



EDK 7.1 PowerPC Tutorial in Virtex-4

Objectives

This tutorial will demonstrate process of creating and testing a PowerPC system design using the Embedded Development Kit (EDK). The tutorial contains these sections:

- System Requirements
- PowerPC System Description
- Tutorial Steps

The following steps are described in this tutorial:

- Starting XPS
- Using the Base System Builder Wizard
- Create – Import IP Peripheral
- Design Modification using Platform Studio
- Implementing the Design
- Defining the Software Design
- Downloading the Design
- Debugging the Design
- Performing Behavioral Simulation of the Embedded System

System Requirements

You must have the following software installed on your PC to complete this tutorial:

- Windows 2000 SP2/Windows XP

Note: This tutorial can be completed on Linux or Solaris, but the screenshots and directories illustrated in this tutorial are based on the Windows Platform.

- EDK 7.1i SP1 or later
- ISE 7.1i SP1 or later

In order to download the completed processor system, you must have the following hardware:

- Xilinx ML403 Evaluation Platform (XC4FX12 FF668)
- Xilinx Parallel -4 Cable used to program and debug the device
- Serial Cable

Note: It should be noted that other hardware could be used with this tutorial. However, the completed design has only been verified on the board specified above. The following design changes are required:

- ◆ Update pin assignments in the system.ucf file
- ◆ Update board JTAG chain specified in the download.cmd

PowerPC System Description

In general, to design an embedded processor system, you need the following:

- Hardware components
- Memory map
- Software application

Tutorial Design Hardware

The PowerPC (PPC) tutorial design includes the following hardware components:

- PowerPC
- PLB Bus
 - ◆ PLB_BRAM_IF_CNTLR
 - ◆ BRAM_BLOCK
 - ◆ PLB_EMC
 - ◆ PLB2OPB_Bridge
- On-chip Peripheral Bus (OPB) BUS
 - ◆ OPB_UARTLITE
 - ◆ 2 - OPB_GPIOs

Tutorial Design Memory Map

The following table shows the memory map for the tutorial design as created by Base System Builder.

Device	Address		Size	Comment
	Min	Max		
PLB_BRAM_CNTLR	0xFFFF_8000	0xFFFF_FFFF	32K bytes	PLB Memory
OPB_UARTLITE	0x4060_0000	0x4060_FFFF	64K bytes	Serial Output
OPB_GPIO	0x4002_0000	0x4002_FFFF	64K bytes	LED output
OPB_GPIO	0x4000_0000	0x4000_FFFF	64K bytes	Push Buttons
PLB_EMC	0x0000_0000	0x0000_FFFF	1M bytes	External Memory Controller

Table 1: Tutorial Design Memory Map

Tutorial Steps

SetUp

- ML403 board with a RS-232 terminal connected to the serial port and configured for 57600 baud, with 8 data bits, no parity and no handshakes.

Creating the Project File in XPS

The first step in this tutorial is using the Xilinx Platform Studio (XPS) to create a project file. XPS allows you to control the hardware and software development of the PowerPC system, and includes the following:

- An editor and a project management interface for creating and editing source code
- Software tool flow configuration options

You can use XPS to create the following files:

- Project Navigator project file that allows you to control the hardware implementation flow
- Microprocessor Hardware Specification (MHS) file
- *Note: For more information on the MHS file, refer to the “Microprocessor Hardware Specification (MHS)” chapter in the Embedded System Tools Guide.*
- Microprocessor Software Specification (MSS) file

Note: For more information on the MSS file, refer to the “Microprocessor Software Specification (MSS)” chapter in the Embedded System Tools Guide.

XPS supports the software tool flows associated with these software specifications. Additionally, you can use XPS to customize software libraries, drivers, and interrupt handlers, and to compile your programs.

I. Starting XPS

To open XPS, select the following:

Start → Programs → Xilinx Platform Studio 7.1i → Xilinx Platform Studio

Select Base System Builder Wizard (BSB) to open the Create New Project Using BSB Wizard dialog box shown in Figure 1.

Click **Ok**.

Use the Project File **Browse** button to browse to the folder you want as your project directory.

Click **Open** to create the system.xmp file.

Click **Ok** to start the BSB wizard.

Note: XPS does not support directory or project names which include spaces.

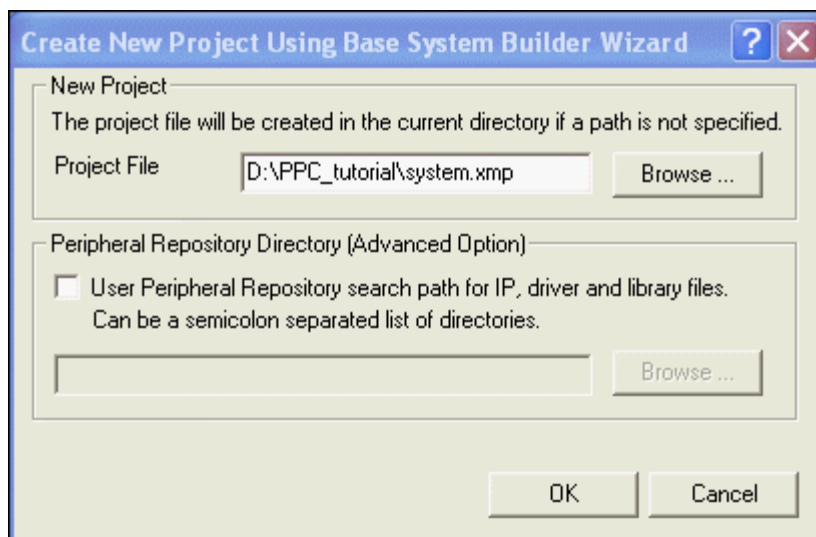


Figure 1: Create New Project Using Base System Builder Wizard

Defining the System Hardware

II. MHS and MPD Files

The next step in the tutorial is defining the embedded system hardware with the Microprocessor Hardware Specification (MHS) and Microprocessor Peripheral Description (MPD) files.

MHS File

The Microprocessor Hardware Specification (MHS) file describes the following:

- Embedded processor: either the soft core MicroBlaze processor or the hard core PowerPC (only available in Virtex-II Pro devices)
- Peripherals and associated address spaces
- Buses
- Overall connectivity of the system

The MHS file is a readable text file that is an input to the Platform Generator (the hardware system building tool). Conceptually, the MHS file is a textual schematic of the embedded system. To instantiate a component in the MHS file, you must include information specific to the component.

MPD File

Each system peripheral has a corresponding MPD file. The MPD file is the symbol of the embedded system peripheral to the MHS schematic of the embedded system. The MPD file contains all of the available ports and hardware parameters for a peripheral. The tutorial MPD file is located in the following directory:

`$XILINX_EDK/hw/XilinxProcessorIPLib/pcores/<peripheral_name>/data`

Note: For more information on the MPD and MHS files, refer to the *“Microprocessor Peripheral Description (MPD)”* and *“Microprocessor Hardware Specification (MHS)”* chapters in the *Embedded System Tools Guide*.

EDK provides two methods for creating the MHS file. Base System Builder Wizard and the Add/Edit Cores Dialog assist you in building the processor system, which is defined in the MHS file. This tutorial illustrates the Base System Builder.

III. Using the Base System Builder Wizard

Use the following steps to create the processor system:

In the Base System Builder – Select “I would like to create a new design”.

In the Base System Builder - Select Board Dialog select the following, as shown in Figure 2:

- Board Vendor: Xilinx
- Board Name: Virtex 4 ML403 Evaluation Platform
- Board Revision: 1

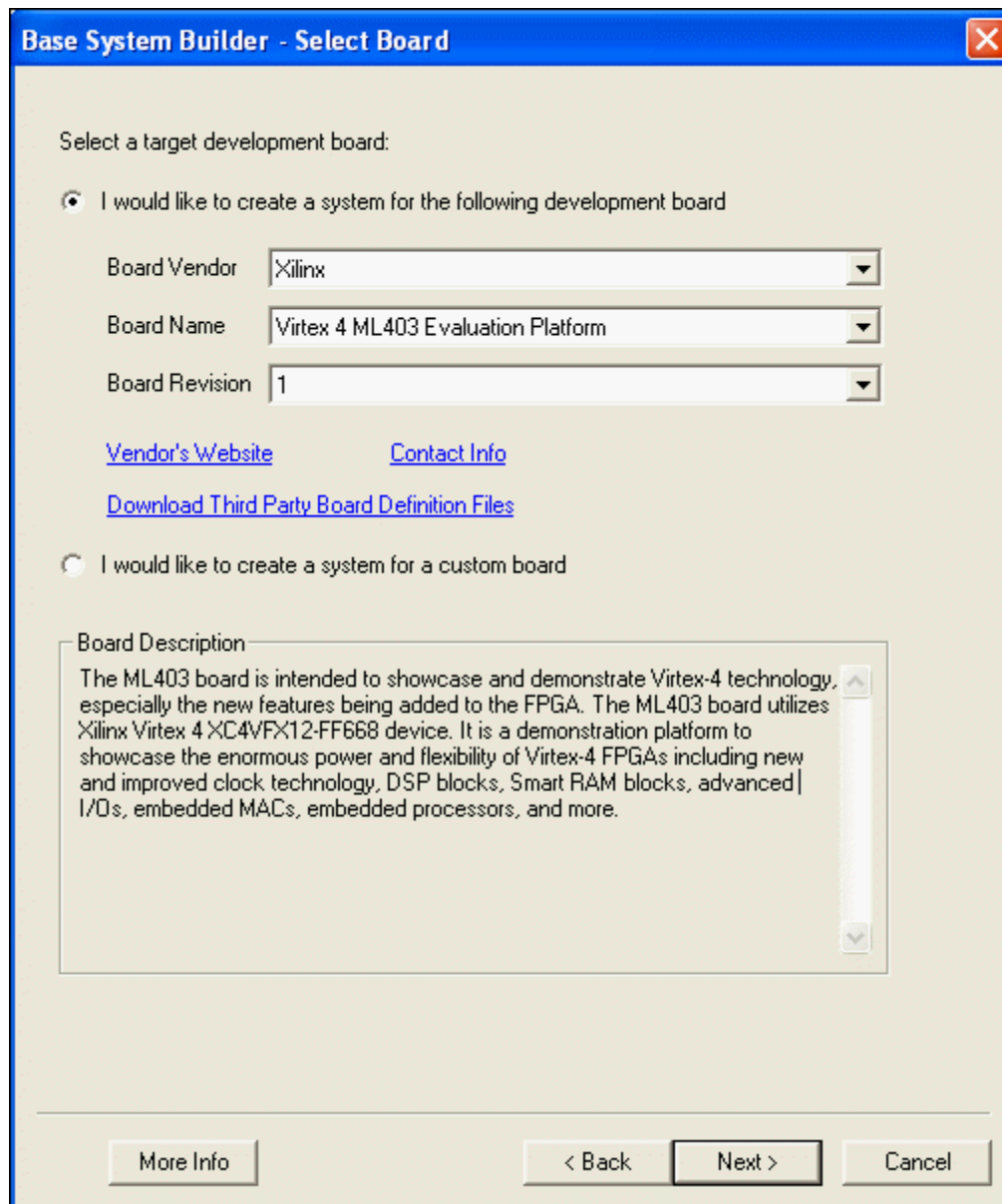


Figure 2: BSB: Select a Board

Click **Next**. Verify that PowerPC is selected.

