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.

Certificate Number FM55022

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

Table of Contents

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 enviroments

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.

1.1 Hardware

1.1.1 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.

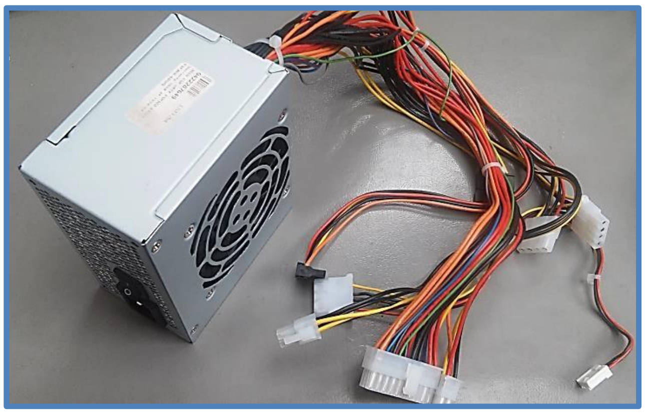


Figure 1: Power supply

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

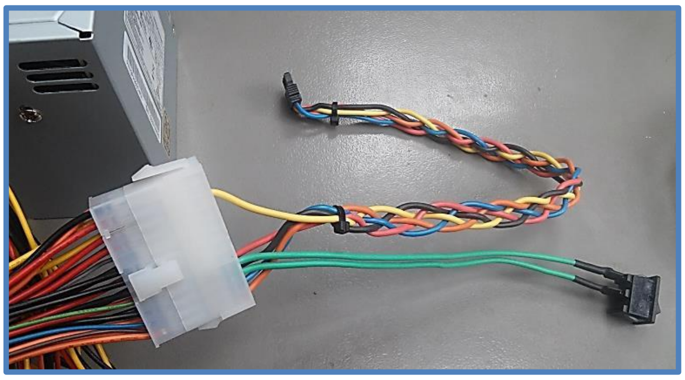


Figure 2: Cable for the PSU

And connect the cable to the board.

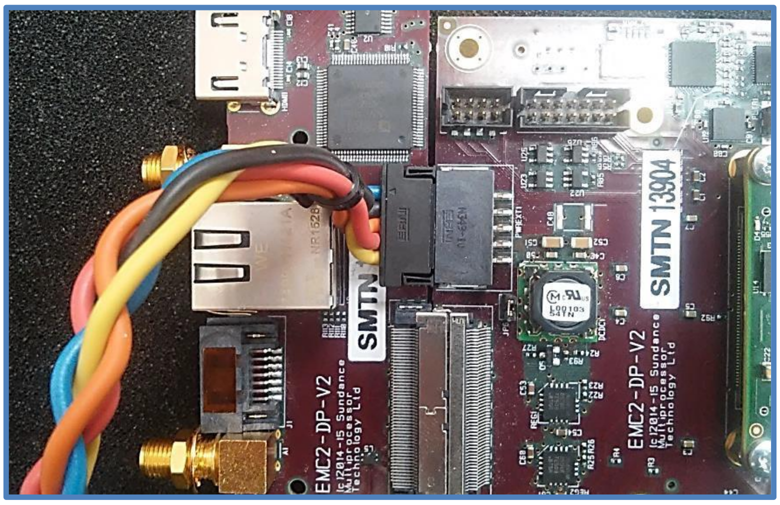


Figure 3: Connecting the EMC² to the PSU

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

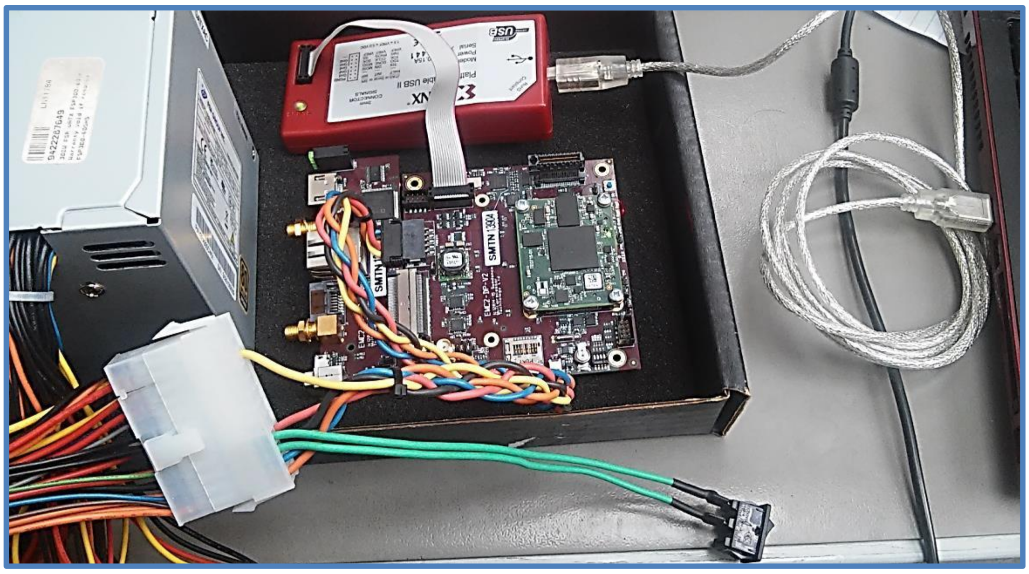


Figure 4: EMC² fully working

1.1.2 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.

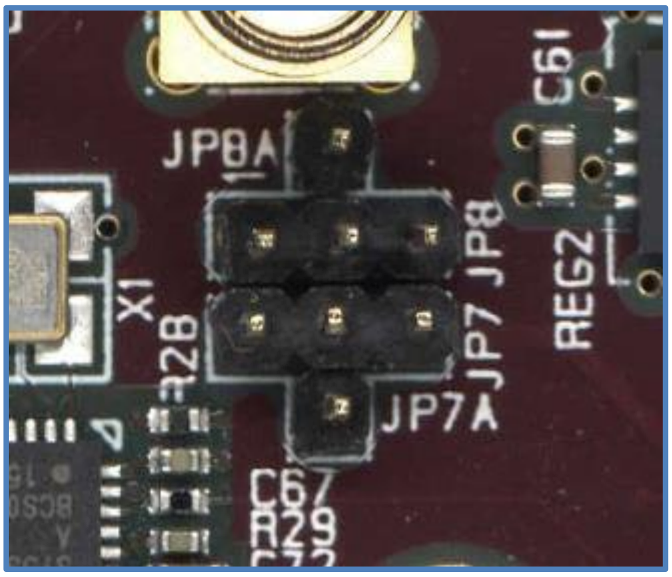


Figure 5: JP7 and JP8

JP7 and JP8 select the different voltages available as follows:

3.3V: Position 1-2

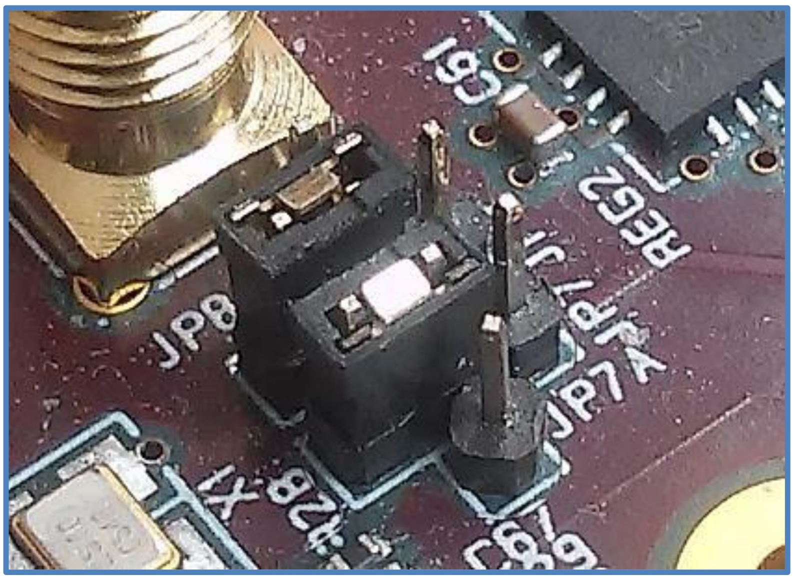


Figure 6: 3.3V selection

2.5V: Position 2-JP7A and 2-JP8A

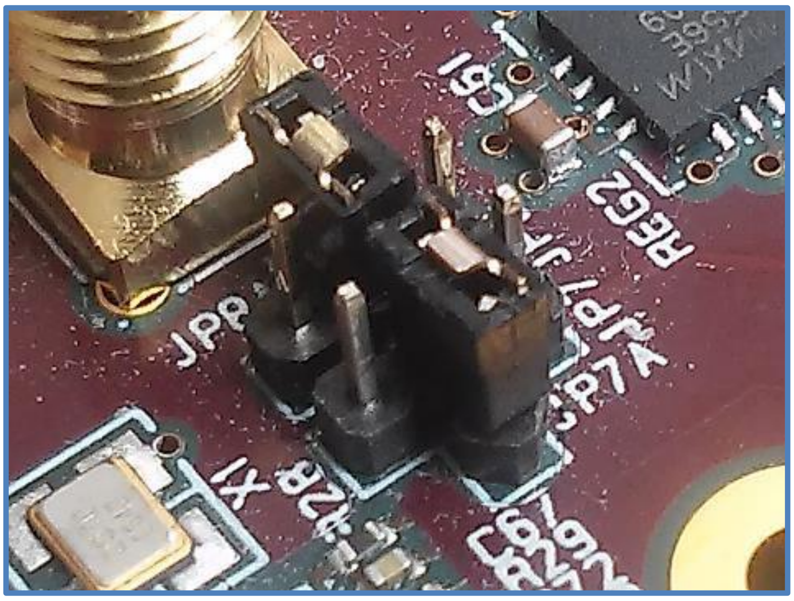


Figure 7: 2.5V selection

1.8V: Position 2-3

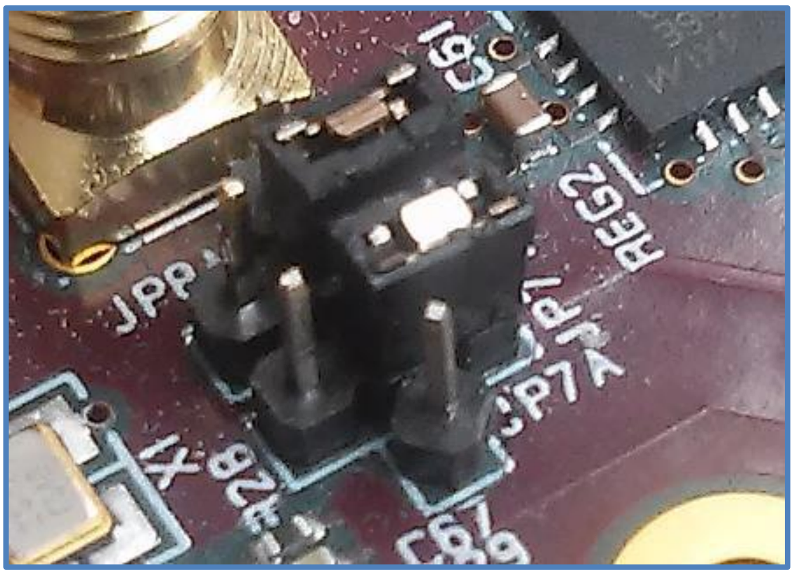


Figure 8: 1.8V selection

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)

1.1.3 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.

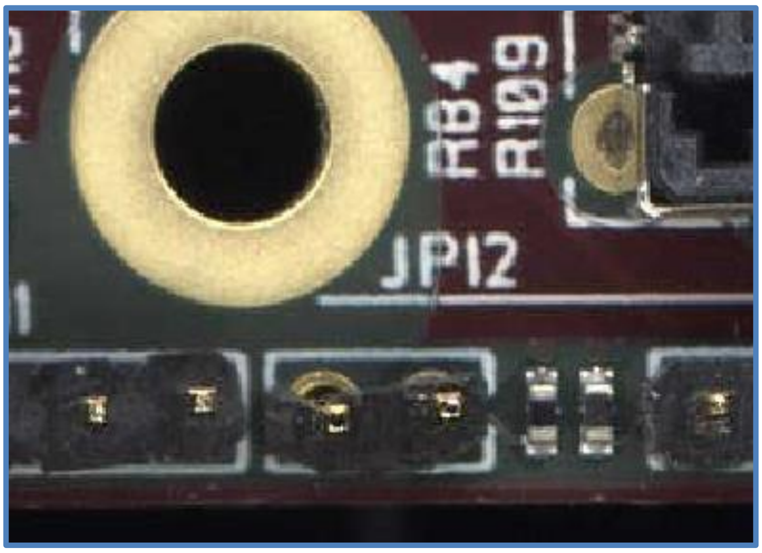


Figure 9: Host Jumper

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:

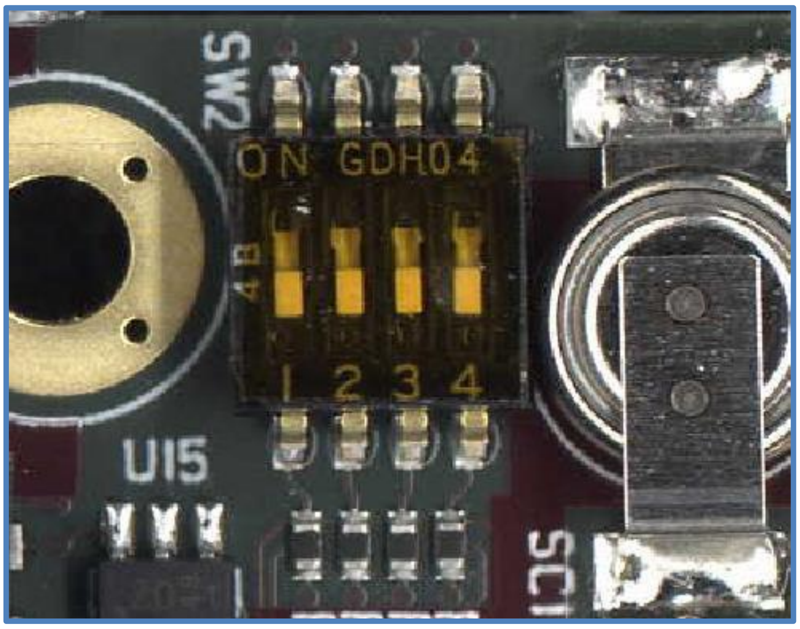


Figure 10: Upstream port selection

(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.

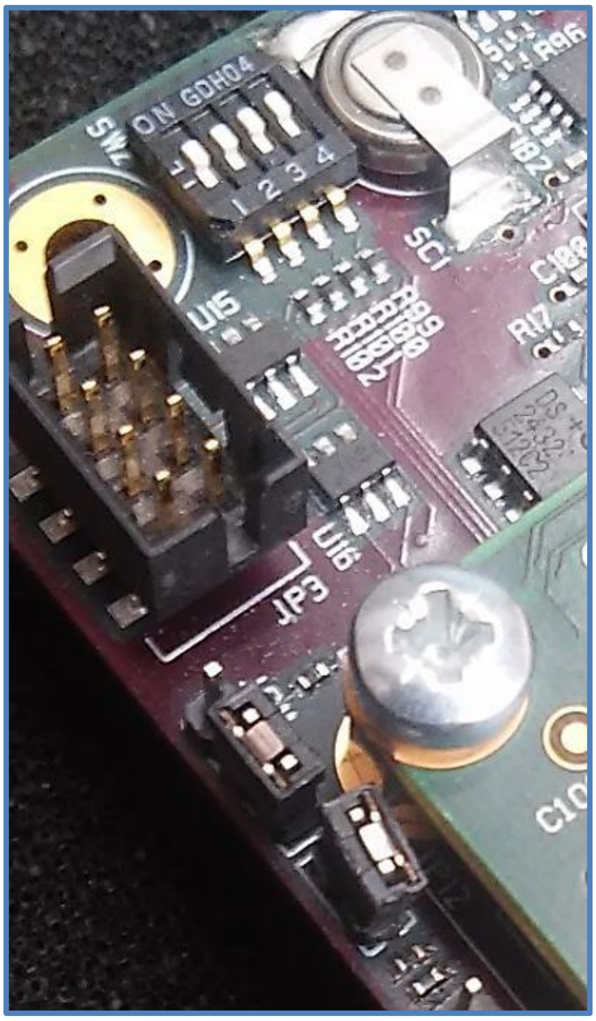


Figure 11: JP12 sets the host mode, SW2 selects port 0 as upstream port

1.1.4 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.

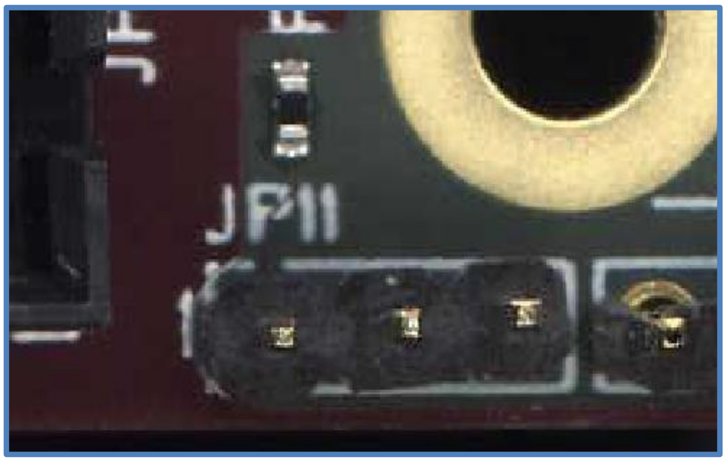


Figure 12: Boot mode selection

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 7, Next to JP12, JP11 is set as SD booting mode.

1.1.5 Information about the SEIC Connector

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

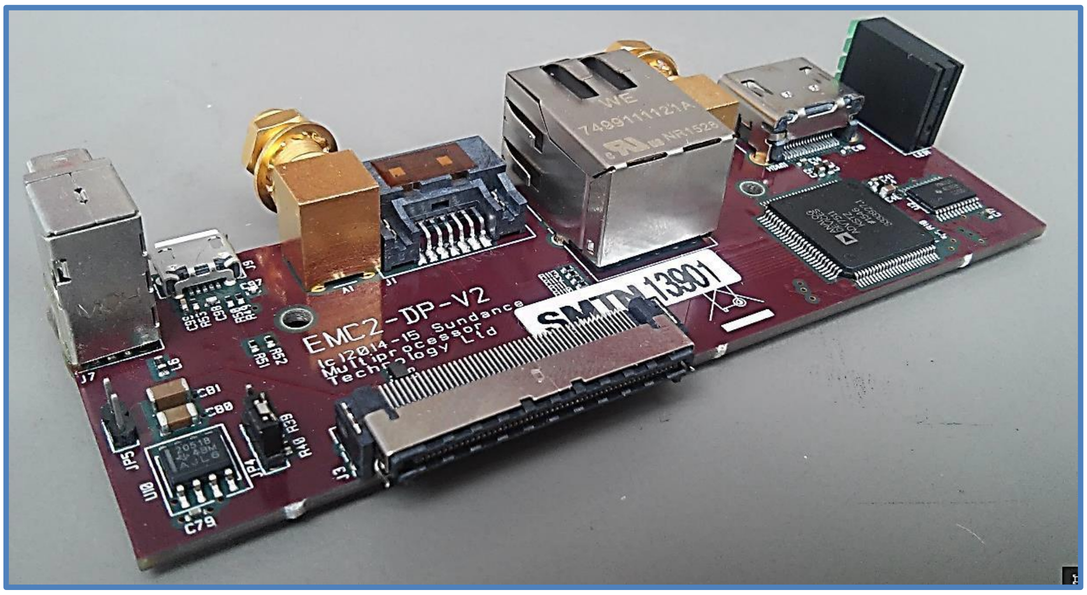


Figure 13: SEIC extension board

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

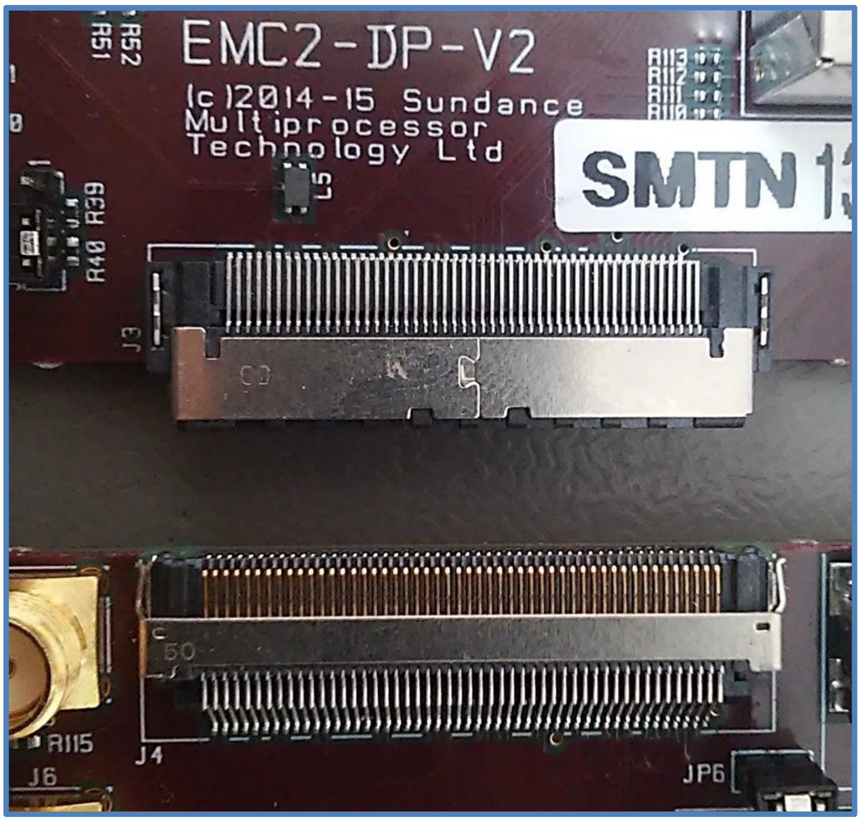


Figure 14: SEIC connector

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.

