CSEE4840: Embedded Systems Design
Project Report
Stereo Depth Extraction (SDE)

Department of Electrical Engineering
School of Engineering and Applied Science
Columbia University

Team Members:
Ang Cui (ac2024)
Jeng-Ming Hwang (jh2026)
Yen Yen Ooi (yo2006)
Kashif Siddiqui (kms2010)
Ting-Hsiang Wu (tsw2008)

TABLE OF CONTENTS

Project Proposal

Design Document

PROJECT PROPOSAL

Introduction

Overview
Our project will utilize an ordinary video camera to extract 3D coordinates from a single light source in a dark background. Using the formulas of stereo-depth extraction (shown below) and a setup of mirrors augmented to the video camera, we will track a light source, and output a stream of 3D coordinates through MINICOM. We will process the video signal with the onboard digital video encoder and proceed to extract two sets of 2D coordinates in each of the sections of video buffer. By using an optimized depth extraction formula, we will compute a 3D coordinate of the light source. The collected data can then be exported to our own visualization application.

Possible Applications
We can make use of the stereo-depth extractor to construct an alarm system that can detector an object that is within a certain radius from our device. This can be used in security systems or even in defense departments. Our version is a prototype; a fully functional version would have an extended range thus enabling a 3D alarm system.

Experiment Setup
Many experiments have been conducted to prove that mirrors can be used to capture stereo images with a single camera. This act is called catadioptric stereo. A catadioptric system is an optical system that has a series of mirrors that compensate for the use of a single camera instead of multiple ones. Using a single camera is very important as it drastically cuts down the complexity of the experiment. If we were using multiple cameras we would have to worry about 2 NTSC inputs and the synchronization of video (if we had decided to go the pre-recorded route). Furthermore, if we had used 2 cameras we would have had to worry about creating a comparator that accepted dual real-time inputs and performed computations on both inputs then produced synchronized outputs. The disadvantage to using a single camera is that you must rectify the image. We have decided to rectify the image in the C code. This rectification is manifested in the corrective manipulation of the right camera X coordinate value that is corrected by subtracting a value of 320 from the value inputted by the NTSC signal.

Features Included
- Tracks two sources of light, one on the left side of the screen, one on the right.
- Tracks the depth of the light source in a range of 1 inch to 12 inches.
- Calculate as accurately as possible the positions of a moving light source from images taken from a digital camcorder

Proposed Setup and Algorithm
We will compare the depth calculated, D, to a lookup table consisting of experimental values of D for corresponding x1 and x2. We will calculate non-empirical values using first order approximation.

Design Issues

1. Setup of the video camera
Our ideal setup would require two cameras, but the video decoder can only take in image input from a single source at any point in time. Instead, we are using mirrors to obtain stereo images using a single video camera.

2. Understanding S-Video input
We will have to research into how the S-Video protocol is written as well as find out the resolution of the video input. We believe that manipulating this protocol will not be too difficult as we only intend to use a black and white input and will only be trying to determine light sources.

3. Extracting the position of the light source
We need to design a component that processes the video image and extract the pixel coordinates of the light source as captured by the video camera.

4. Find an algorithm that decodes the stereo-input coordinates
This algorithm must be simple and efficient - employing as few floating point operations as possible. An effort must be made to keep the algorithm as efficient as possible because it must be able to process somewhere in the vicinity of 30 frames per second. We are still trying to figure out if displaying the coordinates at a resolution rate that is less than 30 frames per second will allow us to render the coordinates sufficiently. A possible solution to minimize the number of floating point operations that need to be carried out is the use of a lookup table.

