FMC-IMAGEON HDMI Display Controller Tutorial

Version 1.0

Revision History

Version	Description	Date
1.0	HDMI Display Controller Tutorial • Vivado 2013.3 version	Mar 20, 2014

Table of Contents

Revision History 1

Table of Contents 2

Table of Figures 3

Table of Tables 4

About this Guide .... - 1 -

Introduction .... - 2 -

Requirements .... - 4 -

Software .... - 4 -

Hardware .... - 4 -

Zynq embedded design tools .... - 4 -

Xilinx Video IP .... - 4 -

Setup .... - 5 -

Extract the tutorial archive on your computer .... - 5 -

Tutorial Overview .... - 6 -

Reusable Components .... - 6 -

HDMI Display Controller Overview .... - 7 -

Embedded Design with Processor .... - 7 -

FMC-IMAGEON - I2C Controller .... - 7 -

FMC-IMAGEON – Video Interfaces .... - 8 -

AXI4-Stream Bridges .... - 8 -

Implement an HDMI Display Controller .... - 9 -

Licensing the Video and Image Processing Pack IP Cores .... - 9 -

Create a new Vivado project .... - 9 -

Create the Embedded Hardware Design with IP Integrator .... - 10 -

Add the I2C Controller .... - 12 -

Add the Video Output Path .... - 15 -

Add the AXI Video DMA .... - 18 -

Build the hardware with Vivado Design Suite .... - 21 -

Create the Embedded Software Application with SDK .... - 24 -

Set up your ZC702 Hardware .... - 30 -

Set up your ZedBoard Hardware .... - 31 -

Execute the HDMI Display Controller Design on Hardware using SDK .... - 31 -

References .... - 33 -
Known Issues and Limitations .... - 34 -
  Hello World Template – error: conflicting types for 'print' .... - 34 -
Troubleshooting .... - 35 -

Table of Figures

Figure 1 – ON Semiconductor Image Sensor with HDMI Input/Output FMC Bundle .... - 2 -
Figure 2 – FMC-IMAGEON Hardware – Connectivity Diagram .... - 3 -
Figure 3 – FMC-IMAGEON Hardware – Block Diagram .... - 3 -
Figure 4 – HDMI Display Controller .... - 7 -
Figure 5 – Create Block Design .... - 11 -
Figure 6 – Designer Assistance – Add IP .... - 11 -
Figure 7 – Designer Assistance – Run Block Automation .... - 11 -
Figure 8 –Run Block Automation .... - 12 -
Figure 9 – AXI IIC - Block Properties .... - 12 -
Figure 10 – Run Connection Automation – /fmc_imageon_iic_0/S_AXI .... - 13 -
Figure 11 – Run Connection Automation – /fmc_imageon_iic_0/IIC .... - 13 -
Figure 12 – External Interface Properties – fmc_imageon_iic .... - 14 -
Figure 13 – External Interface Properties – fmc_imageon_iic_rst_n .... - 14 -
Figure 14 – Validate Design .... - 15 -
Figure 15 – IP Repository Manager .... - 16 -
Figure 16 –HDMI output sub-module .... - 17 -
Figure 17 – Validate Design .... - 21 -
Figure 18 – Create HDL Wrapper .... - 22 -
Figure 19 – Let Vivado manage wrapper and auto-update .... - 22 -
Figure 20 – HDMI Display Controller – Resource Utilization .... - 23 -
Figure 21 – Address Map in SDK system.xml Tab .... - 24 -
Figure 22 – Board Support Package Settings .... - 25 -
Figure 23 – Generate a Linker Script .... - 26 -
Figure 24 – Import from File System – Dialog 1 .... - 27 -
Figure 25 – Import into Folder .... - 28 -
Figure 26 – Import from File System – Dialog 2 .... - 29 -

Table of Tables

Table 1 – New Project Settings .... - 10 -

About this Guide

This tutorial describes how to create a Vivado IP Integrator video design from scratch.

This manual contains the following chapters:

- Chapter “**Introduction**” describes provides an overview and features of the FMC-IMAGEON FMC module, as well as the hardware and software required for this tutorial.
- Chapter “**Tutorial Overview**” provides a general overview of this tutorial.
- Chapter “**Implement an HDMI Display Controller**” describes the steps required to implement an Embedded System that implements an HDMI Display Controller Tutorial.
- Appendix “**References**” provides a list of references to documentation related to the FMC module.
- Appendix “**Known Issues and Limitations**” provides a list of known issues and limitations with the tools and/or IP used in this tutorial.
- Appendix “**Troubleshooting**” provides a list of troubleshooting suggestions for this tutorial.

Introduction

The ON Semiconductor Image Sensor FMC bundle provides several high-definition video interfaces for Xilinx® FMC-enabled baseboards. The FMC module has on-board HDMI input/output interfaces. The ON Semiconductor VITA-2000-color image sensor module provides a high definition camera supporting high frame rates and featuring a global shutter.

This FMC bundle is ideal for developing application for machine vision, motion monitoring, and high-end security and surveillance.

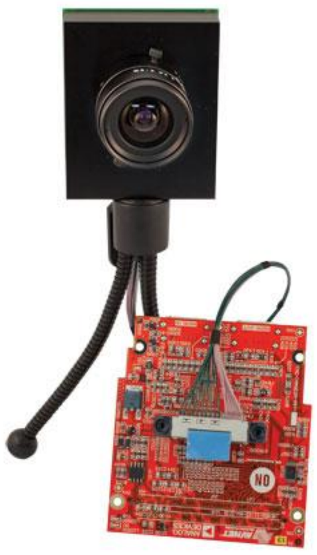


Figure 1 – ON Semiconductor Image Sensor with HDMI Input/Output FMC Bundle

As illustrated in Figure 2 and Figure 3, the FMC module connects to an FMC carrier, and provides the following interfaces:

- HDMI Input
- HDMI Output
- LCEDI Interface for VITA Image Sensor modules

The following block diagram illustrates how the connectivity of the FMC module.

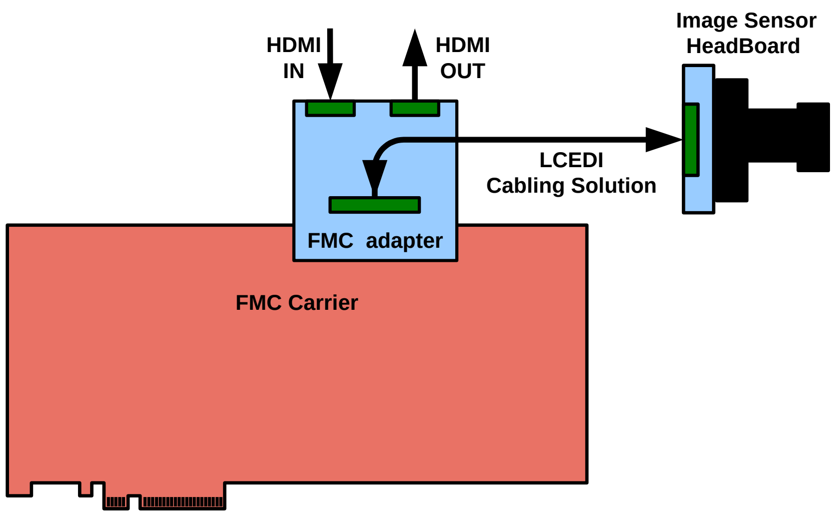


Figure 2 – FMC-IMAGEON Hardware – Connectivity Diagram

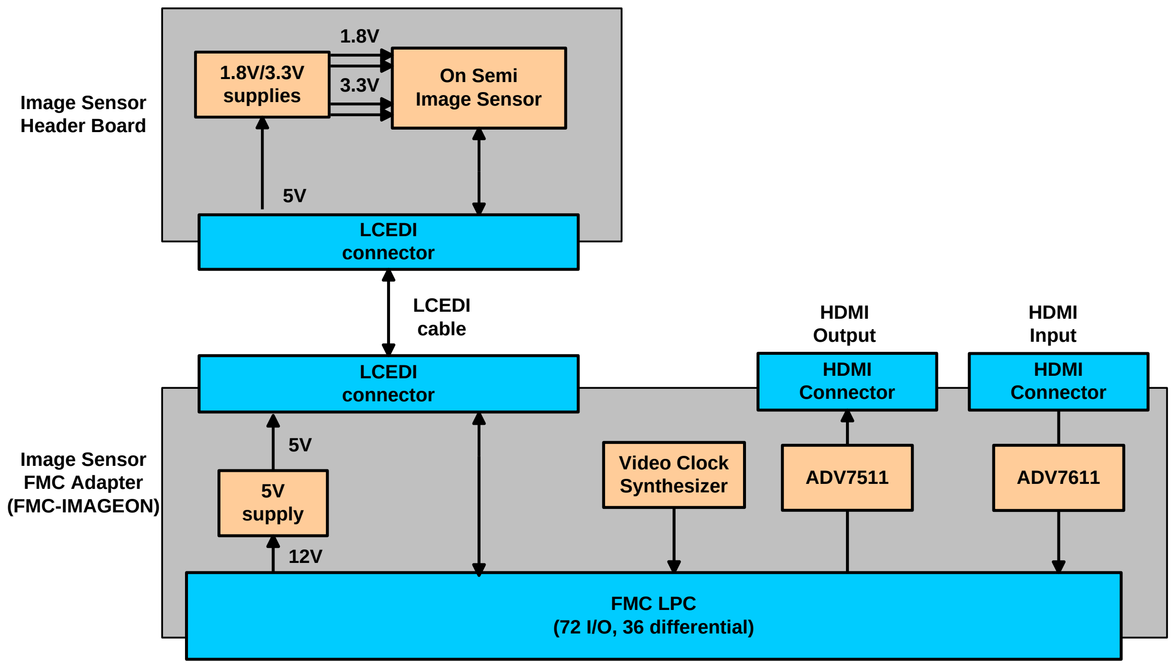


Figure 3 – FMC-IMAGEON Hardware – Block Diagram

Requirements

The software and hardware requirements for this tutorial are described in the following sections.

Software

The software required to build, and run the demonstrations is:

- Xilinx Vivado 2013.3
- Terminal Emulator (HyperTerminal or TeraTerm)

Hardware

The bare minimum required to run this reference design is:

- Computer with a minimum of 4 GB to complete a design[1]
- One of the following Zynq carriers
  - ZC702 (including power supply and cables)
  - ZedBoard (including power supply and cables)
- The Avnet HDMI Input/Output FMC Module (FMC-IMAGEON)
- HDMI (or DVI-D) monitor, including HDMI cable

Zynq embedded design tools

This tutorial assumes that you have experience creating Zynq embedded designs. More specifically, it assumes a working knowledge of Vivado tools, including IPI (Vivado IP Integrator) and SDK.

If you do not have this experience, it is HIGHLY recommended that you follow the **Introduction to Zynq** on-line training course, available on zedboard.org.
http://www.zedboard.org/course/introduction-zynq

Xilinx Video IP

This tutorial will make use of several of Xilinx's Video IP cores.

Although it is possible to complete this tutorial without consulting the datasheets for these Video IP cores, it is strongly recommended to consult the datasheets for each Video IP core to fully understand and appreciate their potential.

[1] Refer to http://www.xilinx.com/design-tools/vivado/memory.htm

Video Timing Controller

http://www.xilinx.com/products/intellectual-property/EF-DI-VID-TIMING.htm

Video Input to AXI4-Stream

http://www.xilinx.com/products/intellectual-property/video_in_to_axi4_stream.htm

AXI4-Stream to Video Output

http://www.xilinx.com/products/intellectual-property/axi4_stream_to_video_out.htm

AXI Video DMA

http://www.xilinx.com/products/intellectual-property/axi_video_dma.htm
In particular interest is the “Triple Frame Buffer Example” chapter

For the full list of Xilinx Video IP Cores, refer to the following web page:
http://www.xilinx.com/ipcenter/video/video_core_listing.htm

Setup

Before you begin this tutorial, carefully read the following sections which will describe how to extract the tutorial archive and how to test your video equipment.