Click **Next**. You will now specify several processor options as shown in Figure 3:

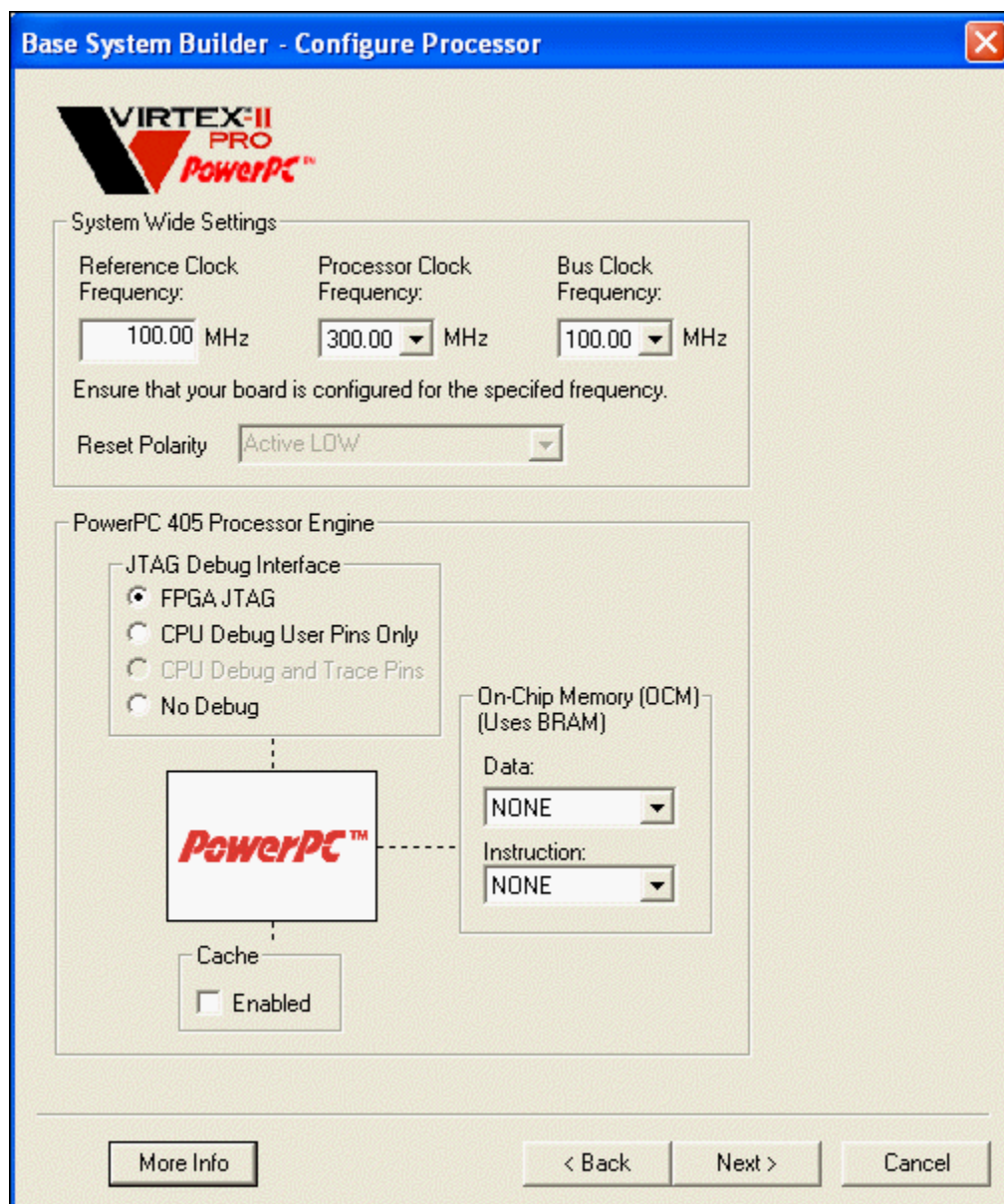


Figure 3: Configure Processor

The following is an explanation of the settings specified in Figure 3:

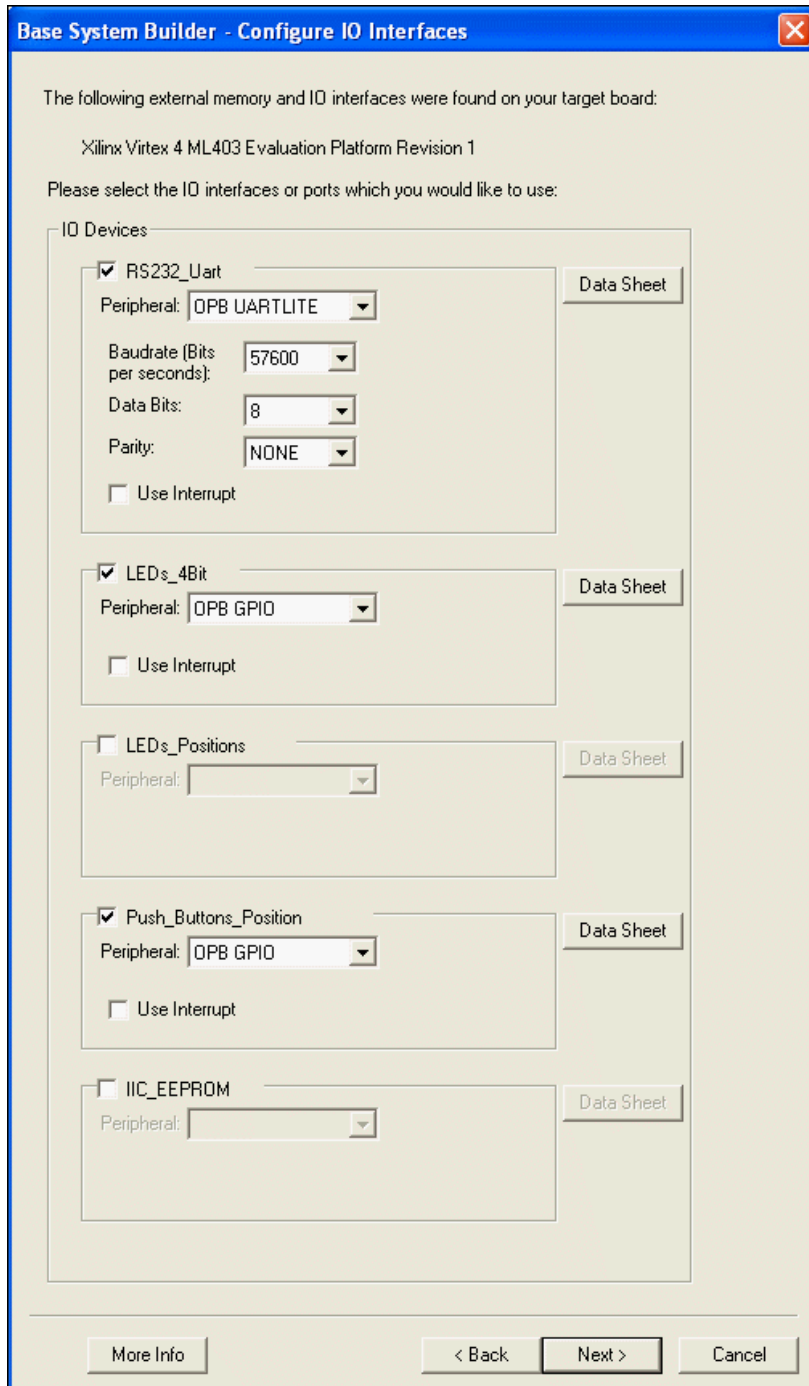
- System Wide Setting:
 - ◆ Processor Clock Frequency: This is the frequency of the clock driving the processor system.
 - ◆ Bus Clock Frequency: This is the frequency of the clock driving the PLB, OPB, and OCM buses.
- Processor Configuration:
 - ◆ JTAG Debug Interface:
 - ◆ FPGA JTAG: The PowerPC JTAG pins will be included in the FPGA JTAG chain.
 - ◆ CPU Debug User Pins Only: This will bring the PowerPC JTAG pins out to user IO.
 - ◆ CPU Debug User and Trace Pins: This option is unavailable because the ML403 board does not have a separate trace header.
 - ◆ No Debug: No debug is turned on.

Note: For more information about the Xilinx Microprocessor Debugger (XMD), refer to the **Xilinx Microprocessor Debugger (XMD) chapter** in the *Embedded System Tools Guide*.

- Users can also specify the size of the On-Chip Memory.
- You can also specify the use of a cache.

Click **Next**. Select the peripheral subset as shown in Figure 4 and Figure 5.

Note: The Baudrate for the OPB UARLITE must be updated to 57600.



The dialog box is titled "Base System Builder - Configure IO Interfaces". It contains the following text and controls:

The following external memory and IO interfaces were found on your target board:

Xilinx Virtex 4 ML403 Evaluation Platform Revision 1

Please select the IO interfaces or ports which you would like to use:

IO Devices

- ☒ RS232_Uart
Peripheral: OPB UARLITE (dropdown)
Baudrate (Bits per seconds): 57600 (dropdown)
Data Bits: 8 (dropdown)
Parity: NONE (dropdown)
☐ Use Interrupt
Data Sheet button
- ☒ LEDs_4Bit
Peripheral: OPB GPIO (dropdown)
☐ Use Interrupt
Data Sheet button
- ☐ LEDs_Positions
Peripheral: (empty dropdown)
Data Sheet button
- ☒ Push_Buttons_Position
Peripheral: OPB GPIO (dropdown)
☐ Use Interrupt
Data Sheet button
- ☐ IIC_EEPROM
Peripheral: (empty dropdown)
Data Sheet button

At the bottom are buttons: More Info, < Back, Next >, and Cancel.

Figure 4: Peripheral Selection

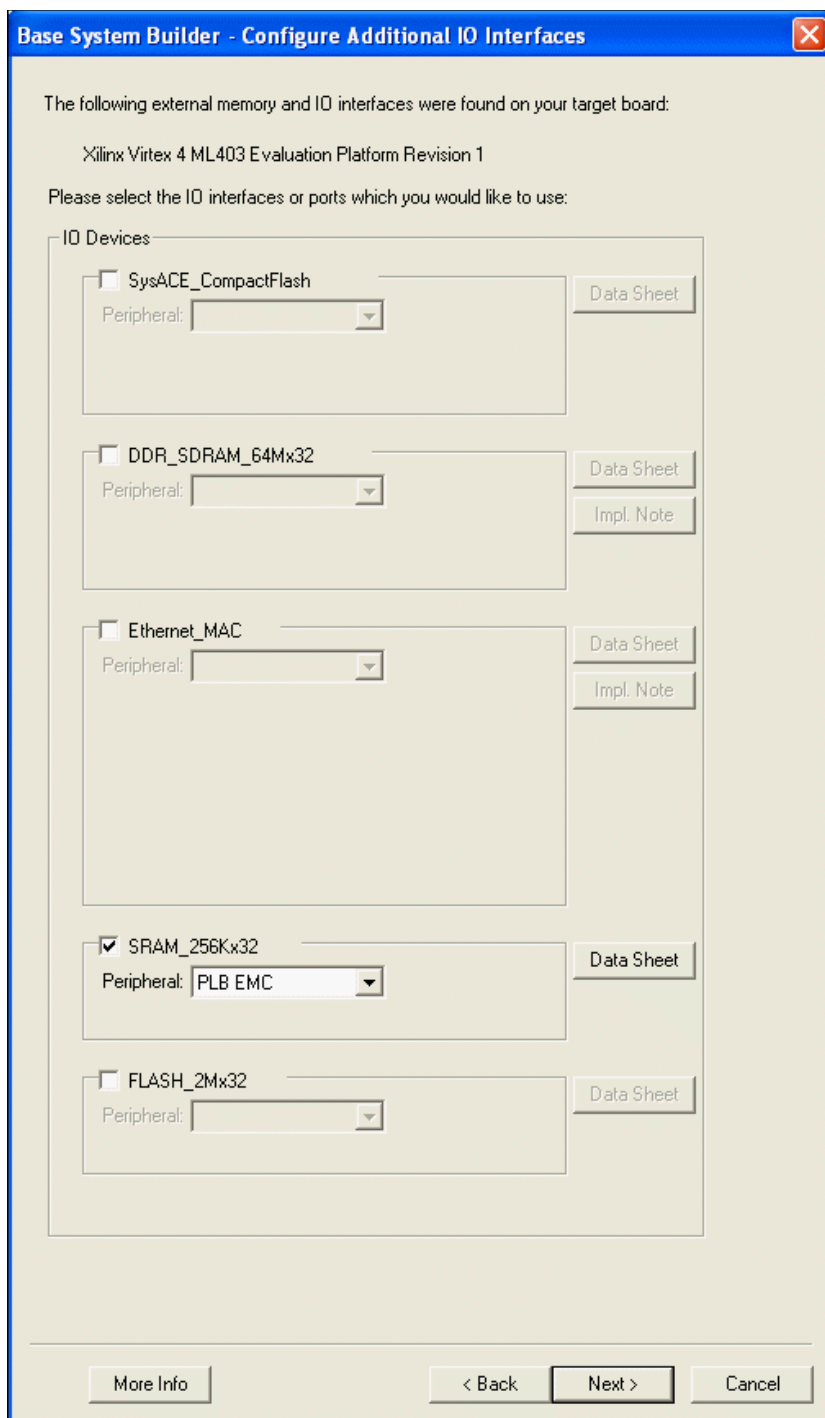


Figure 5: **Peripheral Selection**

Click **Next** through the Configure IO Interface pages.

On the Add Internal Peripherals page, select 32 KB of memory for the PLB BRAM IF CNTLR. This completes the hardware specification and we will now configure the software settings.

