UG0640

User Guide

Bayer Conversion

February 2018

Microsemi

Power Matters.™

Contents

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 4.0

In revision 4.0 of this document, the Resource Utilization section and the Resource Utilization Report were updated. For more information, see Resource Utilization (see page 15).

1.2 Revision 3.0

In revision 3.0 of this document, the Testbench section was updated. For more information, see Testbench (see page 7).

1.3 Revision 2.0

The following is a summary of changes in revision 2.0 of this document.

- Updated the figures such as Block Diagram of Bayer Conversion and Timing Diagram of Bayer Conversion. For more information, see Block Diagram of Bayer Conversion (see page 4) and Timing Diagram of Bayer Conversion (see page 7).
- Updated tables such as Inputs and Outputs of Bayer Conversion, Configuration Parameters, and Resource Utilization Report. For more information, see Inputs and Outputs of Bayer Conversion (see page 5), Configuration Parameters (see page 5), and Resource Utilization Report (see page 15).
- Added the Testbench section as per SAR 75869. For more information, see Testbench (see page 7).

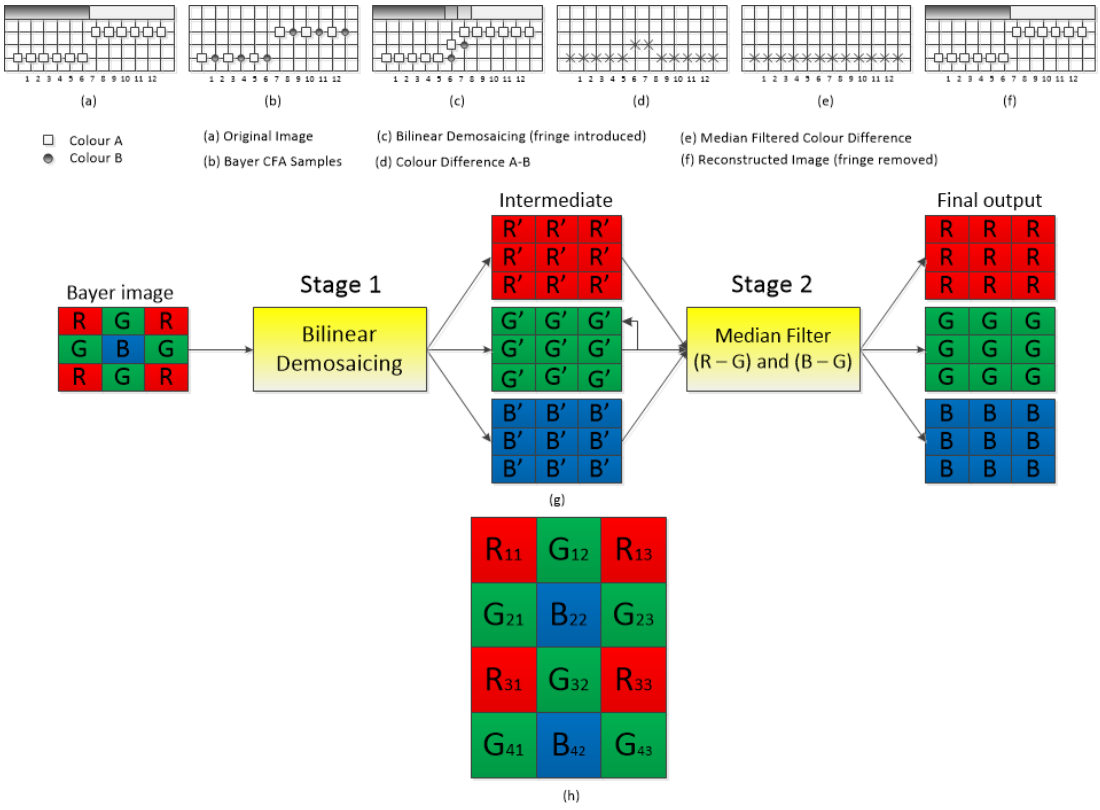
1.4 Revision 1.0

Revision 1.0 is the first publication of this document.

2 Introduction

The Freeman demosaicing algorithm is a two-stage process that combines bilinear interpolation and median filtering. To minimize the zippering artifacts introduced by bilinear demosaicing, Freeman algorithm utilizes color difference such as, red minus green and blue minus green, for median filtering. The filtered differences are then added back to the green plane to obtain the final red and blue planes. In this method, fringes at the edges of different color areas can be eliminated as shown in the following figure, which takes a line with two color components in the Bayer color filter array (CFA).

Figure 1 • Illustration of Freeman Demosaicing Algorithm



The following table describes the illustration of Freeman demosaicing algorithm.

