The Xilinx EDK Toolset: Custom IP Cores

Creating and using custom IP cores

By Jason Agron

What is an IP Core?

- IP = Intellectual Property
- Why is it called this?
  - In our case, IP is “soft”.
    - Not physical.
  - It is merely a (soft) “description” of a device.
    - Usually in VHDL or Verilog.
- Why is this cool?
  - Can be open-source.
    - Can be understood and studied.
    - Can be customized.
  - Portable…
    - It is a model - can be simulated or implemented.
      - FPGA.
      - ASIC.

Where Do IP Cores Come From?

- For those who do not design HW…
  - Many are provided by vendors.
  - Xilinx provides many within it’s IP catalog.
- For those that can design HW…
  - You can make your own.
    - From scratch.
    - Using other soft/hard components.

What Is A Typical IP Core?

- Any digital device that you have seen could be implemented as a soft IP core…
  - As long as it can fit "inside" an FPGA.
- Some examples:
  - CPUs
  - Graphics cards
  - Network cards
  - Specialized processors (DSPs, FPUs, DataFlow)
  - Memory banks

What Do Soft IP Cores Enable?

- They enable a programmer/designer to combine pieces of IP at will in order to form a custom SoC within an FPGA.
- No soldering!
  - Just connect the inputs and outputs of the respective IP cores.
  - Done within VHDL/Verilog or a scripting language.

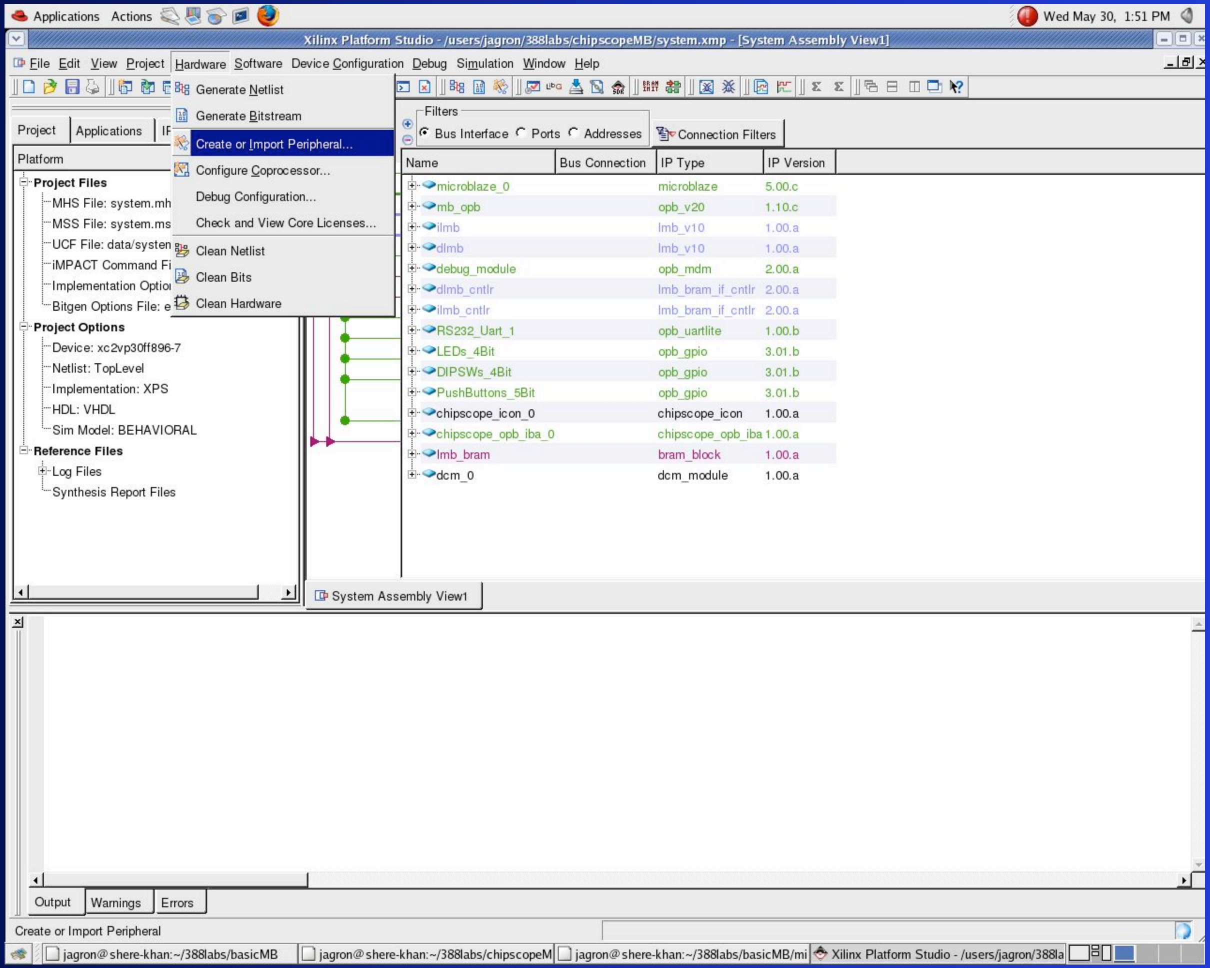
How To Create Custom IP Cores

- XPS has a built-in wizard…
  - Click on “Hardware…”
  - Select “Create or Import Peripheral…”
- The wizard allows one to…
  - Create a new piece of IP.
  - Select it’s interface (PLB, OPB, FSL).
  - Select default features to include.
  - Select it’s generation parameters…
    - VHDL or Verilog, etc.

Our Goal

- Create a custom IP core…
  - With an OPB interface.
  - With 4 SW-accessible registers
  - With Reset/MIR support.
  - Implemented in VHDL.
- The result:
  - Very simple IP core.
  - 4 storage locations (readable/writable).

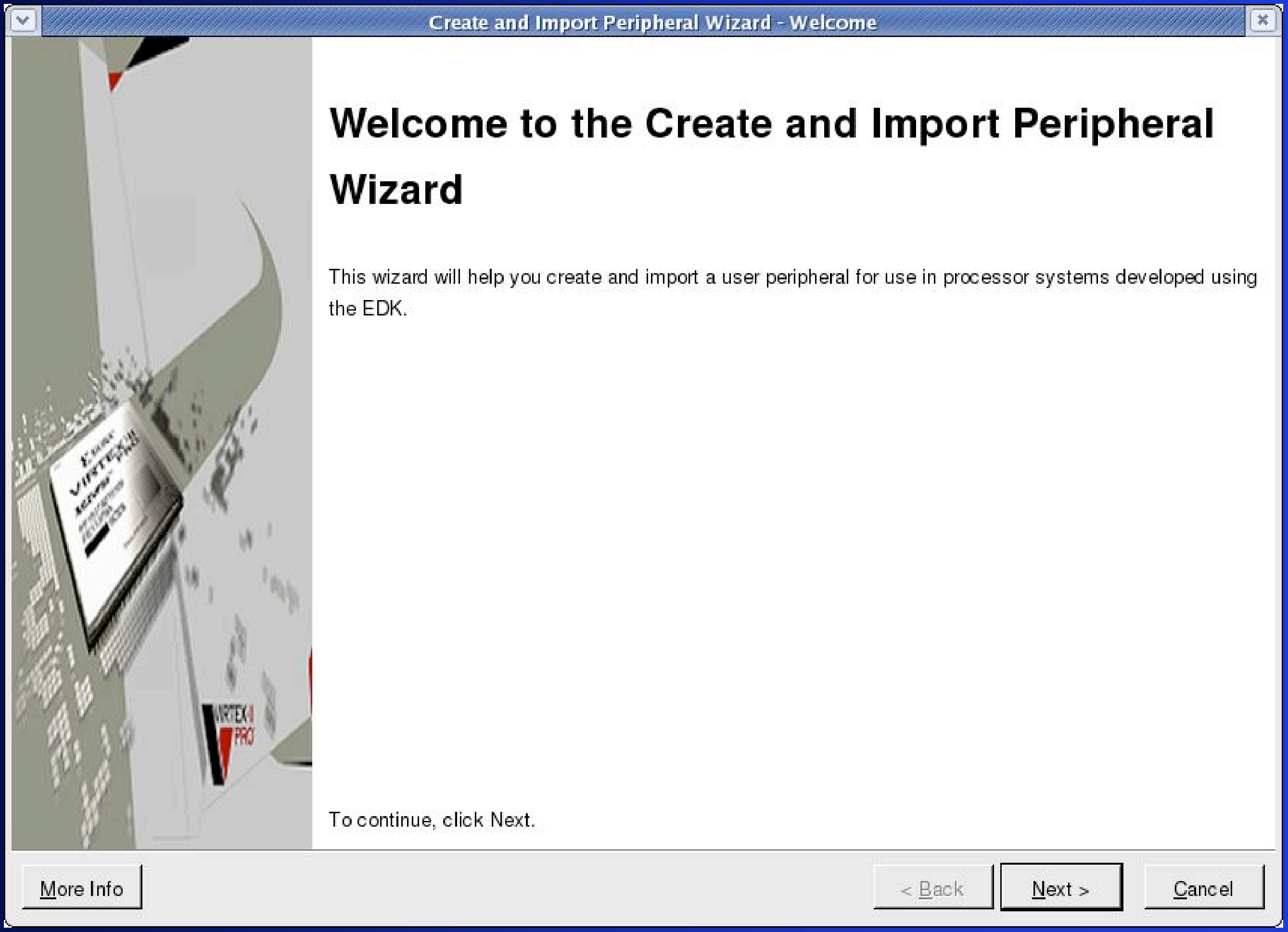
XPS - Creating Custom IP


Applications Actions
Wed May 30, 1:51 PM
Xilinx Platform Studio - /users/jagron/388labs/chipscopeMB/system.xmp - [System Assembly View1]
File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help
Generate Netlist
Generate Bitstream
Create or Import Peripheral...
Configure Coprocessor...
Debug Configuration...
Check and View Core Licenses...
Clean Netlist
Clean Bits
Clean Hardware
Project
Applications
Filters
Bus Interface
Ports
Addresses
Connection Filters
System Assembly View1
Output
Warnings
Errors
Create or Import Peripheral
jagron@shere-khan:~/388labs/basicMB
jagron@shere-khan:~/388labs/chipscopeM
jagron@shere-khan:~/388labs/basicMB/mi
Xilinx Platform Studio - /users/jagron/388la

Platform
- Project Files
  - MHS File: system.mh
  - MSS File: system.ms
  - UCF File: data/systen
  - iMPACT Command Fi
  - Implementation Optio
  - Bitgen Options File: e
- Project Options
  - Device: xc2vp30ff896-7
  - Netlist: TopLevel
  - Implementation: XPS
  - HDL: VHDL
  - Sim Model: BEHAVIORAL
- Reference Files
  - Log Files
  - Synthesis Report Files

Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	5.00.c
mb_opb		opb_v20	1.10.c
ilmb		lmb_v10	1.00.a
dlmb		lmb_v10	1.00.a
debug_module		opb_mdm	2.00.a
dlmb_cntlr		lmb_bram_if_cntlr	2.00.a
ilmb_cntlr		lmb_bram_if_cntlr	2.00.a
RS232_Uart_1		opb_uartlite	1.00.b
LEDs_4Bit		opb_gpio	3.01.b
DIPSWs_4Bit		opb_gpio	3.01.b
PushButtons_5Bit		opb_gpio	3.01.b
chipscope_icon_0		chipscope_icon	1.00.a
chipscope_opb_iba_0		chipscope_opb_iba	1.00.a
lmb_bram		bram_block	1.00.a
dcm_0		dcm_module	1.00.a

XPS - Wizard Startup

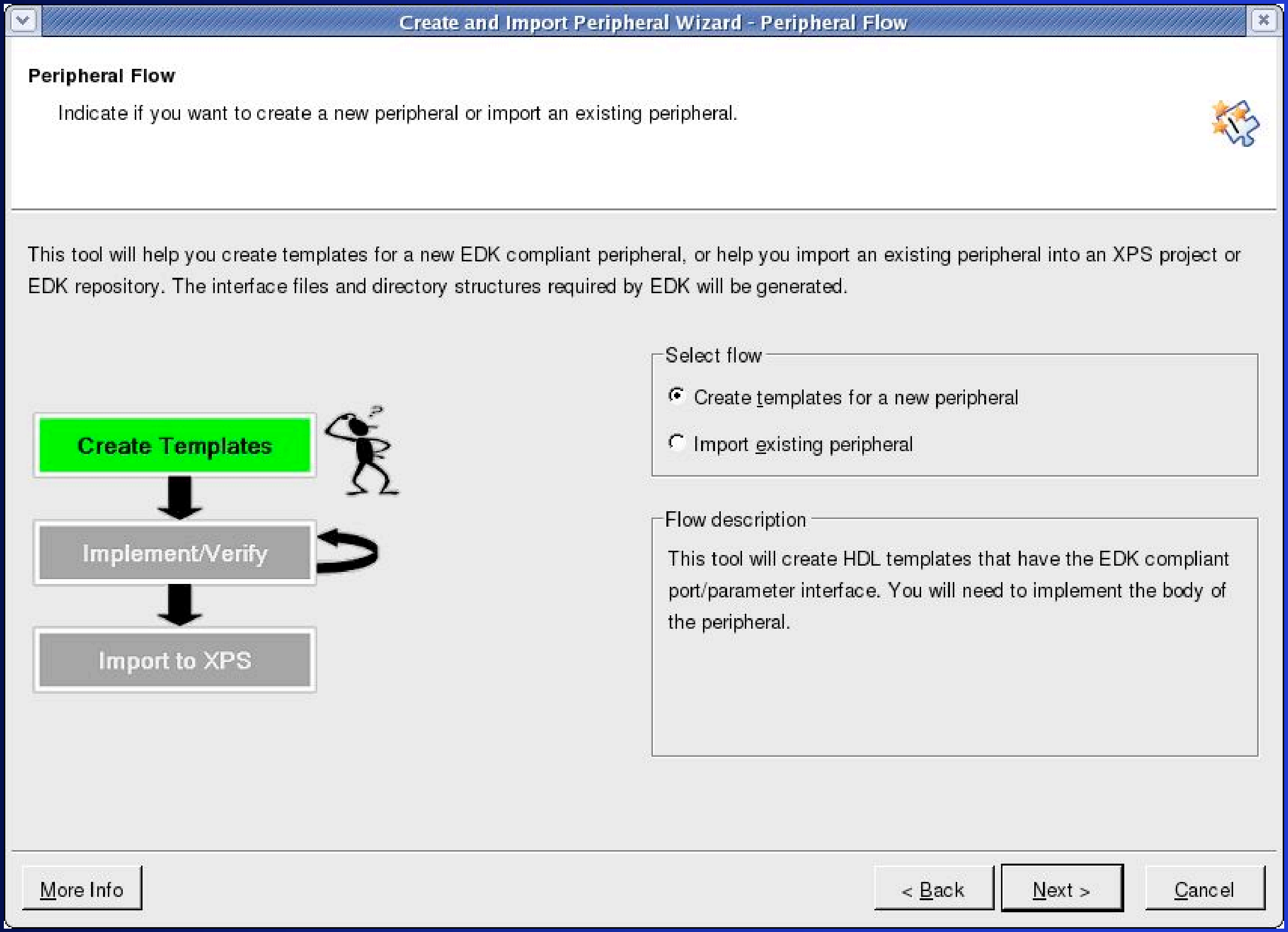

Create and Import Peripheral Wizard - Welcome

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.

XPS - Create New Peripheral



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

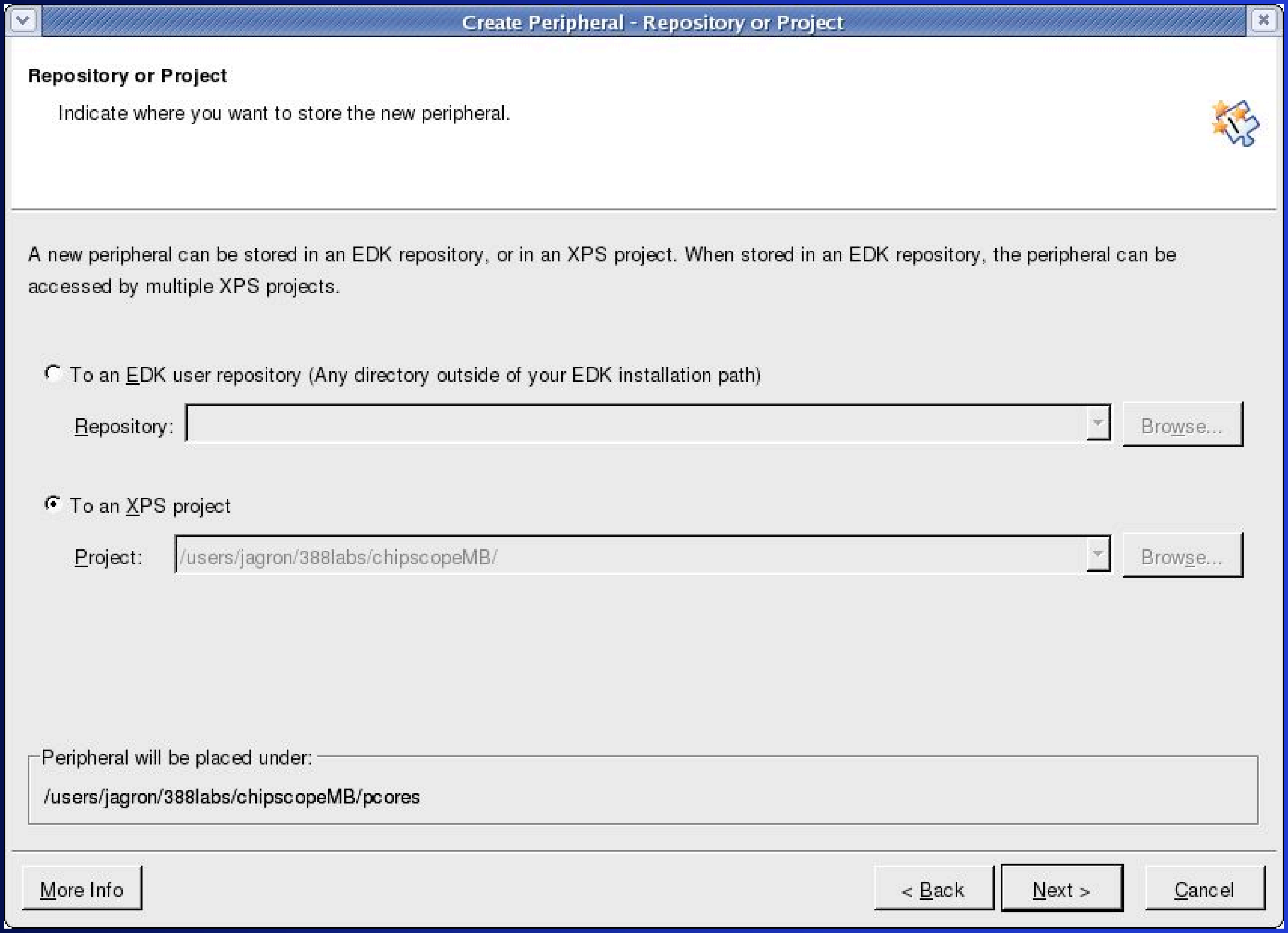
[✓ Create templates for a new peripheral]

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.

XPS - Select Storage Location



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:

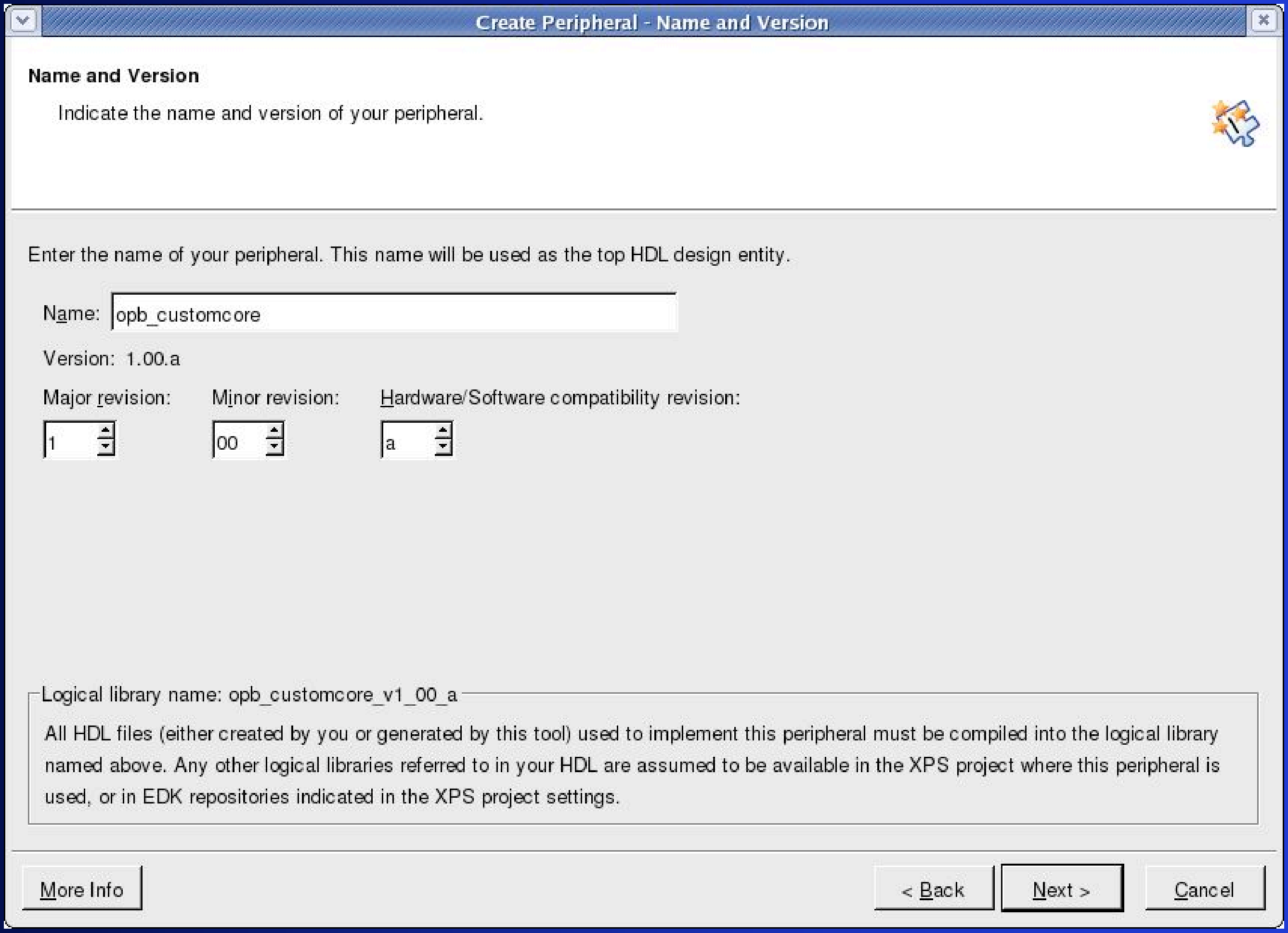
[✓ To an XPS project]

