 6)	(1, 7)	(1, 8)
Floating point	-0.0138257461	-0.0117360430	-0.0100120008	-0.0085902099
Fixed point	-0.0138257742	-0.0117361546	-0.0100120306	-0.0085951090

The maximum difference between the results of floating point and fixed point simulations is 0.057% which might not be discernable by human eye. The difference exists due to the fact that MATLAB uses more efficient algorithm during the entire calculation process and the number of decimal places is not necessarily equal to the number of fractional bits.

4.2.5 VV Block Test

Figure 4.8: First VV block output.

Figure 4.8 shows the *clk* signal, and *vv* output – *vv_out_net*. The elements of the first row of *vv_out_net* outputs are compared with the results produced by floating point calculations.

Please note *vv_out_net* outputs have 113 bits representing each element with binary point set at 32.

Table 4.8: VV block behavioral simulation results comparison (vv_out).

Element #	(1, 1)	(1, 2)	(1, 3)	(1, 4)
Floating point	-37.3807274208	-70.7936114219	-120.5214035029	-184.6256726980
Fixed point	-37.7500000298	-71.4843749702	-121.6562499404	-186.3749999404
Element #	(1, 5)	(1, 6)	(1, 7)	(1, 8)
Floating point	-262.1143645887	-224.7652834635	-164.1209174336	-69.1983103099
Fixed point	-264.5937501192	-226.8789063692	-165.6533202529	-69.8596191108

The maximum difference between the results of floating point and fixed point simulations is 0.948%. The difference exists due to the fact that MATLAB uses higher precision during the entire calculation process and the errors from the previous stages added up.

4.3 Resources Report

In Table 4.9 device utilization summary is shown. The functions such as *natural logarithm*, *arctan*, *square root* and *division* consume large amount of resources.

Table 4.9: Device utilization summary (estimated values).

Logic Utilization	Used	Available	Utilization
Number of Slices	22291	44096	50%
Number of Slice Flip Flops	24293	88192	27%
Number of 4 input LUTs	36571	88192	41%
Number of bonded IOBs	334	1040	32%
Number of BRAMs	34	444	7%
Number of MULT18X18s	62	444	13%
Number of GCLKs	1	16	6%

4.4 Timing Comparison

In Table 4.10 the results obtained from comparison the execution times on PC and from FPGA simulation are shown.

Table 4.10: Results acquired from algorithms experimentation.

	Xilinx Virtex II Pro xc2vp100-6ff1704	Intel Core2 Duo PC
Algorithms executions at particular frequency	80.4 μ s at 25 MHz	3.75 ms running at 2000MHz
Acceleration (real) in comparison to PC	46.7 times	1
Acceleration (perspective) in comparison to PC	3676	1

The results from PC were obtained using the average of 10 runs of *tic toc* MATLAB function. As it can be seen, running only at 25 MHz, FPGA simulation performs 46.7 times faster than PC. If we are going to compare acceleration at frequency of 25 MHz, the speed up of FPGA platform will be 3676 times. PC did have other processes running at the same time; hence, it might have performed better if there were no other programs loaded. However, this result does show the acceleration from FPGA in estimation of the values of pixels in two imaging modes (Structural and Color Doppler Flow) in parallel.

4.5 Summary

In this chapter, behavioral simulations of each designed block (using fixed point architecture) were carried out. The results obtained in this chapter were compared to those obtained by MATLAB simulations. The maximum difference between the results of floating point and fixed point simulations is 0.948%. The difference exists due to the fact that MATLAB uses higher precision during the entire calculation process and the errors from the previous stages

added up. This confirms the design implementation is correct at VHDL level. The performance at FPGA simulation level is shown to be faster than at a PC.

Chapter 5

CONCLUSIONS AND FUTURE WORK

5.1 Major Conclusions

In this project a system capable of estimating pixel values in three imaging modes: Structural, Color-Doppler and Velocity Variance was realized at VHDL level.

Our system is capable of reading recorded and real time data. After the data were read in and sampled in Channels I and Q , bias and gain mismatch were removed from the input data so that the pixel values in different imaging modes can be evaluated correctly.

Lateral averaging was introduced in order to reduce aliasing and to improve SNR in all imaging modes. Also the cases when the algorithm exceeds the range of the input data are discussed and a solution is proposed.

Pixel value estimation was developed for Structural Imaging Mode using XSG blocks. Their implementation was verified by comparing the results with MATLAB. RAM blocks were used as temporary storage for Channels I and Q , which were required for Structural Imaging Mode calculations.

Reading out the data from two different (consecutive) addresses of RAM memory using two indexes (counters) simultaneously was introduced in order to perform the estimation of the pixel values for Color Doppler Imaging Mode. The mean velocity can be estimated using the Kasai autocorrelation equation.

