# An Introduction to EDK and MicroBlaze™ Workshop (Part 2)

## Lab 1

**February 2008**
**Version 1.1**

## Lab Objectives

The objective of Lab 1 is to show how to use the IP creation wizard to create a simple custom IP, add the IP to an existing design and write a simple software application to test the IP.

## Lab Setup

This lab will require the following software and hardware setups.

**Software**

The software requirements for this lab are:

- WindowsXP Professional  (Windows2000 Professional also supported)
- Xilinx ISE 9.2i with Service Pack 3
- Xilinx EDK 9.2i with Service Pack 2
- Xilinx EDK BFM Package

**Hardware**

The hardware setup for this lab is:

- Computer with 1 GB RAM and 1 GB virtual memory (recommended)
- Xilinx Spartan-3A DSP Starter board
- 5V power supply
- USB JTAG cable
- Serial cable

## Creating a Custom IP

A Custom Memory Controller IP core will be created and added to an existing design. In order to test this memory controller, a Custom Memory Block IP core (already designed) will be attached to the memory controller so that the processor can verify the functionality of the controller. A high-level block diagram of the Lab 1 system is shown in the following figure.

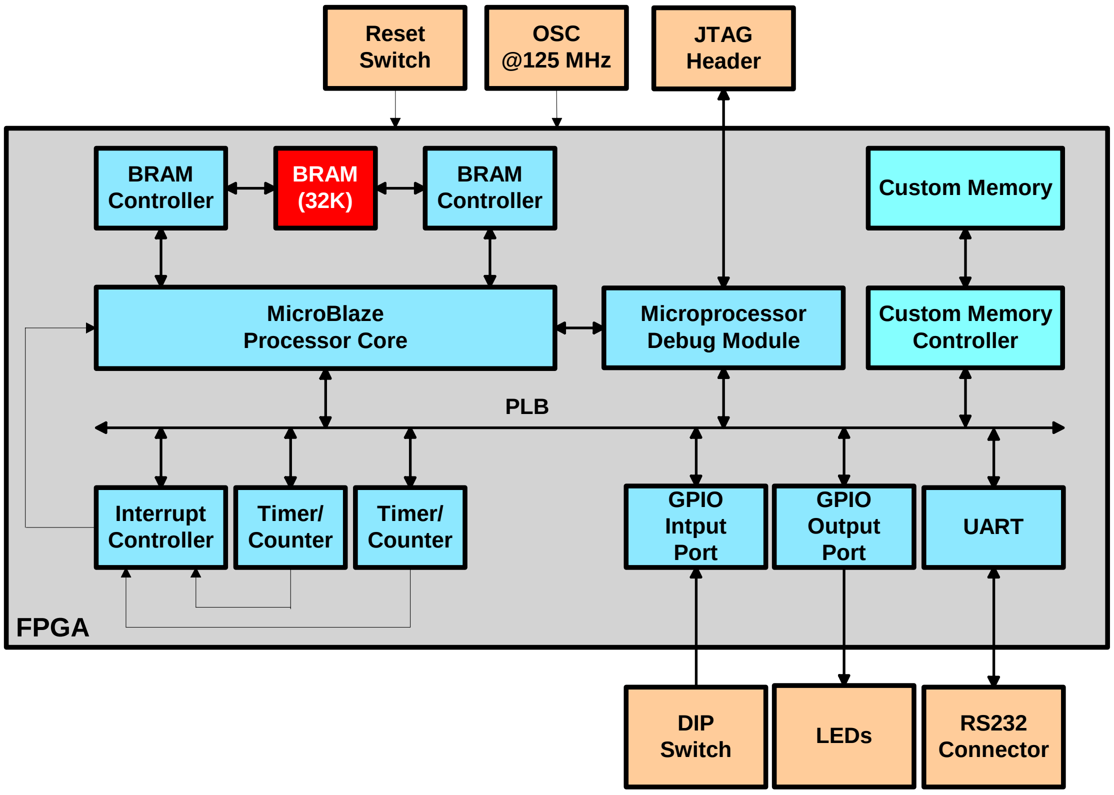


**Figure 1 – Lab 1 System Block Diagram**

The custom IP creation consists of the following steps:

- Create the custom IP using the IP Creation wizard. This will generate HDL templates that need to be modified by the user.
- Design the user logic portion of the custom IP and integrate it into the templates generated by the IP creation wizard.
- Import the custom IP using the IP Import wizard

### Custom Memory Controller IP Core

The following figure shows the Custom Memory Controller I/O signals as well as its connection to the Custom Memory Block and the PLB interface. Although, this memory controller is designed to have separate data in and data out buses, it can easily be modified to support bi-directional data bus.

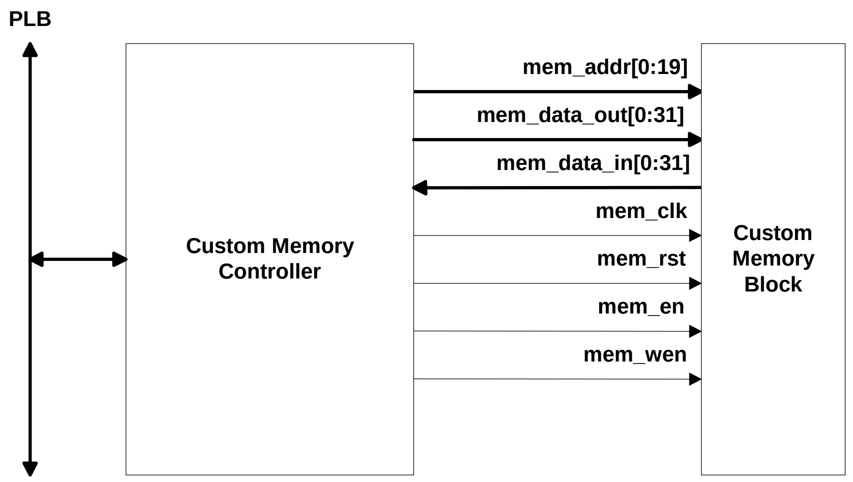
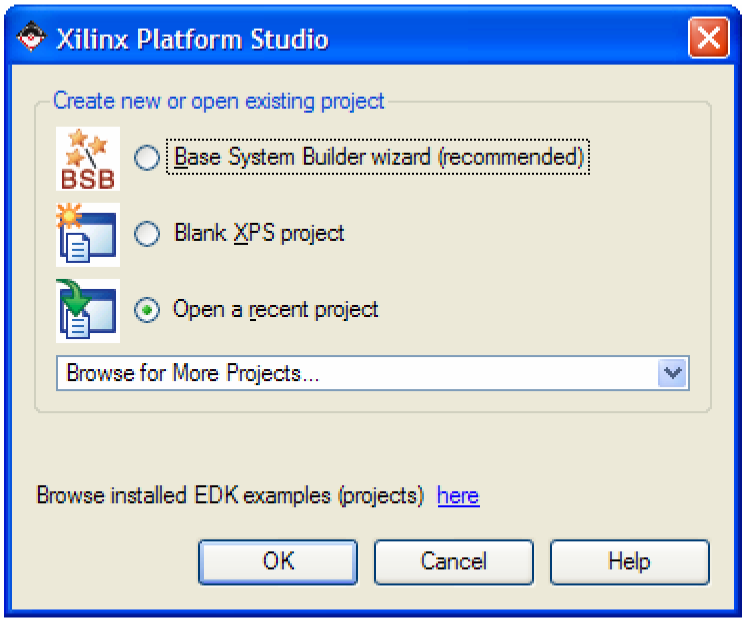


**Figure 2 – Custom Memory Controller IP Core**

## Creating the Custom Memory Controller IP Using the IP Creation Wizard

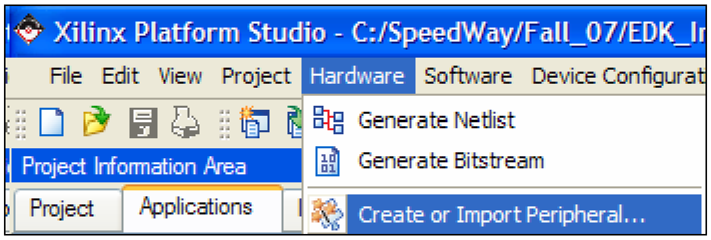
1. Start XPS via **Start > All Programs > Xilinx Platform Studio 9.2i > Xilinx Platform Studio**, the following dialog box will appear.
   a) Select **Open a recent project**.
   b) Click **OK** to continue (**Do Not** browse to the recent projects).

**Note**: If XPS is already started, simply select **File > Open Project** to open an existing project.

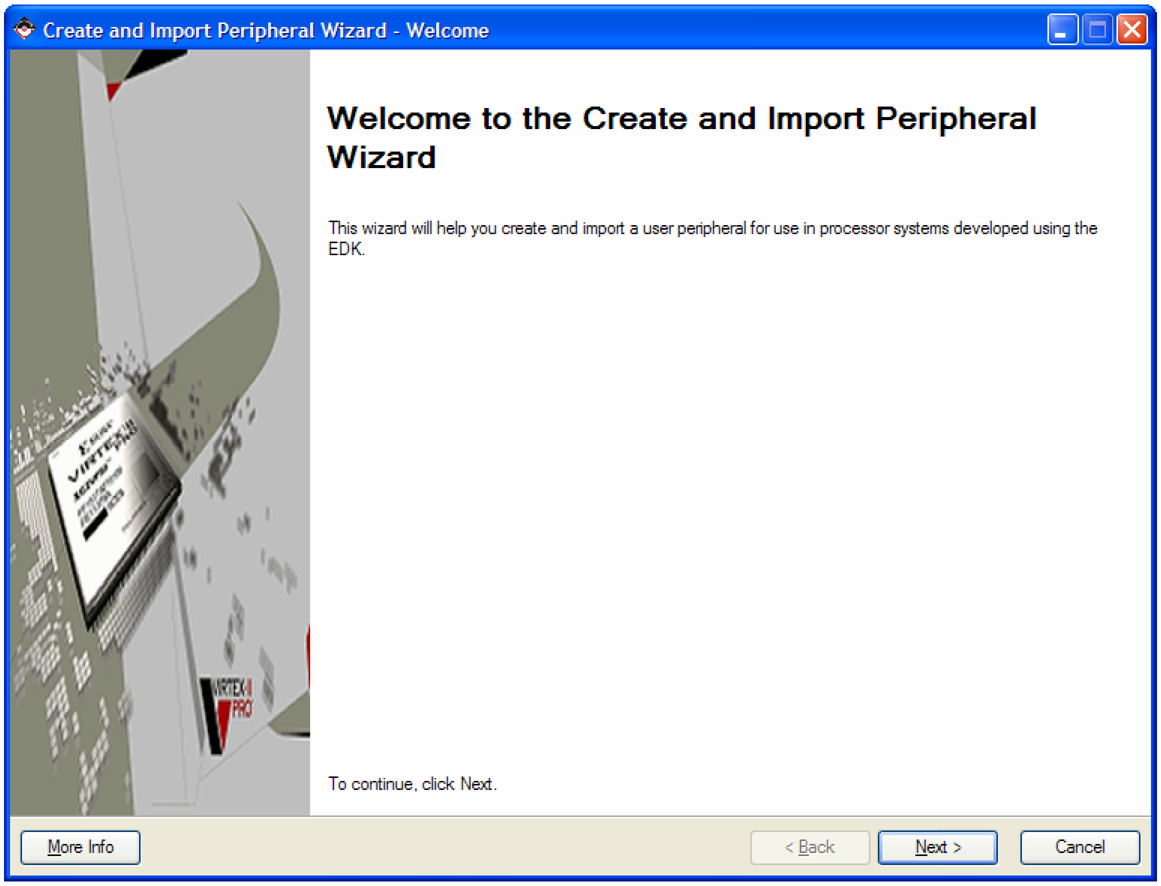

Xilinx Platform Studio
Create new or open existing project
BSB
Base System Builder wizard (recommended)
Blank XPS project
Open a recent project
Browse for More Projects...
Browse installed EDK examples (projects) here
OK
Cancel
Help

2. Browse to the **C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1** folder, select **system.xmp** and click **Open**.

3. Invoke the **IP Creation** wizard by selecting **Hardware > Create or Import Peripheral** from the XPS GUI as shown in the following figure.



4. The **Create and Import Peripheral Wizard** dialog box will appear. Click **Next** to continue.

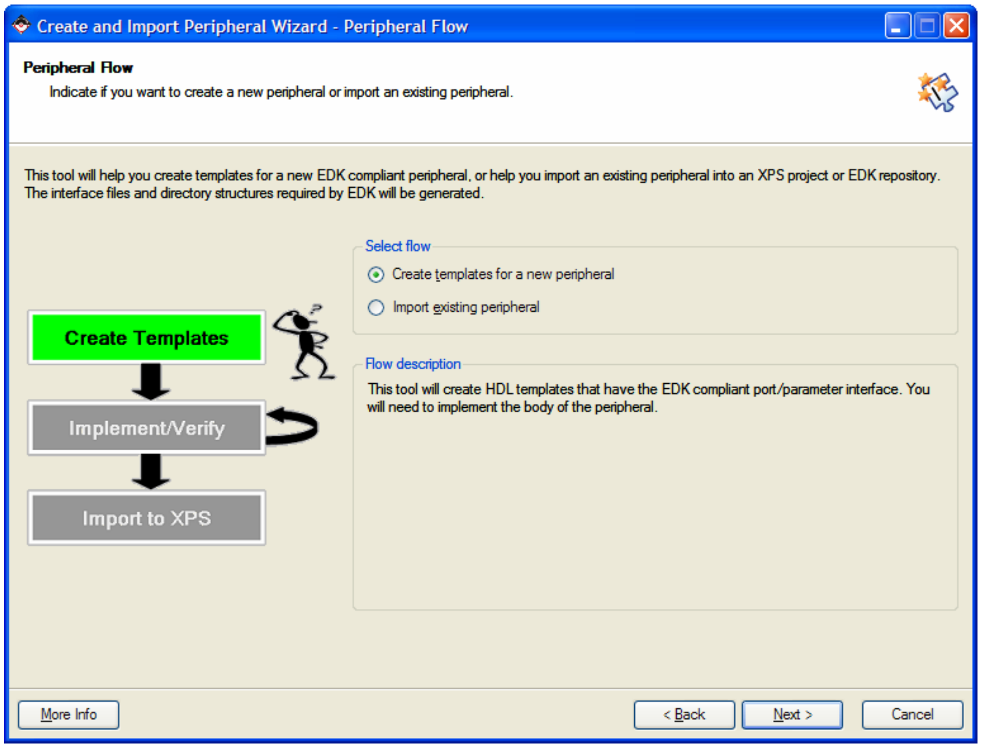


Welcome to the Create and Import Peripheral Wizard

This wizard will help you create and import a user peripheral for use in processor systems developed using the EDK.

To continue, click Next.

5. Select **Create templates for a new peripheral** and click **Next** to continue.


Create and Import Peripheral Wizard - Peripheral Flow

Peripheral Flow

Indicate if you want to create a new peripheral or import an existing peripheral.

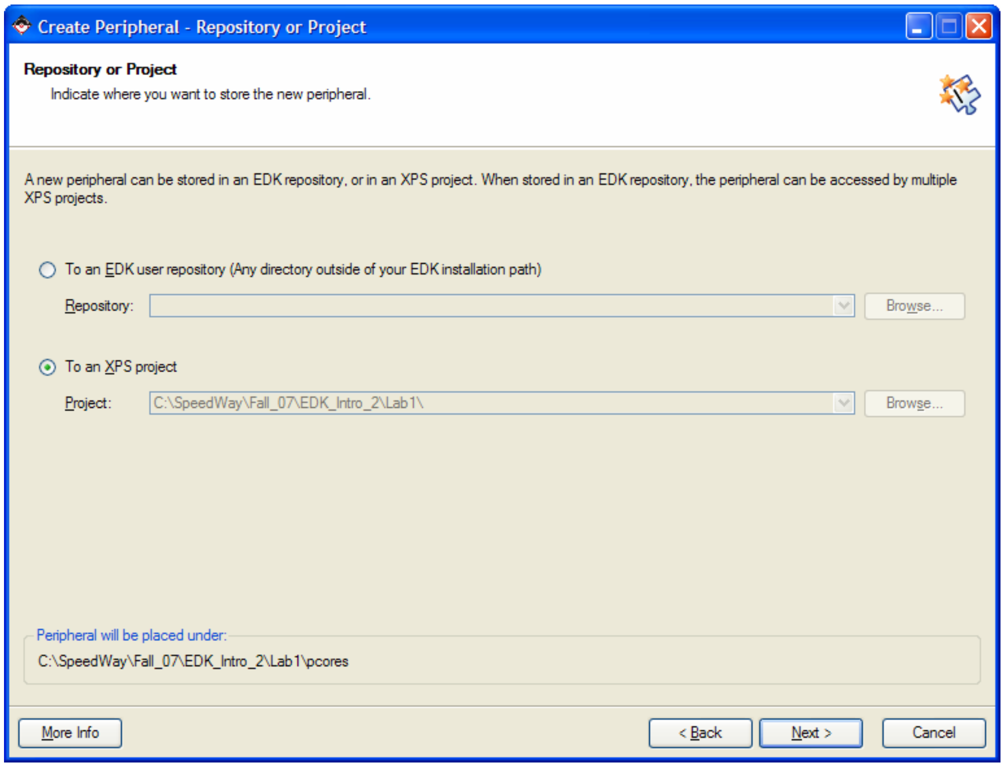
This tool will help you create templates for a new EDK compliant peripheral, or help you import an existing peripheral into an XPS project or EDK repository. The interface files and directory structures required by EDK will be generated.

Select flow
- [✓ Create templates for a new peripheral]
- Import existing peripheral

Flow description

This tool will create HDL templates that have the EDK compliant port/parameter interface. You will need to implement the body of the peripheral.

6. The following dialog box will appear indicating the custom IP core will be generated and stored in the **/pcores** folder of the Lab 1 project directory. Click **Next** to continue.


Create Peripheral - Repository or Project

Repository or Project

Indicate where you want to store the new peripheral.

A new peripheral can be stored in an EDK repository, or in an XPS project. When stored in an EDK repository, the peripheral can be accessed by multiple XPS projects.

- To an EDK user repository (Any directory outside of your EDK installation path)

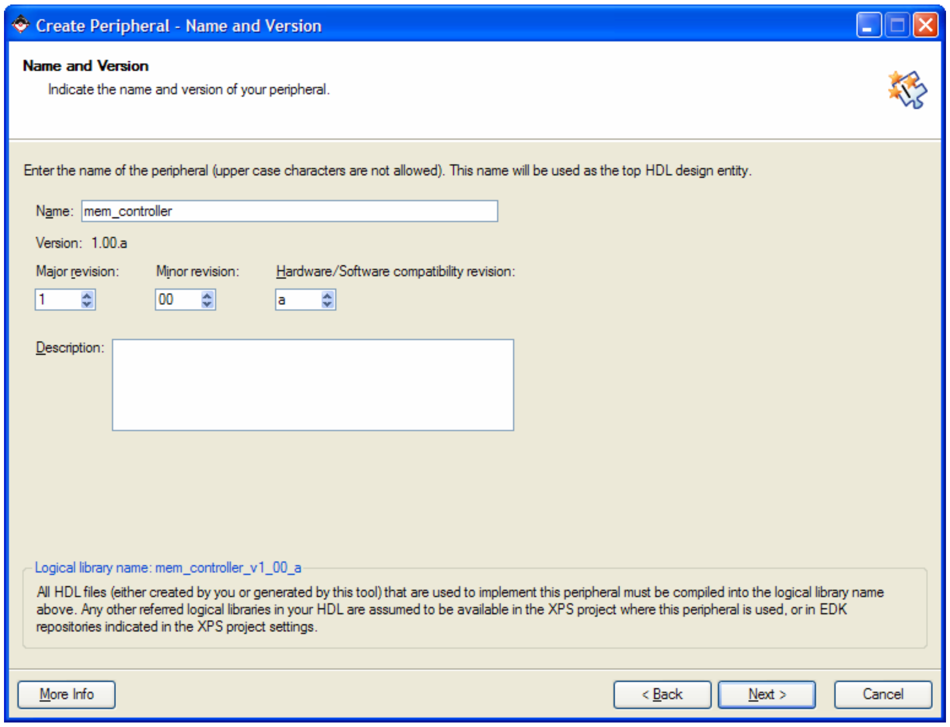
  Repository:
- [✓ To an XPS project]

  Project: C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\

Peripheral will be placed under:

C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores

7. Enter **mem_controller** for the name of the IP and leave the revision of the IP as **1.00.a**. Click **Next** to continue.



**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: mem_controller

Version: 1.00.a

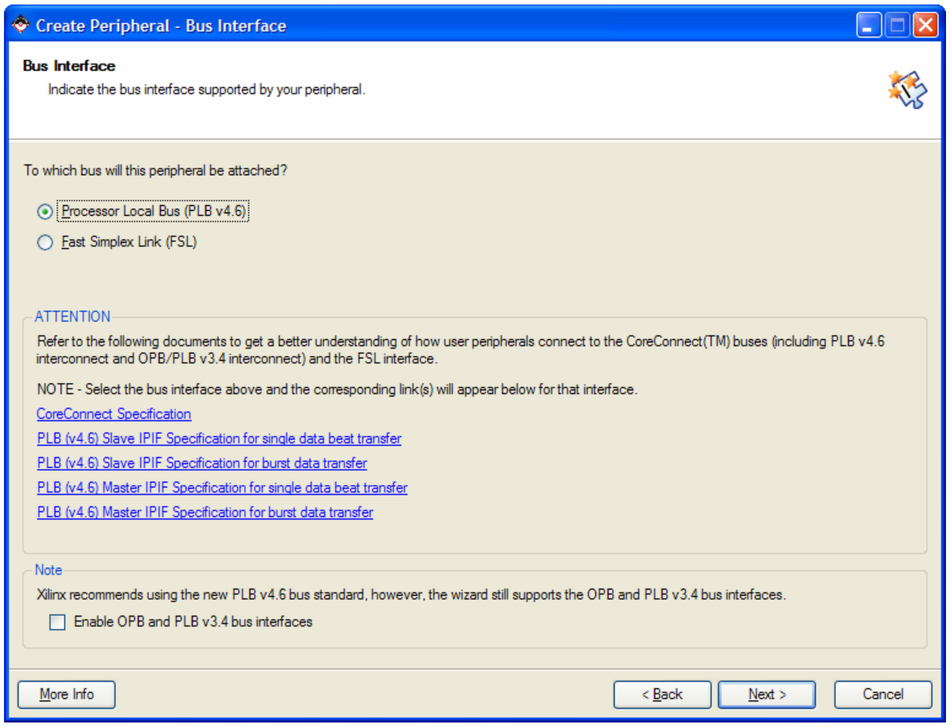
Major revision: 1  Minor revision: 00  Hardware/Software compatibility revision: a

Description:

Logical library name: mem_controller_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.

