Implementation of Image Processing Techniques on Hardware Accelerators

Project report submitted

in partial fulfillment of the requirement for the degree of

Bachelor of Technology

By

DEPARTMENT OF ELECTRONICS ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY (BHU), Varanasi

(April, 2019)

Approval Sheet

This project report entitled "**Implementation of Image processing techniques on hardware Accelerators**", by the following student(s) is approved for the degree of Bachelor of Technology, Electronics Engineering.

Aman Shreshtha (16095005)

Deepak Maurya (16095019)

Asif Hasan (16095084)

Arpit Gupta (16095085)

Examiners

_____

_____

_____

_____

_____

Supervisor

Dr. Kishore P. Sarawadekar

Head

Dr. Manoj Kumar Meshram

Date: _____

Certificate

It is certified that the work contained in the project report titled "**Implementation of Image processing techniques on hardware Accelerators**", by the following student(s) has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Name(s) of student(s)

Aman Shreshtha (16095005)

Deepak Maurya (16095019)

Asif Hasan (16095084)

Arpit Gupta (16095085)

Dr. Kishore P. Sarawadekar
Electronics Engineering
IIT (BHU) Varanasi
April, 2019

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Aman Shreshtha (16095005)

Deepak Maurya (16095019)

Asif Hasan (16095084)

Arpit Gupta (16095085)

Date: _____

Table of Contents

Chapter 1

Introduction

1.1 Image Processing

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps:

- Importing the image via image processing toolbox
- Analyzing and manipulating the image
- Output in which result can be altered image or report that is based on image analysis.

There are two types of methods used for image processing namely, analogue and digital image processing. Analogue image processing can be used for the hard copies like printouts and photographs. Digital image processing techniques help in manipulation of the digital images by using computers or some sort of hardware. The three general phases that all types of data have to undergo while using digital technique are pre-processing enhancement/extraction and display.

1.2 Aim of the project

In this project, we will be dealing with digital images and their processing techniques. The handling of digital images is a subject of widespread interest. Image processing is used to modify pictures to improve them (enhancement, restoration), extract information (analysis, recognition, edge detection) and change their structure (composition, image editing). Hardware accelerators (FPGAs) are increasingly used in modern imaging applications namely image filtering, medical imaging, image compression, and wireless communication. Application specific hardware implementation offers much greater speed than a software implementation. With advances in the VLSI (Very Large Scale Integrated) technology hardware implementation has become an attractive alternative. Implementing complex computation tasks on hardware and by exploiting parallelism and pipelining in algorithms yield a significant reduction in

execution times. The need to process the image in real time leads to implement them in hardware, which offers parallelism, thus significantly reduces the processing time. The drawback of most of the methods is that they use a high-level language for coding, which requires thousands of coding lines for image processing applications which is inefficient as it takes much time. To solve this problem, a tool called Xilinx System Generator (XSG), with graphical interface under the MATLAB-Simulink is used which makes it very easy to handle concerning other software for hardware description. FPGA is a form of highly configurable hardware while DSPs are a specialized form of microprocessors. System Generator is the modeling tool in which designs are captured in the DSP friendly Simulink modelling environment using Xilinx specific Block set.

Chapter 2

Review of Literature

2.1 MicroBlaze

MicroBlaze on Xilinx's Cost-Optimized Portfolio FPGAs offers advances in tool suite and FPGA platform to help simplify development effort and minimize system budgets. Xilinx has focused on improving the usability of the MicroBlaze processor softcore, enabling engineers to rapidly configure an embedded hardware platform and immediately start software coding in an industry standard environment.

As an embedded processor, MicroBlaze enables Xilinx's FPGA portfolio to meet the integration and performance requirements of industrial, communications infrastructure, medical device, automotive, and consumer markets. With the right combination of platform and processor, designers can gear a solution to meet their architectural challenges that incorporate the right combination of I/O peripherals, communications interfaces, real-time capability, and operating system support.

Initially working in this project, we start with MicroBlaze processor, it requires knowledge of System C to operate this processor. It is quite easy to handle it in comparison to Verilog HDL, but still, it is difficult and time-consuming to implement any image processing algorithm on it. Unlike writing code for each part, Simulink is model-based where we use these parametrizable blocks to implement the desired algorithm of image processing instead of writing code for individual blocks.