Table 1 • Description of Illustration of Freeman Demosaicing Algorithm

Legend	Description
(a)	Original image with two color components. This image contains a sharp edge between two different areas. The vertical axis represents the color component intensities.
(b)	Sampled color information captured by the charge-coupled device (CCD) with Bayer CFA.
(c)	Image reconstructed with the noticeable color fringe between two color areas at the positions of pixel 6 and 7, after linear demosaicing.
(d) and (e)	If the color value difference (color A minus color B) is filtered by a median filter with a kernel size 3, the color difference can be eliminated.
(f)	Original image reconstructed without the color fringe existing in bilinear demosaicing.
(g)	Freeman algorithm architecture consisting of bilinear demosaicing and median filtering.
(h)	Bayer pattern image.

The first stage estimates the missing color components for each pixel by bilinear interpolation algorithm as shown in the following formulas, which may change according to the different pixel layouts in the Bayer pattern as shown in the g block in Illustration of Freeman Demosaicing Algorithm.

$R_{22} = (R_{11} + R_{13} + R_{31} + R_{33})/4$

$G_{22} = (G_{12} + G_{21} + G_{32} + G_{23}) / 4$

$B_{22} = B_{22}$

$R_{32} = (R_{31} + R_{33})/ 2$

$G_{32} = G_{32}$

$B_{32} = (B_{22} + B_{42}) / 2$

After the first stage, three intermediate color planes are obtained. In the second stage, the color value differences red minus green (R – G) and blue minus green (B – G) are median filtered. Median filter is widely used in image processing as a non-linear digital filtering technique. The main idea for median filter is running through the input signal entry by entry and replacing each entry with the median of its neighboring entries. The pattern of the neighboring entry is called filter window, which slides over the entire signal, entry by entry. While processing a 2-D image, the formula can be represented as:

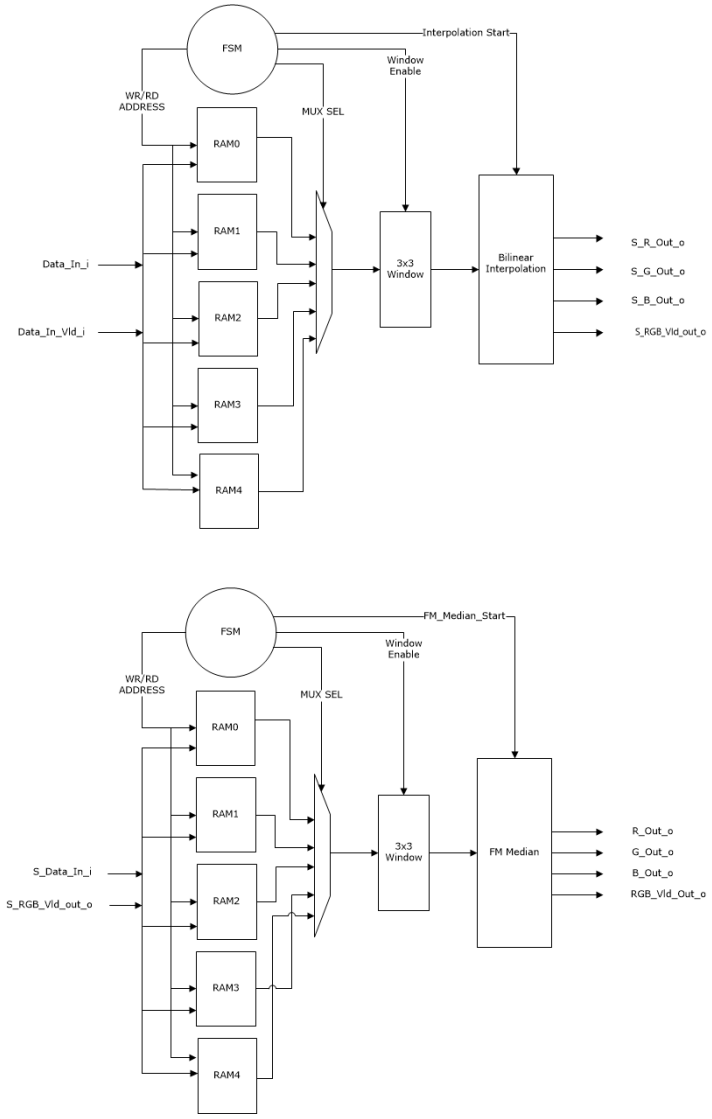
$Y_{ij} = \text{Med} \{X_{ij}\}$

where, X_{ij} = Set of pixels in the filter window.
Median filter is employed to reduce the fringing effects in images by removing sudden jumps in the hue. In Freeman demosaicing, the median filter is based on using a shifting window over the image and calculating the median value of pixels within the window for the output. Since the demosaicing artifacts are generally manifested as small chromatic splotches, median filtering the R – G and B – G color planes tends to eliminate the artifacts efficiently. The final interpolated image is generated by adding the median filtered color difference to the corresponding pixel value. For example, the red value in position 32 of the h block in Figure 1, page 2 is obtained by adding the filtered R – G value to the sampled green value at this position. Estimated results are used only at positions where the original sampled color pixel is different. For example, it is not necessary to estimate blue value at position 22.

