EDK 7.1 MicroBlaze Tutorial in Spartan 3

Objectives

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

- System Requirements
- MicroBlaze 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 or later
- ISE 7.1i SP1 or later

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

- Xilinx Spartan-3 Evaluation Board (3S200 FT256 -4)
- 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

MicroBlaze System Description

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

- Hardware components
- Memory map
- Software application

Tutorial Design Hardware

The MicroBlaze (MB) tutorial design includes the following hardware components:

- MicroBlaze
- Local Memory Bus (LMB) Bus
  - ♦ LMB_BRAM_IF_CNTLR
  - ♦ BRAM_BLOCK
- On-chip Peripheral Bus (OPB) BUS
  - ♦ OPB_MDM
  - ♦ OPB_UARTLITE
  - ♦ 3 - OPB_GPIOs
  - ♦ OPB_EMC

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		
LMB_BRAM	0x0000_0000	0x0000_1FFF	8K bytes	LMB Memory
OPB_MDM	0x4140_0000	0x4140_FFFF	64K bytes	MDM Module
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
OPB_GPIO	0x4004_0000	0x4004_FFFF	64K bytes	DIP switches
SRAM (EMC MEM0)	0x2010_0000	0x201F_FFFF	512K bytes	SRAM Memory

Table 1: **Tutorial Design Memory Map**

Tutorial Steps

SetUp

- Spartan-3 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 MicroBlaze 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

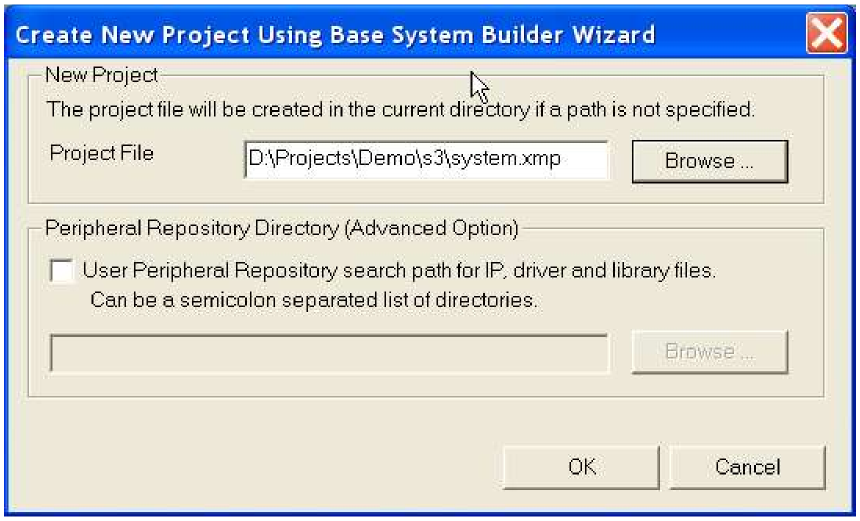
Select Base System Builder Wizard (BSB) to open the Create New Project Using BSB Wizard dialog box shown in Figure 1.

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.


Create New Project Using Base System Builder Wizard

New Project

The project file will be created in the current directory if a path is not specified.

Project File: D:\Projects\Demo\s3\system.xmp

Peripheral Repository Directory (Advanced Option)

User Peripheral Repository search path for IP, driver and library files. Can be a semicolon separated list of directories.

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.

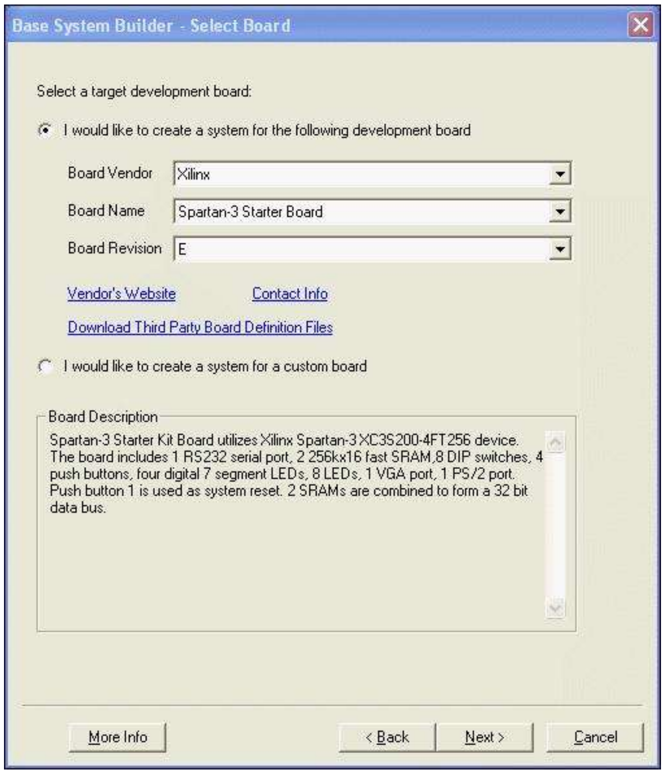
III. Using the Base System Builder Wizard

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 Vendo*r:* Xilinx
- Board Nam*e:* Spartan-3 Starter Board
- Board Revisio*n:* E


Base System Builder - Select Board

Select a target development board:

(●) I would like to create a system for the following development board

Board Vendor: Xilinx

Board Name: Spartan-3 Starter Board

Board Revision: E

Vendor's Website
Contact Info
Download Third Party Board Definition Files

( ) I would like to create a system for a custom board

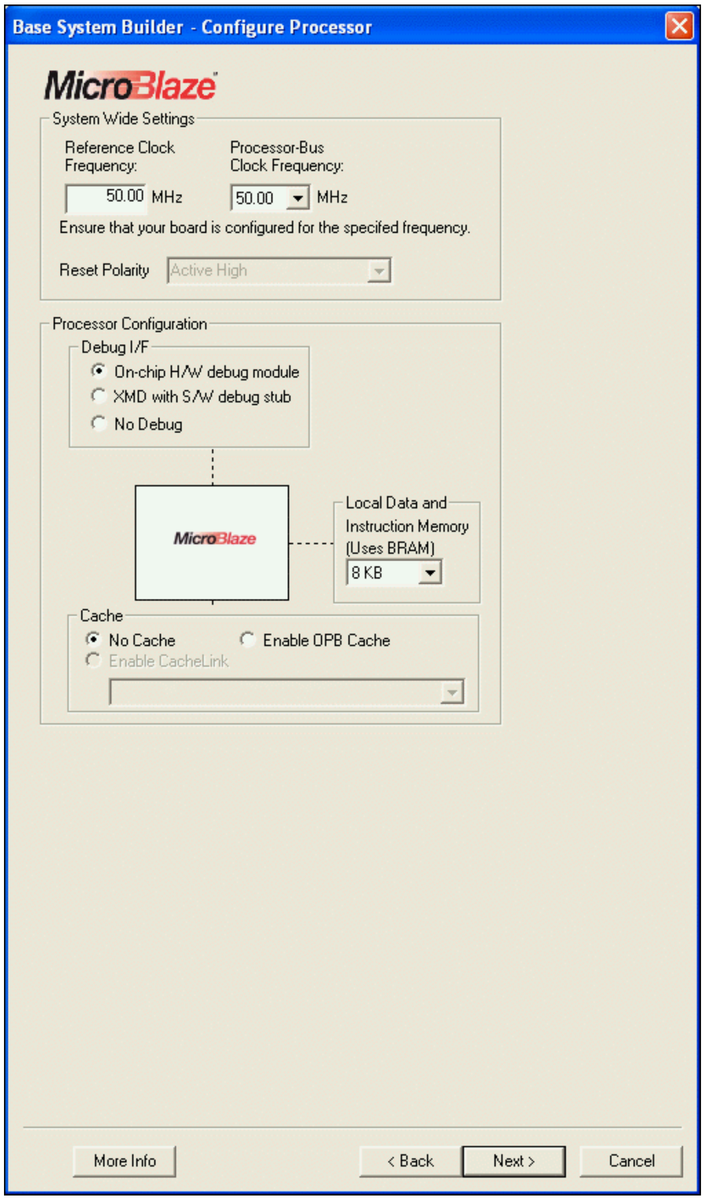
Board Description

Spartan-3 Starter Kit Board utilizes Xilinx Spartan-3 XC3S200-4FT256 device. The board includes 1 RS232 serial port, 2 256kx16 fast SRAM,8 DIP switches, 4 push buttons, four digital 7 segment LEDs, 8 LEDs, 1 VGA port, 1 PS/2 port. Push button 1 is used as system reset. 2 SRAMs are combined to form a 32 bit data bus.

Figure 2: **BSB: Select a Board**

Click ***Next***. MicroBlaze is the only processor option for this board.

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



System Wide Settings

Reference Clock Frequency: 50.00 MHz

Processor-Bus Clock Frequency: 50.00 MHz

Ensure that your board is configured for the specifed frequency.

Reset Polarity: Active High

Processor Configuration

Debug I/F

[✓ On-chip H/W debug module]

XMD with S/W debug stub

No Debug

Local Data and Instruction Memory (Uses BRAM): 8 KB

Cache

[✓ No Cache]

Enable OPB Cache

Enable CacheLink

Figure 3: **Configure Processor**

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

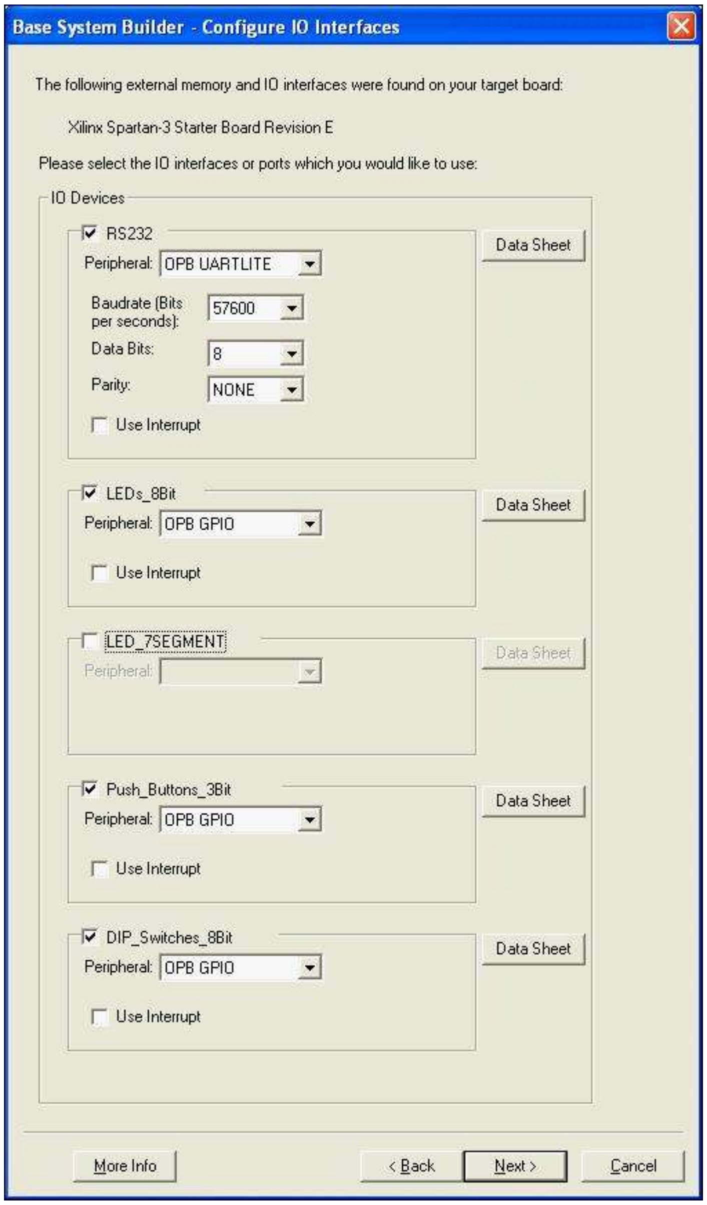
- System Wide Setting:
  - Processor Clock Frequency: This is the frequency of the clock driving the processor system.
- Processor Configuration:
  - Debug I/F:
  - XMD with S/W Debug stub: Selecting this mode of debugging interface introduces a software intrusive debugging. There is a 1200-byte stub that is located at 0x00000000. This stub communicates with the debugger on the host through the JTAG interface of the OPB MDM module.
  - On-Chip H/W Debug module: When the H/W debug module is selected; an OPB MDM module is included in the hardware system. This introduces hardware intrusive debugging with no software stub required. This is the recommended way of debugging for MicroBlaze system.
  - 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 local instruction and data memory.
- You can also specify the use of a cache.