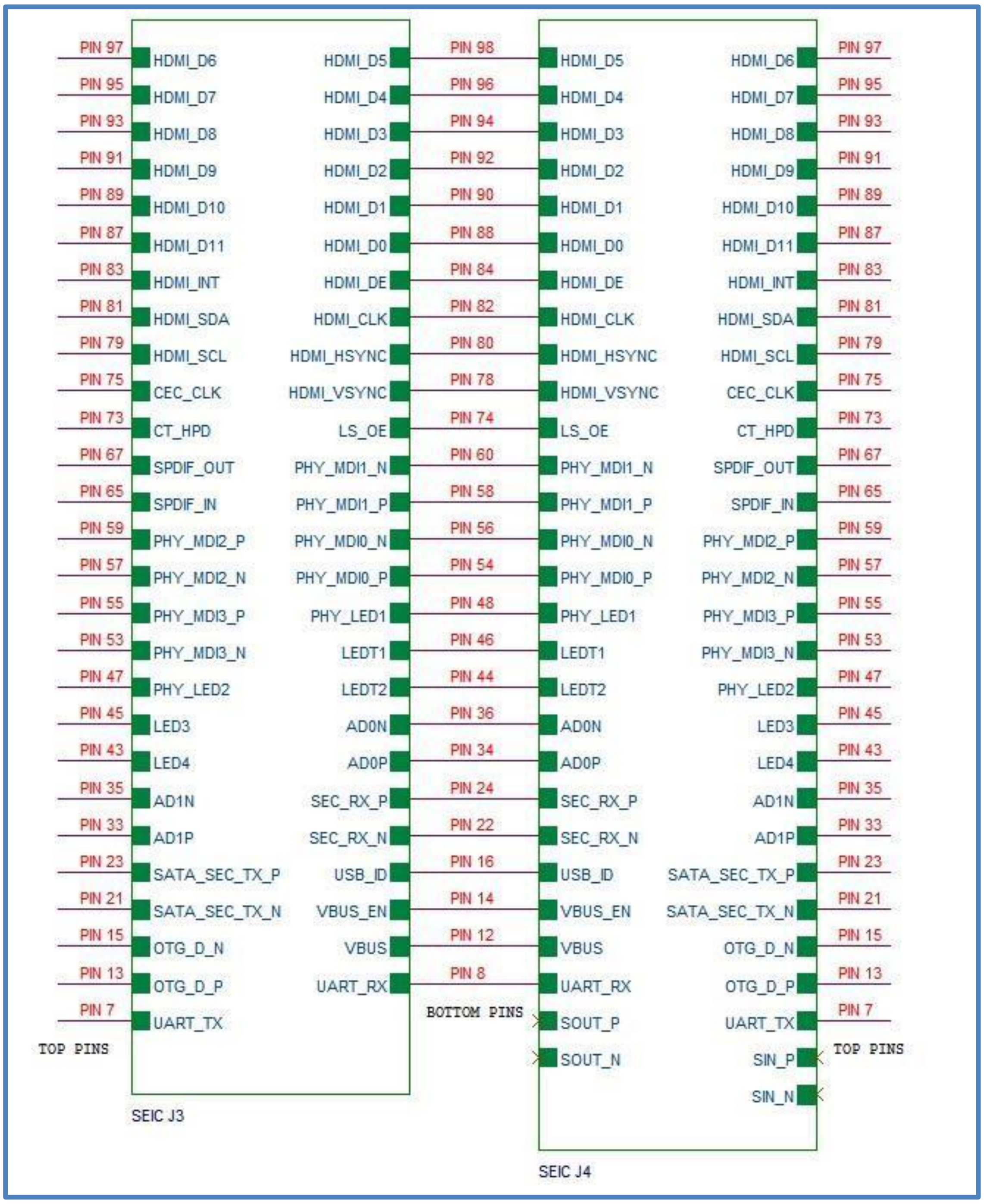


Figure 15: SEIC Connector Pinout

1.2 Software

1.2.1 How can I use Vivado with the EMC²?

The EMC² is well supported in 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 arquitecture.

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

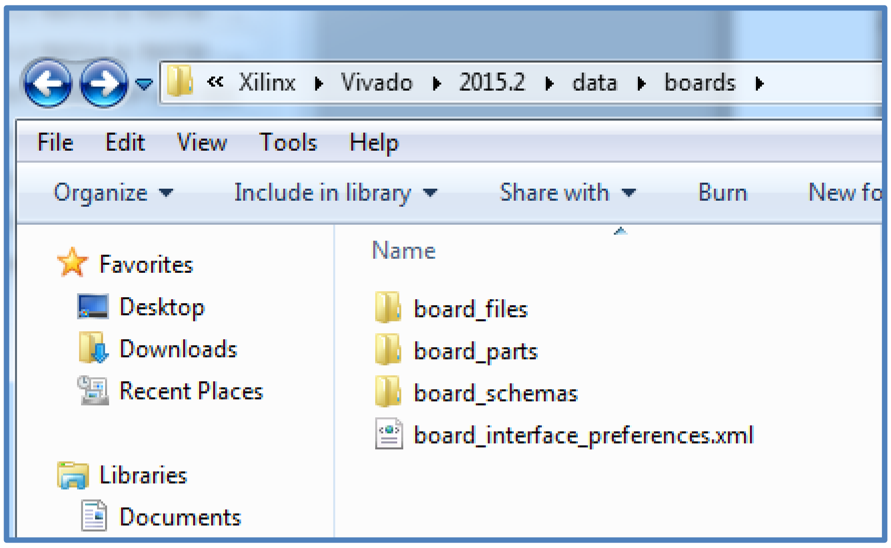
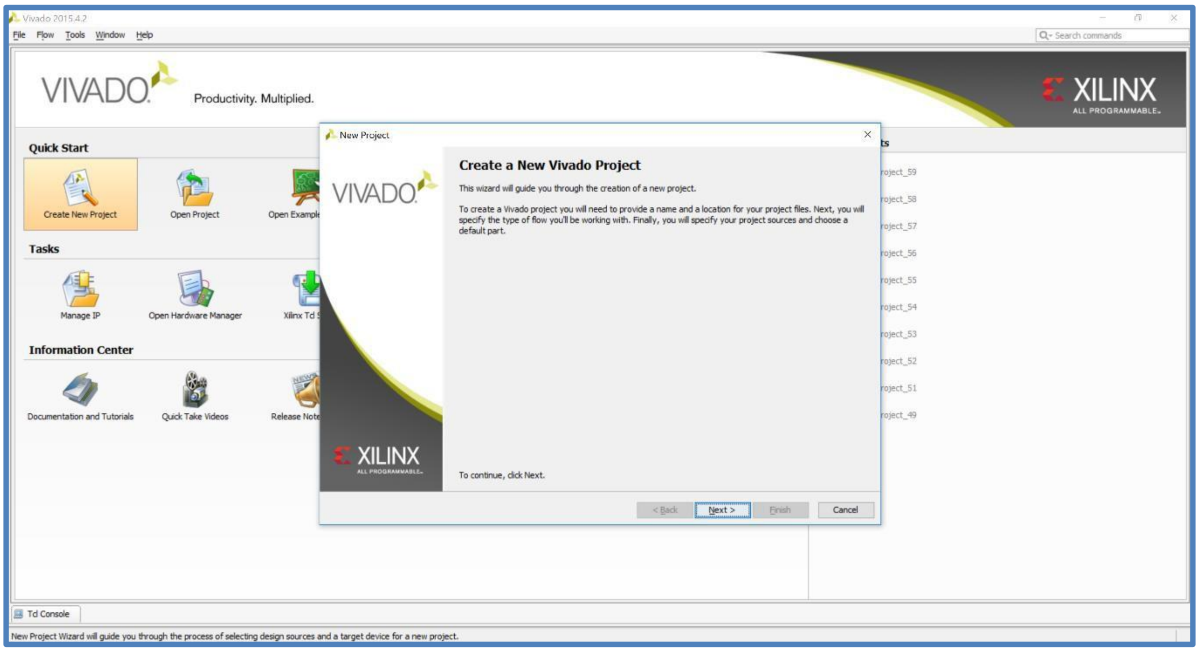


Figure 16: Board files path

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

Open Vivado and create a new project:



Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

To continue, click Next.

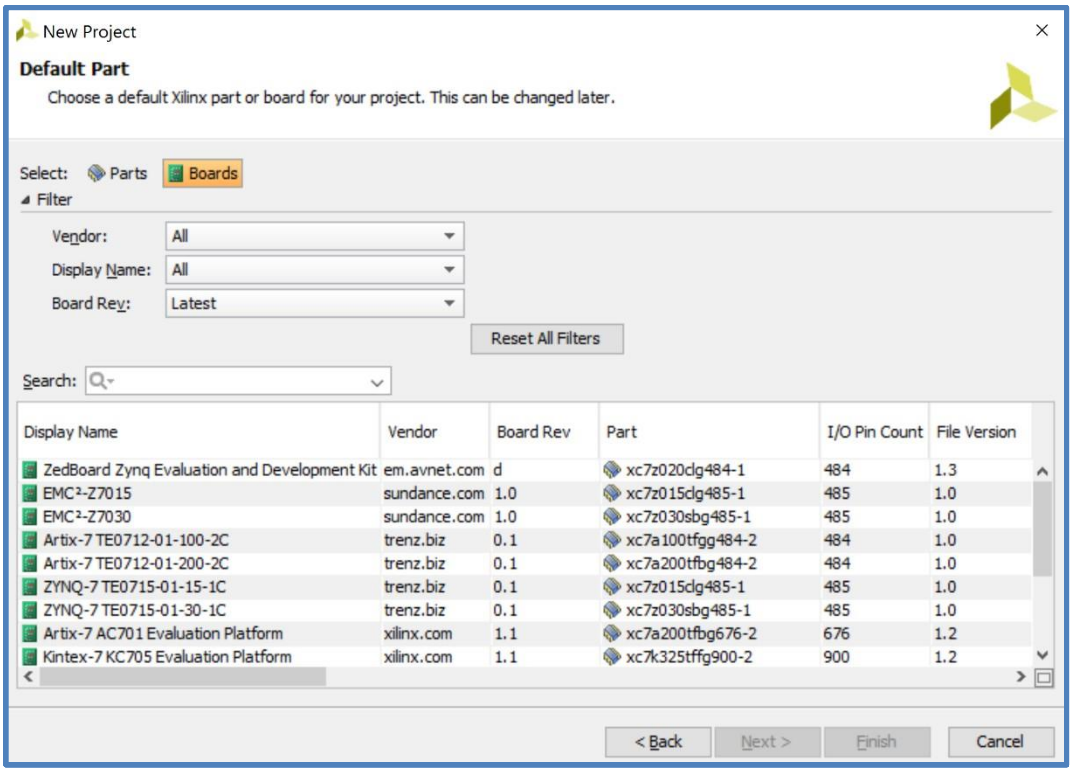
New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.

Figure 17: Create a new project

Clic “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:



Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Vendor: All

Display Name: All

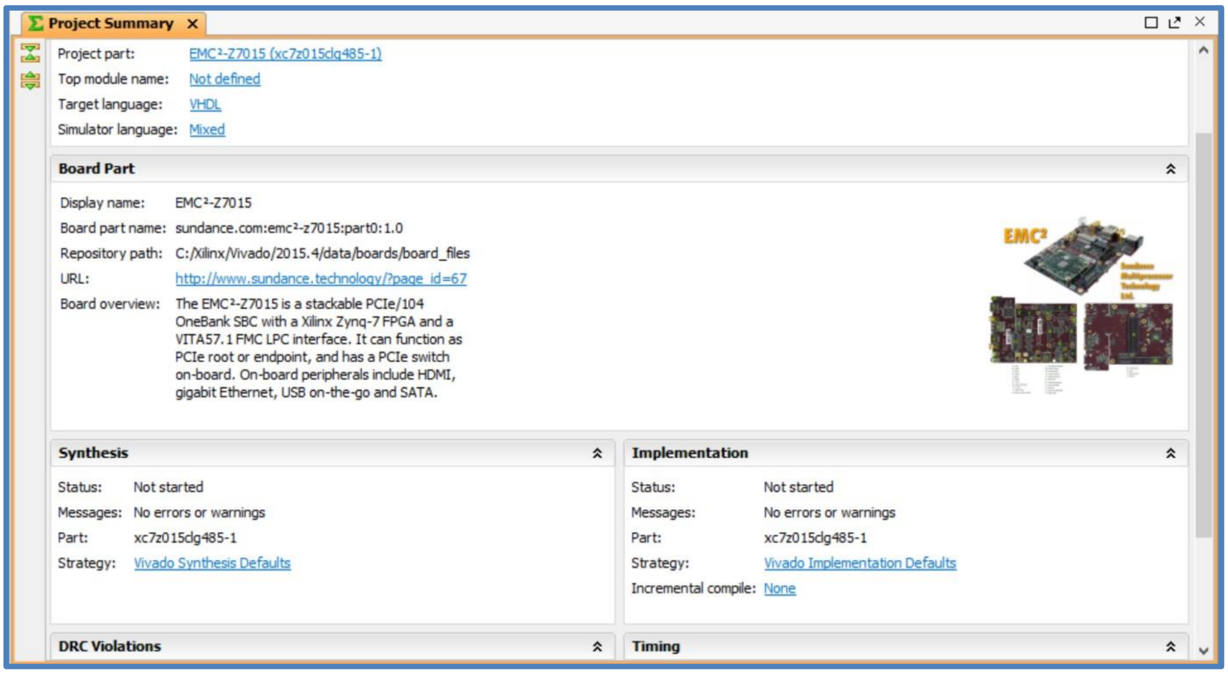
Board Rev: Latest

Search:

Display Name	Vendor	Board Rev	Part	I/O Pin Count	File Version
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484	1.3
EMC²-Z7015	sundance.com	1.0	xc7z015clg485-1	485	1.0
EMC²-Z7030	sundance.com	1.0	xc7z030sbg485-1	485	1.0
Artix-7 TE0712-01-100-2C	trenz.biz	0.1	xc7a100tfgg484-2	484	1.0
Artix-7 TE0712-01-200-2C	trenz.biz	0.1	xc7a200tfbg484-2	484	1.0
ZYNQ-7 TE0715-01-15-1C	trenz.biz	0.1	xc7z015clg485-1	485	1.0
ZYNQ-7 TE0715-01-30-1C	trenz.biz	0.1	xc7z030sbg485-1	485	1.0
Artix-7 AC701 Evaluation Platform	xilinx.com	1.1	xc7a200tfbg676-2	676	1.2
Kintex-7 KC705 Evaluation Platform	xilinx.com	1.1	xc7k325tffg900-2	900	1.2

Figure 18: Board file selection

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



Project Summary

Project part: EMC²-Z7015 (xc7z015clg485-1)
Top module name: Not defined
Target language: VHDL
Simulator language: Mixed

Board Part

Display name: EMC²-Z7015
Board part name: sundance.com:emc²-z7015:part0:1.0
Repository path: C:/Xilinx/Vivado/2015.4/data/boards/board_files
URL: http://www.sundance.technology/?page_id=67
Board overview: The EMC²-Z7015 is a stackable PCIe/104 OneBank SBC with a Xilinx Zynq-7 FPGA and a VITA57.1 FMC LPC interface. It can function as PCIe root or endpoint, and has a PCIe switch on-board. On-board peripherals include HDMI, gigabit Ethernet, USB on-the-go and SATA.

Synthesis

Status: Not started
Messages: No errors or warnings
Part: xc7z015clg485-1
Strategy: Vivado Synthesis Defaults

Implementation

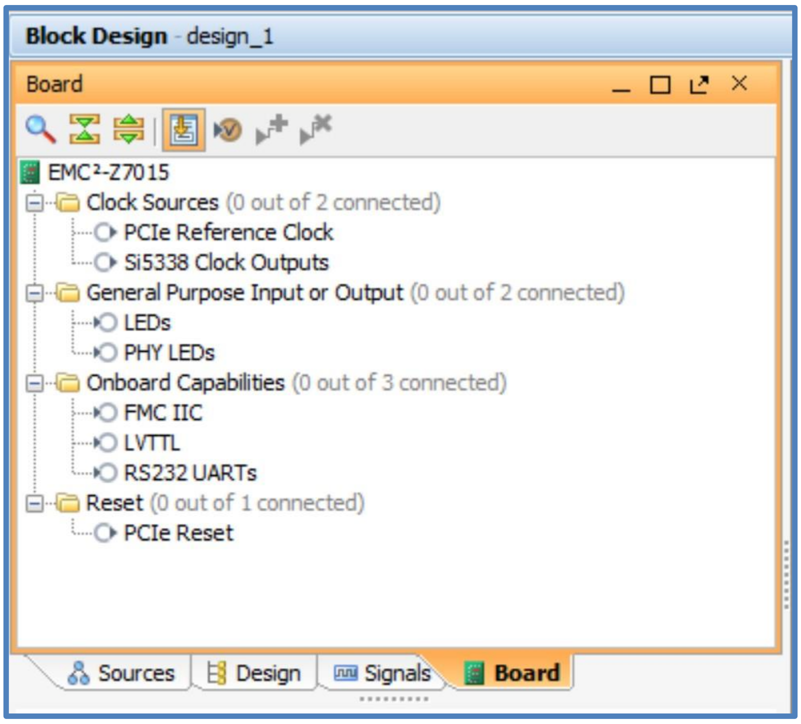
Status: Not started
Messages: No errors or warnings
Part: xc7z015clg485-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

DRC Violations

Timing

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:



Block Design - design_1

Board

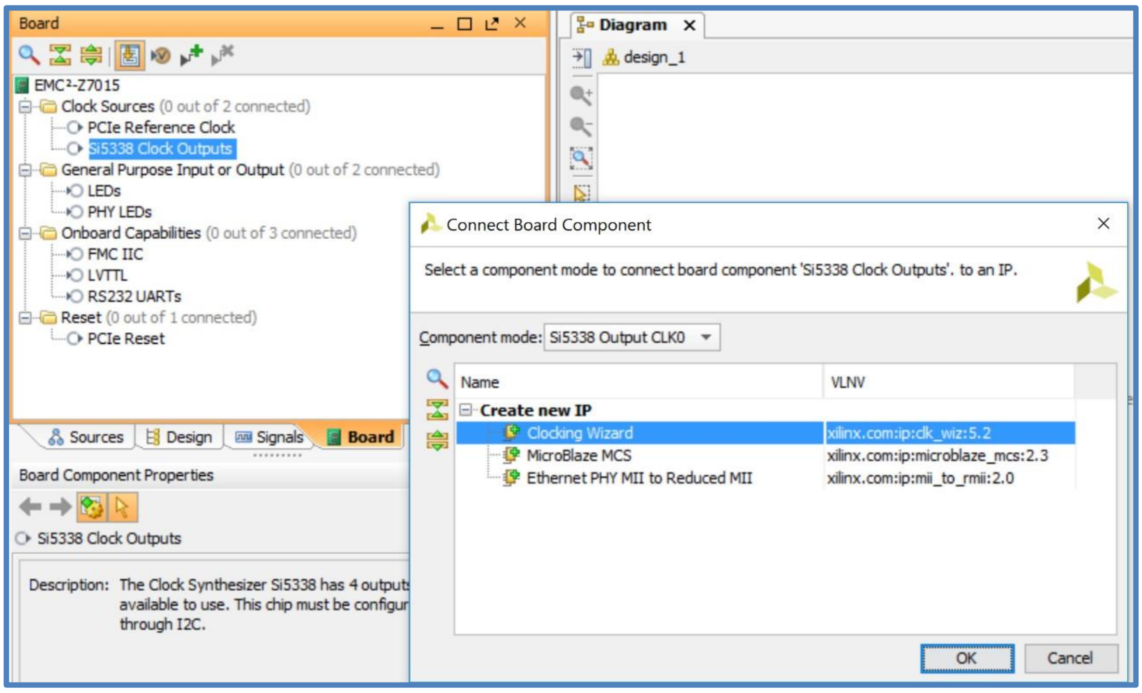
EMC²-Z7015
- Clock Sources (0 out of 2 connected)
  - PCIe Reference Clock
  - Si5338 Clock Outputs
- General Purpose Input or Output (0 out of 2 connected)
  - LEDs
  - PHY LEDs
- Onboard Capabilities (0 out of 3 connected)
  - FMC IIC
  - LVTTL
  - RS232 UARTs
- Reset (0 out of 1 connected)
  - PCIe Reset

Sources | Design | Signals | Board

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 clic on Si5338 Clock Outputs, at Clock Sources:



Description: The Clock Synthesizer Si5338 has 4 outputs available to use. This chip must be configur through I2C.

Select a component mode to connect board component 'Si5338 Clock Outputs'. to an IP.

Component mode: Si5338 Output CLK0

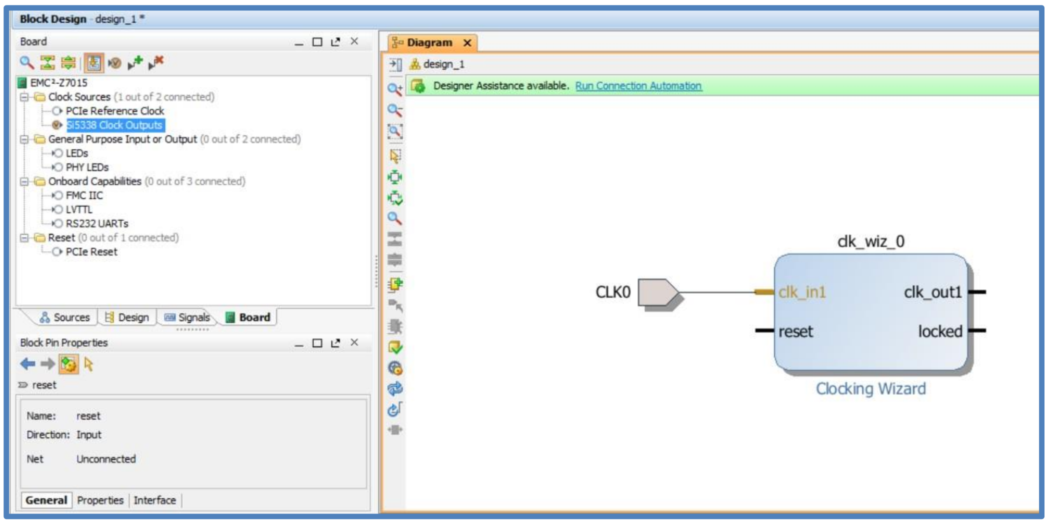
Name	VLNV
Create new IP	
Clocking Wizard	xilinx.com:ip:clk_wiz:5.2
MicroBlaze MCS	xilinx.com:ip:microblaze_mcs:2.3
Ethernet PHY MII to Reduced MII	xilinx.com:ip:mii_to_rmii:2.0

Figure 21: How to use interfaces in IPI

Select Ouput CLK0 as component mode, and Clocking Wizard as IP.