3 Hardware Implementation

The following figure shows the block diagram of Bayer conversion.

Figure 2 • Block Diagram of Bayer Conversion



The previous figure shows the implementation of Bayer conversion module divided into the following modules.

- CFA to RGB conversion
- Freeman median filter

CFA to RGB conversion has five line buffers. A delay of at least one clock cycle is expected between two lines of input. The incoming Bayer image is stored in the line buffers starting from the first line of the Bayer image from the camera. The value of each output pixel O(x, y) is computed using the nine pixels of the image I(x,y) that are inside the 3 × 3 mask with center at M(x, y). The 3 × 3 window logic enable signal is set when the third pixel of the third line is written into the line buffer. The windowing logic reads the 3 × 3 window from the first three line buffers (RAM0, RAM1, and RAM2) and feeds to the bilinear interpolation block, which computes the RGB pixels from Bayer pixel I(x,y). The window then slides right to compute the value of the next pixel in the line. After computing RGB for the first line, the windowing logic slides from RAM0, RAM1 and RAM2 to RAM1, RAM2, and RAM3 to compute RGB data for the second line and goes on till the last line of the Bayer image. After bilinear interpolation, all the pixels are sent out with an output valid signal and along with interpolated RGB data.

In the Freeman median module, the 3 × 3 windowing logic remains same as the CFA to RGB module. However, R – G and B – G are calculated from read out 3 × 3 window and the calculated median is added to R – G and B – G to get the actual filtered value from the interpolated RGB.

3.1 Inputs and Outputs

The following table shows the input and output ports of Bayer conversion.

Table 2 • Inputs and Outputs of Bayer Conversion

Signal Name	Direction	Width	Description
RESET_n_I	Input	-	Active low asynchronous reset signal to design
SYS_CLK_I	Input	-	System clock
Data_In_i	Input	[g_DATAWIDTH – 1 : 0]	Input data
Data_In_Vld_i	Input	-	Set when input data is valid
R_Out_o	Output	[(g_DATAWIDTH) – 1 : 0]	Provides Red plane output
G_Out_o	Output	[(g_DATAWIDTH) – 1 : 0]	Provides Green plane output
B_Out_o	Output	[(g_DATAWIDTH) – 1 : 0]	Provides Blue plane output
RGB_Vld_Out_o	Output	-	Set when output data is valid register

3.2 Configuration Parameters

The following table shows the description of the configuration parameters used in the hardware implementation of Bayer conversion. These are generic parameters and can be varied as per the requirement of the application.

Table 3 • Configuration Parameters

Signal Name	Description
g_DATAWIDTH	Width of the data I/O
g_X_RES_WIDTH	Horizontal resolution bit width
g_DISPLAY_RESOLUTION	Horizontal resolution
g_VERT_DISPLAY_RESOLUTION	Vertical resolution

3.3 FSM Implementation

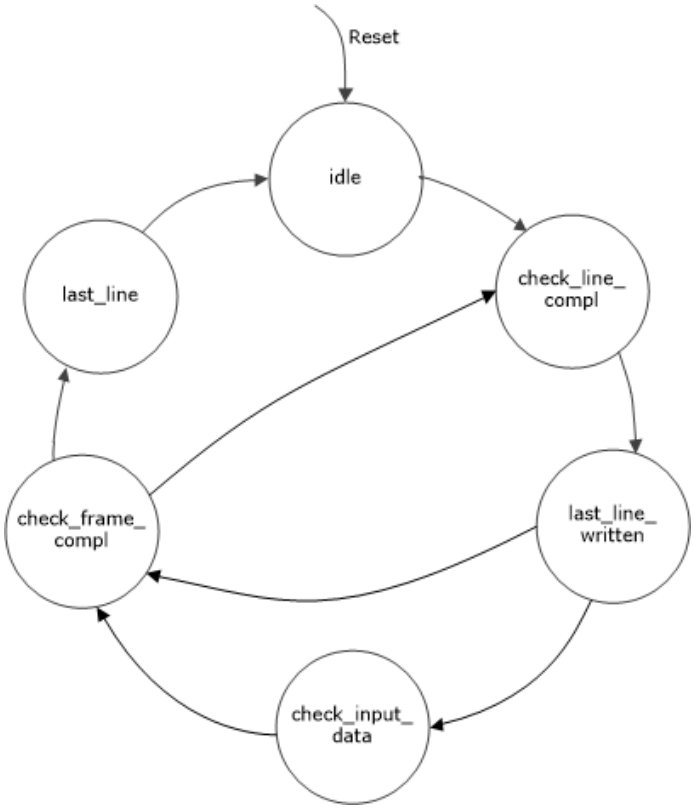
The finite state machine (FSM) of Bayer conversion in the current implementation has the following states.

