Creating a base Zynq design with Vivado IPI 2013.2

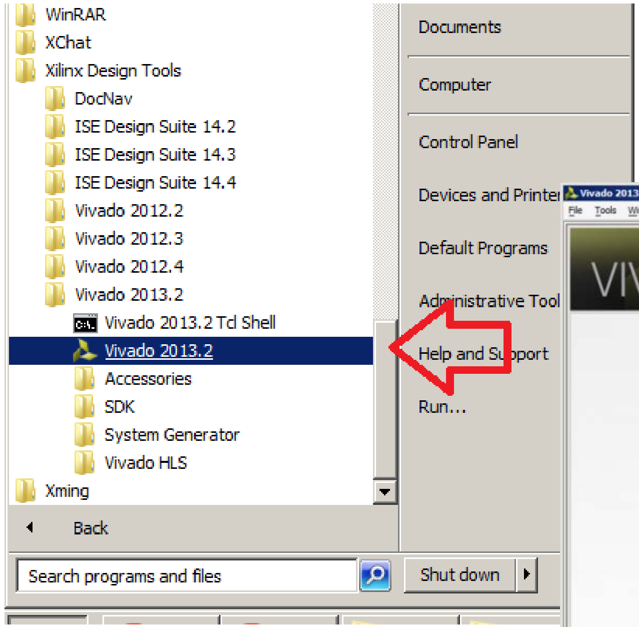
based on:

http://www.zedboard.org/zh-hant/node/1454

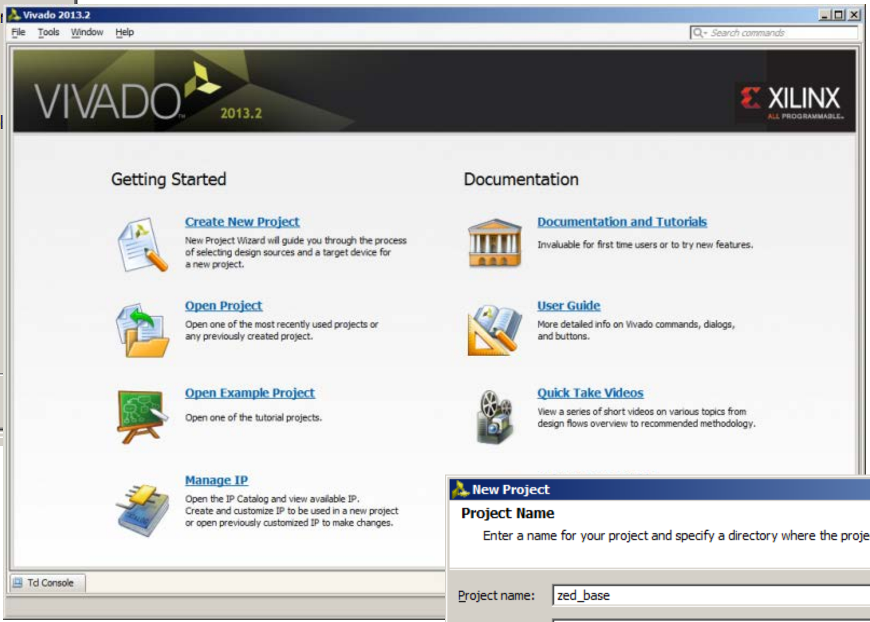
http://xillybus.com/tutorials/vivado-hls-c-fpga-howto-1

Dr. Heinz Rongen
Forschungszentrum Jülich GmbH
Zentralinstitut Systeme der Elektronik (ZEA-2)
H.Rongen@fz-juelich.de

Start Vivado / New Project



1) Start Vivado



Getting Started

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

Open Project
Open one of the most recently used projects or any previously created project.

Open Example Project
Open one of the tutorial projects.

Manage IP
Open the IP Catalog and view available IP. Create and customize IP to be used in a new project or open previously customized IP to make changes.

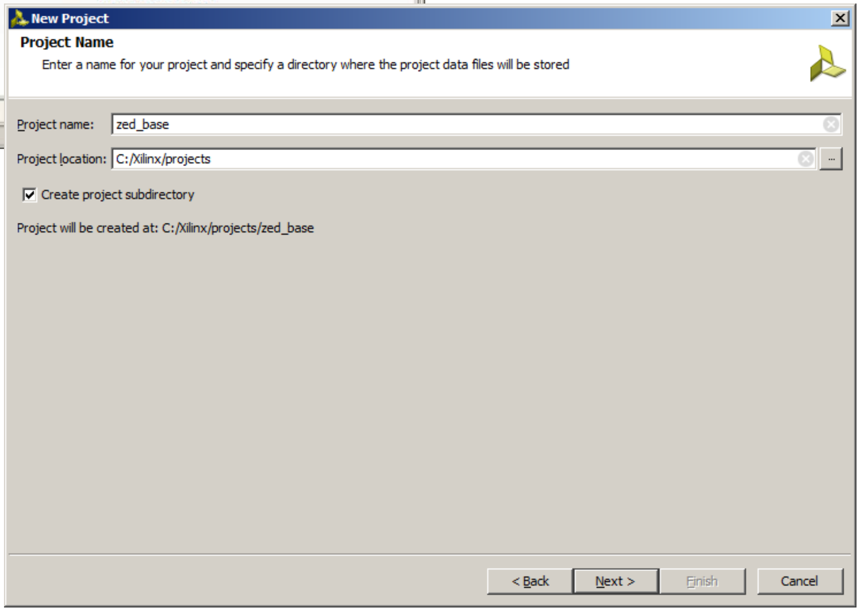
Documentation

Documentation and Tutorials
Invaluable for first time users or to try new features.

User Guide
More detailed info on Vivado commands, dialogs, and buttons.

Quick Take Videos
View a series of short videos on various topics from design flows overview to recommended methodology.

2) Create New Project



Project Name

Enter a name for your project and specify a directory where the project data files will be stored

Project name: zed_base

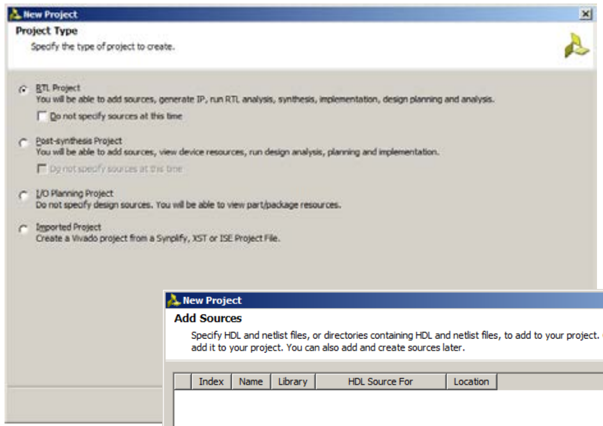
Project location: C:/Xilinx/projects

[✓] Create project subdirectory

Project will be created at: C:/Xilinx/projects/zed_base

3) Project Name and location

New Project



New Project

Project Type

Specify the type of project to create.

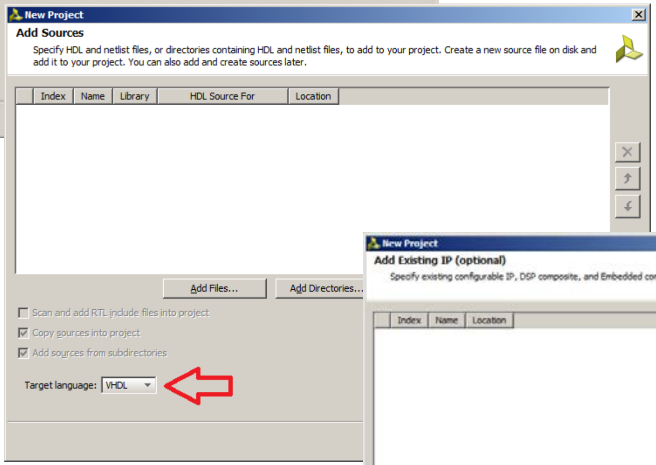
[✓ RTL Project]
You will be able to add sources, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☐ Do not specify sources at this time

Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

Create a RTL project for your base Zynq design.



New Project

Add Sources

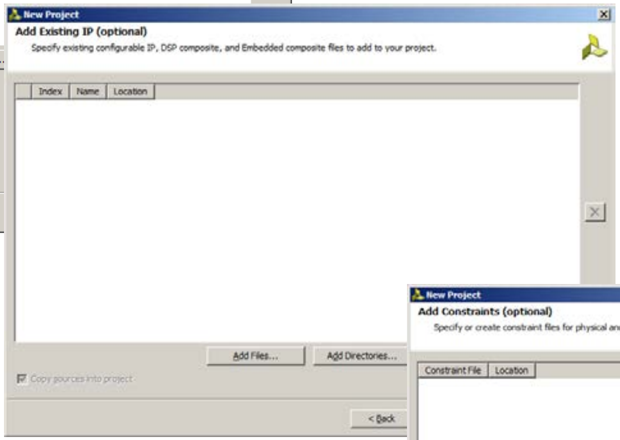
Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

	Index	Name	Library	HDL Source For	Location

☐ Scan and add RTL include files into project
☑ Copy sources into project
☑ Add sources from subdirectories

Target language: VHDL

- no sources to add
- I prefer VHDL



New Project

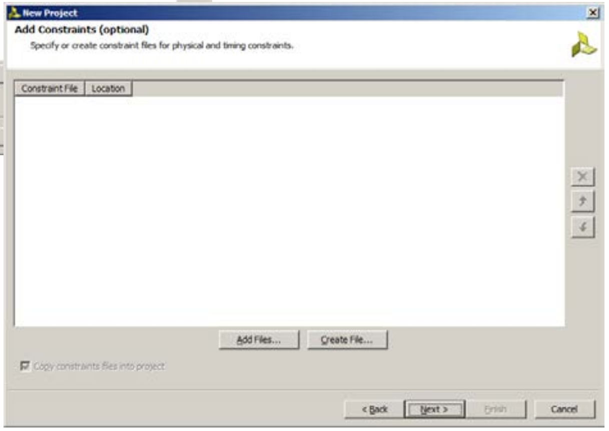
Add Existing IP (optional)

Specify existing configurable IP, DSP composite, and Embedded composite files to add to your project.

	Index	Name	Location

☑ Copy sources into project

- no IP to add



New Project

Add Constraints (optional)

Specify or create constraint files for physical and timing constraints.

