

Creating an OPB IPIF-based IP and Using it in EDK

Author: Mounir Maaref

Abstract	Adding custom logic to an embedded design targeting the Xiliny EPGA can be achieved using
	different methods and techniques. This application note focuses on using the EDK OPB IPIF Interface to achieve such integration.
	This document contains guidelines for choosing the required OPB IPIF Services to use to interface the user logic to the OPB without having to create all the provided IPIF Services.
	Initially, the Create IPIF Wizard is used to generate a user core template, then the user logic HDL is integrated to the template according to the core requirements. Finally, the IPIF Wizard will be used to import the newly created core back into the EDK environment.
	The IPIF Wizard generates a drivers template for the IP. The template is used to access the Custom OPB Core from the System SW Application.
	An example design targeting the Xilinx Reference Platform ML403 is provided to illustrate the design flow, understand the hardware and software implementations, and to test the generated system on the ML403 demonstration board.
Included	Included with this application note is one reference system:
Systems	www.xilinx.com/bvdocs/appnotes/xapp967.zip
Introduction	Adding custom logic to an embedded design can be done using different approaches.
	Custom logic can communicate with the embedded system using the OCM bus in a PowerPC [™] based system. For MicroBlaze [™] systems, the FSL interfaces are an excellent way to make logic directly visible to the processor. Using the OPB/PLB GPIOs with an indexed addressing (if required) can help in integrating user logic to an embedded design.
	The second port, Port B, of the BRAM memories connected to the OCM, PLB, LMB, or OPB bus can make a specific memory region common between the processor and the user logic, thus providing a way of exchanging data between the processor and the FPGA logic.
	Connecting the user logic directly to the OPB or PLB requires understanding the OPB or PLB protocols and designing the required services that simplify the communication with the embedded system.
	To make such a method easy and to shorten the design life cycle of the OPB or PLB custom peripherals, Xilinx provides the necessary OPB IPIF libraries and associated SW tools.
	The focus of this document is on the OPB IPIF services. The document outlines the process for choosing the required OPB IPIF services to interface the user logic to the OPB bus without having to create all the provided IPIF services.
	The first stage of creating such a peripheral is to use the IPIF wizard to generate a User Core Template to which the required modifications are added to the user logic HDL according to the core requirements. The IPIF wizard is used for a second time to import the newly customized core (OPB IPIF services + User HDL) back into the EDK environment, and finally the software

^{© 2007} Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at http://www.xilinx.com/legal.htm. PowerPC is a trademark of IBM Inc. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice. NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

drivers template generated for the custom IP are customized for use with the new IP core functionalities in the software application running on the embedded system.

Figure 1 illustrates the logical steps needed for such a design flow.



Figure 1: Design Flow for an OPB IPIF-based Custom IP

Hardware and Software Requirements

The hardware and software requirements are:

- EDK 8.2i with Service Pack 1 or Higher
- ISE 8.2.03 or Higher
 - HyperTerminal or another terminal emulator
- Xilinx ML403 demo board
- Xilinx Parallel Cable 4 or USB Cable
- Serial Cable

Overview of the OPB IPIF Services

OPB IPIF (On-Chip Peripheral Bus Intellectual Property Interface) provides a standardized connection to the OPB. The IPIF uses a back-end interface standard called the IPIC (IP Interconnect) which helps to connect the user logic to the IPIF services. The IPIF provides options which can be selected by the user, such as:

- Address decoding
- Interrupt management
- Software accessible registers
- IP reset via software-accessible registers

- Module identification register
- Read and write FIFOs between the user logic and the OPB
- Simple DMA capability for the read and transmit sides
- Scatter-Gather DMA (SG DMA) capability for the read and transmit sides

The IPIF services are used in most Xilinx processor IP device implementations and are available for customer use when creating custom OPB peripherals to integrate into an XPS design.

Having a common interface, the OPB IPIF reduces the development effort for custom OPB cores, and promotes higher quality because of less variability.



Block Diagram Figure 2 provides a functional representation of the OPB IPIF.

Figure 2: OPB IPIF Block Diagram

Most of the IPIF Services are optional and may be included or excluded according to the IP requirement.

OPB IPIF Services Software Interface Registers

Most of the IPIF Services are accessible by the software application through a predefined set of registers.

The IPIF registers are grouped by function, and each set of registers belonging to the same group are implemented in the IP if the corresponding function is selected during the build time of the IP core template.

OPB IPIF Services Memory Mapping

Every IPIF Service is mapped to the system memory at a predefined offset, and every register in the function is again mapped to the system memory at a predefined register offset.

Device Interrupt Source Controller Service: The IPIF Service Offset = 0x0.

IP Interrupt Source Controller Service: The IPIF service offset = 0×0 .

Every register functionality, access mode, and offset is defined in the DS414 OPB IPIF data sheet.

Reset Register and MIR Service: The IPIF service offset = 0×0 .

Read FIFO Service: The IPIF service offset = C_RDFIFO_REG_BASEADDR_OFFSET.

Write FIFO Service: The IPIF service offset = C_WRFIFO_REG_BASEADDR_OFFSET.

DMA/Scatter Gather Service: The IPIF Service Offset = C_DMA_REG_BASEADDR_OFFSET + chan_num*64.

chan num = 0 for the transmit (or write) side.

chan_num = 1 for the receive (or read) side.

Every register functionality, access mode, and offset is defined in the DS414 OPB IPIF data sheet.

Identifying the OPB IPIF Services to Use

In the early stages of integrating the user logic to an existing embedded system design, highlight the considerations of how the final OPB custom IP should be made visible to the other peripherals accessing the same OPB. When needed, define the visibility of the IP to the software running on the system via user accessible registers. In addition, indicate if the custom IP generates interrupts and if the IP consumes or produces data at a rate faster or slower than the embedded system speed. It is also required to know if the custom peripheral will need DMA access and burst to transfer and read data form the embedded system memory space.

The objective of this design is to integrate a user logic side that is operating at a different speed than the processor system, and which will therefore require time domain interface over synchronous FIFOs for read and write operations. The IP receives data from the processor system, processes it, and when done processing, will interrupt the processor to provide the results. This embedded design is based on a PowerPC processor with an OPB UART Lite core



to use as a console for user verification. Figure 3 provides a high level description of the embedded design.