- idle: After the module is reset, the FSM is in idle state and waits for the third pixel of the third line of the image to be read to move to the check_line_compl state.
- check_line_compl: FSM waits for the output pixel count to be equal to the display resolution and then moves to the last_line_written state.

- last_line_written: FSM waits for the last input line. If the last line is written, it proceeds to check_frame_compl, else it proceeds to check_input_data.
- check_input_data: FSM waits for the fourth pixel of the last line to be written into the line buffer and then moves to the check_frame_compl state.
- check_frame_compl: FSM waits for the output line count to be equal to the vertical resolution and to move to the last_line state. If the output line is not the last line, it goes to check_line_compl state.
- last_line: FSM waits for the last output pixel of the last line and then moves to idle state.

The following figure shows the FSM implemented.

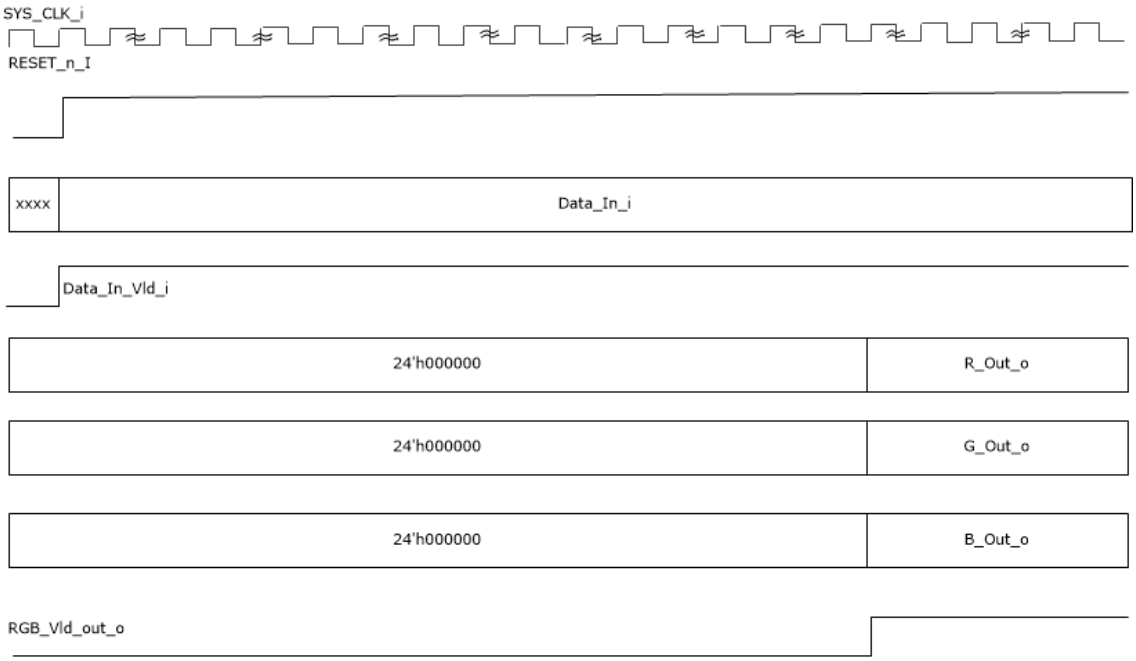
Figure 3 • Bayer Conversion FSM



3.4 Timing Diagrams

The following figure shows the timing diagram of Bayer conversion.

Figure 4 • Timing Diagram of Bayer Conversion



3.5 Testbench

A testbench is provided to check the functionality of Bayer Conversion core. The following table shows the parameters that can be configured according to application.

