DEVELOPMENT AND IMPLEMENTATION OF INTERACTIVE 3D VIDEO ENVIRONMENT ON RUN-TIME RECONFIGURABLE FPGA PLATFORM

By

Sergiy Zhelnakov, B.Sc.
Kiev Polytechnic Institute, Ukraine, 1984

A Thesis
Presented to Ryerson University
In partial fulfillment of the requirements
for the degree of Master of Applied Science
in the program of Electrical and Computer Engineering

Toronto, Ontario, Canada, 2006

Declaration of Authorship

I hereby declare that I am the sole author of this thesis

I authorize Ryerson University to lend this thesis to other institution or individuals for the purpose of scholarly research

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

Borrower's Page

Ryerson University requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Name	Signature	Address	Date

Development and Implementation of Interactive 3D Video Environment on Run-Time Reconfigurable FPGA Platform

Sergiy Zhelnakov,
Master of Applied Science, 2006,
Electrical and Computer Engineering,
Ryerson University

ABSTRACT

Video data processing tasks are traditionally performed either through software-based systems when various algorithms must be applied to the data and when time issue is not critical, DSPs – when certain time constraints are set but when the set of tasks is limited, or ASICs – when the highest performance is required, the set of tasks is fixed and highly optimized, the data stream doesn't change, and the number of data streams is limited.

For a real-time system which must operate on multiple data streams which also can change in time and on which various data processing algorithms must be applied neither of the mentioned approaches can be used. Timing requirements and power limitation does not allow utilization of a sequential CPU. ASIC becomes too big to accommodate multiple processing circuits for each algorithm and associated modes. Only Run-Time Reconfigurable (RTR) FPGA approach allows implementation of such a system.

The thesis presents a real-time stereo vision system with elements of synthesis of interactive 3-D virtual objects designed and implemented on the FPGA-based reconfigurable platform. FPGA chip integrates a hybrid architecture system with multi-mode and multi-stream processing ability for time critical tasks and with embedded microprocessor(s) for computing complex algorithms for 3-D objects synthesis for which timing requirements are not so strict.

An approach for the formal presentation and processing of the 3-D virtual objects and their transformation is also analyzed and presented in this paper. Architecture synthesis and optimization for a hybrid system are also considered.

The experimental results proved the effectiveness of proposed approach: the FPGA-based system-on-chip provides stereo visualization in different modes (actual image and edge detection image), with synthesized 3-D controls (pressed and released buttons).

Keywords: Stereo Vision, 3-D Synthesis, Reconfigurable Platform, FPGA

Acknowledgements

I have been fortunate over the past few years to have worked with my supervising professor, Dr. Lev Kirischian. His guidance, knowledge and support not only contributed much to this work but inspired me to immerse in run-time reconfigurable systems.

My special thanks to professor, Dr. Vadim Geurkov for his valuable suggestions and support during all the time of research and working on the project.

I would also like to thank the Department of Electrical and Computer Engineering for providing facilities and technical resources needed for the research.

Many thanks to the members of the review committee for their participation and consideration. My special gratitude to the OGS for support of this research through scholarships.

I'd like also to express my deep gratitude to Mr. Jim Koch and members of the Embedded Reconfigurable Systems Lab (ERSL), Ph.D. students Mr. Peter (Pil Woo) Chun and Mr. Valeri Kirischian for their technical assistance during the implementation phase of the project.

Last but not least, many thanks to my wife and kids for their understanding, patience and support during all the time of working on this thesis, and to my parents whose wise advises years ago opened the world of electrical engineering for me.

Table of Contents:

List of figures:

List of tables:

List of Acronyms

3-D – Three Dimensional
ACG – Architecture Configuration Graph
ASIC – Application Specific Integrated Circuit
CISC – Complex Instruction Set Computer (or Computing)
CLB – Configurable Logic Block
CPU – Central Processor Unit
CPLD – Complex Programmable Logic Device
DFG – Data Flow Graph
DSP – Digital Signal Processor
FPGA – Field Programmable Gate Array
GPP – General Purpose Processor
HDL – Hardware Description Language
ICM – Image Capture Module
ILP – Instruction Level Parallelism
IOB – Input Output Block
IP Core – Intellectual Property Core
ISE – Integrated Software Environment
LSB – Lowest Significant Bit
LUT – Look-Up Table
MSB – Most Significant Bit
LCD – Liquid Crystal Display
LVDS – Low Voltage Differential Signal
PCB – Printed Circuit Board
PLD – Programmable Logic Device
RISC – Reduced Instruction Set Computer (or Computing)
RTR – Real-Time Reconfigurability
SOC – System-On-Chip
VGA – Video Graphics Array
VHDL – Very High Speed Integrated Circuits (VHSIC) Hardware Description Language
VOM – Visualization Output Module
VPM – Video Processing Module

1. INTRODUCTION

1.1. Motivation

Visual information plays a special role in perception of the surrounding world by humans. Still and motion images are considered the most important informative sources. Through vision we perceive various information and knowledge, directions and instructions for all kinds of our activities.

Special vision systems are widely used in many areas of human activity: in medical service and research, e.g. computer tomography systems (CT), magnetic resonance imaging (MRI), X-ray or ultrasound imaging systems, for telemedicine; in navigation systems for navy and aviation, for spacecrafts, submarines and Unmanned Autonomous Vehicles (UAV) incorporate sophisticated video systems built on the base of advanced image processing algorithms. Video systems play a special role in space exploration as it was demonstrated by the latest NASA's mission to the Mars started in 2004. The images of the Martian surface sent to the Earth by two autonomous rovers Spirit and Opportunity are hard to over-evaluate. Flight simulators provide virtual airplane cabin set-up and environment for training of pilots. Entertainment and art are based both on video and audio perception of a human.

The areas of application of real-time vision systems include also telematic systems used for remote control of manipulators operating in hazardous environment. It is very required for operating robotized manipulators in space, when the actual stereo scene is visualized in real-time and interactive virtual controls images are dynamically generated on the visualization device as well, thus making it possible for an astronaut not to look aside for activating a needed control button.

Video systems also play important role in industry, engineering and science, for example, for research of fast processes in chemistry and mechanics. High speed video recording is used for cars collision research to provide engineers with the information on how to improve a car design to reach better level of driving safety. It is also used in aerodynamics for study and research of airplanes flight characteristics in different environmental conditions and of new materials behavior under critical conditions.

Today's video systems provide possibility to visualize both real-time (or recorded) video and synthesized virtual three-dimensional graphical objects. Various captions, charts, diagrams and artificially created backgrounds are an integral part of modern TV broadcasting.

Different image processing algorithms and techniques are widely employed in today's cinema industry for creating scenes from other planets or prehistoric creatures.

Stereo vision systems provide more realistic view of the surrounding world or virtual reality than just "one-eye" view since scene depth information is there. Stereo vision techniques necessarily integrate two sources of video data, i.e. two video cameras which should be properly oriented in space to produce matched images. Panoramic vision systems are aimed to provide a realistic view of the scene with a very wide angle of view (up to 360°).

The main purpose of real-time multi-channel vision systems is to process in a certain way one or multiple video data streams. Data stream means that the data flow is homogenous and regular. Each unit data bears information just about the current elementary picture component (pixel) brightness. In general case, the system is doing the same operations over each coming pixel or over a group of coming pixels from the current scanning line or from a number of adjacent lines. It means that the computation can be performed in parallel and be deeply pipelined.

A typical approach for real-time vision systems design is that a kind of hybrid architecture is employed for implementation. On the one end of such a system are image sensors with configuring hardware based on PLD devices (CPLD or FPGA). This hardware also performs initial processing on incoming data to facilitate the interface between the data source (video cameras) and the main processing system. The main processing system is usually built on the base of a personal computer or high performance workstation, General Purpose Processor (GPP), Digital Signal Processor (DSP), or Application Specific Integrated Circuit (ASIC). For the visualization of real-time video and synthesized graphical objects the resources of a standard PC with a powerful graphic card are usually used.

The main advantage of using a GPP, as well as host workstation for image processing is that the task implementation migrates into the software design domain. It allows implementing sophisticated algorithms using traditional, and thus well developed, technique of sequencing programming. The evident drawback of this approach is that real-time processing is impossible in this case. A GPP based system is not fast enough to produce a processed unit data at the rate

of real-time data rate. For example, for the standard VGA resolution of 640 x 480 pixels and frame rate of 60 fps a system must output a processed pixel each 40nS. For a RISC microprocessor operating at the clock frequency of 200MHz a clock cycle takes 5nS. Even if it takes four clock cycles per one command execution, only two commands can be allocated within a time slot of 40nS. It's not enough just for reading data from memory. For real-time video systems GPP approach for data processing is not acceptable.

Using DSPs for image processing tasks can significantly facilitate various algorithms implementation, but deep pipelining and parallelism, as it is needed for stream data processing, can not be reached.

From the performance point of view using ASICs for image processing tasks gives the best results. But ASICs are very application specific devices destined only to mass production sector. For example, ASICs for graphic adapters are built on the base of parallel vector processor architecture. The performance of the modern graphic adapters is very impressive (for example, ATI's Radeon X1900 Series graphic adapter's core speed is 650MHz, it incorporates 48 pixel shader processors, 8 vertex shader processor and 256-bit 8-channel GDDR3 SDRAM [1]). But they are designed to perform only a particular processing task directed mostly to the games industry: generating highly realistic synthesized 3-D image. The operational frequencies for modern graphic processing ASICs are close to physical limits, and thus they need complex cooling systems. ASIC based systems are also not reconfigurable, their architecture is fixed. If a system needs to provide, for example, different resolution for different viewing angles, as it may be needed for panoramic vision systems, or to process two or more data streams as it's needed for stereo vision, a standard ASIC can't be used. For a multi-mode system a number of ASICs must be employed. In this case the chips must communicate to each other via PCB's copper traces and the resulting cross-talks limit the performance of the system. Even if a single chip's core can operate at the speed of 500MHz and above, the PCB limits the data rate by approximately 133MHz. For multiple-chip systems the power consumption increases and reliability falls. Besides, time-to-market for the ASIC design is significantly higher than for other approaches.

It means that for processing of a number of data streams which can change in time and which may require different processing algorithms to be applied on data streams, a system must provide high level of parallelism and reconfigurability, and has to be implemented within one

chip. So, the problem is that we are speaking about video systems that can incorporate several algorithms and several modes of each algorithm. This is the requirement for many applications nowadays (e.g. synthesis of complex virtual environments, MPEG 4, 7, 21 video compression, object recognition, etc.)

Timing requirements and power limitations does not allow utilization of sequential CPU. ASIC becomes too big to accommodate multiple processing circuits for each algorithm and associated modes. Only Run-Time Reconfigurable FPGA approach allows implementation of such kind of systems. On the other hand, insertion of artificial virtual images into real live 3D background needs to perform complex algorithms but timing requirements are not so strict. Thus, hybrid architecture which should consist of RTR FPGA and embedded ILP CPU may be an optimal solution. Thus R&D in this area becomes important and highly demanded for creating 3D virtual environment on a chip.

Modern FPGAs have huge amount of logic resources. For example, Xilinx' Virtex-II Pro FPGA XC2VP100 offers logic resources equivalent to 10 million system gates. Virtex-5 FPGA has up to 330,000 logic cells (equivalent to 33 million system gates). These FPGAs incorporate 1 to 4 hard-wired PowerPC microprocessors. A number of various soft-wired microprocessor cores can be embedded into a single FPGA: PicoBlaze, Microblaze, PIC family, i8051 and others. It means that traditional software approaches can be integrated with a pure hardware part of the design in a hybrid SoC (FPGA).

The whole stereo vision system can be implemented on a single FPGA chip: it will incorporate all functional tasks, i.e. image capture from two data sources, data processing, generating two (or more) output video data streams in VGA standard for stereo visualization, synthesis of 3-D stereo virtual objects, etc. The system can be organized in such a way that different modes are possible. For example, in one mode it can generate stereo video output for visualization through shutter glasses, in another mode – to generate stereo video for projection on the screen and visualization through polarized glasses, third mode – visualization of real-time edge detection image, etc. The idea of using FPGA for real-time vision systems is actually not a new one, but the approach of integration of the whole multi-mode, multi-source real-time system on a single FPGA is the only practical solution to create a system which can be easily modified and upgraded. Finally, this system can be dynamically reconfigurable.

1.2. Objectives

The objectives of this work are to research and develop the methodology of optimal implementation of 3-D virtual environment on Run-Time Reconfigurable FPGA with hybrid architecture and practically implement and verify the developed methodology.

These objectives are divided on the following tasks:

- develop and present an architecture of a multi-mode system-on-chip for real-time stereo visualization;
- research and develop a technique for simple 3-D virtual objects synthesis using the computational resources of a single chip system based on FPGA;
- develop and implement an application specific stereo image capture system providing two-channel video stream data of the standard VGA resolution (640 x 480) from a pair of image sensors;
- implement the real-time stereo vision system in hardware; the main functional characteristics of the system are a) video data capture, b) visualization of real-time stereo video, c) synthesis of 3-D virtual graphic objects (simple controls or cursors), d) providing data exchange interface for integration of additional modules for various image processing into the SoC.
- verify implemented algorithms for 3-D virtual object synthesis, edge detection, and color decoding.

1.3. Original contributions

The main contributions to this thesis are as follows:

i) Analysis of existing hardware-based systems for real-time video presentation and data processing with the following areas of research:
   a. Overview of the development of techniques for stereo images visualization;
   b. Literature research on existing hardware-based systems for real-time video presentation, including stereovision and panoramic visualization;
   c. Analysis of approaches and techniques for synthesis of 3-D virtual graphical objects.

ii) Development and optimization of system architecture on the basis of the results of the theoretical background investigation; the hybrid architecture for real-time stereo-vision system with elements of 3-D virtual graphical object is elaborated.

iii) Implementation of developed system in a form of fully functional prototype including:

   a. Image Capture Subsystem (ICS) which integrates two image sensors and control logic implemented on CPLD;
   b. Video Processing Subsystem (VPS) on the base of Virtex-II FPGA from Xilinx which incorporates all data processing tasks;
   c. Visualization Output Subsystem (VOS) which generates output video signal for standard VGA devices. It makes possible to visualize real-time video on a CRT screen through shutter glasses or on the projection screen through polarized glasses.

iv) Successful testing of algorithms and getting first working prototype of multi-mode real-time interactive 3D video system on the base of Run-Time Reconfigurable FPGA platform.

v) Performance versus cost analysis for different size of Virtex-II and Virtex-4 FPGA chip families.

The obtained results give a practical confirmation that a *hybrid based on FPGA SoC system* approach is an optimal solution for designing a multi-mode, reconfigurable real-time stereo vision system operating on multiple data streams,. It can be successfully used for various applications in different areas of science, engineering, industry (medicine, space exploration, navigation, training systems, UAV navigation systems, surveillance systems, computer games industry, and others).

1.4. Thesis organization

The thesis is organized in six chapters:

- Chapter 1 introduces the importance of tasks of real-time video information presentation and analysis for various areas of human activity and reveals the idea of a SoC approach for a fully functional multi-mode real-time stereo vision system with 3-D virtual graphic objects synthesis. The chapter distinguishes the objectives for this research and presents

the main contributions fulfilled for this project successful implementation. It presents also the organization of the thesis and the contents of its chapters.

- Chapter 2 contains a review of existing hardware based systems for real-time vision, including stereo and panoramic vision systems which were reported in journal publications and books. It presents also a brief look on virtual reality systems and practical interest in this kind of systems in various fields of activity. The chapter starts with a brief historical review on the appearance and development of systems for visual information presentation. The chapter concludes with the comparative analysis of the existing hardware oriented systems which outlines practical importance of the design approach elaborated in this thesis.
- Chapter 3 presents research of the theoretical background in two areas related to the area of investigation. First, the geometry of solid bodies in 3-D space is considered as well as the formal approach to the transformations of the objects, namely translation and rotation. The geometry of 3-D objects representation in 2-D projection planes, rasterizing of line segments and object sides visibility aspect is also considered. The formal methods and computations are considered from the point of view of the complexity of their implementation in hardware. Second, the approaches to the architectural synthesis of digital systems are investigated. The possible architectures for the main system components are analyzed based on the principles of synthesis-oriented design approach which allows obtaining optimal hardware/software partitioned and optimized architecture.
- Chapter 4 presents the description of the architecture and design of the implemented real-time stereo vision system with elements of 3-D virtual objects synthesis. Three main subsystems, namely Image Capture Subsystem, Video Processing Subsystem, and Visualization Output Subsystem, their architecture, functionality and interconnections are revealed.
- Chapter 5 contains information about the experimental setup, experimental results and analysis of the results.
- Chapter 6 concludes the thesis and presents summary of the research and design work performed for the project implementation. Future development of the presented methods and techniques as well as of the implemented system is outlined.

2. OVERVIEW OF COMPUTING SYSTEMS FOR REAL-TIME STEREO AND PANORAMIC VISION WITH ELEMENTS OF 3-D VIRTUAL OBJECTS SYNTHESIS

2.1. Introduction

Prior to considering the existing systems for real-time stereo and panoramic vision and elements of virtual environment, a brief look on history of the subject is presented.

Visual reproduction of the real world by the human originates from the ancient ages [2]. It's possible to admit that it started with rock drawings. The taste for real life scenes reproduction expressed in fine arts in the new era.

An interesting fact is that the thoughts of depth perception go also into the distant past. In A.D. 280 Euclid said that depth perception is to receive by means of each eye the simultaneous impression of two dissimilar images of the same object [3].

In the 19th century, stereoscopic images appeared and became rather popular – two slightly different images of the same scene or object viewed at a different angle (like our eyes perceive the real world) are then viewed with special device using mirrors, presented by Sir Charles Wheatstone in 1838, or prisms as in the stereoscope constructed by Sir David Brewster in 1849. This device is considered the first practical stereoscope [3]. The functionality of these devices is illustrated in Figure 2-1. It makes possible for each eye to admit the corresponding view and to block the other.

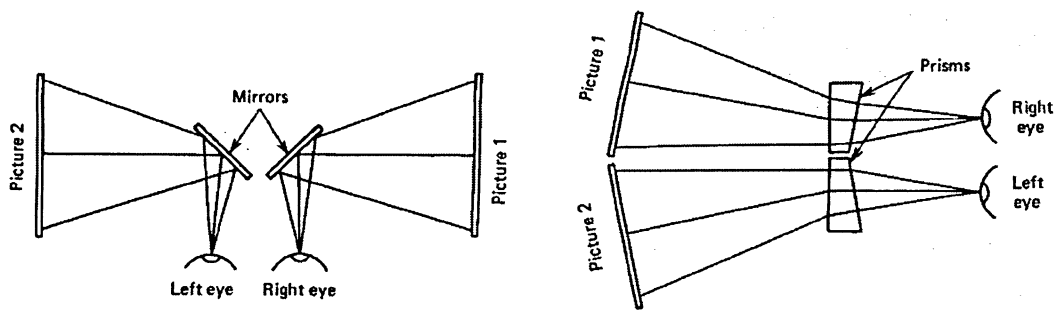


Figure 2-1. Wheatstone and Brewster stereoscopes.[1]

[1] Courtesy to [3]

There were even systems with just one picture – a set of mirrors shifted the original image for each of the eyes so that the effect close to 3-D appeared.

One more technique for stereo imaging is so-called parallax stereogram. It is proposed by F.E. Ives in 1903. A parallax stereogram consists of a fine, vertical slit plate and a specially prepared picture place behind the slit plate as depicted in Figure 2-2.

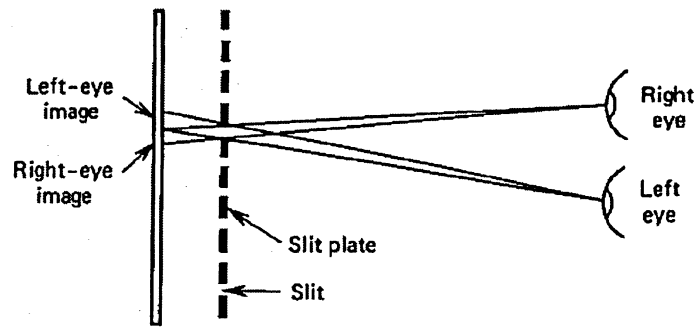


Figure 2-2. Parallax stereogram.[2]

The picture consists of the right-eye and left-eye images printed in fine stripes alternatively, with approximately the same pitch as the slit plate. This technique showed some progress until late in the 1950s. At present, however, the parallax barrier device place is mostly in museums because of some crucial drawbacks among which are darkening of the image, diffraction of light and surface reflection (slits were usually made on a glass surface).

The techniques for stereo images visualization existed before the appearance of photography. In 1839, still photography comes to the scene – a technological jump to authentic visual representation of the real world. It pushed the 3-D visualization devices market since a real life scenes could be photographed and viewed in 3-D. Even the battle scenes from the American Civil War were recorded in this way [4], the value of these images is impossible to underestimate.

There is also other technique for stereoscopic still images viewing which is called the anaglyph system. Its idea is that one lens of the glasses is of red color and the other is green (or blue). The picture viewed without glasses appears as two slightly displaced images, one with red lines, the other one with green (or blue).

The idea of stereoscopic viewing of still images evoked the engineers to work on the development of techniques for stereo (or 3-D) vision of motion pictures. The beginning of the

[2] Courtesy to [3]

20th century brings motion picture to life. *L'arrivée du train* made by Auguste and Louis Lumiére and shown in France in 1903 is officially the first 3-D motion picture made for public exhibition [4].

There are different systems for 3-D motion picture viewing which are in use till our days. One system uses the same anaglyph two-color glasses and correspondingly projected images. It's not the truly color stereo vision, but the human brain has an ability to reconstruct images in a way that perceived objects appear like color ones.

Another approach, the Polaroid system, which is being used for commercial 3-D movies since early 1950s, is based on a light polarizing materials invented by American inventor Edwin H. Land in 1932. In this method, known as Natural Vision, two films are recorded by two recording devices located at a certain distance one from another (usually close to a distance between human eyes). Then the images are projected on a screen through the lenses that polarize light at different angles. The lenses of the glasses which are worn by viewers are similarly polarized. In this way each eye perceives only image projected from the corresponding film [2]. All these systems employ projection method for obtaining 3-D effect.

In the late 1980s, efforts to improve picture quality took two routes: increase in frame rate (Showscan operates at 60 frames per second) or increase in overall picture size height as well as width (IMAX and Futurevision) [2].

It is not fair not to mention television which came into every home in the 50s of the 20th century. In the 1920s and 30s, television came into being based on the inventions and discoveries of many engineers and scientists [5]. The 'first' generation of television sets was electro-mechanical devices. The display (TV screen) had a small motor with a spinning disc and a neon lamp, which generated a blurry reddish-orange picture about half the size of a business card. From the 1940s the TV sets are fully electronic devices. In the recent several years, sophisticated High Definition TV systems which provide amazing image quality are available in each electronics store. But, today's television systems are still not much related to 3-D viewing.

The last but not least aspect related to the present thesis is computer graphics. It is concerned with all aspects of producing pictures or images using a computer. The field began around 50 years ago, with the display of a few lines on a cathode-ray tube (CRT). Now, the images generated by computer are nearly undistinguishable from photographs of real objects.

The range of possible applications of systems based on computer graphics is extremely wide. It is an important communication tool for all engineers to deal with plans, diagrams, schematics, and other illustrations [6]. There are systems for training of pilots with simulated airplanes, generating graphical displays of a virtual environment in real time. Feature-length movies made entirely by computer have been successful, both critically and financially [7]. Computer graphics and the corresponding digital image processing topics are widely used in medicine. Today's architectural design is impossible without applications based on computer graphics. The same story is related to design of electronic digital and analog systems, VLSI chips design and all kinds of software development. No comments are needed about the vast field of computer games.

When 3-D technologies are considered in relation to computer graphics on a personal computer, there are two major techniques. First, the representation of 3-D objects on a flat screen. The objects change their position and orientation in space and the system (very often it is interactive) represents the current view of the objects in real time. This task is performed by special techniques which require powerful graphic adapters, CPU and software utilities and programming languages (like DirectX from Microsoft or Virtual Reality Markup Language, VRML). The performance of a graphic adapter is measured in the number of elementary geometrical figures (the body of the 3-D object is composed of this kind of triangles or polygons) processed per second. For today's graphic adapters this number reaches hundreds of thousand per second, and the cost of most advanced adapters is several hundred dollars. The second technique aimed at real 3-D visualization. One approach is based on using the shutter glasses and the corresponding alternating frames output on the monitor's screen. The second approach utilizes projection of two images through polarization filters. It's also possible to view 3-D still images using the anaglyph system.

Virtual reality (VR) systems are built on the same principles as standard computer graphics applications. They generate two video data streams, one for each eye. The images for each eye are almost the same for the distant objects but quite different for close objects. For close objects the parallax effect is taken into consideration for computation of objects representation on the screens. The viewer usually wears a helmet with LCD glasses, so that each eye admits only its own image. VR systems are usually interactive. There are special

devices (a kind of manipulators like a glove, for example) which allow to interact with virtual (i.e. generated by computer) objects like buttons, doors, etc.

Another 3-D technology which still lives mostly in the physics research labs is holographic vision. Holography offered for the first time a method of spatial imaging satisfactory for accommodation. It means that it provides a genuine 3-D image which may be viewed from different points so that a viewer can walk around a holographic object. Before holography, all the methods proposed or used relied only upon convergence and parallax. But holographic process is accompanied by many practical disadvantages which are mostly attributed to the use of coherent light sources (lasers) in the recording and in most of the reconstructing process. Some of those are: the "filming" can't be done in real sun or artificial light; the recorded and reconstructed images are monochromatic; the object must be still in order that a good hologram be produced, thus the holography is still problematic for moving objects image reconstruction; the setup is expensive [3].

There are at least two common features for all computer graphic systems mentioned above which are related directly to the task of the present thesis. First, they all require significant hardware and software resources. The most of the systems are PC-based with advanced graphic cards. Second, it's problematic for these systems to deal with real-time, i.e. live, video. For the standard VGA resolution of 640 x 480 pixels and frame rate of 60 fps the output data rate is around 25MHz. It means that the system must read data from a certain memory location, perform necessary processing on it and output the resulting pixel data to a visualization device each 40nS. For two video channels as it is needed for stereo vision the data rate becomes 50MHz (20nS per unit data processing). For panoramic vision this value becomes even more challenging. Another parallel process must store incoming data from multiple video data sources into memory. Definitely this kind of tasks must be implemented in hardware and thus pipelined and performed in parallel for maximal performance.

Since the objective of this thesis is an attempt to implement a real-time stereo vision system, expandable to a panoramic vision system, with elements of virtual 3-D elements synthesis as a System-on-Chip, an overview of systems, techniques and approaches is presented below. This overview is an attempt to cover three areas of research. First, a look at real-time video systems (both one-channel and stereo) based on hardware design approach; hardware setups for data capture and data processing. Second, hardware and software setups and

particular techniques for wide angle or panoramic vision systems are considered. Third, a look at approaches used in virtual reality systems which may be used for implementation in a system-on-chip for simple 3-D objects synthesis.

2.2. Real-time video systems

There are specialized embedded systems built on base of the programmable logic and DSP. These systems usually comprise image sensor(s) with necessary control logic, a framegrabber and the processing element (FPGA or DSP). Most of the designs utilize prefabricated development boards and video cameras with a certain data transmission interface which is to be integrated into design.

The idea of using FPGA to process the data flow from an image sensor is not a new one. In 1991, Sartori [8] designed an FPGA based system that captured data directly from the image sensor, implemented an edge detection algorithm, and facilitated a parallel data interface to the host processor. The image resolution was only 32 x 32 pixels with 1-bit monochrome data, and the FPGA was running at 4MHz.

Scalera, et al. [9] designed a system which included an FPGA and DSP that interfaced directly to microsensors and preprocessed the data in order to reduce the amount of information necessary to transmit to the host processor. The goals of that project were to minimize power consumption and size. The typical hardware setup for designs based on FPGA-GPP (here GPP stands for General Purpose Processor) combination is given by the authors and is presented in Figure 2-3. FPGA in this setup is used for data capture, storing data in memory (SRAM) and preliminary data processing with the purpose to facilitate a parallel data interface with the next stages of processing performed by GPP or DSP.

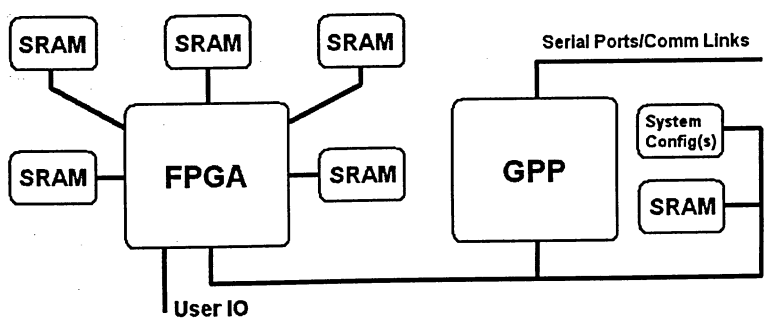


Figure 2-3. Typical Hardware Setup for designs based on FPGA-GPP combination.[3]

[3] Courtesy to [9]

Figure 2-4 illustrates common processing flow for such kinds of systems. In the Scalera, et al. architecture the box "Signal Conditioning" is related to FPGA and the boxes "Signal Processing" and "Communications Processing" correspond to the tasks executed by GPP (or DSP). In general case these boxes (or tasks) can be assigned to processing devices in another manner. For us the setup in which all the tasks are executed in FPGA is of major interest.

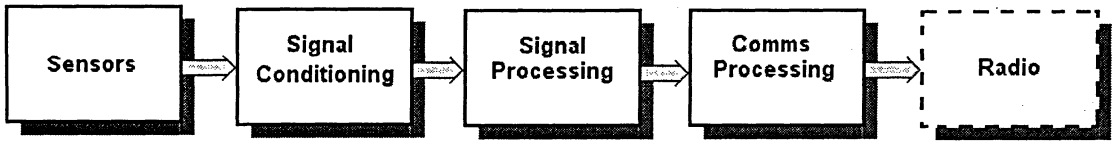


Figure 2-4. Common Processing Flow.[4]

Lienhart, et al. [10] were able to implement real-time video compression on three image sequences in parallel at 30 fps using FPGA. Their input data came from the host PC via the PCI bus, however, it was noted that that the solution could be implemented with a direct camera connection. In that design FPGA is loaded with sophisticated computational tasks such as discrete cosine transform (DCT) of the 8 x 8 pixel blocks of an image, quantization of the results and data compression using entropy coding mechanism.

J.Woodfill and B.V.Herzen presented a Real-Time Stereo-Vision on the PARTS Reconfigurable Computer in 1997 [11]. It consists of 16 FPGAs connected in a partial torus, each associated with two adjacent SRAMs. The application implemented on the PARTS engine was a depth map on the images of 320 x 240 pixels at 42 fps. The device was plugged into a PCI slot of a PC. The data was coming prom the host PC and the processed disparity map was sent to the host processor via the PCI bus.

Leeser, et al [12] presented a system including high quality video camera (1024 x 1024 pixels resolution and up to 30 fps) and frame grabber connected directly to an FPGA processing board. The hardware setup for the system is presented in Figure 2-5. The advantages of this setup include minimizing the movement of large datasets and minimizing the latency by starting to process data before a complete frame has been acquired. The FPGA design consists of two parts running concurrently. One part packs incoming data into 64-bit words and then

[4] Courtesy to [9]

writes and reads data to and from memory. The data is stored in the on-board memory. When the data is read, it is stored in the on-chip memory for faster access. The second part is a custom block which contains the logic for application specific algorithms. The processed data is then stored in output memory chips. Upon the request of a host PC via the PCI bus the data is sent to the PC.

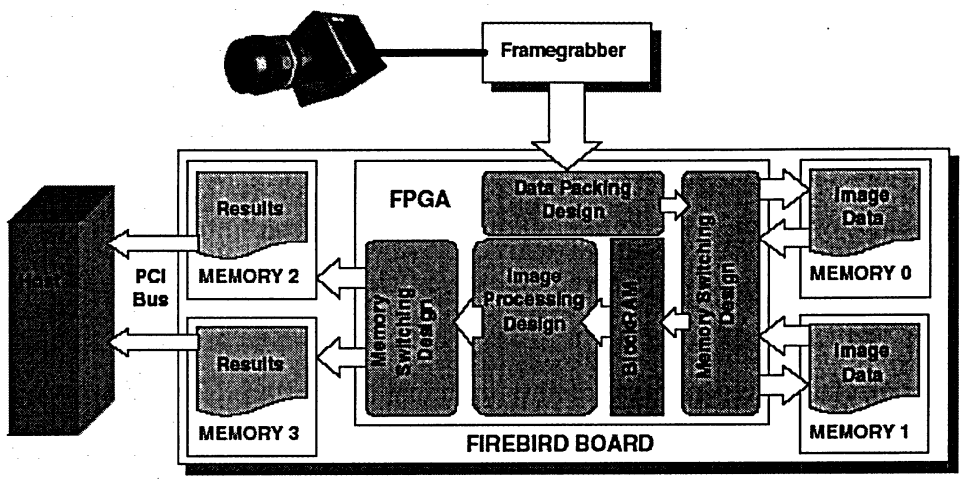


Figure 2-5. Hardware setup of the high quality vision system by Leeser, et al.[5]

The system was designed for two particular applications, one for the real-time implementation of Retinal Vascular Tracing (RTI), the other for Particle Image Velocimetry (PIV) in the computational fluid dynamics. The system was built using the modules available on the market, such as camera module 1M30P from Dalsa, framegrabber module from Dillon Engineering and FPGA board Firebird from Annapolis Microsystems.

An embedded stereoscopic vision system for an autonomous rover is presented by G. Genta, et al. [13]. The system incorporates two CMOS sensors interfaced to a FPGA which configures cameras and captures incoming data. The data then is then passed to a DSP for processing. The on-board processing module provides the rover control system with the information about obstacles and distances to the objects in front of the rover. The compressed data is also sent to a remote host via radio link. The hardware setup and the block diagram of the vision system are presented in Figure 2-6.

[5] Courtesy to [12]

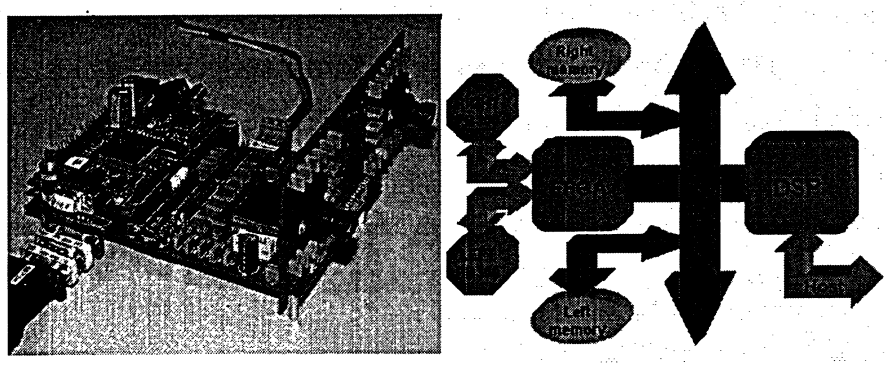


Figure 2-6. Hardware setup and block diagram of the vision system by Genta, et al.[6]

For visual navigation systems extraction of relevant visual information is an important issue. The images are changed when a vehicle moves, the cameras change their orientation in space since the "camera head" may tilt or rotate. Thus, a special technique for the images interpretation, correlation and cameras calibration must be applied. A method called Active Vision or Dynamic vision which serves to this purpose is presented by E.Grosso and M.Tistarelli [14]. A cooperative schema is proposed in which motion and stereo vision is used to define scene structure and determine free space areas. In this approach binocular disparity is computed on several stereo images over time and combined with visual data flow from the same sequence to obtain a relative-depth map of the scene. Active control strategy also considerably reduces the need for calibration of the visual data sources, i.e. video cameras. The existing correlation methods used for computing the disparity map are considered by Hirschmüller [15]. These methods are typically based on adaptive window approach and its simplifications. Hirschmüller presents also his approach for improvements in Real-Time correlation-based stereo vision. His technique includes novel multiple window approach for decreasing errors at object borders, general error filter for invalidation of uncertain matches, and a border correction method which improves object borders further. The overview of methods used for disparity map generation is also given by M.Kuhn, et al [16]. The existing methods are subdivided in three groups based on the used techniques: SSD (sums of squared differences), Census transformation and occlusion detection. Stereo vision based on correspondence matching between camera images is a typical method of 3-D instrumentation. A well-known method for correspondence matching is using of sum of absolute differences

[6] Courtesy to [13]

(SAD). Parallel processor architecture for implementation of this method is presented by K.Miura, et al. [17]. This approach can be taken into consideration for FPGA-based designs.

K.Yoshida and S.Hirose introduced real time stereo-vision with multiple arrayed camera in 1992 [18]. The idea of the multiple arrayed camera (MAC) is to facilitate the correspondence (correlation) computations using simple geometric relationship of images on the same scanning lines. But it requires additional hardware resources and significantly complicates the system. The configuration of multicamera system is presented in Figure 2-7.

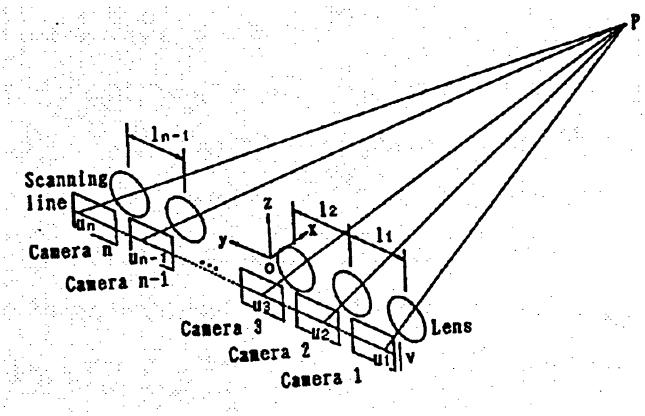


Figure 2-7. Multicamera system configuration.[7]

A trinocular system for feature detection for stereo-vision-based unmanned navigation is presented by B.K.Quek, et al. [19]. They use software methods for advanced features detection form stereo images. The stereo video data source is a trinocular stereo-vision system (Digiclops) manufactured by Point Grey Research Inc. (Canada), see Figure 2-8.

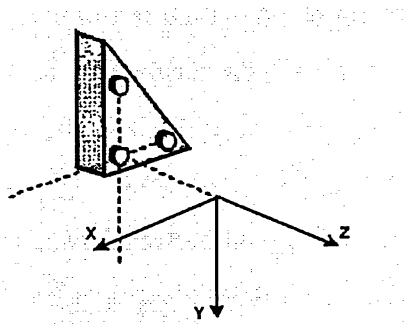


Figure 2-8. Digiclops camera system by Point Grey Research Inc.[8]

[7] Courtesy to [18]
[8] Courtesy to [19]

It is supplied with a software development kit (Triclops SDK) which can be used for disparity images generation. This is an example of a complete system which can be used for a particular task. But again, a PC is needed for working with this system. The geometrical background of the trinocular stereo vision is presented by N.Ayache and F.Lustman [20]. They conclude that the main advantages of trinocular versus binocular stereo are simplicity, reliability, and accuracy. But it requires additional hardware and more computation resources are needed.