DESIGN DOCUMENT

Experiment Setup

In the original proposed stereo-vision setup, we used two sets of parallel mirrors to construct a rectified stereo system. However, this approach is proven to be impractical by the paper, "Rectified Catadioptric Stereo Sensor" by Columbia Professors Joshua Gluckman and Shree Nayar. After spending a tremendous amount of time and resources into developing our periscope apparatus according to our design specifications and conducting numerous experiments in a vain attempt to receive logical experimental results, we found an inherent design flaw. As Professor Gluckman/Nayar's paper points out, "When two sets of parallel mirrors are used a rectified stereo system can be constructed. However, this solution is not practical because the two virtual cameras do not share a common field of view." Unfortunately, we came accross this paper only after putting all the effort in to make our apparatus work. The diagram below illustrates the problem in greater depth.

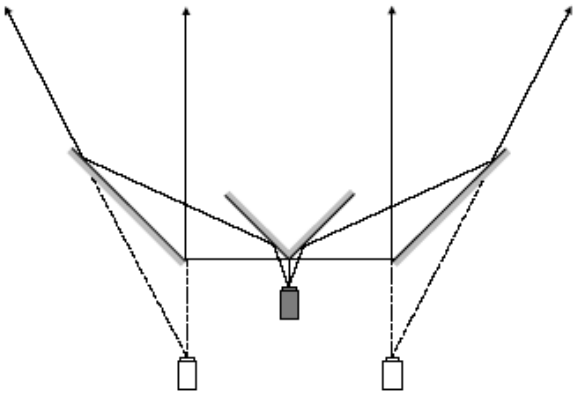


Figure 4: When two sets of parallel mirrors are used a rectified stereo system can be constructed. However, this solution is not practical because the two virtual cameras do not share a common field of view.

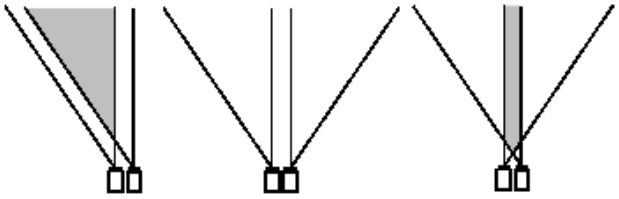


Figure 5: Once rectified, there are three possible configurations of the virtual cameras. However, only the one on the left leads to a practical solution. The middle configuration has no overlapping field of view and the configuration on the right only sees a narrow beam.

The next two diagrams represent another method that we tried but was not feasible given the constraints that we had:

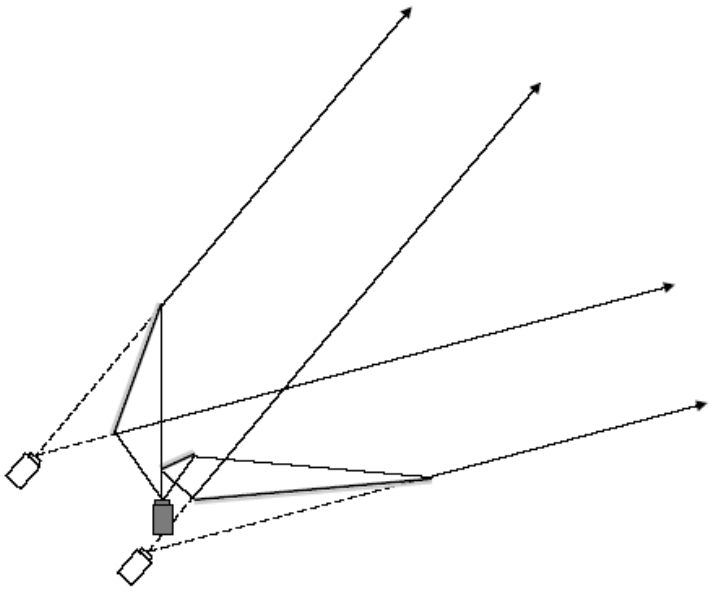


Figure 8: Three mirror rectified stereo. By using three mirrors a rectified stereo system can be designed where the field of view of each virtual camera is half the field of view of the real camera.

