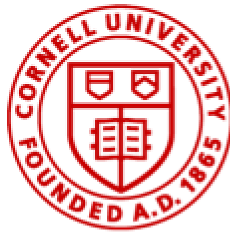
Stereoscopic Depth on an FPGA via OpenCL

A Design Project Report Presented to the Engineering Division of the Graduate School of Cornell University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering (Electrical and Computer)



Submitted by:
Ahmed Kamel
Aashish Agarwal

Date Submitted:
May 19, 2015

Project Advisor:
Dr. Bruce Land

Abstract

Master of Engineering Program
School of Electrical and Computer Engineering,
Cornell University
Design Project Report

Project Title: Stereoscopic Depth on an FPGA via OpenCL

Authors: Ahmed Kamel, Aashish Agarwal

Abstract: One of the greatest benefits of using a field programmable gate array (FPGA) is the ability to do computationally heavy algorithms in an energy efficient environment and with relatively high performance. The FPGA allows for the exploitation of high-level synthesis tools such as OpenCL via the ARM cores that it has available. For the proposed project, slightly horizontally offset image pairs are taken in from a camera that has a stereoscopic lens. These images are then run through two computationally heavy algorithms: bilateral filtering and segment based depth mapping. The bilateral filter preserves the sharpness of the edges within the image while blurring out the excess noise within the frame. The segment based depth mapping uses one image as a reference and pans through the other to find the matching pixels and their position. This is done to construct a disparity map, which is used to generate depth. One of the limitations in working with RTL is the developmental time due to the programming environment. While programming at the register transfer level (RTL) is exceptionally powerful in terms of utilizing the hardware correctly, it takes a large amount of time to develop. One solution for this issue was to use a high-level synthesis language, OpenCL. In previous years OpenCL has been utilized GPU's, but recently Altera has adapted this language for functionally on the FPGA. One characteristic of OpenCL that made it ideal for this project was its ability to parallelize certain tasks, which is very beneficial for image processing. This programming language requires two devices to run at full efficiency, a host and a device. In previous years the host has been a regular computer, while the device was generally a GPU. In order to make this project as portable as possible, the FPGA that was used also contained two ARM cores that take on the role of host while the FPGA itself would be the device.

// Executive Summary heading kept inside?

Table of Contents

Abstract	**1**
Executive Summary	**2**
Introduction	**5**
Background - Depth Map	**6**
Background - OpenCL	**7**
OpenCL - The New Technology	**7**
FPGA	**8**
Range of Solutions	**10**
Depth Extraction System Design	**10**
Preprocessing the Images	**10**
Filtering	**11**
Parameters	**15**
Disparity	**16**
Algorithm	**16**
Depth From Disparity	**17**
OpenCL System Design	**18**
Results / Evaluation	**24**
Speedup on FPGA vs ARM	**27**
Synthesis - Timing & Area	**28**
Synthesis report	**31**

Conclusion 31

References - 33

Data Sheets 35

Appendix 36

Introduction

Implementing a real-time 3D reconstruction of a scene from a low-cost depth sensor can improve the development of technologies in the domains of augmented reality, mobile robotics, and more. However, current implementations require a host computer with a powerful graphics processing unit (GPU), which limits its prospective applications with low cost and low power requirements. This project aims at developing a cheap and fast method of generating a depth map using OpenCL on Altera DE1-SoC. One of the greatest benefits of using a field programmable gate array (FPGA) is the ability to do computationally heavy algorithms in an energy efficient environment and with relatively high performance. The FPGA allows for the exploitation of high-level synthesis tools such as OpenCL via the ARM cores that it has available. For the proposed project, slightly horizontally offset image pairs are taken in from a camera that has a stereoscopic lens. These images are then run through two computationally heavy algorithms: bilateral filtering and segment based depth mapping. The bilateral filter preserves the sharpness of the edges within the image while blurring out the excess noise within the frame. The segment based depth mapping uses one image as a reference and pans through the other to find the matching pixels and their position. This is done to construct a disparity map, which is used to generate depth. One of the limitations in working with RTL is the developmental time due to the programming environment. While programming at the register transfer level (RTL) is exceptionally powerful in terms of utilizing the hardware correctly, it takes a large amount of time to develop. One solution for this issue was to use a high-level synthesis language, OpenCL. In previous years OpenCL has been utilized GPU's, but recently Altera has adapted this language for functionally on the FPGA. One characteristic of OpenCL that made it ideal for this project was its ability to parallelize certain tasks, which is very beneficial for image processing. This programming language requires two devices to run at full efficiency, a host and a device. In previous years the host has been a regular computer,

while the device was generally a GPU. In order to make this project as portable as possible, the FPGA that was used also contained two ARM cores that take on the role of host while the FPGA itself would be the device.

Background - Depth Map

The overall goal of the project is to generate a 3D render of environment; the short term goal is to be able to generate a depth mapping of a single frame. This is done using a specialized camera lens (a stereoscopic lens). There have been an extensive number of previous studies conducted to extract depth from single images as well as multiple images. Reference [1] uses a combination of monocular and binocular cues to recover a depth estimation using the help of machine learning algorithms.

Recently, Microsoft Kinect has become a viable option to extract depth from a 2D scene. The depth map is constructed by analyzing a speckle pattern of infrared laser light. This alternative is a high power solution because it is projecting a large amount of infrared light into the environment. The depth computation is all done by the PrimeSense hardware built into Kinect. However, implementing a low cost solution for the depth map with minimum hardware and lesser time is very important due to emergence of applications of these devices in 3D TV's, virtual realities and telepresence.

This project required some specialized hardware for the process of obtaining the images. It utilized a specialized stereoscopic lens that can be seen in figure 1. that was connected to Panasonic DMC-GF5 camera.

Figure 1: Panasonic Stereoscopic lens

Background - OpenCL

This project uses Altera's OpenCL Software development kit (SDK) on the DE1-SoC board with a dual ARM core on it. Building rich 3D maps of environments is an important task for mobile robotics, with applications in navigation, manipulation, semantic mapping, and telepresence. Intensive research has been conducted in recent decades to solve the problem of finding the correspondence between the right and the left images.

To achieve real time performances from complex algorithms it has become necessary to use massively parallel GPUs or field programmable gate arrays (FPGA) along with General Processors (GP). This heterogeneous computing model is employed in the PC world for graphics, gaming, rendering, server market etc. and now for the handheld/embedded world. To program such systems, OpenCL has emerged as a viable option with its inherent parallel programming nature. It greatly improves speed and responsiveness for a wide spectrum of applications in numerous market categories from gaming and entertainment to scientific and medical software.