The importance of 3-D stereo in conjunction with interactive image control for medical visualization is illustrated by D.Maupu, et al. [21]. Their work relates to the endovascular operations. A primary difficulty with endovascular procedures is that they are typically guided only by flat-projection images nowadays. All 3-D information is collapsed into a projection view making 3-D perception of the vascular network difficult. The proposed technique provides a clinician with a 3-D image of a patient's vasculature so that he could move injection needle to a needed vein more accurately. The approach requires the preoperative creation of a 3-D model of each patient's portal venous system. This 3-D model is then correlated with a real fluoroscopic image of a patient's affected organ. Special algorithms and OpenGL library resources are used to correlate 3-D model and real image since the injected needle causes displacement of the veins. The system includes also a head tracker which allows adjusting the image on a screen depending on the clinician's head location so that it's possible to view the veins on the monitor from different angles. This implementation is actually not a pure real-time application. Besides, it requires rather expensive setup. But it shows the link between 3-D applications in different fields and thus possible implementation of approaches used in, for example, robotic vision, for medical purposes. It also shows importance of virtual objects synthesis as an integral part of vision systems.

A typical approach for real-time vision systems design is that they utilize a set of processing devices such as programmable logic devices (typically FPGA), microcontrollers, DSPs, and, finally, personal computer. From another hand, today FPGA vendors offer devices with logic and computational resources enough to hold all the design inside one chip. This approach is used for some designs. For example, K.Appiah and A.Hunter present their Single-Chip FPGA implementation of Real-time Adaptive Background Model [22]. They use a digital video camcorder as a data source and a prototyping platform with a Xilinx Virtex II FPGA. So-

called switching RAM architecture is implemented in the design which means that the data of the whole frame coming from video camera is written to the memory bank 1, and the next fame data is written to the bank 2. For the next frame the banks are switched again. At the same time, data reading is performed from the bank which is not busy with data writing. The memory banks for the background image are switched by the same scheme. The setup for the system is illustrated in Figure 2-9.

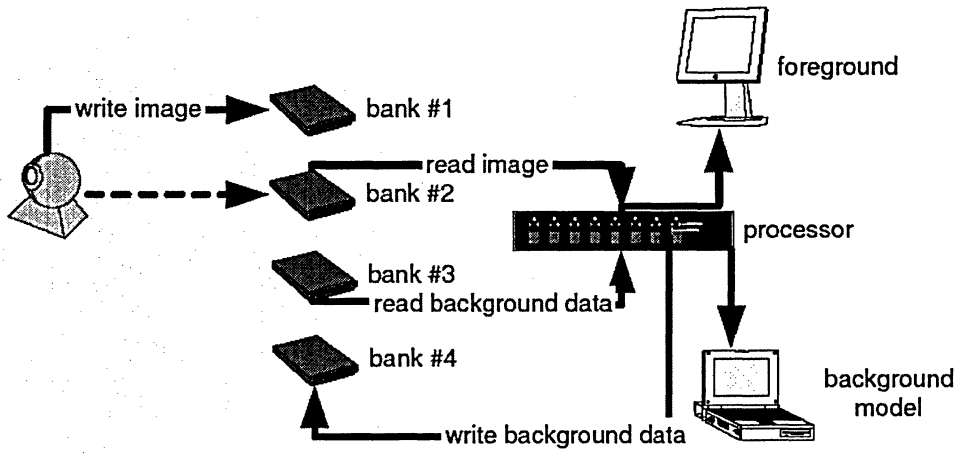


Figure 2-9. Hardware setup for the system with switching RAM architecture.[9]

There are techniques for environment modeling with stereo vision. An overview of existing approaches is given by D.Murray and J.J.Little [23]. They use a surface element model named patchlet in their method. This approach is interesting in a way that virtual object can be not only fully synthesized based on mathematical models but also originate from real images, i.e. real objects.

A high Speed Stereo Vision System operating at frame rates up to 200 fps is presented by J.I.Woodfill, et al. [24]. It implements a DeepSea processor (ASIC) which computes absolute depth based on a highly parallel, pipelined architecture that implements the Census stereo algorithm, see Figure 2-10. In this paper the advantages of stereo depth computation are distinguished as follows. First, stereo is a passive sensing method. Active sensors, which rely on the projection of some signal into the scene, often pose high power requirements or safety issues under certain operating conditions. They are also detectable – an issue in security or defense applications. Second, stereo sensing provides a color or monochrome image which is exactly (inherently) registered to the depth image. This image is valuable in image analysis,

[9] Courtesy to [22]

either using traditional 2-D methods, or novel methods that combine color and depth image data. Third, the operating range and depth (Z) resolution of stereo sensors are flexible because they are simple functions of lens field-of-view, lens separation, and image size. Almost any of operating parameters is possible with an appropriate camera configuration, without requiring any changes to the underlying stereo computation engine. Fourth, stereo sensors have no moving parts, an advantage for reliability.

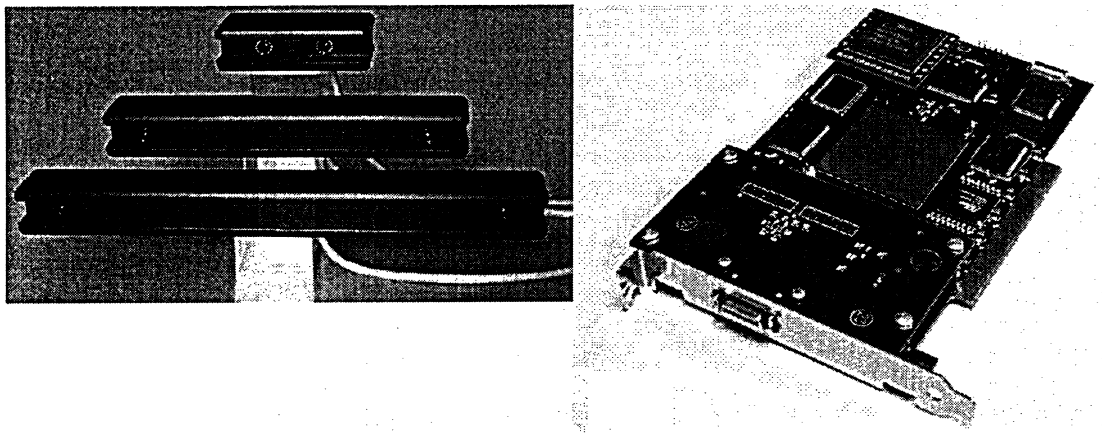


Figure 2-10. Tyzx stereo camera family: 5cm, 22cm, and 33cm baselines and DeepSea processor.[10]

2.3. Wide angle or panoramic vision systems

Panoramic vision systems are also presented by numerous workgroups. One approach to get a panoramic image is to use a specially shaped, curved mirror and a conventional camera, it is presented by J.S. Chahl and M.V. Srinivasan [25] and S.Derrien and K.Konolige [26]. A reflector provides larger fields of view than a wide angle camera. Besides, it provides more predictable geometry of a reflective surface. The shape of the curved reflector is such that it provides a linear relationship between the angle of incidence of light onto the surface, and the angle of reflection onto the sensor array with respect to the centre of the array as illustrated in Figure 2-11. This property ensures that the camera provides uniform sampling of the environment in the vertical plane, independent of elevation angle.

[10] Courtesy to [24]

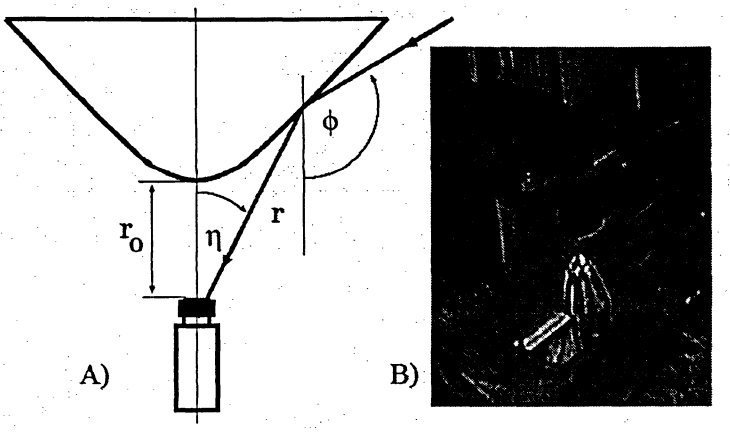


Figure 2-11. Specially shaped (curved) mirror for panoramic vision.[11]

Other group of approaches is based on using a mirror rotation mechanism as presented by T.Nakao and A. Kashitani [27] and illustrated in Figure 2-12, or a single- or double-mirror pyramid as introduced by H.Hua and N.Ahuja [28].

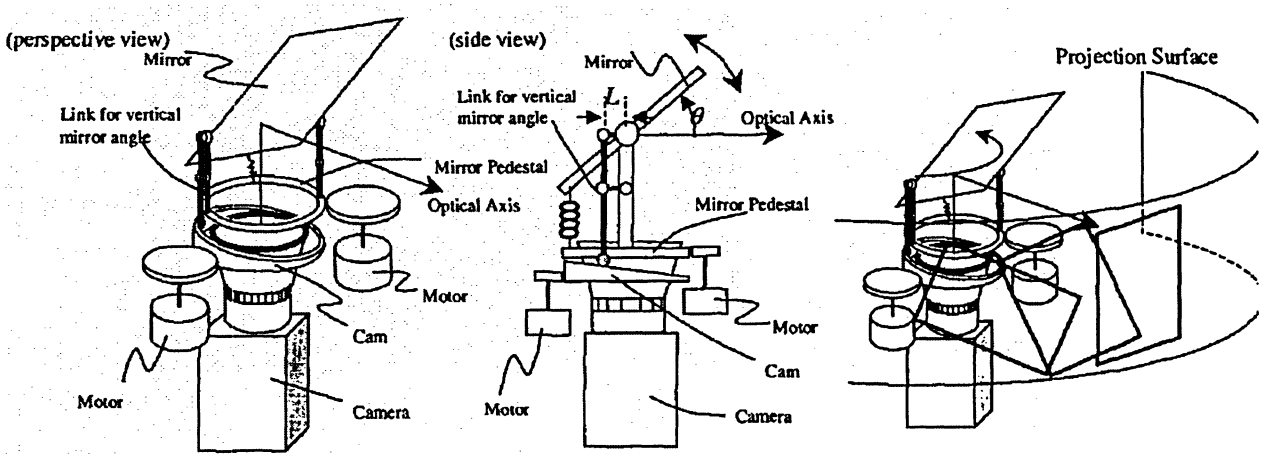


Figure 2-12. Rotating mirror architecture for panoramic vision.[12]

Another approach utilizes the panning methodology when a camera rotates up to a full view of 360° around a rotation axis, as presented by M.Barth and C.Barrows [29] as depicted in Figure 2-13. When compared to wide angle lens or mirror methods, the panoramic view produced by this technique have far greater resolution and inherently contain very high azimuth angle precision. But this approach requires sophisticated mechanics and thus is more complicated and power consuming than other techniques.

---

[11] Courtesy to [25]
[12] Courtesy to [27]

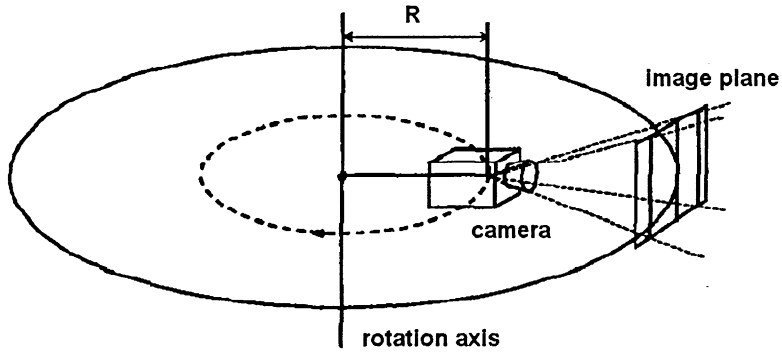


Figure 2-13. Panoramic view using panning methodology.[13]

But single-camera techniques can't provide stereo video. For this purpose a cluster of cameras needed as in a system for Real-time panorama generation and display in tele-immersive applications presented by W-K.Tang, et al. [30], as illustrated in Figure 2-14.

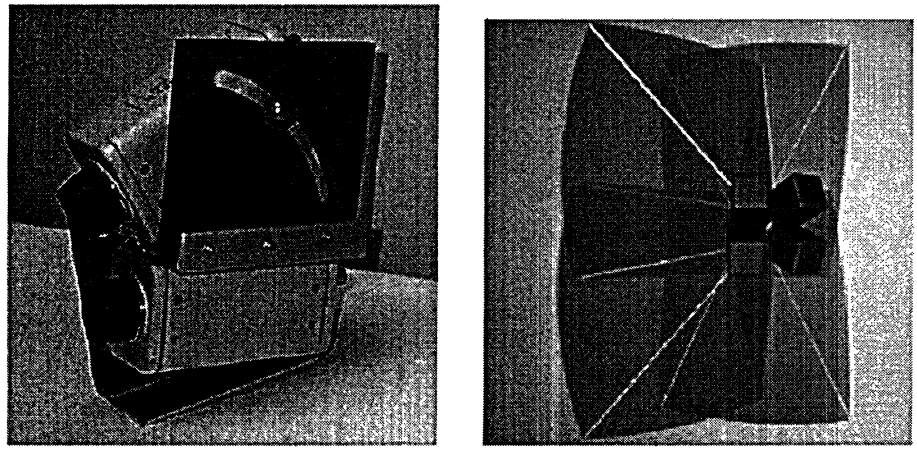


Figure 2-14. Basic capture units and overlapping of camera views.[14]

The paper also gives a view on the classification of systems for virtual reality. Most of the existing systems fall into one of three categories: geometry-based, image-based or hybrid systems. Geometry-based virtual reality system uses geometrical objects to represent scenes. Flight simulators for training pilots are a typical example. Since a real scene can be arbitrary complex, modeling a real scene may result in huge amount of data which cannot be rendered in real-time. From another hand, scenes can be represented using the image-based approach. The

[13] Courtesy to [29]
[14] Courtesy to [30]

reconstructed environment is realistic and the modeling process can be avoided, or at least simplified. Hybrid systems make use of both the geometric and image models.

Both theoretical and practical view on a multicamera setup for generating Stereo panoramic video is given by S.Tzavidas and A.Katsaggelos [31] and by S.Peleg, et al. [32]. The authors reviewed the existing techniques for panoramic video generation and give a theoretical background for the multicamera setup optimization, see Figure 2-15. There is also an issue on errors in composition of panoramic images. The analysis of errors in terms of two intrinsic camera parameters, namely the focal length and the radial distortion coefficient is given by S.B.Kang and R.Weiss [33].

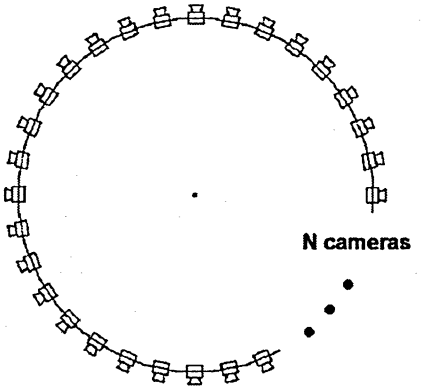


Figure 2-15. Setup for creating stereo panoramic video.[15]

When the image is generated by two or more cameras, as it happens for panoramic imaging systems, an issue of color distortion may arise since cameras can operate at different lighting conditions caused by different viewpoints. Several linear approaches for color transform and mapping are presented by G.Y.Tian, D.Gledhill and D.Taylor [34].

One of the main issues for generating stereo video for panoramic vision is the viewing angle where the image can be perceived by a viewer as stereo. It means that side viewing angles doesn't keep stereo, or depth, information. A human's viewing angle comprises two kinds of viewing fields: binocular and monocular. This issue is discussed by A. Simon, R. Smith, and R.R. Pawlicki [35] and illustrated in Figure 2-16.

[15] Courtesy to [31]

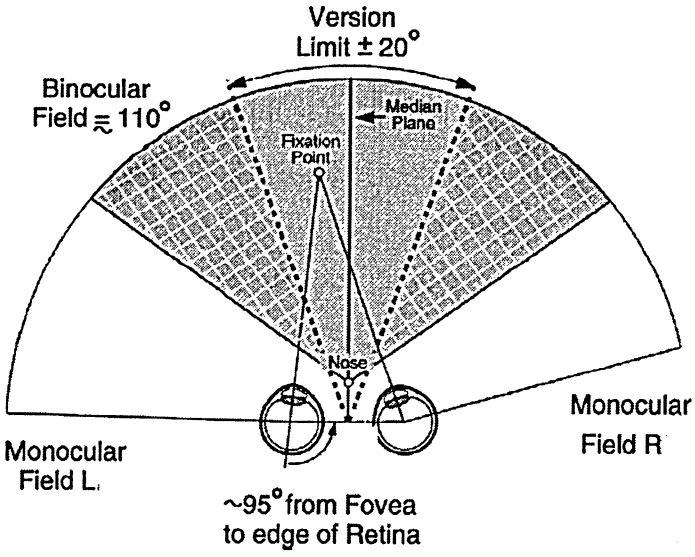


Figure 2-16. Comfortable viewing range for eye motion (version).[16]

The binocular field of human is approximately 110°. A viewer perceives comfortable 3-D view of an object if the object is located within ±20° zone from the median plane. It means that if a fixation point moves outside the version limit of ±20°, the viewer usually fixes the view by rotating his or her head in the direction of an object.

This conclusion is very important and must be taken into consideration for reproduction (e.g. projection) of stereo and panoramic images. For a fixed viewing direction there are viewing zones which doesn't carry 3-D information. It means that for certain applications we can generate stereo images for binocular field of the viewing range, and limit the projection of the side, or monocular, zones, or fields, by just "flat" one camera look. Two examples of systems for panoramic vision are CAVE™ and i-CONE™ [35]. They are designed to present a stereoscopic image with a very wide angle of view (up to 360°). These systems integrate a number of graphics channels and produce a seamless combined image which is continuously updated in accordance with view direction of single head-tracked viewer. Omni-stereo panoramic system provides stereo perception is correct in every view direction without head tracking is also reported [35].

[16] Courtesy to [35]

2.4. Elements of virtual reality technique

Interactive control elements for stereo and panoramic vision systems are considered as a part of virtual reality systems. Virtual Reality (VR) technology is regarded as a natural extension to 3-D computer graphics with advanced input and output devices [37]. A hierarchically structured constraint-based data model for solid modeling in a VR environment is presented by Y.Zhong, et al. [36]. This model is considered for CAD systems. Generating objects for VR systems is a computation intensive task a typical instrumentation setup for which includes a powerful workstation or a PC.

The issue of implementation of VR techniques for possible applications was widely discussed in 90s. In 1994, L.M.Stone, et al. [38] presented a general view on characteristics of VR systems, data in 3-D environments, augmented reality, and virtual reality market opportunities. Three important characteristics are distinguished for VR techniques. First, VRs exhibit high interactivity – there is a tight coupling between the user's actions and the feedback generated those actions. Second, they support embodiment: some sort of representation of the user in the same spatial framework as the data. Third, the VR representation is spatial in nature; virtual objects are situated in a spatial framework. This definition makes no mention of the technologies such as gloves and head-mounted displays with which VR has become popularly associated. The first head-mounted display was demonstrated by Evans and Sutherland in 1965 [39].

Virtual Reality systems are defined in 1995 by Ö. Karaçali [40] and by J.M. Zheng, et al. [41] in 1998 as advanced human-computer interface that simulates a realistic environment which allows participants to interact with it, for example, in the design of industrial products for function, assembly, styling etc. It consists of three-dimensional, interactive, computer generated environments. These environments can be models of real or imaginary worlds. Their purpose is to represent information through synthetic experience. Conceptualization of complex or abstract systems is made of possible by representing their components as symbols that give powerful sensory cues, related in some way to their meaning. The environment exists only inside a computer. Unlike graphic images, the Virtual World is generated from an object data model that can be viewed in real time.

The summary of the basic characteristics of existing real-time video systems implemented in hardware is presented in Table 2-1.

Table 2-1. Summary of existing hardware-based real-time video systems and their basic characteristics.

Video Systems	Real-time	Stereo	Possible extension to panoramic	FPGA based	System-on-chip	Video data source	Frame grabber	Elements of virtual objects synthesis	Stereo visualization
Sartori [8]	✓	-	-	✓	✓	application specific	application specific	-	n/a
Scalera [9]	✓	-	-	✓	-	n/a	n/a	-	n/a
Lienhart [10]	✓	three image sequences	n/a	✓	-	form host PC	n/a	-	n/a
Woodfill-Herzen [11]	✓	✓	n/a	✓	-	form host PC	n/a	-	n/a
Leeser [12]	✓	-	-	✓	-	standard camera module	standard frame grabber	-	n/a
Genta [13]	✓	✓	n/a	✓	-	application specific	application specific	-	n/a
Quek [19]	✓	✓	-	n/a	-	trinocular system	n/a	-	n/a
Maupu [21]	✓	3-D	n/a	(pc based)	-	standard camera module	n/a	✓	3-D graphics
Appiah-Hunter [22]	✓	-	n/a	✓	✓	standard camera module	n/a	Real-time adaptive background modeling	n/a
Woodfill-Gordon-Buck [24]	✓	✓	n/a	✓	-	standard camera module (Tyzx)	n/a	-	n/a
Proposed system	✓	✓	✓	✓	✓	application specific	application specific	✓	CRT with shutter glasses or projection screen

2.5. Conclusion

There are numerous implementations of one-channel or stereo vision systems built on programmable logic devices, mostly on FPGAs. When an application requires intensive computation resources, for example, for advanced image processing tasks, for images correlation or for disparity map generation, a part of the design migrates to a DSP based system or to a PC. From another hand, modern FPGAs have huge logic and computational resources to integrate sophisticated systems inside a single chip. For example, Xilinx Virtex-2Pro FPGA has logic resources equivalent to 10 million system gates. For Virtex-5 FPGA the number is around 33 million. These FPGAs have 1 to 4 embedded PowerPC microcontrollers. Besides, modern FPGAs can efficiently implement practically any 8-bit microcontroller, and available soft cores support popular instruction sets such as the Microchip PIC, Intel 8051, Atmel AVR, Motorola 6502, 8080, and Zilog Z80 microcontrollers [42]. There are also specifically designed soft core microcontrollers like MicroBlaze and PicoBlaze for Xilinx FPGAs.

For multi-mode real-time systems which process a number of data streams (two data streams for stereo vision) FPGA-based approach is the only possible one which can incorporate all necessary tasks in a SoC. The set of tasks related to stereo or panoramic vision includes: multi-channel data capture, deep parallel and highly pipelined data processing (including data processing for synthesis of virtual 3-D objects).

The advantages of this approach are the following:

a) It simplifies hardware design since all the design sits inside one single chip, and thus there are no PCB effects (cross talks and delays). The complexity of the design sits inside the FPGA which can operate at core speed up to 500MHz.
b) It increases reliability (less hardware is employed) and decreases power consumption;
c) The system is reconfigurable, i.e. the design can be easily changed if other algorithms must be implemented on the same platform.
d) The system can be dynamically reconfigurable [43, 44, 45] when a library of computational specific soft IP cores can be stored in on-board non-volatile memory and be loaded into the FPGA chip [46, 47] when necessary [48, 49].

e) The system can be built using a self-restoration mechanism [50]. This technique requires a special approach for the system design.

As a result, a complex system can be implemented utilizing less hardware resources than the system built using traditional approaches, i.e. when all modules which may be needed for a particular task sit on the board. What's more, for processing of a number of data streams which can change in time and on which various processing algorithms can be applied the system must provide high level of parallelism and must be reconfigurable. It means that the system has to be implemented as a SOC and thus only in SRAM-based FPGA.

For existing systems listed in the table above the task of visualization is not of main concern since they mostly perform a kind of intermediate processing between the capture system and host PC. For the proposed system visualization is one of the tasks which are implemented also in hardware. Visualization can be realized in two modes: using shutter glasses and CRT monitor for video output, or projecting two images through polarization filters on a special screen which preserves the polarization of the light beams. A viewer can perceive stereo image through correspondingly polarized glasses.

3. THEORY AND METHODOLOGY

3.1. Introduction

Extraction of relevant visual information means that all video sources (two for stereo vision) must be properly oriented in space and generate images with minimal color distortion which may be caused by different lighting conditions. Besides the mechanical adjustment of the video cameras and lenses, correlation methods are employed to calibrate generated images. The methods for correspondence matching are typically based on calculation of sum of absolute differences (SAD) [17]. This problem becomes of primary importance for navigation systems where precise disparity maps are expected to be generated. For the systems aimed at just visualization of stereo video the exact correspondence of images is not as important and good results can be obtained by simple mechanical adjustment of cameras or projecting devices orientation.

The proposed system's task is to integrate real-time stereo video with synthesized elements of 3-D virtual objects. For synthesis of graphical objects which should be a part of the background image it is important to know the correct motion of the camera and its calibration [51]. Otherwise, the synthesized object won't look natural in the scene. The major issue for the present thesis is to consider the formal approach to virtual 3-D objects generation applicable for a system-on-chip. These images are not tied to the background (they can be used just as virtual buttons or cursors). Thus, precise calibration and correlation of stereo images, as well as the motion of cameras are not within the scope of this thesis work and the mathematical background for this task is not considered. From another side, the task of the implementation includes image capture by video cameras and image color decoding since available on the market CMOS Image Sensors provide data in so-called Bayer pattern so that each pixel (elementary image element) keeps only one color component data. The other color components for each pixel should be taken from the adjacent elements of the image. Certain approaches can be applied to this task realization, and possible solutions should be analyzed and optimized.

Therefore, two theoretical aspects that must be considered for the realization of the presented system can be distinguished and must be analyzed.

First – synthesis of virtual 3-D images. For this task realization the geometry of 3-D objects and their transformation in space as a function of time must be analyzed. Certain mathematical models are resulting from the analysis, and

Second – based on the mathematical models and algorithms which must be implemented a set of possible architectural solutions for implementation of the system's tasks can be obtained based on special methods of architectural synthesis. After that, optimization of the system architecture must be done with the purpose to design a system with optimal resources and performance characteristics.

3.2. Synthesis of virtual 3-D objects

In engineering applications, the picture is not the final result. Instead, the model is the result and the picture is only a means of communicating information about the model [6].

The basic point is that the visible world around us has three dimensions and can be modeled by Euclidian three-dimensional space, or 3-D space. To represent a position in 3-D points are used. The notation ***A(x, y, z)*** stands for a point in 3-D space with the coordinates ***x, y*** and ***z*** (known as Cartesian coordinates) each of which is a distance from the reference point, or the beginning of coordinates, along the orthogonal axes X, Y, and Z. A line segment connecting two points is characterized by two pairs of coordinates. A scalar is a value that describes only magnitude, for example how far the point is from the origin. A vector is a directed line segment.

Points, scalar values and vectors are tools for virtual world creation. An object in 3-D can be represented as a number of points (vertices) connected by line segments. The elementary polygons of an object (or body) create the surface of object. Objects in virtual world can move and change their orientation. If there is a restriction for a body requiring that the distances between all of its points can't change, then we deal with a rigid body. It possesses six degrees of freedom in 3-D space. A single degree of freedom is represented by a scalar value. When it is changed, the change in the object state occurs. In 3-D space an object can move in any of three possible directions, i.e. three ***translations*** are possible along each of axes. Besides, three different ***rotations*** around axes are possible. Thus, there are six degrees of freedom. For a rigid body any transformation can be expressed by two different types of transformations: translation and rotation. If the shape of an object changes too, other kind of transformations must be used.

The most common transformations related to structure deforming are scaling and shearing when the size and the shape of the object change [52]. The methodology of solid objects presentation and transformation in 3-D space and on the projection plane is presented in [54].

An important aspect is how to represent an object in space and how to select a reference system for the description of object transformation in space. Figure 3-1 illustrates possible views on the space. First reference system is the one which is tied to the object itself (object space). The object is located somewhere in reference to other objects and viewers, so we can define a space including all objects (world space). A viewer has his or her own system of reference tied to his or her eye (actually, there are two view spaces for each eye). Finally, we can distinguish a view of the object on a projection plane, or screen (screen space).

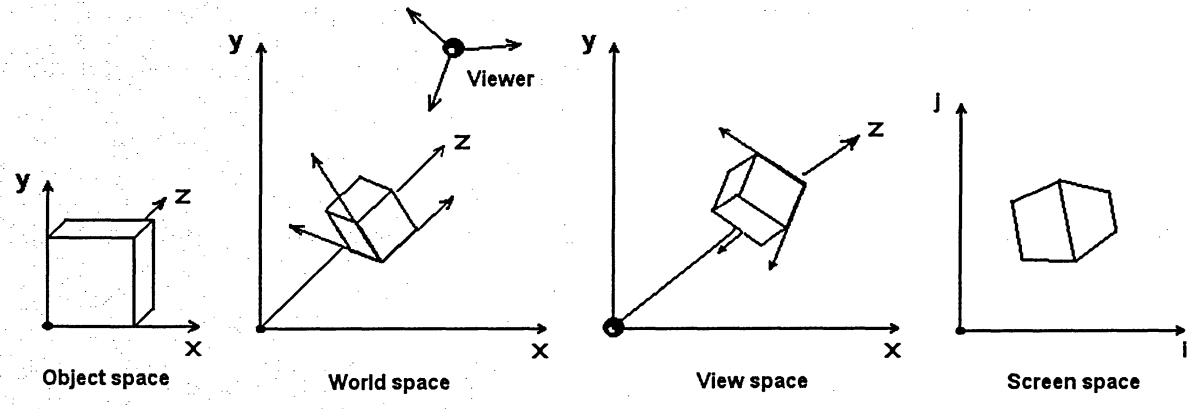


Figure 3-1. Reference systems classification.[17]

In this section the formal approaches to translation and rotation transformation are considered. For the visualization of the synthesized images the projection transformations must be considered as well as the techniques for the pixelwise presentation (or rasterizing) of the line segments in the projection plane. The last issue to be analyzed is visibility of a 3-D figure surfaces. Definitely, some surfaces of a 3-D body are turned to a viewer so that he or she can see them while other surfaces are hidden, i.e. located behind visible surfaces. For a certain surface the visibility aspect can be different for stereo images (i.e. for right and left eye views). A view space will be used for the description of an object location in space. Screen space is used for all manipulations with the projected images an object.

[17] Courtesy to [54]

3.2.1. Translation transformation

Translation transformations can be considered in two ways: either as a transformation of a set of points in a static coordinate system or as a transformation of the space as illustrated in Figure 3-2 for 2-D space. Both approaches are productive and convenient for different situations.

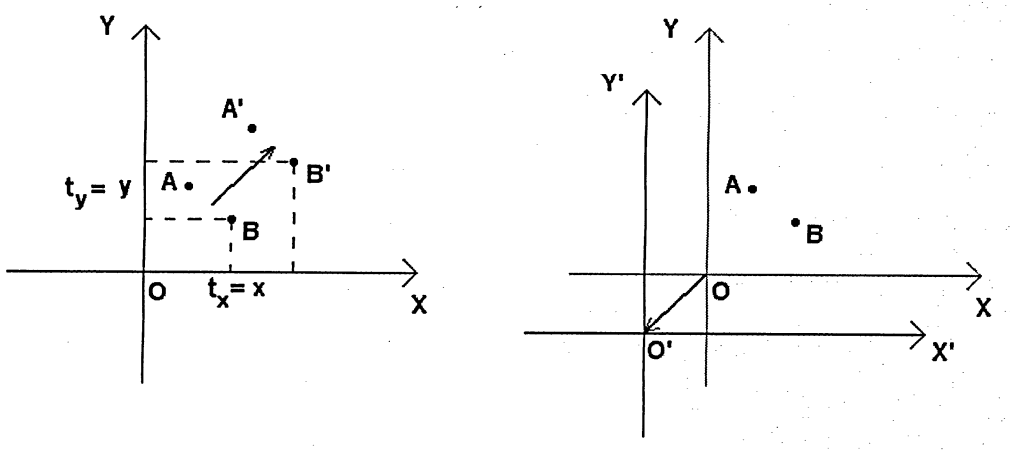


Figure 3-2. Two approaches to translation transformations: translation of a set of points and translation of space.

New coordinates of the displaced points in 3-D space can be expressed as

$$x' = x + t_x \qquad y' = y + t_y \qquad z' = z + t_z, \qquad \text{[3-1]}$$

And the displacement vector can be defined as $\bar{d}(t_x, t_y, t_z)$.

To represent the translation in the matrix form specially adjusted coordinate vectors are used. A unit constant is added to the three coordinate so that a point in 3-D space is represented as $[x \quad y \quad z \quad 1]$. It allows using matrix multiplication for the translation transformation:

$$[x \quad y \quad z \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} = [x + t_x \quad y + t_y \quad z + t_z \quad 1] \qquad \text{[3-2]}$$

So, the translation transformation for a vertex of an object will require 3 addition operations, as illustrated in Table 3-1. For example, for a cube it gives 24 additions, for tetrahedron – 12.

Table 3-1. Number of arithmetic operations required for translation transformation per vertex.

Translation	*Number of operations*			
Operation	**ADD**	**SUB**	**MUL**	**DIV**
Coordinate X computing	1	0	0	0
Coordinate Y computing	1	0	0	0
Coordinate Z computing	1	0	0	0
TOTAL	3	0	0	0

3.2.2. Rotation transformation

The second basic type of transformation is rotation of an object in space. Any rotation can be composed of three components which are rotation about each axis x, y, and z. Consider a 2-D case for finding the coordinates of a point when the system of references rotates counterclockwise by angle α, as depicted in Figure 3-3.

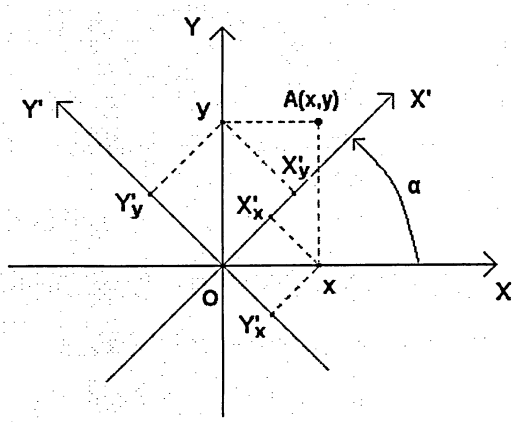


Figure 3-3. Rotation in 2-D plane.[18]

The projections of x and y on the new axis X' and Y' can be expressed as

$$X'_x = x \cdot \cos(\alpha); \qquad Y'_x = (-x) \cdot \sin(\alpha);$$
$$X'_y = y \cdot \sin(\alpha); \qquad Y'_y = y \cdot \cos(\alpha). \qquad \text{[3-3]}$$

And therefore the coordinates of the point A in the new rotated system of references are

$$x' = X'_y + X'_x = y \cdot \sin(\alpha) + x \cdot \cos(\alpha)$$
$$y' = Y'_y + Y'_x = y \cdot \cos(\alpha) - x \cdot \sin(\alpha) \qquad \text{[3-4]}$$

The component Y'_x has negative sign because x is projected on the negative side of the rotated axis Y'.

[18] Courtesy to [54]

Now we can express the rotation of a point in 3-D space. Three factors affecting the form of the 3-D rotation must be taken into account: the kind of reference system; directions of positive rotations; and the order in which rotations are applied. A certain set of these factors must be used in a particular application to obtain adequate result of space transformations. The factors are illustrated in Figure 3-4. The rotation angles are defined as ***roll*** for the *XY* plane turns around the *Z* axis, ***pitch*** for the *ZY* plane turns around the *X* axis, and ***pitch*** for the *ZX* plane turns around the *Y* axis. For the further discussion the Left-handed reference system and the positive direction of rotation as depicted in Figure 3-4 are employed.

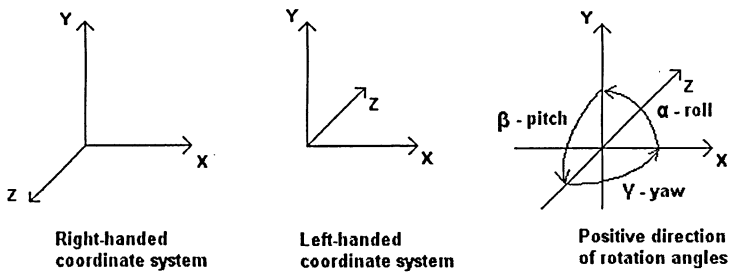


Figure 3-4. Right- and Left-handed coordinate systems and positive direction of rotation angles.

The order of rotations is also important. In general case, applying different sequence of the rotations to an object might cause different resulting orientation of object in 3-D space. With respect to a viewer's head typical order of rotation the sequence γ (yaw) – β (pitch) – α (roll) is widely used, as shown in Figure 3-5.

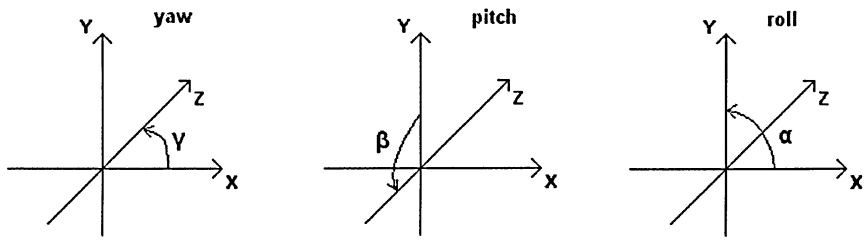


Figure 3-5. Three consecutive rotations: yaw – pitch – roll.[19]

The resulting expressions for three consecutive rotations in accordance with the sequence γ–β–α can be obtained as follows.

[19] Courtesy to [54]

$$\begin{cases} x' = z\sin(\gamma) + x\cos(\gamma) \\ y' = y \\ z' = z\cos(\gamma) - x\sin(\gamma) \end{cases} \quad \begin{cases} x'' = x' \\ y'' = y'\cos(\beta) - z'\sin(\beta) \\ z'' = y'\cos(\beta) + z'\cos(\beta) \end{cases} \quad \begin{cases} x''' = y''\sin(\alpha) + x''\cos(\alpha) \\ y''' = y''\cos(\alpha) - x''\cos(\alpha) \\ z''' = z'' \end{cases} \qquad \textbf{[3-5]}$$

These nine formulas can be applied to the coordinates (x, y, z) and produce as a result the transformed coordinates (x''', y''', z''').