Extract the tutorial archive on your computer

Extract the tutorial archive in the root of your C:\ drive. It will contain the following directories

```
C:\FMC_IMAGEON\2013_3\avnet_fmc_imageon_cores
C:\FMC_IMAGEON\2013_3\constraints
C:\FMC_IMAGEON\2013_3\code
C:\FMC_IMAGEON\2013_3\scripts
```

Tutorial Overview

This tutorial will guide you in creating a video design that implements an HDMI display controller for the Avnet HDMI Input/Output FMC module.

Reusable Components

The tutorial will make use of reusable components for the video input and video output interfaces:

- IP Cores, for each of the video interfaces on the FMC-IMAGEON module
    - FMC-IMAGEON – HDMI Input
    - FMC-IMAGEON – HDMI Output
    - FMC-IMAGEON – VITA Receiver

- TCL scripts : automatically create IP Integrator sub-modules

  For a 24 bits RGB video output path:
    - **fmc_imageon_hdmio_rgb.tcl** : video output path, 24 bits RGB format

- XDC constraints : defines pinout and constraints for various carriers
    - **zc702_fmc_imageon_hdmi_display.xdc** : constraints for ZC702
    - **zedboad_fmc_imageon_hdmi_display.xdc** : constraints for ZedBoard

- C source code : provides example initialization code
    - **fmc_imageon_hdmi_display.c/.h** : init code for HDMI display controller

For more information on these reusable components, refer to the **FMC-IMAGEON - IP CORES for Vivado** document.

HDMI Display Controller Overview

The HDMI display controller will make use of the AXI Video DMA IP core, as shown below.

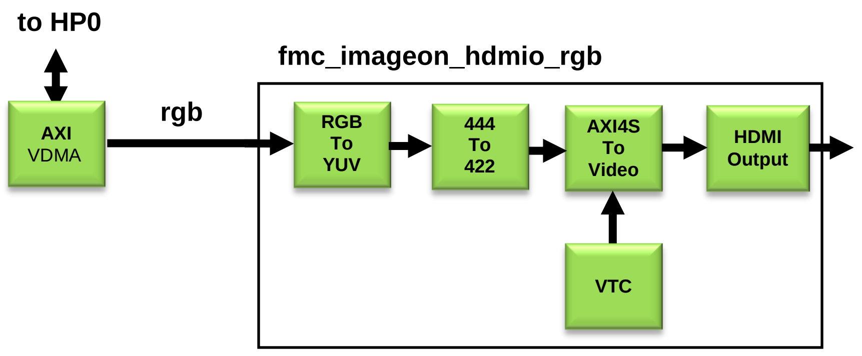


Figure 4 – HDMI Display Controller

The AXI Video DMA core will fetch video content from the Zynq's external memory via one of the High-Performance Ports (HP0), and produce video content in the AXI4-Stream protocol.

Embedded Design with Processor

This design requires an embedded design that contains a processor, as well as the following peripherals:

- external memory
- serial port (UART or USB-UART)

This tutorial will create a Zynq based embedded design.

Alternatively, for non-Zynq based carriers, a MicroBlaze based embedded design can also be created. However, this is not described in this tutorial, and left as an exercise to the user.

FMC-IMAGEON - I2C Controller

Once the embedded processor design is created, an I2C controller is implemented with the following Xilinx IP core:

- AXI I2C Controller
  - not shown in the previous block diagram, this core will allow the processor to configure the FMC-IMAGEON hardware peripherals, including:
    - ADV7611 : HDMI input device
    - ADV7511 : HDMI output device
    - CDCE913 : video clock synthesizer

FMC-IMAGEON – Video Interfaces

The following core, provided with the tutorial, will be used to interface to the ADV7511 device on the FMC-IMAGEON module. It is important to note that, on the FMC-IMAGEON module, the ADV7511 device is used in 16 bits YCbCr 4:2:2 with “embedded sync” mode.

- FMC-IMAGEON HDMI Output
    - this core contains logic that will embed the synchronization signals in the 16 bit video data sent to the ADV7511 HDMI output device.

The HDMI Output IP core makes a generic parallel video interface available to the design.

AXI4-Stream Bridges

The following cores are used to bridge between the HDMI interface and the AXI4-Stream protocol.

- Video Timing Controller
    - this core is capable of:
        - detecting the video timing on a video input interface
        - (re)generating video timing for a video output interface
    - a single VTC core will be used:
        - to generate the video timing required for the HDMI output interface.
- AXI4-Stream to Video Output
    - this core generates a generic parallel video interface (ie. DVI/HDMI) from a AXI4-Stream for Video interface
    - the core includes a FIFO allowing the AXI4-Stream for Video interface to run on a different clock

Implement an HDMI Display Controller

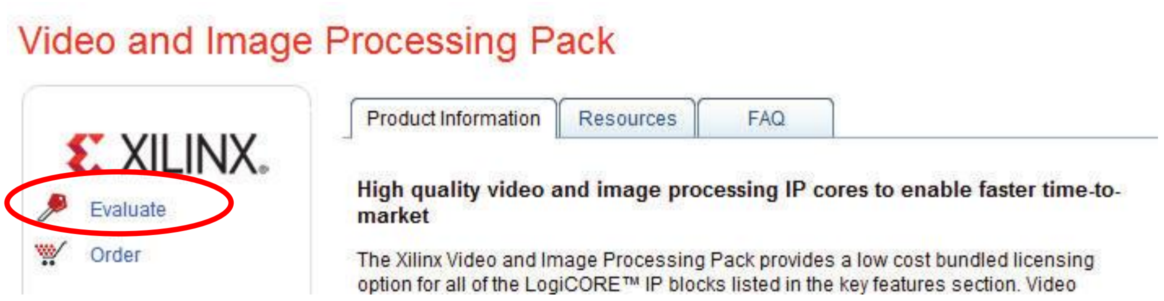
In this section, a new Vivado project will be created, implementing the HDMI Display Controller design for the Avnet FMC-IMAGEON module.

Licensing the Video and Image Processing Pack IP Cores

This design uses several of the Xilinx Video and Image Processing Pack IP cores that must be licensed prior to use. Follow these steps to request an evaluation license:

1. Go to:
   www.xilinx.com/products/intellectual-property/EF-DI-VID-IMG-IP-PACK

2. Click the Evaluate link located on the upper-left of the web page, and follow the online instructions



High quality video and image processing IP cores to enable faster time-to-market

The Xilinx Video and Image Processing Pack provides a low cost bundled licensing option for all of the LogiCORE™ IP blocks listed in the key features section. Video

3. The generated license file is sent by email. Follow the enclosed instructions to add the evaluation license features for the Video and Image Processing Pack.

Create a new Vivado project

To create a new project, start the Vivado™ design and analysis software and create a project with an embedded processor system as the top level.

1. Start the Vivado 2013.3 software.
2. Select **Create New Project** to open the New Project wizard
3. Use the information in the table below to make your selections in the wizard screen

Wizard Screen	**System Property**	**Setting or Command to Use**
Project Name	Project name	Specify the project name, such as: **tutorial**
	Project location	Specify the directory in which to store the project files:

		C:\FMC_IMAGEON\2013_3
	Create Project Subdirectory	Leave this checked.
Project Type	Specify the type of project to create	Use the default selection, specify **RTL Project**.
Add Sources		Do not make any changes on this screen.
Add Existing IP		Do not make any changes on this screen.
Add Constraints		Do not make any changes on this screen.
Default Part	Specify	Select **Boards**.
	Filter	In the Family list, select **Zynq-7000**
	Board list	Select one of the following board: **Zynq-7 ZC702 Evaluation Board** or **ZedBoard Zynq Evaluation and Development Kit**
New Project Summary	Project summary	Review the project summary before clicking **Finish** to create the project.

Table 1 – New Project Settings

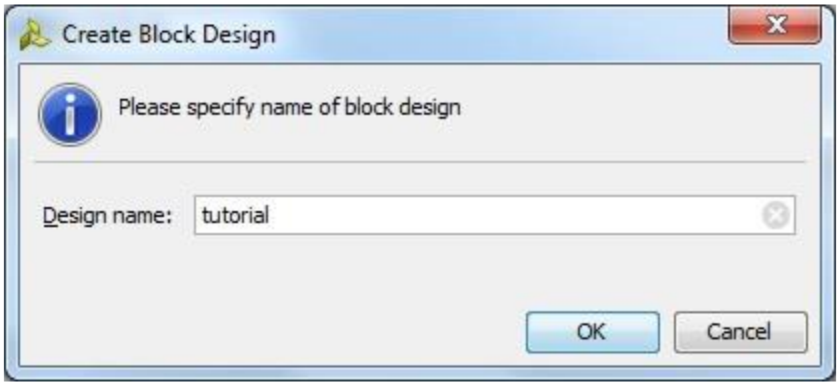
When you click **Finish**, the New Project wizard closes and the project you just created opens in Vivado.

Create the Embedded Hardware Design with IP Integrator

This section will guide you on how to create a basic embedded hardware design. This section may differ slightly for different carriers.

Create a new Vivado IP Integrator block diagram.

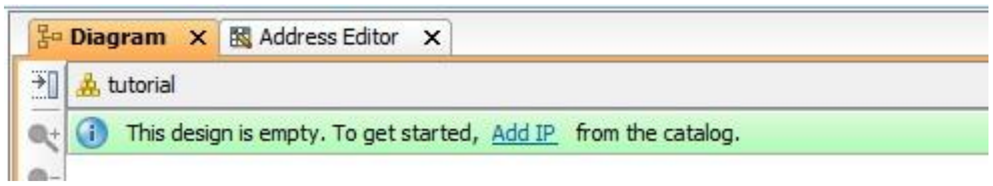
1. Click **Create Block Design** in the **IP Integrator** flow navigator.
   The Create Block Design dialog opens.
2. Specify a design name for the block diagram and click **OK**.
3. Click **Next**.



Design name: tutorial

Figure 5 – Create Block Design

Notice that Vivado IP Integrator provides design assistance:



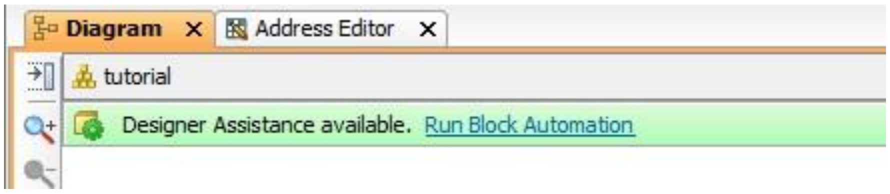
This design is empty. To get started, Add IP from the catalog.

Figure 6 – Designer Assistance – Add IP

Add the ZYNQ7 Processing System core to the block design.

