# Use Vivado to build an Embedded System

## Introduction

This lab guides you through the process of using Vivado to create a simple ARM Cortex-A9 based processor design targeting the Zybo board. Where the instructions refer to both boards, choose the board you are using. You will use Vivado to create the hardware system and SDK (Software Development Kit) to create an example application to verify the hardware functionality.

## Objectives

After completing this lab, you will be able to:

- Create a Vivado project for a Zynq system
- Use the IP Integrator to create a hardware system
- Use SDK to create a standard memory test project
- Run the test application on the board

## Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

This lab comprises 5 primary steps: You will create a top-level project using Vivado, create the processor system using the Vivado IP Integrator, generate the top-level HDL and export the design to SDK, create a Memory Test application in SDK, and finally, test in hardware.

## Design Description

The purpose of the lab exercises is to walk you through a complete hardware and software processor system design. Each lab will build upon the previous lab. The following diagram represents the completed design (**Figure 1**).

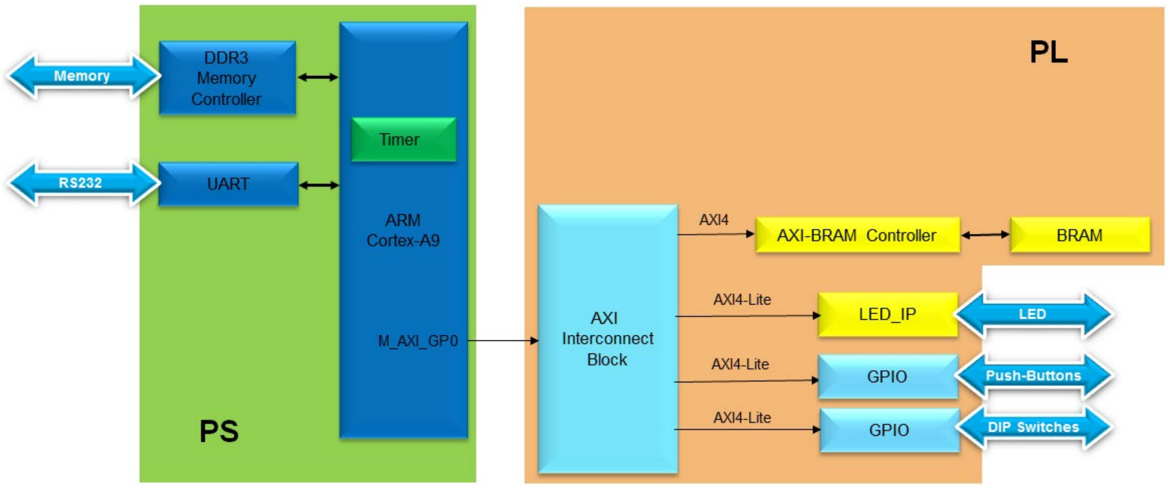


**Figure 1. Completed Design**

In this lab, you will use IP Integrator to create a processing system based design consisting of the following (**Figure 2**):

- ARM Cortex A9 core (PS)
- UART for serial communication
- DDR3 controller for external DDR3_SDRAM memory

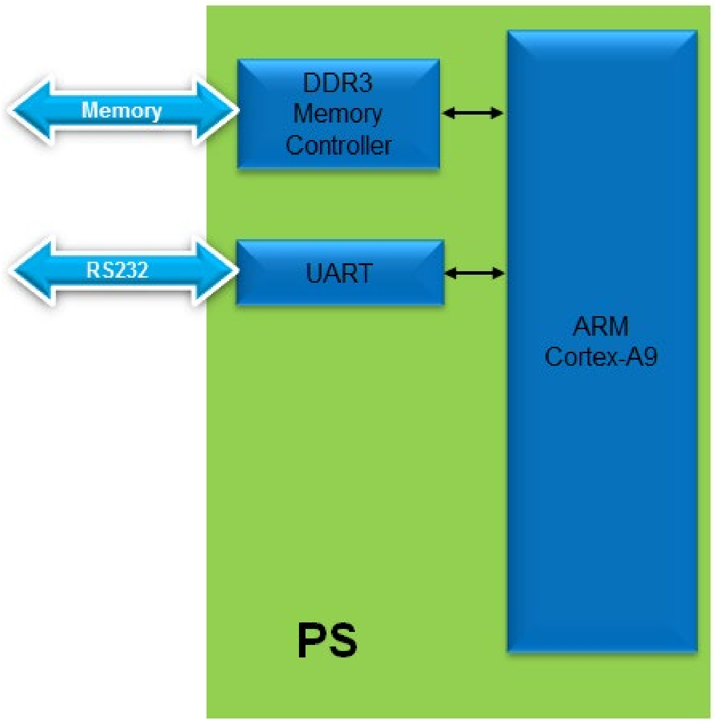
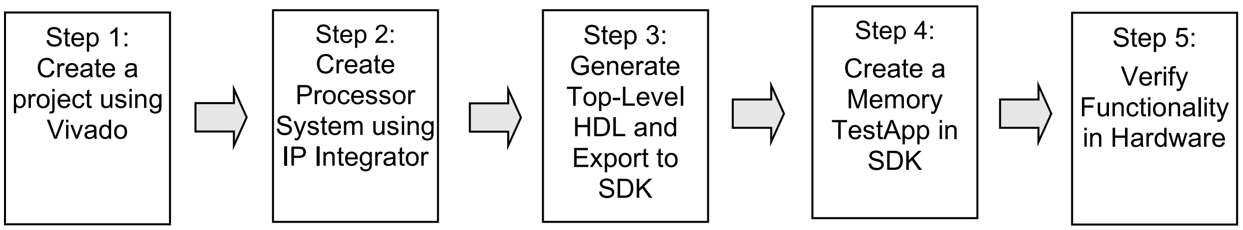


**Figure 2. Processor Design of this Lab**

## General Flow for this Lab


Step 1: Create a project using Vivado
Step 2: Create Processor System using IP Integrator
Step 3: Generate Top-Level HDL and Export to SDK
Step 4: Create a Memory TestApp in SDK
Step 5: Verify Functionality in Hardware

## Create a Vivado Project Step 1

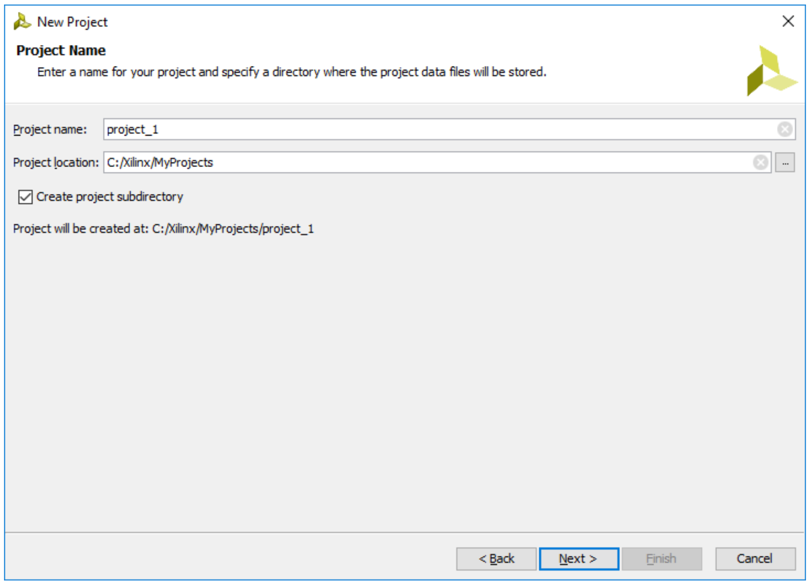
### 1-1. Launch Vivado and create an empty project targeting the Zybo, using the VHDL language.

**1-1-1.** Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2018.3**

**1-1-2.** Click **Create Project** to start the wizard. You will see the *Create a New Vivado Project* dialog box. Click **Next**.

**1-1-3.** Click the Browse button of the *Project Location* field of the **New Project** form, browse to **C:/Xilinx/MyProjects**, and click **Select**.

**1-1-4.** Enter **project _1** in the *Project Name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**. **Make sure there is no space in the path**.


New Project

**Project Name**

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

Project name: project_1

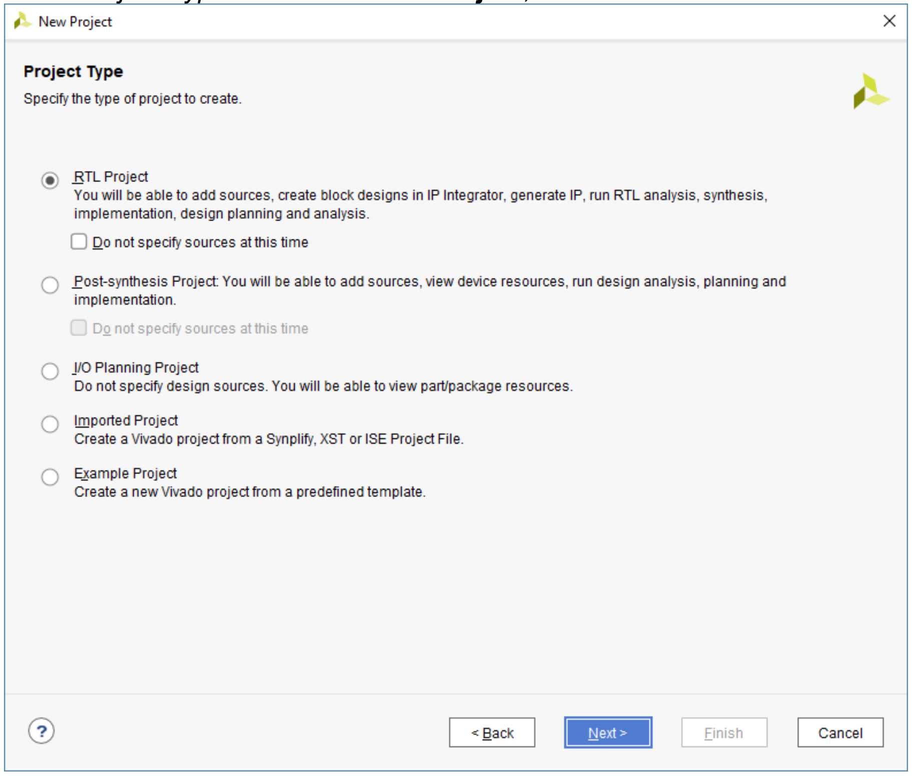
Project location: C:/Xilinx/MyProjects

[✓] Create project subdirectory

Project will be created at: C:/Xilinx/MyProjects/project_1

**Figure 3. Project Name Entry**

**1-1-5.** In the *Project Type* form select **RTL Project**, and click **Next**


New Project

**Project Type**

Specify the type of project to create.

● RTL Project
You will be able to add sources, create block designs in IP Integrator, 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.

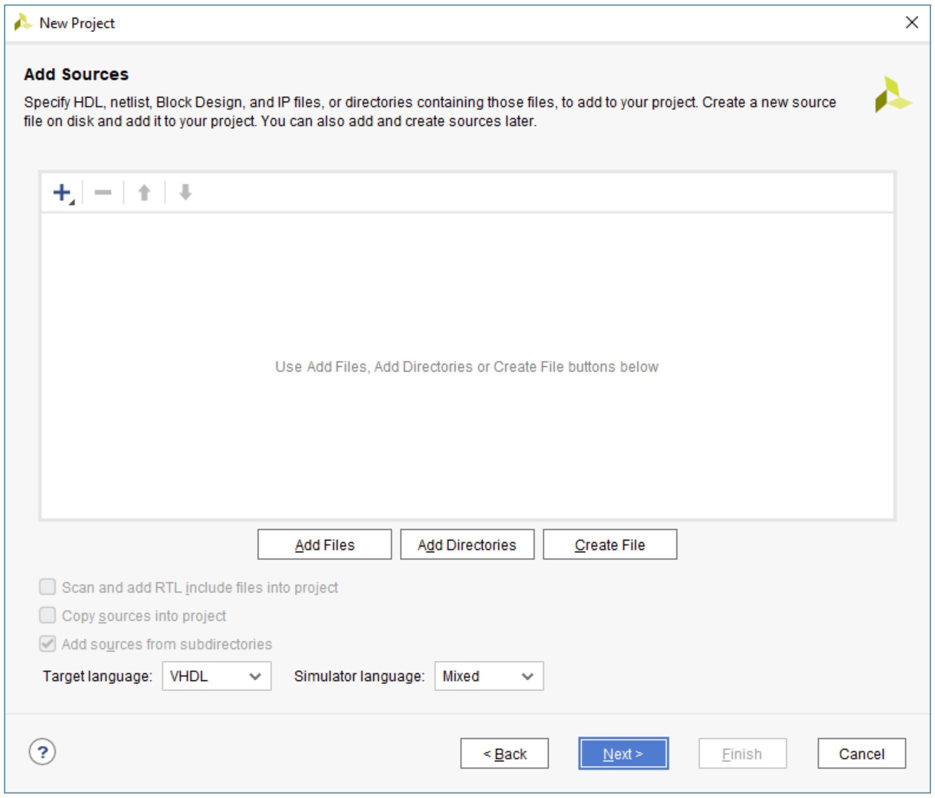
☐ Do not specify sources at this time

○ 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.

