MicroBlaze Tutorial on EDK 10.1 using Spartan 3e

Ahmed Elhossini

January 24, 2010

1 Introduction

The Embedded Development Kit (EDK) from Xilinx allows the designer to build a complete processor system on Xilinx's FPGAs. The systems that can be produced using EDK ranges from a simple single processor architecture to a complex multi-processor system with multiple hardware accelerators. The tool mainly supports two types of processors: i) MicroBlaze which is a reconfigurable soft-core processor and ii) Power-PC which is a hardcore processor implemented in some FPGAs from Xilinx. Depending on the FPGA chip you are using, multiple Microblazes and Power-PCs can be integrated together in a single design. EDK provides C/C++ compilers for both MicroBlaze and Power-PC along with several tools for debugging/profiling of the applications running on each processor. Besides, using ISE, you can perform several types of simulations for the generated architectures which allows the estimation of both the performance and power consumption of the architecture. This tutorial will demonstrate the process of creating and testing a MicroBlaze system design using the Embedded Development Kit (EDK) and Spartan 3e starter board from Xilinx.

1.1 Objectives

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 or Import IP Peripheral
- Design Modification using Platform Studio
- Implementing the Design
- Defining the Software Design
- Downloading the Design

1.2 System Requirements

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

- Windows 2000 SP2/Windows XP
- EDK 10.1i.
- ISE 10.1i.
- Familiarity with Xilinx ISE 10.1 design flow.
- Spartan 3e starter kit and Xilinx USB download 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 if using a different platform:

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

1.3 MicroBlaze System Description

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

- Hardware components
- Memory map
- Software application

1.3.1 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
- PLB Bus
- Multi-Port Memory Controller (MPMC)
- MDM
- XPS_UARTLITE
- 2 - XPS_GPIOs

1.3.2 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_3FFF	16K bytes	LMB Memory
MDM	0x8440_0000	0x8440_FFFF	64K bytes	debug_module
XPS_UARTLITE	0x8400_0000	0x8400_FFFF	64K bytes	RS232_DCE
XPS_GPIO	0x8140_0000	0x8140_FFFF	64K bytes	LEDs_8Bit
XPS_GPIO	0x8142_0000	0x8142_FFFF	64K bytes	DIP_Switches_4Bit
MPMC	0x8600_0000	0x87FF_FFFF	32M bytes	DDR_SDRAM

Table 1: Table 1: Tutorial Design Memory Map

1.4 Tutorial Steps

1.4.1 Setup

Spartan 3e starter 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.

1.4.2 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:

(i) A Project Navigator project file that allows you to control the hardware implementation flow

(ii) A Microprocessor Hardware Specification (MHS) file

Note: For more information on the MHS file, refer to the "Microprocessor Hardware Specification (MHS)" chapter in the Platform Specification Format Reference Manual.

(iii) Microprocessor Software Specification (MSS) file

Note: For more information on the MSS file, refer to the "Microprocessor Software Specification (MSS)" chapter in the Platform Specification Format Reference Manual.

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

2 Starting XPS

(i) To open XPS, select Start → All Programs → Development → Xilinx ISE Design Suite 10.1 → EDK → Xilinx Platform Studio

(ii) Select Base System Builder Wizard (BSB) to open the **"Create New XPS Project Using BSB Wizard"** dialogue box shown in Figure 1.

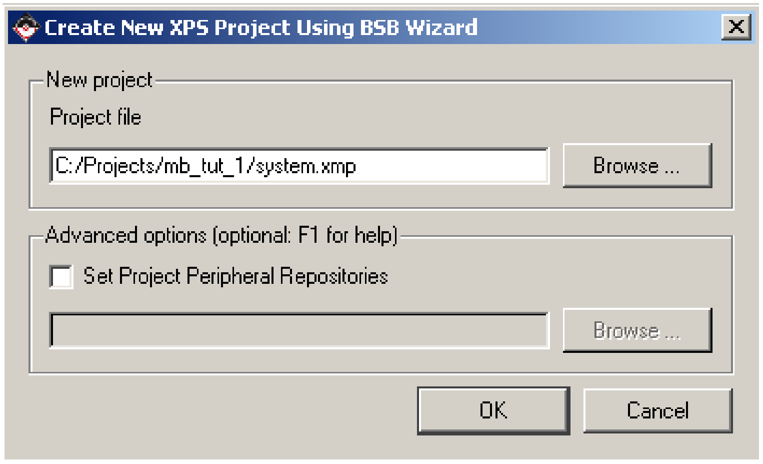
(iii) Click Ok.

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

(v) Click Open to create the system.xmp file then Save.

(vi) Click Ok to start the BSB wizard. The wizard window will appear, which will be used to build the design as will be discussed in following sections.

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


Create New XPS Project Using BSB Wizard

New project

Project file: C:/Projects/mb_tut_1/system.xmp

Advanced options (optional: F1 for help)

☐ Set Project Peripheral Repositories

Figure 1: Create New XPS Project Using Base System Builder Wizard

3 Defining the System Hardware

3.1 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.

3.1.1 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 and Virtex-4 FX 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.

3.1.2 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.

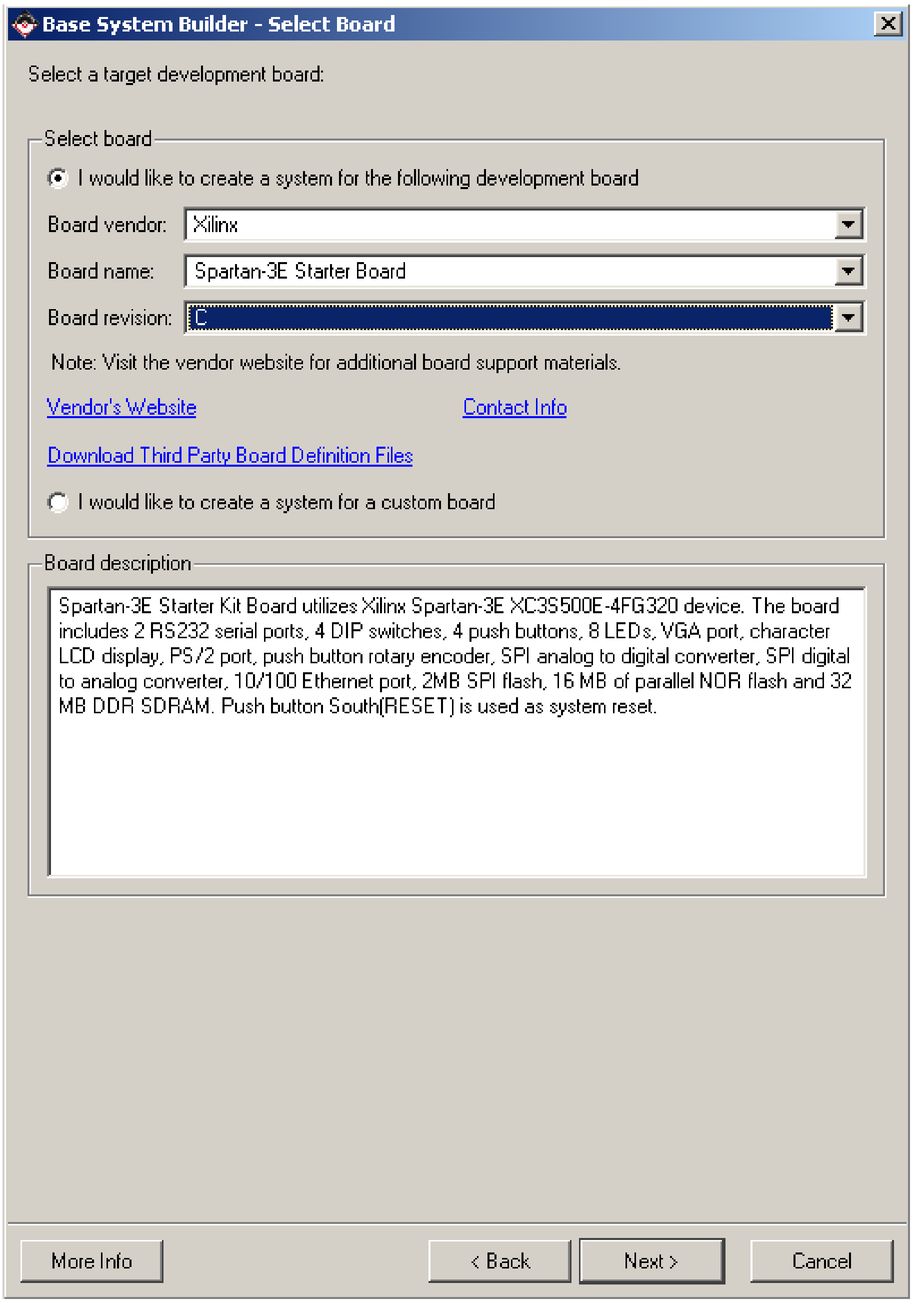
4 Using the Base System Builder Wizard

Use the following steps to create the processor system:

- In the Base System Builder - **Select** "I would like to create a new design" then click **Next.**
- In the Base System Builder - **Select** Board Dialog select the following, as shown in Figure 2:
  - Board Vendor: Xilinx
  - Board Name: Spartan-3E Starter Board
  - Board Revision: C
- Click **Next.** Select the MicroBlaze
- Click **Next.** You will now specify several processor options as shown in Figure 3:

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

- System Wide Setting:
  - Reference clock frequency: This is the on board frequency of the clock.
  - Processor-Bus clock frequency: This is the frequency of the clock driving the processor system.
- Processor Configuration:
  - Debug I/F:
    * On-Chip H/W Debug module: When the H/W debug module is selected; a PLB 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.
    * 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 PLB MDM module.


Base System Builder - Select Board

Select a target development board:

Select board

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

Board vendor: Xilinx

Board name: Spartan-3E Starter Board

Board revision: C

Note: Visit the vendor website for additional board support materials.

