2004

Zooplankton visualization system: design and real-time lossless image compression

Dattatreya Reddy Tetala Satya Surya
Louisiana State University and Agricultural and Mechanical College

ZOOPLANKTON VISUALIZATION SYSTEM: DESIGN AND REAL-TIME LOSSLESS IMAGE COMPRESSION

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

in

The Department of Electrical and Computer Engineering

by
Dattatreya Reddy, Tetala Satya Surya
B. Tech., Jawaharlal Nehru Technological University, 2001
August 2004

ACKNOWLEDGMENTS

First and foremost, I express my sincere appreciation and thanks to my advisor and major professor, Dr. Jerry Trahan, for his constant guidance and valuable comments, without whom, this thesis would not have been successfully completed. I sincerely thank Dr. Mark Benfield for giving me the opportunity to work with him on his project. My gratitude also goes out to Dr. Bahadir Gunturk for his timely advises and Dr. Ramachandran Vaidyanathan for serving in my defense committee.

I thank the members of my family, especially my sister, Mani Manjusha Tetala, also a graduate from LSU, for always being there to inspire, encourage, and support me, and without whom the thesis would have been a much difficult task. Last but certainly not the least, I thank my friends here at LSU, whose company and friendship I dearly cherish. Thank you for making my stay here at LSU a memorable one.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

Abbreviation	Meaning
A	Ampere
AC	Alternating Current
Ah	Ampere per Hour
ASIC	Application Specific Integrated Circuits
AT	Advanced Technology
ATA	Advanced Technology Attachment
BIOS	Basic Input/Output System
CCD	Charge Coupled Device
CF	Compact Flash
CLB	Control Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CTD	Conductivity-Temperature-Depth
dB	Decibel
DC	Direct Current
DMA	Direct Memory Access
EIDE	Enhanced Integrated Drive Electronics
FIFO	First in, first out
FPGA	Field Programmable Gate Array
fps	Frame per second
FSB	Front Side Bus
Gb	1,000,000,000 bits
GB	1,000,000,000 bytes
GCLK	Global clock
HDD	Hard Disk Drive
Hz	Hertz
IOB	Input/Output Block
ISA	Industry Standard Architecture
KB	1,000 bytes
KHz	Kilohertz
L	Liter
L/min	Liters per minute
LAN	Local Area Network
LIP	List of Insignificant Pixels
LIS	List of Insignificant Sets
LSB	Least Significant Bit
LSP	List of Significant Pixels
LUT	Look up tables
LVDS	Low Voltage Differential Signaling
M	Meter

MB	1,000,000 bytes
Mbps	1,000,000 bits per second
MBps	1,000,000 bytes per second
MPixels/sec	1,000,000 pixels per second
MHz	Megahertz
ml	Milliliter
ml/sec	Milliliter per second
mm	Millimeter
MMU	Memory Management Unit
ms	Millisecond
MSB	Most Significant Bit
MSE	Mean Square Error
µsec	Microsecond
nm	Nanometers
OS	Operating System
PCI	Peripheral Component Interconnect
PSNR	Peak Signal to Noise Ratio
ROI	Region of Interest
rpm	Rotations per minute
RTOS	Real Time Operating System
SCSI	Small Computers System Interface
SDRAM	Synchronous Dynamic Random Access Memory
sec	Second
SIMD	Single Instruction Multiple Data
S.M.A.R.T	Self-Monitoring, Analysis, and Reporting Technology
SMBus	System Management Bus
S+P	Sequential and Predictive
SPIHT	Set Partitioning in Hierarchical Trees
TAPS	Tracor Acoustic Profiling System
TBUF	Tri-state buffers
TIFF	Tagged Image File Format
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
UVP	Underwater Video Profiler
V	Voltage
VLSI	Very Large Scale Integration
VPR	Video Plankton Recorder
W	Wattage
ZOOVIS	Zooplankton Visualization System
ZOOVIS-SC	Zooplankton Visualization System-Self Contained

ABSTRACT

In this thesis, I present a design of a small, self-contained, underwater plankton imaging system. I base the imaging system's design on an embedded PC architecture based on PC/104-*Plus* standards to meet the compact size and low power requirements. I developed a simple graphical user interface to run on a real-time operating system to control the imaging system.

I also address how a real-time image compression scheme implemented on an FPGA chip speeds up image transfer speeds of the imaging system. Since lossless compression of the image is required in order to retain all image details, I began with an established compression scheme like SPIHT, and latter proposed a new compression scheme that suits the imaging system's requirements. I provide an estimate of the total amount of resources required and propose suitable FPGA chips to implement the compression scheme. Finally, I present various parallel designs by which the FPGA chip can be integrated into the imaging system.

1

INTRODUCTION

Underwater imaging systems are an important part of oceanographic research. The oceans contain many biological, biogenic, and physical phenomena distributed on fine spatial scales [1], and these imaging systems provide a means to study them. Researchers use some of these systems to study plankton, one of the primary links in marine food chains.

1.1 Introduction to Problem

Zooplankton[1], small planktonic organisms carried on ocean currents, are the most abundant food source in the oceans for fish and other large organisms such as birds and whales. Additionally, many economically important organisms such as crabs and shrimp progress through a larval stage where they are considered zooplankton. Size classes distinguish different groups of zooplankton. Mesozooplankton are 200 – 2000μm.

Zooplankton are useful indicators of future fisheries health because they are a food source for organisms at higher trophic levels and hence are of interest to oceanographic researchers. Typically, researchers study zooplankton using nets, pumps, and cameras. Imaging systems use cameras for quantitative analysis of zooplankton without harming them. Zooplankton are patchily distributed in the oceans in both the vertical and horizontal domains. They frequently occur in thin vertical layers of centimeters to meters thick. Ideally, it is preferred to collect images of zooplankton from above, below, and within thin layers on vertical and horizontal scales of centimeters without disturbing the layer since the images captured in these positions provide detailed information on the types and sizes of zooplankton. Existing imaging

[1] From the Greek, *zoi*, "animal", and *planktos*, "to wander or drift."

systems, however, are large tethered vehicles and unsuitable for discrete sampling of thin zooplankton layers.

1.2 Motivation

The best alternative to the existing systems is to use a non-invasive, internally-recording imaging system. Since no such system exists currently, Dr. Mark Benfield proposed to build a low cost, lightweight, simple to use, non-invasive, internally-recording imaging system. Based on the existing tethered Zooplankton Visualization System[2] (ZOOVIS) [1], Dr. Benfield named the new proposed imaging system Zooplankton Visualization System - Self Contained (ZOOVIS-SC). Dr. Benfield assigned me the task to design and build a fast acting and compact ZOOVIS-SC system that meets the above standards.

1.3 Challenges

The challenges faced in the design of ZOOVIS-SC were the following:

Hardware – Select various components based on the PC architecture and interface them to achieve the maximum possible frame rate at a high resolution.

Software – Develop a real-time operating system and user interface to control the imaging system.

Electrical – Satisfy internal power supply and battery requirements of the imaging system.

Mechanical – Design and construct a pressurized cylindrical enclosure for the imaging system's components.

This thesis deals with the first three categories, particularly hardware. The mechanical aspect of ZOOVIS-SC design, however, does not come under the scope of the thesis.

1.4 Explanation of Hardware Challenges

PC/104-*Plus*[3] modules based on the PC architecture and with built-in ultra-compact stackable modules suit the ZOOVIS-SC's requirements. Hence, I selected this architecture for

[2] ZOOVIS – Zooplankton Visualization and Imaging System is a high-resolution digital imaging system designed to collect quantitative images of zooplankton from a defined volume of water.
[3] Architecture based on a 120 pin, 32-bit, 33MHz form factor PCI-bus.

ZOOVIS-SC design. The advantage of this architecture is the use of a high speed PCI-bus[4] that provides up to 132MBps bandwidth. The use of the PCI-bus by the different modules at various stages of data transfers, however, strains the bandwidth of the bus and increases latency[5]. This severely affects the maximum achievable frame rate of ZOOVIS-SC and brings it down to 5.32fps, well short of the target of 27fps.

The image data transfer rates of ZOOVIS-SC can be enhanced by various methods including a complete change in the architecture of ZOOVIS-SC. I, however, propose to compress image data to enhance the data transfer rates of ZOOVIS-SC. Since loss in image details is not acceptable, the compression needs to be lossless. Further, the compression requires faster encoding, so that there is no additional delay in the data transfer path. Hence, I propose to implement a lossless image compression scheme in real-time on hardware that computes quickly, and improves the overall latency of the data transfer.

1.5 Contribution

I designed and built a working prototype model of ZOOVIS-SC using PC/104-*Plus* modules. I also developed a real-time XP embedded operating system and a simple graphical user interface in Visual C++ to control the ZOOVIS-SC system. I handled the power supply requirements of ZOOVIS-SC using a power control module with lithium ion battery source.

I implemented a compression scheme involving the Sequential and Predictive (S+P) transform [2] followed by a new proposed 4-Cluster[6] bit-ordering method to enhance the data transfer rates. The fast computing, lossless compression scheme utilizes the properties of Set Partitioning in Hierarchical Trees (SPIHT) [3] in the bit-ordering phase. The S+P transform followed by bit-ordering image compression scheme achieves a compression ratio[7] of 1.67:1 on 1024×1024, 10-bit, grayscale images from ZOOVIS-SC.

I implemented the compression scheme in Verilog, and estimated its hardware resource requirements for a Xilinx FPGA chip. The estimated compression speed is 45.5MPixels/sec.

[4] Peripheral Component Interconnect is an interconnection system between a microprocessor and attached devices in which expansion slots are spaced closely for high speed operation.
[5] Latency refers to the delay in time between the sending of a unit of data at the origin and the reception of that unit at the destination end.
[6] A new proposed algorithm presented in Section 7.2.3.
[7] Compression ratio equals to original image size divided by compressed image size.

The compression schemes fits on an XC2S100E FPGA of the Spartan-IIE series. With compression, ZOOVIS-SC can acquire up to 24.76fps based on a pipelined approach for the implementation of image compression.

To summarize, the thesis deals with the design of ZOOVIS-SC, implementing image compression on an FPGA chip, and interfacing the FPGA chip with ZOOVIS-SC. I successfully designed an internally recording imaging system that can be housed in a cylindrical pressure housing of approximately 6" in diameter and 24" long. I propose a design and a suitable FPGA chip for implementation of lossless image compression comprising the S+P transform and 4-Cluster bit ordering on the FPGA. I also propose various pipelined designs by which the FPGA chip integrates into ZOOVIS-SC to achieve performance of up to 24.76fps.

1.6 Thesis Outline

On the basis of the discussion in the previous sections, I have classified the thesis into three parts: Part I presents the design of ZOOVIS-SC, Part II presents the design, and implementation of real-time lossless image compression, and Part III addresses the approaches for hardware implementation of image compression.

In Part I, Chapter 2 presents background on underwater imaging and spells out the requirements for ZOOVIS-SC. Chapter 3 presents the various design requirements for ZOOVIS-SC in detail and gives technical specifications of the hardware required. Chapter 4 discusses the architecture of ZOOVIS-SC in detail, and Chapter 5 concludes Part I discussing the future work required to improve ZOOVIS-SC.

In Part II, Chapter 6 reviews some relevant image compression algorithms to be used and their hardware implementations, Chapter 7 gives the various modifications applied to an existing algorithm to suit it for ZOOVIS-SC images, and Chapter 8 discusses the architecture for hardware implementation. Finally, Chapter 9 presents the performance results of the lossless compression algorithms.

Part III includes only Chapter 10, which discusses the improvement of ZOOVIS-SC after the implementation of lossless image compression and how it can be further improved.

PART – I

DESIGN OF A SELF CONTAINED ZOOPLANKTON VISUALIZATION SYSTEM

2

UNDERWATER IMAGING SYSTEMS

This chapter provides a brief overview of the need for visual imaging systems by oceanographers. The primary focus of the chapter is ZOOVIS, an imaging system designed specifically for quantifying zooplankton distributions and abundances. Besides its structure and function, I also discuss its limitations and the need for a new system. Sections 2.1 and 2.2 summarize information from Benfield *et al*. [1]. Dr. Mark Benfield devised the idea and oversaw the design and construction of the ZOOVIS and ZOOVIS-SC systems.

2.1 Background

Oceanographers have used nets and pumps to take samples of small marine zooplankton. Nets are useful for quantifying zooplankton distributions and abundances on horizontal scales of tens to hundreds of meters and vertical scales of several meters, however, zooplankton are frequently distributed in non-random patches and layers that nets cannot resolve. Moreover, many zooplankton are fragile gelatinous organisms that are damaged or destroyed by nets and pumps.

Limitations in traditional sampling methods lead to the development of optical systems for quantifying zooplankton distributions and abundances. Optical sensors used in these systems can be divided into particle detection and image-forming systems. Particle detectors such as the optical plankton counter [4] use the interruption of a light source by zooplankton and other objects to detect, count, and measure targets as they pass through a sampling tunnel[8].

[8] A hollow tube open at both ends through which ocean water passes and the volume of water in the tunnel at any given moment is studied.

The drawback is that the particle detector provides no information on the two-dimensional spatial distributions of targets nor their likely taxonomic identities.

Image-forming optics use various types of cameras to image organisms along the tow path[9] of the instrument. Quantitative instruments in this category began with a photographic camera mounted in a net and currently include towed systems[10] such as the camera net system [5], the ichthyoplankton recorder [6], Video Plankton Recorder (VPR) [7], *in situ* video recorder [8], and the Shadowed Image Particle Platform and Evaluation Recorder (SIPPER) [9]. In addition, there are profiling systems, such as the Underwater Video Profiler (UVP) [10] and holographic instruments [11, 12]. See Wiebe and Benfield [13] for a review.

In general, non-holographic image forming optical systems provide a means for estimating the spatial distributions and abundances on vertical scales of centimeters or greater. The majority of optical systems utilizes video and typically image small volumes of water to achieve acceptable image resolution characteristics. Image volume and image resolution are inversely related. The use of conventional video formats (NTSC, PAL, SECAM) means that a limited number of pixels (~500 to 600 horizontal scan lines depending on the format) describe each image [13].

High-resolution digital imagers based on Charge Coupled Device (CCD) or Complementary Metal Oxide Semiconductor (CMOS) sensors providing millions of pixels that allow larger areas to be imaged without sacrificing resolution. An image-forming optical instrument called ZOOVIS images large volumes of water at high resolution using these high resolution image sensors [1]. The use of a high-resolution digital still camera with a CCD containing 4.19 mega pixels allows an increased image volume while maintaining acceptable image resolution. ZOOVIS employs a sensor package to quantify both physical and biological parameters on fine vertical scales to depths of 250m. The next section discusses the design of ZOOVIS. The design of ZOOVIS is relevant because no single sensor can quantify all zooplankton or be suited to all potential sampling conditions. Experience gained with ZOOVIS led to design considerations for ZOOVIS-SC.

[9] The cross section area along which the imaging system is towed.
[10] Imaging systems that are dragged along a determined tow path.

2.2 ZOOVIS

ZOOVIS consists of a digital camera and strobe coupled to environmental sensors, and connected to a surface winch, and control and analysis computer via a electro-optical cable. Images from ZOOVIS are transferred from the underwater components (Figure 2.1) via a fiber-optic/conductive cable to a surface control computer, which stores the images and data. Figure 2.1 shows the components of ZOOVIS. The following sections discuss key components.

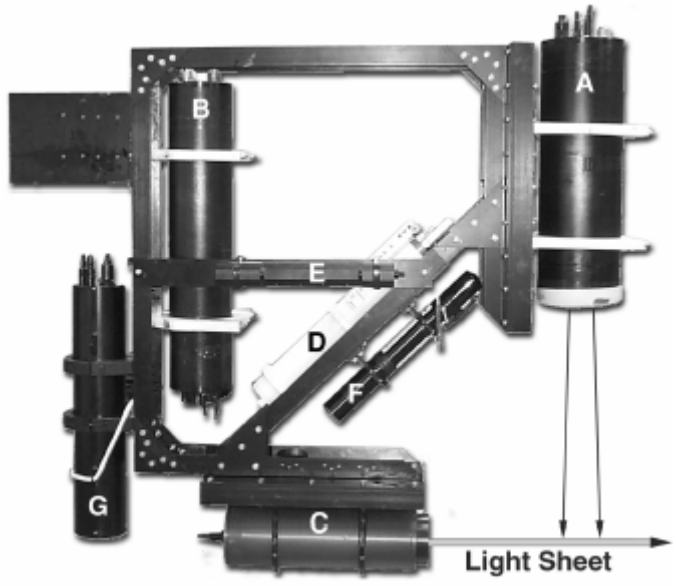


FIGURE 2.1 Underwater components of ZOOVIS: A = camera housing; B = telemetry housing; C = strobe housing; D = CTD; E = transmissometer; and F = acoustic transponder/responder. The horizontal gray line from the strobe illustrates the orientation of the light sheet relative to the field of view of the camera (black arrows). Figure used from [1] with permission of the author.

2.2.1 Camera Housing

The cylindrical pressure vessel containing the camera comprises of the camera housing. A monochrome digital camera inside the camera hosing consists of a CCD sensor capable of collecting 14-bit gray scale images at sampling rates of up to 2.2 Hz [1]. Maximum resolution of the camera is 2048×2048 pixels. A multi-conductor cable supplies power to the camera housing.

2.2.2 Power/Telemetry Housing

The power/telemetry housing consists of a single-board computer, and fiber-optic networking components. The single-board computer has a PCI slot for the camera acquisition card through which the computer sends signals to the camera and captures images. In addition, the computer responds to the commands from a surface PC, and sends data to the surface via a 100Mbps Ethernet network link. The power/telemetry housing also contains power supplies for all components of ZOOVIS.

2.2.3 Strobe Housing

A cylindrical pressure vessel containing the strobe and with a clear optical glass port on one end comprises the strobe housing. The custom built strobe receives a 20μs pulse from the telemetry housing and produces a collimated and relatively flat sheet of light using a linear arc discharge tube and collimating[11] optics [1].

2.2.4 Command and Control

The underwater server application and the surface client application control ZOOVIS [1]. The server takes commands issued by the client and then issues commands to control the camera. The client saves uncompressed 16-bit TIFF image files on the surface hard-drive.

2.3 ZOOVIS-SC

This section discusses the problems faced by existing systems and ZOOVIS in the collection of discrete samples on the scales of thin zooplankton layers and the need for a new imaging system for the study of zooplankton layers.

2.3.1 Problems with Existing Systems

While ZOOVIS provided excellent images of larger zooplankton (>2mm), it could not resolve smaller mesozooplanktons. In addition, existing imaging systems such as the VPR and ZOOVIS are not well suited to work in discrete samples of 2"×2" square area on the scales of thin zooplankton layers. The VPR suffers from a small sample volume[12] and the configuration of its camera and strobe relative to the sampling volume. The layer being imaged must be located between the strobe and the camera, which makes physical disruption

[11] To render parallel to a certain line or direction.
[12] Small sampling volume means that the instrument must remain within the layer for several seconds in order to collect a sufficiently large sample.

of the layer highly likely. Most VPRs are tethered instruments that suffer from limitations imposed by vessel heave. Internal recording VPRs can deploy autonomously, but there is a high probability of them disrupting zooplankton layers.

ZOOVIS is a large tethered vehicle that would likely disrupt the layer with its strobe housing and would be difficult to precisely to hold it in a layer of a few centimeters. Holographic systems can record the contents of liters of water at high resolution; such systems are large, however and the time lag required to process the data makes near-real time examination of the data unlikely. In summary, none of the existing optical imagers are capable of quantifying zooplankton within thin layers.

2.3.2 Proposed New System

Ideally, it is preferred to collect physical samples of zooplankton from above, below, and within thin layers on vertical and horizontal scales of centimeters without disturbing the layer. The best alternative to the existing systems is to use a non-invasive, internally-recording, imaging system that is co-registered with a Tracor Acoustic Profiling System (TAPS) mounted on a high-resolution platform (e.g., slow-drop package). Also the system should be low cost (under $20,000), lightweight, and simple to use. Regrettably, no such system currently exists.

ZOOVIS-SC is a self-contained variant of the original tethered ZOOVIS system designed to fill in the above mentioned gaps. Self-contained implies that there is no need for any real-time link from any external source to the system either for control or other requirements. Unlike ZOOVIS where each component is housed separately and held together by a frame, a single pressure housing contains all ZOOVIS-SC components, excluding the light-source, optics, camera, computer, hard drive, and batteries. ZOOVIS-SC is a single piece of equipment without any on-field assembly requirements and therefore very easy to handle and use.

ZOOVIS-SC is designed simple to use. All a user will have to do is plug in a monitor, keyboard, and mouse into the back of the pressure housing. Once connected, direct communication with the inside computer via an intuitive software interface will allow the system to be configured for start, duration, rate, and termination of image acquisition. At the

end of each deployment of 30 - 60 min, the user can rapidly transfer the contents of the system to a surface hard drive via the disk drive interface on the back of the pressure housing.

ZOOVIS-SC is an optical counterpart to the TAPS acoustic system [14]. TAPS ensonifies a volume located 1.5m in front of the transducers, and ZOOVIS-SC images the contents of a smaller volume that would be co-located with the TAPS sample volume. Thus, the use of both systems will provide both an acoustic and optical record of the contents of water well ahead of the platform carrying the instruments, thus eliminating the danger of physical disruption of the layer and its constituent particles.

2.3.3 Target Specifications Summary of ZOOVIS-SC

- High resolution, fast acting camera.
- High frame rate and fast data transfers.
- Inherent ruggedness and high reliability.
- Low cost.
- Light weight and ease of mobility and usage.
- System should be capable of operating up to 30 – 60 minutes per deployment.

3

DESIGN OF ZOOVIS-SC

Sections 2.3.2 and 2.3.3 discussed the requirements and specifications for the proposed new system ZOOVIS-SC. In accordance with those requirements, this chapter explains the proposed architecture for ZOOVIS-SC in Section 3.1. Section 3.2 discusses the various hardware requirements and technical details, and Section 3.3 discusses the operating system requirements.

3.1 Proposed Architecture – PC/104-*Plus*

Over the past decade, the PC architecture has become an accepted platform for far more than desktop applications. Dedicated and embedded applications for PCs have increased. PC/104, developed in response to this need, has a range of features: reduced space and power constraints, full hardware and software compatibility with the PC bus, and built in ultra-compact stackable modules. This range of features in PC/104 suits the unique requirements of embedded control applications like ZOOVIS-SC. The PC/104-*Plus* specification based on the PC/104 architecture establishes a standard for the use of a high speed PCI-bus in embedded applications [15].

