Michigan State University
ECE 480
Tom Ganley
November 19, 2010

FPGA Co-Simulation of Gaussian Filter Algorithm

Application Note

Abstract

FPGA co-simulation of Gaussian Filter algorithms can be useful in many different applications, such as developing more complex systems to be compatible with FPGA hardware. Using the blocks designed by Xilinx for its System Generator software, a simple algorithm for a Gaussian filter can be designed and tested using Simulink and an FPGA development board. This application note will make use of the Spartan-3A DSP xc3sd3400a-47g676 development board to explain how to co-simulate Xilinx's highest level form of a Gaussian filter on a still image. This document begins by explaining how to convert a still image into a format useable by Matlab and Simulink, shows the construction of a basic algorithm using Xilinx's System Generator blocks, and also illustrates the procedure of implementing the design using the FPGA development board. This note will pay particular attention to the preciseness of data compatibility because all developments in the design process will only be useable if the developments account for the data types communicated by the FPGA hardware

Keywords: **FPGA, System Generator, Co-Simulation, Gaussian Filter**

Index

1) Introduction

Gaussian filters are used in many digital image and video processing systems as a morphological operation for improving the quality of an input image with respect to its ability to have more complex systems operate successfully on it. A Gaussian filter is implemented in a Xilinx System Generator block as a 5x5 window that scans five lines of an image at a time and blurs the entire image. This is useful because in algorithms like edge detection, a noisy pixel with a strong magnitude can cause false positives in the results. This risk can be reduced by blurring the image with a Gaussian filter. The 5x5 filter block, combined with others used to accommodate the data types communicated by Xilinx's Spartan-3A DSP xc3sd3400a-47g676 development board, can be used in the Simulink design environment to develop systems ready for implementation on the FPGA hardware.

During the development stage of a system that will be implemented on an FPGA, it is sometimes simpler and more efficient to use co-simulation to test new additions to the design. Co-simulation is a process that allows the FPGA board to receive data inputs from the host computer, process the data and implement the design, and then return the outputs to the host computer for analysis. The process implies careful design techniques by the user to ensure that all data types used in the Simulink design environment will be compatible with the FPGA hardware. Through careful design techniques, co-simulation can be a very useful tool with a relatively small learning curve.

2) Objective

The objective of this application note is to introduce the user to co-simulation using the Xilinx Spartan-3A DSP xc3sd3400a-47g676 development board and the Simulink design environment, and how to implement a very simple Gaussian filter through the use of co-simulation. The note will guide the user through the processes of preparing the input data to the system, preparing the system for use with the correct FPGA board, building the system, generating the co-simulation block, and implementing the co-simulation.

The system outlined in this note for a 5x5 filter of any kind was not designed by nor was the code provided written by the author of the note. The design was provided by Xilinx as a demo application found at the following location and will be implemented step by step to allow the user to gain experience in building System Generator models:

- {Xilinx Root Location}\11.1\DSP_Tools\nt\sysgen\examples\demos\sysgenConv5x5.mdl

3) Image Preparation

3.1 Properly Store Image of Interest

The first stage in the process for this simple filter example is to save the image of interest into a new directory where the project will be saved. It is recommended that the user chooses a square image for this tutorial so that no changes to the code provided by Xilinx will be necessary. Once an image is chosen, save it as a 256 color bitmap image.

The following image will be used as a case study for the purposes of this application note and has been saved into the project directory as "baseball.bmp":

Figure 1. Input Image

3.2 Convert Image to Grayscale Signal Compatible with FPGA Hardware

For many image and video processing applications, the input image is first converted to grayscale. The following code, written by Xilinx for the sysgenConv5x5 demo, can be used to convert the bitmap image into a grayscale signal that is a suitable data form for the FPGA hardware:

```
[sysgenConv5x5_imageData, map] = imread('baseball.bmp');

lineSize = size(sysgenConv5x5_imageData,1);
NPixels = size(sysgenConv5x5_imageData,1) * size(sysgenConv5x5_imageData,2);

grayScaleImage = 0;
for I = 1:lineSize,
   for J = 1:lineSize,
       pixel = double(sysgenConv5x5_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' in front -- the from workspace
% block expects time followed by data on every row of the input
grayScaleSignal = [ double(0:NPixels-1)'  double(grayScaleSignal)'];
```

Copy and paste the above code into the Matlab Command Prompt to generate all the variables necessary for the co-simulation. The most important of the variables is "grayScaleSignal," which will be used as the input to the co-simulation algorithm.