Vendor's Website Contact Info

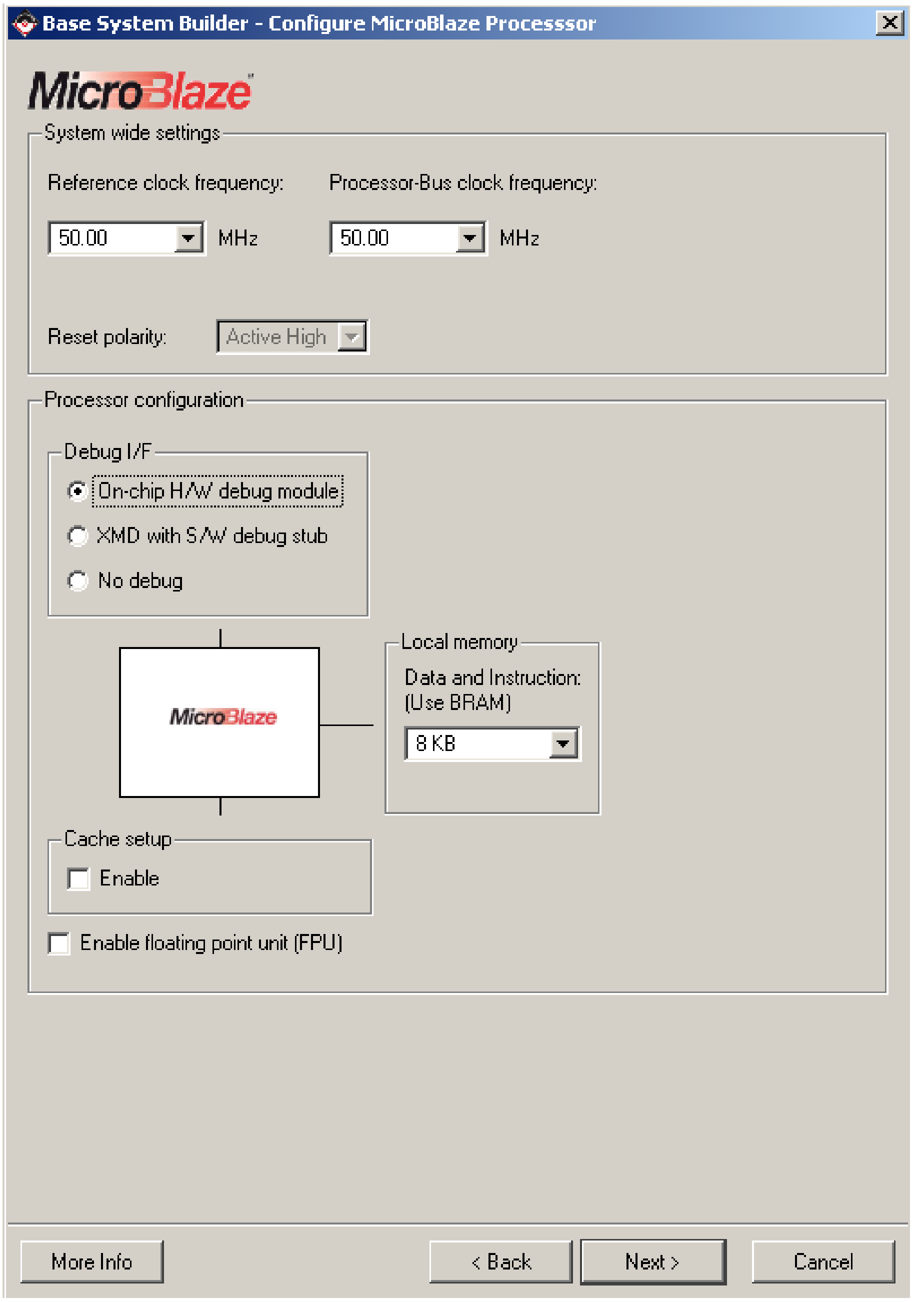
Download Third Party Board Definition Files

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

Board description

Spartan-3E Starter Kit Board utilizes Xilinx Spartan-3E XC3S500E-4FG320 device. The board includes 2 RS232 serial ports, 4 DIP switches, 4 push buttons, 8 LEDs, VGA port, character LCD display, PS/2 port, push button rotary encoder, SPI analog to digital converter, SPI digital to analog converter, 10/100 Ethernet port, 2MB SPI flash, 16 MB of parallel NOR flash and 32 MB DDR SDRAM. Push button South(RESET) is used as system reset.

Figure 2: BSB: Select a Board


Base System Builder - Configure MicroBlaze Processsor

System wide settings

Reference clock frequency: 50.00 MHz

Processor-Bus clock frequency: 50.00 MHz

Reset polarity: Active High

Processor configuration

Debug I/F

(●) On-chip H/W debug module

( ) XMD with S/W debug stub

( ) No debug

Local memory

Data and Instruction: (Use BRAM)

8 KB

Cache setup

[ ] Enable

[ ] Enable floating point unit (FPU)

Figure 3: Configure Processor

    - No Debug: debugging is disabled.

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

  - Users can specify the size of the local instruction and data memory.
  - Cache setup:
    - No Cache: No caching will be used
    - Enable cache link: Caching will be used through the FSL bus
  - You can also specify the use of the floating point unit (FPU).
  - Click **Next.**

Select the peripheral subset (Configure IO Interfaces wizard) as shown in Figure 4, Figure 5 and Figure 6. It should be noted that the number of peripheral shown on each dialogue box is dynamic based upon your computers resolution.

In the first page of the **"Configure IO Interfaces wizard"**, Figure 4:

- RS232_DTE → deselect
- RS232_DCE → select
- XPS UARTLITE baud-rate → 57600, data bits → 8 and Parity → NONE
- LEDs_8Bit → select
- Click **Next**

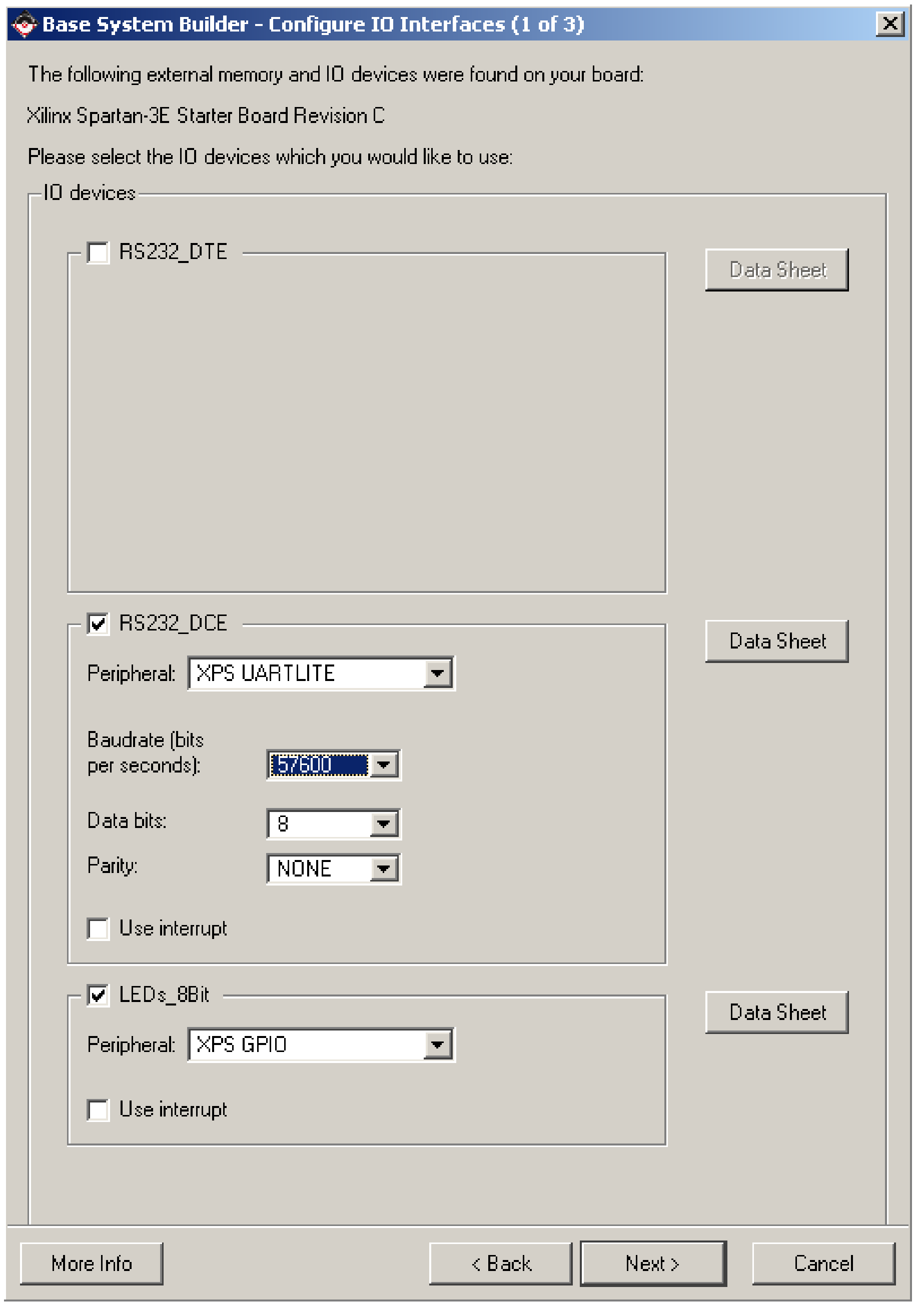
In the second page of the **"Configure IO Interfaces wizard"**, Figure 5:

- DIP_Switch_4Bit → select
- Buttons_4Bit → deselect
- FLASH → deselect
- SPI_FLASH → deselect
- Click **Next**

In the third page of the **"Configure IO Interfaces wizard"**, Figure 6:

- DDR_SDRAM → select
- Ethernet_Mac → **deselect**
- Click **Next** through the Add Internal Peripherals page as we will not add any in this tutorial.
- Click **Next**

This completes the hardware specification and we will now configure the software settings. Using the Software Setup dialogue box as shown in Figure 7, specify the following software settings:


Base System Builder - Configure IO Interfaces (1 of 3)

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

Xilinx Spartan-3E Starter Board Revision C

Please select the IO devices which you would like to use:

IO devices

☐ RS232_DTE

☑ RS232_DCE

Peripheral: XPS UARTLITE

Baudrate (bits per seconds): 57600

Data bits: 8

Parity: NONE

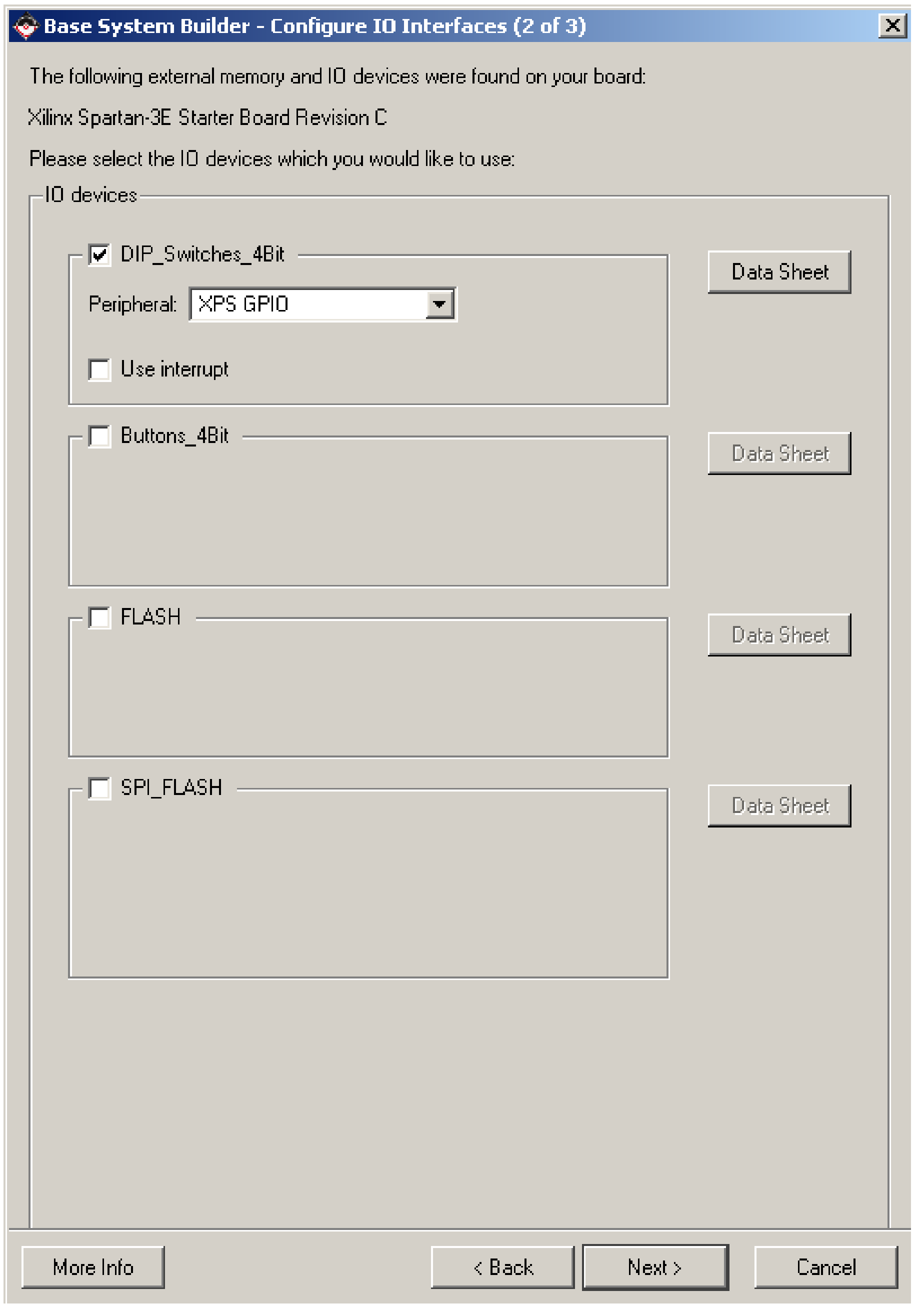
☐ Use interrupt

☑ LEDs_8Bit

Peripheral: XPS GPIO

☐ Use interrupt

Figure 4: Configure I/O Interfaces - 1


Base System Builder - Configure IO Interfaces (2 of 3)