○ Example Project
Create a new Vivado project from a predefined template.

**1-1-6.** In the *Add Sources* form, select **VHDL** as the *Target language* and **Mixed** as the *Simulator language*, and click **Next**



New Project

**Add Sources**

Specify HDL, netlist, Block Design, and IP files, or directories containing those 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.

Use Add Files, Add Directories or Create File buttons below

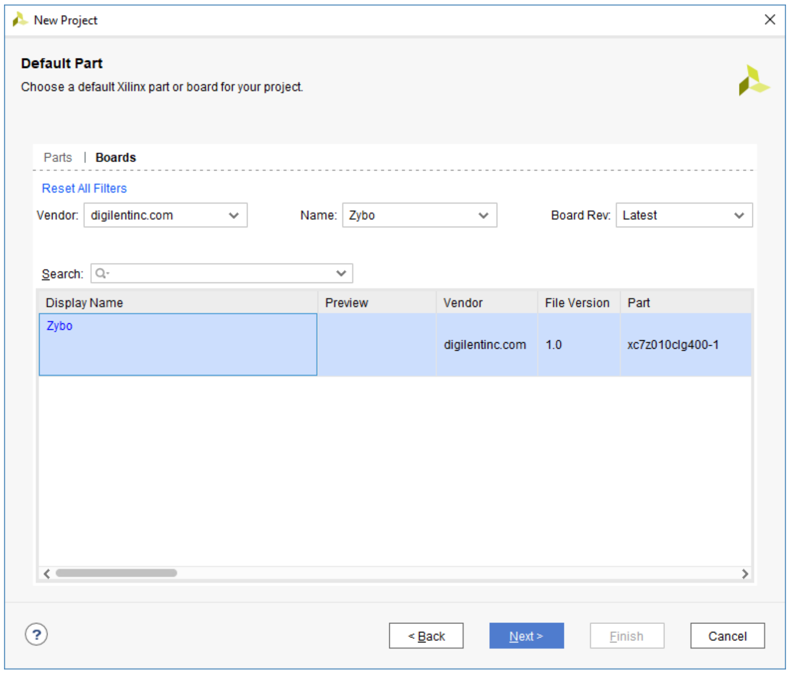
☐ Scan and add RTL include files into project
☐ Copy sources into project
☑ Add sources from subdirectories

Target language: VHDL    Simulator language: Mixed

**Figure 4. Add sources to new project**

**1-1-7.** Click **Next** two more times to skip *Adding Existing IP* and *Add Constraints*

**1-1-8.** In the *Default Part* form, select *Boards*, and select *Zybo* and click **Next**.



New Project

**Default Part**

Choose a default Xilinx part or board for your project.

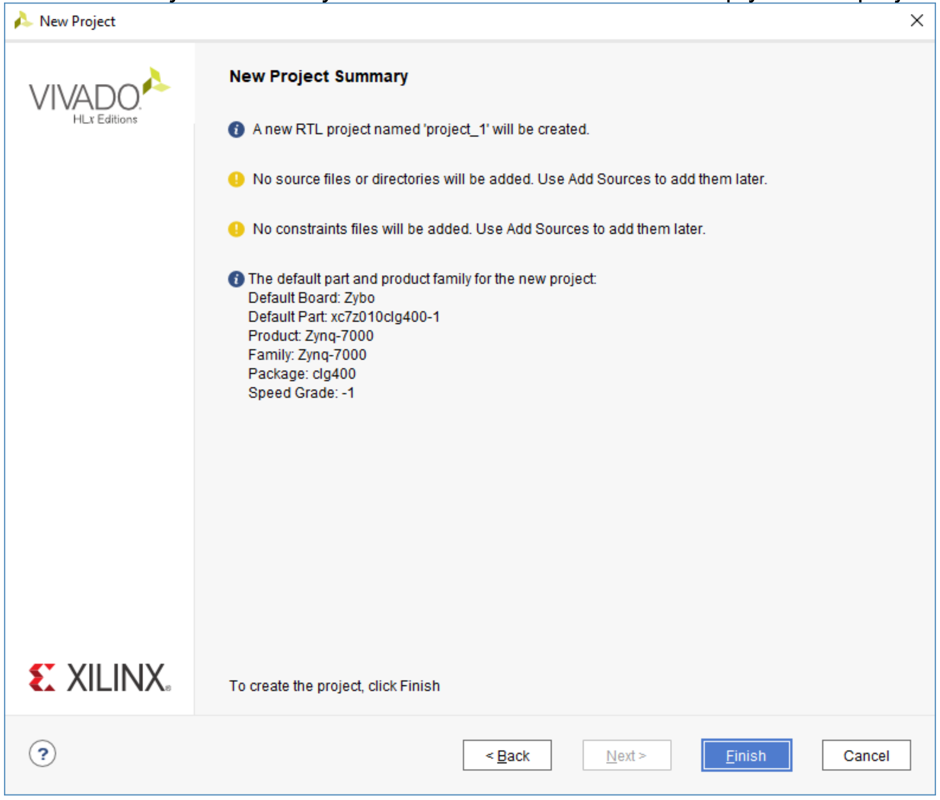
Vendor: digilentinc.com    Name: Zybo    Board Rev: Latest

Search:

| Display Name | Preview | Vendor | File Version | Part |
|---|---|---|---|---|
| Zybo | | digilentinc.com | 1.0 | xc7z010clg400-1 |

**Figure 5. Boards and Parts Selection**

**1-1-9.** Check the *Project Summary* and click **Finish** to create an empty Vivado project.



New Project Summary

A new RTL project named 'project_1' will be created.

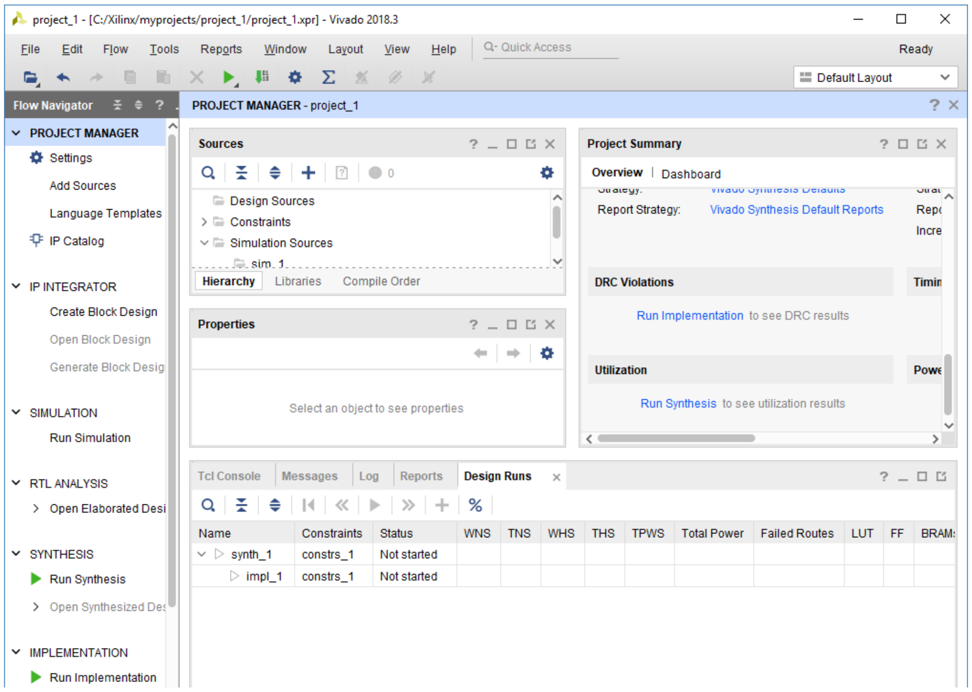
No source files or directories will be added. Use Add Sources to add them later.

No constraints files will be added. Use Add Sources to add them later.

The default part and product family for the new project:
Default Board: Zybo
Default Part: xc7z010clg400-1
Product: Zynq-7000
Family: Zynq-7000
Package: clg400
Speed Grade: -1

To create the project, click Finish

Then you will see the following window.



| Name | Constraints | Status | WNS | TNS | WHS | THS | TPWS | Total Power | Failed Routes | LUT | FF | BRAM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| synth_1 | constrs_1 | Not started | | | | | | | | | | |
| impl_1 | constrs_1 | Not started | | | | | | | | | | |

## Creating the System Using the IP Integrator Step 2

### 2-1. Use the IP Integrator to create a new Block Design, add the ZYNQ processing system block, and import the provided xml file for the board.

**2-1-1.** In the Flow Navigator, click **Create Block Design** under IP Integrator

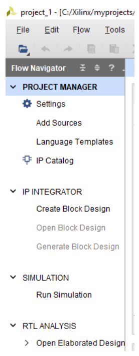
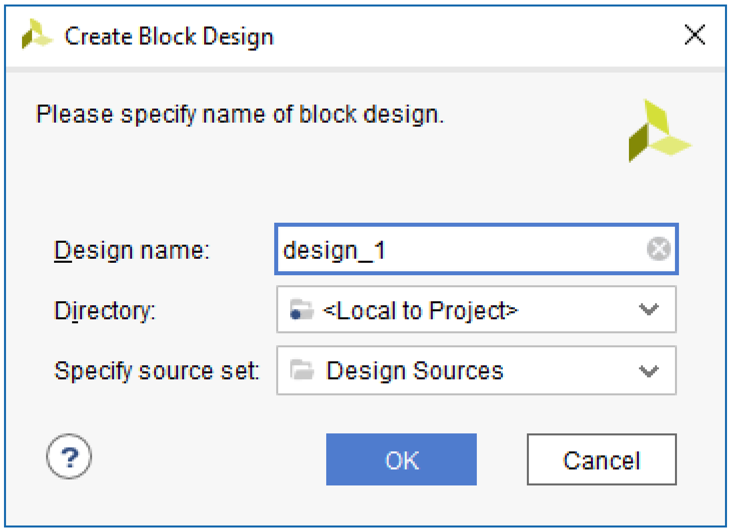


**Figure 6. Create IP Integrator Block Diagram**

**2-1-2.** Enter **design_1** for the design name and click **OK**


Create Block Design

Please specify name of block design.