Click Next. Using the Software Setup dialog box as shown in Figure 6, specify the following software settings:

- Standard Input (STDIN) → RS232
- Standard Output (STDOUT) → RS232

- o Sample Application Selection → Memory Test
- o Sample Application Selection → Peripheral Self Test

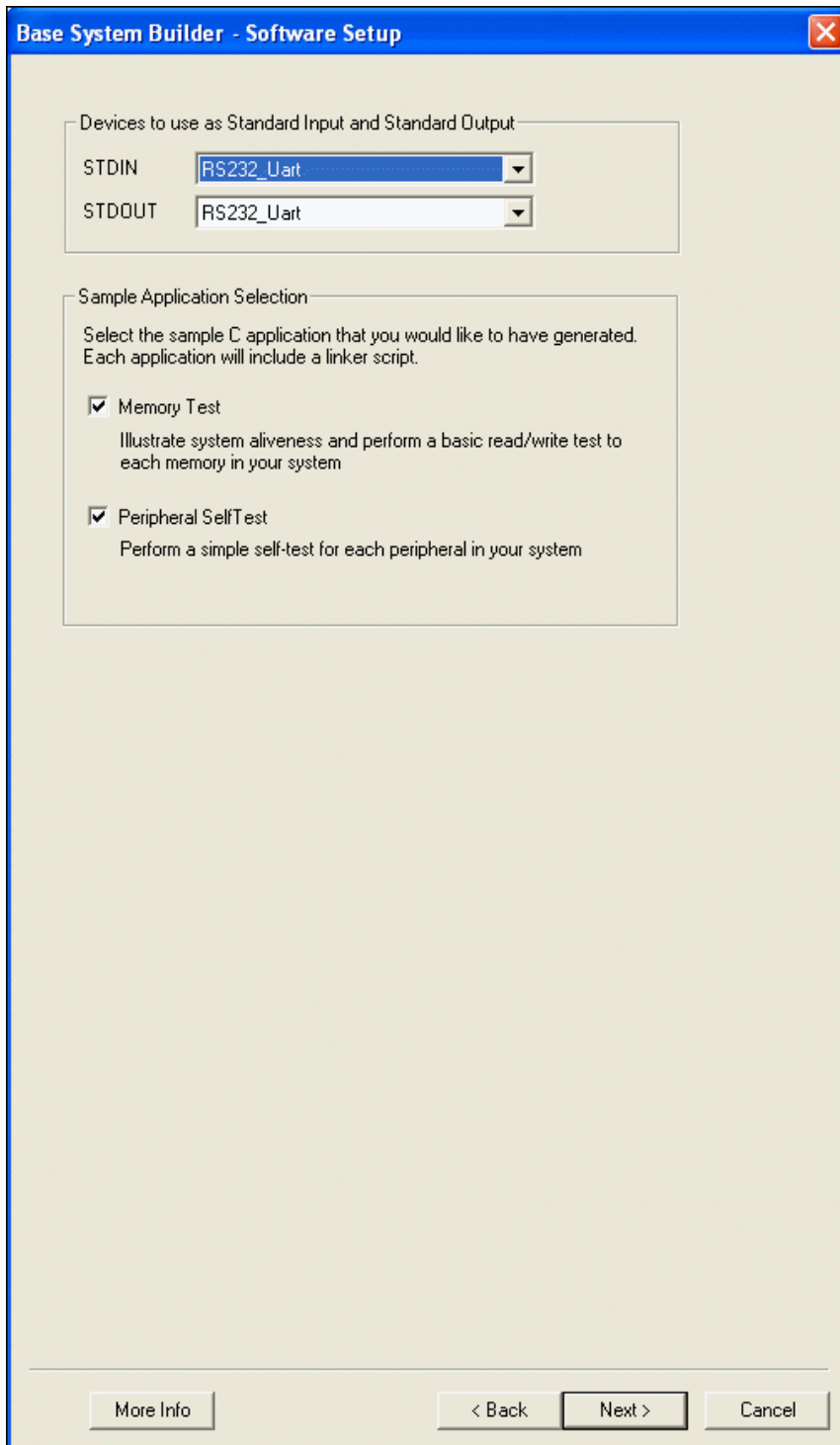


Figure 6: Software Configuration

Click **Next**.

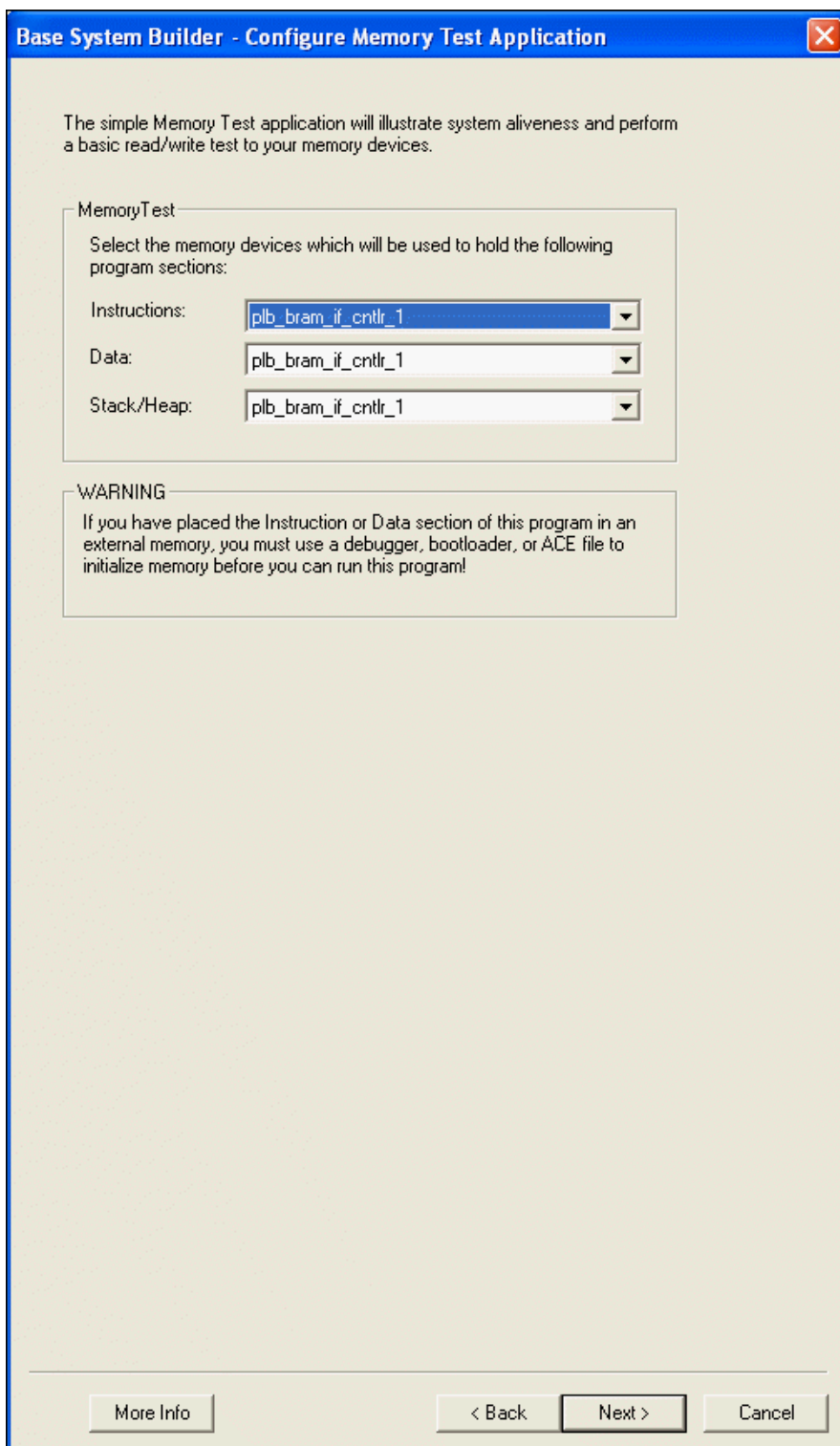


Figure 7: **Configure Memory Test Application**

Using the Configure Memory Test Application dialog box as shown in Figure 7, specify the following software settings:

- Instructions → plb_bram_if_cntlr_1
- Data → plb_bram_if_cntlr_1
- Stack/Heap → plb_bram_if_cntlr_1

Click **Next**. Using the Configure Peripheral Test Application specify the following software settings:

- Instructions → SRAM_256Kx32
- Data → SRAM_256Kx32
- Stack/Heap → SRAM_256Kx32

Click Next. The completed system including the memory map will be displayed as shown in Figure 8. Currently the memory map cannot be changed or updated in the BSB. If you want to change the memory map you can do this in XPS.

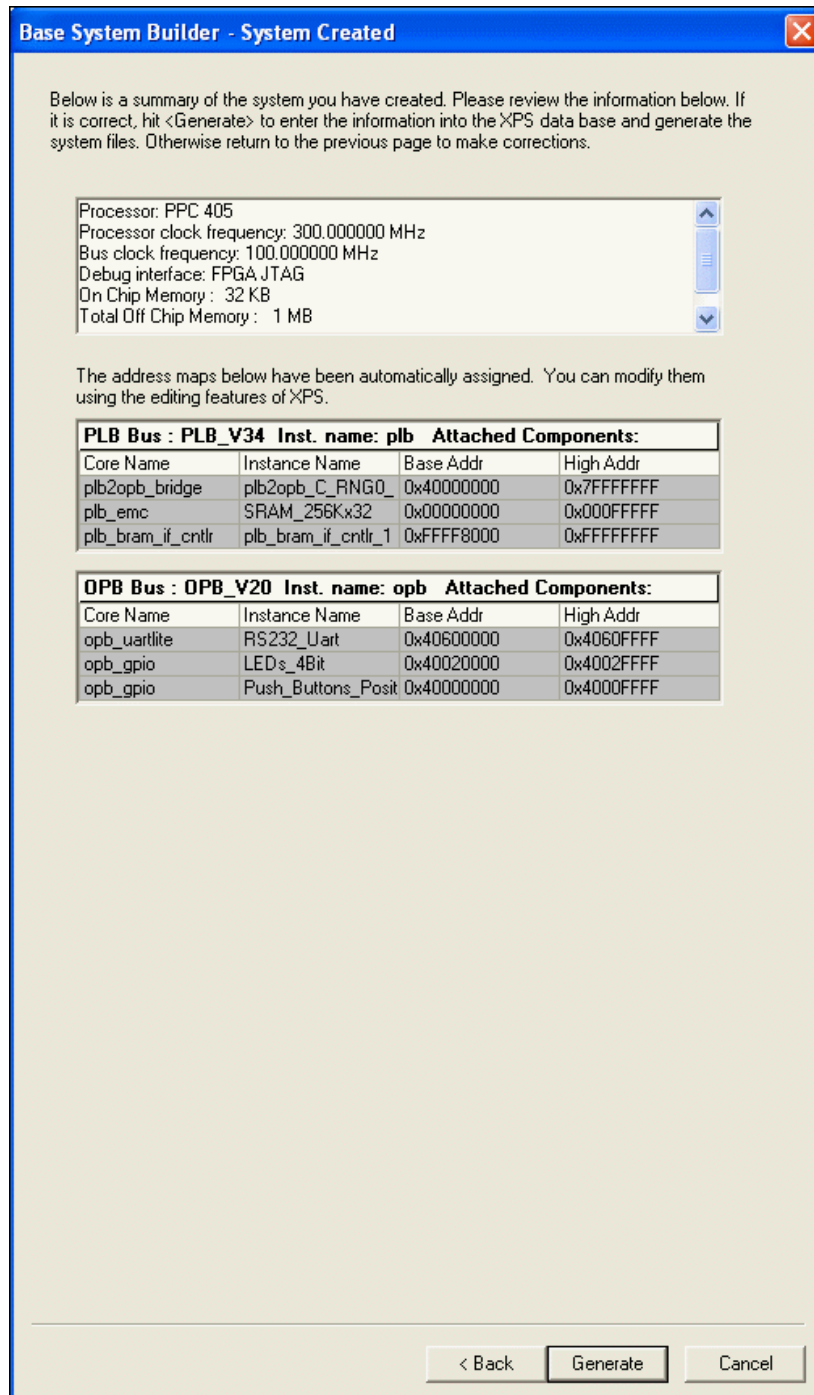


Figure 7: Completed Processor System

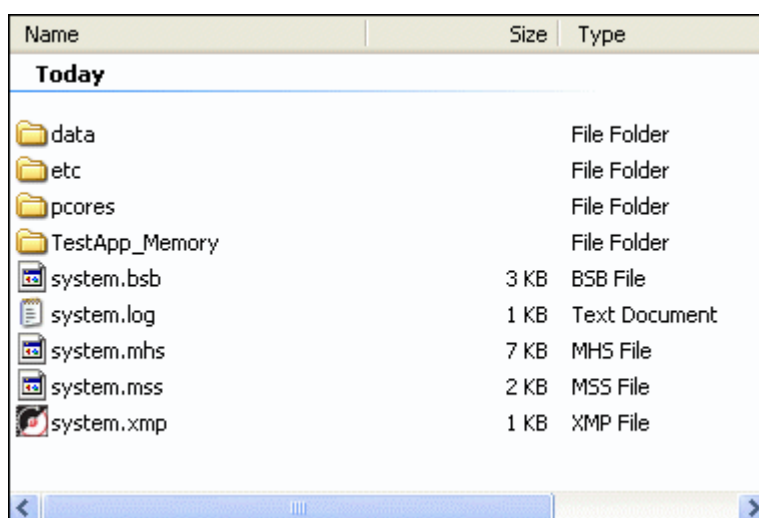
Click **Generate** and then **Finish** to complete the design.

Select Start Using Platform Studio and click OK.

Review

The Base System Builder Wizard has created the hardware and software specification files that define the processor system. When we look at the project directory, shown in Figure 8, we see these as system.mhs and system.mss. There are also some directories created.

- data – contains the UCF (user constraints file) for the target board.
- etc – contains system settings for JTAG configuration on the board that is used when downloading the bit file and the default parameters that are passed to the ISE tools.
- pcores – is empty right now, but is utilized for custom peripherals.
- TestApp_Memory – contains a user application in C code source, for testing the memory in the system.



Name	Size	Type
Today		
data		File Folder
etc		File Folder
pcores		File Folder
TestApp_Memory		File Folder
system.bsb	3 KB	BSB File
system.log	1 KB	Text Document
system.mhs	7 KB	MHS File
system.mss	2 KB	MSS File
system.xmp	1 KB	XMP File

Figure 8: **Project Directory after Base System Builder completes**

Project Options

Base System Builder has also already configured the project settings for us, but take a moment to look at what they are.

select: **Options → Project Options**. As shown in Figure 9, the device information is specified.

