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. 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**).

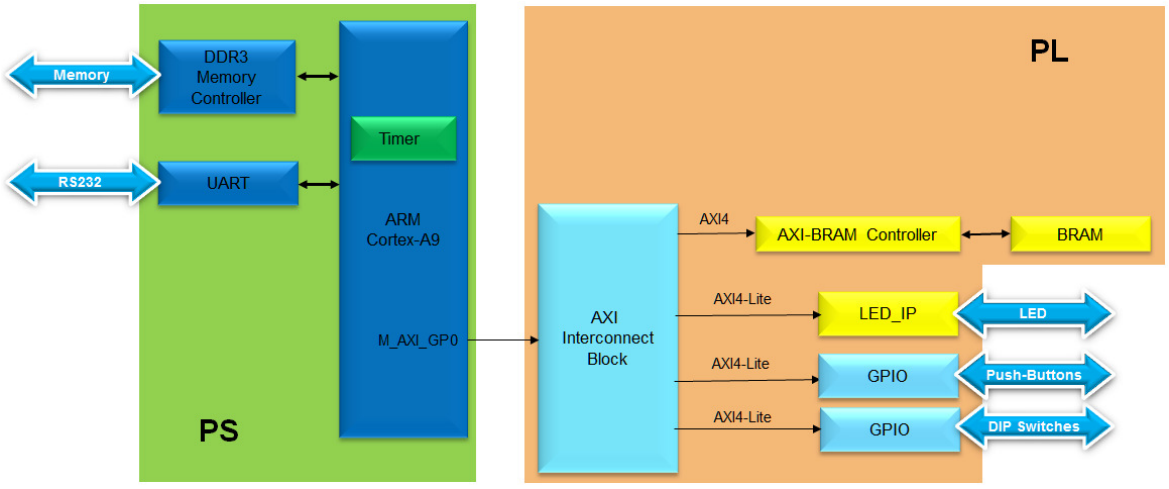


Figure 1. Completed Design

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

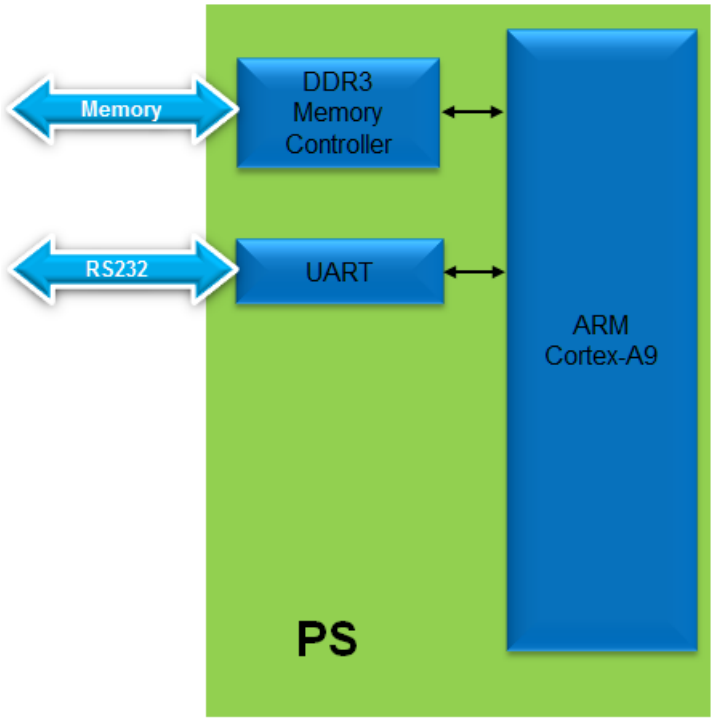
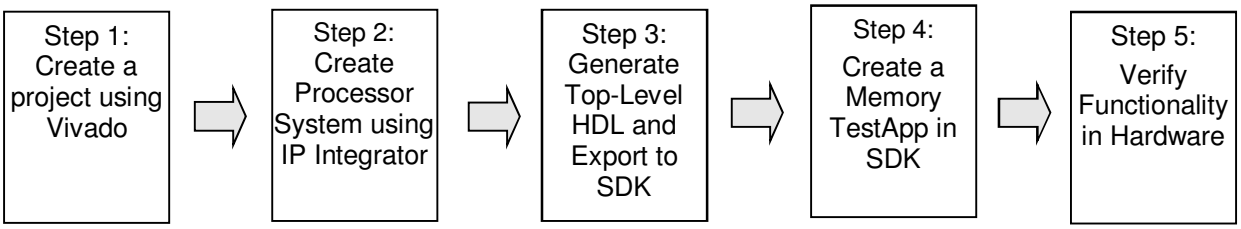


Figure 2. Processor Design of this Lab

General Flow for this Lab



Create a Vivado Project Step 1

1-1. Launch Vivado and create an empty project targeting the ZYBO (having xc7z010clg400-1 device) and using the VHDL language.

1-1-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2013.4 > Vivado 2013.4**

1-1-2. Click **Create New 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:\xup\embedded\labs**, and click **Select**.

1-1-4. Enter **lab1** in the *Project Name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.

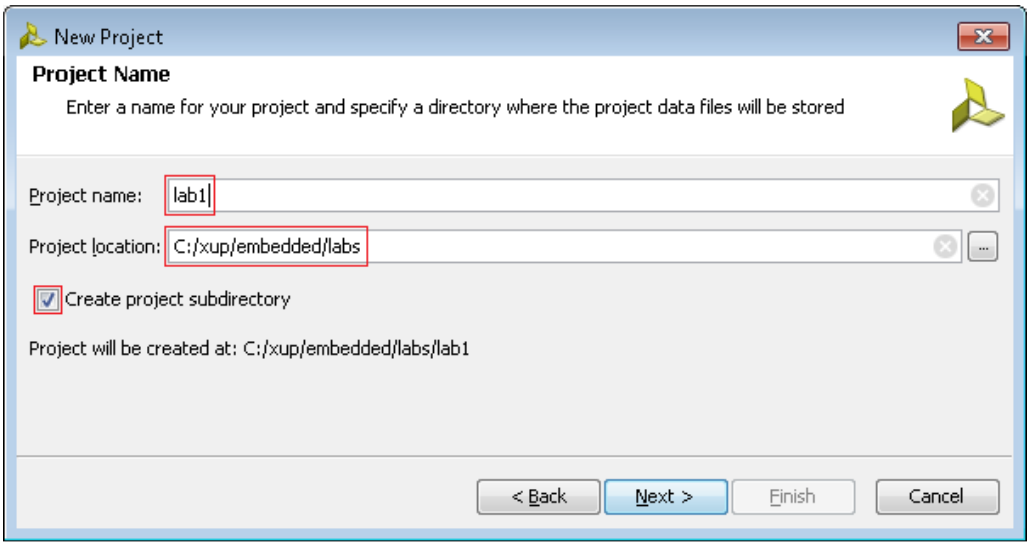


Figure 3. Project Name Entry

1-1-5. Select **RTL Project** in the *Project Type* form, and click **Next**.

