ECE 480 Team 3

Grayscale Conversion of a Color Image Using Simulink and Xilinx Blocks

An Application Note

Emmett Kuhn
11/19/2010

Table of Contents

Executive Summary

This guide will take an image and through Matlab and Simulink programming blocks, convert the image from color to grayscale. It does this using Matlab programming to format the input, Simulink and Xilinx Programming blocks for the algorithm, and finally Matlab programming for the output.

Introduction

This Application Note is a guide to convert a single color image into grayscale using Xilinx ISE Design Suite 12 and Matlab/Simulink 7.10 (R2010a). It does this via Simulink and Xilinx programming blocks for the mathematical algorithm of grayscale. Converting a color image to grayscale can have many different advanced applications. Often it is the first step to multiple image processing actions to an image, sometimes even in full video. This guide is the software side of use in an embedded FPGA for such applications.

Keywords

SLB – Simulink Library Browser

FPGA – Field-Programmable Gate Array

ISE – Integrated Software Environment

RGB – Red, Green, Blue – A stream of data indicating the levels of Red, Green, and Blue in an image

Implementation

Getting Started

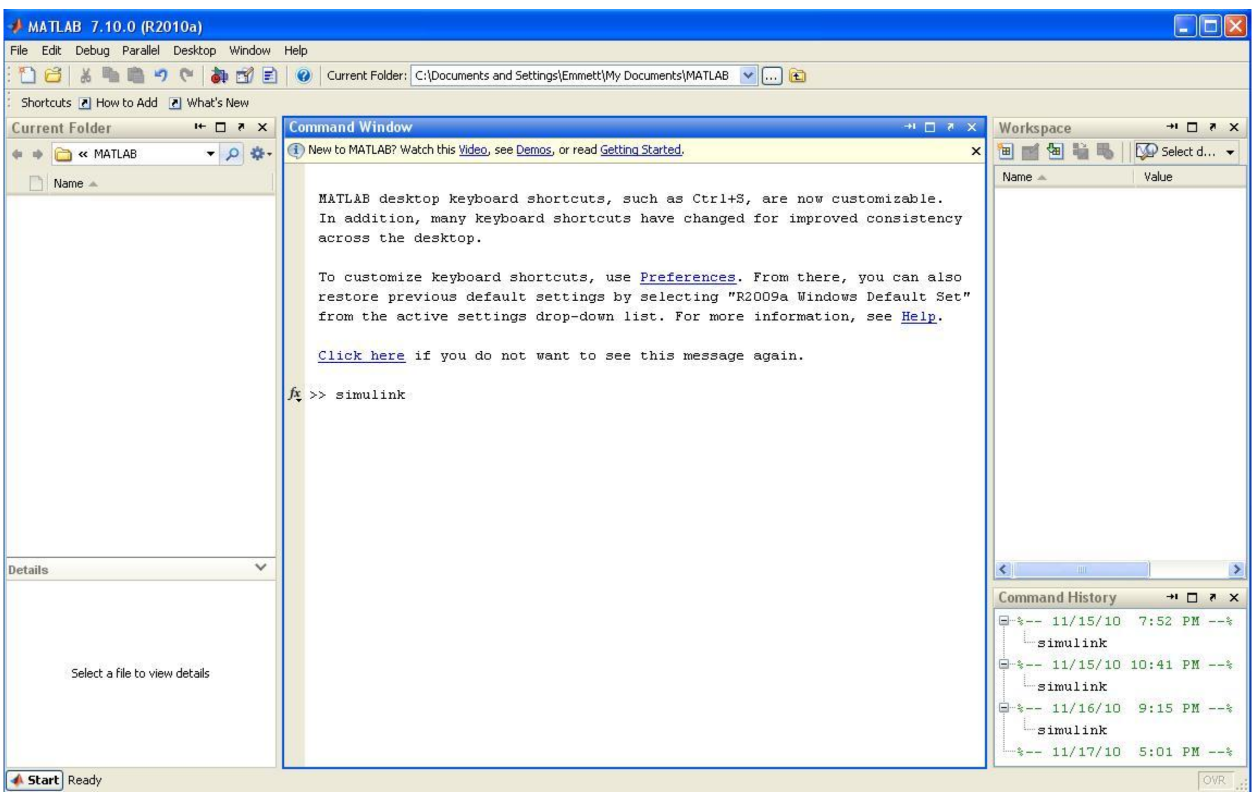
Three files must be saved to the intended directory for a successful grayscale conversion using Simulink and Xilinx blocks.

Save the files “sysgenColorConverter_imageData.mat”, “sysgenColorConverter_PreLoadFcn.m” and “sysgenColorConverter_StopFcn.m” into the specified workspace folder. These files are in Appendix A, Appendix B, and Appendix C. The former inherits the source image from sysgenColorConverter_imageData.mat and then allows that input to be made into Simulink blocks. The latter allows the output to be graphed into Matlab.