Finally, the calculations for Velocity Variance Imaging Mode were carried out in order to enable displaying turbulent Doppler flow since flow turbulence is usually characterized by high flow velocities and variances.

The errors were kept $<1\%$ comparing to the results obtained by MATLAB. The errors exist due to the fact that there is a limited number of bits available to represent decimal places in a number using simple blocks such as *conversion* and *multiplication* and complex blocks such as *ln*, *division*, *square root*, etc. The number of bits required to represent the output of each block has been decided as a tradeoff between the accuracy and the number of FPGA resources.

The design was translated into VHDL and in order to verify validity of the translation behavioral simulations of each designed block were carried out using ModelSim.

The FPGA platform was found to accelerate the computation of streamed data.

5.2 Future Work

This design needs to be downloaded onto an FPGA to verify that no further errors are introduced. The design needs to be modified for processing consecutive frames by running calculations at higher rate than the data is read in and designing control circuit that supports multi-frame processing. It is also be necessary to change the frame size to suit the input scanned frame. Image creation algorithm and outputting images to a monitor should be final steps in displaying the created images. For specific applications where the data needs to be processed at high frame rates, the design may need to be implemented on ASIC.

BIBLIOGRAPHY

- [1] D. Morofke, *2-D Kasai Velocity Estimation for Doppler Optical Coherence Tomography*, MSc Thesis, Ryerson University, 2006.
- [2] P. Tomlins and R. Wang, "Theory, developments and applications of optical coherence tomography", *Journal of Physics D: Applied Physics*, vol. 38, Issue 15, pp. 2519-2535, 2005.
- [3] Z. Chen, T. Milner, X. Wang, S. Srinivas and J. Nelson, "Characterization of blood flow dynamics and tissue structure using optical Doppler tomography," *Photochemistry and Photobiology*, Vol. 67, pp. 56-60, 1998.
- [4] E. I. Galanzha, V. V. Tuchin, V. P. Zharov, "Advances in small animal mesentery models for in vivo flow cytometry, dynamic microscopy, and drug screening," *World Journal of Gastroenterology*, Volume 13, Issue 2, pp. 192-218, 2007.
- [5] E. Sekizuka, C. Ohshio, and H. Minamitani, "Automatic analysis of moving images for the lymphocyte velocity measurement," *Microcirculation annual*, Volume 11, pp. 107-108, 1995.
- [6] J. B. Dixon, D. C. Zawieja, A. A. Gashev, G. L. Cote, "Measuring microlymphatic flow using fast video microscopy," *Journal of Biomedical Optics*, Volume: 10, Issue 6, pp. 064016, 2005.
- [7] E. I. Galanzha, V. V. Tuchin, V. P. Zharov, "In vivo integrated flow image cytometry and lymph/blood vessels dynamic microscopy," *Journal of Biomedical Optics*, Volume:10, pp. 054018, 2005.
- [8] D. Huang, E. A. Swanson, C. P. Lin, J. S. Schuman, W. G. Stinson, W. Chang, M. R. Hee, T. Flotte, K. Gregori, C. A. Puliafito, and J. G. Fujimoto, "Optical Coherence Tomography", *Science*, vol. 254, pp.1178-1181, 1991.
- [9] B. Bouma, and G. J. Tearney, *Handbook of optical coherence tomography*, published by Taylor & Francis, New York, 2002.
- [10] P. M. Fishbane, S. Gasiorowicz, S. T. Thornton, *Physics for Scientists and Engineers with Modern Physics*, 3rd edition. Benjamin-Cummings Publishing Company, 2004.
- [11] J. A. Jensen, *Estimation of Blood Velocities Using Ultrasound: a Signal Processing Approach*. Cambridge University Press, 1996.
- [12] V. X. D. Yang and I. A. Vitkin, "Principles of Doppler optical coherence tomography", in *Handbook of Optical Coherence Tomography in Cardiology*, edited by Evelyn Regar, Ton van Leeuwen and Patrick Serruys, (Taylor and Francis Medical, Oxford, UK: 2006), chapter 32.
- [13] S.H. Yun, G.J. Tearney, J.F. de Boer, N. Iftimia, and B.E. Bouma, "High-speed optical frequency-domain imaging," *Optics Express*, vol. 11, Issue 22, pp. 2953-2963, 2003.
- [14] S. Chinn, E. Swanson, and J. Fujimoto, "Optical coherence tomography using a frequency-tunable optical source," *Optics Letters*, vol. 22, Issue 5, pp. 340-342, 1997.
- [15] C. Deane, "Doppler in Obstetrics Chapter in Doppler in Obstetrics," by series editor: Kypros Nicolaides, Giuseppe Rizzo, Kurt Hecker and Renato Ximenes, *Diploma in Fetal Medicine and ISUOG Educational Series*, 2002.
- [16] J. J. Wild, "The use of ultrasonic pulses for the measurement of biological tissues and the detection of tissue density changes," *Surgery*, vol. 27, pp. 183-188, 1950.