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

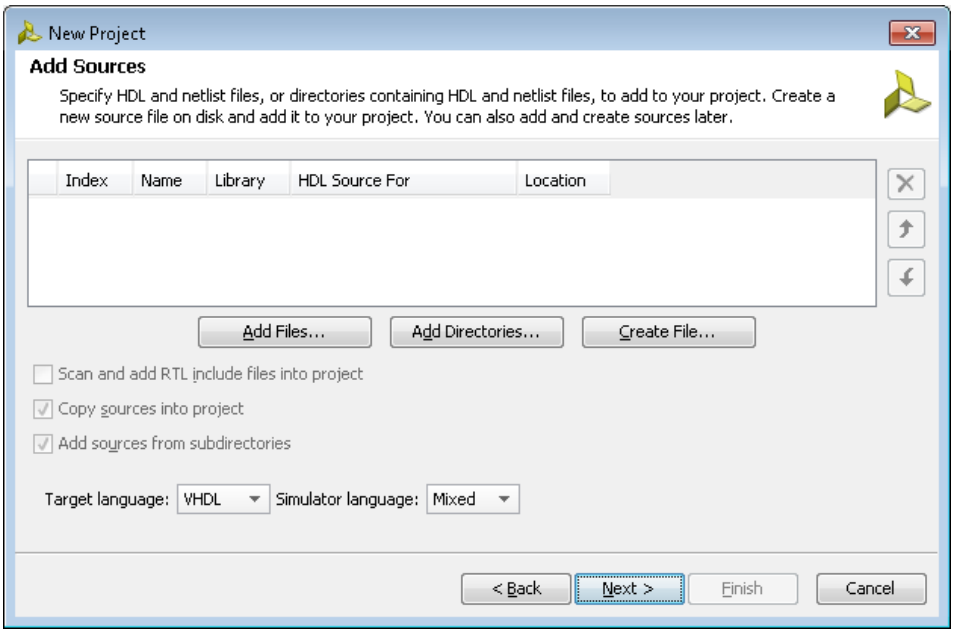


Figure 4. Add sources to new project

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 *Parts*, and using various filters shown in the figure below, select **xc7z010clg400-1** part as it is on the ZYBO board. Click **Next**.

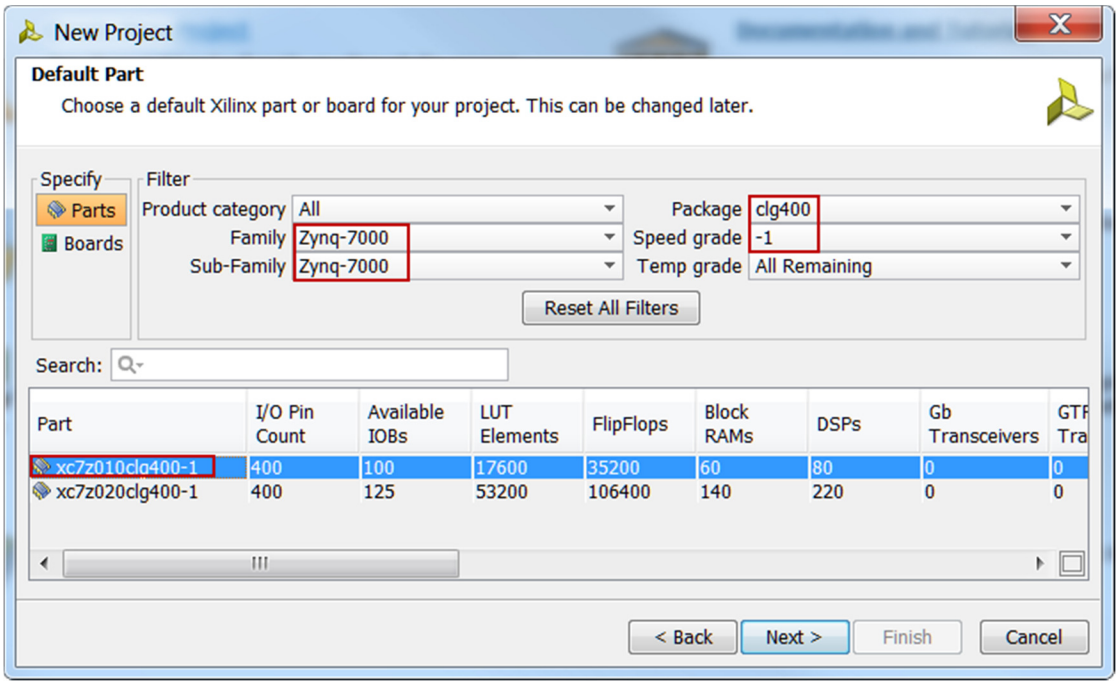


Figure 5. Boards and Parts Selection

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

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 ZYBO board.