4. Click **Add IP** in the design assistance
   or
   Right-click in the block design, then select **Add IP**.
5. Select the **ZYNQ7 Processing System** core,
   then click **ENTER** (or double-click selection)

Notice that Vivado IP Integrator provides additional design assistance:

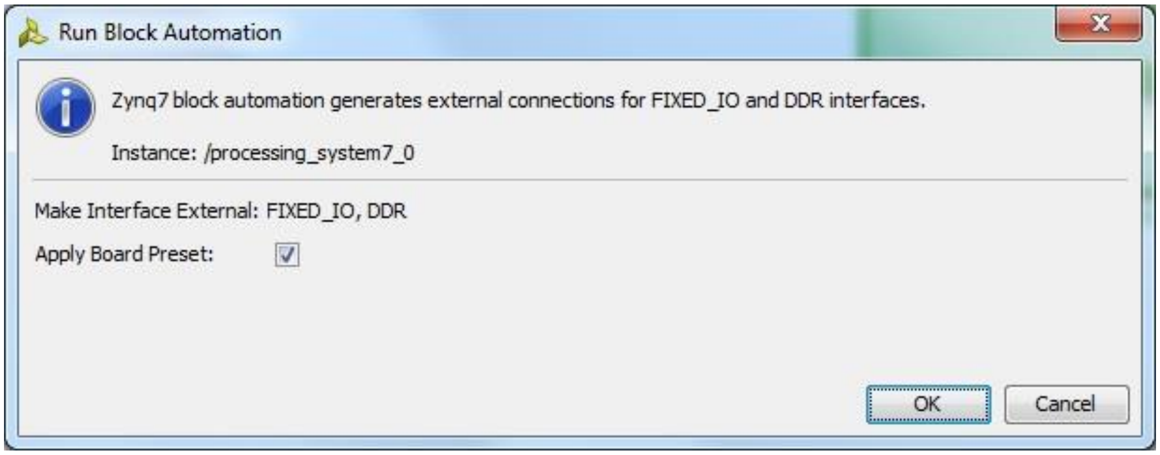


Designer Assistance available. Run Block Automation

Figure 7 – Designer Assistance – Run Block Automation

Configure the ZYNQ7 Processing System core using the designer assistance:

6. Click **Run Block Automation** in the design assistance
7. Select **/processing_system7_0**
8. Make sure the **Apply Board Preset** option is selected.
9. Click **OK**.



Zynq7 block automation generates external connections for FIXED_IO and DDR interfaces.

Instance: /processing_system7_0

Make Interface External: FIXED_IO, DDR

Apply Board Preset: ✓

Figure 8 –Run Block Automation

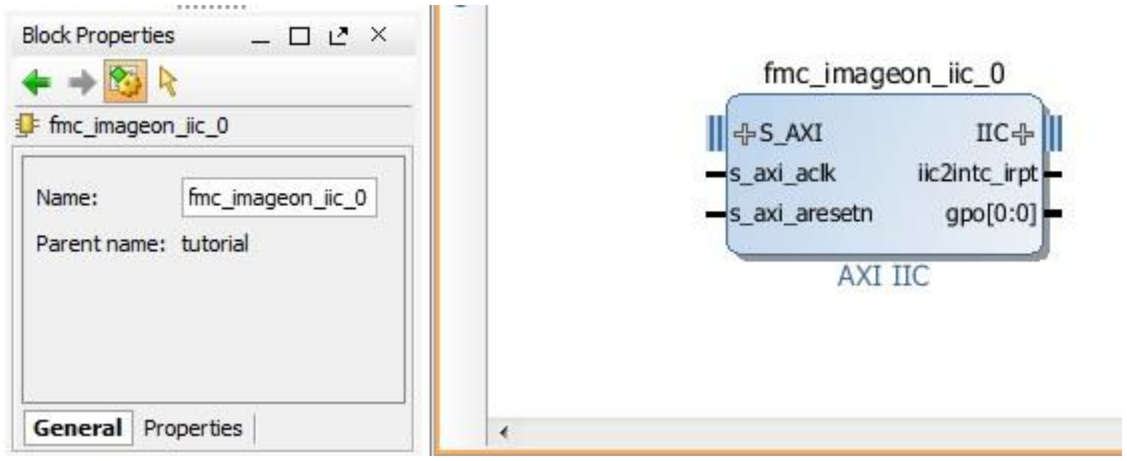
Add the I2C Controller

Add the **AXI IIC** IP core to the design, which will be used to configure the I2C peripherals on the FMC module.

1. **Right-click** in a blank portion of the block design, then select **Add IP**.
2. Select the **AXI IIC** core,
   then click **ENTER** (or double-click selection)

Rename the IIC controller to fmc_imageon_iic_0.

3. Select the **AXI IIC** core in the block diagram.
4. In the **Block Properties** dialog,
   modify the **Name** of the block to **fmc_imageon_iic_0**



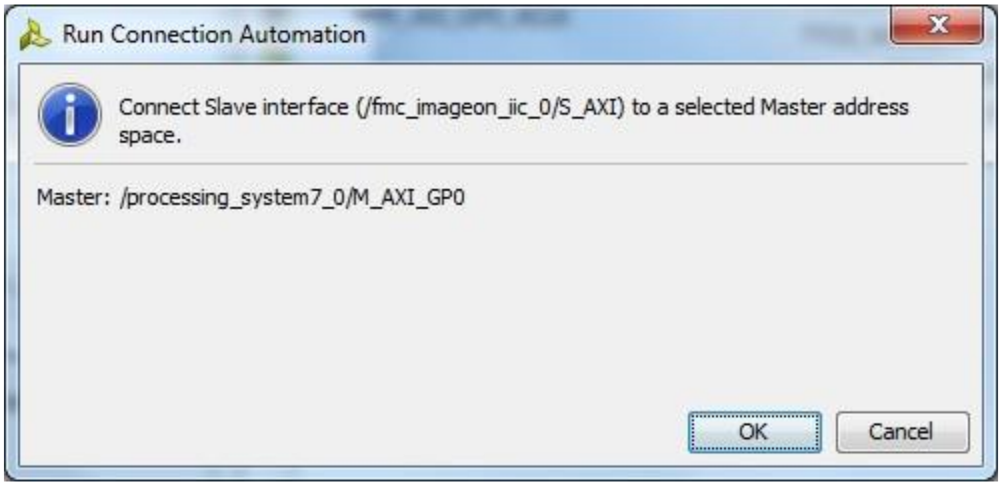
Name: fmc_imageon_iic_0

Parent name: tutorial

Figure 9 – AXI IIC - Block Properties

Use the designer assistance to connect the AXI IIC core to the ZYNQ7 Processing System's GP0 port.

5. Click **Run Connection Automation** in the design assistance
6. Select **/fmc_imageon_iic_0/S_AXI**
7. Click **OK**.


Run Connection Automation

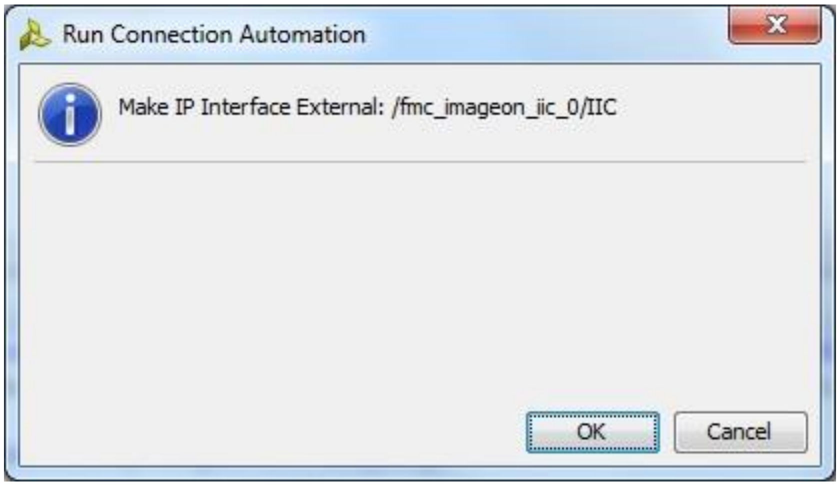
Connect Slave interface (/fmc_imageon_iic_0/S_AXI) to a selected Master address space.

Master: /processing_system7_0/M_AXI_GP0

Figure 10 – Run Connection Automation – /fmc_imageon_iic_0/S_AXI

Use the designer assistance to connect the AXI IIC core's external I/O

8. Click **Run Connection Automation** in the design assistance
9. Select **/fmc_imageon_iic_0/IIC**
10. [If the **Select Board Interface** drop down list appears, specify **Custom**]
11. Click **OK**.

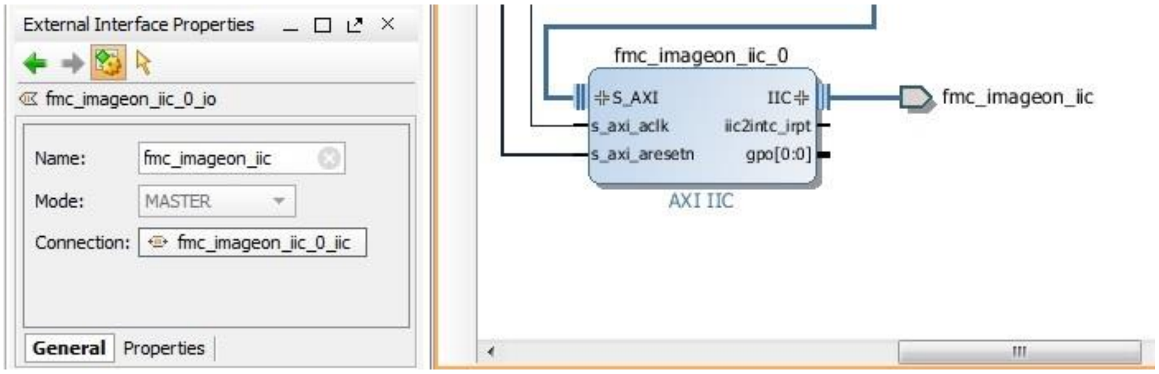

Run Connection Automation

Make IP Interface External: /fmc_imageon_iic_0/IIC

Figure 11 – Run Connection Automation – /fmc_imageon_iic_0/IIC

Rename the external iic_rtl port to fmc_imageon_iic

12. Select the **iic_rtl** external port in the block diagram.
13. In the **External Interface Properties** dialog,
    modify the **Name** of the block to **fmc_imageon_iic**



Name: fmc_imageon_iic

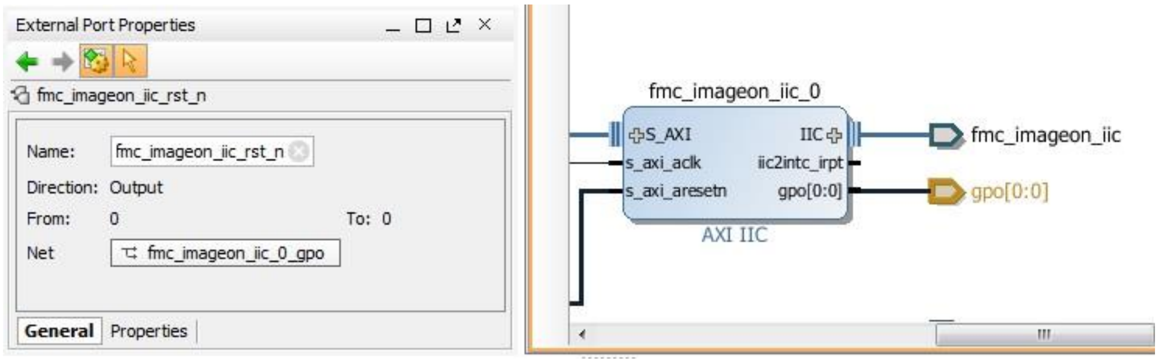
Figure 12 – External Interface Properties – fmc_imageon_iic

Connect the AXI IIC core's gpo[0:0] port to an external port

14. Select the **gpo[0:0]** port on the AXI IIC core
15. **Right-click**, then select **Make External**

Rename the external gpo[0:0] port to fmc_imageon_iic_rst_n

16. Select the **gpo[0:0]** external port in the block diagram.
17. In the **External Interface Properties** dialog,
    modify the **Name** of the block to **fmc_imageon_iic_rst_n**



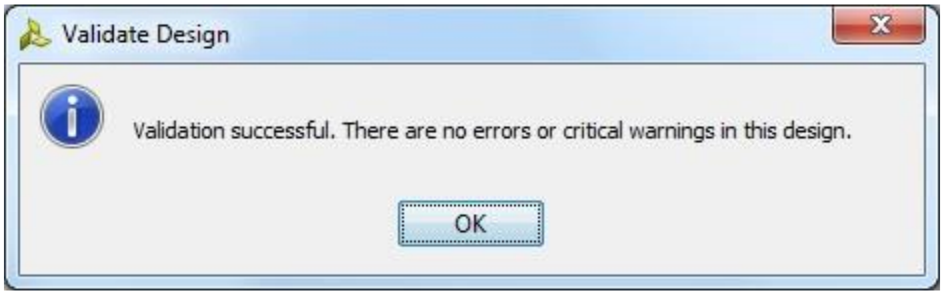
Name: fmc_imageon_iic_rst_n
Direction: Output
From: 0 To: 0

Figure 13 – External Interface Properties – fmc_imageon_iic_rst_n

At this point, validate that you have a correct design.