Figure 3.1 shows the ultra-compact stackable module feature of the PC/104-*Plus* architecture.

3.1.1 Advantages of PC/104-*Plus* Architecture

The PC/104-*Plus* architecture as shown in Figure 3.1 features self-stacking modules with 104-pin ISA and 120-pin PCI bus connectors that allow multiple modules to be added to the system without the burden of backplanes and cartridges. Maximum data throughput on the PCI-bus is 132MBps at 33MHz clock speed. PC/104-*Plus* technology is compatible with

PC/104 and supports 32-bit PCI interconnect. These modules are small[13], require low voltage[14] and power, and generate little heat. The pin-and-socket bus connector and four-corner mounting holes provide resistance to shock and vibration and are very reliable in harsh environments. This is a summary of the advantages of the PC/104-*Plus* architecture presented by the PC/104 Embedded Consortium [15].

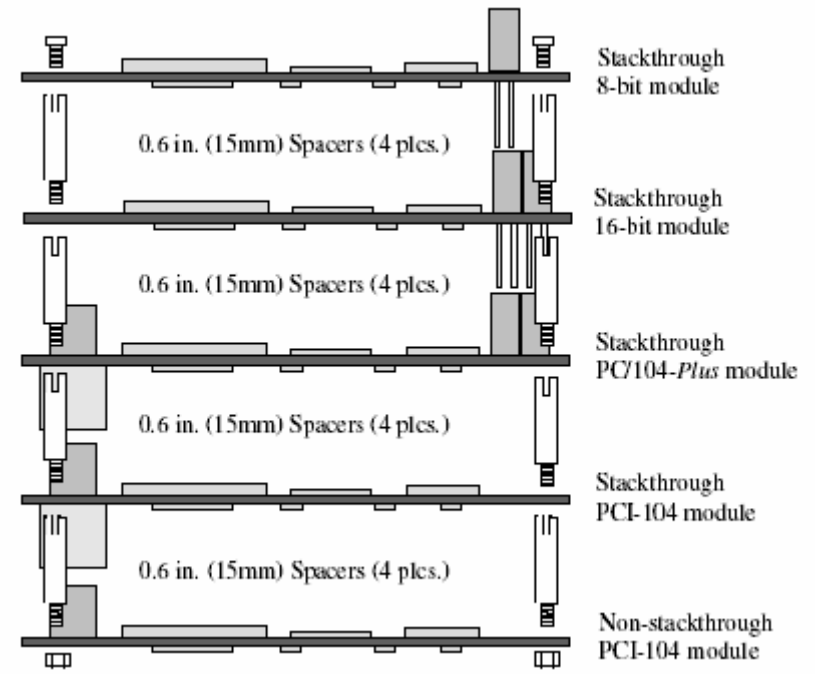


FIGURE 3.1 A typical stack with PCI-104, PC/104, and PC/104-*Plus* modules. The maximum number of modules on the PCI-bus is four plus the host board [15].

3.1.2 Applications of PC/104-*Plus* Technology

This technology suits applications with requirements of high performance single-board computers, full-motion video interfaces, high performance data acquisition, and control interfaces. Communications interface modules such as high speed LANs, 100BaseT, USB, IEEE-1394[15], etc. are also available. This technology also finds use in communications devices, medical instruments, image processing, data loggers, vending machines, test equipment, vehicular systems, industrial control systems, and PCI-bus adapters and bridges.

[13] Specified size is 3.6×3.8 inches.
[14] Typical voltage requirement is 3.3V or 5V DC.
[15] A very fast external bus standard that supports data transfer rates of up to 400Mbps (in 1394a) and 800Mbps (in 1394b).

Based on the above discussion and the requirements summarized in Section 2.3.3, the PC/104-*Plus* based architecture well suits the proposed new system. So, I decided to build ZOOVIS-SC on the PC/104-*Plus* architecture.

3.2 Modules Used in ZOOVIS-SC

The important component for a visual imaging system is the camera/framegrabber combination. For ZOOVIS-SC, this combination should be capable of capturing images at high resolutions of 1024×1024 or 2048×2048 10, 12, or 14-bit images and at high frame rates of 10-15fps and more. This combination requires a strobe unit capable of significantly illuminating the field of view or region of interest (ROI). The cameralink standard seems a viable solution for the interface between the camera and the framegrabber, as it is extremely flexible and is specifically designed for high-speed digital imaging. The other required supporting modules are a CPU to run the system, storage media to store captured images, and a power supply unit.

The following sections discuss the different modules selected to be used in the ZOOVIS-SC system. In each section, the first paragraph gives a summary of the required features for each of the corresponding modules, and the rest of the paragraphs discuss the technical details of the selected hardware and how it fits within the required parameters. Most of the technical details presented in the sections below derive from the respective hardware's technical specification sheet [16, 17, 18, 19, 20] and vendor's website.

3.2.1 Digital Camera (Dalsa 1M28-SA)

The ZOOVIS-SC system requires a small, low cost, high speed, robust camera. Dalsa's 1M28-SA seemed a suitable option. The 1M28-SA is an area scan camera for industrial machine vision and traffic management [16].

The 1M28-SA camera uses a one-mega pixel (1024×1024), CMOS image sensor capable of running at up to 27fps at full resolution. Unlike regular CMOS imagers, this camera features an electronic global non-rolling shutter for "Stop Action" imaging, allowing for smear-free capture of fast moving objects. Its Linear-Logarithmic (LinLog) response allows for up to

120dB[16] of intrascene dynamic range, allowing for better images in low light. The Cameralink MDR26 connector allows access to programmable features and diagnostics. It allows for selectable 8 or 10-bit digitization. The camera's small body and robustness make it perfect for the wear and tear of industrial environments [16]. Further, its spectral responsitivity is well within the visible spectrum.

3.2.2 Acquisition Module (Picasso 104-CL)

The acquisition module, also referred to as the framegrabber, should be compatible with the Dalsa camera, that is, have a digital cameralink base-16 interface, support 8 and 10-bit data input formats, provide sampling rates of at least 28.375MHz, and comply with the PC/104-*Plus* standards. Picasso 104-CL from Arvoo seemed a viable option.

The Picasso framegrabber is a high performance 'plug and play' PC-card for the PCI-bus and provides high-resolution image capture for digital video cameras. It enables each standard PCI system to capture and store single images for image processing or full frame display of real time digital video in a window. This model operates as a PCI-bus master, and it transfers images directly to the system memory without impacting the processor [17].

Additional features include an MMU supporting virtual memory up to 4MB per DMA channel (paged-memory), programmable exposure from 6.375µsec to 419ms, asynchronous serial communication to camera and framegrabber via onboard UART or main board COM-port, and two optically isolated general purpose inputs and outputs.

3.2.3 Central Processing Unit (Lippert Cool RoadRunner III)

The requirements for the CPU based on the camera/framegrabber combination are: it should meet the minimum requirements of camera and framegrabber, have sufficient main and cache memory, sustain high disk drive speeds for fast data transfer to storage media, and comply with PC/104-*Plus* standards. The secondary requirement is that the CPU should allow a different drive, in addition to the primary storage medium on which the operating system would run.

[16]A unit for measuring the relative strength of a signal. Usually expressed as the logarithmic ratio of the strength of a transmitted signal to the strength of the original signal.

The Cool RoadRunner III from Lippert is an all-in-one CPU module conforming to all the above requirements. It comes with AGP4x graphics, 10/100BaseT Ethernet and AC-97 sound on-board, a Compact Flash socket, and an ATA-6-compliant EIDE interface. The system's main memory is expandable up to 512MB SDRAM. An Intel ULV Celeron microprocessor forms the core of the board running at 400MHz and supporting MMX technology[17] and streaming of SIMD extensions. The VIA TwisterT Chipset provides the infrastructure of the CPU module, integrating a Savage4 graphics accelerator from S3. With up to 32MB of graphics memory, the graphics accelerator supports resolutions as high as 1600×1200 at 64K colors. In addition, the board also provides a 2-channel LVDS port, a TV-Out port, a serial and parallel port, two USB host ports, and one IrDA compliant infrared interface.

3.2.4 Storage Media

As discussed in Section 3.3.1, I require two drives, one for running the operating system and the second one for storage of data. The primary drive should comply with the Compact flash socket on the CPU board and the secondary with an ATA-6-compliant EIDE interface.

The best CF format drive available is the 1GB MicroDrive from IBM. It supports CF-II format and uses HDD technology to store information. It has data transfer rates of 13.3MBps with a seek time of 12ms and buffer size of 128KB, so it is ideally suited to run the operating system of ZOOVIS-SC.

Among the best ATA-6 standard HDD's available, the choice for data storage is Travelstar E7K60 from Hitachi. It is an enhanced-availability, 9.5mm high, 7200RPM, 2.5-inch mobile HDD designed for continuous and mission-critical applications. The Travelstar E7K60 pushes performance, capacity, power management, and exceptional quietness. This 60GB drive also supports S.M.A.R.T, which alerts the system of any negative reliability status conditions. The other advantage of this hard disk drive is its "always-on" characteristic. Because of this the drive never has to wake up from sleep mode [19], so it can be accessed more quickly, yielding improved transactional time. All these capabilities give this hard disk drive an edge over the other existing storage solutions.

[17] A set of 57 multimedia instructions built into Intel's microprocessors and other x86-compatible microprocessors.

Other technologies that were in consideration for data storage were SCSI and Firewire. Even though they provide better capabilities, I did not consider them because they required additional hardware to interface with the CPU.

3.2.5 Strobe Unit

The strobe needs to illuminate the volume of water being imaged. The volume imaged depends on the typical dimensions of the organisms being studied. Small image volumes resolve the features of individual zooplankton targets well. Therefore, the required strobe should be focused and intense enough to illuminate these small image volumes. It should have at least 27Hz pulse rate to be synchronized with the camera and should be capable to pulse from a trigger from the framegrabber.

I picked a Miniature Direct Illumination Machine Vision Strobe MVS-4000 from Perkin-Elmer Optoelectronics. It has a spectral bandwidth of 300 to 1100nm, 0.14 lumen-sec/ft^2 photometric light output at 2 feet, and has a flash rate of 50Hz maximum with 6μsec pulse duration. Further, its power requirements are well within the range of the system requirements.

3.2.6 Power Supply Unit

According to PC/104-*Plus* specifications, the module stack requires 3.3V or 5V DC power supply for operation, and there are dedicated pins assigned for power and ground in the PCI and ISA buses. A logical solution to provide DC supply to the module stack is to have a DC supply board that is based on PC/104-*Plus* or PC/104 specifications and can supply power through either of the buses. The other requirement for the power supply module is that it should charge batteries from a direct AC or DC supply source and also inform the CPU module of the battery's charge level.

HESC-104 [20] from Tri-M Engineering is a DC to DC 60W converter and is for low noise, embedded PC/104 computer systems. It takes a wide input range of 6 to 40V DC and is ideal for battery or unregulated input applications. The output voltages are 5V and 12V standard and -5V and -12V optional. The related power supply pins on the PC/104 expansion bus provide the DC voltage, and are also available for off-board use through a screw terminal

block. All the stack modules with the PC/104 bus can use the clean and filtered power in the PC/104, bus and the off-board connector can supply power to the camera and strobe unit.

HESC-104 also includes a flash-based microcontroller that supplies advanced power management and a smart battery charger. It provides up to four stages of battery charging and is SMBus level 3 compatible. Battery charging voltages are from 9.5 to 35V and charging currents are up to 4A. These capabilities suit power supply module requirements. Therefore, I selected HESC-104 as the power supply module.

3.3 Operating System

The ZOOVIS-SC system bases on the PC architecture, and it requires an Operating System (OS) to run. OS' classifies into two types based on usage. They are general OS like Windows XP, OS X, Linux, etc and specific OS' like Real Time Operating System (RTOS) that suit specific user applications like ZOOVIS-SC. Since ZOOVIS-SC does not require all the features presented in the full versions, RTOS is preferable.

The platform defines a second classification of operating systems. Commonly used platforms in PCs are Windows and Linux. Free and propriety versions of RTOS available in the market bases on a Linux platform, but most hardware components does not support Linux. The alternative platform is Windows. Windows CE .NET and Windows XP Embedded are the two versions of Windows embedded operating systems. XP embedded allows for user configuration and selection of components that helps to develop the operating system with only the required components. This gives an edge over CE, so I choose XP embedded as the operating system for ZOOVIS-SC.

4

ARCHITECTURE OF ZOOVIS-SC

This chapter describes the architecture of PC/104 modules introduced in Chapter 3 and gives a detailed description of how these modules are interfaced to form the ZOOVIS-SC system. It also discusses power constraints and software control.

4.1 Interfacing Requirements

The ZOOVIS-SC system partitions into three main interfacing categories: the camera - framegrabber - strobe interface, framegrabber - main memory interface, and main memory - disk drive interface. Description of the modules centers mainly on the features required in ZOOVIS-SC, while neglecting the features that are irrelevant for the system. Figure 4.1 shows the basic outline of ZOOVIS-SC with the various modules discussed in Chapter 3.

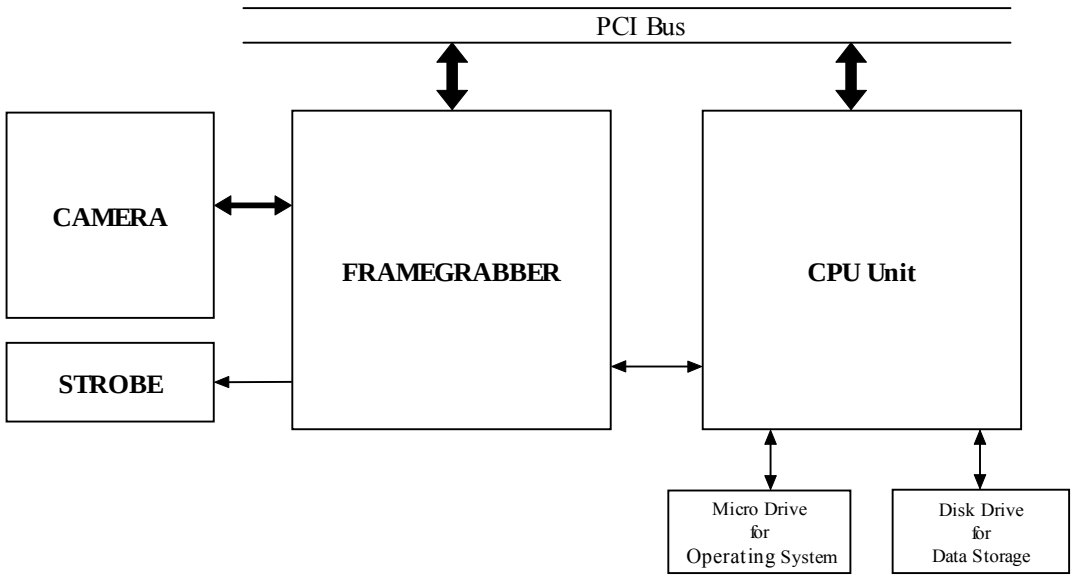


FIGURE 4.1 Basic outline of ZOOVIS-SC components.

4.1.1 Camera - Framegrabber - Strobe Interface

The Dalsa 1M28 camera has two basic modes of operation. One is free-running mode in which continuous exposures are taken without any intervention from the framegrabber, and the other is external sync mode in which an exposure is taken on the positive edge of exsync, an external synchronous signal on CC1 of cameralink. ZOOVIS-SC uses external sync mode. The external sync signal gives control of the exposure trigger and also makes it feasible to synchronize the strobe with the exposure trigger.

The strobe MVS-4000 triggers for a duration of 6μsec on the positive edge trigger. For synchronized triggering of strobe and camera, the CC1 pin from the cameralink connector on the framegrabber taps out and connects to the strobe. Since the pulse rate of the strobe is greater than that of the camera, the issue of missed flash does not arise.

The framegrabber is the core of the system. It links the camera and CPU modules. Figure 4.2 sketches the framegrabber module.

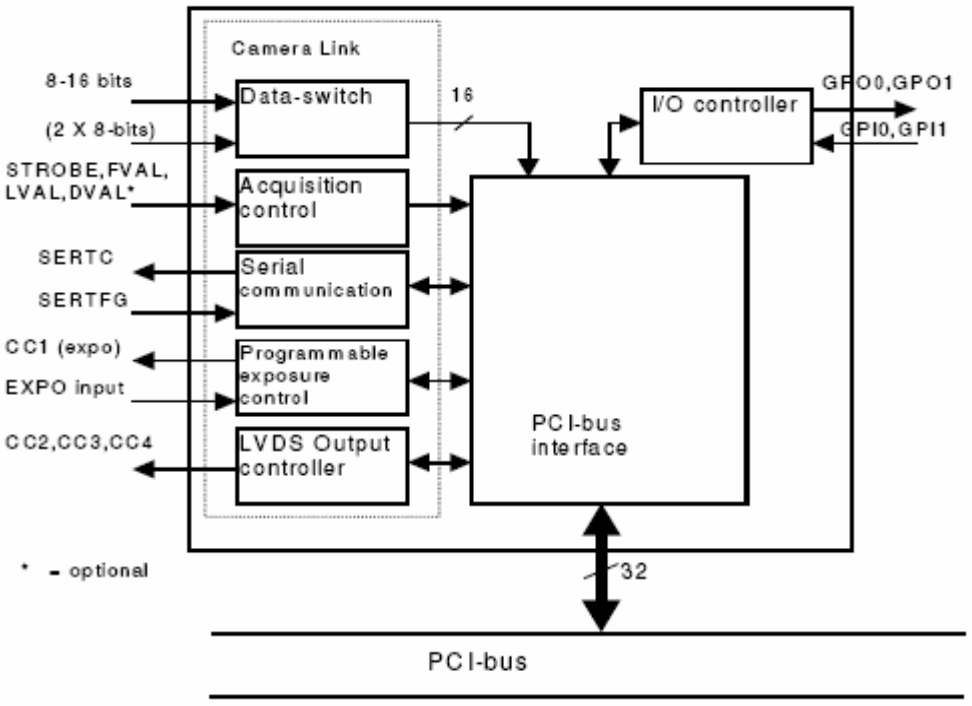


FIGURE 4.2 Functional overview of Picasso 104-CL, the framegrabber module [17].

There are four I/O interfaces for the framegrabber card. They are the PCI 2.2, 120 pin PC/104-*Plus* form factor bus, MDR-26 pin cameralink connector, COM port, and general purpose I/O pins. The framegrabber communicates with the CPU module through the PCI-bus and with the camera through cameralink. The framegrabber acts as a bridge when a null-modem is connected between the COM ports of the framegrabber and CPU modules. This facilitates control of camera settings by software running on the CPU module.

The general purpose I/O pins help transmit signals to and from the framegrabber module. These pins can input an external sync trigger from an external source to the CC1 pin of the cameralink connector on the framegrabber and output a trigger to the strobe. The delay in actually sending or receiving signals on these pins is approximately 90μsec, so ZOOVIS-SC does not use them. Software generated signals trigger the camera and strobe.

4.1.2 Framegrabber – Main Memory Interface

In the functional overview of the framegrabber in Section 4.1.1, one bus that lacked attention was the PCI-bus because it has no role in the camera – framegrabber interfacing. When it comes to the CPU module, however, the PCI-bus is the main interface.

The framegrabber does not have any on-board memory, so the system must immediately transfer the captured data arriving from the camera to an allocated buffer in the main memory. The framegrabber converts the serial data from cameralink to PCI standards and puts that data on the PCI-bus. The framegrabber takes control of the PCI-bus when it is required to transfer data, transfers data to host memory, and then releases the bus.

4.1.3 Main Memory – Disk Drive interface

The interface between main memory and HDD depends on the architecture of the CPU module.

As shown in Figure 4.3, two bridges, the Northbridge and the Southbridge, collect interfaces among the various I/Os and on-board components. The Northbridge connects to processor, SDRAM, and display devices, and the Southbridge connects the rest of the available I/O ports on the module. The interface between Northbridge and Southbridge components is through the PCI-bus.

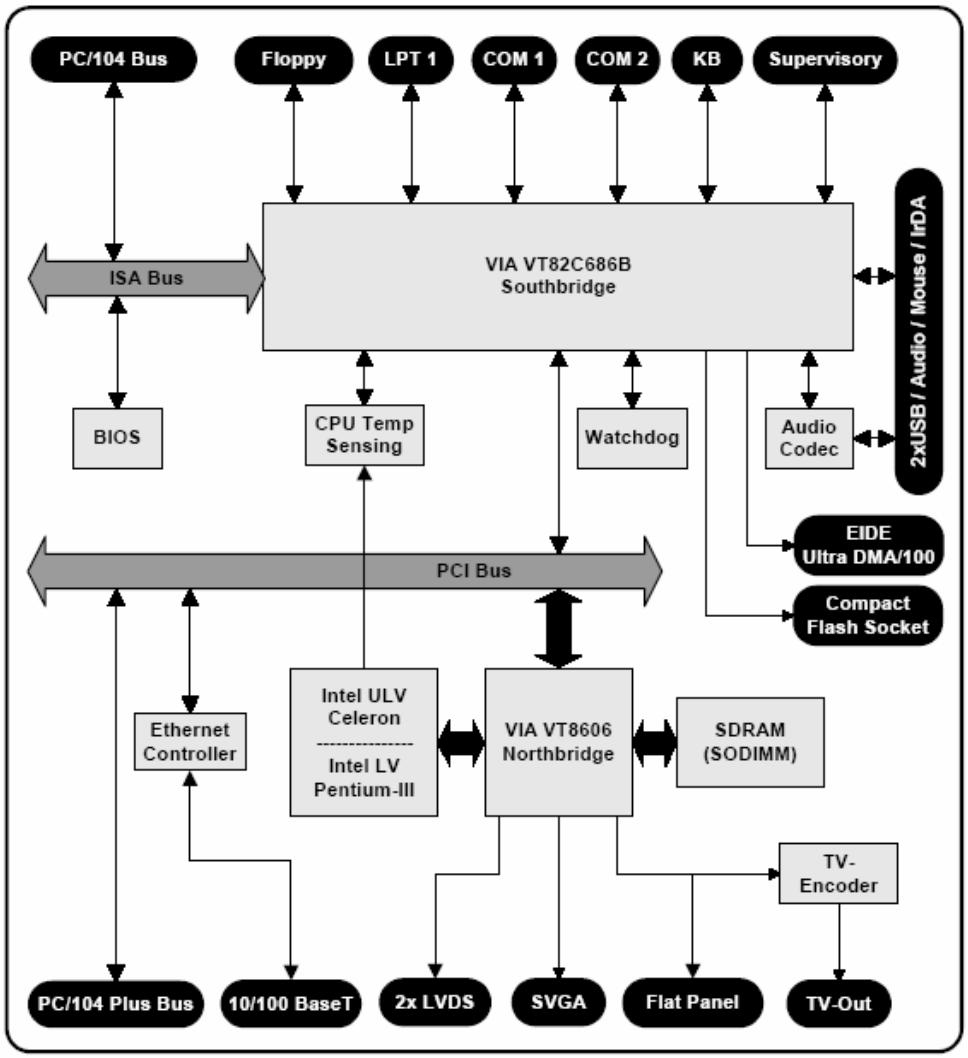


FIGURE 4.3 Complete functional block diagram of the Cool Roadrunner III [18].