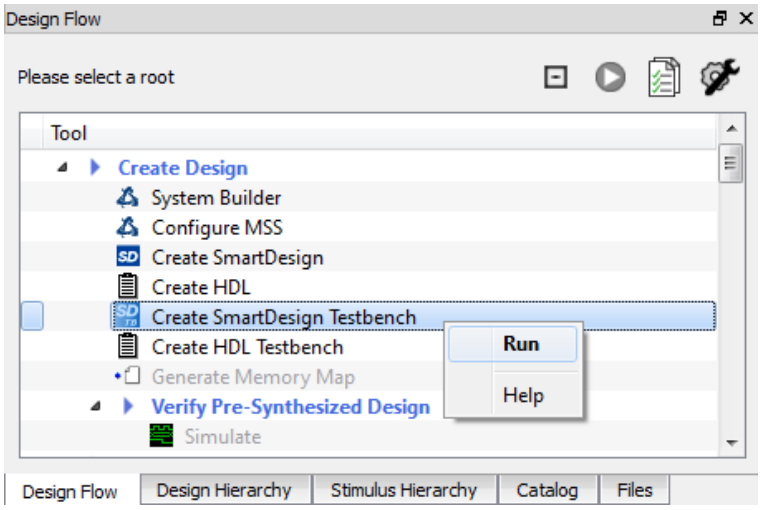
Table 4 • Testbench Configuration Parameters

Name	Description
CLKPERIOD	Clock Period
g_DATAWIDTH	Width of the data I/O
g_X_RES_WIDTH	Horizontal resolution bit width
g_DISPLAY_RESOLUTION	Horizontal resolution
g_VERT_DISPLAY_RESOLUTION	Vertical resolution
WAIT	Number of clock cycles delay between transmission of two input lines
IMAGE_FILE_NAME	Input (image) file name

The following steps describe how to simulate the core using the testbench.

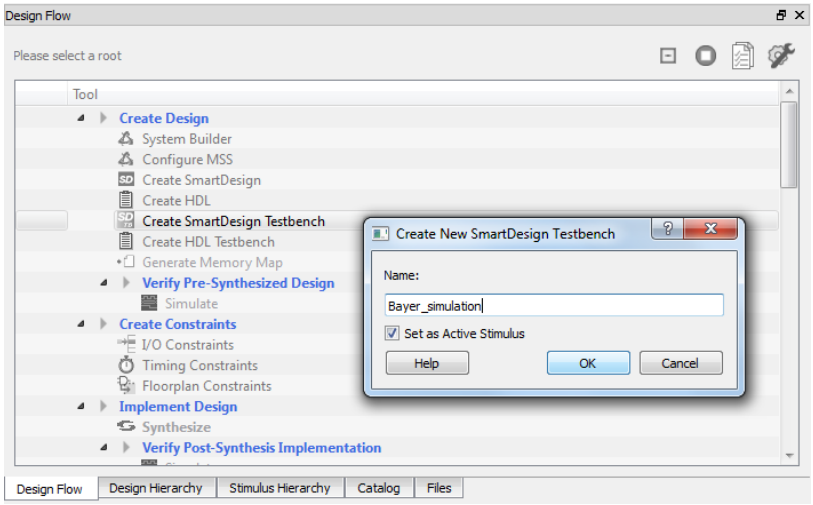
1. In the **Libero SoC Design Flow** window, expand **Create Design**, and double-click **Create SmartDesign Testbench**, or right-click **Create SmartDesign Testbench** and click **Run** as shown in the following figure.

Figure 5 • Creating SmartDesign Testbench



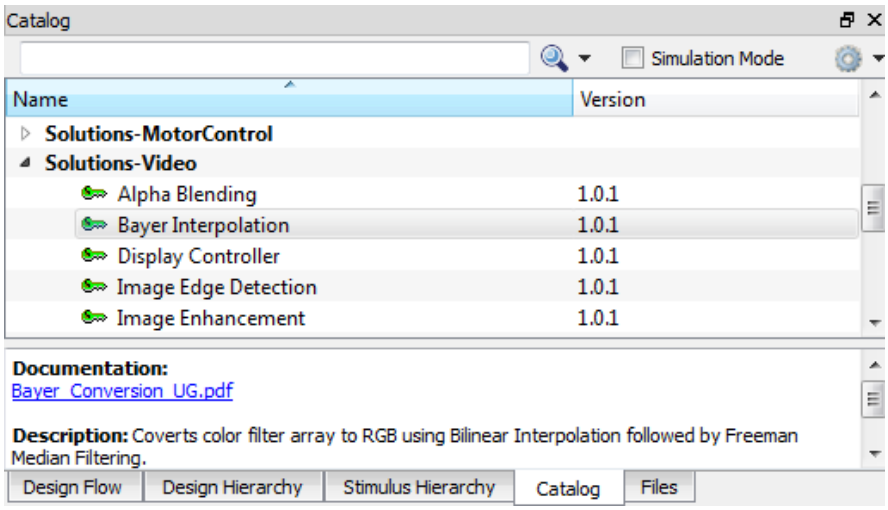
2. Enter a name for the SmartDesign testbench, and click **OK**. SmartDesign testbench is created, and a canvas appears to the right of the Design Flow pane.