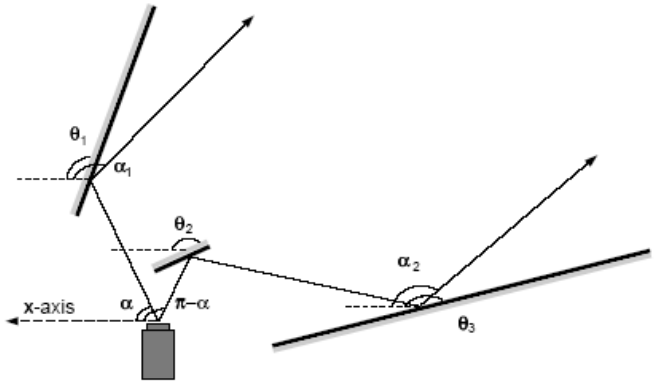


Figure 9: The mirrors must be angled such that two rays at angles α and $\pi - \alpha$ are parallel after being reflected by the mirrors, that is $\alpha_1 = \alpha_2$. Doing so will ensure that there is no rotation between the two virtual cameras.

As a result, we adopted the alternative setup proposed by this paper, the single mirror rectified stereo, to obtain rectified stereo images with a single mirror as shown in the diagram.

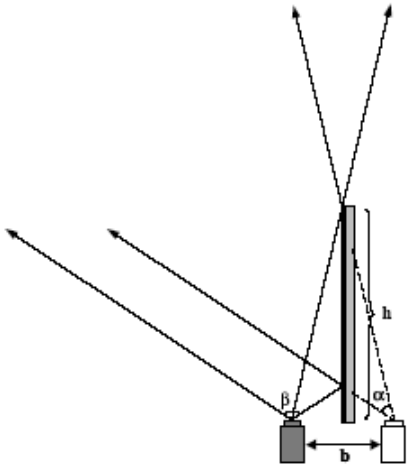


Figure 6: To obtain a rectified image with a single mirror, the normal of the mirror must be parallel to the scanlines of the camera. Note that the field of view of the right virtual camera will be limited by the finite size of the mirror.

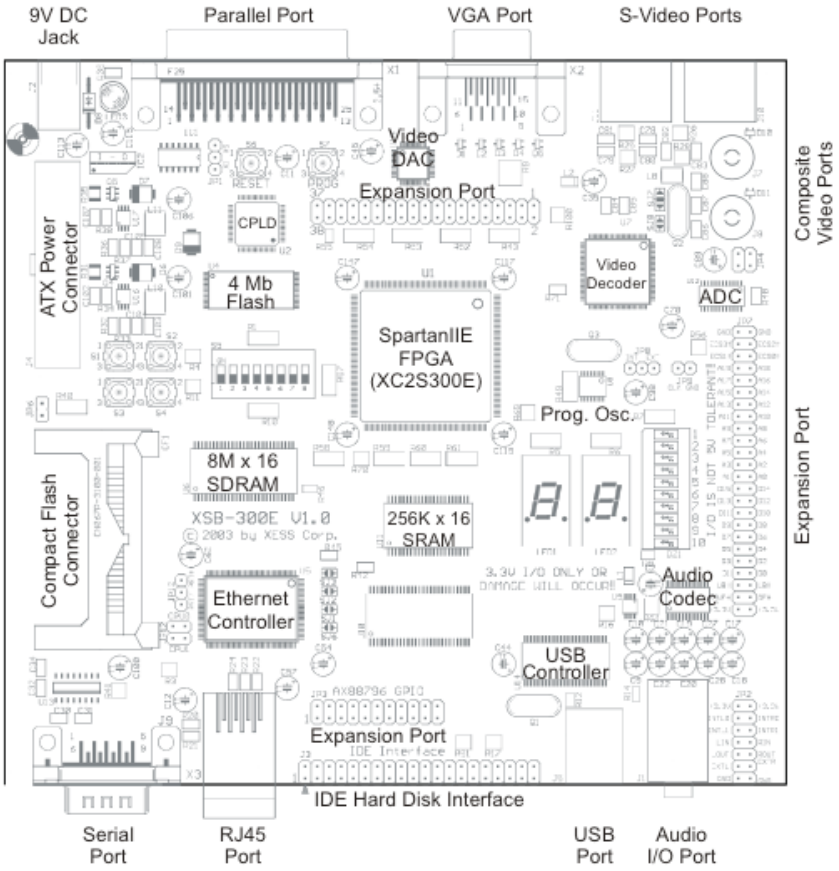
After implementing this setup we realized that due to the camera's overall size and the optics of the lens (ie field of vision is very small on the lens. Its very difficult to get a large enough baseline separation (the value of b in the diagram) using the single mirror setup.