OpenCL - The New Technology

This project utilizes technology that is fairly new and offers a great learning opportunity. Several decisions had to be made before work began, such as what hardware and programming language would be used. Working with an FPGA that is equipped with a processor element offers a unique environment full of challenges. The other aspect is the fact that this technology is fairly new. Alteras OpenCL SDK was only released in 2013, and is still growing with every update. Being on the cutting edge is a great opportunity that does not come about very often. The ability to investigate and characterize the performance of a newly developed platform will offer great insight into the underlying technology.

Another factor that plays a role in some of the design decisions is the prototyping aspect. If the Altera OpenCL SDK proves to be capable of pushing the current bounds of FPGA work, then this project can be set up as a foundation for future course material. On the other end of the spectrum, if OpenCL does not seem to be a feasible addition to any course due to the size of the designs, it can still be used for a final project.

FPGA

The FPGA that was utilized for the course of this project was the DE1-SoC. This FPGA supports OpenCL and is equipped with a Hard Processor System (HPS) that is a Dual-core ARM Cortex-A9. The HPS made this FPGA an ideal choice for this project. Each core runs at 800MHz and is equipped with 1GB of DDR3 SDRAM. The FPGA is part of the Cyclone V SoC family from the Altera family. The FPGA is equipped for 85K programmable logic elements and 4,450 Kbits of embedded memory, 64 MB off-chip SDRAM, 6 fractional PLL's, and 2 hard memory controllers. Another key reason the DE1-SoC was chosen was due to the video support that it contains. It has a VGA DAC with a VGA-out connector, as well as a TV decoder with a TV-in connector. If this project is to be expanded to a real time system, these two ports will play a crucial role. Figure 2 shows the other various connections that the DE1-SoC has and how they are connected to the FPGA.

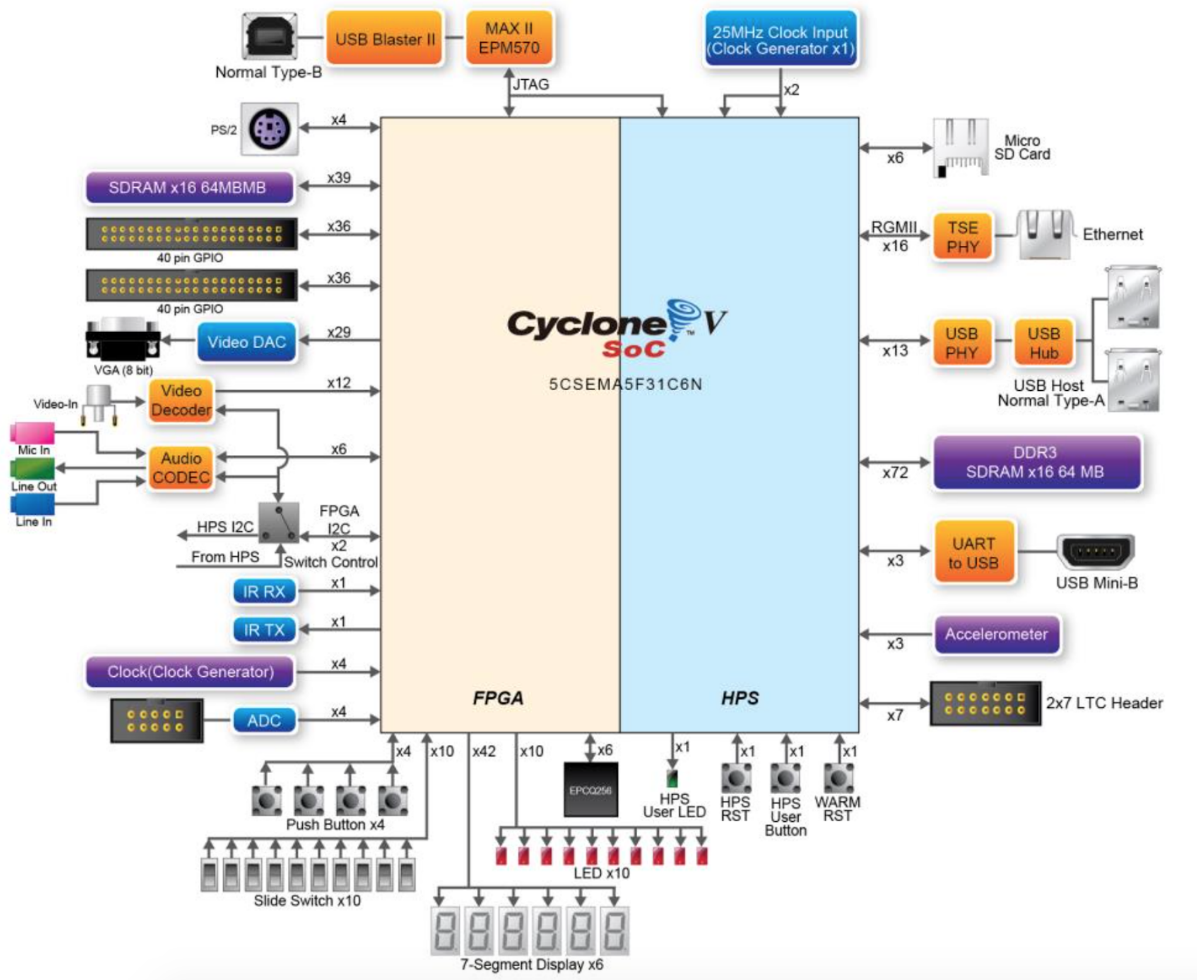


Figure 2: DE1-SoC layout
User Manual [6]

The overall design of the system utilizes both the ARM core and the FPGA to achieve the end goal. Initially the ARM core is used to set up the global memory for the stereoscopic image pair to be read. There are several options available to transfer the images onto the ARM core from the computer. The optimal solution would have been to set up the ethernet port on the FPGA and connect it to the network to do a direct transfer. However, the FPGA was not connected to the schools network due to limited access. Instead a USB drive was used to transfer the image pair from the computer to the ARM core. From there, these images were fed into a filter that removed unnecessary detail so that images can

produce higher quality depth results. These filtered images are then compared using the Sum of Absolute Differences (SAD) equation to generate a disparity map. The depth can then be directly extracted from the disparity map.

Range of Solutions

There are many different ways to extract the depth from a pair of 2D stereoscopic images. The most common is using Microsoft Kinect which uses image sensors to extract depth. However, this is an expensive solution considering the desired application is a low cost embedded system. There can be implementations done at software level in C++ using NIOS processor which also runs on FPGA soft-core, but the main drawback is serial nature of this type of implementation and slower processing speed. OpenCL is an ideal choice to implement this design due to its inherent parallel nature, compatibility with FPGAs, and device independent feature.