The main memory interfaces to the HDD through the Northbridge, PCI-bus, and Southbridge. The Northbridge connects to the main memory through a 64-bit, 115MHz FSB[18] which in turn connects to the PCI-bus through a 32-bit wide bus. The Northbridge chipset contains a built-in bus-to-bus bridge to allow simultaneous concurrent operations on each bus to processor and memory. The Southbridge chipset is a "PCI Super-I/O Integrated Peripheral Controller" [18] and connects to the PCI-bus through a 32-bit wide bus. It is a master mode enhanced IDE controller with dual channel DMA engine and interlaced dual channel commands [18].

[18] Front Side Bus is the data channel connecting the processor, chipset, DRAM, and AGP socket.

To summarize, cameralink is the interface between the camera and the framegrabber, and a tapped signal from CC1 of cameralink is the strobe input. The CPU module can control the camera through the combination of COM port and cameralink. The framegrabber interfaces to the main memory through the PCI-bus, and the main memory interfaces with HDD through the combination of FSB, PCI-bus, and IDE cable. Figure 4.4 presents a complete diagram showing all interfaces among the modules.

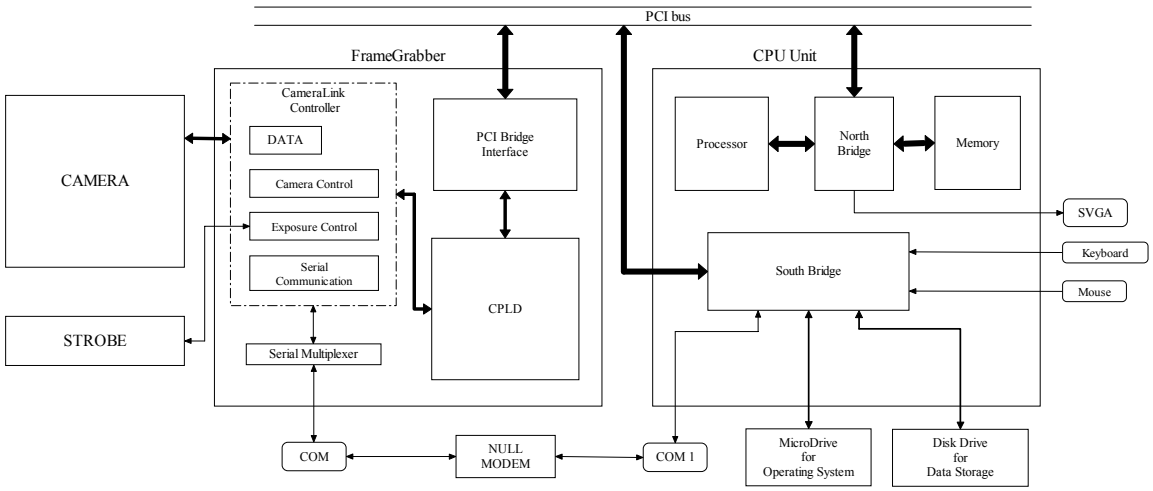


FIGURE 4.4 Architecture of ZOOVIS-SC.

4.2 Interface Transfer Speeds

This section gives an idea about the latency and bandwidth[19] of the three main interfaces of the ZOOVIS-SC system discussed in Section 4.1.

4.2.1 Camera-Framegrabber Interface

The camera operates at a maximum data clock rate of 28.375MHz at its full resolution of 1024×1024, 10-bit pixels and a maximum frame rate of 27fps [16]. The camera connects to the framegrabber through the MDR26 base cameralink.

A 10-bit image at a resolution of 1024×1024 pixels from the camera would be 1.25MB in size. When the camera is working at its maximum frequency, its bandwidth would be 33.75MBps and its latency per image would be 37ms. This bandwidth is much less than the allowable maximum bandwidth on the base configuration of cameralink of 2380Mbps, which

[19] The amount of data that two components can be exchanged over a given period of time.

is equivalent to 297.5MBps. I calculated the above data using the data available in the user manual [16].

4.2.2 Framegrabber-Main Memory Interface

The framegrabber Picasso 104-CL is capable of transmitting 8-bit or 16-bit images. The 10-bit image from the camera changes to either an 8-bit or a 16-bit format by the framegrabber based on the settings. The framegrabber is set to convert the image to 16-bit, to maintain the dynamic range of the image. The framegrabber appends six zero bits at Most Significant Bits (MSB) position to each image coefficient to make up a 16-bit image. This functionality is not a feature of the framegrabber, but is done by default based on the settings when the data converts from cameralink to PCI standards. Further, the padded MSB bits does not affect the intensity levels of the image.

The interface between the framegrabber and CPU modules is the 32-bit, 33MHz PCI-bus. This bus can sustain a maximum bandwidth of 132MBps. The bandwidth required to transfer 27fps at 2MB per frame would be 54MBps. Since the required bandwidth for the interface is much below the maximum bandwidth of the bus, it is capable of handling 27fps. The latency on the PCI-bus for the 2MB image is 15ms.

FSB plays a minor role by contributing to latency in the interface between the framegrabber and main memory. The FSB on the CPU module is 64-bits wide at 100MHz, i.e., 800MBps bandwidth. The latency at this bandwidth for a 2MB image is 2.5ms.

4.2.3 Main Memory-HDD Interface

As described in Section 4.1.3, the interface between the main memory and HDD is internal to the CPU and is through the Northbridge, PCI-bus, and Southbridge. The IDE controller in the Southbridge supports data transfer rates up to 100MBps. The HDD and the IDE cable use the HDD to IDE controller, which also supports this data transfer rate.

Therefore, the IDE cable can sustain the maximum required bandwidth of 54MBps for 27fps. Since this interface also involves the FSB and the PCI-bus, the maximum bandwidth required on the FSB and the PCI-bus would be double, i.e., 108MBps. Since the required bandwidth for the interface is still below the maximum bandwidth of the FSB and PCI-bus, they are still

capable of handling 27fps. Therefore, the total latency on the FSB per image would be 5ms and that on the PCI-bus would be 30ms. Table 4.1 and Figure 4.5 depict the various interface transfer speeds.

TABLE 4.1 Interface Transfer Speeds.

Topology	Cabling	Maximum Bandwidth	Required Bandwidth	Latency
Link	MDR-26 pin Cameralink	297.5MBps	33.7MBps	37ms
Bus	PCI 2.2, 120 pin PC/104-Plus form factor	132MBps	108MBps	30ms
Bus	FSB	800MBps	108MBps	5ms
Cable	80 pin IDE cable	100MBps	54MBps	20ms

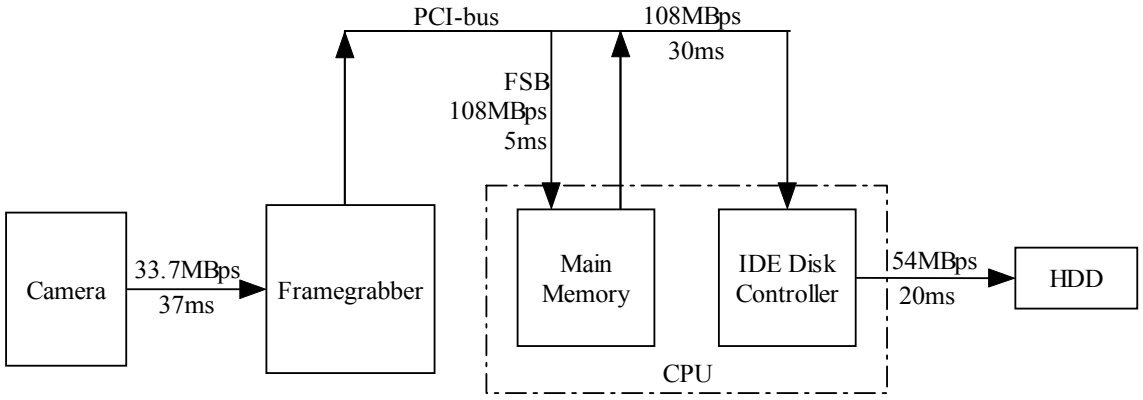


FIGURE 4.5 Interface Transfer Speeds.

4.3 Power Requirements

The power requirements of individual modules and the ability of the power supply module to drive them form the basis of the power scheme. Table 4.2 summarizes voltage and power requirements of the modules.

TABLE 4.2 Power consumption estimates.

Module	Voltage	Current	Power Consumption
CPU Unit	5VDC	2.5A (typical)	12.5 W
Framegrabber	5VDC	1.3A (maximum)	6.5 W
Camera	5VDC	400mA (typical)	2.0 W
Strobe	12VDC	1.1A (maximum)	13.2 W
Disk Drive	5VDC	1.1A (maximum)	5.5 W
Total Consumption			**33.7 W**

Based on the power consumption estimates from Table 4.2, I estimate that the current system configuration requires a power module that can handle approximately 34W of DC power at 5V and 12V. The power module HESC-104 is capable of handling up to 60W.

Because of the system's demand for a compact design, the power supply cable to the various modules must be embedded into existing buses. ISA bus solved the matter by having the capability to transfer 5V DC on its bus to drive the modules. The two modules that are not based on PC/104 standards are the camera and strobe units. Using external connectors provided by the power module for 5V and 12V DC, the power supply cable suits the required connectors on the camera and the strobe units. Figure 4.5 describes the power circuitry.

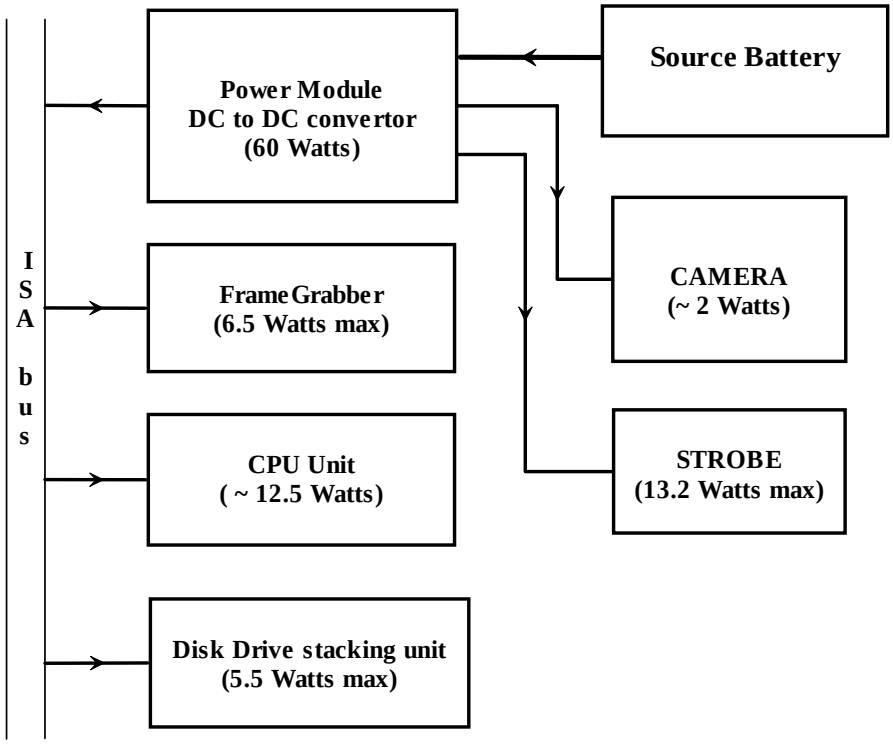


FIGURE 4.5 Power circuitry connecting the modules.

Each deployment of ZOOVIS-SC would be for durations of 30 – 60 minutes, so the battery should hold enough power to run the system for the specified time duration. Based on the calculated power requirements, the battery should be rated at approximately 7Ah to run the system for one hour duration. Any battery that meets the above specifications and can feed input between 6 and 40V DC will suffice.

4.4 Command, Control, and Operation

This section discusses the software control requirements for the ZOOVIS-SC system. From the discussion on interfacing in Section 4.1, the camera will run in external sync mode, and software must generate continuous triggers to capture multiple images. The required user specified parameters are exposure time, frame time, acquisition time[20], delay time[21], and selection of destination directory for image storage.

Integrating all the above requirements, I designed a software model in Visual C using Picasso Imaging libraries. Figure 4.6 depicts the flowchart of the software model. The image capture sequence starts after inputting acquisition time, delay time, and destination directory. The program loads the capture sequence logic[22] parameters to the framegrabber and allocates the required buffer in the main memory based on the input settings. The unit will wait until delay time runs out before starting the actual capture sequence. A time stamp generates by the program at the time the image is captured, and the software saves the image to the disk drive with the time stamp. This loop will repeat until the acquisition time runs out or the disk drive runs out of storage space or the battery runs out of charge.

The flow chart in Figure 4.6 does not involve setting exposure time and frame time. These two parameters are camera settings, and the camera manufacturer did not provide the required software development kit libraries for developing user defined software models. I set these two parameters separately using a sample program “pfremote” provided by the manufacturer.

[20] The time duration for which the capture sequence loop will run.
[21] Wait time before capture sequence loop starts.
[22] The logic involves a loop, which starts by generating a software trigger, transmitting it to the camera and strobe, capturing data from the camera, and ends after transferring it to the disk drive.

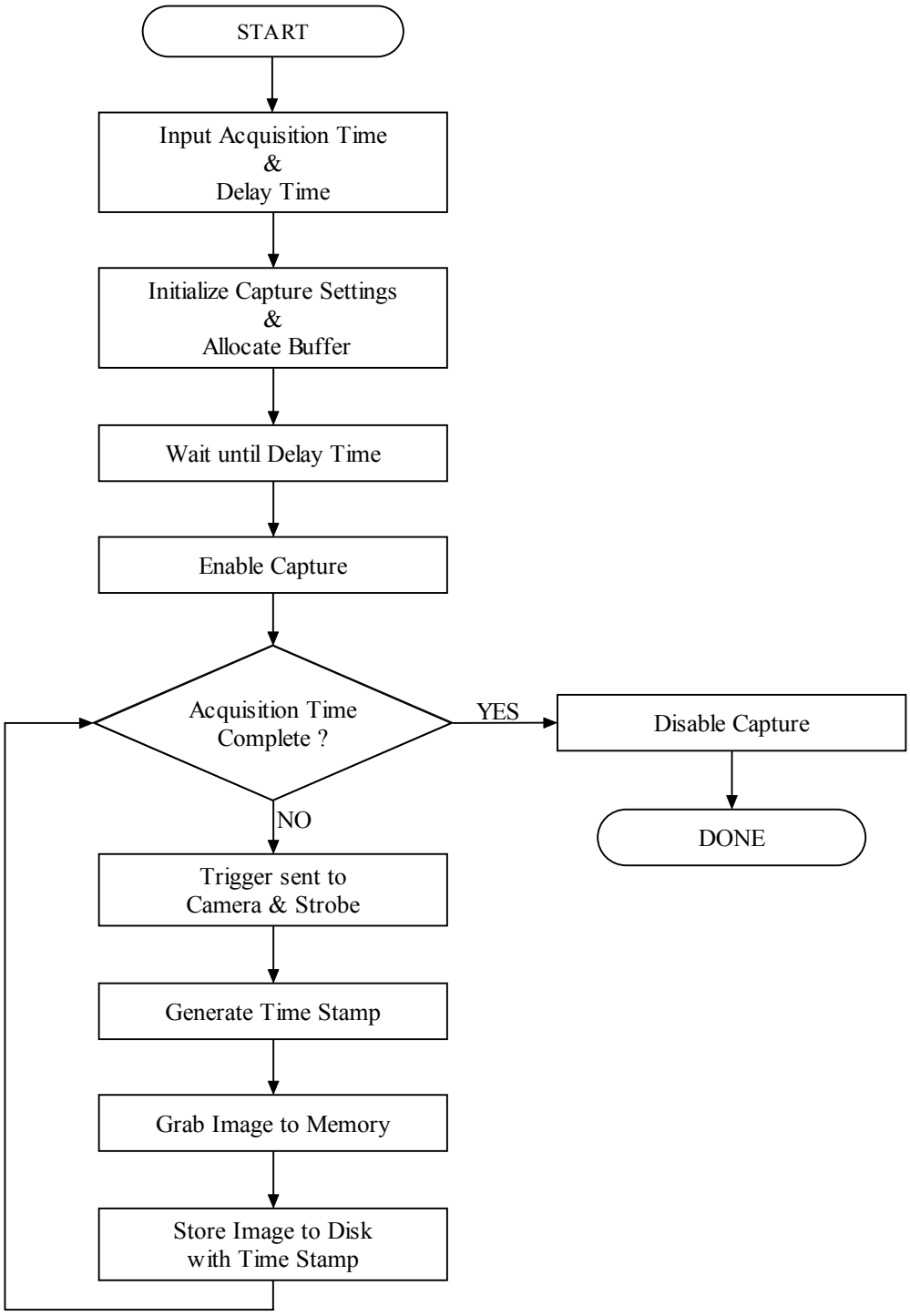


FIGURE 4.6 Flow diagram showing software operation when capturing a sequence of images for a specified time period.

5

ACHIEVEMENTS AND DRAWBACKS OF ZOOVIS-SC

As discussed in Section 2.3, the function of ZOOVIS-SC is the collection of discrete zooplankton samples on the scales of thin zooplankton layers. This requires an easy to use and handle system, and the prototype ZOOVIS-SC meets some of these requirements. ZOOVIS-SC will house the hardware modules discussed in Chapter 3 in a single pressure housing approximately 24" long and 6" diameter, while the strobe will be placed in a small separate housing.

The ZOOVIS-SC system will not require any on field assembly. Configuration for deployment will use a simple software interface and will only require the connection of a monitor, keyboard, and mouse while making sure that the battery is fully charged. During sampling, stabilizing devices could hold the pressure housing in a particular position. After deployment, the user can transfer recorded data from ZOOVIS-SC to any HDD using either an IDE or 100BaseT interface on the back of the housing and can process the downloaded data either immediately or at a later time. Once the battery recharges, ZOOVIS-SC would be ready for another deployment.

For the successful operation of ZOOVIS-SC, it is critical that all modules interface properly and that their data transfer rates approach the targeted frame rate of 27fps. Sections 4.1 and 4.2 discussed the various interfaces and theoretical transfer speeds. The following sections compare the data transfer rates and discuss various drawbacks of the ZOOVIS-SC system.

5.1 ZOOVIS-SC Data Transfer Rates

This section discusses theoretical, estimated, and practical data transfer rates.

5.1.1 Theoretical Data Transfer Rates

Table 5.1 summarizes the theoretical values for data transfer rates discussed in Section 4.2.

TABLE 5.1 Theoretical Data Transfer Rates.

Component Interface	Interface	Latency	
Camera - Main Memory	Cameralink	37ms	54.5ms
	PCI-bus	15ms	
	FSB	2.5ms	
Main Memory - HDD	FSB	2.5ms	37.5ms
	PCI-bus	15ms	
	IDE cable	20ms	

The theoretical transfer rate gives a total latency of 92ms per image. This seals the maximum possible frame rate of the current configuration of the ZOOVIS-SC system to 10.87fps.

5.1.2 Estimated Data Transfer Rates

The theoretical data transfer rates are the maximum possible transfer rates that can be achieved on the link, bus, or cable. I estimate data transfer rates based on various observations from manufacturers and users. I collected the data presented in this section from various sources like white papers, technical support from respective vendors [16, 17, 18, 19], forums, and other Internet sources.

Cameralink streams data at very high reliable rates. Further, the required bandwidth is approximately 12% of its maximum bandwidth of 297.5MBps. Hence, the drop in bandwidth in practical usage of the cameralink will not affect the required bandwidth of 33.7MBps. Similarly, FSB can handle the required throughput of 108MBps, which is approximately 14% of its maximum bandwidth of 800MBps.

From previous works, I estimated the bandwidth on the PCI-bus to be 95MBps, which gives a minimum latency of 42ms for 2MB images. Since the required bandwidth for 27fps is 108MBps and is higher than the estimated maximum bandwidth, I expect the frame rate to fall short of the target.

HDD's with mechanical parts experience a considerable delay. The delay depends on various factors like rotational speed, latency, and seek times. These bring down the performance of an HDD to approximately 33% of its maximum throughput. Therefore, the estimated bandwidth for the interface to the HDD is 33MBps, which gives a latency of 60.6ms. This bottleneck in the data transfer route will further bring down the frame rate. Table 5.2 summarizes the estimated data transfer rates.

TABLE 5.2 Estimated Data Transfer Rates.

Component Interface	Interface	Latency	
Camera - Main Memory	Cameralink	37ms	60.5ms
	PCI-bus	21ms	
	FSB	2.5ms	
Main Memory - HDD	FSB	2.5ms	84.1ms
	PCI-bus	21ms	
	IDE cable	60.6ms	

From Table 5.2, the estimated total latency is 144.6ms, giving a frame rate of 6.92fps.

5.1.3 Practical Data Transfer Rates

The practical data rates measured using C programs utilize the system clock to calculate the latency between the interfaces. The precision of these programs is ±10% of the actual data transfer rates. Hence, the data transfer rates provided in Table 5.3 are the average from several test runs.

TABLE 5.3 Practical Data Transfer Rates.

Component Interface	Interface	Latency
Camera - Main Memory	Cameralink	96ms
	PCI-bus	
	FSB	
Main Memory - HDD	FSB	92ms
	PCI-bus	
	IDE cable	

The total latency of a 2MB image from Table 5.3 is 188ms. This gives an achieved frame rate of 5.32fps which is close to the estimated frame rate of 6.92fps. It is approximately 50% of the theoretically possible frame rate, which is perfectly normal for PC architecture interfaces.