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

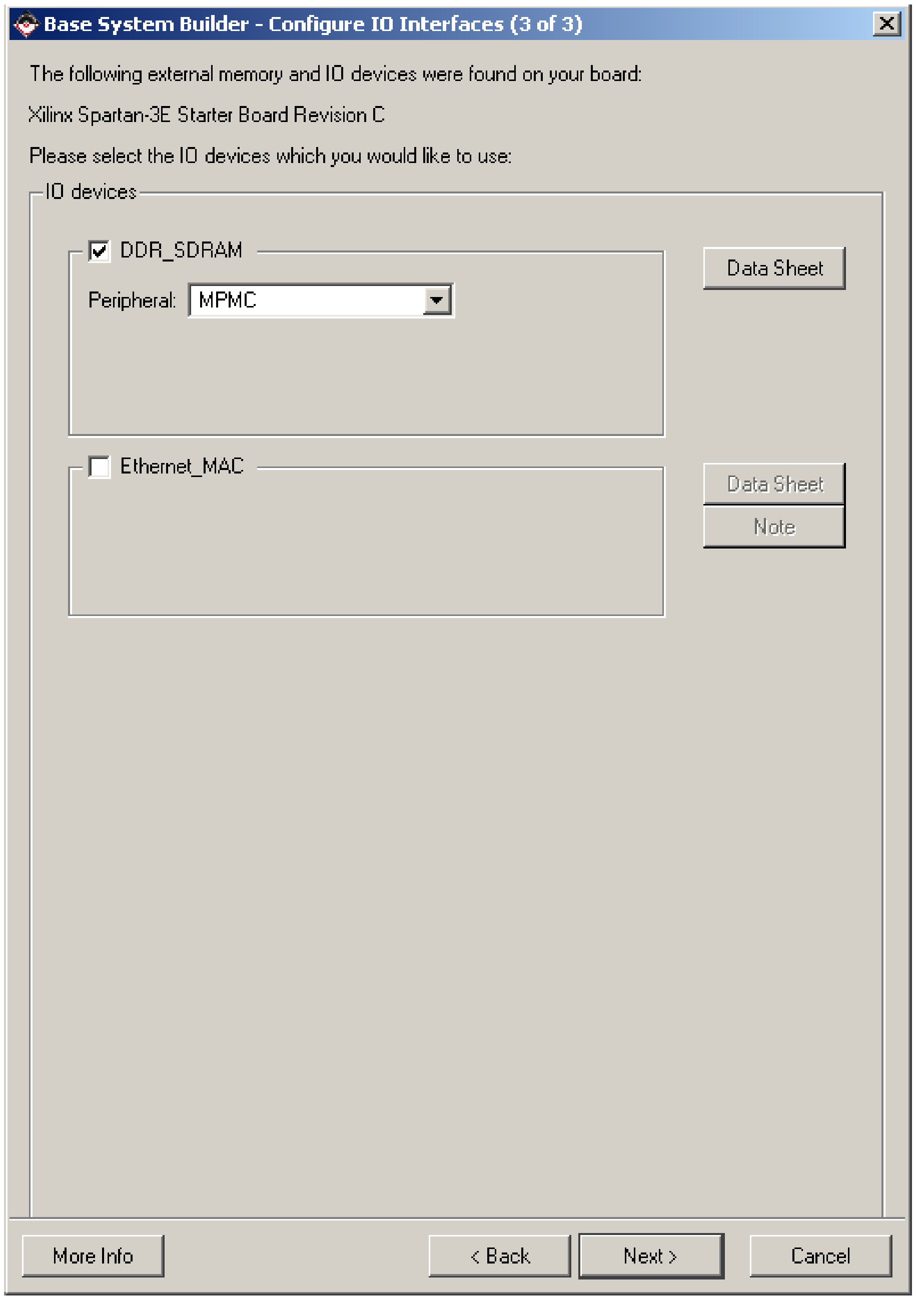
Xilinx Spartan-3E Starter Board Revision C

Please select the IO devices which you would like to use:

IO devices

- [✓] DIP_Switches_4Bit
  - Peripheral: XPS GPIO
  - [ ] Use interrupt
- [ ] Buttons_4Bit
- [ ] FLASH
- [ ] SPI_FLASH

Figure 5: Configure I/O Interfaces - 2


Base System Builder - Configure IO Interfaces (3 of 3)

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

Xilinx Spartan-3E Starter Board Revision C

Please select the IO devices which you would like to use:

IO devices

[✓] DDR_SDRAM

Peripheral: MPMC

[ ] Ethernet_MAC

Figure 6: Configure I/O Interfaces - 3

- Standard Input (STDIN) → RS232
- Standard Output (STDOUT) → RS232
- Boot Memory → *ilmb_cntlr*
- Sample Application Selection → Memory Test
- Click **Next.**

Using the Configure Memory Test Application dialogue box as shown in Figure 8, 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 9. 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.

- Click **Generate** and then **Finish**, to complete the design.
- Select **"Start Using Platform Studio"** and click **OK.**

4.1 Review

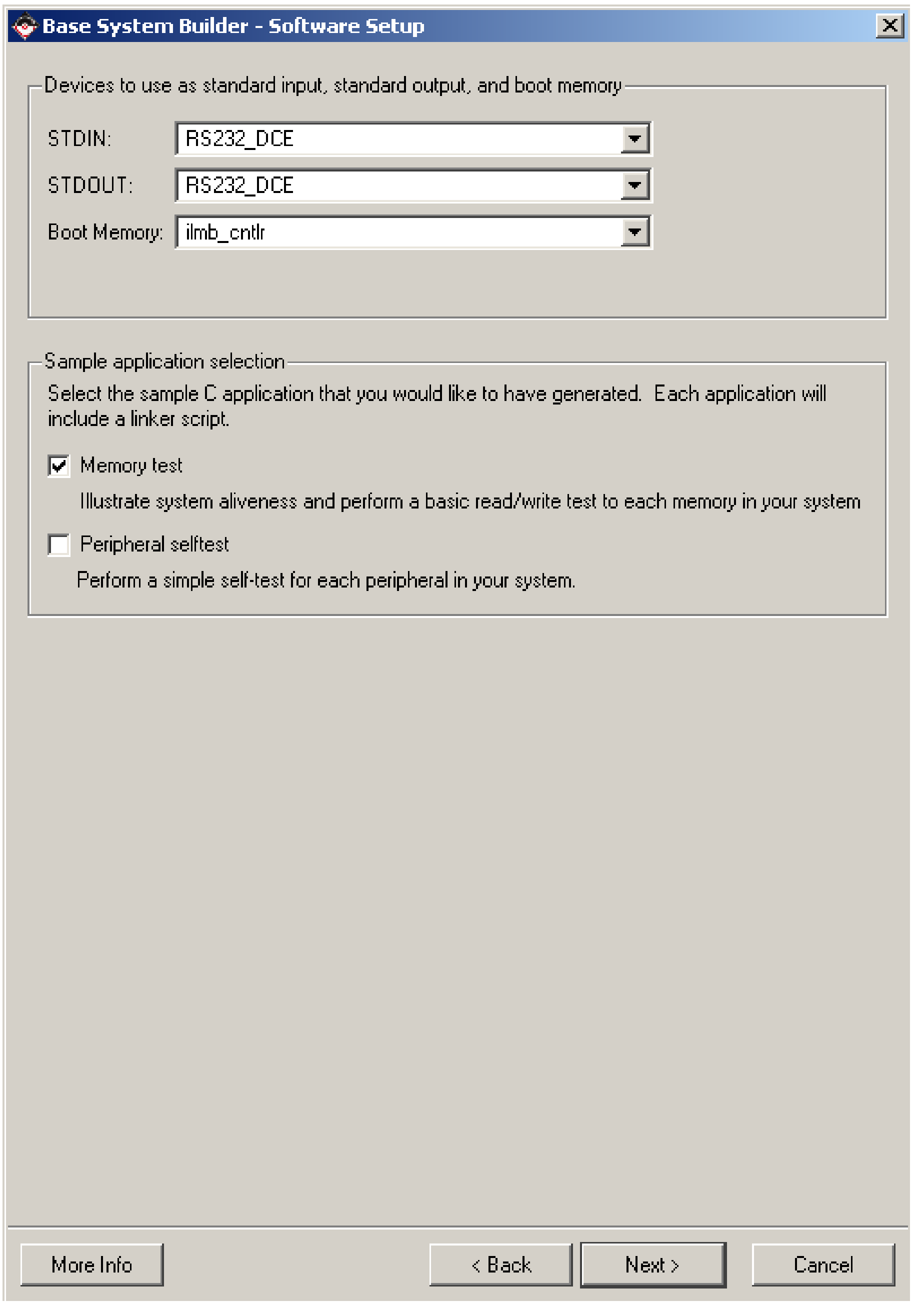
The Base System Builder Wizard has created the hardware and software specification files that define the processor system. When we look at the project directory, shown in Figure 10, we see these as system.mhs and system.mss. There are also some directories created:

- 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.

4.2 Project Options

To see the project options that Base System Builder has configured select: Project → Project Options. As shown in Figure 11, the device information is specified.

Select: Hierarchy and Flow. This window is shown in Figure 12. 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.



