Creating/Using Custom IP Cores

By Jason Agron

Basic Outline:

This document describes how to create and add a custom IP core to the On-Chip Peripheral Bus (OPB) within EDK/XPS. The custom IP core that we will generate is a very simple device consisting of a single control register and 4 general-purpose registers.

The basic steps in creating a custom IP core are:

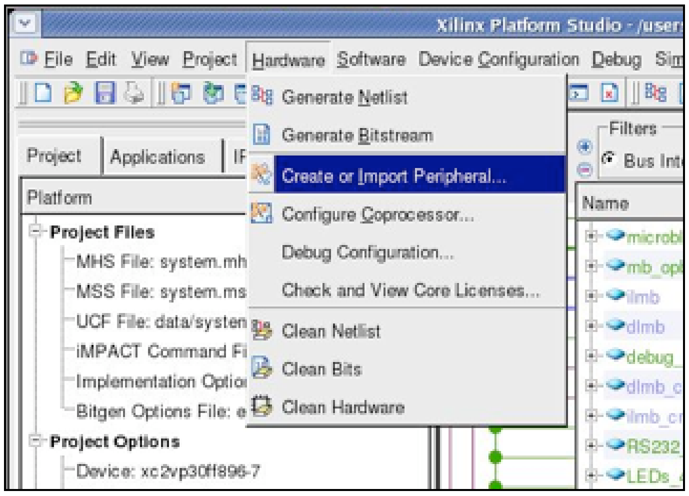
1) Start the "Create/Import Peripheral Wizard".
2) Select a name for the IP core.
3) Select the bus interface of the IP core.
   a. PLB, OPB, or FSL interface.
4) Select the basic services (if any) to be added to the IP core.
5) Select which extra bus interface signals (if any) to be added to the IP core.
6) Select which language to use when generating the IP core.
   a. VHDL or Verilog.

The basic steps in adding a custom IP core to an existing system are.

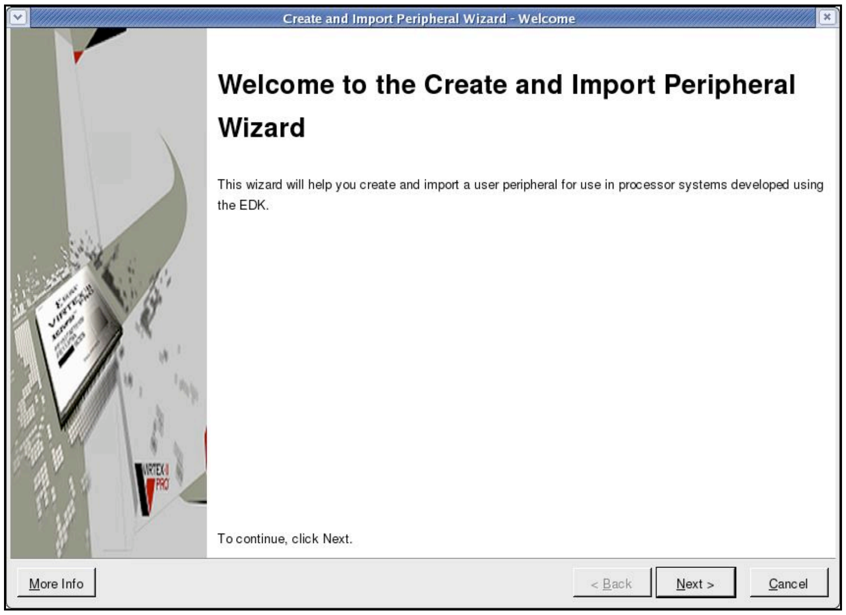
1) Go to the "IP Catalog" Tab and expand the options for "Project Repository".
2) Right-click on the IP core of interest and select "Add IP".
3) Go to the "Bus Interface" view of "System Assembly View" and connect the IP core to the proper bus.
4) Go to the "Address" view of the "System Assembly View" and set the address space for the IP core.

Detailed Steps:

The first step in creating a custom IP core is to start up the "Create/Import Peripheral Wizard". This can be done by first clicking on "Hardware" at the top-level menu of XPS and then selecting "Create or Import Peripheral.



Now, the "Create/Import Peripheral Wizard" will launch, click "Next" to continue.