- [17] J. T. Bushberg, J. A. Seibert, E. M. Leidholdt, J. M. Boone, *The Essential Physics of Medical Imaging*. Lippincott Williams & Wilkins, 2002.
- [18] K. K. Shung, *Diagnostic Ultrasound—Imaging and Blood Flow Measurement*. Taylor & Francis Group (CRC Press), 2006.
- [19] V. X. D. Yang, M. L. Gordon, B. Qi, J. Pekar, S. Lo, E. Seng-Yue, A. Mok, B. C. Wilson, and I. A. Vitkin, "High speed, wide velocity dynamic range Doppler optical coherence tomography (Part I): System design, signal processing, and performance," *Optics Express*, vol. 11, pp. 794-809, 2003.
- [20] W. R. Hendee, E. R. Ritenour. *Medical imaging physics*, 4th edition. New York, NY: Wiley, 2002.
- [21] Z. Ding, Y. Zhao, H. Ren, J. Nelson and Z. Chen, "Real-time phase-resolved optical coherence tomography and optical Doppler tomography", *Optics Express*, Volume 10, Issue 5, pp. 236-245, 2002.
- [22] A. Rollins, S. Yazdanfar, J. Barton, and J. Izatt, "Real time color Doppler optical coherence tomography using an autocorrelation technique," *Proceedings of the SPIE*, Volume 3598, pp. 168-176, 1999.
- [23] V. Westphal, "Real-time, high velocity resolution Color Doppler OCT", *Proceedings of the SPIE*, Volume 4251, pp. 188-191, 2001.
- [24] V. Yang, M. L. Gordon, A. Mok, Y. Zhao, Z. Chen, R. Cobbold, B. C. Wilson, and I. A. Vitkin, "Improved phase-resolved optical Doppler tomography using the Kasai velocity estimator and histogram segmentation," *Optics Communications*, Volume 208, pp. 209–214, 2002.
- [25] C. Kasai, K. Namekawa, A. Koyano and R. Omoto, "Real-Time Two-Dimensional Blood Flow Imaging Using an Autocorrelation Technique," *IEEE Transactions on Ultrasonics*, vol. SU-32, pp. 458-464, 1985.
- [26] H. Larsen, R. Nilson, L. Thrane, D. Kristensen, and P. Andersen, "Optical Doppler coherence tomography based on a field-programmable gate array," *Proceedings of the SPIE*, Volume 5690, pp. 263-270, 2005.
- [27] A. Schaefer, J. Reynolds, D. Marks, and S. Boppart, "Real-time digital signal processing-based optical coherence tomography and Doppler optical coherence tomography," *IEEE Transactions on Biomedical Engineering*, Volume 51, Issue 1, pp. 186- 190, 2004.
- [28] A. Rollins, et al., "Real-time in vivo color Doppler optical coherence tomography", *Journal of Biomedical Optics*, Volume 7, Issue 1, pp. 123-129, 2002.
- [29] S. Yan, "Digital signal processor-based real-time optical Doppler tomography system", *Journal of Biomedical Optics*, Volume 9, Issue 3, pp. 454-63, 2004.
- [30] D. Piao, L. Otis, N. Dutta and Z. Quing, "Quantitative assessment of flow velocity estimation algorithms for optical Doppler tomography imaging," *Applied Optics*, Volume 41, pp. 6118–6127, 2002.
- [31] J. E. Volder, "The CORDIC Trigonometric Computing Technique", *IRE Transactions on Electronic Computers*, Volume EC-8, pp. 330-334, 1959.
- [32] H. Natarajan, *Implementation of Chirp-Z Transform on FPGA*. Florida State University, MSc Thesis, 2004.
- [33] "Virtex II Pro and Virtex II Pro X platform FPGAs: Complete Data Sheet," Xilinx Corporation, DS083, June 2004.
- [34] C. Eddington, "How to use M and Simulink for DSP control and datapath design", *Programmable Logic DesignLine*, 2007.

- [35] J. W. Goodman, "Some fundamental properties of speckle", *Journal of Optical Society of America*, vol. 66, pp. 1145-1150, 1976.
- [36] A. Oppenheim, R. Schaffer, and J. Buck, "*Discrete-Time Signal Processing*", 2nd ed., Prentice Hall, 1998.
- [37] J. M. Schmitt, S. H. Xiang, and K. M. Yung, "Speckle in optical coherence tomography," *Journal of Biomedical Optics*, vol. 4, pp. 95-105, 1999.
- [38] U. Bencharit, J. L. Kaufman, N. M. Bilgutay and J. Saniie, "Frequency and spatial compounding techniques for improved ultrasonic imaging," *IEEE Ultrasonics Symposium*, pp. 1021-1026, 1986.
- [39] K. Abernethy and T. Allen, *Exploring the Digital Domain*, 2nd edition, Thomson Learning, Inc., 2004.