Depth Extraction System Design

The overall system had several steps involved in producing the final result. The images that came from the camera need to be pre-processed, filtered, and finally depth can be extracted. OpenCL allowed this task to be parallelized and developed in an efficiently and timely manner.

Preprocessing the Images

Before the images are ready to be filtered a few issues had to be addressed. The raw stereoscopic images that came in from the camera were in a Multi Picture Object (MPO) format. They were removed from this format and using GNU Image Manipulation Program (GIMP) converted to a 24 bit grayscale Bitmap Image File (BMP) format. The reason this format was used is due to the limitations that Altera's OpenCL SDK has for images. Even though OpenCL has its own libraries for image processing that contain support for 2D and 3D image objects, Altera's SDK does not support these

features as of now. A bitmap library was found on reference [4] that would enable the processing of the images. After careful considering and testing, this library was utilized for the reading and writing of BMP images from the host side of project.

Filtering

Before a stereoscopic image pair can be analyzed for calculating the depth, it first must be run through some form of filtering to reduce salt and pepper noise. The final depth extraction algorithm is based on intensity values of a pixel, and filtering helps false positives. One of the major portions of this project was to determine an appropriate filter to achieve the desired result. The filtering process can be done in either the spatial domain or frequency domain. A spatial domain filter would operate on the images in the format that they were captured, while a frequency domain filter would operate on the image after it has gone through a Fast Fourier Transform (FFT). The initial form of filtering was going to be done in the frequency domain. However, the overhead of incorporating an FFT with the project would have cost too much time, so a spatial domain alternative was determined to be the best course of action.

The idea of spatial image filtering is to have a mask or kernel of a certain size that applies a desired operation to the image pixels under the kernel. By moving the kernel around the image, all pixels are visited by the center of the kernel, leading to the entire image to getting filtered. The operation applied to the underlying image can be either linear or nonlinear. For a linear filter the kernel will be a matrix of coefficients that will be multiplied with the corresponding underlying image pixels. One of the most elementary filters to remove noise is the Gaussian blur. This is the fundamental concept of the bilateral filter; which was used in this project.

The Gaussian filter equation can be seen below:

$$G_\sigma(||p - q||) = \frac{1}{(2\pi\sigma^2)} exp(-\frac{||p-q||^2}{2\sigma^2})$$

Reference [7] (1)

The convolution by a positive kernel is one of the basic operations in image filtering; it directly leads to Gaussian filtering. It is essentially a weighted average of the intensities of surrounding pixels, with the weight decreasing with the spatial distance of the pixel from the center of the window. The spatial distance is defined as in equation one as $||p - q||$ and the term σ is a parameter defining the extension of the window. As this window runs over the entire image, the result is a blurring effect across the edges of the image. The results of such a filter can be seen below, figure 3 shows a raw image before it was run through a Gaussian convolution filter, while figure 4 shows the image after it has been filtered.

Figure 3: Raw unfiltered image

Figure 4: This is the Gaussian filtered image

Edge blurring is one of the desired behaviors for the internals of objects, but it causes a problem when attempting to extract depth. In order to easily extract depth from an image, the perimeter of an object is required to have a sharp edge. The ideal filter would blur out the internal details of an object in an image, while preserving the outer edges. The Gaussian blur is a nice start, but it does not produce the complete desired results.

The bilateral filter is essentially a Gaussian filter that also takes into account the intensities of the pixels along with spatial distance. The idea behind a bilateral filter is that two pixels are close to each other not only when they have spatial locality, but also if they have similarity in the intensity range. The result of the bilateral filter is denoted below as BF[*]:

$$BF[I]_p = \frac{1}{W_p} \sum_{q \varepsilon S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_P - I_q|)$$

Reference [7] (2)

The term W_p is a normalization factor that ensures the pixels' weighted sum is equal to 1 within the window. It is defined as:

$$W_p = \sum_{q \varepsilon S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_P - I_q|)$$

Reference [7] (3)

The bilateral filter is a non-iterative, nonlinear filter which combines domain (spatial closeness) and range (color similarity) filtering. The coefficients of the kernel depend on both variables, thus achieving good filtering behavior where the pixels' similarity is high e.g. inside regions. It also preserves areas where similarity is low, e.g. on edges. The bilateral filtering is less content dependent and provides exceptional results with a fixed set of parameters, thus giving highest quality at extra computational cost compared to other filters of its kind e.g anisotropic diffusion filtering.

Overall equation 2 is a normalized weighted average where $G_{\sigma s}$ is a spatial Gaussian that decreases the influence of distant pixels. $G_{\sigma r}$ is a range Gaussian that decreases the influence of pixels q with an intensity value different from p. Parameters σ_s and σ_r in equations 2 and 3 will control the amount of filtering for the image. As the range parameter σ_r increases, the bilateral filter becomes closer to Gaussian blur because the range Gaussian is flatter i.e., almost a constant over the intensity interval covered by the image. Increasing the spatial parameter σ_s smooths larger features. An important characteristic of bilateral filtering is that the weights are multiplied, which implies that as soon as one of the weight is close to 0, no smoothing occurs. As an example, a large spatial Gaussian coupled with narrow range Gaussian achieves a limited smoothing although the filter has large spatial extent. The range weight enforces a strict preservation of the contours. Figure 5 shows the results of the Figure 3 after running through a bilateral filter.

Figure 5: This is the bilateral filtered image

Parameters

For the bilateral filter, the parameters that were used were carefully decided upon and tested.

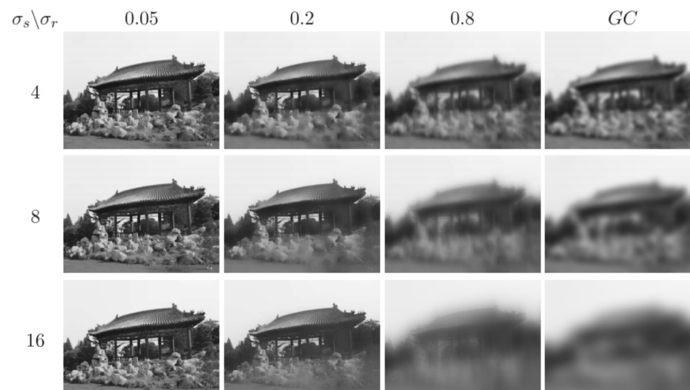


Figure 6: Different parameters for bilateral filter
Reference [7]

Figure 6 displays the various parameters for the bilateral filter that were ran in the paper [6]. By utilizing these results, we decided to go with a sigma factor of .05 and window size of 5.