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

Peripheral will be placed under:

/users/jagron/388labs/chipscopeMB/pcores

XPS - Select Name & 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

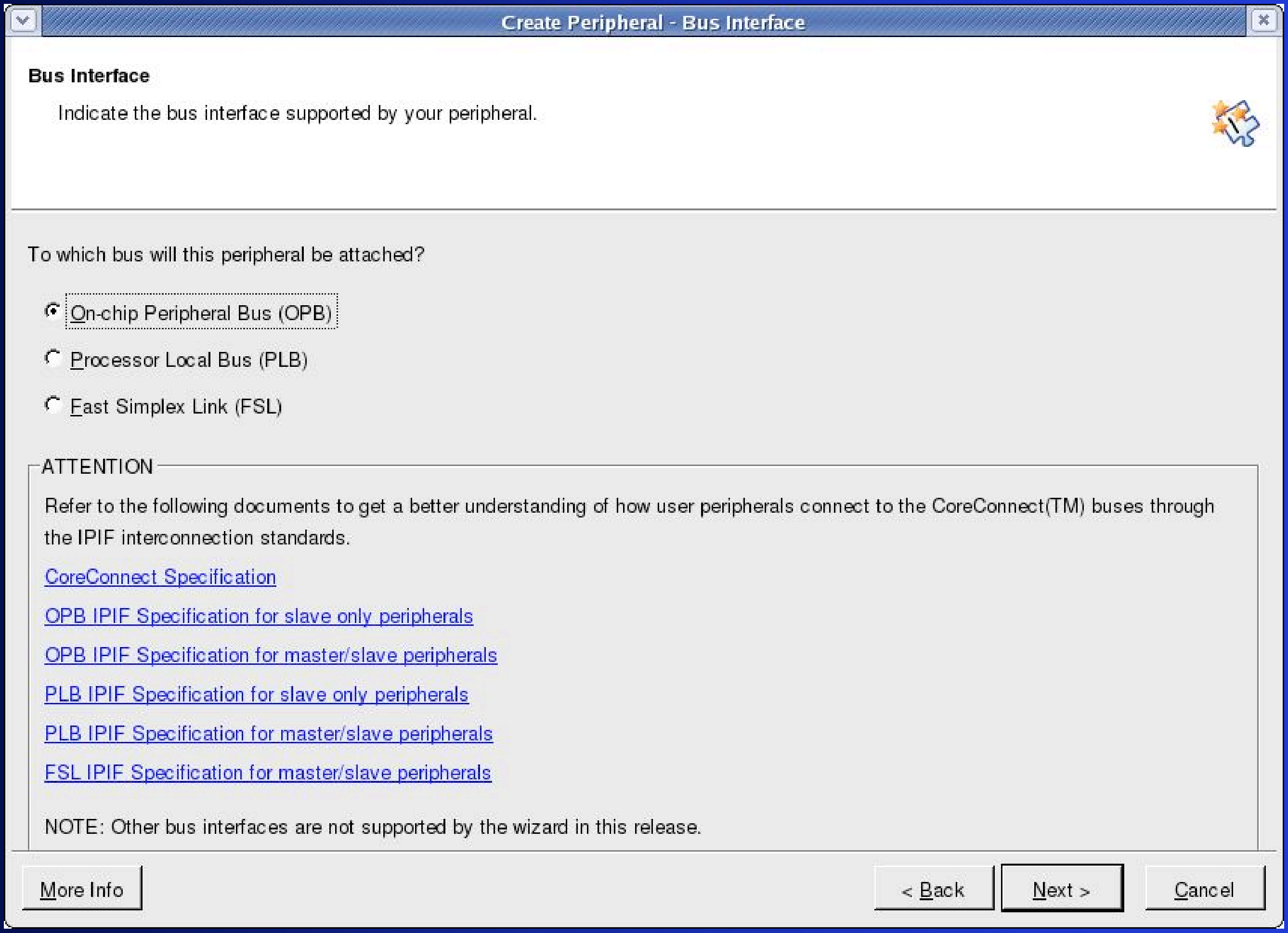
Minor revision: 00

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.

XPS - Interface Selection



Bus Interface

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

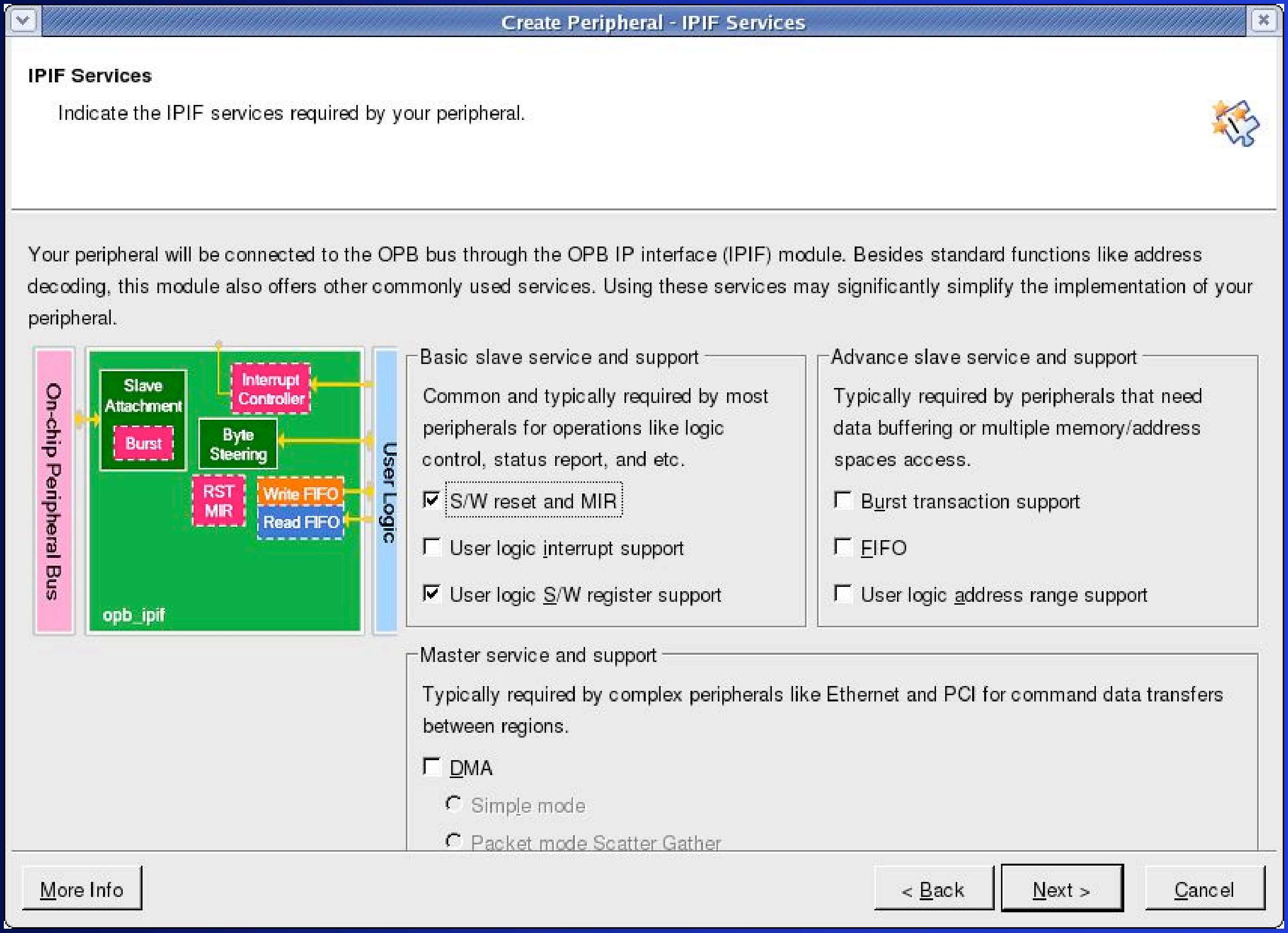
PLB IPIF Specification for slave only peripherals

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.

XPS - Feature/Service Selection



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.

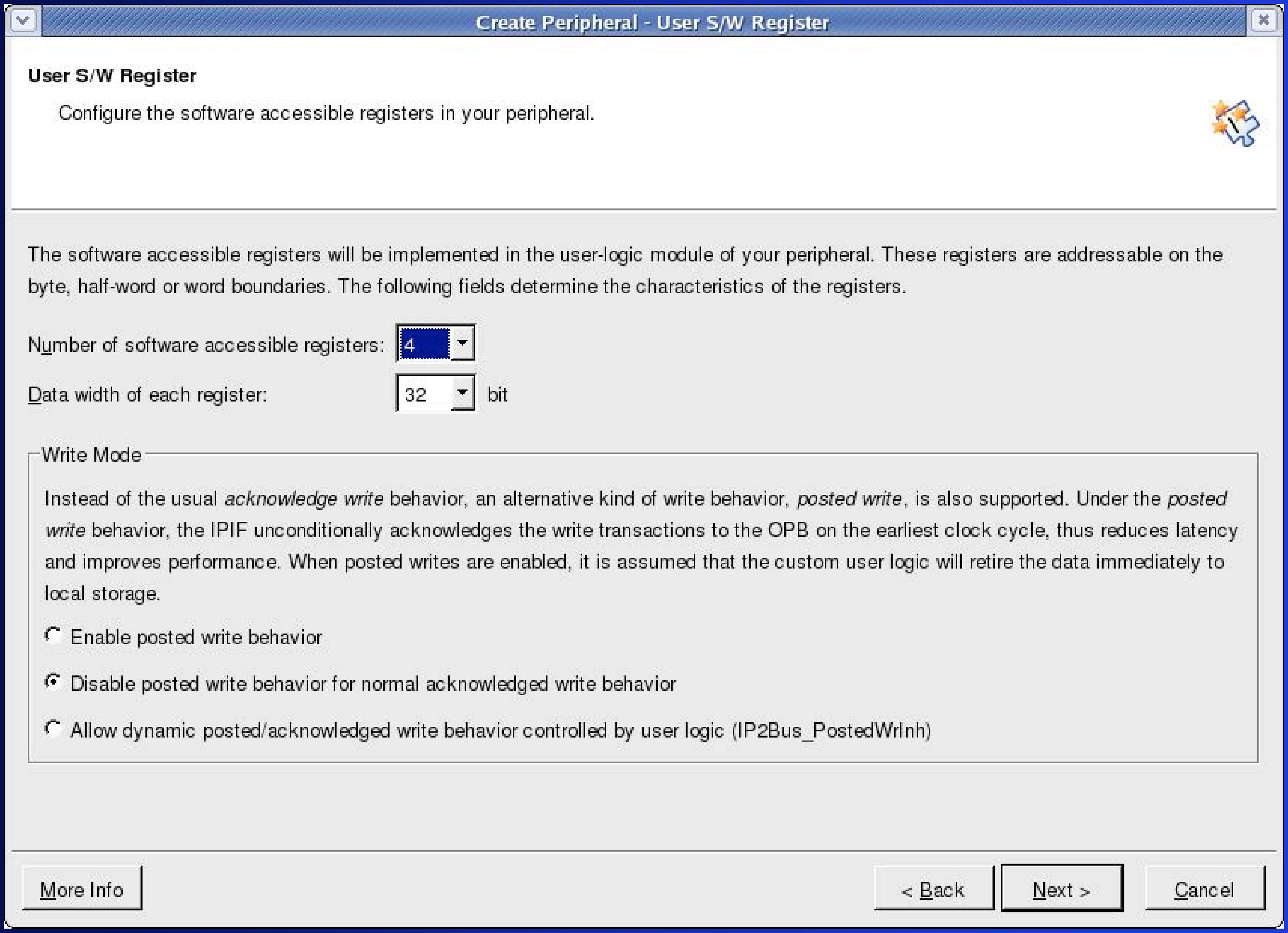
- [ ] Burst transaction support
- [ ] FIFO
- [ ] User logic address range support