Automatically, the input will be assigned to the IP selected.

Remember that this 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.



Designer Assistance available. Run Connection Automation

Name: reset
Direction: Input
Net Unconnected

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.

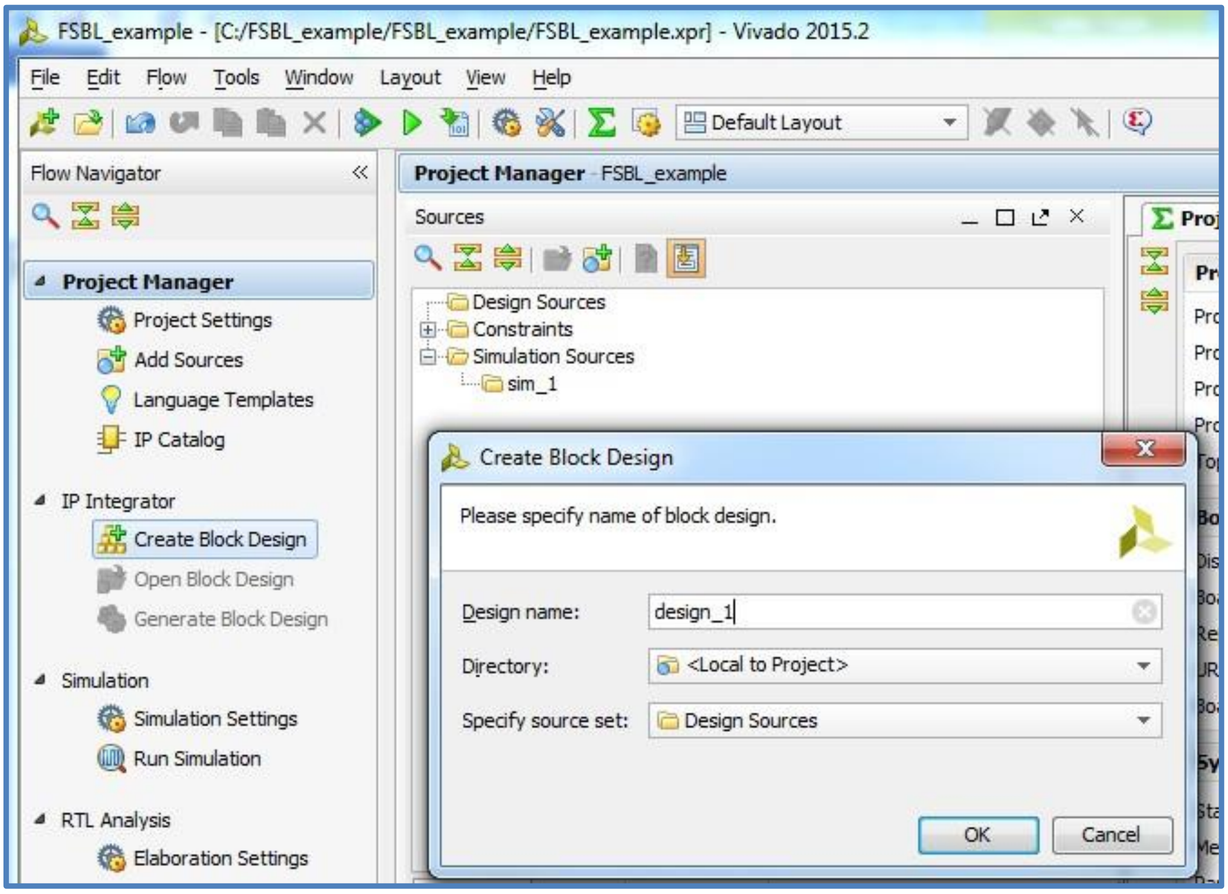
1.2.2 How can I create a Zynq FSBL for the EMC²?

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.

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


FSBL_example - [C:/FSBL_example/FSBL_example/FSBL_example.xpr] - Vivado 2015.2

Create Block Design

Please specify name of block design.

Design name: design_1

Directory: <Local to Project>

Specify source set: Design Sources

Figure 23: Create block design

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

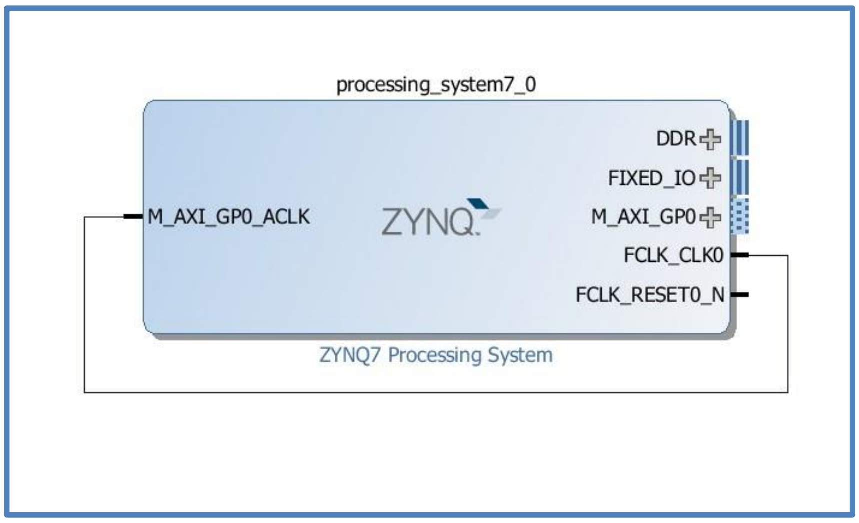


Figure 24: Zynq Processing System

Then, clic Run Block Automation, and press “OK”.

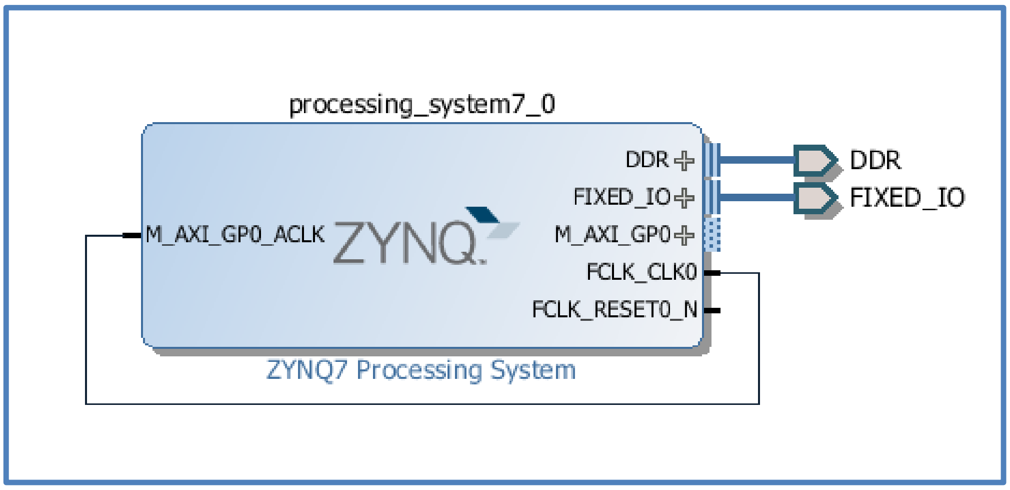


Figure 25: Run Block Automation

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

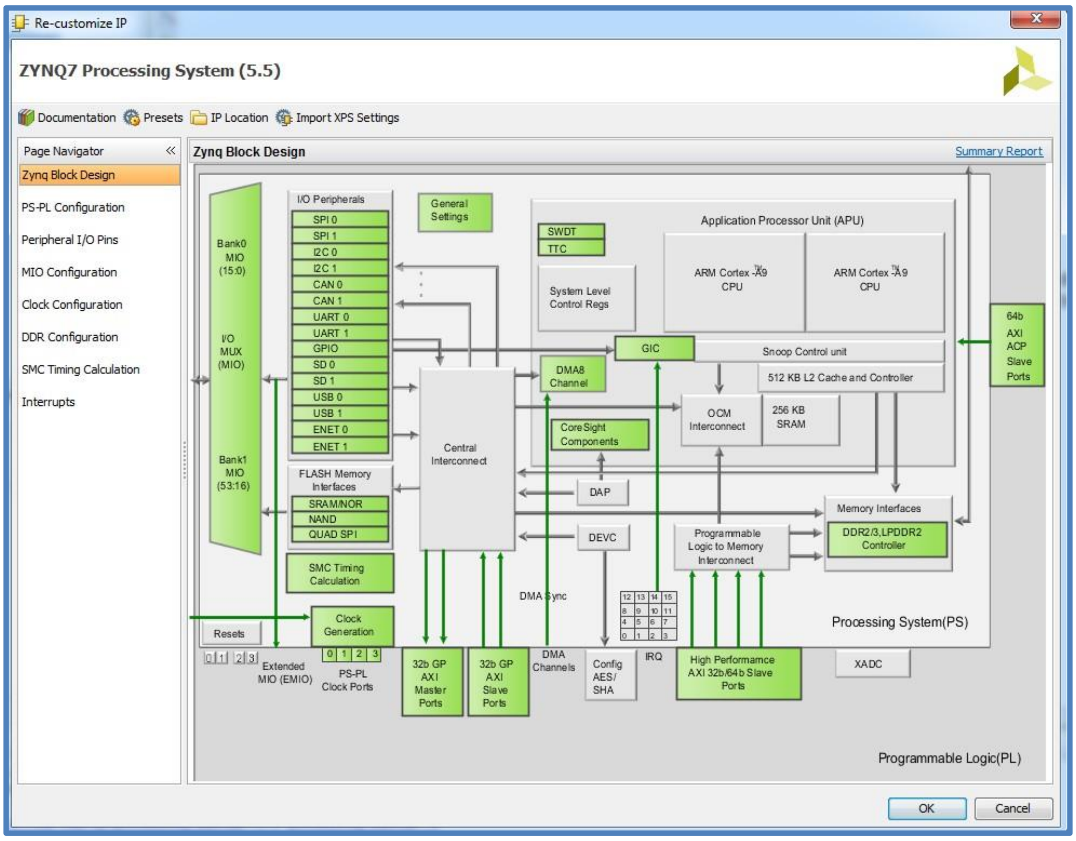


Figure 26: Zynq architecture

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.

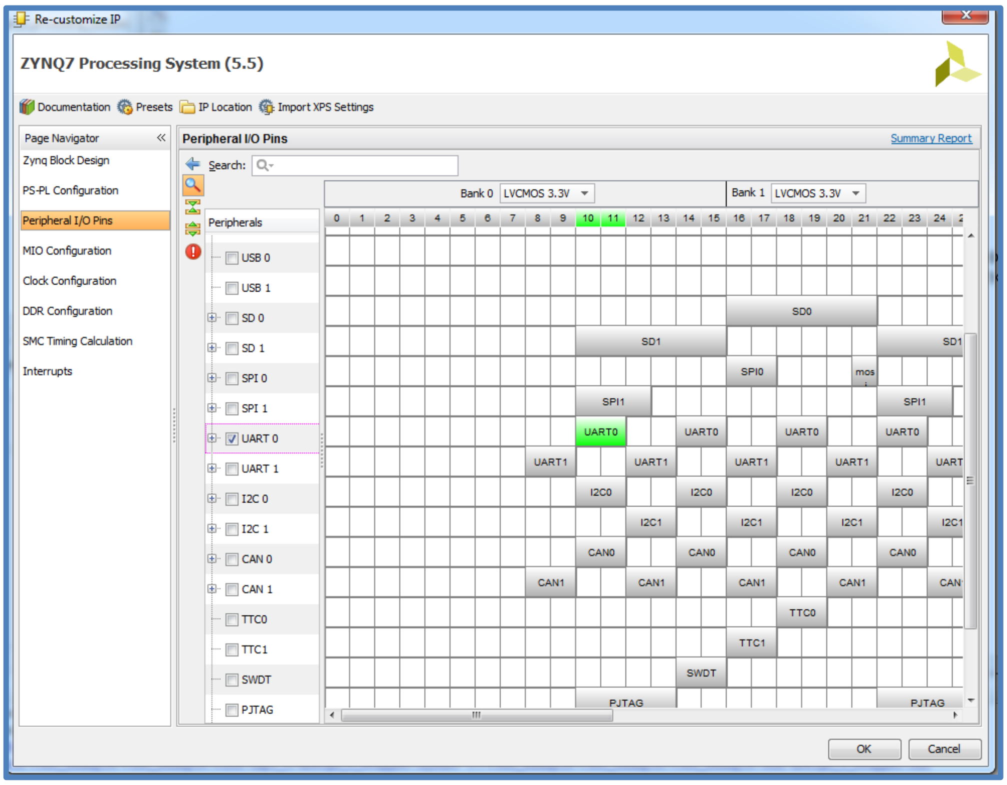
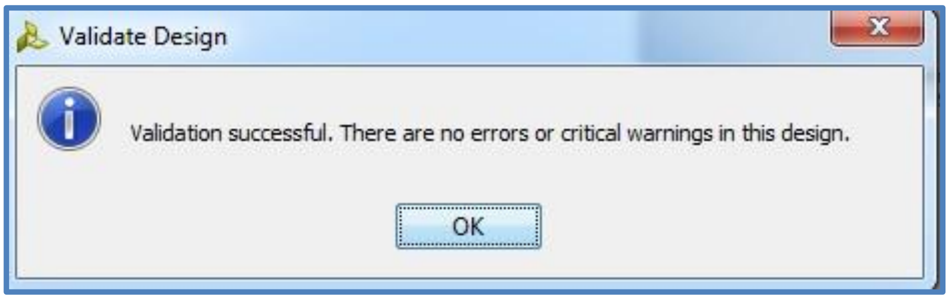


Figure 27: Adding Uart

Clic “Validate design” and then “OK”


Validate Design

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

Figure 28: Validate design

Create a Wrapper of the design, as follows:

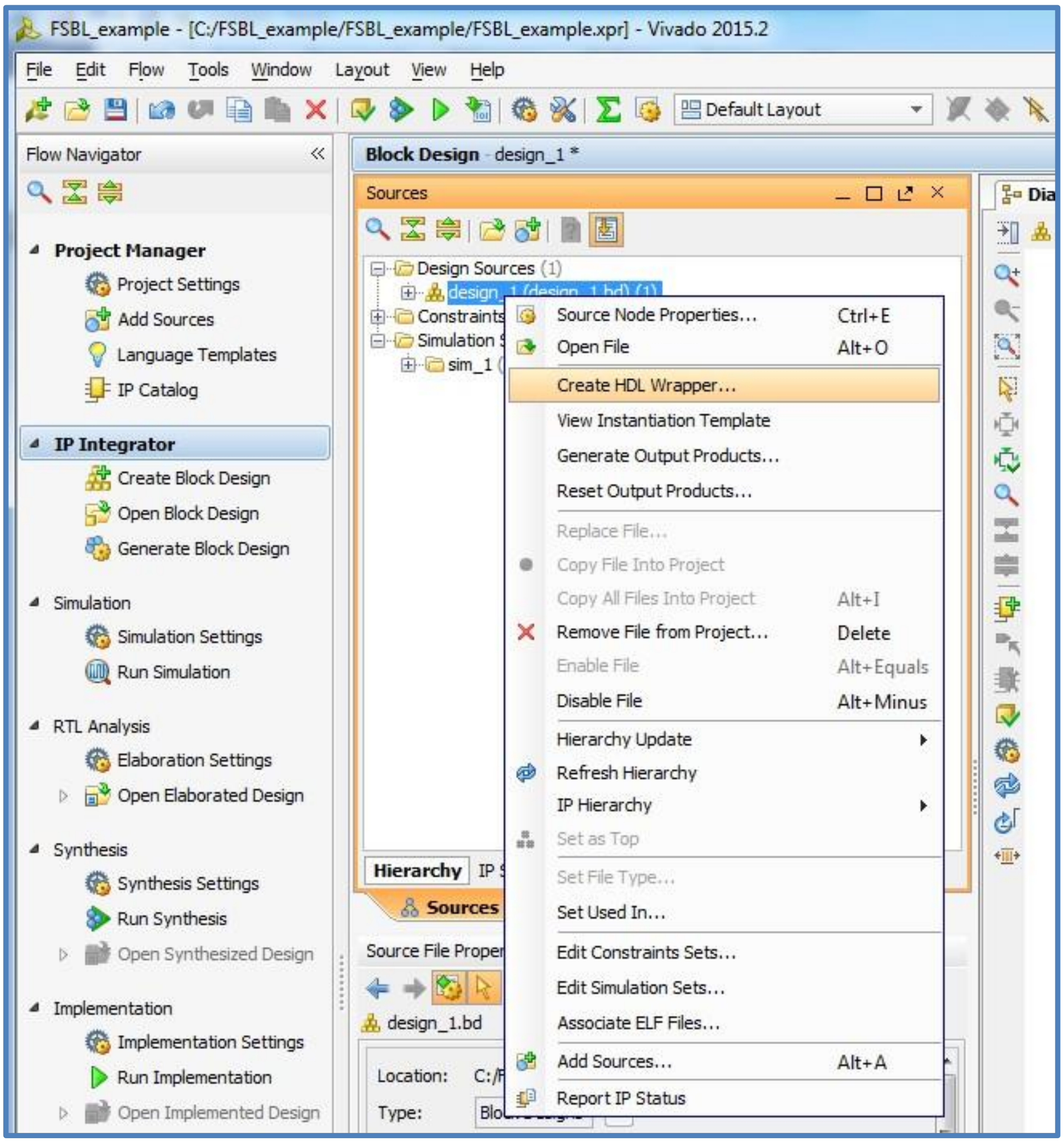

FSBL_example - [C:/FSBL_example/FSBL_example/FSBL_example.xpr] - Vivado 2015.2

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"

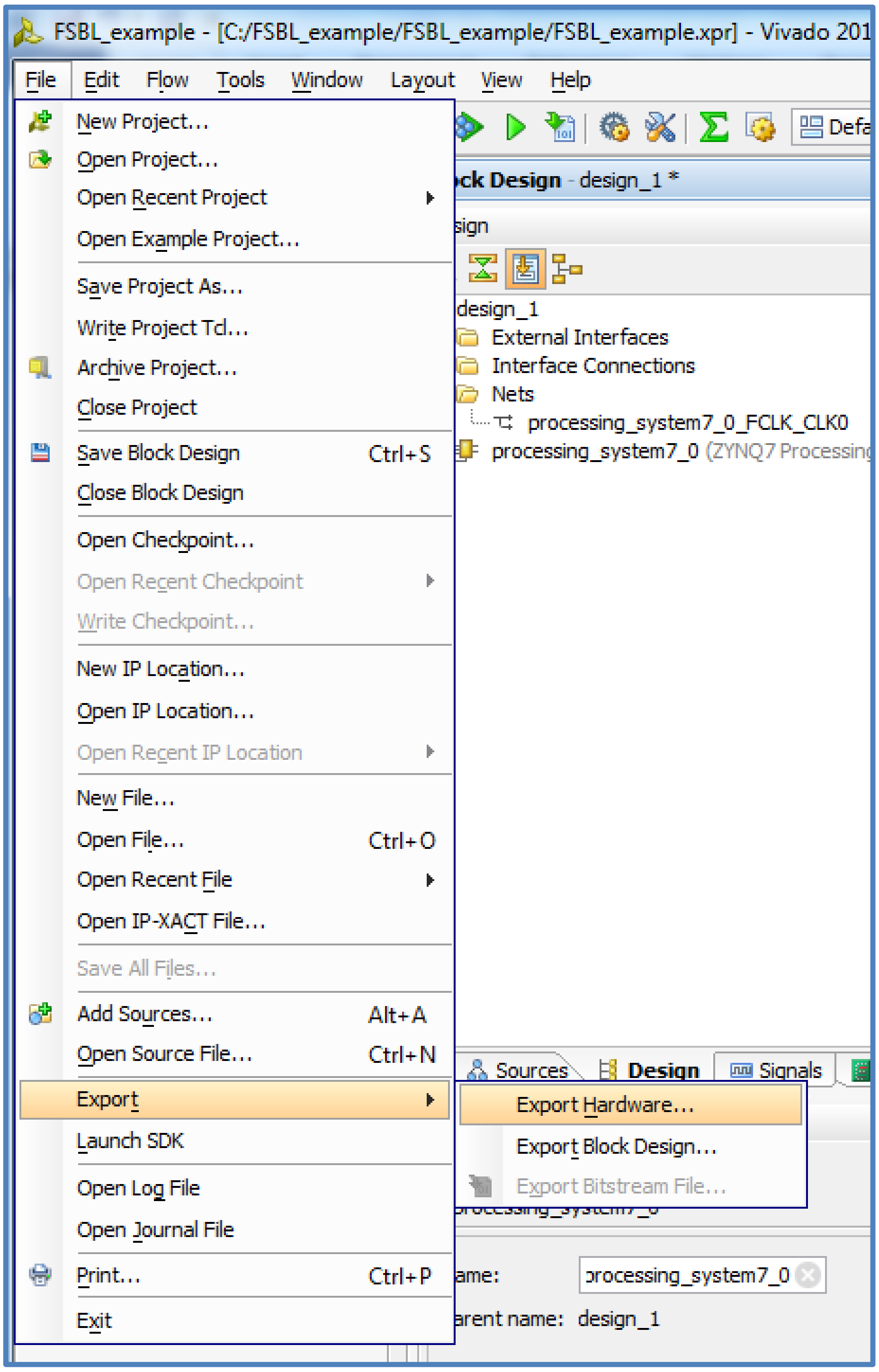
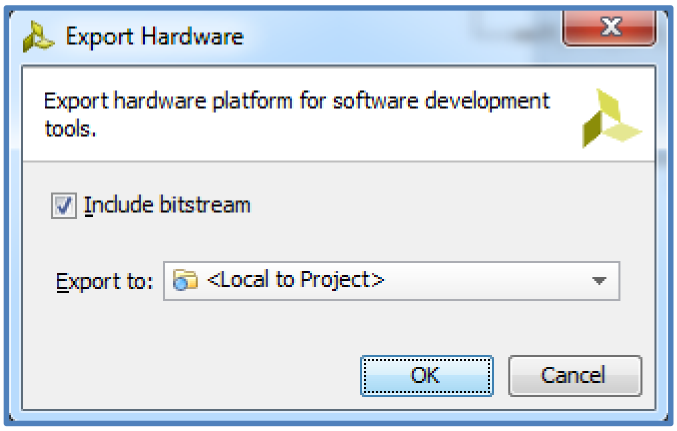


Figure 30: Export hardware

Mark "Include Bitstream", and export.