18. **Right-click** in a blank portion of the block design, then select **Validate Design**

19. If successful, Click **OK** and proceed with the next section.
    Otherwise, fix any errors in the design before proceeding.



Validate Design

Validation successful. There are no errors or critical warnings in this design.

Figure 14 – Validate Design

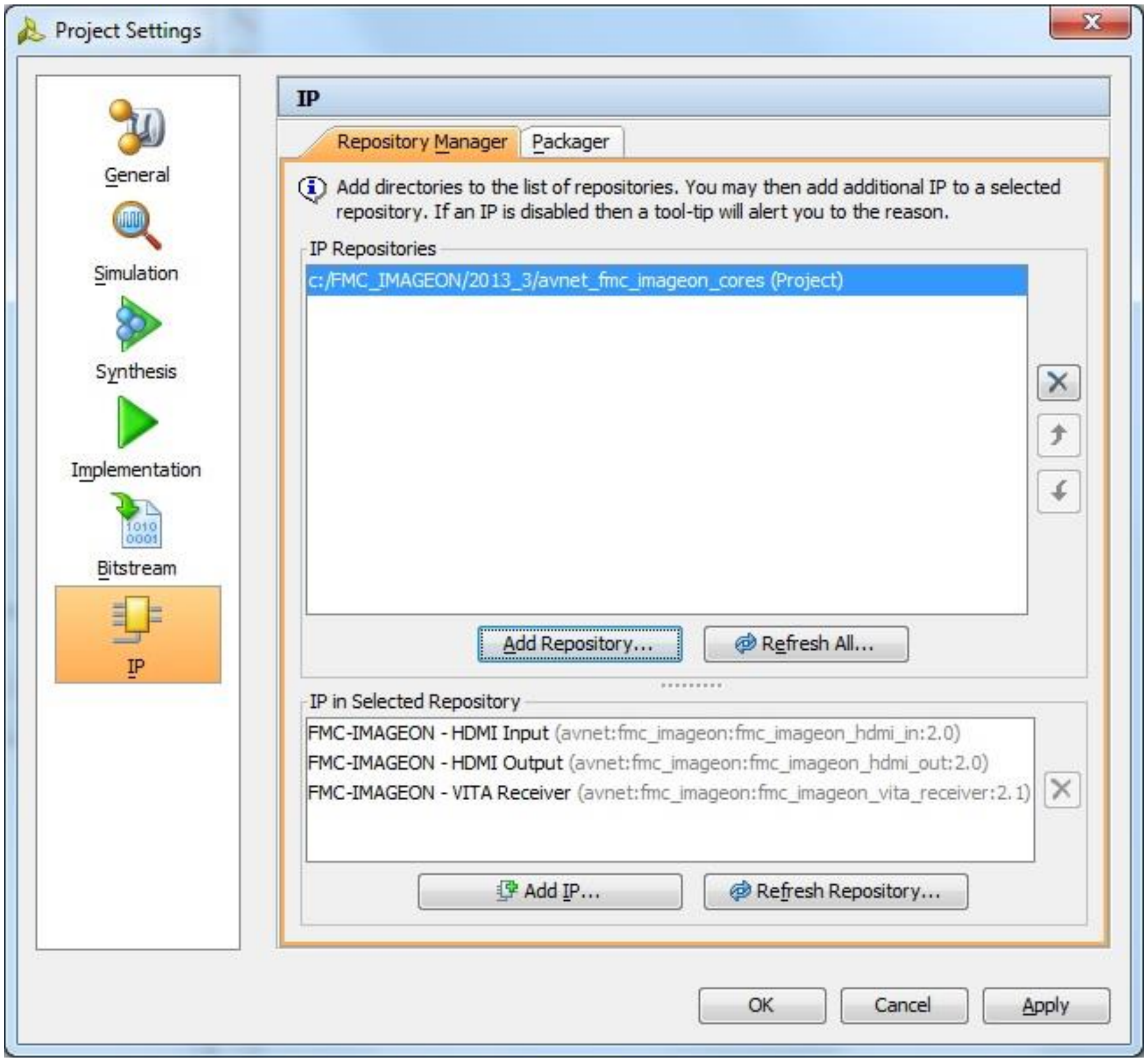
Add the Video Output Path

The video pipeline will make use of Avnet provided IP cores. In order for Vivado Design Suite to recognize these cores, we need to add the location of the Avnet FMC-IMAGEON cores to the repository path.

1. In the Vivado menu, select **Tools** => **Project Settings**.
   The Project Settings dialog opens.
2. Click on the IP icon on the left.
3. In the **Repository Manager** tab, click the **Add Repository** button.
4. Specify the "**C:\FMC_IMAGEON\2013_3\avnet_fmc_imageon_cores**" directory,
   then click the **Select** button.

The following three IP cores should be detected and appear in the **IP in Selected Repository** window:

- FMC-IMAGEON - HDMI Input
- FMC-IMAGEON - HDMI Output
- FMC-IMAGEON - VITA Receiver



IP

Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

IP Repositories

c:/FMC_IMAGEON/2013_3/avnet_fmc_imageon_cores (Project)

IP in Selected Repository

FMC-IMAGEON - HDMI Input (avnet:fmc_imageon:fmc_imageon_hdmi_in:2.0)
FMC-IMAGEON - HDMI Output (avnet:fmc_imageon:fmc_imageon_hdmi_out:2.0)
FMC-IMAGEON - VITA Receiver (avnet:fmc_imageon:fmc_imageon_vita_receiver:2.1)

Figure 15 – IP Repository Manager

5. Click **OK**.

The display controller will work in the 24bits RGB color space. For this reason, the following TCL script will be used to create the video output path:

- **fmc_imageon_hdmio_rgb.tcl** : video output path, 24 bits RGB format

Use the TCL console to run the TCL script.

6. In the **Tcl Console**, enter the following command to generate the video output path.
   source C:/FMC_IMAGEON/2013_3/scripts/fmc_imageon_hdmio_rgb.tcl
   then press {ENTER}

This will create one new video sub-module in your block diagram.

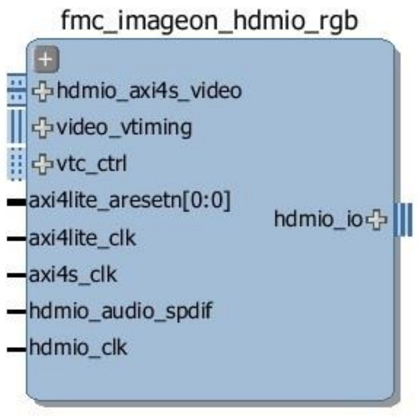


Figure 16 –HDMI output sub-module

Connect the AXI4-Lite clocks to the ZYNQ7 Processing System's FCLK_CLK0 port.

1. Select the fmc_imageon_hdmio_rgb sub-module's **axi4lite_ clk** port
2. Click and hold the left mouse button, then drag to the ZYNQ7's **FCLK_CLK0** port.
3. Release the mouse button to make the connection

Connect the AXI4-Lite resets to the proc_sys_reset core's peripheral_aresetn port.

4. Select the fmc_imageon_hdmio_rgb sub-module's **axi4lite_aresetn** port
5. Click and hold the left mouse button, then drag to the proc_sys_reset core's **peripheral_aresetn[0:0]** port.
6. Release the mouse button to make the connection

Use the designer assistance to connect the VTC core's control port (vtc_ctrl) to the ZYNQ7 Processing System's GP0 port.

7. Click **Run Connection Automation** in the design assistance
8. Select **/fmc_imageon_hdmio_rgb/vtc_ctrl**
9. Click **OK**.

Configure the ZYNQ7 Processing System to generate a 150MHz clock on its FCLK_CLK1 port, which will be used for the AXI4-Stream based interconnect and cores.

10. Double-click on the ZYNQ7 Processing System core.
11. Click on the **Clock Generation** block
12. Expand the **PL Fabric Clocks** section
13. Select the **FCLK_CLK1** clock
14. Specify a **Requested Frequency** of **150MHz**.
15. Click **OK**.

Connect the AXI4-Stream clocks to the ZYNQ7 Processing System's FCLK_CLK1 port.

16. Select the fmc_imageon_hdmio_rgb sub-module's **axi4s_clk** port
17. Click and hold the left mouse button, , then drag to the ZYNQ7's **FCLK_CLK1** port.
18. Release the mouse button to make the connection.

Create an external port for the HDMI output clock in the block diagram.

19. **Select** the fmc_imageon_hdmio_rgb sub-module's **hdmio_clk** port
20. **Right-click**, then select **Make External**

Connect the HDMI I/O ports (hdmio_io) to external ports

21. **Select** the fmc_imageon_hdmio_rgb sub-module's **hdmio_io** port
22. **Right-click**, then select **Make External**

Add the AXI Video DMA

Add the **AXI Video DMA** IP core to the design, which will be used to read video content from the Zynq's external memory.

1. **Right-click** in a blank portion of the block design, then select **Add IP**.
2. Select the **AXI Video Direct Memory Access** core,
   then click **ENTER** (or double-click selection)

Configure the AXI VDMA core's Read Channel, and disable its Write Channel.

3. Double-click on the **AXI Video Direct Memory Access** core.
4. Click on the **Basic** tab
5. In the **Write Channel** section
   a. Click the **Enable Write Channel** checkbox to **disable** it.
6. In the **Read Channel** section
   a. Make sure the **Enable Read Channel** checkbox is **enabled**.
   b. Set the **Line Buffer Depth** to **4096**.
7. Click on the **Advanced** tab
8. In the **Read Channel Options**
   a. Set the **GenLock Mode** to **Master**
   b. Click the **Allow Unaligned Transfers** checkbox to **enable** it.
9. Click **OK**.

Use the designer assistance to connect the AXI Video DMA core's S_AXI_LITE port to the ZYNQ7 Processing System's GP0 port.

10. Click **Run Connection Automation** in the design assistance
11. Select **/axi_vdma_0/S_AXI_LITE**
12. Click **OK**.

Configure the ZYNQ7 Processing System to enable the HP0 high-performance port.

13. Double-click on the ZYNQ7 Processing System core.
14. Click on the **High Performance AXI 32b/64b Slave Ports** block
15. Expand the **HP Slave AXI Interface** section
16. Select the **S AXI HP0 interface** port
17. Click **OK**.

Connect the HP0 clock to the ZYNQ7 Processing System's FCLK_CLK1 port.

18. Select the ZYNQ's S_AXI_HP0_ACLK sub-module's **S_AXI_HP0_CLK** port
19. Click and hold the left mouse button, , then drag to the ZYNQ7's **FCLK_CLK1** port.
20. Release the mouse button to make the connection.

Connect the AXI VDMA clocks to the ZYNQ7 Processing System's FCLK_CLK1 port.