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

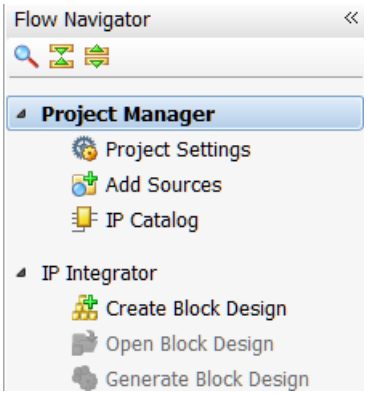


Figure 6. Create IP Integrator Block Diagram

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

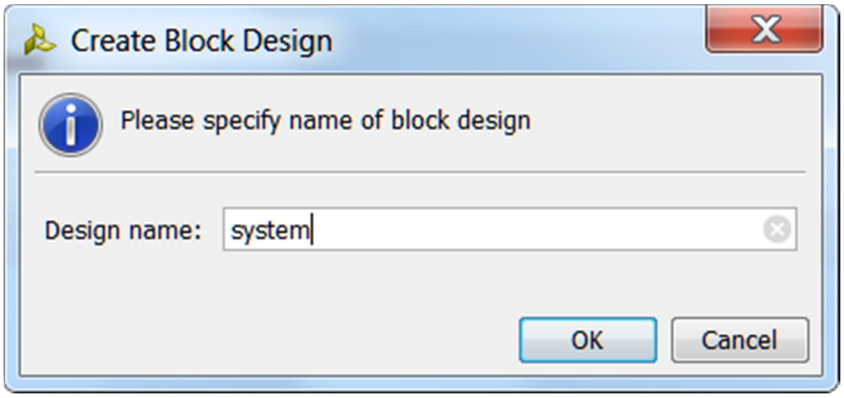


Figure 7. Create New Block Diagram

2-1-3. IP from the catalog can be added in different ways. Click on *Add IP* in the message at the top of the *Diagram* panel, or click the *Add IP icon* in the block diagram side bar, press Ctrl + I, or right-click anywhere in the Diagram workspace and select Add IP

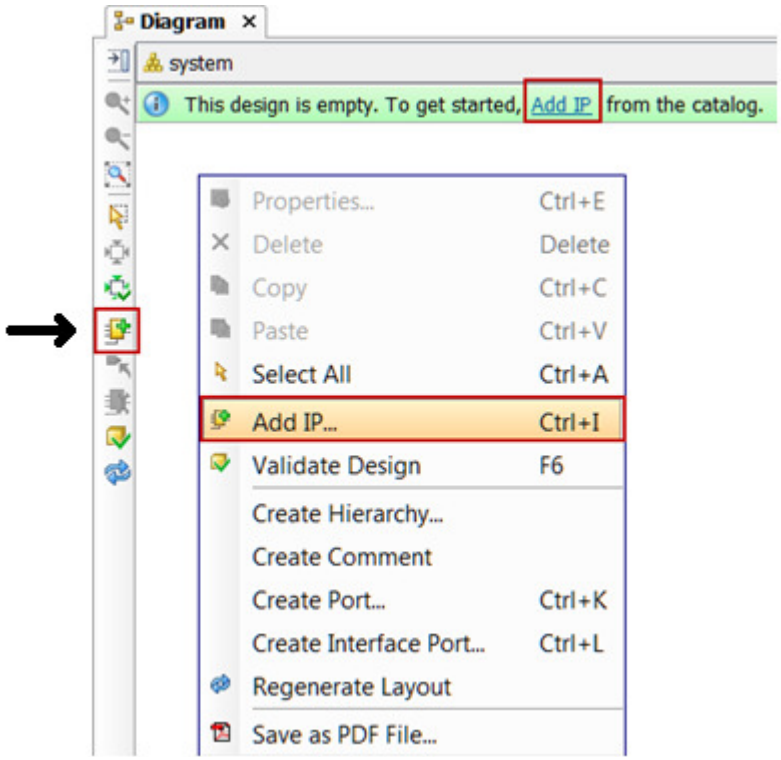


Figure 8. Add IP to Block Diagram

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.

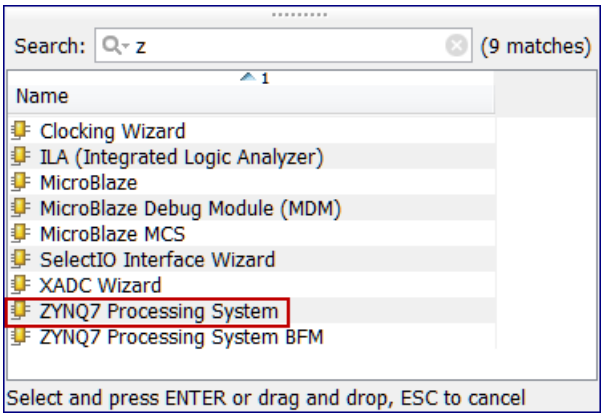


Figure 9. Add Zynq block to the design

2-1-5. Notice the message at the top of the Diagram window that *Designer Assistance* available.

2-1-6. Double-click on the added block to open its *Customization* window.