Figure 3: Embedded System Diagram

To simplify the example, a simple user logic functionality is required. The custom logic will act as a slave peripheral, a destination of data packets generated by the processor. The peripheral loops back the received data within the user logic side. The data is then returned to the processor over the OPB. For the IP to processor synchronization, it is important to avoid data overruns. Figure 4 shows the data flow that the IP should keep up with in the system.



Figure 4: Data Flow Diagram

For exchanging data with to the processor, the hardware structure applicable is the use of FIFOs.

For the IP to processor synchronization using the interrupt will be best implemented, and because the data is packet type, using DMA for packet transfers from the embedded system memory to the OPB IPIF IP could also be used to offload data packet exchange from the processor list of tasks at run time.

Generating the Required OPB IPIF Core Template

This section describes how to use the Create/Import Peripheral Wizard of XPS to create the required IPIF template.

- 1. Unzip the reference design to your local hard drive, then open the project using XPS.
- 2. In the XPS window shown in Figure 5, select Hardware \rightarrow Create or Import Peripheral....



Figure 5: XPS IPIF Wizard Launch

- 3. In the Wizard Welcome Window click Next.
- 4. In the Create/Import User Peripheral Wizard window shown in Figure 6, select Create Templates, and in Select Flow, select Create template for a new peripheral.

Create and Import Periph	eral Wizard 🛛 🔀
Create/Import User Peripheral Indicate if you want to create a new peripheral or import an existing peripheral.	
This tool will help you create tem existing peripheral into an XPS p structures required by EDK will b Create Templates Implement/Verify Import to XPS	nplates for a new EDK compliant peripheral, or help you import an orgect or EDK repository. The interface files and directory be generated. Select Flow C 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.
More Info	< Back Next > Finish Cancel

Figure 6: Create or Import Peripheral Menu

5. In the Repository or Project window shown in Figure 7, select **To an XPS project**. In the Project field, select \$\\xapp967\. In the Peripheral will be placed under: field, select **C:\xapp967\pcores**, then click **Next**.

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 peripl accessed by multiple XPS projects.	heral can be
O To an EDK user repository (Any directory outside of your EDK installation path)	
Repository:	Browse
To an XPS project	
Project: C:\xapp967\	Browse
Peripheral will be placed under: C:\vapp967\pcores	
More Info KBack Next>	Cancel

Figure 7: IPIF Template Repository Selection

 In the Name and Version window, provide the name of the peripheral. In the Name field, enter opb_ipif_template. Make the selections in the revision fields as shown in Figure 8, then click Next.

ereater empheral mane and remon	
lame and Version Indicate the name and version of your peripheral.	×.
nter the name of your peripheral. This name will be used as the	top HDL design entity.
Name: opb_ipif_template	
Version: 1.00.a	
Major revision: Minor revision: Harowaev sortware (compatibility revision:
Logical library name: opb_ipif_template_v1_00_a	
Logical library name: opb_ipit_emplate_v1_00_a All HDL files (either created by you or generated by this tool) us named above. Any other logical libraries referred to in your HDI used, or in EDK repositories indicated in the XPS project settin	ised to implement this peripheral must be compiled into the logical library 1L are assumed to be available in the XPS project where this peripheral i 195.

Figure 8: Create Template Name and Version

7. In the Bus Interface window shown in Figure 9, choose **On-chip Peripheral Bus (OPB)** as the bus to which the IP is attached, then click **Next**.

lus Interface	
Indicate the bus interface supported by your peripheral.	No.
o which bus will this peripheral be attached?	
On-chip Peripheral Bus (0PB)	
O Processor Local Bus (PLB)	
O Fast Simplex Link (FSL)	
ATTENTION	
Refer to the following documents to get a better understanding of how use through the IPIF interconnection standards.	er peripherals connect to the CoreConnect(TM) buses
Refer to the following documents to get a better understanding of how use through the IPIF interconnection standards. <u>CoreConnect Specification</u>	r peripherals connect to the CoreConnect(TM) buses
Refer to the following documents to get a better understanding of how use through the IPIF interconnection standards. <u>CoreConnect Specification</u> OPB IPIF Specification for slave only peripherals	er peripherals connect to the CoreConnect(TM) buses
Refer to the following documents to get a better understanding of how use through the IPIF interconnection standards. <u>CoreConnect Specification</u> OPB IPIF Specification for slave only peripherals <u>OPB IPIF Specification for master/slave peripherals</u>	er peripherals connect to the CoreConnect(TM) buses
Refer to the following documents to get a better understanding of how use through the IPIF interconnection standards. <u>CoreConnect Specification</u> <u>OPB IPIF Specification for slave only peripherals</u> <u>OPB IPIF Specification for master/slave peripherals</u> <u>IPIS IPIF Specification for slave only peripherals</u>	er peripherals connect to the CoreConnect(TM) buses
Refer to the following documents to get a better understanding of how use through the IPIF interconnection standards. <u>CoreConnect Specification</u> 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	er peripherals connect to the CoreConnect(TM) buses
Refer to the following documents to get a better understanding of how use through the IPIF interconnection standards. <u>CoreConnect Specification</u> OPB IPIF Specification for layer only peripherals OPB IPIF Specification for master/slave peripherals PLB IPIF Specification for master/slave peripherals PLB IPIF Specification for master/slave peripherals SLIPIF Specification for master/slave peripherals	er peripherals connect to the CoreConnect(TM) buses
Refer to the following documents to get a better understanding of how use through the IPIF interconnection standards. <u>CoreConnect Specification</u> OPB IPIF Specification for slave only peripherals OPB IPIF Specification for master/slave peripherals PLB IPIF Specification for master/slave peripherals PLB IPIF Specification for master/slave peripherals ESL IPIF Specification for master/slave peripherals SPL IPIF Specification for master/slave peripherals NOTE: Other bus interfaces are not supported by the wizard in this release	er peripherals connect to the CoreConnect(TM) buses
Refer to the following documents to get a better understanding of how use 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 master/slave peripherals PLB IPIF Specification for master/slave peripherals SLIPIF Specification for master/slave peripherals NOTE: Other bus interfaces are not supported by the wizard in this release	er peripherals connect to the CoreConnect(TM) buses

Figure 9: Bus Interface Selection

8. In the IPIF Services window, select the required services as shown in Figure 10.

IPIF Services Indicate the IPIF services required by your periph	ieral	
Your peripheral will be connected to the OPB bus thin decoding, this module also offers other commonly use peripheral.	augh the OPB IP interface (IPIF) module. Id services. Using these services may sign Basic slave service and support Common and typically required by most peripherals for operations like logic control, status report, and etc. (V) SAV reset and MIR) (V) User logic interrupt support User logic SAV register support Master service and support Typically required by complex peripher, data transfers between regions. (V) DMA (o) Simple mode Packet mode Scatter Gather User logic master support	Besides standard functions like address niticantly simplify the implementation of your Advance slave service and support Typically required by peripherals that need data buffering or multiple memory/address spaces access. Bust transaction support IFFO User logic address range support als like Ethernet and PCI for command
More Info	< Ba	ck Next > Cancel

Figure 10: OPB IPIF Services Selection Menu

9. In the FIFO Service window, configure the size of the read and write FIFOs by making the selections shown in Figure 11, then click **Next**.