We used a laser pointer as the light source and recorded a short clip of the stereo images while the light source moved to different depths away from the camera. We then input the video data to the Xilinx board for processing and extracting depths. During the recording, we used carpets to minimize the reflection from the floor so that the reflection does not get recorded as bright points.

Architecture Design

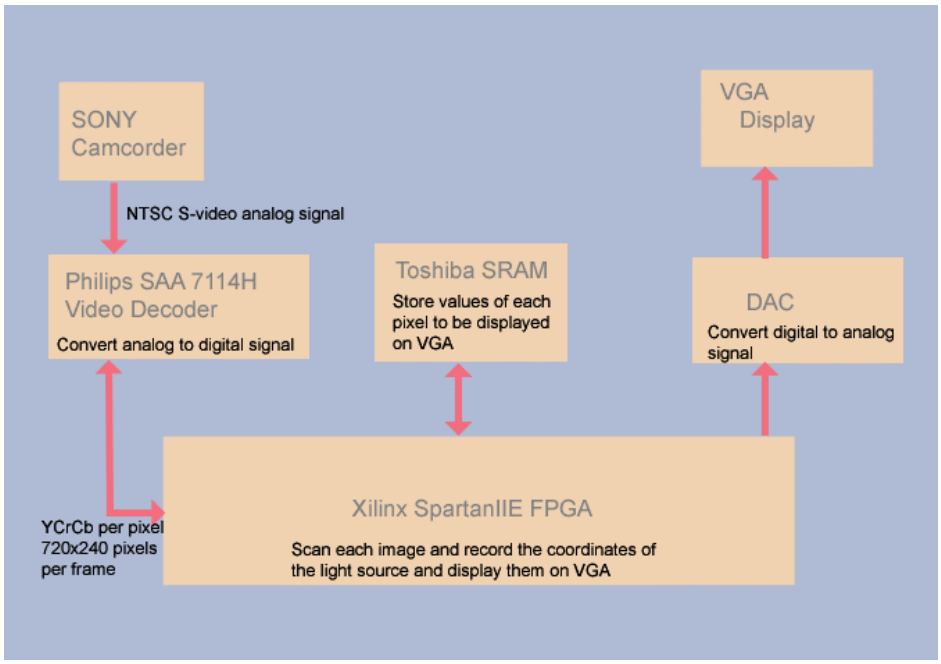
Required Equipment

- Sony Video Camera DCR-TRV103
- Philips SAA7114H video decoder
- Xilinx SpartanIIE FPGA
- Texas Instruments THS8133B video DAC
- Toshiba TC55V16256J/FT SRAM (256K x 16)
- Laser pointer