21. Select the axi_vdma_0 core's **m_axi_mm2s_aclk** port
22. Click and hold the left mouse button, then drag to the ZYNQ7's **FCLK_CLK1** port.
23. Release the mouse button to make the connection.

24. Select the axi_vdma_0 core's **m_axis_mm2s_aclk** port
25. Click and hold the left mouse button, then drag to the ZYNQ7's **FCLK_CLK1** port.
26. Release the mouse button to make the connection.

Use the designer assistance to connect the AXI Video DMA core's M_AXI_MM2S port to the ZYNQ7 Processing System's HP0 port.

27. Click **Run Connection Automation** in the design assistance
28. Select **/processing_system7_0/S_AXI_HP0**
29. Click **OK**.

The AXI Video DMA core's data width is 32 bits, but the fmc_imageon_hdmio_rgb sub-module's data width is 24 bits. In addition, the pixel's component order is not the same.
The pixels stored in external memory have the following format:

[31:24]	unused
[23:16]	RED
[15:8]	GREEN
[7:0]	BLUE

The pixels expected by the Xilinx Video IP have the following format:

[23:16]	RED
[15:8]	BLUE
[7:0]	GREEN

The AXI4-Stream Subset Converter core can be used to extract the desired 24 bits RGB pixels from the 32 bits data dwords, and re-order the color components for the Xilinx Video IP.

30. **Right-click** in a blank portion of the block design, then select **Add IP**.
31. Select the **AXI4-Stream Subset Converter** core,
    then click **ENTER** (or double-click selection)

Connect the AXI Video DMA core's AXI4-Stream port (M_AXIS_MM2S) to the video output path, via the AXI4-Stream Subset Converter core.

32. Select the axi_vdma_0 core's **M_AXIS_MM2S** port
33. Click and hold the left mouse button, then drag to the axis_subset_converter_0 core's **S_AXIS** port.
34. Release the mouse button to make the connection.

35. Select the axis_subset_converter_0 core's **M_AXIS** port.
36. Click and hold the left mouse button, then drag to the fmc_imageon_hdmio_rgb sub-module's **hdmio_axi4s_video** port.
37. Release the mouse button to make the connection.

Connect the AXI4-Stream Subset Converter core's clock ZYNQ7 Processing System's FCLK_CLK1 port.

38. Select the axis_subset_converter_0 core's **aclk** port
39. Click and hold the left mouse button, , then drag to the ZYNQ7's **FCLK_CLK1** port.
40. Release the mouse button to make the connection.

Connect the AXI4-Stream Subset Converter core's **aresetn** port.

41. Select the axis_subset_converter_0 core's **aresetn** port
42. Click and hold the left mouse button, then drag to the proc_sys_reset core's **peripheral_aresetn[0:0]** port.
43. Release the mouse button to make the connection

Run "Validate Design" to allow the data types to propagate through the AXI4-Stream Subset Converter core. Ignore the error message for now.

Configure the AXI4-Stream Subset Converter core to remap the 32 bit XRGB video data from external memory to the 24 bit RBG video data for the video IP cores.

44. **Double-click** on the axis_subset_converter_0 core.
45. In the **Master Interface Signal Properties** section:
    a. Specify **Manual** for the **TDATA Width**
    {NOTE: The TDATA Width parameter may remain disabled,
    closing then re-opening the core's config dialog will enable the TDATA Width parameter}
46. Click **OK.**
47. **Double-click** on the axis_subset_converter_0 core
48. In the **Master Interface Signal Properties** section:
    a. Set the **TDATA Width** to **3**
49. In the **Extra Settings** section:
    a. Set the **TDATA Remap String** to **tdata[23:16],tdata[7:0],tdata[15:8]**
50. Click **OK.**

For this design, the Video Timing Controller will only be used to generate video timing. Inside the fmc_imageon_hdmio_rgb sub-module, configure the Video Timing Controller core to disable its detector.

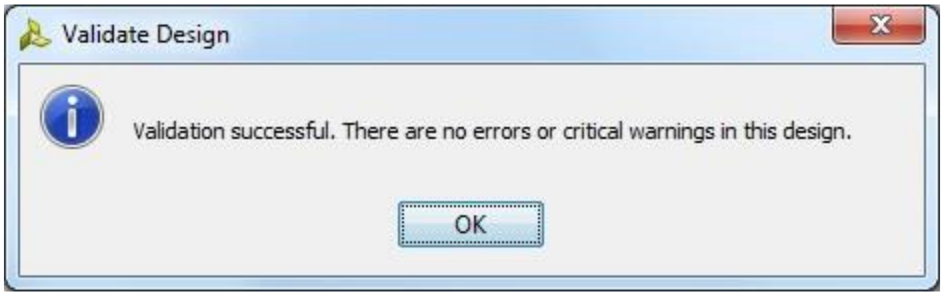
51. **Double-click** the fmc_imageon_hdmio_rgb sub-module.

52. **Double-click** the v_tc_0 core.
53. Set the **Enable Detection** to **disabled** (unchecked).
54. Click **OK**.

55. Select the **video_vtiming** port.
56. **Right-click**, then select **delete** to remove the unused port.

At this point, validate that you have a correct design.

57. **Right-click** in a blank portion of the block design, then select **Validate Design**
58. If successful, Click **OK** and proceed with the next section.
    Otherwise, fix any errors in the design before proceeding.



Validation successful. There are no errors or critical warnings in this design.

Figure 17 – Validate Design

Save the block diagram

59. In the menu, select **File** => **Save Block Design**

Build the hardware with Vivado Design Suite

Create the top level HDL file.

1. In the **Sources** tab, select the block diagram
2. **Right-click**, then select **Create HDL Wrapper**

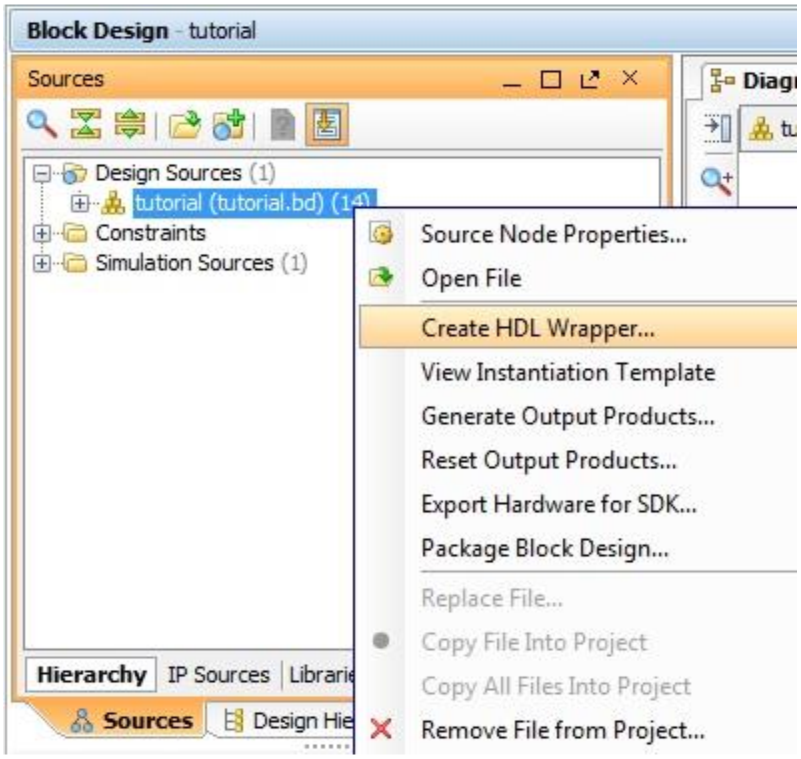
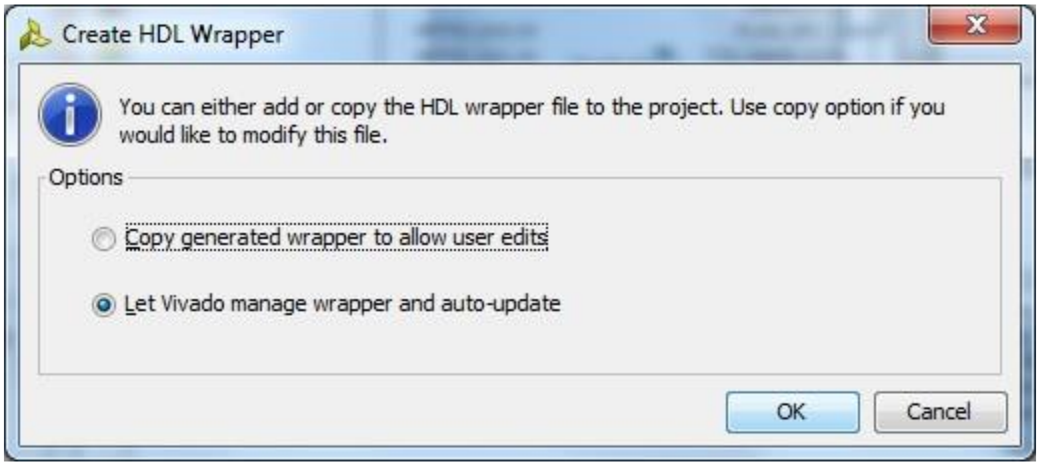


Figure 18 – Create HDL Wrapper

3. When asked whether to add or copy the HDL wrapper, select **Let Vivado manage wrapper and auto-update**.


Create HDL Wrapper

You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.

Options

Copy generated wrapper to allow user edits [✓ Let Vivado manage wrapper and auto-update]

Figure 19 – Let Vivado manage wrapper and auto-update

Add the XDC constraints file.

4. In the Flow Navigator, under the Project Manager section, click **Add Sources**.
5. Select the **Add or Create Constraints** option
6. Click **Next**
7. In the dialog box that opens, click the **Add Files …** button to add an existing XDC file
8. Select one of the following XDC files, depending on your hardware:
    a. For the ZC702:
        ..\constraints\zc702_fmc_imageon_hdmi_display.xdc
    b. For the ZedBoard:
        ..\constraints\zedboard_fmc_imageon_hdmi_display.xdc

    Once selected, click **OK**
9. Click **Finish**.

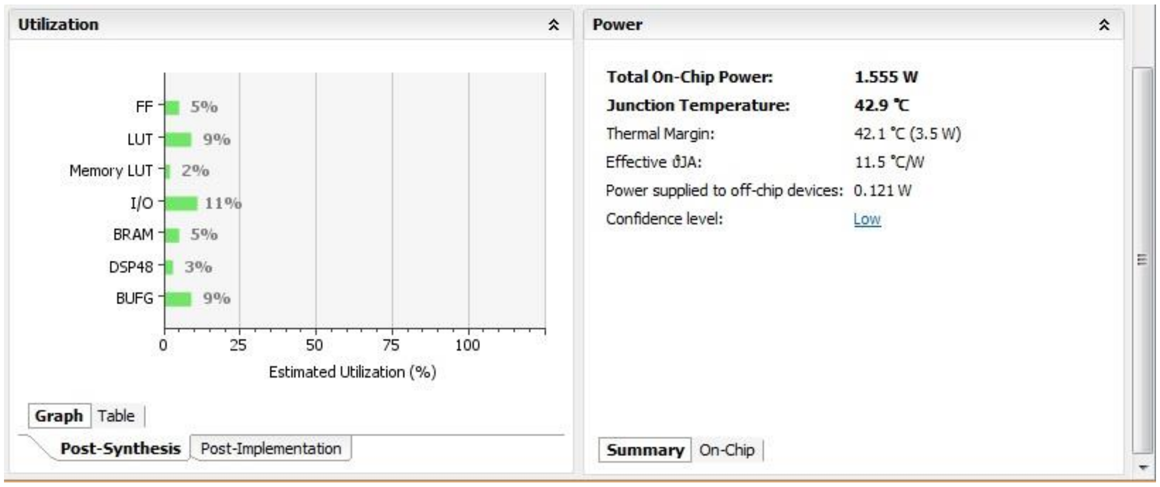
Build the bitstream.