Constraint File	Location

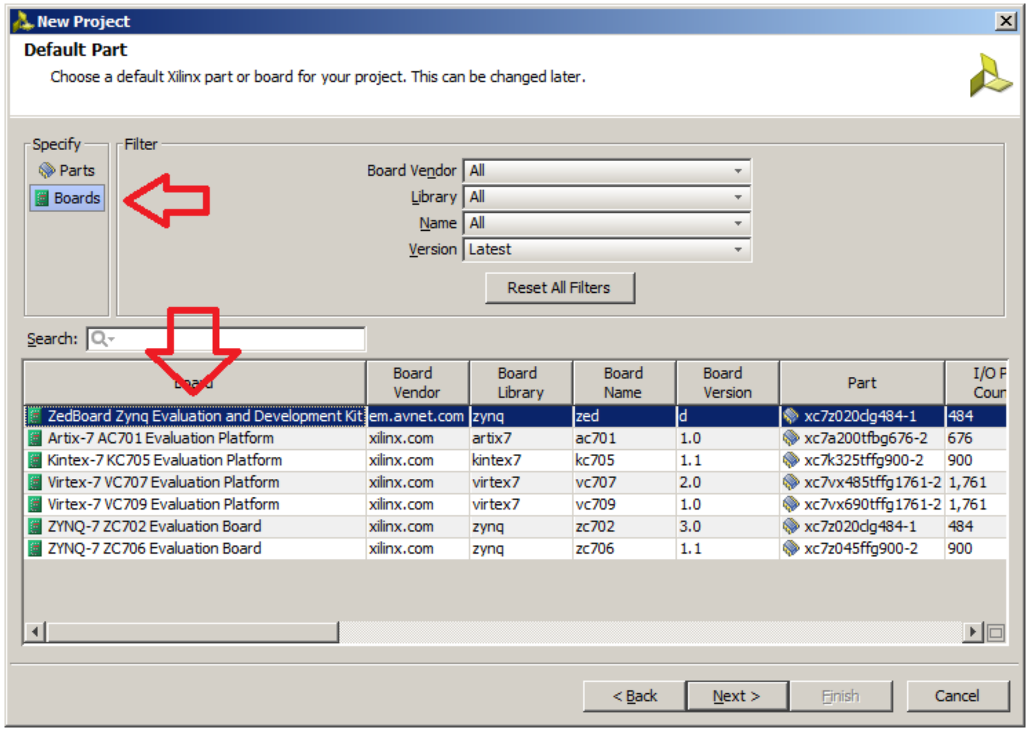
☑ Copy constraints files into project

- no constraints !!

- ISE took UCF (user constraints file)
- Vivado uses the XDC format, which is a series of TCL commands. We will not be adding any constraints to this design, but if we needed too, we would be doing so here.

Select Part / Board

- Two flows that can be seen in the upper left of the page: Parts and Boards.
- Select “Boards” and “Zedboard,”



New Project

Default Part

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

Specify: Parts, Boards

Filter:
Board Vendor: All
Library: All
Name: All
Version: Latest

Search:

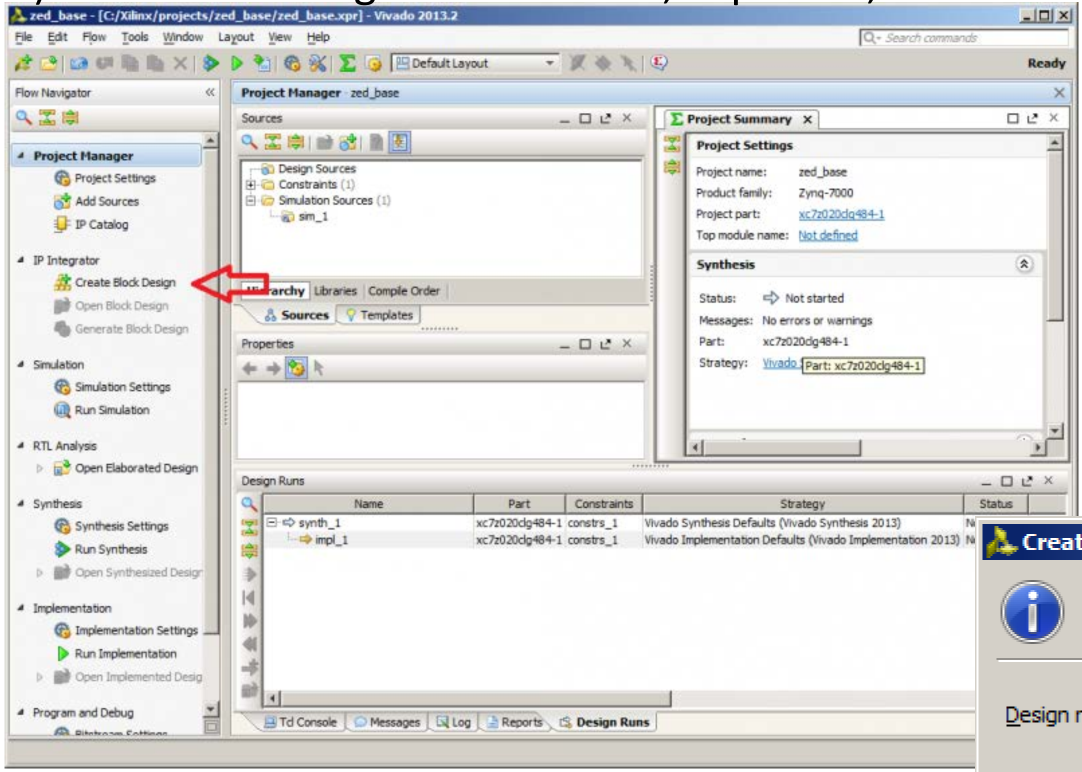
Board	Board Vendor	Board Library	Board Name	Board Version	Part	I/O P Cour
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	zynq	zed	d	xc7z020clg484-1	484
Artix-7 AC701 Evaluation Platform	xilinx.com	artix7	ac701	1.0	xc7a200tfbg676-2	676
Kintex-7 KC705 Evaluation Platform	xilinx.com	kintex7	kc705	1.1	xc7k325tffg900-2	900
Virtex-7 VC707 Evaluation Platform	xilinx.com	virtex7	vc707	2.0	xc7vx485tffg1761-2	1,761
Virtex-7 VC709 Evaluation Platform	xilinx.com	virtex7	vc709	1.0	xc7vx690tffg1761-2	1,761
ZYNQ-7 ZC702 Evaluation Board	xilinx.com	zynq	zc702	3.0	xc7z020clg484-1	484
ZYNQ-7 ZC706 Evaluation Board	xilinx.com	zynq	zc706	1.1	xc7z045ffg900-2	900

The last page of the wizard is a summary of your selection. Review it, and click Finish.

Create

- Now: Project is configured → default view of Vivado.
- You will see the Flow Navigator on the left side of the window.
- launch the various steps of your design process
- including creating a new Block Design with IPI.

1) Find the IP Integrator tree item, expand it, and select 'Create Block Design'.

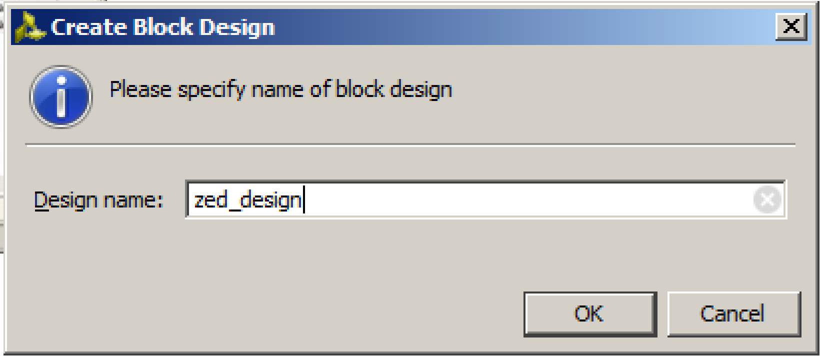


Project Settings
Project name: zed_base
Product family: Zynq-7000
Project part: xc7z020clg484-1
Top module name: Not defined

Synthesis
Status: Not started
Messages: No errors or warnings
Part: xc7z020clg484-1
Strategy: Vivado Part: xc7z020clg484-1

Design Runs

Name	Part	Constraints	Strategy	Status
synth_1	xc7z020clg484-1	constrs_1	Vivado Synthesis Defaults (Vivado Synthesis 2013)	N
impl_1	xc7z020clg484-1	constrs_1	Vivado Implementation Defaults (Vivado Implementation 2013)	N

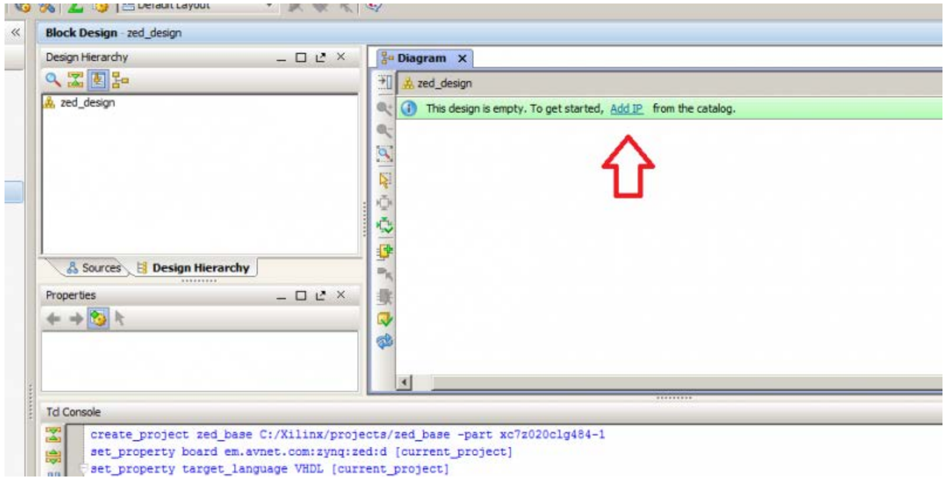


Please specify name of block design

Design name: zed_design

2) Name your block design

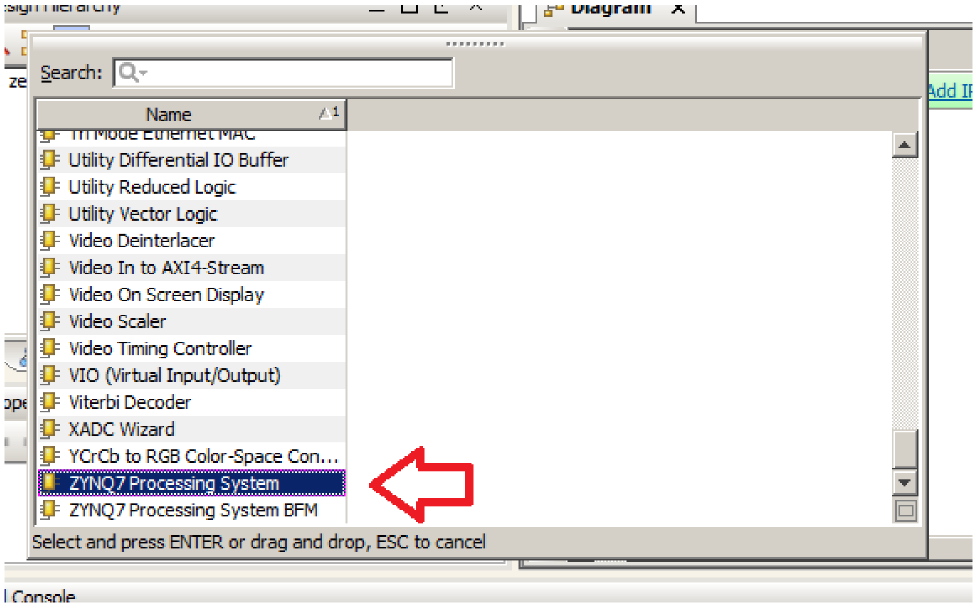
Add IPI Block


Block Design - zed_design
Design Hierarchy
zed_design
Sources
Design Hierarchy
Properties
Diagram
zed_design
This design is empty. To get started, Add IP from the catalog.
Tcl Console

```
create_project zed_base C:/Xilinx/projects/zed_base -part xc7z020clg484-1
set_property board em.avnet.com:zynq:zed:d [current_project]
set_property target_language VHDL [current_project]
```

The IPI block design: is now open, and you can see a "Diagram" tab now appears within the Block Design view on the right side of the Vivado window. Locate the small green advisory bar on the top of the Diagram tab. We want to add the Zynq Processing System (PS) to our design, so we will click the 'Add IP' link within the advisory.