XSB Board

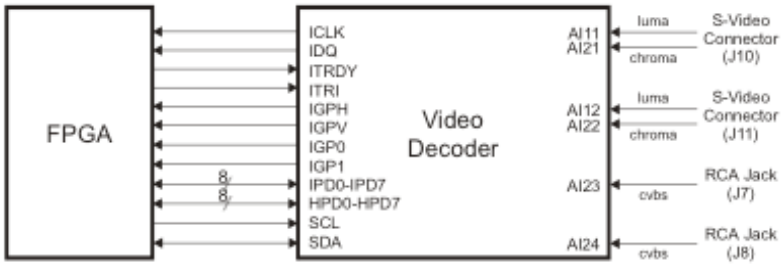
System Architecture



1. **Sony Video Camcorder DCR-TRV103**

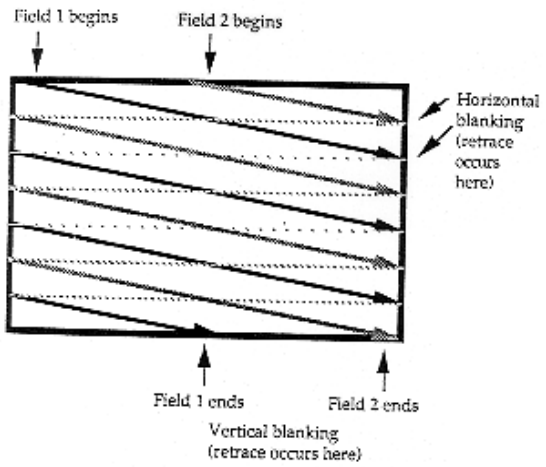
The Sony DCRTRV103 that outputs NTSC Super-video (S-video) signals that can be read by the on-board video decoder. The S-Video signal sends video data in two separate information signals - brightness (Luminance Y) and color (Chrominance Cr Cb).

2. Philips SAA7114H Video Decoder



This is the video decoder that is built into our XSB board. It supports and receives the S-video input from the camcorder. The input signal will be in NTSC at a resolution of (728 x 242). The video decoder will capture the video image and output the luminance and chrominance values for each pixel.

The output pixels also follow the NTSC interlaced scanning format, where the odd-numbered lines are first scanned in succession (odd field), followed by the even-numbered lines (even field).



3. Xilinx SpartanIIE FPGA

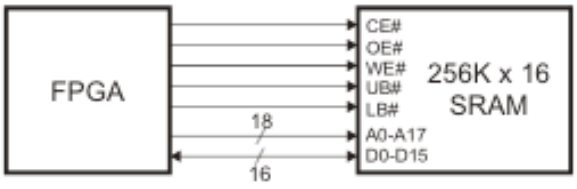
The SpartanIIE Field-Programmable Gate Array (FPGA) is the main entity of programmable logic on the XSB board. Its ability for programmability permits design upgrades in the field with no hardware replacement necessary.

Peripherals:
- opb_xsb300

This peripheral is taken from Lab 3. It acquires the video information from microblaze and stores it in the SRAM with the corresponding addresses. It then outputs the pixel values stored in the SRAM to the video DAC with the required timing signals.
- opb_videodec
- opb_i2ccontroller

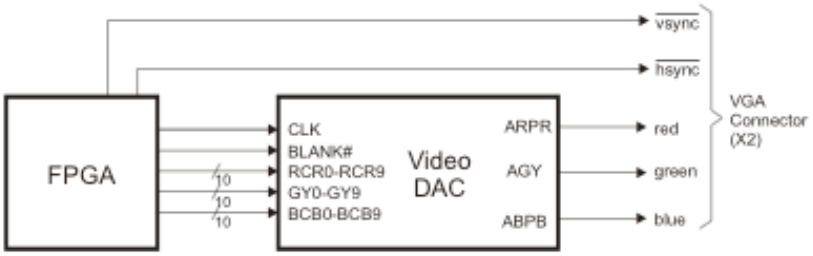
To save computation and processing time, we will discard one field (even field) as our image only deals with a bright spot in a black environment.