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

XPS - S/W Register Selection



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

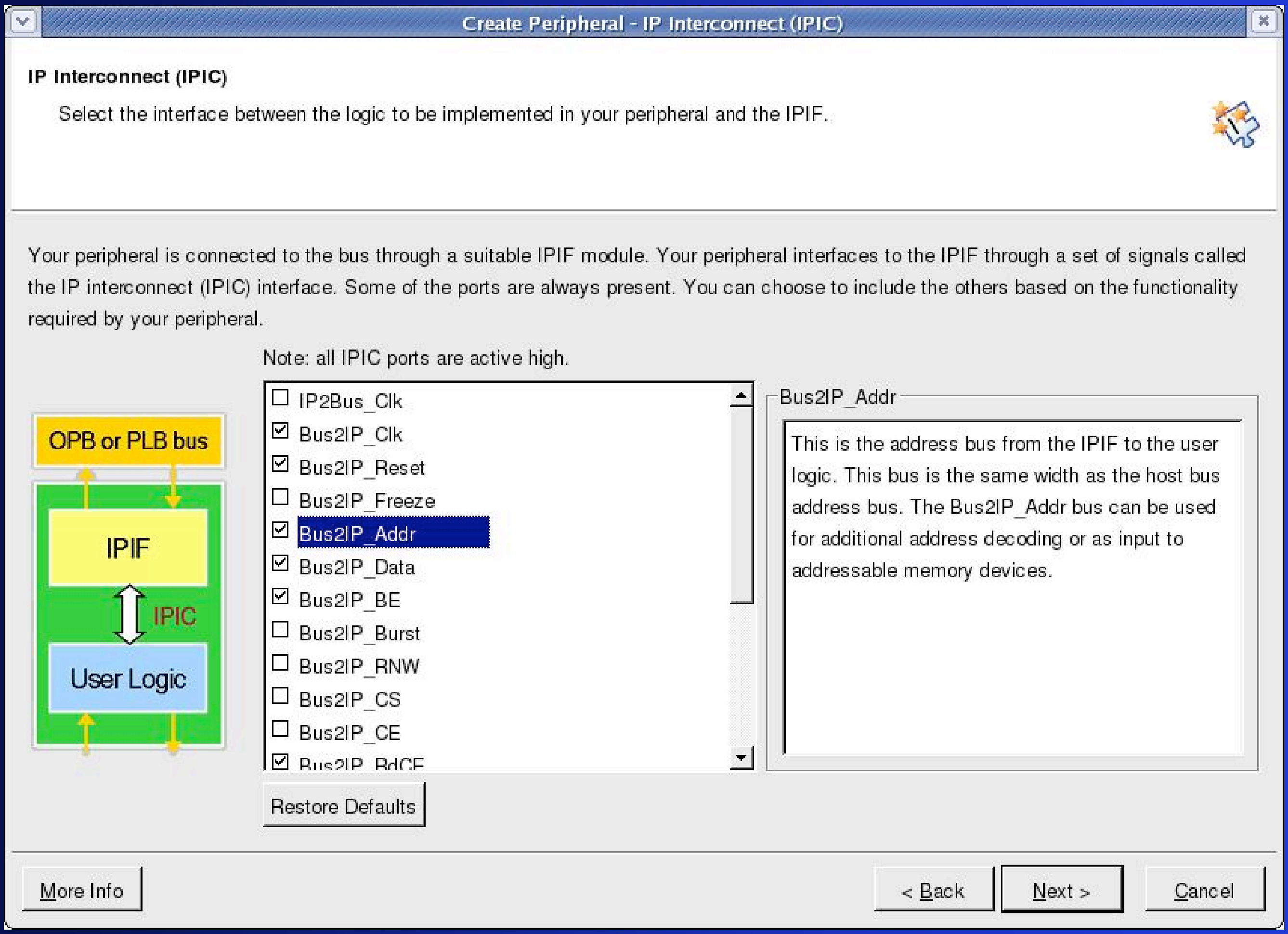
Data width of each register: 32 bit

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)

XPS - IPIC Configuration



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.

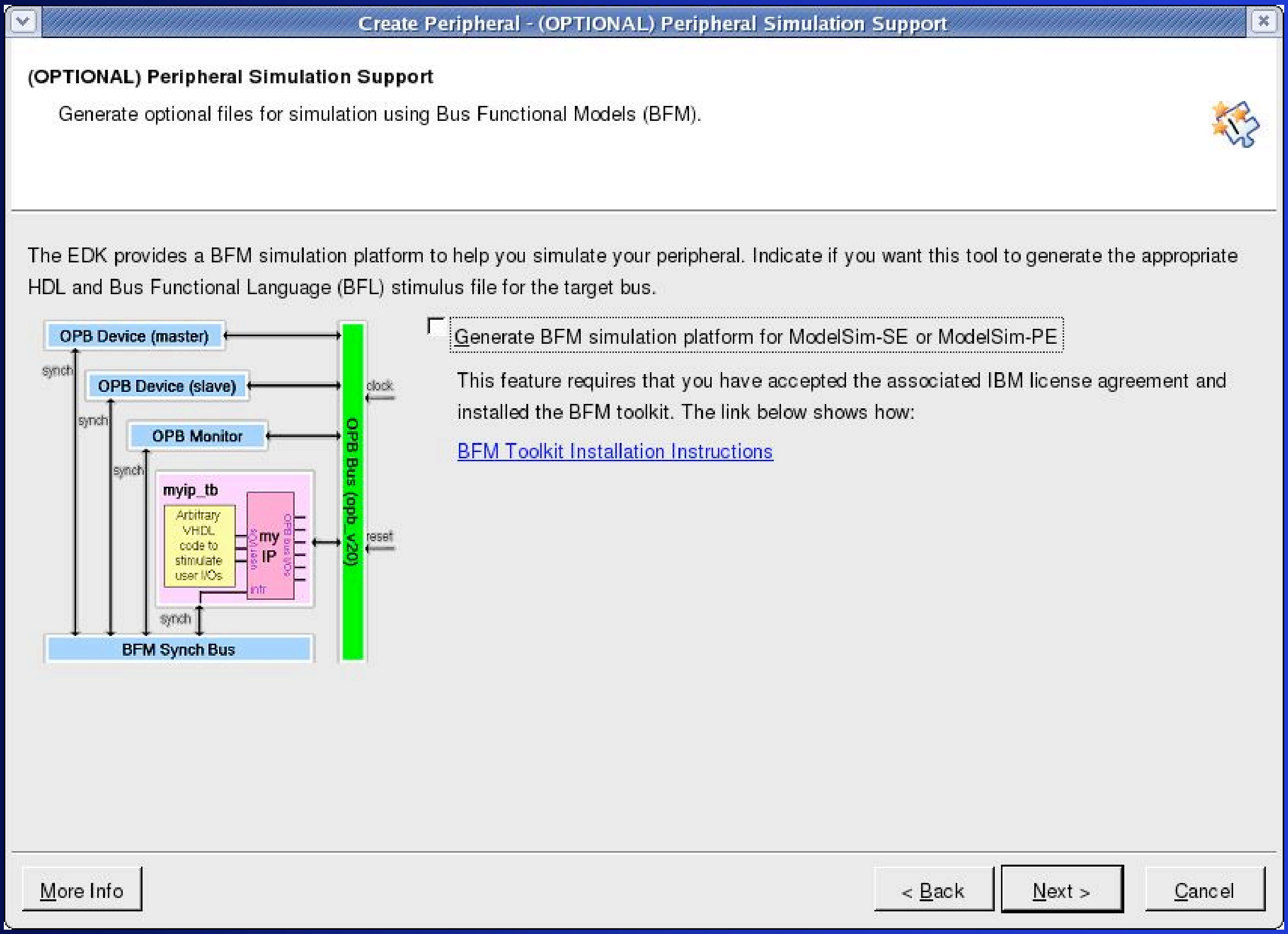
Note: all IPIC ports are active high.

- ☐ 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.