Export Hardware

Export hardware platform for software development tools.

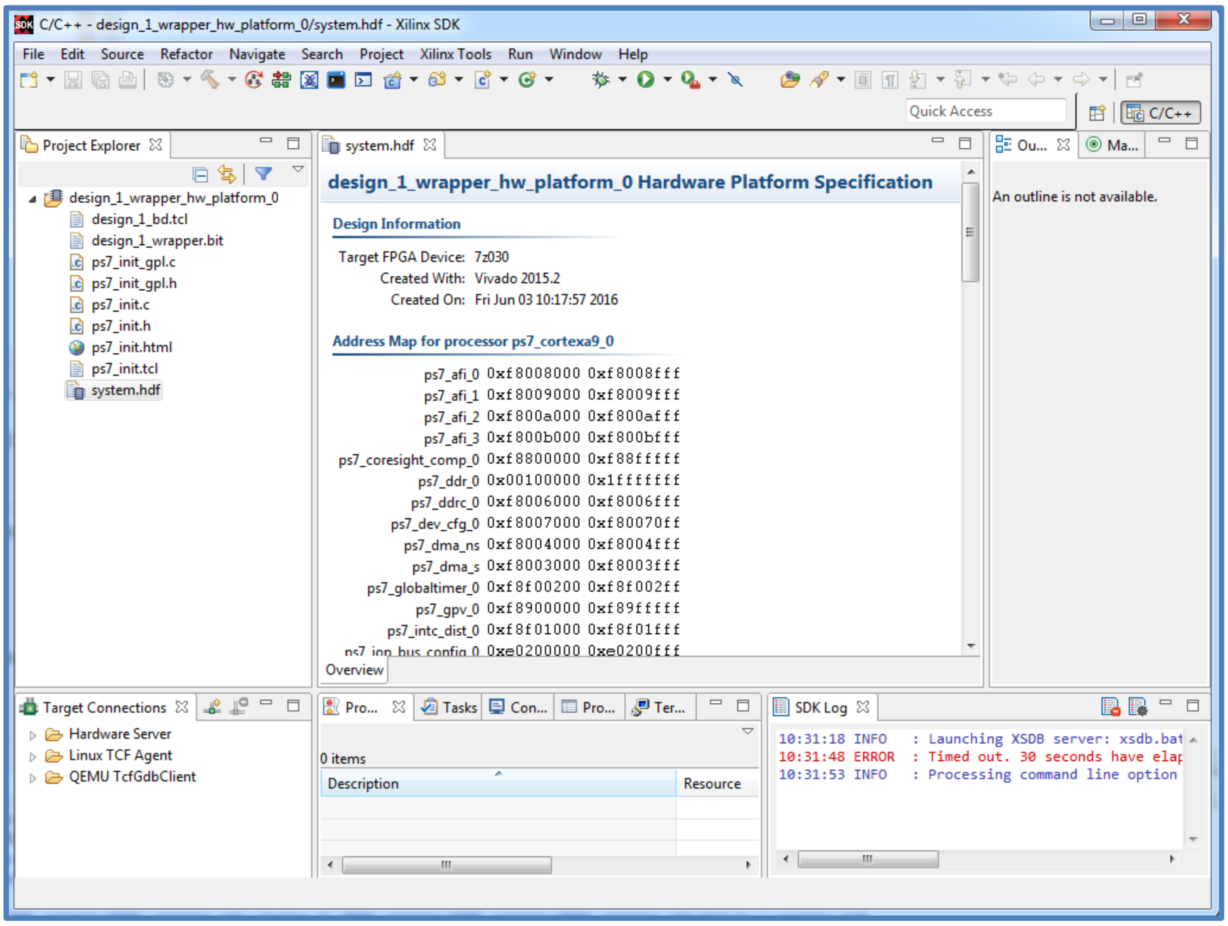
[✓] Include bitstream

Export to: <Local to Project>

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 initialises the PS in the Zynq.



C/C++ - design_1_wrapper_hw_platform_0/system.hdf - Xilinx SDK

design_1_wrapper_hw_platform_0
- design_1_bd.tcl
- design_1_wrapper.bit
- ps7_init_gpl.c
- ps7_init_gpl.h
- ps7_init.c
- ps7_init.h
- ps7_init.html
- ps7_init.tcl
- system.hdf

design_1_wrapper_hw_platform_0 Hardware Platform Specification

Design Information

Target FPGA Device: 7z030
Created With: Vivado 2015.2
Created On: Fri Jun 03 10:17:57 2016

Address Map for processor ps7_cortexa9_0

```
          ps7_afi_0 0xf8008000 0xf8008fff
          ps7_afi_1 0xf8009000 0xf8009fff
          ps7_afi_2 0xf800a000 0xf800afff
          ps7_afi_3 0xf800b000 0xf800bfff
ps7_coresight_comp_0 0xf8800000 0xf88fffff
          ps7_ddr_0 0x00100000 0x1fffffff
         ps7_ddrc_0 0xf8006000 0xf8006fff
      ps7_dev_cfg_0 0xf8007000 0xf80070ff
         ps7_dma_ns 0xf8004000 0xf8004fff
          ps7_dma_s 0xf8003000 0xf8003fff
  ps7_globaltimer_0 0xf8f00200 0xf8f002ff
          ps7_gpv_0 0xf8900000 0xf89fffff
     ps7_intc_dist_0 0xf8f01000 0xf8f01fff
  ps7_iop_bus_config_0 0xe0200000 0xe0200fff
```

An outline is not available.

- Hardware Server
- Linux TCF Agent
- QEMU TcfGdbClient

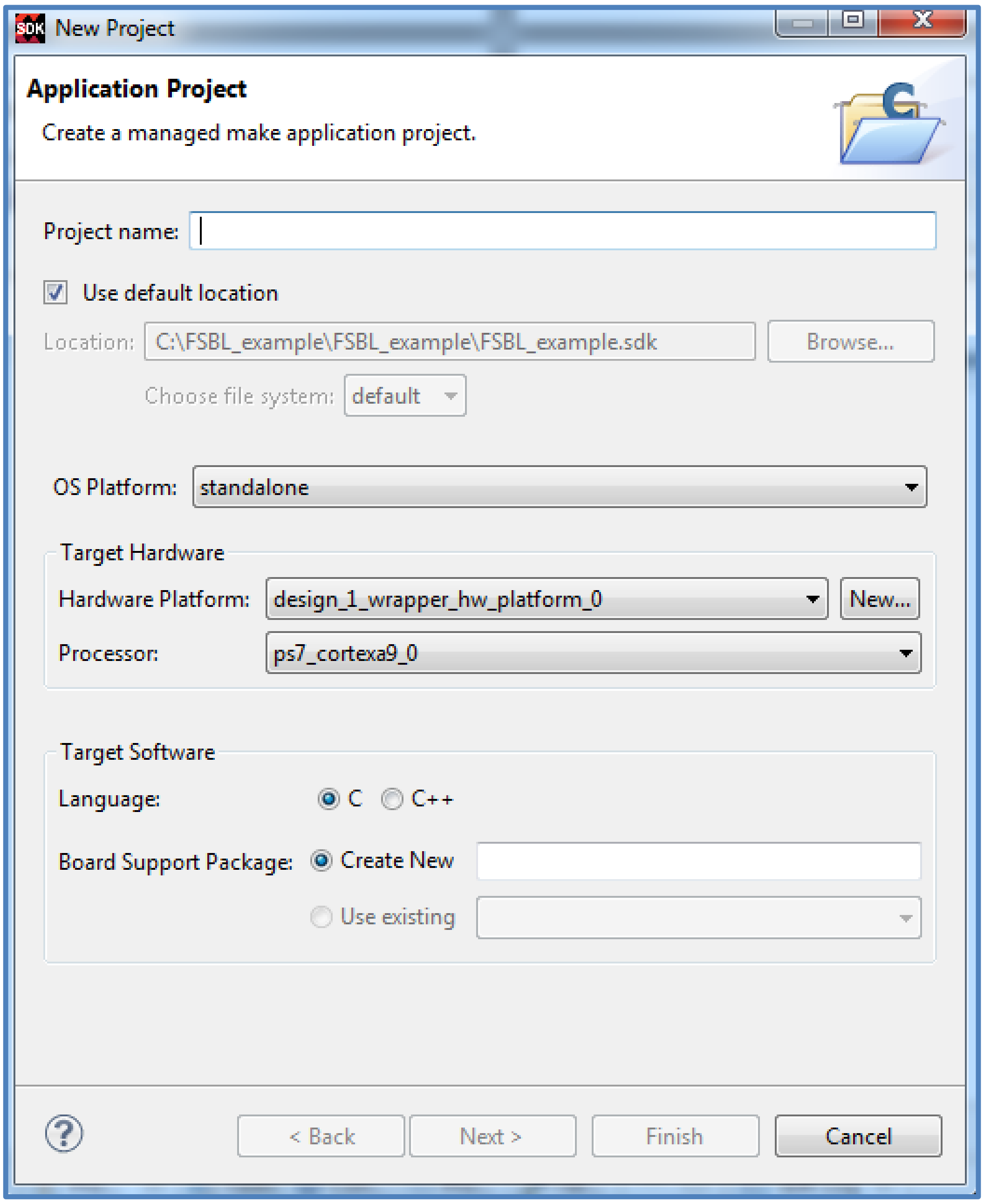
0 items

Description	Resource

```
10:31:18 INFO  : Launching XSDB server: xsdb.bat
10:31:48 ERROR : Timed out. 30 seconds have elap
10:31:53 INFO  : Processing command line option
```

Figure 32: SDK

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


New Project

Application Project

Create a managed make application project.

Project name:

[✓] Use default location

Location: C:\FSBL_example\FSBL_example\FSBL_example.sdk

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: design_1_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Target Software

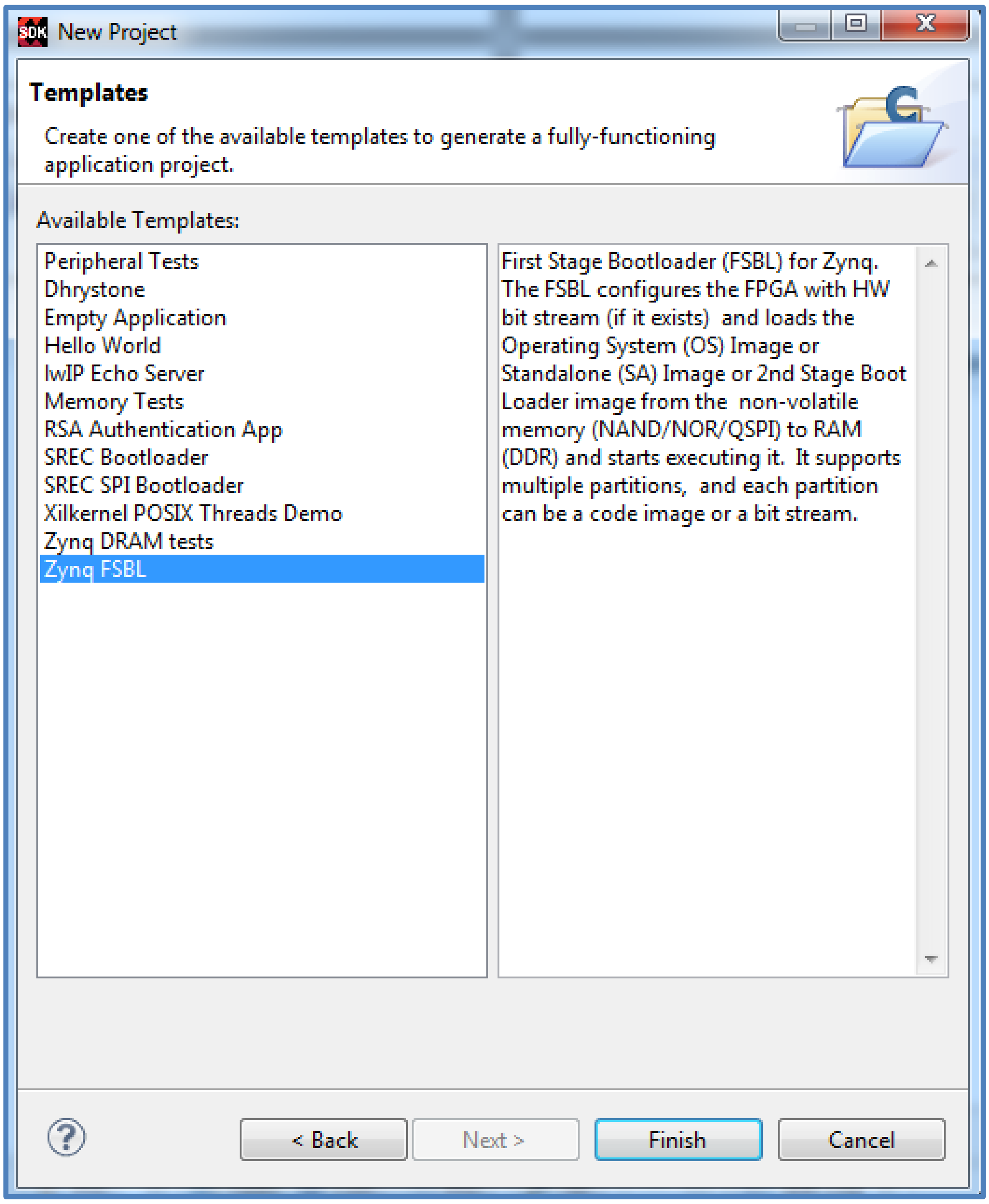
Language: [✓ C] C++

Board Support Package: [✓ Create New] Use existing

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”


New Project

Templates

Create one of the available templates to generate a fully-functioning application project.

Available Templates:

Peripheral Tests
Dhrystone
Empty Application
Hello World
lwIP Echo Server
Memory Tests
RSA Authentication App
SREC Bootloader
SREC SPI Bootloader
Xilkernel POSIX Threads Demo
Zynq DRAM tests
Zynq FSBL

First Stage Bootloader (FSBL) for Zynq. The FSBL configures the FPGA with HW bit stream (if it exists) and loads the Operating System (OS) Image or Standalone (SA) Image or 2nd Stage Boot Loader image from the non-volatile memory (NAND/NOR/QSPI) to RAM (DDR) and starts executing it. It supports multiple partitions, and each partition can be a code image or a bit stream.

Figure 34: Zynq FSBL project

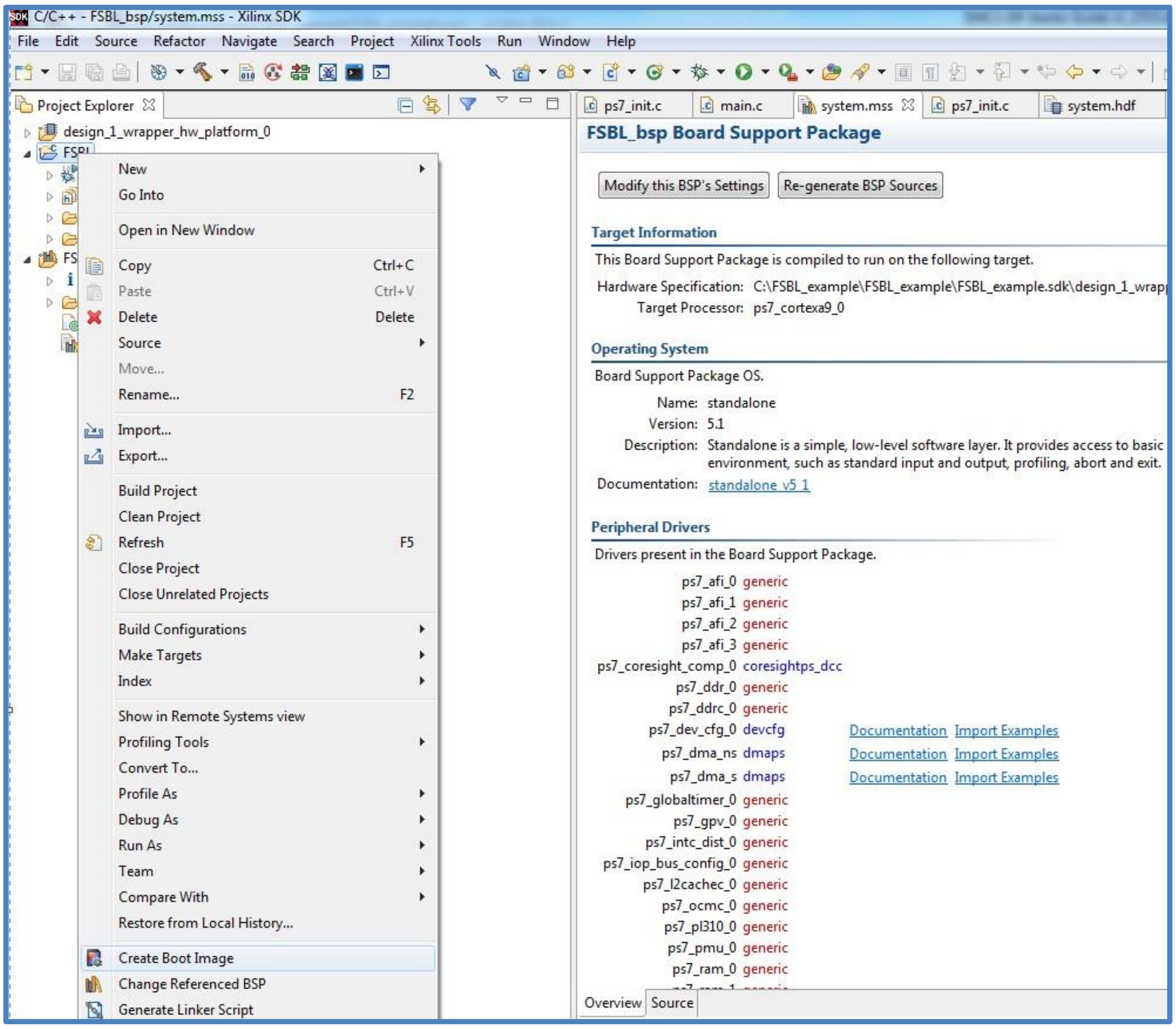
Press “Finish” and the FSBL project will be added.

Now the user has a ps7_init source in this project that can be used for initialise 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).

1.2.3 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.

Clic "FSBL -> Create Boot Image"



FSBL_bsp Board Support Package

Target Information

This Board Support Package is compiled to run on the following target.

Hardware Specification: C:\FSBL_example\FSBL_example\FSBL_example.sdk\design_1_wrap
Target Processor: ps7_cortexa9_0

Operating System

Board Support Package OS.

Name: standalone
Version: 5.1
Description: Standalone is a simple, low-level software layer. It provides access to basic environment, such as standard input and output, profiling, abort and exit.
Documentation: standalone v5 1

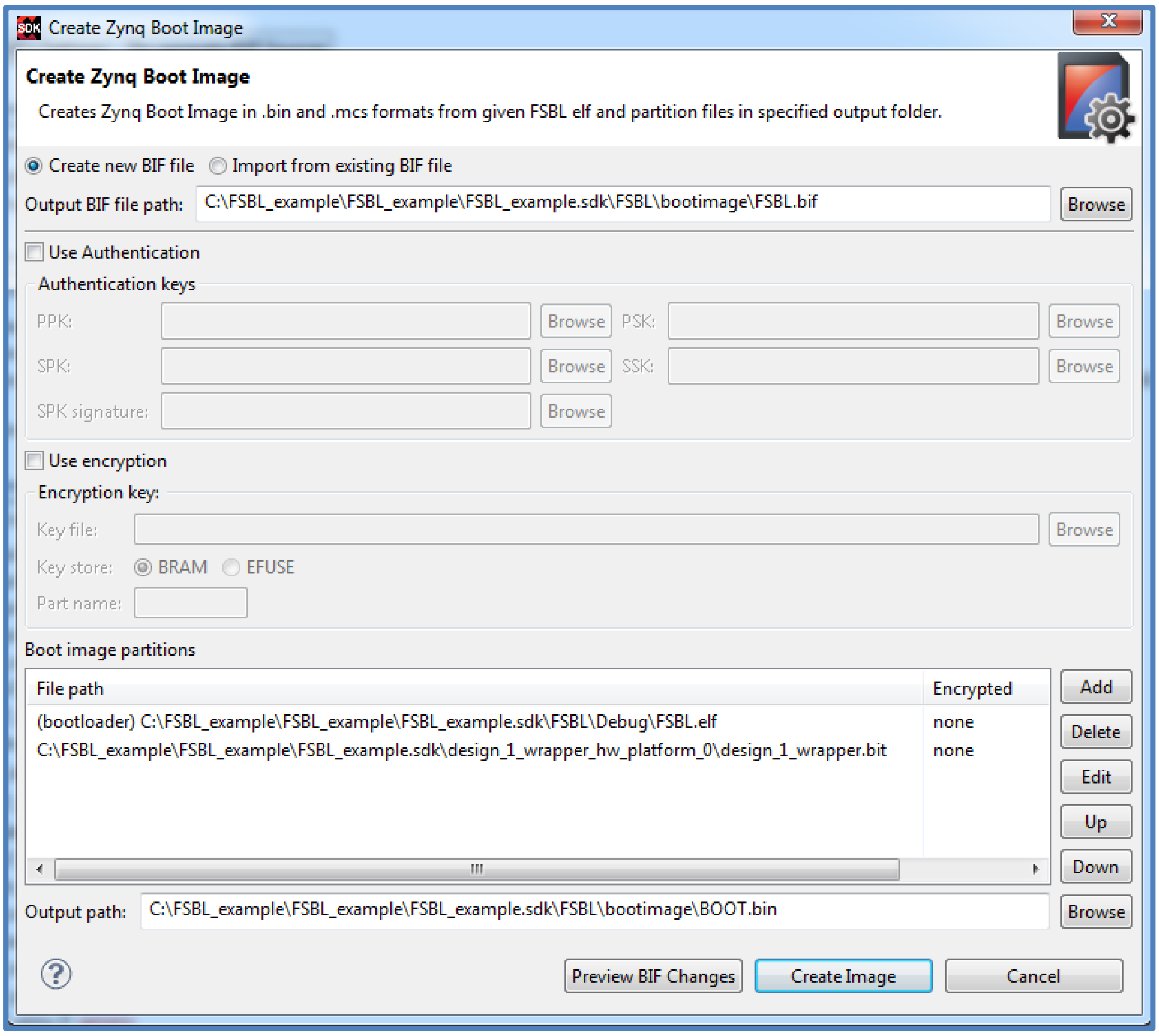
Peripheral Drivers

Drivers present in the Board Support Package.

ps7_afi_0	generic	
ps7_afi_1	generic	
ps7_afi_2	generic	
ps7_afi_3	generic	
ps7_coresight_comp_0	coresightps_dcc	
ps7_ddr_0	generic	
ps7_ddrc_0	generic	
ps7_dev_cfg_0	devcfg	Documentation Import Examples
ps7_dma_ns	dmaps	Documentation Import Examples
ps7_dma_s	dmaps	Documentation Import Examples
ps7_globaltimer_0	generic	
ps7_gpv_0	generic	
ps7_intc_dist_0	generic	
ps7_iop_bus_config_0	generic	
ps7_l2cachec_0	generic	
ps7_ocmc_0	generic	
ps7_pl310_0	generic	
ps7_pmu_0	generic	
ps7_ram_0	generic	

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).