Click ***Next***. Select the peripheral subset as shown in Figure 4 and Figure 5. It should be noted that the number of peripheral shown on each dialog box is dynamic based upon your computers resolution.

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


Base System Builder - Configure IO Interfaces

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

Xilinx Spartan-3 Starter Board Revision E

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

IO Devices

[✓] RS232
Peripheral: OPB UARTLITE
Baudrate (Bits per seconds): 57600
Data Bits: 8
Parity: NONE
[ ] Use Interrupt

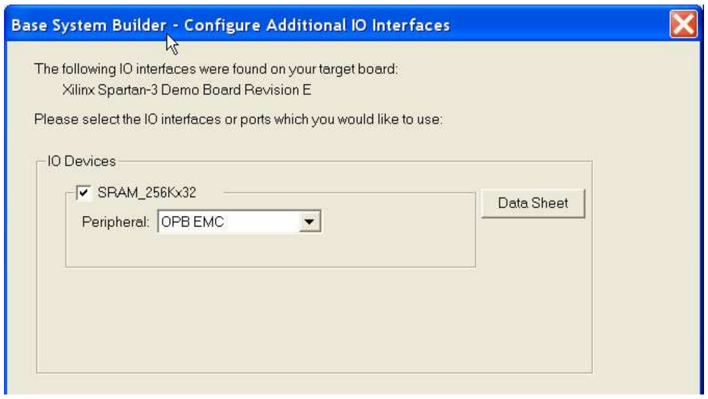
[✓] LEDs_8Bit
Peripheral: OPB GPIO
[ ] Use Interrupt

[ ] LED_7SEGMENT
Peripheral:

[✓] Push_Buttons_3Bit
Peripheral: OPB GPIO
[ ] Use Interrupt

[✓] DIP_Switches_8Bit
Peripheral: OPB GPIO
[ ] Use Interrupt

Figure 4: **Peripheral Selection**


Base System Builder - Configure Additional IO Interfaces

The following IO interfaces were found on your target board:
Xilinx Spartan-3 Demo Board Revision E

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

IO Devices

[✓] SRAM_256Kx32

Peripheral: OPB EMC

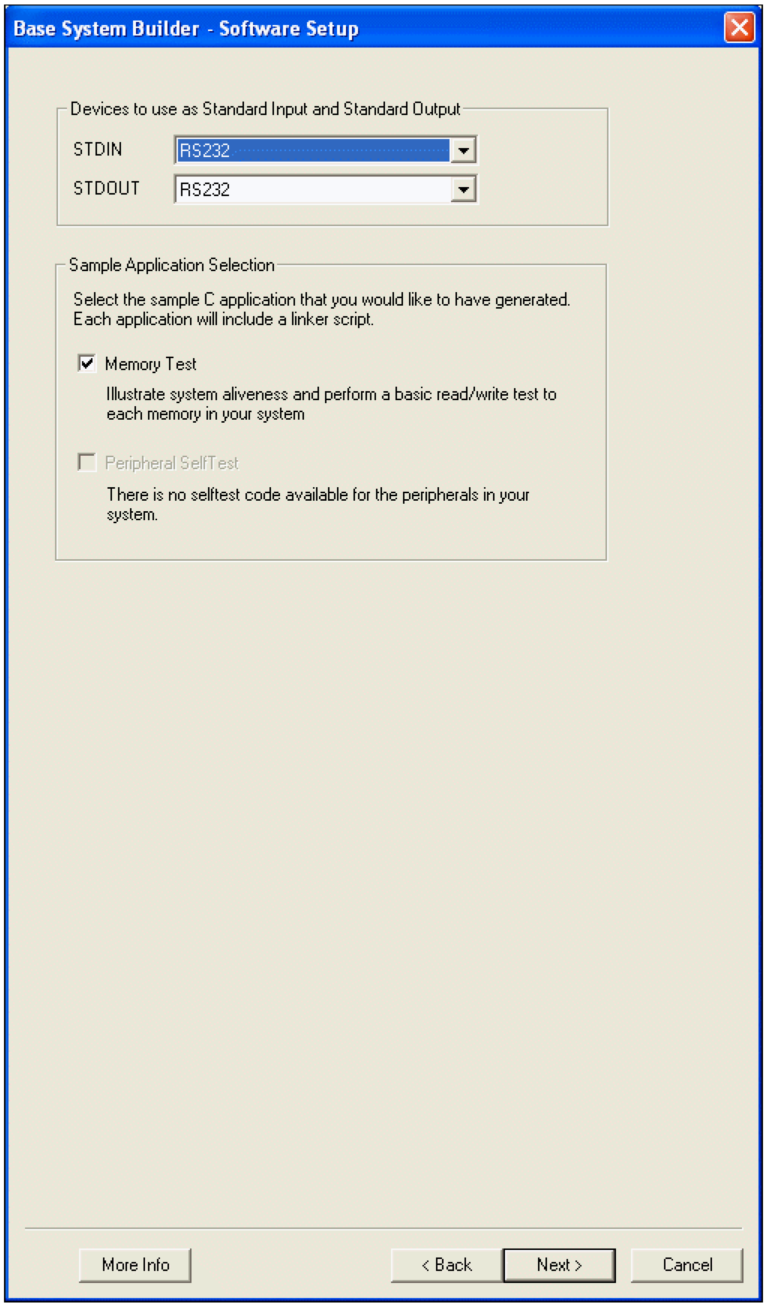
Figure 5: **Peripheral Selection**

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

Click ***Next*** through the Add Internal Peripherals page as we will not add any in this example. This completes the hardware specification and we will now configure the software settings.

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

- Standard Input (STDIN) → RS232
- Standard Output (STDOUT) → RS232
- Sample Application Selection → Memory Test



Devices to use as Standard Input and Standard Output

STDIN RS232

STDOUT RS232

Sample Application Selection

Select the sample C application that you would like to have generated. Each application will include a linker script.

[✓] Memory Test

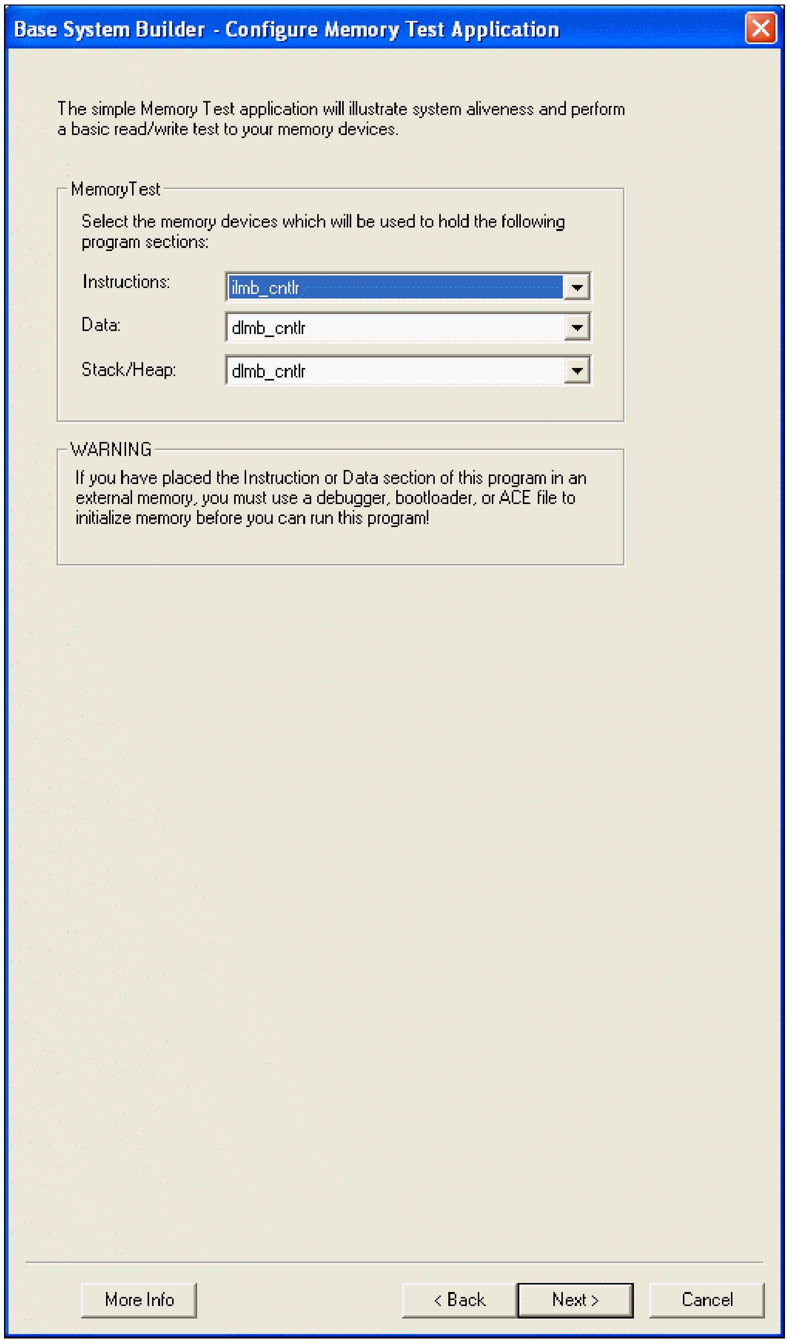
Illustrate system aliveness and perform a basic read/write test to each memory in your system

[ ] Peripheral SelfTest

There is no selftest code available for the peripherals in your system.

Figure 6: **Software Configuration**

Click ***Next***.


Base System Builder - Configure Memory Test Application

The simple Memory Test application will illustrate system aliveness and perform a basic read/write test to your memory devices.