8. The custom IP core will be designed to connect to the PLB interface. So, select **Processor Local Bus (PLB v4.6)** and click **Next** to continue.



**Bus Interface**

Indicate the bus interface supported by your peripheral.

To which bus will this peripheral be attached?

[✓ Processor Local Bus (PLB v4.6)]

Fast Simplex Link (FSL)

ATTENTION

Refer to the following documents to get a better understanding of how user peripherals connect to the CoreConnect(TM) buses (including PLB v4.6 interconnect and OPB/PLB v3.4 interconnect) and the FSL interface.

NOTE - Select the bus interface above and the corresponding link(s) will appear below for that interface.

CoreConnect Specification

PLB (v4.6) Slave IPIF Specification for single data beat transfer

PLB (v4.6) Slave IPIF Specification for burst data transfer

PLB (v4.6) Master IPIF Specification for single data beat transfer

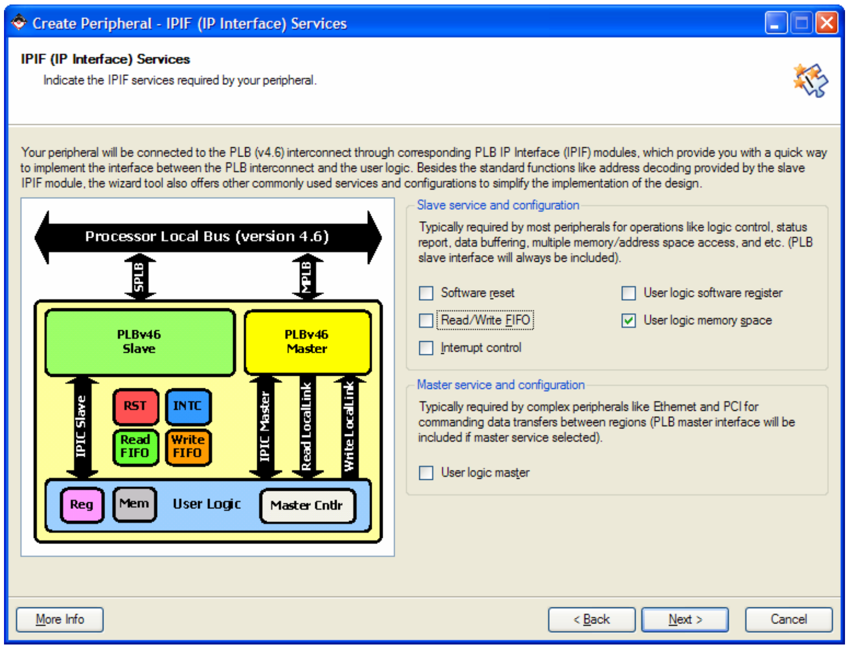
PLB (v4.6) Master IPIF Specification for burst data transfer

Note

Xilinx recommends using the new PLB v4.6 bus standard, however, the wizard still supports the OPB and PLB v3.4 bus interfaces.

☐ Enable OPB and PLB v3.4 bus interfaces

9. This memory controller IP needs to provide address, data, and control signals to the memory it connects to. So, in the **IPIF (IP Interface) Services** dialog box:
    a) Un-check the **User logic software register**.
    b) Check the **User Logic Memory Space**.
    c) Click **Next** to continue.



**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).

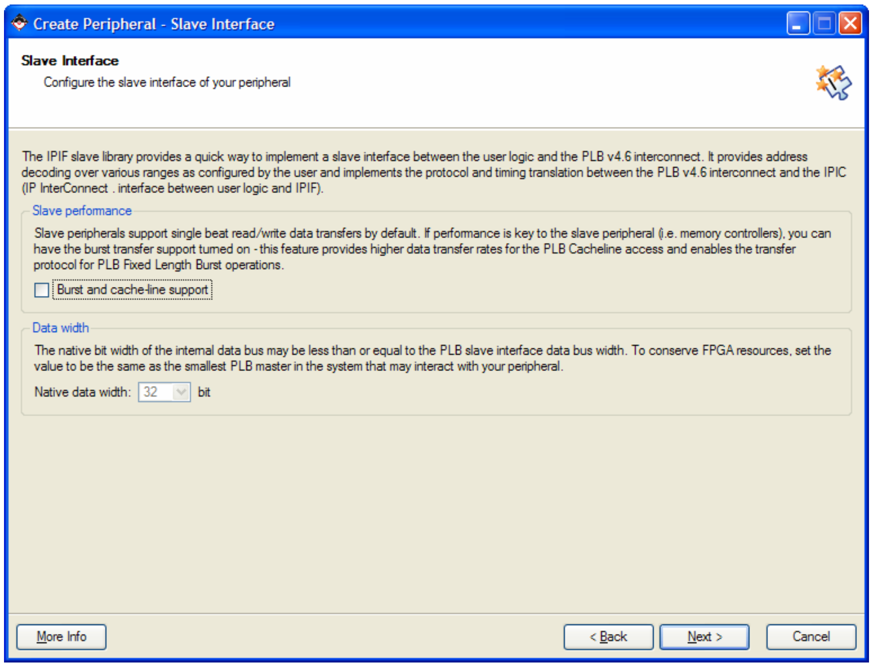
☐ Software reset ☐ User logic software register
☐ Read/Write FIFO ☑ User logic memory space
☐ Interrupt control

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

10. In the **Slave Interface** dialog box, accept the default settings and click **Next** to continue.



**Slave Interface**

Configure the slave interface of your peripheral

The IPIF slave library provides a quick way to implement a slave interface between the user logic and the PLB v4.6 interconnect. It provides address decoding over various ranges as configured by the user and implements the protocol and timing translation between the PLB v4.6 interconnect and the IPIC (IP InterConnect . interface between user logic and IPIF).

Slave performance

Slave peripherals support single beat read/write data transfers by default. If performance is key to the slave peripheral (i.e. memory controllers), you can have the burst transfer support turned on - this feature provides higher data transfer rates for the PLB Cacheline access and enables the transfer protocol for PLB Fixed Length Burst operations.

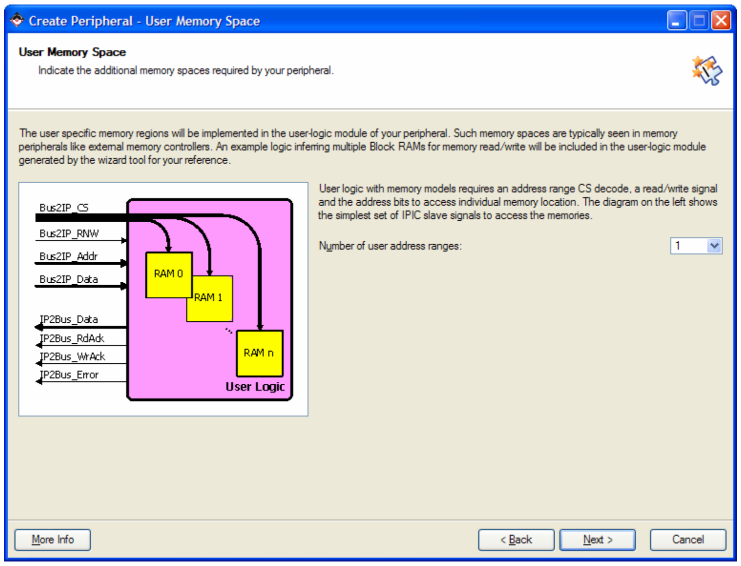
☐ Burst and cache-line support

Data width

The native bit width of the internal data bus may be less than or equal to the PLB slave interface data bus width. To conserve FPGA resources, set the value to be the same as the smallest PLB master in the system that may interact with your peripheral.

Native data width: 32 bit

11. The **User Memory Space** dialog box can be used to allow an IP connected to the processor bus to support multiple memory spaces. This feature is useful for memory controllers that need to be connected to different physical memory banks.
    For example, the controller can be designed to interface to SRAM, Flash and ZBT SRAM. In this case, the controller needs to decode three address spaces to generate the chip select for the three memory devices. For this design, a single memory space is used to interface to the memory. Accept the default settings and click **Next** to continue.



Create Peripheral - User Memory Space

**User Memory Space**

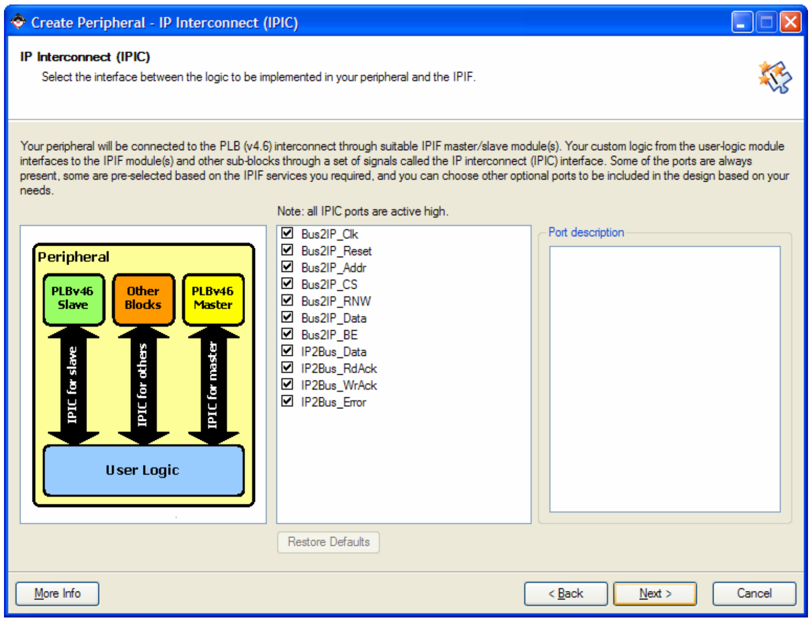
Indicate the additional memory spaces required by your peripheral.

The user specific memory regions will be implemented in the user-logic module of your peripheral. Such memory spaces are typically seen in memory peripherals like external memory controllers. An example logic inferring multiple Block RAMs for memory read/write will be included in the user-logic module generated by the wizard tool for your reference.

User logic with memory models requires an address range CS decode, a read/write signal and the address bits to access individual memory location. The diagram on the left shows the simplest set of IPIC slave signals to access the memories.

Number of user address ranges: 1

12. The **IP InterConnect (IPIC)** dialog box shows the signals that are used to interface the user logic to the **IP InterFace (IPIF)**. The IP creation wizard selects a set of signals based on the selected features in the **IPIF Services** dialog box (step 9). Accept the default settings and click **Next** to continue



Create Peripheral - IP Interconnect (IPIC)

**IP Interconnect (IPIC)**

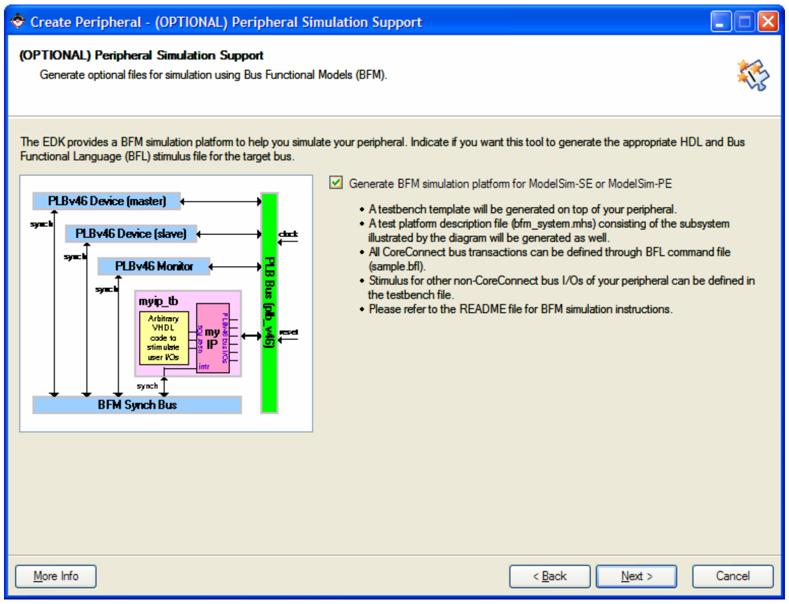
Select the interface between the logic to be implemented in your peripheral and the IPIF.

Your peripheral will be connected to the PLB (v4.6) interconnect through suitable IPIF master/slave module(s). Your custom logic from the user-logic module interfaces to the IPIF module(s) and other sub-blocks through a set of signals called the IP interconnect (IPIC) interface. Some of the ports are always present, some are pre-selected based on the IPIF services you required, and you can choose other optional ports to be included in the design based on your needs.

Note: all IPIC ports are active high.

- ☑ Bus2IP_Clk
- ☑ Bus2IP_Reset
- ☑ Bus2IP_Addr
- ☑ Bus2IP_CS
- ☑ Bus2IP_RNW
- ☑ Bus2IP_Data
- ☑ Bus2IP_BE
- ☑ IP2Bus_Data
- ☑ IP2Bus_RdAck
- ☑ IP2Bus_WrAck
- ☑ IP2Bus_Error

13. Bus Functional Model (BFM) simulation is often used to perform a behavioral simulation of the custom IP prior to integrating the new IP into a system.
    a) Check the **Generate BFM simulation platform for ModelSim-SE or ModelSim-PE** box.
    b) Click **Next** to continue.



Create Peripheral - (OPTIONAL) Peripheral Simulation Support

(OPTIONAL) Peripheral Simulation Support

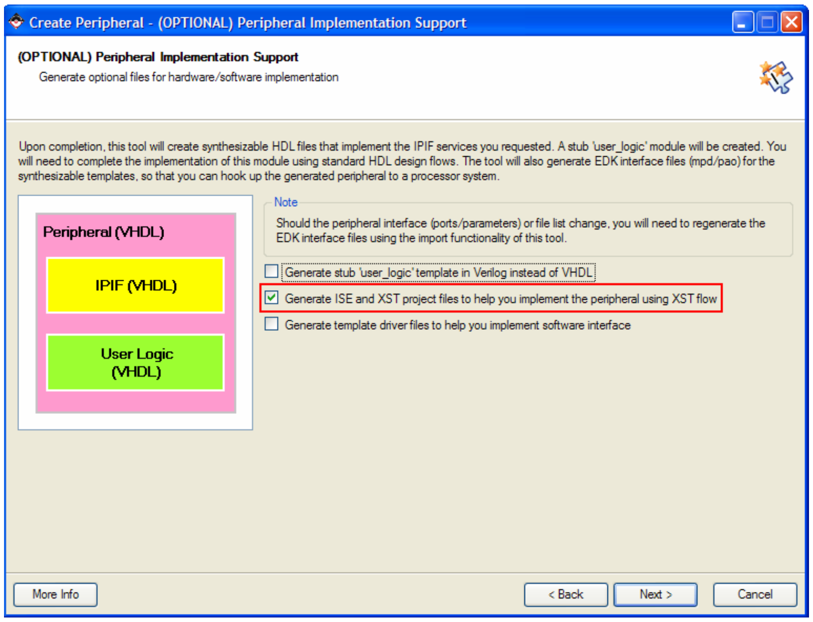
Generate optional files for simulation using Bus Functional Models (BFM).

The EDK provides a BFM simulation platform to help you simulate your peripheral. Indicate if you want this tool to generate the appropriate HDL and Bus Functional Language (BFL) stimulus file for the target bus.

☑ Generate BFM simulation platform for ModelSim-SE or ModelSim-PE

- A testbench template will be generated on top of your peripheral.
- A test platform description file (bfm_system.mhs) consisting of the subsystem illustrated by the diagram will be generated as well.
- All CoreConnect bus transactions can be defined through BFL command file (sample.bfl).
- Stimulus for other non-CoreConnect bus I/Os of your peripheral can be defined in the testbench file.
- Please refer to the README file for BFM simulation instructions.

14. In the **(OPTIONAL) Peripheral Implementation Support** dialog box:
    a) Make sure the middle box is checked (**no other boxes should be checked**) as shown in the following figure. Checking this box allows the IP creation wizard to generate ISE and XST project files to help implement the custom IP using the XST synthesis tool.
    b) Click **Next** to continue.



Create Peripheral - (OPTIONAL) Peripheral Implementation Support

(OPTIONAL) Peripheral Implementation Support

Generate optional files for hardware/software implementation

Upon completion, this tool will create synthesizable HDL files that implement the IPIF services you requested. A stub 'user_logic' module will be created. You will need to complete the implementation of this module using standard HDL design flows. The tool will also generate EDK interface files (mpd/pao) for the synthesizable templates, so that you can hook up the generated peripheral to a processor system.

Note

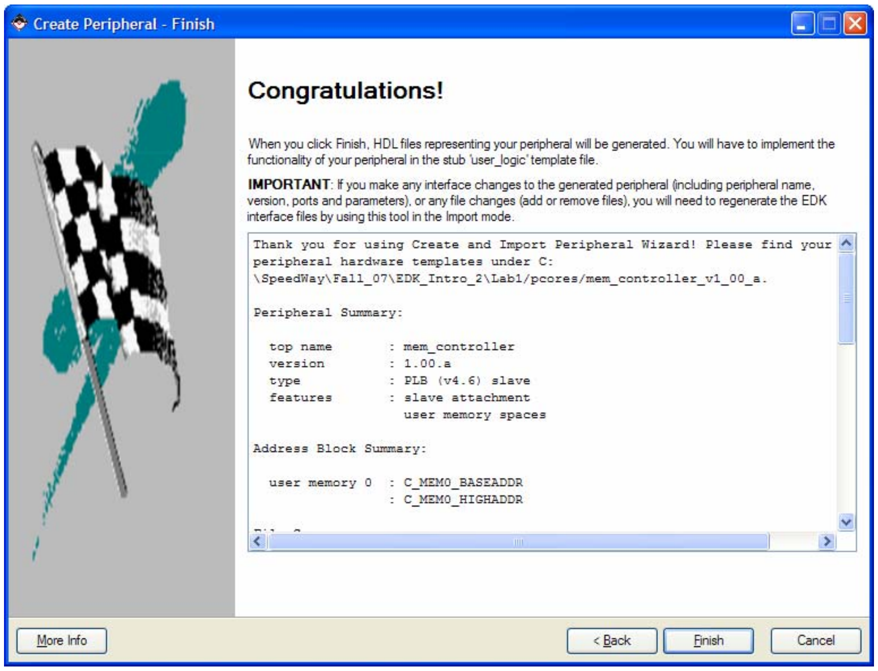
Should the peripheral interface (ports/parameters) or file list change, you will need to regenerate the EDK interface files using the import functionality of this tool.

☐ Generate stub 'user_logic' template in Verilog instead of VHDL

☑ Generate ISE and XST project files to help you implement the peripheral using XST flow

☐ Generate template driver files to help you implement software interface

15. The IP creation is completed. Click **Finish** to continue.



Congratulations!

When you click Finish, HDL files representing your peripheral will be generated. You will have to implement the functionality of your peripheral in the stub 'user_logic' template file.

**IMPORTANT**: If you make any interface changes to the generated peripheral (including peripheral name, version, ports and parameters), or any file changes (add or remove files), you will need to regenerate the EDK interface files by using this tool in the Import mode.

```
Thank you for using Create and Import Peripheral Wizard! Please find your
peripheral hardware templates under C:
\SpeedWay\Fall_07\EDK_Intro_2\Lab1/pcores/mem_controller_v1_00_a.

Peripheral Summary:

  top name       : mem_controller
  version        : 1.00.a
  type           : PLB (v4.6) slave
  features       : slave attachment
                   user memory spaces

Address Block Summary:

  user memory 0  : C_MEM0_BASEADDR
                 : C_MEM0_HIGHADDR
```

***Questions:***

1. *Which folder under the project directory is the custom IP stored in?*
2. *Which folder contains the VHDL templates for the custom IP?*
3. *What is the version of the MPD file for the custom IP?*

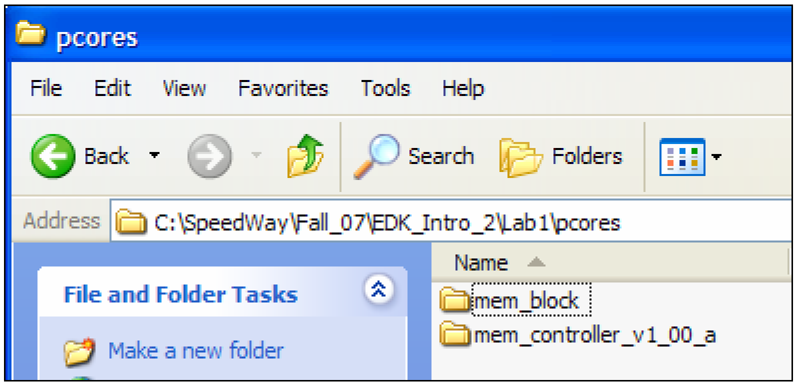
## Implementing the User Logic Portion of the Custom IP

The IP creation wizard creates two VHDL templates for the custom IP core, a top-level VHDL template called **mem_controller.vhd** and a user logic template called **user_logic.vhd**. These files are located in the **C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores\mem_controller_v1_00_a\hdl\vhdl** folder of the custom IP core.

The **user_logic.vhd** template must be modified by the user to include the core logic that interfaces the memory controller to the memory and also provides the interface to the IPIC. The IP creation wizard takes care of implementing the interface from IPIC to IPIF and from IPIF to PLB. Hence, the combination of the user logic and the portion implemented by the IP creation wizard completes the custom memory controller IP design.