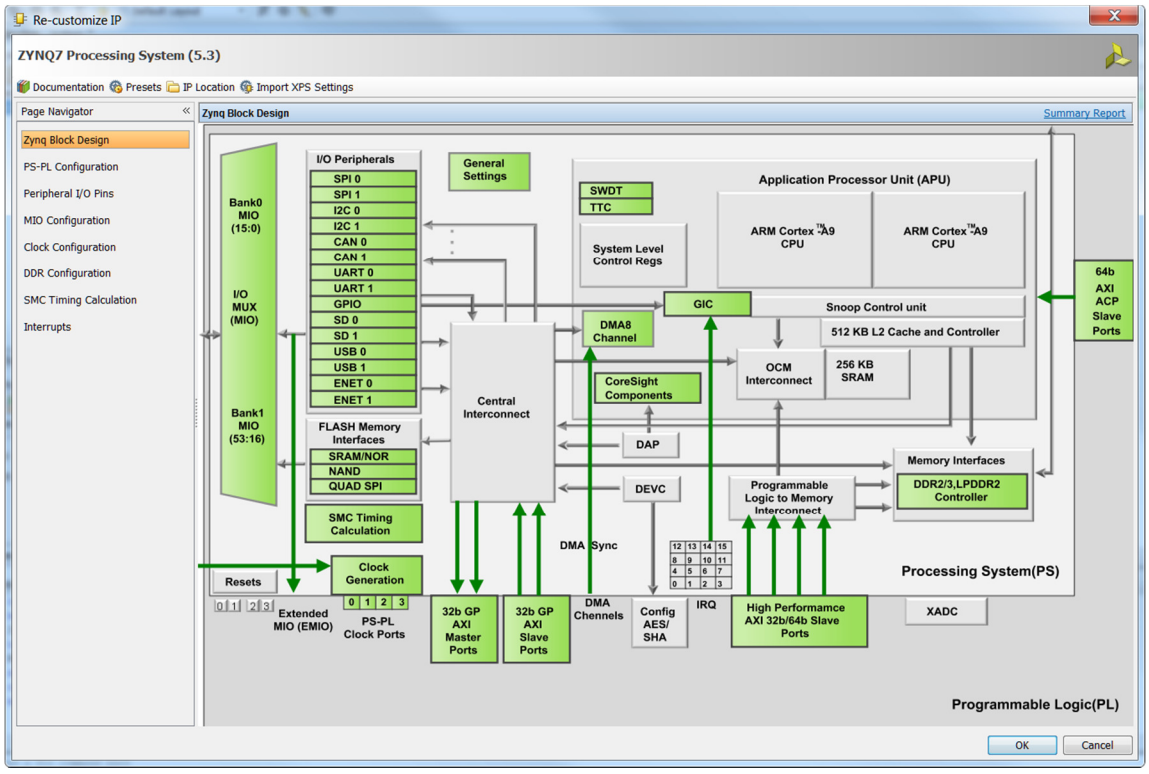


Figure 10. Default configuration form with no peripheral selected

2-1-7. Click on the **Import XPS Settings** button on the top tools bar to import the setting for the ZYBO board using the provided xml file.

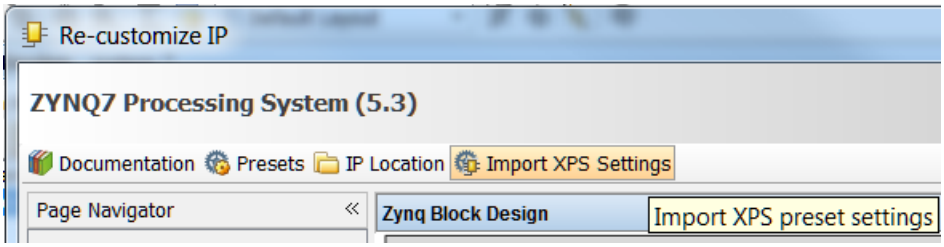


Figure 11. Importing xml file for the ZYBO board

2-1-8. Click on the browse button, browse to c:\xup\embedded\support, select the provided *ps7_system_prj.xml* file, and click **OK**.

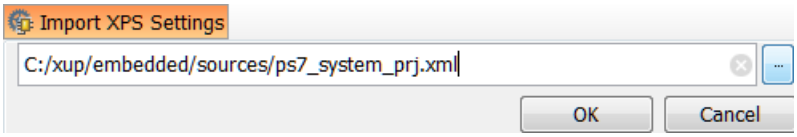


Figure 12. Selecting the xml file

2-1-9. Click **OK**.

Notice now the *Customization* window shows selected peripherals (with tick marks).

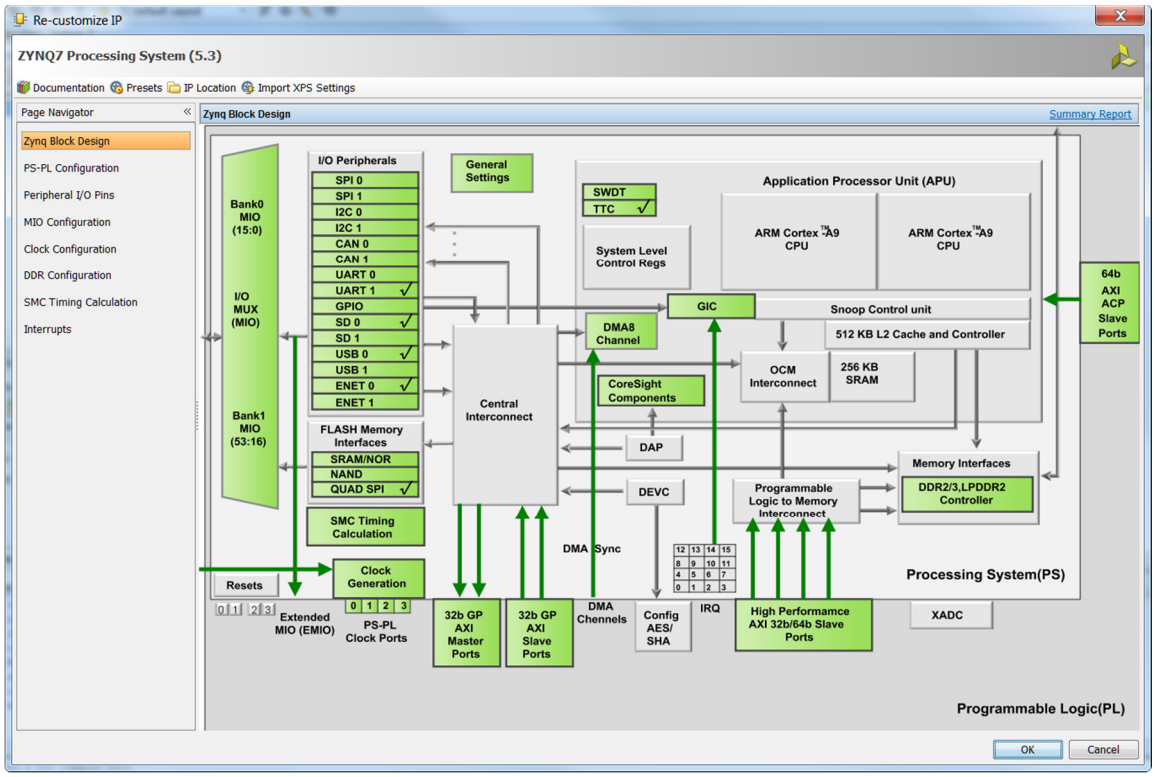


Figure 13. Imported peripherals settings

2-1-10. Click **OK** to close the *Customization* window for now.