MemoryTest

Select the memory devices which will be used to hold the following program sections:

Instructions: ilmb_cntlr

Data: dlmb_cntlr

Stack/Heap: dlmb_cntlr

WARNING

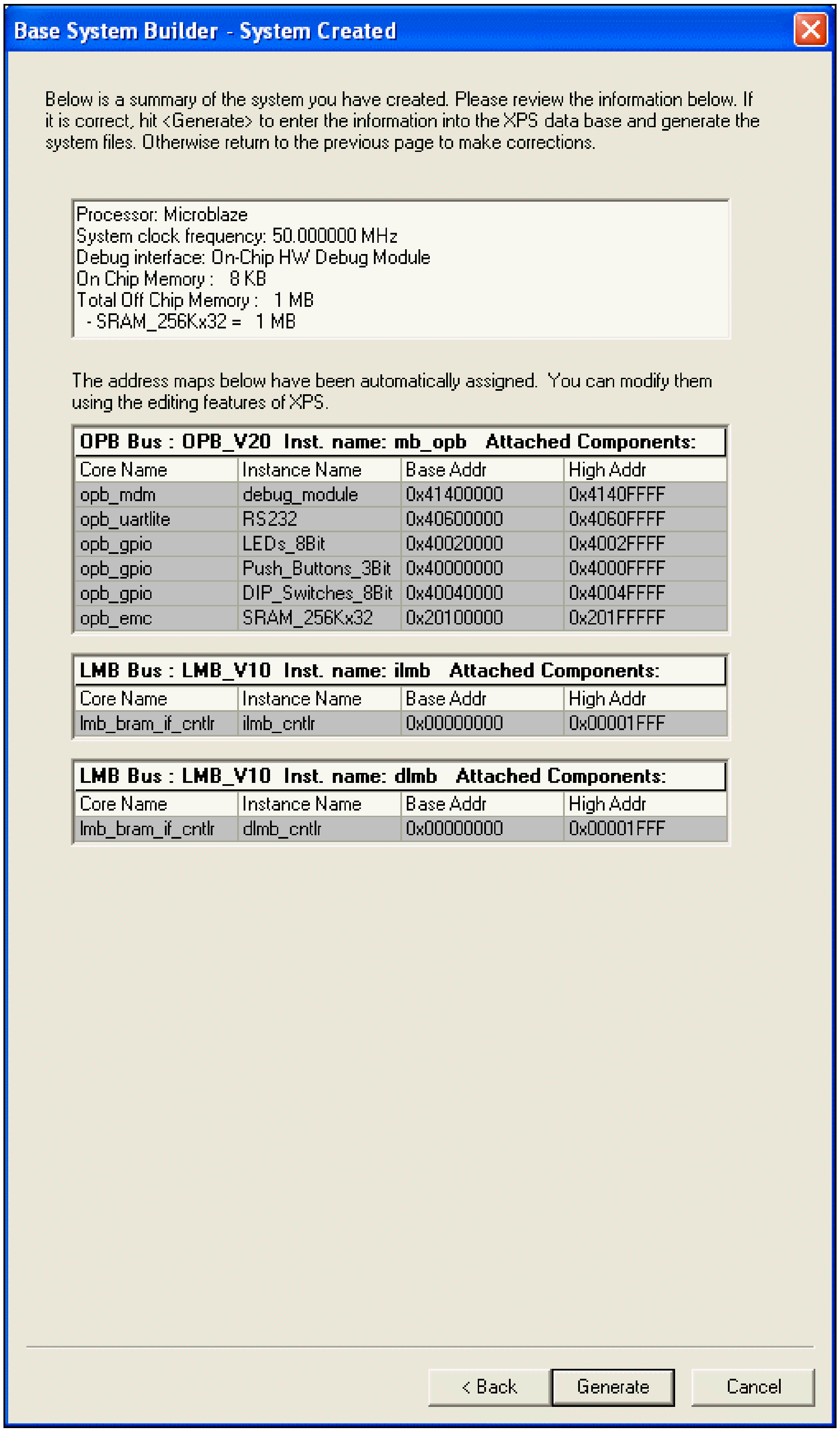
If you have placed the Instruction or Data section of this program in an external memory, you must use a debugger, bootloader, or ACE file to initialize memory before you can run this program!

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 → ilmb_cntlr
- Data → dlmb_cntlr
- Stack/Heap → dlmb_cntlr

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.



Below is a summary of the system you have created. Please review the information below. If it is correct, hit <Generate> to enter the information into the XPS data base and generate the system files. Otherwise return to the previous page to make corrections.

```
Processor: Microblaze
System clock frequency: 50.000000 MHz
Debug interface: On-Chip HW Debug Module
On Chip Memory :   8 KB
Total Off Chip Memory :   1 MB
 - SRAM_256Kx32 =   1 MB
```

The address maps below have been automatically assigned. You can modify them using the editing features of XPS.

OPB Bus : OPB_V20 Inst. name: mb_opb Attached Components:

Core Name	Instance Name	Base Addr	High Addr
opb_mdm	debug_module	0x41400000	0x4140FFFF
opb_uartlite	RS232	0x40600000	0x4060FFFF
opb_gpio	LEDs_8Bit	0x40020000	0x4002FFFF
opb_gpio	Push_Buttons_3Bit	0x40000000	0x4000FFFF
opb_gpio	DIP_Switches_8Bit	0x40040000	0x4004FFFF
opb_emc	SRAM_256Kx32	0x20100000	0x201FFFFF

LMB Bus : LMB_V10 Inst. name: ilmb Attached Components:

Core Name	Instance Name	Base Addr	High Addr
lmb_bram_if_cntlr	ilmb_cntlr	0x00000000	0x00001FFF

LMB Bus : LMB_V10 Inst. name: dlmb Attached Components:

Core Name	Instance Name	Base Addr	High Addr
lmb_bram_if_cntlr	dlmb_cntlr	0x00000000	0x00001FFF

Figure 7: **Completed Processor System**

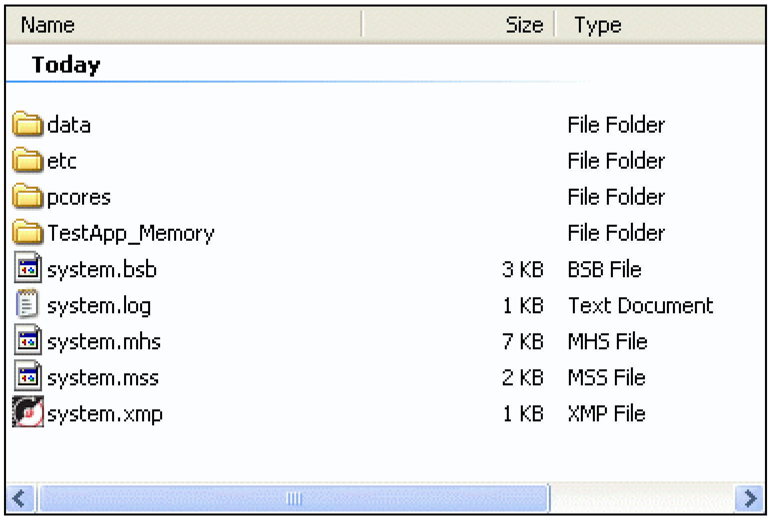
Click ***Generate*** and then ***Finish,*** to complete the design.

Select Start Using Plarform 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.

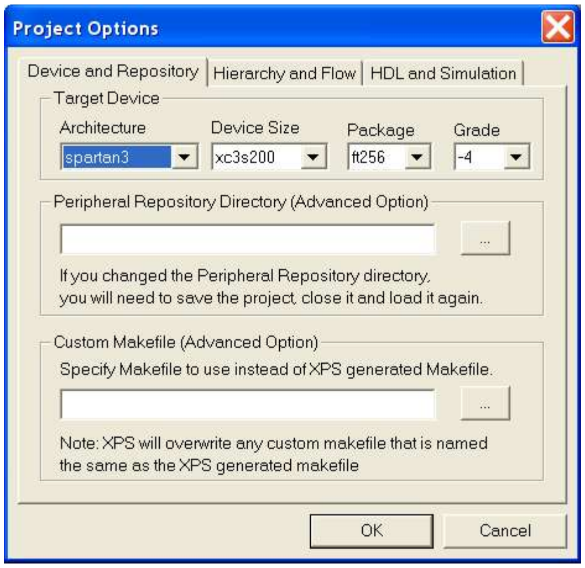


Name	Size	Type
Today		
data		File Folder
etc		File Folder
pcores		File Folder
TestApp_Memory		File Folder
system.bsb	3 KB	BSB File
system.log	1 KB	Text Document
system.mhs	7 KB	MHS File
system.mss	2 KB	MSS File
system.xmp	1 KB	XMP File

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.



Target Device

Architecture	Device Size	Package	Grade
spartan3	xc3s200	ft256	-4

Peripheral Repository Directory (Advanced Option)

If you changed the Peripheral Repository directory, you will need to save the project, close it and load it again.

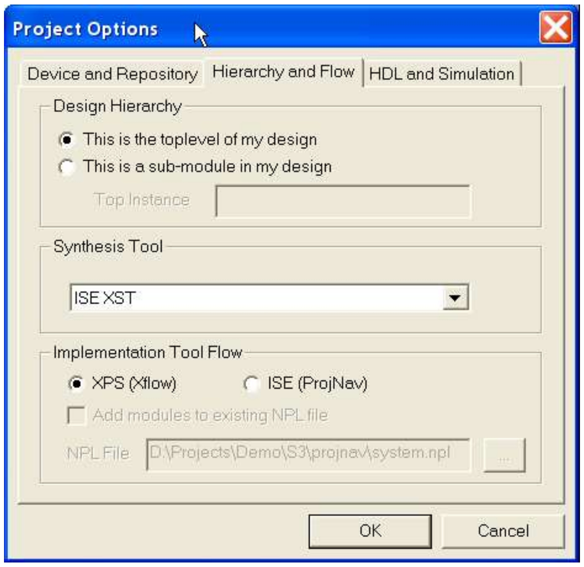
Custom Makefile (Advanced Option)

Specify Makefile to use instead of XPS generated Makefile.

Note: XPS will overwrite any custom makefile that is named the same as the XPS generated makefile

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.



Design Hierarchy

[✓ This is the toplevel of my design]
This is a sub-module in my design
Top Instance

Synthesis Tool: ISE XST

Implementation Tool Flow

[✓ XPS (Xflow)] ISE (ProjNav)
Add modules to existing NPL file
NPL File: D:\Projects\Demo\S3\projnav\system.npl

Figure 10: **Project Options - Hierarchy and Flow**

IV. Create – Import IP Peripheral

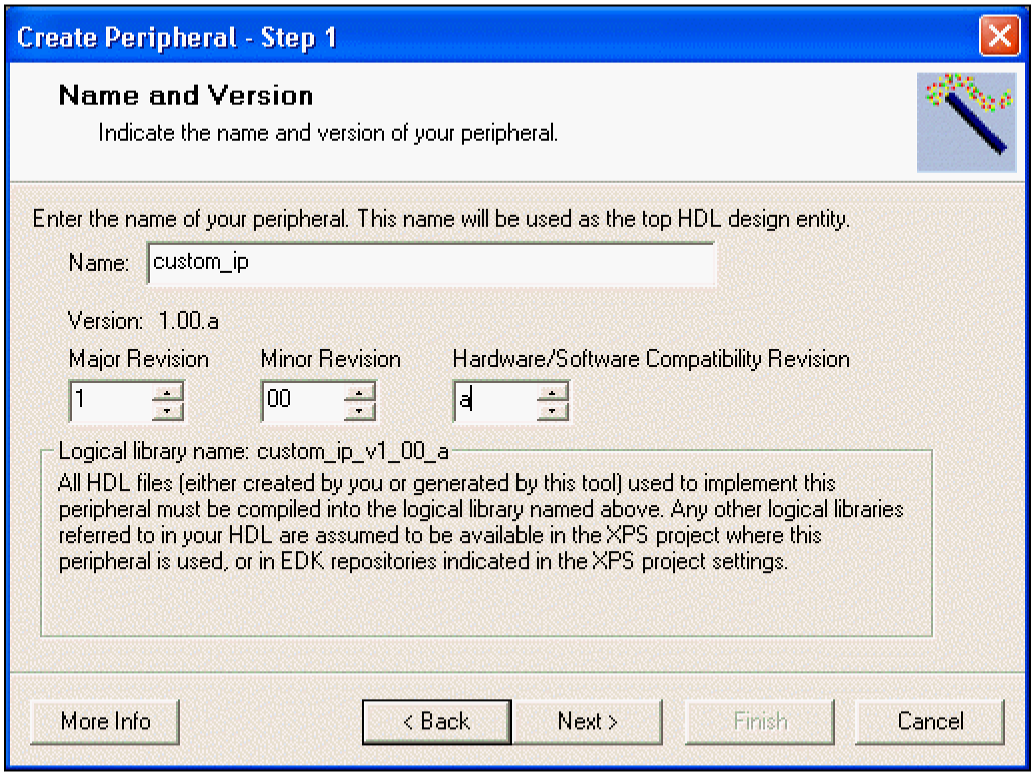
One of the key advantages of building an embedded system in an FGPA is the ability to include customer IP and interface that IP to the processor. This section of the tutorial will walk through the steps necessary to include a custom IP core.