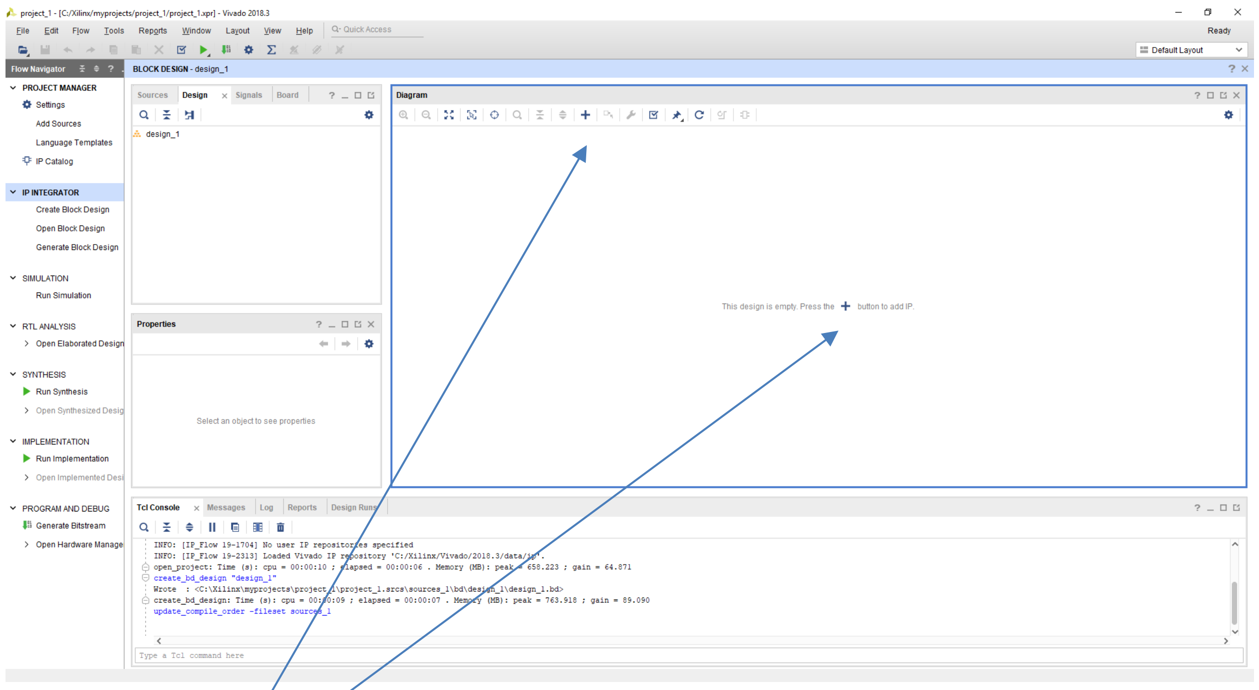
Design name: design_1

Directory: <Local to Project>

Specify source set: Design Sources

OK Cancel

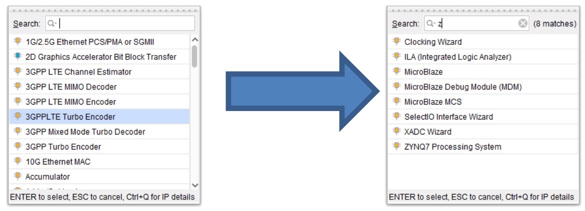
**Figure 7. Create New Block Diagram**



```
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'C:/Xilinx/Vivado/2018.3/data/ip'.
open_project: Time (s): cpu = 00:00:10 ; elapsed = 00:00:06 . Memory (MB): peak = 658.223 ; gain = 64.871
create_bd_design "design_1"
Wrote  : <C:\Xilinx\myprojects\project_1\project_1.srcs\sources_1\bd\design_1\design_1.bd>
create_bd_design: Time (s): cpu = 00:00:09 ; elapsed = 00:00:07 . Memory (MB): peak = 763.918 ; gain = 89.090
update_compile_order -fileset sources_1
Type a Tcl command here
```

**2-1-3.** IP from the catalog can be added by pressing the "+" button on top of or in the middle of the diagram window.

**2-1-4.** Once the IP Catalog is open, type "z" into the Search bar, find and double click on **ZYNQ7 Processing System** entry, or click on the entry and hit the Enter key to add it to the design.



Search:

- 1G/2.5G Ethernet PCS/PMA or SGMII
- 2D Graphics Accelerator Bit Block Transfer
- 3GPP LTE Channel Estimator
- 3GPP LTE MIMO Decoder
- 3GPP LTE MIMO Encoder
- 3GPPLTE Turbo Encoder
- 3GPP Mixed Mode Turbo Decoder
- 3GPP Turbo Encoder
- 10G Ethernet MAC
- Accumulator

ENTER to select, ESC to cancel, Ctrl+Q for IP details

Search: z (8 matches)

- Clocking Wizard
- ILA (Integrated Logic Analyzer)
- MicroBlaze
- MicroBlaze Debug Module (MDM)
- MicroBlaze MCS
- SelectIO Interface Wizard
- XADC Wizard
- ZYNQ7 Processing System

ENTER to select, ESC to cancel, Ctrl+Q for IP details

**Figure 8. Add Zynq block to the design**

**2-1-5.** Notice the message at the top of the Diagram window that *Designer Assistance* available. Click *Run Block Automation* and select **/processing_system7_0**

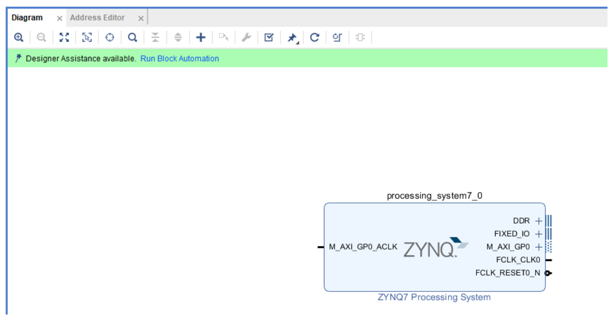
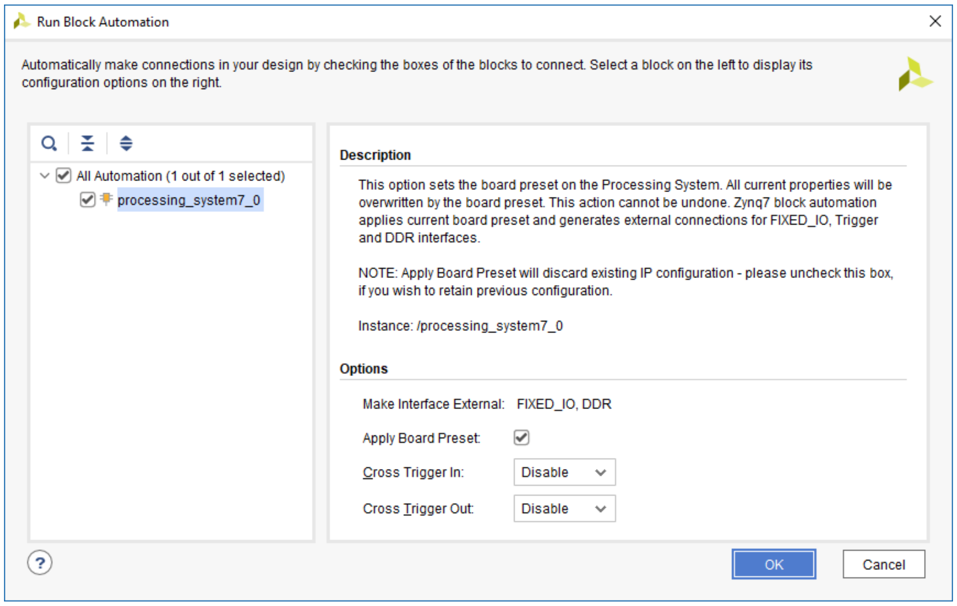


**Figure 9. Run block automation**

**2-1-6.** In the *Run Block Automation* window, leave the default settings, including *Apply Board Preset* checked, and click **OK**


Run Block Automation

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

All Automation (1 out of 1 selected)
processing_system7_0

**Description**

This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone. Zynq7 block automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces.

NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

Instance: /processing_system7_0

**Options**

Make Interface External: FIXED_IO, DDR

Apply Board Preset: ✓

Cross Trigger In: Disable

Cross Trigger Out: Disable

**Figure 10. Block Automation settings**

Once Block Automation has been completed, notice that ports have been automatically added for the DDR and Fixed IO, and some additional ports are now visible. The imported configuration for the Zynq related to the Zybo board has been applied which will now be modified.

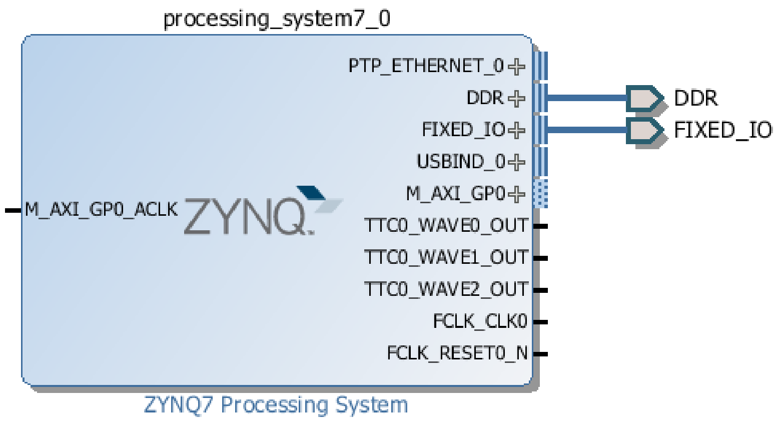


**Figure 11. Zynq Block with DDR and Fixed IO ports**

**2-1-7.** Double-click on the added block to open its *Customization* window.

Notice now the *Customization* window shows selected peripherals (with tick marks). This is the default configuration for the board applied by the block automation.