The matrix representation of three consecutive rotations [3-5] is;

$$[x''' \quad y''' \quad z'''] = [x \quad y \quad z] \cdot \begin{bmatrix} \cos(\gamma) & 0 & -\sin(\gamma) \\ 0 & 1 & 0 \\ \sin(\gamma) & 0 & \cos(\gamma) \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & \sin(\beta) \\ 0 & -\sin(\beta) & \cos(\beta) \end{bmatrix} \cdot \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \textbf{[3-6]}$$

It's possible to express the resulting coordinates (x''', y''', z''') directly from the original coordinates. The first step is to get rid of x', y', z':

$$\begin{cases} x'' = z\sin(\gamma) + x\cos(\gamma) \\ y'' = y\cos(\beta) - z\cos(\gamma)\sin(\beta) - x\sin(\gamma)\sin(\beta) \\ z'' = y\sin(\beta) + z\cos(\gamma)\cos(\beta) - x\sin(\gamma)\cos(\beta) \end{cases} \qquad \textbf{[3-7]}$$

The second step is to eliminate x'', y'', z'':

$$\begin{cases} x''' = x[\sin(\gamma)\sin(\beta)\sin(\alpha) + \cos(\gamma)\cos(\alpha)] + y[\cos(\beta)\sin(\alpha)] + z[\sin(\gamma)\cos(\alpha) - \cos(\gamma)\sin(\beta)\sin(\alpha)] \\ y''' = x[\sin(\gamma)\sin(\beta)\cos(\alpha) - \cos(\gamma)\sin(\alpha)] + y[\cos(\beta)\cos(\alpha)] + z[-\cos(\gamma)\sin(\beta)\cos(\alpha) - \sin(\gamma)\sin(\alpha)] \\ z''' = x[-\sin(\gamma)\cos(\beta)] + y[\sin(\beta)] + z[\cos(\gamma)\cos(\beta)] \end{cases} \qquad \textbf{[3-8]}$$

All the coefficients in the square brackets can be computed only once for a given rotation of an object (i.e. for all the vertices of the object). The resulting equations can be rewritten as:

$$\begin{cases} x''' = m_{x1}x + m_{y1}y + m_{z1}z \\ y''' = m_{x2}x + m_{y2}y + m_{z2}z \\ z''' = m_{x3}x + m_{y3}y + m_{z3}z \end{cases} \quad \text{or} \quad [x''' \quad y''' \quad z'''] = \begin{bmatrix} m_{x1} & m_{y1} & m_{z1} \\ m_{x2} & m_{y2} & m_{z2} \\ m_{x3} & m_{y3} & m_{z3} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \textbf{[3-9]}$$

The expressions for the coefficients m are:

$$\begin{aligned} m_{x1} &= \sin(\gamma)\sin(\beta)\sin(\alpha) + \cos(\gamma)\cos(\alpha) \\ m_{y1} &= \cos(\beta)\sin(\alpha) \\ m_{z1} &= \sin(\gamma)\cos(\alpha) - \cos(\gamma)\sin(\beta)\sin(\alpha) \\ m_{x2} &= \sin(\gamma)\sin(\beta)\cos(\alpha) - \cos(\gamma)\sin(\alpha) \\ m_{y2} &= \cos(\beta)\cos(\alpha) \\ m_{z2} &= -\cos(\gamma)\sin(\beta)\cos(\alpha) - \sin(\gamma)\sin(\alpha) \\ m_{x3} &= \sin(\gamma)\cos(\beta) \\ m_{y3} &= \sin(\beta) \\ m_{z3} &= \cos(\gamma)\cos(\beta) \end{aligned} \qquad \textbf{[3-10]}$$

Rotation transformations are based on trigonometric functions. A common approach for evaluation of trigonometric functions is to use power series. A Maclaurin power series for sine and cosine functions in the narrow domain around zero are:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots \quad \text{and} \quad \cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots \qquad \textbf{[3-11]}$$

Using only two terms of the Maclaurin power series for computing sine and cosine [3-11] in the domain $[-\pi/4, \pi/4]$ gives acceptable results with the error within 2.19% for cosine and 0.34% for sine:

$$\sin(x) = x - \frac{x^3}{3!}, \quad \cos(x) = 1 - \frac{x^2}{2!} \qquad \textbf{[3-12]}$$

For the angles outside of the narrow domain $[-\frac{\pi}{4}, \frac{\pi}{4}]$ the regular nature of *sin* and *cos* functions can be used: the period of the functions is 2π, that is $\sin(x) = \sin(x \pm 2\pi)$ and $\cos(x) = \cos(x \pm 2\pi)$; the relation between *sin* and *cos* functions: $\sin(x) = \cos(x - \frac{\pi}{2})$; and, besides, $\sin(x) = \sin(\pi - x)$, $\sin(x) = -\sin(-x)$, $\cos(x) = -\cos(\pi - x)$, and $\cos(x) = \cos(-x)$.

So, for the rotation computation the system will need to perform operations as it is indicated in Table 3-2Table 3-2.

Table 3-2. Number of arithmetic operations required for rotation transformation computation per vertex.

Rotation	*Number of operations*			
Operation	**ADD**	**SUB**	**MUL**	**DIV**
sin(α)	0	1	3	1
sin(β)	0	1	3	1
sin(γ)	0	1	3	1
cos(α)	0	1	2	1
cos(β)	0	1	2	1
cos(γ)	0	1	2	1
m_{x1}	1	0	3	0
m_{y1}	0	0	1	0
m_{z1}	0	1	3	0
m_{x2}	0	1	3	0
m_{y2}	0	0	1	0
m_{z2}	0	1	3	0
m_{z1}	0	0	1	0
m_{z2}	0	0	0	0
m_{z3}	0	0	1	0
Coordinate X computing	2	0	3	0
Coordinate Y computing	2	0	3	0
Coordinate Z computing	2	0	3	0
TOTAL:	7	9	40	6

3.2.3. Projection transformation

Projection transformation is mapping 3-D world coordinates into 2-D screen. For computer graphics two methods are of particular interest: parallel and perspective projection. The technique of both types of projections is illustrated in Figure 3-6.

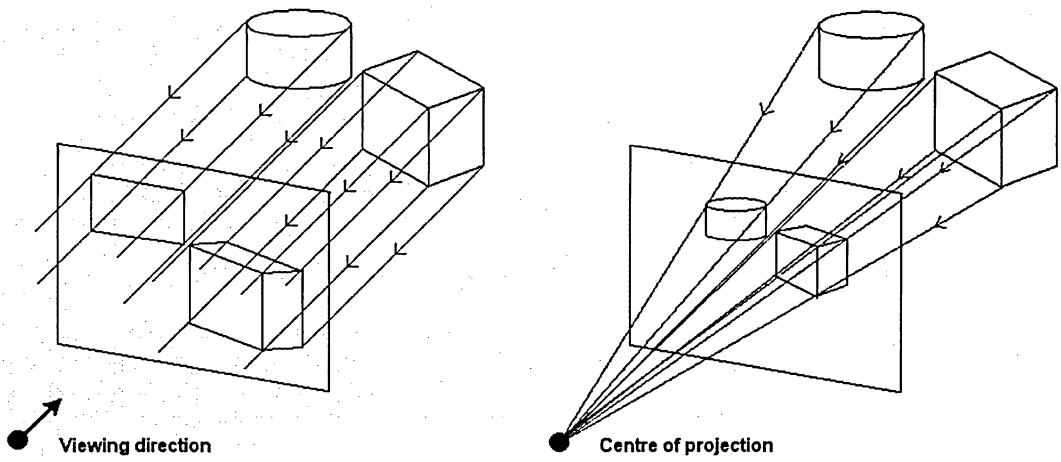


Figure 3-6. Parallel and perspective projection schemes.[20]

The value of angle at which the parallel projection is based defines whether the projection is orthogonal (right angle, 90°) or oblique (all other values of angle). For vision system only perspective projection is of interest since it presents a view of an object that corresponds to how a human eye sees real objects in surrounding 3-D space.

There are two viewing approaches for projected images generation: world-to-screen method and screen-to-world method, see Figure 3-7. The first method creates an image of an object of the 3-D space by projecting volume figures onto the screen space. The second methods defines projected image by tracing the rays form the viewer eye into the space. A ray is cast through every pixel on the screen and the intersections of the ray with all the object's primitives are computed. An intersection closest to the viewer is depicted on the screen. The resulting images are composed of geometric primitives, polygons.

[20] Courtesy to [54]

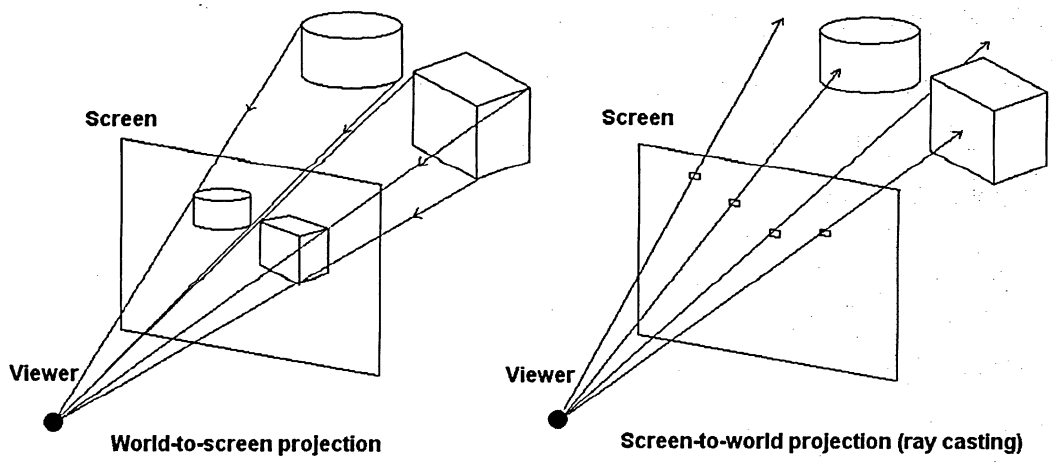


Figure 3-7. Two types of projection methods: world-to-screen and screen-to-world.

The advantage of the screen-to-world is that the task of hidden or blocked surfaces or objects is resolved by itself if this method is used. But it is much more computation intensive than the world-to-screen method since a ray is cast through every pixel of the screen. The screen with resolution 640 x 480 pixels will require more than 300,000 rays to be processed. This is why for our system the world-to-screen method is employed.

The geometry of perspective projection for the world-to-screen method is illustrated in Figure 3-8.

The x coordinate of the projected point A' can be calculated by the following formula:

$$x' = focus \cdot x / z. \qquad \text{[3-13]}$$

The same formula applies for the y coordinate in the projection plane:

$$y' = focus \cdot y / z. \qquad \text{[3-14]}$$

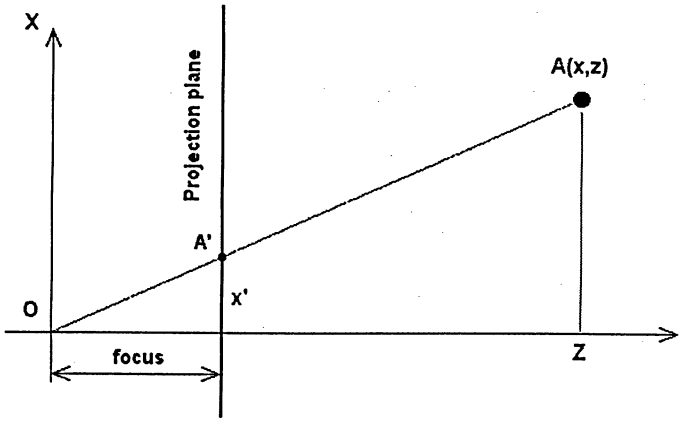


Figure 3-8. Geometry of perspective projection.

The Z coordinate can be used for further depth interpretations, for example when one object is located behind another and thus is not visible or partially visible.

The matrix representation of the perspective transformation can be expressed as:

$$[x \quad y \quad z \quad 1]\cdot\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/focus \\ 0 & 0 & -1 & 0 \end{bmatrix} = [x \quad y \quad -1 \quad z/focus] \qquad [3\text{-}15]$$

To obtain the resulting point coordinates normalized, i.e. the last entry in the matrix form to be equal '1', the matrix [3-15] can be multiplied by the value focus/z:

$$[x \quad y \quad -1 \quad z/focus] = [x\cdot focus/z \quad y\cdot focus/z \quad -focus/z \quad 1] \qquad [3\text{-}16]$$

If for a certain application the depth information is of no importance, the Z coordinate can be discarded, thus the perspective transformation matrix becomes:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/focus \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad [3\text{-}17]$$

For two viewing points (for binocular, or stereo vision) projection onto two planes must be considered [53]. That is, we must obtain two projected images, one for each viewing point as depicted in Figure 3-9.

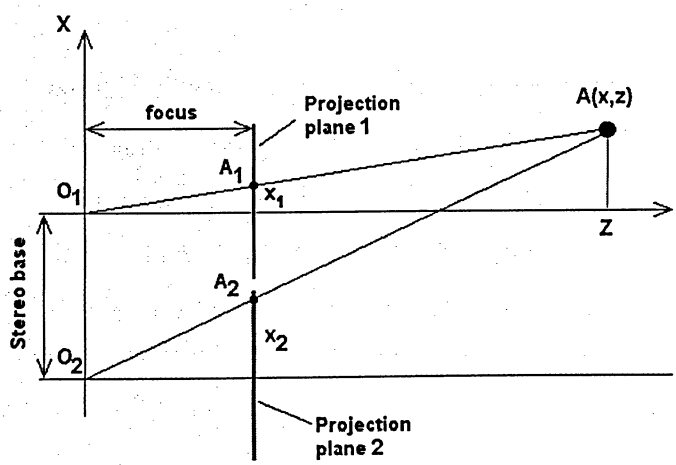


Figure 3-9. Geometry of perspective projection for stereo vision.

For stereo vision systems we must perform all necessary calculations for two independent projected images, i.e. for each viewing point O_1 and O_2. The technique for both images is the same.

The computational complexity of the perspective projection is presented in Table 3-3.

Table 3-3. Number of arithmetic operations required for perspective projection computation per vertex.

Perspective projection	*Number of operations*			
Operation	**ADD**	**SUB**	**MUL**	**DIV**
Coordinate X computing	0	0	1	1
Coordinate Y computing	0	0	1	1
TOTAL	0	0	2	2

The number of arithmetic operations can be decreased since intermediate results can be used for different stages of computing.

3.2.4. Line segments representation for projected images

Representation of line segments on a projection plane is a task which may be resolved straightforward using the line equation $y = ax + b$, but to minimize computing resources involved in this kind of operation another approaches must be considered. The methodology is presented in [7] and [54].

A segment AB on the projection plane and the image bitmap layout are represented on the left and right sides of Figure 3-10 correspondingly. The image on the projection plane is characterized by a certain number of elements in both directions, $size_x$ and $size_y$. For example, for a standard VGA device the number of elements for any image is 640 (in horizontal direction) by 480 (in vertical direction). It means that a line segment must be represented by a certain number of pixels, i.e. discrete image elementary elements.

ΔX and ΔY values are the dimensions of the segment AB along the axes X and Y:

$$\Delta X = X_{end} - X_{start}$$
$$\Delta Y = Y_{end} - Y_{start} \qquad [3\text{-}18]$$

$$\frac{x}{y} = \frac{\Delta X}{\Delta Y} \Rightarrow y = \frac{x \cdot \Delta Y}{\Delta X} \qquad [3\text{-}19]$$

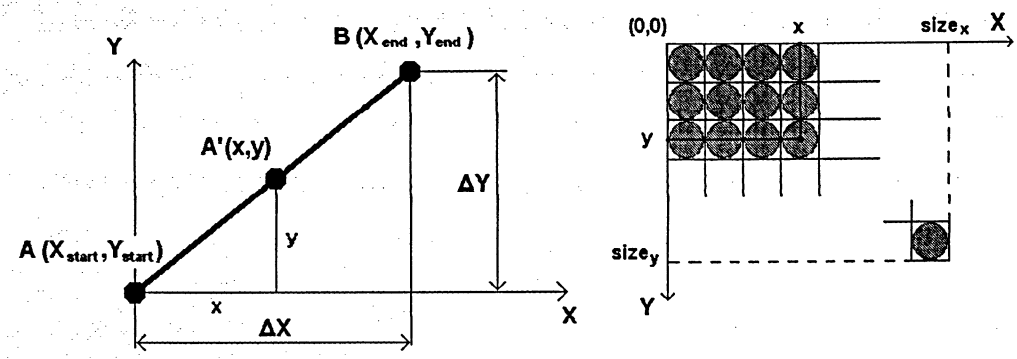


Figure 3-10. A line segment on the projection plane and image bitmap layout.[21]

Using the last formula we can obtain the values of y for all values of x for a given interval ΔX. There is, though, a situation when the resulting result set of pixels representing a line $y=ax+b$ will not represent a continuous path on the screen, as depicted on the right side of Figure 3-11. The left side image is a desired situation, when we have a continuous path of pixels on the screen.

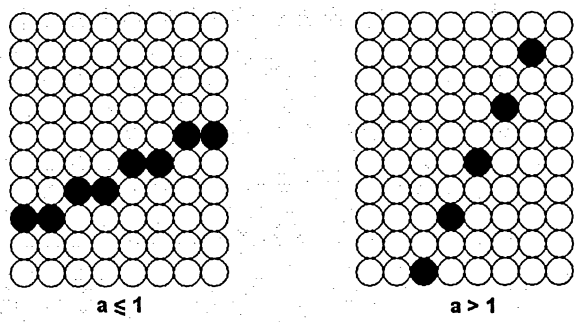


Figure 3-11. Line rasterizing for different values of the slope ratio 'a'.

For the second case the solution is to calculate the x values as a function of y, i.e.

$$x = \frac{y \cdot \Delta X}{\Delta Y} \qquad \text{[3-20]}$$

This method requires one multiplication and one division per each point. Iterative computation technique using forward differences is often used for rasterizing polynomial curves. The idea of forward differences approach is that the value of a function in a point $x+\delta$ can be presented as

$$y(x+\delta) = y(x) + dy, \qquad \text{[3-21]}$$

[21] Courtesy to [54]

That is it is equal to the sum of the function value at point x and the function's forward difference on the interval δ. Since our interest is rasterizing of line segments, ***dy*** has a constant value.

The forward difference ***dy*** can be expressed from [3-21] and [3-19] as follows:

$$dy = y(x+\delta) - y(x) = \frac{\Delta Y}{\Delta X}(x+\delta) - \frac{\Delta Y}{\Delta X}x = \frac{\Delta Y \cdot x + \Delta Y \cdot \delta - \Delta Y \cdot x}{\Delta X} = \frac{\Delta Y}{\Delta X} \cdot \delta \qquad [3\text{-}22]$$

For rasterizing on a graphical display the value of δ is '1'. That is, the corresponding value of the function y for the next discrete x, i.e. for $x+1$, is

$$y(x+1) = y(x) + dy \qquad [3\text{-}23]$$

Where ***dy*** is defined by the previous expression [3-22] as: $dy = \frac{\Delta Y}{\Delta X} \cdot \delta$.

One addition is needed to produce ***dy***. The value of $\Delta Y/\Delta X$ is constant for the whole line segment and has to be computed only once. The computations, though, involve fractional numbers arithmetic.

J.Bresenham from IBM presented his algorithm (in 1965) which significantly reduces the computational resources needed. In his approach the firs step is to find the bigger interval, either ΔX or ΔY, and to iterate the coordinate of this (bigger) interval [7, 54]. A signaling variable is introduced which indicates whether the value of a smaller interval should be incremented at a given point. For the case when $\Delta X > \Delta Y$, $X_{start} < X_{end}$ and $Y_{start} < Y_{end}$ the illustration of the technique is presented in Figure 3-12.

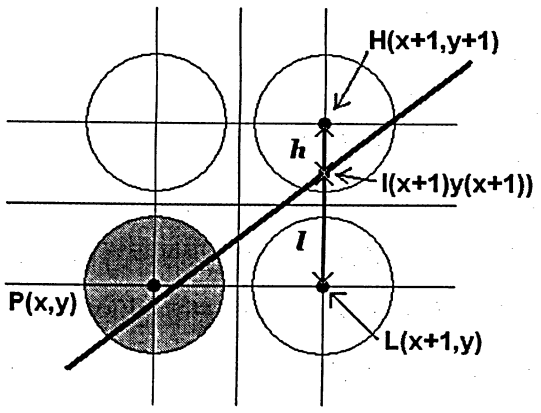


Figure 3-12. Bresenham's algorithm: the line passing through the pixel grid.[22]

[22] Courtesy to [54]

Pixel ***P(x,y)*** was just rendered at coordinates ***(x,y)***. The next pixel can be rendered either ***H(x+1,y+1)***, i.e. higher, or ***L(x+1,y)***, i.e. lower. ***I(x+1,y(x+1))*** stands for actual (fractional) imaginary "location" of the pixel, i.e. ***y(x+1)*** corresponds to the intersection point value. As a result of intersection two segments can be distinguished: ***h*** and ***l***. By comparing the lengths of these segments we can decide whether we should advance to the higher or lower pixel, ***H*** or ***L***, to render the next pixel at the coordinate ***x+1***.

The variables ***h*** and ***l*** can be expressed as follows:

$$h = (y+1) - y(x+1) \Rightarrow h = y + 1 - \frac{\Delta Y}{\Delta X}(x+1) \qquad \textbf{[3-24]}$$

$$l = y(x+1) - y \Rightarrow l = \frac{\Delta Y}{\Delta X}(x+1) - y \qquad \textbf{[3-25]}$$

To evaluate whether $h > l$ or $l < h$ the difference $l-h$ can be examined:

$$l - h = 2\frac{\Delta Y}{\Delta X}(x+1) - 2y - 1 \qquad \textbf{[3-26]}$$

If $l-h>0$, i.e. $l>h$, the intersection point ***I*** is closer to the point ***H*** and the corresponding pixel should be selected for rendering. Correspondingly, ***L*** pixel should be plotted when $l<h$ (or $l \leq h$ to cover all possible situations). The last expression [3-26] can be rewritten as:

$$\Delta X(l-h) = 2\Delta Y \cdot x + 2\Delta Y - 2\Delta X - \Delta X \qquad \textbf{[3-27]}$$

The sign of $\Delta X(l-h)$ is the same as the sign of $(l-h)$ since ΔX is assumed to be positive.

The values of $d = \Delta X(l-h)$ for two consecutive iterations are:

$$\begin{aligned} d_i &= 2\Delta Y \cdot x_{i-1} - 2\Delta X \cdot y_{i-1} + 2\Delta Y - \Delta X \\ d_{i+1} &= 2\Delta Y \cdot x_i - 2\Delta X \cdot y_i + 2\Delta Y - \Delta X \end{aligned} \qquad \textbf{[3-28]}$$

The initial value of ***d*** at the point ***(x=0, y=0)*** is:

$$d_1 = 2\Delta Y - \Delta X \qquad \textbf{[3-29]}$$

The goal of the Bresenham's method is to find the value of d_{i+1} assuming that the value d_i form the previous iteration is known. The difference $d_{i+1} - d_i$ can be found from the equations [3-28] as:

$$d_{i+1} - d_i = 2\Delta Y \cdot x_i - 2\Delta X \cdot y_i - 2\Delta Y \cdot x_{i-1} + 2\Delta X \cdot y_{i-1} = 2\Delta Y(x_i - x_{i-1}) - 2\Delta X(y - y_{i-1}) \qquad \textbf{[3-30]}$$

Taking into account that if for the previous iteration the higher pixel, ***H***, was plotted, then

$$x_i - x_{i-1} = 1 \quad \text{and} \quad y_i - y_{i-1} = 1$$

and
$$d_{i+1} - d_i = 2\Delta Y - 2\Delta X \Rightarrow d_{i+1} = d_i + 2\Delta Y - 2\Delta X \qquad \textbf{[3-31]}$$

The second case, when the lower pixel, ***L***, was selected for plotting of the previous pixel, i.e.

$$x_i - x_{i-1} = 1 \quad \text{and} \quad y_i - y_{i-1} = 0$$

Therefore $$d_{i+1} - d_i = 2\Delta Y \Rightarrow d_{i+1} = d_i + 2\Delta Y \quad [3\text{-}32]$$

Figure 3-13 presents the Bresenham's algorithm.

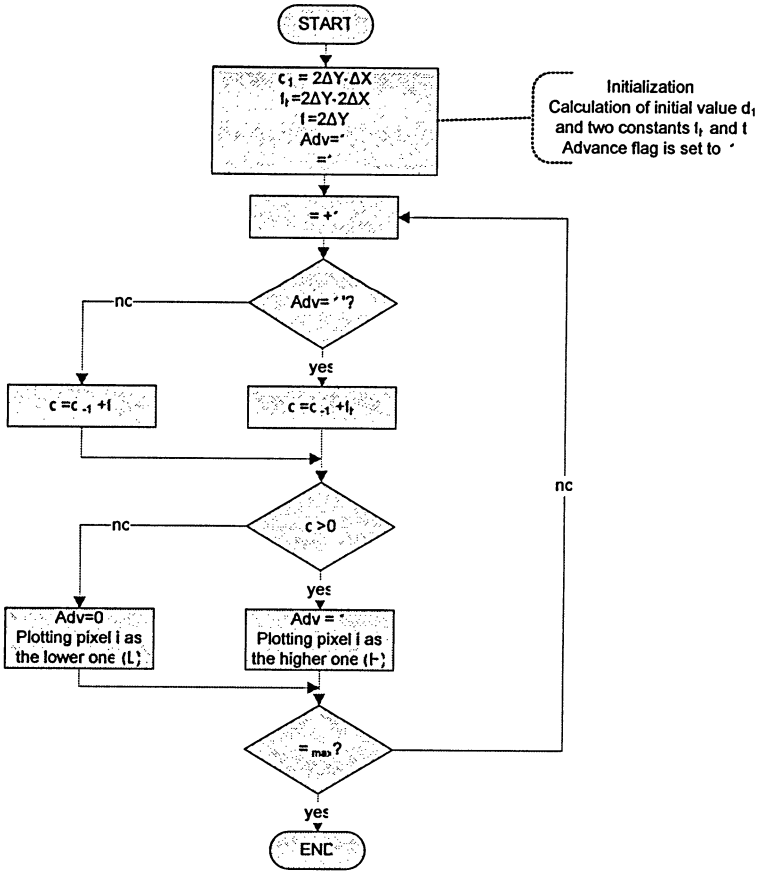


Figure 3-13. Bresenham's algorithm.

For each pixel rendering only one addition and comparison operations are needed. For each line segment the constants t_h and t_l as well as the initial value d_1 are computed only once.

Table 3-4. Number of arithmetic operations required for the Bresenham's algorithm implementation.

Bresenham's algorithm	*Number of operations*				*Note*
Operation	**ADD**	**SUB**	**MUL**	**DIV**	
ΔX computing	0	1	0	0	*Once per line segment*
ΔY computing	0	1	0	0	
d_1 computing	0	1	1	0	
t_h computing	0	1	2	0	
t_l computing	0	0	1	0	
d_i computing	1	0	0	0	*Once per pixel*
TOTAL	1	4	4	0	

Table 3-4 presents the number of arithmetic operations needed for the Bresenham's algorithm implementation per pixel. Only d_i is computed for each pixel. It requires one addition operation. But the amount of computation operations completely depends on the complexity of the object and thus its projected image. The number of pixels to be processed can be hundreds or even thousands. This is not a problem for the proposed system implementation since, as it will be illustrated later in the chapter 4, there is a sufficient time slot to process very complex objects. This time slot is at least of one whole frame duration.

3.2.5. Rasterizing polygons

As soon as the line segments composing the projected image of a 3-D object are plotted, the rasterizing of polygons outlined by these line segments can be done.

A 3-D body can be presented as a set of conjunct polygons as illustrated in Figure 3-14. For example, a cube is combined of six squares and a tetrahedron – of 4 triangles.

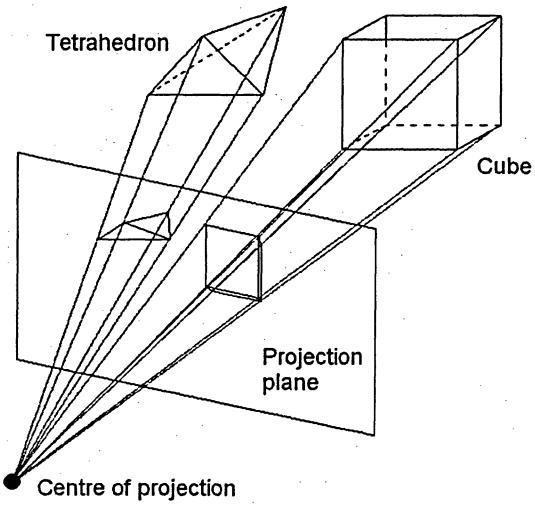


Figure 3-14. Objects in 3-D space and their perspective projection views.

The projected images of 3-D bodies are seen as a set of polygons, in this case of triangles for the tetrahedron and parallelograms for the cube.

In general case the polygons can be of two types: ***convex*** and ***concave***, as depicted in Figure 3-15. The first one has a feature that all lines connecting any two points inside a polygon doesn't leave its boundaries. This feature doesn't relate to concave.

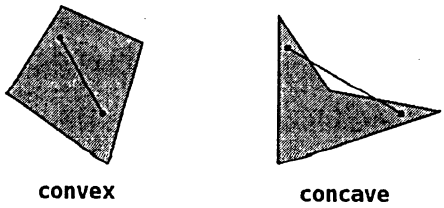


Figure 3-15. Convex and concave polygons.

A concave polygon can always be represented as a set of convex polygons. Furthermore, any polygon on the projection plane can be represented as a set of triangles, so it's possible to consider only triangles as elementary elements of the image. The edges of triangle outline area of a certain texture which can be rather complex. For a simple case we can consider that the area inside triangle has a certain color attribute. An example of a rasterized triangle with a color attribute of the outlined area is presented in Figure 3-16. The pixels are numbered in both horizontal and vertical directions. The direction of the *Y* axis is from top to bottom since for the computer screen the upper left point is usually considered as a reference point with coordinates (0,0). For each scanning line which touches or intersects the triangle we can define two points characterizing the beginning of the rasterized image and its end, or in other words, ***start*** and ***end***.

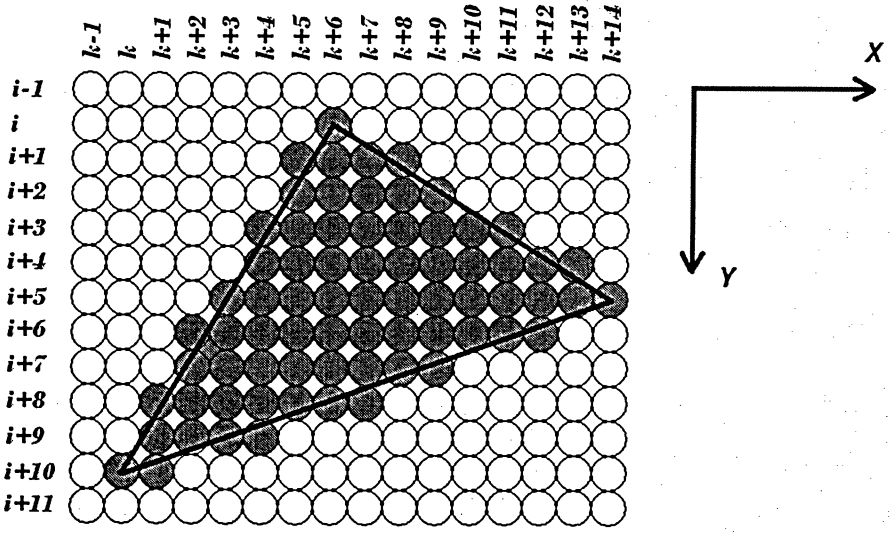


Figure 3-16. Rasterized triangle representation.

This triangle is considered as an object which can be defined by an array of the structure presented in Table 3-5.

Table 3-5. Data structure (array) for storing triangle's pixel lines.

Row number	*Start*	*End*
i	*k+6*	*k+6*
i+1	*k+5*	*k+8*
i+2	*k+5*	*k+9*
i+3	*k+4*	*k+11*
i+4	*k+4*	*k+13*
i+5	*k+3*	*k+14*
i+6	*k+2*	*k+12*
i+7	*k+2*	*k+9*
i+8	*k+1*	*k+7*
i+9	*k+1*	*k+4*
i+10	*k*	*k+1*

Start	*End*
1	*i+10*
K+6	*k+6*
K+5	*k+8*
K+5	*k+9*
K+4	*k+11*
K+4	*k+13*
K+3	*k+14*
K+2	*k+12*
K+2	*k+9*
K+1	*k+7*
K+1	*k+4*
K	*k+1*

The array contains pairs of numbers, ***Start*** and ***End*** , for each scanning line from ***i*** to ***i+11***, as shown on the left side of the table. The data structure can be significantly simplified using the fact that the information about the line number is redundant. We just need to know the starting and ending line numbers or the number of lines the triangle occupies. This structure is depicted on the right side of the table.

Implementation of rasterizing method doesn't require arithmetic operations. It's performed using just comparison operations.

3.2.6. Surface visibility

To present a 3-D object on the projection plane the surface visibility aspect should be taken into account. For example, for the tetrahedron presented in Figure 3-17 the surfaces denoted by triangles ABD and ADC are invisible on the projection plane, or hidden.

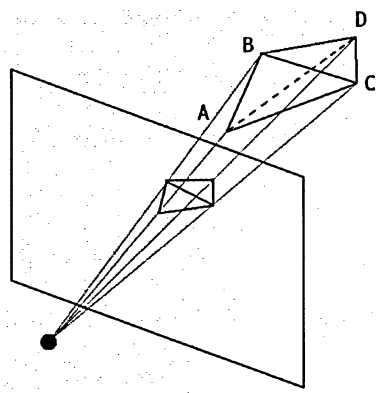


Figure 3-17. Hidden surfaces illustration.

There are a number of algorithms for hidden surface determination: Floating Horizon Algorithm, Roberts Algorithm, Warnock Algorithm, Appel's Algorithm, The Haloed Line Algorithm, Weiler-Atherton Algorithm, z-Buffer Algorithm, List Priority Algorithms, Binary Space Partitioning Algorithms, Scan Line Algorithms [55], Back-Face culling algorithm, Back-to-Front Sorting [54], and others. This aspect has two issues: hidden surfaces of an object itself, and surfaces of an object a view of which is blocked by another object(s). The scope of this thesis is a simpler case when the system synthesizes only one object or several objects which are not blocked by other objects. Thus, the hidden surfaces as a result of a view of an object blocked by other objects are not considered in this thesis.

The task of the project is to implement 3-D objects synthesis for a system-on-chip, thus a straightforward technique called back-face culling is considered since it looks simpler than other approaches but works very good for hidden surfaces determination for a single 3-D object or for non-overlapping images. The idea of the method is to evaluate an angle between the normal to the considered surface and the vector characterizing the viewing direction. The direction of normal to the surface (triangle) is assumed to go outward of the body of the object. Thus, it is obvious that when the value of the angle between the viewing vector and the normal to the surface is within the range $[-\frac{\pi}{2},\frac{\pi}{2}]$ then the surface is face turned to a viewer and thus visible. Otherwise, the surface is turned outward and is invisible, as depicted on the left side of Figure 3-18. Generally, this dependence relates to an angle between the normal vector and a viewing vector directed to any point of the considered triangle.

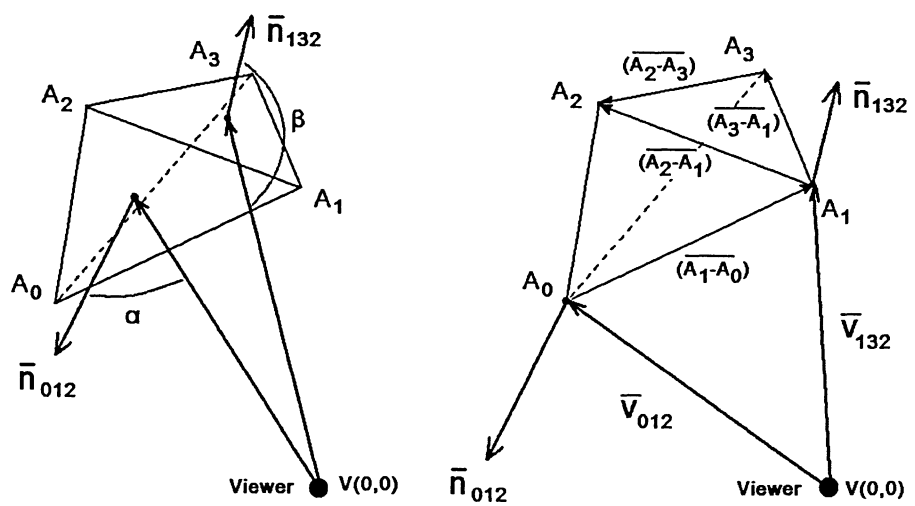


Figure 3-18. Visible and non-visible surfaces: back-face culling determination technique.

To formalize this approach we must find out the coordinates of the normal vector and the viewing vector.

The normal vector can be obtained by the cross product of two vectors connecting the vertices of the triangle. The order by which the vertices are considered must be unified to provide that the normal vector always goes outside of the object's body. Usually an order in which a look from the outside of the body on a side surface (triangle) is assumed and the vertices are taken counterclockwise. For the tetrahedron depicted on the figure above the order is, for example, A_0-A_1-A_2, or A_1-A_3-A_2 for considered triangles.

The normal vectors to both triangles can be expressed as:

$$\overline{n_{012}} = (\overline{A_1 - A_0}) \times (\overline{A_2 - A_1}) \quad \textbf{and} \quad \overline{n_{132}} = (\overline{A_3 - A_1}) \times (\overline{A_2 - A_3}) \qquad \textbf{[3-33]}$$

By evaluation the signs of the dot product of the vectors $\overline{n_{012}} \cdot \overline{v_{012}}$ and $\overline{n_{132}} \cdot \overline{v_{132}}$ the conclusion about visibility of the given surface can be done.

The formal expression for the cross product is following:

$$\overline{n_{012}} = (\overline{A_1 - A_0}) \times (\overline{A_2 - A_1}) = \begin{vmatrix} \bar{i} & \bar{j} & \bar{k} \\ x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \end{vmatrix} \qquad \textbf{[3-34]}$$

Or the coordinates of the vector $\overline{n_{012}}$ can be presented as

$$\left(\begin{vmatrix} y_1 - y_0 & z_1 - z_0 \\ y_2 - y_1 & z_2 - z_1 \end{vmatrix} \; - \begin{vmatrix} x_1 - x_0 & z_1 - z_0 \\ x_2 - x_1 & z_2 - z_1 \end{vmatrix} \; \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_1 & y_2 - y_1 \end{vmatrix} \right) \qquad \textbf{[3-35]}$$

The same can be written for the second normal vector.

The coordinates of the viewing vector $\overline{v_{012}}$ are $(x_0 - 0, y_0 - 0, z_0 - 0)$, i.e. (x_0, y_0, z_0) which are the coordinates of the point A for the viewer located in the reference point ***V(0,0)***.

The dot product of the vectors $\overline{n_{012}}$ and $\overline{v_{012}}$ can be computed as

$$\overline{n_{012}} \cdot \overline{v_{012}} = n_x \cdot v_x + n_y \cdot v_y + n_z \cdot v_z \qquad \textbf{[3-36]}$$

So, to define whether the considered surface is visible or non-visible we must perform 12 subtractions, 9 multiplications, 5 additions and check the sign of the resulting value.

The number of arithmetic operations per each object's surface is summarized in Table 3-6. These operations are performed only once for a given location of a body in space.

Table 3-6. Number of arithmetic operations required for the back-face culling algorithm implementation, per each surface of an object.

Back-face culling algorithm	*Number of operations*			
Operation	**ADD**	**SUB**	**MUL**	**DIV**
normal X coordinate computing	0	5	2	0
normal Y coordinate computing	0	5	2	0
normal Z coordinate computing	0	5	2	0
dot product computing	2	0	3	0
TOTAL	2	15	9	0

The summary of the presented methodology for virtual 3-D objects synthesis is the following:

1) Denote the vertices and the sides of the 3-D objects, define the sides as objects with a certain order of vertices (define triangles);
2) Perform necessary transformation of a virtual object, compute the coordinates of all vertices in a chosen space;
3) Define visible and hidden sides of the object for a given view on the object (for each camera view), define color for visible surfaces;
4) Compute the coordinates of the projected vertices of the 3-D object for both views of two (stereo) cameras;
5) Generate the arrays defining the projected line segments;
6) Rasterize the line segments connecting the projected vertices, i.e. draw the triangles with a certain color attribute;
7) Repeat steps 4-5 for each of visible surfaces of the object for each camera view.

The presented method of the virtual 3-D objects synthesis doesn't consider all aspects of a solid body image presentation. There are other techniques which help to represent objects as realistic as possible. Among these techniques are the following: structure deforming transformations (scaling or shearing), rendering textured polygons and interpolatively shaded polygons, anti-aliasing, clipping, modeling.

We limited the approach because the goal of the thesis is to illustrate that a real-time stereo vision system with synthesis of 3-D virtual elements can be implemented in a system-on-chip. It means that a system which takes into account other aspects of 3-D virtual graphical objects synthesis can also be implemented. Though, the task seems rather complex and requires a thorough development of the methodology. The thesis presents an element of that common

methodology which might be developed afterwards into a common approach for synthesis of virtual 3-D objects of practically any complexity.