In order to save time, the user logic portion of the **mem_controller** IP and also the Custom Memory Block IP (**mem_block**) that is used to test the controller have been implemented. Please follow the steps shown below to copy the **mem_block** IP and the user logic portion of the **mem_controller** IP core to the Lab 1 project folder.

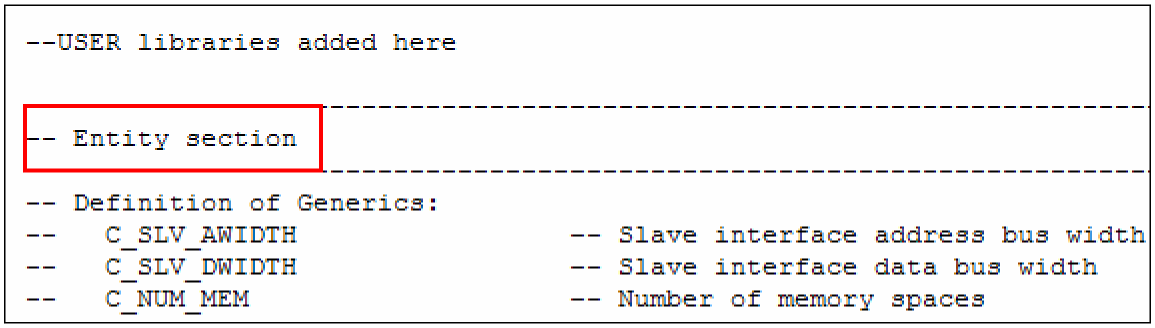
1. Go to the **C:\SpeedWay\Fall_07\EDK_Intro_2\Misc\ip_cores** directory and copy the **mem_block** folder to the **C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores** folder. The **mem_block** folder contains the IP core for the Custom Memory Block. The **C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores** folder should look as shown in the following figure.



C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores

mem_block
mem_controller_v1_00_a

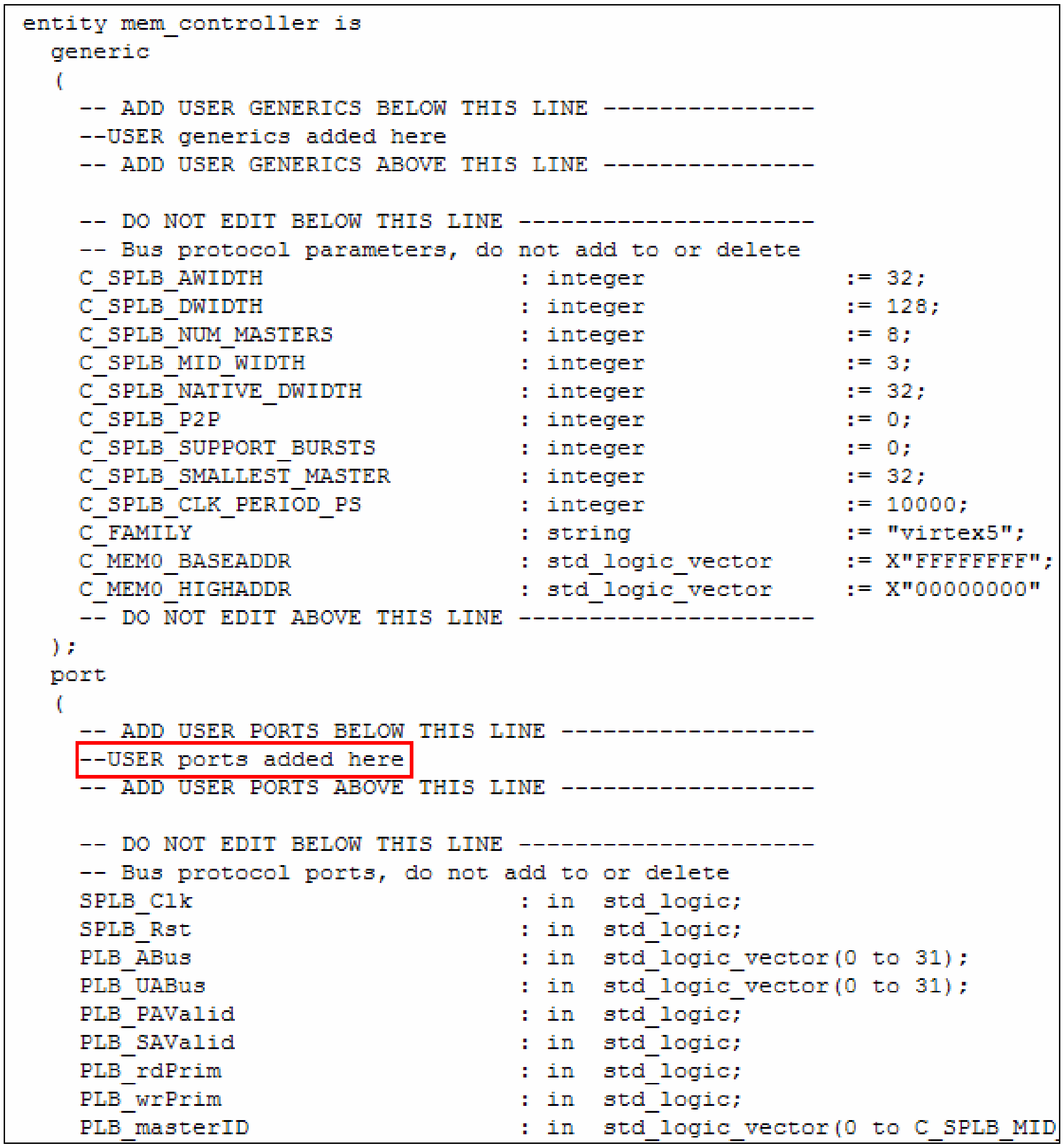
2. Go to the **C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores\ mem_controller_v1_00_a\hdl\vhdl** folder and open the **user_logic.vhd** file (with WordPad).
   a) Delete everything below the **-- Entity section** line as shown in the following figure.
   b) Please keep the file open.



```
--USER libraries added here

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_SLV_AWIDTH                -- Slave interface address bus width
--   C_SLV_DWIDTH                -- Slave interface data bus width
--   C_NUM_MEM                   -- Number of memory spaces
```

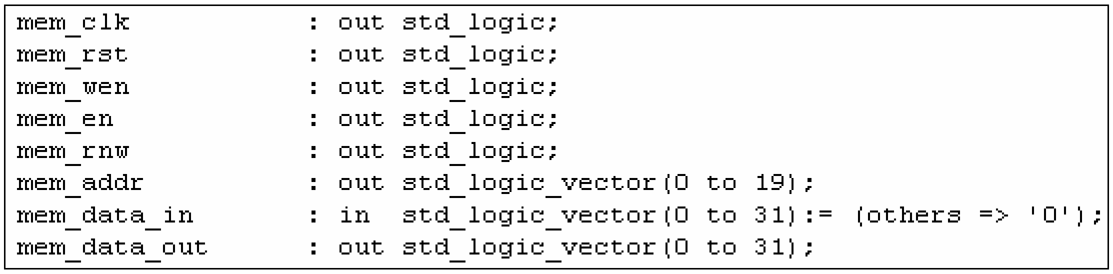
3. Go to the **C:\SpeedWay\Fall_07\EDK_Intro_2\Misc\ip_cores** folder and open the **user_logic_solution.vhd** file (with WordPad).
   a) Copy the entire contents of the **user_logic_solution.vhd** file and paste it in the **user_logic.vhd** file below the **-- Entity section** line.
   b) Close the **user_logic_solution.vhd** file.
   c) **Save** and **Close** the **user_logic.vhd** file.

4. Go to the **C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores\mem_controller_v1_00_a\hdl\vhdl** folder and open the **mem_controller.vhd** file (with WordPad). Locate the **User ports added here** line as shown in the following figure.



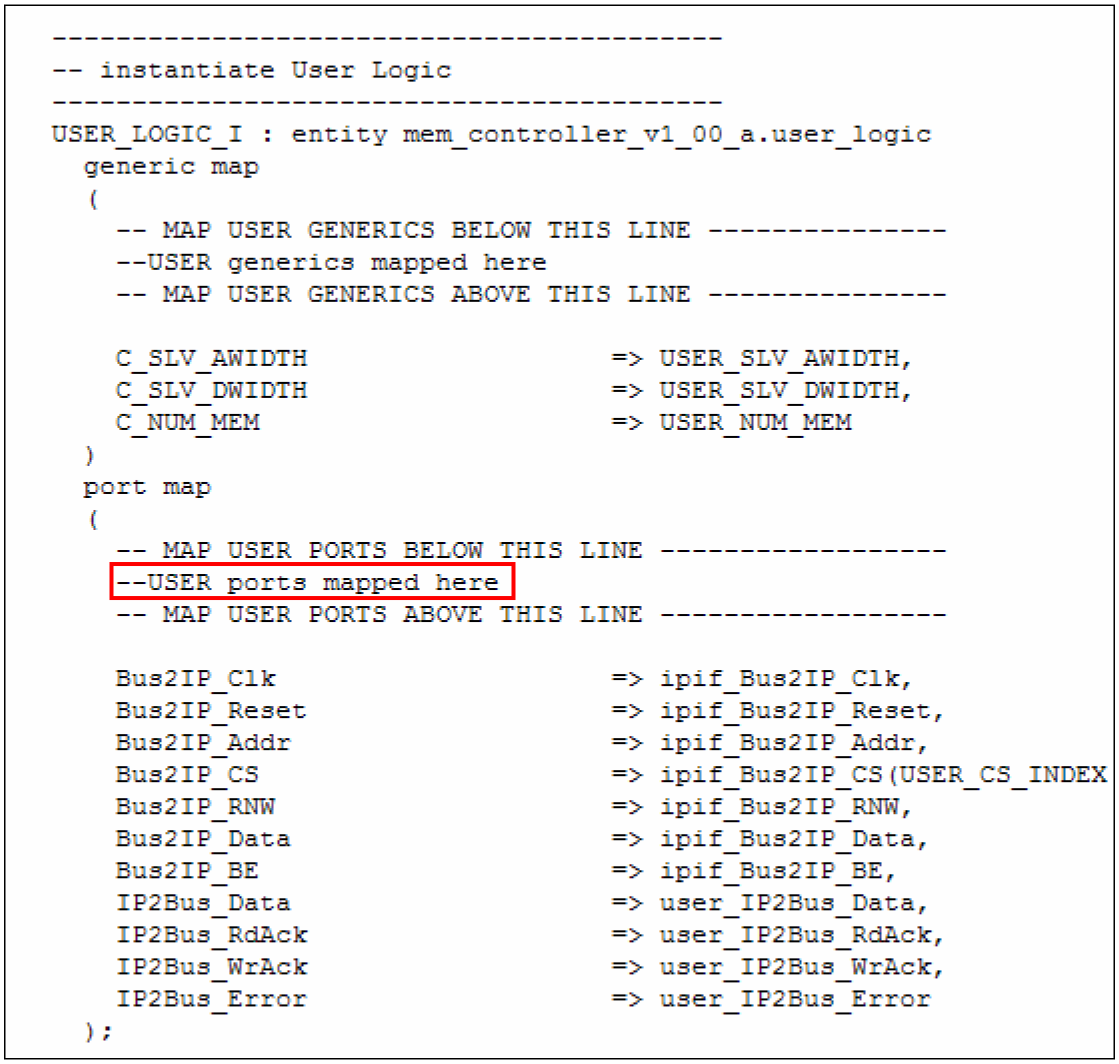
```
entity mem_controller is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_SPLB_AWIDTH                  : integer              := 32;
    C_SPLB_DWIDTH                  : integer              := 128;
    C_SPLB_NUM_MASTERS             : integer              := 8;
    C_SPLB_MID_WIDTH               : integer              := 3;
    C_SPLB_NATIVE_DWIDTH           : integer              := 32;
    C_SPLB_P2P                     : integer              := 0;
    C_SPLB_SUPPORT_BURSTS          : integer              := 0;
    C_SPLB_SMALLEST_MASTER         : integer              := 32;
    C_SPLB_CLK_PERIOD_PS           : integer              := 10000;
    C_FAMILY                       : string               := "virtex5";
    C_MEM0_BASEADDR                : std_logic_vector     := X"FFFFFFFF";
    C_MEM0_HIGHADDR                : std_logic_vector     := X"00000000"
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    SPLB_Clk                       : in  std_logic;
    SPLB_Rst                       : in  std_logic;
    PLB_ABus                       : in  std_logic_vector(0 to 31);
    PLB_UABus                      : in  std_logic_vector(0 to 31);
    PLB_PAValid                    : in  std_logic;
    PLB_SAValid                    : in  std_logic;
    PLB_rdPrim                     : in  std_logic;
    PLB_wrPrim                     : in  std_logic;
    PLB_masterID                   : in  std_logic_vector(0 to C_SPLB_MID
```

5. The **mem_controller** IP ports must be added to the **mem_controller.vhd** file. Replace the **User ports added here** line in the **mem_controller.vhd** file with the text shown in the following figure.



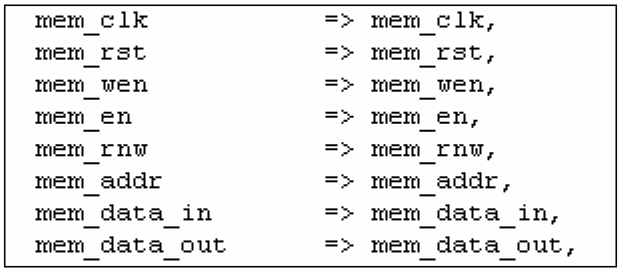
```
mem_clk              : out std_logic;
mem_rst              : out std_logic;
mem_wen              : out std_logic;
mem_en               : out std_logic;
mem_rnw              : out std_logic;
mem_addr             : out std_logic_vector(0 to 19);
mem_data_in          : in  std_logic_vector(0 to 31):= (others => '0');
mem_data_out         : out std_logic_vector(0 to 31);
```

6. Scroll down in the **mem_controller.vhd** file and locate the **User ports mapped here** line as shown in the following figure (toward the end of the file).



```
  ------------------------------------------
  -- instantiate User Logic
  ------------------------------------------
  USER_LOGIC_I : entity mem_controller_v1_00_a.user_logic
    generic map
    (
      -- MAP USER GENERICS BELOW THIS LINE ---------------
      --USER generics mapped here
      -- MAP USER GENERICS ABOVE THIS LINE ---------------

      C_SLV_AWIDTH                   => USER_SLV_AWIDTH,
      C_SLV_DWIDTH                   => USER_SLV_DWIDTH,
      C_NUM_MEM                      => USER_NUM_MEM
    )
    port map
    (
      -- MAP USER PORTS BELOW THIS LINE ------------------
      --USER ports mapped here
      -- MAP USER PORTS ABOVE THIS LINE ------------------

      Bus2IP_Clk                     => ipif_Bus2IP_Clk,
      Bus2IP_Reset                   => ipif_Bus2IP_Reset,
      Bus2IP_Addr                    => ipif_Bus2IP_Addr,
      Bus2IP_CS                      => ipif_Bus2IP_CS(USER_CS_INDEX
      Bus2IP_RNW                     => ipif_Bus2IP_RNW,
      Bus2IP_Data                    => ipif_Bus2IP_Data,
      Bus2IP_BE                      => ipif_Bus2IP_BE,
      IP2Bus_Data                    => user_IP2Bus_Data,
      IP2Bus_RdAck                   => user_IP2Bus_RdAck,
      IP2Bus_WrAck                   => user_IP2Bus_WrAck,
      IP2Bus_Error                   => user_IP2Bus_Error
    );
```

7. The **mem_controller** IP ports must be mapped to the ports of the **user_logic.vhd** component instantiated in the **mem_controller.vhd** file. Replace the **User ports mapped here** line in the **mem_controller.vhd** file with the text shown in the following figure.



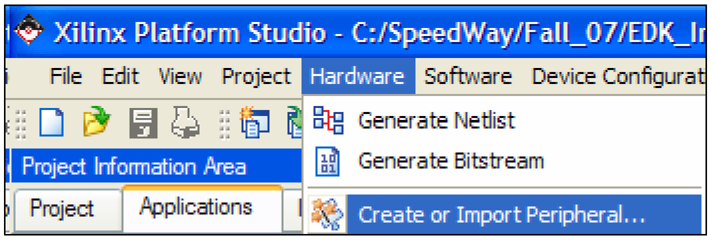
```
mem_clk            => mem_clk,
mem_rst            => mem_rst,
mem_wen            => mem_wen,
mem_en             => mem_en,
mem_rnw            => mem_rnw,
mem_addr           => mem_addr,
mem_data_in        => mem_data_in,
mem_data_out       => mem_data_out,
```

8. **Save** and **Close** the **mem_controller.vhd** file.

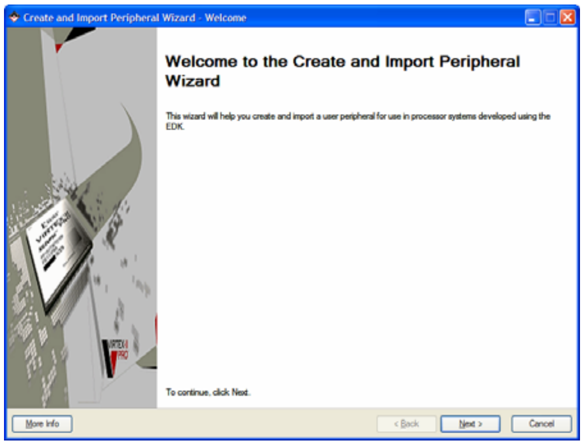
## Importing the Custom IP Using the IP Import Wizard

Up to this point, the custom memory controller IP has been created using the IP Creation wizard and the user logic portion of this IP has been implemented. The last step of implementing a custom IP is to import the IP using the IP Import wizard.

1. Invoke the **IP Import** wizard by selecting **Hardware > Create or Import Peripheral** from the XPS GUI.



2. The **Create and Import Peripheral Wizard** dialog box will appear. Click **Next** to continue.

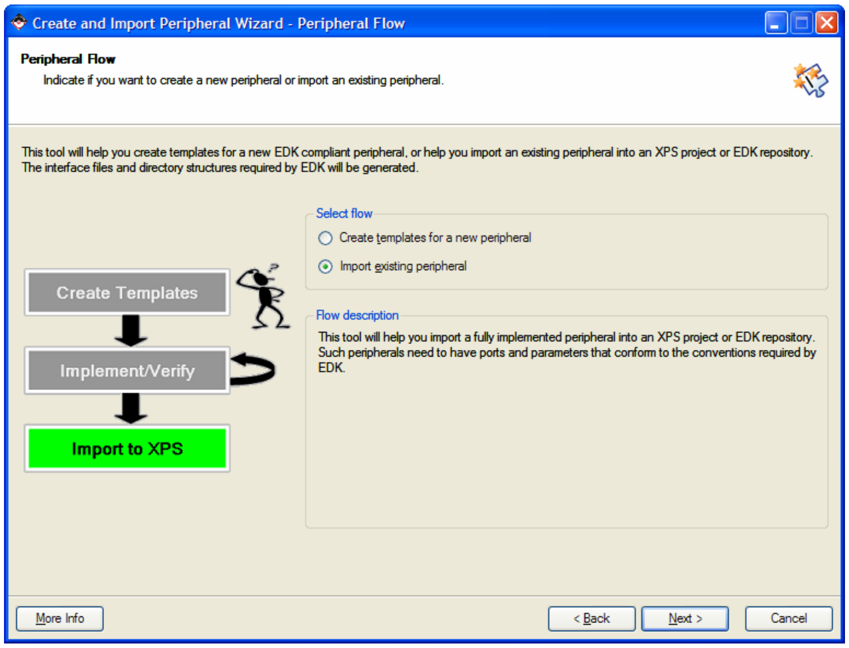


Welcome to the Create and Import Peripheral Wizard

This wizard will help you create and import a user peripheral for use in processor systems developed using the EDK.

To continue, click Next.

3. Select **Import existing peripheral** and click **Next** to continue.


Create and Import Peripheral Wizard - Peripheral Flow

**Peripheral Flow**

Indicate if you want to create a new peripheral or import an existing peripheral.

This tool will help you create templates for a new EDK compliant peripheral, or help you import an existing peripheral into an XPS project or EDK repository. The interface files and directory structures required by EDK will be generated.

Select flow

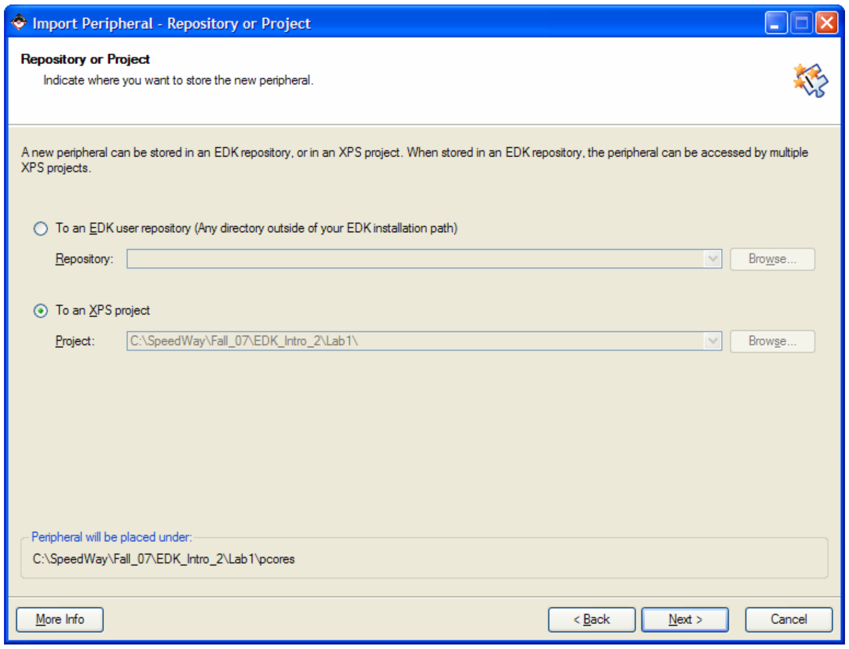
( ) Create templates for a new peripheral

(•) Import existing peripheral

Flow description

This tool will help you import a fully implemented peripheral into an XPS project or EDK repository. Such peripherals need to have ports and parameters that conform to the conventions required by EDK.

