LogiCORE IP Color Correction Matrix v2.0

DS720 July 23, 2010 **Product Specification**

Introduction

The Xilinx LogiCORE™ IP Color Correction Matrix core is a 3 x 3 programmable coefficient matrix multiplier with offset compensation. This core can be used for color correction operations such as adjusting white balance, color cast, brightness, and/or contrast in an RGB image.

The core efficiently uses the 18x18 bit multipliers, adders and registers provided in XtremeDSP™ slices in Spartan®-3A DSP, Spartan-6, Virtex®-5, and Virtex-6 devices resulting in high performance and optimal resource usage.

Features

- Programmable matrix coefficients
- Selectable processor interface
  - EDK pCore Interface
  - General Purpose Processor Interface
  - Constant Interface
- CMY input to RGB output color conversion
- Configurable 8-, 10-, and 12-bit input and output
- Independent clipping and clamping control
- Delay match support for up to three sync signals

Applications

- Pre-processing block for image sensors
- Post-processing core for image data adjustment
- Video surveillance
- Video Conferencing
- Machine Vision

LogiCORE IP Facts Table

Core Specifics					
Supported Device Family[1]	Spartan-3A DSP, Spartan-6, Virtex-5, Virtex-6				
Supported User Interfaces	General Processor Interface, EDK PLB 4.6, Constant Interface				
Supported Operating Systems	Windows XP Professional 32-Bit/64-bit, Windows Vista Business 32-Bit/64-bit, Red Hat Enterprise Linux WS v4.0 32-bit/64-bit, Red Hat Enterprise Desktop v5.0 32-bit/64-bit (with Workstation Option), SUSE Linux Enterprise (SLE) desktop and server v10.1 32-bit/64-bit				
	Resources[2]				**Frequency**
Configuration	**LUTs**	**FFs**	**DSP Slices**	**Block RAMs**	**Max. Freq.**
Spartan-3A DSP	33	71	9	0	200
Spartan-6	71	41	9	0	201
Virtex-5	71	33	9	0	371
Virtex-6	71	41	9	0	371
Provided with Core					
Documentation	Product Specification				
Design Files	Netlists, EDK pCore files, C drivers				
Example Design	Not Provided				
Test Bench	Not Provided				
Constraints File	Not Provided				
Simulation Models	VHDL or Verilog Structural, C, and MATLAB™				
Tested Design Tools					
Design Entry Tools	CORE Generator™ tool, Xilinx Platform Studio (XPS)				
Simulation	ModelSim v6.5c, Xilinx® ISim 12.2				
Synthesis Tools	XST 12.2				
Support					
Provided by Xilinx, Inc.					

1. For a complete listing of supported devices, see the release notes for this core.
2. Resources listed here are for 8-bit input, 8-bit output width configurations. For more complete device performance numbers, see "Core Resource Utilization and Performance".

Overview

There are many variations that cause difficulties in accurately reproducing color in imaging systems. These include:

- Spectral characteristics of the optics (lens, filters)
- Lighting source variations like daylight, fluorescent, or tungsten
- Characteristics of the color filters of the sensor

The Color Correction Matrix provides a method for correcting the image data for these variations. This fundamental block operates on either CMY or RGB data, and processing is “real-time” as a pre-processing hardware block.

As an example, following one of the three color channels through an imaging system from the original light source to the processed image helps understand the functionality of this core.

The blue color channel is a combination of the blue photons from the scene, multiplied by the relative response of the blue filter, multiplied by the relative response of the silicon to blue photons. However, the filter and silicon responses might be quite different from the response of the human eye, so blue to the sensor is quite different from blue to a human being.

This difference can be corrected and made to more closely match the blue that is acceptable to human vision. The Color Correction Matrix core multiplies the pixel values by some coefficient to strengthen or weaken it, creating an effective gain. At the same time a mixture of green or red can be added to the blue channel. To express this processing mathematically, the new blue (Bc) is related to the old blue (B), red (R), and green (G) according to:

$$Bc = K1 \times R + K2 \times G + K3 \times B$$

where K1, K2, and K3 are the weights for each of the mix of red, green, and blue to the new blue.

Extending this concept, a standard 3 x 3 matrix multiplication can be applied to each of the color channels in parallel simultaneously. This is a matrix operation where the weights define a color-correction matrix. In typical applications, color-correction also contains offset compensation to ensure black [0,0,0] levels are achieved.

$$\begin{bmatrix} R_c \\ G_c \\ B_c \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} \qquad \text{Equation 1}$$

As shown in the matrix operation, the input pixels are transformed to a set of corrected output pixels. This can be a very useful function configured as a static application; however, the programmability of the coefficients and offset values allows this function to adapt to changing lighting conditions based on a separate control loop.

Processor Interfaces

The Color Correction Matrix core supports the following three processor interface options:

- EDK pCore Interface
- General Purpose Processor Interface
- Constant Interface

EDK pCore Interface

Many imaging applications have an embedded processor that can dynamically control the parameters within the core to correct for variations within the lighting conditions of the scene being captured. The user can select an EDK pCore interface, which creates a pCore that can be added to an EDK project as a hardware peripheral. This pCore provides a memory-mapped interface for the programmable registers of the matrix coefficients and offsets within the core. These registers are described in Table 1.