4) Build the Filter Model in Simulink

4.1 Create New Model and Insert Blocks

Begin the process by opening a new Simulink model, and save the model to the same location as the input image.

The next step is to insert the input and output variables for the co-simulation into the new model, followed by the Gateway blocks. Begin by opening the Simulink Library Browser, and on the left-hand side, under "Libraries," navigate to Simulink → Sources. Choose the "From Workspace" block as shown below.

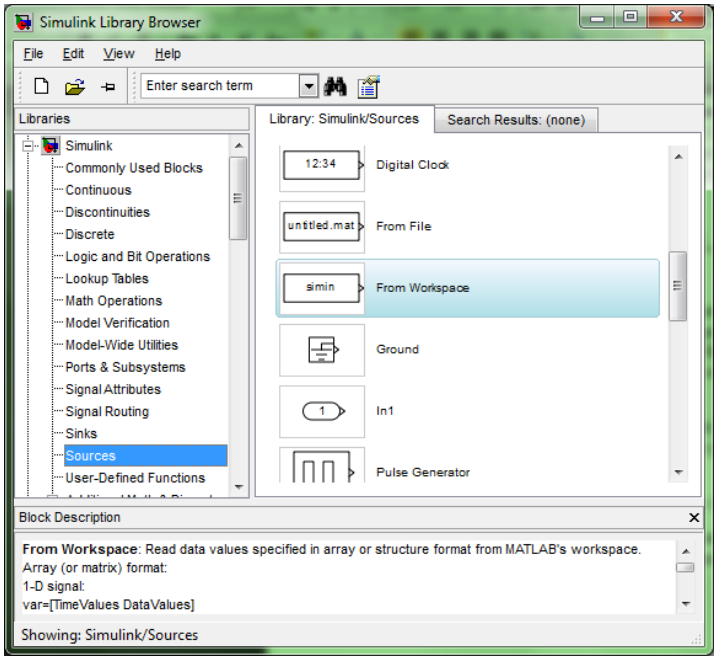

Figure 2. Simulink Library Browser

Drag this icon into the new model to create an instance of this function. Similarly, navigate to Simulink → Sinks, and drag a "To Workspace" block into the model.

To associate the "From Workspace" block with the correct variable from the Workspace, double-click on the block. In the window that opens, specify the Data name as "grayScaleSignal," change the Sample time to 1, uncheck "Interpolate Data," uncheck "Enable zero crossing detection" and change the value for "Form output after final data value by:" to "Setting to zero."

Similarly, double-click the "To Workspace" block. In the window that opens, specify the Variable name as "filteredImage," change the Sample time to 1, and change the "Save format:" field to "Array." Leave all other fields defaulted. This block will receive the output from the FPGA co-simulation and use the data to create a variable in the Workspace named "filteredImage."

Next, the model must have Gateway blocks to allow the system to understand that the Workspace variables are the respective input and output of the algorithm, but the blocks that are within these Gateway blocks are those which will be implemented on the FPGA. Begin by navigating in the Simulink Library Browser to Xilinx Blockset → Basic Elements and add one Gateway In block and one Gateway Out block to the system. In the model, connect "grayScaleSignal" to the input of the Gateway In block and the output of the Gateway Out block to "filteredImage." The model should appear as follows:

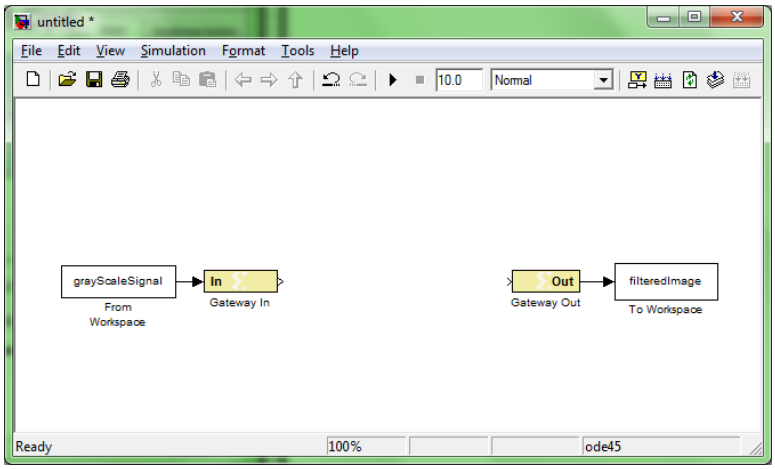

Figure 3. Input and output of the system

Double-click the Gateway In block, and change the number of bits to 8, the Binary point to 0, and change the Output type to Unsigned. Click OK.