Figure 11: FIFO Services Configuration

10. In the Interrupt Services window, configure the IP interrupt services by making the selections as shown in Figure 12, then click **Next**.

Create Peripheral - Interrupt	Service	?
nterrupt Service Configure interrupt handling.		S.
he interrupt handling service in the IPIF, the user-logic and by the other services IPIF Device interrupt Device interrupt Device interrupt Source Control and Registers	available in the IPIF.	pt signals generated
More Info	(Back Next>	Cancel

Figure 12: Interrupt Service Configuration

11. In the IP Interconnect (IPIC) window, customize the IPIC by making the selections as shown in Figure 13, then click **Next**. This is the default setup for this IP.

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. OPB or PLB bus USER Logic USER L
More Info < Back Next > Finish Cancel

Figure 13: IPIC Setup

12. In the (Optional) Peripheral Simulation Support window shown in Figure 14, make no selections, instead click **Next**.

Note: Do not select the BFM Simulation Platform for ModelSim; it is outside the scope of this document



Figure 14: BFM Simulation Support

13. In the (Optional) Peripheral Implementation Support window, make the selections as shown in Figure 15 to control how optional files for hardware and software implementation are generated. Select Generate ISE and XST Project Files to help during the implementation of the Peripheral using the ISE Environment to set up the required synthesis libraries in the ISE Project. Select Generate template driver files to help you implement software interface for help in customizing the IP software interface later in the software application. Click Next.

Create Peripheral - Step 9		
(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.		
Peripheral (VHDL)	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.	
IPIF (VHDL)		
User Logic		
(VHDL)	DL) Generate template driver files to help you implement software interface.	
More Info	< Back Next > Finish Cancel	

Figure 15: ISE Project Generation for the IP and Drivers Template

14. As seen in the Finish Window, the required OPB IPIF Template is generated under the pcores directory of the XPS Project.

Under the Template IP name folder (xapp967\pcores\opb_ipif_template_v1_00_a\), the wizard generates the following directories:

- \dev\ folder where the ISE Project is located for the IP editing.
- \data\ folder which contains the IP HW Platgen API Files (MPD, PAO).
- \hdl\ folder which contains the IP Template HDL files.

Under the Template IP name folder,

(xapp967\drivers\opb_ipif_template_v1_00_a\), the wizard generates the drivers directory for the IP Template:

- \src\ folder where the drivers Source code files (*.h, *.c, makefile) are located.
- \data \ folder where the IP SW libgen API Files (mdd, tcl) are located.

Customizing the User Logic Side of the Template in ISE

The generated OPB IPIF core template includes a user logic side where the user logic is connected to the IPIF through the IPIC set of selected signals. At this stage, the user logic required functionalities are added to the user logic generated HDL file.

The ISE Project generated for customizing the Template is found under the directory

\xapp967\pcores\opb_ipif_template_v1_00_a\dev1\projnav\.

The **opb_ipif_template.ise** project can be opened using ISE.

Customizing User IOs:

This step describes how to add custom user IOs if needed by custom OPB peripheral. Input and output ports may be added as required.

In the user_logic.vhd file under the entity declaration, locate the following lines.

-- ADD USER PORTS BELOW THIS LINE ------

--USER ports added here

-- ADD USER PORTS ABOVE THIS LINE ------

The port FIFOs_Status_Flag has been added as indicated below:

FIFOs_Status_Flag	: out std_logic;
-------------------	------------------

In the signal declaration, the following signals have been added:

signal rdfifo_flag	: std_logic;
signal wrfifo_flag	: std_logic;

At the end of the architecture declaration, the following has been added:

rdfifo_flag <=RFIFO2IP_Full;

wrfifo_flag <=WFIFO2IP_Empty;</pre>

FIFOs_Status_Flag <= rdfifo_flag or wrfifo_flag;

At this stage, a user defined I/O to the user logic has been added. Propagate this addition to the custom OPB IP top level where the user_logic is instantiated.

Save and check the syntax of the user_logic.vhd module.

Open the file opb_ipif_template.vhd.

In the entity port declaration, locate the following:

-- ADD USER PORTS BELOW THIS LINE ------

--USER ports added here

-- ADD USER PORTS ABOVE THIS LINE ------

Add the following port as indicated above:

User_FIFOs_Status_Flag : out std_logic;

Locate the following:

-- Instantiate the User Logic

In the port map section:

-- MAP USER PORTS BELOW THIS LINE ------

--USER ports mapped here

-- MAP USER PORTS ABOVE THIS LINE ------

Map the user port as follows:

FIFOs_Status_Flag => User_FIFOs_Status_Flag,

Save and check the syntax of the opb_ipif_template.vhd file.

Customizing User Logic Interrupts

In the generated template of the user_logic.vhd, example code which generates user logic interrupts is provided by the wizard. The code snippet infers a counter and generates the interrupts whenever the counter rollover (the counter will roll over ~10 sec @ 100 Mhz).

The functionality of the user interrupts can be modified by editing this process. The process example will be commented out.

The desired functionality in this example is that the user IP generates an interrupt whenever the read FIFO of the IPIF service is full.

- rdfifo_flag <= RFIFO2IP_Full;
- interrupt <= rdfifo_flag;
- IP2Bus_IntrEvent <= interrupt;

Save and check the syntax of the user_logic.vhd file.

Custom IP Naming and Version:

For this design, the IP name given to the IP template will be retained. The version numbering will be changed by changing the middle number. The MPD File will be modified automatically by using the IPIF wizard import capability in the next section.

Importing the Custom IP back into EDK

This section outlines how to import the custom OPB IP peripheral back into the EDK environment, after having generated the OPB IPIF Template opb_ipif_template and having customized the user logic side functionalities.