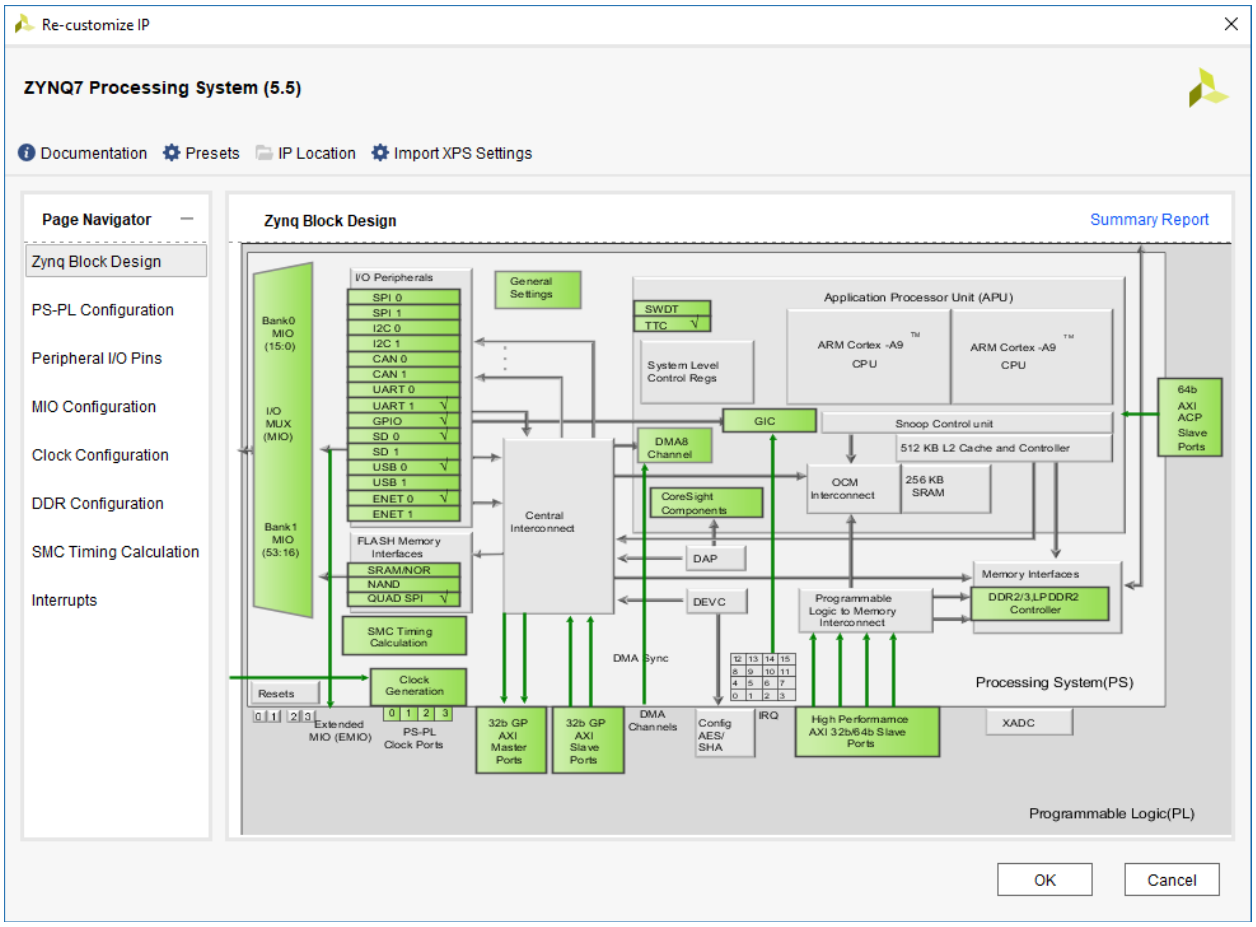


**Figure 12. Imported peripherals settings**

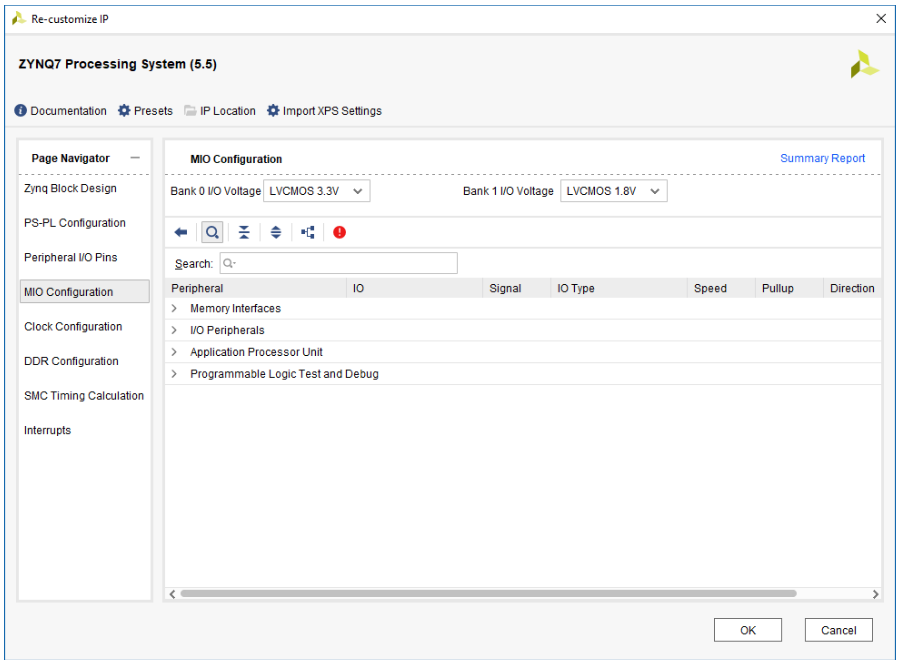
## 2-2. Configure the processing block with just UART 1 peripheral enabled.

**2-2-1.** A block diagram of the Zynq should now be open again, showing various configurable blocks of the Processing System.

At this stage, the designer can click on various configurable blocks (highlighted in green) and change the system configuration.

Only the UART is required for this lab, so all other peripherals will be deselected.

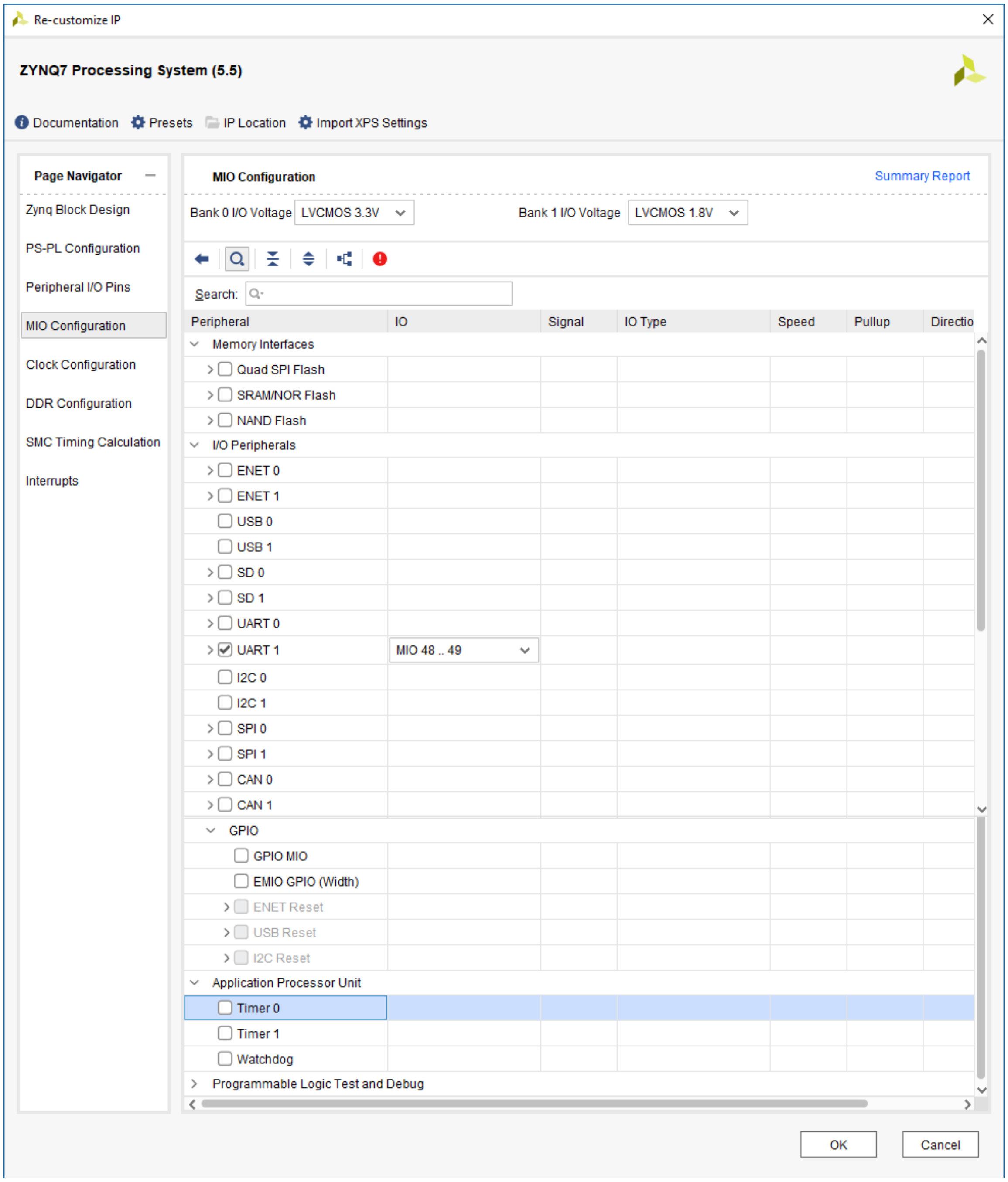
**2-2-2.** Click on one of the peripherals (in green) in the *I/O Peripherals* block, or select the *MIO Configuration* tab on the left to open the configuration form



Bank 0 I/O Voltage LVCMOS 3.3V
Bank 1 I/O Voltage LVCMOS 1.8V

**2-2-3.** Expand I/O peripherals if necessary, and ensure all the following *I/O peripherals are deselected* except *UART 1.*

i.e. Remove: *ENET 0*
*USB 0*
*SD 0*
Expand **GPIO** to deselect *GPIO MIO*
Expand **Memory Interfaces** to deselect *Quad SPI Flash*
Expand **Application Processor Unit** to disable *Timer 0*.


Re-customize IP

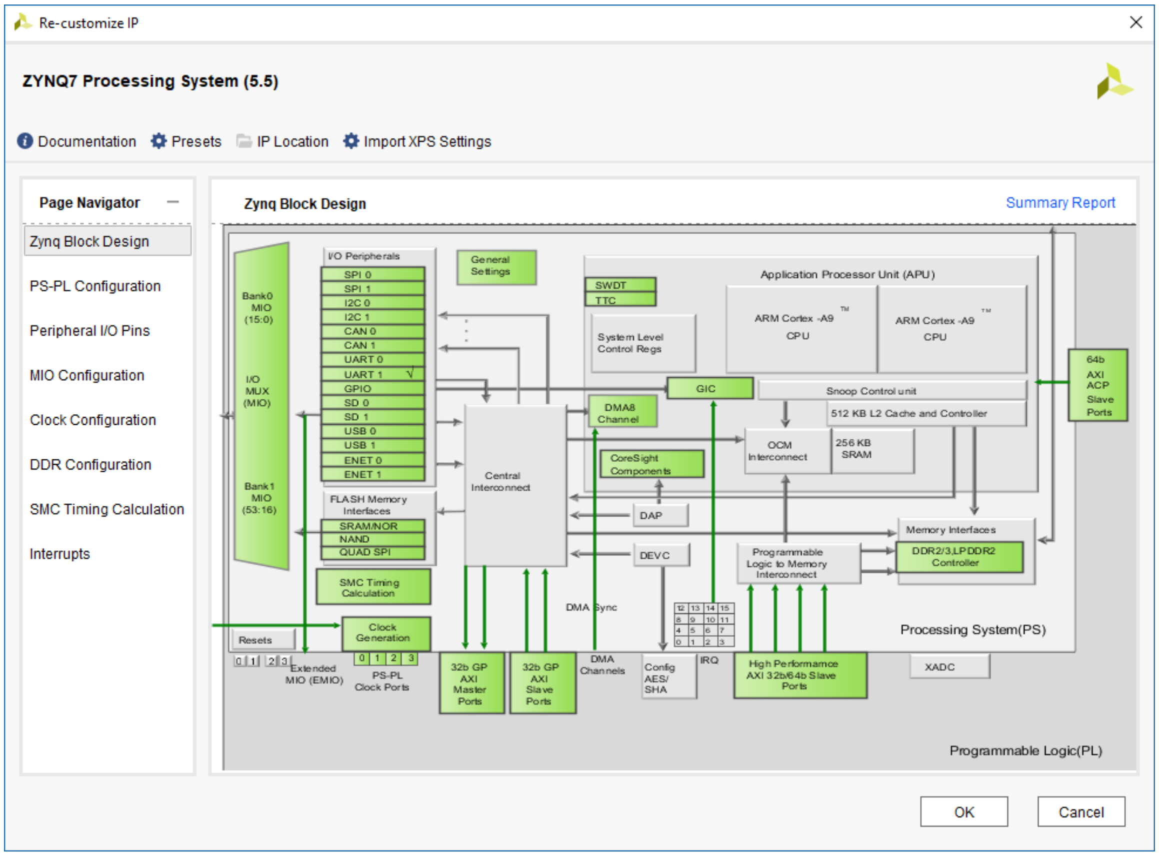
ZYNQ7 Processing System (5.5)

MIO Configuration

Bank 0 I/O Voltage LVCMOS 3.3V    Bank 1 I/O Voltage LVCMOS 1.8V

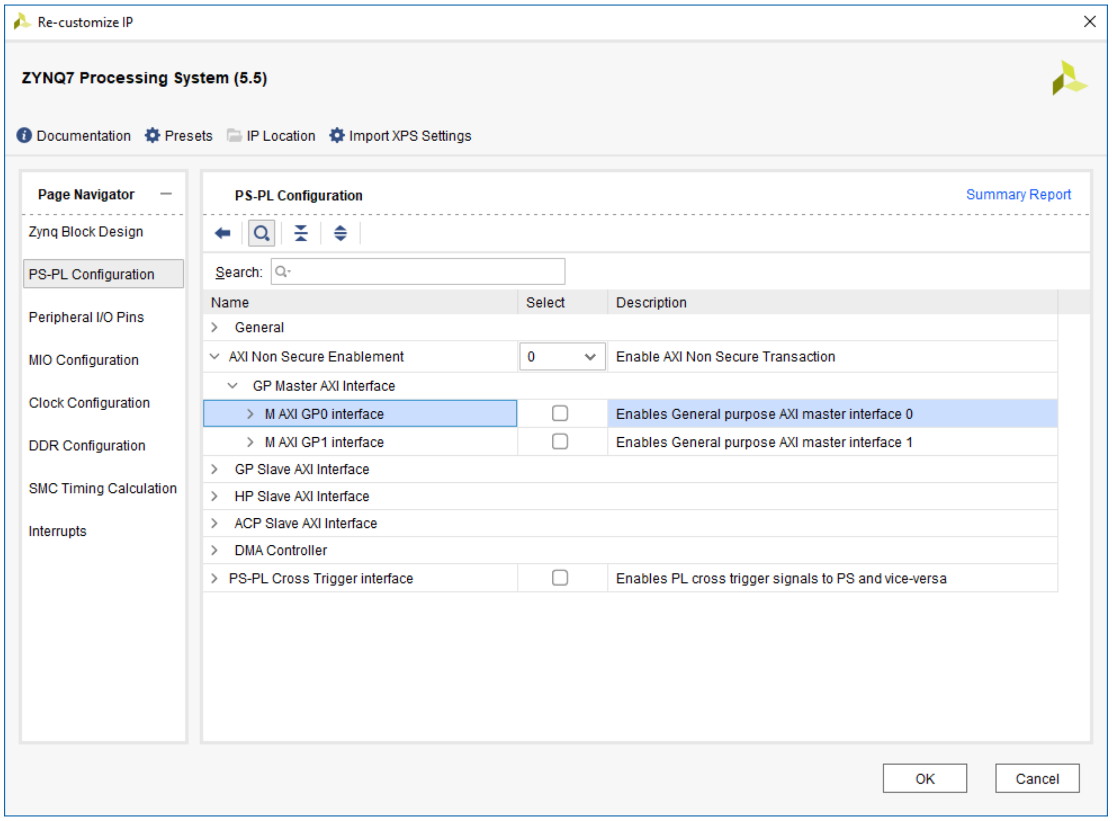
| Peripheral | IO | Signal | IO Type | Speed | Pullup | Directio |
|---|---|---|---|---|---|---|
| Memory Interfaces | | | | | | |
| ☐ Quad SPI Flash | | | | | | |
| ☐ SRAM/NOR Flash | | | | | | |
| ☐ NAND Flash | | | | | | |
| I/O Peripherals | | | | | | |
| ☐ ENET 0 | | | | | | |
| ☐ ENET 1 | | | | | | |
| ☐ USB 0 | | | | | | |
| ☐ USB 1 | | | | | | |
| ☐ SD 0 | | | | | | |
| ☐ SD 1 | | | | | | |
| ☐ UART 0 | | | | | | |
| ☑ UART 1 | MIO 48 .. 49 | | | | | |
| ☐ I2C 0 | | | | | | |
| ☐ I2C 1 | | | | | | |
| ☐ SPI 0 | | | | | | |
| ☐ SPI 1 | | | | | | |
| ☐ CAN 0 | | | | | | |
| ☐ CAN 1 | | | | | | |
| GPIO | | | | | | |
| ☐ GPIO MIO | | | | | | |
| ☐ EMIO GPIO (Width) | | | | | | |
| ☐ ENET Reset | | | | | | |
| ☐ USB Reset | | | | | | |
| ☐ I2C Reset | | | | | | |
| Application Processor Unit | | | | | | |
| ☐ Timer 0 | | | | | | |
| ☐ Timer 1 | | | | | | |
| ☐ Watchdog | | | | | | |
| Programmable Logic Test and Debug | | | | | | |