Add IP dialog list.
When we click the Add IP link, we are presented with a list of available IP to be added to the design. There are a large number, but we are only interested in one: "ZYNQ7 Processing System". Find the Zynq PS within the list (it's in alphabetical order) and double click to add it to your design.

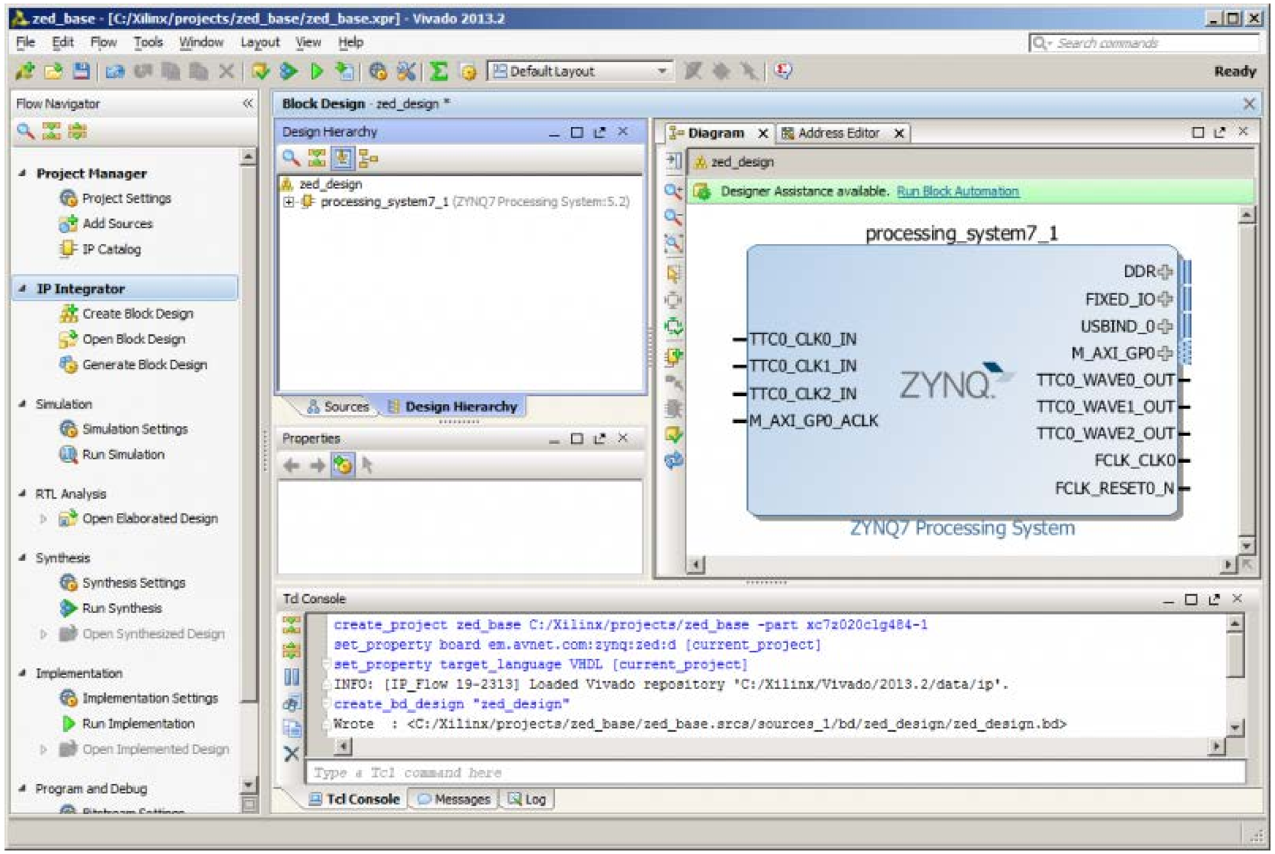


Utility Differential IO Buffer
Utility Reduced Logic
Utility Vector Logic
Video Deinterlacer
Video In to AXI4-Stream
Video On Screen Display
Video Scaler
Video Timing Controller
VIO (Virtual Input/Output)
Viterbi Decoder
XADC Wizard
YCrCb to RGB Color-Space Con...
ZYNQ7 Processing System
ZYNQ7 Processing System BFM

Select and press ENTER or drag and drop, ESC to cancel

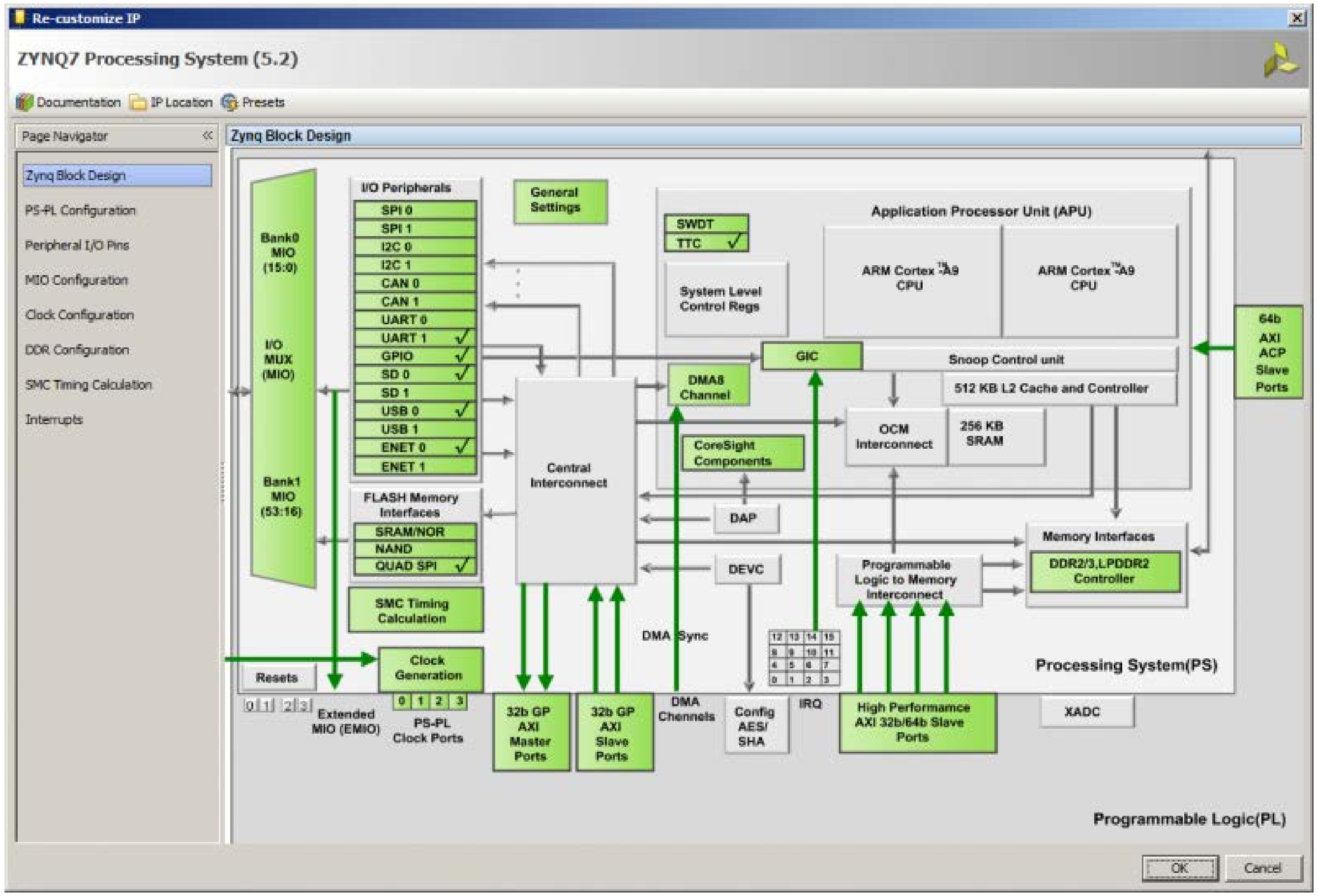
Zynq PS within IPI

- Complete processor system (PS) within the IPI block
- double click to view all of the settings you are used to seeing within XPS
- “Zedboard” is our target: no need to change any of the configuration



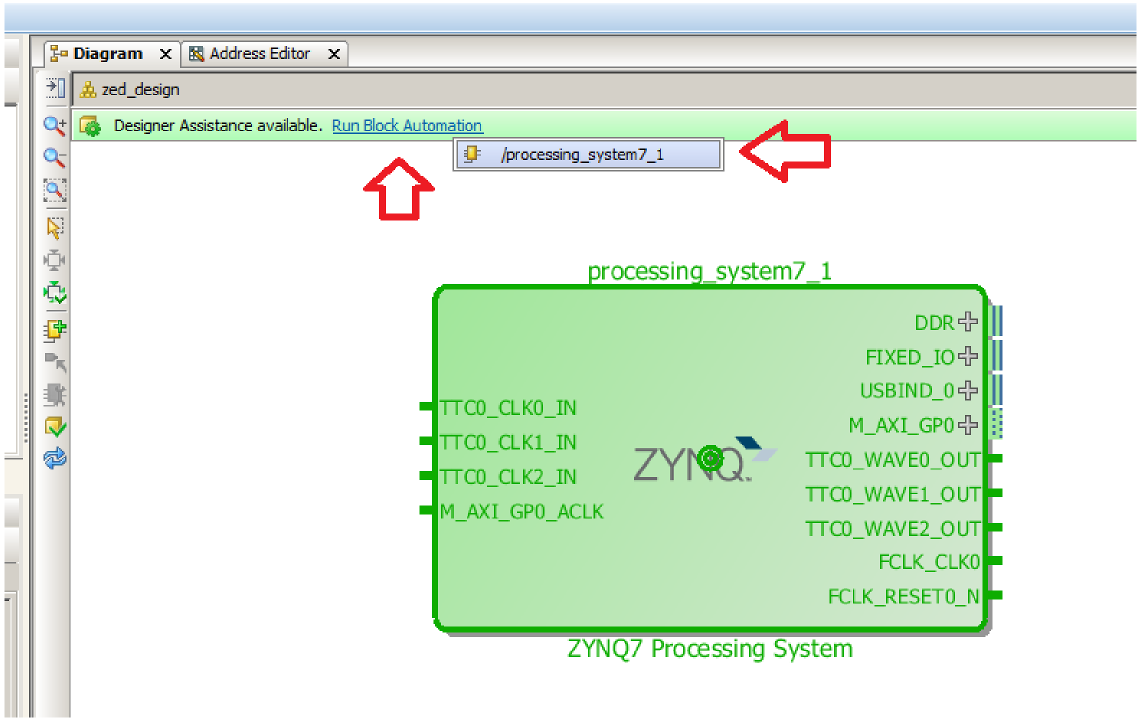
```
create_project zed_base C:/Xilinx/projects/zed_base -part xc7z020clg484-1
set_property board em.avnet.com:zynq:zed:d [current_project]
set_property target_language VHDL [current_project]
INFO: [IP_Flow 19-2313] Loaded Vivado repository 'C:/Xilinx/Vivado/2013.2/data/ip'.
create_bd_design "zed_design"
Wrote  : <C:/Xilinx/projects/zed_base/zed_base.srcs/sources_1/bd/zed_design/zed_design.bd>
```