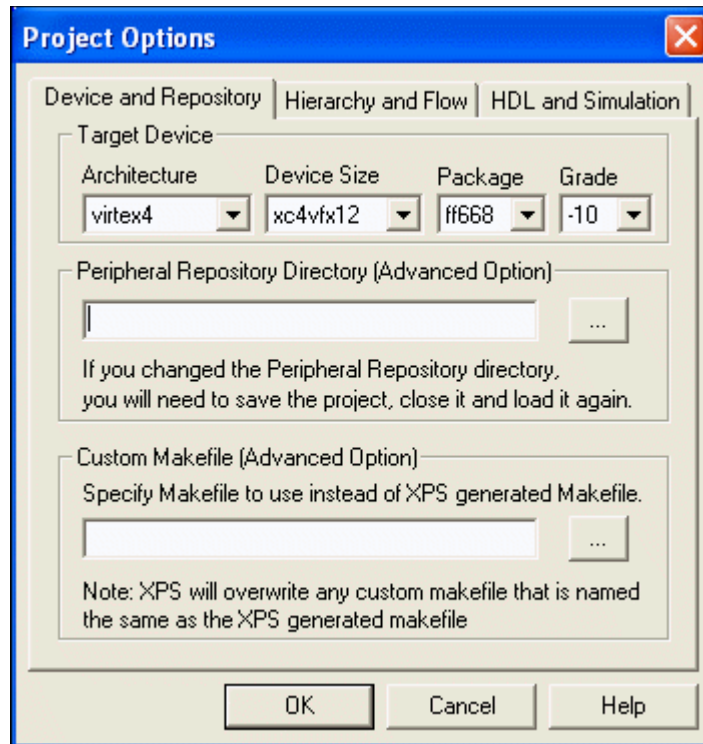


Figure 9: **Project Options - Device and Repository**

Select: **Hierarchy and Flow**. This window is shown in Figure 10. This window provides the opportunity to export the processor system into an ISE project as either the top level system or a sub-module design. We will leave the settings as they are and implement the design using Xflow.

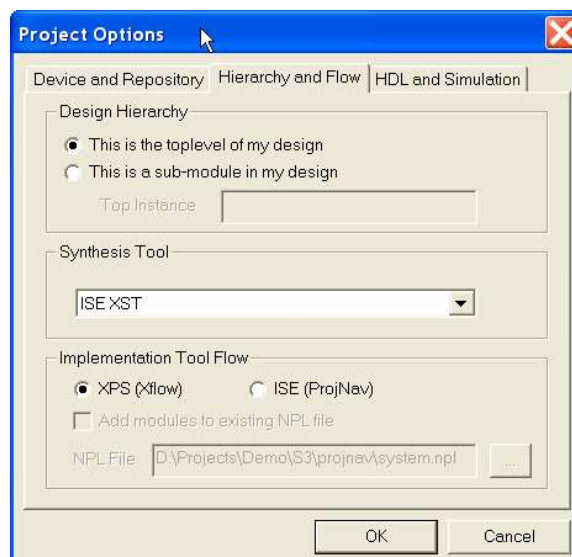


Figure 10: Project Options - Hierarchy and Flow

IV. Create – Import IP Peripheral

One of the key advantages of building an embedded system in an FPGA is the ability to include customer IP and interface that IP to the processor. This section of the tutorial will walk through the steps necessary to include a custom IP core.

In XPS, select **Tools** → **Create/Import Peripheral** to open the Create and Import Peripheral Wizard.

Click **Next**. Select **Create templates for a new peripheral**.

By default the new peripheral will be stored in the project_directory/pcores directory. This enables XPS to find the core for utilization during the embedded system development.

Click **Next**. In the Create Peripheral – Step 1 dialog, enter custom_ip as the name of the peripheral. This is shown in figure 11.

Create Peripheral - Step 1

Name and Version
Indicate the name and version of your peripheral.

Enter the name of your peripheral. This name will be used as the top HDL design entity.

Name:

Version: 1.00.a

Major Revision: Minor Revision: Hardware/Software Compatibility Revision:

Logical library name: custom_ip_v1_00_a

All HDL files (either created by you or generated by this tool) used to implement this peripheral must be compiled into the logical library named above. Any other logical libraries referred to in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK repositories indicated in the XPS project settings.

Figure 11: Create Peripheral – Step 1

In the Create Peripheral – Step 2 dialog, select On-Chip Peripheral Bus (OPB), as this is the bus to which the new peripheral will be connected.

Click **Next**. The Create Peripheral – Step 3 dialog enables the selection of several services. For additional information regarding each of these services, select More Info. Select the User Logic S/W Register Support option.

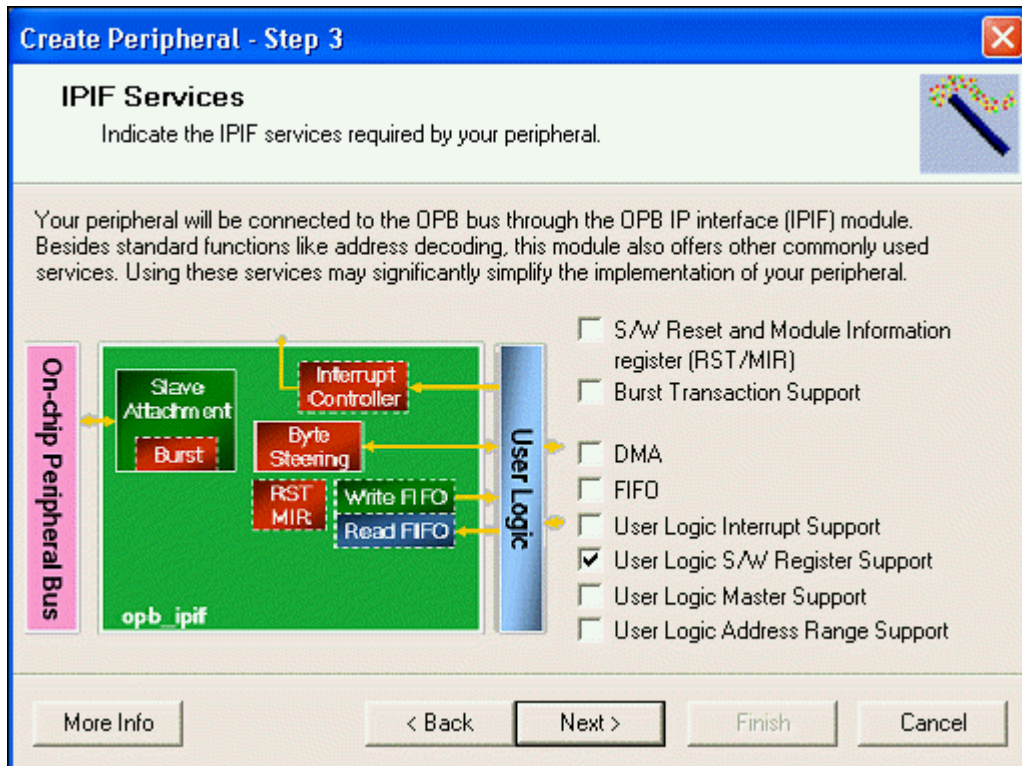


Figure 12: Create Peripheral – Step 3

Click **Next**. In the Create Peripheral –Step 4 dialog change the Number of software accessible registers to 4.

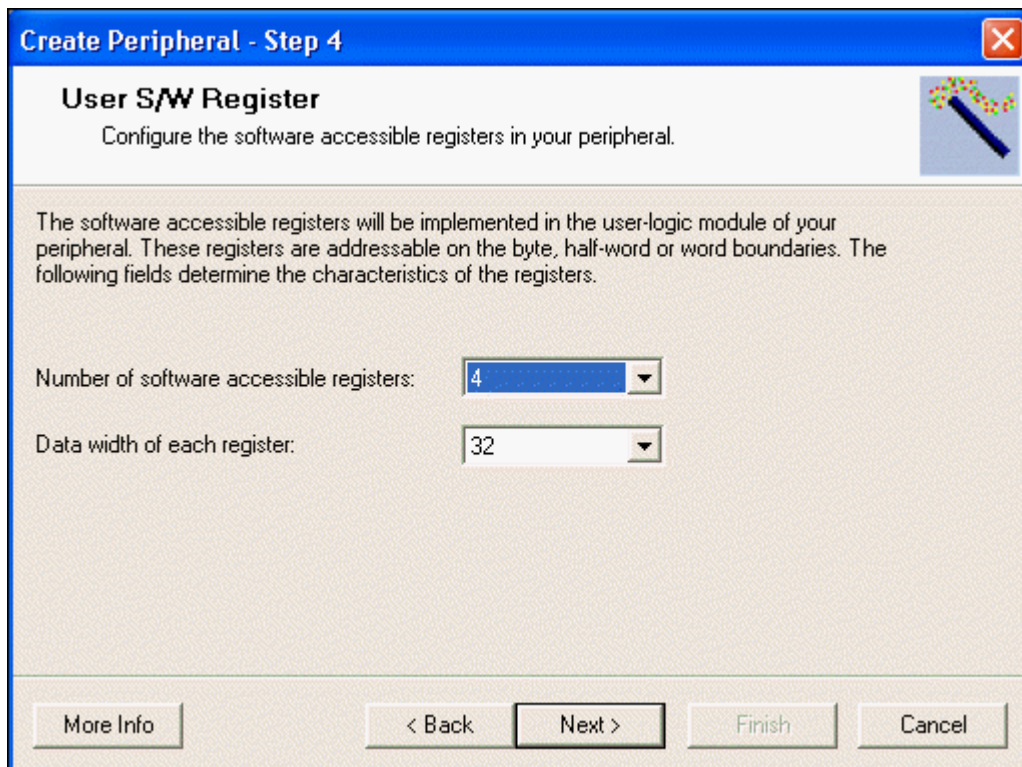


Figure 12: Create Peripheral – Step 4

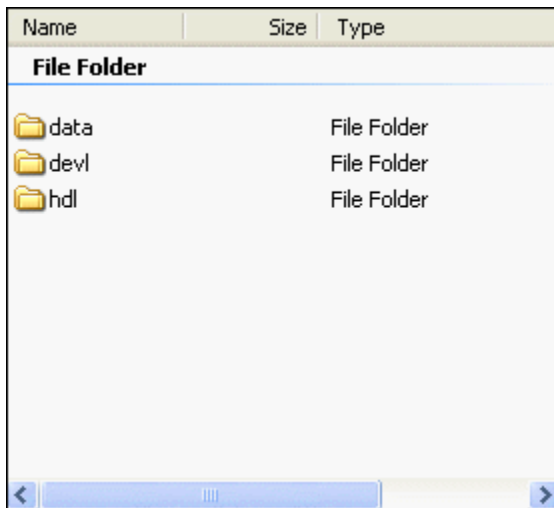
Click **Next**. In the Create Peripheral –Step 5 dialog no additional signal changes are required.

Click **Next**. In the Create Peripheral –Step 6 dialog, a BFM simulation environment can be generated. This tutorial will not cover BFM simulation. Leave the option unchecked.

Click **Next**. In the Create Peripheral –Step 7 dialog, uncheck the Generate ISE and XST project files to help you implement the peripheral using XST flow.

Click **Next** and then Finish.

The Create / Import Peripheral Wizard creates a new directory called custom_ip_v1_00_a in the pcores directory. This new directory contains the following:



The following is a description of the files located in each directory:

- HDL source file(s)

MB_tutorial\pcores\custom_ip_v1_00_a\hdl

vhdl/custom_ip.vhd

This is the template file for your peripheral's top design entity. It configures and instantiates the corresponding IPIF unit in the way you indicated in the wizard GUI and hooks it up to the stub user logic where the actual functionalities should get implemented. You are not expected to modify this template file except certain marked places for adding user specific generics and ports.

vhdl/user_logic.vhd

This is the template file for the stub user logic design entity, either in VHDL or Verilog, where the actual functionalities should get implemented. Some sample code snippet may be provided for demonstration purpose.

- XPS interface file(s)

MB_tutorial\pcores\ custom_ip_v1_00_a\data

custom_ip_v2_1_0.mpd

This Microprocessor Peripheral Description file contains information of the interface of your peripheral, so that other EDK tools can recognize your peripheral.

custom_ip_v2_1_0.pao

This Peripheral Analysis Order file defines the analysis order of all the HDL source files that are used to compile your peripheral.

- Driver source file(s)

MB_tutorial\drivers\ custom_ip_v1_00_a\src:

custom_ip.h

This is the software driver header template file, which contains address offset of software addressable registers in your peripheral, as well as some common masks and simple register access macros or function declaration.

custom_ip.c

This is the software driver source template file, to define all applicable driver functions.

custom_ip_selftest.c

This is the software driver self test example file, which contain self test example code to test various hardware features of your peripheral.

Makefile

This is the software driver makefile to compile drivers.

Now that the template has been created, the user_logic.vhd file must be modified to incorporate the custom IP functionality.

Open user_logic.vhd. Currently the code provides an example of reading and writing to four 32-bit registers. For the purpose of this tutorial, this code will not be modified.

Close user_logic.vhd.