5.2 Drawbacks

As discussed in Section 5.1, the attained frame rate is 5.32fps. From the standpoint of the ZOOVIS-SC system, 5.32fps is not acceptable. In order to collect representative estimates of zooplankton abundance, it is necessary to sample a sufficiently large volume of water. At 5fps and a sample volume of 16ml (4cm×4cm×1cm), the system would quantify only 80ml/sec or 4.8L/min. This is too low to reliably detect organisms in a thin layer. At 27fps, the system would image 25.92L/min. The end result is that the captured images from the current configuration of ZOOVIS-SC would not be sufficient for a detailed study of these marine organisms.

Another set of drawbacks involves the current camera and framegrabber module. The responsitivity of the CMOS sensor in the camera is low in poor lighting at exposure times of 10μsec. This results in grainy images. Experiments have shown that the maximum possible image size from the framegrabber is 1016×1016, instead of the specified 1024×1024. Also the six extra bits that the framegrabber pads at MSB bit position to each of the image coefficients are not all zeros. This results in the intensity values of the captured image going up by a constant factor but does not affect image contrast. A change in camera and framegrabber modules can overcome these drawbacks; however, this is not our primary concern at this stage.

5.3 Possible Solutions

One solution is to get a faster storage device, because the slow HDD is the main bottleneck. HDD's with rotational speeds of 7200rpm are the fastest available in ATA technology. Alternative high speed storage devices would be SCSI HDD's that run at rotational speeds of 10,000rpm and higher. A SCSI interface, however, would require additional hardware and power. Further, this additional hardware should not be on the PCI-bus because the PCI-bus is already working at near theoretical limits as discussed in Section 5.1.2.

PCI is the main bus on an embedded architecture like the PC/104-*Plus* and requiring the HDD controller to not utilize that bus is not possible. The other solution is to get an HDD controller that works at higher PCI bandwidth. This indicates a major change in the basic architecture of ZOOVIS-SC for this solution to be implemented.

An alternative to the above approach would be to reduce the size of the image. Reducing the size of each image to 50% or less permits an increased frame rate of 8fps and higher. Three methods can reduce the size of the image. One method is reduction in image resolution. This results in lesser captured area, which results in less information. A second method is to reduce the dynamic range of the captured images from 10-bits to 8-bits. This degrades the quality of the captured images, affecting the fine details in the images. Therefore, these two methods do not serve the purpose of ZOOVIS-SC.

A third method is to maintain the image at its highest resolution and compress the image. Further, the compression must be lossless to avoid loss of data. This would be a viable solution only if the lossless image compression ultimately reduces the latency on the transfer path.

Image compression done in software is not as fast as that done on dedicated hardware resources. A custom designed Application Specific Integrated Circuits (ASIC) chip with compression implemented is the best option, but ASIC chips are not cost and time effective. Further modifications to implemented algorithms are not possible. This brings out the need for programmable hardware resources.

An FPGA is the solution. FPGAs' provide truly generic hardware and at the same time allow for modifications at any stage. They provide higher orders of magnitude speedup over software simulations [21] and yield significant cost and time savings. They are slower than ASICs, but FPGAs' cost and time effectiveness compensates for this drawback.

5.4 Conclusion

I recommend implementing a lossless image compression on a PC/104-*Plus* standard FPGA to maintain the current architecture and to rectify the architectural drawbacks of ZOOVIS-SC.

PART – II

REAL-TIME LOSSLESS IMAGE COMPRESSION

6

BACKGROUND ON IMAGE COMPRESSION

Lossless compression of images involves a completely reversible scheme by which the original data can be reconstructed exactly. Several fields in addition to underwater imaging, including space science, medical imaging, and nondestructive testing of materials, require lossless image compression. In each of these fields, the features of interest in the images have characteristics similar to noise, so any lossy method that removes noise may also remove meaningful data [22]. This chapter discusses such lossless image compression techniques and their hardware implementation.

6.1 Base Image Compression Algorithm

The image compression algorithm I intend to implement consists of two phases. The first phase comprises multi-resolution transformation of the image, and the second phase comprises bit-ordering of the transformed image coefficients. Bit-ordering methods arrange data bits such that the stored data includes only the significant bits of the data. I refer to this as the base algorithm because it forms the basis from which I develop the image compression algorithm for the ZOOVIS-SC application. The base image compression algorithm, developed by Said and Pearlman [3], uses the DWT transform in the first phase followed by SPIHT bit-ordering in the second phase, which effectively utilizes a multi-resolution pyramid structure for bit-ordering.

SPIHT suits the bit-ordering phase because it displays exceptional characteristics that include fast coding and decoding, can combine with error protection schemes, and accommodates for lossless compression [23]. A multi-resolution S+P transform followed by SPIHT bit-ordering provides an alternative to the base compression algorithm [2].

As discussed above, the generic compression algorithm comprises a multi-resolution transform followed by a bit-ordering phase. Sections 6.1.1 and 6.1.2 presents the DWT and S+P transforms, two different multi-resolution transforms. Section 6.1.3 presents SPIHT, a bit-ordering algorithm.

6.1.1 DWT

The wavelet transform is an image processing technique that can be used in real-time image processing over time-frequency representation techniques such as the discrete fourier transform and the discrete cosine transform [22]. By virtue of its multi-resolution representation capability, many applications effectively use the wavelet transform.

In an octave-band DWT decomposition scheme, a non-uniform band splitting method decomposes the lower frequency part into narrower bands and leaves out the high-pass output at each level without any further decomposition. Figure 6.1 presents a 3-level wavelet decomposition of the Lena image and the structure of subband.

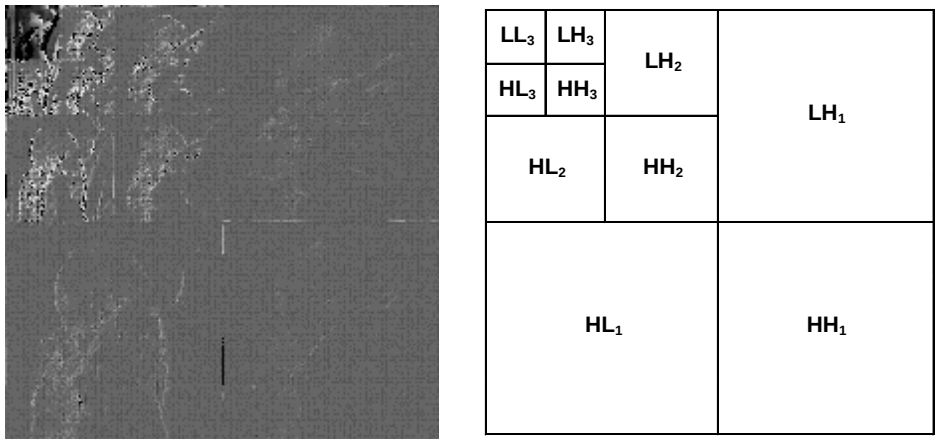


FIGURE 6.1 (a) 3-level octave-band decomposition of Lena image, and (b) low and high frequency subband structure of a 3-level wavelet transform.

Wavelet transform filters are nearly lossless, but compression employing a wavelet transform may not be reversible since they have non-integer tap weights and they produce non-integer transform coefficients, which truncate to finite precision [3]. For perfectly lossless compression one must use an integer multi-resolution transform, such as the S+P transform described below, which yields excellent lossless compression ratios.

6.1.2 S+P Transform

Said and Pearlman [2] proposed an image multi-resolution transform that suits lossless image compression. This section summarizes the S+P transform from Said and Pearlman [2]. The S+P transform uses a simple pyramid multi-resolution scheme, which enhances the high frequency coefficients via predictive coding. The new transformation is similar to subband decomposition, but requires only integer addition and bit-shift operations.

The sequences below represent the low and high frequencies of integer coefficients, c;

$$l[n] = \lfloor (c[2n] + c[2n+1]) / 2 \rfloor \text{ and}$$
$$h[n] = c[2n] - c[2n+1],$$

where $n = 0, .., N/2 - 1$ and $\lfloor . \rfloor$ corresponds to downward truncation. The sequences l and h form the low and high pass components of the S transform of c. Since the sum and difference of two integers correspond to either two odd or two even integers, the fractional part is irrelevant and hence truncated.

The inverse transformation is:

$$c[2n] = l[n] + \lfloor (h[n] + 1) / 2 \rfloor \text{ and}$$
$$c[2n+1] = c[2n] - h[n].$$

The low and high pass components formed after applying the one-dimensional S transform sequentially to columns is shown in Figure 6.2(b). Repeating the one-dimensional S transform sequentially on the rows of the resultant coefficients forms the transformed image shown in Figure 6.2(c). The one-dimensional column transformation followed by the row transformation results in a two-dimensional transformation of the image. The coefficients corresponding to *ll* in Figure 6.2(c) are the mean on 2×2 coefficient blocks, and they form another image with half resolution. A multi-resolution hierarchical pyramid is formed by applying the same transformation to the reduced resolution "mean images" [24]. Figure 6.2(d) represents the multi-resolution pyramid structure.

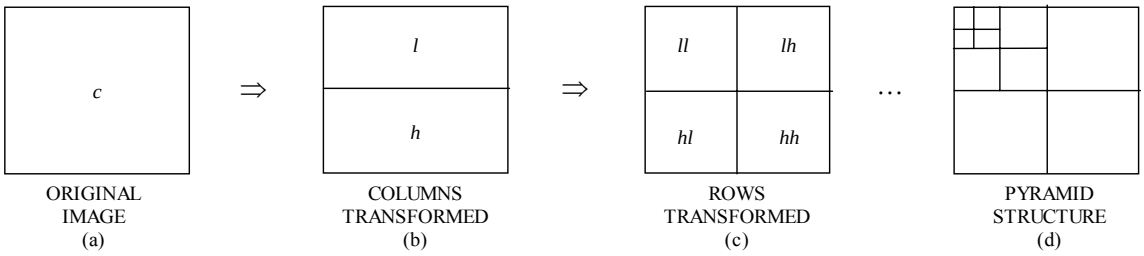


FIGURE 6.2 Construction of an image multi-resolution pyramid from one-dimensional transformations [2].

The sequential transform is simple, efficient, and significantly reduces the first-order entropy. It leaves residual correlation between the high pass components, however. A predictive coding method applied during the one-dimensional sequential transform reduces this correlation.

In the S+P transform, during each one-dimensional transformation, the high pass components are estimated from l and h components calculated during the S transform. Calling the sequence of estimates $\hat{h}$, the difference:

$$h_d[n] = h[n] - \left\lfloor \hat{h}[n] \right\rfloor, \text{ where } n = 0,..,N/2-1$$

replaces the high pass components, h, in the one-dimensional transformed image, forming a new transformed image with smaller first-order entropy. No such estimation is done for the low pass components since in the next transform they split into low and high pass components and the new high pass components transform using the same method. Defining

$$\Delta l[n] = l[n-1] - l[n],$$

the general form of the estimator is:

$$\hat{h}[n] = \sum_{i=-1}^{1} \alpha_i \Delta l[n+i] - \beta_1 h[n+j],$$

where α_{-1}, α_0, α_1, and β_1 are predictors. Table 6.1 presents a set of selected predictors for specific classified images.

TABLE 6.1 Parameters of the set of selected predictors [2].

Predictor	Parameter			
	α_{-1}	α_0	α_1	β_1
A	0	1/4	1/4	0
B	0	2/8	3/8	2/8
C	-1/16	4/16	8/16	6/16

Predictor A has the smallest computational complexity, while B applies to natural images, and C to very smooth medical images. For instance, applying predictor B on $\hat{h}[n]$ and simplifying the notation:

$$\hat{h}[n] = \frac{1}{8}\{2(\Delta l[n] + \Delta l[n+1] - h[n+1]) + \Delta l[n+1]\}.$$

In the image borders, the predictors are:

$$\hat{h}[0] = \Delta l[1]/4, \quad \hat{h}[N/2-1] = \Delta l[N/2-1]/4 .$$

During the inverse one-dimensional transformation, the prediction follows a reverse order for addition,

$$h[n] = h_d[n] + \lfloor \hat{h}[n] \rfloor, \text{ where } n = N/2-1, N/2-2, ..., 0,$$

so the high pass components required to calculate the prediction for the current *n* are recovered. Further, since the transformation was first applied to the columns and then to the rows, the inverse transform must be applied to the rows first and then to the columns.

I selected the S+P transform for the multi-resolution transform phase because it is similar to subband decomposition, but can be computed with only integer addition and bit-shift operations and yields excellent lossless compression of images. Also, numerical results show that the S+P transform yields more compression than with predictive coding methods of similar complexity [2]. Therefore, methods such as SPIHT can exploit the multi-resolution structure and can efficiently compress an S+P transformed image.

6.1.3 SPIHT

SPIHT, presented by Said and Pearlman [3], is an efficient method for lossy and lossless coding of still images. The image is transformed prior to encoding with SPIHT. This section summarizes the SPIHT bit-ordering algorithm from Said and Pearlman [3].

SPIHT exploits the fact that the energy of an image concentrates in the low frequency components. SPIHT coefficients are ordered in hierarchies, called spatial orientation trees, such that the roots in the lowest frequency subband branch successively into higher frequency subbands at the same spatial orientation [25]. The nodes at the highest frequency do not have any offspring. Figure 6.3 shows how the spatial orientation tree is defined in a pyramid constructed with recursive four-subband splitting.

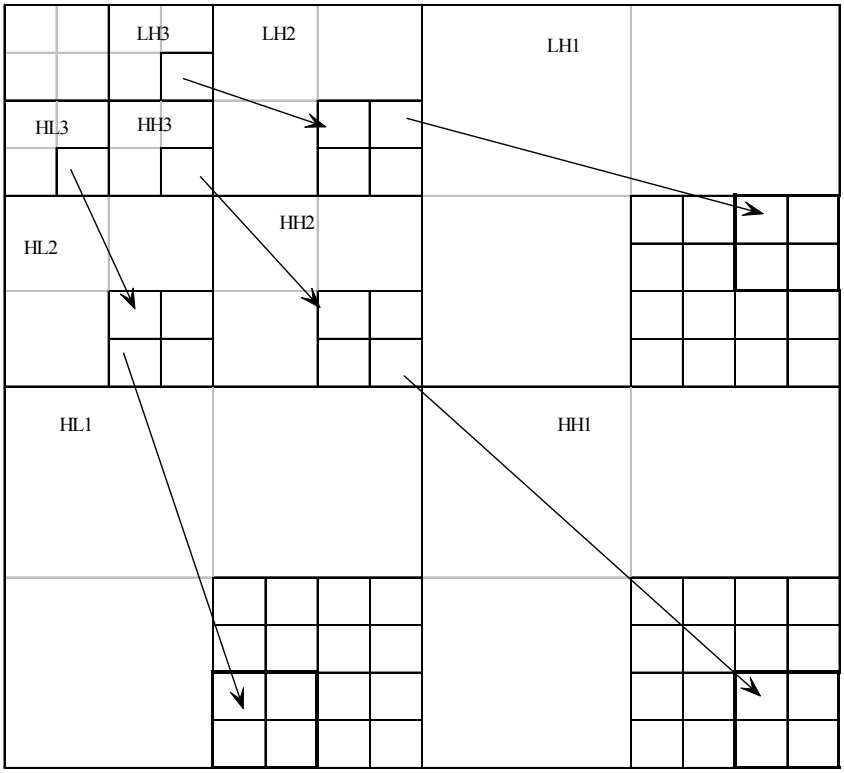


FIGURE 6.3 Parent-offspring dependencies in the spatial-orientation tree after 3-level decomposition.

In a spatial orientation tree structure, the offspring of an image coefficient correspond to the image coefficients of the same spatial orientation in the next finer level of the pyramid. Each node of the tree corresponds to the image coefficients. Therefore in a tree each node should have four offspring, which always form a group of 2×2 adjacent nodes. However, the node in the low frequency region at each level does not have offspring. The tree is thus defined such that each node has either no offspring or four offspring. The nodes at the highest level of the pyramid are the tree roots. As shown in Figure 6.3, these tree roots are also grouped in 2×2 adjacent nodes. In Figure 6.3 the arrows are oriented from the parent node to its four offspring.

The SPIHT coder codes the bit-planes[23] of the coefficients one at a time in decreasing order of priority. The priority is set by coding the most significant bit plane first and the least significant bit planes the last. While coding a bit plane, the SPIHT coder start with the *LL* block first and scans the *HH* block last.

SPIHT encodes and transmits coefficients in multiple passes. In each pass only the coefficient with magnitudes exceeding a certain threshold are encoded. The magnitude test is performed according to the expression

$$\max_{(i,j)\in r}\left\{\left|c_{i,j}\right|\right\} \geq 2^r ,$$

where $r+1$ is the number of significant bits of the threshold and $c_{i,j}$ represents the coefficient at position (i, j) in the image. The significance of a set of coefficients τ for r is defined as

$$S_r(\tau) = \begin{cases} 1, if \ \max_{(i,j)\in r}\left\{\left|c_{i,j}\right|\right\} \geq 2^r \\ 0, Otherwise \end{cases},$$

From the above equation, a coefficient is considered significant or insignificant with respect to a given threshold, depending on whether or not it exceeds that threshold. When all coefficients such that $2^r \leq \left|c_{i,j}\right| < 2^{r+1}$ are checked for significance, a pass is said to have completed and *r* is decremented and the pass is repeated again.

The SPIHT algorithm decomposes each pass into a sorting pass for evaluating coefficient and set significances and a refinement pass for refining the values of the coefficients found significant in the previous passes. Significance information is stored in three ordered lists, called List of Insignificant Sets (LIS), List of Insignificant Pixels (LIP), and List of Significant Pixels (LSP). During the sorting pass, the significance of the coefficients in the LIP is coded, and the coefficients that become significant are moved to the end of the LSP and their signs are coded. Similarly, the significances of the descendant sets are coded and those that become significant are partitioned and further coded as presented in Figure 6.4. The values of the coefficients $c_{i,j}$ in the LSP except the ones included in the last sorting pass

[23] The r^{th} bit plane is bit *r* of every coefficient such that, for a two dimensional image with the coefficients arranged in order of bit position in the third dimension, r^{th} bit plane is a slice (or plane) of level *r* of the third dimension.

are refined in each refinement pass. Specifically, for a coefficient $c_{i,j}$, the refinement pass outputs the r^{th} most significant bit of $\left|c_{i,j}\right|$.

The set of all descendants[24] of a node corresponding to a coefficient at the coordinates (i, j) is termed a type *A* set and denoted by $D(i, j)$. The set of all descendants excluding the offspring[25] set $O(i, j)$ of a node corresponding to a coefficient at coordinates (i, j) is termed a type *B* set and denoted by $L(i, j) = D(i, j) - O(i, j)$.

Figure 6.4 presents the SPIHT coding algorithm from Said and Pearlman [3], where $r+1$ represents the number of significant bits of the threshold, H is set of coordinates of all spatial orientation tree roots (nodes at the highest pyramid level), and $S_r(i, j)$, $S_r(D(i, j))$, and $S_r(L(i, j))$ indicates significance of $c_{i,j}$, $D(i, j)$, and $L(i, j)$, respectively.

1. **Initialization:** output $r = \left\lfloor \log_2 \left(\max_{(i,j)} \{ \left|c_{i,j}\right| \} \right) \right\rfloor$; set the LSP as an empty list, and add the coordinates $(i, j) \in H$ to the LIP, and only those with descendants to the LIS as type A entries.
2. **Sorting Pass:**
   2.1. for each entry (i, j) in the LIP do:
      2.1.1. output $S_r(i, j)$;
      2.1.2. if $S_r(i, j) = 1$, then move (i, j) to the LSP and output the sign of $c_{i,j}$;
   2.2. for each entry (i, j) in the LIS do:
      2.2.1. if the entry is of type *A*, then
      - output $S_r(D(i, j))$
      - if $S_r(D(i, j)) = 1$, then
        - for each $(k,l) \in O(i, j)$ do:
          - output $S_r(k, l)$;
          - if $S_r(k, l) = 1$, then add (k, l) to the LSP and output the sign of $c_{k,l}$;
          - if $S_r(k, l) = 0$, then add (k, l) to the LIP;
        - if $L(i, j) \neq \emptyset$, then move (i, j) to the end of the LIS as an entry of type *B*, else remove entry (i, j) from the LIS;

FIGURE 6.4 SPIHT coding algorithm.
(Figure Continued)

[24] The set of nodes that includes the offspring set and their children.
[25] The four children at the next finer scale with the same spatial orientation.

2.2.2. if the entry is of type *B,* then

- ❖ output $S_r(L(i, j))$;
- ❖ if $S_r(L(i, j)) = 1$, then
  - add each $(k, l) \in O(i, j)$ to the end of the LIS as an entry of type *A*;
  - remove (i, j) from the LIS.

3. **Refinement Pass:** for each entry (i, j) in the LSP, except those included in the last sorting pass (i.e., with current *r* value), output the r^{th} most significant bit of $\left|c_{i,j}\right|$;
4. **Quantization-step update:** decrement *r* by 1 and go to Step 2**.**

The important property of the SPIHT algorithm is that it codes information corresponding to individual bits of the transformed image following a bit-plane binary coding method [3]. It has distinct characteristics, however. The bits are not transmitted in the usual line-by-line order, and tree structures are used in such a way that, with a single coded symbol, the decoder can infer that all the bits in a large region of a given bit plane are zero. Another important feature of SPIHT is that the ordering data is not explicitly transmitted. Instead, it is based on the fact that the execution path of any algorithm is defined by the results of the comparisons on its branching points [3]. So, if the decoder has the same sorting algorithm has the encoder, it can duplicate the encoder's execution path from the results of the magnitude comparisons, and the execution path recovers the ordering information.

In summary, the SPIHT algorithm uses the principles of partial ordering by magnitude, set partitioning by significance of magnitudes with respect to a sequence of decreasing thresholds, ordered bit plane transmission, and self-similarity across scale in a transformed image [3].

6.2 Hardware Implementation

A multi-resolution transform followed by SPIHT is computationally intensive and operates on large data sets. This factor, coupled with the demand for real-time operation in many image processing tasks, makes traditional sequential computers fall short in meeting such requirements. In turn, this necessitates the search for high performance implementations at a reasonable cost. Implementations of image compression group into two major categories: software implementations using programmable parallel systems and dedicated hardware implementations using customized Very Large Scale Integrated (VLSI) devices. Each

implementation category presents different trade-offs in terms of performance, cost, power, and flexibility.