XPS - Simulation Support


Create Peripheral - (OPTIONAL) Peripheral Simulation Support

(OPTIONAL) Peripheral Simulation Support

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

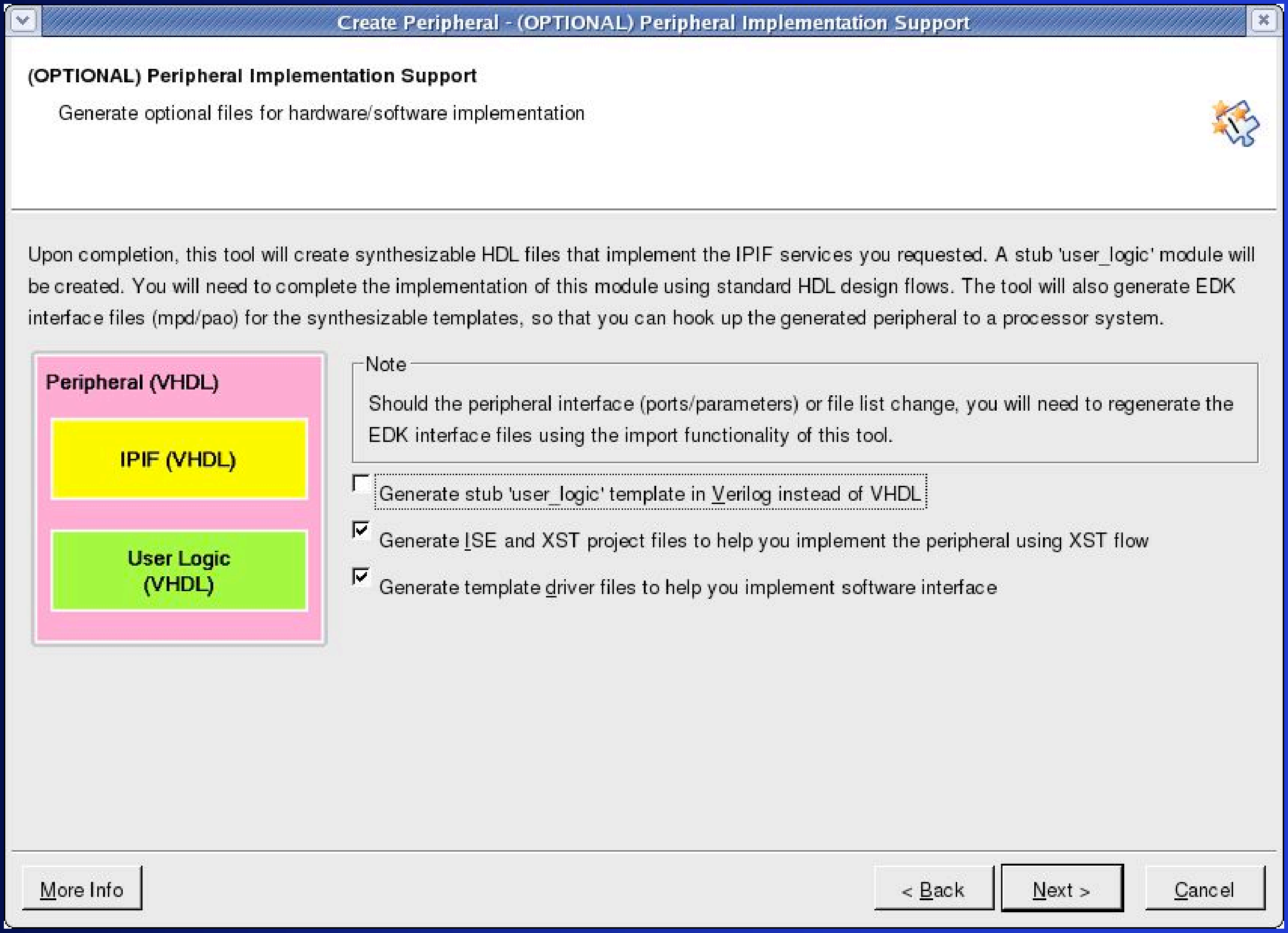
The EDK provides a BFM simulation platform to help you simulate your peripheral. Indicate if you want this tool to generate the appropriate HDL and Bus Functional Language (BFL) stimulus file for the target bus.

☐ 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

XPS - Implementation Support



Create Peripheral - (OPTIONAL) Peripheral Implementation Support

(OPTIONAL) Peripheral Implementation Support

Generate optional files for hardware/software implementation

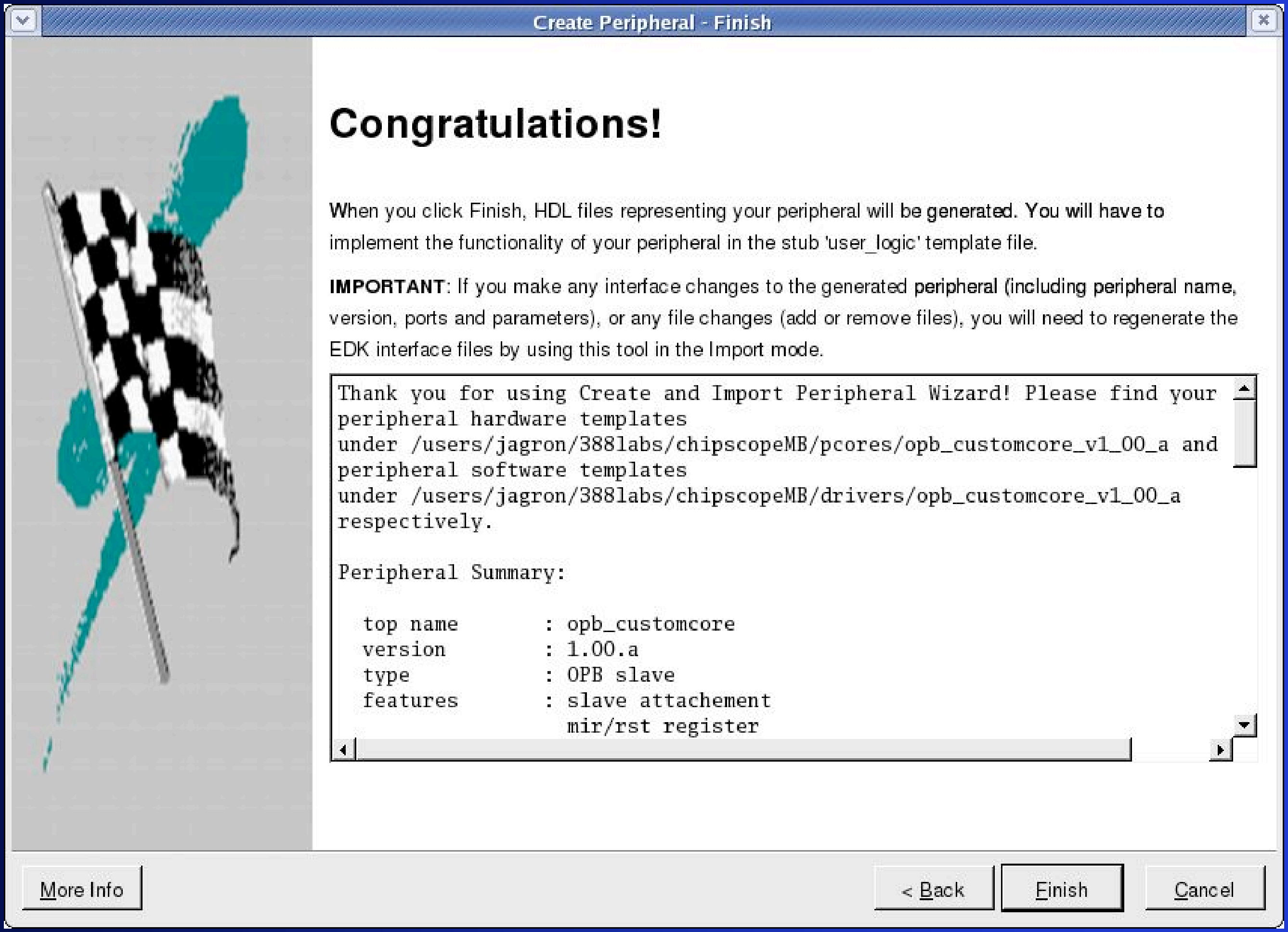
Upon completion, this tool will create synthesizable HDL files that implement the IPIF services you requested. A stub 'user_logic' module will be created. You will need to complete the implementation of this module using standard HDL design flows. The tool will also generate EDK interface files (mpd/pao)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

XPS - IP Creation Complete



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.

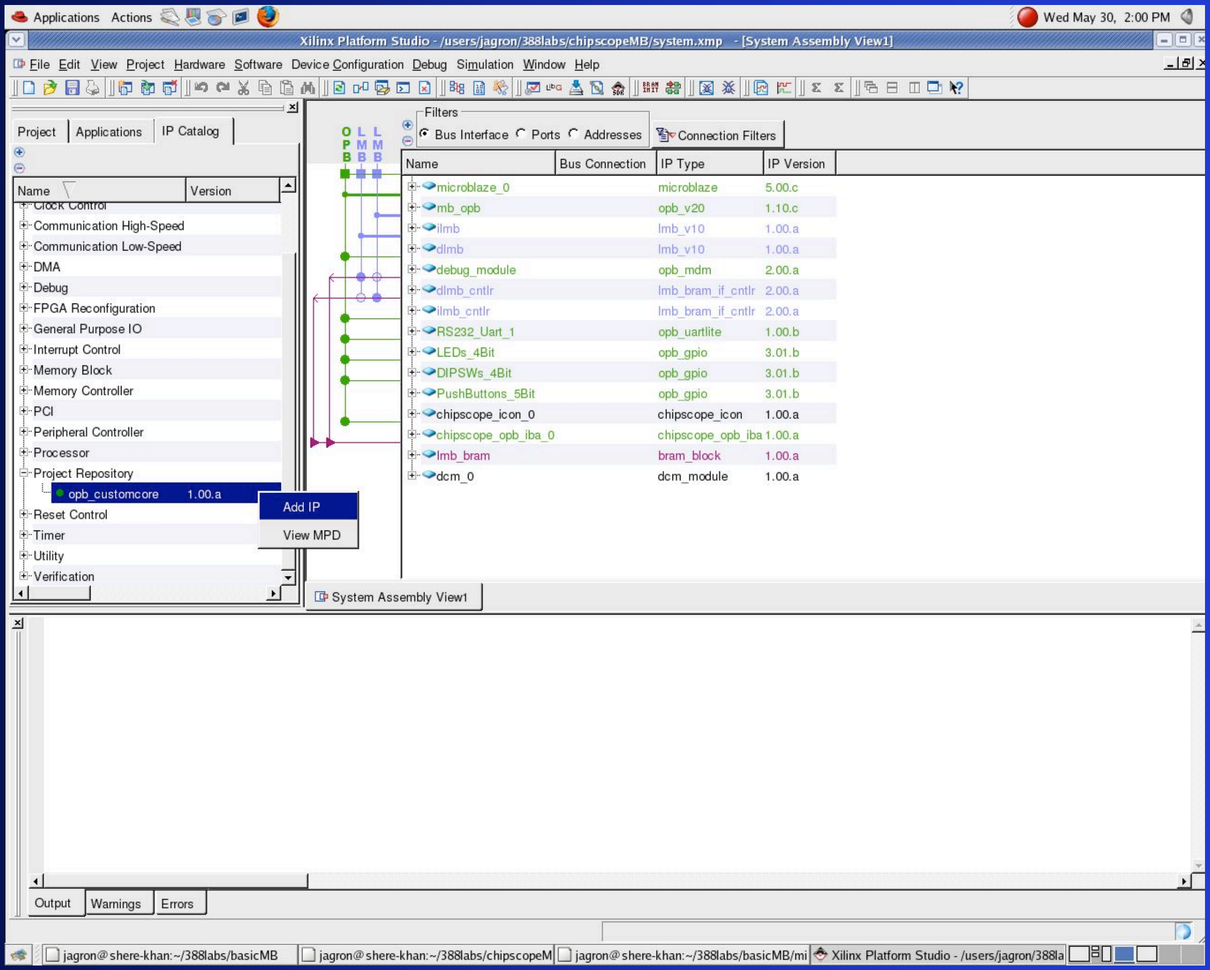
IMPORTANT: If you make any interface changes to the generated peripheral (including peripheral name, version, ports and parameters), or any file changes (add or remove files), you will need to regenerate the EDK interface files by using this tool in the Import mode.

```
Thank you for using Create and Import Peripheral Wizard! Please find your
peripheral hardware templates
under /users/jagron/388labs/chipscopeMB/pcores/opb_customcore_v1_00_a and
peripheral software templates
under /users/jagron/388labs/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
```

IP Created, Now What?