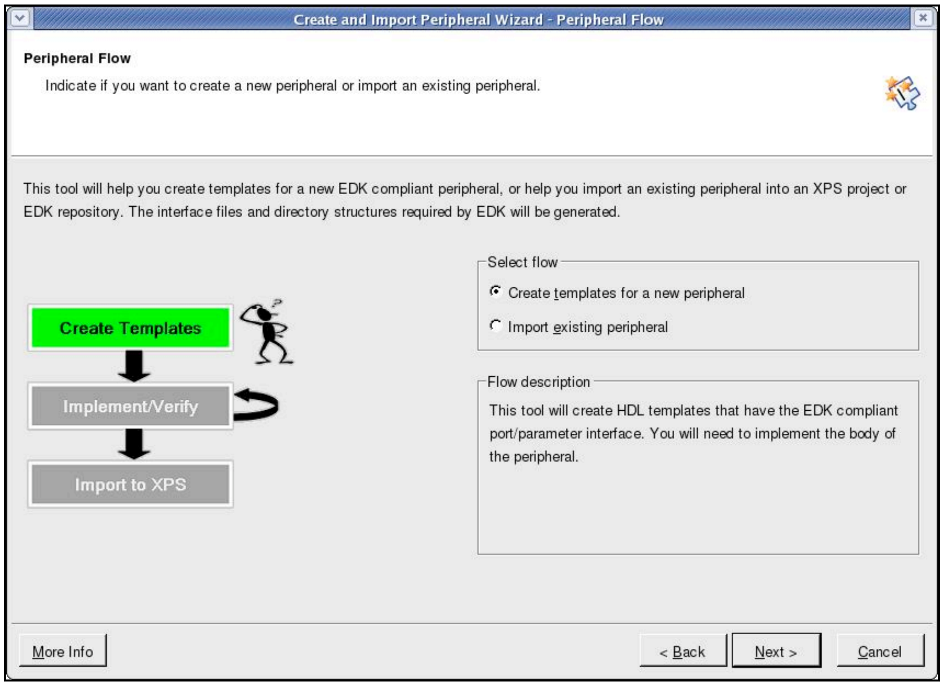


Welcome to the Create and Import Peripheral Wizard

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

To continue, click Next.

Now, the wizard gives you a choice to either create a new peripheral or import an existing one. We will be creating our own custom peripheral from scratch so select "Create Templates For New Peripheral" and click "OK".



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

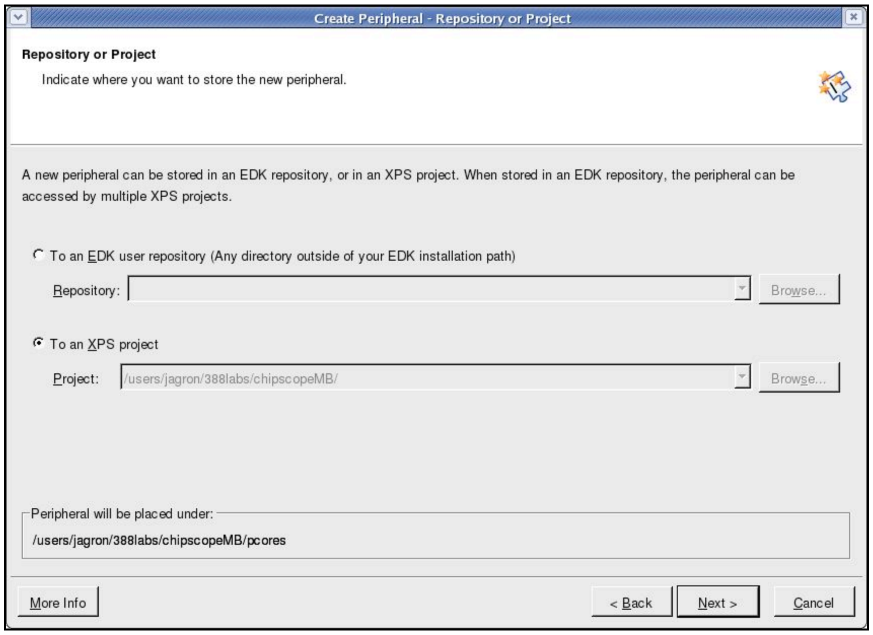
[✓ Create templates for a new peripheral]

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.

The wizard should now show a prompt asking you where you would like to store your custom peripheral. XPS allows you to store peripherals in a project repository (a library of peripherals) or it can store peripherals within the XPS project itself. We will choose the option to store the peripheral within the XPS project as shown in the next figure.


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)

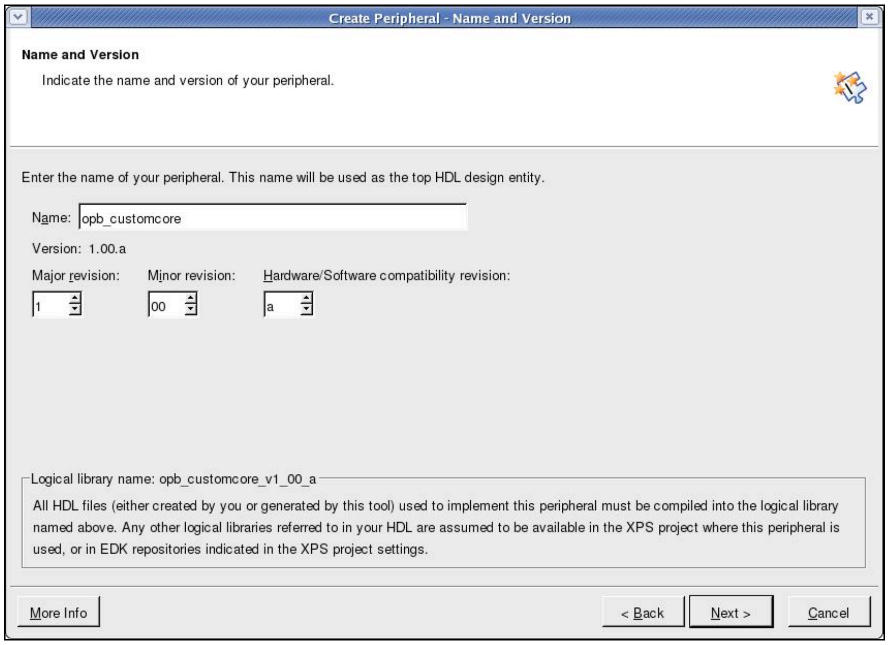
Repository:

(•) To an XPS project

Project: /users/jagron/388labs/chipscopeMB/

Peripheral will be placed under:
/users/jagron/388labs/chipscopeMB/pcores

After selecting "OK", the wizard will prompt you for the name and version number of the new peripheral. The name must not have any spaces and must be in all lower case letters. The version numbers should be left at their default. The purpose of version numbers is to allow multiple versions of an IP core to exist in different forms.