1. From XPS, select Hardware \rightarrow Create/Import Peripheral ... as shown in Figure 16.



Figure 16: XPS IPIF Wizard Launch

2. In the Welcome Window, click **Next**. In the Peripheral Flow window shown in Figure 17, select **Import to XPS** and **Import existing peripheral**, then click **next**.

Create and Import Peripheral Wizard - Perip	heral 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 comp EDK repository. The interface files and directory structures re	liant peripheral, or help you import an existing peripheral into an XPS project or equired by EDK will be generated.
	Select flow
A CAR	Create templates for a new peripheral
Create Templates	Import existing peripheral
L 52	
	How description
	an XPS project or EDK repository. Such peripheral need to have ports and parameters that conform to the conventions required by EDK.
Import to XPS	
H T T T	
More Into	< Back Next> Cancel

DS967_17_120706

Figure 17: Selecting the Import Mode of the IPIF Wizard

3. In the Repository or Project window shown in Figure 18, select **To an XPS project**. In the Project field, select \$\\xapp967\. In the Peripheral will be placed under: field, select **C:\xapp967\pcores**, then click **Next**.

💠 Import Peripheral - Repository or Project	2
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 paccessed by multiple XPS projects.	roject. When stored in an EDK repository, the peripheral can be
To an EDK user repository (Any directory outside of your EDK	installation path)
Bepository:	Browse
To an ≚PS project	
Project: C:\xapp967\system.xmp	Browge
Reviewersh will be placed under	
C:\xapp967\pcores	
More Info	< Back Next> Cancel
	DS967_18_012207

Figure 18: XPS IPIF Wizard Import Directory

 In the name and Version window, provide the name of the peripheral. In the Name field, enter opb_ipif_template. Make the selections in the revision fields as shown in Figure 19, then click Next.

🐓 Import Peripheral - Name and Version		? 🛛
Name and Version Indicate the name of your peripheral and if using the I	EDK peripheral version naming scheme.	15
Enter name of the top VHDL entity or Verilog module of yr Namg: opb_ipif_template	our peripheral.	
Use version: 1.10.a Major revision: Minor revision: 1 1	Software compatibility revision:	
Cogical library name: opb_ipif_template_v1_10_a		
All the files for this peripheral are compiled into the logic either assumed to be available in the current project or imported along with the peripheral. Since all design files above may cause name space conflicts.	al library named above. If the peripheral refers to other lo in the repositories accessible through the current project are compiled in the same directory, using logical libraries	gical libraries, they are settings, or will be other than given
More Info	< Back Next >	<u><u>C</u>ancel</u>
		DS967_19_01220

Figure 19: Import IP Naming and Version

Note that the version of the IP has changed to reflect a change in the IP functionality from the original template.

5. In the Source Files Types window shown in Figure 20, select HDL Source Files (***vhd**, ***v**), then click **Next**.

Import Peripheral - Step 2
Source File Types Indicate the types of files that make up your peripheral.
Indicate the types of files that make up your peripheral. Image: HDL Source Files (*.vhd, *.v) Image: Netlist Files (*.edn, *.edf, *.ngo, *.ngc) Image: Documentation Files (*.doc, *.txt *.pdf, *)
More Info Cancel
D\$967_20_01220

Figure 20: IP Source Files Type Specification

In the HDL Source Files window, make the selections shown in Figure 21 to specify the IP source files that use the pao file of the original OPB IPIF template that was generated by the wizard. Browse to the initial location of this file under: \\ xapp967\pcores\ opb_ipif_template_v1_00_a\data\. Click Next.

and have a second s		
DL Source Files Indicate how this tool should locate the HDL files that make up your peri	pheral.	-
L language used to implement your peripheral: VHDL		
Use data (*.mpd) collected during a previous invocation of this tool		
		Browse
This tool will input the HDL meset and the logical ibranes they are con	iplied into nom the appropriate lines i	
		Browse
Use existing Peripheral Analysis Order file (".pao)		Browse
Use existing <u>Peripheral Analysis</u> Order file (".pao) (xapp967\pcores\opb_ipif_template_v1_00_a\data\opb_ipif_template	2_v2_1_0.pao	Browse
 Use existing Peripheral Analysis Order file (".pao) (xapp967\pcores\opb_ipif_template_v1_00_a\data\opb_ipif_template Browse to your HDL source and dependent library files (".vhd, ".vhd, " 	r_v2_1_0.pao	Browse

Figure 21: IP Source Files Specification

In the **Missing Source File - user logic.vhd** window, the search paths used by the wizard will be specified as shown in Figure 22. A path to the OPB Template is required. Select the **Add** button, then type \\xapp967\pcores\ to add the search path to the IP template location. Click **Continue**.

The wizard failed to locate the source file: opb_ipif_template_v1_00_a user_logic.vhd from the PAO file in the current searchable EDK repositories. Below are the current searchable EDK repositories. You can add extra reference paths and re-parse the PAO file. C:\edk_user_repository\ C:\EDK\hw\ C:\EDK\hw\ C:\EDK\hw\ C:\EDK\hw\ C:\EDK\hw\ C:\EDK\hw\ Cilck Ok to re-parse the PAO file or click Continue to skip the missing source I	Missing Source File - use	r_logic.vh	nd		?
opb_ipif_template_v1_00_a user_logic.vhd from the PAO file in the current searchable EDK repositories. Below are the current searchable EDK repositories. You can add extra referer paths and re-parse the PAO file. C:\edk_user_repository\ C:\EDK\hw\ Bemove Cick Ok to re-parse the PAO file or click Continue to skip the missing source I	The wizard failed to locate the sour	rce file:			
from the PAD file in the current searchable EDK repositories. Below are the current searchable EDK repositories. You can add extra referer paths and re-parse the PAD file. C:\edk_user_repository\ C:\EDK\hw\ Remove Click Ok to re-parse the PAD file or click Continue to skip the missing source I	opb_ipif_template_v1_00_a_u	iser_logic.vh	Ь		
Below are the current searchable EDK repositories. You can add extra referer paths and re-parse the PAO file. C:\edk_user_repository\ C:\EDK\hw\ Remove Click 0k to re-parse the PAO file or click Continue to skip the missing source I	from the PAO file in the current sea	archable EDK	(repositories		
C:\edk_user_repository\ C:\EDK\hw\ Remove Click Ok to re-parse the PAO file or click Continue to skip the missing source I	Below are the current searchable E paths and re-parse the PAO file.	EDK repositor	ries. You car	n add extra i	referenci
C:\EDK\hw\ Remove	C:\edk_user_repository\				id
Click Ok to re-parse the PAO file or click Continue to skip the missing source I	C:\EDK\hw\				
Click Ok to re-parse the PAO file or click Continue to skip the missing source I				Ren	nove
Click Ok to re-parse the PAO file or click Continue to skip the missing source I					
Click Ok to re-parse the PAO file or click Continue to skip the missing source I					
	Click Ok to re-parse the PAO file or	r click Contin	ue to skip th	e missina so	ource file
lik Continue Cancel		ontinue		ancel	
		Unande		ancer	

Figure 22: Adding a Search Path to the IPIF Wizard

In the HDL Analysis Information window shown in Figure 23, the system indicates that the wizard was not able to locate the user_logic.vhd and the opb_ipif_template.vhd source files.