Appendices

A1 Glossary

2's complement

2's complement is one of the ways to encode negative numbers into ordinary binary.

Positive 2's complement numbers are represented as the simple binary with the MSB being '0'. Negative 2's complement numbers are represented as the binary number that when added to a positive number of the same magnitude equals zero. MSB of negative 2's complement numbers is '1'.

Please note since 2's complement of binary numbers is used to represent signed numbers, one extra bit is required for the sign notation.

3 state buffer

It has 3-state control pin and input pin. The 3-state buffer logic is implemented using AND-OR logic rather than 3-state drivers, so that timing is more predictable and less load dependant especially with larger devices. TBUFs in neighboring CLBs can access slice outputs by direct connects.

Beat

Interference between two sounds of slightly different frequencies as shown in Figure A1.1.

Figure A1.1: Beats formation: a) individual waves, b) combined wave [5].

Block memory

In Virtex FPGAs block memory provides efficient data buffering and temporary storage for processing information found in graphics and networking applications. The block memory uses the predefined memory block on the Xilinx chip.

Block SelectRAM+ Memory

The block SelectRAM+ memory resources are 18 Kb of True Dual-Port RAM, programmable from 16K x 1 bit to 512 x 36 bit, in various depth and width configurations.

Coherent Demodulation

It is accomplished by remodulating using a local oscillator (LO) which is at the same frequency and in phase with the original carrier signal.

CLB

Configurable Logic Block which includes four slices, two 3-state buffers, 8 LUTs, 8 Flip-Flops, 8 MULT_ANDs, 2 Arithmetic Carry Chains, 2 SOP Chains, 128 bits of Distributed SelectRAM+ and 128 bits of Shift Registers.

Figure A1.2: CLB [33].

D flip-flop

The function of this flip-flop is to introduce a unit delay in the signal input at D. These flip flops can be used as the basis for shift registers. The next state of the D flip-flop is the same as the data input at any time, regardless of the present state.

Truth table:

Q(t)	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

Demodulation

The process to get the original baseband signal by removing the modulation from an analog signal.

Distributed memory

It is ideal for high speed data processing systems where a small amount of wide memory is needed. Distributed memory utilizes the LUTs configured as memory modules.

Flip-flop

A flip-flop is a bistable multivibrator that has two stable states and thereby is capable of serving as one bit of memory. It is usually controlled by one or two control signals and/or a gate or clock signal. The output often includes the complement as well as the normal output.

In the living tissue of a whole, living organism.

The combination of separate waves in the same region of space to produce a resultant wave.

Device that splits a light beam into two parts and then recombines the parts to form an interference pattern.

Device used to generate a signal which is beat (interfered) against the signal of interest to mix it to a different frequency.

Process where a Radio Frequency or Light Wave's amplitude, frequency, or phase is changed in order to transmit the signal.

Number of words in the RAM.

Carrier-recovery technique, which allows demodulation of incoming signals and recovery of message waveform (carrier signal).

Each slice in Virtex II Pro contains

- Two function generators (F & G)
- Two storage elements
- Arithmetic logic gates
- Large multiplexers
- Wide function capability
- Fast carry look-ahead chain
- Horizontal cascade chain (OR gate)

Figure A1.3: Slice [33].

Speckle

Background echo present from tissue characterization in the received signal, which arises as a natural consequence of the limited spatial-frequency bandwidth of the interference signals measured in optical coherence tomography (OCT) [37] and results in an interference pattern caused by closely spaced (unresolved) and randomly distributed scatterers [38].

Shift-Register LUT (SRL)

It is a shift register of variable length. It has some number of address inputs which select which bit of the shift register to send to the output.

TBUF

See 3 state buffer.

Transmittance (in optics)

Fraction of incident light at a specified wavelength that passes through a sample.

XST

XST is a Xilinx® tool that synthesizes HDL designs to create Xilinx® specific netlist files called NGC files.

Wrap

Wrapping is when a variable that exceeds its maximum value returns to its minimum value and vice-versa.

A2 Selected subsystems used in the design

Figure A2.1: Sampling block (XSG level).

Figure A2.2: RIRO block (XSG level).

Figure A2.3: Str_Acc1 Accumulator block (XSG level).

Figure A2.4: Str_DR block diagram (XSG level).

Figure A2.5: Kasai_Add block (XSG level).

A3 Selected Xilinx System Generator Blocks used in the design

The information specified here are excerpts from Xilinx Simulink manuals with the information persistent to this project added.