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

2-2-1. Click on **Run Block Automation** and select /**processing_system7_1**

Figure 104. Designer Assistance message

2-2-2. Click **OK** when prompted to run automation

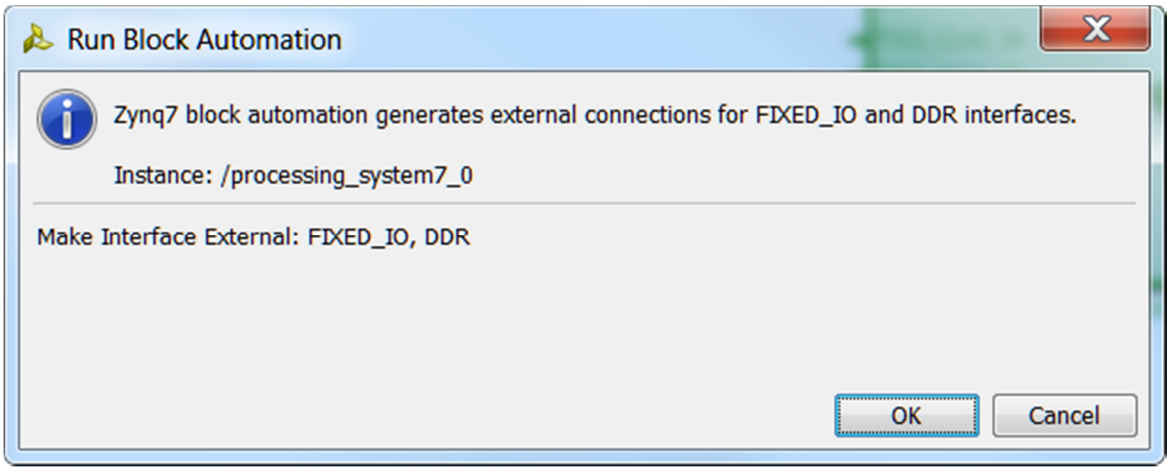


Figure 15. Run Block Automation

Once Block Automation has been complete, 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.

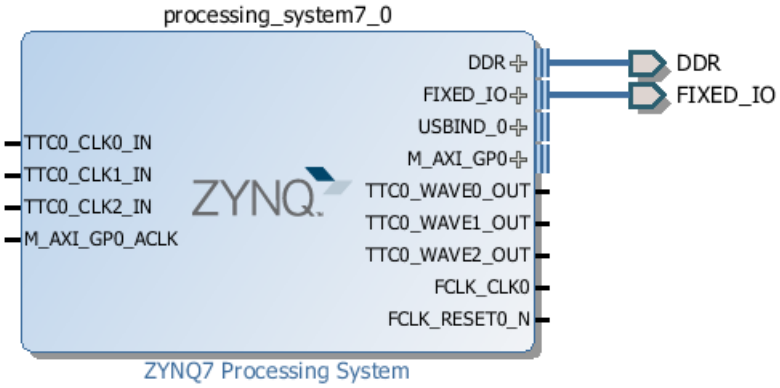


Figure 16.11 Zynq Block with DDR and Fixed IO ports

2-2-3. In the block diagram, double click on the *Zynq* block to open the *Customization* window for the Zynq processing system.

2-2-4. A block diagram of the Zynq should now be open, 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.

2-2-5. Click on one of the peripherals (in green) in the *IOP Peripherals* block, or select the *MIO Configuration* tab on the left to open the configuration form

2-2-6. Expand I/O peripherals if necessary, and deselect all the *I/O peripherals* except *UART 1.*
i.e. Remove: *ENET 0*
USB 0
SD 0

Expand **Memory Interfaces** to deselect *Quad SPI Flash*
Expand **Application Processor Unit** to disable *Timer 0*.