2.2 Hardware Co-simulation

Co-simulation refers to the simulation of heterogeneous systems whose hardware and software components are interacting. Traditionally, the task has been performed only after the prototype hardware became available and with the help of in-circuit emulators and/or other techniques. Co-design is a way to simulate at a very high level of abstraction, prior to the actual implementation. With hardware-software codesign, it is essential to verify correct functionality even before hardware is built. In contrast to the conventional (or homogeneous) simulation of digital hardware, co-simulation should care for the interaction among hardware and software components.

Chapter 3

Current Investigation

3.1 Components

1. Xilinx Vivado Design suite

Vivado Design Suite is a software suite produced by Xilinx for synthesis and analysis of HDL designs, superseding Xilinx ISE with additional features for system on a chip development and high-level synthesis.

Vivado enables developers to synthesize their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Vivado is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors.

2. Xilinx System Generator (XSG)

Xilinx System Generator is an architecture-level design tool to define, test and implement high-performance DSP algorithms on Xilinx devices. System Generator is part of the Vivado Design Suite and provides Xilinx DSP Block set such as adders, multipliers, registers, filters and memories for application specific design. These blocks leverage the Xilinx IP core generators to deliver optimized results for the selected device. Previous experience with Xilinx FPGAs or RTL design methodologies is not required when using System Generator. Designs are captured in the DSP friendly Simulink modeling environment using a Xilinx specific Block set. All of the downstream FPGA implementation steps including synthesis and place and route are automatically performed to generate an FPGA programming file. Advantage of using Xilinx system generator for hardware implementation is that Xilinx Block set provides close integration with MATLAB Simulink that helps in co-simulating the FPGA module with pixel vector provided by MATLAB Simulink Blocks. The System Generator block defines which type of FPGA board will be used, as well as provide several additional options for clock speed, compilation type and analysis. With a library of over 90 DSP building blocks, System Generator allows for faster prototyping and design from a high-level programming stand point. Some blocks such as the M-code and Black box allow for direct programming in MATLAB M-code,

C code, and Verilog to simplify integration with existing projects or customized block behavior. System Generator projects can also easily be placed directly onto the FPGA as an executable bit stream file as well as generating Verilog code for additional optimizations or integration with existing projects within the Xilinx ISE environment.

3. Simulink

Simulink is a simulation and model-based design environment for dynamic and embedded systems, integrated with MATLAB. Simulink, also developed by MathWorks, is a data flow graphical programming language tool for modelling, simulating and analyzing multi-domain dynamic systems. It is basically a graphical block diagramming tool with customizable set of block libraries. It allows you to incorporate MATLAB algorithms into models as well as export the simulation results into MATLAB for further analysis. There are several other add-on products provided by MathWorks and third-party hardware and software products that are available for use with Simulink. Simulink is capable of systematic verification and validation of models through modelling style checking, requirements traceability and model coverage analysis.

4. MATLAB

MATLAB is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python.

5. Field Programmable Gate Array

Field Programmable Gate Arrays (FPGAs) represent reconfigurable computing technology, which is in some ways very suitable for image processing. FPGAs generally consist of logical blocks and some amount of Random Access Memory (RAM), all of which are wired by a vast array of interconnects. All logic in FPGA can be rewired, or reconfigured with different purposes as many times as a designer likes. Continual growth in the size and functionality of FPGAs over recent years has resulted in an increasing interest in their use as implementation platforms for image processing applications. One of the benefits of FPGA is its ability to execute operations in parallel, resulting in remarkable improvement in efficiency. Considering availability, cost, design cycle and ease to handle, FPGA is chosen to implement image

processing algorithms in this project. We have used the Xilinx Kintex – 7 KC705 evaluation board to perform hardware co-simulation of the image processing algorithms. The Kintex-7 FPGA KC705 Evaluation Kit includes all the basic components of hardware, design tools, IP, and pre-verified reference designs including a targeted design enabling high-performance serial connectivity and advanced memory interfacing.