3.3. System architecture synthesis and analysis

Any computational system has a certain architecture (A) which can be presented as combination of three components: Components (C), Links (L) and Procedures (P), i.e. A={C, L, P}. In its turn each component can be implemented either in hardware, or in software. That is, $C=\{C_H, C_S\}$, $L=\{L_H, L_S\}$, $P=\{P_H, P_S\}$. So, in general case, a digital system consists of a set of general purpose processors, memory and application specific hardware circuits, hardwired and virtual links, and hardwired and programmed procedures. The target functionality of a system is defined by the specification of a system which is a set of loosely defined functionalities and certain constraints. The constraints can be any set of the performance constraints (latency, cycle time, data rate, etc.), area constraints (on-chip, on-board, on-system), power consumption constraints, reliability constraints (life time, repairability, testability), and cost (cost of system, of modification, of maintenance, etc.). The methodology for the optimal digital system architecture synthesis is presented in [58].

3.3.1. Design-oriented approach

There are two possible approaches to the design implementation. First is a design-oriented approach. The decision of mapping functionalities into dedicated hardware or implementing them as programs on a General Purpose Processor (GPP) usually depends on estimates of achievable performance and the implementation cost, see Figure 3-19.

This division impacts every stage of the design and is based on the designer's experience and performed on the initial stages of the design process [56, 57]. As a result some portions of the design are often not optimized, i.e. either over-designed or under-designed with respect to the required performance.

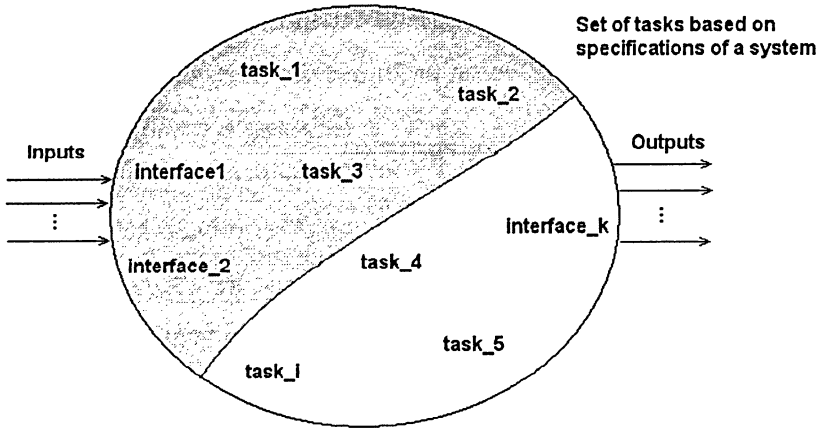


Figure 3-19. A design-oriented approach to system implementation.[23]

3.3.2. Synthesis-oriented approach and design space

The second approach is a synthesis-oriented solution which starts with behavioral description of circuit functionality. It means, that all tasks and processes of the system are described by an appropriate specification language. In the recent years, hardware description languages (HDLs) are used for this purpose. Figure 3-20 illustrates the stages of a design process for a system [58].

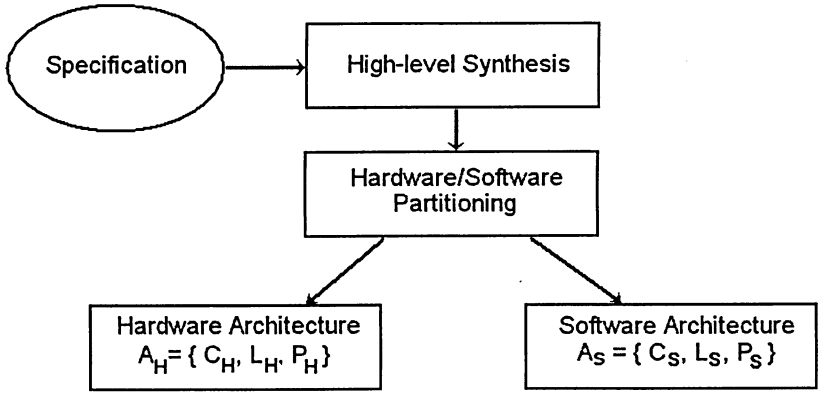


Figure 3-20. System architecture design process stages.

All possible solutions belong to a certain design space. One end of the design space is a purely hardware implementation of the design. A software implementation is on the other end of the space [59]. The left side of Figure 3-21 illustrates the initial layout for the synthesis-oriented approach.

[23] Courtesy to [56]

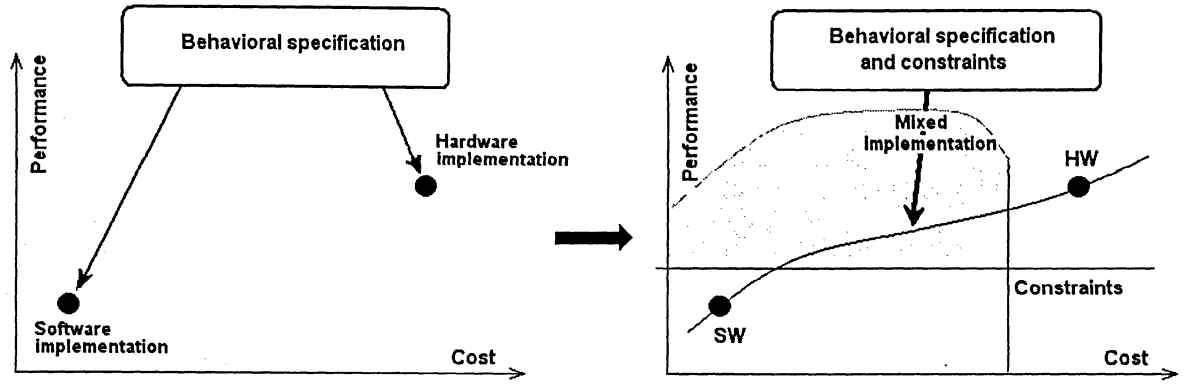


Figure 3-21. A synthesis-oriented approach to system implementation.[24]

A design must satisfy a set of constraints which are performance characteristics like timing, power consumption, or/and others, and cost characteristics like involved parts cost, area on the board, or/and others. Thus, generally speaking, we have a multidimensional design space. The right side of Figure 3-21 illustrates two-dimensional design space since it considers possible solutions based on cost-to-performance trade-offs. Obviously, the architectural solution for the system is limited by the area which is within the shadowed area. The line connecting HW and SW implementations illustrates the set of possible solutions. So, a segment of the line inside the shadowed portion of the design space depicts the sub-space of possible solutions. The next step is to find the optimized variant of all possible architectural solutions for a given behavioral system description which satisfy the set of constraints.

3.3.3. Sequencing graph (or Data Flow Graph)

For synthesis purposes a digital system must be defined in a formal way. It can be specified [58] by

a) a sequencing graph (or Data Flow Graph),

b) a set of functional resources, and

c) a set of constraints.

DFG is a polar and acyclic graph $Gs(V,E)$ where $V=\{V_i: i=1,2, \ldots n\}$ is a vertex set, and $E=\{E(v_i,v_j): i,j=0,1, \ldots n\}$ is a set of dependencies, which for a particular case can be presented as illustrated in Figure 3-22.

[24] Courtesy to [56]

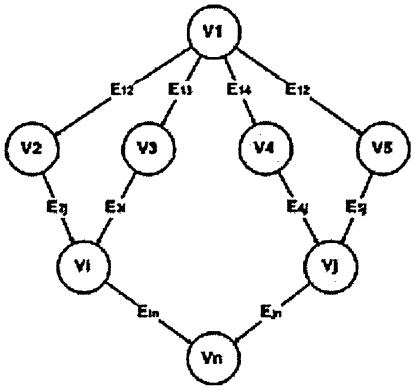


Figure 3-22. Sequencing Graph (or Data Flow Graph) presentation.

In relation to a digital system design, for example, for a system the functionality of which is described by the expression $Y = (x_1 + x_2)/(px_3 + qx_4) + rx_5$, the corresponding DFG looks as presented in Figure 3-23.

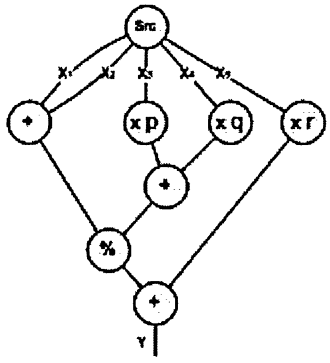


Figure 3-23. DFG for the example task.

Vertices represent functional resources which can be primitive resources (e.g. adder, multiplier, comparator, etc.), application specific resources (e.g. FIR, FFT. etc.), memory resources, and interface resources.

Architectural synthesis and optimization task includes the following stages:

a) Placing the operations in time and in space, i.e. determining the time interval for execution and binding the resources; these tasks are denoted as scheduling and binding.

b) Determining interconnections of the data-path and the logic-level specification of the control unit.

3.3.4. Scheduling

The vertices of the DFG have the following attributes: execution delays, i.e. $D=\{d_i: i=0,1, \ldots n\}$ and start times, i.e. $T=\{t_i: i = 0,1, \ldots n\}$.

The goal of the scheduling task is to determine start times which depend on the precedence constraints specified by the DFG. For example, for the DFG presented in Figure 3-23 the scheduled DFG is depicted in Figure 3-24.

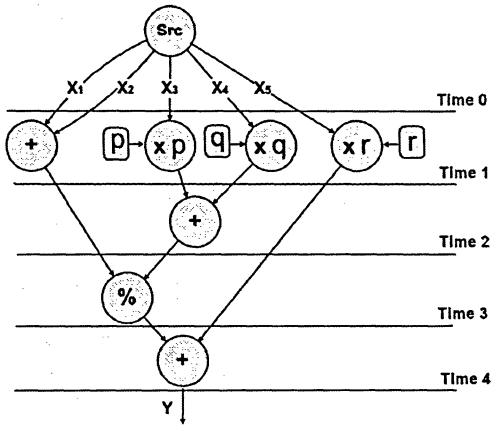


Figure 3-24. Scheduled DFG.

At the moment "T1" three multiplication operations are performed. It's possible to provide the same functionality with other variants of scheduling, as depicted in Figure 3-25.

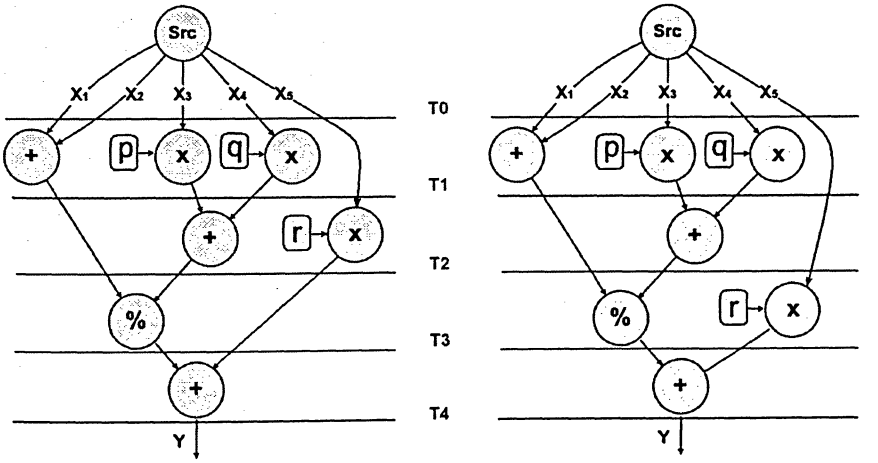


Figure 3-25. Possible variants of scheduling.

3.3.5. Binding

The variants presented in Figure 3-25 produce the same result as the initial one but might differ by other features (e.g. latency), or by the set of resources used for the implementation. It means that the same resource can be used in different time slots. Resource selection is denoted as binding. It assumes that there may be more than one resource applicable to an operation.

Binding is selection of relations between operations on a DFG (V_i) and resources (R_i) [58]. If there is a limitation as to the number of resources used for the implementation then the situation is referred to as constrained scheduling. For example, the task is to find an optimal architectural solution for the considered system with certain constraints, namely latency and area. Figure 3-26 presents possible variants of constrained scheduling and binding for the example.

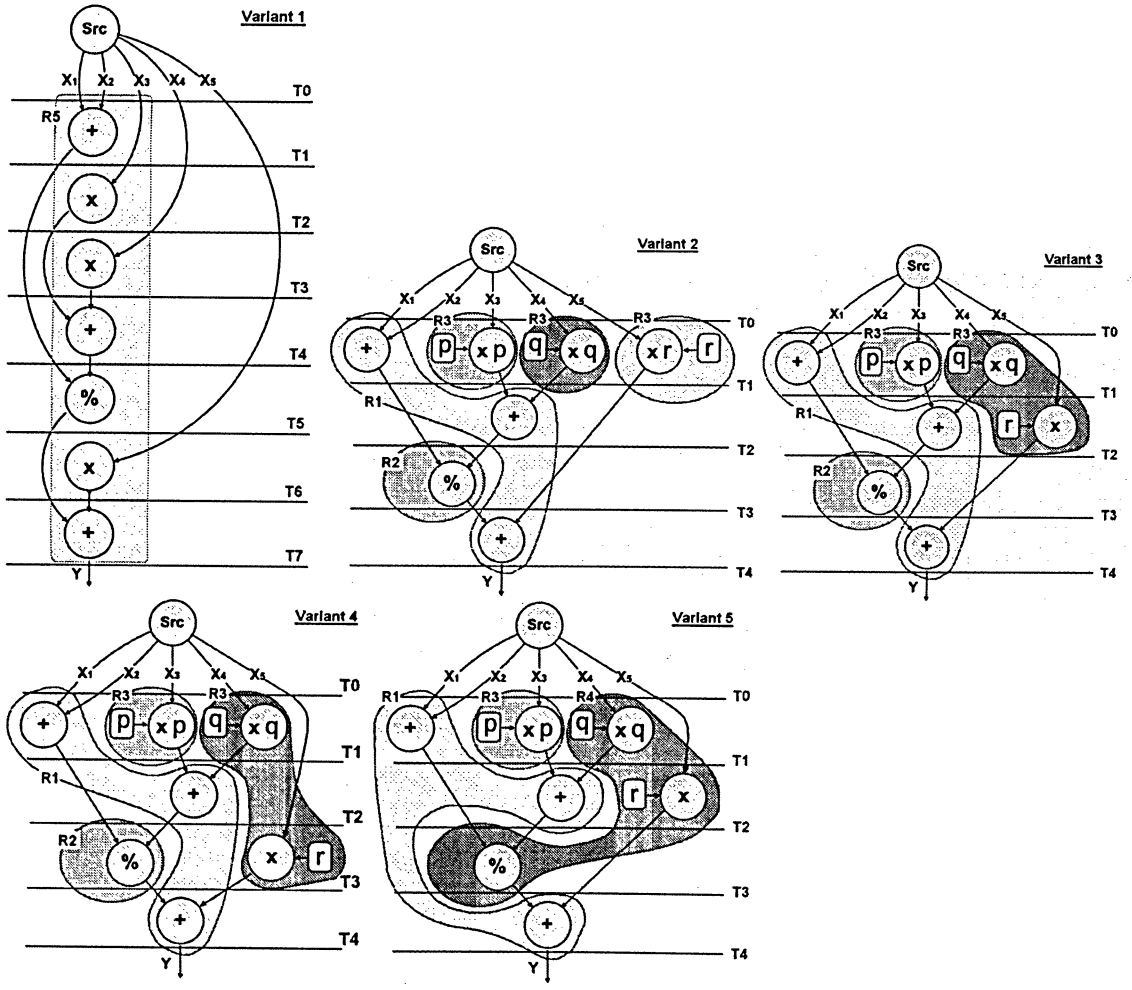


Figure 3-26. Variants of constrained scheduling and binding.

The first variant corresponds to a straightforward solution, i.e. using just one ALU (Arithmetic Logic Unit) or microprocessor which executes one operation (command) at a time. Variants 2 through 5 use different schemes of resources binding.

3.3.6. Architecture optimization

Assume that the resources which can be used to provide the functionality of the design and their area and latency specifications are the following:

R1 – Adder, execution delay, d_1 = 10 t.u. (time units), area – a_1=20 sq.u. (square units);

R2 – Divider, d_2 = 20 t.u., area – a_2=80 sq.u.;

R3 - Multiplier, d_3 = 20 t.u., area – a_3=60 sq.u.;

R4 – Multiplier/Divider, d_2 = 20 t.u., area – a_2=100 sq.u.;

R5 – ALU/μP, d_2 = 20 t.u., area – a_2=200 sq.u.

For the first variant we have latency T_{1L} = 140 t.u., A_1 = 160 sq.u., and for the rest of variants:

T_{2L} = 60 t.u., A_2 = 280 sq.u.;

T_{3L} = 70 t.u., A_3 = 220 sq.u.;

T_{4L} = 60 t.u., A_4 = 220 sq.u.;

T_{5L} = 70 t.u., A_5 = 180 sq.u.

The considered design space and the layout of the variants in the area-latency factors space is presented in Figure 3-27.

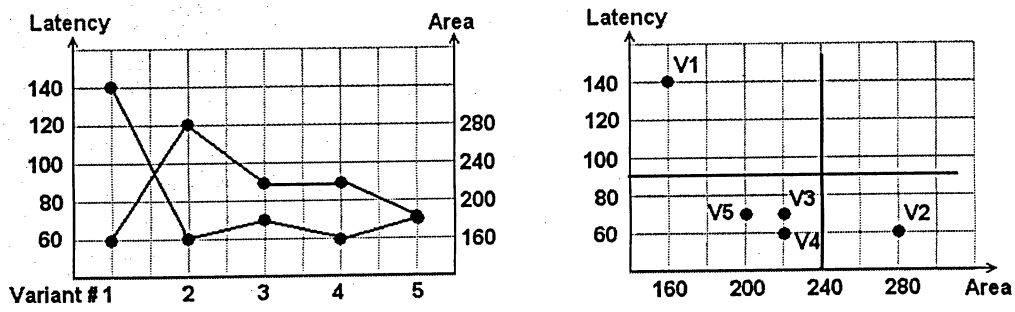


Figure 3-27. Design space for the system.

If, for example, the constraints are set to the values T_{Lmax} = *90t.u.* and A_{max}-*240sq.u.*, then only architecture variants V3, V4, and V5 are within the acceptable portion of the design space (shadowed area). The variants V3 or V4 are the closest to the optimal variant in the latency-area factors (constraints) design space. A similar analysis can be done for other pairs of factors (constraints), for example, cycle-time – latency.

3.3.7. Design space and Architecture Configuration Graph (ACG)

In general case, there are a big number of possible architectures for a targeted task since there are multiple variants of each resource. For example, ripple-carry adder or look-ahead adder, 8-bit or 16-bit adder or multiplier, multiplier with Booth's algorithm or with shifted product, etc. The variants of each resource can be represented graphically as illustrated in Figure 3-28.

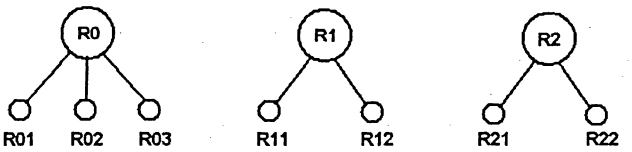


Figure 3-28. Variants of the resources.

The design space then can be represented as an Architecture Configuration Graph (ACG) [43], as depicted in Figure 3-29.

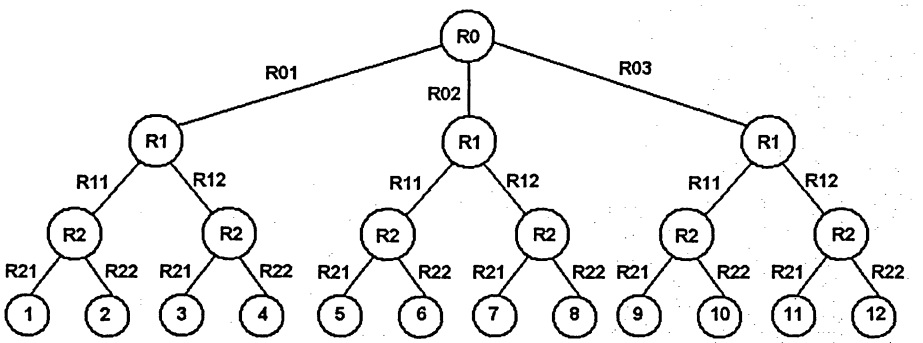


Figure 3-29. Design space as an Architecture Configuration Graph (ACG).

The terminals 1 through 12 in the figure correspond to possible variants of processing architectures.

Depending on the particular task, the performance model should be created. It can represent, for example, the processing time for a block of data, power consumption model, or others.

The design space, or ACG, should be hierarchically arranged [44]. It means that all types of resources represented by ACG must be allocated in optimal order. The optimal order is the one which provides the most monotonic increase or decrease of the performance parameter from left to right terminals. It can be done using "Minimax" method by which the performance value (P) is calculated for both minimal and maximal values for all resources of R0, R1 and R2 (R01

and R03) for the same (fixed) other resources. It is illustrated in Figure 3-30 for two resources, R0 and R1.

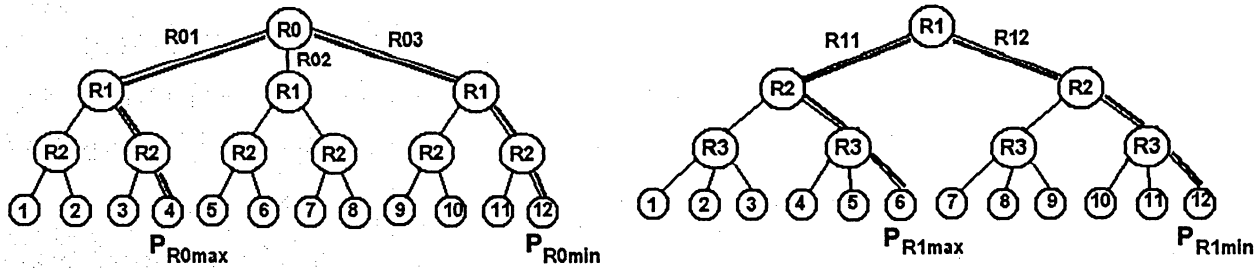


Figure 3-30. "Minimax" method for hierarchical arrangement of ACG.

Then, a coefficient $K_{Ri} = (P_{Ri\max} - P_{Ri\min})/(n-1)$ is calculated for all resources of R_i. The resources are then arranged in an ACG so that of the top of the ACG is the resource with the maximum value of the coefficient K_{Ri}. The resource with the next lower value of the coefficient is to be placed on the second row and so forth.

If n is the number of types of resources and m is the number of possible variants of i resource, then the total number of variants is $N = \prod_{i=1}^{n} m_i$. To arrange the ACG, $(n+1)$ variants must be evaluated.

The total number of variants to be evaluated can be expressed as

$$M = (n+1) + \log_2(\prod_{i=1}^{n} m_i). \qquad [3\text{-}37]$$

3.4. Conclusion

The analysis of formal representation of 3-D bodies in three-dimensional space and two-dimensional projection plane on which translation and rotation transformations are applied is presented in this chapter. The methodology for computing object vertices coordinates as well as rasterizing of line segments and visibility of an object surfaces is also presented. These approaches can be used both for hardware and sequential microprocessor based system architectures.

The methodology for synthesis of system architecture is analyzed. Task scheduling, resources binding and optimal architecture selection are considered and a practical example is presented.

The considered approaches both for formal representation of 3-D objects and synthesis of system architecture will be used for the stereo vision system architecture development as it is presented in the next chapter.

4. DEVELOPMENT OF ARCHITECTURE AND IMPLEMENTATION OF THE SYSTEM

4.1. Introduction

This chapter presents the architectural synthesis of the real-time stereo vision system. The system incorporates a number of tasks each of which requires different level of performance. There are computation intensive time tasks which have very strict timing constraints (e.g. stream data capture, output to VGA), and, from another hand, there are algorithm intensive tasks which require complex computational resources but doesn't have very strict timing requirements.

The architectural solution for the system can be built by design-oriented approach, as it was shown in the chapter 3 when hardware-software (HW-SW) partitioning is done based on the experience of the designer. But for the complex systems the selection of the optimal architecture is not a trivial task and a formal approach, or synthesis-oriented approach, should be employed.

The main tasks of the system are: image capture, output to VGA, color decoding, 3-D synthesis, edge detection, and data read/write to and from memory. Synthesis-oriented approach will be applied to obtain the optimal architecture for the system.

4.2. Architecture synthesis for the real-time stereo vision system

Real-time stereo vision system with elements of 3-D interactive objects synthesis must include three major parts: image capture subsystem, video processing subsystem, and visualization output subsystem, as presented in Figure 4-1.

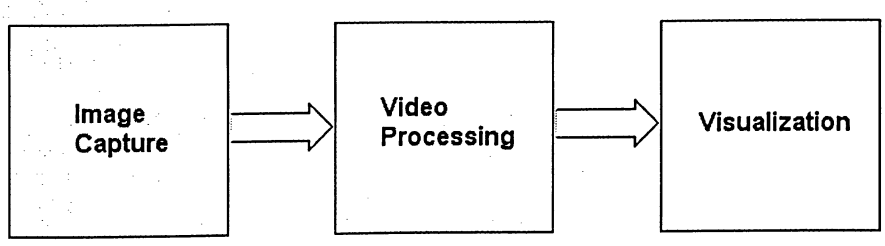


Figure 4-1. Real-time stereo vision system block diagram.

The basic specifications of the system are as follows:

Image resolution: 640 (horizontally) x 480 (vertically) pixels (standard VGA resolution);

2 input channels;

2 output channels;

Frame rate for capture: 30 fps per each channel;

Data resolution: 8 bit, color;

Frame rate for visualization: 60 fps per each channel.

The fastest processes which must be provided by the system:

1. Cycle-time for the Image Capture Subsystem is:

1 / 1.22 x (640 x 480 x 30(fps) x 2(channels)) = **44.5nS per pixel.**

Here 1.22 is the active line coefficient for the image sensor used in the system (780 clock cycles per line / 640 active pixels per line; the numbers are from the datasheet). The corresponding data rate is **22.5 Mbytes/sec.**

2. Cycle-time for the Visualization Subsystem is:

1 / 1.26 x (640 x 480 x 60(fps) x 2(channels))= **21.4nS per pixel.**

Again, 1.26 is the active line coefficient according to the standard VGA specification (31.77μS / 25.17μS). The corresponding data rate is **46.7 x 3(RGB) Mbytes/sec.**

4.2.1. Architecture selection for the Image Capture Subsystem (ICS)

The cycle time the system must provide for capturing data from each of the CMOS Image Sensors is 44.5nS. It means that the system must process each unit data within the specified time. The processing for the ICS includes data capture from the video camera and at the same time passing data in a pre-defined format to the VPS for further processing. From another hand, image sensor chips provide valid pixel data after a certain set-up time, thus making the timing requirements for the capturing device stricter (at least by one order). A RISC microprocessor operating at the clock frequency of 50MHz and performing one command per 2c.c. will require 40nS per command execution. So, it needs to perform two different operations during its one command execution time. Another point is that the microprocessor can't perform only data read/write commands. There is always a set of other commands which must be executed to provide the correct functionality, e.g. loop commands, or strobe detection.

It means that microprocessor can't be employed for the capture subsystem, and thus it should be implemented in hardware (logic).

4.2.2. Architecture selection for the Visualization Output Subsystem (VOS)

The similar explanation relates to the visualization subsystem. The cycle-time the system must provide for this task is 21.4nS. It can't be done with the RISC microprocessors' execution time of 40nS per command. It means that the task of data output to VGA devices must be also implemented in hardware (logic).

4.2.3. Architecture selection for the Video Processing Subsystem (VPS)

VPS performs all data processing: writing and reading data to and from memory, edge detection, output data generating, color decoding, and virtual 3-D interactive objects (controls) synthesis. Each of these tasks is analyzed regarding the optimal architecture for its implementation.

Architectural synthesis for the Edge Detection task

One of the modes in which the system operates is real-time edge detection visualization using Sobel algorithm [60]. The formal expression of this algorithm is presented in Figure 4-2.

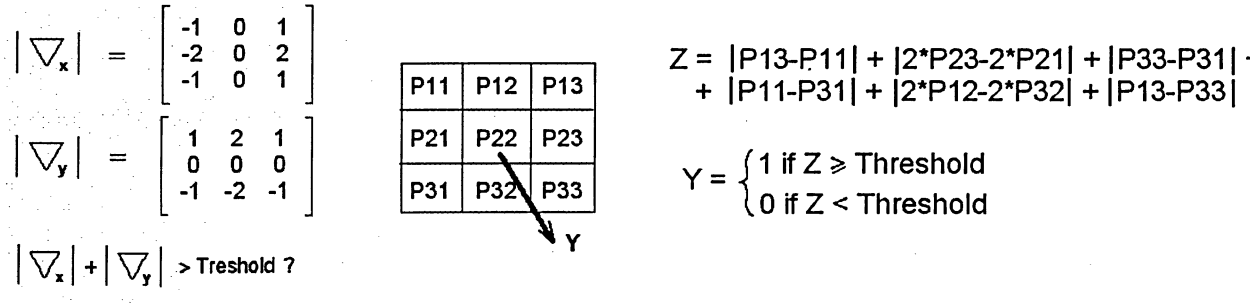


Figure 4-2. Sobel edge detection algorithm specification.

Here P11, P12, ..., P33 are the adjacent pixels on the screen forming 3x3 matrix. For the Sobel algorithm the pixel data from three scanning lines is needed. The output data for the square matrix indicated in Figure 4-2 corresponds to the central pixel P22. To get a result for a given pixel (Y) the calculation is performed using values of 8 adjacent pixels.

The resources needed for the implementation are: modulus subtractor, adder and comparator (these modules can be generated in the Xilinx' Integrated Software Environment, ISE):

- Modulus Subtractor R1 – delay $d_1 = 2$ c.c.
- Adder (16-bit) R2 – $d_2 = 1$ c.c.
- Comparator R3 – $d_3 = 1$ c.c.
- ALU (microprocessor) R4 – $d_{4.1} = 6$c.c. (modulus subtraction), $d_{4.2} = 2$c.c.(addition), $d_{4.3} = 4$c.c. (comparison)

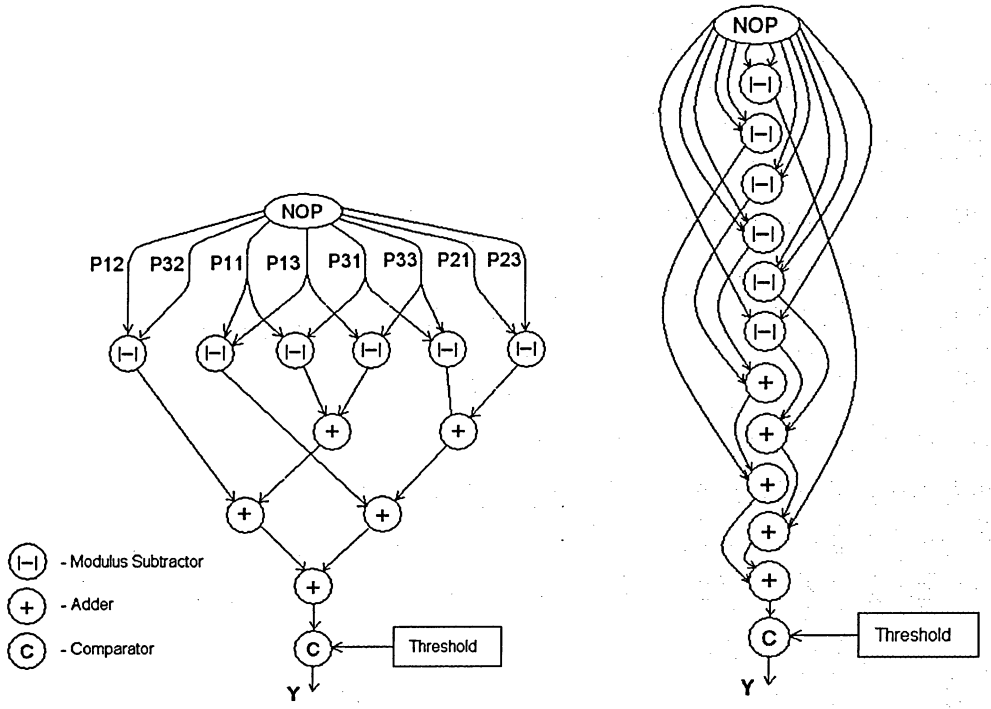


Figure 4-3. Two versions of the DFG for Sobel algorithm implementation (hardware and GPP based).

Two versions of the Sequencing Graph (or DFG) of the process is presented in Figure 4-3. The first DFG relates to logic (hardware) implementation, the second is based on using sequential microprocessor.

Three variants of scheduled Sequencing Graph are considered, see Figure 4-4. For variant 2 two R1 resources are moved to the time slot T2 thus providing sharing of modulus subtractors. Variant 3 corresponds to the microprocessor based design.

Variant 1: Latency $T_L = 6$c.c. (120nS at 50MHz clock);

1 output per 2 cycles (data based on Cycle scheduling performed for this variant).

Variant 2: Latency T_L = 7c.c. (140nS at 50MHz clock)

1 output per 2.5 cycles (data based on Cycle scheduling performed for this variant).

Variant 3: Latency T_L = 50c.c.

1 output per 50c.c. (1mS at 50MHz clock)

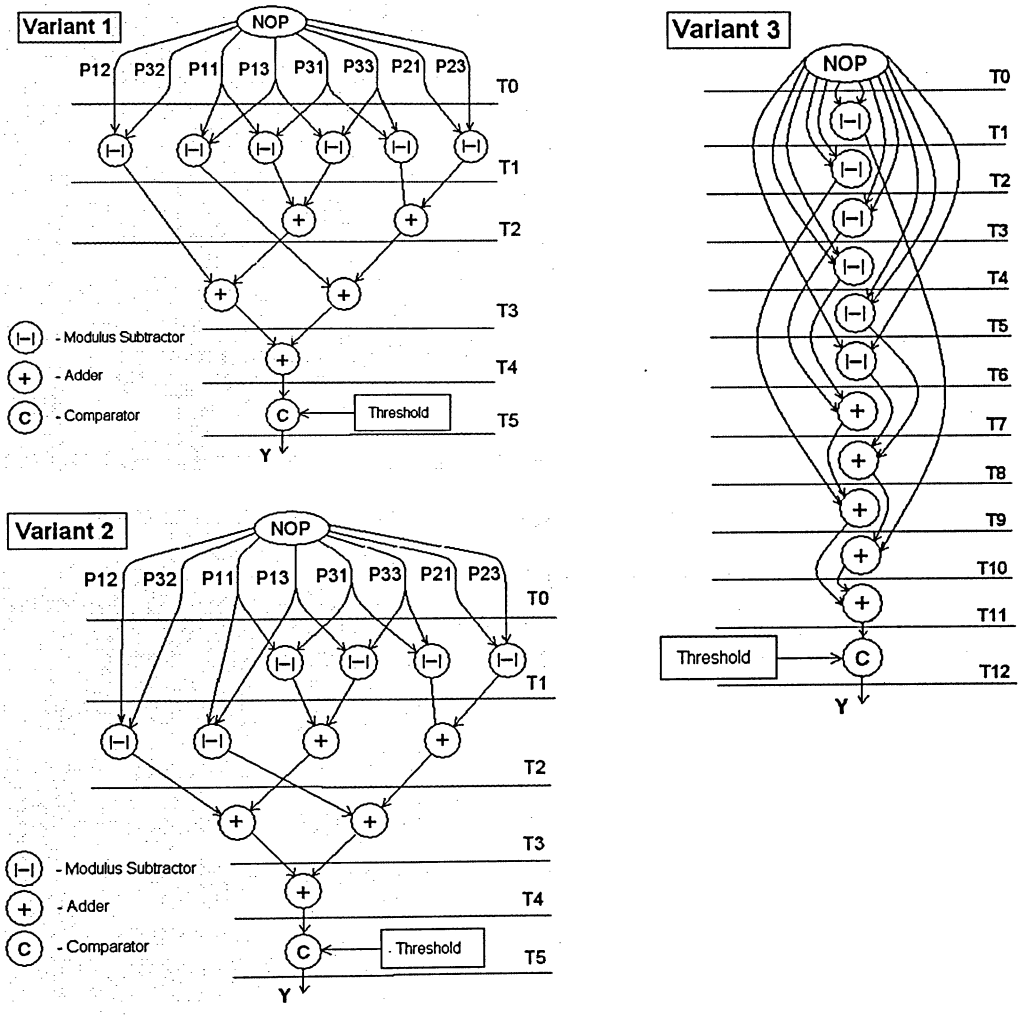


Figure 4-4. Three variants of DFG for Sobel algorithm architecture.

The first conclusion is that the variant #3 is not acceptable since it can't provide necessary timing.

The design will be implemented in FPGA. So, there are no special resource constraints for this design (at least at this design stage). The main concern is the speed of process. Thus, the variant #1 is selected for implementation of the Sobel edge detection algorithm.

For the architectural synthesis the following set of possible resources which can be implemented as modules in the ISE is considered:

- Operational frequency
R0.0: 25 MHz ($\tau_{c.c.}$ = 40nS)
R0.1: 50 MHz ($\tau_{c.c.}$ = 20nS)
R0.2: 100 MHz ($\tau_{c.c.}$ = 10nS)

- Modulus Subtractor
R1.1: τ_{R11} = 1 c.c. (2Mux + Comp.)
R1.2: τ_{R12} = 2 c.c.

- Adder
R2.1: τ_{R21} = 2 c.c. (8-bit)
R2.2: τ_{R22} = 1 c.c. (16-bit)

-Comparator
R3: τ_{R3} = 1 c.c.

The performance model for the best variant selection is Fame Computational Time:

$$T_F = [T_L + (N-1) * T_{cycle} * n] * \tau_{c.c.} \quad \textbf{[4-1]}$$

Where

T_L – latency (in c.c.) for the considered architecture variant,

N = 640 * 480 = 307,200 – number of pixels per frame),

n = 2 – number of clock cycles per one data output for the considered architecture variant,

$\tau_{c.c.}$ – duration of a clock cycle.

So,

$$T_F = [6 + (307{,}200-1) * \max\{ \tau_{R1}, \tau_{R2}, \tau_{R3}\} * 2] * \tau_{c.c.} \quad \textbf{[4-2]}$$

or

$$T_F = 6.14*10^5 * \max\{ \tau_{R1}, \tau_{R2}, \tau_{R3}\} * \tau_{c.c.} \quad \textbf{[4-3]}$$

The target value of T_F is 33.3mS (for 30fpS) since the output for Sobel edge detection technique is performed just for the data flow from one video data source (one camera).

The Design Space is represented by the following Architecture Configuration Graph (hierarchy arrangement is performed but not reflected in this thesis), see Figure 4-5.

Using a technique for the best variant of architecture selection the variant with TF = 30.7 is selected. It corresponds to the resources R0.3, R1.1, and R2.1.