Setting Up Simulink Input Blocks

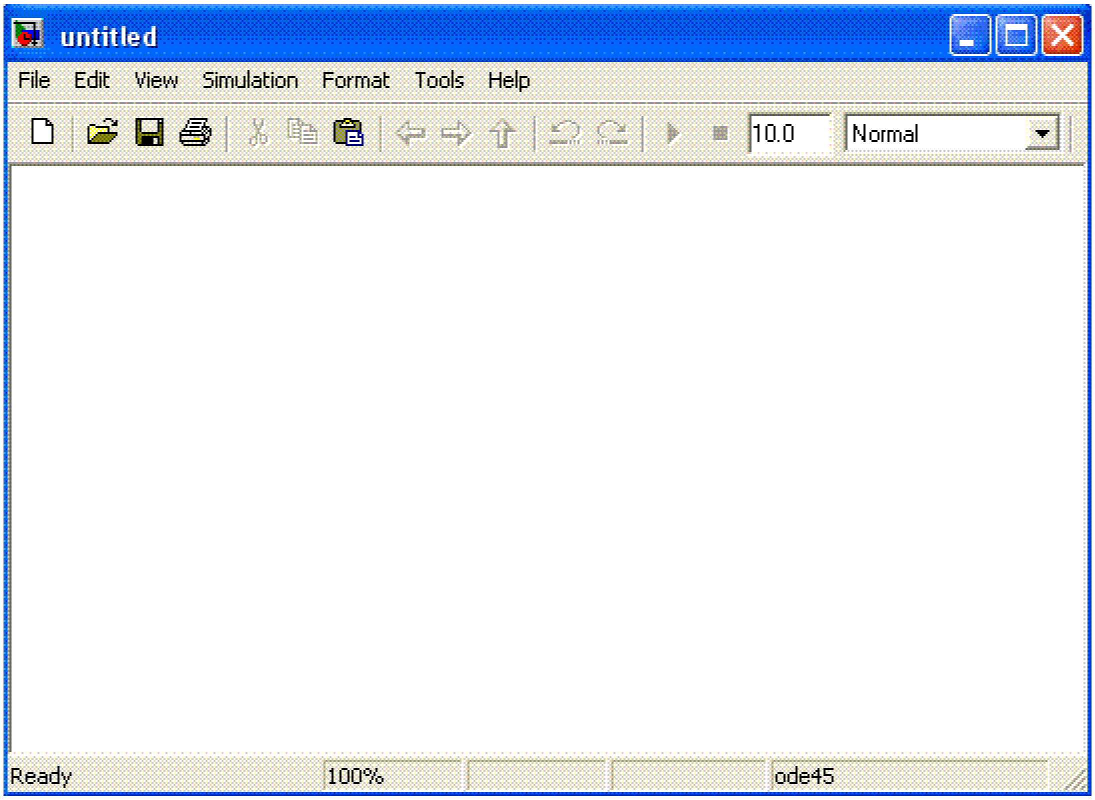
First, start Matlab.

In the Matlab Command Window, type “Simulink”



A window should pop up. This is the Simulink Library Browser (SLB). To use Simulink and Xilinx Blocks, drag them into the model from the SLB.

In the Simulink Library Browser which opens, click File -> New -> Model



First Simulink blocks must be created to inherit the Matlab output of the source file. This will be done by separating the output into separate RGB signals that are very important to grayscaling. Grayscaling works by adding weighted components of the Red, Green, and Blue signals of an image into a single desired linear luminance value.

In the SLB, browse to Simulink -> Sources on the left hand pane. Drag the “From Workspace” block into the model. Double click the model and set the following Parameters:

Data: redSignal

Sample time: 1

Uncheck Interpolate data

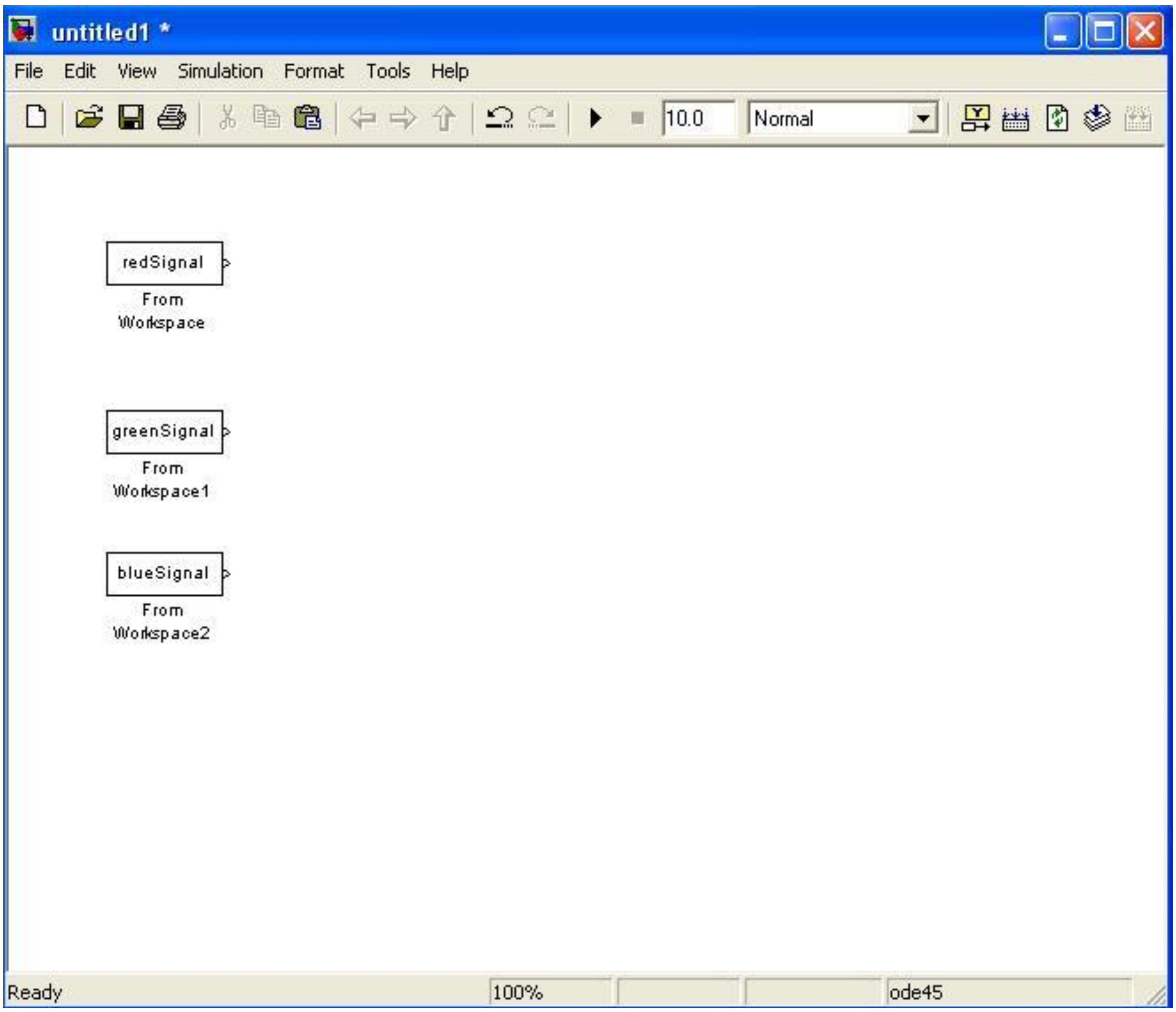
Uncheck Enable zero-crossing detection

Form output after final data value by: Setting to zero

Click “OK”

Repeat this process for two additional “From Workspace” blocks named “greenSignal” and “blueSignal” but otherwise have the same parameters as redSignal.

Model so far:



This would be a good time to save the model. Save it to the directory that the two .m files from the Appendix are saved. This is critical as the files reference each other so the files must be in the same directory.

Next, go back to SLB and navigate to "Xilinx Blockset" and then "Data Types." Then drag 3 of the yellow "Gateway In" blocks into your model. They will be used to navigate into Xilinx blocks to be implemented efficiently on the board.

Now, connect the RGB outputs to the Gateway ins as shown by clicking on the Image From File, then holding control and clicking on each Gateway In. This will create an arrow from one block to the other.

Apply the following settings for each gateway in by double clicking it:

Output Type: Signed (2's Comp)