- We have just created a piece of IP that is now in the **project repository**.
- How do we use it?
  - It must first be added into the system.
    - Added into the system.
    - Connected to the bus.
    - Configured (address range, ports).
- How do we see it?
  - (HINT) IP Catalog Tab

XPS - Adding IP To The System

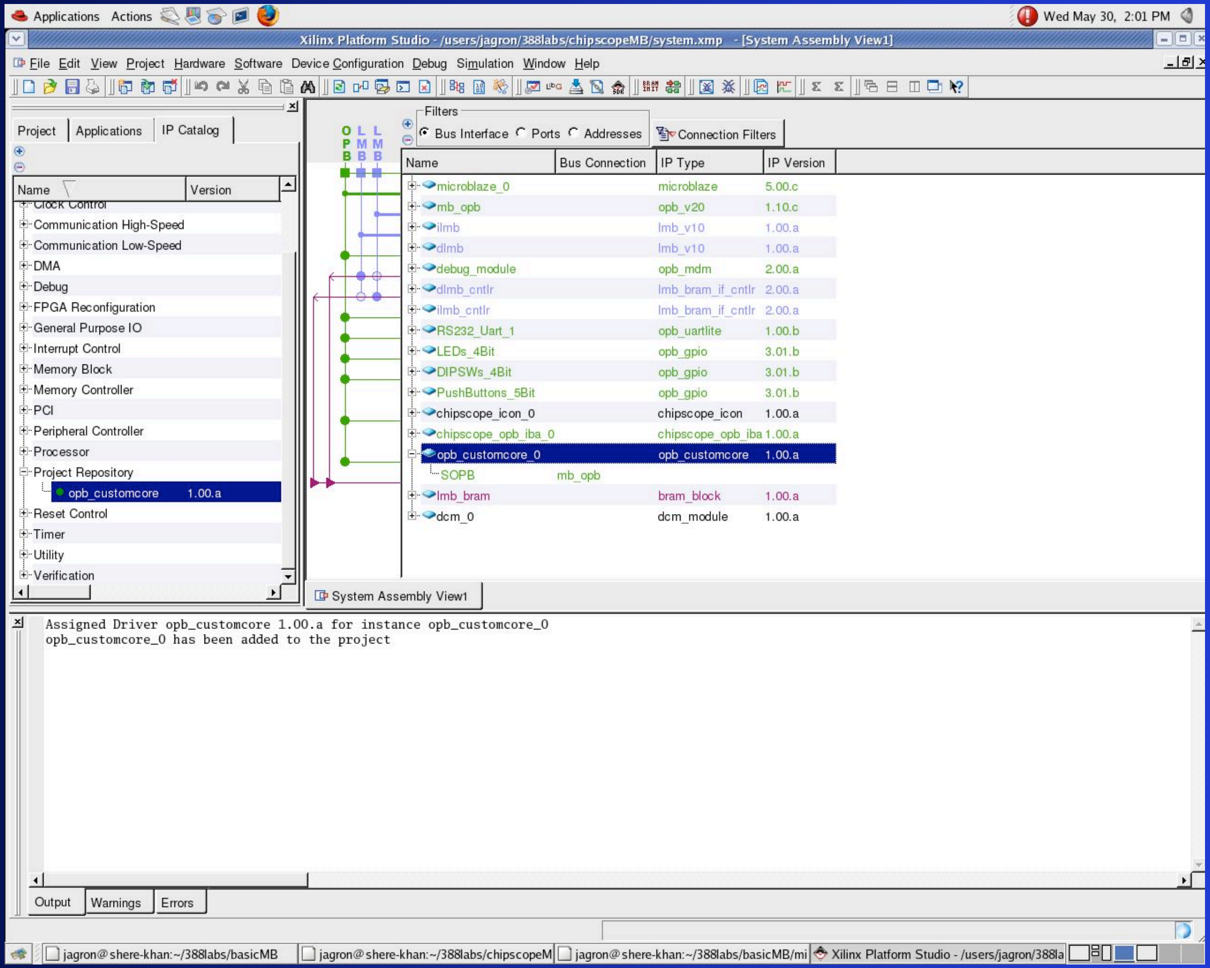

Applications Actions
Wed May 30, 2:00 PM
Xilinx Platform Studio - /users/jagron/388labs/chipscopeMB/system.xmp - [System Assembly View1]
File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help
Project Applications IP Catalog
OPB LMB LMB
Add IP
View MPD
System Assembly View1
Output Warnings Errors
jagron@shere-khan:~/388labs/basicMB
jagron@shere-khan:~/388labs/chipscopeM
jagron@shere-khan:~/388labs/basicMB/mi
Xilinx Platform Studio - /users/jagron/388la

Filters: (● Bus Interface) ○ Ports ○ Addresses — Connection Filters

Name	Version
Clock Control	
Communication High-Speed	
Communication Low-Speed	
DMA	
Debug	
FPGA Reconfiguration	
General Purpose IO	
Interrupt Control	
Memory Block	
Memory Controller	
PCI	
Peripheral Controller	
Processor	
Project Repository	
opb_customcore	1.00.a
Reset Control	
Timer	
Utility	
Verification	

Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	5.00.c
mb_opb		opb_v20	1.10.c
ilmb		lmb_v10	1.00.a
dlmb		lmb_v10	1.00.a
debug_module		opb_mdm	2.00.a
dlmb_cntlr		lmb_bram_if_cntlr	2.00.a
ilmb_cntlr		lmb_bram_if_cntlr	2.00.a
RS232_Uart_1		opb_uartlite	1.00.b
LEDs_4Bit		opb_gpio	3.01.b
DIPSWs_4Bit		opb_gpio	3.01.b
PushButtons_5Bit		opb_gpio	3.01.b
chipscope_icon_0		chipscope_icon	1.00.a
chipscope_opb_iba_0		chipscope_opb_iba	1.00.a
lmb_bram		bram_block	1.00.a
dcm_0		dcm_module	1.00.a

XPS - Connecting IP To Bus

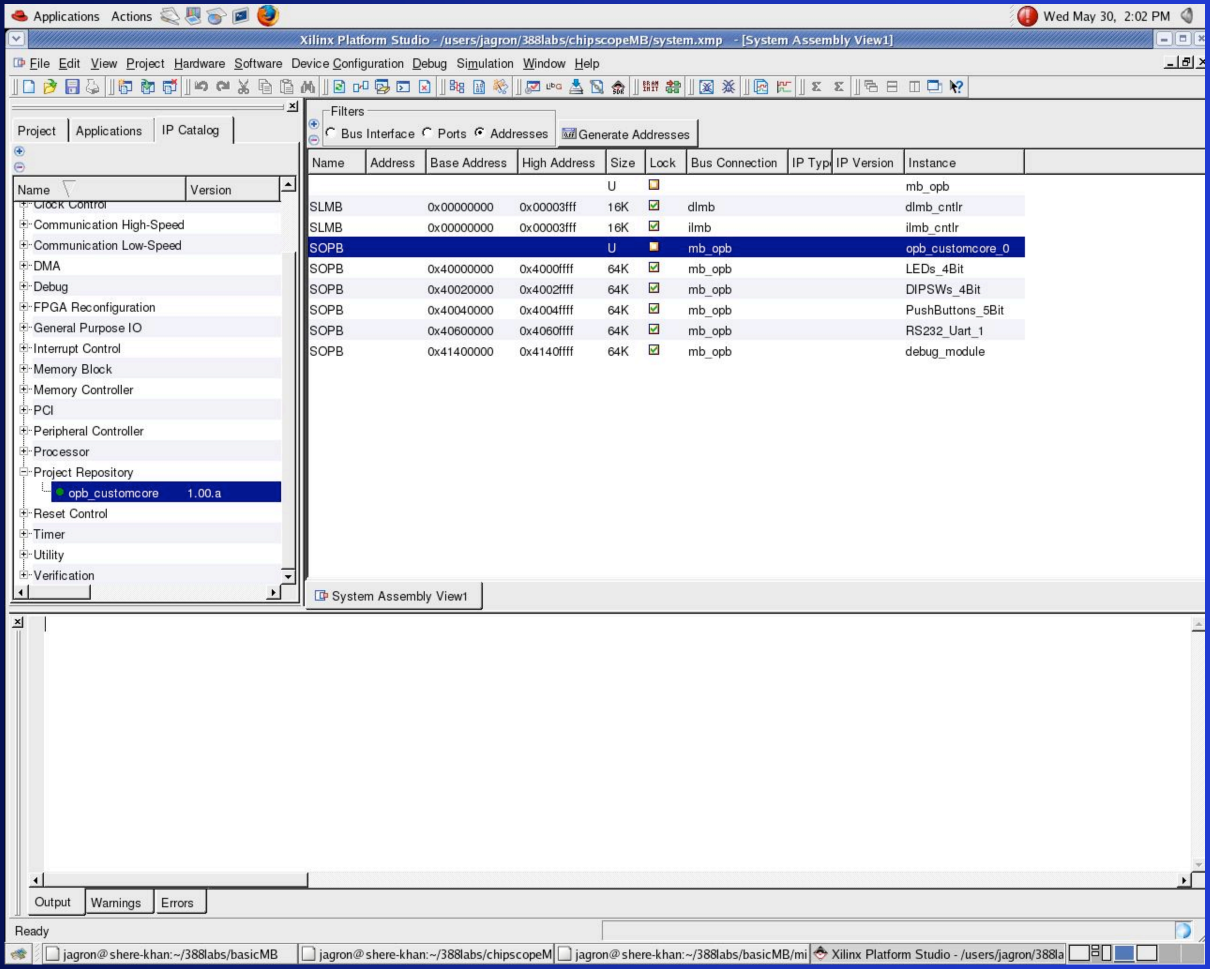

Applications Actions
Wed May 30, 2:01 PM
Xilinx Platform Studio - /users/jagron/388labs/chipscopeMB/system.xmp - [System Assembly View1]
File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help
Project Applications IP Catalog
Filters
Bus Interface Ports Addresses
Connection Filters
OPB LMB LMB
System Assembly View1
Output Warnings Errors
jagron@shere-khan:~/388labs/basicMB
jagron@shere-khan:~/388labs/chipscopeM
jagron@shere-khan:~/388labs/basicMB/mi
Xilinx Platform Studio - /users/jagron/388la

Name	Version
Clock Control	
Communication High-Speed	
Communication Low-Speed	
DMA	
Debug	
FPGA Reconfiguration	
General Purpose IO	
Interrupt Control	
Memory Block	
Memory Controller	
PCI	
Peripheral Controller	
Processor	
Project Repository	
opb_customcore	1.00.a
Reset Control	
Timer	
Utility	
Verification	

Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	5.00.c
mb_opb		opb_v20	1.10.c
ilmb		lmb_v10	1.00.a
dlmb		lmb_v10	1.00.a
debug_module		opb_mdm	2.00.a
dlmb_cntlr		lmb_bram_if_cntlr	2.00.a
ilmb_cntlr		lmb_bram_if_cntlr	2.00.a
RS232_Uart_1		opb_uartlite	1.00.b
LEDs_4Bit		opb_gpio	3.01.b
DIPSWs_4Bit		opb_gpio	3.01.b
PushButtons_5Bit		opb_gpio	3.01.b
chipscope_icon_0		chipscope_icon	1.00.a
chipscope_opb_iba_0		chipscope_opb_iba	1.00.a
opb_customcore_0		opb_customcore	1.00.a
SOPB	mb_opb		
lmb_bram		bram_block	1.00.a
dcm_0		dcm_module	1.00.a

```
Assigned Driver opb_customcore 1.00.a for instance opb_customcore_0
opb_customcore_0 has been added to the project
```

XPS - Locking Address Ranges



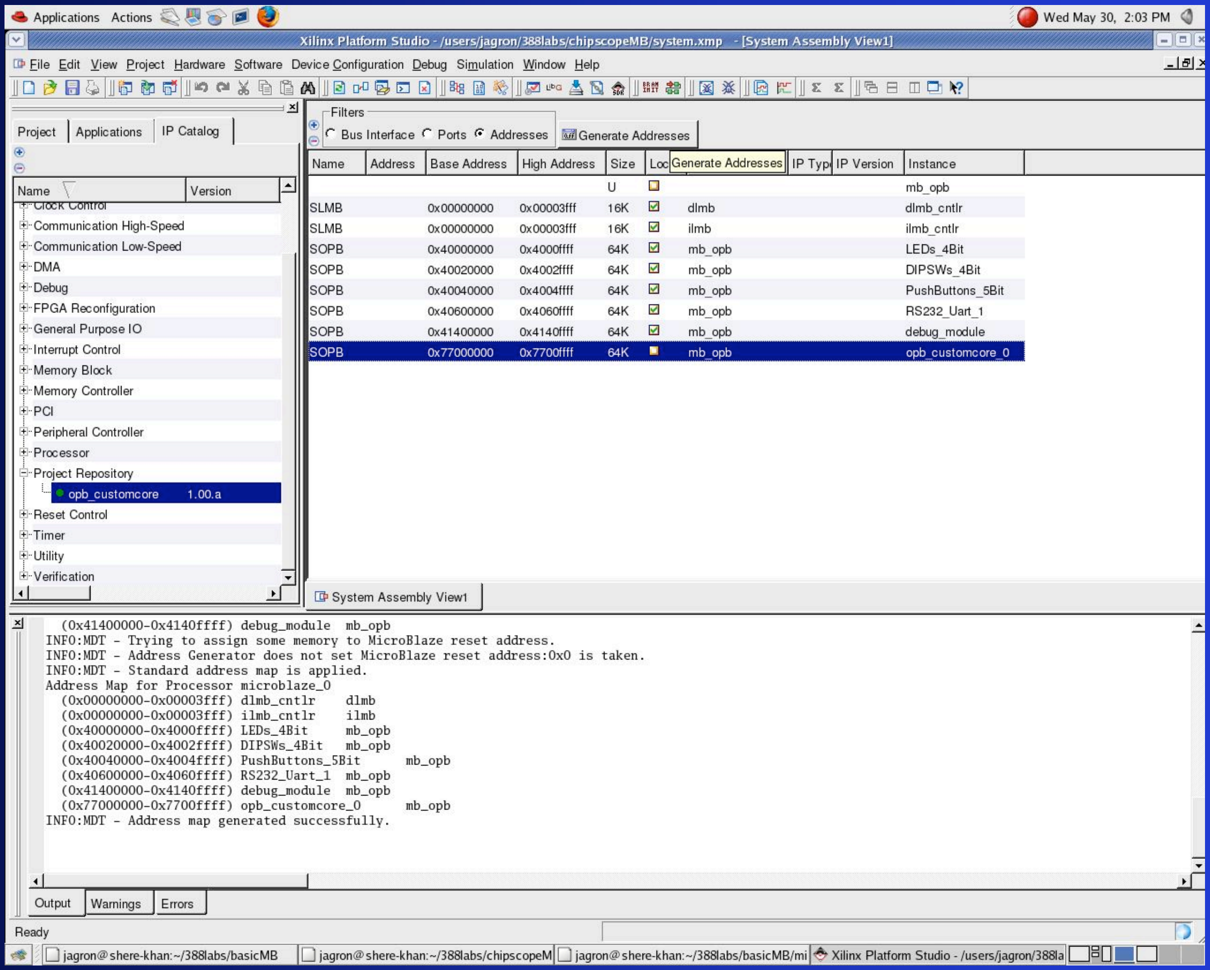
Xilinx Platform Studio - /users/jagron/388labs/chipscopeMB/system.xmp - [System Assembly View1]

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

Name	Address	Base Address	High Address	Size	Lock	Bus Connection	IP Type	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

IP Catalog: Project Repository → opb_customcore 1.00.a

XPS - Generate Address Range For New IP



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

Name	Address	Base Address	High Address	Size	Loc	Generate Addresses	IP Type	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

```
  (0x41400000-0x4140ffff) debug_module  mb_opb
INFO:MDT - Trying to assign some memory to MicroBlaze reset address.
INFO:MDT - Address Generator does not set MicroBlaze reset address:0x0 is taken.
INFO:MDT - Standard address map is applied.
Address Map for Processor microblaze_0
  (0x00000000-0x00003fff) dlmb_cntlr    dlmb
  (0x00000000-0x00003fff) ilmb_cntlr    ilmb
  (0x40000000-0x4000ffff) LEDs_4Bit     mb_opb
  (0x40020000-0x4002ffff) DIPSWs_4Bit   mb_opb
  (0x40040000-0x4004ffff) PushButtons_5Bit      mb_opb
  (0x40600000-0x4060ffff) RS232_Uart_1  mb_opb
  (0x41400000-0x4140ffff) debug_module  mb_opb
  (0x77000000-0x7700ffff) opb_customcore_0      mb_opb
INFO:MDT - Address map generated successfully.
```

Now What?

- The custom IP core is now…
  - Instantiated within the system
    - Via “Add IP”.
  - Connected to the system.
    - Via bus connection and address generation.
- Now, how do we use it?
  - We must write an application that “talks” to it.
- How do you communicate with IP?
  - You know it’s address (hopefully).
  - How does a CPU communicate with addresses???

Memory-Mapped I/O

- CPUs can read/write to addresses.
- Usually addresses refer to memory locations.
  - This is not always true.
  - Anything can be “mapped” into an address space.
    - It doesn’t have to be memory.
- Reads: request information from a specific source.
- Writes: send information to a specific source.
- What programming constructs do we need?

Pointers!!!

- Pointers:
  - A programming construct.
  - Used to “point” to a specific location.
    - Often times memory.
  - Features:
    - A location to point to (address).
    - Something being pointed at (data).

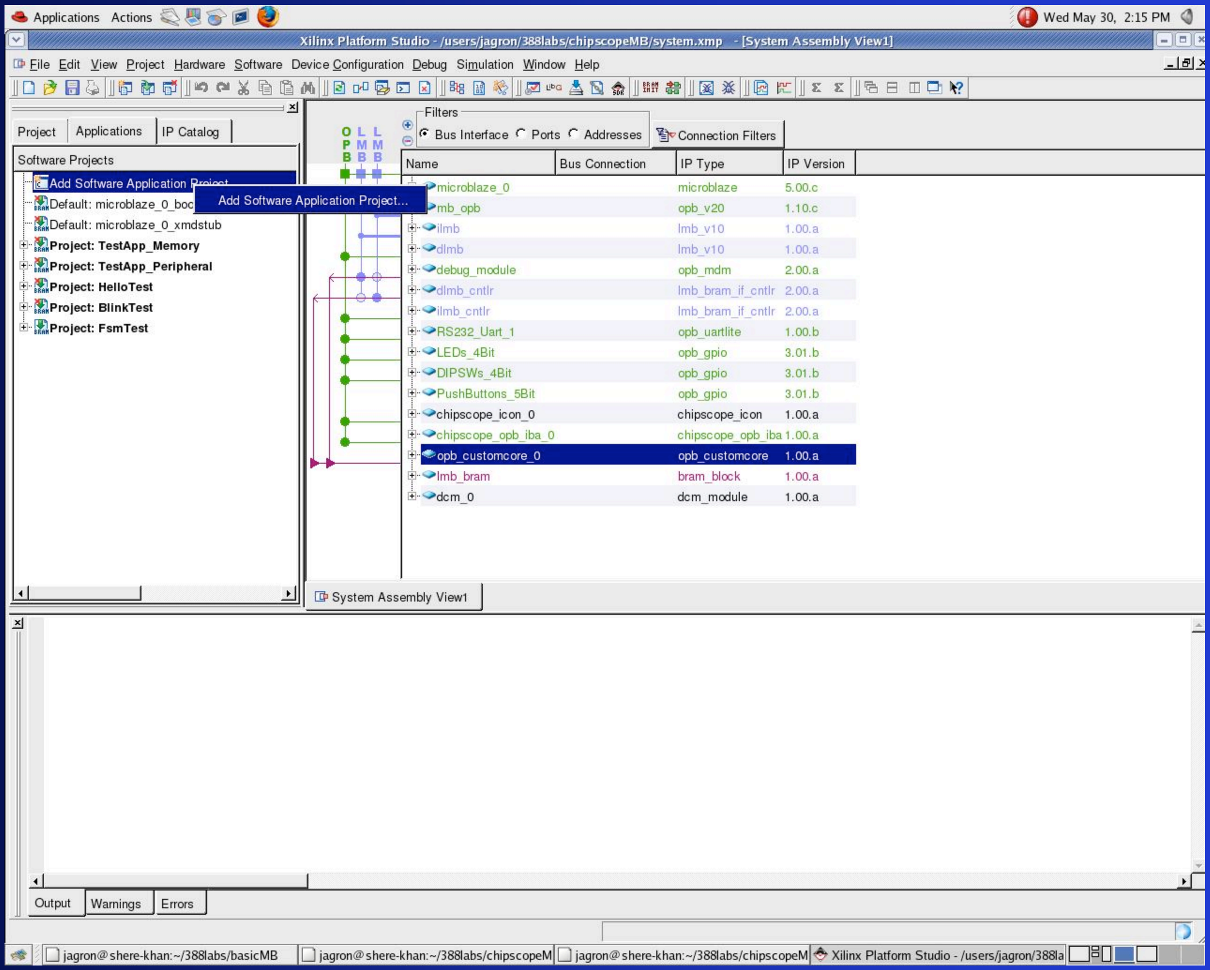
Pointer Example

- A pointer to an integer stored at location 0x5000000.
  - *int *myPtr = (int*)0x5000000;*
- Writing data to the location:
  - **myPtr = <newData>;*
- Reading data from the location:
  - *dataAtLocation = *myPtr;*
- Changing the location being pointed at:
  - *myPtr = <newLocation>;*
- What is the “*” doing????

XPS - Creating An Application

- We now know how to "talk" to IP.
- Now, let's make an application to prove it.
- This can be done in XPS via the…
  - Applications Tab.
- Steps:
  - Create a new application.
  - Associate it with a CPU.
  - Create and define the source program.
  - Run the program.