PS configuration



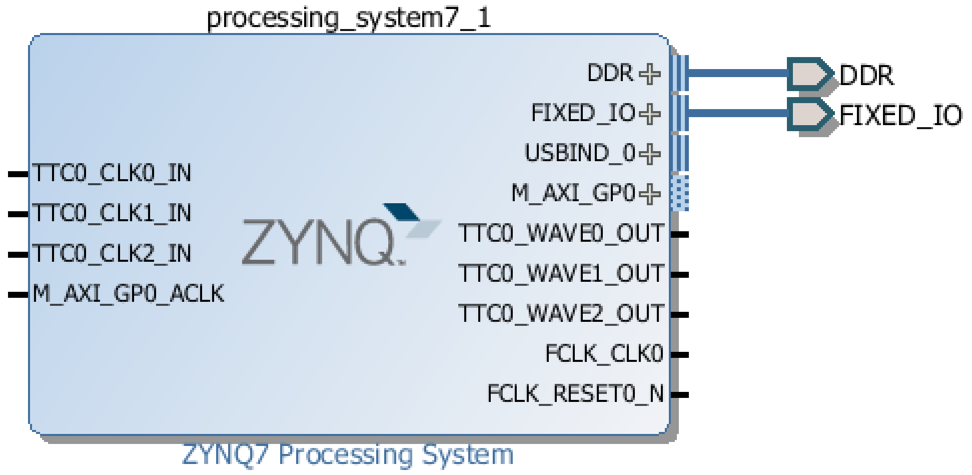
- accessed via double clicking the PS block within IPI.
- Vivado IPI is 'Board Aware'. You don't have to pull out each and every little single to the outside world (DDR3 memory, …)
- 'Run Block Automation‘: Vivado connect the signals that it knows are external

Run Block Automation



- click the 'Run Block Automation'
- Two busses have been defined on the top right of the PS block:
  - DDR and FIZED_IO.
- The DDR bus is, rather explicitly, the DDR bus.
- The FIXED_IO bus is the MIO configuration for the Zedboard
- A pop-up will show asking if you want to 'auto connect' the two busses.
  - Select OK to continue.

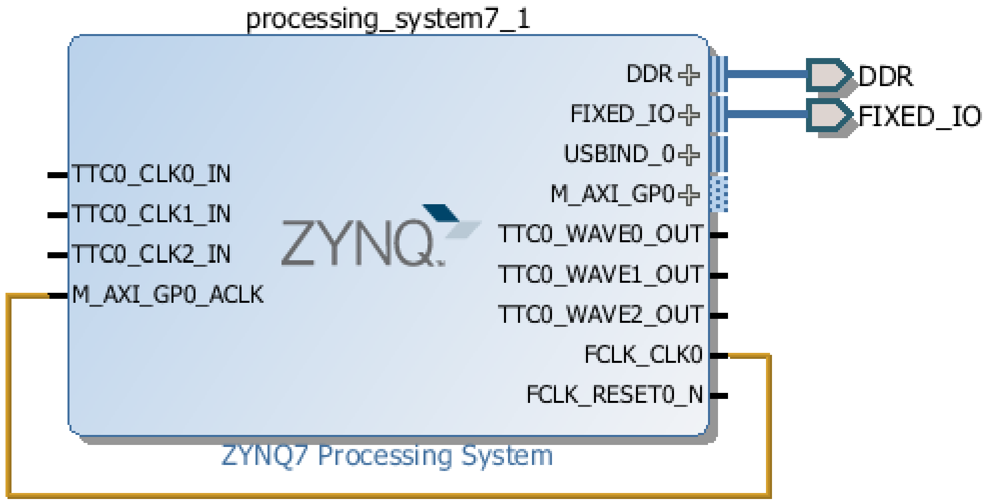
The result of the 'Run Block Automation' execution: PS with connected DDR memory and „Fixed IO“



- A AXI bus coming out of the PS: We need to clock it with one of the four clocks that are produced via the PS.
- On the right side of the PS IPI block you can see the 'FCLK_CLK0' signal - this is one of the four output clocks from the PS.
- On the left side of the block you can see a 'M_AXI_GP0_ACLK' signal - this is the input clock for the single configured AXI bus on the PS.
  - There are a number of other signals/busses present, however we won't be using any of those for our first base Zynq design.

Connect the AXI CLock

- To connect signals we need to select a output signal (right side of a block) and then select a input signal (left side of a block).
- Move your mouse and select the FCLK_CLK0 signal, and then click the M_AXI_GP0_ACLK signal. You will notice after you select the output, the inputs on the left side of the block will highlight with green check markers (these will only show up if you hover over a block).

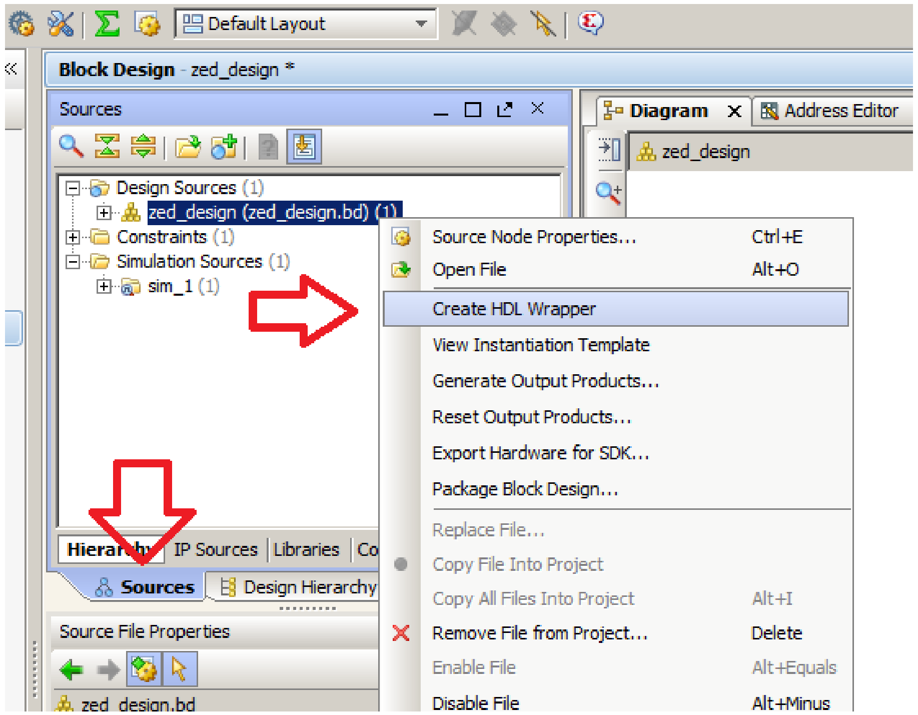


The connected, configured, and ports-made-external'ized PS system.

We've created our PS, configured it, made the necessary ports external, and clocked our signal AXI port.

Create a HDL Wrapper

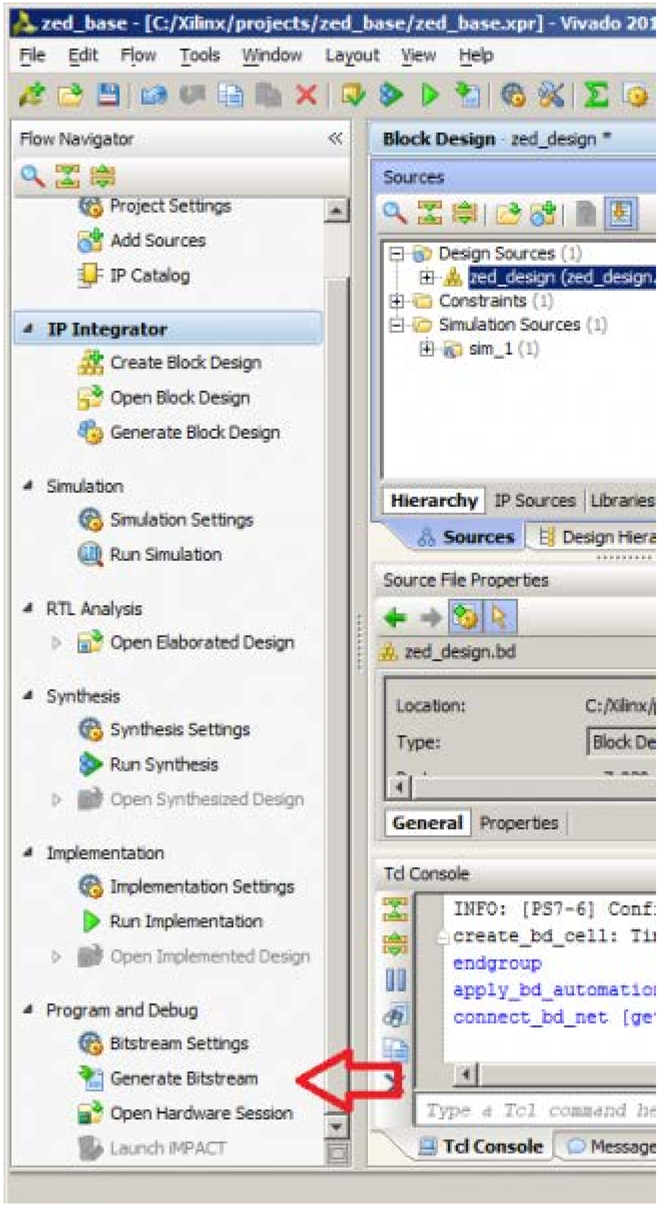
Next we need to create an HDL wrapper for the FPGA logic implementation (Synthesis)



- Make sure you have the Sources tab selected
- Right mouse click on the “zed_design” IPI block and select Create HDL Wrapper.
    - This will generate a HDL wrapper that the Vivado synthesizer understands.
- Now we are ready to generate the bitfile.

This might sound like a large jump, but there isn't anything else in our design - it's almost entirely PS (the only PL portion is that AXI port support logic).

Create Bitstream



```
INFO: [PS7-6] Conf
create_bd_cell: Ti
endgroup
apply_bd_automatio
connect_bd_net [ge
```

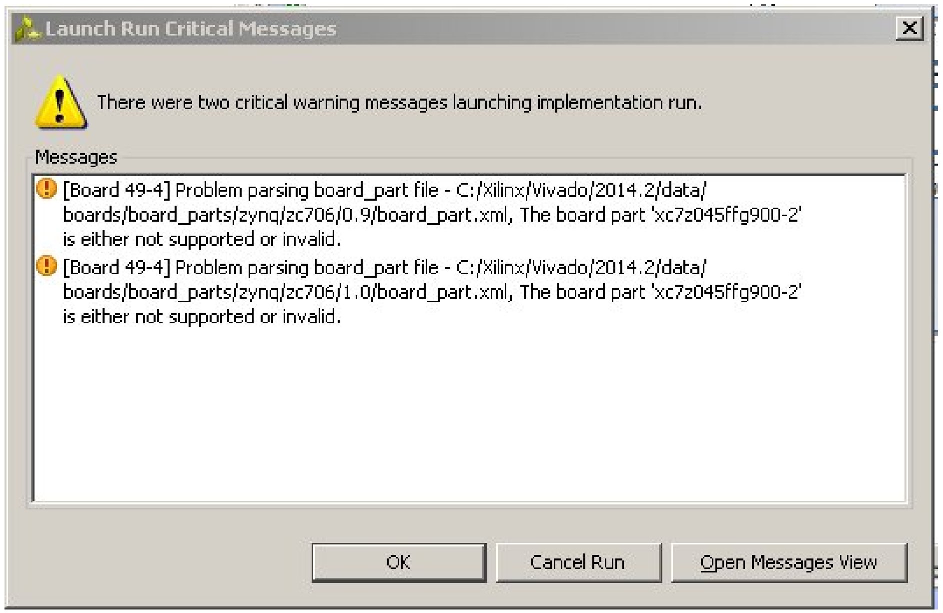
- Within the **Flow Navigator**, find
  'Generate Bitstream'

- When finished: A pop-up will show asking you if you want to open the implemented design, View reports, Open hardware session, or Launch iMPACT.
- For this flow/example we will keep the default and 'Open Implemented Design'. Select the radio button and select OK.