Create Peripheral - Name and Version

Name and Version

Indicate the name and version of your peripheral.

Enter the name of your peripheral. This name will be used as the top HDL design entity.

Name: opb_customcore

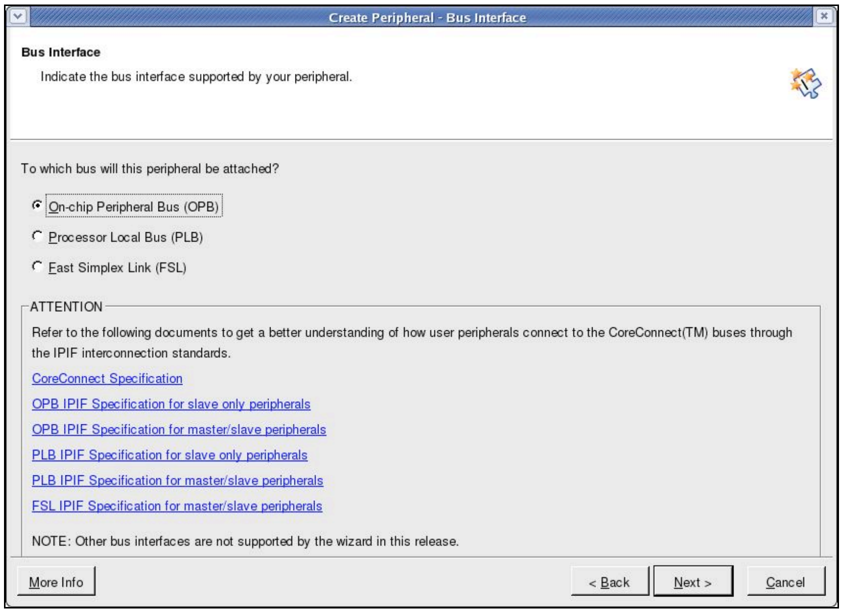
Version: 1.00.a

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

Logical library name: opb_customcore_v1_00_a

All HDL files (either created by you or generated by this tool) used to implement this peripheral must be compiled into the logical library named above. Any other logical libraries referred to in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK repositories indicated in the XPS project settings.

The next step in the process is to select which type of bus interface (often called IPIF – meaning IP Interface) the IP core will use. In the case of MicroBlaze based systems, the main system bus in the On-Chip Peripheral Bus, or OPB, so we will use an OPB IPIF attachment as shown in the following figure.



Indicate the bus interface supported by your peripheral.

To which bus will this peripheral be attached?

[✓ On-chip Peripheral Bus (OPB)]
Processor Local Bus (PLB)
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 through the IPIF interconnection standards.

CoreConnect Specification
OPB IPIF Specification for slave only peripherals
OPB IPIF Specification for master/slave peripherals
PLB IPIF Specification for slave only peripherals
PLB IPIF Specification for master/slave peripherals
FSL IPIF Specification for master/slave peripherals

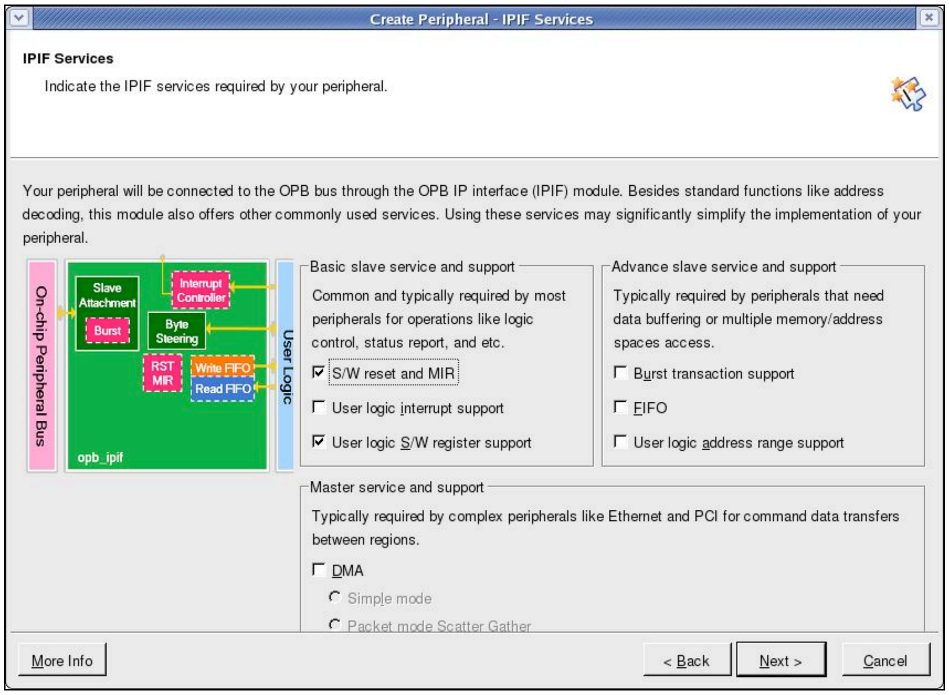
NOTE: Other bus interfaces are not supported by the wizard in this release.