Use the burn analyz Xilir	nx Platform Studio	braries a	nd set the HDL
vhdl	The tool is unable to intuite the top design file.	core	Add Files
vhdl 4	Click OK to take the last file in the analysis order as the top design file, or click Cancel to modify the list.	core core	Add Library
vhdl vhdl vhdl	OK Cancel	core core	Remove
vhdl		core	Move Up
vhdl prod	common_v2_o0_a C/EDK/hw/XilinxProcessorIPLib/r c_common_v2_00_a C/EDK/hw/XilinxProcessorIPLib/r c_common_v2_00_a C/EDK/hw/XilinxProcessorIPLib/r		Move Down
<		>	Modify

Figure 23: Adding Source Files Manually to the IP

6. Add the vhd files manually. Select the Add Files ... button, then in the Select ... HDL source files window shown in Figure 24, specify the location of the source files.

Select one or n	nore HDL so	urce files	? 🔀
Look in:	📄 vhdl	· + € 🕈 📰 •	
My Recent Documents	โร่มีopb_ipif_ โร่มีuser_log	template.vhd c.vhd	
Desktop My Documents			
My Computer			
My Network	File name:		Open
Maces	Files of type:	VHDL Source Files (*.vhd *,vhdl)	Cancel
		DS	967 24 012407



7. After the files have been added, a parsing error of the source code is displayed:

During the custom IP import process, the version of the custom IP has been changed from v.1.00.a to v.1.10.a; these are appended to the custom IP name to form a logical library which is referenced in the IP HDL Files. Open **opb_ipif_template.vhd**, and change:

From:

```
library opb_ipif_template_v1_00_a,
use opb_ipif_template_v1_00_a.all;
```

To:

```
library opb_ipif_template_v1_10_a;
use opb_ipif_template_v1_10_a.all;
```

Change

From:

USER_LOGIC_I : entity opb_ipif_template_v1_00_a.user_logic

To:

USER_LOGIC_I : entity opb_ipif_template_v1_10_a.user_logic

8. In the Bus Interface window, make the selections as shown in Figure 25. Because this is an OPB slave core with a DMA master IP, specify the bus interfaces to be OPB Master and Slave (MSOPB).

Identify the bus interfaces supported by your peripher	al
is interface is a group of related interface ports disting face(s) supported by your peripheral or indicate there	uished by a bus standard (OPB, PLB, DCR, LMB or FSL). Select the bus is no bus interface applies.
Select bus interface(s) On-chip Peripheral Bus interface ODB Master and Slave (MSOPB) Master Interface is optional	Device Control Register bus interface DCR Slave (SDCB)
OPB Slave (SOPB)	Local Memory Bus interface LMB Slave (SLMB)
Processor Local Bus interface PLB Master and Slave (MSPLB) Master interface is optional	Fast Symplex Link bus interface
PLB Slave (SPLB)	FSL Slave (SFSL)

Figure 25: Selecting the OPB as a Bus Interface

In the next window, a confirmation of which bus interface is used and its characteristics appears. Ignore the warning that the M_DBus is not used by this IP. Click **Next**.

9. Define the slave attachment parameters for the IP. In the **SOPB : Parameters** window, make the selections as shown in Figure 26.

🍇 Import Peripheral - SOPB : Parameter	? 🗵
SOPB : Parameter Define the SOPB bus interface parameter(s) for this perip	heral.
The SOPB bus interface is defined by a predefined set of por naming conventions, this tool has automatically done the sele register Space Parameter determine base address: Parameter determine high address:	ts and parameters. If your peripheral follows the standard ictions for you. Otherwise check off the values.
Memory Space Base Address Parameter High A	Address Parameter Cacheable Add
More Info	< Back Next > Cancel

Figure 26: Defining the SOPB Parameters

10. In the Identify Interrupt Signals window, make the selections shown in Figure 27.

Select IP2INTC_Irpt in the Select and configure interrupt(s) window. Under Interrupt sensitivity of port: IP2INTC_Irpt, select High level sensitive.

entify Interrupt Signals	
Identify the interrupt signals on your peripheral.	-
cate the attributes of the interrupt signals by checking the int t. EDK uses this information to automatically connect the inter- elect and configure interrupt(s) User FIFOs Status Flag IP2INTC_Ipt	terrupt port name on the left and then clicking on the radio buttons to th errupt ports of your peripheral. Interrupt sensitivity of port. IP2INTC_Irpt Ealling edge sensitive Low level sensitive Bising edge sensitive High level sensitive

Figure 27: Defining the IP Interrupt Signal and its Sensitivity

11. In the Parameter Attributes window, make the selections shown in Figure 28 to set the default values for the user parameters and for the IPIF parameters.

Double click on Default Value, then select virtex4, then click Next.

arameter Attributes Identify the parameters that require	special handling.		
lect the parameter on the left and fill ripheral into the system it is instantiate List User Parameters only -	in the attribute values to the right. Thes ed in. Attributes:	e attributes help the various tools in EDK	to integrate this
USER_ID_CODE	Parameter Name	C_FAMILY)	
FAMILY	Data Type	string	
	Default Value	virtex4	
	Display <u>a</u> dvanced attribut	es	

Figure 28: Defining Default Values for the User/IPIF Parameters

12. In the Port Attributes window, make the selections shown in Figure 29 to identify the ports that require special handling. Set the attributes to the user ports and user interrupts ports, if any. Because in this custom IP only the interrupt pin comes out from the user logic side, accept the default values.

After the selections have been made, click Next, then click Finish.