10. In the Flow Navigator, under the Program and Debug section, click **Generate Bitstream**. A dialog box appears asking whether all the processes starting for synthesis should be done.
11. Click **Yes**.

The “Bitstream Generation Completed” dialog box will open, asking what to do Next.

12. Select **Open Implemented Design**
13. Click **OK**.

The resource utilization and power estimation for this design can be seen in the Project Summary. Note that results may be different depending on the video format chosen, and for different carriers.



Total On-Chip Power:	**1.555 W**
Junction Temperature:	**42.9 ℃**
Thermal Margin:	42.1 ℃ (3.5 W)
Effective ϑJA:	11.5 ℃/W
Power supplied to off-chip devices:	0.121 W
Confidence level:	Low

Figure 20 – HDMI Display Controller – Resource Utilization

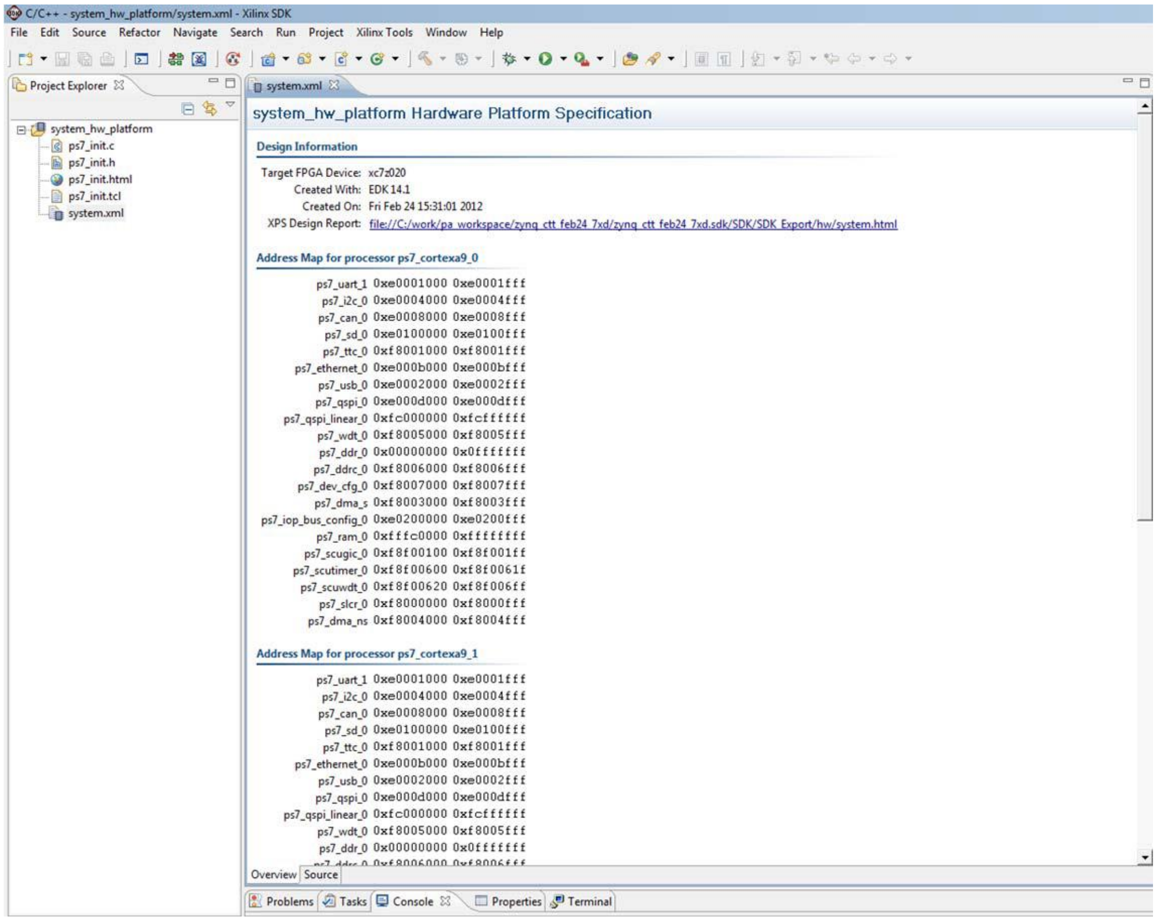
You have successfully created the hardware design !

Create the Embedded Software Application with SDK

Launch SDK from Vivado Design Suite.

1. In the Vivado menu, Select **File > Export > Export Hardware for SDK**.
   The “Export Hardware for SDK” dialog box opens.
   By default, the “Include Bitstream” and “Export Hardware” check boxes are checked.
2. Check the **Launch SDK** check box.
3. Click **OK**. SDK opens.

Notice that when SDK launched, the hardware description file was automatically read in. The system.xml tab shows the address map for the entire Processing System.



system_hw_platform Hardware Platform Specification

Design Information

Target FPGA Device: xc7z020
Created With: EDK 14.1
Created On: Fri Feb 24 15:31:01 2012
XPS Design Report: file://C:/work/pa_workspace/zynq_ctt_feb24_7xd/zynq_ctt_feb24_7xd.sdk/SDK/SDK_Export/hw/system.html

Address Map for processor ps7_cortexa9_0

```
         ps7_uart_1 0xe0001000 0xe0001fff
          ps7_i2c_0 0xe0004000 0xe0004fff
          ps7_can_0 0xe0008000 0xe0008fff
           ps7_sd_0 0xe0100000 0xe0100fff
          ps7_ttc_0 0xf8001000 0xf8001fff
     ps7_ethernet_0 0xe000b000 0xe000bfff
          ps7_usb_0 0xe0002000 0xe0002fff
         ps7_qspi_0 0xe000d000 0xe000dfff
  ps7_qspi_linear_0 0xfc000000 0xfcffffff
          ps7_wdt_0 0xf8005000 0xf8005fff
          ps7_ddr_0 0x00000000 0x0fffffff
         ps7_ddrc_0 0xf8006000 0xf8006fff
      ps7_dev_cfg_0 0xf8007000 0xf8007fff
          ps7_dma_s 0xf8003000 0xf8003fff
ps7_iop_bus_config_0 0xe0200000 0xe0200fff
          ps7_ram_0 0xfffc0000 0xffffffff
       ps7_scugic_0 0xf8f00100 0xf8f001ff
     ps7_scutimer_0 0xf8f00600 0xf8f0061f
       ps7_scuwdt_0 0xf8f00620 0xf8f006ff
         ps7_slcr_0 0xf8000000 0xf8000fff
         ps7_dma_ns 0xf8004000 0xf8004fff
```

Address Map for processor ps7_cortexa9_1

```
         ps7_uart_1 0xe0001000 0xe0001fff
          ps7_i2c_0 0xe0004000 0xe0004fff
          ps7_can_0 0xe0008000 0xe0008fff
           ps7_sd_0 0xe0100000 0xe0100fff
          ps7_ttc_0 0xf8001000 0xf8001fff
     ps7_ethernet_0 0xe000b000 0xe000bfff
          ps7_usb_0 0xe0002000 0xe0002fff
         ps7_qspi_0 0xe000d000 0xe000dfff
  ps7_qspi_linear_0 0xfc000000 0xfcffffff
          ps7_wdt_0 0xf8005000 0xf8005fff
          ps7_ddr_0 0x00000000 0x0fffffff
```

Figure 21 – Address Map in SDK system.xml Tab

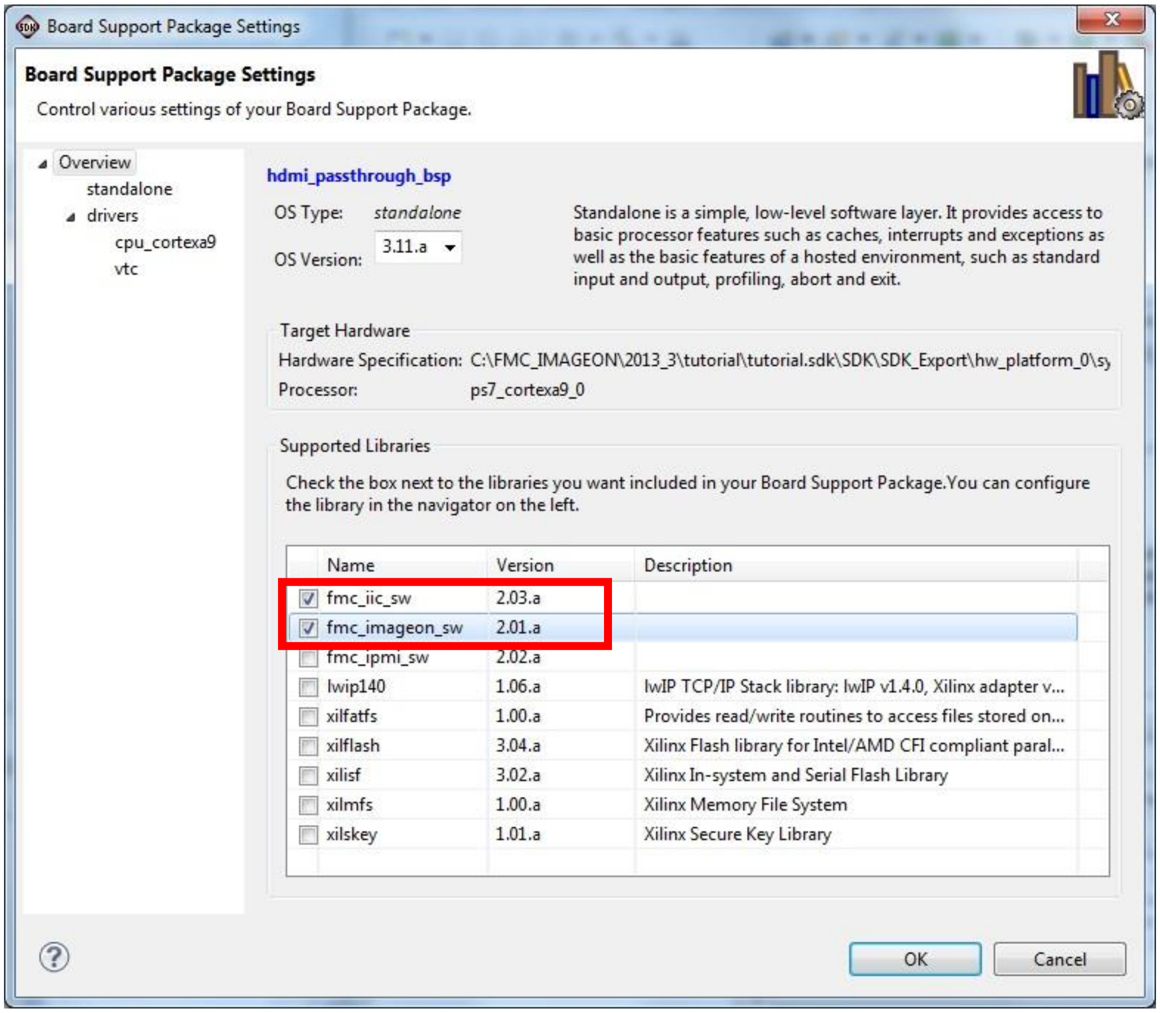
The **avnet_fmc_imageon_cores** directory contains some software drivers that we will use in this design. In order for the project to recognize the contents of this directory, the path must be added to the project repositories, as described below.

4. In the SDK menu, select **Xilinx Tools** => **Repositories**
   The Preferences dialog box opens.

5. In the Local Repositories section, click on the **New …** button.
6. Select the **C:\FMC_IMAGEON\2013_3\avnet_fmc_imageon_cores** directory, then click **OK**
7. Click **OK** in the Preferences dialog box.