Figure 6 • Create New SmartDesign Testbench Dialog



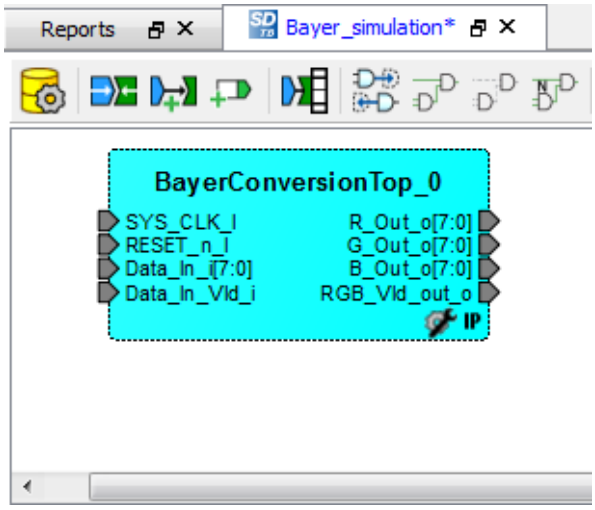
3. In the **Libero SoC Catalog** (**View** > **Windows** > **Catalog**), expand **Solutions-Video**, and drag the Bayer IP core onto the SmartDesign testbench canvas.

Figure 7 • Bayer Interpolation Core in Libero SoC Catalog



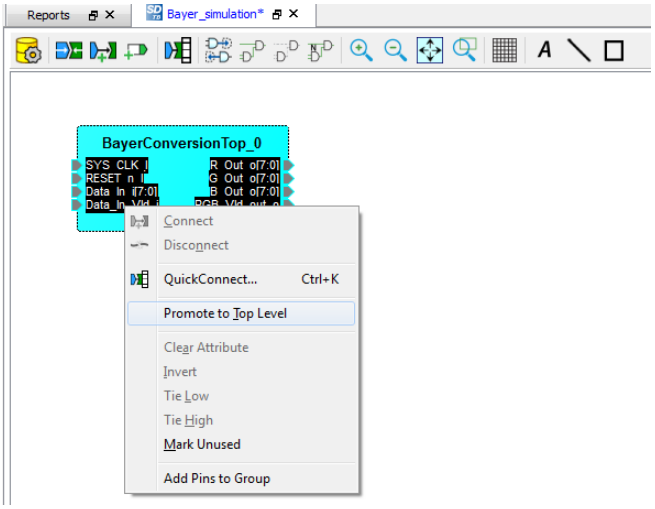
The core appears on the canvas as shown in the following figure.

Figure 8 • Bayer Interpolation Core on SmartDesign Testbench Canvas



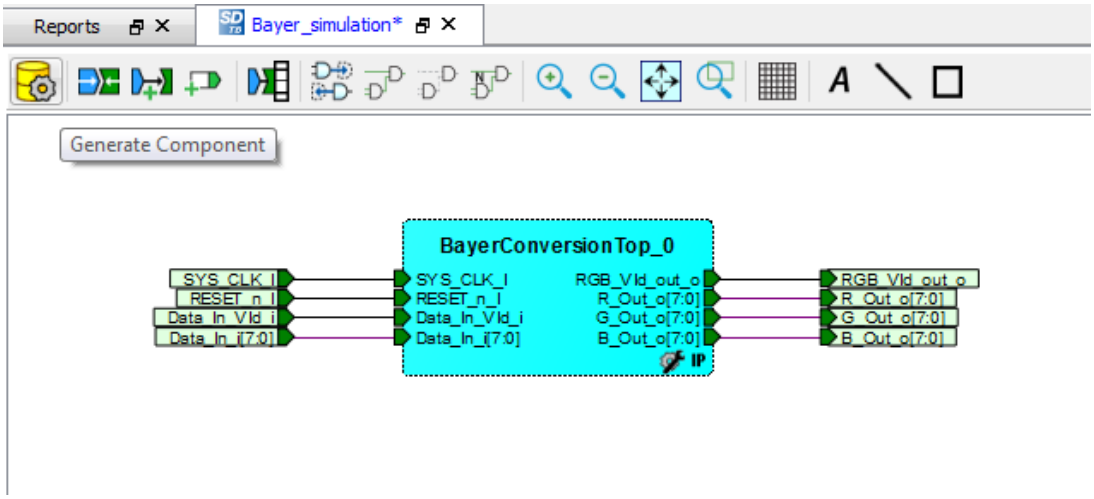
4. Select all the ports of the core, right-click, and select **Promote to Top Level**as shown in the following figure. The ports are promoted to the top level.