Number of bits: 10

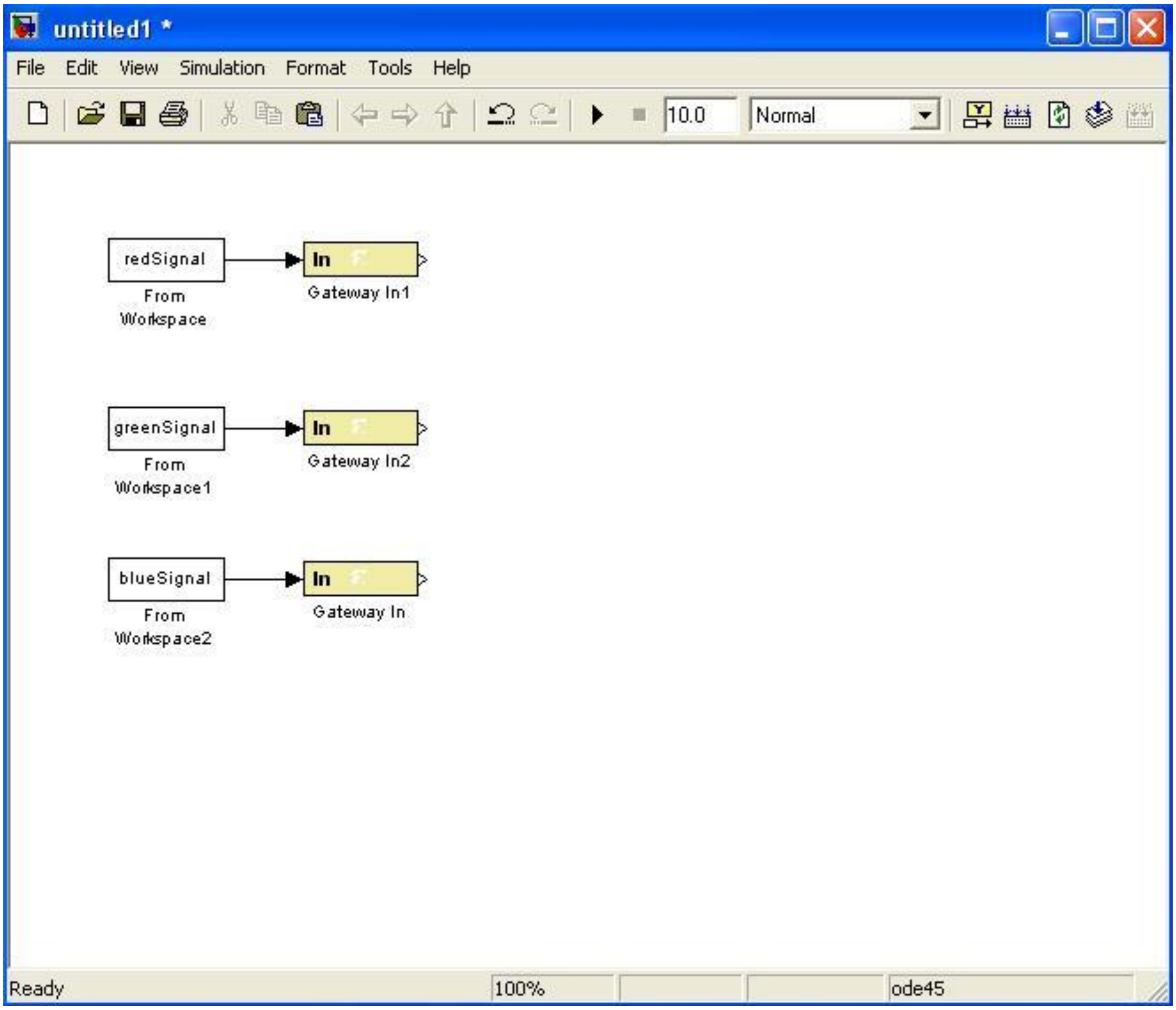
Binary Point: 0

Quantization: Round

Overflow: Saturate

Sample Period: 1

Model up to this point:



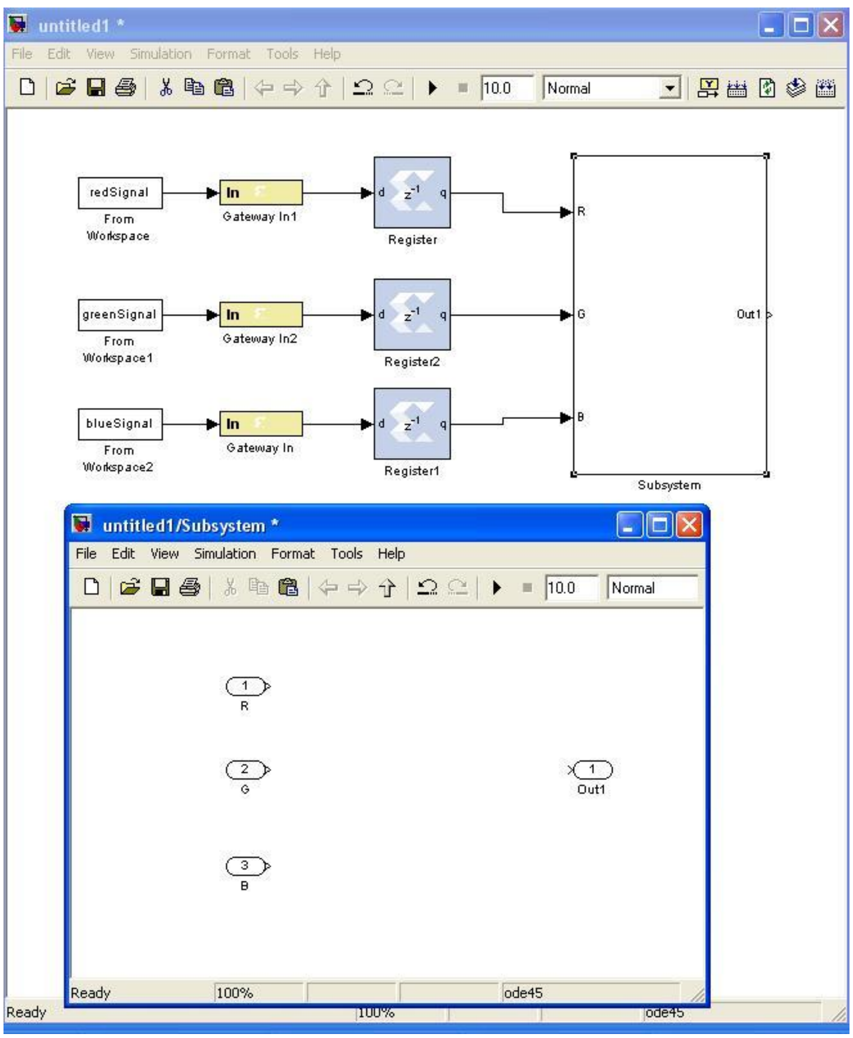
Next, Registers must be added. The Registers can be found under the "Xilinx blockset" then "Basic Elements. Drag 3 Registers into the model. Connect the outputs of the Gateway Ins into an individual register again creating the arrows from the output of the gateways to the inputs of the registers.