In order for XPS to add the new custom IP core to the design, the pcores directory must be rescanned. This can be accomplished by selecting Project → Rescan User IP Directories.

V. Design Modification using Platform Studio

Once a design has been created with the Base System Builder, it can be modified using the Add/Edit Peripherals dialog. This dialog box allows the following design modifications for peripherals:

Add new peripherals

Change/update address map

Change/update peripheral parameters

Change/update peripheral ports

In XPS, select **Project** → **Add/Edit Cores .. (dialog)**. This will open the Add/Edit Hardware Platform Specifications dialog box, as shown in figure 13, used to modify an existing design.

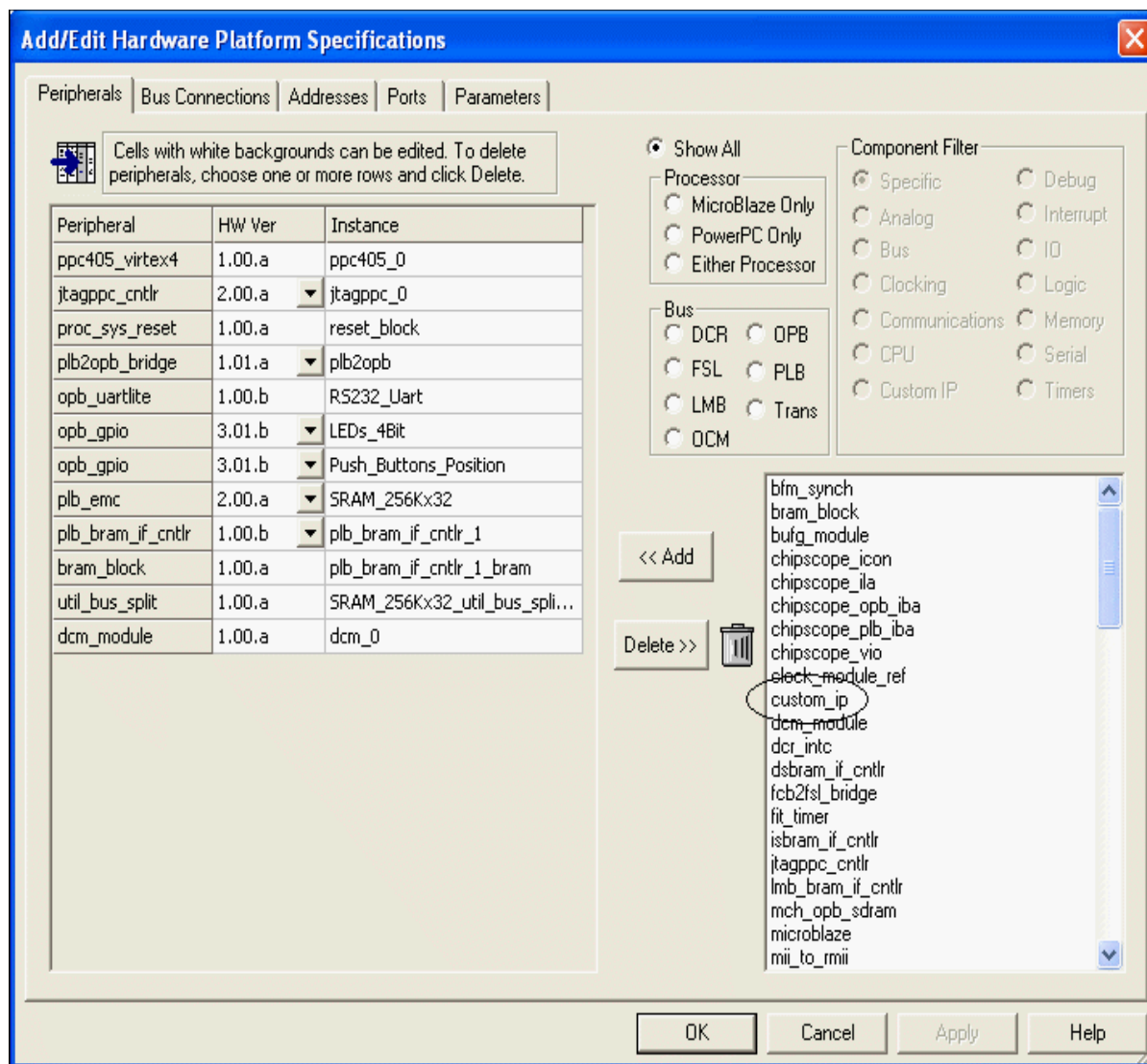


Figure 13: Add/Edit Hardware Platform Specifications - Peripherals

Select the custom_ip peripheral from the complete list of peripherals available to this project, as shown in figure 13.

Click **<< Add** to add the custom_ip peripheral to the project. The instance name for custom_ip peripheral can be changed by clicking in the Instance name field.

Select the Bus Connections tab. Click in the box next to custom_ip_0 sopb, as shown in figure 14, to connect the custom_ip peripheral to the OPB Bus. A connection is made once the "s" is displayed in the box. "s" indicates a slave device, "m" indicates a master device, "bm" indicates a bus monitor. ChipScope Pro cores are an example of a bus monitor.

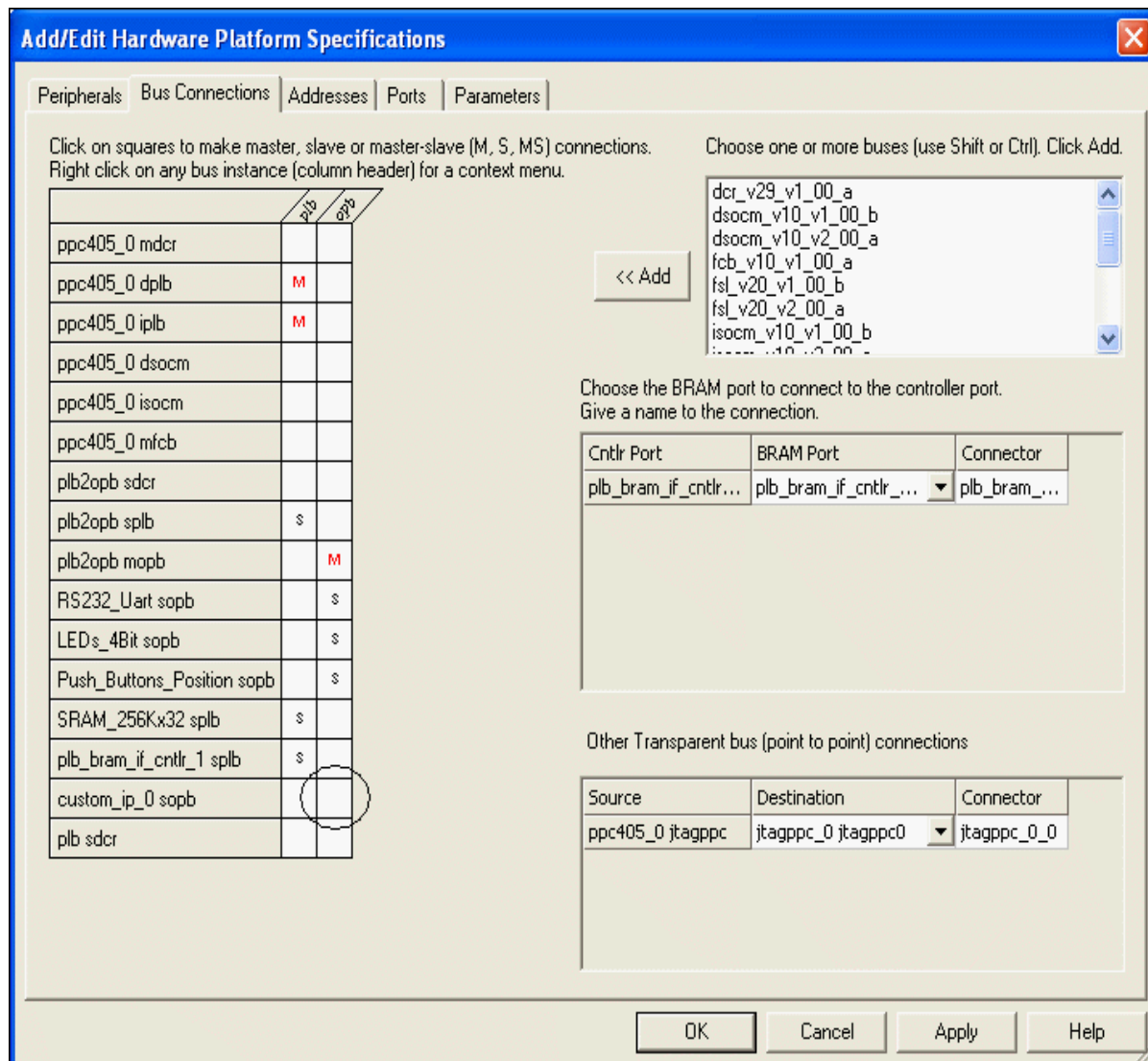


Figure 14: Add/Edit Hardware Platform Specifications – Bus Connections

Select the Addresses tab to define an address for the newly added custom_ip peripheral. The address can be assigned by entering the Base Address or the tool can assign an address. For the purpose of this tutorial, the tool will be used to assign an address.

Click **Generate Addresses**. Click Yes in the Information dialog. Once it successfully generated the address map click Ok.

Select the Ports tab. This tab is used to define connections between peripherals and the external ports. Using the Ports Filter, select custom_IP_0 to display only the ports associated with the custom_ip peripheral. You will notice that there is only one port for this peripheral, OPB_Clk.

Select OPB_Clk and click << Add to add this port to the Internal Port Connections list.

Using the pulldown option for the newly added port select sys_clk_s to connect the OPB_Clk to sys_clk_s.

Click OK to close the Add/Edit Hardware Platform Specifications dialog.

The design is now ready to be implemented.

VI. Implementing the Design

Now that the hardware has been completely specified in the MHS file, you can run the Platform Generator. Platform Generator elaborates the MHS file into a hardware system consisting of NGC files that represent the processor system. Then the Xilinx ISE tools will be called to implement the design for the target board. To generate a netlist and create the bit file, follow these steps:

Start ISE by selecting Start → Programs → Xilinx ISE 7.1i → Project Navigator.

In ISE, select **File → New Project** to create a new Project Navigator project.

In the New Project dialog box shown in figure 15, browse to the XPS project directory and then enter the Project Name, project_navigator.

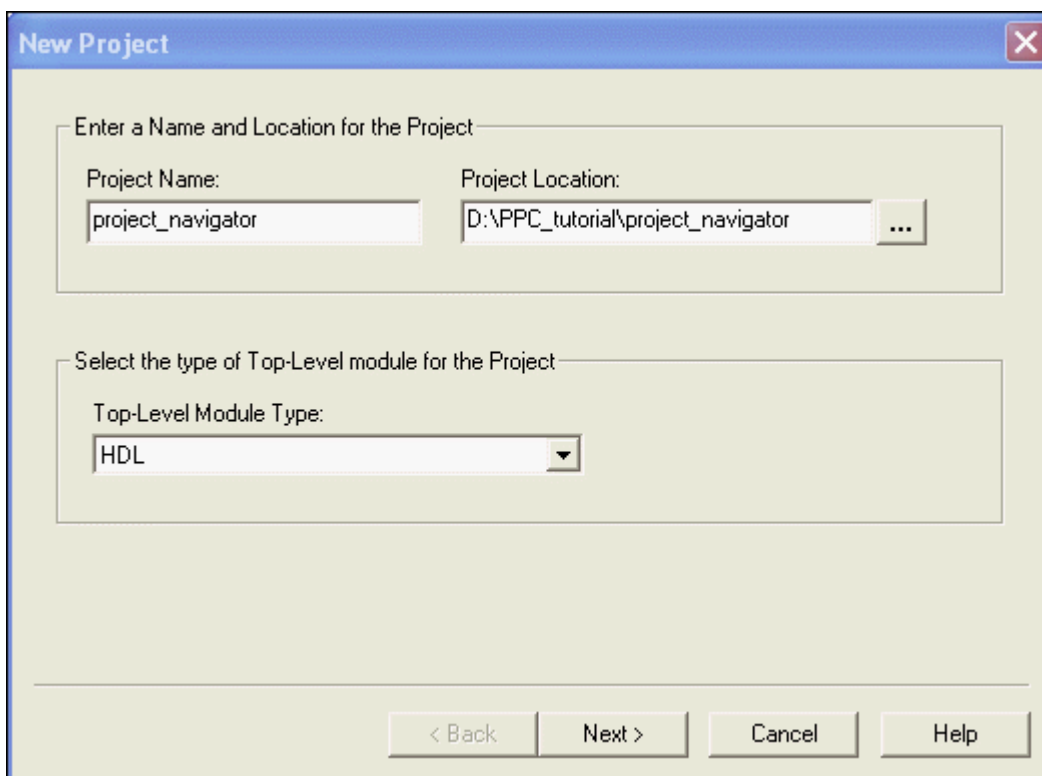


Figure 15: ISE New Project

Click Next. Configure the Device and Design flow as shown in figure 16. It should be noted that these settings are consistent with the XPS project.