Table 1: **EDK pCore Interface Register Descriptions**

Address Offset (hex)	Register Name	Access Type	Default Value (hex)	Description	
0x00	ccm_reg00_control	R/W	0x1	Bit 0	Software Enable 0 – Not enabled 1 – Enabled
				Bit 1	Semaphore for PLB Register Update 0 – Normal Operating Mode, software safe for updating PLB registers 1 – Register Update, stored register values are updated in the core on the next rising edge of `vblank_in`
0x04	ccm_reg01_reset	R/W	0x0	Bit 0	Software Manual Reset 0 – Not Reset 1 – Force immediate reset, hold until bit is cleared
				Bit 1	Software Auto-synchronized Reset 0 – Not Reset 1 – Reset occurs automatically on the next rising edge of `vblank_in`
0x08	ccm_reg04_K11	R/W	from GUI	Matrix Coefficient values are presented in 18.15 fixed point format. 18-bit signed integer values are equivalent to real numbers in the [-4 : 4] range, multiplied by 32768.	
0x0C	ccm_reg05_K12	R/W	from GUI		
0x10	ccm_reg06_K13	R/W	from GUI		
0x14	ccm_reg07_K21	R/W	from GUI		
0x18	ccm_reg08_K22	R/W	from GUI		
0x1C	ccm_reg09_K23	R/W	from GUI		
0x20	ccm_reg10_K31	R/W	from GUI		
0x24	ccm_reg11_K32	R/W	from GUI		
0x28	ccm_reg12_K33	R/W	from GUI		
0x2C	ccm_reg13_ROFFSET	R/W	from GUI	OWIDTH + 1-bit wide signed integer value in the range [-(2^{OWIDTH}) : (2^{OWIDTH})-1]	
0x30	ccm_reg14_GOFFSET	R/W	from GUI		
0x34	ccm_reg15_BOFFSET	R/W	from GUI		

All of the Write registers are readable, enabling the user to verify writes or read back current values. There is an additional feature that effectively disables the core halting further operations, which blocks the propagation of all video signals. This function is controlled by setting the SW enable register, bit 0 of `ccm_reg00_control`, to 0. The default value of SW enable is 1 (enabled).

All registers other than the control and SW reset registers are double-buffered in hardware to ensure no image tearing happens if the K or offset values are modified in the active area of a frame. This double-buffering provides a more flexible and easier to use core because it decouples the register updates from the blanking period, allowing software a much larger window with which to update the parameter values. The updated values for the K and offset registers are latched into the shadow registers immediately after writing them, while the actual offset and matrix multiplier coefficients used are stored in the working registers.

Any Reads of registers during operation return the values stored in the shadow registers. The rising edge of `vblank_in` triggers the values from the shadow registers to be copied to the working registers, when bit 1 of `ccm_reg00_control` is set to 1. This semaphore bit helps to prevent partially updated shadow registers from being copied over to the working registers.

Figure 1 shows a software flow diagram for normal operation and updating registers during the operation of the core. The core can be effectively reset in-system by asserting SW reset (bit 0), which returns all control, coefficient, and offset values to their default values, specified through the Graphical User Interface when the core is instantiated. The core outputs are also forced to 0 until the SW reset bit is deasserted.

The EDK pCore option provides a standard PLB4.6 interface to the cores.

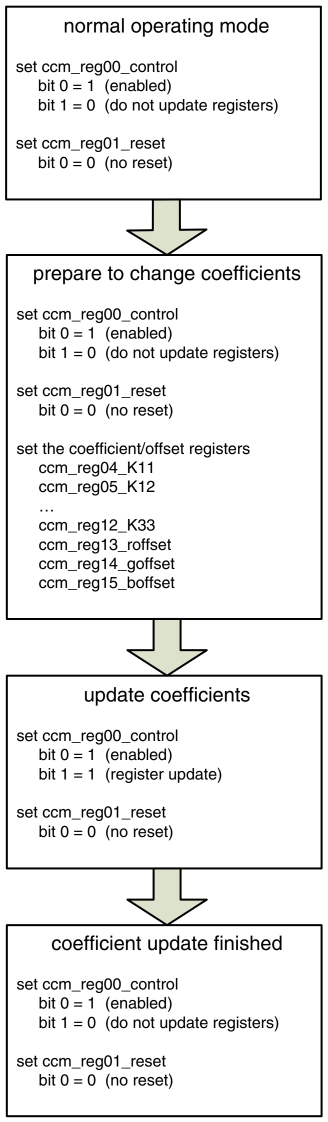


Figure 1: **Color Correction Matrix Programming Flow Chart**

Importing Generated pCore Into EDK

To generate an EDK pCore, select the EDK pCore option in the CORE Generator Graphical User Interface.

A folder is created in the coregen project directory with the Component Name specified in the CORE Generator Graphical User Interface. This folder contains all of the files needed for an EDK pCore, which contains two folders:

`/drivers`

`/pcores`

Any EDK pCore repository, whether internal or external to an EDK/Xilinx Platform Studio (XPS) project, contains matching `/drivers` and `/pcores` folders containing all of the pCores available in that pCore repository. Copy the contents of these generated folders into your EDK pCore repository location.

For XPS to detect the new pCore, select **Project** → **Rescan user repositories** in XPS.

Once XPS detects the new pCore, it appears in the XPS IP Catalog Tab under the repository in which it has been placed as "Color Correction Matrix 2.00.a". Drag-and-drop, or right-click and select **Add IP** to add a new instance of the core to your project.

To connect the Color Correction Matrix pCore to your PLB bus, expand the view of the pCore instance in the System Assembly View's Bus Interfaces tab, and select the associated PLB bus in the SPLB drop-down.

To configure Color Correction Matrix pCore parameters once it has been inserted into your XPS project, right-click the specific instance of the pCore in the System Assembly View window, and select **Configure IP ...** The input and output widths (C_IWIDTH, C_OWIDTH), the clipping (C_MAX) and the clamping values (C_MIN), and PLB bus configuration settings are configurable for this pCore instance.