The next wizard screen allows one to choose which of the optional IPIF services to include within the interface for the new peripheral. The IPIF is a parameterizable interface that helps to simplify interactions with the bus. Some of the commonly used options for the OPB IPIF include:

- The ability to perform burst transactions.
- Built-in FIFO logic.
- Automatic address decode support.
- Reset and MIR (hardware version #) registers.
- Built-in interrupt support.
- Built-in support for software-accessible registers.

In our IP core, we will only use the support for the Reset and MIR as well as the User Logic SW register support. Select these two options (and de-select all others) and click "Next" as shown in the following figure. The IP core that we generate will be organized in the following fashion:

- A top-level file named opb_customcore.vhd that connects:
    - The OPB_IPIF.
        - The bus interface wrapper.
    - The user_logic file.
        - Contains the actual "logic" of the IP core.
        - Can be in VHDL or Verilog.


Create Peripheral - IPIF Services

IPIF Services

Indicate the IPIF services required by your peripheral.

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

Basic slave service and support

Common and typically required by most peripherals for operations like logic control, status report, and etc.

- [✓] S/W reset and MIR
- [ ] User logic interrupt support
- [✓] User logic S/W register support

Advance slave service and support

Typically required by peripherals that need data buffering or multiple memory/address spaces access.

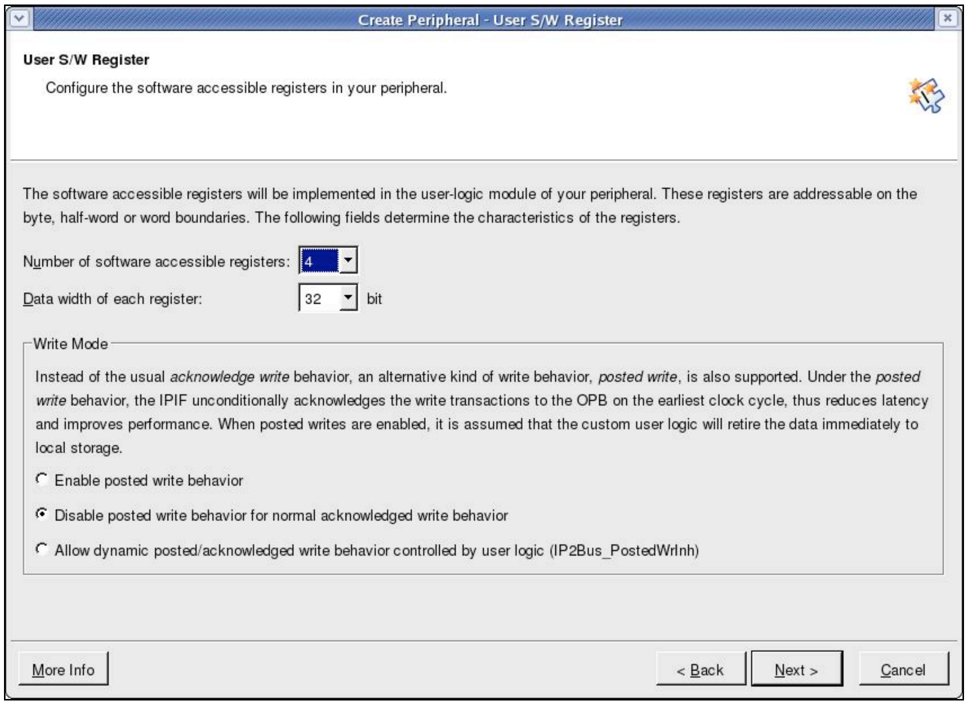
- [ ] Burst transaction support
- [ ] FIFO
- [ ] User logic address range support

Master service and support

Typically required by complex peripherals like Ethernet and PCI for command data transfers between regions.

- [ ] DMA
  - Simple mode
  - Packet mode Scatter Gather

By selecting "User Logic SW Register Support" in the previous wizard window we have enabled the automatic placement of registers within our user logic. Now we must configure the size and number of registers in our IP core. For our use, choose 4, 32-bit wide registers and disable posted write behavior as shown below.


Create Peripheral - User S/W Register

User S/W Register

Configure the software accessible registers in your peripheral.