- Now we need to export the hardware design to SDK
  File -> Export -> Export hardware for SDK

Note: if you don't see this option, you probably didn't open the implemented design. Open the implemented design by finding 'Open Implemented Design' under 'Implementation' within the Flow Navigator).

Bug in 2014.2: Critical warning: Part not supported



There were two critical warning messages launching implementation run.

```
[Board 49-4] Problem parsing board_part file - C:/Xilinx/Vivado/2014.2/data/boards/board_parts/zynq/zc706/0.9/board_part.xml, The board part 'xc7z045ffg900-2' is either not supported or invalid.
[Board 49-4] Problem parsing board_part file - C:/Xilinx/Vivado/2014.2/data/boards/board_parts/zynq/zc706/1.0/board_part.xml, The board part 'xc7z045ffg900-2' is either not supported or invalid.
```

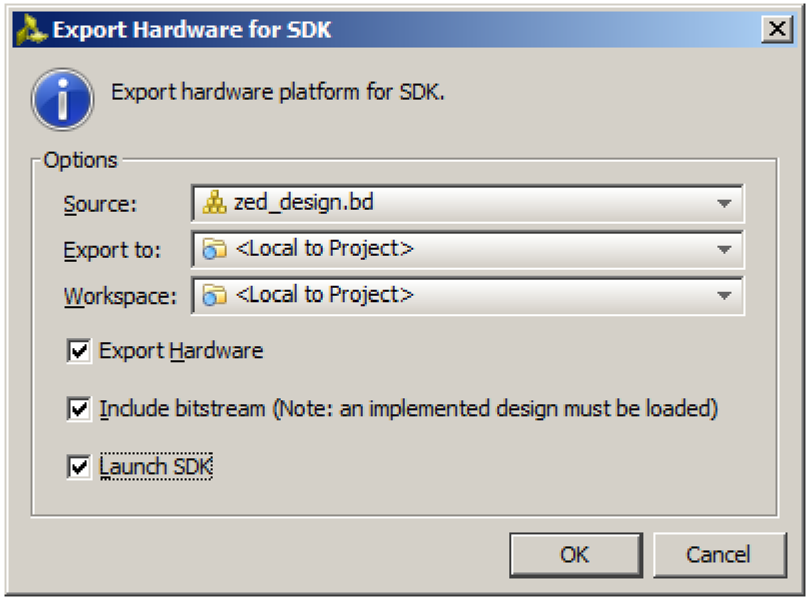
Just click OK

these warnings being critical is an issue of Vivado 2014.2, and can be remedied by making the design tools ignore this specific warning (which was present in earlier versions too, but not critical). The solution is to downgrade the error.
With the project open, type the following in the tcl-console:

set_msg_config -id {Board 49-4} -new_severity {Warning}

https://github.com/RedPitaya/RedPitaya/issues/10

Export the hardware configuration to the SDK


Export Hardware for SDK

Export hardware platform for SDK.

Options

Source: zed_design.bd

Export to: <Local to Project>

Workspace: <Local to Project>

[✓] Export Hardware

[✓] Include bitstream (Note: an implemented design must be loaded)

[✓] Launch SDK

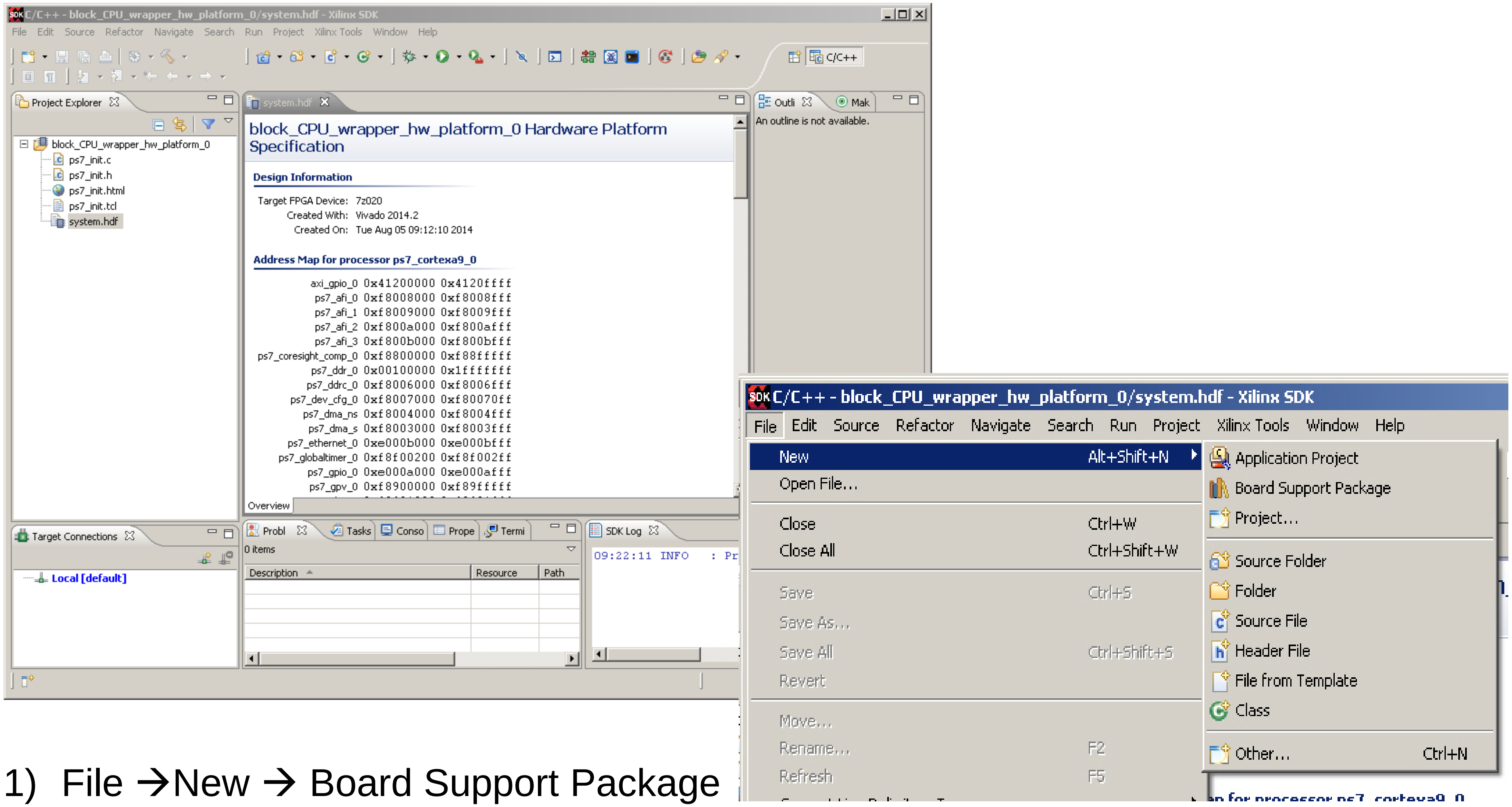
File -> Export -> Export hardware for SDK.

- Export to SDK dialog.
- Make sure you select 'Export Hardware', 'Include bitstream', and 'Launch SDK', and leave all the other fields default and select OK.

Software Development

SDK: Software Development Kit

In Vivado: File → Launch SDK



block_CPU_wrapper_hw_platform_0 Hardware Platform Specification

Design Information

Target FPGA Device:	7z020
Created With:	Vivado 2014.2
Created On:	Tue Aug 05 09:12:10 2014

Address Map for processor ps7_cortexa9_0

axi_gpio_0	0x41200000	0x4120ffff
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_ethernet_0	0xe000b000	0xe000bfff
ps7_globaltimer_0	0xf8f00200	0xf8f002ff
ps7_gpio_0	0xe000a000	0xe000afff
ps7_gpv_0	0xf8900000	0xf89fffff

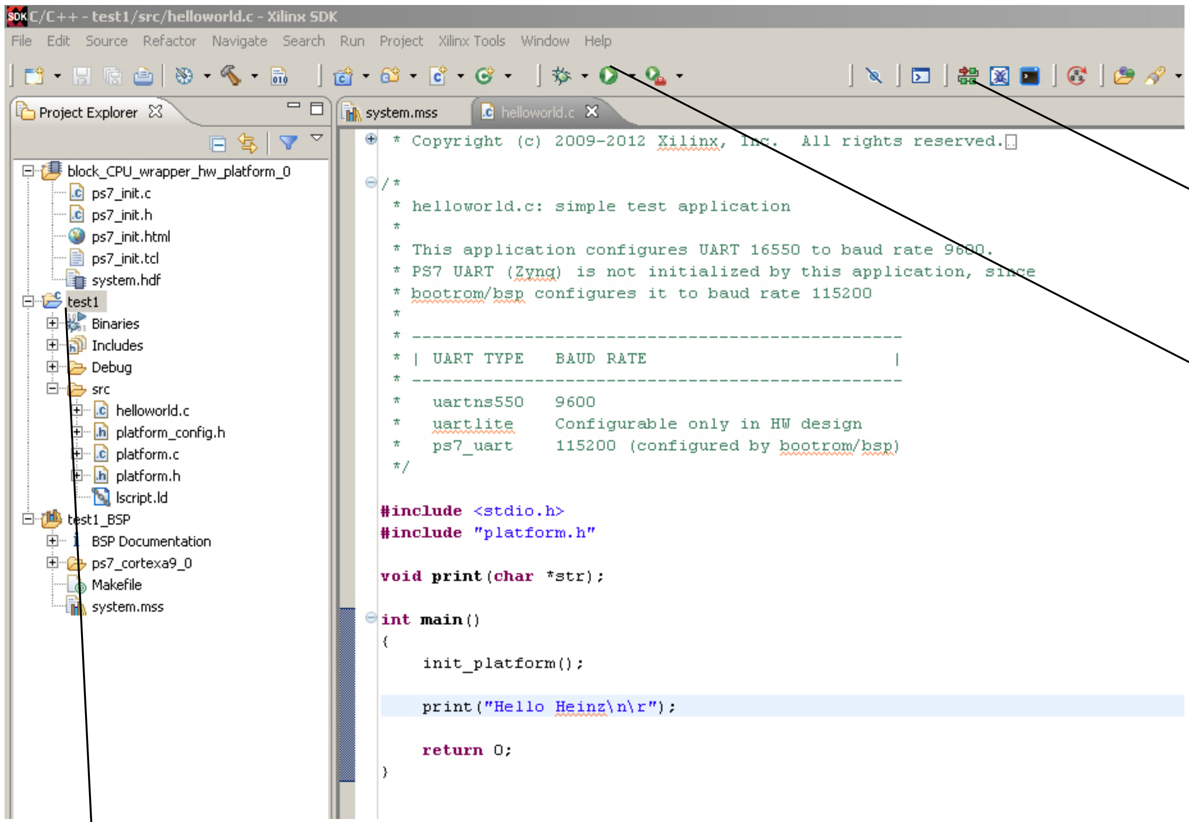
```
09:22:11 INFO    : Pr
```

1) File →New → Board Support Package (Standard options → Finish)
   Generates: *.mms
2) File → New → Application Project