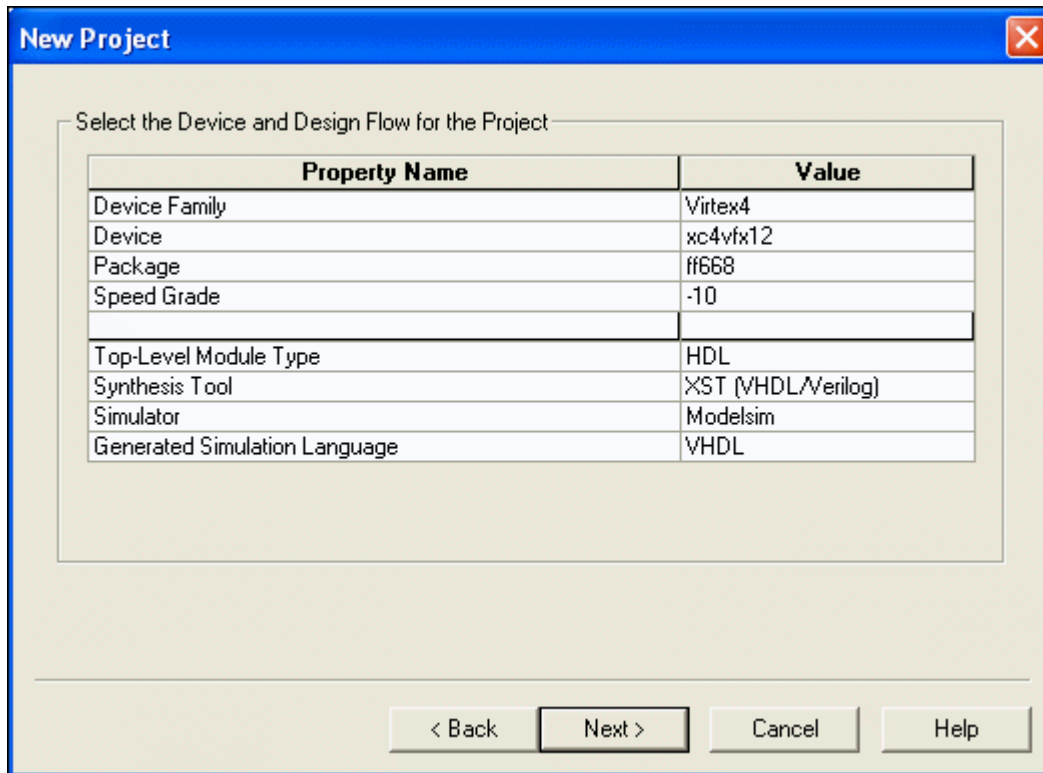


Figure 16: New Project – Device and Design Flow

Click Next. The tutorial will not include a new source file at this time.

Click Next. The tutorial will not include an additional source files at this time.

Click Finish.

In ISE, select Project → Add Source. Select the system.xmp file located in the XPS project directory. Click Open. This will add the system.xmp file as a source file in the ISE project.

Select the system.xmp source file and double click on the View HDL Instantiation Template.

A warning message will be displayed because the netlists for the EDK project have not been created. By clicking OK the netlists will be generated.

Once the process has completed the editor window will contain the instantiation template called system_stub.vhd. Select File → Save As and save system_stub.vhd to the project_navigator directory.

In ISE, select Project → Add Source. Select the system_stub.vhd in the project_navigator directory. The Source Type is a VHDL Design File.

By adding system_stub.vhd to the Project Navigator project the hierarchy is updated as shown in Figure 17.

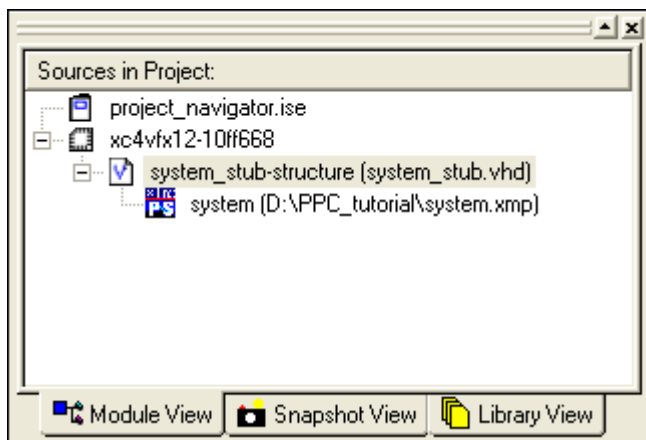


Figure 17: Project Navigator Project Hierarchy

In ISE, select Project → Add Source. Select the system.ucf file in the <xps_project>\data directory. Associate the system.ucf with system_stub.vhd

Select system_stub.vhd and double click on Generate Programming File to implement the design and generate a bit file.

ISE will call XPS to generate the EDK to create the following directories:

- hdl – contains the vhd files that define the processor system
- implementation – contains the NGC files
- synthesis – contains the projects and information from synthesizing the files in the hdl directory to create those in the implementation directory

VII. Defining the Software Design

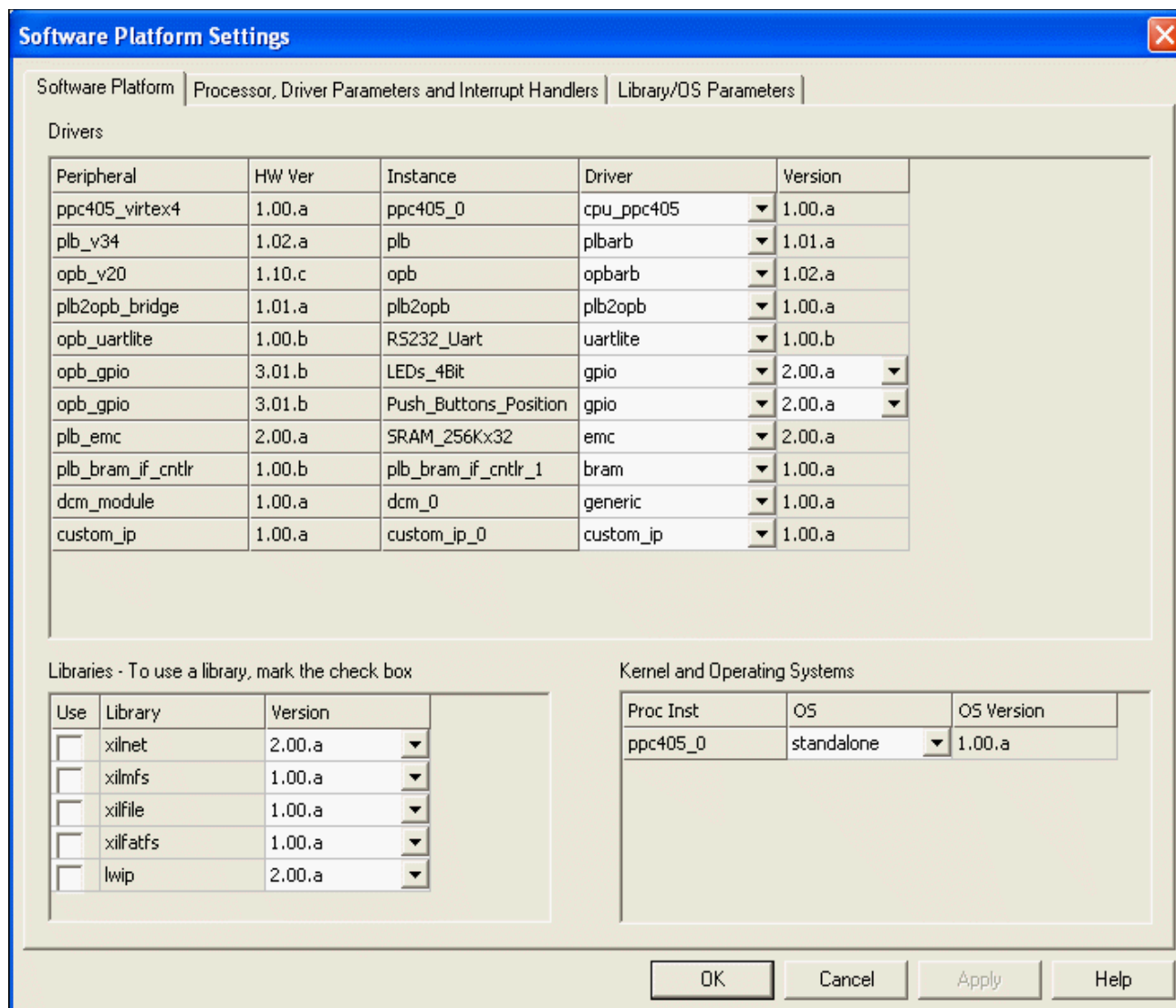
Now that the hardware design is completed, the next step is defining the software design. There are two major parts to software design, configuring the Board Support Package (BSP) and writing the software applications. The configuration of the BSP includes the selection of device drivers and libraries.

Configuration of the BSP

Configuration of the BSP is done using the Software Platform Settings dialog. In XPS, select **Project → Software Platform Settings**. This will open the Software Platform Settings dialog box as shown in Figure 18. The Software Platform Settings dialog box contains three tabs. Each of these tabs is used to control all aspects of the BSP creation. The Software Platform tab allows the user to select the following:

- Drivers: The driver and driver version for each peripheral can be selected. It should be noted that the driver version is not related to the peripheral version.
- Libraries: Select the Xilinx libraries used in the software applications.
- Kernel and Operating Systems: Select the Kernel or Operating System to be used. The following Operating Systems are supported:
 - ◆ Standalone
 - ◆ Xilinx MicroKernel

No changes are required on this tab.

Figure 18: **Software Platform Settings Dialog**

Select the Processor and Driver Parameters tab as shown below in Figure 19. This tab allows the user to configure several Processor and Driver Parameters. No changes are required on this tab.

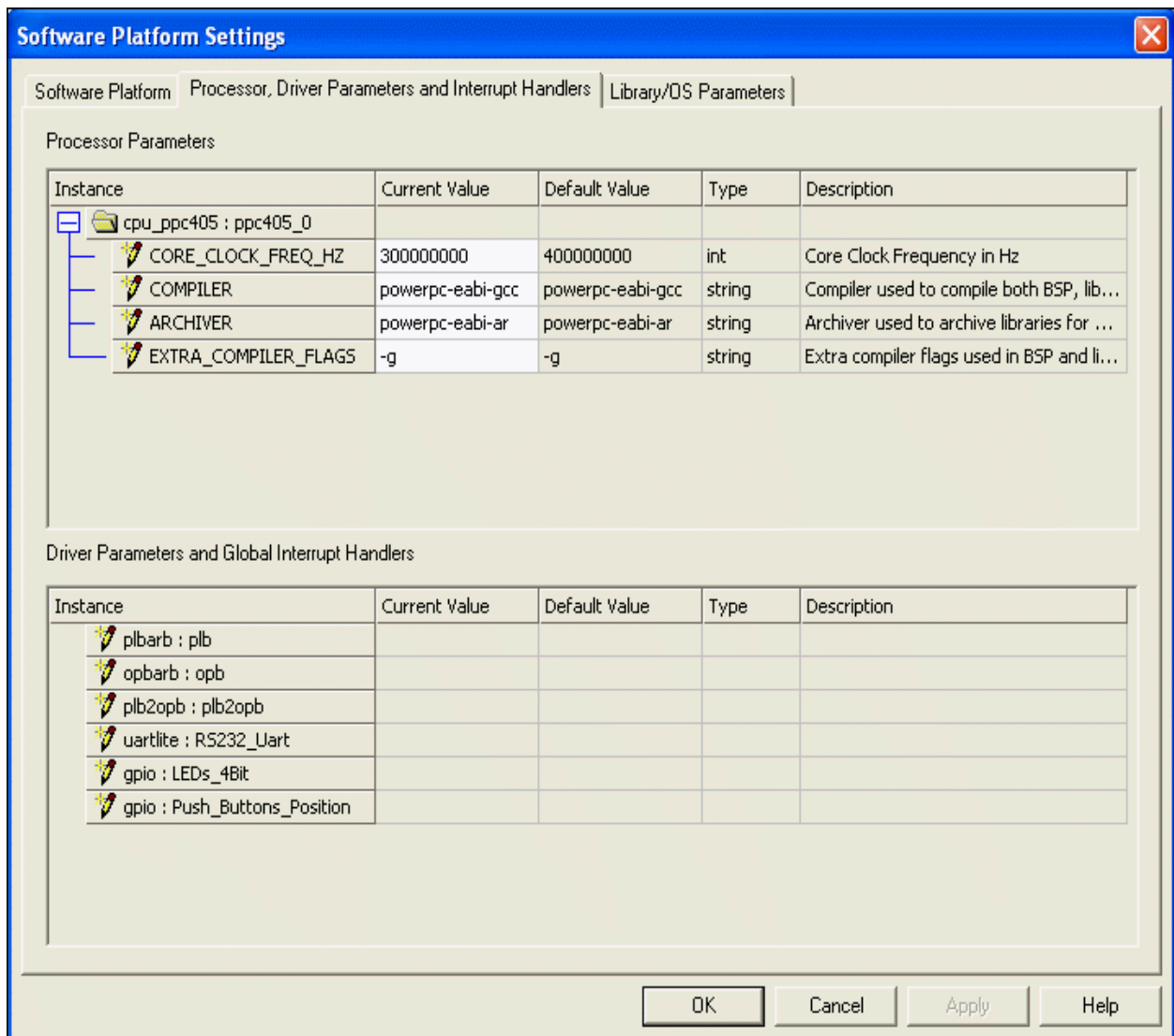


Figure 19: **Processor, Driver Parameters and Interrupt Handler Settings**

Select the Library/OS Parameters tab and configure the Library/OS Parameters tab as shown in Figure 20. This allows the RS232 peripheral to be used for standard IO functions like print, printf, scanf, etc. If the RS232 peripheral is already selected, no change is required.