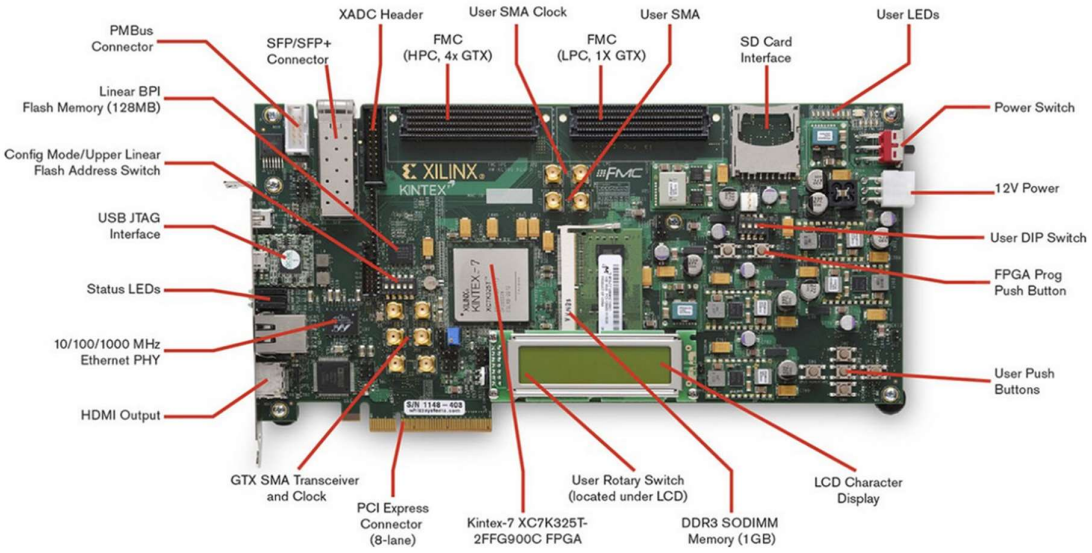


Figure 1: Kintex KC-705 Evaluation Board

3.2 Image Specification

In our experiments, the hardware co-simulation is performed on a square image i.e. the image has equal number of pixel rows and pixel columns. The input image is first converted to a 256-color bitmap and then processed using the FPGA.

3.3 Implemented Image Processing Algorithms

Development of models is based on algorithms used for Image Processing. Some of the basic algorithms are described below:

Image Negative

A negative image is a total inversion, in which light areas appear dark and vice versa. Negative images are useful for enhancing white or grey details embedded in dark regions of an image.

The negative of an image with gray levels in the range [0, L-1] is obtained by using the expression

$$s = L-1-r$$

L-1 = Maximum pixel value

r = Pixel value of an image

In our experiments, we deal with grayscale images with pixel values in the range 0 to 255 where 0 corresponds to dark region and 255 to that of bright region. Image Negative can be implemented simply by subtracting value of each pixels by 255.

Original Image

Image negative

Sobel Edge Detection

The Sobel operators used in image processing particularly at intervals edge detection algorithms. Technically, its discrete differentiation operator, computing an approximation of the gradient of the image intensity perform. At each purpose in the image, the results of the Sobel operator are either the corresponding gradient vector or the norm of this vector. Sobel operator is a smaller amount deteriorated in high levels of noise, and this adds the extendibility to the selection of an operator. The Sobel operator is based on convolving the image with a little, separable, and integer valued filter in the horizontal and vertical direction and is thus comparatively cheap in terms of computations. The operator uses two 3×3 kernels that are convolved with the first image to obtain the edge or high passed image by calculating

approximations of the derivatives -one for horizontal changes, and one for vertical. 3X3 spatial masks for the Sobel operator is given in the figure.

-1	0	+1
-2	0	+2
-1	0	+1

G_x

+1	+2	+1
0	0	0
-1	-2	-1

G_y

Figure 2: Convolution kernel along X and Y direction

These kernels can then be combined to find the absolute magnitude of the gradient at each point using

