EDK MicroBlaze Tutorial

XILINX®

The following table shows the revision history for this document:

	Version	Revision
11/2002	1.0	Initial Xilinx release.
04/2003	1.1	Updated to support the EDK 3.2 release.
9/2003	2.0	Updated to support the EDK 6.1 release.
03/2004	2.1	Updated to support the EDK 6.2 release.

Preface: About This Manual

EDK MicroBlaze Tutorial

About This Manual

This tutorial guides you through the process of finishing and testing a partially completed MicroBlaze system design using the Embedded Development Kit (EDK).

Additional Resources

For additional information, go to http://support.xilinx.com. The following table lists some of the resources you can access from this website. You can also directly access these resources using the provided URLs.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://support.xilinx.com/support/techsup/tutorials/index.htm
Answer Browser	Database of Xilinx solution records http://support.xilinx.com/xlnx/xil_ans_browser.jsp
Application Notes	Descriptions of device-specific design techniques and approaches http://support.xilinx.com/apps/appsweb.htm
Data Book	Pages from *The Programmable Logic Data Book*, which contains device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://support.xilinx.com/partinfo/databook.htm
Problem Solvers	Interactive tools that allow you to troubleshoot your design issues http://support.xilinx.com/support/troubleshoot/psolvers.htm
Tech Tips	Latest news, design tips, and patch information for the Xilinx design environment http://www.support.xilinx.com/xlnx/xil_tt_home.jsp

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *`design_name`*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *`design_name`*
	References to other manuals	See the *Development System Reference Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** [*`option_name`*] *`design_name`*
Braces { }	A list of items from which you must choose one or more	**`lowpwr ={on|off}`**
Vertical bar |	Separates items in a list of choices	**`lowpwr ={on|off}`**

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis ...	Repetitive material that has been omitted	**`allow block`** *`block_name loc1 loc2 ... locn;`*

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current file or in another file in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Virtex-II Handbook*.
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

EDK MicroBlaze Tutorial

This tutorial guides you through the process of finishing and testing a partially completed MicroBlaze system design using the Embedded Development Kit (EDK). The following steps are included in this tutorial:

"Starting XPS"

"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 will illustrate the Base System Builder."

"Generating a Netlist and Creating a Project Navigator Project"

"Implementing the Tutorial Design"

"Configuration of the BSP"

"Finishing the Tutorial Application Code"

"Compiling the Code"

"Downloading the Design"

"Debugging the Design"

"Simulating the Design"

System Requirements

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

- Windows 2000 SP2 / Windows XP
- EDK 6.2i or later
- ISE 6.2i or later

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.

Hardware Requirements

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

- Memec Design Virtex-II Pro Development Board (2VP7 FG456 -6)
- Xilinx Parallel Cable 4 used to program and debug the device
- Serial Cable

Note: It should be noted that other hardware can 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 Hardware 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:

- MB
- Local Memory Bus (LMB) Bus
  - LMB_BRAM_IF_CNTLR
  - BRAM_BLOCK
- On-chip Peripheral Bus (OPB) BUS
  - OPB_SDRAM_CNTLR
  - OPB_GPIO
  - OPB_BRAM_IF_CNTLR
  - OPB BRAM
  - OPB_UARTLITE

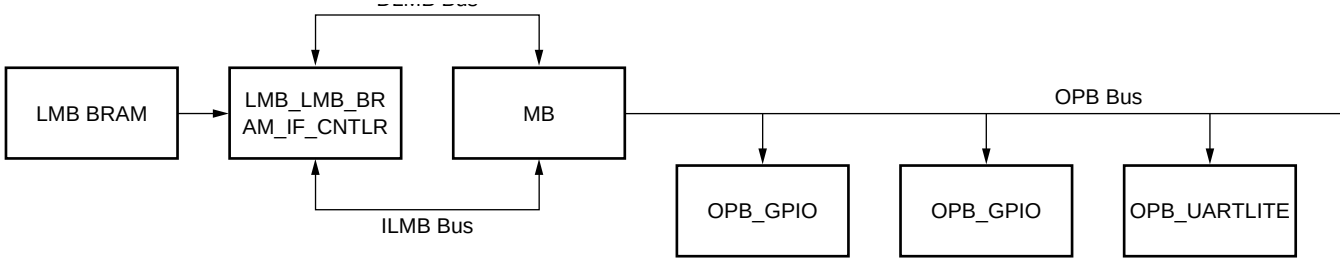

Figure 1: **Tutorial Design Hardware Components**

Tutorial Design Memory Map

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

Table 1: **Tutorial Design Memory Map**

Device	Address		Size	Comment
	Min	Max		
LMB_BRAM	0x0000_0000	0x0000_3FFF	16KB	LMB Memory
OPB_MDM	0x8FFF_8000	0x8FFF_80FF	256B	MDM Module
OPB_UARTLITE	0x8FFF_8100	0x8FFF_81FF	256B	Serial Output
OPB_GPIO	0x8FFF_8200	0x8FFF_83FF	256B	LED Output
OPB_GPIO	0x8FFF_8400	0x8FFF_85FF	256B	Push Buttons
OPB_GPIO	0x8FFF_8600	0x8400_87FF	256B	DIP Switches
OPB_SDRAM	0x8600_0000	0x87FF_FFFF	32MB	OPB SDRAM
OPB_BRAM	0x8400_0000	0x8400_3FFF	16KB	OPB Memory

Starting the Tutorial

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 Software Specification (MHS) file

  Note: For more information on the MHS file, refer to the "Microprocessor Hardware Specification" chapter in the *Embedded Systems Tool Guide*.
- Microprocessor Software Specification (MSS) file

  Note: For more information on the MSS file, refer to the "Microprocessor Software Specification" chapter in the *Embedded Systems Tool 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.

Starting XPS

1. To open XPS, select the following:

   `Start` → **`Programs`** → **`Xilinx Embedded Development Kit`** → **`Xilinx Platform Studio`**

2. **`Select File`** → **`New Project`** → **Base System Builder** (BSB) to open the Create New Project Using BSB Wizard dialog box shown in figure 2.

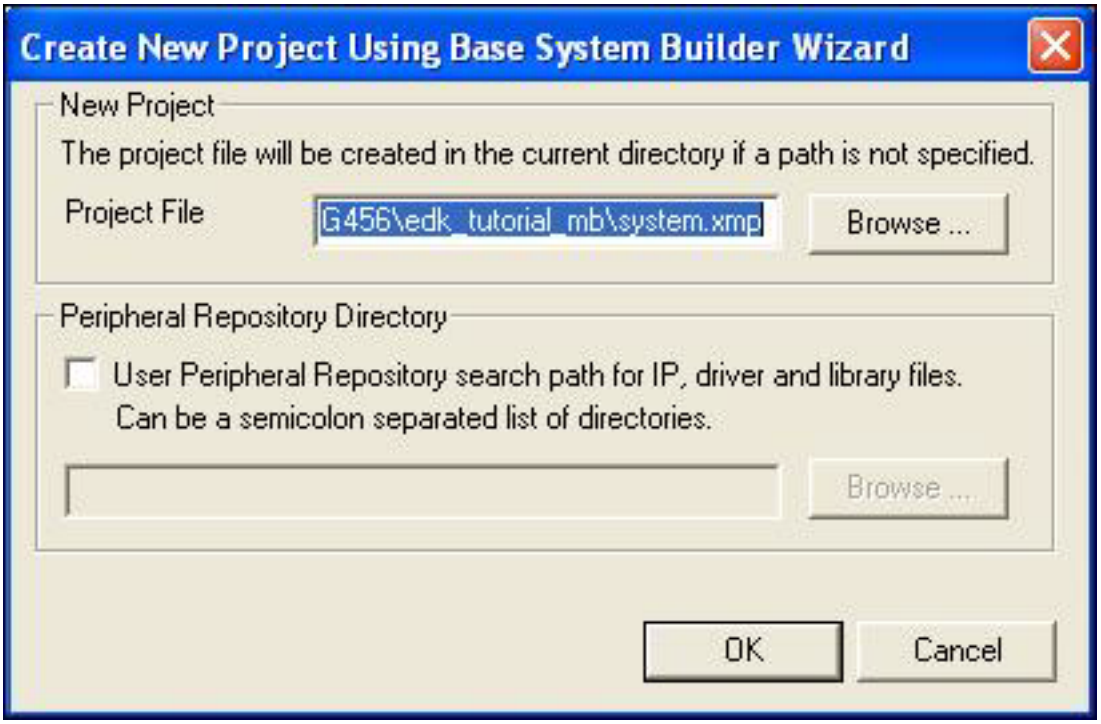

Figure 2: **Create New Project Using Base System Builder Wizard**

3. Use the Project File Browse button to browse to the directory where the tutorial will be located. For the purpose of this tutorial the edk_tutorial_mb directory will be used as shown in figure 3. Click **Open** to create the system.xmp file.

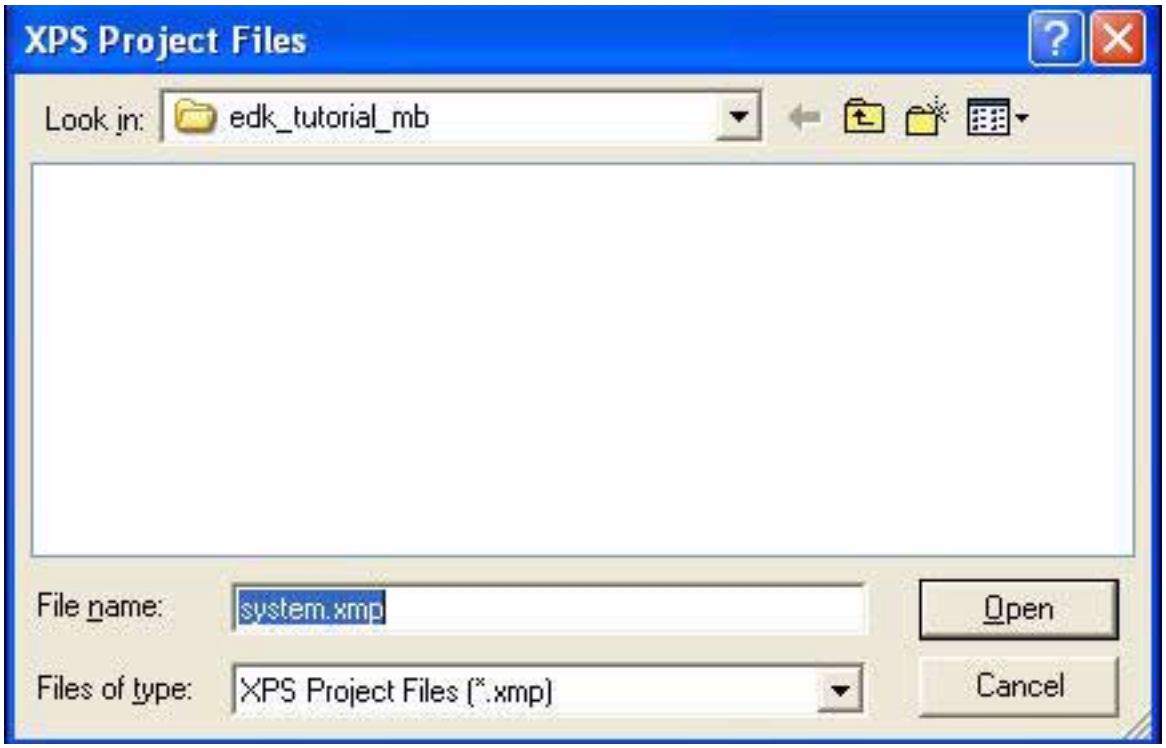

Figure 3: **Specifying XMP file**

4. Click **Open** and then **OK** to start the BSB wizard.

Defining the System Hardware

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” and “Microprocessor Hardware Specification” chapters in the *Embedded Systems Tool 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 will illustrate the Base System Builder.

Using the Base System Builder Wizard

Use the following steps to create the processor system:

1. In the Base System Builder - Select a Board Dialog select the following, as shown in Figure 4:
   - **Board Vendor: Memec Design**
   - **Board Name: Virtex-II Pro P7 fg456 Development Board**
   - **Board Revision: 4**

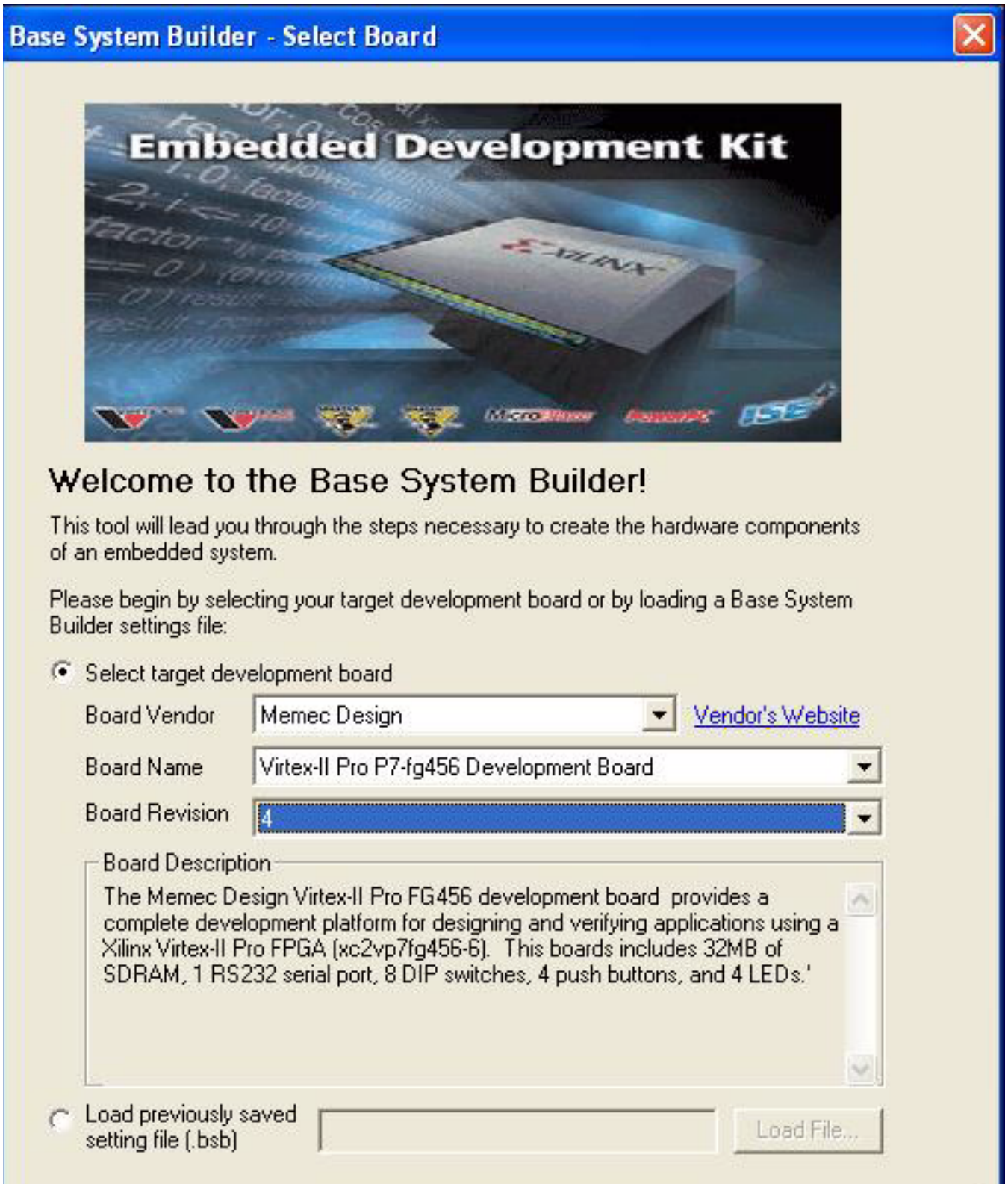

Figure 4: **BSB: Select a Board**

2. Click **Next**. Select the processor to be used in your system, as shown below:

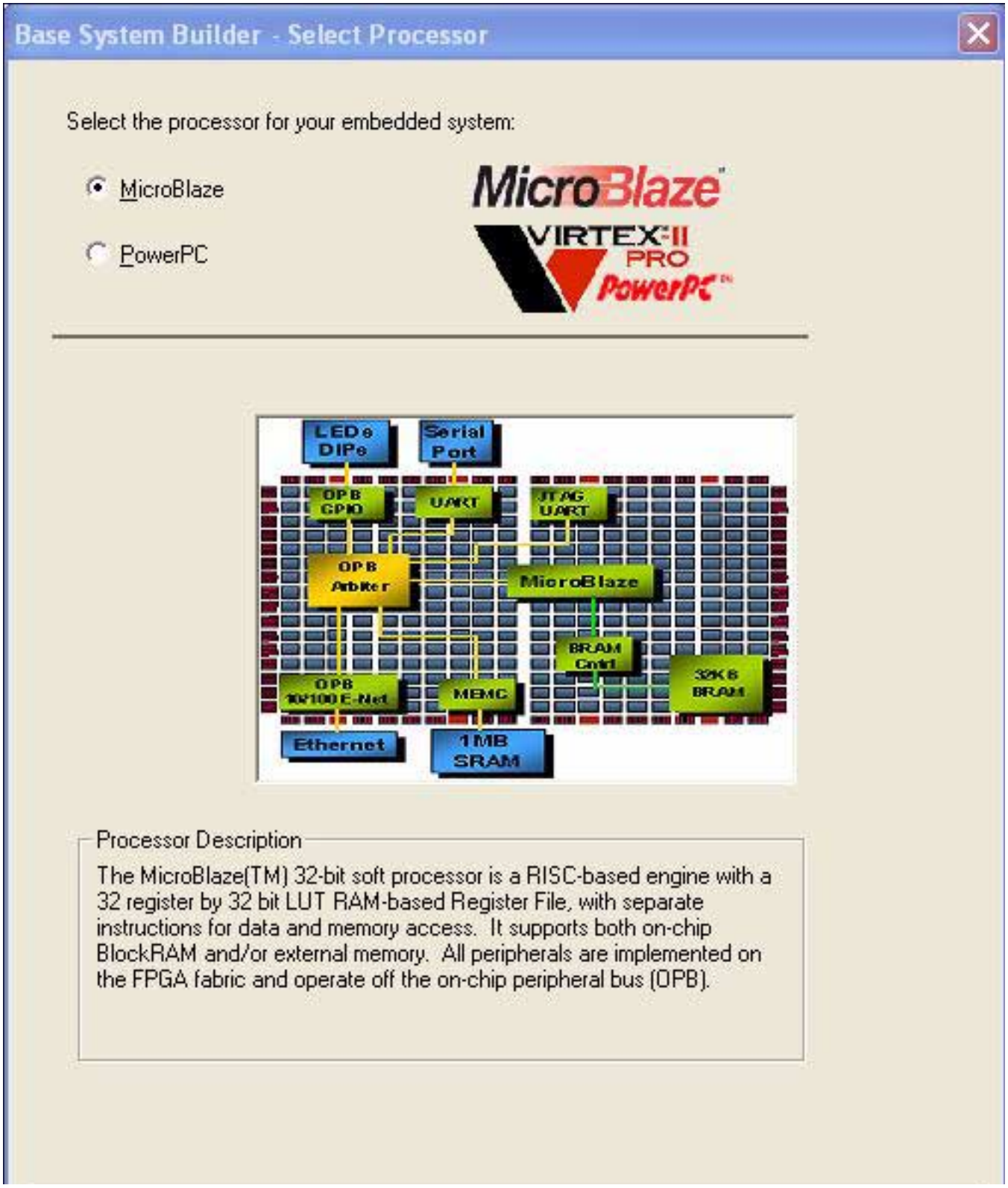

Figure 5: **Processor Selection**

3. Click **Next**. You will now specify several processor options as shown in figure 6:

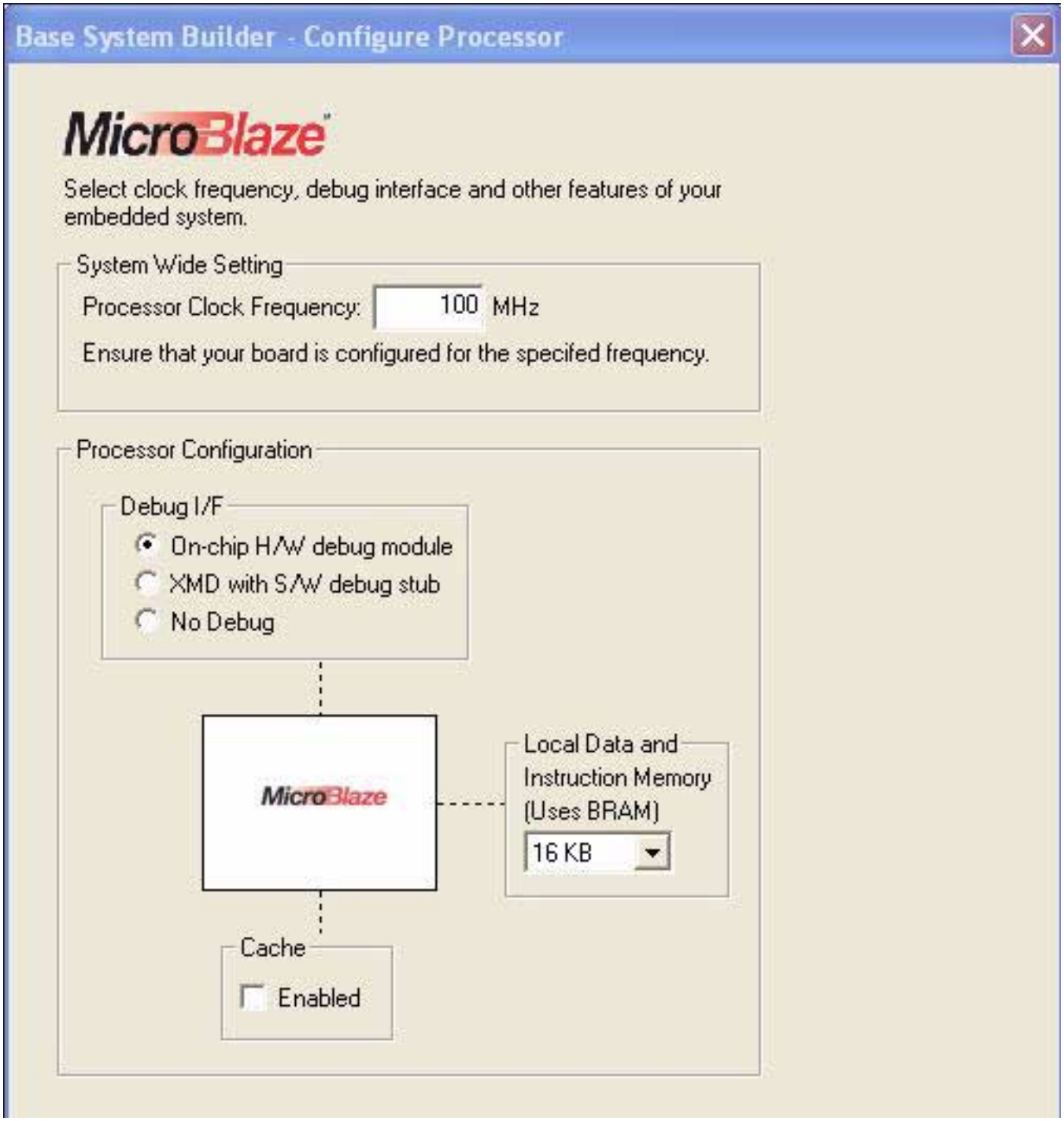

Figure 6: **Configure Processor**

4. The following is an explanation of the setting specified in figure 6:
   - 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 400 byte stub which 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 Embedded Tools Reference Guide.
- Users can also specify the size of the local instruction and data memory. You can also specify the use of a cache.

5. Click **Next.** Select each of the peripheral available as shown in figure 7.

Note: The Baud Rate for the UARTLite must be updated to 115200.

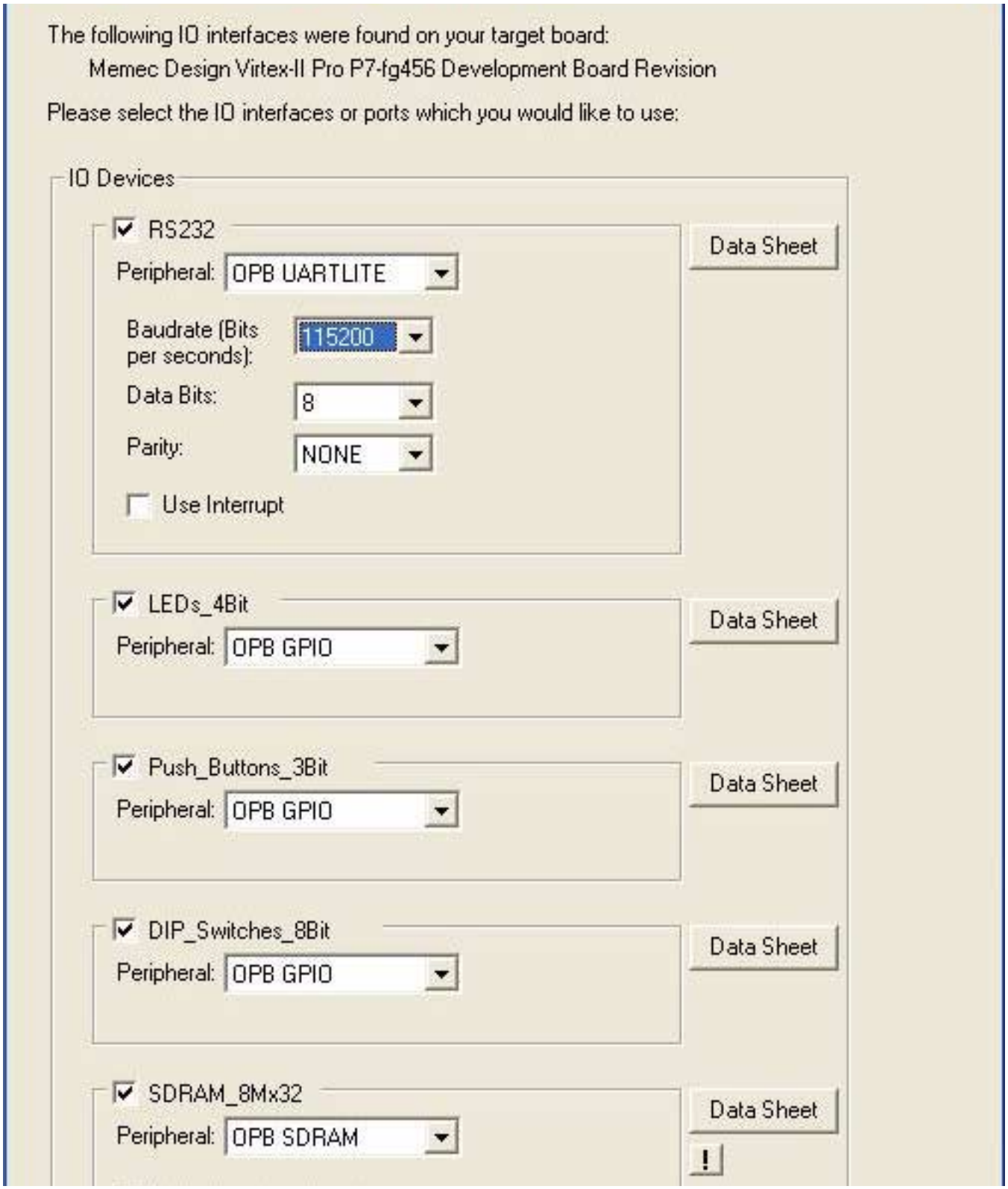

Figure 7: **Port Connections**

6. Click **Next**. Deselect the SysACE_CompactFlash component.
7. Click **Next.** Click on **Add Peripheral.** Add a 16KB OPB BRAM IF CNTLR as shown in figure 8.

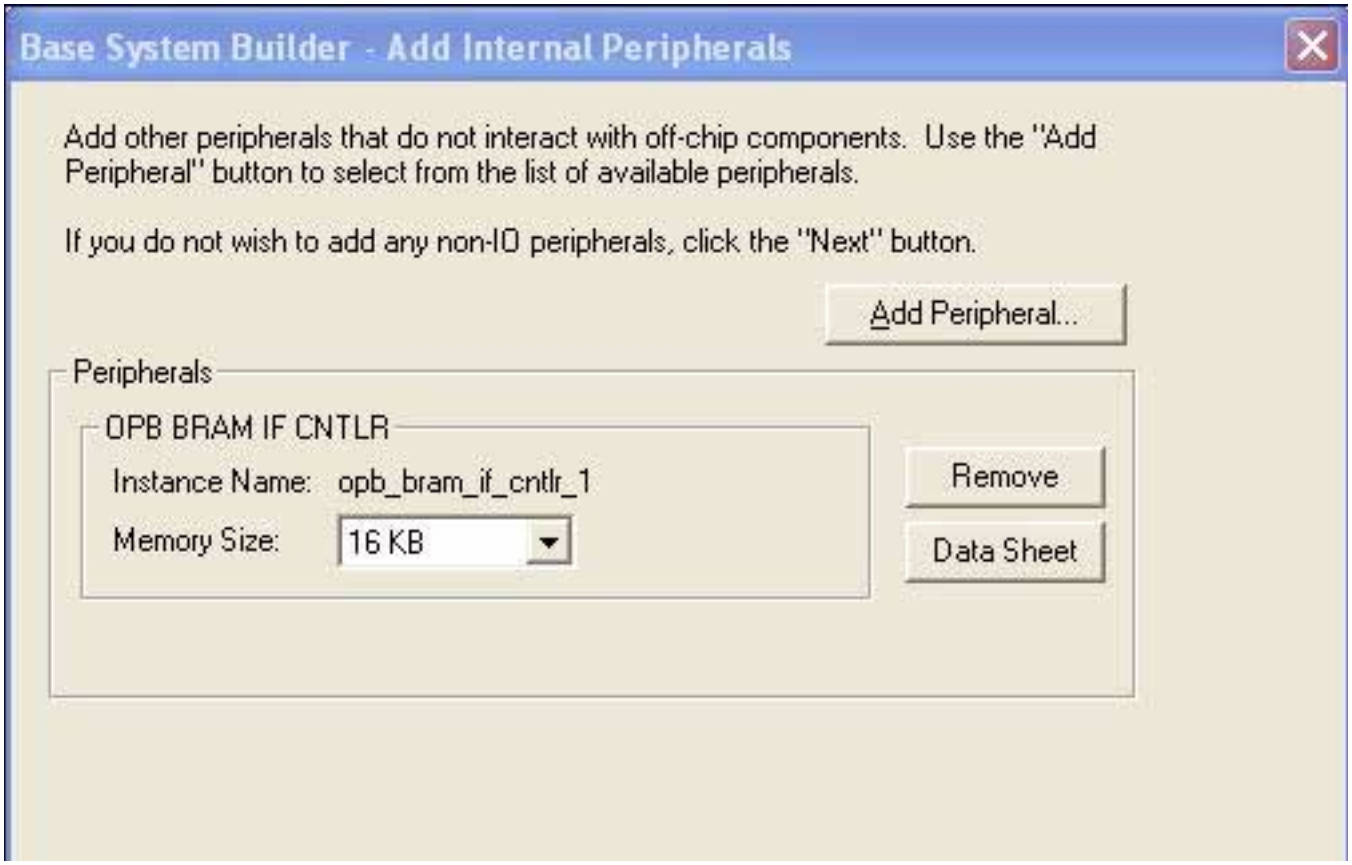

Figure 8: **Adding OPB BRAM CNTLR**

8. Using the Software Configuration dialog box as shown in figure 10, specify the following software settings:
   - Standard Input -> RS232
   - Standard Output -> RS232
   - Instruction ->ilmb_cntlr
   - Data -> dlmb_cntlr
   - Stack / Heap -> dlmb_cntlr

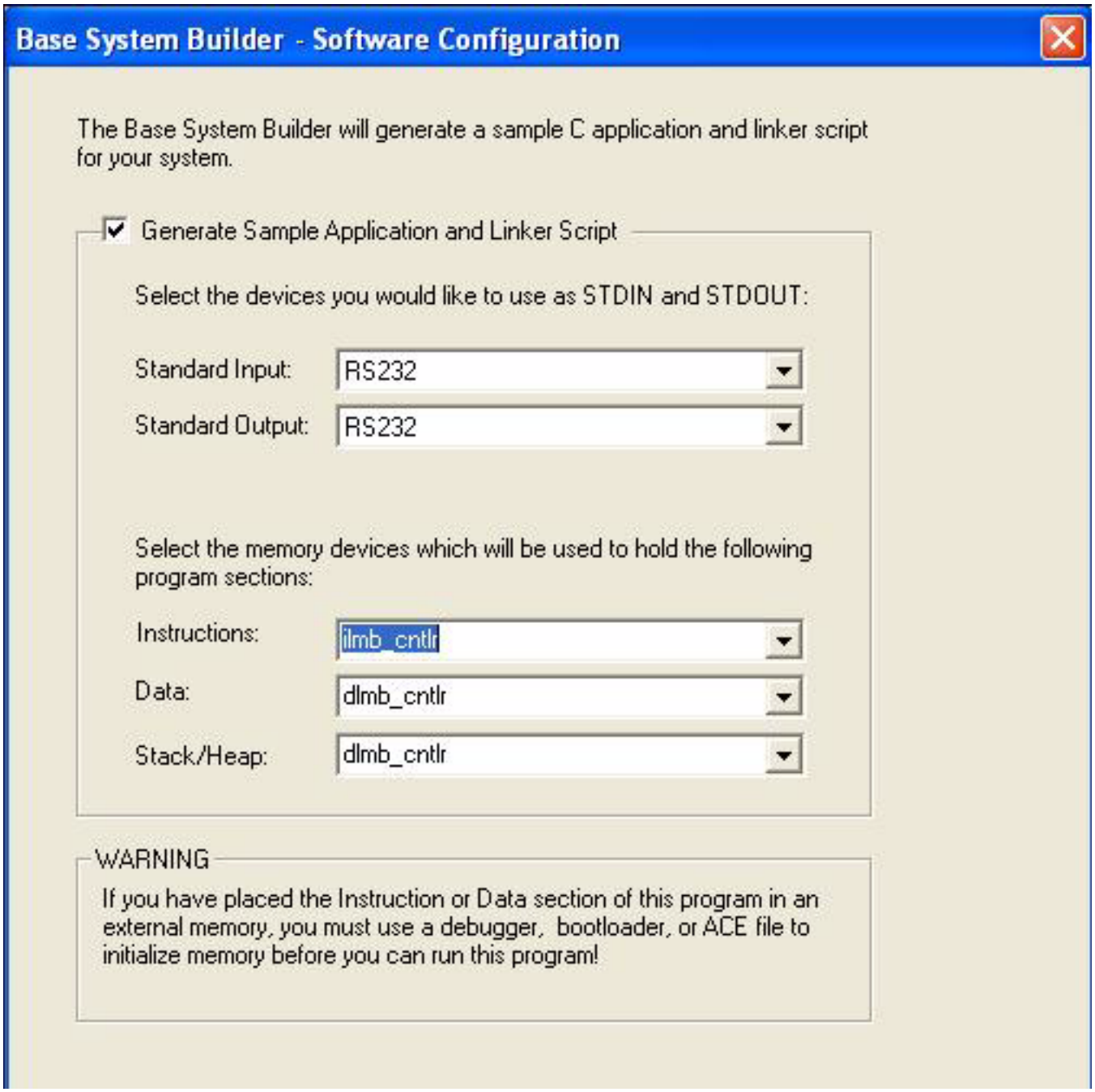

Figure 9: **Software Configuration**

9. Click **Next**. The completed system including the memory map will be displayed as shown in figure 10. Currently the memory map can not be changed or updated in the BSB. If you want to change the memory map you can do this in XPS.

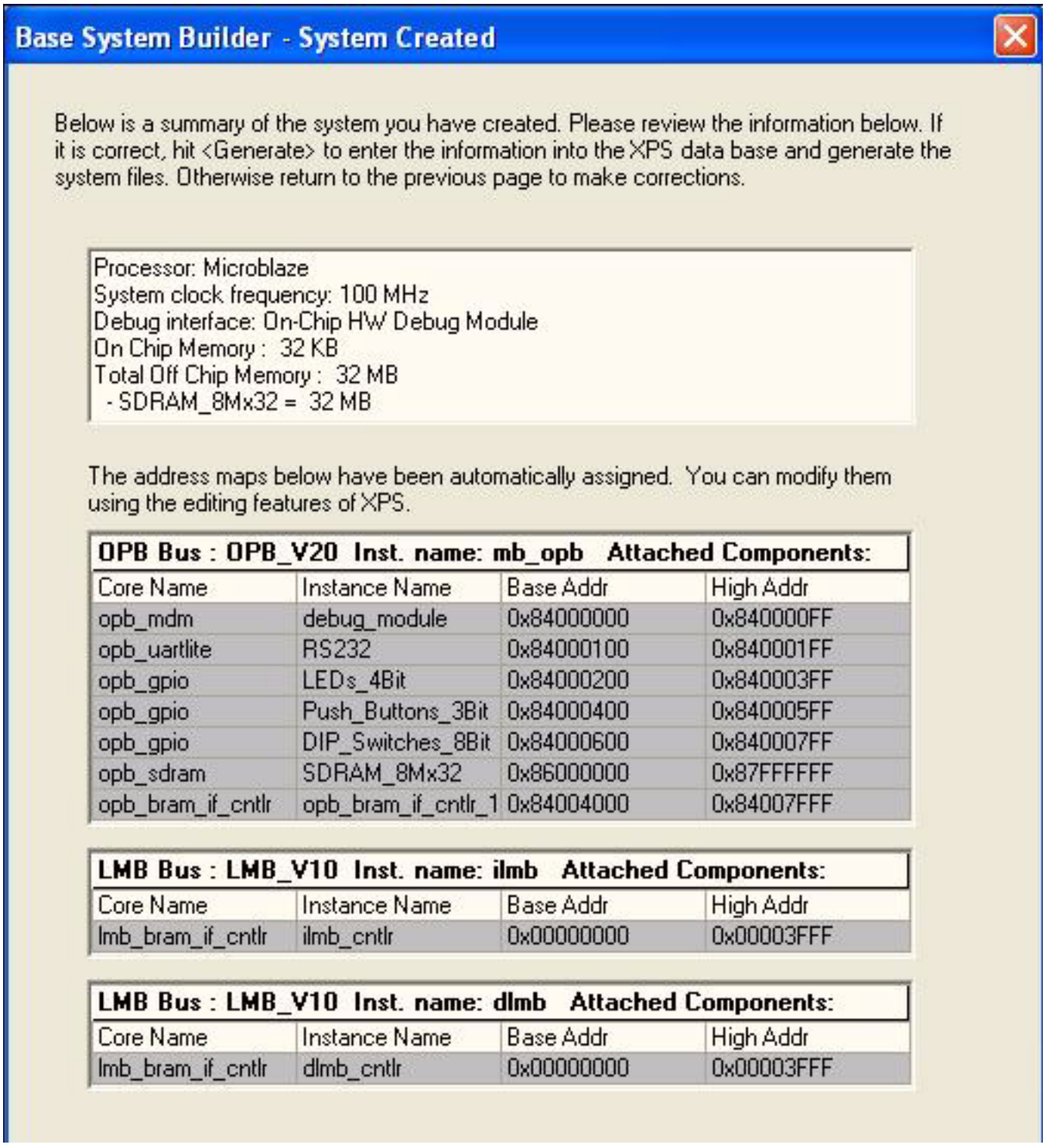

Figure 10: **Completed Processor System**

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

Generating a Netlist and Creating a Project Navigator Project

Now that the hardware has been completely specified in the MHS file, you can run the Platform Generator. The Platform Generator elaborates the MHS file into a hardware system consisting of NGC files that represent the processor system. To generate a netlist and create a Project Navigator project, follow these steps:

1. In XPS, select **Tools** → **Generate Netlist** to create the following directories:
   - implementations
   - hdl
   - synthesis
   - xst
2. To specify the design hierarchy and implementation tool flow, select:

   Options → **Project Options,** select the **Hierarchy and Flow** tab.

   The following dialog box is displayed:

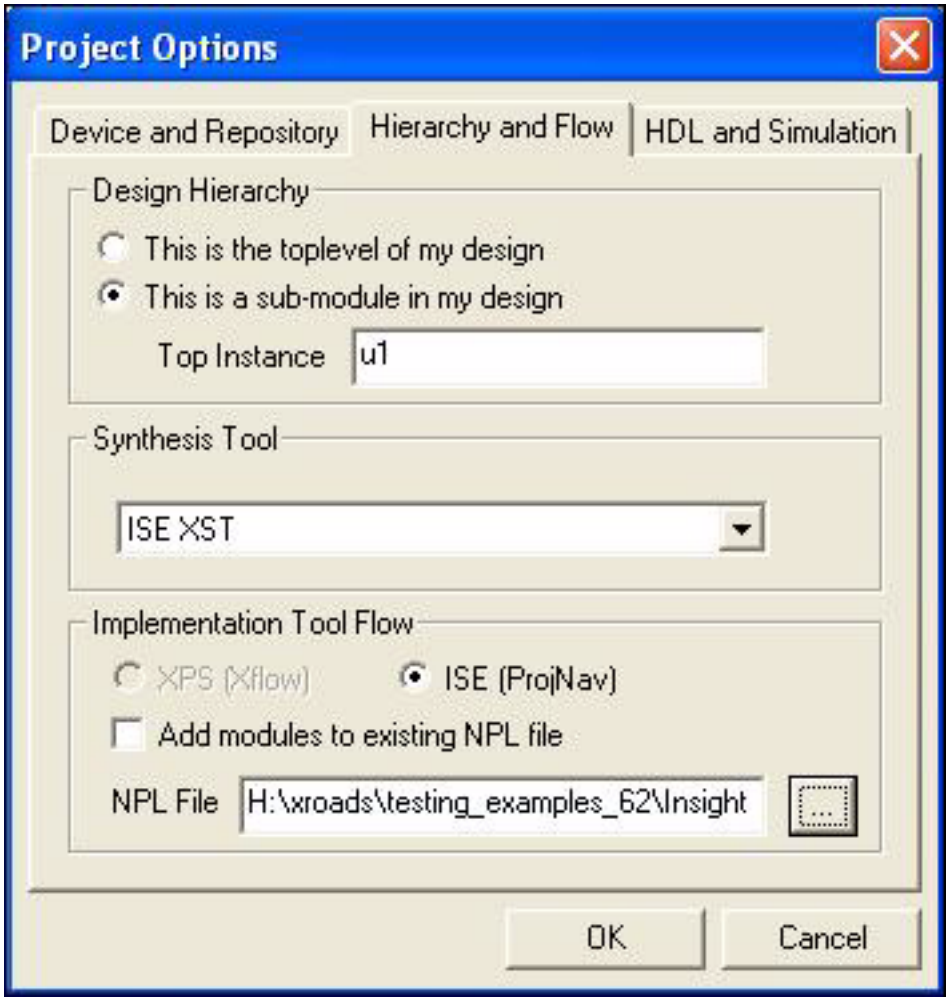

Figure 11: **Project Options**

3. In the Project Options dialog box, select the Hierarchy and Flow tab.
4. Select the following options:

Design Hierarchy: **This is a sub-module in my design**

Top Instance: **u1**

Synthesis Tool: **ISE XST**

Implementation Tool Flow: **ISE (ProjNav)**

In the NPL File field, follow these steps:

a. Click the "…" button to open the dialog shown in figure 13.

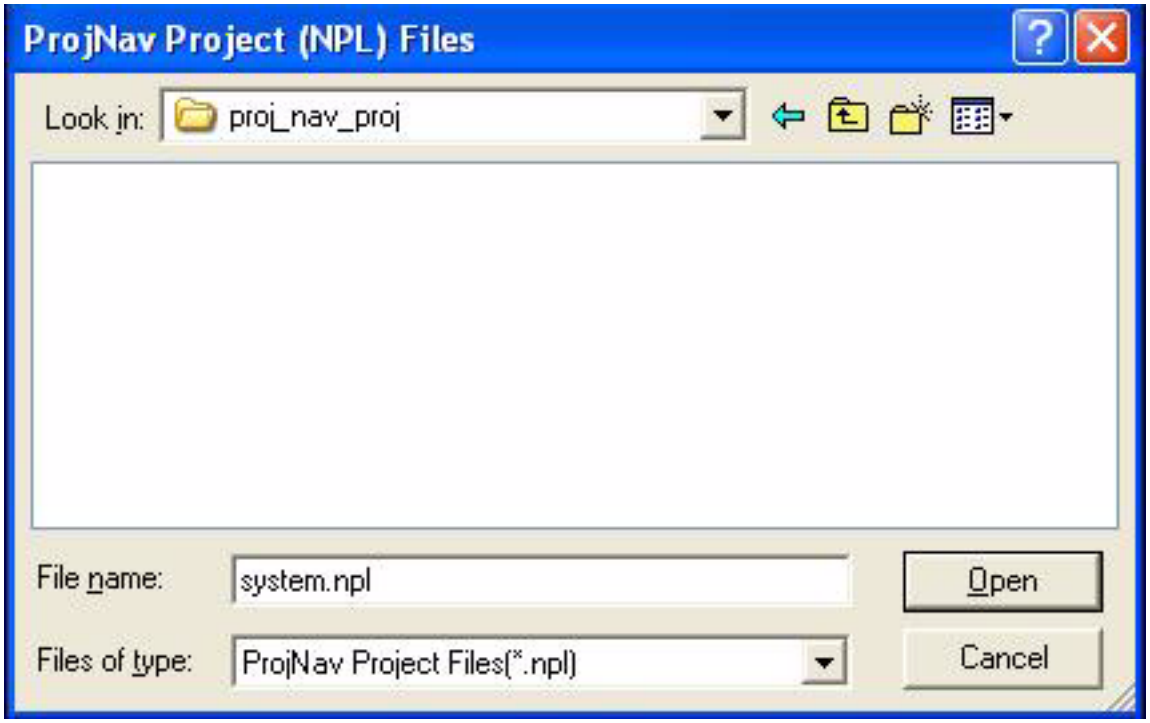

Figure 12: **Specifying NPL file**

b. Create a new directory named proj_nav_proj in the root XPS project directory by using the right mouse button and selecting **New** → **Folder** from the pop-up menu.

c. Select this directory and click **Open**.

Note: Verify that the Project Navigator project is created in the root directory to ensure that it is not deleted when you clean up the XPS project.

5. Click **OK**.
6. In XPS, select the following to create a Project Navigator project in the directory previously specified:

   Tools → **Export to ProjNav**

7. Open the Project Navigator project XPS just created.

Implementing the Tutorial Design

The Project Navigator project created by XPS does not contain all of the information necessary to implement the tutorial design. For example, the UCF file must be added to the project. You can add additional logic to the tutorial design using ISE.

To implement the design, follow these steps:

1. Select **Project** → **Add Source** to add the system.ucf file included in the data directory.
2. If the system_stub.bmm file has not been added to the project, select **Project** → **Add Source** to add the system_stub.bmm file in the implementation directory.
3. Select system_stub.vhd in the Source Window.
4. Right click on **Generate Programming File** in the Process Window and select **Properties...**
5. Under the **Startup options** tab, select JTAG Clock for FPGA Start-up Clock.
6. Click Ok.
7. Double click **Generate Programming File** in the Process Window to generate the uninitialized bit file.

Defining the Software Design

Now that the hardware design is completed, the next step is defining the software design. If you closed XPS, reopen it and load the project located in the edk_tutorial_ppc directory.

There are two major parts to software design, configuring the 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.

1. In XPS, select **Project→ Software Platform Settings**, this will open the Software Platform Settings dialog as shown in figure 13.

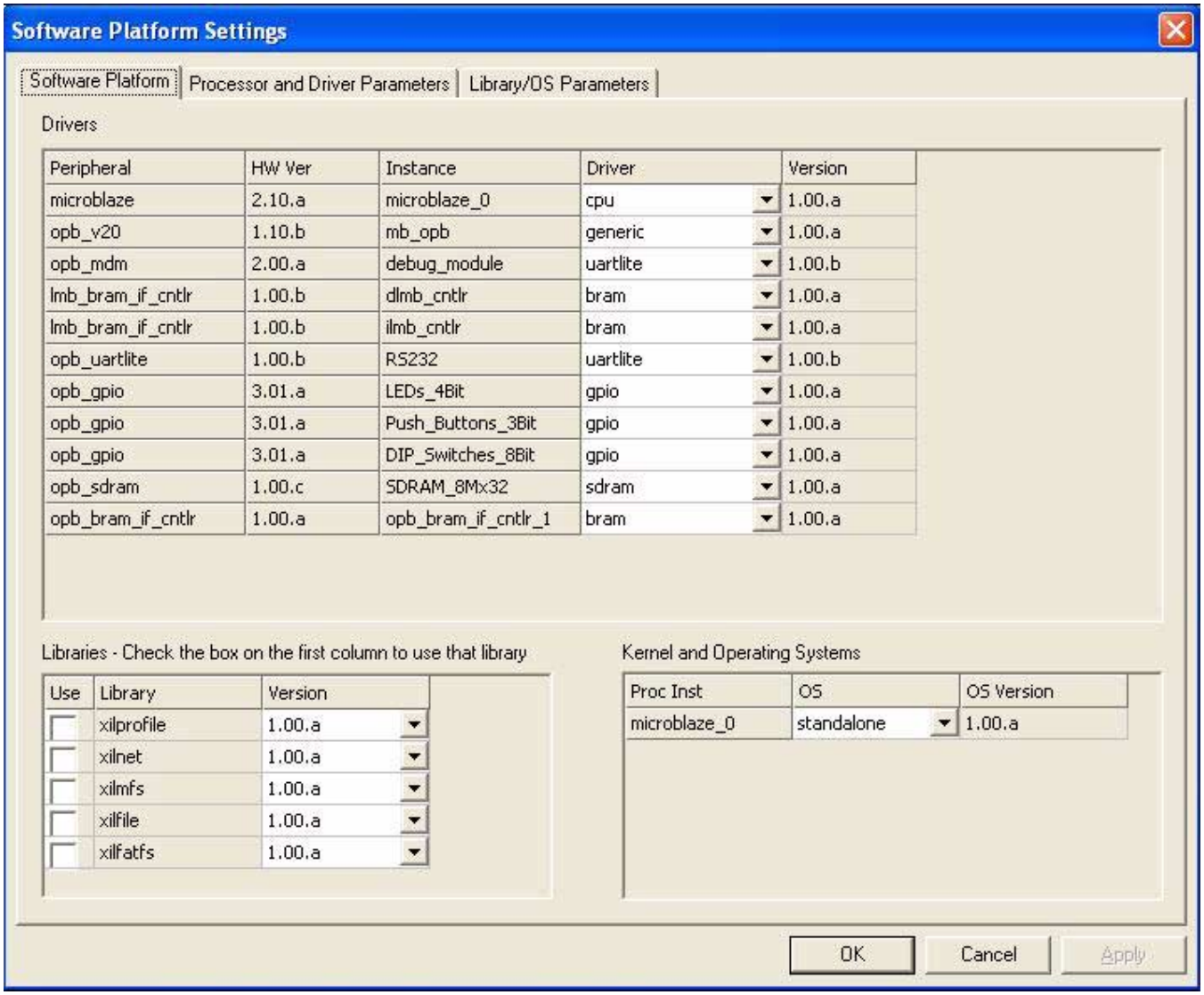

Figure 13: **Software Platform Settings Dialog**

The Software Platform Settings dialog box contains three tabs. Each of these tabs is used to control all aspects of the BSP creation.

2. 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

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

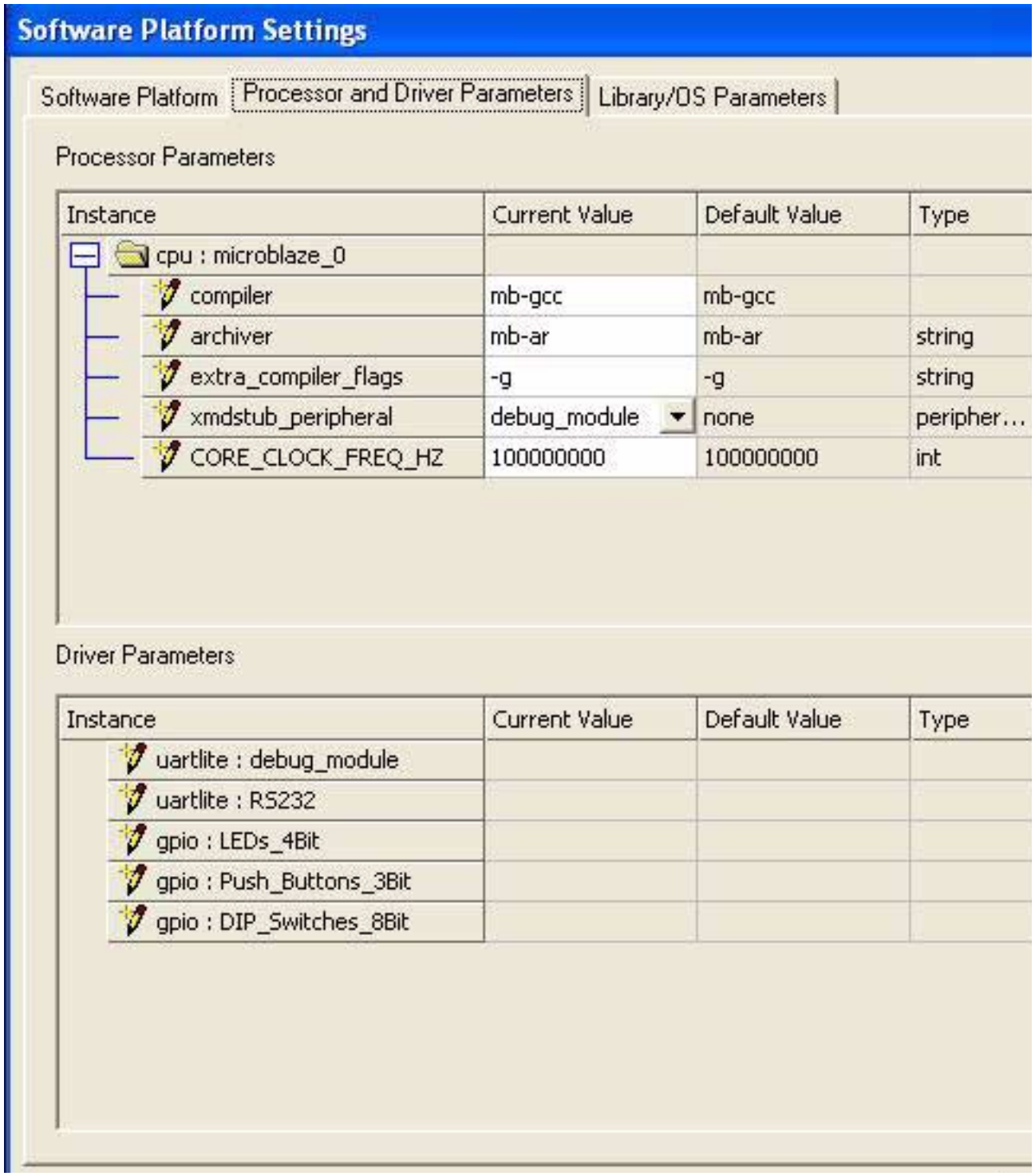

Figure 14: **Processor and Driver Parameters**

4. Select the Library / OS Parameters tab.

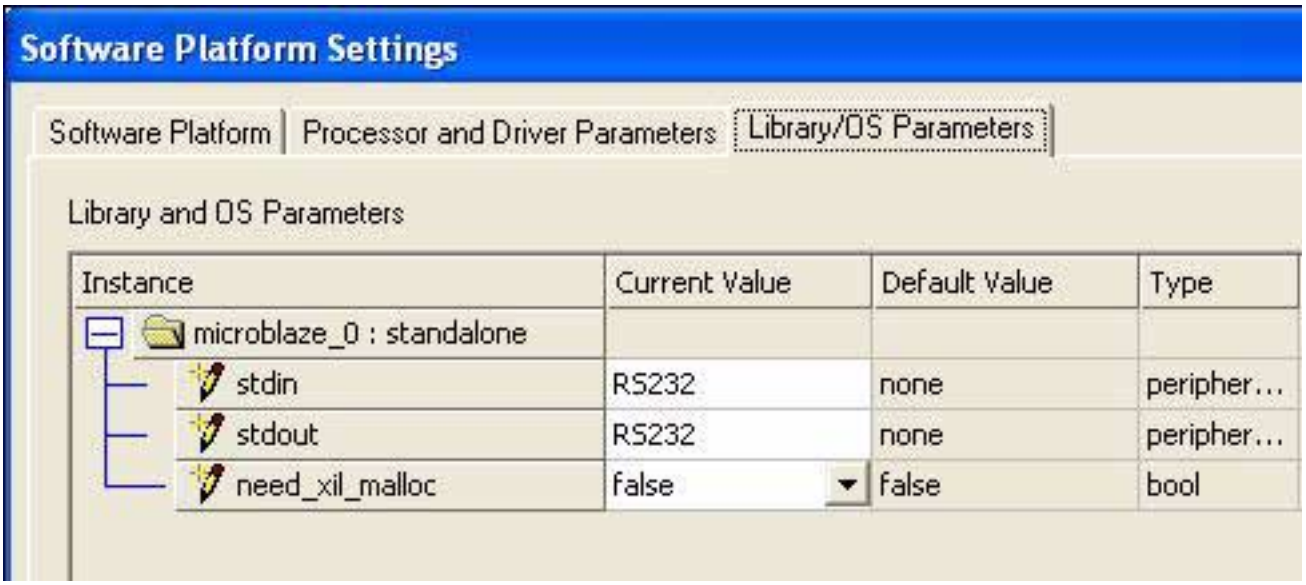

Figure 15: **Library/OS Parameters**

5. Configure the Library / OS Parameters tab as shown in figure 15. This allows the RS232 peripheral to be used for standard IO functions like print, printf, scanf, etc.
6. Click **OK**.
7. 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.
8. Libgen creates the following directories in the ppc405_0:
   - 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 Systems Tool Guide*.

Finishing the Tutorial Application Code

In EDK 6.2, 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 allows 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 16. The Base System Builder (BSB) generates a sample application which tests a subset of the peripherals included in the design.

Figure 16: **Applications Tab**

To complete the C program, follow these steps:

1. Double-click on the **`TestApp.c`** file under Sources to open the TestApp.c file in the editor window.
2. Examine the contents of the file. Notice that it performs basic read and write tests to the different GPIO ports.

3. You will now write to the GPIO utilizing another function.
4. Select the following:

   Start → **Programs** → **Xilinx Embedded Development Kit** → **EDK Documentation**

5. Select **Documents**.
6. Select **Xilinx Drivers** to open xilinx_drivers.pdf.
7. Click on **Driver Summary** at the top of the PDF file.
8. Scroll down until you find the General Purpose I/O section.
9. Click on the xgpio.h header file to open it.
10. When writing to the GPIO peripheral three functions are used to accomplish the following:
    a. Initialize the GPIO
    b. Specify the direction of the GPIO ports
    c. Write to the GPIO registers
11. Scroll down to locate the XGpio_Initialize function.
12. The documentation outlines two parameters:
    - InstancePtr is a pointer to a XGpio instance. The memory the pointer references must be pre-allocated by the caller. Further calls to manipulate the component through the XGpio API must be made with this pointer.
    - DeviceId is the unique id of the device controlled by this XGpio component. Passing in a device id associates the generic XGpio instance to a specific device, as chosen by the caller or application developer.

    With this information, return to the C code in XPS.
13. The first parameter you need to add is a pointer to an Xgpio instance. Note that a variable named gpio has been created. This variable is used as the first parameter in the Xgpio_Initialize function call. Add this variable to the function call. It should now look as follows:

    XGpio_Initialize(&gpio,
14. The second parameter is the device id for the device you want to initialize. This information is in the xparameters.h file. In XPS, select **File** → **Open**.
15. Browse to the edk_tutorial_mb\microblaze_0\include directory and select **xparameters.h**. The xparameters.h file is written by Libgen and provides critical information for driver function calls. This function call is used to initialize the GPIO used as an input for the dip switch found on the board.
16. In the xparameters.h file find the following #define used to identify the LEDS_4BIT peripheral:

    #define XPAR_LEDS_4BIT_DEVICE_ID 2

    Note: The “LEDS_4BIT” matches the instance name assigned in the MHS file for this peripheral.This is board dependant and will need to be specified based upon the board you are using.

    This #define can be used as the DeviceId in the function call.

17. Add the DeviceId to the function call so that it looks as follows:

    XGpio_Initialize(&gpio, XPAR_LEDS_48BIT_DEVICE_ID);

18. Using the documentation specify the direction of the GPIO data as an output.
19. Now that the peripheral has been initialized and the direction of the data has been specified, code a simple while loop to count from 1 to 16. Each value should be written to the GPIO.
20. An example of the function can be found below:

```
while(1) {

  XGpio_DiscreteWrite(&gpio, j);
     j++;
/* Insert some delay so you can see the LEDs changing */
  for (k = 0; k < 10000000; k++);

  if (j > 16)
    j = 0;
}
```

21. In order to use these new functions, an additional header file must be included, include the gpio.h header file.
22. The completed code shown below can be used as example:

```
/*
 * Xilinx EDK 6.2
 *
 * This file is a sample test application
 *
 * This application is intended to test and/or illustrate some
 * functionality of your system. The contents of this file may
 * vary depending on the IP in your system and may use existing
 * IP driver functions. These drivers will be generated in your
 * XPS project when you run the "Generate Libraries" menu item
 * in XPS.
 *
 * Your XPS project directory is at: C:\edk_tutorial_mb
 */

// Located in: microblaze_0/include/xparameters.h
#include "xparameters.h"
#include "xgpio_l.h"
#include "xutil.h"

#include "xgpio.h"

/*
 * Routine to write a pattern out to a GPIO
 * which is configured as an output
 *   PARAMETER C_ALL_INPUTS = 0
 */
void WriteToGPOutput(Xuint32 BaseAddress, int gpio_width) {
   int i=0, j=0, k=0;
   int numTimes = 5;
```

```
  XGpio_mSetDataDirection(BaseAddress, 0x00000000);  /* Set as outputs
*/
   while (numTimes > 0) {
      j = 1;
      for(i=0; i<(gpio_width-1); i++) {
         XGpio_mSetDataReg(BaseAddress, j);
         j = j << 1;
         for (k=0; k<10000000; k++) {
            ; //wait
         }
      }
      j = 1;
      j = ~j;
      for(i=0; i<(gpio_width-1); i++) {
         XGpio_mSetDataReg(BaseAddress, j);
         j = j << 1;
         for (k=0; k<10000000; k++) {
            ; //wait
         }
      }
      numTimes--;
   }
}

/*
 * Routine to read data from a GPIO
 * which is configured as an input
 *   PARAMETER C_ALL_INPUTS = 1
 */
Xuint32 ReadFromGPInput(Xuint32 BaseAddress) {
   Xuint32 data = XGpio_mGetDataReg(BaseAddress);
   return data;
}

//====================================================

int main (void) {

   XGpio gpio;
   int j = 0;
   int k = 0;

   print("-- Entering main() --\n");

   WriteToGPOutput(XPAR_LEDS_4BIT_BASEADDR, 4);
   {
      Xuint32 data = ReadFromGPInput(XPAR_PUSH_BUTTONS_3BIT_BASEADDR);
      xil_printf("Data read from Push_Buttons_3Bit: 0x%x\n", data);
   }

   {
      Xuint32 data = ReadFromGPInput(XPAR_DIP_SWITCHES_8BIT_BASEADDR);
      xil_printf("Data read from DIP_Switches_8Bit: 0x%x\n", data);
   }

   /* Testing SDRAM Memory (SDRAM_8Mx32)*/
   {
      XStatus status;
      xil_printf("Starting MemoryTest for SDRAM_8Mx32:\n");
```

```
      xil_printf("  Running 32-bit test...");
     status = XUtil_MemoryTest32((Xuint32*)XPAR_SDRAM_8MX32_BASEADDR,
1024, 0xAAAA5555, XUT_ALLMEMTESTS);
      if (status == XST_SUCCESS) {
         xil_printf("PASSED!\n");
      }
      else {
         xil_printf("FAILED!\n");
      }
      xil_printf("  Running 16-bit test...");
     status = XUtil_MemoryTest16((Xuint16*)XPAR_SDRAM_8MX32_BASEADDR,
2048, 0xAA55, XUT_ALLMEMTESTS);
      if (status == XST_SUCCESS) {
         xil_printf("PASSED!\n");
      }
      else {
         xil_printf("FAILED!\n");
      }
      xil_printf("  Running 8-bit test...");
      status = XUtil_MemoryTest8((Xuint8*)XPAR_SDRAM_8MX32_BASEADDR,
4096, 0xA5, XUT_ALLMEMTESTS);
      if (status == XST_SUCCESS) {
         xil_printf("PASSED!\n");
      }
      else {
         xil_printf("FAILED!\n");
      }
   }

   /* Testing BRAM Memory (opb_bram_if_cntlr_1)*/
   {
      XStatus status;
      xil_printf("Starting MemoryTest for opb_bram_if_cntlr_1:\n");
      xil_printf("  Running 32-bit test...");
      status =
XUtil_MemoryTest32((Xuint32*)XPAR_OPB_BRAM_IF_CNTLR_1_BASEADDR, 1024,
0xAAAA5555, XUT_ALLMEMTESTS);
      if (status == XST_SUCCESS) {
         xil_printf("PASSED!\n");
      }
      else {
         xil_printf("FAILED!\n");
      }
      xil_printf("  Running 16-bit test...");
      status =
XUtil_MemoryTest16((Xuint16*)XPAR_OPB_BRAM_IF_CNTLR_1_BASEADDR, 2048,
0xAA55, XUT_ALLMEMTESTS);
      if (status == XST_SUCCESS) {
         xil_printf("PASSED!\n");
      }
      else {
         xil_printf("FAILED!\n");
      }
      xil_printf("  Running 8-bit test...");
      status =
XUtil_MemoryTest8((Xuint8*)XPAR_OPB_BRAM_IF_CNTLR_1_BASEADDR, 4096,
0xA5, XUT_ALLMEMTESTS);
      if (status == XST_SUCCESS) {
         xil_printf("PASSED!\n");
```

```
        }
        else {
           xil_printf("FAILED!\n");
        }
    }

    XGpio_Initialize(&gpio, XPAR_LEDS_4BIT_DEVICE_ID);
    XGpio_SetDataDirection(&gpio, 0);

    while(1) {
     XGpio_DiscreteWrite(&gpio, j);
      j++;

     for (k = 0; k < 10000000; k++);

    if (j > 16)
     j = 0;

    }

    print("-- Exiting main() --\n");
    return 0;
}
```

23. Save and close the file.

Linker Scripts

A linker script is required to tell the GNU linker where to place the code. A linker script was written by the Base System Builder(BSB) using the information you provided when specifying the project for this tutorial.

1. Double click on TestAppLinkScr to open the linker script for the TestApp Project. Examine the contents of the file.

   Tutorial Test Question:

   At what address will the .text section be placed? _____
   At what address will the .data section be placed? _____

2. Save and close the file.

Compiling the Code

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

1. In XPS, select **Tools** → **Generate Libraries** to run libgen. Libgen compiles the drivers associated with this design.
2. Select **Tools** → **Compile Program Sources** to run mb-gcc. Mb-gcc compiles the source files.

   Tutorial Test Question:

   At what address has the application code been placed? _____

3. To answer this question, open a Xygwin shell:

   Start → **Programs** → **Xilinx Embedded Development Kit** → **Xygwin Shell**

4. In the Xygwin shell cd to the project directory and cd to the testapp directory.

5. Enter **`mb-objdump -d executable.elf > objdump`**. This command disassembles the executable contents of the executable.elf.
6. Using your favorite editor, open the objdump file you just created.

   Tutorial Test Questions:

   At what address has the application code been placed? _____

   Is there any physical memory at this address? _____
7. Close objdump.

Downloading the Design

Note: This section requires the Insight Virtex-II Pro Demonstration Board. For more information on this board, refer to the Insight Web Site at http://www.insight-electronics.com/

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

1. Connect the host computer to the target board, including connecting the Parallel 4 cable and the serial cable.
2. Start a hyperterminal session with the following settings:
   - **`com1`**
   - Bits per second: **`115200`**
   - Data bits: **`8`**
   - Parity: **`none`**
   - Stop bits: **`1`**
   - Flow control: **`none`**
3. Turn On the board power.
4. Turn all of the DIP switches on except number 1.
5. In XPS, select **`Tools`** → **`Import from ProjNav…`**
6. Select system_stub.bit file in the proj_nav_proj directory.
7. Select system_stub_bd.bmm in the implementation directory.
8. Click **`OK`**.
9. Select **`Tools`** → **`Download`** to create a new bit file that has been updated with the recently compiled code. iMPACT is used to configure the device.

10. Once the device is configured, the hyperterminal should look like the following figure:

Figure 17: **Hyperterminal Output**

11. As the message states, data was read from the DIP switch, the push button switches and then several memory tests were performed.

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 18. 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.

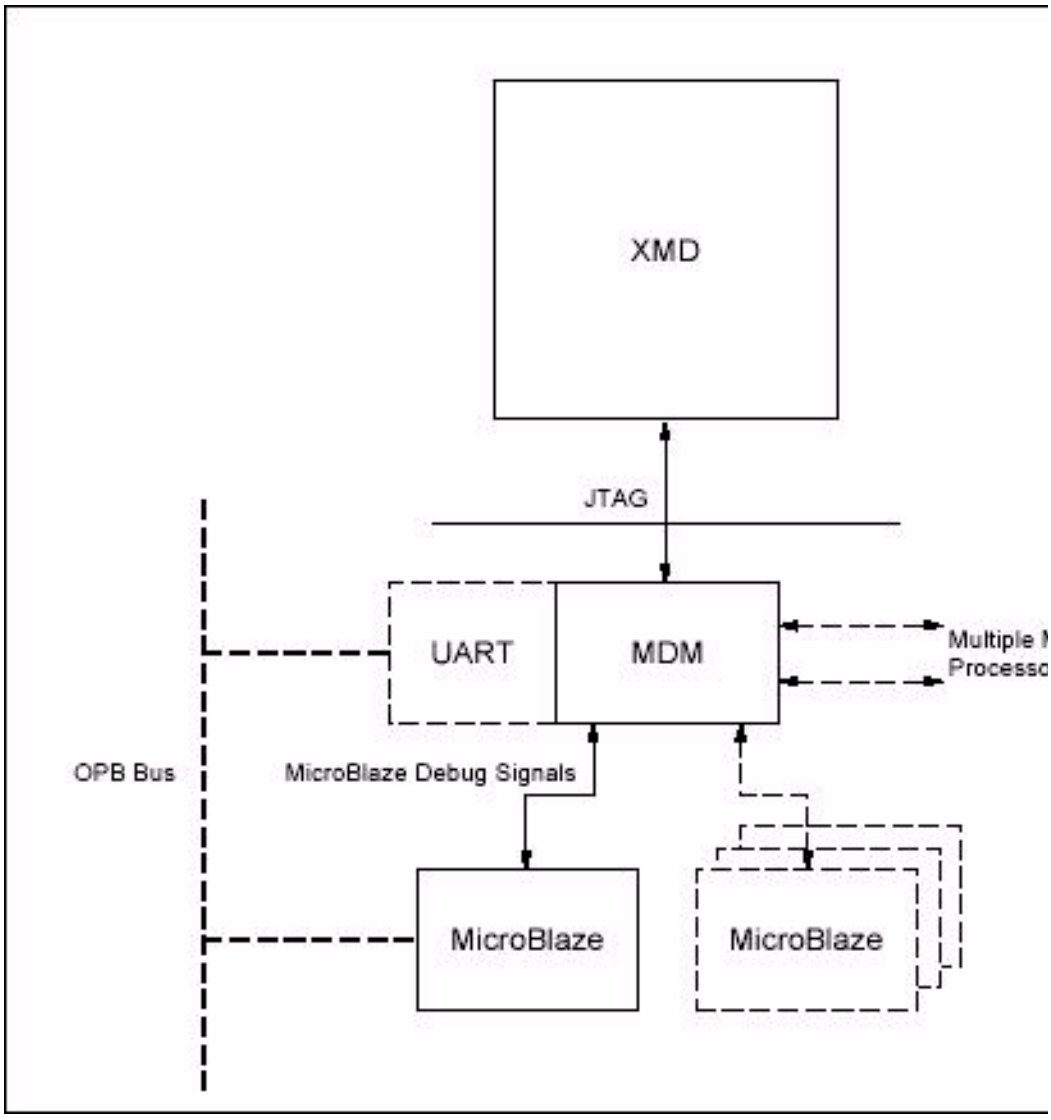

Figure 18: **XMD, MDM and MicroBlaze Connections**

To debug the design, follow these steps:

1. Select **Tools** → **XMD**.
2. After xmd has initialized, enter the following:

   ```
   mbconnect mdm
   ```

3. In XPS, select **Tools** → **Software Debugger** to open the GDB interface.
4. In GDB, select **File** → **Target Settings** to display the Target Selection dialog box as shown in the following figure:

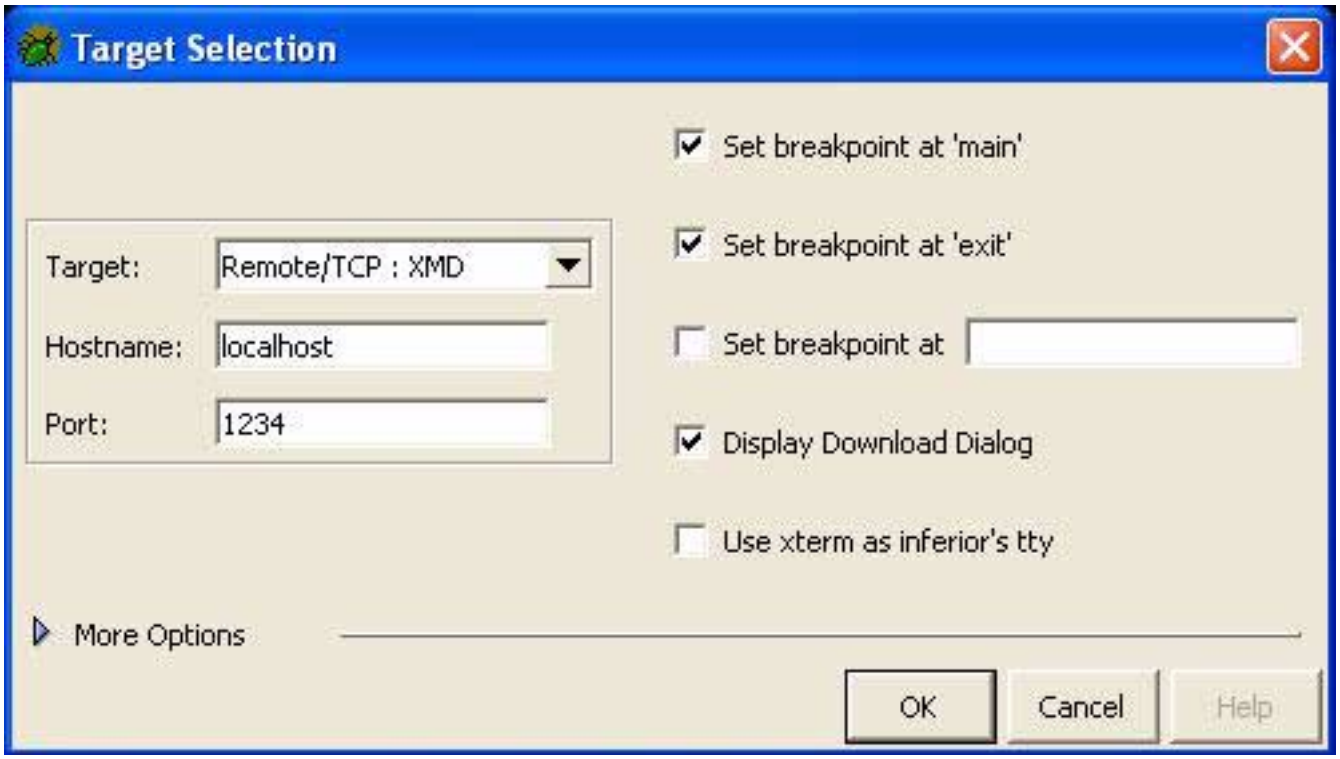

Figure 19: **Target Selection Dialog**

5. Configure the Target Selection dialog box to match figure 19. Click **Ok**.
6. In GDB, select **File** → **Open File**.
7. Select executable.elf in the testapp directory.

   Tutorial Test Questions:

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

   Why can you not see the C code? _____
8. In GDB, select **File** → **Exit**.
9. In XPS, select the Applications tab. Double-click on **Project: TestApp** .
10. Select the Optimization tab.
11. Select **Create symbols for debugging (-g option)**.
12. Click **OK**.
13. Perform the following steps:
    - 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 steps 9-13.
14. Select **Run** → **Run**

There is an automatic breakpoint at main. GDB allows you to single step the C or assembly code. This is an exercise to help you learn how to run GDB.

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

15. Once you have determined the error, recompile the code and download it through GDB.

Simulating the Design

Simulation allows you to verify the hardware and software. XPS provides integration with the SimGen (Simulation Model Generation) tool that generates and configures various simulation models for a specified hardware system. SimGen supports behavioral (VHDL), structural, and timing simulation models. This section of the tutorial demonstrates behavioral VHDL simulation.

Note: When performing a simulation, you must be aware of the components in the design. For example, the simulation of a UART peripheral could take several hundred microseconds depending on the UART baud rate. For the purpose of this tutorial, we recommend that you comment out all references to the UART before generating the simulation model.

To simulate the design, follow these steps:

1. In XPS, select **Options** → **Project Options**. Select the **HDL and Simulation** tab as shown in Figure 20.

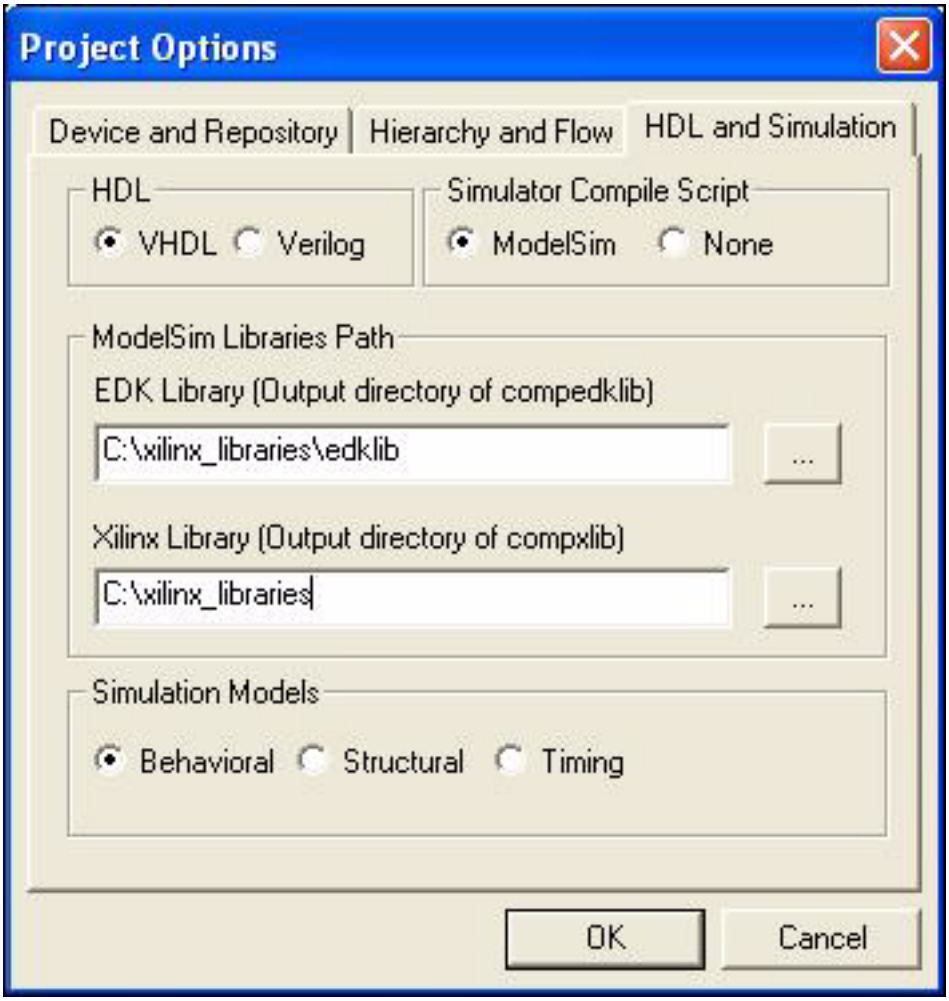

Figure 20: **HDL and Simulation Tab**

2. Using the browse button, specify the path to the EDK Library and the Xilinx Library. For additional information on creating these libraries, refer to the "Getting Started with EDK" documentation.
3. Click **Ok**.

4. Select **Tools** → **Sim Model Generation** to generate a simulation model. This generates the following files in the simulation\behavioral directory:
   - system.do – compiles the simulation source files
   - system_init.vhd – VHDL used to initial BRAMs with the software application
   - system_sim.bmm – BMM file used to generate system_init.vhd

   These files are used to perform the simulation.
5. Open the ISE project in the proj_nav_proj directory.
6. Select **Project** → **New Source** and select VHDL Testbench. The file name should be testbench. Click **Next**.
7. Associate the testbench.vhd with system_stub.vhd. Click **Next** and **Finish**.
8. The created testbench is opened in the Editor Window. This testbench contains no stimulus. Delete the template and add the following processes to provide stimulus to the processor system.

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

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

RS232_RX <= '0';
Push_Buttons_3Bit_GPIO_in <= "001";
DIP_Switches_8Bit_GPIO_in <= "00100010";
```

9. Save the testbench.vhd file.
10. Open the system_init.vhd file located in the simulation\behavioral directory.
11. The system_init.vhd contains the BRAM initialization strings. These strings represent the software application code being stored in the BRAMs. At the end of the system_init.vhd file, you will find the configuration statements which loads the executable contents, written as initialization strings, into BRAM for simulation. This configuration string does not include the full design hierarchy and needs to be modified and included in the testbench. The following configuration must be included at the end of the testbench.vhd:

```
-- Example
--configuration conf_name of entity_name is
--  for arch_name
--    for inst_name : component_name

-- ***** Insert Configuration Statement Here ******

configuration testbench_conf of system_stub_testbench_vhd_tb is
  for behavior
    for uut : system_stub
        for STRUCTURE
       for u1 : system
```

```
          for STRUCTURE
              ffor all : lmb_bram_wrapper use configuration
work.lmb_bram_conf;
          end for;
          for all : opb_bram_if_cntlr_1_bram_wrapper use configuration
work.opb_bram_if_cntlr_1_bram_conf;
          end for;
          end for;
        end for;
         end for;
      end for;
    end for;
end testbench_conf;
```

You will notice that the “for all” statements match the “for all” statements found in the system_init.vhd file.

12. Save and close the testbench.vhd file
13. Select **File** → **New** to create a new do file.
14. Select **File** → **Save As**, to save the file as projnav.do in the proj_nav_proj directory.
15. Add the following lines to the projnav.do:

```
cd ../simulation/behavioral
do system.do
vcom -93 -work work system_stub.vhd
vcom -93 -work work ../../proj_nav_proj/testbench.vhd
vsim -Lf unisim -t ps +notimingchecks work.testbench_conf
add wave *
```

16. Select **File** → **Save.** In the Process Window, right click Simulate Behavioral VHDL Model and select **Properties**.
17. In the Simulation Properties Tab, set Custom Do File to the projnav.do in the proj_nav_proj directory.
18. Uncheck the **Use Automatic Do File** option.
19. Click **OK** to close the dialog box.
20. Double click Simulate Behavioral VHDL Model to start the simulation.
21. In the Modelsim window enter the following command:

```
run 100us
```