In general, custom VLSI circuits are inherently inflexible, and their development is costly and time consuming. Thus, they are not an attractive option for implementing the compression algorithm. With the advent of FPGAs, designers were able to develop and test custom hardware solutions without the need for going through costly and time consuming fabrication procedures.

Section 6.2.1 presents the basic architecture of an FPGA. Section 6.2.2 and 6.2.3 refer to previous architectural implementations of the DWT and S+P transform for the transform phase and SPIHT for the bit-ordering phase, respectively.

6.2.1 FPGA

As the name implies FPGAs are integrated circuits that are field programmable after manufacture. FPGAs maintain the advantages of the custom functionality of VLSI ASIC devices, while avoiding the high development costs and the inability to make design modifications after production [15]. Furthermore, FPGAs inherit design flexibility and adaptability of software implementations.

The typical architecture of an FPGA comprises a regular array of Configurable Logic Blocks (CLB) with routing resources for interconnection and surrounded by programmable Input Output Blocks (IOB) (Figure 6.5). CLBs provide the functional elements for constructing logic, while IOBs provide the interface between the pins of the package and the CLBs. FPGAs find extensive use in prototype development before fabrication of a VLSI design. FPGAs are also used directly in a product.

These advantages of FPGAs have made them capable of being used in imaging applications, which require complex and long computations.

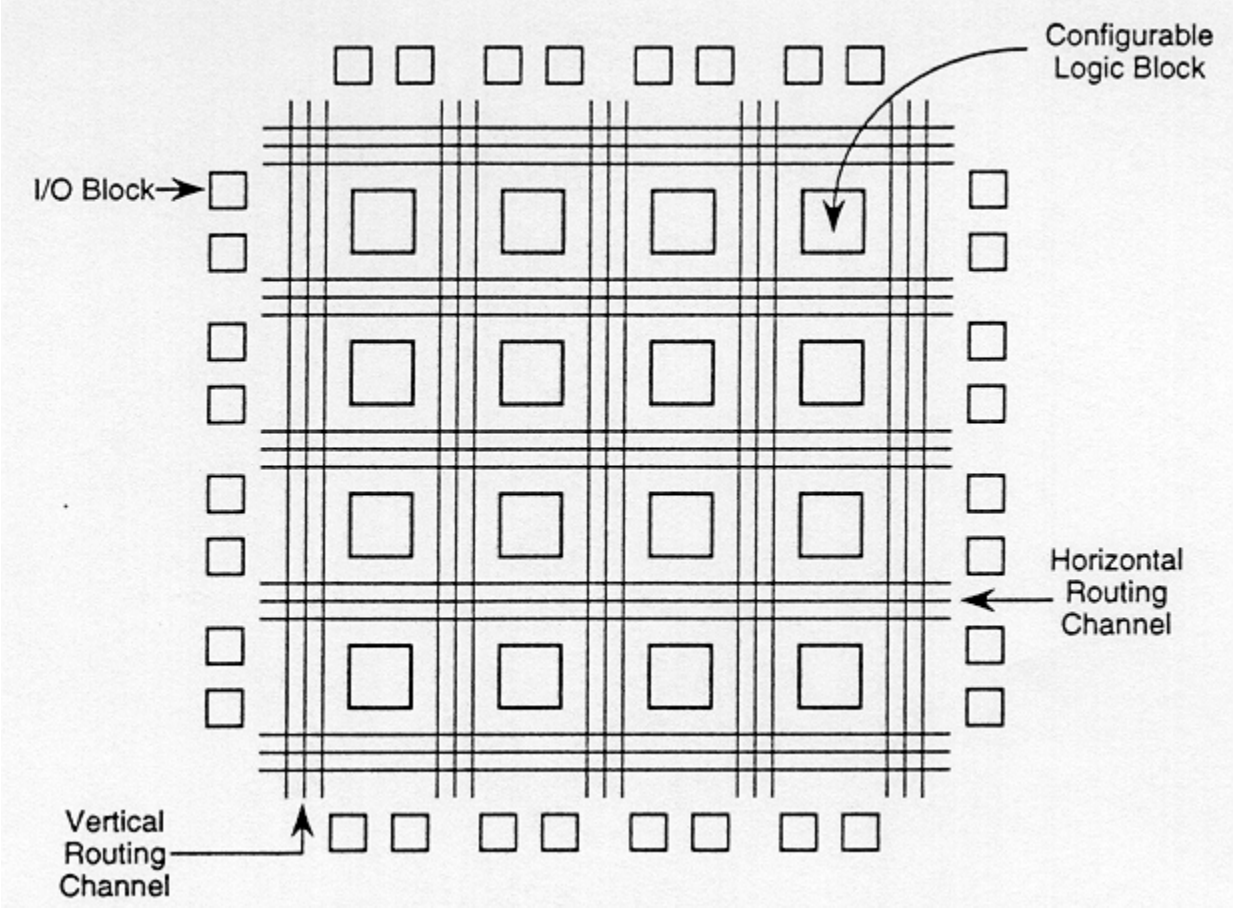


FIGURE 6.5 Typical FPGA architecture [16].

6.2.2 DWT and S+P Transform

Several architectures were proposed for the implementation of the DWT. The first architecture, presented by Knowles [28], uses many large multiplexers for storing intermediate results. Parhi and Nishitani proposed a folded architecture that has shorter latency [29], however, it requires a complex routing and control network. Chakrabarti [30] proposed a systolic architecture, but it also requires parallel hardware and complex routing. Ritter [31] used pipelining to increase the computational power of DWT on FPGAs. Chakrabarti *et al.* [32] presented a survey of DWT architectures.

Figure 6.2 depicts the S+P transform that uses a simple pyramid multi-resolution scheme and is similar to the DWT decomposition in Figure 6.1. Hence, the dataflow of the basic hardware architectures proposed by various researchers are relevant to implement the S+P

transform. The actual computations, however, differ. I intend to modify the folded architecture proposed by Parhi and Nishitani [29] and implement the S+P transform.

6.2.3 SPIHT

Singh [33] proposed an architecture to implement SPIHT on FPGAs. The basic SPIHT algorithm, however, makes use of dynamic data structures to exploit the self-similarities explained in Section 6.1.2. This makes a hardware implementation of the basic SPIHT algorithm very difficult. Ritter [31] modified SPIHT and implemented it on a Virtex XC4000 device; Fry [34] introduced fixed-order SPIHT where the order of the LIP, LIS, and LSP lists is fixed and known beforehand, and implemented it on a Virtex 2000E FPGA.

7

COMPRESSION ALGORITHM: MODIFICATIONS AND IMPROVEMENTS

The S+P transform algorithm discussed in Section 6.1.2 substantially lowers the intensity levels of the image coefficients through its lossless integer multi-resolution transformation. Further, it requires approximately three adders, two subtractors, and shifters for its hardware computations. Therefore, implementation of the S+P transform with its low complexity and low hardware requirements suits well the multi-resolution phase.

Implementation of SPIHT for the bit-ordering phase, however, has certain drawbacks. This chapter discusses these drawbacks of SPIHT, possible modifications, and alternative methods to overcome these drawbacks. The new methods that I propose specifically take into consideration 1024×1024, 10-bit images from ZOOVIS-SC. I have tested and verified the validity of the proposed methods in Matlab. All compression methods in this chapter achieve lossless compression, so I do not compare MSE and PSNR since they are "0" and "∞", respectively.

The images from ZOOVIS-SC will have the same characteristic features as those of ZOOVIS, since both take images of zooplankton in their natural surroundings. ZOOVIS-SC would take images at 1024×1024, 10-bit resolution, whereas ZOOVIS takes images at 2048×2048, 14-bit resolution. Since ZOOVIS-SC is still in its design stage and images of zooplankton in its natural surroundings cannot be obtained using ZOOVIS-SC, I use resized images from ZOOVIS to test and compare results among the various proposed methods and with SPIHT. Figure 7.1 presents sample images of zooplankton in its natural surrounding taken by ZOOVIS.

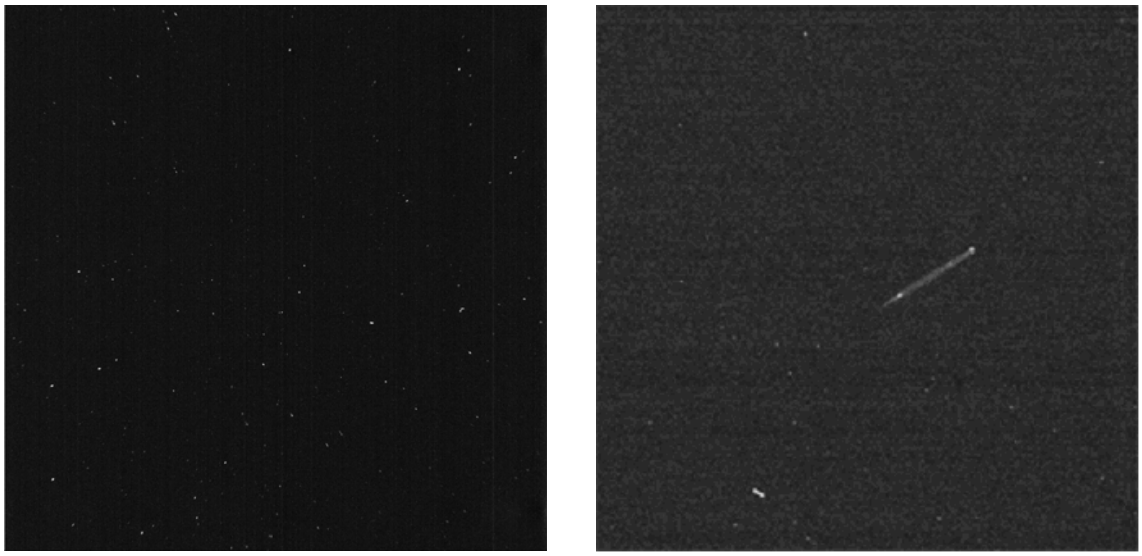


FIGURE 7.1 Images from ZOOVIS camera.

7.1 SPIHT Drawbacks

SPIHT offers excellent compression performance and fast encoding and decoding times. Certain algorithmic and hardware constraints, however, restrict the implementation of SPIHT. Section 7.1.1 discusses the algorithm constraints and Section 7.1.2 discusses the hardware constraints that restrict the implementation of SPIHT.

7.1.1 Algorithmic Constraints

The SPIHT algorithm generates excellent compression ratios for flat images. For images with high intensity coefficients scattered all around the image, however, this ratio falls because these coefficients are significant at a higher threshold than their immediate neighbors. The processing of these high intensity coefficients requires placing its insignificant neighbors in the LIP set and their parents and immediate offspring sets in LIS. This generates an overhead of significance information bits for entries in LIP and LIS because their significance is checked for every threshold value until they are found significant.

The images obtained from ZOOVIS even after the multi-resolution transform phase have high intensity values scattered all around the image and hence, for the bit-ordering phase, SPIHT cannot produce high compression for ZOOVIS images.

7.1.2 Hardware Constraints

As discussed in Section 6.1.2, SPIHT makes use of lists of coefficients to manage the significance information. The two disadvantages with this method are that the lists can grow arbitrarily large, which makes estimation of the required memory resources impossible, and parallelization is difficult because of the hierarchical tree structuring scan order of the coefficients.

7.2 Possible Modifications

The base image compression algorithm comprising DWT followed by SPIHT is popular among image compression schemes because of its progressive image transmission capability. A progressive transmission scheme allows the use of a small part of the losslessly encoded data to recover a lossy version of the original data; thus, a user can browse the lossy data prior to obtaining the full lossless data. This feature is not necessary for the ZOOVIS-SC system, however, because the target is lossless compression. So transfer rates are relevant and not the significance order for compressing and transferring the image. This permits new ways of implementing the bit-ordering phase that utilize some properties of SPIHT and overcome hardware and algorithmic constraints discussed in Section 7.1.

In Section 7.2.3 I propose the 4-Cluster method for the bit-ordering phase. I also propose two other methods in Sections 7.2.1 and 7.2.2 which form the basis of the transition from SPIHT to the 4-Cluster method. Each of the sections is organized as follows: first I discuss the basis for the new method and then present its algorithm; an explanation of the algorithm either precedes or follows the algorithm; then the working of the decoder is discussed followed by the advantages of the method, its compression results on ZOOVIS and Lena images, and comparison with SPIHT.

7.2.1 Independent List Method (Method 1)

The independent list method aims to overcome the drawback that SPIHT may check the significance of many coefficients more than once. The proposed method allocates *n* different LSP lists for an *n*-bit image. When the method checks a coefficient for significance, it places the coefficient in the list that corresponds to the coefficient's threshold value. Repeat this process for each coefficient in the image in sequential order. Hence, the method checks each coefficient for significance only once.

Figure 7.2 describes the Method 1 algorithm on the basis of the SPIHT algorithm description. Variable $c_{i,j}$ represents the current coefficient under consideration, and μ_r gives the total number of entries into each LSP$_r$ list, where $0 \le r \le n$ is the number of significant bits in the coefficient $c_{i,j}$, and r is the number of significant bits of the threshold such that $2^r \le |c_{i,j}| < 2^{r+1}$. The method neglects leading '0' bits and does not transmit the leading '1' of each entry in the LSP lists since it can be inferred from index r of list LSP$_r$. Further, this method does not transmit any bits for $c_{i,j} = 0$. The output bits, in the sequence transmitted, compose the compressed file.

1. **Initialization**: Enter all pixels in raster scan order into the **LIP** list. Create *n* separate **LSP** lists, one for each threshold value.
2. **Sorting Pass**: for each entry $c_{i,j}$ in the **LIP**:
   2.1 If $c_{i,j} \neq 0$, then
   - **Output** $r = 1 + \lfloor \log_2(|c_{i,j}|) \rfloor$ and increment μ_r.
   - **Output** sign of $c_{i,j}$ and move $|c_{i,j}|$ to the **LSP**$_r$ list.

   2.2 If $c_{i,j} = 0$, then **output** $r = 0$.
3. **Output** μ_r for each **LSP**$_r$ list.
4. **Refinement Pass:** for each **LSP**$_r$ list starting with the n^{th} **LSP** list.
   4.1 If $r > 0$, then **output** $r - 1$ LSBs for each **LSP**$_r$ list.
   4.2 If $r = 0$, then output no bits.

FIGURE 7.2 Method 1 Bit-Ordering Algorithm.

Method 1 requires a minimum of 5bpp and a maximum of 15bpp for storing a 1024×1024, 10-bit image. The key idea for the is that the order in which the encoder outputs *r* allows the decoder to match the coefficient $c_{i,j}$ in the LSP$_r$ list to its position (i, j) in the image.

Method 1 reads each coefficient sequentially; the order is not important, as long as the decoder algorithm shares the same order. This allows for partial parallelization of the process. The algorithm can read and process coefficients in parallel, but moves them sequentially to the LSP lists. Suppose I implement eight parallel processes and assign a priority order to each of these eight processes. The hardware handles simultaneous entries to

a list by queuing the entries to the same LSP list based on the priority order. Hence it is a partial parallel process. These modifications overcome the need to check many coefficients more than once and provides for partial parallelization of the process.

The list sizes in Method 1, however, are uncertain. This prevents predetermination of required memory resources of the *n* lists. Further, the need to transmit threshold values of each coefficient results in a coefficient dependent overhead that can be less than or more than the overhead produced in SPIHT. Table 7.1 shows results from Matlab simulations using Method 1 and SPIHT on 1024×1024, 8-bit ZOOVIS and Lena images.

TABLE 7.1 bpp comparison with Method 1.

	ZOOVIS	Lena
SPIHT	6.54bpp	3.80bpp
Method 1	6.59bpp	5.73bpp

Table 7.1 does not indicate any improvement using Method 1 on the image from the ZOOVIS camera, and it shows a degradation of 1.93bpp for the Lena image. Further, this method handless more lists and hence computes more slowly than SPIHT. Further, the significant bits for all entries in the LSP lists are transmitted in the end.

7.2.2 Adjacent Coefficient Method (Method 2)

A close look into Method 1 brings out that in reality for every image pixel the output is a threshold value, a sign bit, and least significant bits (LSB) equivalent in number to one less than the threshold value. The computational complexity of Method 1 is the creation of lists, and its hardware constraint is undetermined memory requirements. Hence, modifying Method 1 by eliminating list sorting and avoiding outputting a threshold value for each coefficient leads to Method 2.

Method 2 pairs two adjacent coefficients and outputs *r*, the number of significant bits of the larger of the two coefficient magnitudes, followed by *r* LSBs of each of the two coefficients. The exception is that if $r = 0$, then no output bits are generated. Figure 7.3 describes Method 2, and Figure 7.4 represents the output bit-ordering for Method 2. The header in Figure 7.4 refers to the four bits allocated to output the value of *r*, where $0 \le r \le 10$ for a 10-bit image.

1. Enter two adjacent coefficients $c_{i,j}$ and $c_{i,j+1}$.
2. **Output** sign of each coefficient.
3. If $c_{i,j} \neq 0$ and $c_{i,j+1} \neq 0$, then **output** $r = 1 + \left\lfloor \log_2[\max\{|c_{i,j}|, |c_{i,j+1}|\}] \right\rfloor$, else **output** $r = 0$.
4. If $r > 0$, **output** r LSBs of $c_{i,j}$ and $c_{i,j+1}$, where $1 \leq r \leq n$, else output no bits.
5. Enter next set of adjacent coefficients and go to Step 2. Repeat process until end of image.

FIGURE 7.3 Method 2 Bit-Ordering Algorithm.

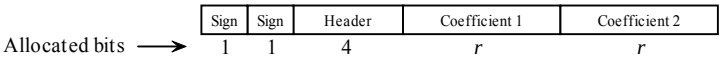


FIGURE 7.4 Output bit-ordering of Method 2.

Method 2 outputs the leading '1' bit of each coefficient. Hence, each coefficient outputs at least r bits of each coefficient compared to $r-1$ bits in Method 1. Further, let r_1 and r_2 denote the number of significant bits in $|c_{i,j}|$ and $|c_{i,j+1}|$, respectively, and assume $r_1 \geq r_2$ without loss of generality, then $r = r_1$, and the algorithm outputs $r_1 - r_2$ extra bits of $c_{i,j+1}$ that Method 1 does not output. Method 2, therefore, outputs fewer header bits, but more coefficient bits than Method 1.

Method 2 requires a minimum of 3bpp and a maximum of 13bpp for storing a 1024×1024, 10-bit image. The decoder algorithm is straight forward.

The advantage of this method over Method 1 is the reduction by up to half in the required header per coefficient. Unlike Method 1, it is possible to estimate the required memory resources for hardware implementation of Method 2. Hence, this method is comparatively much simpler to implement in hardware. Table 7.2 shows results of Matlab simulations comparing SPIHT, Method 1, and Method 2 on 1024×1024, 8-bit ZOOVIS and Lena images.

TABLE 7.2 bpp comparison with Method 2.

	ZOOVIS	Lena
SPIHT	6.54bpp	3.80bpp
Method 1	6.59bpp	5.73bpp
Method 2	6.15bpp	4.86bpp

Table 7.2 indicates a 0.44bpp decrease over Method 1 for the ZOOVIS image. The fall in bpp is less than the potential 2bpp (because of the reduction in header bits) when the coefficients in a pair have different number of significant bits, and the two coefficients output extra '0' bits before the leading '1' to keep up with the common *r* value. Matlab simulations have shown that this method computes faster than Method 1, because of the non-existence of the lists. Therefore, it is better suited for images from the ZOOVIS-SC system compared to SPIHT and Method 1.

7.2.3 4-Cluster Method (Method 3)

In the modifications suggested to SPIHT [25], the first modification explains the significance of joint coding of a 2×2 block of coefficients. Comparing Method 1 to Method 2, the improvement was one *r* value for a pair of coefficients instead of one for each coefficient. Extending this idea of combining the *r* values to the joint coding of a 2×2 block, I now group four coefficients to output one *r* value. The exception of outputting no LSBs of '0' threshold value still holds.

Similar to Method 2, the bpp impact from the header reduces further. The need to output a few more '0' bits before the leading '1' to keep up with the common *r* value increases, because the method now outputs four coefficients at the length of the longest coefficient. Figure 7.5 describes the Method 3 algorithm, and Figure 7.6 shows the output bit order, where *r* is the number of significant bits of the maximum coefficient.

1. Enter four coefficients in a 2×2 block: $c_{i,j}$, $c_{i,j+1}$, $c_{i+1,j}$, and $c_{i+1,j+1}$.
2. **Output** sign of each coefficient.
3. If $c_{i,j} \neq 0$, $c_{i,j+1} \neq 0$, $c_{i+1,j} \neq 0$ and $c_{i+1,j+1} \neq 0$, then **output** $r = 1 + \left\lfloor \log_2[\max\{|c_{i,j}|, |c_{i,j+1}|, |c_{i+1,j}|, |c_{i+1,j+1}|\}] \right\rfloor$, else **output** $r = 0$.

FIGURE 7.5 4-Cluster Bit-Ordering Algorithm.
(Figure Continued)

4. If $r > 0$, **output** r LSBs of $c_{i,j}$, $c_{i,j+1}$, $c_{i+1,j}$, and $c_{i+1,j+1}$, where $1 \le r \le n$, else output no bits.
5. Enter next set of 2×2 block coefficients and go to Step 2. Repeat process until end of image.

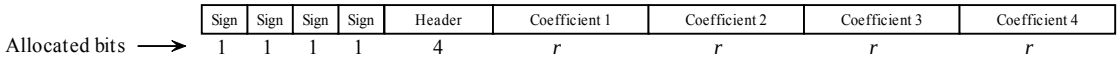


FIGURE 7.6 Output bit-ordering of 4-Cluster Method.

Method 3 requires a minimum of 2bpp and maximum of 12bpp for storing a 1024×1024, 10-bit image. The decoder algorithm is straight forward.

In addition to the advantages provided by Method 2, this method reduces the bit requirement for the header per coefficient to one bit. Table 7.3 shows results of Matlab simulations comparing SPIHT, Method 1, Method 2, and Method 3 on 1024×1024, 8-bit ZOOVIS and Lena images.

TABLE 7.3 bpp comparison with 4-Cluster method.

	ZOOVIS	**Lena**
SPIHT	6.54bpp	3.80bpp
Method 1	6.59bpp	5.73bpp
Method 2	6.15bpp	4.86bpp
Method 3	5.62bpp	4.26bpp

Table 7.3 indicates a 0.53bpp drop from Method 2 for the ZOOVIS image. Comparing Method 3 to SPIHT, there is a drop of 0.92bpp for the ZOOVIS image while dealing with all constraints of SPIHT. In comparing performance on image Lena, Method 3 gives the closest value to SPIHT, but SPIHT gives a better compression ratio.