To set the PLB base address for the Color Correction Matrix pCore, set the address in either the core's configuration or in the System Assembly View's Addresses' tab. The Color Correction Matrix pCore requires an address range of 256 bytes.

To connect the Color Correction Matrix pCore's video input and output signals, expand the entry for the pCore instance in the System Assembly View's Ports tab. Use the drop-down items to associate each port of the Color Correction Matrix pCore with the desired connection within your EDK system. Note that the clk input is the clock rate of the Color Correction Matrix pCore's video-processing logic, which should be connected to the video clock, not the PLB bus clock.

General Purpose Processor Overview

The General Purpose Processor interface exposes all matrix coefficients and offsets as ports. This option is very useful for developers designing a system with a user-defined bus interface (decoding logic and register banks) to an arbitrary processor. The coefficient and offset ports have the control mechanism described in the previous section to prevent tearing or committing partially updated port values. The ports for the General Purpose Processor Interface are described in detail in "Core Symbol and Port Descriptions".

Constant Interface Overview

The Constant Interface assumes the coefficient matrix and offset values are constants. There is no processor interface and the core is not programmable, but can be reset, enabled, or disabled using the `sclr` and `ce` pins. The ports for the Constant Interface are described in detail in "Core Symbol and Port Descriptions".

Programmer's Guide

The software API is provided to allow easy access to the CCM pCore's shared memory registers defined in Table 1.

To utilize the API functions provided, the following two header files must be included in the user C code:

```
#include "ccm.h"
```

```
#include "xparameters.h"
```

The hardware settings of your system, including the base address of your CCM core, are in the `xparameters.h` file. The `ccm.h` file contains the API of functions specifically for controlling the CCM pCore.

The drivers subdirectory of the pCore contains the `example.c` file in the `ccm_v2_00_a/example` subfolder. This file is a sample C program that demonstrates how to use the CCM pCore API. It contains the report_ccm_settings() function, which demonstrates how to use the functions provided by the `ccm.h` driver to read the current status of the core's configuration registers. This file also contains the CCM_Update_Example() function, which provides an example of updating those configuration registers.

Each software register defined in Table 1 has a constant defined in `ccm.h` that is set to the offset for that register. To write to a register, use the CCM_WriteReg() function using the base address of your CCM pCore instance (from `xparameters.h`), the offset of the desired register, and the data to write. For example:

```
CCM_WriteReg(XPAR_CCM_0_BASEADDR, CCM_REG04_K11, 12345);
```

Reading a value from a register also uses the base address and offset for the register:

```
Xuint32 value = CCM_ReadReg(XPAR_CCM_0_BASEADDR, CCM_REG04_K11);
```

For operations that require reading or writing only a single bit, rather than an entire 32-bit word, `ccm.h` provides pre-defined bit masks as shown in Table 2.

Table 2: **Pre-defined Bit Masks**

CCM_CTL_EN_MASK	Bit mask for the software enable bit of the control register
CCM_CTL_RUE_MASK	Bit mask for the register update enable bit of the control register
CCM_RST_RESET	Bit mask for the manual reset bit of the reset register
CCM_RST_AUTORESET	Bit mask for the autoreset bit of the reset register

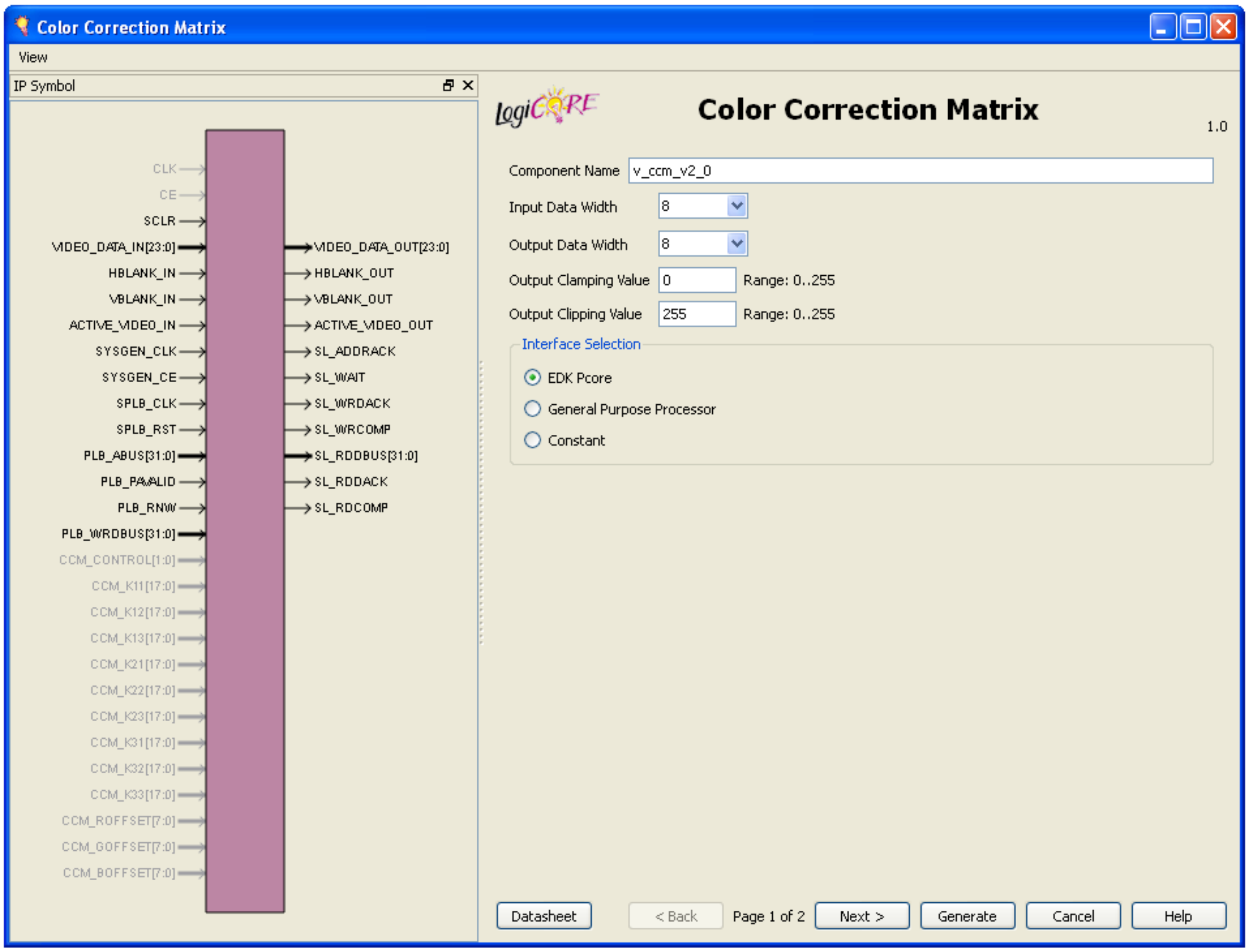
For additional convenience, pre-defined functions are provided in `ccm.h` for the most-used operations as shown in Table 3.