Import Peripheral - Port At ort Attributes	tributes	4	
identity the ports that require spe	cial handling.		
Colocit the part on the left and fill in th	o strukuto usluos to the right. These str	ibutos bolo the usrigue tools in EDK t	~
tegrate this peripheral into the syste	m it is instantiated in.	ibutes help the validus tools in EDIX to	
- List Interrupt Ports only -	Attributes:		
Jser_FIFOs_Status_Flag	Port Name	User_FIFOs_Status_Flag	
	Direction Mode	Output	
	Default Connection	DUS .	
	Vector Dimension		
	Display advanced attribu	ites	
More Info	< B.	ack Next > Ca	ncel

Figure 29: Defining the Advanced Attributes of the User Ports

The importing of the OPB custom IP back into the EDK environment is now completed.

The new custom IP name is opb_ipif_template, and the version is v.1.10.a.

Check the pcores directory to confirm that a new directory,

```
\ \ vapp967\ pcores\ pif_template_v1_10_a\, has been created by the Wizard for this new IP.
```

Configuring the New Custom OPB IP in an XPS Project

In XPS, the MHS file contains the parameters, the bus connections, and the port connections for the cores in the system. This file can be appended using the Add/Edit Core User Interface.

The configuration of the new IP will begin from the last block in the Design Flow for an OPB IPIF-based Custom IP diagram shown in Figure 1. After adding the newly-created IP core, the core will be configured, and then the system hardware will be implemented.

- 1. Open XPS, then open the project located under $\xapp967\xystem.xmp$.
- 2. In XPS, in the Project Information Area (left side of XPS GUI), select the IP Catalog tab.
- 3. Select **opb_ipif_template Version 1.10.a**, then drag and drop it into the system assembly view (right side of XPS GUI).

The custom IP has been added to the system IP components. The IP must now be connected to the bus.

- 4. To do so, make the selections as shown in Figure 30.
 - a. Select Bus Interface under Filters
 - b. Select opb_ipif_template_0 in the system assembly
 - c. Click on the green intersection with the OPB as shown in Figure 30 to connect the opb_ipif_template_0 OPB master and slave interfaces to the system OPB.



Figure 30: Connecting the Custom IP to the OPB Bus

5. In the system assembly view, Figure 30, double click on **opb_ipif_template_0** to open the opb_ipif_template_v1_10_a window in which address configurations can be assigned.

- 6. In the opb_ipif_template_v1_10_a window shown in Figure 31, assign a valid C_BASEADDR to the opb_ipif_template_0, for example: the value 0x60000000.
- 7. Assign a valid C_HIGHADDR to the opb_ipif_template_0, for example: the Value 0×6000 FFFF.

Note: The opb_ipif_template IP requires 0x10000 of the OPB memory space.

2 II	HDL Toggle Names Datasheet Restor
C_BASEADDR	0x6000000
C_HIGHADDR	0x6000ffff
C_OPB_AWIDTH	Auto Computed

Figure 31: Specifying the Address Range for the Custom IP

In the system assembly view shown in Figure 30, select **Ports** under Filters, then connect the custom IP interrupt signal to the OPB interrupt controller. Connect the User_FIFOs_Status_Flag port to the Global output port. This signal will be connected to an LED on the ML403 board.

The custom IP has been added into the XPS Project, and the core parameters have been configured. The system can now be implemented by XPS.

In XPS, click on Hardware and then on Generate bitstream to generate the system hardware.

During the OPB IPIF template generation process, the IPIF wizard generated a <drivers> folder in the project directory area for the template IP. These drivers will make the IP registers interface visible to the software application running on the embedded system.

In addition, during the generation process, some template hardware properties in the user logic side were modified. No changes were made in the software interface side. Therefore, the software drivers remain usable with the new custom IP without any modification.

To add some high-level functionality that the custom IP can perform, new functions based on the software interface to the drivers source code can be added.

The drivers are in the \\xapp967\drivers\ folder.

Custom IP Registers Software Interface:

These definitions are provided in the header file <code>opb_ipif_template.h</code> which is located under \\xapp967\drivers\opb_ipif_template_v1_00_a\src\.

In this header file all the memory mapping of the registers is provided per IPIF service. Note the following offsets:

IPIF Reset/Mir Space Register Offsets

- **IPIF Interrupt Controller Space Offsets**
- IPIF DMA & Scatter Gather Space Offsets
- IPIF Read Packet FIFO Register/Data Space Offsets

Customizing the OPB IPIF SW Interface and Drivers

IPIF Write Packet FIFO Register/Data Space Offsets

For each offset, the relative registers mapping is defined, thereby making the custom OPB IPIF services visible to the software application.

Also given are all the mask definitions required to set up the corresponding IPIF service and to understand the data field values of these registers. Note the following mask definitions:

IPIF Reset/Mir Masks

IPIF Interrupt Controller Masks

IPIF DMA & Scatter Gather Masks

IPIF Read Packet FIFO Masks

IPIF Write Packet FIFO Masks

The function prototypes and the definitions are provided in this header file. These functions allow the handling of the OPB IPIF custom IP services from the software application.

Drivers Functions Description

General Drivers functions

Write a value to the OPB_IPIF_TEMPLATE register:

void OPB_IPIF_TEMPLATE_mWriteReg(Xuint32 BaseAddress, unsigned RegOffset, Xuint32 Data).

Read a value from an OPB_IPIF_TEMPLATE register:

Xuint32 OPB_IPIF_TEMPLATE_mReadReg(Xuint32 BaseAddress, unsigned RegOffset).

A self-test function of the driver/device:

XStatus OPB_IPIF_TEMPLATE_SelfTest(void * baseaddr_p).

Note: This may be a destructive test if resets of the device are performed. baseaddr_p is the base address of the OPB_IPIF_TEMPLATE instance.

Functions for the RESET/MIR Services:

Reset the OPB_IPIF_TEMPLATE via software:

void OPB_IPIF_TEMPLATE_mReset(Xuint32 BaseAddress).

Read module identification information from OPB_IPIF_TEMPLATE device:

Xuint32 OPB_IPIF_TEMPLATE_mReadMIR(Xuint32 BaseAddress).

Functions for the DMA Services:

Reset DMA channel 0 (Write or Transmit side) of OPB_IPIF_TEMPLATE to initial state:

void OPB_IPIF_TEMPLATE_mResetDMA0(Xuint32 BaseAddress).

Reset DMA channel 1 (Read or Receive side) of OPB_IPIF_TEMPLATE to initial state:

void OPB_IPIF_TEMPLATE_mResetDMA1(Xuint32 BaseAddress).