SDK

- File →New →Board support package
- File → New →Application
- See in *.mms for Examples
- xParamerter.h
- xGpio.h
- Xil_io.h
- …

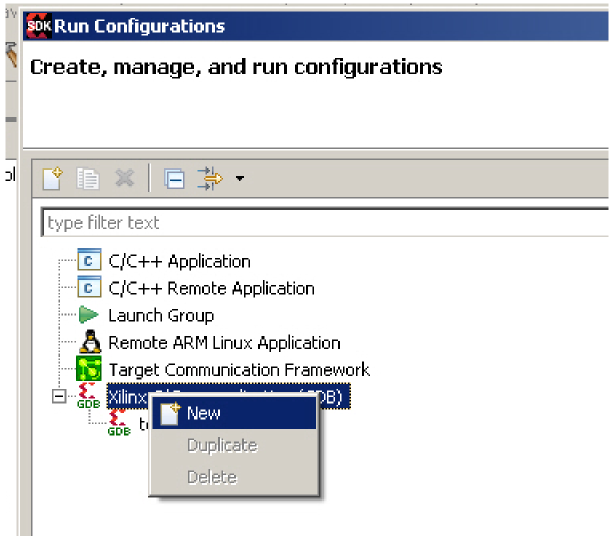
Download to target: Configure the Run Configuration



```
 * Copyright (c) 2009-2012 Xilinx, Inc.  All rights reserved.

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

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

void print(char *str);

int main()
{
    init_platform();

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

    return 0;
}
```

Download configuration (BIT file)

Run CPU code

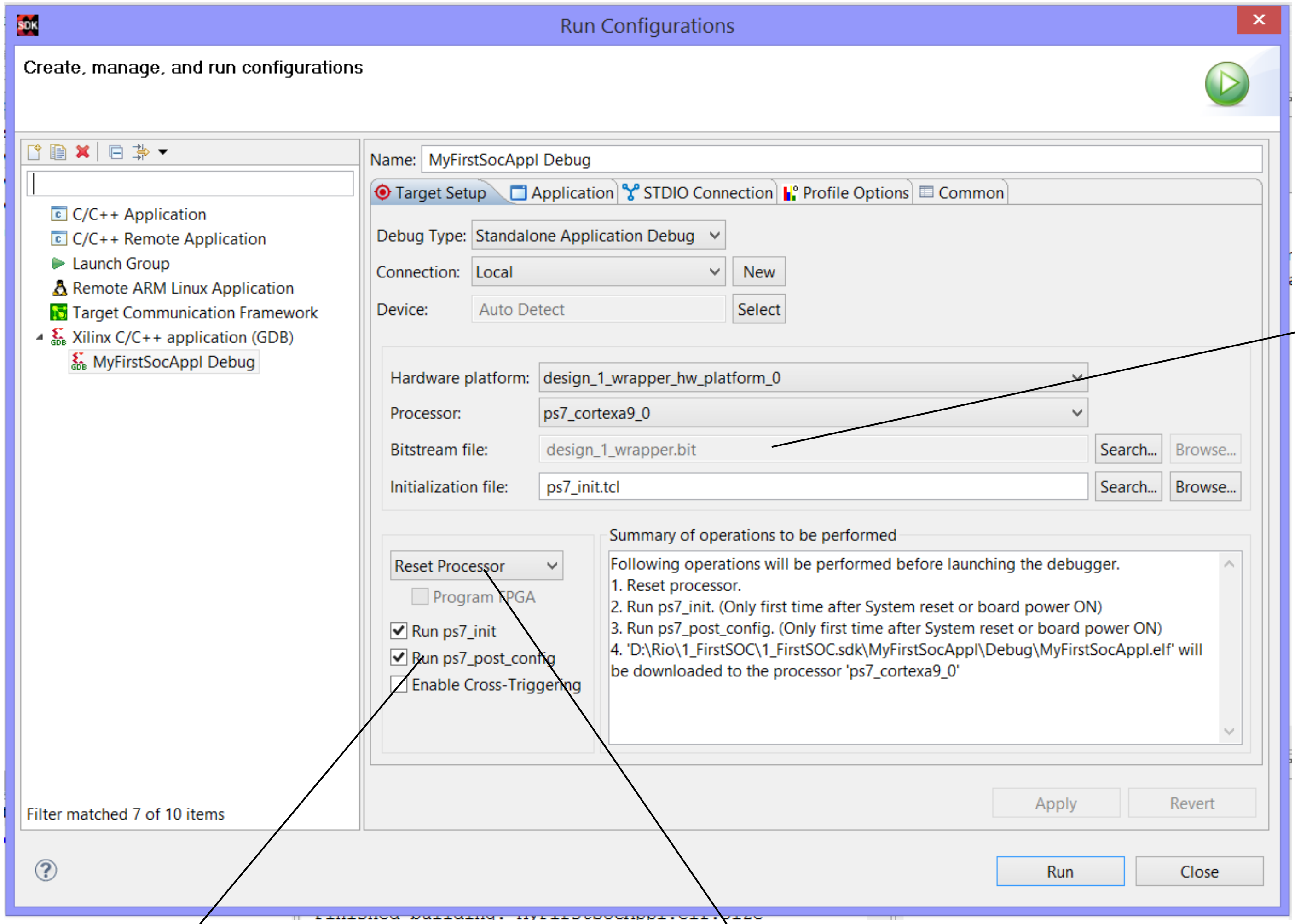


1) Click on the Application project:

Right Mouse button: Run as → Run configurations

- **Xilinx C/C++ Applications → NEW**

Configure the Run Configuration



Run Configurations

Create, manage, and run configurations

- C/C++ Application
- C/C++ Remote Application
- Launch Group
- Remote ARM Linux Application
- Target Communication Framework
- Xilinx C/C++ application (GDB)
  - MyFirstSocAppl Debug

Filter matched 7 of 10 items

Name: MyFirstSocAppl Debug

Target Setup | Application | STDIO Connection | Profile Options | Common

Debug Type: Standalone Application Debug
Connection: Local [New]
Device: Auto Detect [Select]

Hardware platform: design_1_wrapper_hw_platform_0
Processor: ps7_cortexa9_0
Bitstream file: design_1_wrapper.bit [Search...] [Browse...]
Initialization file: ps7_init.tcl [Search...] [Browse...]

Reset Processor
- [ ] Program FPGA
- [x] Run ps7_init
- [x] Run ps7_post_config
- [ ] Enable Cross-Triggering

Summary of operations to be performed

Following operations will be performed before launching the debugger.
1. Reset processor.
2. Run ps7_init. (Only first time after System reset or board power ON)
3. Run ps7_post_config. (Only first time after System reset or board power ON)
4. 'D:\Rio\1_FirstSOC\1_FirstSOC.sdk\MyFirstSocAppl\Debug\MyFirstSocAppl.elf' will be downloaded to the processor 'ps7_cortexa9_0'

Check for BIT file etc.

Enable the PS init scripts

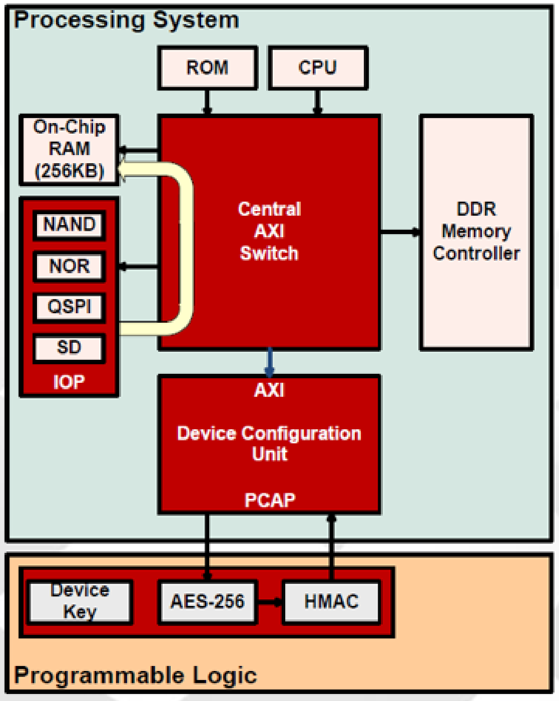
- Reset Entire System: BIT file loading included
- Reset Processor: Only ELF file loading

Click "Apply" and "Close" (or direct "Run")

Zynq-7000 Configuration and Boot

- **Processor First! CPU configures the PS and PL**
  - Standalone PL configuration (without PS configuration) is not supported
  - Configuration under external host control is also possible via JTAG

- **CPU starts executing code from ROM**
  1. Initializes the Cortex-A9 CPU 0
  2. Checks CRC on ROM code
  3. Reads the **boot mode pins** to determine the stage 1 boot mode
     - Can boot from QSPI, NAND, NOR, SD card, or JTAG
     - Stage 1 boot from SD requires SD to be connected to pre-defined MIO pins
  4. Typically, a **first stage boot loader** is read from external non-volatile memory and copied into the OCM (192KB max)
     - If the Execute In Place (XIP) feature is enabled, first stage boot can be executed directly from QSPI or NOR
     - Stage 1 boot header specifies the use of XIP feature

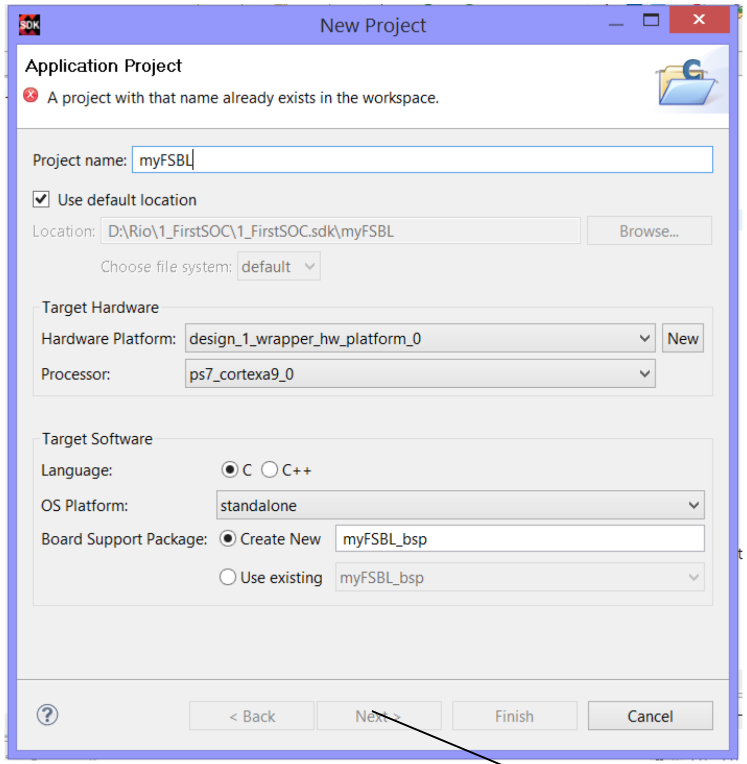


1) Arrow shows the direction of the master

Prepare for a SD card Image

Create a First Step Boot Loader (FSBL)

- File → New → Application



New Project

Application Project

A project with that name already exists in the workspace.