Accumulator

This Xilinx block implements an adder or subtractor-based scaling accumulator. We used adder-based accumulators throughout the design. The block's current input is accumulated with a scaled current stored value. Scaling factor of '1' was used. Reinitialize with input 'b' on reset option was selected so that the output of the accumulator is reset to the data on input port b, not to zero. The reason that had to be done, we had to make sure all values in the input array to the accumulator were captured.

AddSub

This Xilinx block implements an adder or subtractor or adder/subtractor. In this design, we used both adders and subtractors. The output precision was chosen depending on the input(s).

CMult

The Xilinx CMult block implements a gain operator, the output of which is equal to the product of its input multiplied by a constant value. Constant values were specified in the design. If the constant cannot be expressed exactly in the specified fixed-point type, its value is rounded and saturated as needed. A positive value is implemented as an unsigned number, a negative value as signed. The other two parameters which relate to the input are "Constant Number of bits" and "Constant Binary point". Constant Number of bits specifies the bit location of the binary point of the constant, where bit zero is the least significant bit. Constant Binary point specifies position of the binary point. The Output Type tab defines the precision of the output of the CMult block. The output precision must be user defined and varies throughout the design depending on the precision required. Even though the number of bits for each step was calculated, Saturate options was chosen in case of overflow.

Convert block

This block can change the output to the following data types: Boolean, Signed (2's complement) and Unsigned. For Signed (2's complement) and Unsigned data types, the number of bits and binary point can be varied. The binary point parameter indicates the number of bits to the right of the binary point (i.e., the size of the fraction) for the output port. If the number of fractional bits is insufficient to represent the fractional portion of a value, the data can be round to the nearest representable value (or to the value furthest from zero if there are two equidistant nearest representable values), or truncated (i.e., bits to the right of the least significant representable bit are discarded). Throughout the design rounding to the nearest representable value was used.

In case overflow occurs, the options are to Saturate to the largest positive/smallest negative value, to Wrap (i.e., to discard bits to the left of the most significant representable bit), or to Flag as error (an overflow as a Simulink error) during simulation. Flag as error is a simulation only feature. The hardware generated is the same as when Wrap is selected. Even though the number of bits for each step was calculated, Saturate options was chosen in case of overflow.

There are two implementation options: pipeline for maximum performance and do not use pipelining. When “pipeline for maximum performance” is check on, it directs the block to use the latency value to pipeline to the fullest extent possible. Latency is distributed in the following priority based on the selected pipeline option.

Latency (pipeline=0): Delay pipeline at the output.

Latency (pipeline=1): Output register, register before saturation, register before quantization and extra latency as a delay pipeline at the output.

CORDIC ATAN

This Xilinx reference block implements a rectangular-to-polar coordinate conversion using a fully parallel CORDIC (algorithm in Circular Vectoring mode).

That is, given a complex-input $\langle x, y \rangle$, it computes a new vector $\langle m, a \rangle$, where magnitude $m = K \times \sqrt{x^2 + y^2}$, and the angle $a = \arctan(y/x)$. As is common, the magnitude scale factor $K = 1.646760\dots$ is not compensated in the processor, i.e. the magnitude output should be scaled by this factor. Only an angle output (*atan*) was utilized in the design.

CORDIC LOG

This Xilinx reference block implements a natural logarithm circuit using a fully parallel CORDIC algorithm. Only a natural logarithm output (*log x*) was utilized in the design.

Counter

In this project only count-limited type of an up counter was used. It is implemented by combining a free running counter with a comparator. Count limited counters are limited to only 64 bits of output precision. Count limited types of a counter can be configured to step between the initial and ending values, provided the step value evenly divides the difference between the initial and ending values. Unsigned counters were used with *number of bits*, *initial value*, *count to value* and *step* options modified depending on the design specification. Counters used for addressing RAM blocks had to be of the type unsigned, as that was the requirement of RAM blocks. The block uses the Xilinx LogiCORE Counter. Please note that when the counter is disabled it still outputs the last produced output.

Delay block

The delay value is displayed on the block in the form z^{-L} , which is the Z-transform of the block's transfer function. Any data provided to the input of the block will appear at the output after L cycles, where the number of cycles is a non-negative integer specified by user. The rate and type of the data of the output will be inherited from the input. This block is used mainly for matching pipeline delays in other portions of the circuit. For delays longer than 128 c.c.'s, block-RAM-based delay blocks are used. The delay block differs from the register block in that the register allows a latency of only 1 cycle and contains an initial value parameter.