Disparity

The disparity in a stereoscopic pair of images is computed using following equation:

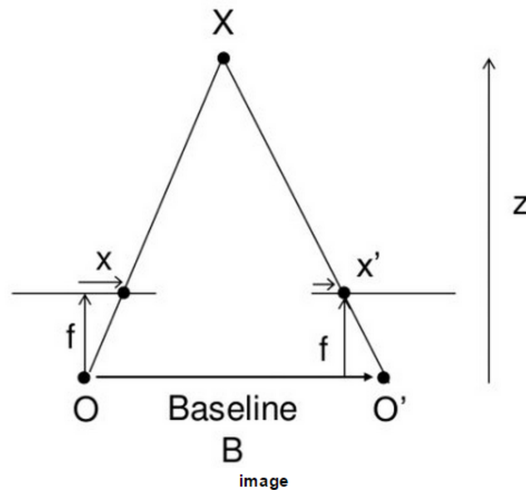


Figure 7: Fundamental visualization of disparity

Figure 7 has equilateral triangle whose equation will yield:

$$Disparity = x - x' = \frac{B*f}{Z}$$

[3] (4)

In equation 4, x and x′ are the distances between the point in the image plane corresponding to the point in a scene and the camera's center. B is the distance between the two lenses' focal points and f is the focal length of camera. So from the above equation, the depth of a point in a scene is inversely proportional to the difference in distance of corresponding image points and their camera centers. With this information, we can derive the depth of all pixels in an image.

Algorithm

The algorithm used in this project is based on a pixel by pixel approach for finding the disparity. SAD is an algorithm for measuring the similarity between image blocks. It calculates the disparity by taking the absolute difference between each pixel in the original block and the corresponding pixel in the

block being used for comparison. The sum of absolute differences is used for a variety of purposes, such as object recognition, or, for this project, the generation of disparity maps for stereo images. The basic steps required for finding the disparity between images is as follows:

- Select the image pixel I1(i, j) in the left image
- Finding matching pixel or correspondence pixel in right image
- find the absolute difference between them (disparity)

Figure 8 provides a visual representation of sum of absolute difference when applied to a stereoscopic image pair. For a pixel (3,4) in the left image, all the corresponding pixel intensities in the right image window are differenced. Then the minimum intensity difference pixel is indexed, which gives the disparity for that corresponding pixel in the left image.

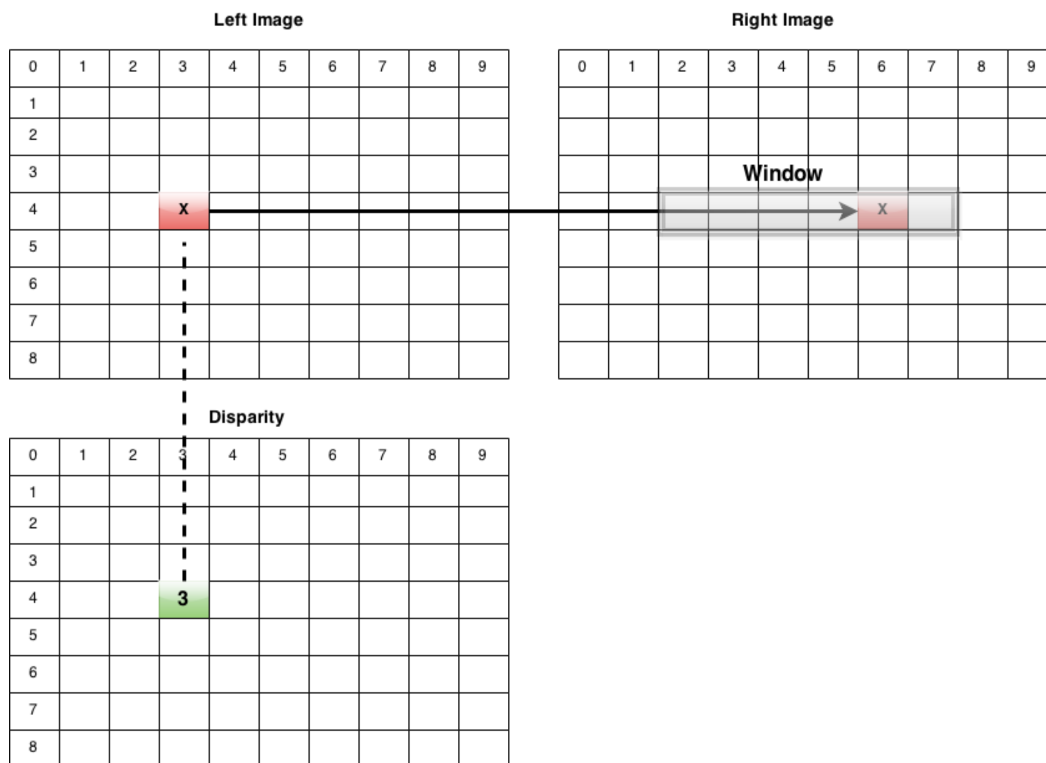


Figure 8: Visualization of disparity mapping

Depth From Disparity

From the calculated disparity the depth is computed using the following formula:

$$z = 255 - disparity \quad (5)$$

In equation 5, z represents the intensity of the pixel that will be mapped.

The original equation that was used is a fundamental equation based off of the lens specifications.

$$z = \frac{f*b}{disparity} \quad (6)$$

In equation 6, f represents the focal length of the lens, b represents the distance between the focal point of the lenses. After doing a visual comparison the results of both equations, 5 and 6, it was agreed upon to use equation 5 for the finals results. One of the reasons this equation was chosen was due to the greyscale mapping nature that it contains. The values for disparity can range between 0 to 50, thus guaranteeing that the mapped pixel will be within the range of 205-250; which is the ideal range to simulate depth.

OpenCL System Design

The OpenCL standard inherently offers the ability to describe parallel algorithms to be implemented on FPGAs. It allows the programmer to do this at a much higher level of abstraction than hardware description languages (HDLs) such as VHDL or Verilog. Although many high-level synthesis tools exist for obtaining this higher level of abstraction, they have all suffered from the same fundamental problem. These tools would attempt to take in a sequential C program and produce a parallel HDL implementation. The difficulty was not so much in the creation of an HDL implementation, but rather in the extraction of thread-level parallelism that would allow the FPGA implementation to achieve high performance. With FPGAs being on the furthest extreme of the parallel spectrum, any failure to extract maximum parallelism is more crippling than on other devices. The OpenCL standard solves many of these problems by allowing the programmer to explicitly specify and control parallelism. The OpenCL

standard more naturally matches the highly-parallel nature of FPGAs than do sequential programs described in pure C.