4. The following dialog box will appear indicating the custom IP core will be imported and stored in the **/pcores** folder of the Lab 2 project directory. Click **Next** to continue.


Import Peripheral - Repository or Project

**Repository or Project**

Indicate where you want to store the new peripheral.

A new peripheral can be stored in an EDK repository, or in an XPS project. When stored in an EDK repository, the peripheral can be accessed by multiple XPS projects.

( ) To an EDK user repository (Any directory outside of your EDK installation path)

Repository:

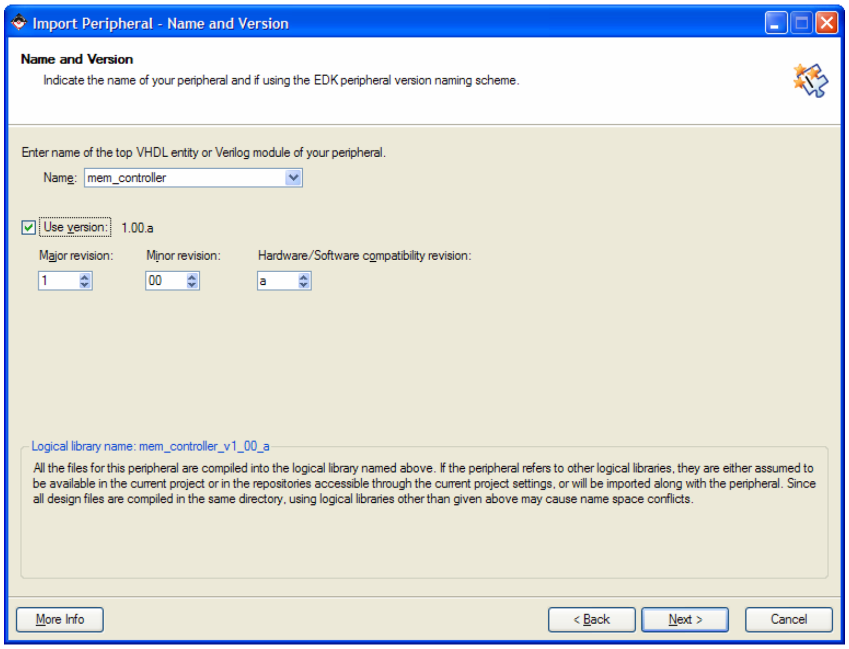
(•) To an XPS project

Project: C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\

Peripheral will be placed under:

C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores

5. In the **Import Peripheral - Name and Version** dialog box:
    a) Select the **mem_controller** from the drop-down box for the **Name** of the IP.
    b) Check the **Use Version: 1.00.a** box as shown in the following figure.
    c) Click **Next** to continue.



Import Peripheral - Name and Version

Name and Version

Indicate the name of your peripheral and if using the EDK peripheral version naming scheme.

Enter name of the top VHDL entity or Verilog module of your peripheral.

Name: mem_controller

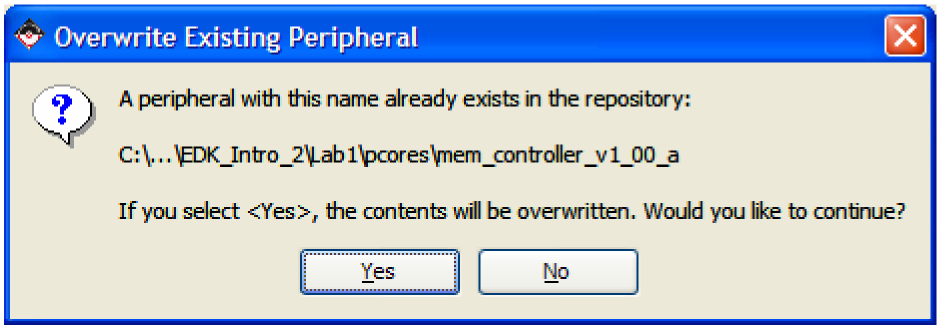
[✓] Use version: 1.00.a

Major revision: 1 Minor revision: 00 Hardware/Software compatibility revision: a

Logical library name: mem_controller_v1_00_a

All the files for this peripheral are compiled into the logical library named above. If the peripheral refers to other logical libraries, they are either assumed to be available in the current project or in the repositories accessible through the current project settings, or will be imported along with the peripheral. Since all design files are compiled in the same directory, using logical libraries other than given above may cause name space conflicts.

6. When the following dialog box appears, click **Yes** to continue.



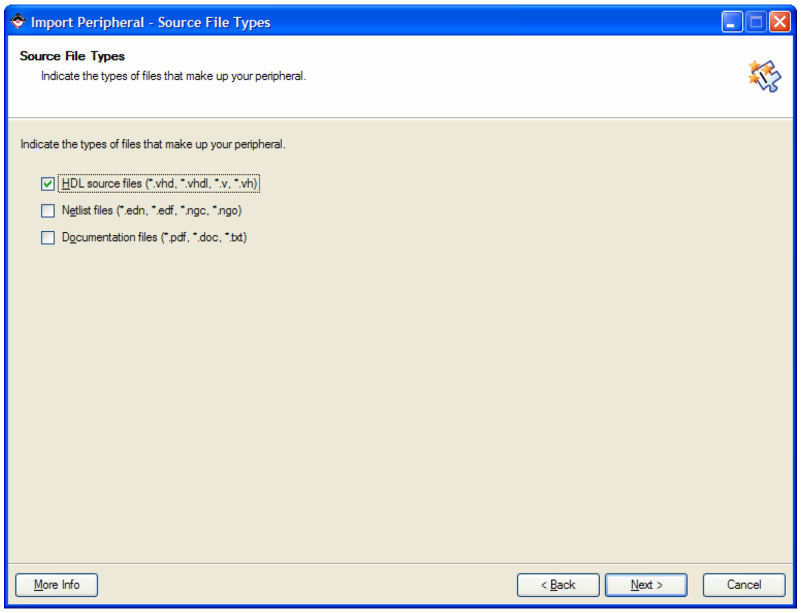
Overwrite Existing Peripheral

A peripheral with this name already exists in the repository:

C:\...\EDK_Intro_2\Lab1\pcores\mem_controller_v1_00_a

If you select <Yes>, the contents will be overwritten. Would you like to continue?

7. Check the **HDL Source Files** box, since no other source type is used to implement this IP core. Click **Next** to continue.



Import Peripheral - Source File Types

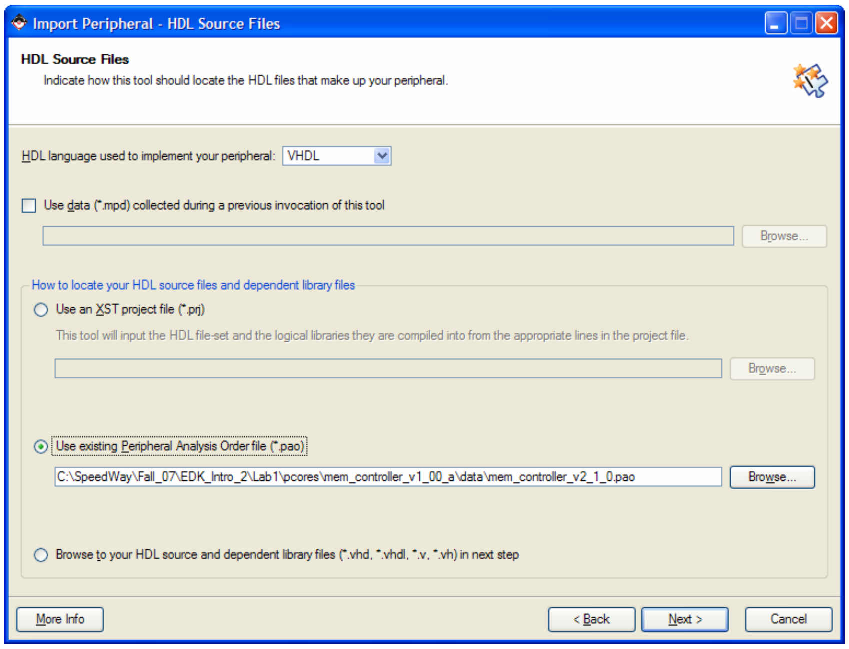
**Source File Types**

Indicate the types of files that make up your peripheral.

Indicate the types of files that make up your peripheral.

- [✓] HDL source files (*.vhd, *.vhdl, *.v, *.vh)
- [ ] Netlist files (*.edn, *.edf, *.ngc, *.ngo)
- [ ] Documentation files (*.pdf, *.doc, *.txt)

8. In the **Import Peripheral - HDL Source Files** dialog box:
   a) Select **VHDL** for the IP implementation.
   b) Select **Use existing Peripheral Analysis Order file (*.pao)**.
   c) Browse to the **C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores \mem_controller_v1_00_a\data** folder, select the **mem_controller_v2_1_0.pao** file and click **Open**.
   d) Click **Next** to continue.



Import Peripheral - HDL Source Files

**HDL Source Files**

Indicate how this tool should locate the HDL files that make up your peripheral.

HDL language used to implement your peripheral: VHDL

[ ] Use data (*.mpd) collected during a previous invocation of this tool

How to locate your HDL source files and dependent library files

○ Use an XST project file (*.prj)

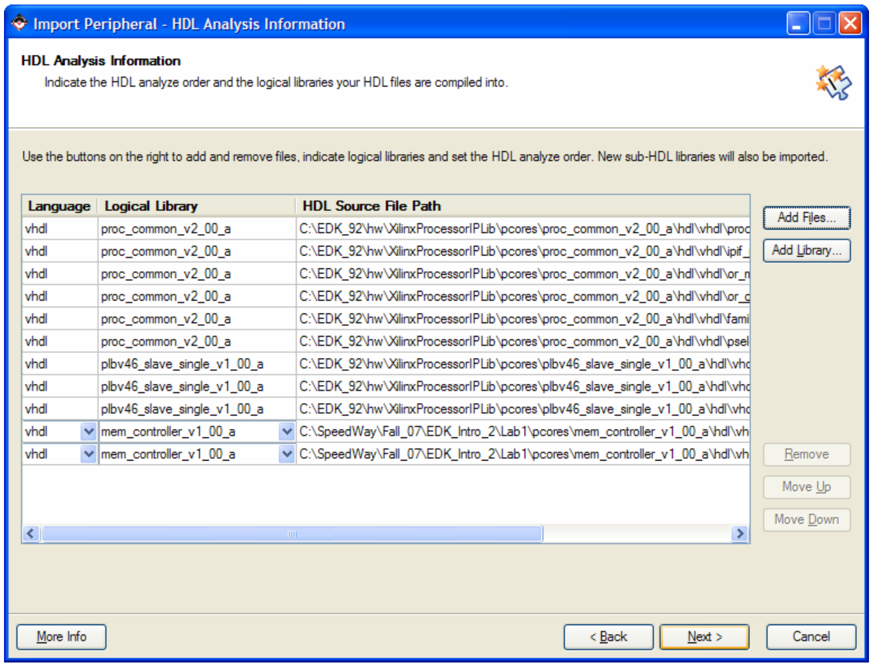
This tool will input the HDL file-set and the logical libraries they are compiled into from the appropriate lines in the project file.

● Use existing Peripheral Analysis Order file (*.pao)

C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores\mem_controller_v1_00_a\data\mem_controller_v2_1_0.pao

○ Browse to your HDL source and dependent library files (*.vhd, *.vhdl, *.v, *.vh) in next step

9. The **HDL Analysis Information** dialog box will appear. Click **Next** to continue.



Import Peripheral - HDL Analysis Information

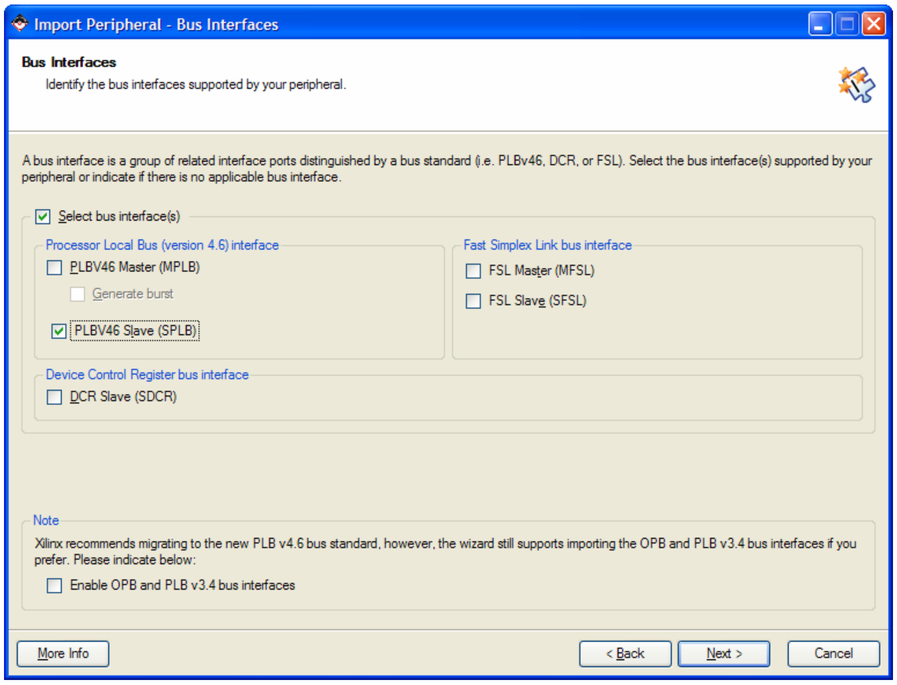
**HDL Analysis Information**

Indicate the HDL analyze order and the logical libraries your HDL files are compiled into.

Use the buttons on the right to add and remove files, indicate logical libraries and set the HDL analyze order. New sub-HDL libraries will also be imported.

| Language | Logical Library | HDL Source File Path |
|---|---|---|
| vhdl | proc_common_v2_00_a | C:\EDK_92\hw\XilinxProcessorIPLib\pcores\proc_common_v2_00_a\hdl\vhdl\proc |
| vhdl | proc_common_v2_00_a | C:\EDK_92\hw\XilinxProcessorIPLib\pcores\proc_common_v2_00_a\hdl\vhdl\ipif_ |
| vhdl | proc_common_v2_00_a | C:\EDK_92\hw\XilinxProcessorIPLib\pcores\proc_common_v2_00_a\hdl\vhdl\or_n |
| vhdl | proc_common_v2_00_a | C:\EDK_92\hw\XilinxProcessorIPLib\pcores\proc_common_v2_00_a\hdl\vhdl\or_g |
| vhdl | proc_common_v2_00_a | C:\EDK_92\hw\XilinxProcessorIPLib\pcores\proc_common_v2_00_a\hdl\vhdl\fami |
| vhdl | proc_common_v2_00_a | C:\EDK_92\hw\XilinxProcessorIPLib\pcores\proc_common_v2_00_a\hdl\vhdl\psel |
| vhdl | plbv46_slave_single_v1_00_a | C:\EDK_92\hw\XilinxProcessorIPLib\pcores\plbv46_slave_single_v1_00_a\hdl\vhc |
| vhdl | plbv46_slave_single_v1_00_a | C:\EDK_92\hw\XilinxProcessorIPLib\pcores\plbv46_slave_single_v1_00_a\hdl\vhc |
| vhdl | plbv46_slave_single_v1_00_a | C:\EDK_92\hw\XilinxProcessorIPLib\pcores\plbv46_slave_single_v1_00_a\hdl\vhc |
| vhdl | mem_controller_v1_00_a | C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores\mem_controller_v1_00_a\hdl\vh |
| vhdl | mem_controller_v1_00_a | C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores\mem_controller_v1_00_a\hdl\vh |

10. Since the **mem_controller** IP is designed to be attached to the **PLB**, check the **PLBV46 Slave (SPLB)**. Click **Next** to continue.



Import Peripheral - Bus Interfaces

**Bus Interfaces**

Identify the bus interfaces supported by your peripheral.

A bus interface is a group of related interface ports distinguished by a bus standard (i.e. PLBv46, DCR, or FSL). Select the bus interface(s) supported by your peripheral or indicate if there is no applicable bus interface.

[✓ Select bus interface(s)]

Processor Local Bus (version 4.6) interface
- PLBV46 Master (MPLB)
- Generate burst
- [✓ PLBV46 Slave (SPLB)]

Fast Simplex Link bus interface
- FSL Master (MFSL)
- FSL Slave (SFSL)

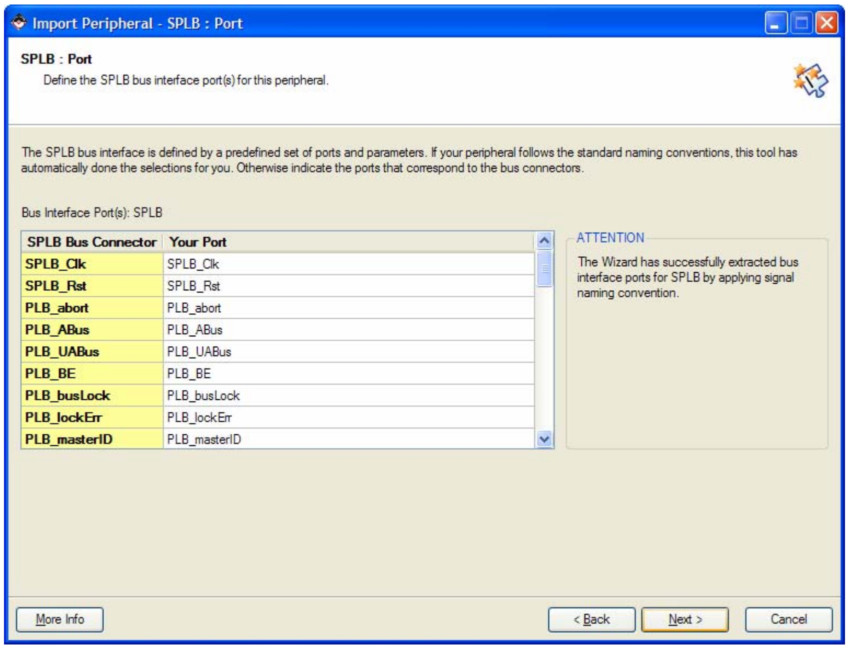
Device Control Register bus interface
- DCR Slave (SDCR)

Note

Xilinx recommends migrating to the new PLB v4.6 bus standard, however, the wizard still supports importing the OPB and PLB v3.4 bus interfaces if you prefer. Please indicate below:

- Enable OPB and PLB v3.4 bus interfaces

11. Since the **PLB** ports have not been modified in the **mem_controller.vhd** file, leave the default net names and click **Next** to continue.



**SPLB : Port**

Define the SPLB bus interface port(s) for this peripheral.

The SPLB bus interface is defined by a predefined set of ports and parameters. If your peripheral follows the standard naming conventions, this tool has automatically done the selections for you. Otherwise indicate the ports that correspond to the bus connectors.

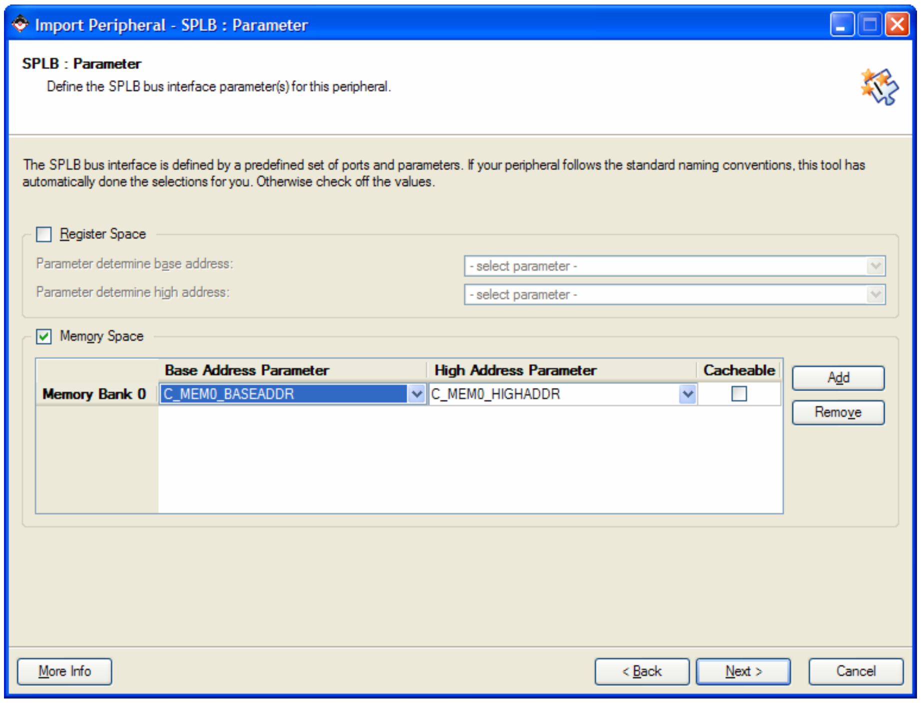
Bus Interface Port(s): SPLB

| SPLB Bus Connector | Your Port |
|---|---|
| SPLB_Clk | SPLB_Clk |
| SPLB_Rst | SPLB_Rst |
| PLB_abort | PLB_abort |
| PLB_ABus | PLB_ABus |
| PLB_UABus | PLB_UABus |
| PLB_BE | PLB_BE |
| PLB_busLock | PLB_busLock |
| PLB_lockErr | PLB_lockErr |
| PLB_masterID | PLB_masterID |

ATTENTION

The Wizard has successfully extracted bus interface ports for SPLB by applying signal naming convention.

12. Since no registers are implemented in the custom IP, **un-check** the **Register Space** box.
    a) Check the **Memory Space**.
    b) Click **Next** to continue.



**SPLB : Parameter**

Define the SPLB bus interface parameter(s) for this peripheral.

The SPLB bus interface is defined by a predefined set of ports and parameters. If your peripheral follows the standard naming conventions, this tool has automatically done the selections for you. Otherwise check off the values.