Project name: myFSBL

[✓] Use default location

Location: D:\Rio\1_FirstSOC\1_FirstSOC.sdk\myFSBL

Choose file system: default

Target Hardware

Hardware Platform: design_1_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Target Software

Language: [✓ C] C++

OS Platform: standalone

Board Support Package: [✓ Create New] myFSBL_bsp

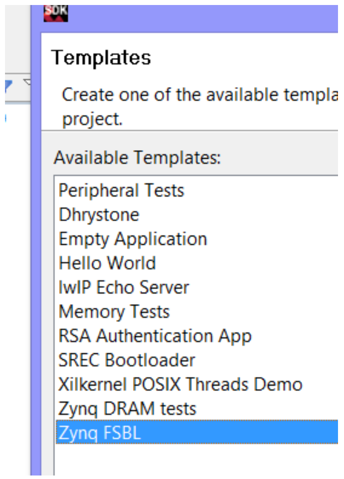
Use existing myFSBL_bsp

- Name is “**_FSBL”

- Let him create a new BSP or the FSBL

Click **NEXT**

and select the **“Zynq FSBL”**



Templates

Create one of the available templa
project.

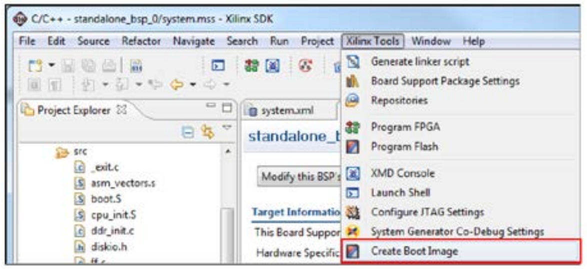
Available Templates:

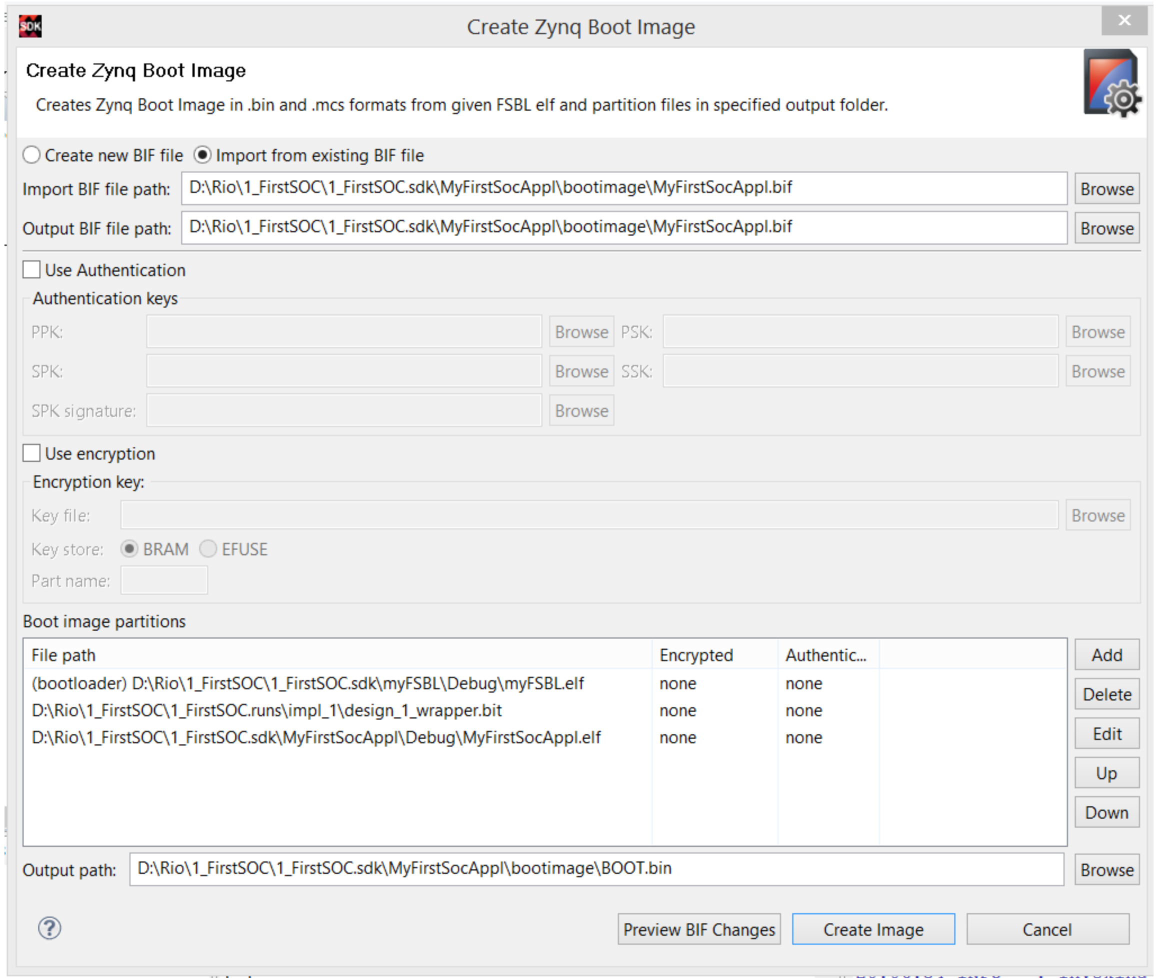
Peripheral Tests
Dhrystone
Empty Application
Hello World
lwIP Echo Server
Memory Tests
RSA Authentication App
SREC Bootloader
Xilkernel POSIX Threads Demo
Zynq DRAM tests
[✓ Zynq FSBL]

The FSBL is now created

Create a Zynq Boot Image for SD Card

Xilinx Tools → 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]

Import BIF file path: D:\Rio\1_FirstSOC\1_FirstSOC.sdk\MyFirstSocAppl\bootimage\MyFirstSocAppl.bif

Output BIF file path: D:\Rio\1_FirstSOC\1_FirstSOC.sdk\MyFirstSocAppl\bootimage\MyFirstSocAppl.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	Authentic...
(bootloader) D:\Rio\1_FirstSOC\1_FirstSOC.sdk\myFSBL\Debug\myFSBL.elf	none	none
D:\Rio\1_FirstSOC\1_FirstSOC.runs\impl_1\design_1_wrapper.bit	none	none
D:\Rio\1_FirstSOC\1_FirstSOC.sdk\MyFirstSocAppl\Debug\MyFirstSocAppl.elf	none	none

Output path: D:\Rio\1_FirstSOC\1_FirstSOC.sdk\MyFirstSocAppl\bootimage\BOOT.bin

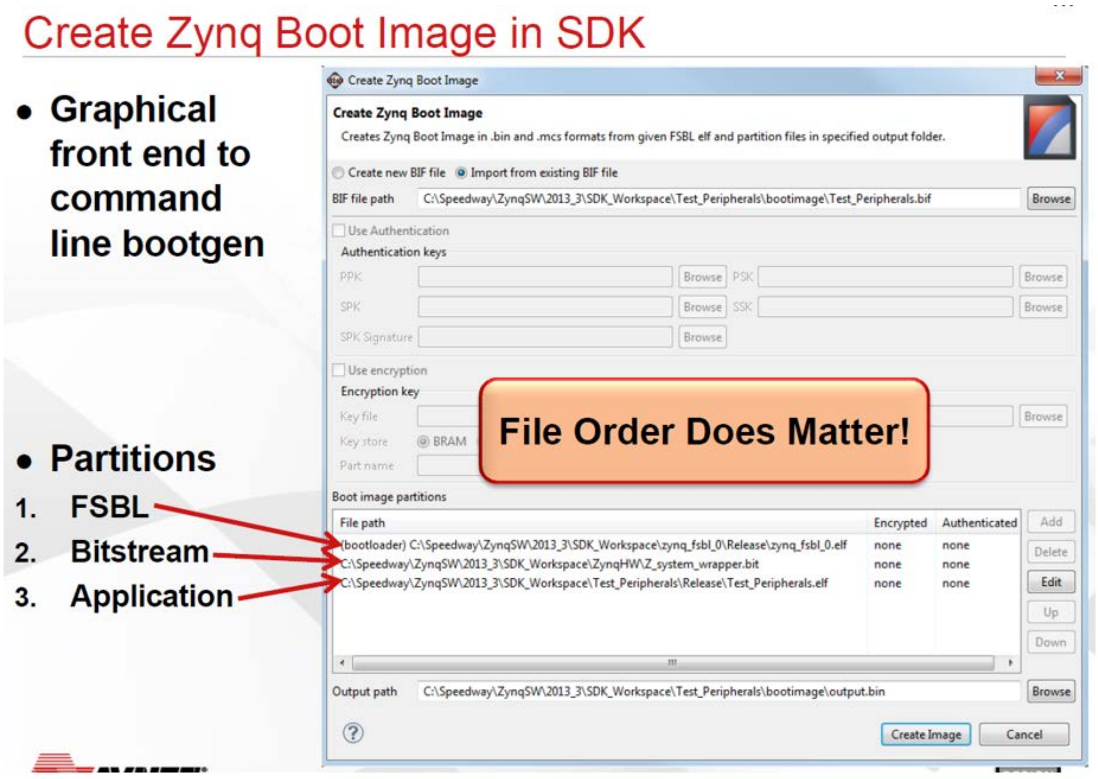
Add the following files:

- the FSBL
- the BIT file
- the ELF file

- Click “Create Image”

- BOOT.BIN is created

Directories to find the components



Create Zynq Boot Image in SDK

- Graphical front end to command line bootgen

- Partitions

1. FSBL
2. Bitstream
3. Application

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]

BIF file path: C:\Speedway\ZynqSW\2013_3\SDK_Workspace\Test_Peripherals\bootimage\Test_Peripherals.bif

Use Authentication

Authentication keys

Use encryption

Encryption key

Key store: BRAM

File Order Does Matter!