Table 3: **Pre-defined Functions**

CCM_Enable(BaseAddress)	Enables the CCM pCore software enable
CCM_Disable(BaseAddress)	Disables the CCM pCore software enable
CCM_RegUpdateEnable(BaseAddress)	Enables the CCM pCore register update enable
CCM_RegUpdateDisable(BaseAddress)	Disables the CCM pCore register update enable
CCM_Reset(BaseAddress)	Asserts the CCM pCore manual reset
CCM_ClearReset(BaseAddress)	Clears the CCM pCore resets
CCM_AutoSyncReset(BaseAddress)	Asserts the CCM pCore auto reset

CORE Generator – Graphical User Interface

The Color Correction Matrix core is easily configured to meet developers' specific needs before instantiation through the CORE Generator™ graphical user interface (GUI). Once developers start to build the Color Correction Matrix core within the CORE Generator system, they are guided through and asked to set various parameters. This section provides a quick reference to the windows and parameters that can be configured at compile time.



Component Name: v_ccm_v2_0
Input Data Width: 8
Output Data Width: 8
Output Clamping Value: 0 Range: 0..255
Output Clipping Value: 255 Range: 0..255
Interface Selection
[✓ EDK Pcore]
General Purpose Processor
Constant

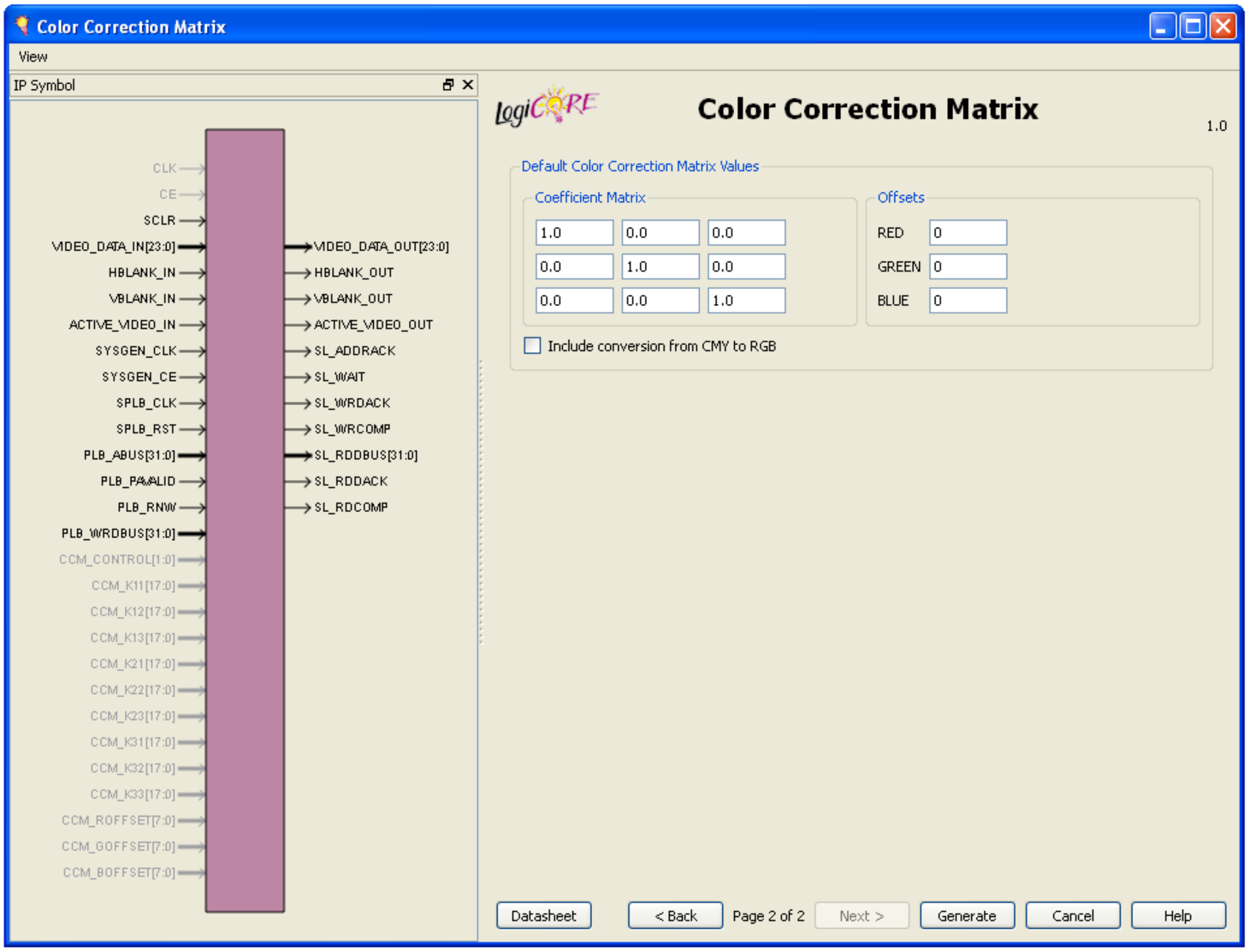
Figure 2: **Graphical User Interface – Screen 1**