**Figure 13. Selecting only UART 1**



**2-2-4.** Select the **PS-PL Configuration** tab on the left.

**2-2-5.** Expand *AXI Non Secure Enablement* > *GP Master AXI interface* and deselect **M AXI GP0** interface.


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

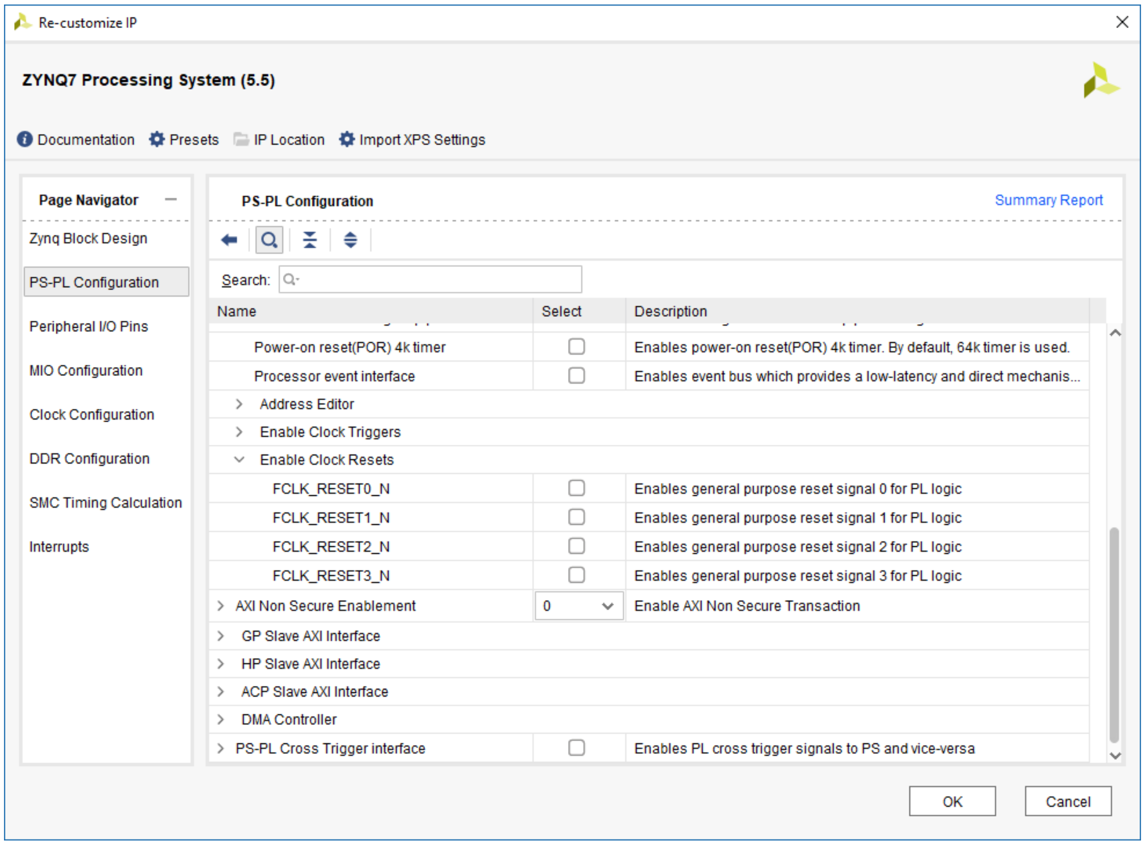
PS-PL Configuration — Summary Report

Search:

| Name | Select | Description |
|---|---|---|
| > General | | |
| ∨ AXI Non Secure Enablement | 0 | Enable AXI Non Secure Transaction |
| ∨ GP Master AXI Interface | | |
| > M AXI GP0 interface | ☐ | Enables General purpose AXI master interface 0 |
| > M AXI GP1 interface | ☐ | Enables General purpose AXI master interface 1 |
| > GP Slave AXI Interface | | |
| > HP Slave AXI Interface | | |
| > ACP Slave AXI Interface | | |
| > DMA Controller | | |
| > PS-PL Cross Trigger interface | ☐ | Enables PL cross trigger signals to PS and vice-versa |

OK | Cancel

**2-2-6.** Expand **General > Enable Clock Resets** and deselect the **FCLK_RESET0_N** option.


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

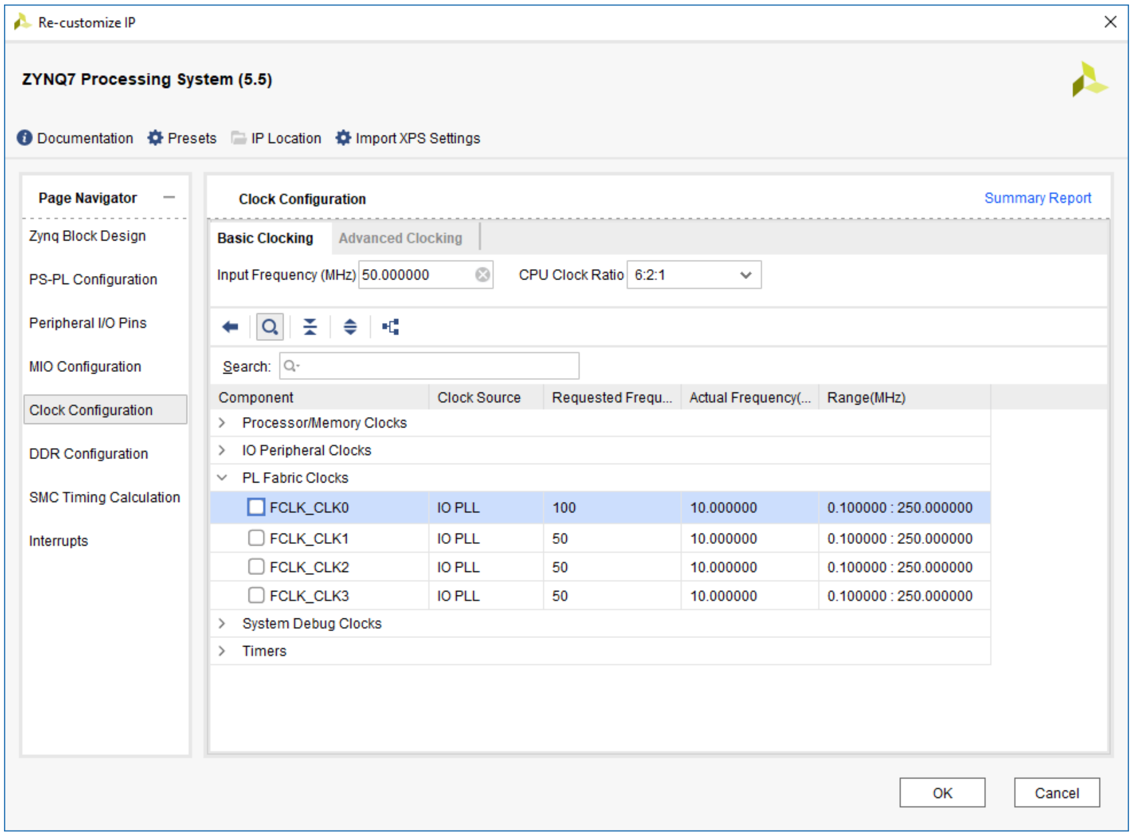
PS-PL Configuration — Summary Report

Search:

| Name | Select | Description |
|---|---|---|
| Power-on reset(POR) 4k timer | ☐ | Enables power-on reset(POR) 4k timer. By default, 64k timer is used. |
| Processor event interface | ☐ | Enables event bus which provides a low-latency and direct mechanis... |
| > Address Editor | | |
| > Enable Clock Triggers | | |
| ∨ Enable Clock Resets | | |
| FCLK_RESET0_N | ☐ | Enables general purpose reset signal 0 for PL logic |
| FCLK_RESET1_N | ☐ | Enables general purpose reset signal 1 for PL logic |
| FCLK_RESET2_N | ☐ | Enables general purpose reset signal 2 for PL logic |
| FCLK_RESET3_N | ☐ | Enables general purpose reset signal 3 for PL logic |
| > AXI Non Secure Enablement | 0 | Enable AXI Non Secure Transaction |
| > GP Slave AXI Interface | | |
| > HP Slave AXI Interface | | |
| > ACP Slave AXI Interface | | |
| > DMA Controller | | |
| > PS-PL Cross Trigger interface | ☐ | Enables PL cross trigger signals to PS and vice-versa |

OK | Cancel

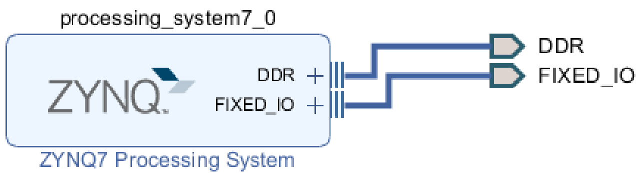
**2-2-7.** Select the **Clock Configuration** tab on the left. Expand the **PL Fabric Clocks** and deselect the **FCLK_CLK0** option and click **OK.**