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

Number of software accessible registers: 4

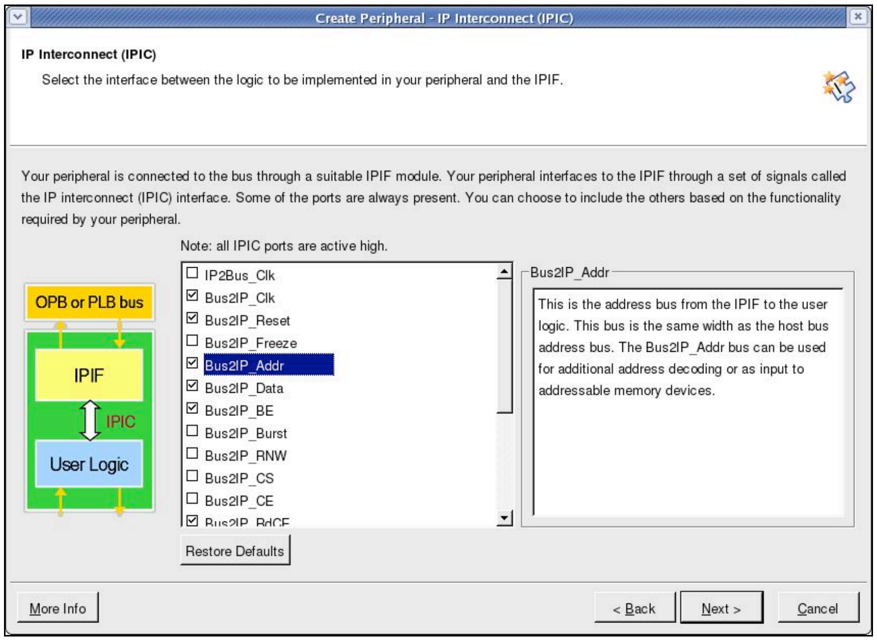
Data width of each register: 32 bit

Write Mode

Instead of the usual *acknowledge write* behavior, an alternative kind of write behavior, *posted write*, is also supported. Under the *posted write* behavior, the IPIF unconditionally acknowledges the write transactions to the OPB on the earliest clock cycle, thus reduces latency and improves performance. When posted writes are enabled, it is assumed that the custom user logic will retire the data immediately to local storage.

- Enable posted write behavior
- [✓ Disable posted write behavior for normal acknowledged write behavior]
- Allow dynamic posted/acknowledged write behavior controlled by user logic (IP2Bus_PostedWrInh)

The next wizard slide allows us to add extra bus signals to our IP core to allow the IP core to use extended features of the bus. In this lab we will not be using any advanced features of the bus, so we will just click "Next" and not modify any of the signals.


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 is connected to the bus through a suitable IPIF module. Your peripheral interfaces to the IPIF through a set of signals called the IP interconnect (IPIC) interface. Some of the ports are always present. You can choose to include the others based on the functionality required by your peripheral.

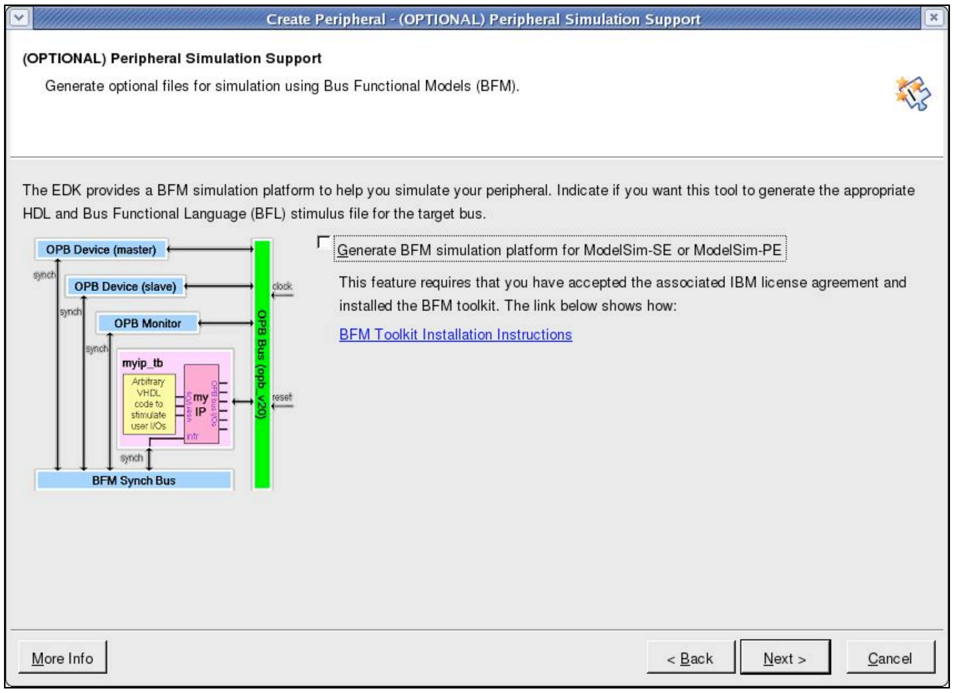
Note: all IPIC ports are active high.

- ☐ IP2Bus_Clk
- ☑ Bus2IP_Clk
- ☑ Bus2IP_Reset
- ☐ Bus2IP_Freeze
- ☑ Bus2IP_Addr
- ☑ Bus2IP_Data
- ☑ Bus2IP_BE
- ☐ Bus2IP_Burst
- ☐ Bus2IP_RNW
- ☐ Bus2IP_CS
- ☐ Bus2IP_CE
- ☑ Bus2IP_RdCE

Bus2IP_Addr

This is the address bus from the IPIF to the user logic. This bus is the same width as the host bus address bus. The Bus2IP_Addr bus can be used for additional address decoding or as input to addressable memory devices.