Devices to use as standard input, standard output, and boot memory

STDIN: RS232_DCE

STDOUT: RS232_DCE

Boot Memory: ilmb_cntlr

Sample application selection

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

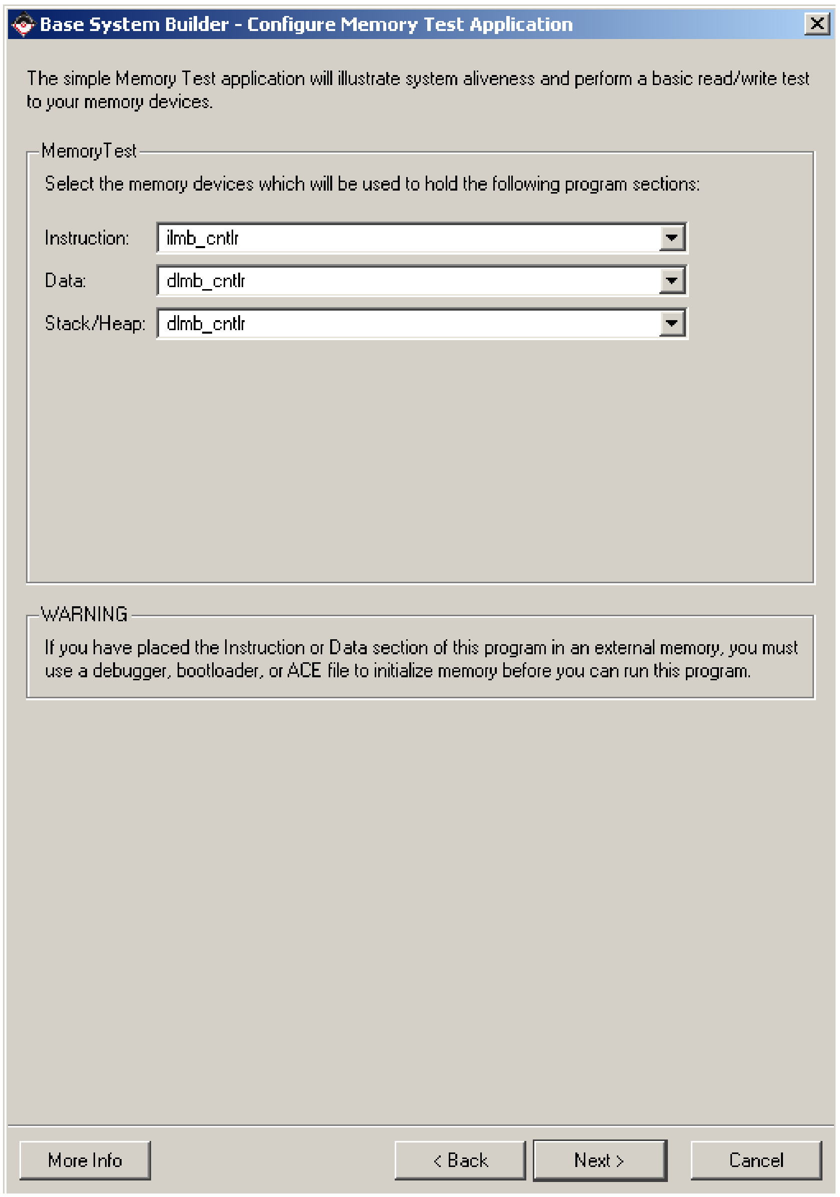
[✓] Memory test

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

[ ] Peripheral selftest

Perform a simple self-test for each peripheral in your system.

Figure 7: Software Setup


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:

Instruction: ilmb_cntlr

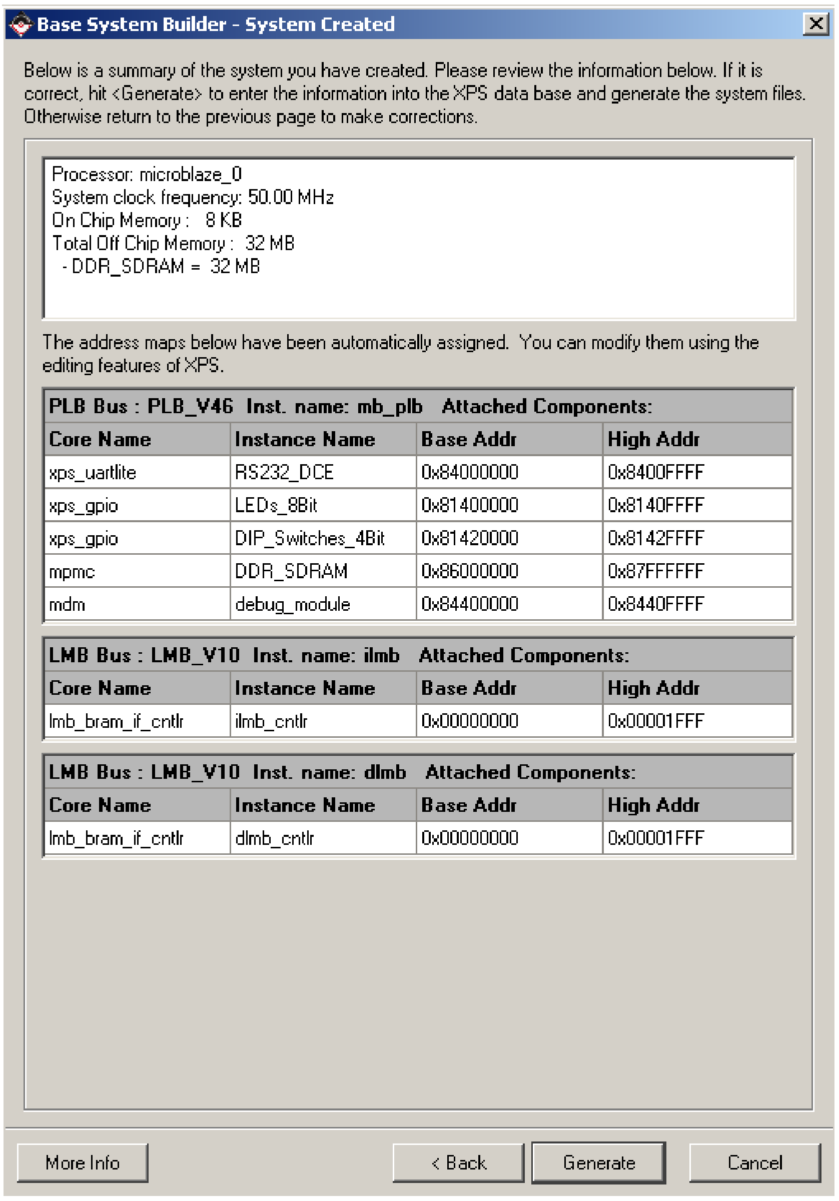
Data: dlmb_cntlr

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 8: Configure Memory Test Application