Creating the Subsystem

Now, to better organize our algorithm, a subsystem will be created. Navigate on the SLB to "Simulink" and then "Commonly Used Blocks." Drag the Subsystem into the model. Double click on the sub system. This opens up another model within your current model for better organization.

Within this new Subsystem, for the moment, there is one input and one output. This application requires three inputs, so select the input, select "copy" and then paste two additional inputs. Also rename these signals "R", "G", and "B" by double clicking their labels and editing them.

Model up to this point:



Next, multipliers need to be added to weight the RGB components and then summing those resultant numbers. In SLB, under the Xilinx Blockset -> Math, drag 3 “C Mult”’s or constant multipliers into the model.

The following settings should be applied in the “basic” tab for each multiplier’s properties:

RMult

Value: 0.3

Number of Bits: 12

Binary Point: 12

Latency: 3

GMult

Value: 0.59

Number of Bits: 12

Binary Point: 12

Latency: 3

BMult

Value: 0.11

Number of Bits: 12

Binary Point: 12

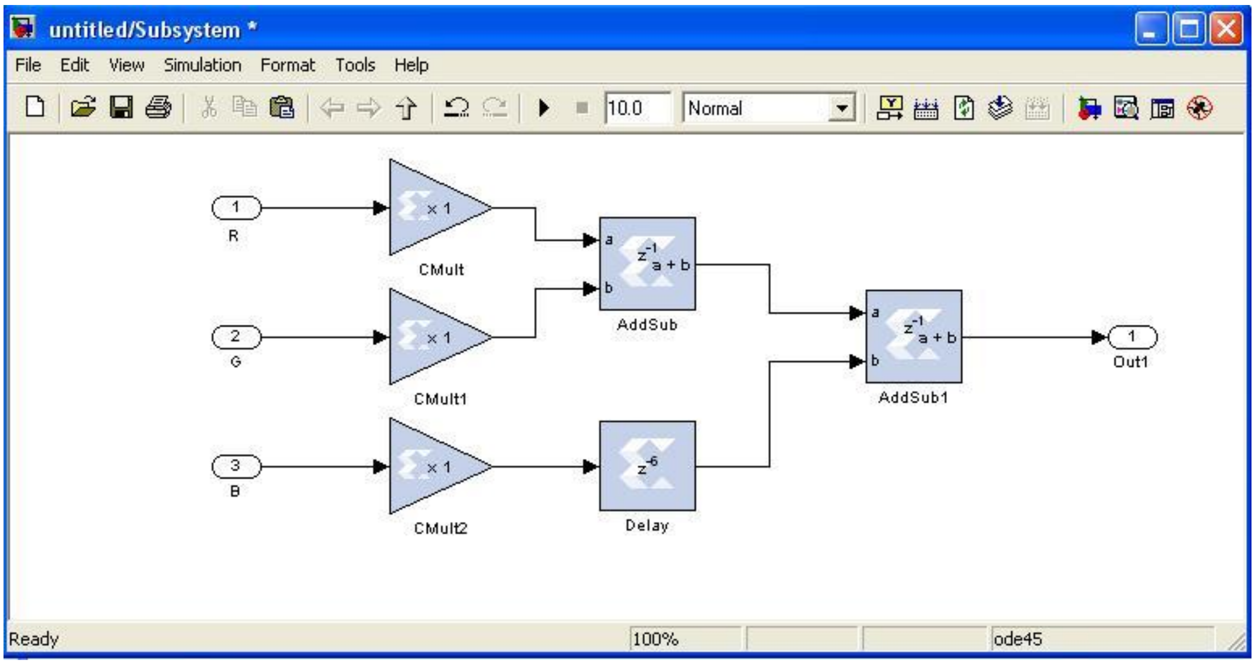
Latency: 3

Arrange them to receive the output of each of the R, G, and B inputs into the Multiplier’s inputs.

Now we need to sum the resultant RGB components with their new weights. Unfortunately, in Xilinx, a 3 input one output summing block is not available. We can, however, add two of the components, while delaying the third, and then add the resulting two inputs.