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: custom_ip

Version: 1.00.a

Major Revision: 1

Minor Revision: 00

Hardware/Software Compatibility Revision: a

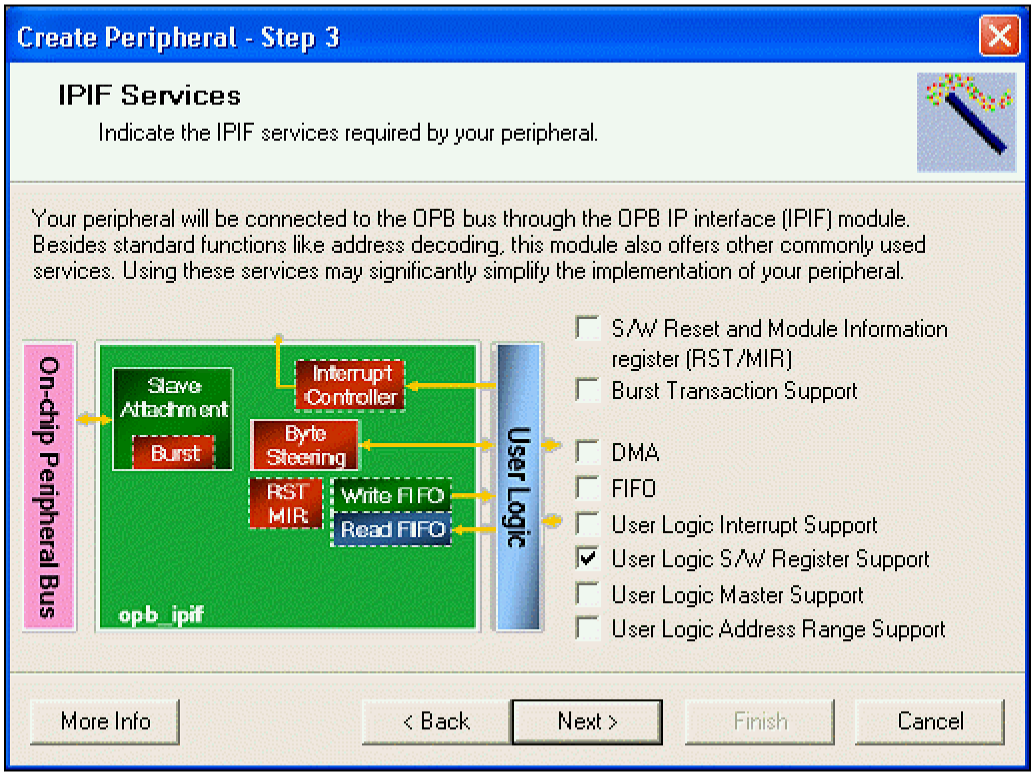
Logical library name: custom_ip_v1_00_a

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.



Create Peripheral - Step 3

IPIF Services

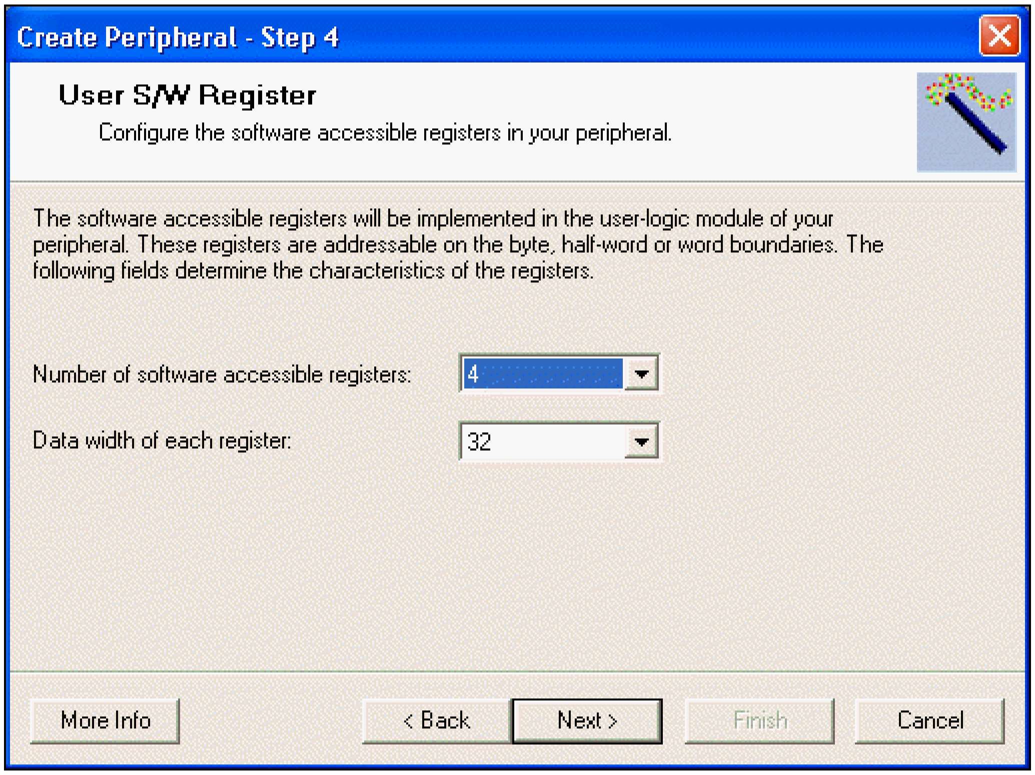
Indicate the IPIF services required by your peripheral.

Your peripheral will be connected to the OPB bus through the OPB IP interface (IPIF) module. Besides standard functions like address decoding, this module also offers other commonly used services. Using these services may significantly simplify the implementation of your peripheral.

- [ ] S/W Reset and Module Information register (RST/MIR)
- [ ] Burst Transaction Support
- [ ] DMA
- [ ] FIFO
- [ ] User Logic Interrupt Support
- [✓] User Logic S/W Register Support
- [ ] User Logic Master Support
- [ ] User Logic Address Range Support

Figure 12: Create Peripheral – Step 3

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



Create Peripheral - Step 4

User S/W Register

Configure the software accessible registers in your peripheral.

The software accessible registers will be implemented in the user-logic module of your peripheral. These registers are addressable on the byte, half-word or word boundaries. The following fields determine the characteristics of the registers.

Number of software accessible registers: 4

Data width of each register: 32

Figure 12: Create Peripheral – Step 4

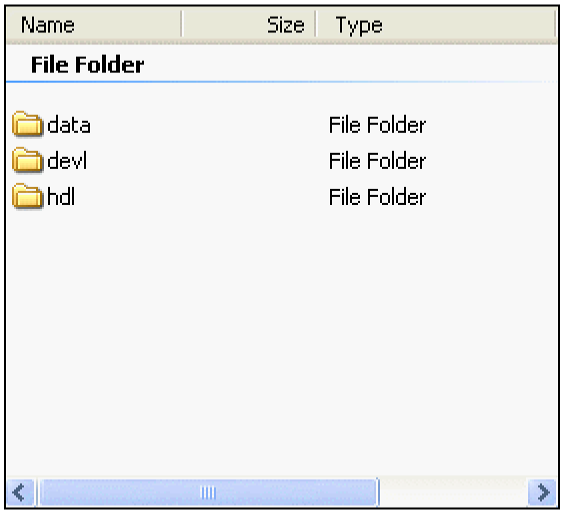
Click ***Next***. In the Create Peripheral –Step 5 dialog no additional signal changes are required.

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:



Name	Size	Type
File Folder		
data		File Folder
devl		File Folder
hdl		File Folder

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 functionalites 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:

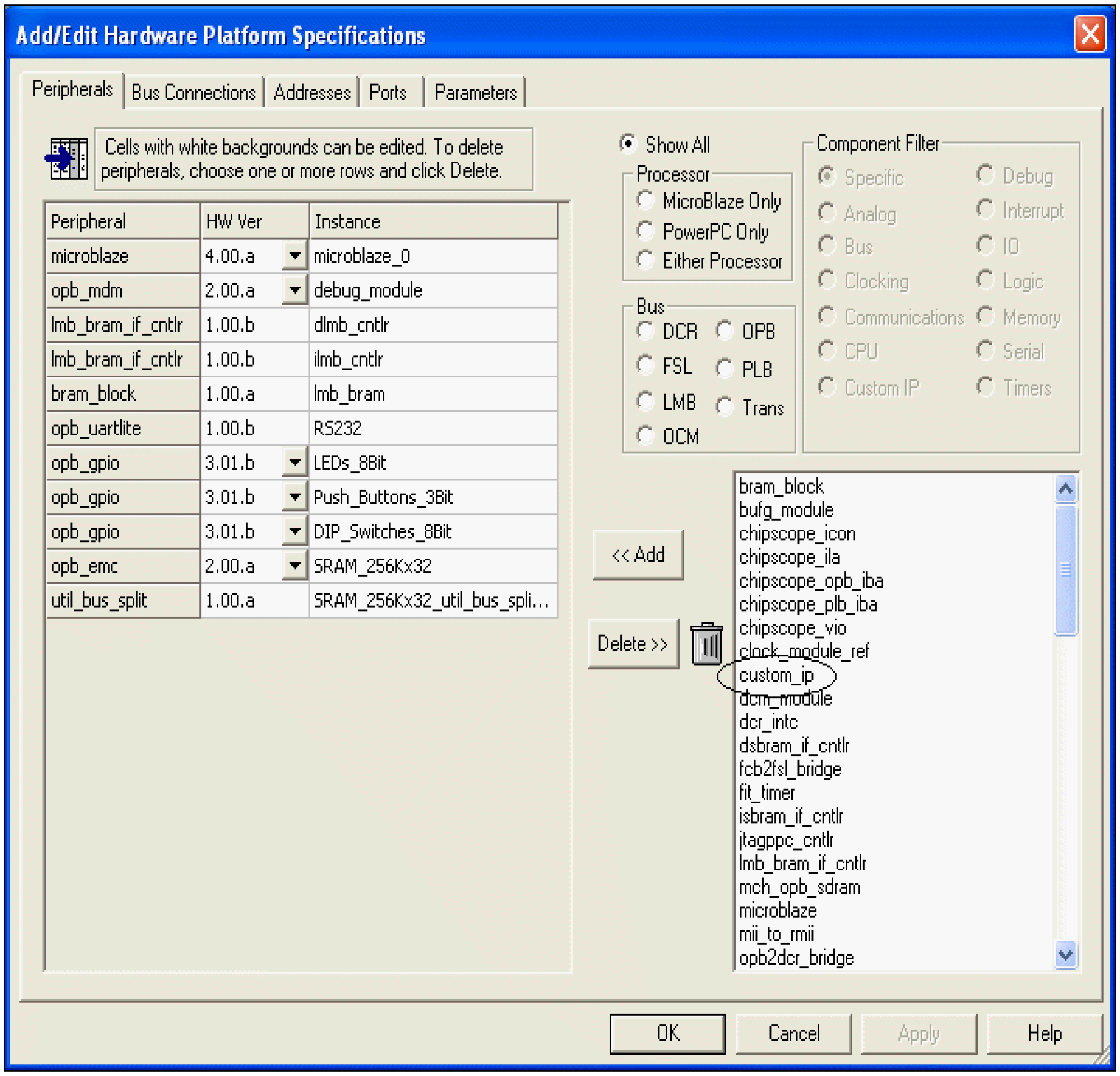
Add new peripherals

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.



Cells with white backgrounds can be edited. To delete peripherals, choose one or more rows and click Delete.

Peripheral	HW Ver	Instance
microblaze	4.00.a	microblaze_0
opb_mdm	2.00.a	debug_module
lmb_bram_if_cntlr	1.00.b	dlmb_cntlr
lmb_bram_if_cntlr	1.00.b	ilmb_cntlr
bram_block	1.00.a	lmb_bram
opb_uartlite	1.00.b	RS232
opb_gpio	3.01.b	LEDs_8Bit
opb_gpio	3.01.b	Push_Buttons_3Bit
opb_gpio	3.01.b	DIP_Switches_8Bit
opb_emc	2.00.a	SRAM_256Kx32
util_bus_split	1.00.a	SRAM_256Kx32_util_bus_spli...