The logic synthesis can be done using Behavioral HDL. This setting is recommended Synplify Pro is used as the downstream logic synthesis tool. The logic synthesis tool will implement the delay as it desires, performing optimizations such as moving parts of the delay line back or forward into blockRAMs, DSP48s, or embedded IOB flip flops; employing the dedicated Shift-Register LUT (SRL) cascade outputs for long delay lines based on the architecture selected; and using flip flops to terminate either or both ends of the delay line based on path delays. However, since Synplify Pro is not used in this project, “*Implement using behavioral HDL*” box should be

left unchecked. The logic synthesis will be performed using Structural HDL. This is the default setting and results in a known, but less-flexible, implementation which is often better for use with XST. In general, this setting produces structural HDL comprising an SRL delay of (L-1) cycles followed by a flip flop, with the SRL and the flip flop getting packed into the same slice. For a latency greater than L=17, multiple SRL/flip flop sets will be cascaded, albeit without using the dedicated cascade routes.

Down Sample

The input signal is sampled at even intervals, at either the beginning (first value) or end (last value) of a frame. The sampled value is presented on the output port and held until the next sample is taken.

There are three possible hardware implementations that vary in terms of implementation efficiency: zero-latency down sample, down sample with latency: sample first value in the frame and sample last value in the frame. The block receives two clock enable signals in hardware, *Src_CE* and *Dest_CE*. *Src_CE* is the faster clock enable signal and corresponds to the input data stream rate. *Dest_CE* is the slower clock enable, corresponding to the output stream rate, i.e., down sampled data. These enable signals control the register sampling in hardware. The zero latency Down Sample block can only be used to sample the first value of the frame. It is the least efficient hardware implementation as the mux introduces a combinational path from *Din* to *Dout*. Therefore, this implementation was not used.

Both Down Sample with latency sample first value in the frame and Down Sample with latency sample last value in the frame blocks were used in the design. When using Down Sample with latency sample first value in the frame two registers are required to correctly sample the input stream. The first register is enabled by the adjusted clock enable signal so that it samples the input at the start of the input frame. The second register samples the contents of the first register at the end of the sample period to ensure output data is aligned correctly.

Figure A3.1: Down Sample: first bit.

When sampling last value in the frame, a register samples the data input data at the end of the frame. The sampled value is presented for the duration of the next frame.

Figure A3.2: Down Sample: last bit.

Please note that the sampling rate must be an integer greater or equal to 2.

Dual Port RAM

This Xilinx block implements a random access memory (RAM). Dual ports enable simultaneous access to the memory space at different sample rates using multiple data widths. This block was used to implement long delays, as recommended by Xilinx description in *Delay* block.

The location in the memory block can be accessed for reading or writing by providing the valid address on each individual address port. A valid address is an unsigned integer from 0 to $d-1$, where d denotes the RAM depth for the particular port. Each write enable port must be a Boolean value. When the WE port is 1, the value on the data input is written to the location indicated by the address line. We used *Read before write* option, where the output is obtained from the location specified by the address line, and hence is the value of the location being written to.

If enable ports are used, latency must be at least 2.

MCode

This block is a container for executing MATLAB function(s) within Simulink. A parameter on the block specifies the M-function name. The block executes the M-code to calculate block outputs during a Simulink simulation. The same code is translated in a straightforward way into equivalent behavioral VHDL/Verilog when hardware is generated.

The MCode block has the following three primary coding guidelines that must be followed:

- All block inputs and outputs must be of Xilinx fixed-point type.
- The block must have at least one output port.
- The code for the block must exist on the MATLAB path or in the same directory as the directory as the model that uses the block.

For further information on the data types and operations supported, please refer to MCode block description.

Mult

This block implements a multiplier. It computes the product of the data on its two input ports, producing the result on its output port. During the implementation *Use embedded multipliers* option is checked by default. It directs the core to use embedded multipliers (available for parallel multipliers). However, according to Xilinx description this option is not available for Virtex II Pro. Decision whether to use full precision or user-defined precision was made based on what the consecutive blocks were.

Mux

This block implements a multiplexer. The block has one select input (type unsigned) and a user-configurable number of data bus inputs, ranging from 2 to 1024. Throughout the design, 3 input multiplexers were used. Full precision was used for the output in all blocks.

Single Port RAM

This block implements a random access memory (RAM) with one data input and one data output port. Values in a Single Port RAM are stored by word, and all words have the same arithmetic type, width, and binary point position.

A single-port RAM can be implemented using either block memory or distributed memory resources in the FPGA. Block memory was used throughout the design. Each data word is associated with exactly one address that must be an unsigned integer in the range 0 to $d-1$, where

d denotes the RAM depth (number of words in the RAM). The initial RAM contents can be specified through the block parameters.

The write enable signal must be *Bool*, and when its value is 1, the data input is written to the memory location indicated by the address input. When the WE is 0, the output port has the value at the location specified by the address line.

No read On write option was chosen throughout the design which means the output value remains unchanged irrespective of change of address or state of the memory. This option was chosen due to the fact that in most cases the data had to be rearranged, and it was not beneficial to have the data output in the original order.

Up Sample

This block increases the sample rate at the point where the block is placed in the design. The output sample period is l/n , where l is the input sample period and n is the sampling rate.