Figure 9 • Promote to Top Level



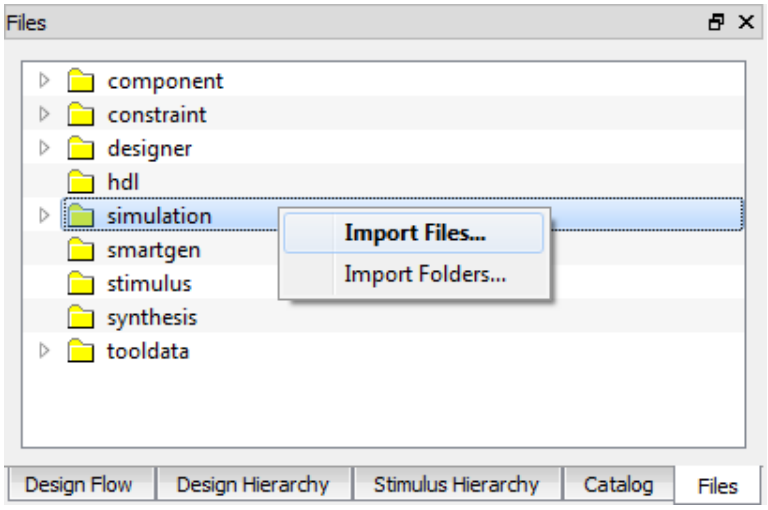
5. Click **Generate Component** from the SmartDesign toolbaras shown in the following figure. The SmartDesign component is generated.

Figure 10 • Generating Bayer Component with Ports Promoted to Top Level



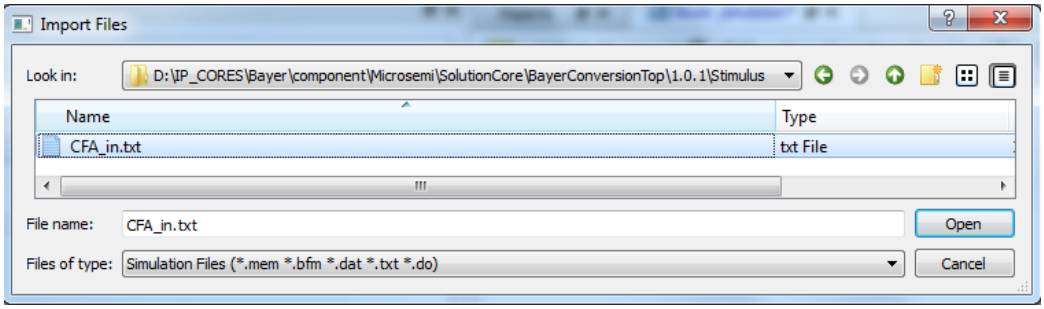
6. From Files window, right-click **simulation** and click **Import Files**as shown in the following figure.

Figure 11 • Import Files



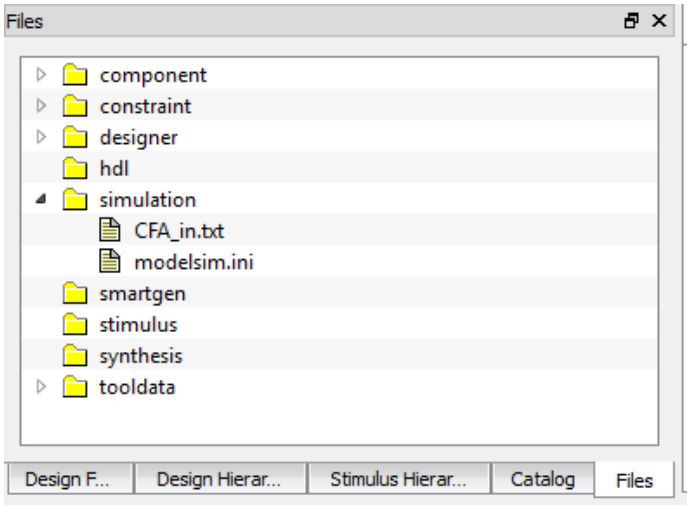
7. Select the file to import and click **Open** to import the image stimulus file. A sample `CFA_in.txt` file (CFA text equivalent of sample image) is provided with the testbench at the following path.
`..\Project_name\component\Microsemi\SolutionCore\BayerConversionTop\2.0.0\Stimulus`
To import a different image, browse the folder that contains the desired image file, and click **Open**.