Double-click the Gateway Out block, and change the IOB timing constraint to "Data rate; set 'FAST' attribute." Click OK.

Next to be added to the model are the 5 line buffers, the 5x5 filter and registers. The registers simulate D-Flip-Flops and should be used to synchronize the input and output with the FPGA clock.

In the Simulink Library Browser, navigate to Xilinx Blockset → Memory, and place two Register blocks from this location into the model. Connect the Gateway In block to the input of one Register block, and connect the output of the other Register block to the Gateway Out block.

In between the Register blocks is where the blocks for the algorithm are placed. First, the Gaussian filter will be implemented using Xilinx's 5x5Filter Block. Find this block by

navigating to Xilinx Reference Blockset → Imaging, and place one instance in the model. Connect this block's output to the input of the Register that is connected to the Gateway Out block. Also, double-click the 5x5Filter block and change the "5x5 Mask" field to "Gaussian."

The 5x5Filter block is a 5x5 mask that scans five lines of the image at a time to apply the chosen filter. Therefore, the system requires a buffer that accepts five lines from the image at a time to be fed into the filter. This can be accomplished using the Virtex2 5 Line Buffer found by navigating to Xilinx Reference Blockset → Imaging. Place one of these blocks into the system model, and connect its input to the output of the Register connected to the Gateway In block. Then connect the buffer's outputs to the respective inputs of the 5x5Filter block.

In order for the 5 Line Buffer to behave properly, the user must specify the length of a line in the image. Therefore, since the image used in this tutorial is a 300x300 pixel image, the 5 Line Buffer is double-clicked, and "300" is entered for "Line Size."

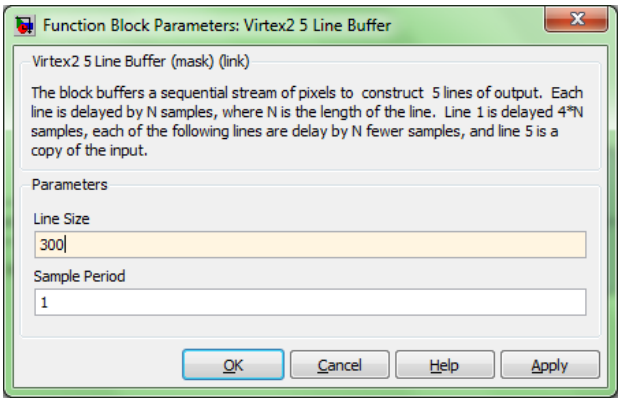

Figure 4. Enter line size of image

The final step in any System Generator design is to add a System Generator block. This allows the design to be synchronized with parameters specific to the FPGA hardware being used. Find the System Generator block by navigating to Xilinx Blockset → Basic Elements. Place one instance anywhere in the system, and double click the block. This will open the System Generator dialogue.

Change the Compilation field by choosing Hardware Co-simulation → Spartan-3A DSP 3400A Development Platform → JTAG. This will allow communication to the board through the JTAG programming cable. Change the Simulink system period to 1/5 second and then click OK. This completes the system design, and the resulting system should appear as follows:

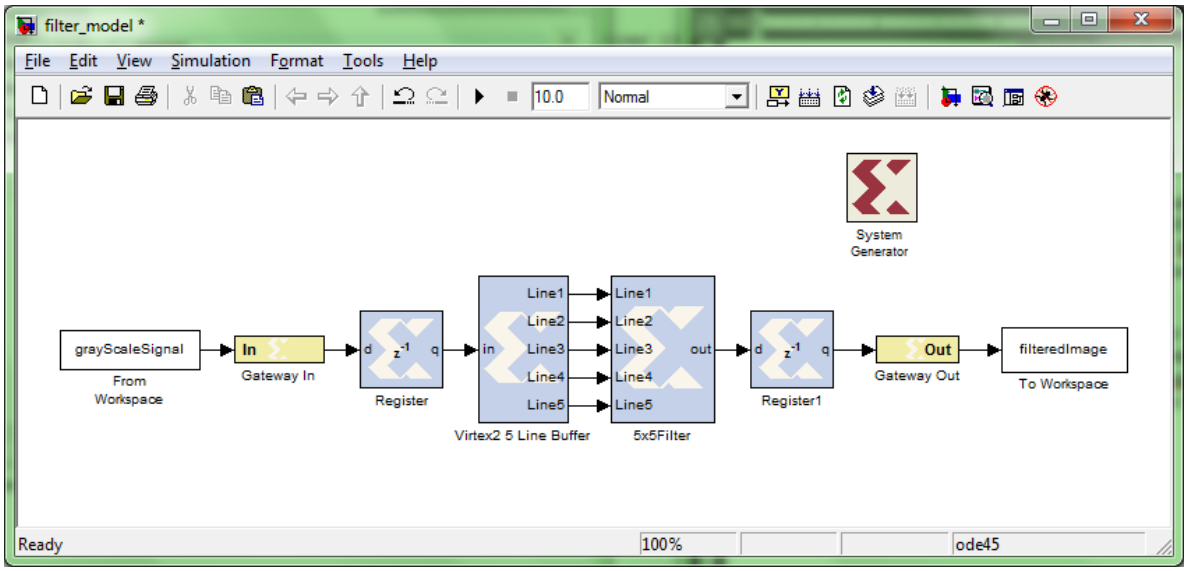

Figure 5. Gaussian filter design using System Generator

5) Co-simulation

5.1 Simulation Time

To begin the co-simulation process, the user must specify the amount of time that the system should run to successfully obtain all output data. To determine the run time for the system, use the following equation:

$$\boldsymbol{T = W^2 + 2 * W + 30}$$

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.

5.2 Generate the Co-simulation Block

Next in the process is generating the co-simulation block. To complete this step, double-click the System Generator block inside the system. Because the dialogue box that opens has already been configured when the block was added to the system, the system is prepared for generation.

Click Generate.

The generation process can take several minutes, and once finished, the output JTAG Co-simulation block will open in a new model. This model is conveniently saved in a directory within the project directory titled "Netlist." The file is named "{*}_hwcosim_lib," where "{*}" is the name of the original system model. Open a new model and copy and paste the co-simulation block into this new model. Save the new model in the original project directory.

5.3 Assemble the Co-simulation System

To begin completion of the co-simulation model, copy and paste the "grayScaleImage" From Workspace block and paste it in the new model. Connect the output of the Workspace block to the input of the JTAG Co-sim block. The JTAG Co-sim block contains all Xilinx blocks within the Gateway blocks of the original system, and even includes the Gateway blocks at its ports.

After the input is connected, copy and paste the output Workspace block from the original system into the co-simulation system. Then connect the output of the JTAG Co-sim block to the input of the output Workspace block.

The final addition to the co-simulation system is to add the System Generator block, which must be added to any System Generator design. For simplicity, copy and paste the System Generator block from the original system into the co-simulation system. This will allow the System Generator block to retain all settings necessary for this co-simulation tutorial.

This concludes the assembly of the co-simulation system model, and the system should appear as follows:

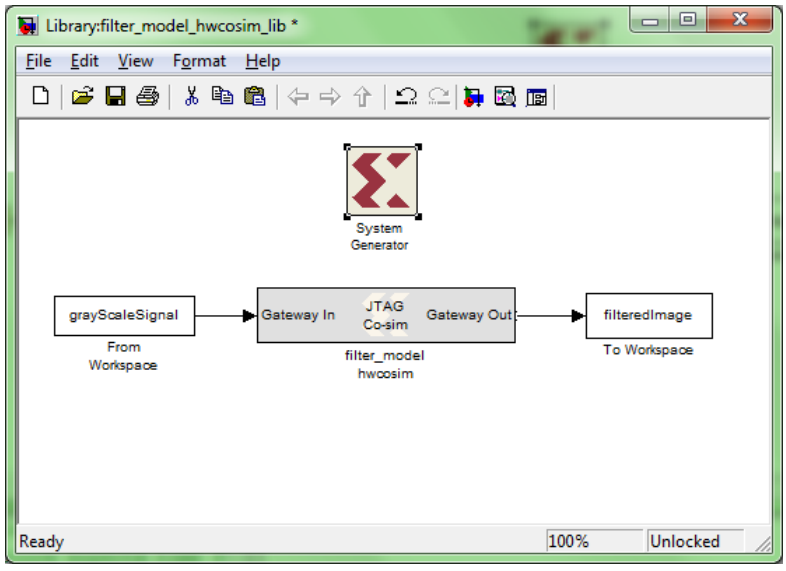

Figure 6. Co-simulation model

5.4 Co-simulation Implementation

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.