The creation of designs for FPGAs using an OpenCL description offers several advantages in comparison to traditional methodologies based on HDL design. Development for software programmable devices typically follows the flow of conceiving an idea: coding the algorithm in a high-level language such as C, and then using an automatic compiler to create the instruction stream. This approach can be contrasted with traditional FPGA-based design methodologies. Here, much of the burden is placed on the designer to create cycle-by-cycle descriptions of hardware that are used to implement their algorithm.

The traditional flow involves the creation of datapaths, state machines to control those datapaths, connecting to low-level IP cores using system level tools (e.g., SOPC Builder, Platform Studio), and handling the timing closure problems. Due to the influences of external interfaces that impose fixed constraints. The goal of an OpenCL compiler is to perform all of these steps automatically for the designers, allowing them to focus on defining and optimizing their algorithm rather than focusing on the tedious details of hardware design. This allows the designer to easily migrate to new FPGAs that offer better performance and higher capacities because the OpenCL compiler will transform the same high-level description into pipelines that take advantage of the new FPGAs. One of the reasons behind this is because OpenCL is device independent, the programmer simply specifies the device that will be used on the host side, and the compiler automatically optimizes the code for that device.

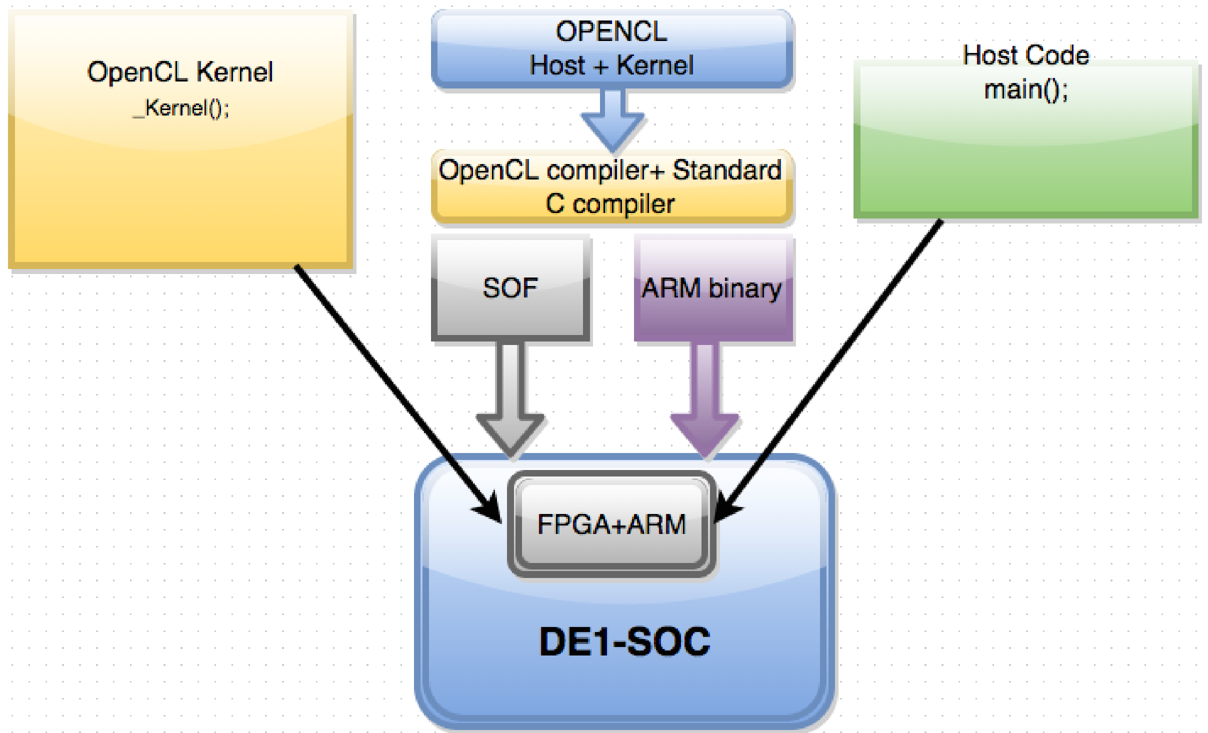


Figure 9: Block diagram of the overall setup.
User manual [6]

Down the middle of figure 9 we see the various compilers. The Altera OpenCL compiler generates an AOCX file that is used to specify the hardware configurations that the FPGA needs. The standard C compiler generates the binaries that will run on the host.

Figure 9 shows how the DE1-SoC board with FPGA+ARM core is used. The host can be written in C++ or C to configure the OpenCL Kernel. There are 6 data structures that the host manages. The first structure, the platform, is used to identify vendors' implementation of OpenCL. It gives a way to identify a way to access the device. The device is the hardware that the kernel will run on; for the course of this project that device is an FPGA. One OpenCL program is not limited to one device, multiple devices can be run using the same host code to achieve large scale parallelism. The next key data structure the host is responsible for is the program. The program container is used to hold a list of

kernels. Kernels are the specific functions or algorithms that will be ran, this is usually the computationally intensive task that is required to be ran. The kernel is also device independent, meaning that it can be ran on any device the that has OpenCL support(GPU's, CPU's, FPGA's). The next big data structure is the command queue. It is the primary source of communication between the host and the device, the host sends the kernels that need to be ran to the command queue to get queued up to be executed. The last vital data structure is the context, which is used to manage the connected devices. Figure 10 shows all of the various data structures are connected together. The DE1-SoC includes a 16-Kbyte memory that is implemented inside the FPGA. This memory is organized as 16K x 8 bits, and spans addresses in the range 0x08000000 to 0x08003FFF.