First, add two adders by navigating the SLB and dragging "AddSub" from Xilinx Blockset -> Math. Double click the AddSubs and be sure to set them to addition and their latency to "1".

Second, add a Delay block from Xilinx Blockset -> Memory and double click and set its latency to "6." This allows the addsubs to be in sync. Connect the blocks as shown:



This subsystem is now completed. Back to the main view of the model, the output of the subsystem must be reverted out of Xilinx blocks and back into Simulink to broadcast to a camera output.

Setting up the Output Blocks

First, the output will have to be converted via a Xilinx Register. This can be implemented in the SLB under "Xilinx Blockset" -> "Basic Elements." The input to this register will be the output of the subsystem. The output of the register will go to a gateway out, almost a mirror to the input into the subsystem. This Gateway out can be found in the same "Basic Elements" folder of the Simulink Library Browser.

Finally, the image must be outputted to a video source. The final block is found in "Simulink" -> "Sinks" as "To Workspace." This will be the final output and its input is the Gateway Out block. This output will be sent to the Matlab file "sysgenColorConverter_StopFcn.m" which will graph the output in Matlab.

Double click the "To Workspace" block and enter the following parameters:

Variable name: ySignal

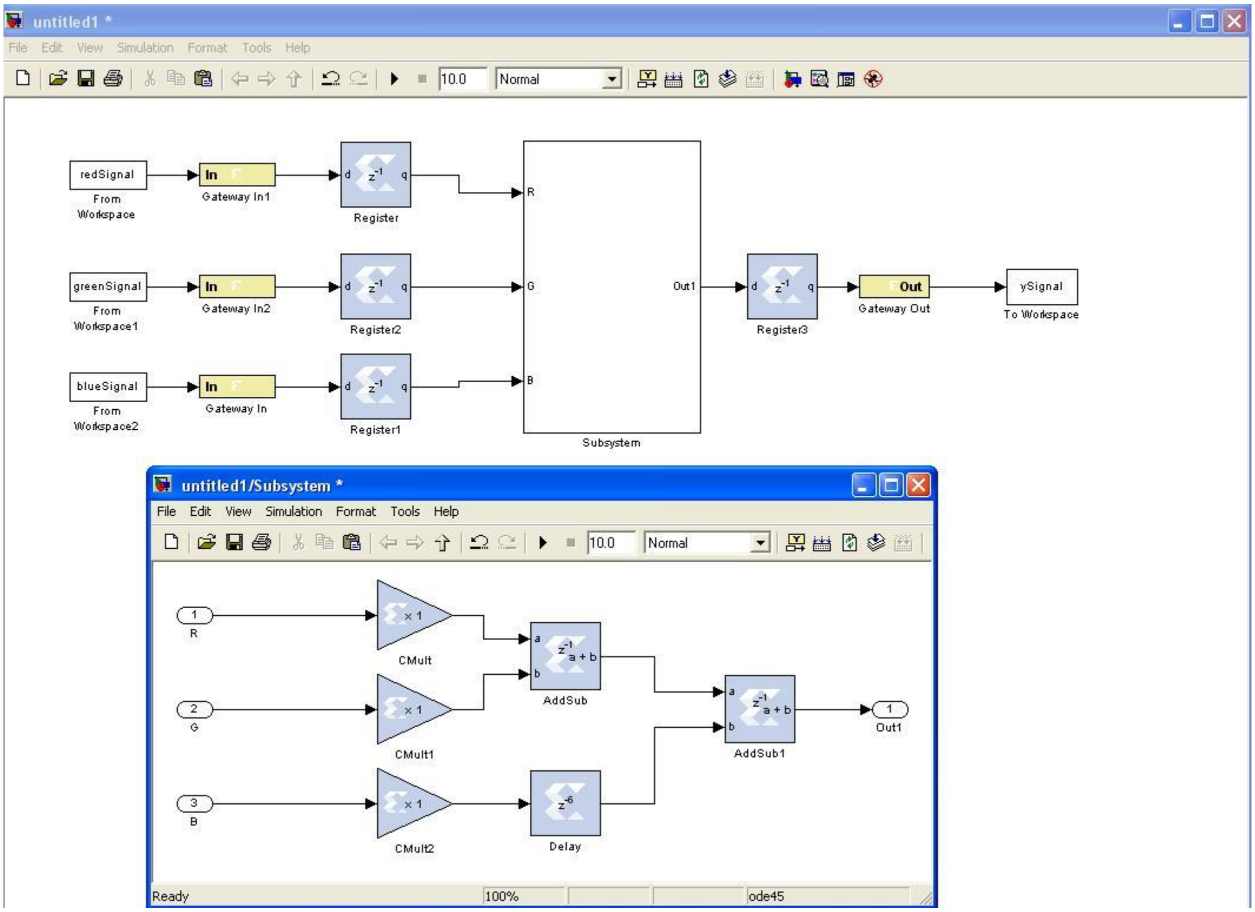
Limit data points to last: inf

Decimation: 1

Sample time: 1

Save format: Array

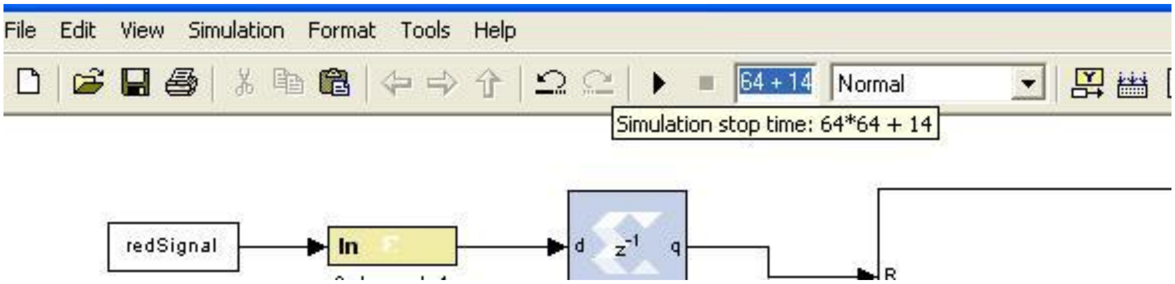
Shown are the final blocks with the subsystem expanded underneath:



Lastly, drag "System Generator" from Xilinx Blockset -> Basic Elements into the model. The location does not matter. The configuration of this block is discussed in depth in "Integrating Custom Logic in Camera Frame Demo" by Jeffrey Olsen.

Running the Model

To run the model we need to set the simulation time to "64*64 + 14." This can be done by typing in the box as shown:



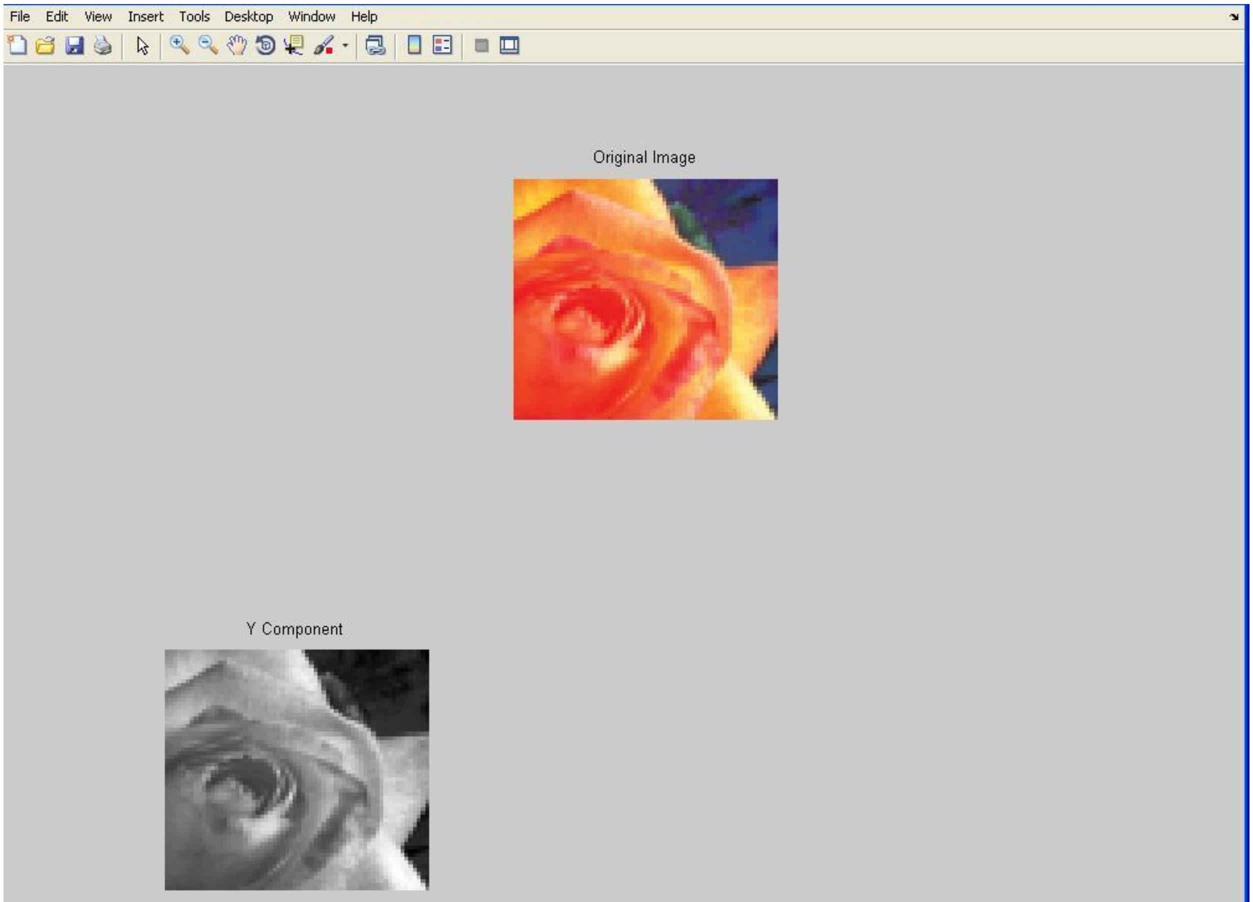
This is to insure a proper clock and to avoid syncing errors in the calculations.