☐ Register Space

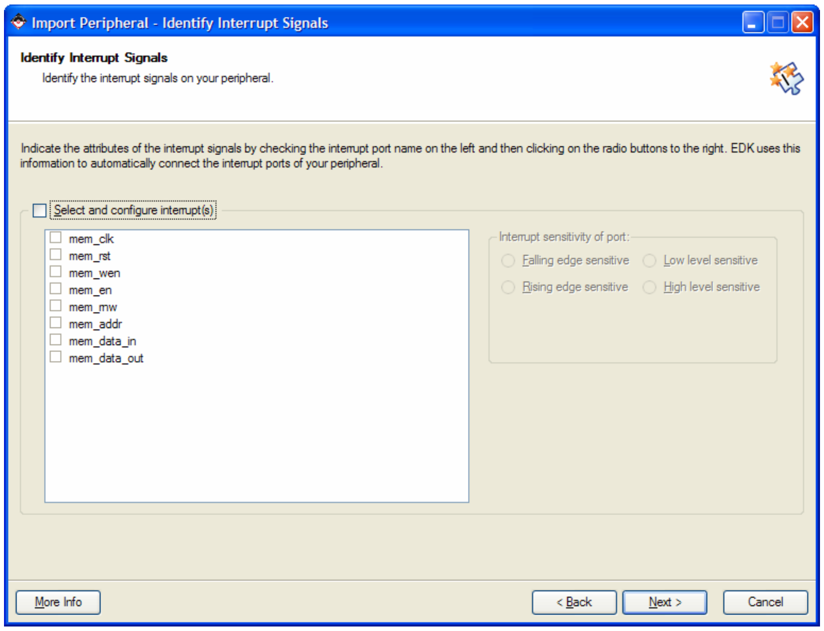
Parameter determine base address: - select parameter -

Parameter determine high address: - select parameter -

☑ Memory Space

|  | Base Address Parameter | High Address Parameter | Cacheable |
|---|---|---|---|
| Memory Bank 0 | C_MEM0_BASEADDR | C_MEM0_HIGHADDR | ☐ |

13. There are no interrupt signals associated with the **mem_controller** IP. Un-check the **Select and configure interrupt(s)** box and click **Next** to continue.


Import Peripheral - Identify Interrupt Signals

Identify Interrupt Signals

Identify the interrupt signals on your peripheral.

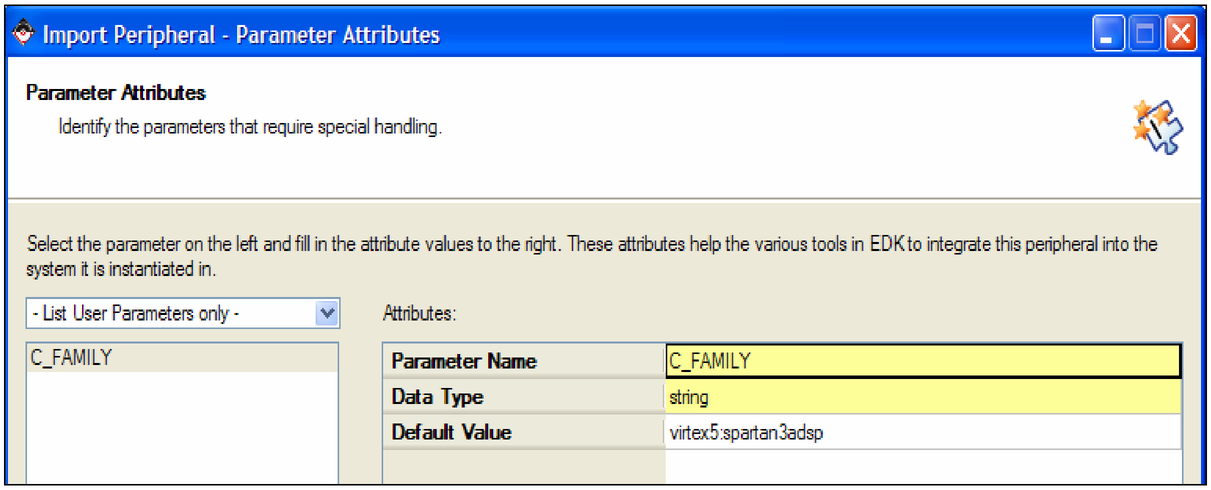
Indicate the attributes of the interrupt signals by checking the interrupt port name on the left and then clicking on the radio buttons to the right. EDK uses this information to automatically connect the interrupt ports of your peripheral.

☐ Select and configure interrupt(s)

☐ mem_clk
☐ mem_rst
☐ mem_wen
☐ mem_en
☐ mem_mw
☐ mem_addr
☐ mem_data_in
☐ mem_data_out

Interrupt sensitivity of port:

14. In the **Import Peripheral - Parameter Attributes** dialog box:
    a) Click on **C_FAMILY**.
    b) Double-click in the **Default Value** field and add **spartan3adsp** to the supported architecture for the **mem_controller** IP in addition to the **virtex5** (in the **Default Value** field, the **virtex5** and **spartan3adsp** are separated by "**:**"). After modifying this field, hit the return key.
    c) Click **Next** to continue.


Import Peripheral - Parameter Attributes

Parameter Attributes

Identify the parameters that require special handling.

Select the parameter on the left and fill in the attribute values to the right. These attributes help the various tools in EDK to integrate this peripheral into the system it is instantiated in.

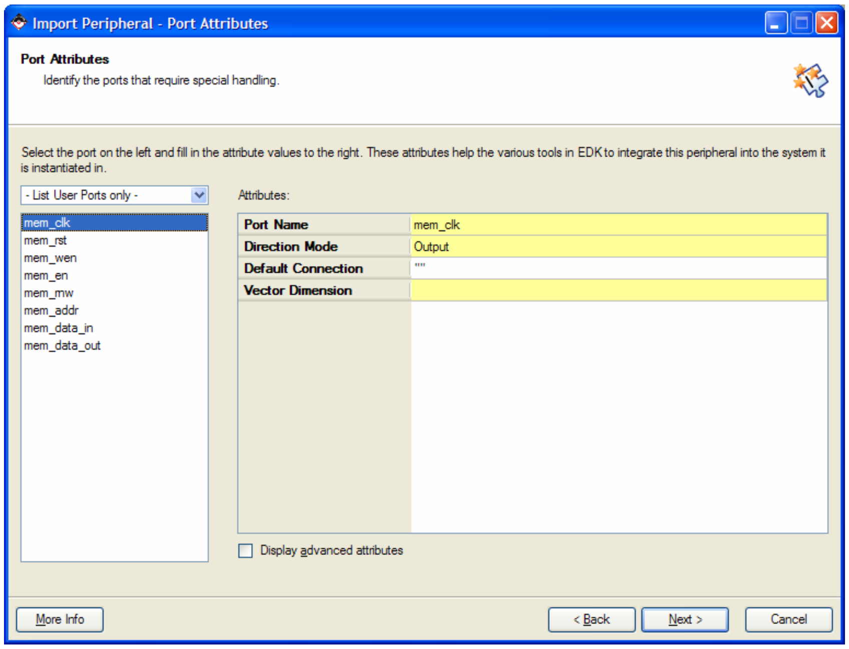
- List User Parameters only -

C_FAMILY

Attributes:

| | |
|---|---|
| Parameter Name | C_FAMILY |
| Data Type | string |
| Default Value | virtex5:spartan3adsp |

15. No modifications will be made to the default port attributes. Click **Next** to continue (to view the default attributes for a port, click on a port as shown in the following figure).



Import Peripheral - Port Attributes
Port Attributes
Identify the ports that require special handling.

Select the port on the left and fill in the attribute values to the right. These attributes help the various tools in EDK to integrate this peripheral into the system it is instantiated in.

- List User Ports only -

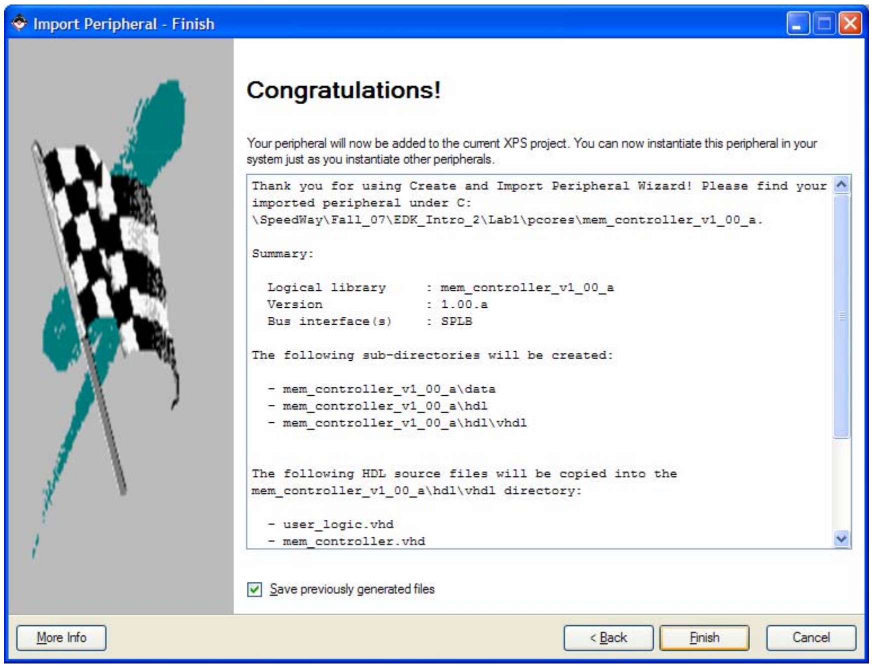
mem_clk
mem_rst
mem_wen
mem_en
mem_rnw
mem_addr
mem_data_in
mem_data_out

Attributes:

| | |
|---|---|
| Port Name | mem_clk |
| Direction Mode | Output |
| Default Connection | "" |
| Vector Dimension | |

☐ Display advanced attributes

16. You have successfully imported the **mem_controller** IP. Click **Finish** to continue.



Congratulations!

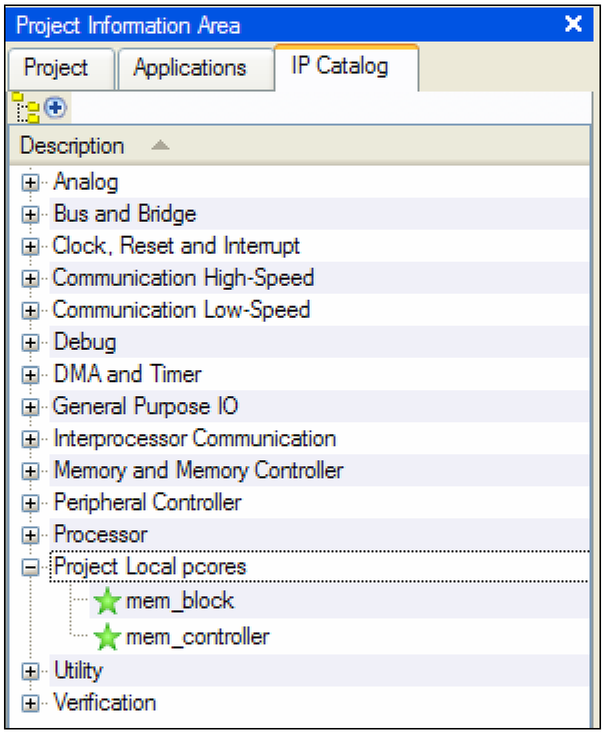
Your peripheral will now be added to the current XPS project. You can now instantiate this peripheral in your system just as you instantiate other peripherals.

```
Thank you for using Create and Import Peripheral Wizard! Please find your
imported peripheral under C:
\SpeedWay\Fall_07\EDK_Intro_2\Lab1\pcores\mem_controller_v1_00_a.

Summary:

  Logical library     : mem_controller_v1_00_a
  Version             : 1.00.a
  Bus interface(s)    : SPLB

The following sub-directories will be created:

  - mem_controller_v1_00_a\data
  - mem_controller_v1_00_a\hdl
  - mem_controller_v1_00_a\hdl\vhdl


The following HDL source files will be copied into the
mem_controller_v1_00_a\hdl\vhdl directory:

  - user_logic.vhd
  - mem_controller.vhd
```

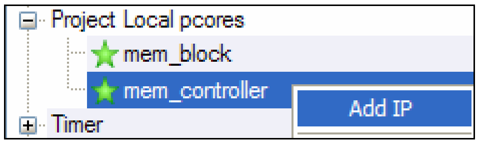
☑ Save previously generated files

## Adding the Custom IP Core to the Design

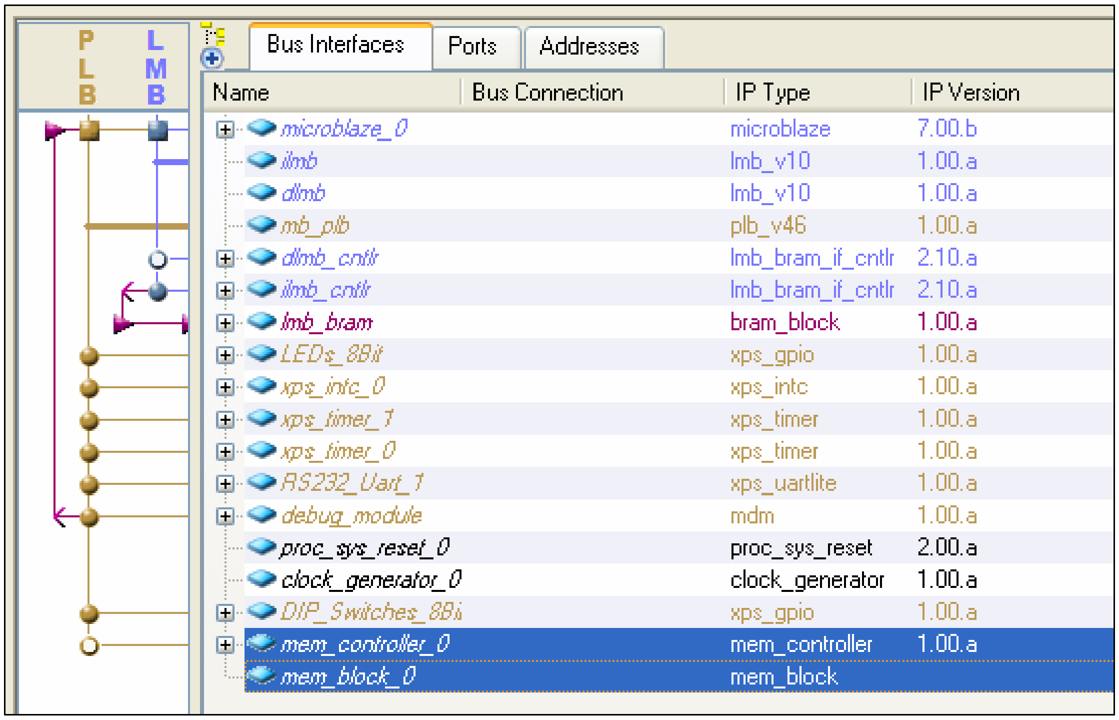
1. Select **Project > Rescan User Repositories** from the XPS GUI to rescan the user pcores.
   a. Click on the **IP Catalog** tab and expand the **Project Local pcores** IP category as shown in the following figure. The **mem_block** as well as the **mem_controller** IP cores are added to the **Project Local pcores** category.



2. Right-click on the **mem_controller** IP and select **Add IP** as shown in the following figure.

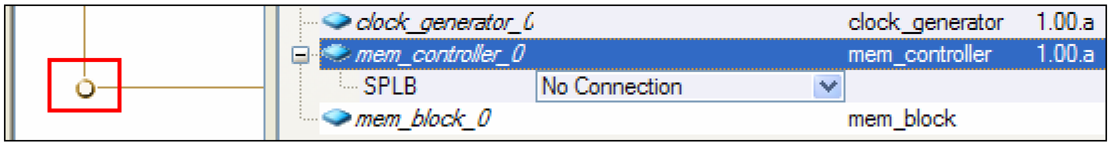


3. Use the same method to add the **mem_block** IP core to the design. After adding the above IP cores to the design, the XPS **System Assembly View** of the project should look similar to the one shown in the following figure.



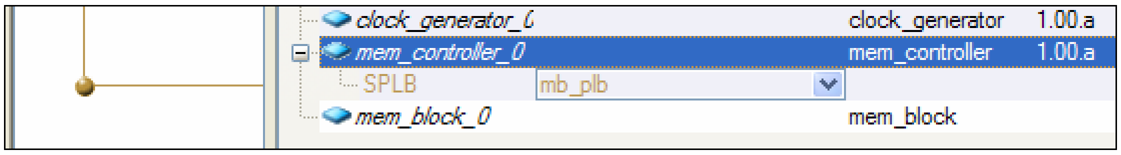
| Name | Bus Connection | IP Type | IP Version |
|---|---|---|---|
| microblaze_0 | | microblaze | 7.00.b |
| ilmb | | lmb_v10 | 1.00.a |
| dlmb | | lmb_v10 | 1.00.a |
| mb_plb | | plb_v46 | 1.00.a |
| dlmb_cntlr | | lmb_bram_if_cntlr | 2.10.a |
| ilmb_cntlr | | lmb_bram_if_cntlr | 2.10.a |
| lmb_bram | | bram_block | 1.00.a |
| LEDs_8Bit | | xps_gpio | 1.00.a |
| xps_intc_0 | | xps_intc | 1.00.a |
| xps_timer_1 | | xps_timer | 1.00.a |
| xps_timer_0 | | xps_timer | 1.00.a |
| RS232_Uart_1 | | xps_uartlite | 1.00.a |
| debug_module | | mdm | 1.00.a |
| proc_sys_reset_0 | | proc_sys_reset | 2.00.a |
| clock_generator_0 | | clock_generator | 1.00.a |
| DIP_Switches_8Bi | | xps_gpio | 1.00.a |
| mem_controller_0 | | mem_controller | 1.00.a |
| mem_block_0 | | mem_block | |

4. The next step of adding an IP core is to connect the IP to the processor bus. Expand the **mem_controller** IP core and click on the **SPLB** connection circle as shown in the following figure.



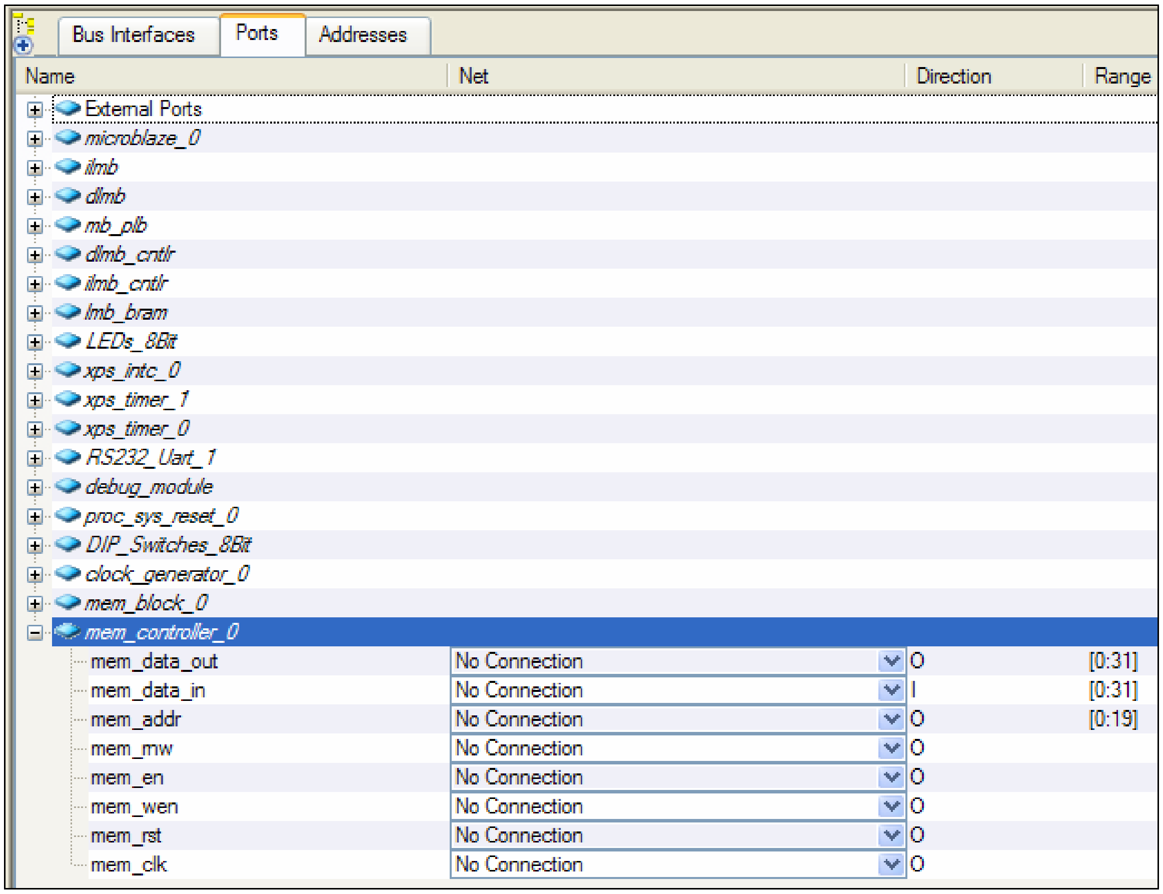
| | | | |
|---|---|---|---|
| clock_generator_( | | clock_generator | 1.00.a |
| mem_controller_0 | | mem_controller | 1.00.a |
| SPLB | No Connection | | |
| mem_block_0 | | mem_block | |

5. After performing the above step, the **mem_controller** IP will be connected to the **mb_plb** as shown in the following figure.



| | | | |
|---|---|---|---|
| clock_generator_( | | clock_generator | 1.00.a |
| mem_controller_0 | | mem_controller | 1.00.a |
| SPLB | mb_plb | | |
| mem_block_0 | | mem_block | |