Figure 12 • Selecting the File to Import



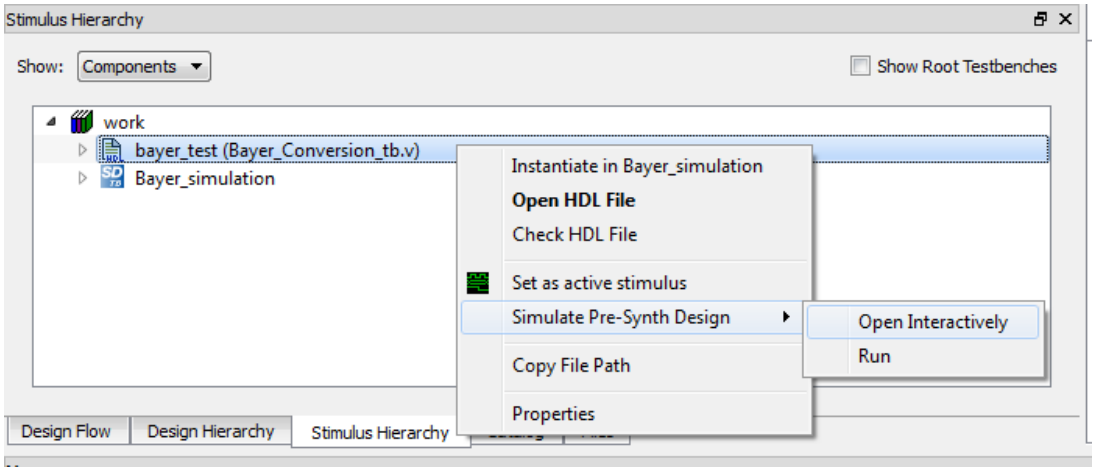
The imported file is listed under simulation as shown in the following illstration.

Figure 13 • Imported File



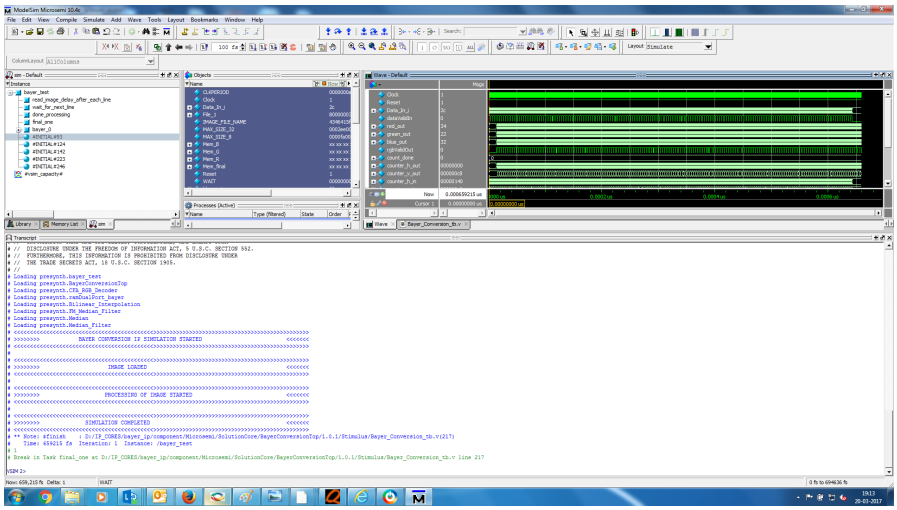
8. In the **Stimulus Hierarchy** window, right-click **bayer_test** (**Bayer_Conversion_tb.v**) testbench file and select Simulate **Pre-Synth Design** > **Open Interactively**. The core is simulated for one frame.

Figure 14 • Simulating Pre-Synthesis Design



The ModelSim tool appears with the testbench file loaded into itas shown in the following figure.

Figure 15 • ModelSim Simulation Window



If the simulation is interrupted because of the runtime limit specified in the DO file, use the `run -all` command to complete the simulation. After the simulation is completed, the testbench output image file appears in the `Files/simulation` folder.

3.6 Simulation Results

The following section provides simulation result. The following illustration shows the input Bayer image.

Figure 16 • Input Bayer Image

The following illustration shows the output RGB image.

Figure 17 • Output RGB Image

3.7 Resource Utilization

Bayer conversion is implemented on the SmartFusion®2 system-on-chip (SoC) field programmable gate array (FPGA) device (M2S150T-1152 FC package) and PolarFire FPGA (MPF300TS_ES - 1FCG1152E package). The following figure shows the resource utilization report after synthesis.

Table 5 • Resource Utilization Report

Resource	Usage
DFFs	1884
4-Input LUTs	3839
MACC	0
RAM1Kx18	20
RAM64x18	0

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

50200640