Boot image partitions

File path	Encrypted	Authenticated
(bootloader) C:\Speedway\ZynqSW\2013_3\SDK_Workspace\zynq_fsbl_0\Release\zynq_fsbl_0.elf	none	none
C:\Speedway\ZynqSW\2013_3\SDK_Workspace\ZynqHW\Z_system_wrapper.bit	none	none
C:\Speedway\ZynqSW\2013_3\SDK_Workspace\Test_Peripherals\Release\Test_Peripherals.elf	none	none

Output path: C:\Speedway\ZynqSW\2013_3\SDK_Workspace\Test_Peripherals\bootimage\output.bin

- **First Step Boot Loader**

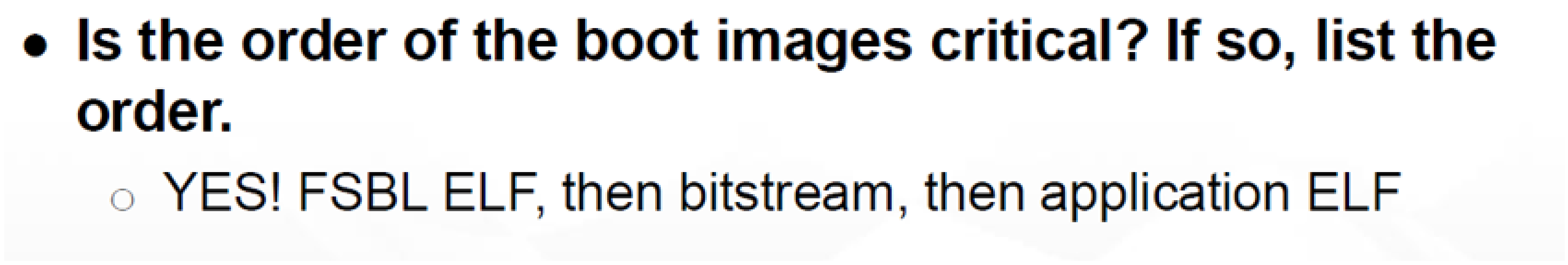
  ***.sdk\myFSBL\Debug**

- **FPGA configuration**

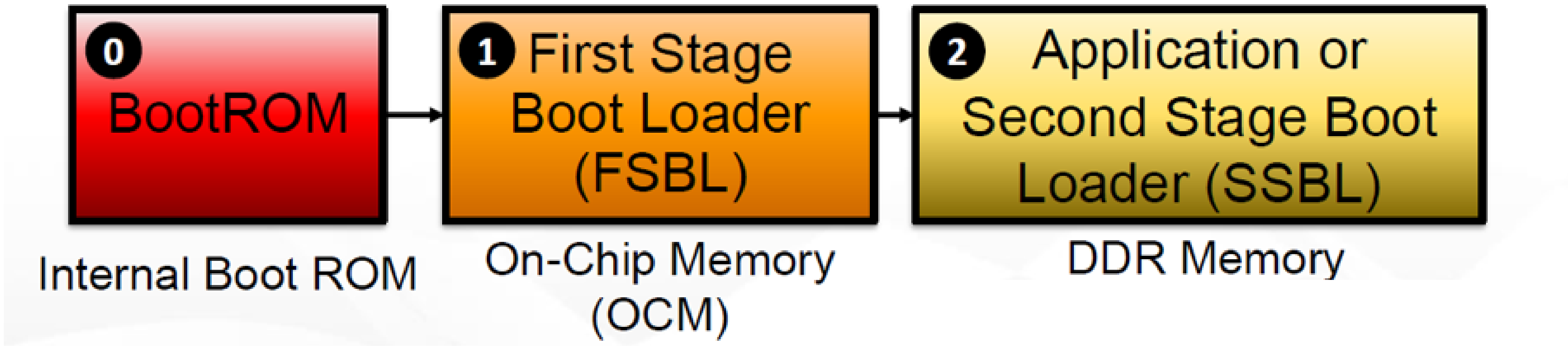
  ***.runs\impl_1**

- **Application**

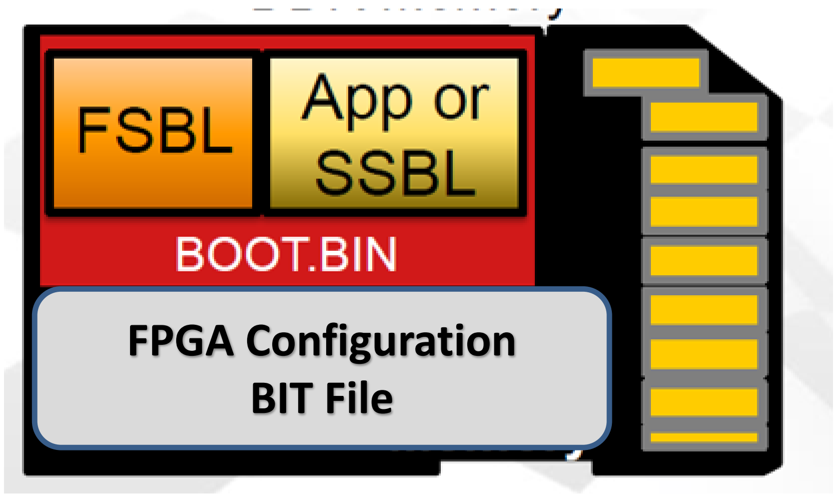
  ***.sdk\MyFirstSocAppl\Debug**



- **Is the order of the boot images critical? If so, list the order.**
  - YES! FSBL ELF, then bitstream, then application ELF

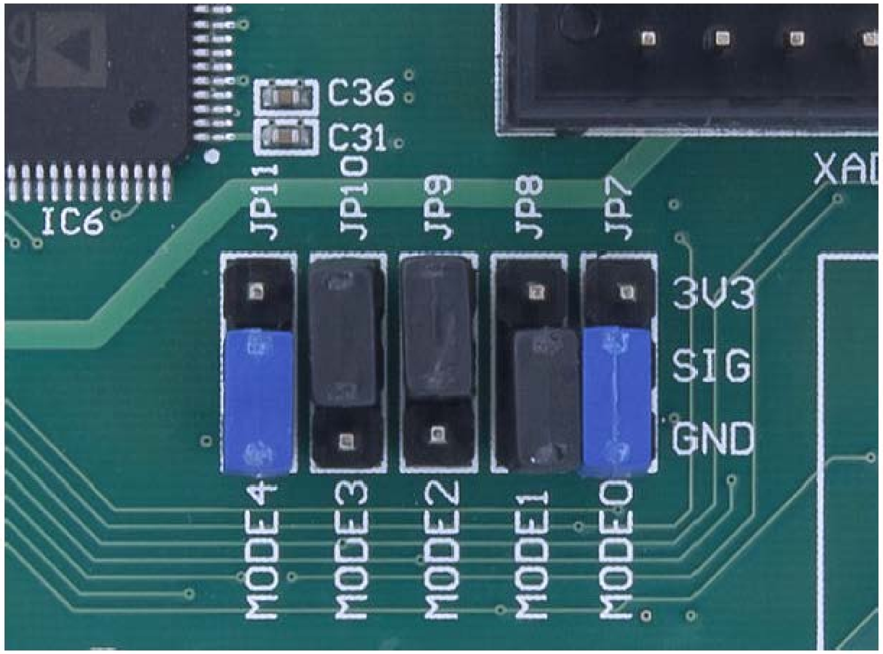

0
BootROM
1
First Stage Boot Loader (FSBL)
2
Application or Second Stage Boot Loader (SSBL)
Internal Boot ROM
On-Chip Memory (OCM)
DDR Memory

- all in **BOOT.BIN**



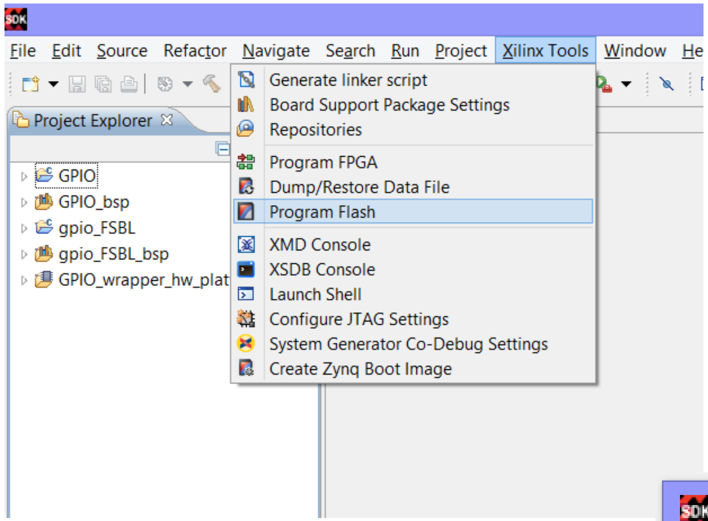
Booting from SD Card

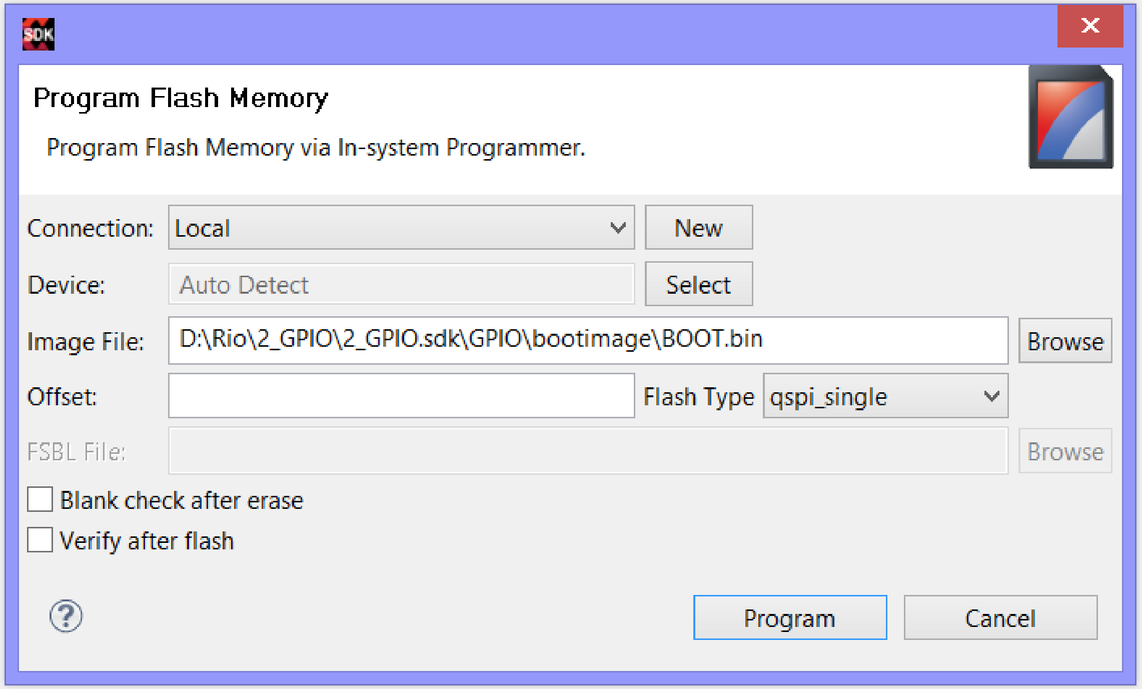
- Copy BOOT.BIN on a SD card
- Set Jumper according
- Insert CD card
- Power Cycle the ZedBoard



Booting from FLASH (QSPI Memory)

- Xilinx Tools → Program Flash





Program Flash Memory

Program Flash Memory via In-system Programmer.

Connection: Local

Device: Auto Detect

Image File: D:\Rio\2_GPIO\2_GPIO.sdk\GPIO\bootimage\BOOT.bin

Offset:

Flash Type qspi_single

FSBL File:

Blank check after erase

Verify after flash

- Browse the BOOT.BIN
- → Program

- Needs some time …

Booting from FLASH (QSPI Memory)

. Turn off the ZedBoard. Verify the Configuration Mode jumpers are set for QSPI boot mode as described and in the figure below:

- MODE3 (JP10) shunted to 3.3V
- All other MODE pins shunted to GND