The next wizard slide allows one to choose to generate a BFM (Bus Functional Model) simulation for the new IP core. The BFM is a high-level simulation model (or testbench) used to test entire bus-based systems. We will not use a BFM simulation model in this lab, so we can de-select the option and click "Next".


Create Peripheral - (OPTIONAL) Peripheral Simulation Support

(OPTIONAL) Peripheral Simulation Support

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

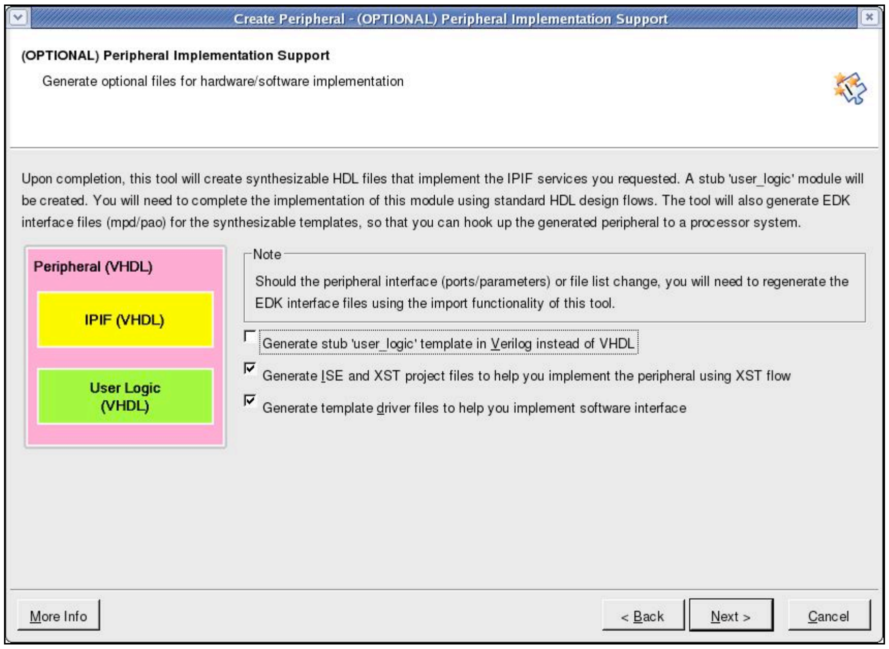
The EDK provides a BFM simulation platform to help you simulate your peripheral. Indicate if you want this tool to generate the appropriate HDL and Bus Functional Language (BFL) stimulus file for the target bus.

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

This feature requires that you have accepted the associated IBM license agreement and installed the BFM toolkit. The link below shows how:

BFM Toolkit Installation Instructions

The next wizard slide allows one to choose implementation options for the IP core such as generating a separate synthesis project and choosing which language to generate the IP core in. The default values for this slide generate the IP core in VHDL and generate a synthesis project as well as template driver files for the IP core. We will use the defaults and click "Next".



Create Peripheral - (OPTIONAL) Peripheral Implementation Support

(OPTIONAL) Peripheral Implementation Support

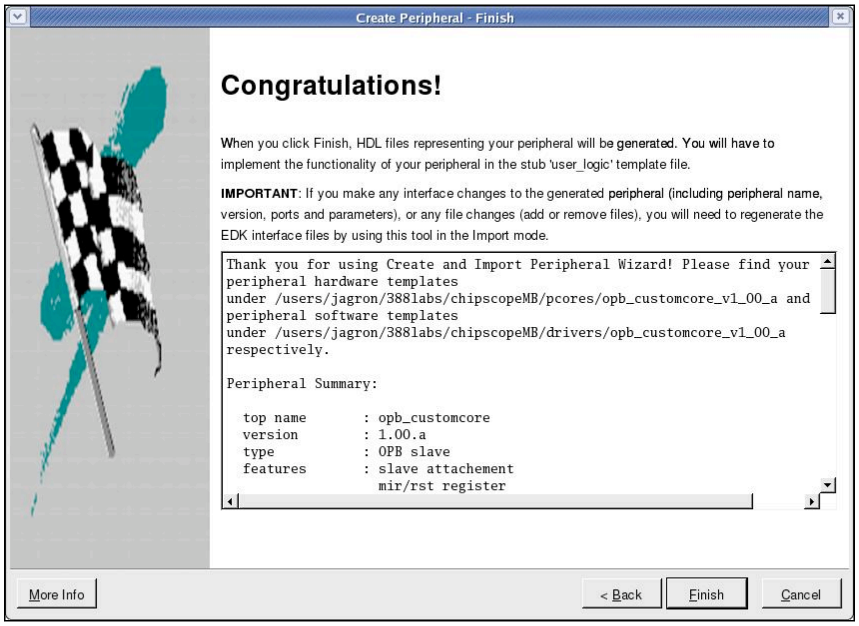
Generate optional files for hardware/software implementation

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
- [x] Generate ISE and XST project files to help you implement the peripheral using XST flow
- [x] Generate template driver files to help you implement software interface

Now, we have finished setting up the IP core template. The next wizard slide allows one to complete the generation of the IP core, which will automatically add the IP core to the "Project Repository" under the "IP Catalog" tab by clicking on "Finish".