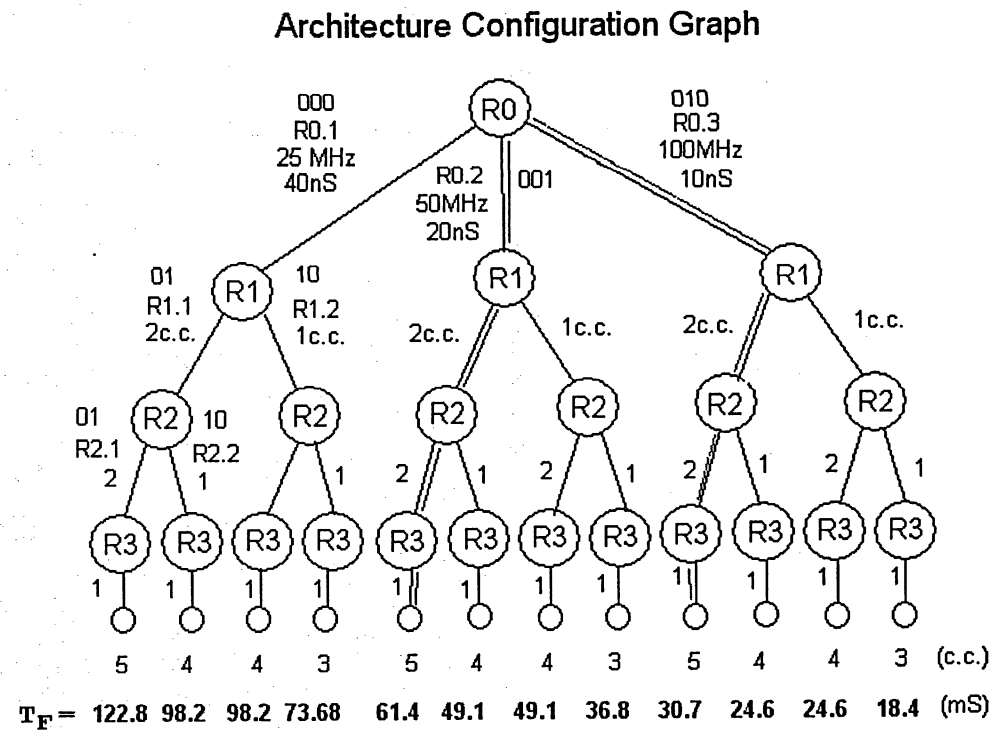


Figure 4-5. Architecture Configuration Graph for Sobel algorithm implementation.

Architecture selection for write to memory and read from memory task

Today's RISC microprocessors can be used in systems with the data rate up to around 20 Mbytes/sec (50nS) but only if a very simple set of operations must be applied on data. For example, the task of capturing and writing pixel data in SRAM chip can be represented by the algorithm presented in Figure 4-6.

If, for example, a command of a RISC microprocessor requires 2c.c. for execution (as it is for the embedded soft-core PicoBlaze microprocessor of Xilinx Virtex FPGAs), then the task needs 24c.c., see Figure 4-6. If the system's clock is 100MHz, the task needs 10nS x 24 = 240nS.

It means that for the VPM the procedure of writing/reading to memory (SRAM) can be only implemented in logic. It can't be implemented with a sequential microprocessor which can't operate as fast as it is needed. Writing/reading to memory procedure is a very straightforward operation which doesn't require any computational resources (with the exception of address calculation which is actually just one increment operation per write or read cycle is employed). This is the task for a state machine (SM).

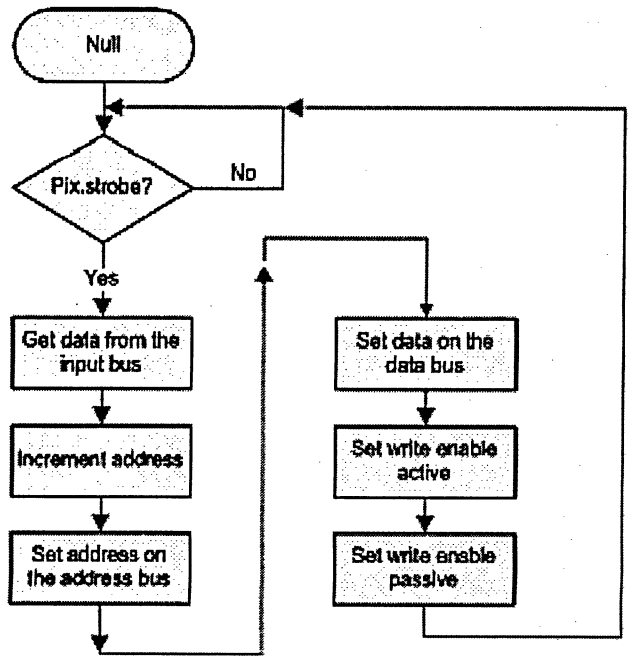


Figure 4-6. Data flow for the capture system.

Colors decoding procedure can also be implemented only in hardware. It is a part of the visualization subsystem task. The system must provide all three color components for each system at the rate of a pixel output, i.e. each 21.4nS.

Architecture selection for 3-D virtual objects synthesis task

As it was illustrated in the chapter 3, the task of 3-D synthesis includes several procedures: computation of the transformed vertices coordinates, computation of projection coordinates, computation of a surface visibility parameter, and computation of rasterizing coordinates.

For computation of the transformed vertices coordinates the DFG can be presented in a number of ways. There are though two main approaches: using hardware and sequential microcontroller. Figure 4-7 illustrates the hardware approach, and Figure 4-8 represents the approach employing sequential GPP. Both illustrated DFGs are scheduled.

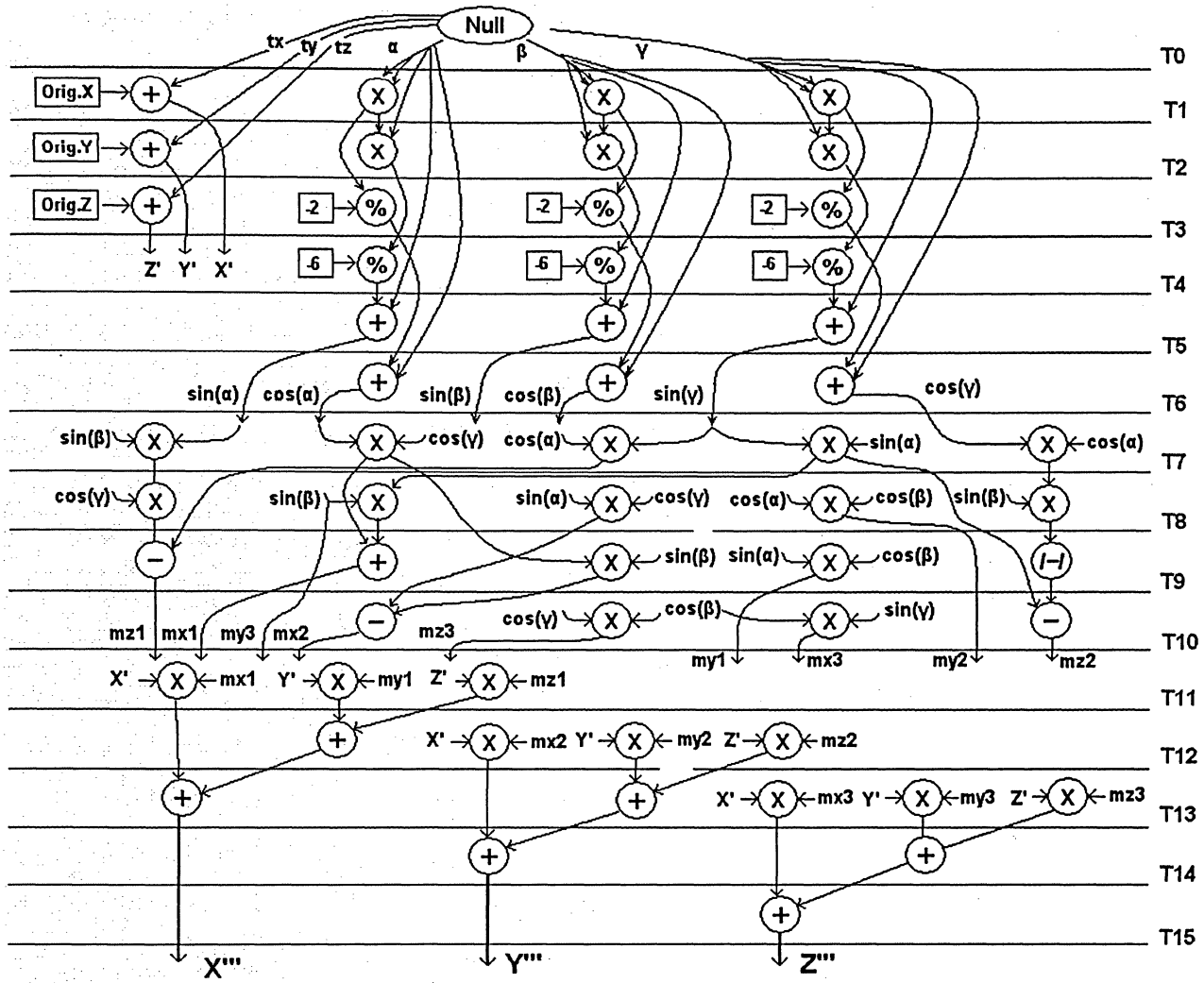


Figure 4-7. DFG for 3-D synthesis task (transformation computation), hardware approach.

Here t_x, t_y, t_z, α, β and γ are the values of translation and rotation along each of the coordinate axis, as it follows form the equation [3-1] and Figure 3-5.

The logic resources of FPGA needed to implement this task are the following:

Adder-Subtractor (16-bit) – delay $d_1 = 1$c.c.

Multiplier (16-bit) - delay $d_2 = 2$c.c.

Divider (16-bit) - delay $d_3 = 16$c.c.

Sign-inverter - delay $d_4 = 2$c.c.

ALU-μP (Adder, subtractor, sign inverter) - delay $d_5 = 8$c.c.

ALU-μP (multiplier) – delay $d_6 = 64$c.c.

ALU-μP (divider) – delay $d_7 = 128$c.c.

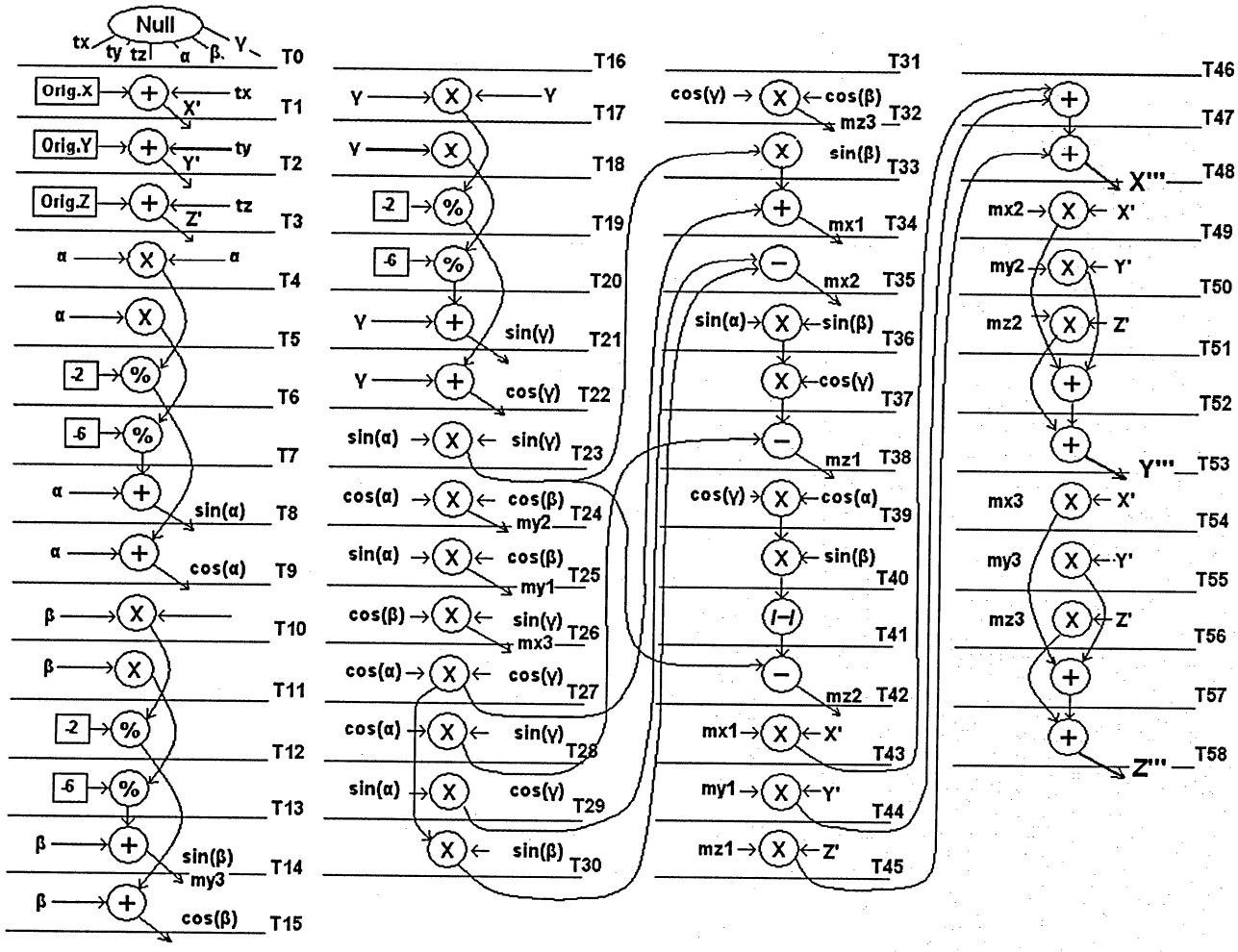


Figure 4-8. DFG for 3-D synthesis task (transformation computation), GPP approach.

For the first variant the latency T_{L1} = 52c.c., and it generates 1 output per 16c.c.

For the second variant the latency is T_{L2} = 2,784c.c., and it generates 1 output per 2,784c.c.

For a single tetrahedron we must process 4 vertices, that is, for the variant #1 the execution time at the clock rate of 50MHz is 2μS (or 1μS at 100MHz); for the variant #2 it is 222μS at 50 MHz clock and 111μS at 100Mhz clock.

The target processing time for this task corresponds to the frame rate. It is 33.33mS for 30fps. The conclusion is that a sequential microcontroller should be used for this task implementation. Even though the corresponding DFG looks more complicated it can be easily implemented by writing a corresponding software code. The computational time for the task is only 0.67% of the target time.

The same approach with the same results was applied to the tasks of computation of projection coordinates, computation of a surface visibility parameter, and computation of rasterizing coordinates. This analysis is not illustrated in this thesis because it is just a repetition of what is shown above.

The only comment is that for the task of computation of rasterizing coordinates the hardware approach can be used if the number of objects or their complexity becomes larger: for tetrahedrons this number is around 30. But for a few objects, as it is realized in the project, using the embedded microcontroller for computations is the best solution.

In general case, the task of 3-D synthesis is not trivial since the volume of computations depends on the object complexity and the number of objects to be synthesized. The target performance time is equal to one frame slot. It is 16.67mS for 60Hz frame rate, or 33.33mS for 30fps. For the proposed technique all the computations for a frame must be done before visualization of the frame starts since the architecture of the system is based on the memory switching scheme. It means that while the whole frame is being captured the incoming data is written to one memory bank and the data for visualization is read from another memory bank. The memory banks are switched when the next frame (form the capture subsystem) starts.

The amount of necessary computations for a 3-D tetrahedron image synthesis and the time which is needed to perform these computations using PicoBlaze embedded microcontroller resources are presented in Table 4-1.

The execution times (in clock cycles) for operations of addition (ADD), subtraction (SUB), multiplication (MUL) and division (DIV) are given for the soft core embedded 8-bit microprocessor PicoBlaze for Xilinx FPGAs. The operations are performed both for integer and fixed point numbers. For each operation the number of PicoBlaze elementary commands is evaluated. The overhead is also taken into account. This evaluation shows that for a tetrahedron synthesis only 0.44mS is needed for all necessary computations. For the output of 60 fps the time slot for a frame is 16.67mS. Thus, it's possible to process up to 37 tetrahedrons just using one embedded PicoBlaze microprocessor, or 18 tetrahedrons for a stereo system. This number increases if to implement a system with multiple PicoBlaze microprocessors. In general, it's possible to implement as many soft core microprocessors as the resources of FPGA allow.

Table 4-1. Estimation of computational time for a tetrahedron image synthesis using PicoBlaze computational resources.

Tetrahedron - computational time estimation using:	**a) PicoBlaze resources**				**b) HW adders and multipliers**			
	Number of operations				*Number of operations*			
Fixed point operations	*ADD*	*SUB*	*MUL*	*DIV*	*ADD*	*SUB*	*MUL*	*DIV*
Transformation (rotation)	28	36	120	24	28	36	120	24
Projection			8	8			8	8
Total	28	36	128	32	28	36	128	32
Plus overhead 20%	34	43	153	38	34	43	154	38
Number of c.c. per operation	8	8	64	128	1	1	16	24
Total c.c. per operation	269	346	9830	4915	34	43	2458	922
Total c.c.				15360				3456
Integer operations								
Transformation (translation)	12	0	0		12	0	0	
Visibility	8	60	36		8	60	36	
Total	20	60	36		20	60	36	
Plus overhead 50%	30	90	54		30	90	54	
Number of c.c. per operation	8	8	64		1	1	16	
Total c.c. per operation	240	720	3456	0	30	90	864	0
Total c.c.				4416				984
Rasterization (Bresenham's Algorithm)								
Initial data computing per line segment		4	4			4	4	
Total for 4 line segments of tetrahedron		16	16			16	16	
Plus overhead 50%		24	24			24	24	
Number of c.c. per operation		8	64			1	16	
Total c.c. per operation		192	1536			24	384	
Total c.c.				1728				408
Average number of points per line segment	200				200			
Number of c.c. per algorithm implementation	28				4			
Total c.c. per segment rasterization				5600				800
Number of edges (segments)	4				4			
Total c.c. per tetrahedrin rasterization				22400				3200
Total c.c. for all computations for tetrahedron:				43904				8048
Execution time at 100MHZ clock, mS				**0.439**				**0.080**

On the other hand, the time frame for 3-D objects synthesis can be expanded to 33.33mS (i.e. twice) since there is no any particular need to re-compute a virtual object at the rate of 60Hz. The rate of 30Hz is acceptable for generating moving objects for visualization purposes. Another approach is to use the embedded microcontroller for the algorithmic portion of the data processing and HW resources (internal multipliers and adders) for speeding up the operations of addition and multiplication. [61, 62] The results of this estimation are presented in the right part of Table 4-1. It gives significantly better results even though the hardware portion of the design becomes more complex in this case.

The embedded microprocessor can be of any type. For example, by using a 16-bit MicroBlaze the computational time decreases due to its more sophisticated arithmetic commands set (including fixed point arithmetic commands) and that it operates on a wider data.

As a result of the architecture analysis and synthesis the real-time stereo video system will employ logic for the tasks of image capture, visualization, color decoding, data storing, edge detection, and an embedded microprocessor for the tasks of necessary computations for 3-D virtual objects synthesis. It confirms that although hardware/software co-design is important for board-level design, it is essential for VLSI implementation [63], as well as for FPGA-based platforms.

4.2.4. Architecture of the system

The proposed system includes 3 subsystems, as illustrated in Figure 4-9 [64]:

- Image Capture Subsystem (ICS),
- Video Processing Subsystem (VPS), and
- Visualization Output Subsystem (VOS).

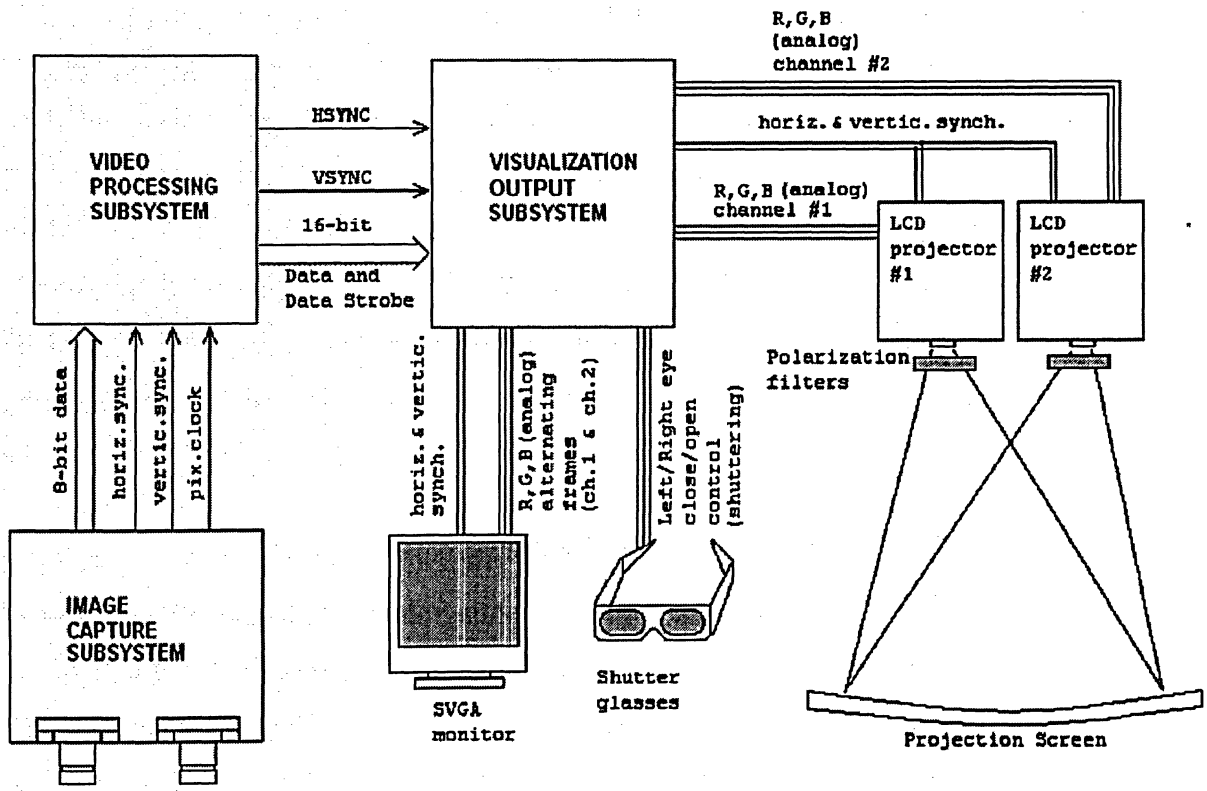


Figure 4-9. Block diagram of the stereo vision system.

The HW-SW partitioning of the tasks the system must perform is presented in Table 4-2:

Table 4-2. HW-SW partitioning of the tasks.

Hardware	Software
Image capture	3-D synthesis
Output to VGA	
Color decoding	
Edge detection	
Data write/read to/from memory	

Video data is captured by the ICS which then passes data to VPS. VPS, first, separates data related to each video channel and stores it into the dedicated memory space. Second, VPS performs all data processing and generates color data and synchronization signals for VOS. VOS, in its turn, generates analog RGB signals in VGA standard and control signals for shutter glasses thus making possible real-time stereo images visualization.

VOS receives RGB data for each output channel from VPS. The system has two output channels for stereo visualization. It drives standard VGA analog inputs. Besides, it generates control signals for shutter glasses. The system incorporates also a circuitry providing analog signals which drive shutter glasses so that a stereo image can be perceived.

The PC is used also for changing the settings of the ICS by means of a GUI module communicating with the ICS via USB port.

The data stored in memory can be used for any kind of image processing tasks. At any instant the data corresponding to two images, one from the video channel one ("left eye") and another from the channel two ("right eye") is accessible for any task that can be integrated into the system.

4.3. Functional description of the system

This section provides the description and the timing diagrams of all implemented hardware modules of the stereo video acquisition and visualization system: Image Capture Module, Video Processing Module, and Visualization Output Module. It also provides functional description of the IP cores of the programmable logic device (FPGA).

4.3.1. Image Capture Module (ICM)

The block diagram of the ICM is presented in Figure 4-10.

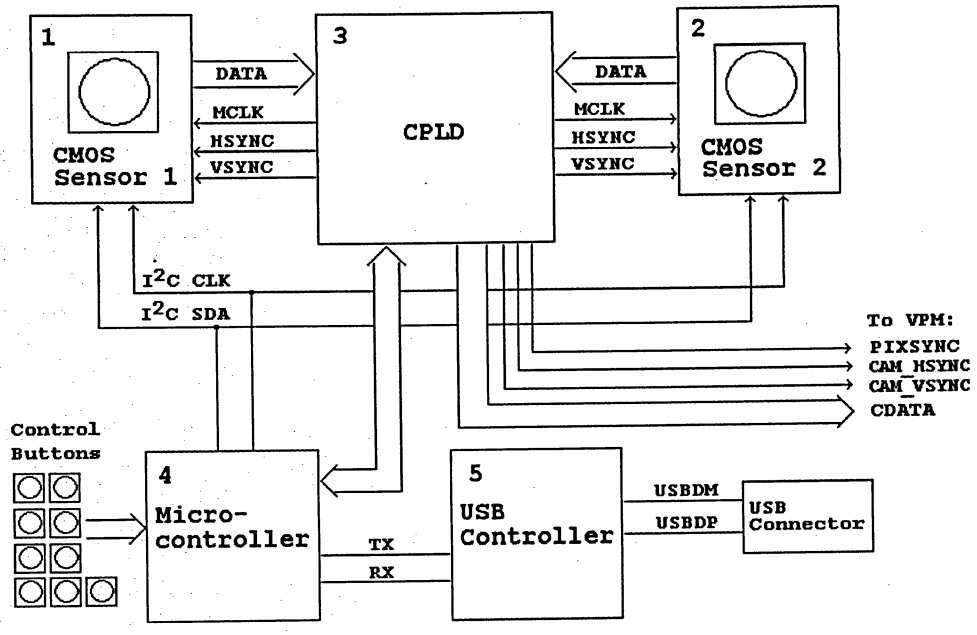


Figure 4-10. Image Capture Module block diagram.

ICM incorporates the following major components: two CMOS sensors (Kodak KAC-9628), CPLD (Xilinx XC95144XL-TQ100), microcontroller (PIC18LF452) and USB controller (FTDI FT232BM).

CPLD provides data capture and packs data in the pre-defined format for sending data to VPM for further processing. It provides CMOS sensors with Master Clock (MCLK) of 12MHz. Thus, a complete scanning line includes 780 full cycles of MCLK. Of these 780 cycles the actual data takes 640 cycles starting from the cycle 136. The starting point for a line or frame is defined by horizontal and vertical synchronization signals (HSYNC and VSYNC).

Upon powering on, the microcontroller initializes both CMOS sensors to operate in SLAVE mode since they need to operate synchronously. In SLAVE mode master clock, horizontal and vertical synchronization signals generated by CPLD feed CMOS sensors. The CPLD operates at 96 MHz clock provided by the on-board oscillator. Master clock frequency for the image sensors is 12 MHz. Each line of data coming from CMOS sensor includes 780 clock pulses (at master clock frequency) 640 of which present actual active pixels values. Lines are separated by horizontal synchro-signals, HSYNC. Each frame includes 504 lines and is

separated one from another by vertical synchro-signals, VSYNC. The number of active lines. i.e. the lines presenting actual image data, is 480. The frame rate is 30 fps.

The format of the data which ICM passes to the VPM is depicted in Figure 4-11.

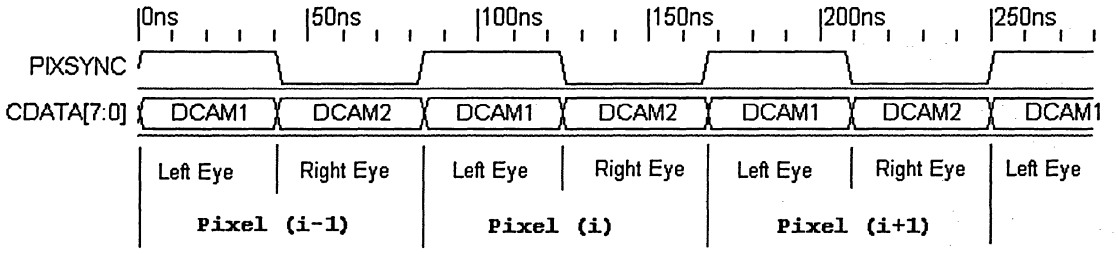


Figure 4-11. ICM output data format.

When PIXSYNC is high the 8-bit data from the CMOS Sensor 1 is on the data lines, and when PIXSYNC is low – the data from the CMOS Sensor 2 is on the data lines.

HSYNC and VSYNC signals are positive pulses of approximately 170nS width. The repetition rate of HSYNC pulses is 15,385HZ (period is 65μS), that of VSYNC pulses is 30Hz (period is 33.33mS).

The resolution of the generated image is 640 x 480 pixels. The color pixel data generated by CMOS sensors is presented in Bayer pattern, i.e. each pixel data keeps only one color component. The layout of the colors is depicted in Figure 4-12.

Since all three color components of the pixel brightness are needed for image processing tasks two missing colors should be taken from the adjacent pixels (from the current and previous line).

There is only one color data value for each pixel. To get all three color data values for each pixel two other colors should be taken from the adjacent pixels.

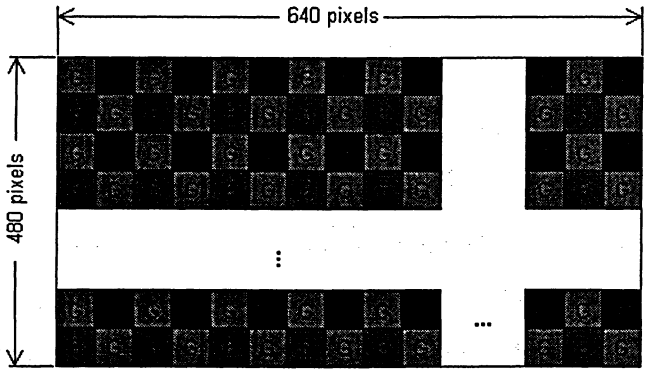


Figure 4-12. Bayer color pattern for the active CMOS sensor area.

Using specially designed software the CMOS sensors can be programmed to operate in different modes. The program provides a convenient GUI for changing the values of CMOS sensors registers. The communication between the ICM board and PC is realized through USB port. The micro-controller sets the CMOS sensors to operate in SLAVE mode at MCLOCK (and thus PIXSYNC) frequency of 12 MHz. There are five groups of buttons on the ICM board: one button is the ICM board reset. Four pairs of buttons perform increase or decrease of Common Video Gain by changing the value of the corresponding register, and separate gain levels for Red, Green and Blue colors in the same way. The description of all control registers of CMOS sensors which can be adjusted can be found in the corresponding datasheet.

4.3.2. Video Processing Module (VPM)

VPM comprises three main components, WRITE CONTROLLER, READ CONTROLLER, and SRAM CONTROLLER, as depicted in Figure 4-13.

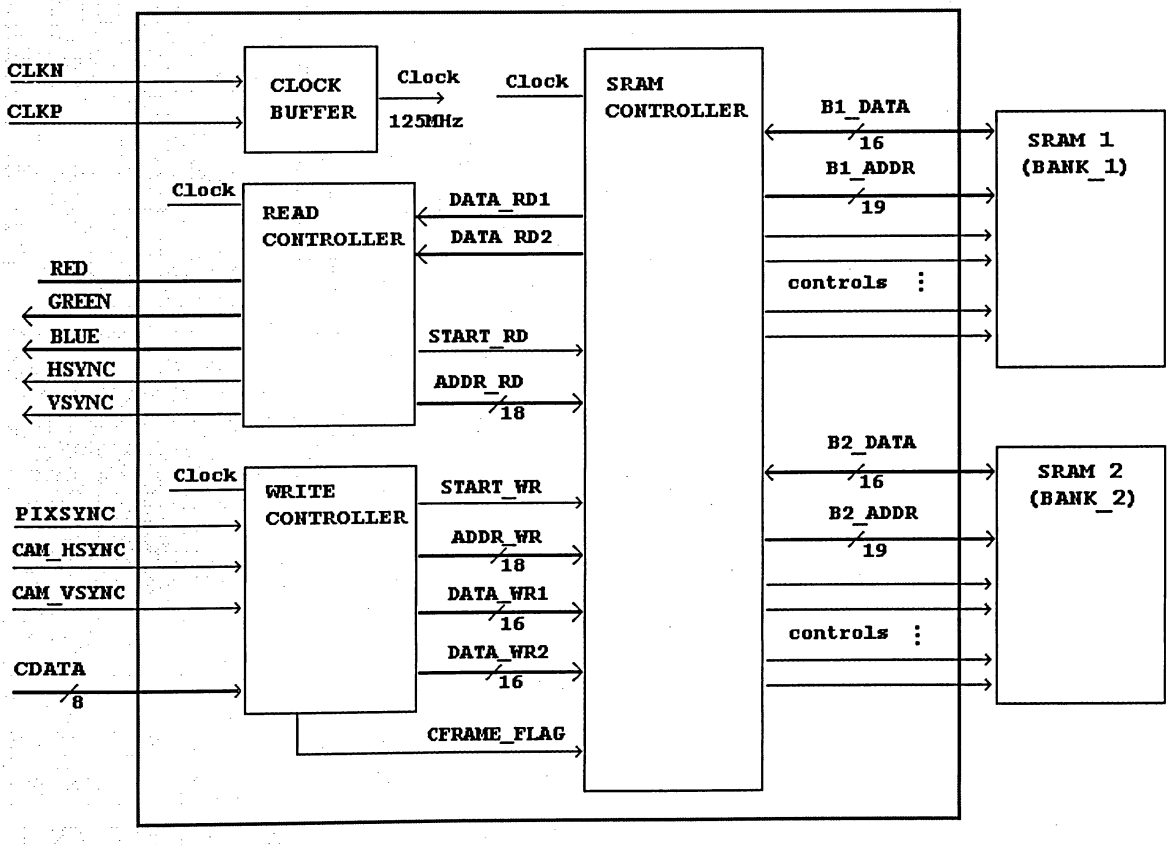


Figure 4-13. Video Processing Subsystem block diagram.

Data capture and writing is performed by the WRITE CONTROLLER. It prepares 16-bit data of each of two incoming data streams to be written to the dedicated space in memory.

READ CONTROLLER fetches data from memory and performs all data processing on the data stored in memory. This is an application specific part of the design in the sense that it can be modified to implement different video systems operating in different modes.

SRAM CONTROLLER provides physical access to the SRAM chips which are located on the board. It writes and reads data to and from two memory banks (two pairs of chips) providing necessary timing for the particular SRAM chips. The external memory architecture is presented in Figure 4-14.

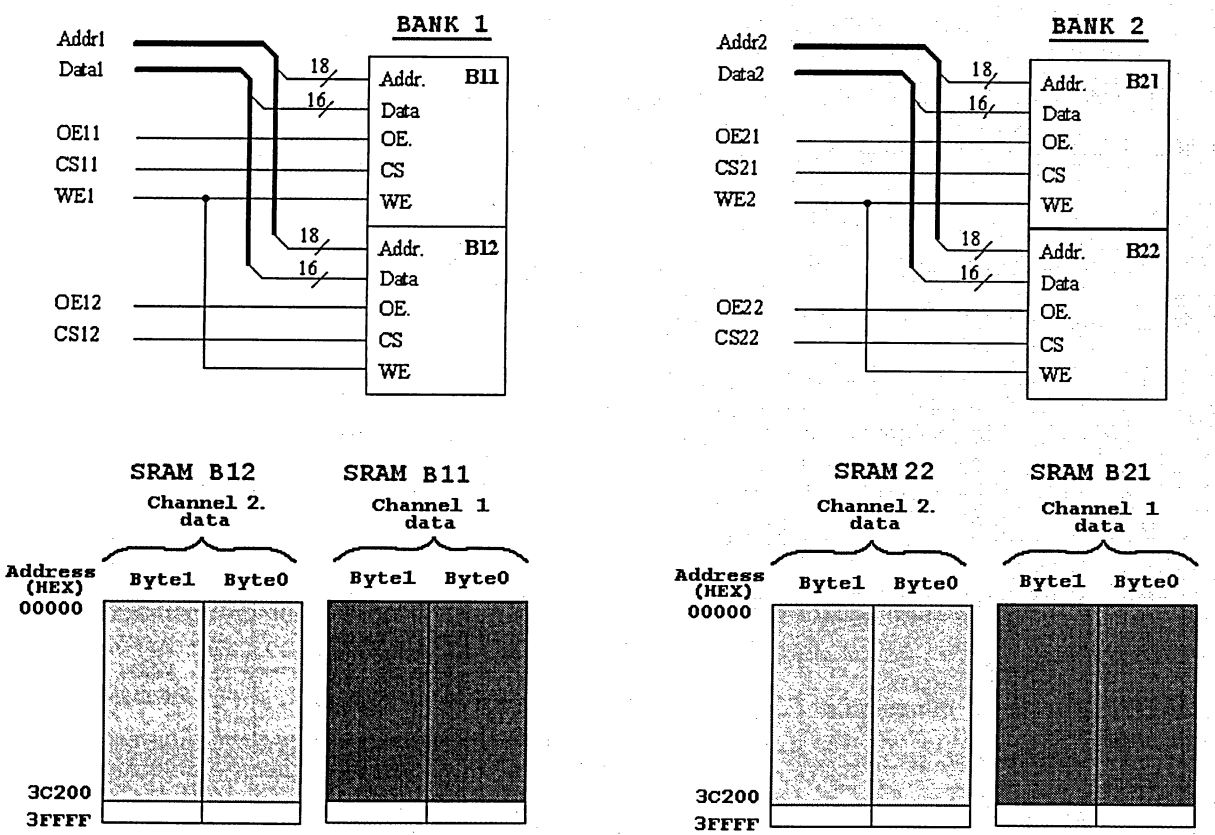


Figure 4-14. External memory architecture and utilization.

The operational phases (reading or writing) for each SRAM chip (or memory bank) are defined by the value of the signal CFRAME_FLAG, generated by WRITE CONTROLLER. It is illustrated in Figure 4-15.

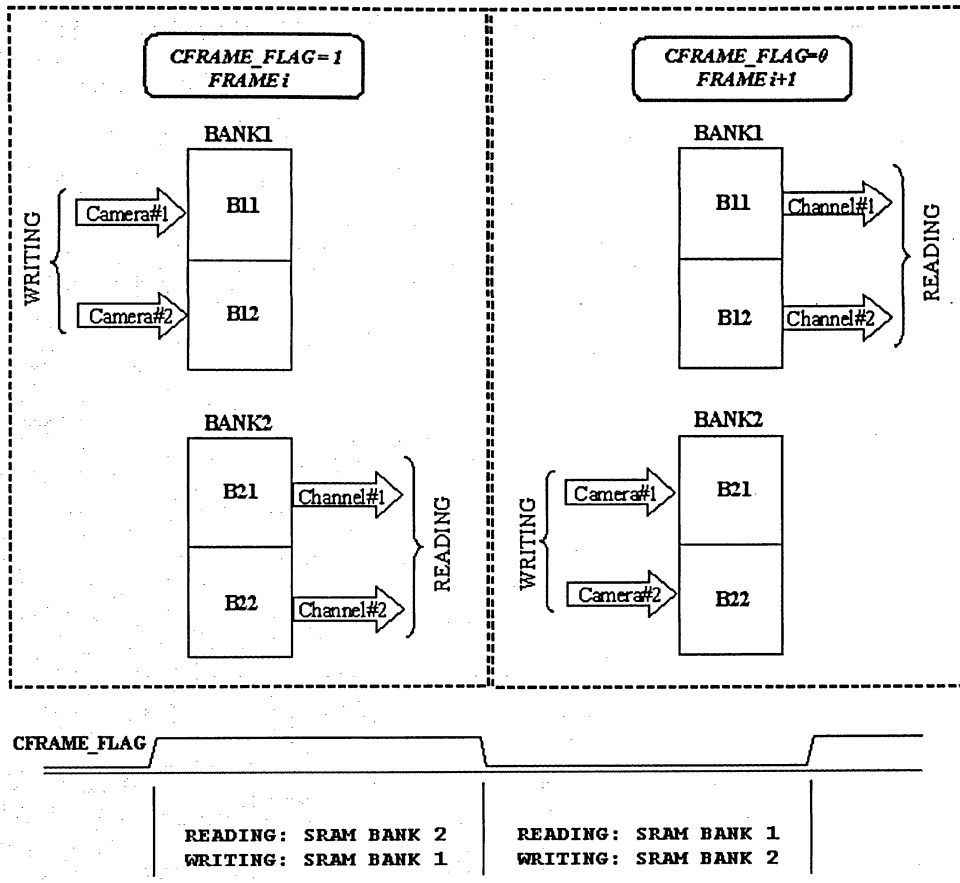


Figure 4-15. Memory banks operational phases definition.