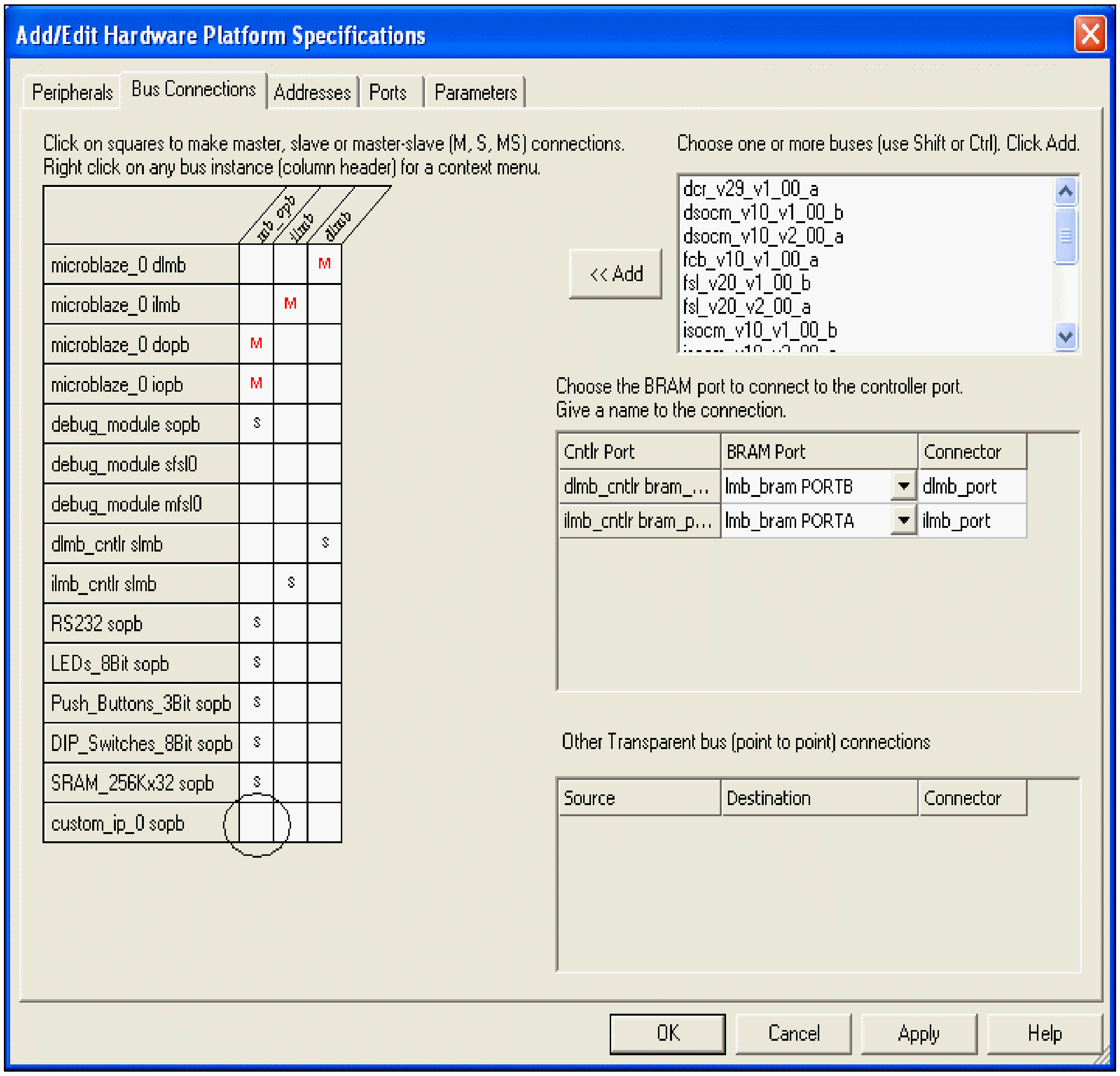
bram_block
bufg_module
chipscope_icon
chipscope_ila
chipscope_opb_iba
chipscope_plb_iba
chipscope_vio
clock_module_ref
custom_ip
dcm_module
dcr_intc
dsbram_if_cntlr
fcb2fsl_bridge
fit_timer
isbram_if_cntlr
jtagppc_cntlr
lmb_bram_if_cntlr
mch_opb_sdram
microblaze
mii_to_rmii
opb2dcr_bridge

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.


Add/Edit Hardware Platform Specifications

Peripherals | Bus Connections | Addresses | Ports | Parameters

Click on squares to make master, slave or master-slave (M, S, MS) connections.
Right click on any bus instance (column header) for a context menu.

	mb_opb	ilmb	dlmb
microblaze_0 dlmb			M
microblaze_0 ilmb		M	
microblaze_0 dopb	M		
microblaze_0 iopb	M		
debug_module sopb	S		
debug_module sfsl0			
debug_module mfsl0			
dlmb_cntlr slmb			S
ilmb_cntlr slmb		S	
RS232 sopb	S		
LEDs_8Bit sopb	S		
Push_Buttons_3Bit sopb	S		
DIP_Switches_8Bit sopb	S		
SRAM_256Kx32 sopb	S		
custom_ip_0 sopb			

Choose one or more buses (use Shift or Ctrl). Click Add.

dcr_v29_v1_00_a
dsocm_v10_v1_00_b
dsocm_v10_v2_00_a
fcb_v10_v1_00_a
fsl_v20_v1_00_b
fsl_v20_v2_00_a
isocm_v10_v1_00_b

<< Add

Choose the BRAM port to connect to the controller port.
Give a name to the connection.

Cntlr Port	BRAM Port	Connector
dlmb_cntlr bram_...	lmb_bram PORTB	dlmb_port
ilmb_cntlr bram_p...	lmb_bram PORTA	ilmb_port

Other Transparent bus (point to point) connections

Source	Destination	Connector

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.

Selct 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.

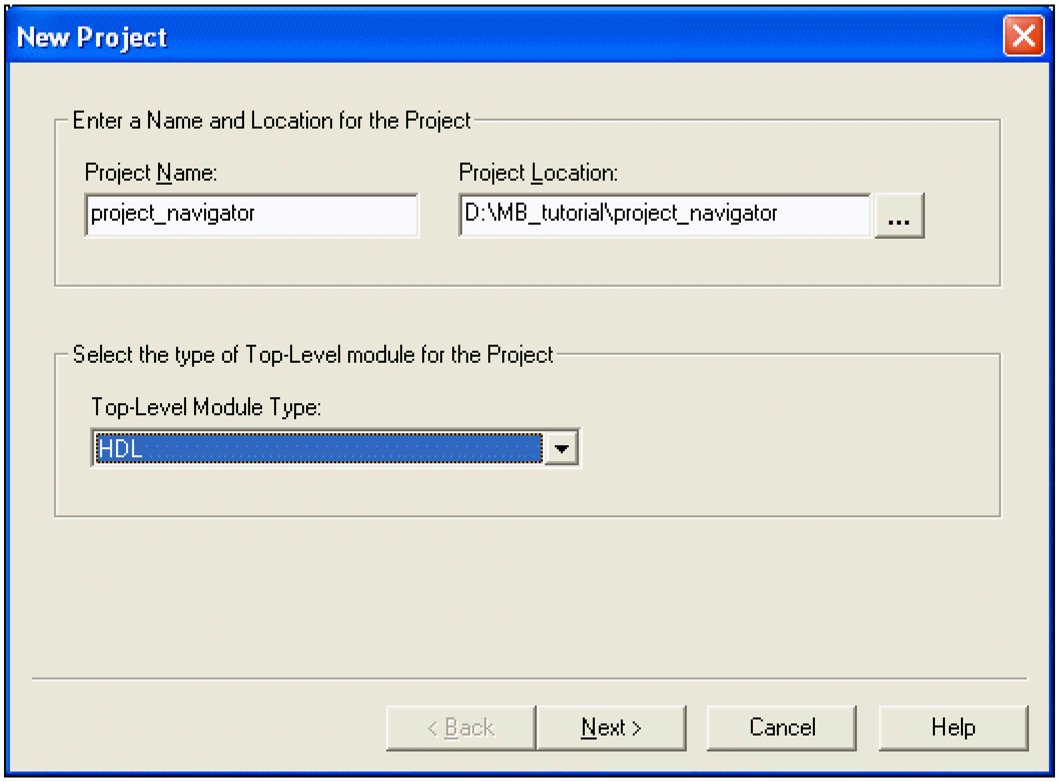
VI. Implementing the Design

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.

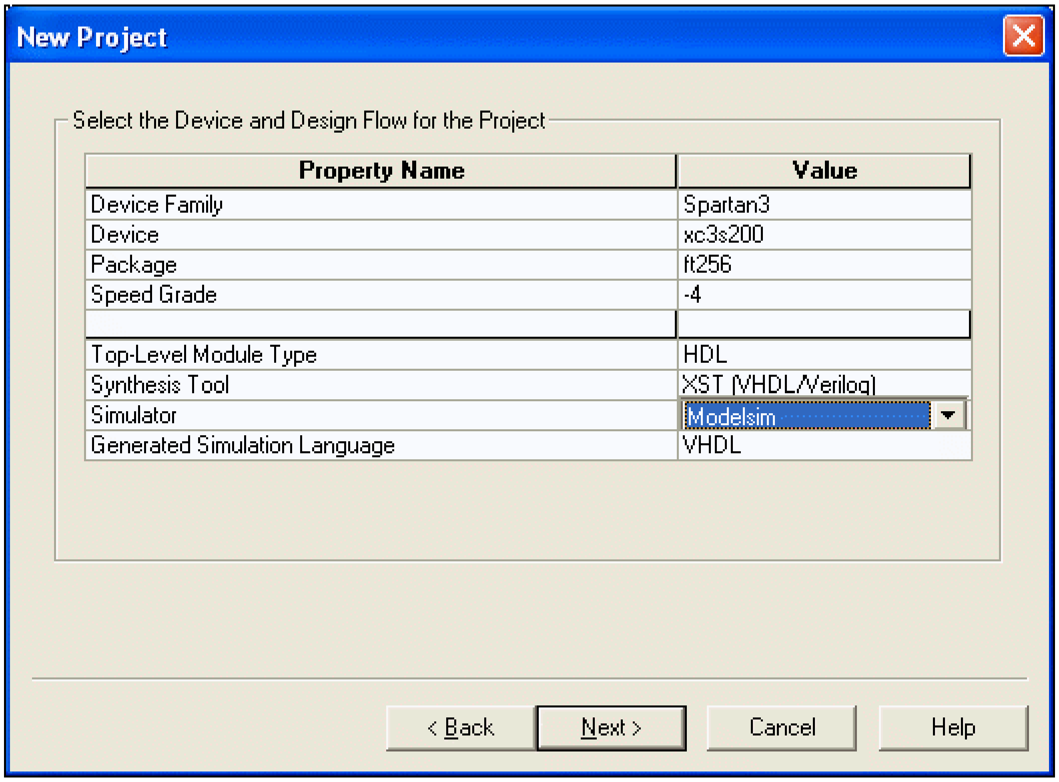
In the New Project dialog box shown in figure 15, browse to the XPS project directory and then enter the Project Name, project_navigator.


New Project
Enter a Name and Location for the Project
Select the type of Top-Level module for the Project
...
< Back
Next >
Cancel
Help

Project Name: project_navigator
Project Location: D:\MB_tutorial\project_navigator
Top-Level Module Type: HDL

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.



Property Name	Value
Device Family	Spartan3
Device	xc3s200
Package	ft256
Speed Grade	-4
Top-Level Module Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim
Generated Simulation Language	VHDL

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.

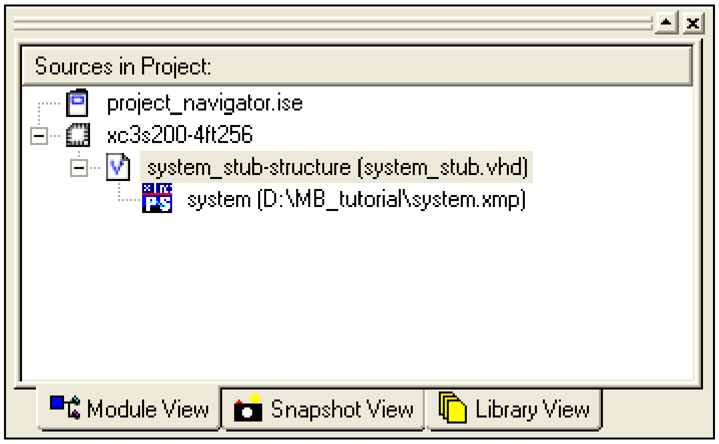


Figure 17: Project Navigator Project Hierarchy

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 vhdl 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

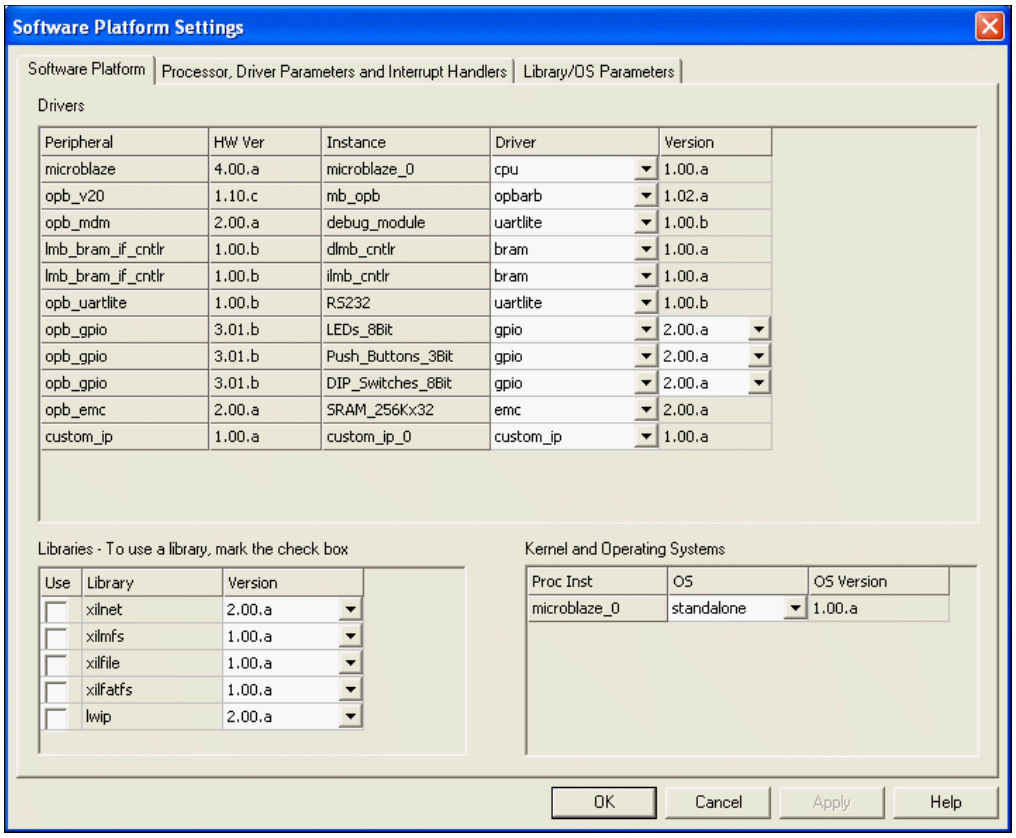
Now that the hardware design is completed, the next step is defining the software design. There are two major parts to software design, configuring the Board Support Package (BSP) and writing the software applications. The configuration of the BSP includes the selection of device drivers and libraries.

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.


Software Platform Settings
Software Platform
Processor, Driver Parameters and Interrupt Handlers
Library/OS Parameters
OK
Cancel
Apply
Help

Drivers

Peripheral	HW Ver	Instance	Driver	Version
microblaze	4.00.a	microblaze_0	cpu	1.00.a
opb_v20	1.10.c	mb_opb	opbarb	1.02.a
opb_mdm	2.00.a	debug_module	uartlite	1.00.b
lmb_bram_if_cntlr	1.00.b	dlmb_cntlr	bram	1.00.a
lmb_bram_if_cntlr	1.00.b	ilmb_cntlr	bram	1.00.a
opb_uartlite	1.00.b	RS232	uartlite	1.00.b
opb_gpio	3.01.b	LEDs_8Bit	gpio	2.00.a
opb_gpio	3.01.b	Push_Buttons_3Bit	gpio	2.00.a
opb_gpio	3.01.b	DIP_Switches_8Bit	gpio	2.00.a
opb_emc	2.00.a	SRAM_256Kx32	emc	2.00.a
custom_ip	1.00.a	custom_ip_0	custom_ip	1.00.a

Libraries - To use a library, mark the check box

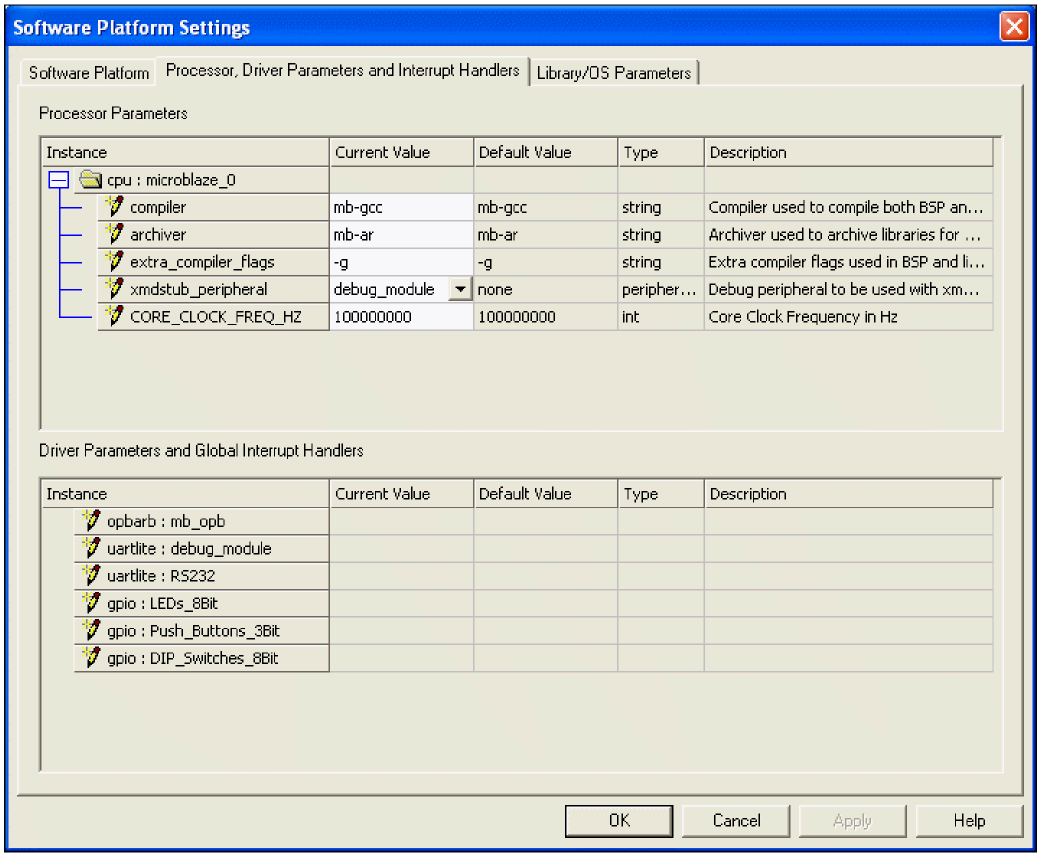
Use	Library	Version
☐	xilnet	2.00.a
☐	xilmfs	1.00.a
☐	xilfile	1.00.a
☐	xilfatfs	1.00.a
☐	lwip	2.00.a

Kernel and Operating Systems

Proc Inst	OS	OS Version
microblaze_0	standalone	1.00.a

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.



Software Platform Settings

Software Platform | Processor, Driver Parameters and Interrupt Handlers | Library/OS Parameters

Processor Parameters

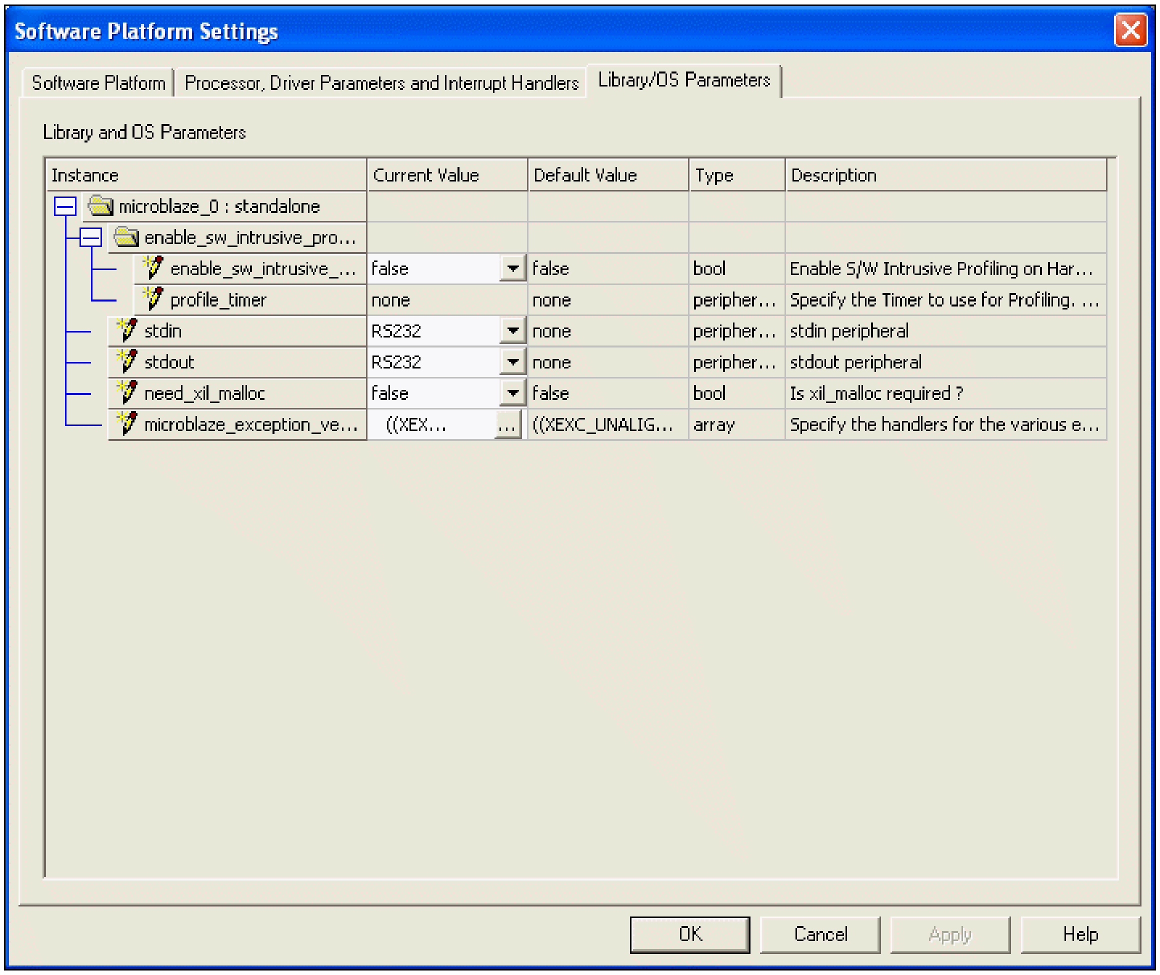
Instance	Current Value	Default Value	Type	Description
cpu : microblaze_0				
compiler	mb-gcc	mb-gcc	string	Compiler used to compile both BSP an...
archiver	mb-ar	mb-ar	string	Archiver used to archive libraries for ...
extra_compiler_flags	-g	-g	string	Extra compiler flags used in BSP and li...
xmdstub_peripheral	debug_module	none	peripher...	Debug peripheral to be used with xm...
CORE_CLOCK_FREQ_HZ	100000000	100000000	int	Core Clock Frequency in Hz

Driver Parameters and Global Interrupt Handlers

Instance	Current Value	Default Value	Type	Description
opbarb : mb_opb				
uartlite : debug_module				
uartlite : RS232				
gpio : LEDs_8Bit				
gpio : Push_Buttons_3Bit				
gpio : DIP_Switches_8Bit				

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.