The first screen (Figure 2) shows a representation of the IP symbol on the left side, and the settable parameters on the right, which are described as follows:

- **Component Name**: The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9 and "_".
- **Input Data Width (IWIDTH)**: Specifies the bit width of the input color channel for each component.
- **Output Data Width (OWIDTH)**: Specifies the bit width of the output color channel for each component.
- **Output Clamping Value**: Specifies the minimum value of the output. Allowable values are from 0 to 2OWDITH-1. The clamping value must be less than the clipping value.
- **Output Clipping Value**: Specifies the maximum value of the output. Allowable values are from 0 to 2OWDITH-1. The clipping value must be larger than the clamping value.

- **Interface Selection**: As described in the previous sections, this option allows for the configuration of three different interfaces for the core.
  - **EDK pCore Interface**: The CORE Generator tool generates a pCore that can be easily imported into an EDK project as a hardware peripheral and coefficients can be programmed via registers. When the EDK pCore interface is selected, the Input- and Output- Data Width, Clipping -, and Clamping Value fields, as well as Page 2 of the GUI are inactive. The parameters of the EDK pCore are controllable via the EDK core configuration GUI and software drivers.
  - **General Purpose Processor Interface**: The CORE Generator tool generates a set of ports to be used to program the core.
  - **Constant Interface**: The matrix coefficients and offsets are constant, and, consequently, no programming is necessary.

The second screen (Figure 3) also shows a representation of the IP symbol on the left side, but has a second set of settable parameters on the right, as described in this section.



Default Color Correction Matrix Values

Coefficient Matrix

1.0	0.0	0.0
0.0	1.0	0.0
0.0	0.0	1.0

Offsets

RED	0
GREEN	0
BLUE	0

☐ Include conversion from CMY to RGB

Figure 3: **Graphical User Interface – Screen 2**

- **Coefficient Matrix**: Enter the floating-point coefficients ranging from [-4, 4] (K in Equation 1) by specifying the 18 bit coefficients with 15 fractional bits of the coefficient matrix. The entered values will be the default used to initialize the core and the values used when the core is reset. Enter the real valued coefficients as floating-point decimal values in the range [-4.0, 4.0] (K in Equation 1). When the core is generated, the floating-point decimal value is converted to an 18-bit vector with 15 fractional bits, which are used internally to the core. The specified coefficient values will be the defaults used to initialize the core and the values used when the core is reset.

- **Offsets**: Enter the offset coefficients (O in Equation 1). These signed coefficients have the same bit width as the output. Enter the offset values (O in Equation 1). These signed integer values must be in the range $[-2^{OWIDTH}, 2^{OWIDTH}]$, and are 1-bit wider than the Output Data Width specified on the first page of the GUI.
- **Include conversion from CMY to RGB**: The Color Correction Matrix core can accept either RGB or CMY inputs. When selecting this conversion, the core will modify the coefficient matrix to also convert the CMY input to RGB output. The equation to calculate the new coefficient matrix is as follows:

$$K' = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

Equation 2

Core Symbol and Port Descriptions

As discussed previously, the Color Correction Matrix core can be configured with three different interface options, each resulting in a slightly different set of ports. The Color Correction Matrix core uses a set of signals that is common to all of the Xilinx Video IP cores called the Xilinx Streaming Video Interface (XSVI). The XSVI, clk, ce, and sclr signals are common to all interface options and are shown in Figure 4 and described by Table 4.

Xilinx Streaming Video Interface

The Xilinx Streaming Video Interface (XSVI) is a set of signals common to all of the Xilinx video cores used to stream video data between IP cores. For a complete description of this interface, see the UG762: Xilinx Streaming Video Interface User Guide. XSVI is also defined as an Embedded Development Kit (EDK) bus type so that the tool can automatically create input and output connections to the core. This definition is embedded in the pCORE interface provided with the IP, and it allows an easy way to cascade connections of Xilinx Video Cores. The Color Correction Matrix IP core uses the following subset of the XSVI signals:

- video_data
- vblank
- hblank
- active_video

Other XSVI signals on the XSVI input bus, such as video_clk, vsync, hsync, field_id, and active_chr do not affect the function of this core.

Note: These signals are neither propagated, nor driven on the XSVI output of this core.

The following is an example EDK Microprocessor Peripheral Definition (.MPD) file definition.

Input side:

```
BUS_INTERFACE BUS = XSVI_IN, BUS_TYPE = TARGET, BUS_STD = XSVI

PORT hblank_i       = hblank,       DIR = I,                          BUS = XSVI_IN
PORT vblank_i       = vblank,       DIR = I,                          BUS = XSVI_IN
PORT active_video_i = active_video, DIR = I,                          BUS = XSVI_IN
PORT video_data_i   = video_data,   DIR = I, VEC=[C_DATA_WIDTH-1:0], BUS = XSVI_IN
```

Output side:

```
BUS_INTERFACE BUS = XSVI_OUT, BUS_TYPE = INITIATOR,  BUS_STD = XSVI

PORT hblank_o        = hblank,       DIR = I,                             BUS = XSVI_OUT
PORT vblank_o        = vblank,       DIR = I,                             BUS = XSVI_OUT
PORT active_video_o  = active_video, DIR = I,                             BUS = XSVI_OUT
PORT video_data_o    = video_data,   DIR = I, VEC=[3*C_DATA_WIDTH-1:0],  BUS = XSVI_OUT
```

The Color Correction Matrix IP core is fully synchronous to the core clock, clk. Consequently, the input XSVI bus is expected to be synchronous to the input clock, clk. Similarly, to avoid clock re-sampling issues, the output XSVI bus for this IP is synchronous to the core clock, clk. The video_clk signals of the input and output XSVI buses are not used.

Constant Interface

As this interface does not provide additional programmability, the Constant Interface has no ports other than the Xilinx Streaming Video Interface, clk, ce, and sclr signals. The Constant Interface Core Symbol is shown in Figure 4.

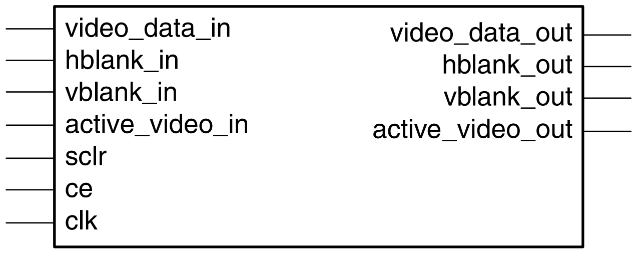


Figure 4: **Core Symbol for the Constant Interface**

Table 4: **Port Descriptions for the Constant Interface**

Port Name	Port Width	Direction	Description
video_data_in	3**IWIDTH*	IN	Data input bus
hblank_in	1	IN	Horizontal blanking input
vblank_in	1	IN	Vertical blanking input
active_video_in	1	IN	Active video signal input
video_data_out	3**OWIDTH*	OUT	Data output bus
hblank_out	1	OUT	Horizontal blanking output
vblank_out	1	OUT	Vertical blanking output
active_video_out	1	OUT	Active video signal output
clk	1	IN	Rising-edge clock
ce	1	IN	Clock enable (active high)
sclr	1	IN	Synchronous clear – reset (active high)

- **video_data_in**: This bus contains the three individual color inputs in the following order from MSB to LSB [red : blue : green] or [cyan : yellow : magenta]. Color values are expected in IWIDTH bits wide unsigned integer representation.

Bits	3IWIDTH-1:2IWIDTH	2IWIDTH-1:IWIDTH	IWIDTH-1:0
Video Data Signals	Red	Blue	Green
	Cyan	Yellow	Magenta

- **hblank_in**: The `hblank_in` signal conveys information about the blank/non-blank regions of video scan lines. This signal is not actively used in the CCM core, but passed through the core with a delay matching the latency of the corrected data.
- **vblank_in**: The `vblank_in` signal conveys information about the blank/non-blank regions of video frames, and is used by the CCM core to detect end of a frame, when user registers can be copied to active registers to avoid visual tearing of the image. This signal is passed through the core with a delay matching the latency of the corrected data.
- **active_video_in**: The `active_video_in` signal is high when valid data is presented at the input. This signal is not actively used in the CCM core, but passed through the core with a delay matching the latency of the corrected data.
- **clk - clock**: Master clock in the design.
- **ce - clock enable**: Pulling `CE` low suspends all operations within the core. Outputs are held, no input signals are sampled, except for reset (`SCLR` takes precedence over `CE`).
- **sclr - synchronous clear**: Pulling `SCLR` high results in resetting all output control signal pins to 0, and resetting the `video_data_out` bus to the clamping values for each color channel. Internal registers within the XtremeDSP slice and D-flip-flops are cleared. However, the core uses SRL16/SRL32-based delay lines for `hblank`, `vblank`, and `active_video` generation, which are not cleared by `SCLR`. This may result in non-zero outputs after `SCLR` is deasserted, until the contents of SRL16/SRL32s are flushed. Unwanted results can be avoided if `SCLR` is held active until SRL16/SRL32s are flushed.
- **video_data_out**: This bus contains RGB output in the same order as `video_data_in`. Color values are represented as OWIDTH bits wide unsigned integers.

Bits	3*OWIDTH-1:2*OWIDTH	2*OWIDTH-1:OWIDTH	OWIDTH-1:0
Video Data Signals	Red	Blue	Green

- **hblank_out, vblank_out and active_video_out**: The corresponding input signals are delayed so active_video and blanking outputs are in phase with the video data output, maintaining the integrity of the video stream. The blanking and active_video outputs are connected to the corresponding inputs via delay lines matching the propagation delay of the RGB processing pipe. Unwanted blanking inputs should be tied high, and corresponding outputs left unconnected, which will result in the trimming of any unused logic within the core.

  The active_video blanking signals do not affect the processing behavior of the core. Asserting or deasserting them will not stall processing or the R, G, B streams, and neither will force video outputs to zero.