Next, run both sysgenColorConverter_imageData.mat and sysgenColorconverter_PreLoadFcn.m in Matlab to generate the input source file.

Now run the model.

Finally, run the sysgenColorConverter_StopFcn.m file which will output the generated grayscale image.

Output Display:



References

"Integrating Custom Logic in Camera Frame Demo" by Jeffrey Olsen

http://www.egr.msu.edu/classes/ece480/capstone/fall10/group03/doc.html

Matlab/Simulink Version 7.10 Help

System Generator for DSP – Getting Started Guide by Xilinx

http://www.xilinx.com/support/sw_manuals/sysgen_gs.pdf

System Generator Demonstrations – Provided via System Generator Install

Appendix A - sysgenColorConverter_imageData.mat

```
function importfile(fileToRead1)
%IMPORTFILE(FILETOREAD1)
%  Imports data from the specified file
%  FILETOREAD1:  file to read

%  Auto-generated by MATLAB on 19-Nov-2010 11:59:20

% Import the file
newData1 = load('-mat', fileToRead1);

% Create new variables in the base workspace from those fields.
vars = fieldnames(newData1);
for i = 1:length(vars)
    assignin('base', vars{i}, newData1.(vars{i}));
end
```

Appendix B – sysgenColorConverter_PreLoadFcn.m

```
% sysgenColorConverter_imageData is a matrix loaded by the model from a .mat
% file before this script is run.

% We now separate out the red green and blue components from the image.

redSignal = sysgenColorConverter_imageData(:,:,1);
greenSignal = sysgenColorConverter_imageData(:,:,2);
blueSignal = sysgenColorConverter_imageData(:,:,3);

NPixels = size(redSignal,1) * size(redSignal,2);

% turn them into vectors (they were arrays):

redSignal = reshape(redSignal,1,NPixels);
greenSignal = reshape(greenSignal,1,NPixels);
blueSignal = reshape(blueSignal,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

redSignal = [ double(0:NPixels-1)'  double(redSignal)'];
greenSignal = [ double(0:NPixels-1)'  double(greenSignal)'];
blueSignal = [ double(0:NPixels-1)'  double(blueSignal)'];
```

Appendix C – sysgenColorConverter_StopFcn.m

```
if (exist('ySignal','var') & exist('NPixels','var'))

  if ((~isempty(ySignal)))

    designLatency = find(ySignal>0 & ySignal<1024);
    if (isempty(designLatency))
      return;
    end
    designLatency = designLatency(1);

    imageSize = size(ySignal);

    if (imageSize(1) >= designLatency+NPixels-1)

      % The initial output of the design (pipeline stages)
      % will be zeros and NaNs.
      % For plotting, we just assume that the first non-zero, non-NaN
      % output is the represents the first pixel.
      h = figure;
      clf;
      set(h,'Name','  Color Conversion Results');
      set(h,'Position',[100 50 1000 800]);

      ySignalN = ySignal(designLatency:designLatency+NPixels-1);
      ySignalN = ySignalN - (min(ySignalN));
      ySignalN = ySignalN / (max(ySignalN));

      subplot(2,3,2);
      image(sysgenColorConverter_imageData), ...
        axis equal, axis square, axis off, title 'Original Image';

      subplot(2,3,4);
      image(reshape([ySignalN ySignalN ySignalN], ...
                    sqrt(NPixels),sqrt(NPixels),3)), ...
          axis equal, axis square, axis off, title 'Y Component';


      prSignalN = prSignal(designLatency:designLatency+NPixels-1);
      prSignalN = prSignalN - (min(prSignalN));
      prSignalN = prSignalN / (max(prSignalN));
      subplot(2,3,5);
      image(reshape([prSignalN prSignalN prSignalN], ...
                    sqrt(NPixels),sqrt(NPixels),3)), ...
```

```
          axis equal, axis square, axis off, title 'PR Component';


      pbSignalN = pbSignal(designLatency:designLatency+NPixels-1);
      pbSignalN = pbSignalN - (min(pbSignalN));
      pbSignalN = pbSignalN / (max(pbSignalN));
      subplot(2,3,6);
      image(reshape([pbSignalN pbSignalN pbSignalN], ...
                    sqrt(NPixels),sqrt(NPixels),3)), ...
          axis equal, axis square, axis off, title 'PB Component';
    end
  end
end
```