In hardware, the Up Sample block has two possible implementations. If the *Copy Samples* option is selected on the block parameters dialog box, the *Din* port is connected directly to *Dout* and no hardware is expended. However, if zero padding is selected, a *mux* is used to switch between the input sample and inserted zeros. The corresponding circuit for the zero padding Up Sample block is shown below. Zero-padding was not used in this design as with zero-adding input signal is output once with $(n-1)$ zeroes interspersed if zero padding is used. It could have been used, but it would have no benefit for this particular design and it would take additional hardware required to implement *mux* block.

Figure A3.3: Up Sample block (zero-padding enabled).

A4 Selected MCode functions

anotb

```
function z = anotb(a,b)
% Output is boolean '1' if input A is '1' and input B is '0'
if (a == xfix({xlBoolean}, 1)) && (b ~= xfix({xlBoolean}, 1))
    z = xfix({xlBoolean}, 1);
else
    z = xfix({xlBoolean}, 0);
end
```

atruebout

```
function z = atruebout(a,b)
% The input B is chosen as the output of this function if input A is '1'
if (a == xfix({xlBoolean}, 1))
    z = b;
else
    z = 0;
end
```

bool_check

```
function z = bool_check(a, b)
% This function produces output (Boolean) = '1'
% as long as either input is not '0'
if (a > 0 || b > 0 || a < 0 || b < 0)
    z = xfix({xlBoolean}, 1);
else
    z = xfix({xlBoolean}, 0);
end
```

boolean_not

```
function z = boolean_not (x)
% This function resets the counters as soon as
% the counters are disabled and keeps the reset on
% until the counter enable flag is issued
if (x == xfix({xlBoolean}, 1))
    z = xfix({xlBoolean}, 0);
else
    z = xfix({xlBoolean}, 1);
end
```

counter32

```
function z = counter32(a)
% Output is Boolean '1' once the input is '32' or greater
if (a < 32)
```

```

    z = xfix({xlBoolean}, 0);
else
    z = xfix({xlBoolean}, 1);
end

```

dynamic_range

```

function z = dynamic_range(x, a, b)
% The output is equal to the input if it falls between '0.3' and '3.6'
% The output is equal to '0.3' if the input is less than '0.3'
% The output is equal to '3.6' if the input is greater than '3.6'
% The input has maximum 10 decimal places, with the rest of the numbers (if
% any) being truncated
    if ((xfix({xlSigned,58, 51},x) < xfix({xlSigned,58, 51},3.6)) &&
(xfix({xlSigned,58, 51},x) > xfix({xlSigned,58, 51},0.3))) && ((a ==
xfix({xlBoolean}, 1))&&(b ~= xfix({xlBoolean}, 1)))
        z = xfix({xlSigned,58, 51},x);
    else
        if (xfix({xlSigned,58, 51},x) > xfix({xlSigned,58, 51},3.6)) && ((a ==
xfix({xlBoolean}, 1))&&(b ~= xfix({xlBoolean}, 1)))
            z = xfix({xlSigned,58, 51},3.6);
        else
            if (xfix({xlSigned,58, 51},x) < xfix({xlSigned,58, 51},0.3)) && ((a ==
xfix({xlBoolean}, 1))&&(b ~= xfix({xlBoolean}, 1)))
                z = xfix({xlSigned,58, 51},0.3);
            else
                z = xfix({xlSigned,58, 51},0);
            end
        end
    end
end
end

```

enable1

```

function z = enable1(x)
% This function produces output (Boolean) = '1'
% as long as input is >= '1'
    if x >= 1
        z = xfix({xlBoolean}, 1);
    else
        z = xfix({xlBoolean}, 0);
    end
end

```

main_mux_selector

```

function z = main_mux_selector(a, b)
% This function is used for selecting counter inputs to RAM blocks in
% Kasai and Structural blocks
% If input A is '1', select input '0' from Mux block - write
% If input B is '1', select input '1' from Mux block - read
% If none of the above is true, select input '2' from Mux block -
% dummy input
    if a == xfix({xlBoolean}, 1)
        z = 0;
    else

```



```

        if b == xfix({xlBoolean}, 1)
            z = 1;
        else
            z = 2;
        end
    end
end

```

mux_selector_RAM

```

function z = mux_selector_RAM(x,y, y2)
% This function produces output = '0' if input x is == Boolean '1', meaning
% the first input has been detected and the data is written in
% output = '1' if input y is >=1 and less than 65 (size of the input array)
% and y2 is not '0'
% output == '2' if none of the above is true, meaning
% data is neither written it nor read out
if x == xfix({xlBoolean}, 1)
    z = 0;
else
    if (y >= 1 && y <65)&&(y2~=0)
        z = 1;
    else
        z = 2;
    end
end
end