EDK pCore Interface

The EDK pCore Interface generates Processor Local Bus (PLB4.6) interface ports in addition to the clk, ce, sclr, and Xilinx Streaming Video Signals. The PLB bus signals are automatically connected when the generated pCore is inserted into an EDK project. The Core Symbol for the EDK pCore Interface is shown in Figure 5. The Xilinx Streaming Video Interface clk, ce, and sclr are described in the previous section (Table 4). For more information on the PLB bus signals, see Processor Local Bus (PLB) v4.6.

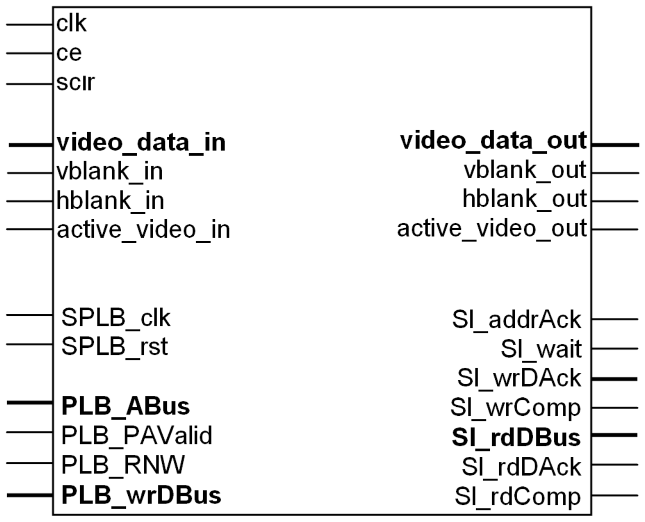


Figure 5: **Core Symbol for the EDK pCore Interface**

General Purpose Processor Interface

The General Purpose Processor Interface exposes all matrix coefficients and offsets as ports. The Core Symbol for the General Purpose Processor Interface is shown in Figure 6. The Xilinx Streaming Video Interface clk, ce, and sclr are described in the previous section. The General Purpose Processor ports are described in Table 5.

The coefficient and offset ports are stored in the port registers within the core, and the port values are latched into the working registers at the rising edge of `vblank_in` when bit 1 of `ccm_control` port is set to 1. At the beginning of the next frame, designated by a rising edge in `vblank_in`, the port registers will all update.

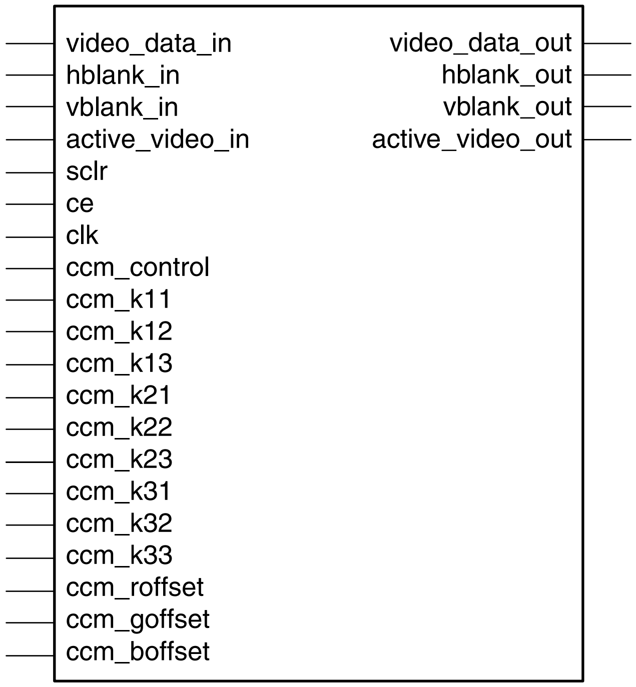


Figure 6: **Core Symbol for the General Purpose Processor Interface**

Table 5: **Ports for the General Purpose Processor Interface**

Port Name	Port Width	Description
ccm_control	2	Bit 0 Software Enable 0 - Not enabled 1 - Enabled Bit 1 Semaphore for PLB Register Update 0 - Normal Operating Mode, software safe for updating PLB registers 1 - Register Update, stored register values are updated in the core on the next rising edge of `vblank_in`
ccm_k11	18	Matrix Coefficient values are presented in 18.15 fixed point format. 18-bit signed integer values are equivalent to real numbers in the [-4 : 4] range, multiplied by 32768.
ccm_k12	18	
ccm_k13	18	
ccm_k21	18	
ccm_k22	18	
ccm_k23	18	
ccm_k31	18	
ccm_k32	18	
ccm_k33	18	
ccm_roffset	OWIDTH + 1 wide	OWIDTH + 1-bit wide signed integer value in the range [-(2^{OWIDTH}) : (2^{OWIDTH})-1]
ccm_goffset	OWIDTH + 1 wide	
ccm_boffset	OWIDTH + 1 wide	

Control Signals and Timing

The propagation delay of the Color Correction Matrix core is independent of parameterization and actual signal (R, G, B, `hblank_in`, `vblank_in`, `active_video_in`) values. Deasserting `CE` suspends processing, which may be useful for data-throttling, to temporarily cease processing of a video stream in order to match the delay of other processing components.

The processing latency of the core is seven `CLK` cycles.