Comparing all the above bit-ordering methods, the 4-Cluster method has much less computational complexity compared to SPIHT and gives better compression ratios than the rest of the methods. Therefore, for the bit-ordering phase, I implemented the 4-Cluster method in ZOOVIS-SC.

8

IMAGE COMPRESSION ARCHITECTURE

This chapter discusses the architectural implementation and resource requirements by the two phases of the lossless image compression algorithm. Section 8.1 discusses the hardware implementation of the multi-resolution transform and bit ordering phases, while Section 8.2 outlines the synthesized Verilog modules, device utilization summary, resources required, and suitable FPGAs.

8.1 Design Overview

Two phases compose the image compression algorithm. The multi-resolution transform phase computes using the S+P transform, and the bit-ordering phase transmits ordered bits from the transformed coefficients using the 4-Cluster method. To differentiate between the hardware and software implementations, I refer to the multi-resolution transform phase as the SP phase, and the bit-ordering phase as the Order phase. Correspondingly, SP module and Order module refer to the Verilog implementation of the image compression algorithm.

Figure 8.1 illustrates the complete architecture of the image compression algorithm, depicting the two phases and their access to memory units and a buffer through the memory controller.

I organize the Sections 8.1.1 and 8.1.2 as follows: First I discuss the general hardware implementation of the algorithm followed by its limitations when implemented on the FPGA and parallel implementation of the algorithm. Second, I present the block diagram of the architecture. An explanation follows the block diagram and discussion closes with the clocks required for the hardware implementation of each phase.

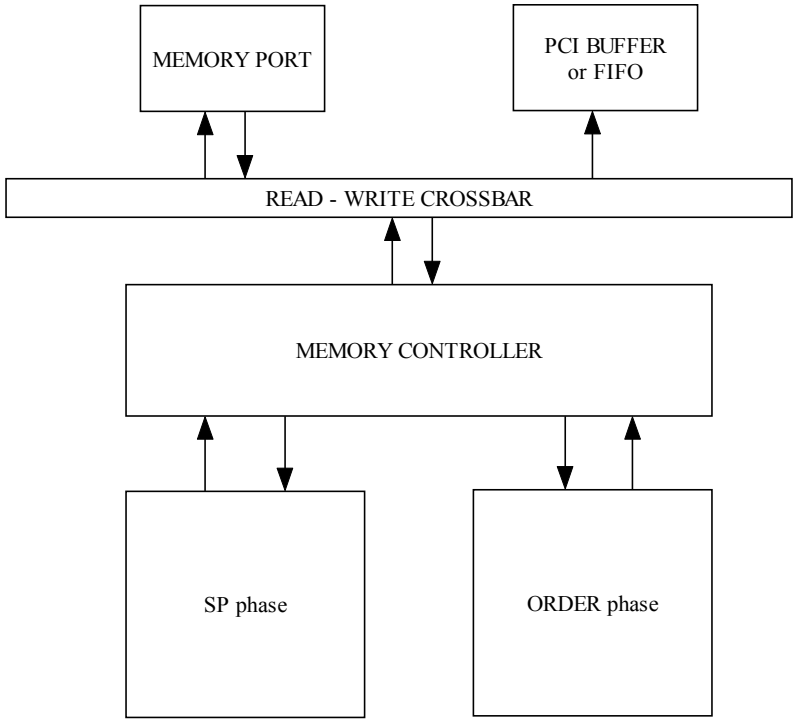


FIGURE 8.1 Image compression architecture.

8.1.1 SP Phase

Fry [34] modified the folded DWT architecture proposed by Parhi and Nishitani [29] and implemented a seven-level DWT, such that the architecture uses the same addressing logic for horizontal and vertical passes. Utilizing the same addressing logic for both the horizontal and vertical passes avoids the requirement of additional resources for each pass. This architecture reads coefficients in rows from one memory unit and writes coefficients in columns to a second memory unit. By the end of the horizontal pass, the image rotates along a 45º diagonal axis. Utilizing the same addressing logic, the vertical pass reads the resultant coefficients in rows from the second memory unit and writes in columns to the first memory unit. However, since the image rotates again along a 45º diagonal axis in the vertical pass, the image restores to its original orientation. The size of the set of coefficients being processed reduces by half from one level to the next level, that is, after a set of horizontal and vertical passes, and the process iteratively continues for each level up to the seventh level.

Assuming that most FPGAs have a 64-bit memory access port, the FPGA can read a maximum of four 16-bit coefficients per clock cycle. To get maximum utilization of the limited width of the memory ports, the parallel architecture reads in four coefficients in a

clock cycle. If writes to the memory used the same memory ports, then writing the four processed coefficients takes place in the next clock cycle. If the FPGA has dual memory ports, then read and write of data can simultaneously take place in the same clock cycle. In this case, the SP phase architecture reads data in one clock cycle and writes in the next clock cycle, overlapping with the next read. Control logic handles switching between memory ports at the end of each pass.

Discussed in Section 6.1.1, the sequential coding part of the S+P transform is a set of simple computations involving an adder, a subtractor, and a shifter that require the current set of coefficients as inputs. The predictive coding part of the S+P transform however, requires previous low (l) and high (h) subband values, in addition to the current low and high subband values to compute predicted high (h_d) subband values. Therefore, intermediate registers hold the current set of low and high subband values computed by the sequential coding part, and the next clock cycle uses the values to compute the next set of high subband values. Hence, for the current set of coefficients, the current cycle computes the low subband values, and the next clock cycle computes the predicted high subband values. Because of the delay in the calculation of predicted high subband values, each write contains low subband values of the current set of input coefficients and high subband values of the previous set of input coefficients. Figure 8.2 illustrates the architecture of the SP phase, where for an $N \times N$ size image, $n = 0,...,N/2-1$.

The read address logic block generates the addresses for the current set of coefficients. Utilizing the generated addresses, the memory ports read the current set of coefficients from memory. Intermediate storage registers block store the calculated low and high subband values in sequential coding. As discussed above, predictive coding uses these stored values in the next clock cycle. Edge condition blocks decide the logic used for computation of predictive high subband values $\hat{h}$. The write address logic block uses the boundary conditions to generate the write addresses and writes computed low and high subband values to memory through the write memory ports. During the horizontal pass, memory port 1 reads the data and memory port 2 writes the data, while for the vertical pass memory port 2 reads the data and memory port 1 writes the data.

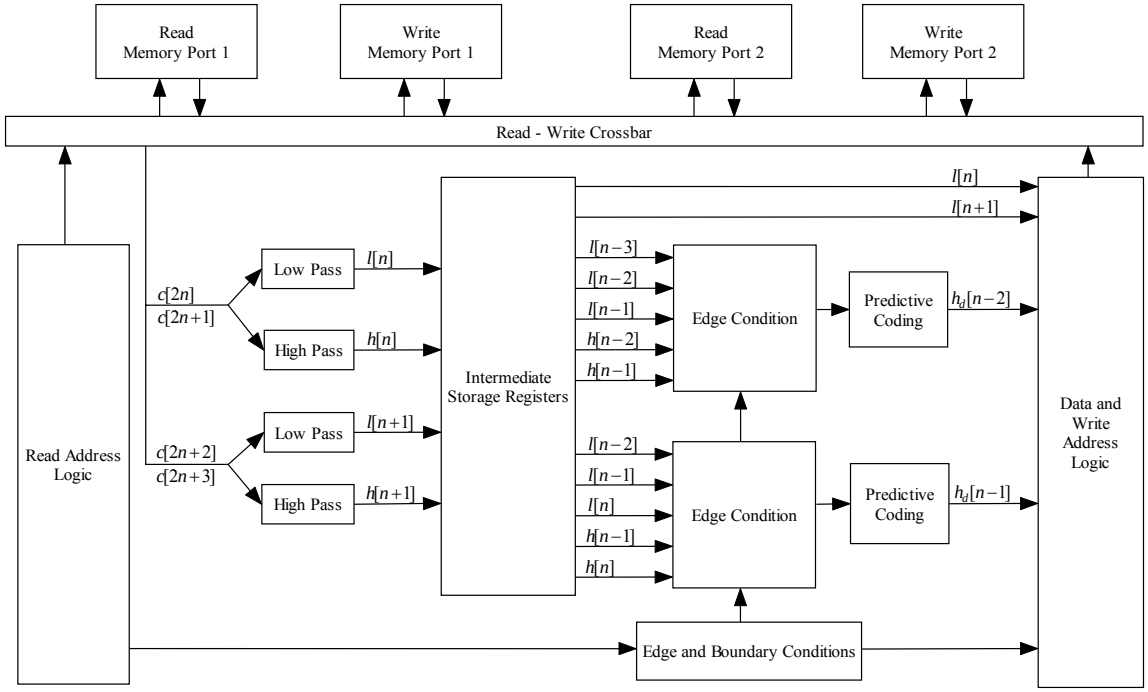


FIGURE 8.2 S+P transform architecture.

From the S+P transform implementation in Figure 8.2, for an $N \times N$ size image, computing four coefficients in parallel would require ¼ N^2 clock cycles. Each level has a horizontal and a vertical pass, requiring a total of ½ N^2 clock cycles to complete one level. Further, each higher level is ¼ the size of the previous level. Therefore, to compute an infinite number of levels, where the read and write overlap in the same clock cycle, the image processes in:

$$\sum_{l=1}^{\infty} \frac{N^2}{2*4^{(l-1)}} = \frac{2}{3} N^2$$

clock cycles. If data was read in one clock cycle and written in the next clock cycle without any overlapping, then the required number of clock cycles doubles to $\frac{4}{3} N^2$.

8.1.2 Order Phase

I proposed the 4-Cluster method implemented in this phase keeping in view its hardware implementation. The four coefficients required by the 4-Cluster method (as discussed in Section 7.2.3) would be a 2×2 block of adjacent coefficients. The architecture requires two pairs of non-continuous addresses to read data from memory and logic blocks in the hardware implementation generates them. Suppose *r* is the number of significant bits of the threshold, then the hardware allocates *r* bits to each of the four coefficients and then fills them with

significant bits from the corresponding coefficients. The hardware implementation allows a maximum bit stream of 64-bits for a set of four coefficients which includes four sign bits and four bits for header.

Once the SP phase completes, the Order phase starts. Based on the assumption in Section 8.1.1, the FPGA can read up to four 16-bit coefficients at a time. Figure 8.3 illustrates the 4-Cluster bit-ordering architecture.

Similar to the SP phase architecture, the read address logic block generates the read addresses for the current set of four coefficients. Based on the algorithm, the hardware computes the sign bit and absolute threshold value of the four coefficients. The bit stream generator block arranges the extracted bits in 32-bit arrays and buffers to the PCI buffer or First in, first out (FIFO). The output bits arranged in a 32-bit FIFO buffer suit the PC/104-*Plus* form factor of a 32-bit wide PCI-bus.

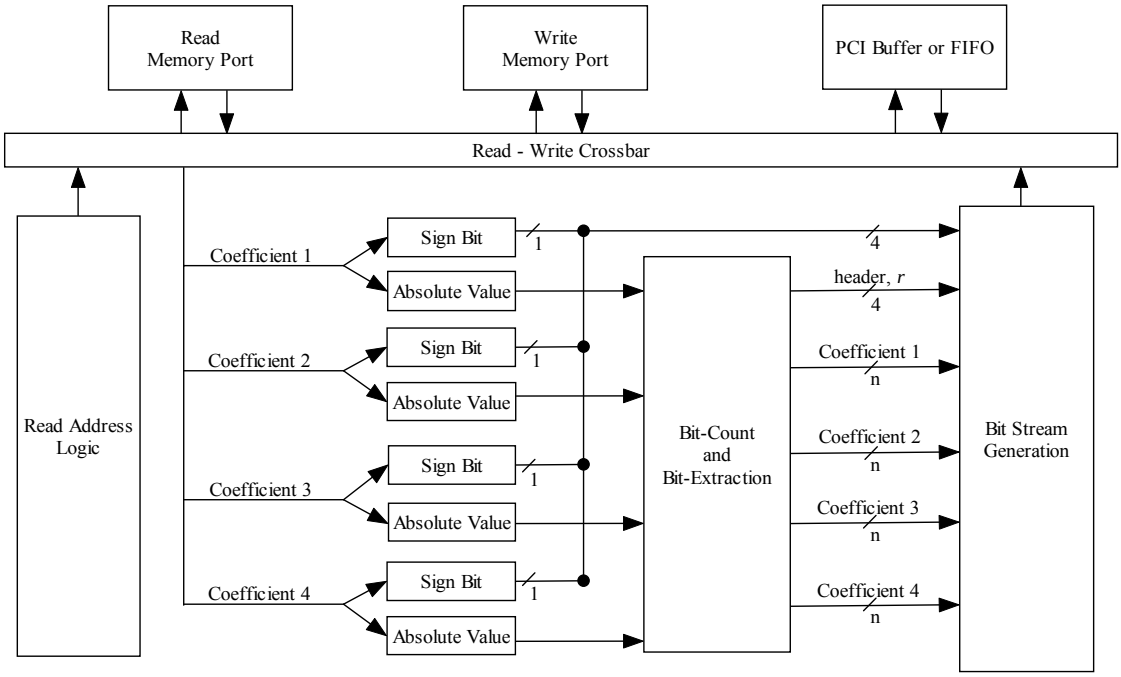


FIGURE 8.3 Order architecture.

From the Order implementation in Figure 8.3, for an $N \times N$ size image, if the architecture reads four coefficients in one clock cycle and writes in the next clock cycle overlapping with the next read, it would take ¼ N^2 clock cycles to compute the image, and if the write does not overlap with the next read, it would take ½ N^2 clock cycles to compute.

8.2 Verilog Synthesis and FPGA Requirements

This section presents the outline model of the Verilog modules and discusses the resources required by the various phases and suitable FPGAs on which the logic implements. I implemented the various modules for the compression algorithm in Verilog HDL and simulated those using ncVerilog, Build Gates, ModelSim, and Xilinx ISE tools. The design process used a UNIX platform where I used ncVerilog to debug and Ambit Build Gates to synthesize the Verilog code. I generated the device utilization summary (Section 8.2.2) and HDL synthesis report (Section 8.2.3) on a Windows platform using Xilinx ISE Foundation Series 6.1i. In an effort to generate the synthesis report, I used ModelSim for further debugging.

8.2.1 Verilog Model

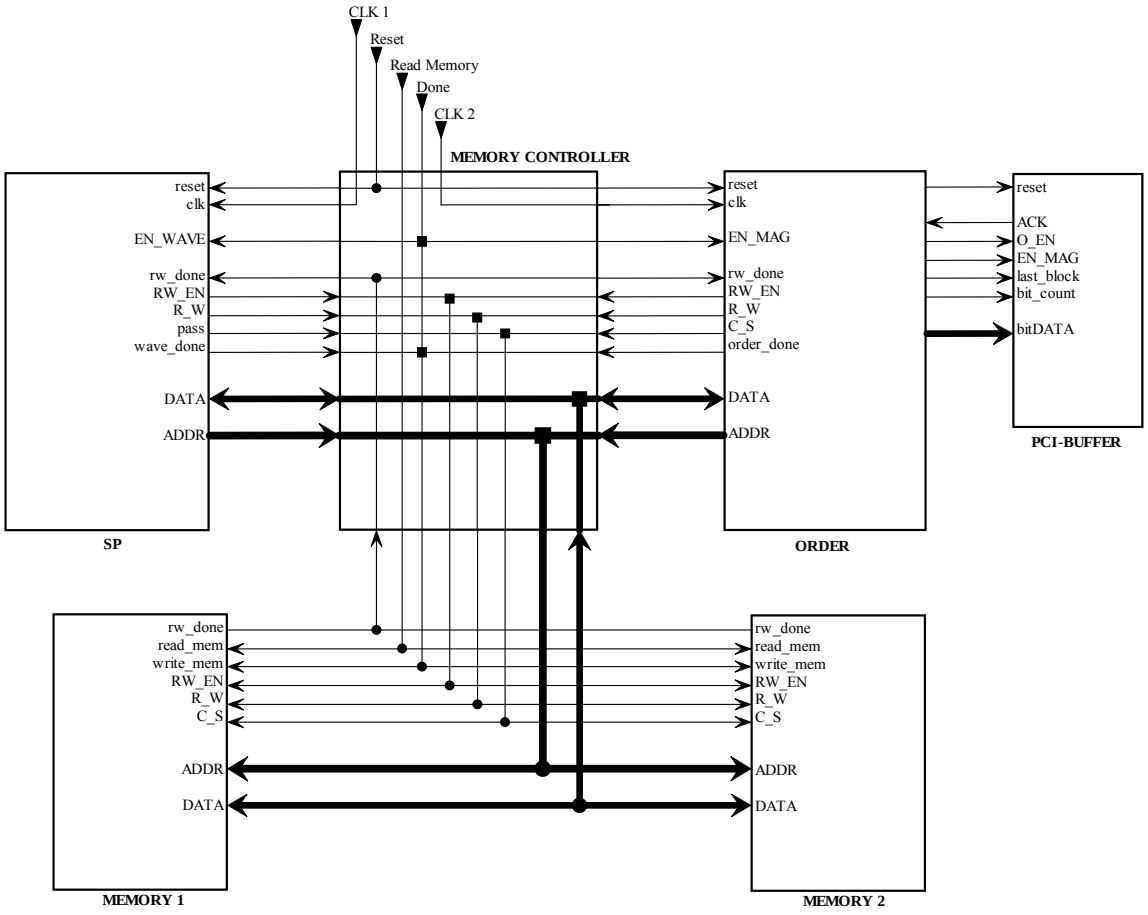


FIGURE 8.4 Verilog model of compression logic.

The SP and Order modules in Figure 8.4 represent the logic implemented in the SP and Order phases, respectively, in the image compression algorithm. The Memory controller controls both these modules, selecting the phase to run and memory unit to be read or written based on handshaking signals provided by the SP and Order modules. The Memory controller allows the SP and Order modules to run on different clock frequencies. The signals connected by solid square patches are selective and have control logic implemented, whereas the signals connected with solid circular patches are directly connected. The Memory 1, Memory 2, and PCI_buffer modules in Figure 8.4 are not synthesizable and are only created to simulate real-time functioning.

8.2.2 Device Utilization Summary

Simulating the Memory controller, SP, and Order modules separately on Xilinx ISE 6.1i, Table 8.1 presents the resultant device utilization summary.

TABLE 8.1 Device utilization summary.

#	Memory controller	SP	Order
Slices	23	280	369
Slice Flip flops	5	266	82
4 input LUTs	40	421	578
IOBs	303	101	175
TBUFs	-	158	86
GCLKs	1	3	2

Note that one can ignore the IOB requirements for the SP and Order modules, since they communicate through the memory controller modules to any external unit outside the FPGA. Further, even though simulation results show 303 IOBs for the Memory controller module, the SP and Order modules require 101 connections each to communicate with the Memory controller module. Therefore, the Memory controller module effectively requires only 101 IOBs to communicate to hardware units outside the FPGA. Further, I only require two separate global clocks, one each for SP and Order module, since the clocks presented in Table 8.1 include edge triggered conditions in the Verilog code that do not run on global clocks. Similarly, I neglect the edge triggered condition in the Memory controller module. Hence, the set of additional clocks presented in Table 8.1 are actually signals generated by

other logic cells that run on global clocks. Table 8.2 summarizes the total resource requirements of the FPGA.

TABLE 8.2 FPGA device utilization requirements.

#	Resources required
Slices	672
IOBs	101
GCLKs	2

8.2.3 Synthesis Report

The HDL synthesis report from Xilinx ISE 6.1i gives a maximum frequency of 70.541MHz for the SP module and 161.186MHz for the Order module. Table 8.3 summarizes the resource requirements for the logic implemented in the Memory controller, SP, and Order modules.

TABLE 8.3 Total resource requirement for the compression logic.

HDL Synthesis Report	mem_ctrl.v	sp.v	order.v
Registers	3	38	9
Latches	-	19	2
Multiplexers	8	6	35
Tristate Buffers	12	26	76
Logic shifters	-	2	-
Adders/Subtractors	-	29	38
Comparators	-	9	17

8.2.4 Suitable FPGA

The Spartan-IIE 1.8V Field-Programmable Gate Array family gives high performance, abundant logic resources as high as 15,552 logic cells with up to 600,000 system gates, and it support system performances beyond 200 MHz [35]. Further, it is fully 3.3V PCI compliant up to 64 bits at 66 MHz. Assuming that the Spartan-IIE series FPGAs from Xilinx would suit our requirements, in this section I find the smallest Spartan-IIE series FPGA that satisfies the slice, IOB, and GCLK requirements discussed in Section 8.2.2.

A Spartan-IIE series FPGA has two 16-bit, 4-input LUTs, and two flip-flops in a slice. It has two slices and two TBUFs in a CLB. Figure 8.5 presents the Spartan-IIE CLB slice.

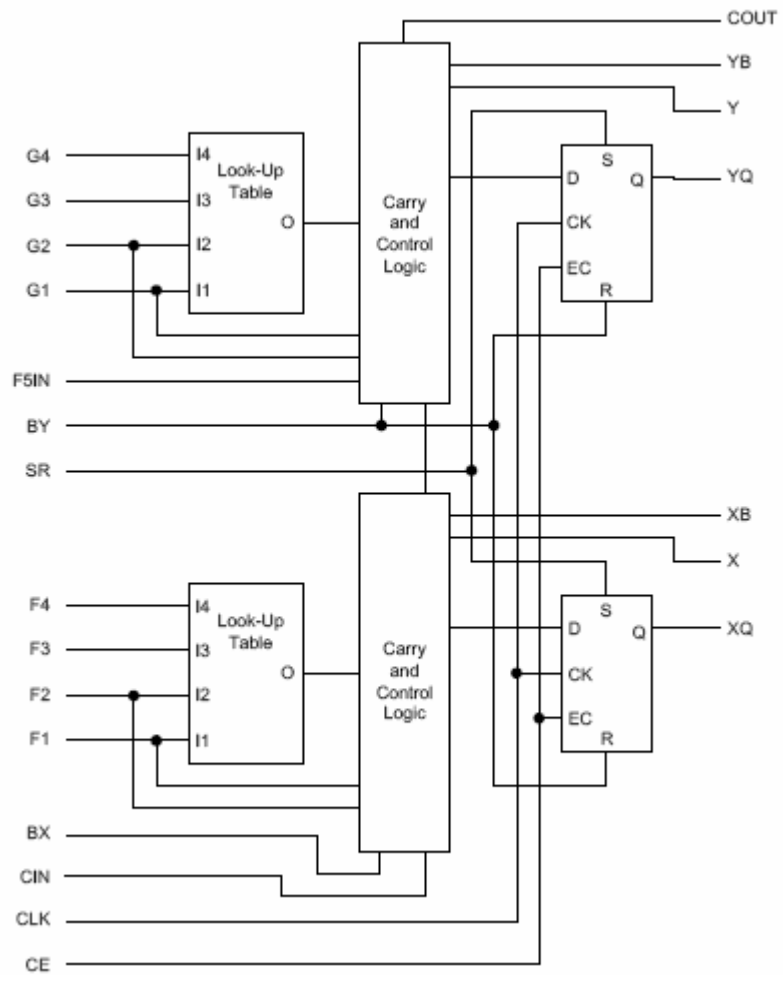


FIGURE 8.5 Spartan-IIE CLB slice (two identical slices in each CLB) [35].