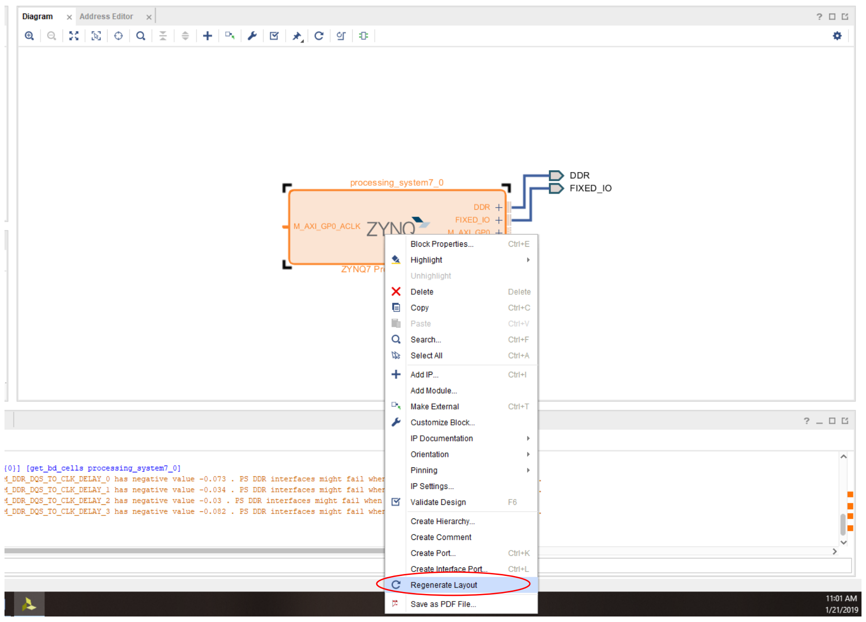
Input Frequency (MHz) 50.000000 CPU Clock Ratio 6:2:1

| Component | Clock Source | Requested Frequ... | Actual Frequency(... | Range(MHz) |
|---|---|---|---|---|
| › Processor/Memory Clocks | | | | |
| › IO Peripheral Clocks | | | | |
| ⌄ PL Fabric Clocks | | | | |
| ☐ FCLK_CLK0 | IO PLL | 100 | 10.000000 | 0.100000 : 250.000000 |
| ☐ FCLK_CLK1 | IO PLL | 50 | 10.000000 | 0.100000 : 250.000000 |
| ☐ FCLK_CLK2 | IO PLL | 50 | 10.000000 | 0.100000 : 250.000000 |
| ☐ FCLK_CLK3 | IO PLL | 50 | 10.000000 | 0.100000 : 250.000000 |
| › System Debug Clocks | | | | |
| › Timers | | | | |

Then your block diagram should look like this:



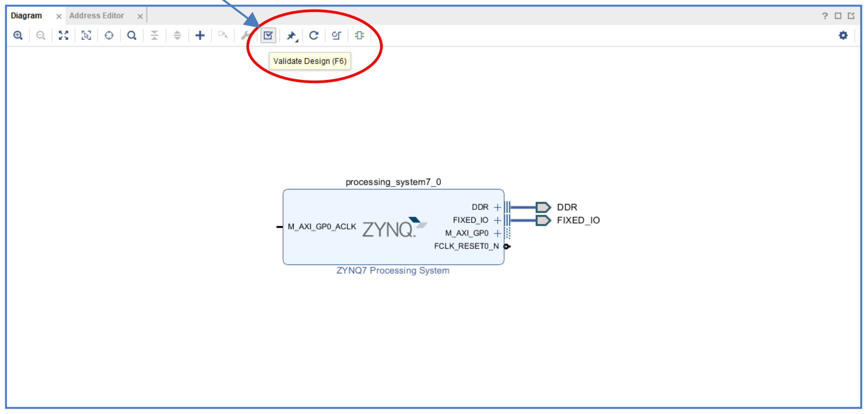
Right click the **Zynq7 Processing System** block on the Diagram window to bring up the options, then Click on the (Regenerate Layout) button

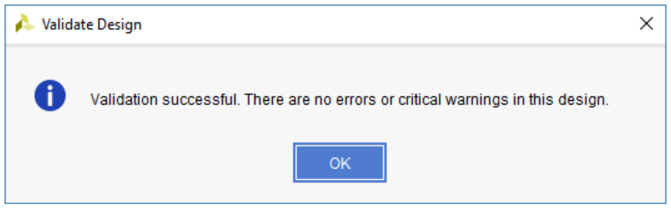


```
{0}] [get_bd_cells processing_system7_0]
M_DDR_DQS_TO_CLK_DELAY_0 has negative value -0.073 . PS DDR interfaces might fail wher
M_DDR_DQS_TO_CLK_DELAY_1 has negative value -0.034 . PS DDR interfaces might fail wher
M_DDR_DQS_TO_CLK_DELAY_2 has negative value -0.03 . PS DDR interfaces might fail when
M_DDR_DQS_TO_CLK_DELAY_3 has negative value -0.082 . PS DDR interfaces might fail wher
```

**Figure 14. Updated Zynq Block**

**2-2-8.** Click on the (Validate Design) button and make sure that there are no errors.

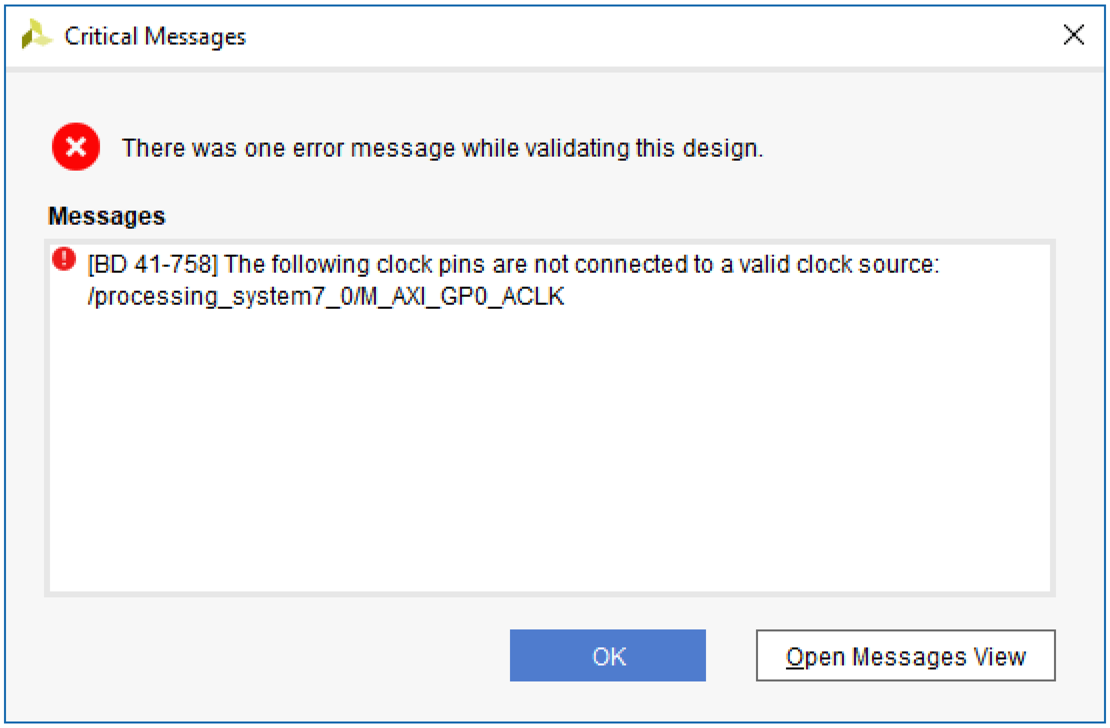




Validation successful. There are no errors or critical warnings in this design.

Click “OK”. If there is any error, go back to previous steps and make sure each step is done correctly.

Following is an example of an error case: It is due to 2-2-6 not done properly.



Critical Messages

There was one error message while validating this design.

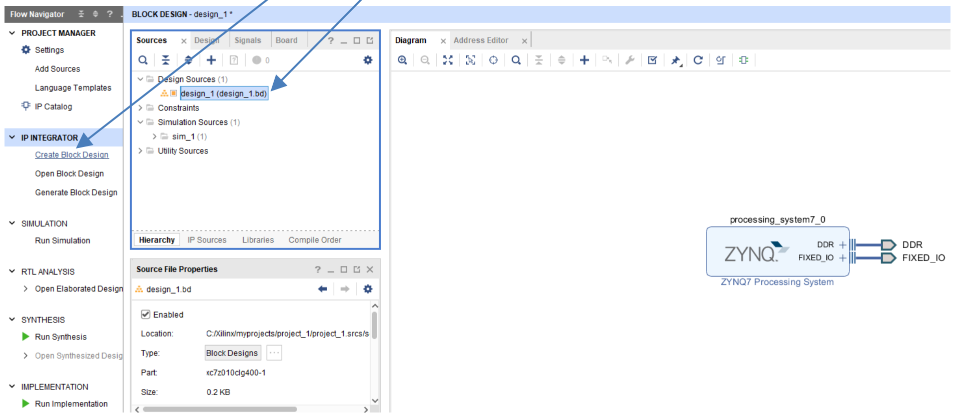
Messages

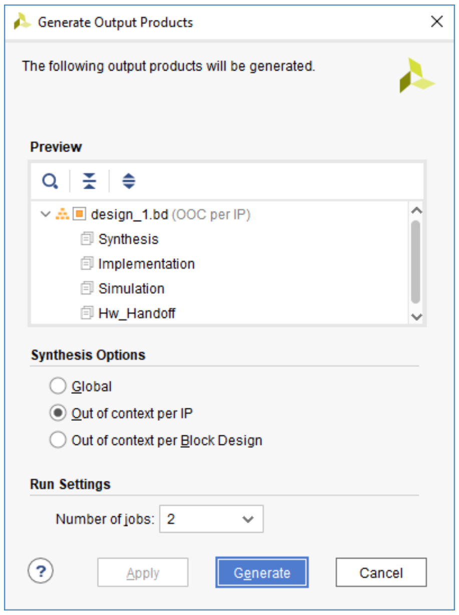
[BD 41-758] The following clock pins are not connected to a valid clock source:
/processing_system7_0/M_AXI_GP0_ACLK

## Generate Top-Level and Export to SDK Step 3

### 3-1. Generate IP Integrator Outputs, the top-level HDL, and start SDK by exporting the hardware.

**3-1-1.** In the sources panel, right-click on *design_1.bd*, and select **Generate Output Products …** and click **Generate** to generate the Implementation, Simulation and Synthesis files for the design (You can also click on **Generate Block Design** in the Flow Navigator pane to do the same)





Generate Output Products

The following output products will be generated.

Preview

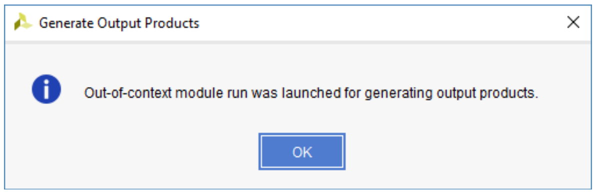
- design_1.bd (OOC per IP)
  - Synthesis
  - Implementation
  - Simulation
  - Hw_Handoff

Synthesis Options

- Global
- [✓ Out of context per IP]
- Out of context per Block Design

Run Settings

Number of jobs: 2



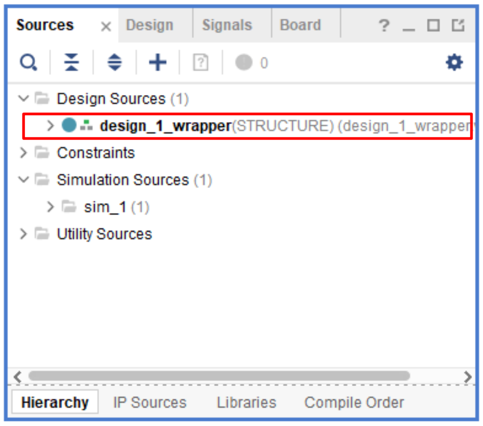
Generate Output Products

Out-of-context module run was launched for generating output products.

Click “OK”.

**3-1-2.** Right-click again on *design_1.bd*, and select **Create HDL Wrapper…** to generate the top-level VHDL model. Leave the *Let Vivado manager wrapper and auto-update* option selected, and click **OK**

The *design_1_wrapper.v* file will be created and added to the project. Double-click on the file to see the content in the Auxiliary pane.



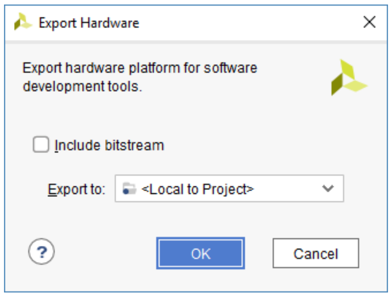
- Design Sources (1)
  - **design_1_wrapper**(STRUCTURE) (design_1_wrapper
- Constraints
- Simulation Sources (1)
  - sim_1 (1)
- Utility Sources

**Figure 15. The HDL Wrapper file generated and added to the project**

**3-1-3.** Notice that the Verilog file is already *Set As the Top* module in the design, indicated by the icon

**3-1-4.** Select **File > Export > Export hardware** and click **OK.** (*Save* the project if prompted)

Note: Since we do not have any hardware in Programmable Logic (PL) there is no bitstream to generate, hence the *Include bitstream* option is not necessary at this time.


Export Hardware

Export hardware platform for software development tools.