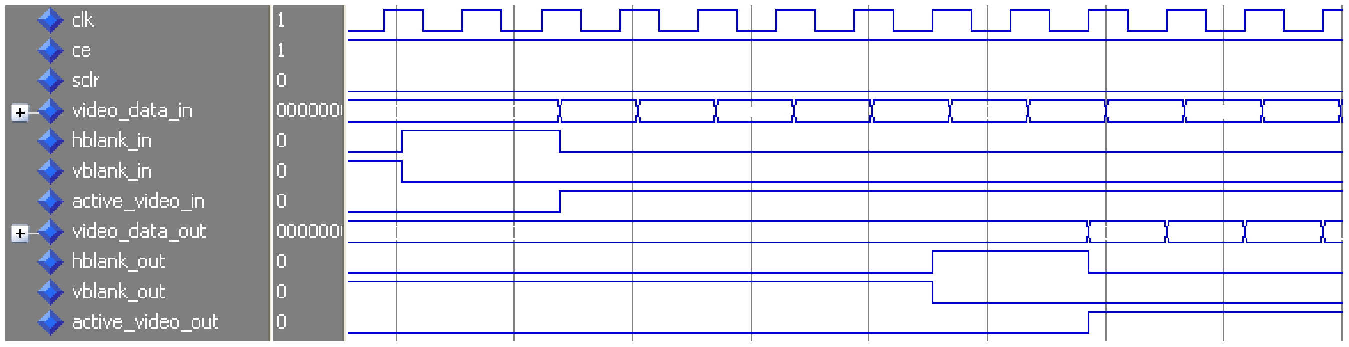


Figure 7: **Timing Signals**

Core Resource Utilization and Performance

For an accurate measure of the usage of primitives, slices, and CLBs for a particular instance, check the **Display Core Viewer after Generation** check box in the CORE Generator interface.

The information presented in Table 6, Table 7, Table 8, and Table 9 is a guide to the resource utilization of the Color Correction Matrix core for all input/output width combinations for Spartan-3 DSP, Spartan-6, Virtex-5, and Virtex-6 FPGA families. The Xtreme DSP Slice count is always 9, regardless of parameterization, and this core does not use any block RAMs, dedicated I/O, or CLK resources. The design was tested using ISE® v12.2 tools with default tool options for characterization data.

Table 6: **Spartan-3A DSP - XC3SD3400A, Speedgrade = 4**

Input Width	Output Width	FFs	LUTs	Slices	XtremeDSP Slices	Clock Frequency (MHz)
8	8	71	33	52	9	200
8	10	83	39	55	9	181
8	12	95	45	62	9	181
10	8	73	36	51	9	200
10	10	85	42	62	9	197
10	12	97	48	67	9	181
12	8	75	39	56	9	191
12	10	87	45	63	9	191
12	12	99	51	68	9	181

Table 7: **Spartan-6 -XC6SLX45FGG676,C, Speedgrade = 3**

Input Width	Output Width	FFs	LUTs	Slices	XtremeDSP Slices	Clock Frequency (MHz)
8	8	71	41	15	9	201
8	10	83	45	19	9	192
8	12	95	52	21	9	210
10	8	73	41	16	9	201
10	10	85	47	18	9	201
10	12	97	53	23	9	149
12	8	75	44	16	9	201
12	10	87	49	20	9	149
12	12	99	56	24	9	201

Table 8: **Virtex-5 - XC5VSX50T, Speedgrade = 1**

Input Width	Output Width	FFs	LUTs	Slices	XtremeDSP Slices	Clock Frequency (MHz)
8	8	71	33	32	9	371
8	10	83	39	33	9	371
8	12	95	45	40	9	362
10	8	73	33	28	9	371
10	10	85	39	36	9	362
10	12	97	45	46	9	362
12	8	75	33	31	9	346
12	10	87	39	40	9	371
12	12	99	45	46	9	362

Table 9: **Virtex-6 - XC6VLX240TFF1156,C, Speedgrade = 2**

Input Width	Output Width	FFs	LUTs	Slices	BRAM18	BRAM8	Clock Frequency (MHz)
8	8	71	41	15	3	0	397
8	10	83	47	20	3	0	405
8	12	95	51	24	3	0	405
10	8	73	40	17	3	0	397
10	10	85	45	22	3	0	388
10	12	97	52	21	3	0	303
12	8	75	42	17	0	3	337
12	10	87	50	20	3	3	397
12	12	99	54	22	3	3	388

Known Issues

For for the latest Known Issues see XTP025.

References

1. Processor Local Bus (PLB) v4.6

Support

Xilinx provides technical support for this LogiCORE IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY.*

License Options

The Color Correction Matrix core provides the following three licensing options:

- Simulation Only
- Full System Hardware Evaluation
- Full

After installing the required Xilinx ISE software and IP Service Packs, choose a license option.

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess core functionality with either the example design provided with the Color Correction Matrix core, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place-and-route the design, evaluate timing, and perform functional simulation of the Color Correction Matrix core using the example design and demonstration test bench provided with the core.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before timing out (ceasing to function), at which time it can be reactivated by reconfiguring the device.

To obtain a Full System Hardware Evaluation license, do the following:

1. Navigate to the product page for this core.
2. Click Evaluate.
3. Follow the instructions to install the required Xilinx ISE software and IP Service Packs.

Full

The Full license key is available when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

- Functional simulation support
- Full implementation support including place and route and bitstream generation
- Full functionality in the programmed device with no time outs

To obtain a Full license key, you must purchase a license for the core. Click on the "Order" link on the Xilinx.com IP core product page for information on purchasing a license for this core. After doing so, click the "How do I generate a license key to activate this core?" link on the Xilinx.com IP core product page for further instructions.

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
04/24/09	1.0	Initial Xilinx release.
07/23/10	2.0	Updated for core version 2.0.

Notice of Disclaimer