$$|G| = (Gx2+Gy2)\ 0.5$$

Typically, an approximate magnitude is computed using the formula

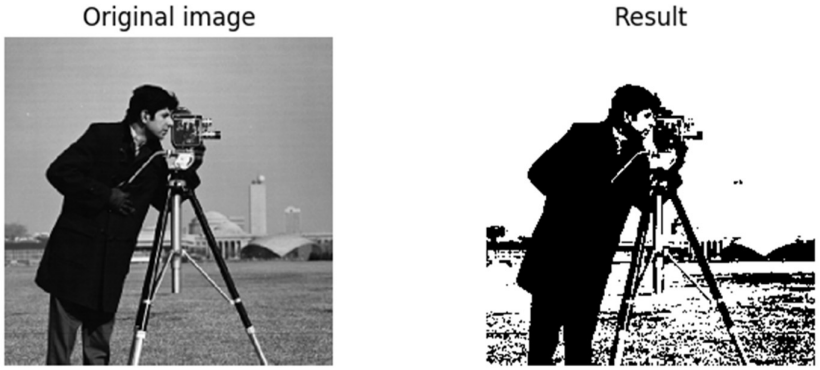
$$|G| = |Gx| + |Gy|$$

This is much faster to compute. Even though the accuracy in Sobel edge detection is relatively low; it has the advantage of simplicity in its calculation.

Image Thresholding

In Image thresholding, we replace each pixel in an image with a black pixel if the image intensity is less than some fixed constant T, or a white pixel if the image intensity is greater than that constant. Image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. Image thresholding is most effective in images with high levels of contrast.

For implementing the algorithm, a suitable constant value of T is taken e.g. 150. A Mux block is used for replacing the original pixels by white (grayscale value 255) or black (grayscale value 0) pixels.



3.4 Workflow of Image processing with Xilinx System Generator

System Generator works within the Simulink model-based design methodology. It uses the Xilinx DSP block-set for Simulink and will automatically invoke Xilinx Core Generator™ to generate highly-optimized netlists for the DSP building blocks. System Generator can execute all the downstream implementation tools to product a bit stream for programming the FPGA. For accomplishing Image processing task using Xilinx System Generator needs two Software tools. One is MATLAB Version R2016a or higher and Vivado Design Suite has to be configured to MATLAB. This result in addition of Xilinx Block set to the MATLAB Simulink environment which can be directly utilized for building the algorithmic model. The algorithms are developed, and models are built for image negative, Sobel edge detection, image thresholding using the library provided by Xilinx Block-set. The image pixels are provided to Xilinx models in the form of an array with a time stamp in Xilinx fixed point format. These models are simulated in MATLAB Simulink environment with suitable simulation time and simulation mode and tested.

The reflected results can be seen using post-processing in MATLAB. Once the expected results are obtained System Generator is configured for suitable FPGA board. FPGA board that used here is Kintex KC705, and the model is implemented for JTAG hardware co-simulation. The System generator parameters are set and generated. On compilation, a main module is generated wrapping all the different blocks along with bitstream file and other useful files then this file is downloaded on FPGA.

Following are the detailed steps involved:

Image pre-processing:

First of all, the image on which the algorithm is to be implemented requires pre-processing in terms of conversion to gray-scale (standard requirement for our purpose and in general for many image processing applications). Moreover, as image is two-dimensional (2D) arrangement, to meet the hardware requirement, the image should be pre-processed and converted to a one-dimensional (1D) vector. For this, we use MATLAB where we convert our bitmap image into a grayscale signal (a vector) that is a suitable data form for the FPGA hardware.

The code for the same is given in the Appendix.

Building the Simulink model:

Next, we create the model for our algorithms in Simulink using standard and Xilinx DSP block sets.

The “From workspace” and “To workspace” blocks basically act as input and output interfaces between the workspace and the co-simulation block. The “From workspace” block sends the data from workspace named “grayScaleSignal” to FPGA co-simulation block as input and “To workspace” block receives the output from the FPGA co-simulation and uses that data to create a variable in the workspace named “filteredImage.”

Apart from this, all Xilinx blocks should be connected between Gateway In and Gateway Out. Between those two blocks, any technique can be designed. All Xilinx blocks work on fixed point but the real-world signals (image, voice signal, etc.) are floating point so here the gateway in and gateway out blocks acts as translators for converting the real-world signal into the desired form.

Figure 3: Data Interfacing block

Rest of the blocks used are algorithm specific.