Table 8.2 gives a total requirement of 672 slices. Therefore the Verilog modules require 336 CLBs in an FPGA. The FPGA also requires 101 IOBs and 2 GCLKs. The XC2S50E FPGA of the Spartan-IIE series has 384 CLBs, 182 IOBs, and 4 GCLKs and, hence, it suits the minimum requirements of the FPGA. It is preferred to select an FPGA with 50% utilization for better performance and future modifications, however. Therefore, the XC2S100E FPGA, the next in line in the Spartan-IIE series with 600 CLBs, 202 IOBs, and 4 GCLKs suits a practical implementation of the image compression algorithm.

9

IMAGE COMPRESSION: PERFORMANCE RESULTS

In Section 8.2 I discussed the design of the image compression scheme using Verilog HDL. In Sections 8.2.2 and 8.2.3 I presented the synthesis and timing analysis for the design using the Xilinx ISE Foundation Series 6.1i tool set.

This chapter discusses the timing analysis and compression results for the implementation of image compression on an FPGA where Sections 9.1 and 9.2 discuss the speed and runtime implementations of the image compression architecture presented in Chapter 8. In the performance results presented, each phase has a separate clock requirement, since I assume each phase computes on different logic blocks. The design can process any square image where the length is a power of 2: 8×8, 16×16, etc., with a dynamic range up to 14bpp. Further, I optimize the design to meet the specifications of the ZOOVIS-SC camera.

The ZOOVIS-SC camera takes 1024×1024, 10-bit, grayscale images. Because of the inherent requirements by the framegrabber module for data transfer, however, the ZOOVIS-SC system stores 16bpp images. The framegrabber pads six extra bits to each pixel to meet the interface and storage standards of 8, 16, or 32 bits. The framegrabber fills these six extra bits with 0's at the MSB positions such that they do not affect the intensity values of the image. Hence, the resultant image is a 1024×1024, 16-bit image and this is considered as input to the two phases of the image compression algorithm. I calculated the latencies presented in the following sections accordingly.

9.1 Simulation 1

The simulation results discussed in this section assume that the FPGA supports a single 64-bit access to memory ports. The read and write cycles, consequently, are different and alternate. Table 9.1 presents the results.

TABLE 9.1 Performance based on Simulation 1.

Phase	Clock Cycles per 1024×1024 image	Clock Cycles per Pixel	Clock Rate	Throughput	Latency
SP	1,398,101	4/3	70MHz	52.5MBps	~ 38ms
Order	524,288	½	160MHz	320MBps	~ 6ms

I observe the Order phase yields higher throughputs than the SP phase. This is because the ordering phase requires fewer clock cycles to compute the image. The Order phase takes half a clock cycle per pixel, whereas the SP phase takes more than one clock cycle per pixel. The SP phase constrains the throughput of the system at 52.5MBps. Since the two phases can operate at different clock frequencies, however, the total latency for processing the image is approximately 44ms which gives a compression speed of 22.7MPixels/sec.

9.2 Simulation 2

The simulation results discussed in this section assume that the FPGA supports dual 64-bit access to memory ports. If the FPGA has dual memory ports, then read and write of data can simultaneously take place in the same clock cycle. Therefore, the two phases read data in one clock cycle and write computed results from that data in the next clock cycle, overlapping with the next read. Table 9.2 presents the results.

TABLE 9.2 Performance based on Simulation 2.

Phase	Clock Cycles per 1024×1024 image	Clock Cycles per Pixel	Clock Rate	Throughput	Latency
SP	699,050	2/3	70MHz	105MBps	~ 19ms
Order	262,144	1/4	160MHz	640MBps	~ 3ms

The SP phase constrains the throughput of the system at 105MBps. Since the two phases can operate at different clock frequencies, however, the total latency for processing the image is

approximately 22ms which gives a compression speed of 45.5MPixels/sec. Further, each phase of the computation consumes less than one clock cycle per pixel.

9.3 Conclusion

The Spartan-E FPGA family from Xilinx has 64-bit, dual port memory blocks and hence can access two different memory blocks simultaneously. If the compression is implemented on the Spartan-IIE FPGA as presented in Section 8.2.4, then the total latency for compression would be approximately 22ms, following Simulation 2. The throughput for the computation process improves if the available memory ports are 128-bits instead of 64-bits. If so, the latency for the image computation will be reduced by half in each case.

The image compression algorithm, which includes SP and Order phases, achieves a compression ratio of 1.67:1 when run on various images taken in test conditions on targets that closely resemble zooplankton, however, the degree of compression is dependent upon the complexity of the image. Therefore in regions of high animal abundance, the images will likely have the greatest complexity and therefore the compression ratio can diminish, however, the use of S+P transform on the image prior to bit-ordering curbs such discrepancies to a certain extent. The original image from the ZOOVIS-SC camera was size 1024×1024, and 10-bits, while the average of the compressed image files was 0.75MB that is 6bpp.

The results discussed in this chapter do not take into consideration the delay for the data to arrive for processing and the delay for the processed data to be stored to disk. It only discusses the latency and throughput of the image compression process in the FPGA. Chapter 10 presents the complete performance results.

PART – III

CONCLUSION

10

IMPLEMENTING IMAGE COMPRESSION IN ZOOVIS-SC

Chapter 5 discusses the need for image compression to improve the ZOOVIS-SC system's frame rate, and Chapter 9 discusses the latency for the implementation of the image compression scheme and the compression ratio achieved. In this chapter, I discuss the improvements that can be achieved by implementing the compression scheme in ZOOVIS-SC. Section 10.1 discusses possible changes in the hardware configuration of ZOOVIS-SC for implementation of the compression scheme, and Section 10.2 discusses possible pipelined approaches by which the same compression scheme can be implemented to improve the performance of the ZOOVIS-SC system.

10.1 Possible Architectures to Integrate an FPGA in ZOOVIS-SC

I must implement the compression scheme on an FPGA based on PC/104-*Plus* standards to be compatible with the existing ZOOVIS-SC architecture. The following sections discuss two possible options for interfacing an FPGA module into the existing stack.

From Section 4.2, I know that the ZOOVIS-SC system captures 10bpp images, but transfers and stores 16bpp images. Further, if the image is compressed as proposed, from Section 9.3, the average data transferred after compression is 6bpp. I therefore calculated the latencies presented in Sections 10.1.1 and 10.1.2 corresponding to the amount of data transferred through each interface.

10.1.1 Separate PC/104-*Plus* FPGA Module

On the basis of the ZOOVIS-SC design, an obvious solution is to get a separate FPGA module based on PC/104-*Plus* standards and stack it onto the existing hardware. The PCI-bus

treats it as the third module, in addition to the framegrabber and host modules. Figure 10.1 presents the data transfer path with estimated latencies of ZOOVIS-SC with FPGA module interfaced. The estimated latencies presented correspond to the amount of data transferred on each interface, on the basis of discussions in Sections 5.1.2 and 9.3.

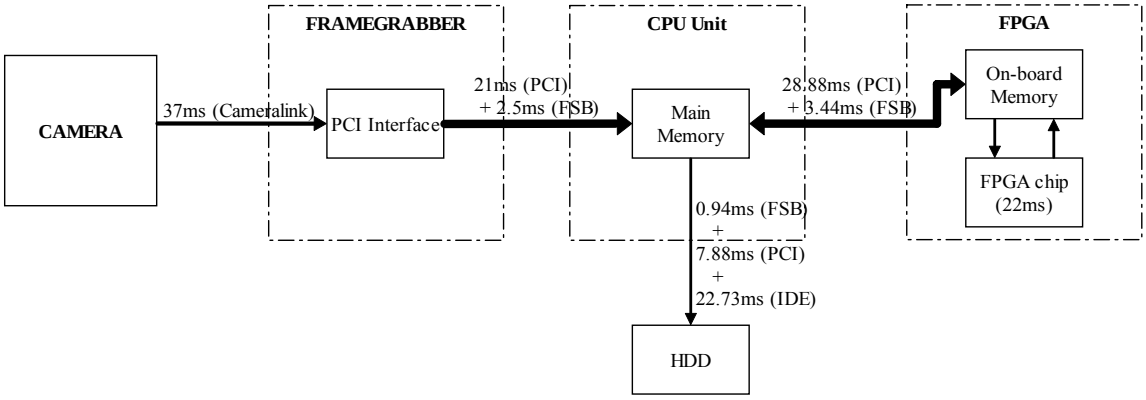


FIGURE 10.1 Data transfer path with separate PC/104-*Plus* FPGA module.

In this architecture, the camera transfers data via framegrabber on the PCI-bus and temporarily stores data in the main memory. The FPGA module reads data from the main memory, processes it, and then transfers the compressed data back to main memory. The HDD stores the compressed data from main memory. The data path between main memory and FPGA module is bidirectional since the same set of buses transfer original data to the FPGA module and compressed data from the FPGA module back to main memory. Table 10.1 summarizes the estimated latencies on each interface for the data transfer path presented in Figure 10.1.

TABLE 10.1 Latency with image compression on separate FPGA module.

Interface	Latency
Cameralink	37ms
Compression	22ms
PCI-bus	57.76ms
FSB	6.88ms
IDE cable	22.73ms

As discussed in Section 5.1.2, the existing ZOOVIS-SC architecture requires 108MBps bandwidth. With the addition of the FGPA module, the PCI-bus requires an additional

bandwidth of 54MBps to transfer original data to the FPGA module and 20.25MBps to transfer back compressed data from the FPGA module. The estimated bandwidth of the PCI-bus is only 95MBps. So, implementing the compression scheme on a separate FPGA module would adversely affect the latency of the ZOOVIS-SC system. The total estimated latency in this architecture from Table 10.1 is 146.37ms, which gives an estimated frame rate of 6.83fps, slower than the current estimated frame rate of 6.92fps without compression derived in Section 5.1.2. Therefore, the disadvantage with this architecture is the excessive usage of the PCI-bus by the various modules.

10.1.2 Framegrabber with Built-In FPGA

An alternative to the above method is to include a user-configurable FPGA on the framegrabber board. Having the image compression on the framegrabber module would reduce the bandwidth requirements on the PCI-bus and also improve the latency of the system because of transmitting a compressed image on the PCI-bus rather than the original image. This method requires replacing the existing framegrabber module with a module that has a built-in user-configurable FGPA and on-board memory. Since the compression scheme would now be implemented in an FPGA chip on the framegrabber, the framegrabber requires on-board memory to hold the data temporarily until it is processed. Figure 10.2 presents the data transfer path with latencies for this architecture.

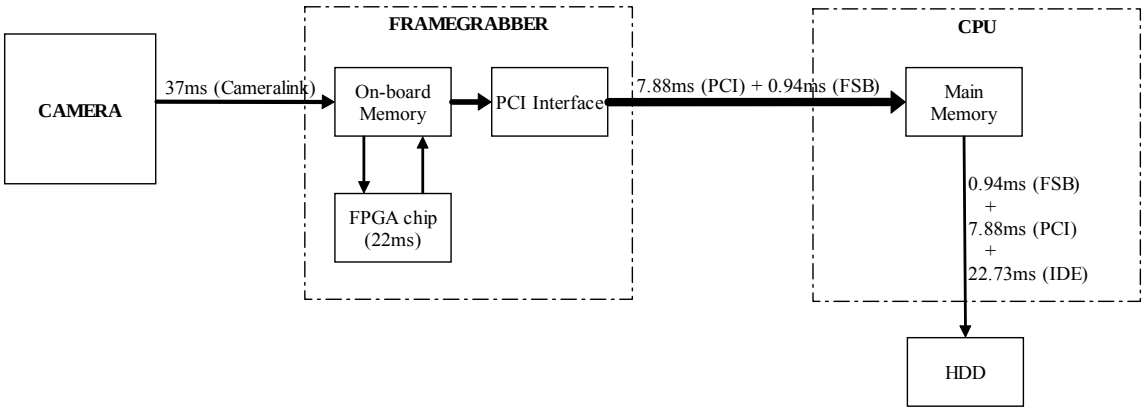


FIGURE 10.2 Data transfer path with FPGA chip built-in framegrabber module.

In this architecture, the camera writes the data into the on-board memory on the framegrabber module. The FPGA reads the image data, compresses it, and writes the compressed image back to the on-board memory. The framegrabber transfers the compressed data to the main memory on the PCI-bus. The HDD stores the compressed data from main memory. Table

10.2 summarizes the estimated latencies on each interface for the data transfer path presented in Figure 10.2.

TABLE 10.2 Latency with image compression on built-in FPGA on framegrabber module.

Interface	Latency
Cameralink	37ms
Compression	22ms
PCI-bus	15.76ms
FSB	1.88ms
IDE cable	22.73ms

From Table 10.2, the total latency when all transfers are done sequentially is 99.37ms. This gives a frame rate of 10.06fps which is 45% more than the current estimated frame rate of 6.92fps without compression derived in Section 5.1.2. Therefore, the compression scheme implemented on the basis of the data transfer path discussed in Figure 10.2 can improve the existing ZOOIVS-SC system.

10.2 Possible Approaches to Pipeline Data Transfers

From discussions in Section 10.1, the image compression implemented in a built-in FPGA chip on the framegrabber is the better of the two possible solutions. Section 10.1.2 considers sequential implementation of data transfers and gives an expected frame rate of 10.06fps. The compression scheme in correspondence with data transfers on various interfaces permits pipelining of the data transfer path. This could substantially bring down the latency. Pipelining the different phases of the compression scheme could further improve the latency of the system.

There are various possible approaches to pipeline the data transfer path. In this section I discuss three such possible pipeline approaches based on the data transfer architecture of ZOOVIS-SC discussed in Section 10.1.

10.2.1 Approach 1

In this approach, the data transfer path divides into two pipelined stages. The first stage comprises data transfer from camera to on-board memory on the framegrabber and the second stage comprises data transfer from on-board memory on the framegrabber to main

memory to HDD and includes data compression in the FPGA chip. Cameralink is the interface involved in the first stage and PCI-bus, FSB, and IDE cable are involved in the second stage. The latency of the first stage is 37ms and latency of the second stage (including compression) is 62.37ms. This method transfers one image in the first stage at the same time as processing the preceding image in the second stage and transferring it. The latency of this approach is 62.37ms, which is the latency of the critical stage. This gives a frame rate of 16.03fps. The implementation of this approach requires more on-board memory, approximately 8MB, and user control over the data flow in the framegrabber.

10.2.2 Approach 2

In this approach, the data transfer path divides into three pipelined stages with the compression scheme pipelined between the two transfer stages discussed in Section 10.2.1. Cameralink is the interface involved in the first stage, compression is done in the second stage, and PCI-bus, FSB, and IDE cable are involved in the third stage. The latency of the three stages are 37ms, 22ms, and 40.37ms respectively. The critical stage latency of this approach is 40.37ms, giving a frame rate of 24.76fps. This implementation requires significant on-board memory, however, and more user control over the vendor's off-the-shelf framegrabber module. In most cases the vendor does not give this capability to the user.

10.2.3 Approach 3

A third approach is to pipeline the two phases of the compression scheme in addition to the pipelined stages discussed in Section 10.2.2. Since the data compression is not the critical latency in Approach 2, pipelining the two phases of the compression scheme is not worthwhile. This approach proves useful for future consideration when data compression develops into the critical path because of reduction in latencies in the other transfer stage

10.3 Conclusion

The target is to design a low cost, lightweight, simple to use, non-invasive, internally-recording imaging system and as discussed in Section 5.1, I have successfully designed a working model of ZOOVIS-SC, the proposed underwater imaging system that meets almost all the requirements. I can enclose the components of ZOOVIS-SC, including its huge lens, into a 6" diameter and 24" long cylindrical pressure housing, and it runs on battery supply.

ZOOVIS-SC gives a practical working frame rate of 5.32fps, which is 19.7% of the maximum possible frame rate of 27fps with the current camera and 48.94% of the maximum possible theoretical frame rate with the current architecture of ZOOVIS-SC of 10.87fps as discussed in Section 5.1.1. Zooplankton layers are not abundant and whenever a body of these organisms is found it is essential to take as many images as possible. Therefore ZOOVIS-SC requires frame rates higher than 5.32fps. In addition, as discussed in Section 5.2 the camera has low responsitivity in low light and the framegrabber module does not give the specified full resolution images.

I expect to improve the frame rate to 10.06fps by implementing image compression on an FPGA chip integrated in ZOOVIS-SC as discussed in Section 10.1.2. At 10fps, the system would quantify a sampling volume of 9.6L/min. If I pipeline the stages as discussed in Section 10.2.2, the system would quantify a sampling volume of 23.8L/min at 24.76fps. Implementation of architecture discussed in Section 10.2.2 requires a new framegrabber module and this could take care of the problems with resolution with the existing framegrabber module.

The proposed compression scheme gives a compression ratio of 1.67:1 and a compression speed of 45.5MPixels/sec. However, the degree of compression is dependent upon the complexity of the image, which is the number of targets in the images. As discussed in Section 9.3, this is overcome to a certain extent by the S+P transform, a phase of the image compression algorithm.

10.4 Future Work

The immediate requirement to enhance ZOOVIS-SC is to find a new camera with better responsitivity and a new framegrabber module that suits the above discussed requirements. A solution of altogether avoiding the PCI-bus is using a custom designed framegrabber module with built-in disk controller. An alternative architecture to PC/104-*Plus* technology is Nano-ITX.

LITERATURE CITED

[1] M. C. Benfield, C. J. Schwehm, R. G. Fredericks, G. Squyres, S. F. Keenan, and M.V. Trevorrow, "Chapter 2: Measurement of Zooplankton Distributions with a High-Resolution Digital Camera System." In P. Strutton and L. Seuront (eds.) *Scales in Aquatic Ecology: Measurement, Analysis and Simulation.* CRC Press, 2003.

[2] A. Said and W. A. Pearlman, "An image multi-resolution representation for lossless and lossy compression," *IEEE Trans. Image Processing*, Vol. 5(9), pp. 1303-1310, Sept. 1996.

[3] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 6(3), pp. 243-250, June 1996.

[4] A. W. Herman, "Simultaneous measurement of zooplankton and light attendance with a new optical plankton counter," *Continental Shelf Res.*, Vol. 8, pp. 205-221, 1998.

[5] P. B. Ortner, L. C. Hill and H. E. Edgerton, "In-situ silhouette photography of oceanic zooplankton," *Deep-Sea Res.A*, Vol. 28, pp. 1569-1576, 1981.

[6] J. Lenz, D. Schnack, D. Peterson, J. Kreikemeiser, B. Hermann, S. Mees and K. Wieland, "The ichthyoplankton recorder: a video recording system for in-situ studies of small-scale distribution patterns. *ICES J. Mar. Sci.*, Vol. 52, pp. 409-417, 1994.

[7] C. S. Davis, S. M. Gallager, M. S. Berman, L. R. Haury and J. R. Strickler, "The Video Plankton Recorder (VPR): design and initial results," *Arch. Hydrobiology. Beih. Erg. Limnol.*, Vol. 36, pp. 67-81, 1992.

[8] P. Tiselius, "An *in situ* video camera for plankton studies: design and preliminary observations," *Mar. Ecol. Prog. Ser.*, Vol. 164, pp. 293-299, 1998.

[9] S. Samson, T. Hopkins, A. Remsen, L. Langebrake, T. Sutton and J. Pattern, " A System for high-resolution zooplankton imaging," *IEEE J. Oceanic Eng.*, Vol. 26, pp. 671-676, 2001.

[10] G. Gorsky, M. Picheral and L. Stemmann, "Use of the underwater video profiler for the study of aggregate dynamics in the North Mediterranean," *Estuarine Coastal Shelf Sci.*, Vol. 50, pp. 121-128, 2000.

[11] J. Watson, V. Chalvidan, J. P. Chambard, G. Craig, A. Diard, G. L. Foresti, B. Forre, S. Entili, P. R. Hobson, R. S. Lampitt, P. Maine, J. T. Malmo and G. Pieroni, "HOLOMAR: high resolution *in situ* HOLOgraphic recording and analysis of MARine organisms and particles," *Proceedings of the Third European Marine Science and Technology Conference (MAST Conference),* Libson, 23-27 May 1998, Vol. 3: *Generic Technologies*, pp. 1197-1211, 1998.

[12] E. Malkiel, O. Alquaddoomi and J. Katz, "Measurements of plankton distribution in the ocean using submersible holography," *Measurement Sci. Technol.,* Vol. 10, pp. 1142-1152, 1999.

[13] P. H. Wiebe and M. C. Benfield, "From the Hensen net toward four-dimensional biological oceanography," *Progress in Oceanography*, Vol. 56(1), pp. 7-136, Jan 2003.

[14] D. V. Holliday and R. E. Pieper, "Bioacoustical oceanography at high frequencies," *ICES Journal of Marine Science*, Vo. 52, pp. 279-296, 1995.

[15] PC/104 Consortium. *PC/104-Plus Specifications*, San Francisco, California, 2004. http://www.pc104.org/technology/PDF/PC104-Plus%20v2.0.pdf.

[16] Dalsa. *1M28-SA Camera User's Manual*, Ontario, Canada, 2004. http://vfm.dalsa.com/docs/manuals/1m28_75_150_user_manual_rev05.pdf?refr.