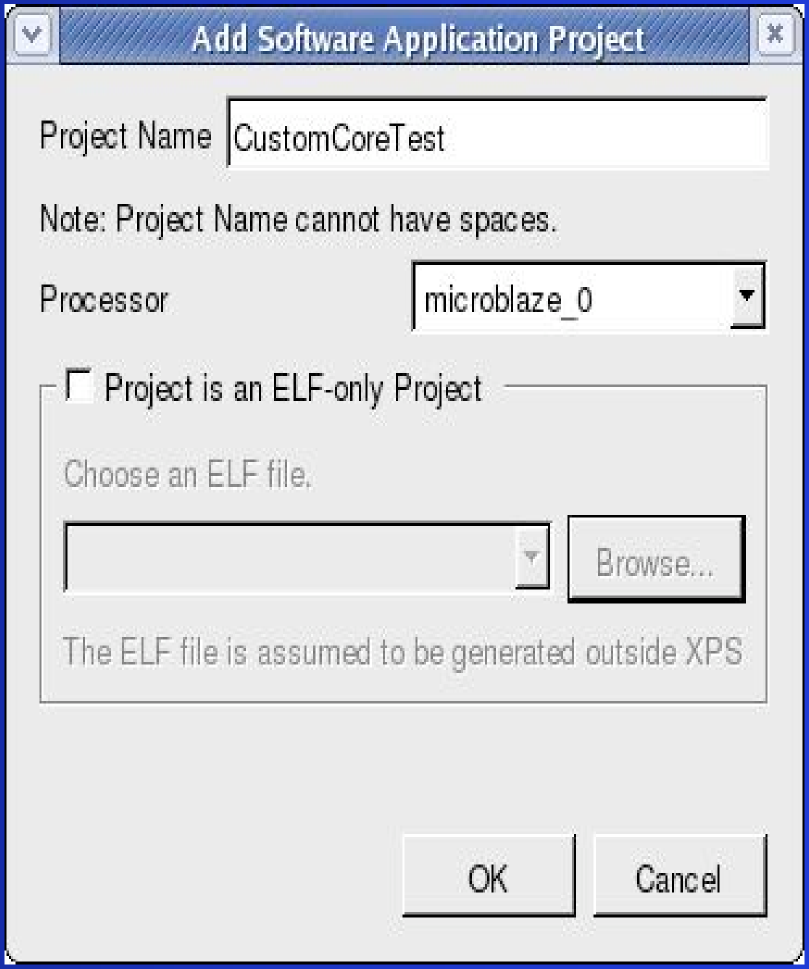
XPS - Creating An Application


Applications
Actions
Wed May 30, 2:15 PM
Xilinx Platform Studio - /users/jagron/388labs/chipscopeMB/system.xmp - [System Assembly View1]
File
Edit
View
Project
Hardware
Software
Device Configuration
Debug
Simulation
Window
Help
Project
Applications
IP Catalog
Software Projects
Add Software Application Project
Add Software Application Project...
Default: microblaze_0_boc
Default: microblaze_0_xmdstub
Project: TestApp_Memory
Project: TestApp_Peripheral
Project: HelloTest
Project: BlinkTest
Project: FsmTest
OPB
LMB
LMB
Filters
Bus Interface
Ports
Addresses
Connection Filters
System Assembly View1
Output
Warnings
Errors
jagron@shere-khan:~/388labs/basicMB
jagron@shere-khan:~/388labs/chipscopeM
jagron@shere-khan:~/388labs/chipscopeM
Xilinx Platform Studio - /users/jagron/388la

Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	5.00.c
mb_opb		opb_v20	1.10.c
ilmb		lmb_v10	1.00.a
dlmb		lmb_v10	1.00.a
debug_module		opb_mdm	2.00.a
dlmb_cntlr		lmb_bram_if_cntlr	2.00.a
ilmb_cntlr		lmb_bram_if_cntlr	2.00.a
RS232_Uart_1		opb_uartlite	1.00.b
LEDs_4Bit		opb_gpio	3.01.b
DIPSWs_4Bit		opb_gpio	3.01.b
PushButtons_5Bit		opb_gpio	3.01.b
chipscope_icon_0		chipscope_icon	1.00.a
chipscope_opb_iba_0		chipscope_opb_iba	1.00.a
opb_customcore_0		opb_customcore	1.00.a
lmb_bram		bram_block	1.00.a
dcm_0		dcm_module	1.00.a

XPS - Application Properties

- We need to give this application project a name.
- We also need to associate it with a processor.
  - Chooses which compiler to use.
- *Why does the compiler matter?*


Add Software Application Project

Project Name: CustomCoreTest

Note: Project Name cannot have spaces.

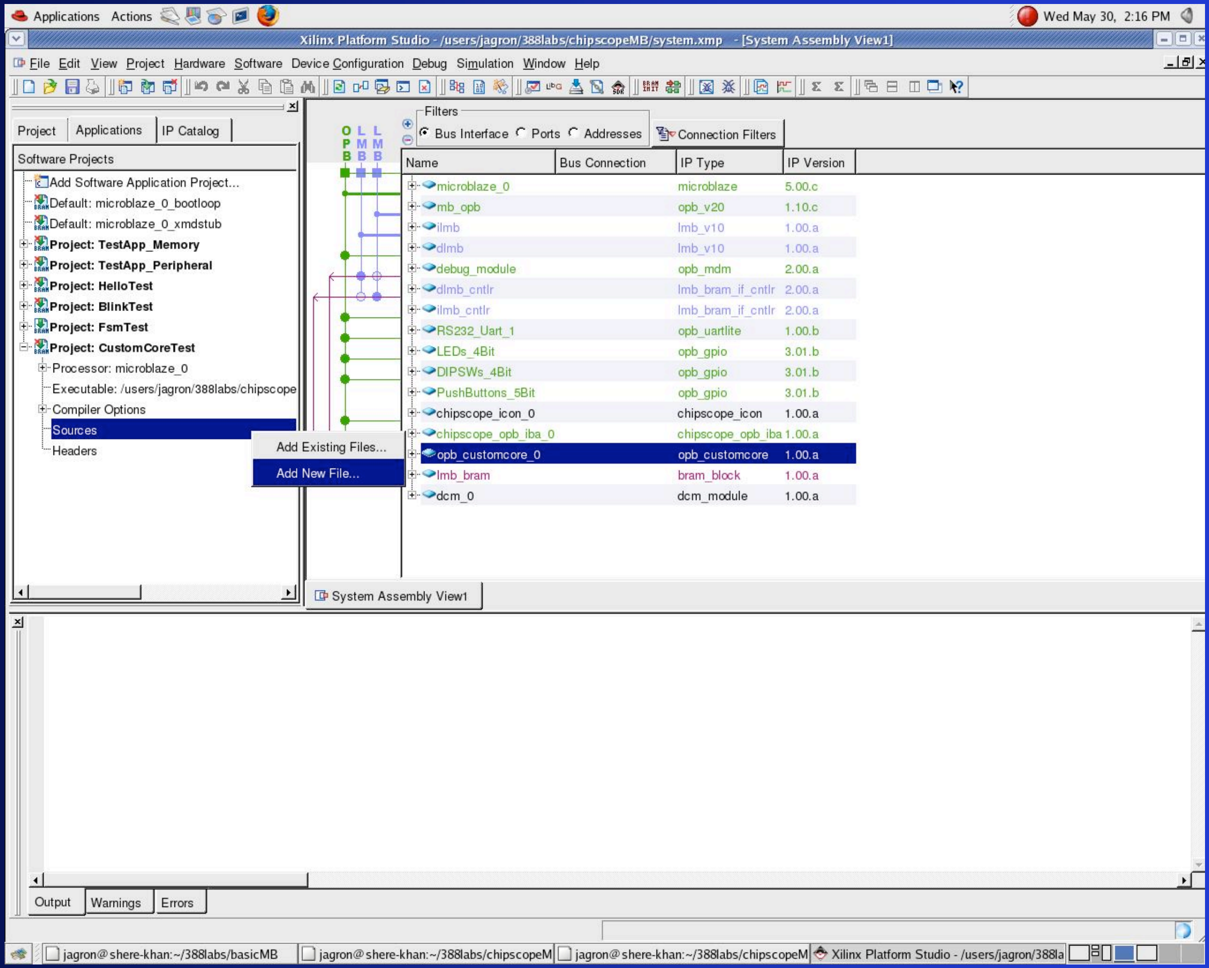
Processor: microblaze_0

☐ Project is an ELF-only Project

Choose an ELF file.

The ELF file is assumed to be generated outside XPS

XPS - New Application Sources



Applications Actions
Wed May 30, 2:16 PM
Xilinx Platform Studio - /users/jagron/388labs/chipscopeMB/system.xmp - [System Assembly View1]
File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help
Project
Applications
IP Catalog
OPB
LMB
LMB
Add Existing Files...
Add New File...
System Assembly View1
Output
Warnings
Errors
jagron@shere-khan:~/388labs/basicMB
jagron@shere-khan:~/388labs/chipscopeM
jagron@shere-khan:~/388labs/chipscopeM
Xilinx Platform Studio - /users/jagron/388la

Software Projects
- Add Software Application Project...
- Default: microblaze_0_bootloop
- Default: microblaze_0_xmdstub
- **Project: TestApp_Memory**
- **Project: TestApp_Peripheral**
- **Project: HelloTest**
- **Project: BlinkTest**
- **Project: FsmTest**
- **Project: CustomCoreTest**
  - Processor: microblaze_0
  - Executable: /users/jagron/388labs/chipscope
  - Compiler Options
  - Sources
  - Headers

Filters: (●) Bus Interface ( ) Ports ( ) Addresses — Connection Filters

Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	5.00.c
mb_opb		opb_v20	1.10.c
ilmb		lmb_v10	1.00.a
dlmb		lmb_v10	1.00.a
debug_module		opb_mdm	2.00.a
dlmb_cntlr		lmb_bram_if_cntlr	2.00.a
ilmb_cntlr		lmb_bram_if_cntlr	2.00.a
RS232_Uart_1		opb_uartlite	1.00.b
LEDs_4Bit		opb_gpio	3.01.b
DIPSWs_4Bit		opb_gpio	3.01.b
PushButtons_5Bit		opb_gpio	3.01.b
chipscope_icon_0		chipscope_icon	1.00.a
chipscope_opb_iba_0		chipscope_opb_iba	1.00.a
opb_customcore_0		opb_customcore	1.00.a
lmb_bram		bram_block	1.00.a
dcm_0		dcm_module	1.00.a

Application Hints

- First, figure out the base address of the custom IP core.
- Important register offsets:
  - Reg0 = base + 0x0.
  - Reg1 = base + 0x4.
  - Reg2 = base + 0x8.
  - Reg3 = base + 0xC.
  - ResetReg = base + 0x100.
    - Reset command to write is 0x0000000A.

Application Hints

- First define pointers to all required registers.
- Make sure to use the *volatile* qualifier!!!
  - What does this do?
  - Why is this needed?
- Make a simple program…
  - Reset state of custom IP core.
  - Write to fill in custom IP core's state.
  - Read back state to verify correct operation.
  - Reset state of custom IP core.
  - Read back state to verify correct operation.