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_0
System clock frequency: 50.00 MHz
On Chip Memory :   8 KB
Total Off Chip Memory :  32 MB
  - DDR_SDRAM =  32 MB
```

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

PLB Bus : PLB_V46 Inst. name: mb_plb Attached Components:

Core Name	Instance Name	Base Addr	High Addr
xps_uartlite	RS232_DCE	0x84000000	0x8400FFFF
xps_gpio	LEDs_8Bit	0x81400000	0x8140FFFF
xps_gpio	DIP_Switches_4Bit	0x81420000	0x8142FFFF
mpmc	DDR_SDRAM	0x86000000	0x87FFFFFF
mdm	debug_module	0x84400000	0x8440FFFF

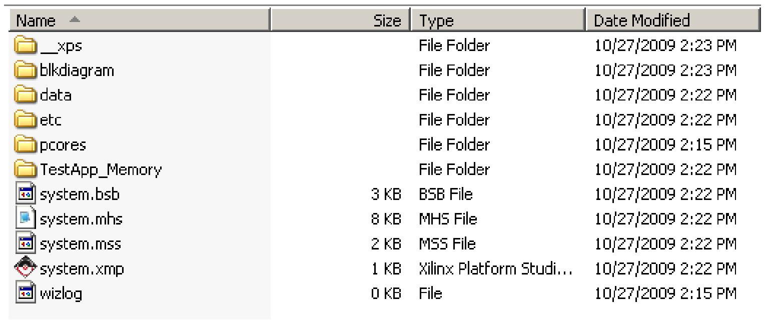
LMB Bus : LMB_V10 Inst. name: ilmb Attached Components:

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:

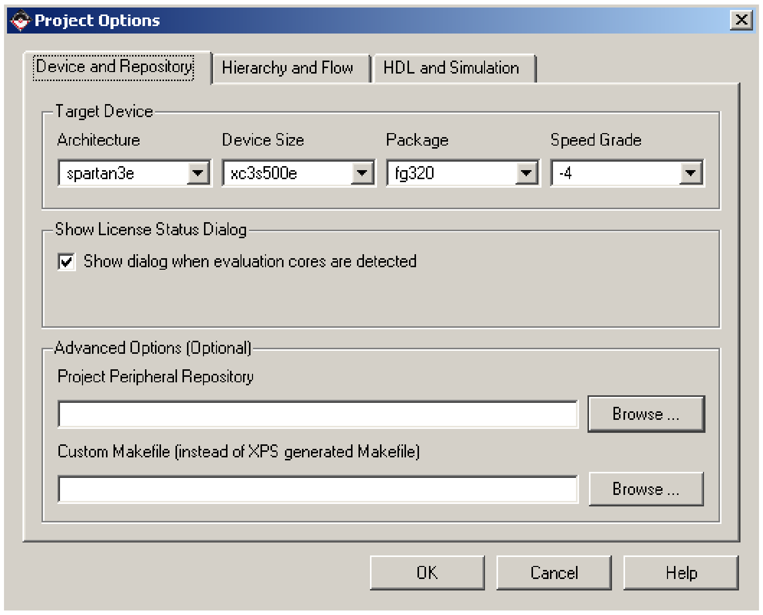
Core Name	Instance Name	Base Addr	High Addr
lmb_bram_if_cntlr	dlmb_cntlr	0x00000000	0x00001FFF

Figure 9: Completed Processor System



Name	Size	Type	Date Modified
__xps		File Folder	10/27/2009 2:23 PM
blkdiagram		File Folder	10/27/2009 2:23 PM
data		File Folder	10/27/2009 2:22 PM
etc		File Folder	10/27/2009 2:22 PM
pcores		File Folder	10/27/2009 2:15 PM
TestApp_Memory		File Folder	10/27/2009 2:22 PM
system.bsb	3 KB	BSB File	10/27/2009 2:22 PM
system.mhs	8 KB	MHS File	10/27/2009 2:22 PM
system.mss	2 KB	MSS File	10/27/2009 2:22 PM
system.xmp	1 KB	Xilinx Platform Studi...	10/27/2009 2:22 PM
wizlog	0 KB	File	10/27/2009 2:15 PM

Figure 10: Project Directory


Project Options

Device and Repository | Hierarchy and Flow | HDL and Simulation

Target Device

Architecture: spartan3e
Device Size: xc3s500e
Package: fg320
Speed Grade: -4

Show License Status Dialog

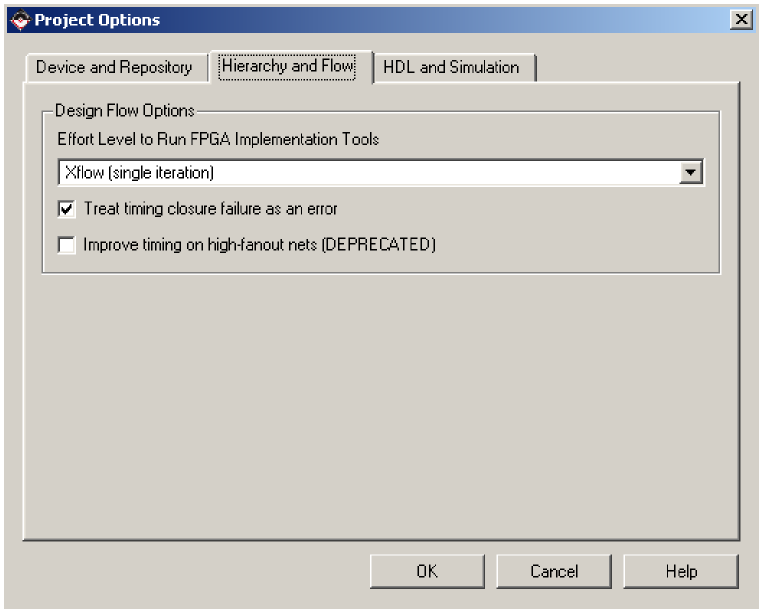
[✓] Show dialog when evaluation cores are detected

Advanced Options (Optional)

Project Peripheral Repository

Custom Makefile (instead of XPS generated Makefile)

Figure 11: Project Options - Device and Repository


Project Options

Device and Repository | Hierarchy and Flow | HDL and Simulation

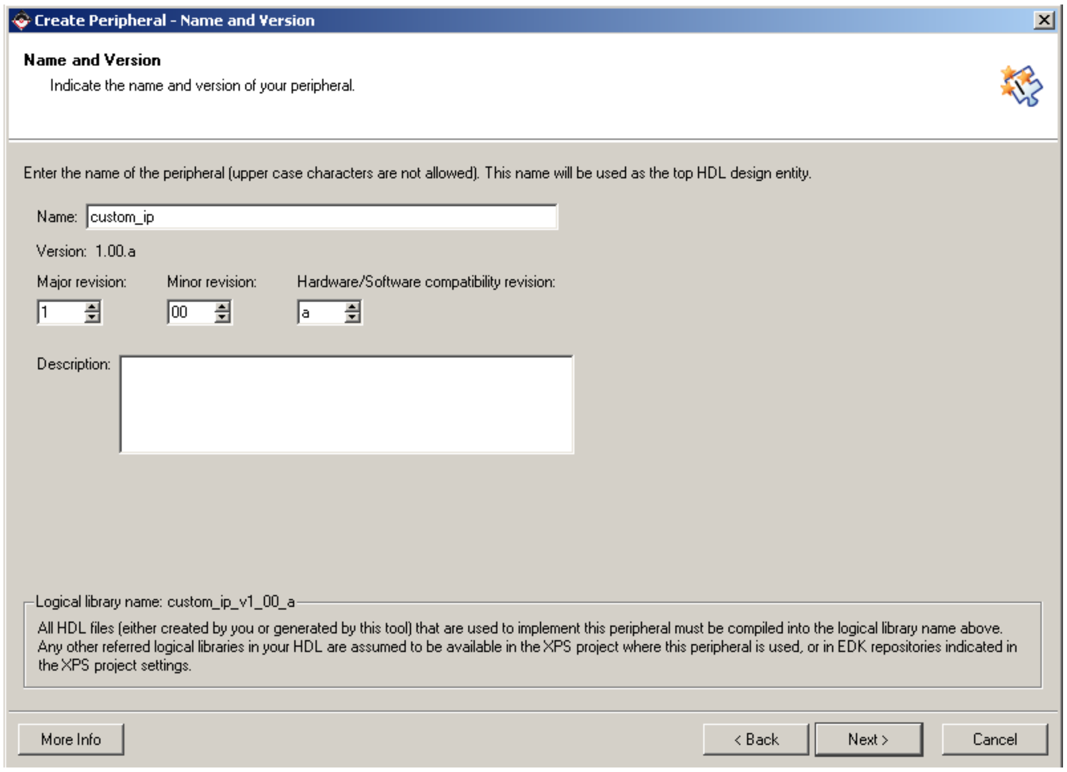
Design Flow Options

Effort Level to Run FPGA Implementation Tools: Xflow (single iteration)

[✓] Treat timing closure failure as an error

[ ] Improve timing on high-fanout nets (DEPRECATED)

Figure 12: Project Options - Hierarchy and Flow


Create Peripheral - Name and Version
Name and Version
Indicate the name and version of your peripheral.

Enter the name of the peripheral (upper case characters are not allowed). 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

Description:

Logical library name: custom_ip_v1_00_a

All HDL files (either created by you or generated by this tool) that are used to implement this peripheral must be compiled into the logical library name above. Any other referred logical libraries 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 13: Create Peripheral - Name and Version

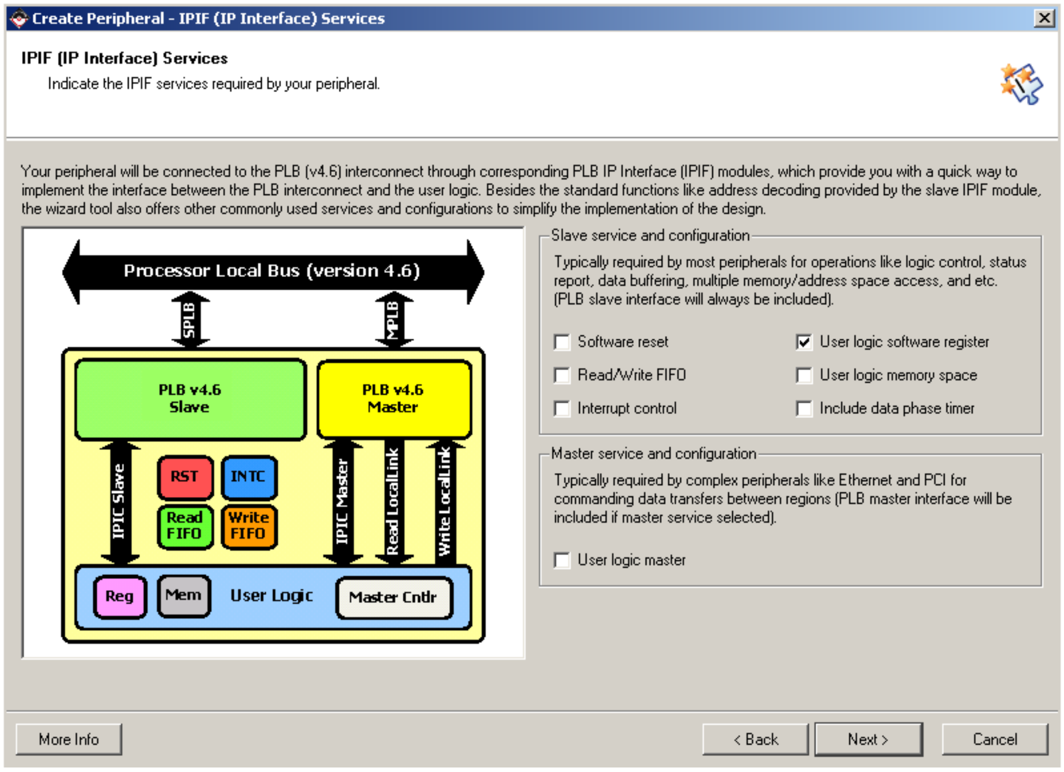
5 Create or Import IP Peripheral

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

- In XPS, select Hardware → Create or 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 - Name and Version dialogue, enter **"custom_ip"** as the name of the peripheral. This is shown in Figure 13.
- Click **Next.** In the Create Peripheral - Bus Interface dialogue. **Select** Processor Local Bus(PLB v4.6), as this is the bus to which the new peripheral will be connected.
- Click **Next.** The Create Peripheral - IPIF Services dialogue 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 as shown in Figure 14.
- Click **Next.** In Slave interface, do not change anything


Create Peripheral - IPIF (IP Interface) Services

IPIF (IP Interface) Services

Indicate the IPIF services required by your peripheral.

Your peripheral will be connected to the PLB (v4.6) interconnect through corresponding PLB IP Interface (IPIF) modules, which provide you with a quick way to implement the interface between the PLB interconnect and the user logic. Besides the standard functions like address decoding provided by the slave IPIF module, the wizard tool also offers other commonly used services and configurations to simplify the implementation of the design.

Slave service and configuration

Typically required by most peripherals for operations like logic control, status report, data buffering, multiple memory/address space access, and etc. (PLB slave interface will always be included).

- [ ] Software reset
- [✓] User logic software register
- [ ] Read/Write FIFO
- [ ] User logic memory space
- [ ] Interrupt control
- [ ] Include data phase timer

Master service and configuration

Typically required by complex peripherals like Ethernet and PCI for commanding data transfers between regions (PLB master interface will be included if master service selected).

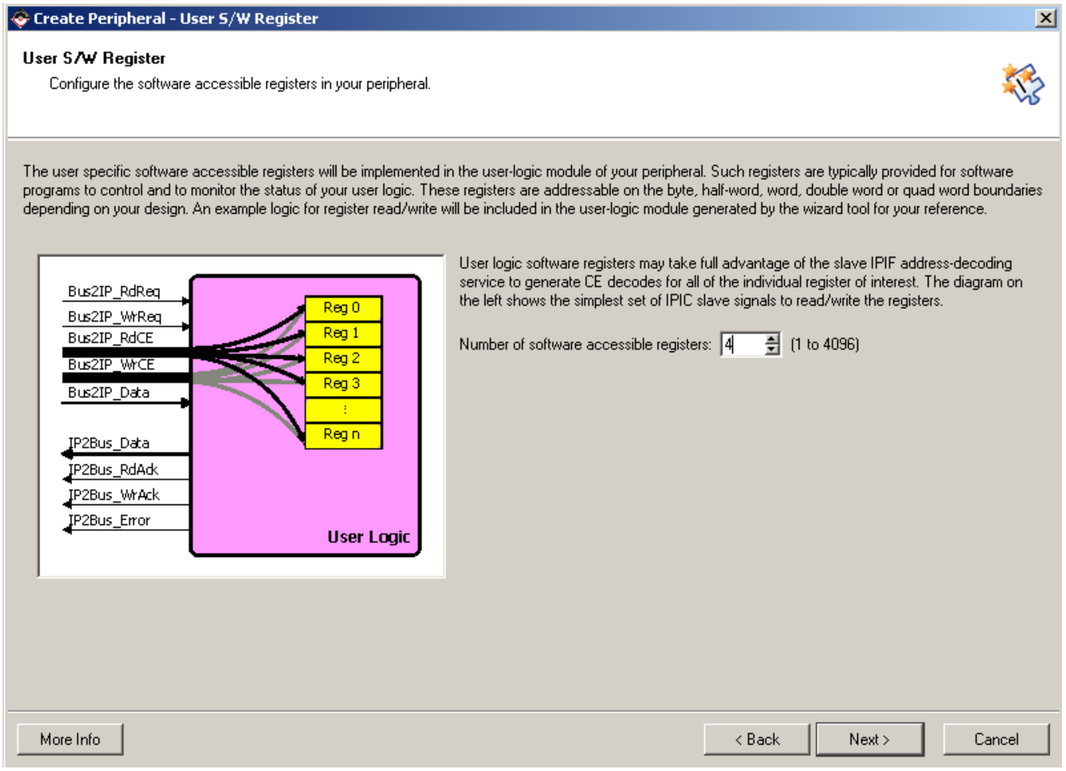
- [ ] User logic master

Figure 14: Create Peripheral - IPIF Services

- Click **Next.** In the Create Peripheral - User S/W Register dialogue, change the Number of software accessible registers to 4 as shown in Figure 15.
- Click **Next.** In the Create Peripheral - IP Interconnect (IPIC).
- Click **Next.** In the Create Peripheral - (OPTIONAL) Peripheral Simulation Support dialogue, a Bus Functional Model (BFM) simulation environment can be generated. This tutorial will not cover BFM simulation. **Leave** the option unchecked.
- Click **Next.** In the Create Peripheral - (OPTIONAL) Peripheral Implementation Support dialogue, **uncheck** the Generate ISE and XST project files to help you implement the peripheral using XST flow.
- Click **Next.**
- Click **Finish.**

The Create or Import Peripheral Wizard creates a new directory called custom_ip_v1_00_a in the pcores directory. This new directory contains **hdl source** directory, **drivers** directory and **data** directory as seen in Figure 16.
In the **hdl** directory several source files cab be identified:

- *mb_tut_1\pcores\custom_ip_v1_00_a\hdl*
  - **custom_ip.vhd**
    This is the template file for your peripheral's top design entity. It configures and instantiates


Create Peripheral - User S/W Register

User S/W Register

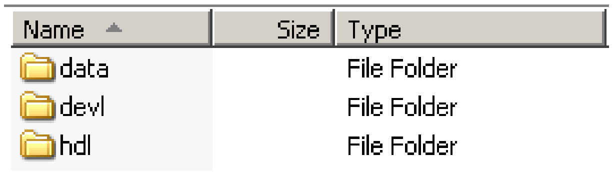
Configure the software accessible registers in your peripheral.