Create a standalone BSP (board support package).

8. In the SDK menu, select **File** => **New** => **Board Support Package**.
   The New Board Support Package Project dialog box opens.
9. In the **Project name** field, type "**hdmi_display_bsp**".
10. Keep the default settings, and click **Finish**.
    The Board Support Package Settings dialog box opens.
11. In the Supported Libraries, select the `fmc_iic_sw` and `fmc_imageon_sw` libraries.



Board Support Package Settings

Control various settings of your Board Support Package.

hdmi_passthrough_bsp

OS Type: standalone

OS Version: 3.11.a

Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Target Hardware

Hardware Specification: C:\FMC_IMAGEON\2013_3\tutorial\tutorial.sdk\SDK\SDK_Export\hw_platform_0\sy

Processor: ps7_cortexa9_0

Supported Libraries

Check the box next to the libraries you want included in your Board Support Package.You can configure the library in the navigator on the left.

	Name	Version	Description
✓	fmc_iic_sw	2.03.a	
✓	fmc_imageon_sw	2.01.a	
☐	fmc_ipmi_sw	2.02.a	
☐	lwip140	1.06.a	lwIP TCP/IP Stack library: lwIP v1.4.0, Xilinx adapter v...
☐	xilfatfs	1.00.a	Provides read/write routines to access files stored on...
☐	xilflash	3.04.a	Xilinx Flash library for Intel/AMD CFI compliant paral...
☐	xilisf	3.02.a	Xilinx In-system and Serial Flash Library
☐	xilmfs	1.00.a	Xilinx Memory File System
☐	xilskey	1.01.a	Xilinx Secure Key Library

Figure 22 – Board Support Package Settings

12. Click **OK**.
    If the Build Automatically setting is enabled, SDK will automatically build the standalone BSP.

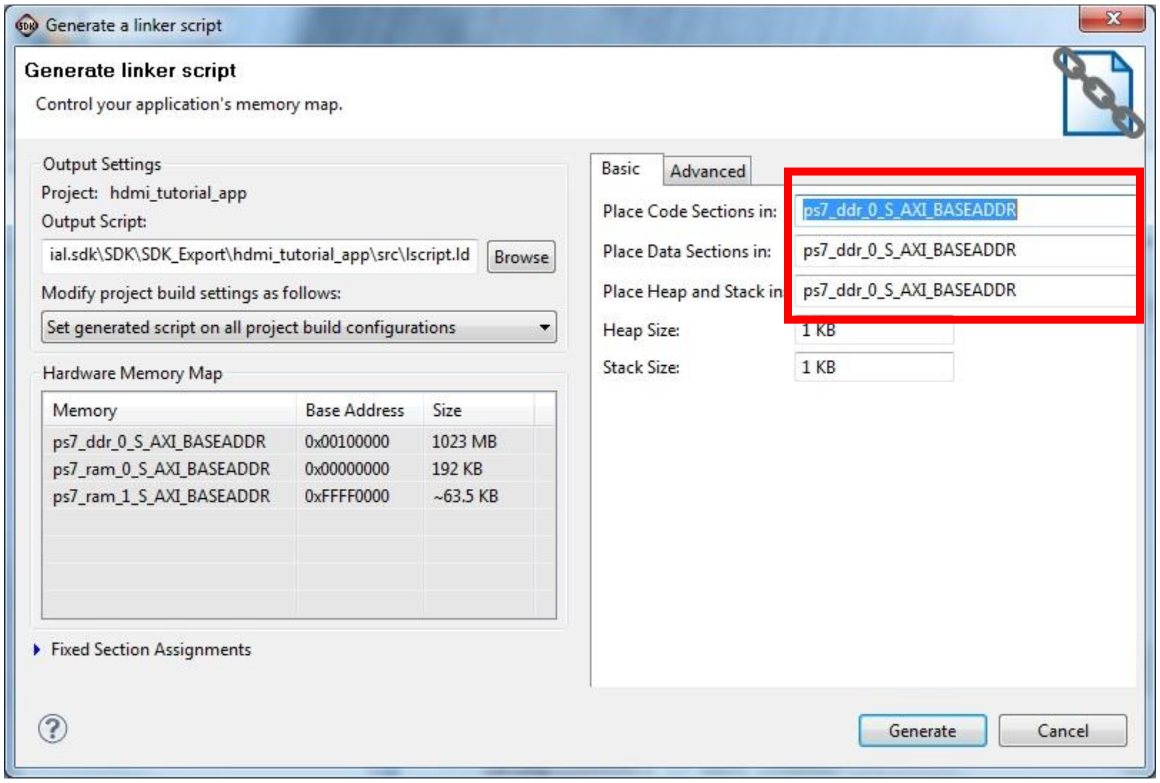
Create a new C project.

13. In the SDK menu, select **File** => **New** => **Application Project**.
    The Application Project dialog box opens.
14. In the **Project Name** field, type "**hdmi_display_app**".
15. For the **Board Support Package**, select **Use Existing**, then select the BSP that was created previously.
16. Click **Next**.
    The Templates dialog box opens.
17. Select the **Hello World** template.
18. Click **Finish**.

Configure the application's memory map to execute from external memory.

19. Right-click on the **hdmi_display_app** application
20. Select **Generate Linker Script**
    This opens the Generate a linker script dialog box.

21. Select the **ps7_ddr_0** memory for each of the Code, Data, Heap and Stack sections.



Output Settings
Project: hdmi_tutorial_app
Output Script: ial.sdk\SDK\SDK_Export\hdmi_tutorial_app\src\lscript.ld
Modify project build settings as follows: Set generated script on all project build configurations

Hardware Memory Map

Memory	Base Address	Size
ps7_ddr_0_S_AXI_BASEADDR	0x00100000	1023 MB
ps7_ram_0_S_AXI_BASEADDR	0x00000000	192 KB
ps7_ram_1_S_AXI_BASEADDR	0xFFFF0000	~63.5 KB

Place Code Sections in: ps7_ddr_0_S_AXI_BASEADDR
Place Data Sections in: ps7_ddr_0_S_AXI_BASEADDR
Place Heap and Stack in: ps7_ddr_0_S_AXI_BASEADDR
Heap Size: 1 KB
Stack Size: 1 KB

Figure 23 – Generate a Linker Script

22. Click **Generate**.
    A dialog box appears asking whether it is OK to overwrite the existing linker script file.
23. Click **Yes**

Import the provided example source files for the hdmi_display

1. In the Project Explorer window, select the **hdmi_display_app** application
2. Right-click, then select **Import** from the pop-up menu.
   The Import wizard appears.
3. Expand the **General** section
4. Select **File System**, then click **Next**.

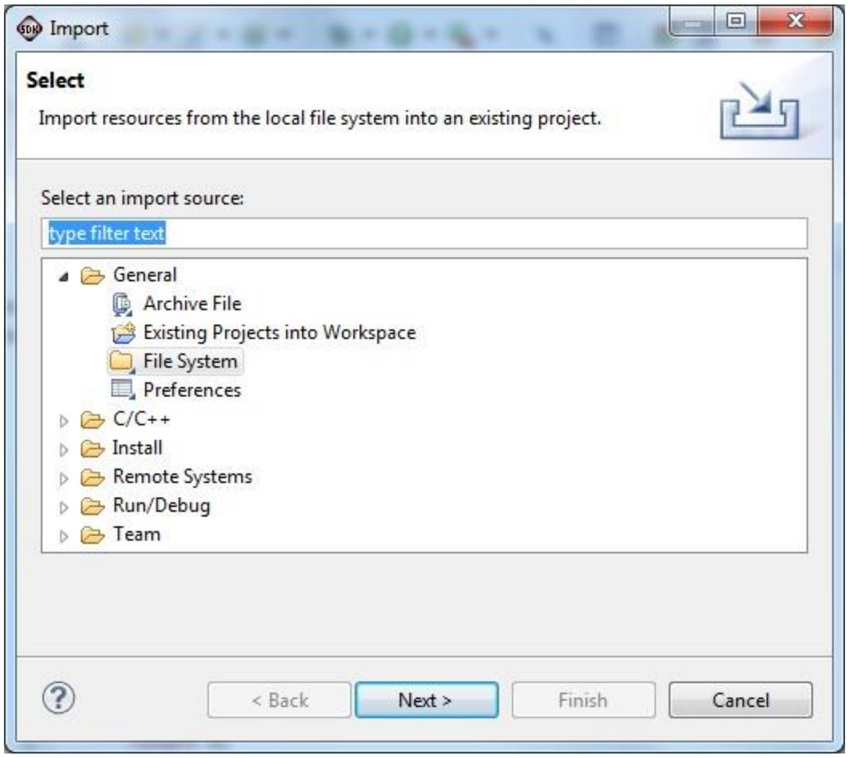


Figure 24 – Import from File System – Dialog 1

The next dialog of the Import wizard appears.

5. Next to the **From directory** field, click the **Browse** button
6. Specify the following directory:
   C:\FMC_IMAGEON\2013_3\code\fmc_imageon_hdmi_display
7. Click **OK**

8. Select the following source files:
   fmc_imageon_hdmi_display.c
   fmc_imageon_hdmi_display.h
   video_resolution.c
   video_resolution.h
   video_generation.c

video_generation.h
video_frame_buffer.c
video_frame_buffer.h

9. Next to the **Into directory** field, click the **Browse** button
10. Specify the following directory : **hdmi_display_app\src**, then click **OK**

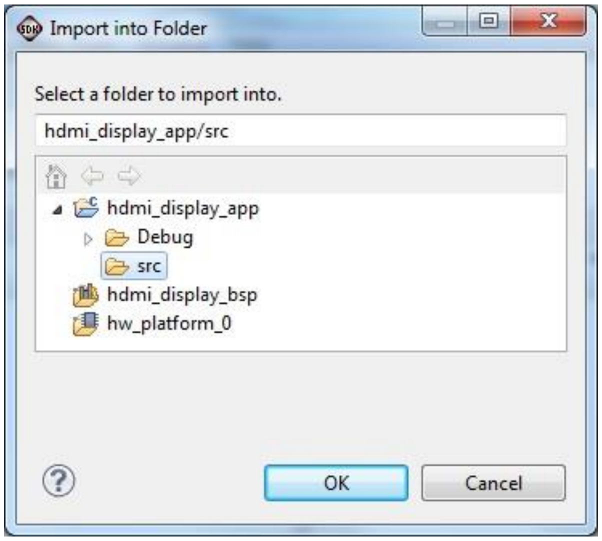
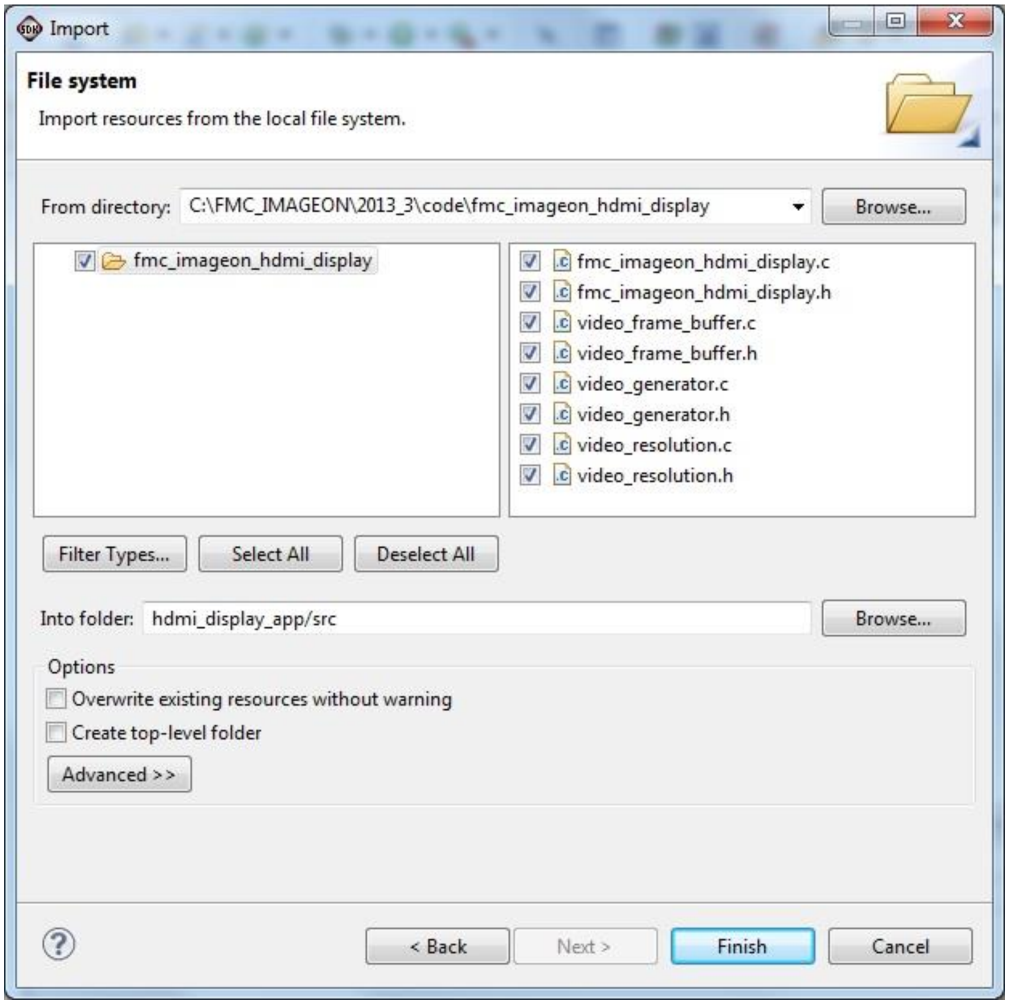


Figure 25 – Import into Folder



From directory: C:\FMC_IMAGEON\2013_3\code\fmc_imageon_hdmi_display

☑ fmc_imageon_hdmi_display

☑ fmc_imageon_hdmi_display.c
☑ fmc_imageon_hdmi_display.h
☑ video_frame_buffer.c
☑ video_frame_buffer.h
☑ video_generator.c
☑ video_generator.h
☑ video_resolution.c
☑ video_resolution.h

Into folder: hdmi_display_app/src

☐ Overwrite existing resources without warning
☐ Create top-level folder

Figure 26 – Import from File System – Dialog 2

11. Click **Finish**.

Modify the hello world application

12. Open the helloworld.c file and edit the source code as follows:

```
/*
 * helloworld.c: simple test application
 */

#include <stdio.h>
#include "platform.h"

#include "fmc_imageon_hdmi_display.h"
fmc_imageon_hdmi_display_t demo;

//void print(char *str);
```

strange bug:

when vtc driver is active, need to modify print declaration to match the one in xil_printf.h

```
void print( const char *str);

int main()
{
    init_platform();

    print("Hello World\n\r");

    demo.uBaseAddr_IIC_FmcImageon = XPAR_FMC_IMAGEON_IIC_0_BASEADDR;
    demo.uDeviceId_VTC_HdmioGenerator =
        XPAR_FMC_IMAGEON_HDMIO_RGB_V_TC_0_DEVICE_ID;
    demo.uDeviceId_VDMA_HdmiDisplay = XPAR_AXI_VDMA_0_DEVICE_ID;
    demo.uBaseAddr_MEM_HdmiDisplay = XPAR_DDR_MEM_BASEADDR + 0x10000000;
    demo.uNumFrames_HdmiDisplay = XPAR_AXIVDMA_0_NUM_FSTORES;
    fmc_imageon_hdmi_display_init( &demo );

    cleanup_platform();

    return 0;
}
```

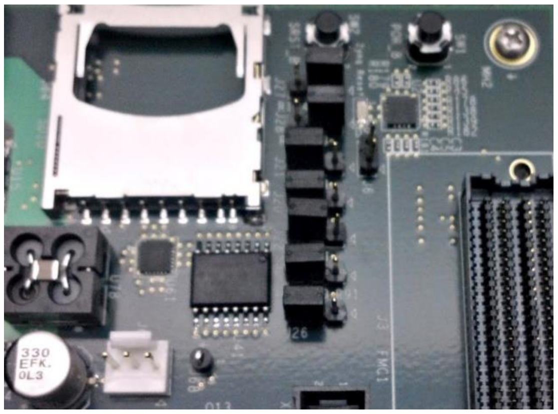
13. If the Build Automatically setting is enabled, SDK will automatically build the application. If not, right-click on the application and select **Build Project** to build the application.

You have successfully created the software application !

Set up your ZC702 Hardware

Setup your ZC7020-based hardware, as described below.

1. Set the ZC702 board's boot mode to cascaded JTAG using jumpers
   a. J21,J20,J22,J25,J26 should all be set to '0'
   b. J27,J28 should be set to '1'



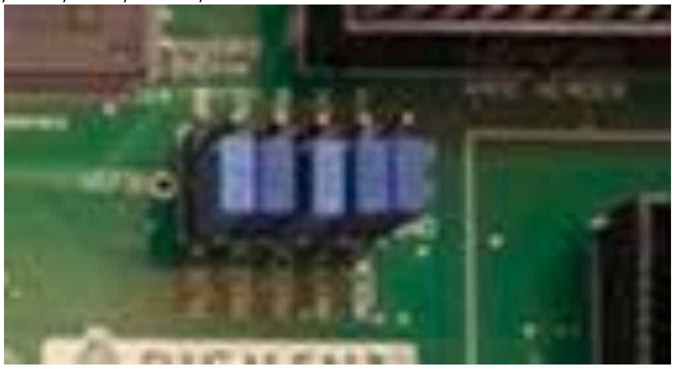
2. Connect a mini USB cable to the ZC702's USB-UART connector (J17)
3. Connect one of the following JTAG connections:
   a. Connect platform USB pod to the ZC702's JTAG header (J2) and set SW10 to '10'
   b. Connect a micro USB cable to the ZC702's on-board Digilent JTAG module and set SW10 to '01'
4. Populate the FMC-IMAGEON board on FMC Slot #2.
5. Connect a DVI or HDMI monitor to the FMC module's HDMI OUT connector

6. Power on the ZC702 board
7. Open a serial communication utility for the COM port assigned on your system.
   Note: The standard configuration for Zynq Processing System is baud rate 115200, 8 bit, parity

Set up your ZedBoard Hardware

Setup your ZedBoard hardware, as described below.

1. Set the ZedBoard's boot mode to cascaded JTAG using jumpers
   a. JP7, JP8, JP9, JP10, JP11 should all be set to '0'



2. Connect a micro USB cable to the ZedBoard's USB-UART connector (J14)
3. Connect one of the following JTAG connections:
   a. Connect platform USB pod to the ZedBoard's JTAG header (J15)
   b. Connect a micro USB cable to the ZedBoard's on-board Digilent JTAG connector (J17)
4. Populate the FMC-IMAGEON board on FMC Slot #1.
5. Connect a DVI or HDMI monitor to the fMC module's HDMI OUT connector
6. Power on the ZedBoard board
7. Open a serial communication utility for the COM port assigned on your system.
   Note: The standard configuration for Zynq Processing System is baud rate 115200, 8 bit, parity

Execute the HDMI Display Controller Design on Hardware using SDK

From SDK, configure the FPGA bitstream and launch the application.

1. In the SDK menu, select **Xilinx Tools** => **Program FPGA**
   The "Program FPGA" dialog opens.
2. Make sure the path to the bitstream is valid
   (HINT : If you moved the project, you will need to update the path to the bitstream file)
3. Click **Program**.
   It will take approximately 10 seconds to program the bitstream to hardware
4. Right-click **hdmi_display_app**
   and select **Run as > Run Configurations**
5. Click **Xilinx C/C++ ELF** and click **New launch configurations**.

6. The new run configuration is created named **hdmi_display_app Debug**.
   The configurations associated with application are pre-populated in the main tab of these launch configurations.
7. Click the **Device Initialization** tab in the launch configurations and check the settings here.
   Notice that there is a configuration path to the initialization TCL file. The path of `ps7_init.tcl` is mentioned here. This is file that was exported when you exported your design to SDK; it contains the initialization information for the processing system.
   (HINT : If you moved the project, you should delete the previous run configuration and create a new one)
8. The STDIO Connection tab is available in the launch configurations settings. You can use this to have your STDIO connected to the console. We will not use this now because we have already launched a serial communication utility. There are more options in launch configurations but we will focus on them later.
9. Click **Apply** and then **Run**.
10. If you get a Reset Status dialog box indicating that the current launch will reset the entire system, click **OK**.
11. You should see something similar to the following on your serial console:

```
Hello World

------------------------------------------------------
--        FMC-IMAGEON HDMI Display Controller        --
------------------------------------------------------

HDMI IIC Initialization ...
HDMI Output Initialization ...
Clear Frame Buffer
Video Frame Buffer Initialization ...
Video DMA (Output Side) Initialization ...
Video Timing Controller (generator) Initialization ...
        Video Resolution = 1080P
Generate Color Bars
HDMI Output Re-Initialization ...

Done


Press ENTER to re-start ...
```

To shift the color bar pattern, press ENTER.

You have successfully executed the HDMI display controller on hardware !

References

All documentation supporting the ON Semiconductor Image Sensor with HDMI Input/Output FMC Bundle is available on the Avnet Design Resource Center (DRC):
http://www.em.avnet.com/fmc-imageon-v2000c

1. Getting Started with the HDMI Input/Output FMC Module
   http://www.em.avnet.com/fmc-imageon ➔ Support Files & Downloads
2. Avnet FMC-IMAGEON – Hardware User Guide
   http://www.em.avnet.com/fmc-imageon ➔ Support Files & Downloads

3. Getting Started with the ON Semiconductor Image Sensor with HDMI Input/output FMC Bundle
   http://www.em.avnet.com/fmc-imageon-v2000c ➔ Support Files & Downloads

The following reference provides links to documentation for video intellectual property (IP).

4. Video and Image Processing IP
   http://www.xilinx.com/ipcenter/video/video_core_listing.htm
5. Video Timing Controller
   http://www.xilinx.com/products/intellectual-property/EF-DI-VID-TIMING.htm
6. Video Input to AXI4-Stream
   http://www.xilinx.com/products/intellectual-property/video_in_to_axi4_stream.htm
7. AXI4-Stream to Video Output
   http://www.xilinx.com/products/intellectual-property/axi4_stream_to_video_out.htm
8. AXI Video DMA
   http://www.xilinx.com/products/intellectual-property/axi_video_dma.htm

The following reference provides links to documentation for AXI interconnect.

9. UG761 - AXI Reference Guide
10. PG065 – AXI4-Stream Infrastructure

Known Issues and Limitations

The following issues are known to exist. When applicable, the workaround used is described.

Hello World Template – error: conflicting types for ‘print’

The “Hello World” C project template has an issue that may manifest itself depending on which drivers are included in the design.

The “print(….)” declaration does not match the declaration in the xil_printf.h file and will result in the following error:

```
helloworld.c:29:6: error: conflicting types for ‘print’
xil_printf.h:39:6: note: previous declaration of ‘print’ was here
```

The solution is to simply fix the “print(…)” declaration as shown below.

```
//void print(char *str);
void print( const char *ptr);
```

Troubleshooting