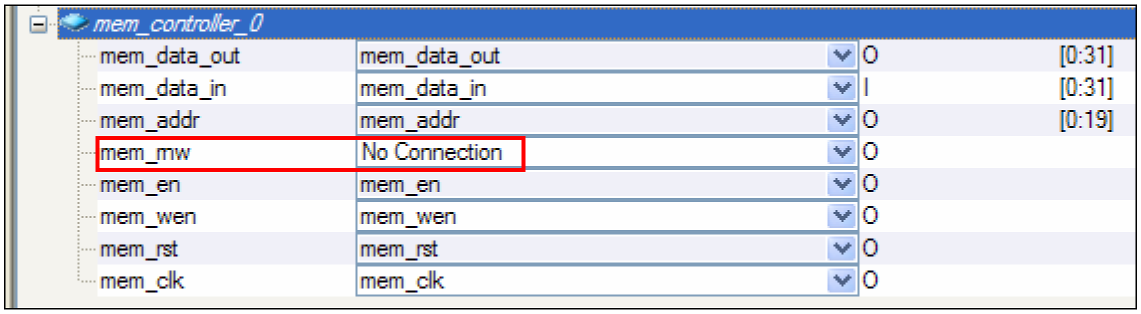
6. The next step is to connect the ports of the IP core. Click on the **Ports** tab and expand the **mem_controller** IP to view its ports as shown in the following figure.



| Name | Net | Direction | Range |
|---|---|---|---|
| External Ports | | | |
| microblaze_0 | | | |
| ilmb | | | |
| dlmb | | | |
| mb_plb | | | |
| dlmb_cntlr | | | |
| ilmb_cntlr | | | |
| lmb_bram | | | |
| LEDs_8Bit | | | |
| xps_intc_0 | | | |
| xps_timer_1 | | | |
| xps_timer_0 | | | |
| RS232_Uart_1 | | | |
| debug_module | | | |
| proc_sys_reset_0 | | | |
| DIP_Switches_8Bit | | | |
| clock_generator_0 | | | |
| mem_block_0 | | | |
| mem_controller_0 | | | |
| mem_data_out | No Connection | O | [0:31] |
| mem_data_in | No Connection | I | [0:31] |
| mem_addr | No Connection | O | [0:19] |
| mem_mw | No Connection | O | |
| mem_en | No Connection | O | |
| mem_wen | No Connection | O | |
| mem_rst | No Connection | O | |
| mem_clk | No Connection | O | |

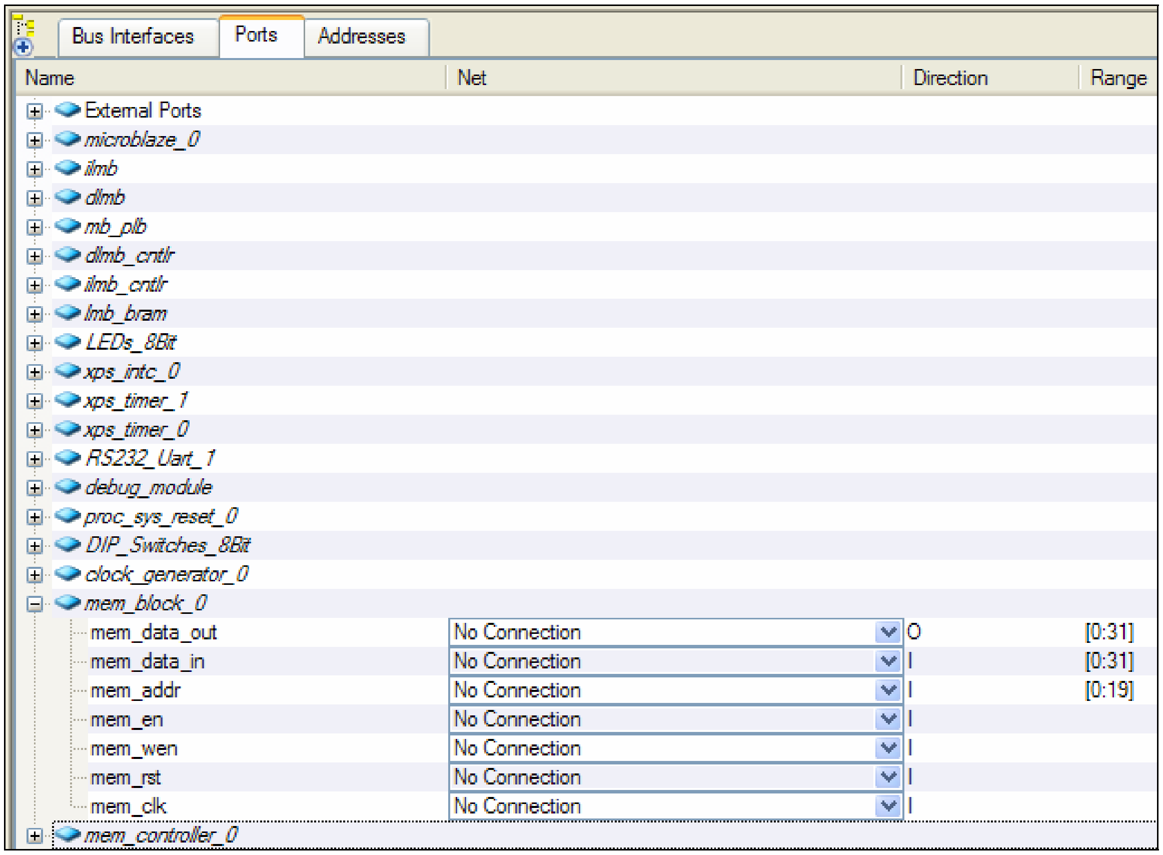
7. Click in the **No Connection** field of the **mem_data_out** port to select it, click again in the **No Connection** field and assign the **mem_data_out** net name to it, and hit the return key to continue.

   Repeat the same steps to assign net names to the rest of the **mem_controller** ports (please leave the **mem_rnw** signal unconnected as this signal will not be used for this lab). After performing the above step, the **mem_controller** net names should look as shown in the following figure.



| mem_controller_0 | | | |
|---|---|---|---|
| mem_data_out | mem_data_out | O | [0:31] |
| mem_data_in | mem_data_in | I | [0:31] |
| mem_addr | mem_addr | O | [0:19] |
| mem_mw | No Connection | O | |
| mem_en | mem_en | O | |
| mem_wen | mem_wen | O | |
| mem_rst | mem_rst | O | |
| mem_clk | mem_clk | O | |

8. Expand the **mem_block** IP core to view its ports.

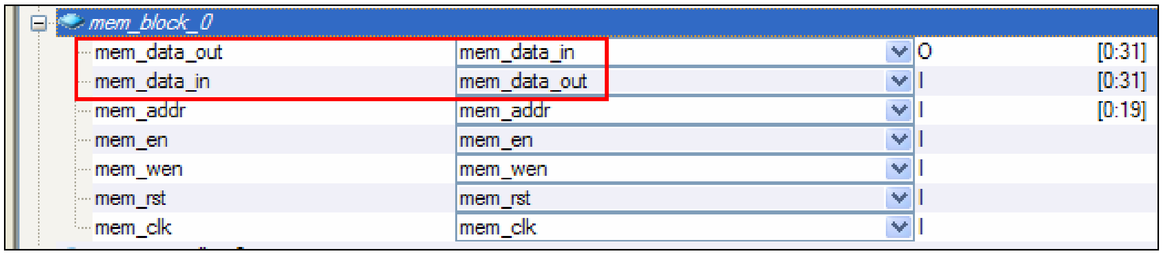


Bus Interfaces | Ports | Addresses

| Name | Net | Direction | Range |
|---|---|---|---|
| External Ports | | | |
| microblaze_0 | | | |
| ilmb | | | |
| dlmb | | | |
| mb_plb | | | |
| dlmb_cntlr | | | |
| ilmb_cntlr | | | |
| lmb_bram | | | |
| LEDs_8Bit | | | |
| xps_intc_0 | | | |
| xps_timer_1 | | | |
| xps_timer_0 | | | |
| RS232_Uart_1 | | | |
| debug_module | | | |
| proc_sys_reset_0 | | | |
| DIP_Switches_8Bit | | | |
| clock_generator_0 | | | |
| mem_block_0 | | | |
| mem_data_out | No Connection | O | [0:31] |
| mem_data_in | No Connection | I | [0:31] |
| mem_addr | No Connection | I | [0:19] |
| mem_en | No Connection | I | |
| mem_wen | No Connection | I | |
| mem_rst | No Connection | I | |
| mem_clk | No Connection | I | |
| mem_controller_0 | | | |

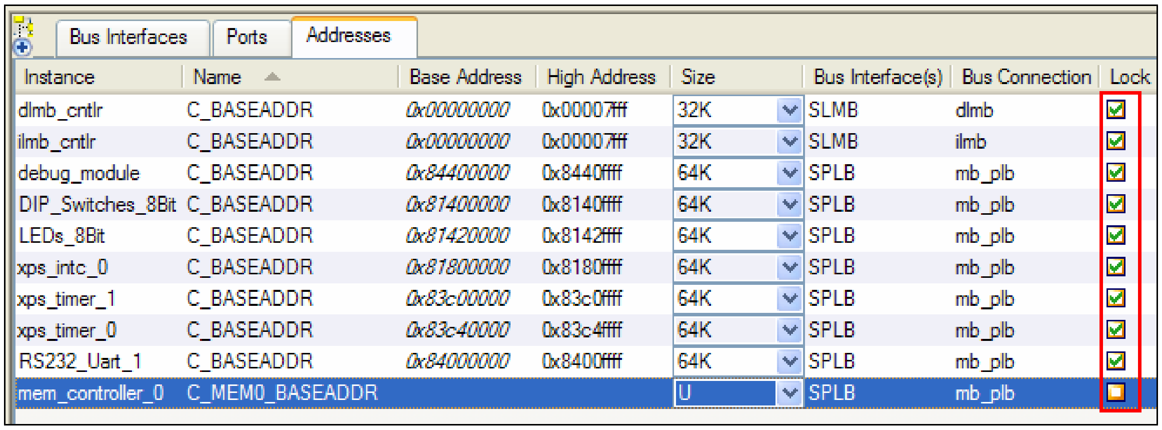
9. Assign net names to the **mem_block** IP ports as shown in the following figure.

**Note**: The **mem_data_in** of the **mem_block** IP core must be connected to the **mem_data_out** bus (output of the mem_controller IP) and the **mem_data_out** of the **mem_block** IP core must be connected to the **mem_data_in** bus (input of the mem_controller IP).



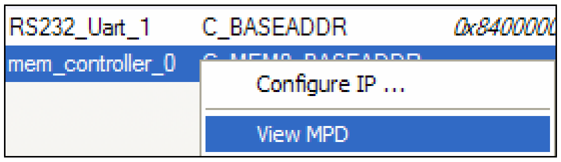
| mem_block_0 | | | |
|---|---|---|---|
| mem_data_out | mem_data_in | O | [0:31] |
| mem_data_in | mem_data_out | I | [0:31] |
| mem_addr | mem_addr | I | [0:19] |
| mem_en | mem_en | I | |
| mem_wen | mem_wen | I | |
| mem_rst | mem_rst | I | |
| mem_clk | mem_clk | I | |

10. Since the **mem_controller** IP connects to the processor bus, it must reside some where in the processor memory space. So, the next step is to assign address space to the IP core. Click on the **Addresses** tab and lock the address space for all IP cores with the exception of the **mem_controller** IP core as shown in the following figure.

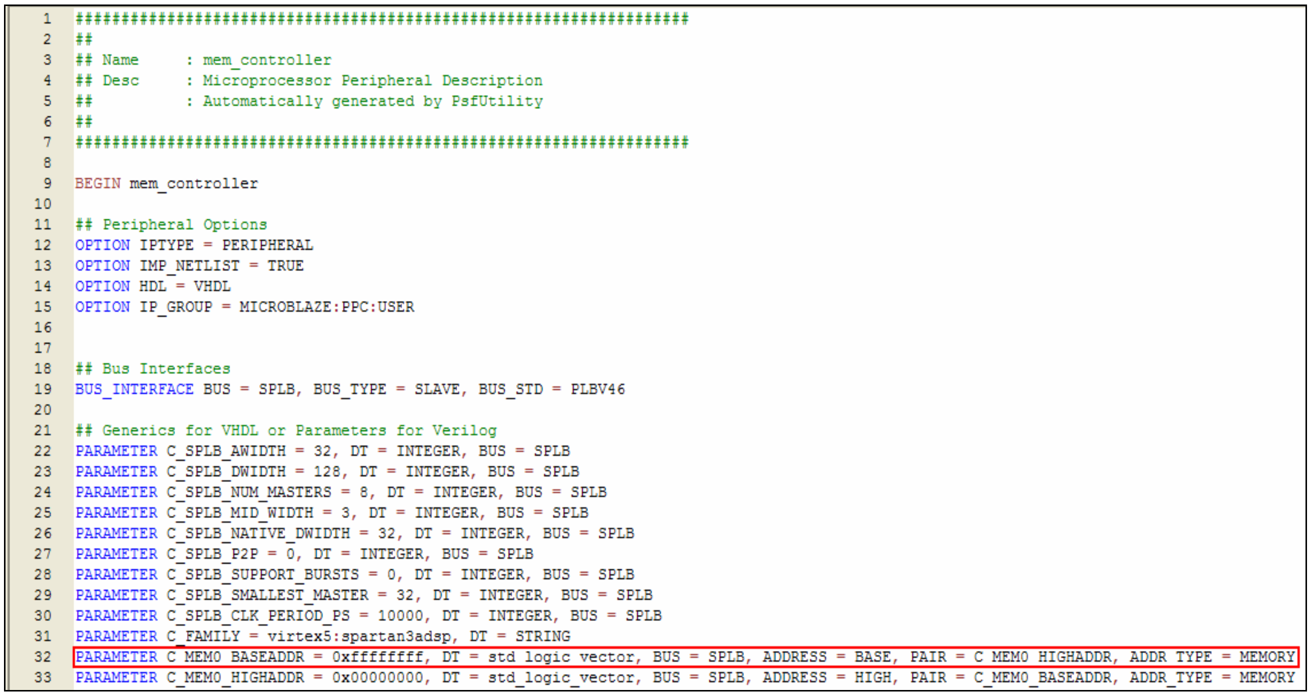


| Instance | Name | Base Address | High Address | Size | Bus Interface(s) | Bus Connection | Lock |
|---|---|---|---|---|---|---|---|
| dlmb_cntlr | C_BASEADDR | 0x00000000 | 0x00007fff | 32K | SLMB | dlmb | ✓ |
| ilmb_cntlr | C_BASEADDR | 0x00000000 | 0x00007fff | 32K | SLMB | ilmb | ✓ |
| debug_module | C_BASEADDR | 0x84400000 | 0x8440ffff | 64K | SPLB | mb_plb | ✓ |
| DIP_Switches_8Bit | C_BASEADDR | 0x81400000 | 0x8140ffff | 64K | SPLB | mb_plb | ✓ |
| LEDs_8Bit | C_BASEADDR | 0x81420000 | 0x8142ffff | 64K | SPLB | mb_plb | ✓ |
| xps_intc_0 | C_BASEADDR | 0x81800000 | 0x8180ffff | 64K | SPLB | mb_plb | ✓ |
| xps_timer_1 | C_BASEADDR | 0x83c00000 | 0x83c0ffff | 64K | SPLB | mb_plb | ✓ |
| xps_timer_0 | C_BASEADDR | 0x83c40000 | 0x83c4ffff | 64K | SPLB | mb_plb | ✓ |
| RS232_Uart_1 | C_BASEADDR | 0x84000000 | 0x8400ffff | 64K | SPLB | mb_plb | ✓ |
| mem_controller_0 | C_MEM0_BASEADDR | | | U | SPLB | mb_plb | ☐ |

11. Prior to assigning address space to the **mem_controller** IP, the MPD file for this IP must be modified to include minimum address size associated with this IP. Right-click on the **mem_controller** IP and select **View MPD** to open the MPD file.

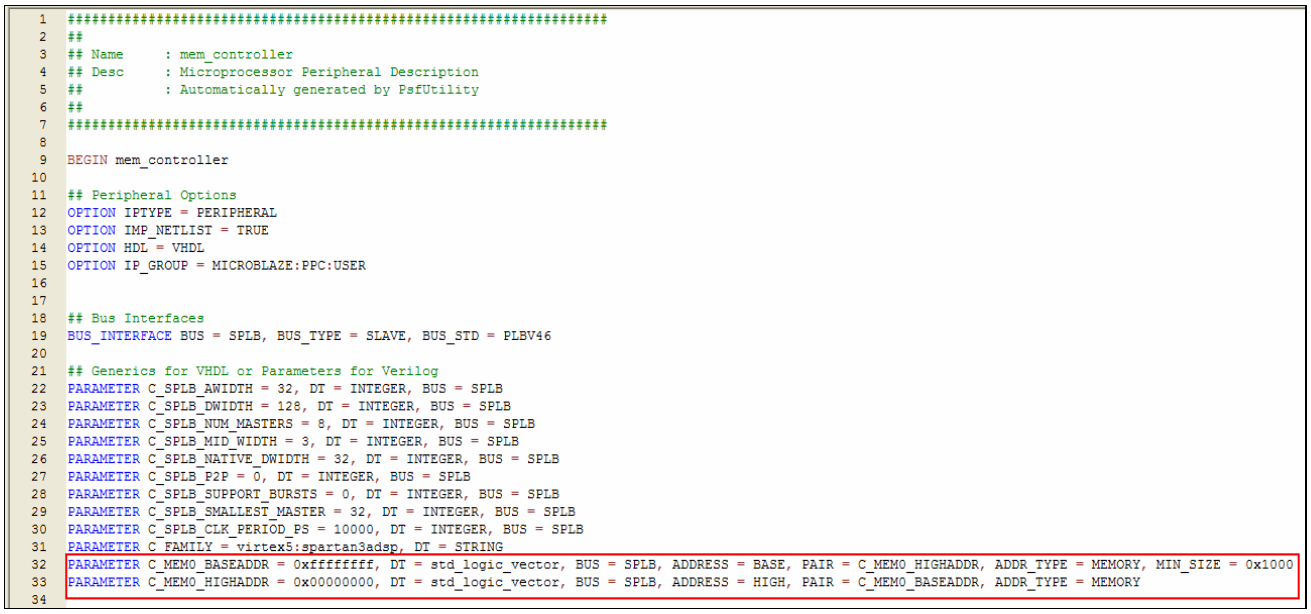


12. Locate the line in the MPD file where the base address for the custom IP is defined as shown in the following figure.



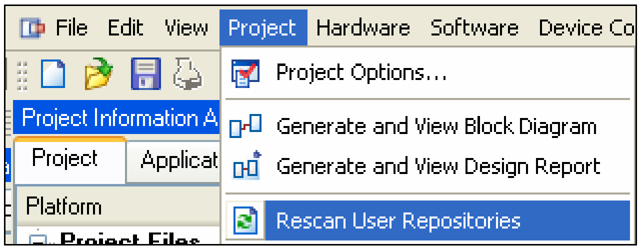
```
######################################################################
##
## Name     : mem_controller
## Desc     : Microprocessor Peripheral Description
##          : Automatically generated by PsfUtility
##
######################################################################

BEGIN mem_controller

## Peripheral Options
OPTION IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
OPTION IP_GROUP = MICROBLAZE:PPC:USER


## Bus Interfaces
BUS_INTERFACE BUS = SPLB, BUS_TYPE = SLAVE, BUS_STD = PLBV46

## Generics for VHDL or Parameters for Verilog
PARAMETER C_SPLB_AWIDTH = 32, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_DWIDTH = 128, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_NUM_MASTERS = 8, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_MID_WIDTH = 3, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_NATIVE_DWIDTH = 32, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_P2P = 0, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_SUPPORT_BURSTS = 0, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_SMALLEST_MASTER = 32, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_CLK_PERIOD_PS = 10000, DT = INTEGER, BUS = SPLB
PARAMETER C_FAMILY = virtex5:spartan3adsp, DT = STRING
PARAMETER C_MEM0_BASEADDR = 0xffffffff, DT = std_logic_vector, BUS = SPLB, ADDRESS = BASE, PAIR = C_MEM0_HIGHADDR, ADDR_TYPE = MEMORY
PARAMETER C_MEM0_HIGHADDR = 0x00000000, DT = std_logic_vector, BUS = SPLB, ADDRESS = HIGH, PAIR = C_MEM0_BASEADDR, ADDR_TYPE = MEMORY
```

13. Edit the above line in the MPD file and add **, MIN_SIZE = 0x1000** (you need to enter a comma followed by the **MIN_SIZE = 0x1000** text) to the end of the line as shown in the following figure. Save the MPD file by selecting **File > Save** from the XPS GUI. **Close** the MPD file via black X in the upper right-hand corner of the screen (do not click on the red X or you will close the XPS GUI).