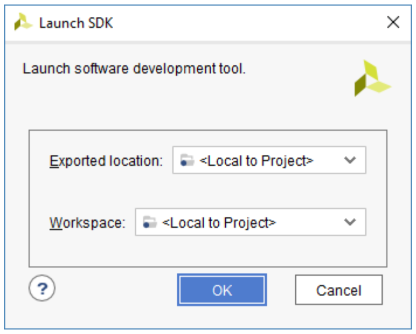
☐ Include bitstream

Export to: <Local to Project>

OK Cancel

**Figure 16. Exporting hardware**

**3-1-5.** Select **File > Launch SDK** leaving the default settings, and click **OK**


Launch SDK

Launch software development tool.

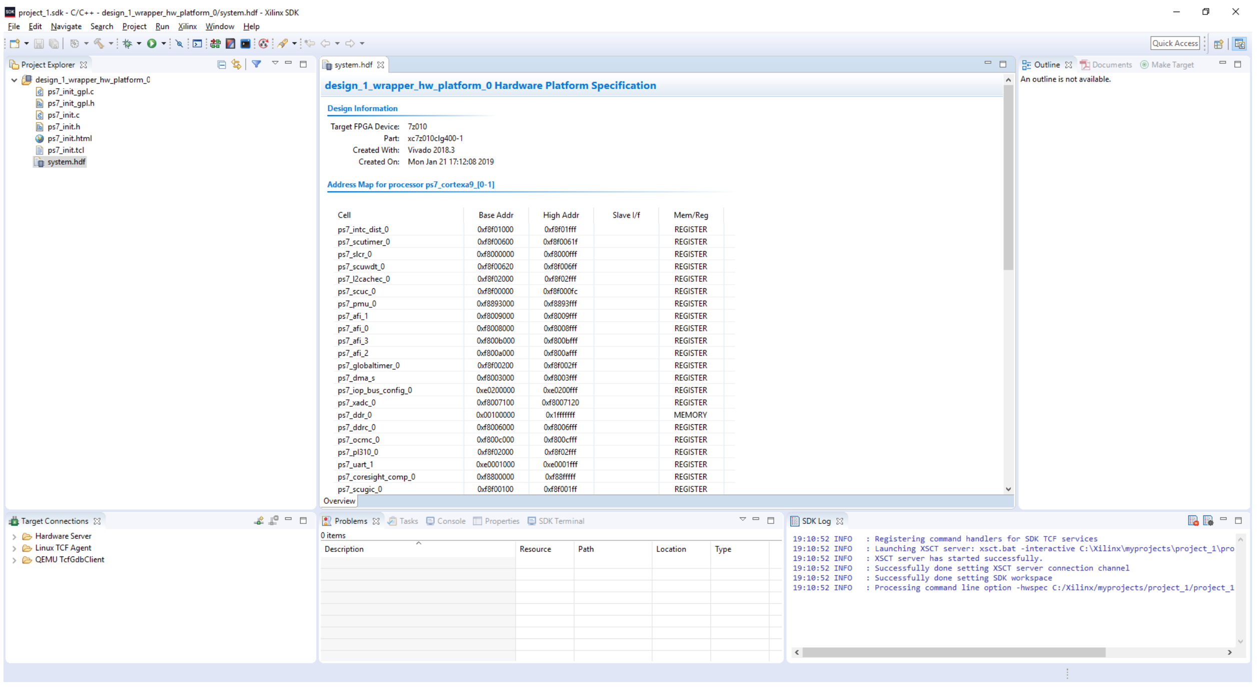
Exported location: <Local to Project>

Workspace: <Local to Project>

OK Cancel

**Figure 17. Launch SDK**

SDK should now be open. If only the Welcome panel is visible, close or minimize this panel to view the *Project Explorer* and *Preview* panel. A Hardware platform project should have been automatically created, and the *design_1_wrapper_hw_platform_0* folder should exist in the Project Explorer panel.


project_1.sdk - C/C++ - design_1_wrapper_hw_platform_0/system.hdf - Xilinx SDK

File Edit Navigate Search Project Run Xilinx Window Help
Project Explorer
system.hdf
Outline Documents Make Target
Target Connections
Problems Tasks Console Properties SDK Terminal
SDK Log
Overview

design_1_wrapper_hw_platform_0
- ps7_init_gpl.c
- ps7_init_gpl.h
- ps7_init.c
- ps7_init.h
- ps7_init.html
- ps7_init.tcl
- system.hdf

An outline is not available.

**design_1_wrapper_hw_platform_0 Hardware Platform Specification**

**Design Information**

| | |
|---|---|
| Target FPGA Device: | 7z010 |
| Part: | xc7z010clg400-1 |
| Created With: | Vivado 2018.3 |
| Created On: | Mon Jan 21 17:12:08 2019 |

**Address Map for processor ps7_cortexa9_[0-1]**

| Cell | Base Addr | High Addr | Slave I/f | Mem/Reg |
|---|---|---|---|---|
| ps7_intc_dist_0 | 0xf8f01000 | 0xf8f01fff | | REGISTER |
| ps7_scutimer_0 | 0xf8f00600 | 0xf8f0061f | | REGISTER |
| ps7_slcr_0 | 0xf8000000 | 0xf8000fff | | REGISTER |
| ps7_scuwdt_0 | 0xf8f00620 | 0xf8f006ff | | REGISTER |
| ps7_l2cachec_0 | 0xf8f02000 | 0xf8f02fff | | REGISTER |
| ps7_scuc_0 | 0xf8f00000 | 0xf8f000fc | | REGISTER |
| ps7_pmu_0 | 0xf8893000 | 0xf8893fff | | REGISTER |
| ps7_afi_1 | 0xf8009000 | 0xf8009fff | | REGISTER |
| ps7_afi_0 | 0xf8008000 | 0xf8008fff | | REGISTER |
| ps7_afi_3 | 0xf800b000 | 0xf800bfff | | REGISTER |
| ps7_afi_2 | 0xf800a000 | 0xf800afff | | REGISTER |
| ps7_globaltimer_0 | 0xf8f00200 | 0xf8f002ff | | REGISTER |
| ps7_dma_s | 0xf8003000 | 0xf8003fff | | REGISTER |
| ps7_iop_bus_config_0 | 0xe0200000 | 0xe0200fff | | REGISTER |
| ps7_xadc_0 | 0xf8007100 | 0xf8007120 | | REGISTER |
| ps7_ddr_0 | 0x00100000 | 0x1fffffff | | MEMORY |
| ps7_ddrc_0 | 0xf8006000 | 0xf8006fff | | REGISTER |
| ps7_ocmc_0 | 0xf800c000 | 0xf800cfff | | REGISTER |
| ps7_pl310_0 | 0xf8f02000 | 0xf8f02fff | | REGISTER |
| ps7_uart_1 | 0xe0001000 | 0xe0001fff | | REGISTER |
| ps7_coresight_comp_0 | 0xf8800000 | 0xf88fffff | | REGISTER |
| ps7_scugic_0 | 0xf8f00100 | 0xf8f001ff | | REGISTER |

Target Connections
- Hardware Server
- Linux TCF Agent
- QEMU TcfGdbClient

0 items

| Description | Resource | Path | Location | Type |
|---|---|---|---|---|

```
19:10:52 INFO  : Registering command handlers for SDK TCF services
19:10:52 INFO  : Launching XSCT server: xsct.bat -interactive C:\Xilinx\myprojects\project_1\pro
19:10:52 INFO  : XSCT server has started successfully.
19:10:52 INFO  : Successfully done setting XSCT server connection channel
19:10:52 INFO  : Successfully done setting SDK workspace
19:10:52 INFO  : Processing command line option -hwspec C:/Xilinx/myprojects/project_1/project_1
```

**Figure 18. SDK C/C++ development view**

The system.hdf file (Hardware Description File) for the Hardware platform should open in the preview pane. Double click system.hdf to open it if it is not.

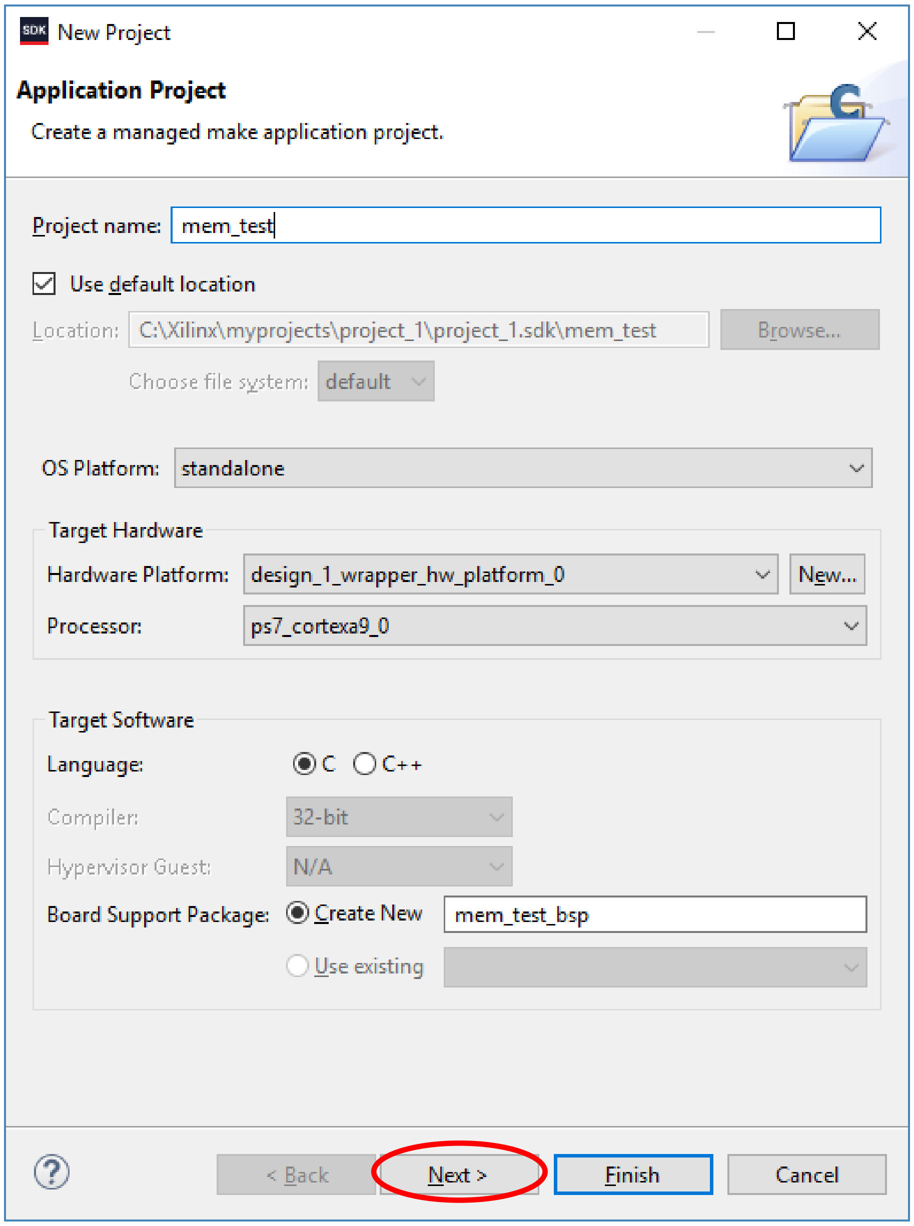
Basic information about the hardware configuration of the project can be found in the .hdf file, along with the Address maps for the PS systems, and driver information. The .hdf file is used in the software environment to determine the peripherals available in the system, and their location in the address map.

## Generate Memory TestApp in SDK Step 4

### 4-1. Generate memory test application using one of the standard projects template.

**4-1-1.** In SDK, select **File** > **New** > **Application Project**

**4-1-2.** Name the project **mem_test**, and in the *Board Support Package* section, leave *Create New* selected and leave the default name *mem_test_bsp* and click **Next.** (Note that this application will run on ps7_cortexa9_0 i.e. core 0 of the two processor cores available.)


New Project

Application Project

Create a managed make application project.

