EDK MicroBlaze Tutorial

EDK MicroBlaze Tutorial

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.

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:

System Requirements

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

- Windows 2000 SP2 / Windows XP
- EDK 3.2 Service Pack 1or later
- ISE 5.2i SP1

Accessing the Tutorial Design Files

To access the tutorial design files, unzip edk_tutorial_mb.zip into the directory of your choice.

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 tutorial design includes the following hardware components:

- MB
- LMB Bus
  - LMB_LMB_BRAM_IF_CNTLR
  - BRAM_BLOCK
- OPB BUS
  - OPB_GPIO
  - OPB_BRAM_IF_CNTLR
  - OPB BRAM
  - OPB_UARTLITE

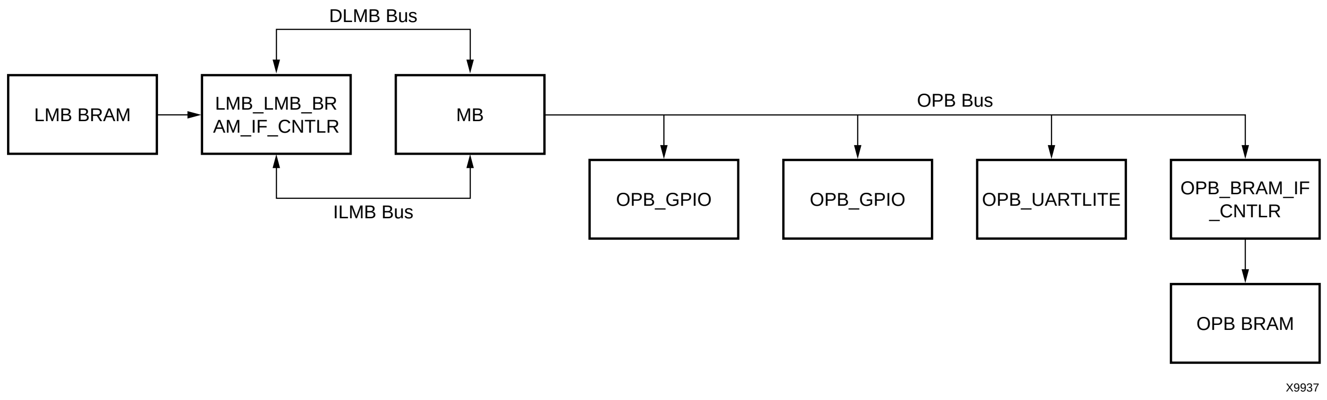


Figure 1: **Tutorial Design Hardware Components**

Tutorial Design Memory Map

The following table shows the memory map for the tutorial design:

Table 1: **Tutorial Design Memory Map**

Device	Address		Size	Comment
	Min	Max		
LMB_BRAM	0x0000_0000	0x0000_3FFF	16kB	LMB Memory
OPB_GPIO1	0xFFFF_4300	0xFFFF_43FF	256B	DIP Switch Input
OPB_GPIO0	0xFFFF_4200	0xFFFF_42FF	256B	LED Output
OPB_UARTLITE	0xFFFF_4500	0xFFFF_45FF	256B	Serial Output
OPB_BRAM	0xFFFF_0000	0xFFFF_3FFF	16kB	OPB Memory

Completing 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 (MSS) file

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

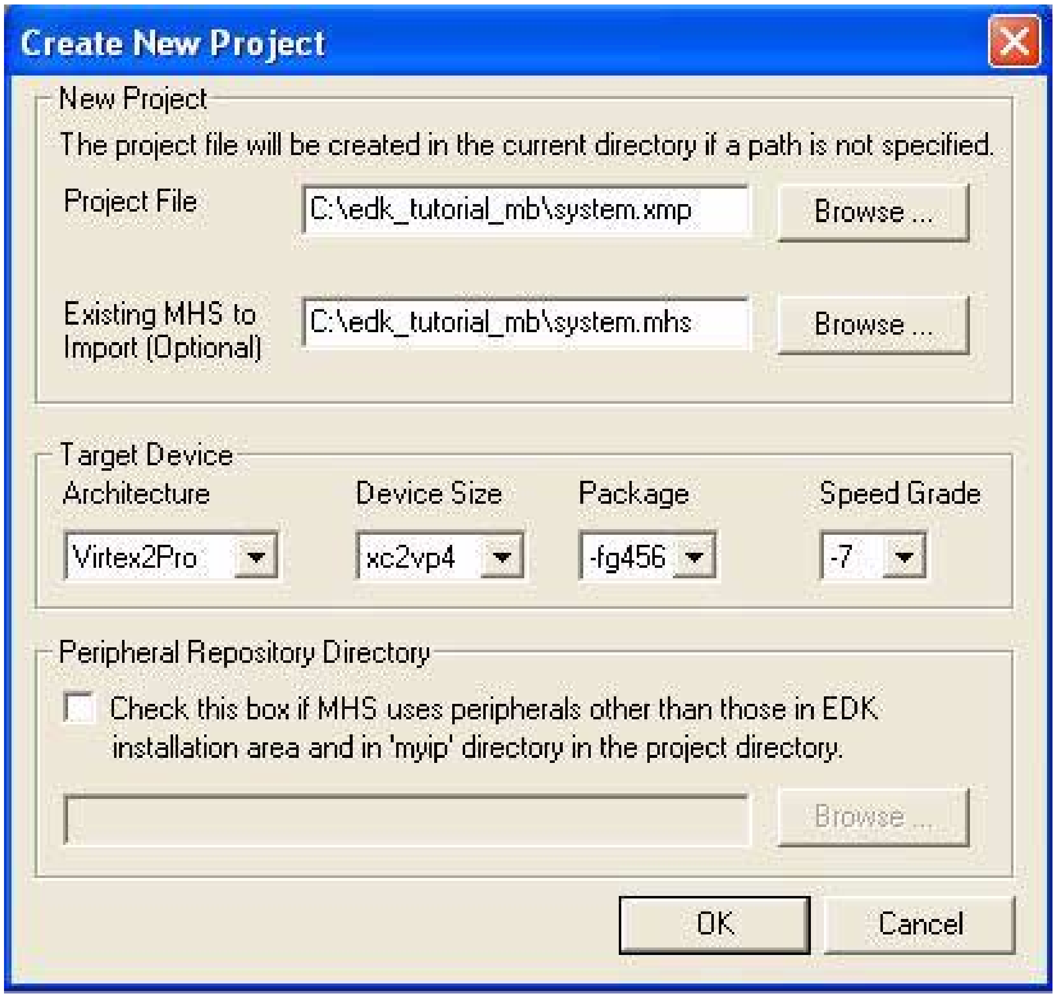
Note: For more information on the MVS file, refer to the "Microprocessor Verification Specification" chapter in the *Embedded Systems Tool Guide*.

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`** to open the Create New Project dialog box shown in the following figure:



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

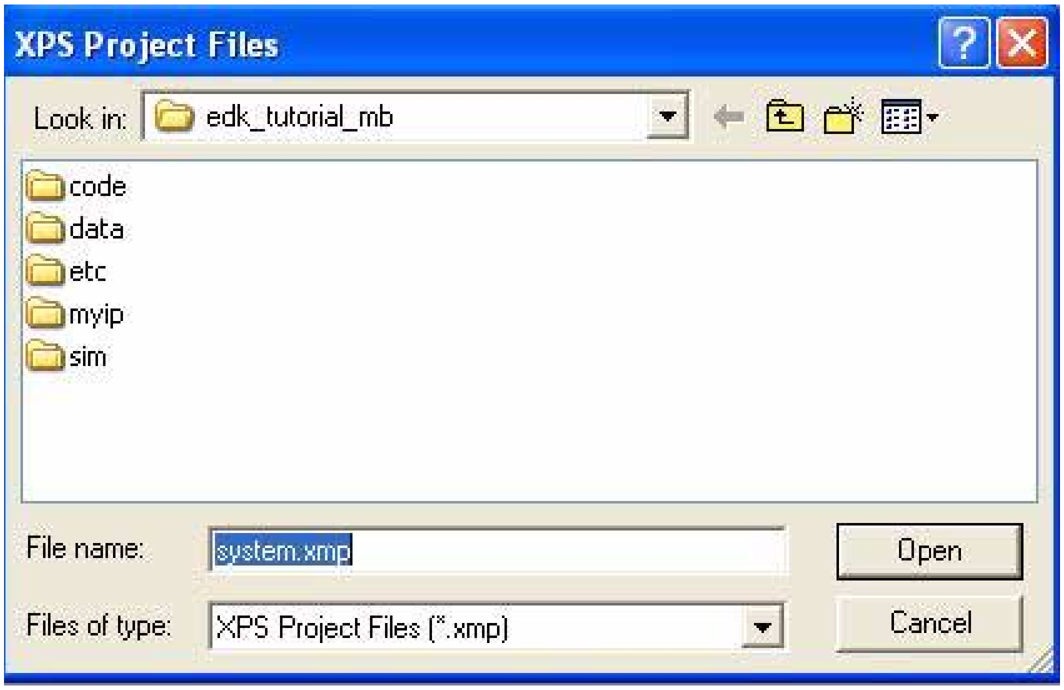
Project File	C:\edk_tutorial_mb\system.xmp
Existing MHS to Import (Optional)	C:\edk_tutorial_mb\system.mhs

Architecture	Device Size	Package	Speed Grade
Virtex2Pro	xc2vp4	-fg456	-7

[ ] Check this box if MHS uses peripherals other than those in EDK installation area and in 'myip' directory in the project directory.

Figure 2: **Create New Project Dialog Box**

3. Use the Project File Browse button to browse to the edk_tutorial_mb folder shown in the following figure. Click **Open** to create the system.xmp file.



File name: system.xmp

Files of type: XPS Project Files (*.xmp)

Figure 3: **XPS Project Files Directory**

4. Use the MHS File to Import Browse button to select the system.mhs file.
5. Set the Target Device to the following:
   - Architecture: Virtex2Pro
   - Device Size: xc2vp4
   - Package: -fg456
   - Speed Grade: -7
6. Click **OK** to create the project.

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:

$EDK / hw / iplib / 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*.

Updating the Tutorial MHS File

The EDK Platform Specification Utility (PsfUtil) allows you to build the MHS file. You can use one of the following modes to run this utility:

- Graphical dialog mode

  In this mode, you can describe the embedded hardware system using graphical selections.

- Textual mode

  In this mode, you can add templates for each peripheral to the MHS file and then manually modify the MHS file.

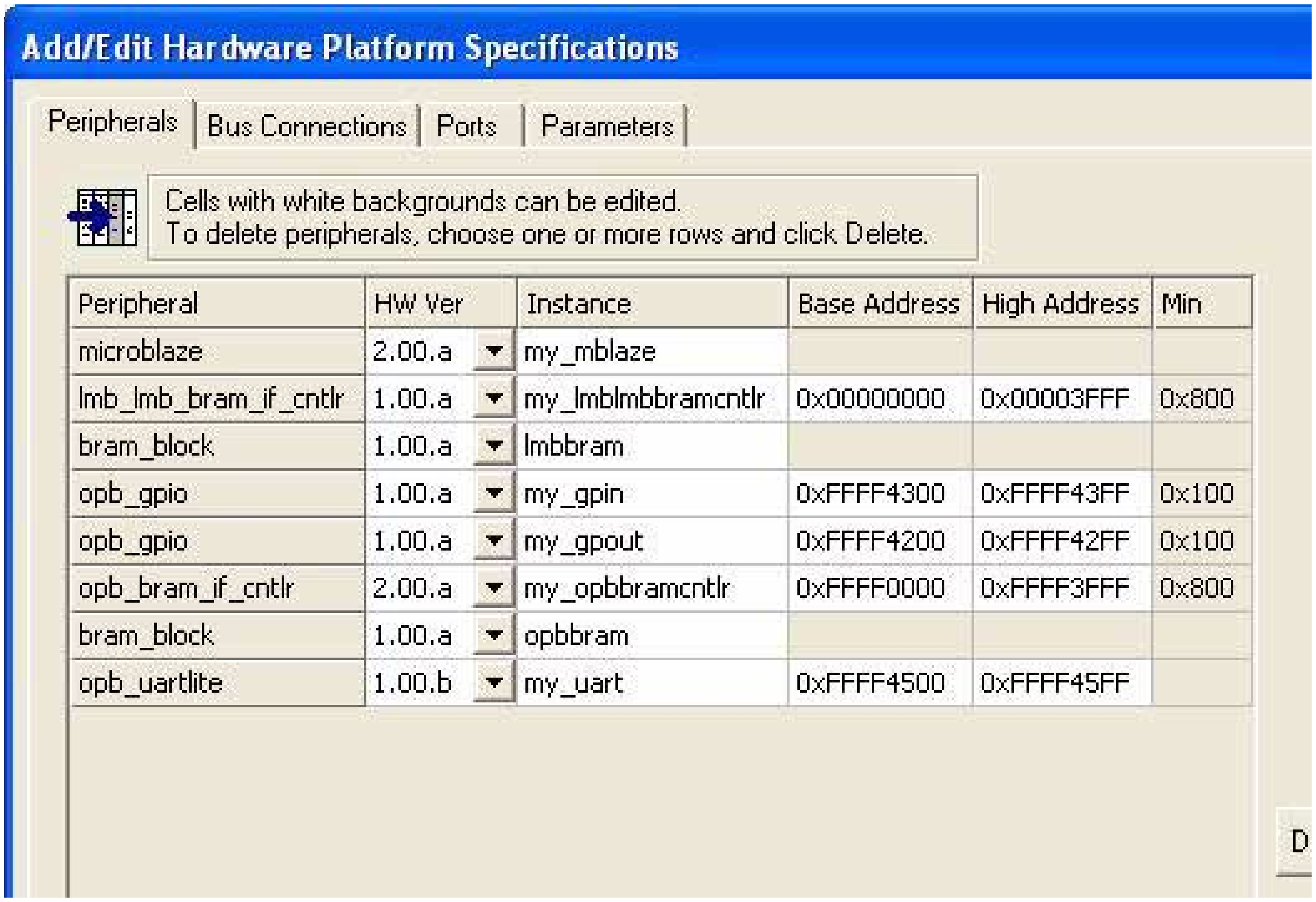
Using the Graphical Dialog Mode

Use the following steps to add peripheral base addresses and external ports to the tutorial MHS file:

1. Open the graphical core tool by selecting the following:

   `Project → Add/Edit Cores`

   The System Settings dialog box appears as shown in the following figure. The peripherals should already be added to the design.



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

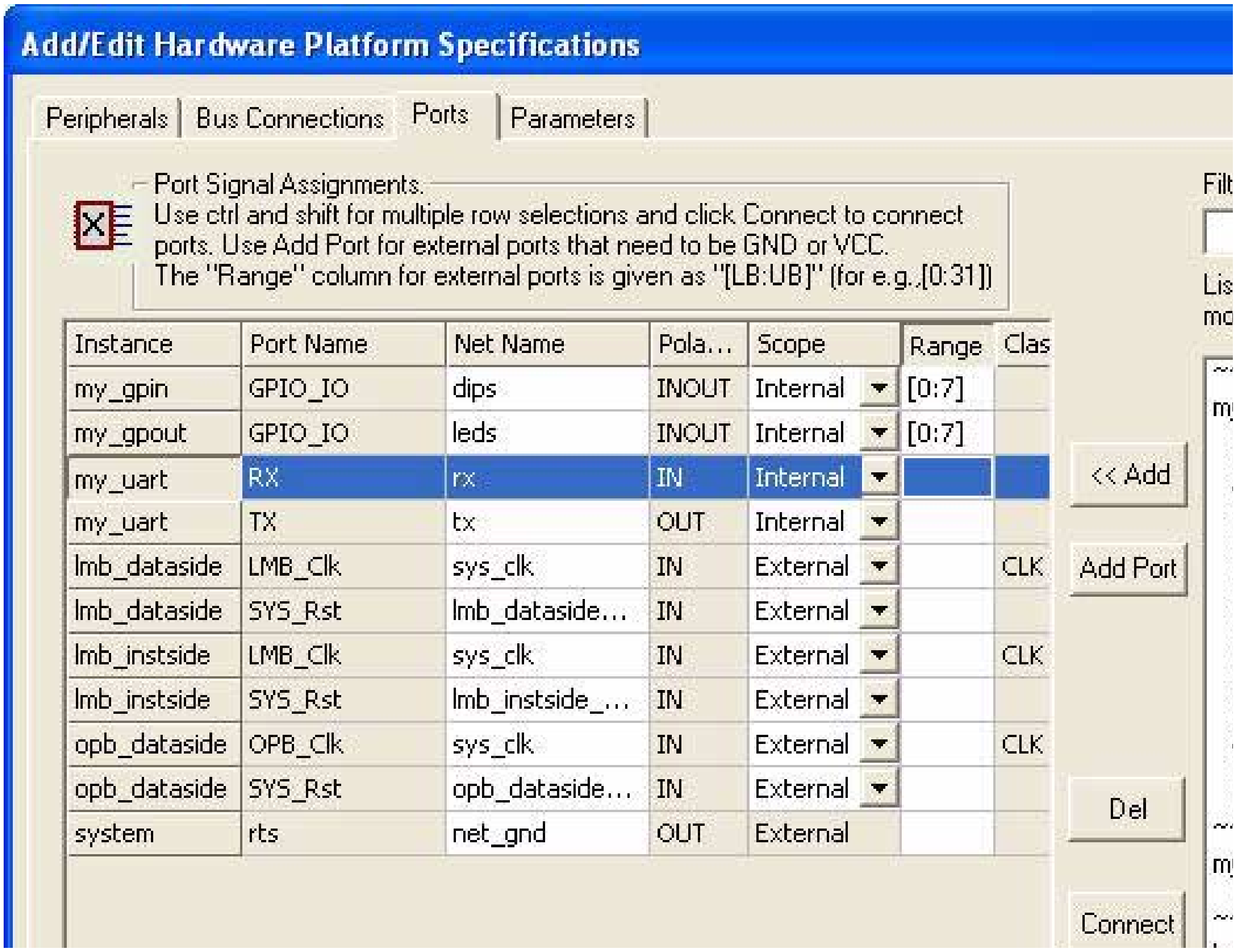
Peripheral	HW Ver	Instance	Base Address	High Address	Min
microblaze	2.00.a	my_mblaze			
lmb_lmb_bram_if_cntlr	1.00.a	my_lmblmbbramcntlr	0x00000000	0x00003FFF	0x800
bram_block	1.00.a	lmbbram			
opb_gpio	1.00.a	my_gpin	0xFFFF4300	0xFFFF43FF	0x100
opb_gpio	1.00.a	my_gpout	0xFFFF4200	0xFFFF42FF	0x100
opb_bram_if_cntlr	2.00.a	my_opbbramcntlr	0xFFFF0000	0xFFFF3FFF	0x800
bram_block	1.00.a	opbbram			
opb_uartlite	1.00.b	my_uart	0xFFFF4500	0xFFFF45FF	

Figure 4: **System Settings**

2. Modify the peripheral Base Address and High Address settings to match the settings in Figure 4 and in the memory map in Table 1. To modify the address values, double-click the white box and type in the address value in hexadecimal format.

Note: The following message will be displayed when the ports tab is selected, “These cores (instance, version) are deprecated cores. We recommend that you choose a core/version that is not deprecated my_lmblmbbramcntlr, 1.00.a Continue? Press yes to continue using the deprecated cores” Click **Yes**.

3. Select the Ports tab to display the Ports settings as shown in the following figure:



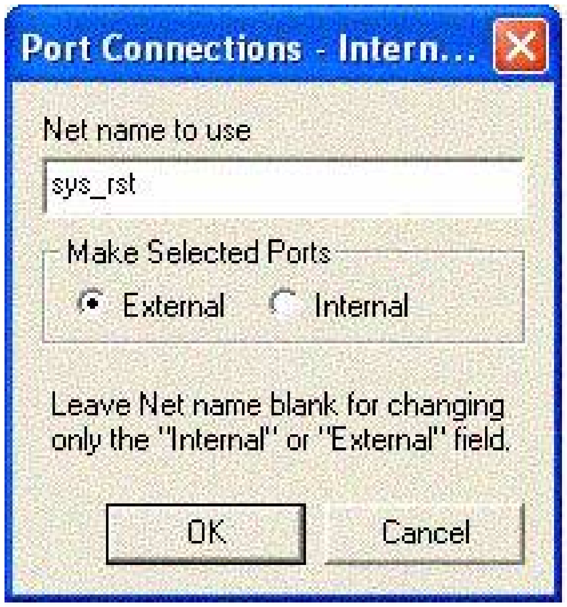
Port Signal Assignments.
Use ctrl and shift for multiple row selections and click Connect to connect ports. Use Add Port for external ports that need to be GND or VCC.
The "Range" column for external ports is given as "[LB:UB]" (for e.g.,[0:31])

Instance	Port Name	Net Name	Pola...	Scope	Range	Clas
my_gpin	GPIO_IO	dips	INOUT	Internal	[0:7]	
my_gpout	GPIO_IO	leds	INOUT	Internal	[0:7]	
my_uart	RX	rx	IN	Internal		
my_uart	TX	tx	OUT	Internal		
lmb_dataside	LMB_Clk	sys_clk	IN	External		CLK
lmb_dataside	SYS_Rst	lmb_dataside...	IN	External		
lmb_instside	LMB_Clk	sys_clk	IN	External		CLK
lmb_instside	SYS_Rst	lmb_instside_...	IN	External		
opb_dataside	OPB_Clk	sys_clk	IN	External		CLK
opb_dataside	SYS_Rst	opb_dataside...	IN	External		
system	rts	net_gnd	OUT	External		

Figure 5: **Ports Settings**

4. There are three reset signals for this system: OPB (**opb_dataside_SYS_Rst**), LMB (**lmb_dataside_SYS_Rst**), and LMB (**lmb_instside_SYS_Rst**). Connect the reset

signals by pressing the **CTRL** key and selecting each reset signal, and then pressing **Connect**. The Port Connections dialog box appears as shown in the following figure:


Port Connections - Intern...

Net name to use: sys_rst

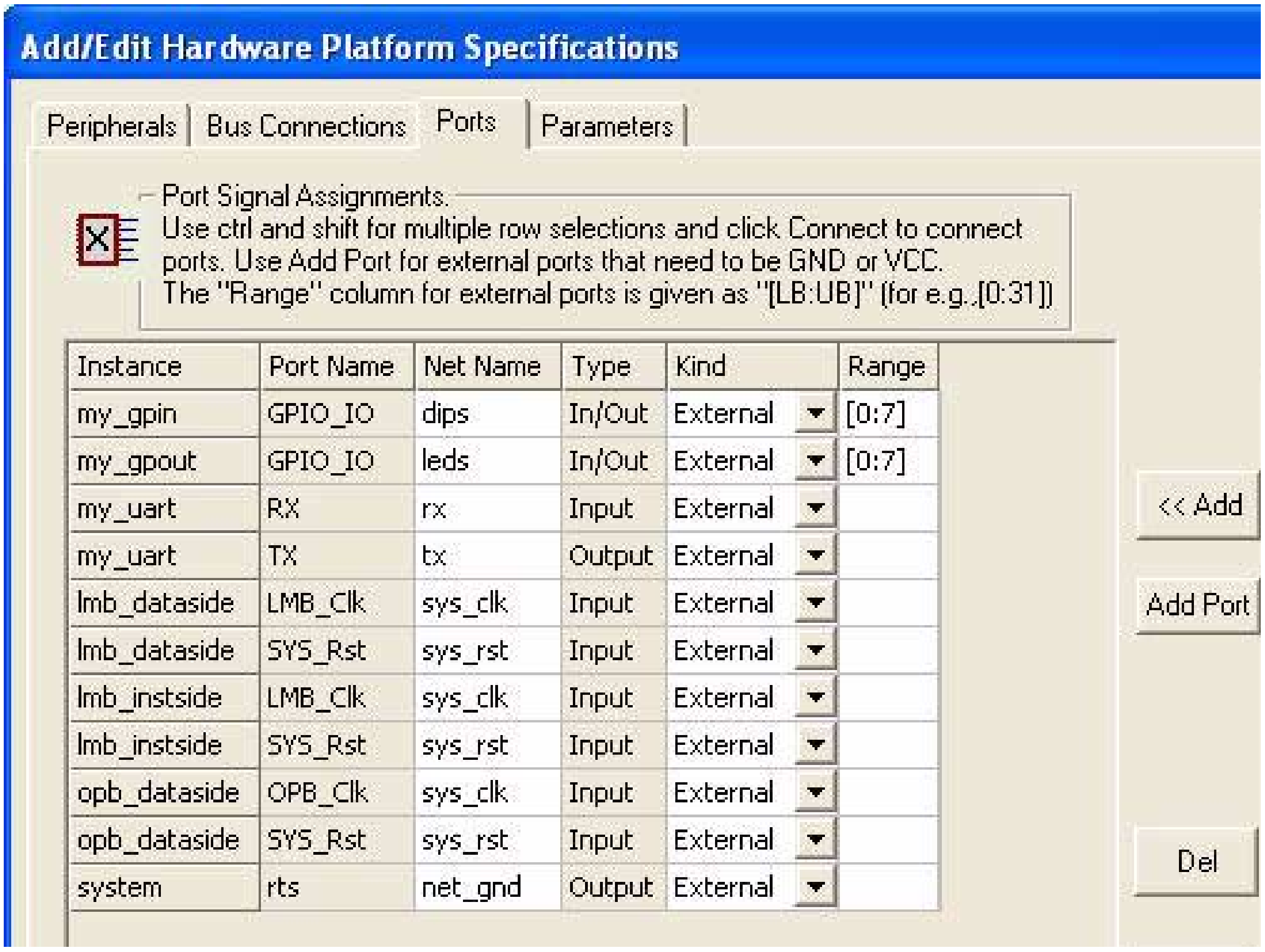
Make Selected Ports: [✓ External] Internal

Leave Net name blank for changing only the "Internal" or "External" field.

Figure 6: **Port Connections Dialog Box**

5. Enter **sys_rst** in the Net Name to Use field and select **External** for the type of port. Click **OK**.
6. Specify the RX, TX, dips, and leds nets as external signals using the pull-down menu in the Kind field.

7. Change the size of each of the GPIO buses to 8 as shown in the following figure:



Port Signal Assignments
Use ctrl and shift for multiple row selections and click Connect to connect ports. Use Add Port for external ports that need to be GND or VCC.
The "Range" column for external ports is given as "[LB:UB]" (for e.g.,[0:31])

Instance	Port Name	Net Name	Type	Kind	Range
my_gpin	GPIO_IO	dips	In/Out	External	[0:7]
my_gpout	GPIO_IO	leds	In/Out	External	[0:7]
my_uart	RX	rx	Input	External	
my_uart	TX	tx	Output	External	
lmb_dataside	LMB_Clk	sys_clk	Input	External	
lmb_dataside	SYS_Rst	sys_rst	Input	External	
lmb_instside	LMB_Clk	sys_clk	Input	External	
lmb_instside	SYS_Rst	sys_rst	Input	External	
opb_dataside	OPB_Clk	sys_clk	Input	External	
opb_dataside	SYS_Rst	sys_rst	Input	External	
system	rts	net_gnd	Output	External	

Figure 7: **Port Connections**

8. Click **Apply** then **OK** to update these values in the MHS file.
9. Open the MHS file by double clicking on “system.mhs” in the System tab to verify these changes.

Adding Additional IP or Hardware to an Embedded System

You can use the following methods to add IP or hardware to an embedded system:

- Add IP or hardware to the MHS file
- Instantiate the embedded system in a top-level wrapper file

To incorporate additional IP or hardware into an embedded system, you must maintain a strict directory structure. The following figure shows a depiction of the tutorial design peripheral directory structure.

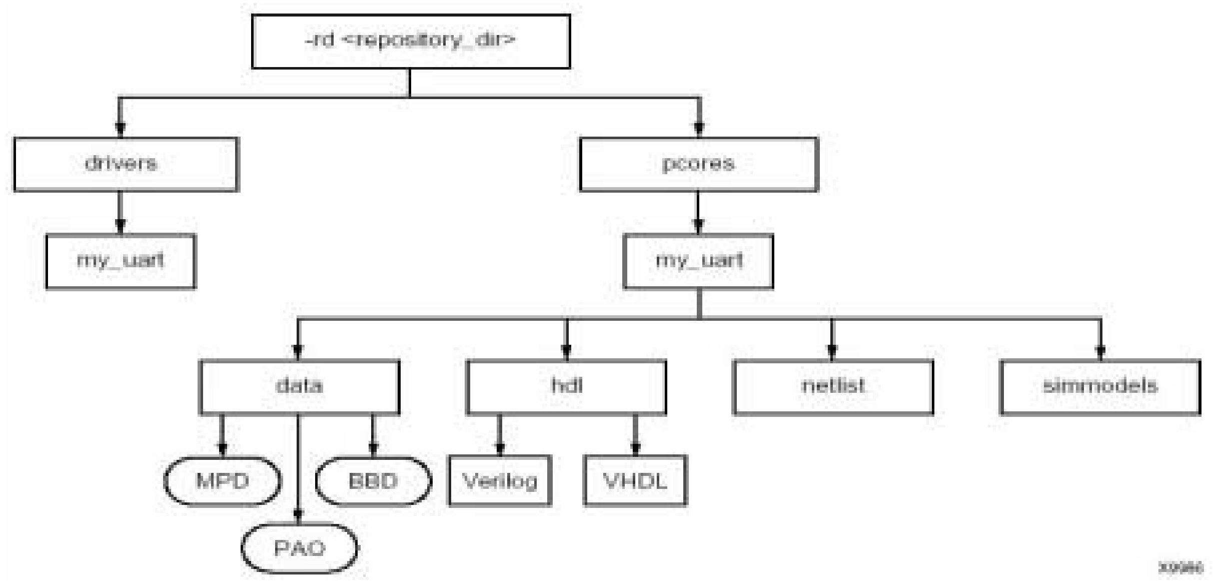


Figure 8: **Peripheral Directory Structure**

Platform Generator uses the following search priority mechanism to locate peripherals:

- Search the pcores directory located in the project directory
- Search <repository_dir>\pcores as specified by the -rd option
- Search $XILINX_EDK/hw/iplib/pcores and $XILINX_EDK/hw/edklib/pcores (UNIX) or %XILINX_EDK%\hw\iplib\pcores and %XILINX_EDK%\hw\edklib\pcores (PC)

Generating a Netlist and Creating a Project Navigator Project

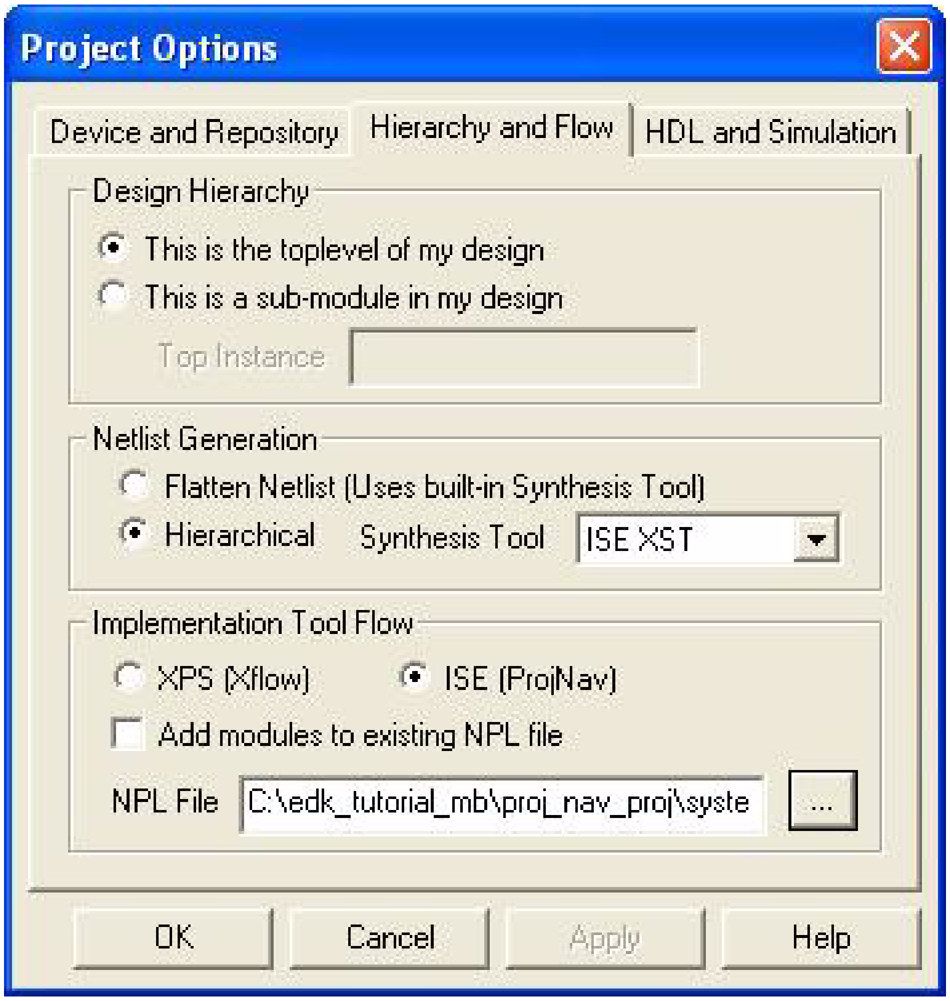
Now that the hardware has been completely specified in the MHS file, you can run the Platform Generator. The Platform Generator elaborates the MHS file into a hardware system consisting of NGC files that represent the processor system.

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`**

   The following dialog box is displayed:



Design Hierarchy
- [✓ This is the toplevel of my design]
- This is a sub-module in my design

Top Instance

Netlist Generation
- Flatten Netlist (Uses built-in Synthesis Tool)
- [✓ Hierarchical] Synthesis Tool: ISE XST

Implementation Tool Flow
- XPS (Xflow) [✓ ISE (ProjNav)]
- Add modules to existing NPL file

NPL File: C:\edk_tutorial_mb\proj_nav_proj\syste

Figure 9: **Project Options**

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

   Design Hierarchy: **`This is the top level of my design`**

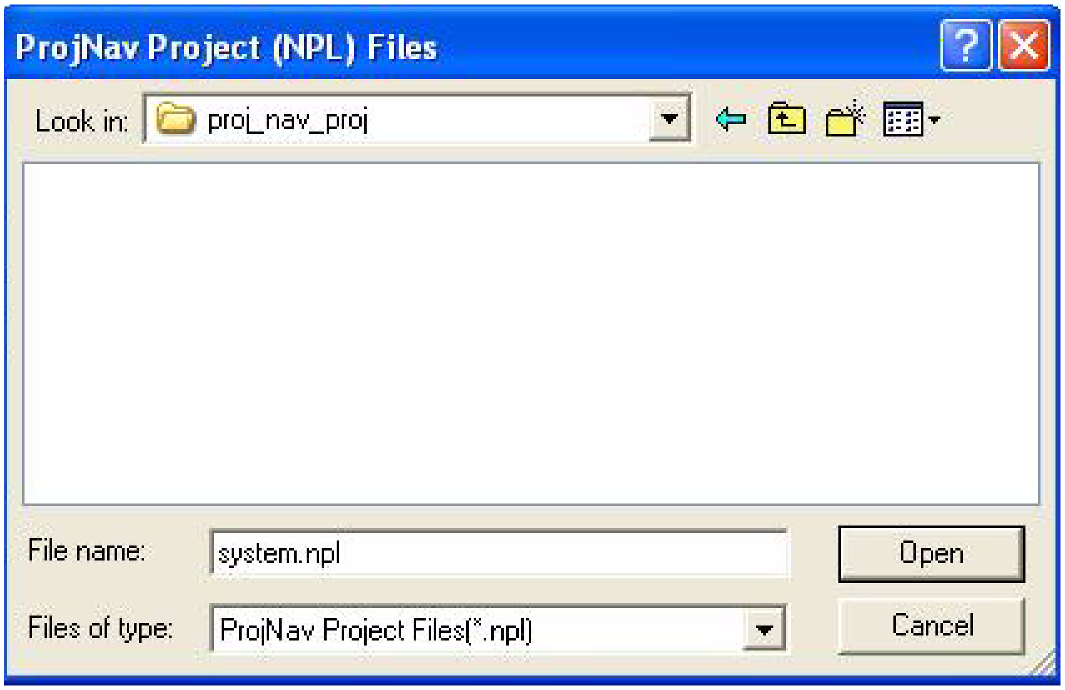
   Netlist Generation: **`Hierarchical`**

   Synthesis Tool: **`ISE XST`**

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

In the NPL File field, follow these steps:

a. Click the "…" button to open the following figure:



File name: system.npl

Files of type: ProjNav Project Files(*.npl)

Figure 10: **ProjNav Project (NPL) Files**

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.

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. Since all of the files currently included in the project are machine generated, any changes to the MHS file will result in these files being regenerated. For this reason, double click on system.vhd in the Source Window to open it, and select **File** → **Save As** to save this file in the proj_nav_proj directory as top.vhd:
2. Remove system.vhd in the Source Window.

3. Select **`Project`** → **`Add Source`** to add top.vhd to the Project Navigator project.
4. Select **`Project`** → **`Add Source`** to add the system.ucf file in the data directory.
5. Select **`Project`** → **`Add Source`** to add the bram_init.bmm file in the implementation directory.
6. Select top.vhd in the Source Window.
7. Right click on **Generate Programming File** in the Process Window and select **Properties...**
8. Under the **Startup options** tab, select JTAG Clock for FPGA Start-up Clock.
9. Click Ok.
10. Double click **`Generate Programming File`** in the Process Window to generate the uninitialized bit file.

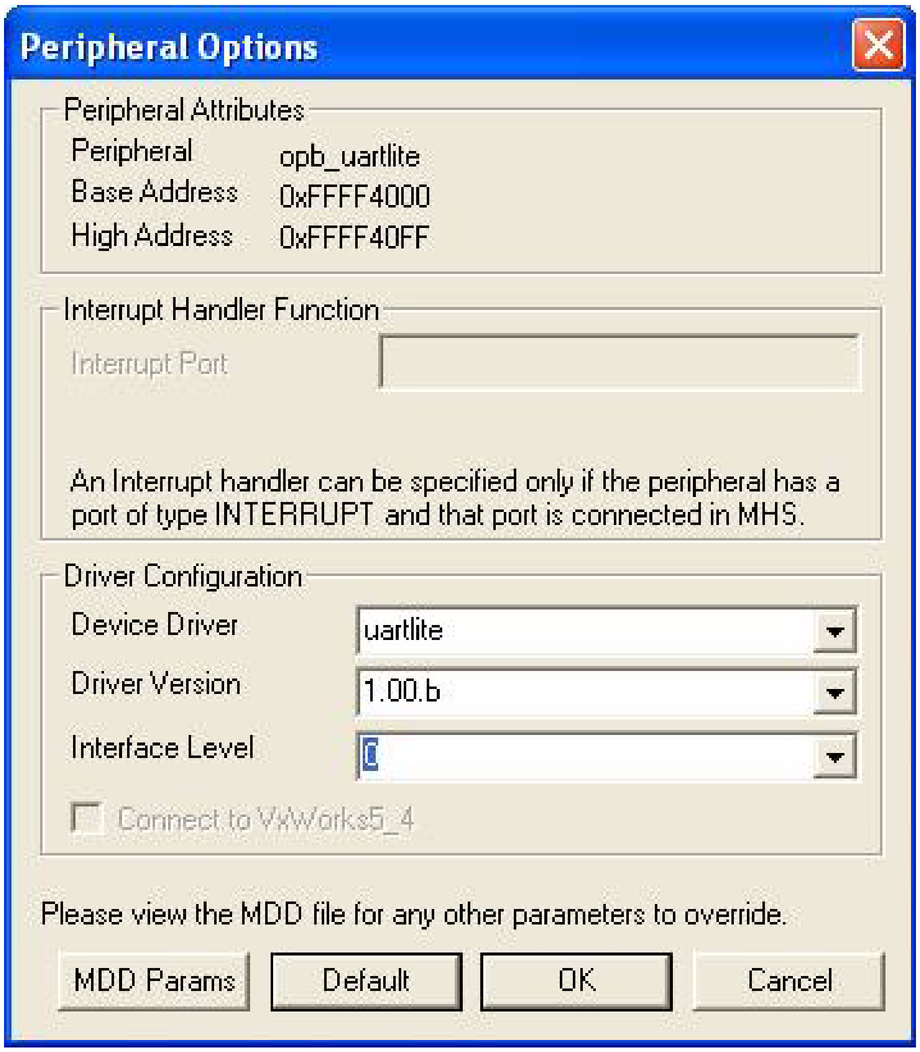
Defining the Software Design

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_mb directory.

Setting the Driver Interface Level

For each of the peripherals utilized in the tutorial design, you need to set the driver interface level as follows:

1. In XPS, select the System tab and double click on "my_uart" to open the Peripheral Options dialog box shown in the following figure. There are two levels of drivers for each peripheral.



Peripheral: opb_uartlite
Base Address: 0xFFFF4000
High Address: 0xFFFF40FF

Interrupt Port

An Interrupt handler can be specified only if the peripheral has a port of type INTERRUPT and that port is connected in MHS.

Device Driver: uartlite
Driver Version: 1.00.b
Interface Level: 0

Connect to VxWorks5_4

Please view the MDD file for any other parameters to override.

Figure 11: **Peripheral Options**

The architecture of the device drivers is layered as shown in the following table. This layered architecture accommodates the many use cases of device drivers and provides portability across operating systems, toolsets, and processors. The architecture provides seamless integration with RTOS (Layer 2) high-level device drivers that are full-featured and portable across operating systems and processors (Layer 1) and low-level drivers for simple use cases (Layer 0). You can use any or all layers.

Table 2: **Drivers Layered Architecture**

Layer 2, RTOS Adaptation
Layer 1, High Level Drivers
Layer 0, Low Level Drivers

2. Set the Interface Level to **`Level 0`**.

3. Click **OK**.
4. Set the Interface Level to **Level 1** for the following peripherals:
   - my_mblaze
   - my_gpin
   - my_gpout

Note: For information on what functions are available for the different driver levels, refer to xilinx_driver.pdf located in the Xilinx_EDK\doc directory.

Setting STDIN/STDOUT with Libgen

Libgen allows you to map printf, scanf, and so forth to an I/O peripheral in your design.

1. Double click on my_mblaze to open the "microblaze instance my_mblaze" dialog box.
2. Select the Processor Property tab.
3. In the Communication Peripherals section, use the pull-down menu for the STDIN and STDOUT Peripheral to select **my_uart**.
4. Click **OK**.
5. In XPS, select **Tools** → **Generate Libraries** to run libgen and compile the drivers associated with the design.
6. Libgen creates the following directories in the my_mblaze directories:
   - 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*.

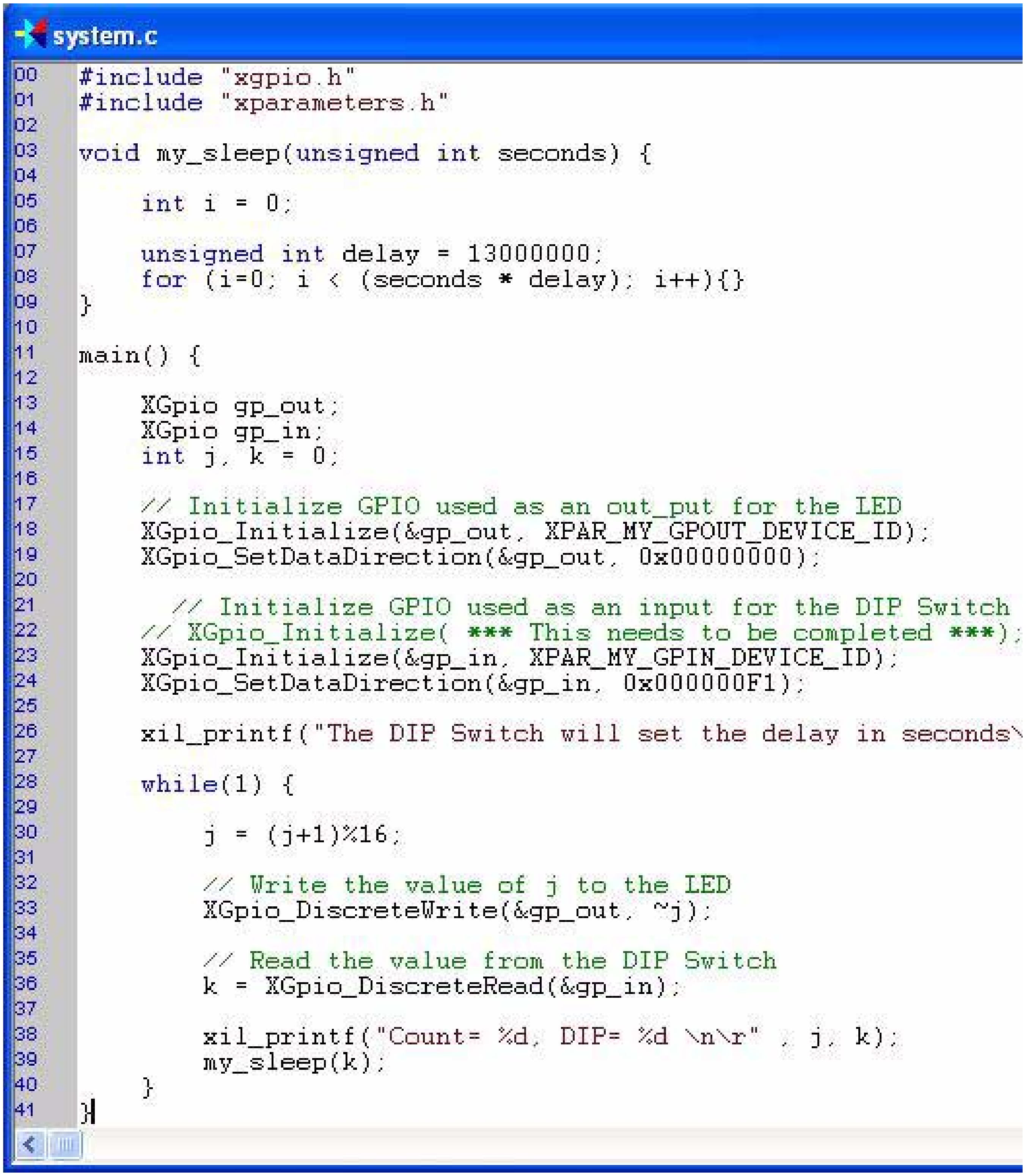
Finishing the Tutorial C Code

An incomplete C program is provided with this tutorial. This section walks you through the steps to complete the program. Specifically, you will complete the Xgpio_Initialize( ) function call. Additionally, you will fix an error in the program in the "Debugging the Design" section.

To complete the C program, follow these steps:

1. Click on **my_mblaze** in the System BSP tree.
2. Select **Project** → **Add Program Sources** to open the Add Source and Header Files to the current processor dialog box.
3. Select the **system.c** file located in the edk_tutorial_mb/code directory.

4. Double click on **`system.c`** to open it in XPS as shown in the following figure:



```
#include "xgpio.h"
#include "xparameters.h"

void my_sleep(unsigned int seconds) {

     int i = 0;

     unsigned int delay = 13000000;
     for (i=0; i < (seconds * delay); i++){}
}

main() {

     XGpio gp_out;
     XGpio gp_in;
     int j, k = 0;

     // Initialize GPIO used as an out_put for the LED
     XGpio_Initialize(&gp_out, XPAR_MY_GPOUT_DEVICE_ID);
     XGpio_SetDataDirection(&gp_out, 0x00000000);

       // Initialize GPIO used as an input for the DIP Switch
     // XGpio_Initialize( *** This needs to be completed ***);
     XGpio_Initialize(&gp_in, XPAR_MY_GPIN_DEVICE_ID);
     XGpio_SetDataDirection(&gp_in, 0x000000F1);

     xil_printf("The DIP Switch will set the delay in seconds\

     while(1) {

          j = (j+1)%16;

          // Write the value of j to the LED
          XGpio_DiscreteWrite(&gp_out, ~j);

          // Read the value from the DIP Switch
          k = XGpio_DiscreteRead(&gp_in);

          xil_printf("Count= %d, DIP= %d \n\r" , j, k);
          my_sleep(k);
     }
}
```

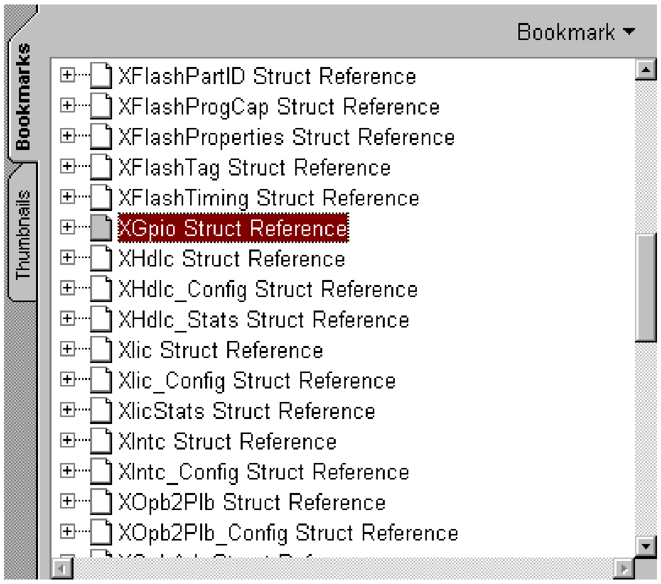
Figure 12: **Tutorial C Code**

5. Select the following:

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

6. Select **`Documents`**.
7. Select **`Xilinx Drivers`** to open xilinx_drivers.pdf.

8. Select **Xgpio Struct Reference** as shown in the following figure:



XFlashPartID Struct Reference
XFlashProgCap Struct Reference
XFlashProperties Struct Reference
XFlashTag Struct Reference
XFlashTiming Struct Reference
XGpio Struct Reference
XHdlc Struct Reference
XHdlc_Config Struct Reference
XHdlc_Stats Struct Reference
XIic Struct Reference
XIic_Config Struct Reference
XIicStats Struct Reference
XIntc Struct Reference
XIntc_Config Struct Reference
XOpb2Plb Struct Reference
XOpb2Plb_Config Struct Reference

Figure 13: **Driver Documentation**

9. Using the **Edit** → **Find** search for XGpio_Initialize function until the detailed description of the function is found..
10. 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.
11. The first parameter you need to add is a pointer to an Xgpio instance. Note that a variable named gp_in 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(&gp_in,
12. 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**.
13. Browse to the edk_tutorial_mb\my_mblaze\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.
14. In the xparameters.h file find the following #define used to identify the MYGPIN peripheral:

    #define XPAR_MY_GPIN_DEVICE_ID 0

    Note: The "MY_GPIN" matches the instance name assigned in the MHS file for this peripheral.

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

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

    XGpio_Initialize(&gp_in, XPAR_MY_GPIN_DEVICE_ID);

16. 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 inst_microblaze/code 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.

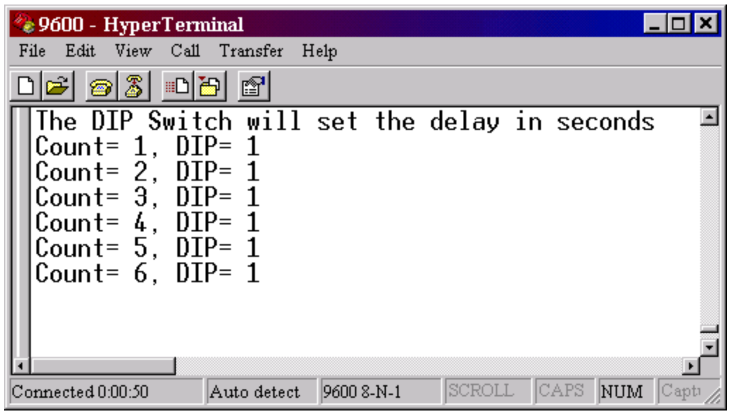
Downloading the Design

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 : 1**`9200`**
   - 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.bit file in the proj_nav_proj directory.
7. Select bram_init_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:


9600 - HyperTerminal
File Edit View Call Transfer Help

```
The DIP Switch will set the delay in seconds
Count= 1, DIP= 1
Count= 2, DIP= 1
Count= 3, DIP= 1
Count= 4, DIP= 1
Count= 5, DIP= 1
Count= 6, DIP= 1
```

Connected 0:00:50 | Auto detect | 9600 8-N-1 | SCROLL | CAPS | NUM | Captu

Figure 14: **Hyperterminal Output**

11. As the message states, the DIP switches control the rate at which the counter counts. By turning on switch two, the count is now delayed by three seconds. The delay can be specific from 0 to 255 in binary using all eight switches.