4. Toshiba TC55V16256J/FT SRAM (256K x 16)



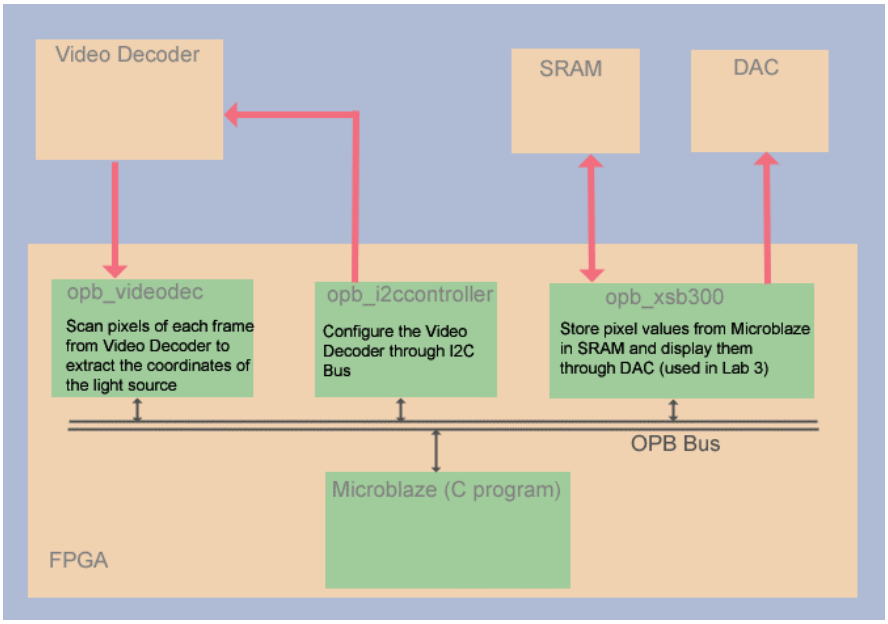
The TC55V16256J/FT is a static random access memory (SRAM) that stores the pixel data from the video decoder to be displayed on the monitor through the DAC.

5. Texas Instruments THS8133B Video DAC



The Video D/A Convertor (DAC) generates analog RGB signals for the VGA display.

Overview of program



From our C-program, we will configure the video decoder using the opb_i2ccontroller module. Using preset values, we will write the appropriate data to the corresponding registers for our video setting (NTSC).

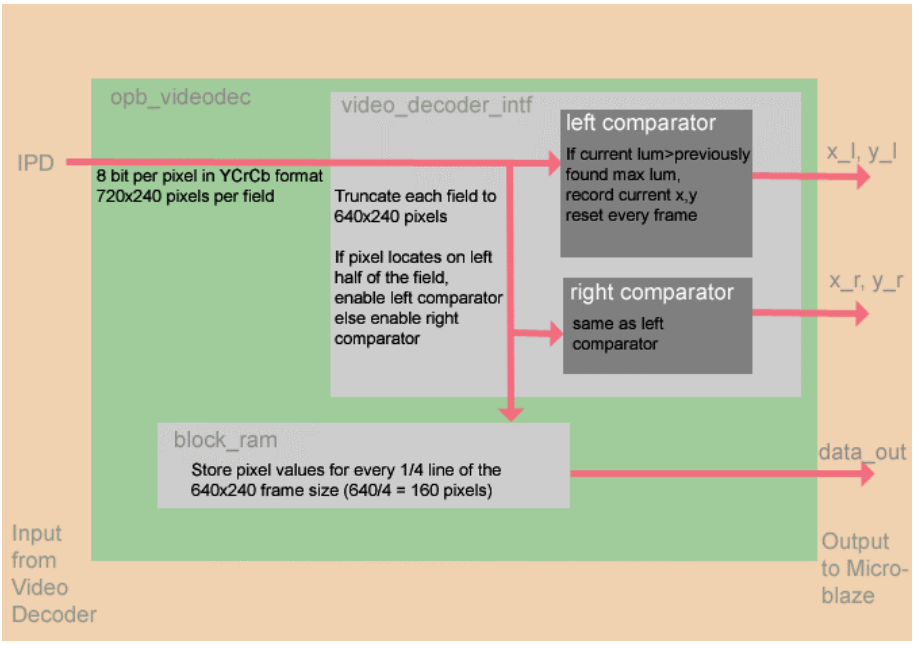
The opb_videodec module receives the video data from the video decoder and stores it in the block_ram on the FPGA.

We retrieve the video data and display it through the DAC onto the VGA monitor through the opb_xsb300 module developed in lab 3.

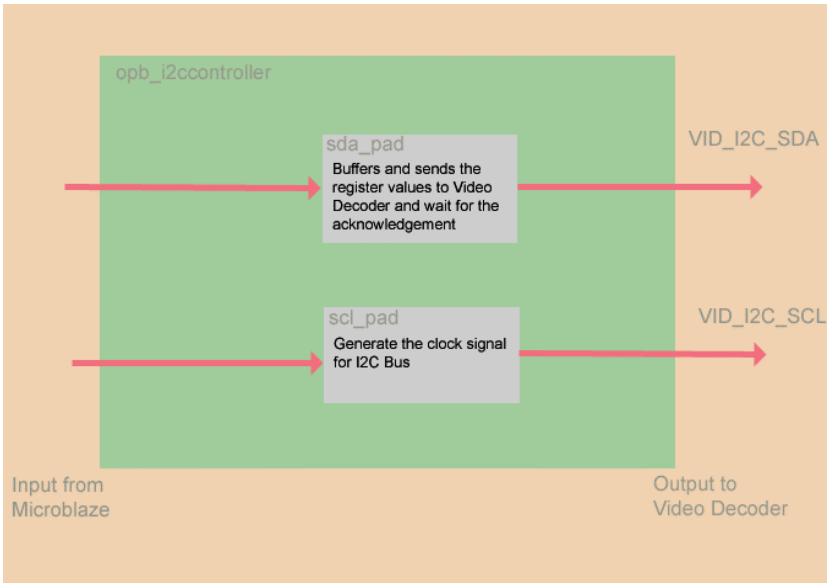
The opb_videodec also computes the 2D coordinates of the brightest pixel on the left and right half of each video frame respectively through our comparator component. The C-program will then retrieve these two coordinates and calculate the depth of the point source through empirical points and linear interpolation.

The results are displayed on the VGA monitor.

Component description



I2C bus



It serves to activate and configure the video decoder. Through the OPB Bus, we send the appropriate register values to the opb_i2ccontroller, which then coordinates the sending of this data into the SCL and SDA ports of the video decoder.

opb_videodec

This module captures the 8-bit luminance value of each pixel from the IPD port of the video decoder. This information is temporarily stored in the block_ram component of the FPGA before being retrieved and displayed by the C-program.

comparator
We have two comparator components that process the left and right side of the each video frame respectively. As NTSC adopts the interlaced scanning technique, we will only process the odd field and discard the even field. For each new frame, the comparator will scan each pixel and output the coordinates of the brightest pixel for that frame.

Software Architecture

(Pre-implementation)
Pseudo code for major components

Light Scanning

1) Search for light
Input: None
Output: 1 x 2D coordinate

```
for(w) {
   for{h) {
      if (has_light_around_point) {
          return coordinate
      }
   }
}
```

2) Track light
Input: 1 x 2D coordinate for last known light position.
Output: 1 x 2D coordinate for current light position.

x_old, y_old = 2D coordinate of known.

from x_old - scan_radius to x_old + scan_radius
from y_old - scan_radius to y_old + scan_radius

```
if (has light around current position)
     return current position
```

3) Save light coordinate
Input: 2 x 2D coordinate
Output: None

Write_Data(cord1, loc1)
Write_Data(cord2, loc2)

3D extraction
Input: 2 x 2D Coordinate
Output: 1 x 3D Coordinate

cord1 = Read_Data(loc1);
cord2 = Read_Data(loc2);

depth = k / tan(theta1) + tan(theta2);

return 3D coordinate

Problems encountered

1. Understanding the whole structure of the opb master and slave concepts

2. I2C bus
Marcio provided us with this module early on in the project

3. Figuring out how to write the opb module for the video decoder
We were making some progress though it was very time-consuming as we had to figure out the tentplate (eg ports, structure, address, communication protocol).

4. Core problem with hardware
Writing vhdl for the fpga and its peripherals was going pretty well, until we encountered timing issues and absence of neccessary buffers. We had alot of help from Marcio from overcoming these difficulties.

5. We realized that near to the end of the project that our initial approach had been disproved by Professor Nair. So we had to employ a new improvised method to get the accurate points. Noise is still affecting the stability of our system and consistency of our results.

Implemented Solution

After experimenting with our proposed setup, apparatus and architecture, we have revised the setup of our project.

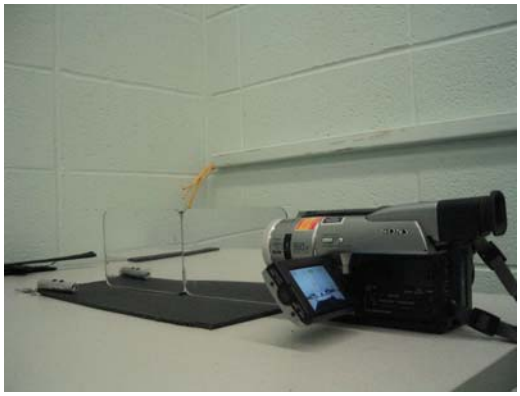
Revised Hardware:

Problems we did not foresee at the start include implementing the necessary buffers to link the video decoder output signals into our opb module. Furthermore, to verify that the co-ordinates we were tracking were correct, we had to borrow heavily from Marcio's opb module that could display video on the monitor.

Instead of trying to store the co-ordinates of the two brightest points, we keep feeding the co-ordinates into signal variables to be picked up through the opb bus whenever the C program calls for it.

Revised Apparatus:

After realizing our mirror setup's inability for stereo coverage, we switched to a Single Mirror Catadioptric Stereo setup proposed by Professor Nayar. This setup involves only long mirror, and provides for an easily calibratable setup. Also, this mirror setup provided us with two rectified view points. This solved a great problem for us, as rectifying images from our original mirror setup involved solving non-linear equations. After spending a great deal of time experimenting with an appropriate video footage of lightsource, which is trackable by our VHDL, we came up with the 10 second video which we used for testing. This video consists of a light source at various depths between 1 and 12 inches in a dark room.



Revised Software:

Instead of tracking the light source in software, as was originally proposed, we track the brightest point on each half of the screen in hardware. However, due to our recent change in the mirror setup, the light coordinate on the right side of the video input had to be flipped around the y-axis. This simple operation was done in software.

The software reads light coordinates from known memory address from our OPB device. Each time coordinates are read, the gui screen is updated with the recent coordinates, and the depth value is calculated by a linear approximation. This was done through experimental trial and error. First, numerous equations were tried for finding depth. We had equations involving trig functions, to third and second order approximations. In the end, after playing with our test footage, we found that the linear approximation gave most accurate results. This is probably due to the following reasons:

1. The depth range is relatively small. (1 to 12 inches)
2. Light source shows up as large dots on both sides fo the screen, taking up many pixels. Overall, the video footage proved to be too rough for tracking in high resolution. Stereo seperation values always ranged between 30 to 100.
3. Optics of the camcorder lens is unkown. To have an accurate function for depth extraction would require us to know the focal-length, apature, peripheral viewing range of the camera. Without these parameters of the camera, the most accurate, economical approximation proved to be the first order approximation.

Lessons Learnt & Advice

Ang: Printf's Suck.
Ting:
Kashif: Google is your best friend. Google everything before you start anything!!
Jeng-ming: Start early! VHDL and FPGA architecture is not easy.
Yen Yen: 1235 Mudd was our second home.

Files

C code
main.c : bulk of code, draws gui, reads input from opb device, calls extract_depth.h for depth calculation, configures the video decoder etc.
extract_depth.h: depth extraction and approximation functions. we experimented with a few, but in the end used the simplest function.

Acknowledgement

We would like to thank Marcio for allowing us to refer/use his VHDL code the I2C bus and the video decoder. We would like to thank Cristian and Josh for answering our queries when we had any. Lastly, we would like to thank Prof. Edwards for taking time off to appraise our project and his kindly advice on graduate school.