Create Peripheral - Finish

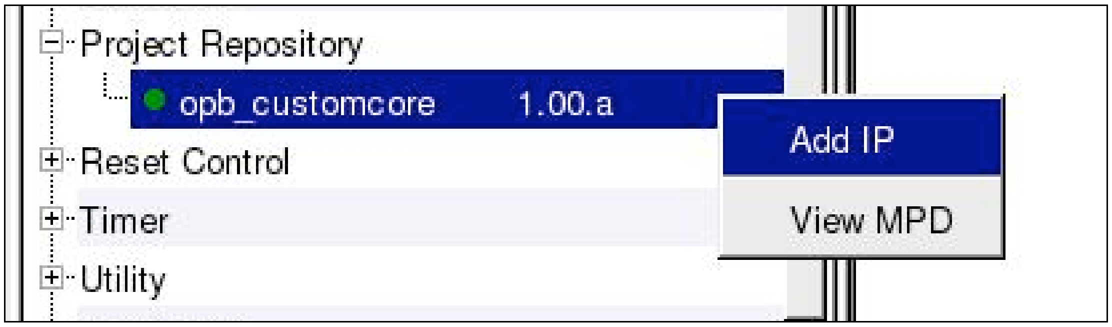
Congratulations!

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.

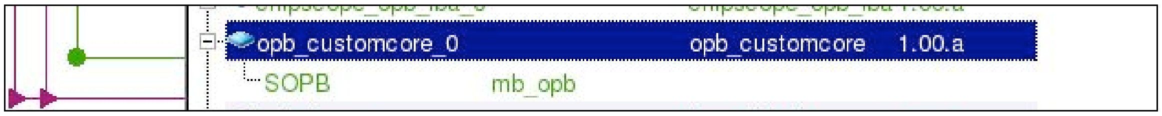
IMPORTANT: If you make any interface changes to the generated peripheral (including peripheral name, version, ports and parameters), or any file changes (add or remove files), you will need to regenerate the EDK interface files by using this tool in the Import mode.

```
Thank you for using Create and Import Peripheral Wizard! Please find your
peripheral hardware templates
under /users/jagron/3881abs/chipscopeMB/pcores/opb_customcore_v1_00_a and
peripheral software templates
under /users/jagron/3881abs/chipscopeMB/drivers/opb_customcore_v1_00_a
respectively.

Peripheral Summary:

  top name        : opb_customcore
  version         : 1.00.a
  type            : OPB slave
  features        : slave attachement
                    mir/rst register
```

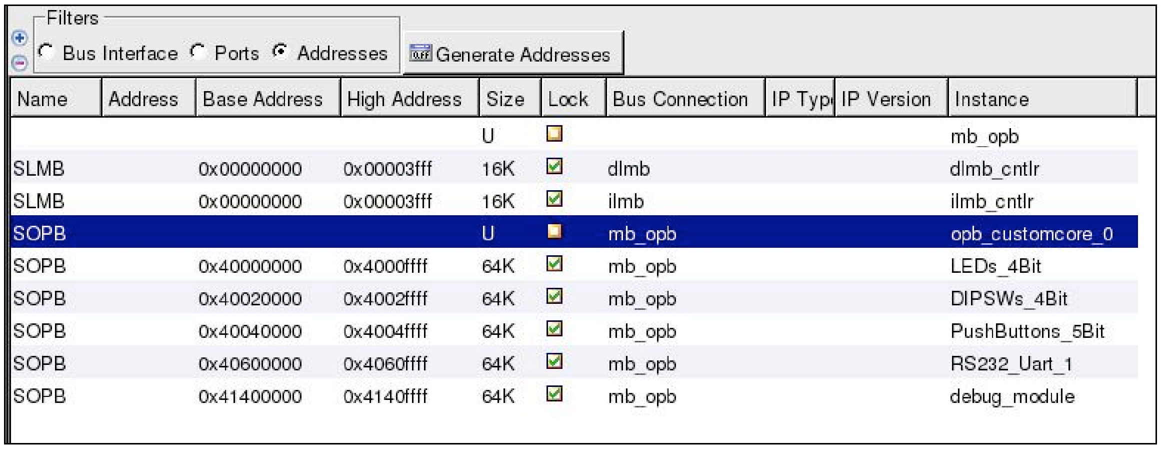
The next step is to add this new IP core to an EDK system. First, go to the "IP Catalog" tab, and then expand the options for "Project Repository". You should now see the newly generated IP core in the list. Right-click on it and select "Add IP" to add the core to the system.



The next step in the process is to connect the IP core to the system bus. Go to "System Assembly View" in the top-right quadrant of XPS and select "Bus Interface" view. The newly added IP core can be connected to the bus by setting its bus interface connection by clicking on the hollow green circle. This will make the circle solid which means that the bus connection has been made.