Software Platform Settings

Software Platform | Processor, Driver Parameters and Interrupt Handlers | Library/OS Parameters

Library and OS Parameters

Instance	Current Value	Default Value	Type	Description
microblaze_0 : standalone				
enable_sw_intrusive_pro...				
enable_sw_intrusive_...	false	false	bool	Enable S/W Intrusive Profiling on Har...
profile_timer	none	none	peripher...	Specify the Timer to use for Profiling. ...
stdin	RS232	none	peripher...	stdin peripheral
stdout	RS232	none	peripher...	stdout peripheral
need_xil_malloc	false	false	bool	Is xil_malloc required ?
microblaze_exception_ve...	((XEX...	((XEXC_UNALIG...	array	Specify the handlers for the various e...

Figure 20: **Library/OS Parameters**

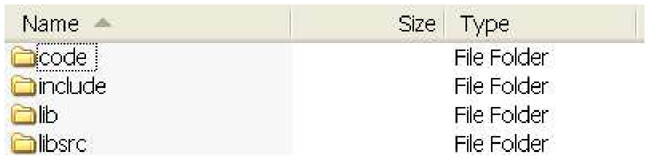
Click ***OK***.

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 ***microblaze_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

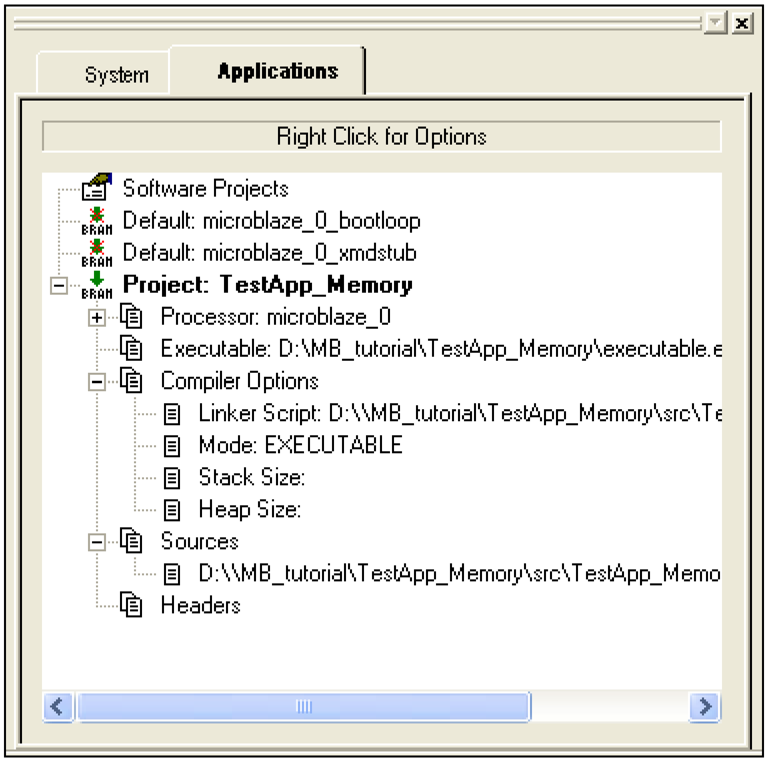
Figure 21: MicroBlaze Drivers Directories

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 downloaded
- 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.



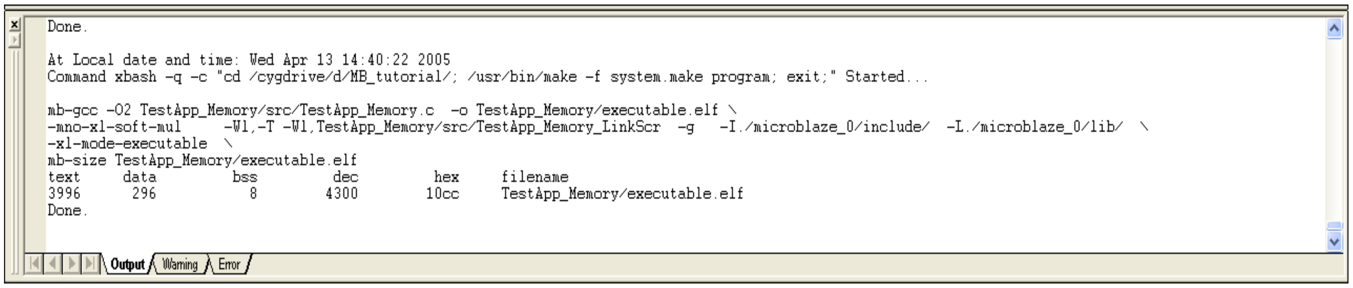
```
Software Projects
Default: microblaze_0_bootloop
Default: microblaze_0_xmdstub
Project: TestApp_Memory
    Processor: microblaze_0
    Executable: D:\MB_tutorial\TestApp_Memory\executable.e
    Compiler Options
        Linker Script: D:\\MB_tutorial\TestApp_Memory\src\Te
        Mode: EXECUTABLE
        Stack Size:
        Heap Size:
    Sources
        D:\\MB_tutorial\TestApp_Memory\src\TestApp_Memo
    Headers
```

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 mb-gcc. Mb-gcc compiles the source files.



```
Done.

At Local date and time: Wed Apr 13 14:40:22 2005
Command xbash -q -c "cd /cygdrive/d/MB_tutorial/; /usr/bin/make -f system.make program; exit;" Started...

mb-gcc -O2 TestApp_Memory/src/TestApp_Memory.c  -o TestApp_Memory/executable.elf \
-mno-xl-soft-mul     -Wl,-T -Wl,TestApp_Memory/src/TestApp_Memory_LinkScr  -g   -I./microblaze_0/include/  -L./microblaze_0/lib/  \
-xl-mode-executable  \
mb-size TestApp_Memory/executable.elf
text      data        bss         dec         hex      filename
3996      296           8        4300        10cc      TestApp_Memory/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 FGPA:

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

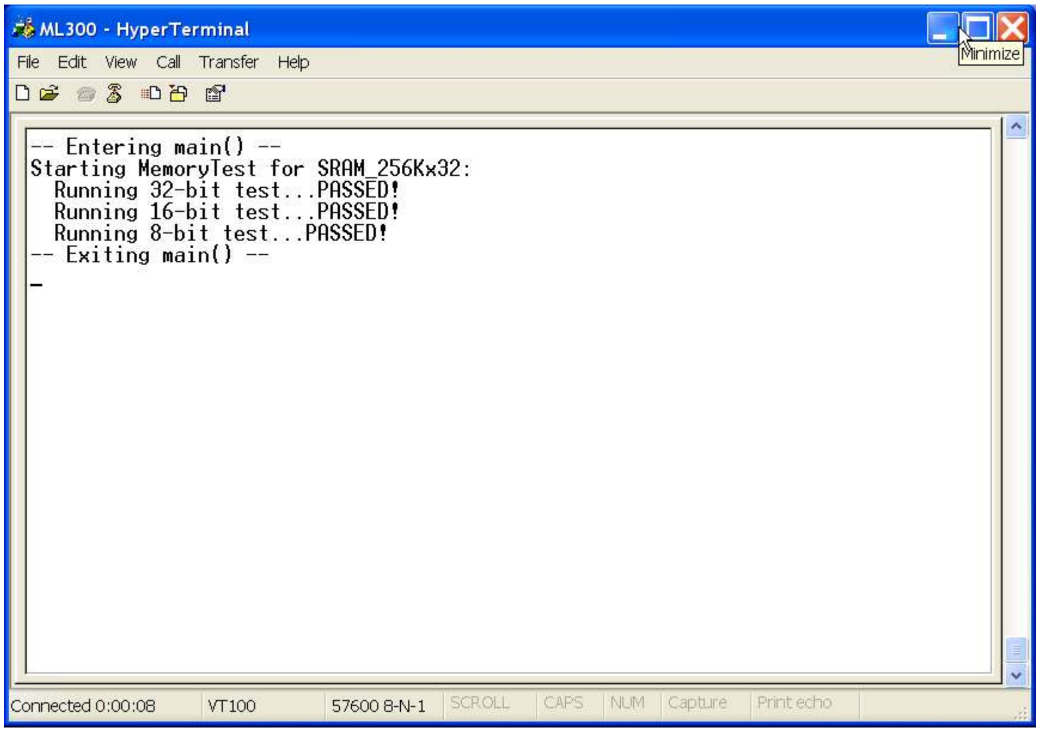
Connect the board power.

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:



```
-- 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 MDM interface. GDB connects to the MicroBlaze core through the MDM 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:

In XPS, select Options → XMD Debug Options

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

Simulator – enables XMD to connect to the MicroBlaze ISS.

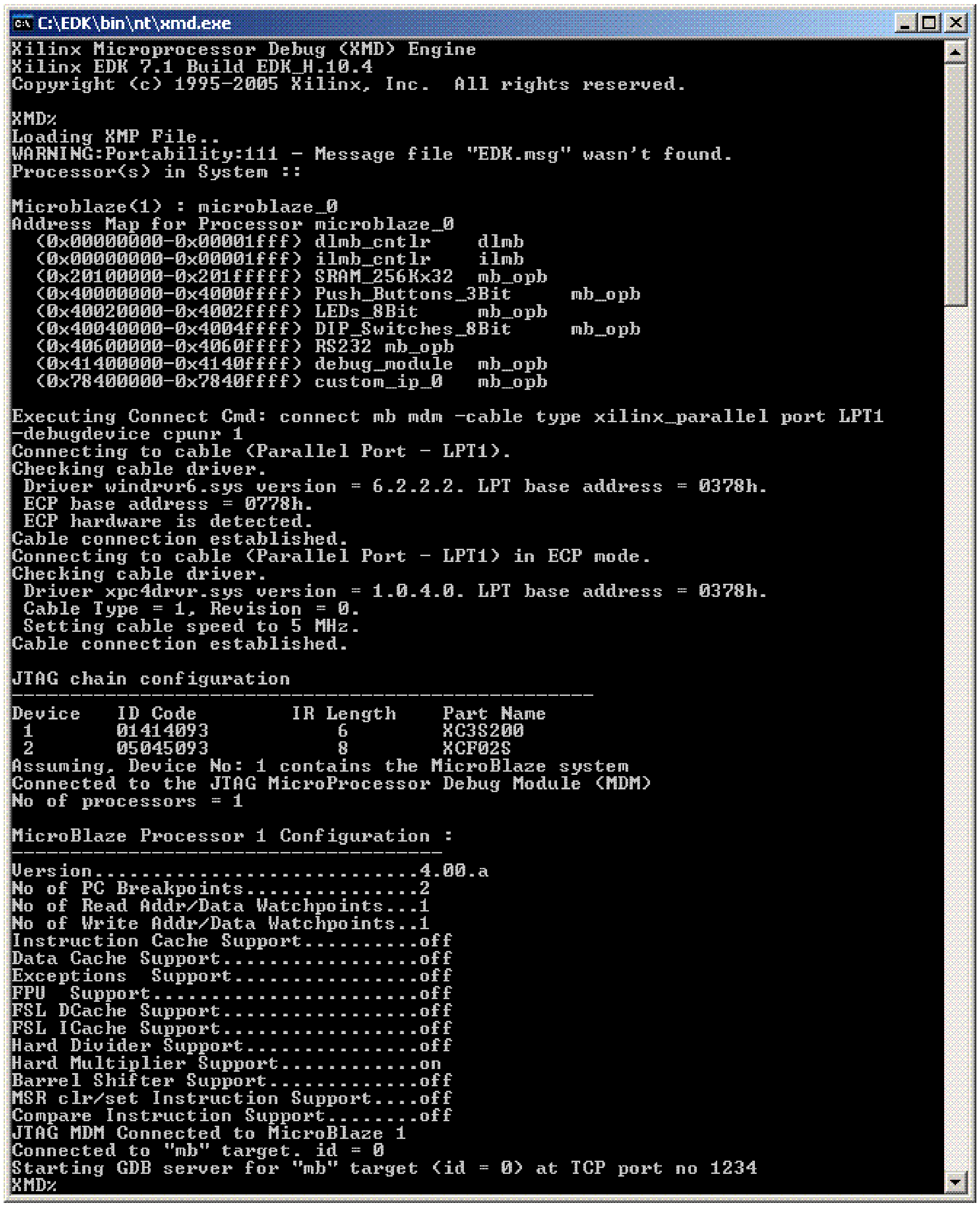
Hardware – enables XMD to connect to the MDM peripheral in the hardware.

