

Overview

This reference design is the getting started design for the PYTHON-1300-C camera module. The camera module features ON Semiconductor's PYTHON-1300 color image sensor. The PYTHON-1300 is a 1/2 inch Super-eXtended Graphics Array (SXGA) CMOS image sensor with a pixel array of 1280 by 1024 pixels. Designed to address the needs of general purpose industrial image sensing applications, the new global shutter image sensor combines flexibility in configuration and resolution with high speed and high sensitivity for the industrial imaging market.

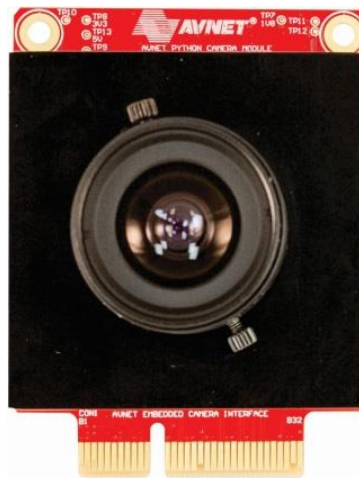


Figure 1 – PYTHON-1300-C Camera Module

Objectives

This tutorial will guide the user how to:

- Retrieve the design files from the public Avnet git repository
- Build the reference design
- Execute the reference design on hardware

Reference Design Overview

The example design uses the Zynq processing system (PS) to initialize the PYTHON-1300-C camera, the HDMI input interface, as well as the HDMI output interface. The design also implements a simple image sensor pipeline (ISP) and video frame buffer inside the programmable logic (PL).

The following figure illustrates the block diagram for the programmable logic (PL) hardware implementation.

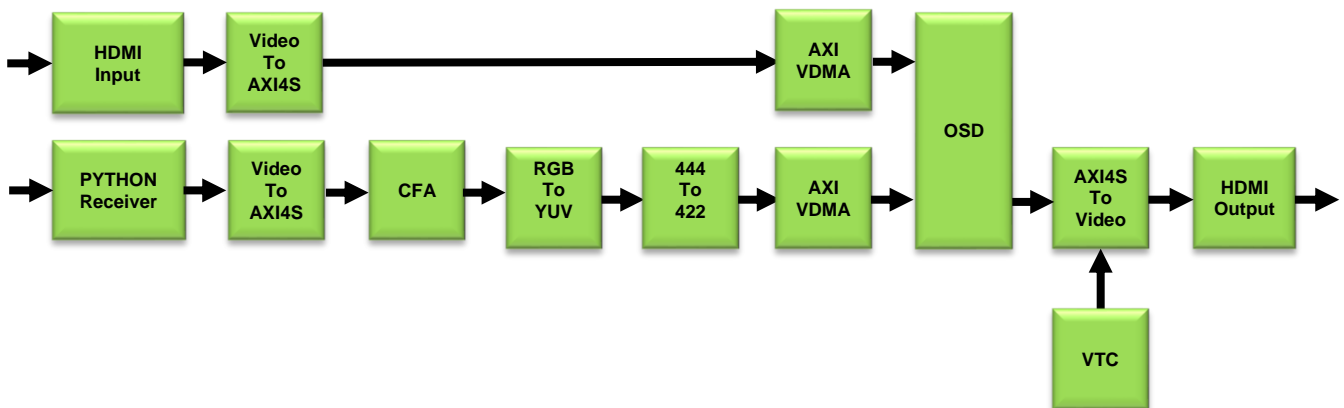


Figure 2 – FMC-HDMI-CAM + PYTHON-1300 Camera Reference Design – Hardware Block Diagram

Valid licenses (hardware evaluation, or full license) are required for the following video IP cores:

- Color Filter Array Interpolation (CFA) v7.0
- Chroma Resampler v4.0
- Video On Screen Display (OSD) v6.0
- RGB to YcrCb Color-Space Converter v7.1
- Video Timing Controller (VTC) v6.1

Experiment Setup

This tutorial makes use of Xilinx Vivado Design Suite in scripting mode in order to create a project. The resulting project can be opened with the graphical (GUI) version of the tools for further analysis and modification.

Software

The software required to build, and execute the reference design is:

- Windows-7 64-bit
- Terminal Emulator (HyperTerminal or TeraTerm)
- Xilinx Vivado Design Suite 2014.4
- MicroZed Board Definition Install for Vivado 2014.4
 - <http://www.microzed.org/support/documentation/1519>

Hardware

The hardware required to build, and execute the reference design is:

- Win-7 PC with a recommended 2 GB RAM available for the Xilinx tools to complete a XC7Z020 design¹
- One of the following supported FMC carriers:
 - ZC702
 - ZedBoard
 - MicroZed 7020 SOM + FMC Carrier Card
- ON Semiconductor PYTHON-1300-C Camera Module (optional)
- DVI or HDMI video source
- HDMI (or DVI-D) monitor (1080P60 capable)
- USB cable (Type A to Micro-USB Type B)
- 4GB MicroSD card

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

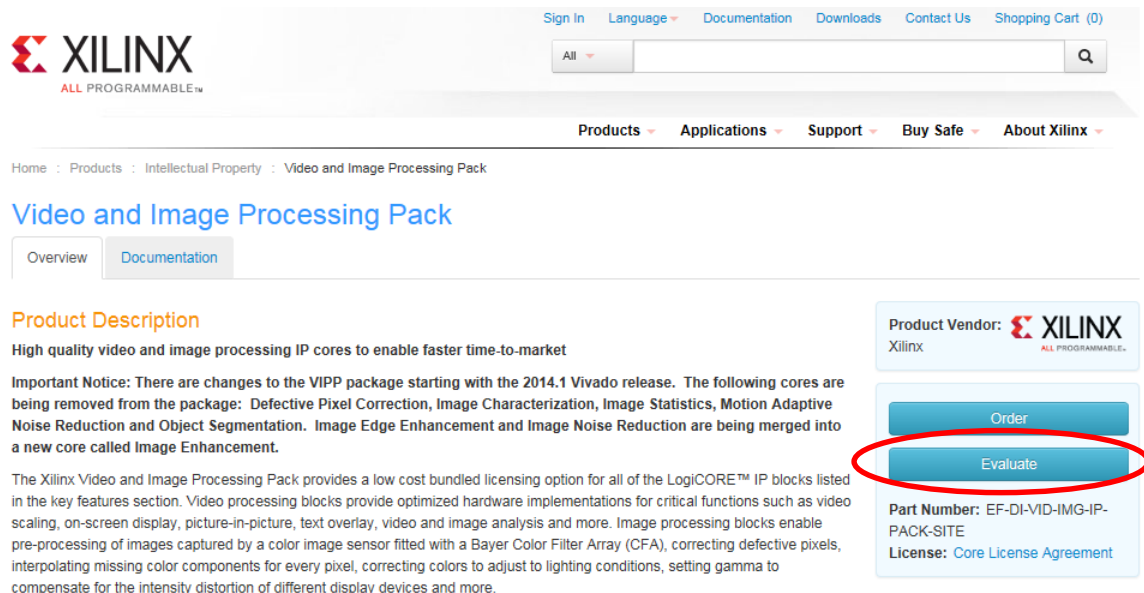
Experiment 1: Licensing the Video and Image Processing Pack IP Cores

This reference design uses several of the Xilinx Video and Image Processing Pack IP cores. In order to build the hardware design, valid licenses (hardware evaluation, or full license) are required for the following video IP cores:

- Color Filter Array Interpolation (CFA) v7.0
- Chroma Resampler v4.0
- Video On Screen Display (OSD) v6.0
- RGB to YcrCb Color-Space Converter v7.1
- Video Timing Controller (VTC) v6.1

Follow these steps to request an evaluation license:

1. Navigate to the “Video and Image Processing Pack” product page on the Xilinx web site :
<http://www.xilinx.com/products/intellectual-property/ef-di-vid-img-ip-pack.html>
2. Click the Evaluate link located on the right of the web page, and follow the online instructions



Sign In Language Documentation Downloads Contact Us Shopping Cart (0)

XILINX
ALL PROGRAMMABLE

Products Applications Support Buy Safe About Xilinx

Home : Products : Intellectual Property : Video and Image Processing Pack

Video and Image Processing Pack

Overview Documentation

Product Description

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

Important Notice: There are changes to the VIPP package starting with the 2014.1 Vivado release. The following cores are being removed from the package: Defective Pixel Correction, Image Characterization, Image Statistics, Motion Adaptive Noise Reduction and Object Segmentation. Image Edge Enhancement and Image Noise Reduction are being merged into a new core called Image Enhancement.

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 processing blocks provide optimized hardware implementations for critical functions such as video scaling, on-screen display, picture-in-picture, text overlay, video and image analysis and more. Image processing blocks enable pre-processing of images captured by a color image sensor fitted with a Bayer Color Filter Array (CFA), correcting defective pixels, interpolating missing color components for every pixel, correcting colors to adjust to lighting conditions, setting gamma to compensate for the intensity distortion of different display devices and more.

Product Vendor: XILINX
Xilinx

Order Evaluate

Part Number: EF-DI-VID-IMG-IP-PACK-SITE
License: Core License Agreement

Figure 3 – Video and Image Processing Pack – product page

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.

Experiment 2: Retrieve the design files

In this section, the design files for the reference design will be retrieved from the Avnet git repository.

1. Navigate to the following web site : <https://github.com/Avnet/hdl>
2. Click the **branch:master** button
3. Specify the following search criteria : fmchc_python
4. Click the **Tags** tab

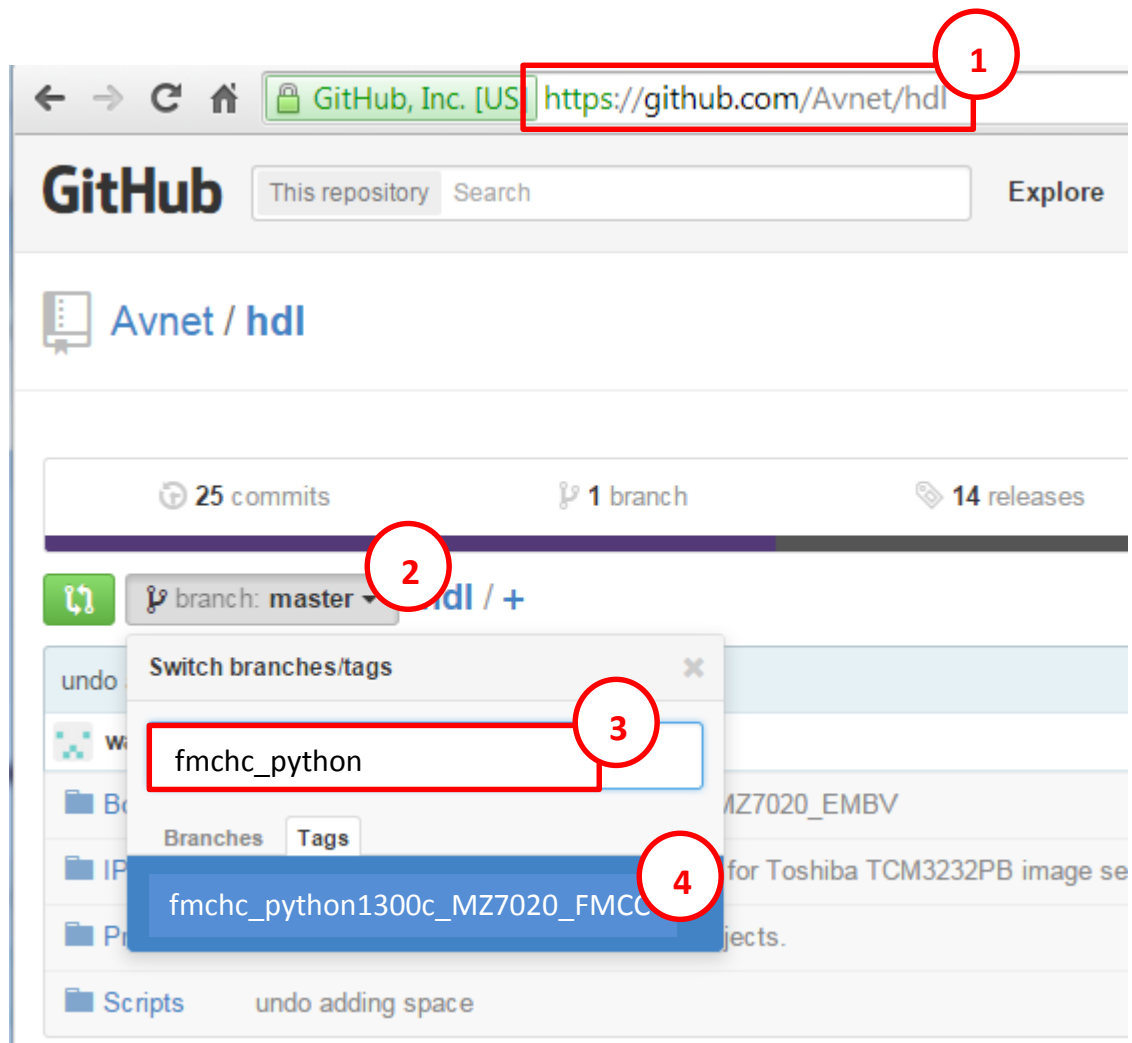


Figure 4 – Avnet GitHub repository – Retrieving specific version with tag

5. Select the **fmchc_python1300c_MZ7020_FMCCC_20150907_211338** tag
This will retrieve a known working version of the design files for the FMC-HDMI-CAM + PYTHON-1300-C reference design.

6. Click the Download ZIP file button

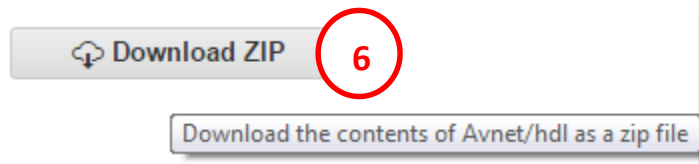


Figure 5 – Avnet GitHub repository – Download ZIP

7. Create an “Avnet” directory in your root C:\ drive
8. Save **hdl-fmchc_python1300c_MZ7020_FMCCC_20150907_211338.zip** file to the **C:\Avnet** directory, and extract the contents of the zip file in this directory
9. Rename the “hdl-fmchc_python1300c_MZ7020_FMCCC_20150907_211338” directory to “hdl”

You should see the following directory structure

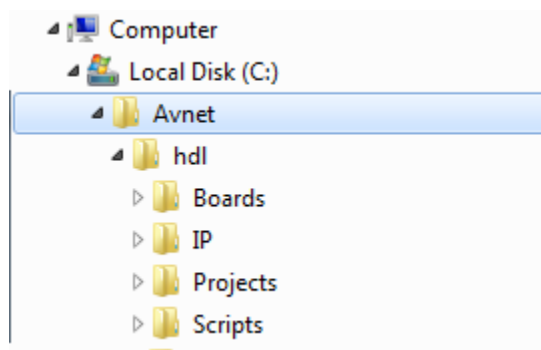


Figure 6 – Extracted C:\Avnet\hdl directory structure

NOTE : the exact directory name is not critical, but it must remain short on Windows machines, due to the directory length limitation of Windows

The **C:\Avnet\hdl** repository contains the following sub-directories:

Directory	Content Description
C:\Avnet\hdl\Boards	contains board related files
C:\Avnet\hdl\IP	contains the IP cores used by the ref designs
C:\Avnet\hdl\Projects	contains project related files
C:\Avnet\hdl\Scripts	contains scripts used to automatically build the designs

For the PYTHON-1300-C reference design, the following content is of interest:

Directory	Content Description
C:\Avnet\hdl\IP\avnet_hdmi_in C:\Avnet\hdl\IP\avnet_hdmi_out	IP cores (including HDL source) for the HDMI input/output interfaces including embedded sync code insertion/detection
C:\Avnet\hdl\IP\onsemi_vita_spi	IP core (including HDL source) for the SPI controller for use with the VITA/PYTHON image sensors
C:\Avnet\hdl\IP\onsemi_vita_cam	IP core (including HDL source) for the VITA/PYTHON camera receiver
C:\Avnet\hdl\Projects\embv_python1300c_fb	files for PYTHON-1300-C reference design
C:\Avnet\hdl\Scripts\make_embv_python1300c_fb.tcl	script to build PYTHON-1300-C reference design for MicroZed 7020

By default, the script will build the design for the ZC702, ZEDBOARD, and MicroZed7020 + FMC Carrier Card.

1. Edit the **make_fmchc_python1300c.tcl** script to only build for your FMC carrier.
As an example, if you have a MicroZed 7020 SOM + FMC carrier card, comment out the build for the ZC702 and ZEDBOARD, as shown below:

```
# Build FMC-HDMI-CAM + PYTHON-1300-C Getting Started design
# for the ZC702 board
#set argv [list board=ZC702 project=fmchc_python1300c sdk=yes]
#set argc [llength $argv]
#source ./make.tcl -notrace

# Build FMC-HDMI-CAM + PYTHON-1300-C Getting Started design
# for the ZedBoard
#set argv [list board=ZEDBOARD project=fmchc_python1300c sdk=yes]
#set argc [llength $argv]
#source ./make.tcl -notrace

# Build FMC-HDMI-CAM + PYTHON-1300-C Getting Started design
# for the MicroZed-7020 + FMC Carrier Card
set argv [list board=MZ7020_FMCCC project=fmchc_python1300c sdk=yes]
set argc [llength $argv]
source ./make.tcl -notrace
```

Figure 7 – Editing the make script to build for MicroZed FMC Carrier Card

Experiment 3: Build the reference design

In this section, the Vivado project will be created and built with TCL scripts, implementing the FMC-HDMI-CAM + PYTHON-1300-C reference design for the selected FMC carrier.

1. From the Start menu, open the “Vivado 2014.4 TCL Shell” console
2. Change to the **C:\Avnet\hdl\Scripts** directory

```
***** Vivado v2014.4 (64-bit)
**** SW Build 1071353 on Tue Nov 18 18:29:27 MST 2014
**** IP Build 1070531 on Tue Nov 18 01:10:18 MST 2014
** Copyright 1986-2014 Xilinx, Inc. All Rights Reserved.

Vivado% cd C:/Avnet/hdl/Scripts
Vivado%
```

Figure 8 – Vivado 2014.4 TCL Shell – Changing to C:/Avnet/hdl/Scripts directory

3. Launch the build with the “**source ./make_embv_python1300c_fb.tcl**” command

```
Vivado% source ./make_fmchc_python1300c.tcl
# set argv [list board=MZ7020_EMBV project=fmchc_python1300c sdk=yes]
# set argc [llength $argv]
# source ./make.tcl -notrace

*-----*
*-----*
*-
*-      Welcome to the Avnet Project Builder      -
*-
*-----*
*-----*

+-----+-----+
| Setting          | Configuration          |
+-----+-----+
| Board            | MZ7020_EMBV            |
+-----+-----+
| Project          | fmchc_python1300c      |
+-----+-----+
| SDK              | yes                    |
+-----+-----+

Version of Vivado acceptable, continuing...
```

Figure 9 – Vivado 2014.4 TCL Shell – Launching the build

As a convenience, before building the hardware design, the scripts will verify if valid licenses are installed for the video IP cores used in the design.

```
***** Check for Video IP core licenses...

+-----+-----+
| Video IP Core | License Status |
+-----+-----+
| v_cfa         | VALID (Hardware Evaluation) |
+-----+-----+
| v_cresample   | VALID (Hardware Evaluation) |
+-----+-----+
| v_osd         | VALID (Hardware Evaluation) |
+-----+-----+
| v_rgb2ycrcb   | VALID (Full License)        |
+-----+-----+
| v_tc          | VALID (Full License)        |
+-----+-----+
```

Figure 10 – Vivado 2014.4 TCL Shell – Video IP Core license verification

Each of the video IP cores requires a full license or hardware evaluation license in order to successfully build a bitstream.

The build will perform the following steps, where {BOARD} will be one of ZC702, ZEDBOARD, or MZ7020_FMCCC:

- Create and build the hardware design with Vivado 2014.4, including the IP Integrator block design

C:\Avnet\hdl\Projects\fmchc_python1300c\{BOARD}\fmchc_python1300c.xpr

- Create and build the SDK workspace, including board support package (BSP), software application, and first stage boot loader (FSBL)

C:\Avnet\hdl\Projects\fmchc_python1300c\{BOARD}\fmchc_python1300c.sdk

- Create the SD card image (BOOT.bin)

C:\Avnet\hdl\Projects\fmchc_python1300c\{BOARD}\BOOT.bin

Experiment 4: Execute the reference design on hardware

This section describes how to execute the reference design on the hardware.

For instructions on how to setup the hardware, please refer to the FMC carrier's User Guide.

Booting from SD card image

The BOOT.bin SD card image created in the previous experiment can be used to execute the reference design on hardware with the FMC Carrier.

For more detailed instructions on how to boot from the SD card, please refer to the FMC Carrier's User Guide.

Booting from JTAG with SDK

The hardware and software can also be loaded to hardware using SDK 2014.4 and a JTAG emulator.

Set up the hardware as described in the FMC Carrier's User Guide, with the following exceptions:

1. Set the FMC Carrier's boot mode to cascaded JTAG
2. Connect the JTAG Programming Cable (Platform Cable, Digilent HS1 or HS2 cable) to the PC using a USB cable and then plug the 14-Pin PC4 header or cable into the FMC Carrier Card's PC4 connector.
3. Launch SDK 2014.4, and specify the following directory for the SDK workspace:

```
C:\Avnet\hdl\Projects\fmchc_python1300c\{BOARD}\  
fmchc_python1300c.sdk
```

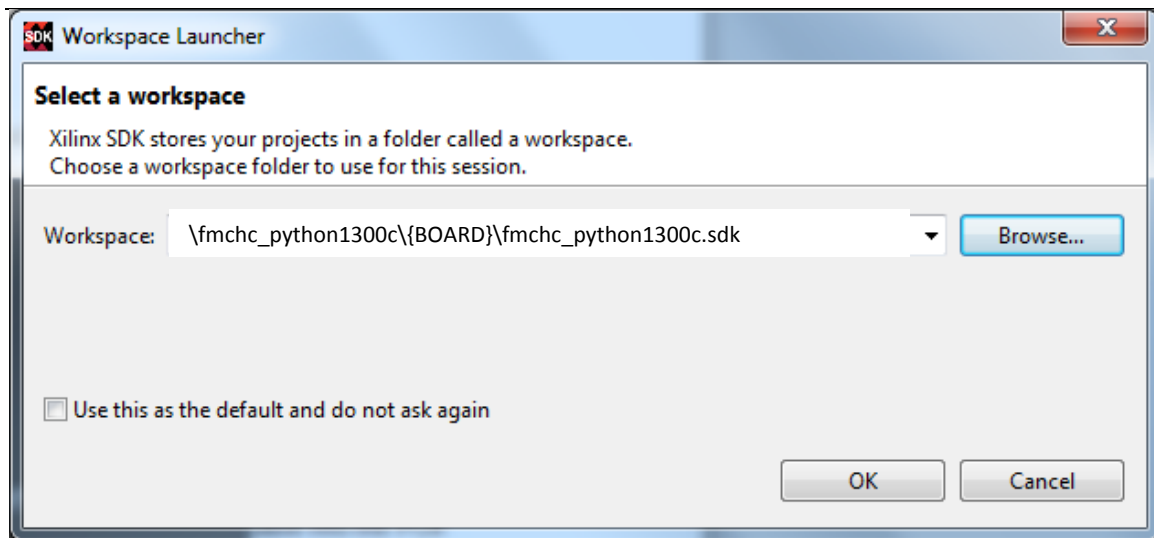


Figure 11 – SDK – Specifying SDK workspace

4. Close the Welcome Window
5. In the SDK menu, select **Xilinx Tools => Repositories**
6. Verify that the following Local Repository is specified:

C:\Avnet\hdl\Projects\fmchc_python1300c\software\sw_repository

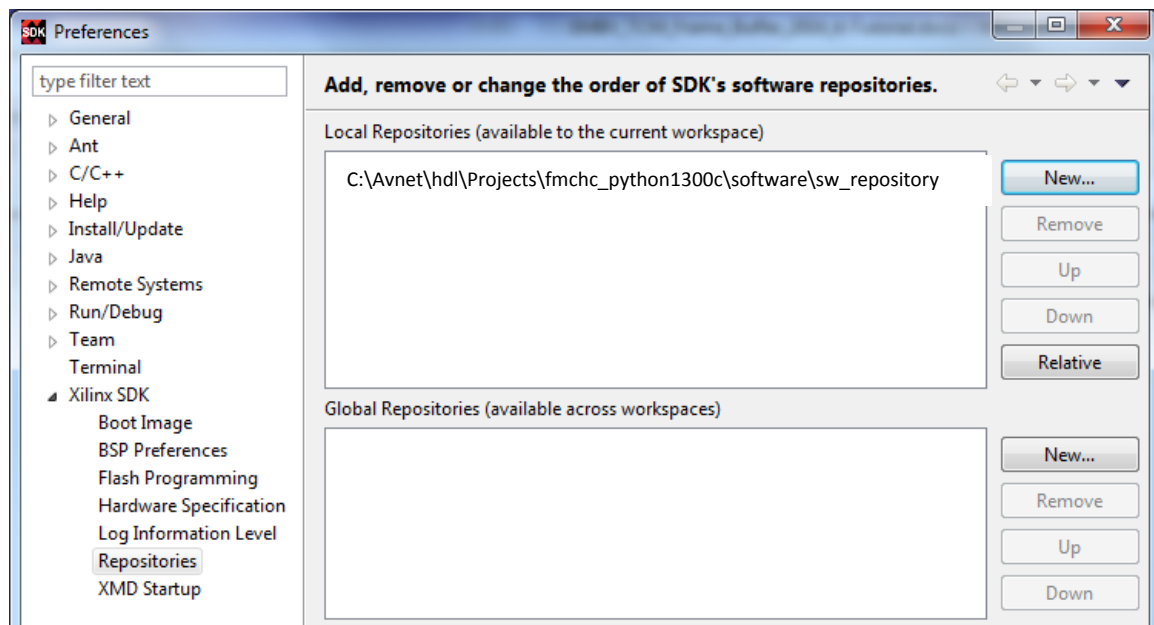


Figure 12 – SDK – Specifying local repository

7. If the local repository is not specified, click on the **New** button, navigate to the following directory, then click **OK**.

C:\Avnet\hdl\Projects\fmhc_python1300c\software\sw_repository

8. When done, click **OK**.

NOTE : The local repository was not saved in the SDK workspace due to a limitation of the SDK's scripting mode.

Now that the SDK workspace is correctly configured, the hardware and software can be loaded and executed on the hardware.

9. In the SDK menu, select **Xilinx Tools => Load FPGA**

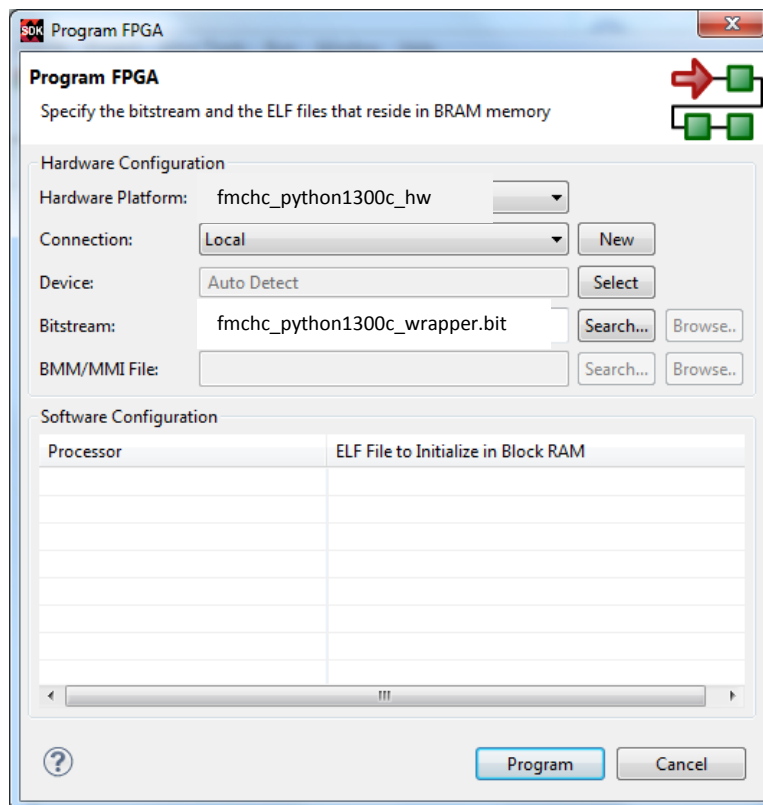


Figure 13 – SDK – Program FPGA

10. Click the **Program** button.
It will take approximately 10 seconds to program the bitstream to hardware

-
11. Right-click **fmchc_python1300c_app**
and select **Run as => Run Configurations.**
 12. Click **Xilinx C/C++ Application (GDB)** and click **New launch configurations.**
 13. The new run configuration is created named **fmchc_python1300c_app Debug**
The configurations associated with application are pre-populated in the main tab
of these launch configurations
 14. Click on the **Application** tab
 15. Next to the Application: edit box, click the **Search** button.
 16. Select the **fmchc_python1300c_app.elf** application, then click **OK.**
 17. Click **Apply** and then **Run.**
 18. If you get a Reset Status dialog box indicating that the current launch will reset
the entire system, click **OK.**
 19. You should see text on the serial console, as shown in the next section.

You will observe the content captured by the PYTHON-1300-C image sensor on the DVI/HDMI monitor.

You have successfully executed the PYTHON-1300-C reference design on hardware !

Using the text based serial console

1. Once the design is running on hardware, you should see something similar to the following on your serial console:

```

-----
--                               FMC-HDMI-CAM + PYTHON-1300-C                               --
--                               Getting Started Design                                   --
-----

FMC-HDMI-CAM Initialization ...
Video Clock Synthesizer Configuration ...
HDMI Output Initialization ...
PYTHON Receiver Initialization ...
PYTHON Sensor Initialization ...
CFA Initialization
VDMA 0 Initialization
VDMA 1 Initialization
OSD Initialization (hdmi=0x00, cam=0xFF)

-----
--                               FMC-HDMI-CAM                               --
--                               Getting Started Design                                   --
-----

General Commands:
    help          Print the Top-Level menu Help Screen
    verbose on    Enable verbose
    verbose off   Disable verbose
Getting Started Commands
    start          start and select video source (hdmi|cam)

-----

FMCHC_PYTHON1300C>

```

Figure 14 – PYTHON-1300-C Camera Reference Design – Serial Console Output

If you have a PYTHON-1300-C camera module, you will observe the content captured by the PYTHON-1300-C image sensor on the DVI/HDMI monitor. If you do not have a camera module, you will see green content, the default value of pixels in the camera frame buffer.

Getting Help for commands

At any time, you can type the “help” command to display the list of commands supported by the demonstration.

```
FMCHC_PYTHON1300C>help
-----
--                               FMC-HDMI-CAM                               --
--                               Getting Started Design                         --
-----
General Commands:
    help          Print the Top-Level menu Help Screen
    verbose on    Enable verbose
    verbose off   Disable verbose
Getting Started Commands
    start         start and select video source (hdmi|cam)

-----

FMCHC_PYTHON1300C>
```

For more detailed information on a particular command, type the command followed by the “help” argument as follows:

```
FMCHC_PYTHON1300C>start help
Syntax :
    start cam    => Start CAM video source
    start hdmi   => Start HDMI video source

FMCHC_PYTHON1300C>
```


Starting the HDMI video pipeline

To enable the HDMI pipeline (HDMI input => HDMI output), press the “**start hdmi**” command. This will initialize the HDMI input interface, the corresponding Video DMA core, as well as configure the OSD core to display the HDMI video path.

If the resolution of the video source is 1080P, the content will fill the entire 1080P resolution output.

```
FMCHC_PYTHON1300C>start hdmi
HDMI Input Initialization
Waiting for ADV7611 to locked on incoming video ...
    ADV7611 Video Input LOCKED
    Input resolution = 1920 X 1080
VDMA 0 Initialization
VDMA 1 Initialization
OSD Initialization (hdmi=0xFF, cam=0x00)

FMCHC_PYTHON1300C>
```

If the resolution of the video source is less than 1080P, the video source will be displayed in the top-left portion of a 1080P resolution output.

```
FMCHC_PYTHON1300C>start hdmi
HDMI Input Initialization
Waiting for ADV7611 to locked on incoming video ...
    ADV7611 Video Input LOCKED
    Input resolution = 1280 X 720
VDMA 0 Initialization
VDMA 1 Initialization
OSD Initialization (hdmi=0xFF, cam=0x00)

FMCHC_PYTHON1300C>
```

Starting the Camera video pipeline

To enable the camera pipeline (HDMI input => HDMI output), press the “**start cam**” command. This will initialize the PYTHON camera, the corresponding Video DMA core, as well as configure the OSD core to display the PYTHON video path.

```
FMCHC_PYTHON1300C>start cam
PYTHON Receiver Initialization ...
PYTHON Sensor Initialization ...
CFA Initialization
VDMA 0 Initialization
VDMA 1 Initialization
OSD Initialization (hdmi=0x00, cam=0xFF)

FMCHC_PYTHON1300C>
```

Enabling Verbose

Additional verbose can be enabled with the “**verbose on**” command. This is useful for diagnostic purposes, when needed.

```
FMCHC_PYTHON1300C>verbose on
      verbose = on

FMCHC_PYTHON1300C>
```

The expected output, in verbose mode, for the camera input is shown below:

```
FMCHC_PYTHON1300C>verbose on
      verbose = on

FMCHC_PYTHON1300C>start cam
      start cam
Video Frame Buffer Initialization ...
PYTHON Receiver Initialization ...
PYTHON Sensor Initialization ...
VITA SYNCGEN - Setting Video Timing
      HSYNC Timing = hav=1280, hfp,=48, hsw=184 (hsp=1), hbp=074 (x4)
      VSYNC Timing = hav=1024, hfp,=01, hsw=03 (hsp=1), hbp=026 (x4)
VITA ISERDES - Setting Training Sequence to 0x000003A6
VITA ISERDES - Setting Manual Tap to 0x00000019
VITA DECODER - Configuring Sync Codes
VITA REMAPPER - Configuring for image lines in normal mode
```

```
VITA REMAPPER - Control = 0x00000001
VITA ISERDES - Asserting Reset
VITA DECODER - Asserting Reset
VITA CRC - Asserting Reset
VITA SPI Sequence 0 - Assert RESET_N pin
VITA ISERDES - Releasing Reset
VITA DECODER - Releasing Reset
VITA CRC - Releasing Reset
VITA SPI Sequence 0 - Releasing RESET_N pin
    VITA_SPI[0x0000] => 0x50D0
    ERROR: Unknown CHIP_ID !!!
VITA SPI Sequence 1 - Enable Clock Management - Part 1
    VITA_SPI[0x0002] <= 0x0001
    VITA_SPI[0x0020] <= 0x3004
    VITA_SPI[0x0014] <= 0x0000
    VITA_SPI[0x0011] <= 0x2113
    VITA_SPI[0x001A] <= 0x2280
    VITA_SPI[0x001B] <= 0x3D2D
    VITA_SPI[0x0008] <= 0x0000
    VITA_SPI[0x0010] <= 0x0003
VITA SPI Sequence 2 - Verify PLL Lock Indicator
    VITA_SPI[0x0018] => 0x0001
VITA SPI Sequence 3 - Enable Clock Management - Part 2
    VITA_SPI[0x0009] <= 0x0000
    VITA_SPI[0x0020] <= 0x3006
    VITA_SPI[0x0022] <= 0x0001
VITA SPI Sequence 4 - Required Register Upload
    VITA_SPI[0x00C5] <= 0x0205
    VITA_SPI[0x00E0] <= 0x3E5E
    VITA_SPI[0x00CF] <= 0x0000
    VITA_SPI[0x0081] <= 0x8001
    VITA_SPI[0x0080] <= 0x4714
    VITA_SPI[0x00CC] <= 0x01E3
    VITA_SPI[0x0029] <= 0x085A
    VITA_SPI[0x002A] <= 0x0011
    VITA_SPI[0x0041] <= 0x288B
    VITA_SPI[0x00D3] <= 0x0E49
    VITA_SPI[0x002B] <= 0x0008
    VITA_SPI[0x0046] <= 0x1111
    VITA_SPI[0x0043] <= 0x0554
    VITA_SPI[0x0042] <= 0x53C6
    VITA_SPI[0x0044] <= 0x0085
    VITA_SPI[0x00D7] <= 0x0107
    VITA_SPI[0x00C2] <= 0x0221
    VITA_SPI[0x00C7] <= 0x001B
    VITA_SPI[0x00C9] <= 0x2710
    VITA_SPI[0x00C8] <= 0x411A
    VITA_SPI[0x00C0] <= 0x0800
VITA SPI Sequence 5 - Soft Power-Up
    VITA_SPI[0x0020] <= 0x3007
    VITA_SPI[0x000A] <= 0x0000
    VITA_SPI[0x0040] <= 0x0001
    VITA_SPI[0x0048] <= 0x2227
    VITA_SPI[0x002A] <= 0x0013
    VITA_SPI[0x0028] <= 0x0003
```

```

VITA_SPI[0x0030] <= 0x0001
VITA_SPI[0x0070] <= 0x0007
VITA_SPI[0x0080] <= 0x4714
VITA ISERDES - Status = 0x61610100
VITA ISERDES - Status = 0x61610100
VITA ISERDES - Waiting for CLK_RDY to assert
VITA ISERDES - Status = 0x61610100
VITA ISERDES - Align Start
VITA ISERDES - Waiting for ALIGN_BUSY to assert
VITA ISERDES - Status = 0x61610304
VITA ISERDES - Waiting for ALIGN_BUSY to de-assert
VITA ISERDES - Status = 0x61610100
VITA ISERDES - Status = 0x61610100
VITA SPI Sequence 6 - Enable Sequencer
VITA_SPI[0x00C0] => 0x0800
                        0x0001
VITA_SPI[0x00C0] <= 0x0801
VITA ISERDES - Enabling FIFO enable
VITA DECODER - Enabling Sync Channel Decoder
VITA DECODER - Control = 0x00000002
VITA CRC - Status = 0x0000000F
VITA CRC - Status = 0x0000000F
VITA CRC - Status = 0x0000000F
VITA CRC - Status = 0x0000000F
VITA CRC - Status = 0x0000000F
VITA CRC - Status = 0x0000000F
VITA CRC - Status = 0x00000000
VITA SPI Sequence CDS
VITA_SPI[0x00C0] <= 0x0800
VITA_SPI[0x00CC] <= 0x01E3
VITA_SPI[0x0041] <= 0x288B
VITA_SPI[0x0029] <= 0x085F
VITA_SPI[0x002A] <= 0x4113
VITA_SPI[0x002B] <= 0x0008
VITA_SPI[0x0048] <= 0x0017
VITA_SPI[0x0180] <= 0xC800
VITA_SPI[0x0181] <= 0xFB1F
VITA_SPI[0x0182] <= 0xFB1F
VITA_SPI[0x0183] <= 0xFB12
VITA_SPI[0x0184] <= 0xF903
VITA_SPI[0x0185] <= 0xF802
VITA_SPI[0x0186] <= 0xF30F
VITA_SPI[0x0187] <= 0xF30F
VITA_SPI[0x0188] <= 0xF30F
VITA_SPI[0x0189] <= 0xF30A
VITA_SPI[0x018A] <= 0xF101
VITA_SPI[0x018B] <= 0xF00A
VITA_SPI[0x018C] <= 0xF24B
VITA_SPI[0x018D] <= 0xF226
VITA_SPI[0x018E] <= 0xF001
VITA_SPI[0x018F] <= 0xF402
VITA_SPI[0x0190] <= 0xF001
VITA_SPI[0x0191] <= 0xF402
VITA_SPI[0x0192] <= 0xF001
VITA_SPI[0x0193] <= 0xF401

```

```

VITA_SPI[0x0194] <= 0xF007
VITA_SPI[0x0195] <= 0xF20F
VITA_SPI[0x0196] <= 0xF20F
VITA_SPI[0x0197] <= 0xF202
VITA_SPI[0x0198] <= 0xF006
VITA_SPI[0x0199] <= 0xEC02
VITA_SPI[0x019A] <= 0xE801
VITA_SPI[0x019B] <= 0xEC02
VITA_SPI[0x019C] <= 0xE801
VITA_SPI[0x019D] <= 0xEC02
VITA_SPI[0x019E] <= 0xC801
VITA_SPI[0x019F] <= 0xC800
VITA_SPI[0x00D8] <= 0x7F00
VITA_SPI[0x01A0] <= 0xC800
VITA_SPI[0x01A1] <= 0xCC02
VITA_SPI[0x01A2] <= 0xC801
VITA_SPI[0x01A3] <= 0xCC02
VITA_SPI[0x01A4] <= 0xC801
VITA_SPI[0x01A5] <= 0xCC02
VITA_SPI[0x01A6] <= 0xC806
VITA_SPI[0x01A7] <= 0xC800
VITA_SPI[0x00DB] <= 0x0020
VITA_SPI[0x01A8] <= 0x0030
VITA_SPI[0x01A9] <= 0x2076
VITA_SPI[0x01AA] <= 0x2071
VITA_SPI[0x01AB] <= 0x0071
VITA_SPI[0x01AC] <= 0x107F
VITA_SPI[0x01AD] <= 0x1072
VITA_SPI[0x01AE] <= 0x1074
VITA_SPI[0x01AF] <= 0x0076
VITA_SPI[0x01B0] <= 0x0031
VITA_SPI[0x01B1] <= 0x21BB
VITA_SPI[0x01B2] <= 0x20B1
VITA_SPI[0x01B3] <= 0x00B1
VITA_SPI[0x01B4] <= 0x10BF
VITA_SPI[0x01B5] <= 0x10B2
VITA_SPI[0x01B6] <= 0x10B4
VITA_SPI[0x01B7] <= 0x00B1
VITA_SPI[0x01B8] <= 0x0030
VITA_SPI[0x01B9] <= 0x0030
VITA_SPI[0x01BA] <= 0x217B
VITA_SPI[0x01BB] <= 0x2071
VITA_SPI[0x01BC] <= 0x2071
VITA_SPI[0x01BD] <= 0x0071
VITA_SPI[0x01BE] <= 0x107F
VITA_SPI[0x01BF] <= 0x1072
VITA_SPI[0x01C0] <= 0x1074
VITA_SPI[0x01C1] <= 0x0076
VITA_SPI[0x01C2] <= 0x0031
VITA_SPI[0x01C3] <= 0x20B6
VITA_SPI[0x01C4] <= 0x00B1
VITA_SPI[0x01C5] <= 0x10BF
VITA_SPI[0x01C6] <= 0x10B2
VITA_SPI[0x01C7] <= 0x10B4
VITA_SPI[0x01C8] <= 0x00B1

```

```
VITA_SPI[0x01C9] <= 0x0030
VITA_SPI[0x00DC] <= 0x3928
VITA_SPI[0x01CA] <= 0x0030
VITA_SPI[0x01CB] <= 0x20F3
VITA_SPI[0x01CC] <= 0x2071
VITA_SPI[0x01CD] <= 0x0071
VITA_SPI[0x01CE] <= 0x0179
VITA_SPI[0x01CF] <= 0x0078
VITA_SPI[0x01D0] <= 0x1074
VITA_SPI[0x01D1] <= 0x0076
VITA_SPI[0x01D2] <= 0x0031
VITA_SPI[0x01D3] <= 0x21BD
VITA_SPI[0x01D4] <= 0x20B1
VITA_SPI[0x01D5] <= 0x00B1
VITA_SPI[0x01D6] <= 0x10BF
VITA_SPI[0x01D7] <= 0x10B2
VITA_SPI[0x01D8] <= 0x10B4
VITA_SPI[0x01D9] <= 0x00B1
VITA_SPI[0x01DA] <= 0x0030
VITA_SPI[0x00DD] <= 0x624A
VITA_SPI[0x00DE] <= 0x624A
VITA_SPI[0x00C0] => 0x0801
                        0x0001
VITA_SPI[0x00C0] <= 0x0801
CFA Initialization
VDMA 0 Initialization
VDMA 1 Initialization
OSD Initialization (hdmi=0x00, cam=0xFF)

FMCHC_PYTHON1300C>
```

Revision History

Date	Version	Revision
7 Sep 2015	2014.4.01	First Version