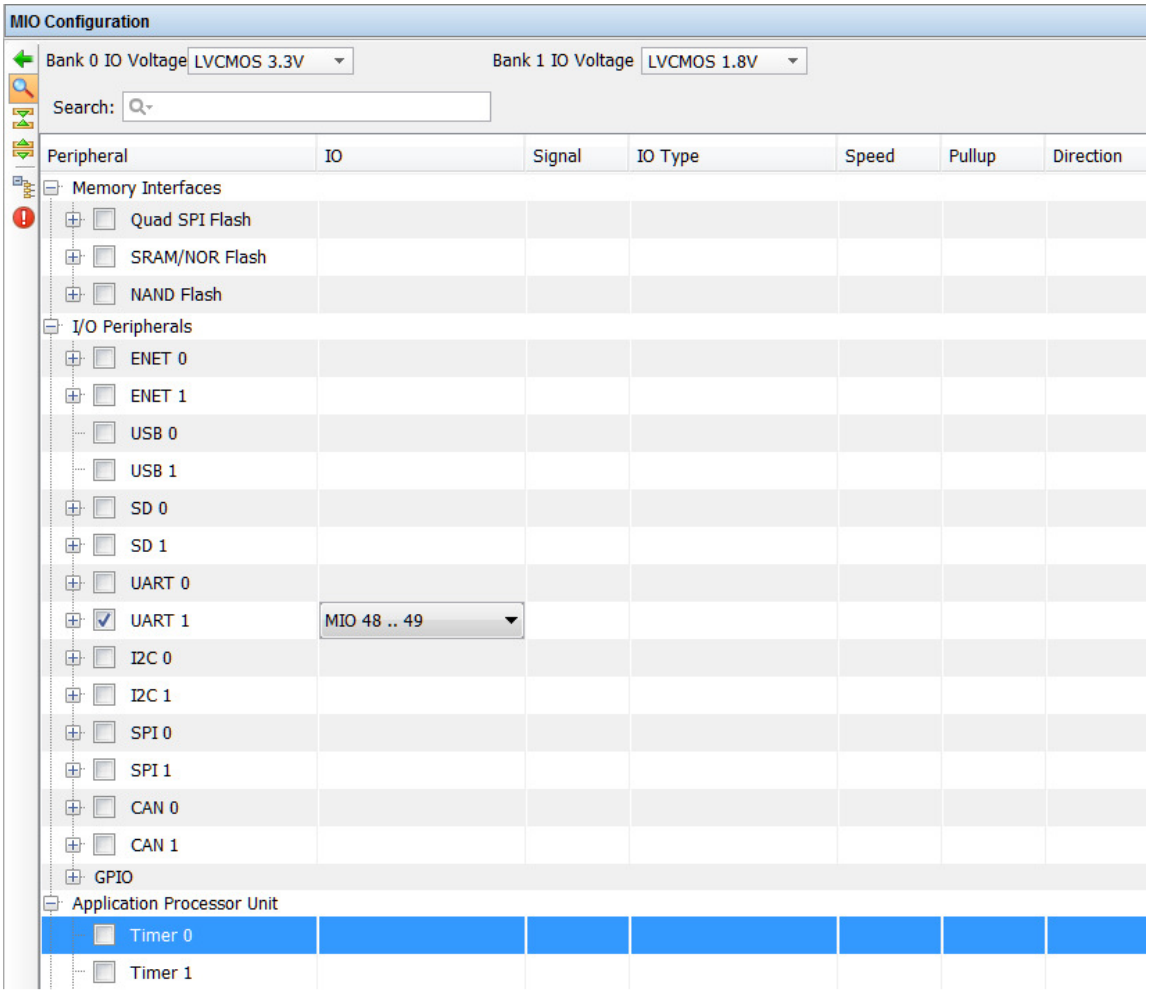


Figure 17. Selecting only UART 1

2-2-7. Select the **PS-PL Configuration** tab on the left. Expand *GP Master AXI interface* and deselect **M AXI GP0** interface. Expand **General > Enable Clock Resets** and deselect the **FCLK_RESET0_N** option.

2-2-8. Select the **Clock Configuration** tab on the left. Expand the **PL Fabric Clocks** and deselect the **FCLK_CLK0** option and click **OK.**

Click on the (Regenerate Layout) button and see the following block diagram.

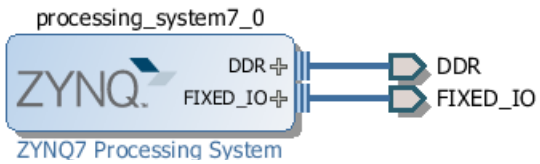


Figure 18. Updated Zynq Block

2-2-9. Click on the (Validate Design) button and make sure that there are no errors.

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 *system.bd*, and select **Generate Output Products …** and click **Generate** to generate the Implementation, Simulation and Synthesis files for the design

3-1-2. Right-click again on *system.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 *system_wrapper.vhd* file will be created and added to the project. Double-click on the file to see the content in the Auxiliary pane.

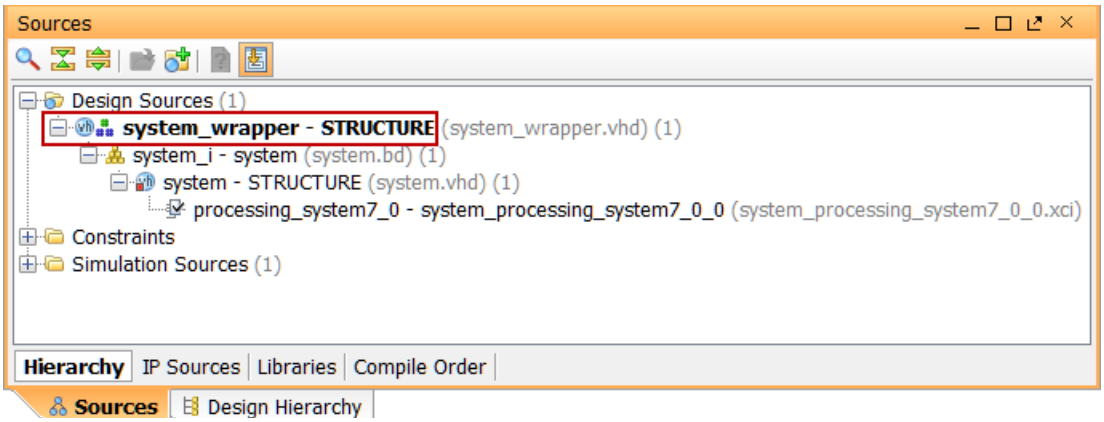


Figure 19. The HDL Wrapper file generated and added to the project

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

You should have the block design open before you export the design to SDK. If it is closed then open the block design by clicking on the Open Block Design under the IP Integrator sub-menu of the Flow Navigator pane.

3-1-4. Select **File > Export > Export hardware for SDK…**

3-1-5. The *Export Hardware for SDK* GUI will be displayed. Select the **Launch SDK** box, and ensure that *Export Hardware* is already selected, and click **OK** to export to, and launch SDK. (**Save** the design if prompted.)

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

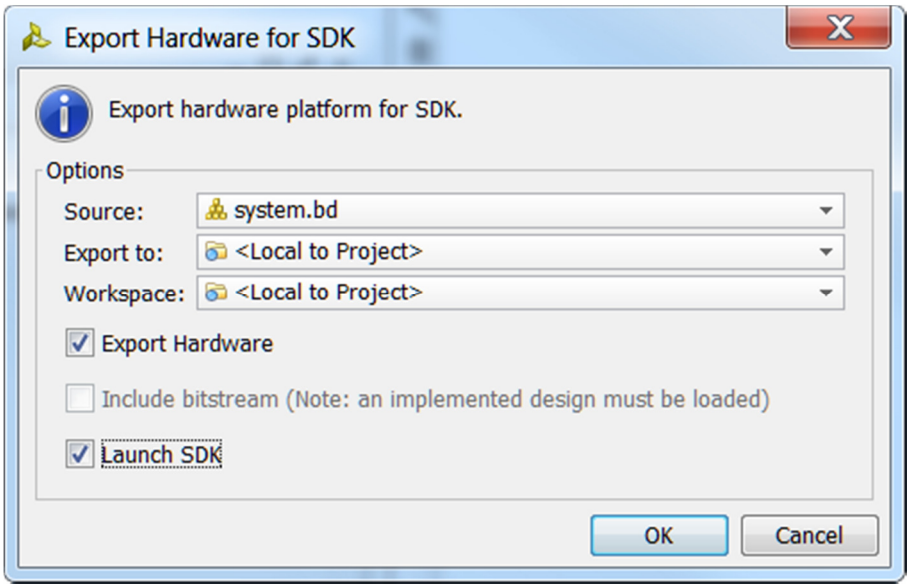


Figure 20. Exporting to SDK

3-1-6. Click **Save** when prompted to save the design.

