ECE532 Digital System Design

Title: Stereoscopic Depth Detection Using Two Cameras

Group #4
Prof: Chow, Paul
Student 1: Robert An
Student 2: Kai Chun Chou
Student 3: Mark Sikora
April 10th, 2015

Final Design Report

Table of Contents

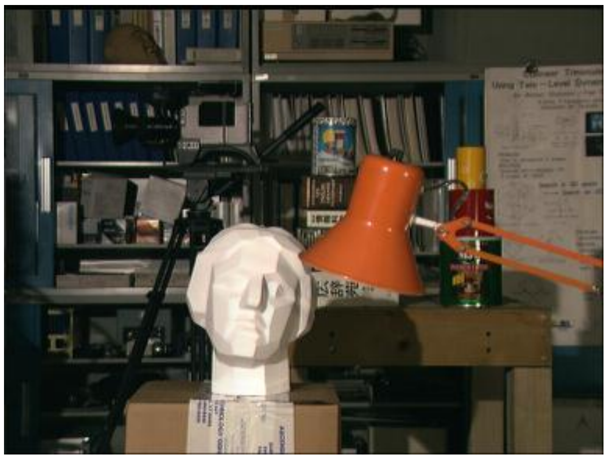
1. Overview

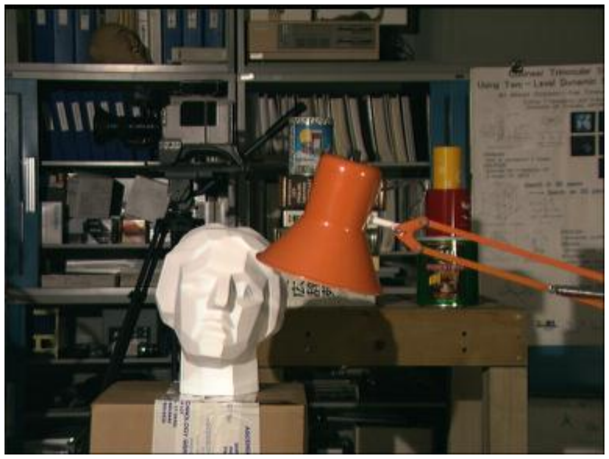
1.1. Motivation

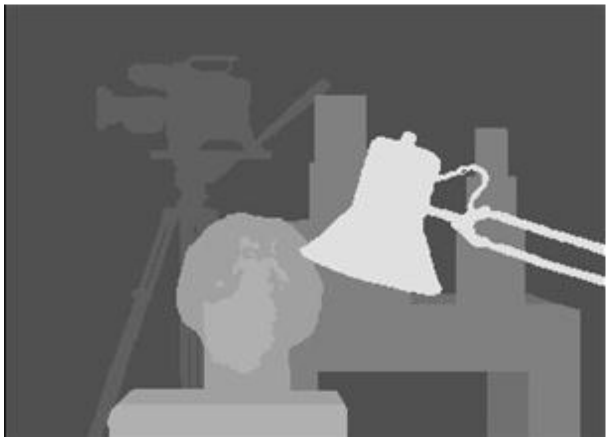
The main motivation for this project comes from the idea of detecting 3-dimensional depth from two 2-dimensional images, similar to how the human eyes work. This technique is known as stereoscopic depth detection and is traditionally implemented in software. However, due to the large size of most images, the computation of stereoscopic depth detection is prohibitively time expansive. We propose a digital system that can process two images faster than software allows by processing the rows of both images in parallel, thereby cutting down the computation time by a factor proportional to the number of rows.

1.2. Goal

Our goal is to develop a digital system capable of using two cameras to detect the distance of objects in the field of view and output the disparity map along with the two original images back through video.

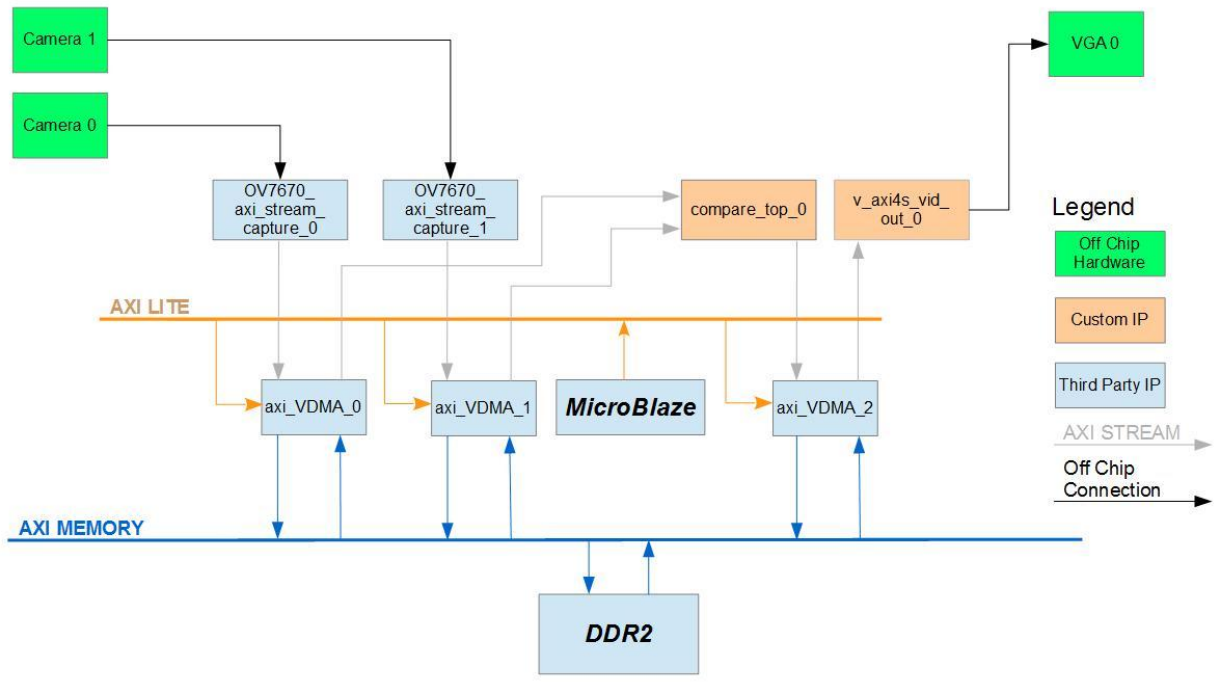






The two cameras would be secured horizontally to achieve manual image rectification thus ensuring the rows of both images are co-planar. The final system will operate in real time which requires the logic blocks to operate on a continuous stream of captured images.

1.3. Block Diagram



1.4. IP Description

The first IP of our system is the OV7670_axi_stream_capture block. This module translates the video input data from the Pmod cameras to AXI stream data for more efficient transferring within our system. The AXI stream data from the two OV7670_axi_stream_capture blocks are then fed into two VDMA blocks where the data is to be buffered in memory. Buffering is necessary to account for the difference in writing and reading speeds of the Pmod cameras and the VGA display. The video data in memory will then be read by the same VDMA blocks that wrote them and fed to the compare_top block, in total two images will be fed to the compare_top. The compare_top IP is our own custom hardware that computes the depth detection. The comparator will produce the third image and send all three images back to memory through a third VDMA. The three images will be fed to the v_axis4_vid_out block by the third VDMA and then from v_axis4_vid_out block to VGA display.

Within our system video data is transferred from OV7670_axi_stream_capture to VDMA to comparator_top to v_axis4_vid_out via AXI stream protocol. Images from VDMA are buffered to memory through the AXI memory bus protocol. Finally the three VDMAs are configured for the proper image width, height and data rate by the MicroBlaze processor through the AXI lite bus protocol.

2. Outcome

2.1 Results

All components of our project has been implemented. However we were not able to get the final design working due to problems integrating the comparator IP block with the AXI stream video interface. Our final result was that we were able to demonstrate a barebones version of our design where we passed the two input images to the VGA out with a third, comparison image simply being a duplicate of one of the input images. When we directly passed the input image to the output through AXI stream, the system worked as expected with the VGA displaying the proper outputs. However when we interrupted the AXI stream and tried to process the input and generate our own output the system would freeze up. We suspect that we are interrupting the system during a VDMA burst cycle and none of the VDMA components would work unless it receives the proper bit stream to end that burst cycle. However we were never able to verify if our theory was correct due to a lack of documentation from Xilinx on this issue and our online forum query was not responded to in time.

2.2 Future Work

The next step for our project is to get the comparator integrated into the the AXI stream interface. This would probably require asking someone at Xilinx why VDMA freezes when an AXI stream is interrupted. Another approach that we can take is to buffer the frames to memory and process them using software thus bypassing the issue of needing to interrupt the AXI stream. However this method would slow down our system significantly.

3. Project Schedule

We could not follow our original milestone plan for the project because we encountered unforeseen difficulties with Xilinx's VDMA module. For the majority of the project we had to parallelize our work with Mark trying to buffer images to memory via VDMA while Robert and Kai Chou tried to follow the original work plan by trying to implement the comparator block.

1. Milestone 1 (February 10)
    a. Original Milestone Planned
        i. Display static image over VGA
        ii. Display camera frames on screen
    b. Actual Milestone Achieved
        i. Succeeded Displaying image from camera to VGA via Bram
        ii. Failed to Display image from camera to VGA via DDR2 using Data Mover IP
        iii. Trying to Display image from camera to VGA via DDR2 using VDMA

2. Milestone 2 (February 24)
    a. Original Milestone Planned
        i. Hint blocks of camera frames with different colors
    b. Actual Milestone Achieved
        i. Trying to stream the video from camera to VGA without buffering by using an Video to AXI stream IP
        ii. Still having trouble buffering the video in DDR2
3. Milestone 3 (March 3)
    a. Original Milestone Planned
        i. Run comparator circuit for single block in static image
    b. Actual Milestone Achieved
        i. Succeeded in streaming the video without buffering via AXI stream
        ii. Trying to catch up to milestone by building the compare logic with Bram
        iii. Discovered that Bram does not have enough memory for our application
4. Milestone 4 (March 10)
    a. Original Milestone Planned
        i. Run comparator for all blocks in static image
        ii. Use color hinting for static image
    b. Actual Milestone Achieved
        i. Complete planning the video pipeline: Video In -> VDMA -> Test Pattern Generator -> Video Out -> VGA
        ii. Implemented FSM for compare logic that would cut between reading state and displaying state

5. Milestone 5 (March 17)
    a. Original Milestone Planned
        i. Run comparator circuit for single block in live image
    b. Actual Milestone Achieved
        i. Completed building the Video Buffering pipeline with VDMA
        ii. Implemented simple comparator logic in FSM that can average two input images and display the output along with the two input images.
6. Milestone 6 (March 24)
    a. Original Milestone Planned
        i. Run comparator circuit for all blocks in live image
        ii. Use color hinting for live image
    b. Actual Milestone Achieved
        i. Wrote C# Windows application to implements our image depth detection algorithm
        ii. Planned out new FSM that would work with the AXI stream video interface.
7. Milestone 7 (March 31)
    a. Actual Milestone Achieved
        i. Finished new FSM for AXI stream interface.
        ii. Finished Verilog version of comparator algorithm

4. Description of the Blocks

4.1 MicroBlaze (Xilinx)

MicroBlaze is a FPGA-based soft processor provided by Xilinx. It is used in our project to initialize the three VDMA blocks we have in our design. It also served as a design tool for us to debug our system.

4.2 AXI VDMA (Xilinx)

The AXI Video Direct Memory Access (AXI VDMA) is an IP provided by Xilinx. It provides high-bandwidth direct memory access between memory and AXI4-Stream type video target peripherals. We used two VDMA to read the two stream data from the two cameras. The comparator IP then access the data from the two VDMA, process the data, and generate a third video stream to the third VDMA. Finally, the third VDMA sends the final stream data to the VGA IP to display frames on the monitor.

4.3 AXI Light (Xilinx)

AXI Light is a micro controller bus for simple, low-throughput memory-mapped communication. The MicroBlaze uses this bus to communicate with VDMAs and set up the registers to initialize them.

4.4 i2c_controller_ip (Course)

The I2C controller is a simple IP wrapper around the example code given in the course for initializing the ov7670 cameras. The module communicates with the camera over an I2C bus and sends the commands required to initialize the camera.

4.5 OV7670_ip (Online)

The OV7670 IP was taken from a source online. The IP takes the pixels spread over two clock cycles and produces an output with a single pixel. The module also generates the valid, end of line, and start of frame signals that the AXI Stream protocol expects.

The original source online had a hard coded frame width, we modified this so that the frame width would be parameterized so we could set it to the correct resolution of our camera. The module also was using a different input format from the camera (RGB 5:6:5) whereas our camera was using a different format (RGB 4:4:4) which required a small modification.

Original version of the ip here:
http://lauri.xn--vsandi-pxa.com/hdl/zynq/xilinx-video-capture.html.

4.6 VGA_IP (Group)

The VGA_IP was created by our group. It reads the data from the AXI Stream to Video Out that is connected to the third VDMA and then has it correctly displayed to the VGA display.
The VideoOut module from Xilinx does not blank the data lines when the video is inactive. This causes either corruption of the output or a failure of the monitor to detect the VGA signal. The VideoOut module also works using active high clocks whereas the VGA specification requires active low clocks so these must be inverted.

4.7 comparator_ip (Group)

The comparator_ip was created by our group. It is the core module of the project. It reads the data from the first two VDMA, processes the data, and then sends data into the third VDMA.

The comparator is composed of multiple state machines performing small tasks which are coordinated by a top level state machine.

The top level state machine starts by reading in a block and a line. If one of these is at the start of a frame and the other isn't the out of sync stream is run until it falls back into sync. If both streams are in sync the comparator block is run using these inputs. After the comparator is finished a new block is read in and the comparator is run again. We run this process until we have either read enough blocks or a end of line is encountered. We read in a new line and repeat the block process over again. We do this until we've read in enough lines to have a full vertical block. The control switches to a video out module which uses the values produced by the comparator to output the correct lines. When the video out finishes we return to the beginning and repeat.

There are two state machines for reading in the input streams. One reads in data in full lines and the other reads in data in blocks. They use their ready output and the valid input to process the correct number of valid pixels. Both are controlled with a go signal which is asserted for a single clock cycle to start the block. After a block has completed it asserts a done output until another go signal arrives.

The comparator block uses the most recent read blocks and lines to compute the differences. We have a large n$\times$m array that contains the partial sums of the differences. The n dimension is the number of blocks in the first image and the m dimension is the number of comparisons to do in the second frame which is based on how many pixels we separate the blocks by. When reading in the first line of a new block the old values in the array are removed and the new values are written in, otherwise the new values are added to the values in the array to accumulate the difference as the lines are read in. The specific algorithm used for the comparison is a squared difference. We take the individual color components of each pixel and subtract them between the two images. Then we take these differences and square them and add them together. This gives us the difference between two pixels, To get the difference for a group of pixels we add all these differences together. The comparator block has a lot of steps to do so each comparison is broken down into multiple cycles. The first cycle is the difference and square step. The second is adding the differences for each pixel. And the last step is adding all the pixels for one line of a block. After all these steps are done the module asserts a done signal to the top block.

The video output block is composed of two parts. The first part is part that determines the distance and what colour to display. This block takes one of the rows produced by the comparator block and reduces it down to a single best block. This requires a large number of logic steps so on each clock cycle the number of blocks is cut in half. After a block is chosen an output colour is selected based on the distance. This pixel is passed to a module which generates an AXI Stream to fill one block on the screen. These block work in parallel blocking until the other has completed until enough lines have been produced.

4.8 Other IPs

The other minor IPs that were used in this design were

- Video Timing Controller: Generate timing signals for VGA output
- UART Lite: Facilitate debugging and loading programs into memory
- Memory Interface Generator: To access DDR RAM
- MicroBlaze Debugging Module: Debugging
- Clock Wizard: Generating the 3 required clocks; 100 MHz for system, 200 MHz for memory, and 25.175MHz for VGA pixel clock

5. Design Tree

The project is split into multiple git repositories each having the same structure. Each repository contains the following

- <project_name>.xpr: The vivado project
- <project_name>.srcs: The folder containing the sources

The main project repository also has these extra items

- <project_name>.sdk: Contains the software to run the system
- docs: Contains a copy of the design report

To setup the project clone all of the following projects into the same location. A name like "ip_repo" is suggested.

https://github.com/marknsikora/ov7670_i2c_controller

https://github.com/marknsikora/ov7670_axi_stream_capture

https://github.com/marknsikora/stereoscopic_comperator_ip

https://github.com/marknsikora/vga_ip

Clone the following project

https://github.com/marknsikora/G4_stereoscopic_depth

And add the ip location that the previous projects were cloned into.

To run the project; generate the bitstream, program the board, and run the software provided in the project sdk.

6. Tips and Tricks

During this term, we were assigned a new board to work with. In order to become successful in the course, we had to independently learn how to use the design software and IPs provided by Xilinx. Although there were useful documentation online for us to read, it was sometimes difficult for us to get the information we needed. Students from previous terms might experience less of it since TAs were be able to answer most of the questions. Self-learning is definitely a key for this course.

6.1 Self-learning

First trick would be to always register to online forums and seek for answers. In this case, we had to register onto Xilinx's forum to download the VDMA documentations. You can try to start a new thread and ask for help or dig into existing threads and study on previous discussions.

6.2 Suggestion

It would be really helpful if we can directly ask for help from Xilinx. For example, we spent the last two weeks trying to debug the VDMA bug and couldn't find the solution in the end. We were not sure if it was the VDMA IP who had the bug or the Xilinx documentation gave wrong information.