Following are the designed Simulink models for various algorithms we implemented:

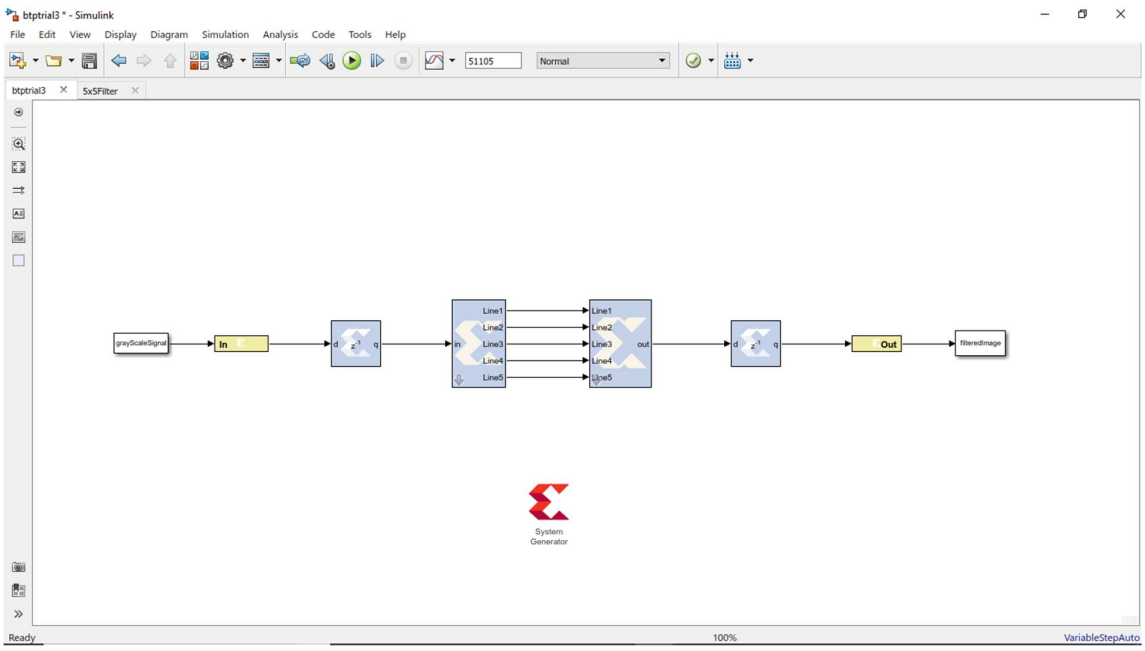


Figure 4: Sobel Edge detection block design using System Generator

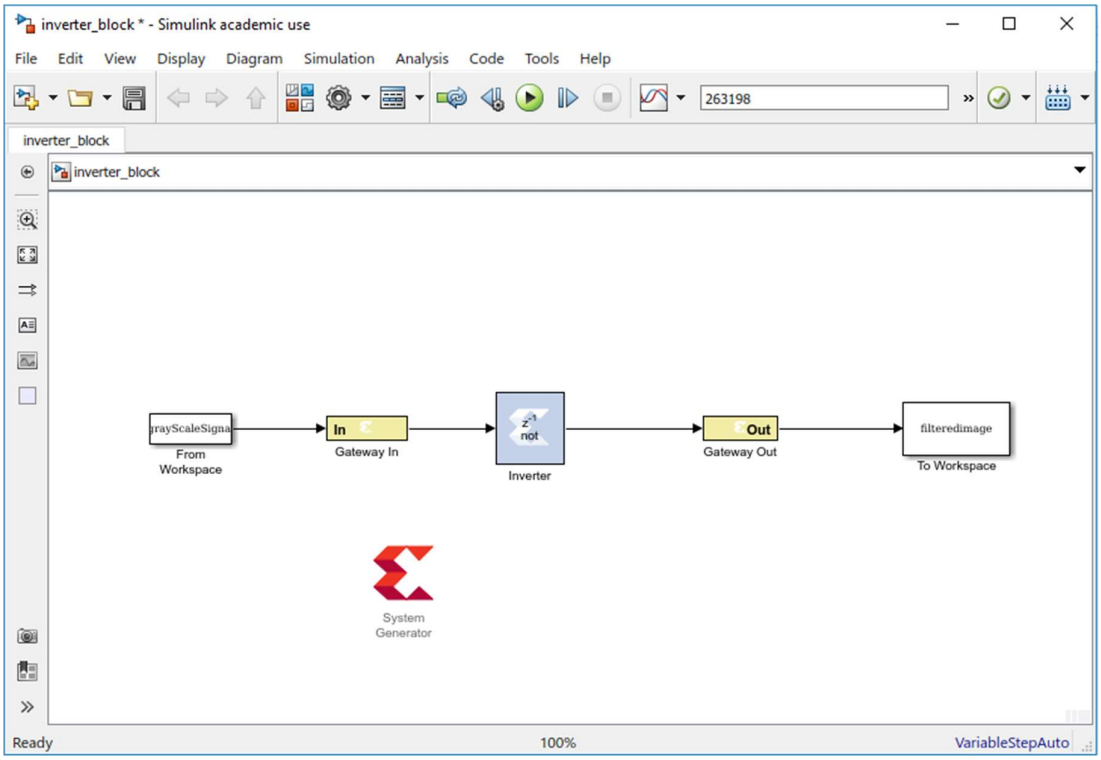


Figure 5: Image Negative block design using System Generator

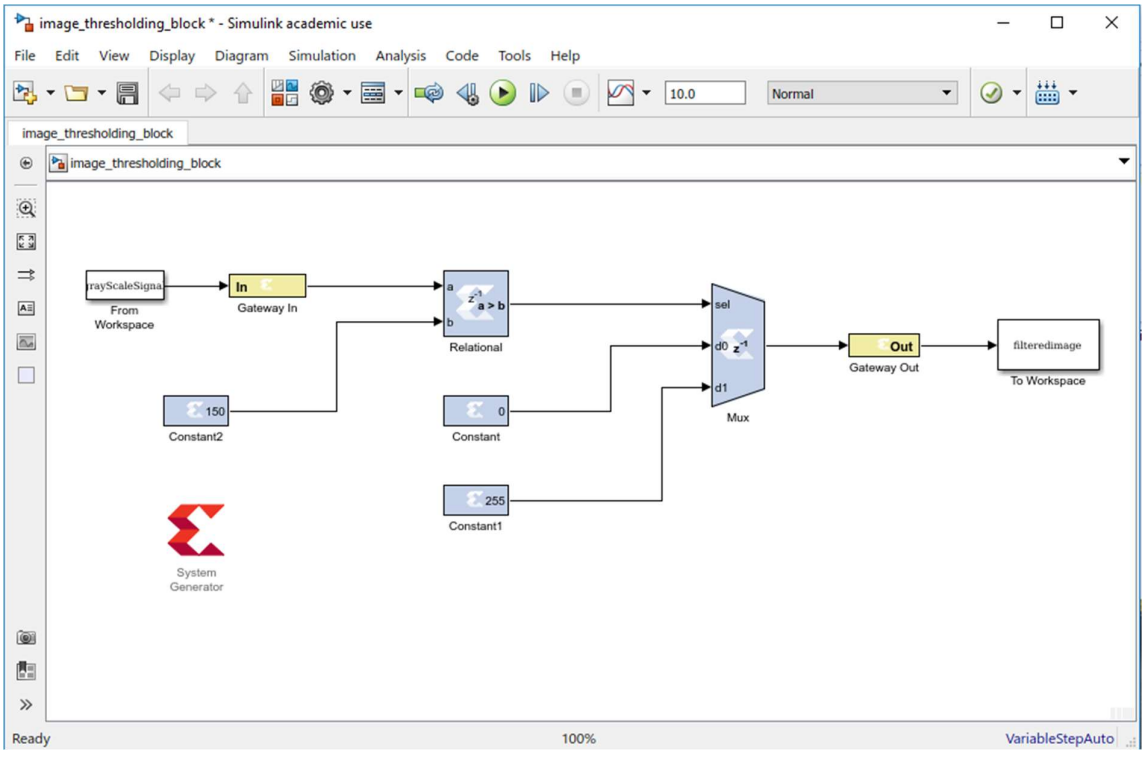


Figure 6: Image thresholding block design using System Generator

Generation of the Co-simulation block:

Once the model design is ready, next comes the generation of the co-simulation block. This is achieved using the system generator token. By configuring the token as per our algorithm and hardware used, we can generate the co-simulation block.