[17] Arvoo Imaging Products. *Picasso 104-CL User Manual,* Montfoort, The Netherlands, 2004. http://www.emjembedded.com/programs/arvoomanuals/userman104cl.pdf

[18] LiPPERT Automationstechnik GmbH. *Cool RoadRunner III Technical Manual*, Mannheim Germany, 2004. http://www.emjembedded.com/programs/lippertmanuals/crr3_r1v0_203.pdf

[19] Hitachi Global Storage Technologies. *Travelstar E7K60 Specifications*, San Jose, California, 2004. http://www.hitachigst.com/tech/techlib.nsf/techdocs/E18F32BF97E21A7786256D490063136D/$file/T7K60_sp(EA)3.0.pdf

[20] Tri-M Engineering. *HESC-104 Technical Manual*, Port Coquitlam, Canada, 2004. http://www.tri-m.com/products/engineering/files/manual/hesc104_man.pdf

[21] S. Hauck, "The role of FPGAs in Reprogrammable Systems," *Proceedings of the IEEE*, Vol.86(4), pp. 615-638, April 1998. http://www.ee.washington.edu/people/faculty/hauck/publications/mFPGAhard.pdf

[22] A. M. Al-Haj, "Fast Discrete Wavelet Transformation using FPGAs and Distributed Arithmetic," *Int. J. Appl. Sci. Eng.*, Vol. 1(2), pp. 160-171, 2003.

[23] A. Said and W. A. Pearlman, "SPIHT Image Compression, "Properties of the Method," http://www.cipr.rpi.edu/research/SPIHT/spiht1.html.

[24] R. P. Rao and W. P. Pearlman, "On Entropy of Pyramid Structures," *IEEE Trans. Inform. Theory,* Vol. 35, pp. 407-412, March 1991.

[25] U. Bayazit and W. A. Pearlman, "Algorithmic Modifications to SPIHT," *IEEE Int. Conf. on Image Processing (ICIP 2001)*, Thessaloniki, Greece. Oct. 2001.

[26] R. Seals and G. Whapshott, *Programmable Logic: PLDs and FPGAs*, UK: Macmillan, 1997.

[27] D. Mlynek, Y. Leblebici, *Design of VLSI Systems, "Chapter 1: Introduction to VLSI Systems*," http://vlsi.wpi.edu/webcourse/ch01/ch01.html.

[28] G. Knowles, "VLSI Architecture for the Discrete Wavelet Transform," *Electron Letters,* Vol. 26(15), pp. 1184-1185, 1990.

[29] K. K. Parhi and T. Nishitani, "VLSI Architectures for Discrete Wavelet Transforms," *IEEE Transactions on VLSI Systems*, Vol. 1(2), pp. 191-202, June 1993.

[30] C. Chakrabarti and M. Vishwanath, "Efficient Realizations of the Discrete and Continuous Wavelet Transforms: From Single Chip Implementations to Mappings on SIMD Array Computers," *IEEE Transactions on Signal Processing*, Vol. 43(3), pp. 759-771, 1995.

[31] J. Ritter and P. Molitor, "A Pipelined Architecture for Partitioned DWT Based Lossy Image Compression using FPGAs," *Proceedings of the Association for Computing Machinery (ACM-2001),* pp. 201-206, 2001.

[32] C. Chakrabarti, M. Vishwanath, and R. M. Owens. "A Survey of Architectures for the Discrete and Continuous Wavelet Transforms," *Journal of VLSI Signal Processing Systems*, pp. 171-192, Nov. 1996.

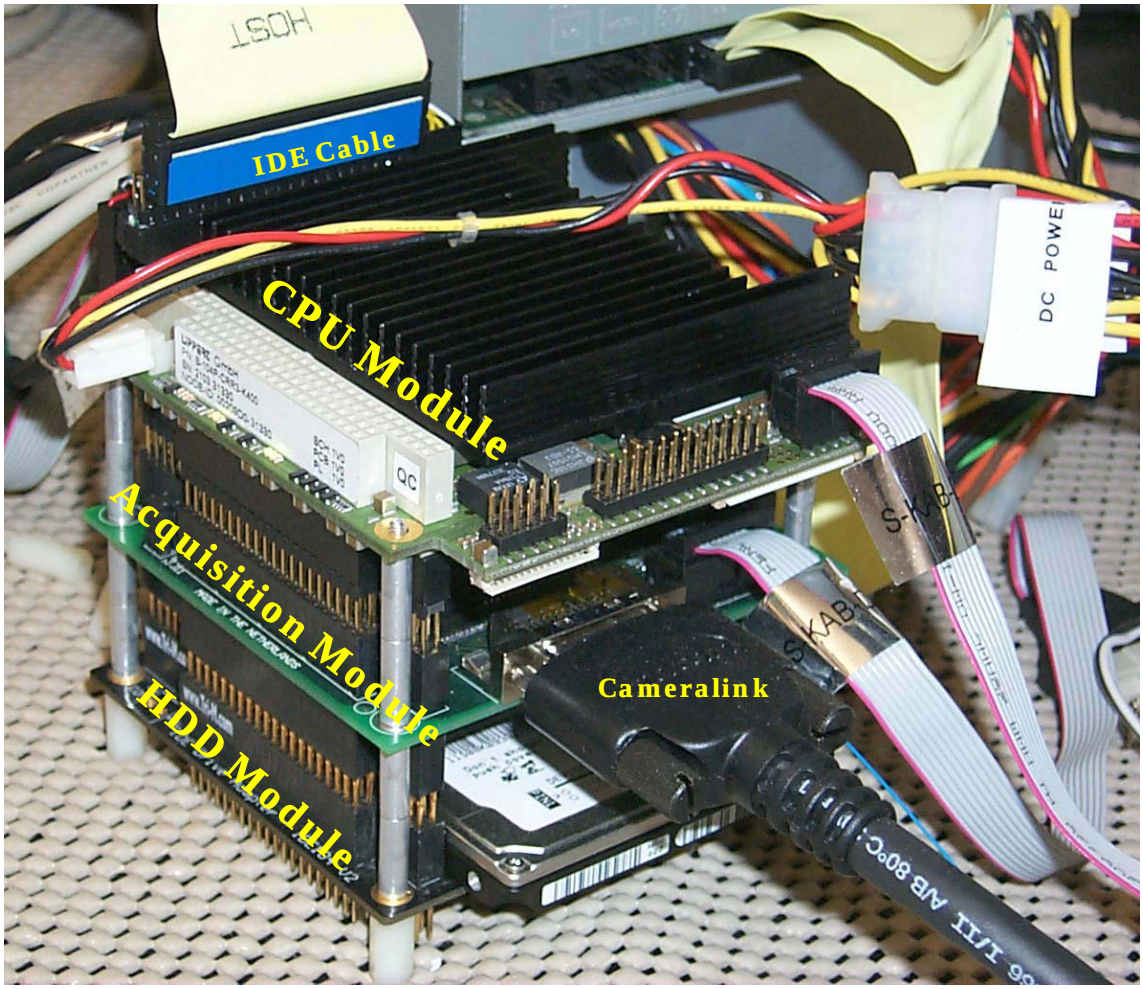
[33] J. Singh, A. Antoniou, and D. J. Shpak, "Hardware Implementation of a Wavelet based Image Compression Coder," *IEEE Symposium on Advances in Digital Filtering and Signal Processing,* pp. 169-173, 1998.

[34] T. W. Fry and S. Hauck, "Hyperspectral Image Compression on Reconfigurable Platforms," *Proceedings of the IEEE Symposium on Field-Programmable Gate Arrays for Custom Computing Machines (FCCM2002),* pp. 251-260, April 22-24, 2002.

[35] Xilinx Corporation. *Spartan-IIE FPGA Family data sheet*, San Jose, California, 2004. http://direct.xilinx.com/bvdocs/publications/ds077.pdf

APPENDIX A

HARDWARE COMPONENTS OF ZOOVIS-SC



PC/104-*Plus* stack comprising of CPU, Acquisition, and HDD modules. Cameralink and IDE cable are also visible.

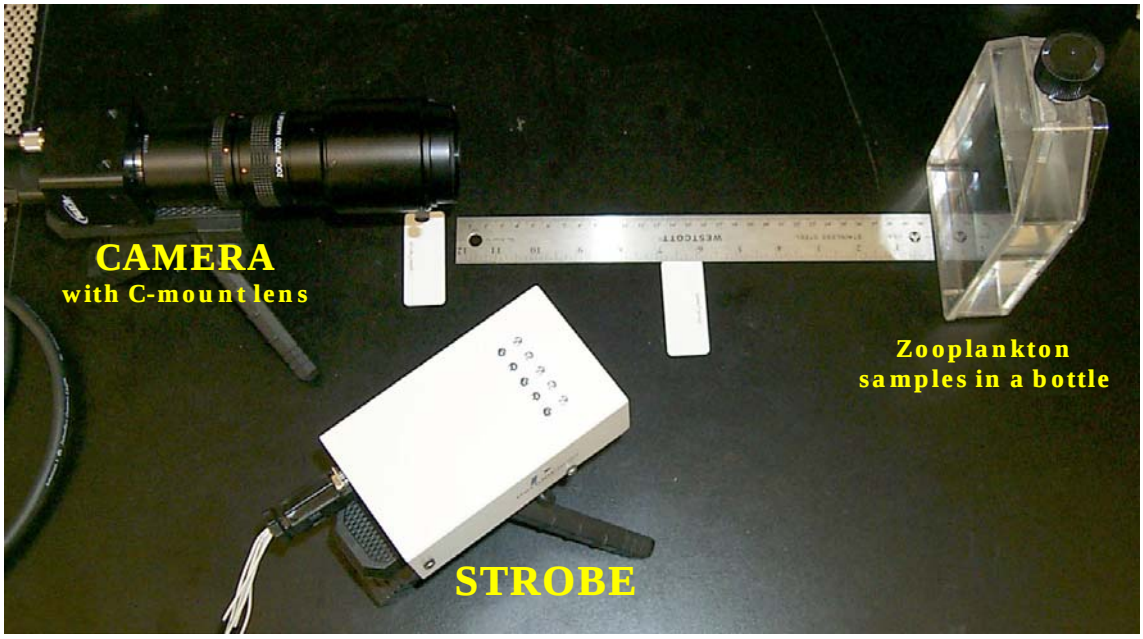
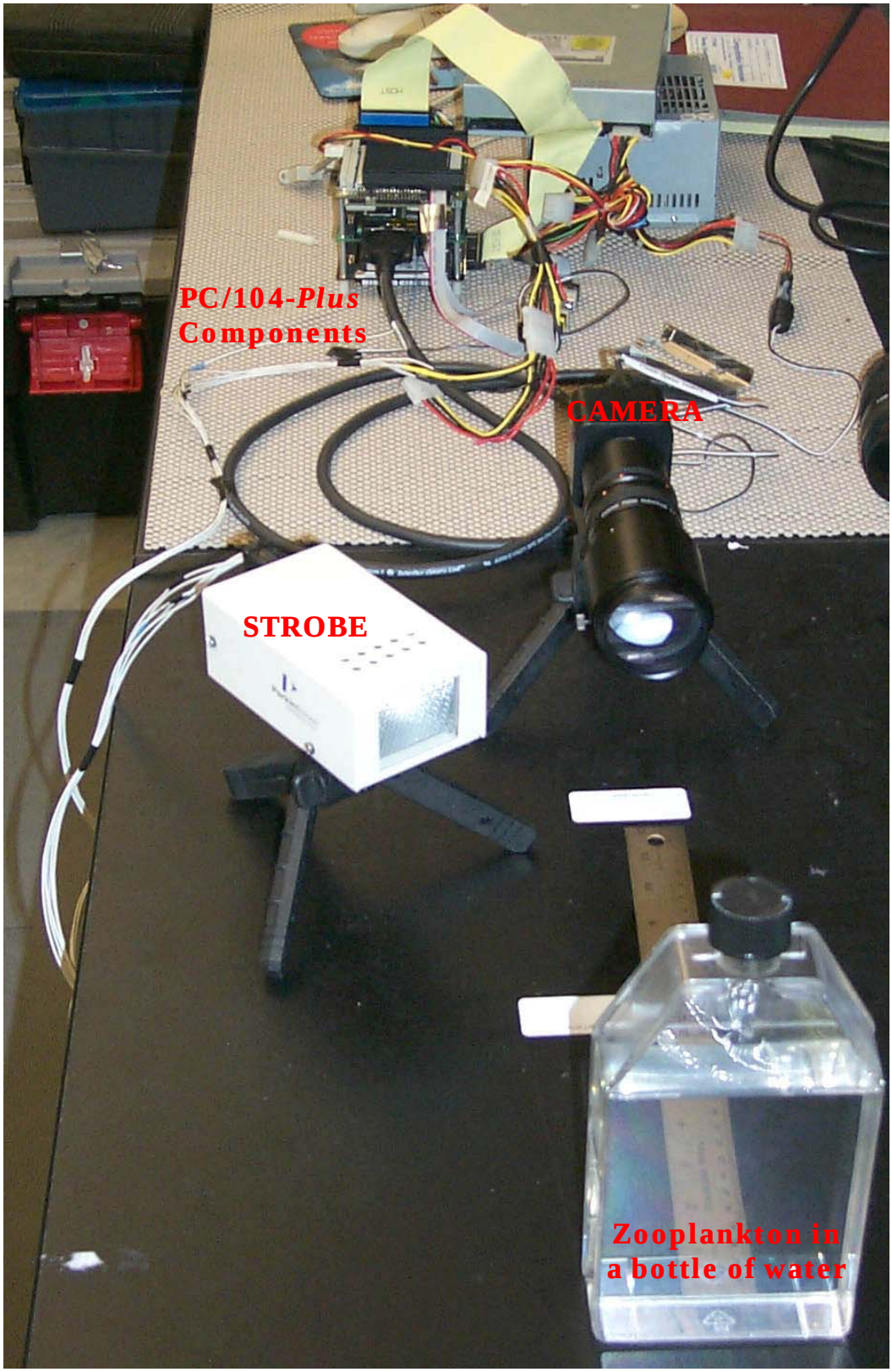


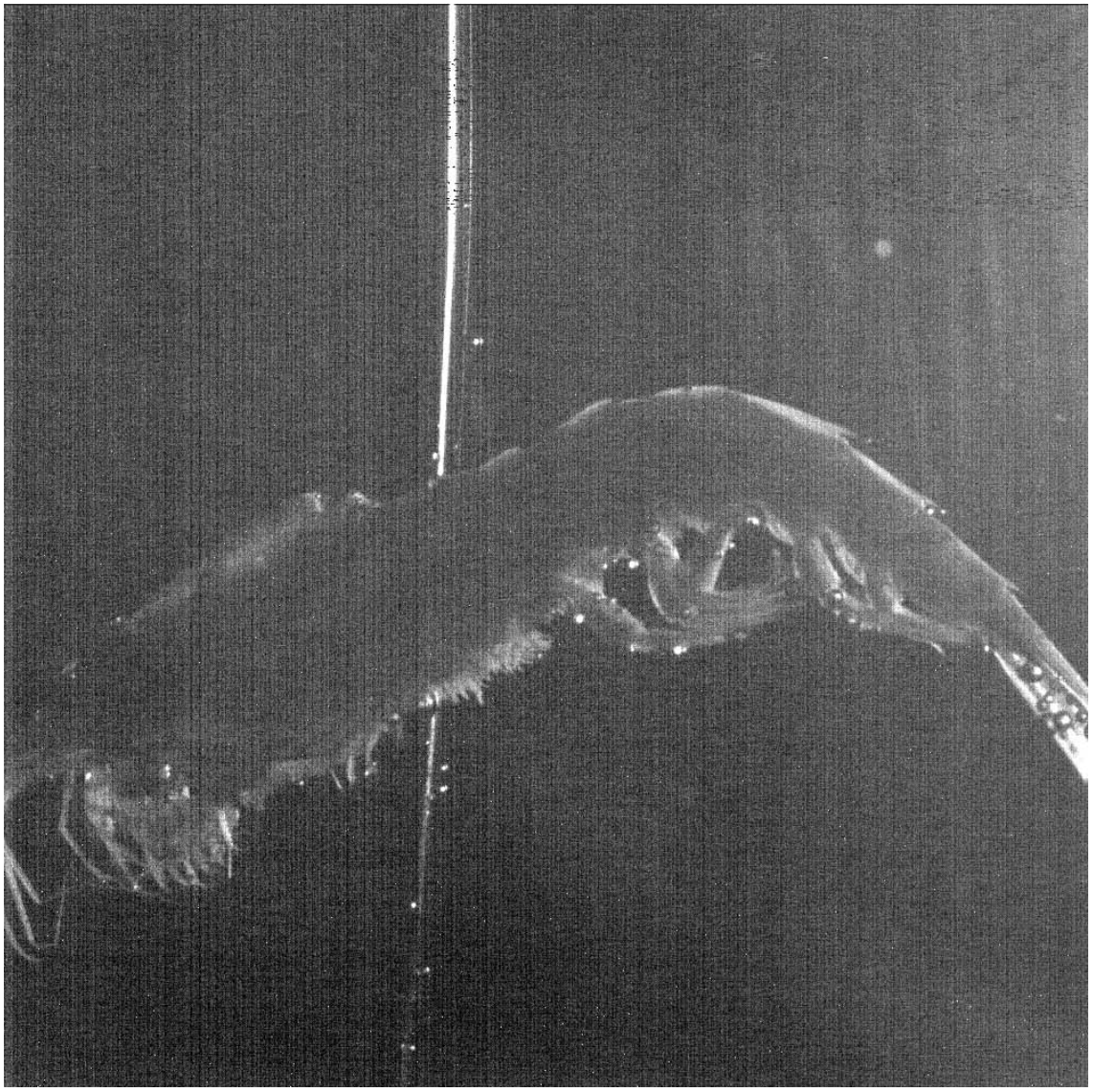
Figure showing camera, strobe and target zooplankton in the lab.



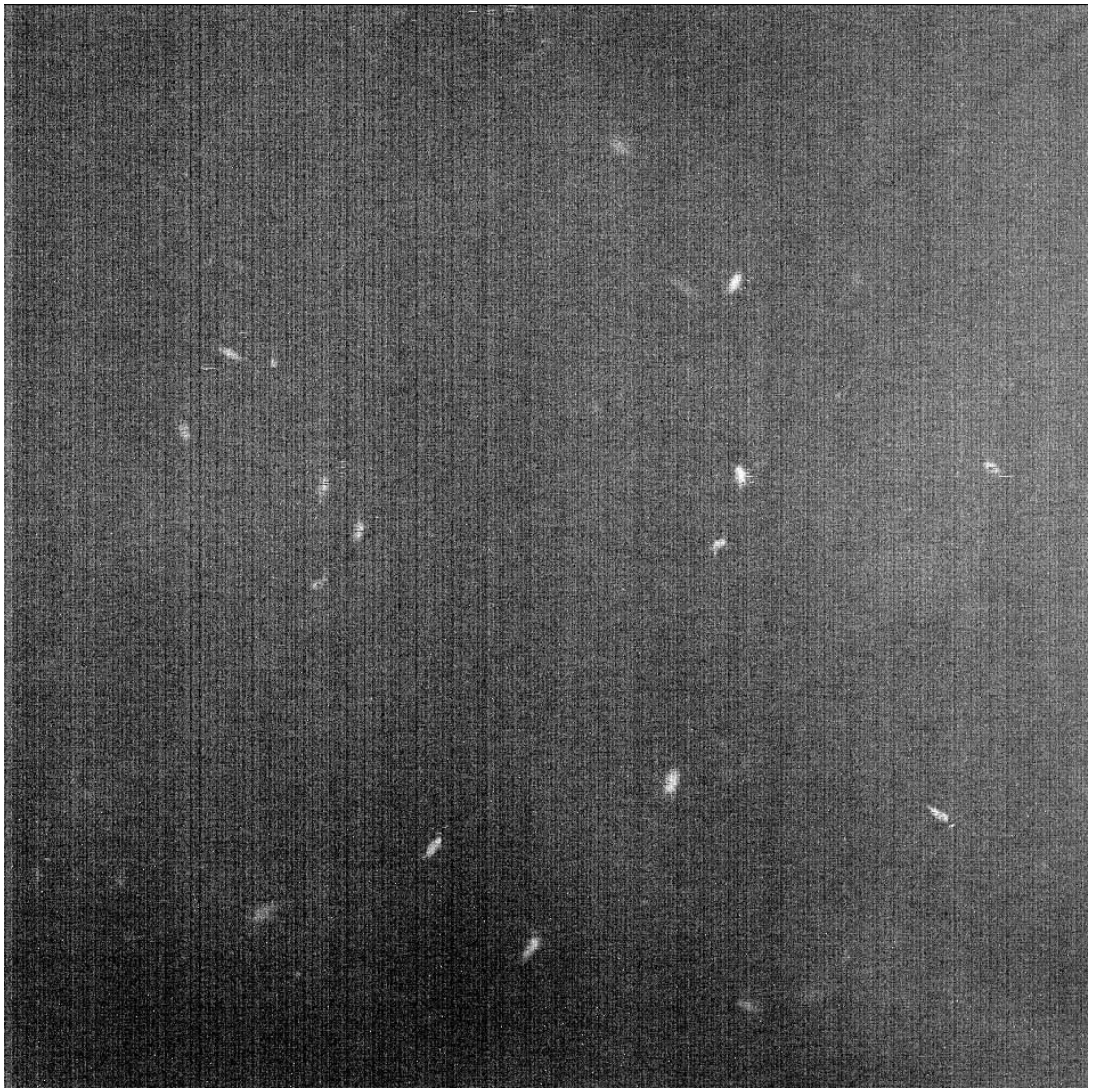
Complete setup of ZOOVIS-SC in the lab.

APPENDIX B

IMAGES TAKEN IN THE LAB BY ZOOVIS-SC



I used a shrimp hung by a wire in a bottle of water to simulate the underwater settings and keep the target in the field of view of the camera.



Live zooplankton collected from the ocean and placed in a bottle of water at 12" from the tip of the camera lens.

VITA

Dattatreya Reddy Tetala was born in Vishakhapatnam, India. He graduated from Grandhi Mallikarjuna Rao Institute of Technology, an affiliate to Jawaharlal Nehru Technological University in Hyderabad, India with a degree in Bachelor of Technology in Electrical and Electronics Engineering in 2001. After he received his undergraduate degree, he came to the United States of America to continue his studies. In the fall of 2001, he enrolled in the graduate program at Louisiana State University in Baton Rouge, Louisiana. He will be awarded the degree in Master of Science in Electrical Engineering at the August 2004 commencement.