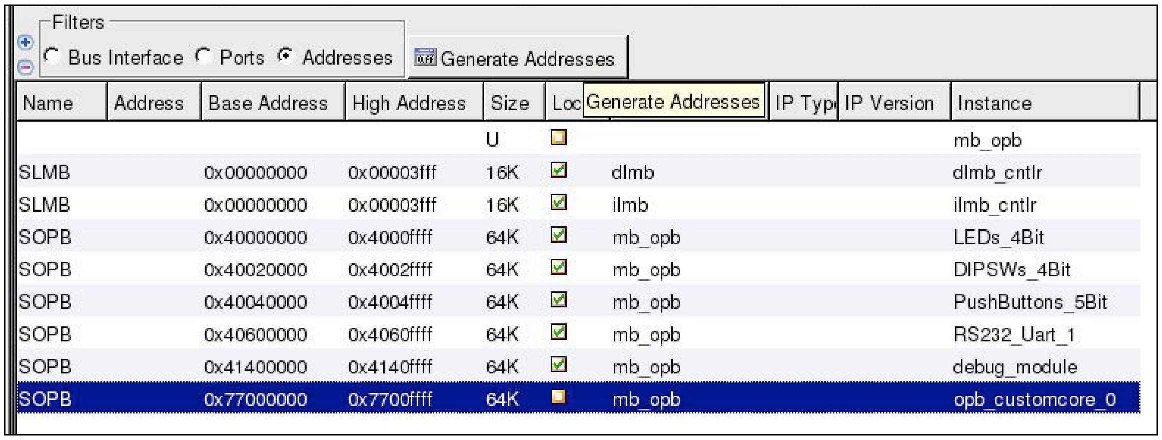
Now the IP core has been connected to the bus, but we still need to set its address space. To do this, first go to the "System Assembly View" and choose the "Addresses" view. Next, we will lock the address ranges of all the other system peripherals except for the newly added IP core.



Filters: Bus Interface, Ports, [✓ Addresses]

Name	Address	Base Address	High Address	Size	Lock	Bus Connection	IP Typ	IP Version	Instance
				U	☐				mb_opb
SLMB		0x00000000	0x00003fff	16K	✓	dlmb			dlmb_cntlr
SLMB		0x00000000	0x00003fff	16K	✓	ilmb			ilmb_cntlr
SOPB				U	☐	mb_opb			opb_customcore_0
SOPB		0x40000000	0x4000ffff	64K	✓	mb_opb			LEDs_4Bit
SOPB		0x40020000	0x4002ffff	64K	✓	mb_opb			DIPSWs_4Bit
SOPB		0x40040000	0x4004ffff	64K	✓	mb_opb			PushButtons_5Bit
SOPB		0x40600000	0x4060ffff	64K	✓	mb_opb			RS232_Uart_1
SOPB		0x41400000	0x4140ffff	64K	✓	mb_opb			debug_module

Now, we can safely click on the "Generate Addresses" button without changing the addresses of any of our other peripherals. By doing this, the XPS tool automatically assigns the newly added IP core an address range by looking at its parameters specified in the core's .mpd file. Make sure to remember and/or write down the address range of the new IP core because this value will be needed when we write software to interact with the IP core.


Filters: ( ) Bus Interface ( ) Ports (•) Addresses | Generate Addresses

Name	Address	Base Address	High Address	Size	Loc	Generate Addresses	IP Typ	IP Version	Instance
				U	☐				mb_opb
SLMB		0x00000000	0x00003fff	16K	✓	dlmb			dlmb_cntlr
SLMB		0x00000000	0x00003fff	16K	✓	ilmb			ilmb_cntlr
SOPB		0x40000000	0x4000ffff	64K	✓	mb_opb			LEDs_4Bit
SOPB		0x40020000	0x4002ffff	64K	✓	mb_opb			DIPSWs_4Bit
SOPB		0x40040000	0x4004ffff	64K	✓	mb_opb			PushButtons_5Bit
SOPB		0x40600000	0x4060ffff	64K	✓	mb_opb			RS232_Uart_1
SOPB		0x41400000	0x4140ffff	64K	✓	mb_opb			debug_module
SOPB		0x77000000	0x7700ffff	64K	☐	mb_opb			opb_customcore_0

Now, the IP core has been fully connected to the system bus. Re-build the hardware platform (by selecting "Update Bitstream"). The next step is to create a new software application that will interact with the IP core. Here are some much-needed facts about the IP core's functionality:

- The IP core has 4 SW-accessible general-purpose (GP) registers.
    - Reg0 = base + 0x0.
    - Reg1 = base + 0x4.
    - Reg2 = base + 0x8.
    - Reg3 = base + 0xC.
    - These registers are readable/writeable.
- The IP core has a write-only reset register.
    - Located at base + 0x100.
    - Writing the code "0x0000000A" to the reset register will automatically set all of the GP registers to 0.

The software "driver" that you will create must do the following:

a. Reset all of the GP registers using the "Reset Command".
b. Infinite Loop:
    i. Write values into each GP register.
    ii. Display values in each GP register via STDOUT.
    iii. Send "Reset Command" to IP core.
    iv. Display values in each GP register via STDOUT.

Additionally the code to reset the IP core and the code to write and read GP registers MUST be encapsulated within functions!!!! This will enhance both the readability and maintainability of your code.