The signal CFRAME_FLAG itself is defined as follows from Figure 4-16. It changes its value with each VSTROBE coming. It means that CFRAME_FLAG signal keeps its value while the data from one frame of the ICM is coming. When the ICM starts to pass the data related to the next frame, the CFRAME_FLAG value inverts. Since VSTROBE rate is 30Hz, CFRAME_FLAG signal changes every 33.33mS.

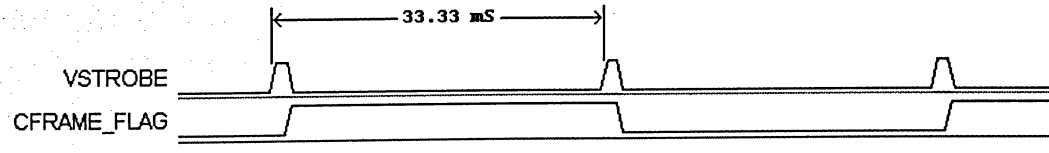


Figure 4-16. CFRAME_FLAG signal definition.

The specification of the data address in the memory is illustrated in Figure 4-17. The location of a pixel on the screen is determined by 19-bit address. Of these 19 bits the bits from

0 to 9 (10 bits) represent the column number of a current pixel in the active window of 640 columns x 480 lines resolution. The bits 10 through 18 (9 bits) represent the row number.

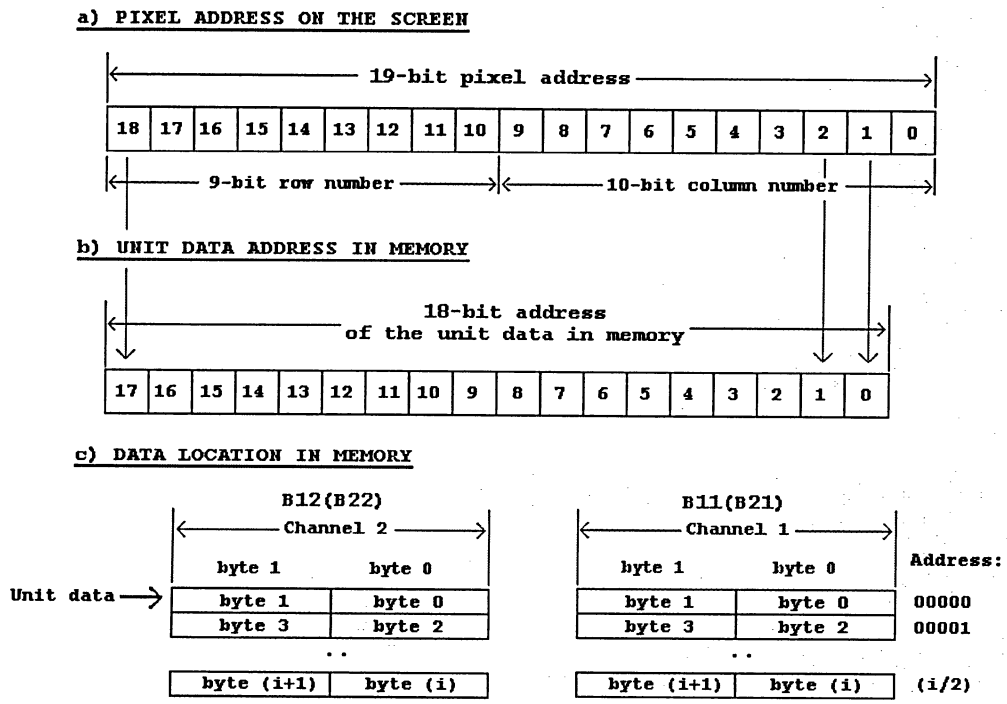


Figure 4-17. Pixel address on the screen and location in memory.

The address of the data in memory (SRAM) corresponds to the pixel address on the screen but the LSB is omitted. This is because the data is written to the memory in pairs, i.e. two adjacent pixels data is written to the SRAM by one write cycle.

The schematics of the internal architecture of FPGA is depicted in Figure 4-18. VPM captures data coming from ICM, stores data in the on-board SRAM and outputs processed data to standard VGA devices for control purposes. The VPM is implemented on the Reconfigurable platform based on the Xilinx FPAGA XC2VP100. It includes the following IP cores which are designed in VHDL [65] using the Xilinx' ISE.

- STROBE GENERATOR (STROBE_GEN);
- CAMERA DATA CAPTURE (CDATA_CAPTURE);
- WRITE CONTROL (WR_CONTR);
- VGA CONTROL (VGA_CONTROL);
- READ CONTROL (RD_CONTR);
- RAM BLOCK CONTROL (RAMB16_CONTR);

- 3D SYNTHESIS (SYNTH_3D);
- VGA DATA OUTPUT (VGA_DATA);
- SRAM CONTROL (MEMCHIP_CONTR).

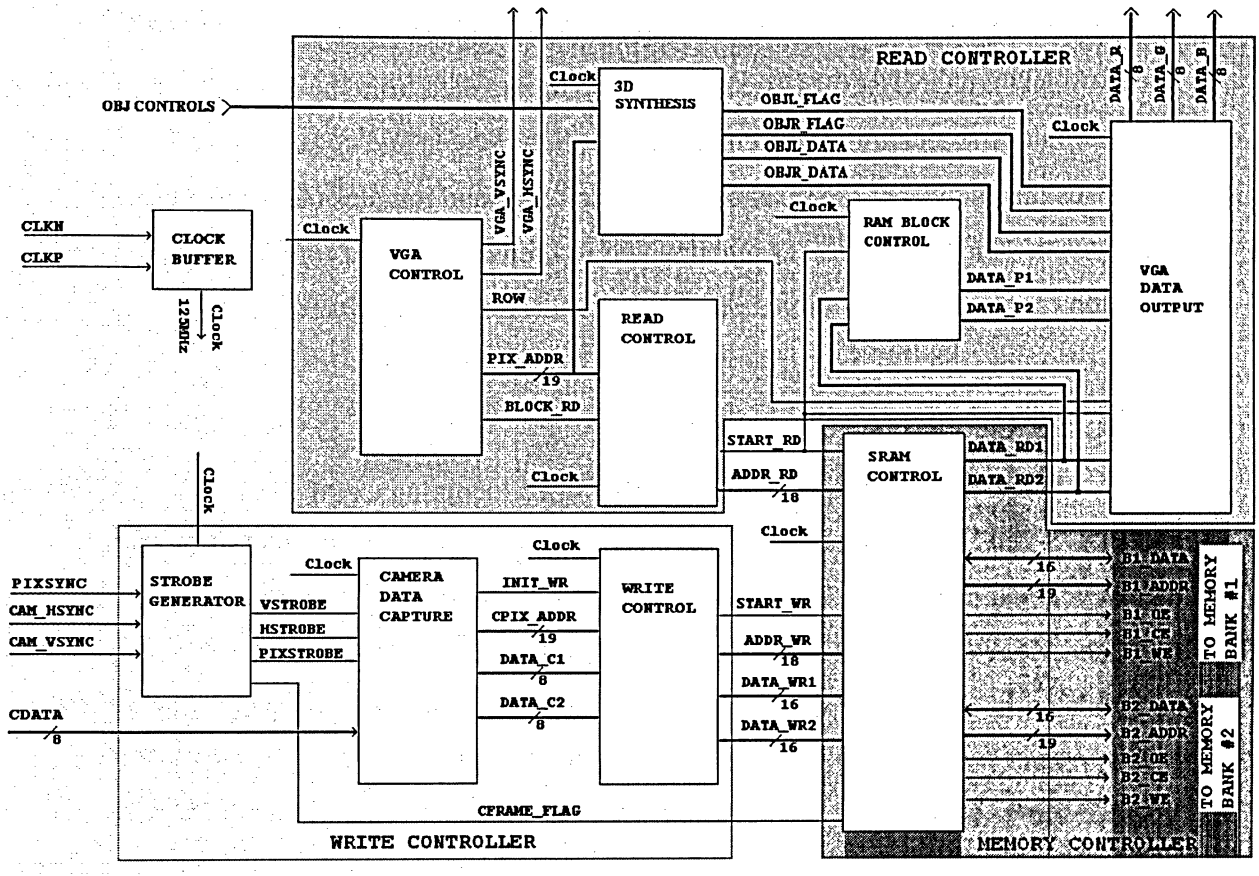


Figure 4-18. Video Processing Module block diagram.

All the IP cores are the components of three main functional subsystems: READ, WRITE and SRAM controllers.

READ controller consists of four IP cores which work as follows.

VGA CONTROL IP core generates vertical and horizontal synchronization signals for a standard VGA device (VGA_VSYNC and VGA_HSYNC) and the address of the current pixel according to the geometric location of the current scanning pixel. Since the actual pixel data is written in the memory in pairs (as was explained above) the system can read data in pairs too. The READ CONTROL IP core generates START_READ signal and ADDR_RD, which is the location of the unit data in memory, to initiate reading process by the SRAM CONTROLLER and to get the needed data after four clock cycles. The data then goes to the VGA DATA

OUTPUT IP core for generating actual RGB signals for the current pixel on the VGA screen. At the same time, the RAM BLOCK CONTROL IP core provides the data of the pixels brightness for the corresponding pixels of the previous line. It's needed for proper color data assignment, i.e. to get all three color components for a given pixel.

WRITE CONTROLLER catches the incoming data from the ICM, generates strobes for calculation of the location of the current pixel on the screen and correspondingly in memory, calculates the address of the pixel and of the unit data and initiates the writing process. It also generates the CFRAME_FLAG signal which is a reference signal for the memory banks selection for different operational phases of the SRAM CONTROLLER.

STROBE GENERATOR IP core

STROBE GENERATOR IP core transforms incoming CAM_HSYNC, CAM_VSYNC and PIX_SYNC signals into positive strobes HSTROBE, VSTROBE and PIXSTROBE one clock cycle wide each. The corresponding timing diagrams are presented in Figure 4-19, Figure 4-20, and Figure 4-21. There are two internal counters COUNT_HS and COUNT_VS which count the number of clock pulses when CAM_HSYNC and CAM_VSYNC strobes are high correspondingly. When the value of these counters equals to 10, the strobes HSYNC and VSYNC are generated.

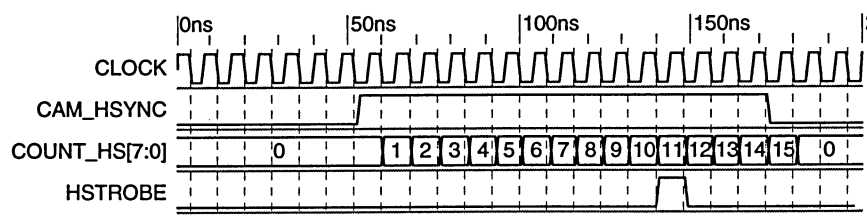


Figure 4-19. STROBE_GEN IP core: HSTROBE signal generation.

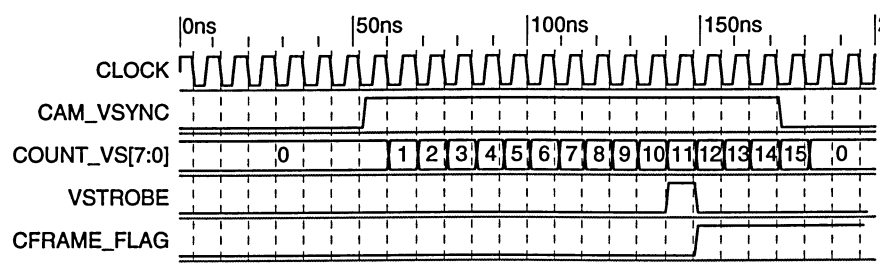


Figure 4-20. STROBE_GEN IP core: VSTROBE and CFRAME_FLAG signals generation.

Each CAM_VSYNC strobe inverts CFRAME_FLAG signal value. This signal is used for selection of the memory banks (SRAM chips) operational phases: reading or writing. For details see the descriptions of the MEMCHIP_CONTR IP core.

The PIXSTROBE signal is defined by the values of two internal signals mclk_int and mclk_int2 by the following expression: ***PIXSTROBE = (mclk_int) AND (NOT mclk_int2).*** Here the signal mclk_int is set to the value of the PIX_SYNC strobe on the falling clock edge, and mclk_int2 sets its value equal to mclk_int on the falling edge of mclk_int signal.

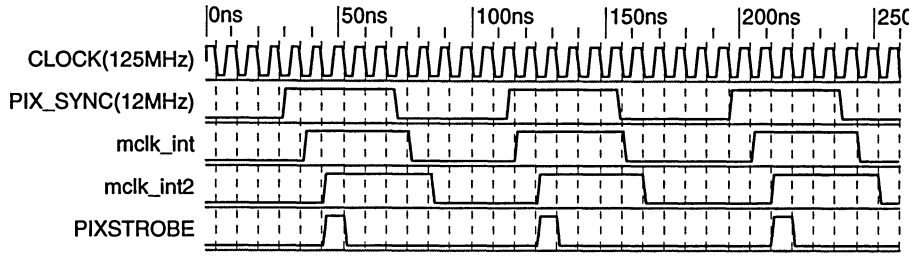


Figure 4-21. STROBE_GEN IP core: PIXSTROBE signal generation

CAMERA DATA CAPTURE IP core

CAMERA DATA CAPTURE IP core counts the coming pixel strobes (PIXSTROBE) and lines (number of HSROBE pulses), generates the address of the current pixel and INIT_WR signal which confirms that the data is captured and ready for storing in memory. The number of pixels is incremented with each PIXSTROBE as it is indicated in Figure 4-22.

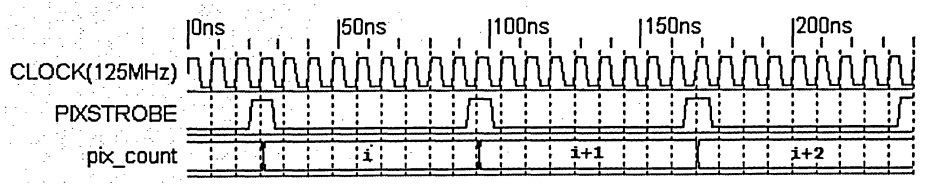


Figure 4-22. CDATA_CAPTURE IP core: pixels counting.

With each HSTROBE coming the pixel counter (pix_count) is initiated (zeroed) and the number of lines in a frame is incremented, see Figure 4-23 when HSTROBE is detected.

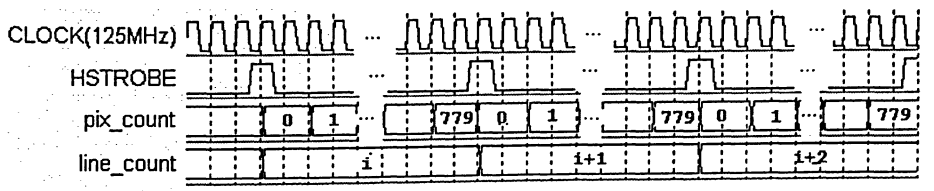


Figure 4-23. CDATA_CAPTURE IP core: lines counting.

Lines counter (line_count) is initiated when VSTROBE is detected, as depicted in Figure 4-24.

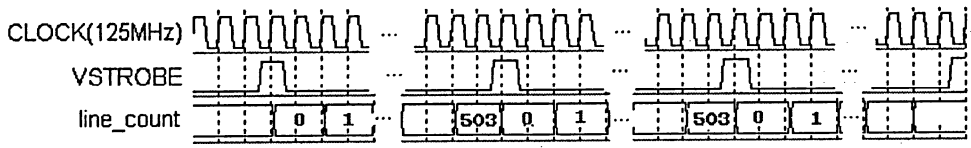


Figure 4-24. CDATA_CAPTURE IP core: lines counter reset.

Pixels and lines counters are concatenated to form the address of a pixel on the screen. The address is 19 bit wide of which 10 lower bits present the column number and 9 most significant bits present the line number. It is illustrated in Figure 4-17.

When the data is captured from both CMOS sensors, the CDATA_CAPTURE module generates signal INIT_WR, as shown in Figure 4-25. Internal signal clk_count counts the clock pulses and initiates its value by each PIXSTROBE signal. Te content of the clk_count defines when the data from one and another channel can be captured. When clk_count signal's value is '0', the data from the channel '1' is latched into the register reg_cam1, when clk_count = '4', the data from the channel '2' is latched into the register reg_cam2. When clk_count = '5', the content of the registers reg_cam_1 and reg_cam2 is connected to the output data lines DATA_C1 and DATA_C2 correspondingly and on the next clock cycle the INIT_WR signal is asserted.

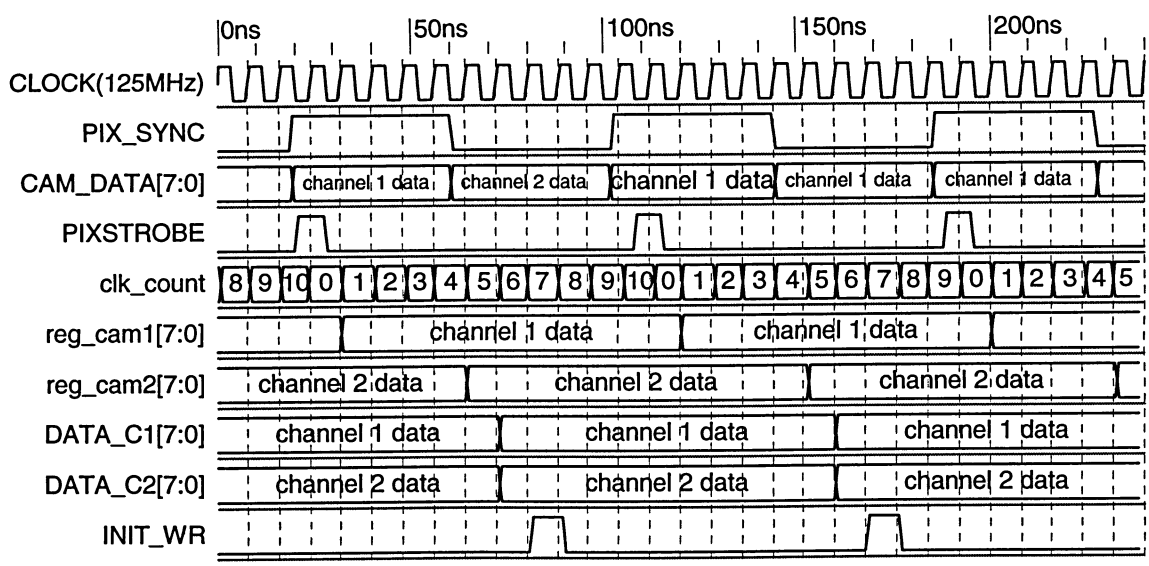


Figure 4-25. CDATA_CAPTURE IP core: timing diagram.

WRITE CONTROL IP core

Upon detection of INIT_WR signal, the WRITE CONTROL IP core combines the values of two consecutive pixels in one 16-bit wide word and when the word is filled with data it generates START_WR signal and at the same time it asserts the data to be written to SRAM on the data lines DATA_WR1 and DATA_WR2 which are 16-bit wide each. It also asserts the address of the data unit to be written to SRAM. This address is an 18-bit number which is actually the CPIX_ADDR value without its LSB (i.e. bit 0). Thus, the VPM writes pixel data to memory in pairs.

The timing diagram of the WR_CONTROL IP core is depicted in Figure 4-26. On detecting the INIT_WR signal the 8-bit values are latched to the upper or lower bits of the registers data_c1_int and data_c2_int depending on the value of the CPIX_ADDR lowest bit. Thus, the pairs of consecutive bytes are recorded into the 16-bit internal registers. Each second INIT_WR strobe the value of the CPIX_ADDR signal's LSB becomes '0'. It indicates that the pair of bytes filled the internal registers and the system can initiate writing process, i.e. storing data in memory. The one clock cycle wide strobe START_WR initiates memory controller to write data in memory.

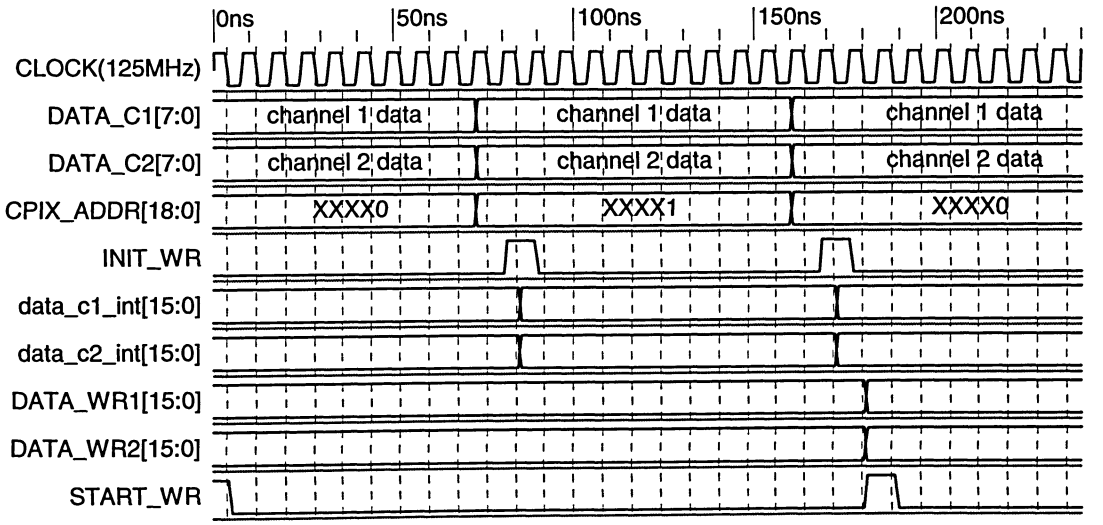


Figure 4-26. WR_CONTR IP core: timing diagram.

SRAM on the reconfigurable platform comprises two pairs of independently controlled IDT71V416 chips. Each chip provides storage space for 256K of 16-bit data.

VGA CONTROL IP core

VGA CONTROL IP core generates vertical and horizontal synchronization signals in accordance with the timing requirements for the standard VGA output device depicted in Figure 4-27. It calculates the coordinates of the current pixel which is to be output to a VGA monitor and the corresponding pixel address (PIX_ADDR). The structure of the pixel address is identical to that presented in Figure 4-17. It also generates BLOCK_RD signal which is passive low when the scanning point is within the active window (active portion of the scanning line which corresponds to the timing slot D in Figure 4-27).

VGA_CONTROL IP core uses clock pulses to calculate all necessary timing parameters to generate HSYNC and VSYNC signals.

It generates also the ROW signal which is "zero" for even rows and "one" for odd rows. This signal is used for color decoding algorithm which is implemented in RAM BLOCK CONTROL (RMB16 _CONTR) module

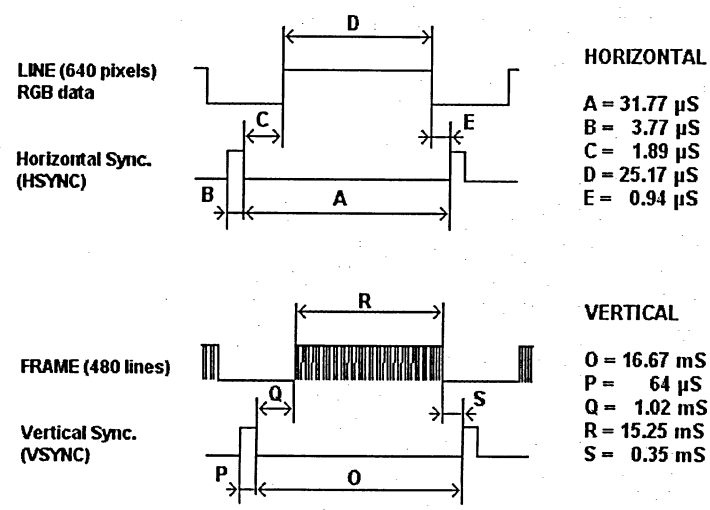


Figure 4-27. VGA timing requirements.

READ CONTROL IP core

READ CONTROL IP core generates START_RD signal for each even value of the PIX_ADDR which comes from the VGA CONTROL module. It means, that the data should be read from SRAM in pairs, in the same manner as it is written. START_RD signal generation is conditioned by the signal BLOCK_RD which is passive low (i.e. it doesn't affect reading) when the current pixel is located in the active window area, see Figure 4-28.

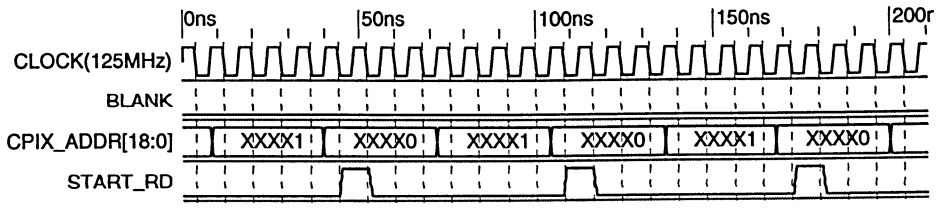


Figure 4-28. RD_CONTROL IP core: timing diagram.

SRAM CONTROL IP core

SRAM CONTROL IP core distributes control signals for both SRAM chips (both memory banks). The operational phases of the SRAM chips are writing or reading. At the same moment of time only one operational phase can be active for a particular SRAM chip. These phases are defined by the value of the CFRAME_FLAG signal. When CFRAME_FLAG signal's value is high, then the VPM writes to the bank 1 (namely chip number 1) and reads from the bank 2 (chip number 2). These phases are depicted in Figure 4-15.
When data is being written to the SRAM, the correspondent SRAM bank can't be used for other purposes when writing is in progress.

The reading data phase purpose is to extract data for the following output to a VGA device. The output to VGA is implemented in the system just for demonstration purposes to make sure that the system is working properly. The corresponding process (designed using the VHDL code) can be changed as needed to perform any task. For each frame there is a timing slot of 33.33mS when any data can be extracted and processed as needed. The timing for the write and read operations to and from SRAM is shown in Figure 4-29 and Figure 4-30 correspondingly.

On the rising edge of the clock following the falling edge when START_RD is asserted the value of the ADDR_BUS is latched into the address register of the SRAM chip. Then, after two clock cycles, the data can be read from the bidirectional data ports of the chip. SRAM CONTROLLER (MEMCHIP_CONTR IP core) reads data after three clock cycles after it detects active (high) START_RD strobe.

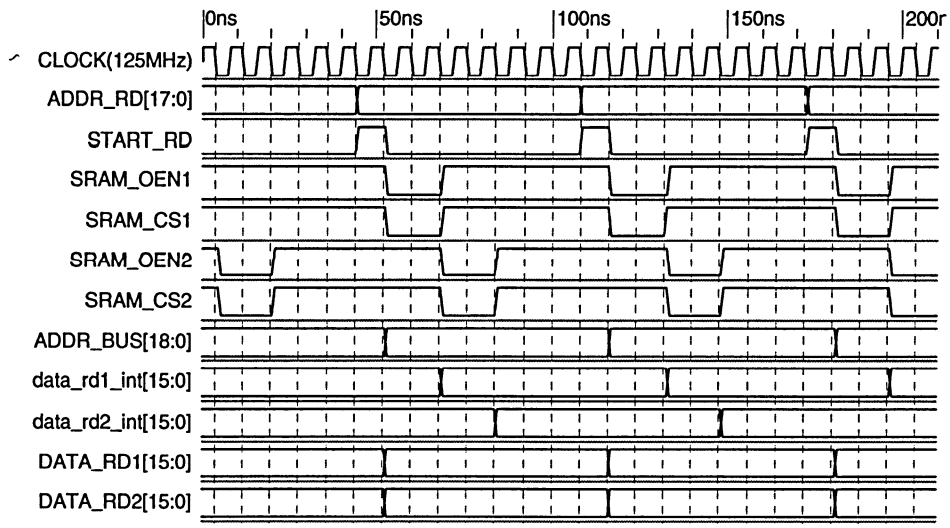


Figure 4-29. MEMCHIP_CONTR IP core: timing diagram, READ operation.

For the visualization system the MEMCHIP_CONTR IP core outputs the data from the previous reading cycle to the DATA_RD1 and DATA_RD2 ports on the next clock cycle after START_RD is asserted. The new data is latched into the internal registers data_rd1_int and data_rd2_int after four clock cycles following the detection of the START_RD signal as explained above. Signal SRAM_OEN (output enable) should be active (low) during reading data from the chips' data ports.

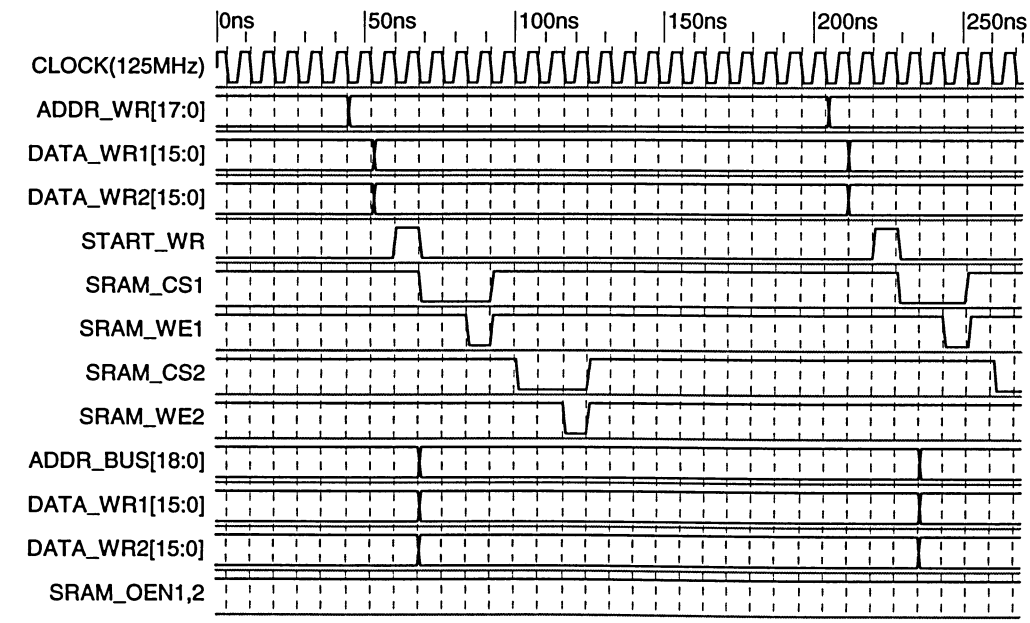


Figure 4-30. MEMCHIP_CONTR IP core: timing diagram, WRITE operation.

Writing operation is initiated by the START_WR signal. The writing procedure first writes 16-bit data to the first SRAM chip of the memory bank and then to the second one. The address lines (19-bit) of each SRAM chip are connected to the FPGA output ports on which the address value is set. The data lines (16-bit) of each SRM chip are connected to bidirectional FPGA's ports DATA1 and DATA2.

The control signals for each SRAM chips are the following:

OE1, OE2 - output enable1, 2 (active low);

CS1, CS2 – chip select 1, 2 (active low);

WE1, WE2 – write enable (active low);

RAM BLOCK CONTROL IP core

RAM BLOCK CONTROL IP core (RAMB16_CONTR) gets the data which is just read form SRAM from the MEMCHIP_CONTR IP core, stores it in the FPGA's internal Block RAM and outputs the pixel values data corresponding to the previous line. This data is used for the color decoding procedure. The process of reading and writing to the internal Block RAM is initiated by the START_RD signal. The address of the data in the Block RAM is the lowest 9 bit of the ADDR_RD. It is actually the column number for the current pixel (actually for a pair of pixels). The timing diagram for the RAMB16_CONTR IP core is presented in Figure 4-31.

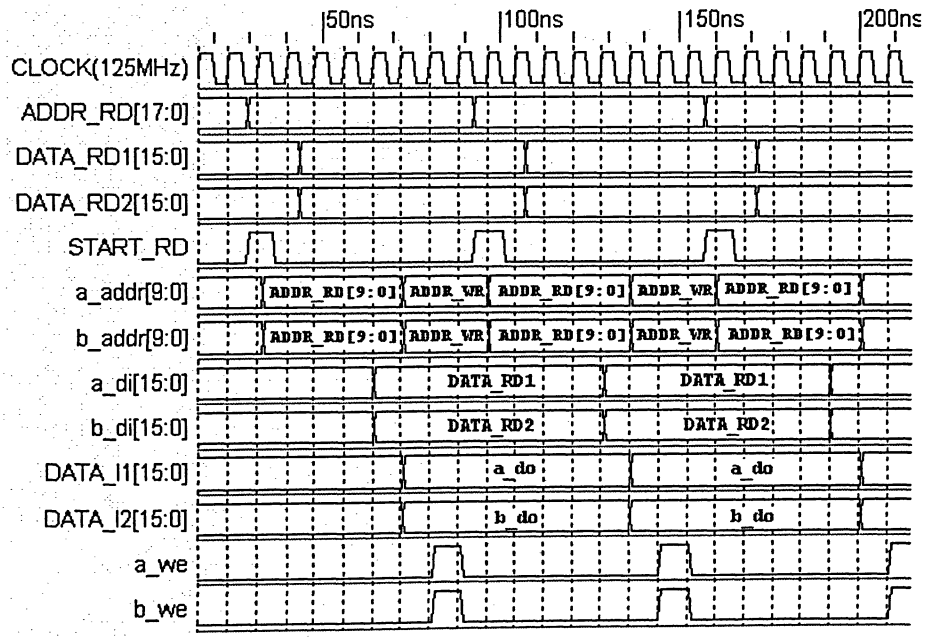


Figure 4-31. RAMB16_CONTR IP core: timing diagram.

The writing to the internal RAM blocks is initiated by the asserted START_RD signal. When START_RD is high, 10 bits of the ADDR_RD are connected to the RAM blocks' address lined a_addr and b_addr. Two RAM blocks are used for storing previous line data for two video channels. On the fourth clock after START_RD is detected the current data which is set on the output ports of the MEMORY CONTROLLER is connected to the data lines of the RAM blocks. On the fifth clock, the data is read from the RAM blocks. Now, the previous line data for the current pixel address is locked into DATA_I1 and DATA_I2 output ports. On the sixth clock pulse the current data is written to the RAM block to be used for the next scanning line. The writing address is derived from the reading address by the following formula:

ADDR_WR[9:0] = NOT ADDR_RD[9] & ADDR_RD[8:0].

Here the sign '&' stands for concatenation (as it is used in VHDL).

Thus, for reading and writing procedures two switching areas of RAM blocks are used.

3D SYNTHESIS IP core

3D SYNTHESIS IP core generates images for virtual interactive 3-D objects. Its schematics is presented in Figure 4-32.

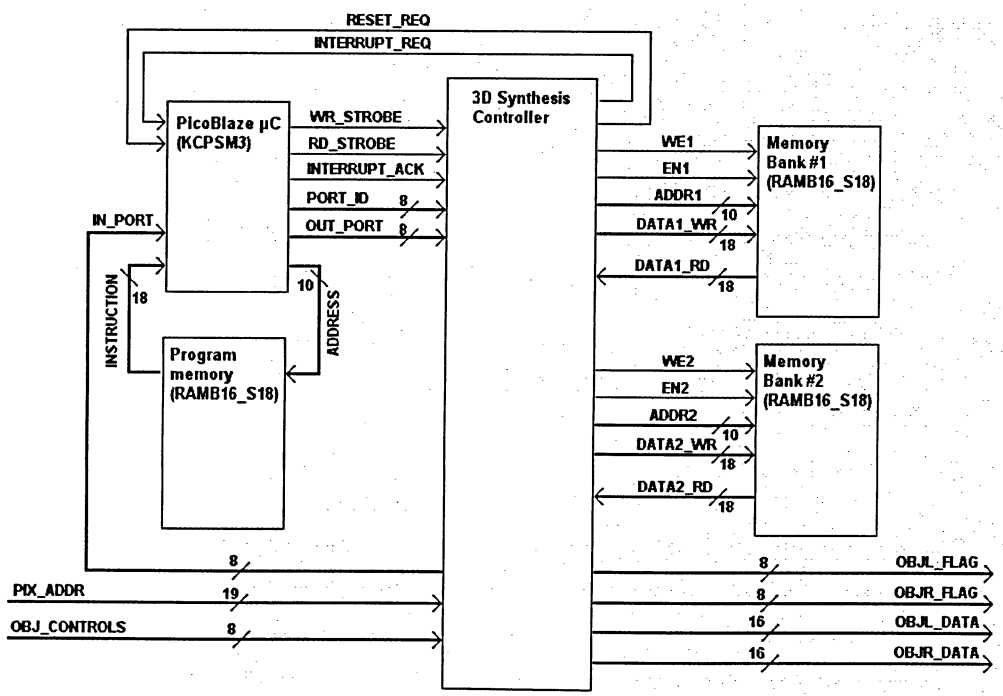


Figure 4-32. 3D Synthesis IP core schematics.

3D SYNTHESIS IP core includes the following core sub-modules: 3D Synthesis controller, PicoBlaze embedded microcontroller with its program memory RAM block and two other RAM Blocks as memory banks. 3D Synthesis controller performs all data processing for 3-D objects images generation. It delegates arithmetic computations to the PicoBlaze embedded microcontroller. The information on objects and data for rasterizing images is stored into two RAM Blocks. One memory bank contains the information on 3-D objects in the format presented in Table 4-3. The second RAM Block contains the rasterizing data for the elementary triangles composing 3-D virtual objects.

For example, object 1 in the table is a tetrahedron with 4 vertices and 4 surfaces of which two are visible on the projection plane.

Table 4-3. Structure of data in memory for a 3-D object.

ADDR	Note	
0		Number of objects=2
1		Object 1 offset=3
2		Object 2 offset=80
3	Object 1	Number of vertices = 4
4	Original	V1: x
5	coordinates	V1: y
6	of vertices	V1: z
7		V2: x
8		V2: y
9		V2: z
10		V3: x
11		V3: y
12		V3: z
13		V4: x
14		V4: y
15		V4: z
16	Translation	tx
17	Translation	ty
18	Translation	tz
19	Rotation	α
20	Rotation	β
21	Rotation	γ
22	Transformed	V1: xt
23	coordinates	V1: yt
24	of vertices	V1: zt
25		V2: xt
26		V2: yt
27		V2: zt
28		V3: xt
29		V3: yt
30		V3: zt
31		V4: xt
32		V4: yt
33		V4: zt

ADDR	Note	
34		Number of surfaces=4
35		Num.of visible surfaces=2
36		Surface 1 color = Color1
37		Surface 2 color = Color2
38		Surface 2 color = '0'
39		Surface 2 color = '0'
40	Ordered	22 (V1:xt)
41	vertices of	28 (V3:xt)
42	visible	25 (V2:xt)
43	surfaces	25 (V2:xt)
44		28 (V3:xt)
45		31 (V4:xt)
46		22 (V1:xt)
47		25 (V2:xt)
48		31 (V4:xt)
49		22 (V1:xt)
50		31 (V4:xt)
51		28 (V3:xt)
52	Coordinates	V1x
53	of vertices	V1y
54	on projection	V1z
55	plane	V2x
56		V2y
57		V2z
58		V3x
59		V3y
60		V3z
61		V4x
62		V4y
63		V4z

ADDR	Note	
64	Code of layout	001
65		52 (V1x)
66		55 (V2x)
67		58 (V3x)
68		010
69		52 (V1x)
70		58 (V3x)
71		61 (V4x)
72		000
73		-
74		-
75		-
76		000
77		-
78		-
79		-
80	Object 2	Number of vertices = 4
81		V1: x
82		V1: y
83		V1: z
84		V2: x
85		V2: y
86		V2: z
87		V3: x
88		V3: y
89		V3: z
90		V4: x
91		V4: y
92		V4: z
...		...