    Tutorial Test Questions:

    If you turn on the second and third DIP switch, is the count delayed by seven seconds?___________

    Does the HyperTerminal DIP output equal seven? ___________

12. The "Debugging the Design" section describes how to solve this software coding error. Close the Hyperterminal.

Debugging the Design

Now that the device is configured, you can debug the software application directly via the xmd_stub software. GDB connects to the xmd_stub core through the Xilinx Microprocessor Debug (XMD) engine utility. 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 as illustrated in the following figure:

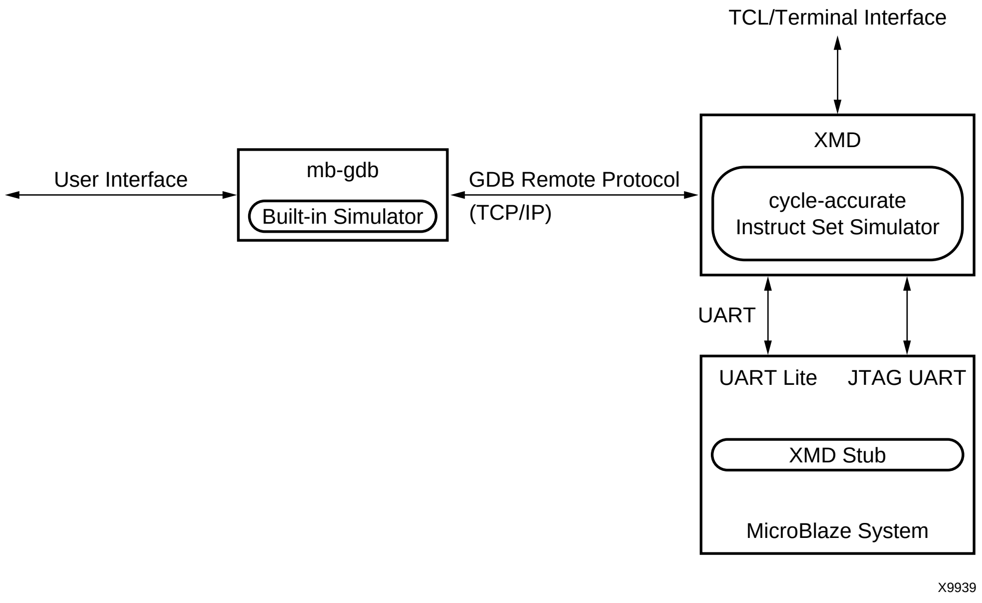


Figure 15: **GDB and XMD Connections**

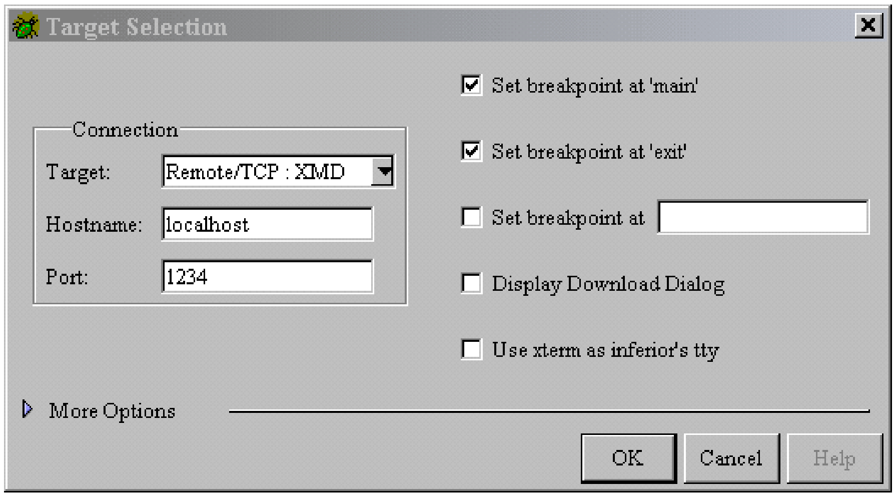
To debug the design, follow these steps:

1. Double click inst_microblaze to open the S/W Settings – microblaze instance my_mblaze dialog box.
2. Select the Processor Property tab. Select **`XmdStub`** and set the Debug Peripheral as **`myuart`**. Click **`OK`**.
3. Select **`Tools`** → **`Compile Program Sources`**.
4. Select **`Tools`** → **`Download`** to download the XMDSTUB.
5. Select **`Tools`** → **`XMD`**.
6. After xmd has initialized, enter the following:

   ```
   mbconnect stub -comm serial -port com1 -baud 19200
   ```

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

8. In GDB, select **File** → **Target Settings** to display the Target Selection dialog box as shown in the following figure:



Target Selection

Connection

Target: Remote/TCP : XMD

Hostname: localhost

Port: 1234

☑ Set breakpoint at 'main'

☑ Set breakpoint at 'exit'

☐ Set breakpoint at

☐ Display Download Dialog

☐ Use xterm as inferior's tty

Figure 16: **Target Selection**

9. Configure the Target Selection dialog box to match Figure 16. Click **Ok**.
10. In GDB, select **File** → **Open File**.
11. Select executable.elf in the inst_microblaze/code directory.

    Tutorial Test Questions:

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

    Why can you not see the C code? ____

12. In GDB, select **File** → **Exit**.
13. In XPS, select **Tools** → **Set Options** → **Compiler Options**.
14. In the microblaze instance inst_microblaze dialog box, select the Optimization tab.
15. Select **Create symbols for debugging (-g option)**.
16. Click **OK**.
17. 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.

18. 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.

19. 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. The following file edk_tutorial_ppc\code\simulation.c can be used as a reference.

To simulate the design, follow these steps:

1. In XPS, the simulation model is specified by right clicking on **`Sim Model`** in the System window and selecting either **`Behavior`**, **`Structural`**, or **`Timing`**.
2. Select **`Tools`** → **`Sim Model Generation`** to generate a simulation model. This generates the following files in the simulation directory:
   - system.do – initializes the simulation environment
   - system_init.do - memory initialization of BRAMs
   - system_comp.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.
3. Open the ISE project in the proj_nav_proj directory.
4. Select **`Project`** → **`Add Source`** and select testbench.vhd in the sim directory. Associate the testbench.vhd with top.vhd.
5. Double click the testbench.vhd file in the Source Window. Examine the testbench.vhd file. Note that the test bench contains several processes to drive the top level signals.
6. The system_init.vhd contains the BRAM initialization strings. The initialization strings are applied to the design utilizing a configuration statement. The following configuration must be included at the end of the testbench.vhd:

```
configuration testbench_conf of testbench is
  for behavior
      for uut : system
                for IMP
                      for all : lmbbram_wrapper use configuration work.lmbbram_conf;
                      end for;
                end for;
      end for;
   end for;
end testbench_conf;
```

7. Save and close the testbench.vhd file
8. Select **`File`** → **`Open`** to open the system_init.do in the simulation directory.
9. Examine the system_init.do file. Note that the testbench.vhd has not been included. Add the following command after the line compiling the system_init.vhd:

   ```
   vcom -93 -work work ../sim/testbench.vhd
   ```
10. Change the line :

    ```
    vcom -93 -work work ./system_init.vhd
    ```

to:

```
vcom -93 -work work ../simulation/system_init.vhd
```

11. Change the line:

    ```
    do system_comp.do
    ```

    to:

    ```
    do ../simulation/system_comp.do
    ```

12. Now that a testbench has been added to the system_init.do file, add the following vsim command line:

    ```
    vsim -Lf unisim -t ps +notimingchecks work.testbench_conf
    ```

    This loads the configuration statement in testbench.vhd.

13. Add the following command to the system_init.do file located in the sim directory:

    ```
    add wave *
    ```

14. Select **File** → **Save As** and save system_init.do as system.do in the sim directory. This protects the file from being overwritten.
15. In the Source Window in ISE, select the testbench.vhd.
16. 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 system.do in the sim 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
    ```