Set/Get DMA control register of OPB_IPIF_TEMPLATE DMA channel:

void OPB_IPIF_TEMPLATE_mSetDMA0Control(Xuint32 BaseAddress, Xuint32 Mask).

void OPB_IPIF_TEMPLATE_mSetDMA1Control(Xuint32 BaseAddress, Xuint32 Mask).

Xuint32 OPB_IPIF_TEMPLATE_mGetDMA0Control(Xuint32 BaseAddress).

Xuint32 OPB_IPIF_TEMPLATE_mGetDMA1Control(Xuint32 BaseAddress).

Get DMA status register of OPB_IPIF_TEMPLATE DMA channel:

Xuint32 OPB_IPIF_TEMPLATE_mGetDMA0Status(Xuint32 BaseAddress).

Xuint32 OPB_IPIF_TEMPLATE_mGetDMA1Status(Xuint32 BaseAddress).

bool OPB_IPIF_TEMPLATE_mDMA0Error(Xuint32 BaseAddress).

bool OPB_IPIF_TEMPLATE_mDMA1Error(Xuint32 BaseAddress).

bool OPB_IPIF_TEMPLATE_mDMA0Done(Xuint32 BaseAddress).

bool OPB_IPIF_TEMPLATE_mDMA1Done(Xuint32 BaseAddress).

DMA channel 0 transfer between source address and destination address:

void OPB_IPIF_TEMPLATE_DMA0Transfer(Xuint32 BaseAddress, Xuint32 SrcAddress, Xuint32 DstAddress, Xuint32 ByteCount).

Note: The destination address must be local to the OPB_IPIF_TEMPLATE device.

DMA channel 1 transfer between source address and destination address:

void OPB_IPIF_TEMPLATE_DMA1Transfer(Xuint32 BaseAddress, Xuint32 SrcAddress, Xuint32 DstAddress, Xuint32 ByteCount).

Note: The source address must be local to the OPB_IPIF_TEMPLATE device.

Functions for the Read Packet FIFO Service:

Reset read packet FIFO of OPB_IPIF_TEMPLATE to its initial state:

void OPB_IPIF_TEMPLATE_mResetReadFIFO(Xuint32 BaseAddress).

Check status of OPB_IPIF_TEMPLATE read packet FIFO module:

bool OPB_IPIF_TEMPLATE_mReadFIFOEmpty(Xuint32 BaseAddress).

Xuint32 OPB_IPIF_TEMPLATE_mReadFIFOOccupancy(Xuint32 BaseAddress).

Read data from OPB_IPIF_TEMPLATE read packet FIFO module:

Xuint32 OPB_IPIF_TEMPLATE_mReadFromFIFO(Xuint32 BaseAddress).

Functions for the Write Packet FIFO Service:

Reset write packet FIFO of OPB_IPIF_TEMPLATE to its initial state:

void OPB_IPIF_TEMPLATE_mResetWriteFIFO(Xuint32 BaseAddress).

Check status of OPB_IPIF_TEMPLATE write packet FIFO module:

bool OPB_IPIF_TEMPLATE_mWriteFIFOFull(Xuint32 BaseAddress).

Xuint32 OPB_IPIF_TEMPLATE_mWriteFIFOVacancy(Xuint32 BaseAddress).

Write data to OPB_IPIF_TEMPLATE write packet FIFO module:

void OPB_IPIF_TEMPLATE_mWriteToFIFO(Xuint32 BaseAddress, Xuint32 Data).

Functions for the Interrupt Service:

Enable all possible interrupts from OPB_IPIF_TEMPLATE device:

void OPB_IPIF_TEMPLATE_EnableInterrupt(void * baseaddr_p).

Note: baseaddr_p: is the base address of the OPB_IPIF_TEMPLATE instance to be worked on

A default Example interrupt controller handler:

void OPB_IPIF_TEMPLATE_Intr_DefaultHandler(void * baseaddr_p). **Note:** baseaddr_p: is the base address of the OPB_IPIF_TEMPLATE instance to be worked on

Customizing the opb_ipif_template drivers:

To add custom functions to the opb_ipif_template drivers, define the required functions prototype, add them to the header file <code>opb_ipif_template.h</code>, then provide their source

code, either as inline macro functions in the same header file or write the body functions in the <code>opb_ipif_template.c</code> source file.

The OPB_IPIF_TEMPLATE_Write1KB functionI resets the custom IP, writes 1024 Bytes to the write FIFO using channel 0 DMA, that is then written and added to the drivers. Its prototype is:

XStatus OPB_IPIF_TEMPLATE_Write1KB(Xuint32 SrcAddress, Xuint32 BaseAddress);

The body of this function is added to the source file <code>opb_ipif_template.c</code>.

XStatus OPB_IPIF_TEMPLATE_Write1KB(Xuint32 SrcAddress, Xuint32 BaseAddress)
{
Xuint32 pfifo_status = 0;

Xuint32 IPIF_Temp_B_Addr;

Xuint32 pfifo_vacancy;

Xuint32 dma_ctrl_mask;

Xuint32 dma_status;

IPIF_Temp_B_Addr = BaseAddress; // Reset the OPB IPIF Template IP: OPB_IPIF_TEMPLATE_mReset(IPIF_Temp_B_Addr);

```
// Check status of OPB_IPIF_TEMPLATE write packet FIFO module:
pfifo_status = OPB_IPIF_TEMPLATE_mWriteFIFOFull(IPIF_Temp_B_Addr);
if (pfifo_status == 1 )
{
  return WRFIFO_FULL_MASK;
}
```

// Check that the Vacancy (in Words) of the Write FIFO can hold the 1KB of Data: pfifo_vacancy = OPB_IPIF_TEMPLATE_mWriteFIFOVacancy(IPIF_Temp_B_Addr);

```
if (pfifo_vacancy < 1024/4 )
{
return WRFIFO_FULL_MASK;
}</pre>
```

// Reset the Channel 0 DMA: OPB_IPIF_TEMPLATE_mResetDMA0(IPIF_Temp_B_Addr);

// Setup the Channel 0 DMA Control Register:

// - The Source Address is a Memory Buffer, therefore the Source Address should be incremented.