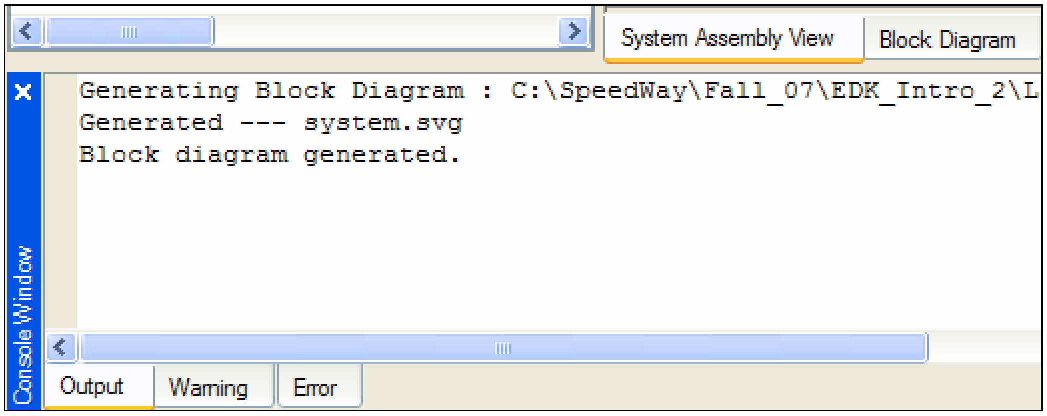


```
######################################################################
##
## Name     : mem_controller
## Desc     : Microprocessor Peripheral Description
##          : Automatically generated by PsfUtility
##
######################################################################

BEGIN mem_controller

## Peripheral Options
OPTION IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
OPTION IP_GROUP = MICROBLAZE:PPC:USER


## Bus Interfaces
BUS_INTERFACE BUS = SPLB, BUS_TYPE = SLAVE, BUS_STD = PLBV46

## Generics for VHDL or Parameters for Verilog
PARAMETER C_SPLB_AWIDTH = 32, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_DWIDTH = 128, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_NUM_MASTERS = 8, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_MID_WIDTH = 3, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_NATIVE_DWIDTH = 32, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_P2P = 0, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_SUPPORT_BURSTS = 0, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_SMALLEST_MASTER = 32, DT = INTEGER, BUS = SPLB
PARAMETER C_SPLB_CLK_PERIOD_PS = 10000, DT = INTEGER, BUS = SPLB
PARAMETER C_FAMILY = virtex5:spartan3adsp, DT = STRING
PARAMETER C_MEM0_BASEADDR = 0xffffffff, DT = std_logic_vector, BUS = SPLB, ADDRESS = BASE, PAIR = C_MEM0_HIGHADDR, ADDR_TYPE = MEMORY, MIN_SIZE = 0x1000
PARAMETER C_MEM0_HIGHADDR = 0x00000000, DT = std_logic_vector, BUS = SPLB, ADDRESS = HIGH, PAIR = C_MEM0_BASEADDR, ADDR_TYPE = MEMORY

```

14. Since the custom IP MPD file has been modified, the user IP folder (**/pcores**) must be scanned to reflect the changes. Scan the **/pcores** folder by selecting **Project > Rescan User Repositories** from the XPS GUI.

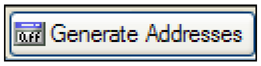


You may need to click on the **System Assembly View** to view the address map.

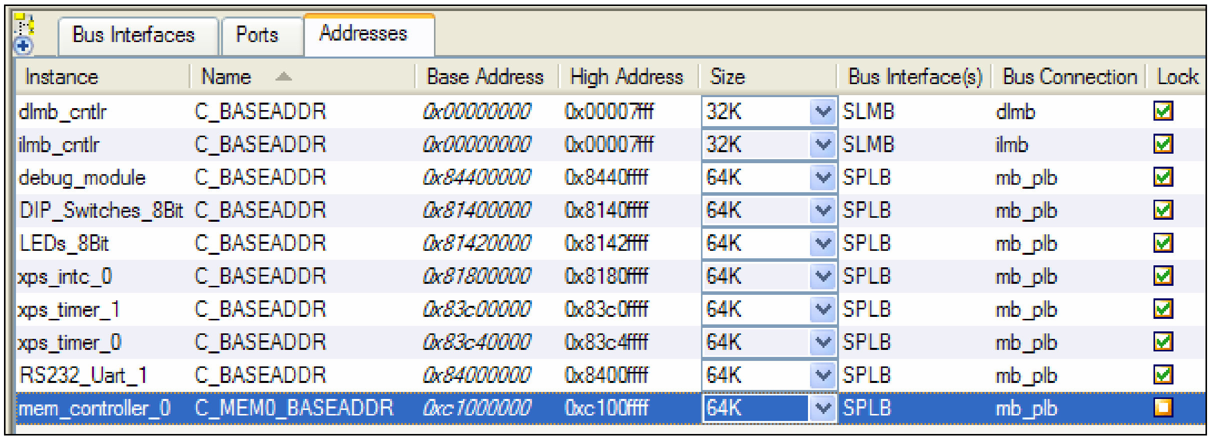


```
Generating Block Diagram : C:\SpeedWay\Fall_07\EDK_Intro_2\L
Generated --- system.svg
Block diagram generated.
```

15. Click on the **Generate Addresses** tab to generate address space for the **mem_controller** IP core.

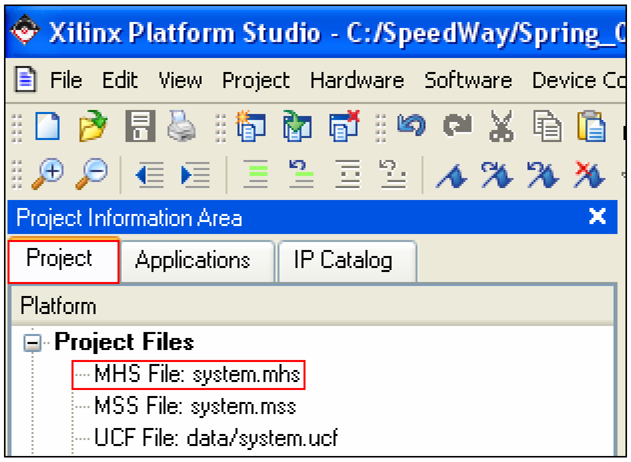


Address space will be assigned to the **mem_controller** IP core as shown in the following figure.

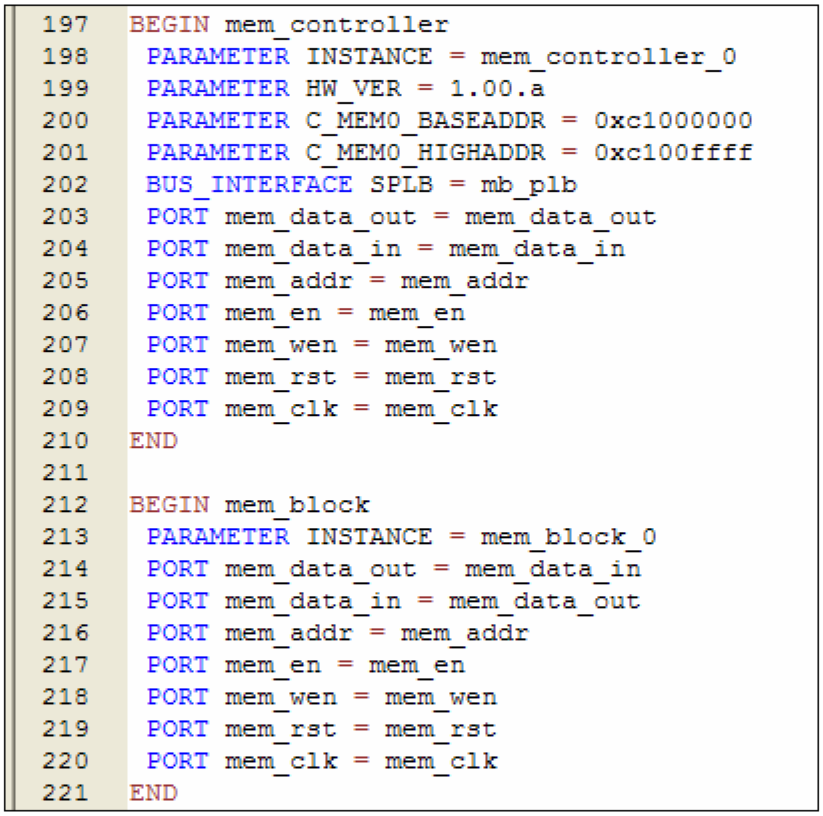


| Instance | Name | Base Address | High Address | Size | Bus Interface(s) | Bus Connection | Lock |
|---|---|---|---|---|---|---|---|
| dlmb_cntlr | C_BASEADDR | 0x00000000 | 0x00007fff | 32K | SLMB | dlmb | ✓ |
| ilmb_cntlr | C_BASEADDR | 0x00000000 | 0x00007fff | 32K | SLMB | ilmb | ✓ |
| debug_module | C_BASEADDR | 0x84400000 | 0x8440ffff | 64K | SPLB | mb_plb | ✓ |
| DIP_Switches_8Bit | C_BASEADDR | 0x81400000 | 0x8140ffff | 64K | SPLB | mb_plb | ✓ |
| LEDs_8Bit | C_BASEADDR | 0x81420000 | 0x8142ffff | 64K | SPLB | mb_plb | ✓ |
| xps_intc_0 | C_BASEADDR | 0x81800000 | 0x8180ffff | 64K | SPLB | mb_plb | ✓ |
| xps_timer_1 | C_BASEADDR | 0x83c00000 | 0x83c0ffff | 64K | SPLB | mb_plb | ✓ |
| xps_timer_0 | C_BASEADDR | 0x83c40000 | 0x83c4ffff | 64K | SPLB | mb_plb | ✓ |
| RS232_Uart_1 | C_BASEADDR | 0x84000000 | 0x8400ffff | 64K | SPLB | mb_plb | ✓ |
| mem_controller_0 | C_MEM0_BASEADDR | 0xc1000000 | 0xc100ffff | 64K | SPLB | mb_plb | ☐ |

16. Click on the **Project** tab and double-click on the **system.mhs** file to open it.

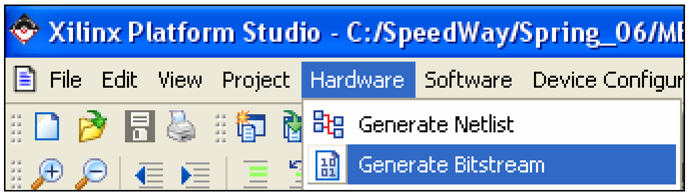

Xilinx Platform Studio - C:/SpeedWay/Spring_C
File Edit View Project Hardware Software Device Co
Project Information Area
Project
Applications
IP Catalog
Platform
Project Files
MHS File: system.mhs
MSS File: system.mss
UCF File: data/system.ucf

17. Notice the **mem_controller** and the **mem_block** IP cores are instantiated in the MHS file as shown in the following figure (the **mem_controller** and the **mem_block** IP core instantiations will be at the end of the MHS file).



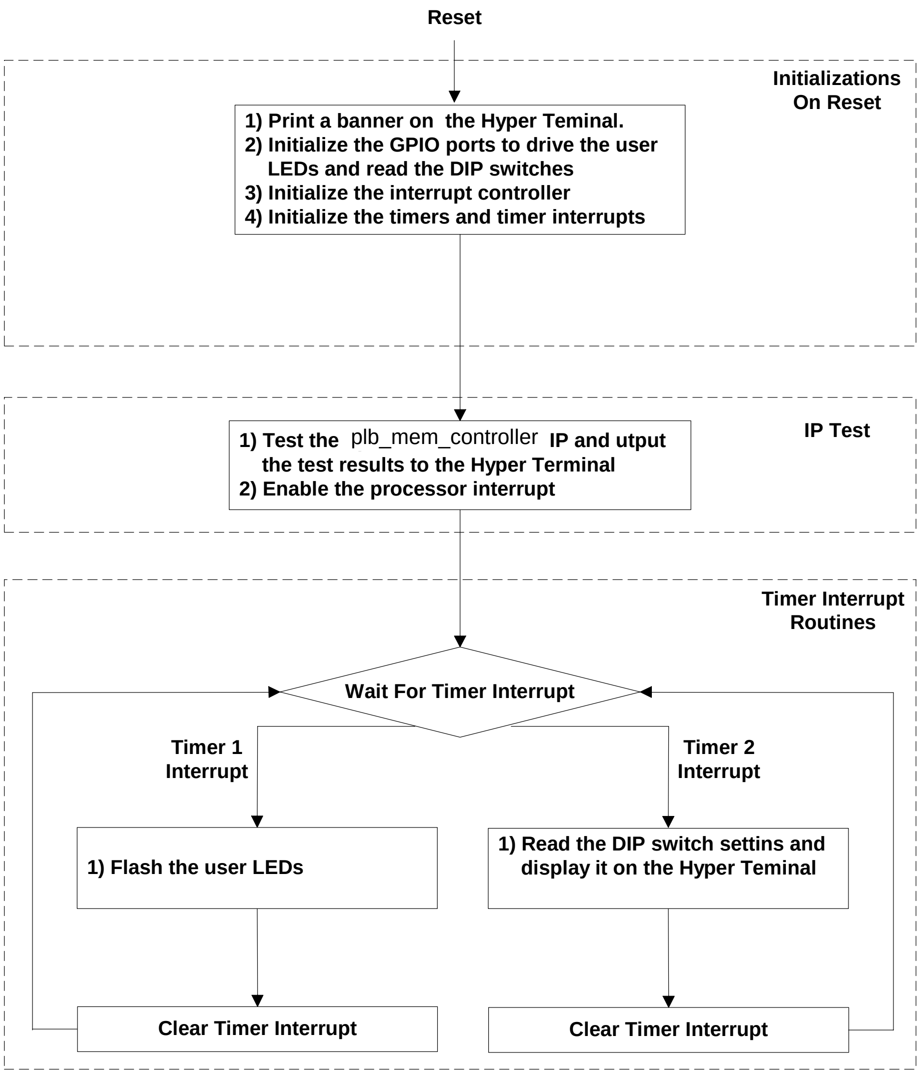
```
BEGIN mem_controller
 PARAMETER INSTANCE = mem_controller_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_MEM0_BASEADDR = 0xc1000000
 PARAMETER C_MEM0_HIGHADDR = 0xc100ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT mem_data_out = mem_data_out
 PORT mem_data_in = mem_data_in
 PORT mem_addr = mem_addr
 PORT mem_en = mem_en
 PORT mem_wen = mem_wen
 PORT mem_rst = mem_rst
 PORT mem_clk = mem_clk
END

BEGIN mem_block
 PARAMETER INSTANCE = mem_block_0
 PORT mem_data_out = mem_data_in
 PORT mem_data_in = mem_data_out
 PORT mem_addr = mem_addr
 PORT mem_en = mem_en
 PORT mem_wen = mem_wen
 PORT mem_rst = mem_rst
 PORT mem_clk = mem_clk
END
```

18. Now that the **mem_controller** and the **mem_block** IP cores have been added to the design, the new hardware platform needs to be re-implemented. Generate a bitstream for the design by selecting **Hardware** > **Generate Bitstream** from the XPS GUI as shown in the following figure.


Xilinx Platform Studio - C:/SpeedWay/Spring_06/MI
File Edit View Project Hardware Software Device Configur
Generate Netlist
Generate Bitstream

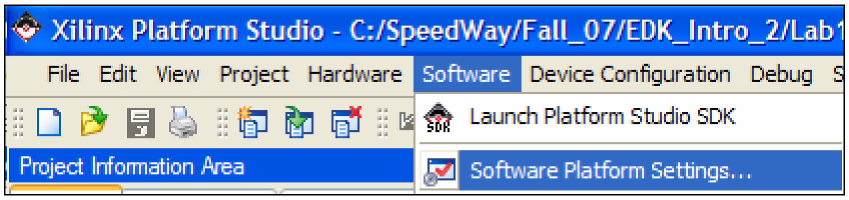
## Application Software

A simple software application (**user_application_2.c**) has been written for this lab. This software will be used to test the custom memory controller IP core in the design. The following figure shows a flow chart for this program.

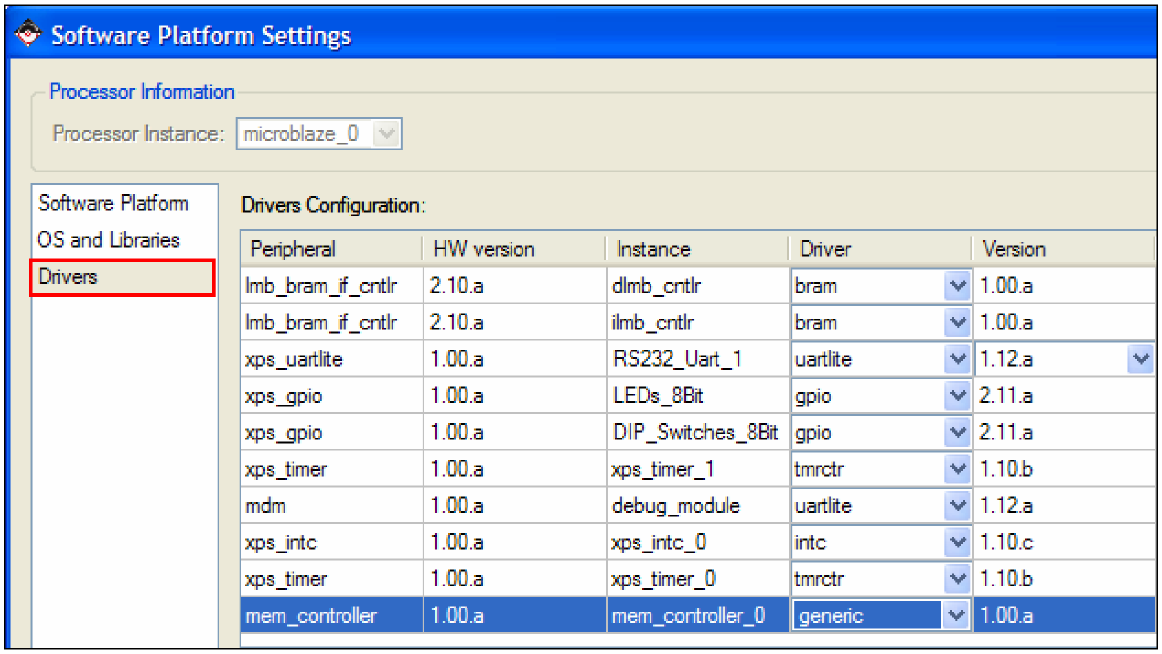

Reset
Initializations On Reset
1) Print a banner on the Hyper Teminal.
2) Initialize the GPIO ports to drive the user LEDs and read the DIP switches
3) Initialize the interrupt controller
4) Initialize the timers and timer interrupts
IP Test
1) Test the plb_mem_controller IP and utput the test results to the Hyper Terminal
2) Enable the processor interrupt
Timer Interrupt Routines
Wait For Timer Interrupt
Timer 1 Interrupt
Timer 2 Interrupt
1) Flash the user LEDs
1) Read the DIP switch settins and display it on the Hyper Teminal
Clear Timer Interrupt
Clear Timer Interrupt

## Setting up Drivers

1. Invoke the **Software Platform Settings** dialog box from the XPS GUI as shown in the following figure.


Xilinx Platform Studio - C:/SpeedWay/Fall_07/EDK_Intro_2/Lab1
File Edit View Project Hardware Software Device Configuration Debug
Launch Platform Studio SDK
Software Platform Settings...
Project Information Area

2. Click on the **Drivers** option (as shown in the following figure) and verify the **generic** version **1.00.a** driver is assigned to the **mem_controller** IP core. Click **OK** to continue.



Processor Instance: microblaze_0

Drivers Configuration:

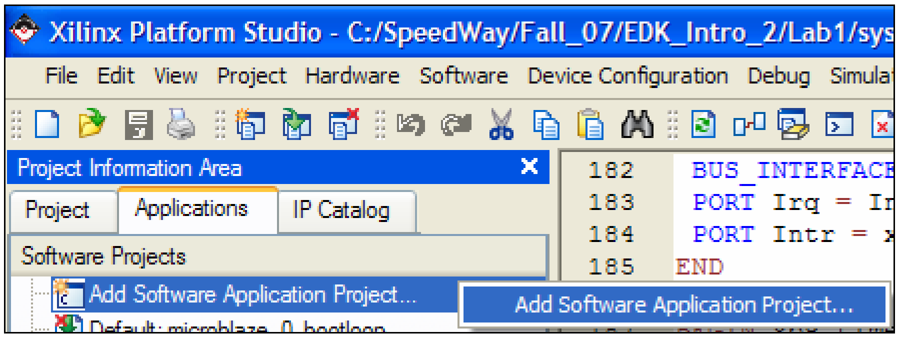
| Peripheral | HW version | Instance | Driver | Version |
|---|---|---|---|---|
| lmb_bram_if_cntlr | 2.10.a | dlmb_cntlr | bram | 1.00.a |
| lmb_bram_if_cntlr | 2.10.a | ilmb_cntlr | bram | 1.00.a |
| xps_uartlite | 1.00.a | RS232_Uart_1 | uartlite | 1.12.a |
| xps_gpio | 1.00.a | LEDs_8Bit | gpio | 2.11.a |
| xps_gpio | 1.00.a | DIP_Switches_8Bit | gpio | 2.11.a |
| xps_timer | 1.00.a | xps_timer_1 | tmrctr | 1.10.b |
| mdm | 1.00.a | debug_module | uartlite | 1.12.a |
| xps_intc | 1.00.a | xps_intc_0 | intc | 1.10.c |
| xps_timer | 1.00.a | xps_timer_0 | tmrctr | 1.10.b |
| mem_controller | 1.00.a | mem_controller_0 | generic | 1.00.a |

## Adding a New Software Project

In the following sections, a new software project will be added to the design and source files will be included to test the **mem_controller** IP core.

1. Copy the **user_application_2** folder from the **C:\SpeedWay\Fall_07\EDK_Intro_2\Misc** directory to the **C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1** folder. The **user_application_2** folder contains the application software for the Lab 1.

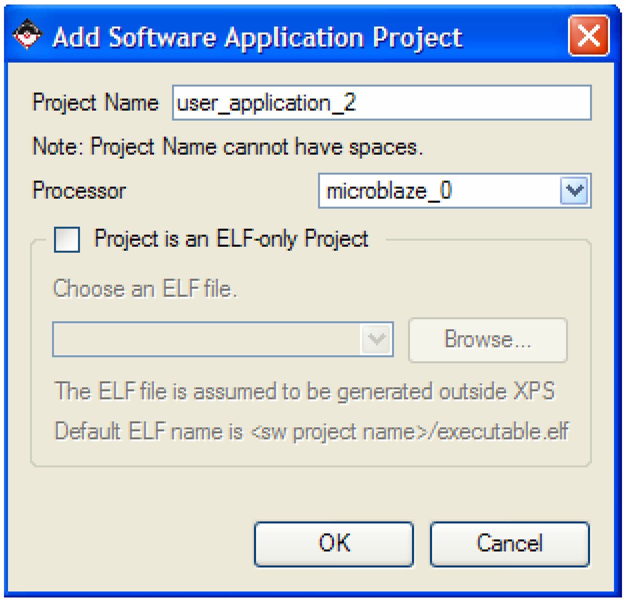
2. Click on the **Applications** tab in the XPS GUI, right-click on the **Add Software Application Project** and then select **Add SW Application Project** as shown in the following figure. The **Add SW Application Project** dialog box will appear.