```

reset64

```

function z = reset64(x)
% This function produces output (Boolean) = '1' when the input is '64',
% meaning the end of demo array has been reached and the reset to
% the counters responsible for writing in/reading out must be reset
% and the mux selector choice needs to be changed as well
if x == 64
    z = xfix({xlBoolean}, 1);
else
    z = xfix({xlBoolean}, 0);
end

```

counter.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter is
  Port (clk:in STD_LOGIC;
        cnt:out STD_LOGIC_VECTOR (6 downto 0)
        );
end counter;

architecture Behavioral of counter is
  signal counter: std_logic_vector(6 downto 0):=(others=>'0');
begin
  process (clk)
  begin
    if (clk'event and clk='1') then
      counter<=counter+1;
    end if;
  end process;
  cnt<=counter;
end Behavioral;
```

A5 Sample Simulation Calculations

Conversion of decimal places to binary as shown in [39]. The number used for the example is

0.4.

First 0 is used to represent '0' in unsigned notation. The rest of the numbers are represented as follows:

$$0.4 \times 2 = 0.8 - 0$$

$$0.8 \times 2 = 1.6 - 1$$

$$0.6 \times 2 = 1.2 - 1$$

$$0.2 \times 2 = 0.4 - 0$$

$$0.4 \times 2 = 0.8 - 0$$

$$0.8 \times 2 = 1.6 - 1$$

Hence, 0.4 in binary is 001100110.

Sampling Block

$$1^{\text{st}}. '00000000000000001' = 0(2^{10}) + 1(2^9) + 0(2^8) + 0(2^7) + 0(2^6) + 0(2^5) + 0(2^4) + 0(2^3) + 0(2^2) + 0(2^1) + 0(2^0) + 1(2^{-1}) = 0.5$$

RIRO Block

$$1^{\text{st}}. '00000000000000001' = 0(2^{10}) + 1(2^9) + 0(2^8) + 0(2^7) + 0(2^6) + 0(2^5) + 0(2^4) + 0(2^3) + 0(2^2) + 0(2^1) + 0(2^0) + 1(2^{-1}) = 0.5$$

Biasing Block

Ibias

$$1^{\text{st}}. '1111111111110000' = -1(2^{11}) + 1(2^{10}) + 1(2^9) + 1(2^8) + 1(2^7) + 1(2^6) + 1(2^5) + 1(2^4) + 1(2^3) + 1(2^2) + 1(2^1) + 1(2^0) + 0(2^{-1}) + 0(2^{-2}) + 0(2^{-3}) + 0(2^{-4}) = -1$$

Qbias

$$1^{\text{st}}. '1111111110011110' = -1(2^{11}) + 1(2^{10}) + 1(2^9) + 1(2^8) + 1(2^7) + 1(2^6) + 1(2^5) + 1(2^4) + 1(2^3) + 0(2^2) + 0(2^1) + 1(2^0) + 1(2^{-1}) + 1(2^{-2}) + 1(2^{-3}) + 0(2^{-4}) = -7 + 0.875 = -6.125$$

Structural Block

S2

$$1^{\text{st}}. '00000011110110000111110101000100010110010001110000-0011-100' = 0(2^6) + 0(2^5) + 0(2^4) + 0(2^3) + 0(2^2) + 0(2^1) + 1(2^0) + 1(2^{-1}) + 1(2^{-2}) + 1(2^{-3}) + 0(2^{-4}) + 1(2^{-5}) + 1(2^{-6}) + 0(2^{-7}) + 0(2^{-8}) + 0(2^{-9}) + 0(2^{-10}) + 1(2^{-11}) + 1(2^{-12}) + 1(2^{-13}) + 1(2^{-14}) + 1(2^{-15}) + 1(2^{-16}) + 0(2^{-17}) + 1(2^{-18}) + 0(2^{-19}) + 1(2^{-20}) + 0(2^{-21}) + 0(2^{-22}) + 0(2^{-23}) + 1(2^{-24}) + 0(2^{-25}) + 0(2^{-26}) + 0(2^{-27}) + 1(2^{-28}) + 0(2^{-29}) + 1(2^{-30}) + 1(2^{-31}) + 0(2^{-32}) + 0(2^{-33}) + 1(2^{-34}) + 0(2^{-35}) + 0(2^{-36}) + 0(2^{-37}) + 1(2^{-38}) + 1(2^{-39}) + 1(2^{-40}) + 0(2^{-41}) + 0(2^{-42}) + 0(2^{-43}) + 0(2^{-44}) + 0(2^{-45}) + 0(2^{-46}) + 1(2^{-47}) + 1(2^{-48}) + 1(2^{-49}) + 0(2^{-50}) + 1(2^{-51}) = 1.92284113687402502534951054258272$$

Kasai atan out

VV Block

 VV_{out}

83

④ 1674-50