The user specific software accessible registers will be implemented in the user-logic module of your peripheral. Such registers are typically provided for software programs to control and to monitor the status of your user logic. These registers are addressable on the byte, half-word, word, double word or quad word boundaries depending on your design. An example logic for register read/write will be included in the user-logic module generated by the wizard tool for your reference.

User logic software registers may take full advantage of the slave IPIF address-decoding service to generate CE decodes for all of the individual register of interest. The diagram on the left shows the simplest set of IPIC slave signals to read/write the registers.

Number of software accessible registers: 4 (1 to 4096)

Figure 15: Create Peripheral - User S/W Register



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

Figure 16: Custom IP Directory Structure

the corresponding IPIF unit in the way you indicated in the wizard GUI and connect it to the stub user logic where the user logic should get implemented. You are not expected to modify this template file except in certain marked places for adding user specific generics and ports.

  - **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 may be provided for demonstration purpose.
  - XPS interface file(s)

In the **data** directory several source files cab be identified:

- *mb_tut_1\pcores\custom_ip_v1_00_a\data*
  - **custom_ip_v2_1_0.mpd** This Microprocessor Peripheral Description file contains interface information of your peripheral so that other EDK tools can recognize the 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.

In the Drivers directory several source files can be identified:

- *mb_tut_1\drivers\custom_ip_v1_00_a\src* :
  - **custom_ip.h** This is the software driver header template file, which contains address offsets of software addressable registers in your peripheral, as well as some common masks, simple register access macros and function declarations.
  - **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 could be **modified** to incorporate the custom IP functionality.

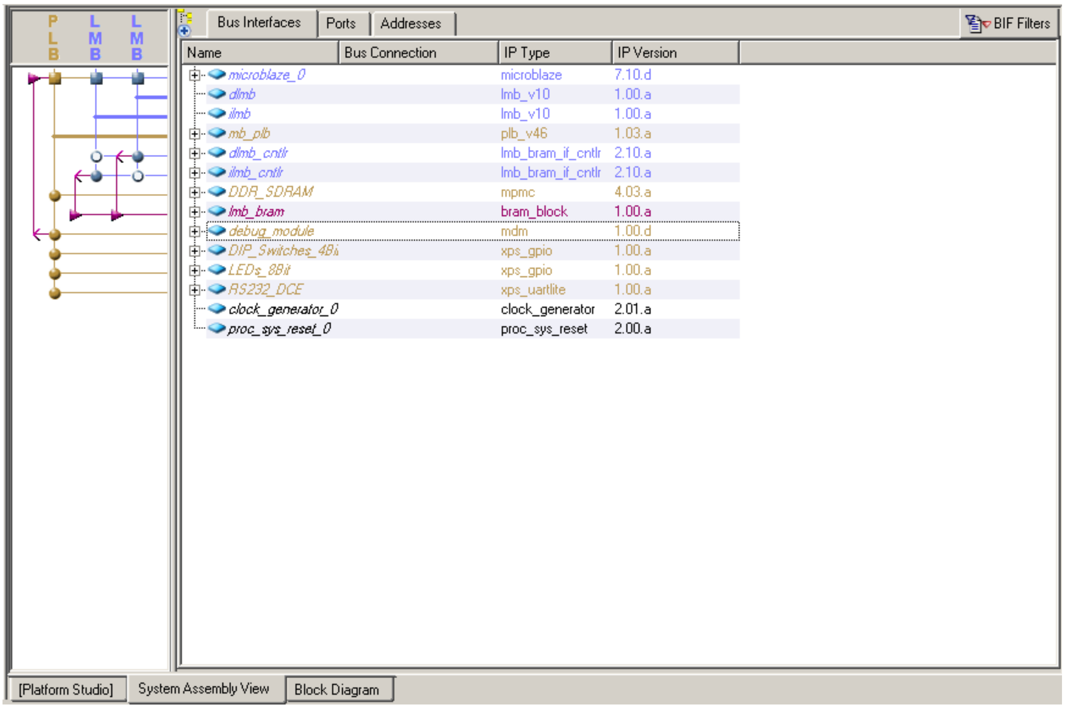
- **Open** user_logic.vhd. Currently the code provides an example of reading and writing to four 32-bit registers. For the purpose of this tutorial, this code will not be modified.
- **Close** user_logic.vhd

  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 Repositories**. XPS also automatically rescans the pcores directory when the project is opened.

6 Design Modification using Platform Studio

Once a design has been created with the Base System Builder, it can be modified from within the System Assembly view.

Double clicking on any of the IP's listed in the System Assembly View allows modification of that particular IP as shown in Figure 17. The System Assembly View has the following tabs:



Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	7.10.d
dlmb		lmb_v10	1.00.a
ilmb		lmb_v10	1.00.a
mb_plb		plb_v46	1.03.a
dlmb_cntlr		lmb_bram_if_cntlr	2.10.a
ilmb_cntlr		lmb_bram_if_cntlr	2.10.a
DDR_SDRAM		mpmc	4.03.a
lmb_bram		bram_block	1.00.a
debug_module		mdm	1.00.d
DIP_Switches_4Bi		xps_gpio	1.00.a
LEDs_8Bit		xps_gpio	1.00.a
RS232_DCE		xps_uartlite	1.00.a
clock_generator_0		clock_generator	2.01.a
proc_sys_reset_0		proc_sys_reset	2.00.a

Figure 17: System Assembly View

Bus Interfaces: With the Bus Interface activated, the patch panel to the left of the System Assembly View gets activated. The bus connectivity of the core is shown when the hierarchy of the IP is expanded.

Ports: With this filter on, the port connections appear when the hierarchy of the IP is expanded. You need to activate this filter to be able to add external ports.

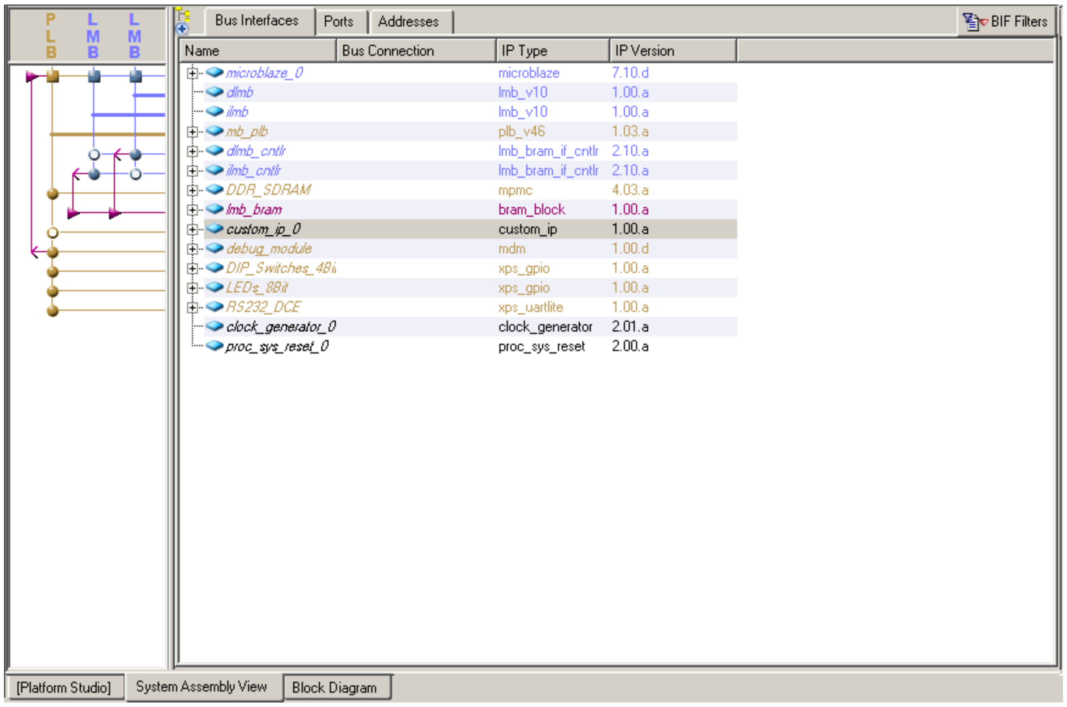
Addresses: The IP's addresses can be viewed when expanding the IP. This is where you can generate addresses for the IP's.

On the left hand side of XPS, the user can choose from three views, Project, Applications and IP Catalog. The IP Catalog tab shows all of the IP that is available to use in the EDK project. **To add new IP**:

- Bring the IP Catalog tab forward.
- Expand the Project Repository hierarchy.
- Select CUSTOME_IP.
- Drag and drop the IP into the System Assembly View or double click on the IP.
- The new IP **custome_ip_0** should appear in the assembly view as shown in Figure 18.

With the Bus Interfaces tab brought forward (Figure 19):

- Press the Connection Filter button and select All
- Expand the custom_ip_0 instance



Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	7.10.d
dlmb		lmb_v10	1.00.a
ilmb		lmb_v10	1.00.a
mb_plb		plb_v46	1.03.a
dlmb_cntlr		lmb_bram_if_cntlr	2.10.a
ilmb_cntlr		lmb_bram_if_cntlr	2.10.a
DDR_SDRAM		mpmc	4.03.a
lmb_bram		bram_block	1.00.a
custom_ip_0		custom_ip	1.00.a
debug_module		mdm	1.00.d
DIP_Switches_4Bi		xps_gpio	1.00.a
LEDs_8Bit		xps_gpio	1.00.a
RS232_DCE		xps_uartlite	1.00.a
clock_generator_0		clock_generator	2.01.a
proc_sys_reset_0		proc_sys_reset	2.00.a

Figure 18: Inserting IP

- High-light the slave PLB connection (SPLB)
- Select the No Connection pull down menu and change it to mb_plb

Note: Right clicking on the Name column in the System Assembly View provides more filtering options.

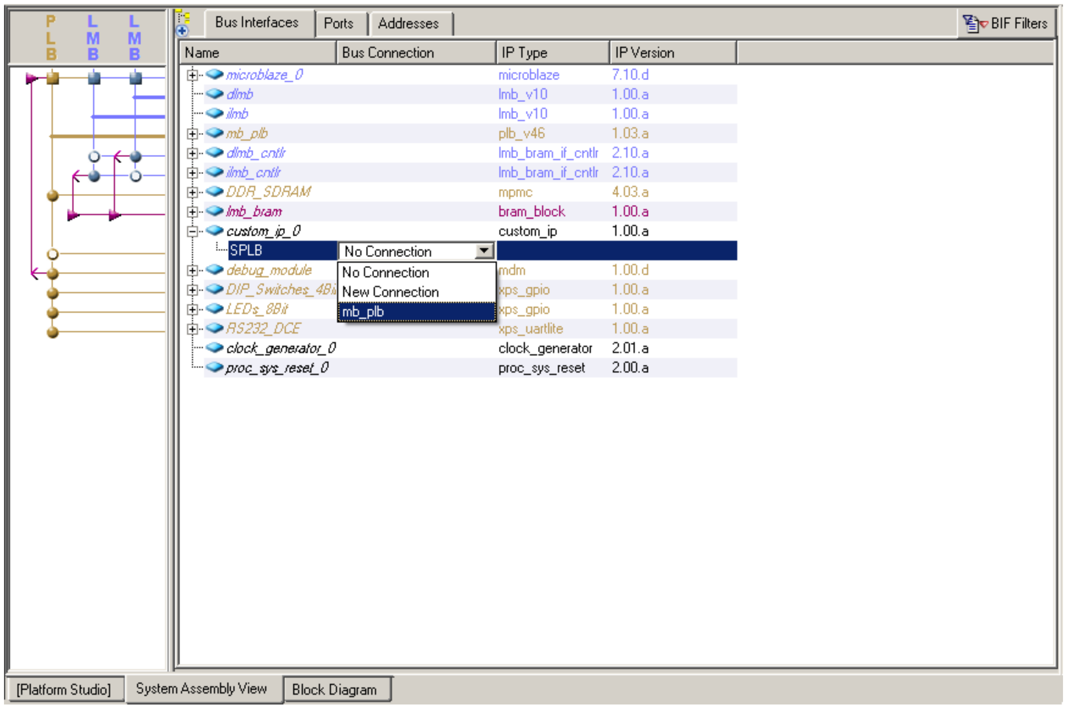
Select the Addresses tab to define an address for the newly added custom_ip peripheral. The address can be assigned by entering the Base Address or the tool can assign an address. For the purpose of this tutorial, the tool will be used to assign an address.

- Change the size of the dlmb_cntlr and ilmb_cntlr to 16K.
- Click ***Generate Addresses.***

A message in the console window will state that the address map has been generated successfully. The design is now ready to be implemented.

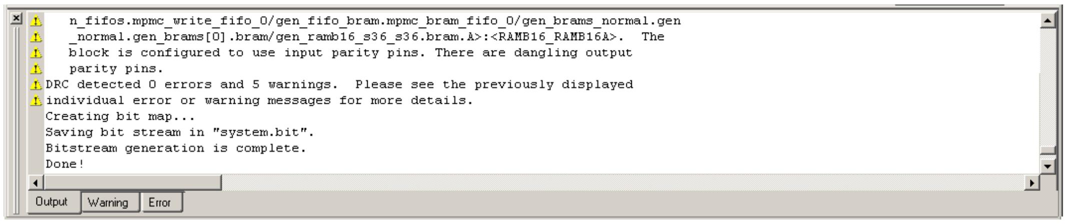
7 Implementing the Design

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:



Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	7.10.d
dlmb		lmb_v10	1.00.a
ilmb		lmb_v10	1.00.a
mb_plb		plb_v46	1.03.a
dlmb_cntlr		lmb_bram_if_cntlr	2.10.a
ilmb_cntlr		lmb_bram_if_cntlr	2.10.a
DDR_SDRAM		mpmc	4.03.a
lmb_bram		bram_block	1.00.a
custom_ip_0		custom_ip	1.00.a
SPLB	No Connection		
debug_module	No Connection	mdm	1.00.d
DIP_Switches_4Bi	New Connection	xps_gpio	1.00.a
LEDs_8Bit	mb_plb	xps_gpio	1.00.a
RS232_DCE		xps_uartlite	1.00.a
clock_generator_0		clock_generator	2.01.a
proc_sys_reset_0		proc_sys_reset	2.00.a

Figure 19: Modifying bus connections