Create Zynq Boot Image

Create Zynq Boot Image

Creates Zynq Boot Image in .bin and .mcs formats from given FSBL elf and partition files in specified output folder.

(●) Create new BIF file ( ) Import from existing BIF file

Output BIF file path: C:\FSBL_example\FSBL_example\FSBL_example.sdk\FSBL\bootimage\FSBL.bif

☐ Use Authentication

Authentication keys

PPK: PSK: SPK: SSK: SPK signature:

☐ Use encryption

Encryption key:

Key file:

Key store: (●) BRAM ( ) EFUSE

Part name:

Boot image partitions

File path	Encrypted
(bootloader) C:\FSBL_example\FSBL_example\FSBL_example.sdk\FSBL\Debug\FSBL.elf	none
C:\FSBL_example\FSBL_example\FSBL_example.sdk\design_1_wrapper_hw_platform_0\design_1_wrapper.bit	none

Output path: C:\FSBL_example\FSBL_example\FSBL_example.sdk\FSBL\bootimage\BOOT.bin

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.

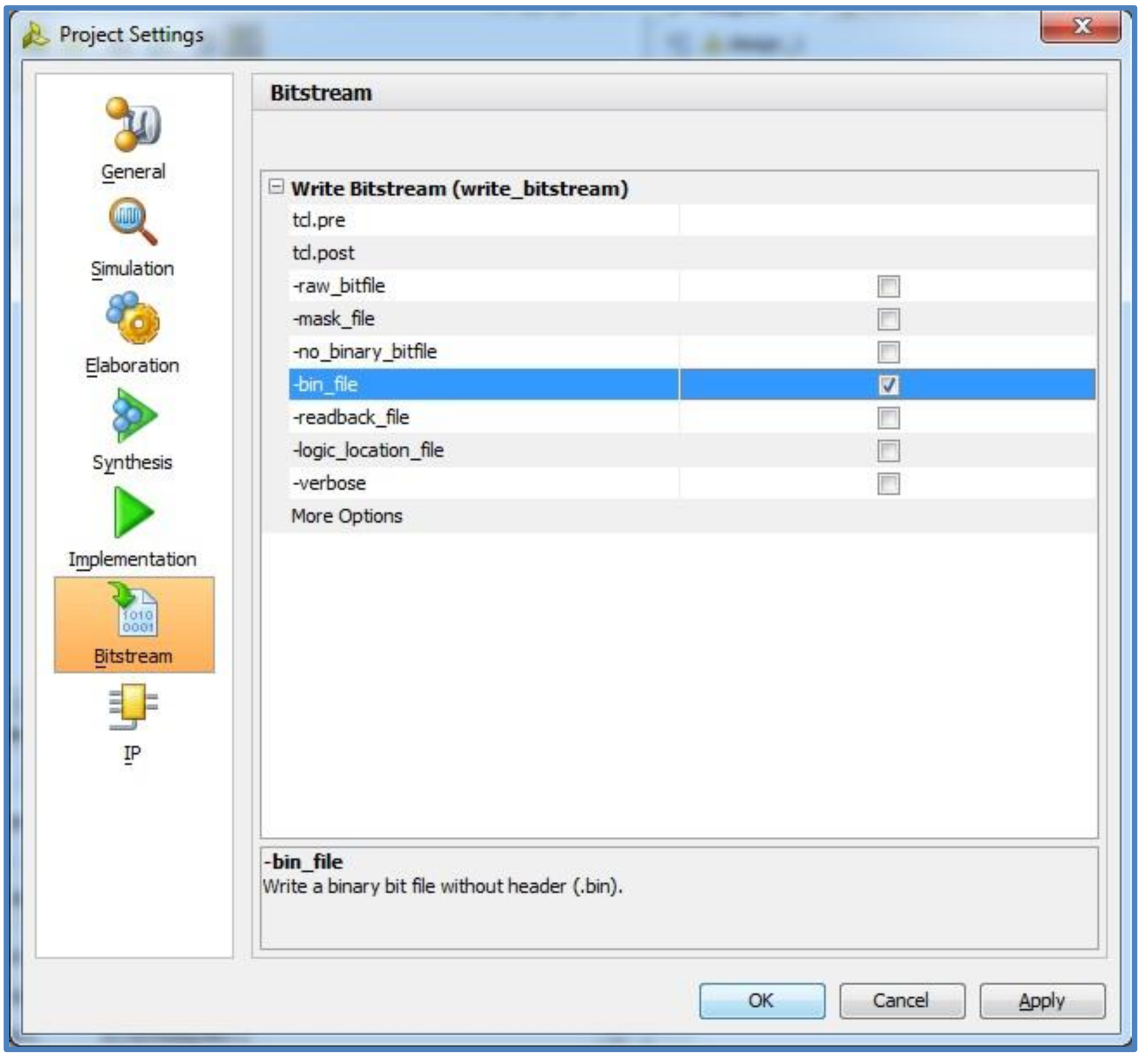
Remember that in this case is assumed that the project run in the Zynq is the FSBL itself. The PS needs to be always configured, that’s the reason why SDK assumes the first file as “bootloader”. To run any other application, there should be 3 partitions: FSBL.elf (bootloader), .bit file (PL) and .elf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

This steps are for Windows users.

1.2.4 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.



Bitstream

Write Bitstream (write_bitstream)

tcl.pre	
tcl.post	
-raw_bitfile	☐
-mask_file	☐
-no_binary_bitfile	☐
-bin_file	☑
-readback_file	☐
-logic_location_file	☐
-verbose	☐
More Options	

-bin_file
Write a binary bit file without header (.bin).

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:

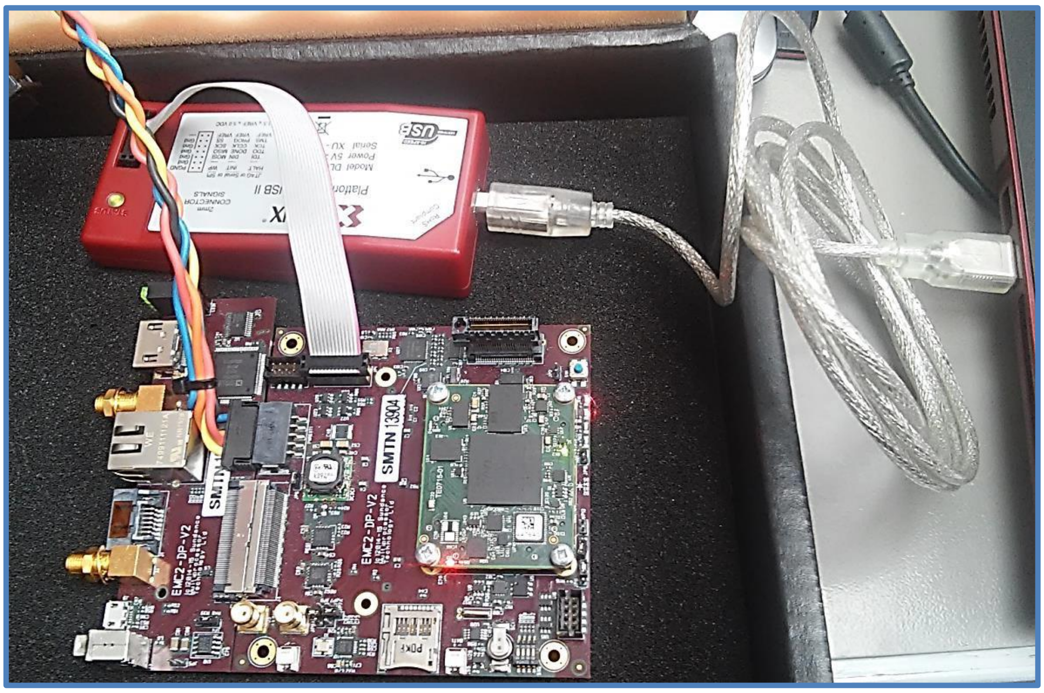
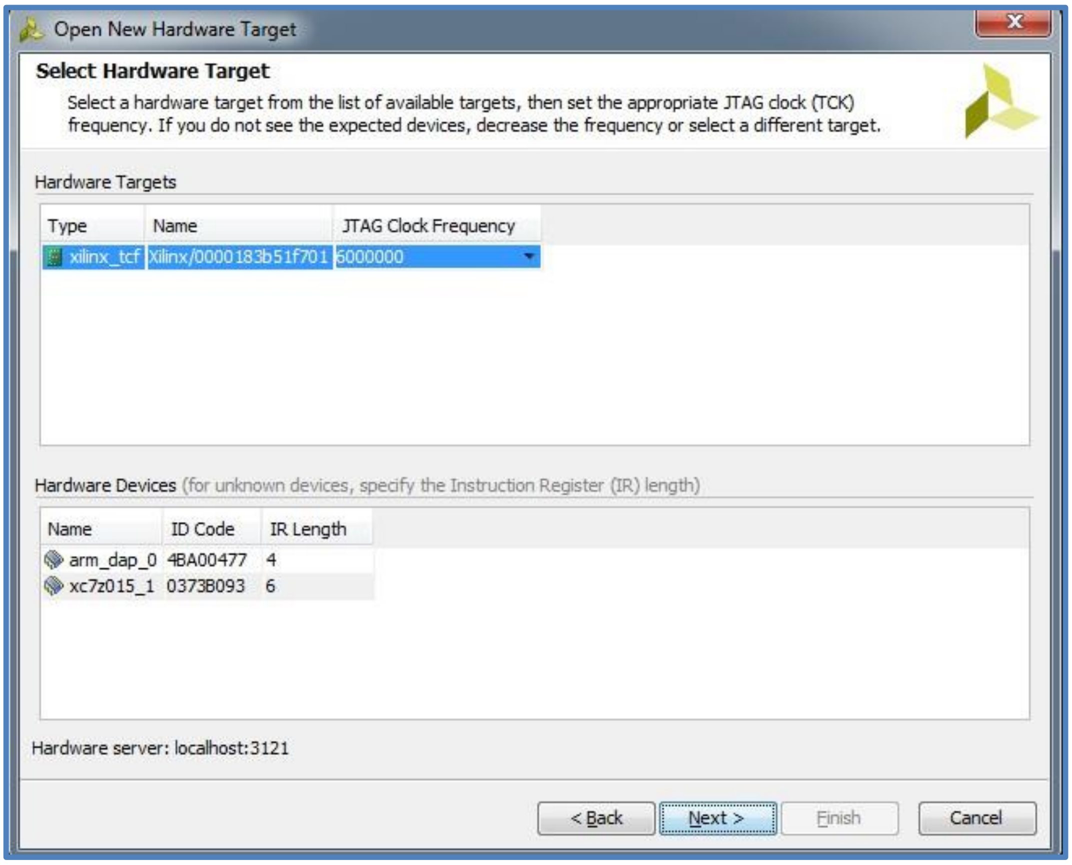


Figure 38: Set the hardware and open the Hardware Manager

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



Select Hardware Target

Select a hardware target from the list of available targets, then set the appropriate JTAG clock (TCK) frequency. If you do not see the expected devices, decrease the frequency or select a different target.

Hardware Targets

Type	Name	JTAG Clock Frequency
xilinx_tcf	Xilinx/0000183b51f701	6000000

Hardware Devices (for unknown devices, specify the Instruction Register (IR) length)

Name	ID Code	IR Length
arm_dap_0	4BA00477	4
xc7z015_1	0373B093	6

Hardware server: localhost:3121

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 Clic “Add Configuration Memory Device”

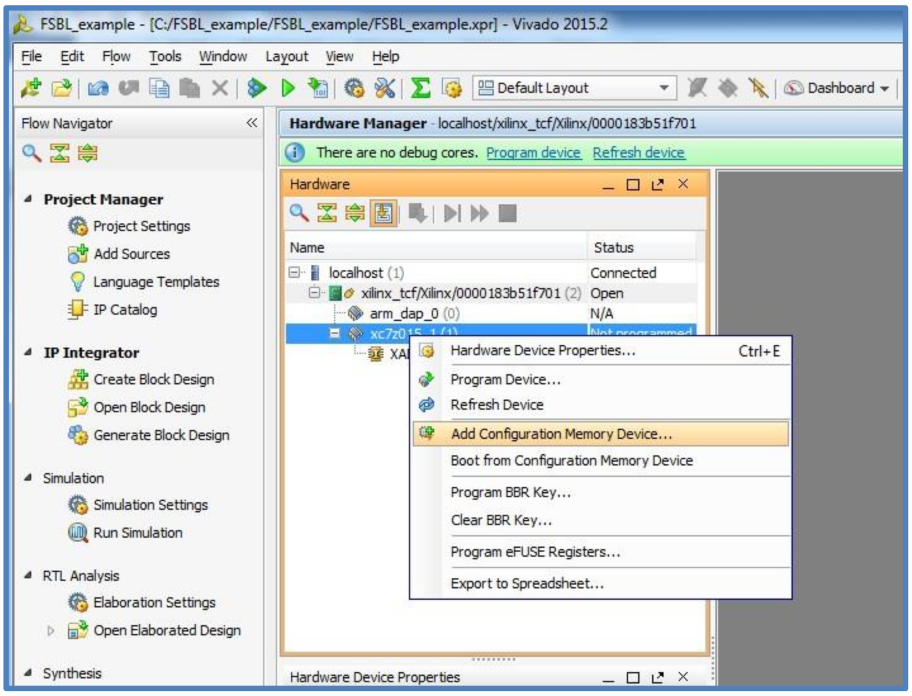
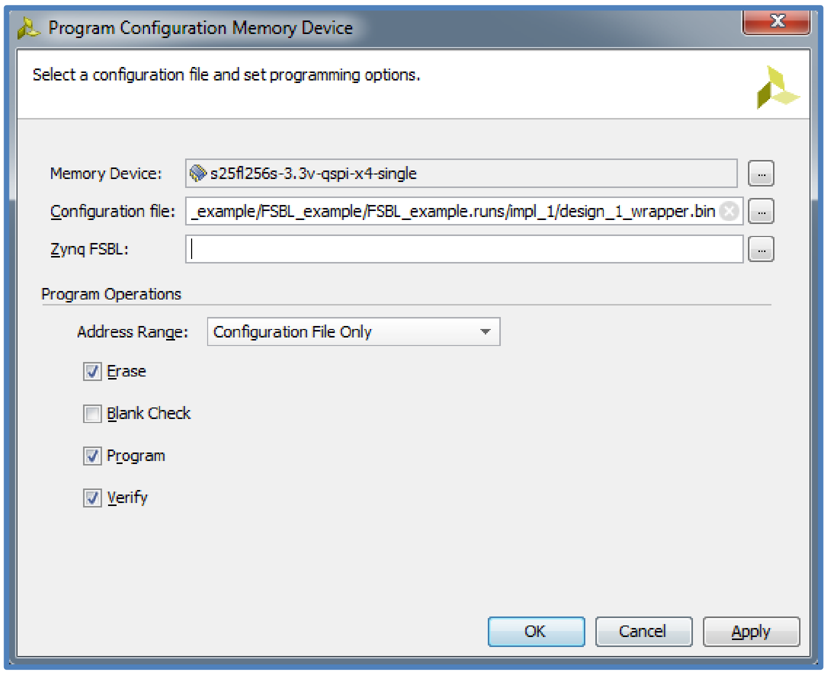


Figure 40: Add Configuration Memory Device

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


Program Configuration Memory Device

Select a configuration file and set programming options.

Memory Device: s25fl256s-3.3v-qspi-x4-single

Configuration file: _example/FSBL_example/FSBL_example.runs/impl_1/design_1_wrapper.bin

Zynq FSBL:

Program Operations

Address Range: Configuration File Only

- [✓] Erase
- [ ] Blank Check
- [✓] Program
- [✓] Verify

Figure 41: Choose the .bin file

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

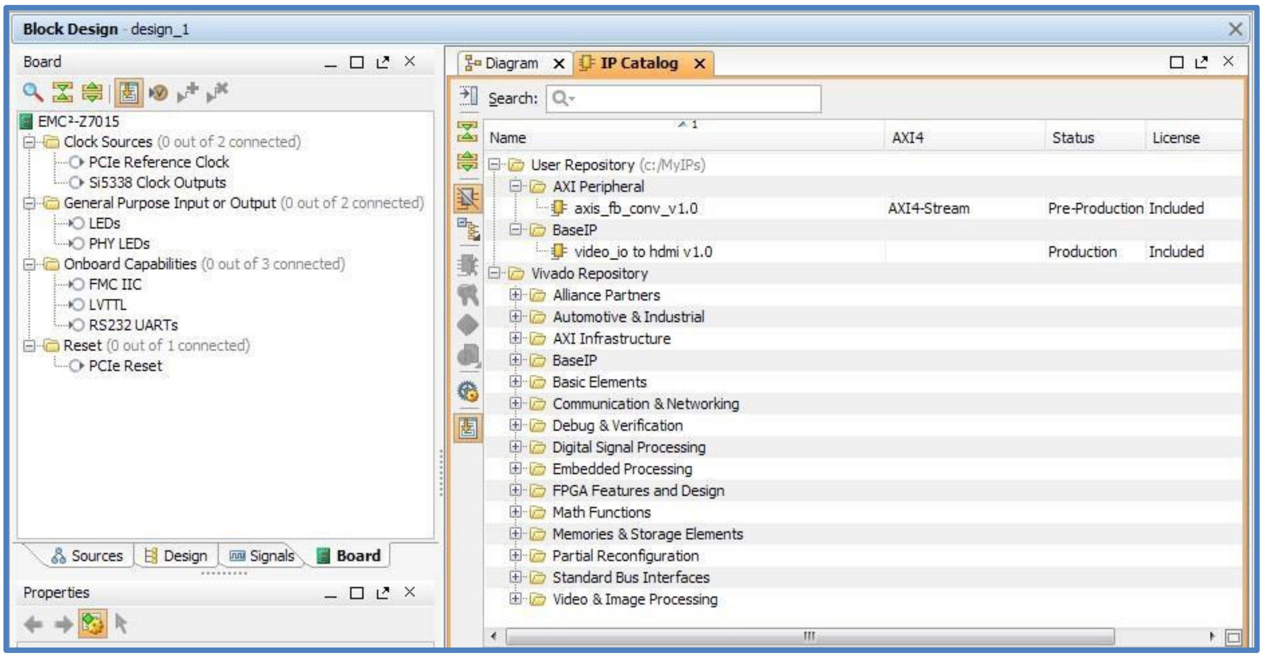
1.2.5 How can I create a HDMI test project?

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".


Block Design - design_1

Board

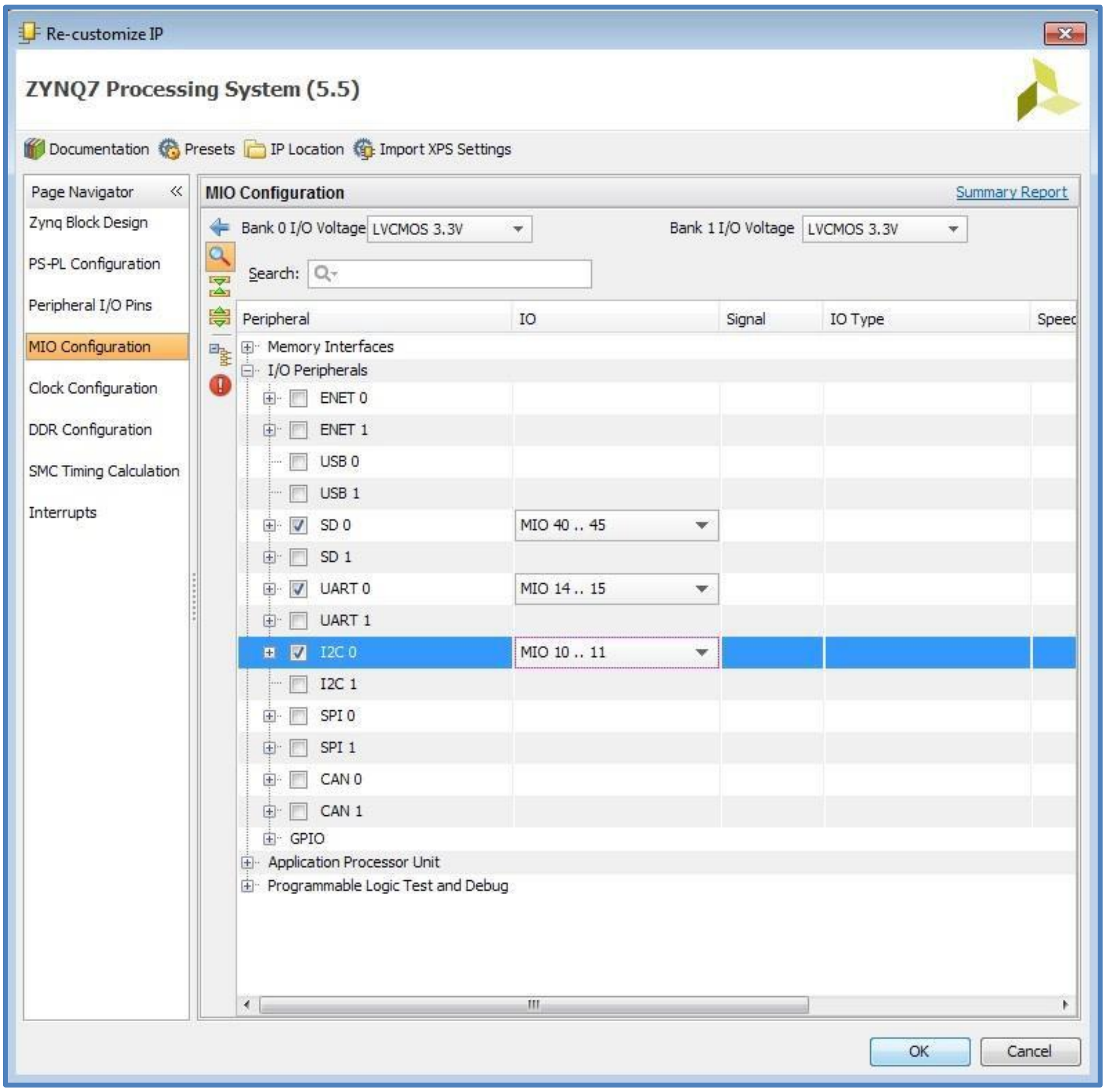
- EMC²-Z7015
  - Clock Sources (0 out of 2 connected)
    - PCIe Reference Clock
    - Si5338 Clock Outputs
  - General Purpose Input or Output (0 out of 2 connected)
    - LEDs
    - PHY LEDs
  - Onboard Capabilities (0 out of 3 connected)
    - FMC IIC
    - LVTTL
    - RS232 UARTs
  - Reset (0 out of 1 connected)
    - PCIe Reset

Name	AXI4	Status	License
User Repository (c:/MyIPs)			
AXI Peripheral			
axis_fb_conv_v1.0	AXI4-Stream	Pre-Production	Included
BaseIP			
video_io to hdmi v1.0		Production	Included
Vivado Repository			
Alliance Partners			
Automotive & Industrial			
AXI Infrastructure			
BaseIP			
Basic Elements			
Communication & Networking			
Debug & Verification			
Digital Signal Processing			
Embedded Processing			
FPGA Features and Design			
Math Functions			
Memories & Storage Elements			
Partial Reconfiguration			
Standard Bus Interfaces			
Video & Image Processing			

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.


Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation | Presets | IP Location | Import XPS Settings

Page Navigator: Zynq Block Design, PS-PL Configuration, Peripheral I/O Pins, MIO Configuration, Clock Configuration, DDR Configuration, SMC Timing Calculation, Interrupts

MIO Configuration — Summary Report

Bank 0 I/O Voltage: LVCMOS 3.3V   Bank 1 I/O Voltage: LVCMOS 3.3V

Search:

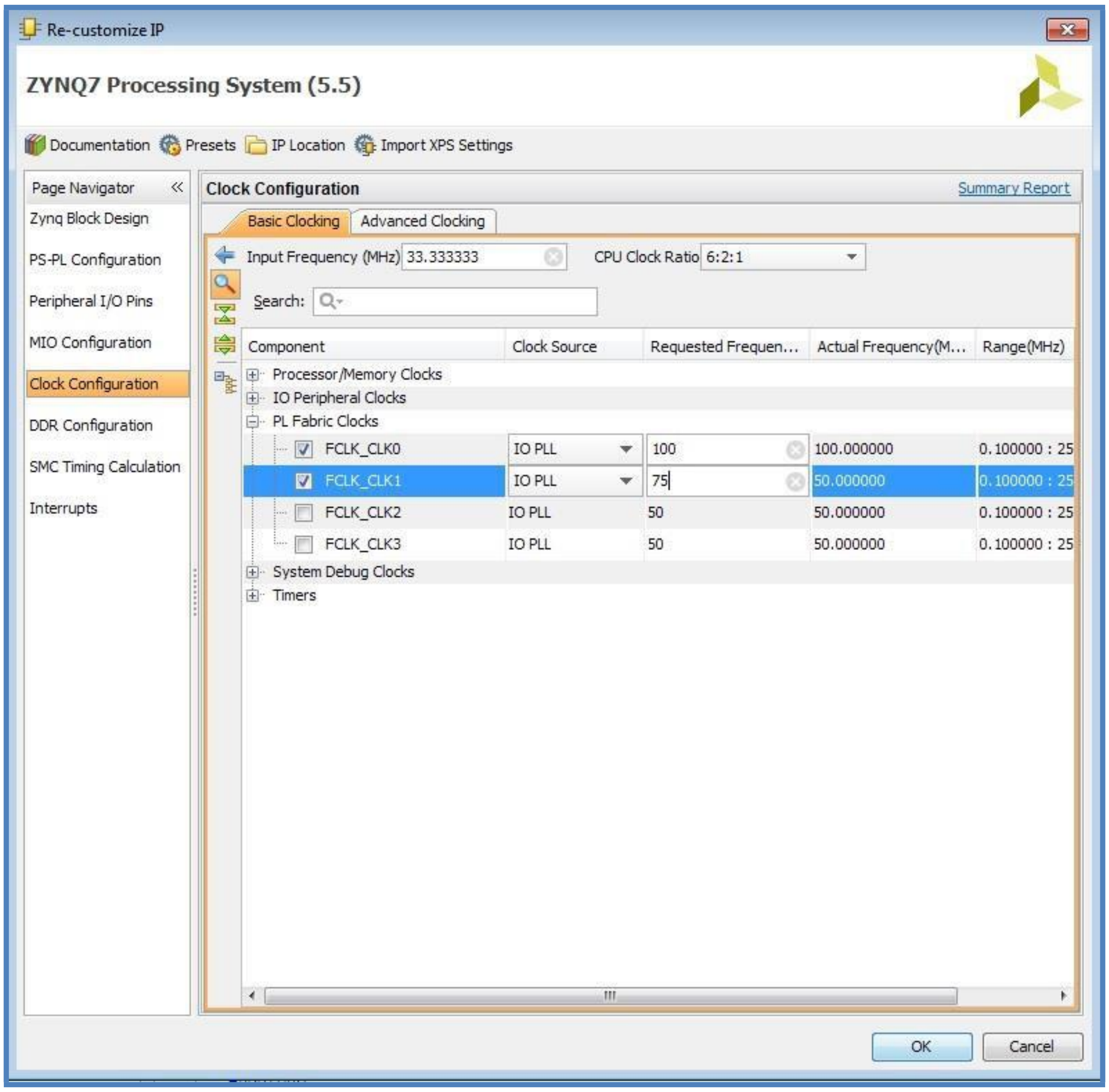
Peripheral	IO	Signal	IO Type	Speed
Memory Interfaces				
I/O Peripherals				
☐ ENET 0				
☐ ENET 1				
☐ USB 0				
☐ USB 1				
✓ SD 0	MIO 40 .. 45			
☐ SD 1				
✓ UART 0	MIO 14 .. 15			
☐ UART 1				
✓ I2C 0	MIO 10 .. 11			
☐ I2C 1				
☐ SPI 0				
☐ SPI 1				
☐ CAN 0				
☐ CAN 1				
GPIO				
Application Processor Unit				
Programmable Logic Test and Debug				

OK | Cancel

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.



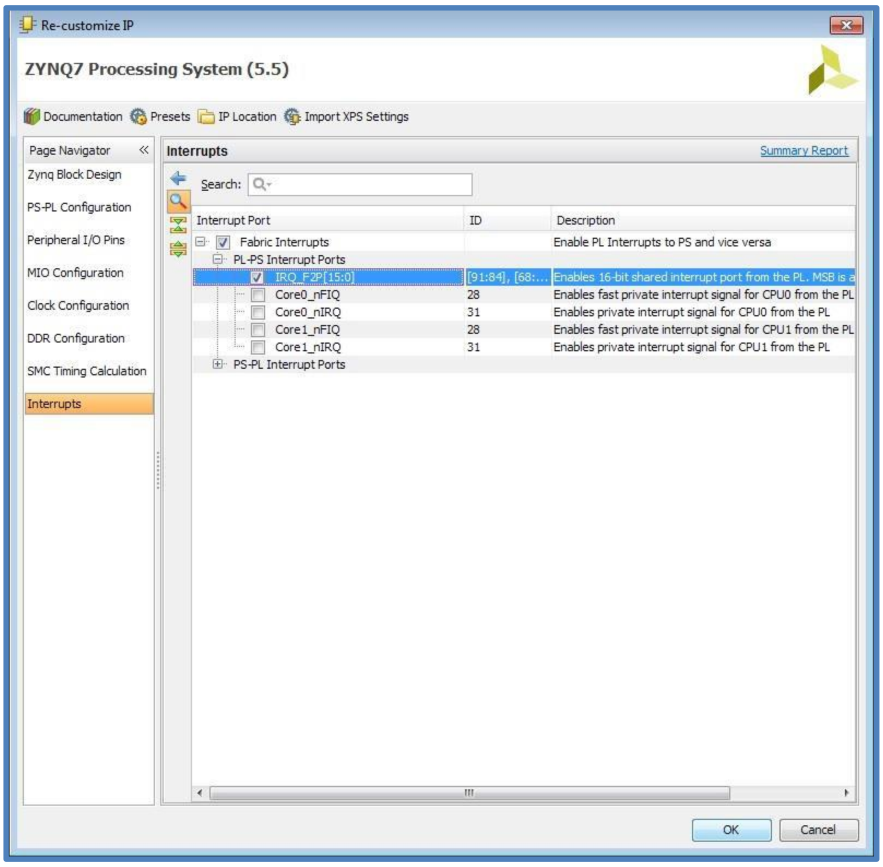
Input Frequency (MHz) 33.333333   CPU Clock Ratio 6:2:1

Search:

Component	Clock Source	Requested Frequen...	Actual Frequency(M...	Range(MHz)
Processor/Memory Clocks				
IO Peripheral Clocks				
PL Fabric Clocks				
☑ FCLK_CLK0	IO PLL	100	100.000000	0.100000 : 25
☑ FCLK_CLK1	IO PLL	75	50.000000	0.100000 : 25
☐ FCLK_CLK2	IO PLL	50	50.000000	0.100000 : 25
☐ FCLK_CLK3	IO PLL	50	50.000000	0.100000 : 25
System Debug Clocks				
Timers				

Figure 44: Configuring clocks

Activate the interrupts as well.



Interrupts

Interrupt Port	ID	Description
☑ Fabric Interrupts		Enable PL Interrupts to PS and vice versa
PL-PS Interrupt Ports		
☑ IRQ_F2P[15:0]	[91:84], [68:...	Enables 16-bit shared interrupt port from the PL. MSB is a
☐ Core0_nFIQ	28	Enables fast private interrupt signal for CPU0 from the PL
☐ Core0_nIRQ	31	Enables private interrupt signal for CPU0 from the PL
☐ Core1_nFIQ	28	Enables fast private interrupt signal for CPU1 from the PL
☐ Core1_nIRQ	31	Enables private interrupt signal for CPU1 from the PL
PS-PL Interrupt Ports		

Figure 45: Interrupts

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

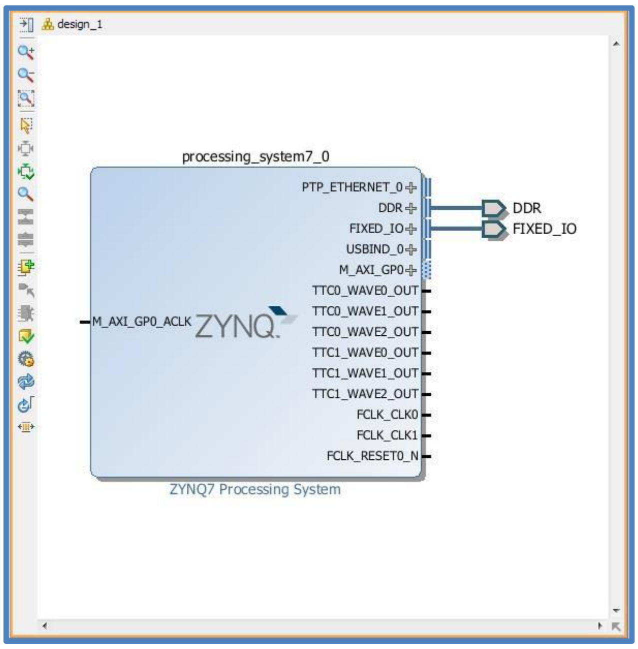


Figure 46: Run Automation

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

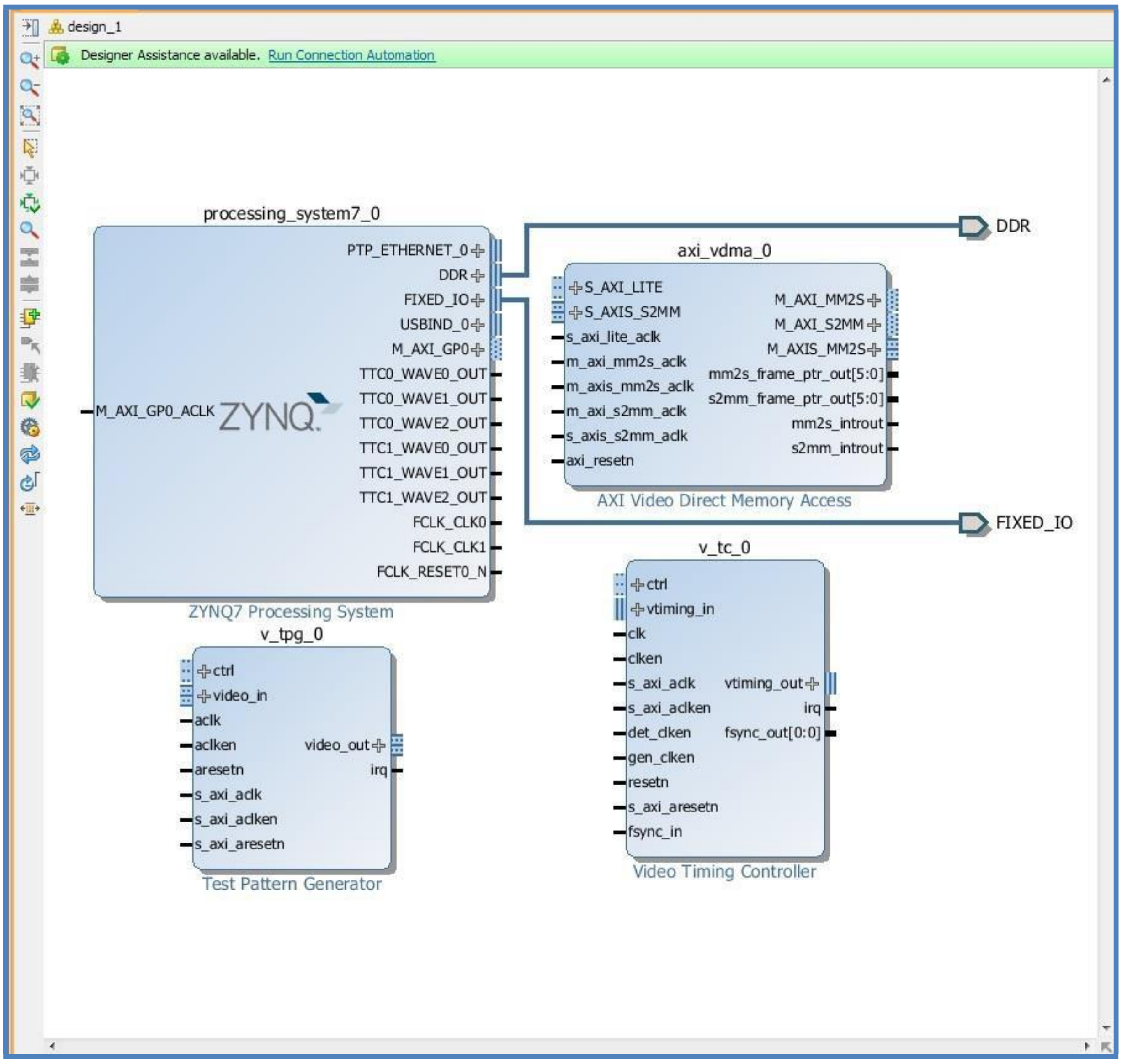
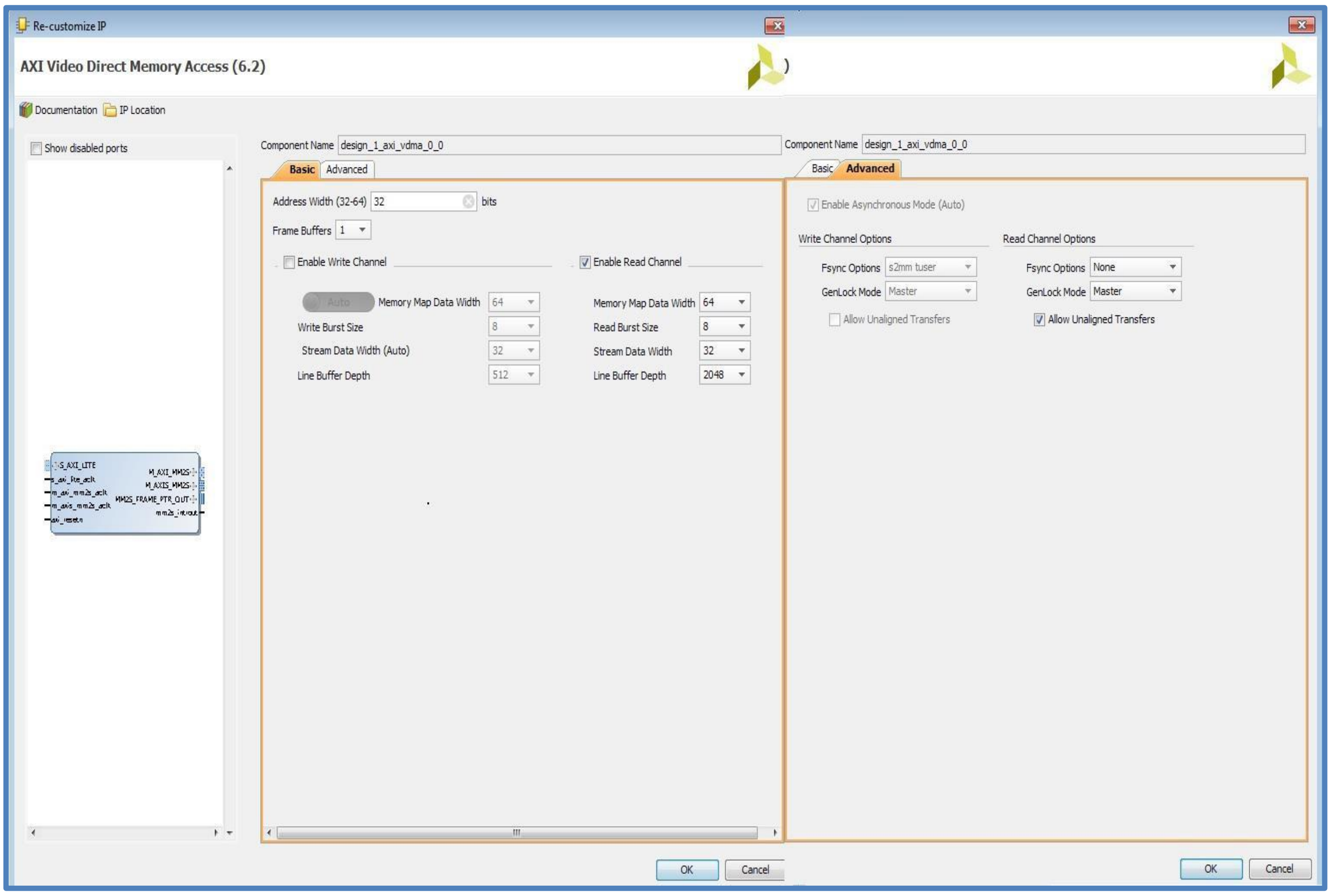


Figure 47: Adding blocks

Configure the Video Direct Memory Access block as follows.



Re-customize IP

AXI Video Direct Memory Access (6.2)

Documentation IP Location

Show disabled ports

Component Name design_1_axi_vdma_0_0

Basic | Advanced

Address Width (32-64) 32 bits

Frame Buffers 1

Enable Write Channel

Auto Memory Map Data Width 64

Write Burst Size 8

Stream Data Width (Auto) 32

Line Buffer Depth 512

Enable Read Channel ✓

Memory Map Data Width 64

Read Burst Size 8

Stream Data Width 32

Line Buffer Depth 2048

Component Name design_1_axi_vdma_0_0

Basic | Advanced

Enable Asynchronous Mode (Auto) ✓

Write Channel Options

Fsync Options s2mm tuser

GenLock Mode Master

Allow Unaligned Transfers

Read Channel Options

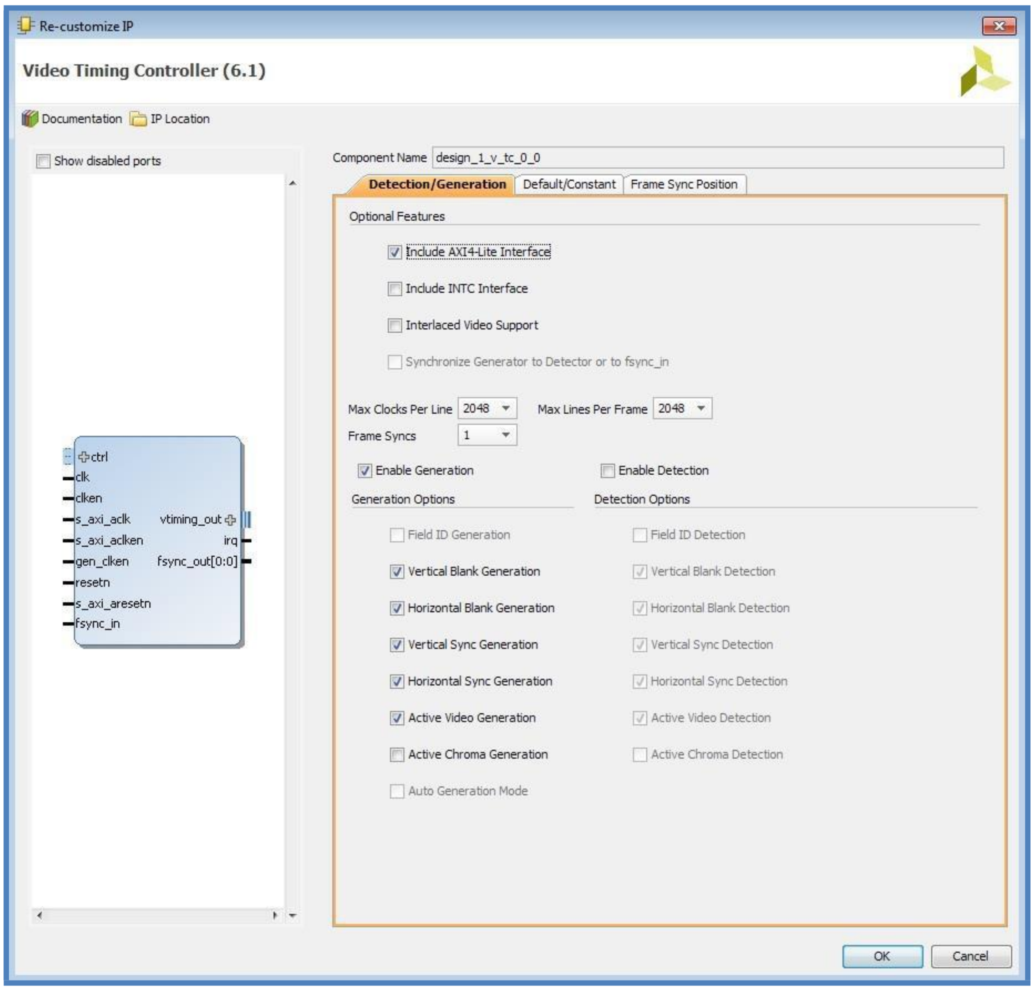
Fsync Options None

GenLock Mode Master

Allow Unaligned Transfers ✓

Figure 48: Configuring DMA

The Video Timing Controller as follows.


Re-customize IP

Video Timing Controller (6.1)

Documentation IP Location

[ ] Show disabled ports

Component Name: design_1_v_tc_0_0

Detection/Generation | Default/Constant | Frame Sync Position

Optional Features

[✓] Include AXI4-Lite Interface

[ ] Include INTC Interface

[ ] Interlaced Video Support

[ ] Synchronize Generator to Detector or to fsync_in

Max Clocks Per Line: 2048   Max Lines Per Frame: 2048

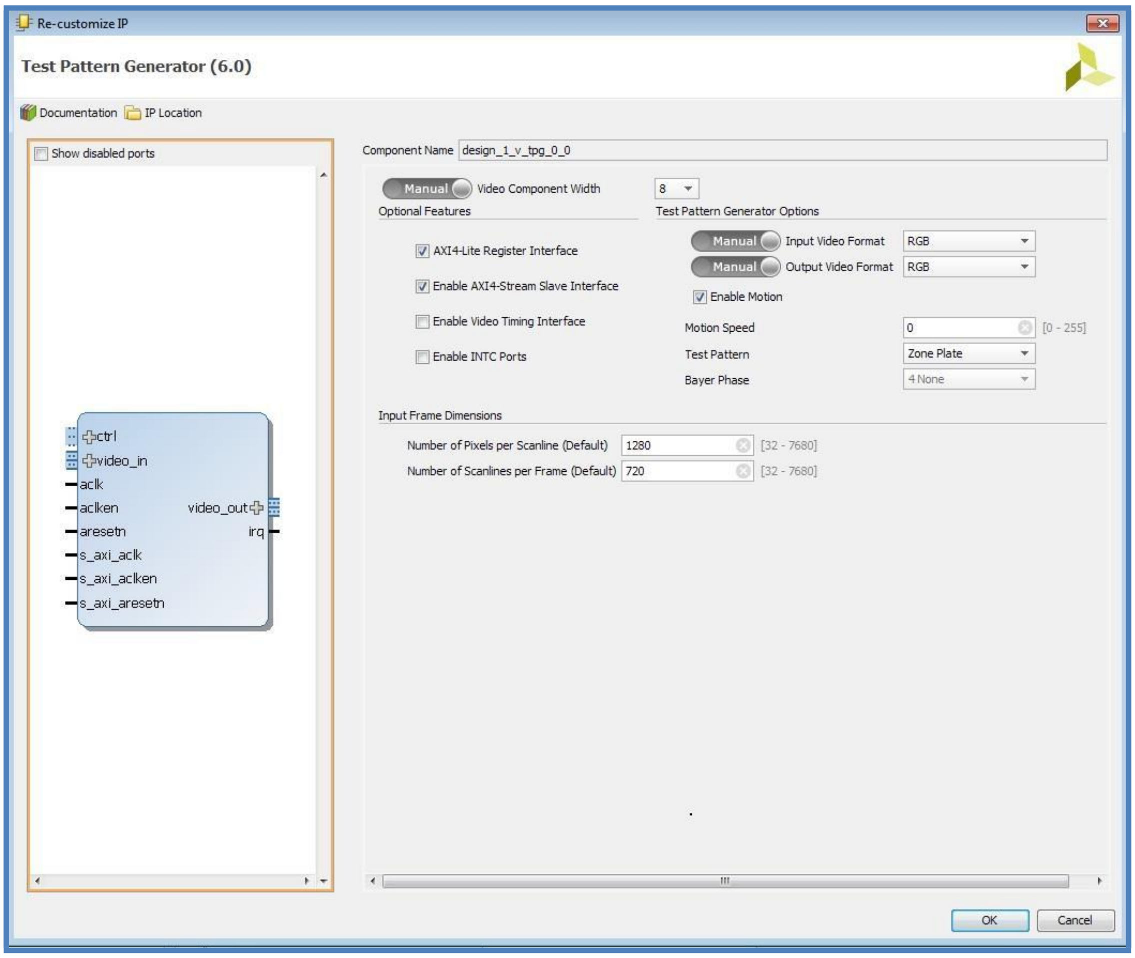
Frame Syncs: 1

[✓] Enable Generation   [ ] Enable Detection

Generation Options	Detection Options
[ ] Field ID Generation	[ ] Field ID Detection
[✓] Vertical Blank Generation	[✓] Vertical Blank Detection
[✓] Horizontal Blank Generation	[✓] Horizontal Blank Detection
[✓] Vertical Sync Generation	[✓] Vertical Sync Detection
[✓] Horizontal Sync Generation	[✓] Horizontal Sync Detection
[✓] Active Video Generation	[✓] Active Video Detection
[ ] Active Chroma Generation	[ ] Active Chroma Detection
[ ] Auto Generation Mode	

Figure 49: Configuring Video Timing Controller

The Test Pattern Generator as follows.


Re-customize IP

Test Pattern Generator (6.0)

Documentation IP Location

Show disabled ports

Component Name: design_1_v_tpg_0_0

Manual — Video Component Width: 8

Optional Features

- [✓] AXI4-Lite Register Interface
- [✓] Enable AXI4-Stream Slave Interface
- [ ] Enable Video Timing Interface
- [ ] Enable INTC Ports

Test Pattern Generator Options

- Manual — Input Video Format: RGB
- Manual — Output Video Format: RGB
- [✓] Enable Motion
- Motion Speed: 0 [0 - 255]
- Test Pattern: Zone Plate
- Bayer Phase: 4 None

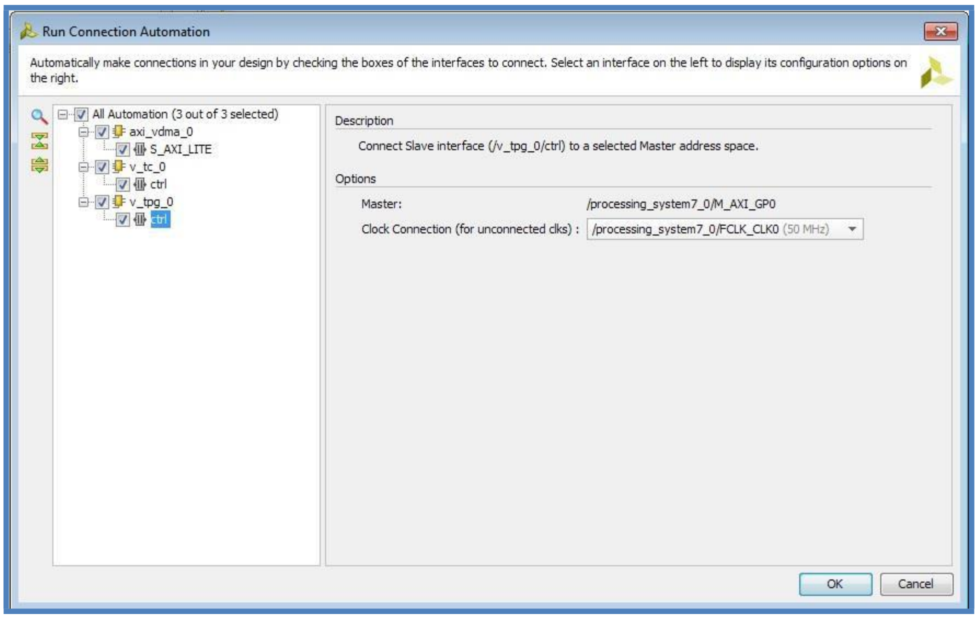
Input Frame Dimensions

- Number of Pixels per Scanline (Default): 1280 [32 - 7680]
- Number of Scanlines per Frame (Default): 720 [32 - 7680]

OK Cancel

Figure 50: Configuring Test Pattern Generator

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


Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

- [✓] All Automation (3 out of 3 selected)
  - [✓] axi_vdma_0
    - [✓] S_AXI_LITE
  - [✓] v_tc_0
    - [✓] ctrl
  - [✓] v_tpg_0
    - [✓] ctrl

Description

Connect Slave interface (/v_tpg_0/ctrl) to a selected Master address space.

Options

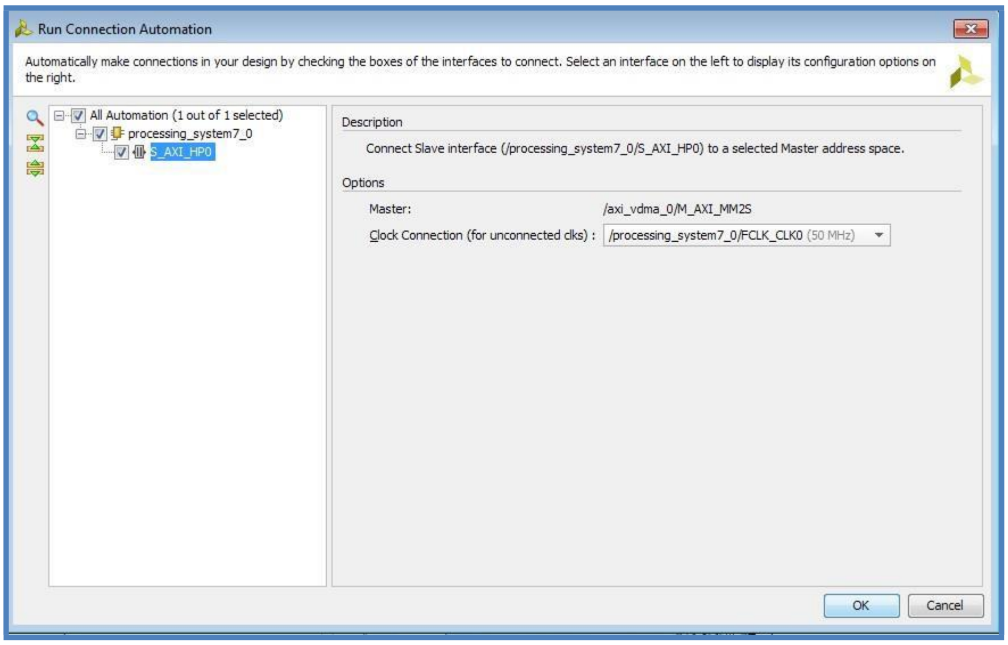
Master: /processing_system7_0/M_AXI_GP0

Clock Connection (for unconnected clks) : /processing_system7_0/FCLK_CLK0 (50 MHz)

OK Cancel

Figure 51: Assigning clocks

Select the Zynq as Master of the system.



Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

- [✓] All Automation (1 out of 1 selected)
  - [✓] processing_system7_0
    - [✓] S_AXI_HP0

Description

Connect Slave interface (/processing_system7_0/S_AXI_HP0) to a selected Master address space.

Options

Master: /axi_vdma_0/M_AXI_MM2S

Clock Connection (for unconnected clks) : /processing_system7_0/FCLK_CLK0 (50 MHz)

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.

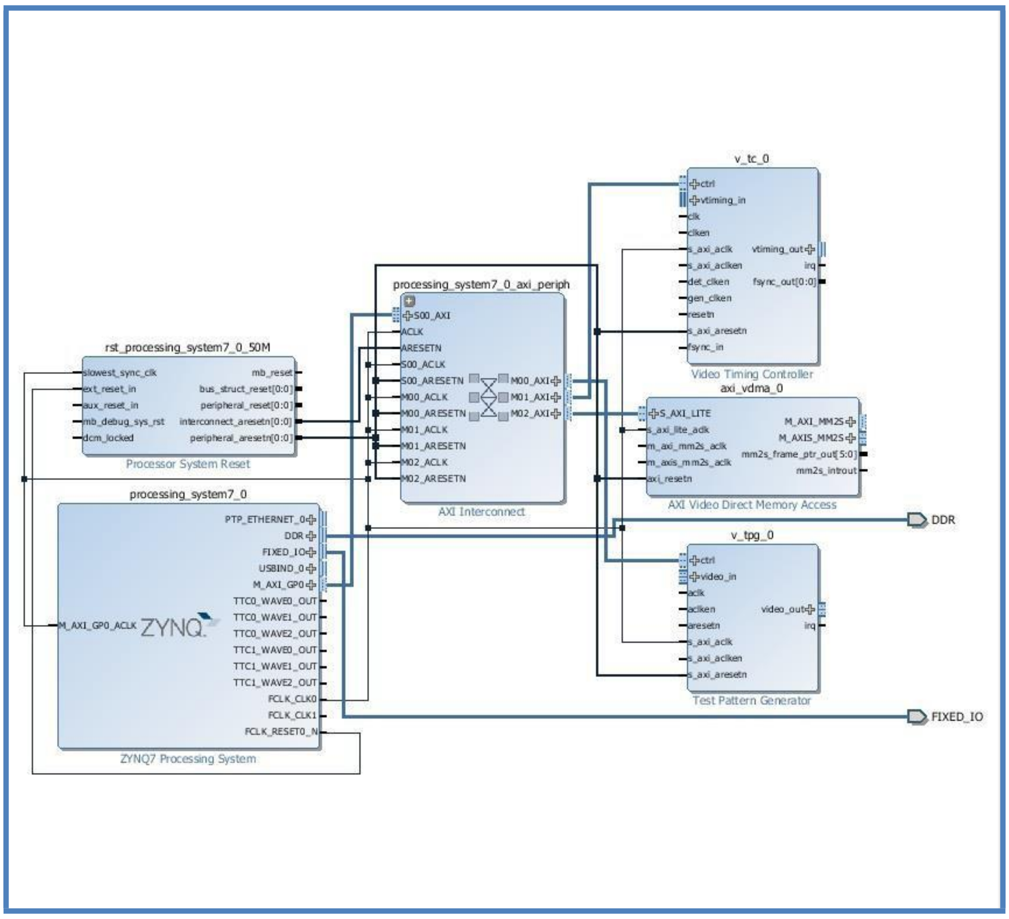
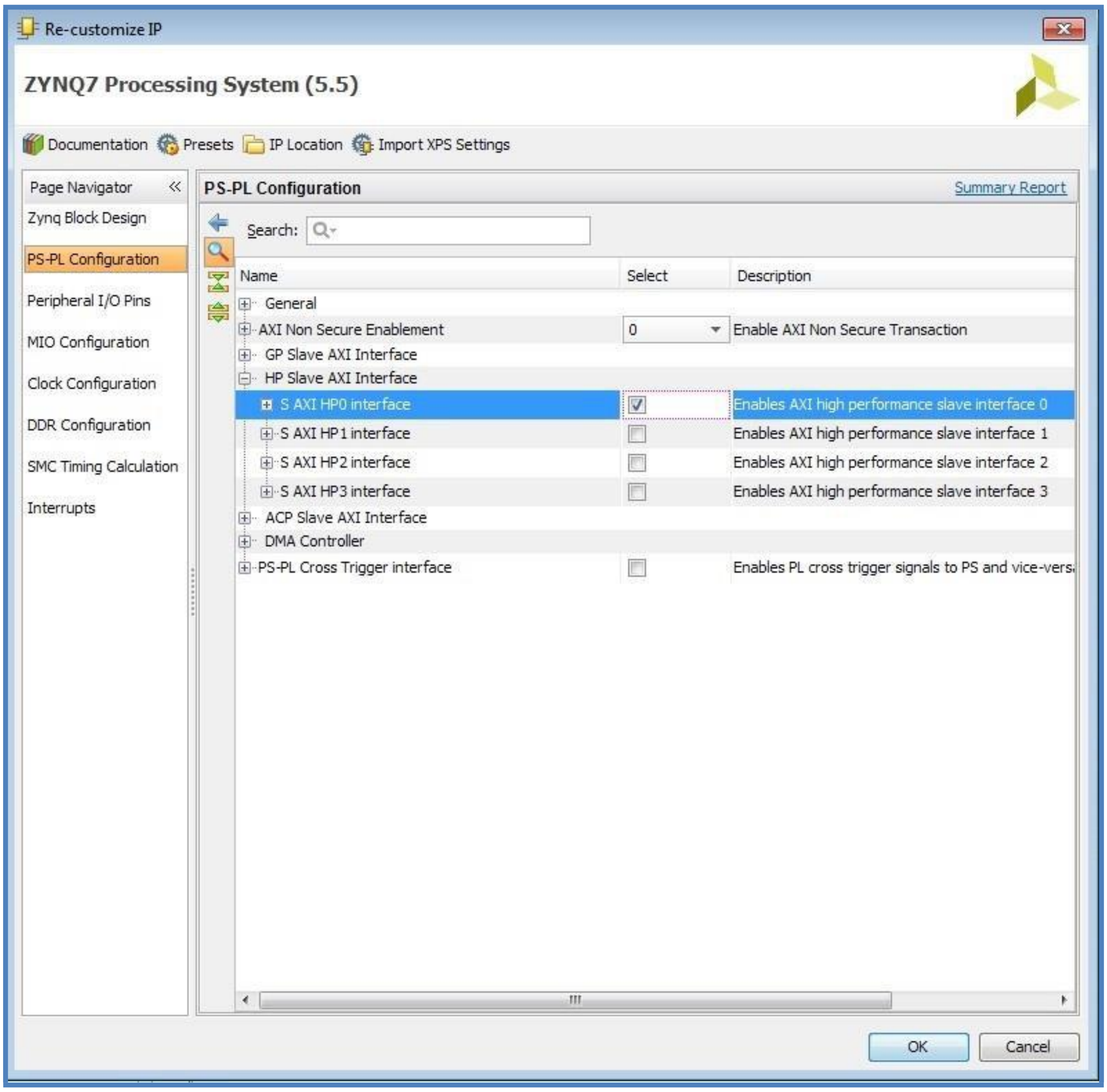


Figure 53: First look

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



PS-PL Configuration

Name	Select	Description
General		
AXI Non Secure Enablement	0	Enable AXI Non Secure Transaction
GP Slave AXI Interface		
HP Slave AXI Interface		
S AXI HP0 interface	✓	Enables AXI high performance slave interface 0
S AXI HP1 interface	☐	Enables AXI high performance slave interface 1
S AXI HP2 interface	☐	Enables AXI high performance slave interface 2
S AXI HP3 interface	☐	Enables AXI high performance slave interface 3
ACP Slave AXI Interface		
DMA Controller		
PS-PL Cross Trigger interface	☐	Enables PL cross trigger signals to PS and vice-vers

Figure 54: Adding a slave port

And the system looks like this after this last step.

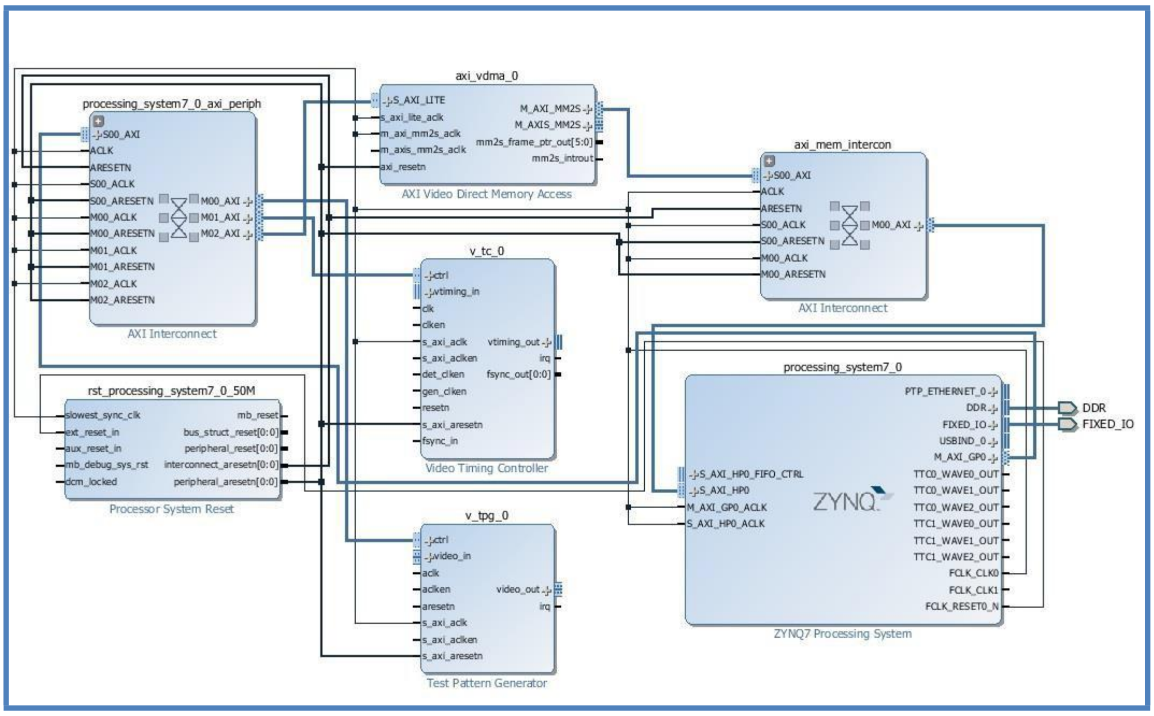


Figure 55: Second look

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

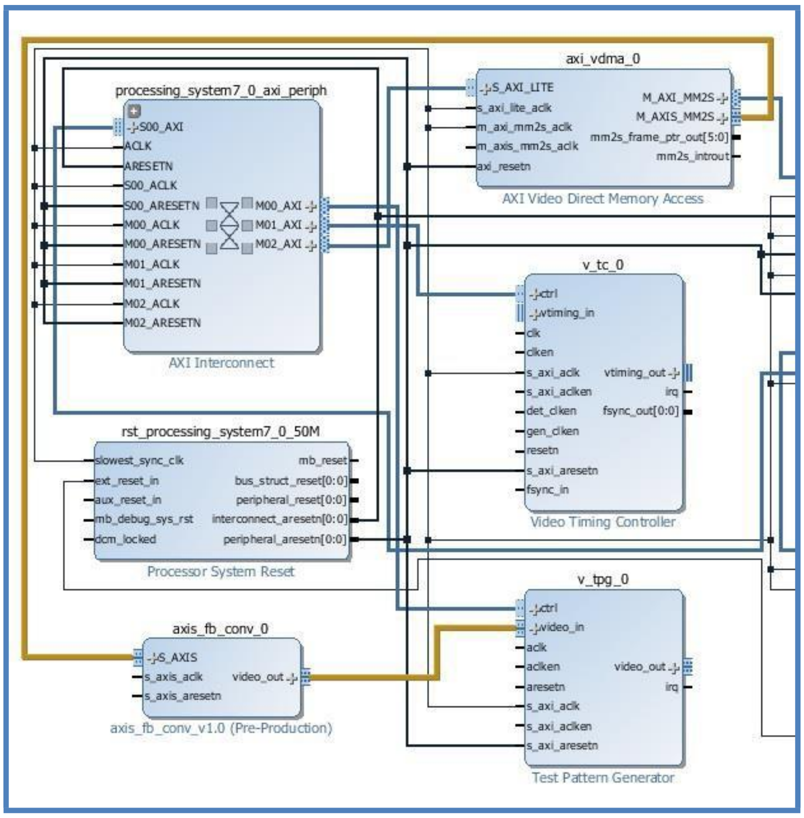


Figure 56: Making connections

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.