For a triangle which is considered as an elementary surface (elementary polygon) for rasterizing, seven possible variants of vertices layout are possible, as depicted in Figure 4-33.

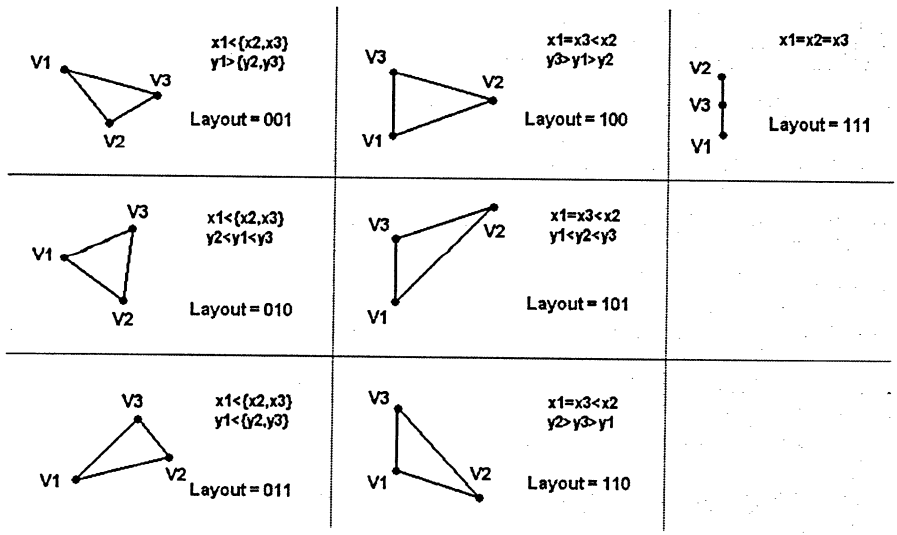


Figure 4-33. Possible layouts of elementary triangle on the projection plane.

The 3D SYNTHESIS IP core operates as follows, see Figure 4-34.

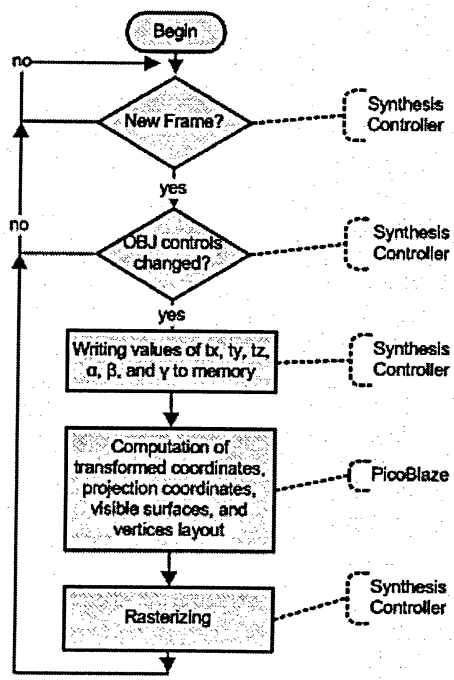


Figure 4-34. 3D SYNTHESIS module functionality algorithm.

First, the structure of the objects is written to the corresponding RAM Block (in this project the corresponding Block RAM is initialized by the data in the VHDL code). Then, the state of OBJ_CONTROLS is checked and the updated values of translation parameters (t_x, t_y, t_z, α, β, and γ) are written to memory. After that, the Synthesis Controller passes the data to the

PicoBlaze for computing new (transformed) coordinates of vertices and the coordinates of the projected vertices. PicoBlaze also computes layout codes for visible surfaces and sets their colors. The resulting data is then written to the memory through the Synthesis Controller. After that, the Synthesis Controller performs rasterizing of the line segments and writes the data to the second RAM Block.

The presentation of the rasterized line segments and surfaces in memory is shown in Table 4-4.

Table 4-4. Data format of the rasterized line segments and surfaces in memory.

Address	Byte 1	Byte 0
0	Y coordinate (beginning of the object)	
1	Yend coordinate	
2	Xstart	Xend
3	Xstart	Xend
4	Xstart	Xend
5	Xstart	Xend
8	...	
n	Xstart	Xend

When the current value of the PIX_ADDR signal corresponds to the beginning of the rasterized image coordinate, the OBJL_FLAG and/or OBGR_FLAG signals are generated and the value corresponding to the surface color is set on the OBJ_LDATA and/or OBJR_DATA. The VGA DATA OUTPUT module substitutes the real-time video data with the data for the synthesized objects when OBJL_FLAG or OBJR_FLAG are set.

For the current version of the design, the objects location is limited by a cube of approximately 11 x 11 x 11 m^3, as illustrated by Figure 4-35. The viewing points (or cameras focus) are placed in the middle of the front side. The shift between cameras is 0.16m.

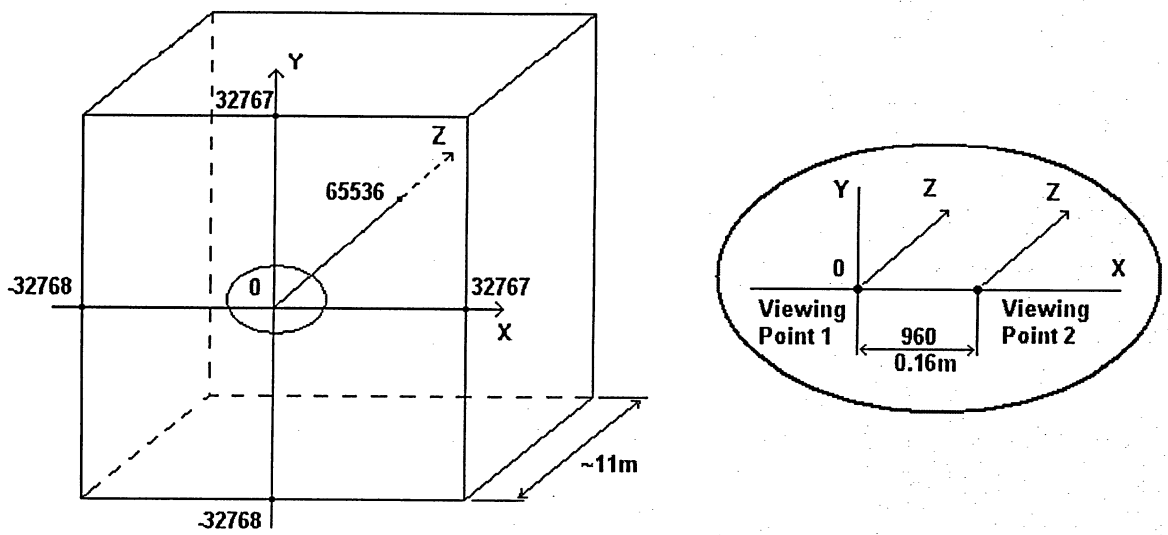


Figure 4-35. Virtual object space configuration.

The actual distances are represented in 16-bit integers, thus 11m corresponds to 65536 distance units. Using this approach the focus distance is calculated as 780 units. All multiplications and divisions are performed on integer numbers. To provide an acceptable accuracy of calculations, which corresponds to 1 pixel on the projection plane (which is 1 unit), the value of focus is multiplied by 2^{14} so that for the calculation of *focus/z* in $x' = x \cdot focus / z$ equation the dividend is a 24-bit number and the divisor (z) is a 16-bit number in the range 0...65536. The values of x and y coordinates are in the range -32768...32768. They are represented as signed integers.

VGA DATA OUTPUT IP core

VGA DATA output IP core extracts all three colors data for each output pixel coming to the VGA device. It uses the data just read from the SRAM and the corresponding data from the previous line provided by RAMB16_CONTR IP core. It operates as follows.

The 16-bit value corresponding to the brightness of a pair of adjacent pixels is read from the external SRAM (*Bank1* and *Bank2*). The corresponding data from the previous line is read from the internal RAM Blocks. Six internal registers are used to latch the data: *rc0* gets bits (0...7) and *rc1* gets bits (8:15) from the 16-bit value corresponding to the current line; in its turn *rp0* gets bits (0...7) and *rp1* gets bits (8...15) from the 16-bit value corresponding to the previous line. Registers *rpp* and *rcp* get the values of *rp1* and *rc1* form the previous reading

step. These six registers allow assigning values for all three colors of each pixel. The registers assignment scheme is shown in Figure 4-36.

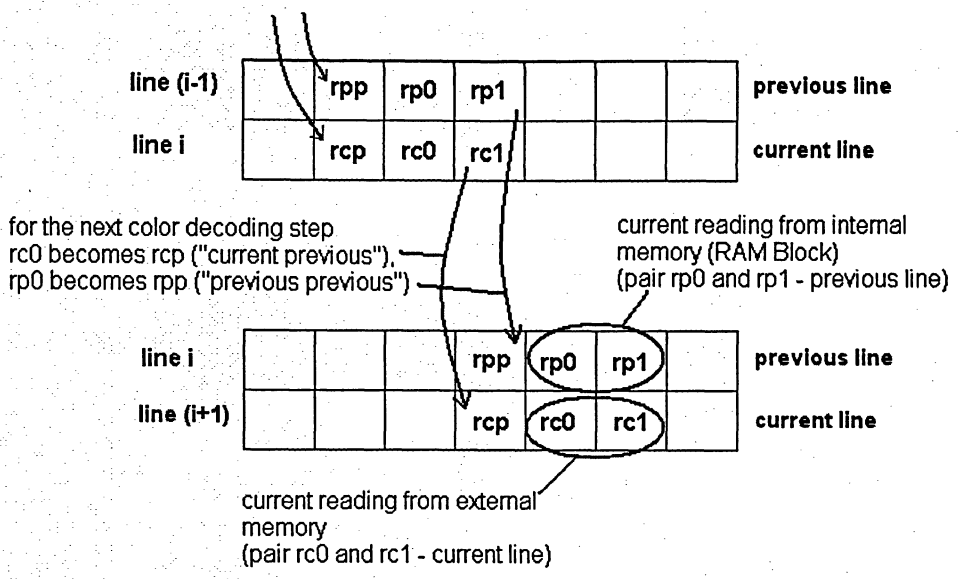
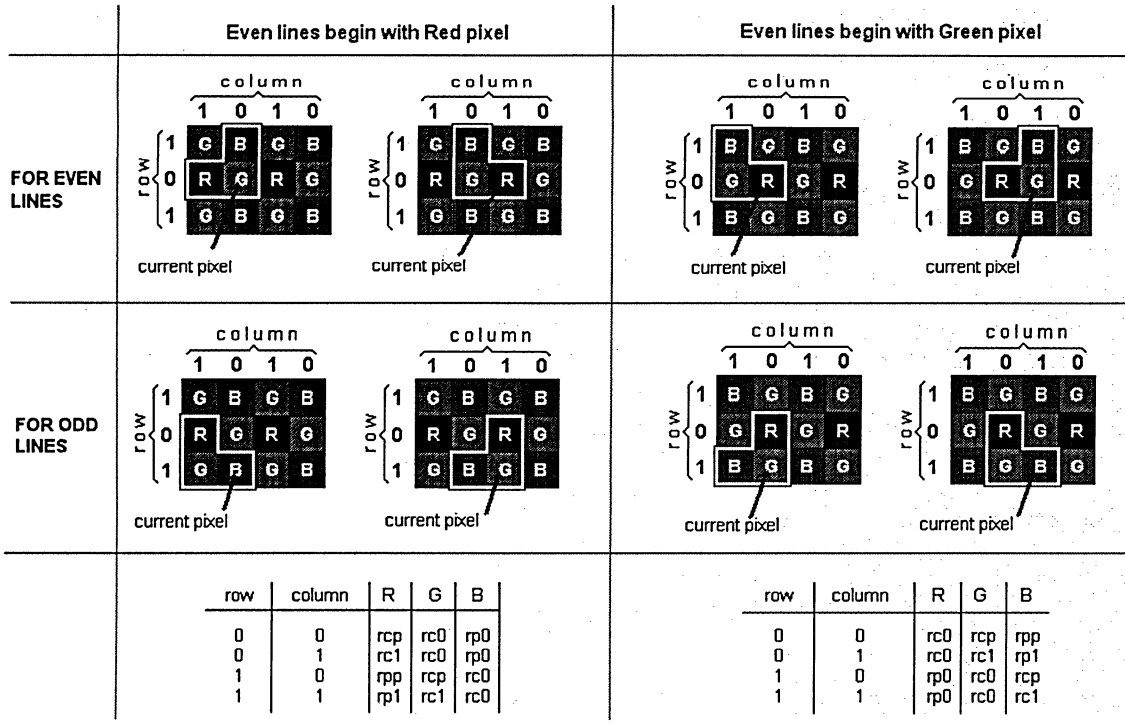


Figure 4-36. Data reading and internal registers assignment scheme.

Two cases for color assignment are possible depending on the starting point for the lines calculation, i.e. which line is considered even or odd. For each case the table of color matching can be determined and used for all three colors assignment for each pixel. Figure 4-37 illustrates this scheme.

Left side of Figure 4-37 corresponds to the case when the line 0 (and all other even lines) begins with Green pixel, its right side corresponds to the case when the line 0 (and all other even lines) begins with Red pixel. On the right side of these figures the tables of colors assignment are presented. One of the tables must be used for both channels in the design.



row	column	R	G	B
0	0	rcp	rc0	rp0
0	1	rc1	rc0	rp0
1	0	rpp	rcp	rc0
1	1	rp1	rc1	rc0

row	column	R	G	B
0	0	rc0	rcp	rpp
0	1	rc0	rc1	rp1
1	0	rp0	rc0	rcp
1	1	rp0	rc0	rc1

Figure 4-37. Color assignment scheme

Figure 4-38 illustrates the multiplexer-based implementation of the presented colors assignment algorithm.

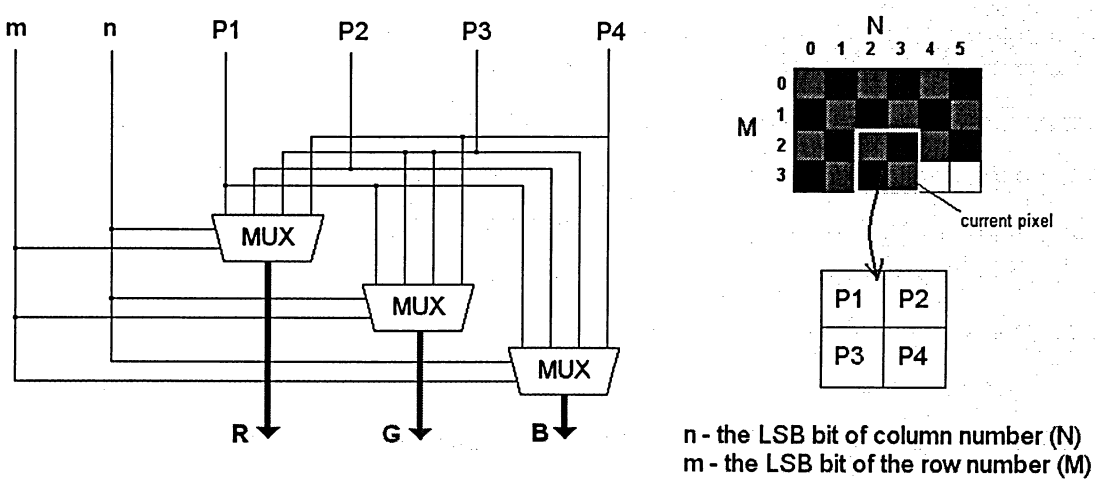


Figure 4-38. Implementation of the multiplexer-based color assignment algorithm.

Figure 4-39 represents the timing diagrams for the VGA_DATA IP core.

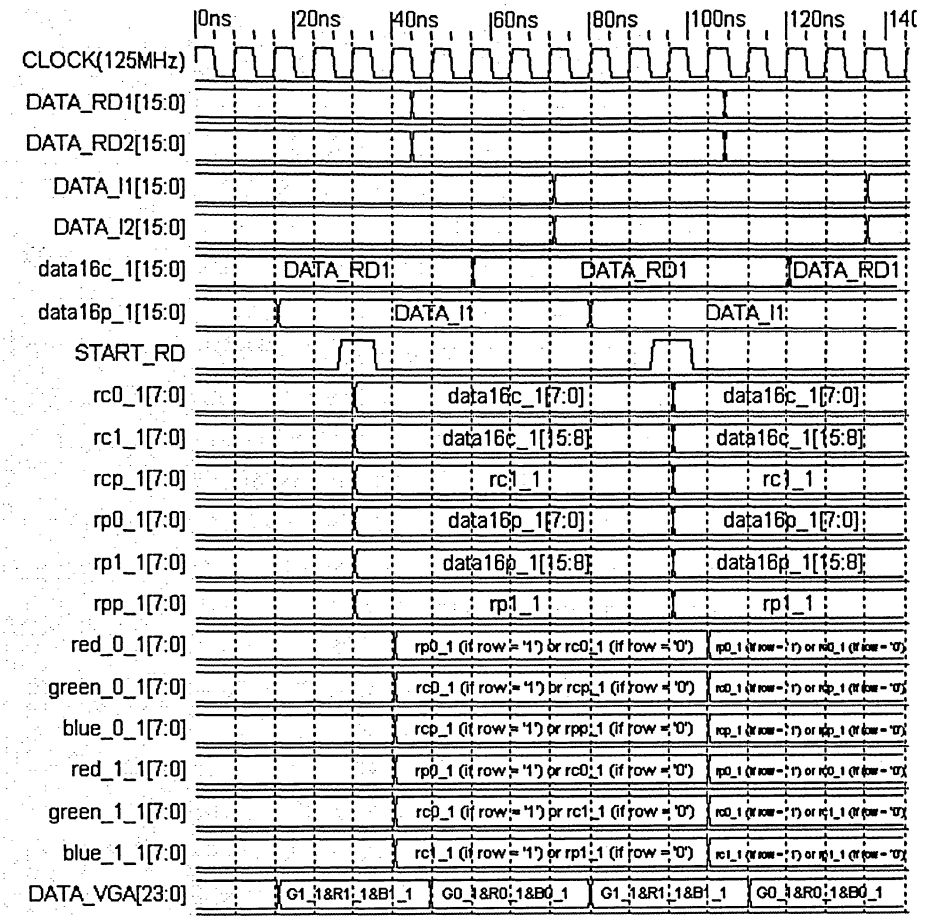


Figure 4-39. VGA_DATA IP core: timing diagram.

4.3.3. Visualization Output Module (VOM)

The block diagram of the VOM is presented in Figure 4-40. The VOM is built on the base of the Xilinx XC95144XL CPLD. It extracts each channel digital RGB data coming from the VPM through the LVDS buffer. It also distributes RGB data to two TI THS8134B DAC chips which drive two VGA outputs.

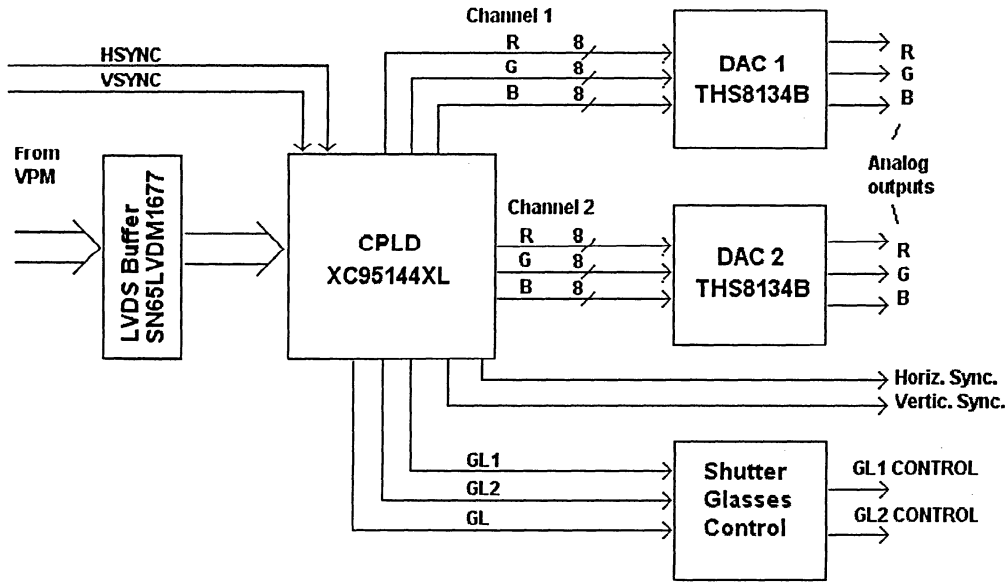


Figure 4-40. Block diagram of the Visualization Output Module.

The VOM generates control signals for shutter glasses switching. The schematics of the analog part of the Shutter Glasses Control module is illustrated in Figure 4-41.

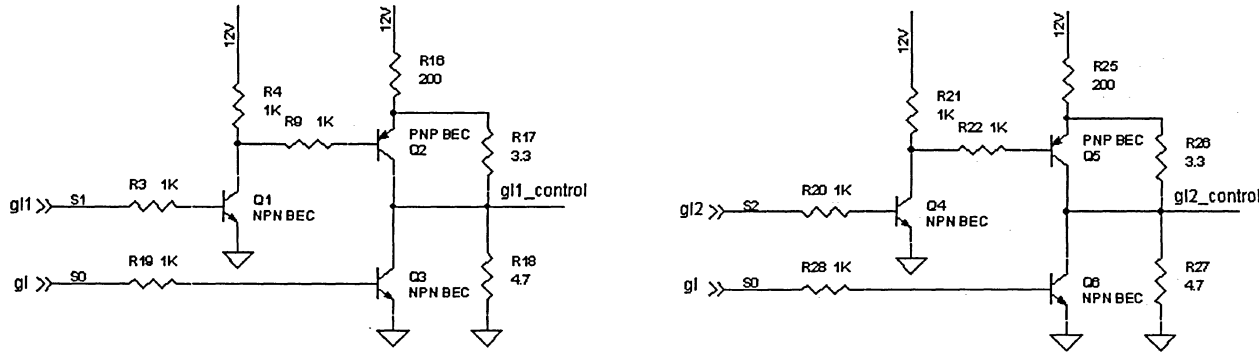


Figure 4-41. Shutter Glasses Control subsystem schematics.

This subsystem includes two identical channels, one for the left and the other for the right liquid crystal filter of the shutter glasses. The timing diagram of the *gl, gl1* and *gl2* digital control signals generated by CPLD signals is illustrated in Figure 4-42.

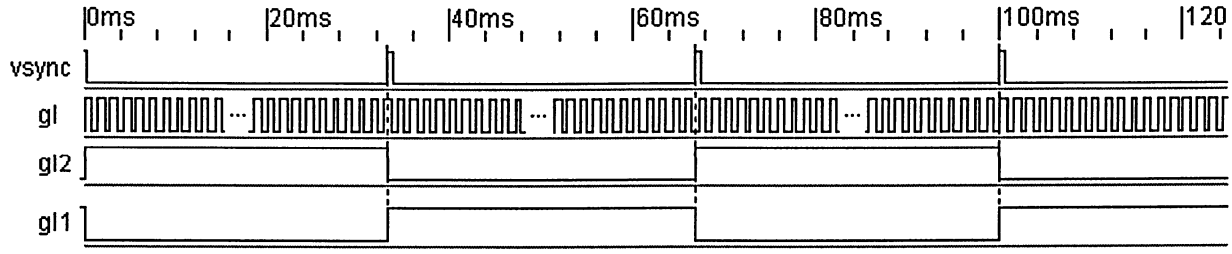


Figure 4-42. Control signals for shutter glasses generated by CPLD.

The frequency of the *gl* signal is approximately 2KHz. The rising edge of the VSYNC signal initiates change of the *gl2* signal which actually selects which eye of the shutter glasses is open and which one is closed.

The signals *gl1* and *gl2* feed the circuitry of the DACM board, which generates actual analog signals for the shutter glasses control. The timing for these signals, *gl1_contr* and *gl2_contr* is presented in Figure 4-43.

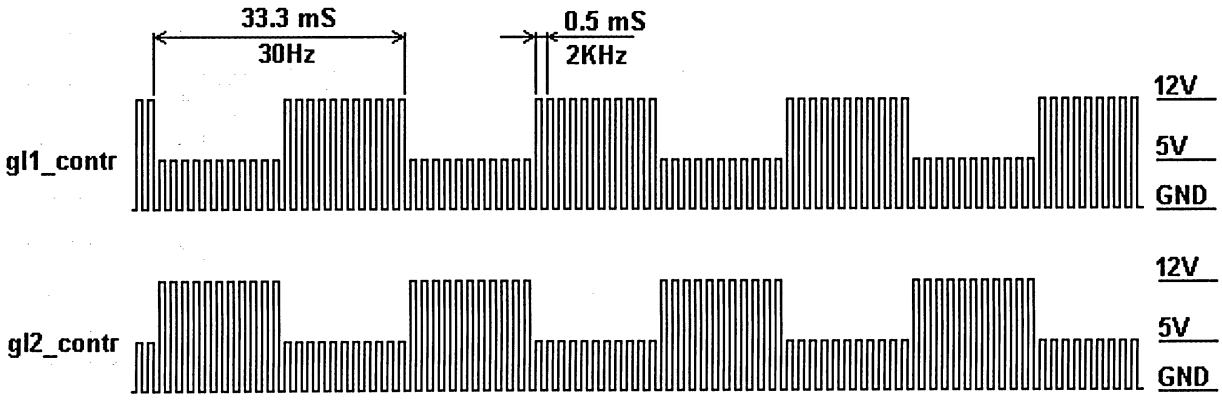


Figure 4-43. Shutter Glasses analog control signals: timing diagram.

4.4. Conclusion

This chapter presents the implementation of the system on Run-Time Reconfigurable FPGA Platform providing Interactive 3-D Video Environment which includes the following:

1. Development of architecture of three hardware modules comprising the system: Image Capture Module (ICM), Video Processing Module (VPM), and Visualization Output Module (VOM). VPM is built using the hybrid architecture implementing logic for processing of two incoming data streams and generating output data for stereo visualization, and embedded PicoBlaze microcontroller for algorithmic and computation intensive part of the system which deals with 3-D virtual objects synthesis. The architecture of ICM and VOM employs straightforward logic since these modules must provide the data rate of 25MBytes/Sec and 60Mbytes/Sec per channel correspondingly.
2. Design of the ICM, VPM, and VOM. The PCBs for the ICM and VOM are designed. The VPM design is based on the Reconfigurable Platform which employs Xilinx Virtex-2 XC2V1000 FPGA, two independently accessible SRAM banks and two LVDS buffers. ICM includes CPLD, two CMOS Sensors providing data for real-time stereo video system. The settings of the CMOS can be changed either through pressing the buttons on the ICM board (Reset, General Video Gain, Red, Green and Blue channels Video Gain adjustment), or via the provided GUI module which accesses the ICM board via the USB interface. The VOM includes the CPLD, two DAC chips and the schematics for providing control of the liquid crystal shutter glasses. All the IP cores are designed in VHDL in Xilinx' development environment ISE, version 7.1, service pack 2. The sub-module for the shutter glasses control deals with analog signals.
3. Implementation of the design: the ICM and VOM are manufactured, assembled and debugged. The IP cores of the ICM, VPM, and VOM are debugged. External logic analyzer HP54620C and the Xilinx' ChipScope Pro tool were used for debugging.
4. Simulation and verification of the design: the functionality of the IP cores and the whole system is simulated and verified. Testbench modules were used for simulation. The timing diagrams presented in this chapter represent the actual timing of the IP cores comprising the system based on the real images obtained using ChipScope Pro tool.

5. EXPERIMENTS AND RESULTS

5.1. Experimental Set-Up

The experimental set-up for the real-time stereo visualization is shown in Figure 5-1.

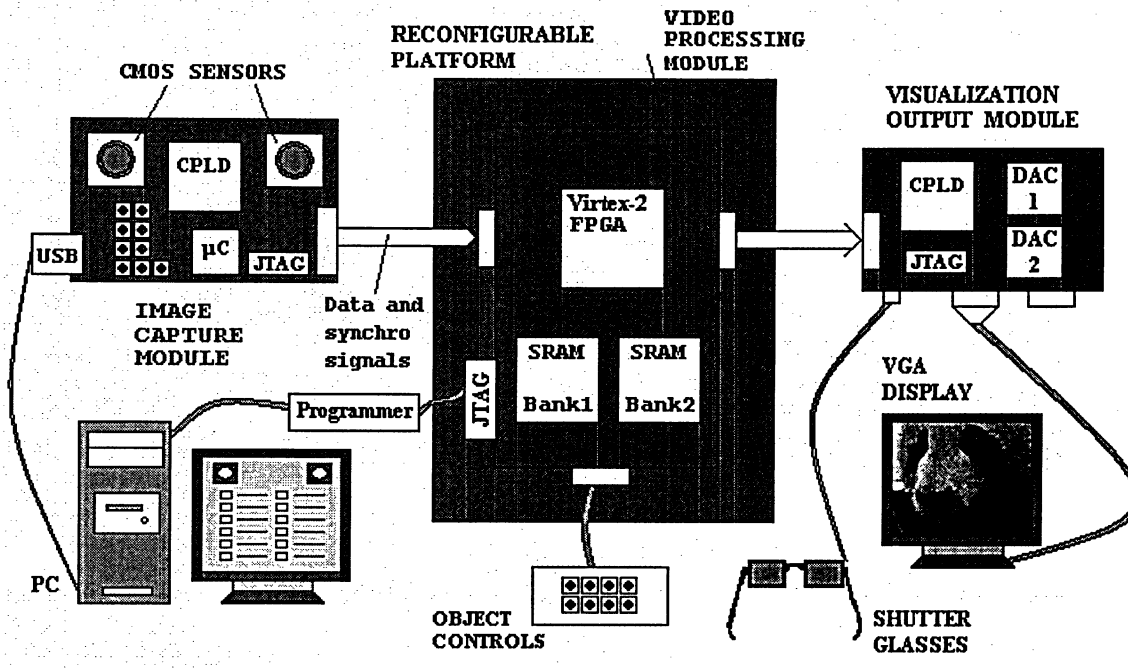


Figure 5-1. Experimental set-up for real-time stereo visualization.

ICM provides stereo video data to the VPM through the cable. Differential buffers on both transmitting and receiving ends are used for data transmission with the purpose to minimize cross-talks and interference. Differential buffers are also used for sending data from VPM to VOM.

CPLDs (XC95144XL) of the ICM and VOM are preprogrammed via JTAG ports using the Parallel Cable IV and the iMPACT utility of the Xilinx' ISE. CMOS Sensors of the ICM settings can be done either through the buttons on-board of the ICM module or via GUI communication with the ICM through USB port. VPM has to be programmed once after it is powered on.

Object Controls switches are connected to the FPGA's IOs. The controls are used for changing the state of the 3-D interactive controls (stereo graphic images) generated by the

system. One switch is used to switch output modes from visualization or the one channel image to visualization of the edge detection image on one of the output channels.

Shutter glasses are connected to the subsystem of the VOM which generates analog control signals switching liquid crystal glasses from dark to transparent state and vice versa.

All the system is designed using VHDL and Xilinx' ISE version 7.1.04i, see Figure 5-2.

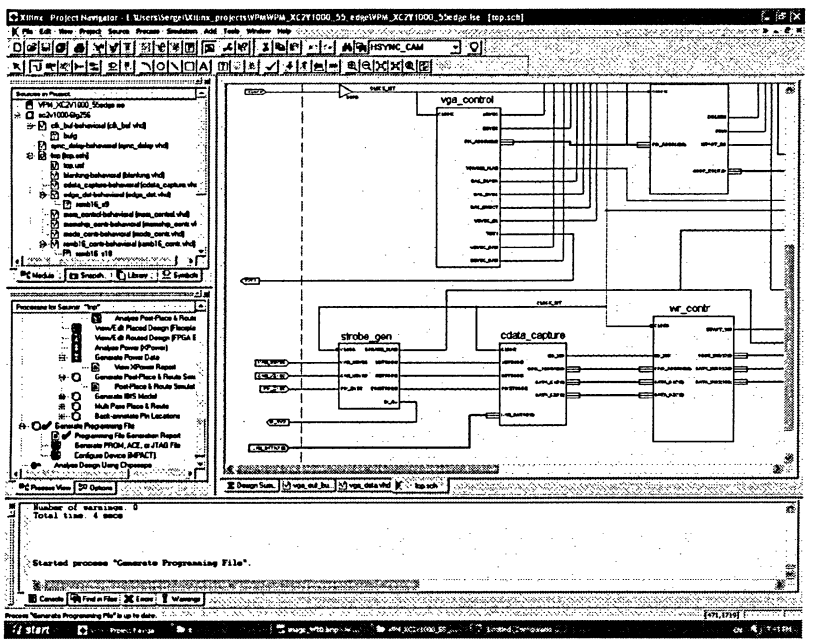


Figure 5-2. Designing system with Xilinx' ISE ver.7.1.04i.

The debugging of the system was done using the HP54620C logic analyzer, see Figure 5-3.

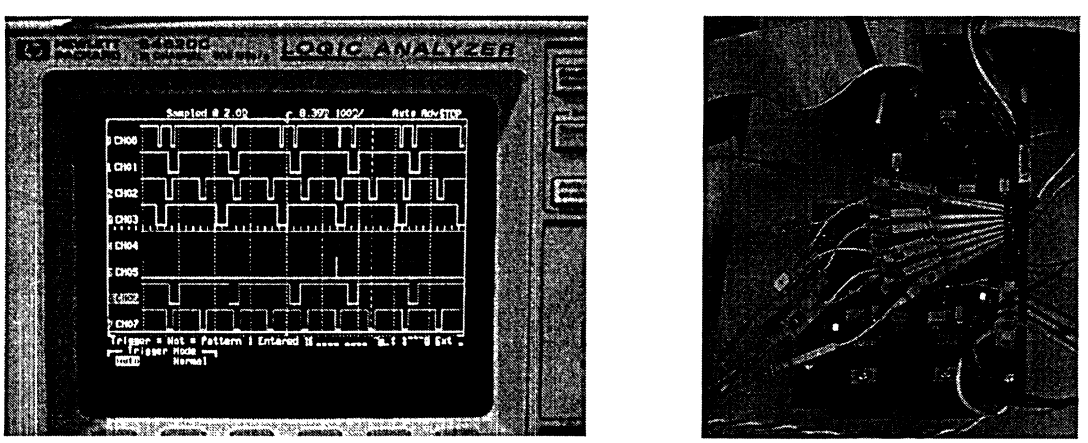


Figure 5-3. Debugging system with HP54620C logic analyzer.[25]

[25] All pictures courtesy of ERSL.

The Xilinx' tool ChipScope Pro, version 7.1i was also used for debugging of the system as illustrated in Figure 5-4.

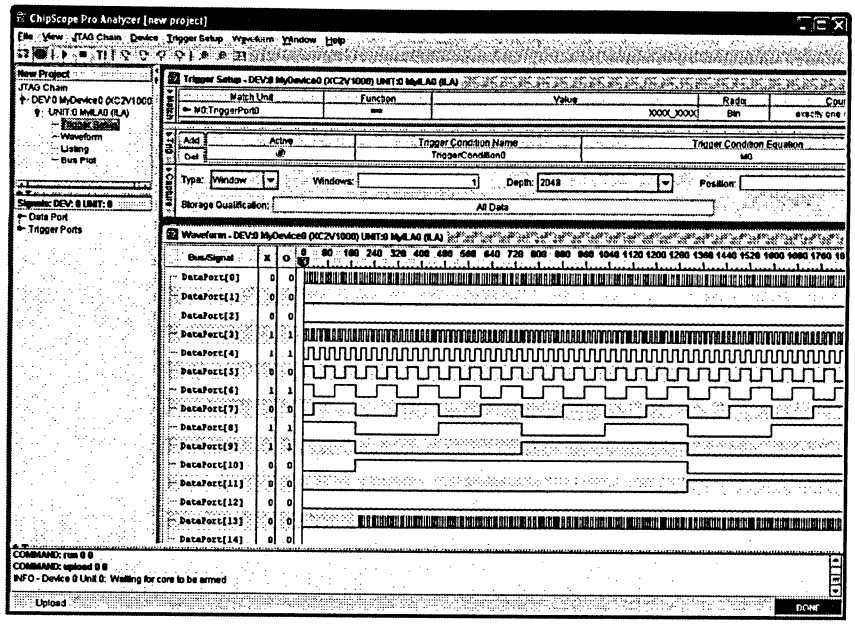


Figure 5-4. Debugging of the system with the Xilinx' ChipScope Pro tool.

The detailed photographic views of the ICM are presented in Figure 5-5. Two CMOS image sensors chips are located at a distance (stereo base) of 160mm which is wider than an average distance between human eyes. The wider stereo base provides better stereo effect. The captured data is passed to the VPM via differential buffers which eliminate interference on the data link from ICM to VPM. Using LVDS buffers also allows to place the ICM further from the processing module (VPM) as it may be needed for other possible applications (e.g. for robot vision systems). The buttons on the ICM allow changing the settings of the CMOS sensors' internal registers. By pressing the buttons the overall video gain can be incremented or decremented as well as video gain for each color component can be changed by pressing other groups of buttons. One button activates RESET for the PIC microcontroller. The microcontroller programs the CMOS sensors via i^2c interface. It also provides communication with the PC through the USB interface which employs USB UART controller (FTDI chip FT232BM). Using specially designed GUI module it's possible to change the value of any of the internal registers of the CMOS Image Sensor chips.

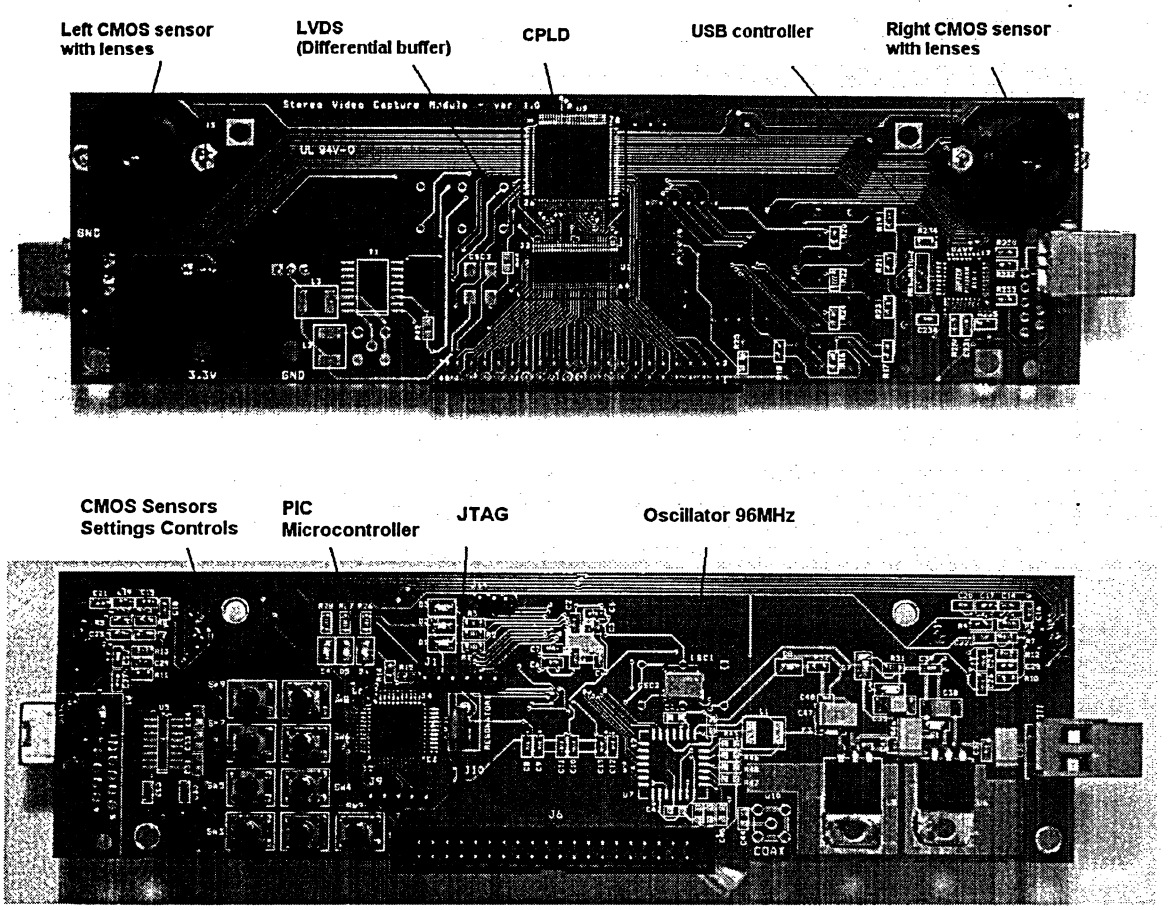


Figure 5-5. Photographic picture of the ICM (front and back sides).[26]

The photographic image of the FPGA-based reconfigurable platform is presented in Figure 5-6. It incorporates the Virtex-2 XC2V1000 FPGA with 10,000 logic cells (equivalent to 1,000,000 system gates). The on-board clock frequency is 106.25MHz which is the operational frequency of the system. The module includes two banks of external memory; each comprises two IDT71V416 SRAM chips. The incoming data from the ICM is received by the differential buffers LVDS 2 and then passed to the FPGA's IOs. The output color data is passed to the Visualization Output Module (VOM) through the differential buffers LVDS 1. The 3-D objects controls and modes selection switch are connected to the VPM through the J2 header. The programming of the FPGA chip is performed via JTAG and Parallel Cable IV.