Project name: mem_test

[✓] Use default location

Location: C:\Xilinx\myprojects\project_1\project_1.sdk\mem_test

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++

Compiler: 32-bit

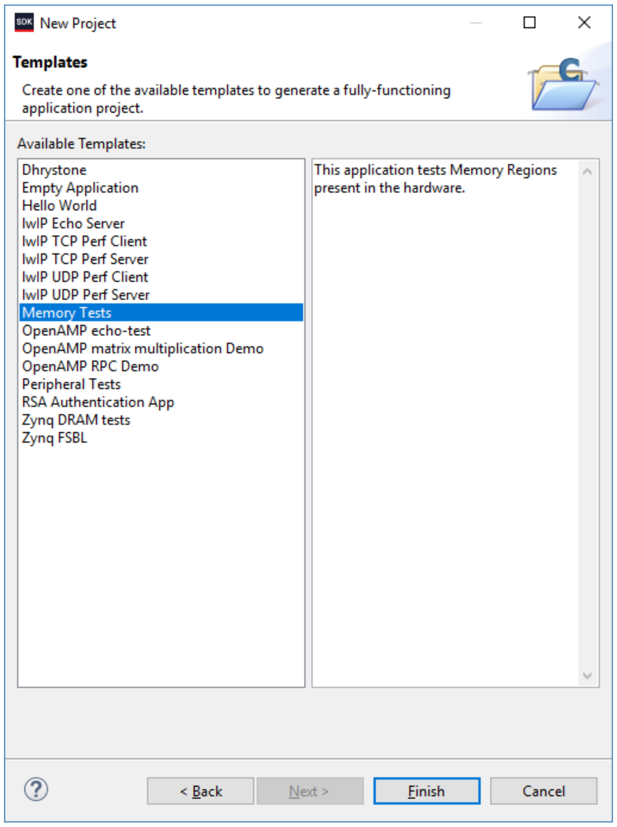
Hypervisor Guest: N/A

Board Support Package: [✓ Create New] mem_test_bsp

Use existing

**Figure 19. Create new SDK application project**

**4-1-3.** Select **Memory Tests** from the *Available Templates* window, and click **Finish.**



New Project

**Templates**

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

Available Templates:

Dhrystone
Empty Application
Hello World
lwIP Echo Server
lwIP TCP Perf Client
lwIP TCP Perf Server
lwIP UDP Perf Client
lwIP UDP Perf Server
Memory Tests
OpenAMP echo-test
OpenAMP matrix multiplication Demo
OpenAMP RPC Demo
Peripheral Tests
RSA Authentication App
Zynq DRAM tests
Zynq FSBL

This application tests Memory Regions present in the hardware.

**Figure 20. Creating Memory Tests C Project**

The **mem_test** project and the board support project **mem_test_bsp** will be created and will be visible in the Project Explorer window of SDK, and the two projects will be automatically built. You can monitor the progress in the Console panel.

**4-1-4.** Expand folders in the Project Explorer view, and observe that there are three projects – *design_1_wrapper_hw_platform_0*, *mem_test_bsp*, and *mem_test*. The *mem_test* project is the application that we will use to verify the functionality of the design. The *hw_platform* includes the *ps7_init* function which initializes the PS as part of the first stage bootloader, and mem_test_bsp is the board support package.

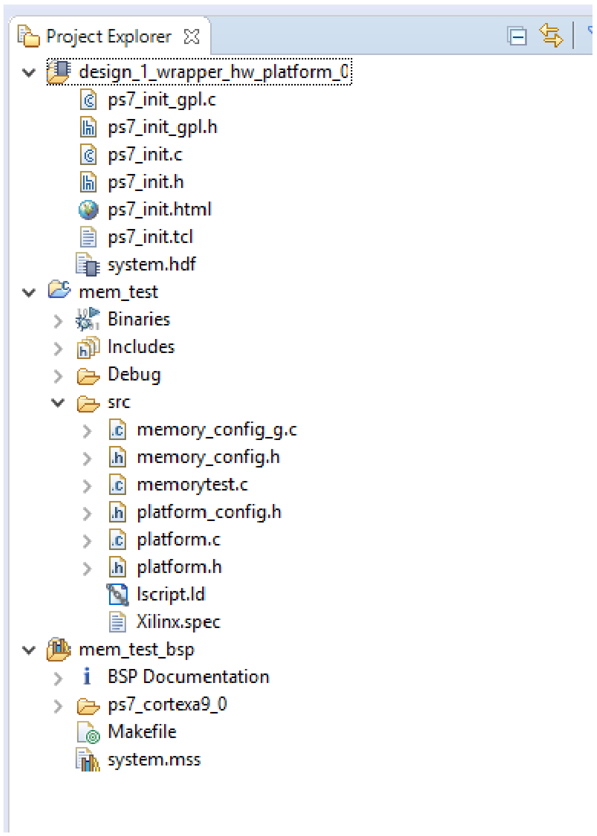


**Figure 21. The Project Explore view**

**4-1-5.** Open the **memorytest.c** file in the mem_test project (under *src*), and examine the contents. This file calls the functions to test the memory.

## Test in Hardware Step 5

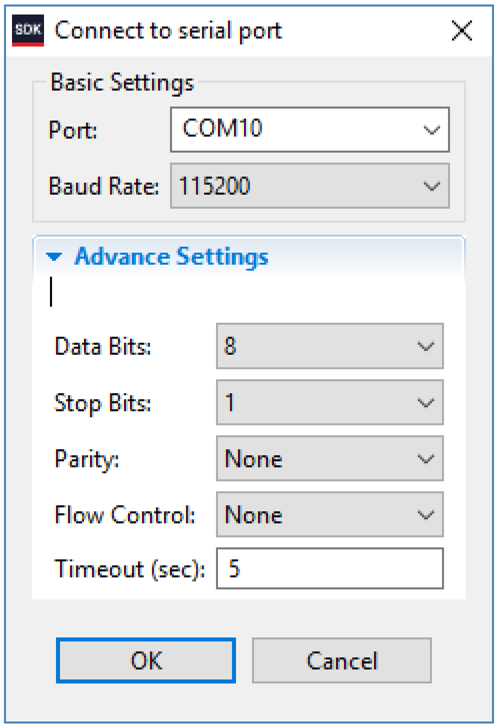
**5-1. Zybo: Make sure that the JP7 is set to select USB power and JP5 is set to JTAG mode. Connect the board with a micro-usb cable and power it ON.**

**Establish the serial communication using SDK's Terminal tab.**

**5-1-1.** Zybo: Make sure that the JP7 is set to select USB power, and JP5 is set to JTAG. Make sure that a micro-USB cable is connected to the JTAG PROG connector (next to the power supply connector). Turn ON the power.

**5-1-2.** Select the SDK Terminal tab. If it is not visible then select **Window > Show view > Terminal**.

**5-1-3.** Click on ✚ and if required, select appropriate COM port (depends on your computer), and configure it with the parameters as shown.



Connect to serial port

Basic Settings

Port: COM10

Baud Rate: 115200

Advance Settings

Data Bits: 8

Stop Bits: 1

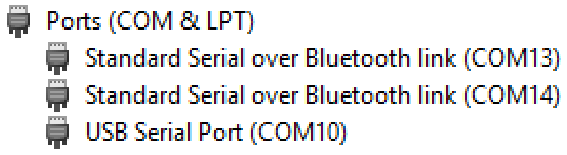
Parity: None

Flow Control: None

Timeout (sec): 5

**Figure 22. SDK Terminal Settings**

You can find the COM port from the Windows Device Manager, in this case, COM10:



Ports (COM & LPT)
- Standard Serial over Bluetooth link (COM13)
- Standard Serial over Bluetooth link (COM14)
- USB Serial Port (COM10)

**Figure 23. COM ports in Windows control panel**

## 5-2. Run the mem_test application and verify the functionality.

**5-2-1.** In SDK, select the **mem_test** project in *Project Explorer*, right-click and select **Run As > Launch on Hardware (GDB)** to download the application, and will execute ps7_init, and then execute mem_test.elf (user application).

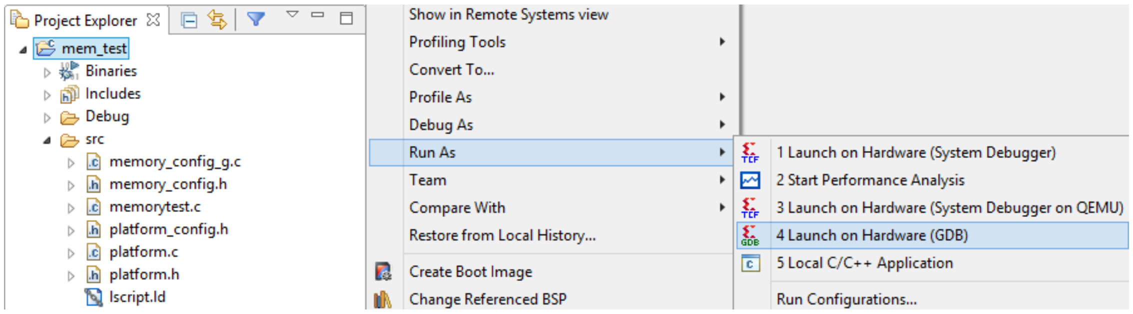
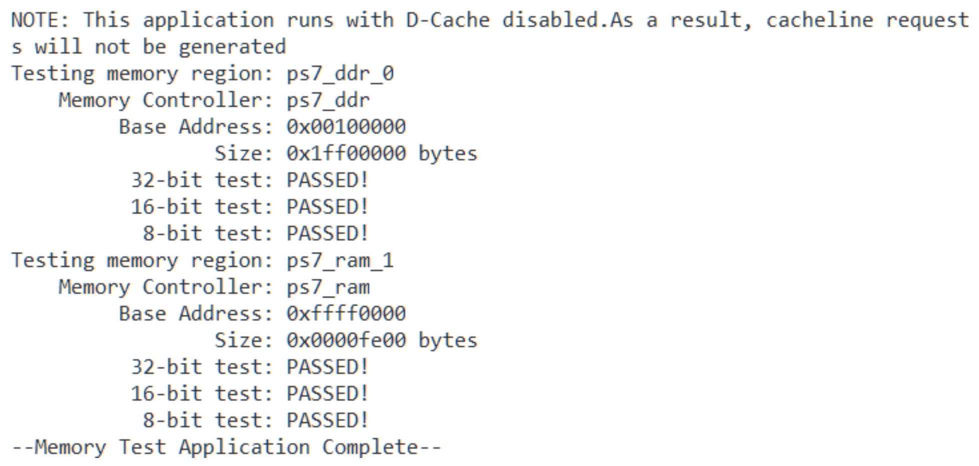


**Figure 24. Launch Application**

**5-2-2.** You should see the following output on the *Terminal* tab.



```
NOTE: This application runs with D-Cache disabled.As a result, cacheline request
s will not be generated
Testing memory region: ps7_ddr_0
    Memory Controller: ps7_ddr
         Base Address: 0x00100000
                 Size: 0x1ff00000 bytes
          32-bit test: PASSED!
          16-bit test: PASSED!
           8-bit test: PASSED!
Testing memory region: ps7_ram_1
    Memory Controller: ps7_ram
         Base Address: 0xffff0000
                 Size: 0x0000fe00 bytes
          32-bit test: PASSED!
          16-bit test: PASSED!
           8-bit test: PASSED!
--Memory Test Application Complete--
```

**Figure 25. SDK Terminal Output**

**5-2-3.** Close SDK and Vivado by selecting **File > Exit** in each program.

## Conclusion

Vivado and the IP Integrator allow base embedded processor systems and applications to be generated very quickly. After the system has been defined, the hardware can be exported and SDK can be invoked from Vivado. Software development is done in SDK which provides several application templates including memory tests. You verified the operation of the hardware by downloading a test application, executing on the processor, and observing the output in the serial terminal window.

Demonstrate the lab on your Zybo board to the instructor and submit your answers to the following questions to Beachboard. Submit it as a single file(word or pdf file) to Beachboard. Each team member is required to submit his/her own answers.

1. After which step we finished building the hardware platform for the embedded system?
2. Briefly describe the major components in the hardware platform.
3. What is the top-level description file for hardware platform? Copy the contents in this file that support the answer you provide in the previous question.
4. What does a bit stream file do? Is there a bit stream file generated for the hardware platform we built in this lab?
5. What is the step that transitions our design from hardware platform to software platform? What are the tools used for hardware design and software design respectively? What information are passed from hardware design tool to software design tool?
6. How many projects are created for building the software platform? Briefly describe the key design information provided in each one of them and the role each one of them play in the software platform.
7. What is a BSP? What information does it provide and what role does it play in building the software platform?
8. What is the top-level description file for software platform? Briefly describe the key information in this file and support your answer by copying the related contents in this file.
9. What is an .elf file? Is there an .elf file generated in this lab?
10. Please provide a screenshot of your embedded system output and block diagram of your hardware platform.