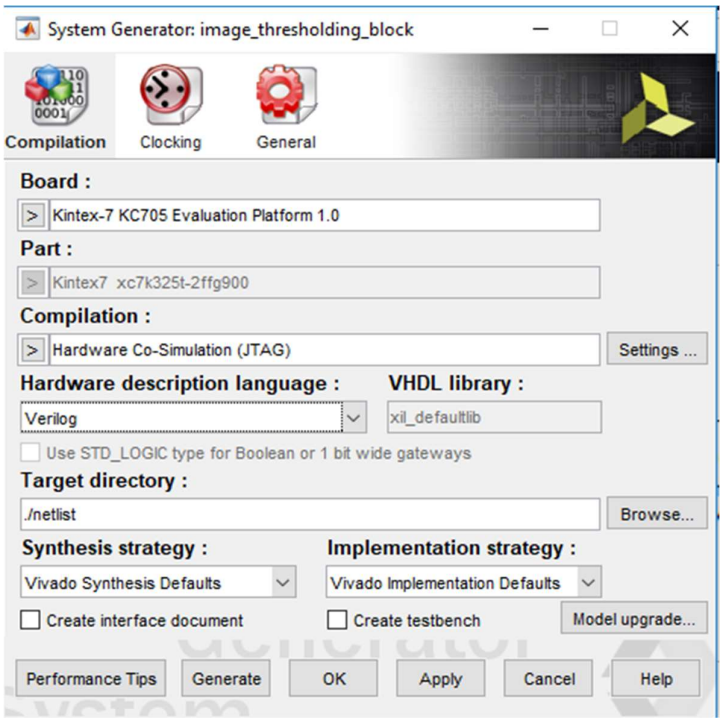


Figure 7: System Generator Dialog Box

Co-simulation implementation on hardware:

Once the co-simulation block is generated, we assemble it by interfacing it with the MATLAB workspace using “From workspace” and “To workspace” blocks. Also, the system generator token is added that basically acts as the hallmark for validating the block for upcoming hardware implementation. We copy-paste the token from the original model as it allows the System Generator block to retain all settings necessary for the co-simulation.

Steps followed in Hardware Co-simulation System generator is configured as:

A.) Compilation Target

- Parts: Defines the FPGA part to be used (Kintex7 KC705 Evaluation 1.0).
- Synthesis tool: Specifies the tool to be used to synthesize the design.

- Hardware Description Language: Specifies the HDL language to be used for compilation i.e. Verilog.

B.) Clocking Tab

- FPGA clock period(ns): Defines the period in nanoseconds of the system clock. (0.2 for our case)
- Clock pin location: Defines the pin location for the hardware clock.

C.) Calling the Code Generator

- The code generator is invoked by pressing the Generate button in the System Generator token dialog box.

Next, we require to estimate the co-simulation time which depends on the size of the image. Following equation gives the co-simulation time:

$$\textbf{Time} = W^2 + 2W + 30$$

This equation will provide ample time for the system, assuming that the input image is a square image and "W" represents the pixel width of the image.

Next the implementation of the co-simulation system consists of only two simple tasks. The first is to connect the FPGA hardware to the host computer, and the second is to run the co-simulation model.

Once the board is connected properly, the simulation time is set to the calculated value and the simulation is started. Upon completion of the simulation run time, the co-simulation process is complete, and the output signal can be accessed through the Workspace output variable, "filteredImage."

Image post-processing:

This step is related to displaying the output signal created by the FPGA hardware. The data that was received from the FPGA is stored in the MATLAB Workspace as a variable named "filteredImage."

Actually, the result from the FPGA board is one dimensional (1D), so we need to convert it into two dimension(2D), exactly opposite to the process what we did while pre-processing. The MATLAB code for the same is given in appendix.

This code basically translates the output data back into a two-dimensional structure that can be read and displayed via MATLAB.

Chapter 4

Results and Discussions

4.1 Results

Figure 8:The post-processed images obtained after co-simulation

4.2 Discussions

- The square bitmap test image used yield outputs that exactly conforms with the expected simulation results.
- Generation of co-simulation block for each algorithm took certain amount of time depending upon the complexity and number of individual low-level blocks used.

- The sampling time of blocks were configured depending upon the number of lines to which data is made available simultaneously
- Only grayscale square images were used as the focus was to develop understanding of algorithm implementation on hardware and test the co-simulation design.

Chapter 5

Conclusion

Based on our project, we can conclude that:

- Hardware implementation of complex image processing algorithms is advantageous in the sense that we can efficiently utilize embedded components such as adders, multipliers as well large memory space.
- Hardware software co-simulation technique is quite effective in implementing image processing algorithms on hardware as well as analyzing the processed output by comparison with software simulated results. Moreover, co-simulation is the perfect solution for testing purposes.
- Xilinx System Generator provides the necessary resources to perform the hardware software co-simulation. It contains many low-level blocks that can be used to create any number of different systems in the Simulink design environment, and co-simulation is a quick and easy solution for testing a system's compatibility with FPGA hardware.