[26] All pictures courtesy of ERSL.

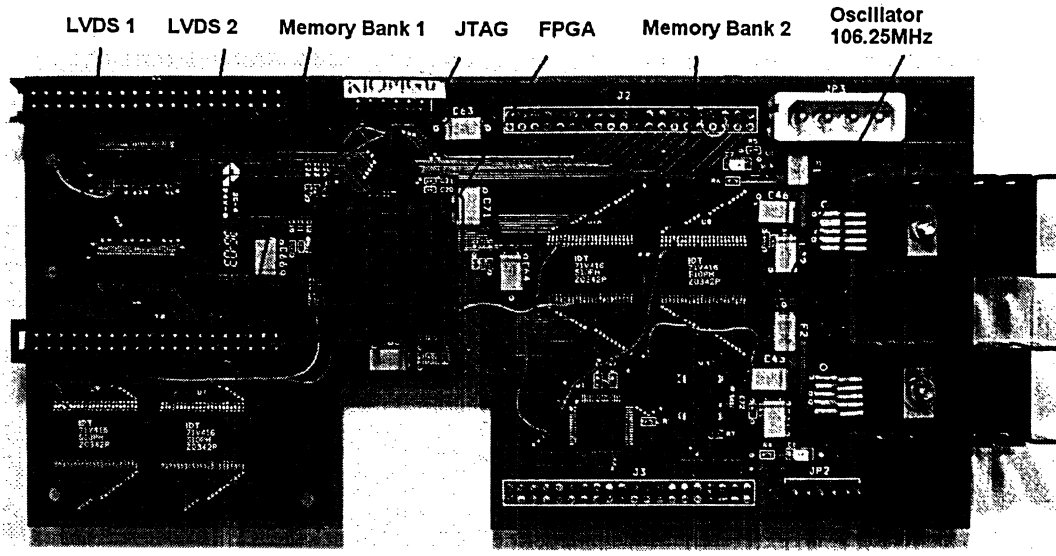


Figure 5-6. Photographic picture of the Reconfigurable FPGA-based platform.[27]

Virtex-2 chip doesn't incorporate hardwired microprocessor PowerPC as does Virtex-2 Pro chip. Soft wired microprocessor cores can be integrated into the system though. The presented system uses the simplest 8-bit PicoBlaze microprocessor core. For simulation free pBlazeIDE from Mediatronix was used [66].

Visualization Output Module is shown in Figure 5-7. It's built on the base of the Xilinx' XC95144XL CPLD which extracts each channel digital RGB data coming from the VPM through the LVDS buffer and distributes RGB data to two TI THS8134B DAC chips which drive two VGA outputs.

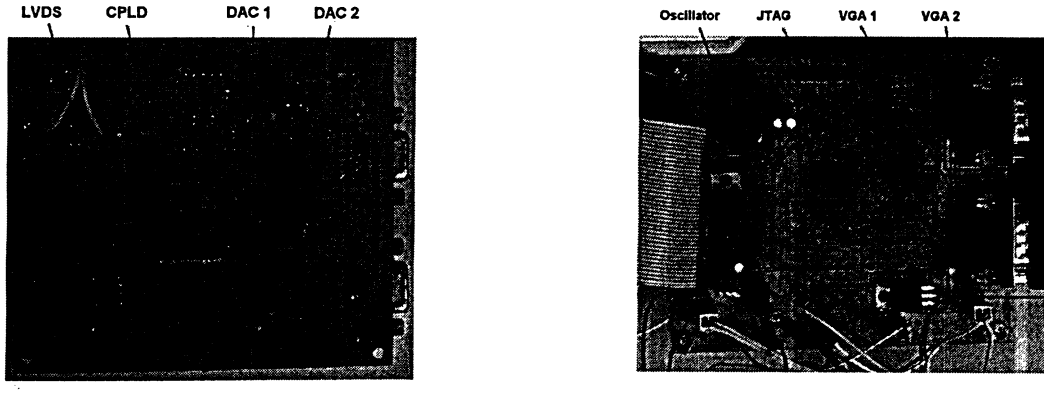


Figure 5-7. Photographic picture of the Visualization Output Module.

[27] All pictures courtesy of ERSL.

The last sub-module serves for driving the shutter glasses. It's built on the base of the Xilinx' XC9536XL CPLD which generates digital control signals which then transformed by transistor-based circuit to analog controls feeding the shutter glasses.

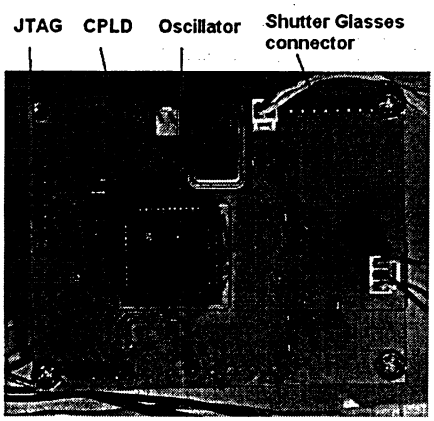


Figure 5-8. Photographic picture of the Shutter Glasses control sub-module.

The general layout of the experiment with real-time stereo vision is presented in Figure 5-9. The output devices are two standard CRT VGA monitors. The system was designed to operate in two configurations: using shutter glasses or using a pair of LCD projectors with polarization filters. This version of the setup needs a special non-depolarizing projection screen. The experiments were limited with CRT monitors as output devices option and shutter glasses as illustrated in Figure 5-10.

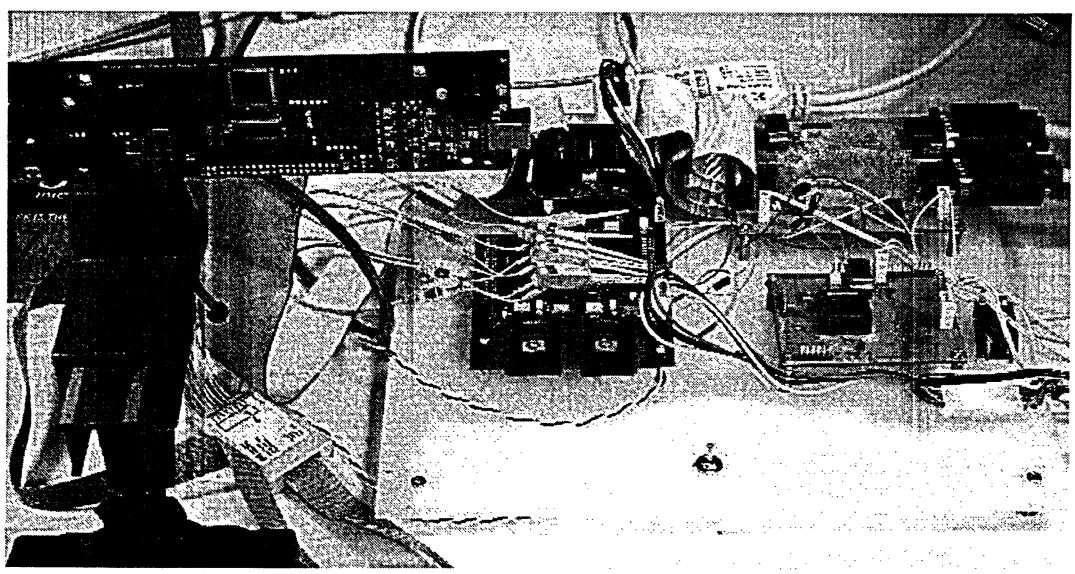


Figure 5-9. General layout of the stereo vision system: photographic image.[28]

[28] All pictures courtesy of ERSL.

5.2. Experimental results

The timing diagrams presented in the section 4.3.2 present the results of systems simulation. They are obtained by ChipScope Pro or Testbench tools.

Upon powering on the system and programming the VPM through Parallel Cable IV, two concurrent processes provide two real-time images on CRT monitors as shown in Figure 5-10.

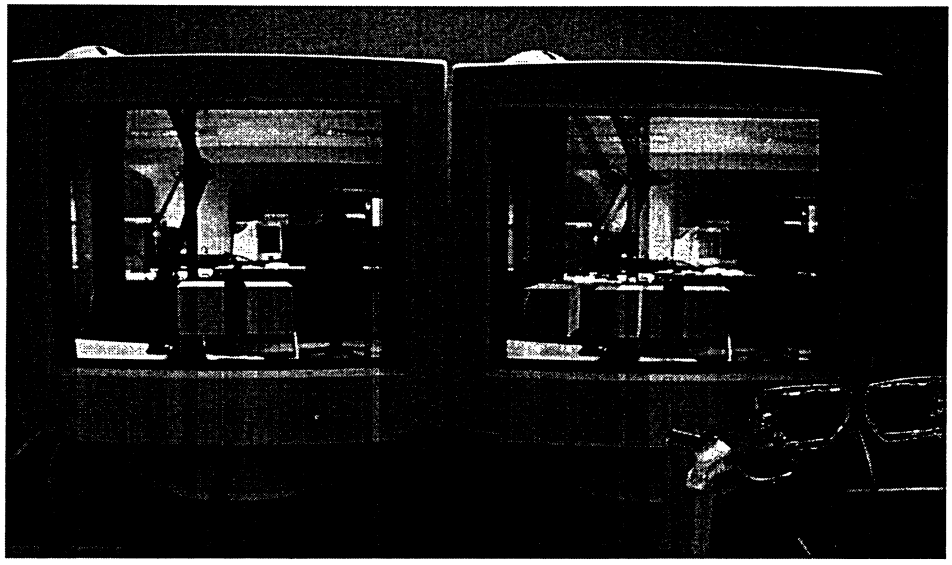


Figure 5-10. Real-time images visualization: both virtual buttons released.[29]

On the left CRT monitor the one-channel output is presented. It corresponds to the left eye view. In the lower part of the screen the images of two released cubic buttons are seen. The rasterizing data (coordinates of the line segments) are computed and stored into the internal Block RAM. Each side has its own texture attribute which is color for the present implementation of the system.

On the right side CRT monitor the generated video output is for real-time stereo visualization of the same background and images of the buttons with both eyes through the shutter glasses. We can see two shifted images of the background and the buttons if to look at the screen without shutter glasses. If to look at the screen through the shutter glasses, each eye perceives

[29] All pictures courtesy of ERSL.

only image form the corresponding channel, i.e. video camera. As a result, the real-time stereo video is perceived by a viewer.

Figure 5-11 illustrate the photographic picture of the same screens but for the case when on of the buttons is pressed. In this case the rasterizing data for the pressed button is different from the original image and the system generates two images of the pressed button which is the left one on the picture.

Figure 5-11. Real-time images visualization: one button pressed.[30]

A similar picture can be obtained if to press the right button.
The system provides proper colors assignment for each pixel as it is seen from the color pictures on the screens.
Another mode of operation of the system is to visualize the real-time edge detection image of the same scene. Figure 5-12 illustrates the second mode of operation. The system was tested with two edge detection algorithms: Robert Cross and Sobel. Robert Cross algorithm is a simpler approach which uses data only from two adjacent lines. Sobel algorithm, as it was shown in Chapter 3, is based on the data of three neighbouring scanning lines. It provides better results [59].

[30] All pictures courtesy of ERSL.

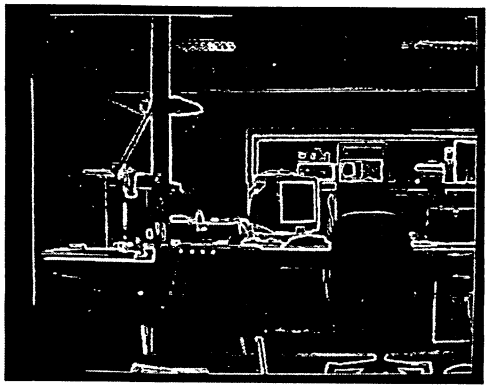


Figure 5-12. Real-time edge detection image visualization.

For stereo visualization the images from left and right cameras should be aligned as much as possible. The purpose of the experiments was to obtain a realistic stereo vision. The acceptable alignment was easily reached by mechanical adjustment of tilt of the lens holders for both image sensors.

5.3. Analysis of results

The presented system proves that high performance multi-channel multi mode systems for operating on data streams can be built on the FPGA-based SoC with hybrid architecture. The system based on Virtex-2 FPGA chip with 1 million system gates can easily process two independent video data streams as it is for stereo vision, it has also enough resources to process more data streams (e.g. 4 or 6) as it may be needed for panoramic vision. The number of streams and the complexity of the processing algorithms are limited only by the resources of the FPGA. The current version of the presented system utilizes only approximately 15% of FPGA slices, as it is seen from Table 5-1 which illustrates the design overview generated by the Xilinx' ISE.

Table 5-1. Design overview for the stereo vision system generated by ISE.

Design Overview for top

Property	Value
Project Name:	e:\users\sergei\xilinx_projects\vpm\vpm_xc2v1000_55_edge
Target Device:	xc2v1000
Report Generated:	Monday 08/28/06 at 14:10
Printable Summary (View as HTML)	top_summary.html

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops:	891	10,240	8%	
Number of 4 input LUTs:	725	10,240	7%	
Logic Distribution:				
Number of occupied Slices:	773	5,120	15%	
Number of Slices containing only related logic:	773	773	100%	
Number of Slices containing unrelated logic:	0	773	0%	
Total Number 4 input LUTs:	813	10,240	7%	
Number used as logic:	725			
Number used as a route-thru:	88			
Number of bonded IOBs:	131	172	76%	
Number of Block RAMs:	7	40	17%	
Number of GCLKs:	1	16	6%	

Performance Summary

Property	Value
Final Timing Score:	28821
Number of Unrouted Signals:	All signals are completely routed.
Number of Failing Constraints:	0

As it was estimated in the section 4.2 the time needed for all computations and rasterizing of one tetrahedron for a system based on the computational resources of one PicoBlaze microcontroller is approximately 0.5mS at the clock rate of 100MHz. It means that at a frame rate of 30fps the system can process at least 60 tetrahedrons, or 30 pairs of synthesized images for stereo vision. For the system which incorporates FPGA's internal resources for multiplication and addition, this computational time is approximately 0.1mS. It gives at least five times greater performance, i.e. 150 pairs of tetrahedrons can be processed. At the operational frequency of 400MHz we can get 600 pairs of tetrahedrons. It's also possible to operate on more data in parallel since only 15% of the chip resources are used in the presented design. It means that estimated value of processed tetrahedrons reaches approximately 4,000 tetrahedrons, or in terms of elementary triangles it gives 16,000 triangles per frame.

Using the same straightforward approach for estimation of the number of triangles which can be processed per frame by the Virtex-5 FPGA XC5VLX330 gives the value of at least 480,000. The presented estimation doesn't take into configuration deep pipe-lined architecture

which will improve the performance by several orders. So, the value of 1 million of elementary triangles processed per frame can be reached.

Besides, the system can perform a number of parallel tasks and can operate in different modes: one mode is real-time stereo or panoramic image visualization, the other mode is edge detection visualization.

The system utilizes only 15% of the resources of the Virtex-2 XC2V1000 FPGA chip. The data which is processed by the system and which is stored in the memory can be used for multiple other tasks for which some resources of the system can be dedicated, for example for disparity map computation, object recognition, image compression and other tasks.

Table 5-2. Modern FPGA resources: overview.

XILINX

	Virtex-2	
	XC2V1000	XC2V8000
System gates	1M	8M
Slices	5,120	46,592
Block Ram, 18Kbit each	40	168
Total Block RAM, Kbit	720	3,024
Multipliers	40	168
DCM	8	12
Internal Clock, MHz	420	420

	Virtex-2Pro	
	XC2VP50	XC2VP100
Logic cells	53,136	99,216
Block RAM, 18 Kbit each	232	444
Total Block RAM, Kbit	7,992	4,320
Multipliers	232	444
DCM	8	12
Internal Clock	400	400

	Virtex-4	
	XCE4LX1000	XCE4LX200
Logic cells	110,592	200,448
Block Ram, 18Kbit each	240	336
Total Block RAM, Kbit	4,320	6,048
Xtreme DSP slices	96	96
DCM	12	12
Internal Clock, MHz	500	500

	Virtex-5	
	XCE5VLX220	XC5VLX330
Logic cells	221,184	331,776
Block Ram, 18Kbit each	384	576
Block Ram, 36Kbit each	192	288
Total Block RAM, Kbit	6,912	10,368
DSP48 slices	128	192
CMT	6	6
Internal Clock, MHz	550	550

ALTERA

	Stratix-GX
	EP1SGX40D(G)
LE (logic elements)	41,250
RAM Blocks:	
M512 (32x18bits)	384
M4K (128x36bits)	183
M-RAM Blocks (4Kx144bits)	4
Multipliers	112
DSP Blocks	14

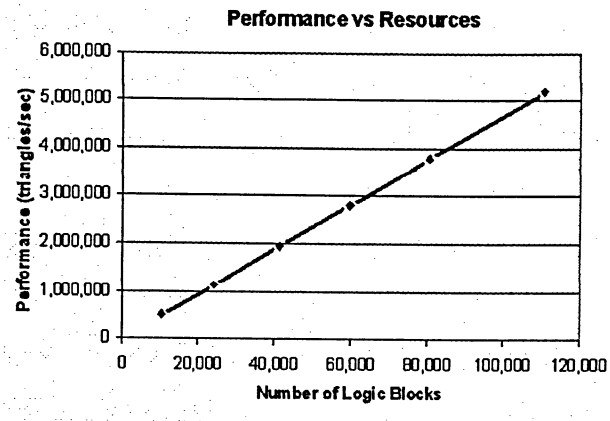
Modern FPGAs have significantly more logic and computational resources than the Virtex-2 XC2V1000 chip used in the presented design.

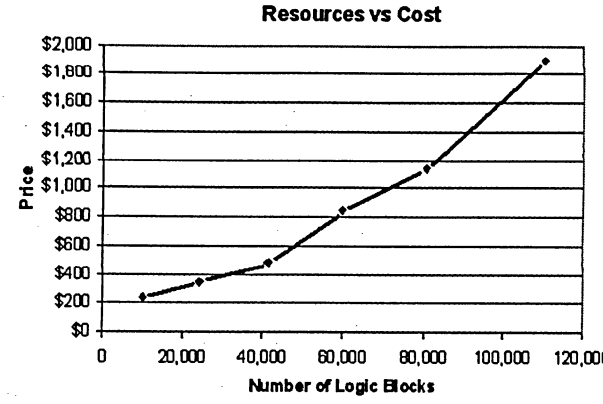
Table 5-2 shows the resources overview of some FPGAs which can be used for design of a specific multi-stream multi-mode video processing system. The table presents latest FPGAs from Xilinx Inc. and Altera Corp. [67]. Other vendors of FPGAs are Actel Corp., Atmel Corp., Lattice Semiconductor Corp., and QuickLogic Corp. [68]. FPGA from different vendors have different architecture of logic blocks, embedded Ram blocks, embedded computational units, different DSP modules, different embedded etc. Some other differences are the following: Actel and Lattice Semiconductors offer FPGA chips with non volatile configuration memory, QuickLogic offers chips with the lowest power consumption while Xilinx and Atmel provide partial reconfigurability feature. Partial reconfigurable architecture allows changing portions of the system's architecture on the fly, thus it becomes possible to design a complex system on a smaller chip.

The presented system doesn't employ partial reconfigurability feature since it utilizes only 15% of logic resources. Besides, the platform must be specially designed to be partially reconfigurable. This is what should be done for the future development of the project.

The diagrams presented in Figure 5-13 illustrate the Performance versus Cost analysis for the given task, i.e. for the real-time stereo vision system with synthesis of interactive 3-D virtual controls.

Straightforward approximation of the performance for the existing Virtex-4 FPGA chips and Virtex-2 chip on which the system is implemented is illustrated by the first plot. Actual dependence is not so straight because of, for example, limitations of routing resources of FPGA chips for dense designs. But for this analysis, even simple straightforward approximation produces important final plot, Performance versus Cost diagram. It shows that there is an optimal FPGA chip size for the implemented system (for the performance/cost criteria). Furthermore, it emphasizes effectiveness of RTR FPGA-based platform which allows to design and implement a complex system on a smaller chip than in case of ASIC design approach when the complete set of IP cores for the whole set of tasks sits in a bigger chip.





	Price	Logic Blocks
XC2V1000	$240	10,240
XC4VLX25	$347	24,192
XC4VLX40	$480	41,472
XC4VLX60	$915	59,904
XC4VLX80	$1,138	80,640
XC4VLX100	$1,839	110,592

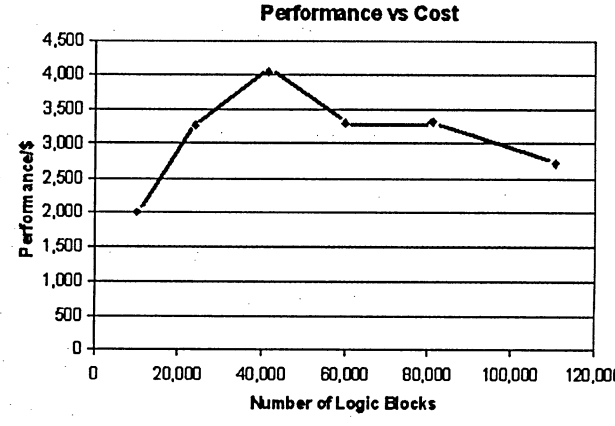


Figure 5-13. Performance vs. Cost analysis.

6. CONCLUSIONS AND FUTURE WORK

The major purpose of this work was to research and develop the methodology for optimal implementation of a real-time stereo-vision system with interactive run-time virtual 3-D objects synthesis on the basis of a RTR FPGA with hybrid architecture and practically implement and verify the developed methodology. The following tasks were distinguished: developing the architecture of a multi-mode system-on-chip for real-time stereo visualization, research and developing a technique for simple 3-D virtual objects synthesis, developing and implementation of the application specific stereo image capture system which provides output data of the standard VGA resolution, implementation of the real-time stereo vision system in hardware, and verification of implemented algorithms for 3-D virtual object synthesis, edge detection, and color decoding.

The research of existing real-time hardware-based video systems is done. The existing systems mostly perform a particular task or a set of tasks for easier interfacing of capturing devices and the main processing device which is typically a GPP, workstation or DSP. I.e., these systems are mostly an intermediate device between image capture device and the host processor.

The analysis of formal representation of 3-D bodies in three-dimensional space and in two-dimensional projection plane is performed. The methodology for computing necessary for object presentation is done and the presented approaches can be used both for hardware and sequential microprocessor based system architectures.

The methodology for synthesis of system architecture is considered. Based on this methodology the architecture of all components of the system is developed.

The fully functional prototype of the real-time stereo vision system based on the developed architecture is designed and implemented. The system is built on the base of the Run-Time Reconfigurable platform which utilizes the Xilinx Virtex-2 FPGA. All data processing is performed by the single FPGA chip, i.e. this is the implementation of a SoC.

The PCBs for the capture and visualization subsystems are designed, manufactured and debugged.

The functionality of the IP cores and the whole system is simulated and verified. The system generates two independent output video channels which can present different real-time images in different modes (one eye view, interlacing frames for stereo visualization using shutter glasses, and edge detection image). The 3-D stereo image of two interactive controls presented as cubic buttons changing their state from "released" to "pressed" is generated.

The following limitations of the presented system can be distinguished:

i) it presents the synthesized images only of simple interactive controls; the state of controls is simulated by pressing the actual buttons;

ii) for storing the data of virtual objects it uses only internal Blocks RAM which might be not big enough to keep the data of more complex objects;

iii) the number of incoming data streams is limited by two – it's enough for a stereo vision system, but not enough for a panoramic vision.

These limitations reflect only the fact that the system is just an illustration of the proposed approach. The future work on the project can be directed to increasing of the number of 3-D virtual controls and other more complex virtual objects. The algorithms for objects surface shading and texturing can be also implemented. Another direction for the future work is to implement moving objects which can shade one another.

Infra-red sensors for hands tracking for manipulating virtual controls can be added to the system, thus the effect of immersing into a virtual reality with real scenes and virtual controls can be reached. The number of incoming video data streams can be more than two. The current number is limited by 2 only because of the limited number of pins of the corresponding connectors. A new reconfigurable platform designed specially for the tasks of panoramic vision system would resolve all these limitations.

The system is easily upgradable and reconfigurable. It can be also built as dynamically reconfigurable system and as partially reconfigurable system when a library of computational specific soft IP cores can be stored in on-board non-volatile memory and be loaded into the FPGA chip when necessary. This approach allows using smaller chips for very complex tasks.

So, the future work on the project of the present thesis should be focused on the increasing of the synthesized 3-D virtual objects complexity. Embedded microprocessor soft cores other than PicoBlaze (e.g. MicroBlaze) and hard wired PowerPC (for Xilinx FPGAs) should be considered for implementation in the system for increasing the performance of the 3-D virtual

objects synthesis part of the design. Dual processor or n-processor architecture is another way to increase the performance of the system. Another aspect to improve the quality of the system is to utilize image sensors with higher resolution (the current resolution is 640 x 480 pixels).

Finally, a panoramic vision system which operates on four or six input video data streams would be a natural evolution of the system.

7. BIBLIOGRAPHY

1. http://www.ati.com
2. *The New Encyclopedia Britannica*, 15th edition, vol.24, 2003.
3. T. Okoshi, *Three-dimensional imaging techniques*, Academic Press Inc., 1972.
4. R.M. Hayes, *3D-movies: a history and filmography of stereoscopic cinema*, McFarland & Company, Inc., Publishers, 1989.
5. *Television History - The First 75 Years*, web: http://www.tvhistory.tv
6. Bruce R. Dewey, *Computer Graphics for Engineers*, Harper and Row, Publishers, 1988.
7. Edward Angel, *Interactive Computer Graphics: A Top-Down Approach using OpenGL®*, Pearson Education, Inc., 2006.
8. A.Sartori, "A smart camera", In W.L.Will Moore, editor, *FPGAs*, Abingdon EE and CS Books, Abingdon, England, chapter 6.6, pp. 353-362, 1991.
9. S.Scalera, M.Falco, and B.Nelson, "A reconfigurable computing architecture for microsensors", *In FCCM 2000 (Field-Programmable Custom Computing Machines) Preliminary Proceedings*, Napa, CA, April 2000, pp. 59-67.
10. G.Lienhart, R.Lay, and K.H.Noffz, R.Manner, "An FPGA-based video compressor for h.263 compatible bit streams", *In Proceedings of the International Conference on Consumer electronics, 2000, ICCE,*,13-15 June 2000, pp. 320-321.
11. John Woodfill and Brian Von Herzen, "Real-Time Stereo Vision on the PARTS Reconfigurable Computer", *In Proceedings of the 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 201-210, 1997.
12. Miriam Leeser, Shawn Miller and Haiqian Yu, "Smart Camera Based on Reconfigurable Hardware Enables Diverse Real-time Applications", *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)*, pp. 147-155, 2004.
13. Giancarlo Genta, Marcello Chiaberge, Nicola Amati, Mauro Padovani, Claudio Sansoe, and Paolo Rolando, "A simple embedded stereoscopic vision system for an autonomous rover", *In Proceedings of the 8th ESA Workshop on Advanced Space Technologies for Robotics and Automation 'ASTRA 2004' ESTEC*, Noordwijk, The Netherlands, November 2-4, 2004, pp. 1-5.
14. Enrico Grosso and Massimo Tistarelli, "Active/Dynamic Stereo Vision", *In IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.17, No. 9, September 1995, pp. 868-879.
15. Heiko Hirschmüller, "Improvements in Real-Time Correlation-Based Stereo Vision", *In Proceedings of the IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV'01)*, 9-10 December 2001, pp. 141-148.
16. Michael Kuhn, Stephan Moser, Oliver Isler, Frank K.Gürkaynak, Andreas Burg, Norbert Felber, Hubert Kaeslin, and Wolfgang Fichtner, "Efficient ASIC

Implementation of a Real-Time Depth Mapping Stereo Vision System", *In Proceedings of the 46th IEEE International Midwest Symposium on Circuits and Systems, MWSCAS '03*, Volume 3, 27-30 December, pp.1478 – 1481, 2003.

17. K.Miura, M.Hariyama, and M.Kameyama, "Stereo Vision VLSI Processor Based on a Recursive Computation Algorithm", *SICE 2003 Annual Conference*, Volume 2, 4-6 Aug. 2003, pp.1564-1567.
18. Kazuhiro Yoshida and Shiego Hirose, "Real-Time Stereo Vision with Multiple Arrayed Camera", *In Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, France, May 1992, pp. 1765-1770.
19. Boon Kiat Quek, Javier Ibañez-Guzmán, and Khiang Wee Lim, "Feature Detection for Stereo-Vision-Based Unmanned Navigation", *In Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems*, 1-3 December 2004, pp. 141-146.
20. Nicolas Ayache and Francis Lustman, "Trinocular Vision for Robotics", *In IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.13, No.1, January 1991, pp. 73-85.
21. Damien Maupu, Mark H. Van Horn, Susan Weeks, and Elizabeth Bullit, "3D Stereo Interactive Medical Visualization", *In IEEE Computer Graphics and Applications*, September/October 2005, pp. 67-71.
22. Kofi Appiah and Andrew Hunter, "A Single-Chip FPGA Implementation of Real-time Adaptive Background Model", *In Proceedings of IEEE International Conference on Field-Programmable Technology*, pp. 95-102, 2005.
23. Don Murray and James J.Little, "Environment modeling with stereo vision", *In Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, September 28 – October 2, 2004, Sendai, Japan, vol.3, pp. 3116-3122, 2004.
24. John Iselin Woodfill, Gaile Gordon, and Ron Buck, "Tyzx DeepSea High speed Stereo Vision System", *In Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'04)*, pp. 41-41, 2004.
25. J.S. Chahl and M.V. Srinivasan, "A complete panoramic vision system, incorporating imaging, ranging, and three dimensional navigation", *In Proceedings of IEEE Workshop on Omnidirectional Vision*, June 2000, pp.104 – 111.
26. S. Derrien and K. Konolige, "Approximating a single viewpoint in panoramic imaging devices", *In Proceedings. IEEE Workshop on Omnidirectional Vision*, 12 June 2000, pp.85 – 90, 2000.
27. T.Nakao and A. Kashitani, "Panoramic camera using a mirror rotation mechanism and a fast image mosaicing", *In Proceedings of 2001 International Conference on Image Processing*, Volume 2, Oct. 2001 pp.1045 – 1048, 2001.
28. Hong Hua and Narendra Ahuja, "A High-Resolution Panoramic Camera", *In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001, CVPR 2001, Volume 1, pp.960-967, 2001.

29. M. Barth and C. Barrows, "A fast panoramic imaging system and intelligent imaging technique for mobile robots", *In Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems '96, IROS 96,* Volume 2, 4-8 Nov. 1996, pp.626 – 633.

30. Wai-Kwan Tang, Tien-Tsin Wong, and Pheng-Ann Heng, "A System for Real-Time Panorama Generation and Display in Tele-Immersive Applications", *In Proceedings of IEEE Transactions on Multimedia*, Vol.7, No.2, April 2005, p.p.280-292.

31. Stavros Tzavidas and Aggelos K. Katsaggelos, "A multicamera setup for generating stereo panoramic video", *IEEE Transactions on Multimedia*, Volume 7, Issue 5, Oct. 2005, pp.880 – 890.

32. Shmuel Peleg, Yael Pritch, and Moshe Ben-Ezra, "Cameras for stereo panoramic imaging", *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2000, Volume 1, 13-15 June 2000, pp.208 – 214.

33. Sing Bing Kang and Richard Weiss, "Characterization of errors in compositing panoramic images", *In Proceedings of 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 17-19 June 1997 pp.103 – 109, 1997.

34. Gui Yun Tian, Duke Gledhill, and Dave Taylor, "Color correction for panoramic imaging", *In Proceedings of Sixth International Conference on Information Visualization*, July 2002, pp.483 – 488, 2002.

35. A. Simon, R.C. Smith, and R.R. Pawlicki, "Omnistereo for panoramic virtual environment display systems", *In Proceedings of IEEE on Virtual Reality*, 27-31 March 2004, pp.67-279, 2004.

36. Steve Bryson, "Direct Manipulation in Virtual Reality", In *The Visualization Handbook*, Editors Charles D. Hansen and Chris R. Johnson, Elsevier Inc., 2005.

37. Yongmin Zhong, Wolfgang Mueller-Wittig, and Ma Weiyin, "A model representation for solid modeling in a virtual reality environment", *In Proceedings of Shape Modeling International*, 17-22 May 2002, pp.183-190, 2002.

38. Linda M. Stone, Thomas Erickson, Benjamin B. Bederson, Peter Rothman, and Raymond Muzzy, "Visualizing data: is virtual reality the key?", *In Proceedings of IEEE Conference on Visualization, Visualization '94*, 17-21 Oct. 1994, pp.410-413.

39. Carl Machover and Steve E. Tice. Virtual reality, "Computer Graphics and Applications", *In IEEE*, Volume 14, Issue 1, Jan. 1994, pp.15-16.

40. Özdogan Karaçali, "Towards user oriented object modeling in virtual reality", *In Proceedings of IEEE Southeastcon '95.* "Visualize the Future", 26-29 March 1995, pp.454-460.

41. J.M. Zheng, K.W. Chan, and I. Gibson, "Virtual reality. Potentials", In *IEEE*, Volume 17, Issue 2, Apr-May 1998, pp.20-23.

42. *Soft core microcontrollers PicoBlaze and MicroBlaze*, web: http://www.xilinx.com.

43. Lev Kirischian, "The method of selection of architecture configuration for executable tasks", *In International Conference PACT'98 Workshop on Reconfigurable Computing*, Paris, France, 1998, pp.125-129.

44. Lev Kirischian, "Optimization of Parallel Task Execution on the Adaptive Reconfigurable Group Organized Computing System", *In Proceedings of PARELEC 2000 International Conference on Parallel Computing in Electrical Engineering*, 2000, pp.100-105.

45. Lev Kirischian, Irina Terterian, Pil Woo Chun, and Vadim Geurkov, "Re-Configurable Parallel Stream Processor with Self-Assembling and Self-Restorable Micro-architecture", *In Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC'04)*, 7-10 September, 2004, pp. 165-170.

46. Valeri Kirischian, Sergiy Zhelnakov, Peter Chun, Lev Kirischian, Vadim Geurkov, "Uniform Reconfigurable Processing Module for Design and Manufacturing Integration", *Proceedings of Advanced Manufacturing Technologies 2005*, London, Canada, pp. 77-82, May 2005.

47. Peter Chun, Valeri Kirischian, Sergiy Zhelnakov, and Lev Kirischian, "Reconfigurable Multiprocessor with Self-optimizing Self-assembling and Self-restoring Micro-architecture", *Presentation on Workshop on Architecture Research using FPGA Platforms*, pp. 52, San Francisco, February, 2005.

48. Pil Woo (Peter) Chun, *Dynamically reconfigurable parallel stream processor*, Thesis (M.A.Sc.), Ryerson University, 2004.

49. Valeri Kirischian, *FPGA based computing platform with temporal partitioning mechanism*, Thesis (M.A.Sc.), Ryerson University, 2005.

50. Irina Terterian, *Self-restoration mechanism for run-time reconfigurable data-stream processors*, Thesis (M.A.Sc.), Ryerson University, 2004.

51. Richard Hartley and Andrew Zisserman, *Multiple View Geometry in computer vision*, Second Edition, Cambridge University Press, 2003.

52. James D. Foley and Andries van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, 1982.

53. Rafael C. Gonzales and Paul Wintz, *Digital Image Processing*, Addison-Wesley Publishing Company, Inc., Second Edition, 1987.

54. Sergei Savchenko, *3D Graphics Programming: Games and Beyond*, Sams Publishing, 2000.

55. David F. Rogers, *Procedural elements for computer graphics*, McGraw-Hill Inc., Second Edition, 1998.

56. Rajesh K. Gupta, and Giovanni De Micheli, "Hardware-Software Cosynthesis for Digital Systems", *In Readings in Hardware/Software Co-Design*, Morgan Kaufmann Publishers, 2002, pp.29-41.

57. Jørgen Staunstrup, "Design Specification and Verification", In Jørgen Staunstrup and Wayne Wolf, *Hardware/Software Co-Design, Principles and Practice*, Kluwer Academic Publishers, 1997.

58. Giovanni De Micheli, *Synthesis and optimization of digital circuits*, McGraw-Hill, Inc., 1994.

59. Wayne Wolf, *Computers as Components: Principles of Embedded Computing System Design*, Morgan Kaufmann Publishers, 2001.

60. Rafael C. Gonzalez and Richard E.Woods, *Digital Image Processing*, Prentice Hall, Second Edition, 2002.

61. John L. Hennessy and David A. Patterson, *Computer Organization and Design: The Hardware/Software Interface*, Second Edition, Morgan Kaufman Publishers, Inc., San Francisco, California, 1998.

62. John L. Hennessy and David A. Patterson, *Computer Organization and Design: The Quantitative Approach*, Morgan Kaufman Publishers, Inc., San Francisco, California, 2003.

63. Giovanni de Micheli, R. Ernst and Wayne Wolf, *Readings in Hardware/Software Co-Design*, Morgan Kaufmann Publishers, 2002.

64. Sergiy Zhelnakov, Valeri Kirischian, Peter Chun, Lev Kirischian, and Vadim Geurkov, "FPGA-based Computing Platform for Stereo-Panoramic Vision", *Presentation at Space Vision and Advanced Robotics Workshop, MDA Space Missions, SVAR 2005*, May 19, 2005.

65. Stephen Brown, Zvonko Vranesic, *Fundamentals of Digital Logic with VHDL design*, McGraw Hill, 2000.

66. Web: http://www.mediatronix.com/pBlazeIDE.htm

67. Web: http://www.altera.com

68. Clive Maxfield, *The Design Warrior's Guide to FPGAs*, Elsevier, 2004.