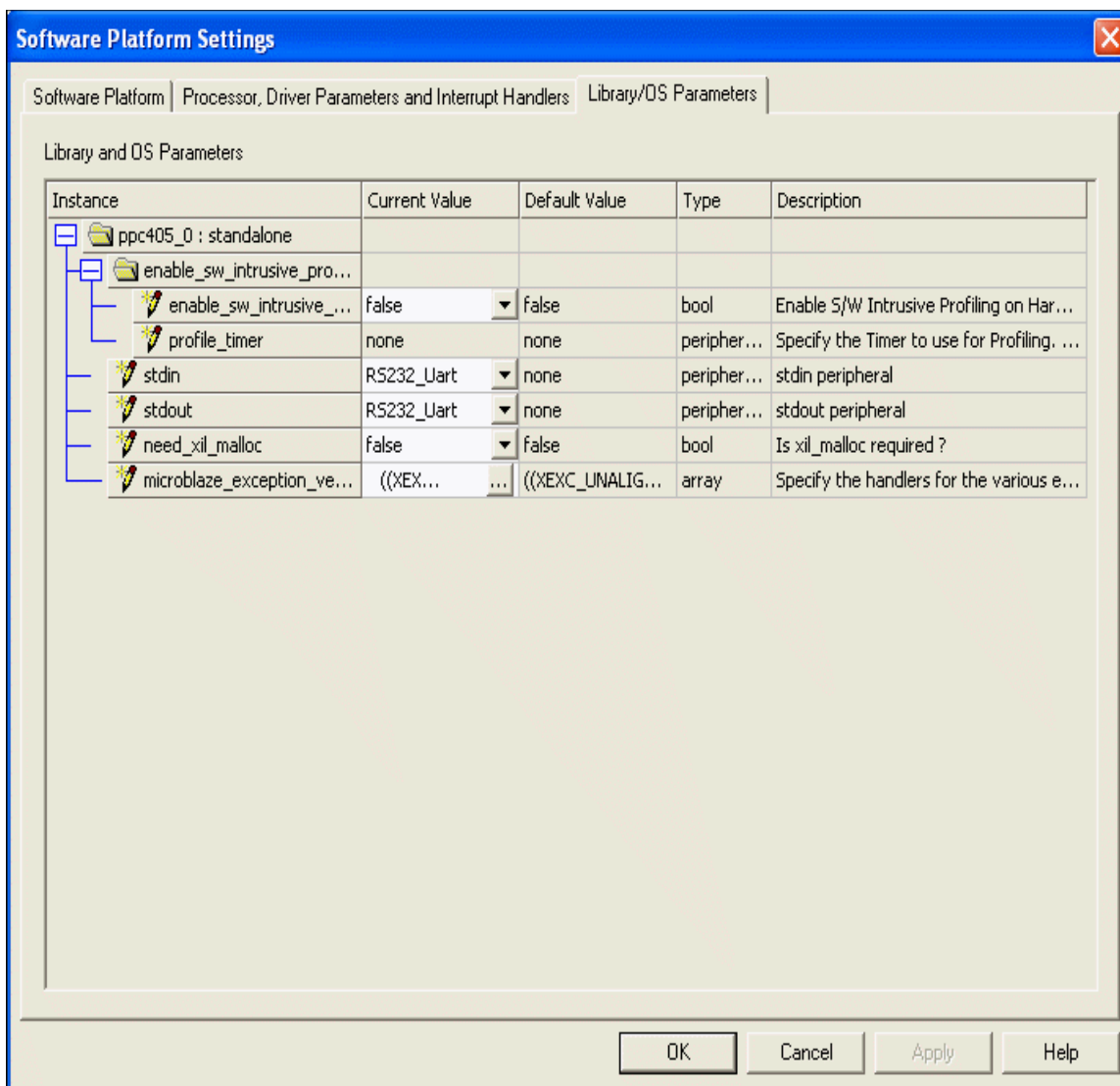


Figure 20: Library/OS Parameters

Click **OK**.

In XPS, select **Tools → Generate Libraries and BSPs** to run LibGen and create the BSP which includes device drivers, libraries, configures the STDIN/STDOUT, and Interrupt handlers associated with the design.

LibGen creates the following directories in the **ppc405_0** directory, shown in Figure 21:

- code: contains the compiled and linked application code in an ELF file
- include: contains the header files for peripherals included in the design (such as **xgpio.h** and **xuartlite.h**)
- lib: contains the library files (such as **libc.a** and **libxil.a**)
- libsrc: contains the source files used to create libraries

Note: **For more information on these files, refer to the Embedded System Tools Guide.**

Name	Size	Type
code		File Folder
include		File Folder
lib		File Folder
libsrc		File Folder

Figure 21: PowerPC Drivers Directories

Building the User Application

In EDK 7.1, XPS provides the ability for the user to create multiple software projects. These projects can include source files, header files, and linker scripts. Unique software projects allow the designer to specify the following options for each software project:

- Specify compiler options
- Specify which projects to compile
- Specify which projects to download
- Build entire projects

Software application code development can be managed by selecting the Applications tab as shown in Figure 22. The Base System Builder (BSB) generates a sample application which tests a subset of the peripherals included in the design.

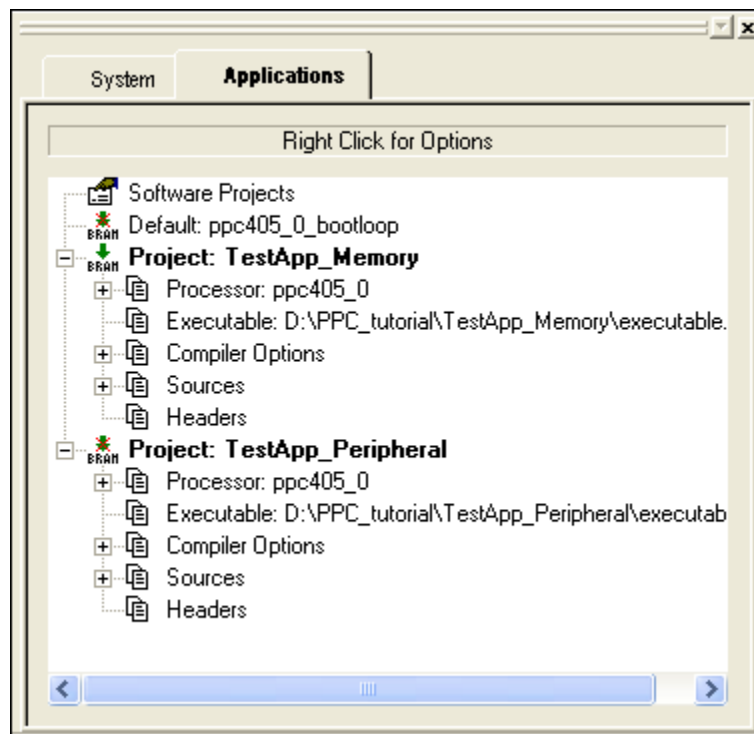


Figure 22: Applications Tab

Compiling the Code

Using the GNU GCC Compiler, compile the application code as follows:

Select **Tools** → **Build All User Applications** to run powerpc-eabi-gcc. Powerpc-eabi-gcc compiles the source files.

```

(Console Log)
Project Opened
At Local date and time: Thu Apr 21 11:51:01 2005
Command xbash -q -c "cd /cygdrive/d/PPC_tutorial/; /usr/bin/make -f system.make program; exit;" Started...

powerpc-eabi-gcc -O2 TestApp_Peripheral/src/TestApp_Peripheral.c -o TestApp_Peripheral/executable.elf \
-Wl,-T-Wl,TestApp_Peripheral/src/TestApp_Peripheral_LinkScr -g -I./ppc405_0/include/ -I./ppc405_0/lib/ \
powerpc-eabi-size TestApp_Peripheral/executable.elf
text      data      bss      dec      hex      filename
1350      808      4096      6254      186e      TestApp_Peripheral/executable.elf
Done.

```

Figure 23: XPS Output Window

VIII. Downloading the Design

Now that the hardware and software designs are completed, the device can be configured. Follow these steps to download and configure the FPGA:

Connect the host computer to the target board, including connecting the Parallel-JTAG cable and the serial cable.

Start a hyperterminal session with the following settings:

- com1 – This is dependant on the com port your serial cable is connected to.
- Bits per second: 57600
- Data bits: 8
- Parity: none
- Stop bits: 1
- Flow control: none

Connect the board power.

In ISE, Select system_stub.vhd in the source window. In the process window double click on Configure Device (iMPACT) under Generate Programming File.

This will initialize the BIT file with the software application selected in XPS and open iMPACT.

With iMPACT configure the FPGA using system_stub.bit located in the project_navigator directory.

After the configuration is complete, you should see a display similar to that in shown in Figure 24:

```

ML300 - HyperTerminal
File Edit View Call Transfer Help
-- Entering main() --
Starting MemoryTest for SRAM_256Kx32:
Running 32-bit test...PASSED!
Running 16-bit test...PASSED!
Running 8-bit test...PASSED!
-- Exiting main() --

```

Figure 24: Hyperterminal Output

IX. Debugging the Design

Now that the device is configured, you can debug the software application directly via the JTAG PPC connections. GDB connects to the PowerPC core through the JTAGPPC and the Xilinx Microprocessor Debug (XMD) engine utility as shown in Figure 19. XMD is a program that facilitates a unified GDB interface and a Tcl (Tool Command Language) interface for debugging programs and verifying systems using the MicroBlaze or PowerPC (Virtex-II Pro) microprocessor. The XMD engine is used with MicroBlaze and PowerPC GDB (mb-gdb & powerpc-eabi-gdb) for debugging. Mb-gdb and powerpc-eabi-gdb communicate with XMD using the remote TCP protocol and control the corresponding targets. GDB can connect to XMD on the same computer or on a remote Internet computer.

To debug the design, follow these steps:

Select Options → XMD Debug Options

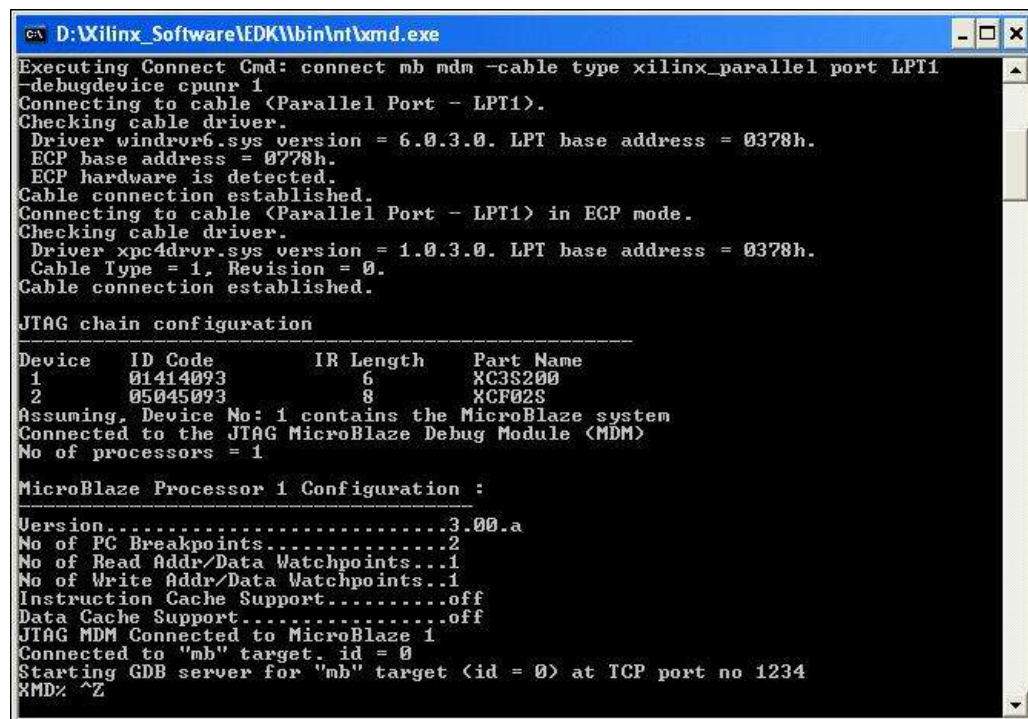
The XMD Debug Options dialog box allows the user to specify the connections type and JTAG Chain Definition. Two connection types are available for PowerPC:

Simulator – enables XMD to connect to the PowerPC ISS.

Hardware – enables XMD to connect to the JTAGPPC peripheral in the hardware.

Select Save.