Future Work:

- **Implementing haze removal technique on FPGA**
  Images of outdoor scenes often contain degradation due to haze, resulting in contrast reduction and color fading. The objective of haze removal technique is to reduce the effect of haze or fog in images clearly & efficiently by using the haze removal algorithm.
- **Implementing custom DSP filters**
  Xilinx System Generator can be used to design DSP filters with the required functionality on a FPGA.
- **Implementing real-time video processing on FPGA**
  Video Processing is widely used for a broad range of applications, such as weather prediction, computerized tomography and artificial intelligence. Owing to the parallel architecture, an FPGA is able to perform high-speed video processing.

Chapter 6

Appendix

MATLAB Image pre-processing Code:

```
[ImageData, map] = imread('lena.bmp');
lineSizw= size (ImageData,1);
Npixels = size (ImageData,1) * size ( ImageData,2);
grayScaleImage = 0;
for I = 1 : lineSize,
   for J = 1 : lineSize,
        pixel = double ( ImageData(I,J) );
        mapValue = map(pixel+1);
        grayPixel = mapValue * 255;
        grayScaleImage  (I,J) = uint8(floor (grayPixel));
    end
end
%turn the array into a vector
grayScaleSignal = reshape ( grayScaleImage, 1, NPixels);
% insert a column of 'time values'
% block expects time followed by data on every row of the
  input
grayScaleSignal  =  [  double(0  :  Npixels-1)'  double(
  grayScaleSignal)' ];
```

MATLAB Image post processing Code:

```
if (exist('filteredImage','var') & exist('lineSize','var') &
   exist('NPixels','var'))

filteredImageSize=size(filteredImage);

designLatency = 20+2*lineSize;

if ((~isempty(filteredImage)) & (filteredImageSize(1) >=
   (designLatency+NPixels-1)))

% Reshape Simulink Output into a 2-D Image

rawImage =
   uint8(floor(reshape(filteredImage(designLatency:designLat
   ency+NPixels-1),

lineSize, lineSize)));

% Plot Original and Filtered Images

h = figure;

clf;

colormap(gray(256));

set(h,'Name',' Filtering Results');

subplot(1,2,1);

image(grayScaleImage), ...

axis equal, axis square, axis off, title 'Original
   Image';Page | 11

subplot(1,2,2);

image(rawImage), axis equal, axis square, axis off;

filterTitle = 'Filtered Image';

title(filterTitle)

colormap(gray(256));

end

end
```

References

[1] Xilinx-FPGA Co-Simulation of Gaussian Filter Algorithm
Michigan State University
ECE 480
Tom Ganley
November 19, 2010

[2] FPGA Implementation for Image Processing Algorithms Using Xilinx System Generator
Neha. P. Raut1, Prof.A.V.Gokhale 2
1P G Scholar VLSI, Department of Electronics Engineering,1,2Yeshwantrao Chavan College of Engineering
Nagpur, Maharashtra, India

[3] Vivado Design Suite Tutorial Model-Based DSP Design Using System Generator
UG948 (v2018.1) April 4, 2018

[4] Implementation of FPGA Based Image Processing Algorithm Using Xilinx System Generator
Deepesh Prakash Guragain1, Pramod Ghimire2, Kapil Budhathoki3
1, 2Asst.Professor, Dept of Electronics and Communication Engineering, Nepal Engineering College, Bhaktapur Nepal
3E-Matrix Pvt. Ltd, Kathmandu, Nepal

[5] SystemGenerator for DSP
Getting Started Guide
UG639 (v 14.3) October 16, 2012

Acknowledgments

i. Project report for "Implementation of Image processing techniques on hardware accelerators" submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology.

ii. Thesis Guide was referred to for preparing the thesis. Specifications regarding thesis format have been closely followed.

iii. The contents of the thesis have been organized based on the guidelines.

iv. The thesis has been prepared without resorting to plagiarism.

v. All sources used have been cited appropriately.

vi. The thesis has not been submitted elsewhere for a degree.