```
   n_fifos.mpmc_write_fifo_0/gen_fifo_bram.mpmc_bram_fifo_0/gen_brams_normal.gen
   _normal.gen_brams[0].bram/gen_ramb16_s36_s36.bram.A>:<RAMB16_RAMB16A>.  The
   block is configured to use input parity pins. There are dangling output
   parity pins.
DRC detected 0 errors and 5 warnings.  Please see the previously displayed
individual error or warning messages for more details.
Creating bit map...
Saving bit stream in "system.bit".
Bitstream generation is complete.
Done!
```

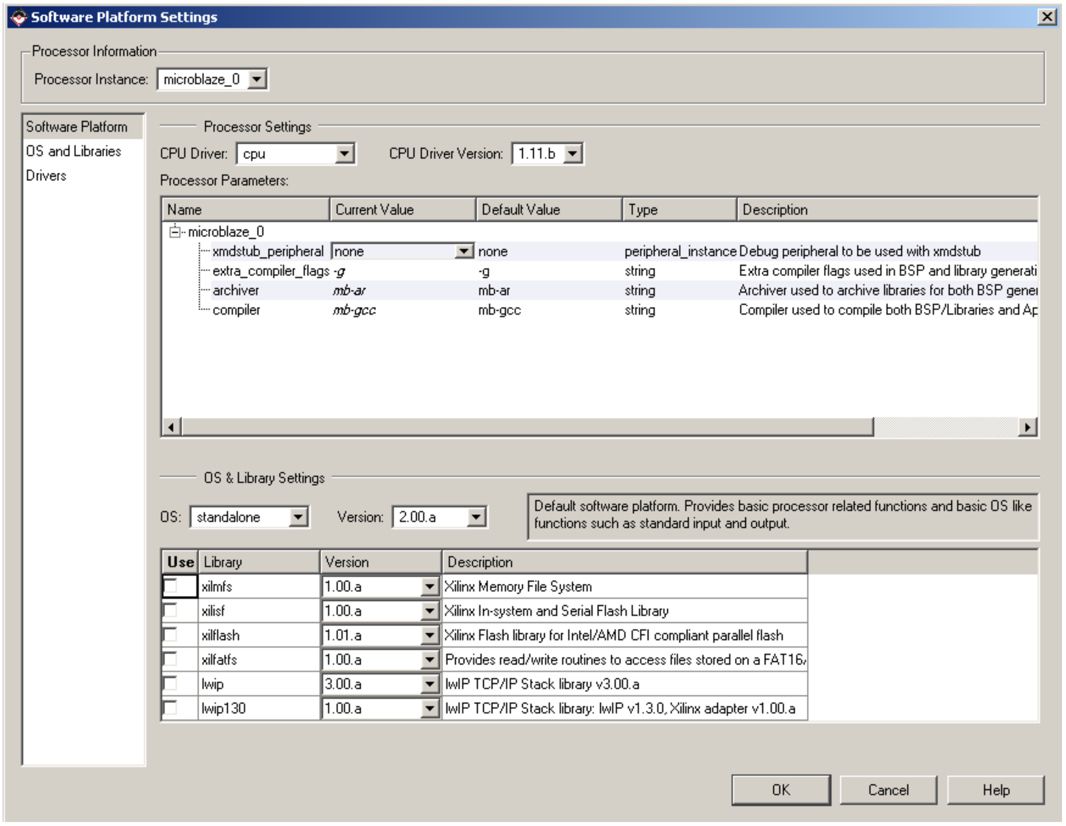
Figure 20: Software Platform Settings Dialog

- **Select** Hardware → Generate Netlist. This will elaborate the MHS file and generate a netlist for the complete system (this will take a while!).
- **Select** Hardware → Generate Bitstream. This will call ISE tools to implement the design and generate a bit file that could be downloaded into the FPGA.

At the end of this step the XPS output screen should look like Figure 20. The bit file that is generate is called **system.bit** which contains all the required information to configure the FPGA except the contents of the block ram (application/data). The bit file will be updated with the application code after defining the software design.

8 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.



Processor Information

Processor Instance: microblaze_0

Processor Settings

CPU Driver: cpu   CPU Driver Version: 1.11.b

Processor Parameters:

Name	Current Value	Default Value	Type	Description
microblaze_0				
xmdstub_peripheral	none	none	peripheral_instance	Debug peripheral to be used with xmdstub
extra_compiler_flags	*-g*	-g	string	Extra compiler flags used in BSP and library generati
archiver	*mb-ar*	mb-ar	string	Archiver used to archive libraries for both BSP gene
compiler	*mb-gcc*	mb-gcc	string	Compiler used to compile both BSP/Libraries and Ap

OS & Library Settings

OS: standalone   Version: 2.00.a

Default software platform. Provides basic processor related functions and basic OS like functions such as standard input and output.

Use	Library	Version	Description
☐	xilmfs	1.00.a	Xilinx Memory File System
☐	xilisf	1.00.a	Xilinx In-system and Serial Flash Library
☐	xilflash	1.01.a	Xilinx Flash library for Intel/AMD CFI compliant parallel flash
☐	xilfatfs	1.00.a	Provides read/write routines to access files stored on a FAT16/
☐	lwip	3.00.a	lwIP TCP/IP Stack library v3.00.a
☐	lwip130	1.00.a	lwIP TCP/IP Stack library: lwIP v1.3.0, Xilinx adapter v1.00.a

Figure 21: Software Platform Settings Dialog

8.1 Configuration of the BSP

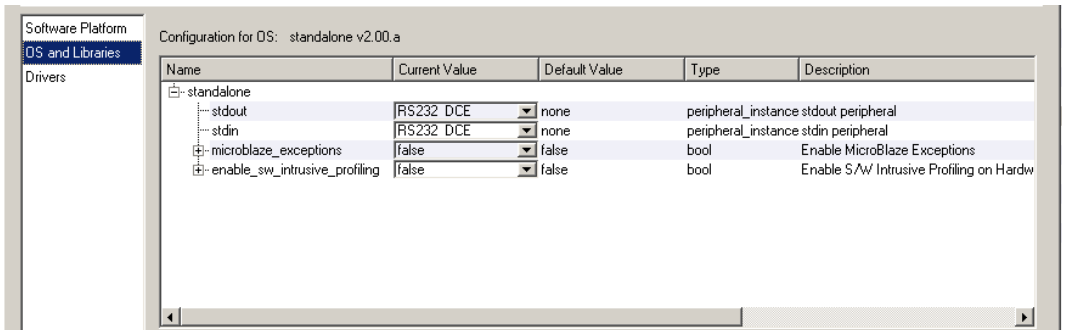
Configuration of the BSP is done using the Software Platform Settings dialogue. Inside of ISE, double click on the EDK project file to open up XPS. In XPS, **select** *Software* → *Software Platform Settings* . . . This will open the Software Platform Settings dialogue box as shown in Figure 21. The Software Platform Settings dialogue box contains three views. Each view is used to control all aspects of the BSP creation.

- The Software Platform view allows the user to modify processor parameters, driver, operating system and libraries. The following Operating Systems are supported:
  - Standalone
  - xilkernel
  - uclinux
  - nucleus

  No changes are required in this view.

- **Select** the OS and Libraries view as shown in Figure 22. This view allows the user to configure OS and library parameters.

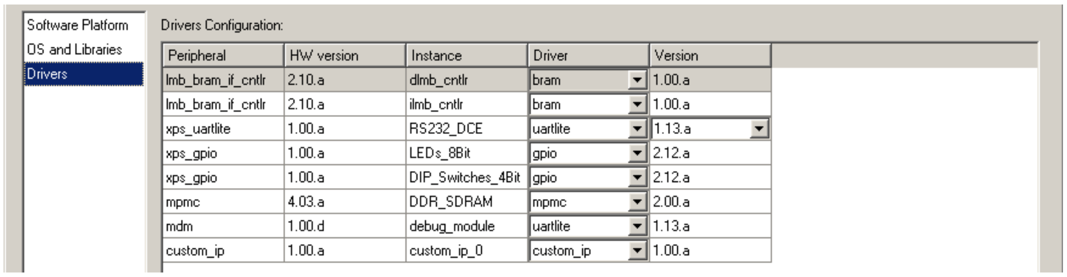
  No changes are required.



Configuration for OS: standalone v2.00.a

Name	Current Value	Default Value	Type	Description
standalone				
stdout	RS232_DCE	none	peripheral_instance	stdout peripheral
stdin	RS232_DCE	none	peripheral_instance	stdin peripheral
microblaze_exceptions	false	false	bool	Enable MicroBlaze Exceptions
enable_sw_intrusive_profiling	false	false	bool	Enable S/W Intrusive Profiling on Hardw

Figure 22: OS and Libraries view



Drivers Configuration:

Peripheral	HW version	Instance	Driver	Version
lmb_bram_if_cntlr	2.10.a	dlmb_cntlr	bram	1.00.a
lmb_bram_if_cntlr	2.10.a	ilmb_cntlr	bram	1.00.a
xps_uartlite	1.00.a	RS232_DCE	uartlite	1.13.a
xps_gpio	1.00.a	LEDs_8Bit	gpio	2.12.a
xps_gpio	1.00.a	DIP_Switches_4Bit	gpio	2.12.a
mpmc	4.03.a	DDR_SDRAM	mpmc	2.00.a
mdm	1.00.d	debug_module	uartlite	1.13.a
custom_ip	1.00.a	custom_ip_0	custom_ip	1.00.a

Figure 23: Drivers view

- **Select** the Drivers view. This view allows you to select the software versions for the peripherals in the system as shown in Figure 23. Notice that the driver version is independent of the HW version.

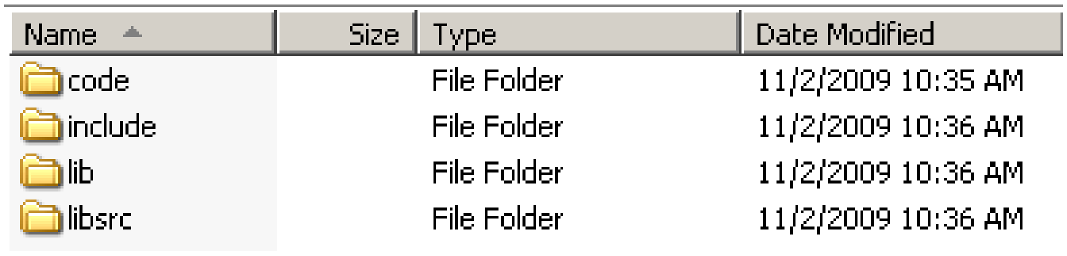
The Interrupt Handlers view allows you to modify the parameters for the interrupts. This project does not have any interrupts.

- Click **OK.**
- In XPS, **select** Software → 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 24:

- 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	Date Modified
code		File Folder	11/2/2009 10:35 AM
include		File Folder	11/2/2009 10:36 AM
lib		File Folder	11/2/2009 10:36 AM
libsrc		File Folder	11/2/2009 10:36 AM

Figure 24: MicroBlaze Drivers Directories

8.2 Building the User Application

In EDK 10.1, XPS provides for the user with the ability 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 25. The Base System Builder (BSB) generates a sample application which tests a subset of the peripherals included in the design.

8.3 Compiling the Code

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

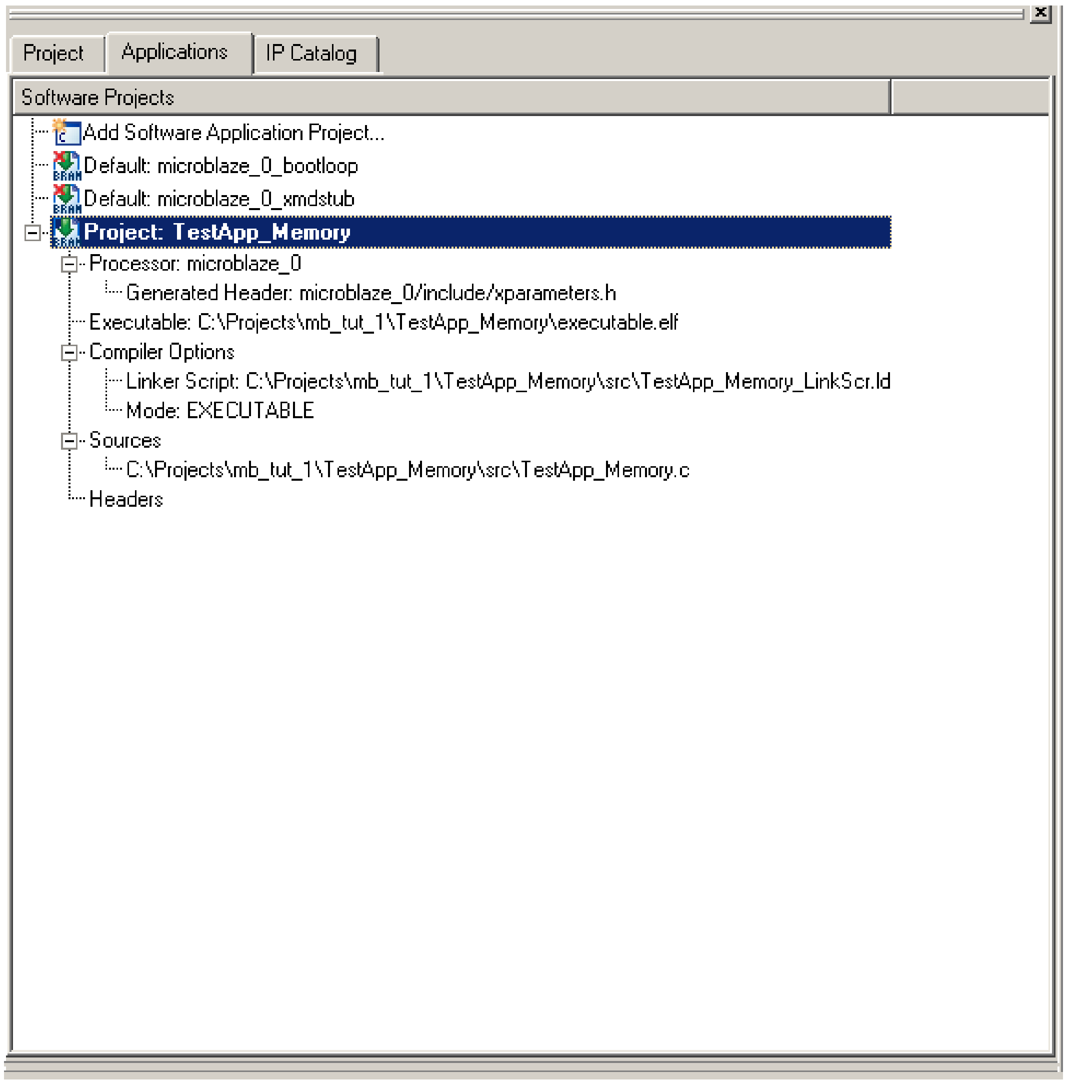
- **Select** Software → Build All User Applications to run mb-gcc. Mb-gcc compiles the source files as shown in Figure 26.

Note: The software flow is independent of hardware implementation. Every time a change is made to the software part of the system, there will be no need to build the hardware. Just update the bitstream file with the new application.

9 Downloading the Design

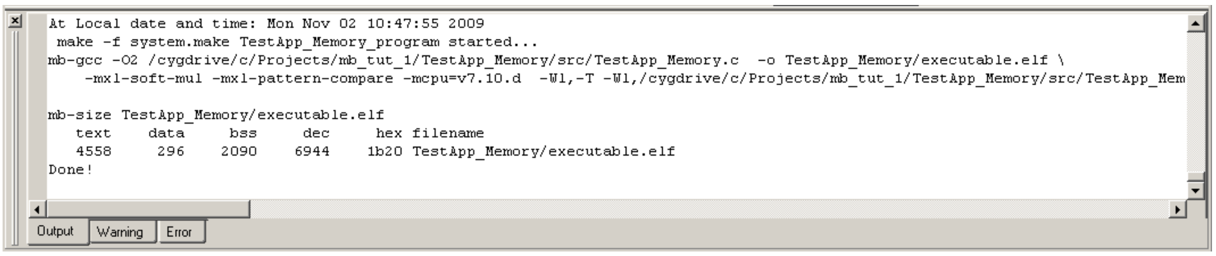
Now that the hardware and software designs are completed, the device can be configured. Follow these steps to download and configure the FPGA:

- Connect the host computer to the target board, including connecting the Xilinx USB download 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



Software Projects
- Add Software Application Project...
- Default: microblaze_0_bootloop
- Default: microblaze_0_xmdstub
- **Project: TestApp_Memory**
  - Processor: microblaze_0
    - Generated Header: microblaze_0/include/xparameters.h
  - Executable: C:\Projects\mb_tut_1\TestApp_Memory\executable.elf
  - Compiler Options
    - Linker Script: C:\Projects\mb_tut_1\TestApp_Memory\src\TestApp_Memory_LinkScr.ld
    - Mode: EXECUTABLE
  - Sources
    - C:\Projects\mb_tut_1\TestApp_Memory\src\TestApp_Memory.c
  - Headers

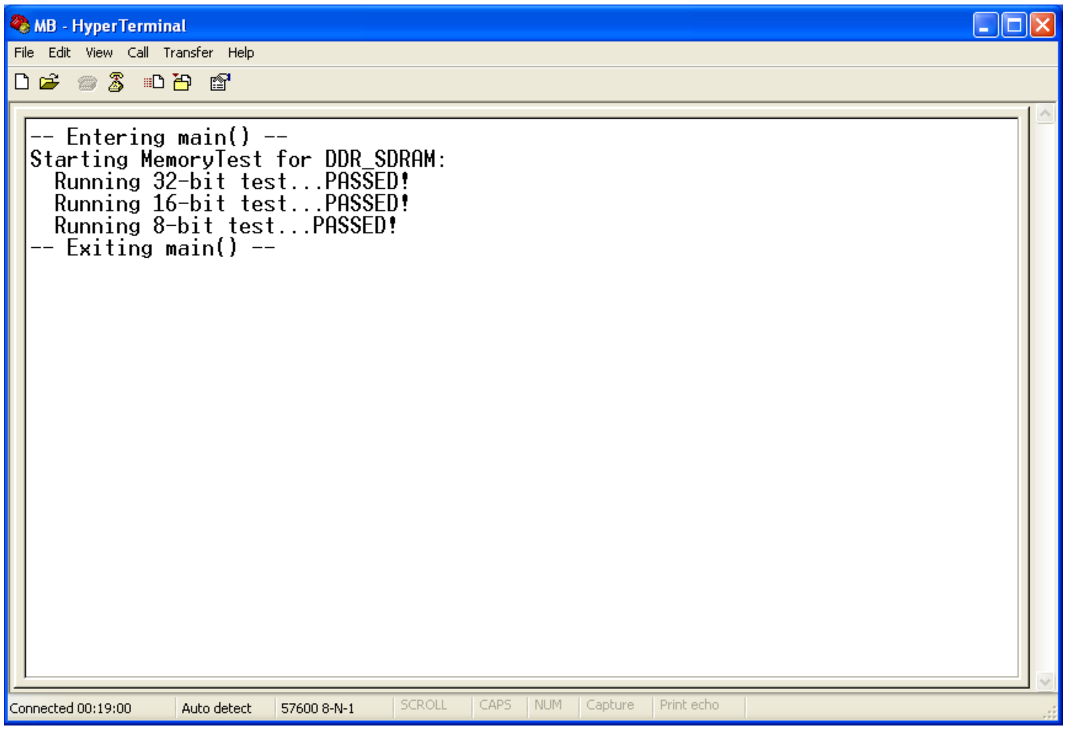
Figure 25: Applications Tab