// - The Destination Address is Local since it is the Write FIFO.

dma_ctrl_mask = DMA_SINC_MASK | DMA_DLOCAL_MASK;

OPB_IPIF_TEMPLATE_mSetDMA0Control(IPIF_Temp_B_Addr, dma_ctrl_mask);

// Set the Source Address and Start the DMA Transfer.
// The destination address is local to the OPB_IPIF_TEMPLATE device:
// In this case it is the Write Packet FIFO:
OPB_IPIF_TEMPLATE_DMA0Transfer(IPIF_Temp_B_Addr, SrcAddress, 0, 1024);

```
// Check if the DMA Transfer has finished:
dma_status = OPB_IPIF_TEMPLATE_mDMA0Done(IPIF_Temp_B_Addr);
while(dma_status != 1 );
```

```
// Check if the DMA Transfer was achieved with No Errors:
dma_status = OPB_IPIF_TEMPLATE_mDMA0Error(IPIF_Temp_B_Addr);
```

```
if ( dma_status == 1)
{
  return DMA_DBE_MASK;
}
```

return XST_SUCCESS;

Assigning the Drivers to the Opb IPIF Template in XPS

In XPS, in the Software Platform Setting window, assign the drivers to the opb ipif template IP, such as for an EDK IP.

In XPS, use the Software Menu. Click on Software Platform Setting.

In the Software Platfom Settings window, select **Drivers**, then in the Drivers Configuration section, select **opb_ipif_template** under the Driver heading as shown in Figure 32.

cessor Informati cessor Instance	on [ppc405_0 ~~]							
Software Platform OS and Libraries Drivers Interrupt Handlers	Drivers Configuratio	Drivers Configuration						
	Perpheral	HW version	Instance	Driver		Version		
	#agppc_onth	2.00.a	#agopc_0	generic	¥	1.00.a		
	plb_v34	1.02.a	plb	pibarb	~	1.01.a		
	opb_v20	1.10.c	opb	opbarb	~	1.02.a		
	ptb2opb_bridge	1.01.a	plb2opb	pib2opb	~	1.00.a		
	opb_uartite	1.00.b	RS232_Uat	uartike	4	1.01.a		
	plb_bram_if_crift	1.00.b	plb_bram_if_cntk_1	bram	~	1.00.a		
	opb_bram_il_critk	1.00.a	opb_bram_it_crifk_1	bram	~	1.00.a		
	opb_bram_il_crift	1.00.a	opb_bram_il_crift_2	beam	4	1.00.a		
	opb_intc	1.00.c	opb_intc_0	inte	~	1.00.c		
	opb_pit_template	1.10 a	oph_pit_template_0	opb_gd_template		1.00.a		

Figure 32: Assigning Drivers to the opb_ipif_template IP

Once LibGen is run from XPS, the drivers will be precompiled into the libxil.a Library, the header file is copied to the <include> folder, and the source files will be copied into the <libsrc> folder.

These LibGen generated folders are located in the XPS project area under the Processor Instance Name folder.

Because the custom IP has interrupt capability, an interrupt handler can be assigned if this IP is used with interrupt in the system. Using the Software Platform Setting GUI, assign the interrupt service routine for the opb_ipif_template IP by selecting **Interrupt Handlers**, then typing the function name, **My_ip_Interrupt_Handler** in the Handler Interrupt column as show in Figure 33. Click **OK**.

🗢 Software Platfo	rm Settings				
Processor Information Processor Instance	n [pp:405_0] ~				
Software Platform	Interrupt Handlers Configuration:				
OS and Libraries	Port Name	Interrupt Handler	Default Handler	Type	Description
Drivers	inte : opb_inte_0				1910///00/2010
Interrupt Handlers	- Irq	My_ip_Interrupt_Handle	Anto_DeviceInterruptHandler	string	Interrupt handler for
	iii opb_ipit_template : opb_ipit_templi IP2INTC_irpt	ate_0	OPB_IPIF_TEMPLATE_Intr_Defaulth	landler string	Interrupt handler for
	¢		1		
					K Cancel
				100	1. (b)

Figure 33: Assigning Interrupt Handler to the opb_ipif Template IP

Writing a SW Application using the New Custom IP

After the software interface has been established, the next step is to write the software for the hardware platform that was just constructed.

The data flow illustrated in Figure 34 is implemented in the application software.



Figure 34: Data Flow Diagram

In the Reference Design associated with this Application Note, a SW Project is given in which the Processor creates a Data Structure in the BRAM Buffer 1, and sets the opb ipif template IP DMA Channel 0 engine to transfer the Data Buffer to its Transmit FIFO.

The Data is then looped back inside the custom logic of the IP to the Receive FIFO; once the Receive FIFO is full, an Interrupt Signal is sent to the Processor to decide what to do with the Data Packet Received by the Custom OPB IP.

This kind of approach is useful for Data Encryption if the user logic is encrypting the Data before looping it back to the Read FIFO of the Custom IP.

Building the SW Application in XPS:

The first step in building the required software application to run on the new hardware platform is to understand the mechanism already implemented on the given design. The software application is represented by the flow chart in Figure 35.

In this application the processor starts by initializing the BRAM Buffer 1 with known data values, it sets the opb ipif template DMA Channel 0 engine to transfer the data buffer from BRAM 1 to its transmit FIFO. Once the DMA operation is done, the processor checks if the DMA operation was successful or not, then waits for an interrupt from the opb ipif template IP.

Once the receive FIFO is full, i.e. the data packet is ready for the processor consumption, the interrupt service handler is called, which triggers another DMA transfer from the receive FIFO using DMA Channel 1. Once the transfer to the memory buffer 2 is finished, the processor proceeds to checking if the content of BRAM 2 buffer is as expected (same as BRAM 1 buffer); a status message is then displayed to the UART which is used as a user interface in this application.

In this transfer process using the transmit and receive DMA engines, the processor needs to initialize the DMA registers accordingly using the set of drivers associated with the custom IP in this design.



Figure 35: DMA Operation with the OPB IPIF Template

Implementing and Verifying the Design

Now the Design is ready to be implemented in order to download it to the Board and verify the System functionality over the UART Terminal.

- 1. Connect the Board to the PC using a Serial Cable and the JTAG Cable and Power up the Board.
- 2. Start a Hyper-terminal session with the following settings:
 - com1 Bits per second: 9600 Data bits: 8 Parity: None Stop bits: 1

Flow control: None

- 3. Implement the Design in XPS and download it to the Board.
- 4. The HyperTerminal Window displays as shown in Figure 36.



Figure 36: HyperTerminal Display

References DS414 OPB IPIF Architecture - http://www.xilinx.com/bvdocs/ipcenter/data_sheet/opb_ipif.pdf

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
2/5/07	1.0	Initial Xilinx release.
2/26/07	1.1	Changed title; Added reference link