Begin by assembling the FPGA hardware as described in the application note, "Spartan-3A DSP FPGA Video Starter Kit Assembly: Getting Started with Demos," by Pascha Grant. Set up all hardware, with the camera and DVI Out connections being negligible, and connect the board to the host computer using the serial to USB cable and the JTAG programming cable.

Once the board is turned on, a list of demo applications will appear in the serial communication GUI. To temporarily erase these applications from the board, press the "Reset FPGA" button located on the FPGA development board. This will allow the co-simulation to access the hardware for the filtering algorithm.

Once the board is connected properly, check that the simulation time is still set to the same value as in the original system design and click the Start simulation button :

Upon completion of the simulation run time, this co-simulation process is complete, and the output signal can be accessed through the Workspace output variable, "filteredImage."

6) Displaying the Output of the Co-simulation

The final step in the co-simulation process is displaying the output signal created by the FPGA hardware. The data that was received from the FPGA is now stored in the Matlab Workspace as a variable named "filteredImage." Use the following code provided in the original demo written by Xilinx by copying and pasting it into the command prompt:

```
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:designLatency+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';
```

```
        subplot(1,2,2);
        image(rawImage), axis equal, axis square, axis off;
        filterTitle = 'Filtered Image';
        title(filterTitle)
        colormap(gray(256));


    end
end
```

This code translates the output data back into a two dimensional structure that can be read by Matlab. The output is then displayed as well as the original input image for comparison.

Below is the output of the co-simulation process used in this tutorial with the original image of a baseball player compared next to the algorithm's output of the baseball player image processed by a Gaussian filter:

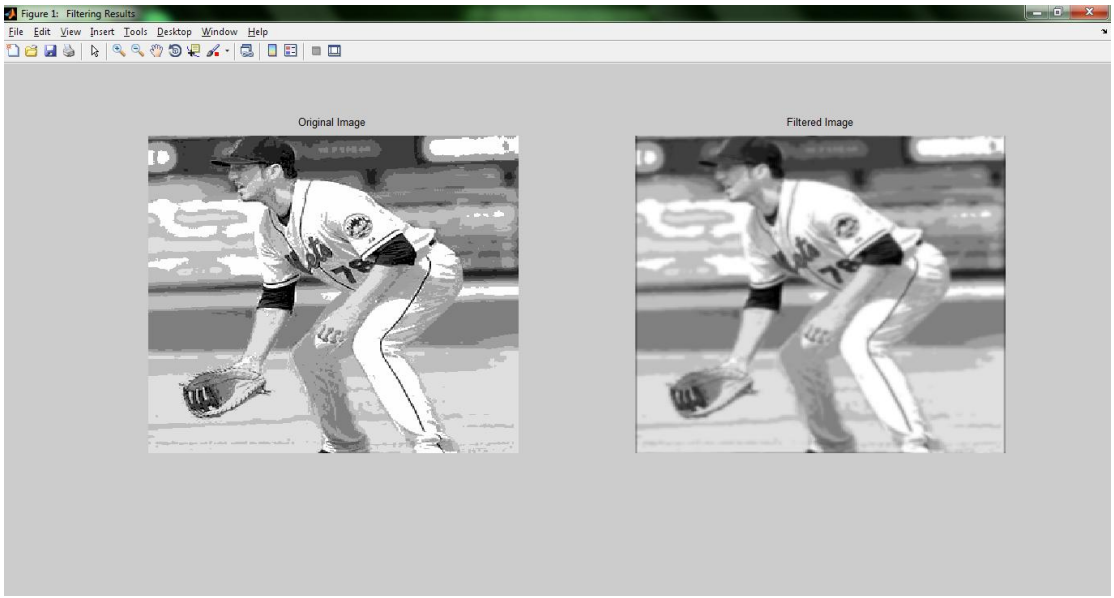

Figure 7. Original Image Vs. Filtered Image

7) Conclusion

This implementation of a Gaussian filter algorithm through the use of co-simulation can be very useful for developing more complex image and video processing systems. Xilinx's System Generator 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. Any system designed using System Generator is only useful if the system will work once programmed to the FPGA development board, and co-simulation is the perfect solution for testing purposes.

8) References

- Grant, Pascha. "Spartan-3A DSP FPGA Video Starter Kit Assembly: Getting Started with Demos"

- Xilinx's sysgenConv5x5 demo, which can be accessed at: {Xilinx Root Location}\11.1\DSP_Tools\nt\sysgen\examples\demos\sysgenConv5x5.mdl