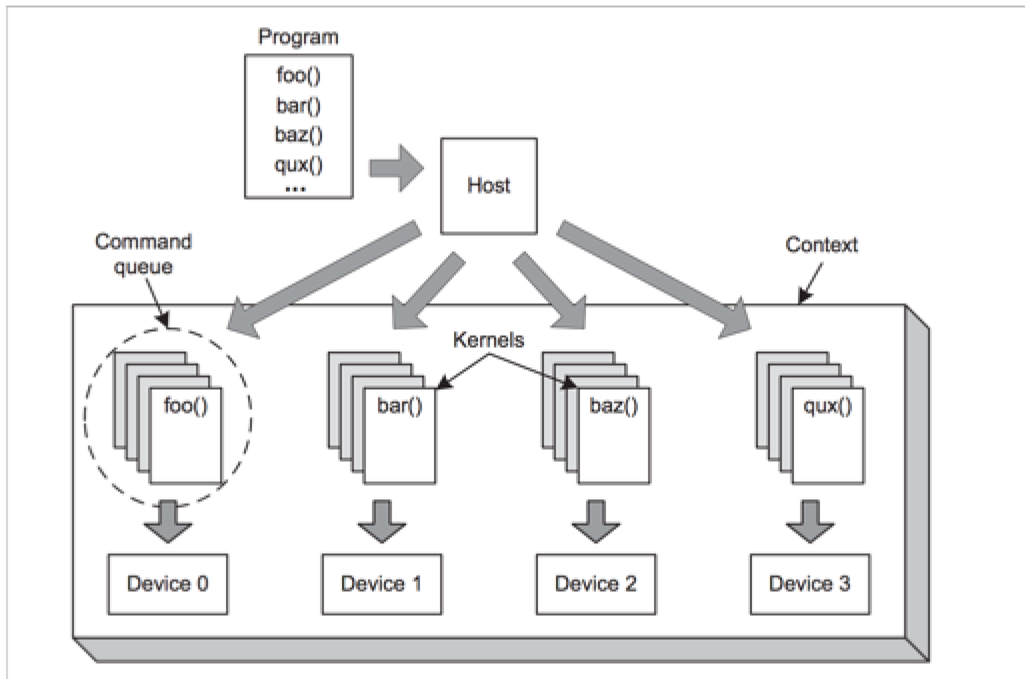


Figure 10: Device Management using Context Switching
Reference [8]

One of the key aspects of using OpenCL is the ability to parallelize tasks. An OpenCL kernel is executed via an array of work items, and all work items run a copy of the same code. Each individual work item is also assigned its own ID, which it can use to index a specific chunk of memory to run its computation on.

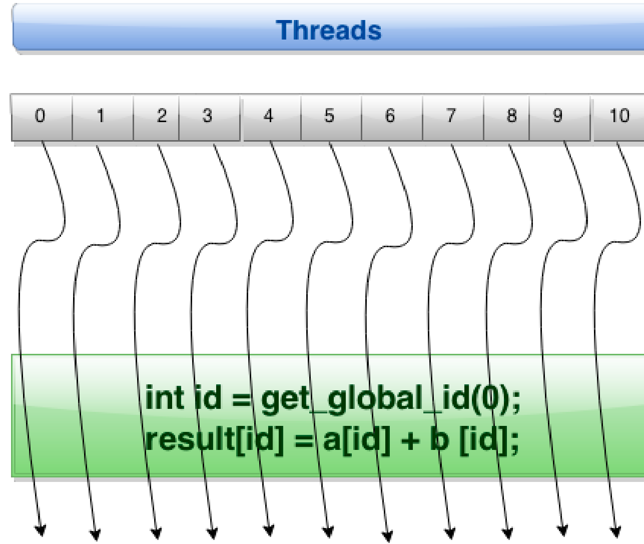


Figure 11: Basic thread layout
Reference [4]

Figure 11 shows how one work item is used to execute data independent operations in parallel via global ID. Several workgroups can be created that encompass these work items as shown in figure 12. OpenCL can divide the monolithic work item array into work groups.

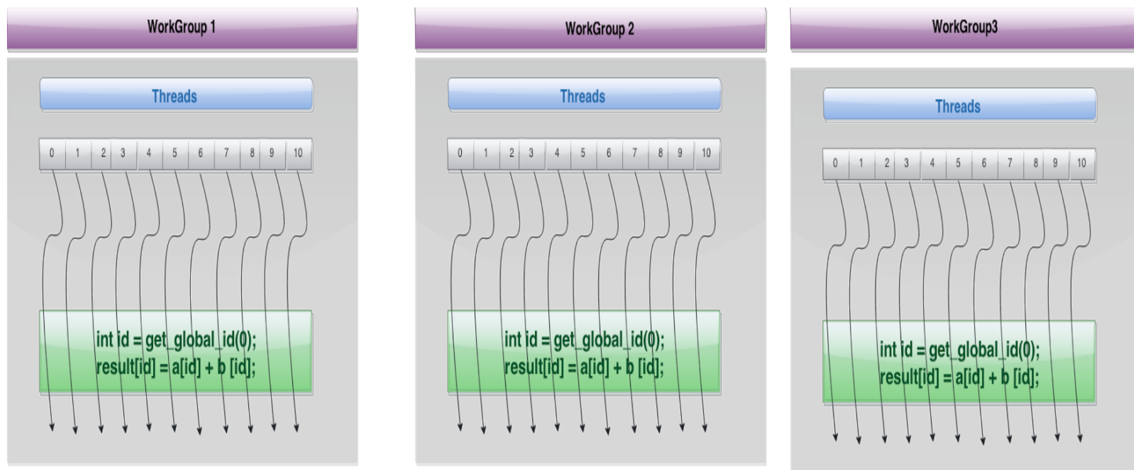


Figure 12: An overview of workgroups
Reference [4]

Parallel work is submitted to available devices by launching kernels. The Kernels run over global dimension index ranges (NDRange), broken up into "work groups", and "work items". These work items executing within the same work group can be synchronized with each other via barriers or memory fences. The work items in different work groups can't sync with each other, except by launching a new kernel. A Kernel is invoked once for each work item and each work item has private memory. Work items are grouped into a work group and each work group is capable of sharing local memory. The total number of all work items is specified by the global work size. global and constants memory is shared across all work work items of all work groups.

Memory management is one of the most important tasks in any program. OpenCL connects the device and host memory using various functions. The mapping between ARM and FPGA however is done through a shared global memory. This global memory is the main means of communicating reads and writes of data between host and device. Its contents visible to all threads but it has long access latency. In order to allocate memory memory on the device side, OpenCL has the function *clCreateBuffer*. This function allocates objects in the device Global Memory. It returns a pointer to the object and requires five parameters: OpenCL context pointer, Flags for access type by device, Size of allocated object, Host memory pointer, if used in copy-from-host mode, & Error variable. In order to transfer data from the host to device, OpenCL has the function clEnqueueWriteBuffer. This function requires nine parameters: OpenCL command queue pointer, Destination OpenCL memory buffer, Blocking flag, Offset in bytes, Sizeof bytes of written data, Host memory pointer, List of events to be completed before execution of this command, & Event object tied to this command. In order to transfer data from device to host, OpenCL has the function clEnqueueReadBuffer. This function also requires nine parameters: OpenCL command queue pointer, Source OpenCL memory buffer, Blocking flag, Offset in

bytes, Sizeof bytes of read data, Destination host memory pointer, List of events to be completed before execution of this command, & Event object tied to this command.

The OpenCL Memory Systems has certain flags that can be invoked.

- __global: large, long latency.
- __private: on-chip device registers
- __local: memory accessible from multiple PEs or work items. It can be SRAM or DRAM.
- __constant: read-only constant cache.