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 has been created, and the *hw_platform_0* folder should exist in the Project Explorer panel. The .xml file for the Hardware platform should be open in the preview pane. Double click system.xml to open it if it is not.

Basic information about the hardware configuration of the project can be found in the .xml file, along with the Address maps for the PS systems, and driver information.

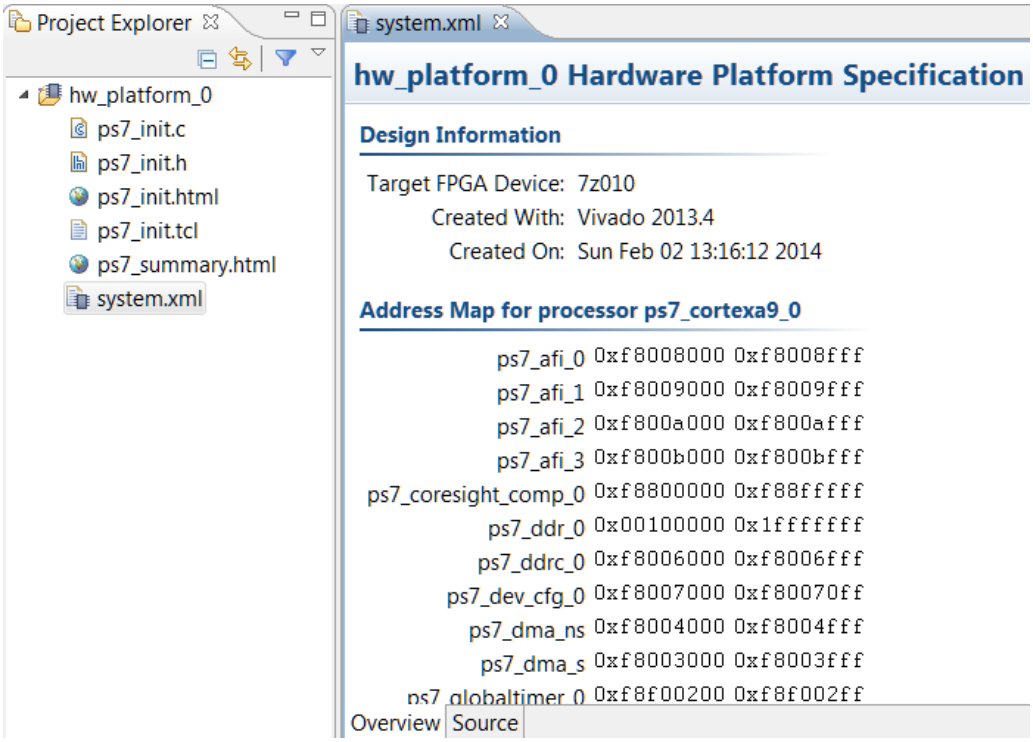


Figure 21. SDK C/C++ development view

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.**

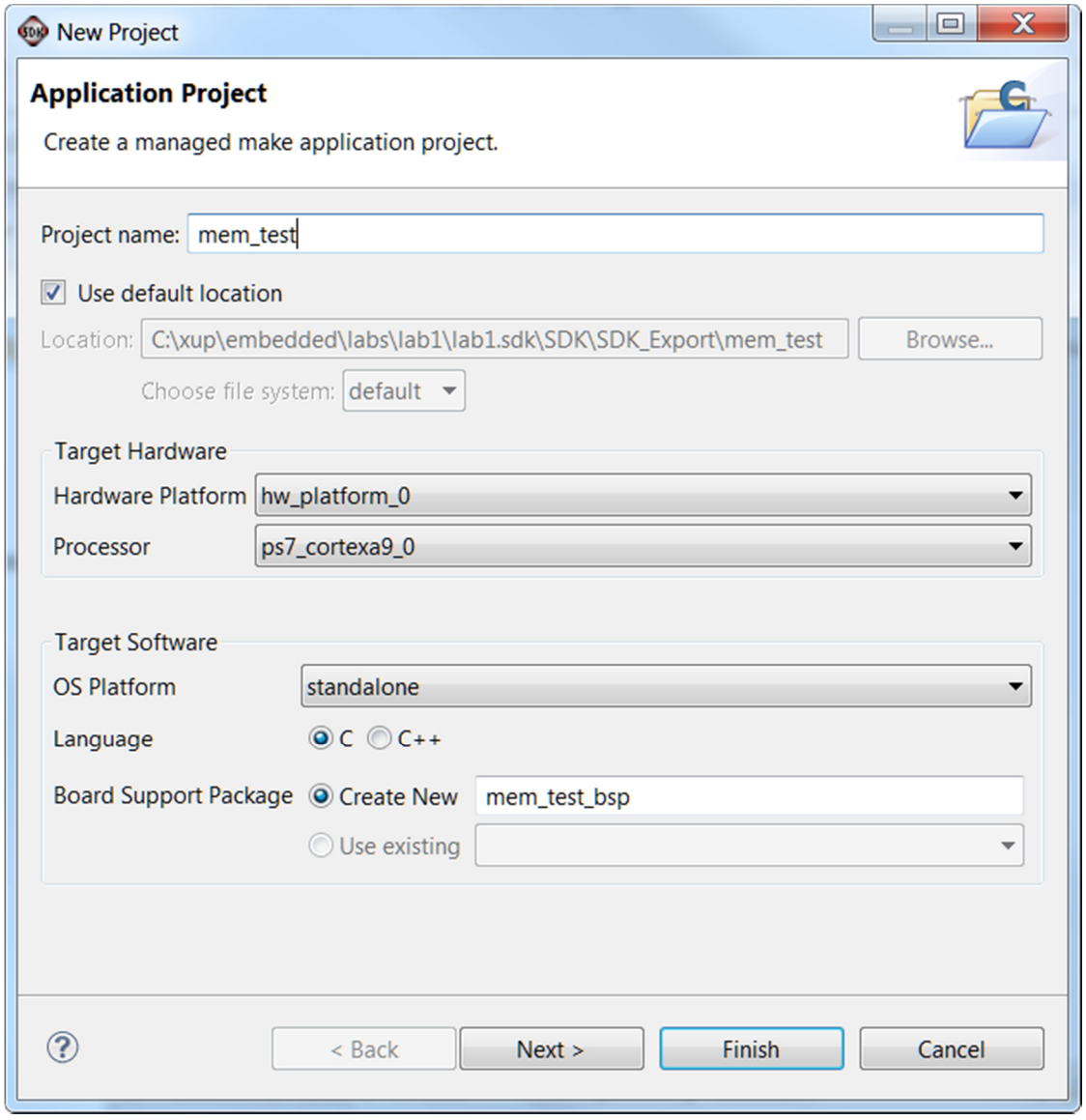


Figure 22. Create new SDK application project

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

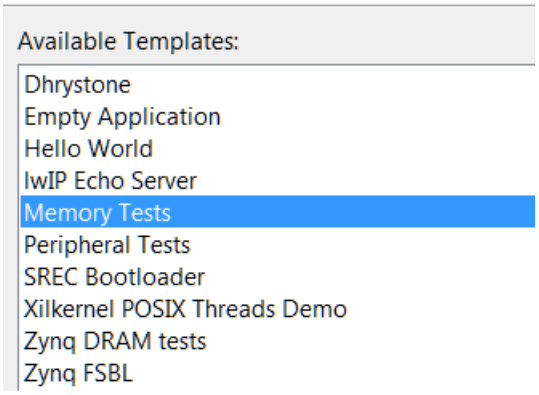


Figure 23. Creating Memory Tests C Project

The **mem_test** project and the board support project **mem_test_bsp** will be created and 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 - *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.

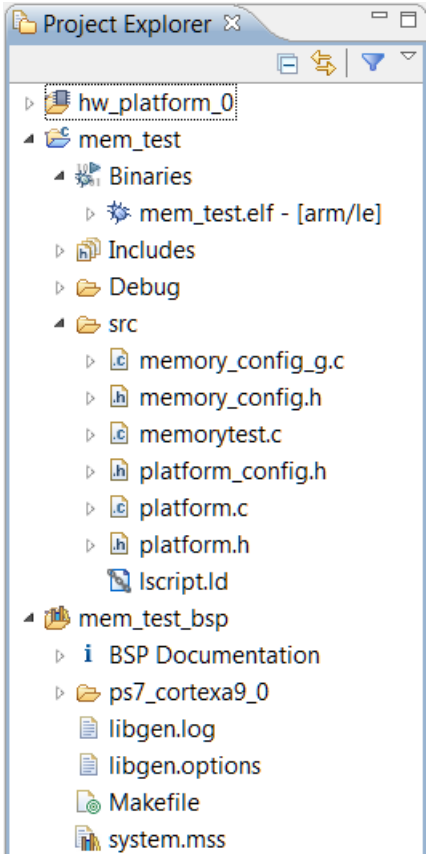