Stub – enables XMD to connect to the JTAG UART or UART via XMDSTUB.

Verify that Hardware is selected.

Select Save.

Select Tools → XMD.

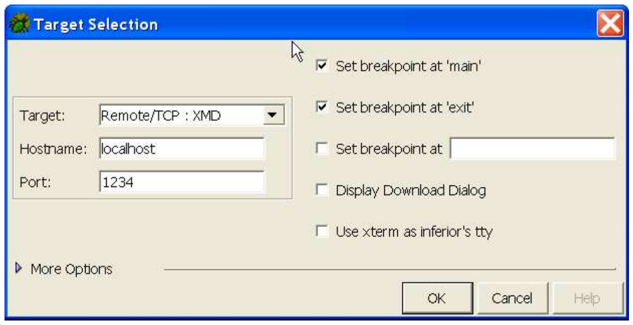


```
C:\EDK\bin\nt\xmd.exe
Xilinx Microprocessor Debug (XMD) Engine
Xilinx EDK 7.1 Build EDK_H.10.4
Copyright (c) 1995-2005 Xilinx, Inc.  All rights reserved.

XMD%
Loading XMP File..
WARNING:Portability:111 - Message file "EDK.msg" wasn't found.
Processor(s) in System ::

Microblaze(1) : microblaze_0
Address Map for Processor microblaze_0
  (0x00000000-0x00001fff) dlmb_cntlr    dlmb
  (0x00000000-0x00001fff) ilmb_cntlr    ilmb
  (0x20100000-0x201fffff) SRAM_256Kx32  mb_opb
  (0x40000000-0x4000ffff) Push_Buttons_3Bit     mb_opb
  (0x40020000-0x4002ffff) LEDs_8Bit     mb_opb
  (0x40040000-0x4004ffff) DIP_Switches_8Bit     mb_opb
  (0x40600000-0x4060ffff) RS232 mb_opb
  (0x41400000-0x4140ffff) debug_module  mb_opb
  (0x78400000-0x7840ffff) custom_ip_0   mb_opb

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 = 6.2.2.2. 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 xpc4drvr.sys version = 1.0.4.0. LPT base address = 0378h.
 Cable Type = 1, Revision = 0.
 Setting cable speed to 5 MHz.
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 MicroProcessor Debug Module (MDM)
No of processors = 1

MicroBlaze Processor 1 Configuration :
-------------------------------------
Version............................4.00.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
Exceptions  Support................off
FPU  Support.......................off
FSL DCache Support.................off
FSL ICache Support.................off
Hard Divider Support...............off
Hard Multiplier Support............on
Barrel Shifter Support.............off
MSR clr/set Instruction Support....off
Compare Instruction 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%
```

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***.



Target Selection

- [✓] Set breakpoint at 'main'
- [✓] Set breakpoint at 'exit'
- [ ] Set breakpoint at
- [ ] Display Download Dialog
- [ ] Use xterm as inferior's tty

Target: Remote/TCP : XMD
Hostname: localhost
Port: 1234

Figure 26: **GDB Target Selection**

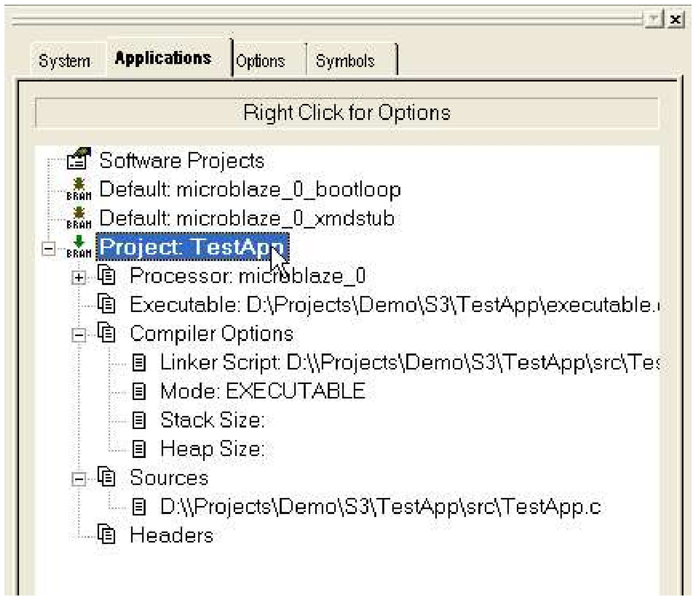
In GDB, select ***File → Open…***

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.

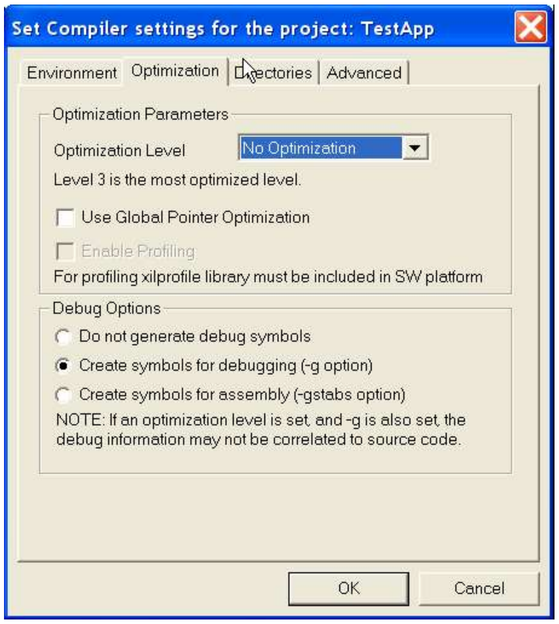


Software Projects
Default: microblaze_0_bootloop
Default: microblaze_0_xmdstub
Project: TestApp
Processor: microblaze_0
Executable: D:\Projects\Demo\S3\TestApp\executable.
Compiler Options
Linker Script: D:\\Projects\Demo\S3\TestApp\src\Tes
Mode: EXECUTABLE
Stack Size:
Heap Size:
Sources
D:\\Projects\Demo\S3\TestApp\src\TestApp.c
Headers

In the Optimization tab;

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

Click ***OK***.



Optimization Parameters

Optimization Level: No Optimization

Level 3 is the most optimized level.

☐ Use Global Pointer Optimization

☐ Enable Profiling

For profiling xilprofile library must be included in SW platform

Debug Options

○ Do not generate debug symbols

● Create symbols for debugging (-g option)

○ Create symbols for assembly (-gstabs option)

NOTE: If an optimization level is set, and -g is also set, the debug information may not be correlated to source code.

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.

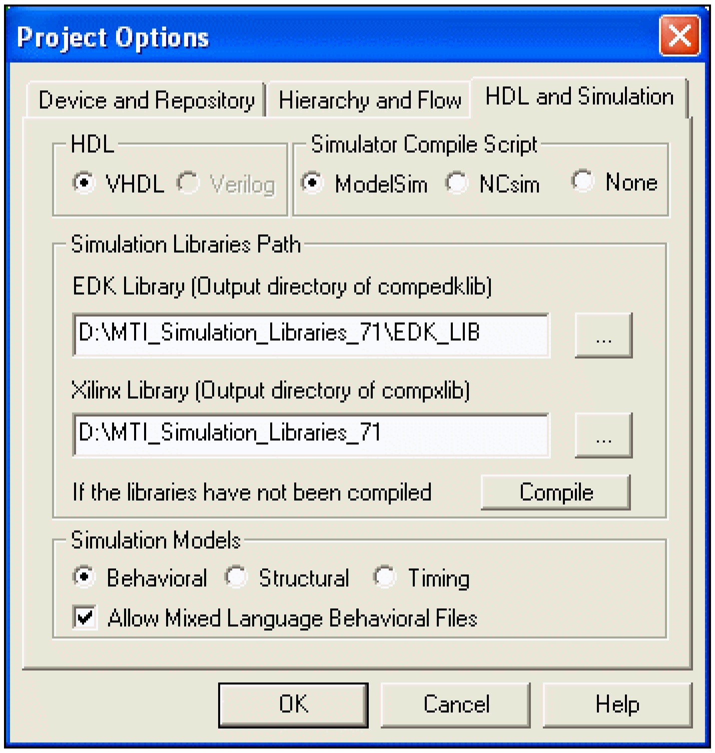
Note: ***The default values displayed in the Registers Window are in hex, while the values displayed in the Source Window are in decimal.***

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 you system. For additional information on compiling the simulation libraries refer to the Platform Studio User Guide Chapter 8.



HDL: [● VHDL] ○ Verilog

Simulator Compile Script: [● ModelSim] ○ NCsim ○ None

Simulation Libraries Path

EDK Library (Output directory of compedklib)

D:\MTI_Simulation_Libraries_71\EDK_LIB

Xilinx Library (Output directory of compxlib)

D:\MTI_Simulation_Libraries_71

If the libraries have not been compiled

Simulation Models: [● Behavioral] ○ Structural ○ Timing

[✓] Allow Mixed Language Behavioral Files

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

fpga_0_RS232_RX_pin <= fpga_0_RS232_TX_pin;
```

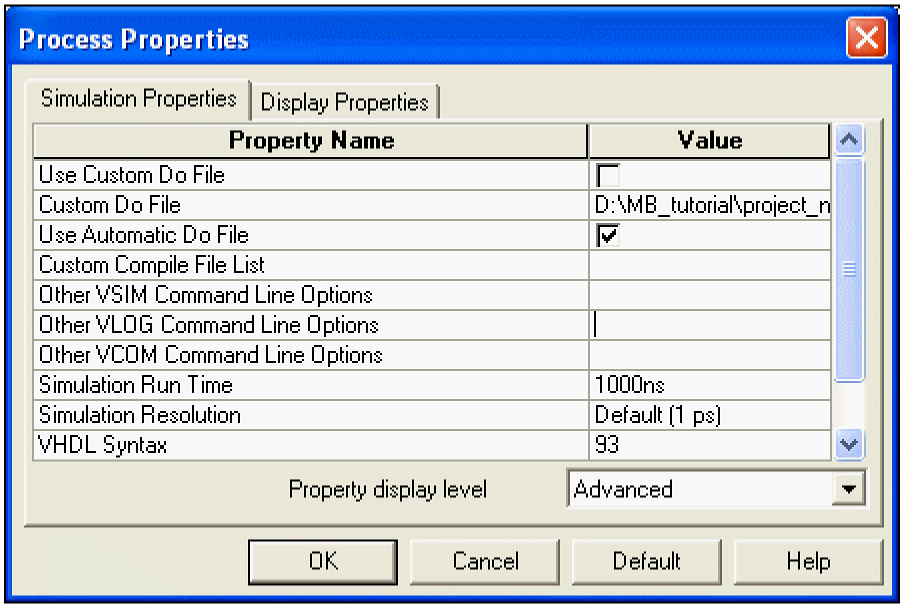
In order to populate the BRAMs with the TestApp_Memory Application, a configuration statement must be utilized. Add the following after the final “END;” statement in the Testbench.vhd file:

```
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



Property Name	Value
Use Custom Do File	☐
Custom Do File	D:\MB_tutorial\project_n
Use Automatic Do File	✓
Custom Compile File List	
Other VSIM Command Line Options	
Other VLOG Command Line Options	
Other VCOM Command Line Options	
Simulation Run Time	1000ns
Simulation Resolution	Default (1 ps)
VHDL Syntax	93

Property display level: Advanced

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.