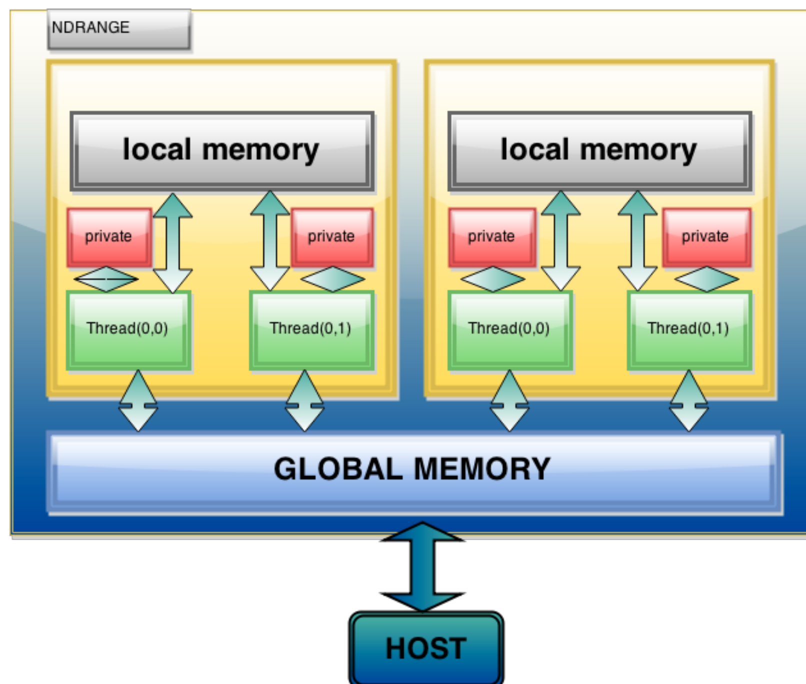


Figure 13: Visualization of memory
Reference [4]

Results / Evaluation

Figures 14 thru 18 display the the results of the raw image going into the bilateral filter and then going through the depth extraction / disparity mapping. The performance, in terms of speed is also evaluated; comparing FPGA to ARM core. The overall area usage of the FPGA as well is also investigated by

utilizing the diagram provided by quartus, as well as the synthesis report generated by the Altera Offline Compiler.

Figure 14: Left unfiltered image

Figure 15: Right unfiltered image

Figure 16: Left Bilateral filtered image

Figure 17: Right Bilateral Filtered Image

Figure 18: Depth map of the filtered images

Speedup on FPGA vs ARM

Different image sizes were run on the ARM core and FPGA. The results of of the computation time can be seen in table 1.

Time(sec)	**ARM**		**FPGA**	
Image	**Bilateral Filter (ms)**	**Depth (ms)**	**Bilateral Filter (ms)**	**Depth (ms)**
1600x904	101.108	38.779	2.942	37.204
1280x723	70.026	24.598	2.155	23.735
1024x578	44.862	15.534	1.44	15.246
720x406	21.416	7.638	0.647	7.483
640x361	17.46	5.882	0.755	5.901

Table 1: Timing results of different resolution images

For the bilateral filter, it is clear to see the increase in speed from using the FPGA. The root cause behind this is due to the complexity of the computation. The bilateral filter algorithm

involves a series of floating point multiplies which causes a significant increase in performance on the FPGA, nearly 32 times faster than the ARM core. These multiplications are easily parallelized and are capable of utilizing the DSP blocks of the FPGA. As we can see from the synthesis report found in table 2, the kernels utilized 71 out of 87 DSP blocks. A majority of the utilized DSP blocks belong to the bilateral filter kernel. Although this is a fairly large percentage for the size of the kernels created, this is partially due to the size of the DE1-SoC.

As for the depth extraction algorithm, it is observed that there is no drastic boost when transitioned to the FPGA; this is due to the lack of complexity of the algorithm. The depth extraction algorithm consists mostly of a comparison between two floating point numbers, no actual multiplication is involved. So the task of parallelizing the algorithm did not produce any significant result.

The speed of both of these algorithm will need to be improved if this project is to be expanded into a real time system. The critical path of the project is the depth extraction algorithm. The algorithm used is a fundamental and basic algorithm; perhaps for a real time expansion this algorithm can be replaced with one more complex and efficient.

Synthesis - Timing & Area

As seen from the synthesis report generated by Quartus, it can be seen that logic utilization is around 63% of the total available on the FPGA. The DE1-SoC is relatively smaller in terms of logic blocks and computation units, as compared to the conventual FPGA that would be utilized for such a project. The

DSP block utilization is 82%. As shown in figure 19 a large majority of the FPGA is utilized for this design. The main reason for this much usage of DSP blocks is due to the fact that the bilateral filter has many floating point multiplications (around 9), additions, and comparisons, which leads to a lot of hardware utilization. The depth block in the system has only one floating point subtraction as well as few comparators which is far less than the bilateral filter. This gives an insight into the trade-offs between area and performance of the system. Bilateral filtering helps improve the performance of the overall depth estimation by preserving edges but is also computation intensive, thus high hardware utilization. One can certainly use another filter like Gaussian or median and implement a different complex algorithm for calculating depth: there are many trade-offs involved. Either the filter can be complex or the algorithm can be computationally simple, a clear design choice that must be made in advance.