Select Tools → XMD.



```

D:\Xilinx_Software\EDK\bin\nt\xmd.exe
Executing Connect Cmd: connect mb mdm -cable type xilinx_parallel port LPT1
-debugdevice cpunr 1
Connecting to cable <Parallel Port - LPT1>.
Checking cable driver.
Driver windrvr6.sys version = 6.0.3.0. LPT base address = 0378h.
ECP base address = 0778h.
ECP hardware is detected.
Cable connection established.
Connecting to cable <Parallel Port - LPT1> in ECP mode.
Checking cable driver.
Driver xpc4drv.sys version = 1.0.3.0. LPT base address = 0378h.
Cable Type = 1, Revision = 0.
Cable connection established.

JTAG chain configuration
-----
Device  ID Code      IR Length  Part Name
1       01414093        6          XC3S200
2       05045093        8          XCF02S
Assuming, Device No: 1 contains the MicroBlaze system
Connected to the JTAG MicroBlaze Debug Module (MDM)
No of processors = 1

MicroBlaze Processor 1 Configuration :
-----
Version.....3.00.a
No of PC Breakpoints.....2
No of Read Addr/Data Watchpoints...1
No of Write Addr/Data Watchpoints..1
Instruction Cache Support.....off
Data Cache Support.....off
JTAG MDM Connected to MicroBlaze 1
Connected to "mb" target. id = 0
Starting GDB server for "mb" target (id = 0) at TCP port no 1234
XMD% ^Z
  
```

Figure 25: XMD Window

In XPS, select **Tools → Software Debugger** to open the GDB interface.

In GDB, select **File → Target Settings** to display the Target Selection dialog box as shown in Figure 26. Click **OK**.

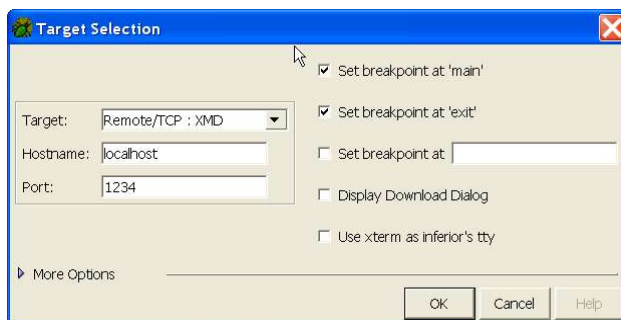


Figure 26: GDB Target Selection

In GDB, select **File → Open...**

Select executable.elf in the TestApp_Memory directory.

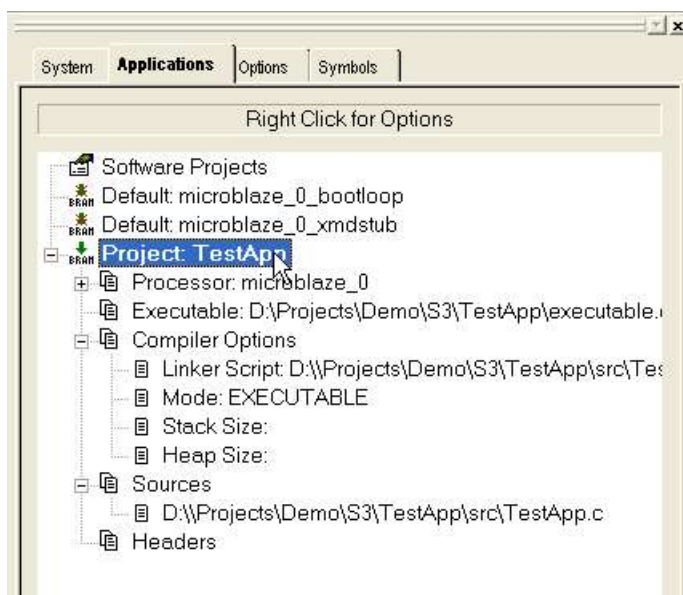
Tutorial Test Questions:

Do you see the C code or the assembly code? _____

Why can you not see the C code? _____

In GDB, select **File → Exit**.

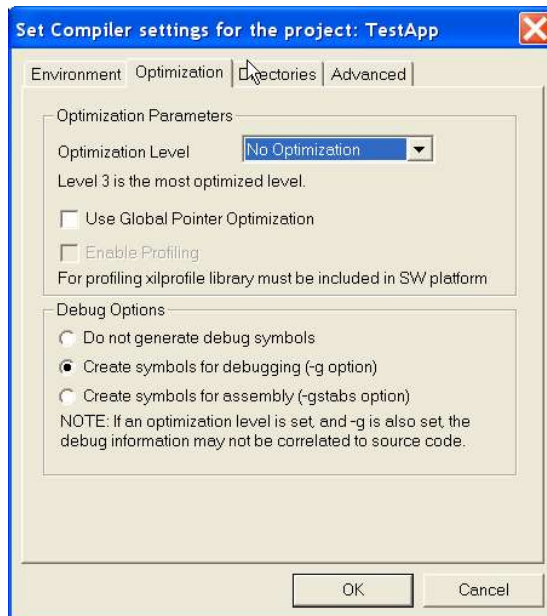
In the Applications window of XPS, double click on the Project: TestApp_Memory label.



In the Optimization tab;

- Set the Optimization Level to **No Optimization**
- Select Create symbols for debugging (-g option).

Click **OK**.



Recompile the code

Load the new executable.elf into GDB

Tutorial Test Question:

Do you see the C code? _____ (If you do not see the C code, repeat the steps above.)

Select Run → Run

There is an automatic breakpoint at main. GDB allows you to single step the C or assembly code.

Note: *The default values displayed in the Registers Window are in hex, while the values displayed in the Source Window are in decimal.*

X. Performing Behavioral Simulation of the Embedded System

Performing a behavioral simulation of the complete system, which includes the embedded processor system, is a powerful verification technique. In order to perform a behavioral simulation of the complete system in ISE, the simulation file for the embedded system must be generated.

In XPS, select **Options** → Project Options. In the Project Options dialog box select the HDL and Simulation tab.

Browse to the precompiled EDK Library and Xilinx Library as shown in figure 27. It should be noted that the paths will be different to match your system. For additional information on compiling the simulation libraries refer to the Platform Studio User Guide Chapter 8.

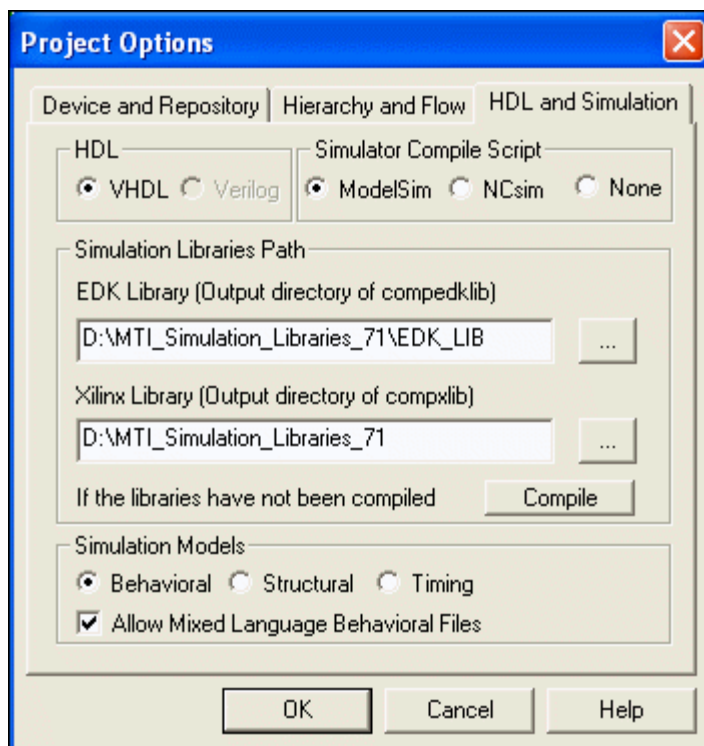


Figure 27: **Project Options – HDL and Simulation tab**

Click Ok. Select Tools → **Generate Simulation HDL Files**. This will generate all of the EDK HDL Simulation files in the EDK\simulation\behavioral directory created by SimGen.

In order to simulate a design incorporating both EDK and ISE files, the simulation environments must be merged. Copy the behavioral directory to the project_navigator directory.

Now that the EDK simulation files have been created and copied to the ISE directory, the ISE simulation environment can be created.

In ISE, select system_stub.vhd and double click on Create New Source in the Process Window.

In the New Source dialog, select the source type as “VHDL Test Bench” and the File Name as “Testbench”

Click Next. Select system_stub as the source file to which the testbench will be associated.

Click Next and Finish. Testbench.vhd will now open in the ISE Editor Window.

Scroll to the bottom of the file and remove the following code:

```
tb : PROCESS
BEGIN
    -- Wait 100 ns for global reset to finish
    wait for 100 ns;

    -- Place stimulus here

    wait; -- will wait forever
END PROCESS;
```

Add the following code:

```
tb_clk : PROCESS
BEGIN
    sys_clk_pin <= '1'; wait for 10 ns;
    sys_clk_pin <= '0'; wait for 10 ns;
END PROCESS;

tb_reset : PROCESS
BEGIN
    sys_rst_pin <= '0'; wait for 5 us;
    sys_rst_pin <= '1'; wait;
END PROCESS;

fpga_0_RS232_Uart_RX_pin <= fpga_0_RS232_Uart_TX_pin;
```

In order to populate the BRAMs with the TestApp_Memory Application, a configuration statement must be utilized. Add the following after the final “END;” statement in the Testbench.vhd file:

```
configuration tb_conf of Testbench_vhd is
for behavior
    for uut: system_stub
        for structure
            for system_i : system
                use configuration work.system_conf;
            end for;
        end for;
    end for;
end for;
end tb_conf;
```

Save and close the Testbench.vhd file.

Select Testbench.vhd in the ISE Source Window. Right-click on “Simulate Behavioral Model” and select Properties...

In the Process Properties dialog box, specify the following settings as shown in figure 28:

- Check -> Use Custom Do File
- Select system_setup.do in the project_navigator\behavioral directory
- Uncheck -> Use Automatic Do File

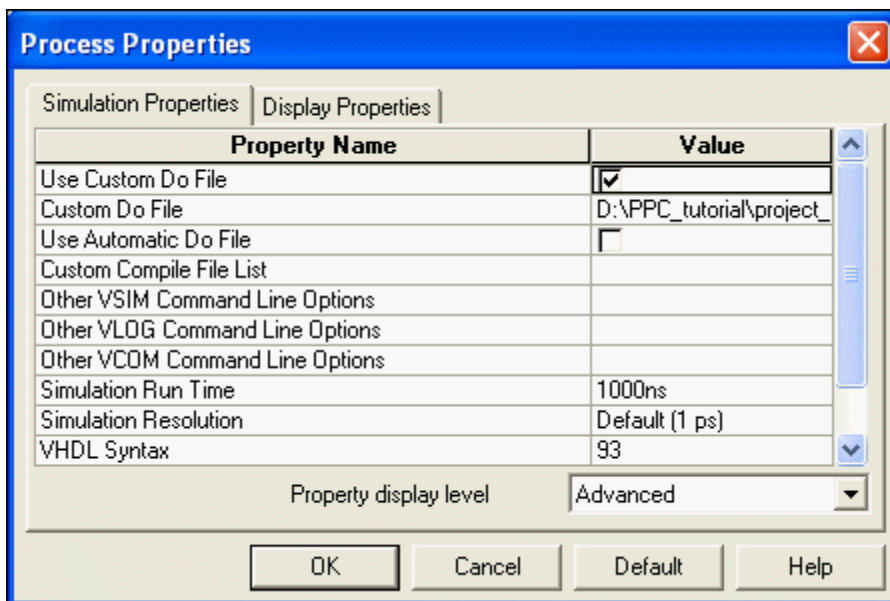


Figure 28: ISE Behavioral Simulation Process Properties

In the project_navigator/behavioral directory, open system_setup.do.

Add the following line as the first line in the file:

```
set tbpath "/testbench_vhd/uut"
cd behavioral
```

This is used to update the hierarchy used in the EDK simulation wave and list DO files.

Change the following command:

```
alias s "vsim -t ps testbench"
```

to:

```
alias s "vsim -t ps tb_conf"
```

Save and close the file.

Open system.do and add the following lines to the end of the file to compile the ISE files:

```
vcom -93 -work work ../system_stub.vhd
vcom -93 -work work ../testbench.vhd
```

Save and close the file.

Start the simulation by right-clicking on Simulate Behavioral Model and selecting "Run" and then "Open without Updating".

Once Modelsim opens the following commands are available:

```
c => compile the testbench by running the EDK compile script.
      Edit the testbench first before running this command.
```

Assumes ISE and EDK libraries were compiled earlier for ModelSim. (see system.do)

- s => load the testbench for simulation. (ModelSim 'vsim' command with 'testbench') After loading the testbench, set up signal displays (optional) and run the simulation. (ModelSim 'run' command)
- l => set up signal list display and launch a list window. ModelSim 'add -list' commands are found in *_list.do scripts. (see system_list.do)
- w => set up signal wave display and launch a waveform window. ModelSim 'add -wave' commands are found in *_wave.do scripts. (see system_wave.do)

At the command prompt type “c” to compile the source

At the command prompt type “s” to load the simulation

At the command prompt type “w” to load the wave window

At the command prompt type “run 300us” to begin running the simulation. It will take several thousand US to run the design to simulate the functionality of the design because of the printf routines.