```
At Local date and time: Mon Nov 02 10:47:55 2009
 make -f system.make TestApp_Memory_program started...
mb-gcc -O2 /cygdrive/c/Projects/mb_tut_1/TestApp_Memory/src/TestApp_Memory.c  -o TestApp_Memory/executable.elf \
    -mxl-soft-mul -mxl-pattern-compare -mcpu=v7.10.d  -Wl,-T -Wl,/cygdrive/c/Projects/mb_tut_1/TestApp_Memory/src/TestApp_Mem

mb-size TestApp_Memory/executable.elf
   text    data     bss     dec     hex filename
   4558     296    2090    6944    1b20 TestApp_Memory/executable.elf
Done!
```

Figure 26: XPS Output Window - Software Compiled



```
-- Entering main() --
Starting MemoryTest for DDR_SDRAM:
  Running 32-bit test...PASSED!
  Running 16-bit test...PASSED!
  Running 8-bit test...PASSED!
-- Exiting main() --
```

Figure 27: Hyperterminal Output

  - Data bits: 8
  - Parity: none
  - Stop bits: 1
  - Flow control: none

- Connect the board power
- In EDK, select Device Configuration → Update Bitstream. This will update the bit file with the application compiled code. Repeat this step each time the application changes.
- Select Device Configuration → Download Bitstream. This will start device configuration software (iMPACT) within EDK and executes the download command file **etc/download.cmd**.
- iMPACT will download the file **download.bit** on the FPGA.

Note: In some cases iMPACT fails to recognize the cable automatically. In this case etc/-download.cmd needs to be modified to manually select the cable. Modify the line setCable -p Auto to point to the correct cable; usb21 for example
After the configuration is complete, you should see a display similar to that in shown in Figure 27: