Unit / Module Description:	PCIe/104 OneBank + ARM + FPGA + FMC carrier
Unit / Module Number:	EMC²-DP V2
Document Issue Number:	1.2_2715
Original Issue Date:	2nd June 2016
Original Author:	Timoteo Garcia

EMC²-DP V2 STARTER'S GUIDE

PCIe/104 OneBank™ Carrier for 40mm x 50mm SoM + VITA57.1 FMC™ Modules

Sundance Multiprocessor Technology Ltd,
Chiltern House, Waterside, Chesham, Bucks, HP5 1PS, UK.

Revision History

Issue	Changes Made	Date	Initials
1.0	First draft.	2/6/16	TG
1.1	Added the board files for EMC². Updated information about boot images. Added HDMI test project tutorial. Added information about the SEIC connector.	23/6/16	TG
1.2	Added warning at 1.1.2	04/8/16	TG
1.3	Fixed errors + update for z7030	05/10/16	TG
1.4	Added HDMI SDSoC Platform tutorial	07/10/16	TG
2.0	Added chapter 4.Demos Changed the structure of 3.6 and 3.7 Added how to create an SDSoC platform in 2016.3 and above	15/10/17	EW

Table of Contents

Table of Figures

1 Introduction

This document is a guide for those who own an EMC²-DP V2, and gives an overview of both hardware and software capabilities in order to set the board up to be used in any system.

This guide provides general information about how to configure the IO voltages, upstream port of the PCIe, flash and SD boot configuration, and quick tutorials of how to make a new project either in Vivado or SDK environments

Any information related to the board that is not found in this document, can be found either in *EMC²-DP Design Specification (QCF51),* or *EMC² Board IO*. In any other case, contact us for more help.

2 Hardware

2.1 How can I connect the EMC² to the PSU?

The EMC² works well with any power supply which provides the 12, 3.3 and 5V necessary for the board to work with all its capabilities. The one we recommend is the power supply FSP300-60GHS.

Figure 1 - Power supply

Connect the power supply to the board, using the following cable:

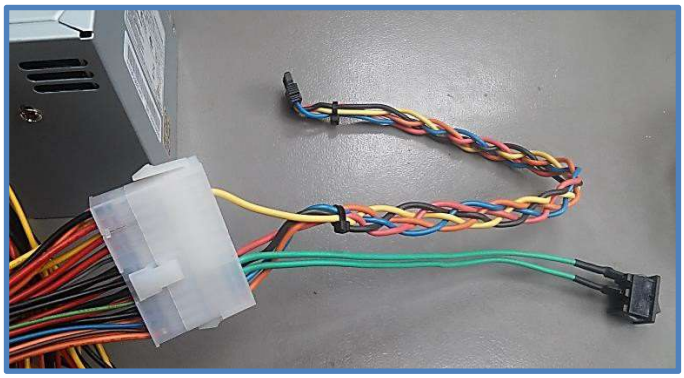


Figure 2 - Cable for the PSU

And connect the cable to the board.

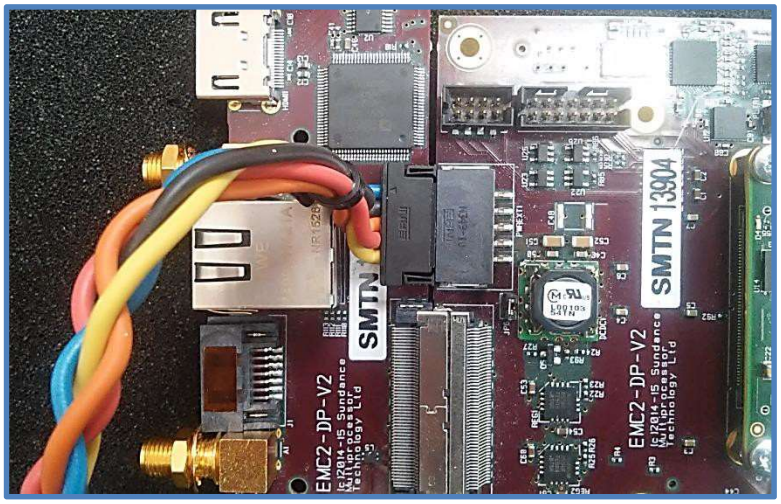


Figure 3 - Connecting the EMC² to the PSU

Turn on the PSU connecting it through the power connector and switching it on.

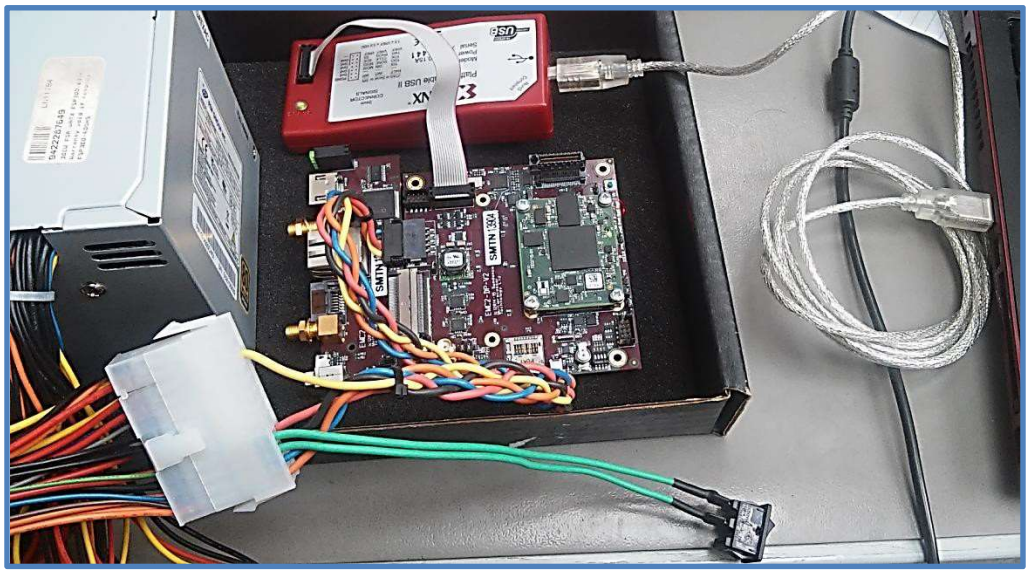


Figure 4 - EMC² fully working

2.2 How can I configure the IO Voltages?

WARNING: Never configure the IO Voltages with the board powered up!

This board needs an external supply of 3.3V for most of the components on board, as well as 5V and 12V for the PCIe and FMC ports. The IO voltages for the FPGA banks can be selected through the jumpers shown in the picture.

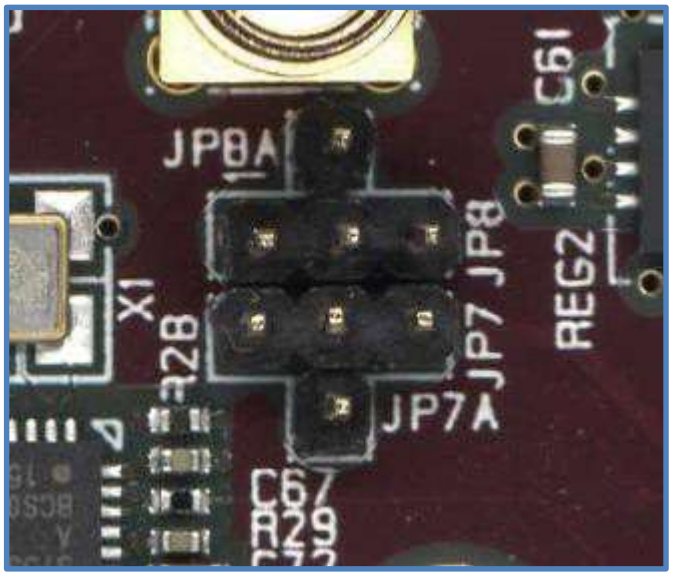


Figure 5 - JP7 and JP8

JP7 and JP8 select the different voltages available as follows:

3.3V: Position 1-2

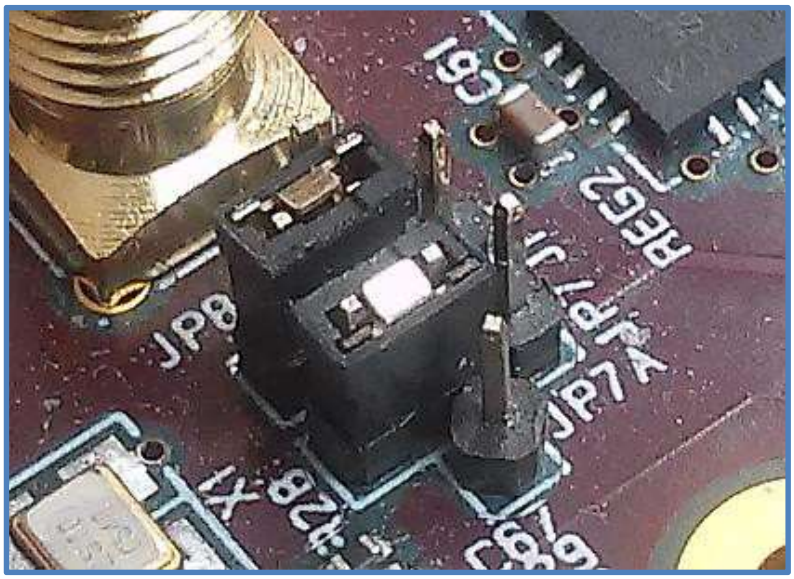


Figure 6 - 3.3V selection

2.5V: Position 2-JP7A and 2-JP8A

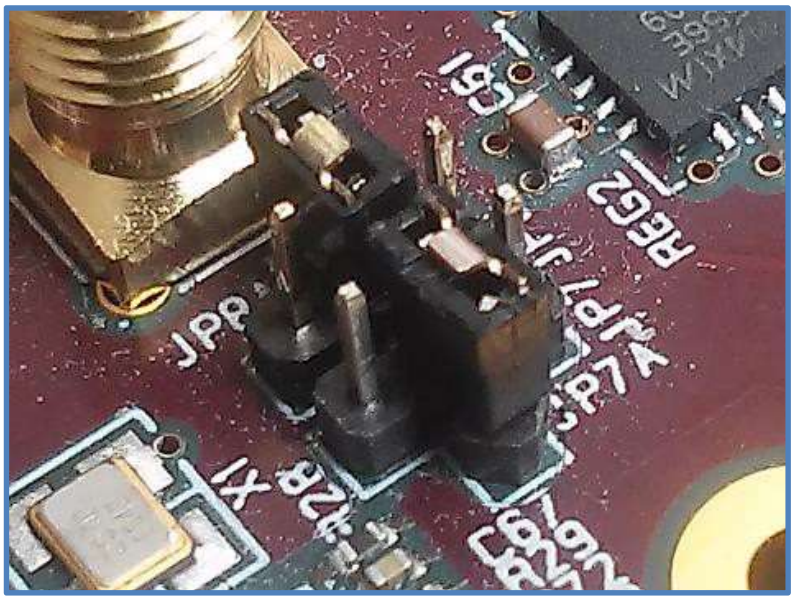


Figure 7 - 2.5V selection

1.8V: Position 2-3

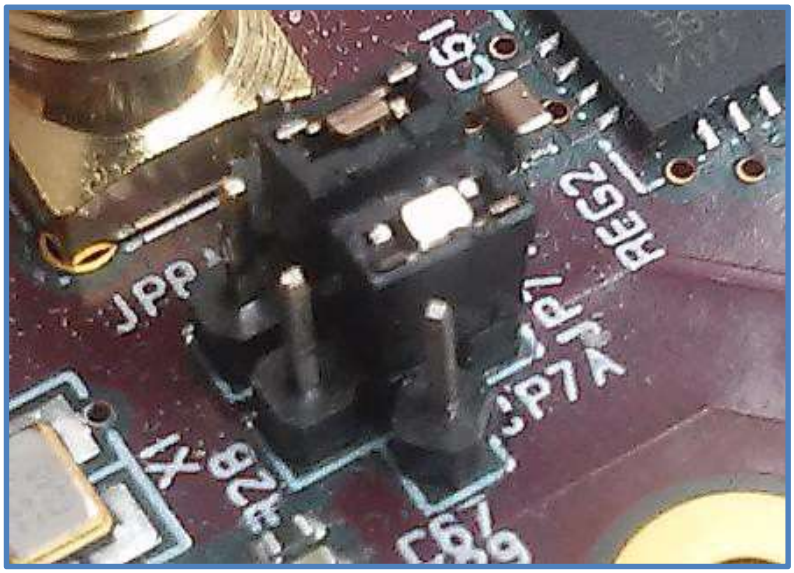


Figure 8 - 1.8V selection

JP7 selects the voltage which feeds mainly the HDMI and SEIC related signals (bank 34), while JP8 selects the voltage which feeds mainly the FMC related signals (banks 35, 13)

2.3 How can I configure the board to use PCIe?

The EMC² can work in host or add/on mode, depending on the application and the needs of the user.

In case of using the EMC² as host, there is one thing the user must do:

- JP12 must be set up, as it's related to the "clock select" pin at the PCIe, and it will provide the 100MHz reference clock.

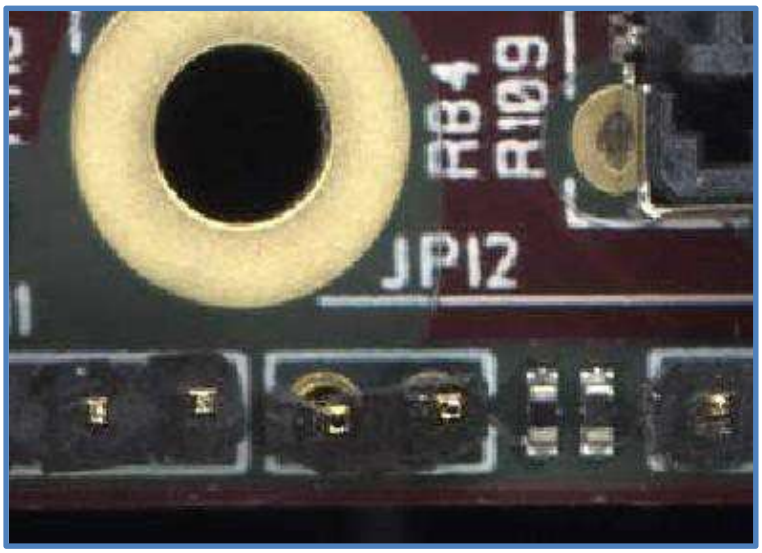


Figure 9 - Host Jumper

If the board is used on a stack, it can be used in add/on mode, where JP12 should be unconnected.

To select the upstream port, SW2 should be configured as follows:

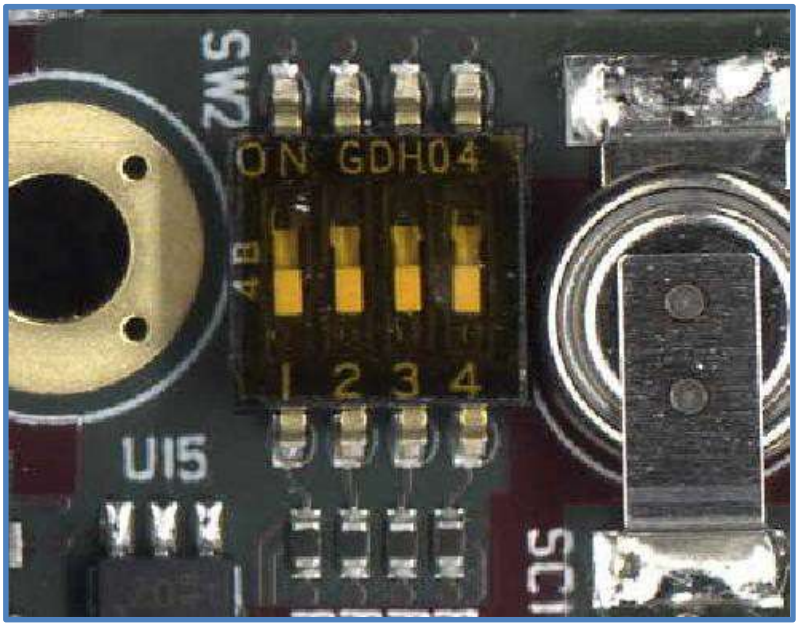


Figure 10 - Upstream port selection

(PEX) Port 0 : (Pcie) Lane 0 : 0000(LLLL) : All On
(PEX) Port 4 : (Pcie) Lane 1 : 0100(LHLL) : On-Off-On-On
(PEX) Port 1 : (Pcie) Lane 4 : 0001(LLLH) : Off-On-On-On
(PEX) Port 5 : (Pcie) Lane 5 : 0101(LHLH) : On-Off-On-Off
(PEX) Port 7 : (Pcie) Lane 6 : 0110(LHHL) : Off-Off-Off-On
(PEX) Port 9 : (Pcie) Lane 7 : 0111(LHHH) : On-Off-Off-Off

Where the PEX Port and Pcie Lane are the same thing, but called differently.
L corresponds to “On” and H to “Off” from SW2.

Here there is an example of how JP12 and SW2 should be to set the board as host.

Figure 11 - JP12 sets the host mode, SW2 selects port 0 as upstream port

2.4 How can I boot from flash or SD card?

The jumper (JP11) is the boot mode for the "flash devices" to be set from the EMC².

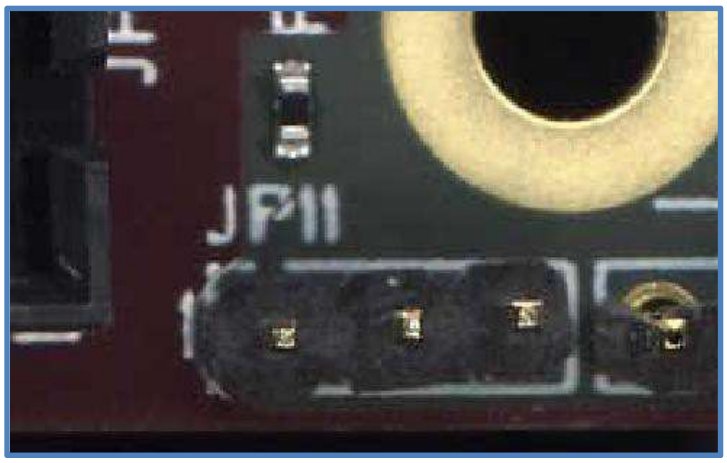


Figure 12 - Boot mode selection

The positions depending on where the user wants to boot from are:

-QSPI flash mode:
Position 1-2, resides on the Zynq MIO 1..6.
-SD card mode:
Position 2-3, (closet to JP12) resides on the Zynq MIO 40..45.

At Figure 11, Next to JP12, JP11 is set as SD booting mode.

2.5 Information about the SEIC Connector

The EMC²-DP has most of its capabilities available at the extension board, accessible through the SEIC connector.

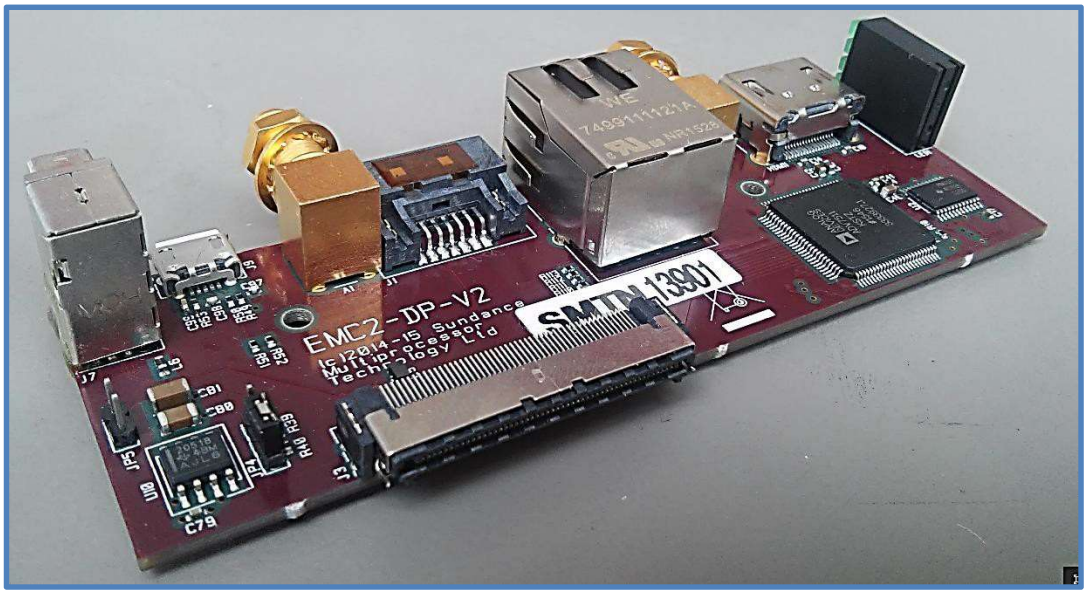


Figure 13 - SEIC extension board

This connector is labelled as J3 and J4 on the extension board and main board respectively.

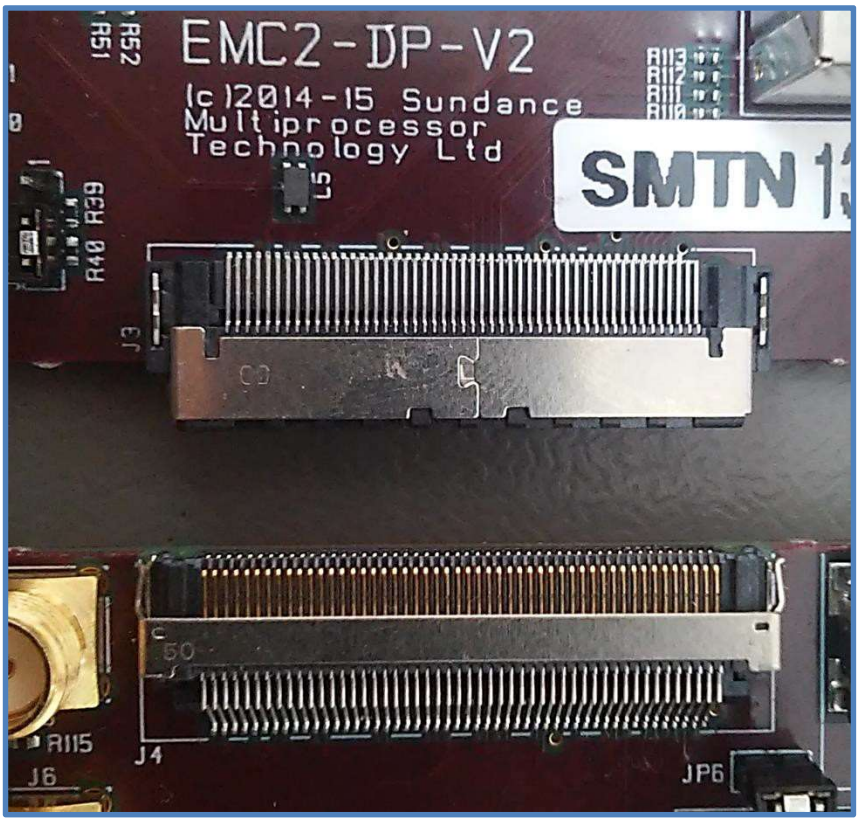


Figure 14 - SEIC connector

The pinout of the SEIC is represented in this schematic, where the top pins of both J3 and J4 are connected, as well as the bottom pins.

Figure 15 - SEIC Connector Pinout

3 Software

3.1 How can I use Vivado with the EMC²?

The EMC² is well supported from Vivado 15.2 and it's recommended to use always the newer versions.

For Vivado 15.4 and newer versions, board files for the EMC²-Z7015 and EMC²-Z7030 are available.

Board files from Trenz Electronic can be used depending on the module installed on the carrier.

This will make life easier for the user when assign constraints related to MGT signals, or MIO pins in Zynq architecture.

All the pin locations for the SEIC devices and FMC are described in the *EMC² Board IO* document.

To include the board files in your system, download the corresponding files, and add them at the installation path of Vivado, normally:

C:\Xilinx\Vivado\2015.X\data\boards\board_files

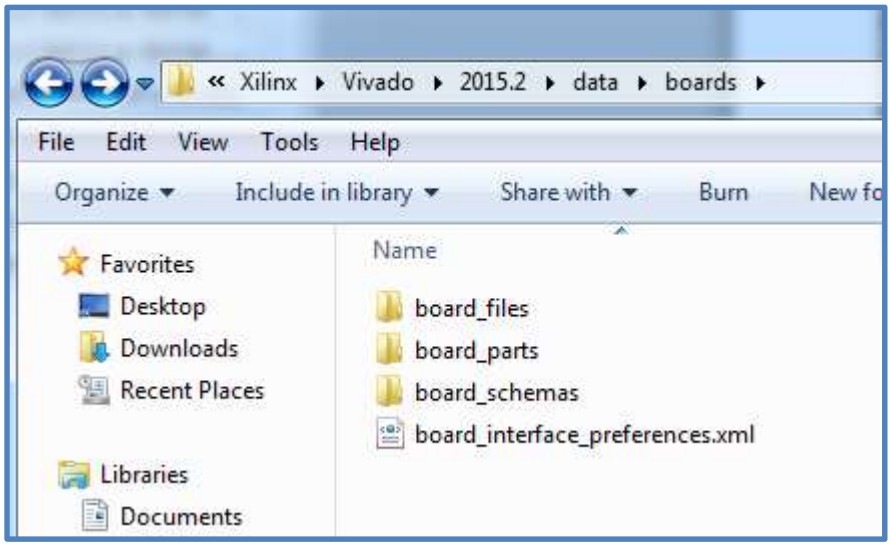


Figure 16 - Board files path

To use the board files in a project, do as follows.

Open Vivado and create a new project:

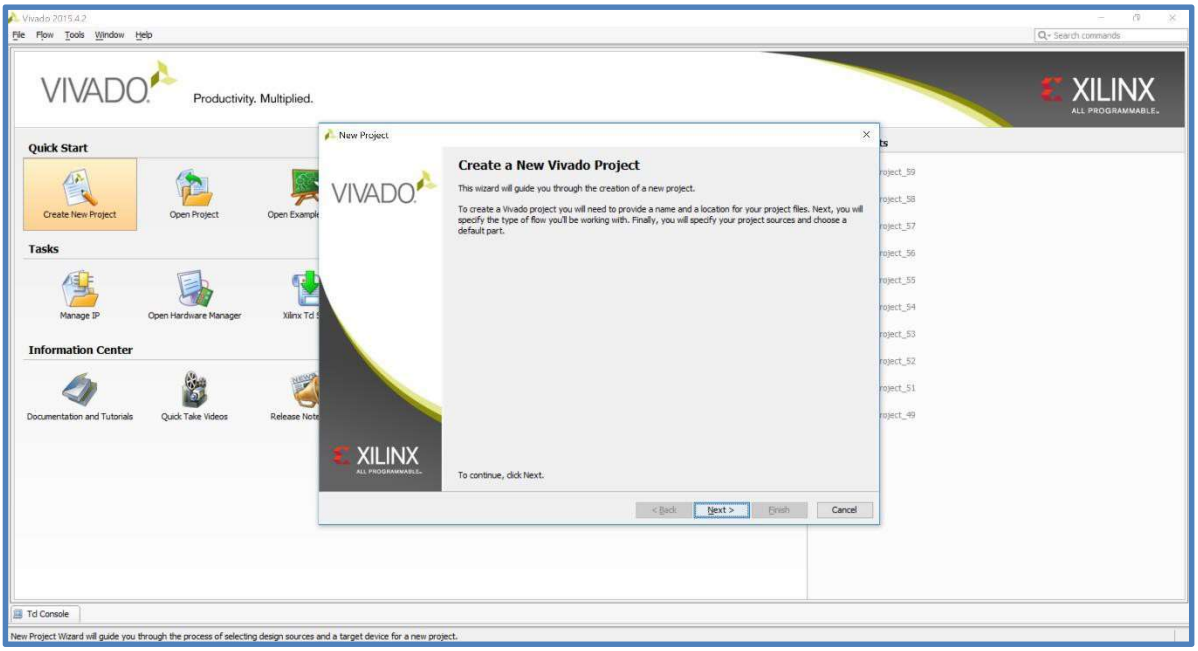


Figure 17 - Create a new project

Click "Next" and select the path of the project.

Choose RTL project and mark the square "Do not specify sources at this time" in case the project won't import any source and it's a blank project.

Then, when Vivado asks for a device part, select "Boards", and choose either the EMC²-Z7015, EMC²-Z7030 or compatible modules from Trenz:

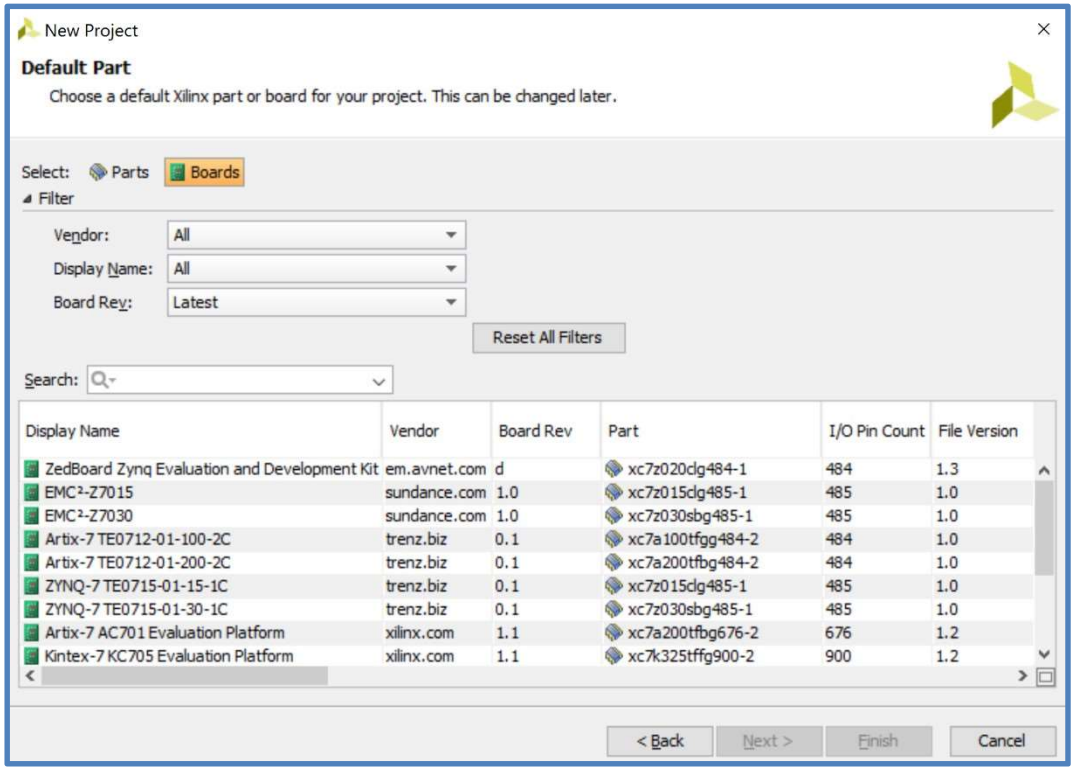


Figure 18 - Board file selection

Once the project is opened, the user can notice the information about the board and the part used.

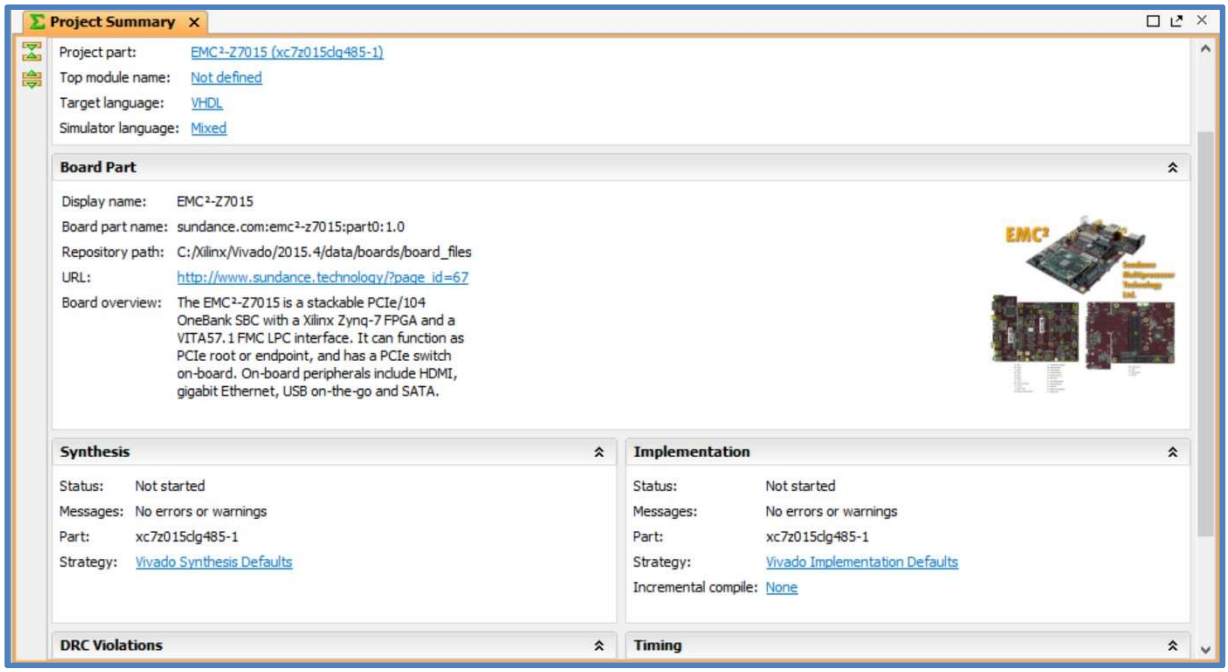


Figure 19 - Board information in Vivado

As soon as the user creates a new block design in IP Integrator, will have available some interfaces already set up and ready to use:

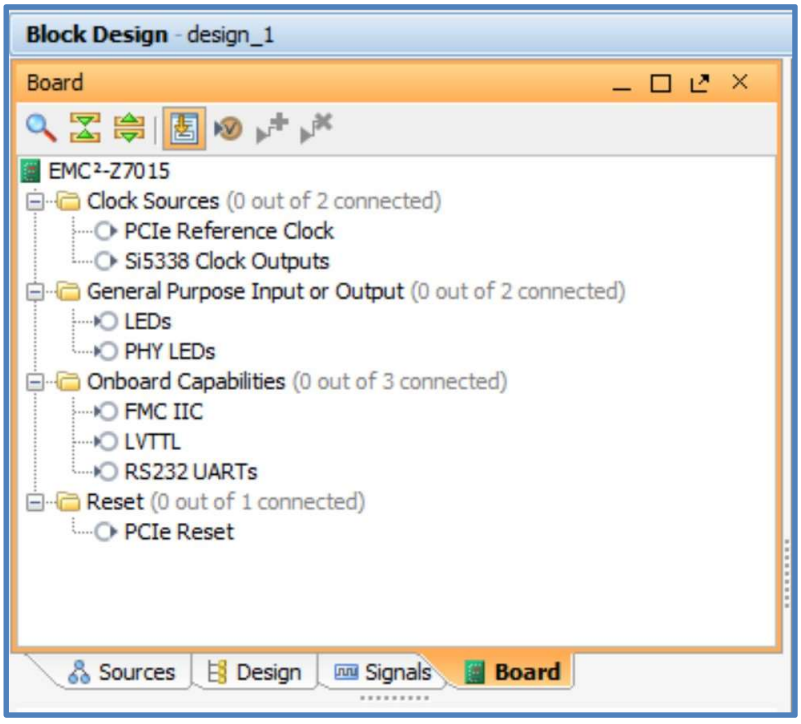


Figure 20 - Board interfaces in IP Integrator

The interfaces available are those ones related to the FPGA pins. To use the rest of the capabilities, they can be used through the MIO pins in the PS of the Zynq.

As an example, to select one of the outputs of the clock synthesizer (Si5338), just double click on Si5338 Clock Outputs, at Clock Sources:

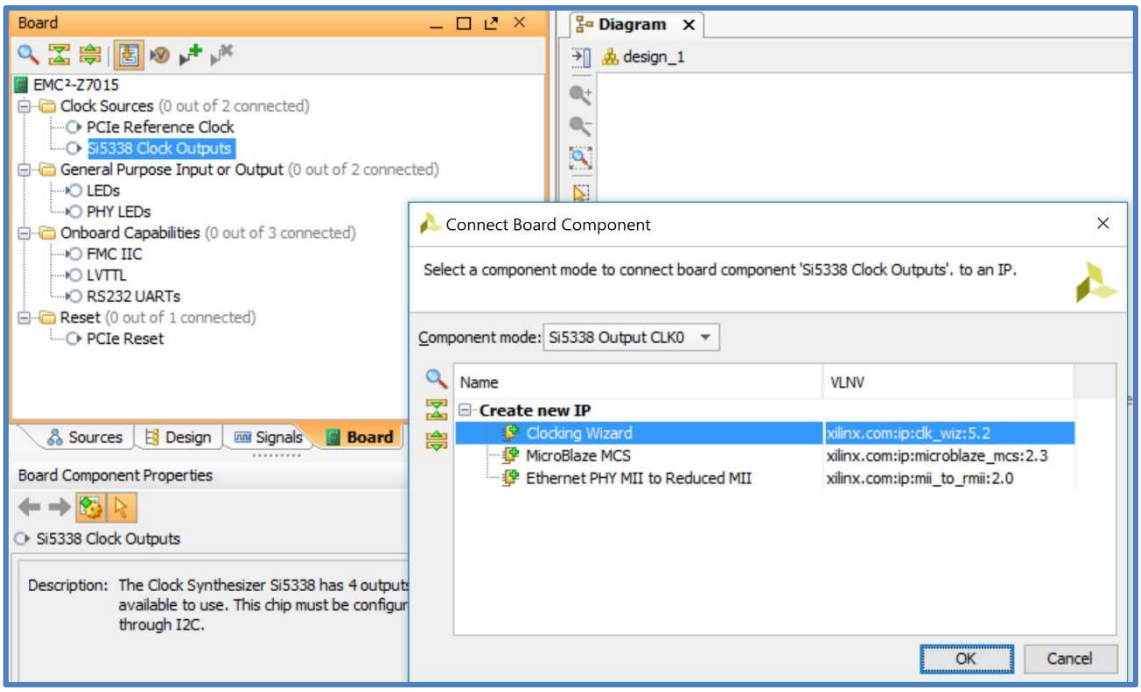


Figure 21 - How to use interfaces in IPI

Select Output CLK0 as component mode, and Clocking Wizard as IP.
Automatically, the input will be assigned to the IP selected.

Remember that these capabilities are made so that the user can access easier to them. The outputs of this clock synthesizer must be preconfigured through IIC in order to use them.

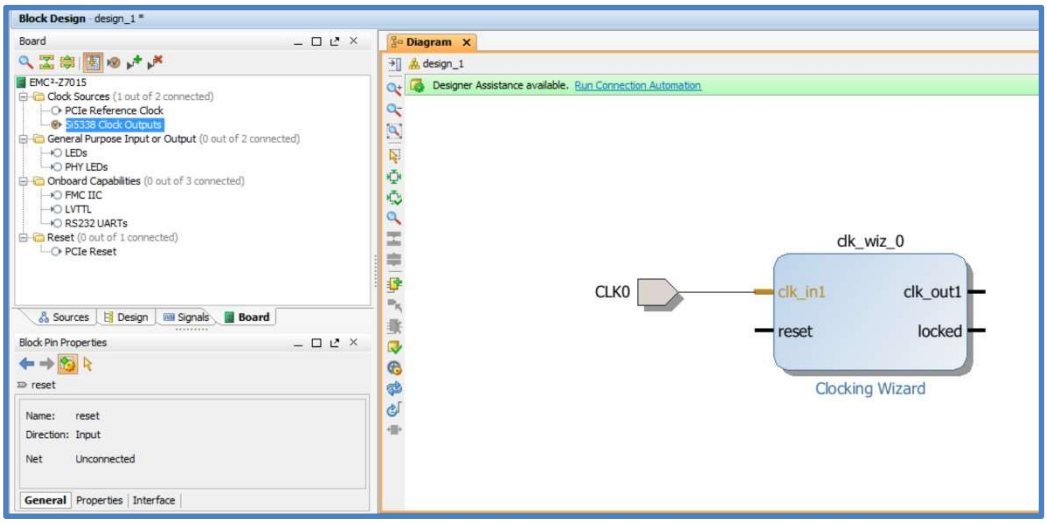


Figure 22 - Block generated through the interface

For the FMC Interface, HDMI and MGT signals on board, there is a constraints file available to use. The HDMI can be tested following the tutorial in this document.

3.2 How can I create a Zynq FSBL for the EMC²?

To create a FSBL is necessary to determine the hardware of the system through Vivado, and export that hardware to SDK environment, where it's possible to create a FSBL based on this hardware defined previously.

As an example, a simple project will be created to show the procedure:

Following the steps mentioned at 1.2.1, a project with a TE0715-01-30 part, called "FSBL_example" has been created.

Click "Create block design" in the GUI of Vivado, and choose a name for it:

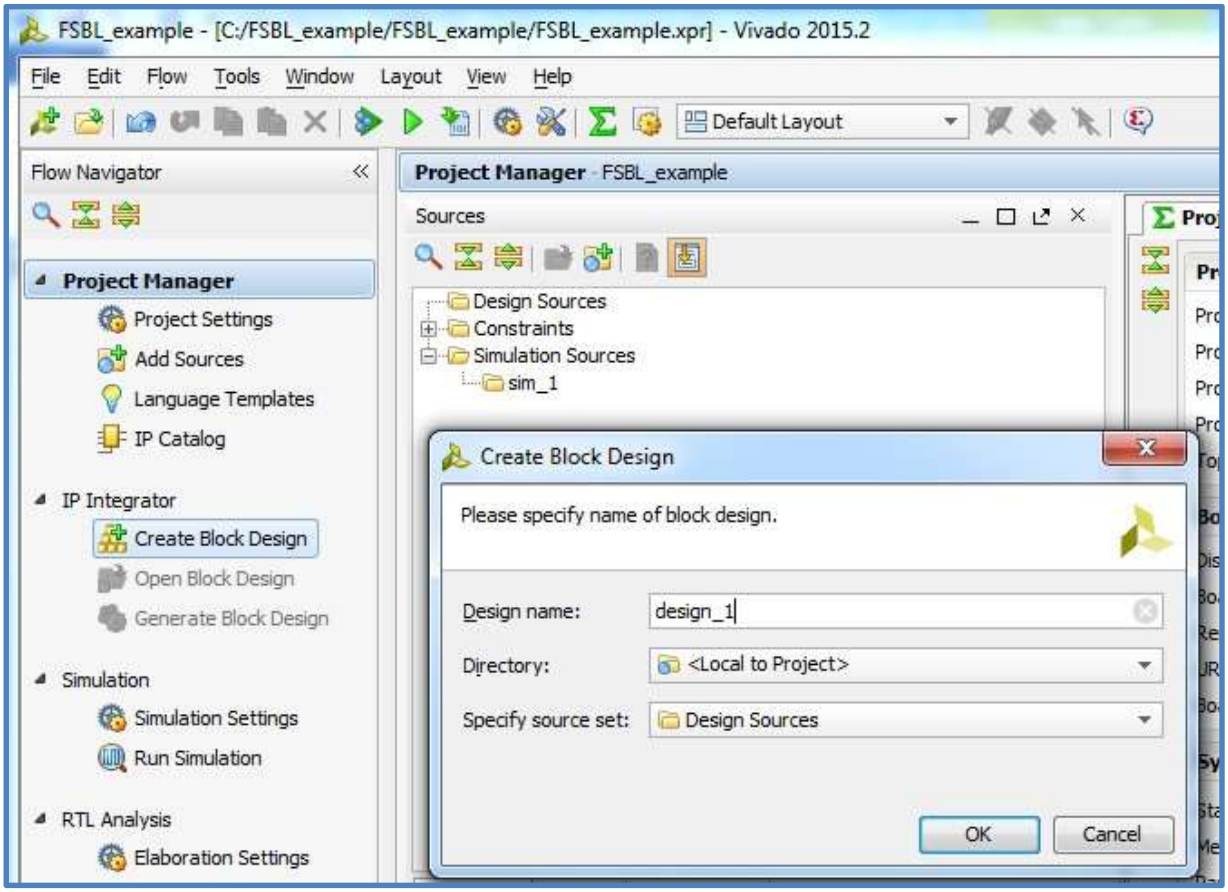


Figure 23 - Create block design

Add a Zynq Processing System IP block, and connect FCLK_CLK0 to M_AXI_GP0_ACLK.

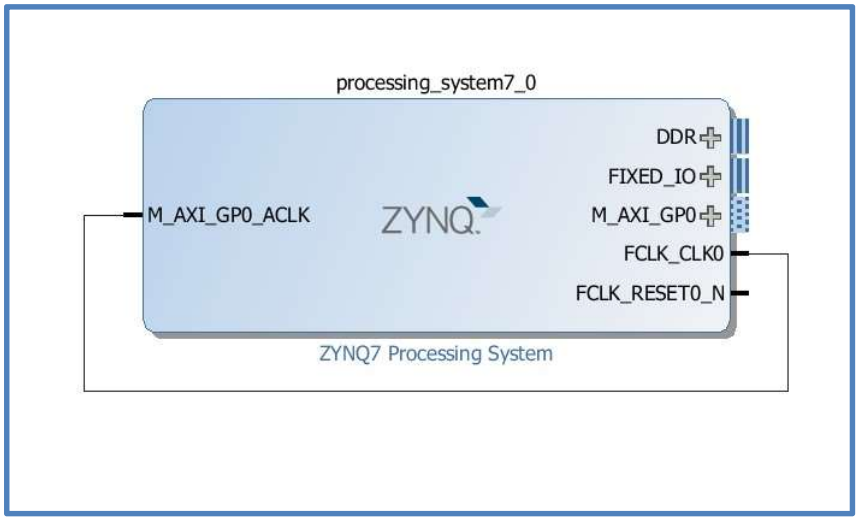


Figure 24 - Zynq Processing System

Then, click Run Block Automation, and press "OK".

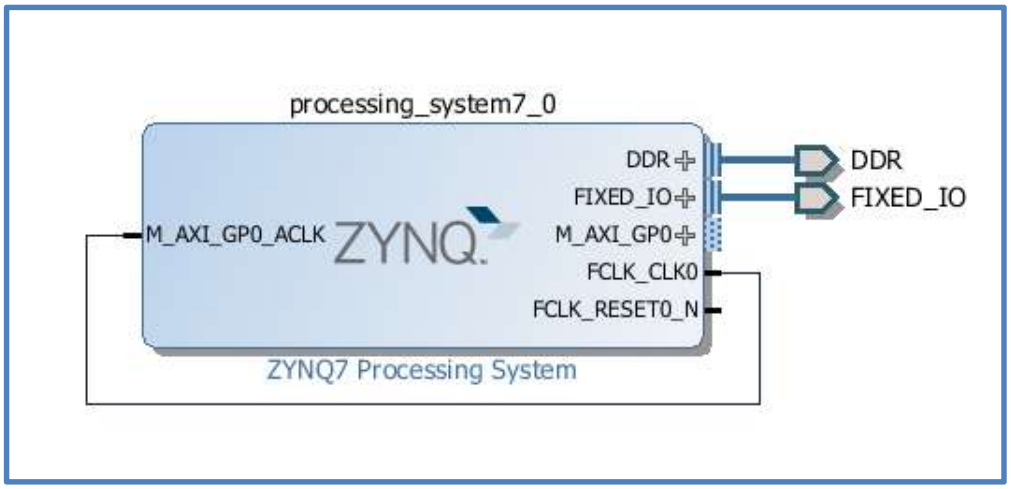


Figure 25 - Run Block Automation

The MIO pins are automatically assigned. If the user needs to add/configure more capabilities, as UART, GPIO pins, etc., make double click in the Zynq Processing System, and select them

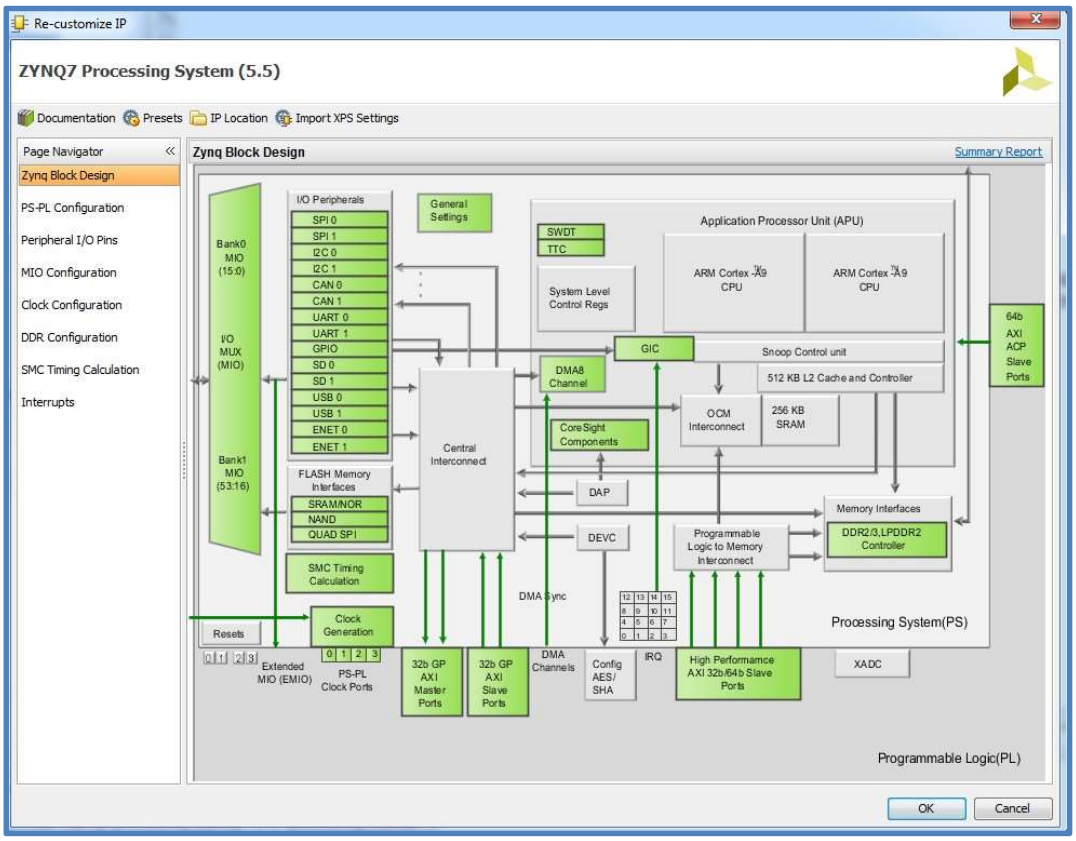


Figure 26 - Zynq architecture

It is highly recommended to read the Zynq documentation from Xilinx to understand the architecture and be able to configure the Zynq properly.

For this example, Uart 0 will be added.

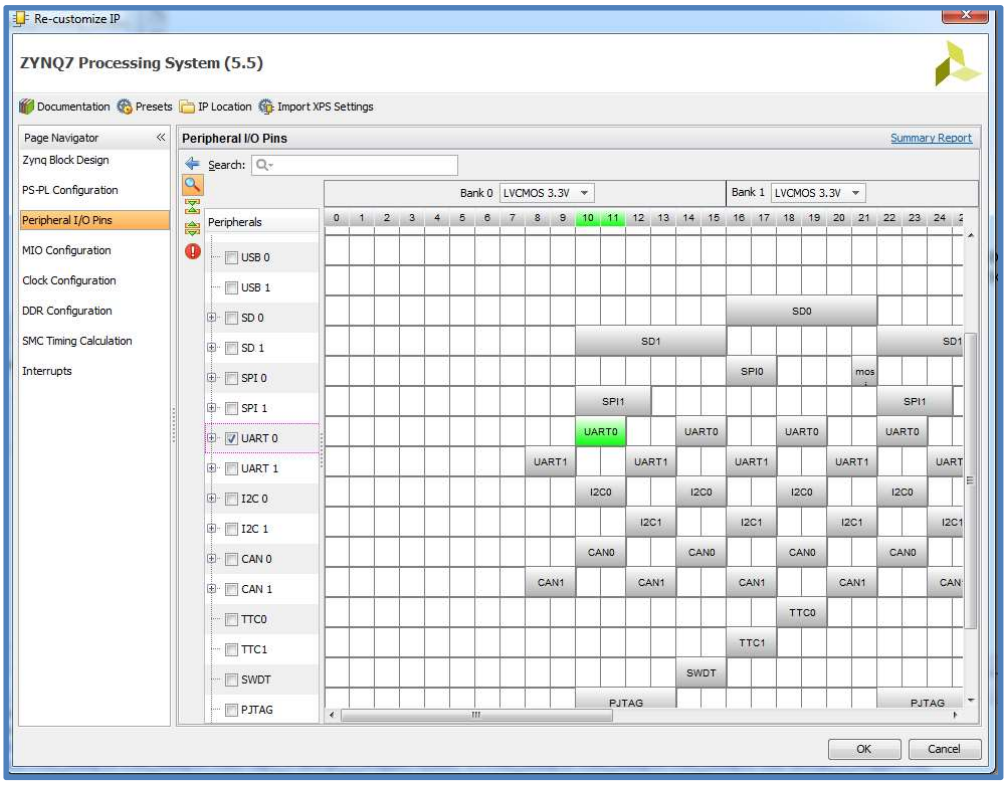


Figure 27 - Adding Uart

Click “Validate design” and then “OK”

Figure 28 - Validate design

Create a Wrapper of the design, as follows:

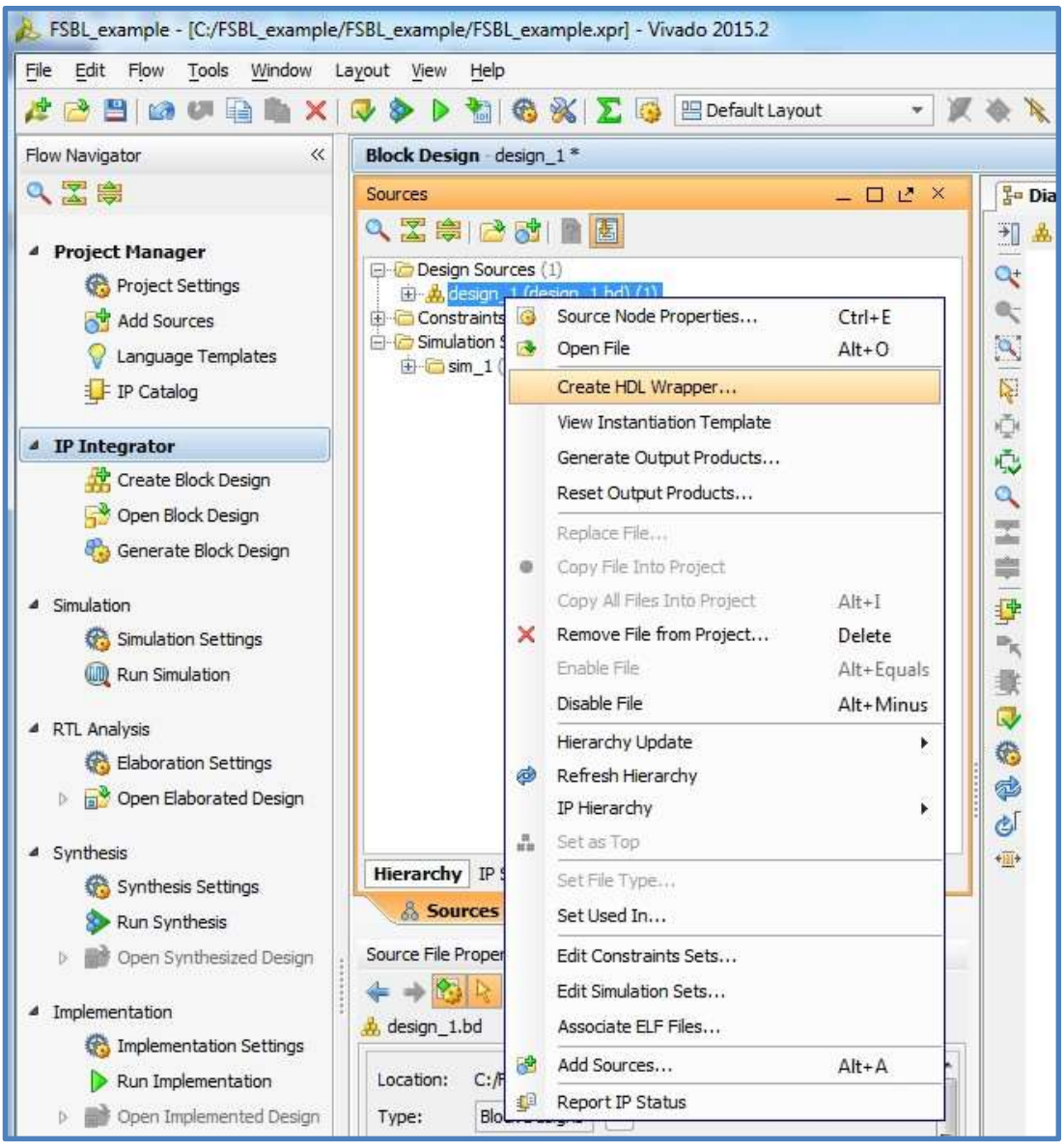


Figure 29 - HDL Wrapper

Generate the Bitstream, tool easy to find in the Flow Navigator.