Figure 24. The Project Explore view

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. Make sure that the JP7 is set to select USB power. Connect the board with a micro-usb cable and power it ON. Establish the serial communication using SDK's Terminal tab.

5-1-1. Make sure that the JP7 is set to select USB power.

5-1-2. Make sure that a micro-USB cable is connected to the JTAG PROG connector (next to the power supply connector). Turn ON the power.

5-1-3. Select the Terminal tab. If it is not visible then select **Window > Show view > Terminal**.

5-1-4. Click on and if required, select appropriate COM port (depends on your computer), and configure it with the parameters as shown.

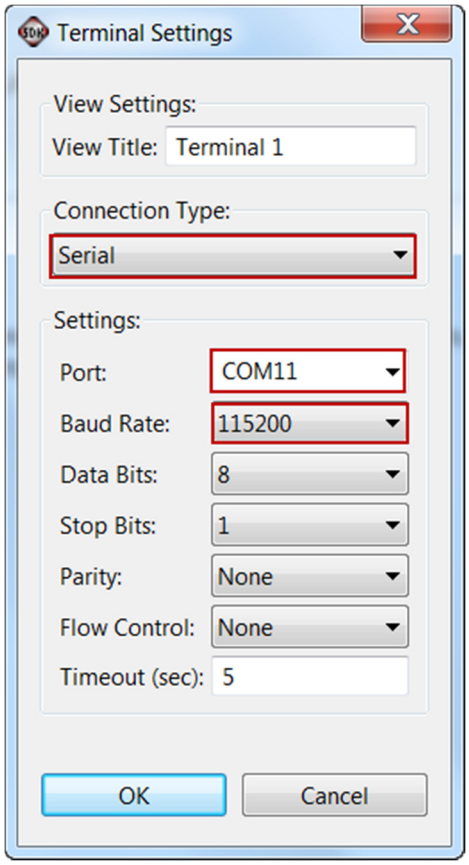


Figure 25. SDK Terminal Settings

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, execute ps7_init, and execute mem_test.elf.

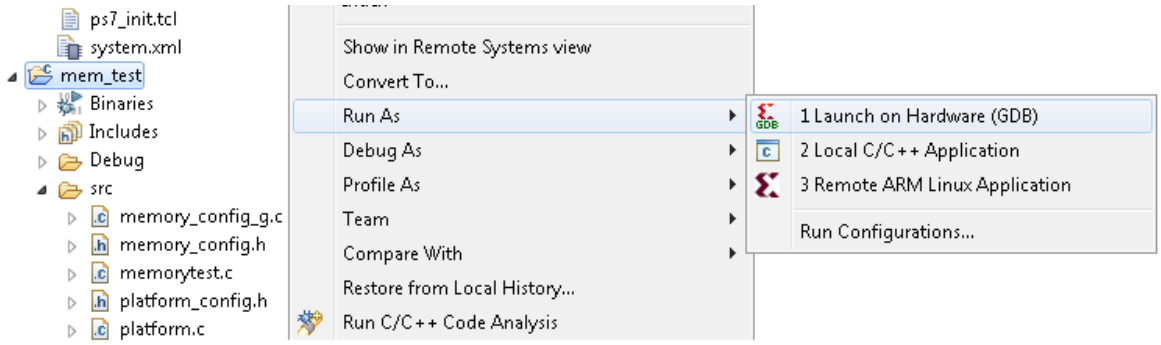


Figure 26. Launch Application

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 27. 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.