Xilinx Platform Studio - C:/SpeedWay/Fall_07/EDK_Intro_2/Lab1/sys
File Edit View Project Hardware Software Device Configuration Debug Simula
Project Information Area
Project
Applications
IP Catalog
Software Projects
Add Software Application Project...
Default: microblaze_0_bootloop
Add Software Application Project...

```
BUS_INTERFACE
PORT Irq = Ir
PORT Intr = x
END
```

3. Enter **user_application_2** for the Project Name and click **OK** to continue.



Project Name: user_application_2

Note: Project Name cannot have spaces.

Processor: microblaze_0

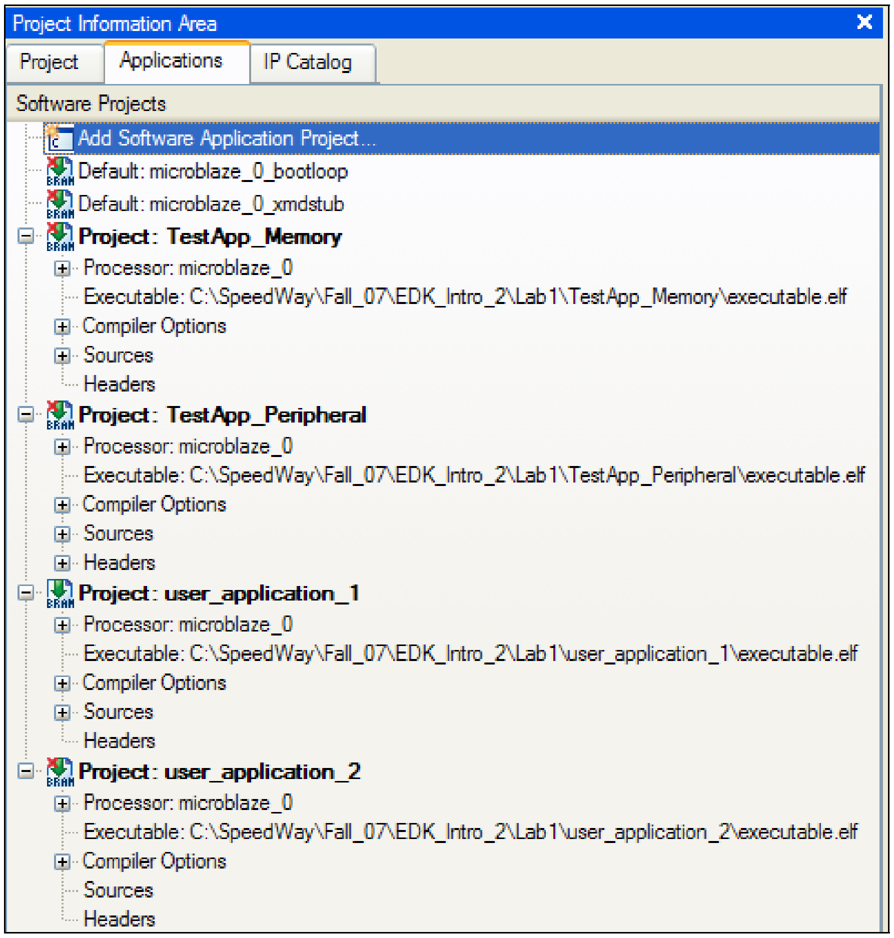
☐ Project is an ELF-only Project

Choose an ELF file.

The ELF file is assumed to be generated outside XPS

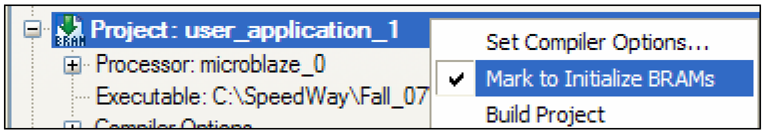
Default ELF name is <sw project name>/executable.elf

4. After performing the above step, the **user_application_2** software project will be added to the project as shown in the following figure.

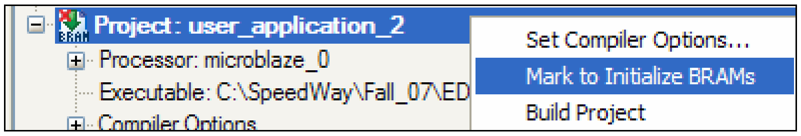


Default: microblaze_0_bootloop
Default: microblaze_0_xmdstub
Project: TestApp_Memory
Processor: microblaze_0
Executable: C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\TestApp_Memory\executable.elf
Compiler Options
Sources
Headers
Project: TestApp_Peripheral
Processor: microblaze_0
Executable: C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\TestApp_Peripheral\executable.elf
Compiler Options
Sources
Headers
Project: user_application_1
Processor: microblaze_0
Executable: C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\user_application_1\executable.elf
Compiler Options
Sources
Headers
Project: user_application_2
Processor: microblaze_0
Executable: C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\user_application_2\executable.elf
Compiler Options
Sources
Headers

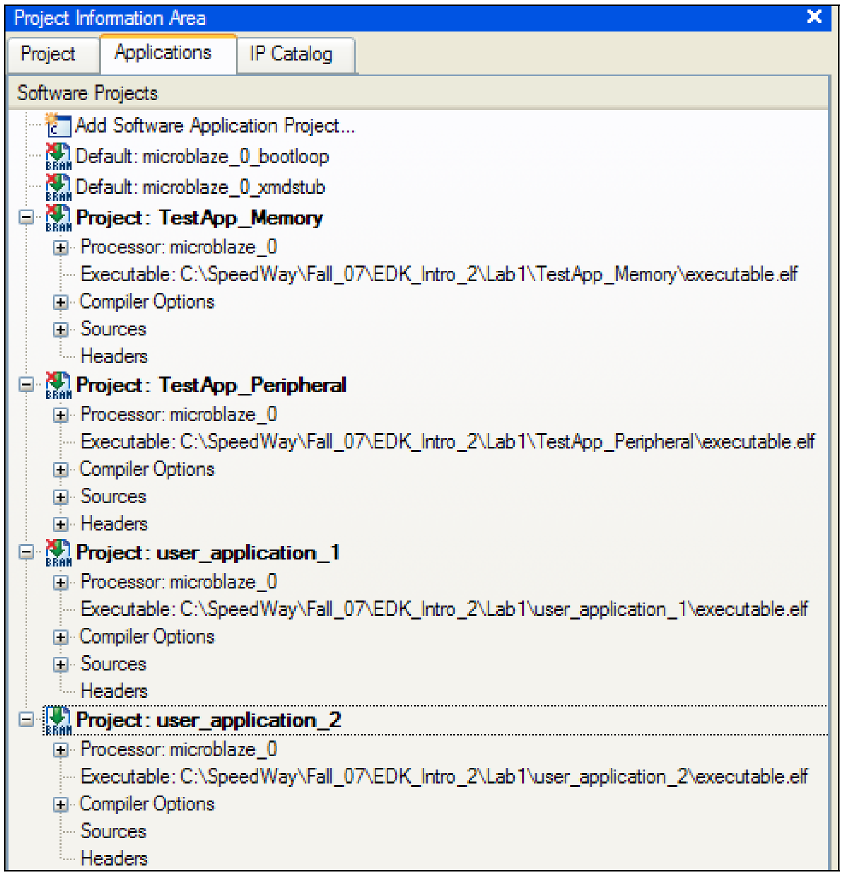
5. Right-click on the **user_application_1** software project and **un-check** the **Mark to Initialize BRAMs**. This will prevent the **user_application_1** software to be loaded into the FPGA Block RAMs when the bit file is loaded into the FPGA.


Project: user_application_1
Processor: microblaze_0
Executable: C:\SpeedWay\Fall_07
Set Compiler Options...
✓ Mark to Initialize BRAMs
Build Project

6. Right-click on the **user_application_2** software project and **check** the **Mark to Initialize BRAMs** as shown in the following figure. This will allow the **user_application_2** software to be loaded into the FPGA Block RAM when the bit file is downloaded to the FPGA.

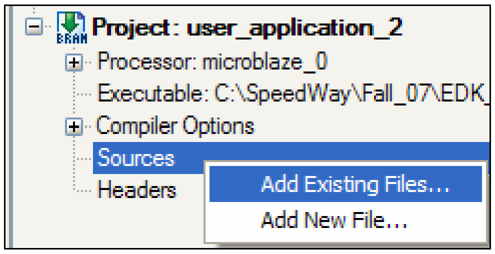


7. After performing the above step, the software projects should look as shown in the following figure. The **user_application_1** is marked not to be loaded into the FPGA Block RAM and the **user_application_2** software project is marked to be loaded into the FPGA Block RAM.

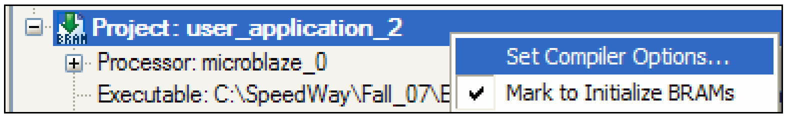


```
Default: microblaze_0_bootloop
Default: microblaze_0_xmdstub
Project: TestApp_Memory
    Processor: microblaze_0
    Executable: C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\TestApp_Memory\executable.elf
    Compiler Options
    Sources
    Headers
Project: TestApp_Peripheral
    Processor: microblaze_0
    Executable: C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\TestApp_Peripheral\executable.elf
    Compiler Options
    Sources
    Headers
Project: user_application_1
    Processor: microblaze_0
    Executable: C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\user_application_1\executable.elf
    Compiler Options
    Sources
    Headers
Project: user_application_2
    Processor: microblaze_0
    Executable: C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\user_application_2\executable.elf
    Compiler Options
    Sources
    Headers
```

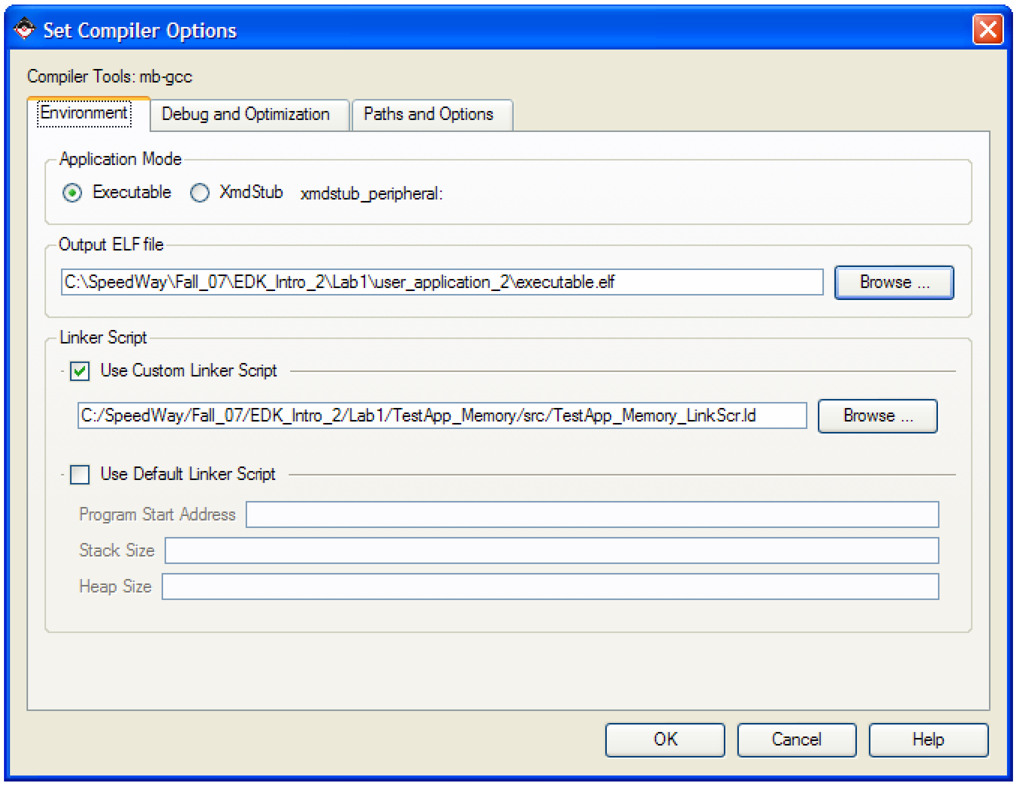
8. After adding the **user_application_2** software project to the design, source files must be added to this software project.
   a) Right-click on the **Sources** under the **user_application_2** software project and then select **Add Existing Files** to add source files to the **user_application_2** software project.
   b) Browse to the **C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\user_application_2** folder, select **user_application_2.c** and then click **Open**.



9. Prior to compiling the **user_application_2** source files, compiler options must be set for this software project and a linker script must be assigned to it as well. Right-click on **user_application_2** project and then select **Set Compiler Options** to invoke the **Set Compiler Options** dialog box.


Project: user_application_2
Processor: microblaze_0
Executable: C:\SpeedWay\Fall_07\E
Set Compiler Options...
✓ Mark to Initialize BRAMs

10. The **Set Compiler Options** dialog box will appear with the **Environment** tab as default.
    a) Make sure **Application Mode** is set to **Executable** (default setting).
    b) Check the **Use Custom Linker Script** box.
    c) Under the **Linker Script** section, browse to the **C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\TestApp_Memory\src** older, select **TestApp_Memory_LinkScr.ld** file and click **Open**. The **Memory Test** linker script can be used for the **user_application_2** software project since both programs are mapped to be stored in the FPAG Block RAM.
    d) Click **OK** to continue.



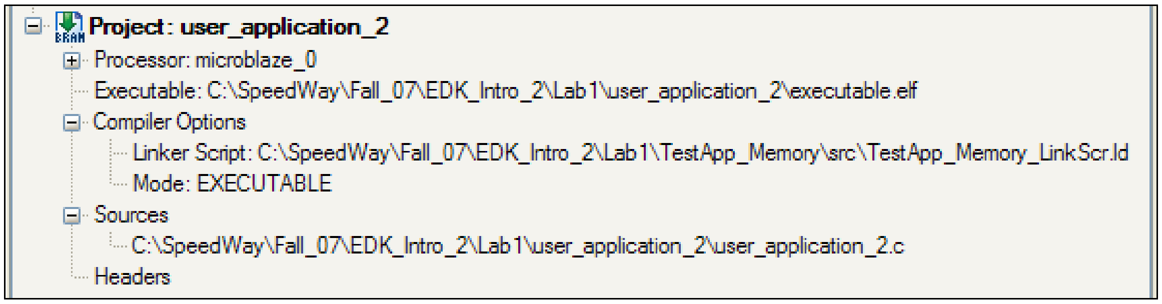
Compiler Tools: mb-gcc

Application Mode
[✓ Executable] XmdStub xmdstub_peripheral:

Output ELF file
C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\user_application_2\executable.elf

Linker Script
[✓] Use Custom Linker Script
C:/SpeedWay/Fall_07/EDK_Intro_2/Lab1/TestApp_Memory/src/TestApp_Memory_LinkScr.ld
[ ] Use Default Linker Script
Program Start Address
Stack Size
Heap Size

11. After adding the **user_application_2.c** and the **TestApp_Memory_LinkScr.ld** linker script to the **user_application_2** software project, the project should look as shown in the following figure.



```
Project: user_application_2
  Processor: microblaze_0
  Executable: C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\user_application_2\executable.elf
  Compiler Options
    Linker Script: C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\TestApp_Memory\src\TestApp_Memory_LinkScr.ld
    Mode: EXECUTABLE
  Sources
    C:\SpeedWay\Fall_07\EDK_Intro_2\Lab1\user_application_2\user_application_2.c
  Headers
```

## Setting Up the Development Board

Please follow the steps shown below to review the jumper settings on the board and run the program.

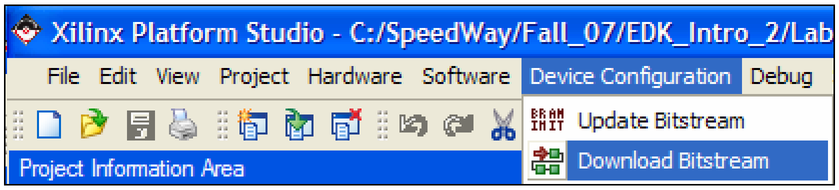
1. Verify the Power switch, **SW1**, is in the **OFF** position.
2. Install jumpers on JP9 pins 3-4 and 5-6 (Mode: M1 and M2)
3. Install jumpers on JP6 pins 1-2 and 3-4
4. Install a jumper on JP11 pins 1-2 (SUSPEND OFF)
5. Install jumpers on JP10 pins 1-2 and 3-4 (3.3V voltage regulator)
6. Install jumpers on JP5 pins 1-2 and 3-4 (1.2V voltage regulator)
7. Install jumpers on JP4 pins 1-2 and 3-4 (2.5V voltage regulator)
8. Install a jumper on JP1 pins 1-2
9. Install a jumper on JP2 pins 1-2 (EXP JX1 VCCO voltage selection)
10. Install a jumper on JP3 pins 1-2 (EXP JX2 VCCO voltage selection)
11. Connect the power supply to the J5 connector on the Spartan-3A DSP Starter board and also plug it into the AC outlet.
12. Connect the USB JTAG cable to J2 and the USB port of the PC.
13. Connect a straight through RS232 cable to the board DB-9 connector (P2) and the serial port of the PC.
14. Slide the power switch to the **ON** position

## Downloading the bit file and running the program

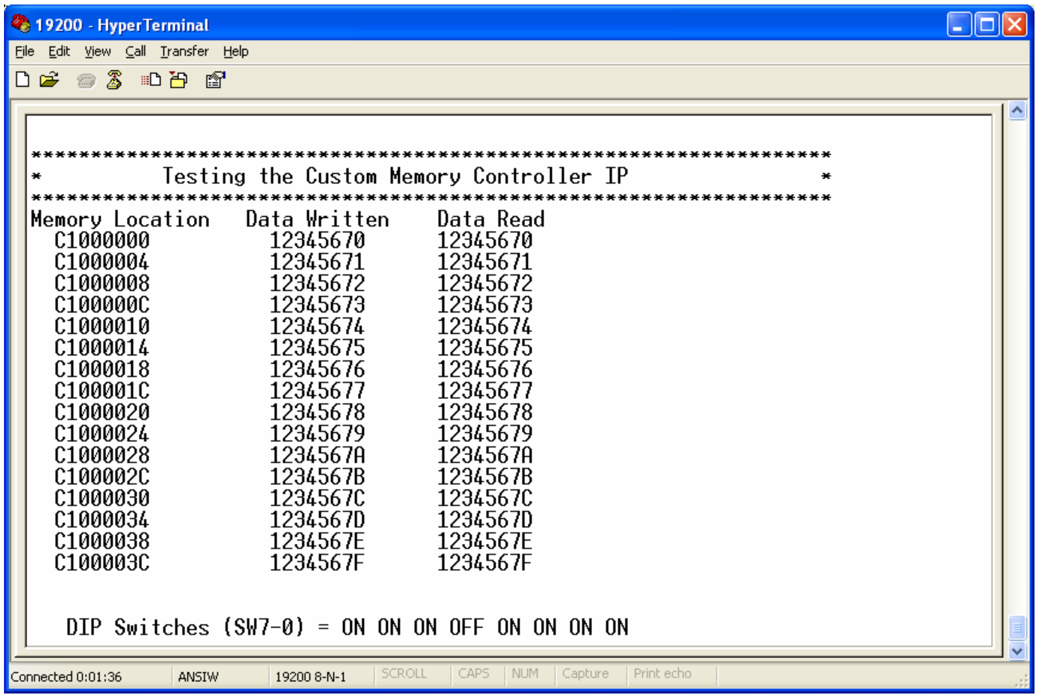
1. Double-click the desktop icon below to start a HyperTerminal session. Alternatively go to the **C:\SpeedWay\Fall_07\EDK_Intro_2\Misc** folder and double-click on the **19200.ht** file to start a HyperTerminal session

The HyperTerminal is set to 19200 baud rate, 8 bits, 1 stop bit, no parity and no hardware flow control to match the settings for the RS232 port during the BSB wizard design creation

2. Compile the **user_application_2** software and download the bit file to the board from the XPS GUI by selecting **Device Configuration > Download Bitstream** as shown in the following figure (invoking the **Download Bitstream** will automatically compile the **user_application_2** software and also run the **Update Bitstream**).



3. You should see the following on the HyperTerminal (only 16 memory locations are tested, however, the **user_application_2.c** file can easily be modified to test all 512 **mem_block** memory locations). Make sure the test passes by examining the data written and data read values.



```
**********************************************************************
*           Testing the Custom Memory Controller IP                 *
**********************************************************************
Memory Location   Data Written    Data Read
  C1000000          12345670       12345670
  C1000004          12345671       12345671
  C1000008          12345672       12345672
  C100000C          12345673       12345673
  C1000010          12345674       12345674
  C1000014          12345675       12345675
  C1000018          12345676       12345676
  C100001C          12345677       12345677
  C1000020          12345678       12345678
  C1000024          12345679       12345679
  C1000028          1234567A       1234567A
  C100002C          1234567B       1234567B
  C1000030          1234567C       1234567C
  C1000034          1234567D       1234567D
  C1000038          1234567E       1234567E
  C100003C          1234567F       1234567F


   DIP Switches (SW7-0) = ON ON ON OFF ON ON ON ON
```

4. Close the project by selecting **File > Close Project** from the XPS GUI.

## Lab 1 Conclusions

Here is a summary of what you have accomplished during the Lab 1 exercise:

- Created a custom IP using the IP creation wizard
- Added the custom IP to the design
- Added a new software project to the design
- Built the design and ran the new software on the Spartan-3A DSP Starter board

## Revision History

| Date | Version | Revision |
|---|---|---|
| 11/5/07 | 1.0 | Initial Release |
| 2/29/08 | 1.1 | Updated for EDK 9.2.02. |

## Answers to questions:

1. The custom IP is stored in the **Lab1\pcores** folder.
2. The custom IP VHDL templates are located in the **Lab1\pcores\mem_controller_v1_00_a\hdl\vhdl** folder.
3. The MPD file for the custom IP (**mem_controller_v2_1_0.mpd**) is located in the **Lab1\pcores\mem_controller_v1_00_a\data** folder. The version of the MPD file is **v2_1_0**.