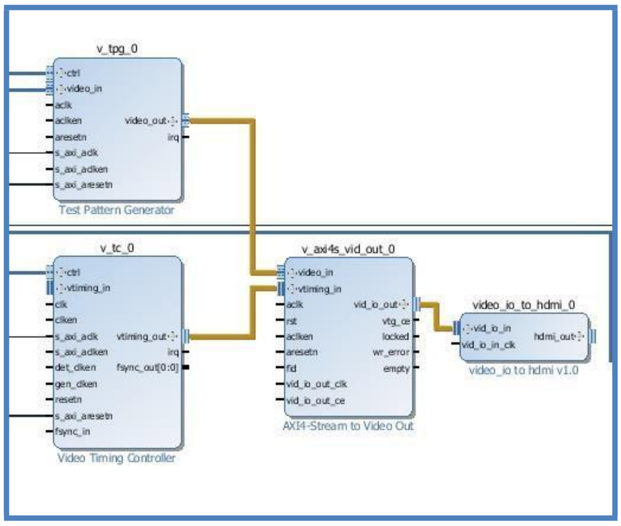


Figure 57: HDMI Interface blocks

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

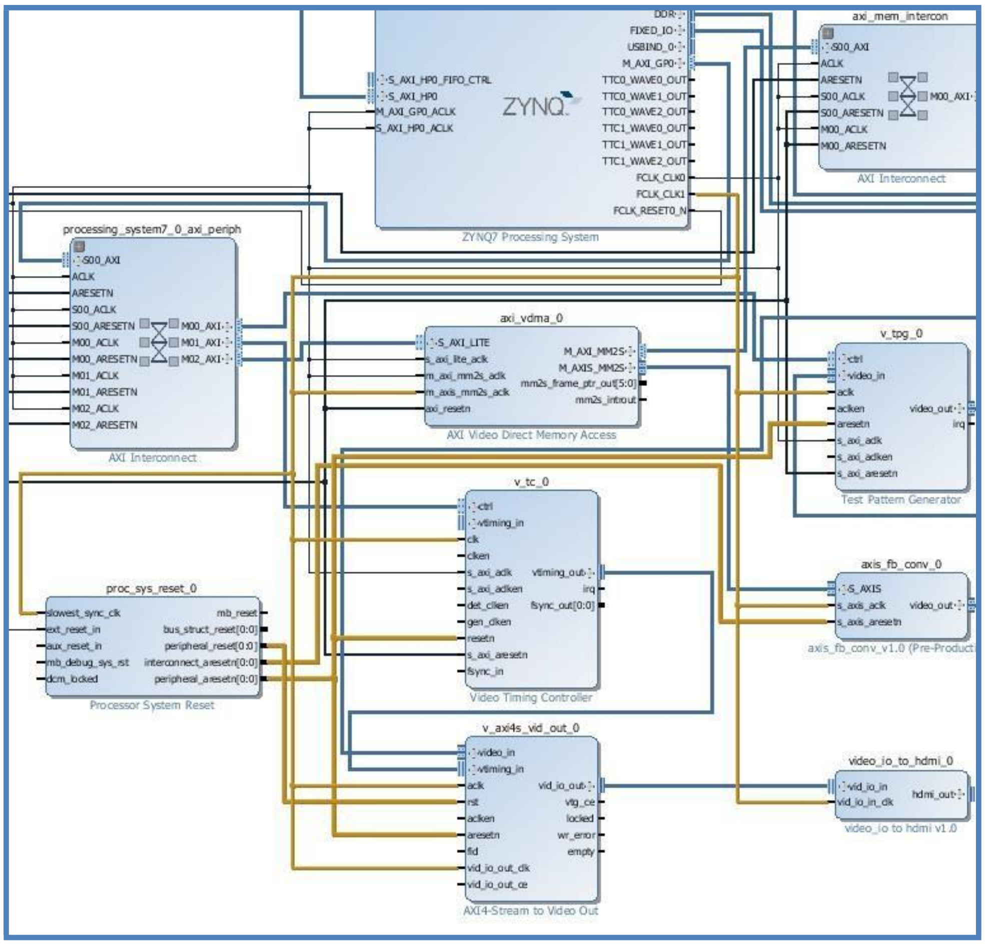


Figure 58: Clock connections

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

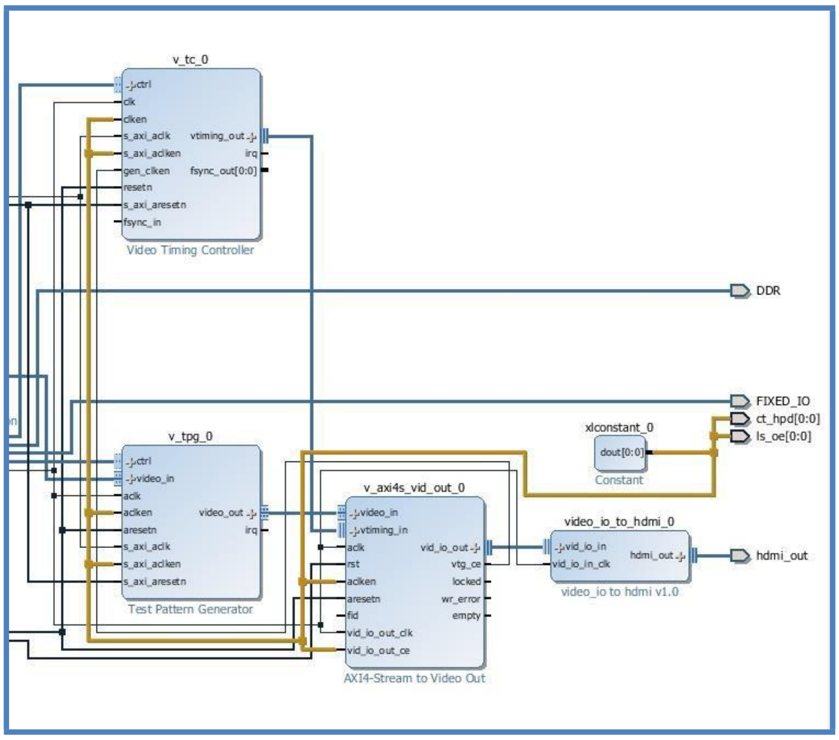


Figure 59: Adding ports

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

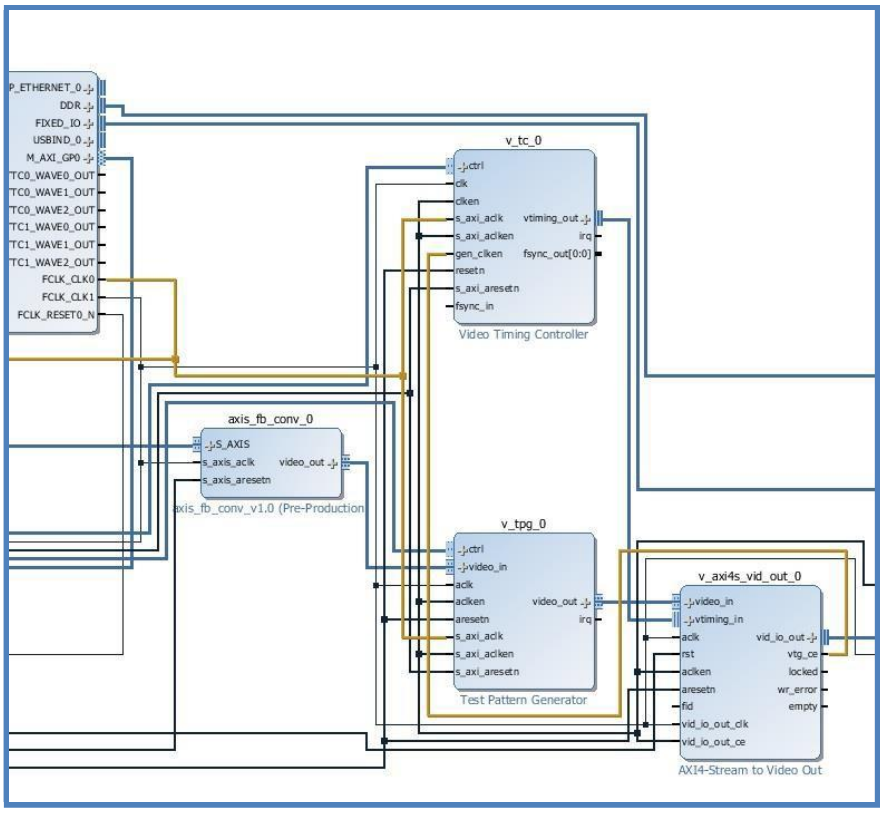


Figure 60: Connecting the rest of the clocks

Configure the interrupts for the Zynq

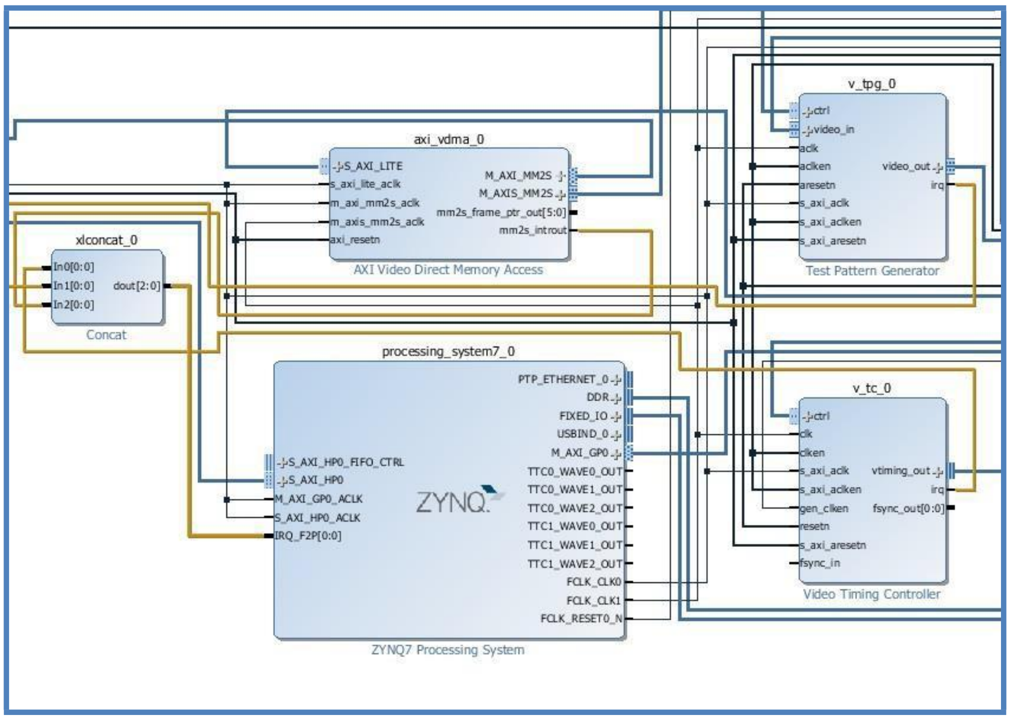


Figure 61: Connecting interrupts

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

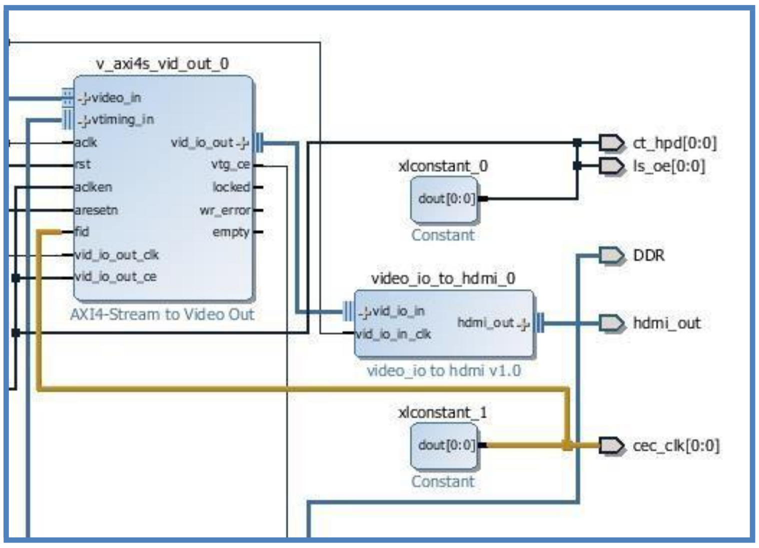


Figure 62: Adding port

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.

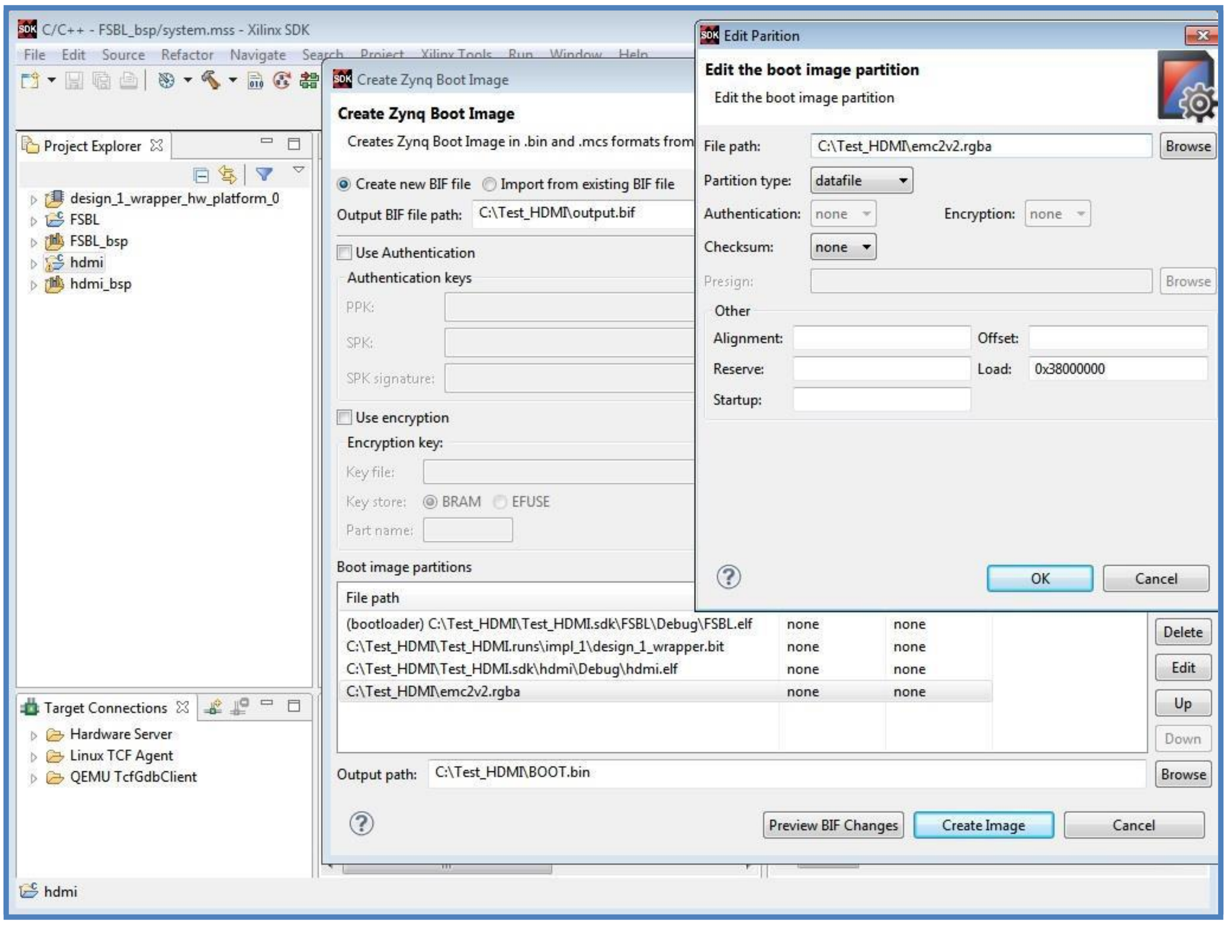
To create the boot file, 4 files are needed:

-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.



Create Zynq Boot Image

Creates Zynq Boot Image in .bin and .mcs formats from

(●) Create new BIF file ( ) Import from existing BIF file

Output BIF file path: C:\Test_HDMI\output.bif

[ ] Use Authentication

Authentication keys

PPK:

SPK:

SPK signature:

[ ] Use encryption

Encryption key:

Key file:

Key store: (●) BRAM ( ) EFUSE

Part name:

Boot image partitions

File path		
(bootloader) C:\Test_HDMI\Test_HDMI.sdk\FSBL\Debug\FSBL.elf	none	none
C:\Test_HDMI\Test_HDMI.runs\impl_1\design_1_wrapper.bit	none	none
C:\Test_HDMI\Test_HDMI.sdk\hdmi\Debug\hdmi.elf	none	none
C:\Test_HDMI\emc2v2.rgba	none	none

Output path: C:\Test_HDMI\BOOT.bin

Edit the boot image partition

Edit the boot image partition

File path: C:\Test_HDMI\emc2v2.rgba

Partition type: datafile

Authentication: none Encryption: none

Checksum: none

Presign:

Other

Alignment: Offset:

Reserve: Load: 0x38000000

Startup:

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.

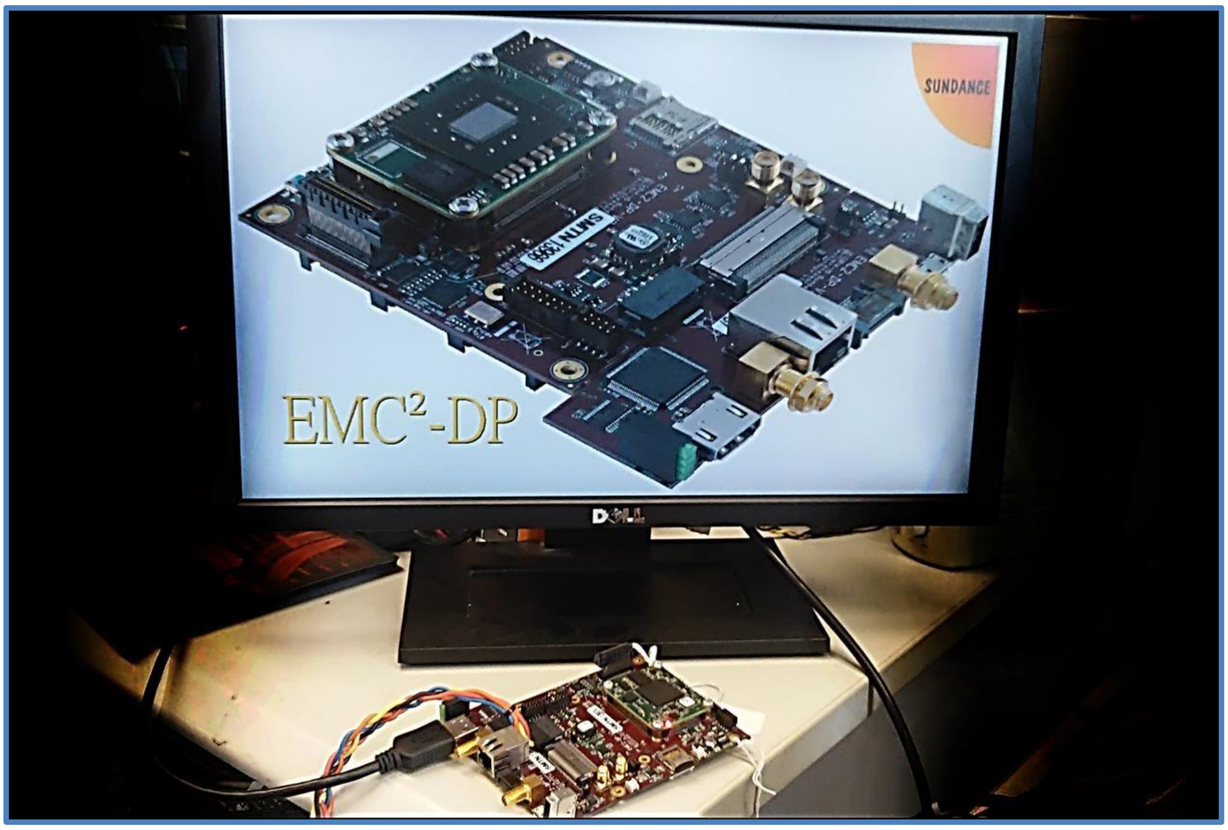


Figure 64: Test running