Now is time to export the hardware. Select “File -> Export -> Export Hardware”

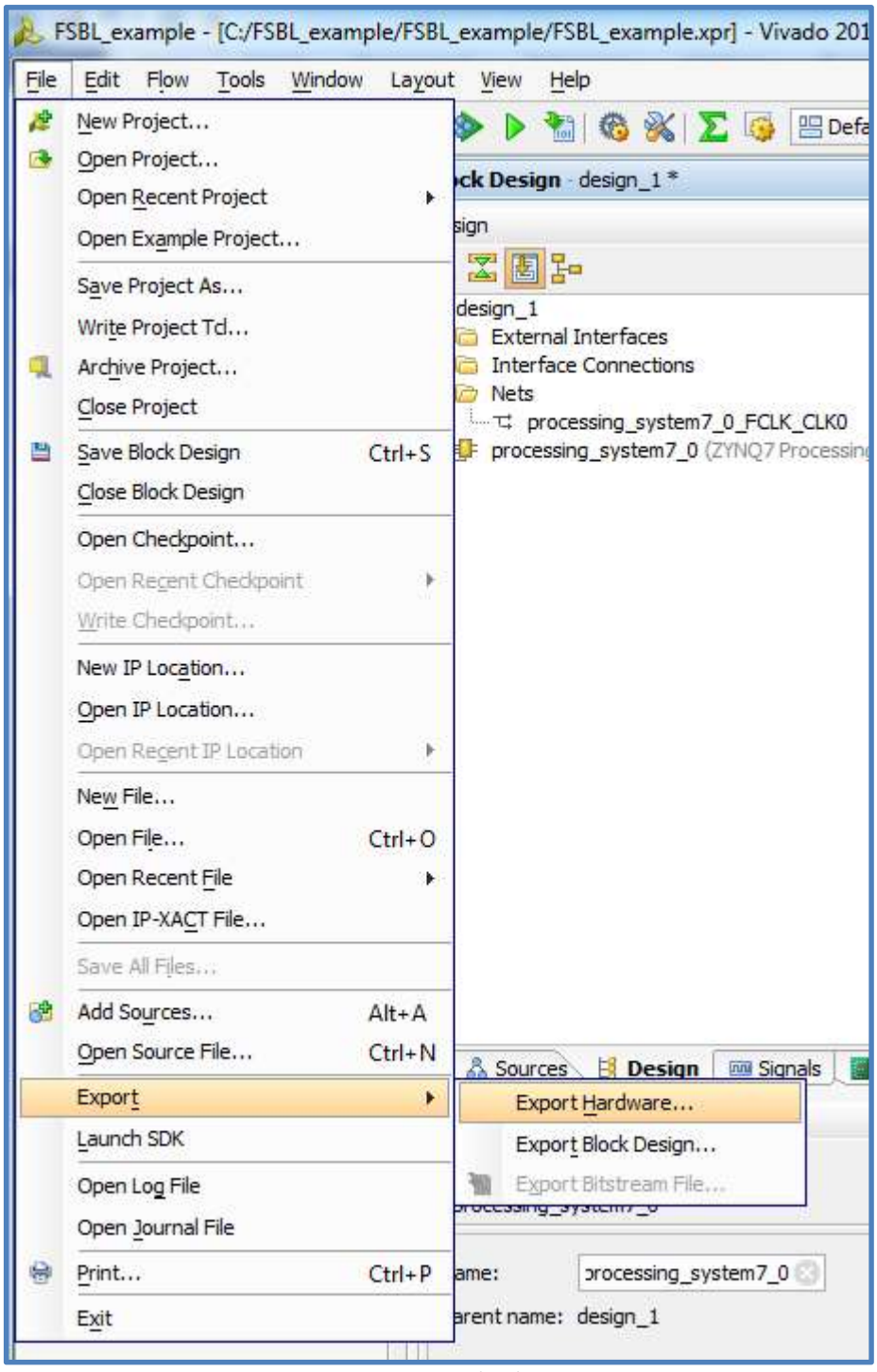


Figure 30 - Export hardware

Mark “Include Bitstream”, and export.

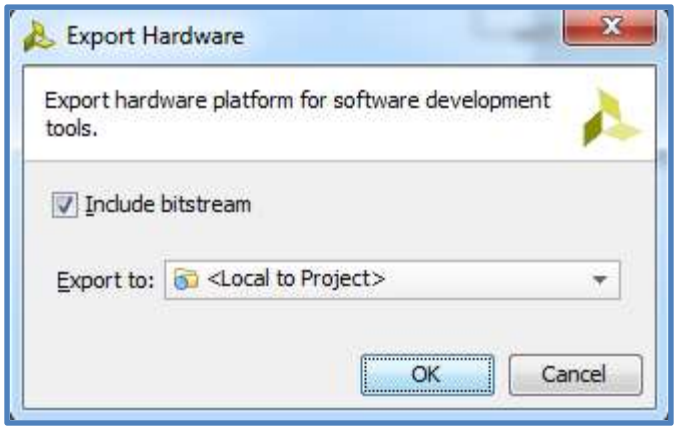


Figure 31 - Export hardware including the bitstream

Launch the SDK ("File -> Launch SDK")

The SDK opens automatically our project, where the user can see the ps7_init project, which initializes the PS in the Zynq.

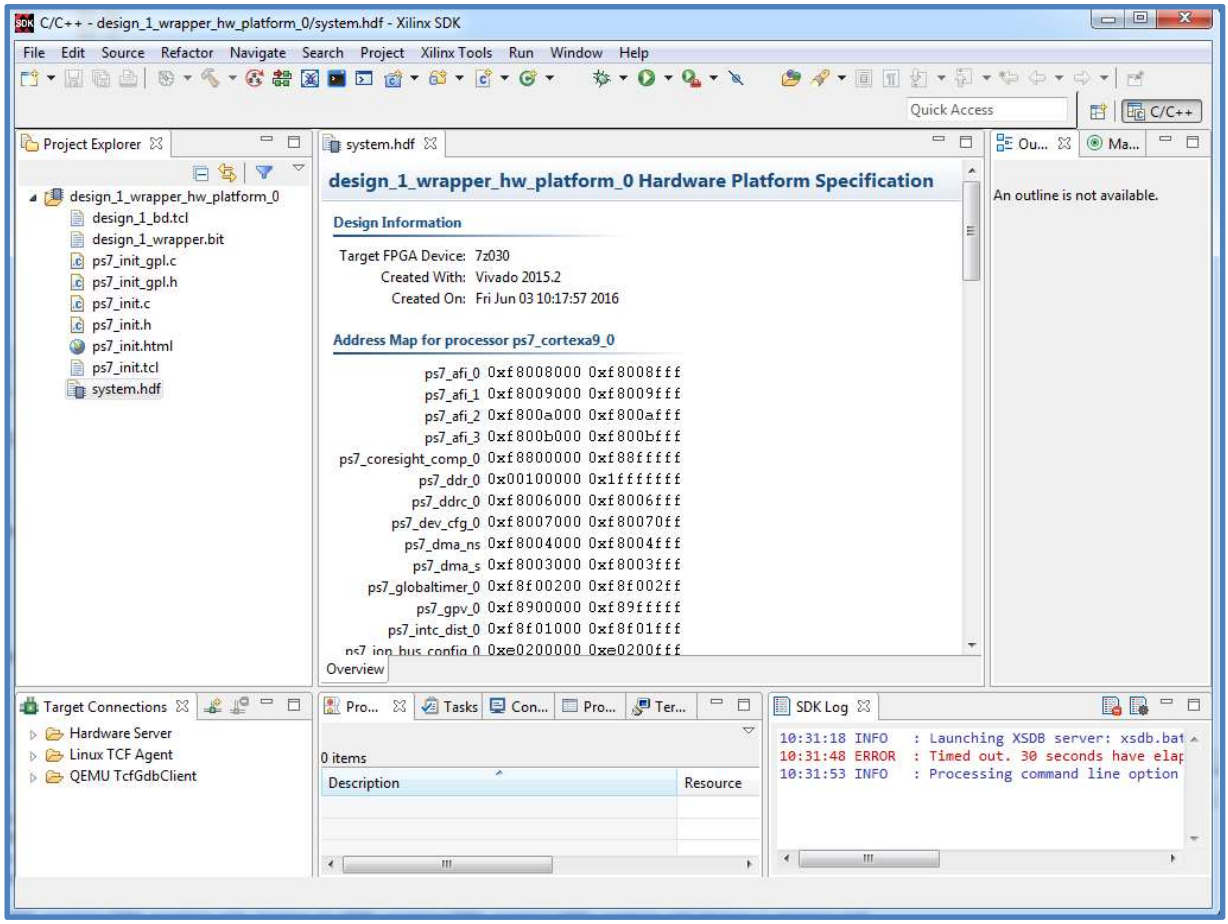


Figure 32 - SDK

To create a FSBL, go to "File -> New -> Application Project"

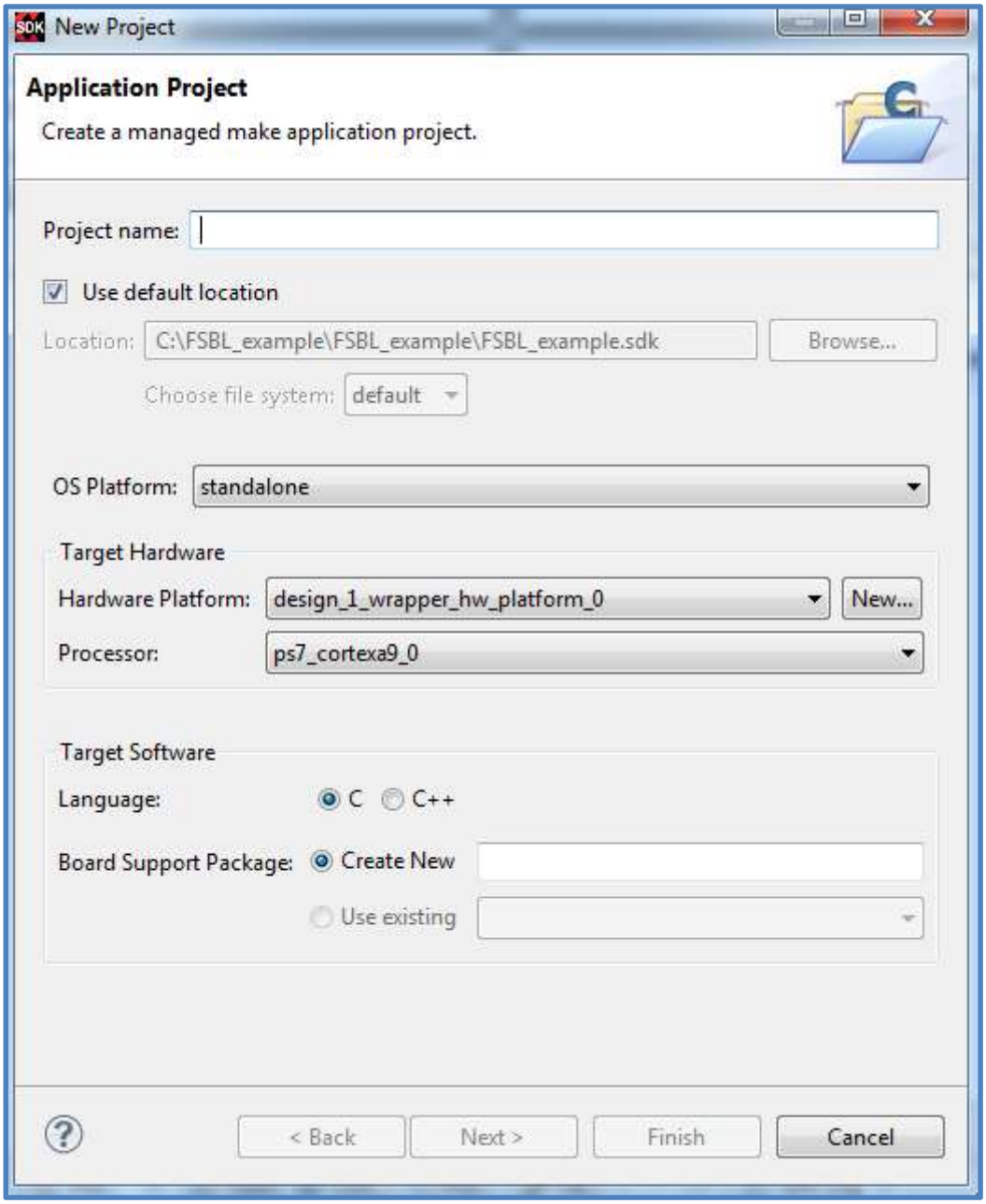


Figure 33 - Export hardware including the bitstream

Give a name to the project, for example “FSBL”. It’s intuitive, as it shows the hardware platform (HDL Wrapper exported previously), the ARM core selected from the Zynq, and the OS platform.

Press next and select “Zynq FSBL

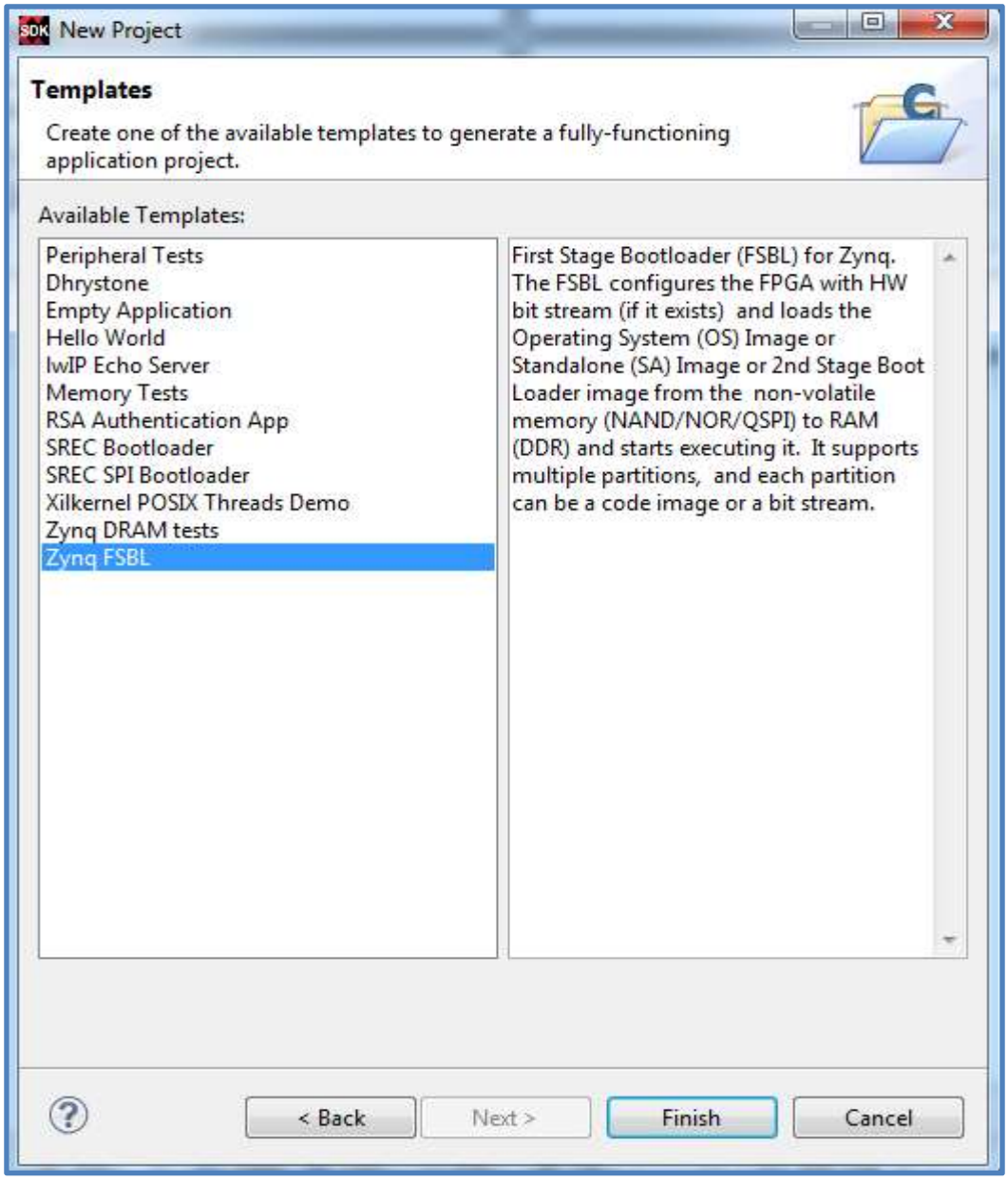


Figure 34 - Zynq FSBL project

Press “Finish” and the FSBL project will be added.

Now the user has a ps7_init source in this project that can be used for initialize the PS at the FSBL. It’s recommended to check out the file system.mss from the BSP project, because depending on the hardware exported, the user can find different code examples from Xilinx for the different peripherals included in the Zynq processing system (UART, DDR, I2C, USB, etc.).

3.3 How can I create SD boot file?

Based on 1.2.2, in this example the FSBL project will be exported as .bif and .bin files, to boot from SD card.

Click “FSBL -> Create Boot Image”

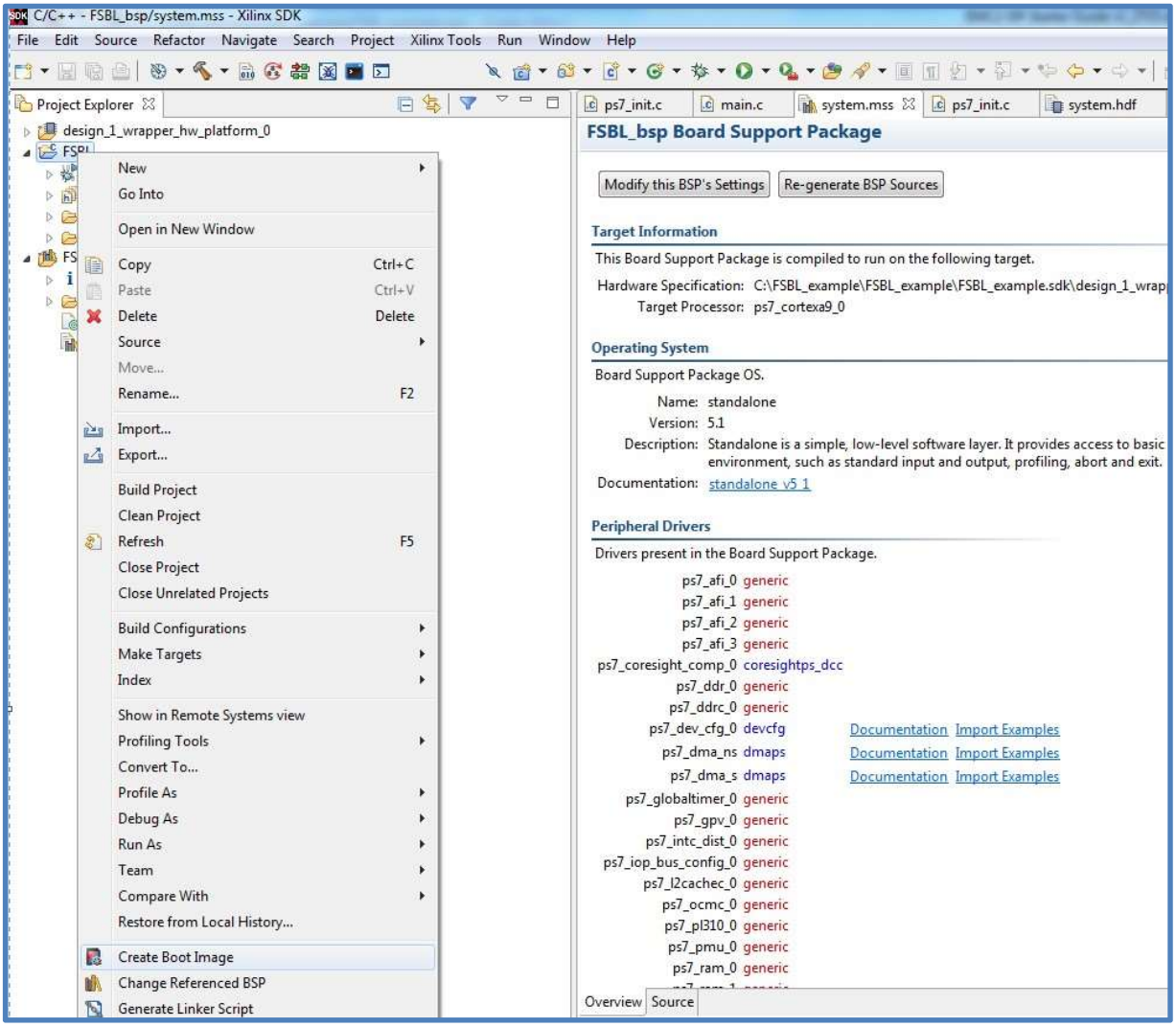


Figure 35 - Create boot image

The user can now see that SDK tries to export a .bif file, from the .elf file (FSBL project in SDK) and the .bit file (Vivado bitstream file).

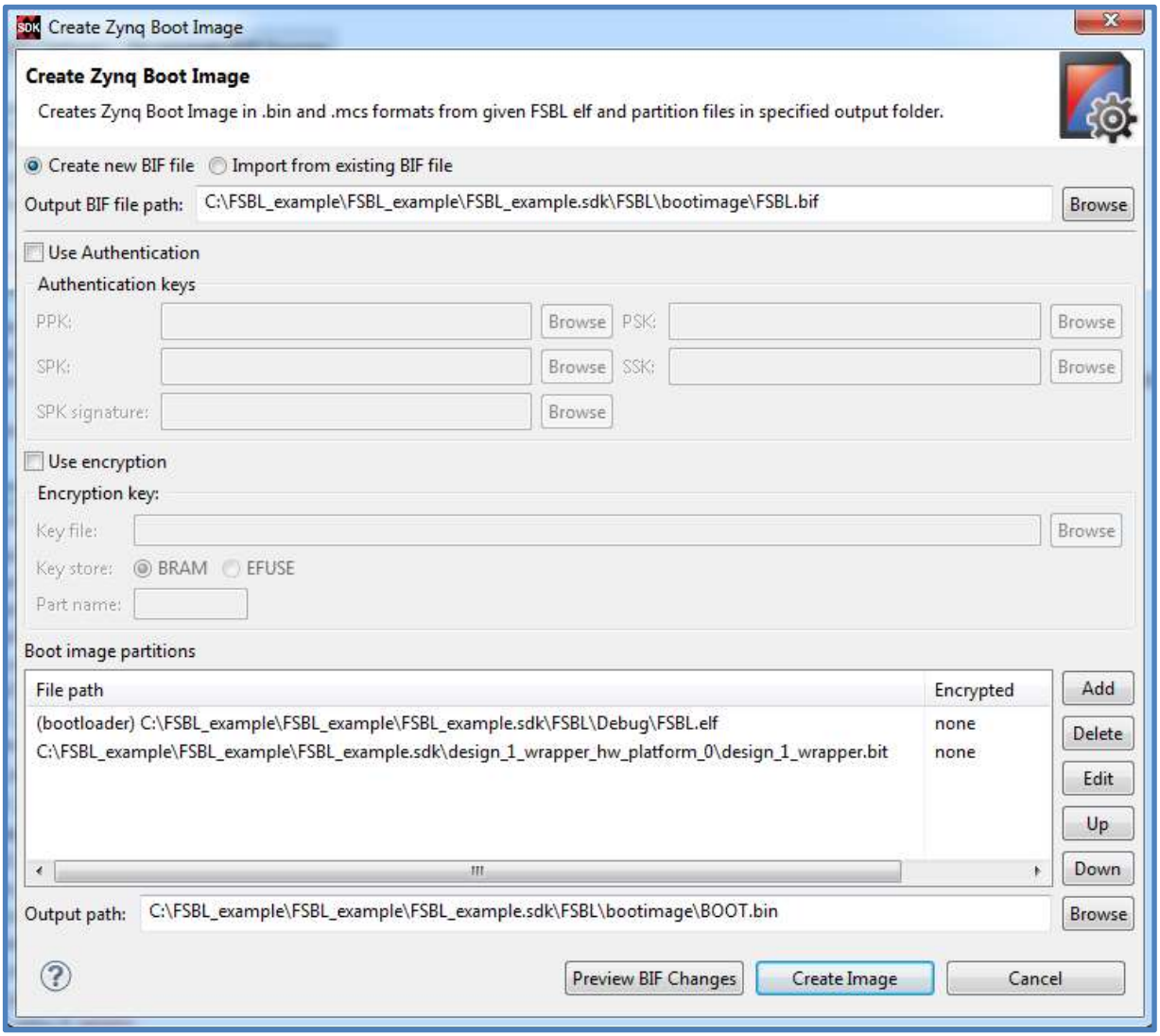


Figure 36 - Create image

Press “Create Image”.
Now in the following path, where “nameoftheproject” was FSBL_example, there should be a .bin file called “BOOT.bin” and a .bif file called “FSBL.bif”
C:\nameoftheproject\nameoftheproject\FSBL_example.sdk\FSBL\bootimage

This .bin file can be included in a SD card, and selecting the jumpers on the EMC² to boot from SD, described in this document, the Zynq will be configured when powering up the board.

Remember that in this case is assumed that the project run in the Zynq is the FSBL itself. The PS needs to be always configured, that’s the reason why SDK assumes the first file as “bootloader”. To run any other application, there should be 3 partitions: FSBL.elf (bootloader), .bit file (PL) and .elf file (application).

These steps are for Windows users.

3.4 How can I program the FPGA from Flash in Vivado?

To program the FPGA from flash, the user needs to erase the flash and program the FPGA with a .bin file.

Open the FSBL_example project created in 1.2.1 in Vivado.

This project has a .bit file generated, but to program the flash a .bin file is needed. To let Vivado know that, it's possible to do it going to "Project Settings" in the Vivado Flow Navigator, and mark the .bin option in Bitstream options.

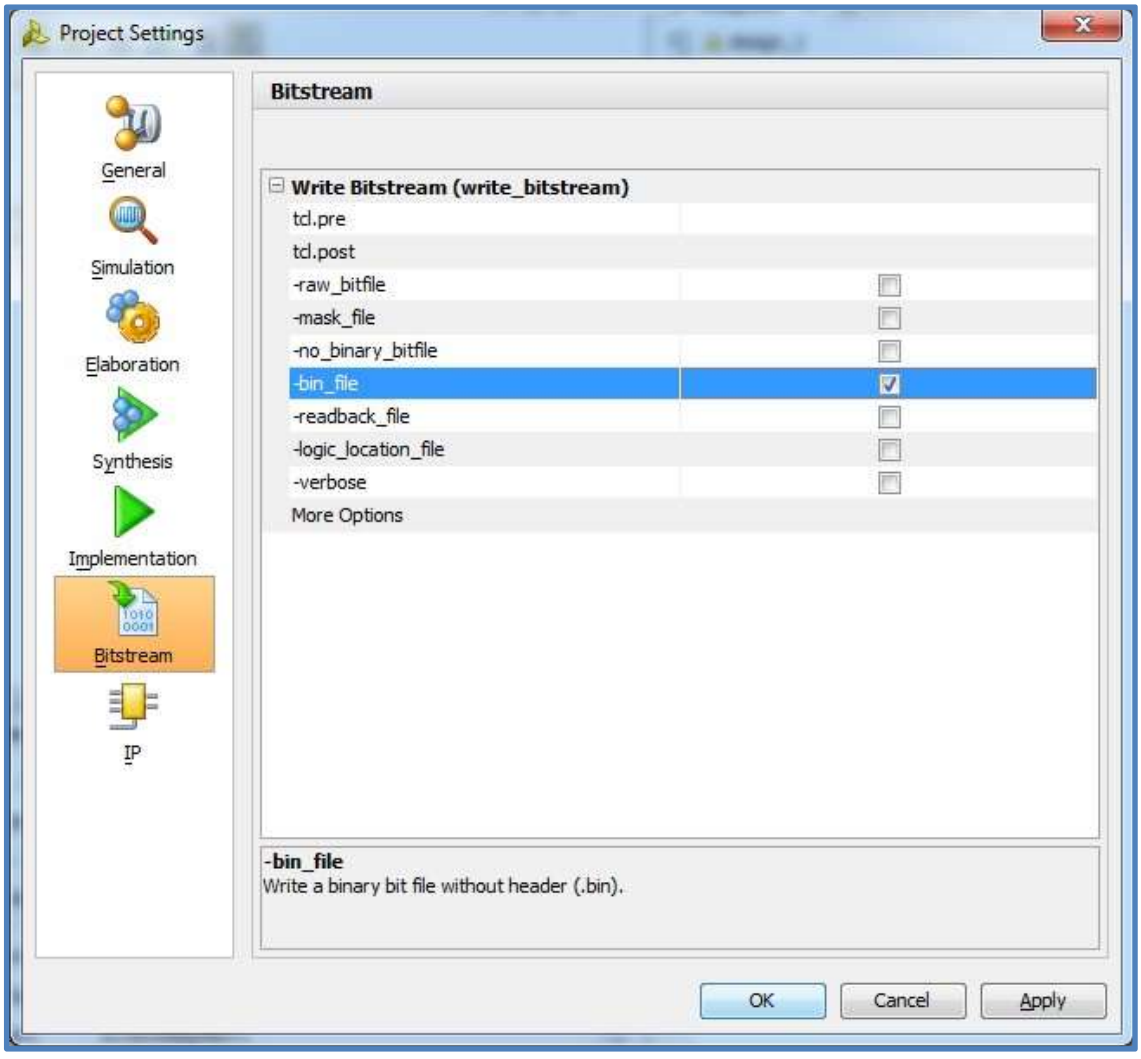


Figure 37 - Project settings

Now rewrite the bitstream, and open the hardware manager.

Connect the EMC² to the computer through JTAG, and power the board up:

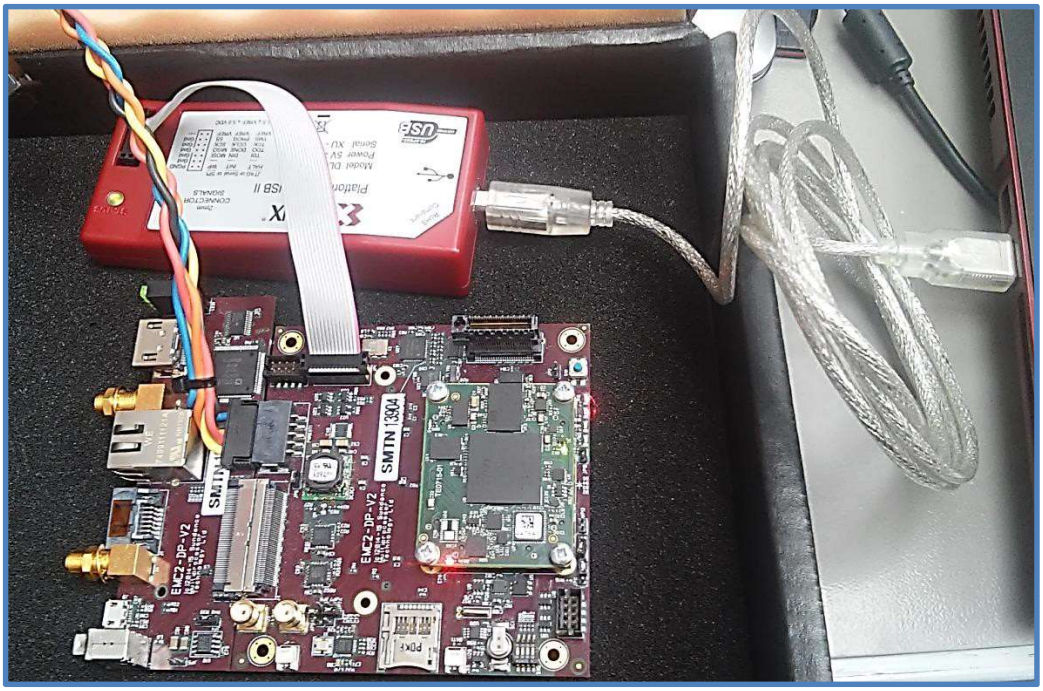


Figure 38 - Set the hardware and open the Hardware Manager

Open a new target, and the EMC² will be recognized by the software

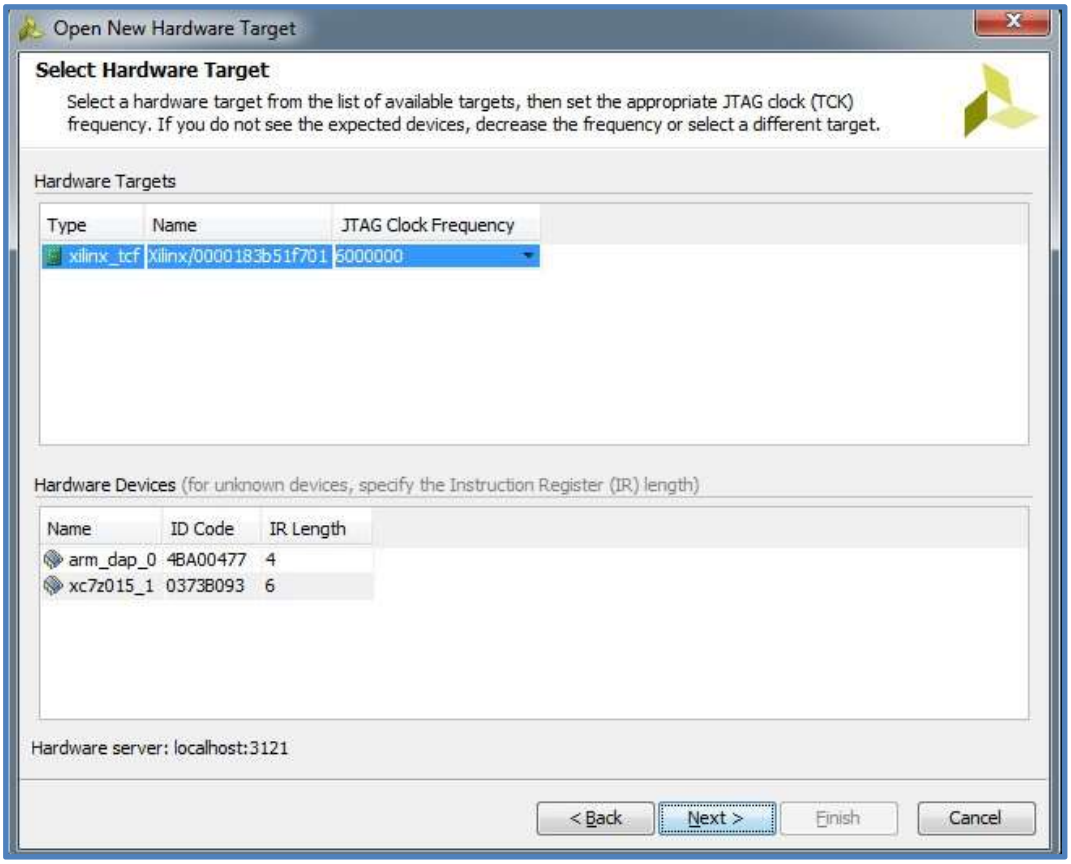


Figure 39 - Open target

The user can see at this point that the hardware manager sees the PL and the PS from the Zynq (FPGA + ARM). Press “Next” and “Finish”.
Not to program the Flash, we have to Click “Add Configuration Memory Device”

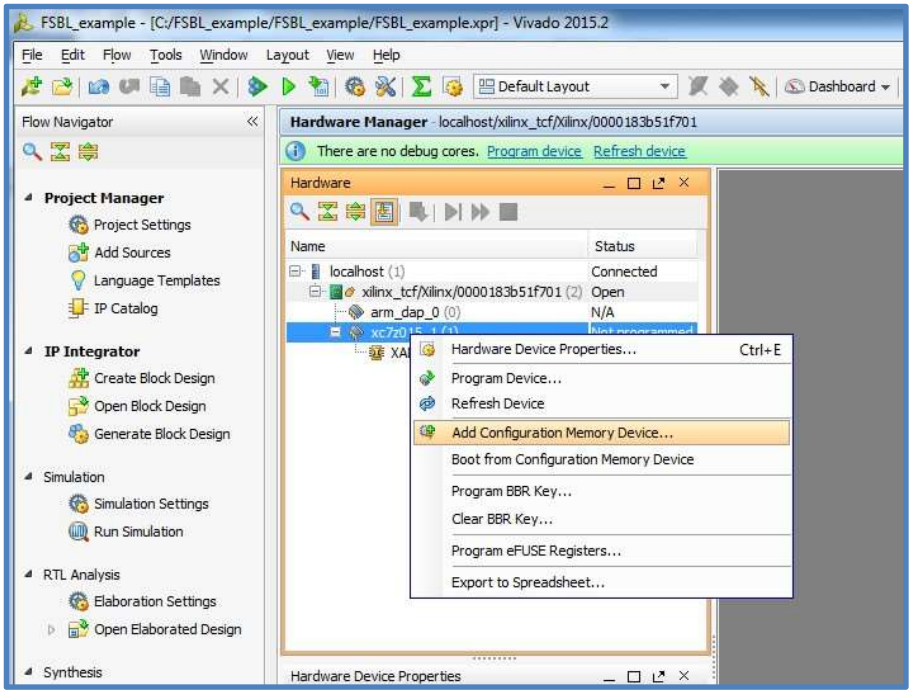


Figure 40 - Add Configuration Memory Device

Select the Flash device, in this case S25FL256S (Spansion).

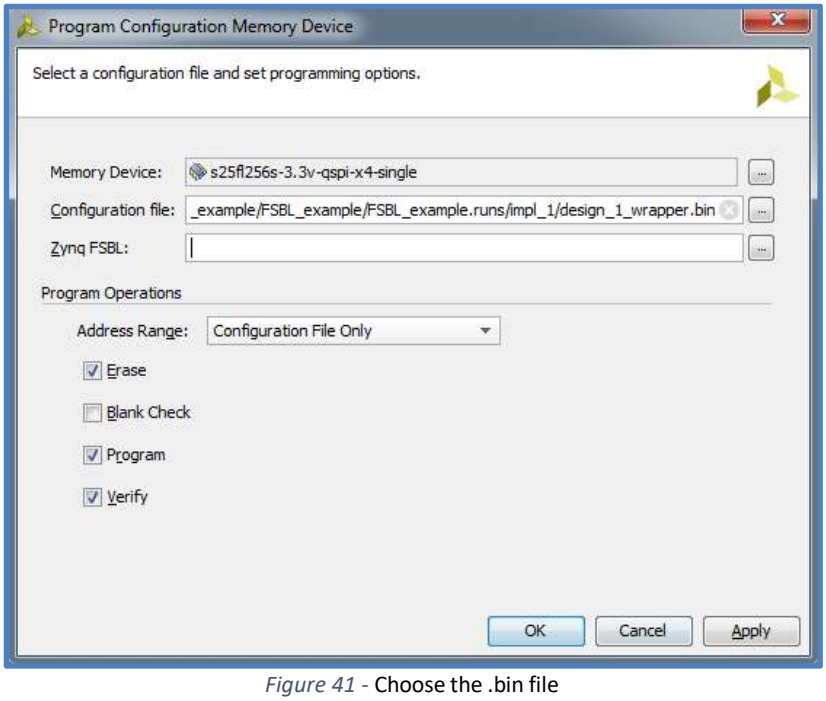


Figure 41 - Choose the .bin file

Choose the .bin file, which should be in the following path, and program the flash.
C:/FSBL_example/FSBL_example/FSBL_example.runs/impl_1/design_1_wrapper.bin

3.5 How can I create a HDMI test project?

To test the HDMI interface, Trenz provides a design here.

In this document it will be shown how to create a working project, which displays a picture through the HDMI port, using Trenz IP Cores.

This project currently works in Vivado 15.2. The tutorial shown from here has been done with the EMC²-Z7015 Board files, and the constraints file provided.

Open a new project in Vivado, and add into the IP catalog the repositories needed for the project, which include the cores “video_io_to_hdmi” and “axis_fb_conv”.

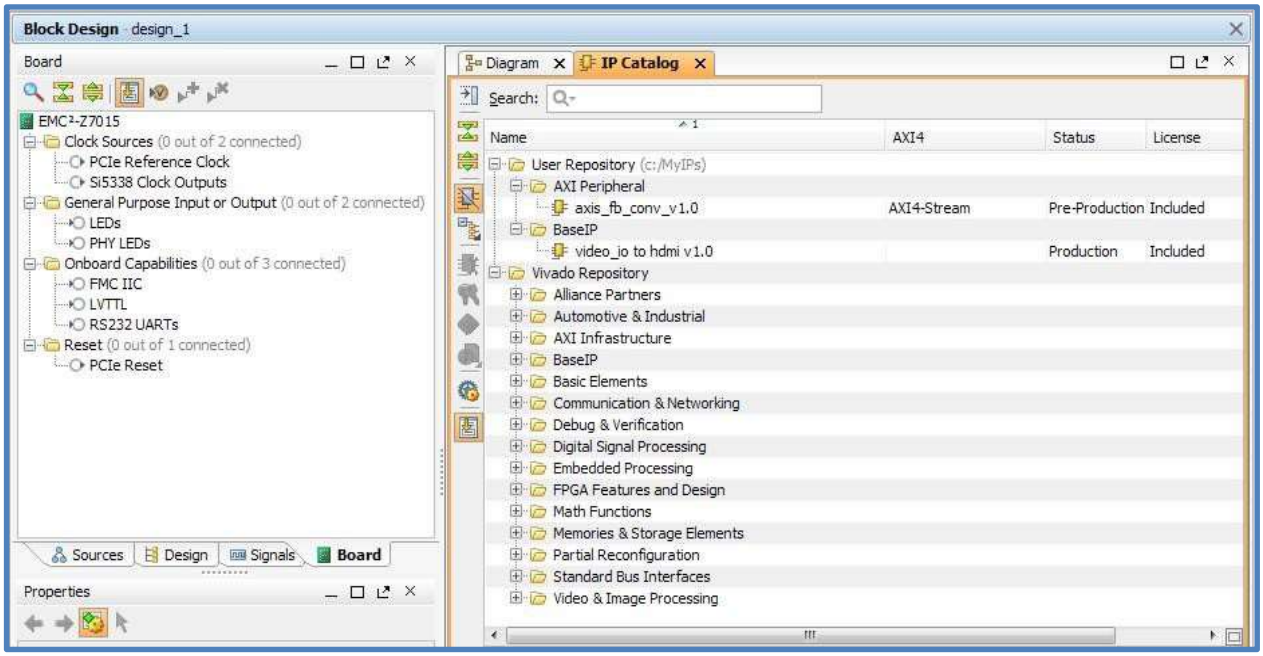


Figure 42 - Adding repositories

Create a new block design in IPI, and add a Zynq Processing System. For this project it will be needed I²C, Uart and SD capabilities assigned to the MIO pins.

The I²C will be the protocol of communication, and the project will be booted through an SD card.

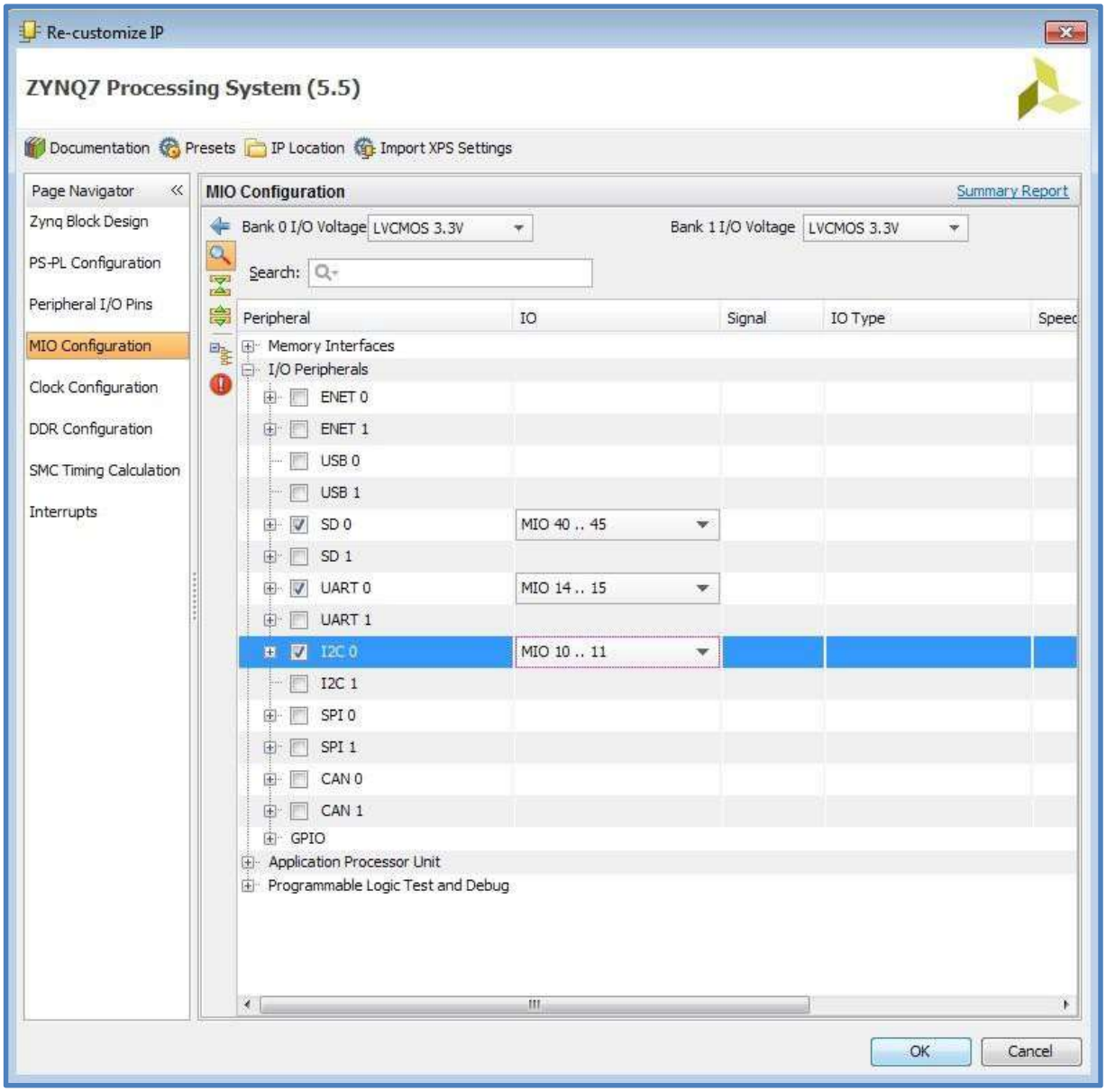


Figure 43 - Selecting MIO pins

Configure the clock outputs of the Zynq. For this project there will be 2 main clock sources, 100MHz and 75MHz respectively.

The 100MHz clock will feed the AXI interconnections, while the 75MHz clock will be related to the video controller blocks.

NOTE: The second clock is originally generated from the ARM PLL, not from IO PLL.

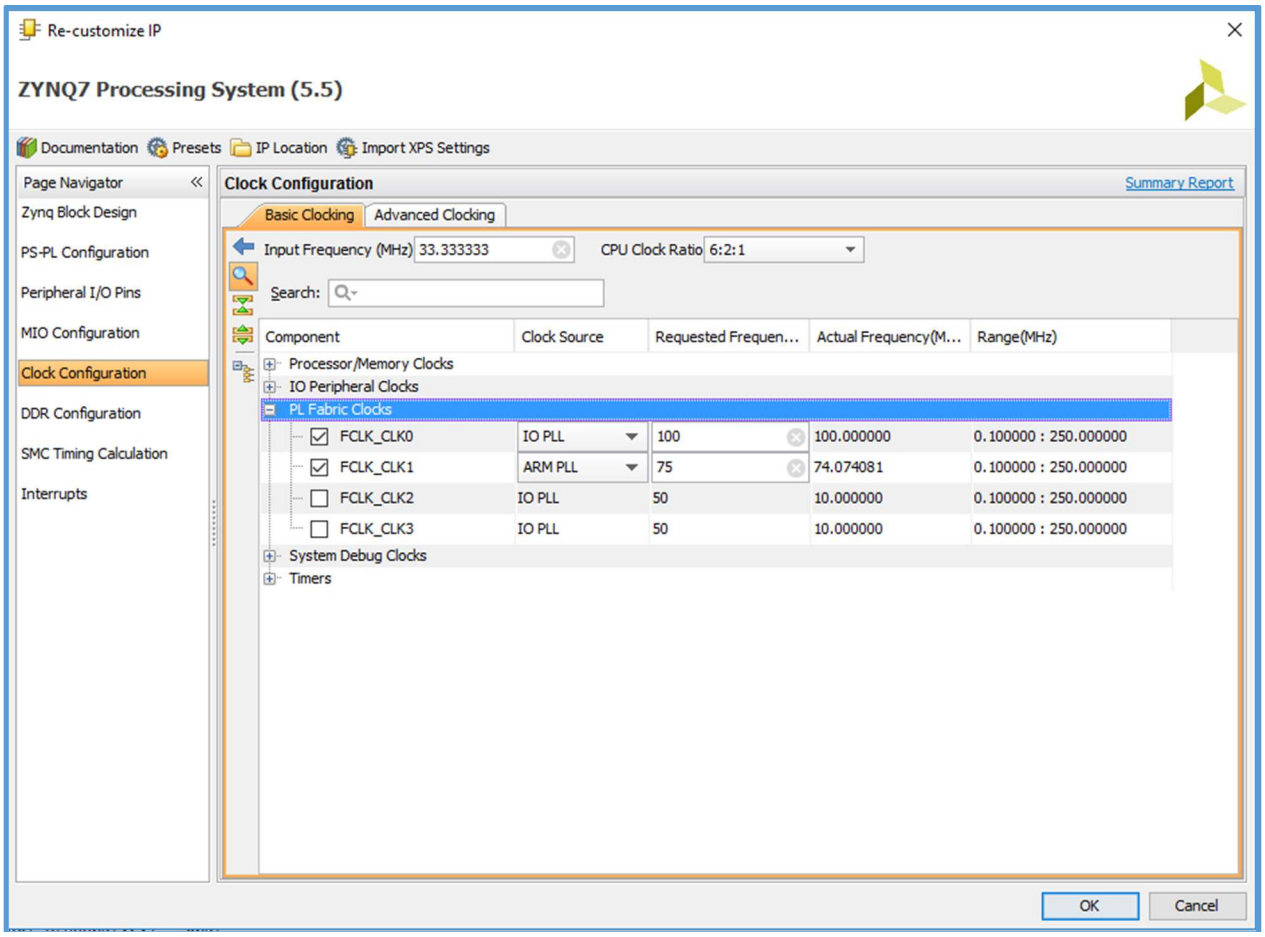


Figure 44 - Configuring clocks

Activate the interrupts as well.

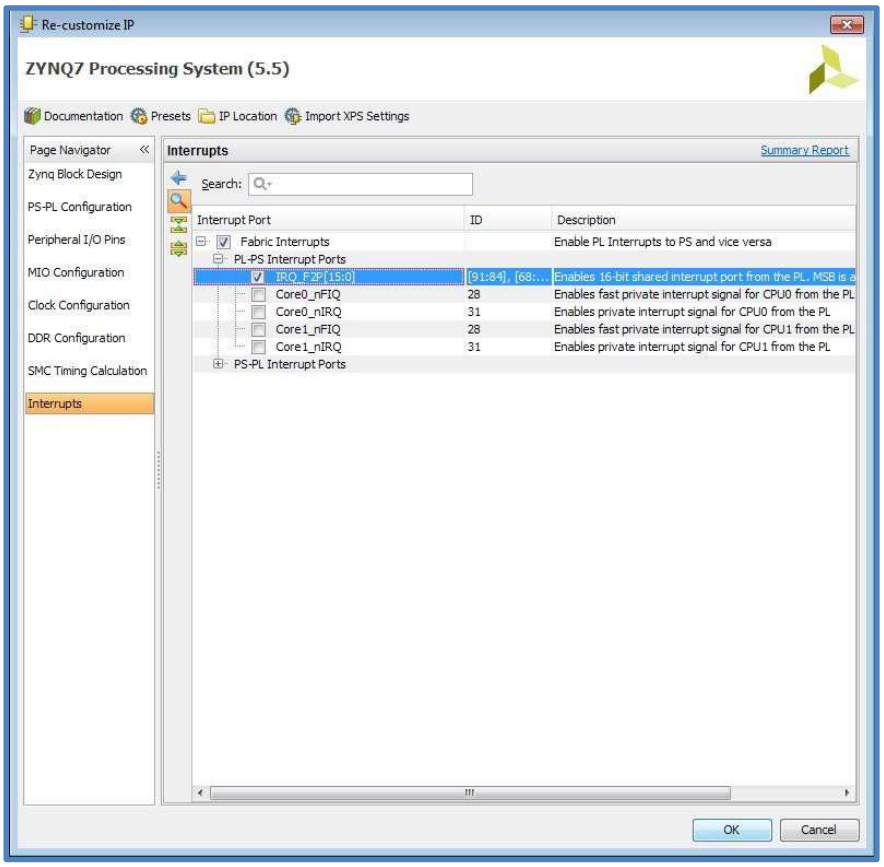


Figure 45 - Interrupts

Run Block Automation, and the Zynq will assign the DDR and FIXED_IO ports.

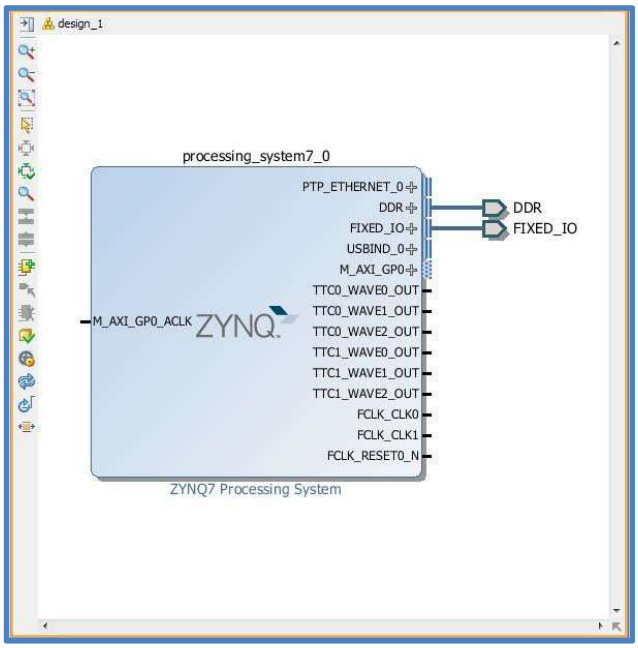


Figure 46 - Run Automation

Add the Video Direct Memory Access, Test Pattern Generator and Video Timing Controller blocks.

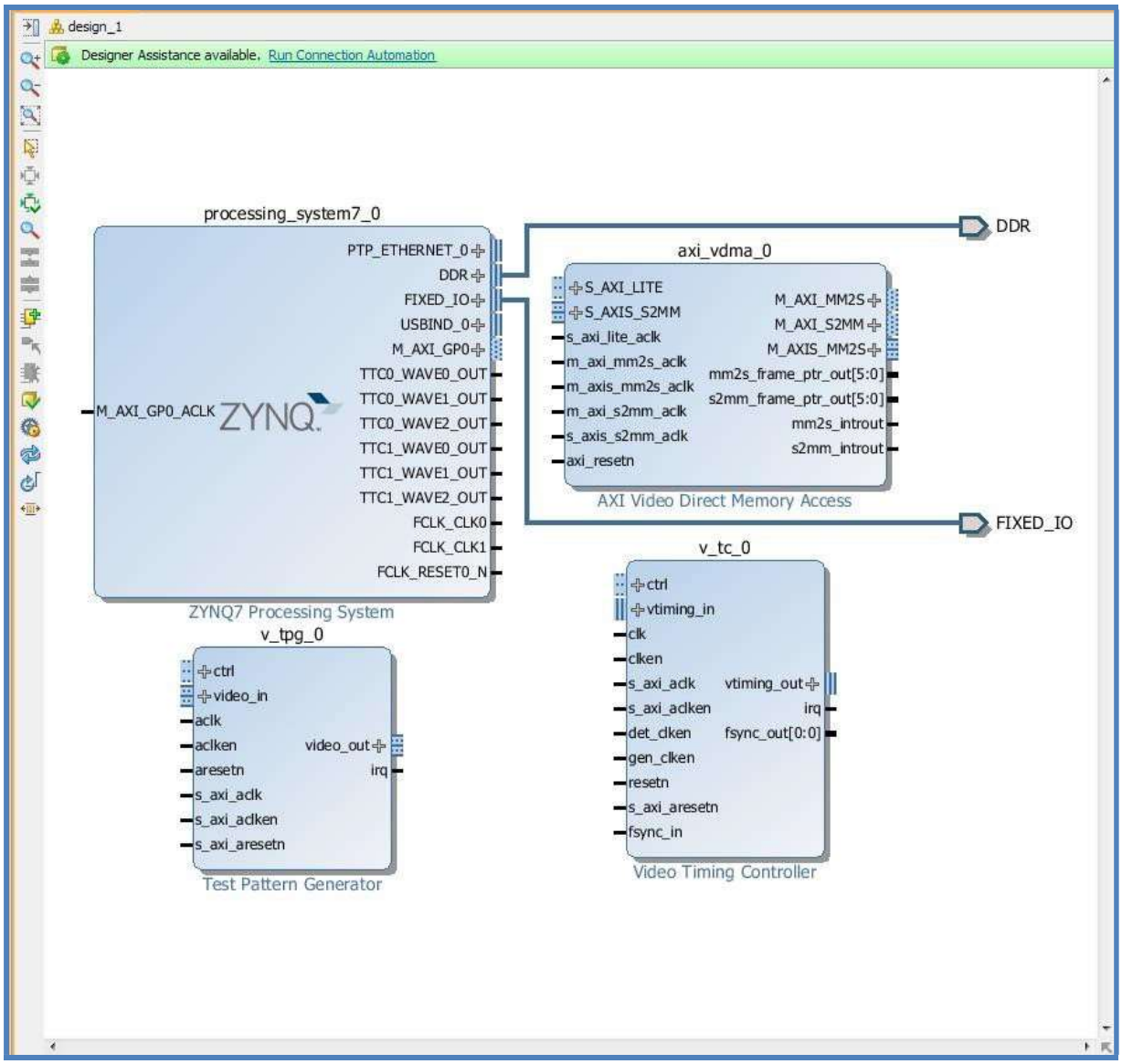


Figure 47 - Adding blocks

Configure the Video Direct Memory Access block as follows.

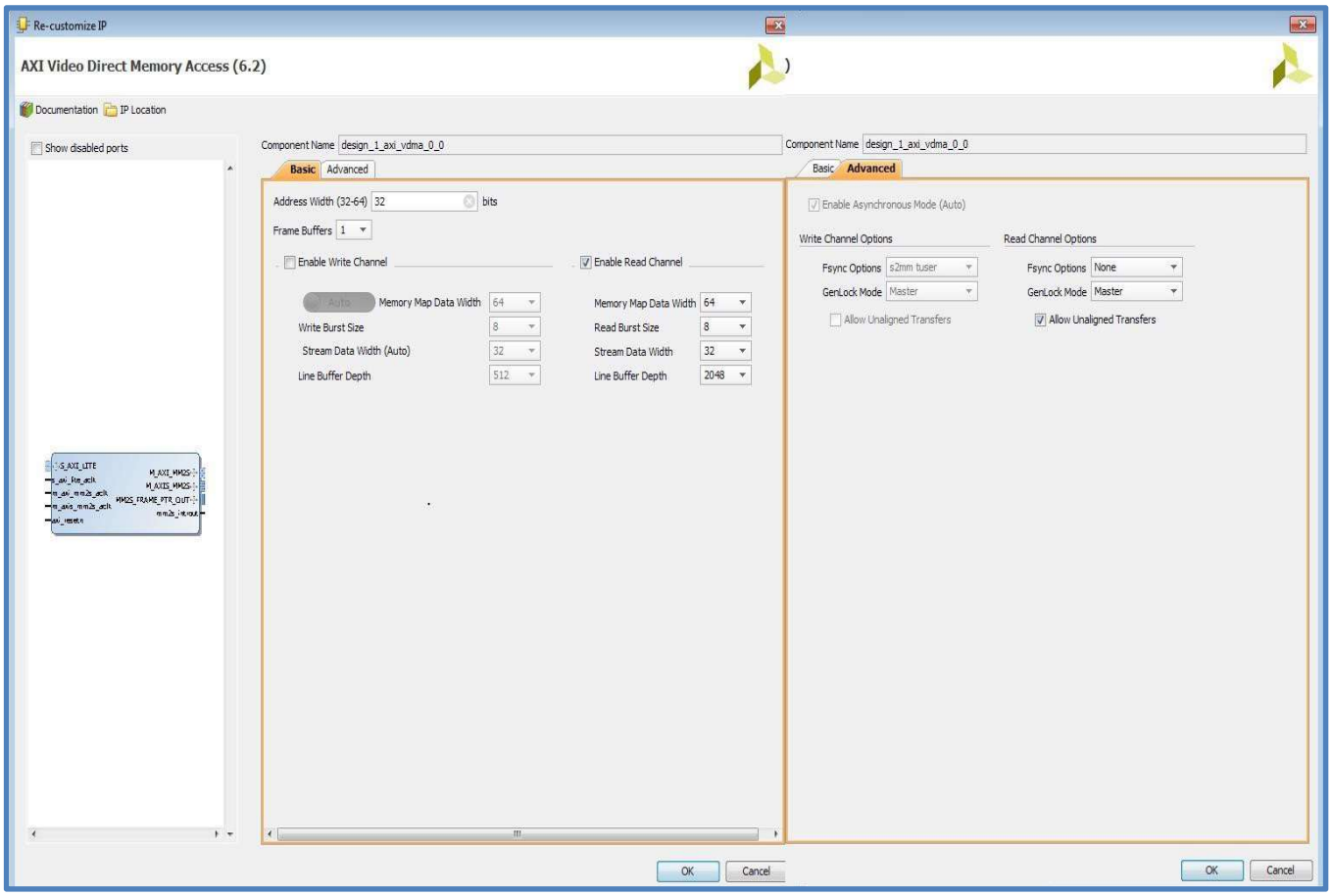


Figure 48 - Configuring DMA

The Video Timing Controller as follows.

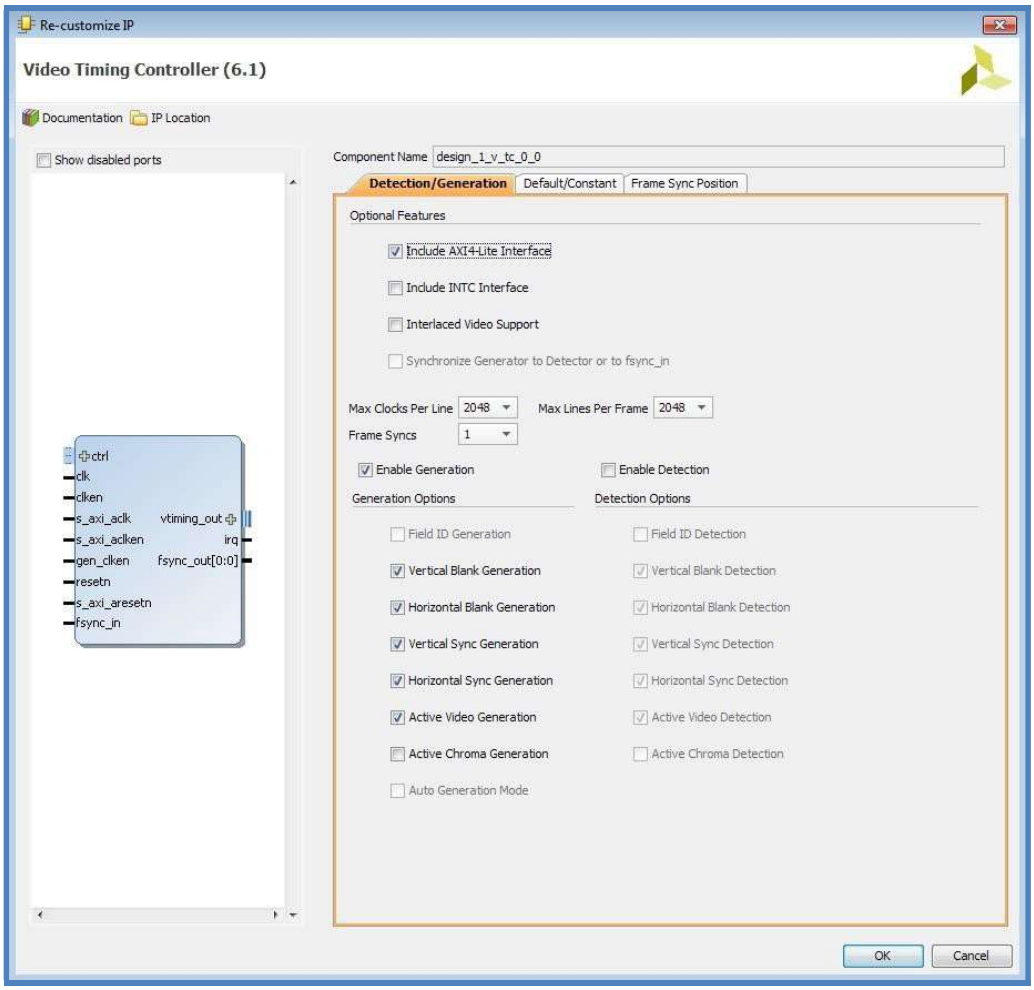


Figure 49 - Configuring Video Timing Controller

The Test Pattern Generator as follows.

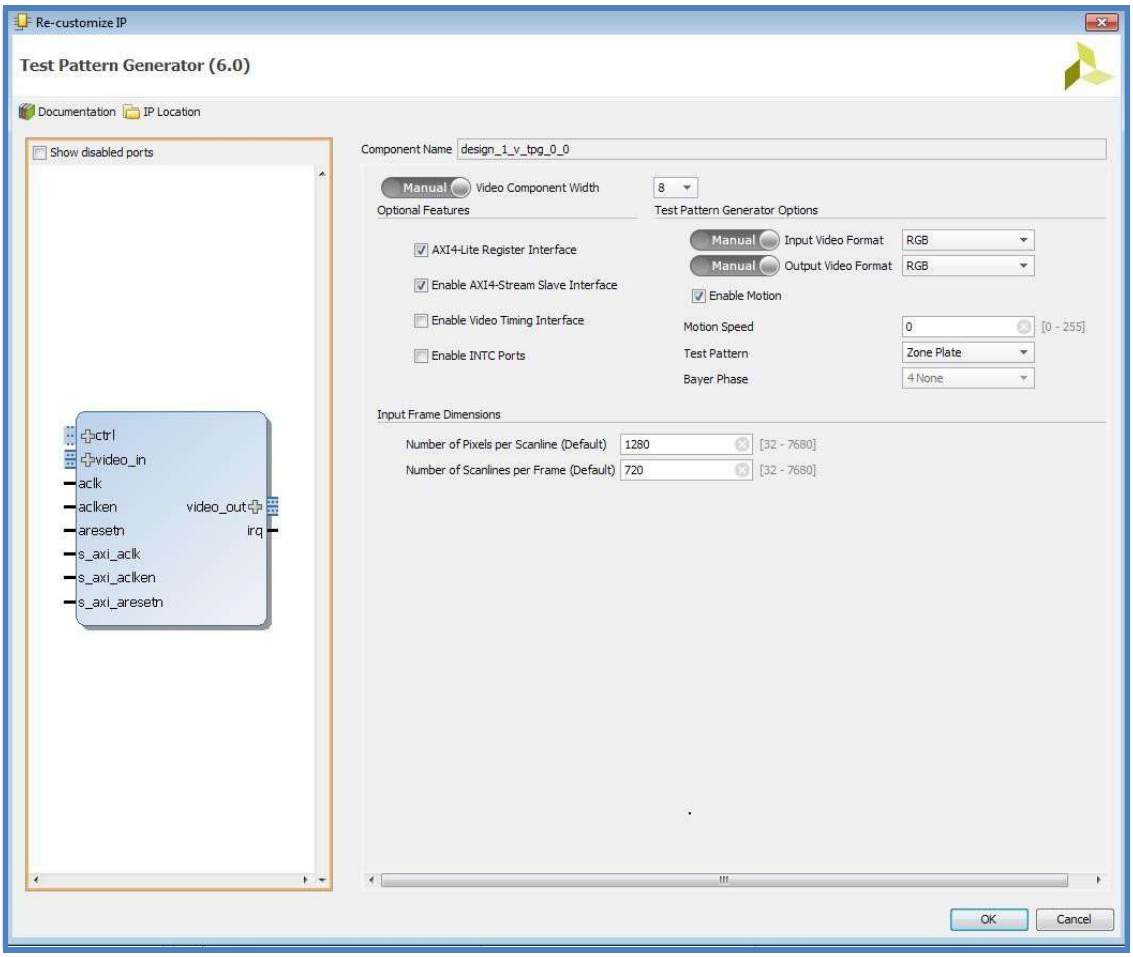


Figure 50 - Configuring Test Pattern Generator

Running Connection Automation, assign the blocks to CLK0, which should be configured as 100MHz.

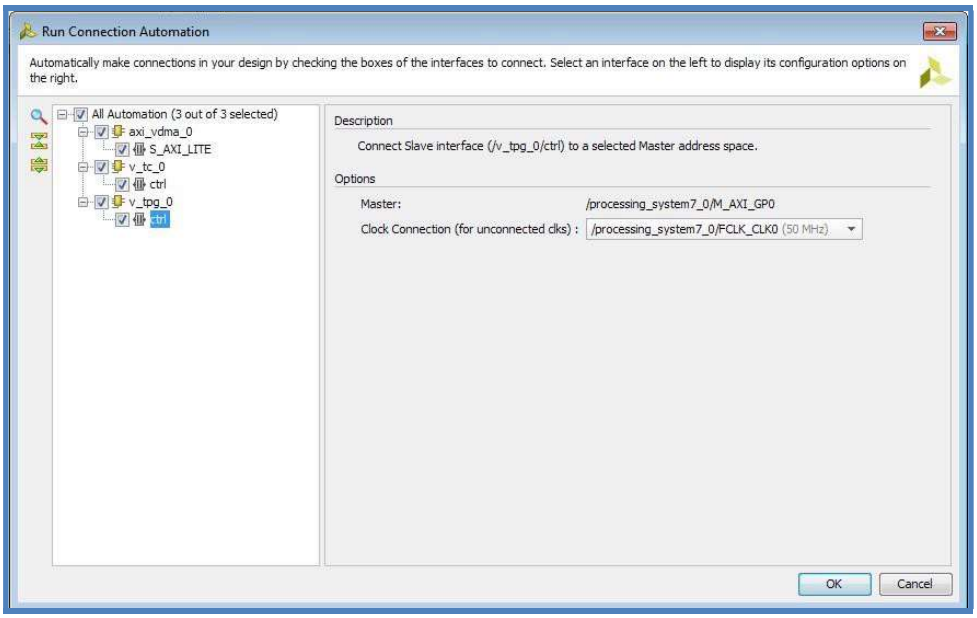


Figure 51 - Assigning clocks

Select the Zynq as Master of the system.

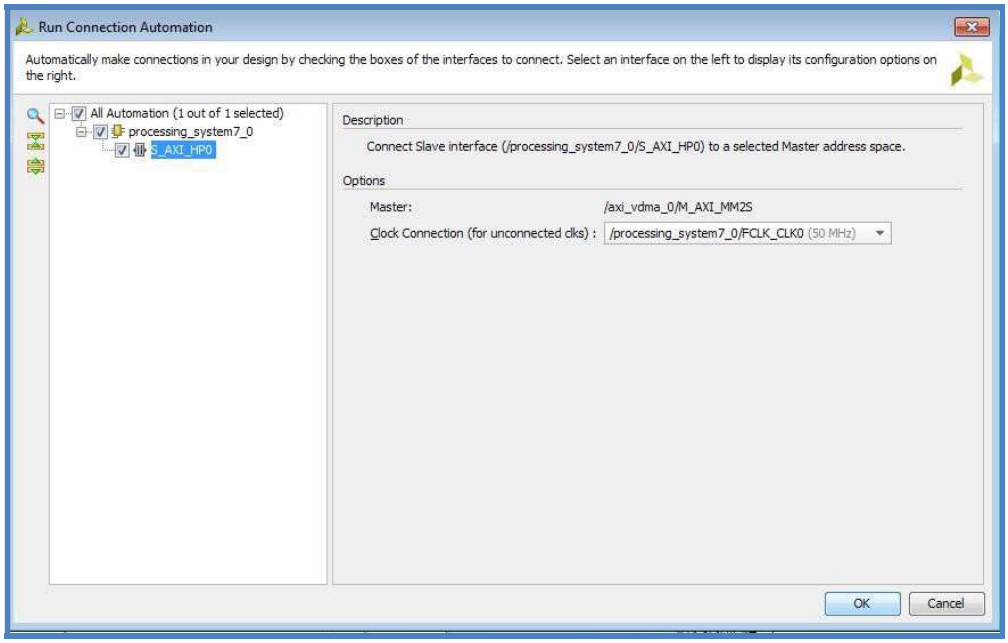


Figure 52 - Adding blocks

The system should look like this for now, having the Zynq as Master of the three blocks which will be the interface with the HDMI onboard.

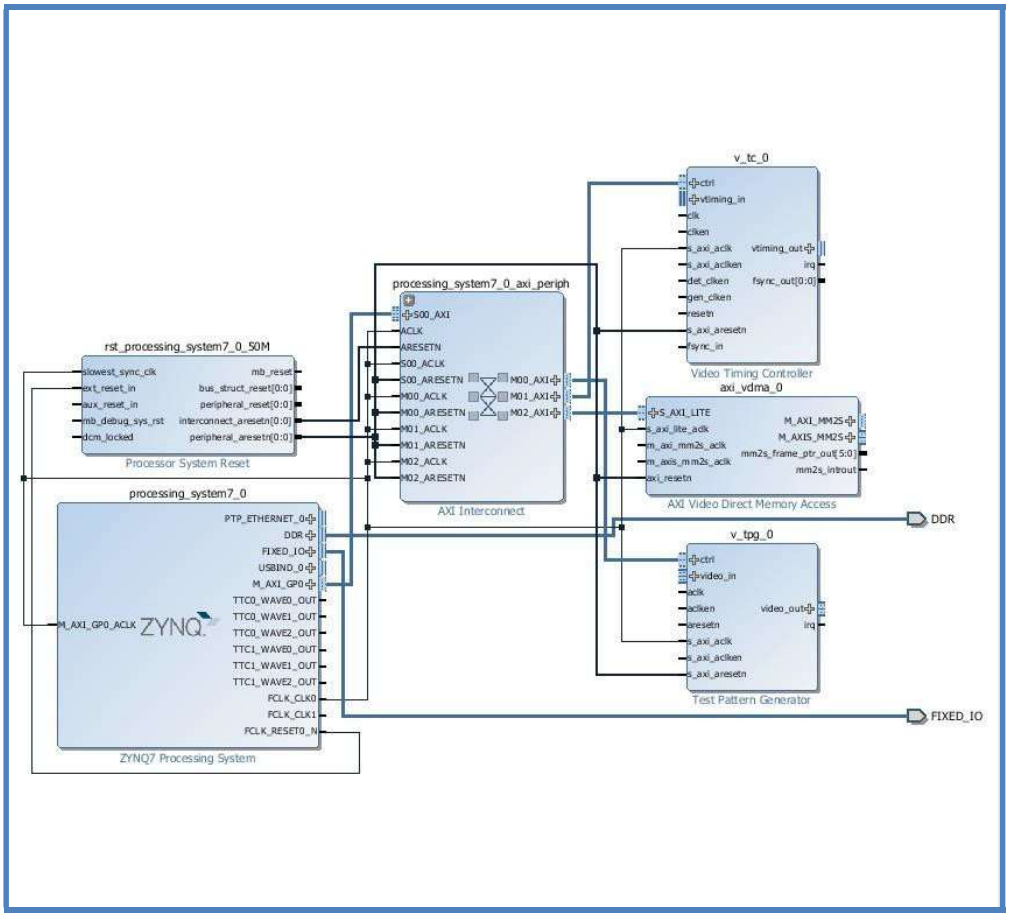


Figure 53 - First look

Open a slave port in the Zynq, so that the DMA can be Master of it.

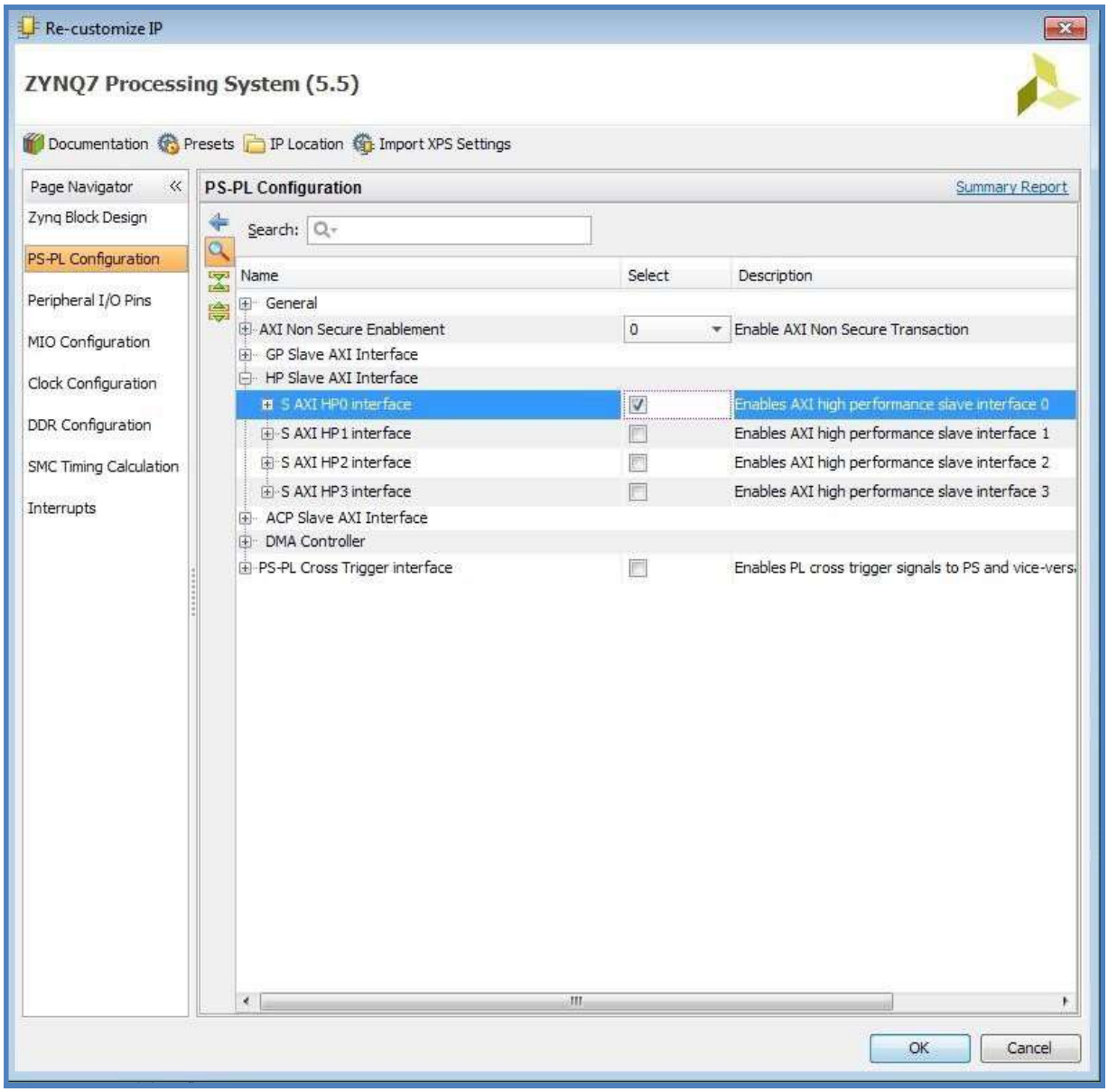


Figure 54 - Adding a slave port

And the system looks like this after this last step.

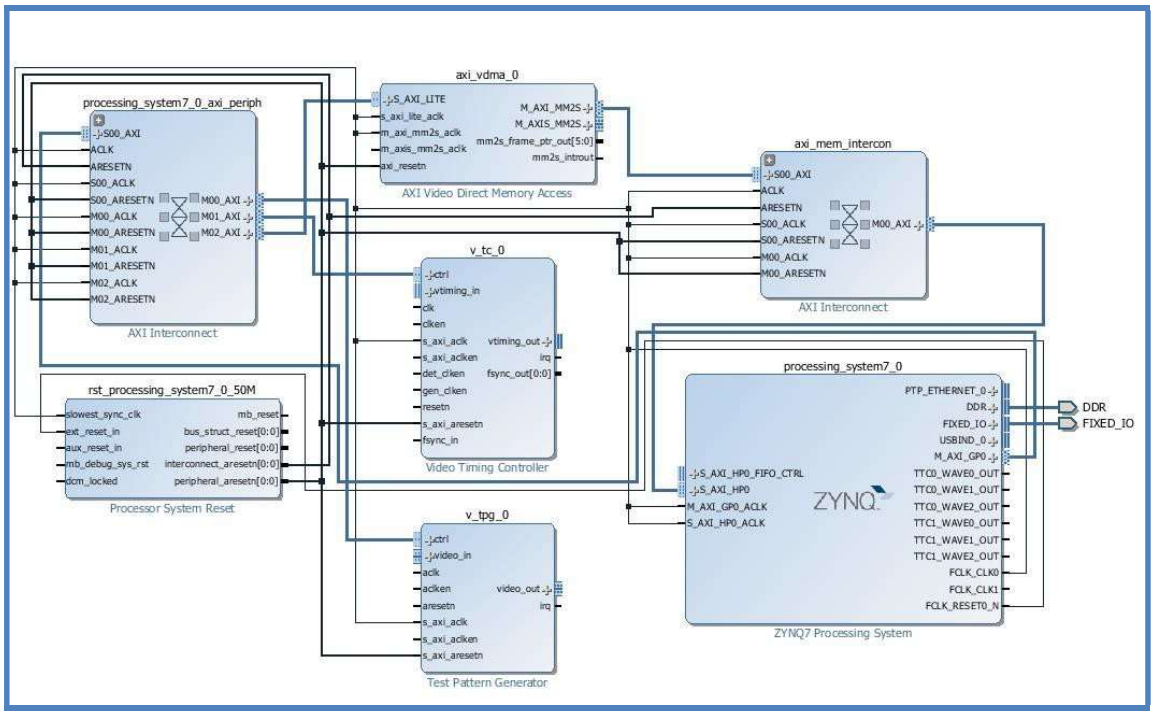


Figure 55 - Second look

Make the DMA Master of the axis_fb_conv block, whose output goes to the Test Pattern Generator.

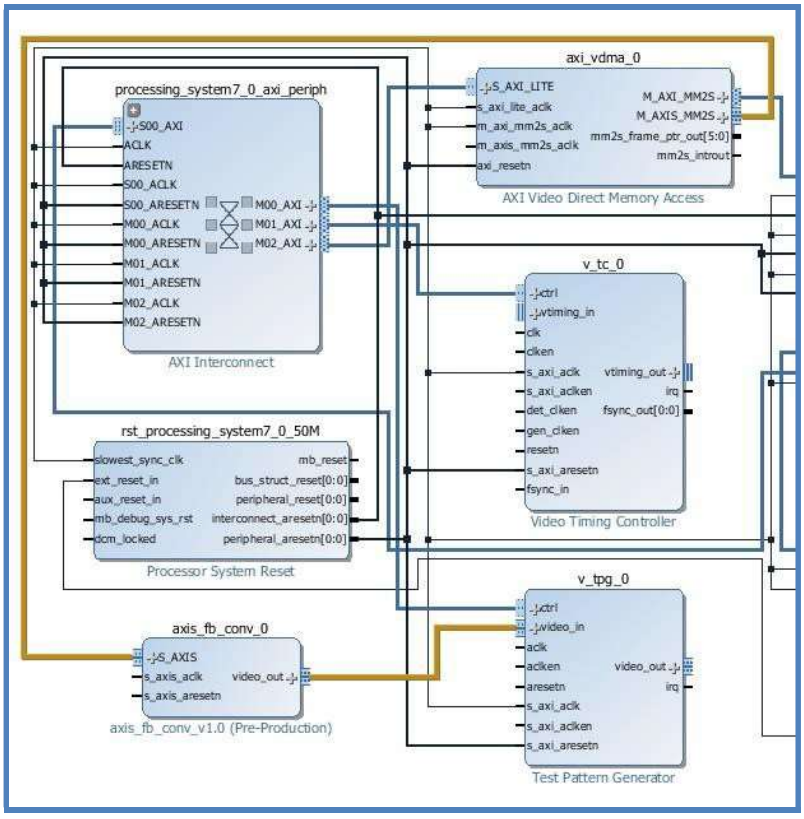


Figure 56 - Making connections

Connect the following outputs into a AXI4 Stream to Video Out block, and its output to the video_io_to_hdmi block provided by Trenz.

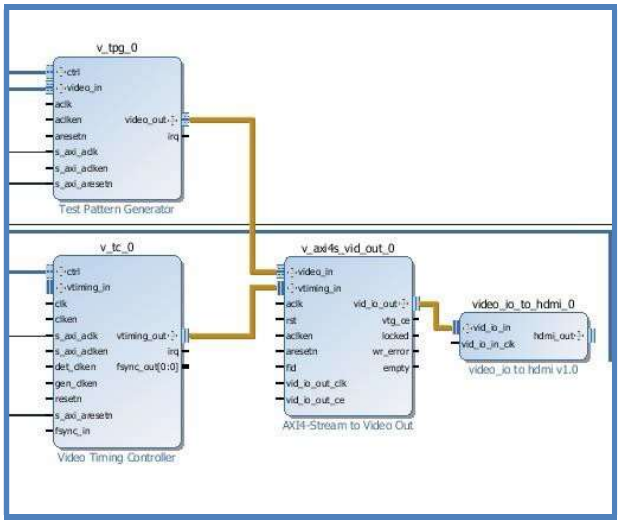


Figure 57 - HDMI Interface blocks

Assign the 75MHz clock to the corresponding pins, respecting the resets and AXI Interconnect pins.

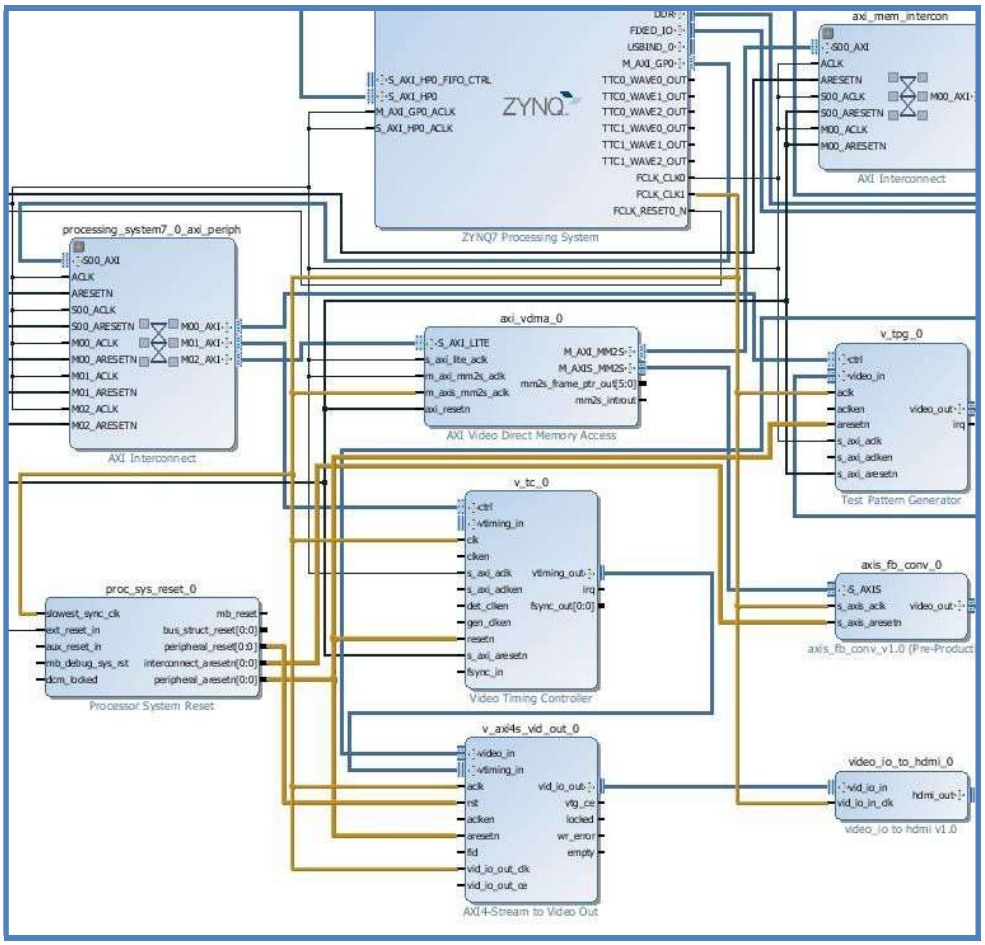


Figure 58 - Clock connections

Create the ct_hpd and ls_oe ports, assigning “VCC” to them, as well as the clock enables.

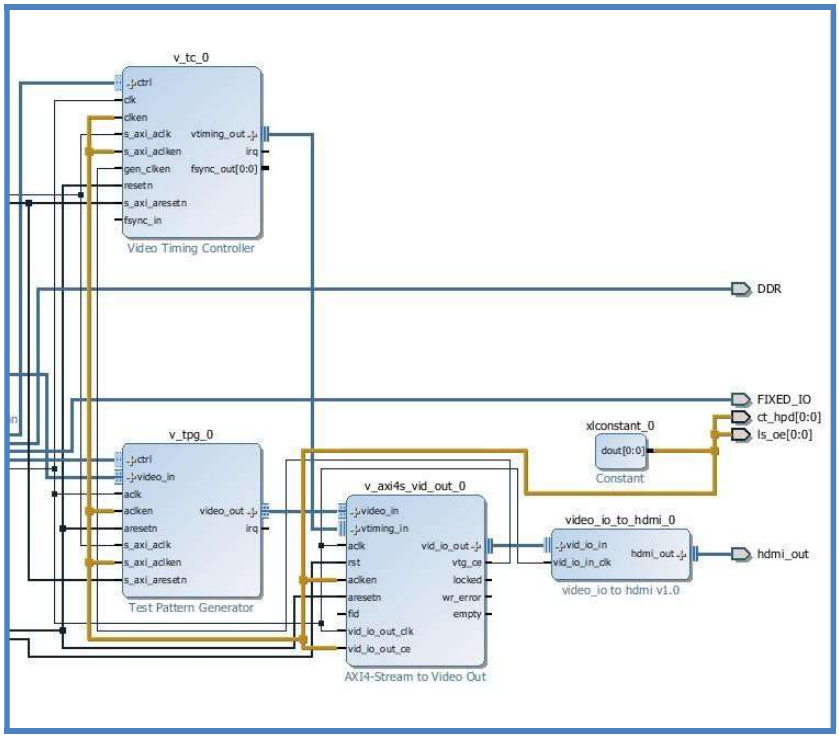


Figure 59 - Adding ports

Connect the rest of the general clocks, and the clock enable of Video Timing Controller from the Stream to Video Out output.

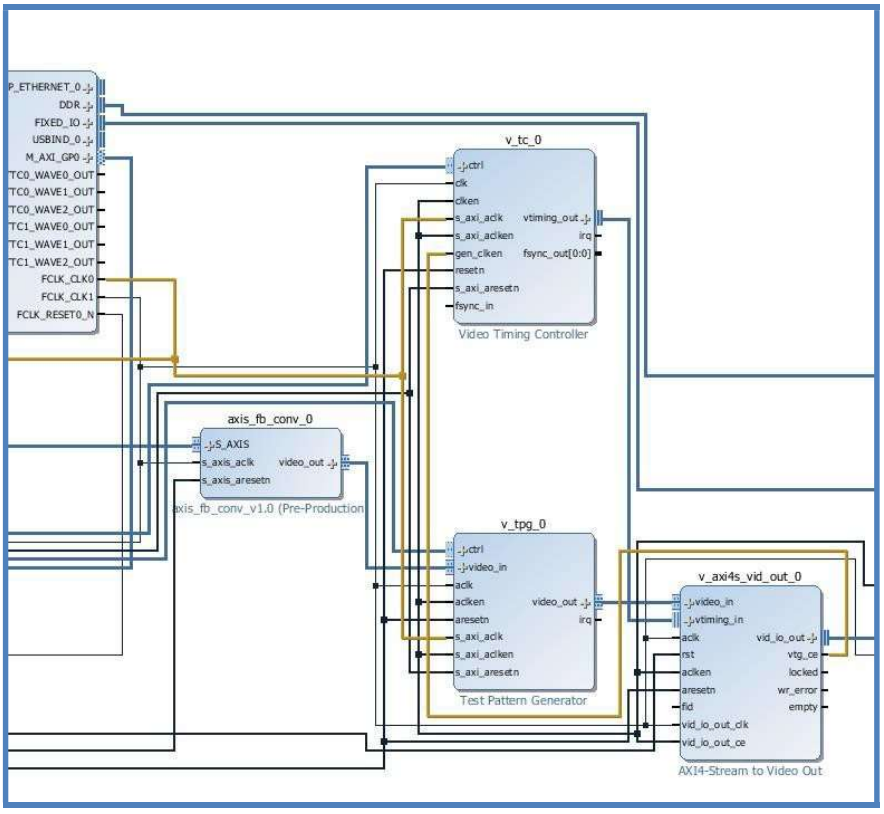


Figure 60 - Connecting the rest of the clocks

Configure the interrupts for the Zynq

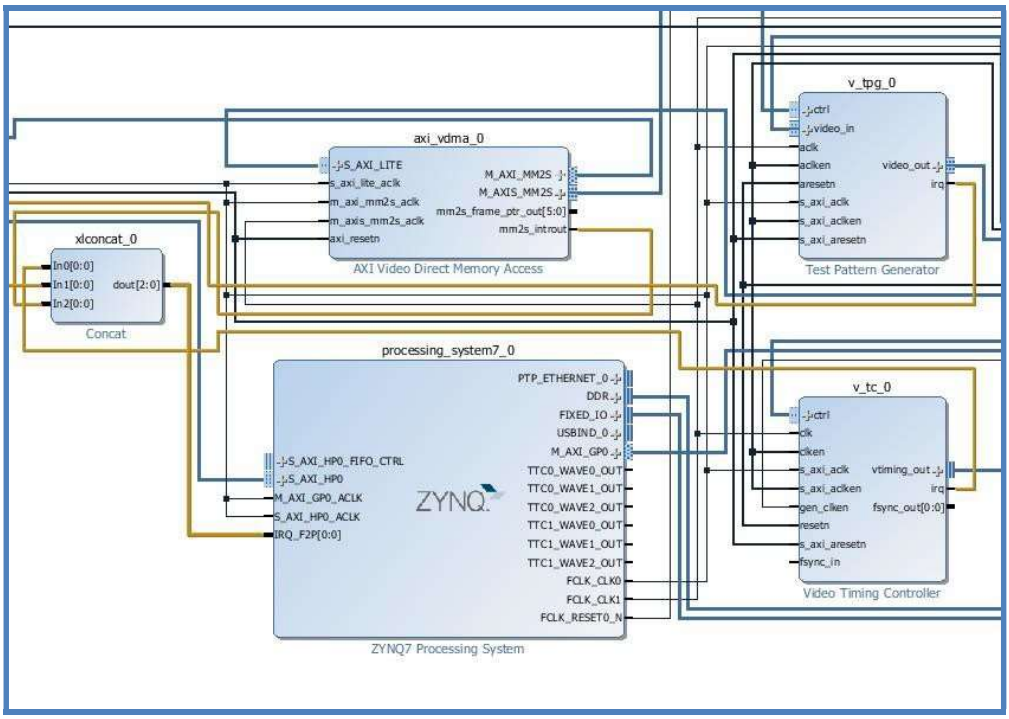


Figure 61 - Connecting interrupts

Create the cec_clk port, and connect it to “ground”.

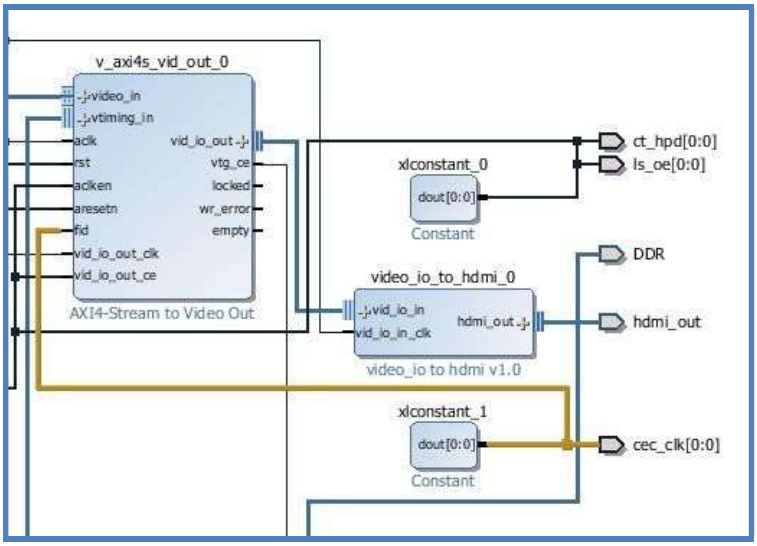


Figure 62 - Adding port

Validate the design, and write the Bitstream.

Export the hardware and launch the SDK. Create a new application project, called FSBL, and import the HDMI application.

To create the boot file, 4 files are needed:

-FSBL.elf file, to configure the PS

-.bit file generated in Vivado.

-hdmi.elf file, application which configures the HDMI chip.

-picture.rgba, addressed with a value of 0x38000000, which is the picture shown in the screen.

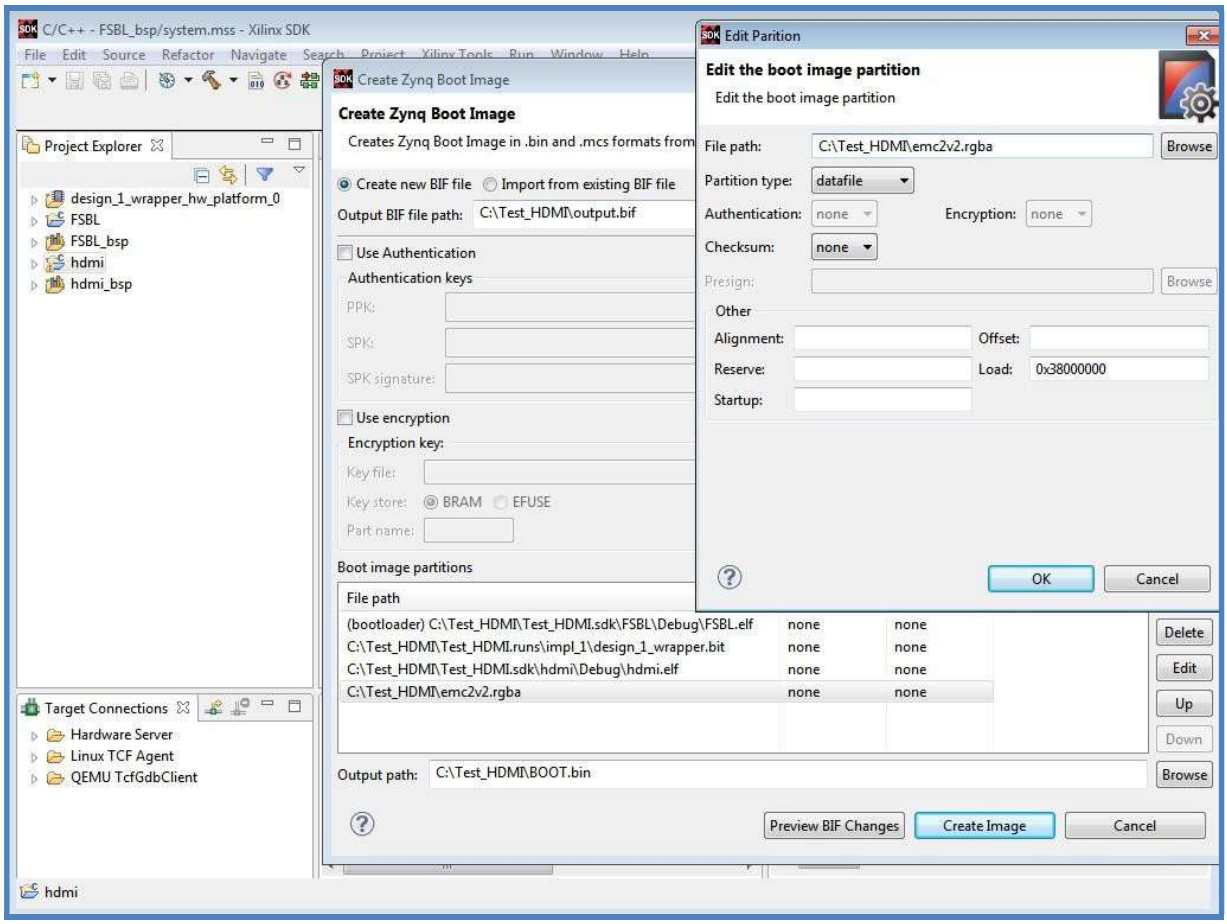


Figure 63 - Creating Boot Image

Create the image, and place the .bin file into an SD Card.

Powering up the board, the PS will be configured, the hardware implemented in the PL, and the HDMI application will be run, generating the picture in the screen.

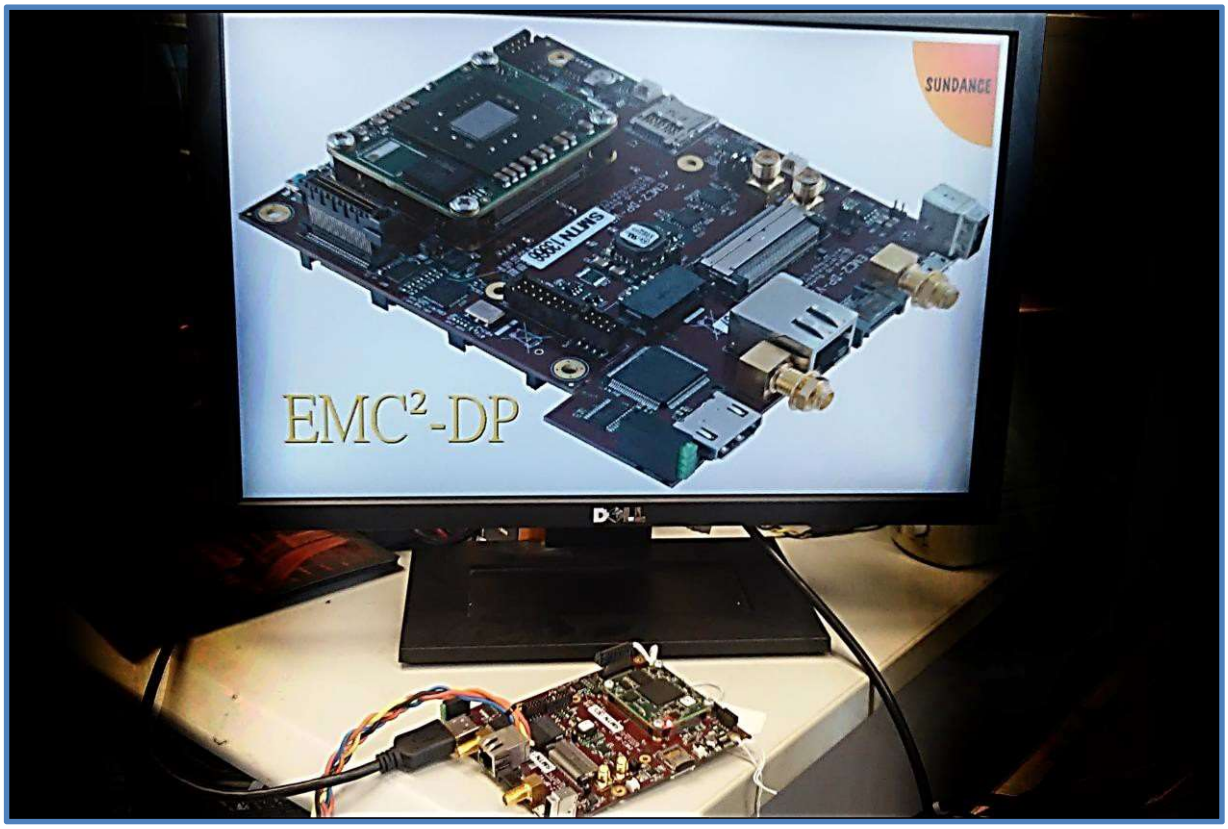


Figure 64 - Test running

3.6 Upgrading

3.6.1 To a new version

Version 16.2 and above

To upgrade the HDMI project to Vivado 16.2, one of the blocks has been updated. This block is the Test Pattern Generator, which doesn't have the AXI clock and reset pins now.

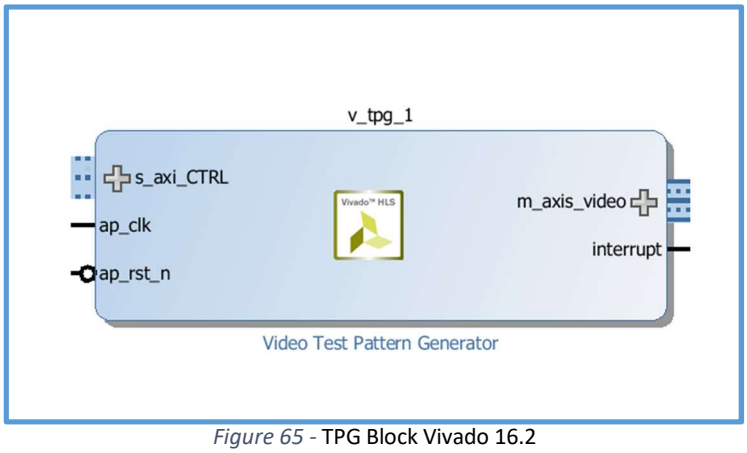


Figure 65 - TPG Block Vivado 16.2

This can be a difference in the design previously described, because the AXI clock and the main clock for the TPG to make data transfers are different.

To validate the design, connect the ap_clk to the 75MHz clock, and consequently, modify the AXI clock and AXI reset connections related at the Interconnect which is master of the TPG, assigning as well the corresponding Process System Reset.

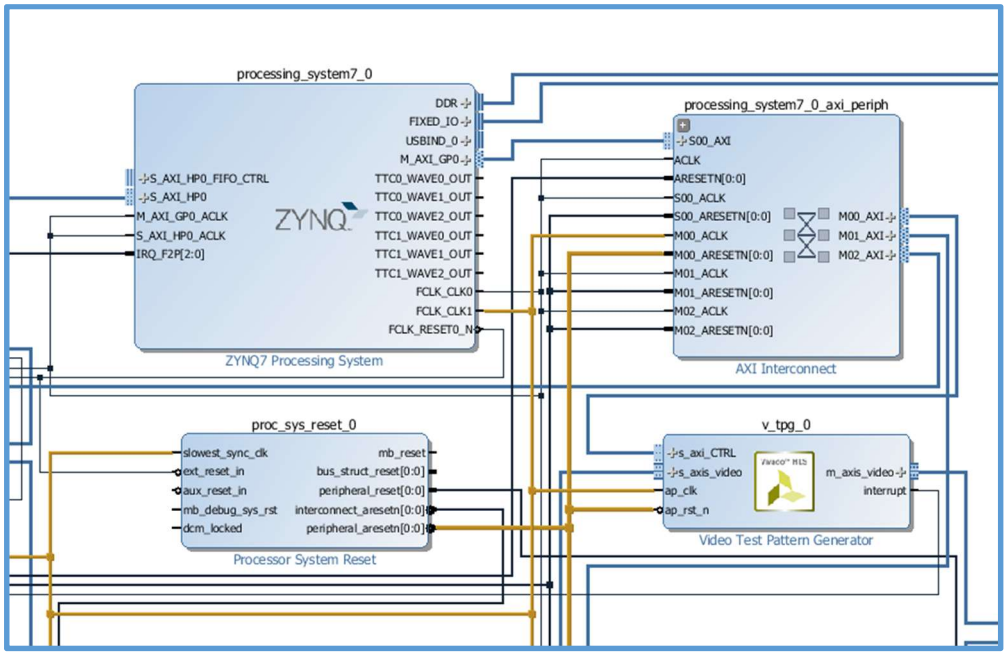


Figure 66 - Correct connections

This change should be enough for the design to be built properly. Remember to export the hardware again, and re-create a platform, FSBL, and HDMI applications related to the new design.

3.6.2 To a different module

Z7030

To target this project for a z7030 module, go to project settings, and change the FPGA part:

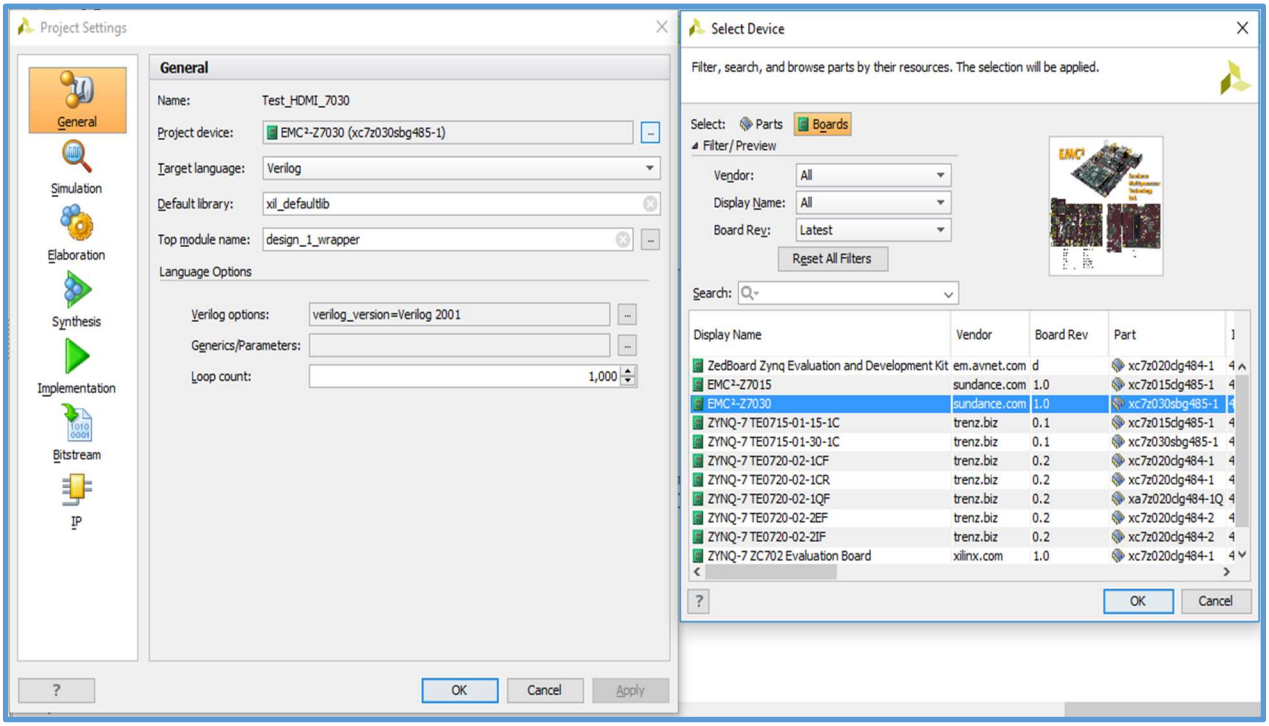


Figure 67 - Target z7030

This change requires to rebuild the project, generating a new bitstream, and therefore modifying the hardware, which forces us to re-create a platform, FSBL and HDMI applications according to the new FPGA architecture.

The main difference between the z7015 and z7030, apart from the huge difference of speed and resources available, is the IO banks in the FPGA. The z7015 has HR (High Range) banks, while the z7030 has both HR and HP (High Performance) banks. This affects to some of the pins available in the z7030 module.

At the Board IO document this aspect is mentioned due to the problems it can cause the use of the z7030 module with some IO standards, as LVDS 25 for differential signals. The fact that some of the pins belong to HP banks in the FPGA, can make trouble with Vivado depending on the IO standards selected.

For this reason, in this HDMI application, for the z7030, the user must set the jumpers onboard to 1.8V.

To use the z7030, it's highly recommended to use a fan to cool down the module, as

the temperature can increase up to 100 degrees or more otherwise, and that can cause timing problems within the FPGA, or even damage the hardware.

TE0820
To target you project for the TE0820 instead of the z7015 or the z7030, you need to remove the Zynq IP block in Vivado and replace it with a Zynq UltraSCALE+ IP block.

3.7 How can I create an SDSoC Platform for EMC²?

3.7.1 Up to version 2016.2 included

The platform created in this tutorial is based on the HDMI test project previously explained. It has been made for SDSoC 16.2, using the EMC² with a z7030 module.

The steps explained in order to create this platform can be used for any other platforms, changing the paths and names specified. For more detailed information about platforms and libraries, check this document out.

This platform has been made for a standalone application. FreeRTOS and Linux will be added in the future.

To understand the mess between files, paths and copy-paste operations, it's recommended to create a folder where the user saves their platforms and workspaces. For this tutorial, a folder called "SDSoC_Tutorial" has been created at the C disk. Remember that for any Xilinx's software, it's recommended not to use spaces in the names or long paths for our files.

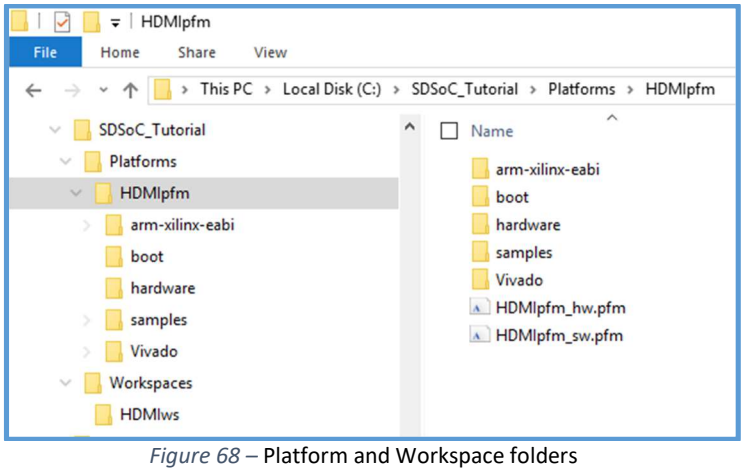


Figure 68 – Platform and Workspace folders

As it's shown above, in "SDSoC_Tutorial the user will include their platforms and workspaces. In this case:

- "HDMIws", workspace where the user will generate their SDSoC projects later.
- "HDMIpfm", in which the user must create 5 folders and 2 files:
  - "**arm-xilinx-eabi**". This folder will contain the libraries and a linker script.
  - "**boot**". This folder will contain the fsbl.elf file and a .bif file which specifies the parameters for a standalone .bin file generation.
  - "**hardware**". This folder will contain the prebuilt files, in order to buy time when rebuilding a project in SDSoC without functions pushed into hardware, avoiding the bitstream generation, which takes time.
  - "**samples**". This folder contains the C/C++ files for the different

applications the user wants to add automatically to the platform
    - "**Vivado**". This folder will contain the design made in Vivado, base of the platform.
    - "**HDMIpfm_sw.pfm**" (*platformname*_sw.pfm). This Platform software description file will contain the necessary arguments for the software to find and compile the libraries, booting or hardware files.
    - "**HDMIpfm_hw.pfm**" (*platformname*_hw.pfm). This Platform hardware description file will contain all the IP definitions of the design. It's generated by Vivado.
- **PUT THE HARDWARE DESIGN INTO THE PLATFORM (VIVADO FOLDER)**

With the design open in Vivado, the bitstream written, and hardware exported, click "File -> Archive project".
(The design in Vivado should have the same name as the platform. To change the name of the design, File -> Save as -> choose new name and path).

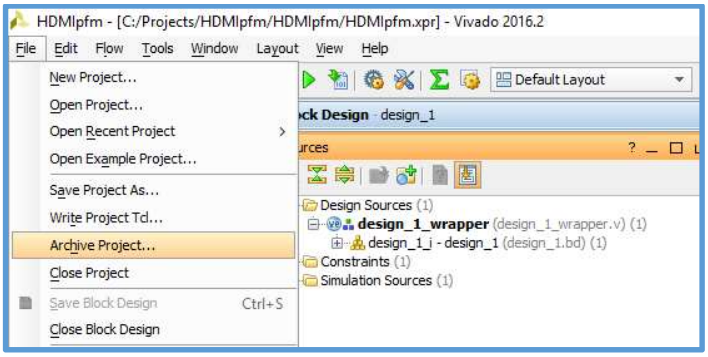


Figure 69 – Archive Project

And click "OK"

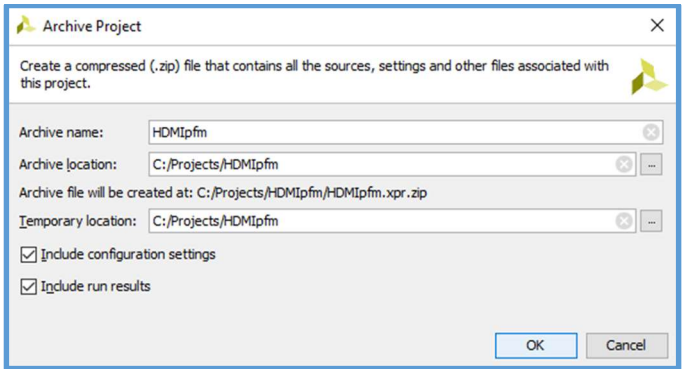


Figure 70 – Archive project. Select path

Extract "HDMIpfm.xpr.zip" and put its contents in the folder "**Vivado**" at "HDMI_Tutorial_platform". Put in this folder the.rgba picture as well ("EMC2_with_logo.rgba")

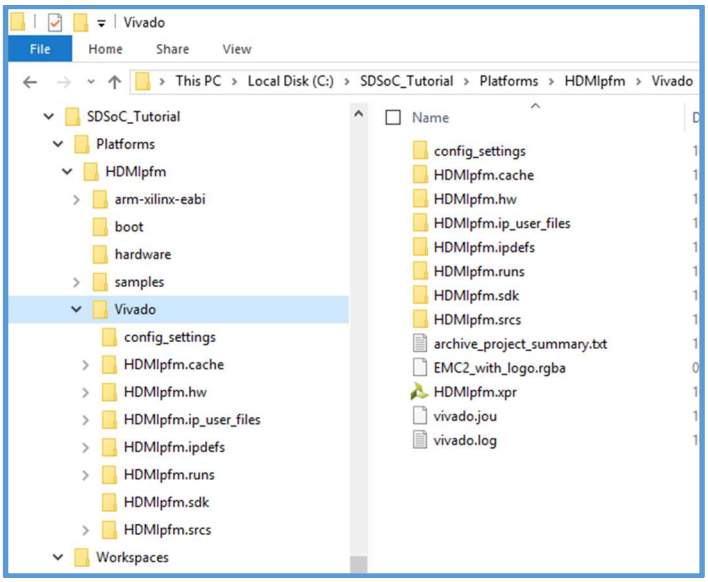


Figure 71 - Move the hardware design

- **CREATE HW PLATFORM, BSP, FSBL AND HELLOWORLD (ARM-XILINX-EABI AND BOOT FOLDERS)**

Open SDSoC 16.2, and select the .sdk folder of the HDMI Vivado project as workspace.

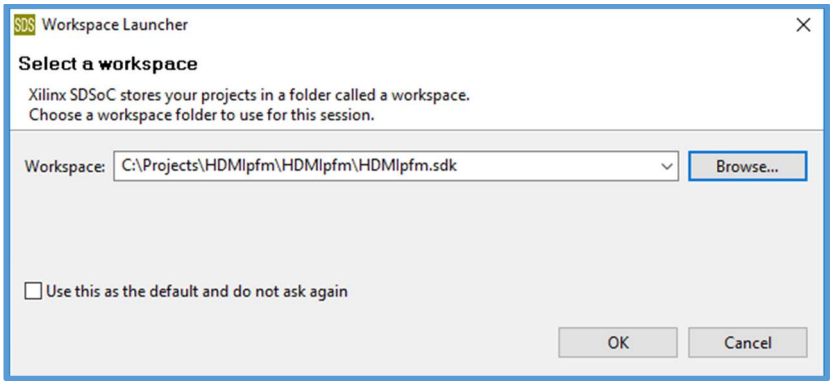


Figure 72 – Select workspace

Create a new Hardware Platform Specification project, and select the exported hardware .hdf file located in .sdk

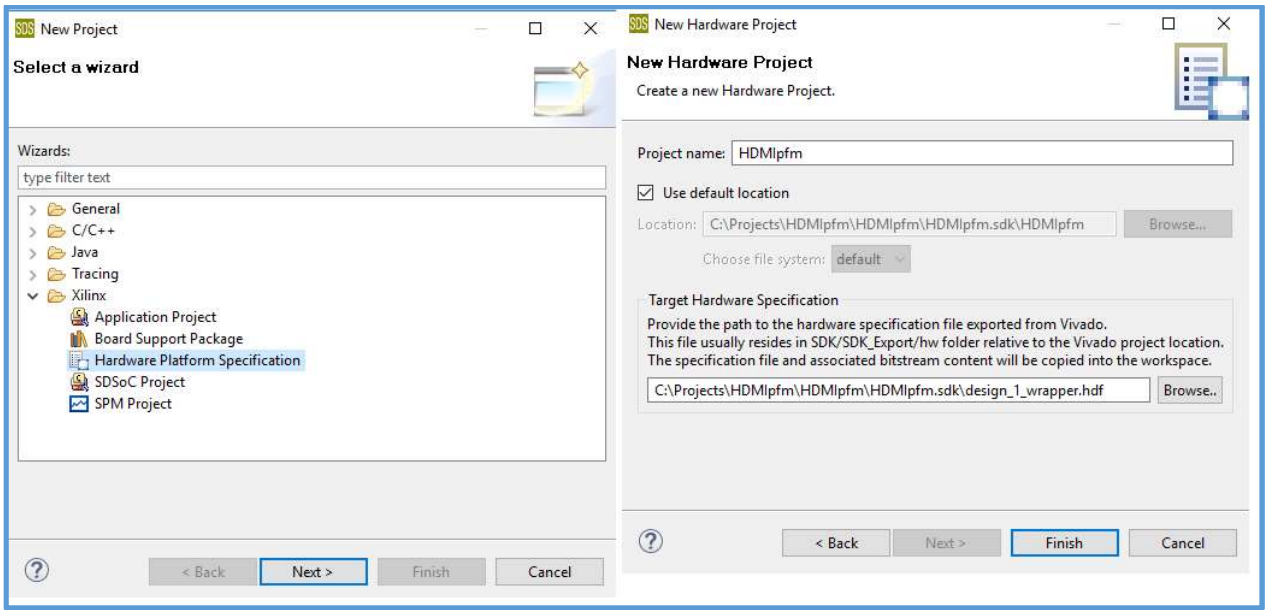


Figure 73 – New HW Platform Specification project

Create a new Board Support Package, using the previous Hardware Platform Specification project.

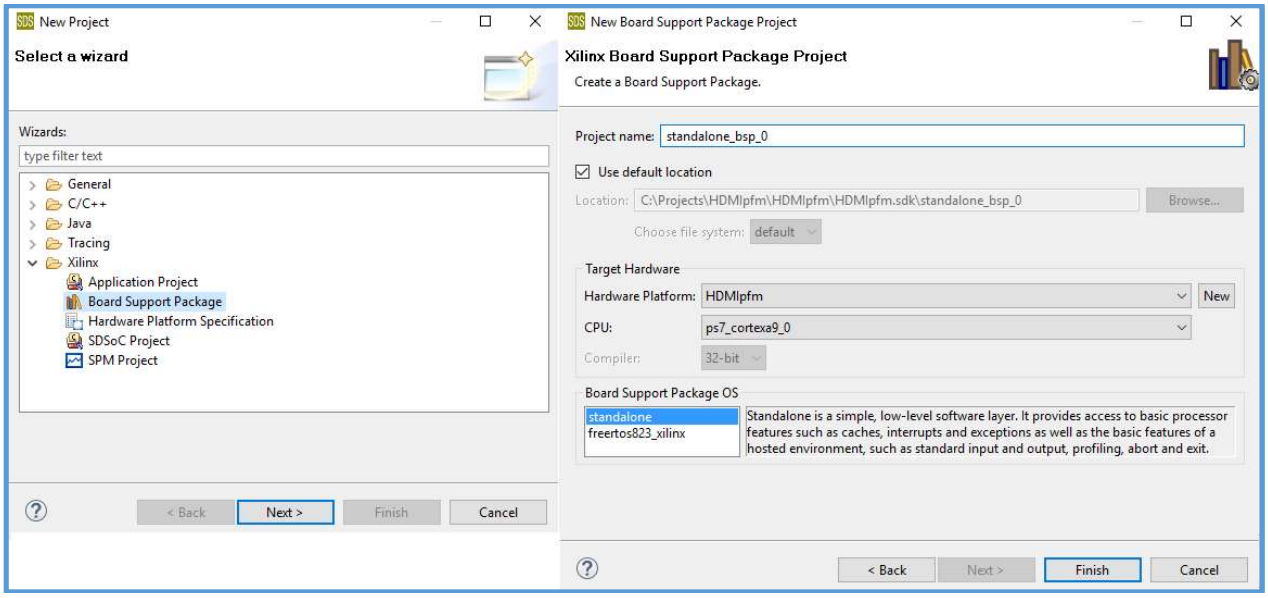


Figure 74 – BSP Project

Select the library “xilffs”, accept, and then build the BSP project.

Create a FSBL project, using the same Hardware Platform and BSP made beforehand.

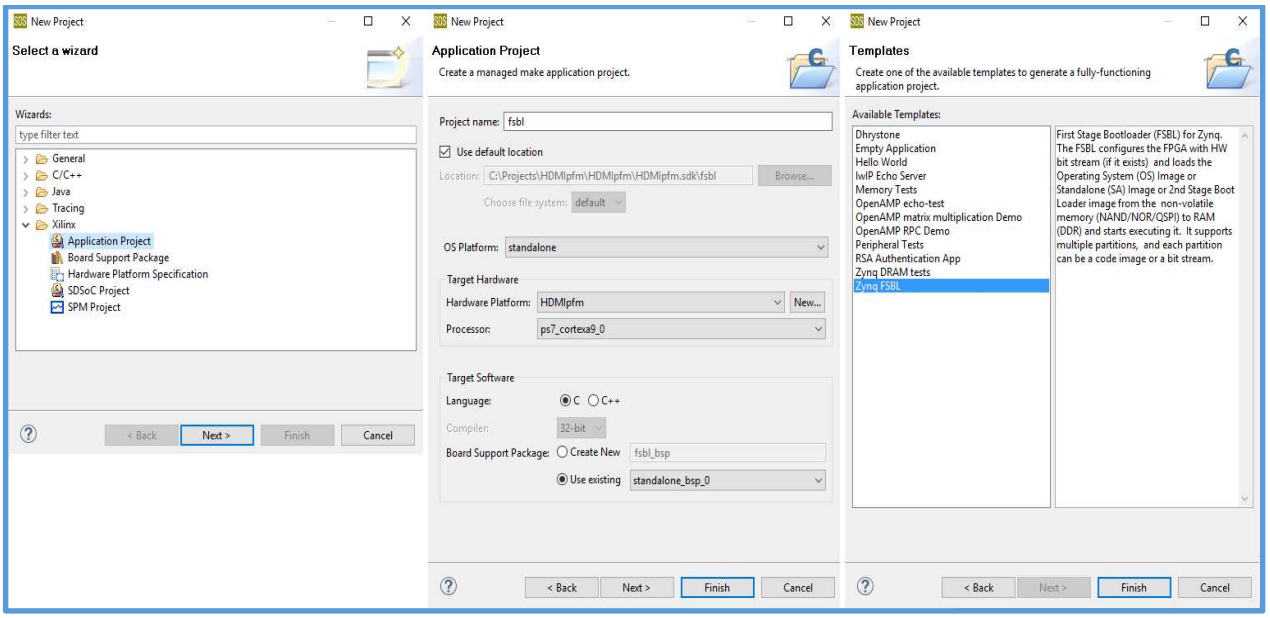


Figure 75 – FSBL Project

Build the FSBL project, and create a Hello World project, following the same steps.

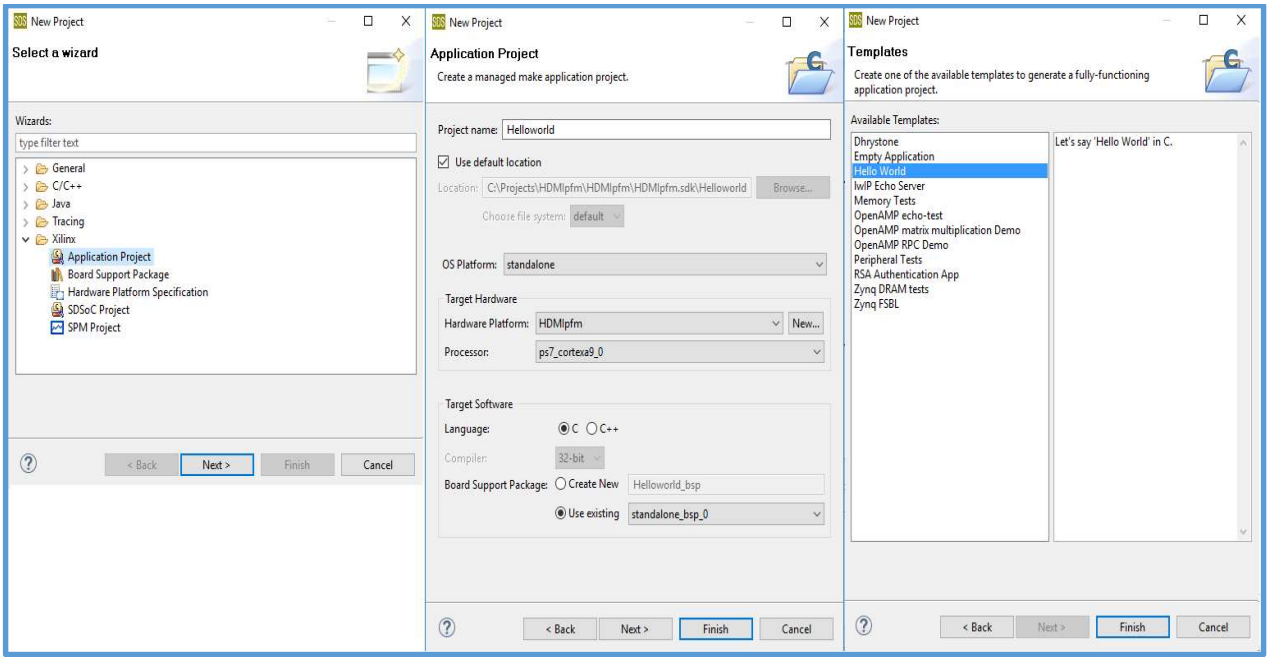


Figure 76 – Hello World Project

Now that all the projects are created and built, go to the .sdk folder and:

- Copy the folders “lib” and “include” from the “standalone_bsp_0” project into “**arm-xilinx-eabi**” at “HDMI_Tutorial_platform”.
- Move “lscript.ld” from the “Helloworld” project into “**arm-xilinx-eabi**” at “HDMI_Tutorial_platform”.
- Move “fsbl.elf” from the “fsbl” project into “**boot**” at “HDMI_Tutorial_platform”.
- Create a file called “generic.readme” in “**boot**”, and write this code in it:

```
-= SD card boot image =-

Platform: <platform>
Application: <elf>

1. Copy the contents of this directory to an SD card.
2. Insert SD card and turn board on.
3. If you are booting Linux, login with username "root" and password
"root".
```

- Create a file called “standalone.bif” in “**boot**”, and write this code in it:

```
/* standalone */
the_ROM_image:
{
[bootloader]<boot/fsbl.elf>
<bitstream>
<elf>
[load = 0x38000000]<Vivado/EMC2_with_logo.rgba>
}
```

As it’s shown, the .rgba picture will be merged with the fsbl.elf file, and the application into the BOOT.bin file for the SD card.

- **CREATE THE PLATFORM HARDWARE AND SOFTWARE DESCRIPTION FILES**

The platform hardware description file is generated by Vivado, the platform software description file must be written by the user.

To generate the hardware description file, in the TCL console in Vivado, write the following commands:

```
source -notrace C:/Xilinx/SDSoC/2016.2/scripts/vivado/sdsoc_pfm.tcl
set pfm [sdsoc::create_pfm HDMIpfm_hw.pfm]
sdsoc::pfm_name $pfm "sundance.com" "xd" "HDMIpfm" "1.0"
sdsoc::pfm_description $pfm "HDMI test using EMC2 z7030"
sdsoc::pfm_clock $pfm FCLK_CLK0 processing_system7_0 0 true rst_processing_system7_0_100M
sdsoc::pfm_clock $pfm FCLK_CLK1 processing_system7_0 1 false proc_sys_reset_0
sdsoc::pfm_axi_port $pfm M_AXI_GP0 processing_system7_0 M_AXI_GP
sdsoc::pfm_axi_port $pfm M_AXI_GP1 processing_system7_0 M_AXI_GP
sdsoc::pfm_axi_port $pfm S_AXI_ACP processing_system7_0 S_AXI_ACP
sdsoc::pfm_axi_port $pfm S_AXI_HP0 processing_system7_0 S_AXI_HP
sdsoc::pfm_axi_port $pfm S_AXI_HP1 processing_system7_0 S_AXI_HP
sdsoc::pfm_axi_port $pfm S_AXI_HP2 processing_system7_0 S_AXI_HP
sdsoc::pfm_axi_port $pfm S_AXI_HP3 processing_system7_0 S_AXI_HP
for {set i 0} {$i < 16} {incr i} {
  sdsoc::pfm_irq $pfm In$i xlconcat_0
}
sdsoc::generate_hw_pfm $pfm
```

Change the SDSoC installation path if necessary, as well as the name of the platform or the platform hardware description file’s name in case is different.
Type “pwd” in the TCL console to see where the file has been generated. Copy it into the platform folder (“HDMIpfm”).

Create the “HDMIpfm_sw.pfm” file if you didn’t before, and write in it:

```
<?xml version="1.0" encoding="UTF-8"?>
<xd:repository xd:vendor="sundance.com" xd:library="xd" xd:name="HDMIpfm " xd:version="1.0"
xmlns:xd="http://www.xilinx.com/xd" >

    <xd:description>EMC2 HDMI Test Platform</xd:description>

    <xd:libraryFiles
```

```
      xd:os="standalone"
      xd:includeDir="arm-xilinx-eabi/include"
      xd:libDir="arm-xilinx-eabi/lib"
      xd:ldscript="arm-xilinx-eabi/lscript.ld"
    />

    <xd:bootFiles
      xd:os="standalone"
      xd:bif="boot/standalone.bif"
      xd:readme="boot/generic.readme"
    />

</xd:repository>
```

- **SAMPLES AND PREBUILT HARDWARE**

Create a folder called “test_hdmi” and a file called “template.xml” in “**Samples**”.

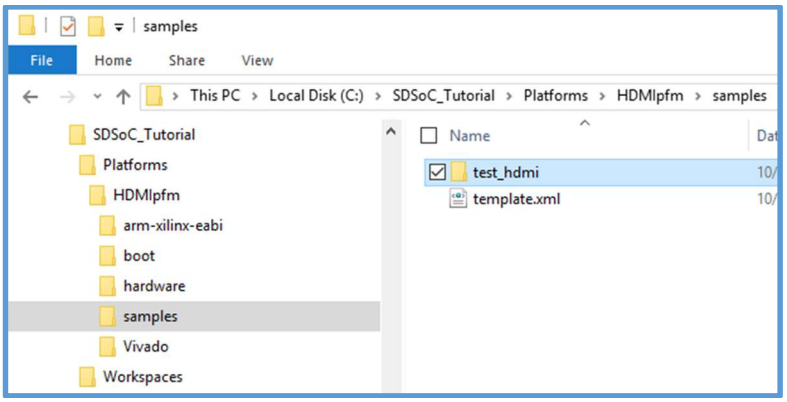


Figure 77 – Samples folder

Copy the samples in the folder “test_hdmi”, and write this code into “template.xml”:

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest:Manifest xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
                   xmlns:manifest="http://www.xilinx.com/manifest">
    <template location="test_hdmi" name="Test HDMI"
              description="Test the HDMI output">
        <supports>
            <and>
                <or>
                    <os name="Standalone"/>
                </or>
            </and>
        </supports>
    </template>
</manifest:Manifest>
```

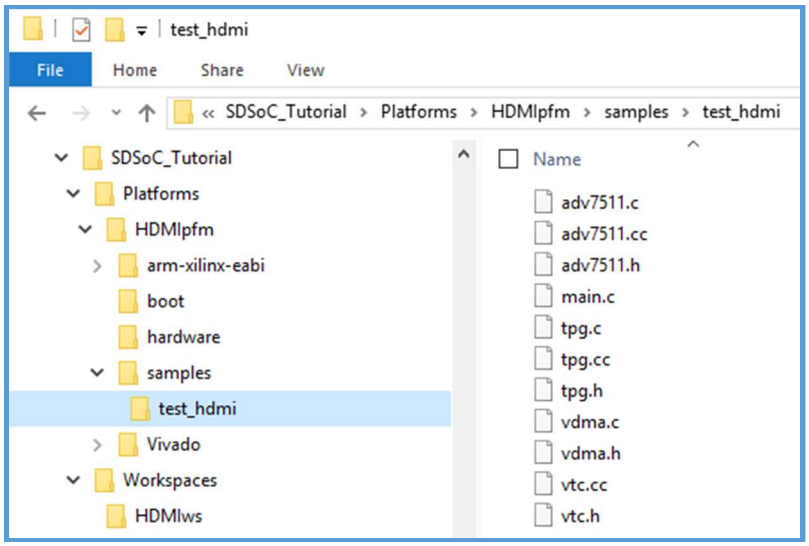


Figure 78 – Copy all the samples into the folder

The last step is to create the pre-built hardware files. To do it, open SDSoC 16.2. Select the folder "HDMIws" created at the beginning of this tutorial as workspace.

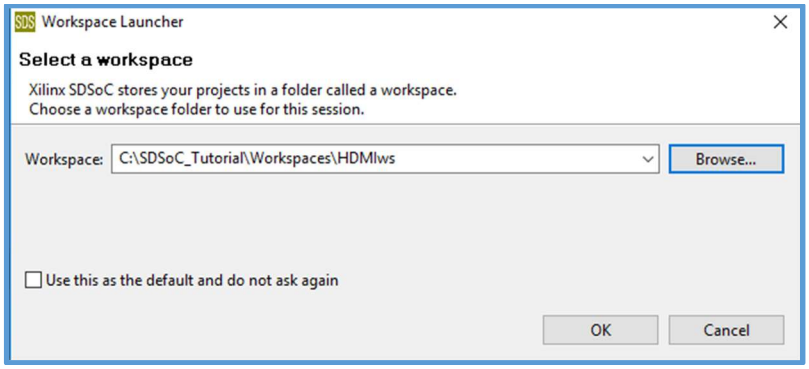


Figure 79 – Select workspace

File -> New -> SDSoC Project. Put any name ("HDMI_Test" in this example), and select "Other" Platform, choosing the path where the platform has been created.
Choose "standalone".
Next -> Test HDMI (Selecting the samples) -> Finish

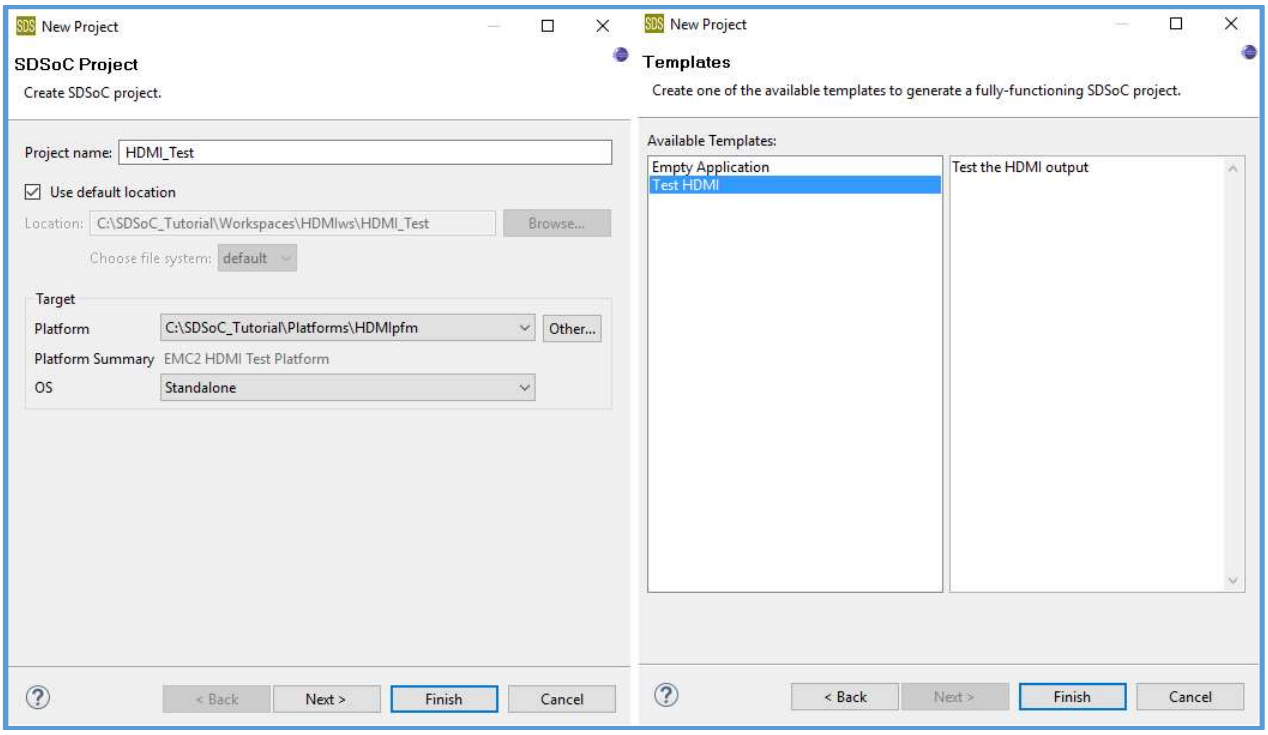


Figure 80 – Choose platform, OS and Template

Select SDRelease as the active build configuration. It can be done:

- Right clic on "HDMI_Test"-> Build configurations -> Set active -> SDRelease
- At the project overview -> Build configurations window -> Active configuration

Then, build the project. It will take time, as SDSoC calls Vivado to write the bitstream. (It took 18min in my case).

As soon as the project is built, create the folder "**hardware**" in "HDMIpfm" if you didn't before, and create a folder called "prebuilt" inside.

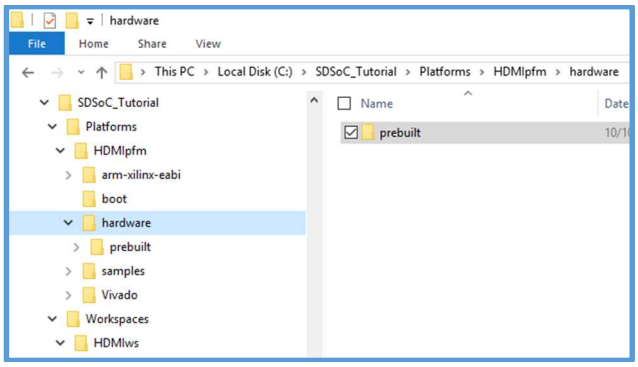


Figure 81 – Create a folder called "prebuilt"

Then, it's necessary to move files into this folder to have a prebuit hardware available in the SDSoC platform.

- In this path:
  C:\SDSoC_Tutorial\Workspaces\HDMIws\HDMI_Test\SDRelease_sds\p0\ipi\HDMIpfm.runs\impl_1
  Copy "bitstream.bit" into "prebuilt".

- Create a folder called “export” in “prebuilt”. In this path:
  C:\SDSoC_Tutorial\Workspaces\HDMIws\HDMI_Test\SDRelease_sds\p0\ipi\HDMIpfm.sdk
  Copy “HDMIpfm.hdf” into “export”. (*platformname*.hdf)

- Create a folder called “hwcf” in “prebuilt”. In this path:
  C:\SDSoC_Tutorial\Workspaces\HDMIws\HDMI_Test\SDRelease_sds\.llvm
  Copy “partitions.xml”, “apsys_0.xml” into “hwcf”.

- Create a folder called “swcf” in “prebuilt”. In this path:
  C:\SDSoC_Tutorial\Workspaces\HDMIws\HDMI_Test\SDRelease_sds\swstubs
  Copy “devreg.c”, “devreg.h”, “portinfo.c”, “portinfo.h” into “swcf”.

To finish, modify the file “HDMIpfm_sw.pfm”, adding this hardware configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<xd:repository xd:vendor="sundance.com" xd:library="xd" xd:name="HDMIpfm " xd:version="1.0"
xmlns:xd="http://www.xilinx.com/xd" >

    <xd:description>EMC2 HDMI Test Platform</xd:description>

    <xd:libraryFiles
      xd:os="standalone"
      xd:includeDir="arm-xilinx-eabi/include"
      xd:libDir="arm-xilinx-eabi/lib"
      xd:ldscript="arm-xilinx-eabi/lscript.ld"
    />

    <xd:bootFiles
      xd:os="standalone"
      xd:bif="boot/standalone.bif"
      xd:readme="boot/generic.readme"
    />

    <xd:hardware
      xd:system="prebuilt"
      xd:bitstream="hardware/prebuilt/bitstream.bit"
      xd:export="hardware/prebuilt/export"
      xd:swcf="hardware/prebuilt/swcf"
      xd:hwcf="hardware/prebuilt/hwcf"
    />

</xd:repository>
```

The folder should look like this at the end:

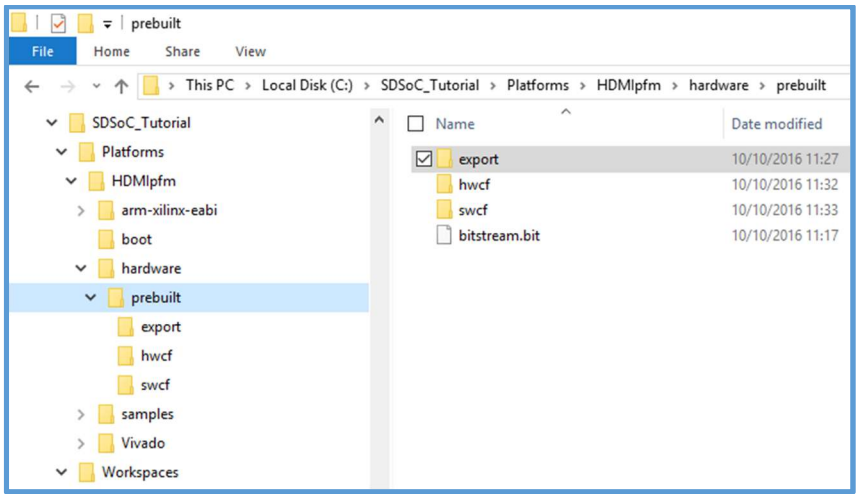


Figure 82 – Prebuilt folder

Close SDSoC. Delete all the contents in the folder “HDMIws” (the workspace). Open SDSoC, selecting the same workspace, and create a new “HDMI_Test” SDSoC project, selecting the corresponding platform, OS and template (Figure 80).

Select SDRelease as build configuration, and build the project. This time, SDSoC will use the prebuilt hardware, and the user will see the building time reduced.
(It took 41 seconds in my case).

Now there should be a folder called “sd_card”, where the user can take the BOOT.bin file to copy it into a SD card, and test the project.

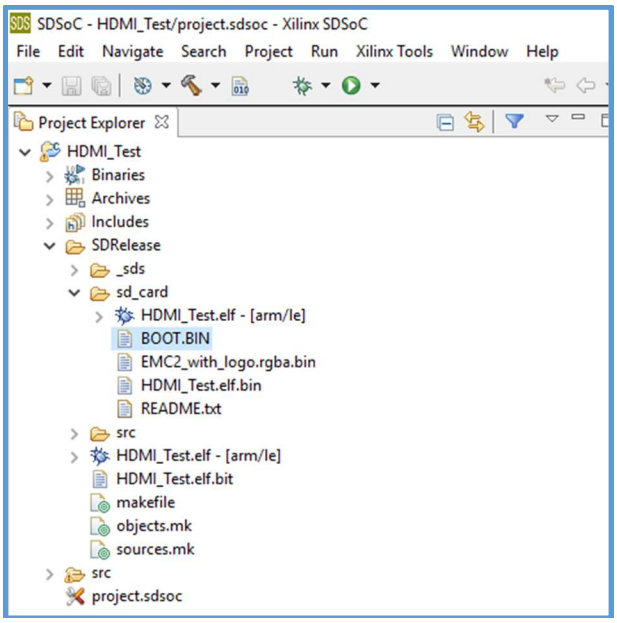


Figure 83 – BOOT.bin file

3.7.2 From version 2016.3 and above

The platform created in this tutorial is based on the HDMI in to HDMI out project It has been made for SDSoC 17.1, using the EMC² with a TE0820 module.

The steps explained in order to create this platform can be used for any other platforms, changing the paths and names specified. For more detailed information about platforms and libraries, check this document out.

This platform has been made for a standalone application. FreeRTOS and Linux will be added in the future.

Prepare the hardware platform

In a temporary folder create your vivado project under *platform_name->hw->vivado*

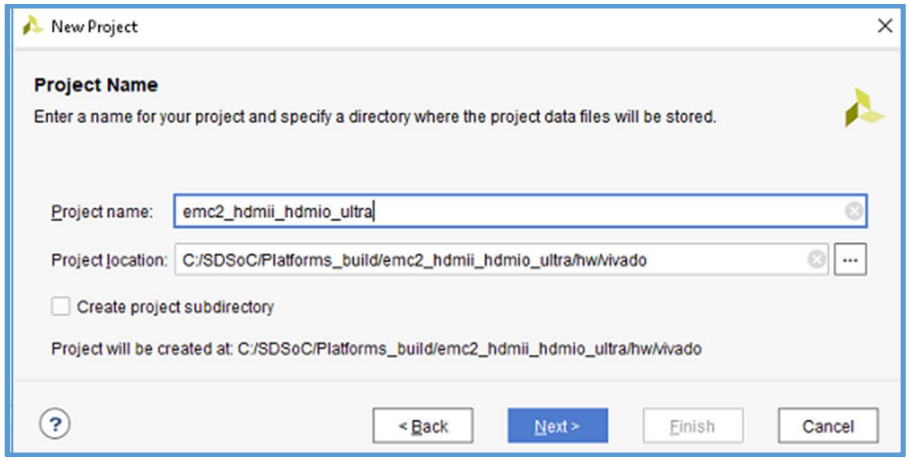


Figure 84 – Create a new Vivado project

Once the Vivado design is build and the bitstream has been generated successfully, prepare the tcl file needed for the platform hardware description file.

The tcl file should look like that:

```
source -notrace c:/Xilinx/SDx/2017.2/scripts/vivado/sdsoc_pfm.tcl
set pfm [sdsoc::create_pfm emc2_hdmii_hdmio_ultra.hpfm]
sdsoc::pfm_name $pfm "sundance.com" "xd" "emc2_hdmii_hdmio_ultra" "1.0"
sdsoc::pfm_description $pfm "emc2 hdmii to hdmio for the TE0820"
sdsoc::pfm_clock $pfm pl_clk0 zynq_ultra_ps_e_0 0 true proc_sys_reset
sdsoc::pfm_clock $pfm pl_clk1 zynq_ultra_ps_e_0 1 false rst_processing_system7_0_142M
sdsoc::pfm_axi_port $pfm M_AXI_HPM1_FPD zynq_ultra_ps_e_0 M_AXI_GP
sdsoc::pfm_axi_port $pfm M_AXI_HPM0_LPD zynq_ultra_ps_e_0 M_AXI_GP
sdsoc::pfm_axi_port $pfm S_AXI_HPC0_FPD zynq_ultra_ps_e_0 S_AXI_HPC
sdsoc::pfm_axi_port $pfm S_AXI_HPC1_FPD zynq_ultra_ps_e_0 S_AXI_HPC
sdsoc::pfm_axi_port $pfm S_AXI_HP1_FPD zynq_ultra_ps_e_0 S_AXI_HP
sdsoc::pfm_axi_port $pfm S_AXI_HP2_FPD zynq_ultra_ps_e_0 S_AXI_HP
sdsoc::pfm_axi_port $pfm S_AXI_HP3_FPD zynq_ultra_ps_e_0 S_AXI_HP
for {set i 5} {$i < 8} {incr i} {
sdsoc::pfm_irq $pfm In$i xlconcat_0
}
sdsoc::generate_hw_pfm $pfm
```

File->Export->Export Hardware…

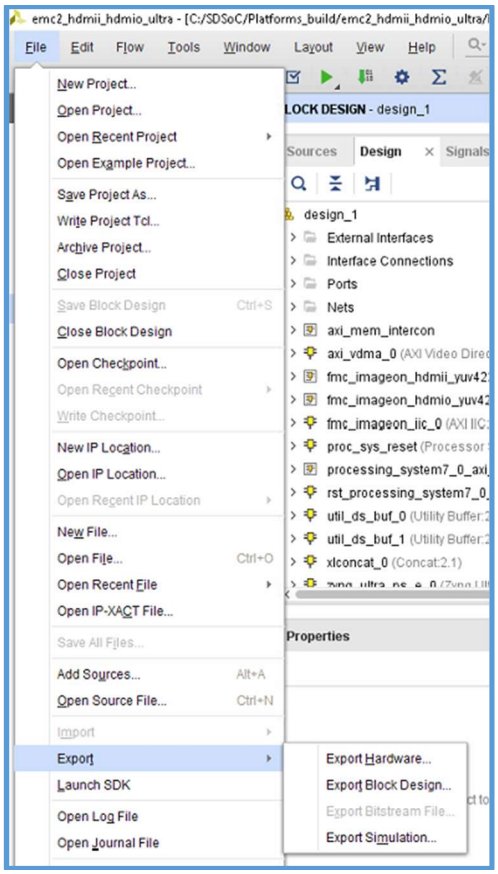


Figure 85 – Export hardware

Prepare the software platform

Then File->Launch SDK

The Xilinx SDx environment should open and the design hardware platform should be automatically generated.

Go to File->new->Application Project and create a HelloWorld project.

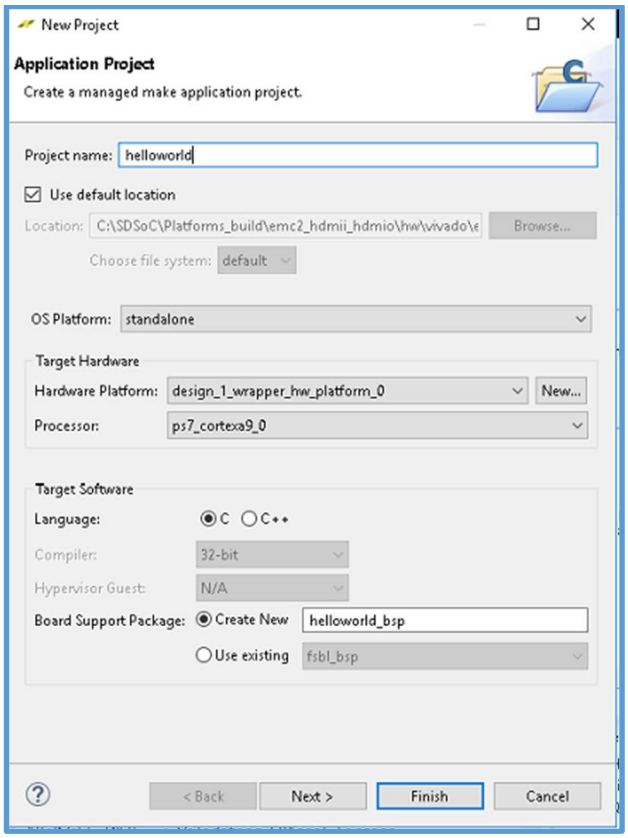


Figure 86 – Create a new SDSoC project

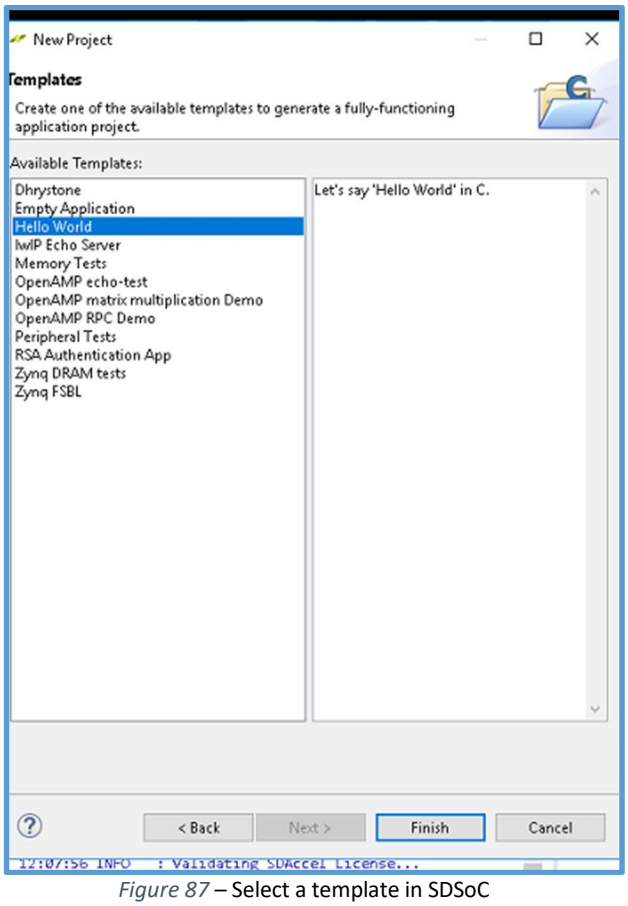


Figure 87 – Select a template in SDSoC

Then compile your project.

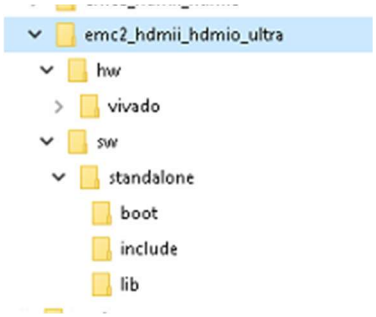
Now create and compile a Zynq FSBL project the same way as above.

In the temporary folder, under the folder *platform_name* create the *folders sw->standalone->boot*

Then copy the folders
*Platform_name->hw->vivado->platform_name.sdk->hello_bsp->psu_cortexa53_0->**include***
*Platform_name->hw->vivado->platform_name.sdk->hello_bsp->psu_cortexa53_0->**lib***

To the folder
Platform_name->sw->standalone

You should have the following folder hierarchy:



In the folder *standalone* copy the file
*Platform_name->hw->vivado->platform_name.sdk->hello->src->**lscript.ld***

In the folder *boot* copy:

- fsbl.elf: it should be in the folder *Platform_name->hw->vivado->platform_name.sdk->fsbl->Debug*
- generic.readme: a readme file
- standalone.bif:

```
/* standalone */
the_ROM_image:
{
  [bootloader]<standalone/boot/fsbl.elf>
  <bitstream>
  <elf>
  [load = 0x38000000]<standalone/boot/sundance_logo.yuv>
}
```

Now you can use the GUI to generate automatically your platform.
Open the *SDx Terminal 2017.1* and type *sdspfm -gui* to start the SDSoC platform Utility.

Fill each field and press the button *Generate*.

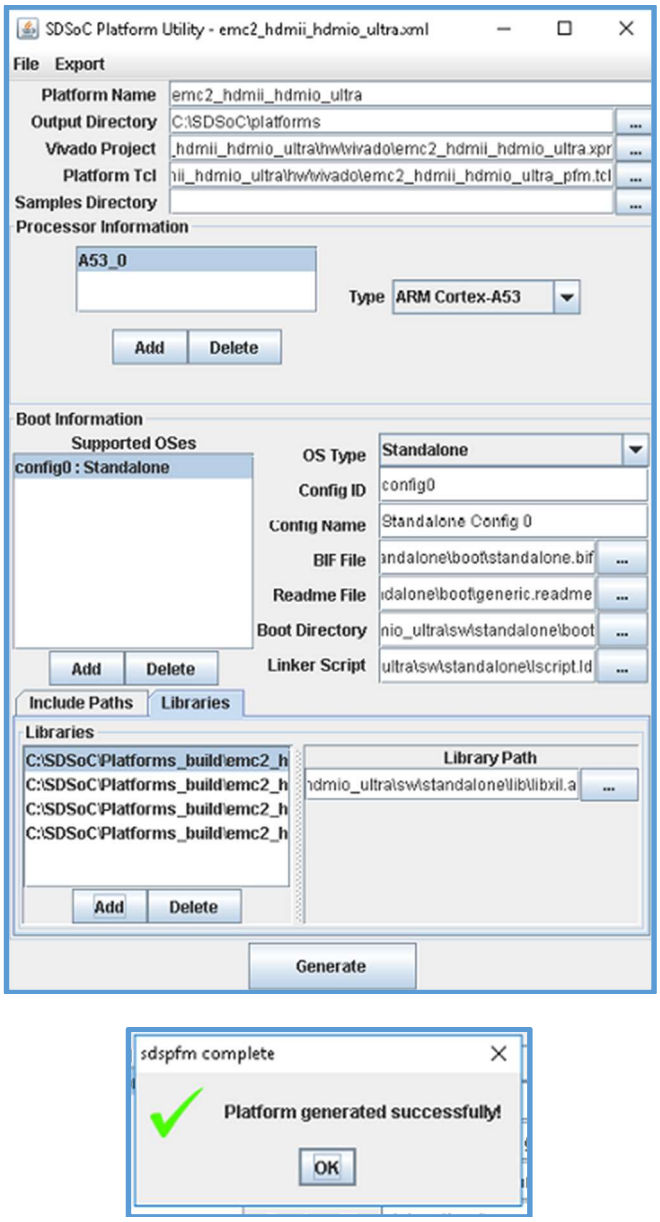


Figure 88 – SDSoC platform Utility

4 Demos

4.1 How to run the application directly from built files?

The EMC2-DP can boot from SD card so the application code would run at power up.

To do so copy the boot.bin file provided by Sundance to a SD card.
The SD card needs to be formatted in FAT32 and the partition made bootable. The SD card should be at least 4Gb.

Make sure the EMC2 is configured for SD boot as explained in section 2.4.

The boot.bin file on the SD card contains the bitstream to configure the FPGA, the FSBL and the software application.

WARNING: If the application targets the TE7030 or the TE0820 module, the J7 and J8

should be set to select a 1.8V voltage (2.2).

4.2 How to build the application in SDSoC?

In SDSoC, go to File->New->Xilinx SDx Project …

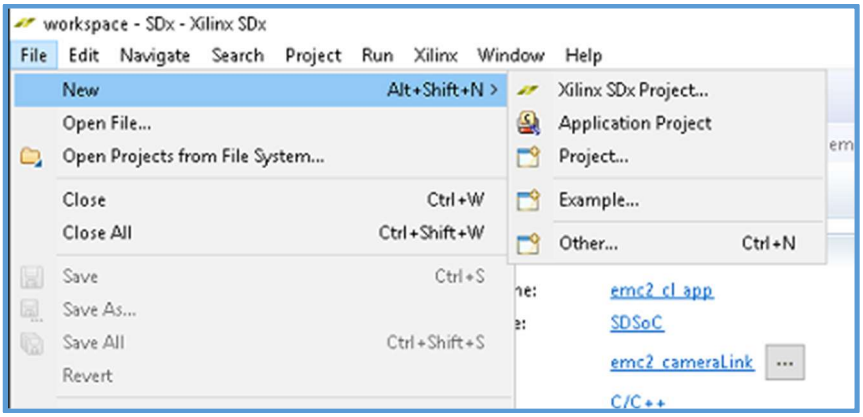


Figure 89 – Create a new SDSoC project

Enter your project name

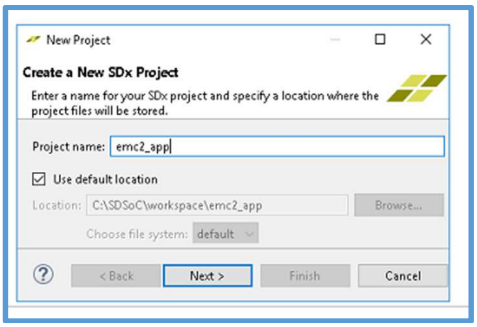


Figure 90 – Create a new SDSoC project (2)

Select the appropriate platform (*emc2_hdmii_hdmio*, *HDMIpfm*, *emc2_cl_plat*).

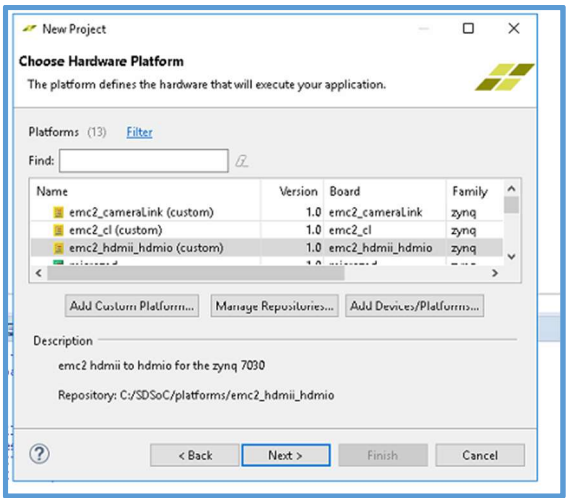


Figure 91 – Select the hardware platform

Then in the window *Template*, select the application you want to build for example *HDMI in to HDMI out Demo* and press *Finish*.

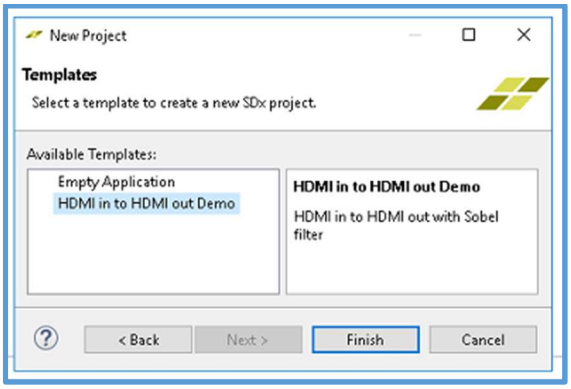


Figure 92 – Select a template in SDSoC

Right click on your project (*emc2_app*) and select *Build Project*

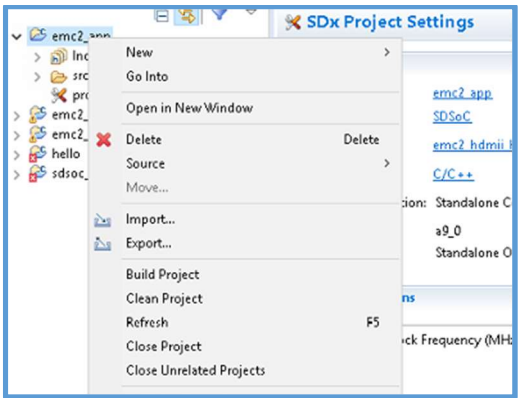


Figure 93 – Build a project in SDSoC

Once your project is built, copy the files from the folder *<your_project>/Release/sd_card* to your SD card. Insert your SD card in you EMC2 and reset it. The processed input video will be displayed on the output HDMI.

4.3 How to accelerate the software into hardware?

Xilinx SDSoC environment enables identified functions to be compiled to HW accelerator blocks together with DMA data movers and merged with the user defined platform in the programmable logic (PL) part of the Zynq device. This helps to offload critical parts of computation to the PL part of the device.

The edge detection code shows an example of the advantages of such hardware acceleration. Its code is written in simple C language but it is selected in SDSoC project options for hardware acceleration.

How to select a function to accelerate in hardware:
In SDSoC, go to *Project Explorer*, under your project name, select the folder *src* then go to the file where the function to accelerate is. Right click the function (for example, sobel_filter_acc), and select *Toggle HW/SW*. The function to be accelerated will appear in the project file (*projects.sdx*) under the *Hardware Functions* section.

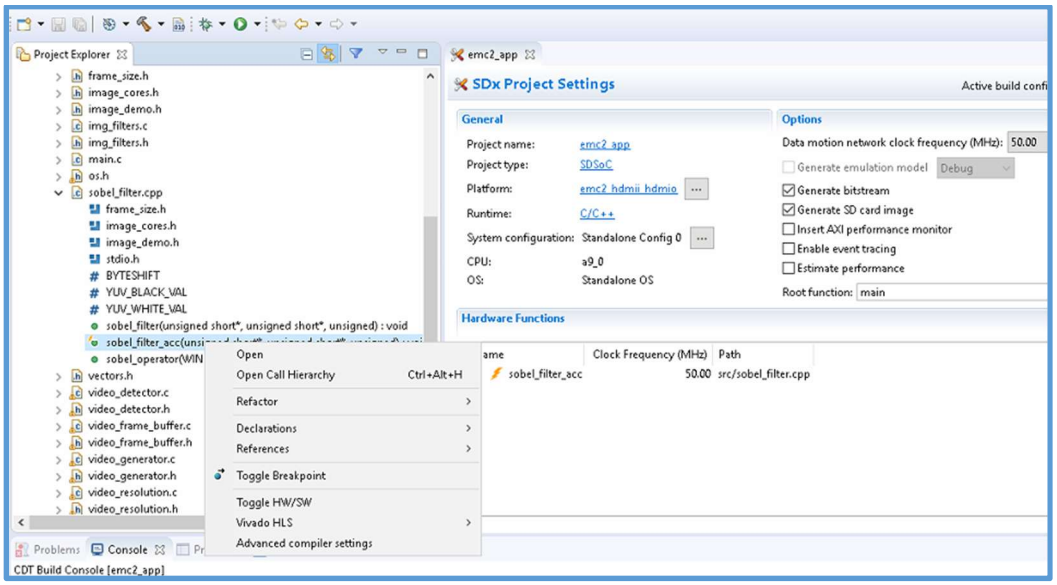


Figure 94 – Select function for hardware acceleration

While building the project, this function is processed by High Level Synthesis tools to be mapped to hardware. This gives far better performance than software only implementation.

4.4 Description

4.4.1 HDMI output application

The demonstration runs on a stand-alone EMC² Development Platform. The purpose of this demo is to use the HDMI feature on the SEIC extension board.
At power up, the FSBL loads an image from the boot.bin partition to a defined address in the DDR. The FSBL also configures the IPs at boot time.
The design then reads this image and send it to the on-board HDMI output where it is displayed as a splash screen on the monitor.
For more details go to 3.5

4.4.2 HDMI input to HDMI output application

The demonstration runs on a stand-alone EMC² Development Platform PCIe/104 OneBank™ featuring a Zynq 7030 SoC with dual ARM9 CPU and a re-configurable FPGA Logic.
The dual ARM processor cores have direct access to the DDR3 memory that provides 1GByte of storage.

The purpose of this demo is to allow real-life data, in this case a video-stream from a HDMI Output device to be loaded into the Zynq's DDR memory, processed and then displayed again on a second HDMI-Input device (typically a monitor).
In this demonstration, a VITA57.1 FMC® compatible Daughter Card is plugged to the EMC²-DP to provide HDMI input/output capabilities. Also, a Sobel filter is implemented to process the video data coming from an HDMI camera.
This demo shows how most computing intensive tasks are implemented, where possible, in hardware, both using standard or custom IP cores and by using High Level

Synthesis tools.

Hardware

The demo is based on a costumed designed platform developed in Vivado 2017.1.
The resolution of the input and output video data is set at 1280x720 with a 65MHz frequency rate.

For the video input and video output we used respectively HDMI input IP and HDMI output IP that already existed. The HDMI input component (ADV7611) and the HDMI output component (ADV7511) are configured by software at start up.
A UART interface is used to configure different part of the system (HDMI, VDMA, VTC).
The Xilinx® LogiCORE™ IP AXI VDMA core provides the high-bandwidth direct memory access between the DDR memory and the HDMI peripherals.
This is a simplified view of the hardware design using Vivado 2017.1

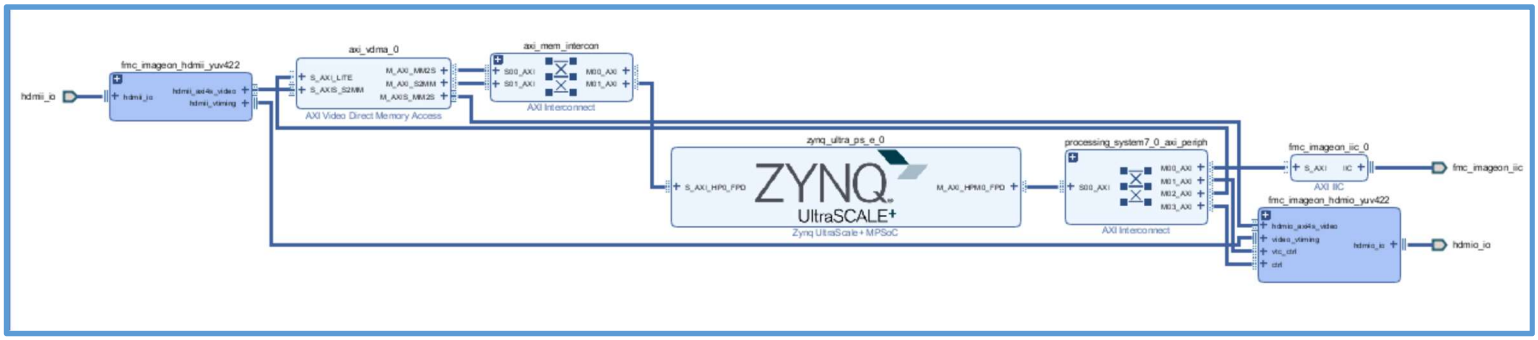


Figure 95 – Demo HDMI in to HDMI out Vivado project

Software

Our demo is a bare-metal application executed on the ARM Cortex A9 processor. The project was developed with SDSoC2017.1 from Xilinx.

In our software the edge detection code is accelerated in hardware to improve performances.
The edge detection algorithm is producing a black and white video stream. Edges in each frame are marked as white and the remaining part of the figure is set as black. The edges are detected by a Sobel filter. Each pixel is filtered by a 3x3 2D FIR filter. A nonlinear decision on the output of the filter provides information as to if the pixel is part of an edge or not. All computation is performed in fixed point. The Sobel filter is applied on the left half of each video frame for display purposes.

When the filter is accelerated, the main purpose of the software code is to control the advancement of VDMA frame pointers. Indeed, a triple buffer is used in order to display the newly computed frame a soon as the Sobel filtering is finished while, at the same time, processing the following input frame.

The output display is vertically split in half. The right side displays the video before being processed while the left side displays the video processed with a Sobel filter.

The software has been developed to highlight the benefit of hardware acceleration. In that respect, the output video display switches between the data processing done in software then done in hardware. The changes occur about every 15 seconds and highlights the greater performances achieved once the software is accelerated. In our

case, the FPS is closed to 100 times faster:

- 0.45 fps (software)
- 42 fps (hardware accelerated)

4.4.3 Camera Link application

This demo project is based on a stand-alone EMC² Development Platform PCIe/104 OneBank™ board featuring a Zynq 7015 SoC.
The board connects to an FMC-CAMERALINK daughter board in order to grab frames from a CameraLink compatible camera. Such frames are stored into DDR memory, processed in some way (in our demo, by a Sobel filter) and then output to an HDMI compatible screen.
This demo shows how most of the computing intensive tasks are implemented, where possible, in hardware, both using standard or custom IP cores and by using High Level Synthesis tools.

In our demo the input signal comes from a Sentech STC-CLC83A camera, which has a resolution of 1024x768, 30 fps, 8 bits per pixel, Bayer color scheme, BASE only configuration. The output video signal goes to the HDMI out chip ADV7511 which is available on the Sundance's SEIC extension board.. The output settings fit XGA standard: 1024x768, 60 fps, RGB.

This demo contains a custom IP core that interface with the CameraLink camera.

For more details about this demo please refer to:
https://hackaday.io/project/27372-cameralink-on-the-zynq-based-tulipp-platform