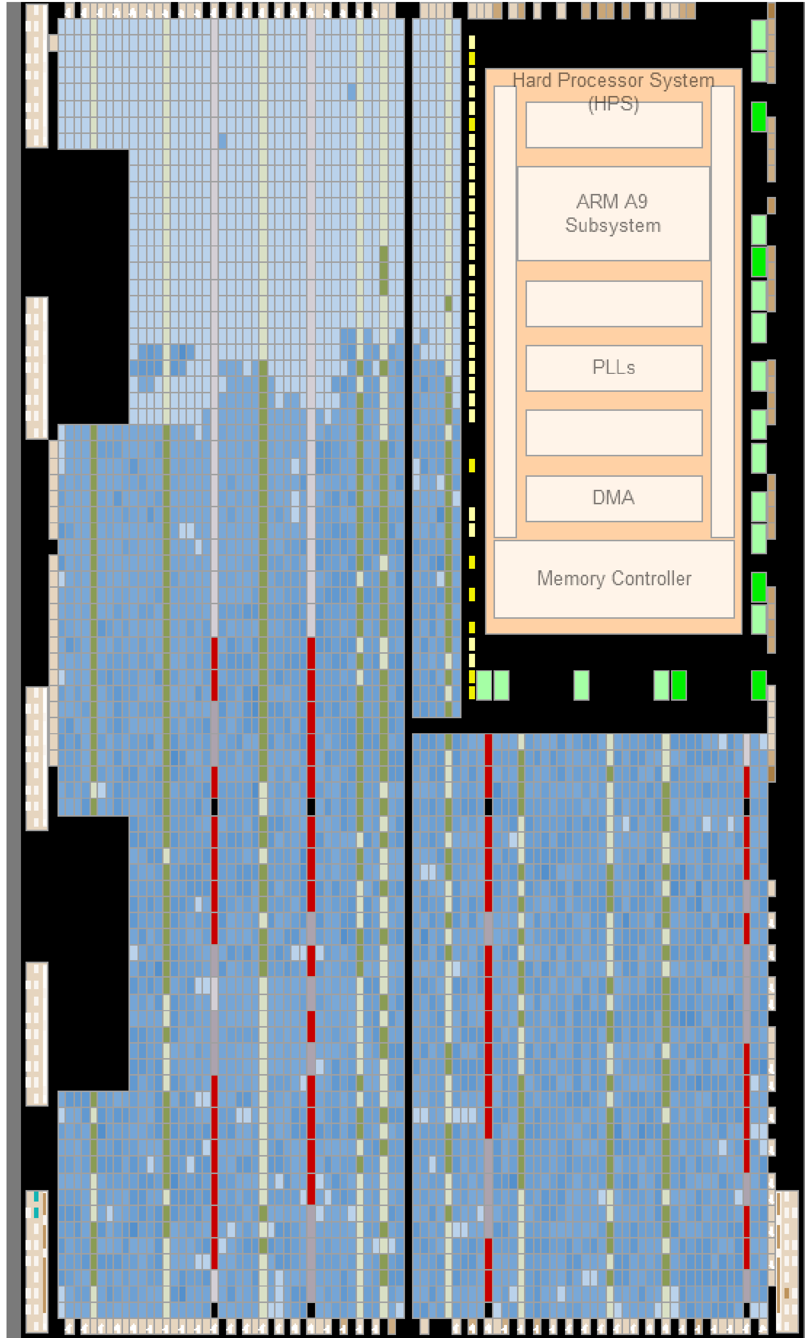


Figure 19: FPGA logic utilization

Synthesis report

Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	20,251 / 32,070 (63 %)
Total registers	42725
Total pins	103 / 457 (23 %)
Total virtual pins	0
Total block memory bits	1,237,800 / 4,065,280 (30 %)
Total DSP Blocks	71 / 87 (82 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	2 / 6 (33 %)
Total DLLs	1 / 4 (25 %)

Table 2: synthesis report of kernels

Conclusion

Various hardware and software challenges arose throughout the course of the project. Due to how recent the OpenCL SDK was released by Altera, the lack of support proved to be daunting. The complete interface between the FPGA and ARM Cores using OpenCL proved to be a challenge to fully understand. One of the biggest issues that arose with the project was attempting to get I/O working properly with the FPGA using OpenCL for an optional expansion. Of all of the challenges though, the

greatest one was the image support. Once the bitmap library was assimilated into the project, the remainder of the project was developed smoothly.

One expansion for this project that can be addressed is the possibility of a real time system. As stated in the evaluation section, the FPGA implementation of the depth extraction and bilateral filtering is still too slow to achieve a reasonable frame rate. These algorithms will need to be optimized a great deal to be usable in a real time environment. One possible optimization might be to enable loop unrolling for the depth extraction algorithm. More specifically the loop that runs through the right image to conduct the disparity mapping can enhanced. Overall though, the decision to go with OpenCL to implement the filter and depth mapping was a great learning opportunity for every member of the team.

References -

1. A. Saxena, J. Schulte and A. Ng. Depth Estimation using Monocular and Stereo cues. Proceedings of the 20th International Joint Conference on Artificial Intelligence, 2007.
2. Azevedo, Roberto Gerson De Albuquerque, Fernando Ismério, Alberto Barbosa Raposo, and Luiz Fernando Gomes Soares. "Real-Time Depth-Image-Based Rendering for 3DTV Using OpenCL." Advances in Visual Computing Lecture Notes in Computer Science (2014): 97-106. Web. 20 Jan. 2015.
3. Bodner, Ben, and Jessica Coulston. "Creating Depth Maps from Monocular and Stereoscopic Images." (n.d.): n. pag. Cias.rit.edu. 28 May 2012. Web. 15 Jan. 2015.
4. "CUDA Memory." CUDA Application Design and Development (2011): 109-31. Web.
5. "CUDA C Programming Guide." CUDA Toolkit Documentation. N.p., 05 Mar. 2015. Web.
6. Karapetsas, Lefteris. "Playing with OpenCL: Gaussian Blurring | Intelligent Rumblings." *Intelligent Rumblings*. N.p., 30 Aug. 2012. Web. 17 Jan. 2015.
7. Paris, Sylvain, Pierre Kornprobst,, Jack Tumblin, and Fredo Durand. "Announcements." Bilateral Filtering: Theory and Applications 4.1 (2008): n. pag. Http://people.csail.mit.edu/sparis/publi/2009/fntcgv/Paris_09_Bilateral_filtering.pdf. Web. 17 Feb. 2015.
8. Scarpino, Matthew. "A Gentle Introduction to OpenCL." Dr. Dobb's. N.p., 03 Aug. 2011. Web. 8 Jan. 2015.
9. Tomasi, C.; Manduchi, R., "Bilateral filtering for gray and color images," *Computer Vision, 1998. Sixth International Conference on* , vol., no., pp.839,846, 4-7 Jan 1998

10. Visentini, Giovanni, and Amit Gupta. "Depth Estimation Using Open Compute Language (OpenCL)." Depth Estimation Using Open Compute Language (OpenCL). N.p., 17 Mar. 2012. Web. 05 Jan. 2015.

Data Sheets

1. Altera SDK for OpenCL Getting Started Guide:

   http://www.altera.com/literature/hb/opencl-sdk/aocl_getting_started.pdf

2. Altera SDK for OpenCL Programming Guide:

   http://www.altera.com/literature/hb/opencl-sdk/aocl_programming_guide.pdf

3. Altera SDK for OpenCL Optimization Guide :

   http://www.altera.com/literature/hb/opencl-sdk/aocl_optimization_guide.pdf

4. Cyclone V SoC Development Board - Reference Manual:

   https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/rm_cv_soc_dev_board.pdf

5. DE1-SoC Manual. Rev E

   http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836&PartNo=4

6. DE1-SoC OpenCL User Manual

   http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836&PartNo=4

Appendix

https://github.com/ayk33/MENG_Project.git